CS 170: Efficient Algorithms and Intractable Problems

Spring 2017

•

Homework 4

Due on Tuesday, Februrary 28h, 2017 at 11:59am

•

Solutions by

Ninh DO

25949105

In collaboration with

None

Problem 1: Short Questions

★★ Level

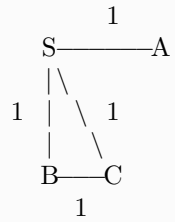
For the following claims, answer yes or no and provide justification. Consider an arbitrary graph G with positive edge weights. Shortest path means least cost path.

a. Let G be a connected undirected graph with positive length on all the edges. Let s be a fixed vertex. Let $d(s, v)$ denote the distance from vertex s to vertex v, i.e., the length of the shortest path from s to v. If we choose the vertex v that makes $d(s, v)$ as small as possible, subject to the requirement that $v \neq s$, then does every edge on the path from s to v have to be part of every minimum spanning tree of G?

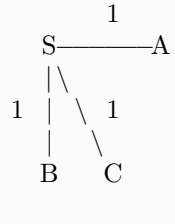
Solution

NO. The problem wording is confusing since it can be interpreted in different ways, but there is an invariant: Given a fixed vertex s, the vertex v that makes $d(s, v)$ as small as possible must be the closest neigbor of s. Because if v is not a neighbor of s, there is at least one vertex, say v', on the shortest path from s to v and lying between s and v. If we just simply choose the vertex v' instead of v, the distance $d(s, v')$ is smaller than $d(s, v)$ and this contradicts the fact that $d(s, v)$ is as small as possible. Similarly, v' is the closest neighbor because if there is another closer neighbor, the distance $d(s, v')$ is not as small as possible.

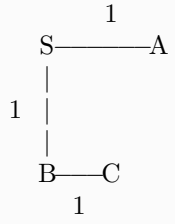
Now the problem becomes "does the shortest edge from s have to be a part of every minimum spanning tree of G?". Consider the following grap G as a counterexample:



which has two MSTs as below



MST 1



MST 2

The vertex C satisfies the condition "$d(s, v)$ as small as possible" but the edges SC does not belongs to MST 2, so it is not a part of *every* MST of G.

b. The same question as above, except now no two edges can have the same length.

Solution

YES. The edge lengths are distinct, so the closest neighbor of S is unique, called C, and SC is the unique shortest edge from S. Consider all MSTs of the graph G, they must have the same weights. If SC is not a part of at least one MST, say MST 1, we replace the S-connecting edge of MST 1 by SC, we obtain the new tree, since the number of edge is still $|V| - 1$, that have smaller weight than the old MST, because SC is shorter than the replaced edge. This contradicts the fact that MST 1 is a MST, so SC must be a part of MST 1 from the very beginning. Generally, SC must be a part of every MST of the graph G.

Problem 2: Huffman Encoding

★★ **Level**

We use Huffman's algorithm to obtain an encoding of alphabet a, b, c with frequencies f_a, f_b, f_c. In each of the following cases, either give an example of the frequencies (f_a, f_b, f_c) that would yield the specified code, or explain why the code cannot possibly be obtained (no matter what the frequencies are)¿

a. Code: $\{0, 10, 11\}$

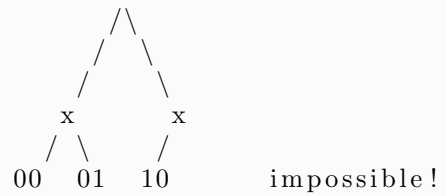
Solution
(90, 20, 10)

b. Code: $\{0, 1, 00\}$

Solution
No code exists, since 0 is prefixing 00

c. Code: $\{10, 01, 00\}$

Solution
No code exists, since 10 is a single child



Problem 3: Preventing Conflict

★★★★ Level

A group of n guests shows up to a house for a party, and any two guests are either friends or enemies. There are two rooms in the house, and the host wants to distribute guests among the rooms, breaking up as many pairs of enemies as possible. The guests are all waiting outside the house and are impatient to get in, so the host needs to assign them to the two rooms quickly, even if this means that it's not the best possible solution. Come up with an efficient algorithm that breaks up at least half the number of pairs of enemies as the best possible solution, and prove your answer.
Hint: Try assigning guests one at a time. Consider how many pairs of enemies are broken up with each iteration.

Solution

Main idea:

If we represent the group of n guests and their enemy relation as a graph G, i.e. guests are vertices and two guests who are enemies are connected by an edge, then drawing a line crossing the graph, we can imagine that each side of the line represents a room, and the number of edges are crossed by the line is the number of enermy breakups. This number should be at least half of the total edge number $|E|$.
To achieve the goal, we apply the greedy algorithm: add one guest to a room at a time, before selecting which room to add that guest, called to-be-seated guest, compute the guest's numbers of conflicts with the guests already in room A and with the guests already in room B, compare two numbers and add the guest to the room with smaller corresponding number of conflicts.
We can loop through the "enemy" list of a to-be-seated guest (list of other guests that share the same edge with the to-be-seated guest) and check if each guest is in room A or room B. The checking takes constant time if we use hash table with good hash function.

Pseudocode:

```
Initialize roomA, roomB are two empty hash table
For guest in guest list:
    nA, nB = 0, 0    # numbers of conflict in each room
    for e in guest's enemy list:
        if roomA.has(e):
            nA++
        else if roomB.has(e):
            nB++
        end
    end
    nA < nB? roomA.add(guest) : roomB.add(guest)
end
```

Proof of correctness:

The solution meets our purpose to break up at least half of enemy pairs because for each iteration, we eliminate at least half enemy connections of a to-be-seated guest, i.e. when adding a guest to a room we always select the room in which the guesst has a smaller number of conflicts, that is we eliminate the edges that represent the guest's conflicts with those guys in the other room, and this number is at least half of the guest's enemies or more. Thus, for the whole implementation, we break up at least half the number of enemy pairs.

Runtime:

We go once through each guest/vertex, and twice through each edge since this is adjacency-list undirected graph. Checking guests in rooms using hash table take constant time, so the total runtime is $O(|V| + |E|)$.

Problem 4: Graph Subsets

★★ **Level**

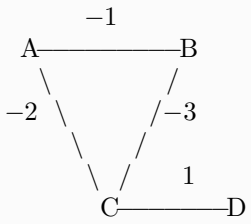
Let $G = (V, E)$ be a connected, undirected graph, with edge weights $w(e)$ on each edge e. Some edge weights *might be negative*. We want to find a subset of edges $E' \subseteq E$ such that $G' = (V, E')$ is connected, and the sum of the edge weights in E' is as small as possible.

1. Is it guaranteed that the optimal solution E to this problem will always form a tree?

   Solution
   NO. If the graph has more than $|V| - 1$ negative edge, than at least those negative edges should be included in E'. Since E' has more than $|V| - 1$ edges, $G' = (V, E')$ is non-tree.

2. Does Kruskal's algorithm solve this problem? If yes, explain why in a sentence or two; if no, give a small counterexample.

   Solution
   NO. Kruskal's algorithm does not solve this problem. Counterexample:

   Kruskal's would add BC, then AC, then skip edge AB since it creates a cycle, then add CD. But the subset $E' = E$ has smaller total weight and make the graph $G' = (V, E')$ connected.

3. Briefly describe an efficient algorithm for this problem. Just the main idea is enough (1-3 sentences). No need for a 4-part solution.

   Solution
   Run Kruskal's algorithm, then add all negative edges. The MST combined with all negative edges is the solution. The MST ensures the graph connectivity and minimum total weight without taking negative edges into account, while adding all negative edges make the total weight as small as possible.

Problem 5: Arbitrage

★★★ **Level**

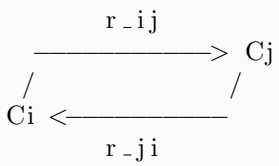
Shortest-path algorithms can also be applied to currency trading. Suppose we have n currencies $C = c_1, c_2, \ldots, c_n$: e.g., dollars, Euros, bitcoins, dogecoins, etc. For any pair i, j of currencies, there is an exchange rate $r_{i,j}$: you can buy $r_{i,j}$ unites of currency c_j at the price of one unit of currency c_i. Assume that $r_{i,j} = 1$ and $r_{i,j} \geq 0$ for all i, j.

(a) The Foreign Exchange Market Organization (FEMO) has hired Oski, a CS170 alumnus, to make sure that it is not possible to generate a profit through a cycle of exchanges, and end with more than one unit of currency i. (That is called *arbitrage*.) Give an efficient algorithm for the following problem: given a set of exchange rates $r_{i,j}$ and two specific currencies s, t, find the most advantageous sequence of currency exchanges for converting currency s into currency t. We recommend that you represent the currencies and rates by a graph whose edge lengths are real numbers.

Solution

Main idea:

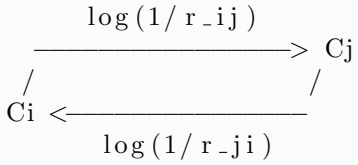
First, we represent currencies $c_i's$ as vertices and rates $r_{i,j}$ as directed edges from c_i to c_j of the graph G. A link of G looks like:



Now we go step by step to modify the graph. The problem is to find the path from $c_s \to c_t$ so that the product $r_{si} \times r_{ij} \times r_{jk} \times \cdots \times r_{rt}$ is maximum, or the inverse product $\dfrac{1}{r_{si} \times r_{ij} \times r_{jk} \times \cdots \times r_{rt}}$ is minimum. Taking the log of the inverse product to decompose it into sum give:

$$\log\left(\frac{1}{r_{si} \times r_{ij} \times r_{jk} \times \cdots \times r_{rt}}\right) = \log\left(\frac{1}{r_{si}}\right) + \log\left(\frac{1}{r_{ij}}\right) + \cdots + \log\left(\frac{1}{r_{rt}}\right) \quad \text{minimum}$$

Now we replace the edges $r_{i,j}$ of G by $\log\left(\dfrac{1}{r_{i,j}}\right)$



The problem turns into finding the shortest path from $c_s \to c_t$ in terms of length/weight $\log\left(\dfrac{1}{r_{i,j}}\right)$. Since the egde weight can be negative if $r_{i,j} > 1$, we use Bellman-Ford algorithm.

Pseudocode:

1. Form a graph G with currencies $c_i's$ as vertices and rates $r_{i,j}$ as edges directing from c_i to c_j.
2. Run Bellman-Ford to find the shortest path from $c_s \to c_t$.
3. Return the sequence from $c_s \to c_t$.

Proof of Correctness:

Solution (cont.)

The shortest path from $c_s \to c_t$ has

$$\log\left(\frac{1}{r_{si}}\right) + \log\left(\frac{1}{r_{ij}}\right) + \cdots + \log\left(\frac{1}{r_{rt}}\right) \quad \text{minimum}$$

$$\text{or} \quad \log\left(\frac{1}{r_{si} \times r_{ij} \times r_{jk} \times \cdots \times r_{rt}}\right) \quad \text{minimum}$$

$$\text{or} \quad \frac{1}{r_{si} \times r_{ij} \times r_{jk} \times \cdots \times r_{rt}} \quad \text{minimum}$$

$$\text{or} \quad r_{si} \times r_{ij} \times r_{jk} \times \cdots \times r_{rt} \quad \text{maximum}$$

That is what we want. The Bellman-Ford algorithm is well-known, so it is correct.

Runtime:

The Bellman-Ford algorithm has the runtime: $O(|V| \cdot |E|)$. In our problem, $|E|$ is determined:

$$|E| = \frac{|V|(|V|-1)}{2} \sim \Theta\left(|V|^2\right)$$

Thus, the total runtime is $O\left(|V|^3\right)$

(b) In the economic downturn of 2016, the FEMO had to downsize and let Oski go, and the currencies are changing rapidly, unfettered and unregulated. As a responsible citizen and in light of what you saw in lecture this week, this makes you very concerned: it many now be possible to find currencies $c_{i_1}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \times r_{i_2,i_3} \times \ldots \times r_{i_{k-1},i_k} \times r_{i_k,i_1} > 1$. This means that by starting with one unit of currency c_{i_1} and then successively converting it to currencies $c_{i_2}, c_{i_3}, \ldots, c_{i_k}$ and finally back to c_{i_1}, you would end up with more than one unit of currency c_{i_1}. Such anomalies last only a fraction of a minute on the currency exchange, but they provide and opportunity for profit.
You decide to step up and help out the World Bank. Given an efficient algorithm for detecting the presence of such an anomaly. You may use the same graph representation as for part (a).

Solution

Main idea:

$$r_{si} \times r_{ij} \times r_{jk} \times \cdots \times r_{rt} > 1$$

$$\text{i.e} \quad \frac{1}{r_{si} \times r_{ij} \times r_{jk} \times \cdots \times r_{rt}} < 1$$

logging both sides gives:

$$\log\left(\frac{1}{r_{si}}\right) + \log\left(\frac{1}{r_{ij}}\right) + \cdots + \log\left(\frac{1}{r_{rt}}\right) < 0$$

The problem turns into finding the negative cycle of the same graph.
Again, we run the Bellman-Ford algorithm, after $|V|-1$ iterations, we run 1 more iteration. If there is any distance decrease, exists a negative cycle.

Pseudocode:

1. Form a graph G with currencies $c_i's$ as vertices and rates $r_{i,j}$ as edges directing from c_i to c_j.
2. Take a vertex and run Bellman-Ford with $|V|-1$ iterations to update the shortest path from $c_s \to c_t$.
3. Run one more iteration and return true if there is a distance decreasing.

Proof of Correctness:

Similar to part (a), the Bellman-Ford is correct and one more iteration running to detect negative cycles.

Runtime:

Same as part (a), the total runtime is $O\left(|V|^3\right)$