MAC 6711 - Tópicos de Análise de Algoritmos

Departamento de Ciência da Computação
Primeiro semestre de 2015

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 $O(n)$.

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***c***a***b* e *c***ab***ccbac***bac***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 e 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. Considere uma pilha implementada em um vetor com as operações usuais de `empilha` e `desempilha`. Para evitar *overflow*, ou seja, para lidar com a situação em que a pilha fica cheia, pode-se implementar uma operação que denominaremos de `realoca`, acionada pela operação `empilha` sempre que a pilha fica cheia. A operação `realoca` aloca um novo vetor de tamanho duas vezes maior que o vetor corrente, copia o conteúdo corrente da pilha para o novo vetor e desaloca o vetor corrente. Suponha que as operações de alocação e desalocação de memória (correspondentes ao `malloc` e `free`, digamos) consumam tempo constante, ou seja, $O(1)$. Se a pilha está com s elementos, qual é o tempo de pior caso das operações `empilha` e `desempilha` neste caso? Faça uma análise amortizada e deduza o custo amortizado destas operações. Ou seja, considere uma seqüência de n operações `empilha` e `desempilha`, realizadas sobre uma pilha inicialmente vazia, e calcule o custo amortizado por operação. Suponha que o vetor alocado inicialmente para a pilha tem apenas uma posição. Sua análise deve ser o mais justa possível em termos assintóticos.

8. 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, proposto por 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)
1  min ← 1
2  para i ← 2 até n faça
3      se y(p[i]) < y(p[min])
4          então min ← i
5  p[1] ↔ p[min]
6  seja q um ponto tal que a reta que passa por q e p[1] é paralela ao eixo x
7  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 $O(n)$.

9. 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)
 1  PRELIMINARES (r, n)
 2  env[0] ← (1, 0, −∞)                    ▷ Reta sentinela x = −∞
 3  env[1] ← r[1]          env[2] ← r[2]
 5  t ← 2
 6  para k ← 3 até n faça
 7     enquanto t > 1 e x(INTER(env[t], r[k])) ≤ x(INTER(env[t − 1], env[t])) faça
 8        t ← t − 1
 9     t ← t + 1
10     env[t] ← r[k]
```

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

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

10. 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.

```
Incrementa (P, N)
1.  i ← 0
2.  enquanto P[i] = 1 faça
3.     P[i] ← 0
4.     i ← i + 1
5.  se N[i] = 1
6.     então N[i] ← 0
7.     senão P[i] ← 1
```

```
Decrementa (P, N)
1.  i ← 0
2.  enquanto N[i] = 1 faça
3.     N[i] ← 0
4.     i ← i + 1
5.  se P[i] = 1
6.     então P[i] ← 0
7.     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.)

$$\begin{array}{c} P = 10001 \\ N = 01100 \\ P - N = 5 \end{array} \quad {\scriptstyle +1} \quad \begin{array}{c} P = 10010 \\ N = 01100 \\ P - N = 6 \end{array} \quad {\scriptstyle +1} \quad \begin{array}{c} P = 10011 \\ N = 01100 \\ P - N = 7 \end{array} \quad {\scriptstyle +1} \quad \begin{array}{c} P = 10000 \\ N = 01000 \\ P - N = 8 \end{array} \quad {\scriptstyle -1} \quad \begin{array}{c} P = 10000 \\ N = 01001 \\ P - N = 7 \end{array} \quad {\scriptstyle -1} \quad \begin{array}{c} P = 10000 \\ N = 01010 \\ P - N = 6 \end{array} \quad {\scriptstyle +1} \quad \begin{array}{c} P = 10001 \\ N = 01010 \\ P - N = 7 \end{array}$$

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 $O(1)$.