Programação 2

7 – Ordenação de Vetores

Tópicos Principais

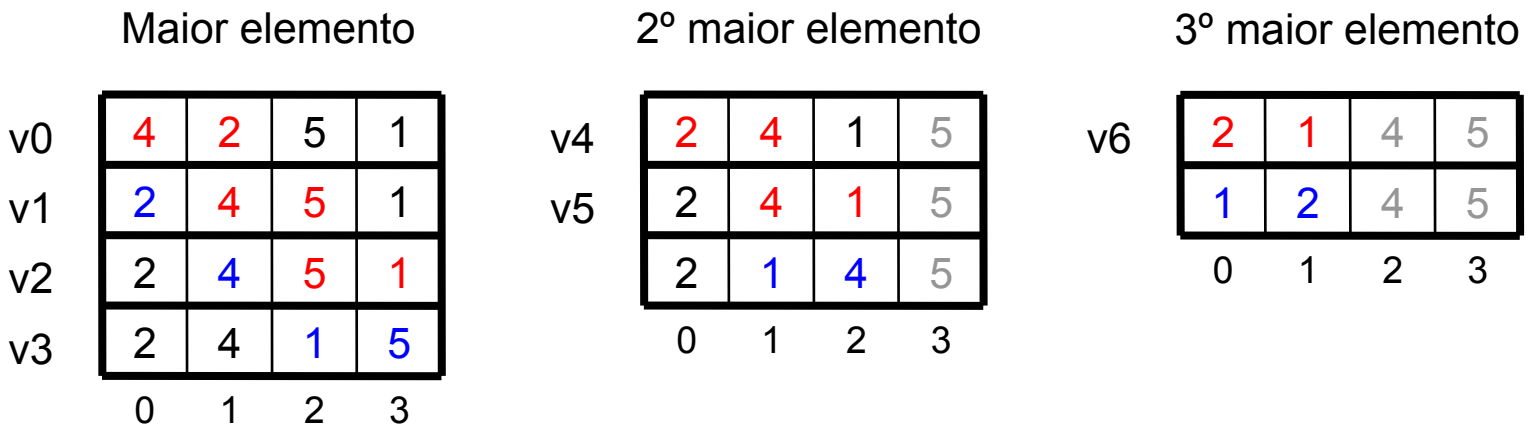
- Introdução
- Ordenação bolha (*bubble sort*)
- Ordenação rápida (*quick sort*)
  - Versão iterativa
  - Versão recursiva

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;
      }
}
```

Ordenação Bolha

- Implementação Iterativa (II):

```
/* Ordenação bolha (2a. versão) */
void bolha (int n, int* v)
{  int i, fim;
   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: $(n-1) + (n-2) + ... + 2 + 1 = (n-1)n / 2 = \mathbf{(n^2 - n) / 2}$
    - algoritmo de ordem quadrática: $\boldsymbol{O(n^2)}$

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 e n>1 */
      bolha_rec(n-1,v);
}
```

Ordenação Rápida

- Ordenação rápida ("*quick sort*"):
  - escolha um elemento arbitrário x, o *pivô*
  - rearrume o vetor de tal forma que x fique na posição correta v[i]
    - x deve ocupar a posição i do vetor sse
      todos os elementos v[0], … v[i-1] são menores que x e
      todos os elementos v[i+1], …, v[n-1] são maiores que x
  - chame recursivamente o algoritmo para ordenar os (sub-)vetores v[0], … v[i-1] e v[i+1], …, v[n-1]
  - continue até que os vetores que devem ser ordenados tenham 0 ou 1 elemento

Ordenação Rápida

- Esforço computacional:
  - melhor caso:
    - pivô representa o valor mediano do conjunto dos elementos do vetor
    - após mover o pivô para sua posição, restarão dois sub-vetores para serem ordenados, ambos com o número de elementos reduzido à metade, em relação ao vetor original
    - algoritmo é $O(n \log(n))$
  - pior caso:
    - pivô é o maior elemento e algoritmo recai em ordenação bolha
  - caso médio:
    - algoritmo é $O(n \log(n))$

Ordenação Rápida

- Rearrumação do vetor para o pivô de x=v[0]:
  - do início para o final, compare x com v[1], v[2], …
    até encontrar v[a]>x
  - do final para o início, compare x com v[n-1], v[n-2], …
    até encontrar v[b]<=x
  - troque v[a] e v[b]
  - continue para o final a partir de v[a+1] e
    para o início a partir de v[b-1]
  - termine quando os pontos de busca se encontram (b<a)
  - a posição correta de x=v[0] é a posição b e v[0] e v[b] são trocados

25 48 37 12 57 86 33 92
25 48 37 12 57 86 33 92
25 48 37 12 57 86 33 92 v[1]>25, a=1
25 48 37 12 57 86 33 92
25 48 37 12 57 86 33 92
25 48 37 12 57 86 33 92
25 48 37 12 57 86 33 92
25 48 37 12 57 86 33 92 v[3]<25, b=3
25 12 37 48 57 86 33 92 troca v[1] com v[3]
25 12 37 48 57 86 33 92 a=b=2

25 12 37 48 57 86 33 92 a=2, pois todos v[2]>25
25 12 37 48 57 86 33 92 b=1, pois v[1]<25 (índices cruzaram)
12 25 37 48 57 86 33 92

12 25 37 48 57 86 33 92
12 25 37 48 57 86 33 92, a=1, no vetor que começa em 37
12 25 37 48 57 86 33 92, b=4, no vetor que começa em 37
12 25 37 33 57 86 48 92, faz a troca, a=2
12 25 37 33 57 86 48 92, b=1 (b<a)
12 25 33 37 57 86 48 92, troca v[0] por v[b]

Ordenação Rápida

- vetor inteiro de v[0] a v[7]

  (0-7) 25 48 37 12 57 86 33 92

- determine a posição correta de x=v[0]=25
  - de a=1 para o fim: 48>25 (a=1)
  - de b=7 para o início: 25<92, 25<33, 25<86, 25<57 e 12<=25 (b=3)

  (0-7) 25 48 37 12 57 86 33 92

  a ↑ *b* ↑

- troque v[a]=48 e v[b]=12, incrementando a e decrementando b
- nova configuração do vetor:

  (0-7) 25 12 37 48 57 86 33 92

  a,b ↑

Ordenação Rápida

- configuração atual do vetor:

  (0-7) 25 12 37 48 57 86 33 92

  a,b ↑

- determine a posição correta de x=v[0]=25
  - de a=2 para o final: 37>25 (a=2)
  - de b=2 para o início: 37>25 e 12<=25 (b=1)
- os índices a e b se cruzaram, com b<a

  (0-7) 25 12 37 48 57 86 33 92

  b ↑ *a* ↑
  - todos os elementos de 37 (inclusive) para o final são maiores que 25 e
    todos os elementos de 12 (inclusive) para o início são menores que 25 – com exceção de 25
- troque o pivô v[0]=25 com v[b]=12, o último dos valores menores que 25 encontrado
- nova configuração do vetor, com o pivô 25 na posição correta:

  (0-7) 12 25 37 48 57 86 33 92

Ordenação Rápida

- dois vetores menores para ordenar:
  - valores menores que 25:

(0-0) 12

    - vetor já está ordenado pois possui apenas um elemento

  - valores maiores que 25:

(2-7) 37 48 57 86 33 92

    - vetor pode ser ordenado de forma semelhante, com 37 como pivô

```
void rapida (int n, int* v){
   if (n > 1) {
      int x = v[0];
      int a = 1;
      int b = n-1;
      do {
         while (a < n && v[a] <= x) a++;  /* teste a<n */
         while (v[b] > x) b--;            /* nao testa */
         if (a < b) {                  /* faz troca */
            int temp = v[a];
            v[a] = v[b];
            v[b] = temp;
            a++; b--;
         }
      } while (a <= b);
      /* troca pivô */
      v[0] = v[b];
      v[b] = x;

      /* ordena sub-vetores restantes */
      rapida(b,v);
      rapida(n-a,&v[a]);
   }
}
```

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
- Quick sort
  - coloque um elemento arbitrário x, o *pivô,* em sua posição k
  - chame recursivamente o algoritmo para ordenar os (sub-)vetores
    v[0], … v[k-1] e v[k+1], …, v[n-1]
  - continue até que os vetores que devem ser ordenados
    tenham 0 ou 1 elemento

Referências

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

Capítulo 16 – Ordenação