Filas de Prioridade e Heaps

Aleardo Manacero Jr.

Filas de prioridade

- O TAD fila, examinado no início do curso, apresenta como característica principal o fato de seus mecanismos de inserção e remoção atuarem apenas em suas extremidades

- O problema com filas é que não podemos diferenciar, para tratamento privilegiado, elementos que porventura estejam nelas armazenados

Filas de prioridade

- Uma forma de tratar de modo diferente os diferentes elementos armazenados numa fila surge com a criação de filas de prioridade

- Numa fila de prioridade atribui-se um valor de priodade a cada elemento e as operações de inserção e remoção passam a ser feitas em função dessa prioridade

Tipos de fila de prioridade

- Embora tenham sido criadas a partir do tipo fila, as filas de prioridade podem ser implementadas a partir de duas estruturas básicas:
  - listas (da definição original de filas)
  - árvores (chamadas de *heaps*)

Operadores em filas de prioridade

- Assim como no TAD fila, os operadores em filas de prioridade se resumem na inserção e remoção de elementos

- No caso de filas de prioridade implementadas na forma de lista, esses operadores dependem de como a lista é organizada internamente

Operadores em listas

- Dependem da organização interna da lista, ou mais precisamente, dela estar ou não ordenada

- Se a lista está ordenada, a remoção ocorre sempre no começo da fila e a inserção em qualquer ponto

- Caso contrário a inserção ocorre sempre no final da fila e a remoção em qualquer ponto

Operadores em árvores

- São análogos aos operadores de inserção e remoção em árvores AVL

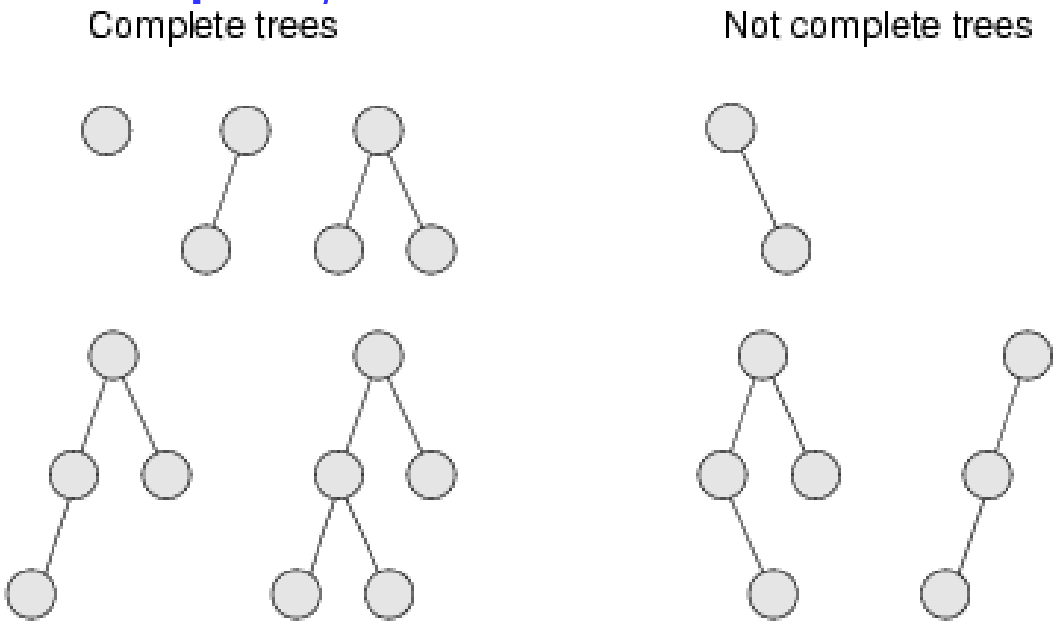
- No caso de heaps a propriedade a ser mantida não se resume ao balanceamento da árvore, mas também às seguintes definições:

  - ser uma árvore completa

  - manter uma propriedade de ordem

Árvore completa

- Uma árvore completa é definida como sendo uma árvore cheia, em que elementos faltantes aparecem apenas no nível mais baixo e, sempre, na subárvore da direita

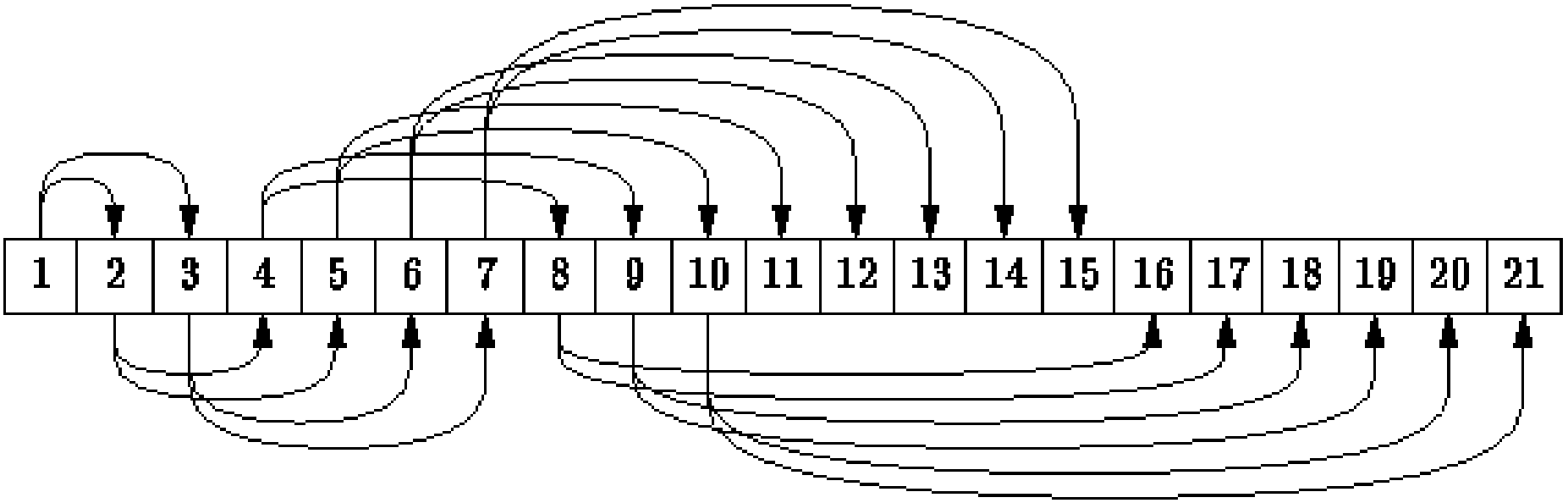


Árvore completa

- Podem ser implementadas como vetores de tamanho proporcional ao da profundidade da árvore

- Assim, o elemento da posição i do vetor tem seus filhos nas posições 2i e 2i+1, assim como seu pai está na posição i/2

Árvore completa



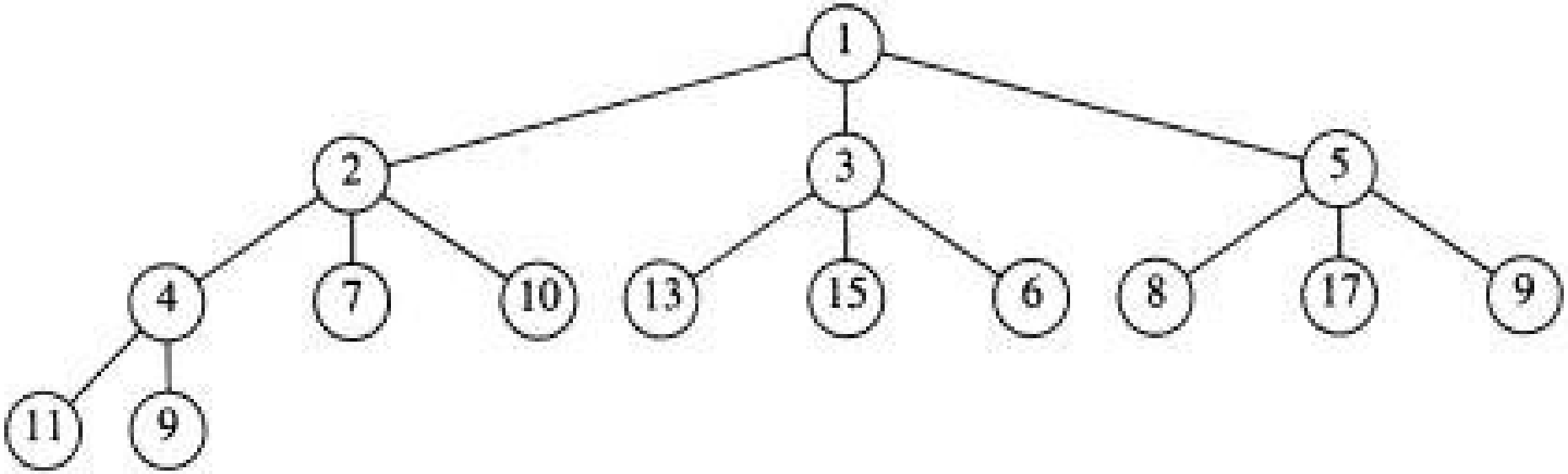
Propriedade de ordem

- Para manter o heap ordenado e facilitar a localição do elemento de maior prioridade, o mesmo é colocado no nó raiz

- Com isso, a prioridade de um nó X qualquer será, sempre, maior ou igual a de seus dependentes

K-heap ou D-heap

- É um heap em que a prioridade de cada nó tem no máximo k (ou d) filhos



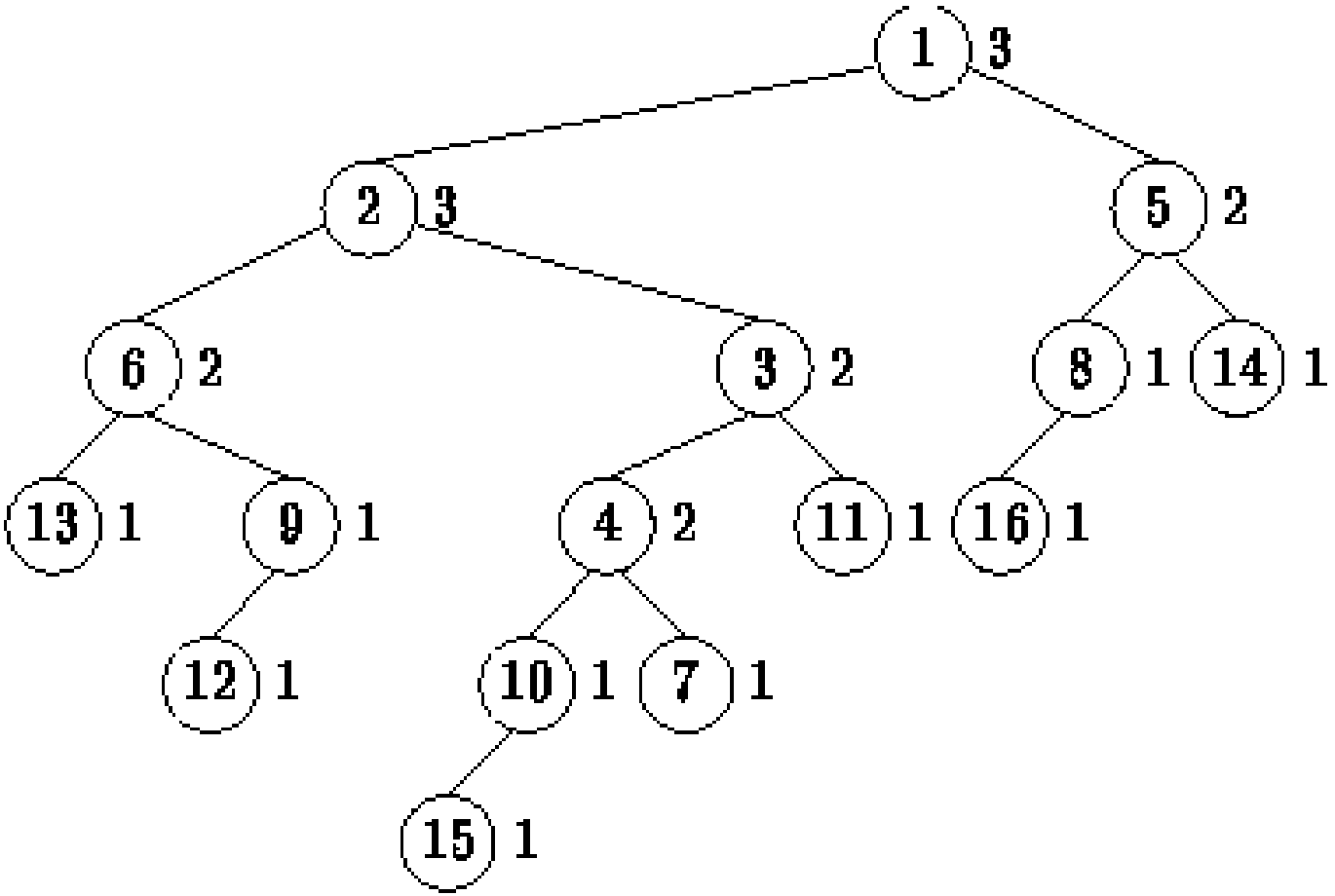
Leftist heaps

- São heaps não balanceados

- Numa leftist heap, existe a tendência da sub-árvore esquerda ser mais profunda que a sub-árvore direita

- Sua estrutura facilita a operação de *merge* entre dois heaps

Leftist heaps

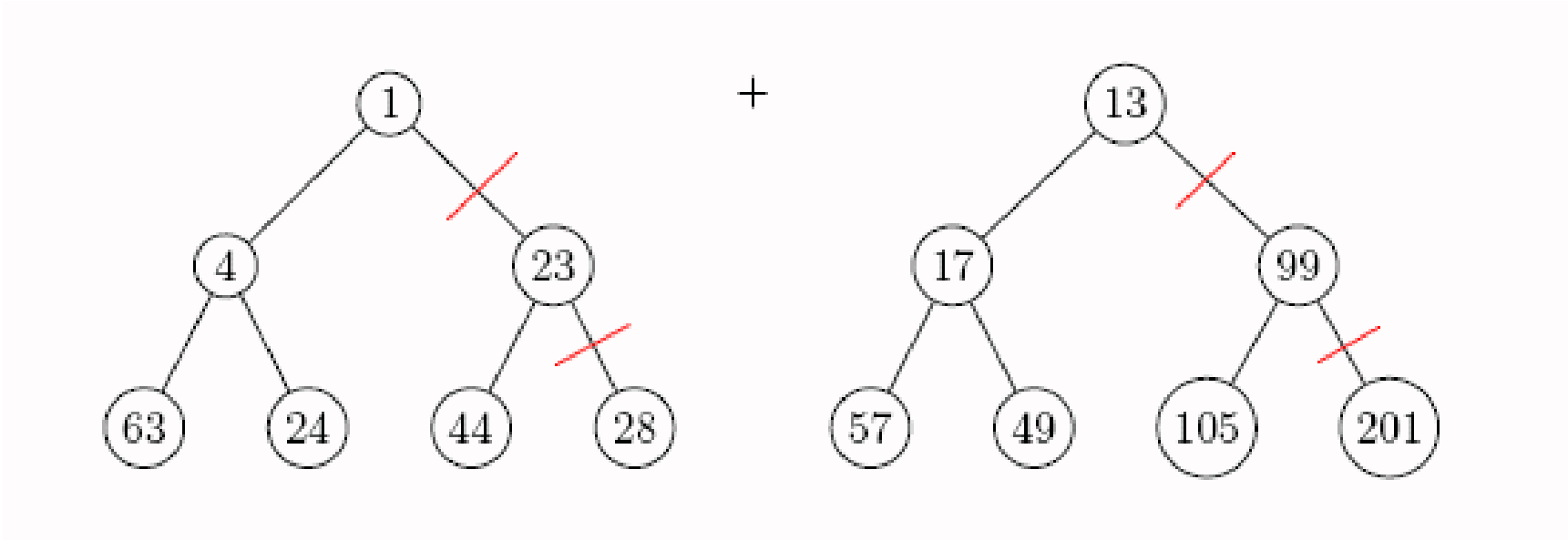


Skew heaps

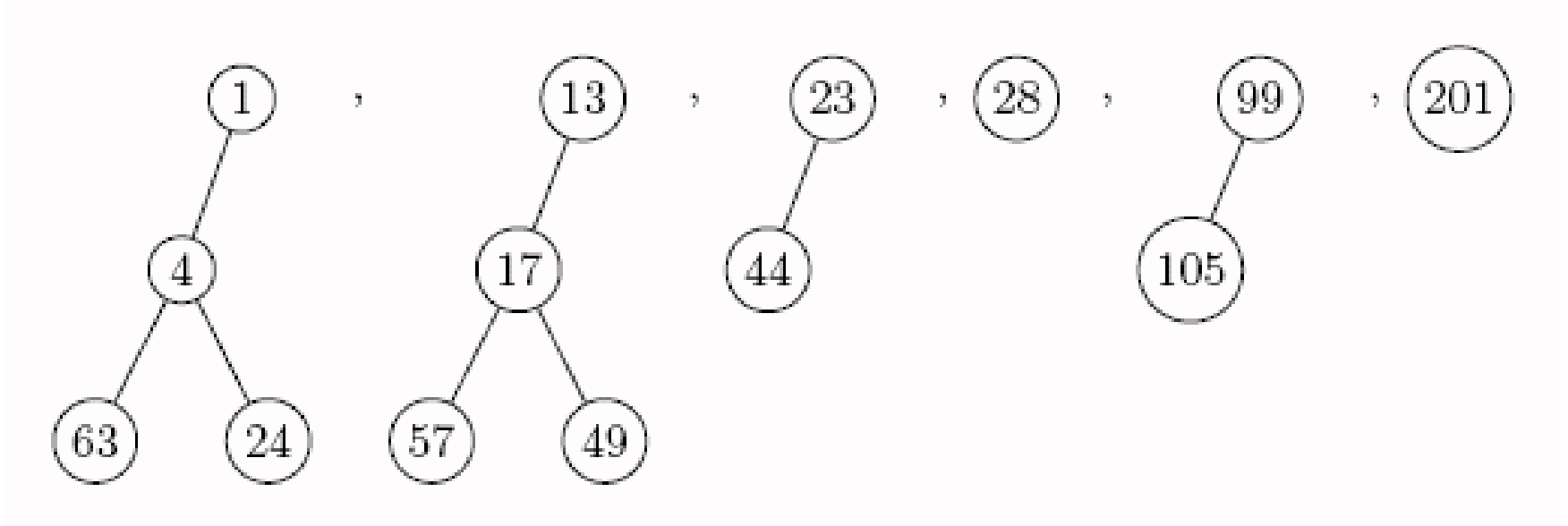
- São uma forma autoajustável de leftist heaps

- Neles o procedimento de merge é feito através de operações de partição das árvores em sub-árvores esquerdas e sua posterior junção

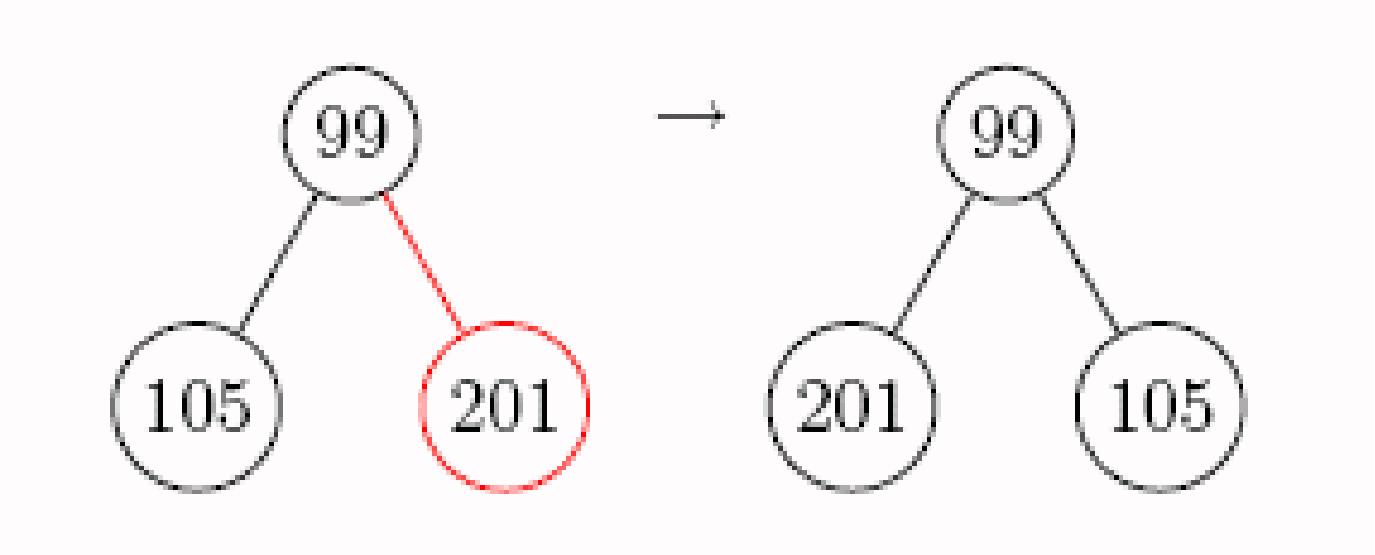
Skew heaps



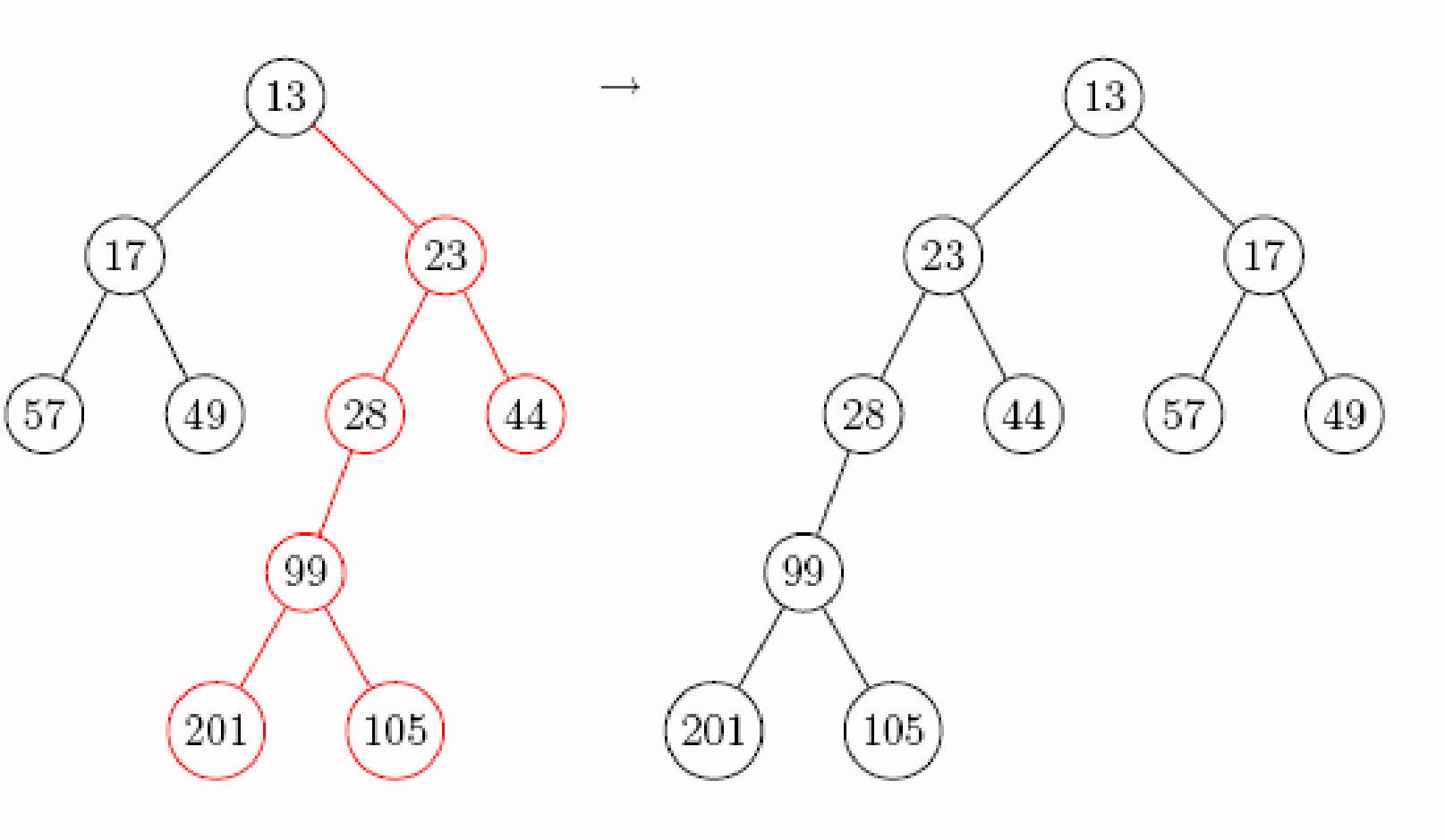
Skew heaps



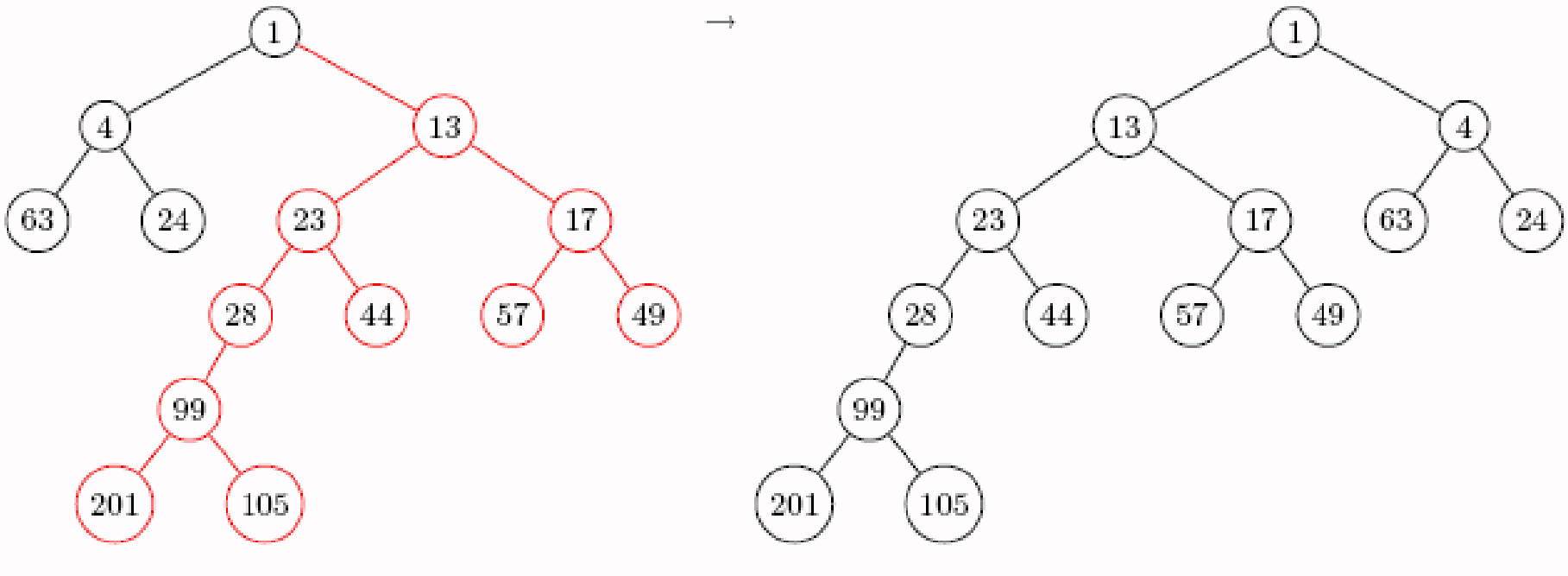
Skew heaps



Skew heaps



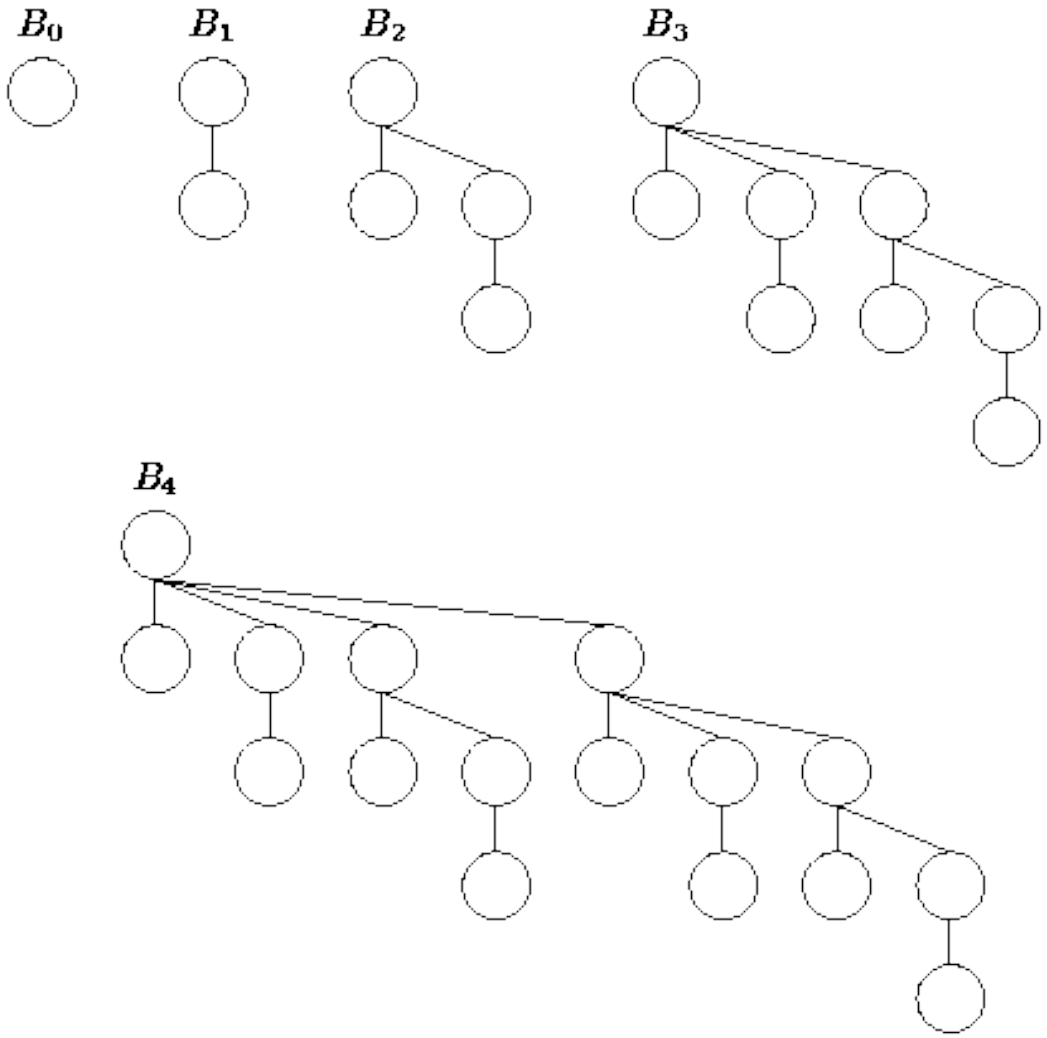
Skew heaps


1
4
13
63
24
23
17
28
44
57
49
99
201
105
→
1
13
4
23
17
63
24
28
44
57
49
99
201
105

Árvores binomiais

- São florestas de árvores heap em que cada árvore tem uma quantidade de elementos na profundidade d determinada pelo coeficiente binomial (k,d)

Árvores binomiais



Árvores binomiais