

## Task A12. Bits and Tree

🕒 15.00 s 📁 2048 MB

Author: Viktor Kozhuharov

You are given a binary string  $S$  of length  $2N$ . Your task is to encode the **longest prefix** you can of  $S$  using an unlabeled undirected tree with  $N$  nodes.

To do this, you need to write two functions: an encoder and a decoder. Your encoder will get the binary string and construct a tree with nodes labeled from  $0$  to  $N-1$ . However, the communication medium is unreliable and will **corrupt the tree** by adding one extra node and connecting it to one of the original  $N$  nodes. The decoder will receive only this corrupted version of the tree, which now has  $N+1$  nodes and  $N$  edges. Additionally, the labels and edges (and the nodes in each edge) will be randomly permuted before being passed to the decoder. This simulates passing an **unlabeled** tree to the decoder. The decoder must rely only on the structure of the tree and cannot assume anything about the labels.

Your decoder must return a string that shares as long a common prefix as you can with the original string  $S$  **no matter where the corruption occurred**. Your score will depend on the minimum length of a common prefix among *all possible corruption scenarios* and among all subtests of all tests.

### Implementation details

You need to implement the following functions:

```
std::vector<std::pair<int, int>> encode(int n, std::vector<bool> data)
```

This function receives: an integer  $N$  (the number of nodes you are allowed to use) and a boolean vector `data`, representing the binary string  $S$ . It should return a list of  $N-1$  edges forming a tree of  $N$  nodes (0-indexed).

```
std::vector<bool> decode(int n, std::vector<std::pair<int, int>> tree)
```

This function receives: an integer  $N$  (which is the original value passed to `encode`) and a vector of  $N$  edges forming a tree with  $N+1$  nodes (corrupted version). It should return a sequence of bits that should match the original `data` to as long as prefix as you can.

For a given test, the grader will run 1 instance of the encoder and  $N$  different ones of the decoder. Each of these instances will have its respective function called  $T$  times - once per subtest of the test.

Your program should not implement a `main` function and it should not interact with the standard input and output. It will be compiled together with a system grader that will handle these parts.

### Constraints

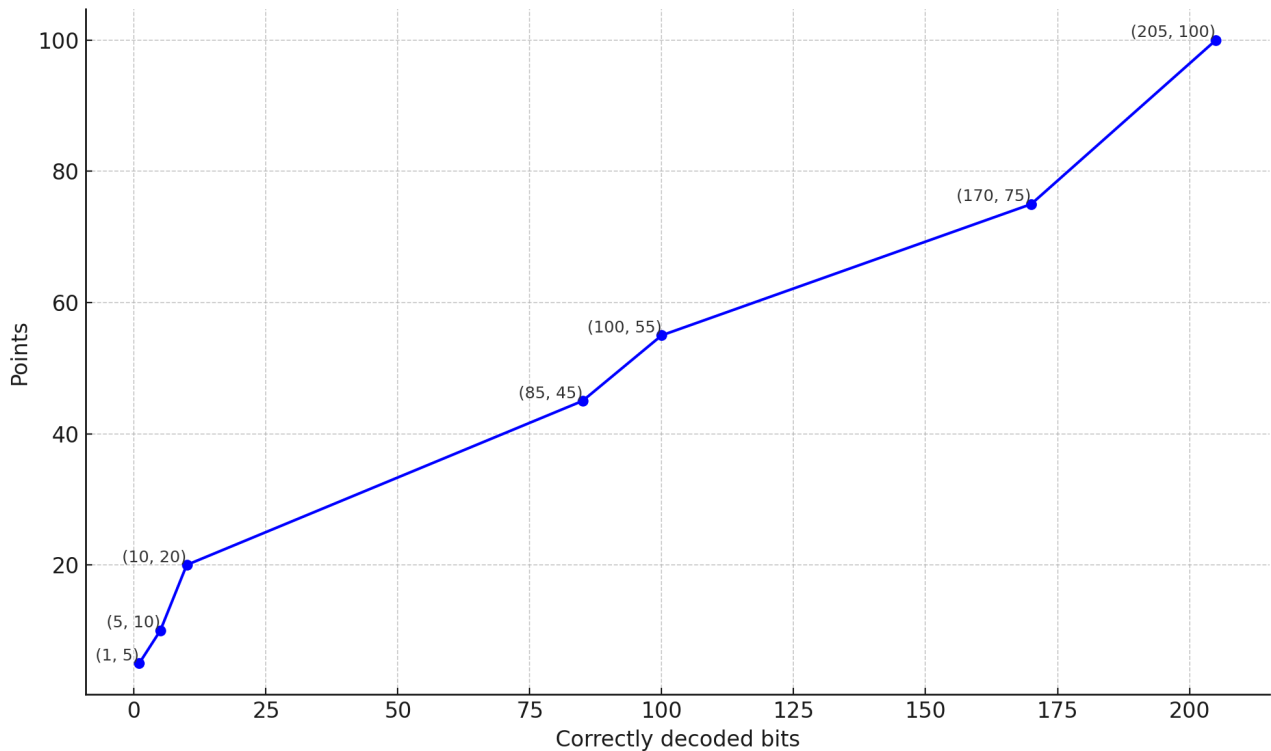
- $T = 2$
- $N = 200$
- $|\text{data}| = 2N$
- Node labels range from  $0$  to  $N-1$  in the uncorrupted tree.
- Node labels range from  $0$  to  $N$  in the corrupted tree.

### Scoring

Your final score for the task (as a fraction of the full points) is based on the number of correctly decoded bits produced by your solution.

If your solution produces an invalid tree, your score will be 0.

Otherwise, your score  $S$  is computed using the following piecewise linear function:



The number of correctly decoded bits is the minimum among all tests, subtests, and corruption scenarios.

### Local Testing

A local grader is provided to help you test your solution. It needs to be compiled together with your program. It does not use separate processes.

It simulates the full encoding/decoding pipeline:

1. Reads an integer  $T$  – the number of subtests.
2. For each test case:
  - (a) Reads an integer  $N$  – the number of nodes available for encoding.
  - (b) Reads an integer  $\ell$  – the length of the binary data string (usually  $2N$ ).
  - (c) Reads  $\ell$  bits (each 0 or 1 without spaces) representing the binary string  $S$ .
  - (d) Calls your `encode(n, data)` function.
  - (e) Tries corrupting the resulting tree by:
    - Iterating through each  $x$  — the index of the node (from 0 to  $N - 1$ ) to attach a new node to.
    - Adds a new node and attaches it to node  $x$ .
    - Randomly permutes the node labels, shuffles the edge lists and swaps at random the 2 ends of each edge.
  - (f) Calls your `decode(n, corrupted_edges)` function with each corrupted tree.
  - (g) Compares each result against the original data and counts how many bits match the original string in the shortest matching prefix.

### Example Interaction

| Input                | Interaction  |
|----------------------|--|
| 1<br>5 10 1101010110 | <pre> encode(5, {1,1,0,1,0,1,0,1,1,0}) returns {{0,1},{1,2},{0,3},{1,4}} decode(5, {{1,0},{0,2},{2,3},{3,4},{5,3}}) returns {1,1,0} decode(5, {{1,0},{0,2},{0,3},{4,0},{5,4}}) returns {1,1,1,0,1,0,1,0} decode(5, {{2,0},{2,1},{1,3},{4,1},{5,4}}) returns {1,1,1,0,0,0,0,0} decode(5, {{0,1},{0,2},{0,3},{4,3},{3,5}}) returns {1,1,1,0,1} decode(5, {{1,0},{1,2},{2,3},{4,2},{5,4}}) returns {1,1,1,0}                     </pre> |