CS 180 - Algorithms and Complexity
Homework 6

Jayant Mehra
Discussion 1D

November 29, 2018

Solution 1. The following is the solution to problem 16 on page 327 of the textbook.

For any node in the tree T, the number of rounds and order from there depends on the subtree of T, T' rooted at the particular node. Thus, for the i^{th} node, the recurrence for the number of rounds is the following:

$$\text{OPT}(i) = \begin{cases} 0 & i \text{ is a leaf node} \\ max(\bigcup_{j=1}^{n} \text{OPT}(j) + j) & \text{otherwise (where } n = \text{\# of children of } i^{th} \text{ node)} \end{cases}$$

Thus, we can start from the leaves and make our way up using the recurrence. The algorithm follows the dynamic programming paradigm because instead of recursing for each node, we start from the leaf and build our way up. For each node, we just look at its children and calculate the number of rounds for that node. At the end, we return OPT(1).

For the optimal sequence, we do the following. For the root node of any subtree T', we go through each of the child in decreasing order of number of rounds it requires (breaking ties arbitrarily), and keep appending everything in the correct order. For example, the ordering for greatest round child can be appended to an empty string. For the second greatest child, we append it starting from the second turn in the current string i.e. we put the first round of this child's ordering in the current string's second round slot and so on. We continue this process for all child nodes each time appending the ordering starting from one position up to obtain the optimal solution.

Algorithm Runtime Analysis For each node in the tree we only look at its neighbors and there can be at most $n - 1$ neighbors (edges) in a tree with n nodes. Therefore, assuming that we have a sorted list of children according to criterion given above, the algorithm takes $\text{O}(n)$ time. If we need to sort, then the sort dominates the run time and makes the algorithm $\text{O}(n \log n)$

Solution 2. The following is the solution to problem 21 on page 330 of the textbook.

We can model the problem as a recurrence where the recurrence depends on the maximum number of days allowed, t, and the current day, i.

$$\text{OPT}(t,i) = \max(\text{OPT}(t-1,i),\ \max(\bigcup_{j=1}^{i-1} \text{OPT}(1000\text{*price}(i) - 1000\text{*price}(j) + \text{OPT}(t-1,j)))$$

The first term in the recurrence is when there is no transaction on the i^{th} day and the optimal solution depends on one less transaction on that day. The second term represents a transaction on the i^{th} day with some other day j wherein the **constraints of increasing price values is maintained** i.e only those new transactions will be considered whose buying price is greater than the selling price of the previous transaction and needless to say the selling price of the new transaction is greater than its buying price.

The values can easily be cached in a 2D table for a DP algorithm. The algorithm builds the table in a top down manner since t, i depend on previous values. The base cases are 0 when $t = 0$ i.e there are no transactions and when we look at the 0^{th} day. We finally return $\text{OPT}(k, n)$. This gives us the maximum profit

To find the actual pairs, we can keep track of where each optimal value comes from using links. Whenever it comes from some transaction made on a particular day, we can link that to the previous transactions. Thus, by backtracking we can find the exact m pairs for the optimal solution.

Algorithm Runtime Analysis The algorithm fills a $k \times n$ table. For each cell in the table, it iterates through at most n entries. Thus, the runtime is $\text{O}(kn^2)$.

Solution 3. The following is the solution to problem 24 on page 331 of the textbook.

The problem can be broken down into subproblems each of which depend on three parameters - number of precincts= i, total number of precinct subsets= j for which we have s total votes.

The total votes s is bounded by the following inequalities : $\frac{mn}{4} < s < \text{Votes(A)} - \frac{mn}{4}$ for a majority.

Thus, the recurrence looks like:

$$\text{OPT}(i,j,s) = \begin{cases} True & \text{if } i = 1, j = 1, s = a_1 (\text{Votes of A in 1}) \\ True & \text{if OPT}(i-1, j-1, s-a_i) = True \\ True & \text{if OPT}(i-1, j, s) = True \\ False & \text{Otherwise} \end{cases}$$

Thus, we need to maintain a table of size $n \times \frac{n}{2} \times$ Votes(A) $-\frac{mn}{4} - 1$ and fill it from OPT(1, 1, a_1) since values depend on the previous values in the recurrence. **At the end we check for all values in the "row" OPT($n, \frac{n}{2}, s$)** since there are a total of n precincts and we need to win in $\frac{n}{2}$ and we need a total of s votes which is bounded by the inequalities above. We are only looking for one Truthy value, thus, even if we find one True for all the s values, we can safely return True.

Algorithm Runtime Analysis We fill a 3D array and each subproblem i.e. cell requires constant time lookups, thus, our runtime is bounded by O(mn^3).

Solution 4. The following is the solution to problem 6 on page 416 of the textbook.

The problem can be solved by modeling the switches and fixtures into a bipartite graph and running the Ford-Fulkerson algorithm to calculate the maximal flow through the network.

Let a light fixture be represented by $l_i \in L$ and a switch by $s_i \in S$. Let L and S be the two disjoint sets of vertices in the bipartite graph. Let there be an edge between a light, l_i, and a switch, s_j, if and only if our subroutine returns that they are visible to each other (we can compare the walls with the line segment joining the light and the switch). Let the capacities of all connections be unity.

We add a source vertex that connects to all nodes $\in S$ and a sink node that has an edge coming from all nodes $\in L$. Let the capacities of these edges be also 1.

We now run the Ford-Fulkerson algorithm to determine the maximum flow through the network. Let the maximum flow be F. If F is equal to n, then that means that there is a flow through every switch to a unique light fixture since the capacity of the edge was 1. Thus, the plan is ergonomic according to the given criterion. Otherwise, it is not.

Algorithm Runtime Analysis It takes O(n^2) to build the bipartite graph. The Ford Fulkerson algorithm runs in O(Δe) where $e =$ O(n^2) edges. Thus, our algorithm is polynomial in its runtime.

Solution 5. The following is the solution to problem 8 on page 418 of the textbook.

(a)Again, we can model our problem as a bipartite graph and run the Ford-Fulkerson algorithm to evaluate of there is enough blood for everybody.

Let the first set in the bipartite graph, G, by {O,A,B,AB} and the second set of G should also consist of the same labelled nodes. There will be an edge between element of set 1 and an element of set 2 iff the first element can donate to the second element. Let the capacities of these edges be ∞. Add a source node that has an edge to every element in the first set

and each edge has a capacity s_O, s_A, s_B, s_AB respectively. Add a sink node that has an edge from each element in the second set with capacities d_O, d_A, d_B, d_AB respectively.

We now run the Ford-Fulkerson algorithm to determine the maximum flow through the flow network. If it is equal to the total demand $(d_O + d_A + d_B + d_AB)$, then the demand is met.
It takes constant time to build the network. The Ford-Fulkerson algorithm runs in $O(\Delta e)$ time where e is the number of edges which is also always constant. Thus, the run time is polynomial.