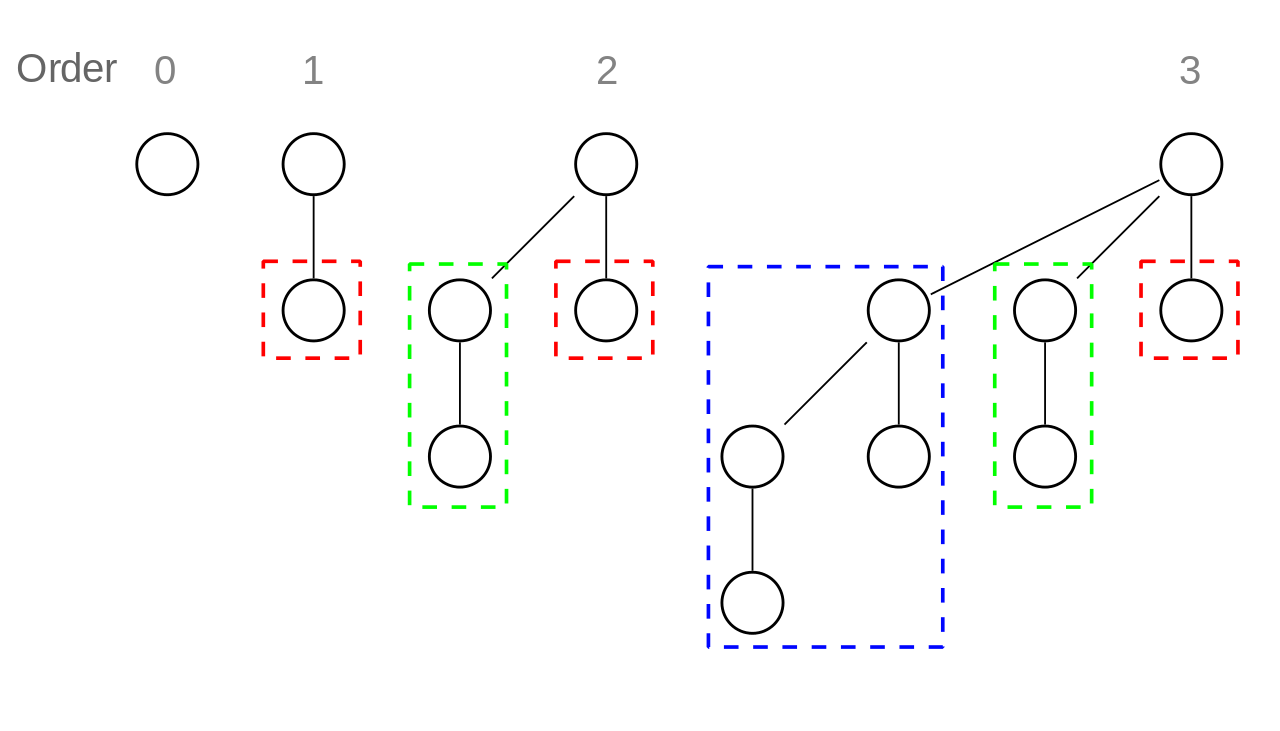
**Binomial Heap (1978, J W J Williams)**

Complexidade:

|  |  |
| --- | --- |
| **Algoritmo** | **Custo** |
| Find-min | O(log n); O(1) |
| Delete-min | O(log n) |
| Insert | O(1) |
| Decrease-key | O(log n) |
| Merge | O(log n) |



É similar a uma binary heap mas também suporta a mesclagem rápida de duas heaps.

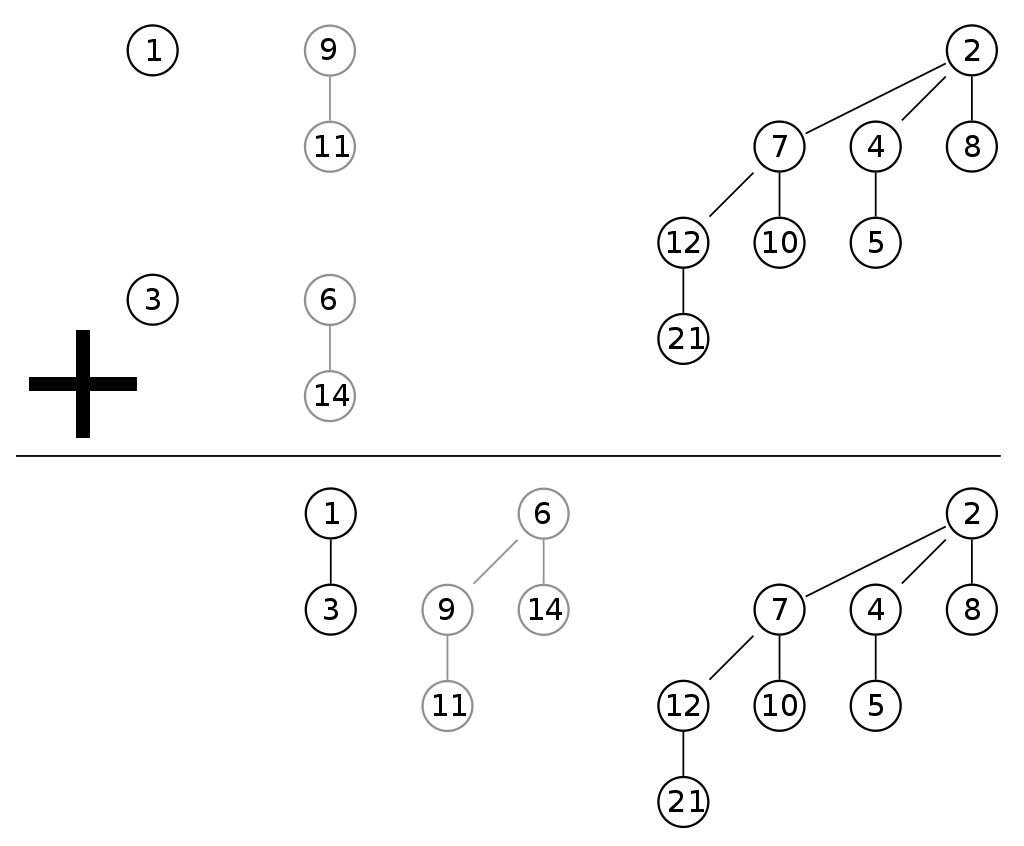
É implementada como um conjunto de binomial trees, que são definidas recursivamente como:

* Uma binomial tree de ordem 0 é um único nó;
* Uma binomial tree de ordem k tem um nó raiz cujos descendentes são raízes de binomial trees de ordem k-1, k-2, ..., 2, 1, nesta ordem.

Uma binomial tree de ordem K tem 2K nós e altura K.

Uma binomial tree de ordem K pode ser construída a partir de duas de ordem K-1 definindo-se uma delas como descendente da outra.

O nome vem da sua forma: uma binomial tree de ordem n tem nós na profundidade d.



**A estrutura de uma binomial heap**

Uma binomial heap é implementada como um conjunto de binomial trees que satisfaz as propriedades de binomial heaps:

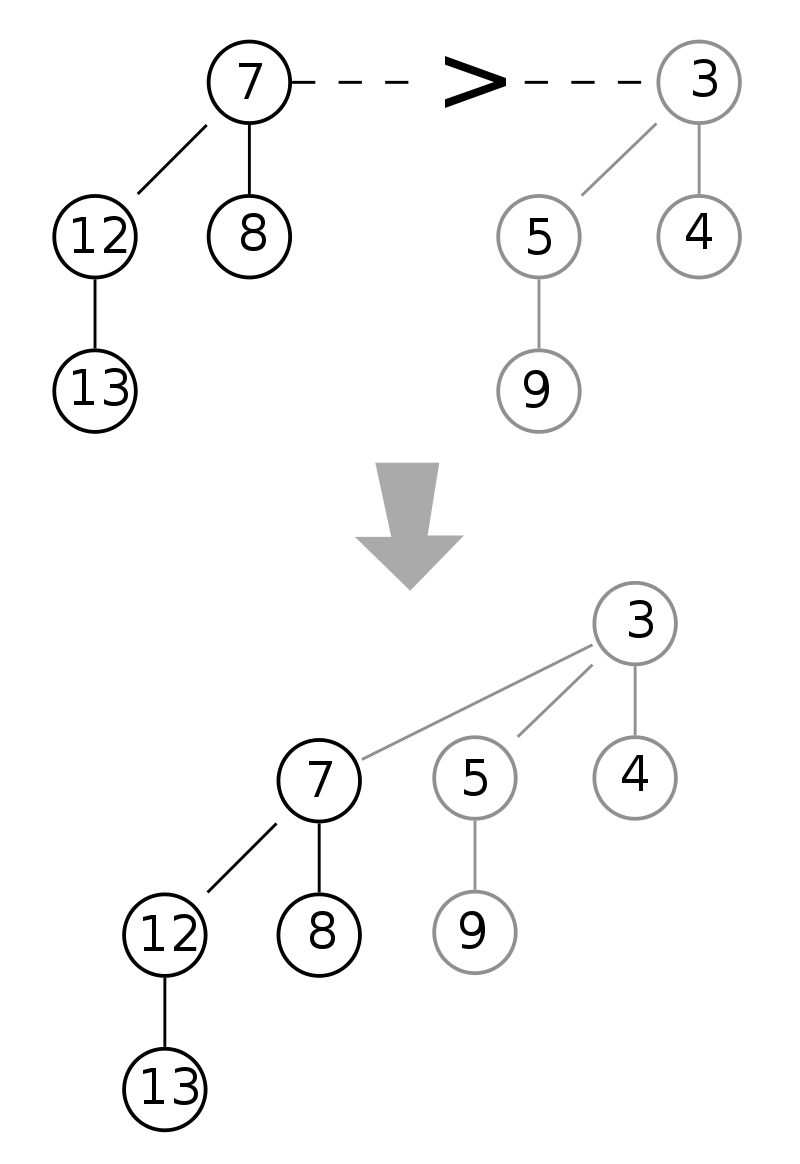
* Cada binomial tree em uma heap obedece a propriedade de min-heap: A chave de um nó é menor que a chave de seus descendentes.
* Só pode haver ou 1 ou 0 binomial trees de cada ordem, incluindo a ordem 0. (eg. Não podem haver duas binomial trees de ordem 3 em uma única heap.)

A primeira propriedade garante que a raiz da binomial tree tenha a menor chave em toda a árvore, o que se aplica a toda a heap.

A segunda propriedade implica que uma binomial heap com n nós consiste de no máximo 1 + log2 n binomial trees. O número e ordem dessas árvores são determinados unicamente pelo número de nós: cada binomial tree corresponde aos bits 1 na representação binária de n. Por exemplo, 13 é 1101 em binário, 2³ + 2² + 20, assim, uma binomial heap com 13 nós consistirá de 3 binomial trees de ordens 3, 2 e 0.

**Implementação**

Como nenhuma operação requer acesso randômico aos nós da raiz, as raízes podem ser armazenadas em uma linked list, ordenada pela ordem das binomial trees em ordem crescente.



**Merge**

A mesclagem de duas árvores é feita entre duas árvores de mesma ordem em uma mesma binomial heap. A raiz das duas é comparada e aquela com a menor raiz passará a ser a árvore principal e aquela com raiz maior passará a ser uma subárvore dela.

MergeTree(TreeA, TreeB)

If TreeA.root < TreeB.root

TreeA.AddSubtree(TreeB)

Else

TreeB.AddSubtree(TreeA)

A mesclagem de duas heaps pode ser usada como subrotina na maioria das outras operações. As listas de raízes de ambas as heaps são percorridas simultaneamente.

Se apenas uma das heaps contiver uma árvore de ordem j, esta árvore é movida para a heap mesclada. Se ambas as heaps contém uma árvore de ordem j, as duas árvores são mescladas em uma árvore de ordem j+1. Essa operação pode fazer com que seja necessário mesclar a árvore recém-criada com outra árvore de ordem j+1 futuramente. Cada árvore tem ordem de no máximo log n, sendo assim, o tempo de execução é O(log n).

MergeHeap(HeapA, HeapB)

While not(HeapA.end() and HeapB.end())

Tree = MergeTree(HeapA.CurrentTree(), HeapB.CurrentTree())

If not Heap.CurrentTree().Empty()

Tree = MergeTree(Tree, Heap.Current(Tree())

Heap.AddTree(Tree)

Heap.Next(); HeapA.Next(); HeapB.Next()

**Insert**

A inserção de um elemento em uma heap pode ser feita criando-se uma nova heap com este elemento e então mesclando ela à heap principal. A inserção toma, então, tempo O(log n). Mas uma série de inserções consecutivas tem tempo amortizado de O(1).

**Find Minimum**

Deve-se percorrer a heap checando cada raiz, como há log n raízes, esta operação leva tempo O(log n).

Pode-se manter um ponteiro para a raiz de valor mínimo para se reduzir o tempo de busca para O(1).

**Delete Minimum**

Primeiro se encontra esse elemento, então se remove ele de sua binomial tree e obtém-se uma lista das suas subárvores. Em seguida deve-se transformar esta lista em uma nova binomial heap. O último passo é mesclar a nova heap a heap original. Esta operação toma tempo O(log n).

DeleteMin(Heap)

Min = Heap.Trees.First()

For Each Current in Heap.Trees()

If Current.Root < Min.Root then Min = Current

For each Tree in Min.SubTrees()

Temp.AddTree(Tree)

Heap.RemoveTree(Min)

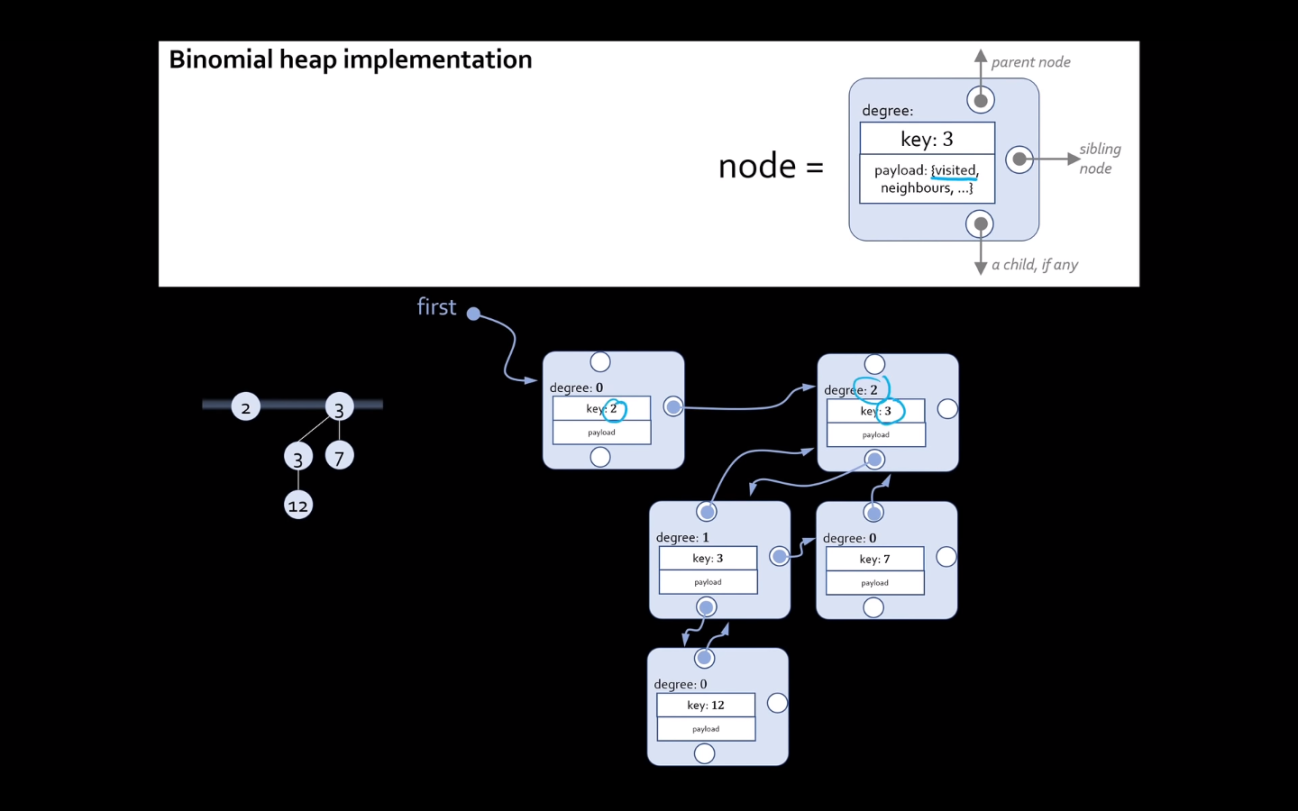
Merge(Heap, Temp)

**Decrease Key**

Depois de diminuir a chave de um elemento, caso ele seja menor que seu ascendente, troque os dois de lugar. Repita esse processo até a árvore estar correta novamente. Essa operação leva tempo O(log n).

**Delete**

Diminua a chave do elemento para -∞ e então remova o menor elemento da heap.



Representação de uma Binomial Heap