PUC-Rio | Departamento de Informática
Data: 07 de novembro de 2001 | Período: 2001.2

Análise de Algoritmos

Primeira Prova

Questão 1 (2.5 pontos): Assinale *verdadeiro* (V) ou *falso* (F) para cada afirmativa abaixo e justifique (através de demonstração ou contra-exemplo, conforme apropriado).

1. ( ): $\log 3^n = O(\log 2^n)$

   Verdade, pois $\log 3^n = n \log 3 = O(n) = O(n \log 2) = O(\log 2^n)$.

2. ( ): $f(n) = O(g(n)) \Rightarrow 2^{f(n)} = O(2^{g(n)})$

   Falso, considerando-se por exemplo o seguinte contra-exemplo: $f(n) = 2^n$ e $g(n) = 2^{n-1}$, $f(n) = O(g(n))$. Existem $c \in R$ e $n_0 \in N$ tais que $2^{2^n} \leq c \cdot 2^{2^{n-1}}, \forall n > n_0$? Ora, $2^{2^n} \leq c \cdot 2^{2^{n-1}} \Rightarrow c \geq 2^{2^n - 2^{n-1}} = 2^{2^{n-1}}$, que não pode ser uma constante.

3. ( ): $2^n = O(2^{n-1})$

   Verdade, pois $2^n = 2 \cdot 2^{n-1} = 2 \cdot O(2^{n-1}) = O(2^{n-1})$.

4. ( ): $n! = O(n^n)$

   Verdade, pois $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n \leq n \cdot n \cdot n \cdot \ldots \cdot n = n^n$.

5. ( ): $3^n = O(2^n)$

   Falso. Existem $c \in R$ e $n_0 \in N$ tais que $3^n \leq c \cdot 2^n, \forall n > n_0$? Ora, $3^n \leq c \cdot 2^n \Rightarrow c \geq (3/2)^n$, que não pode ser uma constante.

6. ( ): Sejam $f(n)$ e $g(n)$ duas funções definidas sobre os números inteiros. Então, $f(n) = O(g(n))$ ou $g(n) = O(f(n))$.

   Falso, considerando-se por exemplo o seguinte contra-exemplo, entre muitos outros possíveis: $f(n) = 0$ e $g(n) = 1$ para n par, $f(n) = 1$ e $g(n) = 0$ para n ímpar. Logo, não existem $c \in R$ e $n_0 \in N$ tais que $f(n) \leq c \cdot g(n)$ ou $g(n) \leq c \cdot f(n)$, $\forall n > n_0$.

Questão 2 (2.5 pontos): Seja $a = (a_1, a_2, \cdots, a_n)$ um vetor que armazena n itens (números reais). Especifique o algoritmo de menor complexidade possível para relacionar o número de ocorrências de cada item. Analise a complexidade deste algoritmo.

Os números são reais, logo não podem ser utilizados como índices. O algoritmo proposto é:

ordenar os elementos do vetor a
corrente $\leftarrow$ `flag`
para $i = 1, \ldots, n$ faça:
    se $a_i \neq$ *corrente* então
        se $i \geq 2$ então imprimir *corrente*, *contador*
        contador $\leftarrow 1$

$corrente \leftarrow a_i$
senão $contador \leftarrow contador + 1$
fim-para
imprimir $corrente, contador$

A complexidade do algoritmo é $O(n \log n)$, correspondente à etapa de ordenação do vetor. O loop "para" tem complexidade $O(n)$.

Questão 3 (2.5 pontos): Seja $G = [V, E]$ um grafo orientado representado por listas de sucessores, onde $V = \{1, 2, \ldots, n\}$ é o conjunto de nós e E é o conjunto de arcos. Especifique detalhadamente o algoritmo de menor complexidade possível para transformar esta representação em uma representação por lista de predecessores. Descreva cuidadosamente as estruturas de dados utilizadas e analise a complexidade deste algoritmo.

As listas de sucessores são percorridas, colocando-se os elementos apropriados nas listas de predecessores:

inicializar como vazias as n listas de predecessores
para $i = 1, \ldots, n$ faça:
se ainda houver um elemento p não-tratado na lista de sucessores do nó i
então inserir i na lista de predecessores do nó p
fim-faça

A inserção de um elemento numa lista de predecessores tem complexidade $O(1)$. Como este algoritmo visita todos os nós e todas as arestas exatamente uma vez, sua complexidade é $O(m+n)$.

Questão 4 (2.5 pontos): Seja $G = [V, E]$ um grafo conexo não-orientado, onde $V = \{1, 2, \ldots, n\}$ é o conjunto de nós e E é o conjunto de arestas. Define-se um vértice de articulação de G como sendo qualquer vértice $v \in V$ cuja eliminação transforma G em um grafo não-conexo. Especifique detalhadamente um algoritmo de baixa complexidade para determinar os vértices de articulação de G. Analise a complexidade deste algoritmo. Sugestão: pense numa idéia simples, baseada na utilização de outro(s) algoritmo(s) que você conheça.

Como um vértice de articulação de G é qualquer vértice $v \in V$ cuja eliminação transforma G em um grafo não-conexo, basta testar esta condição para cada vértice:

para $i = 1, \ldots, n$ faça:
eliminar o nó i temporariamente de G
aplicar uma busca em profundidade ao grafo resultante
se todos os demais nós forem alcançáveis
então o nó i não será um vértice de articulação
senão o nó i será um vértice de articulação
recolocar o nó i definitivamente em G
fim-para

Supondo-se o grafo G armazenado sob forma de listas de adjacências, a eliminação e a reinserção de um nó qualquer podem ser feitas em tempo $O(m + n)$. Da mesma forma, cada busca em profundidade tem complexidade $O(m+n)$. Logo, a complexidade deste algoritmo (que não necessariamente é o melhor possível) é $O(n \cdot (m + n))$.

Observações:

- Prova sem consulta de qualquer espécie.

- Apresente os algoritmos solicitados sob forma de descrições algorítmicas e formais (pseudo-linguagens, por exemplo), e não como códigos em determinada linguagem de programação ou usando frases imprecisas ou ambíguas em Português ou outra língua.
- A correção da prova levará em consideração a exatidão, o rigor e o formalismo na especificação dos algoritmos e das estruturas de dados.
- Justifique sua afirmações e as idéias empregadas no projeto dos algoritmos.
- Defina a notação utilizada e explique os conceitos empregados.
- Justifique todos seus cálculos e afirmações.
- Erros crassos de Português serão penalizados.