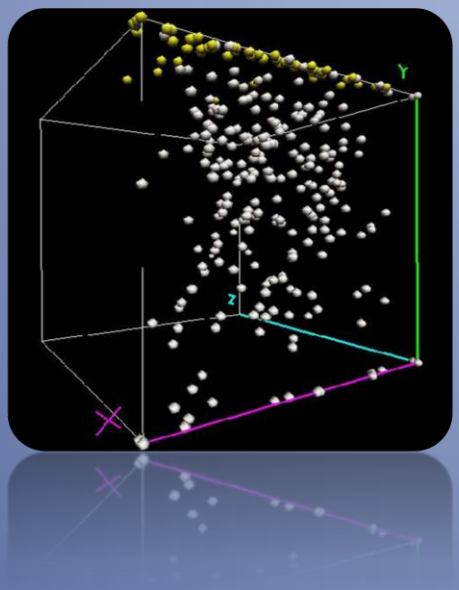

ALGORITMOS E ESTRUTURAS DE DADOS III

Tutorial 12 (usa o compilador de linguagem C Dev-C++ versão 4.9.9.2)

Parte 3 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 011. Agora que você seguiu todos os passos até aqui, está pronto para prosseguir com este tutorial.

2 MERGE ENTRE VETORES

Após tanta teoria, um exemplo de merge entre vetores ordenados deve dar alguma luz sobre como poderemos fazer um Mergesort Externo para ordenar um arquivo realmente grande.

Listagem 1:

```
// Feito no Code::Blocks
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

using namespace std;

int compare(const void * a, const void * b) {
   return( *(int*)a - *(int*)b );
}

int main() {
   const int MAXA = 60;
   const int MAXB = 10;
   const int MAXC = MAXA + MAXB;
   int ArrayA[MAXA];
   int ArrayB[MAXB];
   int ArrayC[MAXC];
   int pA; // ponteiro do ArrayA
   int pB; // ponteiro do ArrayB
   int pC; // ponteiro do ArrayC

   srand(time(NULL));

   printf("Gerando os vetores...\n\n");

   // a intenção é que haja repetição dentro do vetor
   for(pA=0;pA<MAXA;pA++) ArrayA[pA] = rand() % MAXA / 2 + 1;
   // a intenção é que haja repetição dentro do vetor
   for(pB=0;pB<MAXB;pB++) ArrayB[pB] = rand() % MAXB / 2 + 1;

   printf("Vetor A desordenado: %d elementos...\n", MAXA);
   for(pA=0;pA<MAXA;pA++) {
      printf("%02d ", ArrayA[pA]);
      if((pA % 19 == 0) && (pA != 0)) putchar('\n');
   }
   printf("\n\n");

   printf("Vetor B desordenado: %d elementos...\n", MAXB);
   for(pB=0;pB<MAXB;pB++) {
      printf("%02d ", ArrayB[pB]);
      if((pB % 19 == 0) && (pB != 0)) putchar('\n');
   }
   printf("\n\n");

   printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");
   printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n\n");

   // ordenando os vetores
   printf("Ordenando os vetores...\n\n");
   qsort(ArrayA, MAXA, sizeof(int), compare);
   qsort(ArrayB, MAXB, sizeof(int), compare);

   printf("Vetor A ordenado: %d elementos...\n", MAXA);
   for(pA=0;pA<MAXA;pA++) {
      printf("%02d ", ArrayA[pA]);
      if((pA % 19 == 0) && (pA != 0)) putchar('\n');
   }
   printf("\n\n");

   printf("Vetor B ordenado: %d elementos...\n", MAXB);
   for(pB=0;pB<MAXB;pB++) {
      printf("%02d ", ArrayB[pB]);
      if((pB % 19 == 0) && (pB != 0)) putchar('\n');
   }
   printf("\n\n");

   printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");
   printf("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-\n\n");
```

```
    // intercalando os vetores
    printf("Intercalando os vetores: %d elementos...\n\n", MAXC);
    pA = 0; pB = 0; pC = 0;
    while(pC < MAXC) {
       if((pA < MAXA) && (pB < MAXB)) {
          if(ArrayA[pA] <= ArrayB[pB])
             ArrayC[pC++] = ArrayA[pA++];
          else ArrayC[pC++] = ArrayB[pB++];
       }
       else
       if(pB >= MAXB) // ArrayB já está totalmente intercalado
          ArrayC[pC++] = ArrayA[pA++];
       else
       if(pA >= MAXA) // ArrayA já está totalmente intercalado
          ArrayC[pC++] = ArrayB[pB++];
    }

    printf("Vetor C intercalado: ");
    printf("A(%d) e B(%d) em ");
    printf("C(%d) elementos...\n", MAXA, MAXB, MAXC);
    for(pC=0;pC<MAXC;pC++) {
       printf("%02d ", ArrayC[pC]);
       if((pC % 20 == 0) && (pC != 0)) putchar('\n');
    }
    printf("\n\n");

    return 0;
}
```

A *listagem 1* mostra um algoritmo simples que gera dois vetores A e B com certo número de elementos e um vetor C com a quantidade de B juntos. Os vetores A e B são gerados aleatoriamente, mas de forma que haja, propositalmente, elementos repetidos. Depois os vetores A e B são ordenados usando-se **qsort** (a função nativa de ordenação Quicksort do C). A rotina realizada em seguida é a mais interessante, pois é a rotina de intercalação dos vetores A e B dentro do vetor C. Assim, ao término da intercalação, o vetor C contém todos os elementos do vetor A e do vetor B, mas de tal sorte, que estão ordenados.

3 MERGESORT EXTERNO

Como pode ser visto na *listagem 1*, não é difícil modificar o código para que faça a ordenação de um arquivo muito grande em disco. O vetor A pode ser usado para receber os dados lidos do arquivo a ordenar, por exemplo, até 100 elementos. Esses elementos seriam então ordenados usando-se **qsort**. Em seguida seriam gravados em um arquivo temporário em disco (TEMP1). Depois leríamos mais 100 elementos do arquivo em disco a ordenar para o vetor A, que seria novamente ordenado usando-se outra vez **qsort**. Usaríamos uma modificação da rotina de intercalação para intercalar o vetor ordenado em memória com o arquivo temporário em disco (TEMP1), gerando outro arquivo temporário (TEMP2). Apagamos o primeiro arquivo temporário (TEMP1). Renomeamos TEMP2 para TEMP1. Repetimos o processo de ler elementos do arquivo em disco a ordenar, ordená-los, intercalá-los com TEMP1, gerando TEMP2. Fazemos isso até termos lido, ordenado e intercalado todos os elementos do arquivo em disco a ordenar. Finalmente renomeamos TEMP2 para FINAL, e temos nosso arquivo em disco ordenado.

O programa da *listagem 2*, usa os conceitos do programa da *listagem 1* para ordenar o arquivo de 10000 números inteiros que está no blog. A ordenação se dá em etapas, cada uma lendo, ordenando e intercalando 100 inteiros por vez. Coloque o arquivo com os *10000 números inteiros* na mesma pasta do programa e o renomeie para **listagem.dat**.

Listagem 2:

```
// Feito no Code::Blocks
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

//const bool DEBUG = true;
const bool DEBUG = false;

using namespace std;

int compare(const void * a, const void * b) {
   return( *(int*)a - *(int*)b );
}

int HouverArquivoASerLido(FILE *fp) {
   return(!feof(fp));
}

int main() {
   FILE *fpDAT; // ponteiro para o arquivo de dados
   FILE *fpTMP; // ponteiro para o arquivo temporário
   FILE *fpEND; // ponteiro para o arquivo final

   const int MAX_ITENS_VETOR = 100; // máxima
quantidade lida possível

   const char f_dat     [80] = "listagem.dat"; // arq. de dados
   const char f_tmp     [80] = "listagem.tmp"; // arq. temp.
   const char f_end     [80] = "listagem.end"; // arq. final

   const char F_APPEND  [ 5] = "a+"; // adicionar dados
   const char F_READONLY[ 5] = "rt"; // somente leitura
   const char F_WRITE   [ 5] = "w";  // criar novo

   int Vetor[MAX_ITENS_VETOR];
   int posVetor;                  // índice do Vetor
   int qtdInteirosLidosArqOriginal;  // quantidade real lida
   int nValorParaIntercalar;         // valor a intercalar, lido de
fpTMP
   int IOResult;                  // código de status do SisOp de
operação com arquivos

   // abrindo o arquivo original para leitura
   if((fpDAT = fopen(f_dat,F_READONLY)) == NULL) {
      printf("\nErro ao abrir %s para leitura.\n\n", f_dat);
      exit(1);
   }

   // zeramos o arquivo temporário
   if((fpTMP = fopen(f_tmp,F_WRITE)) == NULL) {
      printf("\nErro ao abrir %s para gravação.\n\n", f_tmp);
      exit(1);
   }
   fclose(fpTMP);
```

```
    // zeramos o arquivo final
    if((fpEND = fopen(f_end,F_WRITE)) == NULL) {
        printf("\nErro ao abrir %s para gravação.\n\n", f_end);
        exit(1);
    }
    fclose(fpEND);

    while(HouverArquivoASerLido(fpDAT)) {
        // a intenção é ler MAX_ITENS_VETOR inteiros, mas
podem ser menos
        printf("Tentando ler %d inteiro(s)...\n",
MAX_ITENS_VETOR);

        // tenta ler MAX_ITENS_VETOR inteiros
        // qtdInteirosLidosArqOriginal = quantidade real de
inteiros lidos
        qtdInteirosLidosArqOriginal = 0;
        while((HouverArquivoASerLido(fpDAT)) &&
(qtdInteirosLidosArqOriginal < MAX_ITENS_VETOR)) {
            fscanf(fpDAT, "%d",
&Vetor[qtdInteirosLidosArqOriginal++]);
        }
        printf("%d de %d inteiro(s) lido(s)...\n\n",
qtdInteirosLidosArqOriginal, MAX_ITENS_VETOR);

        printf("Vetor desordenado: %d elemento(s)...\n",
qtdInteirosLidosArqOriginal);
        for(posVetor = 0; posVetor < qtdInteirosLidosArqOriginal;
posVetor++) {
            printf("%02d ", Vetor[posVetor]);
            if((posVetor % 19 == 0) && (posVetor != 0))
putchar('\n');
        }
        printf("\n\n");

        // ordenando o vetor
        printf("Ordenando o vetor...\n\n");
        qsort(Vetor, qtdInteirosLidosArqOriginal, sizeof(int),
compare);

        printf("Vetor ordenado: %d elemento(s)...\n",
qtdInteirosLidosArqOriginal);
        for(posVetor = 0; posVetor < qtdInteirosLidosArqOriginal;
posVetor++) {
            printf("%02d ", Vetor[posVetor]);
            if((posVetor % 19 == 0) && (posVetor != 0))
putchar('\n');
        }
        printf("\n\n");

        // intercalando o vetor e o arquivo
        printf("Intercalando o vetor: %d elemento(s)...\n\n",
qtdInteirosLidosArqOriginal);

        // abrindo o arquivo temporário para leitura
        // na primeira execução do while este arquivo estará
vazio
        // nas próximas passagens este arquivo conterá os
elementos intercalados
        if((fpTMP = fopen(f_tmp,F_READONLY)) == NULL) {
            printf("\nErro ao abrir %s para leitura.\n\n", f_tmp);
            exit(1);
        }

        // abrindo o arquivo final para gravação
        if((fpEND = fopen(f_end,F_APPEND)) == NULL) {
            printf("\nErro ao abrir %s para gravação.\n\n",
f_end);
            exit(1);
        }

        posVetor = 0;
        if(HouverArquivoASerLido(fpTMP)) fscanf(fpTMP, "%d",
&nValorParaIntercalar);

        while((posVetor < qtdInteirosLidosArqOriginal) ||
(HouverArquivoASerLido(fpTMP))) {
            if((posVetor < qtdInteirosLidosArqOriginal) &&
(HouverArquivoASerLido(fpTMP))) {
                if((Vetor[posVetor] > nValorParaIntercalar) &&
(HouverArquivoASerLido(fpTMP))) {
                    if(DEBUG) { printf("de temp: Vetor[%d]= %d,
f_tmp= %d", posVetor, Vetor[posVetor], nValorParaIntercalar);
getchar(); }
                    fprintf(fpEND, "%05d ", nValorParaIntercalar);
                    fscanf(fpTMP, "%d", &nValorParaIntercalar); //
avança no arquivo temporário
                }
                else {
                    if(DEBUG) { printf("de vet : Vetor[%d]= %d,
f_tmp= %d", posVetor, Vetor[posVetor], nValorParaIntercalar);
getchar(); }
                    fprintf(fpEND, "%05d ", Vetor[posVetor++]); //
avança no Vetor
                }
            }
            else
            if(posVetor >= qtdInteirosLidosArqOriginal) { // Vetor
já está totalmente intercalado
                if(DEBUG) { printf("so temp: Vetor[%d]= %d, f_tmp=
%d", posVetor, Vetor[posVetor], nValorParaIntercalar);
getchar(); }
                fprintf(fpEND, "%05d ", nValorParaIntercalar);
                if(HouverArquivoASerLido(fpTMP)) fscanf(fpTMP,
"%d", &nValorParaIntercalar); // avança no arquivo
temporário
            }
            else
            if(!HouverArquivoASerLido(fpTMP)) { // Arquivo
temporário já está totalmente intercalado
                if(DEBUG) { printf("so vet : Vetor[%d]= %d, f_tmp=
%d", posVetor, Vetor[posVetor], nValorParaIntercalar);
getchar(); }
                fprintf(fpEND, "%05d ", Vetor[posVetor++]); //
avança no Vetor
            }
        }

        printf("Intercalados %d inteiros em %s...\n\n",
qtdInteirosLidosArqOriginal, f_end);
```

```
        // apagamos o arquivo temporário (já intercaldo no
arquivo final)
        fclose(fpTMP);
        IOResult = remove(f_tmp);
        if(IOResult == 0)
            printf("Arquivo temporario %s removido com
sucesso...\n", f_tmp);
        else printf("Erro ao deletar o arquivo temporario
%s...\n", f_tmp);

        // renomeamos o arquivo final (atual) para temporário,
para a nova rodada de intercalação
        fclose(fpEND);
        IOResult = rename(f_end, f_tmp);
        if(IOResult == 0)
            printf("Arquivo %s renomeado para %s com
sucesso...\n", f_end, f_tmp);
        else printf("Erro ao renomear o arquivo %s para
%s...\n", f_end, f_tmp);
    }

    fclose(fpDAT);

    // finalmente obtemos o arquivo final ordenado
    fclose(fpEND);
    fclose(fpTMP);
    IOResult = rename(f_tmp, f_end);
    if(IOResult == 0)
        printf("Arquivo %s renomeado para %s com
sucesso...\n", f_tmp, f_end);
    else printf("Erro ao renomear o arquivo %s para %s...\n",
f_tmp, f_end);

    return 0;
}
```

4 Exercícios Propostos

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

2. Usando o algoritmo da *listagem 2*, faça uma modificação para que ordene a estrutura **enderecos** (Tutorial AED 011) pela chave composta {estado, cep, rua, nome}. Baixe o arquivo listagem.dat.

5 Terminamos

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

Corra para o próximo tutorial.