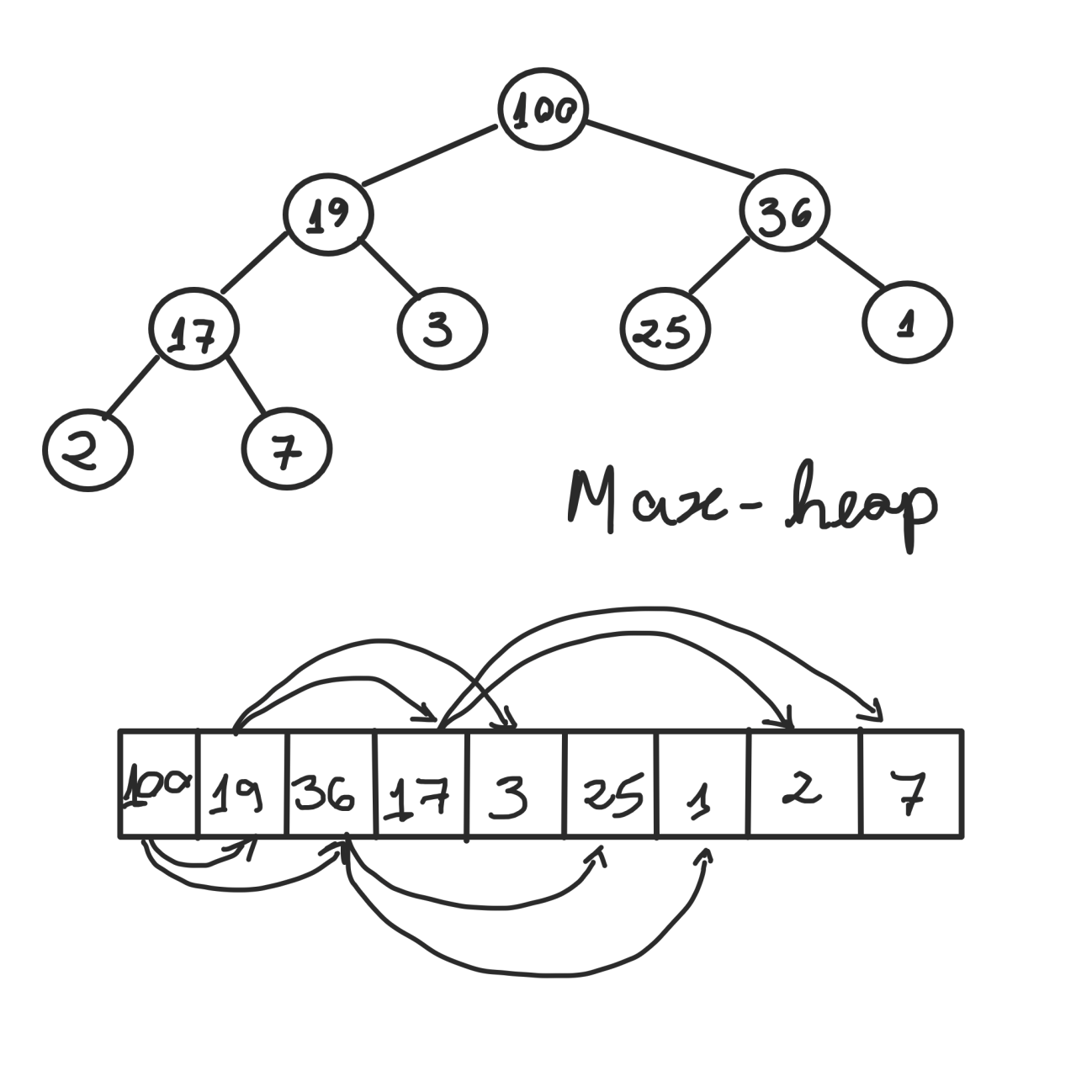
**Binary Heap (1964, J W J Williams)**

Complexidade:

|  |  |  |
| --- | --- | --- |
| **Algoritmo** | **Média** | **Pior Caso** |
| Espaço | O(n) | O(n) |
| Busca | O(n) | O(n) |
| Inserção | O(1) | O(log n) |
| Exclusão | O(log n) | O(log n) |
| Olhar | O(1) | O(1) |

****

Assume a forma de uma árvore binária. É uma forma comum de se implementar uma fila de prioridade.

É definida como uma árvore binária segue as seguintes propriedades:

* **Propriedade de forma:** Todos os níveis, exceto o mais profundo, estão preenchidos. O último deve ser preenchido da esquerda para a direita;
* **Propriedade de heap:** A chave armazenada em cada nó deve ser ≤ (ou ≥ para uma max-heap) que as chaves de seus descendentes.

Max-heaps são aquelas em que a chave de um nó é ≥ que as chaves de seus descendentes e min-heaps são aquelas em que a chave de um nó é ≤ que as chaves de seus descendentes.

**Operações em uma Heap**

As operações de inserir ou extrair modificam a heap para que ela se encaixe na propriedade de forma primeiro, adicionando ou removendo um elemento ao fim da heap. Em seguida a propriedade de heap é restaurada percorrendo-se a heap. Ambas as operações levam O(log n).

Inserção

Para fazer uma inserção é preciso executar uma operação de ***up-heap*** *(bubble-up, percolate-up, trickle-up, swim-up, heapify-up ou cascade-up),* seguindo o seguinte algoritmo:

1. Adicione o elemento ao fim da heap.
2. Compare o elemento adicionado ao seu pai; parar se a posição de ambos estiver correta.
3. Senão, trocar o elemento pelo seu pai e repetir o passo 2 em sua nova posição.

Extração

O procedimento para a extração da raiz da heap é chamado de down-heap *(bubble-down, sift-down, sink-down, trickle-down, heapify-down, cascade-down, or extract max/min)*. Os passos para a extração são os seguintes:

1. Substituir a raiz da heap pelo seu último elemento.
2. Comparar a nova raiz aos seus descendentes, se todos estão na posição correta, parar.
3. Senão, trocar o elemento por um dos seus descendentes e voltar ao passo anterior. Trocar pelo menor descendente em uma min-heap e pelo maior em uma max-heap.

Os passos 2 e 3 são executados pelo algoritmo a seguir (primeiro elemento no índice 1, tamanho no índice 0):

MaxHeapify (Heap, Root)

Left ← 2 \* Root

Right ← 2 \* Root + 1

Largest ← Root

If Left ≤ Heap[0] and Heap[Left] > Heap[Largest]:

Largest ← Left

If Right ≤ Heap[0] and Heap[Right] > Heap[Largest]:

Largest ← Right

If Largest ≠ Root:

Swap Heap[Root] and Heap[Largest]

MaxHeapify(Heap, Largest)

**Construindo uma Heap**

Uma das formas de se construir uma heap é começar com uma heap vazia e inserir os elementos um a um de forma a manter as propriedades de heap. Esta aproximação é conhecida como Williams’ method (quem apresentou a heap) e executa em tempo O(n log n).

A forma mais eficiente é inserir ais elementos arbitrariamente mantendo apenas a propriedade de forma da heap. Então, do nível inferior ao superior, usar o heapify na raiz de cada subárvore até que a propriedade de heap seja restaurada, o custo dessa abordagem é O(n). Essa também é a forma mais rápida de mesclar duas heaps.

Algoritmo para a construção de uma heap (índice inicial é 1, o índice 0 armazena o tamanho da heap):

BuildMaxHeap(Items)

For each index i from floor(Items[0] / 2) down to 1:

MaxHeapify(A, i);

**Implementação da Heap**

São normalmente armazenadas em arrays. Não é necessário espaço para ponteiros, pois as raízes e seus descendentes podem ser encontrados aritmeticamente, já que ela sempre é completa.

Seja *n* o número de elementos na heap e *i* um índice arbitrário válido do array armazenando a heap. Se a raiz da árvore está no índice 0, com índices válidos de 0 a n-1, então cada elemento *a* no índice *i* tem:

* Descendentes nos índices 2i + 1 e 2i + 2.
* Ascendente no índice floor((i-1) / 2).

Se a raiz está no índice 1, com índices variando de 1 a n, então cada elemento a no índice *i* tem:

* Filhos nos índices 2i e 2i + 1.
* Pai no índice floor(i / 2).