PUC Minas

Lista 5 – Teoria da Complexidade

INFORMAÇÕES DOCENTE						
CURSO: ENGENHARIA DE SOFTWARE	DISCIPLINA: FUNDAMENTOS DE PROJETO E ANÁLISE DE ALGORITMOS	TURNO	MANHÃ	TARDE	NOITE	PERÍODO/SALA:
					x	5º
PROFESSOR (A): João Paulo Carneiro Aramuni						

Lista 5 - Gabarito

Teoria da Complexidade - Algoritmos Polinomiais e Exponenciais

1) Qual das opções representa um tempo de execução polinomial?

a) $O(2^n)$

b) $O(n!)$

c) $O(n^3)$

d) $O(n^n)$

e) $O(n * 2^n)$

Resposta: c) $O(n^3)$

Explicação: Um tempo de execução polinomial é aquele cuja complexidade pode ser expressa como $O(n^k)$, onde k é uma constante. O $O(n^3)$ se encaixa nessa definição. As demais são exponenciais ou superexponenciais.

Definição formal: Um algoritmo tem tempo polinomial se seu tempo de execução T(n) pode ser limitado por uma função polinomial, ou seja, existe k tal que $T(n) = O(n^k)$.

Por que $O(n^3)$: Esse termo significa que, quando você dobra o tamanho da entrada, o tempo de execução aumenta em um fator de até $2^3 = 8$. Em geral, polinômios crescem de modo "suave" em comparação a exponenciais.

Contraste com as demais:

- $O(2^n)$ e $O(n * 2^n)$ são exponenciais: cada acréscimo na entrada duplica ou mais que duplica o tempo.
- $O(n!)$ e $O(n^n)$ crescem ainda mais rápido do que exponenciais.

Importância prática: Algoritmos polinomiais geralmente são considerados "eficientes" ou "aproximadamente viáveis" para entradas de tamanho realista (e.g., centenas ou milhares), enquanto exponenciais tornam-se impraticáveis muito antes disso.

2) Um algoritmo exponencial geralmente:
a) Executa em tempo constante
b) Cresce mais lentamente que um algoritmo linear
c) É sempre mais eficiente que um algoritmo polinomial
d) Cresce mais rapidamente que qualquer função polinomial
e) É classificado como pertencente à classe P

Resposta: d) Cresce mais rapidamente que qualquer função polinomial

Explicação: Algoritmos exponenciais possuem crescimento muito rápido, como $O(2^n)$ ou $O(n!)$, o que os torna ineficientes para entradas grandes, diferentemente dos algoritmos polinomiais.

Crescimento exponencial: Funções como 2^n, 3^n ou $n!$ dobram ou multiplicam o tempo de execução por um fator constante sempre que n aumenta em 1.

Comparação com polinômios: Para um polinômio n^k, cada acréscimo em n aumenta o valor de forma aditiva/combinatória, não multiplicativa em grande escala.

Impacto prático: Se um algoritmo de força bruta que testa todas as combinações tiver complexidade $O(2^n)$, ele poderá rodar rápido para $n = 20$ (≈1 milhão de casos), mas para $n = 50$ já são mais de um quatrilhão de casos – impossível na prática.

3) Qual das opções caracteriza corretamente um algoritmo polinomial?
a) Tempo de execução cresce com 2^n
b) Usa força bruta sempre
c) Tempo de execução é $O(n^k)$, com k constante
d) Não pode ser implementado em tempo real
e) Só é usado para problemas NP

Resposta: c) Tempo de execução é $O(n^k)$, com k constante

Explicação: A definição de algoritmo polinomial é exatamente essa: um algoritmo com complexidade $O(n^k)$, onde k é constante. É o critério para estar na classe P.

Condição necessária e suficiente: A forma $O(n^k)$ define com precisão a classe dos algoritmos "polinomiais".

Exemplos comuns:
- Busca simples em vetor: $O(n)$ (k=1).

- Ordenação por mergesort: $O(n \log n)$ (considerado quase-polinomial, mas geralmente incluído como eficiente).
- Multiplicação de matrizes: $O(n^3)$ no algoritmo ingênuo.

Importância teórica: A classe P, central na teoria da complexidade, é o conjunto de todos os problemas decidíveis em tempo polinomial.

4) Um algoritmo de complexidade $O(2^n)$ torna-se impraticável quando:
a) n é menor que 5
b) n é negativo
c) n aumenta muito, pois o tempo cresce rapidamente
d) O problema é determinístico
e) o número de entradas é constante

Resposta: c) n aumenta muito, pois o tempo cresce rapidamente

Explicação: À medida que n aumenta, 2^n cresce muito rapidamente, tornando algoritmos exponenciais inviáveis na prática, mesmo para valores de entrada relativamente pequenos (como 50 ou 100).

Crescimento em potências de 2: Cada incremento de n dobra o número de operações.

Limite prático: Mesmo com processadores modernos, um algoritmo $O(2^n)$ com $n = 60$ exigiria mais tempo do que a idade do universo para completar.

Uso real: São usados apenas para inputs muito pequenos ou como garantia teórica, mas em aplicações práticas busca-se heurísticas ou aproximações.

5) Um exemplo de problema que pode ser resolvido com um algoritmo polinomial é:
a) Subconjunto somatório
b) Multiplicação de matrizes
c) Caixeiro viajante
d) Satisfatibilidade booleana (SAT)
e) Cobertura de vértices mínima

Resposta: b) Multiplicação de matrizes

Explicação: A multiplicação de matrizes pode ser resolvida por algoritmos com complexidade $O(n^3)$ ou melhores (como Strassen), o que é polinomial. Os demais são problemas tipicamente NP-difíceis ou NP-completos.

PUC Minas

Multiplicação de matrizes:
- Algoritmo clássico: $O(n^3)$.
- Métodos avançados (Strassen, Coppersmith–Winograd): até $O(n^{2.37})$ ou similares.

Problemas NP-completos (C, D, E):
- Caixeiro viajante, SAT e Cobertura de vértices não têm algoritmos de tempo polinomial conhecidos.
São resolvidos em tempo exponencial no pior caso ou via aproximações/polynomial-time heuristics.

Teoria da Complexidade - Algoritmos Determinísticos e Não Determinísticos

6) Um algoritmo determinístico:
a) Pode retornar resultados diferentes em execuções iguais
b) Usa aleatoriedade para decidir caminhos
c) Sempre segue um mesmo caminho de execução para a mesma entrada
d) É mais lento que um não determinístico
e) Não pode ser implementado em máquinas reais

Resposta: c) Sempre segue um mesmo caminho de execução para a mesma entrada

Explicação: Algoritmos determinísticos têm comportamento previsível: sempre que executados com a mesma entrada, produzem a mesma saída, seguindo o mesmo fluxo de execução.

Comportamento pré-determinístico: Dada uma configuração inicial (código + entrada + estado da máquina), um algoritmo determinístico sempre faz a mesma sequência de operações.

Exemplo prático: Um simples laço for(i=0; i<n; i++) que soma elementos de uma lista terá exatamente o mesmo fluxo de instruções e resultados toda vez.

Modelagem formal: Máquina de Turing determinística, onde cada estado e símbolo leva a uma única transição.

7) O que caracteriza um algoritmo não determinístico?
a) Ele é baseado em decisões lógicas estritas
b) Executa sempre na mesma ordem
c) Pode "escolher" soluções corretas sem explorar todas
d) É mais fácil de implementar
e) Tem menos complexidade que o determinístico

Resposta: c) Pode "escolher" soluções corretas sem explorar todas

Explicação: No modelo teórico, algoritmos não determinísticos "adivinham" uma solução correta entre muitas possibilidades e a verificam rapidamente. É uma abstração usada para definir a classe NP.

Conceito teórico: Em cada passo, a máquina não determinística pode “ramificar” para vários caminhos de execução simultaneamente. Se qualquer ramificação encontra solução, o problema é resolvido.

Analogia: É como tentar todas as senhas possíveis ao mesmo tempo, e se uma estiver certa, você “sabe” instantaneamente.

Uso em complexidade: Define a classe NP: problemas que uma máquina não determinística resolve em tempo polinomial.

8) O modelo de máquina teórica usado para algoritmos não determinísticos é:
a) Máquina de Turing determinística
b) Autômato finito
c) Máquina de estados
d) Máquina de Turing não determinística
e) Pilha de execução

Resposta: d) Máquina de Turing não determinística

Explicação: A máquina de Turing não determinística é o modelo teórico que permite múltiplas transições possíveis para o mesmo estado/símbolo, representando a execução de um algoritmo não determinístico.

Máquina de Turing não determinística (MTND): Igual à MT normal, mas sua função de transição pode retornar um conjunto de possibilidades.

Formalização: Se existe algum caminho de configuração que leva ao estado de aceitação em $\leq p(n)$ passos, o MTND aceita a entrada.

Importância: É um modelo-chave para a definição de NP e para analisar problemas NP-completos.

9) Qual das alternativas é verdadeira sobre algoritmos não determinísticos?
a) São impossíveis de simular em computadores reais
b) Não podem resolver problemas NP
c) Só funcionam para problemas polinomiais
d) Podem ser simulados por algoritmos determinísticos com tempo exponencial
e) Sempre retornam a melhor solução

Resposta: d) Podem ser simulados por algoritmos determinísticos com tempo exponencial

Explicação: Embora não existam máquinas não determinísticas reais, podemos simular seus comportamentos em máquinas determinísticas, geralmente com custo exponencial de tempo.

Simulação determinística: Cada ramificação do não determinismo pode ser seguida sequencialmente; como há exponencialmente muitas ramificações, a simulação leva tempo exponencial.
Exemplo: Um MTND com ramificação binária de profundidade n se torna 2^n caminhos em MT determinística.

Resultado: Embora teórico, permite comparar classes de complexidade.

10) Qual é uma consequência do não determinismo na computação?
a) Algoritmos sempre falham
b) Problemas fáceis tornam-se impossíveis
c) Possibilidade teórica de resolver problemas difíceis em tempo polinomial
d) O tempo de execução diminui sempre
e) Todos os algoritmos tornam-se imprevisíveis

Resposta: c) Possibilidade teórica de resolver problemas difíceis em tempo polinomial

Explicação: Se tivéssemos acesso a máquinas não determinísticas, poderíamos resolver certos problemas (como SAT ou caixeiro viajante) em tempo polinomial — por isso a questão de P vs NP é tão importante.

Visão ideal: Se pudéssemos “tentar tudo de uma vez”, problemas como SAT ou TSP teriam soluções em tempo polinomial.

P vs NP: O grande mistério é se há um algoritmo determinístico que simula esse não determinismo em tempo polinomial — i.e., se P = NP.

Implicações práticas: Caso P=NP, criptografia baseada em dificuldade computacional cairia por terra.

Teoria da Complexidade - Classes P, NP, NP-Completo e NP-Difícil

11) A classe P contém:
a) Problemas sem solução conhecida
b) Problemas resolvíveis em tempo polinomial
c) Problemas que só podem ser verificados, não resolvidos
d) Problemas de tempo exponencial
e) Apenas problemas não determinísticos

Resposta: b) Problemas resolvíveis em tempo polinomial

Explicação: P é a classe dos problemas que podem ser resolvidos (isto é, decididos) por um algoritmo determinístico em tempo polinomial.

Definição formal: P = $\bigcup_{k \in N} DTIME(n^k)$.

Significado prático: Problemas "eficientemente solucionáveis".
Exemplos típicos: Ordenação, busca em grafos, cálculo de caminhos mínimos (Dijkstra), fluxos em rede (Ford–Fulkerson).

12) A classe NP contém:
a) Apenas problemas sem solução exata
b) Problemas que podem ser resolvidos com algoritmos determinísticos rápidos
c) Problemas cujas soluções podem ser verificadas em tempo polinomial
d) Problemas que não podem ser verificados
e) Problemas que estão fora de qualquer classe conhecida

Resposta: c) Problemas cujas soluções podem ser verificadas em tempo polinomial

Explicação: NP (nondeterministic polynomial time) é a classe de problemas para os quais, dada uma solução, conseguimos verificar sua validade em tempo polinomial.

Definição formal: NP = problemas decididos por uma MTND em tempo polinomial.

Equivalência verificação: Uma máquina determinística verifica uma "prova" ou certificado em tempo polinomial.

Exemplos: SAT, clique em grafos, cobertura de vértices.

13) Um problema é NP-completo se:
a) Está em P
b) É mais fácil que qualquer problema em NP
c) Está em NP e todo problema de NP pode ser reduzido a ele
d) Não pode ser resolvido por nenhuma máquina
e) Está fora da classe NP

Resposta: c) Está em NP e todo problema de NP pode ser reduzido a ele

Explicação: Um problema NP-completo é tão difícil quanto qualquer outro problema em NP. Se um deles for resolvido em tempo polinomial, todos os problemas em NP também podem ser.

Redução polinomial: Transformação de instâncias de qualquer problema em NP para instâncias deste problema, em tempo polinomial.

Dificuldade "mais alta" de NP: É o "topo" da classe NP — o mais difícil dentro de NP.

Exemplo clássico: SAT foi o primeiro problema provado NP-completo (Teorema de Cook–Levin).

14) Qual das afirmações é verdadeira?
a) Todo problema NP-completo é NP-difícil
b) Todo problema em P é NP-completo
c) Nenhum problema em NP pode ser resolvido
d) Todo problema NP-difícil está em NP
e) P = NP já foi provado verdadeiro

Resposta: a) Todo problema NP-completo é NP-difícil

Explicação: A classe NP-difícil inclui todos os problemas que são ao menos tão difíceis quanto os problemas em NP. Os problemas NP-completos são uma interseção entre NP e NP-difícil.

NP-difícil: Conjunto de problemas pelo menos tão difíceis quanto qualquer em NP (toda NP reduzível a eles), mas não precisam estar em NP.

Portanto: Se um problema é NP-completo, ele está em NP e, por definição, é NP-difícil.

Contraexemplo da B e D: Há problemas NP-difíceis fora de NP (como o problema da parada geral), e todo problema em P não é necessariamente NP-completo (pois teria que ser o mais difícil da NP).

15) Um problema NP-difícil é caracterizado por:
a) Ser impossível de verificar
b) Estar necessariamente em NP
c) Ser no máximo tão difícil quanto qualquer problema de NP
d) Ser tão difícil quanto qualquer problema em NP, mas pode não estar em NP
e) Ter solução em tempo polinomial

Resposta: d) Ser tão difícil quanto qualquer problema em NP, mas pode não estar em NP

Explicação: Problemas NP-difíceis podem ou não estar em NP. O importante é que qualquer problema em NP pode ser reduzido a eles, o que os torna ao menos tão difíceis quanto qualquer problema em NP.

Definição formal: Um problema X é NP-difícil se, para todo problema Y em NP, existe uma redução polinomial $Y \leq p\ X$.

Distinção de NP-completo: NP-completo = NP ∩ NP-difícil; NP-difícil pode incluir problemas ainda mais gerais, possivelmente não decidíveis em tempo polinomial nem verificáveis nesse tempo.

Exemplo fora de NP: Problemas de otimização mais gerais ou problemas indecidíveis com restrições que os tornam tão difíceis quanto NP, mas sem verificar soluções em polinomial.
