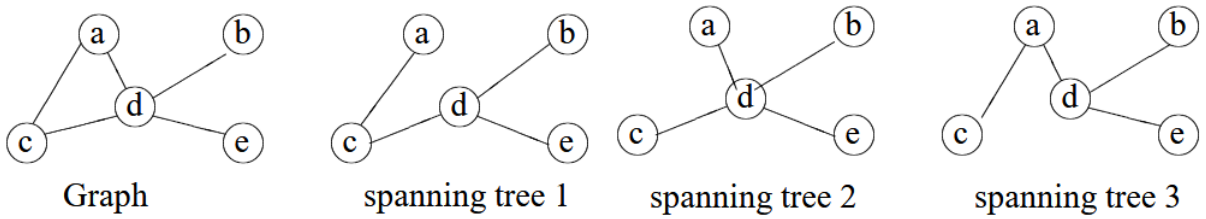
4.2.3 O problema da árvore geradora mínima 331

Problema da Árvore Geradora Mínima

The Minimum Spanning Tree-MST Problem - Árboles Generadores

Alguns problemas de otimização combinatória podem ser formulados através de um tipo de grafo específico conhecido como árvore. Uma árvore é um grafo conexo (existe caminho entre qualquer par de seus nós) e acíclico (sem ciclos). Ou seja, uma árvore é um grafo conexo sem ciclos.

Dado um grafo conectado e não orientado, uma árvore geradora desse grafo é um subgrafo que é uma árvore e conecta todos os nós. Um único grafo pode ter muitas árvores diferentes. Uma árvore de arborescência mínima (geradora mínima-MST) para um grafo ponderado, conectado e não orientado é uma árvore geradora com peso/custo/distância menor ou igual ao peso/custo/distância de todas as outras árvores geradoras. O custo de uma árvore geradora é a soma dos custos de cada aresta da árvore de arborescência.



http://www.cse.ust.hk/~dekai/271/notes/L07/L07.pdf

Modelagem matemática

Seja um grafo G=(N,E) não direcionado com N={1, 2, ..., n}. Suponhamos que o nó 1 oferta n−1 unidades de um produto e os demais nós demandam 1 única unidade deste produto. O modelo matemático para construir a árvore geradora mínima é:

Variáveis:

$y_{ij} \in \{0, 1\} \rightarrow$ se a aresta (i, j) é usada ou não

$x_{ij} \in \mathbb{Z}_+ \rightarrow$ quantidade de fluxo através da aresta (i, j)

Parâmetros:

$c_{ij} \rightarrow$ peso/custo/distância unitário do fluxo em (i, j), $c(i, j) \geq 0$

Função objetivo:

$$\min Z = \sum_{(i,j)\in E} c_{ij} y_{ij}$$

Restrições:

$$\sum_{(i,j)\in E} x_{ij} - \sum_{(j,k)\in E} x_{jk} = 1, \qquad j = 2, \dots, n$$

$$\sum_{(1,j)\in E} x_{1j} = n - 1$$

$$y_{ij} \leq x_{ij} \leq (n-1)y_{ij}$$

Obs: as restrições não controlam *loops*.

O problema da árvore geradora mínima pode ser resolvido otimamente utilizando-se procedimentos gulosos no sentido de sempre escolher, sucessivamente, as arestas de menor custo.

1) Algoritmo de Prim

Um algoritmo famoso para o problema para determinar a solução ótima do problema do MST é o algoritmo de Prim. No algoritmo de Prim, iniciamos com uma árvore formada por um único nó (qualquer nó do grafo) e vamos adicionando à árvore, a cada passo, o nó que estiver mais próximo (ou aquele ligado através da aresta de menor custo). O algoritmo foi proposto em 1930 pelo matemático Vojtěch Jarník.

Podemos resumir o algoritmo de Prim assim:

- Suponha que T é uma subárvore (não necessariamente geradora) de um grafo não orientado conexo G com custos nas arestas. A *franja* (≡*fringe*) de T é o conjunto de todas as arestas de G que têm uma ponta em T e outra fora. Portanto, a franja de T é o leque[1] do conjunto de nós de T.
- Cada iteração do algoritmo começa com uma subárvore T. No início da primeira iteração, T consiste em um único nó. O processo iterativo pode ser descrito assim. Enquanto a franja de T não estiver vazia:
    1. escolha uma aresta da franja que tenha custo mínimo;
    2. seja e a aresta escolhida;
    3. acrescente e a T.[2]

O algoritmo de Prim tem caráter guloso: em cada iteração abocanha a aresta mais barata da franja sem se preocupar com o efeito global dessa escolha.

[1] O *leque* de um conjunto X de nós de um grafo não orientado é o conjunto de todas as arestas que têm uma ponta em X e outra no complemento de X.

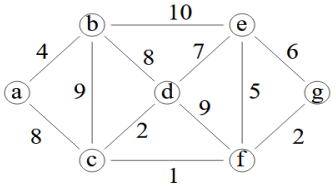
[2] Enquanto a árvore não contém todos os nós no grafo, encontre a aresta mais barata que sai da árvore e adicione-a à árvore.

O algoritmo de Prim, sempre mantem uma árvore conectada começando com um único nó. Analisa-se todas as arestas do nó atual para outros e encontra o de menor custo entre eles. Em seguida, adiciona-se o nó de menor custo à árvore, aumentando seu tamanho em 1. Em n–1 etapas, todos os nós estarão conectados numa árvore.

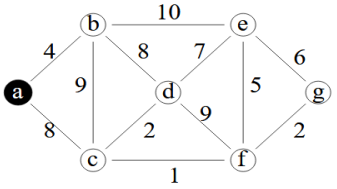
Exemplo

Prim's Algorithm

Worked Example

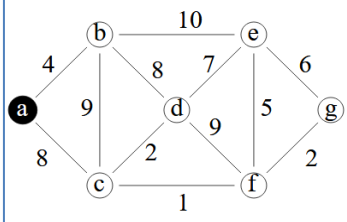


Connected graph

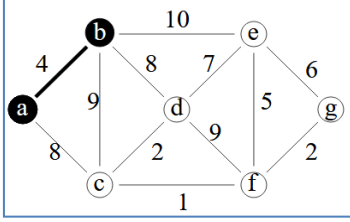


Step 0
S={a}
V \ S = {b,c,d,e,f,g}
lightest edge = {a,b}

Prim's Example – Continued

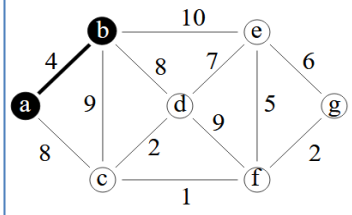


Step 1.1 before
S={a}
V \ S = {b,c,d,e,f,g}
A={}
lightest edge = {a,b}

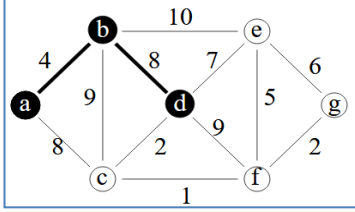


Step 1.1 after
S={a,b}
V \ S = {c,d,e,f,g}
A={{a,b}}
lightest edge = {b,d}, {a,c}

Prim's Example – Continued

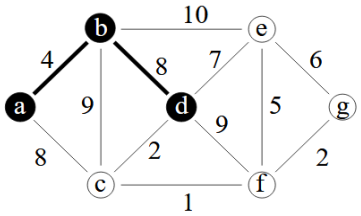


Step 1.2 before
S={a,b}
V \ S = {c,d,e,f,g}
A={{a,b}}
lightest edge = {b,d}, {a,c}

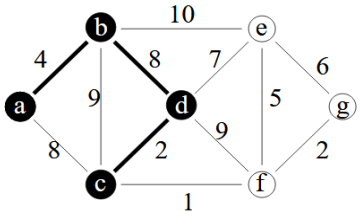


Step 1.2 after
S={a,b,d}
V \ S = {c,e,f,g}
A={{a,b},{b,d}}
lightest edge = {d,c}

Prim's Example – Continued

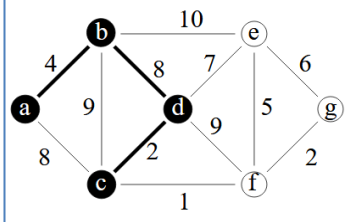


Step 1.3 before
S={a,b,d}
V \ S = {c,e,f,g}
A={{a,b},{b,d}}
lightest edge = {d,c}

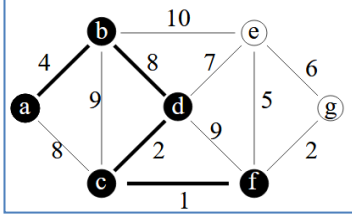


Step 1.3 after
S={a,b,c,d}
V \ S = {e,f,g}
A={{a,b},{b,d},{c,d}}
lightest edge = {c,f}

Prim's Example – Continued

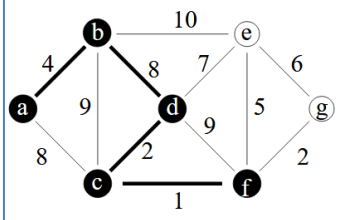


Step 1.4 before
S={a,b,c,d}
V \ S = {e,f,g}
A={{a,b},{b,d},{c,d}}
lightest edge = {c,f}

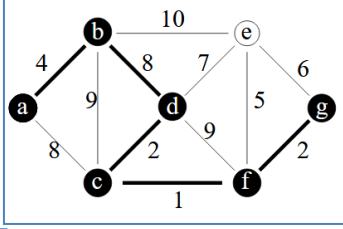


Step 1.4 after
S={a,b,c,d,f}
V \ S = {e,g}
A={{a,b},{b,d},{c,d},{c,f}}
lightest edge = {f,g}

Prim's Example – Continued

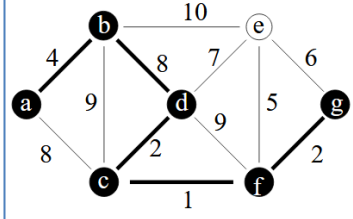


Step 1.5 before
S={a,b,c,d,f}
V \ S = {e,g}
A={{a,b},{b,d},{c,d},{c,f}}
lightest edge = {f,g}

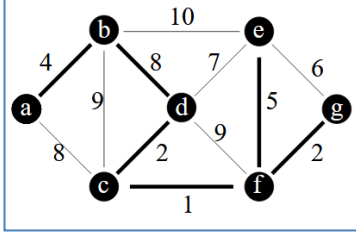


Step 1.5 after
S={a,b,c,d,f,g}
V \ S = {e}
A={{a,b},{b,d},{c,d},{c,f},
{f,g}}
lightest edge = {f,e}

Prim's Example – Continued



Step 1.6 before
S={a,b,c,d,f,g}
V \ S = {e}
A={{a,b},{b,d},{c,d},{c,f},
{f,g}}
lightest edge = {f,e}



Step 1.6 after
S={a,b,c,d,e,f,g}
V \ S = {}
A={{a,b},{b,d},{c,d},{c,f},
{f,g},{f,e}}
MST completed

2) Algoritmo de Kruskal

O algoritmo de Kruskal também determina a solução ótima para o problema MST. O algoritmo começa com uma floresta de árvores formadas por apenas um único nó cada, e procura, a cada passo, uma aresta que, além de conectar duas árvores distintas da floresta, possua custo mínimo. Suponha que, numa determinada iteração, o algoritmo escolheu a aresta (u,v) para ser inserida no conjunto A e que a aresta (u,v) conecte a árvore C_1 à árvore C_2. Note que, dentre todas as arestas que conectam duas componentes distintas nesta iteração, (u,v) é uma das que possui o menor custo, pois ela foi escolhida assim. O algoritmo foi desenvolvido em 1956 por Joseph Kruskal.

O Algoritmo de Kruskal pode assim descrito:

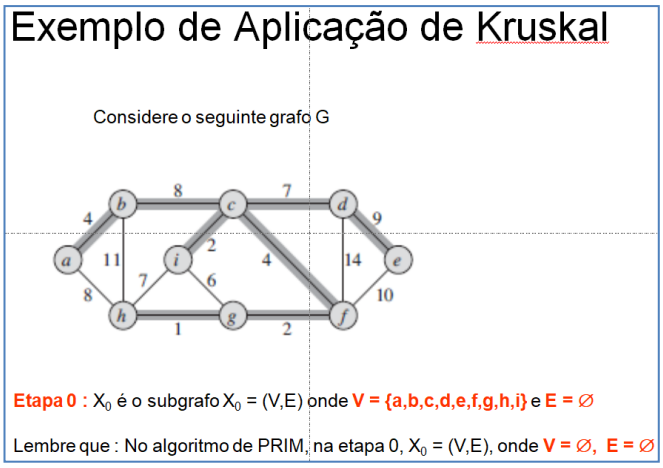
- Considerando cada nó como uma árvore independente, o algoritmo procura a aresta de menor custo que conecta duas árvores. Os nós das árvores selecionadas passam a fazer parte de uma mesma árvore.
- O processo se repete até que todos os nós façam parte de uma mesma árvore ou quando não se pode encontrar uma aresta que satisfaça essa condição.

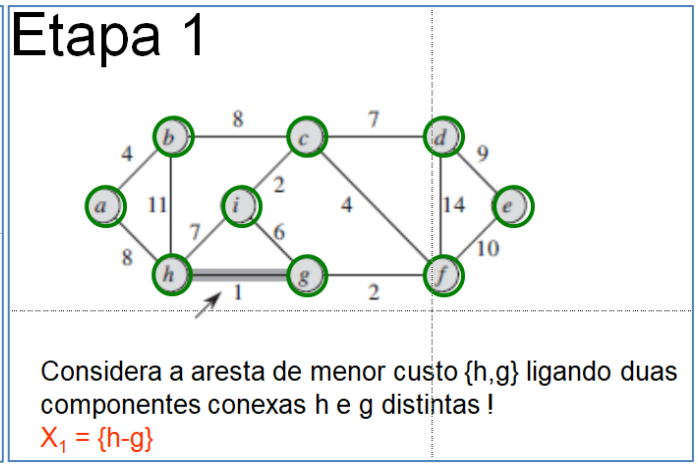
No algoritmo de Kruskal, não é mantida uma única árvore, mas uma floresta. Em cada etapa, analisa-se a aresta de menor custo e que não gera um ciclo na floresta atual. Esta aresta tem que unir necessariamente duas árvores na floresta atual em uma. Uma vez que inicia com n árvores de um único nó, em n–1 passos todos eles estarão conectados se o grafo for conexo.

No caso do Prim, ao invés de conectar árvores de uma floresta, o algoritmo mantém uma única árvore que cresce a cada iteração com a inserção de um nó.

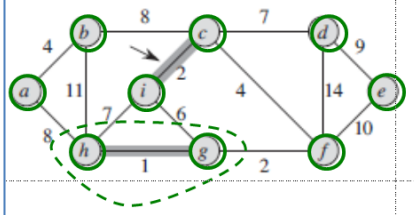
O algoritmo de Prim é significativamente mais rápido quando o grafo é denso com muito mais arestas do que nós. Kruskal funciona melhor em situações típicas (grafos esparsos), pois usa estruturas de dados mais simples.

Exemplo





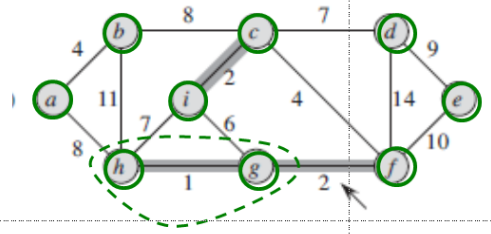
Etapa 2



X1 = {h-g}

Considera a 2a aresta de menor custo {i,c} – ligando duas componentes conexas distintas i e c
X2 = {h-g,i-c}

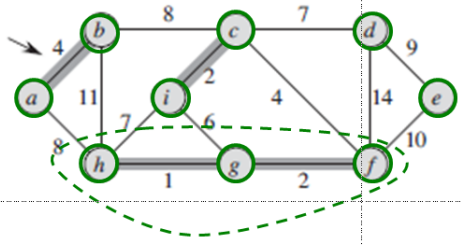
Etapa 3



X2 = {h-g,i-c}

Considera a 3a aresta de menor custo {g,f} – ligando duas componentes conexas distintas {h-g} e {f}
X3 = {h-g,i-c,g-f}

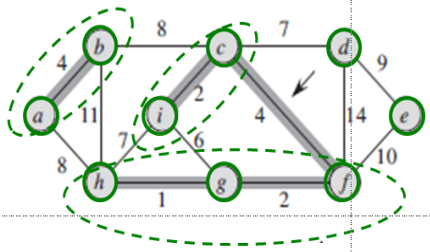
Etapa 4



X3 = {h-g,i-c,g-f}

Considera a 4a aresta de menor custo {a,b}, ligando as componentes conexas distintas {a} e {b}
X4 = {h-g,i-c,g-f,a-b}

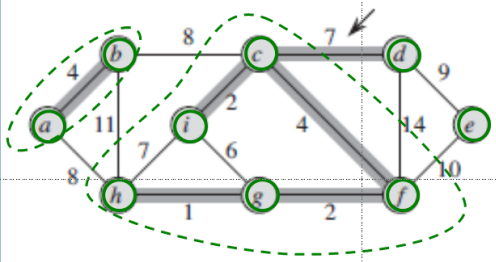
Etapa 5



X4 = {h-g,i-c,g-f,a-b}

Considera a 5a aresta de menor custo {c,f}, ligando duas componentes conexas distintas
X5 = {h-g,i-c,g-f,a-b,c-f}

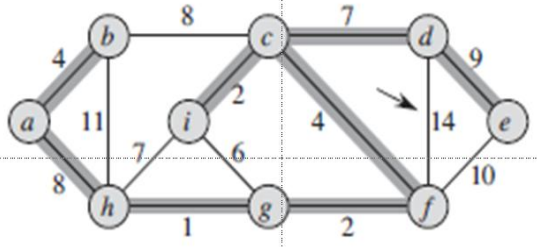
Etapa 6



X5 = {h-g,i-c,g-f,a-b,c-f}
Considera a 6a aresta de menor custo {c,d} ligando duas componentes conexas distintas.

Repare que a aresta **i-g** liga vértices da mesma componente conexa de X6, logo **não pode ser considerada**, embora seja de menor custo !

Resultado final



Exercício: continue as demais etapas a partir da sétima até chegar neste resultado final.

Outros algoritmos MST

Baruvka's Algorithm: https://en.wikipedia.org/wiki/Bor%C5%AFvka%27s_algorithm

Atualmente, o algoritmo MST mais rápido é um algoritmo aleatório proposto por Karger, Klein e Tarjan em 1993. http://cs.brown.edu/research/pubs/pdfs/1995/Karger-1995-RLT.pdf