PMR3201 - Computação para Automação
Exercício Programa 2 - 2019

Algoritmo de caminho mínimo - Transporte logístico multimodal

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

Deadline: 09/05/2019 - 23h59min (SISTEMA MOODLE)

1 Introdução

A primeira evidência de que se tem notícia quanto à aplicação de grafos remonta ao ano 1736 em que Leonhard 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 Kneiphof, dividindo-se em seguida em duas partes.

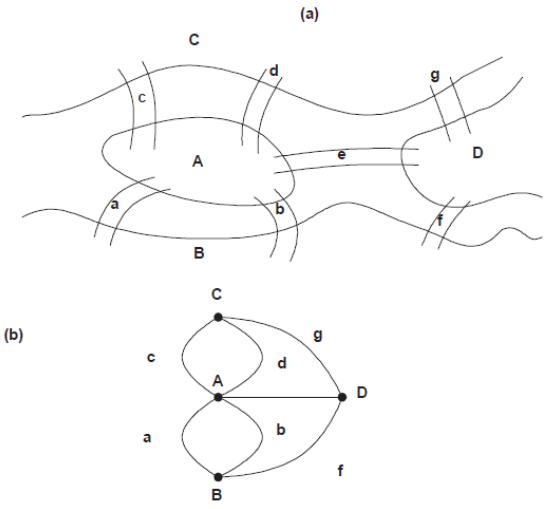

Figura 1: (a) Seção do rio Pregal em Kögnisberg. (b) Representação através de grafo.

Assim sendo, existem quatro áreas de terra que ladeiam o rio (Vide Figura 1). Essas áreas de terra estão interligadas por sete pontes, $(a \dots g)$. As áreas de terra estão assinaladas pelas letras $(A \dots D)$. O problema da ponte de Kögnisberg consiste em determinar se ao partir de alguma área de terra é possível atravessar todas as pontes exatamente uma única 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.

Um link para o artigo original é apresentado aqui `http://www.math.dartmouth.edu/~euler/docs/originals/E053.pdf`

Muitos outros problemas podem ser naturalmente formulados através de grafos. Por exemplo, pode-se tentar estabelecer qual é a menor distância entre duas cidades sendo que existem vários caminhos alternativos passando por cidades distintas. No

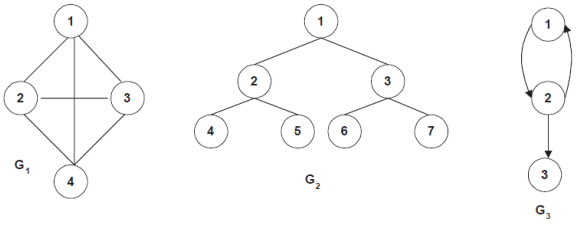

Figura 2: Exemplos de grafos.

projeto de trilhas de conexão de componentes em placas de circuito impresso é possível determinar trilhas de tal forma a serem de menor comprimento possível e sem que haja colisão entre elas. Na representação das atividades que compõem um projeto em larga escala é possível calcular qual o tempo total de projeto dado o tempo requerido para cada sub-tarefa 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.

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 arcos diferentes. Muitas vezes existe um peso $w_{(v_1,v_2)}$ associado a cada arco que pode ser entendido como custo, distância, etc. de acordo com a aplicação.

Um grafo pode ser representado 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 as fases de um projeto de construção civil com os pesos dos arcos interpretados como 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.

Em um 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`, 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.

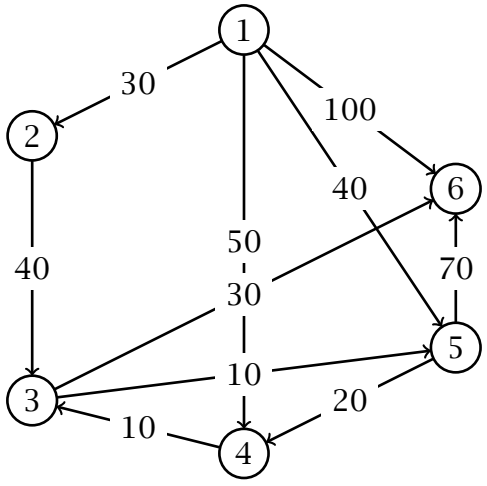

Figura 3: Grafo direcionado.

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 dos arcos conectados ao vértice 1. O peso do arco 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 utilizando-se 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 encurtada utilizando-se do vértice 2.

A segunda iteração descobre que entre os vértices 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'},
        }
```

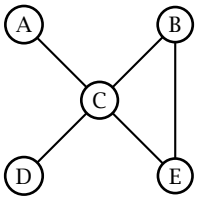

Figura 4: Grafo não direcionado sem peso

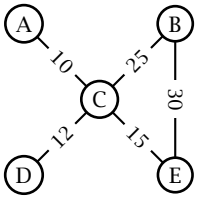

Figura 5: Grafo não direcionado com peso

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 Especificação do problema

O problema mais simples de logística terrestre é a escolha de um caminho que miniza a distãncia percorrida entre dois pontos. 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 ferrovias e suas respectivas distâncias. As ferrovias quando existem são representadas em azul e são paraleas às rodovias.

Na Figura 6 os vértices do grafo são identificados por *labels* (A . . . 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.

Uma empresa de operações logísticas de transporte rodoviário estabelece roteirizações que minimizam o caminho a ser percorrido. Eventualmente, caminhos mais longos podem ser vantajosos devido à velocidade máxima permitida, custo de combustível, custo de pedágio, etc. mas de uma maneira geral as operações logísticas unimodais tem a distância percorrida diretamente correlacionada ao tempo gasto e ao custo total da operação.

Em um sistema multimodal usualmente se estabelece um *trade-off* entre Tempo total TT × Custo total CT. As lojas *online* internacionais oferecem como opção ao consumidor a escolha de diferentes opções de frete com diferentes preços incluindo a opção de escolha de diferentes empresas de logística de transporte.

Dentro desse contexto é proposto aqui um problema de logística de transporte multimodal onde existem dois modais: um sistema rodoviário e um sistema ferroviário. O sistema rodoviário atinge todos as capitais dos estados mas o sistema ferroviário só cobre uma parte do território nacional.

Para cada trecho rodoviário ou ferroviário representado por um arco é atribuído um conjunto de pesos, i.e., um par de valores $(t_{\alpha\beta}, c_{\alpha\beta})$, onde $t_{\alpha\beta}$ é o tempo de percurso e $c_{\alpha\beta}$ o custo do percurso entre os vértices α e β.

Deseja-se descobrir qual o caminho entre dois vértices que resulta no menor tempo total $TT_{\alpha\beta}$ e o caminho que resulta no menor custo total $CT_{\alpha\beta}$

A seguir apresenta-se os trechos rodoviários e os pesos $(t_{\alpha\beta}, c_{\alpha\beta})$ correspondentes:

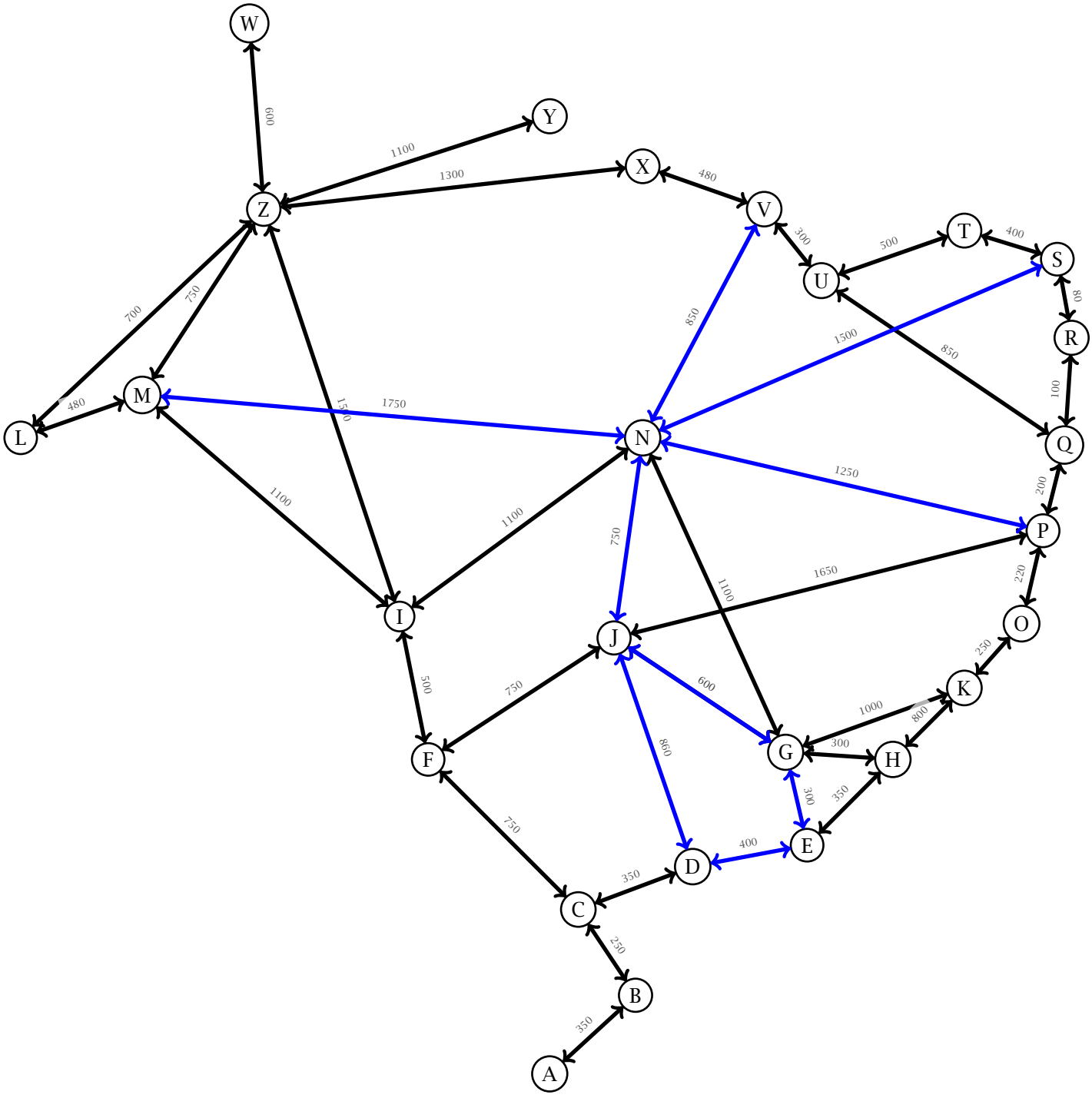

Figura 6: Grafo representando as capitais dos estados e suas interconexões rodoviárias e ferroviárias. Ferrovias em azul são paralelas às rodovias.

1. (A,B): (3.5, 200)
2. (B,C): (2.5, 200)
3. (C,D): (3.5, 200)
4. (C,F): (7.5, 500)
5. (D,E:) (4, 200)
6. (E,G): (3, 200)
7. (E,H): (3.5, 200)
8. (G,H): (3, 200)
9. (G,K): (10, 500)
10. (G,N): (11, 1000)
11. (G,J): (6, 500)
12. (F,J): (7.5, 500)
13. (F,I): (7.5, 500)
14. (I,M): (11, 1000)
15. (I,Z): (1500, 1000)
16. (J,N): (7.5, 500)
17. (G,J): (6, 500)
18. (N,P): (12.5, 1000)
19. (N,V): (8.5, 500)
20. (N,M): (17.5, 1500)
21. (N,S): (15, 1000)
22. (H,K): (8, 500)
23. (K,O): (2.5, 200)
24. (O,P): (2.2, 200)
25. (P,Q): (2, 200)
26. (Q,R): (1, 200)
27. (R,S): (0.8, 200)
28. (S,T): (4, 200)
29. (T,U): (5, 200)
30. (U,V): (3, 200)
31. (V,X): (4.8, 200)
32. (X,Z): (13, 100)
33. (Z,Y): (11, 1000)
34. (Z,W): (6, 500)
35. (Z,M): (7.5, 500)
36. (Z,L): (7, 500)
37. (L,M): (4.8, 200)
38. (I,N): (11, 1000)
39. (J,D): (8.6, 500)
40. (J,P): (16.5, 1500)
41. (U,Q): (8.5, 500)

A seguir apresenta-se os trechos ferroviários e os pesos $(t_{\alpha\beta}, c_{\alpha\beta})$ correspondentes:

1. (D,E): (8, 200)
2. (E,G): (6, 200)
3. (J,N): (15, 300)
4. (G,J): (12, 300)
5. (N,P): (25, 300)
6. (N,V): (17, 300)
7. (N,M): (35, 500)
8. (J,D): (17.4, 300)
9. (N,S): (30, 400)

5.1 Representação de diferentes modais

Os diferentes modais podem ser representados como se fossem grafos distintos. Podemos representar os vértices da malha rodoviária acrescentando a letra R aos *labels*: {AR, ..., WR}. Da mesma, forma os vértices da malha ferroviária são representados através do seguinte conjunto de *labels* {DT,ET,GT,JT,MT,NT,PT,VT}.

Os dois grafos podem ser representados numa mesma estrutura já que DR e DT por exemplo são interpretados como vértices distintos muito embora se referem ao mesmo local. Para cada par de vértices representando o mesmo local dois arcos unidirecionais cujos pesos representam o custo de troca de modal.

Por exemplo, vamos supor o grafo hipotético aonde {AT,BT,CT} são os vértices que definem uma malha ferroviária e {BR,DR,ER} os vértices de uma malha rodoviária. A mudança de modal só pode ser feita através dos vértices BR e BT. Existe um custo de mudança de modal distinto para cada direção. Para uma mudança do modal rodoviário (BR) para o modal ferroviário (BT) existe um custo p enquanto que o custo é q no sentido inverso.

Através dessa representação é possível utilizar o algoritmo de Djkistra como apresentado acima.

Uma possível representação na linguagem Python é apresentada a seguir:

```
graph = { 'AT' : {'BT':x},
      'BT' : {'AT':x, 'CT':y, 'BR':q},
      'CT' : {'BT':y},
      'DR' : {'BR':z},
      'BR' : {'BT':p, 'DR':z, 'ER':w},
```

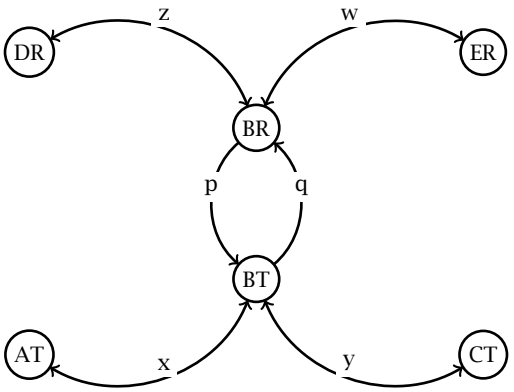

Figura 7: Exemplo de grafo com mudança de modal em (BT,BR).

```
        'ER' : {'BR':w}
    }
```

Para o modelo do sistema multimodal proposto existem pesos fixos de tempo e custo para a troca da carga da parte rodoviária para a parte ferroviária por exemplo para um nó B que possui conexão rodoviária e ferroviária:

- (BR,BT): $(t_{BR,BT}, c_{BR,BT}) = (6, 100)$.

E pesos distintos para o sentido inverso:

- (BT,BR): $(t_{BT,BR}, c_{BT,BR}) = (2, 50)$.

Os pesos podem ser interpretados como o custo logístico de tempo e monetário para fazer a mudança de modal.

6 Para você fazer

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. Para cada elemento do dicionário, que representa um vértice, deve ser armazenado um array contendo as informações de tempo e custo: $[t_{\alpha\beta}, c_{\alpha\beta}]$.

   Além do método constructor, os seguintes métodos devem constar:

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

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

   (c) `AcrescentaArco(v1,v2,w)`: Acrescenta um novo arco entre os vértices `v1` e `v2` com peso dado por `w=[t,c]`.

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

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

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

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

```
A -> C:[10,1]
B -> C:[25,3] E:[30,7]
C -> A:[10,2] B:[25,3] D:[12,6] E:[15,1]
D -> C:[12,5]
E -> C:[12,1] B:[30,2]
```

2. Algoritmo de menor tempo/custo: você deve implementar o algoritmo 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:[10,1]
B C:[25,3] E:[30,7]
C A:[10,2] B:[25,3] D:[12,6] E:[15,1]
D C:[12,5]
E C:[12,1] B:[30,2]
```

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

   (a) Qual o menor tempo total para a logística de transporte de carga entre as seguintes capitais:

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

   Deve ser indicado o caminho percorrido e qual modal utilizado em cada trecho incluindo o tempo e custo associado.

   (b) Idem para o menor custo total.

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

6. O código fonte do seu programa deve ser submetido no sistema Moodle.

7. **DEADLINE: 09/05/2019 - 23h59min.**

7 Referências

1. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". Introduction to Algorithms (Second ed.). MIT Press and McGraw–Hill. pp. 595–601. ISBN 0-262-03293-7.

2. Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". Numerische Mathematik. 1: 269–271.