HW4 (Shortest Path and Maximum Flow)

Name: Isaac Boaz (Solo)

May 29, 2024

Q1: You are thinking of buying a new electric car, but like many potential buyers you are worried about the range of the car's battery. A given car, fully charged, can travel D miles before its battery runs out. You live in Bellingham and know the road network along with the distances between cities. In your network, you only include cities with charging stations, so every time you visit a city you know you can recharge your battery.

- Show how to determine in linear time whether there is a feasible route from Bellingham to any other city for a car with range D. Keep in mind you can never travel between two cities that are more than D miles, since your car would grind to a halt. (3 points)

  Let's say T is the target city we are trying to reach from Bellingham, and that we set our starting city to be Bellingham. For each neighboring city v of our current city, we can check if the distance between the current city and v is less than or equal to D. We'll make a list of all of these cities, and pick one at random. From that city, we will repeat the above process, making sure to not visit cities that we have already been to. If we ever reach the target city, then we know that there is a feasible route.

  This algorithm will run at worst-case linear time in terms of the number of cities, as we will visit a maximum of V cities.

- You have a list of potential electric cars to buy. Give an $O((V + E) \log V)$ algorithm to determine the minimum fuel tank capacity you need to get from Bellingham to a particular city. (5 points)

  We can use Dijkstra's algorithm (with a min-heap) to find the shortest path from Bellingham to the target city. Once we have the shortest path, we can find the maximum distance between any two cities in the path, which will only take $O(V)$ time.

Q2: You are given a set of currency exchange rates among multiple currencies. The goal is to determine if there is an arbitrage opportunity – a situation where you

can start with a certain amount of one currency, follow a sequence of exchanges, and end up with more of the same currency than you started with. Please explain how you can possibly use the shortest path algorithm to detect arbitrage opportunities. (6 points)

We can imagine each currency as a node, and the weighted edges as the ratio required to convert from one currency to another. To clarify, this would mean that if \$1 USD = 0.85 EUR, then the edge from USD to EUR would have a weight of 0.85, and the edge from EUR to USD would have a weight of $\frac{1}{0.85} \approx 1.176$. Now, if we start from one currency and find the shortest path to itself, we would expect (in a non-arbitrage scenario) that the shortest path's distance would be 1 or less.

If there was a path whos distance was greather than 1, then we would have found an arbitrage opportunity. This is because we could start with 1 unit of the currency, convert it to another currency, convert it back to the original currency, and end up with more than 1 unit of the original currency.

Note that this algorithm doesn't entirely work, since the *shortest path* might ignore arbitrage opportunities. Thus, we can actually flip the weights of the edges to be negative, and then run the shortest-path algorithm (assuming it supports negative weights). While we will not find the shortest path, we will know if a negative cycle exists, which would indicate an arbitrage opportunity.

Q3: A flood happens and farmers in the area find their fields are quickly becoming inundated. They need to move their cows to higher ground that isn't too far away. There are n farms which will be flooded each containing c_i cows for $i = 1, \ldots, n$. Luckily, there are k neighboring farms willing and able to house their cows. Each cow needs to be herded to a safe farm within a 2 hour cow-walk away. However, each safe farm has to keep space for their own cows, so the safe farms $j = 1, \ldots, k$ only have a capacity to accept p_j cows.
Give a polynomial time algorithm that takes in the parameters c_i and p_j and the cow-walk times to each farm and determines whether moving all the cows to the farms on higher ground is possible along with a plan when it is. Hint: This can be cast as a network flow problem. (6 points)

Using the hint provided, we can create a network flow problem where the source has edges connecting to each farm, with the edge's capacity being the number of cows at that farm. For each safe farm, we can create an edge from the safe farm to the sink, with the capacity being the number of cows that can be accepted at that safe farm. We can then create edges from each farm to every safe farm that is within 2 hours of walking, with the capacity being the number of cows originally at the farm.

With the problem created, we can run the Ford-Fulkerson algorithm to determine if it is possible to move all the cows to the safe farms. If the maximum flow equals the total number of cows, then it is possible to move all the cows to the safe farms. If the maximum flow is less than the total number of cows, then it is not possible to move all the cows to the safe farms.

Finding the actual plan requires using the residual graph to determine the flow of cows from each farm to each safe farm.