Teoria da complexidade

→ Classifica os problemas conforme sua complexidade de tempo.

→ Qual modelo usar?
↳ Como vimos, $A = \{a^k b^k : k \geq 0\}$ tem tempos diferentes em modelos diferentes.

→ Definimos TIME $(t(n))$ como sendo a coleção de todas as linguagens que são decidíveis por uma máquina de Turing determinística de tempo $O(t(n))$.
↳ $A \in \text{TIME}(n^2)$, $A \in \text{TIME}(n)$

→ Felizmente, os modelos determinísticos razoáveis são todos polinomialmente equivalentes.

A classe P

DEFINIÇÃO: P é a classe de linguagens que são decidíveis em tempo polinomial sobre uma máquina de Turing determinística de uma fita. Em outras palavras,

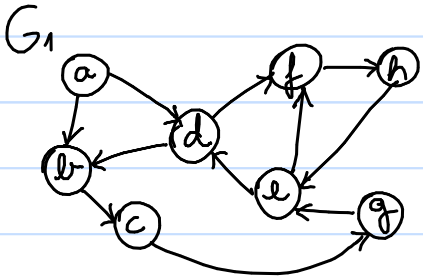
$$P = \bigcup_k \text{TIME}(n^k).$$

Analisar um algoritmo para mostrar que ele leva tempo polinomial requer examinar cada estágio para assegurar que ele pode ser implementado em tempo polinomial em um modelo determinístico razoável.
Isso inclui também um método de codificação razoável.

Exemplos

CAMINHO $= \{ \langle G, u, v \rangle$: G é um digrafo que tem caminho entre u e v $\}$.

G_1



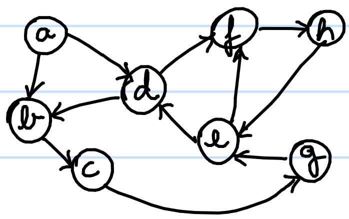
Representações razoáveis para grafos:
- lista dos vértices e arestas
- matriz de adjacências
- lista de adjacências

$\langle G_1, a, g \rangle$
$\langle G_1, c, h \rangle$
$\langle G_1, g, a \rangle$

→ CAMINHO $\in$ P

Basta mostrar um algoritmo de tempo polinomial que decide CAMINHO.

M = " Sobre a entrada $\langle G, u, v \rangle$:
(1) Marque u. $O(n+m)$
(2) Repita enquanto houver vértices a marcar: $O(n)$
Se há um arco xy em G com x marcado e y não marcado, marque y. $O(m+m+m) = O(n+m)$ $\Big\}\ O(n^2)$
(3) Se v estiver marcado, aceite.
Caso contrário, rejeite." $O(n)$



tempo $O(n^2)$
na representação lista de vértices e arestas

A classe P

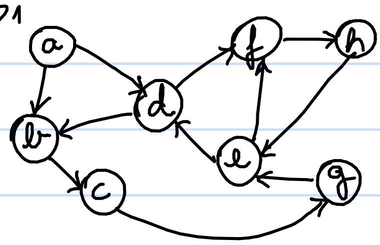
→ Todos os problemas estão em P?
↳ os indecidíveis não estão
↳ mas alguns decidíveis também (ainda) não.

→ Por que nem todos os problemas estão em P?
↳ Talvez seus algoritmos de tempo polinomial ainda não foram encontrados...
↳ Talvez eles não podem estar mesmo...

Problemas que não estão em P

CAMINHOHAM = { ⟨G, u, v⟩ : G é um digrafo e existe caminho hamiltoniano de u a v }.

G_1



É bem mais fácil convencer que ⟨G, u, v⟩ ∈ CAMINHOHAM do que ⟨G, u, v⟩ ∉ CAMINHOHAM.

$\langle G_1, a, g \rangle$ $\langle G_1, c, h \rangle$ $\langle G_1, a, h \rangle$

A classe NP

DEFINIÇÃO: Um verificador para uma linguagem A é um algoritmo V tal que

$$A = \{w : V \text{ aceita } \langle w, c\rangle \text{ para alguma cadeia } c\}.$$

V tem tempo polinomial se roda em tempo polinomial sobre $|w|$ (independe de c).

c é chamado certificado (positivo).

DEFINIÇÃO: NP é a classe das linguagens que têm verificadores em tempo polinomial.

Exemplos

CAMINHOHAM $\in$ NP

Basta mostrar que $\langle G, u, v\rangle \in$ CAMINHOHAM tem um certificado que possa ser verificado em tempo polinomial.

↳ Dada uma sequência de vértices, precisamos verificar se (i) é caminho, (ii) é de u a v, (iii) é Hamiltoniano. Para (i), basta que cada vértice da sequência tenha arco para o próximo. São $O(n)$ vértices na sequência e cada um pede tempo $O(m)$ para verificar se o arco existe $\therefore O(n+m)$. Para (ii) e (iii), basta verificar se a sequência começa em u, termina em v e tem todos os n vértices $\therefore$ tempo $O(n)$.

O certificado é a sequência de vértices.

CAMINHO $\in$ NP

Para qualquer $\langle G, u, v\rangle \in$ CAMINHO, uma sequência de vértices que forma um caminho de u a v. Leva tempo $O(n+m)$ verificá-la, como discutido acima.

CLIQUE = $\{ \langle G, k \rangle :$ G é um grafo com uma clique de tamanho $k \}$

CLIQUE $\in$ NP

Um conjunto de vértices é um certificado: basta verificar se tem k vértices e se cada um deles é vizinho de todos os outros. Como $k = O(n)$, esse teste pode ser feito em tempo $O(nm)$.

SUBSETSUM = $\{ \langle S, t \rangle : S = \{x_1, \dots, x_k\}$, cada $x_i \in \mathbb{Z}$, e $t \in \mathbb{Z}$ e para algum $S' \subseteq S$ vale que $\sum_{x \in S'} x = t \}$

SUBSETSUM $\in$ NP

Um subconjunto de S é um certificado: basta somá-los e verificar se o resultado é t.

A classe NP

TEOREMA: Uma linguagem está em NP se e somente se ela é decidida por alguma máquina de Turing não-determinística em tempo polinomial.

Se $A \in NP$, existe verificador V de tempo n^k:

N = "Sobre w, $|w| = n$:

(1) não-deterministicamente selecione cadeia c, $|c| \leq n^k$

(2) Rode V sobre $\langle w, c \rangle$

(3) Se V aceita, aceite.

(4) Se nenhum aceitou, rejeite."

Se A é decidível por uma MND N em tempo n^k:

V = "Sobre $\langle w, c \rangle$:

(1) Simule N sobre w, tratando cada símbolo de c como uma descrição da escolha não-determinística a fazer.

(2) Se esse ramo aceita, aceite. Caso contrário, rejeite."

→ Definimos $NTIME(t(n))$ como sendo a coleção de todas as linguagens que são decidíveis por uma máquina de Turing não-determinística de tempo $O(t(n))$.

↳ Assim, $NP = \bigcup_{k} NTIME(n^k)$.

P vs. NP

→ Claramente, $P \subseteq NP$ (Por quê?)

→ Será que $NP \subseteq P$?

↳ Será que $P = NP$?

Será que decidir pertinência rapidamente tem o mesmo nível de dificuldade que verificar pertinência rapidamente?

→ Deterministicamente, até agora, só conseguimos resolver as linguagens em NP em tempo exponencial.

$$NP \subseteq EXPTIME = \bigcup_{k} TIME(2^k)$$

mas não sabemos se NP está em uma classe menor.

Redução também é útil aqui!

DEFINIÇÃO: A linguagem A é redutível à linguagem B se existe uma função computável $f: \Sigma^* \to \Sigma^*$ tal que $\forall w$

$$w \in A \text{ se e somente se } f(w) \in B.$$

Se A é redutível a B em tempo polinomial, o que ganhamos?

$$I_A \xrightarrow{f} I_B \longrightarrow \boxed{M_B} \longrightarrow \text{aceita / rejeita}$$

Se $B \in P$, então $A \in P$

↳ Se $A \notin P$, então $B \notin P$