

Florestas geradoras maximais de custo mínimo em grafos dinâmicos

Chung Jin Shian

Orientadora: Cristina Gomes Fernandes

Departamento de Ciência da Computação, Instituto de Matemática, Estatística e Ciência da Computação,
Universidade de São Paulo

Resumo

Grafos dinâmicos permitem modelar problemas em que o grafo sofre alterações ao longo do tempo. Um dos problemas fundamentais nesse contexto é a manutenção de uma árvore geradora de custo mínimo de um grafo dinâmico. Estudamos vários algoritmos propostos por Holm, de Lichtenberg e Thorup [1] para variantes desse problema. O foco foi no algoritmo para manter uma floresta maximal de custo mínimo (MSF) no contexto decremental, em que se dá suporte eficiente à remoção de arestas. Esse algoritmo foi implementado e testado em grafos dinâmicos com dezenas de milhares de vértices.

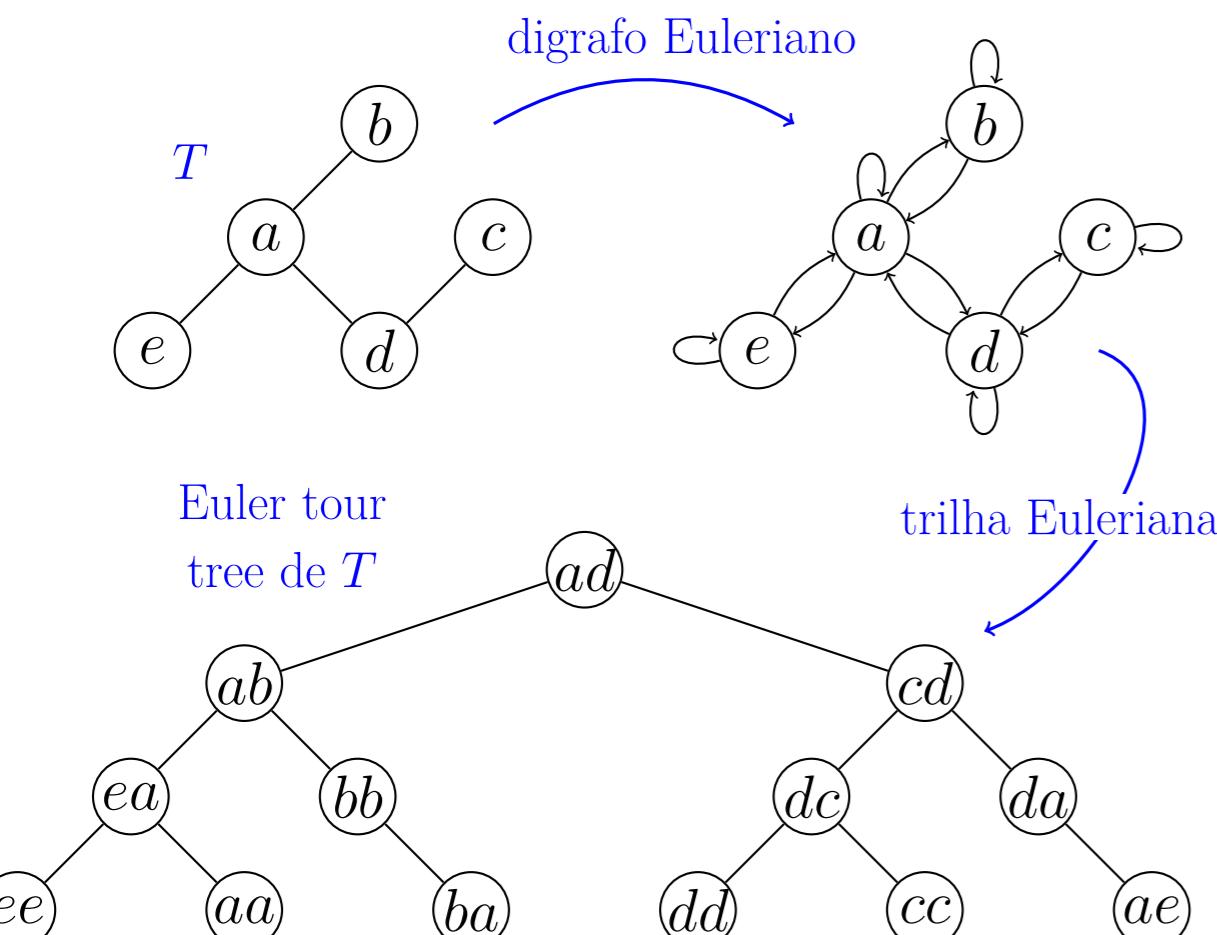
Conexidade em grafos dinâmicos

O problema da conexidade em grafos dinâmicos visa uma implementação eficiente da biblioteca abaixo:

- **grafoDinâmico(n):** constrói e devolve um grafo dinâmico com n vértices e sem arestas;
- **conectadosGD(G, u, v):** devolve verdadeiro se os vértices u e v estão na mesma componente de G e falso caso contrário;
- **adiconeGD(G, u, v):** adiciona a aresta uv no grafo G ;
- **removeGD(G, u, v):** remove a aresta uv do grafo G .

Ideia: Fatiar o grafo G em níveis. Cada aresta de G possui um nível entre 1 e $\lceil \lg n \rceil$, onde n é o número de vértices de G . Uma aresta, ao ser inserida em G , começa com o nível $\lceil \lg n \rceil$ e, durante o algoritmo, seu nível vai sendo decrementado. Seja G_i o subgrafo de G com as arestas de G de nível menor ou igual a i . Para cada nível i , o algoritmo mantém uma floresta maximal F_i de G_i . Além disso, ele mantém também o subgrafo R_i de G com as arestas de nível i que não estão em F_i , chamadas de arestas reserva.

Cada grafo R_i é mantido por suas listas de adjacências. Já cada floresta F_i é mantida em uma estrutura de dados específica para florestas dinâmicas, baseada em Euler tour trees. Cada componente de F_i é armazenada como uma Euler tour tree. Em nossa implementação, Euler tour trees são implementadas como splay trees.



O algoritmo mantém as seguintes invariantes:

- F_i é floresta maximal de G_i para $1 \leq i \leq \lceil \lg n \rceil$;
- $F_i \subseteq F_{i+1}$ para $1 \leq i \leq \lceil \lg n \rceil - 1$;
- Cada componente de F_i tem no máximo 2^i vértices.

Nos pseudocódigos abaixo, esboçamos a remoção de uma aresta do grafo. As Euler tour trees carregam informação extra para que a implementação das linhas 6, 7, 10 e 11 do método **substituaAresta** seja eficiente.

removeGD(G, u, v)

Entrada: Recebe dois vértices u e v adjacentes do grafo G .

Efeito: Remove a aresta uv do grafo G .

```
1   i ← G.nível[u, v]
2   nível[u, v] ← NIL
3   L ← G.nívelMax
4   se uv ∈ G.Fi então
5       para j ← i até L faça
6           removeFD(G.Fj, u, v)
7       substituaAresta(G, i, u, v)
8   senão
9       removeLA(G.Ri, u, v)
```

substituaAresta(G, i, u, v)

Entrada: Recebe dois vértices u e v de componentes distintas do grafo G , e um nível i .

Efeito: Adiciona à floresta, se existir, uma aresta de G substituta para xy , de nível $\geq i$.

```
1   L ← G.nívelMax
2   para j ← i até L faça
3       Tu ← splay(G.Fj, nó[u, u])
4       Tv ← splay(G.Fj, nó[v, v])
5       se Tu.tam > Tv.tam então Tu ↔ Tv
6       enquanto Tu.arestasDeNível > 0 faça
7           nóXY ← procureArestaDeNível(Tu)
8           Tu ← splay(nóXY)
9           rebaixaNívelDaAresta(G, nóXY, j)
10      enquanto Tu.arestasReservasDeNível > 0 faça
11          nóXX ← procureNóIncideArestaDePesoMinimo(Rj, Tu)
12          Tu ← splay(nóXX)
13          (x, x) ← nóXX.vértices
14          para y ∈ G.Rj[x] faça
15              se é substituta, inclui e termina, senão rebaixa
                  se testeSubstituta(G, x, y, j) então retorna
```

testeSubstituta(G, x, y, j)

Entrada: Recebe dois vértices adjacentes x e y do grafo G e um nível j .

Saída: Devolve verdadeiro se a aresta xy é substituta e falso caso contrário.

```
1   removeLA(G.Rj, x, y)
2   se conectadosGD(G, x, y) então
3       G.nível[x, y] ← j - 1
4       adiconeFD(G.Rj-1, x, y)
5       retorna falso
6   senão
7       L ← G.nívelMax
8       para k ← j até L faça
9           adiconeFD(G.Fk, x, y)
10      retorna verdadeiro
```

Na nossa implementação, que utiliza splay trees, alguns dos métodos têm consumo amortizado por operação.

- **grafoDinâmico(n):** $O(n \lg n)$;
- **conectadosGD(G, u, v):** amortizado $O(\lg n)$;
- **adiconeGD(G, u, v):** amortizado $O(\lg n)$;
- **removeGD(G, u, v):** amortizado $O(\lg^2 n)$.

MSF decremental

O problema da manutenção de uma MSF decremental do grafo visa uma implementação eficiente dos métodos da biblioteca abaixo:

- **MSFDecremental(n, E):** constrói e devolve o grafo ponderado G com n vértices e as arestas ponderadas dadas no conjunto E ;
- **consultePesoMSF(G):** devolve o peso de uma MSF do grafo ponderado G ;
- **removeMSF(G, u, v):** remove a aresta uv do grafo ponderado G .

Ideia: No loop das linhas 10 a 15 do método **substituaAresta**, busca-se por uma aresta substituta numa ordem arbitrária. Na implementação da MSF decremental, a ideia é olhar as arestas candidatas à substituta em ordem crescente de peso. Para isso, as listas de adjacências de R_i são substituídas por min-heaps, onde a chave para um vizinho v é o peso da aresta uv . Assim, nas linhas 11 e 14 do **substituaArestaMSF**, basta procurar por uma aresta em R_i que incida em T_u , que tenha o menor peso e que conecte T_u e T_v , para podemos manter o peso mínimo de uma MSF do grafo ao longo das remoções de arestas.

Na figura abaixo, veja que, nas linhas 11 e 14, passamos a procurar uma aresta de peso mínimo em R_i incidente a T_u . As linhas 1 a 10 são idênticas às do método **substituaAresta**.

substituaArestaMSF(G, i, u, v)

Entrada: Recebe dois vértices u e v de componentes distintas do grafo G , e um nível i .

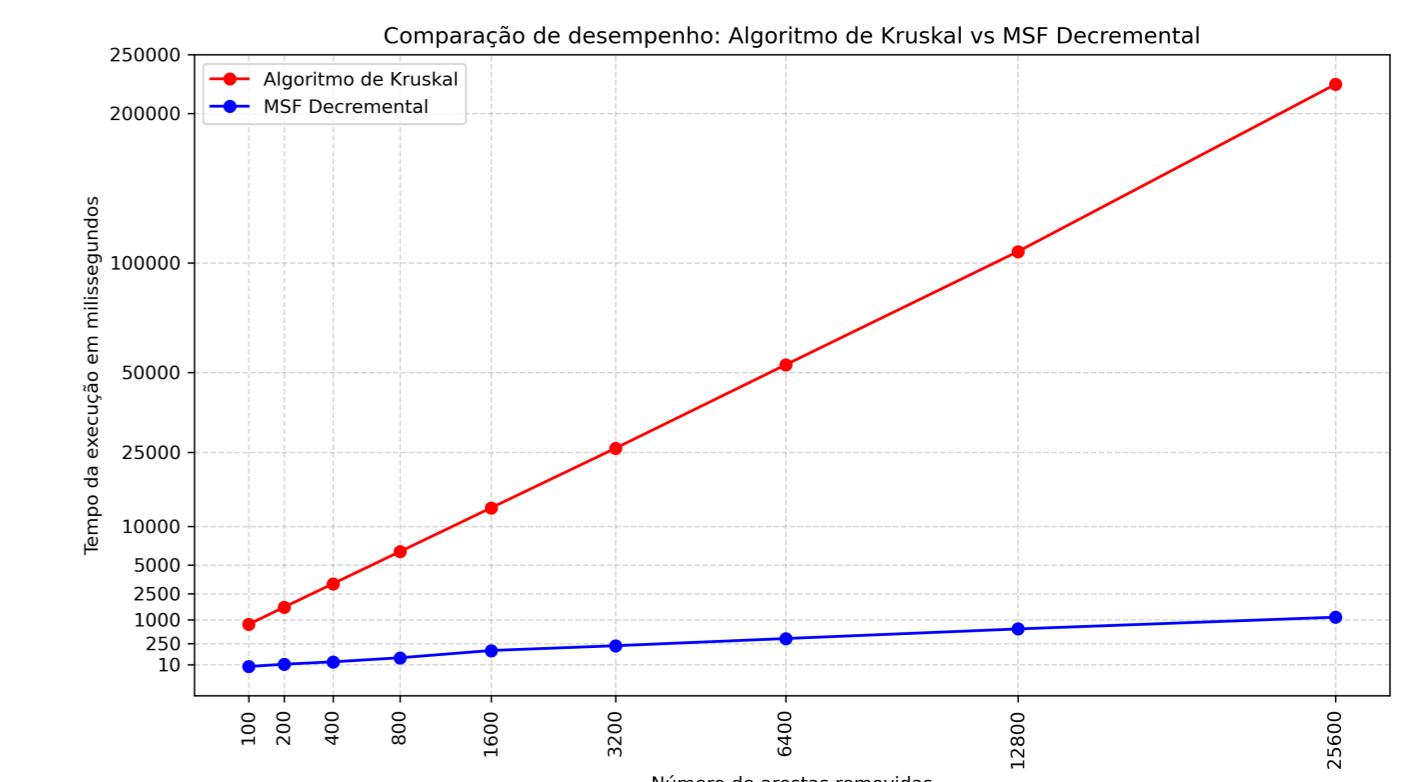
Efeito: Adiciona à floresta, se existir, uma aresta de G substituta para xy , de nível $\geq i$.

```
1   L ← G.nívelMax
2   para j ← i até L faça
3       Tu ← splay(G.Fj, nó[u, u])
4       Tv ← splay(G.Fj, nó[v, v])
5       se Tu.tam > Tv.tam então Tu ↔ Tv
6       enquanto Tu.arestasDeNível > 0 faça
7           nóXY ← procureArestaDeNível(Tu)
8           Tu ← splay(nóXY)
9           rebaixaNívelDaAresta(G, nóXY, j)
10      enquanto Tu.arestasReservasDeNível > 0 faça
11          nóXX ← procureNóIncideArestaDePesoMinimo(Rj, Tu)
12          Tu ← splay(nóXX)
13          (x, x) ← nóXX.vértices
14          (y, w) := consulteMinLMSF(Rj, x)           ▷ vizinho de x de aresta de peso
                                                       mínimo em Rj
15          se testeSubstitutaMSF(G, x, y, j) então retorna
```

Abaixo está o consumo de tempo da implementação:

- **MSFDecremental(n, E):** $O(|E| \lg n)$;
- **consultePesoMSF(G):** $O(1)$;
- **removeMSF(G, u, v):** amortizado $O(\lg^2 n)$.

Testamos a remoção das arestas de um grafo G com 20.000 vértices e mais de 140.000 arestas, realizando uma comparação da performance do algoritmo de Kruskal e da MSF decremental que implementamos.



Arestas removidas	peso da MST	Algoritmo de Kruskal (ms)	MSF Decremental (ms)
100	33759198	813	5
200	33799435	1634	12
400	33846203	3268	23
800	33966269	6527	51
1600	34146709	13057	133
3200	34535747	26085	211
6400	35314449	52854	366
12800	37131685	106333	649
25600	40782042	223986	1118

Comentários finais

No texto do TCC, além da descrição detalhada dos algoritmos apresentados aqui bem como de suas análises de consumo de tempo, também descrevemos brevemente a ideia por trás de um algoritmo para o problema da MSF totalmente dinâmico, que dá suporte eficiente não só à remoção de arestas, como também à inserção de arestas. Esse algoritmo faz uso do algoritmo da MSF decremental apresentado acima.

Informações e contato

Para mais informações, acesse a página do trabalho: <https://linux.ime.usp.br/~cjinhian/>
Endereço para contato: cjinhian77@usp.br

Referências

- [1] Holm, J., de Lichtenberg, K., Thorup, M., "Polynomial Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity," *Journal of the ACM*, 48(4): 723–760, 2001.