TRILHAS DE APRENDIZAGEM

Grupo:

João Vítor da Silva – 12011BSI223

Israel Lúcio De Lima Vaz - 12011BSI220

Pedro Milvar Santos Vieira - 11921BSI207

A) Criação do grafo, com inserção/remoção de vértices e arestas. Os vértices podem ser estruturas ou podem ser armazenados em vetor e referenciados a partir de um número no grafo. Ainda, pode-se usar matriz ou lista de adjacências.

```c
#include <stdlib.h>
#include <stdio.h>

#define grauZero 0 //vamos definir aqui as cores apenas para fim ditaticos
#define grauUm 1
#define grauDois 2

struct adjacencia{
    int vertice;
    int peso;
    struct adjacencia *prox;
};
typedef struct adjacencia adjacencia;

struct vertice{
    adjacencia *head;
};
typedef struct vertice vertice;

struct Grafo{
    int arestas;
    int vertices;
    vertice *adj;
};
typedef struct Grafo Grafo;


Grafo *grafo_create(int v){
    Grafo *g;
    g = malloc(sizeof(Grafo));

    //g->recurso = rec;
    g->vertices = v;
    g->arestas = 0;
    g->adj = malloc(sizeof(vertice));

    for(int i=0; i<v; i++){
        g->adj[i].head = NULL;
    }

    return g;
}
```

```c
adjacencia *criaAdj(int v, int peso){
    adjacencia *aux;
    aux = malloc(sizeof(adjacencia));

    aux->vertice = v;
    aux->peso = peso;
    aux->prox = NULL;

    return aux;
}


int addAresta(Grafo *g, int vi, int vf, int p){
    if(g == NULL)
        return 0;
    if((vf < 0 ) || (vf >= g->vertices))
        return 0;
    if((vi < 0) || (vf >= g->vertices))
        return 0;

    adjacencia *novo = criaAdj(vf, p);

    novo->prox = g->adj[vi].head;
    g->adj[vi].head = novo;
    g->arestas++;

    return 1;
}
```

```c
int removeAresta(Grafo *g,int vi, int vf){
    if(g == NULL)
        return 0;
    if((vf < 0 ) || (vf >= g->vertices))
        return 0;
    if((vi < 0) || (vf >= g->vertices))
        return 0;

    adjacencia *aux = g->adj[vi].head, *aux2, *aux3 = g->adj[vi].head;
    int cont=1;

    while(aux){
        if(aux->vertice == vf){
            if(aux->prox != NULL){
                aux2 = g->adj[vi].head->prox->prox;         //aux2 = null
                g->adj[vi].head = g->adj[vi].head->prox;    //head = segundo
                g->adj[vi].head->prox = aux2;               //
                g->arestas--;

                 if(cont == 1){
                    return 1;
                }break;

            } else{
                    if(cont == 1){
                        g->adj[vi].head = g->adj[vi].head->prox;
                        g->arestas--;
                        return 1;
                    }
                    g->adj[vi].head = g->adj[vi].head->prox;
                    g->arestas--;
                    break;
                }
        }
        cont++;
        aux = aux->prox;
        g->adj[vi].head = g->adj[vi].head->prox;
    }

    if(cont == 2){
        aux3->prox = NULL;
        g->adj[vi].head = aux3;
    }
```

```
    }
    g->adj[vi].head = aux3;
    return 0;
}

int removeVertice(Grafo *g, int vi){
    if(g == NULL){
        return 0;
    }
    if(vi < 0)
        return 0;

    g->adj[vi].head = NULL;
    //g->adj[vi] = NULL;

    return 1;
}
```

Atualmente os grafos são amplamente usados na matemática, mas sobretudo na área programação. Formalmente, um grafo é uma coleção de vértices (V) e uma coleção de arestas (A) constituídos por pares de vértices. É uma estrutura usada para representar um modelo em que existem relações entre os objetos de uma certa coleção.

B) Busca do vértice de maior grau, que, para a trilha, representa um recurso com peso importante no fluxo.

```
void maiorGrau(Grafo *g){

    int qtd=0, maior=0, maiorV=0;
    for(int i=0; i<g->vertices; i++){
        adjacencia *aux = g->adj[i].head;
        while(aux){
            aux = aux->prox;
            qtd++;
        }

        if(maior < qtd){
            maior = qtd;
            maiorV = i;
        }
        qtd=0;
    }

    printf("Vertice com mais adjacencias: v%d\n", maiorV);
}
```

A busca de maior grau pode ser utilizada para o entendimento de qual ponto de maior relevância em um determinado caminho, pois ele é um ponto de maior relevância neste grafo. Como exemplo podemos olhar as cidades escolhidas como capital de um país a qual ela seria uma referência de trajeto para os demais locais pois ela é um ponto de convergência das arestas (rodovias/malhas aéreas)

C) Dados dois recursos (vértices), verificar se existe caminho entre os mesmos.

D) A partir de um vértice, encontrar o menor caminho para os outros vértices a ele conectados.

```
int procuraMenorDistancia (int *dist, int *visitado, int NV ){
        int i, menor = -1, primeiro = 1;
        for(i = 0; i < NV; i++) {
                if(dist[i] >= 0 && visitado[i] == 0) {
                if(primeiro) {
                        menor = i;
                        primeiro = 0;
                } else{
                        if(dist[menor] > dist[i])
                                menor = i;
                }
        }
}
    return menor;
}
```

```
void menorCaminho_Grafo (Grafo *gr, int ini, int *ant, int *dist) {
        int i, cont, NV, ind, *visitidao, u;
        cont= NV = gr->nro+vertices;
        visitado = (int*) malloc(NV * sizeof(int));

        for(i=0; i < NV; i++) {
            ant[1] = -1;
            dist[i] = -1;
            visitado[i] = 0;
    }

    dist[ini] = 0;
    while(cont > 0){
            u = procuraMenorDistancia(dist, visitado, NV);
            if (u == -1)
                    break;
            visitado[u] = 1;
            cont--;
        }

        free(visitado);
    ind = gr->arestas[u][i];
        if(dist[ind] < 0) {
                dist[ind] = dist[u] + 1;
                ant[ind] = u;
    }else {
        if(dist[ind] > dist[u] + 1) {
                dist[ind] = dist[u] + 1;
                ant[ind] = u;
        }
}
```

Ao saber o menor caminho entre dois vértices podemos saber qual o menor custo para se passar entre um e outro. A aplicação prática disso se deve quando pensamos em um determinado trajeto até um ponto específico e procuramos a menor rota de chegada entre esses dois pontos, pois isso se definiria em um custo menor.

E) Usando busca em profundidade, encontrar recursos fortemente conectados (Algoritmo)

```
void profundidade (Grafo *g){
        int cor[g->vertices];//crio um vetor com arranjo de cores (o valor tem que bater com o do grafo)

        int u;
        for(u=0;u<g->vertices;u++){ //inicialmente todos vérties são brancos
                cor[u] = grauZero;
        }

        for(u=0;u<g->vertices;u++){
                if (cor[u] == grauZero) // se for branco visita todos em profundida
                visitaP(g,u,cor); // repare que é recursivo, visita até o final depois volta
        }
}

void visitaP(Grafo *g, int u, int *cor){
        cor[u] = grauUm; //visitar um nó marca como amarelo
        adjacencia *v = g->adj[u].head; //visito adjacencia

        while (v){ //então visitamos sua adjacencia, aresta u e v recursivamente
                if(cor[v->vertice]==grauZero) //se a cor for branco eu visito
                visitaP(g,v->vertice,cor); //veja que visito chamando a função de novo, ou seja recursivo
                v = v->prox;
        }

        cor[u] = grauDois;
}
```

Achar componentes conectados. componentes fortemente conectados, ordenação topológica e resolução de puzzles como labirintos e quebra-cabeças.

F) Impressão do grafo.

```
void imprime(Grafo *g){
    printf("Vertices: %d. Arestas: %d. \n",g->vertices,g->arestas);

    for(int i=0; i<g->vertices; i++){
        printf("v%d: ", i);
        adjacencia *aux = g->adj[i].head;

            while(aux){
                printf("v%d(%d) ",aux->vertice,aux->peso);
                aux = aux->prox;
            }
            printf("\n");
    }
}
```

A impressão do grafo é a forma mais concreta de visualização de todas as buscas anteriores, pois com ele fica palpável as informações buscadas.