Teoria da Computação

Prof. Sergio D Zorzo
Departamento de Computação - UFSCar
2023 / 2
8

Intratabilidade

Introdução

- Nessa aula, iremos partir da análise sobre:
  - o que pode ser resolvido por um computador (e o que não pode)
- E passaremos a analisar:
  - o que pode ser resolvido por um computador em um tempo realístico (e o que não pode)
- Ou seja, o fato de um problema ser decidível não é suficiente
  - Pois ele pode levar um longo tempo
  - Ou pode exigir muito espaço em memória

O que é “muito tempo”?

- Qual a linha divisória entre problemas tratáveis e problemas intratáveis?
- Na teoria da complexidade/intratabilidade
  - Tempo polinomial (tratáveis)
  - Tempo exponencial (intratáveis)
- Na realidade, existem problemas com tempo intermediário
  - Ex: $n^{\log_2 n}$
  - Esses também são intratáveis
- Na prática, a maioria dos problemas se enquadra em polinomial ou exponencial

O que é “muito tempo”?

- Para todas análises que se seguem:
  - Diferenças polinomiais em tempo de execução são consideradas pequenas
  - Diferenças exponenciais em tempo de execução são consideradas grandes
- Razão: a diferença entre a taxa de crescimento de polinômios e exponenciais
  - Ex: n^3 e 2^n, para n = 1000
  - 1000^3 = 1000000000 é grande, mas não muito, em termos computacionais
  - 2^{1000} = mais do que o número de átomos no universo

Algoritmos exponenciais

- Normalmente não tem utilidade prática
  - Conseguem chegar a uma decisão apenas para instâncias MUITO pequenas
- Tipicamente envolvem busca exaustiva
  - Busca pela força bruta
    - Ex: fatorar um número em seus primos, buscando-se todos os possíveis divisores
  - Às vezes, pode-se evitar a busca por força bruta
    - Entendendo-se melhor o problema, revelando um algoritmo de tempo polinomial

Diferenças polinomiais

- Todos os modelos computacionais determinísticos são polinomialmente equivalentes
  - Máquina de Turing determinística com uma fita
  - Máquina de Turing determinística com múltiplas fitas
  - Computador real
  - Outros dispositivos determinísticos semelhantes
- Mas.....
  - Programadores trabalham duro para tentar fazer programas rodarem duas vezes mais rápido
  - E agora – teoricamente diz-se que rodar em n ou n^{1000} é equivalente?

Diferenças polinomiais

- Na prática, diferenças polinomiais nem sempre são ignoráveis
- Mas essas diferenças tendem a serem resolvidas com o tempo
- Ao passo que para diferenças SIGNIFICATIVAS não há perspectiva iminente de solução
  - Como por exemplo: polinomial X exponencial
  - É justamente pela diferença na magnitude citada anteriormente

A classe P

- Portanto, definimos uma importante classe de linguagens:

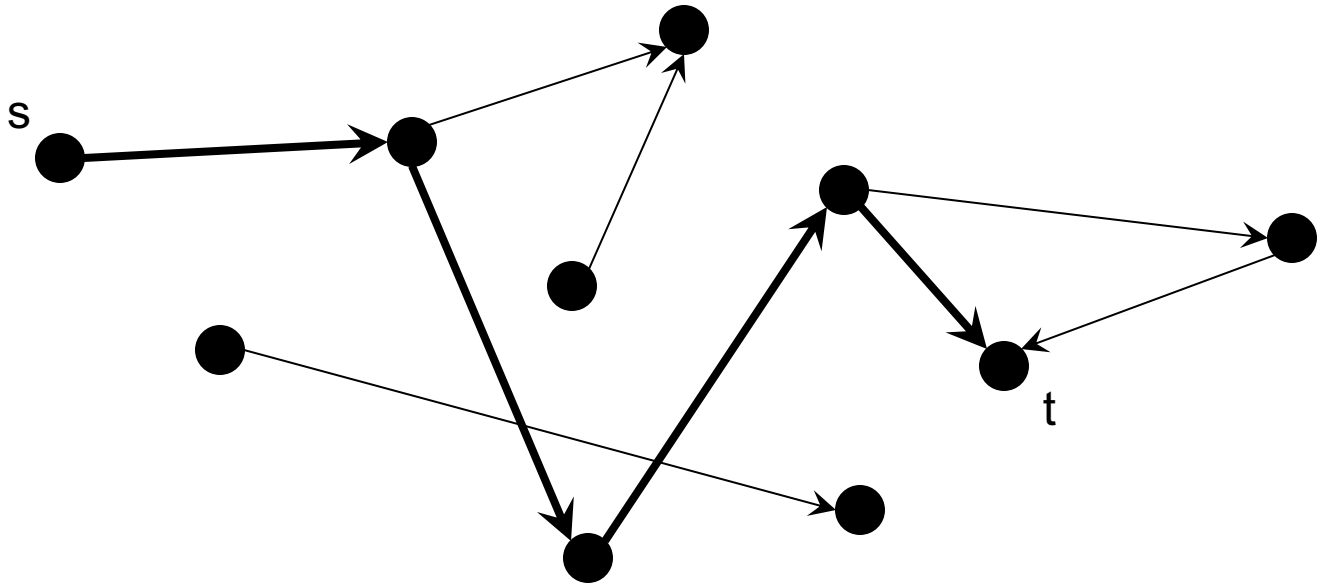
P é a classe de linguagens que são decidíveis em tempo polinomial sobre uma máquina de Turing determinística de uma fita

- P é invariante para todos os modelos de computação polinomialmente equivalentes à DTM de uma fita
- P corresponde aproximadamente à classe de problemas que são realisticamente solúveis em um computador

- Veremos a seguir alguns exemplos de problemas em P

CAM

- CAM = {(G,s,t) | G é um grafo direcionado, s e t são nós de G, e G tem um caminho direcionado de s para t}



CAM

- Existe um algoritmo exponencial que decide CAM
  - Usando busca pela força bruta, analisando todos os caminhos que passam por no máximo todos os nós de G, e encontrando um que envolve s e t
  - Guarde esse algoritmo na memória (usaremos depois)
- Mas existe um algoritmo de tempo polinomial que decide CAM
  - Ou: Existe uma MT determinística que sempre para e decide CAM
  - Portanto CAM pertence a P

CAM

- Um algoritmo polinomial para CAM:
  1. Ponha uma marca sobre o nó s
  2. Repita o seguinte até que nenhum nó adicional seja marcado:
  3. Faça uma varredura em todas as arestas de G. Se uma aresta (a,b) for encontrada indo de um nó marcado a para um nó não-marcado b, marque o nó b
  4. Se t estiver marcado, aceite. Caso contrário, rejeite.

PRIM-ES

- Dois números são primos entre si se 1 é o maior inteiro que divide ambos.
  - Ex: 10 e 21 são primos entre si, embora nenhum deles seja um número primo
  - Ex: 10 e 22 não são primos entre si, pois ambos são divisíveis por 2

- PRIM-ES = {(x,y)|x e y são primos entre si}

PRIM-ES

- Existe um algoritmo exponencial
  - Busca por força bruta todos os possíveis divisores de ambos os números e aceita se nenhum é maior que 1
- Existe um algoritmo melhor, chamado algoritmo euclidiano:
  1. Repita até que y=0
     a. Atributa x ← x mod y
     b. Intercambie x e y
  2. Dê como saída x
  3. Se x for 1, aceite. Caso contrário, rejeite

A classe NP

- Evitar a força bruta é o objetivo
  - Muitas vezes permitem sair do exponencial e chegar no polinomial
  - Mas existem muitos (MUITOS) problemas interessantes, práticos e úteis, para os quais ainda não se conseguiu evitar a força bruta
  - Não se sabe se existem algoritmos de tempo polinomial que os resolvem

A classe NP

- Por que ninguém ainda encontrou algoritmos de tempo polinomial para esses problemas?
  - Não sabemos a resposta para essa questão!!
  - Talvez esses problemas tenham algoritmos de tempo polinomial, mas ninguém ainda descobriu! Talvez exista algum princípio desconhecido necessário.
  - Ou talvez, alguns desses problemas simplesmente NÃO PODEM ser resolvidos em tempo polinomial. Eles podem ser intrinsecamente difíceis.

A classe NP

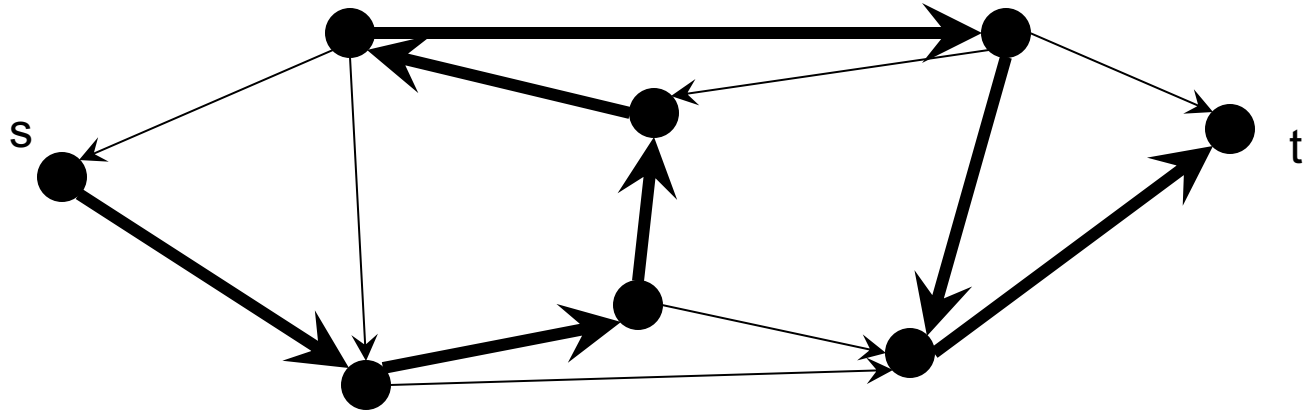
- Ou seja, aparentemente, existe uma classe, chamada NP, que é diferente de P (na verdade, NP é superconjunto de P)
  - Onde residem muitos problemas que não possuem solução polinomial
- Importante repetir: esses problemas têm uma solução, eles são decidíveis, existe um decisor!!!
  - Mas simplesmente, a solução (algoritmo) leva tempo polinomial
  - Isto significa, na prática, que uma entrada de tamanho razoável, como 1.000.000.000, por exemplo, pode levar um tempo equivalente à idade conhecida do universo

A classe NP

- Mas sabemos pelo menos UMA coisa sobre a classe NP
  - Os problemas em NP possuem uma característica em comum
  - Todos são **verificáveis** em tempo polinomial
  - Ou seja, uma vez sabendo a resposta, é fácil
    Veremos alguns exemplos para ilustrar esse fenômeno

CAMHAM

- Um caminho hamiltoniano em um grafo direcionado G é um caminho direcionado que passa por cada nó exatamente uma vez
- CAMHAM = {(G,s,t) | G é um grafo direcionado com um caminho hamiltoniano de s para t}

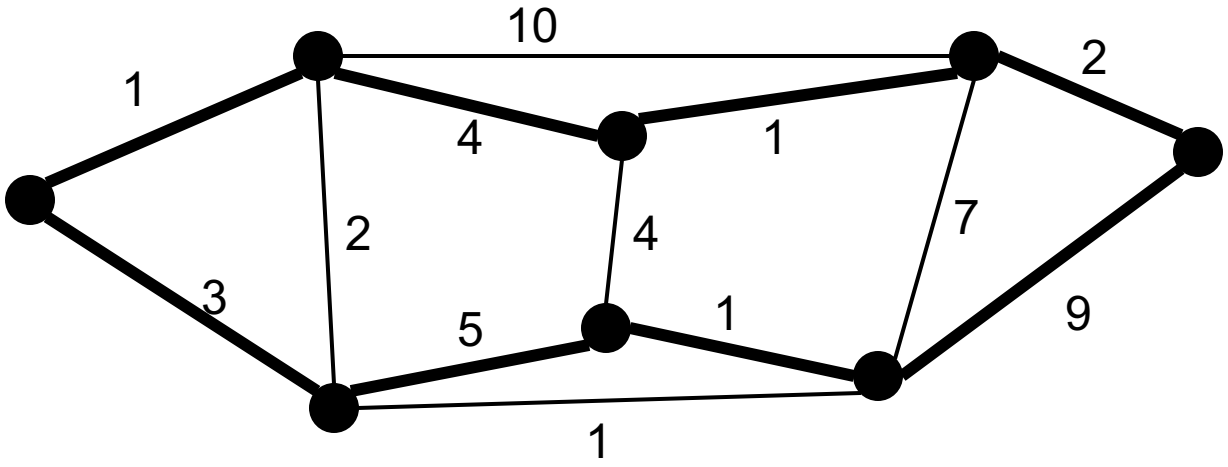


CAMHAM

- Lembra-se do algoritmo exponencial para CAM?
  - Ele fazia uma busca por todos os caminhos que passam por no máximo todos os nós de G. Cada um é um caminho potencial. (passo 1)
  - Para cada caminho potencial, basta testar se o mesmo é um caminho hamiltoniano. (passo 2)
  - Ou seja, o passo 1 leva tempo exponencial, mas o passo 2 leva tempo polinomial
- Bom, ainda assim, são necessários os 2 passos, o que resulta em tempo total exponencial
  - Mas já descobrimos algo.... vejamos outro exemplo

TSP – Traveling Salesman Problem

- O problema do caixeiro-viajante consiste em, dado um grafo não-direcionado G, onde cada aresta possui um peso inteiro, e um limite de peso W, encontrar um circuito hamiltoniano de peso total no máximo W



W=28

TSP

- Não foi encontrada ainda uma solução* que execute em tempo menor do que exponencial, mas:
  - Uma vez que se gaste tempo exponencial para encontrar uma solução (passo 1)
  - Pode-se, em tempo polinomial, verificar que a mesma é válida (passo 2)
- Novamente: o tempo total é exponencial
  - Mas existe um passo polinomial envolvido

* Existem soluções aproximadas bastante satisfatórias

Verificabilidade polinomial

- Alguns problemas, muito embora não conheçamos uma forma rápida (polinomial) de determinar uma solução
  - Uma vez descoberto, é possível verificar que a solução é válida
- Em CAMHAM, encontrar um caminho hamiltoniano é difícil, mas verificar que ele existe mesmo e é real, é fácil
- No TSP, encontrar um circuito hamiltoniano cujo peso total não ultrapasse um limite é difícil, mas verificar que ele existe mesmo e é real, é fácil

Verificador

- Um verificador para uma linguagem A é um algoritmo B, onde
  - A={w | V aceita (w,c) para alguma cadeia c}
- Ou seja, dada uma cadeia w, e alguma informação adicional c, o verificador consegue determinar que w é membro de A
  - Essa informação extra c é chamada certificado, ou prova, da pertinência a A
  - Ex:
    - No CAMHAM, c é o caminho hamiltoniano de s a t
    - No TSP, c é o circuito hamiltoniano com peso máximo W

Verificador

- Um verificador de tempo polinomial executa em tempo polinomial no comprimento de w
- Ou seja, caso exista um verificador em tempo polinomial V para uma linguagem A
  - A é polinomialmente verificável

A classe NP

- Chegamos a outra definição vital de uma classe de linguagens

NP é a classe de linguagens que têm verificadores de tempo polinomial

- Obviamente, todo problema em P também está em NP ($P \subseteq NP$)
  - Mas veremos mais sobre isso daqui a pouco

A classe NP

- NP = Nondeterministic Polynomial time
  - Ou: tempo polinomial não-determinístico
- Existe uma caracterização alternativa que usa máquinas de Turing como base

NP é a classe de linguagens que são decidíveis em tempo polinomial sobre uma máquina de Turing não-determinística de uma fita

- Novamente, é óbvio que todo problema em P também está em NP ($P \subseteq NP$)
  - Mas calma

A classe NP

- É fácil imaginar a relação entre a definição com base em verificadores e a definição com base em MTs
  - V é um algoritmo, portanto existe uma MT Mv que o implementa
  - Se V é polinomial, a Mv também é
- Mv aceita entrada (w,c) em tempo polinomial
  - Mas precisamos construir uma NTM M que aceita w (sem c) em tempo polinomial
  - Construiremos M de forma a executar 3 passos:
    1. M primeiro “adivinha” c, usando seu poder não-determinístico
    2. Em seguida, M simula Mv
    3. Se Mv aceita, M aceita. Se Mv rejeita, M rejeita

A classe NP

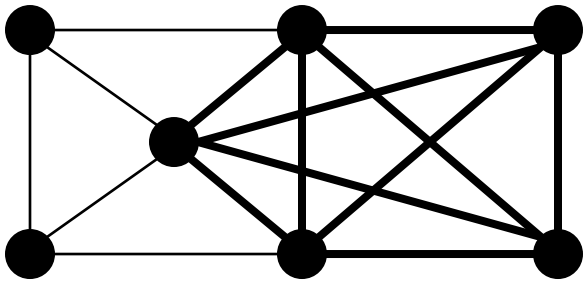
- O passo 1 é polinomial com relação ao tamanho de c (na verdade, basta “adivinhar” |c| símbolos, portanto o passo 1 leva um tempo constante)
  - Adicionamente c tem tamanho polinomial em relação à entrada w
  - Pois de outra forma, V não seria polinomial
  - Portanto o passo 1 é polinomial em relação à entrada w
- O passo 2 é polinomial devido à própria definição de V
- O passo 3 executa obviamente em tempo constante (na verdade, apenas um movimento é necessário)
- Portanto, M executa em tempo polinomial

Exemplos de problemas em NP

- Veremos agora alguns exemplos de problemas em NP
- Na verdade, os problemas anteriores, CAM, PRIM-ES, CAMHAM e TSP todos são exemplos de problemas em NP
  - Sabemos que CAM e PRIM-ES estão em P também
  - Mas para outros, não sabemos se estão em P (CAMHAM e TSP)
  - A seguir, apresentaremos alguns problemas que estão em NP, e aparentemente não estão em P

CLIQUE

- Um clique em um grafo não-direcionado é um subgrafo no qual todo par de nós está conectado por uma aresta. Um k-clique é um clique que contém k nós.
- CLIQUE = {(G,k) | G é um grafo não-direcionado com um k-clique}



Um grafo com um 5-clique

CLIQUE

- CLIQUE está em NP
- Prova:
  - O clique é o certificado c
  - Um verificador V para CLIQUE:
    1. Teste se c é um conjunto de k nós em G
    2. Teste se G contém todas as arestas conectando nós em c
    3. Se ambos os testes retornam positivo, aceite. Caso contrário, rejeite.
- Prova alternativa
  - Uma NTM polinomial que decida CLIQUE:
    - Mesmo procedimento acima, mas utilizando um passo não-determinístico que “adivinha” c

SOMA-SUBC

- Temos uma coleção de números x1,x2,...,xk e um número-alvo t. Desejamos determinar se a coleção contém uma subcoleção que soma t.
- SOMA-SUBC = {(S,t) | S = {x1,x2,...,xk} e para algum {y1,y2,...,yl} ⊆ S, temos Σyi = t}

- S={4,11,16,21,27}
- t=25
- (S,t) pertence a SOMA-SUBC
  - Pois 4 + 21 = 25

SOMA-SUBC

- SOMA-SUBC está em NP
- Prova
  - O subconjunto é o certificado c
  - Um verificador V para SOMA-SUBC:
    1. Teste se c é uma coleção de números que somam t
    2. Teste se S contém todos os números em c
    3. Se ambos os testes retornam positivo, aceite. Caso contrário, rejeite.
- Prova alternativa
  - Uma NTM polinomial que decide SOMA-SUBC
    - Mesmo procedimento acima, mas usando um passo não-determinístico que encontra c

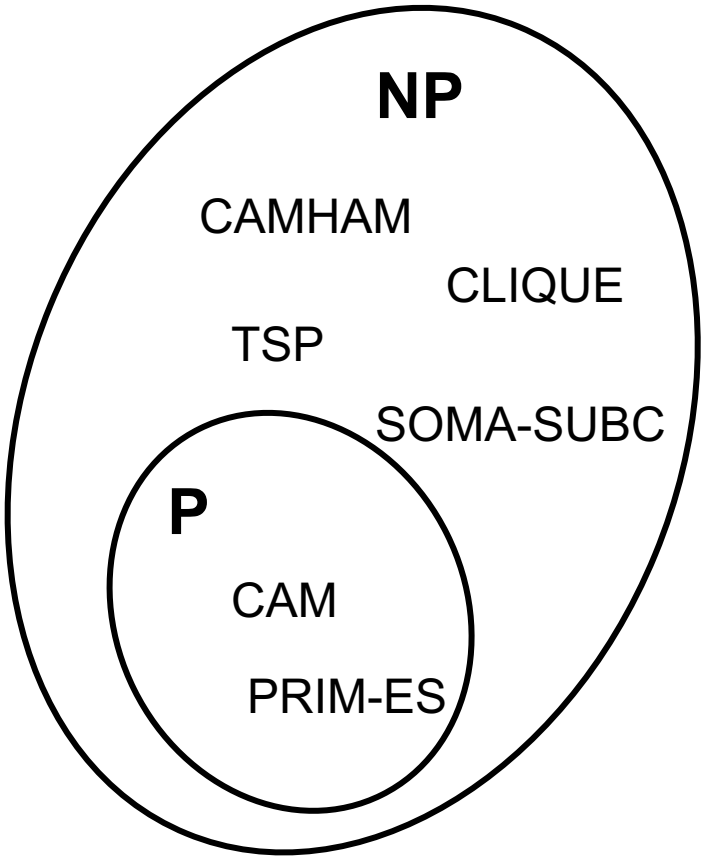
P versus NP

P versus NP

- P = classe das linguagens para as quais pertinência pode ser DECIDIDA rapidamente
- NP = classe das linguagens para as quais pertinência pode ser VERIFICADA rapidamente
- Existem muitos exemplos de problemas em P
  - CAM, PRIM-ES
- Existem muitos exemplos de problemas em NP, que aparentemente não estão em P
  - CAMHAM, CLIQUE, TSP, SOMA-SUBC

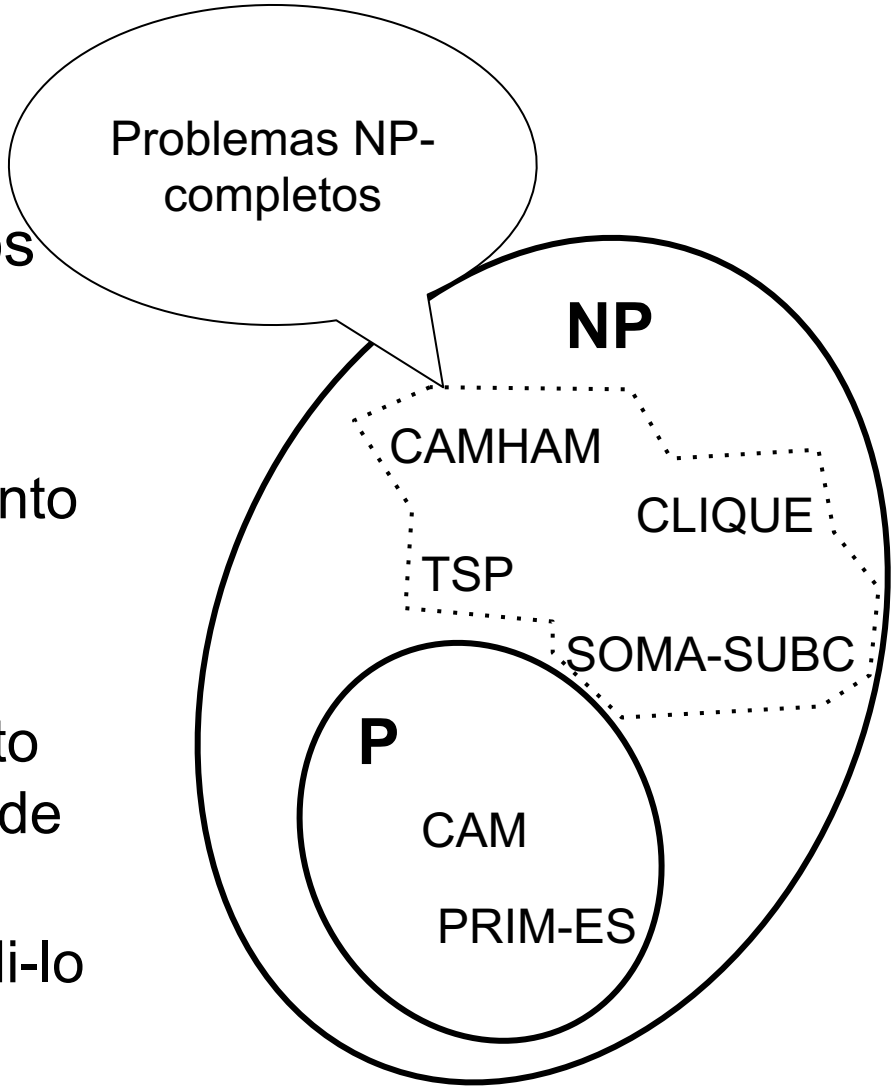
P ≠ NP?

- Aparentemente P ≠ NP
- Porque “aparentemente”?
  - Porque muitas pessoas já tentaram resolver esses problemas em tempo menor que exponencial, e nunca conseguiram.
- Ou seja, apesar de provável, ninguém ainda conseguiu provar a existência de uma única linguagem que está em NP, mas não está em P



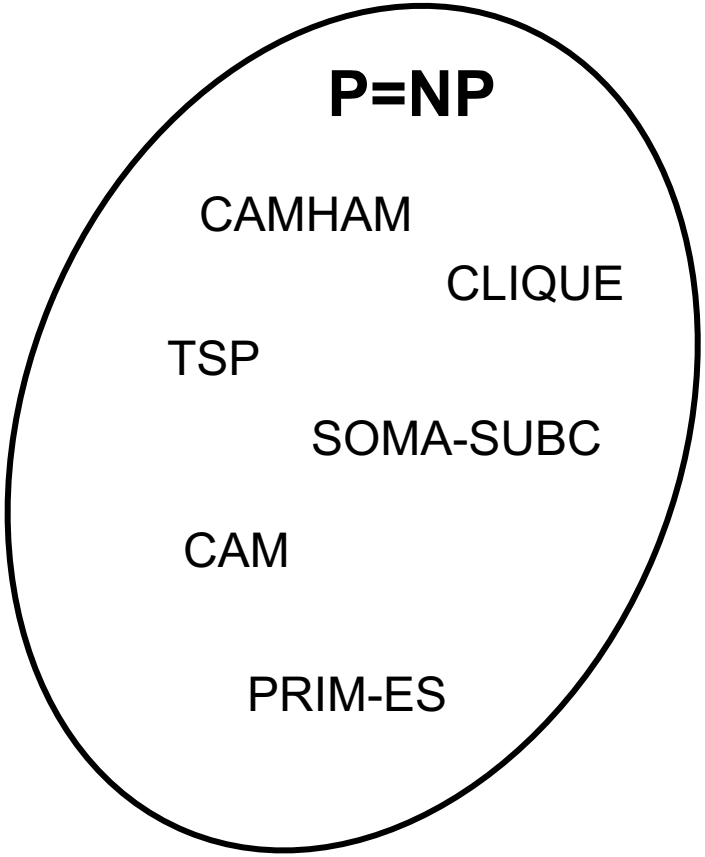
P ≠ NP?

- Tais problemas são chamados de NP-completos
- Uma classe especial de problemas
  - Encapsula justamente o ponto de dúvida
- Se P≠NP
  - Todo problema NP-completo compartilha essa propriedade de complexidade
  - Ou seja, é impossível decidi-lo em tempo polinomial



P = NP?

- Mas pode ser que P = NP!
  - Talvez exista algum princípio matemático desconhecido
- Se P=NP, qualquer problema polinomialmente verificável seria polinomialmente decidível



P versus NP

- Como provar que P = NP?
  - Basta encontrar um único algoritmo de tempo polinomial que decide um único problema NP-completo
    - Com isso, todos problemas NP-completos poderiam ser decididos também, pois todos são relacionados entre si
  - Ou: basta encontrar uma forma mais eficiente de simular uma NTM em uma DTM
    - Até agora, o melhor método conhecido leva tempo exponencial

P versus NP

- Como provar que P ≠ NP?
  - É necessário provar que não existe algoritmo rápido para substituir a força bruta
  - Ou provar que o método conhecido, de tempo exponencial, para simular uma NTM em uma DTM, é o melhor possível

E se P=NP?

- Primeiro: a prova iria levar eventualmente a métodos eficientes para resolver TODOS problemas NP-completos
  - E são muitos
- Criptografia, por exemplo, depende da dificuldade em resolver problemas NP-completos
  - Primalidade de um número e algoritmos de chave pública
  - Consequências “negativas” = teremos que pensar em novas maneiras para proteger informações

E se P=NP?

- Logística: problema do caixeiro-viajante
  - Consequências “positivas” = otimizar a utilização de meios de transporte
- Biologia: predição de estruturas protéicas tridimensionais
  - Consequências “positivas” = avanços na medicina e farmácia
- Matemática: prova formal de qualquer teorema
  - Exemplo: último teorema de Fermat levou 3 séculos para ser provado

E se P≠NP?

- As consequências seriam menos catastróficas
  - Obs: aqui catástrofe = mudanças rápidas, para o bem e/ou para o mal
- Na verdade, a maioria trabalha com base nessa hipótese
  - Cada nova tentativa frustrada de provar que P=NP aumenta a confiança de que P≠NP
  - Décadas de tentativas são quase que uma prova "estatística" de que P≠NP!
  - Como consequência muitos pesquisam soluções eficientes parciais, ou aproximadas, ao invés de tentar encontrar uma solução eficiente que seja exata
- Mas se alguém achasse a prova formal, teríamos a certeza de que estamos no caminho certo

NP-completude

NP-completude

- Enquanto não se tem a prova de que P≠NP
- Foi observado um fenômeno interessante
  - Certos problemas em NP tem sua complexidade relacionada àquela da classe inteira
  - São problemas cuja complexidade representa a “essência” de NP-P
  - Todos problemas em NP compartilham dessa “essência”
  - Esses problemas são chamados de NP-completos

NP-completude

- A meta da observação desse fenômeno é o seguinte teorema:
  - Se algum problema NP-completo está em P, então P=NP
  - Ou seja, originalmente, era um caminho para se tentar provar que P=NP
- Mas existem também consequências práticas:
  - Se um problema é NP-completo, isso é uma forte evidência de que não existe solução polinomial!
    - Não é uma prova, devido à questão em aberto P vs NP
    - Mas para todos os efeitos práticos, assume-se que não vale a pena tentar encontrar um algoritmo de tempo polinomial
    - O melhor a fazer é buscar soluções alternativas

NP-completude na prática

- Segundo a wikipedia, existem mais de 3000 problemas reconhecidamente NP-completos
- Existe uma GRANDE chance de você se deparar com alguns deles durante sua vida profissional
- Portanto, o estudo da NP-completude não tem impacto somente na teoria
- Saber reconhecer um problema NP-completo, ou identificar um novo problema NP-completo é importante

Complexidade de espaço

- Classes PSPACE e NPSPACE
  - Linguagens decididas por DTMs e NTMS, respectivamente
  - Utilizando quantidade de espaço (fita) polinomial
- Outra classificação de problemas
  - Diferente de P e NP, pois espaço é reutilizável

- Existe o mesmo conceito de problemas PSPACE-completos e PSPACE-difíceis

Como resolver problemas intratáveis?

- RP – Polinomial aleatório
  - Máquinas de Turing baseadas em aleatoriedade
  - Para alcançar tempos de execução médios polinomiais (ainda que existam casos exponenciais, mas esses são pouco frequentes)
  - Algoritmo Monte Carlo
  - Garante algumas respostas, mas pode encerrar sem chegar a uma conclusão
- ZPP – Polinomial probabilística de erro zero
  - MT aleatória que sempre dá a resposta certa, mas pode levar um tempo exponencial
  - Algoritmo Las Vegas
  - Quicksort

Como resolver problemas intratáveis?

- Aproximações
- Heurísticas
- Estatísticas
- Soluções localizadas

Fim

O que você deve levar dessa disciplina? (ou deveria)

- Conceitos de linguagens formais
- Conceito de problemas (pertinência de cadeias em linguagens)
- Conceitos dos autômatos finitos, de Pilha e de Turing:
  - Execução passo-a-passo
  - Conceito de aceitação/relação autômato-classe de linguagens
  - Projeto de autômatos
  - Detalhes de implementação
    - Tempo de execução/simulação das diferentes versões

O que você deve levar dessa disciplina? (ou deveria)

- Conceito de determinismo/não-determinismo
  - E seus impactos nos autômatos
  - E em problemas da vida real
- Gramáticas livre de contexto
  - Projeto/interpretação/ambiguidade
- Indecidibilidade
  - Problemas indecidíveis originais
  - Reduções
- Intratabilidade
  - Problemas intratáveis originais
  - Reduções de tempo polinomial

Fim