DCC207 – Algoritmos 2

Aula 07 – Introdução a Teoria da Complexidade (Parte 01)

Professor Renato Vimieiro

DCC/ICEx/UFMG

Introdução

- Teoria da complexidade é uma sub-área de Teoria da Computação que se dedica a estudar e classificar a complexidade de problemas computacionais
- Inicialmente, havia um interesse maior em se definir o que poderia ou não ser computado
- Isso requeria a formalização do conceito de algoritmo, através de um modelo de computação
- Tal formalização foi conquistada na década de 1930 com a proposição da máquina de Turing e o teorema de Church-Turing
- Contudo, dentre os problemas que se mostravam computáveis no modelo de Turing, há ainda uma divisão daqueles que são tratáveis (há solução eficiente) dos que são intratáveis
- Em outras palavras, os problemas computáveis podem ser classificados conforme sua complexidade. Esse é o tema dessa e das próximas aulas.

Problemas de decisão

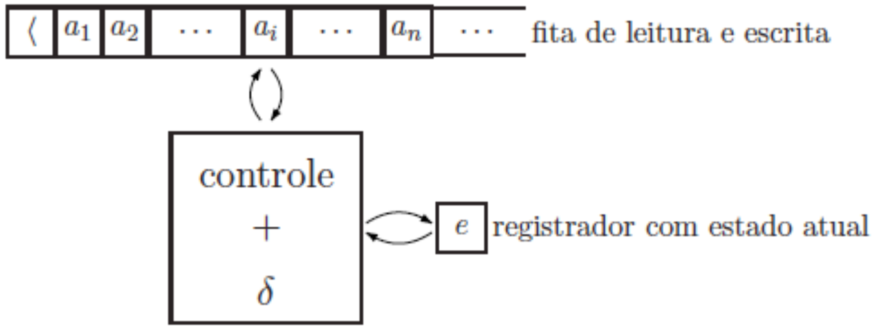
- Os problemas computáveis apresentam diversas naturezas, como:
  - Problemas de decisão
  - Problemas de busca
  - Problemas de otimização
- Para simplificar a discussão, restringiremos a atenção a problemas de decisão
- Problemas de decisão são problemas envolvendo n parâmetros cuja resposta é sim ou não.
  - Determinar se n é primo
  - Determinar se 11 é primo
  - Determinar se um grafo G possui um ciclo
  - Determinar se x+y=z

Problemas de decisão

- Uma instância de um problema de decisão é obtida quando se atribui valores específicos aos parâmetros do problema:
  - Determinar se 439409123 é primo
- Um problema de decisão é solúvel se existe um algoritmo que, para qualquer instância do problema, responde sim ou não
- Problemas de decisão podem ser postos como linguagens formais. Dessa forma, a solução de um problema de decisão é um algoritmo que reconhece a linguagem equivalente
- Exemplo:
  - Usando um alfabeto composto somente por 0s, podemos representar os números naturais de forma unária; <1> = 0, <5>=00000, <n>=0^n
  - Sendo assim, a linguagem L = {0^n | n é primo} representa o problema de decisão de determinar se um número n é primo
  - Determinar se n é primo é o mesmo que determinar se <n> ∈ L

Máquina de Turing

- Como dito anteriormente, a formalização do conceito de algoritmo foi feito através da máquina de Turing
- De maneira informal, a máquina de Turing se assemelha a um autômato
- Sua arquitetura está ilustrada na figura abaixo. Ela possui uma fita de leitura e escrita; um registrador de estado; e uma função de transição e controle que atualiza o conteúdo da fita e o registrador de estado



Máquina de Turing

- Uma palavra escrita na fita de entrada da máquina de Turing é aceita por ela se, ao processar seus símbolos, ela entra em um estado de aceitação
- Como a MT formaliza o conceito de algoritmo, as noções de complexidade de tempo e espaço se traduzem diretamente para as operações da máquina:
  - Tempo = número de transições que a máquina realiza para aceitar/rejeitar a palavra
  - Espaço = número de células da fita que ela utiliza para realizar a computação

Máquina de Turing

- Considere o problema de se determinar se uma palavra $w \in \{0^k 1^k \mid k \geq 0\}$
- Ele pode ser resolvido por uma MT da seguinte forma:
  - Varra a fita até o final, procurando por algum 0 após um 1. Se ocorrer rejeite.
  - Repita até que todos os 0s ou 1s tenham sido consumidos:
    - Busque o 0 mais à esquerda na fita e apague-o
    - Busque o 1 mais à direita e apague-o
  - Verifique se a fita está vazia. Se estiver, aceite. Senão, rejeite.
- Como podemos perceber, são necessárias $O(n^2)$ transições para resolver o problema ($|w| = n$)
- Por outro lado, são usadas somente $O(n)$ células.

Máquina de Turing

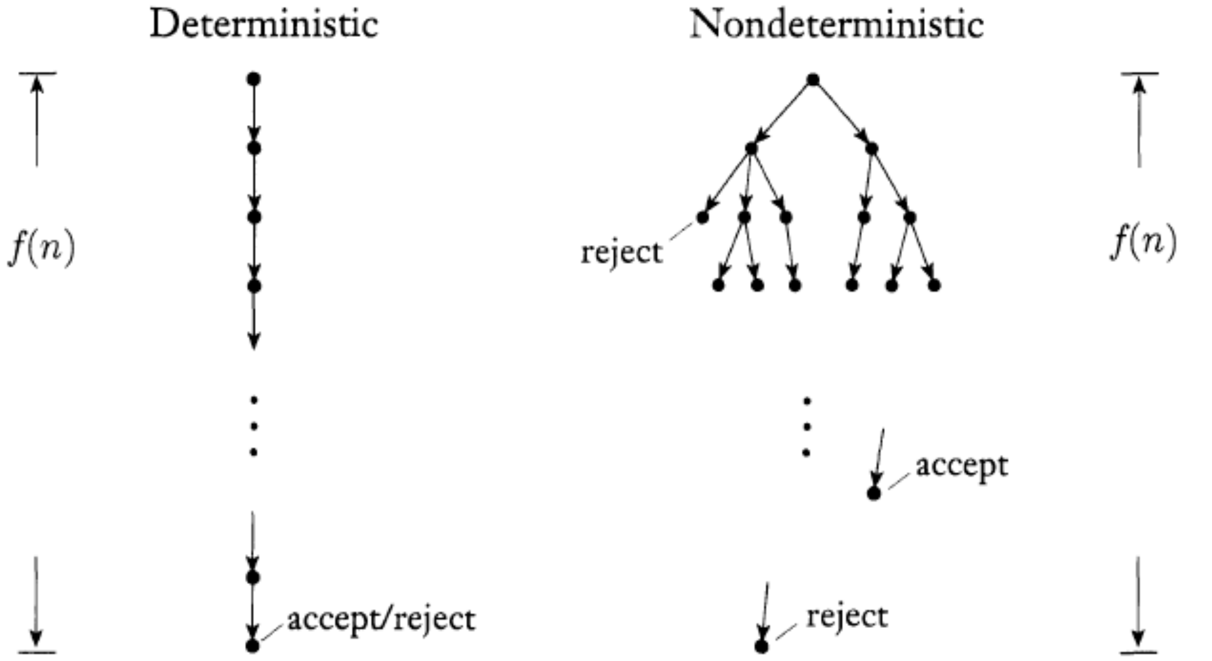
- O modelo de Turing admite, pelo menos, duas versões:
  - Máquina de Turing Determinística
  - Máquina de Turing Não-Determinística
- Na versão determinística, para cada par estado-símbolo, existe uma única transição realizável
- Na versão não-determinística, para cada par estado-símbolo, existe n transições possíveis
  - A MT não-determinística pode ser vista como contendo um oráculo que define qual a transição a ser escolhida para que a computação termine em aceitação
  - Alternativamente, ela pode ser vista como uma versão ‘massivamente paralela’ da determinística que realiza todas as computações possíveis em paralelo sem custo adicional

Máquina de Turing

- Dizemos que uma linguagem pertence à classe TIME(t(n)) se existe uma máquina de Turing determinística que a decida em tempo O(t(n))
- Dessa forma, TIME(t(n)) é a classe das linguagens decidíveis (problemas solúveis) em tempo O(t(n))
  - A linguagem $\{0^k 1^k \mid k \geq 0\} \in TIME(n^2)$
- Analogamente, uma linguagem pertence à classe NTIME(t(n)) se existe uma máquina de Turing não-determinística que a decida em tempo O(t(n))
  - A linguagem $\{\langle G,k \rangle \mid \text{O grafo G tem uma clique de tamanho k}\} \in NTIME(n^2)$

Relação de complexidade entre tipos de MTs

- Teorema: Toda MT **não-determinística** que decide uma linguagem em tempo t(n) possui uma MT **determinística** equivalente com tempo $2^{O(t(n))}$



Classe P

- A relação de complexidade entre os dois tipos de MTs induz uma separação nos problemas
  - Aqueles que podem ser resolvidos eficientemente por uma MT **determinística**
  - Aqueles que podem ser resolvidos eficientemente por uma MT **não-determinística**
- O teorema anterior demonstra que nem toda solução não-determinística possui uma determinística equivalente eficiente
- A classe P é a classe das linguagens (problemas) que possuem soluções determinísticas eficientes
  - $P = \bigcup TIME(n^k)$
- De uma forma geral, a classe P define a classe dos problemas computacionalmente tratáveis

Exemplos de problemas da classe P

- PATH = {<G,s,t> | existe caminho de s para t no grafo direcionado G}
  - Busca em profundidade começando em s
- RELPRIME = {<x,y> | x e y são primos relativos}
  - Algoritmo de Euclides para MDC
- PRIME = {<x> | x é primo}
  - Algoritmo de Agrawal, Kayal e Saxena
  - Proposto em 2002; até então não existia prova de que PRIME ∈ P

Classe NP

- A classe NP é a classe das linguagens (problemas) que possuem soluções não-determinísticas eficientes
  - NP = $\bigcup$ NTIME(n^k)
- Uma definição alternativa é baseada na possibilidade de se verificar uma solução em tempo polinomial
- Como MTs determinísticas podem ser vistas como casos particulares de MTs não-determinísticas, temos que
  - P $\subseteq$ NP
- Contudo, não sabemos nada a respeito da inversa. Podem existir soluções determinísticas eficientes para certos problemas, as quais ainda são desconhecidas
- Sabemos que P $\subseteq$ NP $\subseteq$ EXPTIME = $\bigcup$ TIME(2^{n^k}) já que toda MT não-determinística pode ser simulada por uma MT determinística em tempo $2^{O(t(n))}$
- A classe NP não define a classe de problemas intratáveis, a menos que P ≠ NP

Exemplos de problemas da classe NP

- CLIQUE = {<G,k> | O grafo G possui uma clique de tamanho k}
  - Uma clique em um grafo é um subgrafo induzido completo
- Uma solução não-determinística para CLIQUE seria:
  - Não-deterministicamente escolha um conjunto de k vértices de G
  - Teste se eles formam uma clique
  - Se sim, aceite, senão rejeite
- A escolha dos vértices é feita em tempo O(n)
- O teste de clique é feito em tempo $O(n^2)$
- O custo total é $O(n^2)$
- Como a escolha é não-determinística:
  - Um conjunto que forma uma k-clique é definido pelo oráculo, ou
  - Todos os subconjuntos de vértices são testados em paralelo sem custo adicional

Exemplos de problemas da classe NP

- $SUBSET\_SUM = \{\langle S,w \rangle \mid S \subseteq \mathbb{Z} \wedge \exists T \subseteq S\ sum(T) = w\}$
- Uma solução não-determinística seria:
  - Escolha, não-deterministicamente, um subconjunto T de elementos
  - Verifique se a soma é igual a w
  - Se a soma for w, aceite, senão rejeite
- Nesse caso, tanto a escolha quanto o teste possuem tempo O(n)
- Novamente a escolha não-determinística permite avaliar ‘todas’ as possibilidades em tempo polinomial
- Note, contudo, que a implementação de não-determinismo não é exatamente factível nos computadores atuais
  - Isso nos restringe a soluções determinísticas exponenciais num primeiro momento; isto é, simular deterministicamente as computações realizadas pela MT não-determinística (algoritmo força bruta)

Classe co-NP

- Considere o problema de determinar se um grafo não possui uma clique de tamanho k
  - co-CLIQUE = {<G,k> | O grafo G não possui uma clique de tamanho k}
- Nesse caso, a simples constatação de que G não possui uma clique de tamanho k envolve a computação de **todos** os subconjuntos de tamanho k
  - Não é claro, portanto, que co-CLIQUE ∈ NP
- A mesma análise pode ser feita com o SUBSET_SUM
- Note, no entanto, que verificar se <G,k> ∉ co-CLIQUE é relativamente simples; basta encontrar uma CLIQUE de tamanho k
- Os problemas complementares aos NP formam uma classe em si
  - co-NP = {L^c | L ∈ NP}
- Veja que P ⊆ NP ⋂ co-NP
  - Por que?
- Não se sabe se NP ≠ co-NP. Tal demonstração levaria à conclusão de que P ≠ NP.
- Isso mostra que a percepção de que os problemas em co-NP são mais difíceis não é exatamente verdadeira
- Eles também podem ser resolvidos não-deterministicamente em tempo polinomial

Leitura

- Seções 7.1, 7.2, 7.3 (Introduction to the Theory of Computation, 2nd ed., Michael Sipser)