NATIONAL UNIVERSITY OF SINGAPORE

CS2010 – DATA STRUCTURES AND ALGORITHMS II

(Semester 1: AY2016/17)

Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. Do **NOT** open the question paper until you are told to do so.
2. This assessment paper consists of **Sixteen (16)** printed pages and **Ten (10)** questions with possible subsections.

   Important tips: Pace yourself! Do **not** spend too much time on one (hard) question.
3. This is an **Open Book Assessment**. You can check the lecture notes, tutorial files, problem set files, CP3 book, or any other books that you think will be useful.
4. When this Assessment starts, **please immediately write your Student Number** below.
5. You may write your answers in pencil except student number below which should be written with a pen.
6. All the best!

STUDENT NUMBER:

A								

(Write your Student Number above legibly with a **pen**.)

Questions	Possible	Marks
Q1-3	18	
Q4-7	12	
Q8	22	
Q9	25	
Q10	23	
Total	100	

Section A – Basic (18 Marks)

For the questions in this section, V = size of vertex and E = size of the edge set.

1. List down 3 different graph problems which can be solved in O(V+E) time, the algorithm that will solve them in O(V+E) time and the kind of input graph that will give the O(V+E) time bound. Each entry is 1 mark. **[9 marks]**

Problem	Algorithm	Graph
SSSP problem	Breadth First Search	unweighted graph
Counting Components in a graph	Depth First Search	Undirected graph
SSLP problem	One pass bellman ford	DAG

Note: SSSP on tree, SSSP on unweighted graph etc are all same problem with different input graphs !

2. For each of the algorithm listed below, give one way of optimizing the algorithm. Each entry is 1 mark. **[3 marks]**

Problem	Optimization
Bellman Ford	Stop when no more edges are relaxed during a pass of the outer loop
Prim's Algorithm	Stop when all vertices have been added to the MST
Kruskal's Algorithm	Stop when there is only one disjoint set left

3. This question is based on the following 5×5 grid. **[6 marks]**

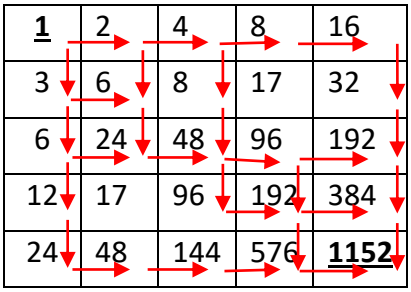
1	2	4	8	16
3	6	8	17	32
6	24	48	96	192
12	17	96	192	384
24	48	144	576	**1152**

Source is the cell (0,0) with value 1, and destination is the cell (4,4) with value 1152.

We can move from a cell with value X to a cell North/South/East/West of it with value Y if Y is divisible by X, i.e $Y \bmod X = 0$.

a) Looking at the above grid, is there a way to start from any cell in the grid, move through the grid and come back to the cell itself? **[1 marks]**

No. This is a DAG



b) What is the shortest path from the source to the destination in terms of the sum of the values in the cells along that path. **[2 marks]**

1791. The cells are (0,0), (0,1), (0,2), (0,3), (0,4), (1,4), (2,4), (3,4), (4,4).

c) What is the total number of possible paths from the source to the destination? **[3 marks]**

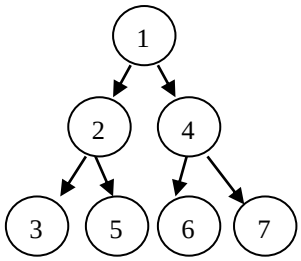
22. 4 in the submatrix (0,0) to (2,2) and 5 in submatrix (2,2) to (4,4) thus giving 20.

There is another 2 paths, one from top left to top right then to bottom right, and the other from top left to bottom left then to bottom right. Thus 22 in all.

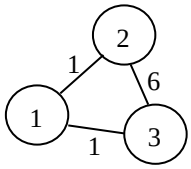
Section B – Analysis (12 Marks)

Prove (the statement is correct) or disprove (the statement is wrong) the following statements below. If you want to prove it, provide the proof or at least a convincing argument. If you want to disprove it, provide at least one counter example. 3 marks per each statement below (1 mark for saying correct/wrong, 2 marks for explanation):

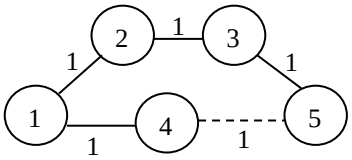
4. The k[th] smallest element in a min heap of size n is at most $O(logk)$ edges away from the root element. **[3 marks]**
   False. The kth smallest element (which can be O(lgn) smallest element) can be O(logn) edges away (and not O(lglgn) edges away). That is, it can be a leaf of the min heap. For example the 3rd smallest element in the heap below is a leaf vertex.



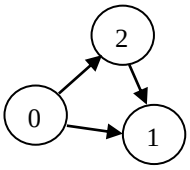
5. In an undirected connected graph, if 2 edges in a cycle have the same smallest edge weight, then either one but not both of them will be included in a valid MST of the graph. **[3 marks]**
   False. Both edges could be retained as shown in the graph below.



6. Any MST of a constant edge weight and undirected graph G with node k picked as the root is also the SSSP spanning tree on G with k as source vertex. **[3 marks]**
   False. See graph below. edge (4,5) can be removed to get an MST of the graph, but this will not be the SSSP spanning tree with source vertex 1 since SP from 1 to 5 will be wrong.



7. DFS can be used to detect cycles in an undirected graph by checking if a neighbor v of the vertex u being processed has already been visited and v is not the predecessor of u. It can be used in the same way to detect cycles in a directed graph. **[3 marks]**
   False. In the directed graph below, if DFS start from vertex 0, then vertex 1 will be visited twice and DFS will then falsely report that there is a cycle when there aren't any.



Section C – Application (70 Marks)

Partial marks will be awarded for correct answers which do not meet the time complexity required (or if the required time complexity of not given, then any correct answer that is less efficient than the lecturer's answer) . You can write in pseudo-code. Provide enough details to all user defined DSes/algorithms/modifications to taught DSes and algorithms so as to show understanding of the solution. *Sub-questions marked with * in the marks bracket can potentially be more difficult.*

8. **Graphs, graphs and more graphs [22 marks]**

For **all** the sub-questions below, the graphs are simple graphs. Also the graph/tree is stored in an **adjacency list** which you can assume to be implemented as Vector<Vector<IntegerPair>> in Java.

Assume an undirected edge is represented as a bidirectional edge in the adjacency list. Assume that bidirectional edges are allowable only to represent undirected edges.

a) Give the best algorithm you can think of that will return 1 if a given connected graph G is also an undirected graph and return 0 if it is a directed graph. **[7 marks]**

A graph is undirected if for an edge (a,b) b is in the neighbor list of a and a is in the neighbor list of b. Thus the edge is represented two times.

Go through adjacency list and sum up size of neighbor list for each vertex. Keep this in a variable C. → O(V) time

If C is not divisible by 2 (since each undirected edge represented 2 times) then it is an undirected graph , otherwise do the following:

Keep another variable C' initialized to 0. Go through the adjacency list again, and now go through neighbor list for each vertex. For each edge (a,b) hash it to a hashset/hashmap if (b,a) is not already in the hashset/hashmap. If (b,a) is in the hashset/hashmap simply increment C' by 1. → O(V+E)

At the end, if C = C'*2 then it is an undirected graph otherwise it is a directed graph.

Time complexity = O(V+E)

Easier solution

I realized that since I said only bidirectional edge can represent undirected edges, then for an undirected graph, a vertex v with >= 1 neighbor (since connected each vertex must have >= 1 neighbor) must have a bidirectional edge from each of those neighbors back to v. So just go to vertex 0 in adjacency list. Then simply go to neighbor list of 1st neighbor of 0 and check if 0 is in the list. If it is not, then it is directed graph else it is an undirected graph. O(V) solution.

Any solution with time complexity better or equal to O(V+E) will get full marks.

b) Give the best algorithm you can think of that will return true if a given spanning tree $T(V, E)$ of a weighted, undirected and connected graph $G(V, E')$ is also an MST of G and false otherwise. **[7 marks]**

1.) Get the total weight W of T(V,E) →O(V) since this is a tree

a. Go through adjacency list of T(V,E). For each vertex, go through neighbor list and sum up the edge weight. Accumulate this sum for all vertices into W.

2.) Run Prim's to get an MST of G. Compare cost of MST with W. If they are the same T(V,E) is an MST of G. Otherwise it is not. → O(ElogV)

Time complexity = O(ElogV)

A wrong solution is to check that the MST (generated by Prim's or Kruskal's) and T is structurally the same, that is, all the edges are the same. This is because G may have multiple MSTs.

c) Give the best algorithm you can think of that will return true if an edge (a, b) of an undirected and connected graph $G(V, E)$ is a bridge and false otherwise. A bridge of a connected graph is an edge which when removed will disconnect the graph. **[8 marks]**

1.) Run DFS on vertex 0. → O(V+E)

a. During DFS If current vertex == a or b and neighbor to be visited == b or a, do not recursively visit neighbor. Otherwise proceed as per normal.

b. Keep a count C on the number of vertices visited (increment by 1 when visited[v] is set to true)

2.) if C == |V|, (a,b) is not a bridge, otherwise it is a bridge. → O(1)

Time complexity = O(V+E)

9. **Chain of command [25 marks]**

In the army, there is a strict chain of command, and each soldier has a direct superior who he will report to. Any soldier can have 1 or more soldiers of a lower rank under his direct command. This relationship can be modelled as shown below

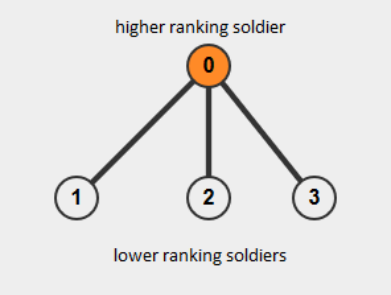


Fig 1

Also, a soldier will never have more than 1 direct superior. Thus the following will never happen

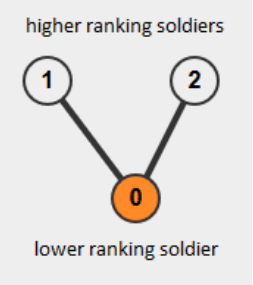


Fig 2

The army chain of command can then be modelled as shown below, where there is always 1 Commander-in-Chief (vertex 3 in the example in Fig 3) who will command the entire army.

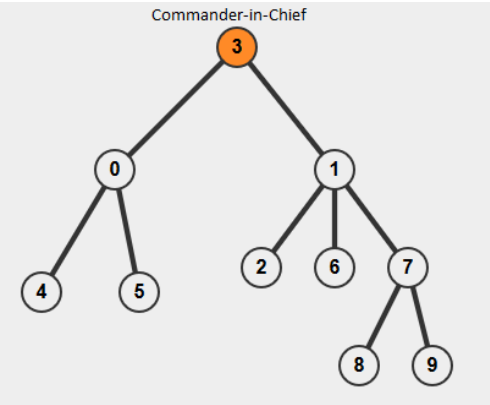


Fig 3

Given this chain of command, a soldier will obey the orders of his superior, his superior's superior and so on until the commander-in-chief. He does not need to obey the order of anyone else.

For example, in **Fig 3**, 7 is the direct superior of 8, while 1 is the direct superior of 7. If 1

issues an order to 8, he will have to obey it since there is chain of command from 8 to 1. However if 0 issues a command to 7, he does not have to obey it since there is no chain of command from 7 to 0.

There can be degenerate cases where the chain of command is simply a straight line or where the chain of command is flat with 1 direct superior and everyone else below him as shown below

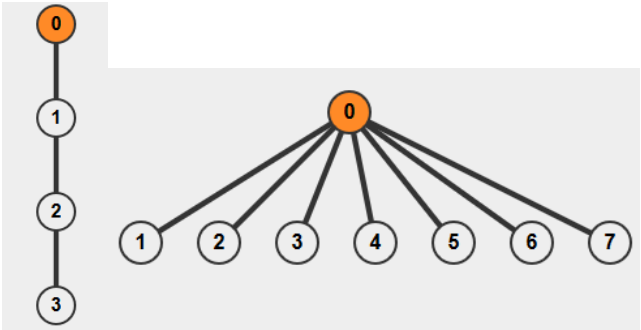


Fig 4

Let each soldier be identified by a number i, where i >= 0.
Let this chain of command be modelled as an integer array C of size N, where N = number of soldiers in the army and can be huge (N > 100000000).
Index i will then refer to soldier i, and $C[i]$ will store the number of i's direct superior.
If a soldier does not have any direct superior $C[i]$ will store i.

In answering any of the sub-questions below, you cannot make changes to C.

*Note: A soldier can refer to army personnel of any rank (from private to commander-in-chief).

a) In Fig 3, how many superiors are there (inclusive of the commander-in-chief) from vertex 8 to vertex 3 (commander-in-chief) and how many superiors are there from vertex 6 to vertex 3? **[2 marks]**
3 and 2 respectively.

b) Write an algorithm to answer in $O(K)$ time the query **numSuperiors(*i*)** which will return K the number of superiors there are in the chain of command from soldier i to the commander-in-chief (the number of superiors should include the commander-in-chief too if i is not the commander-in-chief himself) **[4 marks]**

```
numSuperiors(i)
{
  numS = 0
  while (C[i] != i)
    numS += 1
    i = C[i]
  return numS
}
```

c) When a soldier i is issued an order from soldier j, where $i \neq j$, write a function **obeyOrder(*i,j*)** which will return true if i should obey j and false otherwise. **obeyOrder(*i,j*)** should run in time $O(K)$ where K is the number of superiors in the chain of command from i to the commander-in-chief. **[4 marks]**

```
obeyOrder(i,j)
{
  while (C[i] != i)
    if C[i] == j
      return true
    i = C[i];
  return false;
}
```

d) Perform pre-processing of C in $O(N)$ time and using only an extra integer array D, so that you can answer in $O(1)$ time the query **numSuperiors(i)** (as defined in part b). **[7 marks]**

```
preprocess()
{
  for (i=0 to N-1)
    D[i] = -1
  for (i=0 to N-1)
    computeNumSuperiors(i)
}

computeNumSuperiors(i)
{
  if C[i] == i // base case
    D[i] = 0
  if (D[i] != -1) // already computed
    return D[i]
  else // compute numSuperiors for i
    D[i] = computeNumSuperiors(C[i])+1;
  return D[i]
}


numSuperiors(i){return D[i]}
```

Preprocessing is DP on the graph with a memo table to save the solution for each state/vertex/soldier i. Since each state in the graph is only processed once, in all O(N) states were processed. In processing a vertex i, the outgoing edge from the state (edge to direct superior) is processed (recursive numSuperior2 call to C[i]). Thus in all O(N) edges are processed. So total time taken is O(N).

A less efficient solution is to compute numSuperior(i) without memorization. In this case for a complete and full binary tree, nodes at each level h (with leaf level being h=0) is accessed 2^h times. Thus the total number of accessing of nodes is $(n/2)+(2*n/4)+(4*n/8)+..(2^{lgn}*1)$ = O(n*lgn). Total number of times edge is traversed is also O(n*lgn) thus total time complexity is O(n*lgn).

e) Let C_{ik} be the portion of the chain of command that starts from soldier i and ends at soldier k. Let **length**(C_{ik}) = (number of soldiers in C_{ik}) -1. If there is no chain of command from i to k, then **length**(C_{ik}) = 0.

The **rank** of k is defined to be **rank**(k) = Max(**length**(C_{ik})) for all i from 0 to N-1

In Fig 3, rank of soldier 5 is 0 since there is no other soldier with chain of command to him. Also rank of soldier 1 will be 2 since the longest chain of command ending at 1 has 3 soldiers (9-7-1 or 8-7-1).

Assuming you can answer **numSuperiors(i)** in O(1) time. Pre-process C in $O(N)$ time and answer the query **rank**(i) for any soldier i in O(1) time. This time you can use any data structure(s) you need to help you. **[*8 marks]**

1. Create a list of lists T of size N (similar to an adjacencyList).
2. For 0 to N-1, add i to back of list at T[numSuperior(i)]. Also keep track of maximum value for numSuperior(i) and save this in maxS. → O(N)
3. Create an array R of size N and initialize all value to 0. → O(N)

```
preprocessRank() {
  // Start from vertices at the greatest depth then move up.
  // This will ensure all soldier ranks are computed correctly
  for (i=maxS to 0)
     for (j=0 to length of T[i]-1)
       computeRank(T[i][j])
}

computeRank(i)
{
  // either commander-in-chief or rank of those in chain
  // of command above it already have rank calculated
  if (C[i] == i or R[C[i]] != 0)
     return
  else
     R[C[i]] = R[i]+1
     computeRank(C[i])
}

Rank(i) {return R[i]}
```

Analysis of the run time is similar to c). Thus total time = O(N)

Another solution:

1.) build the tree (rooted at commander-in-chief) from C. That is reverse all the edges. Use an adjacency list A. → O(N)

```
for i = 0 to N-1
   // add i to the neighbor list of its superior C[i]
   A[C[i]].add(i)
```

2.) Now do DP from the root down. Let R be the array of size N to store the rank for each soldier, and initialize it to -1.

```
computeRank(i) {
  if (A[i].size() == 0) // is a leaf
     R[i] = 0
  else if (R[i] != -1) // rank already computed
     return R[i]
  else
     // rank of a i must be the maximum rank among all
     // its children
     R[i] = max(computeRank(j)+1) for ∀ children j of i
  return R[i]
}
```

3.) go through C to find index for commander-in-chief and store in root

4.) To get rank for all soldiers simply call computeRank(root).

Analysis: computeRank will go through all N vertices/states and for each state loop through all edges/transition to children. Thus in total O(N+E). Since this is a tree E=N-1, so this is O(N).

some wrong answers

1. rank(i) = (max(numSuperiors(j)) for all j is child of i) - numSuperior(i)
2. rank(i) = (max(numSuperiors(j)) for j = 0 to N-1) - numSuperior(i)
3. process from soldiers without any children (in any order), but only update rank of superiors once (i.e if rank of superior is not -1 then don't update). This can be wrong if the order of processing is not correct.

10. **Cat burglar "Q" [23 marks]**

First time

A modern day robin hood codenamed "Q" steals from the rich who were made wealthy through exploiting the poor. Tonight he is intent on stealing from a one such tyrant. He has a map of the house which is a one floor building represented as a $(N+1)*(M+1)$ grid.

"Q" can only enter the house from cell (0,0), i.e the top left corner and the safe where the gold is stored is in cell (N,M), i.e the bottom right corner. He can only exit the house from cell (N,M). Each cell contains an integer value i, where $i \geq 1$.

"Q" can only move from a cell to the adjacent cell up/down/left/right to it. Each cell contains an integer value, and the time in minutes to move from a cell A to an adjacent cell B and vice versa is $|value\ in\ A\ -\ value\ in\ B|$.
You can assume that $|value\ in\ A\ -\ value\ in\ B|$ will never be 0.

However, there are also K infra-red triggers spread through the house, and they are represented by "T" cells in the grid. If "Q" steps into such a cell, the alarm will be triggered and he will be caught. An example is show in diagram 1.

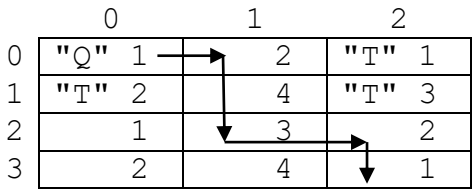


diagram 1: Example grid with $N = 3$, $M = 2$ and $K = 3$. "." are not shown to save space

Assume the grid is given as a 2D array H of (s,i) pair. Where s is a string with value = "T" if the cell contains a trigger and "." otherwise. i is the integer value in the cell.

a) Given a grid of size $(N+1)*(M+1)$, and some value for K, "Q" wants the quickest way get to cell (N,M) without stepping into any of the "T" cells. In **diagram 1**, it will be the path as shown, taking 6 mins.

   Model this as graph and output the fastest time for "Q" to move from cell (0,0) to cell (N,M). If he cannot do so without setting off any trigger, simply output -1

   i. What are the vertices in your graph? **[2 marks]**
   Each vertex represents a cell in the grid.

ii. How are the vertices linked? What are the weights on the edges? **[3 marks]**
For a vertex representing cell (r,c) where H[r][c].first == ".", there is an undirected edge to a cell (r',c') and vice versa if H[r'][c'].first == ".", r' == r$\pm$1 and c' == c or r' = r and c' == c$\pm$1, and r',c' is within the boundary of the grid. The weight of the edge is | H[r][c].second- H[r'][c'].second |.

iii. Give the best algorithm you can think of to solve this problem and analyze its time complexity. **[5 marks]**
This is an undirected non-negative weighted graph and the problem is to find the shortest path from vertex (0,0) to vertex (N,M). Simply run original/modified Dijkstra's from vertex (0,0) and return the shortest path cost to (N,M) and also the path itself by tracing the predecessors from the predecessor array.
If shortest path cost to (N,M) is ∞ then there is no way for Q to escape.

Time = O((N*M)*log(N*M))

Second time

After a few months, "Q" decided to "hit" the same house. However after he steps into the house (at cell (0,0)), he receives a message from his hacker sidekick. He checks the message and realizes it is a new map of the house. The owners have changed their house security so that **EVERY** cell now contains a trigger except cell (0,0) and cell (N,M).

If the triggers are set off L times, the house will seal itself off trapping "Q" inside. Also, if "Q" steps into a cell with a trigger multiple times, each time will add to number of times the triggers are set off.

However, there are also K' exposed electric mains, represented as "E" in the grid, spread throughout the house except cell (0,0) and cell (N,M) so that if "Q" switches off an electric main, it will disable the trigger located in that cell. Thus you can consider stepping into any of these cells will not set off the trigger. All cells with no exposed electric main are represented as "."

This time, since "Q" is now familiar with the layout of the house, he can move **from any cell to its adjacent up/down/left/right cells in 1 minute**. An example is shown in diagram 2. The cells (0,0) and (3,2) do not have a trigger, while stepping into "E" cells will not set off the trigger.

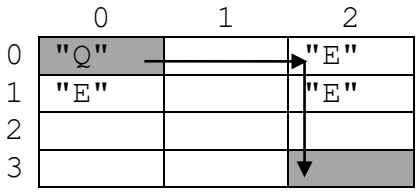


diagram 2: Example grid with $N = 3$, $M = 2$ and $K' = 3$. "." are not shown to save space

If $L = 3$, "Q" can safely and most quickly reach cell (N,M) using the path as shown, taking only 5 minutes and the triggers are only set off 2 times (at cells (0,1) and (2,2)).

The grid is now given as a 2D array H of strings. H[a][b] = "E" if the cell(a,b) contains an electric main. Otherwise H[a][b] = "."

b) Given a grid of size $(N + 1) * (M + 1)$ and a value for K' and L, write an algorithm to output the fastest time for "Q" to get to cell (N,M) without becoming trapped in the house. If he cannot reach cell (N,M) without becoming trapped, output -1.
Again model this as a graph and solve it.

i. What are the vertices in your graph? **[2 marks]**
Each vertex represent an integer triplet (r,c,b) where (r,c) is a cell in the grid and b = 0 if H[r][c] = "E" and 1 otherwise.

ii. How are the vertices linked? What are the weights on the edges? **[3 marks]**

Each vertex representing (r,c,b) is connected to the vertices up/down/left/right of it and vice versa, i.e an undirected edge, if they are within the grid boundary. Edge weight is 1 for all edges.

iii. Give the best algorithm you can think of to solve this problem and analyze its time complexity. **[8 marks]**

This is shortest path on an unweighted undirected graph, with a limitation on the number of moves made, it is similar to PS5 subtask C but where there can be vertices which do not cause increase to "number of junctions/alarm triggering". Modify BFS to solve this.

Assuming graph stored as adjacency list with hashmap to map vertex triplet to a vertex number. Let D and P be 2D arrays of size (N*M)*(L) . D[i][j] == length of the shortest path from source vertex to vertex i setting off the triggers j times. P[i][j] == (i',j') where i' is predecessor of i in shortest path from source vertex to vertex i setting off the triggers j' times, where j' == j or j-1.

Initialize D[i][j] = INF for all i and j. D[source vertex][j] = 0 for all j. Source vertex is the vertex representing cell (0,0)

1.) For the BFS queue, enqueue (v,t) where v is the vertex, t is number of times the triggers have been set off.
2.) At the start enqueue (vertex representing (0,0), 0)
3.) While queue is not empty
    a. dequeue the current head (v,t) of the queue
    b. if v is vertex representing destination cell, return D[v][t]
    c. Go through the neighbors u of vertex v
        i. If (u.b = 1 and t+1 < L and D[u][t+1] == INF)
            Enqueue (u,t+1)
            D[u][t+1] = D[v][t]+1
            P[u][t+1] = (v,t)
        ii. If (u.b = 0 and D[u][t] == INF)
            Enqueue (u,t)
            D[u][t] = D[v][t]+1
            p[u][t] = (v,t)
4.) return -1

Get path by tracing predecessors of P[v][t]. → O(L+K') at most use L normal cells and K' "E" cells.

Time complexity = O(L*N*M) since computing D[i][j] for each value of j and all values of i takes O(N*M) time and there are L of such j values.

Wrong DP solution:

Naïve modelling problem as a DAG, where each cell only has an additional parameter J representing the number of triggers left, then doing DP will not work as this will not convert the graph into a DAG. This is because adjacent "E" cells can have edges to each other no matter the number of triggers left since stepping into them does not increment the number triggers, causing cycles to occur. Thus this is not a DAG.

A correct DAG modeling for DP:

The proper way to model as a DAG is to include a 2nd parameter R which is the number of edges used to move to the current cell. It will have value from 0 to K'+L (K'+L since this is the maximum edges that any valid path can take).

For a pair of vertices A (r,c,J,R) and B (r',c',J',R'), there is a directed edge from A to B if

1.) They are adjacent
2.) A and B are both "E" cells and J' = J, R' = R+1
    or
3.) B is not an "E" cell and J' = J-1, R' = R+1

This way the "E" cells cannot form cycles since R is always increasing.

Memoization: Need a 4D array D of size (N+1)*(M+1)*L*(K'+L) = O(L*N*M*(K'+L)) or if map all the vertices to numbers, just a 1D array of the same size.

Initialization: Set shortest path distance of vertex (0,0,L-1,0) to be 0, the rest to infinity.

Then this is simply SSSP on a DAG. Perform one-pass bellman ford. →O(L*N*M*(K'+L))

The sp from (0,0) to (N,M) is then min among D[N][M][0 to L-1][0 to K'+L]. →O(L*(K'+L))

If all of them are infinity then there is no valid shortest path.

Total time = O(L*N*M*(K'+L))

~~~ END OF PAPER ~~