7. tutorial

Greedy and Knapsack. The KNAPSACK problem is the following: we are given a collection of n items of sizes $a_1, \ldots, a_n \in \mathbb{N}$ and a capacity $K \in \mathbb{N}$, and the task is to find the largest subset of items which fits in the knapsack, that is, whose total size is at most K.
The algorithms for MST which we have described are all an example of *greedy algorithms*, that is, at each iteration, they make a choice which is best *in the moment*, without any regard for the future (will this block off the path to some better solution?). This greedy approach is a heuristic which can be applied to many problems (at each iteration, make a step which gives the most improvement *in this moment*), and in some cases, it can be proven that it approximates the optimum well. However, MST is one of the rare problems where the greedy approach optimally solves the problem.
Show that a greedy approach will not work for KNAPSACK. (This is not a well-defined question: there are several natural ways to define "a greedy approach to KNAPSACK". Explore the ones which come to you.)

Still connected. Design a linear-time algorithm for the following task. You are given an undirected graph $G = (V, E)$. Decide whether there is an edge $e \in E$ such that $G - e$ is still connected. Can you improve the algorithm to run in time $\mathcal{O}(n)$?

MST properties. The following statements may or may not be correct. In each case, either prove it (if it is correct) or give a counterexample (if it isn't correct). Always assume that the graph $G = (V, E)$ is undirected. Do not assume that edge weights are distinct unless this is specifically stated.

(1) If graph G has more than $|V|-1$ edges, and there is a unique heaviest edge, then this edge cannot be part of a minimum spanning tree.
(2) If G has a cycle with a unique heaviest edge e, then e cannot be part of any MST.
(3) Let e be any edge of minimum weight in G. Then e must be part of some MST.
(4) If the lightest edge in a graph is unique, then it must be part of every MST.
(5) If e is part of some MST of G, then it must be a lightest edge across some cut of G.
(6) If G has a cycle with a unique lightest edge e, then e must be part of every MST.
(7) Prim's algorithm works correctly when there are negative edges.

Array → BVS. Design an algorithm which takes on input a sorted array of numbers, and creates a perfectly balanced BST in linear time.

BST_SPLIT. Design a `BST_SPLIT` operation which takes on input a BST T and a value s and outputs two BSTs T_1, T_2 such that T_1 contains those values of T which are smaller than s, and T_2 contains those values of T which are $\geq s$.

BST improvements. Consider a general BST maintaing (key, value) pairs, sorted by the key. Implement the following operations while maintain the $\mathcal{O}(\text{dept})$ complexity.

(1) Min, Max, Average of a certain interval of keys. (E.g. $\max(r, s)$ should return the largest value of keys between r and s.)
(2) Assume that the values are matrices $A_1, \ldots, A_m$ of size $n \times n$ and for any interval of keys r, s we want to be able to quickly compute what is the matrix product $A_r \cdot A_{r+1} \cdots A_s$. (Caution: this operation is not commutative. Unlike for max above, the order of matrix multiplications matters.)
(3) Adding δ to all values in a given interval.