Algoritmos Gulosos

Ricardo Dutra da Silva

Universidade Tecnológica Federal do Paraná

Árvore Geradora Mínima

Entrada

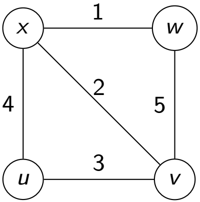
Uma grafo $G = (V, E)$ com custo c_e para cada aresta $e \in E$.

Saída

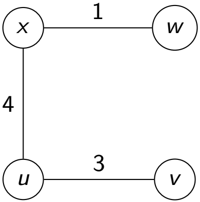
Árvore Geradora $T = (V, E')$, $E' \subseteq E$, tal que o custo da árvore $\sum_{e \in E'} c_e$ é mínimo.

Árvore Geradora Mínima

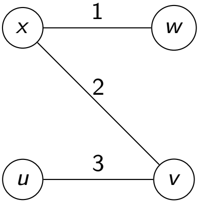
G



Árvore geradora T



Árvore geradora mínima T



Árvore Geradora Mínima

- O algoritmo de Prim começa com um corte $C = \{v\}$, com um vértice arbitrário $v \in V$.
- A escolha gulosa é uma aresta do corte com menor peso, $(u, v) \in E$, com $u \in C$ e $v \in V \setminus C$.
- O vértice v é incluído no corte e temos um novo subproblema com um vértice a menos.

Árvore Geradora Mínima

Algoritmo: $\textsc{Prim}(G = (V, E))$

`/* Inicia árvore. */`

$T = (V, E' \leftarrow \emptyset)$

`/* Escolhe vértice arbitrário. */`

$v \leftarrow \textsc{EscolheVertice}(V)$

$C \leftarrow \{v\}$

enquanto $C \neq V$ **faça**

    $(u, v) \leftarrow \textsc{ArestaMinima}(C, V \setminus C)$

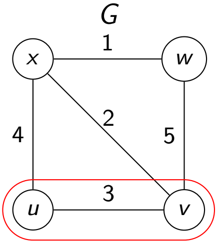
    $E' \leftarrow E' \cup \{(u, v)\}$

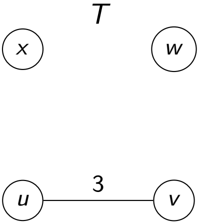
    $C \leftarrow C \cup \{v\}$

Árvore Geradora Mínima

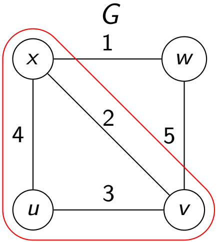
- Inicialização da árvore.
- Corte destacado em vermelho.

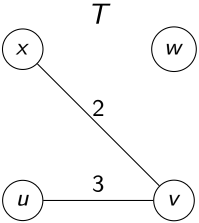
Árvore Geradora Mínima



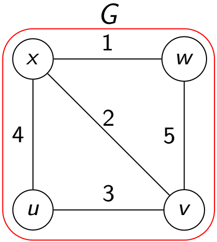


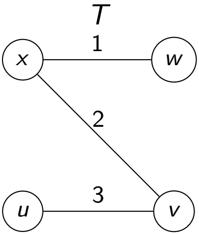
Árvore Geradora Mínima





Árvore Geradora Mínima





Teorema

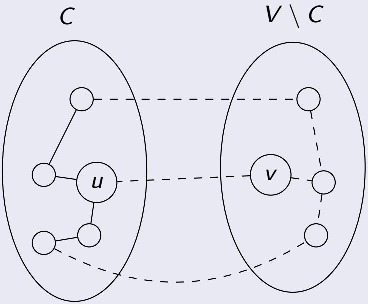
O algoritmo de Prim computa uma árvore geradora.

Demonstração.

Assumimos G conexo.
Temos um invariante que T é uma árvore geradora de C.
Vale antes da primeira iteração. $\square$

Demonstração.

Em qualquer iteração temos um corte.



□

Demonstração.

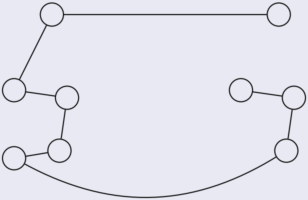
Ao adicionar o vértice v em C juntamente com a aresta (u, v), continuamos com um grafo conectado, que gera C.
(u, v) não gera ciclo em C. Para gerar um ciclo (u, v) deveria conectar dois vértices em C, mas $v \notin C$.
Temos uma árvore após a iteração.
No final, todos os vértices pertencem a C e o invariante é válido. $\square$

Teorema

A aresta de menor custo $e \in E$ pertence à AGM.

Demonstração.

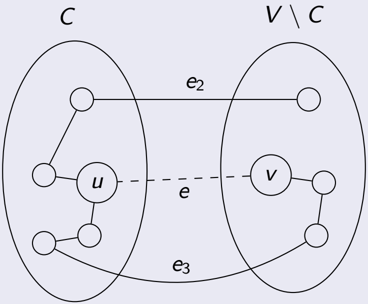
Suponha uma AGM sem a escolha gulosa.



$\square$

Demonstração.

Consideramos um corte em que e é uma aresta de menor custo.

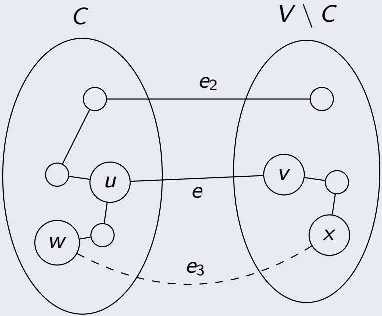


□

Árvore Geradora Mínima

Demonstração.

Se incluírmos e na árvore, geramos um ciclo. Podemos substituir e pela outra aresta no corte que forma o ciclo. Temos uma nova árvore T'.



□

Árvore Geradora Mínima

Demonstração.

Temos então que

$$\sum_{e \in T} - \sum_{e \in T'} = c_{w,x} - c_{u,v} \geq 0$$

Portanto, T' é AGM. $\square$

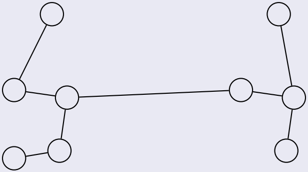
Árvore Geradora Mínima

Teorema

Se T é uma AGM para V, então ao remover uma aresta qualquer de T temos que as subárvores resultantes também são ótimas.

Demonstração.

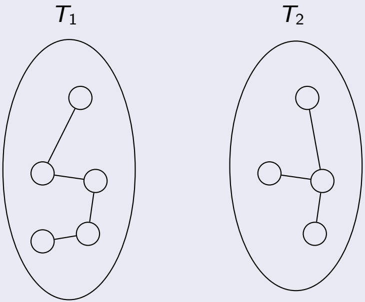
Consideramos uma AGM qualquer.



□

Demonstração.

Removendo uma aresta e qualquer temos duas subárvores T_1 e T_2.



$\square$

Demonstração.

Vamos supor que T_1 não seja ótima, portanto há T_1' tal que $w(T_1') < w(T_1)$.
Podemos escrever o custo de T como abaixo e substituir a nova árvore no lugar de T_1

$$\begin{aligned} w(T) &= w(T_1) + w(T_2) + w(e) \\ &\geq w(T_1') + w(T_2) + w(e) \\ &= w(T') \end{aligned}$$

obtendo uma árvore com custo menor que T. Portanto, por contradição, T_1 é ótima.
De forma análoga, provamos T_2 ótima.

□

Árvore Geradora Mínima

- O tempo do do algoritmo de Prim depende do custo para encontrar a aresta de menor custo.
- Usando um heap podemos fazer a operação em tempo $\mathcal{O}(n \lg n)$.

Árvore Geradora Mínima

Algoritmo: $\textsc{Prim}(G = (V, E))$

```
/* Heap auxiliar H.                                            */
```

para $v \in V$ **faça**
    $v.c \leftarrow \infty$
    $v.p \leftarrow$ nulo
    $\textsc{Insere}(H,v)$
$s \leftarrow \textsc{EscolheVertice}(V)$
$s.c \leftarrow 0$
$\textsc{Atualiza}(H,s)$
enquanto $H \neq \emptyset$ **faça**
    $u \leftarrow \textsc{Remove}(H)$
    para $e = (u, v) \in E$ **faça**
        se $v.c > c_e$ e $v \in H$ **então**
            $v.c \leftarrow c_e$
            $v.p \leftarrow u$
            $\textsc{Atualiza}(H,v)$

Análise da complexidade do algoritmo:

- linhas 1 a 4 $\mathcal{O}(n \lg n)$;
- linhas 5 a 6 $\mathcal{O}(1)$;
- linhas 7 $\mathcal{O}(\lg n)$;
- linhas 8 $\mathcal{O}(n)$;
- linhas 9 $\mathcal{O}(n \lg n)$;
- linhas 10 $\mathcal{O}(m)$;
- linhas 11 a 13 $\mathcal{O}(m)$;
- linhas 14 $\mathcal{O}(m \lg n)$.

Portanto, o algoritmo tem complexidade $\mathcal{O}(n \lg n + m \lg n)$ ou $\mathcal{O}((n+m) \lg n)$. Se o grafo é conexo, $\mathcal{O}(m \lg n)$.