SCC-601– Introdução à Ciência da Computação II

Ordenação e Complexidade – Parte 7

Lucas Antiqueira

ShellSort

ShellSort

- Melhoria da inserção simples
  - Inserção simples compara elementos adjacentes
    - Se o menor elemento estiver na posição mais a direita, n-1 comparações e movimentos são necessários para movê-lo à primeira posição
  - ShellSort permite a comparação e troca entre elementos distantes
    - Elementos separados por h posições são ordenados
      - Essa seqüência é dita estar h-ordenada: v[i] <= v[i+h].

ShellSort

- Exemplo

Vetor original

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

ShellSort

- 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

ShellSort

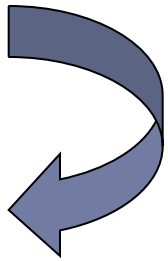
- Exemplo

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



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

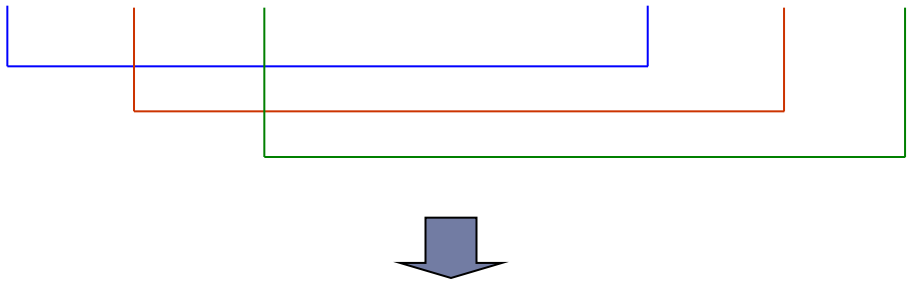
ShellSort

- Idéia: 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 próximo da ordenação final
  - h deve ser reduzido até que valor?

ShellSort

- 25 57 48 37 12 92 86 33

Passo 1, h=5: 25 57 48 37 12 92 86 33

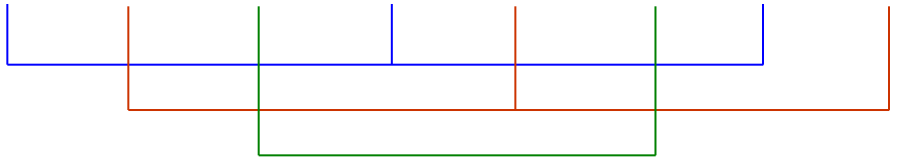


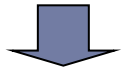
25 57 33 37 12 92 86 48

ShellSort

- 25 57 48 37 12 92 86 33

Passo 2, h=3: 25 57 33 37 12 92 86 48



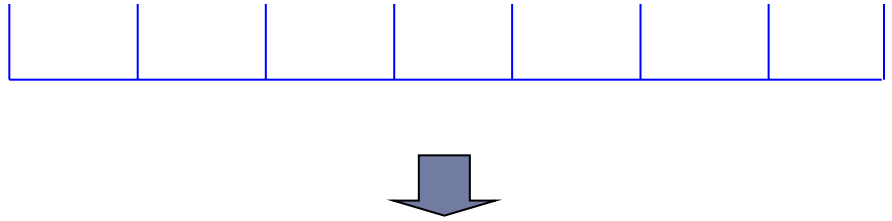


25 12 33 37 48 92 86 57

ShellSort

- 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 equivale a qual algoritmo?

ShellSort

- 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
  - Incrementos decrescentes
- O último h deve sempre ser 1

ShellSort

- n=15, h=5

j=1 → v[0] v[5] v[10]
j=2 → v[1] v[6] v[11]
j=3 → v[2] v[7] v[12]
j=4 → v[3] v[8] v[13]
j=5 → v[4] v[9] v[14]

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

ShellSort

- 25 57 48 37 12 92 86 33

Passo 1 (incremento 5):

???

ShellSort

- 25 57 48 37 12 92 86 33

Passo 1 (incremento 5):

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

ShellSort

- 25 57 48 37 12 92 86 33

Passo 1 (incremento 5):

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

Passo 2 (incremento 3):

???

ShellSort

- 25 57 48 37 12 92 86 33

Passo 1 (incremento 5):

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

Passo 2 (incremento 3):

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

ShellSort

- 25 57 48 37 12 92 86 33

Passo 1 (incremento 5):

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

Passo 2 (incremento 3):

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

Passo 3 (incremento 1):

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

ShellSort

- Quais valores de h devemos utilizar para um vetor de tamanho arbitrário n?

ShellSort

- Quais valores de h devemos utilizar para um vetor de tamanho arbitrário n?

- Várias sequências para h têm sido experimentadas

- Knuth mostrou que a sequência abaixo é difícil de ser batida por mais de 20% (por outra sequência) em eficiência no tempo de execução:

$$h(s) = 3h(s-1) + 1, \quad \text{para} \quad s > 1$$
$$h(s) = 1, \quad \text{para} \quad s = 1$$

Essa sequência corresponde a h=1, 4, 13, 40, 121, 364, 1093, ...

ShellSort

- Pode ser utilizado o seguinte algoritmo para definir h:
  - Determina-se h ≥ n, onde n é o tamanho do vetor
  - Para se obter o primeiro h a ser utilizado, divide-se h por 3
  - Os próximos h são sempre 1/3 (divisão inteira) do anterior

n=1 → h={}
n=2 → h={1}
n=3 → h={1}
n=4 → h={1}
n=5 → h={4,1}
...
n=14 → h={13,4,1}

n=15 → h={13,4,1}
...
n=41 → h={40,13,4,1}
...
n=122 → h={121,40,13,4,1}
...
n=365 → h={364,121,40,13,4,1}
...
n=1094 → h={1093,364,121,40,13,4,1}

ShellSort

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

        for (h=1; h<n; h=3*h+1)
                ;
        while (h>1) {
                h = h/3;
                for (i=h; i<n; i++) {
                        x = v[i];
                        j = i;
                        while (j>=h && v[j-h] > x) {
                                v[j] = v[j-h];
                                j = j - h;
                        }
                        v[j] = x;
                }
        }
}
```

ShellSort

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

        for (h=1; h<n; h=3*h+1)
                ;
        while (h>1) {
                h = h/3;
                for (i=h; i<n; i++) {
                        x = v[i];
                        j = i;
                        while (j>=h && v[j-h] > x) {
                                v[j] = v[j-h];
                                j = j - h;
                        }
                        v[j] = x;
                }
        }
}
```

Encontra h≥n

ShellSort

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

        for (h=1; h<n; h=3*h+1)
                ;
        while (h>1) {
                h = h/3;
                for (i=h; i<n; i++) {
                        x = v[i];
                        j = i;
                        while (j>=h && v[j-h] > x) {
                                v[j] = v[j-h];
                                j = j - h;
                        }
                        v[j] = x;
                }
        }
}
```

Para cada h

ShellSort

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

        for (h=1; h<n; h=3*h+1)
                ;
        while (h>1) {
                h = h/3;
                for (i=h; i<n; i++) {
                        x = v[i];
                        j = i;
                        while (j>=h && v[j-h] > x) {
                                v[j] = v[j-h];
                                j = j - h;
                        }
                        v[j] = x;
                }
        }
}
```

Algoritmo de inserção modificado

ShellSort

Exercício

Executar o algoritmo anterior para o vetor
(25 57 48 37 12 92 86 33)

ShellSort

- Por que o método tem esse nome?

ShellSort

- Por que o método tem esse nome?
  - Foi criado por Donald Shell, em 1959

ShellSort

- Qual a complexidade do ShellSort?

ShellSort

- É um método eficiente, mas difícil de ser analisado.

Conjectura 1 $C(n) = O(n^{1.25})$

Conjectura 2 $C(n) = O(n(log_e n)^2)$

- Exemplo de análise:

  Robert Sedgewick, Analysis of Shellsort and Related Algorithms

  http://www.cs.princeton.edu/~rs/shell/paperF.pdf

- Complexidade de espaço: ?

ShellSort

- É um método eficiente, mas difícil de ser analisado.

Conjectura 1 $C(n) = O(n^{1.25})$

Conjectura 2 $C(n) = O(n(log_e n)^2)$

- Exemplo de análise:

  Robert Sedgewick, Analysis of Shellsort and Related Algorithms

  http://www.cs.princeton.edu/~rs/shell/paperF.pdf

- Complexidade de espaço: $O(n)$

ShellSort

- ShellSort é estável?

ShellSort

- ShellSort é estável? Não
  - Justamente pelo fato de ocorrerem trocas entre elementos distantes

Comparação empírica

- Ordem aleatória dos elementos
  - Tempos normalizados (divididos) pelo tempo do algoritmo mais rápido.

	500	5.000	10.000	30.000
Inserção	11,3	87	161	–
Seleção	16,2	124	228	–
Shellsort	1,2	1,6	1,7	2
Quicksort	1	1	1	1
Heapsort	1,5	1,6	1,6	1,6

[1] Nívio Ziviani. *Projeto de Algoritmos – com implementações em C e Pascal.* Thomson Editora, 2a. edition, 2004.

Comparação empírica

- Ordem ascendente dos elementos (já ordenado)

	500	5.000	10.000	30.000
Inserção	1	1	1	1
Seleção	128	1.524	3.066	–
Shellsort	3,9	6,8	7,3	8,1
Quicksort	4,1	6,3	6,8	7,1
Heapsort	12,2	20,8	22,4	24,6

Comparação empírica

- Ordem descendente dos elementos

	500	5.000	10.000	30.000
Inserção	40,3	305	575	–
Seleção	29,3	221	417	–
Shellsort	1,5	1,5	1,6	1,6
Quicksort	1	1	1	1
Heapsort	2,5	2,7	2,7	2,9

Comparação empírica

- ShellSort, QuickSort e HeapSort possuem a mesma ordem de grandeza na maior parte dos casos
- QuickSort é o mais rápido para todos os vetores com elementos aleatórios
- Para vetores aleatórios pequenos, ShellSort é melhor do que o HeapSort
- O método da Inserção Direta é o mais rápido para todos os tamanhos de vetores ordenados, mas é o mais lento para vetores inversamente ordenados
- O método da Inserção Direta é melhor do que o método da Seleção Direta para vetores com elementos aleatórios

Créditos

Aula baseada nos materiais dos profs.
Thiago A. S. Pardo e Alneu A. Lopes