ep1\_13837428.c

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <float.h>

#include "grafo\_listaadj.h"

//#include "grafo\_matrizadj.h"

typedef struct{

int origem;

int destino;

} Caso;

void procuraMenorDaAGMax(int v\_inicio, int v\_destino, int ant[], Peso key[], Peso\* menor){

if(\*menor > key[v\_destino]) \*menor = key[v\_destino];

if(v\_inicio == v\_destino) return;

else procuraMenorDaAGMax(v\_inicio, ant[v\_destino], ant, key, menor);

}

int extraiMax(Grafo\* g, Peso key[], bool esta\_na\_AGMax[]){

float valor\_max\_i = -1.0;

int max;

for (int i = 0; i < g->nVertices; i++){

if (esta\_na\_AGMax[i] == false && key[i] > valor\_max\_i){

valor\_max\_i = key[i];

max = i;

}

}

return max;

}

Peso AGMaxPrim(Grafo\* g, int v\_inicio, int v\_destino){

Peso key[g->nVertices];

int ant[g->nVertices];

bool esta\_na\_AGMax[g->nVertices];

Apontador atual = VERTICE\_INVALIDO;

Apontador adj;

Peso peso;

int vertice;

for (int i = 0; i < g->nVertices; i++){

key[i] = -1.0;

esta\_na\_AGMax[i] = false;

}

//valores da raiz

key[v\_inicio] = 0;

for(int i = 0; i < g->nVertices; i++){

printf("peso[%d]=%.2f\n", i, key[i]);

}

ant[v\_inicio] = -1;

for(int i = 0; i < g->nVertices; i++){

printf("ant[%d]=%d\n", i, ant[i]);

}

//percorrendo os vértices e extraindo o com maior peso

for (int i = 0; i < g->nVertices; i++) {

int u = extraiMax(g, key, esta\_na\_AGMax);

printf("u=%d\n", u);

esta\_na\_AGMax[u] = true;

adj = proxListaAdj(g, atual, u, &peso, &vertice);

while (adj != VERTICE\_INVALIDO){

atual = adj;

if (esta\_na\_AGMax[vertice] == false && peso > key[vertice]){

ant[vertice] = u;

key[vertice] = peso;

}

adj = proxListaAdj(g, atual, u, &peso, &vertice);

}

atual = VERTICE\_INVALIDO;

}

key[v\_inicio] = INFINITY;

for(int i = 0; i < g->nVertices; i++){

printf("peso[%d]=%.2f\n", i, key[i]);

}

Peso menor = INFINITY;

procuraMenorDaAGMax(v\_inicio, v\_destino, ant, key, &menor);

printf("menor=%.2f\n", menor);

return menor;

}

void tamCarreta(Peso resultados[], int consultas){

for(int i = 0; i < consultas; i++){

if(resultados[i] >= 4.5) resultados[i] = 4.5;

if(resultados[i] < 4.5 && resultados[i] >= 4.0) resultados[i] = 4.0;

if(resultados[i] < 4.0 && resultados[i] >= 3.5) resultados[i] = 3.5;

if(resultados[i] < 3.5 && resultados[i] >= 3.0) resultados[i] = 3.0;

if(resultados[i] < 3.0 && resultados[i] >= 2.5) resultados[i] = 2.5;

}

}

int main(int argc, char\*\* argv){

FILE \*fp;

int n\_vertices, n\_arestas, consultas;

fp = fopen(argv[1], "r");

fscanf(fp, "%d %d %d", &n\_vertices, &n\_arestas, &consultas);

Grafo g;

if(!inicializaGrafo(&g, n\_vertices)) return fprintf(stderr, "ERRO na inicializacao.");

//imprimeGrafo(&g);

int v1, v2;

float peso;

for(int i = 0; i < n\_arestas; i++){

fscanf(fp, "%d %d %f", &v1, &v2, &peso);

printf("passada %d: %d %d %f\n",i, v1, v2, peso);

if(!insereAresta(v1, v2, peso, &g)) return fprintf(stderr, "ERRO na inicializacao.");

}

//imprimeGrafo(&g);

Caso \*casos = (Caso\*) malloc(consultas \* sizeof(Caso));

for(int i = 0; i < consultas; i++){

fscanf(fp, "%d %d", &casos[i].origem, &casos[i].destino);

//printf("posicao %d, origem:%d, destino:%d\n", i, casos[i].origem, casos[i].destino);

}

fclose(fp);

Peso resultados[consultas];

Peso menor;

for(int i = 0; i < consultas; i++){

menor = AGMaxPrim(&g, casos[i].origem, casos[i].destino);

resultados[i] = menor;

}

printf("menor = %.2f\n", menor);

tamCarreta(resultados, consultas);

for(int i = 0; i < consultas; i++){

printf("resultado[%d]=%.2f\n", i, resultados[i]);

}

fp = fopen("saida.txt", "w");

for(int i = 0; i < consultas; i++)

fprintf(fp, "%.1f\n", resultados[i]);

fclose(fp);

imprimeGrafo(&g);

}

grafo\_listaadj.c

#include <stdio.h>

#include <stdbool.h>

#include <stdlib.h>

#include "grafo\_listaadj.h"

bool inicializaGrafo(Grafo\* g, int nv){

if(nv <= 0){

fprintf(stderr, "ERRO na chamada de inicializaGrafo: Numero de vertices manor que zero.\n");

return false;

}

g->nVertices = nv;

if(!(g->lista\_adj = (Apontador\*) calloc(nv+1, sizeof(Apontador)))){

fprintf(stderr, "ERRO na alocacao de memoria na funcao inicializaGrafo\n");

return false;

}

g->nArestas = 0;

return true;

}

bool verificaValidadeVertice(int v, Grafo\* g){

if(v > g->nVertices){

fprintf(stderr, "ERRO na chamada de verificaValidadeVertice: vertice %d esta acima do numero de vertices total %d.\n", v, g->nVertices);

return false;

}

if(v < 0){

fprintf(stderr, "ERRO na chamada de verificaValidadeVertice: vertice %d deve ser positivo.\n", v);

return false;

}

return true;

}

bool insereAresta(int v1, int v2, Peso peso, Grafo\* g){

if(!(verificaValidadeVertice(v1, g) && verificaValidadeVertice(v2, g))){

fprintf(stderr, "ERRO na validação dos vértices.");

return false;

}

if(v1 == v2){

fprintf(stderr, "ERRO: vértices iguais, self-loops não permitidos!");

return false;

}

Apontador novo\_v1;

if(!(novo\_v1 = (Apontador) calloc(1, sizeof(Aresta)))){

fprintf(stderr, "ERRO na alocação da nova Aresta");

}

novo\_v1->vdest = v2;

novo\_v1->peso = peso;

novo\_v1->prox = g->lista\_adj[v1];

g->lista\_adj[v1] = novo\_v1;

Apontador novo\_v2;

if(!(novo\_v2 = (Apontador) calloc(1, sizeof(Aresta)))){

fprintf(stderr, "ERRO na alocação da nova Aresta");

}

novo\_v2->vdest = v1;

novo\_v2->peso = peso;

novo\_v2->prox = g->lista\_adj[v2];

g->lista\_adj[v2] = novo\_v2;

g->nArestas++;

return true;

}

void imprimeGrafo(Grafo\* g){

Apontador p;

for(int i = 0; i < g->nVertices; i++){

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

p = g->lista\_adj[i];

while(p){

printf("(%d, %f); ", p->vdest, p->peso);

p = p->prox;

}

printf("\n");

}

}

Apontador proxListaAdj(Grafo\* g, Apontador atual, int v, Peso\* peso, int\* vertice){

if(atual == NULL){

//fprintf(stderr, "ERRO: Apontador atual é nulo");

atual = g->lista\_adj[v];

\*peso = atual->peso;

\*vertice = atual->vdest;

return atual;

}

if(atual->prox != ARESTA\_NULA){

\*peso = atual->prox->peso;

\*vertice = atual->prox->vdest;

return atual->prox;

}

return NULL;

}

grafo\_listaadj.h

#include <stdbool.h>

#define VERTICE\_INVALIDO NULL

#define ARESTA\_NULA NULL

typedef float Peso;

typedef struct str\_aresta{

int vdest;

Peso peso;

struct str\_aresta\* prox;

} Aresta;

typedef Aresta\* Apontador;

typedef struct{

Apontador\* lista\_adj;

int nVertices;

int nArestas;

} Grafo;

bool inicializaGrafo(Grafo\* g, int nv);

bool verificaValidadeVertice(int v, Grafo\* g);

bool insereAresta(int v1, int v2, Peso peso, Grafo\* g);

Apontador proxListaAdj(Grafo\* g, Apontador atual, int v, Peso\* peso, int\* vertice);

void imprimeGrafo(Grafo\* g);

grafo\_matrizadj.c

#include <stdio.h>

#include <stdbool.h>

#include "grafo\_matrizadj.h"

bool inicializaGrafo(Grafo\* g, int nv){

if(nv > MAX\_VERTICES){

fprintf(stderr, "ERRO na chamada de inicializaGrafo: numero de vertices maior que o maximo permitido de %d.\n", MAX\_VERTICES);

return false;

}

if(nv <= 0){

fprintf(stderr, "ERRO na chamada de inicializaGrafo: numero de vertices manor ou igual a zero.\n");

return false;

}

g->nVertices = nv;

g->nArestas = 0;

for(int i = 0; i < g->nVertices; i++){

for(int j = 0; j < g->nVertices; j++){

g->matriz\_adj[i][j] = ARESTA\_NULA;

}

}

return true;

}

bool verificaValidadeVertice(int v, Grafo\* g){

if(v > g->nVertices){

fprintf(stderr, "ERRO na chamada de verificaValidadeVertice: vertice %d esta acima do numero de vertices total %d.\n", v, g->nVertices);

return false;

}

if(v < 0){

fprintf(stderr, "ERRO na chamada de verificaValidadeVertice: vertice %d deve ser positivo.\n", v);

return false;

}

return true;

}

bool insereAresta(int v1, int v2, Peso peso, Grafo\* g){

if(!(verificaValidadeVertice(v1, g) && verificaValidadeVertice(v2, g))){

fprintf(stderr, "ERRO na validação dos vértices.");

return false;

}

if(v1 == v2){

fprintf(stderr, "ERRO: vértices iguais, self-loops não permitidos");

return false;

}

g->matriz\_adj[v1][v2] = peso;

g->matriz\_adj[v2][v1] = peso;

g->nArestas++;

return true;

}

void imprimeGrafo(Grafo\* g){

for(int i = 0; i < g->nVertices; i++){

for(int j = 0; j < g->nVertices; j++){

printf("%f ", g->matriz\_adj[i][j]);

}

printf("\n");

}

}

Apontador proxListaAdj(Grafo\* g, Apontador atual, int v, Peso\* peso, int\* vertice){

if(!verificaValidadeVertice(v, g)) return VERTICE\_INVALIDO;

atual++;

while((atual < g->nVertices) && (g->matriz\_adj[v][atual] == ARESTA\_NULA)) atual++;

if(atual >= g->nVertices){

return VERTICE\_INVALIDO;

}

\*peso = g->matriz\_adj[v][atual];

\*vertice = atual;

return atual;

}

grafo\_matrizadj.h

#include <stdbool.h>

#include <stdio.h>

#define MAX\_VERTICES 100

#define ARESTA\_NULA -1.0

#define VERTICE\_INVALIDO -1

typedef float Peso;

typedef struct {

Peso matriz\_adj[MAX\_VERTICES][MAX\_VERTICES];

int nVertices;

int nArestas;

} Grafo;

typedef int Apontador;

bool inicializaGrafo(Grafo\* g, int nv);

bool verificaValidadeVertice(int v, Grafo\* g);

bool insereAresta(int v1, int v2, Peso peso, Grafo\* g);

Apontador proxListaAdj(Grafo\* g, Apontador atual, int v, Peso\* peso, int\* vertice);

void imprimeGrafo(Grafo\* g);