Pablo Acuna
CSCI 4020
Anshelevich

Computer Algorithms Homework 4

16. There are many sunny days in Ithaca, New York; but this year, as it happens, the spring ROTC picnic at Cornell has fallen on a rainy day. The ranking officer decides to postpone the picnic and must notify everyone by phone. Here is the mechanism she uses to do this.

Each ROTC person on campus except the ranking officer reports to a unique superior officer. Thus the reporting hierarchy can be described by a tree T, rooted at the ranking officer, in which each other node v has a parent node u equal to his or her superior officer. Conversely, we will call v a direct subordinate of u. See Figure 6.30, in which A is the ranking officer, B and D are the direct subordinates of A, and C is the direct subordinate of B.

To notify everyone of the postponement, the ranking officer first calls each of her direct subordinates, one at a time. As soon as each subordinate gets the phone call, he or she must notify each of his or her direct subordinates, one at a time. The process continues this way until everyone has been notified. Note that each person in this process can only call direct subordinates on the phone; for example, in Figure 6.30, A would not be allowed to call C.

We can picture this process as being divided into rounds. In one round, each person who has already learned of the postponement can call one of his or her direct subordinates on the phone. The number of rounds it takes for everyone to be notified depends on the sequence in which each person calls their direct subordinates. For example, in Figure 6.30, it will take only two rounds if A starts by calling B, but it will take three rounds if A starts by calling D.

Give an efficient algorithm that determines the minimum number of rounds needed for everyone to be notified, and outputs a sequence of phone calls that achieves this minimum number of rounds.

Subproblems: Our subproblems will all subtrees $T^{'}$ in T. With this we will define OPT($T^{'}$) to be the minimum number of rounds for everyone in $T^{'}$ to be notified once the root has the message. Define T_1, ..., T_k to be all the child subtrees of $T^{'}$. We will also say that we sorted the child subtrees such that OPT[T_1] $\geq$, ..., $\geq$ OPT[T_k]

Recursion Relation: OPT($T^{'}$) = $min_{1 \leq j \leq k}$ [j + OPT(j)]

This is a correct relation because are looking at the subtrees direct childern looking through all of them finding which one has the smallest sum of its optimal solution and sorted j value which is used to determine which the order in which each should get called first because biggest subtree.

Algorithm: The base case for this problem will be the leaves, all leave node subtrees solution will be: OPT[$T^{'}$] = 0. Building from the base case, we will traverse our way up the tree to the root. The solution will be the entire tree OPT[T]. To obtain the sequence we can recursively trace back through the sorted order and print the list of rounds.

Analysis: Since there are n nodes in the tree, there are n subtrees, and therefore n subproblems. To compute each subproblem we must sort first which takes O(nlogn). And to obtain the optimal solution we have to iterate at most n-1 previous solutions, therefore the sorting is larger and the total runtime is O(n^2logn).

Pablo Acuna
CSCI 4020
Anshelevich

Computer Algorithms Homework 4

24. Gerrymandering is the practice of carving up electoral districts in very careful ways so as to lead to outcomes that favor a particular political party. Recent court challenges to the practice have argued that through this calculated redistricting, large numbers of voters are being effectively (and intentionally) disenfranchised.

Computers, it turns out, have been implicated as the source of some of the "villainy" in the news coverage on this topic: Thanks to powerful software, gerrymandering has changed from an activity carried out by a bunch of people with maps, pencil, and paper into the industrial-strength process that it is today. Why is gerrymandering a computational problem? There are database issues involved in tracking voter demographics down to the level of individual streets and houses; and there are algorithmic issues involved in grouping voters into districts. Let's think a bit about what these latter issues look like.

Suppose we have a set of n precincts P_1, P_2, . . . , P_n, each containing m registered voters. We're supposed to divide these precincts into two districts, each consisting of n/2 of the precincts. Now, for each precinct, we have information on how many voters are registered to each of two political parties. (Suppose, for simplicity, that every voter is registered to one of these two.) We'll say that the set of precincts is susceptible to gerrymandering if it is possible to perform the division into two districts in such a way that the same party holds a majority in both districts.

Give an algorithm to determine whether a given set of precincts is susceptible to gerrymandering; the running time of your algorithm should be polynomial in n and m.

Subproblems: OPT[j, p, x, y] = true if it is possible to have at least x Party A voters in distict 1 and at least y Party A voters in district 2, while using p of the 1 through j precincts for district 1. If it is not possible then it is false. Lets also define the array of registered Party A voters for each precinct to be A_1, ..., A_n. This will allow us to make the recursion relation.

Recursion Relation: OPT[j, p, x, y] = OPT[j-1, p-1, diff(x-A_j), y] or OPT[j-1, p, x, diff(y-A_j)]

```
diff(n):
    if n ≥ 0
        return n
return 0
```

This is a correct relation because we have two choices for adding a precinct, we either add it to distict one or district two. If we add it to distict 1 we check if it was possible for the p-1 precincts to have x-A_j voters and the rest have y voters. Or we will look at not including it in distict 1 which is the opposite as the first option.

Algorithm: The base cases will be OPT[j, p, 0, 0] for all j and p, in this case it is possible to have 0 Party A members in both districts therefore, OPT[j, p, 0, 0]=true. All other cells will be false. We can build this up in increasing j,p,x,y. They all start at 1 and their loop ends at n, j, mn/4, and mn/4 respectively. The solution will be at OPT[n, n/2, nm/4, nm/4]. That is because if it is true then we have at least half of each district is party A voters. Therefore we have gerrymandering. We can also run this with Party B afterwards.

Analysis: We have n^4m^2 subproblems and from the relation takes constant time to compute. Therefore our runtime is O(n^4m^2)