Construção e Análise de Algoritmos

aula 20: Estratégias gulosas de aproximação

1 Introdução

Há algumas aulas atrás, quando apresentamos a ideia dos algoritmos gulosos, nós vimos que a estratégia gulosa para o problema da partição de conjuntos não funcionou.

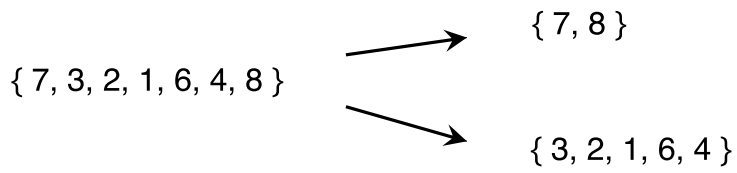
Depois disso, nós vimos 4 exemplos onde a estratégia gulosa consegue encontrar uma solução ótima para o problema.

Agora é a hora de examinarmos novamente exemplos onde a estratégia gulosa não tem garantia de otimalidade.

Relembre o problema da partição de conjuntos:

- dividir os números de um conjunto C em dois subconjuntos A e B,
  de modo que as somas dos números em A e B sejam o mais próximas possível.

Por exemplo,



E relembre a estratégia gulosa que nós apresentamos na aula 15:

- ordene o conjunto C e vá colocando os números um a um,
  do maior para o menor, no subconjunto com a menor soma no momento
  (em caso de empate, coloque o número em A).

Nós já vimos que essa estratégia não encontra uma solução ótima para o exemplo acima.

Mas, a solução gulosa não é tão ruim assim: nessa solução a diferença entre as duas somas é igual a 4, e na solução ótima essa diferença é igual a 0.

Mas, será que essa estratégia gulosa sempre encontra uma solução *relativamente boa* em todos os casos?

E, mais importante, como é que nós podemos saber que a solução gulosa é relativamente boa quando nós não temos a solução ótima para comparar com ela?

A resposta é:

- Em alguns casos, é possível formular um *argumento de aproximação* para a estratégia gulosa.

Como no caso dos argumentos de otimalidade, não é preciso conhecer a solução do problema para formular o argumento.

E o argumento de aproximação nos dá uma estimativa da distância entre a solução gulosa e uma solução ótima (no pior caso).

Dessa maneira, nós conseguimos saber o que o termo "*relativamente boa*" significa.

Vejamos o que se pode dizer no caso do problema da partição.

Considere o exemplo

$$C \;=\; \{\, 20,\, 19,\, 18,\, 17,\, 16,\, 15,\, 14,\, 13,\, 12 \,\}$$

Não é difícil ver que a estratégia gulos produz a seguinte solução para o problema

$$A \;=\; \{\, 20,\, 17,\, 16,\, 13,\, 12 \,\} \qquad\qquad B \;=\; \{\, 19,\, 18,\, 15,\, 14 \,\}$$

que possui diferença $78 - 66 \;=\; 12$.

Agora, veja o que acontece quando nós trocamos os elementos 20 e 14 de posição

$$A' \;=\; \{\, 17,\, 16,\, 14,\, 13,\, 12 \,\} \qquad\qquad B' \;=\; \{\, 20,\, 19,\, 18,\, 15 \,\}$$

A troca reduz a diferença a 0, e isso nos dá a solução ótima do problema.

Nesse ponto, alguém pode ter a ideia de multiplicar todos os números por 10

$$C \;=\; \{\, 200,\, 190,\, 180,\, \ldots,\, 130,\, 120 \,\}$$

e dessa maneira aumentar a distância da solução ótima para a solução gulosa de 12 para 120.

Mas, não parece razoável dizer que a estratégia gulosa teve um desempenho pior no segundo exemplo do que no primeiro.

Quer dizer, olhar para as diferenças talvez não seja a melhor maneira de comparar a solução gulosa com a solução ótima.

Uma outra ideia consiste em olhar para o tamanho das partições (ou a sua soma).

Suponha que a soma total dos elementos do conjunto C seja igual a M.

Então, o melhor que a solução ótima pode fazer é produzir duas partições de tamanho $M/2$ — i.e., quando a diferença é igual a 0.

Para estimar a qualidade de uma estratégia gulosa para o problema, nós podemos verificar o

quão perto ela chega desse resultado.

No primeiro exemplo acima, a solução ótima consiste de duas partições de tamanho 72, enquanto que a solução gulosa consiste de partições de tamanhos 78 e 66.

Comparando os tamanhos das maiores partições de cada solução, nós obtemos

$$78 \quad = \quad \underbrace{\frac{13}{12}}_{*} \times 72$$

Note que o fator $(*)$ não muda quando nós multiplicamos todos os números por 10!

Nesse outro exemplo

$$C \quad = \quad \{\, 8,\, 7,\, 6,\, 5,\, 4 \,\}$$

a solução ótima novamente consiste de duas partições de tamanhos iguais, enquanto que a estratégia gulosa produz partições de tamanhos 17 e 13.

Portanto, o fator de comparação nesse caso é de 17/15.

O que se pode ver é que o fator muda de exemplo para exemplo.

Mas, é possível mostrar* que o fator de comparação nunca é maior do que 7/6.

E isso nos dá uma garantia sobre a qualidade da solução gulosa em qualquer caso.

A seguir, nós vamos ver outros exemplos de aproximação gulosa.

2 O problema da cobertura de vértices

> *A quebra de sigilo telefônico de um grupo de suspeitos de um crime revelou todo o histórico de ligações que eles fizeram entre sim no último mês.*
>
> *Mas, o histórico não revela o conteúdo das conversas.*
>
> *Para isso, é preciso fazer interrogatórios individuais.*
>
> *Em princípio, a polícia gostaria de saber tudo o que foi dito e ouvido nessas conversas.*
>
> *Mas, para fazer as investigações andarem mais rápido, eles gostariam de interrogar o menor número de pessoas possível.*
>
> *Como fazer para escolher os suspeitos que serão interrogados?*

Como usual, a historinha nos dá um problema de otimização.

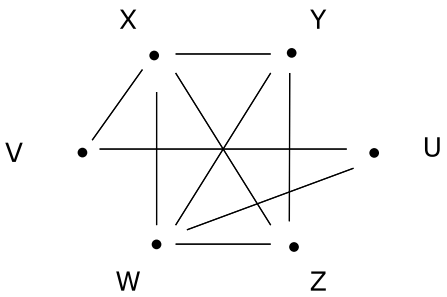
Para formular uma estratégia gulosa para ele, é importante entender o que está acontencendo.

*por meio de um argumento complicado

Uma boa maneira de visualizar o problema consiste em modelar a situação na forma de um grafo

- cada pessoa corresponde a um *vértice* do grafo
- e nós colocamos uma aresta entre dois vértices se as respectivas pessoas conversaram entre si

Por exemplo,



Agora, é fácil ver que uma solução para o problema corresponde a

- um subconjunto de vértices S tal que toda aresta (u, v) do grafo possui ao menos um de seus vértices em S.

A ideia aqui é que, para uma aresta (u, v) qualquer, nós podemos dizer que os vértices u e v *cobrem* essa aresta, e o objetivo é encontrar um subconjunto de vértices que cobrem todos as arestas do grafo — daí o nome do problema.

Uma solução ótima para o problema consiste em uma cobertura de vértices com o menor tamanho possível.

Examinando o grafo exemplo acima, parece natural escolher primeiro um dos vértices que cobrem o maior número de arestas: X ou W.

Suponha que nós escolhemos X.

Em seguida, nós escolhemos o vértice que cobre o maior número de arestas que ainda estão descobertas: o vértice W.

E assim por diante.

Pronto, nós já temos uma estratégia gulosa.

No exemplo acima, essa estratégia produz a solução $\{X, W, U, Y\}$ (assumindo o desempate pela ordem alfabética).

E não é muito difícil verificar que essa solução é ótima.

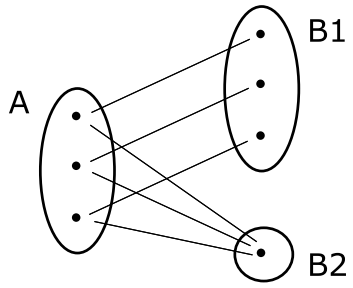
(Quer tentar?)

Mas, será que essa estratégia encontra uma soluçã ótima em todos os casos?

Infelizmente não ...

Na verdade, é bem fácil enganá-la.

Veja só.

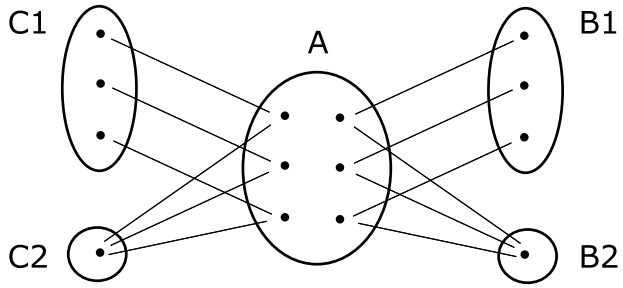


Nesse exemplo, a solução ótima claramente é dada pelo subconjunto A.

Mas, ao optar pelo vértice em B_2 na primeira escolha, o algoritmo guloso acaba encontrando uma solução com 4 vértices.

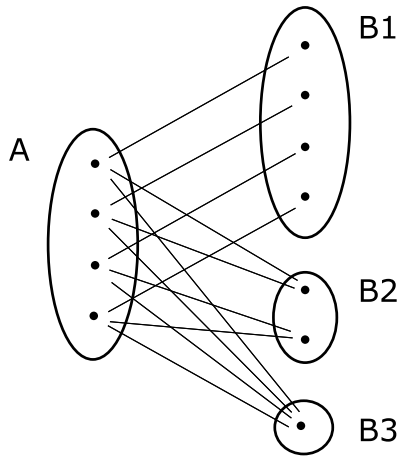
(Você consegue ver?)

Nesse exemplo, a diferença entre as soluções é de apenas 1 vértice, mas não é difícil fazer essa diferença aumentar



Aqui a diferença entre as soluções é de 2 vértices, mas nós já sabemos que esse tipo de exemplo não é muito significativo.

Considere esse outro exemplo



Aqui, o algortimo guloso seleciona o vértice em B_3 primeiro, pois ele possui 4 arestas descobertas.

Quando isso é feito, os vértices em A passam a ter apenas 2 arestas descobertas — o mesmo que os vértices em B_2.

Nesse ponto, a gente imagina que o algoritmo guloso dá azar e opta pelos vértices em B_2.

Com essa opção, o algoritmo guloso precisa selecionar mais 4 vértices, e termina com uma solução de tamanho 7.

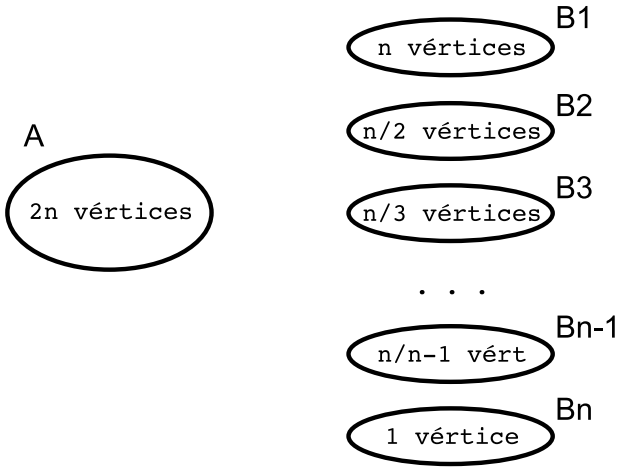
Mas, como é fácil de ver, a solução ótima é dada pelo subconjunto A que possui apenas 4 vértices.

Além disso, se examinarmos o fator de aproximação, nós podemos verificar que ele está aumentando

- 4/3 no primeiro exemplo
- 7/4 no segundo exemplo

O próximo passo consiste em eliminar a dependência do argumento com relação ao "azar" do algoritmo guloso, e descobrir como construir exemplos desse tipo arbitrariamente grandes.

Depois de muita tentativa e erro (e bastante papel gasto como rascunho), nós chegamos à seguinte construção



A ideia aqui é que

- cada vértice em B_1 está ligado a 2 vértices em A
- cada vértice em B_2 está ligado a 4 vértices em A
- ...
- cada vértice em B_{n-1} está ligado aos $2n/(n/n-1) = 2(n-1)$ vértices de A
- o único vértice em B_n está ligado aos $2n$ vértices de A

A solução ótima para o problema, mais uma vez, é dada pelo subconjunto A.

E, dessa vez, o algoritmo guloso é forçado a escolher os vértices dos subconjuntos $B_1, B_2, \ldots, B_n$, construindo uma solução de tamanho

$$n + \frac{n}{2} + \frac{n}{3} + \ldots + 1 + \quad = \quad n \cdot \underbrace{\left(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right)}_{*}$$

Para comparar as duas soluções, é preciso estimar o valor da expressão $(*)$.

Alguns poucos truques matemáticos sujos[†] nos dão o seguinte resultado

$$1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \quad = \quad \Theta(\log n)$$

Portanto, a solução do algoritmo guloso tem tamanho $\Theta(n \log n)$.

E, comparando com a solução ótima, nós obtemos o fator de aproximação $\Theta(\log n)$.

Esse resultado não é nem muito bom nem muito ruim.

Ele não é muito bom pois ele mostra que a qualidade da solução gulosa vai degradando cada vez mais, a medida que n aumenta.

Mas, ele não é muito ruim pois a função $\log n$ cresce muito devagar.

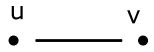
A seguir, nós vamos ver que é possível fazer melhor que isso.

Estratégia gulosa alternativa

Olhando para o problema a partir de um ponto de vista diferente, nós podemos obter uma estratégia gulosa mais eficiente.

A ideia é olhar para as arestas, ao invés dos vértices.

Quer dizer, quando focamos a atenção em uma aresta qualquer

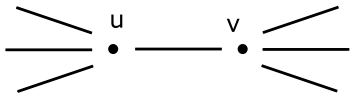


nós podemos dizer com certeza que ao menos um dos vértices u e v deve estar na solução (talvez os dois) — (Porque?)

Portanto, se nós decidirmos incluir os vértices u e v na solução gulosa, então nós não vamos estar errando por muito — i.e., no máximo por um fator de 2.

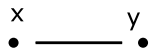
Ao colocar os vértices u e v na solução, não apenas a aresta (u, v) fica coberta, mas também qualquer outra aresta de u e de v

[†]ou uma rápida olhada na internet



A seguir, nós concentramos a nossa atenção em uma outra aresta que ainda está descoberta.

Por exemplo,



Ao fazer isso, nós podemos dizer com certeza que ao menos um dos vértices x e y deve estar na solução do problema (talvez os dois) — mesmo que a solução já contenha u e v.

(Porque?)

Portanto, se nós decidimos incluir os vértices x e y na solução gulosa, então nós não vamos estar errando por muito — novamente, no máximo um fator de 2.

A ideia é continuar dessa maneira até que não restem mais arestas descobertas.

Resumindo, o algoritmo guloso é o seguinte

```
S  <--  vazio

B  <--  todas as arestas do grafo

Enquanto  ( B não está vazio )
 {
    (u,v)  <--  aresta qualquer de B

    Incluir os vértices  u,v  em  S

    Remover todas as arestas de  u  e  v  de  B
 }
Retorna (S)
```

Argumento de aproximação

Na apresentação da estratégia gulosa, nós já demos o palpite de que a solução gulosa é no máximo 2 vezes maior do que a solução ótima.

A seguir, nós vamos formular um argumento que demonstra esse fato mais claramente.

E a ideia, mais uma vez, é focar a atenção nas arestas ao invés dos vértices — não importa o que aconteça, não olhe para o lado!

(. . .)

3 Cobertura de conjuntos

Esse problema é uma generalização do anterior.

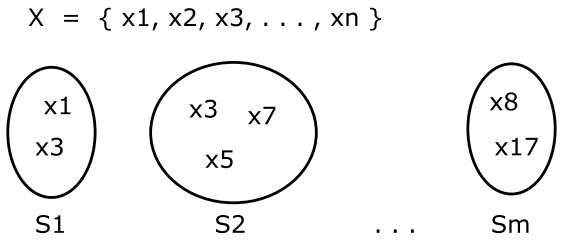
Quer dizer, a ideia é que agora as "arestas"do grafo podem ter mais do que dois vértices.

Vejamos a historinha.

> *Ao longo da investigação, a polícia descobriu que esse era um grupo criminoso muito sofisticado mesmo.*
>
> *Os interrogatórios confirmaram que os suspeitos eram todos culpados, e ainda revelaram a participação de várias outras pessoas.*
>
> *Mas, as investigações não conseguiram determinar como a coisa foi armada.*
>
> *Quer dizer, o plano não foi formulado durante as ligações, e tampouco houve uma reunião envolvendo todo mundo.*
>
> *As diferentes partes do esquema foram definidas em reuniões com um número restrito de pessoas.*
>
> *Agora, os investigadores querem saber tudo o que foi dito e ouvido nessas reuniões.*
>
> *E para isso, é preciso interrogar pessoas que estiveram presentes em todas elas.*
>
> *Como antes, a ideia é interrogar o menor número de pessoas possível.*

Essa situação pode ser representada como um *hipergrafo* — i.e., um grafo onde as arestas são subconjuntos arbitrários de vértices, e não apenas pares.

Mas, um hipergrafo também pode ser visto como uma família de subconjuntos



O problema da cobertura de conjuntos pode ser formulado da seguinte maneira:

- Dado um conjunto

$$X \quad = \quad \{ x_1,\ x_2,\ x_3,\ \ldots,\ x_n \}$$

  e uma família de subconjuntos de X

$$F \quad = \quad \{ S_1,\ S_2,\ S_3,\ \ldots,\ S_m \}$$

encontrar um subconjunto $Y \subseteq X$ de tamanho mínimo tal que

$$Y \cap S_j \neq \emptyset \qquad \text{, para todo } j.$$

A primeira observação que nós podemos fazer é que a estratégia gulosa que funcionou bem para um grafo simples não possui uma garantia razoável de aproximação.

É fácil ver isso.

Suponha que todos os subconjuntos S_j possuem, digamos, $n/2$ elementos.

E suponha também que existe um certo elemento x_k que está em todos os subconjuntos.

Agora, lembre que a estratégia gulosa consiste em escolher uma "aresta"(ou subconjunto) qualquer, e selecionar os seus vértices para a solução.

Nesse exemplo, isso corresponde a selecionar $n/2$ vértices logo no primeiro passo.

De fato, ao fazer isso todos os subconjuntos ficam cobertos.

(Porque?)

Dessa maneira, o algoritmo guloso termina com uma solução de tamanho $n/2$.

Mas, a solução ótima contém apenas um vértice: $\{x_k\}$.

Portanto, a solução gulosa é $n/2$ vezes maior do que a solução ótima.

E isso é muito ruim mesmo.

Mas, a seguir, nós vamos ver que a primeira estratégia gulosa mantém a mesma garantia de desempenho que ela tinha nos grafos simples.

Estratégia gulosa #1

A versão da primeira estratégia gulosa para esse problema é a seguinte:

- selecionar o elemento x_i que está no maior número de subconjuntos S_j
  eliminar esses subconjuntos do problema,
  e continuar dessa maneira.

A seguir, nós vamos demonstrar que essa estratégia tem fator de aproximação $\Theta(\log n)$.

Suponha que o conjunto X tem n elementos.

E suponha que a solução ótima tem tamanho k.

A ideia é mostrar que o laço do algoritmo guloso realiza no máximo $k \log n$ voltas.

Vejamos.

A primeira observação é que deve existir ao menos um elemento na solução ótima que está em ao menos n/k subconjuntos.

(Porque?)

Logo, como o algoritmo guloso escolhe o elemento de X que aparece na maior quantidade de subconjuntos, digamos x_1, esse elemento deve estar em ao menos n/k subconjuntos.

Note que x_1 não precisa fazer parte da solução ótima.

Mas, nós temos a certeza de que na primeira volta do laço, ao menos n/k subconjuntos são eliminados do problema.

Isso significa que agora o problema contém no máximo

$$n' \quad = \quad n \; - \; \frac{n}{k} \quad = \quad n \cdot \left(1 \; - \; \frac{1}{k}\right) \quad \text{subconjuntos}$$

A observação seguinte é que deve existir ao menos um elemento na solução ótima que aparece em ao menos n'/k desses subconjuntos

(Porque?)

Além disso, esse elemento não é s_1, pois ele não aparece em nenhum deles.

Isso significa que, na sua segunda escolha, o algoritmo guloso pode selecionar um elemento s_2 que deve estar em ao menos n'/k desses subconjuntos.

Novamente, s_2 não precisa fazer parte da solução ótima.

Mas, nós temos a certeza de que na segunda volta do laço, ao menos n'/k subconjuntos são eliminados do problema.

E isso significa que agora o problema contém no máximo

$$n'' \quad = \quad n' \; - \; \frac{n}{k} \quad = \quad n' \cdot \left(1 \; - \; \frac{1}{k}\right) \quad \text{subconjuntos}$$

E isso, por sua vez, implica que

$$n'' \quad = \quad n \cdot \left(1 \; - \; \frac{1}{k}\right)^2$$

Bom, é fácil ver que nós já entramos em repetição.

E que após a terceira volta do laço o problema contém no máximo

$$n''' \quad = \quad n \cdot \left(1 \; - \; \frac{1}{k}\right)^3 \quad \text{subconjuntos}$$

A coisa continua dessa maneira até que, após a t-ésima volta nós temos

$$n \cdot \left(1 \; - \; \frac{1}{k}\right)^t \quad < \quad 1$$

Nesse momento, não resta mais nenhum subconjunto no problema, e a execução do algoritmo acabou.

A nossa tarefa agora é determinar o valor de t.

A ideia é utilizar mais um fato matemático bem conhecido:

$$\left(1 \; - \; \frac{1}{k}\right)^t \quad \simeq \quad e^{-t/k}$$

Utilizando esse fato para reescrever a equação acima, nós obtemos

$$n \cdot e^{-t/k} \quad < \quad 1$$

que é equivalente a

$$e^{t/k} \quad > \quad n$$

Finalmente, passando o logaritmo natural em ambos os lados, nós obtemos

$$\frac{t}{k} \quad > \quad \ln n$$

que é equivalente a

$$t \quad > \quad k \cdot \ln n$$

Ou seja, o laço do algoritmo realiza no máximo $k \cdot \ln n$ voltas.

Como a cada volta do laço o algoritmo guloso seleciona um vértice, a solução encontrada tem no máximo $k \cdot \ln n$ vértices.

Comparando essa solução com a solução ótima, nós obtemos o fator de aproximação da estratégia gulosa:

$$\mathsf{F} \quad = \quad \frac{|\text{solução gulosa}|}{|\text{solução ótima}|} \quad = \quad \frac{k \cdot \ln n}{k} \quad = \quad \Theta(\log n)$$