Trabalho - Trilhas de Aprendizagem

Professora: Maria Adriana Vidigal

Alunos: Lorenzo Machado Burgos - 12011BCC005
Marcus Vinícius Torres Silva - 12011BCC025

O trabalho tem como objetivo praticar os conceitos vistos em aulas sobre Grafos. Em específico, manipular grafos do tipo direcionado e ponderado, com estrutura fixa. O objetivo geral do grafo é replicar as funções disponíveis em https://www.moodle.ufu.br/.

O trabalho é dividido em seis tópicos, nos quais deverão ser criadas funções com o objetivo de extrair informações do grafo. Esses tópicos são:

- **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.
- **B.** Busca do vértice de maior grau, que, para a trilha, representa um recurso com peso importante no fluxo.
- **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.
- **E.** Usando busca em profundidade, encontrar recursos fortemente conectados (Algoritmo).
- **F.** Impressão do grafo.

No tópico **A**, a dupla optou por escolher estruturar o grafo usando lista de adjacências, utilizando uma estrutura como uma lista encadeada. Para este tópico, tem-se as funções para criar um grafo, inserir e remover vértices e arestas(arco).
Para criar o grafo, temos uma struct grafo, que contém os campos que correspondem ao número de vértices, número de arestas e uma estrutura para o nó de um vértice.

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

Além disso, tem-se duas outras estruturas: 1. nó de um vértice, e 2. nó adjacência. As estruturas vistas até agora estão presentes em uma biblioteca chamada “**grafo.h**”.

Exemplo:
```
struct noVert {
   Vertice vert;
   struct noVert *prox;
```

```
   struct noAdj *ladj;
};
typedef struct noVert *Vert;

struct noAdj {
   Vertice vert;
   int peso;
   struct noAdj *prox;
};
typedef struct noAdj *Adj;
```

No tópico **B**, para encontrar o vértice de maior grau do grafo, optamos por uma função que calcula o grau de um vértice, e uma função que encontra o vértice de maior grau comparando os vértices.

Exemplo:

```
int grauVertice(Grafo G, Vertice v) {
   struct noVert *nv;
   struct noAdj *na;
   int cont = 0;
   if (G == NULL) return 0;
   for (nv = G->vertices; nv!=NULL; nv = nv->prox) {
    if(!strcmp(nv->vert->nome,v->nome) && !strcmp(nv->vert->tipo, v->tipo)
&& !strcmp(nv->vert->acao, v->acao)){
         for (na = nv->ladj; na != NULL; na = na->prox)
            cont = cont + 1;  // grau de saída
      }
      else{
         for (na = nv->ladj; na != NULL; na = na->prox)
             if(!strcmp(na->vert->nome,v->nome) && !strcmp(na->vert->tipo,
v->tipo) && !strcmp(na->vert->acao, v->acao))
                 cont = cont + 1; // grau de entrada
      }
   }
   return cont;
}

Vertice verticeMaiorGrau(Grafo G, Vertice v){

    Vert nv = G->vertices, maiorVert;
    int cont = 0, aux = 0;

    if(G == NULL) return 0;
    cont = grauVertice(G,nv->vert);

    aux = cont;
```

```
    for(maiorVert = nv; nv != NULL; nv = nv->prox){
        cont = grauVertice(G,nv->vert);
        if(cont > aux){
            maiorVert = nv;
            aux = cont;
        }
    }

    v = maiorVert->vert;
    return v;
}
```

vertice.h

O nó da lista representa um vértice. A struct vertice agrupa nela 3 variáveis de tipo char, o nome, tipo e ação.

1.Os tipos possíveis são: Arquivo, Página Pasta, Tarefa e URL

2. As ações disponíveis são: Baixar (Download), Enviar (Upload) e Vizualizar ela é criada por 3 tipos char pois fica mais fácil e interativo o programa aplicativo.

Pilha

IniciaPilha()

A função prepara a pilha para ser utilizad, apontando *prox para NULL

VaziaPilha

Checa se a pilha está vazia ou não. Se a base aponta pra NULL como é no caso de uma pilha recem inicializada, ela está vazia. Caso contrártio existe um ou mais nós.

Topo

Checa se a pilha está vazia ou não. Se não estiver pilha ponta para o topo que aponta para o vertice

Empilha

O primeiro passo é alocar espaço para esteve novo nó da pilha, o que é feito com ajuda da função aloca(). Como é uma pilha, seu último elemento (que é esteve novo), deve apontar para NULL, pois isso caracteriza o fim da pilha.

Adicionado o elemento, vamos procurar o último elemento da pilha. Temos o ponteiro *PILHA que aponta para a base. Se a pilha estiver vazia, fazemos o ponteiro *prox apontar para esteve novo nó

Desempilha

Primeiro fazemos uma checagem se a pilha está vazia, se estiver, não há nada a ser feito, pois não há nó para ser retirado da pilha.

Do contrário, vamos utilizar dois ponteiros para struct NO, o "ultimo" e o "penultimo". Basicamente, o que vamos fazer é que o "ultimo" aponte para o último elemento da pilha e o "penultimo" aponte para o último nó da pilha.