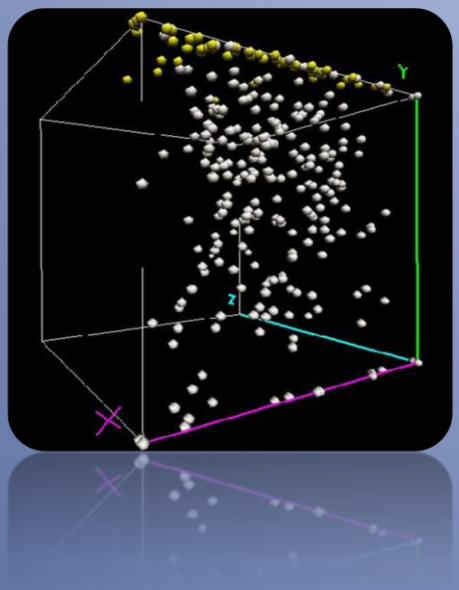

ALGORITMOS E ESTRUTURAS DE DADOS III

Tutorial 11 (usa o compilador de linguagem C Dev-C++ versão 4.9.9.2)

Parte 2 de 3 sobre ordenação externa: intercalação balanceada.

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.

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**, **Seleção**, **Shellsort**, **Heapsort** e **Quicksort**. Você pode usar os links anteriores ou fazer uso do livro-texto.

Se você estiver lendo este tutorial tenha certeza de ter seguido os Tutoriais AED 001 a 010. Agora que você seguiu todos os passos até aqui, está pronto para prosseguir com este tutorial.

2 ARQUIVOS EM DISCO

2.1 ESTRUTURA ENDERECOS

Muitos dos programas aplicativos que requerem uma ordenação precisam ter um agrupamento de dados ordenados. Uma lista postal é um excelente exemplo, porque o nome, a rua, a cidade, o estado e o CEP estão todos relacionados. Quando essa unidade conglomerada de dados é ordenada, uma chave de ordenação é usada, mas toda a estrutura é trocada. Para entender como isso é feito, primeiro criemos uma estrutura. Uma estrutura conveniente é

Listagem 1:

```
struct enderecos {
   char nome[60];
   char rua[60];
   char cidade[30];
   char estado[3];
   char cep[10];
}
```

estado tem extensão de três caracteres e **cep** de 10 caracteres, porque uma matriz sempre precisa de um caractere a mais do que o comprimento máximo da *string* para armazenar o terminador nulo.

Como é razoável que uma lista postal possa ser arranjada como uma matriz de estruturas assuma, para esse exemplo, que a rotina ordenará uma matriz de estruturas do tipo **enderecos**. A rotina é mostrada aqui:

Listagem 2:

```
// um quicksort para estruturas do tipo enderecos
void quick_struct(struct enderecos item[], int count) {
   qs_struct(item, 0, count – 1);
}

void qs_struct(struct enderecos item[], int left, int right) {
   register int i, j;
   char *x;
   struct enderecos temp;
   i = left; j = right;
   x = item[(left + right) / 2].cep;
   do {
      while((strcmp(item[i].cep, x) < 0) && (i < right)) i++;
      while((strcmp(item[j].cep, x) > 0) && (j > left)) j--;
      if(i <= j) {
         temp = item[i];
         item[i] = item[j];
         item[j] = temp;
         i++; j--;
      }
   } while(i <= j);
   if(left < j) qs_struct(item, left, j);
   if(i < right) qs_struct(item, i, right);
}
```

2.2 Tipos de Acesso a Arquivos em Disco

Existem dois tipos de arquivos em disco: sequenciais e de acesso aleatório. Se qualquer tipo de arquivo em disco for pequeno o bastante, ele pode ser lido para a memória e as rotinas de ordenação de matrizes apresentadas anteriormente podem ser utilizadas para ordená-lo. Porém, muitos arquivos em disco são muito grandes para serem ordenados facilmente na memória e exigem técnicas especiais. Este tutorial mostrará uma maneira pela qual arquivos em disco de acesso aleatório podem ser ordenados.

Arquivos em disco de acesso aleatório têm duas vantagens principais sobre arquivos sequenciais em disco. Primeiro, eles são mais fáceis de manter. Você pode atualizar informações sem precisar copiar toda a lista. Segundo, eles podem ser tratados como uma matriz muito grande em disco, o que simplifica enormemente a ordenação.

Tratar um arquivo de acesso aleatório como uma matriz significa que você pode usar a base da Quicksort com poucas modificações. Em lugar de indexar uma matriz, a versão em disco da Quicksort deve usar **fseek()** para pesquisar os registros apropriados no disco.

Cada situação de ordenação difere da estrutura exata de dados que é ordenada e da chave que é usada. Todavia, a ideia geral de ordenação de arquivos de acesso aleatório em disco pode ser compreendida desenvolvendo-se um programa de ordenação para estruturas do tipo **enderecos**, a estrutura para a lista postal definida anteriormente. O programa-exemplo a seguir assume que o número de elementos é fixado em 100. Em aplicações reais, o número de registro a ser ordenado é determinado dinamicamente.

2.3 Copiando Arquivos em Disco

Copiar arquivos em disco pode ser importante para as rotinas de intercalação balanceada. Relembrando, para ordenar um arquivo muito grande em disco, que não cabe na memória do computador, é preciso ordená-lo em partes, cada parte sendo intercalada ordenadamente com a anterior, já ordenada. Veja na figura abaixo o diagrama do processo de ordenação e intercalação.

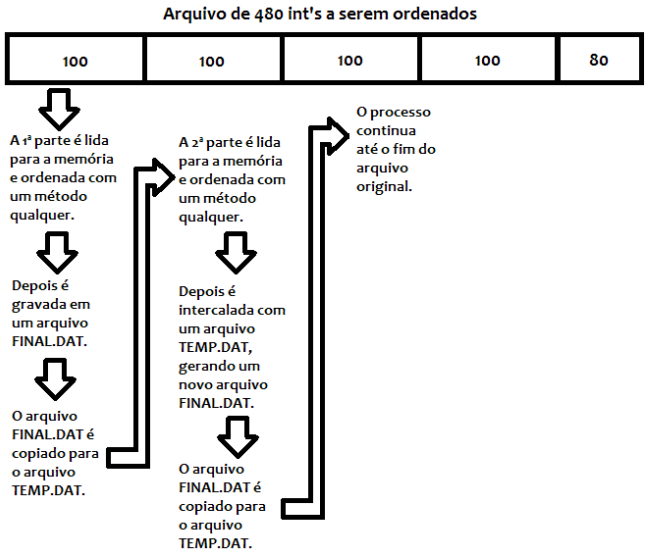

FIGURA 1: ORDENAÇÃO POR INTERCALAÇÃO DE UM ARQUIVO MUITO GRANDE, QUE NÃO CABE NA MEMÓRIA FÍSICA DO COMPUTADOR

Assim, pode ser preciso gerar arquivos temporários a todo instante, copiando o arquivo intercalado na passagem anterior para um arquivo temporário, a ser intercalado na passagem atual da ordenação. Na *listagem 3* você pode ver um programa para cópia de arquivos. O programa trabalha no modo *prompt* e os arquivos a serem copiados são passados na linha de comando, ao estilo MS-DOS.

Exemplo:
copia origem.ext destino.ext ↵

Se quiser testar o programa enquanto o executa no Dev-C++, basta colocar os arquivos de origem e destino nos parâmetros. Vá no menu Executar ▸ Parâmetros... e preencha a caixa de diálogo *Parâmetros* de acordo com a imagem abaixo.

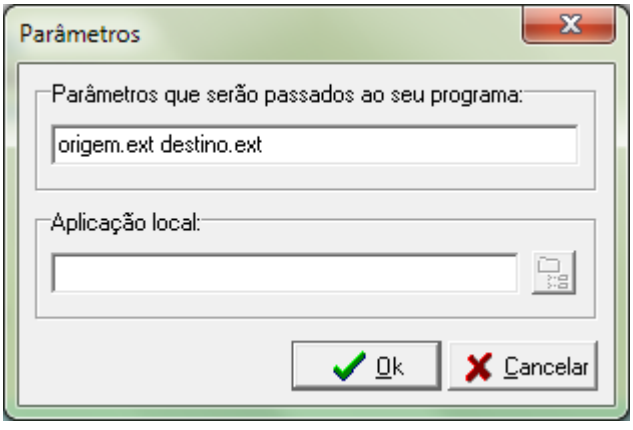

FIGURA 2: PARÂMETROS NO DEV

Clique em Ok e execute o programa.

Se estiver usando o Code::Blocks 10.05, vá no menu Project ▸ Set programs' arguments... e ajuste a caixa de diálogo *Select target* de acordo com a imagem abaixo.

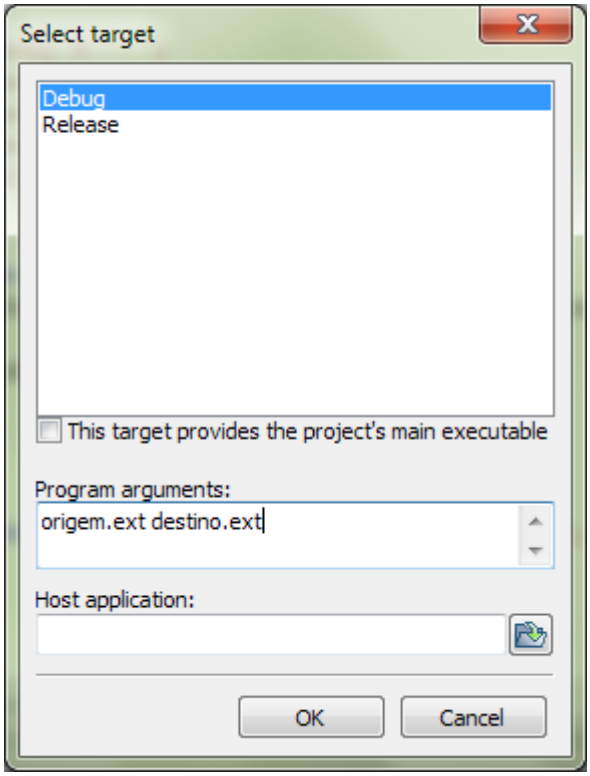

FIGURA 3: PARÂMETROS NO CODE::BLOCKS

Clique em OK.

Listagem 3:

```
// programa que copia arquivo

#include <stdio.h>

int main(int argc, char *argv[]) {
   FILE *original,*copia;
   char caracter;

   if(argc < 3) {
      printf("\nSintaxe correta:\n\n");
      printf("copiar <ARQ. ORIGEM> <ARQ. DESTINO>\n\n");
      exit(1);
   }

   while(argv[1]) {
      if(*argv[1] == *argv[2]) break;
      printf("\nNome <ARQ. DESTINO> inválido.\n\n");
      exit(1);
   };

   if((original = fopen(argv[1],"rb")) == NULL) {
      printf("\nErro ao abrir <ARQ. ORIGEM>.\n\n");
      exit(1);
   }

   if((copia = fopen(argv[2],"wb")) == NULL) {
      printf("\nErro ao criar <ARQ. DESTINO>.\n\n");
      exit(1);
    }

   while(!feof(original)) {
      caracter = getc(original);
      if(!feof(original)) putc(caracter, copia);
   }

   fclose(original);
   fclose(copia);

   printf("\n%s copiado para %s ");
   printf("com sucesso.\n\n",argv[1], argv[2]);

   return(0);
 }
```

3 QUICKSORT EXTERNO

Listagem 4:

```
// ordenação em disco para estruturas do tipo enderecos
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// altere NUM_REGISTROS de acordo com a quantidade
// de registros existentes no arquivo
const int NUM_REGISTROS = 1599;

// listagem 1
struct enderecos {
   char nome[60];
   char rua[60];
   char cidade[30];
   char estado[3];
   char cep[10];
} ainfo;

void quick_disk(FILE *fp, int count);
void qs_disk(FILE *fp, int left, int right);
void swap_all_fields(FILE *fp, long i, long j);
char *get_cep(FILE *fp, long rec);

int main(void) {
   FILE *fp;
   if((fp=fopen("listagem.dat", "rb+"))==NULL) {
      printf("Arquivo nao pode ser aberto para E/S.\n");
      exit(1);
   }

   // coloque aqui uma rotina para impressão do
   // arquivo antes de ordenar

   quick_disk(fp, NUM_REGISTROS);
   fclose(fp);
   printf("Lista ordenada.\n");

   // coloque aqui uma rotina para impressão do
   // arquivo após a ordenação

   return 0;
}

void quick_disk(FILE *fp, int count) {
   qs_disk(fp, 0, count – 1);
}
```

```
void qs_disk(FILE *fp, int left, int right) {
   long int i, j;
   char x[100];

   i = left; j = right;
   // obtém o CEP intermediário
   strcpy(x, get_cep(fp, (long) (i + j) / 2));
   do {
      while((strcmp(get_cep(fp, i), x) < 0) && (i < right)) i++;
      while((strcmp(get_cep(fp, j), x) > 0) && (j > left)) j--;
      if(i <= j) {
         swap_all_fields(fp, i, j);
         i++; j--;
      }
   } while(i <= j);
   if(left < j) qs_disk(fp, left, (int) j);
   if(i < right) qs_disk(fp, (int) i, right);
}

void swap_all_fields(FILE *fp, long i, long j) {
   char a[sizeof(ainfo)], b[sizeof(ainfo)];

   // primeiro lê os registros i e j
   fseek(fp, sizeof(ainfo) * i, 0);
   fread(a, sizeof(ainfo), 1, fp);
   fseek(fp, sizeof(ainfo) * j, 0);
   fread(b, sizeof(ainfo), 1, fp);

// em seguida escreve-os de volta em posições trocadas
   fseek(fp, sizeof(ainfo) * j, 0);
   fwrite(a, sizeof(ainfo), 1, fp);
   fseek(fp, sizeof(ainfo) * i, 0);
   fwrite(b, sizeof(ainfo), 1, fp);
}

// devolve um ponteiro para o código CEP
char *get_cep(FILE *fp, long rec) {
   struct enderecos *p;

   p = &ainfo;
   fseek(fp, rec * sizeof(ainfo), 0);
   fread(p, sizeof(ainfo), 1, fp);
   return ainfo.cep;
}
```

4 Exercícios Propostos

1. Usando uma modificação do algoritmo da *listagem 4*, ordene o arquivo de 10000 números inteiros que está no blog.

2. Usando outra modificação do algoritmo da *listagem 4*, ordene o arquivo de 10000 números em ponto flutuante que está no blog.

3. Usando o algoritmo da *listagem 4*, faça uma modificação para que ordene a estrutura **enderecos** pela chave composta {estado, cep, rua, nome}.

4. Usando o esquema da *figura 1*, crie um programa de ordenação por intercalação para ordenar o arquivo de 10000 números inteiros que está no blog.

5 Terminamos

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

Corra para o próximo tutorial.