Ordenação – Parte 1

Algoritmos e Estrutura de Dados

Prof. Emanuel Estrada

Introdução

- **Problema fundamental**: **Permutar** (rearranjar) os elementos de um vetor $v[0..n-1]$ de tal modo que eles fiquem em ordem crescente, isto é, de tal forma que tenhamos $v[0] \leq v[1] \leq \ ... \leq v[n-1]$.
- O processo de organizar os dados em uma determinada ordem é parte importante de muitos métodos e técnicas em computação
  - Uma série de algoritmos de busca, intercalação, utilizam ordenação como parte do processo
  - Aplicações em geometria computacional, banco de dados, entre outras necessitam de listas ordenadas para funcionar

Introdução

- A saída de um algoritmo de ordenação deve satisfazer duas condições:
  1. Estar em uma ordem crescente ou decrescente
  2. Ser uma permutação da entrada

```
Entrada: 6 5 7 1 4 3 2

Saídas possíveis:  1 2 3 4 5 6 7
                   7 6 5 4 3 2 1



Entrada: V U X Z Y

Saídas possíveis:  U V X Y Z
                   Z Y X V U
```

Motivação

- O estudo de algoritmos de ordenação é importante pois, entre outros:
  1. permite compreender e praticar uma série de conceitos importantes:
     - as notações assintóticas,
     - análises de pior, melhor e caso esperado,
     - paradigmas de projetos de algoritmos,
     - compromisso entre uso de espaço e consumo tempo, etc.
  2. novos algoritmos e extensões de algoritmos já existentes continuam sendo propostas.
  3. ordenação é parte de muitos métodos em computação – é preciso conhecer os principais algoritmos e saber escolher qual utilizar.
  4. é possível que um dia alguém lhe pergunte a maneira mais eficiente de ordenar 1 milhão de inteiros de 32 bits.

https://www.youtube.com/watch?v=S1wIkyPW0Ns

Terminologia

- Os números a serem ordenados, em geral, fazem parte de um **registro**.
- Cada registro contém uma **chave**, que é o valor a ser ordenado, e o restante do registro consiste em **dados satélite**.

Terminologia

- **Ordenação interna:** os dados estão na memória principal e todo o processo é feito usando a memória principal.
- **Ordenação externa:** os dados estão em uma memória secundária/auxiliar.

Terminologia

- Um algoritmo de ordenação é **estável** se para todo registro i, j, sendo que $k[i] = k[j]$,
  - Se $k[i]$ precede $k[j]$ no arquivo de entrada, então $k[i]$ precede $k[j]$ no arquivo ordenado

Ou seja, o algoritmo preserva a ordem relativa original dos registros de valor de chave igual.

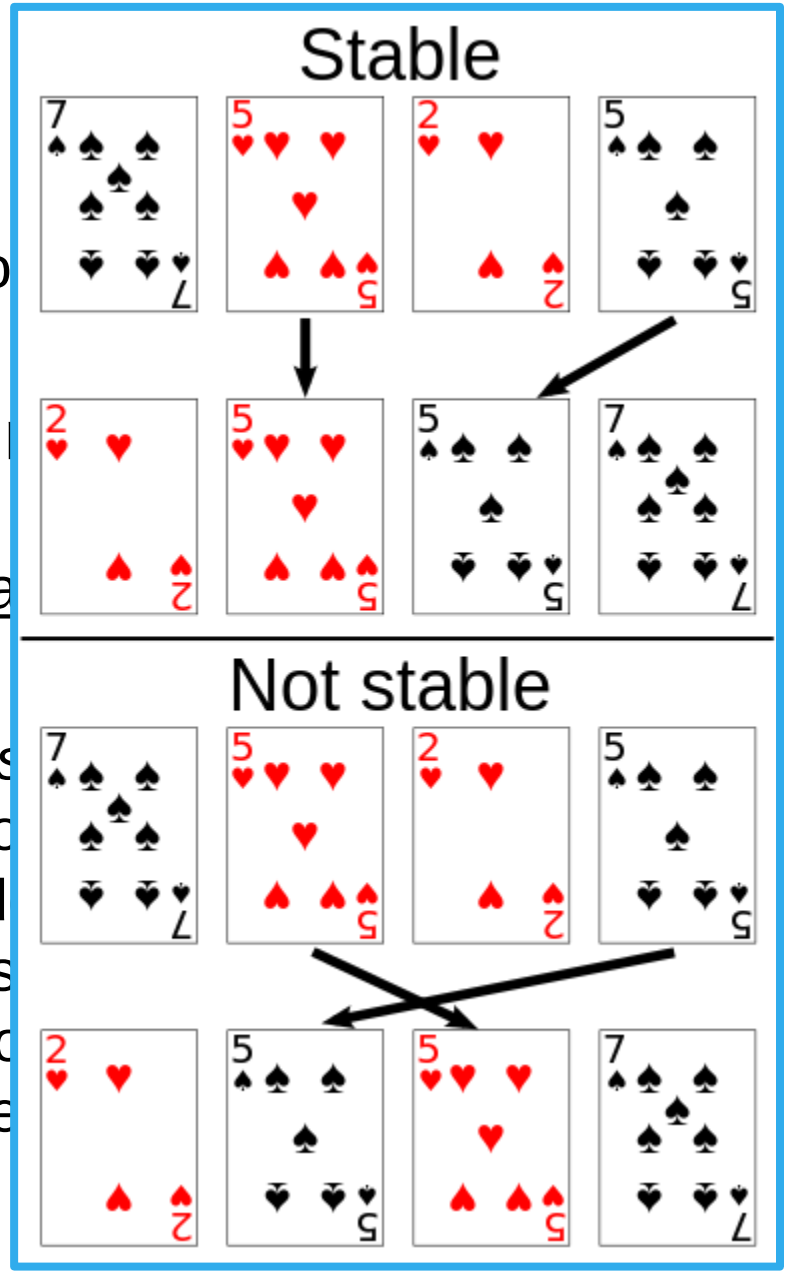
- Por exemplo, se cada elemento de um vetor é um *struct* com dois campos: o primeiro campo contém o nome de uma pessoa e o segundo contém o ano de nascimento. Suponha que o vetor original tem dois João da Silva, primeiro o que nasceu em 1990 e depois o que nasceu em 1995. Se o vetor for ordenado por um algoritmo estável com base no primeiro campo, os dois João da Silva continuarão na mesma ordem relativa: primeiro o de 1990 e depois o de 1995

Terminologia

- Um algoritmo de ordenação é **estável** se para todo $k[i] = k[j]$,
  - Se $k[i]$ precede $k[j]$ no arquivo de entrada, então $k[i]$ ordenado

Ou seja, o algoritmo preserva a ordem relativa origina de chave igual.

- Por exemplo, se cada elemento de um vetor é um s primeiro campo contém o nome de uma pessoa e c de nascimento. Suponha que o vetor original tem d primeiro o que nasceu em 1990 e depois o que nas for ordenado por um algoritmo estável com base nc dois João da Silva continuarão na mesma ordem re e depois o de 1995



Terminologia

- Ordenação sobre registro
- Ordenação sobre endereço

Terminologia

- *Out-of-place*: algoritmos que necessitam espaço auxiliar proporcional a n.
- Ex.: Inversão de um vetor

```
void inverteVetor1(int *A, int n){
    int B[n];

    int i;
    for (i = 0; i < n; i++)
        B[n-i-1] = A[i];
    for (i = 0; i < n; i++)
        A[i]=B[i];
}
```

Terminologia

- *In-place*: algoritmos que necessitam $O(1)$ de espaço extra.
- Ex.: Inversão de um vetor

```
void inverteVetor2(int *A, int i, int f){
    for (i = 0; i <= f; i++) {
        int temp = A[i];
        A[i] = A[f];
        A[f--] = temp;
    }
}
```

Terminologia

- *In-place*: algoritmos que necessitam $O(1)$ de espaço extra.
- Ex.: Inversão de um vetor recursiva

```
void inverteVetor2Rec(int *A, int i, int f){
     int tmp;
     if(i<f){
          tmp = A[i];
          A[i] = A[f];
          A[f] = tmp;
          inverteVetor2Rec(A, i+1, f-1);
     }
}
```

Eficiência

- Complexidade de tempo do algoritmo
- Espaço de memória necessário
- Adequação da simplicidade (e tamanho) do problema com o método utilizado
- **Medida de eficiência**
  - A **contagem de operações** para análise de eficiência é feita pelo número de operações críticas, geralmente:

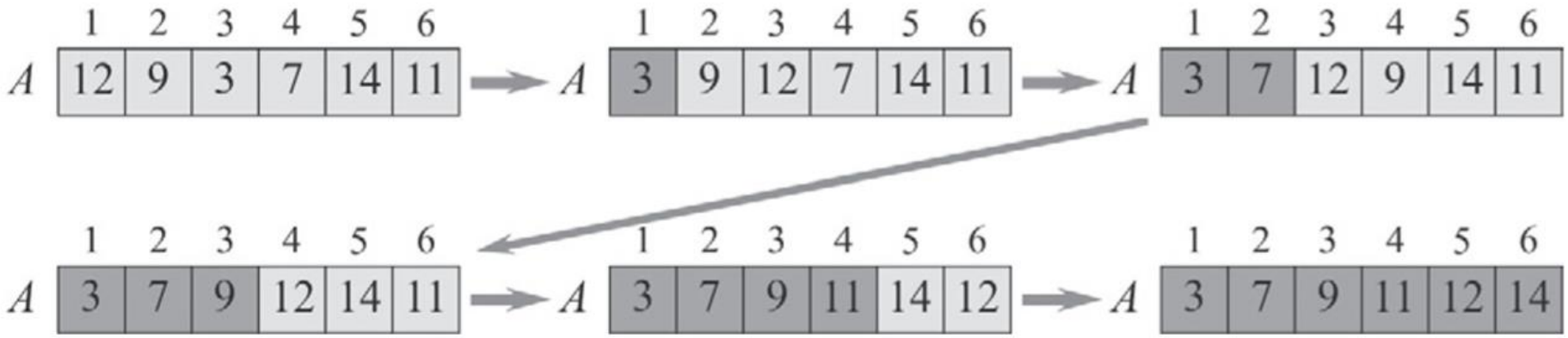
1. Comparação entre chaves
2. Movimento de registros ou ponteiros
3. Troca entre registros

Ordenação Quadrática

- Algoritmos de ordenação $O(n^2)$
- O pior desempenho classifica esses algoritmos
  - Exemplo: Insertion sort, selection sort, bubble sort

Selection Sort (Ordenação por Seleção)

- O algoritmo de ordenação por seleção ordena um arranjo procurando repetidamente o menor elemento da parte não ordenada e colocando-o na início dessa parte.
- Dois sub-arranjos são considerados no arranjo a ser ordenado
  - O sub-arranjo já ordenado.
  - O sub-arranjo restante que não está ordenado.



Selection Sort

```
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;
   }
}
```

Selection Sort - Desempenho

- O algoritmo de seleção tem consumo de tempo , tanto pior caso quanto no melhor caso, sempre proporcional a n^2.

Bubble Sort (ordenação por flutuação)

- **Estratégia**: Percorrer a lista de elementos diversas vezes, a cada passagem fazendo flutuar para o final o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

```
passo 1          passo 2          passo 3
   7 9 6 1          7 6 1 9          6 1 7 9
   7 9 6 1          6 7 1 9          1 6 7 9
   7 6 9 1          6 1 7 9          1 6 7 9
   7 6 1 9
```

Bubble sort explicado com dança cigana:
https://www.youtube.com/watch?v=lyZQPjUT5B4

Bubble Sort (ordenação por flutuação)

```
void bubble( int v[], int n)
{
  int i, j, aux;
  int 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--;
  }
}
```

Bubble Sort - Desempenho

- Tal como o algoritmo de seleção, a implementação do *bubble sort* apresentada tem consumo de tempo , tanto pior caso quanto no melhor caso, sempre proporcional a n^2.

Insertion Sort (Ordenação por Inserção)

- Já visto em aula ☺
- Melhor caso: linear
- Pior caso: quadrático

```
void insertionSort( int A[], int n)
{
     int i, j, x;

     for(j=1;j<n;j++){
          x = A[j];
          i = j-1;
          while((i>=0) && (A[i]>x)){
               A[i+1]=A[i];
               i = i - 1;
          }
          A[i+1]=x;
     }
}
```

Considerações finais

- Importante conhecer o tamanho da entrada e como os registros estão organizados antes de escolher o método
- Em qual caso vocês usaria um algoritmo de ordenação quadrática no lugar de um algoritmo de ordenação de ordem de crescimento menor? Dos algoritmos de ordenação quadrática, qual você escolheria?

Exercícios

1. Considerando o algoritmo de ordenação por inserção:

a) Na função *insertionSort*, troque a comparação $A[i] > x$ por $A[i] >= x$. A nova função continua produzindo uma ordenação crescente de $v[0..n-1]$?

b) O que acontece se trocarmos $for\ (j = 1$ por $for\ (j = 0$ no código da função *insertionSort*?

c) Que acontece se trocarmos $A[i+1] = x$ por $A[i] = x$ no código da função *insertionSort*?

d) Altere o algoritmo de ordenação por inserção para permutar os elementos de um vetor inteiro $A[0..n-1]$ de modo que eles fiquem em ordem decrescente.

e) MOVIMENTAÇÃO DE DADOS: Quantas vezes, no pior caso, o algoritmo de inserção copia um elemento do vetor de um lugar para outro? Quantas vezes isso ocorre no melhor caso?

Exercícios

2. Considerando o algoritmo de ordenação por seleção:

a) Na função *selecao*, o que acontece se trocarmos $for\ (i\ =\ 0$ por $for\ (i\ =\ 1$? O que acontece se trocarmos $for\ (i\ =\ 0;\ i\ <\ n-1$ por $for\ (i\ =\ 0;\ i\ <\ n$?

b) Na função *selecao*, troque a comparação $v[j]\ <\ v[\min]$ por $v[j]\ <=\ v[\min]$. A nova função continua correta?

c) MOVIMENTAÇÃO DE DADOS: Quantas vezes, no pior caso, o algoritmo de seleção copia um elemento do vetor de um lugar para outro? Quantas vezes isso ocorre no melhor caso?

d) Escreva uma função que permute os elementos de um vetor inteiro v[0..n-1] de modo que eles fiquem em ordem decrescente. Inspire-se no algoritmo de seleção.

Exercícios

3. Considerando o algoritmo de ordenação por flutuação
   a) MOVIMENTAÇÃO DE DADOS: Quantas vezes, no pior caso, o algoritmo de flutuação copia um elemento do vetor de um lugar para outro? Quantas vezes isso ocorre no melhor caso?
   b) Escreva uma função que permute os elementos de um vetor inteiro v[0..n-1] de modo que eles fiquem em ordem decrescente. Inspire-se no algoritmo de flutuação.

Exercícios

- Faça a análise do tempo de execução para o pior e melhor caso da seguinte implementação do bubble sort:

```
void bubbleSort(int arr[], int n)
{
  int i, j;
  int swapped;
  for (i = 0; i < n - 1; i++){
    swapped = 0;
    for (j = 0; j < n - i - 1; j++){
      if (arr[j] > arr[j + 1]){
        swap(&arr[j], &arr[j + 1]);
        swapped = 1;
      }
    }
    if (swapped == 0)
      break;
  }
}
```

Exercícios

4. O algoritmo de seleção é estável?
5. O algoritmo de inserção é estável?
6. O algoritmo de flutuação é estável?

Referências

- Cormen, “Introduction to Algorithms”
- Slides do prof. Moacir Ponti Jr. - www.icmc.usp.br/~moacir
- Slides do prof. Maura Barbat
- Material do prof. Paulo Feofiloff - http://www.ime.usp.br/~pf/analise_de_algoritmos/