Universidade Federal de Minas Gerais
Programa de Pós-Graduação em Ciência da Computação
Exame de Qualificação 1º Estágio
1º Semestre de 2019

Em 22/03/2019, **09:00 horas**.

Prova individual sem consulta com duração de **3 horas**.

Observações:

1. A prova deve ser resolvida no próprio caderno de questões.
2. As questões desta prova estão nas páginas seguintes, as quais estão numeradas de 1 a 6.
3. Faz parte da prova a interpretação das questões. Caso você ache que falta algum detalhe nos enunciados ou nos esclarecimentos, você deverá fazer as suposições que achar necessárias e escrever essas suposições juntamente com as respostas.
4. **Todas** as respostas devem ser justificadas.
5. Somente serão corrigidas respostas legíveis.
6. Não se esqueça de escrever seu **nome completo em todas as páginas**.

Desejamos a você uma boa prova!

A COPEQ

Atenção: Esta prova contém um total de 6 (seis) questões, das quais você deve fazer 4 (quatro). Marque abaixo as questões que devem ser consideradas para avaliação:

1 2 3 4 5 6 (selecione até quatro)

Nome completo:_____

Assinatura: _____

Nome completo:

Questão 1

Preencha a tabela abaixo com V (verdadeiro) ou F (falso) assumindo que c e k são constantes positivas. Para cada linha da tabela forneça uma justificativa baseada na definição de O, Ω e Θ.

	$f(n) = O(g(n))$	$f(n) = \Omega(g(n))$	$f = \Theta(g(n))$
$f(n) = n^k, g(n) = c^n$			
$f(n) = n^3 \log_8(n), g(n) = 3n \log_2(n)$			
$f(n) = 8^n, g(n) = 4^n$			
$f(n) = \log_2(n^{log_2(19)}), g(n) = log_2(13^{log_4(n)})$			

Solução:

	$f(n) = O(g(n))$	$f(n) = \Omega(g(n))$	$f = \Theta(g(n))$
$f(n) = n^k, g(n) = c^n$	V	F	F
$f(n) = n^3 \log_8(n), g(n) = 3n \log_2(n)$	F	V	F
$f(n) = 8^n, g(n) = 4^n$	F	V	F
$f(n) = \log_2(n^{log_2(19)}), g(n) = log_2(13^{log_4(n)})$	V	V	V

Assumindo k e c maiores do que 1. A primeira função é polinomial em n e a segunda é exponencial.

Os logs estão relacionados por uma constante e n^3 cresce ascintoticamente mais rápido do que $3n$.

Note que $8^n = 2^n \cdot 4^n$.

Sabendo que $\log(a^b) = b \cdot \log(a)$ e que os logs estão todos relacionados por constantes o resultado é imediato.

Questão 2

Dada uma lista de palavras de até 20 letras pretende-se imprimir na tela todas as palavras da lista que sejam anagramas de outras palavras na lista. Desenvolva um algoritmo (pseudo-código) para resolver tal problema e analise sua complexidade computacional. Respostas corretas cuja complexidade não seja $O(n \log(n))$ valerão apenas a metade da questão.

Anagrama: rearranjo das letras de outra palavra. Exemplo: ator / rota.

Solução: Como o tamanho das palavras está limitado por uma constante (20) podemos ordenar alfabeticamente as letras de cada palavra em $O(n)$ (tempo constante para cada palavra) e armazenar a nova lista a parte. Cada palavra ordenada deverá ter um ponteiro à palavra original. Uma vez feito isso é possível ordenar essa nova lista alfabeticamente em $O(n \log(n))$ usando por exemplo merge sort. Com isso, resta apenas percorrer a nova lista mais uma vez e toda vez que uma palavra ordenada seja igual à palavra ordenada seguinte ou à palavra ordenada anterior, imprimir dita palavra. Esse último passo pode ser feito em $O(n)$. Portanto o algoritmo todo é $O(n \log(n))$.

Questão 3

Considere o problema de determinar se existe um caminho simples (sem repetição de vértices) com pelo menos k arestas entre dois vrértices a e b de um grafo. Prove que esse problema é NP-Completo usando o fato de que o problema de determinar se existe um caminho hamiltoniano no grafo é NP-Completo. Não esqueça de mostar que o problema está em NP.

Caminho hamiltoniano: Existe um caminho simples que passe por todos os vértices do grafo?

Solução: Para verificar que o problema está em NP basta considerar um caminho entre a e b como um certificado de solução SIM. Para checar o certificado basta verificar que realmente seja caminho simples entre a e b e que tem pelo menos k arestas. Isto pode ser feito checando que o caminho comece em a, termine em b tenha $k+1$ vértices diferentes e que entre dois vértices consecutivos haja sempre uma aresta do grafo. Pode ser feito em tempo linear no tamanho do grafo.

Para reduzir caminho hamiltoniano ao novo problema copiamos o grafo de entrada com n vértices e acrescentamos of vértices a e b conetados a todos os vértices do grafo original. Esse grafo terá caminho entre a e b com pelo menos $n+1$ arestas se e somente se o grafo original tiver um caminho hamiltoniano. Ida) Pegue o caminho entre a e b e elimine as duas extremidades. O que restou deve ser um caminho hamiltoneano pois é um caminho simples com $n-1$ arestas. Volta) Pegue o caminho hamiltoniano e coloque a no começo e b no final. Sempre será possível pois tanto a quanto b tem aresta com todos os vértices do grafo original.

Questão 4

Para um grafo direcionado acíclico $D = (V, A)$, uma ordenação dos vértices de D, $v_1, \ldots, v_n$ é uma ordenação topológica se não existe arco (v_i, v_j) para $i > j$.

Dada uma ordenação topológica $v_1, \ldots, v_n$, dizemos que um par de vértices consecutivos (v_i, v_{i+1}) é um salto se $(v_i, v_{i+1}) \notin A(D)$.

Em cada um dos itens abaixo você deve elaborar um algoritmo. Seu algoritmo deve ser tão eficiente quanto possível.

(a) Escreva um algoritmo que receba um grafo direcionado acíclico D e encontre uma ordenação topológica dos vértices de D. Analise a complexidade de seu algoritmo.

(b) Escreva um algoritmo que receba um grafo direcionado acíclico D e uma ordenação topológica de D e determine o número de saltos da ordenação topológica. Analise a complexidade de seu algoritmo.

Solução:

1. *Duas soluções simples são possíveis: uma variação de busca em largura, onde um vértice entra na fila sempre que todos os seus vizinhos de entrada já foram visitados, e uma variação de busca em profundidade, na qual um vértice é adicionado (no início) da ordem topológica sempre que a chamada recursiva correspondente àquele vértice tiver terminado de visitar seus vizinhos.*

2. *Basta, para cada par de vértices consecutivos da ordem topológica, verificar se eles são adjacentes em D.*

Questão 5

No problema da mochila, é dada uma capacidade C, um conjunto de n objetos, juntamente com dois vetores de número inteiros v_i e p_i, correspondente aos valores e aos pesos dos objetos, respectivamente. Deve determinar qual o maior valor possível que pode ser transportado na mochila, respeitando sua capacidade.

Em uma variação do problema da mochila, conhecida como "problema da mochila com repetições" há uma quantidade ilimitada de cada um dos objetos. Desta forma, mais de uma cópia de um mesmo objeto pode ser inserida na mochila.

(a) Forneça um algoritmo de programação dinâmica para o problema da mochila com repetições. Analise a complexidade de seu algoritmo.

(b) Se $C > 2^n$, é possível resolver o problema de forma mais eficiente? Justifique.

Solução:

1. *A programação dinâmica é similar à do problema clássico. Se $M(i, j)$ é o maior lucro de uma mochila de capacidade j, podendo utilizar os i primeiros objetos, então temos que $M(i, j) = \max(M(i-1, j), M(i, j-p_i) + v_i)$, caso $j \geq p_i$. Os demais detalhes são idênticos ao caso clássico.*

2. *Neste caso, é mais eficiente utilizar o algoritmo de força bruta, de complexidade $O(n2^n)$.*

Questão 6

Dizemos que um vetor $v_1, \ldots, v_n$ está "exponencialmente ordenado" se, para todo i, os elementos $v_{2^i}, \ldots, v_{2^{i+1}-1}$ estão ordenados.

Nos itens abaixo, seu algoritmo deve ser tão eficiente quanto possível.

(a) Escreva um algoritmo para determinar se um vetor está exponencialmente ordenado. Analise sua complexidade.

(b) Dado um vetor exponencialmente ordenado e um elemento x, escreva um algoritmo para determinar se x pertence ao vetor. Analise sua complexidade.

Solução:

1. *Basta notar que os intervalos são disjuntos e percorrer cada intervalor comparando cada pa de elementos consecutivos.*

2. *O algoritmo mais eficiente é utilizar busca binária em cada um dos intervalos. Como há um máximo de* $O(\log n)$ *intervalos, temos complexidade total* $O(\log^2 n)$.