Longest common subsequence

PAGE No.
DATE / /

```
// Algorithm to find longest common subsequences of grades among 20 students

// Input: grades [20][8] where each inner list represent seq of grade for each student

// output: LCS among all 20 students

LCS (grades):
common_LCS = grades[0]

for i=1 to lg do
    common_lcs = LCS (common_LCS, grades)

if (common_LCS == 0)
    break;

return common_LCS:

procedure _LCS (x, y):
    m = x.length()
    n = y.length()

Let c[0,... m]... n] be table to
                    store LCS

for (i=0 to m)
    c[i][0] = 0;
for (j=0 to n)
    c[0][j] = 0;
```

```
for i=1 to m :
    for j=1 to n :
      if ( x[i-1] == y[j-1])
          c[i][j] = c[i][j] + 1
      else
          c[i][j] = max (c[i-1][j], c[i][j-1])

lcs_res = [ ]
i = m
j = n

while (i>0 and j>0)
   if (x[i-1] == Y[j-1])
      append  x[i-1] to LCS_res ;
         i--;
         j--;

   else if  c[i-1][j] > c[j][j-1] :
            i--;

   else  j--;

reverse_lcs.res ;
return_lcs_res ;
```

- Testcase c:

Testcase 1, 2, 5 ⇒ Normal positive test case

Testcase 3 ⇒ Invalid grades

Testcase 4 ⇒ Seq. length not 30 characters

Testcase 6 ⇒ missing data fields

Testcase 7 ⇒ empty csv file no data to process

- Time complexity:

a) Brute method:
Brute method will yield exponential time complexity. due to fact that we are generating all combinations.

Thus, time complexity = $O(2^n)$
where n = length of string.

b) DP approach:
we have 2 nested loops resulting in

$$\sum_{i=1}^{m+1} \sum_{j=1}^{n+1} (1) = \sum_{i=1}^{m+1} (n+1) - 1$$

$$= \sum_{i=1}^{m+1} n$$

$$= n \sum_{i=1}^{m+1} (1)$$

$$= n(m+1-1)$$

$$= n * m$$

∴ Time complexity $= O(n * m)$

Matrix chain multiplication

PAGE No.
DATE / /

Algorithm :

I) Brute force

// list of matrix dimensions where dimensions [i] = // (rows, cols)
// output : min cost of multiplying matrices

1> main algorithm

```
function matrixchainbrute (dim)
    if (dim.empty()):
        print ("No matrices available")
        return
let n = dim.length() // no.of matrices
min cost = bruteforcecost (0, n-1)
print (mincost) // printing min cost
```

2> helper function // input indice i,j representing 2 matrices

// outputs : minimum cost

```
function bruteforcecost (i, j, dim):
    if i == j : return 0
    let mincost = ∞
for each k from i to j-1:
```

```
let cost = bruteforcecost(i,k) +
           bruteforcecost(k+1, j) +
           (dim[i] * rows)(dim[k].cols) *
                dim[j].cols.

if cost < mincost
    mincost = cost
return mincost;
```

II) optimal DP Algo:

```
// Input : list of matrix dimensions
           where each dim[i] = (rows, cols)
// output : minimum cost
```

① Main algorithm :

```
function matrixchain(dim):
    if (dim.empty()): print "no matrices available"

    let n = dim.length()
    memo[n][n] = {-1}    // min mult cost
    seq[n][n] = {-1}     // all entries filled with -1
                         // split points for optimal order
mincost = calculate
          cost(0, n-1, memo, seq)
print(mincost)
printoptimalseq(0, n-1, seq)
```

② Helper Functions

A) calculate cost (i, j, memo, seq)

// Inputs : DP ~~tables~~ & indices i, j
// outputs : minimum cost for multiplication

```
function calculatecost (i, j, memo, seq):
    if i == j : return 0
    if memo [i][j] != -1
        return memo [i] [j]

    let mincost = ∞
    for each k from i to j-1 :
    let cost = calculate cost (i, k) +
               calculate cost (k+1, j) +
               dim [i].rows * dim[k].cols
                    * dim [j]. cols
    mincost = min (mincost, cost)
        seq [i][j] = k // store split point
    memo [i][j] = mincost
    return mincost
```

B) PRINT-SEQ : printoptimalseq (i, j, seq)

// Inputs : seq [i][j] // denotes split points encountered
// Indices i, j
// output : prints optimal seq of mult

```
function printoptimalseq (i, j, seq):
    if i == j : print ('c') + (i+1) return
    print ('(')
```

printoptimalseq (i, seq[i][j])
printoptimalseq (seq[i][j]+1, j]
print (')')

- Testcases :

positive testcases :

1) $A_1[6,7] \times A_2[7,5] \times A_3[5,4]$

2) $A_1[6,8] \times A_2[8,5] \times A_3[5,4] \times A_4[4,6]$

3) $A_1[6,8] \times A_2[8,5] \times A_3[5,4] \times A_4[4,6] \times A_6[6,3]$

4) $A_1[6,8] \times A_2[8,5] \times A_3[5,4] \times A_4[4,6] \times A_5[6,3] \times A_6[3,5]$

5) $A_1[6,8] \times A_2[8,5] \times A_3[5,4] \times A_4[4,6] \times A_5[6,3] \times A_6[3,5] \times A_7[5,7] \times A_8[7,2]$

Negative testcases :

6) No matrices available

7) Incompatible matrices
$A_1[6,7] \times A_2[5,4]$

8) missing data
$A_1[6,7] \times A_2[7,0] \times A_3[0,4]$

} 0 cost

9) not exactly negative but just no optimizations possible
$A_1[4,4] \times A_2[4,4] \times A_3[4,4]$

Time complexity:

By using Dynamic programming

If we have matrices A_i from $i=1$ to n where each pair $A_i \times A_{i+1}$ has dim $p_i \times p_{i+1}$

Subproblem $m[i][j]$ → minimum number of scalar mult.

$A_i \times A_{i+1} \times A_j$

we split product around k $\quad i \le k < j$

computing cost for 2 subproblems

$m[i][k]$ ÷ minimum cost to multiply $A_i \times A_{i+1} \cdots \times A_k$

$m[k+1][j]$ = minimum cost $A_{k+1} \times A_{k+1} \times \cdots A_j$

from this we form recurrence rel^n

$$m[i][j] = \min \left(m[i][k] + m[k+1][j] + \underbrace{p_{i-1} \times p_k \times p_j}_{\text{cost to multiply 2 resulting matrices from split}} \right)$$

when $i=j$, we have Base case $m[i][i]=0$

For each pair (i,j) $i<j$
we compute $m[i][j]$ Because of
2 nested loops we have

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n-1} O(1)$$

$$\sum_{i=1}^{n-1} (n-i) \Rightarrow (n-1)+(n-2)+\ldots$$

// sum of first $n-1$ integers

we have sum $= n(n+1)/2$

$$(n-1)(n-1+1)/2$$

$$\Rightarrow (n-1)\,n/2$$

$$\Rightarrow \frac{n^2}{2} - \frac{n}{2}$$

ignoring constant & lower order term

$$\simeq O(n^2)$$

now, for each such computed $m[i][j]$
we break problem about $i \le k < j$
this will need us to iterate over k
that is for each k value which will
range from $i \le k < j$. $0 \le k < j+i$

ultimately

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=0}^{j-1} O(1)$$

↳ $O(n)$ operations
$*$ $O(n^2)$ operation
$= O(n^3)$ operation

↳ $1+2+3\ldots n-1$

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

approximating cubically.

$$\frac{1}{2} \sum_{k=1}^{n-1} k(k+1)$$

$$\frac{1}{2} \sum_{k=1}^{n-1} k^2 + \sum_{k=1}^{n-1} k$$

$$\Rightarrow \frac{1}{2} \left((n-1)n(2n-1) + \frac{(n-1)n}{2} \right)$$

$$\Rightarrow \frac{n}{2}(n-1) \times \frac{2n+2}{3}$$

$$\Rightarrow \frac{(n-1)n(2n+2)}{6} \simeq O(n^3)$$

∴ Time complexity : $O(n^3)$