CS 124: Problem Set 3

Chao Cheng
Collaborators: Hao Wang, Steve Li

February 4, 2021

Problem 1. Let us create a directed graph $G(V, E)$ such that each currency c_i corresponds to a vertex, and each exchange rate $r_{i,j}$ corresponds to a directed edge from c_i to c_j. To determine if a risk-free currency exchange exists, we see that in order for any chain of exchange rates to be greater than 1, or, mathematically, if

$$r_{a_1,a_2} \cdot r_{a_2,a_3} \cdot \ldots \cdot r_{a_{k-1},a_k} \cdot r_{a_k,a_1} > 1$$

then taking the inverse of both sides of the inequality, we have that

$$\frac{1}{r_{a_1,a_2}} \cdot \frac{1}{r_{a_2,a_3}} \cdot \ldots \cdot \frac{1}{r_{a_{k-1},a_k}} \cdot \frac{1}{r_{a_k,a_1}} < 1$$

and then taking the log of both sides, we have

$$\begin{gathered}
\log \frac{1}{r_{a_1,a_2}} + \log \frac{1}{r_{a_2,a_3}} + \ldots + \log \frac{1}{r_{a_{k-1},a_k}} + \log \frac{1}{r_{a_k,a_1}} < 0 \\
-\log r_{a_1,a_2} - \log r_{a_2,a_3} - \ldots - \log r_{a_{k-1},a_k} - \log r_{a_k,a_1} < 0
\end{gathered}$$

Let us assign weights to edges such that for an edge (v_i, v_j) from the vertex corresponding to c_i to the vertex corresponding to c_j, its weight is $w(v_i, v_j) = -\log r_{i,j}$. Then we have that a risk-free currency exchange exists if there exists a negative cycle in the graph using the edge weights assigned.

To determine whether a negative cycle exists, we add a vertex v_0 to the graph with edges from v_0 to all other vertices, each with weight 0, then use the Bellman-Ford algorithm starting from v_0 to check if there are any negative cycles (which would imply that there exists at least one risk-free currency exchange), returning True if a negative cycle is detected and False if no negative cycles are detected (i.e. Bellman-Ford returns a shortest path). The reason why adding a vertex does not change the output of our algorithm is because the directed edges from v_0 to every other vertex with 0 weight allow Bellman-Ford starting from v_0 to reach any potential negative cycles without introducing any new negative cycles.

The time complexity of this algorithm is $O(n^3)$, or equivalently, $O(V^3)$, since creating the graph takes $O(n^2) = O(V^2)$ time and Bellman-Ford takes $O(n^3) = O(VE) = O(V^3)$ time (since the graph can have up to $O(V^2)$ edges).

Problem 2. Suppose that the weight of edge $e = (v_a, v_b)$ is increased, where $e \in T$. Let T_a denote the subtree of T starting from v_a and T_b denote the subtree of T starting from v_b after e is removed from T. We iterate over all edges with one vertex in T_a and the other vertex in T_b, keeping track of the edge with the lowest weight, e_{min}. Then, we connect T_a and T_b using e_{min} to obtain the new minimum spanning tree.

This algorithm is correct because after removing e, we have a subset $X \subseteq T$. Let S be the set of vertices in T_a and $V - S$ be the set of vertices in T_b (since all the vertices not in T_a are in T_b); then no edge in X crosses between S and $V - S$. Therefore by the cut property, e_{min} is part of a minimum spanning tree. We also have that T_a is a minimum spanning tree of S. To see why, if we remove a vertex v_a from T_a, then the cut property says that the minimum weight edge from v_a to any other vertex in the graph must be part of a minimum spanning tree. If this edge is not in T_a, then it is not in T either, which is a contradiction since we are given that T is a minimum spanning tree, and therefore the minimum weight edge connecting v_a to the rest of the graph is in T_a, so T_a is a minimum spanning tree of S. For the same reason, T_b is also a minimum spanning tree of $V - S$, and so connecting T_a and T_b with e_{min} results in a new minimum spanning tree.

The time complexity of this algorithm is linear: $O(E)$, since iterating over the edges between T_a and T_b to find the minimum-weight edge takes $O(E)$ time.

Problem 3. Let X denote the set $X = \{1, 2, \ldots, n\}$ with $n = 3b$ elements, for any $b \geq 1$. Suppose $A = \{1, 4, \ldots, 3b-2\}$, $B = \{2, 5, \ldots, 3b-1\}$, and $C = \{3, 6, \ldots, 3b\}$ (i.e. $A = \{x \in X | x \equiv 1 \mod 3\}$, $B = \{x \in X | x \equiv 2 \mod 3\}$, and $C = \{x \in X | x \equiv 0 \mod 3\}$). Then $A \cup B \cup C = X$, so the minimal set cover of X has size $k = 3$.

In addition to A, B, and C, let us also consider the following subsets: let G_1 be the first "half" of X, i.e. $G_1 = \{1, 2, \ldots, \frac{n}{2}\}$, let G_2 be the first half of the remaining half of X, i.e. $G_2 = \{\frac{n}{2}+1, \ldots, \frac{3n}{4}\}$, let G_3 be the first half of the remaining quarter of X, i.e. $G_3 = \{\frac{3n}{4}+1, \ldots, \frac{7n}{8}\}$, and so on where each subsequent G_i is the first half of the remaining elements of X.

Then the greedy algorithm will first choose G_1, since the number of uncovered elements in G_1 is $|G_1| = \frac{n}{2}$, which is larger than the number of uncovered elements in A, B, or C (since $|A| = |B| = |C| = \frac{n}{3}$).

The greedy algorithm will subsequently choose G_2, since the number of uncovered elements in G_2 is $|G_2| = \frac{n}{4}$, which is larger than the number of uncovered elements in A, B, or C, each of which has $\frac{n}{6}$ uncovered elements (since half of the original $\frac{n}{3}$ elements have already been covered by G_1).

It follows that the greedy algorithm will continue to pick G_i each time until it reaches the point where the number of uncovered elements less than 3, because then the number of uncovered elements in G_i will be 1, and either A, B, or C will also have 1 uncovered element. Then the algorithm will arbitrarily choose any of these subsets until there are no uncovered elements left.

Since the greedy algorithm chooses a subset of size $\frac{n}{2}$, then $\frac{n}{4}$, then $\frac{n}{8}$, and so on until the subsets to choose from are each size 1 (in which case it chooses arbitrarily), the size of the set cover generated by the algorithm is strictly greater than $\log n$, and therefore we have shown that for the example above, the set cover returned by the greedy algorithm has size $\Omega(\log n)$.

For other values of $k > 3$, we can amend the X so that $X = \{1, 2, \ldots, bk\}$ with size $n = bk$ for any $b \geq 1$ (i.e. n is a multiple of k). Our minimal set cover then simply comprises of the sets of the form $\{x \in X | x \equiv a \mod k\}$ for each $a = 0, 1, \ldots k-1$, and the size of this minimal set cover is k. The definition of G_i remains the same, and the algorithm will continue to choose G_i over any of the subsets in the minimal set cover until $|G_i| = 1$, and therefore for any such set cover problem, the set cover returned by the greedy algorithm has size $\Omega(\log n)$.

Problem 4. 1. For two machines, let us consider two cases with respect to the job with the maximum running time (denoted j_{max}, with running time r_{max}). Here, $T = \sum_{i=1}^{n} r_i$ denotes the sum of all the running times. We will show that in both cases, the greedy algorithm's completion time is bounded by the best scenario times a factor of $\frac{3}{2}$. We note that in any scenario, the completion time is at least as long as r_{max}, and at least as long as $\frac{T}{2}$.

- **Case 1:** $r_{max} \leq \frac{T}{2}$. In this case, the best scenario occurs when all the jobs are allocated evenly so that the two machines total loads are equal, and the completion time for each is $\frac{T}{2}$ (note that though it may not always be numerically possible for the two total loads to be numerically equal, any distribution of loads that results in the two total loads being as close as possible will result in a total load time greater than $\frac{T}{2}$ and thus will be greater than this hypothetical minimum completion time). For the worst scenario of the greedy algorithm, we see that the difference in total load times can be at most r_{max}. To see why, first distribute all the jobs except j_{max} so that the two total loads are equal (or as close to equal as possible); then each total load is equal to $\frac{T-r_{max}}{2}$. Adding j_{max} to either machine will result in the worst scenario, and the overall completion time will be $\frac{T-r_{max}}{2} + r_{max} = \frac{T+r_{max}}{2}$. Since $r_{max} \leq \frac{T}{2}$, we have that

$$\frac{T + r_{max}}{2} \leq \frac{T + \frac{T}{2}}{2} = \frac{3T}{4} = \left(\frac{3}{2}\right)\left(\frac{T}{2}\right)$$

  so the $\frac{3}{2}$ bound holds for this case.

- **Case 2:** $r_{max} > \frac{T}{2}$. In this case, the best scenario occurs when all the jobs except j_{max} are allocated to one machine, while j_{max} is allocated to the other machine; the overall completion time is then r_{max}, since all the other jobs combined take less time than j_{max} (since $T - r_{max} < \frac{T}{2} < r_{max}$). The worst scenario of the greedy algorithm occurs when all the jobs except j_{max} are first distributed equally (or as equally as possible) between the two machines, so that each total load is equal to $\frac{T-r_{max}}{2}$. Adding j_{max} to either machine will result in the overall completion time being $\frac{T-r_{max}}{2} + r_{max} = \frac{T+r_{max}}{2}$. Since $\frac{T}{2} < r_{max}$, we have that

$$\frac{T + r_{max}}{2} < r_{max} + \frac{r_{max}}{2} = \left(\frac{3}{2}\right)(r_{max})$$

  so the $\frac{3}{2}$ bound holds for this case.

As an example of when a factor of $\frac{3}{2}$ is achieved, consider when $r_{max} = \frac{T}{2}$. Then the best scenario is when all the jobs except j_{max} are allocated to one machine, while j_{max} is allocated to the other machine, so that the completion time is $\frac{T}{2}$. The worst scenario of the greedy algorithm is when all the jobs except j_{max} are allocated evenly among the two machines so that the completion time is $\frac{T}{4}$, and then adding j_{max} to either machine increases the overall completion time to $\frac{T}{4} + \frac{T}{2} = \frac{3T}{4}$, which equals $(\frac{3}{2})(\frac{T}{2})$.

2. For m machines, we again consider two cases with respect to the job with the maximum time, j_{max}, with running time r_{max}. Again, $T = \sum_{i=1}^{n} r_i$ denotes the sum of all running times.

- **Case 1:** $r_{max} \leq \frac{T}{m}$. In this case, the best scenario occurs when all the jobs are allocated evenly across the machines, and the overall completion time is $\frac{T}{m}$. The worst scenario of the greedy algorithm occurs when first, all the jobs except j_{max} are allocated evenly across the machines so that the completion time for each machine is $\frac{T-r_{max}}{m}$, and then

j_{max} is added to any of the machines so that the overall completion time for all machines becomes $\frac{T-r_{max}}{m} + r_{max}$. Since $r_{max} \leq \frac{T}{m}$, we have that

$$\frac{T - r_{max}}{m} + r_{max} \leq \frac{T - \frac{T}{m}}{m} + \frac{T}{m} = \frac{(2m-1)T}{m^2} = \left(\frac{2m-1}{m}\right)\left(\frac{T}{m}\right)$$

so we have a $\frac{2m-1}{m}$ bound for this case.

- **Case 2:** $r_{max} > \frac{T}{m}$. In this case, the best scenario occurs when j_{max} is allocated to one machine, and the rest of the jobs are allocated evenly among the other machines. Since $r_{max} > \frac{T}{m}$, and the completion time for each of the other machines is at best $\frac{T-r_{max}}{m}$ which is less than r_{max} (since $\frac{T-r_{max}}{m} < \frac{T}{m} < r_{max}$), we know that the overall completion time for the best scenario is r_{max}. The worst scenario of the greedy algorithm occurs when all the jobs except j_{max} are first distributed evenly across all m machines, so that the completion time for each machine is $\frac{T-r_{max}}{m}$, and then j_{max} is added to any of the machines so that the overall completion time for all machines becomes $\frac{T-r_{max}}{m} + r_{max}$. Since $r_{max} > \frac{T}{m}$, we have that

$$\frac{T - r_{max}}{m} + r_{max} < r_{max} - \frac{r_{max}}{m} + r_{max} = \frac{(2m-1)r_{max}}{m} = \left(\frac{2m-1}{m}\right)(r_{max})$$

so we have a $\frac{2m-1}{m}$ bound for this case.

Thus for either case, we have an upper bound for the greedy algorithm that is the best scenario times a factor of $\frac{2m-1}{m}$. We will show examples of when this factor is achieved for $m = 3, 4, 5$. We note that in any scenario, the completion time is at least as long as r_{max}, and at least as long as $\frac{T}{m}$.

- For $m = 3$, consider when $r_{max} = \frac{T}{3}$. The best case scenario results in an overall completion time of $\frac{T}{3}$, and the worst scenario of the greedy algorithm is when all the jobs except j_{max} are first allocated evenly among each machine so that the completion time is $\frac{T-r_{max}}{3} = \frac{2T}{9}$, and then j_{max} is added to any machine so that the overall completion time is $\frac{2T}{9} + \frac{T}{3} = \frac{5T}{9}$, which equals $(\frac{5}{3})(\frac{T}{3})$, and thus satisfies the factor of $\frac{2m-1}{m} = \frac{5}{3}$.
- For $m = 4$, consider when $r_{max} = \frac{T}{4}$. The best case scenario results in an overall completion time of $\frac{T}{4}$, and the worst scenario of the greedy algorithm is when all the jobs except j_{max} are first allocated evenly among each machine so that the completion time is $\frac{T-r_{max}}{4} = \frac{3T}{16}$, and then j_{max} is added to any machine so that the overall completion time is $\frac{3T}{16} + \frac{T}{4} = \frac{7T}{16}$, which equals $(\frac{7}{4})(\frac{T}{4})$, and thus satisfies the factor of $\frac{2m-1}{m} = \frac{7}{4}$.
- For $m = 5$, consider when $r_{max} = \frac{T}{5}$. The best case scenario results in an overall completion time of $\frac{T}{5}$, and the worst scenario of the greedy algorithm is when all the jobs except j_{max} are first allocated evenly among each machine so that the completion time is $\frac{T-r_{max}}{5} = \frac{4T}{25}$, and then j_{max} is added to any machine so that the overall completion time is $\frac{4T}{25} + \frac{T}{5} = \frac{9T}{25}$, which equals $(\frac{9}{5})(\frac{T}{5})$, and thus satisfies the factor of $\frac{2m-1}{m} = \frac{9}{5}$.

From here on, it is clear that we can generalize the example of setting $r_{max} = \frac{T}{m}$ to achieve the bound of $\frac{2m-1}{m}$ for any m.

Problem 5. a) We can split the two n-digit numbers, x and y into three parts, like so:

$$x = 10^{\frac{2n}{3}}a + 10^{\frac{n}{3}}b + c \qquad y = 10^{\frac{2n}{3}}d + 10^{\frac{n}{3}}e + f$$

Then we have

$$\begin{aligned} xy &= 10^{\frac{4n}{3}}ad + 10^{n}bd + 10^{\frac{2n}{3}}cd + 10^{n}ae + 10^{\frac{2n}{3}}be + 10^{\frac{n}{3}}ce + 10^{\frac{2n}{3}}af + 10^{\frac{n}{3}}bf + cf \\ &= (ad)10^{\frac{4n}{3}} + (ae+bd)10^{n} + (af+be+cd)10^{\frac{2n}{3}} + (bf+ce)10^{\frac{n}{3}} + cf \end{aligned}$$

Now consider the following 6 products:

$$\begin{aligned} &X = ad && Y = be && Z = cf \\ &U = (a+b)(d+e) && V = (b+c)(e+f) && W = (a+c)(d+f) \end{aligned}$$

and note that

$$\begin{aligned} U &= (a+b)(d+e) = ad + (ae+bd) + be \implies ae + bd = U - X - Y \\ V &= (b+c)(e+f) = be + (ce+bf) + cf \implies bf + ce = V - Y - Z \\ W &= (a+c)(d+f) = ad + (cd+af) + cf \implies af + cd = W - X - Z \end{aligned}$$

Thus, we can write the earlier product xy as

$$\begin{aligned} xy &= (ad)10^{\frac{4n}{3}} + (ae+bd)10^{n} + (af+be+cd)10^{\frac{2n}{3}} + (bf+ce)10^{\frac{n}{3}} + cf \\ &= (X)10^{\frac{4n}{3}} + (U-X-Y)10^{n} + \big((W-X-Z)+Y\big)10^{\frac{2n}{3}} + (V-Y-Z)10^{\frac{n}{3}} + Z \end{aligned}$$

meaning we can compute xy by performing only 6 multiplications.

b) Since we are splitting the problem into 6 smaller multiplication problems of size $\frac{n}{3}$, and multiplying by powers of 10 takes $O(n)$ time, and the addition operations take $O(n)$ time, the algorithm's recurrence relation is

$$T(n) = 6T\left(\frac{n}{3}\right) + O(n)$$

By the Master Theorem Case 1, the running time of this algorithm is $\Theta(n^{\log_3 6}) \approx \Theta(n^{1.631})$. Since the running time of splitting each number into two parts is only $\Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$, we would rather split each number into two parts instead of three parts because the running time of the two-part algorithm is faster.

c) Since we are splitting the problem into 5 smaller multiplication problems of size $\frac{n}{3}$, and again, since multiplying by powers of 10 and the addition operations all take $O(n)$ time, the algorithm's recurrence relation is

$$T(n) = 5T\left(\frac{n}{3}\right) + O(n)$$

By the Master Theorem Case 1, the running time of this algorithm is $\Theta(n^{\log_3 5}) \approx \Theta(n^{1.465})$, which is faster than the two-part algorithm ($\Theta(n^{1.585})$), so we would rather split each number into three parts.