Задача E

1) Алгоритм

Задача сводится к нахождению максимальной XOR-суммы для путей, проходящих через фиксированный узел в графе, который состоит из узлов, соединенных дорогами, каждая из которых имеет вес.

Для решения этой задачи был реализован следующий алгоритм:

- Дерево представлено в виде списка смежности, где каждая вершина связана с другими вершинами и весом дороги, соединяющей их.

- Для нахождения путей и XOR-сумм используется метод обхода в глубину (calculatePaths). Мы сохраняем XOR-сумму путей от узла f до всех остальных узлов.

Построение всех возможных путей - после вычисления всех XOR-сумм из узла f до других узлов, формируется список possiblePaths, содержащий значения XOR для всех узлов, кроме узла f.

Для нахождения максимальной XOR-пары был использован подход с двоичным деревом (BinaryTree), который позволяет эффективно находить максимальное значение XOR для заданного числа. Мы добавляем все значения XOR в двоичное дерево, а затем находим максимальную XOR-пару для каждого значения в списке possiblePaths.

Реализация двоичного дерева

Для нахождения максимальной XOR-пары был использован BinaryTree, который представлен как дерево поиска битов:

- Дерево состоит из узлов BinaryNode, где каждый узел имеет две ветви, соответствующие битам 0 и 1.
- Добавление значения в дерево происходит путем побитового обхода каждого бита числа, начиная с самого старшего.
- Для поиска максимальной XOR-пары значение XOR на каждом уровне определяется исходя из противоположного бита текущего числа, что позволяет максимизировать XOR.

2) Сложность решения

Вычисление всех XOR-сумм путей осуществляется с помощью обхода в глубину, что требует O(N), где N — количество узлов в дереве. Добавление и поиск в двоичном дереве выполняется за O(32) для каждого значения, так как числа представлены в виде 32-битных целых чисел. Следовательно, сложность добавления всех значений и поиска максимальной XOR-пары составляет O(N).

Итоговая сложность: Алгоритм имеет линейную сложность O(N), что делает его эффективным для данной задачи, N может достигать 100000.

Память используется для хранения графа, массива pathXors для хранения XOR-сумм путей, и двоичного дерева. В худшем случае, память требуется для хранения всех узлов графа и их связей, что составляет O(N).

Обоснование корректности - использование обхода в глубину гарантирует, что все возможные пути от узла f до других узлов будут корректно найдены. Двоичное дерево позволяет эффективно находить максимальную XOR-сумму для двух значений, что является ключевым для данной задачи.

3) Распечатка кода

```
#include <array>
#include <iostream>
#include <memory>
#include <vector>

class PathFinder {
private:
  struct BinaryNode {
    std::array<std::unique_ptr<BinaryNode>, 2> branches;
    BinaryNode() : branches{nullptr, nullptr} {
    }
  };

  class BinaryTree {
  private:
    std::unique_ptr<BinaryNode> root;
```

```
  public:
    BinaryTree() : root(std::make_unique<BinaryNode>()) {
    }

    void addPath(int value) {
      BinaryNode* current = root.get();
      for (int pos = 31; pos >= 0; --pos) {
        int bit = (value >> pos) & 1;
        if (!current->branches[bit]) {
          current->branches[bit] =
std::make_unique<BinaryNode>();
        }
        current = current->branches[bit].get();
      }
    }

    int findMaxXorPair(int value) {
      BinaryNode* current = root.get();
      int result = 0;

      for (int pos = 31; pos >= 0; --pos) {
        int bit = (value >> pos) & 1;
        int opposite = 1 - bit;

        if (current->branches[opposite]) {
          result |= (1 << pos);
          current = current->branches[opposite].get();
        } else if (current->branches[bit]) {
          current = current->branches[bit].get();
        } else {
          break;
        }
      }
      return result;
    }
  };

  std::vector<std::vector<std::pair<int, int>>> graph;
```

```
  std::vector<int> pathXors;
  std::vector<bool> visited;
  int vertexCount;
  int shopLocation;

public:
  PathFinder(int n, int shop)
      : graph(n + 1), pathXors(n + 1), visited(n + 1,
false), vertexCount(n), shopLocation(shop) {
  }

  void addRoad(int from, int to, int weight) {
    graph[from].emplace_back(to, weight);
    graph[to].emplace_back(from, weight);
  }

  void calculatePaths(int vertex, int currentXor) {
    visited[vertex] = true;
    pathXors[vertex] = currentXor;

    for (const auto& [nextVertex, weight] : graph[vertex]) {
      if (!visited[nextVertex]) {
        calculatePaths(nextVertex, currentXor ^ weight);
      }
    }
  }

  int findMaxDiscount() {
    calculatePaths(shopLocation, 0);

    std::vector<int> possiblePaths;
    for (int i = 1; i <= vertexCount; ++i) {
      if (i != shopLocation) {
        possiblePaths.push_back(pathXors[i]);
      }
    }

    BinaryTree xorTree;
```

```
    int maxDiscount = 0;

    for (int value : possiblePaths) {
      xorTree.addPath(value);
    }

    for (int value : possiblePaths) {
      maxDiscount = std::max(maxDiscount,
xorTree.findMaxXorPair(value));
    }

    return maxDiscount;
  }
};

int main() {
  int n, f;
  std::cin >> n >> f;

  PathFinder solver(n, f);

  for (int i = 0; i < n - 1; ++i) {
    int u, v, w;
    std::cin >> u >> v >> w;
    solver.addRoad(u, v, w);
  }

  std::cout << solver.findMaxDiscount() << std::endl;
  return 0;
}
```