Question 1  [12 marks] Greedy

Suppose there are \(n\) houses built along an east-west road. The position of each house is specified by its distance from the western endpoint; they are given in an array \(P[1...n]\), where \(P[i] = x\) denotes the distance of house \(i\) from the western endpoint. A number of communication towers are to be built. Each tower can cover the houses placed in distance of at most \(R\) from it. The goal is to place towers on the road so that every house is covered by at least one tower.

Give a greedy algorithm to find the minimum number of towers required. Provide

- Pseudocode for the algorithm,
- a proof that it is correct (i.e., produces the minimum possible number of towers), and
- give the asymptotic runtime.

Question 2  [8 marks] Greedy vs. Dynamic Programming

In the topic of greedy algorithms, we solved a problem named Scheduling to minimize lateness. Suppose we want to solve this problem using dynamic programming.

Give a definition of the subproblems, provide the base cases, derive and justify a recursive relation, analyze the runtime of your algorithm (if implemented - you do not need to give the implementation) and state a conclusion on the feasibility of using dynamic programming to solve this problem.

Question 3  [12 marks] Graphs

The input is an undirected unweighted graph \(G = (V, E)\) and two nodes \(w\) and \(v\) in \(G\). The desired output is a single number: the number of shortest \(v-w\) paths in \(G\). (Here, “shortest” means the minimal number of edges.) There could be multiple shortest paths; we are looking for the number of shortest paths, not the list of all such paths.

Provide an \(O(n + m)\) algorithm to solve this problem.

Give pseudocode for your algorithm, justify its correctness, and analyze its runtime (i.e., explain why it has the desired complexity).
Question 4 [12 marks] Graphs

The input is an acyclic directed unweighted graph \( G = (V, E) \) containing \( m = |E| \) edges and \( n = |V| \) nodes. The edges in the graph define a dependency relationship between the nodes. We define that a node \( v \) depends on node \( u \neq v \) if there is a directed path from \( u \) to \( v \); that is, if either edge \((u, v)\) exists in the graph, or there is another node \( z \) so that \((z, v)\) is an edge in the graph and \( z \) depends on \( u \). A node \( u \) is an ultimate node if every other node \( v \) in the graph, either \( u \) depends on \( v \) or \( v \) depends on \( u \).

Give an algorithm that outputs a list of all ultimate nodes in the graph. Give pseudocode, provide a correctness argument, and analyze the runtime of your algorithm. A few examples are shown in the following figure: