Assignment 4 Hongdao Meng hm3424

Q1.a

① Define Table: $T[i,j]$: the minimum total cost to travel from $(1,1)$

② initialise: $T[1,1] = cost(1,1)$, stay there and simply pay toll at $(1,1)$

For first row $T[1,j] = T[1,j-1] + cost(1,j)$, only move right along the top row.

For first column: $T[i,1] = T[i-1,1] + cost(i,1)$, only move down along first column.

③ relationship: If we're allowed to move only right or down,

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

$(i>1, j>1)$

if diagonal moves are allowed:

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

④ final result: $T[n,n]$ (from $(1,1)$ to (n,n))

⑤ Pseudo-code:

```
Initialise 2D Table T of size nxn
T[1,1] = cost(1,1)
for j=2 to n:
    T[1,j] = T[1,j-1] + cost(1,j)
for i=2 to n:
    T[i,1] = T[i-1,1] + cost(i,1)
for i=2 to n:
    for j=2 to n:
        T[i,j] = cost(i,j) + min(T[i-1,j], T[i,j-1])
return T[n,n]
```

⑥ Runtime: ∵ each $T[i,j]$ takes $O(1)$

∴ $O(n^2)$

Q1b.

```
PrintPath(T, cost, n):
    i = n
    j = n
    path = [ ]
    while (i, j) != (1, 1):
        path.push_front((i, j))
        if T[i, j] == T[i-1, j] + cost(i, j):      # move to whichever predecessor yields T[i,j]
            i = i-1
        else:
            j = j-1
    path.push_front((1, 1))
    print(path).
```

Q2a.

DP solution:

① Sort the rocks by finishing mile $f(i) = i + D[i]$. Start $s_i = i$

intervals $(s_1, f_1), \cdots (s_{n'}, f_{n'})$

② Define $DP[k]$ = the maximum number of rocks that can be transport if we only consider the first k rocks in sorted order.

③ Initialization

$DP[0] = 0$ (with no intervals, no rocks are transported)

④ Relationship Between Entries

define $p(k) = \max(j : 1 \le j < k \text{ and } f_j < s_j)$

$P(k)$ gives the index of the rightmost interval that doesn't overlap with interval k.

$$DP[k] = \max(DP[k-1], 1 + DP[p(k)])$$

⑤ Final result: $DP[n']$

⑥ Pseudocode (D, n)

```
intervals=[ ]
for i=1 to n:
  finish= i+D[i]
  if finish ≤ n : intervals.append((start:i, finish:finish))
n'= intervals.size( )
sort. intervals by finish
P[1... n']
for k=1 to n':
    P[k] =0
    for j=k-1 downto 1:
        if intervals[j].finish ≤ intervals[k].start :
           P[k]=j
           break
DP[0]=0
for k=1 to n':
    DP[k] = max (DP[k-1], 1+DP[P[k]] )
return DP[n']
```

Build interval $O(n)$

Sort $O(n \log n)$

Fill DP: $O(n^2)$

⑦ Runtime:

$O(n \log n)$

Q2b. if DP[k]=DP[k-1] , interval k wasn't used
elif DP[k] = 1+ DP [p(k)] , k was chosen.

Since we backtrack from the last interval to the first we insert the chosen mile markers at the front of our solution list to obtain them in forward order.

```
Print (DP , P , intervals , n'):
    k = n'
    solution = [ ]
    while k>0 :
        if DP[k] == DP[k-1] :
            k = k-1
        else :
            solution.push_front( intervals[k] .start )
            k = P[k]
    print(solution)      # The list of mile markers where rock
                         are picked up
```

Q3

① DP Table definition:

Let $T(i,j)$ denote the maximum total arc-value that can be obtained from the subarray $C[i..j]$. If no arc can be drawn, $T(i,j)=0$

② initialization : $T(i,i)=0$ for all i, single character can't from arc.

For $i>j$, $T(i,j)=0$.

③ Relationship and filling the table.

for ~~i≤j~~ $i<j$:

(3.1) skip $C[i]$: don't use char i in any arc, in this case:

$$T(i,j)=T(i+1,j)$$

(3.2) Pair $C[i]$ with $C[k]$ ($i<k\le j$ and $C[k]==C[i]$)

The arcs drawn must be non-crossing, so subproblem divide into:

the subarray between $i+1$ and $k-1$: $T(i+1,k-1)$

the subarray from $k+1$ to j : $T(k+1,j)$

with a fixed k, we have:

$$\text{Candidate Value} = T(i+1,k-1)+val(C[i])+T(k+1,j)$$

Combining these two case, we get: take the maximum over all such k with $C[k]=C[i]$.

$$T(i,j)=\max\left(T(i+1,j),\ \max_{\substack{k\in[i+1,j]\\ C[k]=C[i]}}\left\{T(i+1,k-1)+val(C[i])+T(k+1,j)\right\}\right)$$

④ Final result: $T(1,n)$

⑤ Pseudocode

```
Maxvalue(C, n):
    for i=1 to n:
        T[i][i] = 0

    for length=2 to n:
        for i=1 to n-length+1:
            j=i+length-1
            bestValue = T[i+1][j]    # skip C[i]

            for k=i+1 to j:    # pair C[i] with C[k]
                if C[k] == C[i]:
                    candidate = T[i+1][k-1] + value(C[i]) + T[k+1][j]
                    if candidate > bestValue:
                        bestValue = candidate

            T[i][j] = bestValue

    return T[1][n]
```

func: value(c) will return the arc value for color c.

⑥ Runtime:

DP Table Size: $O(n^2)$ entries

Transition Cost: For each $T(i,j)$, we scan k from $i+1$ to j, which take $O(n)$ time in the worst case.

Overall,

total time is $O(n^3)$

Q4a.

① sort the boxes

sort the boxes in non-~~de~~ increasing order by height.

② Define DP Table:

DP[h] be the minimum total weight needed to build a tower of height exactly h. We only need to maintain states of 0, 1, 2, ..., M

DP[0] = 0

For all h from 1 to M, DP[h] = float('inf')

③ Relationship and Recurrence:

For each box i, iterate over current heights from M to 0

- If tower of height h is achievable, then by placing box i on top we obtain a new height: newh = min(T, h + H[i])
- update the DP value: DP[newh] = min(DP[newh], DP[h] + W[i])

④ final result: DP[M]

⑤ Pseudocode:

```
MinTowerWeight(H, W, M, n):
    sortedBoxes = sort((H[i], W[i], i) for i=1 to n)
    for h = 0 to M:
        DP[h] = float('inf')
    DP[0] = 0
    for h=0 to M:
        parent[h] = (null, null)
    for each box in sortedBoxes:
        boxH = box.H      boxW = box.W    boxIndex = box.index
        for h = T down to 0:
            if DP[h] != 'inf':   newH = min(T, h + boxH)
                if DP[h] + boxW < DP[newH]:  DP[newH] = DP[h] + boxW
                                             parent[newH] = (h, boxIndex)
    return DP[M], parent
```

⑥ Runtime:

Sort: $O(n\log n)$

DP update: $O(nT)$

∴ Total $O(n\log n + nT)$

(b) PrintTowerSolution (parent, M) # backtrack M to 0 using parent pointer.

```
state = M
solution = [ ]
while state != 0 :
    (preState, boxIndex) = parent[state]
    solution.push(boxIndex)
    state = preState
reverse (solution)   # reverse to get bottom to top.
print (solution)
```

Q5

① DP Table Definition

$T[i,j]$ for $i \geq 1$, $j \leq n$ $T[i,j]$ is the maximum chance that the message , starting from some input computer in column 1, safely reaches computer (i,j).

② initialization

In the first column : $T[i,1] = 1 - B(i,1)$

For any other cells , if there is no valid path to (i,j), then $T[i,j]$ remains 0 .

③ Filling the DP Table

$$T[i,j] = (1 - B(i,j)) \times \max(T[p,q] : (p,q) \in Pred(i,j)).$$

The message reaches (i,j) with a success probability equal to the product of the success probability of computer (i,j) and the best success probability among all its direct predecessors .

④ final answer : $T[n,n]$

if $T[n,n]$ is 0 , then no safe route is possible.

⑤ Pseudocode:

```
MaxTransSuccess (B , n) :
  for i = 1 to n :
     T[i,1] = 1 - B[i,1]
  for j = 2 to n :
     for i = 1 to n :
        T[i,j] = 0
  for j = 2 to n :
     for i = 1 to n :
       max Predecessor = 0
       foreach (p,q) in Pred(i,j),
          if T[p,q] > maxPredecessor:
             max Predecessor = T[p,q]
       T[i,j] = (1 - B[i,j]) * maxPredecessor

  Return T[n,n]
```

⑥ Runtime :

DP Table has $n \times n$ entries.

For each entry (i, j), we consider a constant number of cells.

computation $O(1)$.

$\therefore$ Overall time is $O(n^2)$