Documentação técnica do software de Controle de Voos e Rotas Aéreas.

Desenvolvedores: Jefferson Marques Costa Alves.
Paulo Henrique Xavier.

Formiga/MG
2018

1. Introdução.

A teoria dos 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, G(V,E), onde V é um conjunto não vazio de objetos denominados vértices (ou nós) e E 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 uma direção associada (indicada 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. Dígrafos são também usados para representar máquinas de estado finito. O desenvolvimento de algoritmos para manipular grafos é um tema importante da ciência da computação.

2. Desenvolvimento.

1. Linguagem utilizada.

C é uma linguagem de programação compilada de propósito geral, estruturada, imperativa, procedural, padronizada pelo ISSO, criada em 1972 por Dennis Ritchie, no AT&T Bell Labs, para desenvolver o sistema operacional Unix (que foi originalmente escrito em Assembly).

C é uma das linguagens de programação mais populares e existem poucas arquiteturas para as quais não existem compiladores para C.

2. Estruturas.

Estrutura de conexão, usada quando é o grafo de voos.
Armazena a duração de um determinado voo.
Armazenar o identificador de um determinado voo.
Armazenar a duração em um formato de horas.

```
typedef struct {
	int  duracao;
	char identificador_voo[10];
	char duracao_horario[6];
}Conexao;
```

Estrutura do aeroporto que será usada na matriz.
Armazenar o ID de uma determinada cidade/aeroporto.
Armazenar a abreviação da cidade.
Armazenar o nome do aeroporto.
Armazenar a quantidade de voos que ele tem para um determinado lugar.
Armazena um Vetor de conexões, que nada mais é que, a construção de uma terceira dimensão, fazendo com que a matriz vire um cubo de memória.

```
typedef struct {
	int  id;
	char abreviacao_cidade[4];
	char nome_cidade[70];
	char nome_aeroporto[50];
	int qtdeVooDePara;
	Conexao *vetor_Conexao;
}Aeroporto;
```

Estrutura do grafo.
Armazena o controle se o grafo/dígrafo é ou não ponderado (se tem pesos ou nao).
Armazena o controle se é um grafo ou um dígrafo.
Armazena a quantidade de vértices que este grafo possui.

Armazena uma matriz de inteiros, que nela há as ligações das rotas existentes.
Armazena uma matriz de aeroportos, que nel há os dados dos aeroportos.

```
typedef struct {
        int eh_ponderado;
        int eh_digrafo;
        int nro_vertices;
        int **matriz_Rotas;
        Aeroporto **arestas;
}Grafo;
```

Estrutura de controle para os grafos, eles compartilham os mesmos vértices.
Armazena o grafo de rotas, que nele há os dados das rotas.
Armazena o grafo de voos, que nele há os dados dos voos.
Armazena uma matriz bi dimencional, usada em um algoritmo, nela há o menor caminho de cada voo.
Armazena um vetor de rotas, que nele há as rotas de voos, para ter um controle mais fácil.

```
typedef struct {
        Grafo *rotas;
        Grafo *voos;
        int **matriz_bi;
        Aeroporto *vetor_rotas;
}Controle;
```

3. Principais funções.

Controle *cria_controle(): Função responsável por alocar a variável de controle e colocar seus atributos nulos.
Entrada de dados: Não recebe nada de parâmetro.
Saída de dados: Retorna a variável de controle alocada na memória.

Grafo *cria_Grafo(int nro_vertices, boolean eh_ponderado, boolean eh_digrafo): Função responsável por criar um grafo ou dígrafo, e alocar suas matrizes.
Entrada de dados: Ela recebe por parâmetro, a quantidade de vértices que vai ser usado para alocar as matrizes deste grafo/dígrafo, e também recebe duas variáveis boolean de controle, para ver se eh um grafo ou um dígrafo e se o mesmo vai ser ponderado ou não (se vai ter peso ou nao).
Saída de dados: Grafo* - Retorna um grafo que foi criado, com suas matrizes já alocadas, sendo a matriz de aeroportos, que aqui mesmo já é iniciada suas variáveis, e a também a matriz de inteiros que é preenchida por 0.

Controle *libera_Controle(Controle *controle): Função que é responsável por liberar da memória a variável de controle do programa.
Entrada de dados: Recebe a variável controle, que vai ser liberada da memória.
Saída de dados: Controle* - Retorna NULL.

Grafo *libera_Grafo(Grafo *gr): Função que é responsável por liberar o grafo da memória.
Entrada de dados: Recebe um grafo de parâmetro, para ser liberado da memória.
Saída de dados: Grafo* - Retorna NULL.

Aeroporto **libera_Matriz_Aeroporto(int nro_vertices, Aeroporto **matriz): Função que é responsável por liberar a matriz de aeroportos da memória.
Entrada de dados: Recebe por parâmetro a quantidade de vértices existentes e também a matriz de aeroportos.
Saída de dados: Aeroporto** - Retorna NULL.

int **libera_Matriz_Inteiro(int nro_vertices, int **matriz): Função que é responsável por liberar a matriz de inteiros da memória.
Entrada de dados: Recebe por parâmetro a quantidade de vértices existentes e também a matriz de inteiros.
Saída de dados: int** - Retornando NULL.

boolean insere_Rotas(Grafo *gr, int origem, int destino): Função que é responsável por inserir rotas no grafo.
Entrada de dados: Recebe de parâmetro o grafo, recebe também o índice de origem e recebe o índice de destino.
Saída de dados: boolean - Retorna false, se o grafo estiver nulo ou ele estiver com algum problema, ou retorna true caso tenha inserido as rotas com sucesso.

boolean leitura(Controle *controle, char url[]): Função responsável por ler todos os dados necessários do arquivo, e armazenar esses dados, nas estruturas certas.
Entrada de dados: Recebe de parâmetro a variável de controle, e o url do arquivo a ser lido.
Saída de dados: Retorna false se houve um erro na leitura do arquivo e retorna true caso a leitura tenha tenha sido um sucesso.

int **algoritmo_de_prim(int qtdeVertices, int **matriz_bi, int inicio): Função responsável por mostrar a arvore geradora minima de acordo com o vertice desejado.
Entrada de dados: Recebe de parâmetro a quantidade de vertices que tem no grafo, a matriz bidimensional com o menor custo e o id do vertice raiz desejado.
Saída de dados: Retorna uma matriz resposta com a arvore resultante.

void cria_Prim(Grafo *gr, int **matriz, int de): Função responsável por criar arquivos .dot do grafo de prim.
Entrada de dados: Recebe o grafo de parâmetro.
Saída de dados: void.

int conta_Aeroportos(FILE *arq): Função responsável por contar a quantidade de vertices que tem no arquivo txt.
Entrada de dados: Ponteiro para o arquivo.
Saída de dados: int - A quantidade de aeroportos existentes.

void cria_Dot_Voos(Grafo *gr): Função responsável por criar arquivos .dot do grafo de voos.
Entrada de dados: Recebe o grafo de parâmetro.
Saída de dados: void.

void cria_Dot_Rotas(Grafo *gr): Função responsável por criar arquivos .dot do grafo de rotas.
Entrada de dados: Recebe o grafo de parâmetro.
Saída de dados: void.

int **conversor_matriz_tri_bi(Grafo *gr): Transforma a matriz tridimensional em uma matriz bidimensional, sera pego a menor duração
Entrada de dados: Recebe um grafo.
Saída de dados: int** - Retorna a matriz bidimensional com o menor custo.

int *caminho_de_para(Grafo *gr, Aeroporto *origem, Aeroporto *destino, int **matriz_bi, int *duracao): Função responsável por achar o menor caminho de um aeroporto origem para um destino.
Entrada de dados: Recebe um grafo, um aeroporto origem, um aeroporto destino, a matriz com menor custo e uma variável de duração que sera modificada aqui dentro para ser usada fora.
Saída de dados: int* - Retorna o vetor com os ids.

void voo_sem_escala(Grafo *gr, Aeroporto *origem, int id): Função que é responsável por criar um .dot para mostrar no graphviz os voos diretos de um determinado aeroporto.
Entrada de dados: Recebe de parâmetro o grafo, o aeroporto de origem, e o ID do aeroporto de origem.
Saída de dados: void.

int *aeroportos_Fora_De_Servico(Grafo *gr, Aeroporto *origem, Aeroporto *destino, int *tam_vetor_trajeto): Função responsável por mostrar quais aeroportos que se ficarem fora de serviço conseguiriam impedir voce de chegar da origem para o destino.
Entrada de dados: Recebe um grafo, um aeroporto origem, um aeroporto destino e uma variável de tamanho que sera modificada aqui dentro para ser usada fora.
Saída de dados: int* - Retorna um vetor com os índices desse aeroportos.

3. Conclusão.

Durante o desenvolvimento deste trabalho houve bastante dúvida, principalmente em algumas definições do que os algoritmos faziam e corretude das respostas, mas depois de quebrarmos a cabeça e estudarmos mais um pouco as dúvidas foram sumindo.

Contudo, esse trabalho foi de grande ajuda para aumentar ainda mais o conhecimento, tanto na prática na hora de desenvolver quanto em saber a fundo como os grafos funcionam.

4. Referências.

BEZERRA, Karlisson. **Algoritmo de Prim**. Disponível em: <https://github.com/hacktoon/1001/blob/master/other-languages/graphs/prim/prim.java>. Acesso em: 12 nov. 2018.