PMR3201 - Computação para Automação
Exercício Programa 2 - 2018

Algoritmo de caminho mínimo - Dijkstra's Algorithm

Prof. Dr. Thiago de Castro Martins
Prof. Dr. Marcos de Sales Guerra Tsuzuki
Prof. Dr. Newton Maruyama

Deadline: 17/05/2018 - 23h59min

1 Introdução

A primeira evidência de que se tem notícia quanto à aplicação de grafos remonta ao ano 1736 em que Euler fez uso deles para solucionar o agora clássico problema da ponte de Kögnisberg. Na cidade de Kögnisberg (Prússia Oriental), o rio Pregal flui em torno da ilha de Keiphof, dividindo-se em seguida em duas partes.

Assim sendo, existem quatro áreas de terra que ladeiam o rio (figura 1). Essas áreas de terra estão interligadas por sete pontes, $(a \ldots g)$. As áreas de terra estão assinaladas pelas letras $(A \ldots D)$. O problema da ponte de Kögnisberg consiste em se determinar, se ao partir de alguma área de terra, é possível atravessar todas as pontes exatamente uma vez para em seguida, retornar à area de terra inicial. Um caminho possível consistiria em iniciar na área de terra B, atravessar a ponte a para a ilha A; pegar a ponte e para chegar à area de terra D, atravessar a ponte g, chegando a C; cruzar a ponte d até A; cruzar a ponte b até B e a ponte f chegando a D.

Esse caminho não atravessa precisamente todas as pontes uma única vez, nem tão pouco retorna à área B. Euler provou que era impossível atravessar cada ponte uma única vez e retornar ao ponto inicial.

Esse problema pode ser resolvido representando as áreas de terra como vértices e as pontes como arcos de um grafo (na verdade um multigrafo) conforme ilustrado na Figura 1. Definindo o grau de um vértice como sendo o número de arcos que lhe são incidentes, Euler mostrou que existe um caminho com ponto de início em qualquer vértice, que passa através de cada arco exatamente uma vez e termina no vértice inicial contanto que seja de valor par o grau do vértice. O caminho que cumprir essas condições é dito Euleriano e o que não cumprir é dito não Euleriano.

Não existe nenhum caminho Euleriano no caso das pontes de Kögnisberg, uma vez que todos os vértices tem grau ímpar.

Muitos outros problemas podem ser naturalmente formulados através de grafos. Por exemplo, pode-se tentar se estabelecer qual é a menor distancia entre duas cidades sendo que existem vários caminhos alternativos passando por cidades distintas. No projeto de trilhas de conexão de componentes em placas de circuito impresso é possível se determinar trilhas de tal forma a serem de menor comprimento possível e sem que haja colisão entre trilhas. Na representação das atividades que compõem um projeto em larga escala é possível se calcular qual o tempo total de projeto dado o tempo requerido para cada subtarefa e a sequência em que as sub-tarefas devem ser executadas. Além dessas áreas citadas, os grafos desempenham papel importante na análise de circuitos elétricos, modelagem e análise de sistemas de manufatura, etc.

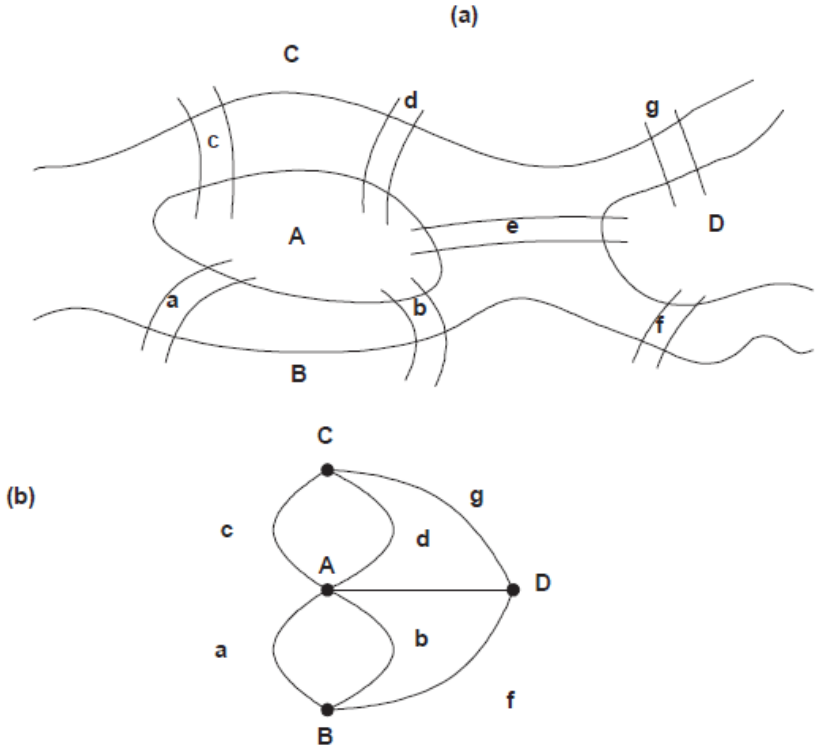

Figura 1: (a) Seção do rio Pregal em Kognisberg. (b) Representação através de grafo

2 Terminologia e Notação

Um grafo G é uma coleção de vértices V e arcos E. Cada arco pode ser representada através de um par de vértices. $V(G)$ e $E(G)$ são respectivamente o conjunto de vértices e arcos do grafo $G(V, E)$.

Um grafo *não direcionado* não possui direção especial em relação aos vértices que representam qualquer arco. Assim sendo, os pares (v_1, v_2) e (v_2, v_1) representam o mesmo arco.

Num grafo *direcionado* cada arco é representado por um vértice inicial e um vértice final. Por exemplo, (v_1, v_2) onde v_1 é o vértice de início e v_2 é o vértice final. Dessa forma, os pares (v_1, v_2) e (v_2, v_1) representam bordas diferentes. Muitas vezes existe um peso $w_{(v_1,v_2)}$ associado a cada borda que pode ser entendido como custo, distância, etc. de acordo com a aplicação.

Um grafo pode ser ilustrado graficamente através da utilização de círculos para a representação dos vértices e linhas conectando os vértices representando os arcos. A Figura 2 ilustra três grafos G_1,G_2 e G_3. Os grafos G_1 e G_2 são *não direcionados* enquanto que G_3 é um grafo *direcionado*.O grafo G_2 é também uma árvore. As árvores podem ser definidas como casos especiais de grafos.

3 Algoritmo de caminho mínimo - Dijkstras's Algorithm

A aplicação mais frequente de grafos é a utilização de algoritmos para se encontrar o caminho mínimo entre os vértices em um grafo direcionado ou não direcionado com pesos positivos.

O grafo pode representar um circuito elétrico com os arcos representando a distância entre conectores, fases de um projeto de construção civil, etc. Como os pesos dos arcos são geralmente interpretados como

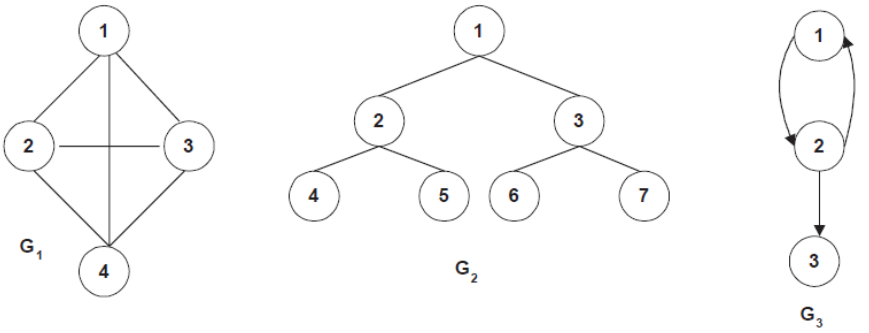

Figura 2: Exemplos de grafos.

custos, o problema é frequentemente descrito como a descoberta do caminho de custo mínimo entre um determinado vértice e todos os outros. Esse algoritmo foi primeiramente descrito por Edsger W. Dijkstra e por isso leva o seu nome.

Num problema com `V` vértices inicialmente deve-se descobrir um conjunto de vértices dos quais são conhecidas as menores distâncias em relação ao vértice inicial aqui denominado 1; esse conjunto de vértices será denominado `S`. O vértice 1 é por *default* o primeiro vértice a ser colocado no conjunto `S`. Obviamente o caminho mais curto entre o vértice 1 e o mesmo é zero.

O algoritmo se desenrola num processo de duas etapas, adicionando vértices no conjunto `S` até que todos pertençam a ele:

1. Pegue um vértice que não se encontra em `S` e denomine-o `M`, e para o qual a distância entre 1 a `M` é mínimo. Adicione `M` em `S`.

2. Para cada vértice ainda não em `S`, verifique se a distância entre esse vértice e o vértice 1 pode ser encurtada utilizando o vértice `M`. Se é possível então atualize o elemento correspondente no array `Dist[]`.

Adicionalmente, deve ser criado um array `Path[]` que contêm para cada vértice, o vértice que é o seu antecessor para o caminho mais curto a partir do vértice 1.

Esse algoritmo pode ser descrito da seguinte forma:

```
S := {1};
inicialize o array Dist[] com o peso dos arcos conectados ao vértice 1;
  para i := 1 até V-1 faça:
     {
      escolha um vértice M, que não esteja em S para o qual Dist[M] é minimo;
      adicione o vertice M ao conjunto S;
      para cada vertice J ainda não em S faça:
           Dist[J] := min(Dist[J], Dist[M]+Edge[M,J]);
           Path[J] := M
     }
```

Para melhor entendimento desse algoritmo apresenta-se um exemplo considerando o grafo direcionado ilustrado na Figura 3.

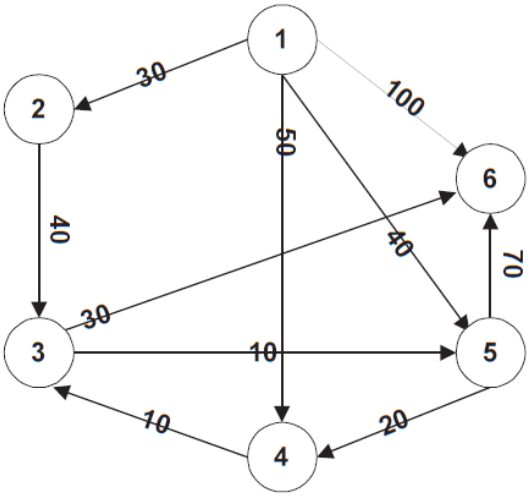

Figura 3: Grafo direcionado.

Os passos de execução do algoritmo podem ser resumidos através da Tabela 1 a seguir:

Iter	S	M	Dist[2]	Dist[3]	Dist[4]	Dist[5]	Dist[6]
Inicial	{1}	-	30	Infinito	50	40	100
1	{1,2}	2	30	70	50	40	100
2	{1,2,5}	5	30	70	50	40	100
3	{1,2,5,4}	4	30	60	50	40	100
4	{1,2,5,4,3}	3	30	60	50	40	90
5	{1,2,5,4,3,6}	6	30	60	50	40	90

Tabela 1: Execução do algoritmo passo a passo.

Os elementos do array `Dist[]` são inicializados como sendo os pesos das bordas conectadas ao vértice 1. O peso da borda não existente entre o vértice 1 e 3 é colocado como infinito. Nesse ponto temos `S={1}`.

A primeira iteração procura um vértice que não está em `S` para o qual o valor correspondente de `Dist[]` seja o menor. Ou seja, o vértice 2, então `S={1,2}`. No segundo passo da iteração checa-se se é possível encurtar a distância entre o vértice 1 e um outro vértice se utilizando do novo vértice que foi incorporado em `S`. A resposta é afirmativa pois para se chegar em 3 se utilizando do vértice 2 a distância cai de infinito para 70. O conteúdo de `Dist[3]` deve ser atualizado adequadamente. Nenhuma outra distância pode ser encurtado se utilizando do vértice 2.

A segunda iteração descobre que dentre os vértice que não pertencem a `S`, o vértice 5 é o que tem a menor distância. O vértice 5 deve então ser acrescentado a `S`, então `S={1,2,5}`. Nenhum caminho para outros vértices pode ter a sua distância encurtada através da utilização do vértice 5.

A terceira iteração determina que o vértice 4 tem o menor valor de `Dist[]` entre os vértices que não estão em `S`, então temos `S={1,2,5,4}`. Agora percebemos que é possível encurtar o caminho entre 1 e 3 através do vértice 4.

Na quarta iteração o vértice 3 é adicionado no conjunto `S={1,2,5,4,3}`, e descobre-se que é possível encurtar o caminho entre 1 e 6 através do vértice 3 passando pelo vértice 4.

A última iteração adiciona 6 em S. Todos os vértices agora passam para o conjunto S, significando que foram achados todos os menores caminhos entre o vértice 1 e os outros vértices.

4 Representação de grafos na linguagem Python

Um grafo pode ser representado de diversas formas, por exemplo, como array ou como listas.

Para esse EP você deverá utilizar *dicionários*, um tipo de dado da linguagem Python, que é bastante conveniente para a representação de grafos.

Por exemplo, um grafo não direcionado sem peso com 5 vértices e 5 arcos é descrito a seguir:

- A -> C
- B -> C, E
- C -> A, B, D, E
- D -> C
- E -> C, B

pode ser representado como *dicionário* através da seguinte forma:

```
graph = { 'A' : {'C'},
          'B' : {'C', 'E'},
          'C' : {'A', 'B', 'D', 'E'},
          'D' : {'C'},
          'E' : {'C', 'B'},
        }
```

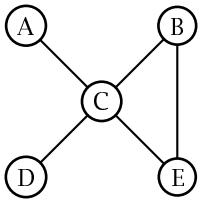

Figura 4: Grafo não direcionado sem peso

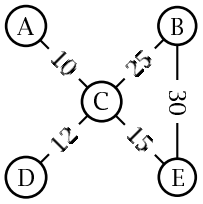

Figura 5: Grafo não direcionado com peso

Para o caso do grafo não direcionado com peso, ilustrado na figura 5, a representação na linguagem Python através de *dicionário* pode ser realizada como abaixo:

```
graph = { 'A' : {'C':10},
        'B' : {'C':25, 'E':30},
        'C' : {'C':10, 'B':25, 'D':12, 'E':15},
        'D' : {'C':12},
        'E' : {'C':15, 'B':30},
       }
```

O conteúdo do dicionário pode ser acessado da seguinte forma:

```
In [20]: graph['C']
Out[20]: {'A': 10, 'B': 25, 'D': 12, 'E': 15}

In [21]: graph['C']['B']
Out[21]: 25
```

5 Para você fazer:

O problema principal de logística de transporte terrestre é a minização da distância a ser percorrida entre os diversos pontos de entrega e/ou coleta de mercadorias. O problema pode ser modelado através de um grafo não direcionado.

Como exemplo apresenta-se, na figura 6, um grafo hipotético onde os vértices representam as capitais dos estados do Brasil e os arcos com peso representam as rodovias e suas respectivas distâncias.

Na figura 6 os vértices do grafo são identificados por *labels* $(A \ldots W)$. As correspondências *label*-cidade são apresentadas a seguir:

- A: Porto Alegre, B: Florianópolis, C: Curitiba,
- D: São Paulo, E: Rio de Janeiro, F: Campo Grande,
- G: Belo Horizonte, H: Vitória, I: Cuiabá,
- J: Goiânia, K: Salvador, L: Rio Branco,
- M: Porto Velho, N: Palmas, O: Aracaju,
- P: Maceió, Q: Recife, R: João Pessoa,
- S: Natal, T: Fortaleza, U: Teresina,
- V: São Luis, X: Belém, Y: Macapá,
- Z: Manaus, W: Boa Vista.

Os seguintes passos devem estar presentes no seu programa:

1. Estrutura de dados: o tipo de dado para representação de grafos será a classe denominada `Grafo`.

   A classe `Grafo` deverá utilizar um dicionário de dicionários para a representação do grafo.

   Além do método constructor, os seguintes métodos devem constar:

   (a) `AcrescentaVertice(v)`: Acrescenta um novo vértice `v`.

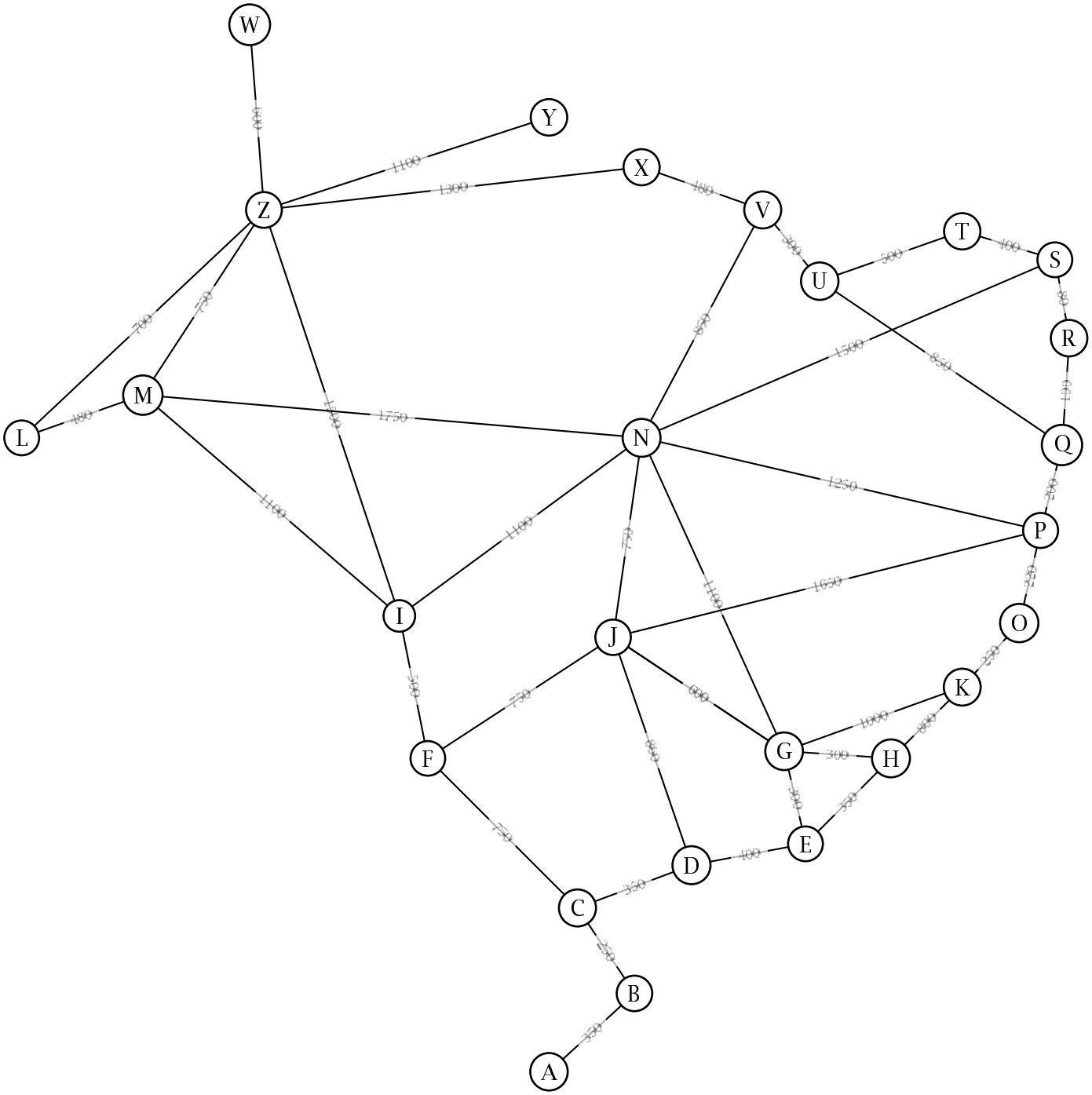

Figura 6: Grafo não direcionado com peso representando as capitais dos estados e suas interconexões rodoviárias.

(b) `RemoveVertice(v)`: Remove o vértice `v` (Obs: não deve remover se houverem arcos conectados ao vértice).

(c) `AcrescentaArco(v1,v2)`: Acrescenta um novo arco entre os vértices `v1` e `v2`.

(d) `RemoveArco(v1,v2)`: remove o arco entre `v1` e `v2`.

(e) `Path <- AchaMenorCaminho(v1,v2)`: acha o menor caminho entre `v1` e `v2`, `Path` é uma lista que contêm a sequencia dos vértices a serem visitados.

(f) ImprimeGrafo(): imprime na tela uma representação alfanumérica do seu grafo, por exemplo:

```
A -> C:10
B -> C:25 E:30
C -> A:10 B:25 D:12 E:15
D -> C:12
E -> C:15 B:30
```

2. Algoritmos de menor distância: você deve implementar os algoritmos de Djikstra.

3. Entrada de dados: o grafo em questão deve ser lido de um arquivo. Nesse arquivo cada linha representa um arco com o seu vértice inicial e final e o seu peso. A seguir apresenta-se um exemplo:

```
A C:10
B C:25 E:30
C A:10 B:25 D:12 E:15
D C:12
E C:15 B:30
```

   Você deve observar a Figura 6 e contrstruir o arquivo de entrada de dados. **OBS:** Lembre-se que para a construção do dicionário deve-se utilizar as strings correspondentes, 'A', 'B', 'C', etc.

4. Utilizando a infraestrutura descrita acima construa o seu programa principal de modo a responder as seguintes perguntas:

   (a) Qual a menor distância entre as seguintes capitais:

   - A - Y
   - G - Y
   - D - U
   - Q - J
   - V - K
   - L - H
   - T - W
   - X - K

5. As respostas devem ser impressas na tela e também em um arquivo em formato ASCII.

6. Escreva um relatório de no máximo uma página A4 contendo as seguintes informações:

(a) Uma descrição da sua solução de maneira resumida.

(b) Uma análise dos resultados obtidos com os arquivos teste.

7. O código fonte do seu programa em conjunto com o seu relatório no formato PDF devem ser submetidos no sistema Moodle.

8. **DEADLINE: 17/05/2018 - 23h59min.**