Algoritmos de ordenação elementares

Universidade Federal do Ceará - Campus de Crateús

Roberto Cabral
rbcabral@crateus.ufc.br

09 de Outubro de 2017

Estrutura de Dados

Introdução

- Colocar um vetor numérico em ordem crescente ou decrescente é o primeiro passo na solução de muitos problemas práticos.

Introdução

- Colocar um vetor numérico em ordem crescente ou decrescente é o primeiro passo na solução de muitos problemas práticos.
- Um vetor pode ser ordenado de muitas maneiras diferentes: algumas elementares, outras mais sofisticadas e eficientes.

Introdução

- Colocar um vetor numérico em ordem crescente ou decrescente é o primeiro passo na solução de muitos problemas práticos.
- Um vetor pode ser ordenado de muitas maneiras diferentes: algumas elementares, outras mais sofisticadas e eficientes.
- Pode-se usar basicamente duas estratégias para ordenar os dados:

Introdução

- Colocar um vetor numérico em ordem crescente ou decrescente é o primeiro passo na solução de muitos problemas práticos.
- Um vetor pode ser ordenado de muitas maneiras diferentes: algumas elementares, outras mais sofisticadas e eficientes.
- Pode-se usar basicamente duas estratégias para ordenar os dados:
    - inserir os dados na estrutura respeitando sua ordem.

Introdução

- Colocar um vetor numérico em ordem crescente ou decrescente é o primeiro passo na solução de muitos problemas práticos.
- Um vetor pode ser ordenado de muitas maneiras diferentes: algumas elementares, outras mais sofisticadas e eficientes.
- Pode-se usar basicamente duas estratégias para ordenar os dados:
    - inserir os dados na estrutura respeitando sua ordem.
    - a partir de um conjunto de dados já criado, aplicar um algoritmo para ordenar seus elementos.

O problema da ordenação

- Um vetor $v[0 \dots n-1]$ é crescente se $v[0] \leq v[1] \leq \dots \leq v[n-1]$. O problema da ordenação de um vetor consiste no seguinte:

O problema da ordenação

- Um vetor $v[0 \ldots n-1]$ é crescente se $v[0] \leq v[1] \leq \ldots \leq v[n-1]$. O problema da ordenação de um vetor consiste no seguinte:

Problema

Rearranjar (ou seja, permutar) os elementos de um vetor $v[0 \ldots n-1]$ de tal modo que ele se torne crescente.

Pergunta

Como verificar se um dado vetor $v[0 \ldots n-1]$ é crescente???

Algoritmo de Inserção (*insertion sort*)

- O algoritmo de ordenação por inserção é muito popular; ele é frequentemente usado para colocar em ordem um baralho de cartas.

```
/*Esta função rearranja o vetor v[0..n-1] em ordem crescente*/
void insercao(int n, int *v){
  int i, j, x;
  for(j=1;j<n;j++){
    x = v[j];
    for(i=j-1;i>=0 && v[i] > x; i--){
      v[i+1] = v[i];
    }
    v[i+1] = x;
  }
}
```

Algoritmo de Inserção

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:

Algoritmo de Inserção

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. o vetor $v[0 \ldots n-1]$ é uma permutação do vetor original e

Algoritmo de Inserção

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. o vetor $v[0 \ldots n-1]$ é uma permutação do vetor original e
  2. o vetor $v[0 \ldots j-1]$ é crescente.

Algoritmo de Inserção

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. o vetor $v[0 \dots n-1]$ é uma permutação do vetor original e
  2. o vetor $v[0 \dots j-1]$ é crescente.
- Estas propriedades invariantes são trivialmente verdadeiras no início da primeira iteração, quando j vale 1, e permanecem verdadeiras no início das iterações subsequentes.

Algoritmo de Inserção

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. o vetor $v[0 \dots n-1]$ é uma permutação do vetor original e
  2. o vetor $v[0 \dots j-1]$ é crescente.
- Estas propriedades invariantes são trivialmente verdadeiras no início da primeira iteração, quando j vale 1, e permanecem verdadeiras no início das iterações subsequentes.
- No início da última iteração, j vale n e portanto o vetor $v[0 \dots n-1]$ está na ordem desejada.

Desempenho do algoritmo

- O consumo de tempo da função `insercao` é proporcional ao número de execuções da comparação "$v[i] > x$.

Desempenho do algoritmo

- O consumo de tempo da função `insercao` é proporcional ao número de execuções da comparação "$v[i] > x$.
- Para cada valor de j, a variável i assume no máximo j valores.

Desempenho do algoritmo

- O consumo de tempo da função `insercao` é proporcional ao número de execuções da comparação “$v[i] > x$.
- Para cada valor de j, a variável i assume no máximo j valores.
- Como j varia de 1 a n, o número de execuções da comparação "$v[i] > x$ é igual a $\Sigma_{j=1}^{n-1}$ no pior caso.

Desempenho do algoritmo

- O consumo de tempo da função `insercao` é proporcional ao número de execuções da comparação "$v[i] > x$.
- Para cada valor de j, a variável i assume no máximo j valores.
- Como j varia de 1 a n, o número de execuções da comparação "$v[i] > x$ é igual a $\Sigma_{j=1}^{n-1}$ no pior caso.
- A soma vale $n(n-1)/2$ e este número é essencialmente igual a $n^2/2$ quando n é grande.

Desempenho do algoritmo

- O consumo de tempo da função `insercao` é proporcional ao número de execuções da comparação "$v[i] > x$.
- Para cada valor de j, a variável i assume no máximo j valores.
- Como j varia de 1 a n, o número de execuções da comparação "$v[i] > x$ é igual a $\Sigma_{j=1}^{n-1}$ no pior caso.
- A soma vale $n(n-1)/2$ e este número é essencialmente igual a $n^2/2$ quando n é grande.
- Pode-se dizer portanto que essa função é O(n^2).

Desempenho do algoritmo

- O consumo de tempo da função `insercao` é proporcional ao número de execuções da comparação "$v[i] > x$.
- Para cada valor de j, a variável i assume no máximo j valores.
- Como j varia de 1 a n, o número de execuções da comparação "$v[i] > x$ é igual a $\Sigma_{j=1}^{n-1}$ no pior caso.
- A soma vale $n(n-1)/2$ e este número é essencialmente igual a $n^2/2$ quando n é grande.
- Pode-se dizer portanto que essa função é O(n^2).
- **Qual a complexidade de melhor caso?**

Desempenho do algoritmo

- O consumo de tempo da função `insercao` é proporcional ao número de execuções da comparação "$v[i] > x$.
- Para cada valor de j, a variável i assume no máximo j valores.
- Como j varia de 1 a n, o número de execuções da comparação "$v[i] > x$ é igual a $\Sigma_{j=1}^{n-1}$ no pior caso.
- A soma vale $n(n-1)/2$ e este número é essencialmente igual a $n^2/2$ quando n é grande.
- Pode-se dizer portanto que essa função é O(n^2).
- **Qual a complexidade de melhor caso?**
- **Qual a complexidade em termos de espaços?**

Problemas

Problema 2

Na função `insercao`, troque a comparação "$v[i] > x$" por "$v[i] >= x$". A nova função continua correta?

Problema 3

Que acontece se trocarmos "for (j = 1" por "for (j = 0" no código da função `insercao`? Que acontece se trocarmos "$v[i+1] = x$" por "$v[i] = x$"?

Problema 4

Escreva uma versão recursiva do algoritmo de ordenação por inserção.

Algoritmo de Seleção

- O algoritmo de ordenação por seleção é baseado na ideia de escolher o menor elemento do vetor, depois o segundo menor, e assim por diante.

```c
/*rearranja o vetor v[0..n-1] em ordem crescente*/
void selecao(int n, int *v){
  int i, j, min, x;
  for(i=0;i<n-1;i++){
    min = i;
    for(j=i+1;j<n; j++){
      if(v[j] < v[min]) min =j;
    }
    x = v[i];
    v[i] = v[min];
    v[min] = x;
  }
}
```

Algoritmo de Seleção (*selection sort*)

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:

Algoritmo de Seleção (*selection sort*)

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. $v[0 \dots n-1]$ é uma permutação do vetor original

Algoritmo de Seleção (*selection sort*)

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. $v[0 \ldots n-1]$ é uma permutação do vetor original
  2. $v[0 \ldots i-1]$ está em ordem crescente e

Algoritmo de Seleção (*selection sort*)

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. $v[0 \ldots n-1]$ é uma permutação do vetor original
  2. $v[0 \ldots i-1]$ está em ordem crescente e
  3. $v[i-1] \leq v[j]$ para $j = i, i+1, \ldots, n-1$.

Algoritmo de Seleção (*selection sort*)

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. $v[0 \ldots n-1]$ é uma permutação do vetor original
  2. $v[0 \ldots i-1]$ está em ordem crescente e
  3. $v[i-1] \leq v[j]$ para $j = i, i+1, \ldots, n-1$.
- O invariante 3 diz que $v[0 \ldots i-1]$ contém todos os elementos "pequenos" do vetor original e $v[i \ldots n-1]$ contém todos os elementos "grandes"

Algoritmo de Seleção (*selection sort*)

- Para entender o algoritmo, basta observar que no inicio de cada repetição for `for` externo:
  1. $v[0 \dots n-1]$ é uma permutação do vetor original
  2. $v[0 \dots i-1]$ está em ordem crescente e
  3. $v[i-1] \leq v[j]$ para $j = i, i+1, \dots, n-1$.
- O invariante 3 diz que $v[0 \dots i-1]$ contém todos os elementos "pequenos" do vetor original e $v[i \dots n-1]$ contém todos os elementos "grandes"
- Os três invariantes garantem que no início de cada iteração os elementos $v[0], \dots, v[i-1]$ já estão em suas posições definidas.

Desempenho do algoritmo

- O algoritmo de ordenação por seleção compara a cada interação um elemento com os outros, visando encontrar o menor.

Desempenho do algoritmo

- O algoritmo de ordenação por seleção compara a cada interação um elemento com os outros, visando encontrar o menor.
- Dessa forma, podemos entender que não existe um melhor caso mesmo que o vetor esteja ordenado ou em ordem inversa serão executados os dois laços do algoritmo, o externo e o interno.

Desempenho do algoritmo

- O algoritmo de ordenação por seleção compara a cada interação um elemento com os outros, visando encontrar o menor.
- Dessa forma, podemos entender que não existe um melhor caso mesmo que o vetor esteja ordenado ou em ordem inversa serão executados os dois laços do algoritmo, o externo e o interno.
- A complexidade deste algoritmo será sempre O(n^2).

Problemas

Problema 5

Na função `selecao`, que acontece se trocarmos "for $(i = 0$" por "for $(i = 1$"? Que acontece se trocarmos "for $(i = 0; i < n - 1$" por "for $(i = 0; i < n''$?

Problema 6

Na função `selecao`, troque a comparação "$v[j] < v[min]$" por "$v[j] <= v[min]$". A nova função continua correta?

Problema 7

Escreva uma versão recursiva do algoritmo de ordenação por seleção.

Ordenação por flutuação (*bubble sort*)

- O algoritmo de ordenação por flutuação é baseado na ideia de escolher o maior elemento do vetor (lembra a forma como as bolhas de um tanque de água procuram seu nível).

```
/*rearranja o vetor v[0..n-1] em ordem crescente*/
void bubble(int n, int *v){
  int i, j, aux,k;
  k=n-1;
  for(i=0;i<n;i++){
    for(j=0;j<k; j++){
      if(v[j] > v[j+1]){
        aux = v[j];
        v[j] = v[j+1];
        v[j+1] = aux;
      }
    }
    k--;
  }
}
```

Ordenação estável

- Um algoritmo de ordenação é estável se não altera a posição relativa de elementos que têm um mesmo valor.

Ordenação estável

- Um algoritmo de ordenação é estável se não altera a posição relativa de elementos que têm um mesmo valor.
- Por exemplo, se o vetor tiver dois elementos de valor 13, um algoritmo de ordenação estável manterá o primeiro 13 antes do segundo.

Ordenação estável

- Um algoritmo de ordenação é estável se não altera a posição relativa de elementos que têm um mesmo valor.
- Por exemplo, se o vetor tiver dois elementos de valor 13, um algoritmo de ordenação estável manterá o primeiro 13 antes do segundo.
- Suponha, por exemplo, que os elementos de um vetor são pares da forma (d, m) que representam datas de um certo ano: a primeira componente representa o dia e a segunda o mês.

Exemplo

- Suponha que o vetor está em ordem descrente das componentes d:

  (1,12), (7,12), (16,3), (25,9), (30,3), (30,6), (31,3).

Exemplo

- Suponha que o vetor está em ordem descrente das componentes d:

  (1,12), (7,12), (16,3), (25,9), (30,3), (30,6), (31,3).

- Agora ordene o vetor pelas componentes m. Se usarmos um algoritmo de ordenação estável, o resultado estará em ordem cronológica:

  (16,3), (30,3), (31,3), (30,6), (25,9), (1,12), (7,12).

Exemplo

- Suponha que o vetor está em ordem descrente das componentes d:

  (1,12), (7,12), (16,3), (25,9), (30,3), (30,6), (31,3).

- Agora ordene o vetor pelas componentes m. Se usarmos um algoritmo de ordenação estável, o resultado estará em ordem cronológica:

  (16,3), (30,3), (31,3), (30,6), (25,9), (1,12), (7,12).

- Se o algoritmo de ordenação não for estável, o resultado pode não ficar em ordem cronológica:

  (30,3), (16,3), (31,3), (30,6), (25,9), (7,12), (1,12).

Algoritmos de ordenação elementares

Universidade Federal do Ceará - Campus de Crateús

Roberto Cabral
rbcabral@crateus.ufc.br

09 de Outubro de 2017

Estrutura de Dados