Unidade IV:

Ordenação Parcial

PUC Minas

Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

Agenda

- Introdução
- Seleção Parcial
- Inserção Parcial
- Heapsort Parcial
- Quicksort Parcial

Agenda

- **Introdução**

- Seleção Parcial

- Inserção Parcial

- Heapsort Parcial

- Quicksort Parcial

Introdução

- Consiste em obter os *k* primeiros elementos de um arranjo ordenado com *n* elementos
- Quando *k* = *1*, o problema se reduz a encontrar o mínimo (ou o máximo) de um conjunto de elementos
- Quando *k* = *n* caímos no problema clássico de ordenação

Aplicações

- Facilitar a busca de informação na Web com as **máquinas de busca**:
  - É comum uma consulta na Web retornar centenas de milhares de documentos relacionados com a consulta
  - O usuário está interessado apenas nos *k* documentos mais relevantes
  - Em geral, *k* é menor do que *200* documentos
  - Normalmente, são consultados apenas os dez primeiros
  - Assim, são necessários algoritmos eficientes de ordenação parcial

Agenda

- Introdução
- **Seleção Parcial**
- Inserção Parcial
- Heapsort Parcial
- Quicksort Parcial

Seleção Parcial

- Um dos algoritmos mais simples

- Princípio de funcionamento:

    - Selecione o menor item do vetor

    - Troque-o com o item que está na primeira posição do vetor

    - Repita essas duas operações com os itens $n - 1, n - 2, \ldots, n - k$.

Seleção Parcial

```
void selecaoParcial(Vetor A, int n, int k) {
    int i , j , min;
    for ( i = 1; i <= k ; i++) {
        min = i ;
        for ( j = i + 1; j <= n; j++){
            if (A[ j ].chave < A[min].chave) min = j;
        }
        swap(A[min], A[ i ]) ;
    }
}
```

Seleção Parcial

```
void selecaoParcial(Vetor A, int n, int k) {
    int i , j , min;
    for ( i = 1; i <= k ; i++) {
        min = i ;
        for ( j = i + 1; j <= n; j++){
            if (A[ j ].chave < A[min].chave) min = j;
        }
        swap(A[min], A[ i ]) ;
    }
}
```

Seleção Parcial

- Comparações entre chaves e movimentações de registros:

$$C(n) = kn - \frac{k^2}{2} - \frac{k}{2}$$

$$M(n) = 3k$$

Agenda

- Introdução
- Seleção Parcial
- **Inserção Parcial**
- Heapsort Parcial
- Quicksort Parcial

Inserção Parcial

- Pode ser obtido a partir do algoritmo de ordenação por Inserção efetuando uma modificação simples:
    - Tendo sido ordenados os primeiros k itens, o item da k-ésima posição funciona como um pivô
    - Quando um item entre os restantes é menor do que o pivô, ele é inserido na posição correta entre os k itens de acordo com o algoritmo original

Inserção Parcial

```
void insercaoParcial (int A, int n, int k) {
    int i , j ;
    for (i = 2; i <= n; i++) {
        x = A[ i ] ;
        if ( i > k) j = k ; else j = i − 1;
        A[0] = x ; /* sentinela */
        while ( j > 0 && x.chave < A[ j ].chave) {
            A[ j +1] = A[ j ] ;
            j−−;
        }
        A[ j +1] = x;
    }
}
```

Inserção Parcial

```
void insercaoParcial (int A, int n, int k) {
    int i , j ;
    for (i = 2; i <= n; i++) {
        x = A[ i ] ;
        if ( i > k) j = k ; else j = i − 1;
        A[0] = x ; /* sentinela */
        while ( j > 0 && x.chave < A[ j ].chave) {
            A[ j +1] = A[ j ] ;
            j−−;
        }
        A[ j +1] = x;
    }
}
```

Inserção Parcial

- No anel mais interno, na i-ésima iteração o valor de C_i é:

$$
\begin{aligned}
&\text{Melhor caso} &&: C_i(n) = 1 \\
&\text{Pior caso} &&: C_i(n) = i \\
&\text{Caso médio} &&: C_i(n) = \tfrac{1}{i}(1 + 2 + \cdots + i) = \tfrac{i+1}{2}
\end{aligned}
$$

- Assumindo que todas as permutações de n são igualmente prováveis, o número de comparações é:

$$
\begin{aligned}
&\text{Melhor caso} &&: C(n) &&= (1 + 1 + \cdots + 1) = n - 1 \\
&\text{Pior caso} &&: C(n) &&= (2 + 3 + \cdots + k + (k+1)(n-k)) \\
& && &&= kn + n - \tfrac{k^2}{2} - \tfrac{k}{2} - 1 \\
&\text{Caso médio} &&: C(n) &&= \tfrac{1}{2}(3 + 4 + \cdots + k + 1 + (k+1)(n-k)) \\
& && &&= \tfrac{kn}{2} + \tfrac{n}{2} - \tfrac{k^2}{4} + \tfrac{k}{4} - 1
\end{aligned}
$$

Agenda

- Introdução
- Seleção Parcial
- Inserção Parcial
- **Heapsort Parcial**
- Quicksort Parcial

Heapsort Parcial

- Utiliza um *heap* para informar o menor item do conjunto
- Na primeira iteração, o menor item que está em **A[1]** (raiz do *heap*) é trocado com o item que está em **A[n]**
- Em seguida, o *heap* é refeito
- Novamente, o menor está em **A[1]**, troque-o com **A[n-1]**
- Repita as duas últimas operações até que o k-ésimo menor seja trocado com ***A[n − k]***
- Ao final, os *k* menores estão nas *k* últimas posições do vetor **A**

Heapsort Parcial

```
void heapsortParcial(int *A, int n, int k) {
    int esq = 1, dir, Aux = 0;
    constroi (A, n) ;    /* constroi o heap */
    dir = n;
    while (Aux < k) {  /* ordena o vetor */
        x = A[1] ;
        A[1] = A[n − Aux] ;
        A[n − Aux] = x;
        dir−−; Aux++;
        refaz(esq, dir , A) ;
    }
}
```

Heapsort Parcial

- Constrói um heap com custo *O(n)*
- O procedimento refaz tem custo *O(lg(n))*
- O procedimento heapParcial chama o refaz k vezes
- Logo, o algoritmo apresenta a complexidade:

$$O(n + k \log n) = \begin{cases} O(n) & \text{se} \quad k \leq \frac{n}{\log n} \\ O(k \log n) & \text{se} \quad k > \frac{n}{\log n} \end{cases}$$

Agenda

- Introdução
- Seleção Parcial
- Inserção Parcial
- Heapsort Parcial
- **Quicksort Parcial**

Quicksort Parcial

- Assim como o Quicksort, sua versão parcial é o algoritmo de ordenação parcial mais rápido em várias situações

- A alteração no algoritmo para que ele ordene apenas os *k* primeiros itens dentre *n* itens é simples

- Basta abandonar a partição à direita toda vez que a partição à esquerda contiver *k* ou mais itens. Assim, a única alteração necessária no Quicksort é evitar a chamada recursiva *Ordena(i,dir).*

Quicksort Parcial

- Considere **k = 3** e **D** o pivô para gerar as linhas **2** e **3**.
- A partição à esquerda contém dois itens e a partição à direita, quatro itens.
- A partição à esquerda contém menos do que ***k*** itens.
- Logo, a partição direita não pode ser abandonada.
- Considere **E** o pivô na linha 3.
- A partição à esquerda contém três itens e a partição à direita também.
- Assim, a partição à direita pode ser abandonada.

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

Quicksort Parcial

```
void quicksort(int esq, int dir, int k) {
    int i = esq,   j = dir,   pivo = array[(dir+esq)/2];
    while (i <= j) {
        while (array[i] < pivo)
            i++;
        while (array[j] > pivo)
            j--;
        if (i <= j)
        {    swap(i, j);   i++;      j--;  }
    }
    if ( j − esq >= k − 1) { if (esq < j ) quicksort(esq, j , k ) ; return; }
    if (esq < j)
        quicksort(esq, j, k);
    if (i < dir)
        quicksort(i, dir, k);
}
```

Quicksort Parcial

- A análise do Quicksort é difícil
- O comportamento é muito sensível à escolha do pivô
- Podendo cair no melhor caso $O(k \times lg(k))$
- Ou em algum valor entre o melhor caso e $O(n \times lg(n))$