DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Estrutura de Dados

Ordenação: Introdução e métodos elementares

Professores: Luiz Chaimowicz e Raquel Prates

Ordenação

- Objetivo:
  - Rearranjar os itens de um vetor ou lista de modo que suas chaves estejam ordenadas de acordo com alguma regra.
- Estrutura:
  - um vetor v vai ser um `Item *v` ou `Item v[max]`

```
typedef int TipoChave;
typedef struct {
    ChaveTipo chave;
    /* outros componentes */
} Item;
```

Critérios de Classificação

- Localização dos dados:
  - Ordenação interna: todas as chaves estão na memória principal.
  - Ordenação externa: chaves na memória principal e na memória secundária.
- Estabilidade:
  - Relacionado com o manutenção da ordem relativa entre chaves de mesmo valor.
  - Método é estável se a ordem relativa dos registros com a mesma chave não se altera após a ordenação.

Critérios de Classificação

- Uso da memória:
  - ***In place:*** transforma os dados de entrada utilizando apenas um espaço extra de tamanho constante.
- Movimentação dos dados:
  - Direta: registro todo é acessado e deve ser movido
  - Indireta: apenas as chaves são acessadas e ponteiros são rearranjados e não o registro todo.
- Adaptabilidade:
  - Sequência de operações executadas conforme a entrada.
  - Não adaptável: operações executadas independe da entrada.
- **Comparação de Chaves** x Outros

Critérios de Avaliação

- Seja n o número de registros em um vetor, considera-se duas medidas de complexidade:
  - Número de comparações C(n) entre as chaves;
  - Número de trocas ou movimentações M(n) de itens;

```
#define Troca(A, B) {Item c = A; A = B; B = c; }

void Ordena(Item *v, int n) {
   int i, j;
   for (i = 0; i < n-1; i++) {
      for (j = n-1; j > i; j--) {
         if (v[j-1].chave > v[j].chave)  /* comparações */
            Troca(v[j-1], v[j]);  /* trocas */
}
```


Método Bolha

- Ideia:
  - Passa no arquivo e troca elementos adjacentes que estão fora de ordem, até os registros ficarem ordenados.

Método Bolha

- Algoritmo
  - Supondo movimentação da esquerda para direita no vetor;
  - Cada elemento é comparado com o seguinte. Se a ordem estiver invertida, a posição dos dois é trocada;
  - Quando o maior elemento do vetor for encontrado, ele será trocado até ocupar a última posição;
  - Na segunda passada, o segundo maior será movido para a penúltima posição do vetor.
  - E assim por diante durante n-1 passadas...

Método Bolha

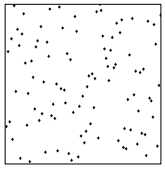
6 5 3 1 8 7 2 4

Método Bolha

```
void Bolha (Item *v, int n)
{
  int i, j;

  for(i = 0; i < n-1; i++)
    for(j = 1; j < n-i; j++)
      if (v[j].chave < v[j-1].chave)
        Troca(v[j-1], v[j]);
}
```


Método Bolha



```
void Bolha (Item *v, int n) {
int i, j;

  for(i = 0; i < n-1; i++)
    for(j = 1; j < n-i; j++)
      if (v[j].chave < v[j-1].chave)
        Troca(v[j-1], v[j]);
}
```

E	X	E	M	P	L	O
E	E	M	P	L	O	X
E	E	M	L	O	P	X
E	E	L	M	O	P	X
E	E	L	M	O	P	X
E	E	L	M	O	P	X
E	E	L	M	O	P	X
E	E	L	M	O	P	X

Método Bolha: Complexidade

- Comparações – C(n):

```
void Bolha (Item *v, int n) {
int i, j;

  for(i = 0; i < n-1; i++)
    for(j = 1; j < n-i; j++)
      if (v[j].chave < v[j-1].chave)
        Troca(v[j-1], v[j]);
}
```

i	comparações
0	n-1
1	n-2
2	n-3
...	...
n-2	1

$$C(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

Método Bolha: Complexidade

Movimentações – M(n)

```
void Bolha (Item *v, int n) {
int i, j;

  for(i = 0; i < n-1; i++)
    for(j = 1; j < n-i; j++)
      if (v[j].chave < v[j-1].chave)
        Troca(v[j-1], v[j]); // 3 movimentações
}
```

- Pior Caso: M(n) = 3xC(n)
  - Vetor inversamente Ordenado
- Melhor Caso: M(n) = 0
  - Vetor Ordenado

Método Bolha

- Vantagens
  - Algoritmo simples, in-place
  - Algoritmo estável
  - Com pequena otimização, detecta um vetor ordenado em uma passada: melhor caso O(n) para entrada ordenada
- Desvantagens
  - Não adaptável em termos de comparações (sem otimização)
  - Ineficiente: pior e médio casos: $O(n^2)$

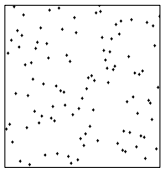
- Possível Melhoria
  - Parar o algoritmo quando não forem efetuadas trocas em uma passagem (menos comparações)

```
void Bolha (Item *v, int n) {
int i, j, trocou;

  for(i = 0; i < n-1; i++){
    trocou = 0;
    for(j = 1; j < n-i; j++)
      if (v[j].chave < v[j-1].chave){
        Troca(v[j-1], v[j]);
        trocou = 1;
      }
    if (!trocou) break;
  }
}
```


Variação Bolha: Ordenação Par-Ímpar

```
void ParImpar(Item *v, int n) {
int ordenado = 0;
while(!ordenado) {
   ordenado = 1;
   for(int i = 0; i < n-1; i += 2)
      if(v[i] > v[i+1]) {
         Troca(v[i], v[i+1]);
         ordenado = 0;
      }
   for (int i = 1; i < n-1; i += 2)
      if(v[i] > v[i+1]) {
         Troca(v[i], v[i+1]);
         ordenado = 0;
      }
}
```



Método Seleção

- Seleção do n-ésimo menor (ou maior) elemento da lista
- Troca do n-ésimo menor (ou maior) elemento com a n-ésima posição da lista
- Uma única troca por vez é realizada

8
5
2
6
9
3
1
4
0
7

Método Seleção

```
void Selecao (Item *v, int n)
{
  int i, j, Min;

  for (i = 0; i < n - 1; i++)
  {
    Min = i;
    for (j = i + 1 ; j < n; j++)
    {
      if (v[j].chave < v[Min].chave)
        Min = j;
    }
    Troca(v[i], v[Min]);
  }
}
```


Método Seleção

```
void Selecao (Item *v, int n)
{
  int i, j, Min;

  for (i = 0; i < n - 1; i++)
  {
    Min = i;
    for (j = i + 1 ; j < n; j++)
    {
      if (v[j].chave < v[Min].chave)
        Min = j;
    }
    Troca(v[i], v[Min]);
  }
}
```

(E)	X	E	M	P	L	O
E	X	(E)	M	P	L	O
E	E	X	M	P	(L)	O
E	E	L	(M)	P	X	O
E	E	L	M	P	X	(O)
E	E	L	M	O	X	(P)
E	E	L	M	O	P	(X)
E	E	L	M	O	P	X

Método Seleção: Complexidade

- Comparações – C(n):
  - Melhor, pior e caso médio:

$$C(n) = \sum_{i=0}^{n-2}(n-1)-(i+1)+1 = \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1$$

$$= n(n-1) - \frac{(n-1)(n-2)}{2} - (n-1)$$

$$= \frac{n^2-n}{2} = O(n^2)$$

- Movimentações – M(n):
  - Melhor, pior e caso médio:

$$M(n) = 3(n-1) = O(n)$$

Método Seleção

```
void Selecao (Item *v, int n)
{
  int i, j, Min;

  for (i = 0; i < n - 1; i++)
  {
    Min = i;
    for (j = i + 1 ; j < n; j++)
    {
      if (v[j].chave < v[Min].chave)
        Min = j;
    }
    Troca(v[i], v[Min]);
  }
}
```

i	comparações
0	n-1
1	n-2
2	n-3
...	...
n-2	1

$$C(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

Método Seleção: Complexidade

- Comparações – C(n):
  - Melhor, pior e caso médio:

$$C(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2 - n}{2} = O(n^2)$$

- Movimentações – M(n):
  - Melhor, pior e caso médio:

$$M(n) = 3(n-1) = O(n)$$

Método Seleção

- Vantagens:
  - M(n)=O(n): Custo linear no tamanho da entrada para o número de movimentos de registros – a ser utilizado quando há registros muito grandes
  - In place
  - Admite implementação eficiente usando listas lineares. Exercício: Como?
- Desvantagens:
  - C(n)=O(n^2) em todos os casos: Não adaptável (não importa se o arquivo está parcialmente ordenado);
  - Algoritmo não é estável: Exercício: Por que?

Método Inserção

- Algoritmo utilizado pelo jogador de cartas
  - As cartas são ordenadas da esquerda para direita uma a uma.
  - O jogador escolhe a segunda carta e verifica se ela deve ficar antes ou na posição que está.
  - Depois a terceira carta é classificada, deslocando-a até sua correta posição.
  - O jogador realiza esse procedimento até ordenar todas as cartas.

6 5 3 1 8 7 2 4

Método Inserção

6 5 3 1 8 7 2 4

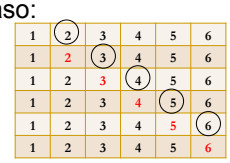
Método Inserção

```
void Insercao(Item *v, int n) {
int i,j;
Item aux;
  for (i = 1; i < n; i++) {
    aux = v[i];
    j = i - 1;
    while (( j >= 0 ) && (aux.Chave < v[j].Chave)) {
      v[j + 1] = v[j]; //Move para direita
      j--;
    }
    v[j + 1] = aux;
  }
}
```

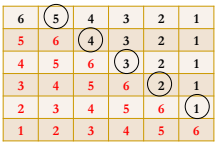
E	(X)	E	M	P	L	O
E	X	(E)	M	P	L	O
E	E	X	(M)	P	L	O
E	E	M	X	(P)	L	O
E	E	M	P	X	(L)	O
E	E	L	M	P	X	(O)
E	E	L	M	O	P	X

Método Inserção: Exemplos

- Melhor Caso:



- Pior Caso:



Método Inserção: Complexidade

- Comparações – C(n):
  - Anel interno: i-ésima iteração, valor de C_i:
    - melhor caso: $C_i = 1$
    - pior caso: $C_i = i$
  - Anel externo: $\sum_{i=1}^{n-1} C_i$
  - Complexidade total:
    - Melhor caso (itens já estão ordenados)

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 = O(n)$$

    - Pior caso (itens em ordem reversa):

$$C(n) = \sum_{i=1}^{n-1} i = \frac{(n-1)(n)}{2} = \frac{n^2}{2} - \frac{n}{2} = O(n^2)$$

Método Inserção: Complexidade

- Movimentações - M(n):
  - **Melhor caso:** *O(1) – vetor já ordenado*
  - **Pior caso:** *O(n²): 2(n-1) + n(n-1)/2* – sempre entra no loop interno: pior caso das comparações

Método Inserção: Complexidade

- Movimentações - M(n):
  - 2 movimentações no loop externo + 1 no loop interno
  - **Melhor caso:** *2(n-1)* – nunca entra no loop interno
  - **Pior caso:** *2(n-1) + n(n-1)/2* – sempre entra no loop interno: pior caso das comparações

Método Inserção – Melhoria

- Uso de um sentinela

```
void Insercao(Item *x, Indice n){
  Indice i, j;
  Item aux;
  for (i = 2; i <= n; i++){
    aux = v[i];
    j = i -1;
    A[0] = aux; /* sentinela */
    while (x.Chave < v[j].Chave){
      v[j+1] = v[j];
      j--;
    }
    v[j+1] = aux;
  }
}
```

Evita uma comparação de índice a cada iteração ao custo de uma eventual comparação de chaves

Método Inserção

- Vantagens:
  - Algoritmo adaptável à ordenação inicial da entrada:
    - Melhor caso C(n)=O(n): É o método a ser utilizado quando o arquivo está "quase" ordenado.
    - Melhor caso M(n)=O(1) É um bom método quando se deseja adicionar uns poucos itens a um arquivo ordenado.
  - O algoritmo de ordenação por inserção é **estável.**
- Desvantagens:
  - Ineficiente para entradas grandes O(n²) Alto custo de movimentação de elementos no vetor.

Exercícios

- 1. Mostre um exemplo que mostre que o Método de Seleção não é estável.
- 2. É possível criar uma versão do Método de seleção estável? Justifique sua resposta e em caso de você achar que é possível, mostre como o algoritmo deve ser implementado.
- 3. Implemente o Método da Bolha recursivo.

Exercícios

- 1. Mostre um exemplo que mostre que o Método de Seleção não é estável.
  - Exemplo de entrada para Selection Sort: **7**,3,2,7,**1**
  - Ao final do primeiro round, o 1º 7 é trocado (*swap*) de lugar com 1: [**1**,3,2,7,**7**], alterando a ordem relativa entre os elementos repetidos 7

Exercícios

- 2. É possível criar uma versão do Método de seleção estável? Justifique sua resposta e em caso de você achar que é possível, mostre como o algoritmo deve ser implementado.
  - SIM.
  - Implementação estável de selection sort:
  - Usar lista encadeada e, ao final da **fase i**, inserir o elemento mínimo na pos **i** da lista ao invés de realizar *swap*:
  - Versão swap: [**3**,5,**2**,2],[**2**,5,**3**,2],[2,**5**,3,**2**],[2,**2**,3,**5**]…
  - Versão lista: [3,5,**2**,2],[**2**,3,5,2],[2,3,5,**2**],[2,**2**,3,5]…

Exercícios

- 3. Implemente o Método da Bolha recursivo
  - Caso base: se n=1, return;
  - Faça um apassada do Bubble sort para fixar o último elemento n-1 do subvetor corrente
  - Chame recursivamene para todos os elementos exceto o último do subvetor

```
void bubbleSort(int arr[], int n)
{
    if (n == 1)
        return;

    for (int i=0; i<n-1; i++)
        if (arr[i] > arr[i+1])
            swap(arr[i], arr[i+1]);

    bubbleSort(arr, n-1);
}
```