CSE 101 Homework 1

Winter 2023

This homework is due on gradescope Friday January 20th at 11:59pm on gradescope. Remember to justify your work even if the problem does not explicitly say so. Writing your solutions in LaTeXis recommend though not required.

Question 1 (Rearranging Furniture, 40 points)**.** *John has a couch that he would like to move from one location in his apartment to another. Unfortunately, his apartment has a twisty maze of narrow hallways and it is not clear if there is room to make the move. The apartment is contained in a cube S feet on a side and the walls and other obstacles are specified by a collection of N triangles arranged in this space. The couch is a solid body whose surface is described by N triangles, but it can be moved and rotated freely so long as it does not collide with the walls. John has the current position and orientation of the couch as well as the desired position and orientation.*

Suppose that it is possible for John to move his couch from its starting position to its ending position in such a way that it never comes within closer than an inch to any obstacle in the house. Give an algorithm which runs in time polynomial in $S \cdot N$ to provide a path (given as a sequence of continuous movements of the couch) to bring the couch from its current location to the desired one without colliding with any obstacles in his apartment.

Hint: You will need to find a way to discretize the problem, turning it into a sequence of tiny steps between a finite number of possible states.

Let's define couch position by its 3D coordinates (x, y, z) and its rotation (yaw, pitch, roll). This way, we can represent any possible couch position by a tuple of 6 numbers (x, y, z, yaw, pitch, roll). We can see that any couch position can be represented by such a tuple, and every tuple corresponds to a couch position (which may be colliding with some obstacles, or going out of bounds).

The main objective of this problem is to create a graph, where each vertex is a possible couch state, and vertices are connected by an edge if they are close to each other and you can move the couch from one state to another. In order to make it possible and have a finite number of states, we have to discretize the possible couch states. After creating the graph, we need to prove that these discrete positions would lead to a valid path in the continuous version of the problem. Then, we can just run known graph algorithms on the discrete graph and find the solution.

Let's define a constant k equal to one-twentieth of an inch. Let α be an angle small enough that if we rotate a segment of length S around one of the ends, the other end will not move more than k, but big enough that for 2α this does not hold. We can define vertices of our graph by positions of the couch where the coordinates are multiples of k and rotations are multiples of α. It's obvious that each coordinate will have $O(S)$ possible values. For rotation, we can see that we enclose a circle when we consider all possible rotation angles. If we rotate a segment of length S, distances between adjacent rotations will be smaller than k, and the sum of their lengths will be smaller than $2\pi S$, so there will be $O(S)$ possible rotation angles as well. Hence, there are $O(S^6)$ vertices in our graph.

Let's define a valid couch position as a position that is not closer than $10k$ (half of an inch) to any obstacle. We can calculate the validity of a position in $O(N^2)$ time since for each couch triangle we need to check the distance to each obstacle triangle and to every wall. Therefore, we calculate the validity of all possible positions in our graph in $O(N^2S^6)$.

We create edges in our graph by connecting adjacent valid couch positions – couch positions that differ by at most k in each coordinate and at most α in each rotation. Since there are $O(S^6)$ unique couch positions, there are also $O(S^6)$ edges. It is clear that we can move the couch safely through edges since each edge changes each coordinate/rotation by at most k/α, so any point of the couch will move at most $6k$ towards any obstacle, but we had left a margin of $10k$ with obstacles when defining valid couch positions, so each edge defines a valid movement.

Now it's worth noting, that any path in our graph describes a valid movement. Since we can move the couch between positions if there is an edge between vertices (we just change coordinates and angles in a continuous way and the proof above makes sure we do not collide with anything), then any path is a sequence of valid moves.

If we analyze the route that never comes closer than an inch ($20k$) to an obstacle, we can see that for every moment, there are positions represented by vertices of our graph that differ at most k in coordinates and at most α in rotations. These positions are obviously valid (since when moving at most k in coordinates and α in angles, we move at most $6k$ closer to an obstacle, but we have a $10k$ margin) and therefore are all connected by edges. So while traversing the continuous movement, we see there exists a path of connected valid vertices between the beginning and the end. Specifically, the beginning and ending positions are close to some valid vertices, we will mark one of each as the beginning and end vertex (we can move the couch to these positions without obstruction). This means there surely exists a path between the beginning and end vertices in our graph.

After creating the graph and knowing that any path is a valid sequence of couch movements, we just have to find a path between the starting and ending vertices, which we can do using an explore algorithm. What's important, the explore algorithm itself doesn't return a path, so we have to modify it. One way to do this is to store for each vertex from which other vertex it was reached for the first time during the explore algorithm. This way, after the explore algorithm is finished, we can backtrack from the ending vertex and retrieve the path. It works in complexity $O(V + E) = O(S^6)$, so the overall complexity of our algorithm is $O(S^6N^2)$.

Question 2 (Lab Safety, 40 points)**.** *Rose is a chemist. In a complicated experiment she has n containers each containing one of k different chemicals. Furthermore, there are m pairs of containers where chemicals can travel from one container to the other and back. Unfortunately, if all k chemicals are allowed to mix in the same container, it will produce toxic byproducts and Rose needs to be able to determine whether this is possible with her setup.*

(a) Give an algorithm that given descriptions of the containers, the connected pairs and a description of which chemical starts in each container determines whether or not there is any container which every chemical is capable of reaching. Your algorithm should have runtime $O(n + m)$. [15 points]

(b) Suppose instead of being bi-directional, some of the paths between containers can only be traversed in one direction. Can the algorithm from part (a) still work in this context in $O(n+m)$ time? Why or why not? [10 points]

(c) For the more complicated version of this problem discussed in part (b), give an algorithm that runs in time $O(k(n + m))$. [15 points]

Rose is working with a graph with n vertices and m edges. For simplicity of the solution, we can assume all k chemicals are integers from 1 to k.

(a) We first need to run the algorithm to compute the connected components and then for each component, we need to check if it contains all k chemicals. For each component, we can run explore and mark all the different chemicals we encountered in the visited containers. One way to do this is to run explore to check if the size of the component is smaller than k. If it is, obviously there

cannot be a toxic mix in that component. If it's greater or equal to k, we can create an array of size k filled with zeros. Then we run explore, if we come across chemical i, we put a 1 in the i-th cell in our array. If after that, all k chemicals are marked with ones, it means that they are all in the same component, so they can flow into the starting container from which we ran explore algorithm. If on the other hand, there is some chemical that we didn't come across, it means it doesn't exist in this component, so there is no way for all chemicals to mix in any container of this component. Since we can treat all of the connected components independently, we should output YES if for any connected component there are all k chemicals inside it. If in none of the components there is a possibility of achieving the toxic mix, then we can output NO. Overall, we can detect the possibility of a toxic mix in O(n+m) complexity.

(b) No, it does not work. Since the edges are directed, chemicals cannot flow both ways. This means that we cannot split the graph into independent components where chemicals can flow between any two containers. This was a key observation in the previous task, so without it the algorithm won't work.

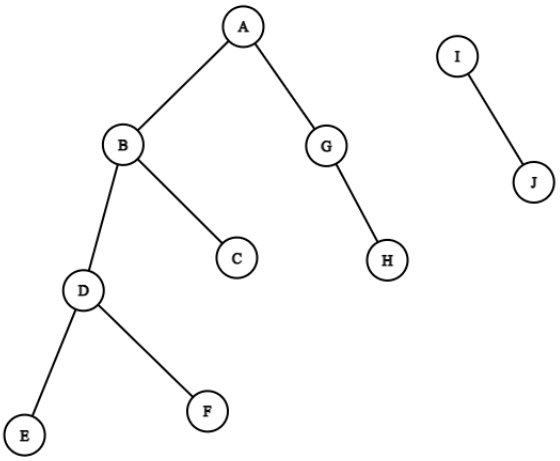
(c) A naive solution to this problem would be running an explore algorithm starting from each vertex, marking all the vertices that the starting vertex can reach, and checking if any of the vertices are reached by all k chemicals at the end of the algorithm. However, this algorithm would have a complexity of $O(n(n+m))$, since we are running an explore algorithm n times. In order to improve the complexity to $O(k(n+m))$, we need to run the explore algorithm exactly k times, once for each chemical.

For each chemical c we can create an artificial container with c in it to act as a source container, and add edges from this container to all containers containing c in the actual graph. This way, running explore from this container is like running explore from all containers containing c at the same time, with complexity $O(n+m)$. Since we run the explore algorithm once for each chemical, the overall complexity of exploring is $O(k(n+m))$. After running the explore algorithm for each chemical, we just have to check if any vertex was reached by all k chemicals, which can be done in $O(nk)$. Therefore the overall complexity of our algorithm is $O(k(n+m))$.

Question 3 (Preorder and Postorder and Number of Edges, 20 points)**.** *Suppose that G is an undirected graph with 10 vertices $A, B, C, D, E, F, G, H, I, J$. When running DFS on G the vertices and assigned preorder and postorder numbers shown in the table below. What are the largest and smallest possible numbers of edges in G?*

Vertex	*Pre-*	*Post-*
A	*1*	*16*
B	*2*	*11*
C	*3*	*4*
D	*5*	*10*
E	*6*	*7*
F	*8*	*9*
G	*12*	*15*
H	*13*	*14*
I	*17*	*20*
J	*18*	*19*

Let's draw the DFS tree/forest. It will help us visualize the problem.



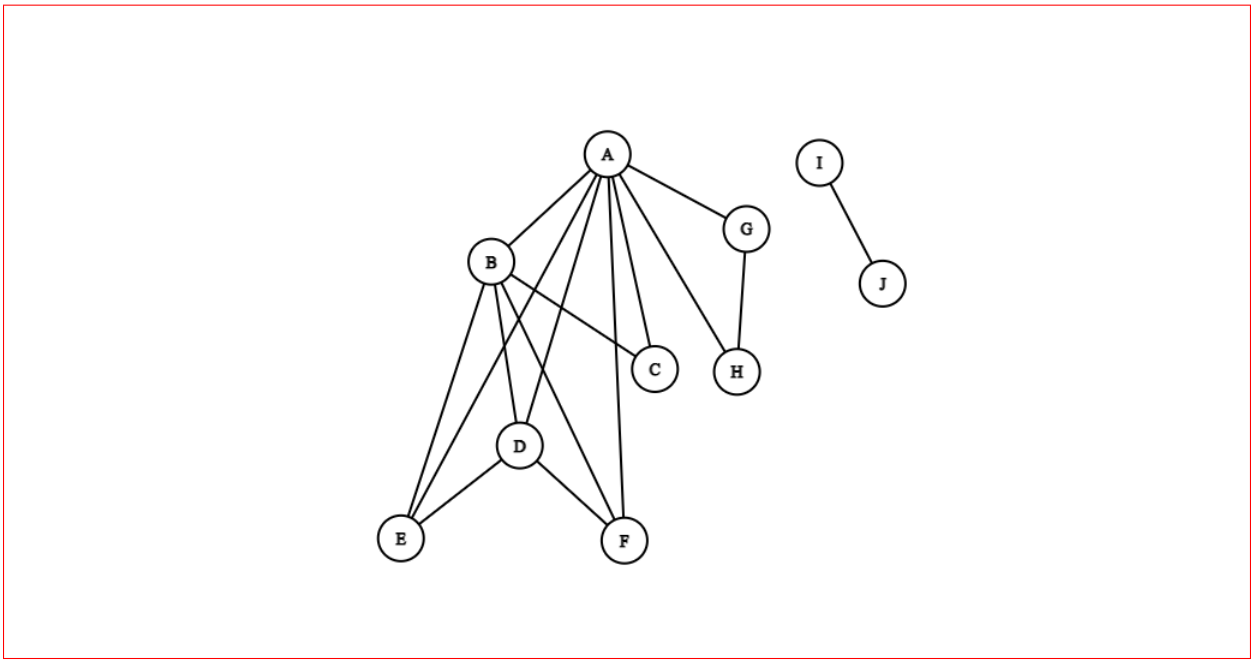
A connected component with n vertices has at least $n-1$ edges. Hence, G has at least 8 edges. The forest drawn above is an example of an 8-edge graph satisfying the given tables.

Lemma: Assume a connected component G and a DFS tree T in this graph. All the edges in G can only connect a vertex and its ancestor in T.
Proof: Let's consider an edge between vertices u and v. Let's assume v is visited first by the DFS. At that time, u is not visited, so before leaving v, our DFS will have to visit u. If not, it will just traverse our edge to visit u. Since u will be visited while we are traversing v's subgraph, u will be in v's DFS subtree, which means v will be an ancestor of u, which ends the proof.

The proven lemma suggests that we can only add edges to ancestors in order not to break the pre/post orders of the DFS tree. So for each vertex, we can add all the edges to the ancestors. Since any other edge is forbidden by the lemma, this will create the largest possible graph. The table below shows for each vertex how many ancestors it has.

Vertex	Ancestors
A	0
B	1
C	2
D	2
E	3
F	3
G	1
H	2
I	0
J	1

Summing these up, we conclude that our graph can have at most 15 edges. The graph below is an example of a valid graph with 15 edges.



Question 4 (Extra credit, 1 point)**.** *Approximately how much time did you spend working on this homework?*