NPTEL Questions

CP — July-Nov 2021

Week 5 — BFS and DFS

1. (2 points) You are given a **n**×**n** grid and there are people on some positions of the grid. Some of these people are infected with COVID, let's call the initial set of infected people as $D_{initial}$. An infected person can transmit the disease to the people who are in adjacent cells. Two cells are adjacent if they share a border (in particular, diagonal positions are not adjacent).

   After some time the people in $D_{initial}$ will potentially infect other people who in-turn may infect more people, let's call the final set of infected people as D_{final}. Your task is to cure exactly one person from $D_{initial}$ so that the final number of infected people (i.e size of D_{final}) is minimised. If there are ≥ 2 nodes that minimise size of D_{final}, return any of them. Note that the person you cure can get infected again.

   (a) Create a Graph where nodes are people in the grid. Add undirected edges between people who are adjacent in the grid. Which of the following is/are correct approaches to solve this problem ?

   A. Calculate the degree of each node in $D_{initial}$ and remove the node with the maximum degree.

   B. For each node in $D_{initial}$ compute the size of the component which contains that node. Return the node which is a part of the largest component.

   C. Both the approaches Fail.

   Explanation: Observe that if there are two nodes of one component in $D_{initial}$ then removing either of them won't affect D_{final}. For every component, if there are ≥ 2 nodes of that component in $D_{initial}$ discard all nodes of that component from $D_{initial}$. Among all nodes that are not discarded return the node which is part of the largest component. It may happen that all nodes of $D_{initial}$ are discarded in that case we can return any node.

   (b) What is the time complexity to find the size of the component corresponding to a node in $D_{initial}$? You have to give the answer for both adjacency list and adjacency matrix representation. The first entry in each of the following options is for adjacency list and the second is for adjacency matrix.

   A. $O(n)$, $O(n^2)$

   B. $O(n)$, $O(n^3)$

   C. $O(n^2)$, $O(n^2)$

   D. $O(n^2)$, $O(n^3)$

   E. $O(n^2)$, $O(n^4)$

   Explanation: You will have to run a BFS/DFS from the node to calculate the size of it's component. The time complexity of BFS/DFS is $O(|V|+|E|)$ for adjacency list and $O(|V|^2)$ for adjacency matrix. In this Graph $|V|$ is $O(n^2)$, $|E|$ is also $O(n^2)$ because every vertex can have at most 4 edges.

2. (3 points) You are assigned to perform n tasks. You are also given an array $constrains$. Each element of $constrains$ is of the form $(task1, task2)$ where $task1$ and $task2$ are tasks that you have to perform and $task1$ must be performed before $task2$. The $constrains$ array will always be such that there exists at-least one valid ordering to complete the tasks.

   (a) Suppose you are given 5 tasks to complete and $constrains =$ [(task1, task2),
   (task1, task4),
   (task1, task5),
   (task2, task3),
   (task4, task5)]
   In how many ways you can complete all tasks with these constrains.

   A. 5

   B. 6

   C. 7

   D. 8

Explanation: The following are all the valid orderings.
task1, task2, task3, task4, task5
task1, task4, task5, task2, task3
task1, task2, task4, task3, task5
task1, task2, task4, task5, task3
task1, task4, task2, task3, task5
task1, task4, task2, task5, task3

(b) Consider the following approach to find a valid order to complete the tasks.

```
Create a Graph G. The nodes of the Graph are tasks.
For every element (task1, task2) in constrains
add a directed edge between task1 and task2.
Let Ans be an empty array.
mark all nodes as unvisited

RECURSIVE_FUNCTION(node):
    mark node as visited
    for v such that (node,v) is an edge:
        if node is un-visited:
            RECURSIVE_FUNCTION(v)
    Ans.append(node)

for node in G:
    if node is un-visited:
        RECURSIVE_FUNCTIONnode)
Print elements of Ans in Reverse order
```

A. This approach always works.
B. This approach never works.
C. Depends on the Graph.

Explanation: A valid ordering of tasks is a topological sort of the created Graph. The described approach performs a DFS traversal of the Graph. Topological sorting is one of the applications of DFS.

(c) Consider the following approach to find a valid order to complete the tasks.

```
Create a Graph G similar to the approach above.
Let Ans be an empty array.
Pick any vertex of in-degree 0 from G and append it to Ans array.
Delete the vertex and all edges incident to it.
Repeat the above two steps till there a vertex in the Graph.
Print elements of Ans
```

A. This approach always works.
B. This approach never works.
C. Depends on the Graph.

Explanation: A valid ordering of tasks is a topological sort of the created Graph. The above approach is valid because at each step a task(node) of in-degree 0 is taken and in-degree 0 means that the task can be done before any other task(node) currently in the Graph.

3. (3 points) Suppose you are the Captain of a ship and you have been provided with a map of the surrounding waters. The map is in the form of a **m**×**n** binary matrix grid, in which 1's represent land and 0's represent water. Now, an island can be formed by 4-directionally connecting the adjacent lands(1's),i.e.,

horizontally or vertically(no diagonal connection), and every island is surrounded by water(0's). All the 4 edges of the grid are surrounded by water, which means, the possible locations of islands present are all that can be seen in the grid. As the Captain, in order to consider all your possibilities, it's your task to first determine the number of islands present in the water grid!!!

(a) Consider the following approaches based on traversal alternatives available in the options and think which one to choose.

I. Iterate through all the cells in the matrix, and if a land(1) is encountered, do a **Breadth-First Search(BFS)** Traversal to sink all it's adjacent lands, and then increase the counter by 1.

II. Iterate through all the cells in the matrix, and if a land(1) is encountered, do a **Depth-First Search(DFS)** Traversal to sink all it's adjacent lands, and then increase the counter by 1.

A. Only I is correct

B. Only II is correct

C. Both I and II are correct

D. Neither I nor II is correct

Explanation: Both the BFS and DFS traversal can effectively mark the adjacent lands, and hence, will avoid repetitive counting

(b) Now that the Captain has determined the total number of islands present in the grid, it is decided that they will head for the island with the largest area. Consider the following approach to find the island with max area and decide if it will give us the correct answer or not?

```
Iterate over the cells of the grid:
        If a land(1) is encountered:
          Sink that land, and calculate the area of
          island by performing dfs in all 4-directions.
          Update the maxArea, and islandPositions accordingly.
```

A. Yes, it will give the correct answer.

B. No, this won't work.

C. Can't say.

Explanation: Just a slight variation of the previous part, kind-of intuitive approach.

(c) What will be the time complexity of implementing the approach discussed above?

A. $O(m+n)$ **B.** $O(mn)$ C. $O((mn)^2)$

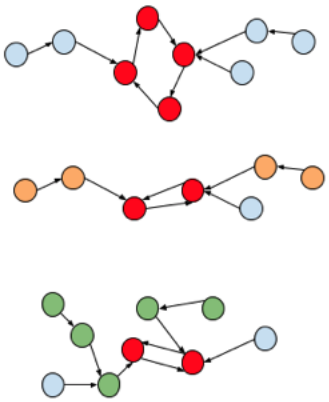
Explanation: Max number of DFS calls can be mn (in the case, when all elements of the grid are 1s) and the number of iterations are also mn. So, the total time taken $= 2mn$, hence, the overall time complexity $= O(mn)$.

4. (2 points) You are a teacher with N kids in your class, and each one has a different student ID number from 1 through N. Every kid in your class has a single best friend forever (BFF), and you know who that BFF is for each kid. BFFs are not necessarily reciprocal – that is, B being A's BFF does not imply that A is B's BFF.

Your lesson plan for tomorrow includes an activity in which the participants must sit in a circle. You want to make the activity as successful as possible by building the largest possible circle of kids such that each kid in the circle is sitting directly next to their BFF, either to the left or to the right. Any kids not in the circle will watch the activity without participating.

Your goal is to seat the greatest number of kids that can be in the circle. Let this number be K.

We can model this situation with a directed graph G where the nodes are kids and the edges go from each kid to that kid's BFF. This type of graph has a particular property: each connected component of the underlying undirected graph is made up of a cycle, and if you were to delete this cycle then you would be left with a collection of disjoint paths. For an example of this, see the image below.



Note: Each part below has a weightage of 0.25 marks.

(a) Suppose $N = 10$ and the graph has one component with a cycle on students $1, 2, 3, 4, 5$, and the students $6, 7, 8, 9, 10$ have BFFs $1, 2, 3, 4, 5$ respectively. What is the value of K?

Expected answer: 5

Explanation: The circle can consist of the cycle on students $1, 2, 3, 4, 5$. Note that nobody else can be added to this circle, and there is no other way to form a larger circle.

(b) Suppose $N = 5$ and 1 and 2 are mutual BFFs (i.e, 1 is the BFF of 2 and 2 is the BFF of 1). Further, suppose 5's BFF is 4, 4's BFF is 3, and 3's BFF is 2. What is the value of K?

Expected answer: 5

Explanation: In this case, we can seat everyone in the following order: $1, 2, 3, 4, 5$. Even though 5 and 1 are not BFFs, 5 is seated next to 4, and 1 is seated next to 2.

(c) Consider the graph G. Is it possible for any connected component in the underlying undirected graph to have size one? In other words, can G have an isolated vertex?

A. Yes **B. No**

Explanation: Notice that everyone has at least one BFF, so there are no isolated vertices.

(d) Consider the graph G. Is it possible for any connected component in the underlying undirected graph to be acyclic?

A. Yes **B. No**

Explanation: Notice that if a connected component has r vertices in the underyling undirected graph, it has r edges, and since any acyclic graph on r vertices has at most $r - 1$ edges, the connected components cannot be acyclic.