Fundamentos em Ciência da Computação

Aleardo Manacero Jr. Norian Marranghello

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

O que é Ciência da Computação

O início dessa disciplina deve fazer por uma demarcação de territórios, definindo os vários contextos de trabalho com o computador, indicando o tipo de aprendizagem necessária em cada contexto e quais são os cursos que oferecem esse tipo de conhecimento.

Dentro desse cenário será trabalhado com mais ênfase o curso de bacharelado em ciência da computação, por razões bastante óbvias é claro. Nesse capítulo introduziremos os principais conceitos a respeito desse curso, assim como de outros cursos de nível superior na área de informática.

Um desses conceitos é o fato de cursos de ciência da computação demandarem uma elevada carga de conhecimento em matemática. Durante os demais capítulos desse texto estaremos apresentando as razões que geram tal necessidade através de estudos de problemas de computação que são resolvidos com o auxílio de determinados tópicos de matemática. Entretanto, para chegar nessa conclusão é preciso começar com um pouco de história sobre os cursos superiores na área de informática no Brasil.

1.1 Histórico

Os primeiros cursos na área no Brasil surgiram quase que simultaneamente em quatro instituições diferentes, Unicamp, USP (IME), UFBA e PUC-RJ, isso no final da década de 60. Na época não havia nenhuma norma sobre como seriam tais cursos e sobre quais as necessidades de formação em cada um deles. As pessoas que criaram tais cursos eram, em sua maioria, matemáticos e engenheiros, sendo que alguns tiveram contato com a computação durante seus doutorados no exterior. Isso fez com que os cursos criados tivessem o perfil de seus criadores, sendo muito distintos em cada um dos casos.

Essa situação persistiu ao longo dos vinte anos seguintes, com cada curso sendo criado segundo o perfil do grupo que o gerou. Isso, por si só, não é problema algum. O problema é que os nomes dados aos cursos acabavam por ser, muitas vezes, conflitantes, com cursos parecidos e nomes distintos e cursos distintos de mesmo nome. Isso, entretanto, foi considerado como uma situação normal para cursos atuando numa área de conhecimento ainda muito jovem (lembrem-se que os primeiros computadores surgiram durante a segunda grande guerra mundial, apenas sessenta anos atrás).

Durante a década de 70 a comunidade atuando em computação no país aumentou, permitindo até a criação de uma sociedade acadêmica para congregar os pesquisadores da área, que é a SBC - Sociedade Brasileira de Computação. O que a computação é hoje no Brasil se deve muito ao que a SBC realizou nos últimos 25 anos, sendo portanto importantíssimo

que todos nós nos associemos à mesma, participando ativamente de suas atividades.

Para um profissional de computação o trabalho da SBC é mais visível em dois aspectos: regulamentação da profissão e padrões para cursos da área. O primeiro não cabe nesse texto, bastando dizer que nossa profissão não é regulamentada (portanto não se pode exigir diplomas para exercê-la) e que a posição da SBC é pela manutenção dessa liberdade, porém com uma regulamentação que evite ataques de conselhos como CREA e CFA. Isso pode ser visto através de uma proposta de projeto de lei a ser enviada ao Congresso Nacional, obtida em **http://www.dcc.ufmg.br/ bigonha/Sbc/plsbc.html** .

Já o segundo aspecto diz diretamente respeito ao assunto aqui tratado. O estabelecimento de padrões para cursos da área de computação é, na realidade, um fenômeno mundial, com muitas entidades envolvidas com o problema, como é o caso da ACM (*Association for Computer Machinery*) e IEEE (*Institute of Electrical and Electronic Engineers*). Em particular, essas duas divulgaram final de 2001 um currículo conjunto para Ciência da Computação ([1]) e estão trabalhando em outros currículos para Engenharia de Computação, Tecnologia da Informação e para Engenharia de Software.

No Brasil é a SBC quem tem trabalhado no estabelecimento de tais currículos, sendo que em nosso caso definiu-se que a melhor estratégia seria a geração de currículos de referência, que poderiam ser usados como base para a criação de currículos individuais em cada instituição de ensino. Esse trabalho começou no final da década de 80, com a criação, dentro do Congresso Anual da SBC, de um espaço para que coordenadores de curso e demais interessados discutissem propostas de como estruturar cursos e matérias de computação (o Workshop de Ensino de Informática, WEI). O segundo passo foi a criação do primeiro currículo de referência, em 1991, atendendo apenas os cursos de ciência da computação.

Durante os últimos dez anos a SBC refez o currículo de ciência da computação por duas vezes, incluíndo Engenharia de Computação e também currículos de referência para cursos de Sistemas de Informação (que fora do Brasil é chamado de Tecnologia da Informação) e de Licenciatura em Computação [2].

Nas próximas páginas examinaremos as diferenças entre esses cursos e também algumas outras modalidades de ensino surgidas nos últimos anos.

1.2 Cursos superiores em informática

No Brasil, após a promulgação da nova Lei de Diretrizes e Bases do Ensino, em 1996, passou a existir várias formas de ensino no nível superior, que são: graduação plena, ensino seqüencial e ensino tecnológico. Os dois últimos são cursos mais rápidos, voltados para a formação imediata de profissionais para ocupar nichos específicos do mercado de trabalho. Ocupam de certo modo o mesmo lugar dos antigos cursos de Tecnólogos e os de licenciatura curta. A formação acelerada nesses cursos é feita com o prejuízo de uma formação mais sólida e abrangente, obtida apenas com os cursos de graduação plena. Esse prejuízo de formação impede, também, que egressos desses cursos tenham acesso a programas de mestrado e doutorado.

Da experiência adquirida pela oferta de cursos seqüenciais na área de informática (por faculdades e universidades particulares) é possível constatar que a demanda por tais cursos é formada predominantemente por pessoas que já trabalham na área mas não possuem educação formal. É raro que estudantes que acabaram o ensino médio procurem por tais cursos. Como são cursos geralmente focados em nichos de mercado (e como a legislação

proíbe que se use o nome dos cursos regulares de graduação), cursos seqüenciais, e também de tecnólogos, apresentam nomes como "Gestão em Tecnologia de Informação", "Gestão em Bancos de Dados", "Administração de Redes de Computadores", etc.

Como dito, tais cursos permitem uma formação mais rápida, em dois ou três anos. Isso permite que se chegue antes ao mercado de trabalho. Eles também fornecem uma formação mais especializada, tratando com mais detalhes a tecnologia que seja seu centro de ação. Se por um lado isso é vantajoso ao fornecer ao aluno aquilo que as empresas querem naquele momento, também é extremamente perigoso pois o que as empresas querem pode mudar velozmente e, nesse caso, a formação se deu apenas em algo que não serve para mais nada.

Os cursos de graduação plena procuram formar um outro tipo de profissional. Formam aquele indivíduo com um conhecimento mais abrangente, menos especializado (ou menos voltado para produtos), mas que tenha uma grande capacidade de se adaptar a novas tecnologias. Essa formação os coloca mais tarde no mercado, mas garante, entretanto, condições de concorrência para um leque maior de nichos tecnológicos.

No Brasil a LDB define que os currículos dos cursos de graduação devem ser montados a partir de um conjunto de normas chamadas de "Diretrizes Curriculares". Para a área de computação e informática as diretrizes curriculares indicam a existência de quatro diferentes cursos de graduação, que foram definidos a partir de discussões dentro da SBC (dentro dos WEI para ser mais preciso). As quatro denominações são "Ciência da Computação" (BCC), "Engenharia de Computação" (EC), "Sistemas de Informação" (BSI) e "Licenciatura em Computação" (LC).

A diferenciação entre esses cursos se dá, num primeiro momento, pelo papel do computador dentro da atuação profissional. Nessa ótica o computador pode ser visto como uma atividade meio, isso é, como uma ferramenta para o trabalho do profissional, ou como uma atividade fim, em que ele é o objetivo do trabalho. No primeiro caso, atividade meio, estão os cursos de BSI e de LC. Como atividade fim estão os cursos de BCC e EC.

E' principalmente essa diferenciação entre o computador ser meio ou fim que dá as características básicas dos vários cursos da área. A descrição de cada um deles, feita a seguir, procura estabelecer exatamente como essa influência se manifesta.

1.2.1 Licenciatura em Computação

Trata-se de um curso bastante novo, com ainda poucas ofertas no país. O campo de atuação de um profissional formado em licenciatura em computação é o ensino de computação em escolas de ensino médio e fundamental, além de servir como apoio aos professores de outras disciplinas no uso do computador como uma ferramenta de auxílio ao ensino das mesmas. Numa linha adicional de trabalho, esse profissional pode se dedicar ao desenvolvimento de aplicativos educacionais, tanto do lado de ensino quanto de gestão de ensino.

Do ponto de vista de mercado, esse ainda é um campo em desenvolvimento, que tanto pode se tornar excelente se as políticas governamentais caminharem na direção de oferecer uma formação digital melhor aos nossos alunos, quanto pode se tornar inexistente no caso contrário. Como existe uma inércia natural no sistema educacional, acredita-se que os profissionais em formação encontrarão um excelente campo de atuação, que deve ser ainda mais ampliado nos próximos anos.

Quanto ao tipo de formação necessária para esse profissional, é evidente que o mesmo deverá cursar uma série de disciplinas pedagógicas que lhe permitirão conhecer e aplicar as diferentes metodologias de ensino, assim como entender as normas de funcionamento das

diferentes escolas.

Do lado computacional, como o computador será uma ferramenta para seu trabalho, é preciso ter conhecimento de uma série de linguagens e técnicas de programação. Não é preciso um conhecimento profundo das tecnologias envolvidas no funcionamento do computador, nem das bases conceituais e teóricas da computação.

Da mesma forma, se o computador é ferramenta, passa a ser necessária uma formação mais ampla nas várias aplicações que o mesmo pode ter na escola, exigindo então uma maior carga de trabalho em atividades complementares.

1.2.2 Sistemas de Informação

Os chamados cursos de bacharelado em sistemas de informação são, na prática, uma transformação dos antigos cursos de análise de sistemas, agora adequados aos novos modelos de tecnologia da informação. Assim como aqueles, esses também têm o computador como um meio, isso é, o objetivo aqui é fornecer ferramentas computacionais como aplicações em outros campos do saber.

Vistos sob esse ponto de vista, são esses os profissionais com o maior campo de atuação dentro da informática. Estima-se que cerca de 80% das vagas em informática no Brasil se destinem a profissionais formados em cursos de BSI. Esse mercado não está refletido, ainda, no número de cursos oferecidos, uma vez que apenas nos últimos cinco anos é que faculdades e universidades passaram a oferecer cursos de BSI. Essa ausência faz com que os postos em aberto sejam ocupados por profissionais vindos de cursos de BCC e EC, além de outros com as mais diversas formações.

O tipo de trabalho a ser desenvolvido por esses profissionais vai desde o desenvolvimento de aplicativos para empresas, tais como bancos de dados, sistemas de controle (estoque, produção, etc.) e sistemas de gestão, até o gerenciamento de sistemas computacionais, o que envolve a especificação, supervisão e manutenção de redes de computadores e demais equipamentos.

Esses campos de atuação indicam que a formação necessária é ligada ao conhecimento das várias tecnologias (do ponto de vista de sua aplicação) e de várias técnicas e linguagens de programação (para o desenvolvimento dos aplicativos). A necessidade de interação com profissionais de outras áreas (os usuários dos aplicativos) faz com que seja preciso também uma formação complementar bastante ampla, o que inclui matérias como gestão de organizações, economia e contabilidade, administração de empresas, etc.

Assim como para os cursos de licenciatura, não é necessário um conhecimento profundo de teorias e conceitos fundamentais da computação, assim como das várias tecnologias que serão aplicadas, uma vez que seu papel é o de usar tais tecnologias e não desenvolvê-las.

1.2.3 Engenharia de Computação

Esses cursos possuem como característica essencial o fato de ainda não terem uma identidade bem definida, isso tanto no Brasil quanto no mundo. Aqui a SBC tem feito um esforço nos últimos anos para o estabelecimento de uma diferenciação precisa entre os cursos de EC e de BCC (e engenharia elétrica, EE, de quebra). Essa falta de identidade é natural e veio da origem desses cursos, em meados da década de 80, quando os mesmos surgiram procurando ocupar um vazio entre os cursos de BCC e de EE.

Como conseqüência dessa falta de identidade surge também o problema da falta de um

nicho específico no mercado de trabalho. A inexistência do nicho não impede, entretanto, que os profissionais de EC tenham uma boa aceitação no mercado.

Indo então para a descrição do que seria a engenharia de computação, pode-se dizer que ela representa o segmento de computação preocupado com o desenvolvimento de sistemas dedicados aos processos industriais e ao desenvolvimento dos próprios computadores. Nesse sentido seu campo de atuação envolve o projeto de computadores, controladores e sistemas embarcados (ou embutidos). O trabalho em qualquer dessas áreas é parcial ou completamente equivalente ao trabalho de engenheiros eletricistas ou de cientistas da computação. A vantagem do engenheiro de computação é que o mesmo domina todas as vertentes do trabalho e não apenas uma de suas facetas (hardware ou software), como acontece com os outros dois.

Sendo EC um curso em que o computador é o fim, fica evidente que passa a ser necessário um conhecimento mais amplo em alguma das tecnologias envolvidas, sendo que nesse caso falamos de arquitetura de computadores. O domínio desse tipo de tecnologia (e de suas aplicações em processos industriais) demanda um forte conhecimento de física (eletricidade) e de matemática. Além dessas matérias passa a ser necessário também o domínio dos fundamentos teóricos de computação.

Diferentemente dos cursos em que computação é uma atividade meio, para os cursos de EC não é preciso o conhecimento de muitas técnicas e linguagens de programação, uma vez que o objetivo do trabalho do engenheiro não é desenvolver aplicativos para outros usuários.

Os cursos de EC existentes no país foram criados, em sua maioria, dentro de um conceito em que a instituição proponente definiria se o perfil do seu curso seria mais próximo da engenharia elétrica ou ciência da computação. Essa liberdade regulou por bastante tempo o processo de autorização e reconhecimento de tais cursos, com o MEC indicando comissões de avaliação orientadas pelo perfil do curso (CEEEng ou CEEInf), o que permitiu uma melhor avaliação dos cursos mas manteve a indefinição quanto ao perfil mais preciso do que seria engenharia de computação.

Hoje temos cursos que claramente optaram por um dos modelos (elétrica ou computação), como é o caso da UFSCar com o curso moldado pela ciência da computação. Temos também cursos que evitaram fazer qualquer opção, mantendo na prática as duas opções, como é o caso da Unicamp, em que os alunos optam por uma das áreas após o quinto semestre do curso. Em todos os casos, uma característica comum é a presença de algumas matérias apenas por exigência do CREA (resistência dos materiais, química, fenômenos de transporte, por exemplo) e um forte embasamento matemático (pelo menos nos bons cursos da área).

1.2.4 Ciência da Computação

Para terminar a descrição dos cursos de graduação falta falar do curso em que vocês estão matriculados, que é o de bacharelado em ciência da computação. Esse curso, assim como EC, tem o computador como objetivo fim, isto é, o profissional formado aqui deveria trabalhar pelo desenvolvimento da computação propriamente dita, sem a preocupação em criar aplicativos para os usuários finais.

Essa característica limitaria bastante o mercado de trabalho para esses profissionais, que ficaria restrito a pesquisa e desenvolvimento de tecnologias básicas (sistemas operacionais, compiladores, gerenciadores de bancos de dados, etc.). Como esse mercado não representa 20% das necessidades empresariais (no Brasil ainda menos), qual seria a razão do expressivo número de cursos de BCC? - A explicação tem razões históricas e mercadológicas, uma vez

que o nome "Ciência da Computação" tinha um poder de atrair alunos muito mais forte do que "Análise de Sistemas". Isso fez com que muitos cursos fossem criados antes de 1996 com a denominação de BCC porém com perfis totalmente distintos. Do ponto de vista de mercado, não havia problema de empregar os egressos de cursos de BCC (por mais voltados para pesquisa que eles fossem) pois a base dada nos mesmos era suficiente para que os profissionais rapidamente se adaptassem a situação momentânea do mercado.

Como a essência de um curso de BCC está no desenvolvimento de pesquisas e novas tecnologias para a computação, uma característica essencial é de que o mesmo permita aos alunos o envolvimento com trabalhos de pesquisa. Essa é, aliás, a principal razão para que o MEC indique que cursos de BCC (e de EC) devam ser em período integral. Por sua vez, a interação com trabalhos de pesquisa apenas ocorre se os docentes do curso estiverem envolvidos com pesquisa, o que não é frequente e é o que diferencia os bons cursos dos demais.

Assim como ocorre com EC, os cursos de BCC devem ter uma grande base teórica, o que resulta em um bom número de disciplinas de matemática e de fundamentos teóricos de computação. Do ponto de vista das tecnologias, cursos de BCC devem ser estruturados de forma a permitir o contato dos alunos com as mais variadas tecnologias e, além disso, o domínio de uma ou mais delas.

Mais uma vez, por ter o computador como um fim, não é preciso o conhecimento de um número maior de linguagens de programação, uma vez que a função dos futuros profissionais não é o de criar aplicativos dos mais variados. Esse aspecto, aliás, é o que mais frustra aqueles que fazem a opção por um curso de BCC mas que esperavam na realidade um de BSI. O contato com o computador, embora exista, não é tão intenso, com a existência de várias disciplinas que abordam temas de computação do ponto de vista estritamente teórico.

O que se pode concluir dessa discussão é que cursos de BCC são bastante árduos, com pequeno uso de computadores e muito desenvolvimento teórico. Isso é verdade e faz com que aqueles que tenham mais interesse em aplicativos sofram bastante durante o curso. Apesar disso, por tratar sempre de tópicos de fronteira no desenvolvimento da computação, o BCC é também bastante atrativo e instigante para aqueles que desejam trabalhar com novos desafios a cada instante.

A descrição básica do que seria um curso de BCC pode ser encerrada aqui. Maiores detalhes sobre esses cursos aparecem na próxima seção, que discute como o curso de BCC oferecido pela Unesp em São José do Rio Preto está estruturado.

1.3 Como é o nosso BCC

Antes de falarmos de sua situação atual é importante termos um pouco de história. Este BCC está em funcionamento desde 1987, quando a primeira turma ingressou através de um vestibular adicional, em abril daquele ano.

O projeto do curso contou com a ajuda de vários professores que estavam, na época, envolvidos com os cursos de bacharelado e licenciatura em matemática. Os cursos de matemática oferecidos pelo Ibilce já eram bastante tradicionais, sendo dos melhores do país. Para se ter uma idéia, vários de seus ex-alunos são hoje professores das melhores universidades brasileiras, inclusive com vários deles como pesquisadores em computação. Esses professores, em colaboração com o Prof. Odelar Leite Linhares, que antes já havia colaborado na estruturação dos cursos de computação da Unicamp e da USP em São Carlos, montaram um curso típico de computação, com uma forte base matemática.

Ao longo dos anos vários professores mais voltados para a computação foram contratados e formados no departamento. Esses professores trabalharam por vários anos até a proposição do currículo atualmente em vigor, o qual apesar de possuir uma componente tecnológica mais intensa não dispensou a formação básica sólida e abrangente, que era a razão do sucesso do curso até então. Aliás, vários dos alunos das primeiras turmas são hoje professores doutores em boas universidades, devendo isso em grande parte à base oferecida pelo curso.

Embora o curso em sua forma original fosse bastante bom, faltava-lhe uma maior densidade de tópicos em novas tecnologias. Para se ter uma idéia, não havia nenhuma disciplina obrigatória em redes de computadores. Isso, na época de sua criação era absolutamente normal, uma vez que as redes apenas se popularizaram no Brasil no início dos anos 90. A busca por um novo currículo era fundamentada principalmente por lacunas desse tipo. Com isso o trabalho de elaboração de um novo currículo começou em 1991, sendo concluído no início de 1996 e implantado a partir de 1997.

Os cinco anos transcorridos foram consumidos em propostas (duas) que foram abandonadas em favor da proposta do atual currículo. O problema da primeira proposta era a sua extensão, que exigia cinco anos de curso com elevada carga de trabalho. A segunda proposta era basicamente uma redução da primeira, sendo bastante convencional e foi abandonada com a idéia de flexibilização do currículo introduzida em 1995. Essa história (e toda a descrição do currículo) pode ser vista em [3], apresentado no 31st Frontiers In Education, en Reno, EUA.

O currículo atual do curso (que deve sofrer uma nova reestruturação em breve, para atendimento da LDB e de algumas diretrizes da própria Unesp para padronização de cursos similares) está fundamentado em duas premissas: ninguém aprende sem praticar de algum modo sobre o assunto e ninguém domina (nem tem interesse por) todas as áreas de conhecimento dentro da computação. Quando associadas tais premissas indicam que não se deve prender muito o aluno em sala de aula, permitindo que esse pratique sobre os conteúdos, e que não é preciso ensinar tudo com a mesma profundidade.

Assim, o curso foi estruturado em ênfases, com cada aluno cursando, a partir do terceiro ano, apenas as disciplinas de sua ênfase e algumas outras que sejam de seu interesse. Os dois primeiros anos foram estruturados de forma a garantir a base teórica indispensável (muita matemática e fundamentos de computação) e também os conhecimentos básicos sobre as várias tecnologias, de modo a permitir que os alunos escolham a ênfase de sua preferência.

Hoje o que se pode dizer sobre esse currículo é que o mesmo antecipou as diretrizes curriculares da área, que afirmam exatamente que é preciso um conhecimento básico sobre todas as tecnologias e aprofundado sobre uma delas. Isso ainda não é visto nos demais cursos de BCC, que possuem uma carga horária elevada e tentam dar um conhecimento aprofundado em todas as tecnologias para todos os alunos, sem levar em consideração preferências pessoais.

Para uma melhor descrição desse (e de qualquer outro) currículo é preciso fazer um mapeamento entre as disciplinas do curso e as matérias indicadas pelas diretrizes curriculares. Esse mapeamento é feito através da divisão das matérias das diretrizes em três grupos principais, que são ciência da computação, matemática e formação tecnológica. Outras matérias envolvem física e eletricidade, ciência dos sistemas de informação, formação complementar e suplementar, que não serão examinadas aqui. As demais são descritas na seqüência.

1.3.1 Ciência da Computação

Trata-se de uma matéria da área de formação básica, envolvendo basicamente programação (linguagens e técnicas de programação), computação e algoritmos (teoria de computação) e arquitetura de computadores (hardware).

Em nosso curso essa matéria é coberta através de disciplinas como Programação I, Programação II, Estrutura de Dados, Circuitos Lógicos, Fundamentos em Sistemas de Automação e Controle Digital, Fundamentos em Sistemas de Informação e Fundamentos em Linguagens e Teoria da Computação. Outras disciplinas que também cobrem em parte essa matéria, porém dependem da ênfase escolhida, são Organização e Recuperação de Informações e Programação Concorrente.

1.3.2 Matemática

Uma das motivações para a futura reestruturação curricular é exatamente a matéria de matemática, que atualmente possui um foco maior na chamada *matemática do contínuo* (por tratar com funções contínuas), que deverá ser alterado para a matemática discreta (funções discretas), seguindo a nova orientação da área.

Hoje essa matéria é coberta por várias disciplinas, a maioria ainda voltadas para o contínuo. Temos Cálculo Diferencial e Integral I, II e III, Geometria Analítica e Vetores, Álgebra Linear e Aspectos Formais da Computação, todas ministradas por docentes do departamento de Matemática. Temos ainda Cálculo Numérico, Probabilidade e Estatística, Métodos de Otimização I e Teoria dos Grafos, que são ministradas por docentes do departamento de ciências de computação e estatística, DCCE, além de uma ênfase em computação científica que tem uma forte conexão com matemática.

Dentre as alterações previstas para o futuro currículo está um desdobramento da disciplina Aspectos Formais da Computação em duas disciplinas, uma sobre Lógica Matemática e outra sobre Álgebra, e também uma redução de carga horária das disciplinas de Cálculo (sem alteração profunda de conteúdo, entretanto).

1.3.3 Formação tecnológica

Esta é, na realidade, uma área de concentração de diversas matérias, tais como compiladores, banco de dados, sistemas operacionais, redes de computadores, sistemas distribuídos, engenharia de software, etc. Nesse grupo estão todas as matérias com vínculo tecnológico, sendo portanto o coração do futuro exercício profissional.

Dentro do previsto pelas diretrizes curriculares os alunos de BCC devem ser expostos a todas as tecnologias e aprofundar conhecimentos em pelo menos uma delas. Como já indicado, em nosso currículo o aprofundamento é feito pelas disciplinas das várias ênfases (ver na página do curso na Internet, em http://www.dcce.ibilce.unesp.br/comp). Já a visão geral das várias matérias é obtido através de um conjunto de cinco disciplinas de Fundamentos nas várias ênfases do curso (sistemas de computação, sistemas de informação, linguagens e teoria da computação, sistemas de automação e controle digital e computação científica).

1.3.4 As ênfases do BCC

Como já deve estar claro, o nosso curso é composto por cinco ênfases. A escolha por uma das ênfases é feita ao final do segundo ano, quando teoricamente já foram cursadas todas as disciplinas de fundamentos. Nessa versão do currículo o processo de escolha por ênfase é livre, isso é, pode-se escolher qualquer ênfase independentemente do desempenho do aluno.

Ao escolher uma ênfase o aluno passa ater que cumprir um conjunto de seis disciplinas obrigatórias daquela ênfase (três consideradas básicas e três avançadas). Além das disciplinas de sua ênfase cada aluno tem que cursar quatro disciplinas complementares e outras quatro optativas, sendo que as complementares devem ser escolhidas entre as básicas das demais ênfases, uma complementar por ênfase. Já as optativas podem ser quaisquer outras disciplinas oferecidas (desde que não sejam obrigatórias nem estejam sendo computadas como complementares).

Desse modo garante-se que o aluno que integralizar o curso terá uma boa formação em alguma das ênfases e, ao mesmo tempo, uma formação razoável nas demais áreas (nas quais provavelmente não gostaria de trabalhar). Essa formação nas outras áreas é garantida primeiro pelas disciplinas de fundamentos e depois pela exigência da disciplina complementar.

1.3.5 O projeto final

Independente da ênfase escolhida, todo aluno precisa apresentar o chamado projeto final de graduação para completar o curso. Esse projeto é apresentado numa disciplina (chamada "Projeto Final"), que pode ser cursada a partir do sexto semestre (normalmente no oitavo).

Esse projeto pode ser realizado tanto dentro do próprio Ibilce, fazendo parte do projeto de pesquisa de algum docente, quanto em empresas, durante o estágio do aluno. Note-se que não existe a obrigação de um estágio, embora isso seja bastante interessante para aqueles que queiram ir para o mercado de trabalho corporativo.

A exigência básica do projeto final é de que ele envolva um trabalho cuja realização apenas seja possível com os conhecimentos adquiridos nas disciplinas das várias ênfases, ou ainda além delas em alguns casos. Essa característica faz com que grande parte dos estágios oferecidos aos nossos alunos não possa ser caracterizada como um projeto final. Infelizmente essa é uma situação de mercado que aos poucos estamos mudando (através de nossos ex-alunos) e que faz com que muitos alunos evitem a realização de estágios pelo medo de uma carga de trabalho dobrada.

Com o passar dos anos a nossa experiência diz que é possível a realização de um estágio e de um projeto final, desde que o projeto final comece a ser trabalhado bastante cedo. De fato, começar o trabalho do projeto final ainda na metade do terceiro ano é uma tendência que tem apenas produzido bons resultados, pois quando o aluno finalmente se matricula na disciplina Projeto Final ele já está, na prática, como o trabalho praticamente concluído.

1.4 Considerações finais

Vimos durante esse capítulo um pouco da estrutura dos cursos superiores na área de computação e informática no país. Nesse processo detalhamos os perfis dos quatro cursos de graduação plena, fechando essa descrição com o curso de bacharelado em ciência da computação. Após a revisão dos tipos de cursos de graduação gastou-se um tempo com uma descrição do curso de BCC oferecido na Unesp/Rio Preto, encerrando então nossa discussão.

Nos próximo capítulos teremos a essência real dessa disciplina, que é a de mostrar porque certos tópicos de matemática são necessários para se fazer computação. Trabalharemos inicialmente com as disciplinas do primeiro semestre do curso, sempre numa abordagem em que primeiro vocês tomarão contato com alguma aplicação interessante e depois verão que ela apenas é possível com o uso de matemática.

Capítulo 2

Geometria vetorial

Uma das áreas de matemática com uso mais intenso em computação é o de geometria. Isso se deve principalmente à importância de interfaces gráficas, nas quais imagens representam informações.

O que veremos nesse capítulo é um resumo do que é o estudo de geometria analítica (GA) num curso superior e quais partes da mesma são importantes para a resolução de diversos problemas computacionais. Para tornar mais interessante nosso estudo trabalharemos inicialmente sobre um aplicativo de grande uso na engenharia, que é uma ferramenta de auxílio ao projeto (CAD), de software livre chamada **qcad**. Depois de usarmos o qcad faremos um exame do que é preciso para se projetar esse tipo de software e o que a geometria analítica ajuda nesse processo.

2.1 O qcad

Como mencioando, o qcad é uma ferramenta de apoio ao projeto que permite a elaboração de desenhos de peças através do computador. O qcad não é uma ferramenta sofisticada como o AutoCAD, porém permite um bom entendimento do processo de projeto e também a realização de projetos interessantes em duas dimensões. Claro que existem CADs muito mais poderosos que ele, começando já pelo fato de permitirem visualizações em três dimensões e análises (simulações) de movimentos (para verificar possíveis interferências). Os mesmos não apresentam, entretanto, nenhuma diferença conceitual que faça com que suas necessidades em termos de ferramental matemática sejam menores (ou mais simples) do que o qcad.

O trabalho a ser realizado no qcad é bastante simples. Basta desenhar a peça descrita a seguir (figura 2.1), colocando todas as dimensões e tudo o mais no desenho.

Feito isso, deve-se realizar algumas modificações no mesmo, procurando verificar como é fácil desenhar (e alterar) uma peça ou mesmo um conjunto de peças (que formariam máquinas) usando-se o computador.

A questão que fica é como o computador está tratando as imagens que você criou e manipulou? A resposta vem das formas como computadores armazenam as informações representadas nas imagens.

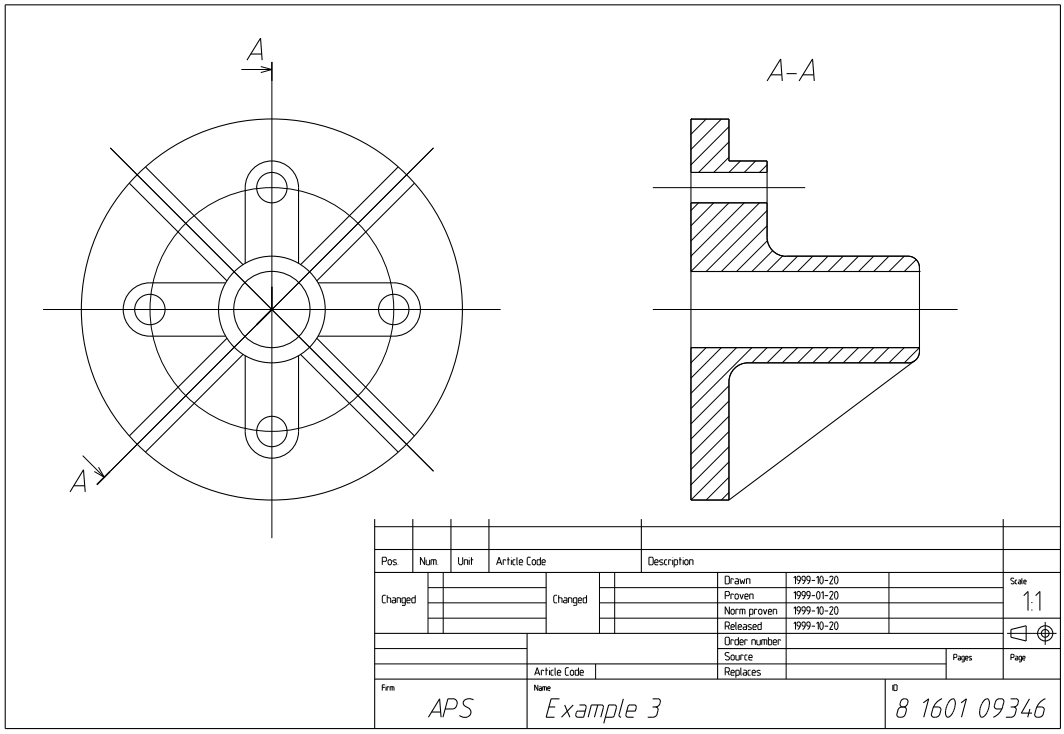

Figura 2.1: Exemplo para uso do qcad

2.2 Armazenamento e manipulação de imagens

Nossos computadores apresentam suas imagens no chamado formato de *pixels*. Um *pixel* é na realidade um ponto da imagem (um elemento do desenho). Os monitores de vídeo são vendidos com resoluções de x pontos por polegada, sendo que quanto mais pontos melhor a resolução (porém com mais memória para guardar as imagens).

Essa não é a única forma de tratar imagens. Um outro tipo de monitor (dito vetorial ou rasterizado) trata as imagens não como um conjunto de pontos mas sim como um conjunto de linhas desenhadas em qualquer direção. Neles a resolução passa a depender do feixe de fótons colidindo com a superfície do monitor, sendo que quanto mais largo o feixe pior a resolução.

A existência desses dois tipos de terminais permitiu que as imagens pudessem ser armazenadas também de dois modos: por *pixel* e por linha. O primeiro caso funciona para monitores comuns, sendo que o segundo é mais adequado para monitores rasterizados (que são raros). Tudo seria simples se a apresentação das imagens tivesse que ser feita exatamente na sua forma de armazenamento, o que não é verdade (felizmente).

Do que foi visto com o qcad é mais ou menos fácil de perceber que uma ferramenta daquele tipo armazena e trata suas informações no formato vetorial. Fica muito difícil alterar imagens já salvas se tivermos que alterar todos os pixels da mesma, o que não ocorre se temos que alterar duas ou três linhas quando as imagens são armazenadas através de elementos geométricos.

Assim, a imensa maioria dos aplicativos para desenhos faz uso de elementos geométricos para armazenar todas as informações sobre o desenho (claro que isso vale para desenhos com

perfil geométrico e não para pinturas e fotos). A seguir apresentamos uma série de formas para a definição de elementos geométricos.

- Pontos
  1. Usando o cursos para apontar a posição na tela
  2. Dando suas coordenadas pelo teclado
  3. Dando o deslocamento (x,y,z) a partir do ponto atual
  4. Intersecção de duas linhas
  5. Marcando pontos em intervalos fixos num elemento
- Linhas
  1. Usando dois pontos já definidos
  2. Um ponto e ângula da linha com a horizontal
  3. Um ponto e a linha normal ou tangente a uma dada curva
  4. Um ponto e alinha paralela ou perpendicular a outra
  5. Tangente entre duas curvas
- Arcos e circulos
  1. Usando centro e raio
  2. Especificando o centro e um ponto no circulo
  3. Curva passando por três pontos já definidos
  4. Curva tangente a três linhas
  5. Dando o raio e tangente a duas linhas ou curvas
- Cônicas
  1. Especificando cinco pontos do elemento
  2. Especificando três pontos e uma condição de tangência
- Curvas
  1. Métodos matemáticos (numéricos) como b-splines, splines cúbicas ou Bezier, em que se faz o ajuste da curva entre dois pontos específicos através de interpolação
- 1. Os métodos para geração de curvas são utilizados para criar as linhas que formama superfície, ou por superfícies de revolução ou por intersecção de duas superfícies

Embora o armazenamento final das imagens ocorra na forma de elementos geométricos, a tela dos nossos computadores manipula apenas pixels. Assim, é preciso que se converta a imagem representada por um elemento geométrico (por exemplo *CIRCULO(RAIO=20, CENTRO(100,50))* numa suposta representação de um circulo através de seu raio e centro) para um conjunto de pontos que a represente graficamente.

Esse processo resulta em uma certa imprecisão na imagem, como pode ser vista na figura 2.2, que apresenta os pixels exageradamente ampliados. O problema é quais pixels devem ser usados em cada trecho da linha de modo a torna-la aparentemente continua.

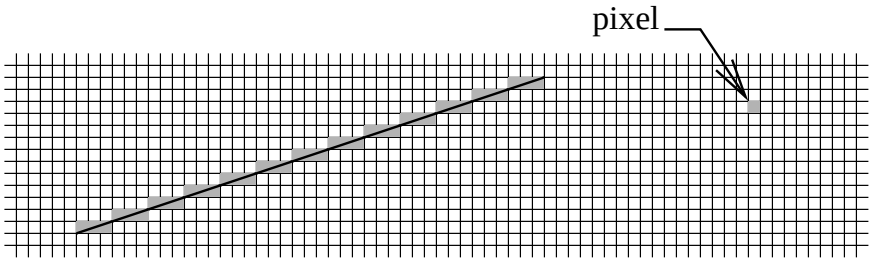

Figura 2.2: Imprecisão entre pixels e elementos geométricos

Ao transformar as imagens em pixels surge um segundo problema, que é o de ter que manipular grandes quantidades de dados para aplicar pequenas transformações numa figura, como desloca-la da posição atual, gira-la num certo ângulo ou mesmo amplia-lar ou reduzi-la. Em todos os casos esse processo envolve uma série de manipulações algébricas sobre matrizes. Essas operações são aplicadas sobre cada ponto da imagem, resultando na necessidade de um grande poder de processamento para fazer tais atualizações. A seguir apresentamos as operações necessárias para transformações em duas dimensões. Para três dimensões temos matrizes $3X3$, sendo que para rotações é preciso uma matriz para cada eixo.

Transformações no plano

Nas transformações a seguir considerar que o ponto em sua posição original é representado pela matriz $\mathbf{A = (x,y)}$ e que suas novas coordenadas, após a transformação, serão dadas pela matriz **B**.

- **Translação** B = A + C

  em que $\mathbf{C = (m,n)}$, com m correspondendo ao deslocamento no eixo X e n no eixo Y;

- **Escala** B = A . C

  em que $C = \begin{vmatrix} m & 0 \\ 0 & n \end{vmatrix}$, sendo m e n respectivamente os fatores de escala nos eixos X e Y;

- **Rotação** B = A . C

  em que $C = \begin{vmatrix} \cos(\alpha) & sen(\alpha) \\ -sen(\alpha) & \cos(\alpha) \end{vmatrix}$, sendo α o ângulo de rotação desejado (em relação ao ângulo atual formado entre o ponto, a origem dos eixos e o eixo X);

Representação por elementos geométricos

É mais ou menos evidente que as transformações de escala, translação e rotação se tornam muito mais simples se a representação da figura for feita por elementos geométricos, em que se aplicaria as matrizes de transformação apenas aos pontos característicos de cada elemento (ponto inicial e final de uma linha, por exemplo), ajustando-se os demais pontos a partir dos mesmos.

Uma outra vantagem na representação geométrica está no momento de armazenar a imagem. Examinando a figura 2.3 fica fácil de perceber que a representação por pixels implicaria, por exemplo, no armazenamento de 80.000 pontos (um byte por ponto se estamos trabalhando com apenas 255 níveis de cinza), enquanto que a figura completa pode ser armazenada com a representação de três elementos (um círculo, um quadrado e um triangulo),

adicionados de relações de pertinência e profundidade, ao dizer que a figura X é dada pela união do quadrado e do triangulo, excluindo-se da mesma parte do círculo, e que tem uma profundidade (largura) p.

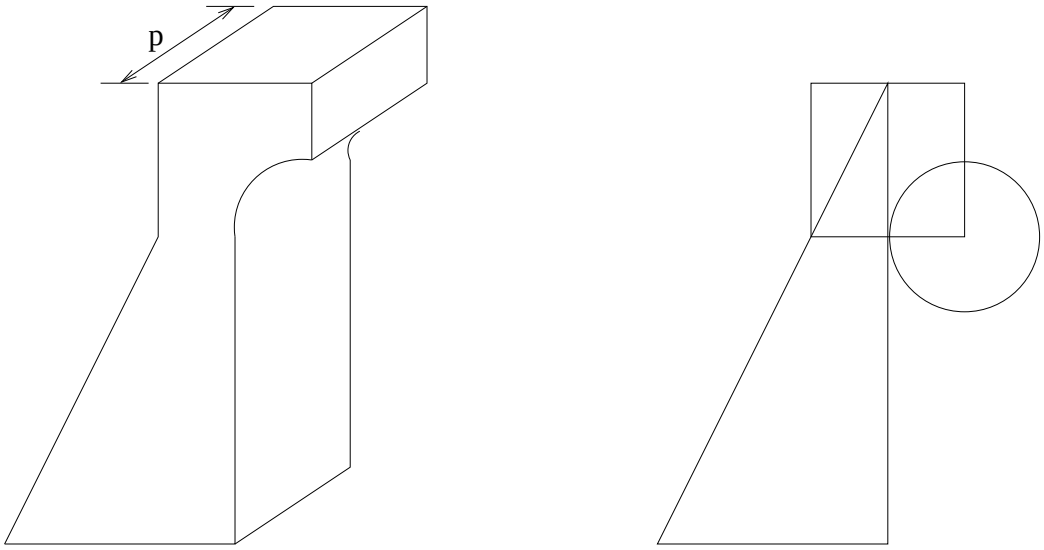

Figura 2.3: Representação por elementos geométricos

2.3 A disciplina de Geometria Analítica e Vetores

Do que examinamos até o momento não transparece que a matemática necessária para a implementação de um CAD seja complicada. De fato, grande parte das manipulações examinadas são simples aplicações de regras geométricas simples. Então, qual a razão de todas as equações e da manipulação de vetores que aparece em GAV?

A resposta também é simples. Perceber que a representação de desenhos através da composição de elementos geométricos é fácil. Difícil é encontrar a técnica mais simples de compor tais elementos se não se sabe como as equações que os representam podem ser associadas e manipuladas. É para isso que a geometria aparece. É também para isso que os vetores aparecem.

Embora provavelmente ainda não tenham aparecido em GAV, os vetores tomam um importante papel nesse processo todo. Na prática um vetor nada mais é do que a representação matemática de uma magnitude e direção no espaço. Isso significa que podemos representar linhas através de um vetor e, principalmente, usar vetores como poderosas ferramentas no momento de definir linhas, curvas, etc., através de pontos e formas tangenciais, paralelas ou normais a outros elementos.

Além disso, as transformações que acabamos de ver para escala, rotação e translação também podem ser traduzidos como operações sobre vetores. Assim, é de fundamental importância que se aprenda corretamente como vetores funcionam e como são as operações sobre os mesmos.

Uma extensão também importante de vetores são os chamados espaços vetoriais, os quais permitem a definição de superfícies de uma forma mais simples do ponto de vista geométrico. Os espaços vetoriais são inicialmente vistos em GAV, mas são intensamente trabalhados na

disciplina de álgebra linear.

2.4 Considerações finais

A manipulação de entidades geométricas é o corpo principal de aplicações de GAV em computação. Seu uso pode ser na construção de imagens, como acabamos de examinar, como também no processamento de imagens capturadas por algum dispositivo (camêras de vídeo, fotográficas, etc.).

No caso de processamento de imagens podemos usar os conceitos de GAV para, por exemplo, localizar elementos elipsoidais em uma mamografia (indicariam a existência de calcificações possivelmente cancerosas). Podemos usar a mesma forma de tratamento para identificar biscoitos defeituosos numa esteira de empacotamento de uma indústria, eliminando aqueles que não tenham a forma exata de um círculo ou um quadrado, por exemplo.

As aplicações em reconhecimento de padrões são bastante facilitadas se, ao invés de tratarmos ponto por ponto das imagens, pudermos tratar alguns pontos e através de uma representação algébrica do padrão confirmarmos a existência ou não dos demais pontos. Esse processo é conhecido como Transformada de Hough, tendo grande aplicação em reconhecimento de imagens.

Já as aplicações de reconhecimento de imagens em nosso dia a dia são bastante numerosas e diferenciadas, indo desde sistemas de controle de qualidade até sistemas de direção automática de veículos, passando por aspectos ainda difíceis de atingir como é o caso de recuperação artificial da visão humana.

Abrindo um parênteses, é preciso dizer que a matemática apresentada aqui como sendo útil ao campo de computação gráfica e de processamento de imagens não é, nem de perto, toda a matemática necessária nesses campos. Não apresentaremos aqui tópicos como transformadas de Fourier e outros conceitos igualmente importantes por estarem muito distantes daquilo que facilmente compreenderiamos aqui. A conclusão aqui é que essas áreas são usuárias intensas de matemática, como veremos aliás nos próximos capítulos.

Capítulo 3

Cálculo

A computação foi criada inicialmente para a resolução de problemas numéricos. Tanto é que a palavra *"computer"* (traduzida para computador) quer dizer aquele que faz contas. Isso já nos dá uma idéia da importância do cálculo dentro da computação. Aqui nossa preocupação é diferente, isso é, não nos interessa mostrar que computadores resolvem problemas de cálculo e sim que técnicas de cálculo resolvem problemas de computação.

Partindo desse ponto, consegue-se enxergar dois diferentes campos de aplicação do cálculo integral em computação, o primeiro em que o cálculo é usado como uma ferramenta direta na construção de uma aplicação; e o segundo em que ele é usado como base para uma outra técnica matemática que será, por sua vez, usada na construção de alguma aplicação. Como exemplo do primeiro campo temos o uso de equações diferenciais (já veremos o que são) na resolução de problemas de robótica e outros dispositivos móveis. Para o segundo campo temos a construção de métodos probabilísticos, os quais podem ser usados em aplicações como filtragem de imagens digitais, simulação de sistemas e na análise de desempenho de sistemas, que será a aplicação inicial em que trabalharemos.

3.1 Prevendo o desempenho de um sistema

A previsão do desempenho de um sistema qualquer, computacional ou não, é uma tarefa de grande importância para o projeto desse sistema. Essa importância vem do fato que quanto melhor o desempenho de um sistema mais atrativo o mesmo se torna, uma vez que todo mundo quer o ótimo segundo alguma métrica. Assim, sempre procuramos ou pelo carro mais econômico ou mais rápido ou mais confortável. A escolha pelo carro ideal faz uma composição desses parâmetros segundo a preferência do comprador.

O mesmo vale para um computador ou para um programa de computador (velocidade, custo, estabilidade, etc.), chegando portanto ao nosso campo de interesse. No caso de computadores e programas o que se busca sempre é uma boa relação custo-desempenho, em que desempenho normalmente significa velocidade de execução. Na prática, quando temos programas muito grandes (alguns levam dias para resolver um problema) passa a ser crucial atingirmos o desempenho ótimo, economizando tempo de máquina e de manutenção de pessoal (esperando pela resposta).

A questão é, então, como atingir um desempenho ótimo partindo-se das fases iniciais de projeto do programa ou do computador, que chamaremos de sistema computacional daqui por diante. Prever o desempenho de um sistema computacional antes mesmo de sua existência é importante para evitar que determinadas decisões em seu projeto inviabilizem

a obtenção de um ótimo desempenho. Com isso o nosso problema passa a ser como obter medidas de desempenho se não temos o sistema funcionando. Uma questão parecida (e até mais importante) é determinar como será o desempenho de um sistema computacional em funcionamento daqui há alguns meses ou anos.

Para ambos problemas a solução está no uso de ferramentas para predição de desempenho (ou análise de desempenho). Essas ferramentas existem em três formatos básicos:

1. ***Benchmarks***, em que os dados para análise são obtidos através de medições fisicamente realizadas sobre o sistema real. É óbvio que isso não é possível se o sistema ainda não existe ou se as condições de teste não podem ser efetivadas;

2. **Simulação**, em que se cria um modelo do sistema e, através de algum mecanismo de imitação de seu funcionamento, são obtidas medidas de seu desempenho. Pode ser aplicada quando o sistema não existe, mas tem sua precisão determinada pela precisão em que o modelo imita o sistema;

3. **Resolução analítica**, em que as medidas de desempenho são obtidas pela resolução analítica de um modelo que represente o sistema real através de equações matemáticas. Assim como simulação, a resolução analítica pode ser aplicada em qualquer fase do projeto de um sistema computacional e, do mesmo modo, tem sua precisão determinada pela fidelidade com que as equações representam o sistema.

As aplicações em que modelos de resolução analítica são úteis incluem, por exemplo, a determinação de quanto tempo um dado programa levaria para executar em uma dada máquina, qual a capacidade de transmissão é necessária para que uma rede de computadores atenda uma determinada demanda, qual volume de serviços um provedor *web* pode atender com uma dada configuração, etc.

Uma técnica para resolver tais problemas é o uso de teoria de filas, a qual explicaremos mais adiante. Para entender como podemos extrair resultados como os indicados no paragrafo anterior iremos trabalhar um pouco com uma ferramenta que simula aplicações em filas.

3.2 Simulador de filas

O simulador que vocês usarão é cortêsia de Ran Shalgi [6], que forneceu uma série de métodos escritos em Java, que podem ser executados na forma de *applets*, diretamente de algum navegador. Embora sem os recursos necessários a um simulador de filas real, esse aplicativo é bastante visual e permite um melhor entendimento do que acontece na execução de um sistema de filas.

Esse simulador é composto de três janelas. Uma que apresenta o conjunto de servidores e filas existentes, outra com mensagens sobre o estado do sistema de filas e uma terceira com estatísticas sobre o sistema. O controle da simulação é feito através da janela principal, com os botões "Start Simulation", "Stop Simulation" e "Redefine", como pode ser visto na figura 3.1.

O simulador define uma fila de atendimento e um padrão de serviços (com um ou mais servidores). A teoria de filas trabalha exatamente na criação de equações matemáticas para determinar o tamanho das filas, o tempo médio de espera por atendimento, o quanto que o servidor fica ocupado, etc. Nos exemplos que serão examinados podemos enxergar os

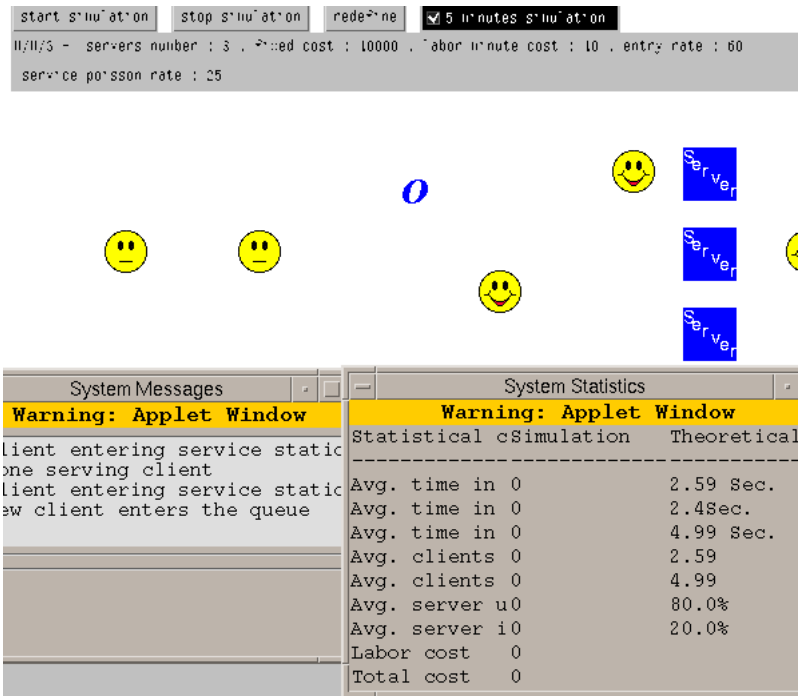

Figura 3.1: Tela do simulador de filas

clientes da fila como sendo mensagens trafegando numa rede e os servidores como canais de comunicação em uso (ou servidores de e-mail, etc.). Na prática, a mudança entre um sistema físico (problema a ser resolvido) e outro é apenas como as equações serão montadas.

Acessem a página *http://www.dcce.ibilce.unesp.br/ãleardo/cursos/fcc/queue/applet.html* e executem o simulador pressionando o botão de início. Esperem alguns minutos (dois ou três) para ter uma idéia do seu andamento. Depois disso anotem os valores da janela de estatísticas e se preparem para uma segunda execução, exatamente com o mesmo sistema. Executem pelo mesmo tempo e verifiquem os valores estatísticos. Compare os resultados e discuta com os colegas.

O terceiro passo é redefinir o sistema. Numa primeira tentativa use o mesmo sistema (M/M/S) e altere apenas o número de clientes entrando por minuto, passando-o de 60 para 75. Execute a simulação e veja o que ocorre.

Faça outras alterações, principalmente mudando o tipo de sistema (M/M/1, M/F/1, M/U/1 e M/N/1), em que se altera o padrão de atendimento dos servidores.

As conclusões a serem tiradas desse experimento é que podemos usar teoria das filas para modelar um sistema real, do qual queremos ter informações que não podem ser medidas diretamente, e que precisamos entender o que é essa tal de teoria de filas. O primeiro passo para esse entendimento é dado a seguir.

3.2.1 Teoria de filas

Como o próprio nome diz, teoria de filas trata de como as filas se comportam. Matematicamente uma fila nada mais é do que um conjunto de elementos que constantemente é alterado pela inserção de novos elementos e pela remoção de elementos antigos, sendo que a ordem de remoção é a mesma da inserção (ou seja, o primeiro a entrar na fila é também o primeiro a sair dela).

Como podemos, então, definir as taxas de inserção e de remoção na fila? Essas taxas são, em sua maioria, não-determinísticas, isso é, não podem ser consideradas constantes (ou fixas). Isso implica em que tais taxas são probabilísticas, ou seja, existe uma certa probabilidade de que clientes sejam cheguem numa dada freqeüência e que os servidores demorem um certo intervalo de tempo para atender cada cliente. Esses valores variam, entretanto, de cliente para cliente, segundo uma dada função de distribuição de probabilidades (nos exemplos vimos as funções normal, uniforme, exponencial e de poisson).

O problema em teoria de filas é, portanto, a identificação de qual função de distribuição de probabilidades melhor representa as taxas de chegada e atendimento do sistema sendo modelado. Quanto melhor for essa representação, melhores serão os resultados extraídos do modelo.

O poder de abstração de teoria de filas depende apenas da capacidade de entendimento do analista quanto ao que o sistema é e como ele funciona. Apesar de nosso simulador trabalhar apenas com uma fila, é perfeitamente possível usarmos combinações variadas de filas e servidores, tanto em cascata como em malha fechada (os clientes sempre voltam ao início da fila).

A questão passa a ser como identificar as diferentes distribuições de probabilidade e qual a matemática que torna as mesmas tratáveis. Essa matemática é o cálculo e a identificação das distribuições depende de um bom entendimento de estatística e do sistema a ser modelado.

3.3 Breve introdução ao cálculo

Dentro do cálculo existem dois operadores fundamentais, a derivada e a integral de funções. A partir desses operadores podemos resolver equações bastante complexas, inclusive as chamadas equações diferenciais, que são identificadas por reunirem numa mesma expressão uma função e sua derivada.

Conceito geométrico de derivada

A derivada de uma função pode ser entendida geometricamente como sendo a inclinação da reta tangente à função calculada ponto a ponto, isso é, a derivada de uma função também é uma função, mas que representa as inclinações das tangentes á primeira função. Essa noção, embora imprecisa e totalmente intuitiva, é bastante útil em algumas aplicações da derivada, tais como no cálculo de pontos de máximo e/ou mínimo de uma determinada função. Nas disciplinas de Cálculo I e II podem ser vistos mais detalhes e melhores conceitos do que esse, os quais podem ser encontrados também nos bons livros de cálculo, que ficam como uma indicação para um estudo referencial nesse tópico.

Conceito geométrico de integral

A integral, que se define matematicamente como sendo o operador inverso da derivada, tais como multiplicação e divisão, pode ser entendida geometricamente como sendo a área contida entre uma dada função e o eixo sobre o qual está sendo integrado, isso é, se a integração for em relação a variável x, então o resultado diz, respeito à área contida entre a curva da função e o eixo x. Assim como para o caso das derivadas, não nos alongaremos nessa conceituação pois uma melhor oportunidade para isso ocorre dentro das disciplinas de Cálculo I e II.

Equações diferenciais

Essa classe de equações, vistas em Cálculo III, não tem uma interpretação imediata, como é o caso das operações anteriores. Mas podemos tentar apresentá-la através de um exemplo clássico da física de movimentos, isso é, achar a equação de movimento de um corpo de massa m em queda no espaço, com resistência do ar dada por uma constante K. Em notação de cálculo temos que essa equação pode ser tirada da seguinte forma:

$F = m.a$ em que a aceleração de um corpo é dada pela derivada da sua velocidade com relação ao tempo, ou seja,

$F = m.\frac{dv}{dt}$ EQ. 1

Mas também sabemos que a força agindo sobre um corpo durante sua queda é a resultante de duas componentes, uma dada pela ação gravitacional (m.g) c outra pela ação da resistência do ar (K.v) as quais têm mesma direção porem sentidos opostos, ou seja:

$F = m.g - k.v$ EQ. 2

Igualando **EQ. 1** e **EQ. 2** temos:

$m.\frac{dv}{dt} = m.g - K.v$ EQ. 3

Podemos notar dessa equação que a mesma apresenta a velocidade e sua derivada em dois pontos, logo fica impossível resolvê-la de forma tradicional pois para determinar a velocidade temos que saber sua derivada, a qual é desconhecida e que para ser determinada precisa da velocidade. A forma de resolver isso é através da resolução de equações diferenciais, as quais resultam numa equação que representa uma família de soluções e não apenas uma única e bem determinada solução. Para que a **EQ. 3** tenha uma única solução é necessário que se estabeleçam certas condições de contorno, que nada mais são do que restrições físicas para o problema. Por exemplo, para a **EQ. 3** temos a seguinte solução geral:

$$v = C.e^{\frac{-Kt}{m}} + \frac{m.g}{K}$$

De modo a obtermos uma solução particular para essa equação temos que determinar o valor da constante C, o que é feito através da criação de condições de contorno. Nesse caso uma condição de contorno típica seria considerar a velocidade $v=0m/s$ quando $t=0s$. A determinação de qual seria a equação resultante fica sugerida como um bom exercício ao leitor.

3.4 Algumas aplicações: uma descrição rápida

3.4.1 Uso de equações diferenciais para o controle de dispositivos móveis

Uma aplicação clássica de equações diferenciais é o controle de movimento de dispositivos controlados por computador, em que todo o dimensionamento de trajetória, incluindo-se velocidade, posição e aceleração, é feito através da resolução de equações diferenciais nos moldes da **EQ. 3**, trocando-se a resistência do ar pela força de atrito, por exemplo.

3.4.2 Uso de integrais em computação gráfica

Dentro de computação gráfica uma aplicação implícita de técnicas de integração é o uso de filtros baseados em equalização de histogramas para a melhoria de imagens. O que se faz é procurar distribuir de forma mais constante a quantidade de pontos da imagem com cada nível de cinza. Com esta técnica podemos, por exemplo, tornar uma imagem com muitos pontos escuros numa imagem em que uma parte desses pontos escuros é transformada em pontos mais claros, através da equação de transformação dada a seguir.

$$q(p) = \frac{M}{N^2} \int_0^p h(s)ds$$

Em que M é o número de níveis de cinza na imagem e N é o número de *pixels* em cada eixo da imagem (pensando em imagens quadradas).

Vale observar que nesse caso a operação de integração faz parte de um contexto diferente daquele usado no controle de trajetória, isso é, a integral foi formulada a partir de conceitos do que é um histograma, que nada mais é do que uma ferramenta de análise estatística de dados. Num histograma temos em um eixo os diferentes valores possíveis para uma variável (os tons de cinza por exemplo) e no outro temos as quantidades de elementos da amostra que possuem um dado valor (os *pixels* de uma imagem seriam esses elementos).

Como exemplo da aplicação da técnica de equalização de histograma podemos ver a figura 3.2. Nessa figura temos na parte superior a imagem original, mais escura e com um histograma que mostra que a maioria dos *pixels* está em um dos extremos do eixo de tons de cinza. Na parte inferior, após a equalização do histograma (reparem na melhor distribuição de *pixels* ao longo do eixo) temos uma imagem mais clara e nítida. Alguns outros exemplos, incluindo uma versão colorida deste, podem ser vistos na página dessa disciplina, em *http://www.dcce.ibilce.unesp.br/ãleardo/cursos/fcc/histogr/imagens.html*.

3.5 Usando cálculo em probabilidade e estatística

O que veremos a seguir é a construção de ferramentas da probabilidade e estatística a partir dos operadores de cálculo integral. Isso é possível quando fazemos o exame de alguns princípios básicos da teoria de probabilidades usando a ótica do cálculo integral. Assim, podemos fazer a construção de funções de distribuição de probabilidades a partir da integração de funções de densidade de probabilidades, determinarmos valores médios, modas e medianas de funções de densidade, etc.

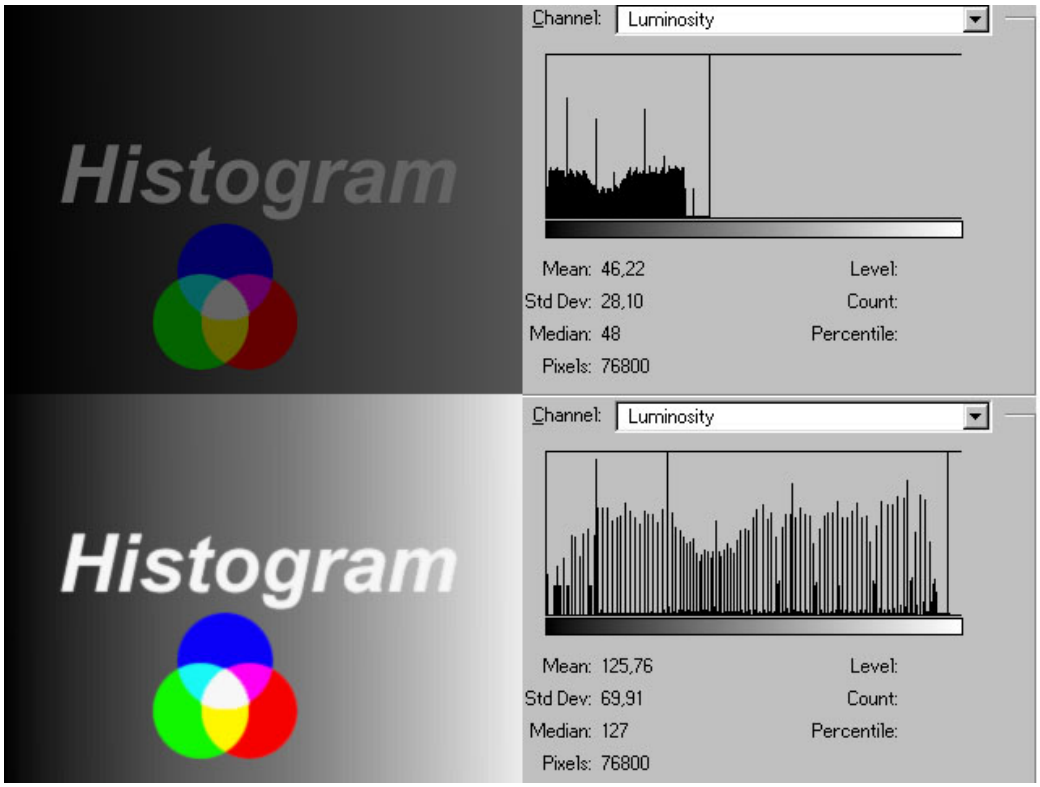

Figura 3.2: Equalização de histograma

Primeiro é necessário compreendermos o significado físico dessas funções, isso é, o que elas significam conceitualmente. Começando então pela *função de densidade de probabilidades* (**fdp**), temos que ela nos indica qual a probabilidade de ocorrência de um certo evento dentro de um dado espaço amostral, lembrando que o espaço amostral nos dá o conjunto de todos os eventos possíveis. Assim, ao jogarmos uma moeda para o alto, a probabilidade de ocorrência de cara é igual à de ocorrência de coroa, sendo ambas iguais a 1/2. Do mesmo modo, se jogarmos dois dados, o conjunto de valores possíveis é igual a 11 (do 2 até o 12), sendo que nem todos os valores teriam igual probabilidade. A fdp seria uma função discreta com os valores vistos na tabela 3.1.

x	2	3	4	5	6	7	8	9	10	11	12
fdp	1/36	2/36	3/36	4/36	5/36	6/36	5/36	4/36	3/36	2/36	1/36

Tabela 3.1: Distribuição de probabilidades para arremesso de dois dados

A fdp também pode ser uma função contínua, caso o seu espaço amostral seja contínuo (ou possa ser aproximado como contínuo), como aparece na figura 3.3.

Ocorre, no entanto, que para muitas aplicações o uso da fdp não produz informações suficientes, ou pelo menos não nos dá essas informações de forma direta. Um caso simples em que isso ocorre é quando queremos saber qual a probabilidade de que ocorra um evento maior que um conjunto de eventos. No caso dos dados poderíamos querer saber por exemplo qual a probabilidade de ocorrer um número menor ou igual a seis. Nessas situações o que

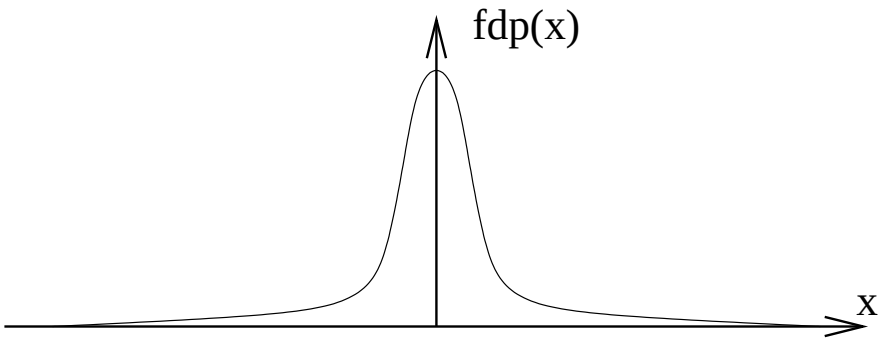

Figura 3.3: Função distribuição de probabilidades (fdp) contínua

se usa é a *função distribuição de probabilidades* (**FDP**), que nos dá o valor cumulativo das probabilidades de ocorrência dos eventos quando esses estão ordenados.

A FDP pode ser calculada a partir da fdp. No caso de fdp discreta, o que se faz é o somatório da função até o ponto de cálculo desejado. No exemplo dos dados, a FDP para o resultado menor ou igual a seis seria então a soma dos valores da fdp para 2, 3, 4, 5 e 6, resultando em 15/36. No caso da fdp ser contínua, esse somatório é transformado em uma integral, dada por:

$$F(x) = \int_{-\infty}^{x} f(u)du$$

em que $F(x)$ é a FDP e $f(u)$ é a fdp

Apenas a título de exemplo, a FDP da função apresentada no gráfico anterior resulta na função delineada na figura 3.4.

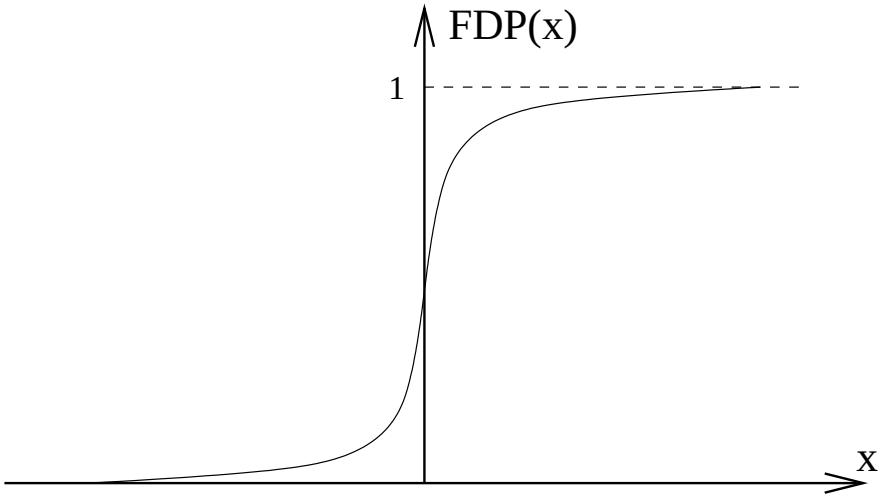

Figura 3.4: Função densidade de probabilidades (FDP) contínua

Outro fator a ser observado para a FDP é que o valor máximo da mesma é UM, indicando

que a probabilidade de ocorrência de algum evento é TOTAL. Isso significa que o valor da integral dada acima, quando x for infinito, será igual a 1. Esse resultado vai ser importante para a definição de algumas medidas estatísticas que veremos a seguir.

3.5.1 Momentos e medidas

Para fazermos medidas dentro da estatística é necessário que consideremos os momentos "angulares", ou pesos, que cada evento possui na medida. A fórmula geral para o cálculo de um momento é dada por:

$$E\{x^n\} = \int_{-\infty}^{+\infty} x^n f(x)dx$$

em que $f(u)$ é a fdp do problema em estudo

Média ou esperança:

A partir da equação anterior são tirados os valores de alguns momentos especiais. O primeiro deles é a média, ou esperança, de uma certa fdp, que é dada pelo cálculo da equação quando n=1. Assim, temos:

$$m = E\{x\} = \int_{-\infty}^{+\infty} xf(x)dx$$

O grande problema da média é que a mesma não leva em consideração a existência de poucos casos com valores muito extremos, que acabam prejudicando a sensibilidade da mesma. Além disso, existem fdp's que, apesar de contínuas e bem comportadas, não possuem uma média, por mais estranho que isso possa parecer. Um exemplo disso é a chamada Fórmula de Cauchy, que tem uma aparência normal, possui mediana e moda (outras duas medidas estatísticas) mas não possui valor médio.

Mediana:

Uma medida que é considerada melhor do que a média é a chamada mediana, que nada mais é do que o valor que divide a fdp em duas porções cujas áreas tenham o mesmo valor. Como se sabe que a área de uma função é a sua integral, temos que a FDP nos dá o valor da área de uma fdp. Como, além disso, sabemos que o valor da integral de fdp, por toda a reta, é igual a l, temos que o cálculo do ponto em que isso ocorre pode ser feito de forma relativamente simples, bastando descobrir o valor em que ocorre:

$$m = E\{x\} = \int_{-\infty}^{a} xf(x)dx = 1/2$$

Diferentemente do que ocorre com o valor médio, a mediana pode assumir valores múltiplos, basta que a fdp não seja bem comportada, tendo uma região (em que estaria contida a mediana) em que o valor da função é zero. Assim, todos os pontos contidos nesse intervalo são candidatos a serem a mediana por dividirem a fdp em duas áreas de mesmo valor. Na figura 3.5 temos um exemplo em que isso ocorre.

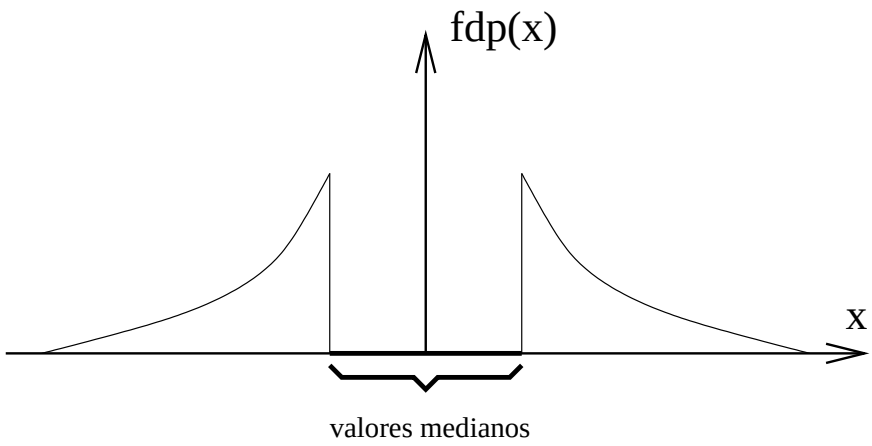

Figura 3.5: fdp com valores múltiplos de mediana

moda:

A moda de uma função fdp é o valor para o qual a função assume seu valor máximo, isso é, a moda é um valor que representa uma tendência da fdp em assumir um determinado valor, a sua moda. Matematicamente isso é definido como sendo o valor X_i tal que:

$$f(X_i) > f(X_i + \epsilon) \qquad \text{para todo } \epsilon \text{ próximo de } 0$$

Isso nos leva até a fórmula para o cálculo do máximo de funções, na qual temos que o valor máximo é encontrado nos pontos X_i para os quais a derivada primeira da função se iguala a zero, enquanto a sua derivada segunda possui um valor negativo (isso será visto em Cálculo I). Assim, para calcularmos a moda de uma fdp basta calcularmos suas derivadas.

Assim como ocorre com o valor mediano da fdp, a moda pode assumir múltiplos valores, isso é, uma dada fdp pode ser multimodal, o que significa que a mesma possui vários valores para os quais a fdp tende a seguir. Exemplos de funções multimodais podem ser encontrados quando a fdp possui platôs como um máximo ou ainda em funções em que o valor máximo se repita em momentos distintos, tal como funções senoidais. Na figura 3.6 temos uma fdp para cada um desses casos.

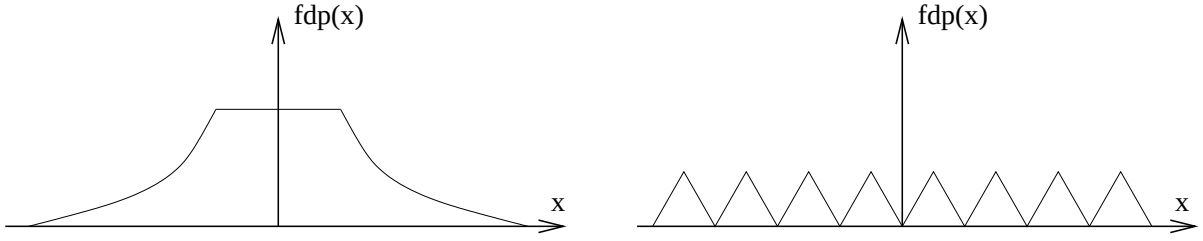

Figura 3.6: fdp's com modas múltiplas

3.5.2 Momentos centrais

Os momentos centrais de uma Função representam outras formas de medida estatística. O que eles fazem é verificar como os valores da fdp variam com relação ao valor médio da mesma. Esse tipo de medida é bastante útil quando queremos saber se as observações feitas em um dado experimento possuem ou não uma boa qualidade, isso é, se num dado experimento os dados estiverem muito dispersos, então os momentos centrais terão um valor grande, indicando que ou a experiência foi mal conduzida ou que o problema não possui um comportamento que possa ser reproduzido através de um valor médio. A fórmula geral dos momentos centrais é dada por:

$$\mu_n = E\{(x-m)^n\} = \int_{-\infty}^{+\infty} (x-m)^n f(x)dx$$

Variância:

Um dos momentos centrais bastante importantes é a variância, que é o momento central de ordem 2, ou seja, o momento central com *n=2*. Essa medida é usada como uma referência da distância entre cada ponto da fdp e o seu valor médio. A fórmula da variância então é dada por:

$$\sigma^2 = \int_{-\infty}^{+\infty} (x-m)^2 f(x)dx$$

Desvio padrão:

O maior problema no uso da variância é que a mesma não possui a mesma dimensão da média, isso é, enquanto a média é um momento de primeira ordem, a variância é de segunda ordem (ou quadrática). Para conseguir uma medida de primeira ordem, que tenha o mesmo significado da variância, calculamos o desvio padrão, que é simplesmente o valor positivo da raiz quadrada da variância, ou seja:

$$\sigma = +\sqrt{E\{(x-m)^2\}} = +\sqrt{\sigma^2}$$

3.6 Considerações finais

O uso de cálculo dentro da computação existe em diversas aplicações. Embora a grande maioria delas possa, de certa forma, ser entendidas como sendo aplicações de cálculo em outras ciências, existem também muitas aplicações em que o uso das ferramentas contidas em cálculo são fundamentais para que aplicações computacionais possam existir. Nesse capítulo fizemos um rápido exame em algumas aplicações, dando um maior destaque para o uso de cálculo na construção de ferramentas da teoria de probabilidades.

Vimos também que existem aplicações em que técnicas de resolução de equações diferenciais são utilizadas para a determinação de trajetórias de objetos móveis controladas por computador. Essas técnicas permitem que se resolvam problemas extremamente complexos

de movimento, incluindo-se equações vetoriais de movimento. Esse trabalho é necessário para que os objetos controlados possam ser efetivamente dominados, sem que ocorram problemas de colisões ou desvios de rota.

O enfoque maior em técnicas de probabilidade teve, na realidade, o objetivo secundário de mostrar que nem sempre aquilo que aprendemos tem uma aplicação direta naquilo que estaremos fazendo. Muitas vezes, como ocorre nesse caso, aquilo que está sendo aprendido vai ser utilizado como ferramenta básica para a construção de outras teorias, ou seja, aqui vimos como é que o cálculo diferencial e integral é útil para que possamos criar modelos para a teoria de probabilidades, a qual, por sua vez, é que nos serve para que possamos fazer simulações, análises de desempenho ou mesmo tratamento de imagens digitalizadas.

Encerrando, podemos agora ver claramente que a importância do cálculo não se resume simplesmente nos pontos em que o mesmo pode ser aplicado diretamente no desenvolvimento de modelos computacionais, sendo também bastante útil na construção de outras ferramentas matemáticas, aliás, essas ferramentas não se resumem apenas em técnicas probabilísticas, abrangendo a construção de modelos numéricos, técnicas de otimização, etc..

Capítulo 4

Álgebra linear

Álgebra linear faz parte do ferramental matemático essencial em diversas áreas de aplicação, tais como o estudo de engenharias, de ciências humanas, sociais e de computação, entre outras.

Álgebra linear é a parte da matemática voltada para o estudo de métodos para a manipulação de matrizes e vetores. Uma área em que tem larga utilização é a de resolução de equações e inequações lineares, em que se representa as equações através de matrizes. Por um ponto de vista mais geométrico estuda os espaços vetoriais, quando se descobre como arranjos numéricos retangulares (matrizes) podem representar linhas, ângulos e outros aspectos geométricos. O estudo das transformações lineares sobre matrizes mostra como certos dados de entrada podem ser transformados em saídas para um certo sistema.

Há diversas aplicações em que o respectivo modelo matemático resulta em uma matriz quadrada que depende de parâmetros importantes, mas desconhecidos, cujos valores devem ser ajustados de modo que ela passe a não possuir inversa. Na linguagem da álgebra linear, tais parâmetros são conhecidos como autovalores (eigenvalues), as matrizes não inversíveis são ditas singulares e os sistemas envolvidos nesta modelagem são conhecidos genericamente como autosistemas (eigensystems). A título de ilustração, podemos dizer que uma classe importante das aplicações mencionadas no início deste parágrafo são as originárias do modelagem de fenômenos oscilatórios, tais como o movimento das asas de aviões quando submetidas a diversas forças aerodinâmicas, ou como as variações da economia sob diversas forças de mercado, ou ainda como o comportamento dos inúmeros componentes de circuitos eletrônicos submetidos a diversas forças eletromagnéticas, etc... Portanto, também é na álgebra linear que se estudam autosistemas, suas diversas decomposições e formas canônicas, como ferramentas para o estudo de transformações lineares na modelagem do comportamento de sistemas complexos.

Neste capítulo estudaremos uma aplicação da álgebra linear na análise do modelo de um sistema computacional hipotético. Neste estudo serão vistos apenas os conceitos essenciais ao entendimento do exemplo proposto e, sempre que possível, de forma bastante intuitiva. Todos os conceitos aqui apresentados serão vistos com mais rigor e formalidade nas disciplinas Álgebra Linear, Métodos de Otimização I e Teoria dos Grafos, bem como seus aspectos numéricos e computacionais na disciplina Cálculo Numérico e, para alguns, problemas não-lineares em Métodos de Otimização II.

4.1 Roteamento de pacotes em redes de computadores

Um problema interessante dentro do contexto de redes de computadores é a otimização de rotas a serem seguidas pelos diversos pacotes em trânsito na Internet. Para se ter uma idéia da complexidade dessa tarefa, vejam a figura 4.1, que apresenta uma configuração das conexões entre os vários *backbones* dedicados à entidades acadêmicas e de pesquisa dos Estados Unidos. Se incluíssemos a parte comercial da rede naquela figura ficaria difícil de ver qualquer coisa.

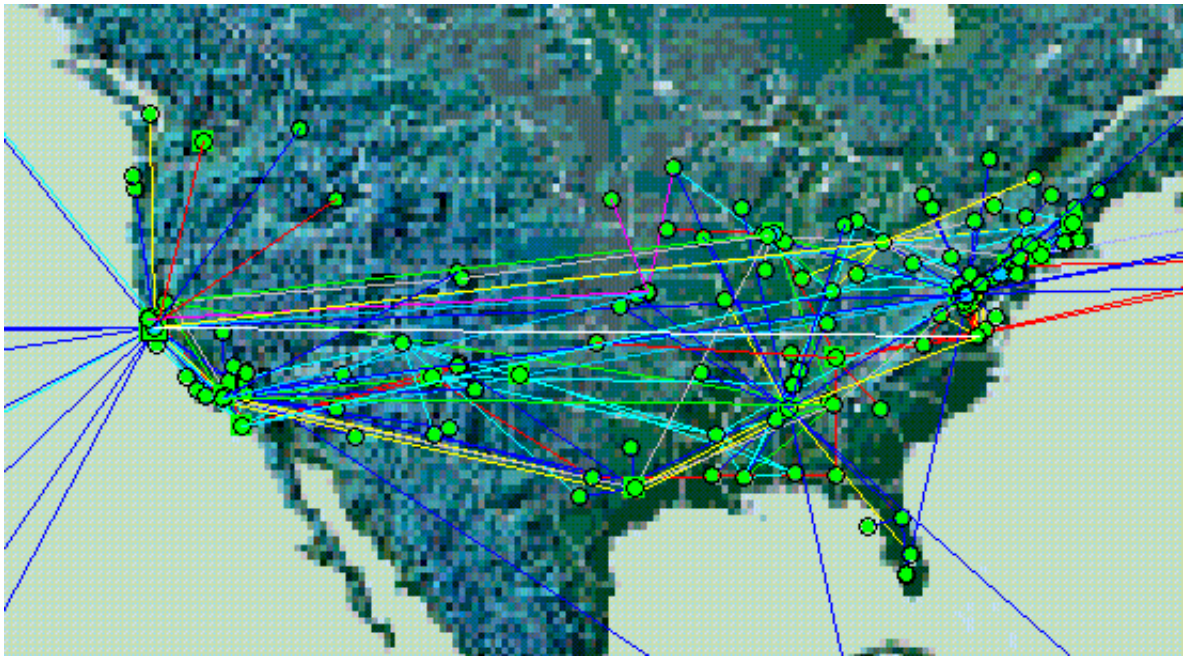

Figura 4.1: Malha de conexões da Internet acadêmica nos EUA. Extraída de *http://www.caida.org/tools/visualization/mapnet/Backbones*

Como já mencionado no capítulo anterior, um problema no processo de transmissão é a identificação de qual a melhor rota a ser seguida por cada pacote de informação. Lá o problema era identificar como os vários servidores atenderiam tais pacotes, independentemente dos motivos que gerassem a demanda pelo serviço. Aqui o problema é determinar como essa demanda é gerada de modo a não criar sobrecargas nem congestionamentos ao longo da rede, operando numa situação ótima (ou muito próximo dela). Esse problema faz parte da categoria de problemas de fluxo em redes (que tem variantes nas áreas de transporte, energia elétrica, manufatura, etc.).

Para entender exatamente do que estamos tratando, considere a malha de roteadores da Internet apresentada na figura 4.2. Para essa malha considere que temos, num dado instante, um conjunto de pacotes a serem transmitidos definidos na tabela 4.1 e que os tempos para envio dos pacotes é dado pela tabela 4.2, que apresenta os tempos de envio (sem contar a espera) em cada servidor e o tempo gasto no caminho entre cada par de servidores. Determine, então, qual a melhor rota para cada um dos pacotes da tabela 4.1.

Que técnica você aplicou para tentar fazer esse roteamento? Será que é possível definir uma técnica algébrica para fazer esse trabalho? Se isso for possível então também é possível

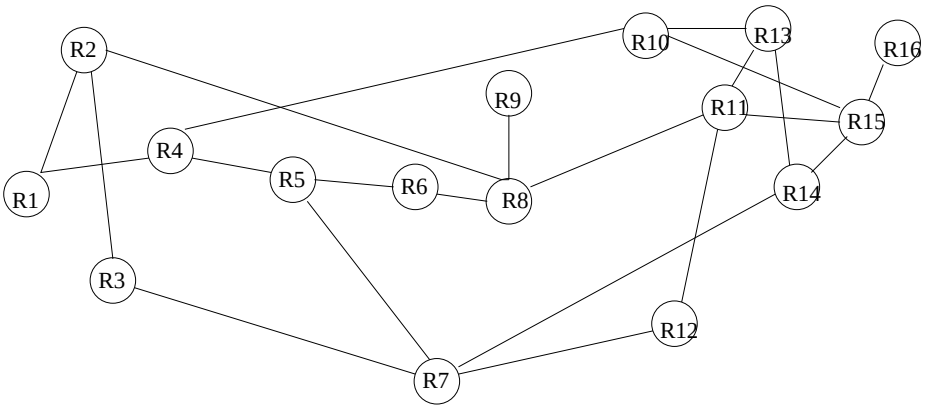

Figura 4.2: Malha de roteadores para exemplo

Roteador	Pacotes			
R1	R6	R2	R6	R14
R2	R5			
R4	R6	R3	R15	
R7	R1	R1	R3	
R11	R6	R15	R2	
R13	R12			
R14	R16	R3		
R15	R5	R9		

Tabela 4.1: Pacotes a serem transmitidos em cada roteador. O destino de cada pacote é dado pelo nome do roteador. A seqüência de envio é a que aparece listada.

construir um algoritmo para resolver esse problema. Felizmente a resposta é positiva, sendo que a técnica a ser utilizada vem da teoria de fluxo em redes. Assim como fizemos com teoria de filas no capítulo anterior, resta descrever o que é fluxo em redes e qual a matemática por trás da mesma.

4.1.1 Fluxo em redes

O conceito fundamental da determinação de fluxo em redes é o fato de que se existem pontos de parada na rede (seus vértices) e caminhos possíveis entre essas paradas (as arestas de ligação entre os vértices), então é possível otimizar os fluxos entre as paradas através, pelo menos, de busca exaustiva pela melhor solução. Claro que o uso de busca exaustiva inviabiliza o método, entretanto, a partir dela é possível fazer reduções do espaço de busca (eliminando soluções ruins) de modo a tornar o problema tratável.

Todo o trabalho dentro de fluxo em redes passa a ser, portanto, tentativas de encontrar bons algoritmos para a redução do espaço de busca. Dependendo da categoria do tipo de problema a ser resolvido (sua dinâmica) existem diferentes algoritmos para a determinação dos fluxos ótimos. No caso particular de roteamento de pacotes, um algoritmo bastante conhecido é o de caminhos mínimos proposto por Edger Dijkstra. Nesse texto não trataremos com mais detalhes esse algoritmo (nem outros para dizer a verdade) pois isso tomaria um

Roteador	T_{int}	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15	R16
R1	2	-	2	-	4	-	-	-	-	-	-	-	-	-	-	-	-
R2	2	3	-	2	-	-	-	-	2	-	-	-	-	-	-	-	-
R3	2	-	3	-	-	-	-	4	-	-	-	-	-	-	-	-	-
R4	1	1	-	-	-	1	-	-	-	-	5	-	-	-	-	-	-
R5	1	-	-	-	2	-	2	2	-	-	-	-	-	-	-	-	-
R6	2	-	-	-	-	3	-	-	2	-	-	-	-	-	-	-	-
R7	1	-	-	2	-	2	-	-	-	-	-	-	4	-	4	-	-
R8	1	-	3	-	-	-	1	-	-	2	-	3	-	-	-	-	-
R9	2	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
R10	2	-	-	-	2	-	-	-	-	-	-	-	-	2	-	1	-
R11	1	-	-	-	-	-	-	-	1	-	-	-	2	3	-	1	-
R12	2	-	-	-	-	-	-	2	-	-	-	2	-	-	-	-	-
R13	2	-	-	-	-	-	-	-	-	-	2	1	-	-	2	-	-
R14	2	-	-	-	-	-	-	1	-	-	-	-	-	3	-	5	-
R15	1	-	-	-	-	-	-	-	-	-	2	2	-	-	4	-	4
R16	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-

Tabela 4.2: Tempos de transmissão. A primeira coluna diz qual o tempo gasto internamente em cada roteador. As demais colunas dão os tempos para envio entre dois roteadores ligados (cada valor indica o tempo de envio do roteador linha para o roteador coluna), sabendo-se que o tempo de A para B pode ser diferente do tempo de B para A. Exemplo: Uma mensagem de R6 para R5 gasta 3 u.t., enquanto de R5 para R6 gasta apenas 2 u.t.

tempo que não temos. Para os curiosos, o algoritmo de Dijkstra aparece em diversos livros, como [4, 5, 7, 8].

Do ponto de vista do curso de computação, o estudo de fluxo em redes é feito basicamente nas disciplinas Teoria dos Grafos e Métodos de Otimização I. O vínculo dessas disciplinas com álgebra linear está no fato de que o conteúdo de ambas depende fortemente dos conceitos de espaços vetoriais e de manipulação de matrizes, vistos nessa disciplina. Apesar do nome, a disciplina de Redes de Computadores aborda o tema apenas no tratamento do problema de roteamento de pacotes, quando apresenta o algoritmo de Dijkstra e vários outros, apontando também que, em geral, os métodos de roteamento de fato utilizados usam abordagens mais heurísticas[1], evitando um consumo elevado de processamento. Nesse caso, os métodos exatos servem para a validação (ou comparação) da heurística proposta.

4.2 Modelagem de sistemas computacionais

Outra área em que temos a aplicação de técnicas advindas da álgebra linear é a modelagem de sistemas, em especial de sistemas computacionais. Antes de falarmos de sistemas computacionais é preciso definir o que é modelagem, a qual pode ser vista como a atividade de fazer "cópias" de um sistema real em algo que possa ser processado computacional ou algebricamente. Essa cópia recebe o nome de modelo, sendo que os modelos podem assumir um formato algébrico (podendo ou não ser resolvido no computador) ou funcional (quando precisaria ser simulado). A criação de modelos que sejam fiéis ao sistema é, na prática, um dos grandes objetivos de quem trabalha com o desenvolvimento da computação.

Indo agora ao problema específico de sistemas computacionais, podemos defini-los como sendo um conjunto de equipamentos e técnicas utilizados para o processamento de dados e informações. Atualmente, sua composição mais básica é dada pela estrutura da figura 4.3, atribuída a John von Neumann e constitui-se de uma unidade central de processamento (UCP), uma unidade de memória e unidades para a entrada e a saída (E/S) dos

[1]Uma heurística é uma regra (conhecimento comum) não exata que resolve um problema de forma relativamente eficiente. Heurísticas são largamente usadas nas áreas de planejamento, sistemas especialistas e reconhecimento de padrões, por exemplo

dados/informações.

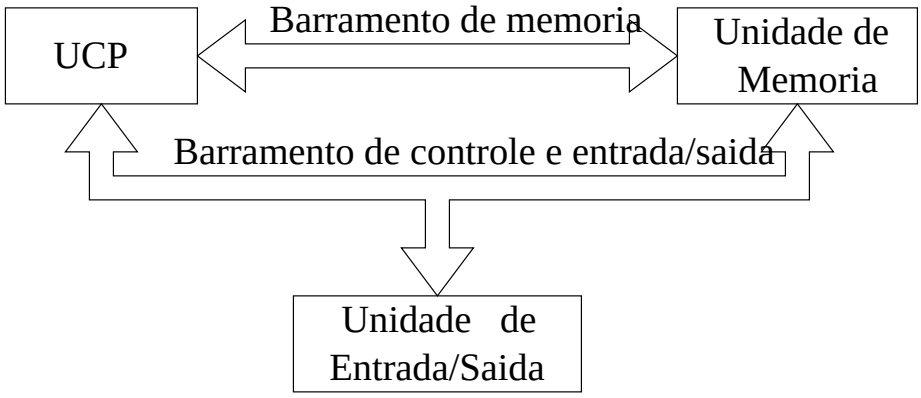

Figura 4.3: Composição básica de um sistema computacional

A UCP gerencia todo o sistema e efetua as operações necessárias de forma a transformar um conjunto de dados/informações, submetidos aos dispositivos de entrada, em outro conjunto de dados/informações úteis ao usuário, que os espera junto aos dispositivos de saída do sistema. Para tanto, a UCP é constituída por dois módulos básicos: a unidade de controle (que faz o gerenciamento do funcionamento de todo o sistema) e a unidade lógica e aritmética (ULA). Mais conhecida por sua sigla, a ULA é a responsável pelas transformações produzidas sobre os dados e informações apresentados ao sistema computacional. A UCP comunica-se com as demais unidades do sistema por meio dos barramentos de memória e de controle e E/S.

A unidade (ou sistema) de memória é composta por dispositivos para o armazenamento eletrônico de dados e informações. Usualmente existem níveis hierárquicos de memória definidos segundo a capacidade de armazenamento e a velocidade de acesso de cada dispositivo. Os dispositivos mais comuns são componentes de estado sólido, discos e fitas magnéticos e, mais recentemente, discos eletro-ópticos. Os componentes de estado sólido compreendem as chamadas memórias de acesso aleatório (RAM - Random Access Memories) e as memórias apenas de leitura (ROM - Read Only Memories). Os discos e fitas magnéticos são os dispositivos de E/S mais antigos, usados desde os primórdios dos computadores. Os discos eletro-ópticos, muito populares atualmente (CD-ROMs e DVDs), têm a mesma aplicação dos dispositivos eletromagnéticos, usando porém a luz para a transmissão de informações. A velocidade de acesso é maior nos componentes de estado sólido, enquanto a capacidade de armazenamento é maior nos dispositivos magnéticos e ópticos.

A unidade de E/S consiste de dispositivos para a troca de informações com o ambiente externo ao computador tais como teclados e mice (para entrada de dados), monitores de vídeo (atualmente utilizados tanto para entrada quanto para a saída de dados), impressoras e plotters (para a saída de dados), bem como de seus respectivos circuitos de interfaceamento.

Para que um sistema computacional execute as transformações que desejamos em certos conjuntos de dados ou de informações, precisamos dizer-lhe o que deve ser feito e como ele deverá executar tais atividades. O conjunto de instruções que indica ao sistema computacional como ele deve se comportar é descrito em um programa de computador. Uma vez escrito e corrigido, o programa pode ser simplesmente armazenado em um disco ou em outro

tipo de memória disponível no sistema ou pode ser submetido para execução pela máquina. Quando um programa está sendo executado pelo sistema computacional ele é chamado de processo, ou seja, um processo é um conjunto de atividades que ocorre dentro de um sistema computacional. Quando sistema computacional pode executar mais de um processo, concorrentemente, dizemos que o sistema é multitarefas ou multiprocessado. Esse é o caso dos sistemas modernos, como UNIX, VMS, NT e MacOS.

Os equipamentos disponíveis ao sistema para a execução dos processos, tais como dispositivos de entrada e saída de dados e informações, são chamados genericamente de recursos do sistema e, normalmente, podem ser compartilhados pelos vários processos que estiverem executando no sistema. Entretanto, para que não haja interferência de um processo no outro, é necessário que o sistema estabeleça certas regras de funcionamento. Por exemplo, se tivermos duas pessoas utilizando editores de texto em um sistema, então cada cópia do editor é um processo distinto do ponto de vista do sistema computacional. Imaginemos que o sistema dispõe de uma única impressora e que os dois usuários terminaram de digitar seus textos e que ambos desejam imprimi-los. Se o sistema não impuser qualquer restrição quanto ao uso da impressora, corremos o risco dos dois processos tentarem imprimir simultaneamente e obterem acesso única impressora. A saída pode ser desastrosa, como vemos na figura 4.4.

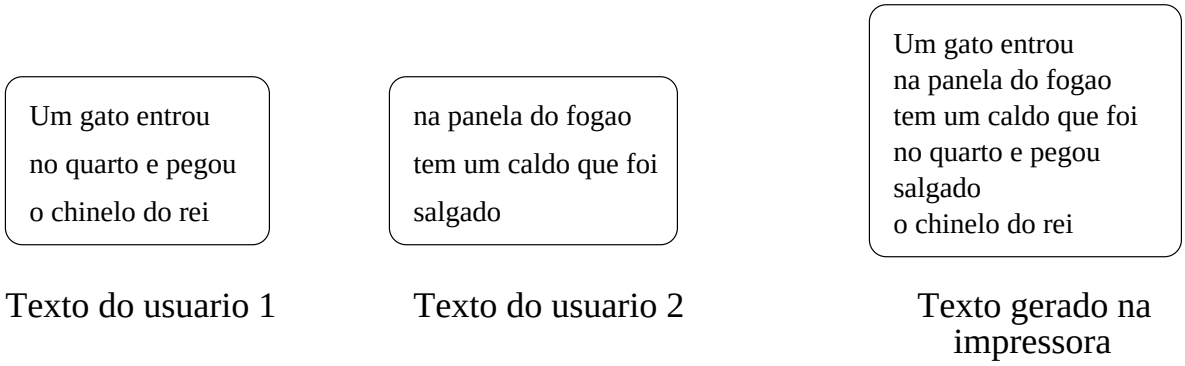

Figura 4.4: Problemas no tratamento de concorrência por recursos

Para contornar este tipo de problema os sistemas computacionais restringem o acesso a seus recursos, de acordo com certas políticas de uso. Em nosso exemplo, uma política seria a concessão da impressora a apenas um dos processos, fazendo com que o outro espere o primeiro terminar sua impressão para, somente após, dar-lhe o acesso ao recurso desejado. Todavia, talvez o processo que ganhou acesso à impressora primeiro seja de um usuário que está imprimindo uma versão de um longo relatório de atividades (500 páginas) a ser entregue dentro de 30 dias e o que ficou esperando seja o de um usuário que precisa imprimir uma solicitação (de meia página) para o reparo na rede elétrica do laboratório a ser entregue dentro de 30 minutos ao setor de manutenção para que o laboratório possa continuar funcionando durante o período noturno. Evidentemente esta é uma situação bastante improvável, mas ilustrativa do tipo de análise que se deve fazer ao planejar um sistema. Deveríamos comprar outra impressora para ser usada somente em casos urgentes? Ou deveríamos fazer com que o usuário pudesse submeter seu processo urgente com precedência sobre os demais? Neste caso, o que aconteceria se o segundo usuário chegasse depois do primeiro começar

a imprimir seu relatório? Ou devemos simplesmente fechar o laboratório durante a noite? Para que possamos estudar tais possibilidades costumamos fazer modelos dos sistemas em desenvolvimento e estudar as diversas alternativas e suas conseqüências antes de construí-los. Este estudo técnico através da modelagem e da simulação dos sistemas é importante tanto do ponto de vista econômico, para que não se desperdice dinheiro com opções de compra erradas, quanto do ponto de vista de eficiência, uma vez que com determinadas opções podemos obter um desempenho do sistema melhor do que com outras opções. O objetivo desta seção é mostrar um exemplo de como a álgebra linear pode auxiliar-nos nestas tarefas.

4.2.1 Modelo de um sistema computacional

Admitamos a existência de um conjunto de processos, os quais compartilham o uso de um conjunto de recursos do sistema. Há dois tipos de processos (dois editores de texto e três programas para a edição de desenhos) e três tipos de recursos compartilhados (um disco rígido, três impressoras laser e dois *plotters*). Chamemos os processos de *proc-p* e *proc-q* e os recursos de *rec-r*, *rec-s* e *rec-t*, respectivamente. Cada processo é cíclico e durante as etapas de seu ciclo necessita de acesso exclusivo a diversos subconjuntos dos recursos disponíveis no sistema. As necessidades dos processos são apresentadas na figura 4.5.

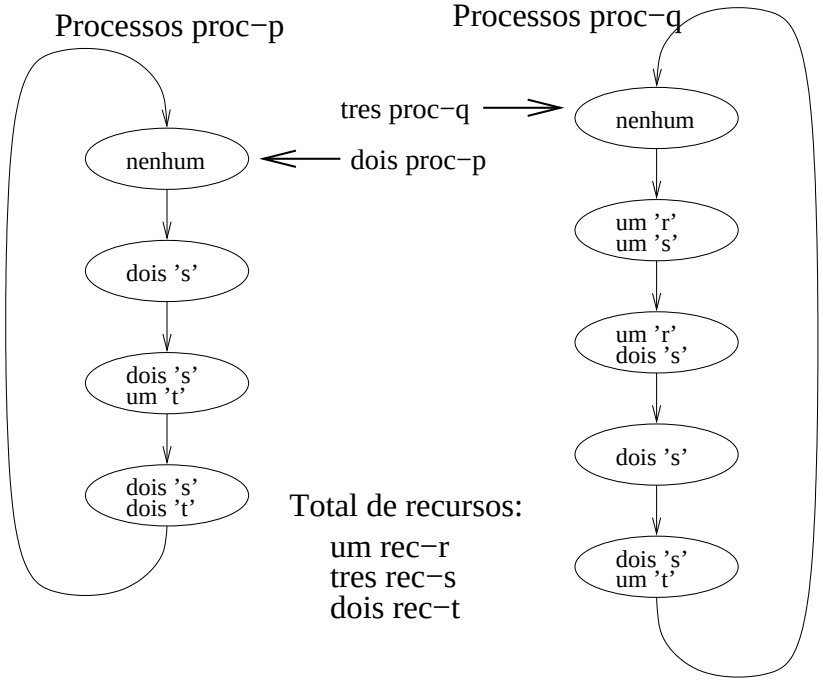

Figura 4.5: Estados dos processos no sistema de alocação de recursos

Podemos ver que os *proc-p* podem assumir quatro estados diferentes. No primeiro estado não necessita de qualquer recurso do sistema. No segundo estado são necessários dois recursos do tipo *rec-s*, e assim por diante. Analogamente, *proc-q* têm cinco estados diferentes e, para cada um desses estados o correspondente subconjunto de recursos necessários está especificado. Ao todo, o sistema contém dois *proc-p*, três *proc-q*, um *rec-r*, três *rec-s* e dois *rec-t*. As duas setas indicam que todos os processos iniciam nos estados no topo do diagrama.

Na figura 4.5 especificamos as demandas dos processos descrevendo os seus estados possíveis. Alternativamente, poderíamos especificar as demandas descrevendo as ações possíveis,

como mostrado na figura 4.6. Neste caso, as duas setas indicam que os processos iniciam sua operação executando uma das ações no topo do diagrama.

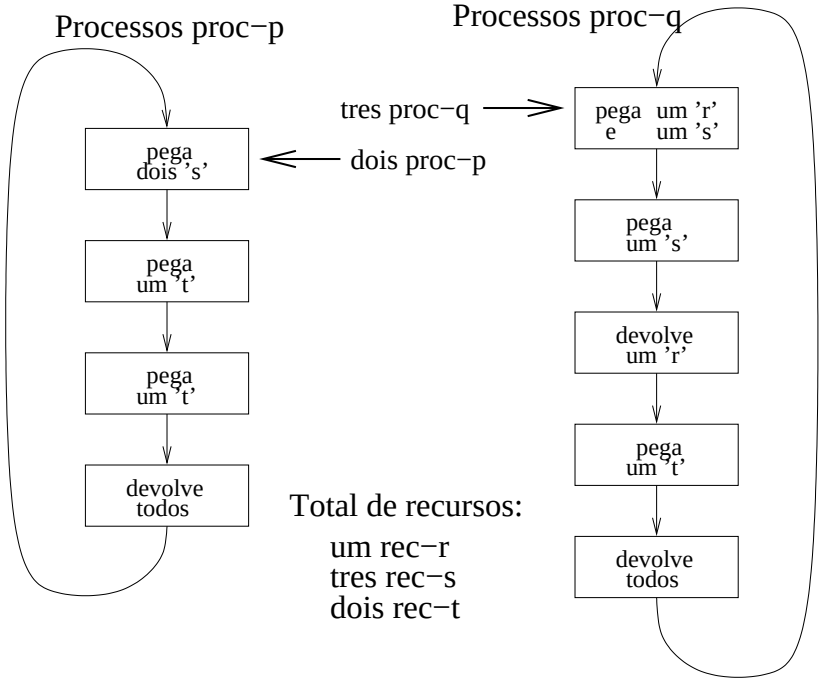

Figura 4.6: Ações dos processos no sistema de alocação de recursos

A especificação da figura 4.5 é orientada a estados porque ela dá uma descrição explícita dos estados possíveis. As ações possíveis, necessárias para a ocorrência das mudanças de estados, são descritas apenas implicitamente (elas podem ser deduzidas dos estados). Em contraste, a especificação da figura 4.6 é orientada a ações porque ela dá uma descrição explícita das ações possíveis. Os estados possíveis - os quais ocorrem entre as ações - são descritos apenas implicitamente (eles podem ser deduzidos das ações). De fato, elas são equivalentes, isto é, as mesmas informações podem ser deduzidas de qualquer uma das duas figuras.

4.2.2 Modelos usando redes de Petri

Embora equivalentes, os diagramas das figuras 4.5 e 4.6 podem ser combinados em um único diagrama. Uma das formas de se fazer isso é através de uma notação conhecida por rede de Petri do tipo lugar-transição. Uma rede de Petri nada mais é do que um grafo (uma primeira visão do conceito de grafos será dada provavelmente na disciplina de Estrutura de Dados) dirigido e bipartido, em que um conjunto de vértices representa os estados possíveis do sistema e o outro conjunto representa as ações aplicáveis ao mesmo (o grafo é bipartido por ter dois conjuntos disjuntos de vértices). Os dois conjuntos são interligados por arestas que têm início e fim, isso é, começam em um vértice e terminam em outro. No caso de grafos bipartidos essas arestas sempre ligam vértices em conjuntos opostos. O grafo da figura 4.7 mostra a rede de Petri equivalente aos diagramas anteriormente apresentados.

Os estados do sistema de alocação de recursos são indicados por meio de elipses, chamadas de lugares. Há doze lugares diferentes. Três dos lugares representam a disponibilidade dos três tipos de recursos do sistema (R, S e T). Quatro dos lugares representam os quatro estados

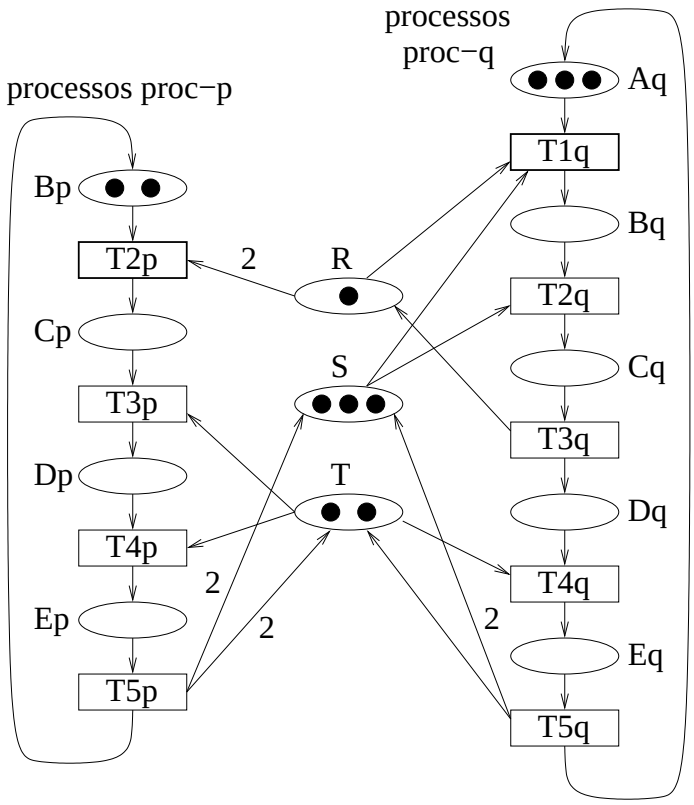

Figura 4.7: Sistema de alocação de recursos descrito por uma rede de Petri.

possíveis dos *proc-p* (Bp, Cp, Dp e Ep), enquanto os cinco lugares restantes representam os estados pelos quais os *proc-q* podem passar (Aq, Bq, Cq, Dq e Eq). Os nomes dos lugares não têm qualquer significado formal, mas são de grande importância prática para a legibilidade do grafo, além de tornarem mais fácil a referência a cada um dos lugares. O uso de nomes significativos para os lugares é análogo ao uso de nomes bem escolhidos para variáveis e procedimentos em programas escritos em linguagens de programação tradicionais, tais como a PASCAL e a C.

Cada lugar pode conter um número variável dinamicamente de pequenos pontos pretos, chamados de marcas. Uma distribuição de marcas nos lugares é dita uma marcação do grafo. Na figura 4.7 temos a distribuição inicial de marcas nos lugares, a qual é chamada de **marcação inicial** e é representada por $\mathbf{M}_0$. As duas marcas em Bp dizem que há dois proc-p e que ambos começam no lugar Bp. Analogamente, há três proc-q e todos iniciam no lugar Aq. As marcas nos lugares R, S e T mostram-nos o número de recursos disponíveis no sistema e seu respectivo tipo.

As ações no sistema de alocação de recursos são representadas por retângulos, os quais são chamados de transições, na linguagem gráfica que estamos utilizando. Aliás, este é o motivo pelo qual o grafo apresentado na figura 4.7 é dito uma rede do tipo lugar-transição, isto é, porque os estados do sistema são representados por lugares e as ações por transições. Neste grafo há nove transições diferentes. Quatro representam as ações possíveis para os proc-p (T2p, T3p, T4p e T5p), enquanto as demais cinco representam as ações dos proc-q (T1q, T2q, T3q, T4q e T5q). Como no caso dos lugares, os nomes das transições não têm significado formal, mas têm importância prática equivalente. Referimo-nos coletivamente

aos lugares e às transições das redes de Petri como nós (uma vez que ambos correspondem aos nós do grafo subjacente).

A rede lugar-transição também contém um conjunto de arcos direcionados, os quais conectam lugares a transições ou vice-versa, mas nunca nós de mesmo tipo entre si. A cada nó deve ser associado um número inteiro positivo, dito expressão do arco. Convencionou-se que quando esta expressão tem valor unitário ela é omitida.

Um nó x é dito de entrada de outro nó y, *sse* existe um arco dirigido de x para y (o termo *sse* é uma forma sincopada para a expressão "se e somente se"). Analogamente, um nó x é dito de saída de outro nó y, *sse* existe um arco dirigido de y para x. Na terminologia de redes de Petri podemos falar em lugares, em transições e em arcos, de entrada e de saída. Por exemplo, a transição T2p tem dois arcos de entrada, conectando-a aos lugares de entrada Bp e S, e um arco de saída, conectando-a ao lugar de saída Cp. Analogamente, o lugar S tem dois arcos de entrada, conectando-o com as transições de entrada T5p e T5q, e tem três arcos de saída, conectando-o com as transições de saída T2p, T1q e T2q.

Até o momento vimos os aspectos sintáticos das redes de Petri lugar-transição, vejamos agora seus aspectos semânticos, ou seja, o comportamento destes grafos. Uma rede lugar-transição pode ser vista como um tabuleiro de um jogo em que as marcas são as peças do jogo e só podem ser posicionadas nos lugares da rede. Cada transição representa um movimento potencial neste jogo, o qual passaremos a denominar de jogo de Petri. Um movimento é possível *sse* cada lugar de entrada de determinada transição contiver pelo menos o número de marcas prescrito no arco de entrada que liga o lugar correspondente à transição em questão. Se isto ocorrer dizemos que a transição está habilitada. Na marcação inicial, mostrada na figura 4.7, a transição T2p está habilitada porque há pelo menos uma marca no lugar Bp e duas em S. A transição T1q também está habilitada porque há pelo menos uma marca em cada um dos lugares Aq, R e S. Todas as outras transições estão desabilitadas porque há menos marcas do que as necessárias em pelo menos um dos seus lugares de entrada. Usamos uma linha mais grossa para indicar que as transições T2p e T1q estão habilitadas.

Quando uma transição está habilitada a movimentação correspondente pode ser executada. Quando isto acontece dizemos que a transição ocorreu, ou que ela foi disparada. O efeito da ocorrência de uma transição é a remoção de marcas dos lugares de entradas e a criação de novas marcas nos lugares de saída da transição. O número de marcas removidas/criadas é especificada pela expressão dos arcos de entrada/saída correspondentes. Por exemplo, a ocorrência da transição T2p transforma a marcação inicial M_0 (figura 4.7) na marcação M1, mostrada na figura 4.8. Formalmente, estas transformações correspondem à destruição de um certo número de marcas dos lugares de entrada da transição (como já dissemos, número este correspondente aos valores das expressões dos respectivos arcos de entrada) e a produção de marcas completamente novas nos lugares de saída (como também já dissemos, estas em número correspondente aos valores das expressões dos respectivos arcos de saída). Portanto, formalmente não há qualquer relação entre as marcas removidas de um lugar de entrada qualquer com aquelas criadas em algum lugar de saída. Apenas nossa interpretação intuitiva do modelo do sistema que estamos representando permite que imaginemos, em certas circunstâncias, que determinadas marcas se comportem como que fluindo através dos lugares da rede.

Analogamente, a ocorrência de T1q transformaria a marcação M_0 na marcação M2. Neste caso: retiramos uma marca do lugar R, outra do lugar S e outra do lugar Aq; criamos uma nova marca no lugar Bq; a transição T1q torna-se desabilitada; e a transição T2q passa à condição de habilitada. O restante do grafo permanece como na figura 4.7. Dizemos que

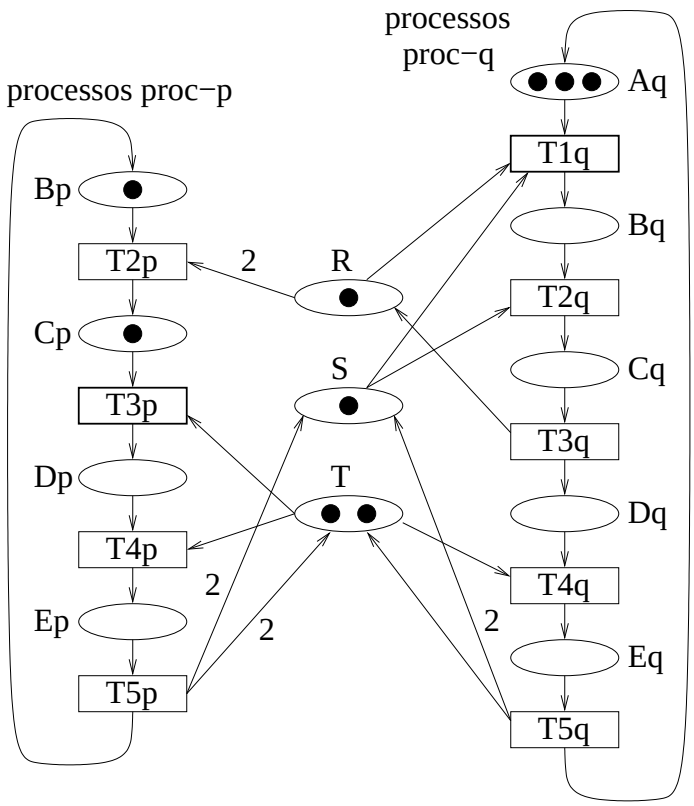

Figura 4.8: Grafo da rede de Petri após a ocorrência de T2p

ambas as marcações, M1 e M2, são diretamente alcançáveis de M_0 - pela ocorrência de T2p e T1q, respectivamente.

Na marcação M1 (figura 4.8) ambas as transições T3p e T1q estão habilitadas, como indicado pelos retângulos com traço mais grosso. A ocorrência de T3p resulta em uma marcação, a qual podemos chamar de M3, diretamente alcançável de M1, mas apenas alcançável de M_0 - pela ocorrência de T2p seguida da ocorrência de T3p.

Vimos que, em M_0, tanto a transição T2p quanto a transição T1q estão habilitadas. Isto significa que cada uma delas pode ocorrer. Além do mais, no lugar S, compartilhado pelas duas transições, há marcas suficientes para que cada uma delas utilize o número de marcas necessários ao seu disparo, sem a necessidade de disputa com a outra. Neste caso, dizemos que as transições T2p e T1q estão habilitadas concorrentemente em M_0. Isto significa que ambas as transições podem ocorrer ao mesmo tempo, ou em paralelo. Também podemos dizer que o passo S1=T2p, T1q está habilitado em M_0. Portanto, duas ou mais transições são habilitadas concorrentemente *sse* forem independentes entre si, isto é, se puderem operar sobre conjuntos disjuntos de marcas. Por exemplo, enquanto as transições T2p e T1q estão habilitadas concorrentemente em M_0, as transições T2p e T2q, apesar de estarem ambas (individualmente) habilitadas em M2, não sendo independentes, não estão habilitadas concorrentemente nesta marcação, pois a ocorrência de uma desabilita a outra, uma vez que devem compartilhar os dois recursos remanescentes no lugar S.

4.2.3 Descrição matricial do modelo do sistema

Toda a discussão feita até aqui parte de uma representação conceitual do sistema. A questão que fica é como podemos extrapolar a análise de um sistema como o representado nas figuras 4.5, 4.6 e 4.7 para qualquer momento de sua execução? A resposta vem da álgebra linear, mais precisamente do tratamento vetorial de matrizes. Esse tratamento, aliás, é a fundamentação teórica que alimenta as redes de Petri. Nos próximos parágrafos apresentaremos como é a representação matricial das redes de Petri e como é que operadores vetoriais apresentam respostas sobre o funcionamento de um sistema modelado através de tais redes.

Podemos representar uma rede de Petri através de duas matrizes, **D-** e **D+**, que representam as funções de entrada e saída dos conjuntos de nós. Cada matriz tem m linhas, uma para cada transição, e n colunas, uma para cada lugar da rede. As matrizes D- e D+ representam, respectivamente, as funções de entrada das transições e as funções de saída das transições. As funções de entrada e saída podem ser definidas como mapeamentos dos arcos ligando os lugares e transições da rede de Petri.

Formalmente, definimos por função de entrada de uma transição t_j o mapeamento desta transição em uma coleção de lugares $I(t_j)$, ditos lugares de entrada da transição, os quais podem ser caracterizados graficamente como aqueles com arcos originários no lugar $p_i \in I(t_j)$ e com término na transição t_j. Da mesma forma, definimos por função de saída de uma transição t_j o mapeamento desta transição em uma coleção de lugares $O(t_j)$, ditos lugares de saída da transição, os quais podem ser caracterizados graficamente como aqueles com arcos originários na transição t_j e com término nos lugares $p_i \in O(t_j)$.

Assim, podemos definir a matriz D- como sendo D-[j,i] = # $(p_i, I(t_j))$ e a matriz D+ por D+[j,i] = # $(p_i, O(t_j))$. Em ambas as expressões o símbolo # representa o valor da expressão do arco ligando p_i com t_j.

Habilitação de uma transição

Como dissemos, a notação matricial permite que sejam verificadas certas condições em um modelo em rede de Petri de um sistema. Tais verificações são possíveis graças a operações sobre com vetores sobre essas matrizes. Em particular, a verificação da habilitação ou não de uma transição pode ser feita através do vetor **e[j]**, unitário de m componentes, dos quais *m-1* são nulos, exceto o j-ésimo elemento. A transição t_j é representada pelo vetor e[j]. Então uma transição t_j estará habilitada em uma marcação μ se $\mu \geq e[j].D-$ e o resultado do disparo da transição t_j na marcação μ é:

$\delta(\mu, t_j) = \mu - e[j].D- + e[j].D+ = \mu + e[j].(-D- + D+) = \mu + e[j].D$, em que definimos a matriz composta D = D+ - D-.

Seqüência de disparos

Uma seqüência de disparos nada mais é do que uma série de transições que são disparadas para simular a execução do sistema. Então, podemos definir uma seqüência σ de disparos das transições $t_1 t_2 \cdots t_k$, como:

$$\begin{array}{rcl} & & \delta(\mu, \sigma) = \delta(\mu, t_1 t_2 \cdots t_k) \\ \delta(\mu, \sigma) & = & \mu + e[j_1].D + e[j_2].D + \cdots + e[j_k].D \\ & = & \mu + (e[j_1] + e[j_2] + \cdots + e[j_k]).D \\ & = & \mu + f(\sigma).D \end{array}$$

Na equação anterior o vetor $f(\sigma) = e[j_1] + e[j_2] + \cdots + e[j_k]$ é chamado de vetor de disparos

da seqüência $t_1 t_2 \cdots t_k$ e seu i-ésimo elemento, $f(\sigma)_i$ corresponde ao número de vezes que a transição t_i dispara na seqüência σ. Portanto, o vetor de disparos é um vetor de números inteiros e não negativos.

Exemplo

Utilizando os conceitos aqui apresentados, o grafo da figura 4.7 pode ser descrito pelas seguintes matrizes de entrada, D-, e de saída, D+, apresentadas na tabela 4.3.

$$D^- = \begin{array}{c} \\ T2p \\ T3p \\ T4p \\ T5p \\ T1q \\ T2q \\ T3q \\ T4q \\ T5q \end{array} \begin{array}{c} \begin{array}{cccccccccccc} Bp & Cp & Dp & Ep & R & S & T & Aq & Bq & Cq & Dq & Eq \end{array} \\ \left[\begin{array}{cccccccccccc} 1 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

$$D^+ = \begin{array}{c} \\ T2p \\ T3p \\ T4p \\ T5p \\ T1q \\ T2q \\ T3q \\ T4q \\ T5q \end{array} \begin{array}{c} \begin{array}{cccccccccccc} Bp & Cp & Dp & Ep & R & S & T & Aq & Bq & Cq & Dq & Eq \end{array} \\ \left[\begin{array}{cccccccccccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}\right] \end{array}$$

Tabela 4.3: Matrizes de entrada, D-, e de saída, D+, para o sistema exemplo

A marcação inicial (M_0) do sistema é descrita por um vetor, obtido a partir da figura 4.7, no qual cada entrada corresponde ao número de processos ou recursos existentes em cada um dos estados locais possíveis ao sistema:

$$M_0 = \left| \begin{array}{cccccccccccc} Bp & Cp & Dp & Ep & R & S & T & Aq & Bq & Cq & Dq & Eq \\ 2 & 0 & 0 & 0 & 1 & 3 & 2 & 3 & 0 & 0 & 0 & 0 \end{array} \right|$$

A combinação das matrizes de entrada e de saída formam a matriz de trabalho, D = D+ - D-, que está representada na tabela 4.4.

$$D = D^+ - D^- = \begin{array}{c} \\ T2p \\ T3p \\ T4p \\ T5p \\ T1q \\ T2q \\ T3q \\ T4q \\ T5q \end{array} \begin{array}{c} \begin{array}{cccccccccccc} Bp & Cp & Dp & Ep & R & S & T & Aq & Bq & Cq & Dq & Eq \end{array} \\ \left[\begin{array}{cccccccccccc} -1 & 1 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 1 & 0 & 0 & 0 & -1 \end{array}\right] \end{array}$$

Tabela 4.4: Matriz D = D+ - D-, para o sistema exemplo

Finalmente, o disparo de cada uma das transições é representado por um vetor unitário

e[j], como definido anteriormente. Este vetor tem todos os elementos nulos, exceto aqueles que representam a transição que deve disparar (a ação que deve ocorrer). Por exemplo, se em determinado instante quisermos disparar uma vez a transição *T1q*, teríamos que descrever esta ação pelo vetor:

$$e[T1q] = \left| \begin{array}{cccccccccc} T2p & T3p & T4p & T5p & T1q & T2q & T3q & T4q & T5q \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right|$$

Além disto, o valor do elemento corresponde ao número de vezes que determinada transição dispara. Por exemplo, se quiséssemos disparar uma seqüência σ incluindo duas vezes a transição T4p, uma vez a transição T2q e três vezes a transição T3q, não necessariamente nesta ordem, o vetor e[σ] usado para representar esta seqüência de disparos no nosso sistema computacional hipotético seria dado por:

$$e[\sigma] = \left| \begin{array}{cccccccccc} T2p & T3p & T4p & T5p & T1q & T2q & T3q & T4q & T5q \\ 0 & 0 & 2 & 0 & 0 & 1 & 3 & 0 & 0 \end{array} \right|$$

Com isto, temos o sistema da figura 4.7 descrito matricialmente. Na seção seguinte usaremos estas matrizes para exemplificar o método de análise do grafo que representa nosso sistema computacional hipotético.

4.2.4 Análise do modelo

O conjunto de matrizes e vetores descritos na seção anterior podem ser usados para diversos tipos de análise sobre o sistema computacional. Inicialmente, vejamos como verificar se a ação T1q pode ser executada a partir do estado inicial do sistema. Para tanto, devemos verificar se $M_0 \geq$ e[T1q] . D-. Utilizando as matrizes mostradas na seção anterior, executamos a multiplicação *e[T1q]* . *D-* e obtemos o vetor [0 0 0 0 1 1 0 1 0 0 0 0]. Comparando cada elemento deste vetor com os respectivos elementos do vetor M_0, também apresentado na seção anterior, chegamos a conclusão que todos os elementos de M_0 são maiores ou iguais aos elementos correspondentes em e[T1q] . D- e que, portanto, a transição T1q está habilitada na marcação inicial.

Isto significa que, no início do funcionamento de nosso sistema computacional hipotético, um dos processos *q* pode executar sua primeira atividade, a qual necessita de um disco rígido (recurso *R*) e de uma impressora (recurso *S*). Significa, também, que ambos os recursos mencionados estão disponíveis quando o sistema começa a funcionar. O resultado desta execução é um novo estado para o sistema, o qual chamaremos de M_1 e que pode ser previsto através da seguinte equação matricial: $M_1 = M_0 +$ e[T1q] . D. Novamente, utilizando as matrizes apresentadas na seção anterior, fazemos as operações necessárias e obtemos o estado resultante:

$$M_1 = \left| \begin{array}{cccccccccccc} Bp & Cp & Dp & Ep & R & S & T & Aq & Bq & Cq & Dq & Eq \\ 2 & 0 & 0 & 0 & 0 & 2 & 2 & 2 & 1 & 0 & 0 & 0 \end{array} \right|$$

Esta marcação nos mostra que temos os dois processos *p* ainda no estado inicial Bp, que o disco e uma das impressoras, inicialmente disponíveis, estão sendo utilizados por um dos processos *q*, o qual passou do estado Aq ao estado Bq, e que os outros dois processos *q* permanecem no estado Aq.

Perguntamos agora se é possível executar dois processos q simultaneamente a partir do estado inicial do sistema? Isto equivale a perguntar se é possível disparar a transição T1q duas vezes seguidas, a partir da marcação M_0. O que significa que devemos determinar se $M_0 \geq f(\sigma).D-$, com $\sigma = T1q.T1q$ e $f(\sigma) = [\ 0\ 0\ 0\ 0\ 2\ 0\ 0\ 0\ 0\ ]$.

Para responder a esta questão, inicialmente multiplicamos $f(\sigma).D-$, o que resulta no vetor [0 0 0 0 2 2 0 2 0 0 0 0]. Comparando este vetor com M_0 vemos que o quinto elemento de M_0 (*el5*) é menor que o elemento correspondente do produto $f(\sigma).D-$, ou seja, $M_0^{el5} = 1 < (f(\sigma).D-)^{el5} = 2$. Portanto, não é possível executar dois processos q, simultaneamente, a partir do estado inicial do sistema, pois, como a quinta posição de M_0 representa o recurso R, ambos os processos precisam acessar o único disco existente no sistema e isto não pode ser feito simultaneamente. Isto indica ao projetista do sistema que, ou ele tolera a execução de um processo q por vez, ou ele deve comprar outro disco para o sistema.

Um segundo tipo de análise que se pode fazer é procurar saber se uma determinada seqüência de ações pode ser executada. Vejamos se a seqüência $\sigma = T2p\ T1q\ T2q$ pode ser executada, a partir do estado inicial do sistema. Desse modo o vetor de disparos fica sendo $f(\sigma) = [\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ ]$, o que torna $f(\sigma).D- = [\ 1\ 0\ 0\ 0\ 1\ 4\ 0\ 1\ 1\ 0\ 0\ 0\ ]$. Comparando com a marcação inicial vemos que $M_0^{el6} = 3 < (f(\sigma).D-)^{el6} = 4$, o que indica que faltam impressoras no nosso sistema para executar a seqüência σ de ações. Também vemos que $M_0^{el9} = 0 < (f(\sigma).D-)^{el9} = 1$ o que indica que a ação T1q deveria ter sido executada, pelo menos uma vez. Na verdade, isto não indica que a seqüência não pode ser executada, já que a atividade T1q faz parte da seqüência. Entretanto, indica que as ações de σ não podem ser executadas simultaneamente.

Uma consideração a ser feita, com relação a estas análises, é que, ao contrário do que vimos até agora, não precisamos analisar as atividades do sistema sempre em relação ao seu estado inicial, M_0. Por exemplo, pode interessar-nos saber se a atividade T4p pode ser executada a partir de um estado M_x dado por $[\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 2\ 1\ 0\ 0\ 0\ ]$. Para tanto, devemos verificar se $M_x \geq e[T4p].D-$. Fazendo o produto de e[T4p] pela matriz D- obtemos o vetor $[\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ ]$. Comparando este vetor com M_x vemos que $M_x \geq e[T4p].D-$. Portanto, a atividade T4p pode ser executada a partir do estado global M_x. O estado resultante da execução desta atividade será dado por $M_{x+1} = M_x + e[T4p].D$. Ao realizarmos essas operações obtemos $M_{x+1} = [\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 2\ 1\ 0\ 0\ 0\ ]$. Quando isto ocorre dizemos que o estado M_x é transformado no estado M_{x+1} pela execução da atividade T4p.

Como exercício, sugerimos que o aluno procure analisar outras alternativas de percorrimento do grafo, executando os procedimentos algébricos necessários e tentando relaciona-los com seu significado no sistema computacional hipotético.

4.3 Considerações finais

Os casos examinados nesse capítulo são apenas dois exemplos de aplicação de álgebra linear em computação. No último exemplo temos uma aplicação de amplo uso em modelagem e verificação de sistemas, principalmente sistemas complexos que envolvam problemas de concorrência por recursos ou necessidade de sincronismo entre agentes computacionais, sendo portanto usada para se fazer computação. Já o primeiro deles, **fluxo em grafos**, envolve mais a aplicação de técnicas de otimização vindas da álgebra linear para a solução de um

problema econômico. Isso significa que, embora o exemplo utilizado fosse computacional, as mesmas técnicas podem ser aplicadas a uma grande gama de outros problemas.

Na prática, o uso de álgebra linear em otimização leva à sua utilização em problemas econômicos, políticos, sociais e científicos sempre que houver necessidade de maximizarmos, ou minimizarmos, os valores ou os efeitos de certos parâmetros para otimizar algum aspecto de determinada atividade. Esta área é genericamente conhecida como programação matemática e tem como uma de suas principais subclasses a programação linear, na qual aplicam-se métodos matriciais para a solução de problemas envolvendo apenas equações lineares e desigualdades. Várias dessas aplicações serão apresentadas ao aluno no decorrer do seu curso de graduação, outras serão encontradas somente durante a vida profissional ou em cursos de pós-graduação. Para que o futuro profissional possa escolher corretamente os métodos mais apropriados a cada aplicação é extremamente importante um bom conhecimento das possibilidades e limitações de cada ferramenta matemática disponível.

Capítulo 5

Matemática discreta

Uma das disciplinas da área de matemática com mais importância dentro do atual currículo é a de Aspectos Formais da Computação, ministrada no segundo semestre do curso e que tem como tema principal o estudo de matemática discreta. No próximo currículo essa disciplina deve ser desmembrada em outras duas, que são Lógica Matemática e Elementos de Álgebra.

Nesse capítulo trataremos do conteúdo de matemática discreta já fazendo a distinção entre lógica e ágebra, o que é previsto para o próximo currículo. Nas próximas páginas iniciaremos nosso estudo de lógica matemática e, ao final do capítulo trabalharemos com a álgebra.

5.1 Construção de programas corretos

Mantendo o formato adotado nos capítulos anteriores, é preciso primeiro identificar um problema da computação cuja resolução ou é facilitada ou apenas é possível com essa parte da matemática. Aqui trabalharemos com a verificação da correção de um dado programa. Os programas que vocês escreveram até o momento são bastante simples, sem nenhum ponto realmente complexo a ser tratado. No capítulo anterior foi examinado o exemplo de um sistema computacional em que se tinha que controlar a disputa por recursos do sistema entre vários processos. Esse tipo de problema é tratado dentro da programação concorrente e, tem como dificuldade inerente o trabalho de se verificar se um conjunto de programas concorrentes está ou não corretamente programado. É exatamente esse problema que trataremos inicialmente aqui.

Para termos um problema para trabalhar, tentem especificar todos os programas que seriam necessários para fazer o controle de um avião de grande porte (seu piloto automático), de forma a que o mesmo pudesse voar mesmo na ausência de um piloto. Façam uma lista de todos os controles (velocidade, altitude, direção, temperatura da cabine, freio aerodinâmico, flaps, etc.) e todos os sensores envolvidos. Considerem que cada controle e sensor terá um programa escrito para a sua manipulação e que todos esses programas terão que se comunicar e trocar informações, de modo a realizar suas tarefas com sucesso. Contabilizem quantos programas teriam e quantos pontos de interação devem existir. Façam isso antes de continuar a leitura desse texto.

Definir quais programas (que chamaremos processos daqui por diante) estão envolvidos no controle do avião e como eles devem se relacionar é uma tarefa trabalhosa porém dependente do grau de detalhes que iremos colocar no controle. Embora trabalhosa, essa atividade é perfeitamente factível e deve gerar modelos bastante precisos do controle. O problema nosso é a implementação computacional desses processos todos de forma a garantir que, independentemente da velocidade de execução dos vários processadores a bordo e da freqüência em que as informações sobre sensores são disponibilizadas e em que as ações sobre os atuadores são realizadas, nosso avião vai voar e chegar ao destino de forma correta e sem sobressaltos.

Imaginem, por exemplo, que um dos processos controla o ângulo do avião em relação à direção pretendida de vôo[1]. Suponham que esse processo recebe informações de dois outros processos, um que calcula o ângulo necessário a partir de dados sobre ventos e outro que determina a diferença entre o ângulo atual e o necessário. A partir dessas informações esse processo determina quanto deve ser corrigida a posição do avião e faz essa operação.

Suponham agora que o processo que determina a diferença entre os ângulos atual e necessário, por alguma razão, deixa de atualizar essa informação por alguns segundos. O processo que faz a atuação sobre a posição do avião vai considerar que deve corrigir essa posição a cada vez que for executado e, fazê-la de fato. Com isso, após algumas correções de posição o avião estaria consideravelmente fora de sua rota, criando, possivelmente, uma situação de grande risco para os passageiros.

Para evitar esse tipo de situação, todos os processos atuando no controle do avião devem ser escritos segundo restrições da programação concorrente. O problema então passa a ser como verificar se uma coleção de algumas dezenas ou mesmo centenas de processos concorrentes estão interagindo de forma correta, ou seja, realizam suas ações apenas quando elas fizerem sentido (no exemplo dado não adianta corrigir a rota se a informação sobre o erro não for atualizada). A questão então é como verificar se os mesmos estão corretos.

A engenharia de software disponibiliza uma série de ferramentas de teste de software, que podem ser facilmente utilizadas nesses casos. Infelizmente, a engenharia de software também afirma que teste de software pode apenas garantir que o programa funciona para os casos testados, nunca afirmando que o mesmo estaria totalmente livre de falhas. Esse procedimento é válido para um grande número de aplicações, na realidade para todas em que não exista um grande risco para a vida humana. Nesses casos o procedimento de garantia de que o processo funciona tem que ser mais afirmativo, chegando na prática a demonstrar matematicamente que os processos estão corretos.

A prova da correção de um processo pode ser feita de duas maneiras. Na primeira delas parte-se dos processos prontos e aplica-se uma série de teoremas e regras de inferência (construídas a partir da lógica matemática), demonstrando que o resultado desejado é obtido sob qualquer circunstância. Na segunda forma aplicam-se as mesmas regras de inferência e teoremas para, determinar qual a seqüência de comandos é necessária para produzir o resultado desejado e como podemos garantir que essa seqüência vai ser sempre atendida. Esse último procedimento faz parte da área da computação que estuda métodos formais de construção de programas corretos.

Como pode ser observado, as duas técnicas usam teoremas e regras de inferência vindos da lógica matemática. Em particular, as modificações necessárias formam aquilo que chamados de **lógica de programação**, que não deve ser confundida nem com lógica de programas (que é a arte de construir algoritmos que funcionem de forma lógica), nem com programação

[1]Por questões de ventos laterais, aviões normalmente voam "de lado" para corrigir o arraste lateral.

lógica (que é um paradigma[2] de programação ao qual pertence a linguagem Prolog). Na próxima seção fazemos uma breve introdução aos mecanismos de lógica de programação e de sua conexão com a lógica matemática.

5.1.1 Lógica de programação

Dentro de nosso curso a lógica de programação é examinada na disciplina Programação Concorrente, quando é mais do que necessária. Sua complexidade impede que façamos aqui um estudo mais detalhado da mesma. Entretanto, uma pequena apresentação, de modo conceitual, é perfeitamente possível e é o que faremos.

Na prática, a lógica de programação pode ser vista como uma série de regras que representam matematicamente as alterações no estado de um processo em execução no sistema. Assim, as regras existentes devem tratar exatamente dos vários tipos de ação a que um processo possa ser submetido. Teríamos então regras para comandos de decisão, de atribuição, de composição de comandos sequenciais, etc. A tabela 5.1 apresenta uma série de regras e axiomas em lógica de programação. Em cada regra aparece sua versão conceitual e sua versão formal, matemática. Nessa tabela (e na seguinte) usa-se os símbolos $\wedge$ para indicar o operador E, $\vee$ para o operador OU e $\neg$ para o operador NÃO.

Axioma/Regra	Forma matemática	Significado funcional
Axioma da atribuição	P_e^x x:= e $\{P\}$	diz que se um processo estava num estado P, ele passa ao estado P, com a variável x valendo e, após a execução da atribuição $x := e$
Regra da consequência	$\frac{P' \Rightarrow P, \{P\}S\{Q\}, Q \Rightarrow Q'}{\{P'\}S\{Q'\}}$	diz que se é possível provar que executando o comando S a partir do estado P chega-se ao estado Q, e que os estados P' e P são parcialmente equivalentes, o mesmo acontecendo com Q e Q', então é verdade que saíndo do estado P' chega-se ao estado Q' ao executar o comando S
Regra de alternativa	$\frac{P \wedge \neg(B_1 \vee \cdots \vee B_n) \quad \{P \wedge B_i\}S_i\{Q\}, 1 \leq ileqn}{\{P\}IF\{Q\}}$	diz que ao executar um comando IF a partir de um estado P, então chega-se ao estado Q se ou o estado Q é equivalente ao P se todos os bloqueios do comando forem falsos (negados) ou se um deles (B_i) for verdadeiro e o comando por ele guardado (S_i) levar a Q a partir do estado P

Tabela 5.1: Exemplos de axiomas e regras de inferência de lógica de programação

A lógica de programação contém bem mais regras e axiomas do que as listadas na tabela 5.1. Aqui apresentamos apenas essas três regras para que se tenha um sabor de como podemos fazer a prova de um programa qualquer. O importante nesse processo é que todas essas regras apenas fazem sentido se forem elaboradas a partir da solidez presente na lógica matemática, com todos os seus teoremas e axiomas. Na próxima seção faremos uma apresentação da lógica matemática e, em sua conclusão, trabalharemos com mais uma aplicação computacional fortemente dependente dessa lógica.

[2]Forma de escrever e executar programas

5.2 O que é lógica?

O ramo da matemática que trabalha com a verificação formal de teoremas é a lógica, que possui hoje em dia diversas variantes, desde a tradicional lógica clássica (ou lógica de primeira ordem) até lógicas não convencionais, como a lógica nebulosa (fuzzy), na qual nem tudo tem um valor absolutamente verdadeiro ou falso.

Dentro da computação atual pode-se dizer que tudo funciona a partir da lógica. Isso porque todo o funcionamento dos circuitos eletrônicos que compõe a máquina funcionam com o princípio da eletrônica binária, em que um circuito apenas pode estar ligado ou desligado, que é facilmente mapeado para os valores verdadeiro e falso da lógica. Em um nível de abstração mais elevado, temos a lógica usada na construção de algoritmos, em que usamos relações do tipo E, OU, NÃO, para combinar condições de variáveis do programa na elaboração de restrições de execução. Todas essas restrições e operadores vêm da formalização matemática da lógica clássica (filosófica), presentes na chamada lógica de primeira ordem.

Na prática, a formalização matemática da lógica começou na metade do século XIX, quando George Boole iniciou seus trabalhos na chamada álgebra booleana, com o objetivo de estruturar matematicamente as linhas de raciocínio da filosofia (o que não é possível até hoje, afinal de contas filosofar não implica necessariamente em usar argumentação rigorosa). Apesar de não atingir seu objetivo, a criação da álgebra booleana permitiu a formalização de todo um arcabouço de axiomas e regras de inferência que formam a lógica de primeira ordem (e o cálculo proposicional, que é o conjunto de seus operadores).

Outra ferramenta muito útil na construção dos operadores lógicos é o diagrama de Venn, que na realidade veio da teoria dos conjuntos (que faz parte da álgebra). Na figura 5.1 voce pode examinar algumas relações de conjuntos apresentadas através de diagramas de Venn. Observe que as relações lógicas são extremamente relacionadas às relações de pertinência da teoria de conjuntos.

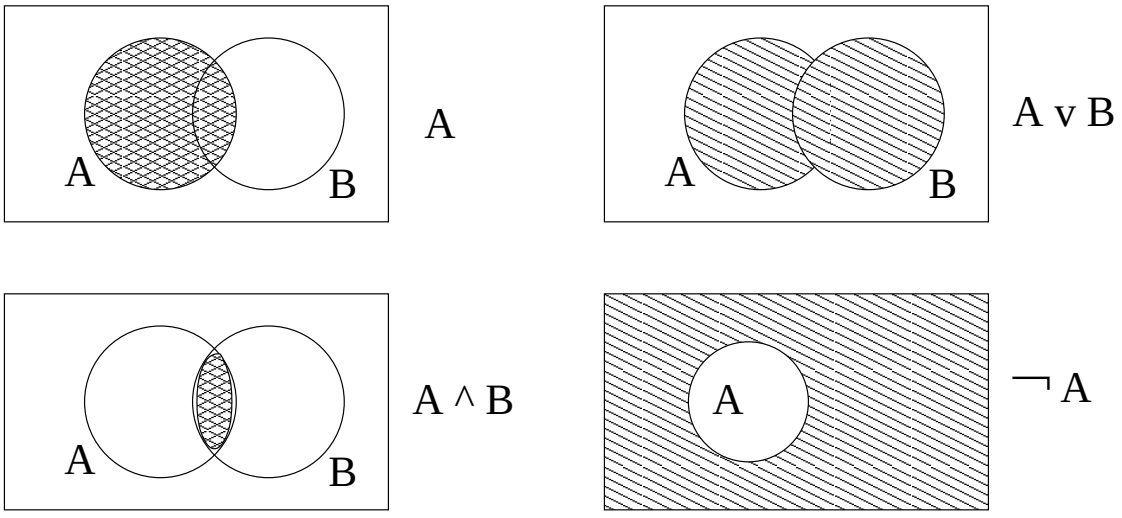

Figura 5.1: Exemplos de diagramas de Venn

Com a lógica de primeira ordem podemos construir provas de validade para uma quantidade imensa de situações em que o número de diferentes possibilidades seja finito. A tabela 5.2 apresenta uma série de leis de equivalência da lógica de primeira ordem e que são, por

sua vez, a base da construção da lógica de programação e de todo o processo de álgebras baseadas em apenas dois valores.

Lei	Forma matemática	Significado funcional
Negação	$P = \neg(\neg P)$	Indica que o contrário do contrário de algo é ele mesmo
Meio excluído	$P \vee \neg P = verdade$	Diz que afirmar que ou isso ou não isso é sempre verdade
Contradição	$P \wedge \neg P = falso$	Aqui, ao contrário, tem-se que algo não pode ser verdade e falso ao mesmo tempo
Implicação	$P \Rightarrow Q = \neg P \vee Q$	Diz que se algo implica em outra coisa, então ou essa outra coisa é verdade ou a primeira é falsa
Igualdade	$(P = Q) = (P \Rightarrow Q) \wedge (Q \Rightarrow P)$	Duas coisas são iguais apenas se uma pode implicar na outra e vice-versa
Simplificação OU	$P \vee P = P$	algo ou algo tem o valor de algo
	$P \vee verdade = verdade$	algo ou verdade é sempre verdade!
	$P \vee falso = P$	
	$P \vee (P \wedge Q) = P$	faça a análise
Simplificação E	$P \wedge P = P$	
	$P \wedge verdade = P$	
	$P \wedge falso = falso$	algo e falso é sempre falso!
	$P \wedge (P \vee Q) = P$	faça a análise
De Morgan	$\neg(P \wedge Q) = \neg P \vee \neg Q$	são leis muito importantes
	$\neg(P \vee Q) = \neg P \wedge \neg Q$	construa diagramas de Venn para elas

Tabela 5.2: Leis de equivalência do cálculo proposicional

Mais uma vez, as leis representadas na tabela 5.2 são apenas um subconjunto das leis de cálculo proposicional. Na realidade, a lógica apresentada até aqui é um subconjunto da lógica matemática, que além da lógica de primeira ordem possui diversas alternativas, sendo que a mais importante para a computação é a chamada lógica nebulosa (*fuzzy*), que é de vital importância em técnicas aplicadas em inteligência artificial (e em suas diversas aplicações). Para entender o que é lógica nebulosa podemos usar um exemplo bastante simples. Na lógica clássica a resposta para a pergunta "Vai chover hoje?" apenas pode ser *sim* ou *não*, não podendo assumir valores distintos desses. Na lógica nebulosa admite-se um conjunto maior de respostas, como "quase certeza que não", "provavelmente sim", etc. O *nebuloso* do nome dessa lógica vem exatamente da possibilidade de respostas imprecisas.

5.2.1 Mais uma aplicação

Saindo do rigor matemático da construção correta de programas através da lógica de programação, uma outra linha de aplicações de lógica em computação está na inteligência artificial. Assim como no restante das áreas de trabalho em computação, aqui também a lógica é necessária em qualquer situação. Vamos examinar, entretanto, um pequeno jogo que, para ser resolvido computacionalmente, precisa de uma quantidade considerável de inferências lógicas.

O nosso exemplo é um antigo jogo, contruído na linguagem LISP, chamado **animal**. Nesse jogo o computador tenta descobrir um animal a partir de uma série de perguntas e respostas trocadas com o usuário. Em geral as perguntas são sobre características do animal, como por exemplo "esse animal tem quatro patas?" ou "ele voa?". Com as perguntas o computador vai construíndo uma seqüência de inferências do tipo "Se tem quatro patas e vive na água e seu filhote mama, então é um hipopótamo".

Nesse jogo nem sempre o computador consegue chegar a uma resposta, sendo que sempre que isso ocorrer ele solicita do usuário uma característica adicional que permitisse identificar o animal. Essa característica é então absorvida e passa a fazer parte do conhecimento armazenado pelo computador, podendo então ser usada numa próxima ocasião.

Dentro desse contexto, o importante é como decidir qual a próxima pergunta a fazer, quando se tem informação suficiente e quando não se tem mais nada para tentar obter essa informação. Dentro de um conceito de sistema um programa que faça isso recebe o nome de sistema especialista, sendo usualmente construído a partir de três módulos básicos: motor de inferências, memória de trabalho e conjunto de produções.

O motor de inferências (MI) é que toma todas as decisões sobre que ações realizar a cada instante. No exemplo do programa *animal*, o MI decidiria o que perguntar e se já tinha uma resposta ou não para o problema. O conjunto de produções (CP) armazena todos os conhecimentos obtidos pelo sistema, normalmente armazenados em regras do tipo SE-ENTÃO. Em nosso exemplo armazenaria coisas do tipo *"Se mamífero então pergunte se é carnívoro"*. Já a memória de trabalho (MT) armazena as informações sobre o problema atual, ou seja, de todo o conjunto de produções vai para a MT apenas aquilo que se conhecer do problema. A figura 5.2 mostra as relações existentes entre esses módulos.

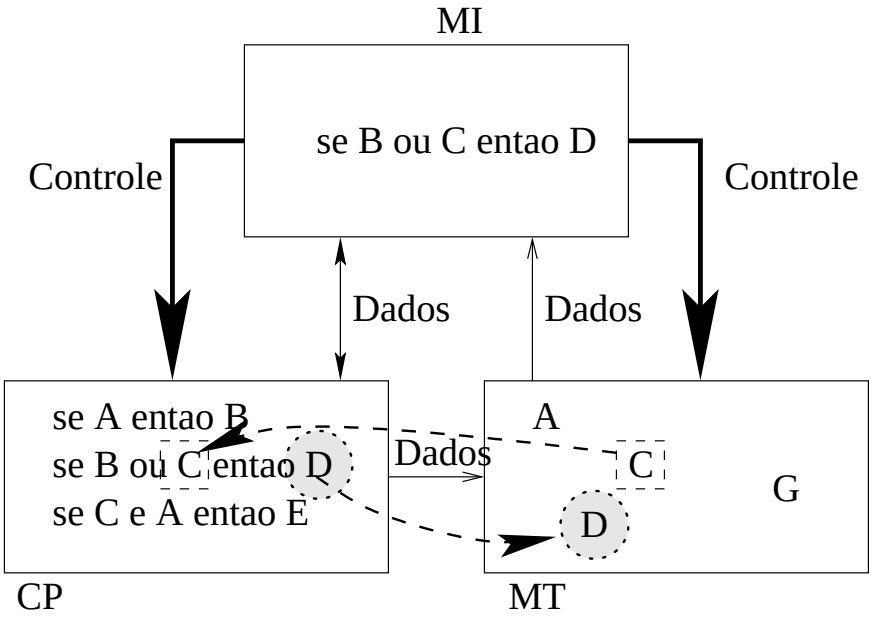

Figura 5.2: Diagrama básico de um sistema especialista

Por essa figura é possível entender que a peça de informação **D** pode ser acrescida à memória de trabalho (e portanto ao processo de resolução do problema) porque existe uma regra de inferência que diz que se as peças de informação **B** ou **C** existirem (forem verdade),

então é possível afirmar que **D** também existe (ou é verdade). Como na memória de trabalho existe a informação de que **C** é verdade, então podemos inferir a regra como sendo verdadeira e inserir **D** na MT. Todo esse processo é fruto da lógica de primeira ordem. Aos interessados é possível fazer um estudo mais completo sobre as várias formas de se armazenar e tratar conhecimento dentro da disciplina Sistemas Inteligentes.

Para fechar essa seção precisamos apenas lembrar algo colocado há algumas páginas, quando se mencionou a forte relação existente entre lógica e álgebra de conjuntos. Na prática essa relação é que torna a álgebra, nosso próximo tópico, tão importante dentro da computação.

5.3 Uma aplicação de álgebra

A álgebra é a parte da matemática que cuida de técnicas de relação entre conjuntos e funções. Dentro da álgebra é possível determinar se uma dada função é ou não aplicável a um dado conjunto de valores e, em caso de sua aplicação, quais as transformações que essa função causa no conjunto.

Para começarmos a entender o uso de álgebra em computação vamos à uma área ainda não abordada nesse texto, que é a de banco de dados. Muitos dos que entram para a computação têm o entendimento de que banco de dados é apenas construir aplicações em que um conjunto de informações que tenham algum tipo de relação serão armazenadas em uma base e que algumas consultas serão feitas para extrair informações dessa base. Isso, entretanto, está muito longe da verdade quando se fala em grandes bases de dados, com centenas de milhares de informações formando seu conteúdo.

Nas grandes bases de dados (e para alguns tipos específicos de dados) é preciso ter não apenas técnicas eficientes de consulta e inserção de dados, como também técnicas de mineração (ou prospecção) de dados que permitam selecionar informações por critérios de proximidade (ou semelhança). Na realidade isso equivale a ter que decidir, por exemplo, qual o grau de semelhança entre as chaves de busca *rato*, *gato* e *galo*. Como decidir o que é semelhante a uma dada chave depende da definição de uma métrica que relacione as duas informações. Essa métrica deve estabeler níveis de similaridade que possam definir a partir de que ponto podemos dizer que esses dados pertenceriam à mesma categoria.

Nosso problema passa a ser, portanto, como definir a tal métrica. Para não complicar muito nosso exemplo vamos trabalhar apenas com chaves textuais[3], ou seja, chaves como as definidas no parágrafo anterior. Dito isso, que métrica nos daria o grau de diferença entre as chaves lá listadas? Essa métrica serviria para diferenciar outras chaves? Qual a resposta dessa métrica para as chaves *rato* e *rata*? E para *gala* e *galo*?

Desses exemplos é fácil perceber que métricas simples podem ser facilmente distorcidas, afinal de contas rata e rato podem ser considerados um casal, o que não é verdade para galo e gala. Desse modo é preciso que, qualquer que seja a métrica básica, se estabeleça um critério de contextualização da chave, evitando que erros grosseiros possam perturbar o resultado final.

A definição de quais métricas e quais relações de contextualização devem ser aplicadas sobre uma base de dados depende, formalmente, de relações algébricas. Infelizmente nosso espaço é pequeno para melhor descrever o significado de relações algébricas, grupos, anéis

[3]Na verdade esse tipo de trabalho se torna muito mais útil e interessante quando tratamos, por exemplo, de chaves de imagens, quando teríamos que decidir quais imagens seriam semelhantes.

e outras construções topológicas que servem de base para a construção de mecanismos de consulta em bases de dados que sejam rápidos, eficientes e precisos (sempre com uma certa probabilidade de erro, é claro).

Um outro aspecto algébrico em banco de dados, na verdade muito mais intenso, é o uso de álgebra relacional (teoria dos conjuntos) na formalização e construção de toda a teoria de bancos de dados relacionais (e também suas extensões). Para entender melhor isso precisamos antes voltar aos conceitos básicos de união e intersecção de conjuntos, além das propriedades sobre conjuntos vazios, unicidade de elementos, etc.

Formalizar uma consulta em um banco de dados, quando construído com rigor, pode ser entendido como o processo de extração de elementos pertencentes a um dado conjunto ou à união ou intersecção de dois ou mais conjuntos. A álgebra relacional atua nesse momento através da utilização dos quantificadores existenciais e universais ($\exists$ e $\forall$). Detalhes na implementação de procedimentos de consulta e inserção de informações em uma base de dados são baseados, portanto, em conceitos como *apenas uma cópia de cada elemento pode existir no conjunto*, *se existir uma cópia outra não pode ser inserida*, e assim por diante. Embora não faça parte desse contexto, existem álgebras mais complexas que permitem que um dado conjunto tenha várias cópias de um elemento (chamados superconjuntos), o que traria complicações para o campo de BD relacional. Isso, entretanto, é campo para novos trabalhos.

5.4 Considerações finais

Os conceitos básicos de computação estão intimamente relacionados à aplicação de lógica e de álgebra. Na prática, tudo o que fazemos com o computador apenas é possível de se justificar com o auxílio dessas duas ferramentas da matemática. Os algoritmos que especificamos, as máquinas que contruímos, enfim, tudo o que se relaciona à execução coordenada de tarefas em ambiente eletrônico, podem ser entendidos como manipulações lógicas e algébricas sobre variáveis de entrada do sistema, levando à produção de resultados (variáveis) de saída.

Nesse capítulo vimos como a álgebra e a lógica são usadas, de modo central, em algumas das aplicações de computação e, de modo implícito, em toda a computação. Apresentamos também, pela primeira vez, uma aplicação de matemática em banco de dados. Na prática banco de dados é única e exclusivamente uma sucessão de aplicações de lógica, teoria dos conjuntos e relações algébricas. A visão simplista dessa área como não necessitando de matemática apenas é possível para sistemas de pequeno porte, que poderiam ser criados por qualquer pré-adolescente com o mínimo de conhecimento em Windows. Como não é esse o nosso caso, então para nós a área de banco de dados precisa ter uma boa formalização matemática, a qual vem essencialmente da matemática discreta.

Capítulo 6

Computação Científica

Nos capítulos iniciais deste curso examinamos algumas aplicações de disciplinas teóricas da matemática, procurando demonstrar a importância de um aprendizado sólido dos conceitos nelas apresentados para o futuro profissional em ciência da computação. Entretanto, em diversos momentos fizemos menção ao fato de que aquelas técnicas eram modificadas para serem possíveis de execução no computador[1]. A sistematização de técnicas de resolução de problemas matemáticos, de modo a torná-las computacionalmente factíveis, é conhecida como computação científica.

Dentro da computação científica existem várias matérias e/ou disciplinas, tais como Cálculo Numérico, Métodos de Otimização I e II, Probabilidade e Estatística, Fundamentos em Computação Científica, Software Numérico, Métodos de Elementos Finitos, etc. Nem todas são obrigatórias a todos os alunos, mas todas têm sua importância e sua aplicabilidade. A matemática examinada em cada uma dessas disciplinas têm, por si só, um forte componente computacional, que pode ser mais ou menos enfatizado durante a disciplina, mas que aparece, sem sombra de dúvida, como uma forte linha de aplicação de computação.

Algumas das disciplinas listadas no último parágrafo já foram abordadas em capítulos anteriores, na qualidade de usuárias da base fornecida em outras disciplinas da matemática. Neste capítulo nos concentraremos apenas na definição das principais classes de problemas que recebem tratamento numérico.

6.1 Métodos numéricos

Os chamados métodos numéricos podem ser divididos em cinco grandes grupos: zeros de funções, sistemas de equações, ajuste de curvas, integração e resolução de equações diferenciais. Em cada um desses grupos existem dezenas de aplicações diferentes, algumas delas em computação. Em comum a todos os métodos existe a preocupação com o desenvolvimento de técnicas algoritmicas, que possam ser executadas de modo lógico e mecânico. É essa preocupação que permite o uso de computadores para a resolução de problemas que possam ser formulados numericamente.

Do ponto de vista formal, que será devidamente estudado na disciplina de **Cálculo Numérico** no terceiro semestre, a criação de um método numérico deve obedecer condições de modo que o método possa:

[1]Na realidade, existe uma área da própria computação (a computabilidade) que examina se um dado problema pode ou não ser resolvido computacionalmente e qual a complexidade dessa tarefa).

- Ser computacionalmente tratável;
- Possuir um algoritmo mecânico;
- Estabelecer limites de erro admissíveis;
- Estabelecer critérios de convergência para um resultado.

Esse conjunto de restrições faz com que os métodos tenham uma evolução contínua, adaptando-se ao potencial de cálculo das máquinas utilizadas. Um exemplo típico disso é busca por economia de memória dos primeiros métodos, que hoje não precisam se preocupar tanto com isso.

Analisando agora cada restrição em separado, tem-se que a primeira delas diz respeito ao tempo que o computador levará para apresentar uma resposta. Quando o problema precisa de um tempo de computação superior ao limite da paciência humana (modificada pela necessidade do resultado, é claro) diz-se que o mesmo não é computacionalmente tratável. É óbvio que os limites de tratamento para cada problema são alterados dinamicamente pelo avanço da tecnologia. Exemplos típicos são a atual capacidade de armazenamento em memória e a velocidade de processamento presentes nos computadores de hoje quando comparados com máquinas de pouco mais do que cinco anos atrás.

Já a necessidade de um algoritmo mecânico para a resolução do problema está ligada ao modelo de processamento usado nos computadores. Como se sabe, embora tenham uma grande capacidade de manipulação de dados os computadores não são tão bons em raciocínio lógico[1]. Isso torna necessário a geração de algoritmos que possam ser seguidos pelo computador.

Todo método numérico apresenta erros em seus resultados. Esses erros são de dois tipos: aproximação pelo computador e aproximação pelo método. O primeiro caso é inevitável pois como o computador representa números em código binário, existe consequentemente um erro de representação desses números (tente por exemplo representar o número 1.1 em binário) e um erro natural de arredondamento dos operandos dentro de um certo número de casas decimais. A segunda forma de erro é introduzida pelo método numérico, sendo portanto perfeitamente quantificável e deve ser fornecida por quem estiver especificando o método.

Por fim, o último critério trata do estudo sobre a convergência do método. Deve-se entender convergência como a tendência do método em aproximar-se ou não da solução do problema. Existem métodos que sempre convergem para o resultado e existem outros que nem sempre o fazem. A escolha por qual método aplicar deve atentar para o critério de convergência e para a velocidade de convergência dos vários métodos, adequando-as para o problema em que se está trabalhando.

Vamos então a um rápido exame sobre os cinco grupos de métodos numéricos listados há pouco.

6.2 Zeros de funções

Os métodos deste grupo se preocupam em determinar os zeros de funções, ou seja, os valores de x para os quais $f(x)$ é igual a zero. Tais técnicas são muito importantes em aplicações como

[1] Mesmo o *Deep Blue*, que derrotou o enxadrista Gary Kasparov, se trata de um exercício de processamento paralelo e não de inteligência artificial. Seu modo de operar consiste basicamente em examinar o maior número possível de alternativas dentro de um critério MINIMAX.

Método	Converg.	Estabil.	Critério básico
Bisecção	lenta	estável	divisão do intervalo em duas partes e busca por valores de f(x) com sinais opostos
Newton-Raphson	rápida	instável	precisa de uma solução inicial (chute) e a partir dela calcula a derivada da função naquele ponto e determina um novo ponto na intersecção da derivada com o eixo x
Secante	rápida	instável	aproxima a derivada de Newton-Raphson por uma reta secante à função, calculada usando dois chutes iniciais.

Tabela 6.1: Métodos para determinação de zeros de funções.

descobrir a partir de que momento um certo investimento passará a ser lucrativo, distância máxima entre postes de iluminação e transmissão de energia ou mesmo a quantidade de memória RAM numa máquina para se atingir uma determinada relação custo-desempenho.

Na disciplina de **cálculo numérico** serão examinados com detalhes vários métodos numéricos para a determinação de zeros de funções. Aqui faremos apenas um breve resumo desses métodos, como pode ser visto na Tabela 6.1. Nessa tabelas são colocadas informações sobre a velocidade de convergência e sobre a estabilidade (se produz a resposta sempre ou não) desses métodos.

6.3 Resolução de sistemas de equações

Neste grupo ficam os métodos que tratam de sistemas de equações lineares, os quais podem receber um tratamento formal do ponto de vista matemático dentro de álgebra linear. Os métodos desta classe são aplicados em problemas como os examinados no capítulo sobre álgebra linear, ou seja, em métodos de otimização, análise de malhas em circuitos elétricos, redes de Petri, etc. A Tabela 6.2 apresenta alguns dos principais métodos desse grupo. Nessa tabela aparecem informações sobre a estabilidade dos métodos e sobre o tamanho do problema a ser tratado com o algoritmo.

Método	Tamanho	Estabil.	Critério básico
Eliminação de Gauss	pequeno	estável	Anula os elementos da diagonal inferior da matriz e calcula a solução por substituição a partir da última equação.
Gauss-Jordan	pequeno	estável	Transforma a matriz em uma identidade. A solução fica automaticamente determinada.
Gauss-Seidel	grande	instável	Calcula a solução através de aproximações sucessivas a partir de chutes iniciais.
Jacobi	grande	instável	Tem funcionamento parecido com o Método de Gauss-Seidel, apenas refazendo as aproximações mais lentamente.

Tabela 6.2: Métodos para resolução de sistemas de equações.

6.4 Ajustes de curvas

O nosso terceiro grupo de métodos procura determinar funções contínuas para problemas em que existam apenas alguns pontos determinados. Isso é bastante útil em computação gráfica, por exemplo, quando se pode determinar qual é a superfície mais suave que passa por um conjunto de pontos de um desenho num sistema CAD. A Tabela 6.3 mostra alguns dos métodos para ajuste de curvas existentes. Vale observar que em computação gráfica os mais usados são *spline* e *bezier*. Como nesses métodos o importante é determinar o quanto uma curva ficaria fora da curva ideal, então nessa tabela apresentamos também a dimensão do erro em cada método.

Método	Erro	Critério básico
Interpolação polinomial	grande	aproxima a função dos pontos através da solução de um sistema de equações.
Newton	baixo	Gera a função através de cálculos de diferenças divididas a partir dos pontos iniciais.
Spline	baixo	Imita a interpolação polinomial, porém calculando apenas intervalos pequenos da curva, que são depois convenientemente ajustados.
Bezier	baixo	faz a aproximação de curvas usando aproximações por segmentos de reta calculados repetidamente.

Tabela 6.3: Métodos para ajustes de curvas.

6.5 Integração

Os métodos mais usados para integração de funções se baseiam na característica geométrica de que a integral de uma função corresponde à área entre a função e o eixo de integração. Desse modo, são propostos vários métodos para calcular essa área a partir de polígonos internos a curva da função. Em computação, a aplicação desses métodos de integração ocorre em modelos de análise de desempenho que usem métodos estatísticos para simulação ou transformação espacial, tais como equalização de histogramas ou resolução de transformadas de Fourier no processamento de sinais. A Tabela 6.4 apresenta dois métodos de integração bastante utilizados, incluindo informações sobre o grau de erro na aplicação do método..

Método	Erro	Critério básico
Trapézio	médio	aproxima a integral através da soma das áreas de trapézios.
Simpson	baixo	aproxima a integral através da soma das áreas de polígonos de base retangular e topo formado por pequenas parabolas.

Tabela 6.4: Métodos para integração de funções.

6.6 Resolução de equações diferenciais

As aplicações dos métodos de resolução de equações diferenciais em computação são exatamente as mesmas já apresentadas quando as equações diferenciais foram apresentadas no capítulo 3. Nesse momento é difícil descrever os me'todos que tratam desse problema, assim apenas indicaremos aqui quais são esses métodos, sem uma análise de como eles operam. Devemos lembrar, entretanto, que esses métodos serão estudados com a devida profundidade na disciplina "Cálculo Numérico", em conjunto com o estudo das demais classes de métodos apresentadas nas seções anteriores.

Os métodos mais usados nessa classe estão agrupados em dois conjuntos de métodos, um englobando os **métodos de passo múltiplo** e outro incluíndo os **métodos de Runge-Kutta**. Em todos os casos existem várias ordens de aplicação, segundo a quantidade de informações necessárias para a sua aplicação, sendo que os métodos de ordem maior são mais precisos porém implicam em custos computacionais também maiores.

6.7 Considerações finais

A principal conclusão a ser tirada nesse capítulo é que tudo o que foi examinado nos capítulos anteriores apenas pode ser executado no computador através da aplicação de métodos numéricos especificados para transformar o modelo matemático analítico em um modelo algoritmico. Assim, do ponto de vista do cientista de computação é fundamental que tais métodos sejam compreendidos em toda sua magnitude, principalmente naquilo que se refere ao esforço computacional exigido e à precisão numérica oferecida por eles.

Além disso, é importante perceber que o bom entendimento dos métodos aqui descritos somente é possível se existir também um bom entendimento das técnicas analíticas envolvidas, isto é, apenas é possível entender o funcionamento de métodos de integração quando se sabe o significado físico e operacional de uma integral, por exemplo. Com isso podemos fechar o ciclo de definições sobre a importância da matemática dentro da computação.

Finalmente, vários métodos de computação científica não foram abordados nesse capítulo ou por já termos examinado algo sobre eles nos capítulos anteriores ou por que os mesmos são extensões, para aplicações específicas, de métodos aqui examinados. A sua ausência nesse momento não deve influenciar o impacto da necessidade de um bom conhecimento de matemática para um bom exercício da computação.

Referências Bibliográficas

[1] ACM/IEEE-CS, "Computing Curricula", IEEE Press, 2002. (disponível em *www.computer.org/education/cc2001/final/index.htm*)

[2] Sociedade Brasileira de Computação, "Currículo de Referência para cursos de graduação em computação - CR99". (disponível em *www.sbc.org.br/educacao/cr99.pdf*)

[3] Manacero Jr., A., e outros; "A flexible curriculum for computer science undergraduate major", em *Proceedings of 31st Frontiers In Education Conference - FIE2001*, Reno, 2001. (disponível em *www.dcce.ibilce.unesp.br/~aleardo/csc.pdf*)

[4] McHugh, J.A.; "Algorithmic graph theory", Prentice Hall, 1990

[5] Roberts, E.S.; "Programming Abstractions in C: a second course in computer science", Addison Wesley, 1997

[6] Shalgi, R.; "Queue Simulator", disponível em *http://www.math.tau.ac.il/ rshalgi/final/*, com último acesso feito em 27/01/2003.

[7] Tanenbaum, A.S.; "Computer Networks, 3rd edition", Prentice Hall, 1996

[8] Tenenbaum, A.M., Langsam, Y. e Augenstein, M.J.; "Data Structures Using C", Prentice Hall, 1990.