HOMEWORK-7

SRI MANVITH VADDEBOYINA
1231409457

~~a) For every buy and sell ~~there is~~ a transaction fee is paid only once either during a buy (or) during the selling. We can only sell after a stock is bought.~~

1. a) The subproblem to be solved is

$OPT(i)$ = Maximum sum of subarray including i^{th} element from array 1 to i

b) $OPT(i) = \text{Math.max}(OPT(i-1) + a(i), a(i))$

- $OPT(i-1) + a(i)$ → Sum of subarray till $(i-1)^{th}$ index + current value of array
- $a(i)$ → Current element is more than sum of subarray till $(i-1)^{th}$ index.

c) Algorithm:

Initialise dp array of size N.

$dp[0] = a[0]$ → Base case, when only 1 element is present then max subarray sum is element itself.

$dp[i] = -\infty$ $\forall$ $i = 1$ to $N-1$

① for $i = 1$ to $N-1$ → (0 indexed)
  $dp[i] = \max(dp[i-1] + a[i], a[i])$

② Find maximum value of dp array, this gives the maximum subarray sum possible.

d) (i) Base cases:

$dp[0] = a[0]$ → when only 1 element is present then subarray sum is element itself.

$dp[i] = -\infty$
$\forall\ i = 1$ to $N-1$

↳ Initiating all other values to $-\infty$. Since computing maximum.

(ii) Final answer:

Maximum value of dp array will be the final answer.

e) <u>Complexity</u>: $O(N) + O(N)$ → find max and

↑ Computing dp array

$T.C = \underline{O(N)}$

2. Given n days, prices list of stock.

a) OPT (i) → Maximum profit obtained till i-th day performing any no. of transactions

We will use another variable to determine corresponding state of transaction.

state → 0 ⇒ implying no holding of stock
→ 1 ⇒ implying holding of stock

OPT(i, state) → is the amount of cash holding till i th day performing transaction.

whenever we buy a stock, we buy a stock, we reduce cash by "prices[i] + fee" and change state to 1 whenever we sell a stock, we increase cash by prices[i] and change state to 0.

If we do not do any transaction then we will leave state as it is.

b) OPT(i, state) = $\max_{state=0} \begin{cases} \underbrace{-prices[i] - fee + OPT(i+1, 1)}_{\rightarrow BUY}, \\ OPT(i+1, 0) \rightarrow \text{NO CHANGE} \end{cases}$

$\max_{state=1} \begin{cases} \underbrace{prices[i] + OPT(i+1, 0)}_{\rightarrow SELL} \\ OPT(i+1, 1) \rightarrow \text{NO CHANGE} \end{cases}$

c) Recursive solution [top down] using memoization
Initialise a map < string, Integer>

Cal (index, state)
  key = index + '#' + state
  if (map. Contains (key))
      return map. get (key)

  if (index > N) return 0
  res = - ∞
  if state = 0
      res = max (- prices [index] - fee
                + Cal (index +1, 1),
            cal (index +1, 0))

  if state = 1 :
      res = max (prices [index] +
                Cal [index +1, 0)),
            Cal (index +1, 1))
    map. put (key, res)

  return res;

Answer = Cal (1, 0) → index = 1 start
                      State = 0 → not holding initially.

d) Base Case : if no stocks i.e N = 0
   Answer = 0   No stocks to perform transaction

e) Complexity = O (N × 2) = O (N)
               ↑ state = 0 or 1

3. Given array 'a' of size N.

a) OPT(i) = Maximum happiness we can get for the array 1 to i.

For each index we will store multiplier (M) that is the multiplier used for the i^{th} value to get maximum happiness.

Initially all multipliers will be ~~[illegible]~~ initialised to 1.

$$M[i] = 1 \quad \forall i = 1 \text{ to } N$$

b) Recurrence Relation:

OPT(i) = Compute max value of all $\forall_{j=1}^{i}$

if $a(j) < a(i)$ Compute $OPT(j) + \underbrace{(M[j]+1)}_{\text{next multiplier for } a[i]}\ \underbrace{a[i]}_{\text{current value}}$

M[i] will be stored based on the maximum value of happiness attained.

Suppose max value is obtained by considering OPT(j) then $M[i] = M[j]+1$

c) Algorithm:

Initialise dp array of size N ;

Initialise multipliers to 1 $M[i]=1 \ \forall i = 1 \text{ to } N$

$dp[1] = a[1]$ Base case.

for i=2 to N
  for j=1 to i
    if $(a[j] < a[i])$
      $dp[i] = \max(dp[i], dp[j] + (M[j]+1)a[i])$
    if $(dp[i] < dp[j] + (M[j]+1)a[i])$
      multiplier = $M[j]+1$;
  endfor
  $M[i]$ = multiplier
end for

Answer = $dp[N]$.

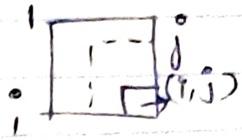
d) (i) Base Case: $dp[1] = a[1]$ → array of size 1
$M[i] = 1 \quad \forall i = 1 \text{ to } N$

(ii) Final answer = $dp[N]$ → last index value.

e) Time Complexity:

$$\underset{1^{st}\text{ loop}}{1} + \underset{2\text{ loop}}{2} + 3 + \cdots + \underset{n^{th}\text{ loop}}{n-1} = \frac{n(n-1)}{2} = O(n^2)$$

4 a) $OPT(i,j)$ = Maximum possible square size including $g(i)(j)$ in given binary matrix 'g'. only considering submatrix from $(1,i)$ to $(1,j)$



b) $OPT(i,j) = \min$ of $(OPT(i-1,j), OPT(i,j-1), OPT(i-1,j-1)) + 1$

max square possible including (i,j) is the minimum possible square till $(i-1,j)$ $(i,j-1)$ $(i-1,j-1)$ $+1$

c) <u>Algorithm:</u>

Initialise dp array of size same as binary matrix

Initialise dp array first row and first column with corresponding binary matrix values.

[m → rows , n → columns]

FOR $i=1$ to m $dp[0][i] = g[0][i]$
FOR $j=1$ to n $dp[j][0] = g[j][0]$

Initialising all values $\forall_{i=1 \text{ to } m}$ $\forall_{j=1 \text{ to } n}$ $dp[i][j] = g[i][j]$

for $i=2$ to m
    for $j=2$ to n

if $dp[i][j] = 1$

$$dp[i][j] = \min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]) + 1$$

Maximum length of square possible will be maximum value in dp array.

d) Base case: Initialising all values of binary matrix to dp array

$dp[i][j] = 1$ if $g[i][j] = 1$ ⇒ Square matrix of size 1×1

Final answer: Maximum value in the entire dp array is the answer.

e) Complexity = $O(NM) + O(NM) + O(NM)$

⇒ Initiating dp array | updating dp array using recurrence relation | finding max in dparray.

T.C = $\underline{\underline{O(NM)}}$