Construção e Análise de Algoritmos

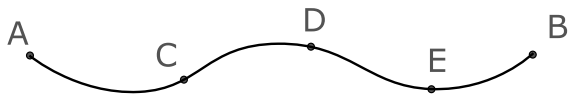
aula 17: Caminhos mais curtos

1 Introdução

> *Imagine que o governo do estado construiu diversas estradas entre as cidades mencionadas na aula passada — só imagine porque, como as cidades não existem, nada foi feito.*
>
> *E imagine também que um habitante, digamos, de Quiçá Dá, quer descobrir a maneira mais rápida de chegar a Pacotin.*
>
> *Ele pega o mapa e, depois de examiná-lo por alguns instantes, se dá conta de que existem inúmeras maneiras de fazer esse trajeto — muitas mesmo, talvez infinitas ...*
>
> *Um pouco confuso com essa observação, ele pensa que talvez fosse uma boa ideia passar em Guaranámiranga no meio do caminho.*
>
> *Mas então ele percebe que agora ele precisa descobrir o caminho mais curto até Guaranámiranga.*
>
> *Depois de pensar um pouco mais sobre o problema, ele resolve calcular caminhos mais curtos de Quiçá Dá até todas as outras cidades do mapa.*

A nossa historinha de hoje não apenas apresenta um problema de otimização, mas também começa a esboçar um raciocínio para a sua solução.

A primeira observação importante é que, se você quer encontrar um caminho de A até uma cidade distante B, então você vai precisar passar por cidades intermediárias C, D, E, ...

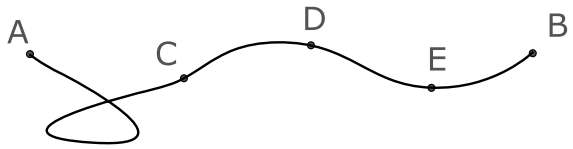


e isso significa que, na prática, você também vai encontrar caminhos de A até C, de A até D, etc.

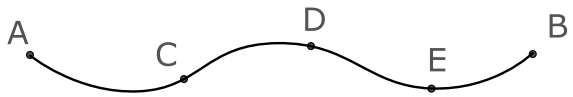
A segunda observação importante é que, se você quer encontrar um caminho mais curto de A até B que passa por C, então você terá que encontrar também um caminho mais curto de A para C.

Porque?

Ora, porque se você tem um caminho de A até B passando por C, e o trecho entre A e C não é o mais curto possível



então você pode trocar esse trecho por um outro mais curto



e, dessa maneira, reduzir a distância do caminho inteiro.

Quer dizer, isso significa que o primeiro caminho não era o mais curto possível.

A terceira observação importante é que, antes de começarmos a resolver o problema, nós não temos a menor ideia de onde o caminho vai passar — por exemplo, se ele vai passar em C ou não.

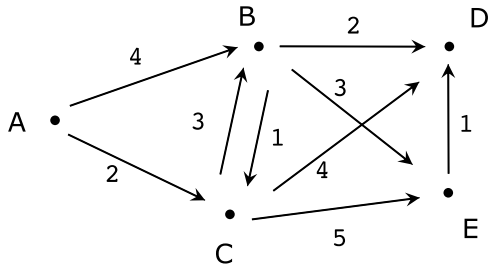
Quer dizer, na prática, nós começamos encontrando pequenos caminhos mais curtos de A para cidades próximas, depois nós encontramos caminhos que vão um pouquinho mais longe, daí as coisas começam a se encaixar, e eventualmente nós chegamos até o destino B (em nosso pensamento).

É por isso que o nosso personagem, depois de pensar um pouco sobre o assunto, resove encontrar caminhos mais curtos até todas as outras cidades.

2 Estratégia gulosa

Para tornar as coisas mais concretas, nós vamos mais uma vez pensar sobre o problema em termos de grafos.

Abaixo nós temos um pequeno exemplo.



Dessa vez, nós utilizamos arestas direcionadas para levar em conta a possibilidade de estradas de mão única.

Os números ao lado de cada aresta indicam a distância percorrida quando se passa por aquela estrada, e a distância de um caminho é dada pela soma das distâncias das suas arestas.

Por exemplo, o seguinte caminho de A até D

$$A \xrightarrow{4} B \xrightarrow{1} C \xrightarrow{5} E \xrightarrow{1} D$$

tem distância 11.

O nosso problema consiste em encontrar caminhos mais curtos de um vértice qualquer, digamos A, até todos os outros.

Isto é claramente um problema de otimização.

E nós começamos perguntando

- *O que seria uma estratégia gulosa para esse problema?*

Para responder essa pergunta, é útil examinar a estrutura do nosso problema em mais detalhe.

Como nas duas aulas anteriores, a solução do problema é uma coleção: a coleção dos caminhos mais curtos de A até todos os outros vértices.

Mas, dessa vez, nós não vamos construir essa solução selecionando caminhos de uma coleção maior.

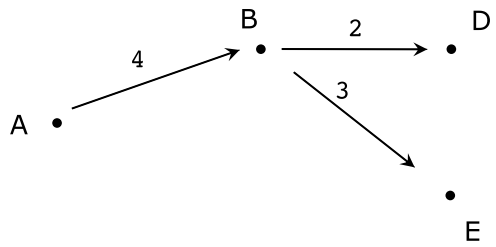
Quer dizer, nós não temos acesso direto a todos os caminhos que começam em A e suas respectivas distâncias — até porque, existem infinitos deles.

Tudo o que nós temos é o grafo.

Além disso, note que os caminhos são coleções aeles próprios também: coleções das arestas que formam os caminhos.

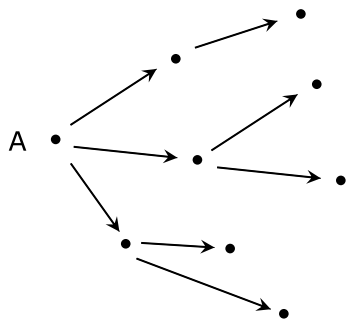
Em, além do mais, uma mesma aresta pode fazer parte de dois caminhos mais curtos.

No exemplo acima, os caminhos mais curtos de A até B e E são



e ambos compartinham a aresta A → B.

Portanto, a melhor maneira de pensar sobre a solução do nosso problema é como uma árvore de caminhos mais curtos



Quer dizer, para encontrar o caminho mais curto, digamos, de A até J no grafo, nós encontramos o caminho de A até J na árvore solução — e só existe um.

Portanto, o nosso problema consiste em construir uma árvore de caminhos mais curtos para o grafo, começando no vértice A.

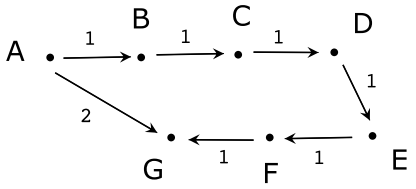
Agora as coisas ficaram mais fáceis.

Como uma árvore é uma coleção de arestas, a estratégia de solução consiste em selecionar arestas (gulosamente) para fazer parte da árvore.

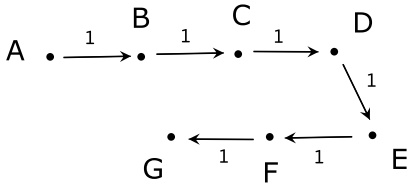
Certo, mas qual é uma boa regra para escolher essas arestas?

Intuitivamente, nós deveríamos dar preferência a arestas com menor distância.

Mas, o exemplo abaixo mostra que é preciso fazer isso com cuidado.



Quer dizer, se a cada passo nós escolhemos a aresta de menor distância que ainda não está na árvore (e que não forma ciclo), nós vamos acabar encontrando o seguinte caminho de A até G



que claramente não é ótimo.

Esse exemplo mostra que a distância de uma única aresta não é informação suficiente para decidir a questão.

Quer dizer, apesar da aresta F $\rightarrow$ G ter distância apenas 1, é preciso levar em conta que é necessário percorrer uma distância igual a 5 para se chegar até F.

Portanto, o valor que deveria estar associado à aresta F $\rightarrow$ G é $5 + 1 = 6$, e não 1.

Quando fazemos isso, fica claro que a aresta A $\rightarrow G$ é uma escolha melhor, e assim nós encontramos o caminho ótimo de A até G.

Mas, ainda falta um detalhe para transformar essas ideias em uma estratégia gulosa para a solução do problema.

Note que nós só podemos associar o valor $5 + 1$ à aresta F $\rightarrow$ G quando nós descobrimos o caminho de distância 5 até F.

Quer dizer, no início, quando nós ainda não conhecemos caminho algum, a maioria das arestas possui valor desconhecido.

Isto é, no início, nós só sabemos o valor das arestas que saem do vértice A — que possuem valor igual à sua própria distância.

A ideia, então, é que a estratégia gulosa escolhe a aresta de menor valor nesse conjunto, digamos A $\rightarrow$ D.

Quando fazemos isso, nós descobrimos um caminho mais curto até D.

(Porque?)

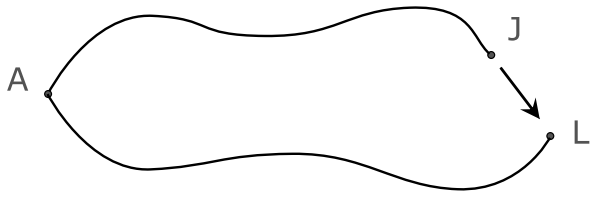
E, nesse momento, nós podemos calcular o valor das arestas que saem de D, que são adicionadas ao conjunto para fazer a segunda escolha.

A ideia é continuar dessa maneira até construir uma árvore (i.e., uma coleção com $n - 1$ arestas).

Mas, ainda é preciso discutir um último detalhe.

É possível que, ao selecionar uma aresta de menor valor do conjunto, digamos J $\rightarrow$ L, nós obtenhamos uma aresta que leva a um vértice que já foi alcançado.

Nesse caso, nós acabamos de encontrar um segundo caminho para L



E isto também corresponde a formar um ciclo — isto é, sem levar em conta a direção das arestas.

Mas, nós não precisamos nos preocupar com isso:

- O primeiro caminho encontrado até um vértice tem sempre distância menor ou igual à distância de qualquer outro caminho que é encontrado mais tarde.

Portanto, nessa situação, nós apenas descartamos a aresta J $\rightarrow$ L.

Pronto, agora nós já temos uma estratégia gulosa para o problema.

Mas, será que ela sempre encontra caminhos mais curtos para todos os vértices (em todos os casos)?

3 Argumento de otimalidade

Sim.

E nós vamos utilzar a nossa estratégia padrão para demonstrar esse fato.

Isto é, nós vamos comparar a árvore solução S encontrada pelo nosso algoritmo com uma outra árvore de caminhos T — que foi encontrada sabe-se lá como ...

O objetivo é mostrar, para todo vértice V, que o caminho de A até V em S tem distância menor ou igual a distância do caminho de A até V em T.

Ou, em linguagem matemática

$$S \preceq T \quad \equiv \quad \text{Dist}_S(A \rightsquigarrow V) \leq \text{Dist}_T(A \rightsquigarrow V) \text{ , para todo V}$$

Para fazer isso, seja $v_1, v_2, v_3, \ldots, v_{n-1}$ a ordem em que os vértices foram alcançados durante a construção da árvore S, na execução do algoritmo guloso.

Considere primeiramente o vértice v_1.

Nós podemos afirmar com certeza que $\text{Dist}_S(A \rightsquigarrow v_1) \leq \text{Dist}_T(A \rightsquigarrow v_1)$.

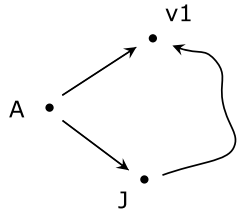
Porque?

Bom, esse caminho é formado por apenas uma aresta: A $\rightarrow v_1$.

E daí, das duas uma: ou esse é o caminho até v_1 em T ou ele não é.

Suponha que não.

Nesse caso, o caminho de A até v_1 em T deve começar com outra aresta, digamos A $\rightarrow$ J



Mas, lembre que, na primeira escolha, o algoritmo guloso seleciona uma aresta de distância mínima que sai do vértice A.

Isso significa que

$$\text{dist}(A \to v_1) \quad \leq \quad \text{dist}(A \to J)$$

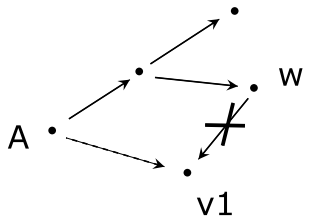
o que, por sua vez, implica que

$$\text{Dist}_S(A \rightsquigarrow v_1) \quad \leq \quad \text{Dist}_T(A \rightsquigarrow v_1)$$

Certo.

A seguir, nós modificamos ligeiramente a árvore T par que ela fique um pouco mais parecida com a árvore S — um truque que nós aprendemos na aula passada.

Especificamente, nós adicionamos a aresta A $\to$ v_1 a T, e removemos de T a aresta que chegava em v_1, digamos $w \to v_1$



$$T' \quad = \quad T \ \cup \ \{A \to v_1\} \ - \ \{w \to v_1\}$$

Não é difícil ver que T' é uma árvore de caminhos.

(Porque?)

Além disso, essa operação só pode afetar os caminhos de T que passam por v_1.

E, se afetar, só pode reduzir as suas distâncias.

(Porque?)

Portanto, para todo vértice V,

$$\text{Dist}_{T'}(A, V) \quad \leq \quad \text{Dist}_T(A, V)$$

Isto é, $T' \preceq T$.

A seguir, considere o vértice v_2.

Como no caso anterior, ou o caminho de A até v_2 em T' é igual ao caminho em S, ou não.

Suponha que não.

Nesse caso, nós concentramos a nossa atenção sobre a última aresta desse caminho, digamos $x \to v_2$

E, então, nós observamos que das duas uma: ou a aresta $x \to v_2$ estava no conjunto de arestas no momento da segunda escolha do algoritmo guloso, ou ela não estava.

Suponha primeiramente que ela estava.

Se isso é o caso, então $x = A$ ou $x = v_1$.

A observação chave, a seguir, é que os caminhos de A até A e v_1 são os mesmos em S e T'.

Isso significa que a distância do caminho de A até v_2 em T' é igual o valor de $A \to v_2$ no conjunto de arestas.

Ora, mas essa não foi a aresta selecionada pelo algoritmo guloso na segunda escolha.

Isso significa que

$$\text{Dist}_S(A \rightsquigarrow v_2) \quad \leq \quad \text{Dist}_{T'}(A \rightsquigarrow v_2)$$

Agora, suponha que $x \to v_2$ não estava no conjunto de arestas no momento da segunda escolha.

Nessa caso, nós vamos fazer uma operação mais delicada.

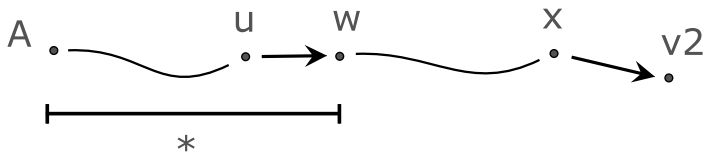
Nós vamos andar par trás no caminho de A até v_2 em T'

Nós vamos caminhar até encontrar uma aresta $u \to w$ que está no conjunto de arestas.

Essa aresta certamente existe.

(Porque?)

E, quando nós encontramos essa aresta, basta repetir o argumento acima para o trecho de caminho



Portanto, nós conseguimos verificar, em ambos os casos, que

$$\text{Dist}_S(A \rightsquigarrow v_2) \quad \leq \quad \text{Dist}_{T'}(A \rightsquigarrow v_2)$$

A seguir, nós removemos a aresta $x \to v_2$ de T', e fazemos com que o caminho de A até v_2 em T' seja igual ao caminho em S:

$$T'' \quad = \quad T' \; - \; \{x \to v_2\} \; \cup \; \Big\{A \to v_2 \;\; \text{ou} \;\; v_1 \to v_2\Big\}$$

Não é difícil ver que

- T'' é uma árvore de caminhos
- os caminhos até v_1, v_2 em T'' são iguais os caminhos em S
- $T'' \preceq T'$

A seguir, nós vamos dar mais um passo no argumento para mostrar que nós entramos em repetição.

Considere o vértice v_3.

O caminho de A até v_3 em T'' é igual ao caminho em S ou não.

Suponha que não, e seja $z \to v_3$ a última aresta nesse caminho.

Então, das duas uma: ou $z \to v_3$ estava no conjunto de arestas no momento da terceira escolha do algoritmo guloso, ou ela não estava.

Suponha primeiramente que ela estava.

Se isso é o caso, então $z = \{A, v_1, v_2\}$.

Além disso, como os caminhos até v_1, v_2 em T'' são os mesmos que em S, nós podemos concluir que o valor da aresta $z \to v_3$ no conjunto é igual a $\text{Dist}_{T''}(A \rightsquigarrow v_3)$.

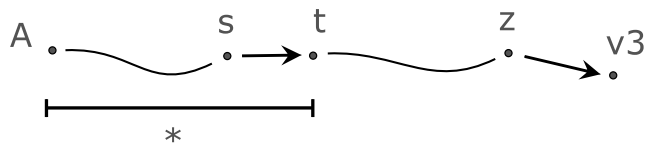
Ora, mas essa não foi a aresta selecionada pelo algoritmo guloso na terceira escolha.

Isso significa que

$$\text{Dist}_S(A \rightsquigarrow v_3) \quad \leq \quad \text{Dist}_{T''}(A \rightsquigarrow v_3)$$

Agora, suponha que $z \to v_3$ não estava no conjunto no momento da terceira escolha.

Então, nós andamos para trás no caminho até encontrar a primeira aresta que estava, digamos $s \to t$



Agora, basta repetir o argumento para o trecho $(*) : A \rightsquigarrow t$.

Portanto, nós conseguimos verificar, em ambos os casos, que

$$\text{Dist}_S(A \rightsquigarrow v_3) \quad \leq \quad \text{Dist}_{T''}(A \rightsquigarrow v_3)$$

A seguir, nós removemos a aresta $z \to v_3$ de T'', e fazemos com que o caminho de A até v_3 em T'' seja igual ao que é em S

$$T'' \quad = \quad T' \; - \; \{x \to v_2\} \; \cup \; \Big\{A \to v_3 \;\; \text{ou} \;\; v_1 \to v_3 \;\; \text{ou} \;\; v_2 \to v_3\Big\}$$

Essa operação produz uma árvore de caminhos T''', onde os caminhos até v_1, v_2, v_3 são iguais aos caminhos em S, e que satisfaz $T''' \preceq T''$.

Prosseguindo dessa maneira, nós construímos uma sequência de árvores

$$T''^{\cdots} \quad \preceq \quad \ldots \quad \preceq \quad T'' \quad \preceq \quad T' \quad \preceq \quad T$$

E, dado que $T''^{\cdots}$ é a própria árvore S, nós temos o resultado desejado:

$$S \quad \preceq \quad T$$

4 Algoritmo e análise de complexidade

Agora que sabemos que a nossa estratégia gulosa é ótima, só nos resta construir o algoritmo e analisar o seu tempo de execução.

A implementação da estratégia gulosa na forma de um algoritmo é imediata:

```
  Procedimento  Caminhos+Curtos  ( grafo G, vértice inicial A )
   {
1.      Marcar o vértice  A  como alcançado

2.      A.dist  <--  0

3.      B  <--  conjunto com todas as arestas que saem do vértice A,
                  com a sua própria distância como valor

4.      S  <--  vazio

5.      Enquanto ( S ainda não contém  n-1  arestas )
         {
6.           (v -> u)   <--  aresta de menor valor em  B

7.           Se ( u ainda não foi alcançado )
              {
8.                u.dist  <--  valor de  (v -> u)
```

```
 9.                Marcar o vértice  u  como alcançado

10.                Incluir a aresta  (v -> u)  em  S

11.                Incluir toda aresta  (u -> z)  em  B,
                      com valor  u.dist + dist(u -> z)
                }
            }
12.     Retorna (S)
    }
```

Ignorando as linhas `3, 6, 11`, que manipulam o conjunto `B`, é fácil ver que

- o trecho antes do laço executa em tempo $O(1)$
- o laço realiza $O(n)$ voltas, e todas as instruções no seu interior (com a exceção da linha 11) executam em tempo $O(1)$

A seguir, nós observamos que o algoritmo realiza as seguintes operações sobre o conjunto `B`:

- inserção de um novo elemento
- remoção do menor elemento

Isso nos dá a ideia de implementar o conjunto `B` como um heap.

Esse heap irá armazenar arestas do grafo.

Logo, denotando o número total de arestas do grafo por m, nós observamos que as operações sobre ele são realizadas em tempo $O(\log m)$.

Finalmente, é chegada a hora de levar em conta as linhas `3, 6, 11`.

Nesse ponto, nós observamos que essas instruções inserem e removem de `B` cada aresta do grafo no máximo uma vez.

Isso significa que a contribuição dessas linhas para o tempo de execução do algoritmo é no máximo

$$m \cdot O(\log m) \quad = \quad O(m \log m)$$

Portanto, o tempo de execução do algoritmo é

$$O(1) \;+\; O(n) \;+\; O(m \log m) \quad = \quad O(m \log m)$$