Advanced DSA — 6

Content

- Rotten Oranges
- Bob & Chocolates
- Alice & Party

Rotten Oranges *

Problem Description

Given a matrix of integers **A** of size N x M consisting of 0, 1 or 2.

Each cell can have three values:

The value 0 representing an empty cell.

The value 1 representing a fresh orange.

The value 2 representing a rotten orange.

Every minute, any fresh orange that is adjacent (Left, Right, Top, or Bottom) to a rotten orange becomes rotten. Return the minimum number of minutes that must elapse until no cell has a fresh orange. If this is impossible, return -1 instead.

Note: Your solution will run on multiple test cases. If you are using global variables, make sure to clear them.

Problem Constraints

1 <= **N**, **M** <= 1000

0 <= **A[i][j]** <= 2

```
A = [   [2, 1, 1]
        [1, 1, 0]
        [0, 1, 1]    ]
```

Input 2:

```
A = [   [2, 1, 1]
        [0, 1, 1]
        [1, 0, 1]    ]
```

Example Output

Output 1:

4

Output 2:

-1

t=0

R	F	F
F	F	
	F	F

t=1

R	R	F
R	F	
	F	F

t = 2

R R R

R R

F F

t = 3

R R R

R R

R F

t = 4

R R R

R R

R R

---

t = 0

R F F

F F

F R

t = 1

R R F

R F

R R

t = 2

R R R

R R

R R

Algo steps

→ Use queue to traverse in a BFS fashion putting all the rotten oranges as the source of BFS.

→ Do BFS and check the last level of BFS.

→ Finally check if all the oranges are rotten if not return -1.

Pseudocode

```
int rottenOranges (A[][]) {
    R // no. of rows
    C // no. of cols
    EMPTY = 0 , FRESH = 1 , ROTTEN = 2
    maxTime = 0
    queue // init
    for ( r → 0 to R-1) {
        for ( c → 0 to C-1) {
            if ( A[r][c] == ROTTEN) {
                queue.add ({ 0 , r , c })
                            time rows cols
            }
        }
    }
```

```
DIRX = {0 , 1 , 0 , -1}
DIRY = {1 , 0 , -1 , 0}
while (queue.size() >0) {
    time , r , c = queue.poll()
    maxTime = max (maxTime , time)
    for (i ⟶ 0 to 3) {
        nr = r + DIRX[i]
        nc = c + DIRY[i]
        ntime = time +1
        if ( nr <0 || nc<0 || nr >=R || nc>=C)
            continue
        if ( A[nr][nc] == FRESH) {
            A[nr][nc] = ROTTEN
            queue.add ({ntime, nr, nc})
        }
    }
}
// check no fresh orange remain
// if fresh orange exist return -1

return maxTime
}
```

TC : O(R*C)

SC : O(R*C)

Bob & Chocolates

You are in a chocolate shop that sells **N** number of different chocolates. You are given that the price of each chocolate is **B[i]** and the sweetness of each chocolate is **C[i]**.

You have decided that the total price of your purchases will be **atmost A**. You can buy each chocolate at most once. What is the *maximum* sweetness we can get using **atmost A** rupees?
Please read the examples given below carefully to better understand the problem

Problem Constraints

$1 <= N <= 10^3$

$1 <= A <= 10^5$

$1 <= B[i] <= 10^3$

$1 <= C[i] <= 10^3$

```
A = 10
B = [4, 8, 5, 3]
C = [5, 12, 8, 1]
```

Input 2:

```
A = 4
B = [4, 5, 1]
C = [1, 2, 3]
```

$$\begin{tikzcd} & dp[i-1][j] \\ dp[i][j] \arrow[ur, "dont"] \arrow[dr, "take"'] & \\ & dp[i-1][j-prices[i]] \end{tikzcd}$$

Example Output

Output 1:

```
13
```

Output 2:

```
3
```

Space optimised iterative solution.

```
                        A            B               C
int  knapsack ( amount , prices , sweetness ) {
    N = prices.length

    dp[N+1][amount +1]

    for ( i ⟶ 1 to N ) {
        p = prices[i-1]
        s = sweetness[i-1]
        for ( j ⟶ 0 to amount ) {
            dont = dp[i-1][j]
            if ( j >= p ) {
                take = s + dp[i-1][j-p]
            }
            dp[i][j] = max(take, dont)
        }
    }
    return dp[N][amount]    {~10^8 huge}
}
```

```
                          A          B              C
int   knapsack ( amount , prices , sweetness ) {
    N = prices . length

    dp [amount + 1]

    for ( i ⟶ 1 to N ) {
        p = prices [i-1]
        s = sweetness [i-1]
        prev = deep copy of dp
        for ( j ⟶ 0 to amount ) {
            dont = prev[j]
            if ( j >= p ) {
                take = s + prev [j - p]
            }

            dp [j] = max (take, dont)
        }
    }
    return dp [amount]
}
```

TC : O(N* Amount)

SC : O(amount)

Alice & Parks

Alice visits the land of amusement parks. There are a total of **A** amusement parks numbered from 1 to A. Some amusement parks are connected to each other by bidirectional bridges given by array B.

Alice hates to cross these bridges as they require a lot of effort. He is standing at amusement park 1 and wants to reach amusement park A. Find the **minimum number** of bridges that he shall have to cross, if he takes the optimal route.

Return -1 if it is not possible to reach amusement park A.

Please look at the examples below for better understanding of the problem.

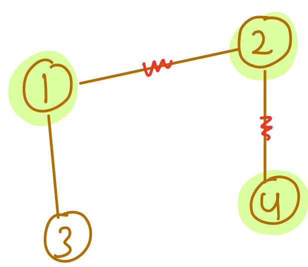
Problem Constraints

1 <= A <= 10^4

1 <= B.size() <= 10^5

```
A = 4
```

```
B = [
        [1, 2]
        [2, 4]
        [1, 3]
    ]
```

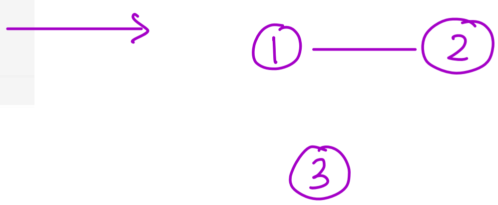


used 2 bridges to reach 4

Input 2:

```
A = 3
```

```
B = [
        [1, 2]
    ]
```



no way to reach 3 ans = -1

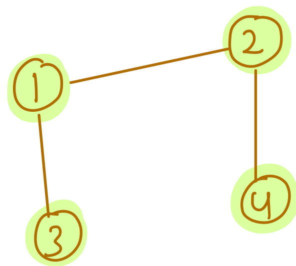
Example Output

Ouput 1:

```
2
```

Output 2:

```
-1
```



~~{0, N1}~~ ~~{1, N2}~~ ~~{1, N3}~~ {2, N4}

Pseudocode

```
int   alice&Bridges ( N,  edges[] ) {
      // Build a graph from edges
      queue   // init
      visited [N+1]   // false      visited[i] = true
      queue.add ( {0, 1} )

      while ( queue.size() >0 ) {
            bridges , node  =  queue.poll ()
            if ( node == N ) { return bridges }

            for ( nei : graph[node] ) {
                  if (! visited[nei] ) {
```

```
            visited[nei] = true
            queue.add({briges+1, nei})
         }
      }
   }
   return -1
}
```