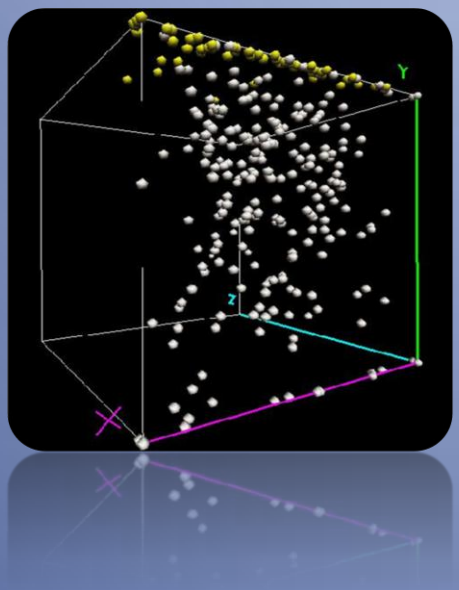

ALGORITMOS E ESTRUTURAS DE DADOS III

Tutorial 2 (usa o compilador de linguagem C Dev-C++ versão 4.9.9.2)

Parte 2 de 3 sobre o algoritmo de ordenação shell (concha) conhecido como Shellsort.

1 Introdução

Esta série de tutoriais sobre *Algoritmos e Estruturas de Dados III* foi escrita usando o **Microsoft Windows 7 Ultimate**, **Microsoft Office 2010**, **Bloodshed Dev-C++** versão 4.9.9.2 (pode ser baixado em http://www.bloodshed.net), referências na internet e notas de aula do professor quando estudante. Ela cobre desde os algoritmos de ordenação, passando pela pesquisa em memória primária e culminando com a pesquisa em memória secundária.

Nós entendemos que você já conhece o compilador Dev-C++. No caso de você ainda não o conhecer, dê uma olhada nos tutoriais Dev-C++ 001 a 017, começando pelo Tutorial Dev-C++ - 001 - Introdução.

Se não tem problemas com a linguagem C/C++ e o compilador Dev-C++, então o próximo passo é saber ler, criar e alterar arquivos em disco usando linguagem C/C++. Se ainda não sabe como fazê-lo, dê uma olhada nos tutoriais Dev-C++ 001 e 002, começando pelo Tutorial Dev-C++ 001 – Criação, Leitura e Alteração de Arquivos.

Se sabe todas as coisas anteriores, então a próxima etapa é conhecer os algoritmos mais básicos de ordenação. Em minhas notas de aula você encontra um material básico, porém detalhado e com algoritmos resolvidos, dos principais métodos de ordenação existentes.

Adotaremos o livro ***Projeto de Algoritmos com Implementação em Pascal e C***, Editora Cengage Learning, de Nivio Ziviani, como livro-texto da disciplina. Nele você encontrará os métodos de ordenação que iremos estudar.

Seu próximo passo será estudar os algoritmos de ordenação por **Inserção** e por **Seleção**. Você pode usar os links anteriores (em inglês) ou fazer uso do livro-texto.

Em seguida, você precisa conhecer o algoritmo Shellsort. Para isto, você pode seguir o **Tutorial AED 001**, desta série, e/ou ler o capítulo referente no livro-texto.

Se você estiver lendo este tutorial tenha certeza de ter seguido o Tutorial AED 001. Agora que você seguiu todos os passos até aqui, está pronto para prosseguir com este tutorial.

2 O Algoritmo de Ordenação Shellsort

O sort por inserção é lento porque troca apenas elementos adjacentes. Por exemplo, se o menor elemento estiver no final do arranjo, são necessários N passos para colocá-lo em seu devido lugar. O Shellsort é uma extensão simples do sort por inserção, que ganha velocidade por permitir a troca de elementos distantes entre si.

A ideia é arranjar os dados de tal forma que, tomando-se seu $h_{\text{-ésimo}}$ elemento (a partir de qualquer posição), tem-se um arranjo ordenado, dito h-ordenado. Falando de outra forma, um arranjo h-ordenado é composto de h arranjos ordenados, tomados juntos. Através da h-ordenação para valores grandes de h, pode-se mover elementos dentro do arranjo por grandes distâncias, e assim ordenar mais facilmente para valores menores que h. Usando-se este procedimento para qualquer sequência de valores que termine no tamanho 1 produzirá um arranjo totalmente ordenado.

Uma maneira de implementar o Shellsort é, para cada h, usar o sort de inserção independentemente em cada um dos h subarranjos. A escolha dos valores de h pode ser feita pelo critério de Knuth [Niemann 97][1], como se segue:

Faça $h_1 = 1$, $h_{s+1} = 3h_s + 1$, e pare com h_t quando $h_t + 2 \geq N$.

Assim, os valores de h são computados como se segue:

$h_1 = 1$

$h_2 = (3 * 1) + 1 = 4$

$h_3 = (3 * 4) + 1 = 13$

$h_4 = (3 * 13) + 1 = 40$

$h_5 = (3 * 40) + 1 = 121$

$h_6 = (3 * 121) + 1 = 364$ etc.

Uma propriedade muito importante [Sedgewick 86][2] é a seguinte:

Shellsort nunca realiza mais do que N3/2 comparações (para os incrementos 1, 4, 13, ...).

[1] Niemann, Thomas. *Sorting and searching algorithms: a cookbook*. http://www.geocities.com/SoHo/2167/book.html.

[2] Sedgewick, Robert. *Algorithms*. Addison Wesley, 1986.

Uma implementação para o Shellsort é vista na *listagem 1* e na *listagem 2*:

2.1 Funcionamento do Algoritmo

Acompanhe pela listagem 1. Primeiro ciclo do-while gera sequência: 1 4 13 40 121 364 1093 3280 ...

Segundo ciclo for é executado para os valores de h por ordem inversa.

- Operação (vetor com tamanho 100):
  - Para cada valor de h, 40, 13, 4, 1:
    - Utilizar ordenação por inserção para criar subvetores ordenados dentro de vetor com tamanho 100.
      - Vector fica $h_{-ordenado}$.
    - Para h = 40 existem 40 subvetores ordenados, cada um com 2/3 elementos.
    - Para h = 13 existem 13 subvetores ordenados, cada um com 7/8 elementos.
    - ...
    - Para h = 1 existe 1 (subvetor ordenado, com 100 elementos.

2.2 Complexidade

Complexidade depende da sequência de valores utilizada:

- Sequência 1, 4, 13, 40, 121, 364, 1093, ...
  - $O(N^{3/2})$ comparações
- Sequência 1, 8, 23, 77, 281, 1073, 4193, ...
  - $O(N^{4/3})$ comparações
- Sequência 1, 2, 3, 4, 6, 9, 8, 12, 18, 27, 16, 24, ...
  - $O(N(\log N)^2)$ comparações

2.3 Implementações

Listagem 1:

```
void shellSort(int *vet, int size) {
  int i, j, value;
  int gap = 1;
  do {
    gap = 3 * gap + 1;
  } while(gap < size);
  do {
    gap /= 3;
    for(i = gap; i < size; i++) {
      value = vet[i];
      j = i - gap;
      while(j >= 0 && value < vet[j]) {
        vet [j + gap] = vet[j];
        j -= gap;
      }
      vet [j + gap] = value;
    }
  } while(gap > 1);
}
```

Listagem 2:

```
void shellSort(int *vetor, int tamanho)
{
  int i = (tamanho - 1) / 2;
  int chave, k, aux;

  while(i != 0)
  {
    do
    {
      chave = 1;
      for(k = 0; k < MAX - i; ++k)
      {
        if(vetor[k] > vetor[k + i])
        {
          aux = vetor[k];
          vetor[k] = vetor[k + i];
          vetor[k + i] = aux;
          chave = 0;
        }
      }
    } while(chave == 0);
    i = i / 2;
  }
}
```

2.4 Usando Listas

Para sequência de dados muito grande, talvez seja melhor uma abordagem usando lista e ordenação por seleção.

Uma sugestão poderia ser:

Interface:

```
typedef struct node *link;
struct node {Item item; link next;};
link NEW(Item, link);
link init(int);
void show(link);
link sort(link);
```

Seleção:

```
link listselection(link h) {
  link max, t, out = NULL;
  while(h->next != NULL) {
    max = findmax(h);
    t = max->next; max->next = t->next;
    t->next = out; out = t;
  }
  h->next = out;
```

```
  return(h);
}
```

A cada passo retira o máximo elemento da lista atual, e coloca no topo da nova lista.

Exercício de fixação
Implemente um algoritmo usando lista e ordenação por seleção para ordenar a lista de 10000 inteiros, fornecida no meu blog.

4 TERMINAMOS

Terminamos por aqui. Clique no menu Arquivo, depois clique na opção Sair.

Corra para o próximo tutorial.