PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Curso de Bacharelado em Sistemas de Informação

Alaene Rufino de Sousa, Diego Henrique Rodrigues de Siqueira, Jean Pierre Soares Oliveira Filho, Lucas Teixeira Matos Diniz, Matheus Gontijo Dias

DESENVOLVER UMA APLICAÇÃO QUE RECEBA COMO ENTRADA UM CONJUNTO DE DADOS ACERCA DE UMA MALHA AÉREA E RESPONDA ALGUNS TIPOS DE PERGUNTAS

Belo Horizonte

2018

Alaene Rufino de Sousa, Diego Henrique Rodrigues de Siqueira, Jean Pierre Soares Oliveira Filho, Lucas Teixeira Matos Diniz, Matheus Gontijo Dias

DESENVOLVER UMA APLICAÇÃO QUE RECEBA COMO ENTRADA UM CONJUNTO DE DADOS ACERCA DE UMA MALHA AÉREA E RESPONDA ALGUNS TIPOS DE PERGUNTAS

Projeto Interdisciplinar das matérias de Algoritmos em Grafos, Engenharia de Requisitos e Banco de Dados, ministrada por Eveline Alonso Veloso, Juliana Amaral e Rodrigo Baroni respectivamente.

Orientador: Prof. Eveline Alonso Veloso

Belo Horizonte

2018

LISTA DE ILUSTRAÇÕES

Figura 1 – Resultado Esperado do Programa Rodando 4
Figura 2 – Buscar em Profundidade 5
Figura 3 – Busca em Amplitude 6
Figura 4 – Kruskal 7
Figura 5 – Tarjan 7
Figura 6 – Dijkstra 8

SUMÁRIO

1 INTRODUÇÃO 4

2 MODELAGEM E SOLUÇÃO 5
2.1 Modelagem 5
2.2 Solução 8

3 CONCLUSÃO 10

1 INTRODUÇÃO

Neste trabalho devemos resolver o problema de uma malha área conforme o enunciado a seguir: A malha aérea de uma região pode ser representada por um grafo, no qual os vértices são os aeroportos dessa região. Um grafo não-dirigido pode ser utilizado para modelar as diversas rotas; enquanto um grafo dirigido pode ser usado para representar os diversos voos. Uma rota possui vários voos. Por exemplo, há voos de Confins para Salvador na parte da manhã, da tarde e da noite.

O trabalho deve ser desenvolvido respeitando as seguintes tarefas:

- Criar uma estrutura de dados que seja capaz de suportar os dois grafos: o de rotas e o de voos, os quais partilham os vértices, que são os aeroportos da região modelada. O grafo que representa as rotas deve ter apenas uma aresta conectando cada par (origem, destino) de aeroportos atendidos; enquanto o grafo de voos deve ter apenas uma aresta conectando cada par ordenado (origem, destino) de aeroportos. Existem vários pesos associáveis às arestas: distância, duração do voo, horários dos voos (apenas para o grafo que representa os diversos voos).
- Dados uma origem e um destino, desenvolver um algoritmo que determine a viagem com menor custo em termos de: número de conexões, distância total percorrida, tempo total de voo, duração total da viagem (considerando-se que pode haver esperas nas conexões. Nesse caso, utilize o primeiro horário de voo possível).
- Desenvolver um algoritmo que determine se é possível, para todos os aeroportos da região, a partir de um aeroporto atingir qualquer outro (com ou sem escalas). Se isso não for possível, indique os conjuntos de aeroportos que, separadamente, atendem essa condição. Se isso for possível, indique quais os aeroportos que, se ficassem fora de serviço (apenas um de cada vez), impediriam essa situação para o conjunto de aeroportos em operação restante.
- Suponhamos que seja preciso chegar ao aeroporto B para uma reunião importante à hora H. Desenvolver um algoritmo que determine o último voo em que se pode sair do aeroporto A sem chegar atrasado ao destino.
- Suponhamos que você pretenda montar uma empresa de carga aérea, com uma frota na qual cada aeronave fará voos de ida e volta apenas entre dois aeroportos. Quantas aeronaves, no mínimo, serão necessárias e quais rotas serão efetivamente usadas, se o objetivo é atender todos os aeroportos, com um consumo total mínimo (não é importante o tempo total que uma encomenda demorará para chegar ao destino, mas apenas garantir que há uma rota até esse destino).

E responder as seguintes questões:

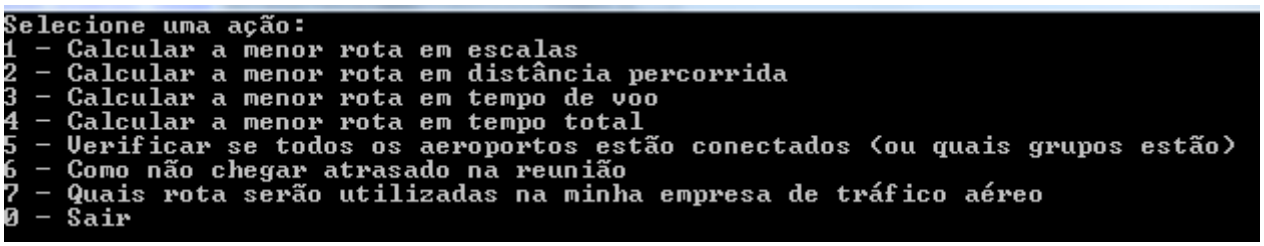


Figura 1 – Resultado Esperado do Programa Rodando

2 MODELAGEM E SOLUÇÃO

2.1 Modelagem

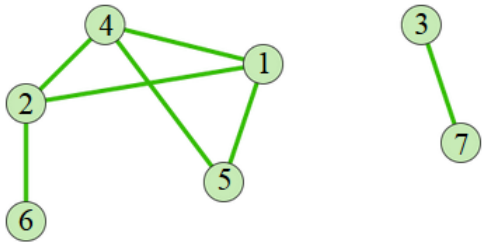
Para atender os requisitos foram necessários a utilização dos seguintes algoritmos.

- Busca em Profundidade: Para verificar se o grafo tem ciclos.

Busca em Profundidade

```
DFS(G)
1   Para todo v em G
2       Se v não visitado então
3           DFS-Visit(G, v)

DFS-Visit(G, v)
1   Marque v como visitado
2   Para todo w em Adj(v)
3       Se w não visitado então
4           Insira aresta (v, w) na árvore
5           DFS-Visit(G, w)
```



Busca em Profundidade

```
DFS(G)
1  for every vertex u of G
2      cor[u]←white; π[u]←NIL
3  time←0
4  for every vertex v of G
5      if cor[v]=BRANCO then
6          DFS-Visit(u)

DFS-Visit(u)
1  cor[u]←CINZA
2  time ← time+1
3  d[u]← time
4  for  each v in Adj[u]
5      if cor[v]=BRANCO then
6          π[v]←u;
7          DFS-Visit[u]
8  cor[u]←NEGRO
9  time ←time+1
10 f[u] ←time+1
```

Figura 2 – Buscar em Profundidade

- Busca em Amplitude: Menor caminho em escalas pois as arestas não eram ponderadas.

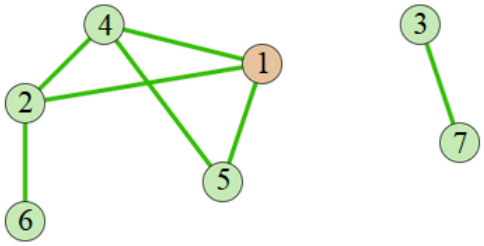
Busca em Largura

BFS(G)

```
1  for every vertex s of G  not explored yet
2       do Enqueue(S,s)
3       mark vertex s as visited
4       while S is not empty do
5            u ← Dequeue(S);
6            For each  v in Adj[u] then
7                 if v is unexplored then
8                      mark edge (v,u) as tree edge
9                      mark vertex v as visited
10                     Enqueue(S,v)
```

Notação

- Adj[u] : lista dos vértices adjacentes a u em alguma ordem
- Dequeue(S): Remove o primeiro elemento da fila S
- Enqueue (S,v) : Adiciona o nó v na fila S



BFS(G)

```
1  for every vertex s of G  not explored yet
2       do Enqueue(S,s);
3       mark vertex s as visited
4       while S is not empty do
5            u ← Dequeue(S);
6            For each  v in Adj[u] then
7                 if v is unexplored then
8                      mark edge (v,u) as tree edge
9                      mark vertex v as visited
10                     Enqueue(S,v)
```

Figura 3 – Busca em Amplitude

- Kruskal: Para encontrar a Arvore Geradora Miníma que são as rotas que serão utilizadas na tal da empresa de tráfego aéreo

KRUSKAL

$AGM\_Kruskal(G(V,A),\ w)$
$X \leftarrow \{\ \}$
para cada vértice $v \in V$ *faça*
criarConjunto(v)
fim para
$A' \leftarrow$ *ordenar as arestas de A por peso crescente*
para cada aresta $(u,v) \in A'$ *faça*
se $conjuntoDe(u) \neq conjuntoDe(v)$ *então*
$X \leftarrow X \cup \{(u,v)\}$
aplicarUnião(u,v)
fim se
fim para
retorne X
fim.

Figura 4 – Kruskal

- Tarjan: Onde é preciso encontrar todos os componentes fortemente conexos do grafo.

```
void GRAPHscTarjan( Graph G) {
   for (v = 0; v < G->V; ++v)                          /*1*/
      pre[v] = post[v] = pa[] = -1;                    /*2*/
   cnt0 = cnt1 = 0;                                    /*3*/
   N = 0;                                              /*4*/
   for (v = 0; v < G->V; ++v)                          /*5*/
      if (pre[v] == -1) {                              /*6*/
         pa[v] = v;                                    /*7*/
         strongR( G, v);                               /*8*/
      }                                                /*9*/
}

void strongR( Graph G, vertex v) {
   stack[N++] = v;                                     /*10*/
   pre[v] = cnt0++;                                    /*11*/
   for (a = G->adj[v]; a != NULL; a = a->next) {       /*12*/
      if (pre[a->w] == -1) {                           /*13*/
         pa[a->w] = v;                                 /*14*/
         strongR( G, a->w);                            /*15*/
      }                                                /*16*/
   }                                                   /*17*/
   post[v] = cnt1++;                                   /*18*/
   if (isRoot( v)) {                                   /*19*/
      do {                                             /*20*/
         u = stack[--N];                               /*21*/
         printf( "%d ", u);                            /*22*/
      } while (u != v);                                /*23*/
      printf( "\n");                                   /*24*/
   }                                                   /*25*/
}
```

Figura 5 – Tarjan

- Dijkstra: Menor caminho em tempo e distancia, pois levou em conta os pesos das arestas. Distancia

e Tempo de voo e realizado adaptações para encontrar outros dados solicitados.

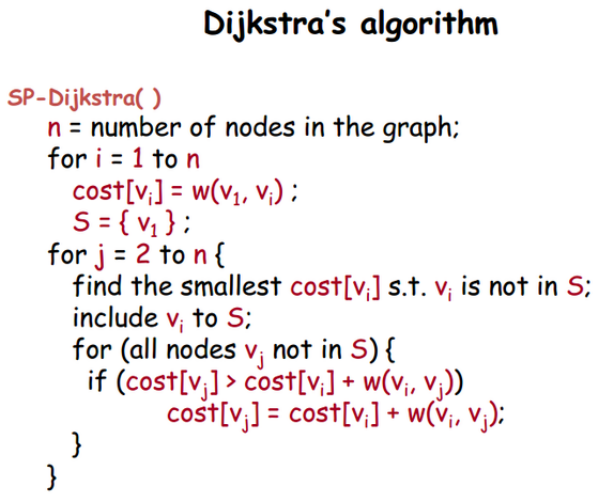


Figura 6 – Dijkstra

E com isso conseguimos atender os requisitos deste trabalho.

2.2 Solução

- Criar uma estrutura de dados que seja capaz de suportar os dois grafos: o de rotas e o de voos, os quais partilham os vértices, que são os aeroportos da região modelada. O grafo que representa as rotas deve ter apenas uma aresta conectando cada par (origem, destino) de aeroportos atendidos; enquanto o grafo de voos deve ter apenas uma aresta conectando cada par ordenado (origem, destino) de aeroportos. Existem vários pesos associáveis às arestas: distância, duração do voo, horários dos voos (apenas para o grafo que representa os diversos voos).
  - As estruturas de dados utilizadas para representar um grafo de rotas foi um Grafo Não Dirigido composto por uma lista de vértices, que são os aeroportos, e uma lista de arestas que são as rotas de um aeroporto de origem até um de destino. E para a representação de grafos de vôos foi utilizado um grafo dirigido composto por uma lista de vértices, que são os aeroportos, e uma lista de arestas os vôos.
- Dados uma origem e um destino, desenvolver um algoritmo que determine a viagem com menor custo em termos de: número de conexões, distância total percorrida, tempo total de voo, duração total da viagem (considerando-se que pode haver esperas nas conexões. Nesse caso, utilize o primeiro horário de voo possível).
  - Menor custo em termos de: número de conexões -> Foi utilizado o algoritmo de Busca em Amplitude.
  - Menor curso em termos de: distância total percorrida, tempo total de voo e duração total da viagem -> Foi utilizado o algoritmo de Dijkstra.
- Desenvolver um algoritmo que determine se é possível, para todos os aeroportos da região, a partir de um aeroporto atingir qualquer outro (com ou sem escalas). Se isso não for possível, indique os conjuntos de aeroportos que, separadamente, atendem essa condição. Se isso for possível, indique

quais os aeroportos que, se ficassem fora de serviço (apenas um de cada vez), impediriam essa situação para o conjunto de aeroportos em operação restante.

  - Para essa especificação foi utilizado o algoritmo de Tarjan.

- Suponhamos que seja preciso chegar ao aeroporto B para uma reunião importante à hora H. Desenvolver um algoritmo que determine o último voo em que se pode sair do aeroporto A sem chegar atrasado ao destino.

  - Foi utilizado o algoritmo de Dijkstra.

- Suponhamos que você pretenda montar uma empresa de carga aérea, com uma frota na qual cada aeronave fará voos de ida e volta apenas entre dois aeroportos. Quantas aeronaves, no mínimo, serão necessárias e quais rotas serão efetivamente usadas, se o objetivo é atender todos os aeroportos, com um consumo total mínimo (não é importante o tempo total que uma encomenda demorará para chegar ao destino, mas apenas garantir que há uma rota até esse destino).

  - Foi utilizado o algoritmo de Kruskal para encontrar a árvore geradora mínima(AGM).

3 CONCLUSÃO

Com este trabalho podemos perceber que a teoria de grafos pode ser aplicada em contextos diversos, mas que possuam um objetivo aproximado que no trabalho apresentado foi encontrar as menores rotas para não se perder uma reunião, por exemplo.

A teoria dos grafos é um ramo da matemática que estuda as relações entre os objetos de um determinado conjunto, no nosso caso é uma malha aeroviária. E que demanda certo conhecimento para colocar em prática as estruturas e algoritmos existentes para atender um caso específico.

Com tudo, podemos apreciar e melhorar nossas skills em conhecimento de desenvolvimento e análise de modelagem para algum problema que esteja declarado. Tendo em vista que foi-se necessário a utilização de quase todos(diria em grande parte) os algoritmos apresentados em sala pela professora Eveline Alonso Veloso.