Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest, Ronald L. Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden

Apresentação Baseada:

- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende

Caminhos Mínimos

- Vimos dois algoritmos para encontrar o caminho mais curto entre um determinado vértice fonte s e todos os outros vértices.
- Dijkstra e Bellman-Ford.
- Mas com frequência estaremos interessados em encontrar o caminho mínimo entre quaisquer dois vértices.

Caminho Mínimo entre Todos os Pares de Vértices

Dado um grafo direcionado $G = (V, A)$, com n vértices e m arcos, em que cada arco $a \in A$ tem um custo c_a (pode ser negativo). Desejamos encontrar o custo mínimo para ir de u a v para todos os pares $u, v \in V$ ou detectar se houver um ciclo de custo negativo.

Caminho Mínimo entre Todos os Pares de Vértices

- Podemos resolver esse problema, chamando o algoritmo de Caminho Mínimo de Única fonte algumas vezes. Quantas execuções seriam necessárias?
  1. 1
  2. $n - 1$
  3. n
  4. n^2
- Se o grafo não tem tem custos negativos, poderíamos usar o Algoritmo de Dijkstra, e obter uma complexidade de

$$O(nm \log n) \begin{cases} O(n^2 \log n) \text{ em grafos esparços} \\ O(n^3 \log n) \text{ em grafos densos} \end{cases}$$

- De fato não sabemos se algum algoritmo pode ser mais rápido que $O(n^3)$, uma vez que precisa ser calculado para todos os pares, e podemos imaginar que um trabalho linear por par seria necessário. (Mas lembre do algoritmo de Strassen para multiplicação de matrizes que era subcúbico).

Caminho Mínimo entre Todos os Pares de Vértices

- Mas se o grafo tem tem custos negativos, poderíamos usar o Bellman-Ford (executado n vezes):

$$O(n^2 m) \begin{cases} O(n^3) \text{ em grafos esparços} \\ O(n^4) \text{ em grafos densos} \end{cases}$$

- Parece ser uma complexidade muito grande. De fato o algoritmo de Floyd-Warshall que veremos executa em $O(n^3)$ independente do número de arcos, e funciona mesmo com os custos negativos.
- Portante é pelo menos tão bom quanto n execuções do Bellman-Ford. Muito melhor para grafos densos.
- Para grafos sem custos negativos, executar o Dijkstra n vezes ainda é melhor, mas para grafos densos ambos são competitivos.

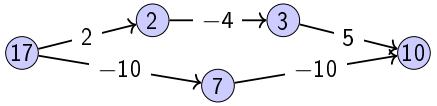
Caminho Mínimo entre Todos os Pares de Vértices

- Esse problema pode aparecer por exemplo se queremos saber se uma relação tem um fechamento transitivo (se existe um caminho orientado entre qualquer par de vértices).
- Assim como o Bellman-Ford, adaptações do Floyd-Warshall podem resolver vários problemas práticos.

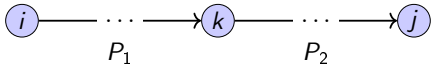
Ideias - Subestrutura Ótima

- Como vimos no Bellman-Ford, encontrar qual subproblema pode ser resolvido não é tão simples. Acabamos por perceber que o subproblema estava no número de arcos que poderíamos.
- **Idéia:** Vamos limitar o número de vértices que vamos usar. E também limitar quais vértices podemos usar.
- Considere uma ordenação arbitrária de $V = 1, 2, 3, \ldots, n$, e seja o prefixo $V^{(k)} = \{1, \ldots, k\}$.

- Suponha que G não tem ciclos negativos. Fixado uma origem $i \in V$, e um destino $j \in V$ e $k \in \{1, 2, \ldots, n\}$. Seja P o caminho de custo mínimo (portanto sem ciclos) entre i e j com todos os nós internos em $V^{(k)}$.
- Por exemplo, seja $i = 17$, $j = 10$ e $k = 5$.



- Caso 1: Se k não está em P, então P é um caminho de custo mínimo com todos os vértices internos em $V^{(k-1)}$.
- Caso 2: Se k está em P então podemos dividir P em dois pedaços.



- P_1 é o caminho de custo mínimo entre i e k com todos os nós internos em $V^{(k-1)}$ e P_2 é o caminho de custo mínimo entre k e j com todos os nós internos em $V^{(k-1)}$

- Seja S um vetor tridimensional indexado em i, j e k.
- $S[i,j,k]$ vai representar o custo do caminho mínimo entre i e j em que todos os nós internos estão em $\{1,2,\ldots,k\}$ (ou $+\infty$ se tal caminho não existir)
- Qual deve ser $S[i,j,0]$ se (I) $i=j$, (II) $(i,j)\in A$, (III) $i\neq j$ e $(i,j)\notin A$.
  a. 0, 0, e $+\infty$
  b. 0, c_{ij} e c_{ij}
  c. 0, c_{ij} e $+\infty$
  d. $+\infty$, c_{ij} e $+\infty$

Algoritmo 1: Floyd-Warshall(G)

Entrada: Um grafo caminho G

Saída: Os custos dos caminhos mínimos entre todos os pares de vértices

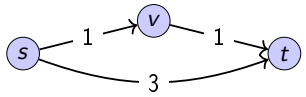
1 **para** *todo* $i \in V$ **faça**
2 **para** *todo* $j \in V$ **faça**
3 $$S[i,j,0] = \begin{cases} 0 \text{ se } i=j \\ c_{ij} \text{ se } (i,j)\in A \\ +\infty \text{ se } i\neq j \text{ e } (i,j)\notin A \end{cases}$$
4 **para** $k \in \{1,\ldots,n\}$ **faça**
5 **para** *todo* $i \in V$ **faça**
6 **para** *todo* $j \in V$ **faça**
7 $$S[i,j,k] = \min \begin{cases} S[i,j,k-1] \\ S[i,k,k-1]+S[k,j,k-1] \end{cases}$$

- Corretude: Pela subestrutura ótima e indução
- Complexidade: O(1) por subproblema, total de $O(n^3)$.
- E se o grafo tiver um ciclo de custo negativo?
- Nesse caso, no final do algoritmo você terá um $S[i,i,n]<0$ para pelo menos um $i\in V$
- E se precisamos reconstruir o caminho entre i e j?
- Nesse caso podemos guardar um vetor bidimensional auxiar $T[i,j]$ que vai guardar o vértice interno com o índice mais alto no caminho de i até j.

Algoritmo de Johnson

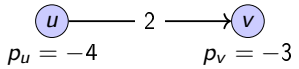
- Relembrando que podemos resolver o problema com n chamadas ao algoritmo de caminhos mínimos de única fonte:
  - Custos não negativos: $O(mn\log n)$ com o Dijkstra
  - Caso geral: $O(mn^2)$ com o Bellman-Ford
- Mas eu ainda quero usar o Dijkstra mesmo com pesos negativos. Será possível?
- O algoritmo de Johnson:
  - Faz 1 chamada ao Bellman-Ford
  - Faz n chamadas ao Dijkstra
- Portanto $O(mn)+O(nm\log n)=O(nm\log n)$

- Podemos só somar um valor em todas as arestas para tornar todos os custos positivos? Não! Suponha no grafo abaixo que somamos 2 em todas os arcos, perceba que o menor caminha muda. Só funcionaria se todos os caminhos tivessem o mesmo número de arestas.



- Nem tudo está perdido, podemos sim usar um rebalanceamento para se livrar dos pesos negativos.

- Seja $G = (V, A)$ um grafo direcionado com custos c_a nos arcos, podendo ser negativos. Fixe um número real p_v para cada vértice $v \in V$.
- Para cada arco $a = (u, v)$ de A, faça $c'_a = c_a + p_u - pv$

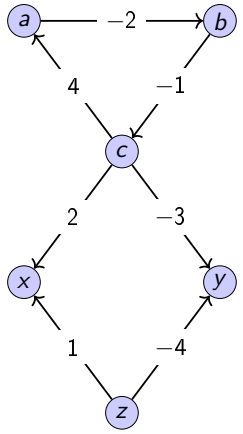


- Se o $s - t$ caminho P tem custo L com os custos originais, qual é o custo L' de P com os custos modificados?

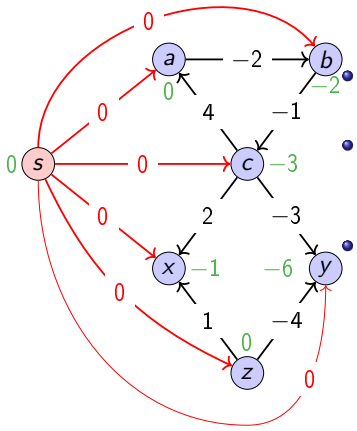
a. L
b. $L + p_s + P_t$
c. $L + p_s - P_t$
d. $L - p_s + P_t$

$$\begin{aligned} L' &= \sum_{a \in P} c'_a \\ &= \sum_{a=(u,v) \in P} c_a + p_u - p_v \\ &= \left(\sum_{a \in P} c_a \right) + p_s - p_t = L + p_s - p_t \end{aligned}$$

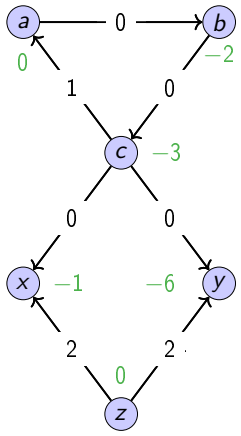
- Note que dessa forma qualquer caminho foi alterado exatamente pelo mesmo custo.
- Dessa forma os caminhos mínimos foram preservados (obviamente não o seu custo, mas é fácil de obtê-lo)
- E se encontramos valores p_v de forma que todos os arcos fiquem com custos não-negativos?
- Como encontrar esses valores? Alias, será que tais valores existem? 🤔
- Se o grafo não tiver ciclos negativos, SIM, esses valores existem!



- A ideia consiste em executar um algoritmo para o problema do caminho mínimo de única fonte.
- Um problema: É preciso alcançar todos os vértices, no grafo ao lado qualquer vértice que você escolher para ser sua fonte não vai alcançar todos os outros.
- Truque, inserir um novo vértice fictício que se liga a todos os outros por um arco de custo 0.



- Note que adicionar s e os novos arcos não cria nenhum novo caminho para qualquer $u, v \in V$.
- como existem arcos com custo negativo, nos resta executar o Bellman-Ford (o que também detecta ciclos negativos)
- Esse são exatamente os valores que procuramos. p_v é o custo do $s - t$ caminho mínimo.



- Podemos então para cada arco $a = (u, v)$ calcular o novo custo

$$c'_a = c_a + p_u - p_v$$

- Agora todos os custos ficaram não negativos 😎 (pelo menos para esse exemplo)
- Agora não precisamos mais usar o Bellman-Ford e podemos usar o Dijkstra!

Algoritmo 2: Johnson(G)

Entrada: Um grafo caminho G

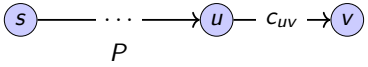
Saída: Os custos dos caminhos mínimos entre todos os pares de vértices

1 Formar G', adicionando à G um vértice s extra e um novo arco (s, v) para todo $v \in V$ com custo 0;
2 Executar o Bellman-Ford em G' com fonte s (se detectar ciclo negativo, terminar);
3 **para** $v \in V$ **faça** $p_v =$ custo do $s - t$ caminho mínimo em G';
4 **para** $(u, v) \in A$ **faça** $c'_{uv} = c_{uv} + p_u - p_v$;
5 **para** $v \in V$ **faça** Executar o Dijkstra em G com fonte em v e os custos $\{c'_a\}$, obtendo os custos $d'(u, v)$;
6 **para** *cada par* $u, v \in V$ **faça** $d(u, v) = d'(u, v) - p_u + p_v$;
7 devolva d;

Tempo de execução:

$$O(n) + O(mn) + O(m) + O(nm \log n) + O(n^2) \qquad = \qquad O(mn \log n)$$

- Corretude: Como os caminhos de custo mínimo são preservados, se não houver custos negativos as execuções do Dijkstra os encontram corretamente, logo o algoritmo de Jonhson funciona.
- Resta demonstrar que de fato os custos $\{c'_a\}$ são não negativos.
- Considere uma arco (u, v), por construção p_u é custo do caminho mínimo P de s a u, e p_v é custo do caminho mínimo de s a v.



Como $P + (u, v)$ é um possível caminho para v, certamente o caminho de custo mínimo tem um valor menor ou igual a esse. Ou seja

$$p_v \leq p_u + c_{uv}$$
$$0 \leq p_u + c_{uv} - p_v$$
$$c'_{uv} = p_u + c_{uv} - p_v \geq 0$$

Tempo polinomial

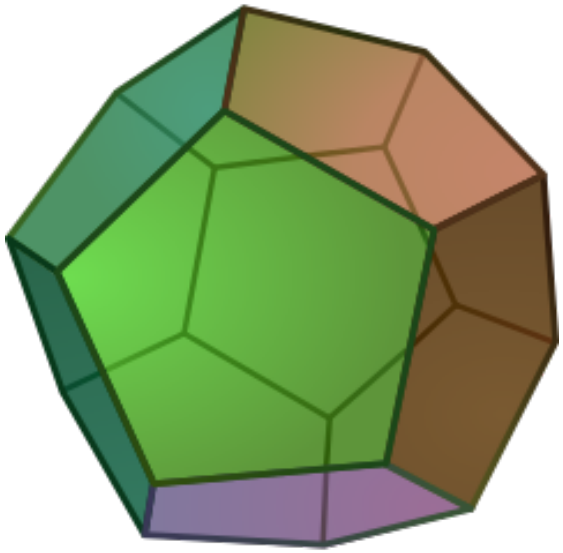
- Vimos vários problemas e algoritmos eficientes para resolve-los
  - Ordenação - $O(n \log n)$
  - Multiplicação - $O(n^{1.586})$
  - Multiplicação de Matrizes - $O(n^{2.8})$
  - Busca em Grafos - $O(m+n)$
  - Encontrar Componentes fortemente conexas - $O(m+n)$
  - Caminhos Mínimos - $O(m \log n)$
  - Escalonamento Ponderado - $O(n \log n)$
  - Compressão de Texto - $O(n \log n)$
  - MST - $O(m \log n)$
- Será que qualquer problema pode sempre ser resolvido em tempo $O(n^3)$, ou $O(n^4)$?
- Ou pelo menos será que qualquer problema pode ser resolvido em tempo $O(n^k)$ para qualquer k constante? Todo problema pode ser resolvido em **tempo polinomial**?
- A resposta é: **Muito provavelmente não!**

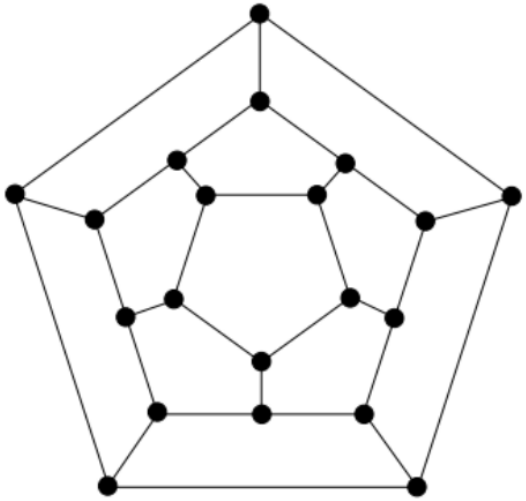
- Quando falamos de Caminhos entre todos os pares de vértices, em grafos com ciclos de pesos negativos, se proibíssemos os ciclos o problema era bem definido, mas não podia ser resolvido de maneira eficiente.
- De fato o problema da mochila que resolvemos em $O(nW)$ também não foi resolvido em tempo polinomial no tamanho da entrada. Uma vez que para escrever W precisamos $m = \log W$ bits. Portanto o tempo de execução é $O(n2^m)$, exponencial no tamanho da entrada!
- Esses dois problemas estão entre os chamados NP-Difíceis! Para os quais ainda não se conhecem algoritmos de tempo polinomial!
- De fato até aqui (exceto pelo problema da mochila) foi selecionado um conjunto muito particular de problemas, aqueles que podem ser resolvidos em tempo polinomial. Mas **MUITOS** outros problemas práticos talvez não possam.

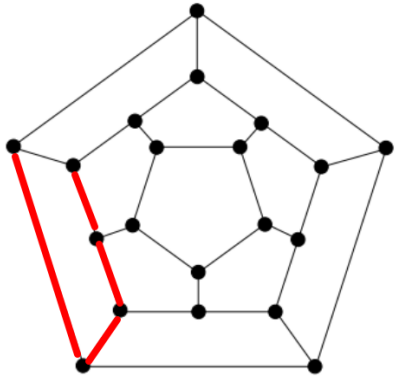
- Formalmente as classes de complexidade P, NP, NP-completo e outras, são melhor definidas sobre os problemas de decisão, para os quais a resposta é simplesmente **SIM** ou **NÃO**. Mas os problemas tem uma forte relação entre si.
  - Caminho mínimo: Dado G, um vértice s e um número k existe um caminho de s para qualquer outro vértice com custo no máximo k?
  - MST: Dado G e um inteiro k, existe uma Árvore geradora mínima de custo no máximo k?
  - Compressão de Texto: Dado um texto T e um número k, existe uma compressão desse texto com no máximo k bits?
- O problemas que podem ser decididos em tempo polinomial formam a classe de complexidade

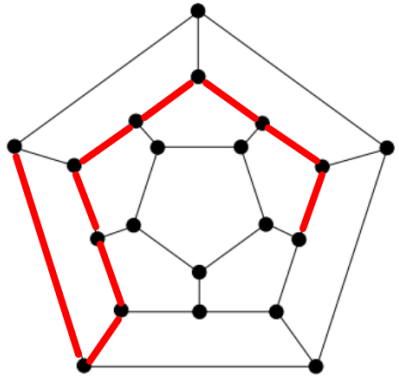
$$P$$

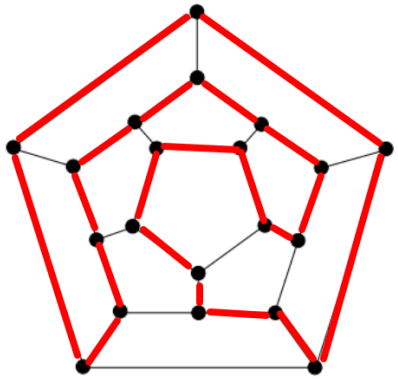
Sir William Rowan Hamilton (1805-1865)

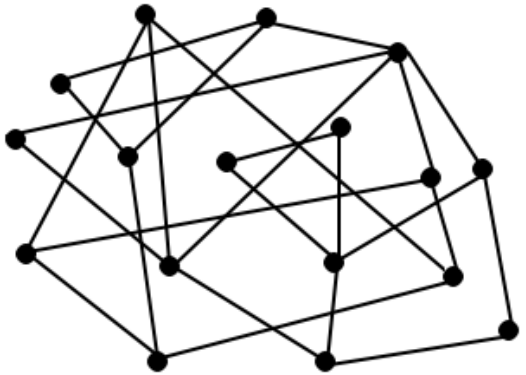


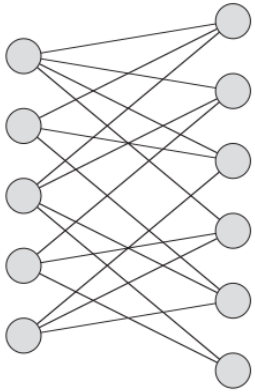












- Um **ciclo hamiltoniano** de um grafo G é um ciclo que passa por todos os vértices exatamente uma vez.

Problema do Ciclo Hamiltoniano

Dado um grafo não orientado $G = (V, E)$. Decidir se G possui um ciclo hamiltoniano.

- Esse problema é NP-Difícil e não se conhece nenhum algoritmo de tempo polinomial para resolve-lo!

Mas são tão parecidos...

Dado um grafo orientado $G = (V, E)$,

encontrar o caminho mais **curto** entre dois vértices é fácil, $O(VE)$.

Encontrar o caminho mais **longo**, é NP-Difícil.

Mas são tão parecidos...

Dado um grafo orientado $G(V, E)$,

decidir se existe um ciclo que passe por todas as **arestas** exatamente uma vez é fácil, $O(E)$.

Decidir se existe um ciclo que passe por todos os **vértices** exatamente uma vez é NP-Difícil.

Mas são tão parecidos...

Uma formula booleana, contêm variáveis cujo valor são 0 ou 1, conectivos $\wedge$ (e), $\vee$ (ou) e $\neg$ (negação); e parenteses.

$$(x_1 \wedge x_2 \vee x_3) \wedge (x_1 \vee \neg x_3) \vee x_4$$

Existem uma atribuição das variáveis que torne a formula verdadeira?

Mas são tão parecidos...

Decidir se uma formula em **forma normal 2-conjuntiva** é satisfazível, por exemplo:

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)$$

é fácil.

Decidir se uma formula em **forma normal 3-conjuntiva** é satisfazível, é NP-Difícil.

O nome desse problema é **3-CNF-SAT**

P, NP e NP-Completo

- Os problemas da Classe **P** são aqueles que podem ser resolvidos em tempo polinomial
- Os problemas da Classe **NP** que conseguem decidir SIM em tempo polinomial para uma instância x se um certificado C_x for fornecido.

Classe NP

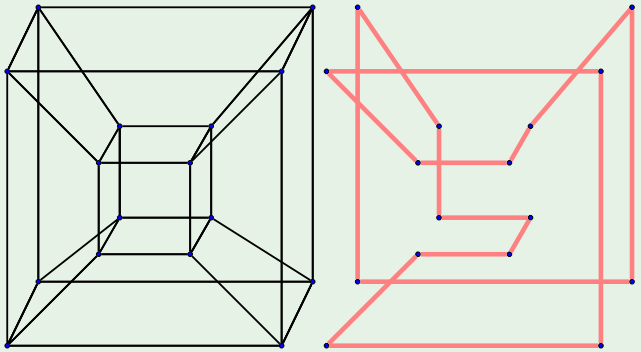
2-CNF-SAT

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)$$

$x_1 = 1$, $x_2 = 0$ e $x_3 = 0$

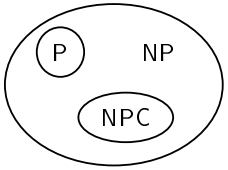
Classe NP

Ciclo Hamiltoniano



Classe NP

- Obviamente todo problema em P está em NP, basta descartar o certificado e decidir baseado apenas na instância.
- O certificado tem que ter tamanho polinomial, senão eu gastaria um tempo exponencial só para lê-lo.



Reduções

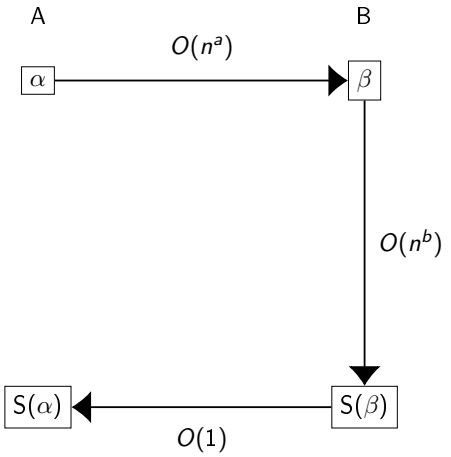
Suponha que temos um problema de decisão A que queremos resolver em tempo polinomial.

Agora suponha que temos outro problema de decisão B que já sabemos que pode ser resolvido em tempo polinomial.

Finalmente, suponha que conhecemos um procedimento que transforma uma instância α do problema A, em uma instância β no problema B. Tal que:

- A transformação leva tempo polinomial
- A resposta de α para A é a mesma da instância β para B.

Reduções



Reduções

- Temos então um algoritmo polinomial para resolver o problema A.