CS5800 Algorithms

Recitations from Sep 30, 2019 to Oct 4, 2019

Ravi Sundaram

Problem 1 (Graphs)

1. What's the minimal and maximal number of edges an undirected connected graph can have? (connected means there's a path between any two vertices in the graph).
2. Suppose we have an undirected graph $G = (V, E)$. The degree of a vertex v (that we denote by $deg(v)$) is the number of edges connected to the vertex. What is the sum $\sum_{v \in V} deg(v)$ equal to?
3. Use the last part to show that the number of vertices that have odd degree is even.
4. An Eulerian cycle is a path that begins at some vertex s and also ends at s, and the path visits all the edges in the graph. Prove that a connected graph has an Eulerian cycle if and only if every vertex in the graph has even degree.

Solution:

1. $n - 1$ is the fewest, $\binom{n}{2}$ is the most.
2. Each edge is connected to 2 vertices, so it adds 2 degree. So the sum of degrees will be $2|E|$ (where $|E|$ means the number of edges in the graph).
3. If the number of vertices with odd degree is odd, then the total degree will be a sum of an odd number of odd numbers with an even number of even numbers which results in an odd number, which is impossible since by the last part we know the total degree is always even.
4. First suppose the graph G has an Eulerian cycle. Then let's follow this hypothetical Eulerian cycle. Every time we visit a vertex, we have to leave it (to return to the initial vertex). So each time we visit a vertex, we enter and then exits which adds degree 2. So every vertex will have even degree.

   Now for the other direction, suppose every vertex has even degree. We show that there's an Eulerian cycle constructively (we construct it through an algorithm). Pick an arbitrary vertex v. Follow a random path starting from v, and after we traverse through an edge, we delete the edge from the graph. First note that we will never get stuck on a vertex that's not v. Since the degree of each vertex is even, after we visit a vertex and leave, the degree will still be even. It might become 0, but then we won't ever visit that vertex again. The only way to get stuck if we visit a vertex with degree 1, but that's impossible. But now that we returned to v, there still might be edges remaining in the graph. Since the graph is connected, at least one of the vertices that still has edges remaining will have been visited. So pick one of these vertices and repeat the process. And the path we take we can insert into the original path. And we continue this until there are no more edges left.

As an example, consider $G = (V, E)$, where $V = \{a, b, c, d, e\}$ and $E = \{(a, c), (a, d), (b, c), (b, d), (c, d), (c, e), (e, d)\}$. Say our first path we start from a, and we follow the path $(a, c), (c, d), (a, d)$ (so $a \to c \to d \to a$). Now, we are left with the edges $E = \{(b, c), (b, d), (c, e), (e, d)\}$. We can start from c, and go $c \to e \to d \to b \to c$. Now, we can insert that into the middle of the first path, so our final path looks like $a \to (c \to e \to d \to b \to c) \to d \to a$.

...

■

Problem 2 (Data Stuctures + Graph Traversal)

1. Say we have a queue Q that is empty initially. We run the following sequence of operations. $Q.push(5), Q.push(2), Q.push(4), Q.pop(), Q.push(6)$. What does the queue look like now?
2. Now say we have a stack S that is empty initially. We run the following sequence of operations. $S.push(5), S.push(2), S.push(4), S.pop(), S.push(6)$. What does the stack look like now?
3. Show how you can construct a queue using two stacks as the underlying data structures.
4. Say you are given the graph $G = (V, E)$ where $V = \{a, b, c, d, e, f\}$, $E = \{(a, b), (a, c), (a, d), (b, e), (c, f), (d, f), (e, f)\}$. We are going to run through a BFS traversal of this graph together as a class. We put vertices in the queue based on their order in the alphabet.
5. Say you are given the graph $G = (V, E)$ where $V = \{a, b, c, d, e, f\}$, $E = \{(a, b), (a, c), (a, d), (b, e), (c, f), (d, f), (e, f)\}$. Now run through a DFS traversal individually. Write the sequence of nodes visited. We put vertices in the stack based on their order in the alphabet.

Solution:

1. (Front of queue) 6,4,2
2. (Bottom of stack) 5,2,6
3. We have two stacks, let's call one the in stack and the other the out stack. Whenever $queue.push(v)$ is called, we do $in.push(v)$. Whenever $queue.dequeue()$ is called, we try $out.pop()$. If out is empty, then we pop all the elements from in and insert them into out (the ordering of the elements will get flipped by this). Then we return $out.pop()$.
4. For BFS, we start at a, then we visit b, c, d. Then we pop b from the queue, and we visit e. Then we we pop c from the queue and we visit f. Then we pop d from the queue (nothing left to visit/ add to queue), then we pop e, then we pop f (nothing to visit).
5. We add b, c, d to stack, then we pop d and label it as visited. Then we add f to stack, pop it and label it as visited. Then we add c, e to stack, then pop e and label it as visited. Then add b to stack, then pop it and label it as visited. Then we pop c, label it as visited. Then we pop c, then b, and then the stack is empty and we're done.

...

■

Problem 3 (Shortest Path Variants)

1. You are given a directed graph $G = (V, E)$, where each edge has a weight $w(u, v) > 1$, and two vertices s and t. You want to find the shortest path between s and t, but now the weight of the shortest path is defined as the product of all the edges in path.

2. You are given a directed graph $G = (V, E)$, where each edge has a weight $w(u, v) > 0$, and two vertices s and t. You want to find the shortest path between s and t, but if there are multiple shortest paths, you want to return the one with the fewest number of edges. Hint: Think of how you can modify G.

Solution:

1. Remember, $\log(ab) = \log(a) + \log(b)$, so we can modify all the edge weights by taking the log and then running Dijkstra's.

2. If the edge weights can be real, then this problem is actually rather tricky. So we'll assume all the edge weights are integers. Then we can we solve this in a couple ways. One, we can add $1/(2|E|)$ to each edge weight, and then run Dijkstra's. Then note, that a path that was longer than the shortest path in the orginal graph can never become shorter than the shortest path in the new graph, since our increment will only add at most $1/2$ to the modified path weight, and we know the initial difference had to be at least one. And if two paths were tied, the one with less edges will have less increments added, so it will be shorter.
   A second way to solve this is by increasing the weights. We can take each edge weight w_i and set it to $w_i(|E| + 1) + 1$ in the new graph. Then note that if a path between s and t had original cost P_i, then the new cost will be $P_i(|E| + 1) + |P_i|$, (where $|P_i|$ denotes the number of edges in the path). Similar idea as the first solution, a path that's longer in the original path will never become shorter in the modified graph, since there must be a difference of at least one, and $|E| + 1 > |P_i|$. And the length of the path will be the tiebreaker between two paths that were initially the same weight.

...

■

Problem 4 (Extras...)

1. Prove by contradiction that if there's an inversion in an array, then there is an inversion between neighboring elements a_i and a_{i+1}. That is, $a_i > a_j$ for $i < j \neq i + 1$ implies that $a_k > a_{k+1}$ for some $i \leq k \leq j$.

2. (Taken from Jeff Erickson's CS 374 course) A party of n people have come to dine at a fancy restaurant and each person has ordered a different item from the menu. Let $D_1, D_2, ..., D_n$ be the items ordered by the diners. Since this is a fancy place, each item is prepared in a two-stage process. First the head chef (there is only one head chef) spends a few minutes on each item to take care of the essential aspects and then hands it over to one of the many sous-chefs to finish it off. Assume that there are an infinite number of sous-chefs who can work in parallel on the items oncce the head chef is done. Each dish D_i take h_i units of time for the head chef followed by s_i

units of time for the sous-chef (the sous-chefs are identical). The diners want all their items served at the same time which means that the last item to be finished defines when they can be served. The goal of the restaurant is to serve the diners as early as possible. Consider the following greedy algorithms that order the items according to different criteria. For each of them either describe a counter example that shows that the order does not yield an optimum solution or give a proof that the ordering yields an optimum solution for all instances.

(a) Order items in increasing order of $h_i + s_i$.

(b) Order items in decreasing order of $h_i + s_i$.

(c) Order irems in increasing order of h_i.

(d) Order items in decreasing order of h_i.

(e) Order items in increasing order of s_i.

(f) Order items in decreasing order of s_i.

3. Let G be a connected undirected graph. Suppose we start with two coins on two arbitrarily chosen vertices of G. At every step, each coin must move to adjacent vertex. Describe and analyze an algorithm to compute the minimum number of steps to reach a configuration where both coins are on the same vertex or to report correctly that no such configuration is reachable. The input to your algorithm consists of a graph $G = (V, E)$ and three vertices $u, v, t \in V$ (which may or may not be distinct).

4. Given a directed graph $G = (V, E)$, where each edge (u, v) has a weight $w(u, v)$, two vertices s and t in G and an integer c. Suppose you are allowed to set the weight of c edges in G to 0. Design an algorithm to compute the shortest path from s to t. Hint: Don't modify Dijkstra's, but instead think of how you can modify the original graph.