# Exame Preliminar para o Doutorado

*IME–USP, Fevereiro, 1999*

## Prova de Análise de Algoritmos

**Instruções:**

($i$) O candidato pode resolver todas as questões.

($ii$) A banca considerará questões cujos valores somem até 10 pontos de modo que a soma total das notas obtidas seja máxima.

($iii$) Mencione os teoremas e propriedades usados para justificar suas afirmações.

**Questão 1** [1 ponto]
É verdade que $2^{n+1} = O(2^n)$? E será que $2^{2n} = O(2^n)$?

**Questão 2** [1 ponto]
Encontram-se na literatura duas definições alternativas para $\Omega(n)$; vamos diferenciá-las por índices:

$f(n) = \Omega_1(g(n))$ se existem $n_0$, $A > 0$ tais que $f(n) > Ag(n)$ para $n > n_0$;
$f(n) = \Omega_2(g(n))$ se existe $A > 0$ tal que $f(n) > Ag(n)$ para infinitos valores de $n$.

Prove ou dê contra-exemplo:

1. $f(n) = \Omega_1(g(n)) \Longrightarrow f(n) = \Omega_2(g(n))$.
2. $f(n) = \Omega_2(g(n)) \Longrightarrow f(n) = \Omega_1(g(n))$.

**Questão 3** [1 ponto]
Dada uma matriz $A_{n\times m}$ de inteiros considere os seguintes algoritmos baseados em comparações para encontrar o maior elemento:

1. Acha-se o máximo de cada linha, armazenando-se os resultados num vetor, e depois acha-se o máximo do vetor.
2. Acha-se o máximo de cada coluna, armazenando-se os resultados num vetor, e depois acha-se o máximo do vetor.

Supondo-se que $n < m$, qual deles realiza menos comparações?

**Questão 4** [1 ponto]
Escreva um algoritmo $O(n)$ que ordena $n$ números inteiros no intervalo $[1, n^3]$. Justifique sua resposta.

**Questão 5** [2 pontos]

Você tem um arquivo ordenado de $n$ strings de comprimento no máximo 100; os strings são sobre um alfabeto de 255 caracteres. Aí passa um raio cósmico/vírus/professor e altera aleatoriamente o centésimo caractere (isto é, o último caractere) de cada string que tenha comprimento 100. Mostre como reordenar o arquivo com $O(n)$ comparações de strings. Note que a comparação de strings é a única forma permitida de acesso aos dados - em particular, não se dispõe de um mecanismo de acesso a uma posição específica de um string.

**Questão 6** [2 pontos]

O algoritmo abaixo recebe um grafo dirigido $G$, com vértices numerados $1, 2, \ldots, n$ por meio de uma matriz quadrada $W$, onde $W_{ij} > 0$ se existe aresta de $i$ para $j$ e $W_{ij} = \infty$ caso contrário. Ele devolve uma matriz $D$ onde $D_{ij}$ é o custo mínimo de um caminho dirigido de $i$ a $j$, usando as entradas de $W$ como custos nas arestas.

```
1 D ← W
2 for k ← 1 to n
3     do for i ← 1 to n
4            do for j ← 1 to n
5                   d_ij ← min(d_ij, d_ik + d_kj)
6            od
7     od
8 od
9 return D
```

Os números nas linhas são para facilitar a referência.

1. Analise a complexidade desse algoritmo.
2. Mostre como alterar esse algoritmo para obter, junto com o custo mínimo, um caminho de custo mínimo para cada par de vértices. Qual a complexidade desse novo algoritmo?

**Questão 7** [3 pontos]

($i$) Queremos manter uma fila de prioridades com itens cujas prioridades são dadas pelas suas chaves, que são inteiros não-negativos menores ou iguais a um inteiro fixo $k$. (Quanto menor a chave, maior a prioridade; desempates podem ser feitos de forma arbitrária.) As operações que executaremos sobre a nossa fila serão as operações de ($a$) inicialização da fila (como a fila vazia), ($b$) inserção de um item com uma dada chave, ($c$) extração de um item de máxima prioridade, ($d$) reclassificação de um dado item, alterando sua chave para um valor dado. Descreva como podemos implementar tal fila de forma que cada uma das operações acima possa ser executada em tempo constante. (Aqui é crucial que assumimos que $k$ é constante.)

($ii$) Queremos um algoritmo $\mathcal{A}$ que encontra uma árvore geradora de peso mínimo. A entrada do algoritmo será um grafo $G$, com uma função peso nas arestas

$$w\colon E(G) \to \mathbb{Z}.$$

Suponha que temos as seguintes hipóteses adicionais sobre as nossas entradas: (H1) o grafo $G$ sempre tem $O(|V(G)|)$ arestas, (H2) os pesos das arestas são sempre não-negativos e menores ou iguais a um inteiro fixo $k$, isto é, $0 \leq w(e) \leq k$ para toda aresta $e$ de $G$. Descreva um tal algoritmo $\mathcal{A}$ de complexidade de tempo $O(|V(G)|)$.

**Questão 8** [3 pontos]

Um banco de dados tem suporte para conjuntos do seguinte tipo:

*elemento*: record

**valor**: *string*

**ref**: *pointer.*

O fabricante garante que, dada uma referência a um elemento **E**, é possível, em tempo constante, ler e alterar **E.ref**. Além disso, uma função especial permite zerar o campo **ref** de todos os membros de um conjunto em tempo desprezível.

Nesse banco de dados foram montados dois conjuntos, de *Nomes* e *Sobrenomes*. É dada (como uma lista ligada) uma coleção $\mathcal{C}$ de registros, cada um contendo, de significante, dois campos - **nome** e **sobrenome**, que são referências para elementos dos respectivos conjuntos. Pede-se para colocar esses registros em ordem alfabética por nome-sobrenome, da forma usual. A comparação de strings é uma operação primitiva.

Considere as seguintes duas formas de atacar o problema:

1. Ordenar diretamente, usando um bom algoritmo (por exemplo, mergesort), usando comparações de strings para comparar os registros:
   **R1** < **R2** se e só se **R1.nome→valor** < **R2.nome→valor**
   ou **R1.nome→valor** = **R2.nome→valor**
   e **R1.sobrenome→valor** < **R2.sobrenome→valor**.

2. (a) Separar em blocos com o mesmo nome:
   Monta-se uma lista $\mathcal{L}$ de nomes usados por registros em $\mathcal{C}$. Em cada um desses nomes pendura-se uma lista ligada de referências aos registros que usam esse nome. Essa lista de listas é montada com uma varredura em $\mathcal{C}$, atribuindo-se aos elementos de *Nomes* uma referência à sua posição em $\mathcal{L}$. Assim, para cada registro **R**,

   se **R.nome→ref**=0, coloque **R.nome** em $\mathcal{L}$, colocando em **R.nome→ref** sua posição; pendure na respectiva sublista uma referência a **R**.

   (b) Ordenar cada sublista por sobrenome dos respectivos registros usando mergesort.

   (c) Ordenar $\mathcal{L}$ pelos nomes usando mergesort.

   (d) Concatenar tudo, usando a lista final de referências para ordenar $\mathcal{C}$.

**Problema:** Compare as ordens das complexidades desses algoritmos sob as condições:

($i$) Totalmente gerais.

($ii$) Cada nome ocorre $\Theta((\lg n)^r)$ vezes, para algum $r > 1$.

($iii$) Cada nome ocorre $\Theta(n^\epsilon)$ vezes, para algum $\epsilon$ entre 0 e 1.