DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Estrutura de Dados

Ordenação: Heapsort

Professores: Luiz Chaimowicz e Raquel Prates

Heapsort

- Possui o mesmo princípio de funcionamento da ordenação por seleção.
- Algoritmo:
  1. Selecione o menor (maior) item do vetor.
  2. Troque-o com o item da primeira (última) posição do vetor.
  3. Repita estas operações com os n – 1 itens restantes, depois com os n – 2 itens, e assim sucessivamente.
- No seleção, o custo para encontrar o menor (ou o maior) item entre n itens é n-1 comparações.
- Isso pode ser reduzido utilizando uma fila de prioridades.

Fila de Prioridades

- Definição:
  - Estrutura de dados composta de chaves, que suporta duas operações principais: inserção de um novo item e remoção do item com a maior chave.
  - A chave de cada item reflete a prioridade em que se deve tratar aquele item.
- Aplicações:
  - SO: alocação de processos, fila de impressão
  - Atendimento: prioridade de idosos, triagem
  - Etc....

Fila de Prioridades

- Operações
  - Constrói a fila de prioridade com N itens
  - Insere um novo item
  - Retira o maior item
  - Altera a prioridade de um item
  - ...

Fila de Prioridades

- Representações
  - Lista sequencial ordenada
  - Lista sequencial não ordenada
  - **Heap**

	Constrói	Insere	Retira máximo	Altera prioridade
Lista ordenada	O(N log N)	O(N)	O(1)	O(N)
Lista não ordenada	O(N)	O(1)	O(N)	O(1)
Heaps	O(N log N)	O(log N)	O(log N)	O(log N)

Heap

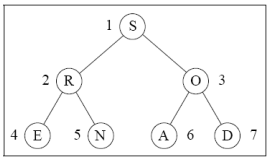
- É uma sequência de itens com chaves c[1], c[2], ... , c[n], tal que, para todo i = 1, 2, ..., n/2 :

$$c[i] >= c[2i],\ c[i] >= c[2i + 1],$$

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

Heap

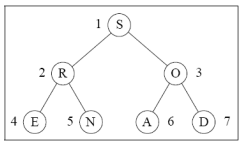
- Essa definição pode ser facilmente visualizada em uma **árvore binária completa**:
  - Será um heap se cada nó for maior ou igual seus filhos.
  - Com isso, a maior chave estará na raiz



Heap

- Representação vetorial para de árvore
  - Nós são numerados de 1 a n
  - O primeiro é chamado raiz
  - O nó floor(k/2) é o pai do nó k, $1 < k \leq n$
  - Os nós 2k e 2k+1 são filhos da esquerda e direita do nó k, para $1 \leq k \leq n/2$.

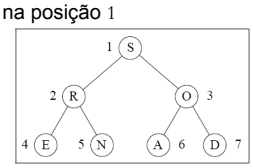
1	2	3	4	5	6	7
S	*R*	*O*	*E*	*N*	*A*	*D*



Heap

- Representação por meio de vetores é compacta
- Permite caminhar pelos nós da árvore facilmente
  - Filhos de um nó i estão nas posições $2i$ e $2i + 1$
  - O pai de um nó i está na posição floor$(i/2)$
  - A maior chave sempre está na posição 1

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



Construção de um heap

- Dado um vetor A[1], A[2], ..., A[n], os itens A[**n/2** + 1], A[n/2 + 2], ..., A[n] formam um *heap:*
  - Neste intervalo não existem dois índices i e j tais que j = 2i ou j = 2i + 1. (nós-folhas da árvore)

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

- Os elementos restantes vão sendo inseridos no heap da direita para a esquerda, reorganizando-os caso a condição de heap seja violada

Construção do Heap

- Exemplo:

	1	2	3	4	5	6	7
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*	*S*
Esq = 3	*O*	*R*	***S***	*E*	*N*	*A*	***D***
Esq = 2	*O*	*R*	*S*	*E*	*N*	*A*	*D*
Esq = 1	***S***	*R*	***O***	*E*	*N*	*A*	*D*

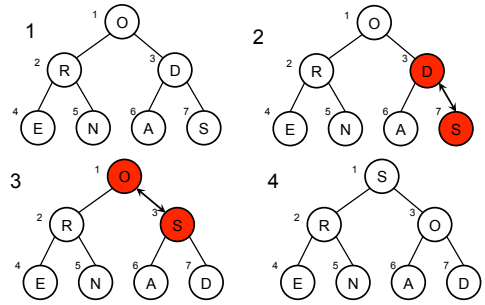
Construção do Heap

- Exemplo:
  - Os itens de A[4] a A[7] formam um *heap.*
  - O *heap é estendido para a esquerda* (Esq = 3), englobando o item A[3], pai dos itens A[6] e A[7].
  - A condição de *heap é violada:*
    - O *heap é refeito trocando os itens D e S.*
  - O item R é incluindo no *heap (Esq = 2), o que* não viola a condição de *heap.*
  - O item O é incluindo no *heap (Esq = 1).*
  - A Condição de *heap violada:*
    - O *heap é refeito trocando os itens O e S,* encerrando o processo.

Construção do Heap

- Visualizando como uma árvore

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



Heap – Construção do heap

```
void Constroi(Item *A, int n) {
  int Esq;

  Esq = n / 2 + 1;
  while (Esq > 1) {
    Esq--;
    Refaz(Esq, n, A);
  }
}
```


Refaz a condição do heap

```
void  Refaz(int Esq, int Dir, Item *A){
  int i, j;
  Item x;
  i = Esq;
  j = i * 2; //A[j] eh filho de A[i]
  x = A[i];
  while (j <= Dir){
    if (j < Dir)
       //A[j+1] eh o 2o filho de A[i]
       if (A[j].Chave < A[j+1].Chave) j++;
    if (x.Chave >= A[j].Chave) break;
 //o maior dos filhos deve ser menor do que o pai
    A[i] = A[j];//"sobe" os filhos for a de lugar
    i = j;
    j = i *2;//vamos checar os netos …
  }
  A[i] = x;//insere A[esq] na posicao correta
}
```


Constrói o heap

```
void Constroi(Item *A, int n) {
  int Esq;

  Esq = n / 2 + 1;
  while (Esq > 1) {
    Esq--;
    Refaz(Esq, n, A);
  }
}
```


Fila de prioridades com Heap

- Para obter o elemento com maior prioridade, remove ele e reconstrua o heap

```
Item RetiraMax(Item *A, int *n) {
  Item Maximo;
  if (*n < 1)
    printf("Erro: heap vazio\n");
  else {
    Maximo = A[1];
    A[1] = A[*n];
    (*n)--;
    Refaz(1, *n, A);
  }
  return Maximo;
}
```


Ordenação com 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 Refaz 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

```
void Heapsort(Item *A, int *n) {
  int Esq, Dir;
  Item x;
  Constroi(A, n); /* constroi o heap */
  Esq = 1; Dir = *n;
  while (Dir > 1)
  { /* ordena o vetor */
    x = A[1];
    A[1] = A[Dir];
    A[Dir] = x;
    Dir--;
    Refaz(Esq, Dir, A);
  }
}
```


Heapsort - Exemplo

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

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

Constrói Heap

Heapsort

- **Análise**
  - O procedimento Refaz gasta cerca de log n operações, no pior caso.
  - Logo, Heapsort gasta um tempo de execução proporcional a n log n, no pior caso.

Heapsort

- 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*.

Exercício

- Execute o heapsort no vetor abaixo. Mostre os passos da construção do heap e da execução do heapsort

05	23	51	25	03	84	48	21	65