Métodos de Ordenação

ICC2

Prof. Thiago A. S. Pardo

Baseado no material do Prof. Rudinei Goularte

Quick-sort

- Idéia básica: dividir para conquistar
  - Dividir o vetor em dois vetores menores que serão ordenados independentemente e combinados para produzir o resultado final

Quick-sort: exemplo

25 57 35 37 12 86 92 33

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 ponteiros inicializados

i → ← j

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

```
25 57 35 37 12 86 92 33        ponteiros inicializados
 i                      j
 →                      ←
25 57 35 37 12 86 92 33        procura-se i>=pivô
    i                   j
```

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

```
25 57 35 37 12 86 92 33        ponteiros inicializados
 i                      j
 →                      ←
25 57 35 37 12 86 92 33        procura-se i>=pivô
    i                   j
25 57 35 37 12 86 92 33        procura-se j<=pivô
    i                   j
```

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

```
25 57 35 37 12 86 92 33        ponteiros inicializados
 i                      j
 →                     ←

25 57 35 37 12 86 92 33        procura-se i>=pivô
    i                   j

25 57 35 37 12 86 92 33        procura-se j<=pivô
    i                   j

25 33 35 37 12 86 92 57        ***troca***
    i                   j
```

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

```
25 57 35 37 12 86 92 33        ponteiros inicializados
 i                      j
 →                      ←

25 57 35 37 12 86 92 33        procura-se i>=pivô
    i                   j

25 57 35 37 12 86 92 33        procura-se j<=pivô
    i                   j

25 33 35 37 12 86 92 57        ***troca***
    i                   j

25 33 35 37 12 86 92 57        procura-se i>=pivô
          i             j
```

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

```
25 57 35 37 12 86 92 33        ponteiros inicializados
 i                      j
 →                     ←

25 57 35 37 12 86 92 33        procura-se i>=pivô
    i                   j

25 57 35 37 12 86 92 33        procura-se j<=pivô
    i                   j

25 33 35 37 12 86 92 57        ***troca***
    i                   j

25 33 35 37 12 86 92 57        procura-se i>=pivô
          i             j

25 33 35 37 12 86 92 57        procura-se j<=pivô
          i  j
```

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25	57	35	37	12	86	92	33	ponteiros inicializados
i →							← j	
25	57	35	37	12	86	92	33	procura-se i>=pivô
	i						j	
25	57	35	37	12	86	92	33	procura-se j<=pivô
	i						j	
25	33	35	37	12	86	92	57	***troca***
	i						j	
25	33	35	37	12	86	92	57	procura-se i>=pivô
			i				j	
25	33	35	37	12	86	92	57	procura-se j<=pivô
			i	j				
25	33	35	12	37	86	92	57	***troca***
			i	j				

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57   procura-se i>=pivô
ij

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57 procura-se i>=pivô
ij

25 33 35 12 37 86 92 57 procura-se j<=pivô
ij
→ como i e j se cruzaram, fim do processo

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57 procura-se i>=pivô
ij

25 33 35 12 37 86 92 57 procura-se j<=pivô
ij
→ como i e j se cruzaram, fim do processo

Todos à esquerda do pivô são menores ou iguais a ele
→ v[0]...v[j]<=pivô

Todos à direita do pivô são maiores ou iguais a ele
→ v[i]...v[n-1]>=pivô

Quick-sort: exemplo

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57 procura-se i>=pivô
ij

25 33 35 12 37 86 92 57 procura-se j<=pivô
ij
→ como i e j se cruzaram, fim do processo

Todos à esquerda do pivô são menores ou iguais a ele
→ v[0]...v[j]<=pivô

Todos à direita do pivô são maiores ou iguais a ele
→ v[i]...v[n-1]>=pivô

➔ Recomeça-se com os subvetores

Código

```
int quicksort(int a[], int p, int r) {
   int t;
   if (p < r) {
      int v = (rand()%(r-p))+p;
      int pivo = a[v];
      a[v] = a[r];
      a[r] = pivo;

      int i = p-1;
      int j = r;
      do {
          do { i ++; } while (a[i] < pivo);
          do { j --; } while ((a[j] > pivo) && (j > p));
          if (i < j) {
             t = a[i], a[i] = a[j], a[j] = t;
          }
      } while (i<j);
      a[r] = a[i];
      a[i] = pivo;

      quicksort_A(a, p, i-1);
      quicksort_A(a, i+1, r);
   }
}
```

Quick-sort

- Complexidade
  - Se vetor já ordenado com escolha do pivô como um dos extremos (elemento 0 ou n-1)
    - Subvetores desiguais, com n chamadas recursivas da função partição, eliminando-se 1 elemento em cada chamada
    - Cada chamada recursiva faz n comparações
      - $T(n)=O(n^2)$, no pior caso
        - Igual ao bubble-sort

Quick-sort

- Complexidade
  - Se a escolha do pivô divide o arquivo em partes iguais
    - O vetor de tamanho n é dividido ao meio, cada metade é dividida ao meio, ..., m vezes => $m \approx \log_2 n$
    - Cada parte do vetor realiza n comparações (com n = tamanho da partição atual do vetor)
    - Pelo método da árvore de recorrência, tem-se que $T(n)=O(n \log_2 n)$, no melhor caso

Quick-sort

- Complexidade
  - Caso médio (Sedgewick e Flajelot, 1996)
    - $T(n) \approx 1{,}386n \log_2 n - 0{,}846n = O(n \log_2 n)$

Quick-sort

- Complexidade
  - Um dos algoritmos mais rápidos para uma variedade de situações, sendo provavelmente mais utilizado do que qualquer outro algoritmo
  - Pergunta: como evitar o pior caso?

Quick-sort

- Complexidade

  - Um dos algoritmos mais rápidos para uma variedade de situações, sendo provavelmente mais utilizado do que qualquer outro algoritmo

  - Pergunta: como evitar o pior caso?

    - Boa abordagem: escolher 3 elementos quaisquer do vetor e usar a mediana deles como pivô

Ordenação por Inserção

- Idéia básica: inserir um dado elemento em sua posição correta em um subconjunto já ordenado

  - Inserção Simples, ou inserção direta

  - Shell-sort, ou classificação de shell ou, ainda, classificação de incremento decrescente

Inserção Simples

- Idéia básica
  - Ordenar o conjunto inserindo os elementos em um subconjunto já ordenado
    - No i-ésimo passo, inserir o i-ésimo elemento na posição correta entre x[0],...,x[i-1], que já estão em ordem
      - Elementos são realocados

Inserção Simples

- Idéia básica
  - Exemplo

Vetor original

10	30	31	15	50	60	5	22	35	14

Realocando o elemento 15

10	30	31	15	50	60	5	22	35	14

30 e 31 são realocados e 15 é inserido

10	15	30	31	50	60	5	22	35	14

Por que o método se chama inserção simples?

Inserção Simples

- Complexidade de tempo de melhor caso
  - Vetor ordenado: $O(n)$

- Complexidade de tempo de pior caso
  - Vetor ordenado inversamente: $O(n^2)$

- Complexidade de espaço: $O(n)$

Inserção Simples

- Inserção simples é eficiente em arquivos "quase" ordenados

- Um dos métodos mais intuitivos

Shell-sort

- Shell-sort: melhoria da inserção simples

  - Inserção simples movimenta elementos adjacentes
    - Se o menor elemento estiver na posição mais a direita, n-1 comparações e movimentos são necessários

  - Shell-sort permite a troca de elementos distantes
    - Elementos separados por h posições são ordenados de tal forma que todo h-ésimo elemento está em uma seqüência ordenada
      - Essa seqüência é dita estar h-ordenada

Shell-sort

- Exemplo

Vetor original

10	30	31	15	50	60	5	22	35	14
0	1	2	3	4	5	6	7	8	9

Shell-sort

- Exemplo

Vetor original

10	30	31	15	50	60	5	22	35	14
0	1	2	3	4	5	6	7	8	9

h=4 ➔ elementos nas posições 0, 4 (0+4) e 8 (4+4)

10	30	31	15	50	60	5	22	35	14
0	1	2	3	4	5	6	7	8	9

Shell-sort

- Exemplo

Vetor original

10	30	31	15	50	60	5	22	35	14
0	1	2	3	4	5	6	7	8	9

h=4 ➔ elementos nas posições 0, 4 (0+4) e 8 (4+4)

10	30	31	15	50	60	5	22	35	14
0	1	2	3	4	5	6	7	8	9

4-ordenado

10	30	31	15	35	60	5	22	50	14
0	1	2	3	4	5	6	7	8	9

Shell-sort

- Idéia básica: dividir a entrada em sub-conjuntos de elementos de distância h e aplicar inserção simples a cada um, sendo que h é reduzido sucessivamente
  - A cada nova iteração, o vetor original está “mais” ordenado

Shell-sort: exemplo

- 25 57 48 37 12 92 86 33

Passo1, h=5: 25 57 48 37 12 92 86 33

25 57 33 37 12 92 86 48

Shell-sort: exemplo

- 25 57 48 37 12 92 86 33

Passo2, h=3: 25 57 33 37 12 92 86 48

25 12 33 37 48 92 86 57

Shell-sort: exemplo

- 25 57 48 37 12 92 86 33

Passo 3, h=1: 25 12 33 37 48 92 86 57

12 25 33 37 48 57 86 92

Quando h=1, shellsort=?

Shell-sort: exemplo

- 25 57 48 37 12 92 86 33

Passo 3, h=1: 25 12 33 37 48 92 86 57

12 25 33 37 48 57 86 92

Quando h=1, shellsort=inserção simples

Shell-sort

- Os índices h são os incrementos que são adicionados a cada posição do vetor para se ter o próximo elemento do sub-conjunto

- A cada iteração, h decresce
  - Daí o nome "<u>incrementos decrescentes</u>" do método

- O último incremento deve sempre ser 1

Shell-sort

- n=15, h=5

1 – x[0] x[5] x[10]
2 – x[1] x[6] x[11]
3 – x[2] x[7] x[12]
4 – x[3] x[8] x[13]
5 – x[4] x[9] x[14]

- O i-ésimo elemento do j-ésimo conjunto é: x[(i-1)*h+j–1]

Shell-sort

- 25 57 48 37 12 92 86 33

Passo 1 (incremento 5):

(x[0], x[5])
(x[1], x[6])
(x[2], x[7])
(x[3])
(x[4])

Passo 2 (incremento 3):

(x[0], x[3], x[6])
(x[1], x[4], x[7])
(x[2], x[5])

Passo 3 (incremento 1):

(x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7])

Shell-sort

- Por que o método tem esse nome?

Shell-sort

- Implementação

Shell-sort

```
void shellsort(int v[], int n, int incrementos[], int numinc)
{
        int incr, i, j, h, aux;
        for (incr=0; incr<numinc; incr++) {
                h=incrementos[incr];
                for (i=h; i<n; i++) {
                        aux=v[i];
                        for (j=i-h; j>=0 && v[j]>aux; j-=h)
                                v[j+h]=v[j];
                        v[j+h]=aux;
                }
        }
}
```

Shell-sort

- Exercício
  - Executar o algoritmo anterior para o vetor (25 57 48 37 12 92 86 33)
  - 3 incrementos: 5, 3 e 1

Shell-sort

- Foi demonstrado que, com uma seqüência adequada de incrementos de h, shell-sort é aproximadamente $O(n(\log n)^2)$

  - Tarefa para casa: buscar essa demonstração/prova da complexidade do shellsort

Shell-sort

- Escolha dos incrementos é importante
  - Knuth (1973) sugere
    - Defina uma função recursiva h tal que:
      - h(1) = 1 e h(i + 1) = 3 * h(i) + 1
    - Exemplo de seqüência de incrementos: 1, 4, 13, 40, 121, 364, 1.093, 3.280, etc.
      - Aplicada no sentido inverso