CMSC 27200 Assignment 5

Rachel Hwang

February 12, 2014

- **Exercise 6.8**

  a. Counterexample: Consider a sequence of robot attacks with robot counts $\{9, 3, 8, 6\}$ and an EMP discharge function with power (indexed by days charged) $\{3, 4, 5, 7\}$. This given algorithm will attempt to destroy all robots in the final wave, charging for 4 days to destroy 6 robots on the last day. The optimal strategy, however, is clearly to discharge once per wave, destroying 3 robots each time for a total of 12 robots destroyed.

  b. The maximum number of robots that can possibly by destroyed by time j is given by the following recurrence

$$opt(0) = 0$$
$$opt(j) = \max_{i<j}(\min(x_j, f(j-i)) + opt(i))$$

  Proof: By induction. The base case is trival, since $opt(1)$ is obviously equal to $(\min(x_1, f(1)) + opt(0) = min(x_1, f(1))$. Now consider $opt(k)$ where k is the final day we consider. We will of course fire the emp since we have no reason to save charge. However, the discharge power will depend on when we last fired, therefore we want to maximize a combination of charge power and robots destroyed in total at the time of the last discharge. We must consider all possible days before k as possible discharge days. This gives us precisely the expression for *opt* defined above.

  Complexity: Using memoization, we need only compute each $opt(i)$ once. Since the algorithm takes at most n time to compute the maximum at each of n calls to *opt*, it runs in $O(n^2)$ time.

- **Exercise 6.15**

  a. Counter example: consider a set of 8 events with possible locations in {0,1,2,3,4}, taking place in the location sequence [4,4,4,4,0,0,0,0].

  The given algorithm will move the telescope to location 4 to catch the earliest legal event e_4. After observing this event, the telescope moves back to location 0 to catch the last event, resulting in a total of two events seen. However, the optimal solution is clearly just to remain at location 0 for the entire sequence, resulting in 4 events seen.

  b. Consider that we must see the nth event. We want a way to determine, based on the last event we will observe, what the penultimate observed event is. Assume that we observe some event j. In order to observe some event i, it must be the case that $|d_j - d_i| \leq j - i$, since we cannot observe j after i if that does not hold. It also must be that $|d_0 - d_i| \leq i$, since if that does not hold, i is not acessible from the starting point. And of course $i < j$. Given these criteria, we can design a function which returns the maximum number of events possible to observe as follows

$$opt(j) = 1 + \max_{i<j}(opt(i))$$

  Where i is any event before j which meets all the stated criteria. Essentially, at each step, we are choosing the best possible valid event, begining with the last event n.

  This algorithm is correct using induction. At each step we must choose one event to add to the list, and this algorithm ensures that we make the optimal choice. Using memoization, this algorithm runs in $O(n^2)$ time, since we calculate $opt(j)$ for all j.

- **Exercise 6.24**
  If we can create a set of $n/2$ precincts in which one party A wins by a minimal margin, we know that if A does not win in the other district, it is *impossible* to gerrymander a win for A, since swapping any winning precincts to the other district will cause A to lose in the original district. Thus, in order to solve this problem, we want to find the minimal win margin in a group of k precincts for some party A. In order to do this, we can define a function *opt*, which returns the minimum difference of voters in parties A and B greater than some number x. The relevent arguments for *opt* to consider are the number of districts k, an index i of the highest district index it considers, and x, the margin to beat. The recurrence is as follows:

$$opt(0, i, x) = \begin{cases} 0 & \text{if } x < 0 \\ \infty & \text{if } x \geq 0. \end{cases}$$

$$opt(k, i, x) = min \begin{cases} opt(k, i-1, x) \\ d_i + opt(k-1, i-1, x - d_i) \end{cases}$$

$$opt(k, i, x) = \infty \text{ if } k > i$$

  Now for the full algorithm: we can easily calculate the total number of voters in each party. We need only consider the party with a majority, since it would be impossible for a party without an overall majority to benefit from gerrymandering. Let's say party A holds an overall majority. Given the *opt* function defined above, we can simply calculate $opt(n/2, n, 0)$, the smallest win margin for A for a group of $n/2$ precincts drawings from any of the total n precincts. We may then subtract this number from the total number of voters in A. If the resulting number of voters is positive, A can benefit from gerrymandering. Otherwise, neither party can.

  This algorithm will run in $O(mn^3)$ time with memoization. We need to calculate the *opt* values for every combination of arguments involved in calculating $opt(n/2, n, 0)$, which will be $O(n/2 \cdot n \cdot 2mn) = O(mn^3)$.

- **Exercise 7.6**
  This problem can be solved by reimagining the bulbs and switches as nodes in a bipartite graph. All the bulbs will be in one group, and all the bulbs will be in the other. We may then add an edge from each switch to each lightbulb so long as the bulb is visible from the lightswitch. We can define visibility between a switch as bulb as able to have an edge which does not intersect any of the walls. Thus, the algorithm is as follows:

  For each switch, add an edge to each bulb visible from that switch.
  Take the resulting graph and run the bipartite perfect matching algorithm on it. Return the number of matchings.
  If number of matchings is equal to the number of switches n, then each bulb is paired with a visible switch, and the house is ergonomic. Otherwise, the house is not ergonomic.

  Complexity: We must first find all edges, checking each edge for visibility from each switch. This runs in $O(mn^2)$ time, since we must for each of n switches find an edge to each of n lightbulbs. For each of these prospective edges, we must check each of m walls. After adding the edges to the graph, we must run a perfect matching algorithm which the book gives to be $O(n^3)$. The total complexity of the algorithm is $O(mn^2 + n^3)$, meaning that the complexity is determined by whether we have more walls than switches.

- **Exercise 7.9**
  As in the previous exercise, we can rephrase this problem in terms of a bipartite graph: we imagine every hospital has having $n/2$ berths, where each berth can hold a single patient. We have in total n patients and n berths, and want to find a perfect matching between them. We begin by adding to the bipartite graph an edge between every berth and every patient, but we only add this edge if the patient lives within 30 minutes of the hospital that berth is from.

  If we can check whether a patient lives in range of a hospital in constant time, the construction of this graph will take a maximum of $O(n^2)$, since we must check n berths for each of n people.

  Once we have constructed the graph, we can again run the bipartite matching algorithm to find the maximum number of perfect matches, running in $O(n^3)$ time. Thus, the algorithm runs in $O(n^3)$ total.