Ordenação: métodos elementares

Aula 5

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

Introdução

- Operação básica em Computação
- Rearranjar, ou permutar, os elementos de um vetor $v[0..n-1]$ de tal forma que se torne crescente.
- Um vetor $v[0..n-1]$ é **crescente** se $v[0] \leqslant v[1] \leqslant \cdots \leqslant v[n-1]$.
- Idéias simples
- Tempo de execução de pior caso quadrático

Introdução

- Operação básica em Computação
- Rearranjar, ou permutar, os elementos de um vetor $v[0..n-1]$ de tal forma que se torne crescente.
- Um vetor $v[0..n-1]$ é **crescente** se $v[0] \leqslant v[1] \leqslant \cdots \leqslant v[n-1]$.
- Idéias simples
- Tempo de execução de pior caso quadrático

Introdução

- Operação básica em Computação
- Rearranjar, ou permutar, os elementos de um vetor $v[0..n-1]$ de tal forma que se torne crescente.
- Um vetor $v[0..n-1]$ é **crescente** se $v[0] \leqslant v[1] \leqslant \cdots \leqslant v[n-1]$.
- Idéias simples
- Tempo de execução de pior caso quadrático

Introdução

- Operação básica em Computação
- Rearranjar, ou permutar, os elementos de um vetor $v[0..n-1]$ de tal forma que se torne crescente.
- Um vetor $v[0..n-1]$ é **crescente** se $v[0] \leqslant v[1] \leqslant \cdots \leqslant v[n-1]$.
- Idéias simples
- Tempo de execução de pior caso quadrático

Introdução

- Operação básica em Computação
- Rearranjar, ou permutar, os elementos de um vetor $v[0..n-1]$ de tal forma que se torne crescente.
- Um vetor $v[0..n-1]$ é **crescente** se $v[0] \leqslant v[1] \leqslant \cdots \leqslant v[n-1]$.
- Idéias simples
- Tempo de execução de pior caso quadrático

Método das trocas sucessivas

- Popularmente conhecido como método da bolha ou *Bubble sort*
- A cada passo, posiciona o maior elemento de um subconjunto de elementos do vetor de entrada na sua localização correta neste vetor

Método das trocas sucessivas

- Popularmente conhecido como método da bolha ou *Bubble sort*
- A cada passo, posiciona o maior elemento de um subconjunto de elementos do vetor de entrada na sua localização correta neste vetor

Método das trocas sucessivas

```
/* Recebe um número inteiro n >= 0 e um vetor v de números inteiros
   com n elementos e rearranja o vetor v de modo que fique crescente */
void trocas_sucessivas(int n, int v[MAX])
{
   int i, j;
   for (i = n - 1; i > 0; i--)
      for (j = 0; j < i; j++)
         if (v[j] > v[j+1])
            troca(&v[j], &v[j+1]);
}
```

Método das trocas sucessivas

- Para entender a função `trocas_sucessivas` basta observar que no início de cada repetição do `for` externo vale que:
  - o vetor $v[0..n-1]$ é uma permutação do vetor original,
  - o vetor $v[i+1..n-1]$ é crescente e
  - $v[j] \leqslant v[i+1]$ para $j = 0, 1, \ldots, i$.
- Tempo $O(n^2)$ no pior caso

Método das trocas sucessivas

- Para entender a função `trocas_sucessivas` basta observar que no início de cada repetição do `for` externo vale que:
  - o vetor $v[0..n-1]$ é uma permutação do vetor original,
  - o vetor $v[i+1..n-1]$ é crescente e
  - $v[j] \leqslant v[i+1]$ para $j = 0, 1, \ldots, i$.
- Tempo $O(n^2)$ no pior caso

Método da seleção

- Também conhecido como *Selection sort*
- Baseado na idéia de escolher um menor elemento do vetor, depois um segundo menor elemento e assim por diante

Método da seleção

- Também conhecido como *Selection sort*
- Baseado na idéia de escolher um menor elemento do vetor, depois um segundo menor elemento e assim por diante

Método da seleção

```
/* Recebe um número inteiro n >= 0 e um vetor v de números inteiros
   com n elementos e rearranja o vetor v de modo que fique crescente */
void selecao(int n, int v[MAX])
{
   int i, j, min;
   for (i = 0; i < n - 1; i++) {
      min = i;
      for (j = i+1; j < n; j++)
         if (v[j] < v[min])
            min = j;
      troca(&v[i], &v[min]);
   }
}
```

Método da seleção

- Para entender como e por que a função `selecao` funciona, basta observar que no início de cada repetição do `for` externo valem os seguintes invariantes:
  - o vetor $v[0..n-1]$ é uma permutação do vetor original,
  - o vetor $v[0..i-1]$ está em ordem crescente e
  - $v[i-1] \leqslant v[j]$ para $j = i, i+1, \ldots, n-1$.
- Tempo $O(n^2)$ no pior caso

Método da seleção

- Para entender como e por que a função `selecao` funciona, basta observar que no início de cada repetição do `for` externo valem os seguintes invariantes:
  - o vetor $v[0..n-1]$ é uma permutação do vetor original,
  - o vetor $v[0..i-1]$ está em ordem crescente e
  - $v[i-1] \leqslant v[j]$ para $j = i, i+1, \ldots, n-1$.
- Tempo $O(n^2)$ no pior caso

Método da inserção

- Também conhecido como *Insertion sort*
- Método muito popular
- É frequentemente usado quando alguém joga baralho e quer manter as cartas de sua mão em ordem

Método da inserção

- Também conhecido como *Insertion sort*
- Método muito popular
- É frequentemente usado quando alguém joga baralho e quer manter as cartas de sua mão em ordem

Método da inserção

- Também conhecido como *Insertion sort*
- Método muito popular
- É frequentemente usado quando alguém joga baralho e quer manter as cartas de sua mão em ordem

Método da inserção

```
/* Recebe um número inteiro n >= 0 e um vetor v de números inteiros
   com n elementos e rearranja o vetor v de modo que fique crescente */
void insercao(int n, int v[MAX])
{
   int i, j, x;
   for (i = 1; i < n; i++) {
      x = v[i];
      for (j = i - 1; j >= 0 && v[j] > x; j--)
         v[j+1] = v[j];
      v[j+1] = x;
   }
}
```

Método da inserção

- Para entender a função `insercao` basta observar que no início de cada repetição do `for` externo, valem os seguintes invariantes:
  - o vetor $v[0..n-1]$ é uma permutação do vetor original e
  - o vetor $v[0..i-1]$ é crescente.
- Tempo $O(n^2)$ no pior caso

Método da inserção

- Para entender a função `insercao` basta observar que no início de cada repetição do `for` externo, valem os seguintes invariantes:
  - o vetor $v[0..n-1]$ é uma permutação do vetor original e
  - o vetor $v[0..i-1]$ é crescente.
- Tempo $O(n^2)$ no pior caso

Exercícios

1. Que acontece se trocarmos a realação `i > 0` pela relação `i >= 0` no código da função `trocas_sucessivas`? Que acontece se trocarmos `j < i` por `j <= i`?
2. Troque a relação `v[j] > v[j + 1]` pela relação `v[j] >= v[j + 1]` no código da função `trocas_sucessivas`. A nova função continua produzindo uma ordenação crescente de $v[0..n-1]$?
3. Escreva uma versão recursiva do método de ordenação por trocas sucessivas.

Exercícios

4. Que acontece se trocarmos `i = 0` por `i = 1` no código da função `selecao`? Que acontece se trocarmos `i < n-1` por `i < n`?
5. Troque `v[j] < v[min]` por `v[j] <= v[min]` no código da função `selecao`. A nova função continua produzindo uma ordenação crescente de $v[0..n-1]$?
6. Escreva uma versão recursiva do método de ordenação por seleção.
7. No código da função `insercao`, troque `v[j] > x` por `v[j] >= x`. A nova função continua produzindo uma ordenação crescente de $v[0..n-1]$?

Exercícios

8. No código da função `insercao`, que acontece se trocarmos `i = 1` por `i = 0`? Que acontece se trocarmos `v[j+1] = x` por `v[j] = x`?
9. Escreva uma versão recursiva do método de ordenação por inserção.
10. Escreva uma função que rearranje um vetor $v[0..n-1]$ de modo que ele fique em ordem estritamente crescente.
11. Escreva uma função que permute os elementos de um vetor $v[0..n-1]$ de modo que eles fiquem em ordem decrescente.