Projeto e Análise de Algoritmos - PAA

Cristiano Vasconcellos

06/08/2019 "

Apresentação do ~~contexto histórico~~ limite da computação

↳ Solução: Completa & Consistente & Decidível

Teorema da Incompletude de Gödel

Formalização de Algoritmo: (existem problemas não indutivos computacionalmente)
* Cálculo λ
* Máquina de Turing
* Funções Recursivas Parciais

PROBLEMA INDECIDÍVEL: Problema da Parada (HALT)

Ementa: Aprender como analisar um Algoritmo
Aprender as classes de Problemas

" 08/08/2019 "

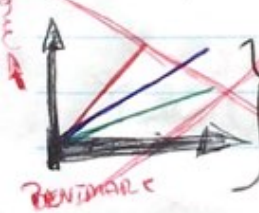
Transparência: PAA1.pdf (do 0 até o número 14) Senha Moodle: PAA2019

Bibliografia: Algoritmos de Thomas Cormen
ou
Sanjay Dasgupta

PROVA (25%) PROVA (25%) TRABALHO (25%) PROJETO (25%)

→ complexidade (MELHOR, MÉDIO, PIOR)

Comparações por: ~~TEMPO~~ BENCHMARK



método bom: dizer que a complexidade é linear

matematicamente ignora as constantes além disso $O(n^2) \cdot O(n^2) = O(n^2)$

$O(m/2)$ ou $O(m+1)$ nunca usar constantes somando ou multiplicando

* é errado preferir isso

13/08/2019

$0 < \varepsilon < 1 < c$

$1 < \log\log n < \log n < n^{\varepsilon} < n^{c} < n^{\log n} < c^{n} < n^{n} < c^{c^{n}}$

problemas considerados Intratáveis

Transparência: PAA1.pdf (de 14 até 16)

Ordenação por Inserção
* melhor caso: ele não faz nem deslocamento $O(n)$
* pior caso: ele faz todos os deslocamentos $O\left(\frac{n(n-1)}{2}\right)$

função de domínio assintótico: $O(n^2)$

a linguagem C tem controle total do Heap alloc

Ordenação Bolha (Bubble Sort)
* melhor caso: não entra sempre no if $O(n^2)$
* pior caso: entra sempre no if $O(n^2)$

→ com alterações no código (flag) pode ser $O(n)$ melhor

→ nota: por mais que a complexidade seja igual é importante deixar claro que durante a execução no pior e no melhor caso, existem diferenças no tempo de execução. (melhor executa mais rápido)

Fatorial (recursivo) — Tempo $O(n)$ — Espaço $O(n)$ → Recursão tem custo de memória
(não é recursão de cauda)
↳ multiplicação tem tempo constante
↳ não existe "melhor" ou "pior" caso

Fatorial (iterativo) — Tempo $O(n)$ — Espaço $O(1)$

* pilha de chamada
* pilha de controle

(slide 16)
fim

15/08/2019

Transparência: PAA.pdf (de 16 até 21)

Recursivo — recursão de cauda

Pesquisa binária * cada vez que o valor dobra, a quantidade de chamadas aumenta em 1

complexidade de tempo de execução — tempo de execução da parte recursiva — tempo de execução da parte não-recursiva

$$T(n) \leq T(n/2) + O(1)$$

parte recursiva

$$T(1) = O(1)$$

para 1 elemento

Forma genérica:

$$\begin{cases} T(8) = T(4) + 1 & T(8) = 4 \\ T(4) = T(2) + 1 & T(4) = 3 \\ T(2) = T(1) + 1 & T(2) = 2 \\ \quad\quad 1 \end{cases} \equiv \begin{cases} T(n) = T(n/2) + 1 & n/2^0 \\ T(n/2) = T(n/4) + 1 & n/2^1 \\ T(n/4) = T(n/8) + 1 & n/2^2 \end{cases}$$

$$\vdots$$

$$T(n/2^{x-1}) = T(n/2^x) + 1$$

$$T(8) = \log_2 8 + 1$$
$$T(2) = 3 + 1$$
$$T(8) = 4 \text{ "}$$

$$\begin{cases} n/2^x = 1 \\ n = 2^x \\ x = \log_2 n \end{cases}$$

tempo: $O(\log n)$ — notação assintótica, ignora as constantes

espaço usado durante a execução do problema e não os dados de entrada carregados

$$\log_4 16 = \frac{\log_2 16}{\log_2 4}$$

$$\log_8 n = \frac{\log_2 n}{\log_2 8}$$

Iterativo

Pesquisa binária

tempo $O(\log n)$

espaço $O(1)$

20/08/2019

Transparência: PAA1.pdf (de 21 a 23)

$T(1) = 1$

algoritmo de divisão e conquista

Merge Sort

espaço

↳ recursivo: $O(\log n)$ } pilha

↳ iterativo: $O(n)$ } função merge

$O(n)$

merges — tamanho vetor (quantidade)

$T(8) = 2T(4) + 8$

$2T(4) = 2^2 T(2) + 2^2 \cdot 4$

$2^2 T(2) = 2^3 \underbrace{2T(1)}_{1} + 2^3 \cdot 2$

forma genérica

$T(n) = 2 \cdot T(n/2) + n$

$2T(n/2) = 2^2 T(n/2^2) + n$

$2^2 T(n/2^2) = 2^3 T(n/2^3) + n$

$\vdots$

$2^{x-1} T(n/2^{x-1}) = 2^x T(n/2^x) + n$

formalização

$n/2^x = 1$

$n = 2^x$

$x = \log_2 n$

Teorema Mestre

$T(n) = n \cdot 1 + \frac{1}{n} (\log_2 n)$

$T(n) = n + n \log_2 n$

↳ alguns casos

$O(n \log n)$

↳ melhor e pior caso

$T(n) = 2^x T(1) + n + \dots + n + n + n + n$

$T(n) = 2^{\log_2 n} + x \cdot n$

$T(n) = n + n \log_2 n$

03/09/2019

Transparência: PAA pdf (de 23 até 37)

trocar pivô próprio vetor (sem memória auxiliar)

Quick Sort

↳ parte não recursiva: $O(n)$

* relação de recorrência

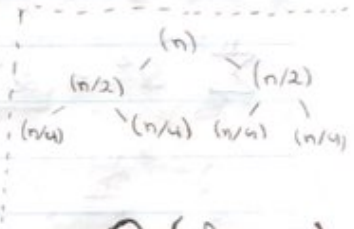
$T(n) = T(1) + T(n-1) + O(n)$

$T(n) = 2T(n/2) + O(n)$ = Merge Sort

$T(8) = 8 + 8 + 8 + 8 = 32$

$T(16) = 80$

↳ $T(16) = 16 + 16 \log_2 16$



$O(n \log n)$ ← complexidade de tempo

MELHOR CASO $O(\log n)$ espaço

* $T(8) = T(7) + 8$
$T(7) = T(6) + 7$
$T(6) = T(5) + 6$
$T(5) = T(4) + 5$
$T(4) = T(3) + 4$
$T(3) = T(2) + 3$
$T(2) = T(1) + 2$

$T(8) = 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$

Forma genérica

$T(n) = T(n-1) + n$
$T(n-1) = T(n-2) + n - 1$
$T(n-2) = T(n-3) + n - 2$
$\vdots$
$T(n-(x-1)) = T(n-x) + n - (x-1)$

$T(n) = n + (n-1) + (n-2) + (n-3) + \ldots + 2 + 1$

$T(n) = \frac{n(n+1)}{2}$

$T(n) = \frac{n^2 + n}{2}$

$O(n^2)$ TEMPO — PIOR CASO — $O(n)$ ESPAÇO

relação a direita da árvore é n (1) (n), (2) (n-1), (3) (n-2), (4) (n-3)

E04: 13:50
Rafael Castro
dia 22/10/3

06/09/2019

Transparência: PAA1.pdf (do 29 até o 33)

Somatório:
* Distributiva $c \cdot \sum_{i=0}^{n} i = \sum_{i=0}^{n} ci$
* Associativa
* Comutativa

Técnica da perturbação

$$\sum_{i=0}^{n} 2^i = 2^0 + 2^1 + 2^2 + \dots + 2^{n-1} + 2^n + 2^{n+1}$$

$$\sum_{i=0}^{n} 2^i + 2^{n+1} = 2^0 + \sum_{i=0}^{n} 2^{i+1}$$

$$\sum_{i=0}^{n} 2^i + 2^{n+1} = 1 + \sum_{i=0}^{n} 2 \cdot 2^i$$

$$\sum_{i=0}^{n} 2^i + 2^{n+1} = 1 + 2\sum_{i=0}^{n} 2^i$$

$$2^{n+1} - 1 = 2\sum_{i=0}^{n} 2^i - \sum_{i=0}^{n} 2^i$$

fórmula fechada

$$\boxed{2^{n+1} - 1} = \sum_{i=0}^{n} 2^i$$

FIM

Exercício $\sum_{i=0}^{n} i2^i = 0 \cdot 2^0 + 1 \cdot 2^1 + 2 \cdot 2^2 + \dots + n2^n + (n+1)2^{(n+1)}$

$$\sum_{i=0}^{n} i2^i + (n+1)2^{(n+1)} = 0 \cdot 2^0 + \sum_{i=0}^{n} (i+1)2^{(i+1)}$$

$$\sum_{i=0}^{n} i2^i + (n+1)2^{(n+1)} = \sum_{i=0}^{n} i2^{(i+1)} + 2^{(i+1)}$$

$$\sum_{i=0}^{n} i2^i + (n+1)2^{(n+1)} = \sum_{i=0}^{n} i2^i \cdot i2 + 2^i \cdot 2$$

$$\sum_{i=0}^{n} i2^i + (n+1)2^{(n+1)} = \sum_{i=0}^{n} i2^i + 2^i$$

$$\sum_{i=0}^{n} i2^i + (n+1)2^{(n+1)} = \sum_{i=0}^{n} i2^i + \sum_{i=0}^{n} 2^i$$

$$(n+1)2^{(n+1)} = 2\sum_{i=0}^{n} i2^i + 2^i - \sum_{i=0}^{n} i2^i$$

$$(n+1)2^{(n+1)} = \sum_{i=0}^{n} i2^i + 2^i$$

05/09/2019

Solução $\sum_{i=0}^{n} i2^i$

$$0 \cdot 2^0 + \sum_{i=0}^{n} (i+1)2^{i+1} = \sum_{i=0}^{n} i2^i + (n+1)2^{n+1}$$

$$\sum_{i=0}^{n} (i2^{i+1} + 2^{i+1}) = \sum_{i=0}^{n} i2^i + n2^{n+1} + 2^{n+1}$$

$$\sum_{i=0}^{n} i2^{i+1} + \sum_{i=0}^{n} 2^{i+1} = \sum_{i=0}^{n} i2^i + n2^{n+1} + 2^{n+1}$$

$$\sum_{i=0}^{n} 2i2^i + \sum_{i=0}^{n} 2 \cdot 2^i = \sum_{i=0}^{n} i2^i + n2^{n+1} + 2^{n+1}$$

$$2\sum_{i=0}^{n} i2^i + 2\sum_{i=0}^{n} 2^i = \sum_{i=0}^{n} i2^i + n2^{n+1} + 2^{n+1}$$

$$2\sum_{i=0}^{n} i2^i + 2(2^{n+1} - 1) = \sum_{i=0}^{n} i2^i + n2^{n+1} + 2^{n+1}$$

$$2\sum_{i=0}^{n} i2^i - \sum_{i=0}^{n} i2^i = n2^{n+1} + 2^{n+1} - (2(2^{n+1} - 1)$$

$$\sum_{i=0}^{n} i2^i = n2^{n+1} + 2^{n+1} - 2 \cdot 2^{n+1} + 2$$

$$\sum_{i=0}^{n} i2^i = n2^{n+1} - 2^{n+1} + 2$$

12/09/2019

Solução da Torre de Hanoi

$$T(n) = 2T(n-1) + 1$$

$$T(n) = 2T(n-1) + 1$$
$$2T(n-1) = 2^2T(n-2) + 2$$
$$2^2T(n-2) = 2^3T(n-3) + 2^2$$
$$2^3T(n-3) = 2^4T(n-4) + 2^3$$

$$2^{x-1}T(n-x) = 2^xT(n-1) + 2^{x-1}$$

$$T(n) = 1 + 2^1 + 2^2 + 2^3 + \dots + 2^{n-1}$$

$$T(n) = \sum_{i=0}^{n-1} 2^i$$

Exercício anterior

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \rightsquigarrow \sum_{i=0}^{n-1} 2^i = 2^n - 1$$

Complexidade de Tempo de $O(2^n)$

Exercício 1

a) $T(n/2) + n$

$$T(n) = T(n/2) + n$$
$$T(n/2) = T(n/2^2) + n/2$$
$$T(n/2^2) = T(n/2^3) + n/2^2$$
$$\vdots$$
$$T(n/2^{x-1}) = T(n/2^x) + n/2^x$$

$$\frac{n}{2^x} = 1$$
$$n = 2^x$$
$$x = \log_2 n$$

$$T(n) = \frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \dots + \frac{n}{2^{x-1}}$$

Per barsão

$$\sum_{i=0}^{x} n/2^i = n\sum_{i=0}^{x} 1/2^i \qquad 1/2^0 + 1/2^1 + 1/2^2 + \dots + 1/2^x + 1/2^{x+1}$$

$$1/2^0 + \sum_{i=0}^{x} 1/2^{x+1} = \sum_{i=0}^{x} 1/2^i + 1/2^{x+1}$$

$$n\left(2 - \frac{1}{2^x}\right) = \sum_{i=0}^{x} \frac{1}{2^i}$$

$n(2 - 1/n)$ $\quad 2n - 1$

24/09/2019

Transparência: PAA1.pdf (de 33 até 38)

Heap Binário *

a raiz dele sempre será maior que suas folhas

Heapify e conceito de heap

Transparência: PAA1.pdf (de 33 até 38)

a árvore da esquerda possui 2/3 dos elementos = pior caso

$$T(n) = T(2n/3) + O(1)$$
$$T(1) = O(1)$$

26/09/2019

$$\sum_{i=0}^{h} (h-i).2^i \qquad h = \lfloor \log_2 n \rfloor$$

$$\sum_{i=0}^{h} (h.2^i - i2^i)$$

$$\sum_{i=0}^{h} h.2^i - \sum_{i=0}^{h} i2^i$$

$$h \sum_{i=0}^{h} 2^i - \sum_{i=0}^{h} i2^i$$

$$2.2^h.h - h - 2.2^h.h + 2.2^h - 2$$

$$2.2^h - h - 2$$

$$2.2^{\log_2 n} - \log_2 n - 2$$

$$2n - \log_2 n - 2$$

$$O(n)$$

Melhor Ω Médio Θ Pior O

26/09/2019

Transparência: PAA pdf (de 36 até o 40)

Insertion
Melhor caso: $O(n)$
Pior caso: $O(n^2)$

Merge
Melhor caso: $O(n \log n)$
Pior caso: $O(n \log n)$

PYTHON tem inteiros mas tem inteiros de forma restrita

Algoritmos com Inteiros (Multiplicação)

x e y vai até $2^n - 1$

01/10/2019

Transparência: PAA pdf (de 40 até o 45)

"Algoritmos com Inteiros (Multiplicação)"

$T(n) = T(n-1) + O(n)$
$T(1) = O(1)$

JÁ RESOLVEMOS $O(n^2)$

sem recursividade $O(n) + O(n) + O(n) = O(n)$

com recursividade $T(n-1)$

$O(n)$

08/10/2019

Transparências: PAA.pdf (de 40 até 52)

Tema do trabalho:
Criptografia RSA
07/11/2019
ENTREGA

Primos relativos são números que só possuem o número 1 como divisor em comum

$C = M^e \bmod n$ → deverão ser primos
potência modular (Biblioteca) — pode usar
p9

Temos que gerar um número randômico e devemos garantir que seja primo. Não podemos usar bibliotecas prontas para gerar um número primo

deveremos implementar

* Teste de primalidade
* Relação de convergência
* Inverso multiplicativo modular
* Chave pública
* Chave privada
* Algoritmo de Euclides extendido

* Para facilitar cada caracter pode ser convertido de ASCII

Gerar gráfico de tempo que for levado para criptografar + tempo descriptografar + tempo de geração das chaves (p e q) + fazer o gráfico de força bruta (dois) (teste até 32 bits)

Criptografia
+
Descriptografia
+
gerar chave pública
gerar chave privada
+
Algoritmo de Euclides extendido
+
Criar um algoritmo de força bruta
+
Método de força bruta

10/10/2019

Transparência: PAA1.pdf (de 52 até 60)

Grafo ponderado = (ou valorado) valor nas arestas

Grafo completo = todos conectados

Problema das PONTES

Multigrafo = vários caminhos

(grafo simples) bom loop = diagonal gerada

$G = (V, A)$
- A → arestas
- V → vértices

Complexidade 1 para ver as arestas

Complexidade n^2 para espaço

Matriz de Adjacência

Lista de Adjacência

Complexidade $O(n)$ para ver as arestas (pior caso) — n = M

Complexidade n^2 no pior caso de espaço

Complemento de um grafo

complemento → $O(n^2)$ → conversão

+ grafos simples não tem loop

todos grau par com exceção de início e fim

Caminho Euleriano: passar em todas as arestas uma vez.

Ciclo Euleriano: passar uma vez e voltar a posição de início.
↳ todos os vértices devem ter grau par

Exercício

Dado um grafo → dizer se ele tem ciclo euleriano → Se sim "Dizer o ciclo"

15/10/2019

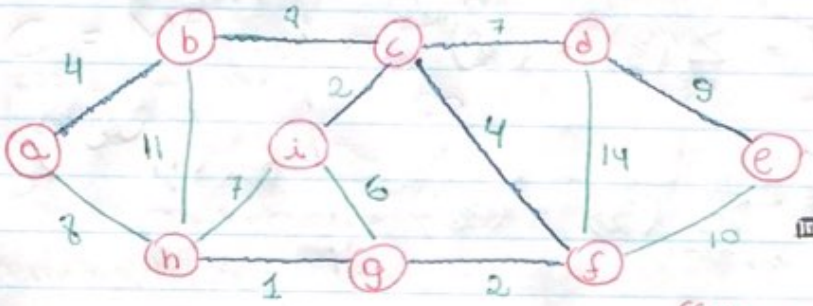
Transparência: PAA1.pdf (dg 60 até 63)

Caixeiro viajante
Grafo ponderado
→ força bruta (n-1)!
→ melhor solução local
*analisa um a um pelo menor caminho

é um grafo particular (existem ciclos)

Arvore Geradora Minima

→ conectar todos os vertices com custo mínimo



⇒ MST - PRIM

"Marcar todos os vertices como infinito inicialmente e depois ir substituindo pelos menores custos a sua origem"

$Q = \{a, b, c, d, e, f, g, h, i\}$

Custo = 4 + 8 + 7 + 9 + 4 + 2 + 2 + 1 = 37

$O(|V| \log |V|)$

$O(|A| \log |V|)$

DOMINA ARESTAS

17/10/2019

Revisão Prova

$$\sum_{i=0}^{n} \left(\frac{3}{2}\right)^i = \left(\frac{3}{2}\right)^0 + \left(\frac{3}{2}\right)^1 + \left(\frac{3}{2}\right)^2 + \cdots + \left(\frac{3}{2}\right)^n + \left(\frac{3}{2}\right)^{n+1}$$ (Add)

$$1 + \sum_{i=0}^{n} \left(\frac{3}{2}\right)^{i+1} = \sum_{i=0}^{n} \left(\frac{3}{2}\right)^i + \left(\frac{3}{2}\right)^{n+1}$$

$$1 + \sum_{i=0}^{n} \frac{3}{2}\left(\frac{3}{2}\right)^i = \sum_{i=0}^{n} \left(\frac{3}{2}\right)^i + \left(\frac{3}{2}\right)^{n+1}$$

$$1 + \frac{3}{2}\sum_{i=0}^{n} \left(\frac{3}{2}\right)^i = \sum_{i=0}^{n} \left(\frac{3}{2}\right)^i + \left(\frac{3}{2}\right)^{n+1}$$

$$\frac{3}{2}\sum_{i=0}^{n} \left(\frac{3}{2}\right)^i - \frac{2}{2}\sum_{i=0}^{n} \left(\frac{3}{2}\right)^i = \left(\frac{3}{2}\right)^{n+1} - 1$$

$$\frac{1}{2}\sum_{i=0}^{n} \left(\frac{3}{2}\right)^i = \left(\frac{3}{2}\right)^{i+1} - 1$$

$$\sum_{i=0}^{n} \left(\frac{3}{2}\right)^i = 2\left(\frac{3}{2}\right)^{n+1} - 2$$

$T(n) = 3T(n/2) + n$
$T(1) = 1$

$O(n^{\log_2 3})$

PODE CAIR NO INT

PROVA
- Saber grafos e Dijstra e PRIM
- Relações de recorrência caem na prova
- A prova vai ter uma solução de implementação.
- Fazer um algoritmo simples e dar a complexidade
- Podem ter soluções de verdadeiro ou falso → justificativa
- Algoritmos de ordenação caem na prova (saber complexidade)

17/10/2019

Revisão Prova

$T(n) = T(\frac{n}{2}) + \log_2 n$

$T(1) = 1$

$T(n) = T(\frac{n}{2}) + \log_2 n$

$T(\frac{n}{2}) = T(\frac{n}{2^2}) + \log_2 \frac{n}{2}$

$T(\frac{n}{2^2}) = T(\frac{n}{2^3}) + \log_2 \frac{n}{2^2}$

$T(\frac{n}{2^3}) = T(\frac{n}{2^4}) + \log_2 \frac{n}{2^3}$

$T(\frac{n}{2^{i-1}}) = T(\frac{n}{2^i}) + \log_2 \frac{n}{2^{i-1}}$

$\frac{n}{2^i} = 1$

$n = 2^i$

$i = \log_2 n$

$T(n) = \log_2 n + \log_2 \frac{n}{2} + \log_2 \frac{n}{2^2} + \ldots + \log_2 \frac{n}{2^{i-1}} + 1$

$T(n) = \log_2 n + (\log_2 n - \log_2 2) + (\log_2 n - \log_2 2^2) + \ldots + (\log_2 n - \log_2 2^{i-1}) + 1$

$T(n) = \log_2 n + (\log_2 n - 1) + (\log_2 n - 2) + \ldots + (\log_2 n - (i-1)) + 1$

$T(n) = \log_2^2 n - 1 - 2 - \ldots - (i-1) + 1$

log ao quadrado

$T(n) = \log_2^2 n - \sum_{i=0}^{i-1} i + 1$

$T(n) = \log_2^2 n - \left(\frac{\log_2^2 n - \log_2 n}{2}\right) + 1$

$T(n) = \log_2^2 n - \frac{\log_2^2 n}{2} + \frac{\log_2 n}{2} + 1$

$T(n) = \frac{\log_2^2 n + \log_2 n}{2} + 1$

$O(\log n)$

24/10/2019

Transparência: PAA2.pdf (de 0 até 7)

Problemas

tratáveis x INTRATÁVEIS

normalmente só temos soluções de força bruta

Decisão → sim / não

Otimização → várias resposta (ou apenas uma)
↳ melhores respostas

CLASSE **NP**: está definido somente em problemas de decisão

Satisfazibilidade de Fórmulas Booleanas

verificar se existe uma atribuição de valores booleanos para as variáveis que ocorrem na fórmula, de tal maneira que o resultado seja verdadeiro.

Exemplo:

NÃO É SATISFAZÍVEL

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

29/10/2019

Transparência: PAA2.pdf (de 8 até 14)

P vs NP

Redução de Problemas
↳ Transforma a entrada do problema A em uma entrada para o problema B

Graças ao SAT foi possível definir NP-completo

Teorema de Cook-Levi: SAT está em P se e somente se P = NP

Um problema X é NP-Completo se:

1. $X \in NP$
2. $X \leq_p X$ para todo $X' \in NP$

31/10/2019

Transparência: PAA2.pdf (de 14 até 21)

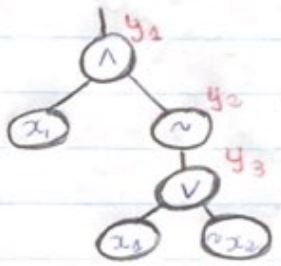
3 - CNF - SAT

3-CNF-SAT $\in$ NP

A verificação da resposta é linear ao número de operadores da expressão

$SAT' \leq_p$ 3-CNF-SAT → Sim / → Não

Dada uma fórmula $\phi = x_1 \wedge \sim(x_1 \vee x_2)$ → árvore sintática abstrata



$\phi' = y_1 \wedge (y_1 \leftrightarrow (x_1 \wedge y_2)) \wedge (y_2 \leftrightarrow \sim y_3) \wedge (y_3 \leftrightarrow (x_1 \vee \sim x_2)$

$\rightarrow (y_1 \wedge x_1 \wedge \sim y_2) \vee (y_1 \wedge \sim x_1 \wedge y_2) \vee (y_1 \wedge \sim x_1 \wedge \sim y_2) \vee (\sim y_1 \wedge x_1 \wedge y_2)$

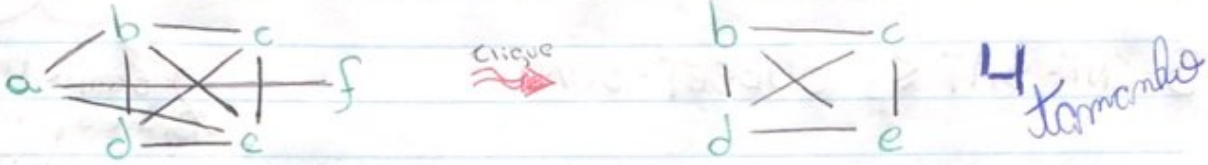
$\rightarrow (y_1 \vee p \vee q) \wedge (y_1 \vee \sim p \vee \sim q) \wedge (y_1 \vee \sim p \vee q) \wedge (y_1 \vee p \vee \sim q)$

... incompleto

05/11/2019

Transparência: PAA2.pdf (de 22 até 25)

Clique



$SAT \leq_p 3\text{-}CNF\text{-}SAT \leq_p CLIQUE$

geral sem 3

$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$ fórmula qualquer

07/11/2019

Transparência: PAA2.pdf (de 26 até 29)

Cobertura de Vértices

$SAT \leq_p 3\text{-}CNF\text{-}SAT \leq_p$ CLIQUE $\leq_p$ COBERTURAVÉRTICES (trivial)

Otimização = NP-Hard

Decisão = NP

19/11/2019

Transparência: PAA2.pdf (do 30 até o 32)

SUBSET-SUM

$3\text{-CNF-SAT} \leq_p \text{SUBSET-SUM}$

Sempre transforme o problema em um problema de decisão

26/11/2019

Transparência: PAA2.pdf (do 33 até o 44)

Programação Dinâmica

↳ resolve problemas combinando as soluções de subproblemas

$T(n) = T(n-1) + O(1)$
$T(1) = 1$

28/11/2019

Transparência: PAA2.pdf (do 43 até o 48)

$O(1)$
$O(t)$
$+ O(n.t)$
$O(n.t)$

tempo e espaço $O(t)$

↳ a soma desejada (cresce exponencialmente)

↳ número de elementos do conjunto

* pseudo-polinomial

Fim