Análise de Algoritmos

Segunda Lista de Exercícios

Reconhecimento de Padrões em Palavras

1. Indução em i.

   BI: Para $i = 1$, $f(i) = 0$ e $F[i] := 0$. Na primeira iteração com `while`, $j = 0$. Então o segundo `while` não será executado. Portanto, ao incrementar os índices i e j tem-se que $j < i$ e $\lambda = P_0 \sqsupset_{\max} P_1$. Observe que em ambos os casos $p_i = p_j$ e $p_i \neq p_j$ tem-se que $F[2] = f(2)$.

   PI: Suponha que $F[l] = f(l)$, $\forall l \leq i$ e, no início do `while` mais externo, $0 < j < i$ e $P_{j-1} \sqsupset_{\max} P_{i-1}$. O primeiro passo será provar que ao incrementar os índices i e j tem-se a invariante $0 < j < i$ e $P_{j-1} \sqsupset_{\max} P_{i-1}$.

   Se $p_i \neq p_j$, então gera-se uma sequência $j = j_1, j_2, \ldots, j_r$ através da iteração com o segundo `while`, onde, por hipótese de indução, $j_r = 0$ ou $P_{j_l - 1} \sqsupset P_{j_{(l-1)}-1}$ e $p_{j_l} \neq p_{j(l-1)}$, $\forall 1 < l \leq r$ e $p_{j_r} = p_i$. Observe que, por hipótese de indução, $P_{j-1} \sqsupset_{\max} P_{i-1}$. Suponha que $P_m \sqsupset_{\max} P_i$ para algum $m > j_r$. Então $j_l < m < j_{l-1}$ para algum $1 < l < r$. Assim, $P_{j_l - 1} \sqsupset P_{m-1} \sqsupset P_{j_{(l-1)}-1}$ e $p_m = p_i \neq p_{j_{(l-1)}}$. Isso contradiz a maximalidade de $f(j_{(l-1)}) = j_l$ com essas propriedades, dada pela definição da função f. Portanto, $P_{j_r} \sqsupset_{\max} P_i$. Assim, ao sair do `while`, $j = j_r < i$. Após o incremento de i e j tem-se a invariante. A prova para $j_r = 0$ é análoga.

   Se $p_i = p_j$, então de $P_{j-1} \sqsupset_{\max} P_{i-1}$ tem-se que $P_j \sqsupset_{\max} P_i$. Assim, após o incremento de i e j obtém-se a invariante.

   Observe que o valor de j (que nesse ponto é igual a $j+1$) não é alterado no restante do código, antes do início da próxima iteração. Agora basta provar que as últimas linhas do código atribuem o valor correto para $F[i+1]$.

   Se $p_{i+1} \neq p_{j+1}$, então $F[i+1] = j+1$ e de $P_j \sqsupset_{\max} P_i$ tem-se que $f(i+1) = j+1$.

   Se $p_{i+1} = p_{j+1}$, então $F[i+1] = F[j+1] \overset{HI}{=} f(j+1)$. Suponha que $f(j+1) = r+1$. Então, pela definição da função f, r é o maior valor tal que $P_r \sqsupset P_j$ e $p_{r+1} \neq p_{j+1}$. Logo $P_r \sqsupset P_i$ e $p_{r+1} \neq p_{i+1}$. Suponha que $f(i+1) = l+1 > r+1$. Tem-se que $l \neq j$, pois, pela definição de f, $p_{l+1} \neq p_{i+1}$. Suponha que $l > j$. Então $P_l \sqsupset P_i$, mas $P_j \sqsupset_{\max} P_i$. Contradição. Se $l < j$, então, como $P_j \sqsupset P_i$, $P_l \sqsupset P_j$ e $p_{l+1} \neq p_{i+1} = p_{j+1}$. Mas r é maximal com essa propriedade. Contradição. Logo, $f(i+1) = r+1 = f(j+1)$. Portanto, para $f(j+1) > 0$, $F[i+1] = f(i+1)$. Para $f(j+1) = 0$ a prova é análoga.

2. AAAB

i	1	2	3	4
$F[i]$	0	0	0	3

   ABABBABABBAB

i	1	2	3	4	5	6	7	8	9	10	11	12
$F[i]$	0	1	0	1	3	0	1	0	1	3	0	1

ABABBABABBABBABABBAB

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$F[i]$	0	1	0	1	3	0	1	0	1	3	0	1	8	0	1	0	1	3	0	0

3. A idéia é incluir uma regra para o estado correspondente a ter encontrado o padrão no texto. Assim, basta incluir um caracter que não faz parte do alfabeto na posição $m + 1$, onde $|P| = m$, e permitir mais uma iteração do loop mais externo do algoritmo (while).

```
calcule_erro_KMP()
{
  int i=1, j=0;

  padrao[m+1]='#';
  erro_KMP[i]=0;
  while (i<=m)
    {
     while (j>0 && padrao[i] != padrao[j]) j=erro_KMP[j];
     i++;
     j++;
     if (padrao[i]==padrao[j]) erro_KMP[i]=erro_KMP[j];
     else erro_KMP[i]=j;
   }
}
```

4. AAAB

i	1	2	3	4
$mJ[i]$	7	6	5	1

ABABBABABBAB

i	1	2	3	4	5	6	7	8	9	10	11	12
$mJ[i]$	16	15	14	13	12	16	15	14	6	12	3	1

ABABBABABBABBABABBAB

i	1	2	3	4	5	6	7	8	9	10
$mJ[i]$	32	31	30	29	28	27	26	25	19	23
i	11	12	13	14	15	16	17	18	19	20
$mJ[i]$	22	21	20	11	23	22	6	20	3	1

5. Analogamente a mudança realizada no algoritmo KMP para tratar todas as ocorrências do padrão no texto, precisa-se incluir um estado para tratar o caso no qual o padrão é encontrado. Para isso, basta incluir na posicão 0 um caracter que não faça parte do alfabeto que compõem o texto. Assim, para um padrão P qualquer de comprimento m teria-se um valor $mJ[0]$ dando a informação de qual salto seria apropriado, de acordo com o algoritmo BM, após encontrar uma ocorrência de P.

6. Opcional

7. O algoritmo KMP tem um bom desempenho para alfabetos com poucos caracteres, como o binário por exemplo, enquanto o algortimo BM se destaca para busca de padrões em linguagem natural onde o padão tem comprimento maior ou igual a 5 (veja 5.3[Baa88]). Essa diferença se deve ao fato da pre-computação do BM não compensar para tais alfabetos, pois a chance de haver um padrão onde não apareça algum desses caracteres é mínima.

Programação Dinâmica

8. (a) Seja $\#o(n)$ o número de operacões aritméticas para determinar a fatoração ótima de multiplicação de uma sequência de n matrizes. Observe que $\#o(2) = 4$ e $\#o(1) = 0$. Suponha que $\#o(i) \geq 2^i$, para todo $1 < i < n$. Note que $\#o(n) = 2\sum_{i=1}^{n-1} \#o(i)$. Portanto,

$$\begin{aligned} T(n) = 2\sum_{i=1}^{n-1} \#o(i) &= 2\sum_{i=2}^{n-1} \#o(i) \\ &\geq 2\sum_{i=2}^{n-1} 2^i \quad (IH) \\ &= 2\big(2^n - 1 - 3\big) \\ &= 2^3(2^{n-2} - 1) \\ &\geq 2^n, \quad \text{para}\, n > 2 \end{aligned}$$

   Portanto, $T(n) \in \Omega(2^n)$

   (b) Duas matrizes $M_{n\times n}$ e $factor_{n\times n}$ são inicializadas, onde n é o comprimento da sequência de matrizes. O primeiro passo do algoritmo é preencher a diagonal $M[i,i]$ com 0's. Os passos seguinte serão calcular para cada diagonal acima da diagonal principal o valor de $M[i,j]$. Esse valor representa o número ótimo de operações para fazer a multiplicação das matrizes $A_i, \cdots, A_j$. NA matriz $factor$ são guardados os valores de k tal que o número ótimo é atingido. Para a primeira diagonal teremos i variando de 1 até $n-1$ onde, para cada i, $j = i + 1$. Como essa diferença entre i e j representa uma multiplicação entre duas matrizes, tem-se trivialmente que $M[1,2] = 600$ e $factor[1,2] = 1$, $M[2,3] = 1200$ e $factor[2,3] = 2$, $M[3,4] = 2400$ e $factor[3,4] = 3$. Assim, ao final da computação para $diagonal = 1$ tem-se que

$$M = \begin{pmatrix} 0 & 600 & - & - \\ - & 0 & 1200 & - \\ - & - & 0 & 2400 \\ - & - & - & 0 \end{pmatrix} \quad factor = \begin{pmatrix} - & 1 & - & - \\ - & - & 2 & - \\ - & - & - & 3 \\ - & - & - & - \end{pmatrix}$$

Para $diagonal = 2$, teremos i variando de 1 até 2. Como exemplo, para i igual a 1 tem-se que $j = 3$. Assim

$$M[1,3] = min_{1 \leq k \leq 3}(M[1,k] + M[k+1,j] + 800d_k)$$

Observe que para calcular esse valor, apenas valores de $M[i,j]$ já calculados anteriormente são utilizados. Assim, $M[1,3] = 2800$ e $factor[1,3] = 1$. Analogamente tem-se que $M[2,4] = 1520$ e $factor[2,4] = 3$. Finalmente, para $diagonal = 3$, temos que $M[1,4] = 1680$ e $factor[1,4] = 1$. Ao final da computação tem-se então

$$M = \begin{pmatrix} 0 & 600 & 2800 & 1680 \\ - & 0 & 1200 & 1520 \\ - & - & 0 & 2400 \\ - & - & - & 0 \end{pmatrix} \quad factor = \begin{pmatrix} - & 1 & 1 & 1 \\ - & - & 2 & 3 \\ - & - & - & 3 \\ - & - & - & - \end{pmatrix}$$

(c) O algoritmo só é aplicado para um número n de matrizes maior ou igual a 3, onde para $n = 3$ uma comparacão é feita entre d_1 e d_2 para determinar qual é a maior. Assim,

$$\begin{aligned} T(n) = \sum_{i=3}^{n}(i-2) &= \sum_{i=1}^{n-2} i \\ &= \frac{1}{2}(n-2)(n-1) \\ &= \frac{n^2}{2} - \frac{3}{2}n + 1 \end{aligned}$$

Portanto, $T(n) \in \Theta(n^2)$.

Seja A_1, A_2A_3, A_4 a sequência de matrizes apresentada em (b). Pelo algoritmo com a técnica voraz teria-se a ordem de multiplicação representada por $A_1 * (A_2 * (A_3 * A_4))$, com o número de operações de $(15 \cdot 40 \cdot 4) + (2 \cdot 15 \cdot 4) + (20 \cdot 2 \cdot 4) = 2680$. Por (b) sabe-se que a fatoração ótima é representada por $A_1 * ((A_2 * A_3) * A_4)$, com 1680 operações.

(d)
```
showOrder(int l, int m){

  int k;

  if (l == m)
     printf("A%d", l);

  else

    {
```

```
      k = factor[l][m];
      printf("(");
      showOrder(l,k);
      printf("*");
      showOrder(k+1, m);
      printf(")");

    }
}
```

Veja algoritmo 6.2 em [Baa88].

9.

10. (a)

$$maxcom(x\delta, y\sigma) = \begin{cases} maxcom(x,y)+1, & \text{se } \delta = \sigma \\ \texttt{max}\big(maxcom(x\delta,y), maxcom(x,y\sigma)\big), & c.c. \end{cases}$$

(b)

```
BEGIN

 FOR (i=1,i<=n,i++)
 {
   IF (x[i] == y[1])
      M[i,1] = 1;
   ELSE
      M[i,1] = 0;
 }

 FOR (j=1,j<=n,j++)
 {
   IF (x[1] == y[j])
      M[1,j] = 1;
   ELSE
      M[1,j] = 0;
 }

 FOR (i=2,i<=n,i++)
 {
    FOR (j=2,j<=n,j++)
    {
      IF (x[i] == y[j])
        M[i,j] = M[i-1,j-1] + 1;
      ELSE
        M[i,j] = max(M[i-1,j],M[i,j-1]);
```

```
    }
}

END.
```

Tem-se n comparações para cada um dos dois primeiros FOR's . Assim,

$$T(n) = 2n + \sum_{i=2}^{n} \sum_{j=2}^{n} 2 = 2n + 2(n-1)^2 < 2n + 2n^2$$

Portanto, $T(n) \in O(n^2)$.

(c)

```
MostraSubSeq()

{

i,j= n;
k = M[i,j];

WHILE (k>0)
 {


   IF (M[i,j] == M[i-1,j])
     i--;

   ELSE IF (M[i,j] == M[i,j-1])
             j--;

        ELSE
          {
            PX[k] = i;
            PY[k] = j;
            i--;
            j--;
            k--;
          }

 }

 k = M[n,n];

PRINT "Um subsequencia maximal comum a x e y:"

FOR (i=0,i<=k,i++)
    PRINT "x PX[i]";
```

```
 FOR (i=0,i<=k,i++)
     PRINT "y PY[i]";

}
```

11. (a)

```
BEGIN

 FOR (i=0,i<=m,i++)
   D[i,0] = i;


 FOR (j=1,j<=n,j++)
   D[0,j] = 0;

 FOR (i=1,i<=m,i++)
 {
    FOR (j=1,j<=n,j++)
    {
      k = D[i-1,j-1];

      IF (P[i] != T[j])
        k++;

      l = min(D[i-1,j],D[i,j-1]) +1;
      D[i,j] = min(k,l);
    }
 }

END.
```

(b)

12. (a)

```
BEGIN

 FOR (i=0,i<=n,i++)
   M[i,0] = 1;

 FOR (j=1,j<=C,j++)
   M[0,j] = 0;

 FOR (i=1,i<=n,i++)
 {
    FOR (j=1,j<=C,j++)
    {
      k = S[i];
```

```
        IF ( (j-k >= 0 AND M[i-1,j-k] == 1) OR M[i-1,j] == 1)
          M[i,j] = 1;
        ELSE
         M[i,j] = 0;
      }
   }

END.
```

(b)