COMPLEXIDADE E COMPUTABILIDADE

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Introdução

Problemas tratáveis vs. intratáveis

- Tratáveis: se conhece algoritmos de custo polinomial
- Intratáveis: não se conhece algoritmos de custo polinomial

Problemas intratáveis

- Escalonamento de processos
- Gerenciamento de memória (fragmentação)
- Circuito Hamiltoniano
- Caixeiro viajante (TSP)

Introdução

Relembrando...

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

[1]

Considerando: 10^{12} (1 trilhão) operações por segundo

- 2^{100} operações $\approx 4 \cdot 10^{10}$ anos

[1] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Introdução

Problemas de decisão: solução consiste em sim ou não

- Seja $G = (V, E)$, $x_1, x_2, ..., x_k \in V$, $C \in \mathbb{N}$, existe um caminho que passa por todos estes vértices, cujo custo seja no máximo C?

Se um problema P não for de decisão (e.g., qual o custo do menor caminho passando pelos vértices dados?), existe um problema de decisão P' que ajuda a resolver P em tempo igual ao tempo de resolver o problema P', a menos de um fator polinomial.

Agenda

NP-Completo

Problemas intratáveis pertencem à classe: NP-Completo

- NP-Completo = NP $\cap$ NP-Difícil

Classe NP: classe de problemas para os quais existe um algoritmo de custo polinomial, porém que não é determinístico (*non-deterministic polynomial time*)

- Inclui os problemas polinomiais

Linguagem de programação não-determinística

- Todos os comandos de uma linguagem de programação tradicional, mais um salto-nd
- Os problemas com algoritmos polinomiais também pertencem à classe NP, só não fazem uso do salto-nd

NP-Completo

Algoritmo não-determinístico para um problema de decisão

- Sim: existe pelo menos uma maneira de fazer escolhas de execução nos *saltos-nd* de forma que a resposta do algoritmo seja sim
- Não: não existe maneira de fazer escolhas que levem a uma resposta sim (i.e., todas combinações levam a um não)

Problema clique

- Seja $G = (V, E)$ um grafo não-direcionado e sem peso nas arestas e $k \in \mathbb{N}$ tal que $k \leq |V|$, existe um subgrafo completo de G com k vértices?

NP-Completo

Algoritmo: bool clique(Graph G, int k)

```
for i ← 0 to n(G) − 1 do
    salto-nd{chosen[i] ← true | chosen[i] ← false};
for v ← 0 to n(G) − 1 do
    if chosen[v] then
        cont ← 0;
        w ← first(G, v);
        while w < n(G) do
            if chosen[w] then cont++;
            w ← next(G, v, w);
        if cont ≠ k − 1 then return false;
return true;
```

Custo (com salto-nd): $\Theta(|\ V\ |) + \Theta(|\ V\ | + |\ E\ |) = \Theta(|\ V\ |^2)$
Custo (sem salto-nd): $\Theta(2^{|V|}) + \Theta(|\ V\ | + |\ E\ |) = \Theta(2^{|V|})$

NP-Completo

NP-Difíceis: todos os problemas com complexidade maior ou igual ao problema SATisfatibilidade

- SAT: seja α uma expressão booleana na forma conjuntiva normal (FCN, ou CNF em inglês – uma conjunção de disjunções)), existe uma valoração das variáveis de α que torna α verdadeira?
- Exemplo: $\alpha = (x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$, neste caso a resposta seria sim ($x = \mathit{false}, y = \mathit{false}, z = \mathit{false}$)

NP-Completo

Z3: SMT solver desenvolvido pela Microsoft

z3 Microsoft Research

Is this formula satisfiable?

```
(declare-fun x () Bool)
(declare-fun y () Bool)
(declare-fun z () Bool)
(assert (and (and (or (or x y) (not z)) (or (or (not x) y) z)) (or (or (not x) (not y)) (not z))))
(check-sat)
(get-model)
(exit)
```

► tutorial home video permalink
'►' shortcut: Alt+B

```
sat
(model
  (define-fun z () Bool
    false)
  (define-fun y () Bool
    false)
  (define-fun x () Bool
    false)
)
```

[2]

Comando: `./z3 -smt2 sat_ex`

[2] Link: https://rise4fun.com/z3

NP-Completo

Redução polinomial (RP): é um algoritmo polinomial que transforma uma instância de x de um problema P em uma instância y de um problema P' de forma que: $P(x) = sim \Leftrightarrow P'(y) = sim$.

- Seja x uma instância de SAT, $RP(x) = y$, onde y é uma instância de P' tal que $SAT(x) = sim \Leftrightarrow P'(y) = sim$.
- Se P' tem complexidade C, SAT pode ser resolvido com complexidade C + o custo (polinomial) de RP

NP-Completo

Redução polinomial de SAT à clique

- Seja $\alpha = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, construa o grafo $G = (V, E)$, onde $V \leftarrow \{v_{i,j}, \text{onde } i \text{ é a cláusula e } j \text{ é a variável}\}$, e $E \leftarrow \{(v_{i,j}, v_{k,l}), \text{onde } i \neq k \text{ e } v_{i,j} \neq \neg\, v_{k,l}\}$
  - $\mid V \mid <$ quantidade de símbolos em α, $\mid E \mid \leq \mid V \mid^2$

Exemplo: $\alpha = (x \vee y \vee \neg\, z) \wedge (\neg\, x \vee y \vee z) \wedge (\neg\, x \vee \neg\, y \vee \neg\, z)$

NP-Completo

Se $SAT(\alpha) = sim \Rightarrow clique(RP(\alpha), m) = sim$

- Se $SAT(\alpha) = sim$, como α está na CNF, há pelo menos um literal em cada cláusula com valor verdadeiro. O subgrafo formado a partir destes literais é um $m - clique$.

Se $clique(RP(\alpha), m) = sim \Rightarrow SAT(\alpha) = sim$

- Pela definição de *RP*, cada vértice corresponde a um literal de uma cláusula diferente e as valorações não podem se contradizer. Se $clique(RP(\alpha), m) = sim$, então é possível valorar as variáveis correspondentes a estes vértices de forma a tornar α satisfatível.

NP-Completo

Como resolver problemas NP-Completos?

- Backtracking
- Branch and bound
- Algoritmos de aproximação
- Programação dinâmica
- Heurísticas

Agenda

Computabilidade

Existem problemas tratáveis e intratáveis

Existem problemas decidíveis e indecidíveis

- Problema da parada: decidir se um programa termina de executar para uma entrada finita ou se ele nunca terminará de executar.
- Entscheidungsproblem: decidir se uma afirmação em lógica de primeira ordem é universalmente válida.
    - SAT solvers
    - SMT solvers (Z3: sat, unsat e unknown)

Existem problemas que não são computáveis

- Existem infinitos programas (enumeráveis – $\aleph_0$)
- Existem infinitas funções matemáticas (não-enumeráveis)
- Nem toda função matemática pode ser computada

Computabilidade

O problema da parada: suponha que exista um algoritmo A:

- $A(P, I) = 1$, se P termina sobre I
- $A(P, I) = 0$, se P não termina sobre I

Seja Q:

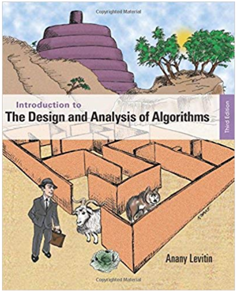
- $Q(P)$ termina se $A(P, P) = 0$
- $Q(P)$ não termina se $A(P, P) = 1$

Substituindo Q por P (contradição!):

- $Q(Q)$ termina se $A(Q, Q)$ não termina
- $Q(Q)$ não termina se $A(Q, Q)$ termina

Agenda

Bibliografia + leitura recomendada



Capítulo 11 (pp. 401–409)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

COMPLEXIDADE E COMPUTABILIDADE

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil