MAC 6711 - Tópicos de Análise de Algoritmos

Departamento de Ciência da Computação

Primeiro semestre de 2014

Lista 6

1. **(32.1-2 do CLRS)** Mostre que, se todos os caracteres do padrão $P[1\,..\,m]$ são distintos, o algoritmo ingênuo que busca P em um texto $T[1\,..\,n]$ pode ser modificado para consumir tempo $\mathrm{O}(n)$.

2. **(32.1-3 do CLRS)** Suponha que o padrão P e o texto T são cadeias de caracteres de comprimentos m e n respectivamente, escolhidas aleatoriamente de um alfabeto $\Sigma_d = \{0, 1, \ldots, d-1\}$, para $d \geq 2$. Mostre que o número esperado de comparações entre caracteres do texto e do padrão no algoritmo trivial é $(n - m + 1)\frac{1-d^{-m}}{1-d^{-1}} \leq 2(n - m + 1)$.

3. **(32.1-4 do CLRS)** Suponha que o padrão P pode conter ocorrências de um caracter *vão* $\diamondsuit$, que pode ser casado a uma cadeia arbitrária de caracteres (inclusive com a cadeia vazia). Por exemplo, o padrão $ab\diamondsuit ba\diamondsuit c$ ocorre no texto *cabccbacbacab* de duas maneiras diferentes: *c***ab***cc***ba***cba***c***ab* e *c***ab***ccbac***ba***c***ab*. Note que o caracter vão pode ocorrer um número arbitrário de vezes no padrão, mas não ocorre no texto nenhuma vez. Descreva um algoritmo polinomial para determinar se um tal padrão ocorre em um texto T, e analise o consumo de tempo do seu algoritmo.

4. **(13.3.2 e 13.4.4 do PF)** Dê um exemplo em que a versão 1 do algoritmo de Boyer-Moore faz o maior número de comparações entre caracteres do padrão e do texto. Dê um exemplo em que a versão 2 do algoritmo de Boyer-Moore faz o maior número de comparações entre caracteres do padrão e do texto.

5. **(13.4.1 do PF)** Calcule a tabela v_2 para o caso em que todos os caracteres do padrão são iguais. Calcule a tabela v_2 para o caso em que todos os caracteres do padrão são distintos.

6. **(13.5.1 do PF)** Escreva o código do algoritmo de Boyer-Moore (que usa v_1 e v_2).

7. **(32.4-1 do CLR)** Calcule a função prefixo π para o padrão *ababbabbababbababbabb* quando o alfabeto é $\Sigma = \{a, b\}$.

8. **(32.4-3 do CLR)** Mostre como determinar as ocorrências de um padrão $P[1\,..\,m]$ em um texto $T[1\,..\,n]$ examinando a função prefixo para a cadeia de caracteres PT (concatenação de P com T).

9. **(32.4-5 do CLR)** Descreva um algoritmo linear para determinar se um texto T é uma rotação cíclica de uma outra cadeia de caracteres T'. Por exemplo, *arco* e *coar* são rotações uma da outra.

10. Dados $n + 1$ pontos $p, p_1, \ldots, p_n$, dizemos que p é uma *combinação convexa* de $p_1, \ldots, p_n$ se existem números reais não-negativos $\lambda_1, \ldots, \lambda_n$ tais que (a) $\sum_{i=1}^{n} \lambda_i = 1$ e (b) $\sum_{i=1}^{n} \lambda_i\, p_i = p$. O *fecho convexo* de uma coleção finita de pontos é o conjunto de todas as combinações convexas de pontos da coleção. O *casco convexo* de uma coleção finita de pontos é o polígono que delimita o fecho convexo dessa coleção. Como o casco convexo é um polígono, ele pode ser dado por uma seqüência de pontos: os vértices do polígono. Note que os pontos dessa seqüência são sempre pontos da coleção original de pontos.

    O seguinte algoritmo, de Graham, determina o casco convexo da coleção $\{p_1, \ldots, p_n\}$. Vamos supor, para simplificar, que não há na coleção três pontos colineares. No pseudo-código abaixo, para três pontos distintos p, q e w, denotamos por $\theta(p, q, w)$ o ângulo entre a reta que passa por p e q e a reta que passa por q e w.

```
GRAHAM(p, n)
1  PRELIMINARES (p, n)
2  p[n + 1] ← p[1]
3  c[1] ← p[1]
4  t ← 1
5  para k ← 2 até n faça
6      t ← t + 1
7      c[t] ← p[k]
8      enquanto t > 2 e θ(c[t − 1], c[t], p[k + 1]) ≤ 180^o faça
9          t ← t − 1
10 devolva c
```

```
PRELIMINARES(p, n)
min ← 1
para i ← 2 até n faça
    se y(p[i]) < y(p[min])
        então min ← i
p[1] ↔ p[min]
seja q um ponto tal que a reta que passa por q e p[1] é paralela ao eixo x
ordene p[2 . . n] de modo que θ(q, p[1], p[2]) < · · · < θ(q, p[1], p[n])
```

Demonstre que as linhas 2 a 10 do algoritmo de Graham consomem tempo $\mathrm{O}(n)$.

11. Num exercício de uma lista anterior, um aluno chamado Omar descreveu um algoritmo bem interessante para encontrar o "envelope" de uma coleção de retas dadas.

    Uma reta (no plano) é determinada pela equação $ax + by = c$. Ou seja, uma tal reta é dada pelos três números a, b e c. Neste problema, é dada uma coleção de retas $\{r_1, \ldots, r_n\}$, onde cada reta r_i é dada por a_i, b_i e c_i, com $b_i \neq 0$, O *coeficiente angular* de uma reta r dada por $ax + by = c$, com $b \neq 0$, é o número $-a/b$. (De fato, como $b \neq 0$, podemos reescrever a equação da reta como $y = (-ax + c)/b$.) O *deslocamento* de r é o número c/b.

    No código abaixo, a rotina INTER recebe duas retas não-paralelas como parâmetros e devolve o ponto de interseção entre elas.

```
OMAR(r, n)
PRELIMINARES (r, n)
env[0] ← (1, 0, −∞) hspace*2cm ▷ Reta sentinela x = −∞
env[1] ← r[1]
env[2] ← r[2]
t ← 2
para k ← 3 até n faça
    enquanto t > 1 e x(INTER(env[t], r[k])) ≤ x(INTER(env[t − 1], env[t])) faça
        t ← t − 1
    t ← t + 1
    env[t] ← r[k]
```

```
PRELIMINARES(r, n)
ordene r[1 . . n] pelo coeficiente angular, deixando,
 em caso de empate, a de maior deslocamento antes
j ← 1                          ▷ Remove retas paralelas desnecessárias
para i ← 2 até n faça
    se r[j] e r[i] não têm o mesmo coeficiente angular
        então j ← j + 1
                r[j] ← r[i]
```

Demonstre que as linhas 2 a 10 do algoritmo do Omar consomem tempo $O(n)$.

12. Considere a seguinte alternativa para representar um contador binário com k bits. O contador é representado por dois vetores com k posições, $P[0\,..\,k-1]$ e $N[0\,..\,k-1]$, onde, para cada i, no máximo um entre $P[i]$ e $N[i]$ vale 1. (O valor real do contador é a diferença $P - N$.)

    Abaixo estão implementações das operações de **Incrementa** e **Decrementa** para tal representação. Para simplificar, assumimos que o número k é grande o suficiente para que os algoritmos abaixo não acessem um índice inválido.

    Algoritmo Incrementa (P, N)
    ```
    i ← 0
    enquanto P[i] = 1 faça
        P[i] ← 0
        i ← i + 1
    se N[i] = 1
        então N[i] ← 0
        senão P[i] ← 1
    ```

    Algoritmo Decrementa (P, N)
    ```
    i ← 0
    enquanto N[i] = 1 faça
        N[i] ← 0
        i ← i + 1
    se P[i] = 1
        então P[i] ← 0
        senão N[i] ← 1
    ```

    Abaixo está um exemplo destes algoritmos em ação. (Note que qualquer número exceto o zero pode ser representado de diversas maneiras.)

$P = 10001$		$P = 10010$		$P = 10011$		$P = 10000$		$P = 10000$		$P = 10000$		$P = 10001$
$N = 01100$	$+1$	$N = 01100$	$+1$	$N = 01100$	$+1$	$N = 01000$	-1	$N = 01001$	-1	$N = 01010$	$+1$	$N = 01010$
$P - N = 5$		$P - N = 6$		$P - N = 7$		$P - N = 8$		$P - N = 7$		$P - N = 6$		$P - N = 7$

    Suponha que o contador começa de $(0, 0)$ (ou seja, $P = 0^k$ e $N = 0^k$), e sejam aplicadas n operações, cada uma delas sendo um **Incrementa** ou um **Decrementa**. Suponha que $n < 2^k$, de modo que de fato os algoritmos não acessem um índice inválido dos vetores. Neste caso, o custo de pior caso do **Incrementa** e do **Decrementa** é $\Theta(\lg n)$.

    Prove que o custo amortizado do **Incrementa** e do **Decrementa** é $O(1)$.