CLASSES DE COMPLEXIDADE E REDUÇÕES

DCE529 - Algoritmos e Estruturas de Dados III

Atualizado em: 29 de fevereiro de 2024

Iago Carvalho

Departamento de Ciência da Computação

INTRODUÇÃO

De forma ampla, podemos dizer que um problema é *fácil* ou *difícil*

- *Fácil*: tratável, tempo polinomial: $\mathcal{O}\left(p(n)\right)$
- *Difícil*: intratável, tempo exponencial: $\mathcal{O}\left(c^n\right)$, onde $c > 1$

Problemas *fáceis*

- Caminho mínimo
- Árvore geradora mínima
- Componentes conexos
- Programação linear
- Números primos
- ...

Problemas *difíceis*

- Caminho máximo
- Árvore de Steiner
- Satisfabilidade
- Programação inteira
- Caixeiro viajante
- ...

CLASSES DE PROBLEMAS

Problemas *fáceis* podem ser resolvidos em tempo polinomial

- Considera-se que estão na classe P

Problemas *difíceis* não podem ser resolvidos em tempo polinomial

- Tempo exponencial
- Considera-se que estão na classe NP

EXISTEM ALGORITMOS POLINOMIAIS PARA PROBLEMAS EM *NP*?

REDUTIBILIDADE EM PROBLEMAS *NP*

Um problema em *NP* pode ser transformado em outro problema em *NP* em tempo polinomial

- Esta premissa também é válida para problemas em *P*

Caso encontre-se um algoritmo polinomial para um problema em *NP*, então todos os problemas em *NP* poderão ser resolvidos em tempo polinomial

PROBLEMAS DE DECISÃO E A CLASSE *NP*

Problemas de decisão são a base para o estudo de classes de complexidade

Um problema de decisão aceita duas respostas

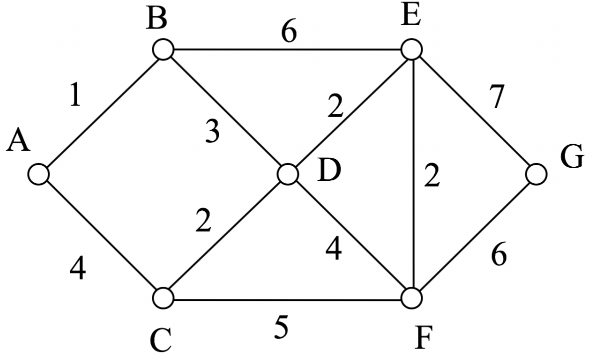
- Sim
- Não

A classe *NP* contém problemas de decisão onde esta pergunta (sim, não) pode ser respondida em tempo polinomial

- Verificar a solução $\neq$ computar a solução

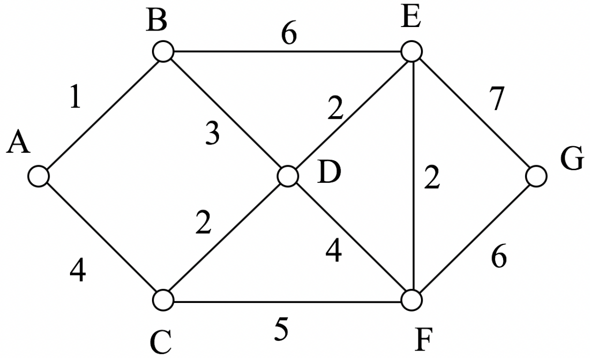
Problema do caminho mínimo: existe um caminho entre A e G com peso menor ou igual a k?

E com peso maior do que k?



Problema do caminho mínimo: existe um caminho entre A e G com peso menor ou igual a k? $\rightarrow$ *Fácil*

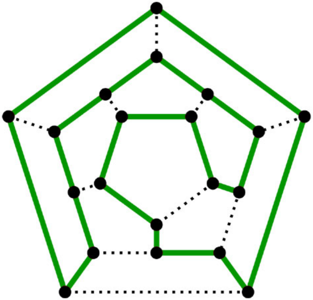
E com peso maior do que k? $\rightarrow$ *Difícil*



Problema do caminho hamiltoniano: existe um caminho que passe por todos os vértices do grafo uma única vez?

Grau dos vértices ≤ 2: $\rightarrow$ *Fácil*

Caso contrário: $\rightarrow$ *Difícil*



DETERMINISMO E NÃO-DETERMINISMO

Um *algoritmo determinista* é aquele em que o resultado de cada operação é definido de forma única

- No mundo real, só existem algoritmos determinísticos
- Todos os algoritmos que vocês já implementaram até hoje são determinísticos

Um *algoritmo* ***não*** *determinista* é capaz de, magicamente, escolher a melhor resposta instantaneamente

- Escolhe dentre um conjunto de respostas possíveis

FUNÇÃO ESCOLHE

Obter o menor número de uma matriz

- Algoritmo determinista: $\mathcal{O}(nm)$
- Algoritmo não-determinista: $\mathcal{O}(1)$
  - Função *escolhe*

83	67	39	85	11	21	87
25	48	74	7	15	74	90
13	10	87	57	3	75	36
19	47	89	48	16	7	81
79	40	68	70	25	59	96

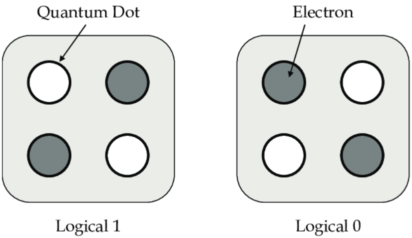
DETERMINISMO E NÃO-DETERMINISMO

Uma máquina de Turing determinística é um dispositivo de computação capaz de rodar somente algoritmos determinísticos

- Processadores atuais

Já uma máquina de Turing não determinística é aquela capaz de rodar algoritmos determinísticos e não-determinísticos

- Computação quântica
- Quantum-Dot Cellular Automata



CARACTERIZAÇÃO DAS CLASSES P E NP

P: Conjunto de problemas que podem ser resolvidos em tempo polinomial por uma máquina de Turing determinística

NP: Conjunto de problemas que podem ser resolvidos em tempo polinomial por uma máquina de Turing não-determinística

CARACTERIZAÇÃO DAS CLASSES P E NP

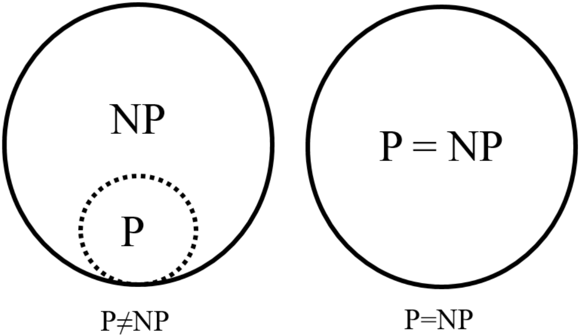
É fácil perceber que $P \subseteq NP$

Entretanto, um dos maiores problemas em computação (e da matemática) é provar

- $P = NP$?
- $P \neq NP$?

Se existem algoritmos polinomiais deterministas para todos os problemas em NP, então $P = NP$

CARACTERIZAÇÃO DAS CLASSES *P* E *NP*



CARACTERIZAÇÃO DAS CLASSES P E NP

Muitos problemas em NP podem ou não pertencer a P

- Não conhecemos algoritmos polinomiais para eles
- Isto não quer dizer que tais algoritmos não existam

Se conseguirmos provar que um problema não pertence a P, então não precisaríamos mais procurar algoritmos eficientes para os problemas em NP

- Ninguém nunca conseguiu provar algo semelhante

Como não existe tal prova, então existe a esperança de que $P = NP$

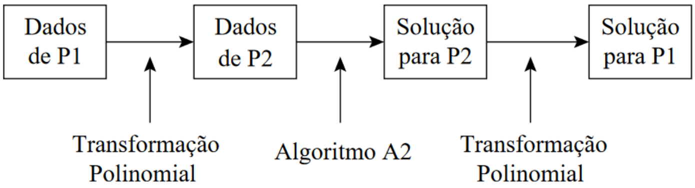
TRANSFORMAÇÃO POLINOMIAL

Sejam P_1 e P_2 dois problemas de decisão

Suponha que exista um algoritmo A_2 que resolva o problema P_2

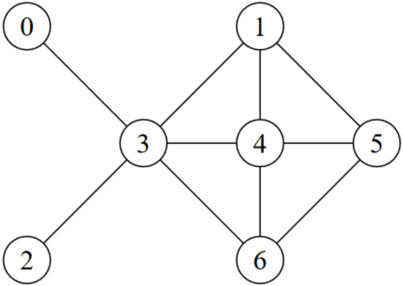
Caso seja possível transformar P_1 em P_2 (e vice-versa), então podemos utilizar o algoritmo A_2 para resolver o problema P_1

- Transformações devem ser polinomiais



Seja $G = (V, E)$ um grafo. O conjunto independente $V' \subseteq V$ é tal que $i, j \in V' \iff (i, j) \notin E$

- V' é um grafo totalmente desconectado
- Todos par de vértices em V' não é adjacente

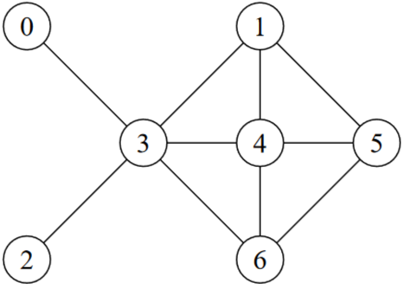


$$V' = \{0, 1, 2, 6\}$$

TRANSFORMAÇÃO POLINOMIAL - CLIQUE

Seja $G = (V, E)$ um grafo. O clique $V' \subseteq V$ é tal que $i, j \in V' \iff (i, j) \in E$

- V' é um subgrafo completo de G
- Todos par de vértices em V' é adjacente



$$V' = \{1, 3, 4\}$$

TRANSFORMAÇÃO POLINOMIAL

Seja P_1 o problema do clique e P_2 o problema do conjunto independente

- Seja $G = (V, E)$ uma instância de P_1
- Seja $\overline{G} = (V, E)$ uma instância de P_2
- É possível transformar G em $\overline{G}$ em tempo polinomial
- É possível transformar $\overline{G}$ em G em tempo polinomial

Mostre que G possui um clique de tamanho $\geq k$ se e somente se $\overline{G}$ possui um conjunto independente de tamanho $\geq k$

TRANSFORMAÇÃO POLINOMIAL

Se existe um algoritmo que resolve o conjunto independente em tempo polinomial, ele pode ser utilizado para resolver clique também em tempo polinomial

Diz-se que clique $\propto$ conjunto independente

- $P_1 \propto P_2$ indica que P_1 é polinomialmente transformável em P_2

Esta relação é transitiva

- $P_1 \propto P_2$ e $P_2 \propto P_3$, então $P_1 \propto P_3$

Dois problemas P_1 e P_2 são polinomialmente equivalentes se e somente se $P_1 \propto P_2$ e $P_2 \propto P_1$

CLASSE *NP*-COMPLETO

Um problema de decisão P_1 é dito ser *NP*-completo se

1. $P_1 \in NP$
2. Para todo problema $P' \in NP$-Completo, temos que $P' \propto P_1$

Este *framework* pode ser utilizado para provar que um problema é *NP*-Completo

- São os problemas *difíceis*
- Complexidade $O(c^n)$, onde
  - $c > 1$
  - n é o tamanho da entrada

COMO RESOLVER PROBLEMAS NP-COMPLETOS

Usar algoritmos exponenciais *eficientes*

- Técnicas baseadas em podas
- *Branch-and-bound*

Utilizar heurísticas, meta-heurísticas ou algoritmos aproximativos

- Último assunto de nossa disciplina