Problem 1-1 Restaurant Location

Drunken Donuts, a new wine-and-donuts restaurant chain, wants to build restaurants on many street corners with the goal of maximizing total profit.

The street network is described as an undirected graph $G = (V, E)$, where the potential restaurant sites are the vertices in the graph. Each vertex u has a nonnegative integer value p_u, which describes the potential **profit** of site u. Two restaurants cannot be built on adjacent vertices (to avoid self-competition). Design an algorithm that outputs the chosen set $U \subseteq V$ of sites that maximize the total profit $\sum_{u \subseteq U} p_u$.

1 Consider the following greedy strategy. Choose the highest profit vertex u_0 in the tree (breaking ties according to some order on vertex names) and put it into U. Remove u_0 from further consideration, along with all of its neighbors in G. Repeat until no further vertices remain. Give a counterexample to show that this algorithm does not always give a restaurant placement with the maximum profit.

Solution: Consider the counter example $(1, 1, 1, 2)$ with the 2 connected to the ones. The strategy would output 2, when the maximum profit is 3.

2 Give an efficient algorithm to determine a placement with maximum profit.

Solution: Pick any vertex, denote it as u_0, traverse the tree with DFS and sort the vertices based on their completion times into an array N. Children nodes appear before their parents. For each vertex, create two functions, $A(v)$ and $B(v)$ which represent the maximum profit at the subtree rooted at v including and excluding v.

For a leaf node v

$A(v) = p_v$, and $B(v) = 0$

For non-leaf nodes

$A(v) = p_v + \sum_{u \in v.children} B(u)$

$B(v)$ is $\sum_{u \in v.children} \max(B(u), A(u))$

For each node in N, compute $A(v_i)$ and $B(v_i)$. The final node corresponds to the root of the subtree, at which point the maximum strategy is $\max(A(v_n), B(v_n))$.

Correctness: Consider the proof via induction.

In the base case with one node v, we know that the maximum value is p_v which for a leaf node is $\max(A(v), B(v)) = \max(p_v, 0) = p_v$.

Inductive hypothesis: consider the situation where we have $n + 1$ nodes. There are two combinations to consider, including and excluding the additional node u. The maximum profit must be one of the two combinations. N guarantees that children nodes are processed before their parents, so the $A(v)$ and $B(v)$ are accurate.

Timing: The algorithm runs in $O(V)$ time. Trees have $\leq V-1$ edges. Therefore, traversing G and sorting the nodes by their reverse finishing times takes $O(V)$ time. The time it takes to find $A(v)$ and $B(v)$ is also bounded by the $O(V)$ number of edges from each vertex. Therefore, the entire algorithm runs in $O(V)$.

3 Suppose that, in the absence of good market research, DD decides that all sites are equally good, so the goal is simply to design a restaurant placement with the largest number of locations. Give a simple greedy algorithm for this case, and prove its correctness.

Solution: As previously, perform a DFS search of the tree, and put the nodes into an array N by finishing time. When considering a node, add it, and remove its parent from N. Iterating over N finds the maximum profit.

Correctness: Children appear before parents in N. Therefore, leave nodes will be added first, removing their parents. Each iteration over N adds either a leaf node in the existing subtree, or skips over a parent node that's been removed. In any optimal selection, it's possible to create another optimal selection by removing a parent node, and swapping it with a leaf node. Therefore, our method of selecting the leaf nodes of the remaining subtree guarantees that our solution is optimal.

4 Now suppose that the graph is arbitrary, not necessarily acyclic. Give the fastest correct algorithm you can for solving the problem.

Solution: The cyclic nature of the graph breaks our previous algorithms. The problem is equivalent to the Maximum Independent Set Problem which is NP-Complete. Therefore, the best we can do is to try every combination of vertices, and for each combination, track its profit. The algorithm would run in $O(2^V|E|)$ time.

Problem 1-2 Radio Frequency Assignment

Prof. Wheeler at the Federal Communications Commission (FCC) has a huge pile of requests from radio stations in the Continental U.S. to transmit on radio frequency 88.1 FM.

The FCC is happy to grant all the requests, provided that no two of the requesting locations are within Euclidean distance 1 of each other (distance 1 might mean, say, 20 miles). However, if any are within distance 1, Prof. Wheeler will get annoyed and reject the entire set of requests. Suppose that each request for frequency 88.1 FM consists of some identifying information plus (x, y) coordinates of the station location. Assume that no two requests have the same x coordinate, and likewise no two have the same y coordinate. The input includes two sorted lists, L_x of the requests sorted by x coordinate and L_y of the requests sorted by y coordinate.

The problem here is a variant of the closest point problem, which is described in a separate document.

1 Suppose that the map is divided into a square grid, where each square has dimensions $\frac{1}{2} \times \frac{1}{2}$. Why must the FCC reject the set of requests if two requests are in, or on the boundary of, the same square?

Solution: The maximum distance between two requests within a $\frac{1}{2} \times \frac{1}{2}$ square is $\frac{\sqrt{2}}{2}$ which is < 1.

2 Design an $O(n \log n)$ algorithm for the FCC to determine whether the pile of requests contains two that are within Euclidean distance 1 of each other; if so, the algorithm should also return an example pair.

Solution: The tricky part of the algorithm is how to merge the subproblems.

Divide: Through L_x, compute the midpoint to divide the array into two subsets of size $n/2$. Then, divide the elements in L_y accordingly into two subsets.

Conquer: If a subset contains ≤ 2 requests, check whether the two requests collide. Otherwise, divide the subset further.

Merge: Let S be the intersection region between two subsets, with width 2 centered on the vertical boundary dividing the two regions. Sort the elements in the two subsets into S by y values, and for each element of S, check if the request intersects with any of the succeeding 7 requests in S. If they do, return that pair.

Correctness: For any subsets containing ≤ 2 requests, our algorithm checks whether the two requests collide. If there's no collision, then a request can intersect with 7 succeeding requests. To prove that, first, consider that if we divide S into squares of size $1/2$. There's only one request in each square, otherwise our divide step would have returned a collision.

For every request $r \in L$, another request within distance 1 is in the 2×1 rectangle spanning the boundary. There are a total of 8 requests, and excluding r, there are 7.

3 Describe how to modify your solution to determine whether there are three requests, all within distance 1 of each other. For full credit, your algorithm should run in $O(n \log n)$ time, where n is the number of requests.

There are a few modifications in the divide and merge steps. During divide, if there are $C = 3$ requests, we check if the requests are within a distance 1 of each other with three comparisons. During the merge step, for each request r, track the requests that r intersects with, and iterate over every pair in the requests to find two that could intersect. Since there are a total of ≤ 7 compatible requests, there are ≤ 21 combinations to look through, which is still a constant. Therefore, the algorithm is still in $O(n \log n)$ time.