Q1

Let there be n symbols, and $n-1$ operations between them. We solve the following two subproblems: How many ways are there to make the expression staring from at the lth symbol and ending at rth symbol evaluate to $true$(T), and how many ways to do the same but evaluate to $false$(F). Assume example like "$true$ OR $false$ AND $true$ NAND $false$ NOR $true$", $T(1,2)$ would be the number of ways of making "$true$ and $false$" evaluate to true with correct bracketting and in this case, $T(1,2) = 1$. Otherwise, $T(1,2) = 0$.

The base case is that $T(i,i)$ is 1 if symbol i is $true$, and 0 if symbol i is $false$. The reverse applies to $F(i,i)$.

Otherwise, for each subproblem, we 'split' the expression around an operator m so that everything to the left of the operator is in its own bracket, and everything to the right of the operator is in its own bracket to form two smaller expressions. For example, splitting the sample expression around "AND" would give "($true$ OR $false$) AND ($true$ NAND $false$ NOR $true$)". We then evaluate the subproblems on each of the two sides, and combine the results together depending on the type of operator we are splitting by, and whether we want the result to evaluate true or false. We solve both subproblems in parallel:

$$T(l,r) = \sum_{m-l}^{r-1} Tsplit(l,m,r)$$

$$F(l,r) = \sum_{m-l}^{r-1} Fsplit(l,m,r)$$

$$TSplit(l,m,r) = \begin{cases} T(l,m) \times T(m+1,r) \\ \quad \text{if operator m is 'AND'} \\ T(l,m) \times F(m+1,r) + T(l,m) \times T(m+1,r) + F(l,m) \times T(m+1,r) \\ \quad \text{if operator m is 'OR'} \\ T(l,m) \times F(m+1,r) + F(l,m) \times T(m+1,r) + F(l,m) \times M(m+1,r) \\ \quad \text{if operator m is 'NAND'} \\ F(l,m) \times F(m+1,r) \\ \quad \text{if operator m is 'NOR'} \end{cases}$$

$$FSplit(l,m,r) = \begin{cases} T(l,m) \times F(m+1,r) + F(l,m) \times T(m+1,r) + F(l,m) \times F(m+1,r) \\ \quad \text{if operator m is 'AND'} \\ F(l,m) \times F(m+1,r) \\ \quad \text{if operator m is 'OR'} \\ T(l,m) \times T(m+1,r) \\ \quad \text{if operator m is 'NAND'} \\ T(l,m) \times T(m+1,r) + T(l,m) \times F(m+1,r) + F(l,m) \times T(m+1,r) \\ \quad \text{if operator m is 'NOR'} \end{cases}$$

Note that the equations inside the $TSplit$ and $FSplit$ functions are chosen to correspond with the truth tables of the corresponding operator.

The final solution is given by $T(1,n)$.

The complexity is $O(n^3)$. There are n^2 different ranges that l and r could cover, and each needs the evaluations of $TSplit$ or $FSplit$ at up to n different splitting points.

Q2

Setup

First, denote the 2D map called $Martix$, and $Martix[c][r](1 \leq c \leq C$ and $1 \leq r \leq R)$ representing the elevation of the terrain at that square.

Subproblem

For every $r \in [1, C]$ and $c \in [1, C]$, we define the subproblem $P[c][r]$ to be "find the number of moves from lower elevation to higher elevation from $Martix[1][R]$ to $Martix[c][r]$ ".

Let $Opt[c][r]$ be the total moves from lower elevation to higher elevation along the path from $Martix[1][R]$ to $Martix[c][r]$, which is the optimal solution to the subproblem $P[c][r]$.

Build-up order

As the question mentioned we can only move to the strictly right square or strictly down square, so for the first column and first row, we should solve these subproblems first and then do recursion.

For the first column $Martix[1][r]$ (r from $R-1$ to 1), can be only derived by the previous square on the left. And same to the first row $Opt[c][R]$ (c from 2 to C), can be only derived by the previous square on the up.

So solve the subproblems in the order $P[1][R], P[1][R-1], \ldots, P[1][1], P[2][R]P[3][R] \ldots P[C][R]$ and then $P[2][R-1], \ldots, P[2][1], P[3][R-1], \ldots, P[3][1], \ldots, P[C][R-1], \ldots, P[C][1]$. From left to right, up to bottom.

Base case

Because the start square is $Martix[1][R]$, so we set $Opt[1][R] = 0$. So for the first column $Opt[1][r]$ (r from $R-1$ to 1), can be only derived by the previous square on the left, so

$$Opt[1][r] = \begin{cases} Opt[1][r+1] + 0 & Martix[1][r+1] \geq Martix[1][r] \\ Opt[1][r+1] + 1 & Martix[1][r+1] < Martix[1][r] \end{cases}$$

The objective is to find the minimum moves from lower elevation to higher elevation, so if $Martix[1][r+1] > Martix[1][r]$, which means from high elevation to lower elevation, so $Opt[1][r]$ do not need to add one, otherwise $Opt[1][r]$ need to add one. And same to the first row $Opt[c][R]$ (c from 2 to C),

$$Opt[c][R] = \begin{cases} Opt[c-1][R] + 0 & Martix[c-1][R] \geq Martix[c][R] \\ Opt[c-1][R] + 1 & Martix[c-1][R] < Martix[c][R] \end{cases}$$

Recursion

Now we do recursion, in order to make it clear, we denote

$$E(p1, p2) = \begin{cases} 0 & Martix(p1) \geq Martix(p2) \\ 1 & Martix(p1) < Martix(p2) \end{cases}$$

$p1$ and $p2$ are two points, and format is $p = [c, r]$.

For r from $R-1$ to 1, and c from 2 to C, we have

$$Opt[c][r] = min\{Opt[c-1][r] + E([c-1,r],[c,r]), Opt[c][r+1] + E([c,r+1],[c,r])\}$$

Final solution

The final solution is given by $Opt[c][1]$.

Time complexity

The complexity is $O(C * R)$ because we only traverse the entire matrix elements once.

Q3

Setup

Assume three activities are named 0, 1 and 2. And $E[i][0]$, $E[i][1]$ and $E[i][2]$ are stand for the enjoyment that get out of 0, 1 and 2 on ith day respectively.

Subproblem

For $1 \leq i \leq N$, we define the subproblem $P(i)$ to be "find the maximum total enjoyment from the first day to ith day and the same activity are not allowed between two consecutive days".

Let $T(i)$ be the total duration of the optimal solution to the subproblem $P(i)$.

Build-up order

Solve the subproblems in the order $P(1), \ldots, P(n)$.

Base case

$$T[0][0] = E[0][0]$$

$$T[0][1] = E[0][1]$$

$$T[0][2] = E[0][2]$$

$$T[0] = max\{T[0][0], T[0][1], T[0][2]\}$$

As, we are not allowed to do same activity two days in a row, so we use $T[i][a]$ to store the case that total enjoyment ends with a activity on ith day.

Recursion

For all $i > 1$, we have

$$T[i][0] = max\{T[i-1][1] + E[i][0], T[i-1][2] + E[i][0]\}$$

$$T[i][1] = max\{T[i-1][0] + E[i][1], T[i-1][2] + E[i][1]\}$$

$$T[i][2] = max\{T[i-1][0] + E[i][2], T[i-1][1] + E[i][2]\}$$

$$T[i] = max\{T[i][0], T[i][1], T[i][2]\}$$

Final solution

The final solution is given by,

$$T[n] = max\{T[n][0], T[n][1], T[n][2]\}$$

Time complexity

Clearly such algorithm runs in time linear in the number of all N days.

Q4

Subproblem

For start vertex S and destination vertex X, $S, X \in V$, and length j, $j \in [1, k]$. We define subproblem $P(S, X, j)$ to be "find the longest length from vertex S to vertex X through j edges"

Let $dp[S][X][j]$ be the longest path(i.e. total weight is maximum) from node S to node X using exactly J edges in total, which is the optimal solution to the subproblem $P(S, X, j)$.

Build-up order

For every vertex $S \in V$, and for every vertex $X \in V$, also for length $j \in [1, k]$. We handle subproblems in the order $P(S_1, X_1, 1), \ldots, P(S_1, X_1, k), P(S_1, X_2, 1), \ldots, P(S_1, X_2, k), \ldots, P(S_n, X_n, 1), \ldots, P(S_n, X_n, k)$.

We can use nested loops to traverse all vertices as start node and destination node.

Base case

For vertex $S \in V$,

$$dp[S][S][0] = 0$$

For vertices S and X, $S, X \in V$, and length $j = 1$, if S and X are connected,

$$dp[S][X][1] = weight(S, X)$$

Otherwise we don't store $dp[S][X][j]$.

Recursion

For start vertex S, destination vertex X, $S, X \in V$ and length $j \in [2, k]$, we have,

$$dp[S][X][j] = max\{dp[S][Y][j-1] + weight(Y, X)\}$$

for all Y which has an edge from Y to X and $Y \in V$.

Final solution

The final solution of the problem is the maximum value returned by any of these subproblems,

$$max\{dp[S][X][k] : S, X \in V\}$$

Time complexity

The complexity is $O(V^2 * E * k)$ because for two vertices, start vertex and destination vertex, which means one subproblem, the algorithm runs in $O(E * k)$, and there are V^2 subproblems, so the total runs in $O(V^2 * E * k)$.