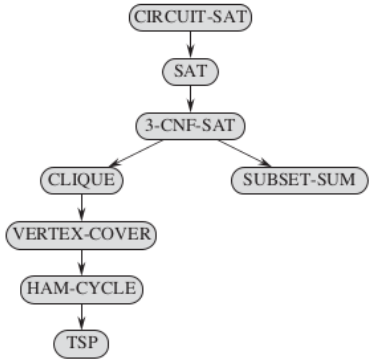
Algoritmos

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- 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
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

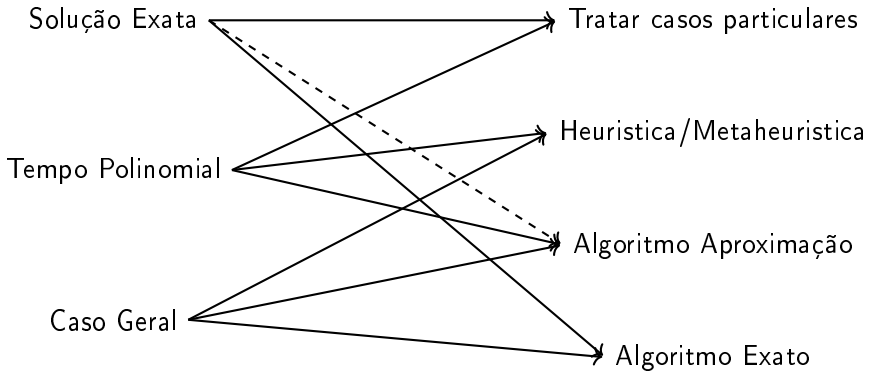


NP-Difícil

- Quando nos deparamos com um problema NP-Completo ou NP-Difícil é provável que não consigamos encontrar uma solução exata em tempo polinomial.

O que fazer então?

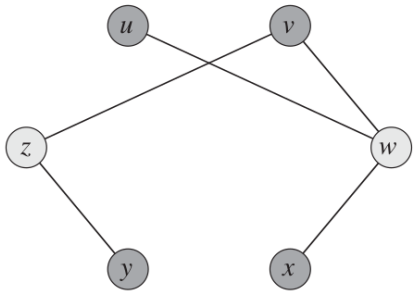
Se $P \neq NP$ não conseguiremos para um problema NP-Difícil:



Cobertura por Vértices

VERTEX-COVER

Dado um grafo não orientado $G = (V, E)$ com n vértice e m arestas, e um inteiro k decidir se existe uma cobertura por vértices $V' \subseteq V$ de tamanho k.



Qual é o tamanho de uma cobertura por vértices de tamanho mínimo em um grafo estrela com n vértices e em um grafo clique de tamanho n.

a) 1 e $n-1$
b) 1 e n
c) 2 e $n-1$
d) $n-1$ e n

- Uma solução força bruta:
- Considerando que k seja pequeno em relação a n podemos tentar todos os conjuntos com k vértices.
- São $\binom{n}{k}$ conjuntos.
- Se $k <<< n$ então a complexidade do algoritmo é $\approx \Theta(n^k)$.
- Podemos fazer melhor?

Teorema

Dada uma aresta (u, v),
$G_u = G$ deletando u e todas as suas arestas adjacentes, e
$G_v = G$ deletando v e todas as suas arestas adjacentes.

G tem uma cobertura de tamanho k, se e somente se
G_u ou G_v tem uma cobertura de tamanho $k - 1$.

- ($\rightarrow$) Suponha que G tem uma cobertura V' de tamanho k. Como a aresta (u, v) precisa estar coberta, pelo menos um dos seus extremos tem que estar em V'.
- Sem perda de generalidade, suponha que $u \in V'$.
- O vértice u cobre apenas as arestas adjacentes a u, e portanto todas as demais arestas devem estar cobertas pelos $k - 1$ vértices restantes de V', e portanto
- $V \backslash \{u\}$ é uma cobertura para G_u e tem $k - 1$ vértices
- ($\leftarrow$) Exercício.

Algoritmo 1: BuscaCobertura

Entrada: Um grafo $G(V, E)$ e um inteiro k
Saída: Verdadeiro se G contêm uma cobertura de tamanho k

```
1 se |E| > 0 e k == 0 então devolve Falso
2 se |E| == 0 então devolve Verdadeiro
3 Escolhe uma aresta qualquer (u, v) ∈ E
4 Cria G_u = G sem u e suas arestas adjacentes
5 Cria G_v = G sem v e suas arestas adjacentes
6 se BuscaCobertura(G_u, k − 1) então devolve Verdadeiro
7 se BuscaCobertura(G_v, k − 1) então devolve Verdadeiro
8 devolve Falso
```


Complexidade de BuscaCobertura:

- A cada chamada recursiva, fazemos outras 2.
- A profundidade da recursão é no máximo k.
- Portanto o número de chamadas recursivas é no máximo 2^n.
- Cada chamada recursiva leva tempo $O(m)$ para criar G_u e G_v.
- Portanto a complexidade total é $O(2^n m)$, portanto exponencial. (Claro que é).
- Ainda muito melhor que o $\Theta(n^k)$ da força bruta.

Algoritmo Exato para o TSP

- Dado um grafo não-direcionado $G = (V, E)$, encontrar o custo mínimo de ciclo que visita exatamente uma vez todos os vértice de V.
- Uma solução força bruta:
- Testar todos os $n!$ percursos possíveis.
- Resolve 12, 13, talvez 14 vértices.
- Podemos fazer melhor? Vamos tentar fazer um algoritmo de Programação Dinâmica!

Tentativa 1

- Vamos entender o ciclo como um caminho que começa no vértice 1 vai até algum vértice j e depois viaja direto de j para 1, fechando o ciclo.
- Podemos usar a ideia do algoritmo de Bellman-Ford, para encontrar o caminho de 1 até j que usa no máximo i arestas.
- Para todo $i \in \{0, 1, \ldots, n\}$ e todo destino $j \in \{1, 2, ..., n\}$, temos L_{ij} o custo de um caminho mínimo de 1 até j que usa no máximo i arestas.

- Podemos então pegar todos os $L_{n-1,j}$ e somar com o custo de c_{j1}.
- Isso vai formar um ciclo de caixeiro viajante? Infelizmente não!
- O Bellman-Ford apenas indica o máximo de arestas que podem ser usadas.

Tentativa 2

- Podemos modificar um pouco a ideia do algoritmo de Bellman-Ford, para encontrar o caminho de 1 até j que usa **EXATAMENTE** i arestas.
- Para todo $i \in \{0, 1, \ldots, n\}$ e todo destino $j \in \{1, 2, ..., n\}$, temos L_{ij} o custo de um caminho mínimo de 1 até j que usa **exatamente** i arestas.

- Novamente, podemos então pegar todos os $L_{n-1,j}$ e somar com o custo de c_{j1}. Certo?
- Infelizmente também não. Note que o caminho mínimo de $n-2$ arestas até um vértice k pode passar por j,
- Dessa forma estaríamos repetindo vértices (e deixando alguns de fora).

Tentativa 3

- Podemos tentar modificar mais um pouco a ideia para encontrar o caminho de 1 até j que usa no **exatamente** i arestas e **NÃO REPETE** vértices.
- Para todo $i \in \{0, 1, \ldots, n\}$ e todo destino $j \in \{1, 2, ..., n\}$, temos L_{ij} o custo de um caminho mínimo de 1 até j que usa **exatamente** i arestas e **não repete** vértices.

- Note que ainda o caminho mínimo de $n-2$ arestas até um vértice k pode passar por j.
- Como garantir que não vai haver repetições de vértices?
- O único jeito é resolver mais subproblemas, para cada possível conjunto de vértices.

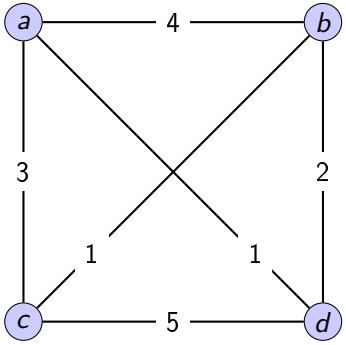
Subestrutura Ótima

- Para todo destino $j \in \{1, 2, ..., n\}$ e todo subconjunto $S \subseteq V$ que contêm 1 e j.
- L_{Sj} é o custo de um caminho mínimo de 1 até j que visita todos os vértices de S (e somente eles).

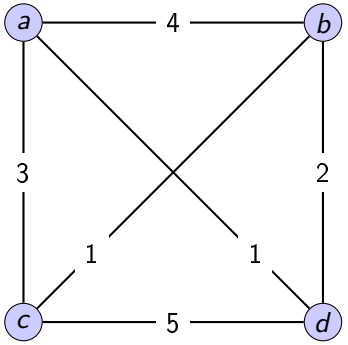
- Considere o seguinte exemplo:
- Queremos descobrir $L_{\{1,3,7,8,9\},7}$ ou seja queremos o caminho de 1 até 7 que visita exatamente uma vez os vértices 1, 3, 7, 8 e 9. Quais opções temos?
- Podemos ir de 1 até 3 visitando $\{1,3,8,9\}$ depois para 7.
- Podemos ir de 1 até 8 visitando $\{1,3,8,9\}$ depois para 7.
- Podemos ir de 1 até 9 visitando $\{1,3,8,9\}$ depois para 7.

Dessa forma podemos escrever a seguinte recorrência:

$$L_{S,j} = \begin{cases} c_{1j}, \text{ se } S = \{1, j\} \\ \min_{k \in S, k \neq j} \left\{ L_{S \setminus \{j\}, k} + c_{kj} \right\} \end{cases}$$

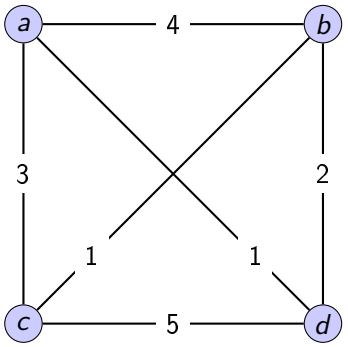


S	b	c	d
$\{a, b\}$	4	-	-
$\{a, c\}$	-	3	-
$\{a, d\}$	-	-	1



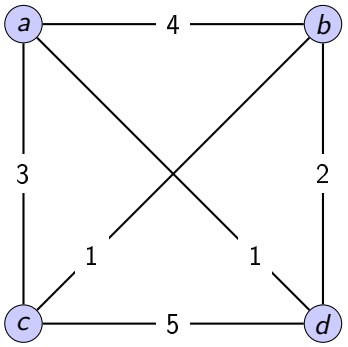
S	b	c	d
$\{a, b\}$	4	-	-
$\{a, c\}$	-	3	-
$\{a, d\}$	-	-	1

S	b	c	d
$\{a, b, c\}$	4	4	-
$\{a, b, d\}$	3	-	6
$\{a, c, d\}$	-	6	8



S	b	c	d
$\{a,b,c\}$	4	4	-
$\{a,b,d\}$	3	-	6
$\{a,c,d\}$	-	6	8

S	b	c	d
$\{a,b,c,d\}$	7	4	6



S	b	c	d
$\{a,b,c,d\}$	7	4	6

Por fim a ultima aresta,
$(b,a) = 7 + 4 = 11$
$(c,a) = 4 + 3 = 7$
$(d,a) = 6 + 1 = 7$

Algoritmo 2: BellmanHeldKarp

Entrada: $G(V,E)$, $V = \{1,2,...,n\}$, custos c_{ij} para $(i,j) \in E$
Saída: Custo mínimo de um percurso de caixeiro viajante em G

```
1 //A[S][j] indica o custo mínimo de chegar em j passando uma vez por cada vértice de S ⊆ V
2 A[2^{n-1} − 1][n − 1];
3 para j = 2 até n faça
4     A[{1, j}][j] = c_{1j};
5 para s = 3 até n faça
6     para todo S com |S| = s e 1 ∈ S faça
7         para j ∈ S\{1} faça
8             A[S][j] = min_{k∈S\{1,j}}(A[S\{j}][k] + c_{kj});
9 devolve min_{j∈S\{1}}(A[V][j] + c_{j1});
```


Complexidade.

- Temos $O(2^n)$ possíveis escolhas de S.
- Para cada S temos $O(n)$ problemas (um para cada membro de S)
- O total de subproblemas é $O(n2^n)$
- Para cada subproblema temos que procurar o mínimo em $O(n)$ subproblemas.
- Dessa forma a complexidade total de BellmanHeldKarp é $O(n^2 2^n)$

Considere que vamos utilizar um computador de 4Ghz

n	$n!$	$n^2 2^n$
10	0 s	0s
15	300 s	0s
18	18 dias	0s
20	19 anos	0,1 s
23	200 milênios	1 s
35	?	3 horas
40	?	5 dias