INF1007: Programação 2

6 – Ordenação de Vetores

Tópicos

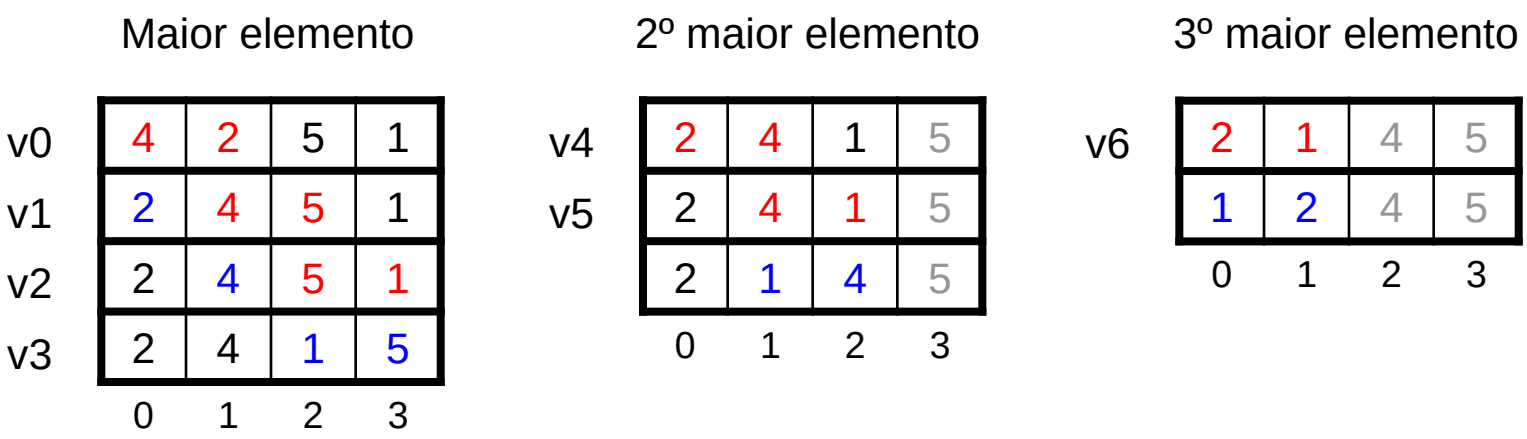
- Introdução
- Ordenação bolha (*bubble sort*)
- Ordenação por seleção (*selection sort*)

Introdução

- Ordenação de vetores:
  - entrada: vetor com os elementos a serem ordenados
  - saída: mesmo vetor com elementos na ordem especificada
  - ordenação:
    - pode ser aplicada a qualquer dado com ordem bem definida
    - vetores com dados complexos (structs)
      - chave da ordenação escolhida entre os campos
      - elemento do vetor contém apenas um ponteiro para os dados
      - troca da ordem entre dois elementos = troca de ponteiros

Ordenação Bolha

- Ordenação bolha:
  - processo básico:
    - quando dois elementos estão fora de ordem, troque-os de posição até que o i-ésimo elemento de maior valor do vetor seja levado para as posições finais do vetor
  - continue o processo até que todo o vetor esteja ordenado



Ordenação Bolha

```
25 48 37 12 57 86 33 92    25x48
25 48 37 12 57 86 33 92    48x37 troca
25 37 48 12 57 86 33 92    48x12 troca
25 37 12 48 57 86 33 92    48x57
25 37 12 48 57 86 33 92    57x86
25 37 12 48 57 86 33 92    86x33 troca
25 37 12 48 57 33 86 92    86x92
25 37 12 48 57 33 86 92    final da primeira passada
```

o maior elemento, 92, já está na sua posição final

Ordenação Bolha

```
25 37 12 48 57 33 86 92    25x37
25 37 12 48 57 33 86 92    37x12 troca
25 12 37 48 57 33 86 92    37x48
25 12 37 48 57 33 86 92    48x57
25 12 37 48 57 33 86 92    57x33 troca
25 12 37 48 33 57 86 92    57x86
25 12 37 48 33 57 86 92    final da segunda passada
```

o segundo maior elemento, 86, já está na sua posição final

Ordenação Bolha

```
25 12 37 48 33 57 86 92    25x12 troca
12 25 37 48 33 57 86 92    25x37
12 25 37 48 33 57 86 92    37x48
12 25 37 48 33 57 86 92    48x33 troca
12 25 37 33 48 57 86 92    48x57
12 25 37 33 48 57 86 92    final da terceira passada
```

Idem para 57.

```
12 25 37 33 48 57 86 92    12x25
12 25 37 33 48 57 86 92    25x37
12 25 37 33 48 57 86 92    37x33 troca
12 25 33 37 48 57 86 92    37x48
12 25 33 37 48 57 86 92    final da quarta passada
```

Idem para 48.

```
12 25 33 37 48 57 86 92    12x25
12 25 33 37 48 57 86 92    25x33
12 25 33 37 48 57 86 92    33x37
12 25 33 37 48 57 86 92    final da quinta passada
```

Idem para 37.

```
12 25 33 37 48 57 86 92    12x25
12 25 33 37 48 57 86 92    25x33
12 25 33 37 48 57 86 92    final da sexta passada
```

Idem para 33.

```
12 25 33 37 48 57 86 92    12x25
12 25 33 37 48 57 86 92    final da sétima passada
```

Idem para 25 e, conseqüentemente, 12.

```
12 25 33 37 48 57 86 92    final da ordenação
```

Ordenação Bolha

- Implementação Iterativa (I):

```
/* Ordenação bolha */
void bolha (int n, int* v)
{
   int fim,i;
   for (fim=n-1; fim>0; fim--)
      for (i=0; i<fim; i++)
         if (v[i]>v[i+1]) {
            int temp = v[i];      /* troca */
            v[i] = v[i+1];
            v[i+1] = temp;
         }
}
```

maior elemento
(n=4; fim=n-1=3)

4	2	5	1
2	4	5	1
2	4	5	1
2	4	1	5

2º maior elemento
(fim=n-2=2)

2	4	1	5
2	4	1	5
2	1	4	5

3º maior elemento
(fim=n-3=1)

2	1	4	5
1	2	4	5

0 1 2 3

Ordenação Bolha

- Implementação Iterativa (II):

```
/* Ordenação bolha (2a. versão) */
void bolha (int n, int* v)
{  int fim,i;
   for (fim=n-1; fim>0; fim--) {
        int troca = 0;
        for (i=0; i<fim; i++)
            if (v[i]>v[i+1]) {
                  int temp = v[i];  /* troca */
                  v[i] = v[i+1];
                  v[i+1] = temp;
                  troca = 1;
             }
       if (troca == 0) return;     /* não houve troca */
   }
}
```

pára quando há

uma passagem inteira

sem trocas

Ordenação Bolha

- Esforço computacional:
  - esforço computacional ≅ número de comparações
    ≅ número máximo de trocas
    - primeira passada: n-1 comparações
    - segunda passada: n-2 comparações
    - terceira passada: n-3 comparações
    - ...
  - tempo total gasto pelo algoritmo:
    - T proporcional a (n-1) + (n-2) + ... + 2 + 1 = (n-1+1)*(n-1) / 2 = n*(n-1) / 2
    - algoritmo de ordem quadrática: $O(n^2)$

Ordenação Bolha

- **Implementação recursiva:**

```
/* Ordenação bolha recursiva */
void bolha_rec (int n, int* v)
{  int i;
   int troca = 0;
   for (i=0; i<n-1; i++)
        if (v[i]>v[i+1]) {
           int temp = v[i];   /* troca */
           v[i] = v[i+1];
           v[i+1] = temp;
           troca = 1;
      }
   if (troca != 0) && (n>1)  /* houve troca */
      bolha_rec(n-1,v);
}
```

maior elemento
bolha_rec(4,v);

4	2	5	1
2	4	5	1
2	4	5	1
2	4	1	5

2º maior elemento
bolha_rec(3,v);

2	4	1	5
2	4	1	5
2	1	4	5

3º maior elemento
bolha_rec(2,v);

2	1	4	5
1	2	4	5

0 1 2 3

Ordenação Bolha

- Algoritmo genérico (I):
  - independente dos dados armazenados no vetor
  - usa uma função auxiliar para comparar elementos

```
/* Função auxiliar de comparação */
static int compara (int a, int b)
{
   if (a > b)
      return 1;
   else
      return 0;
}
```

Ordenação Bolha

```
/* Ordenação bolha (3a. versão) */
void bolha (int n, int* v)
{  int fim, i;
   for (fim=n-1; fim>0; fim--) {
       int troca = 0;
       for (i=0; i<fim; i++)
           if (compara(v[i],v[i+1])) {
               int temp = v[i];     /* troca */
               v[i] = v[i+1];
               v[i+1] = temp;
               troca = 1;
          }
       if (troca == 0)          /* não houve troca */
          return;
   }
}
```

Ordenação Por Seleção

- Ordenação por seleção (“*selection sort”):*
  - consiste em uma seleção sucessiva do menor/ maior valor contido na parte ainda desordenada dos elementos

  – A cada iteração, encontra-se a posição do maior valor na parte não ordenada do vetor, colocando-o na posição final da parte não ordenada do vetor.

```
Enquanto há elementos no segmento desordenado do vetor

    Encontra o maior valor do segmento desordenado

    Troca  o elemento final do segmento  desordenado  c/o maior

    “Move” a posição final do  segmento desordenado para o início
          do segmento ordenado

fim_enquanto
```

Ordenação Por Seleção

```
void selecao (int *v, int n)
{
      int fim, i, maior;
      int aux;
      for (fim=n-1; fim>0; fim--)
      {
            maior = 0;
            for(i=1;i<=fim;i++)
                 if (v[i]>v[maior])
                       maior =i;
            aux=vet[fim];
            vet[fim]=vet[maior];
            vet[maior]=aux;
      }
}
```

MAIOR=[3]

50	70	10	80	20	40
0	1	2	3	4	5

TROCA COM [5]

50	70	10	40	20	80
0	1	2	3	4	5

MAIOR=[1]

50	70	10	40	20	80
0	1	2	3	4	5

TROCA COM [4]

50	20	10	40	70	80
0	1	2	3	4	5

MAIOR=[0]

50	20	10	40	70	80
0	1	2	3	4	5

TROCA COM [3]

40	20	10	50	70	80
0	1	2	3	4	5

MAIOR=[0]

40	20	10	50	70	80
0	1	2	3	4	5

TROCA COM [2]

10	20	40	50	70	80
0	1	2	3	4	5

MAIOR=[1]

10	20	40	50	70	80
0	1	2	3	4	5

TROCA COM [1]

10	20	40	50	70	80
0	1	2	3	4	5

Resumo

- Bubble sort
  - quando dois elementos estão fora de ordem,
    troque-os de posição até que o i-ésimo elemento de maior valor do vetor seja levado para as posições finais do vetor
  - continue o processo até que todo o vetor esteja ordenado
- Selection sort
  - A cada iteração, encontra-se a posição do maior valor na parte não ordenada do vetor, colocando-o na posição final da parte não ordenada do vetor.

Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)

Capítulo 16 – Ordenação