Sabrina Araújo da Silva nº USP 12566182

MAC0338 - ANÁLISE DE ALGORITMOS

LISTA 5

exercícios 1 e 3

1. **Problema 15-2 do CLRS** (Como imprimir nitidamente) Considere o problema de imprimir nitidamente um parágrafo em uma impressora. O texto de entrada é uma sequência de n palavras de comprimentos $l_1, l_2, \ldots, l_n$, medidos pelo número de caracteres. Queremos imprimir esse parágrafo com nitidez em uma série de linhas que contêm no máximo M caracteres cada uma. Nosso critério de "nitidez" é dado a seguir. Se uma determinada linha contém palavras de i até j, onde $i \leq j$, e deixamos exatamente um espaço entre as palavras, o número de espaços extras no final da linha é $M - j + i - \sum_{k=i}^{j} l_k$, que deve ser não-negativo para que as palavras caibam na linha. Desejamos minimizar a soma, sobre todas as linhas exceto a última, do cubo do número de espaços extras no final das linhas. Escreva um algoritmo de programação dinâmica para imprimir um parágrafo de n palavras nitidamente em uma impressora. Analise o tempo de execução e os requisitos de espaço do seu algoritmo.

```
cria vetores r[n,n], s[1...n] e p[1...n]

IMPRIMEPARAGRAFO (M,n,l)
1  para i ← 1 até n faça
2      para j ← i até n faça
3          q ← (M - j + i - Σ_{k=i}^{j} l_k)^3
4          se q < 0
5              r[i,j] ← ∞
6          senão r[i,j] ← q
7  para i ← n decrescendo até 1 faça
8      s[i] ← r[i,n]
9      p[i] ← n
10     para j ← n decrescendo até i faça
11         se r[i,j-1] não for ∞ faça
12             se s[i] > s[j] + r[i,j-1] faça
13                 s[i] ← s[j] + r[i,j-1]
14                 p[i] ← j
15 para i = 1 até n faça
16     imprime palavra i
17     se p[i] = i
18         pula linha
```

- análise do tempo de execução: $O(n^2)$

linha	consumo
1	$O(n)$
2 a 6	$O(n^2)$
7 a 9	$O(n)$
10 a 14	$O(n^2)$
15 a 18	$O(n)$
total	$O(10n^2 + 7n) = O(n^2)$

- requisitos de espaço: $O(n^2)$

matriz r e vetores s e p

ALGORITMO:

Inicialmente são criados:

- a matriz r para armazenar o cubo do número de espaços extras no final da linha para cada divisão de palavras no parágrafo;
- o vetor s para armazenar o custo mínimo de espaços extras no parágrafo;
- o vetor p para armazenar a ordem no qual as palavras foram encaixadas no parágrafo.

Nas linhas 1 a 6, o algoritmo o algoritmo percorre a matriz e calcula o cubo do número de espaços extras: $(M - j + i - \sum_{k=i}^{j} l[k])^3$, caso seja negativo (as palavras não cabem na linha) o valor infinito é armazenado.

Nas linhas 7 a 14, o algoritmo percorre a matriz da direita para a esquerda armazenando o menor valor possível de espaços extras e guarda onde é feita a divisão de palavras entre as linhas.

Na linha 15 a 18, o algoritmo usa o vetor p para imprimir a ordem ótima das palavras no parágrafo.

3. Adapte o algoritmo dado em aula para que calcule uma matriz s que pode ser usada para determinar uma ordem ótima de multiplicação das matrizes. Dê um algoritmo recursivo que recebe as matrizes $A_1, \ldots, A_n$ em uma matriz tridimensional A, recebe a matriz s e índices i e j, com $i \leq j$, e faz o produto das matrizes $A_i \cdots A_j$ usando a ordem ótima dada em s.

```
MATRIX-CHAIN-ORDER(p, n)
1  para i ← 1 até n faça
2      m[i,i] ← 0
3  para l ← 2 até n faça
4      para i ← 1 até n-l+1 faça
5          j ← i+l-1
6          m[i,j] ← ∞
7          para k ← i até j-1 faça
8              q ← m[i,k] + p[i-1] p[k] p[j] + m[k+1, j]
9              se q < m[i,j]
10                 então m[i,j] ← q
11                 s[i,j] ← k
12 retorna s

MULT-ORDEM-OTIMA(A, s, i, j, c)
1  se i = j faça
2      matriz ← A[c]
3      c ← c+1
4      retorna matriz
5  m1 ← MULT-ORDEM-OTIMA(A, s, i, s[i][j], c)
6  m2 ← MULT-ORDEM-OTIMA(A, s, s[i][j], j, c)
7  produto ← MULT-MATRIZ(m1, m2)  // multiplica as matrizes m1 e m2
8  retorna produto

MAIN()
1  s ← MATRIX-CHAIN-ORDER(p, n)  // p é um vetor p[0... n]
2  c ← 1  // contador para a matriz ser multiplicada
3  produto-ordem-otima ← MULT-ORDEM-OTIMA(A, s, 1, n-1, c)
```

- A função MATRIX-CHAIN-ORDER(p, n) foi adaptada na linha 11 para armazenar os pontos (i, j) que mostram onde dividir as matrizes para que fiquem na ordem ótima de multiplicação.
- O algoritmo MULTI-ORDEM-OTIMA(A, s, i, j, c) recursivamente multiplica as matrizes de acordo com com os pontos de divisão em s[i, j].