Teoria da Complexidade

Para muitos problemas computacionais,
algoritmos razoáveis não existem!

Os melhores algoritmos requerem
quantidades de tempo ou espaço enormes
tornando-os praticamente inúteis.

Introdução

- Objetivos:
  - Introduzir os conceitos básicos da teoria da complexidade através de problemas clássicos.
- Tópicos:
  - O problema da mesa de jantar
  - O problema dos pacotes turísticos
  - Taxa de crescimento de funções
  - Problemas tratáveis X não tratáveis
  - Reduções de problemas
  - A classe NP
  - Soluções Imperfeitas: Aproximações

O problema da mesa de jantar

Problema: Sentar todos os convidados ao redor de uma mesa redonda, de forma que as pessoas dos dois lados se gostem mutuamente.

Exemplo

	John	Mary	Bob	Jane	Alice
John		♥		♥	
Mary	♥		♥		♥
Bob		♥			♥
Jane	♥	♥	♥		♥
Alice			♥	♥	

Solução para o Exemplo

<u>Problema:</u> Sentar todos os convidados ao redor de uma mesa redonda, de forma que as pessoas dos dois lados se gostem mutuamente.

	John	Mary	Bob	Jane	Alice
John		♥		♥	
Mary	♥		♥		♥
Bob		♥			♥
Jane	♥	♥	♥		♥
Alice			♥	♥	

P versus NP:

http://www.claymath.org/Millennium_Prize_Problems/P_vs_NP/

Suponha que você está organizando acomodações no alojamento para um grupo de 400 estudantes. O espaço é limitado e somente 100 dos estudantes têm lugar. Para complicar, o diretor do ICMC lhe passou uma lista de pares de estudantes brigões/encrenqueiros e pediu que nenhum estudante desta lista apareça na lista final.

- Este é um exemplo do que cientistas da computação chamam de problema NP, desde que é fácil checar se uma lista de 100 estudantes proposta satisfaz os requisitos (i.e., nenhum par da lista proposta também aparece na lista negra do diretor),

  Entretanto

  a tarefa de gerar tal lista parece tão difícil como completamente impraticável.

- Na verdade, o número total de formas de escolher 100 estudantes de 400 que pediram vaga no alojamento é maior que o número de átomos de todo o universo conhecido!
- Assim, mesmo no futuro, não se espera construir um supercomputador capaz de resolver o problema por força bruta; isto é, checando-se toda possível permutação de 100 estudantes.

Algoritmo Ingênuo/Simples

- Para cada permutação de convidados
  - **Verifique se cada convidado gosta do outro sentado ao lado.**

Veja que existem permutações repetidas....e temos que considerar somente

$\frac{1}{2}$ (n-1)! possibilidades

Quanto tempo levaria?
(no pior caso)

convidados	passos
n	(n-1)!
5	24
15	87178291200
100	$\approx 9 \cdot 10^{155}$

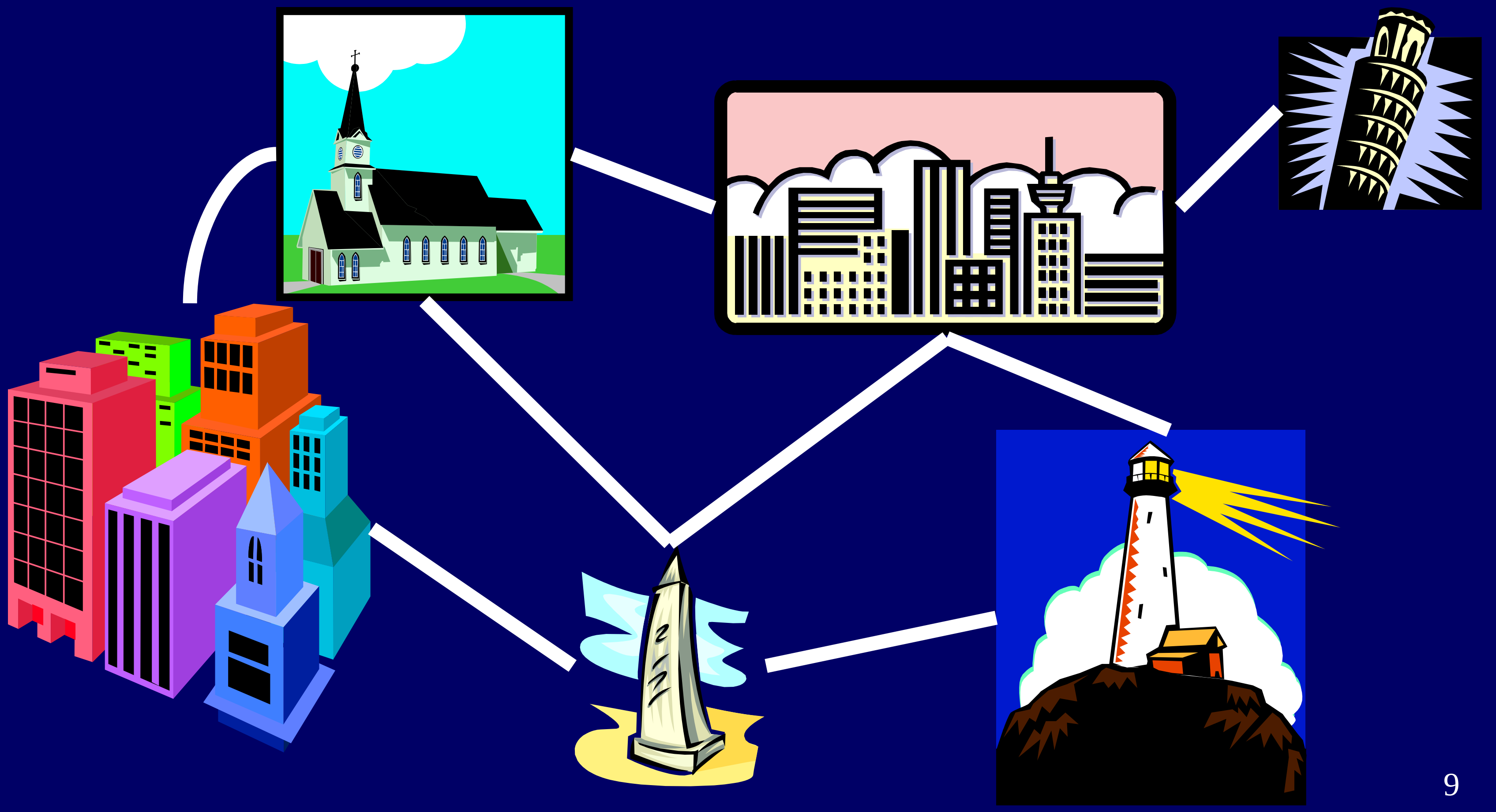
Suponha que o seu computador execute 10^{10} instruções por segundo, isto levará

$\approx 3 \cdot 10^{138}$ anos!

O número de microseg. desde o Big-Bang tem 24 dígitos

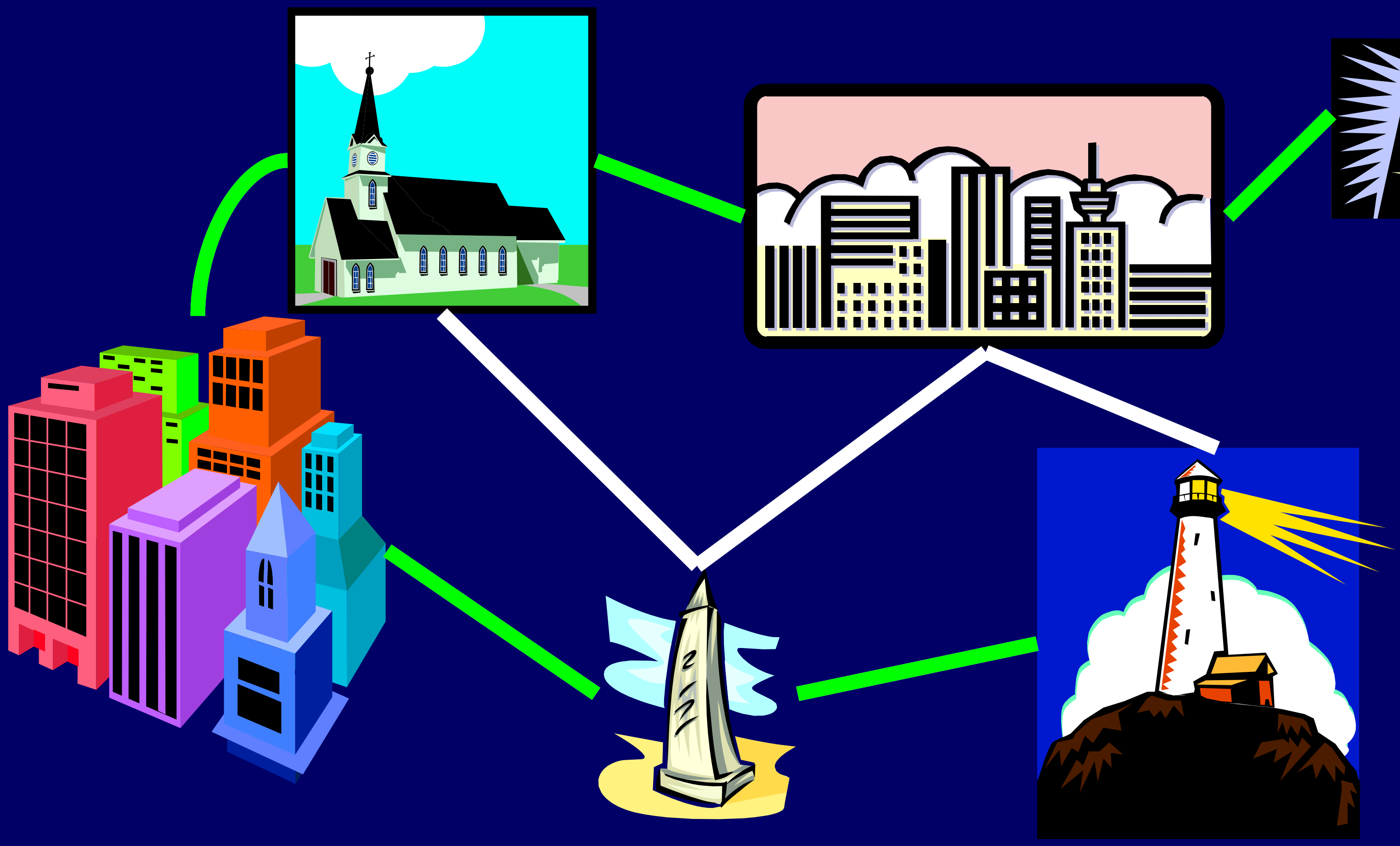
Pacotes Turísticos

Problema Planejar uma viagem para visitar cada lugar exatamente uma vez.



Solução para o Exemplo

Problema Planejar uma viagem para visitar cada lugar exatamente uma vez.



Algoritmo Ingênuo/Simples (Backtracking)

- Para cada lugar
  - **Tente todos os lugares alcançáveis que não foram visitados ainda.**
  - **Repita o processo até não conseguir mais.**

Quanto tempo levaria? (no pior caso)

lugares	passos
n	$n!$
5	120
15	1307674368000
100	$\approx 9 \cdot 10^{157}$

Se o seu computador é capaz de executar 10.000 instructions por segundo, isto levará 4 anos!

É um Problema Tratável?

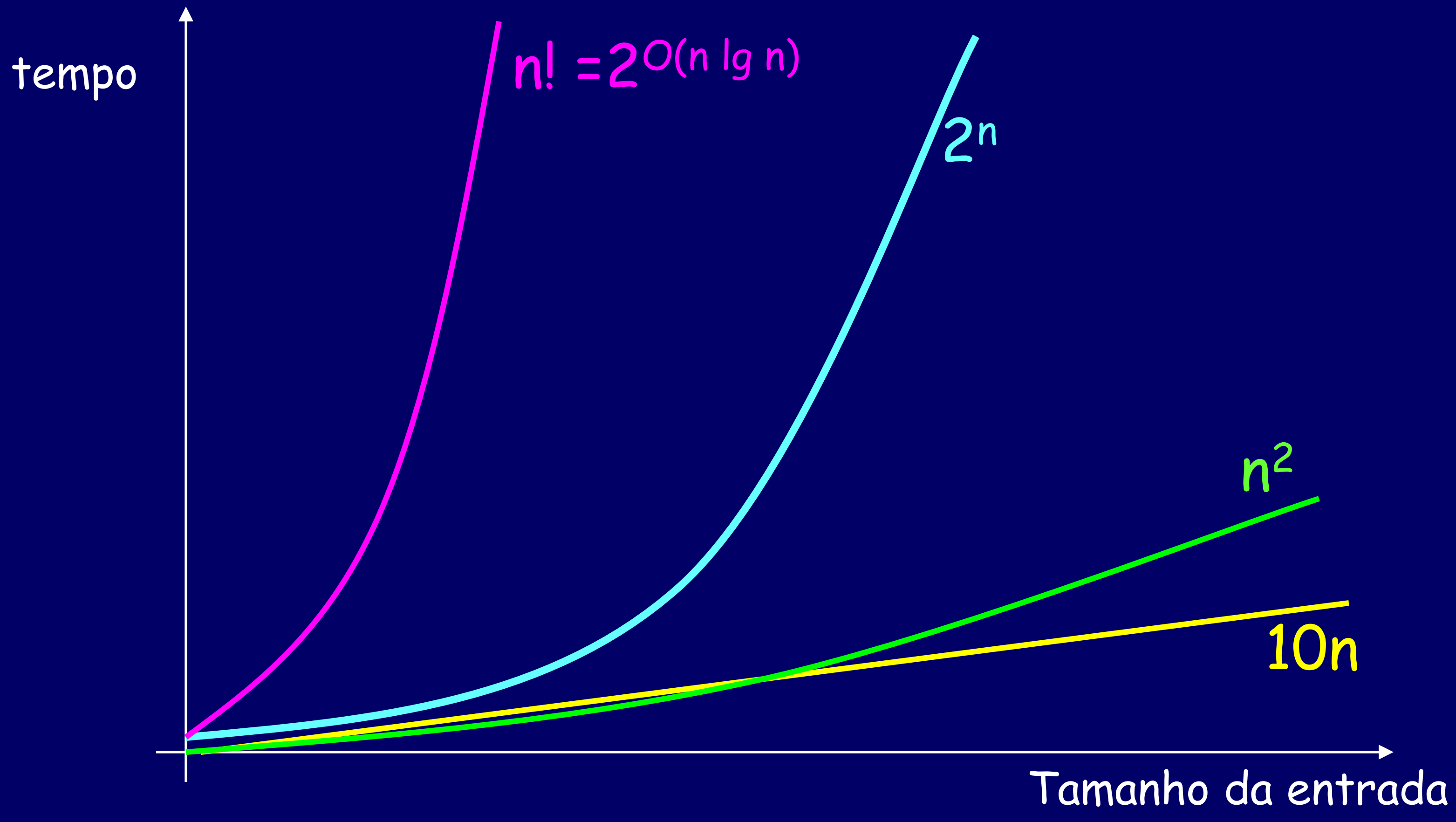
- Sim! E aqui está um algorítmo eficiente para ele!
- Não! E eu posso provar! Observe que provar a intratabilidade inerente de problemas é difícil. Torre de Hanoi é em exemplo de problema provadamente intratável, de tempo exponencial.

E se ficamos no meio do caminho??

Tarefa para a próxima aula...

- Você pode projetar um algoritmo eficiente para o problema do pacote turístico, considerando que não há ciclos entre lugares?

Taxa de crescimento de certas funções



Onde fica N^n ?

O Mundo de acordo com a Complexidade

Tempo razoável/tratável

polinomial $\equiv n^k$

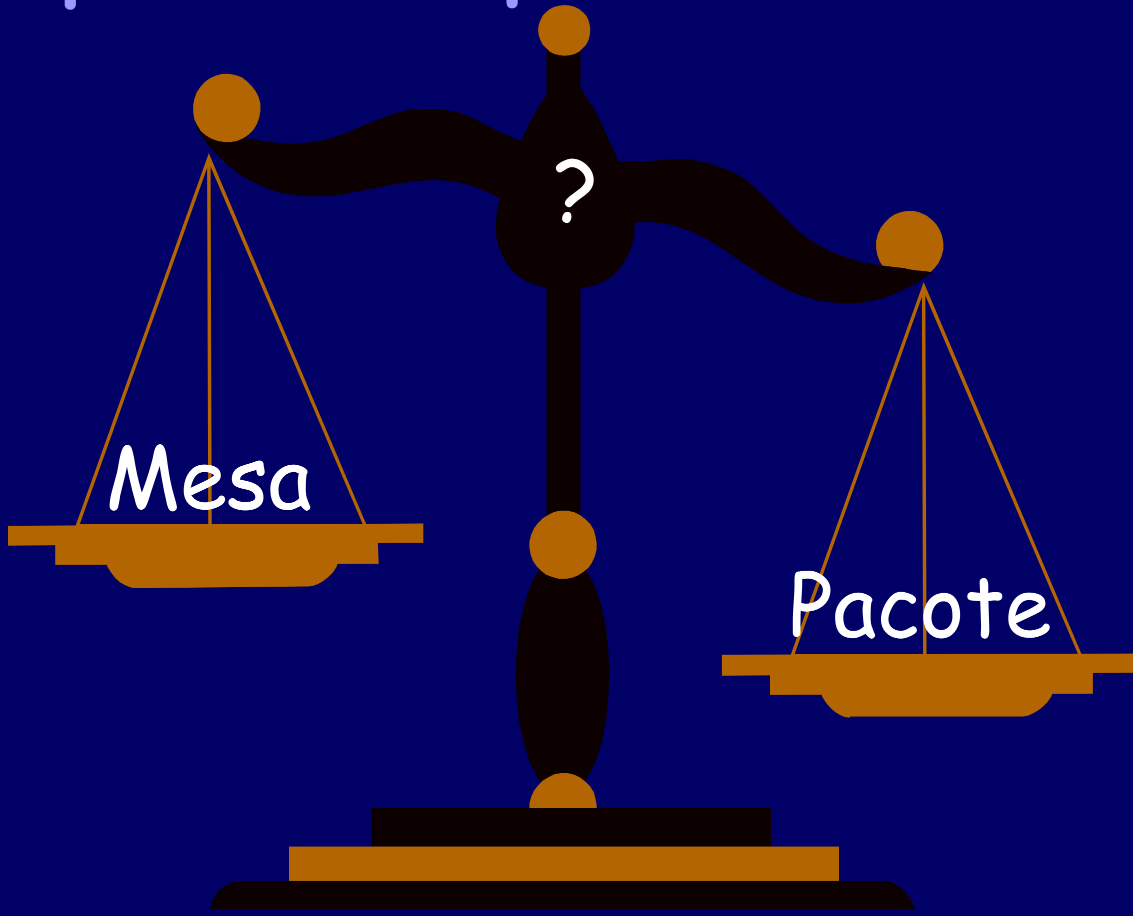
Algoritmo requerendo, no pior caso, tempo $\log n$, n, $n^2 \ldots n^k$ para um k fixo

Tempo Não-razoável /Intratável

exponencial $\equiv 2^n$

Algoritmo requerendo, no pior caso, tempo 5^n, $n!$, n^n ... e outros super-polinomiais

Um algoritmo pode ser muito pior do que outro?



Reduções Polinomiais

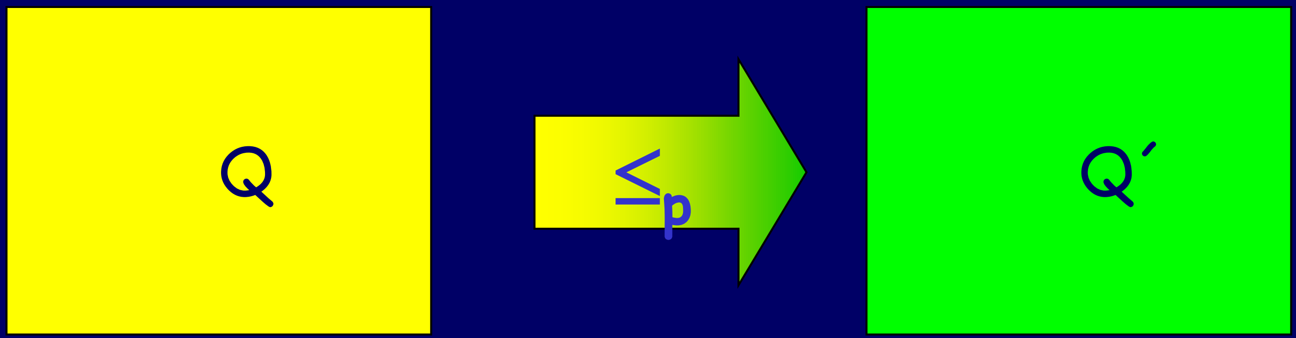
Intuitivamente, um problema Q pode ser reduzido a outro problema Q´se qualquer instância de Q pode ser facilmente refraseada (usada para resolver) como uma instância de Q.

Por exemplo, o problema de resolver equações lineares em x se reduz ao problema de resolver equações quadráticas.

Dada uma instância $ax + b = 0$, nós transformamos ela para $0x^2 + ax + b = 0$, cuja solução fornece uma solução para $ax + b = 0$.

Assim, se um problema Q se reduz a outro problema Q´, então Q não é tão mais difícil de resolver do que Q´

Reduções Polinomiais



Q não pode ser radicalmente mais difícil que Q´,
pois a diferença de Q para Q´ é de um fator polinomial

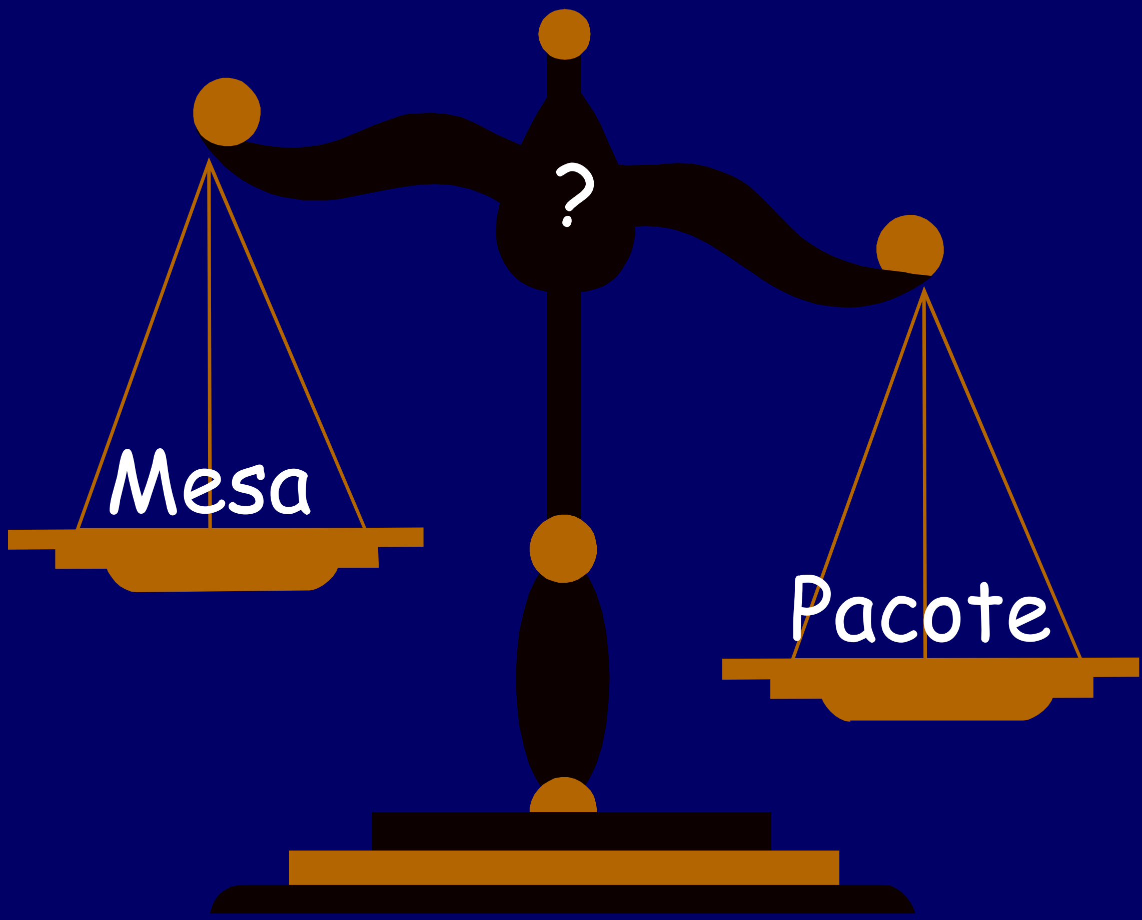
Em outras palavras: Q´ é pelo menos tão difícil quanto Q

Reduções Polinomiais

- Formalmente, redução polinomial de um problema Q a um outro problema *P*, é um
- algoritmo polinomial que transforma uma instância *x* de Q em um instância *y* de *P*, de forma que:

- Q(*x*) = "SIM" se e somente se *P*(*y*) ="SIM".

Qual é o mais difícil?

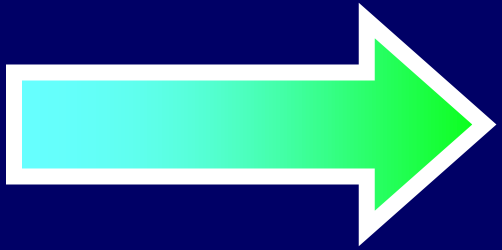


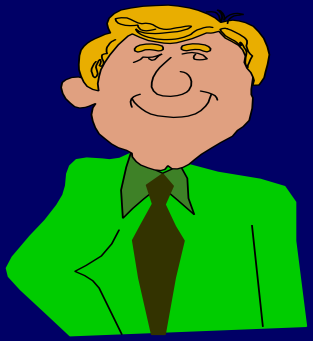
Reduzir Pacote para Mesa

1ª Observação: Os problemas não são tão diferentes

lugar

convidado





"diretamente alcançável de …"

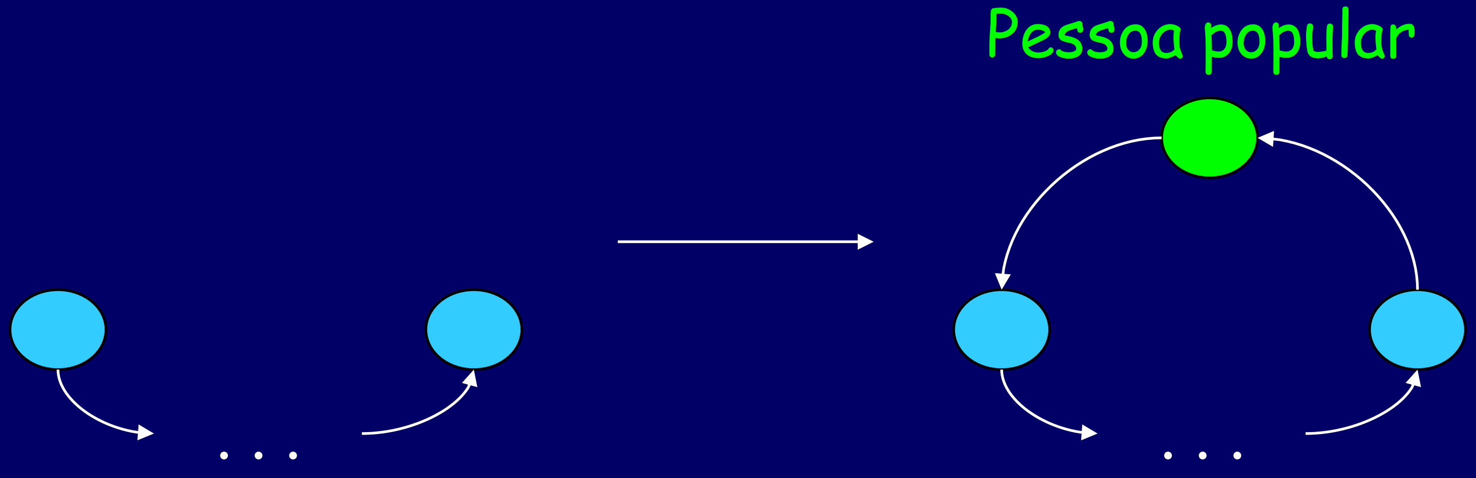
"querido por…"

Reduzir Pacote para Mesa 2a Observação: Completando o círculo

- Vamos convidar para nossa festa uma pessoa popular, i.e uma que possa sentar perto de qualquer um.

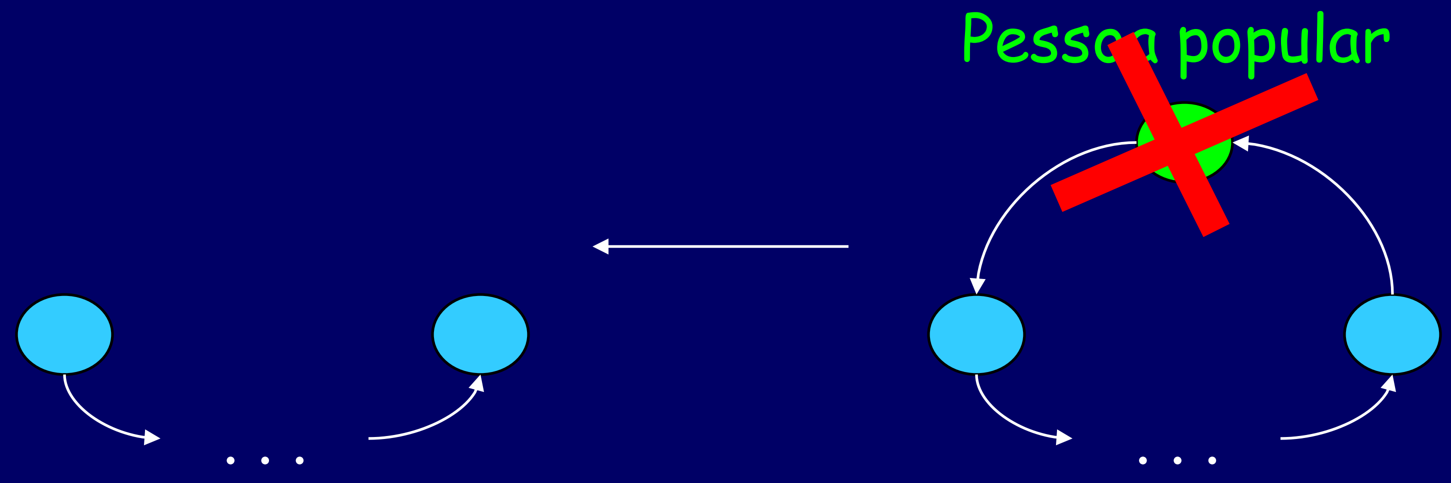
Reduzir Pacote para Mesa

- Se existe uma viagem, existe também uma forma de sentar todos os convidados ao redor da mesa.



Reduzir Pacote para Mesa

- Se existe uma forma de sentar, nós podemos facilmente encontrar uma viagem (sem viagem, sem solução para a mesa).



Conclusão

O problema da mesa é pelo menos tão difícil quanto o problema do pacote

- Embora não apresentamos um algoritmo eficiente para os problemas
- Nem provamos que eles não tem um
- Nós conseguimos mostrar uma afirmação muito poderosa relacionada com a dificuldade entre eles

- Veja que nós também podemos reduzir o problema da mesa ao problema do pacote.

- Além disso, existe uma grande classe de problemas que podem ser reduzidos eficientemente um a outro.

Gerenciamento de filas para uso de CPU
(escalonamento)

Satisfatibilidade

Caixeiro Viajante

NPC

Caminho/Circuito
Hamiltoniano

Layout de
VLSI´s

Possue centenas de problemas diferentes

Cada um reduzível a todos os outros

- ☑ algorítmos exponenciais
- ☐ algorítmos eficientes

P =? NP

O primeiro problema NPC provado

- Em 1971, o problema da satisfatibilidade para o cálculo proposicional (fórmula booleana) foi provado estar em NPC (Teorema de Cook) fornecendo um começo para outras provas de problemas NPC

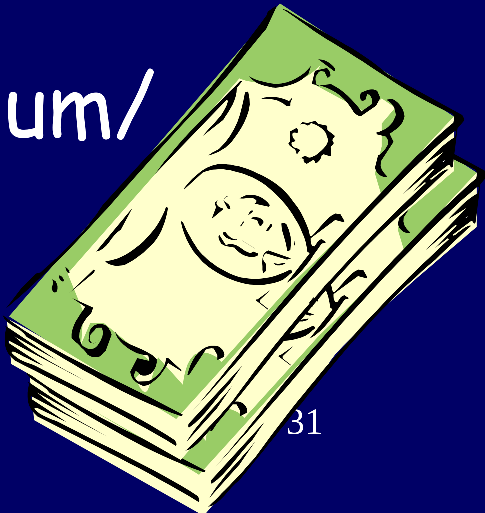
Uma fórmula f é válida quando for verdadeira em todas as suas interpretações.

Uma fórmula f é satisfatível se é verdadeira em alguma interpretação.

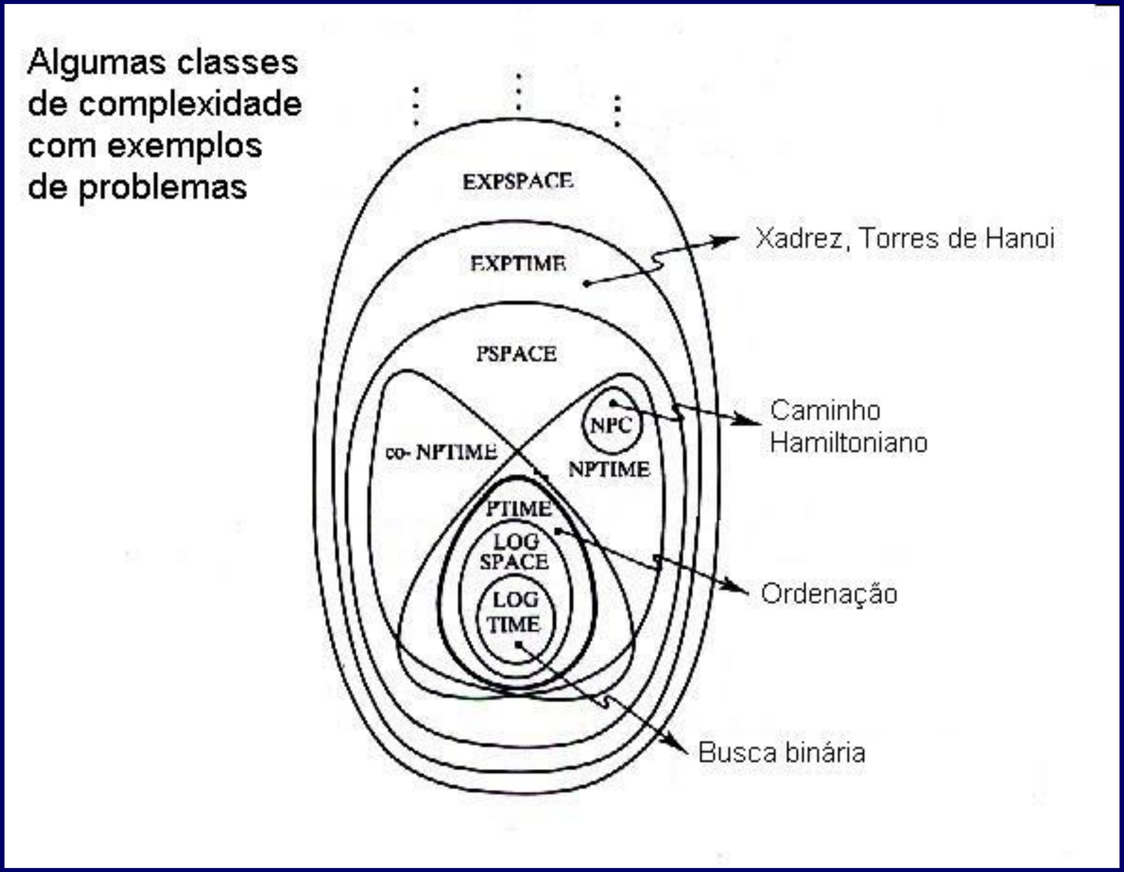
Uma interpretação é um mapeamento dos símbolos de f em {V,F}.

Como o estudo da Complexidade de algoritmos pode torná-lo milionário?

- P =? NP é a questão aberta mais fundamental da Ciência da Computação.
- Resovê-la pode torná-lo famoso
- … bem como rico…
  http://www.claymath.org/millennium/



"Testar se N é Primo" e o uso de primos em algoritmos de criptografia



Primos está em P (6/agosto/2002)

Antiga posição de Primos:

Co-NP inter NP

RSA/chaves públicas usam inteiros (chave) que são o produto de 2 primos de 128/512 bits. Precisamos fatorar a chave, que é mais difícil do que testar se um número é primo.

http://www.flonnet.com/fl1917/19171290.htm

http://en.wikipedia.org/wiki/AKS_primality_test

http://en.wikipedia.org/wiki/Primality_test

Algoritmo para geração da chave
http://www.di-mgt.com.au/rsa_alg.html#keygen

1) Generate two large random primes, p and q, of approximately equal size such that their product *n = pq* is of the required bit length, e.g. 1024 bits.
2) Compute n = pq and (φ) phi = (p-1)(q-1).
3) Choose an integer e, 1 < e < phi, such that gcd(e, phi) = 1.
4) Compute the secret exponent d, 1 < d < phi, such that ed ≡ 1 (mod phi).
5) The public key is (n, e) and the private key is (n, d). The values of p, q, and phi should also be kept secret.

- n is known as the *modulus*.
- e is known as the *public exponent* or *encryption exponent*.
- d is known as the *secret exponent* or *decryption exponent*.

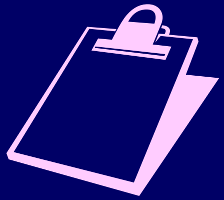
Encryption

Sender A does the following:-
Obtains the recipient B's public key (n, e).
Represents the plaintext message as a positive integer m.
Computes the ciphertext c = m^e mod n.
Sends the ciphertext c to B.

Decryption

Recipient B does the following:-
Uses his private key (n, d) to compute m = c^d mod n.
Extracts the plaintext from the integer representative m.

Sumário



- Nós trabalhamos com 2 problemas:
  - O problema da Mesa (CICLO-HAMILTONIANO)
  - O problema do Pacote (CAMINHO-HAMILTONIANO)

Sumário

- Embora dissemos pouco sobre eles, nós conseguimos mostrar que as suas dificuldades computacionais são relacionadas.

- Além disso, dissemos que eles fazem parte de uma grande classe de problemas, chamada NPC = NP completo.

- Se conseguírmos um algorítmo polinomial para algum deles então resolveremos todos de forma polinomial, isto é, P = NP. Esta questão ainda está aberta.

- Enquanto não temos este algoritmo polinomial veremos uma forma de “vencer” a complexidade deles: relaxar a exigência e usar algoritmos aproximados e heurísticos

Exemplos de problemas NP-C

- Escolham problemas menos CLIQUE – CONJUNTO INDEPENDENTE de VÉRTICES

Exemplos:

- 3SAT,
- NAESAT (not-all-equal SAT),
- Cobertura de vértices,
- CAM HAM (ou CICLO HAM),
- podem também escolher CAIXEIRO VIAJANTE pela importante prática deste e de como vem sendo resolvido com heurísticas (embora seja um exemplo de CAM-CICLO HAM – as vezes a prova usa problemas diferentes para a redução),
- mochila (knapsack),
- soma de subconjuntos,
- 3-coloring (pode um mapa ser colorido com 3 cores?),
- Partição em grafos (cut), etc.