/\*\*

\* TRABALHO DE GRAFOS - ENGENHARIA DE COMPUTAÇÃO - TARDE

\*

\* Desenvolvido na IDE DEV-C++

\*

\* EQUIPE:

\* - Cláudia Mariana Costa Maia

\* - Jorge Girão Limaverde Lima

\* - Thalia Nascimento de Sousa Freitas

\* - Ulric Marques Ferreira

\*/

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include <locale.h>

#include <limits.h>

/\* Vértices de grafos são representados por variáveis do tipo 'vertice',

para facilitar a identificação.\*/

#define vertice int

/\* Variáveis do tipo 'INF' representam o valor infinito, que será usado para indicar

que dois vértices não possuem ligação. Também será usado durante o algoritmo de Dijkstra,

para indicar que não é possível chegar a um vértice partindo da origem. \*/

#define INF INT\_MAX

/\* A estrutura 'grafo' representa um grafo.

O campo 'adj' é um ponteiro para a matriz de adjacências do grafo.

O campo 'V' contém o número de vértices e o campo 'A' contém o número de arcos do grafo. \*/

struct grafo {

int V;

int A;

int \*\*adj;

};

/\* Um 'Grafo' é um ponteiro para um grafo, ou seja, um 'Grafo' contém o endereço de um grafo. \*/

typedef struct grafo \*Grafo;

void limpaTela(){

#if defined(\_\_linux\_\_) || defined(\_\_unix\_\_) || defined(\_\_APPLE\_\_)

system("clear");

#endif

#if defined(\_WIN32) || defined(\_WIN64)

system("cls");

#endif

}

/\* A função alocaMatriz() aloca, se houver memória disponível, uma matriz

com linhas 1,2,...,l e colunas 1..c. Cada elemento da matriz recebe valor val. \*/

int \*\*alocaMatriz(int l, int c, int val) {

vertice i, j;

int \*\*m = malloc((l+1) \* sizeof (int \*));

if (m == NULL){

return NULL;

}

for (i = 1; i <= l; ++i){

m[i] = malloc((c+1) \* sizeof (int));

if (m[i] == NULL){

for (j = 1; j <= i; ++j){

free(m[j]);

}

free(m);

return NULL;

}

}

for (i = 1; i <= l; ++i){

for (j = 1; j <= c; ++j){

m[i][j] = val;

}

}

return m;

}

/\* A função inicializaGrafo() constrói um grafo com vértices 1,2,...,V e nenhum arco. \*/

Grafo inicializaGrafo(int V) {

Grafo G = malloc(sizeof \*G);

if(G == NULL){

return NULL;

}

G->V = V;

G->A = 0;

G->adj = alocaMatriz(V, V, INF);

if(G->adj == NULL){

return NULL;

}

return G;

}

/\* A função insereArco() insere um arco v-w com custo c no grafo G.

A função supõe que v e w são distintos, positivos e menores ou iguais a G->V.

Se o grafo já tem um arco v-w, a função não faz nada. \*/

void insereArco(Grafo G, vertice v, vertice w, int c) {

if (G->adj[v][w] == INF) {

G->adj[v][w] = c;

G->A++;

}

}

/\* A função mostraGrafo() imprime, para cada vértice v do grafo G,

em uma linha, todos os vértices adjacentes a v. Também imprime os custos de cada arco. \*/

void mostraGrafo(Grafo G) {

vertice v,w;

printf("\n");

for (v = 1; v <= G->V; ++v) {

printf("%2d:", v);

for (w = 1; w <= G->V; ++w)

if (G->adj[v][w] != INF)

printf(" %2d", w);

printf("\n");

}

printf("\n");

for (v = 1; v <= G->V; ++v) {

for (w = 1; w <= G->V; ++w){

if (G->adj[v][w] != INF){

printf("Arco: %2d->%2d Custo: %2d\n", v, w, G->adj[v][w]);

}

}

}

}

/\* A função auxiliar distanciaMinima() encontra um vértice da fronteira de Z

cuja distância à origem seja mínima.\*/

int distanciaMinima(Grafo G, int dist[], int Z[])

{

int minimo = INF, indiceMenorValor;

vertice v;

for (v = 1; v <= G->V; v++){

if (Z[v] == 0 && dist[v] <= minimo){

minimo = dist[v];

indiceMenorValor = v;

}

}

return indiceMenorValor;

}

/\* A função caminhoMinimo() constrói o caminho de custo mínimo do vértice origem

ao vértice t, que foi calculado pelo algoritmo de Dijkstra, a partir do vetor anterior[]. \*/

void mostraCaminho(int anterior[], int t)

{

if (anterior[t] == - 1){

printf("\nCaminho: ");

return;

}

mostraCaminho(anterior, anterior[t]);

printf("(%d %d) ", anterior[t], t);

}

/\* Recebe um grafo G com custos positivos nos arcos, um vértice s e um vértice t.

Calcula e exibe um caminho de custo mínimo do vértice s ao vértice t, bem como o

seu custo. \*/

void Dijkstra(Grafo G, int s, int t)

{

/\* O vetor dist[] guarda as distâncias mínimas dos vérticesem relação à origem.\*/

vertice dist[(G->V)+1];

/\* Z[i] terá valor 1 se a distância mínima entre o vértice s e o vértice i

já tiver sido calculada. Caso contrário, Z[i] terá valor 0.\*/

vertice Z[(G->V)+1];

/\* O vetor anterior[] é usado para reconstruir o caminho mínimo de s a t.

anterior[i] guarda o vértice w que forma o arco w-i. \*/

vertice anterior[(G->V)+1];

int i, j;

vertice v;

/\* Inicialização dos vetores \*/

for (i = 1; i <= G->V; i++){

anterior[i] = -1;

dist[i] = INF, Z[i] = 0;

}

/\* A distância entre o vértice s e ele mesmo é 0. \*/

dist[s] = 0;

/\* Laço principal\*/

for (j = 1; j <= G->V; j++) {

/\* Escolhe um vértice u na fronteira de Z tal que dist[u] é mínimo. \*/

int u = distanciaMinima(G, dist, Z);

/\* A distância mínima de s a u já foi calculada,

então o vértice passa a fazer parte de Z. \*/

Z[u] = 1;

/\* A fronteira de Z mudou, então é preciso atualizar os vetores

distancia[] e anterior[]. \*/

for (v = 1; v <= G->V; v++){

if (!Z[v] && G->adj[u][v] != INF &&

dist[u] != INF && dist[u]+G->adj[u][v] < dist[v]){

anterior[v] = u;

dist[v] = dist[u] + G->adj[u][v];

}

}

}

/\* Finalmente, mostra o caminho mínimo retornado pelo algoritmo

de Dijkstra e também seu custo, se tal caminho existir. Caso contrário,

exibe uma mensagem dizendo que não existe caminho entre s e t. \*/

if(dist[t] != INF){

printf("\nO custo mínimo do vértice %d ao vértice %d é: %d\n", s, t, dist[t]);

mostraCaminho(anterior, t);

}

else{

printf("\nNão existe caminho entre o vértice %d e o vértice %d.\n", s, t);

}

}

/\* A função caminhoMinimo() lê o nome de um arquivo e, ao localizá-lo, lê o grafo contido no mesmo,

bem como os custos de seus arcos, e encontra um caminho de custo mínimo entre um vértice origem

e um vértice destino utilizando o algoritmo de Dijkstra. \*/

void caminhoMinimo(){

/\* Cabeçalho \*/

limpaTela();

printf("### Receber grafo a partir de um arquivo ###\n\n");

/\* Variáveis para manipulação do arquivo \*/

char \*nomeArquivo;

FILE \*file;

/\* Variáveis para leitura e manipulação dos dados contidos no arquivo \*/

int vertices, arcos, origem, destino;

vertice v, w;

int custo;

Grafo G;

/\* Variáveis auxiliares \*/

int i=0;

nomeArquivo = malloc(sizeof(char)\*35); /\* Aloca string com até 35 caracteres (valor arbitrário) \*/

if(nomeArquivo == NULL){

printf("[x] Falha! Não há memória disponível no sistema");

free(nomeArquivo);

return;

}

setbuf(stdin,NULL);

printf("Digite o nome do arquivo de texto (SEM SUA EXTENSÃO): ");

scanf(" %s", nomeArquivo); //Lê o nome do arquivo;

strcat(nomeArquivo,".txt\0"); //Adiciona a extensão '.txt' mais caractere '\0' (final da string) ao nome do arquivo;

file = fopen(nomeArquivo,"r"); //Abre arquivo no modo de leitura;

if(file == NULL){ /\* Arquivo não encontrado \*/

free(nomeArquivo);

printf("\n\n[!] Arquivo não localizado");

printf("\n[i] Verifique se o arquivo existe ou se você digitou o nome corretamente");

printf("\n\n[i] Pressione <ENTER> para voltar ao MENU DE OPÇÕES...");

setbuf(stdin,NULL);

getchar();

return;

}

else{

printf("\n\tArquivo: %s\t",nomeArquivo);

/\* Lê a primeira linha do arquivo, contendo a quantidade de vértices,

a quantidade de arcos, o vértice origem e o vértice destino. \*/

fscanf(file,"%d %d %d %d",&vertices, &arcos, &origem, &destino);

printf("\nVértices: %d\tArcos: %d\t Origem: %d\t Destino: %d\n", vertices, arcos, origem, destino);

G = inicializaGrafo(vertices);

if(G == NULL ){ /\* Falta de memória \*/

printf("\n[x] Falha! Não há memória disponível no sistema");

printf("\n\n[i] Pressione <ENTER> para voltar ao MENU DE OPÇÕES...");

setbuf(stdin,NULL);

getchar();

return;

}

while(i < arcos){ /\* Lê as próximas m linhas contendo os arcos do grafo \*/

/\* Lê um arco do grafo \*/

fscanf(file,"%d %d %d",&v, &w, &custo);

insereArco(G, v, w, custo);

i++;

setbuf(stdin,NULL);

}

mostraGrafo(G);

/\* Chama a função Dijkstra passando o grafo G, o vértice origem e o vértice destino.

A função devolve um caminho mínimo para sair da origem e chegar ao destino,

e o custo desse caminho. \*/

Dijkstra(G, origem, destino);

fclose(file); //Fecha o arquivo

/\* Espera o usuario pressionar uma tecla para voltar ao menu principal \*/

printf("\n\n[i] Pressione <ENTER> para voltar ao MENU DE OPÇÕES...");

setbuf(stdin,NULL);

getchar();

}

}

/\* A função menuDeOpcoes() exibe o menu de opções, lê a opção escolhida pelo usuário e a retorna. \*/

char menuDeOpcoes(){

char opcao;

//Mostra o menu de opções. Repete o processo enquanto a opção escolhida pelo usuário não for válida.

do{

//Monta menu.

limpaTela();

printf("### GRAFOS ###\n\n");

printf("### MENU DE OPCOES ###\n\n");

printf("C - Caminho mínimo\n");

printf("F - Finalizar\n");

printf("\n[i] Escolha uma opção acima e pressione <ENTER>: ");

setbuf(stdin,NULL);

opcao = toupper(getchar()); //Lê a opção digitada pelo usuário e a transforma em letra maiúscula.

/\* Exibe aviso de advertência, caso a opção escolhida pelo usuário seja inválida \*/

if((opcao!='C') && (opcao!='F')){

printf("#");

printf("\n\n[!] Opção inválida\n");

printf("[i] Pressione <ENTER> e tente novamente...");

setbuf(stdin,NULL);

getchar();

}

//Testa validade. Se a opção escolhida for válida, o laço termina. Caso contrário, mostra o menu novamente.

} while((opcao!='C') && (opcao!='F'));

return opcao;

} /\* Fim menuDeOpcoes() \*/

/\* FUNÇÃO PRINCIPAL \*/

int main(){

setlocale(LC\_ALL, "portuguese");

int finalizarPrograma = 0; //Flag para quando o usuário desejar finalizar o programa.

/\* Laço do-while: Chama a função 'menuDeOpções' e abre a função correspondente à escolha do usuário.

\* Repete o processo enquanto o mesmo não desejar finalizar o programa.

\*/

do{

switch(menuDeOpcoes()){

//Opção 'CAMINHO MÍNIMO'

case 'C':

caminhoMinimo();

break;

//Opção 'FINALIZAR'

case 'F':

printf("Finalizar\n");

finalizarPrograma = 1;

break;

}

//Testa validade para saber se o usuário deseja finalizar o programa.

} while(!finalizarPrograma);

return 0;

} /\* Fim main() \*/