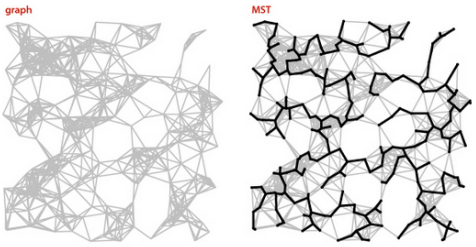
PRIM

Algoritmo de Kruskal para o problema MST

Esta página trata do problema das árvores geradoras de peso mínimo em grafos. Trata-se de uma generalização do problema da árvore geradora discutido no capítulo *Árvores geradoras de grafos*. Esse problema é mais conhecido pelas iniciais MST de *minimum spanning tree*. A página trata, em particular, do célebre algoritmo de Kruskal para o problema da MST.

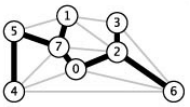
Esta página é inspirada no capítulo 23 do livro CLRS.



Árvores geradoras de peso mínimo

Suponha que cada aresta uv de um grafo tem um *peso* $p[uv]$. Cada peso é um número positivo, negativo, ou nulo. Vamos supor que os pesos são inteiros (embora isso não seja essencial). Diremos que p é um *vetor de pesos*. O *peso* de um subgrafo T do grafo é a soma dos pesos das arestas de T e será denotado por $p(T)$.

EXEMPLO A. A tabela abaixo atribui um peso $p[a]$ a cada aresta a do grafo da figura. (Neste exemplo, o peso de cada aresta é proporcional ao comprimento geométrico do segmento de reta que representa a aresta na figura.)



a	45	47	57	07	15	04	23	17	02	12	13	27	62	36	60	64
$p[a]$	35	37	28	16	32	38	17	19	26	36	29	34	40	52	58	93

O conjunto de arestas 07 71 75 54 02 23 26 define uma árvore geradora do grafo. O peso dessa árvore é 35 + 28 + 16 + 17 + 19 + 26 + 40 = 181. Nenhuma outra árvore geradora tem peso menor (mas isso não é óbvio).

Uma *MST* (= *minimum spanning tree*) de um grafo com vetor de pesos p é uma árvore geradora T do grafo que minimiza $p(T)$.

> PROBLEMA DA MST: Dado um grafo G com pesos inteiros arbitrários nas arestas, encontrar uma MST de G.

É claro que uma instância do problema tem solução se e somente se G é conexo, ou seja, se cada vértice de G estiver ao alcance de cada um dos demais.

Exercícios 1

1. Mostre que uma instância do problema da MST tem solução se e somente se G é conexo.
2. Mostre que o seguinte algoritmo guloso não resolve o problema da MST. Seja $a_1, a_2, \ldots, a_m$ a sequência das arestas em ordem crescente de pesos. No início da primeira iteração, pinte um vértice de preto e os demais vértices de branco. Para $i = 1, 2, \ldots, m$ faça o seguinte: se uma das pontas de a_i é preta e a outra ponta é branca, acrescente a_i à árvore e pinte de preto a ponta que era branca.
3. Discuta a seguinte afirmação: Para resolver o problema da MST num grafo conexo com n vértices, basta escolher as $n-1$ arestas de menor peso.

O algoritmo de Kruskal

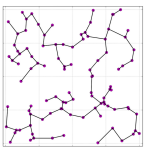
J. Kruskal descobriu (1956) um algoritmo muito eficiente para o problema da MST.
O algoritmo de Kruskal é iterativo. Cada iteração começa com uma floresta geradora F de G. A cada iteração, o algoritmo acrescenta a F uma aresta de G. O critério de escolha da nova aresta é guloso: o algoritmo escolhe uma aresta *de peso mínimo* dentre as que são externas à floresta. Uma aresta a é considerada *externa* a F se satisfaz duas condições naturais e óbvias: (1) não pertence a F e (2) o grafo $F + a$ é uma floresta.
O algoritmo pode, então, ser resumido assim:

```
enquanto existem arestas externas,
    escolha uma aresta externa de peso mínimo,
    seja uv a aresta escolhida,
    troque F por F + uv.
```

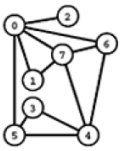
No início da primeira iteração, a floresta F não tem aresta alguma (e o conjunto de vértices de F é $V(G)$). No início da última iteração, G não tem arestas externas a F.

No fim da execução do algoritmo, a floresta F será conexa desde que o grafo G seja conexo. Sob essa hipótese, F será uma MST, como mostraremos mais abaixo.

> EXEMPLO B. O vídeo *Kruskal's Algorithm Animation* no YouTube aplica o algoritmo de Kruskal a um grafo cujos vértices são pontos aleatórios no plano. O grafo é completo, ou seja, todo par de vértices é uma aresta. O peso de cada aresta é a distância geométrica entre suas pontas.

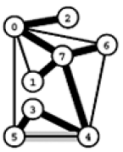


EXEMPLO C. Considere o problema de encontrar uma MST no grafo com pesos nas arestas definido a seguir. As arestas são apresentadas em ordem crescente de peso.



```
53 71 76 20 07 01 43 54 74 06 46 05
 0  1  2  3  4  5  6  7  8  9  9 11
```

Veja o rastreamento da execução do algoritmo de Kruskal. Cada linha da tabela registra as arestas da floresta F e o peso $p(F)$ da floresta no início de uma iteração.



```
-                       0
53                      0
53 71                   1
53 71 76                3
53 71 76 20             6
53 71 76 20 07         10
53 71 76 20 07 43      16
53 71 76 20 07 43 74   24
```

A última linha dá as arestas de uma MST.

Prova da correção do algoritmo. Vamos supor que o grafo G é conexo. Para mostrar que o algoritmo de Kruskal está correto, basta provar o seguinte invariante do processo iterativo: no início de cada iteração,

F é subgrafo de alguma MST de G. (*)

Essa proposição é obviamente verdadeira no início da primeira iteração, quando F não tem aresta alguma. Suponha agora que estamos no início de alguma iteração (exceto a última) e que F faz parte de uma MST M. Seja uv a aresta que o algoritmo escolhe nessa iteração. Precisamos provar que a floresta $F + uv$ faz parte alguma MST.

Como uv é externa a F, uma das pontas de uv está em uma componente de F e a outra ponta está em outra componente. Seja U o conjunto de vértices de uma dessas componentes.

Agora considere a relação entre uv e M. Se uv é aresta de M, nada mais temos a provar. Suponha agora que uv não é aresta de M e seja C o único caminho de u a v em M. Alguma aresta xy de C tem uma ponta em U e outra ponta em $\overline{U}$. Segue daí que xy não pertence a F e $F + xy$ é uma floresta, donde xy é externa a F. Como xy não foi escolhida durante a iteração corrente, temos $p(xy) \geq p(uv)$. Observe agora que

$$M - xy + uv$$

é uma árvore geradora. O peso dessa nova árvore não é maior que o peso de M. Logo, a nova árvore geradora é uma MST. Essa nova MST contém $F + uv$. Era o que queríamos provar.

Exercícios 2

1. Diga *o quê* o algoritmo de Kruskal faz.
2. Qual a diferença entre provar que um algoritmo está correto e descrever os passos do

algoritmo em português ("o algoritmo faz isso, depois faz aquilo ...", etc.)?

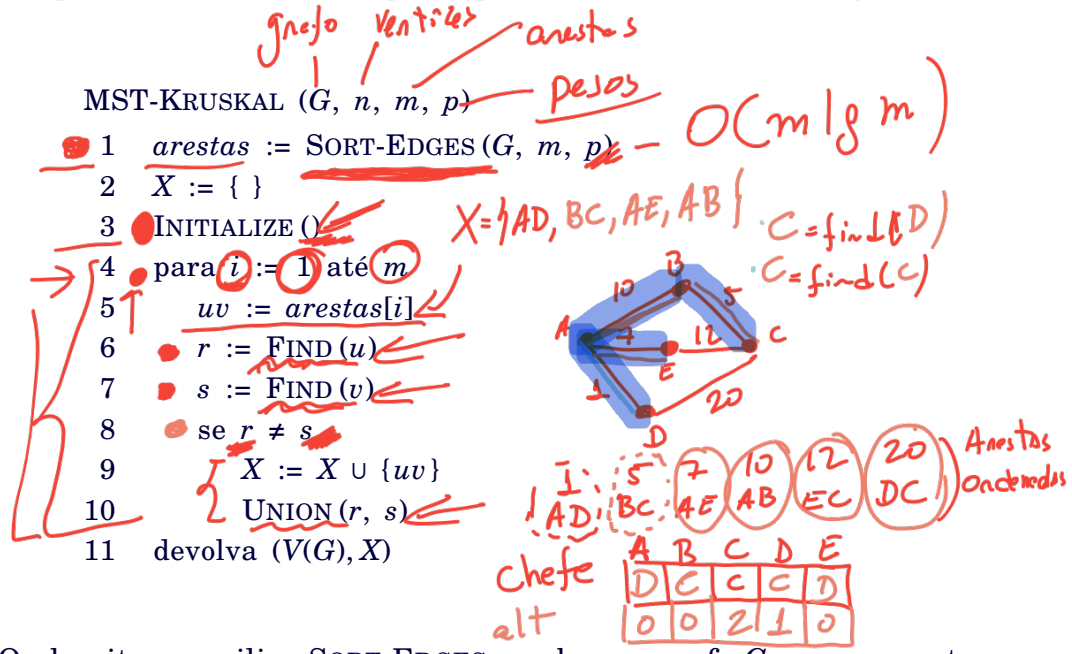
3. Prove que uma aresta é externa a F se e somente se tem uma ponta em uma componente de F e a outra ponta em outra componente.
4. É verdade que uma MST contém todas as arestas de peso mínimo de G?
5. ★ Preencha os detalhes do esboço da prova de (*). Prove os passos da prova que não forem óbvios.
6. *Grafo desconexo.* Que acontece se o algoritmo de Kruskal for aplicado a um grafo desconexo?

Implementação do algoritmo de Kruskal

Para implementar o algoritmo de Kruskal, é preciso inventar uma maneira eficiente de decidir se uma dada aresta de G é externa a F. É claro que uma aresta é externa a F se e somente se tem uma ponta em uma componente de F e a outra ponta em outra componente. Isso sugere que o tipo abstrato de dados union-find (veja o capítulo *A estrutura union-find*) pode ser útil.

O algoritmo MST-KRUSKAL abaixo recebe um grafo conexo G com n vértices, m arestas, e um peso inteiro arbitrário $p[uv]$ para cada aresta uv. O algoritmo devolve uma MST de G.

```
MST-KRUSKAL (G, n, m, p)
1   arestas := SORT-EDGES (G, m, p)
2   X := { }
3   INITIALIZE ()
4   para i := 1 até m
5      uv := arestas[i]
6      r := FIND (u)
7      s := FIND (v)
8      se r ≠ s
9         X := X ∪ {uv}
10        UNION (r, s)
11  devolva (V(G), X)
```



O algoritmo auxiliar SORT-EDGES recebe um grafo G com m arestas e um peso $p[uv]$ para cada aresta uv e devolve um vetor $arestas[1..m]$ que contém todas as arestas em ordem crescente de pesos.

No início de cada iteração, X é o conjunto de arestas de uma floresta geradora, que chamaremos F. Os algoritmos auxiliares FIND e UNION atuam sobre as componentes de F:

- FIND recebe um vértice e devolve o "diretor" da componente de F a que esse vértice pertence;
- UNION recebe os "diretores" r e s de duas componente de F, faz a fusão das duas componentes, e designa r ou s como "diretor" da nova componente;
- INITIALIZE prepara a estrutura de "diretores" sobre a qual FIND e UNION operam.

Desempenho do algoritmo. Vamos supor que as implementações de INITIALIZE, FIND e UNION usam a heurística *union-by-rank*. Então INITIALIZE consome $O(n)$ unidades de tempo, cada execução de FIND consome $O(\lg n)$ e cada execução de UNION consome $O(1)$ unidades de tempo. Com isso, o bloco de linhas 2-10 do algoritmo MST-KRUSKAL consome $O(n + m \lg n)$ unidades de tempo.

A linha 1 do algoritmo consome $O(m \lg m)$ unidades de tempo. Como $m < n^2$, podemos dizer que a linha 1 consome $O(m \lg n)$ unidades de tempo. Assim, o consumo de tempo total de MST-KRUSKAL está em

$$O(n + m \lg n).$$

Poderíamos lançar mão das implementações de INITIALIZE, FIND e UNION que usam a heurística *path compression* (além da heurística *union-by-rank*). Com isso, o bloco de linhas 2-10 do algoritmo MST-KRUSKAL consumiria apenas $O(n + m \log^* n)$ unidades de tempo, em termos amortizados. Mas o consumo de tempo total do algoritmo continuaria em $O(n + m \lg n)$ por conta da linha 1, que rearranja as arestas em ordem crescente de peso.

Exercícios 3

1. Execute o algoritmo MST-KRUSKAL sobre o grafo definido pela matriz de pesos abaixo, com "-" representando "∞".

	1	2	3	4	5	6	7	8
1	-	3	-	4	5	-	-	-
2	3	-	9	-	6	-	-	-
3	-	9	-	-	8	2	-	-
4	4	-	-	-	6	-	6	7
5	5	6	8	6	-	9	-	8
6	-	-	2	-	9	-	-	-
7	-	-	-	6	-	-	-	8
8	-	-	-	7	8	-	8	-

2. Mostre que algoritmo MST-KRUSKAL consome $\Omega(n + m \lg n)$ unidades de tempo no pior caso. Que parte do algoritmo impede uma implementação mais rápida?

3. Escreva uma versão do algoritmo MST-KRUSKAL que calcule apenas *o peso* de uma árvore de peso mínimo. Escreva uma versão enxuta, sem variáveis e comandos supérfluos.

4. Suponha que o peso de cada aresta pertence ao conjunto 1..9. Descreva informalmente a implementação assintoticamente mais rápida que você conhece para o algoritmo de Kruskal nesse caso. Qual o consumo de tempo, em notação O, de sua implementação?

5. ★ *Árvore geradora de peso máximo.* Seja MST-Z um algoritmo que resolve o problema da MST. Use esse algoritmo para resolver o seguinte problema: Dado um grafo conexo G e um vetor de pesos sobre as arestas de G, encontrar uma árvore geradora de G que tenha peso *máximo*.

6. Seja G um grafo conexo e p um vetor de pesos sobre as arestas de G. Sejam a e b duas arestas de G. Encontre uma árvore geradora T de G que contenha a e b e minimize a soma dos pesos das arestas que não estão em T.

7. ★ *Grafos desconexos.* Modifique a definição de árvore geradora de modo que toda instância do problema da MST tenha solução (mesmo que o grafo seja desconexo). Escreva uma versão do algoritmo MST-KRUSKAL para essa versão modificada do problema.

8. *Bottleneck spanning tree.* Descreva um algoritmo que resolva o seguinte problema: dado um grafo conexo com vetor de pesos p, encontrar uma árvore geradora cuja aresta mais pesada é o mais leve possível, ou seja, encontrar uma árvore geradora cujo peso seja min $\max_{uv} p[uv]$, sendo o mínimo tomado sobre todas as árvores geradoras e o máximo sobre todas as arestas da árvore).

Veja o verbete *Minimum spanning tree* na Wikipedia.

---

www.ime.usp.br/~pf/analise_de_algoritmos/
Atualizado em 2021-12-31

Departamento de Ciência da Computação
Instituto de Matemática e Estatística da USP