CS 170 Efficient Algorithms and Intractable Problems
Spring 2017 Prasad Raghavendra and Sanjam Garg

Homework 4

Instructions: You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

If using LaTeX (which we recommend), you may use the homework template linked on this Piazza post to get started.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc). For questions asking you to give an algorithm, respond in what we will refer to as the *four-part format* for algorithms: main idea, pseudocode, proof of correctness, and running time analysis.

Read the Homework FAQ Piazza post on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 12 homework assignments, the lowest two scores will be dropped.

Special Questions:

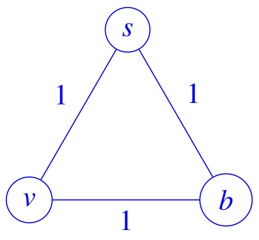
- *Shortcut questions*: Short questions are usually easy questions that give you opportunities to practice basic materials. However, we understand that some of you are very familiar with the topics and do not want to spend too much time on easy questions. Therefore, we design shortcut questions for this purpose. A shortcut question usually has multiple parts that build upon each other and are ordered by their difficulty level. You can work on those in order or start from wherever you like. However you only need to submit the last part you are able to solve. For example, if a question has 5 parts (a, b, c, d, e), you are confident about part e, you should submit part e without any of the previous four parts. If you are confident about d but not sure about e, you should submit d for grading purposes. Please clearly indicate in your submission which part you are submitting.
- *Redemption questions*: It is important that you carefully read the posted solutions, even for problems you got right. To encourage this, you have the option of submitting a redemption file, a few paragraphs in which you explain, for each problem you choose to cover, what you did wrong and what the right idea was in your own words (not cutting and pasting from the solution!), and appending it to your homework. For example, suppose that as you review your solutions to HW1, you realize you had misunderstood question 3 and answered it incorrectly. You would write down what you just learned, and then submit it in your HW2 assignment the following week. Because these are mainly for your benefit, feel free to format them however is most useful for you.
- *Extra credit questions*: We might have some extra credit questions in the homework for people who really enjoy the materials. However, please note that you should do the extra credit problems only if you really enjoy working on these problems and want an extra challenge. It is likely not the most efficient manner in which to maximize your score.

Due Tuesday, February 28, at 11:59am

1. (★★ level) Short Questions

1. Let G be a connected undirected graph with positive lengths 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: False. Consider the following counterexample:

   Observe that the shortest path from s to v has one edge. But (s,b) and (b,v) form a minimum spanning tree that doesn't include (s,v). So (s,v) is not part of every minimum spanning tree.

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

   Solution: True. First let's analyze the the definition. We need to find a vertex v that minimizes $d(s,v)$. Because the edge weights are positive, v has to be a neighbor a s. Or in other words, part (h) claims the lightest edge that is incident on s has to part of every minimum MST. Let us call this edge e. Assume, for contradiction, that e is not part of some MST T. Let us add e to T. This creates a cycle, which goes through e to s and exit s through another edge e'. We now remove e'. Removing an edge from a cycle keeps the graph connected and a tree. This creates a tree which is lighter than T which contradicts our assumptions that T is a MST.

2. (★★ level) Huffman Encoding

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: $\{0,10,11\}$

(b) Code: $\{0,1,00\}$

(c) Code: $\{10,01,00\}$

Solution:

(a) $(f_a, f_b, f_c) = (2/3, 1/6, 1/6)$ gives the code $\{0,10,11\}$.

(b) This encoding is not possible, since the code for a (0), is a prefix of the code for c (00).

(c) This code is not optimal since $\{1,01,00\}$ gives a shorter encoding. Also, it does not correspond to a *full* binary tree and hence cannot be given by the Huffman algorithm.

3. (★★★★ level) Preventing Conflict

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: Let the guests be nodes in a graph $G = (V, E)$ and let there be an edge between two nodes if they are enemies. The number of conflicts prevented after partitioning nodes into two disjoint sets A and B (also called a cut) is the number of edges between A and B. One by one, we will assign nodes to either set A or B. When we are considering node v, we should know how many edges it has going into A and how many edges it has going into B. We greedily assign it to the set where it has a smaller total number of edges to. This cuts at least half of the total edges.

Pseudocode:

```
1.  Initialize empty sets $A$ and $B$
2.  For each node $v \in V$:
3.        Initialize $n_A := 0$, $n_B := 0$
4.        For each $\{v, w\} \in E$:
5.              Increment $n_A$ or $n_B$ if $w \in A$ or $w \in B$, respectively
6.        Add $v$ to set $A$ or $B$ with lower $n_A$ or $n_B$, breaking ties arbitrarily
7.  Output sets $A$ and $B$
```

Proof of Correctness: In each iteration, when we consider a vertex v, its edges are connected to other vertices already in these three disjoint sets: A, B, or the set of not-yet-assigned vertices. Let the number of edges in each case be n_A, n_B and n_X respectively. Suppose we add v to A, then n_B edges will be cut, but n_A edges can never be cut. Since $\max(n_A, n_B) \geq \frac{n_A+n_B}{2}$, each iteration will cut at least $\frac{n_A+n_B}{2}$ edges, and in total at least $\frac{|E|}{2}$ of the edges will be cut. Let $\texttt{greedycut}(G)$ be the number of edges cut by our algorithm, and $\texttt{maxcut}(G)$ be the best possible number of edges cut. Thus

$$\frac{\texttt{greedycut}(G)}{\texttt{maxcut}(G)} \geq \frac{\texttt{greedycut}(G)}{|E|} \geq \frac{|E|/2}{|E|} \geq \frac{1}{2}$$

Running Time: Each vertex is iterated through once, and each edge is iterated over at most twice, thus the runtime is $O(|V| + |E|)$. Hopefully this is fast enough to please the impatient guests!

4. (★★ level) Graph Subsets

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. Consider a graph with every edge present (a complete graph), each with weight -1. Then the solution is clearly taking every edge of G, and this is definitely not a 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. As in the example above, the answer is not always a tree. Kruskals algorithm always returns a tree.

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:
   Solution 1:

   Add all the negative edges to the answer. When adding a negative edge, merge the two vertices. Then, after adding all negative edges, run Kruskal's Algorithm on the modified graph so far. This will determine what remaining edges to add to ensure connectivity.

   Solution 2:

   Run Kruskal's. Then, add all negative edges that are not already part of the MST.

5. (★★★ level) **Arbitrage** 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 $r_{i,j}$ units of currency c_j at the price of one unit of currency c_i. Assume that $r_{i,i} = 1$ 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; that is, for any currency $i \in C$, it is not possible to start with one unit of currency i, perform a series 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:

We represent the currencies as the vertex set V of a complete directed graph G and the exchange rates as the edges E in the graph. Finding the best exchange rate from s to t corresponds to finding the path with the largest product of exchange rates. To turn this into a shortest path problem, we weigh the edges with the negative log of each exchange rate. Since edges can be negative, we use Bellman-Ford to help us find this shortest path.

Pseudocode:

```
1: function BESTCONVERSION($s,t$)
2:     $G \leftarrow$ Complete directed graph, $c_i$ as vertices, edge lengths $l = \{-\log(r_{i,j}) \mid (i,j) \in E\}$.
3:     dist,prev $\leftarrow$ BELLMANFORD($G,l,s$)
4:     return Best rate: $e^{-\texttt{dist}[t]}$, Conversion Path: Follow pointers from $t$ to $s$ in prev
```

Proof of Correctness:

To find the most advantageous ways to converts c_s into c_t, you need to find the path $c_{i_1}, c_{i_2}, \cdots, c_{i_k}$ maximizing the product $r_{i_1,i_2} r_{i_2,i_3} \cdot \cdots \cdot r_{i_{k-1},i_k}$. This is equivalent to minimizing the sum $\sum_{j=1}^{k-1}(-\log r_{i_j,i_{j+1}})$. Hence, it is sufficient to find a shortest path in the graph G with weights $w_{ij} = -\log r_{ij}$. Because these weights can be negative, we apply the Bellman-Ford algorithm for shortest paths to the graph, taking s as origin. The correctness of the entire algorithm follows from the proof of correctness of Bellman-Ford.

Runtime:

Same as runtime of Bellman-Ford, $O(|V|^3)$ since the graph is complete.

(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 may now be possible to find currencies $c_{i_1}, \dots, c_{i_k}$ such that $r_{i_1,i_2} \times r_{i_2,i_3} \times \cdots \times r_{i_{k-1},i_k} \times r_{i_k,i_1} > 1$. This means that by starting with one unit of currency c_{i_1} and then successively converting it to currencies $c_{i_2}, c_{i_3}, \dots, c_{i_k}$ 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 an opportunity for profit.

You decide to step up to 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:

Just iterate the updating procedure once more after $|V|$ rounds. If any distance is updated, a negative cycle is guaranteed to exist, i.e. a cycle with $\sum_{j=1}^{k-1}(-\log r_{i_j,i_{j+1}}) < 0$, which implies $\prod_{j=1}^{k-1} r_{i_j,i_{j+1}} > 1$, as required.

Pseudocode: This algorithm takes in the same graph constructed in the previous part.

```
1: function HasArbitrage($G$)
2:     dist,prev $\leftarrow$ BellmanFord($G, l, s$)
3:     dist$^*$ $\leftarrow$ Update all edges one more time
4:     return True if for some $v$, dist[$v$] $>$ dist$^*$[$v$]
```

Proof of Correctness:

Same as the proof for the modification of Bellman-Ford to find negative edges.

Runtime:

Same as Bellman-Ford, $O(|V|^3)$.

Note:

Both questions can be also solved with a variation of Bellman-Ford's algorithm that works for multiplication and maximizing instead of addition and minimizing.