Submission to Problem Set 6

Name: *Deepanshu Parihar*

Problem 1. (7 points) Project management

Suppose you are a high-level manager in a software firm and you are managing n software projects. You are asked to assign m of the programmers in your firm among these n projects. Assume that all of the programmers are equally competent.

After some careful thought, you have figured out how much benefit i programmers will bring to project j. View this benefit as a number. Formally put, for each project j, you have computed an array $A_j[0..m]$ where $A_j[i]$ is the benefit obtained by assigning i programmers to project j. Assume that $A_j[i]$ is nondecreasing with increasing i.

Design a dynamic programming algorithm to determine how many programmers you will assign to each project such that the total benefit obtained over all projects is maximized. Analyze its running time.

Answer:
Let OPT[i,j] be the optimal benefit obtained for taking i programmers for project j.
Base case is-
if programmers are 0 and projects are not, then benefit is the sum of A[0] for all values of j.

Recursion is-
OPT[i,j]=max(Aj(k)+OPT(i-k,j-1)) for all values of k lying between 0 and i.

Algorithm-
We make another array K[i,j] to keep track of programmers assigned to each project
for j=0 to n
. for i=0 to m
OPT[i,j]=max(Aj(k)+OPT(i-k,j-1)) for all values of k lying between 0 and i.

Change value of K[i,j] to K[k,j].
So iterating through the loops takes O(mn) time and and to find the assignments of programmers to the project we iterate through K and calculate at max for m enteries so we get a total time of $O(m^2n)$

Problem 2. (7 points) Organizing congressional districts

The midterm elections are just over, and it was a really close election. As the neutral Election Commissioner for the state of Dynamica, you are worried that the two parties Maroon and Gray will indulge in reorganizing the precincts so as to make favorable maps.

There are n precincts $P_1, P_2, \ldots, P_n$, each containing m registered voters. You also find out the number m_i of registered Maroon voters and g_i of registered Gray voters in precinct P_i. We say that the precincts are *vulnerable* if there is a way to split the n precincts into two districts, each containing $n/2$ precincts in such a way that the same party holds a majority in both districts (assume n is even).

Give a dynamic programming algorithm to determine whether the given precincts are vulnerable. Your algorithm must take time polynomial in the total number of voters in the n precincts.

Answer:

Problem 3. (6 points) Planning a company party

You are consulting for a corporation that is planning a company party. The company has a hierarchical structure that can be captured as a tree rooted at the president. Thus, every node of the tree represents an employee and the parent $\pi(v)$ of any node v (except the root) represents the immediate supervisor of the person. The personnel office has ranked each employee v with a personality rating $\sigma(v)$, which is a nonnegative integer. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.

Given the tree structure of the company hierarchy and the personality ratings of each person, describe a dynamic programming algorithm to make up a guest list that maximizes the sum of the personality ratings of the guests in the list. Analyze the time complexity of your algorithm in terms of the number of employees.

Answer: This is similar to the vertex cover problem that we did in class.
So we take
OPT[v]= Personality rating of optimal solution for tree rooted at V.
There are two cases if vertex is part of the optimal solution or not.
Recursion is -
OPT[V]= Max(OPT[V]=$\sigma(V) + \sum OPT(x)$, where x is grandchildren of v,
$\sum$ OPT(c) , where c is children of v)
Base Case- if there is only one person in the company then we take it so σ of that one vertex

Algorithm-
0.Create a list guest initializing all vertices to be False
1. View the tree as directed graph with edges from vertex v to its parent.
2.Run topological sort
3.for v in topological sort
4.if v is leaf OPT[v]= $\sigma(V)$
5.else
OPT[V]= Max(A = OPT[V]=$\sigma(V) + \sum OPT(x)$, where x is grandchildren of v,
B=$\sum$ OPT(c) , where c is children of v) and
6.If A evaluates to be true then evaluate guest[vertex]=True. Else B automatically calls OPT[V] again.

Since topological sort is being used the complexity is O(e+v)