|  |  |
| --- | --- |
|  | **[Design & Analysis of Algorithm]**  **[BSCS – 5 A]**  **Department of Computer Science**  **Bahria University, Lahore Campus** |

**Assignment: 3**

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Roll No: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| **Evaluation of CLO** | **Question Number** | **Marks** | **Obtained Marks** |
| **CLO statement**   * **CLO: Demonstrate an understanding of algorithm design process and different problem solving techniques** | 1 | 5 |  |
| **Total Marks** | | **5** |  |

**Problem Title: Cafe Management [5 points]**

**Sweet Donuts, a new Coffee-and-donuts Café chain, wants to build cafes on many street corners of Lahore with the goal of maximizing their total profit. The street network is described as an undirected graph *G = (V, E)*, where the potential cafe sites are the vertices of the graph. Each vertex u has a nonnegative integer value *Pu*, which describes the potential profit of site u. Two cafes cannot be built on adjacent vertices (to avoid self-competition). You are supposed to design an algorithm that outputs the chosen set *U ⊆ V* of sites that maximizes the total profit ⅀u∈U Pu.**

**First, for parts (a)–(c), suppose that the street network G is acyclic, i.e., a tree.**

1. [1 points] Consider the following “greedy” cafe-placement algorithm: Choose the highest-profit vertex u0 in the tree (breaking ties according to some order on vertex names) and put it into U. Remove u0 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 cafe placement with the maximum profit.

Solution: E.g., the tree could be a line (9, 10, 9).

1. [2 points] Give an efficient algorithm to determine a placement with maximum profit.

Solution: [Algorithm] We can use dynamic programming to solve the problem in O(n) time.

1. First pick any node u0 as the root of the tree and sort all the nodes following a depth-first search (DFS).
2. Store the sorted nodes in array N. Due to the definition of DFS, a parent node appears earlier than all of its children in N.
3. For each node v, define A(v), the best cost of a placement in the subtree rooted at v if v is included, and B(v), the best cost of a placement in the subtree rooted at v if v is not included.
4. The following recursion equations can be developed for A() and B(): If v is a leaf, then A(v) = pv and B(v) = 0. If v is not a leaf, then A(v) = pv + B(u) u∈v.children B(v) = max(A(u), B(u)) u∈v.children For each node v in N, in the reverse order, compute A(v) and B(v).
5. Finally the max(A(u0), B(u0)) is the maximum profit. The placement achieving this maximum profit can be derived by recursively comparing A() and B() starting from the root. The root u0 should be included if A(u0) > B(u0) and excluded otherwise.
6. If u0 is excluded, we go to all of u0’s children and repeat the step; if u0 is included, we go to all of u0’s grandchildren and repeat the step. This algorithm outputs an optimal placement by making one pass of the tree.

**[Correctness]** In the base case where v is a leaf node, the algorithm outputs the optimal placement which is to include the node. In an optimal placement, a node v is either included, which removes all its children, or not, which adds no constraints. By induction, if all the children of v have correct A() and B() values, then A(v) and B(v) will also be correct and the maximum profit at v is derived. Since the array N is sorted using DFS and processed in the reverse order, child nodes are guaranteed to be processed before their parents.

**[Timing Analysis]** Sorting all nodes using DFS takes O(n) time. The time to compute A(v) and B(v), given the values for the children, is proportional to degree(v). So the total time is the sum of all the degrees of all the nodes, which is O(|E|) where |E| is the total number of edges. For a tree structure, |E| = n − 1 so the time for finding all A() and B() is O(n). Finally, using the derived A() and B() to find the optimal placement visits each node once and thus is O(n). Overall, the algorithm has O(n) complexity.

1. [1 points] Suppose that, in the absence of good market research, owner decides that all sites are equally good, so the goal is simply to design a cafe placement with the largest number of locations. Give a simple greedy algorithm for this case, and prove its correctness.

[Solution Algorithm] Similar to Part (b), pick any node u0 as the root of the tree and sort all the nodes following a depth-first search (DFS) and store the sorted nodes in array N. For each valid node in N, in the reverse order, include it and remove its parent from N.

[Time Analysis] Sorting nodes using DFS takes O(n) time. The greedy algorithm also takes O(n) time since it processes each node once. Overall the algorithm has O(n) complexity.

1. [1 points] Now suppose that the graph is arbitrary, not necessarily acyclic. Give the fastest correct algorithm you can for solving the problem.

Solution: A simple algorithm is to try all possible subsets of vertices for U (2V subsets in total), test whether each has the required independence property (only one node should be included for each edge, |E| edges in total), compute the total profit for each valid solution (which takes O(V )), and take the best. This algorithm runs in O(2V |E|) time. This is the intended solution. In fact, this problem is exactly Maximum Independent Set problem, which is known to be NP-complete. So unless P = NP, it has no polynomial-time algorithm. (In fact, assuming something stronger called the Exponential Time Hypothesis, there is no 2o(V ) -time algorithm.