Algoritmos de Ordenação (Parte 1)

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

Sumário

- Ordenação por Troca
    - Bubble sort
    - Quicksort

Introdução

- Ordenação (ou classificação): Tornar mais simples, rápida e viável a recuperação de uma determinada informação, em um conjunto grande de informações

- Terminologia básica:
  - Arquivo de tamanho n é uma sequência de n itens $(X_1, X_2, ..., X_n)$
  - O i-ésimo componente do arquivo é chamado de item
  - Uma chave é associada a cada registro
  - Ordenação pela chave

Introdução

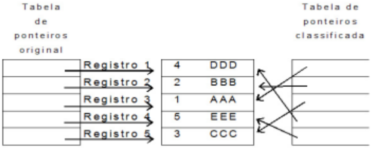
- Terminologia básica:
  - Ordenação interna
  - Ordenação externa
  - Ordenação estável
  - Ordenação pode ocorrer sobre os próprios registros ou sobre uma tabela auxiliar de ponteiros

Registro 1	4	DDD
Registro 2	2	BBB
Registro 3	1	AAA
Registro 4	5	EEE
Registro 5	3	CCC

Arquivo Original

1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Arquivo classificado



Introdução

- Devido à relação entre a ordenação e a busca, surge uma pergunta: o aquivo deve ser classificado ou não?

- Eficiência de métodos de ordenação
  - Tempo para a execução do método

  - Espaço de memória necessário

- Normalmente o tempo gasto é medido pelo número de operações críticas
  - Comparação de chaves

  - Movimento de registros

Introdução

- O resultado da mensuração do desempenho é uma fórmula em função de n
- Existem vários tipos de métodos de ordenação (*e.g.*, troca, seleção, inserção)
- Não existe um método de ordenação considerado superior aos outros

Sumário

Ordenação por Troca

Ordenação por Troca

- Em cada comparação pode haver troca de posições entre itens comparados

- Exemplos de algoritmos de ordenação por troca:
  - *Bubble sort* (Bolha)

  - *Quicksort*

- Percorre o arquivo sequencialmente várias vezes, na qual cada elemento é comparado com o seu sucessor
- Fácil compreensão e implementação
- Um dos métodos de ordenação menos eficiente
- O método é estável

- Implementação

```
void bubblesort(int v[], int n){
  int i, j, x;

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

- Implementação (com uma melhoria)

```
void bubblesort(int v[], int n){
  int i, j, x, troca = 1;

  for (i = 0; (i < n - 1) && troca; i++){
    troca = 0;

    for (j = 0; j < n - i - 1; j++)
     if (v[j] > v[j + 1]){
       x = v[j];
       v[j] = v[j + 1];
       v[j + 1] = x;
       troca = 1;
     }
  }
}
```

- Conjunto completo de iterações *i* (em verde, estão os elementos que foram ordenados em cada iteração)

iteração	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
-	25	57	48	37	12	92	86	33
0	25	48	37	12	57	86	33	92
1	25	37	12	48	57	33	86	92
2	25	12	37	48	33	57	86	92
3	12	25	37	33	48	57	86	92
4	12	25	33	37	48	57	86	92
5	12	25	33	37	48	57	86	92

- Na quinta iteração, os elementos estão completamente ordenados

- Eficiência do *bubble sort* com melhorias
  - O número de trocas não é maior que o número de comparações
  - Vantajoso em conjuntos pré-ordenados
  - Redução do número de comparações em cada passagem:
    - 1a. passagem: $n - 1$ comparações
    - 2a. passagem: $n - 2$ comparações
    - ...
    - (n - 1)a. passagem: 1 comparação
  - Total: $\frac{(n^2+n)}{2}$
  - $O(n^2)$

- Links interessantes:
  - Dança húngara: https://www.youtube.com/watch?v=lyZQPjUT5B4
  - Simulador gráfico do *bubble sort*: https://visualgo.net/bn/sorting

- O *quicksort* adota a estratégia de divisão e conquista
  - Divisão: particionar o arranjo $X[p...q]$ em dois sub-arranjos $X[p...r-1]$ e $X[r+1...q]$, tais que $X[p...r-1] \leq X[r] \leq X[r+1...q]$
  - Conquista: ordenar os dois sub-arranjos $X[p...r-1]$ e $X[r+1...q]$ por chamadas recursivas do *quicksort*

- Procedimento $quicksort(X, p, q)$
  1. definir o pivô r e as posições $i = p$ e $j = q$
  2. enquanto $i \leq j$, trocar de posição os elementos maiores (lado esquerdo do arranjo) com os itens menores (lado direito) que o pivô
  3. $quicksort(X, p, j)$
  4. $quicksort(X, i, q)$

- Implementação

```
void quicksort(int x[], int esq, int dir){
   int i = esq, j = dir, pivo = x[(i + j) / 2], aux;

   do{
      while (x[i] < pivo)
        i++;

      while (x[j] > pivo)
        j--;

      if (i <= j){
        aux = x[i];
        x[i] = x[j];
        x[j] = aux;
        i++;
        j--;
      }
   }while (i <= j);

   if (j > esq)
      quicksort(x, esq, j);

   if (i < dir)
      quicksort(x, i, dir);
}
```

- Exemplo

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
25	57	48	37	12	92	86	33

- Para $esq = 0$, $dir = 7$ e $pivo = X[(0+7)/2] = X[3] = 37$, temos

i	j	pivo	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	
0	7	37	25	57	48	37	12	92	86	33	
1	7	37	25	57	48	37	12	92	86	33	Troca
1	7	37	25	33	48	37	12	92	86	57	Após troca
2	6	37	25	33	48	37	12	92	86	57	
2	5	37	25	33	48	37	12	92	86	57	
2	4	37	25	33	48	37	12	92	86	57	Troca
2	4	37	25	33	12	37	48	92	86	57	Após troca
3	3	37	25	33	12	37	48	92	86	57	
4	2	37	25	33	12	37	48	92	86	57	

- Após a execução, fazemos mais duas chamadas recursivas
  - quicksort(v, esq, j) => quicksort(x, 0, 2)
  - quicksort(v, i, dir) => quicksort(x, 4, 7)

- Exemplo (continuação: quicksort(x, 0, 2))
  - Para $esq = 0$, $dir = 2$ e $pivo = X[(0+2)/2] = X[1] = 33$, temos

i	j	pivo	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	
0	2	33	25	33	12	37	48	92	86	57	
1	2	33	25	33	12	37	48	92	86	57	Troca
1	2	33	25	12	33	37	48	92	86	57	Após Troca
2	1	33	25	12	33	37	48	92	86	57	

- Após a execução, fazemos mais uma chamada recursiva (uma chamada não é realizada porque i é igual a dir)
  - quicksort(v, esq, j) => quicksort(x, 0, 1)

- Exemplo (continuação: quicksort(x, 4, 7)), supondo que quicksort(x, 0, 1) foi executada
  - Para $esq = 4$, $dir = 7$ e $pivo = X[(4+7)/2] = X[5] = 92$, temos

i	j	pivo	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]	
4	7	92	12	25	33	37	48	92	86	57	
5	7	92	12	25	33	37	48	92	86	57	Troca
5	7	92	12	25	33	37	48	57	86	92	Após Troca
6	6	92	12	25	33	37	48	57	86	92	
7	5	92	12	25	33	37	48	57	86	92	

- Após a execução, fazemos mais uma chamada recursiva (uma chamada não é realizada porque i é igual a dir)
  - quicksort(v, esq, j) => quicksort(x, 4, 5)

- Mesmo que o tempo de execução no pior caso seja $O(n^2)$, em média, o tempo é de $O(n \log_2 n)$
- No melhor caso, o algoritmo tem custo de tempo de $O(n \log_2 n)$
- O método não é estável

- Links interessantes:
  - Dança húngara:
    https://www.youtube.com/watch?v=ywWBy6J5gz8
  - Simulador gráfico do *quicksort*:
    https://visualgo.net/bn/sorting

Referências I

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.
Introduction to Algorithms.
Third edition, The MIT Press, 2009.

Horowitz, E., Sahni, S. Rajasekaran, S.
Computer Algorithms.
Computer Science Press, 1998.

Rosa, J. L. G.
Métodos de Ordenação. SCE-181 – Introdução à Ciência da Computação II.
Slides. Ciência de Computação. ICMC/USP, 2018.

Ziviani, N.
Projeto de Algoritmos - com implementações em Java e C++.
Thomson, 2007.