HW 2

Intro Algorithms EN.601.433
Fall 2023
Due: Friday 9/29/2023 at 11:59 PM

Instructions

- This homework is worth 10% of your final grade.
- This homework is due Friday 9/29/2023 at 11:59 PM on Gradescope. Solutions will be posted on Canvas at midnight. No late homework will be accepted. To access Gradescope, click the Gradescope link on Canvas.
- You are **required** to type your homework. We will provide the LaTeX source code to the homework assignments, which you may optionally use as a template. If you install LaTeX on your computer, you can generate a PDF of your homework by running the command:

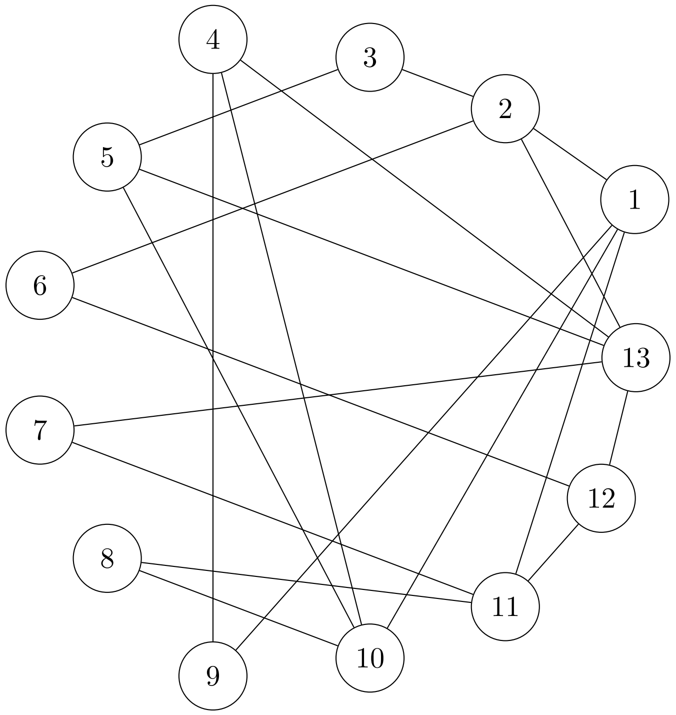
  ```
  latexmk -pdf my-homework.tex
  ```

  Alternately, you can use `https://www.overleaf.com/edu/jhu` as a web based LaTeX editor.
- You can collaborate with one other person. Collaboration should happen through verbal communication, scratch paper, whiteboards, etc. You are **not** allowed to copy homework solutions from another student. All students are required to submit their own write-up.
- You are **not** allowed to use any website to find solutions to these problems.
- Do not write your name on your homework. Instead, write your Johns Hopkins Student ID. This will allow us to grade your homework anonymously.
- For questions that require you to write an algorithm out, you can either use pseudo-code or English description.
- Make sure that you correctly assign which page a problem is located on when uploading to Gradescope.
- For problems that require graphs or diagrams, you are allowed to include an image of a drawing instead of typesetting the graph.[1]

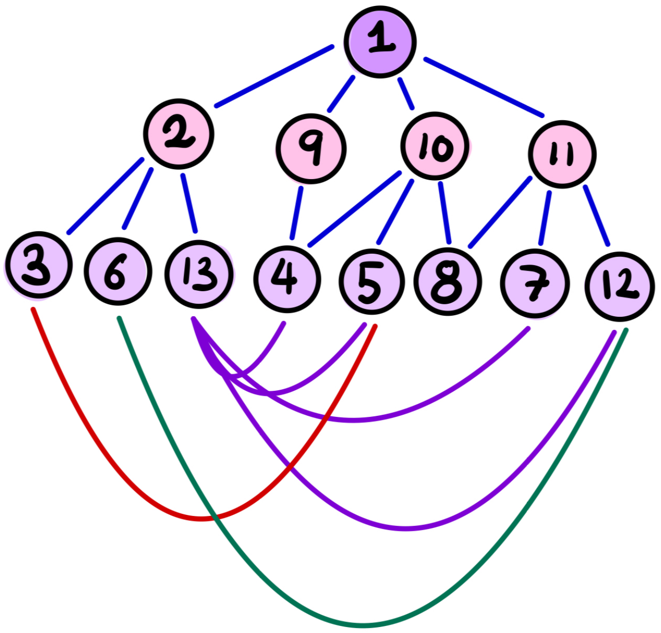
[1] If you are using LaTeX, you can follow the instructions here `https://www.overleaf.com/learn/latex/Inserting_Images` to embed an image in LaTeX.

1 Determine Bipartite using BFS (3 points)

Create a Breadth-First-Search (BFS) tree for the following graphs and deduce whether or not it is a bipartite graph.



1.1 Solution: Breadth-First Search Tree, with Bipartite Shading

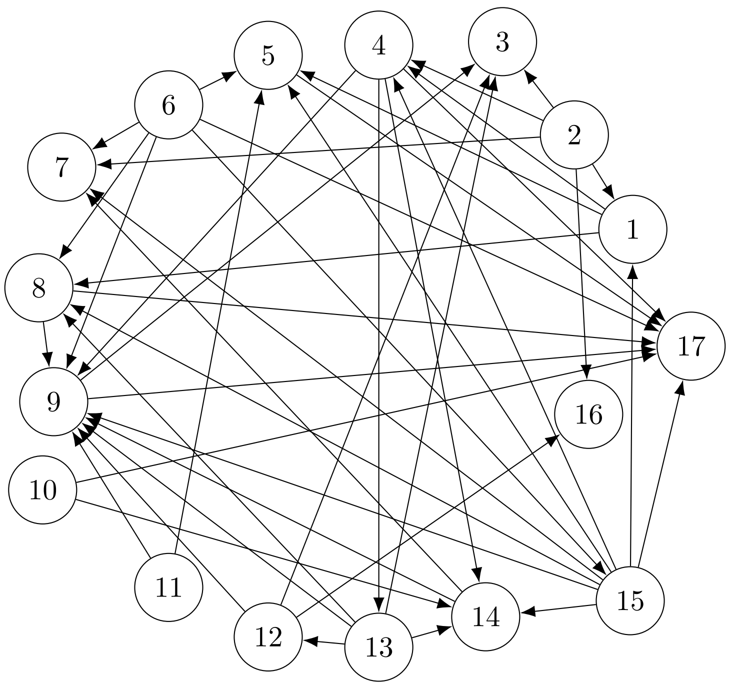


Above: The BFS Tree created using root node 1 with attempted bipartite shading

Because the tree above contains at least one odd cycle, for example, cycle containing nodes 1, 2, 3, 5, and 10, it is not bipartite.

2 Topological Ordering (2 Points)

Find a topological ordering of the Directed Acyclic Graph (DAG) shown below. Use numerical order with the smallest numbers first to break ties.



2.1 Solution:

The topological ordering is: 2, 6, 10, 11, 15, 1, 4, 5, 13, 8, 12, 14, 7, 9, 3, 16, 17

3 Detect a Cycle (5 Points)

Give an algorithm to detect whether a given undirected graph contains a cycle. If the graph contains a cycle, then your algorithm should output a cycle. (It should not output all the cycles in the graph, just one cycle.) The running time of your algorithm should be $O(m + n)$ for a graph with n nodes and m edges.

3.1 Algorithm Solution

Pick a vertex V as a root and perform a Depth-First Search Algorithm, maintaining a tree T. During the creation of T during the Depth-First Search Algorithm, at each point, starting from root V, when we encounter an edge to a vertex we have not yet encountered, the new vertex is added to the tree, along with the edge from the parent to the vertex. When we encounter an edge that leads to a vertex we have seen before, we know that there is a cycle. Once we know that there is a cycle, we can return the cycle by tracing the path in the DFS tree backwards until the cycle is complete and displaying it in the correct order, taking linear time. If the Depth-First

Search Algorithm completes without finding a repeat edge, that means there is no cycle. Each DFS Algorithm takes linear time as well, since each edge and vertex is only visited once.

4 All paths go through r_{ome} (8 points)

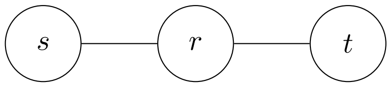
There's a natural intuition that two nodes that are far apart in a communication network—separated by many hops have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.

Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $\frac{n}{2}$. Either prove or disprove there exists a node r which is not equal to s or t, such that deleting r from G will destroy all paths between s and t.

Your solution should include both:

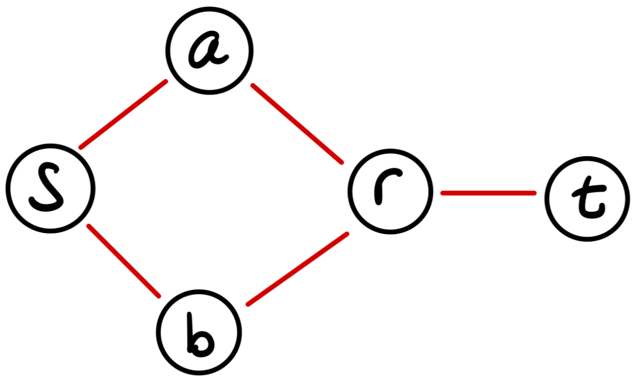
- Draw a n-node graph (for $n \geq 4$) where the shortest path between s and t is strictly greater than $\frac{n}{2}$. You should use this part of the problem to develop an intuitive understanding of the graph's properties. You are not required to type this part of your solution, you can submit a photo of a hand-drawn graph.[2]

  To get you started, here is an example 3-node graph containing nodes s, t, and r as described above.

- Construct a proof or counterexample that exist a node r for all graphs G where distance between s and t is strictly greater than $\frac{n}{2}$ and that removing r will destroy all paths between s and t.

4.1 Example Graph:



Shown above is a graph with 5 nodes, where the shortest path between s and t are greater than 5/2 (3).

[2]If you are using LaTeX, you can follow the instructions here `https://www.overleaf.com/learn/latex/Inserting_Images` to embed an image in LaTeX.

4.2 Solution Proof:

Let's use proof by contradiction. Assume there does not exist a node r in the graph G containing n nodes such that removing r destroys all paths between s and t. From the claim above, the distance between s and t is greater than $\frac{n}{2}$, which means that there are at least $\frac{n}{2}$ nodes between nodes s and t.

For the assumption to hold true that node r does not exist, there must be at least 2 nodes in every level so that more than one path exists to t in every level. If we multiply $\frac{n}{2} * 2$, this gives us a total of n nodes total in the layers between s and t. Since we have not yet accounted for s or t. This means that there are $n + 2$ nodes total in the entire graph.

Therefore, we must have a contradiction, since we stated that G contains n nodes. Therefore, there can't be at least 2 nodes in every layer, which means that there exists at least one layer that only consists of one node which we can call node r. Deleting r will delete all paths from s to t because there are no other nodes int eh alyer that r resides in so all s - t paths are broken. Therefore, we have proven that such a node r from the claim above exists.

5 Some Short Paths in a Graph (8 points)

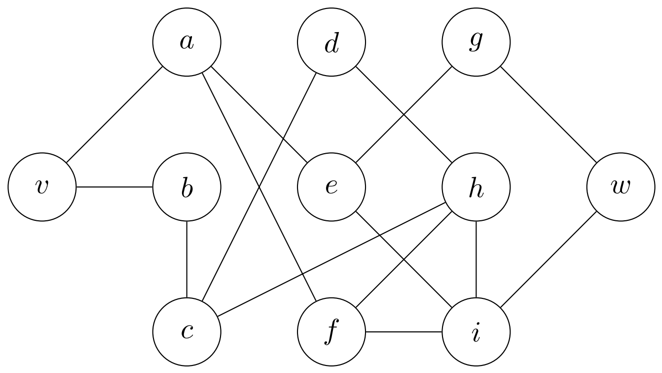
A number of art museums around the country have been featuring work by an artist named Mark Lombardi (1951–2000), consisting of a set of intricately rendered graphs. Building on a great deal of research, these graphs encode the relationships among people involved in major political scandals over the past several decades: the nodes correspond to participants, and each edge indicates some type of relationship between a pair of participants. And so, if you peer closely enough at the drawings, you can trace out ominous-looking paths from a high-ranking U.S. government official, to a former business partner, to a bank in Switzerland, to a shadowy arms dealer.

Such pictures form striking examples of social networks, which, as we discussed in book chapter 3.1, have nodes representing people and organizations, and edges representing relationships of various kinds. And the short paths that abound in these networks have attracted considerable attention recently, as people ponder what they mean. In the case of Mark Lombardi's graphs, they hint at the short set of steps that can carry you from the reputable to the disreputable.

Of course, a single, spurious short path between nodes v and w in such a network may be more coincidental than anything else; a large number of short paths between v and w can be much more convincing. So in addition to the problem of computing a single shortest v-w path in a graph G, social networks researchers have looked at the problem of determining the number of shortest v-w paths.

This turns out to be a problem that can be solved efficiently. Suppose we are given an undirected graph $G = (V, E)$, and we identify two nodes v and w in G. Give an algorithm that computes the number of shortest v-w paths in G. (The algorithm should not list all the paths; just the number suffices.) The running time of your algorithm should be $O(m + n)$ for a graph with n nodes and m edges.

Demonstrate your algorithm working on the graph given below.



5.1 Solution: Algorithm to Calculate the Number of Shortest Paths between Two Vertices

```
Mark all nodes and edges in graph $G$ UNVISITED except starting node $v$;
Introduce 2 arrays, shortestDistance[] which contains the length of the shortest path from the starting node to the current node, and shortestPaths[] which contains the number of shortest paths from the starting node to the current node. Set every shortestDistance[] to infinity, and set every shortestPaths[] to 0. Then, set shortestPaths[] of the root node as 1, and shortestDistance[] as 0.
WHILE there exists an unvisited node $Z$ {
    FOR every neighbor node of the current node and every neighbor node after that {
        IF the node is unvisited, mark as visited;
        IF (shortestDistance[neighbor node] > shortestDistance[current node] + 1) {
            shortestDistance[neighbor node] = shortestDistance[current node] + 1;
            shortestPaths[neighbor node] = shortestPaths[current node];
        }
        IF (shortestDistance[neighbor node] = shortestDistance[current node] + 1) {
            shortestPaths[neighbor node] += shortestPaths[current node];
        }
    }
}
```

5.2 Solution: Algorithm Performed on Graph

Suppose we want to find the number of shortest paths from vertex v to w in the above graph. The algorithm works as follows:

Consider source node v. shortestPaths[v] = 1, shortestDistance[v] = 0.
Go to node a. Mark visited. shortestDistance[a] = 1. shortestPaths[a] = 1.
Go to node b. Mark visited. shortestDistance[b] = 1. shortestPaths[b] = 1.
Go to node e. Mark visited. shortestDistance[e] = 2. shortestPaths[e] = 1.
Go to node f. Mark visited. shortestDistance[f] = 2. shortestPaths[f] = 1.
Go to node c. Mark visited. shortestDistance[c] = 2. shortestPaths[c] = 1.
Go to node g. Mark visited. shortestDistance[g] = 3. shortestPaths[g] = 1.
Go to node i. Mark visited. shortestDistance[i] = 3. shortestPaths[i] = 1.
Go to node h. Mark visited. shortestDistance[h] = 3. shortestPaths[h] = 1.
Go to node d. Mark visited. shortestDistance[d] = 3. shortestPaths[d] = 1.
Go to node h. shortestDistance[h] = 3. shortestPaths[h] = 2.
Go to node i. shortestDistance[i] = 3. shortestPaths[i] = 2.

Go to node w. shortestDistance[w] = 4. shortestPaths[w] = 1.
Go to node w. shortestDistance[w] = 4. shortestPaths[w] = 2.
Go to node i. shortestDistance[i] = 3. shortestPaths[i] = 2.
Go to node w. shortestDistance[w] = 4. shortestPaths[w] = 3.
Go to node h. shortestDistance[h] = 3. shortestPaths[h] = 2.
Go to node i. shortestDistance[i] = 3. shortestPaths[i] = 2.
Go to node w. shortestDistance[w] = 4. shortestPaths[w] = 3.
Return shortestPaths[w], which is 3. There are 3 shortest paths from v - w.

6 Wireless Network Connectivity (5 points)

Some friends of yours work on wireless networks, and they're currently studying the properties of a network of n mobile devices. As the devices move around (actually, as their human owners move around), they define a graph at any point in time as follows: there is a node representing each of the n devices, and there is an edge between device i and device j if the physical locations of i and j are no more than 500 meters apart. (If so, we say that i and j are "in range" of each other.)

They'd like it to be the case that the network of devices is connected at all times, and so they've constrained the motion of the devices to satisfy the following property: at all times, each device i is within 500 meters of at least $n/2$ of the other devices. (We'll assume n is an even number.) What they'd like to know is: Does this property by itself guarantee that the network will remain connected?

Here's a concrete way to formulate the question as a claim about graphs.

Claim: Let G be a graph on n nodes, where n is an even number. If every node of G has degree at least $n/2$, then G is connected.

Decide whether you think the claim is true or false, and give a proof of either the claim or its negation.

6.1 Solution: Proof by Contradiction

Proof: We will consider a graph G with all the mentioned properties in the problem above. Let's assume, for the sake of contradiction, that graph G is not connected. In order for G to be not connected, there has to be at least one node in G that is isolated from the rest of G.

However, according to the claim above, every node has to have edges to at least $\frac{n}{2}$ other nodes. This means that these $(\frac{n}{2} + 1)$ nodes form a connected section of the graph that is disconnected from the other components. Let's call this section S.

If S is a connected section of the graph and contains $(\frac{n}{2} + 1)$ nodes, the other connected section of the graph, which we will call S', must contain $n - (\frac{n}{2} + 1) = \frac{n}{2} - 1$ nodes.

However, these $\frac{n}{2} - 1$ nodes cannot from a separate connected component S' among themselves, since every node in G must have edges to $\frac{n}{2}$ other nodes. Because of this, every node in S' should have at least one edge to a node in S. Therefore, we have a contradiction.

Every node in S' must be connected to at least one node in S. Therefore, A and B are connected, which contradicts our original assumption that G is disconnected.

Conclusion: If G follows the given conditions, G must be connected.