Tutorial 4, Design and Analysis of Algorithms, 2021

1. Given an $m \times n$ matrix T of real numbers, prove that (S, I) is a matroid, where S is the set of columns of T and $A \in I$ if and only if the columns in A are linearly independent.

2. Let S be a finite set and let $S_1, S_2, \ldots, S_k$ be a partition of S into nonempty disjoint subsets. Define the structure (S, I) by the condition that $I = \{ A \mid |A \cap S_i| \leq 1, \forall i \in [1..k] \}$. Prove that (S, I) is a matroid. That is, the set of all sets A that contain at most one member of each subset in the partition determines the independent sets of a matroid.

3. Prove that if (S, I) is a matroid, then (S, I') is a matroid, where

$$I' = \{ A' \mid S - A' \text{ contains some maximal } A \in I \}.$$

   That is, the maximal independent sets of (S, I') are just the complements of the maximal independent sets of (S, I).

4. Professor Borden proposes a new divide-and-conquer algorithm for computing minimum spanning trees, which goes as follows. Given a graph $G = (V, E)$, partition the set V of vertices into two sets V_1 and V_2 such that $|V_1|$ and $|V_2|$ differ by at most 1. Let E_1 be the set of edges that are incident only on vertices in V_1, and let E_2 be the set of edges that are incident only on vertices in V_2 . Recursively solve a minimum-spanning-tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Finally, select the minimum-weight edge in E that crosses the cut (V_1, V_2), and use this edge to unite the resulting two minimum spanning trees into a single spanning tree. Either argue that the algorithm correctly computes a minimum spanning tree of G, or provide an example for which the algorithm fails.

5. Show how to transform the weight function of a weighted matroid problem, where the desired optimal solution is a *minimum-weight* maximal independent subset, to make it a standard weighted-matroid problem. Argue carefully that your transformation is correct.

6. (a) Solve the following instance of the 0/1 *Knapsack Problem* using the *Dynamic Programming* algorithm:
   $(I_i)_{i=1}^5 = (I_1, I_2, I_3, I_4, I_5)$,
   $(w_i)_{i=1}^5 = (1, 2, 3, 4, 5)$,
   $(p_i)_{i=1}^5 = (5, 4, 3, 2, 1)$,
   $W = 8$.

   (b) Suppose that in a 0/1 *Knapsack Problem*, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value (for example, as given in problem 6(a)). Give an efficient algorithm (having time complexity $O(n \log n)$) to find an optimal solution to this variant of the knapsack problem, and argue that your algorithm is correct.

7. There is a tower of floors, and an egg dropper with ideal eggs. The physical properties of the ideal egg is such that it will shatter if it is dropped from floor n^* or above, and will have no damage whatsoever if it is dropped from floor $n^* - 1$ or below. The problem is to find a strategy such that the egg dropper can determine the floor n^* in as few egg drops as possible. Design an efficient *Dynamic Programming* algorithm for solving this problem, and find its complexity. Show the working of your algorithm for $n = 10$ floors, and $m = 2$ eggs.

8. Let $G = (V, E)$ be an undirected graph with n nodes. Recall that a subset of the nodes is called an *independent set* if no two of them are joined by an edge. Finding large independent sets is difficult in general; but here we will see that it can be done efficiently if the graph is "simple" enough. Call a graph $G = (V, E)$ a *path* if its

nodes can be written as $v_1, v_2, \ldots, v_n$, with an edge between v_i and v_j if and only if the numbers i and j differ by exactly 1. With each node v_i, we associate a positive integer weight w_i. Consider, for example, the five-node path drawn in Figure 1. The weights are the numbers drawn inside the nodes. The goal in this question is to solve the following problem:
Find an independent set in a path G whose total weight is as large as possible.
Design an efficient *Dynamic Programming* algorithm that takes an n-node path G with weights and returns an independent set of maximum total weight. The running time of your algorithm should be polynomial in n, independent of the values of the weights. Give a formal correctness proof for your algorithm. Find the time complexity of your algorithm and show its working on the given example (Figure 1).

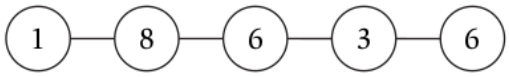

Figure 1: A path with weights on the nodes. The maximum weight of an independent set is 14.

9. Suppose you are managing the construction of billboards on a Highway that runs for M miles. The possible sites for billboards are given by numbers $\{x_1, x_2, \ldots, x_n\}$, each in the interval $[0, M]$. If you place a billboard at location x_i, you receive a revenue of $r_i > 0$. Regulations imposed by the Highway Department require that no two of the billboards be within less than or equal to 5 miles of each other. You have to place billboards at a subset of the sites so as to maximize your total revenue, subject to this restriction.
Example: Suppose $M = 20, n = 4$,
$\{x_1, x_2, x_3, x_4\} = \{6, 7, 12, 14\}$, and
$\{r_1, r_2, r_3, r_4\} = \{5, 6, 5, 1\}$.
Then the optimal solution would be to place billboards at x_1 and x_3, for a total revenue of 10. Design an efficient *Dynamic Programming* algorithm that takes an instance of this problem as input and returns the maximum total revenue that can be obtained from any valid subset of sites. The running time of the algorithm should be polynomial in n. Show the working of your algorithm on the given example.

10. Let $G = (V, E)$ be a directed graph with nodes $v_1, \ldots, v_n$. We say that G is an ordered graph if it has the following properties:
(1) Each edge goes from a node with a lower index to a node with a higher index. That is, every directed edge has the form (v_i, v_j) with $i < j$.
(2) Each node except v_n has at least one edge leaving it. That is, for every node v_i, $i = 1, 2, \ldots, n-1$, there is at least one edge of the form (v_i, v_j).
Design an efficient *Dynamic Programming* algorithm that takes an ordered graph G and returns the length of the longest path that begins at v_1 and ends at v_n (the length of a path is the number of edges in the path). Find the time complexity of your algorithm. Show the working of your algorithm on the example graph in Figure 2.

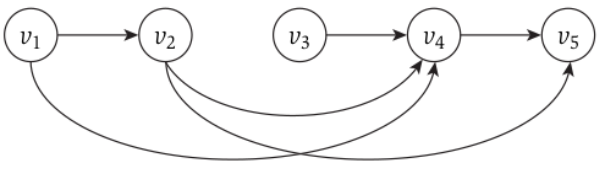

Figure 2: Example graph for problem 10.