DCC207 – Algoritmos 2

Aula 10 – Introdução a Teoria da Complexidade (Parte 04)

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- A discussão de complexidade computacional dos problemas até agora girou em torno do tempo de processamento
- Embora tempo seja um fator importante para determinar a dificuldade de um problema, ele não é o único fator
- Os problemas podem também ser organizados em classes conforme o espaço requerido para a computação
- Elas são úteis para classificar problemas que, em princípio, não podem ser encaixados nas classes de tempo vistas anteriormente
- Muitos problemas de IA (particularmente jogos) se encaixam nessa categoria

Jogo Geography

- Um jogo de criança popular nos EUA, chamado Geography, consiste em dois jogadores escolherem cidades/capitais alternadamente. A cidade escolhida por um jogador deve iniciar com a última letra da cidade escolhida pelo jogador anterior, exceto a primeira jogada que é livre. Perde o jogo, o jogador que não conseguir enumerar uma outra cidade ou repetir alguma dita anteriormente.
- Exemplo: Palmas -> São Paulo -> Olinda -> Araguari -> Ibirité
- O problema de decisão associado é definir se existe uma estratégia vencedora para o jogador 1, iniciando em um vértice arbitrário
  - Como verificar uma solução em tempo polinomial?
  - Como obter uma solução não-deterministicamente em tempo polinomial?
- Não é trivial demonstrar que o problema pertence a P ou NP

Classe de problemas de espaço

- Da mesma forma que fizemos com a análise do tempo, dizemos que um problema A ∈ SPACE(f(n)) se existe uma Máquina de Turing que computa o problema com espaço O(f(n)) (acessa O(f(n)) células durante a computação)
- Especificamente:
  - A ∈ PSPACE se A pode ser computado por uma MT determinística em espaço polinomial
  - A ∈ NPSPACE se A pode ser computador por uma MT não-determinística em espaço polinomial
- Ao contrário da avaliação do tempo, o Teorema de Savitch mostra que toda MT não-determinística pode ser simulada por uma determinística com espaço $O(f^2(n))$
  - Em outras palavras, PSPACE=NPSPACE

PSPACE

- De maneira trivial, sabemos que P $\subseteq$ PSPACE.
  - Se o problema admite solução em tempo polinomial, essa solução deve acessar um número polinomial de células, caso contrário não seria polinomial.
- Mas, e quanto aos problemas NP? O que podemos dizer?
- Teorema: NP $\subseteq$ PSPACE.
- Prova: Vamos demonstrar que 3SAT $\in$ PSPACE. Como 3SAT é NP-completo, segue que todos problemas em NP também pertencem a PSPACE

3SAT ∈ PSPACE

- Considere o problema de contar números em binário entre 0 e 2^n-1.
- Podemos implementar esse algoritmo de forma determinística usando n células da máquina
- A cada iteração, o algoritmo atualiza a contagem somando 1 ao bit menos significativo, e atualizando o valor dos demais.
- É perceptível que, embora o algoritmo rode em tempo $O(2^n)$, ele utiliza apenas $O(n)$ células.
- Mesmo o algoritmo não executando nada de interessante, ele evidencia um princípio importante para diferenciar a análise de tempo e espaço:
  - O espaço pode ser reutilizado para as computações de maneira que o tempo, por definição, não pode.

3SAT ∈ PSPACE

- Voltando ao 3SAT, podemos usar o algoritmo anterior para resolvê-lo de forma ingênua, porém em espaço polinomial
- Se considerarmos que a função booleana contém n variáveis, podemos representar o conjunto de variáveis por um vetor de bits
- Então, usamos o algoritmo anterior para gerar uma atribuição (subconjunto) diferente de variáveis
- Após isso, testamos se essa atribuição faz com que a função avalie a 1. Isso também pode ser feito em espaço polinomial (só precisamos de espaço para armazenar as cláusulas, o que requer espaço O(n)).

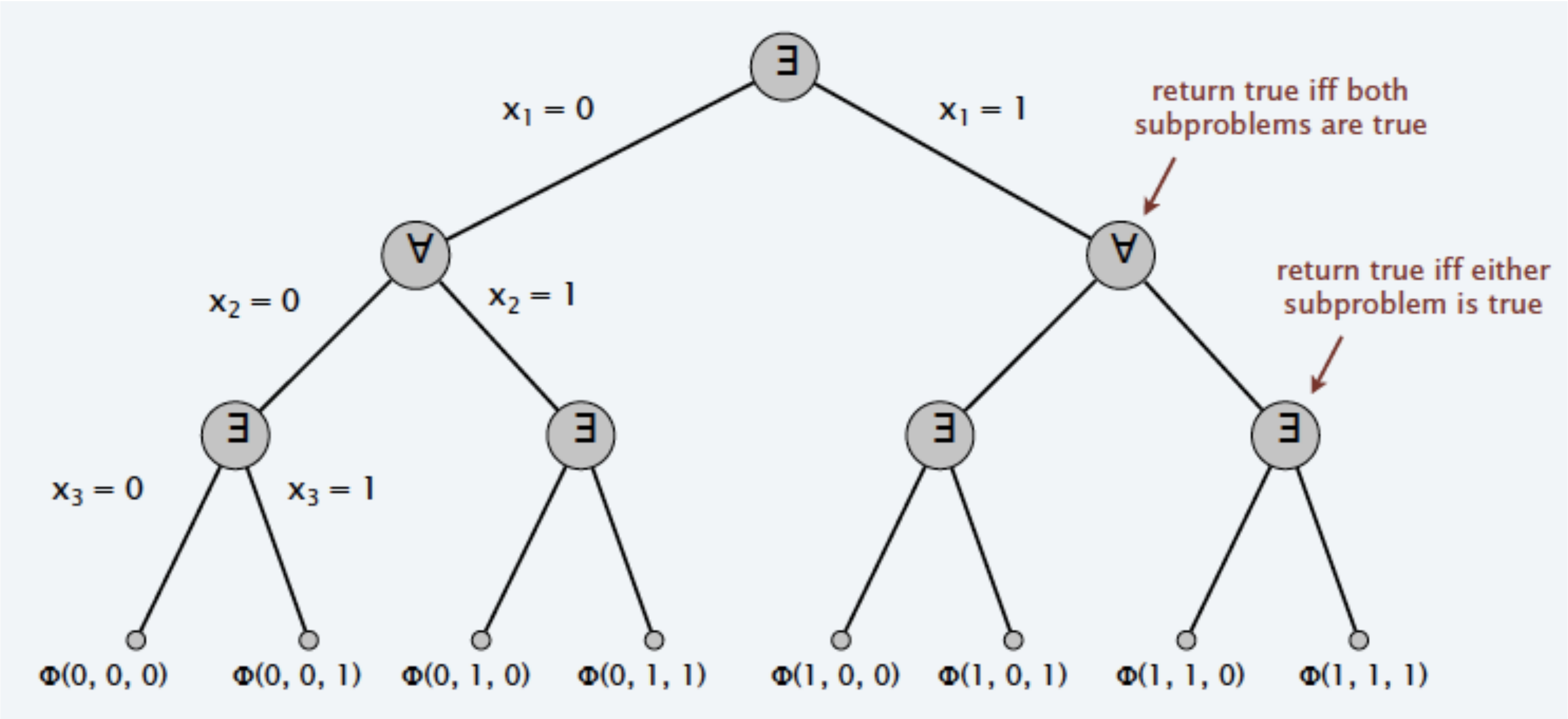
QSAT ∈ PSPACE

- Como já discutimos, o problema da satisfabilidade se mostrou complexo o suficiente para não sabermos se há solução determinística eficiente para ele
- Esse problema aparenta ser ainda mais difícil se incluirmos quantificadores às funções
- Formalmente:
  - Seja $\varphi(x1, x2, ..., xn)$ uma função na CNF
  - A expressão $\exists x1\ \forall x2\ \exists x3 \ldots \forall xn\text{-}1\ \exists xn\ \varphi(x1, x2, ..., xn)$ é satisfazível?
- $\varphi(x1, x2, x3) = (x1 \lor x2 \lor x3) \land (x1 \lor x2 \lor \neg x3) \land (\neg x1 \lor x2 \lor x3) \land (\neg x1 \lor \neg x2 \lor \neg x3)$
- $\exists x1\ \forall x2\ \exists x3\ \varphi(x1, x2, x3)$ é satisfazível?

QSAT ∈ PSPACE

```
If the first quantifier is ∃ x_i then
  Set x_i = 0 and recursively evaluate the quantified expression
                  over the remaining variables
  Save the result (0 or 1) and delete all other intermediate work
  Set x_i = 1 and recursively evaluate the quantified expression
                  over the remaining variables
  If either outcome yielded an evaluation of 1, then
     return 1
  Else return 0
  Endif
If the first quantifier is ∀ x_i then
  Set x_i = 0 and recursively evaluate the quantified expression
                  over the remaining variables
  Save the result (0 or 1) and delete all other intermediate work
  Set x_i = 1 and recursively evaluate the quantified expression
                  over the remaining variables
  If both outcomes yielded an evaluation of 1, then
     return 1
  Else return 0
  Endif
Endif
```

QSAT ∈ PSPACE



https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/09PSPACE.pdf

Planejamento automatizado

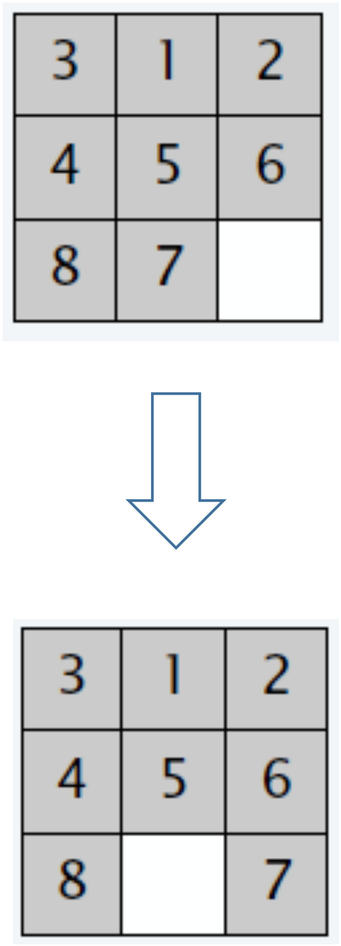
- Planejamento automatizado é uma sub-área da IA que estuda o processo de raciocínio de escolha e organização de ações para atingir metas previamente estabelecidas
  - As escolhas são baseadas em, além das metas, os efeitos de cada ação
- A área tem diversas aplicações como:
  - Robótica (criação de robôs autônomos)
  - Indústria (automatização de processos de manufatura, e logística)
  - Jogos

Planejamento automatizado

- Formalmente:
  - C = {C1, C2, ..., Cn} é um conjunto de condições/situações do problema
  - O = {O1, O2, ..., Ok} é um conjunto de ações/operações
  - Cada ação Oi possui uma lista Pi de pré-requisitos (condições) para ser executada
  - A execução da ação Oi faz com que uma lista Ai de condições se tornem verdadeiras, e uma lista Di de condições se tornem falsas
  - Uma configuração é $C' \subseteq C$ que são verdadeiras em um dado instante
- O problema então consiste em determinar se existe uma sequência de ações que nos leve de uma configuração inicial C0 a uma final C*

Quebra-cabeças deslizante

- O quebra-cabeças de 8 peças é um exemplo da aplicação de planejamento automatizado
- Condições: Cij peça i está no quadrado j
- C = {Cij | 1 <= i,j <= 9}
- C0 = {C12, C23, C31, C44, C55, C66, C78, C87, C99}
- C* = {C11,C22, ..., C99}
- Oi = movimentações legais de peças
  - Oi -> Pi = {C12, C23, C31, C44, C55, C66, C78, C87, C99}
  - Oi -> Ai = {C79, C98}
  - Oi -> Di = {C78,C99}



Contador binário

- O contador binário é outro exemplo de problema de planejamento
- Ci = i-ésimo bit é 1
- C = {C1, C2, ..., Cn} os n bits necessários para contar de 0 a 2^n-1
- C0 = ∅; C* = C
- Oi = i-ésimo bit é marcado com 1
  - Pi = {C1, ..., Ci-1}; Ai = {Ci}; Di = Pi
- Solução: ∅ -> {C1} -> {C2} -> {C1,C2} -> {C3} ...
- Qualquer solução requer 2^n-1 passos

Planejamento automatizado ∈ EXPTIME

- O problema do planejamento automatizado pode ser modelado por um grafo
  - Vértices são configurações -> $|V| = 2^n$
  - Arestas são transições entre configurações através da aplicação de uma ação
  - Solução: caminho de C0 a C*
- Menor caminho pode ter tamanho 2^n-1 (contador binário)
  - A simples verificação de uma solução não pode ser feita em tempo polinomial
  - Planejamento automatizado ∈ EXPTIME
  - Solução: BFS no grafo de configurações

Planejamento automatizado ∈ PSPACE

- Construir o grafo explicitamente ou aplicar BFS/DFS para construí-lo durante a execução pode requerer espaço exponencial
- Na demonstração de seu teorema, Savitch utilizou uma modificação de dividir-e-conquistar para solucionar o problema. Essa abordagem pode ser utilizada aqui também.
- O problema inicial é equivalente a resolver Path(C0,C*,2^n)
  - Determinar se há caminho de C0 a C* com no máximo 2^n passos
- BFS = Path(C0,C',2^n-1) ∧ Path(C',C*,1)
- Alternativamente, podemos fazer Path(C0,C',2^{n-1}) ∧ Path(C',C*,2^{n-1}); isto é escolher um 'ponto-médio' do caminho e verificar se há como chegar até ele de C0, e dele a C*.
- Não é necessário armazenar todas as configurações intermediárias, apenas se há tal caminho
- Custo O(f(n)log 2^n) = O(n f(n)) -> f(n) custo (polinomial) armazenamento configurações (condições, ações e suas listas)

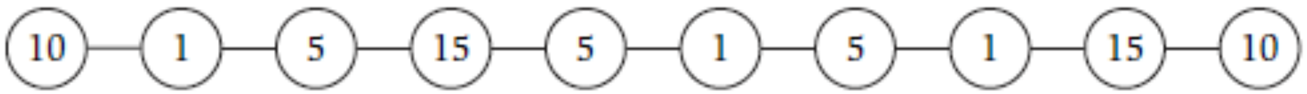
```
boolean hasPath(c1, c2, L) {
   if (L ≤ 1) return correct answer
                 enumerate using binary counter
   foreach configuration c′ {
      boolean x = hasPath(c1, c′, L/2)
      boolean y = hasPath(c2, c′, L/2)
      if (x and y) return true
   }
   return false
}
```

Problemas PSPACE-Completos

- Um problema B é PSPACE-completo se:
  - $B \in PSPACE$
  - $\forall A \in PSPACE, A \leq_p B$ (PSPACE-difícil)
- QSAT foi demonstrado ser PSPACE-completo (Stockmeyer e Meyer 1973)
- Portanto, $PSPACE \subseteq EXPTIME$
  - Algoritmo anterior computa a resposta em tempo $O(2^n)$
- $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$, $P \neq EXPTIME$
  - Intuição é que os conjuntos sejam todos distintos
- Podemos demonstrar que outros problemas são PSPACE-completos, reduzindo QSAT a eles

Problemas PSPACE-Completos

- Competitive Facility Location Problem: considere um jogo em que dois jogadores disputam territórios para instalação de empresas. Cada território só pode ser dominado por um jogador. Territórios adjacentes a um dominado não podem ser dominados. O lucro obtenível de cada território é fixo e pré-determinado. Os jogadores jogam em turnos, iniciando pelo jogador 1. Ganha o jogador que obtiver o maior lucro.
- O jogo pode ser modelado como um grafo.
  - Vértices: territórios (com pesos, representando lucro)
  - Arestas: adjacência entre territórios
  - Objetivo: Conjunto independente de maior valor agregado
- Um problema de decisão associado: existe uma estratégia para o jogador 2 garantir lucro de, no mínimo, B unidades

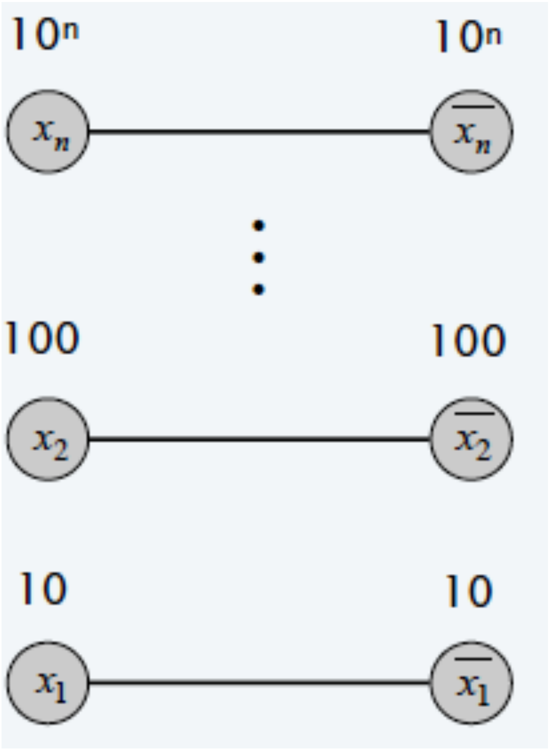


B=20 -> sim

B=25 -> não

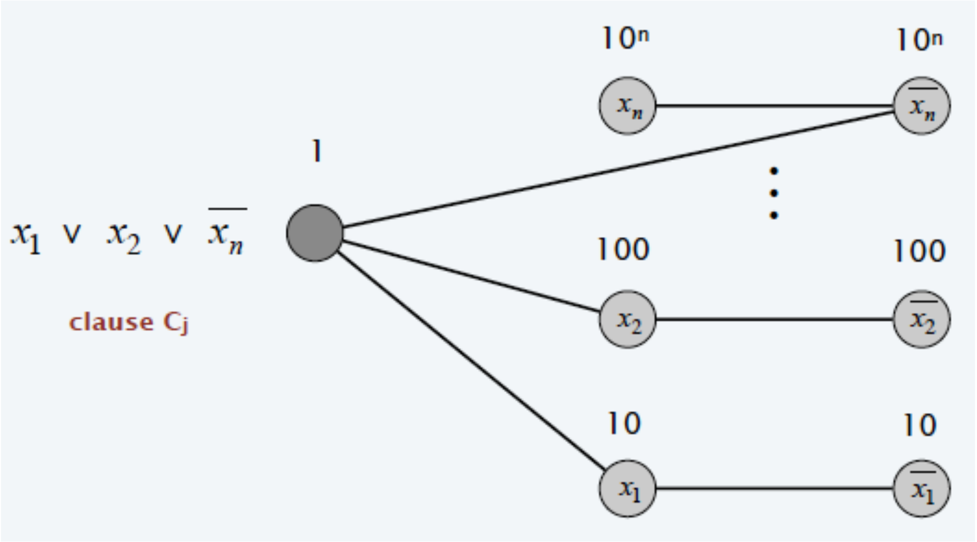
Problemas PSPACE-completos

- Podemos solucionar o problema com espaço polinomial, usando uma estratégia similar ao algoritmo para QSAT
  - Temos n escolhas para cada passo recursivo
- Agora, reduziremos QSAT ao problema. Para isso, considere φ(x1, x2, ..., xn) = C1 ∧ C2 ∧ ... ∧ Ck uma instância de QSAT com n (ímpar de) variáveis e k cláusulas
- Para cada variável xi, criamos dois vértices vi e vi', representando respectivamente xi e ¬xi. Adicionamos também uma aresta ligando os dois
  - Como o problema restringe a solução a um conjunto independente, xi e ¬xi não serão ambos escolhidos
- Atribuímos o peso c^i a cada vértice vi e vi'; c ≥ k+2
- Fazemos B = $c^{n-1} + c^{n-3} + ... + c^2 + 1$
- Com essa configuração, o jogador 2 atinge somente B-1 na melhor estratégia (escolhendo vi ou vi', tal que i é par)



Problemas PSPACE-completos

- Como o número de variáveis é ímpar, a última jogada sempre é do jogador 1
- Então, para permitir que o jogador 2 possa vencer, vamos adicionar um novo vértice para cada cláusula com peso 1.
- Esse novo vértice será adjacente a todos os literais que compõem a cláusula
- Dessa forma, a única chance do jogador 2 vencer é escolhendo pelo menos um desses vértices
  - Isso só pode ocorrer se nenhum literal da cláusula tiver sido escolhido
- Portanto, a função é satisfazível sse o jogador 2 não possuir uma estratégia que sempre garanta sua vitória



Leitura

- Capítulo 9 (Kleinberg e Tardos)