Documentação Técnica do algoritmo de Controle de Voos e Rotas aéreas

Emanuel Resende Melo

Instituto Federal de Minas Gerais - IFMG Campus Formiga
Rua Padre Albérico, 440 - São Luiz, Formiga - MG, 35570-000.

emanuelmelo0049538@gmail.com

1. Introdução.

A teoria dos grafos ou de grafos é um ramo da matemática que estuda as relações entre os objetos de um determinado conjunto. Para tal são empregadas estruturas chamadas de grafos, onde é um conjunto não vazio de objetos denominados vértices (ou nós) e (do inglês *edges* - arestas) é um subconjunto de pares não ordenados de V.

Dependendo da aplicação, arestas podem ou não ter direção, pode ser permitido ou não arestas ligarem um vértice a ele próprio e vértices e/ou arestas podem ter um peso (numérico) associado. Se as arestas têm um sentido associado (indicado por uma seta na representação gráfica) temos um dígrafo (grafo orientado). Um grafo com um único vértice e sem arestas é conhecido como grafo trivial.

Estruturas que podem ser representadas por grafos estão em toda parte e muitos problemas de interesse prático podem ser formulados como questões sobre certos grafos. Por exemplo, a estrutura de ligações da Wikipédia pode ser representada por um dígrafo: os vértices são os artigos da Wikipédia e existe uma aresta do artigo *A* para o artigo *B se e somente se A* contém um link para *B*. Dígrafos são também usados para representar maquinas de estado finitos. O desenvolvimento de algoritmos para manipular grafos é um tema importante da ciência da computação.

2. Desenvolvimento.

2.1. Linguagem Utilizada.

A linguagem que utilizamos para desenvolver esse algoritmo na disciplina é a linguagem C.

C é uma linguagem de programação de propósito geral, estruturada, imperativa, procedural, padronizada pela (ISO), criada em 1972 por Dennis Ritchie na empresa AT&T para desenvolvimento do Sistema operacional UNIX.

2.2. Estruturas.

ESTRUTURA DAS ROTAS

```
typedef struct TypeGraphMatriz
{
   int **Mat;
   int qntArestas;
   int qntVertices;
   int Digrafo;
} TypeGraphMatriz;
```

Essa estrutura é utilizada para criar um grafo de Rotas, cujo iremos armazenas nele as ligações entre as rotas.

QntArestas - Variável para armazenar a quantidade de arestas;

qntVertices: Variável que ira guardar quantos vértices tem nesse grafo.

Digrafo: Variavel que ira guardar se o grafo é um dígrafo ou não.

ESTRUTURA DO AEROPORTO

```
typedef struct Tipo_Voo
{
   int id;
   int duracao;
   char identificador_voo[10];
   int quantidade_voo;
   char duracao_horario[6];
   char de[4];
   char para[4];
   int paradas;
} Tipo_Voo;
```

Nessa estrutura iremos armazenas todos os dados dos voos, armazenando varias variáveis para termos total controle do voo.
Armazena o ID do voo.
Armazena a Duração do Voo.
Armazena o identificador do voo como por exemplo (ABQ-1).
Armazena a quantidade de voos para aquele aeroporto.
Armazena o Horário do Voo.
Armazena o Aeroporto de Destino.
Armazena o Aeroporto de Origem.
Armazena a quantidade de paradas do voo.

ESTRUTURA DO AEROPORTO

```
typedef struct Aeroporto
{
   char sigla[4];
   char cidade[50];
   char aeroporto[50];
   int x;
```

```
   int y;
   int id;

} Aeroporto;
```

Nessa estrutura armazenamos todos os dados para o controle do aeroporto.
Armazenamos a Sigla do aeroporto que condiz com a abreviação do seu nome.
Armazena a cidade do aeroporto.
Armazena o estado do aeroporto.
Armazena o X em relação ao eixo X do aeroporto.
Armazena o Y em relação ao eixo Y do aeroporto.
Armazena o ID do aeroporto.

ESTRUTURA DA ADJACENCIA

```
typedef struct adjacencia
{
   int id_voo;
   int predecessor;
   int distancia;
   struct adjacencia *prox;
} ADJACENCIA;

typedef struct vertice
{
   ADJACENCIA *ligar;
} VERTICE;

typedef struct grafo
{
   int qnt_Vertices;
   int qnt_Arestas;
   VERTICE *adj;
} GRAFO;
```

Nessa estrutura realizamos uma ligação por Lista de adjacência. Onde interligarmos os vértices e armazenamos os Ids dos voos que partem de cada vértice.
Armazenamos o predecessor que quer dizer o destino daquele voo.
Armazenamos o ID do voo.
Armazenamos a Distancia do Voo.
Armazenamos a quantidade de Vértices.
Armazenamos a quantidade de arestas.
E interligamos tudo pela lista de adjacência.

3. Funções.

int numero_deVERTICES(FILE *arq);
Função responsável por buscar no arquivo a quantidade de vértices

GRAFO *criarGrafo_adj(int v)
Função responsável por criar o grafo da lista de adjacência.

ADJACENCIA *criaAdj(int v, int distancia, int predecessor)
Cria a lista de adjacência

bool criaAresta(GRAFO *gr, int de, int para, int distancia, int predecessor)
Cria aresta na lista

Tipo_Voo *ler(GRAFO *voos, TypeGraphMatriz *rotas, FILE *arq, int vertices, Aeroporto *aeroporto, GRAFO *voo_rotas)
Lê o arquivo e armazena na estrutura.

int pesquisar_id_voo(GRAFO *voos, char de[10], Aeroporto *a)
Retorna o id do voo procurado pela Sigla dele.

void possibilidade_voo(char de[10], char para[10], Tipo_Voo *voo)
Printa as possibilidades de voo.

int retornar_posicao_voo(Tipo_Voo *dados, Aeroporto *aeroporto, char de[10], GRAFO *voos)
Retorna a posição do voo

void DFS(TypeGraphMatriz *rotas, Aeroporto *aeroporto, int origem, int destino, int paradas)
Realiza a busca do algoritmo DFS.

void DFS_Routh(TypeGraphMatriz *rotas, Aeroporto *aeroporto, int inicial, int origem, int destino, int paradas, int tamanho, int *guardar_rotas)
Realiza a rotas do dfs

void mostrar_voo_semconaxao(int id, Tipo_Voo *voo, GRAFO *gr)
Mostra todos os voos sem conexões.

int retornar_posicao_voo_id(Tipo_Voo *voo, char *de)
Passa por parâmetro a origem ele te retorna todos os voos com aquela origem.

int retornar_voo_viaDEPARA(Tipo_Voo *voo, char de[10], char para[10])
/Busca qual voo tem a origem e destino passados pelo usuário.

bool existeAberto(GRAFO *g, bool *aberto)
Função auxiliar da dijskra que pesquisa se ainda existe variável auxiliar aberta.

void *dijskra(GRAFO *g, int s, int destino,Aeroporto*aeroporto)
Função dijkra utilizada na letra D, para calcular o menor caminho.

void relaxa(GRAFO *g, int *d, int *p, int u, int v)
funcao auxiliar da dijskra usada para o menor caminho

int menorDist(GRAFO *g, bool *aberto, int *d);
Função auxiliar da dijkra para devolver a vértice de menor distancia.

void inicializaD(GRAFO *gr, int *d, int *p, int s);
Função auxiliar da dijkra para iniciar os vetores.

int calcula_distancia(int x1, int x2, int y1, int y2)
Calcula a Distancia dos eixos x1,x2 em relação a y1 e y2.

4. Conclusão

Vimos profundamente como é o funcionamento dos grafos na pratica. Porem houve bastante duvidas em métodos de aplicações e algoritmo para realizar diversas coisas. Mas todavia com o tempo e pesquisas a grande parte das duvidas foram sanadas, porem, algumas ainda permaneceram e irei atrás de sanar todas as duvidas que ainda estão perduradas. Grafos são de grande ajuda e facilitam diversas coisas dentre o meio da sociedade, podemos perceber isso utilizando nessa aplicaçao que grafo vai bem além do que podemos ver rasamente.