CSCE 586 Take Home Final Exam

Marvin Newlin

11 December 2018

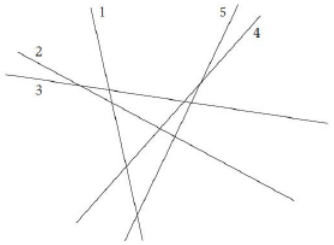
Collaborators: Andrew Watson

1. Problem 1

Hidden surface removal is a problem in computer graphics that scarcely needs an introduction — when Woody is standing in front of Buzz you should be able to see Woody but not Buzz; when Buzz is standing in front of Woody, ... well, you get the idea.

The magic of hidden surface removal is that you can often compute things faster than your intuition suggests. Here's a clean geometric example to illustrate a basic speed-up that can be achieved. You are given n non-vertical lines in the plane, labeled $L_1, \ldots, L_n$, with the i^{th} line specified by the equation $y = a_i x + b_i$. We will make the assumption that no three of the lines all meet at a single point. We say line L_i is *uppermost* at a given x-coordinate x_0 if its y-coordinate at x_0 is greater than the y-coordinates of all the other lines at x_0: $a_i x_0 + b_i > a_j x_0 + b_j$ for all $j \neq i$. We say line L_i is *visible* if there is some x-coordinate at which it is uppermost — intuitively, some portion of it can be seen if you look down from "$y = \infty$."

An example with five lines (labeled $1 - 5$ is shown here with all lines except for 2 are visible.



(1) (2 points) Which algorithmic paradigm will you use to solve this problem?

A. Brute force
B. Greedy
C. Divide and Conquer
D. Dynamic Programming
E. other: _____

Sol'n: Greedy

(2) (3 points) Why did you choose the algorithmic paradigm selected above to solve this problem?

Sol'n: An approach that sorts based on slope and then performs local comparisons between the intersections in order of increasing slope seems to be the most straightforward approach, making this a greedy approach.

(3) (15 points) Give an algorithm that takes n lines as input, and in $O(n \log n)$ time returns all of the lines that are visible. Provide a clear description of the algorithm (an English description or pseudo-code is fine)

Sol'n:
Sort the list $L_1, ..., L_n$ by increasing slope so that $L_1 < ... < L_n$. Label the intersection of each line L_i with its immediate successor $L_j, j = i + 1$ so that we have a collection of intersections $X = (1, 2), (2, 3), ..., (n-1, n)$ with x_{ij} as the x-value of the intersection of (i, j). Then our collection has the property $(1, 2) < (2, 3) < ... < (n-1, n)$. However, if there is a k such that $(i, j) < (j, k)$ but $x_{jk} < x_{ij}$ then remove $(i, j)\&(j, k)$ from X and replace with (i, k). Each distinct element i in this final collection X is the list of visible lines. [2]

(4) (10 points) Perform asymptotic analysis of your algorithms running time. Also, consider the runtime performance of a best case, worst case, and average case input model scenario.

Sol'n: The runtime can be broken down into two categories, sort and search. The sort can be performed in $n \log(n)$ time with a function like merge sort. Our algorithm iterates through the list one full time plus the fraction $\frac{k}{n}$ that it iterates through to remove and replace intersections. This fraction becomes increasingly smaller as we replace it 2 or more elements with a single element. Thus, the runtime of the search part can be expressed as $n + \frac{k}{n} + \frac{k+j-k}{n} + ... \leq n + n = 2n$ Thus the total runtime of the algorithm is $O(2n) + O(n \log n) = O(n \log n)$. [2].

(5) (10 points) Provide a proof that your algorithm works correctly.

Sol'n: If our algorithm does not provide the correct output then one of two cases has happened. There is an line supposedly visible that is not or there is a visible line that is not included in our list. If the first case has happened then this line L_j meets the condition such that $(i, j) < (j, k)$ but $k_{ij} > x_{jk}$. But our algorithm removes lines that fit this condition and replaces them with the element (i, k) so this is a contradiction. On the other hand, if there is a line that is not represented in our list then it similarly had the condition L_j where $(i, j) < (j, k)$ but $k_{ij} > x_{jk}$. But this meas the line isn't visible so this is a contradiction since we assumed it was a visible line. Thus, our algorithm provides the correct output.

2. Problem 2

II. In a bipartite graph $G = (X \cup Y, E)$ a matching M is a subset of the edges from E chosen in such a way that such that $\forall e \in M$ one end is in X and the other in Y. A *maximum matching* is a matching with the maximum number of edges – if any edge can be added then it is no longer a matching. A given bipartite graph can have only one maximum matching. Consider the following algorithm for finding a matching in a bipartite graph:

> As long as there is an edge whose endpoints are unmatched, add it to the current matching M.

(1) (2 points) Which algorithmic paradigm best describes this algorithm?

A. Brute force
B. Greedy
C. Divide and Conquer
D. Dynamic Programming
E. other: _____

Sol'n: Greedy

(2) (3 points) Why did you choose the algorithmic paradigm selected above?

Sol'n: The step taken in each iteration of the algorithm is based solely upon a local choice (availability), so this makes it a greedy algorithm.

(3) (5 points) Give an example of a bipartite graph G for which this algorithm does not return the maximum matching

Sol'n: The figures below illustrate that the greedy algorithm does not always return a maximum matching.

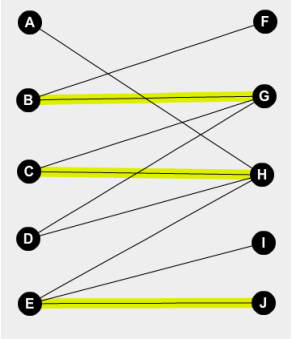


Figure 1: Possible maximal matching produced by greedy algorithm

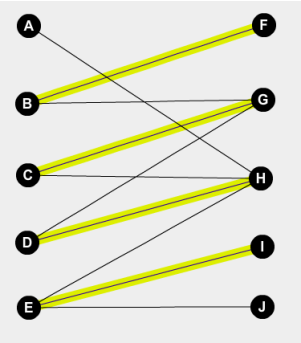


Figure 2: A maximum matching in this graph

(4) (5 points) Let M and M' be matchings in a bipartite graph G. Suppose that $|M'| > 2|M|$. Show that there is an edge $e' \in M'$ such that $M \cup \{e'\}$ is a matching in G.

Proof: Suppose that $|M'| > 2|M|$. Since there are more than 2 times the number of edges in M' than in M, then there are more than 2 times the number of vertices since each edge saturates two vertices. This means that there are at least two vertices saturated by M' but not M such that the edge $e' = (u', v') \in M'$. Thus, the edge $e' = (u', v')$ can be added to the matching M since it does not saturate any existing edges of M. Thus, $M \cup \{e'\}$ is also a matching.

(5) (5 points) Using the previous claim (and your supporting proof) to further prove that the algorithm is optimal or that the algorithm is ρ-optimal approximate (in this case be sure to derive the value of ρ as part of your proof).

Claim: The greedy algorithm above is a 2-approximation to a maximum matching.
Proof: Let M be a matching produced by the greedy algorithm and let M' be a maximum matching in the bipartite graph G. Since our algorithm terminated, we can't add any more edges to it so our matching M is maximal so $\nexists e' = (u, v)$ such that $M \cup \{e'\}$ is still a matching. Therefore, by the contrapositive of the statement proved above in (4), $|M'| \leq 2|M|$. Therefore, our matching M is a 2-approximation of a maximum matching.

3. Problem 3

III. (40 points) You and a friend have been trekking through various far-off parts of the world, and have accumulated a big pile of souvenirs. At the time you weren't really thinking about which of these you were planning to keep, and which your friend was going to keep, but now the time has come to divide everything up.

Here's a way you could go about doing this. Suppose there are n objects, labeled $1, 2, \ldots, n$, and object i has an agreed-upon *value* x_i. (We could think of this, for example, as a monetary re-sale value; the case in which you and your friend don't agree on the value is something we won't pursue here.) One reasonable way to divide things would be to look for a *partition* of the objects into two sets, so that the total value of the objects in each set is the same.

This suggests solving the following *Number Partitioning* problem. You are given positive integers $x_1, \ldots, x_n$; you want to decide whether the numbers can be partitioned into two sets S_1 and S_2 with the same sum:

$$\sum_{x_i \in S_1} x_i = \sum_{x_j \in S_2} x_j.$$

Show that *Number Partitioning* is NP-complete using the *Subset Sum* problem.

Solution: We first show that *Number Partitioning* is $\in NP$. Suppose we are given a certificate t of *Number Partitioning* $x_1, ..., x_n$ partitioned into two sets $S_1 = \{x_a, ...x_j\}$ and $S_2 = \{x_k, ..., x_p\}$, where $|S_1| + |S_2| = n$. Then simply by iterating through each list once and summing the value of each item we can check whether the values of the two sets are equal, i.e. $\sum_{x_i \in S_1} x_i = \sum x_j \in S_2 x_j$. Since we only check each list once and the size of the two lists together is n, then we can do this in $O(n)$ time. Therefore, *Number Partitioning* $\in NP$.

Claim: *Subset Sum* $\leq_p$ *Number Partitioning*
Proof: Suppose we have a black box to solve *Number Partitioning*. Additionally, suppose we have an instance of *Subset Sum* $w_1, ...w_n$ with target W. We show that we can transform this instance of *Subset Sum* into an instance of *Number Partitioning* if and only if we can transform an instance of *Number Partitioning* into an instance of *Subset Sum*.
To construct our instance of *Subset Sum*, take $S = \sum_{w_i} w_i$ and add $|S - 2W|$ as an $(n+1)^{st}$ element of our list of n items. Thus, our collection is now $\{w_1, ..., 2_n, |S - 2W|\}$, and the total sum of all of the elements in this set is $2S - 2W$ which is an even number. Feed this into the black box for *Number Partitioning* and accept if the black box for *Number Partitioning* accepts it.[2], [3]
This is a polynomial transformation since we can calculate the value $|S - 2W|$ and add it to the set $O(n)$ time.

Now, we show that $\{w_1, ..., w_n\}, W \in$ *Subset Sum* if and only if $\{w_1, ..., 2_n, |S - 2W|\} \in$ *Number Partitioning*. If there is a subset of $\{w_1, ..., w_n\}$ that sums to W, then the rest of that subset sums to $S - W$ and so then there is a valid partition of $\{w_1, ..., w_n\}$ that sums to $S - W = \frac{W}{2}$.
Conversely, if we can partition $\{w_1, ..., 2_n, \frac{W}{2}\}$ such that the value of each partition is $\frac{W}{2}$, then one partition has to be the element $\frac{W}{2}$ and the other partition has to be $\{w_1, ..., w_n\}$ which sums to $S = \frac{W}{2}$. Therefore, since *Subset Sum* is $NP - complete$ by thm 8.23 in [1], and since *Number Par-*

titioning $\in NP$ and *Subset Sum* $\leq_p$ *Number Partitioning*, *Number Partitioning* is $NP - complete$.

4. References

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

[2] A. Watson. "Final Exam Study Group." CSCE-586. Air Force Institute of Technology. Wright Patterson AFB. 10 Dec 2018. Discussion.

[3] M. Nakayama, "Homework 13 Solutions," CS 341: Foundations of Computer Science II. [Online]. Available: https://web.njit.edu/ marvin/cs341/hw/hwsoln13.pdf. [Accessed: 10-Dec-2018].