Algoritmos e Estruturas de Dados

Marcelo Lobosco
DCC/UFJF

Classes de Problemas

Parte 2 - Aula 07

Agenda

- Classes de Problemas
  - Problemas NP-Completos

Introdução

- Maioria dos algoritmos apresentados são algoritmos de tempo polinomial
  - Sobre entradas de tamanho n, tempo de execução no pior caso é $O(n^k)$, para alguma constante k
  - Problemas tratáveis
- Pergunta: todos os problemas podem ser resolvidos em tempo polinomial?
  - Não, alguns nem podem ser resolvidos...
  - ...como o problema de parada de Turing
    - Dada a descrição de um programa de computador e uma entrada, decidir se o programa pára de executar ou executa para sempre
  - … enquanto para outros nenhum algoritmo polinomial resolve o problema: intratáveis

Introdução

- Tópico da aula de hoje: problemas cujo “status” desconhecido
  - NP-Completo
  - Não se sabe se é polinomial ou não
  - Não foi descoberto algoritmo de tempo polinomial, nem se provou que não exista
  - Questão chamada de P $\neq$ NP
- Aspecto torturante dos problemas NP-Completos: vários deles semelhantes a problemas que têm algoritmos de tempo polinomial

Introdução

- Caminho mais curto e mais longo
  - Podemos encontrar caminhos mais curtos a partir de uma única origem em um grafo orientado G = (N,A) com tempo O(N.A)
  - Porém encontrar o caminho mais longo entre dois vértices é problema NP-Completo
- Caminho Euleriano e Hamiltoniano
  - Podemos encontrar as arestas da viagem de Euler (percorrer cada aresta exatamente uma vez, embora podendo visitar um vértice mais de uma vez) no tempo O(A)
  - Ciclo Hamiltoniano (ciclo simples que contém cada vértice em N) é NP-Completo

Introdução

- Satisfabilidade 2-CNF e 3-CNF
  - Verifica se fórmula booleana na forma normal k-conjuntiva é avaliada como 1 para alguma atribuição de 0's e 1's para suas variáveis.
  - Forma 2-CNF: OR com dois termos
  - Forma 3-CNF: OR com três termos
  - Algoritmo polinomial para determinar se fórmula de 2-CNF é capaz de satisfação
  - Mas determinar se fórmula 3-CNF é capaz de satisfação é NP-completo

Caráter NP-completo e classes P e NP

- Classe P consiste dos problemas que podem ser resolvidos em tempo polinomial ($O(n^k)$)
- Classe NP consiste dos problemas que são verificáveis em tempo polinomial
- Qualquer problema em P também está em NP
  - Podemos resolvê-lo em tempo polinomial sem sequer ter um certificado
  - $P \subseteq NP$

Caráter NP-completo e classes P e NP

- Problema NP-Completo se está em NP e é tão difícil quanto qualquer problema em NP
  - Definiremos formalmente o que significa ser tão difícil quanto qualquer problema em NP mais adiante
  - Se qualquer problema NP-completo pode ser resolvido em tempo polinomial, então todo problema NP-completo tem um algoritmo de tempo polinomial
  - Muitos teóricos acha que problemas NP-completos são intratáveis
  - Contudo grande esforço para provar que problemas NP-Completos são intratáveis ainda não chegou a resultado conclusivo

Introdução

- Porque estudar problemas NP-Completos?
- Se você puder determinar que um problema é NP-Completo, melhor escolha para implementação é:
  - Utilizar algoritmo de aproximação ou
  - Resolver um caso especial tratável

Visão Geral das Técnicas Para Mostrar que Problema é NPC

- Diferente das técnicas empregadas ao longo do curso para avaliar complexidade de algoritmos
- Não queremos mostrar o quão fácil é resolver esse problema, mas o quanto consideramos difícil resolvê-lo
- Três conceitos fundamentais para mostrar que problema é NP-Completo
  - Problemas de decisão
  - Problemas de otimização
  - Reduções

Problemas de Decisão versus Problemas de Otimização

- Problema de otimização: cada solução possível (válida) tem um valor associado, e desejamos encontrar a solução com melhor valor
  - Exemplo: problema do menor caminho
- Caráter NP-Completo não se aplica diretamente a problemas de otimização, mas a problemas de decisão
  - Resposta simplesmente “sim” ou “não”

Problemas de Decisão versus Problemas de Otimização

- Contudo existe relacionamento entre problemas de otimização e de decisão
  - Problema de otimização pode ser formulado como problema de decisão relacionado, impondo limite sobre valor a ser otimizado
  - Por exemplo, existe caminho entre u e v consistindo em no máximo k arestas?
- Relacionamento utilizado para mostrar que problema de otimização é difícil
  - Problema de decisão é de certo modo “mais fácil” ou, pelo menos, “não mais difícil”
  - Podemos resolver problema de decisão resolvendo problema de otimização, e depois comparando resultado obtido com valor k

Problemas de Decisão versus Problemas de Otimização

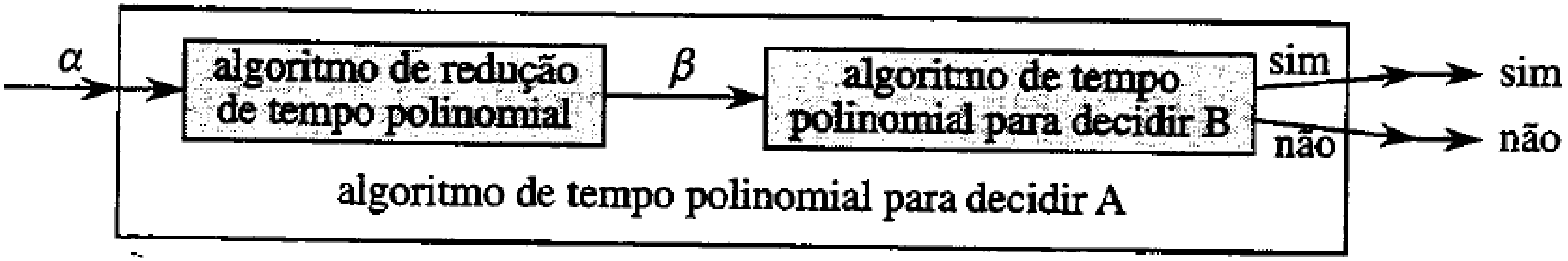
- Ou seja, se problema de otimização é fácil, seu problema de decisão relacionado também é facil
- Assim, se pudermos fornecer evidências de que problema de decisão é difícil, também fornecemos evidências de que seu problema de otimização relacionado é difícil
  - Apesar de restingirmos a atenção para problemas de decisão, a teoria de problemas NP-completos frequentemente tem implicações para problemas de otimização

Reduções

- Noção anterior de mostrar que problema não é mais fácil/difícil que outro se aplica até quando ambos os problemas são de decisão
- Tiramos proveito dessa idéia em quase todas as provas do caráter NP-Completo
  - Vamos considerar um problema de decisão, p.ex. A, que gostaríamos de resolver em tempo polinomial
  - Chamamos a entrada para um determinado problema de instância
  - Agora suponha que exista um problema de decisão diferente, B, que já sabemos como resolver em tempo polinomial
  - Finalmente suponha que temos procedimento que transforma qualquer instância α de A em alguma instância β de B

Reduções

- Características da transformação
  - Demora tempo polinomial
  - Respostas são as mesmas: resposta para α de A é sim se e somente se a resposta para β de B também for sim
- Algoritmo de redução



Reduções

- Como cada etapa do algoritmo demora tempo polinomial, as três juntas também demoram tempo polinomial
  - Ou seja, reduzindo a solução do problema A à solução do problema B, usamos a facilidade de B para provar a facilidade de A
  - No caso dos problemas NP-Completos, queremos mostrar o contrário, o quanto o prolema é difícil
  - Suponha que já tenhamos algoritmo A para o qual já sabemos que não existe nenhum algoritmo de tempo polinomial
  - Suponha ainda que temos redução em tempo polinomial transformando instâncias de A em instâncias de B

Reduções

- Prova agora é simples para mostrar que problema A é NP-Completo: usando contradição
  - Suponha que exista algoritmo de tempo polinomial para B
  - Então, usando o método da figura anterior, teríamos um método para resolver A em tempo polinomial
  - Isso contradiz nossa hipótese de que não existe nenhum algoritmo de tempo polinomial para A
  - Naturalmente que no caso do problema NP-Completo não podemos supor que não exista absolutamente nenhum algoritmo de tempo polinomial para A
  - Mas prova semelhante: problema B é NP-Completo se problema A também for

Primeiro Problema NP-Completo

- Como técnica de redução se baseia em ter um problema já conhecido como NP-Completo para provar que outro problema diferente também é, precisamos de um primeiro problema NP-Completo
  - Problema da satisfabilidade de circuitos
  - Circuito combinacional booleano composto por AND, OR e NOT, e desejamos saber se existe conjunto de entradas que faça saída ser 1

Tempo Polinomial

- Vamos formalizar noção de problemas que podem ser resolvidos em tempo polinomial
- Considerados tratáveis por questões filosóficas
  - Poucos problemas práticos exigem tempo na ordem de um polinômio de grau alto (p.ex., $\Theta(n^{100})$)
  - Além do mais, após um algoritmo divulgado pela primeira vez, surgem na sequência algoritmos mais eficientes

Problemas Abstratos

- Para entender a classe de problemas de tempo polinomial, primeiro vamos formalizar noção de problemas
- Definimos problema abstrato Q como uma relação binária sobre conjunto I de instâncias de problemas e um conjunto S de soluções de problemas
  - Por exemplo, uma instância do caminho mais curto é um grafo e dois vértices
  - Uma solução é uma sequência de vértices no grafo
  - Problema do caminho mais curto em si é relação que associa cada instância de um grafo e dois vértices com um caminho mais curto no grafo que conecta os dois vértices

Problemas Abstratos

- Formulação mais geral que o necessário, já que teoria de problemas NP-Completos restringe atenção a problemas de decisão
- Podemos ver um problema de decisão abstrato como uma função que mapeia conjunto I de instâncias de problemas para o conjunto de solução {0,1}

Codificações

- Se programa de computador deve resolver problema abstrato, instâncias de problemas devem ser representadas de modo que programa reconheça
- Codificação de conjunto S de objetos abstratos é um mapeamento *e* de S para o conjunto de cadeias binárias
- Por exemplo, codificação dos naturais N = {0, 1, 2, 3...} como cadeias {0, 1, 10, 11, 100, …}
- Objeto composto pode ser representado pela combinação das representações de suas partes constituintes

Codificações

- Desse modo, algoritmo que resolve algum problema de decisão abstrato na realidade toma codificação de uma instância de problema como entrada
- Chamamos de problema concreto um problema cujo conjunto de instâncias é o conjunto de cadeias binárias
- Dizemos que algoritmo resolve um problema concreto no tempo $O(T(n))$ se, quando fornecida instância i de comprimento $n=|i|$, o algoritmo pode produzir a solução no tempo máximo $O(T(n))$
- Então problema concreto pode ser resolvido em tempo polinomial se existe algoritmo para resolvê-lo no tempo $O(n^k)$ para alguma constante k

Codificações

- Podemos agora definir formalmente classe de complexidade P como o conjunto de problemas de decisão concretos que podem ser resolvidos em tempo polinomial.
- Codificações podem ser usadas para mapear o problema abstrato em um problema concreto
- Dado um problema de decisão abstrato Q que mapeia um conjunto de instâncias I para {0,1}, uma codificação *e*: $I \rightarrow \{0,1\}^*$ pode ser usada para induzir um problema de decisão concreto relacionado *e*(Q)
- Se solução para instância de problema abstrato $i \in I$ é $Q(i) \in \{0,1\}$, então a solução para a instância de problema concreto $e(i) \in \{0,1\}^*$ também é Q(i)

Introdução

- Objetivo da aula: avaliar a dificuldade intrínseca à natureza de cada problema
- Algoritmo dito eficiente se puder ser resolvido por um algoritmo polinomial no tamanho da sua entrada
  - Critério de Edmonds
- Pergunta: dado um determinado problema, existe algoritmo que o resolva em tempo polinomial?
  - O fato do algoritmo não existir hoje não implica que tal algoritmo não possa ser obtido posteriormente
  - Necessária classificação que independa dos algoritmos disponíveis

Introdução

- Problemas com cota inferior exponencial
  - Classificados como intratáveis
  - Impossível a obtenção de algoritmos com solução exata em tempo polinomial
- Vários problemas com cota inferior polinomial são resolvidos através de algoritmos exponenciais
  - Existem algoritmos polinomiais para estes problemas?
- Problemas que podem ser resolvidos por algoritmos polinomiais no tamanho de sua entrada
  - Chamados de tratáveis
  - Pertencem a classe P dos problemas algoritmos

Problemas de Decisão

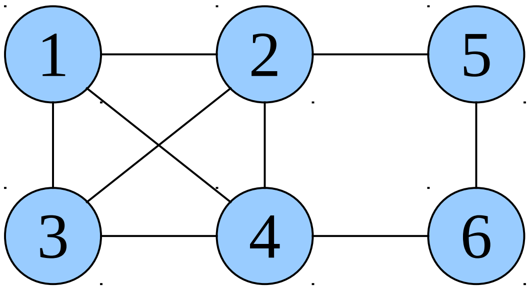
- Diversos problemas classificados em três categorias
  - De acordo com as características do problema apresentado
    - Decisão
      - Desejamos responder sim ou não a uma indagação
        - Ex: Desejamos saber se existe ou não um circuito hamiltoniano em um grafo com custo inferior a k
    - Localização
      - Estamos interessados na determinação de uma certa estrutura satisfazendo a algumas restrições
        - Exibir o circuito hamiltoniano com custo inferior a k
    - Otimização
      - Desejamos encontrar estrutura de forma a minimizar ou maximizar função objetivo associada ao problema
        - Circuito hamiltoniano de menor custo

Problema de Decisão

- Ao resolver problema de localização resolvemos, implicitamente, problema de decisão
  - Mesmo para problema de otimização e localização
  - De forma geral, podemos dizer que problema de decisão tem mesmo grau de dificuldade que problema de localização ou otimização
  - Grau de dificuldade inerente a cada problema poderá ser simplificada analisando-se apenas os problemas de decisão
    - Se problema de decisão não pode ser resolvido em tempo polinomial, o problema de otimização associado também não poderá ser resolvido

Problema de Decisão

- Exemplos de problemas de decisão para os quais não se conhece nenhum algoritmo polinomial que os resolva:
  - Problema do clique:
    - Clique: Um subgrafo C de G será um clique se, para qualquer par de vértices i e j de C, a aresta (i,j) pertencer a G
    - Suponha que sejam dados um grafo G e um inteiro k>0
      - Existirá em G um clique de tamanho maior ou igual a k?

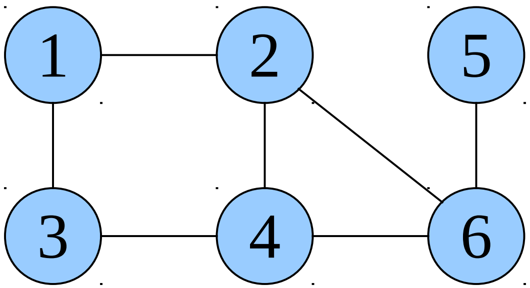


k = 4? Sim

k = 5? Não

Problema de Decisão

- Exemplos (cont.)
  - Conjunto independente de vértices
    - Dado um grafo não orientado G(N,A), dizemos que $N' \subseteq N$ é um subconjunto independente de vértices de G se, para todo par de vértices u,v em V', a aresta (u,v) não pertence a A
    - Dado um grafo G qualquer, existirá algum conjunto independente de vértices de tamanho maior ou igual a k?



$k \leq 3$? Sim

$k \geq 4$? Não

Algoritmos Não-Determinísticos

- Algoritmos determinísticos: o resultado de cada operação é definido de maneira única
- Algoritmos não determinísticos removem esta restrição
  - Supõem a existência de algumas operações cuja saída não é definida de forma única
    - Limitada por um conjunto pré-definido de possibilidades
- No modelo não-determinístico, além dos comandos determinísticos usuais, temos três novas funções
  - Escolha, Fracasso e Sucesso

Algoritmos Não-Determinísticos

- Escolha(S) retorna um elemento do conjunto S
  - Não existe regra que especifique como a escolha é realizada
- Sucesso e Fracasso sinalizam o final do processamento com sucesso e fracasso, respectivamente
  - Um algoritmo não determinístico termina em Fracasso se e somente se não existir nenhum conjunto de escolhas que nos leve a um término com Sucesso
- Tempo de processamento para as funções: O(1) passos

Algoritmos Não-Determinísticos

- Interpretação determinística de um algoritmo não-determinístico
  - Suponha paralelismo ilimitado
  - Cada vez que função Escolha executada, algoritmo faz cópias de si mesmo
    - Cada uma delas correspondendo a uma possível escolha
    - Cada cópia executada em paralelo
    - Primeira cópia que obter Sucesso cessa imediatamente execução das demais cópias
    - Cópia que obter Fracasso cessa apenas própria execução
    - Implementação pode ser feita através de algoritmo determinístico probabilístico

Algoritmos Não-Determinísticos

- Exemplos de algoritmos não determinísticos:
  - Problema da busca em uma lista desordenada
    - A: lista desordenada com n elementos
    - x: elemento a ser encontrado nessa lista
    - Algoritmo:

```
Inicio
J = Escolha (1,...n)
Se A[j] = x então
   imprima (J);
   Sucesso;
fim
imprima (0);
```

Algoritmos Não-Deteminísticos

- Complexidade do Algoritmo: O(1)
  - Qualquer outro algoritmo determinístico para o mesmo problema terá limite inferior dado por $\Omega(n)$.
- Problema da ordenação de n elementos
  - A: lista desordenada de n elementos inteiros
  - Algoritmo:

```
Inicio
B = 0; // vetor B inicializado
para i = 1 até n faça
    j = Escolha (1,...,n);
    Se B[j] != 0 então Fracasso;
    B[j] = A[i];
fim
```

Algoritmos Não-Determinísticos

```
para i = 1 até n-1 faça
    se B[i] > B[i+1] então Fracasso;
fim
imprime B;
Sucesso;
Fim
```

- Complexidade: $O(n)$
  - Em qualquer algoritmo determinístico para o mesmo problema teremos um limite inferior dado por $\Omega(n\log n)$

Classes P e NP dos Problemas de Decisão

- Dizemos que problema π pertence a classe NP dos problemas de decisão se tivermos um algoritmo não determinístico polinominal para π
- Um problema pertence a classe P se existir algum algoritmo que o resolva em tempo polinomial
- Vimos que um algoritmo determinístico pode ser visto como um caso particular de um algoritmo não-determinístico

$$P \subseteq NP$$

Classes P e NP dos Problemas de Decisão

- Será entretanto que P está contido propriamente em NP?
  - Um dos problemas mais famosos da área de Ciência da Computação (P = NP ou P ≠ NP?)
    - Problema ainda em aberto
      - Se existem algoritmos polinomiais deterministicos para todos os problemas em NP, então P = NP
      - Por outro lado, a prova de que P ≠ NP parece exigir técnicas ainda desconhecidas
    - Vale US$1 milhão! (Pago pelo instituto Clay)
- Acredita-se que NP>>P
  - Para muitos problemas em NP, não existem algoritmos polinomiais conhecidos, nem um limite inferior não-polinomial provado

Classes P e NP dos Problemas de Decisão

- Muitos problemas práticos em NP podem ou não pertencer a P (não conhecemos nenhum algoritmo deterministico eficiente para eles)
- Se conseguirmos provar que um problema não pertence a P, então não precisamos procurar por uma solução eficiente para ele
  - Como não existe tal prova, sempre há esperança de que alguém descubra um algoritmo eficiente
- Quase ninguém acredita que NP = P
  - Existe um esforço considerável para provar o contrário, mas a questão continua em aberto!

Classes P e NP dos Problemas de Decisão

- Para verificar se um problema de decisão π pertence a classe NP, duas etapas devem ser executadas
  - Exibição e reconhecimento
- Na etapa da exibição, apresentamos uma justificativa à resposta “sim” para π cuja etapa de reconhecimento seja realizada por um algoritmo polinomial no tamanho de sua entrada
  - No caso de termos um reconhecimento exponencial, não podemos afirmar se π pertence ou não a NP

Classes P e NP dos Problemas de Decisão

- Exemplo: Ciclo Hamiltoniano
  - Suponha G=(N,A) um grafo conexo e não-orientado
  - Algoritmo:

```
Inicio
Solução = Vazio; // Inicializa lista solução
Solução[1] = 1; // nó inicial
para i = 2 até N faça
J = Escolha (2,...,N);
Solução[i] = J;
fim
para i = 1 até N-1 faça
para j = i+1 até N faça
    se Solução[i] = Solução [j] então Fracasso;
fim
fim
Sucesso;
Fim
```

Classes P e NP dos Problemas de Decisão

- Etapa de exibição: escolhe aleatoriamente nós para o conjunto solução
- Etapa de reconhecimento: verifica se a solução é válida
  - Realizada por um algoritmo polinomial de ordem $O(n^2)$
  - Podemos portanto dizer que o problema do ciclo hamiltoniano pertence à classe NP dos problemas de decisão

Classes P e NP dos Problemas de Decisão

- Exemplo: Problema da Satisfatibilidade
  - Seja E uma expressão booleana na forma normal conjuntiva (FNC)
    - Forma normal conjuntiva (CNF): conjunção ($\wedge$) de cláusulas
      - Cláusula: disjunção ($\vee$) de literais
      - Literal: variável simples ou negada (~)
      - Exemplo: $(A \vee \sim B \vee \sim D \vee \sim F) \wedge (C \vee B \vee \sim D) \wedge (\sim A \vee F \vee D)$
  - No máximo n literais em cada cláusula
  - Problema: verificar se a fórmula CNF é satisfazível, isto é, se existe uma atribuição de valores lógicos às variáveis que torna a fórmula verdadeira

Classes P e NP dos Problemas de Decisão

- Vetor X de n elementos armazena as atribuições feitas a cada literal
- Algoritmo:

Inicio

Leia (E);

Para I = 1 até N faça

X[I] = Escolha (V,F);

Fim

Se E(X[1],...,X[N]) então Sucesso;

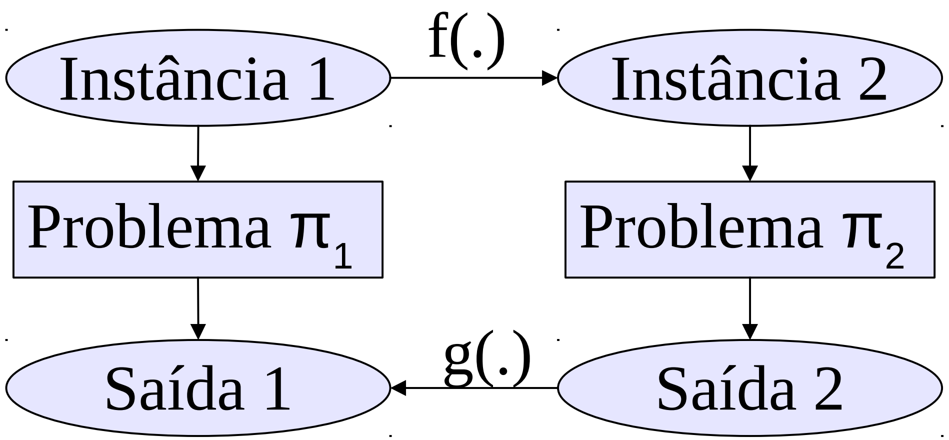
Fracasso;

Fim

- Complexidade do algoritmo SAT não determinístico: O(n)
  - SAT pertence à classe NP dos problemas de decisão

Problemas NP-Árduos e NP-Completos

- Conceito de redução
  - Um problema π_1 é redutível a um problema π_2 se, para qualquer instância de π_1, uma instância de π_2 poderá ser construída em tempo polinomial tal que resolvendo π_2 resolve-se implicitamente π_1.
    - Notação: $\pi_1 \, \alpha \, \pi_2$



Problemas NP-Árduos e NP-Completos

- Informalmente falando: redutibilidade de π_1 a π_2 implica que π_1 pode ser considerado caso particular de π_2
  - Desta forma, π_2 será pelo menos tão difícil quando π_1
  - Quando $\pi_1 \alpha \pi_2$ e $\pi_2 \alpha \pi_1$, dizemos que ambos os problemas são equivalentes
  - Redutibilidade é função transitiva
    - $\pi_1 \alpha \pi_2$ e $\pi_2 \alpha \pi_3$, então $\pi_1 \alpha \pi_3$

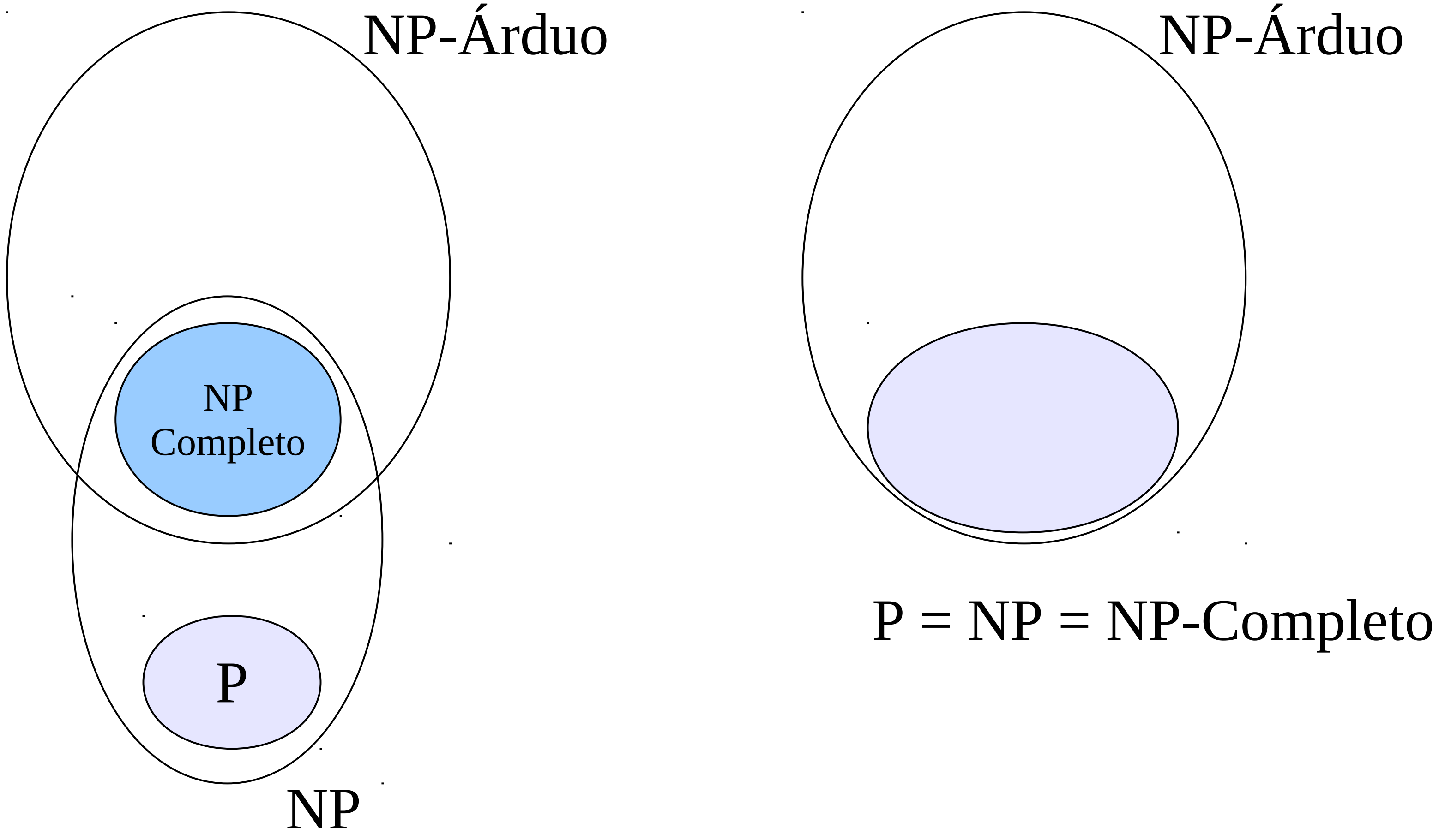
Problemas NP-Árduos e NP-Completos

- Problema NP-Árduo
  - Um problema de decisão π é NP-árduo se π' é redutível a π qualquer que seja π' pertencente à classe NP
- Proposição (Teorema de Cook)
  - Todos os problemas em NP podem ser reduzidos polinomialmente a um único problema de lógica denominado problema da satisfatibilidade (SAT)
- Problema NP-Completo
  - Um problema de decisão π é NP-Completo se for NP-Árduo e pertencer a NP

Problemas NP-Árduos e NP-Completos

- Proposição: Se um problema de decisão π_1 é redutível a outro problema de decisão π_2 e π_2 está na classe P, então π_2 pertence a classe P
- Proposição: Seja π um problema NP completo. O problema π pertence a P se e somente se P = NP
  - Se NP admitir um único problema que possa ser resolvido por um algoritmo polinomial, então todos os problemas em NP admitirão também um algoritmo polinomial em sua resolução

Problemas NP-Árduos e NP-Completos



Problemas NP-Árduos e NP-Completos

- Conjectura mais aceita atualmente: P ≠ NP
  - Neste caso mais conveniente utilizar técnicas heurísticas na abordagem de problemas NP-Completos
    - Estamos interessados em soluções aproximadas obtidas através de algoritmos determinísticos polinomiais

Outros Problemas NP-Completos

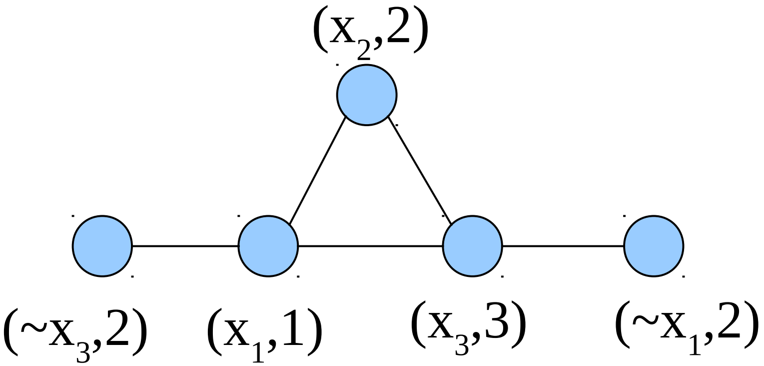
- Reduzindo um problema NP-Árduo conhecido a outro problema de decisão π, podemos afirmar, pela transitividade da redução, que π é NP-Árduo
  - Se ainda π pertencer a NP, teremos π NP-Completo
  - Segue portanto que problemas da classe NP-Completos são equivalentes entre si

Outros Problemas NP-Completos

- Proposição: SAT α Clique
  - Demonstração:
    - Seja B uma expressão booleana na FNC com variáveis lógicas x_i onde i = 1, ..., n
    - Devemos mostrar como construir a partir de B um grafo G=(N,A) tal que G terá uma clique de tamanho maior ou igual a k se B for satisfazível
    - Para uma expressão B qualquer, construimos G=(N,A) como abaixo:
      - N = { (x,i) / x é literal na cláusula i}
      - A = { [(x,i),(y,j)]/ x ≠ y e x ≠ ~y para i ≠ j}
    - Desta forma, se B é verdadeira haverá pelo menos um literal x em cada cláusula C_i, tal que x é verdadeiro
      - Assim se S = {(x,i) / x é verdadeiro em C_i}, S formará uma clique de tamanho k

Outros Problemas NP-Completos

- Exemplo:
  - Considere a seguinte expressão booleana na FNC
    - $E = (x_1) \wedge (\sim x_1 \vee x_2 \vee \sim x_3) \wedge (x_3)$
    - Temos então o seguinte conjunto de vértices
      - $N = \{(x_1,1), (\sim x_1,2), (x_2,2), (\sim x_3,2), (x_3,3)\}$
      - Arestas



    - Se B verdadeira para alguma atribuição dos literais, então teremos clique de tamanho maior ou igual a 3

Próximas Aulas...

- Revisão e Prova