Problemas do Caminho Mínimo e do Caixeiro Viajante

Marina Andretta

ICMC-USP

2 de março de 2019

Grafos

Grafos podem ser ferramentas bastante importantes na modelagem de alguns problemas.

Formalmente, um grafo G é dado por um par $G = (V, A)$, em que V é um conjunto de vértices e A é um conjunto de arestas. Uma aresta é um par (u, v), com $u, v \in V$.

Em alguns casos, podemos ter valores associados aos vértices ou às arestas do grafo.

Vamos denotar por n_G o número de vértices de G e m_G o número de arestas de G.

Grafos

Se o grafo G é orientado, uma aresta (u, v) indica que do vértice u atingimos o vértice v.

Se o grafo G é não orientado, uma aresta (u, v) indica que do vértice u atingimos o vértice v e de v atingimos u.

Nos nossos problemas, a menos que seja dito o contrário, usaremos grafos não orientados.

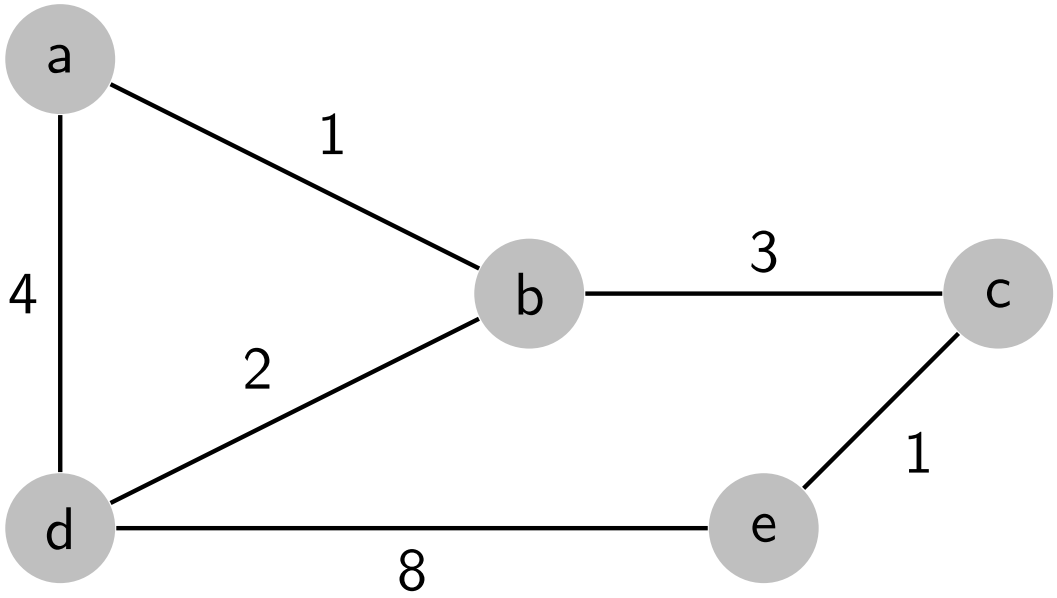
Representação de grafos

Podemos representar de diversas formas. Veremos algumas delas usando o exemplo de $G = (V, A)$, com

- $V = \{a, b, c, e, d\}$;
- $A = \{(a, b), (a, d), (b, c), (b, d), (c, e), (d, e)\}$;
- valor c associado a cada aresta, valendo $\{1, 4, 3, 2, 1, 8\}$, respectivamente.

Representação de grafos - desenho

Podemos usar um desenho:



Representação de grafos - matriz de adjacência

A matriz de adjacência M_G de G é uma matriz de $\{0,1\}^{n_G \times n_G}$ definida da seguinte forma:

$$M_{G\,ij} = \begin{cases} 1, & \text{se existe uma aresta do vértice } i \text{ ao vértice } j \text{ em } G; \\ 0, & \text{caso contrário.} \end{cases}$$

Representação de grafos - matriz de adjacência

A matriz de adjacência M_G de G é uma matriz de $\{0,1\}^{n_G \times n_G}$ definida da seguinte forma:

$$M_{G\,ij} = \begin{cases} 1, & \text{se existe uma aresta do vértice } i \text{ ao vértice } j \text{ em } G; \\ 0, & \text{caso contrário.} \end{cases}$$

No nosso exemplo, a matriz de adjacência de G é dada por

$$M_{G\,ij} = \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array}$$

Algumas definições

O grau de um vértice v de um grafo G é denotado por $g(v)$ (ou $g_G(v)$) e representa o número de arestas que "saem" de v.

No nosso exemplo, $g(a) = 2$ e $g(d) = 3$.

Algumas definições

O grau de um vértice v de um grafo G é denotado por $g(v)$ (ou $g_G(v)$) e representa o número de arestas que "saem" de v.

No nosso exemplo, $g(a) = 2$ e $g(d) = 3$.

Se existe uma aresta (u, v) em G, dizemos que os vértices u e v são vizinhos ou adjacentes.

No nosso exemplo, os vizinhos de e são $\{c, d\}$. E os vizinhos de b são $\{a, c, d\}$.

Algumas definições

Um caminho em um grafo $G = (V, A)$ é uma sequência vértices de V, $(v_{i_1}, v_{i_2}, ..., v_{i_k})$ de forma que $(v_{i_j}, v_{i_{j+1}})$ é uma aresta de G, para $j = 1, ..., k-1$. Além disso, não pode haver arestas repetidas.

No nosso exemplo, (a, b, d, e) é um caminho. Já (a, b, e) não é um caminho, porque não há aresta do vértice b ao vértice e.

Algumas definições

Um caminho em um grafo $G = (V, A)$ é uma sequência vértices de V, $(v_{i_1}, v_{i_2}, ..., v_{i_k})$ de forma que $(v_{i_j}, v_{i_{j+1}})$ é uma aresta de G, para $j = 1, ..., k-1$. Além disso, não pode haver arestas repetidas.

No nosso exemplo, (a, b, d, e) é um caminho. Já (a, b, e) não é um caminho, porque não há aresta do vértice b ao vértice e.

Um ciclo (ou circuito) em um grafo $G = (V, A)$ é um caminho em que o vértice inicial é igual ao vértice final.

No nosso exemplo, (a, b, d, a) é um ciclo. E (a, b, c, e, d) também é um ciclo.

Problema do Caminho Mínimo

Na aula passada terminamos pensando em como modelar o Problema do Caminho Mínimo.

Sem perda de generalidade, podemos pensar que neste problema é dado um grafo $G = (V, A)$ (neste caso, não orientado), com vértices $V = \{v_1, v_2, ..., v_n\}$, e custos c_{ij} para cada aresta (v_i, v_j) em A. Para facilitar a notação, se existe uma aresta (v_i, v_j) em A, dizemos que (v_j, v_i) também está em A.

Como esperado, o custo de um caminho é dado pela soma dos custos das arestas usadas no caminho.

Queremos encontrar um caminho de v_1 a v_n que tenha o menor custo possível.

Problema do Caminho Mínimo

Claramente este é um problema importante de ser resolvido na prática.

Existem muitas variações deste problema. Algumas delas são:

- casos em que os custos das arestas podem ser negativos, casos em que não podem;
- casos em que o grafo é orientado, casos em que não é;
- casos em que deseja-se determinar o caminho de menor custo entre 2 vértices específicos, casos em que deseja-se o caminho de menor custo entre um dado vértice e todos os outros.

Existem também muitas formas de resolver este problema (e suas variações). Vamos ver algumas delas.

Modelagem do Problema do Caminho Mínimo como um problema de otimização

Uma forma de modelar o Problema do Caminho Mínimo como um problema de otimização é a dada a seguir.

Defina uma variável x_{ij} para cada aresta (v_i, v_j) em A.

A variável x_{ij} terá valor 1 caso v_i e v_j apareçam nesta ordem no caminho escolhido (ou seja, no caminho escolhido, vai-se do vértice v_i para o vértice v_j).

Caso contrário, x_{ij} terá valor 0.

Função objetivo

Com essas variáveis definidas, o objetivo passa a ser minimizar a função

$$\sum_{i,j|(v_i,v_j)\in A} c_{ij}x_{ij}.$$

Restrições

Agora vamos definir as restrições do modelo.

A primeira coisa que vamos impor é que haja apenas uma aresta "saindo" de v_1 e nenhuma "entrando".

Restrições

Agora vamos definir as restrições do modelo.

A primeira coisa que vamos impor é que haja apenas uma aresta "saindo" de v_1 e nenhuma "entrando".

Isso pode ser feito com as restrições

$$\sum_{j|(v_j,v_1)\in A} x_{j1} = 0 \quad \text{e} \quad \sum_{j|(v_1,v_j)\in A} x_{1j} = 1.$$

Ou ainda,

$$\sum_{j|(v_j,v_1)\in A} x_{j1} - \sum_{j|(v_1,v_j)\in A} x_{1j} = -1.$$

Restrições

Precisamos também impor que haja apenas uma aresta "chegando" em v_n e nenhuma "saindo".

Restrições

Precisamos também impor que haja apenas uma aresta "chegando" em v_n e nenhuma "saindo".

Isso pode ser feito com as restrições

$$\sum_{j|(v_j,v_n)\in A} x_{jn} = 1 \quad \text{e} \quad \sum_{j|(v_n,v_j)\in A} x_{nj} = 0.$$

Ou ainda,

$$\sum_{j|(v_j,v_n)\in A} x_{jn} - \sum_{j|(v_n,v_j)\in A} x_{nj} = 1.$$

Restrições

Precisamos também garantir que as arestas escolhidas formem um caminho.

Restrições

Precisamos também garantir que as arestas escolhidas formem um caminho.

Uma maneira de fazer isso seria pedindo que, para cada vértice v_i que faz parte do caminho, com exceção de v_1 e v_n, tanto a quantidade de arestas do caminho que "chegam" em v_i, com a quantidade que sai, fosse igual a 1. Mas não sabemos de antemão quais são os vértices que fazem parte do caminho.

Restrições

Uma alternativa é impor que, para cada vértice v_i, com exceção de v_1 e v_n, a quantidade de arestas do caminho que "chegam" em v_i é igual à quantidade de arestas do caminho que "saem" de v_i.

Restrições

Uma alternativa é impor que, para cada vértice v_i, com exceção de v_1 e v_n, a quantidade de arestas do caminho que "chegam" em v_i é igual à quantidade de arestas do caminho que "saem" de v_i.

Note que, mesmo que um vértice não faça parte do caminho, isso ainda deve valer (já que não deveria nem "chegar" e nem "sair" nenhuma aresta do caminho nesses vértices).

Restrições

Uma alternativa é impor que, para cada vértice v_i, com exceção de v_1 e v_n, a quantidade de arestas do caminho que "chegam" em v_i é igual à quantidade de arestas do caminho que "saem" de v_i.

Note que, mesmo que um vértice não faça parte do caminho, isso ainda deve valer (já que não deveria nem "chegar" e nem "sair" nenhuma aresta do caminho nesses vértices).

Apenas com essa restrição, poderia acontecer de um vértice ter várias arestas do caminho "chegando" (e a mesma quantidade "saindo").

Restrições

Uma alternativa é impor que, para cada vértice v_i, com exceção de v_1 e v_n, a quantidade de arestas do caminho que "chegam" em v_i é igual à quantidade de arestas do caminho que "saem" de v_i.

Note que, mesmo que um vértice não faça parte do caminho, isso ainda deve valer (já que não deveria nem "chegar" e nem "sair" nenhuma aresta do caminho nesses vértices).

Apenas com essa restrição, poderia acontecer de um vértice ter várias arestas do caminho "chegando" (e a mesma quantidade "saindo").

Mas, como estamos minimizando o custo das arestas usadas no caminho, isso não irá acontecer.

Restrições

Esta restrição pode ser escrita da seguinte forma

$$\sum_{j|(v_j,v_i)\in A} x_{ji} - \sum_{j|(v_i,v_j)\in A} x_{ij} = 0,$$

para todo $i = 2, ..., n-1$.

Modelo completo

Assim, nosso modelo fica:

$$
\begin{aligned}
\text{Minimizar} \quad & \textstyle\sum_{i,j|(v_i,v_j)\in A} c_{ij}x_{ij} \\
\text{sujeita a} \quad & \textstyle\sum_{j|(v_j,v_1)\in A} x_{j1} - \sum_{j|(v_1,v_j)\in A} x_{1j} = -1, \\
& \textstyle\sum_{j|(v_j,v_n)\in A} x_{jn} - \sum_{j|(v_n,v_j)\in A} x_{nj} = 1, \\
& \textstyle\sum_{j|(v_j,v_i)\in A} x_{ji} - \sum_{j|(v_i,v_j)\in A} x_{ij} = 0, \qquad i = 2, ..., n-1, \\
& x_{ij} \in \{0,1\}, \qquad \forall i,j|(v_i,v_j) \in A.
\end{aligned}
$$

Exemplo

Para o nosso exemplo de grafo, suponha que estamos interessados em encontrar o caminho de menor custo de a até e.

Neste caso, as variáveis são x_{ab}, x_{ba}, x_{ad}, x_{da}, x_{bc}, x_{cb}, x_{bd}, x_{db}, x_{ce}, x_{ec}, x_{de}, x_{ed}. E o modelo é

$$
\begin{aligned}
\text{Minimizar} \quad & x_{ab} + x_{ba} + 4x_{ad} + 4x_{da} + 3x_{bc} + 3x_{cb} + 2x_{bd} + 2x_{db} + \\
& x_{ce} + x_{ec} + 8x_{de} + 8x_{ed} \\
\text{sujeita a} \quad & x_{ab} + x_{ad} - (x_{ba} + x_{da}) = -1, \\
& x_{ce} + x_{de} - (x_{ec} + x_{ed}) = 1, \\
& x_{ab} + x_{cb} + x_{db} - (x_{ba} + x_{bc} + x_{bd}) = 0, \\
& x_{bc} + x_{ec} - (x_{cb} + x_{ce}) = 0, \\
& x_{ad} + x_{bd} + x_{ed} - (x_{da} + x_{db} + x_{de}) = 0, \\
& x_{ab}, x_{ba}, x_{ad}, x_{da}, x_{bc}, x_{cb}, x_{bd}, x_{db}, x_{ce}, x_{ec}, x_{de}, x_{ed} \in \{0, 1\}.
\end{aligned}
$$

Algoritmos para resolver o problema

Para encontrar uma solução para o modelo, há vários métodos.

Uma possibilidade é usar um programa já implementado que resolve problemas de Programação Linear Inteira (PLI) de maneira exata (ou seja, garante que a solução encontrada é ótima, quando ela existe.

Existem vários destes programas disponíveis, tanto comerciais como livres. Alguns exemplos são o CPLEX, Gurobi, SCIP, GLPK, dentre outros.

Para este modelo em particular, é possível mostrar que relaxando as restrições $x_{ij} \in \{0, 1\}$ para $0 \leq x_{ij} \leq 1$, a solução ótima sempre tem valores 0 ou 1 para as variáveis, para qualquer instância do problema. Por isso, programas que resolvem problemas de Programação Linear (PL), que são mais eficientes do que programas para resolver PLI, podem ser usados.

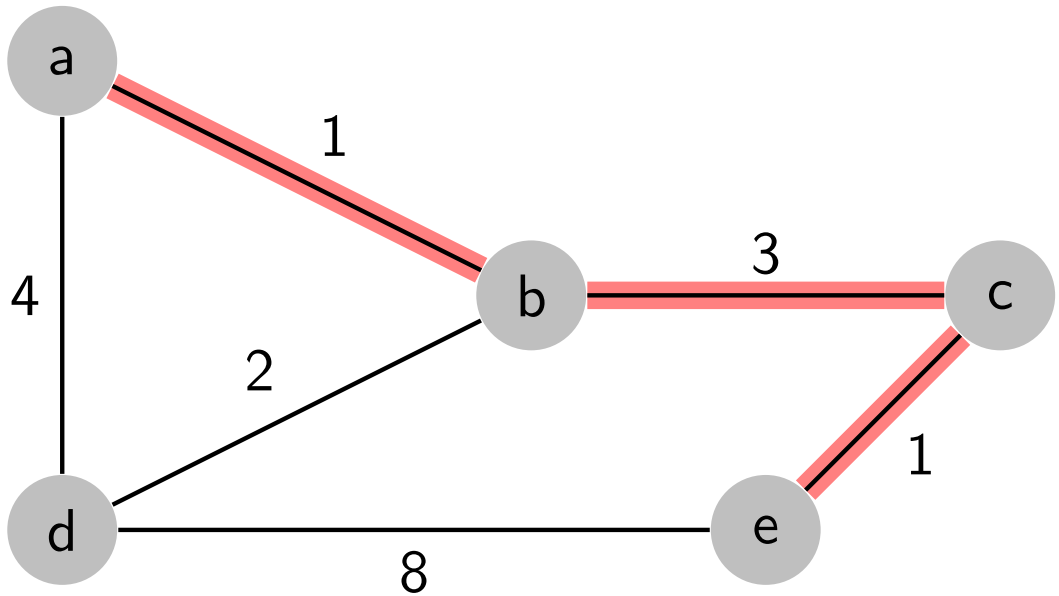
Algoritmos para resolver o problema

Usando o OpenSolver, no Google SpreadSheets, obtemos, para o nosso exemplo, a solução ótima $x_{ab} = x_{bc} = x_{ce} = 1$ e o restante das variáveis valendo 0. O valor da função objetivo nesta solução é 5.

Algoritmos para resolver o problema

Usando o OpenSolver, no Google SpreadSheets, obtemos, para o nosso exemplo, a solução ótima $x_{ab} = x_{bc} = x_{ce} = 1$ e o restante das variáveis valendo 0. O valor da função objetivo nesta solução é 5.

Isso significa que o caminho mínimo de a até e é dado pelos vértices (a, b, c, e) e o custo é 5.



Algoritmos para resolver o problema

Para este exemplo, em particular, é fácil ver que a solução obtida é ótima. Mas, em casos gerais, isso não é simples.

Por isso, esses algoritmos exatos têm toda uma teoria que garante que a solução encontrada (quando existe) é ótima.

Algoritmo guloso

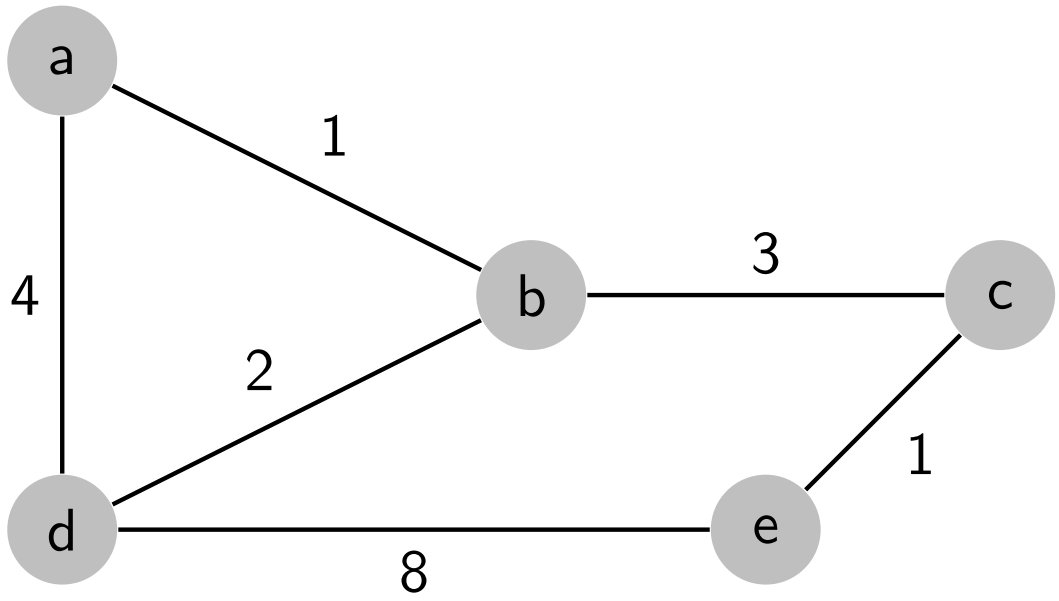
Uma outra possibilidade para encontrar uma solução para o modelo é usar um algoritmo diretamente no grafo, sem usar um programa de resolve problemas de otimização.

Uma estratégia simples e bastante intuitiva é um algoritmo guloso (ou *greed*, em inglês). Este algoritmo funciona da seguinte forma:

1. Defina w como o vértice v_1 e defina P como um caminho vazio.
2. Insira w no final de P.
3. Dentre todas as arestas que vão de w para algum vértice que ainda não está em P, escolha a aresta (w, u) com menor custo c_{wu}.
4. Faça w receber receber u.
5. Se u for o vértice v_n, pare e devolva P como solução. Senão, volte para o passo 2.

Algoritmo guloso

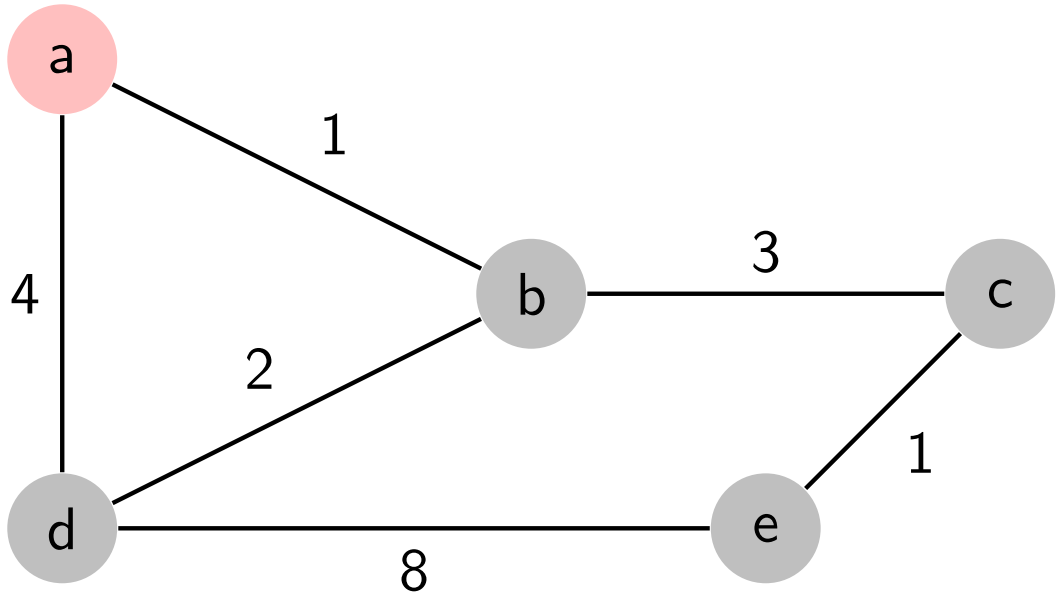
Voltando ao nosso exemplo, vamos executar este algoritmo e ver a solução encontrada.



$$P = ().$$

Algoritmo guloso

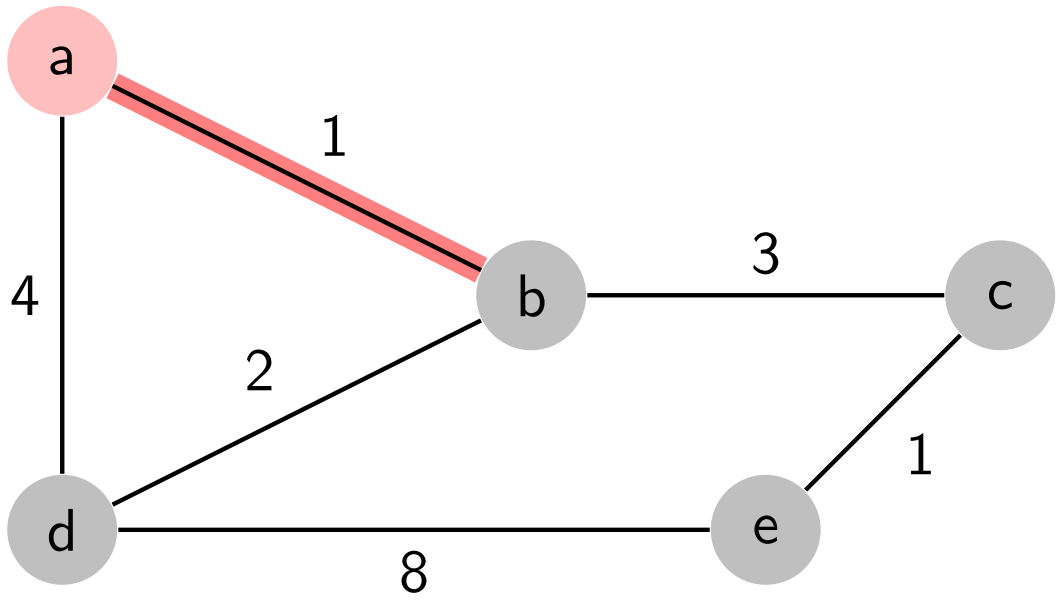
Primeiramente, selecionamos o vértice a e o colocamos em P.



$$P = (a).$$

Algoritmo guloso

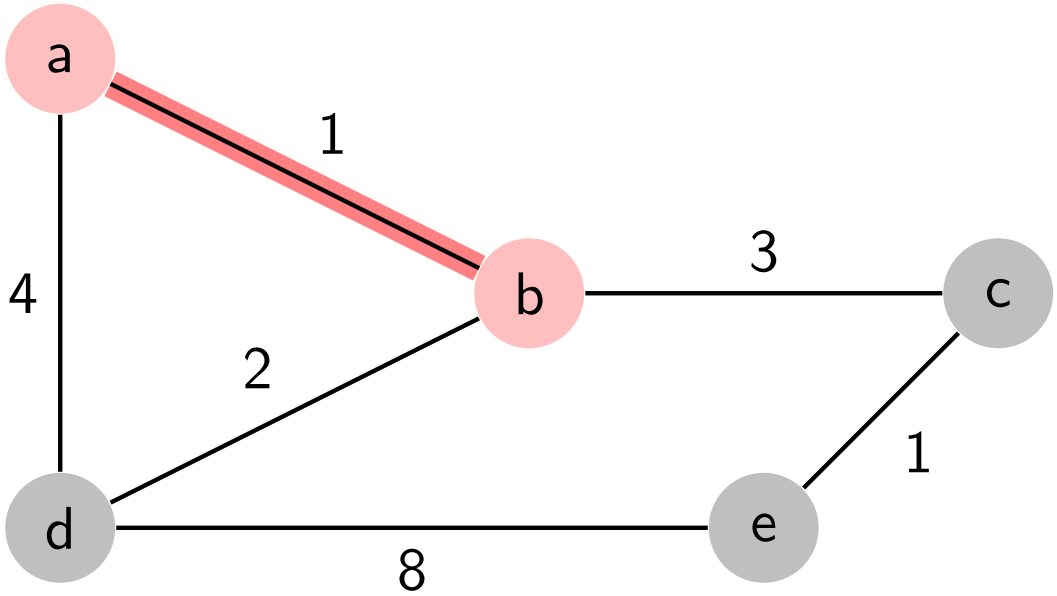
Dentre as arestas que saem de a, a de menor custo é (a, b).



$$P = (a).$$

Algoritmo guloso

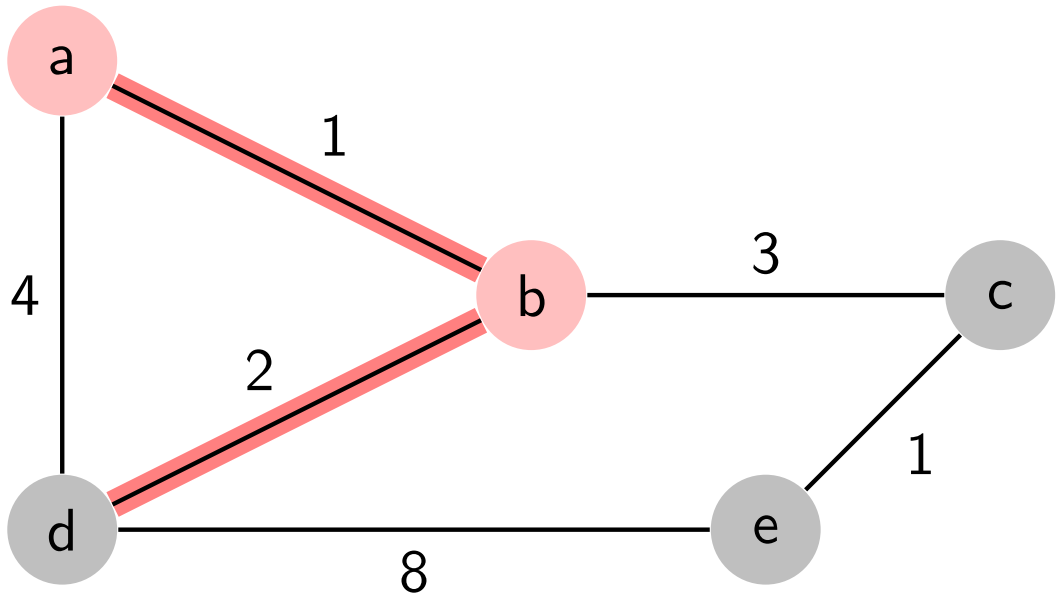
Então selecionamos b e o colocamos em P.



$$P = (a, b).$$

Algoritmo guloso

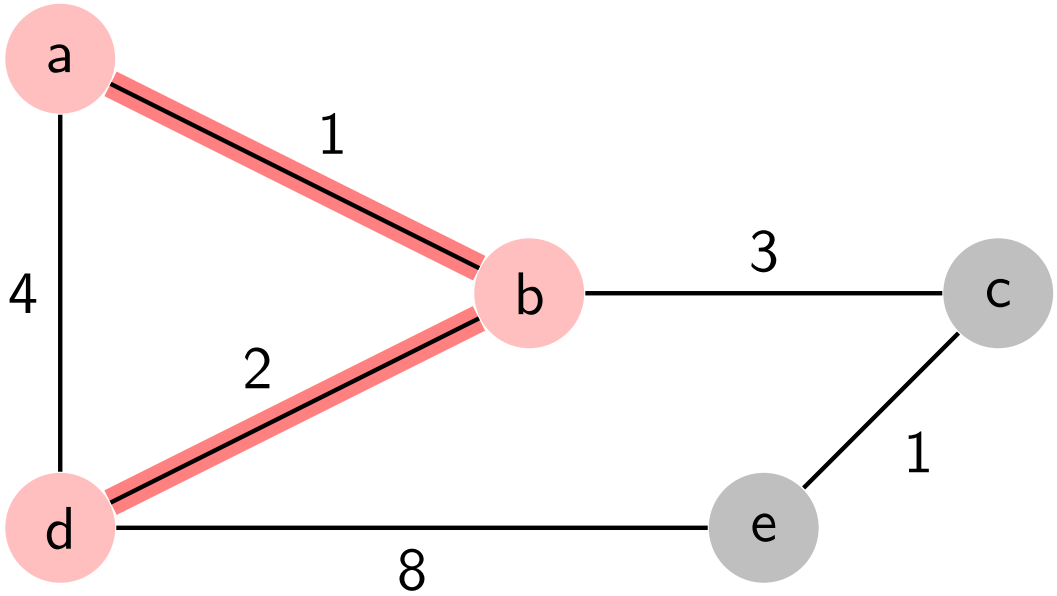
Dentre as arestas que saem de b e vão para vértices que não estão em P, a de menor custo é (b, d).



$$P = (a, b).$$

Algoritmo guloso

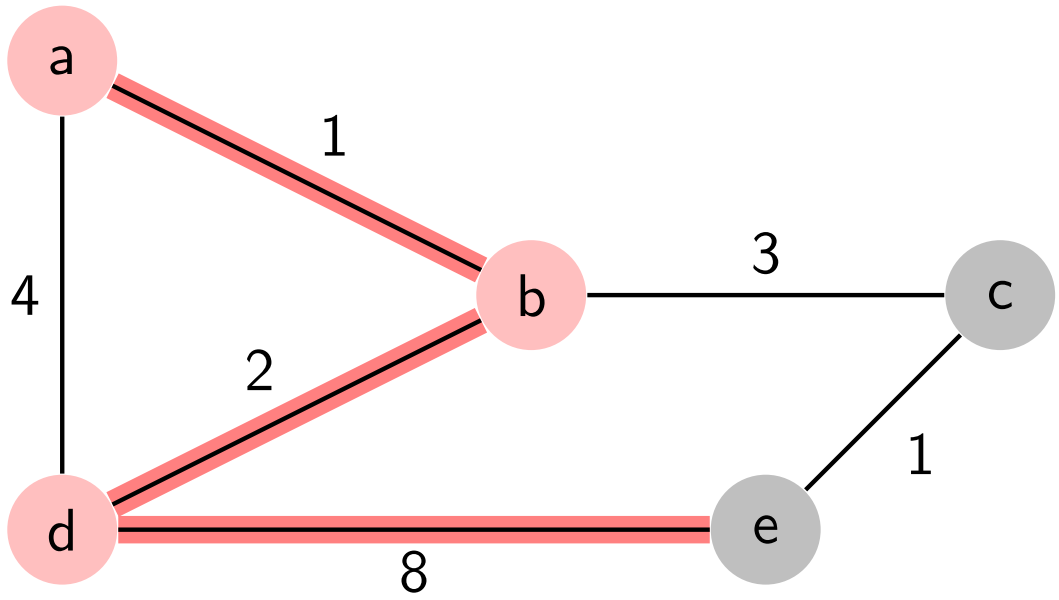
Então selecionamos d e o colocamos em P.



$$P = (a, b, d).$$

Algoritmo guloso

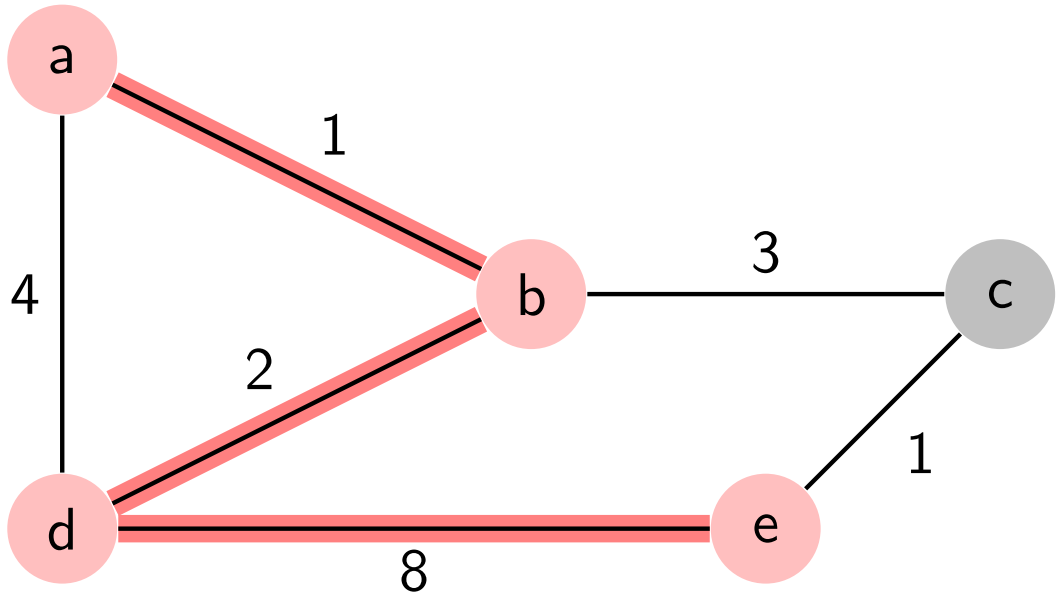
Dentre as arestas que saem de d e vão para vértices que não estão em P, a de menor custo é (d, e).



$$P = (a, b, d).$$

Algoritmo guloso

Então selecionamos e e o colocamos em P. Como e é o vértice final, o algoritmo pára com P como solução.



$$P = (a, b, d, e), \text{ com custo } 11.$$

Algoritmo guloso

Como podemos ver, não é garantido que o algoritmo guloso encontre a solução ótima. No entanto, em alguns casos, ele pode encontrar.

Por exemplo, se no exemplo o custo da aresta (b, c) fosse 1, o algoritmo guloso encontraria a solução (a, b, c, e), que seria a ótima neste caso.

Este algoritmo é um exemplo de heurística, que encontra uma solução de forma rápida, mas não garante que a solução seja ótima.

Algoritmo de Dijkstra

Talvez o algoritmo mais conhecido para resolver o Problema do Caminho Mínimo seja o algoritmo de Dijkstra.

Para este algoritmo existe uma demonstração que garante que ele sempre encontra uma solução ótima.

Além disso, é um algoritmo "rápido", ou seja, pode ser executado em tempo e espaço polinomial no tamanho da entrada.

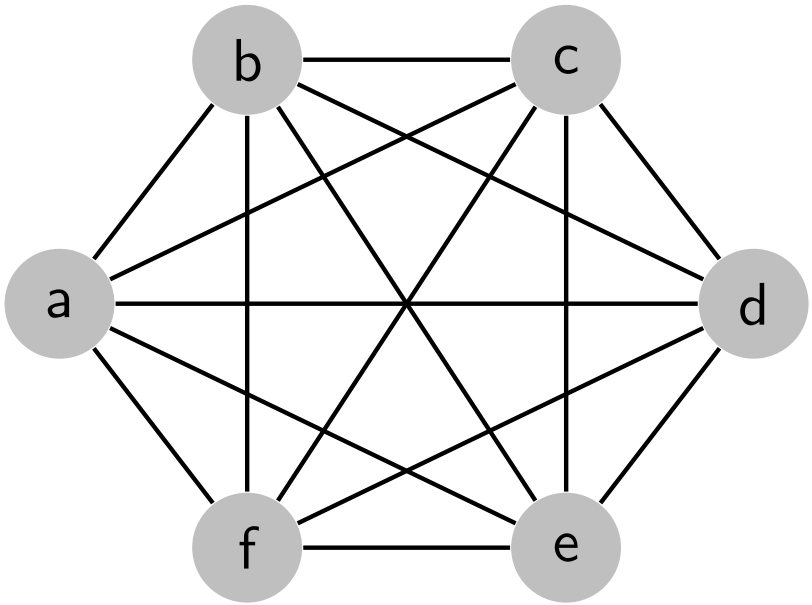
No entanto, ele funciona apenas para grafos com valores de custos positivos. Para o caso em que há custos negativos, existem outros algoritmos para resolver o problema, como o Algoritmo de Bellman-Ford, que também é polinomial, mas não tão eficiente como o Algoritmo de Dijkstra.

Problema do Caixeiro Viajante

Um outro problema bastante conhecido, que é um pouco parecido com o Problema do Caminho Mínimo, é o Problema do Caixeiro Viajante (ou *Travelling Salesperson Problem*, TSP).

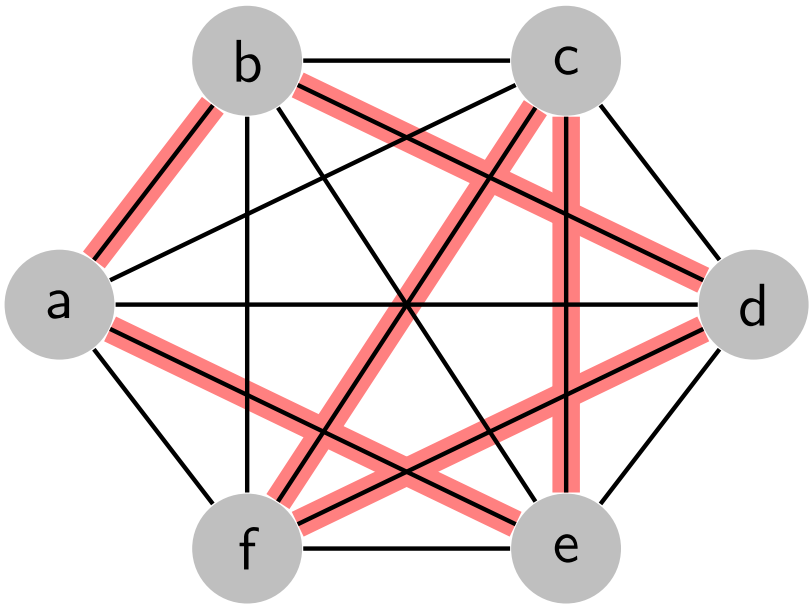
Neste problema temos também um grafo, com custos c não-negativos associados às arestas, e queremos encontrar um ciclo (ou circuito) que passa por todos os vértices exatamente uma vez (também chamado de ciclo hamiltoniano), de forma que o custo do ciclo seja mínimo.

Exemplo



	a	b	c	d	e	f
a	-	1	2	3	1	2
b	1	-	7	1	4	3
c	2	7	-	3	1	1
d	3	1	3	-	8	1
e	1	4	1	8	-	2
f	2	3	1	1	2	-

Exemplo



	a	b	c	d	e	f
a	-	1	2	3	1	2
b	1	-	7	1	4	3
c	2	7	-	3	1	1
d	3	1	3	-	8	1
e	1	4	1	8	-	2
f	2	3	1	1	2	-

O ciclo ótimo é dado por (a, b, d, f, c, e, a), com custo 6.

Neste caso, é fácil ver que ele é ótimo, porque o custo de todas arestas é maior ou igual a 1 e há 6 arestas no ciclo. Além disso, este é o único ciclo com custo 6.

Problema do Caixeiro Viajante

Como no caso do Problema do Caminho Mínimo, há diversas variações do Problema do Caixeiro Viajante. Algumas delas são:

- Muitas vezes, o grafo considerado neste problema é completo, ou seja, existem arestas para todo par de vértices do grafo. Este é o caso do grafo do exemplo anterior.

- Em alguns casos os custos nas arestas satisfazem a desigualdade triangular, ou seja, para todo conjunto de 3 vértices u, v e w, $c(u, v) \leq c(u, w) + c(w, v)$. Chamamos esta variação do problema de Problema do Caixeiro Viajante Métrico.

  No exemplo anterior, isso não acontece. Note que $c(a, d) = 3 > 1 + 1 = c(a, b) + c(b, d)$.

Problema do Caixeiro Viajante

- Em alguns casos o grafo é orientado e o custo de uma aresta que vai de u a v pode ser diferente do custo de v a u.

  Neste caso o problema é conhecido como Problema do Caixeiro Viajante Assimétrico.

Problema do Caixeiro Viajante

Em muitos casos o custo em cada aresta (u, v) representa a distância do vértice u ao vértice v.

A aplicação mais comum do TSP é o caso em que os vértices representam cidades, as arestas representam estradas que ligam as cidades e os custos das arestas representam o comprimento dessas estradas. O objetivo é passar por todas as cidades exatamente uma vez e voltar para a cidade de origem, de modo que a distância total percorrida seja a menor possível.

Este é um problema muito importante para a Ciência da Computação e a área de Otimização. Muitas informações e curiosidades podem ser encontradas aqui: http://www.math.uwaterloo.ca/tsp/index.html.

Exercício

Existem muitos problemas que, quando escritos usando grafos, podem ser vistos como uma variação do Problema do Caixeiro Viajante.

Pesquise esses problemas e apresente para a turma. Problemas diferentes e descritos claramente irã valer pontos.