Name: Sindhuja Yerramalla
UUID: U00839259.
email : Syrrmlla@memphis.edu

HomeWork-4, Algorithms and Problem Solving

Note :- I have referred textbook, lecture notes and so many random websites like chegg & stack overflow and also worked with few of my classmates.

Problem1 : Problem 6.10 from DPV

Let P[i][j] be the probability of obtaining j heads in i coins

Basecase for recursion formulation is P[0][0] = 1
(probability of getting 0 heads in 0 coins)

Recursion can be given as $P[i][j] = P[i-1][j](1-P[i]) + P[i-1][j-1]P[i]$

where probabilities of n coins be in P[1, 2, ----- n]

This represents the probabilities of both getting head in i^{th} coin and reducing the j and not obtaining head & not reducing the j

Algorithm:

```
let P be a n x (k+1) array of zeros
P[0][0] = 1
for i from 1 to n:
    P[i][0] = P[i-1][0](1-P[i])
    for j from 1 to min(i,k):
        P[i][j] = P[i-1][j](1-P[i]) + P[i-1][j-1]P[i]
return P[n][k]
```

And the time complexity for the above algorithm is $O(n^2)$ //

Problem 2 : Problem 6.6 from DPV

This can be solved by DP. Let dP[i][j] be true if it is possible to parathesis the substring from index i to index j (inclusive) in such a way that the value of the resulting expression is "a".

Base Case by recursion : dp[i][i] = 1 if it's a at i else 0

Recursion : dp[i][j] depends on the values of dp[i][j-1], dp[i+1][j] and dp[i+1][j-1]

The values of dp for substrings of length 2 and higher are then filled in by the function using a nested loop. It considers all possible ways to divide each substring of length k into two non-empty substrings for each substring in order to find a way to parenthesize the two substrings so that their product evaluates to "a", and then it uses the multiplication table to combine the two products to create a product that evaluates to "a" for the entire substring. If a solution is discovered, the function sets dp[i][i+k-1] to 1, indicating that the substring can be parenthesized in order for it to evaluate to "a". Time complexity will be $O(n^3)$

Algorithm :

```
function can_parenthesize(s)
    n = length(s)
    dp = array of size n x n, initialized to 0
    for i from 0 to n-1 do:
        if s[i] == "a" then
            dp[i][i] = 1
    for k from 2 to n do
        for i from 0 to n-k do
            j = i+k-1
            for p from i to j-1 do
                if dp[i][p] == 1 and dp[p+1][j] == 1 and M[s[p], s[j]] == "a" then
                    dp[i][j] = 1
    return dp[0][n-1]
```

Here M is the multiplication table given in the question.

Problem 3 : problem 6.12 from DPV.

Let us assume a DP array dp[n][n] where n is the number of vertices in the polygon.

We assume the subproblem as A(i,j) where i,j represent the vertex numbers in the given polygon and finds out the cost of triangulation in the sub polygon formed by those vertices.

Base Case : dp[i][i+1] = 0 for all i

Recursion Case : dp[i][j] = min(dp[i][j], distance(P[i], P[j]) + dp[i][k] + dp[k][j])

where distance gives the distance between the vertices and k ranges from i+1 to j-1 . so this gives the distance based on the cost of the previously calculated vertices and the newly calculated cost

Algorithm:

n = length(P)

dp = array of size n×n, initalized to infinity

Base Case : for i from 1 to n-2 do

dp[i][i+1] = 0

Algorithm

```
for i from 1 to n-2:
    dp[i][i+1] = 0
for len from 3 to n:
    for i from 1 to n-len+1:
        j = i + len - 1
        for k from i+1 to j-1:
            cost = distance(P[i], P[j]) + dp[i][k] + dp[k][j]
            dp[i][j] = min(dp[i][j], cost)
return dp[1][n]
```

Problem 4: Problem 6.17 from DPV.

Let us assume a DP array A[v][n] where v is the change requested and n is the denomination of the coins i.e $x_1, x_2, \ldots x_n$

Base Case: A[0][j] = true because we can make 0 change with any coins given

Recursion:

A[i][j] = A[i][j-1] or A[i-x_j][j] (if i >= x_j)

A[i][j] = A[i][j-1] (if i < x_j)

Algorithm

```
Function make_change (x[1, ... n], v):
  A = new boolean [v+1][n+1]
  for j=0 to n:
    A[0][j] = true
  for i=1 to v:
    for j=1 to n:
      A[i][j] = A[i][j-1]
      if i >= x[j]:
        A[i][j] = A[i][j] or A[i-x[j]][j]
  return A[v][n]
```

After base case, if we can generate the change with i coins then we can do the same change with i+1 coins [induction theory]. Else we take other possibilities by considering change for i-x[j] coins

Time complexity is $O(n^2)$.