CSCE 586 Take Home Midterm Exam

Marvin Newlin

8 November 2018

Collaborators: Andrew Watson, Josiah Watson, Nick Echeverry, Terrence Yi, Matsui

1. Give an $O(|V|+|E|)$-time algorithm to remove all the cycles in a directed graph $G = (V, E)$.
   Removing a cycle means removing an edge of the cycle. If there are k cycles in G, the algorithm should only remove $O(k)$ edges. You should try to make your algorithms as efficient as possible.

   (1) (5 points) Provide a clear description of the algorithm (an English description or pseudo-code is fine)
   Solution:
   Since we need an $O(|V| + |E|)$-time algorithm, DFS immediately comes to mind. Since DFS produces a tree we know that it will be free of cycles. Thus, our algorithm is DFS in a directed graph with some modifications.

   Note: This algorithm does DFS on the graph G and also maintains a color. The values for the colors are the following:

   White: Unvisited nodes
   Gray: Current node we are doing DFS on and all of its descendants.
   Black: Nodes we have finished processing.

   Assumption: The graph G is simple (there are not multiple directed edges (v, w) for some pair of nodes $v, w \in G$.

   Description:
   Mark every node in G white. Then while there are still nodes colored white do the following:
   Pick a node v, and perform DFS on that node. As we visit v and it's descendants in the recursive DFS call, mark it grey. If we encounter a node w that is already colored gray (there is a directed cycle), then delete that edge (v, w).

Once DFS returns to the original starting node v, color it and all it's descendants black.
Repeat for every node v that is still colored white.

Once all nodes are colored black, return the modified graph. [1,2]

(2) (5 points) Perform asymptotic analysis of your algorithms running time. Consider a best case, worst case, and average case input model scenario.

Solution:
Since our algorithm is a modified version of Depth First Search, we know that its runtime is $O(|V| + |E|)$ because it visits every vertex and every edge. This is the runtime for all cases best, average, and worst.
Additionally, our algorithm removes one edge from every cycle that it discovers, so if there are k cycles in the graph we only remove $O(k)$ edges.

(3) (5 points) Provide a proof that your algorithm works correctly.

Solution:
We know that DFS on a directed graph returns a directed tree, so there will be no cycles in the output of our returned graph. Additionally, we know that a cycle with n vertices contains n edges, so deleting one edge from the cycle removes the cycle.

Within the DFS search, if we encounter a node w that has already been colored gray it means that we have already visited that node and then left it and are now back at that node. This means that we have created a cycle so we delete the edge between our current node and the already gray node w and this removes the cycle for the reasons stated in the above paragraph. Encountering a black node in the DFS is not a problem because if the node is colored black it means that there was no directed edge from the black node to our current node so there was no cycle.

2. Chapter 5, Problem 4
We can model their structure as consisting of the points $1, 2, 3, ..., n$ on the real line; and at each of these points j, they have a particle with charge q_j. (Each charge can be either positive or negative.) The total net force on particle j, by Coulomb's Law, is equal to

$$F_j = \sum_{i<j} \frac{Cq_iq_j}{(j-i)^2} - \sum_{i>j} \frac{Cq_iq_j}{(j-i)^2}$$

Help them out by designing an algorithm that computes all the forces F_j in $O(n \log n)$ time.

(1) (10 points) Provide a clear description of the algorithm (an English description or pseudo- code is fine)

Solution:

Since this is a problem of vector multiplication, this problem can be broken down into a convolution. The trick is determining what the vectors should be that make up the convolution. Looking at the calculation itself we see that we can factor out a Cq_j from each term (since j is constant with respect to the sum) and it yields

$$F_j = Cq_j(\sum_{i<j} \frac{q_i}{(j-i)^2} + \sum_{i>j} \frac{-q_i}{(j-i)^2})$$

This allows us to determine what our vectors should be. Our first vector $a = (q_1, ..., q_n)$ and second vector $b = (n^{-2}, (n-1)^{-2}, ..., 1, 0, -1, ..., -(n-1)^{-2}, -n^{-2})$ We get the values of b from every possible combination (positive and negative) of the distance between i and j.

The number of elements of a is n and the number of elements of b is $2n-1$ so the convolution of these vectors $a * b$ contains about $O(3n)$ elements.

(2) (10 points) Perform asymptotic analysis of your algorithms running time. Consider a best case, worst case, and average case input model scenario.

Solution: Since we know that this is now a convolution, we know that we can calculate the convolution of 2 vectors in $O(n \log n)$ time (5.15 in [2]).

To calculate the forces we again have to iterate through the n points and multiply them by the elements of the convolution but this is an $O(n)$ operation so the overall runtime in all cases remains $O(n \log n)$.

(3) (5 points) Provide a proof that your algorithm works correctly.

Solution: Since we can turn this into a convolution, we know that we can break it down from $O(n^2)$ to $O(n \log n)$ based on the proof of convolutions (p. 234-245 in text). The n in the vector b is the distance between i and j in the sum and then the negative elements of the sum are provided by multiplying the negative sections of vector b by vector a. We then iterate through the convolution vector and multiply each element by Cq_j to get each element of the force vector.

3. Chapter 6, Problem 9

Setup:

x_i: Amount of data to be processed on day where i, $i = 1, ..., n$

s_j: Amount of data we can process j days after a reboot where j, $j = 1, ..., n$ and $s_1 > s_2 > ... > s_n > 0$.

(1) (10 points) part a

Give an example of an instance with the following properties. – There is a "surplus" of data in the sense that $x_i > s_1$ for every i.
– The optimal solution reboots the system at least twice. In addition to the example, you should say what the optimal solution is. You do not need to provide a proof that it is optimal.

Solution:

Let $x = (500, 500, 500, 500, 500)$ and $s = (100, 25, 5, 2, 1)$. Clearly, this satisfies the condition that $x_i > s_1, \forall i$. The optimal solution is the following: 100, reboot, 100, reboot, 100. Thus we also satisfy the condition that the optimal solution reboots twice or more.

(2) part b
i. (10 points) Provide a clear description of the algorithm (an English description or pseudo-code is fine)

Solution:

Examining the problem, we see that on each day i we have 3 cases.
Case 1, we choose the minimum of our data to process and our processing ability plus the optimum solution from the next day.
Case 2, (restart) we choose the optimum solution of the next day and the processing ability one day after reboot.
Case 3, we are after the n^{th} day so we don't process anything.

Our optimal solution has to be optimized with respect to i and j so we can express our optimal solution mathematically as

$$OPT(i,j) = \max \begin{cases} \min(x_i, s_j) + OPT(i+1, j+1) & \\ OPT(i+1, 1) & \\ 0 & \text{if } i, j > n \end{cases}$$

To do this with dynamic programming, we generate a matrix M that stores these entries of each OPT solution for each i and j. Then as we search for the optimal value, if it has been calculated, we just access it. Otherwise we calculate it and populate the table.

```
Algorithm

OPT(i, j)

IF i or j > n

        Return 0

IF M[i, j] exists

        Return M[i, j]

ELSE

        a ← min(x_i, s_j) + OPT(i+1, j+1)

        b ← OPT(i+1, 1)

        M[I, j] = max(a, b)

        Return M[i, j]
```

ii. (10 points) Perform asymptotic analysis of your algorithms running time. Consider a best case, worst case, and average case input model scenario.
Solution: Since this is a dynamic programming problem we have to think about space as well as time complexity. Our algorithm is a recursive one. Beginning with x_1 and s_1. As we go through our algorithm we recurse all the way until we get to $i = n$ and $j = n$ (depending on reboots) and then we begin to fill out the table.

Since we have to fill an $n \times n$ table we have to use $O(n^2)$ space. Since our algorithm fills out the $n \times n$ table then its time complexity in all cases is $O(n^2)$.

iii. (5 points) Provide a proof that your algorithm works correctly.
Solution: Our goal is to maximize the amount of data we can process in total. At each step $OPT(i, j)$ we maximize one of three elements. Either we take the minimum of (x_i, s_j)+ OPT(next day, next processing capacity), OPT(next day, processing capacity after reboot), and 0. If we end up with a solution that is not optimal, then at some step we chose one of the three options that did not maximize our total processing ability on a given day. But the algorithm chooses the maximum processing ability on each day so this is a contradiction.

4. Chapter 6, Problem 12
Setup:
n servers, $s_1, ..., s_n$. Placement cost c_i for each server s_i that a file is placed at. Access cost of $j - i, j > i$ for server s_j that file is found at. We want to find a configuration of servers $(s_1, ..., s_{n-1})$ that minimizes total cost (access and placement)

(1) (10 points) Provide a clear description of the algorithm (an English description or pseudo-code is fine)
Solution: Suppose that $OPT(j)$ is the optimal solution at server j, assuming that we put a copy of the file at server s_j. Then we have to find the next highest server to place the server at, server s_i. Assume that we behave optimally up to server s_i, so then the total cost is given by the cost for optimal behavior up to s_i plus the access cost for servers $i+1, ..., j$ plus the placement cost at s_j. The access cost for servers $i+1, ..., j$ is

$$0 + 1 + ... + (j - i - 1) = \begin{pmatrix} j-i-1 \\ 2 \end{pmatrix}$$

We can then determine the total minimal cost at s_j as follows:

$$OPT(j) = c_j + \min_{0 \leq i < j} (OPT(i) + \begin{pmatrix} j-i-1 \\ 2 \end{pmatrix})$$

Additionally, define $OPT(0) = 0$ and $\begin{pmatrix} 1 \\ 2 \end{pmatrix} = 0$ We then build a table T and store the entries at each level j. Since we need an ending point as well we set $T[n] = c_n$ [3].
(2) (10 points) Perform asymptotic analysis of your algorithms running time. Consider a best case, worst case, and average case input model scenario.
Solution: At each j in our algorithm we check the cost of placement at each of the i servers between 0 and j. This gives us a runtime of $O(j)$ for each iteration. Since we have n iterations our runtime then becomes

$$\sum_{j=1}^{n} O(j)$$

This is the sum of the first n numbers which is well known as $\frac{n(n-1)}{2}$ so we end up with a total runtime of $O(n^2)$ in all cases. In order to get $OPT(n)$, we then iterate through our table to find the configuration.

(3) (5 points) Provide a proof that your algorithm works correctly.
Solution: Suppose that we end up with a configuration that is not of minimum total cost. Then at some step j, we did one of two things.
Case 1: We placed a copy of the file at the server when it cost more than the access cost.
Case 2: We chose not to place a copy of the file when it actually would have been cheaper to.

If we have case 1, we get the situation that our optimization of all the servers up to i plus the access costs up to j was not minimized as dictated by the algorithm. Thus we cannot have case 1.

If we have case 2, then we chose some server k between 0 and j such that there was another value i that had a lower total cost, so again we didn't follow the algorithm. Therefore case 2 can't happen either.

5. References

[1] "Detect Cycle in a directed graph using colors - GeeksforGeeks." [Online]. Available: https://www.geeksforgeeks.org/detect-cycle-direct-graph-using-colors/. [Accessed: 07-Nov-2018].

[2] J. Kleinberg and E. Tardos, Algorithm Design. Pearson, 2006.

[3] "Algorithm Tutorial Sheet." [Online]. Available: http://www.cse.iitd.ernet.in/ amitk/SemI-2017/tut6.pdf. [Accessed: 07-Nov-2018].