(a) Since we want to have as few jump gates as possible and the context implies that jumps should always be possible in both directions, a plausible constraint would be to say that the input graph is undirected and acylic, therefore a tree.

(b) Basic idea: DFS, if no children, do nothing, otherwise if min. 1 without station place stuff

- Input: A tree $T = (V, E)$ as an adjacency list with n nodes and $n - 1$ edges.
- Output: An array A of length n that stores a boolean for any system $1 \leq i \leq n$ at $A[i]$ stating whether or not a station is build in system i.
- Correctness criteria: Let S be our solution and $p(A)$ be defined as the total count of all stations for a solution A. For any other solution S' it must hold: $p(S) \leq p(S')$ for S to be correct and optimal.
- Algorithm: We are using a recursive DFS and extend its functionality with a simple check. First we pick a random node as a root and start the recursion.
  When we inspect a node that has no children we don't do anything, since this must mean we've reached a leaf vertex. The only exception we make here is if $n = 1$, since we would trivially have to put a station in the only existing system.
  If the current vertex has children we check if any of the children misses a maintenance station. If so, we add a station the current system, therefore add *true* in our boolean Array A.
  In the end we return A.

- Algorithm correctness

  Beweis. When our algorithm traverses a leaf we don't place a station there since we could only cover at most one other system. If we place a station at the parent instead, we also cover any other potential children, as well as any parents. Therefore we are potentially making a worse decision if placing a station at a leaf node.
  Now, given any node i that is not a leaf. Assuming that the children sub-trees rooted at i are chosen optimally. If any of these children do not have a station we must place one in system i, which is exactly what our algorithm does. Otherwise we don't.
  Since we make an optimal decision for our children nodes and our recursion places only necessary stations our solution must be correct and especially optimal. If we were to remove any station the correctness criteria wouldn't be met.

  □

- Runtime:

  Beweis. We know that DFS has a runtime of $O(n)$ in a tree. Since we check all children of every vertex for a station exactly once, this adds an additional $O(n)$. Therefore our total runtime is $O(n)$.