MATH 416, PROBLEM SET 5

Comments about homework.

- Solutions to homework should be written clearly, with justification, in complete sentences. Your solution should resemble something you'd write to teach another student in the class how to solve the problem.
- You are encouraged to work with other 416 students on the homework, but solutions must be written independently. Include a list of your collaborators at the top of your homework.
- You should submit your homework on Gradescope, indicating to Gradescope where the various pieces of your solutions are. The easiest (and recommended) way to do this is to start a new page for each problem.
- Attempting and struggling with problems is **critical** to learning mathematics. Do not search for published solutions to problems. I don't have to tell you that doing so constitutes academic dishonesty; it's also a terrible way to get better at math.

  If you get stuck, ask someone else for a hint. Better yet, go for a walk.

Warning. These are not necessarily *model solutions*; they are meant to help you understand the problems you didn't totally solve and maybe to give you alternative solutions. Sometimes I will give less or more detail here than I would expect from you.

Problem 1.

(a) Prove the Handshake Lemma: in a graph, the sum of the degrees is twice the number of edges.
(b) Deduce that every graph has an even number of odd-degree vertices.
(c) Prove that if a graph G has exactly 2 vertices of odd degree, then there must be a path from one to the other.

Solution. (a) There are many proofs. The easiest is by induction on the number of edges, but the best is the following direct combinatorial argument.

Count the pairs (e, v) where e is an edge incident to v. Grouping these pairs by edges, we count $2|E|$. Grouping them by vertices, we count $\sum_{v \in V} \deg(v)$.

(b) This follows immediately from part (a). The sum of the degrees is an even number, so the number of odd terms must be even.

(c) Suppose that x and y are vertices of odd degree. If there is no path from x to y, then x and y lie in different connected components, say C and D. Now C induces a subgraph with at least one vertex of odd degree; by

part (b) it must have another. So we've identified at least three vertices of odd degree. □

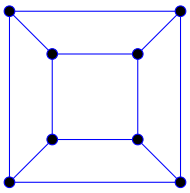
(The argument in part (c) uses the fact that the degree of a vertex in G is the same as its degree in the subgraph induced by its connected component.)

Problem 2.

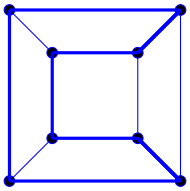
(a) Give an example of a connected graph with 8 vertices in which every vertex has degree 3. What is the length of the shortest cycle in your graph? the longest?

(b) Give an example of a connected graph with 10 vertices in which every vertex has degree 3. What is the length of the shortest cycle in your graph? the longest?

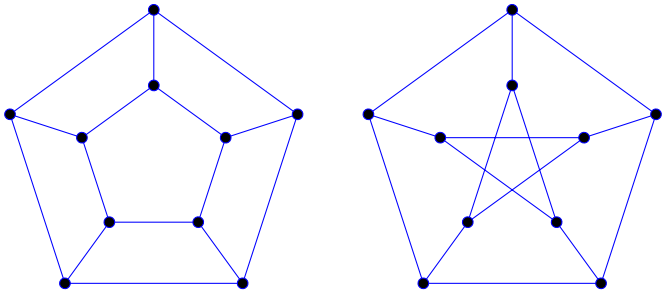
Solution. (a) Here's an example.



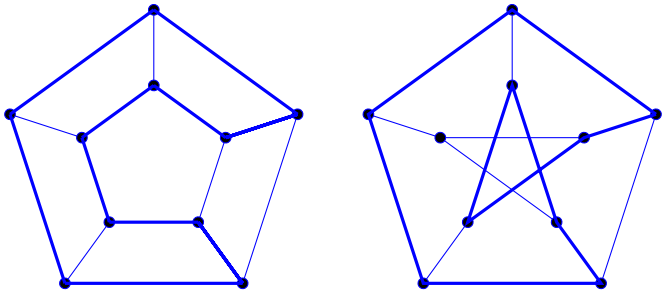
The shortest cycle has length 4. The longest has length 8:



(b) Here are two examples: the one on the right is called the Petersen Graph.



The shortest cycle in each graph has length 5. The longest has length 10 in the graph on the left and 9 in the graph on the right:



Problem 3. Consider the problem of taking a graph G and two vertices $s, t \in G$ and determining the number of shortest paths from s to t in G. Give a linear-time algorithm that solves this problem.

(*Hint:* Adapt BFS. Your algorithm should run in linear time if the graph is represented by an adjacency list.)

Solution. The solution I originally had in mind folds the computation into BFS, but as two of you pointed out to me, it is easier to run BFS first and then compute the number of shortest paths level-by-level.

We will solve the apparently more difficult problem of computing the number of shortest paths from s to *every* vertex in the graph, not just the t we're given. We will write $\text{count}(x)$ for the number of shortest paths from s to a vertex x. The following lemma is the main point.

Lemma 1. $(s, s_1, \ldots, s_n, y)$ is a shortest path from s to y iff $(s, s_1, \ldots, s_n)$ is a shortest path to s_n and s_n is a neighbor of y.

As a consequence, we have the following Claim.

Claim. If $t \in L_{k+1}$, then $\text{count}(t) = \sum_x \text{count}(x)$, where the sum is over all $x \in L_k$ adjacent to t.

Now the algorithm writes itself. Start by running BFS so that we have access to the layers L_k of the BFS tree. Notice that it considers each edge only once, so it runs in linear time.

Algorithm 1: Number of shortest paths

```
1  foreach v in V do
2      set count(v) to 0 ;                    // initialize counts
3  set count(s) = 1 ;
4  set k = 1 ;
5  while L_k is nonempty do
6      foreach y in L_k do
7          foreach x adjacent to y do
8              if x ∈ L_{k-1} then
9                  count(y) += count(x) ;
10     k++ ;
11 return count(t)
```

Problem 4. Prove that the following are equivalent for a graph $G = (V, E)$:

(a) G is a *tree*, i.e., a connected acyclic graph;
(b) G is connected and $|E| = |V| - 1$;
(c) G is acyclic and $|E| = |V| - 1$;
(d) for any vertices $x, y \in V$ there is a unique path from x to y.
(e) G has no cycles but for any $x, y \in V$ with $e = \{x, y\} \notin E$ the graph $G + e := (V, E \cup \{e\})$ has a cycle.

Solution. We first prove (a) $\Leftrightarrow$ (d).

Suppose first that G is connected but (d) fails, so that there are vertices x, y in G with two different paths from x to y. Let

$$p_0 = x, p_1, \ldots, p_{n-1}, p_n = y \quad \text{and} \quad q_0 = x, q_1, \ldots, q_{m-1}, q_m = y$$

be two different paths from x to y. Let $k \geq 1$ be least such that $p_k \neq q_k$. Let $l, l' \geq k$ be least such that $p_l = q_{l'}$. Then

$$p_{k-1}, p_k, \ldots, p_l = q_{l'}, q_{l'-1}, \ldots, q_{k-1} = p_{k-1}$$

is a cycle in G.

For the converse, suppose that there is a cycle

$$x_0, x_1, \ldots, x_l = x_0$$

in G. Since cycles cannot repeat vertices (and $l \geq 2$), there are two different paths x_0, x_1 and $x_0, x_l, x_{l-1}, \ldots, x_2, x_1$ from x_0 to x_1.

Now we prove

Lemma 2.

(i) Every tree with n vertices has exactly $n - 1$ edges.
(ii) Every acyclic graph with k connected components and n vertices has exactly $n - k$ edges.
(iii) Every acyclic graph with n vertices has $\leq n - 1$ edges.

Proof of Lemma **??**. For (i), one could use Problem 5 and induction on n. But better to construct a bijection directly, as follows. Choose a root! Let r be any vertex of T. Since we showed (a) iff (d), we know that for each vertex x there is a unique path from x to r. The *parent* of a vertex $x \neq r$ is the unique vertex x^* that is first (after x) on the unique path from x to r. Every edge is incident to one parent and its child. So there is a bijection from the set of edges to the set of non-root vertices, given by sending an edge to the child vertex to which it is incident. This proves that the number of edges is $n - 1$.

For (ii), suppose that G is an acyclic graph with k connected components. Say that the connected components of G have vertex counts $n_1, \ldots, n_k$, respectively. Each connected component is a tree, so the j^{th} one has $n_j - 1$ vertices, by part (i). Every edge is part of a connected component, so in total there are

$$(n_1 - 1) + (n_2 - 1) + \cdots + (n_k - 1) = n - k$$

edges.

(iii) follows easily from (ii). □

Lemma 3.

(i) If G is connected and the edge e is part of a cycle in G, then $G - e$ is connected.
(ii) If G is a connected graph with n vertices, then G has $\geq n - 1$ edges.
(iii) If G is a connected graph with n vertices and a cycle, then G has $\geq n$ edges.

Proof of Lemma **??**. Suppose that $x_0, x_1, \ldots, x_{l-1}, x_l = x_0$ is a cycle in G and $e = \{x_k, x_{k+1}\}$. Now let v and w be any vertices of G. There is a path from v to w. If the path doesn't use e, then we're done. If it does use e, then replace e with the path $x_{k+1}, x_{k+2}, \ldots, x_{l-1}, x_0, \ldots, x_k$. So v and w are still connected in $G - e$. This proves (i).

Suppose that G is a connected graph with n vertices. If G has no cycles, then by Lemma **??**(i) G has $n - 1$ edges.

Suppose that G has a cycle. Remove an edge from this cycle. Continue removing edges from cycles in this way until no cycles remain. By part (i) of the present lemma, the graph obtained at the end is acyclic and is connected. But it still has n vertices, so by Lemma **??**(i), it has $n - 1$ edges. So there must have been $\geq n$ edges to start with. □

Now we're ready. We have (a) ⇒ (b) by Lemma **??**(i).

(b) ⇒ (c) follows from Lemma **??**(iii).

(c) ⇒ (e): Assume (c) and let $e = \{x, y\} \notin E$. Then $G + e$ is a graph with $|V| = |E|$, so Lemma **??**(iii) implies that $G + e$ has a cycle.

(e) ⇒ (a): Suppose that (e) holds, so that G is acyclic. Suppose toward a contradiction that G is disconnected. Let x and y be vertices of G in different connected components, say C and D, so that in particular $\{x, y\}$ is not an edge of G. By assumption, $C + D + \{x, y\}$ has a cycle. But now we apply Lemma **??**(i) to $C + D + \{x, y\}$ and $e = \{x, y\}$ to see that $C + D$ is connected, contradicting the fact that they are different connected components of G. □

Problem 5. Show that every tree with at least 2 vertices must have a *leaf*, i.e., a vertex of degree 1.

Solution. There are many proofs. Here's one that uses the Handshake Lemma and the edge count for trees. Suppose that T is a tree with n vertices. By Problem 4, T has exactly $n-1$ edges. Look at the sum given by the Handshake Lemma:

$$\sum_v \deg(v) = 2|E| = 2n - 2.$$

Each of the terms $\deg(v)$ is ≥ 1 since T is connected, and not all of them can be ≥ 2, since then the sum is $2n$. So one of them must be 1. □

Other options:

- Choose a root and look at a vertex on the maximum level.
- Look at one end of a path of maximum possible length in the tree.

Problem 6. Let G be a connected graph.

(a) Prove that if any edge e in G is part of a cycle, then removing e does not disconnect G.

(b) Using part (a), briefly describing an algorithm for finding a spanning tree of G that starts with the set of edges E and deletes edges until what remains is the edges of a spanning tree.

Solution. See the proof of Lemma **??**(i) of Problem 4.

Problem 7. Let G be a strongly connected digraph and let T be the DFS tree obtained from a particular ordering of the vertices of G. Prove that, if all the forward edges (in the sense of the DFS tree T) are removed from G, then the resulting graph is still strongly connected.

Solution. If (x, y) is a forward edge (in the sense of T), then x is a non-child ancestor of y in T. This means that there is a path $x, x_1, \ldots, x_n, y$ with all edges in T. Thus, any path in G that uses the forward edge (x, y) can be modified not to use the forward edge (x, y) by replacing (x, y) with $(x, x_1, \ldots, x_n, y)$. All of the new edges are tree edges, so not forward edges, so in this way any path in G can be modified to include only non-forward edges.

In conclusion, if there is a path in G from vertex u to vertex v, then there is a path from u to v in G^*, the graph obtained by removing all forward edges. So if G is strongly connected, then so is G^*.

Problem 8. You just discovered your best friend from elementary school on Twitbook. You both want to meet as soon as possible, but you live in two different cites that are far apart. To minimize travel time, you agree to meet at an intermediate city, and then you simultaneously hop in your cars and start driving toward each other. But where exactly should you meet?

You are given a weighted graph $G = (V, E)$, where the vertices V represent cities and the edges E represent roads that directly connect cities. Each edge e has a weight $w(e)$ equal to the time required to travel between the two cities. You are also given a vertex p, representing your starting location, and a vertex q, representing your friend's starting location. Describe and analyze an algorithm to find the target vertex t that allows you and your friend to meet as quickly as possible.

Solution. The goal is to minimize the time it takes for you to meet. If we carry out Dijkstra's algorithm at the start point s and t we get two distance functions the distance function

$$d_s \colon V \to \mathbb{N} \text{ and } d_t \colon V \to \mathbb{N}$$

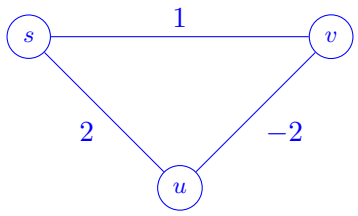
that encode the distance from s and t respectively. So it suffices to sort V by $\max(d(s), d(t))$ and find the minimum. The max function tells you the total time it takes for both to arrive at a given city. Minimizing it tells you the city that you should drive to in order to meet as quickly as possible.

Problem 9.

(a) Give an example of a weighted graph with negative weights and a source vertex s on which Dijkstra's algorithm does not produce the minimum-weight path from s to some vertex v. In your example, there should be no negative-weight cycles (so that there *is* a min-weight path from s to v).

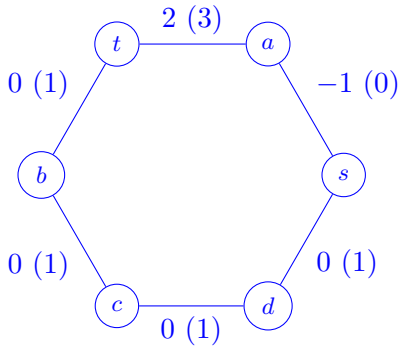
(b) Given a weighted graph, possibly with negative weights, you might try to add a sufficiently large number to all the weights so that they all become nonnegative and then run Dijkstra's algorithm on the modified graph. Does this successfully find min-weight paths (in the original graph)? Either prove that it works or give a counterexample.

Solution. (a)



The unique cycle in this example has weight 1, so there are no negative-weight cycles. Dijkstra's Algorithm will start at s, visit v and update $\text{dist}(v) = 1$, and then visit u and update $\text{dist}(u) = 1 - 2 = -1$. Then it will stop. But there is a weight-0 path from s to v, so Dijkstra's Algorithm does not produce the correct output.

(b) Doesn't work. The problem is that the same value is added to all edges, so that the weights of paths with more edges change more than the weights of paths with fewer edges. Here is an example.



(The modified weights are in blue.) The only cycle in this graph has weight $-1 + 2 + 0 = 1$, so there are no negative-weight cycles.

The min-weight path from s to t in the modified graph is s–a–t (which has weight $3 < 4$), while min-weight path from s to t in the original graph is s–d–c–b–d (which has weight $0 < 1$ in the original graph). It really has

nothing to do with Dijkstra's Algorithm; it's just that adding a constant to all edge-weights can change which path has minimal weight. □