

### Задача A12. Битовая строка и дерево

⌚ 15.00 с 📄 2048 МБ

Вам дана битовая строка  $S$  длины  $2N$ . Ваша задача — закодировать **наибольший возможный префикс** строки  $S$  с помощью неориентированного непомеченного дерева, содержащего  $N$  вершин.

Для этого вам нужно реализовать две функции: кодировщик и декодировщик. Ваш кодировщик получит битовую строку и построит дерево с вершинами, пронумерованными от 0 до  $N - 1$ . Однако канал связи ненадёжен и **искажает дерево** при передаче, добавив одну дополнительную вершину и соединив её с одной из исходных  $N$  вершин. Декодировщик получит только эту искажённую версию дерева, которая теперь содержит  $N + 1$  вершин и  $N$  рёбер. Кроме того, номера вершин и рёбра (и порядок вершин в каждом ребре) будут случайным образом переставлены перед передачей декодировщику. Это используется для симулирования передачи непомеченного дерева декодировщику, декодировщик должен полагаться только на структуру дерева и не может использовать информацию о номерах вершин и ребер.

Ваш декодировщик должен вернуть строку, которая совпадает с исходной строкой  $S$  на префиксе как можно большей длины **независимо от того, где произошло искажение**. Ваш балл будет зависеть от минимальной длины общего префикса среди *всех возможных сценариев искажения* и среди всех тестовых примеров каждого теста.

#### Детали реализации

Вам необходимо реализовать следующие функции:

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

Эта функция получает: целое число  $N$  (количество вершин, которые можно использовать) и вектор `data`, представляющий битовую строку  $S$ . Она должна вернуть список из  $N - 1$  рёбер, образующих дерево из  $N$  вершин (нумерация с 0).

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

Эта функция получает: целое число  $N$  (исходное значение, переданное в `encode`) и вектор из  $N$  рёбер, образующих дерево с  $N + 1$  вершинами (искажённая версия). Она должна вернуть последовательность битов, которая должна совпадать с исходными данными `data` на префиксе как можно большей длины.

Для каждого теста система запустит 1 экземпляр функции кодировщика и  $N$  различных экземпляров декодировщика. У каждого из этих экземпляров функция будет вызвана  $T$  раз — по одному разу на каждый тестовый пример.

Ваша программа не должна реализовывать функцию `main` и не должна взаимодействовать со стандартным вводом и выводом. Она будет скомпилирована вместе с системным грейдером, который обрабатывает эти части.

### Ограничения

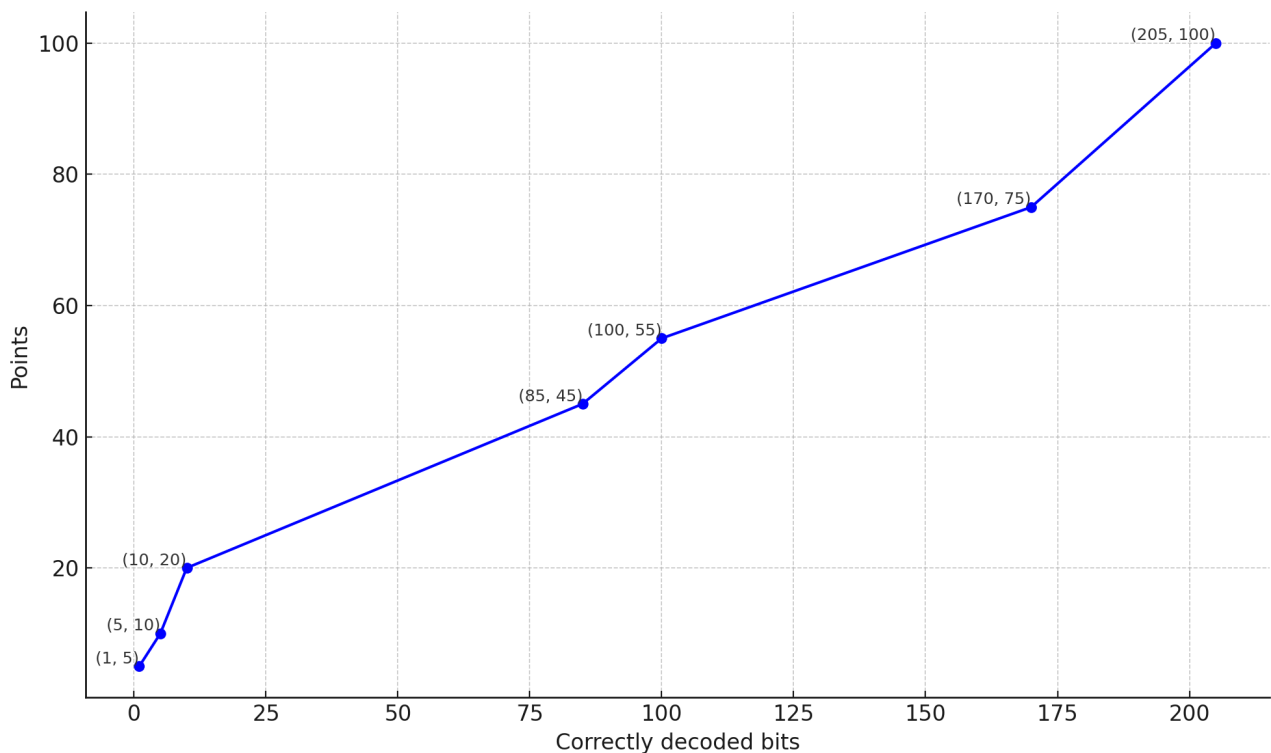
- $T = 2$
- $N = 200$
- $|\text{data}| = 2N$
- Индексы вершин в исходном дереве от 0 до  $N - 1$ .
- Индексы вершин в искажённом дереве от 0 до  $N$ .

### Система оценки

Итоговый балл за задачу (в долях от максимального) зависит от количества правильно декодированных битов, полученных вашим решением.

Если ваше решение возвращает некорректное дерево, вы получите 0 баллов.

В противном случае ваш балл  $S$  вычисляется с помощью следующей кусочно-линейной функции:



Количество правильно декодированных битов — это минимум среди всех тестов, тестовых примеров и сценариев искажения.

### Локальное тестирование

Для удобства тестирования предоставляется локальный грейдер. Он должен быть скомпилирован вместе с вашей программой и не использует отдельные процессы.

Он имитирует полный процесс кодирования/декодирования:

1. Читает целое число  $T$  — количество тестовых примеров.
2. Для каждого тестового примера:
  - (a) Читает целое число  $N$  — количество вершин, доступных для кодирования.
  - (b) Читает целое число  $\ell$  — длину битовой строки (обычно  $2N$ ).
  - (c) Читает  $\ell$  битов (каждый 0 или 1 без пробелов), представляющих строку  $S$ .
  - (d) Вызывает вашу функцию `encode(n, data)`.
  - (e) Искажает полученное дерево:
    - Перебирает каждый  $x$  — индекс вершины (от 0 до  $N-1$ ), к которой будет добавлена новая вершина.
    - Добавляет новую вершину и соединяет её с вершиной  $x$ .
    - Случайным образом переставляет метки вершин, перемешивает список рёбер и меняет местами вершины в каждом ребре.
  - (f) Вызывает вашу функцию `decode(n, corrupted_edges)` для каждого искажённого дерева.
  - (g) Сравнивает каждый результат с исходными данными и подсчитывает минимальное количество битов, совпадающих с исходной строкой в общем префиксе.

### Пример взаимодействия

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