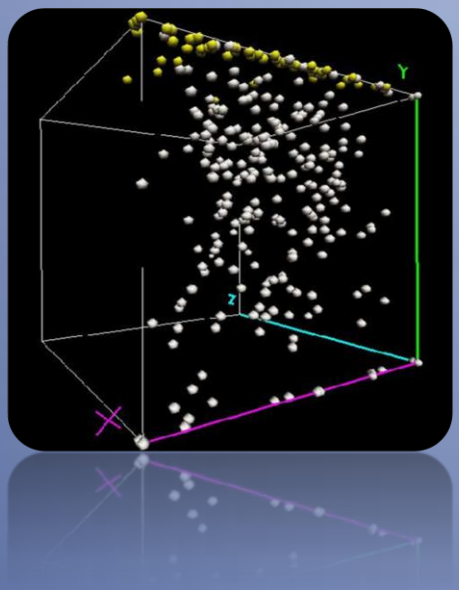

ALGORITMOS E ESTRUTURAS DE DADOS III

Tutorial 16 (usa o compilador de linguagem C Dev-C++ versão 4.9.9.2)

Pesquisa em memória primária: pesquisa binária.

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), **Code::Blocks** versão 10.05 (pode ser baixado em http://www.codeblocks.org) 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.

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.

Se você seguiu todos os passos até aqui, está pronto para prosseguir com este tutorial.

2 Pesquisa em Memória Primária

2.1 Pesquisa Binária

A pesquisa ou busca binária (em inglês *binary search algorithm* ou *binary chop*) é um algoritmo de busca em vetores que segue o paradigma de divisão e conquista. Ela parte do pressuposto de que o vetor está ordenado e realiza sucessivas divisões do espaço de busca comparando o elemento buscado (chave) com o elemento no meio do vetor. Se o elemento do meio do vetor for a chave, a busca termina com sucesso. Caso contrário, se o elemento do meio vier antes do elemento buscado, então a busca continua na metade posterior do vetor. E finalmente, se o elemento do meio vier depois da chave, a busca continua na metade anterior do vetor.

2.2 Análise de Complexidade

A complexidade desse algoritmo é da ordem de $\Theta(\log_2 n)$, em que n é o tamanho do vetor de busca. Apresenta-se mais eficiente que a Busca linear cuja ordem é $O(n)$.

2.3 Pseudocódigo

Um pseudocódigo recursivo para esse algoritmo, dados V o vetor com elementos comparáveis, n seu tamanho e e o elemento que se deseja encontrar:

```
BUSCA-BINÁRIA (V[], início, fim, e)
   i recebe o índice do meio entre início e fim
   se (v[i] = e) entao
      devolva o índice i   # elemento e encontrado
   fimse
   se (inicio = fim) entao
      não encontrou o elemento procurado
   senão
     se (V[i] vem antes de e) então
       faça a BUSCA-BINÁRIA(V, i+1, fim, e)
     senão
       faça a BUSCA-BINÁRIA(V, inicio, i-1, e)
     fimse
   fimse
```

2.4 Exemplo de Código em C

```
//em C para se passar um vetor como parâmetro para uma
função, tem que se passar o ponteiro do vetor
int PesquisaBinaria ( int *array, int chave , int N) {
    int inf = 0; //Limite inferior
                 // (o primeiro elemento do vetor em C é zero)
    int sup = N-1; //Limite superior
                   // (termina em um número a menos 0 à 9
                   // são 10 numeros )
    int meio;
    while (inf <= sup) {
        meio = inf + (sup-inf)/2;
        if (chave == array[meio])
            return meio;
        else if (chave < array[meio])
            sup = meio-1;
        else
            inf = meio+1;
    }
    return -1;   // não encontrado
}
```

2.5 Um exemplo completo

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

using namespace std;

const int MAXN = 10000;
int Tab[MAXN];

void doGeraTabela() {
   int i;

   srand(time(NULL));
   for(i = 0; i < MAXN; i++) {
      Tab[i] = rand() % 10000;
   }
}

void doSort() {
   int i;
   int j;
   int aux;

   printf("\nOrdenando. Aguarde...\n");
   for(i = 0; i < MAXN - 1; i++)
      for(j = i + 1; j < MAXN; j++)
         if(Tab[i] > Tab[j]) {
            aux = Tab[i];
            Tab[i] = Tab[j];
            Tab[j] = aux;
         }
   printf("Ordenado.\n\n");
}

void doPesquisaSequencial(int v) {
   int i;

   printf("Pesquisa sequencial em andamento...\n");

   for(i = 0; i < MAXN; i++)
      if(Tab[i] == v)
         printf("Valor %d encontrado na posicao %d...\n", v, i);

   printf("\nFim da pesquisa sequencial...\n\n");
}
```

```
void doPesquisaBinaria(int v) {
   int i;
   int esq;
   int dir;

   printf("Pesquisa binaria em andamento...\n");

   doSort();

   esq = 0;
   dir = MAXN - 1;

   do {
      i = (esq + dir) / 2;
      if(v > Tab[i])
         esq++;
      else dir--;
   } while ((v != Tab[i]) && (esq <= dir));

   if(v == Tab[i])
      printf("Valor %d encontrado na posicao %d...\n", v, i);
   else printf("Valor %d nao encontrado...\n", v);

   printf("\nFim da pesquisa binaria...\n\n");
}

int main() {
   int op;
   int valor;

   doGeraTabela();

   do {
      printf("\n\n1. Pesquisa sequencial\n");
      printf("2. Pesquisa binaria\n");
      printf("Opcao: "); scanf("%d", &op);
      printf("\n\n");

      if(op != 3) {
         printf("Valor procurado: ");
         scanf("%d", &valor);
         printf("\n\n");

         switch(op) {
            case 1:
               doPesquisaSequencial(valor);
               break;

            case 2:
               doPesquisaBinaria(valor);
               break;
         }
      }
   } while(op != 3);

   return 0;
}
```

3 EXERCÍCIOS

1. Invente um vetor-exemplo de entrada e use o algoritmo de pesquisa binária para buscar um valor qualquer fornecido pelo usuário. Lembre-se, a busca binária só funciona em vetores previamente ordenados.

4 TERMINAMOS

Terminamos por aqui. Mais em http://flavioaf.blogspot.com.

Corra para o próximo tutorial.