Universidade Federal de Lavras

Trabalho de Grafos

Franciscone Luiz de Almeida Júnior
Igor Carvalho Cruz

Introdução

A Teoria dos Grafos é um ramo da matemática que estuda as relações entre objetos de um determinado conjunto. Esse conceito teve como primeiro trabalho o artigo de LeonhardEuler sobre as sete pontes de Königsberg, o que ilustra a aplicação de um problema real da teoria de grafos. Atualmente a teoria dos grafos tem aplicabilidade variada, podendo ir desde a obter ou gerar algum dado em um caminho entre cidades (Google Maps, exemplo), até obter dados para estrategias de marketing de pessoas que possuem alguma conexão nas redes sociais.

No contexto desse trabalho a Teoria dos Grafos é utilizada para resolver problemas que envolvem caminhos entre cidades e de entre localizações de uma mesma cidade. Esse problemas possuem certas peculiaridades que devem ser consideradas, os quais serão descritos nos tópicos a seguir.

Os problemas

1. O problema se resume em: Existem cidades conectadas por rodovias de mão dupla e em cada uma dessas cidades existe um serviço de aluguel de veículos que possui veículos que comportam uma certa quantidade de passageiros. Dadas a conexões entre as cidades, a capacidade máxima de transporte entre elas, as cidades de origem e destino e o número total de passageiros deve ser encontrado o melhor caminho que executa o menor número de viagens para levar todos os passageiros ao destino.

Algoritmos Implementados

Questão 01: Para resolver a questão 1 foi usado o Algoritmo de Dijkstra, com algumas modificações. Basicamente a questão pede o caminho que contenha a aresta de maior peso, contanto que esse peso não seja reduzido por uma outra aresta de peso menor no caminho. Na figura 1 por exemplo, se fosse necessário nesse contexto calcular o melhor caminho partindo do vértice 1 em destino ao vértice 2 a resposta seria o caminho 1 – 3 – 2, pois mesmo que exista uma ligação entre os vértices 1 e 4 de peso 22, esse peso seria reduzido e limitado pelo peso 1 da aresta (4, 2), daí a melhor escolha seria a primeira opção (As outras ligações saindo do vértice 1 foram ignoradas nessa explicação, pois os pesos são menores que as exemplificadas).

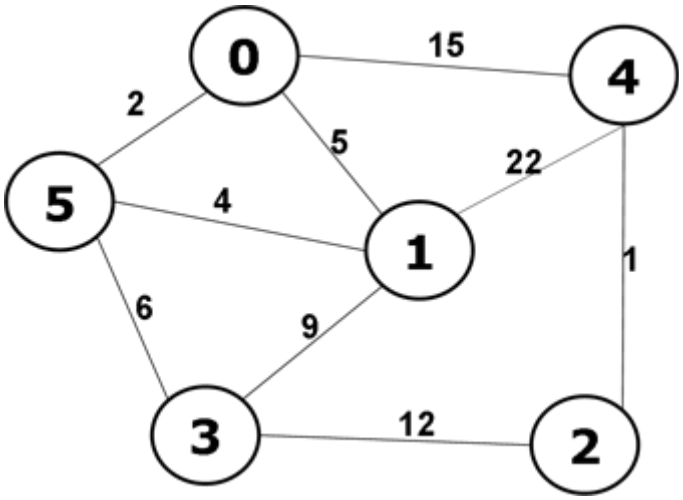


Figura 1: Grafo de exemplo.

Dado um grafo G=(V, E), onde u e v são vértices pertencentes a V, algoritmo de Dijkstra tem por ideia inicial encontrar o caminho mínimo dado de u ate v, utilizando busca em largura, ou seja, a cada vértice explorado ele verifica todos adjacente ate que se percorra todo grafo e encontre o

menor caminho para v atualizando seu valor mínimo encontrado a cada iteração ate que todo grafo seja verificado. Seu pseudocódigo pode ser verificado na figura 2:

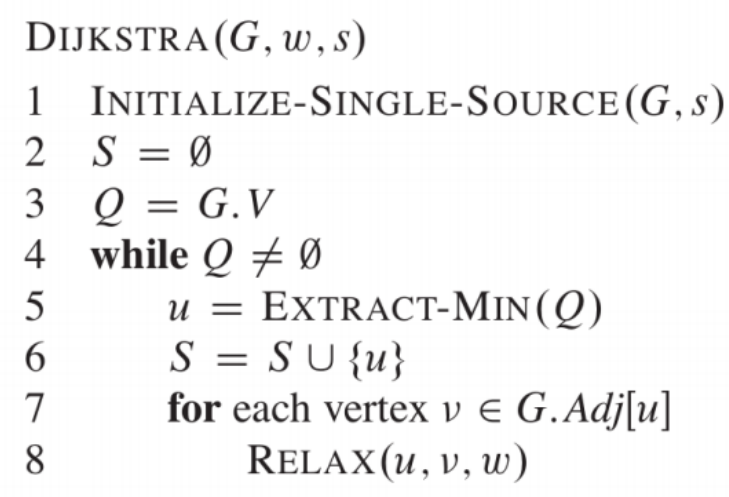


Figura 2: Pseudocódigo algoritmo de Dijkstra.

Para resolução do problema dado, onde devia-se encontrar um caminho onde se passasse com a maior quantidade de passageiros e com menor viajem ate um destino dado, foi realizada uma modificação na função RELAX. A modificação consiste em atribuir, caso o vértice atual seja o de partida o valor do maior custo para ele (instrução feita no primeiro if), e ao decorrer da exploração dos demais vértices são verificados se os valores dos pesos da aresta de conexão são menores, caso isso ocorra esse valor é definido como o limitante de passageiros daquele caminho (instrução feita no eles). Nesse segundo caso além do fator do peso da aresta se a condição no "*if*" não for satisfeita é preciso considerar o custo do vértice ancestral na próxima comparação, pois se o custo do vértice u for menor que o de uma nova aresta encontrada e o peso dessa mesma aresta for menor que o do vértice ancestral esse sera o novo limitante do caminho (representado pelo primeiro *else if*). Caso nenhuma dessas condições sejam satisfeitas o custo do caminho é mantido como o custo do vértice ancestral. Código em C++ pode ser verificado na figura 3.

```
void Grafo::relaxaVertice(Vertice* u, Vertice* v, Vertice *ancU, Aresta* w){
    if(u->rotulo == partida){
        if(u->custo < w->peso)
            u->custo = w->peso;
        v->custo = w->peso;
        v->anc = u->rotulo;
    }
    else if(u->rotulo != destino){
        if(w->peso < u->custo){
            u->custo = w->peso;
            v->custo = w->peso;
            v->anc = u->rotulo;
        }
        else if((w->peso > u->custo) and (w->peso < ancU->custo)){
            u->custo = w->peso;
            v->custo = w->peso;
            v->anc = u->rotulo;
        }
        else{
            v->custo = u->custo;
            v->anc = u->rotulo;
        }
    }

}
```

Figura 3: Código modificado da função Relax.

O modulo principal do algoritmo de Dijkstra não sofreu muitas alterações. A única mudança significativa foi o fato de que sempre que o “custo” do vértice mudasse ele deveria ser inserido novamente na fila de prioridade para que fosse explorado. Para ordenação dos vértices foi usada uma fila de prioridade que possui um modulo chamado mudarCusto(), iria remover e inserir novamente o vértice com o custo alterado para que a fila se mantivesse ordenada, esse processo em cada iteração teria no pior dos casos uma complexidade de O(n), isso no pior dos casos seria feito n vezes o que resultaria em uma complexidade total de $O(n^2)$.

TABELA DE COMPARAÇÕES

Caso nº:	**Vert. Partida**	**Vert. Destino**	**Caminho**	**Tempo**	**Num. Viagens**
01	1	7	1 – 2 – 4 – 7	0.069 ms	5
02	3	7	3 – 4 – 7	0.045 ms	3

Tabela 1: Comparações da questão 1.

Questão 02: A base desta questão é resolver um problema onde se deseja percorrer todas as arestas, com pesos dados em distancia entre ponto, a partir de um ponto inicial dado e retornar a este mesmo ponto com um tempo calculado no circuito.

A ideia de resolução é aplicar o algorítimo de Fleury que consiste em basicamente percorrer todas arestas do grafo com a restrição que só se deve passar por uma aresta ponte em ultimo caso, ou seja, se este é o único caminho disponível para percorrer. A figura 4 mostra o pseudocódigo do algoritmo de Fleury.

Algorithm 2 Algoritmo de Fleury

```
1:  procedure Fleury(G)                                   ▷ G = (V, E).
2:      Escolha um vértice v qualquer de V;
3:      C ← {v};
4:      repeat
5:          Escolha a aresta (v, w) não marcada utilizando a regra da ponte;
6:          Atravesse (v, w);
7:          C ← C ∪ {w};
8:          Marque (v, w);
9:          v ← w;
10:     until todas as arestas sejam marcadas
11:     C ← C ∪ {w};
12:     return C;                          ▷ Contém a ordem as arestas visitadas.
```

Figura 4: Pseudocódigo do algoritmo de Fleury.

O problema começa quando o ponto de partida não faz parte do conjunto de vértices do grafo, logo deve criar arestas adjacentes a este ponto para os demais pontos do grafo, e "euleriza-lo", garantir que todos vértices possuam grau par de entrada e saída.

TABELA DE COMPARAÇÕES

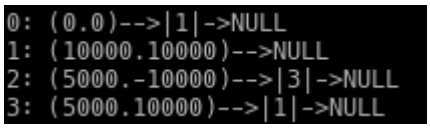
Caso nº:	**Partida Pertence ao grafo?**	**Num. Vértices**	**Num. Arestas**	**Necessário Eulerizar?**	**Tempo**
01	Sim	4	8	Não	3:55

Descrição da resolução do exercício:

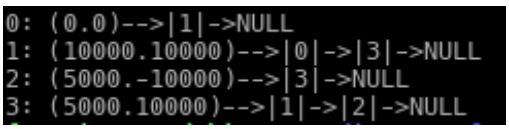
Durante a montagem do código encontramos várias dificuldades entre elas principalmente em relação a leitura das linhas em branco do arquivo, a modificação feita foi inserir um char 'b' e esse char sera considerado o final de um caso.

A montagem do grafo foi feita utilizando lista de adjacência, primeiro consideramos cada conjuntos de pontos um vértice(x, y) onde cada vértice recebia um "*label*" em ordem sua ordem de criação. As adjacência foram feitas utilizando um vector que guarda o "*label*" do vértice adjacente (Figura 1).

Como é um grafo direcionado foram adicionados a lista de adjacência as demais adjacências em seus respectivos vértices (Figura 2).



1. Figura: Lista de Adjacência após leitura de dados



2. Figura: Lista de Adjacência após atualizar adj.

O código não “euleriza” grafos.

O calculo é feito a partir do momento em que se passa por cada aresta é somado a variável atributo da classe grafo, distanciaTotal, e a partir dela, ao final da execução da função ImprimirCaminho é feita uma função que faz o calculo desta distancia, em metros, para o tempo em horas e minutos.