Universidade Federal de Uberlândia
Faculdade de Computação

Algoritmos e Estruturas de Dados II
Trilha de aprendizado

Alunos: Isabelli Prudêncio Tedeschi Davi **Matrícula:** 12011BCC018

1. DESCRIÇÃO DO PROBLEMA

1.1 Descrição do problema geral

O objetivo do trabalho “Trilha de Aprendizagem” é criar um sistema, em forma de grafos direcionados e ponderados, que representa a trilha de um aluno na disciplina no sistema Moodle.

São considerados grafos direcionados e ponderados, pois há uma direção que o aluno trilha e há pesos nas arestas.

Nessa trilha, o aluno passa por estruturas, Recursos, que é configurado por meio de um Nome, um Tipo e uma Ação, a quantidade de vezes que um aluno passa para um desses recursos é definido por um número, que representa o peso da aresta.

Por fim, foi pedido que se criasse um grafo direcionado e ponderado que contivesse as informações lidas de um arquivo texto, que contém o resumo dos acessos do aluno à página da disciplina no sistema Moodle e após essa formação do grafo, fosse possível criar opções para modifica-lo ou apresentar suas informações.

1.2 Descrição do problema de forma específica

O programa inicia com a leitura de um arquivo texto, cujas informações são colocadas em um grafo.

Após a primeira leitura e formação do grafo, é apresentado opções ao usuário para escolher a próxima ação de manipulação do mesmo.

As opções se resumem em adicionar e remover tanto arestas quanto vértices; buscar, dentre os vértices, aquele de maior grau; verificar se existe caminho entre dois recursos; encontrar o menor caminho para os outros vértices a ele conectados; encontrar recursos fortemente conectados e, por último, há a opção de impressão do grafo.

1.3 Descrição do método usado para criar o grafo

Devido às opções disponibilizadas ao usuário, foi utilizado o método de criação e manipulação de grafos de Lista de Adjacências.

Esse método, além de facilitar certas aplicações das escolhas do menu, pode ser mais adequada, caso o grafo não seja muito denso.

2. INFORMAÇÕES SOBRE O PROGRAMA (Estruturas)

2.1 Identificação dos recursos – “registro.h”

O tipo do recurso é identificado pelo nome Registro e este contém as informações: nome, tipo e ação.

```
struct Registro{
    char nome[20];
    char tipo[20];
    char acao[20];
};
typedef struct Registro Registro;
```

2.2 Formatação do grafo – “grafo.h”

No grafo, há uma parte da estrutura que contém as informações gerais do mesmo, ou seja, quantos arcos e vértices possui e um ponteiro, esse último sendo uma ligação para a próxima estrutura, onde são adicionados os vértices. Essa estrutura é identificada pelo tipo *struct grafo Grafo.*

```
struct grafo {
   int NumVert;
   int NumArco;
   struct noVert *vertices;
};
typedef struct grafo *Grafo;
```

A estrutura *struct noVert,* contém todos os vértices que fazem parte do grafo. Cada elemento dessa estrutura contém o Registro do vértice, um id, para identificá-lo, uma estrutura que o liga aos próximos vértices e outra estrutura que o liga à outra ponta da aresta.

```
struct noVert {
   int id; //
   Registro vert; //
   struct noVert *prox;
   struct noAdj *ladj;
};
```

A estrutura do tipo *struct noAdj* recebe informações do Registro do vértice e um id, porém, como contém a “outra ponta da aresta”, só tem uma estrutura que liga os elementos aos outros vértices que também estão conectados ao vértice da segunda estrutura mencionada, *noVert.*

```
struct noAdj {
   int id;
   Registro vert; //
   struct noAdj *prox;
   int peso;
};
```

2.3 Estrutura usada na função Busca em Profundidade Tempo

A estrutura é do tipo *struct visitaTempo* e contém as informações do vértice, seu tempo de descoberta e tempo de finalização.

```
struct visitaTempo{
  Registro vert; //
  int tempoDescoberta;
  int tempoFinalizacao;
  int id;
};
```

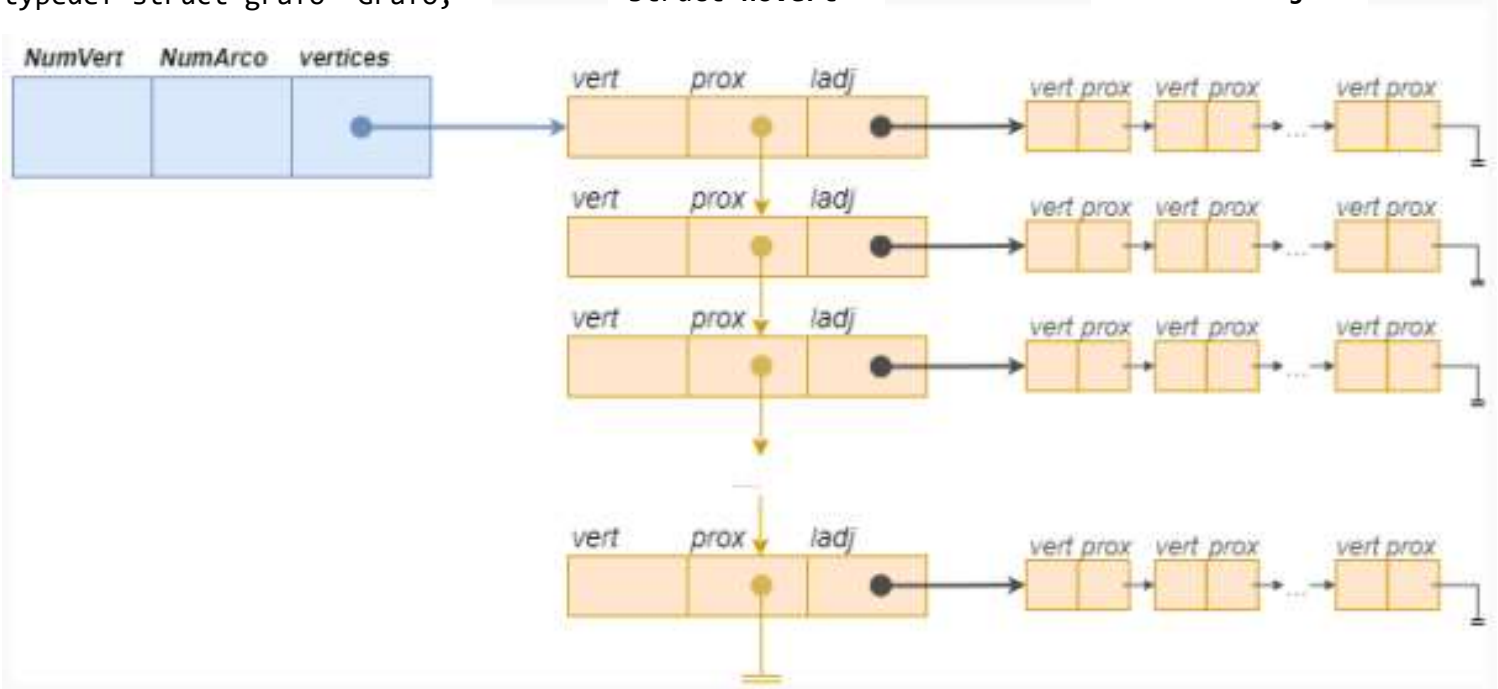
2.4 Formatação da Pilha

A Pilha, por exemplo, utiliza a forma da estrutura No, que tem tipo, *struct No*. Nela, as informações do vértice, seu Registro, e o próximo elemento conectado a ele, são guardados.

```
typedef struct No{
  Registro vert; //
  Struct No *prox;

}No;
```

Representação da Lista de Adjacências, que forma o grafo

`typedef struct grafo *Grafo;` `struct` **`noVert`** `struct` **`noAdj`**



3. SOLUÇÃO

3.1

Os dados devem ser fornecidos ao programa em um arquivo texto que contém o resumo dos acessos do aluno à página da disciplina no sistema Moodle. Por exemplo, cada linha/bloco do arquivo pode fornecer uma tripla: *recurso_1, peso, recurso_2* em que cada recurso é descrito por seu nome, tipo e ação. Dois recursos representam o mesmo vértice apenas se os três campos forem iguais.

Para ler o arquivo texto e guardar as informações necessárias, primeiro foi necessário abrir o arquivo:

```
FILE *file = fopen("texto.txt", "r");
```

Após ser identificado qual arquivo texto vai ser aberto e lido, as informações deste foram lidas linha a linha e colocadas cada informação, separadas por vírgula, em seus respectivos tipos.

```
while (fscanf(file, "%[^\n] ", linha) != EOF){

    token = strtok(linha, ",");
    strcpy(aux[0].nome, token);
    token = strtok(NULL, ",");
    strcpy(aux[0].tipo, token);
    token = strtok(NULL, ",");
    strcpy(aux[0].acao, token);

    token = strtok(NULL, ",");
    peso[j] = atoi(token);

    token = strtok(NULL, ",");
    strcpy(aux[1].nome, token);
    token = strtok(NULL, ",");
    strcpy(aux[1].tipo, token);
    token = strtok(NULL, ",");
    strcpy(aux[1].acao, token);
}
```

Figura 1 – Leitura dos tipos

Na Figura 1, acima, as informações são inseridas em um Registro auxiliar, que ajuda adicionar os vértices de maneira correta (uma explicação mais detalhada do modo de inserção, virá a seguir).

3.1.1 Criação do grafo

A função de criação não possui nada como entrada, mas retorna um grafo nulo, para isso foi necessário alocar um elemento G com o tipo Grafo e inicializa-lo com o número de arestas e vértices como zero.

(GRAFO.C)

```
Grafo criarGrafo() {

    Grafo G;
    G = (Grafo) malloc(sizeof (Grafo));
    if (G!=NULL){
      G->NumArco = 0; // NÚMERO DE ARESTAS
      G->NumVert = 0; // NÚMERO DE VÉRTICES
      G->vertices = NULL;
    }
    return G;
}
```

3.1.2 Inserção de vértices

A inserção de vértices foi dividida em duas funções.

A primeira é chamada no programa do usuário, ao carregar os elementos do arquivo texto, a partir do vetor X[ ] que contém todos os vértices.

Essa função tem como entrada o grafo e o Registro a ser adicionado. Se o vértice não existir ele é adicionado e tem um retorno numérico 1 caso tenha tido sucesso na inserção e 0, caso contrário.

(MAIN.C)

```
int confirma = 0;
for(i=0; i<k; i++){
   confirma = inserirNovoVertice(g, x[i]);
}
if(confirma == 1)
   printf("\nVértices adicionados com sucesso\n");
```

A segunda função de inserção se encontra no menu de opções, funciona no mesmo método que foi explicado acima. O único diferencial é o fato das informações do Registro, novo vértice, são passados manualmente.

(MAIN.C)

```
printf("\n\nIndique as informações do vértice:\n");
printf("Nome: ");fflush(stdin);scanf("%[^\n]", nome);
printf("Tipo: ");fflush(stdin);scanf("%s", tipo);
printf("Ação: ");fflush(stdin);scanf("%s", acao);

strcpy(novo.nome, nome); strcpy(novo.tipo, tipo); strcpy(novo.acao, acao);


inserirNovoVertice(g, novo);
```

3.1.3 Inserção de arestas

No início do programa as arestas são adicionadas de acordo com o arquivo texto e a ordem em que os registros aparecem. Uma aresta é formada por dois vértices e um peso e dessa forma que elas são adicionadas no grafo.

(MAIN.C)

```
int m = 0;
for(i=0; i<2*j; i=i+2){
      confirma = inserirArco(g, Arestas[i], Arestas[i+1], peso[m]);
      m++;
}
if(confirma == 1)
      printf("Arestas adicionadas com sucesso\n");
```

Outra função de inserção de arestas também aparece no menu de opções do usuário. Para criar uma aresta é necessário que os vértices que ela liga já existam. Foi criado um sistema de id, em que todos os vértices possuem um número que os representa. Dessa forma, o usuário passa os ids que ele quer conectar e o peso de tal aresta para que esta seja criada.

(MAIN.C)

```
printf("\n\nIndique o id do primeiro vértice: ");
scanf("%d", &id);

printf("Indique o id do segundo vértice: ");
scanf("%d", &id2);

printf("\nQual é o peso da aresta que vai ser adicionada? ");
scanf("%d", &pesos);
printf("\n\n");

if(inserirArco2(g, id, id2, pesos)==0)
      printf("\nO arco já existe!");
```

Como se trata de uma função auxiliar, quando esta é chamada, é identificado os Registros a partir dos id's, é chamada a primeira função mencionada e o arco é criado.

(GRAFO.C)

```
vert1 = acha_vertice(G, v1);
vert2 = acha_vertice(G, v2);

inserirArco(G, vert1, vert2, peso);
```

3.1.4 Remoção de Vértice

Remoção de vértice ocorre no menu de opções, em que é passado o grafo e o id do vértice que quer ser retirado para a função e lá é feita as modificações necessárias.

No caso de remoção de vértices, além deste ser removido da lista de vértices, tem que ser retirado, também, da lista de adjacências.

Após a remoção, como esse grafo trabalha com ids para seus vértices, estes têm que ser atualizados.

A função devolve o vértice que foi retirado.

(MAIN.C)

```
printf("\n\nIndique o id do vértice a ser retirado: ");
fflush(stdin);
scanf("%d", &id);

aux = removerVertice2(g, id);

printf("O vértice retirado foi: %s", aux.nome);
```

3.1.5 Remoção de Arestas

A remoção das arestas é feita por meio dos id's. Uma vez dado os id's dos vértices, eles são conferidos, para ter certeza de que existem, e depois, é retirado o arco.

A função que é chamada recebe o grafo e os ids dos vértices que vão ser retirados como entrada e retorna um valor número que representa se teve sucesso (1) ou falha (0) na remoção.

(MAIN.C)

```
printf("Indique o informações do primeiro vértice: ");
scanf("%d", &id);

printf("\nIndique as informações do segundo vértice: ");
scanf("%d", &id2);

if(removerArco2(g, id, id2)==1)
     printf("\nAresta retirada com sucesso!\n");
else
     printf("\nErro ao retirar aresta! Vértices sem conexão!\n");
```

3.2 Busca do vértice de maior grau

Para calcular o maior grau de um vértice em um grafo direcionado, é necessário calcular tanto os graus de entrada quando os de saída. Isto é, percorrer todo o grafo em busca das aparições do vértice em questão na lista de adjacência dos outros vértices e, também, buscar o número de vértices que a lista de adjacência do vértice procurado tem.

A função que retorna o maior grau do vértice faz uma comparação com todos os graus e retorna o grau e vértice que interessa.

Apesar de devolver um vértice que tem o maior grau, pode ser que tenha mais de um vértice que contém essa quantidade.

(MAIN.C)

```
printf("\nO maior grau é: ");
aux = maior_grau(g);
printf("\nO vértice de maior grau é: ");
printf("%s", aux.nome);
```

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

Para a função de verificação de caminho entre dois vértices, é tido como valor de entrada o grafo e os dois ids dos vértices e como saída um valor numérico, que confirma se há um caminho de conexão (1) ou se não há, (0).

Essa verificação é feita pelo método de Busca em Profundidade, em que é passado por todos os vértices e vértices de adjacência em busca de um caminho entre os outros vértices do grafo. Nessa busca, se é encontrado o segundo vértice dado como referência há o retorno de 1, caso passe por todo o grafo e não encontre o que procura, o retorno é 0.

(GRAFO.C)

```
for (nv = G->vertices; nv!=NULL; nv = nv->prox){
     if(strcmp(vt.nome, nv->vert.nome)==0){
          for (na = nv->ladj; na != NULL; na = na->prox){
               if(strcmp(v2.nome, na->vert.nome)==0){
                    return 1;
               }
               if (FoiVisitado(visitados, na->vert)==0){
                    if(!Empilha(&pilha, na->vert)){
                         EsvaziaPilha(&pilha);
                         EsvaziaPilha(&visitados);
                         return 0;
                    }
               }
               continue;
          }
     }
}
```

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

Para resolver o exercício foi usado o algoritmo de Dijkstra.

Esse algoritmo é um dos algoritmos que calcula o caminho de custo mínimo entre vértices de um grafo. Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo deste vértice para todos os demais vértices do grafo. Ele parte de uma estimativa inicial para o custo mínimo e vai sucessivamente ajustando esta estimativa.

Apesar de ser bem complexo pensar nesse algoritmo por não envolver números e os vértices serem do tipo Registro, por terem id, o processo é facilitado.

No menu do usuário, é pedido o id de um vértice e a partir dele é calculado o caminho mínimo entre ele e os outros vértices.

(MAIN.C)

```
printf("\nDigite a id: ");
scanf("%d", &id);
caminhoMaisCurto(g,id);
```

3.5 Usando busca em profundidade, encontrar recursos fortemente conectados.

Para resolver o exercício foi usado o algoritmo de Kosaraju.

Esse algoritmo se baseia em fazer várias buscas no grafo até que se encontre os recursos fortemente conectados.

Primeiramente foi utilizada a função Busca em Profundidade Tempo, dado o primeiro vértice, que foi usado como convenção, é percorrido o grafo e é encontrado os tempos de finalização dos vértices.

Com esse tempo de finalização em um vetor, é chamada uma função de Ordenação, em que os tempos são ordenados em ordem crescente dentro de uma Pilha, o topo é o maior tempo.

Após essa organização, é feita no grafo transposto do que estava sendo utilizado uma Busca em Profundidade com os elementos da Pilha, anteriormente formada.

O resultado dessa Busca em Profundidade são os elementos fortemente conectados.

No menu disponibilizado ao usuário, a opção de descobrir os elementos fortemente conectados, tem como entrada o grafo principal e retorna todos os conjuntos formados a partir do grafo.

(MAIN.C)

```
printf("\n\nGrupo Fortemente conectado:");
Kosaraju(g);
```

3.6 Impressão do grafo.

A função de impressão passa por todos os vértices e vértices adjacentes efetuando a impressão.

(GRAFOS.C)

```
void imprimirListaAdj(Grafo G) {
if (G == NULL) return;

struct noVert *nv;
struct noAdj *na;

printf("\n\nLista de Adjacencias:");
for (nv = G->vertices; nv!=NULL; nv = nv->prox) {
     printf("\n(id = %d) Vertice %s:",nv->id, nv->vert.nome);
     for (na = nv->ladj; na != NULL; na = na->prox){
          printf("(%s - id:%d)",na->vert.nome, na->id);

     }
}
}
```

4. ESTRUTURAS QUE AUXLIARAM

4.1 Grafo transposto

Um grafo transposto é um grafo que tem suas arestas de maneira oposta.

Na criação da função, a quantidade de vértices é a mesma então houve uma cópia dos vértices do grafo original, após a formação dos vértices foi criado uma função para que a troca de arestas fosse feita.

(GRAFOS.C)

```
 for (nv = G->vertices; nv!=NULL; nv = nv->prox) {
    strcpy(aux.nome, nv->vert.nome);
    strcpy(aux.tipo, nv->vert.tipo);
    strcpy(aux.acao, nv->vert.acao);
    inserirNovoVertice(Gt, aux);
}

for (nv = G->vertices; nv!=NULL; nv = nv->prox) {
    if(nv->ladj == NULL)
        continue;
    strcpy(aux1.nome, nv->vert.nome);
    strcpy(aux1.tipo, nv->vert.tipo);
    strcpy(aux1.acao, nv->vert.acao);
    for (na = nv->ladj; na != NULL; na = na->prox){
        strcpy(aux2.nome, na->vert.nome);
        strcpy(aux2.tipo, na->vert.tipo);
        strcpy(aux2.acao, na->vert.acao);
        pesoAux = na->peso;
        inserirArco(Gt, aux2, aux1, pesoAux);
    }
}
```

4.2 Pilha

Devido sua utilização nas funções de Busca em Profundidade e Busca em Profundidade tempo, foi adicionado esse TAD como um dos recursos dos grafos.

Nesse TAD é encontrado funções de criação, inserção, remoção e leitura do topo em uma pilha.

(PILHA.H)

```
typedef struct No* Pilh;

Pilha IniciaPilha();
int VaziaPilha(Pilha pilha);
Registro Topo(Pilha pilha);
int Empilha(Pilha *pilha, Registro x);
Registro Desempilha(Pilha *pilha);
int EsvaziaPilha(Pilha *pilha);
```

5. CASOS DE USO

5.1 Criação do grafo – Leitura do arquivo texto

```
****GRAFOS*****


Vértices adicionados com sucesso
Arestas adicionadas com sucesso


Lista de Adjacencias:
(id = 8) Nome: aaa - Tipo: arquivo - Acao: visualizacao:  (Nome: bbb - id:7)

(id = 7) Nome: bbb - Tipo: arquivo - Acao: visualizacao:  (Nome: ccc - id:5)

(id = 6) Nome: ddd - Tipo: arq - Acao: vis:  (Nome: aaa - id:8)

(id = 5) Nome: ccc - Tipo: arq - Acao: vis:  (Nome: eee - id:3)(Nome: ddd - id:6)

(id = 4) Nome: fff - Tipo: arquivo - Acao: visualizacao:  (Nome: ggg - id:1)

(id = 3) Nome: eee - Tipo: arquivo - Acao: visualizacao:  (Nome: fff - id:4)

(id = 2) Nome: iii - Tipo: arq - Acao: vis:

(id = 1) Nome: ggg - Tipo: arquivo - Acao: visualizacao:  (Nome: eee - id:3)

(id = 0) Nome: hhh - Tipo: arq - Acao: vis:  (Nome: iii - id:2)(Nome: ggg - id:1)
```

5.1.2 Inserção de vértice

```
 ---------------------------------------------------
  [1]Inserir novo vértice
  [2]Remover vértice
  [3]Inserir nova aresta
  [4]Remover aresta
  [5]Verificar vértice de maior grau
  [6]Verificar se existe caminho entre dois vertices
  [7]Encontrar o menor caminho
  [8]Encontrar recursos fortemente conectados
  [9]Imprimir
  [10]Sair
 ---------------------------------------------------

Digite a opcao desejada: 1


Indique as informações do vértice:
Nome: jjj
Tipo: arq
Ação: vis
```

```
 ---------------------------------------------------

Digite a opcao desejada: 9


Lista de Adjacencias:
(id = 9) Nome: jjj - Tipo: arq - Acao: vis:

(id = 8) Nome: aaa - Tipo: arquivo - Acao: visualizacao:  (Nome: bbb - id:7)

(id = 7) Nome: bbb - Tipo: arquivo - Acao: visualizacao:  (Nome: ccc - id:5)

(id = 6) Nome: ddd - Tipo: arq - Acao: vis:  (Nome: aaa - id:8)

(id = 5) Nome: ccc - Tipo: arq - Acao: vis:  (Nome: eee - id:3)(Nome: ddd - id:6)

(id = 4) Nome: fff - Tipo: arquivo - Acao: visualizacao:  (Nome: ggg - id:1)

(id = 3) Nome: eee - Tipo: arquivo - Acao: visualizacao:  (Nome: fff - id:4)

(id = 2) Nome: iii - Tipo: arq - Acao: vis:

(id = 1) Nome: ggg - Tipo: arquivo - Acao: visualizacao:  (Nome: eee - id:3)

(id = 0) Nome: hhh - Tipo: arq - Acao: vis:  (Nome: iii - id:2)(Nome: ggg - id:1)


 ---------------------------------------------------
```

5.1.3 Remoção de vértice

```
 ----------------------------------------------------
  [1]Inserir novo vértice
  [2]Remover vértice
  [3]Inserir nova aresta
  [4]Remover aresta
  [5]Verificar vértice de maior grau
  [6]Verificar se existe caminho entre dois vertices
  [7]Encontrar o menor caminho
  [8]Encontrar recursos fortemente conectados
  [9]Imprimir
  [10]Sair
 ----------------------------------------------------

Digite a opcao desejada: 2


Indique o id do vértice a ser retirado: 3
O vértice retirado foi: eee

 ----------------------------------------------------
```

```
 ----------------------------------------------------

Digite a opcao desejada: 9


Lista de Adjacencias:
(id = 8) Nome: jjj - Tipo: arq - Acao: vis:

(id = 7) Nome: aaa - Tipo: arquivo - Acao: visualizacao:  (Nome: bbb - id:6)

(id = 6) Nome: bbb - Tipo: arquivo - Acao: visualizacao:  (Nome: ccc - id:4)

(id = 5) Nome: ddd - Tipo: arq - Acao: vis:  (Nome: aaa - id:7)

(id = 4) Nome: ccc - Tipo: arq - Acao: vis:  (Nome: ddd - id:5)

(id = 3) Nome: fff - Tipo: arquivo - Acao: visualizacao:  (Nome: ggg - id:1)

(id = 2) Nome: iii - Tipo: arq - Acao: vis:

(id = 1) Nome: ggg - Tipo: arquivo - Acao: visualizacao:

(id = 0) Nome: hhh - Tipo: arq - Acao: vis:  (Nome: iii - id:2)(Nome: ggg - id:1)


 ----------------------------------------------------
```

5.1.4 Inserção de aresta

```
 ---------------------------------------------------
  [1]Inserir novo vértice
  [2]Remover vértice
  [3]Inserir nova aresta
  [4]Remover aresta
  [5]Verificar vértice de maior grau
  [6]Verificar se existe caminho entre dois vertices
  [7]Encontrar o menor caminho
  [8]Encontrar recursos fortemente conectados
  [9]Imprimir
  [10]Sair
 ---------------------------------------------------

Digite a opcao desejada: 3


Indique o id do primeiro vértice: 2
Indique o id do segundo vértice: 4

Qual é o peso da aresta que vai ser adicionada? 7


ccc foi conectado a iii!

 ---------------------------------------------------
```

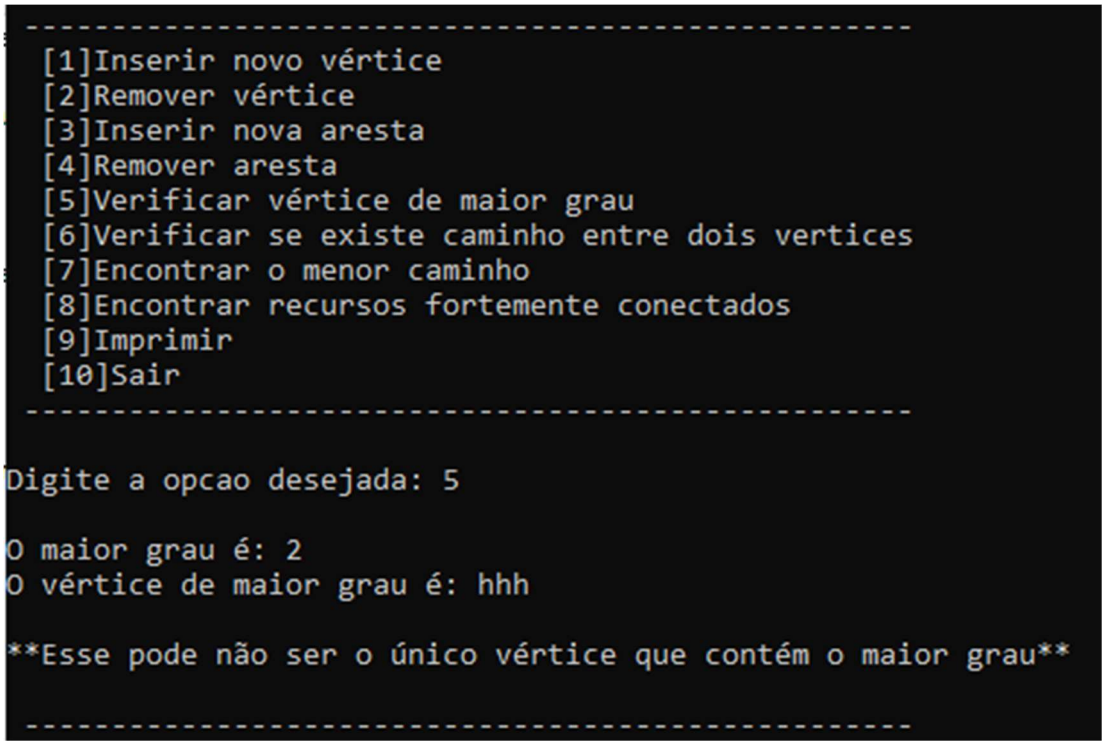
```
 ---------------------------------------------------

Digite a opcao desejada: 9


Lista de Adjacencias:
(id = 8) Nome: jjj - Tipo: arq - Acao: vis:

(id = 7) Nome: aaa - Tipo: arquivo - Acao: visualizacao:  (Nome: bbb - id:6)

(id = 6) Nome: bbb - Tipo: arquivo - Acao: visualizacao:  (Nome: ccc - id:4)

(id = 5) Nome: ddd - Tipo: arq - Acao: vis:  (Nome: aaa - id:7)

(id = 4) Nome: ccc - Tipo: arq - Acao: vis:  (Nome: ddd - id:5)

(id = 3) Nome: fff - Tipo: arquivo - Acao: visualizacao:  (Nome: ggg - id:1)

(id = 2) Nome: iii - Tipo: arq - Acao: vis:  (Nome: ccc - id:4)

(id = 1) Nome: ggg - Tipo: arquivo - Acao: visualizacao:

(id = 0) Nome: hhh - Tipo: arq - Acao: vis:  (Nome: iii - id:2)(Nome: ggg - id:1)


 ---------------------------------------------------
```

5.1.5 Remoção de aresta

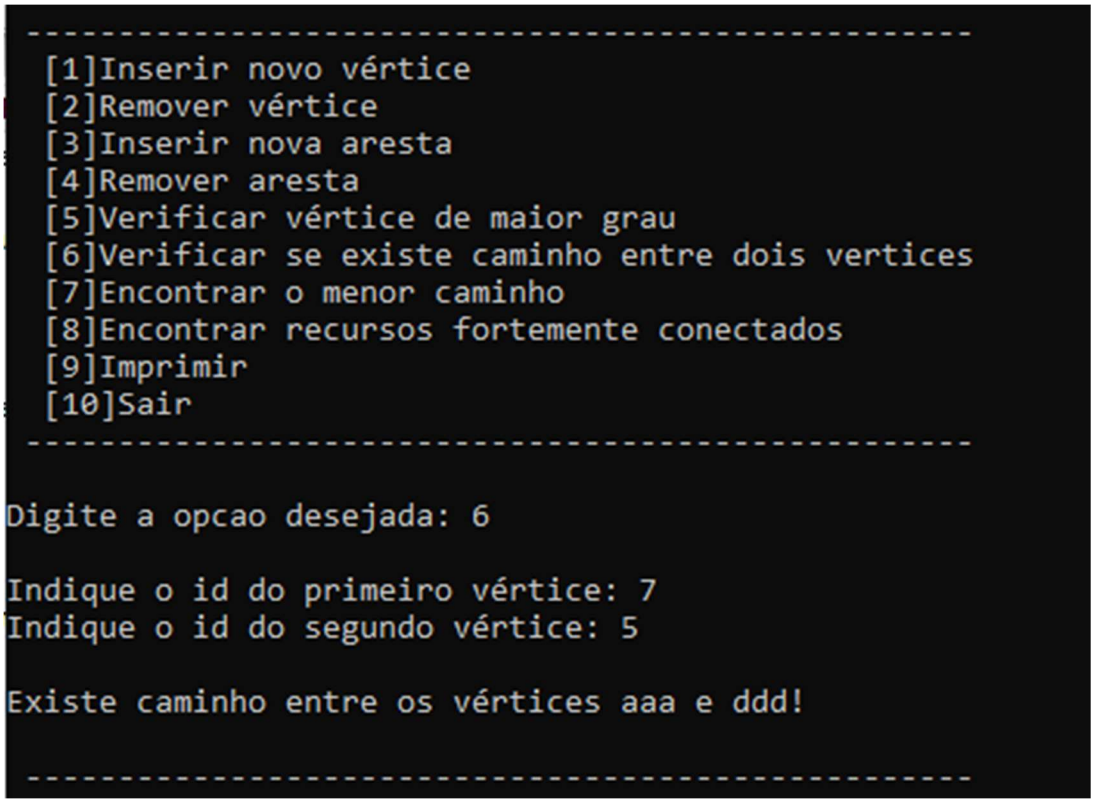
```
 ---------------------------------------------------
  [1]Inserir novo vértice
  [2]Remover vértice
  [3]Inserir nova aresta
  [4]Remover aresta
  [5]Verificar vértice de maior grau
  [6]Verificar se existe caminho entre dois vertices
  [7]Encontrar o menor caminho
  [8]Encontrar recursos fortemente conectados
  [9]Imprimir
  [10]Sair
 ---------------------------------------------------

Digite a opcao desejada: 4


     Para ser retirada a aresta...
Indique o informações do primeiro vértice: 2

Indique as informações do segundo vértice: 4

ccc foi desconectado a iii!

Aresta retirada com sucesso!


 ---------------------------------------------------
```

```
 ---------------------------------------------------

Digite a opcao desejada: 9


Lista de Adjacencias:
(id = 8) Nome: jjj - Tipo: arq - Acao: vis:

(id = 7) Nome: aaa - Tipo: arquivo - Acao: visualizacao:  (Nome: bbb - id:6)

(id = 6) Nome: bbb - Tipo: arquivo - Acao: visualizacao:  (Nome: ccc - id:4)

(id = 5) Nome: ddd - Tipo: arq - Acao: vis:  (Nome: aaa - id:7)

(id = 4) Nome: ccc - Tipo: arq - Acao: vis:  (Nome: ddd - id:5)

(id = 3) Nome: fff - Tipo: arquivo - Acao: visualizacao:  (Nome: ggg - id:1)

(id = 2) Nome: iii - Tipo: arq - Acao: vis:

(id = 1) Nome: ggg - Tipo: arquivo - Acao: visualizacao:

(id = 0) Nome: hhh - Tipo: arq - Acao: vis:  (Nome: iii - id:2)(Nome: ggg - id:1)


 ---------------------------------------------------
```

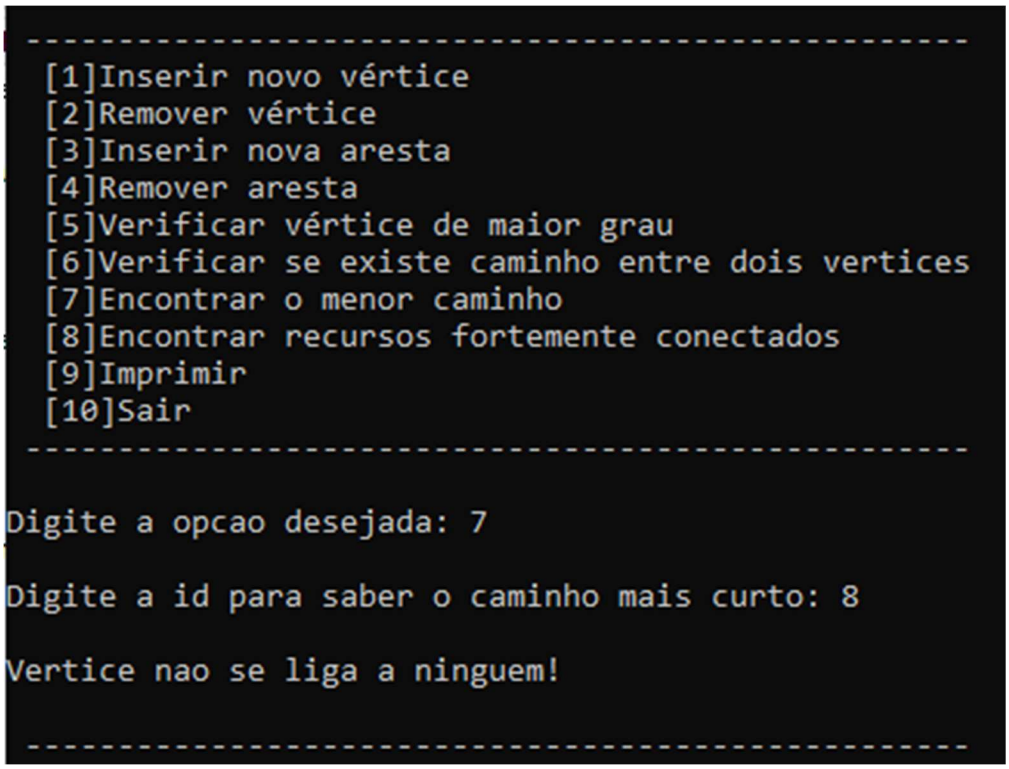
5.2 Busca de maior grau

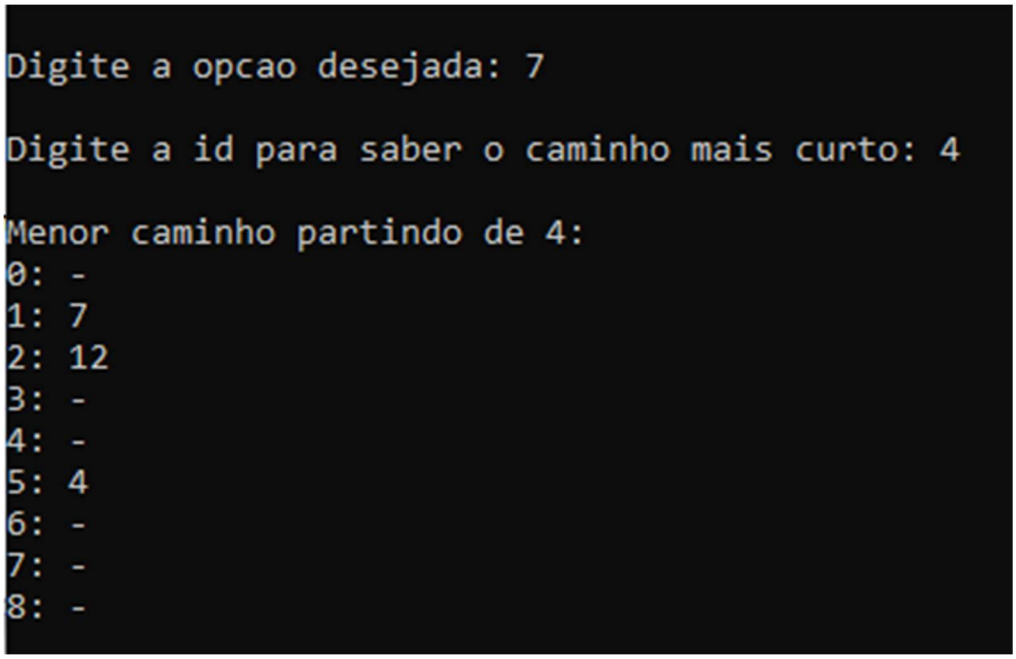


5.3 Verificar se existe caminho entre dois vértice

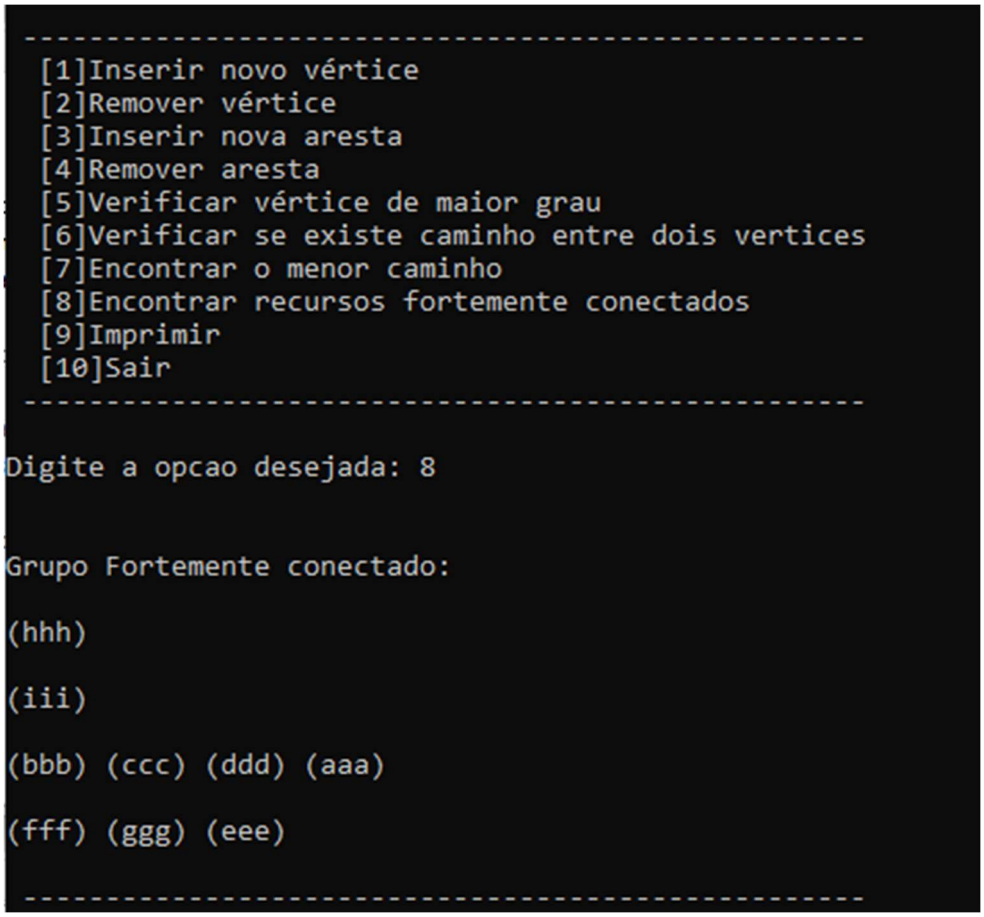


5.4 Encontrar o menor caminho para os outros vértices conectados a ele





5.5 Encontrar recursos fortemente conectados



5.6 Impressão – Final