unipac.br
Barbacena

Bacharelado em Ciência da Computação

Estruturas de Dados

Material de Apoio

Parte XI – *Método HeapSort*

Prof. Nairon Neri Silva
naironsilva@unipac.br

2º sem / 2021

Filas de Prioridade

- É uma estrutura de dados onde a chave de cada item reflete sua habilidade relativa de abandonar o conjunto de itens rapidamente.

- Aplicações:
  - SOs usam filas de prioridades, nas quais as chaves representam o tempo em que eventos devem ocorrer.
  - Métodos numéricos iterativos são baseados na seleção repetida de um item com maior (menor) valor.
  - Sistemas de gerência de memória usam a técnica de substituir a página menos utilizada na memória principal por uma nova página.

Filas de Prioridade

Operações:

1. Constrói uma fila de prioridades a partir de um conjunto com n itens.
2. Informa qual é o maior item do conjunto.
3. Retira o item com maior chave.
4. Insere um novo item.
5. Aumenta o valor da chave do item i para um novo valor que é maior que o valor atual da chave.
6. Substitui o maior item por um novo item, a não ser que o novo item seja maior.
7. Altera a prioridade de um item.
8. Remove um item qualquer.
9. Agrupar duas filas de prioridades em uma única.

Filas de Prioridade

- Para representar uma fila de prioridades, pode-se utilizar uma lista linear ordenada; porém a melhor representação de fila de prioridades é através de uma estrutura de dados chamada *heap*.

Heap

- As operações das filas de prioridades podem ser utilizadas para implementar algoritmos de ordenação.
- Basta utilizar repetidamente a operação Insere para construir a fila de prioridades.
- Em seguida, utilizar repetidamente a operação Retira para receber os itens na ordem reversa.

Heap

- Há dois tipos de heap:
  - min-heap:
  - max-heap

- Abordaremos o max-heap.

Heap

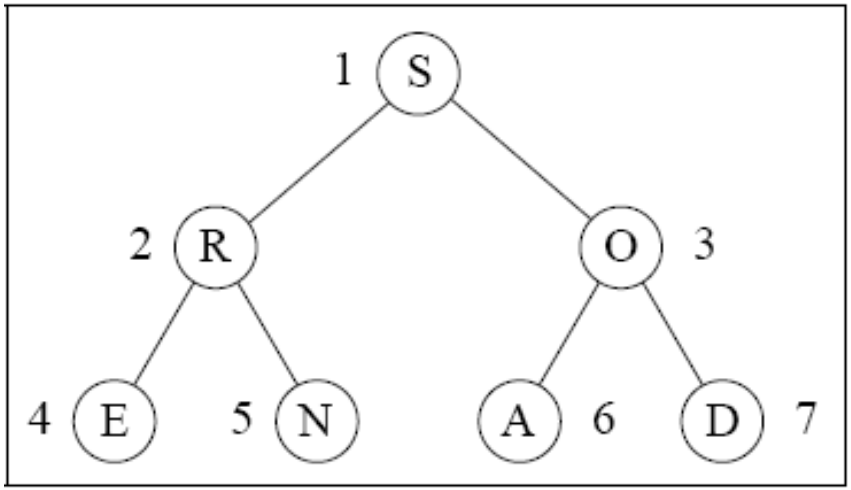
- Definição: é uma sequência de itens com chaves `c[1], c[2], ... , c[n]`, tal que:

```
c[i] >= c[2*i],
c[i] >= c[2*i + 1],
```

para todo `i = 1, 2, ..., n/2`

Heap

- A definição pode ser facilmente visualizada em uma árvore binária completa:

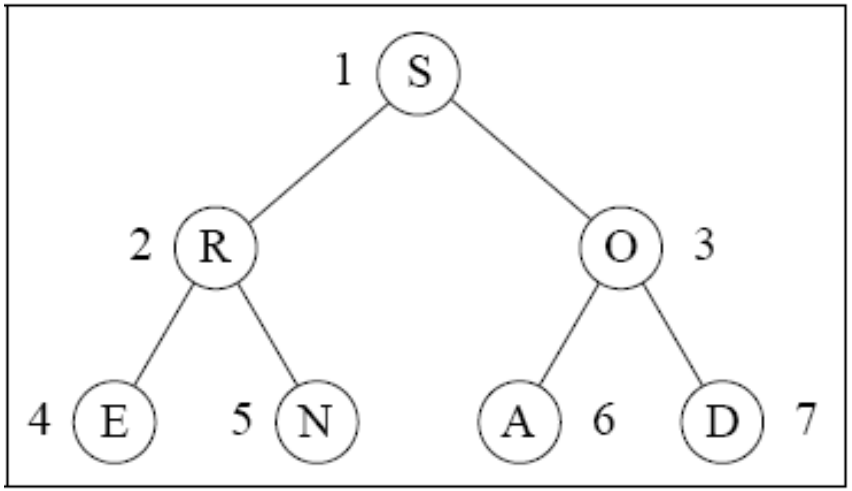


Heap

- Árvore binária completa:
  - Os nós são numerados de 1 a n.
  - O primeiro nó é chamado raiz.
  - O nó [k/2] é o pai do nó k, para 1 < k <= n.
  - Os nós 2k e 2k + 1 são os filhos à esquerda e à direita do nó k, para 1 <= k <= [k/2].
  - As chaves na árvore satisfazem a condição do *heap*.
  - A chave em cada nó é maior do que as chaves em seus filhos.
  - A chave no nó raiz é a maior chave do conjunto.

Heap

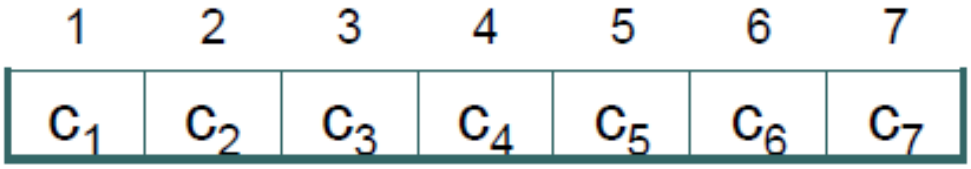
- Uma árvore binária completa pode ser representada por um array:

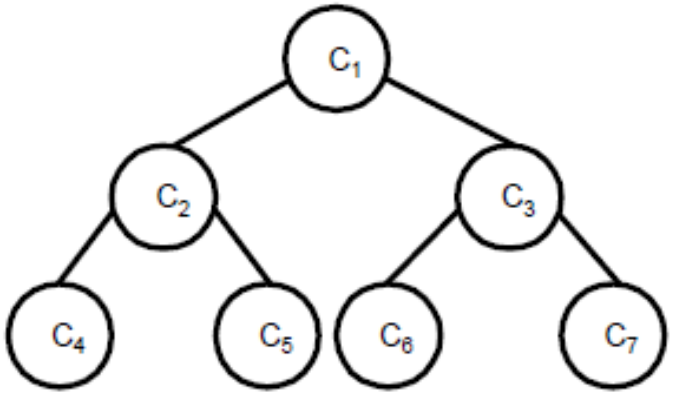


1	2	3	4	5	6	7
S	*R*	*O*	*E*	*N*	*A*	*D*

Heap

- Para um vetor com índices de 1 a 7 haveria o seguinte heap associado:





Heap - Características

- A representação é extremamente compacta, e permite caminhar pelos nós da árvore facilmente.

- Os filhos de um nó i estão nas posições 2i e 2i + 1, e o pai de um nó i está na posição i/2 (i div 2).

- Na representação do *heap* em um arranjo, a maior chave está sempre na posição 1 do vetor.

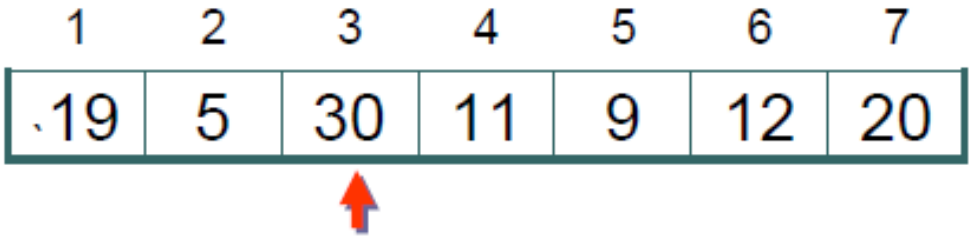
- Os algoritmos para implementar as operações sobre o *heap operam ao longo de* um dos caminhos da árvore.

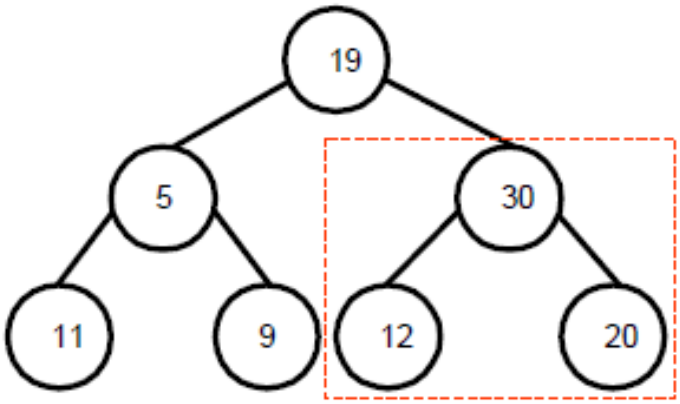
Heap - Características

- O algoritmo não necessita de nenhuma memória auxiliar.
- Dado um vetor A[1], A[2], ..., A[n], os itens A[n/2 + 1], A[n/2 + 2], ..., A[n] formam um *heap* se neste intervalo não existem dois índices i e j tais que j = 2i ou j = 2i + 1.

Construção de um Heap

- A transformação começa a partir do nó que ocupa a posição `n div 2` do vetor, neste caso a posição de índice 3:





Como esta sub-árvore respeita o critério para formação do heap, volta-se para o nó associado à posição anterior do vetor.

Construção de um Heap

- Os próximos nós serão obtidos decrementando-se o índice no vetor até atingir o índice 1

Construção de um Heap - Algoritmo

```
//função para refazer/reconstruir a condição de heap

void  refazHeap(int esq, int dir, int v[]){
    int i = esq;
    int j, aux;
    j = (i+1) * 2 - 1;
    aux = v[i];
    while (j <= dir){
        if (j < dir){
            if (v[j] < v[j+1])
            j++;
        }
        if (aux >= v[j])
            break;
        v[i] = v[j];
        i = j;
        j = i * 2 ;
    }
    v[i] = aux;
}
```

Construção de um Heap - Algoritmo

```
//função para construir o heap:
void constroiHeap(int v[], int n){
    int esq;
    esq = n / 2;
    while (esq > 0){
        esq--;
        refazHeap(esq, n, v);
    }
}
```

Heapsort

- Algoritmo:
  1. Construir o *heap.*
  2. Troque o item na posição 1 do vetor (raiz do *heap) com o item da posição n.*
  3. Use o procedimento constroiHeap para reconstituir o *heap para os itens* A[1], A[2], ..., A[n - 1].
  4. Repita os passos 2 e 3 com os n - 1 itens restantes, depois com os n - 2, até que reste apenas um item.

Heapsort - Algoritmo

```
//função que implementa o Heapsort

void heapSort(int v[], int n){
    int esq, dir;
    int x;
    constroiHeap(v, n-1); //constroi o heap
    esq = 0;
    dir = n - 1;
    while (dir > 0){ // ordena o vetor
        x = v[0];
        v[0] = v[dir];
        v[dir] = x;
        dir--;
        refazHeap(esq, dir, v);
    }
}
```

Heapsort - Exemplo

Vamos ordenar o seguinte conjunto de dados usando o algoritmo HeapSort:

V = {9, 4, 10, 8, 2, 1, 7}

Heapsort - Características

- Vantagens:
  - O comportamento do Heapsort é sempre O(n log n), qualquer que seja a entrada.
- Desvantagens:
  - O anel interno do algoritmo é bastante complexo se comparado com o do Quicksort.
  - O Heapsort não é **estável**.
- Recomendado:
  - Para aplicações que não podem tolerar eventualmente um caso desfavorável.
  - Não é recomendado para arquivos com poucos registros, por causa do tempo necessário para construir o *heap*.

Heapsort - Características

- Método proposto por Floyd e Williams, em 1964;
- É considerado um método de ordenação de bom desempenho;
- O desempenho no pior caso é semelhante ao caso médio;
- É considerado uma variação do método de ordenação por seleção;
- Faz um uso “comportado” da memória;

Heapsort - Exercício

Ordene o seguinte conjunto de dados usando o algoritmo HeapSort:

V = {3, 10, 11, 4, 1, 14, 2, 12, 0}

Referências

- FORBELLONE, André Luiz Villar; EBERSPACHER, Henri Frederico. *Lógica de Programação*. Makron books.
- GUIMARAES, Angelo de Moura; LAGES, Newton Alberto Castilho. *Algoritmos e estruturas de dados*. LTC Editora.
- FIDALGO, Robson. *Material para aulas*. UFRPE.

- NELSON, Fábio. *Material para aulas: Algoritmo e Programação.* UNIVASP.
- FEOFILOFF, P., *Algoritmos em linguagem C*, Editora Campus, 2008.
- ZIVIANI, N., *Projeto de algoritmos com Implementações em Pascal e C*, São Paulo: Pioneira, 2d, 2004.
- http://www.ime.usp.br/~pf/algoritmos/
- MELLO, Ronaldo S., *Material para aulas: Ordenação de Dados*, UFSC-CTC-INE
- MENOTTI, David, *Material para aulas: Algoritmos e Estrutura de Dados I*, DECOM-UFOP