COMBSORT

O que é, como funciona e complexidade: O combsort é um algoritmo de ordenação que foi proposto em 1980 por Włodzimierz Dobosiewicz. Ele é uma variação do algoritmo bubblesort, mas é mais eficiente em casos médios e piores.

O algoritmo combsort trabalha da seguinte maneira: ele compara elementos que estão separados por uma lacuna (gap) inicialmente grande. A lacuna é reduzida a cada iteração até que se torna 1, no qual o algoritmo é equivalente ao bubblesort.

A vantagem do combsort em relação ao bubblesort é que a lacuna utilizada permite que elementos distantes sejam comparados e trocados. Com isso, o algoritmo é capaz de mover elementos que estão muito distantes do seu local correto de forma mais rápida do que o bubblesort.

O desempenho do combsort é considerado muito bom em comparação com outros algoritmos de ordenação como o mergesort e o quicksort. Ele tem complexidade de tempo O(n log n) no melhor caso e O(n^2) no pior caso, mas na prática ele tende a ser mais rápido do que esses algoritmos em casos médios e piores.

Documentação do código:

O primeiro arquivo a ser abordado é, **aluno.h**,o mais simples entre todos, já que tem como função apenas receber os protótipos das funções.

```
1   typedef struct aluno Aluno;
2   Aluno *cria_Aluno();
3   void combSort(char lista[][50], int n);
4   int contador();
```

O arquivo possui apenas estas **4 linhas** de código, não existe nenhum ***include*** dentro do mesmo, pois por ser o arquivo de cabeçalho, não vão funções propriamente ditas nele.

O próximo arquivo se trata do **aluno.c**, que é onde todas as funções serão escritas. O arquivo completo possui **107 linhas** incluindo os comentários. No primeiro momento são incluidas as bibliotecas necessárias e criada a estrutura de alunos.

```c
#include <stdio.h> //Biblioteca de entrada e saída.
#include <stdlib.h> //Biblioteca de funções para alocação dinâmica.
#include <string.h> //Biblioteca de funções para manipular strings.
#include <unistd.h> //Biblioteca para criar e acessar ficheiros.
#include "aluno.h" //Biblioteca criada.
//implementação da struct alunos.
struct aluno
{
    char nome[50];
    char matricula[15];
    char documento[20];
};
```

Como pode ser visto o **aluno.h** é incluído neste arquivo, pois é nele que o **aluno.c** se baseia para fazer as funções. Para se definir a estrutura do tipo aluno é usada uma estrutura de formato ***struct aluno{ <elementos da estrutura> }***, neste caso não é necessário usar o comando ***typedef***, pois o mesmo já se encontra no cabeçalho **aluno.h**.

A estrutura do tipo aluno recebe nome, matrícula e documento, todos do tipo caractere, representado por ***char***, pois além de facilitar a organização, também previne as inconsistências do programa, tais como iniciar a matrícula e o documento como 0.

Agora seguindo para a parte em que são atribuídos os elementos dos alunos.

```c
//implementação da função cria_Aluno
Aluno *cria_Aluno(Aluno *aluno)
{
    //Alocando a memoria
    aluno = (Aluno *)malloc(sizeof(Aluno));
    if (aluno == NULL)
    {
        printf("ERRO!\n");
        exit(1);
    }
     //Imprime as seguintes menssagens na tela do usuário.
    //Ler as informações digitadas pelo usuário.
    printf("Informe o nome do aluno: \n");
    scanf(" %[^\n]s", aluno->nome);
    printf("Informe a matricula do aluno: \n");
    scanf(" %[^\n]s", aluno->matricula);
    printf("Informe o documento do aluno: \n");
    scanf(" %[^\n]s", aluno->documento);

    FILE *aluno_txt;

    aluno_txt = fopen("Alunos.txt", "at");
//Escreve os dados que o usuário digitou no arquivo.
    fprintf(aluno_txt, "Nome: %s\tMatricula: %s\tDocumento: %s\n", aluno->nome, aluno->matricula, aluno->documento);
//fecha o arquivo
    fclose(aluno_txt);
//Imprime as seguintes menssagens na tela do usuário.
    printf("Aluno criado com sucesso! \n");

    return (aluno);
}
```

Primeiramente é escrita a função da exata mesma forma que se encontra no **aluno.h**, logo em seguida um aluno é alocado dinamicamente, ocorrendo logo em seguida se a alocação teve êxito, caso não tenha, o programa é encerrado. A seguir

são coletados os dados do aluno que serão repassados pelo usuário. Em seguida é criado a variável do tipo ***FILE***, que indica que é uma variável que se refere a um arquivo. Usando a função ***fopen*** o arquivo é aberto em modo de edição, mais a frente será explicado o motivo de ser aberto assim, logo após usando o ***fprintf*** é escrito no arquivo as informações do aluno. Usando o ***fclose*** o arquivo é fechado e é mostrado uma mensagem de êxito ao usuário. A função então retorna o aluno.

Agora é aplicado o combsort em si.

```
44  //Aplicando no combsort
45  void combSort(char lista[][50], int n) {
46      FILE* abrir; //c1
47      int lacuna = n; //c2
48      int trocado = 1;//c3
49      int i, j; //c3
50      char temp[50]; //c4
51      int controle = 0; //c4
52
53      while (lacuna > 1 || trocado == 1) { //c5*n
54          lacuna = lacuna / 1.3;
55          if (lacuna < 1) {
56              lacuna = 1;
57          }
58
59          trocado = 0;
60          for (i = 0, j = i + lacuna; j < n; i++, j++) { //n*n
61              if (strcmp(lista[i], lista[j]) > 0) {
62                  strcpy(temp, lista[i]);
63                  strcpy(lista[i], lista[j]);
64                  strcpy(lista[j], temp);
65                  trocado = 1;
66
67              }
68          }
69      }
70      abrir = fopen("Alunos.txt", "wt");
71
72      while(controle < n) {
73          fprintf(abrir, "%s", lista[controle]);
74          controle++;
75      }
76
77      fclose(abrir);
78  }
```

Inicialmente são criadas todas as variáveis a serem utilizadas, as principais são a ***lacuna*** e a ***trocado,*** a primeira tem como função receber o total de linhas que serão percorridas pelo algoritmo, a outra é uma variável de controle que verifica se foi feita a troca de dois elementos. É usado um laço de repetição que roda enquanto ***lacuna*** for maior que 1 ou ***trocado*** ser igual a 1, ou seja, foi efetuada uma troca, logo após a ***lacuna*** é dividida por 1.3, que foi o valor escolhido que mais se encaixa ao que o algoritmo propõe, após isso, se ***lacuna*** for menor do que 1 ela será definida como 1, depois trocado recebe o valor 0, indicando que não houve troca, o programa então entra em outro laço de repetição que faz a parte de trocar os elementos, neste caso, o algoritmo está organizando os alunos por ordem alfabética, assim guardando os elementos em um vetor organizado, ao fim do laço é atribuído 1 para ***trocado,*** se tiver ocorrido a troca.

Após isso é aberto o arquivo no qual os alunos estão em modo de escrita, logo em seguida é escrito no arquivo o vetor de alunos organizados. No fim, o arquivo é fechado.

Por fim é feita a função que conta quantas linhas tem o arquivo que contem os alunos, ou seja, quantos alunos têm no arquivo.

```
88  int contador() {
89
90      FILE* abre;
91      char linha[100];
92      int numLinhas = 0;
93
94      abre = fopen("Alunos.txt", "rt");
95      if(abre == NULL) {
96          printf("ERRO ao abrir o arquivo!");
97          exit(1);
98      }
99
100     while(fgets(linha, 100, abre) != NULL) {
101         numLinhas++;
102     }
103
104     fclose(abre);
105 //Retorna um ponteiro para ponteiro, para o primeiro índice da matriz.
106     return(numLinhas);
107 }
```

Aqui são criadas as variáveis necessárias, em seguida é aberto o arquivo que contém os alunos em modo de leitura, é feita uma verificação se o arquivo foi aberto, caso haja alguma falha o programa é encerrado.

Em seguida é usado um laço de repetição que passa por todas as linhas do arquivo, assim contando elas.

Ao fim da função é retornado a quantidade de linhas do arquivo.

Partindo agora para o último arquivo, vamos para o **main.c** o arquivo onde indica o funcionamento de verdade da aplicação. o arquivo contém ao todo **43 linhas**.

```c
#include <stdio.h>
#include "aluno.c"

int main(){
// declaração das variaveis e criação dos ponteiros
    int opc;
    Aluno * alunos;
    FILE* teste;
    char nomes[20][50];
    int qnt_linhas;

// estrutura de repetição
    while (opc != 2) {



        printf("Digite 1 para criar um aluno, e 2 para fechar o programa: \n");
        scanf("%d", &opc);
// Analisa se o usuário vai criar um aluno
        if (opc == 1) {
            cria_Aluno(alunos);

        }

    }
//declarando variaveis
    qnt_linhas = contador();

    int i = 0;

    teste = fopen("Alunos.txt", "rt");

    while(i < qnt_linhas) {
        fgets(nomes[i], 50, teste);
        i++;
    }

    combSort(nomes, qnt_linhas);



    fclose(teste);

    return 0;
}
```

No início são incluídas as bibliotecas a serem utilizadas, sendo uma delas a **aluno.c**, que é onde se encontram as funções. Já dentro da função principal, a ***main,*** primeiro são criadas as variáveis a serem utilizadas.

Logo em seguida é usado um laço de repetição que permite o usuário criar alunos até digitar a tecla **2**, sendo assim, para criar um aluno ele aperta **1**, ao apertar é usada a função ***cria_Aluno***, após isso é usada a função de ***contador*** para contar quantos alunos foram cadastrados.

Após isso é aberto o arquivo em modo de leitura, que logo em seguida entra em outro laço de repetição para guardar os nomes dos alunos em uma matriz.

Em seguida é aplicado o combsort na matriz preenchida anteriormente. E no fim o programa fecha o arquivo aberto.

REFERÊNCIAS

ChatGPT - utilizado para explicar as funcionalidades e a complexidade do **combsort.**

Repositório - Onde o código se encontra.