CS 5800: Algorithms SEC 05 Summer Full 2024 (Seattle)
Homework 4

Payal Chavan

06/14/2024

Question 1: *Undirected vs. directed connectivity.*

Part (a): Prove that in any connected undirected graph G = (V, E) there is a vertex $v \in V$ whose removal leaves G connected. (Hint: Consider the DFS search tree for G.)

Solution (a):

To prove that in any connected undirected graph G = (V, E) there is a vertex $v \in V$ whose removal leaves G connected, we can perform the Depth First Search (DFS) algorithm. Following are the steps to approach the proof:

1. Let us consider a depth first search tree (DFS) traversal on a graph (G), with an arbitrary vertex (u).
2. Given that Graph (G) is connected, the DFS will traverse through all the vertices in the graph.
3. During the traversal, the edges in the DFS tree can be categorized as either tree edges or back edges.
4. After removing vertex (v) from the graph (G), due to its connectedness, there will still exist a path between any pair of vertices in (G).
5. When we remove vertex (v) from the graph, the resulting DFS tree will have a subtree rooted at each child of (v). These subtrees remain connected to the rest of the graph through (v).
6. So there exists a vertex $v \in V$, which is a neighbor of (u), such that removing (v) along with its incident edge will not disconnect the graph (G).
7. The removal of vertex (v) from the graph (G) will still leave (G) connected.
8. Let's take an example to prove this.
   Suppose, we have a connected undirected graph (G) having vertices (V): P, Q, R, S, T. And the edges (E) are: PQ, QT, PS, SR, QR.
   The degree of each vertex is two because each vertex is connected to two edges. However, vertex T has a degree of one.
   Now, let's apply DFS on this graph, we get the order of traversal as: P–> Q–> R–> S–> T
   Now, even after removing vertex T from the undirected graph, the condition still holds, and the graph remains connected.
9. Therefore, in any connected undirected graph G = (V, E), there exists a vertex $v \in V$ whose removal leaves G connected.

Part (b): Give an example of a strongly connected directed graph G = (V, E) such that, for every $v \in V$, removing v from G leaves a directed graph that is not strongly connected.

Solution (b):

1. In a directed graph, strongly connected components ensure that every vertex is reachable from every other vertex. A graph is considered strongly connected only when all its vertices have directions to both their incoming and outgoing edges.

2. Let us take an example to illustrate this.
   Suppose, we have a connected directed graph (G), and it's vertices $V = \{v1, v2, v3\}$ and edges are given by $E = \{(v1, v2), (v2, v3), (v3, v1)\}$.
3. All the vertices have indegree one. Also, it is a strongly connected directed graph where every vertex is reachable from every other vertex.
4. The only way to traverse this graph is by going from $v1$ to $v2$, then from $v2$ to $v3$, and finally from $v3$ back to $v1$.
5. Now, if we consider removing any vertex $v \in V$ from this graph, the resulting graph will not be strongly connected. For example:
   By removing $v3$ it is no longer possible to reach $v1$ from $v2$. Similarly, for the remaining vertices, they do not adhere to the property of being part of a strongly connected directed graph, because each and every vertex is not reachable to any other vertices.
6. Therefore, the directed graph $G = (\{v1, v2, v3\}, \{(v1, v2), (v2, v3), (v3, v1)\})$ serves as an example where removing any vertex v from G leaves a directed graph that is not strongly connected.

Part (c): In an undirected graph with 2 connected components it is always possible to make the graph connected by adding only one edge. Give an example of a directed graph with two strongly connected components such that no addition of one edge can make the graph strongly connected.

Solution (c):

1. In a graph composed of two disjoint cycles, each cycle forms an individual strongly connected component. However, adding just one edge is insufficient because it only enables traversal from one component to another, without allowing a return path.
2. Let's illustrate this with an example.
   Suppose, we have a directed graph G = (V, E), with vertices V: $\{v1, v2, v3, v4\}$ and edges E: $\{(v1, v2), (v2, v3), (v3, v1), (v4, v4)\}$.
3. In this graph we have 2 strongly connected components:
   $\{v1, v2, v3\}$ - In a directed graph, these three vertices constitute a strongly connected component because there exists a directed path between any pair of vertices within this subset.
   $\{v4\}$ - Vertex $v4$ constitutes a separate strongly connected component.
4. The addition of any single edge cannot connect the two strongly connected components in this graph. If we were to add an edge between any vertex in $\{v1, v2, v3\}$ and $\{v4\}$, it would not create a directed path from $v4$ to any vertex in $\{v1, v2, v3\}$ due to the cycle present within $\{v1, v2, v3\}$.
5. Therefore, this example illustrates a directed graph with two strongly connected components that cannot be made strongly connected by adding just one edge.

Question 2: *Undirected Graph*

Part (a): Run DFS on the graph starting at node A and show the exploration tree, including pre/post values. Process vertices in alphabetical order.

Solution (a):

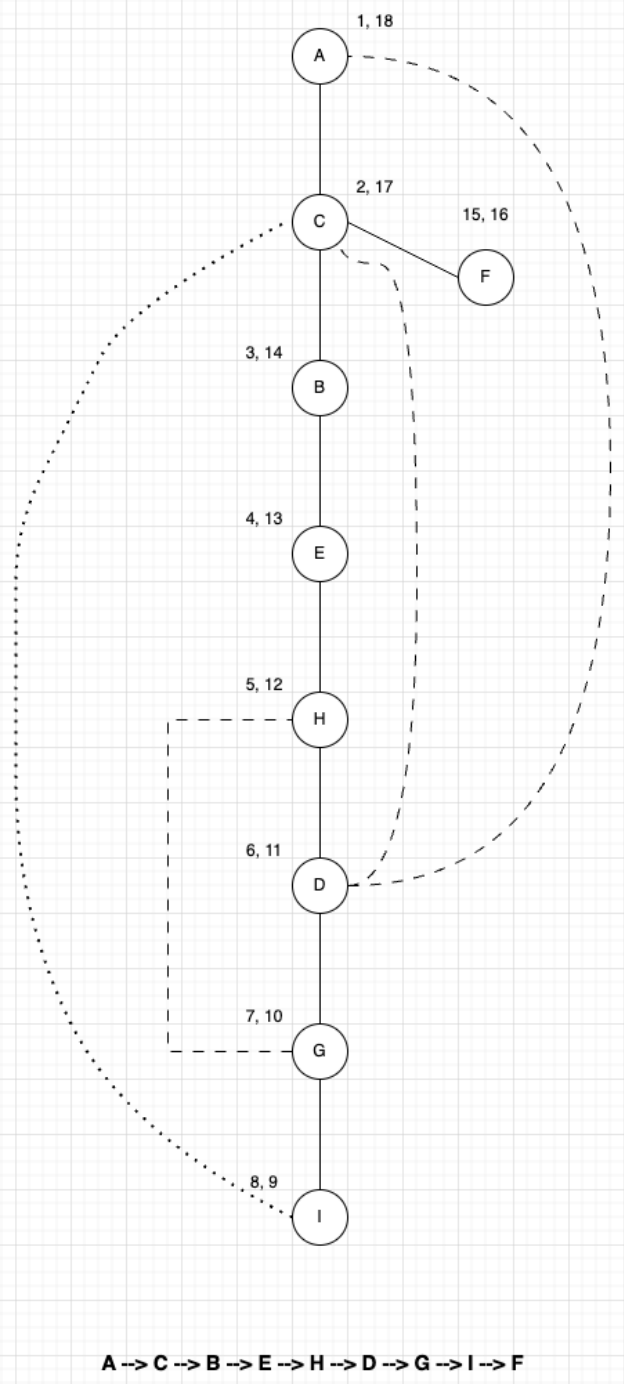

Figure 1: DFS Exploration Tree on Undirected Graph.

Part (b): Run BFS on the graph starting at node A. Show the exploration tree.

Solution (b):

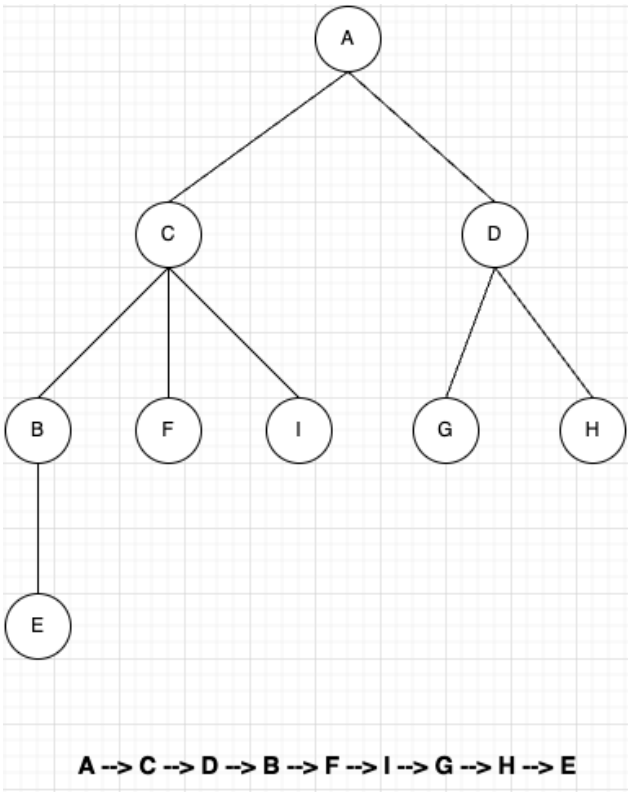

Figure 2: BFS Exploration Tree on Undirected Graph.

Question 3: *Directed Graph*

Part (a): Starting with node I, run DFS on the reverse graph, storing pre/post.

Solution (a):

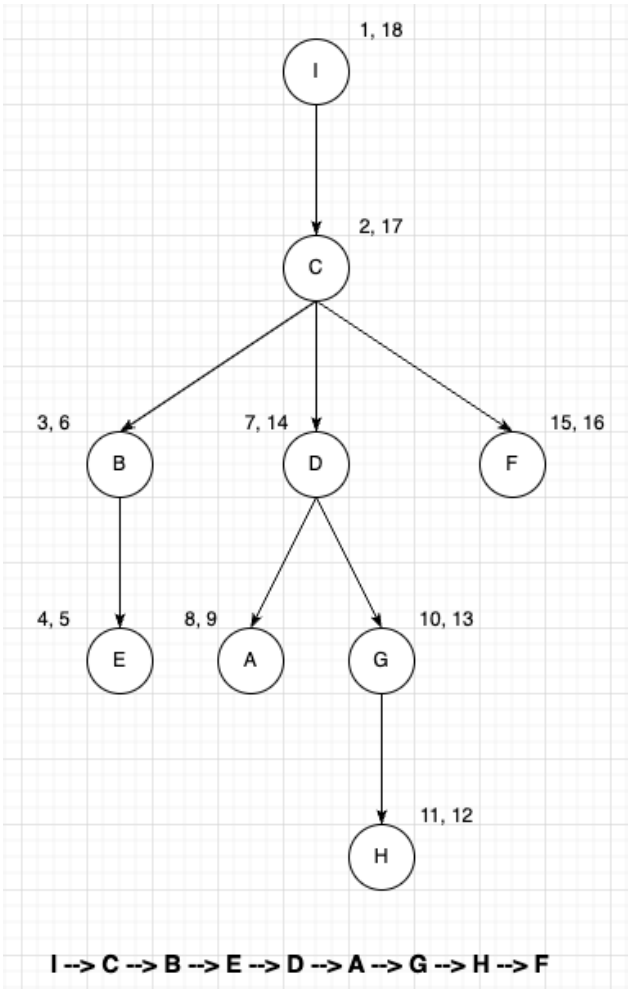

Figure 3: DFS Exploration Tree on Directed Graph.

Part (b): Run connected components on the original graph, initialize the DFS search using the vertices in decreasing order of their post numbers. Once a vertex is included in a component, it cannot be added to another. Show the connected components and the meta-graph giving the connectivity of each group. Identify which groups are sources and which sinks.

Solution (b):

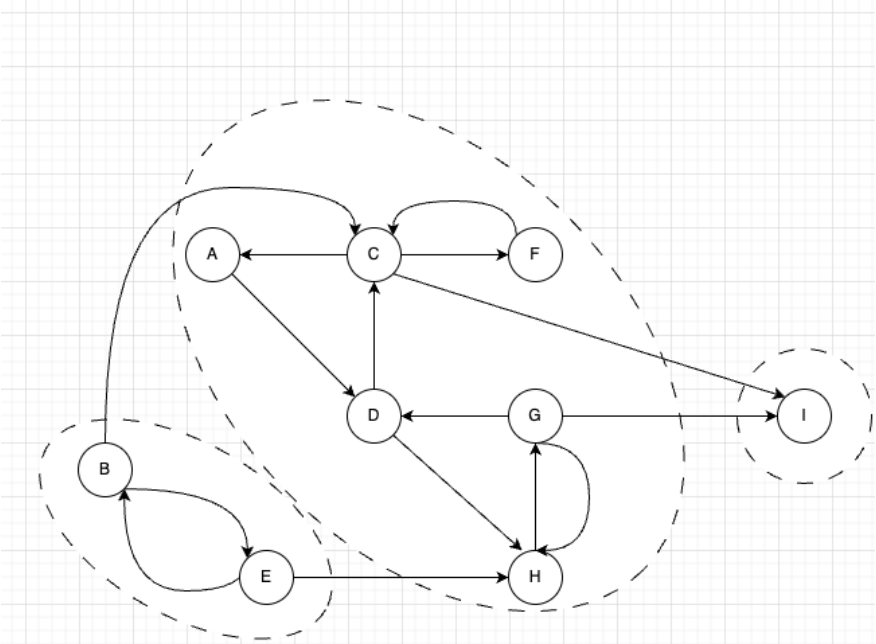

Fig a: A directed graph and it's connected components.

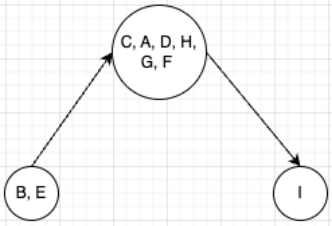

Fig b: The meta-graph.

In a DAG, we have 2 sources i.e., (B, E) and 1 sink i.e (I).

Figure 4: A directed graph with connected components and it's meta graph.

When we explored the DFS tree, we got the post numbers as follows:
'E': 5, 'B': 6, 'A': 9, 'H': 12, 'G': 13, 'D': 14, 'F': 16, 'C': 17, 'I': 18

With these post numbers, we got 3 connected components on our original graph.

We then got a meta-graph which is a directed acyclic graph (DAG) with 2 sources(B and E) as we can see B and E does not have incoming edge(s) and 1 sink (I) because node I does not have any outgoing edge(s).

Question 4: *Directed Weighted Graph*

Part (a): Create the DFS tree, starting at node A. Show all forward, back, and cross edges and pre/post numbers. How do you know that back edges exist in the graph? The existence of a back edge implies what?.

Solution (a):

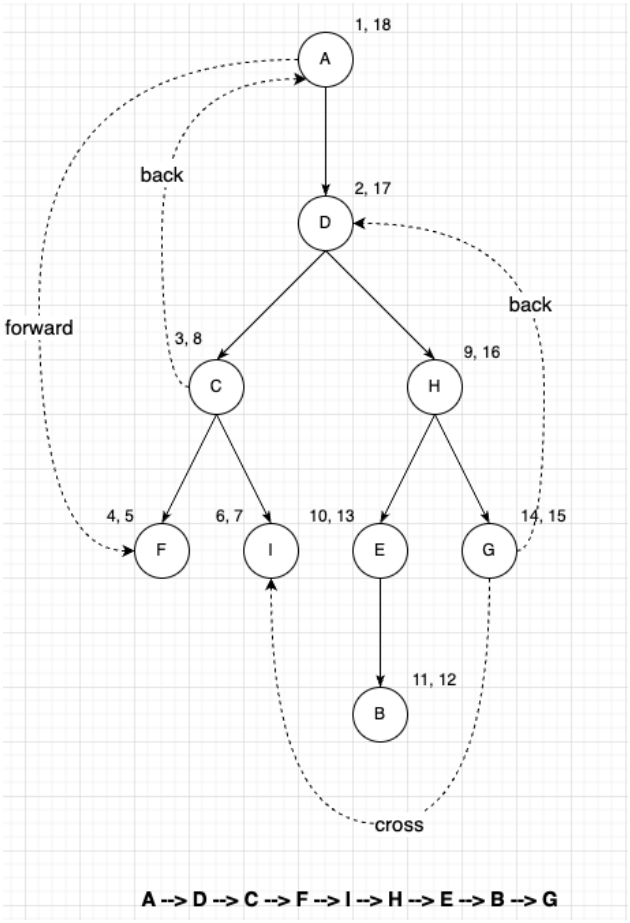

Figure 5: DFS Exploration Tree on Directed Weighted Graph.

In a depth-first search (DFS) or breadth-first search (BFS) traversal, a back edge—also called a backward edge—connects a vertex to one of its predecessors.
A back edge in a graph allows us to go from a descendant node back to an ancestor node.
In a graph, back edges are essential for identifying cycles. During a DFS traversal, the existence of a back edge signifies the existence of a cycle.
For example, from the above fig. DFS exploration tree for a graph, we can see that there is a edge from C to A, as A is the ancestor of C, this is considered as back edge.

Part (b): Execute Dijkstra's Algorithm on the graph using the source node A, showing the resulting spanning tree and distances. Does it work on this specific graph with negative edges? Why, or why not?

Solution (b):

N	A	B	C	D	E	F	G	H	I
1	0	inf	inf	5	inf	8	inf	inf	inf
2	0	inf	9	5	inf	8	inf	9	inf
3	0	inf	7	5	inf	8	inf	9	inf
4	0	inf	7	5	inf	8	inf	9	14
5	0	inf	7	5	inf	8	inf	9	14
6	0	inf	7	5	11	8	2	9	14
7	0	inf	7	5	11	8	2	9	0
8	0	inf	7	5	11	8	2	9	0
9	0	16	7	5	11	8	2	9	0

Fig a: Dijkstra's Algorithm with node A as the starting point.

Shortest distances from node A to all other nodes:
A: 0
B: 16
C: 7
D: 5
E: 11
F: 8
G: 2
H: 9
I: 0

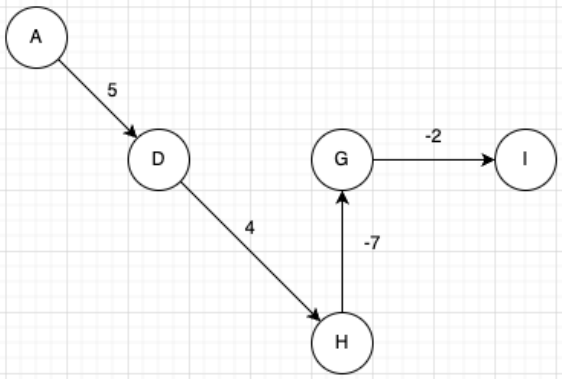

Fig b: Resulting Spanning tree with distances.

Figure 6: Dijkstra's Algorithm and it's resulting spanning tree.

Dijkstra's method takes a greedy approach, selecting options that are locally optimal at each stage. It makes the assumption that it will eventually identify the globally optimal path by choosing the minimum-weight edge at each iteration.

This greedy technique works effectively when all edge weights are non-negative because, once a node is "closed," or eliminated from consideration, the shortest path to that node is guaranteed to be minimal. The technique works under the assumption that the shortest path is the one that leads to a closed node.

So, in our case, we have the shortest path as given: $A \rightarrow D \rightarrow H \rightarrow G \rightarrow I$ with the shortest distance as 0. $(A \rightarrow D = 5, D \rightarrow H = 4, H \rightarrow G = -7, G \rightarrow I = -2)$.
As we can see algorithm chose the path from $H \rightarrow G = -7$ which has minimum cost over other edge $H \rightarrow E$.
We could have minimum cost path from $A \rightarrow D \rightarrow C \rightarrow I$, but it did not consider this path because it already found minimum cost edge path. So, it prematurely closes the path from vertex D.

Therefore, even though we are able to find the shortest path, its not a valid path because of negative cost edges.

Question 5: *Directed Weighted Graph*

Part (a): Execute the Bellman-Ford Algorithm on this graph, showing 9 iterations in a table like the one below. Does the graph have a shortest path solution?

Solution (a):

N	A	B	C	D	E	F	G	H	I
0	0	inf	inf	inf	inf	inf	inf	inf	inf
1	0	inf	4	5	11	8	2	9	inf
2	0	16	3	5	10	7	1	8	0
3	0	15	2	4	9	6	0	7	-1
4	0	14	1	3	8	5	-1	6	-2
5	0	13	0	2	7	4	-2	5	-3
6	0	12	-1	1	6	3	-3	4	-4
7	0	11	-2	0	5	2	-4	3	-5
8	0	10	-3	-1	4	1	-5	2	-6
9	0	9	-4	-2	3	0	-6	1	-7

Fig: Bellman-Ford Algorithm with node A as the starting point.

Figure 7: Bellman-Ford Algorithm

The Bellman-Ford algorithm is designed to find the shortest paths from a single source vertex to all other vertices in a weighted directed graph, even when some edges have negative weights.

Its processes $|V| - 1$ times (where |V| is the number of vertices) to find the minimum distance between each $u \rightarrow v$.

To find if there is any negative cycle, it again processes vertices to find minimum cost between $u \rightarrow v$, and if any distance between vertices changes, then we can say that there exists a negative weight cycle.

From the above fig. we have 9 vertices, and we tried to find the minimum cost between nodes for $|V| - 1 = (8)$ times. But for the 9th iteration distance between vertices changed(reduced).
This indicates that we have negative weight cycle. So, we cannot find the shortest path.

Part (b): What is different about this graph versus the prior graph?

Solution (b):
In Bellman-Ford Algorithm, the negative-edge weights are handled appropriately. While in Dijkstra's, if we try using negative-edge weights, it can give incorrect results for finding shortest path distance.
Also in Bellman-Ford, the algorithm stops the iteration once it encounters negative cycles during execution.

Reflection:
I acquired a deeper understanding of graphs, including directed and undirected graphs, as well as strongly connected components. Additionally, I gained insights into BFS search, DFS search, Djikstra's Algorithm, and the Bellman-Ford Algorithm.

Acknowledgements:
I would like to express my sincere gratitude to the following individuals and resources for their invaluable contributions to this assignment:
1) Prof. Bruce Maxwell: Thank you, Professor Bruce Maxwell for your guidance in this assignment. Your expertise in algorithms greatly influenced my work.
2) Classmates and TA's: I appreciate the discussions of my classmates, and TA's who clarified my doubts.
3) DPV Algorithms Textbook: This textbook was a useful resource for me to understand the basics of algorithms.
4) https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/: This website provided clear explanations for understanding DFS.
5) https://www.programiz.com/dsa/bellman-ford-algorithm: This website served as a useful resource to gain understanding of Bellman Ford Algorithm.
6) https://www.programiz.com/dsa/dijkstra-algorithm: This was a useful resource to understand Dijkstra's Algorithm.