Тепляков Владислав БПИ2310
SET 6
Задача A1

Часть 1

Алгоритм 1

- **Сортировка ребер**: Необходимо упорядочить все m ребер по их весу. Для выполнения этой задачи требуется время, пропорциональное $O(m\log m)$, если использовать функцию std::sort.
- **Проверка связности**: Для каждого ребра мы проводим m итераций, проверяя, остаётся ли граф T без этого ребра связным, используя DSU. Создание DSU и объединение $m-1$ рёбер занимает время, пропорциональное $O(m \cdot \alpha(n))$, а проверка связности всех n вершин — $O(n \cdot \alpha(n))$. Таким образом, общая сложность проверки составляет $O(m \cdot (m+n) \cdot \alpha(n))$. Что является оптимальным, если не использовать более продвинутые структуры данных, такие как DSU с возможностью удаления ребра.
- **Работа алгоритма**: "В целом, алгоритм функционирует за время $O(m\log m+m \cdot (m+n) \cdot \alpha(n)) \Rightarrow O(m \cdot (m+n) \cdot \alpha(n))$.

Алгоритм 2

- **Рандомизация ребер**: Для выполнения этой операции применяется встроенная функция random_shuffle, которая позволяет достичь времени выполнения $O(n)$.
- **Проверка на наличие циклов**: Чтобы избежать циклов, необходимо проверить, является ли граф, образованный объединением множеств T и {e}, связным. Для этого используется алгоритм DSU, который за время $O(\alpha(n))$ проверяет принадлежность вершин к одному множеству.
- **Работа алгоритма**: В целом, алгоритм функционирует за время, пропорциональное $O(m*\alpha(n))$, что приближенно соответствует линейному времени.

Алгоритм 3

- **Рандомизация ребер**: Для выполнения этой операции применяется встроенная функция random_shuffle, которая позволяет достичь времени выполнения O(n).
- **Проверка на наличие цикла**: Чтобы проверить наличие цикла при добавлении ребра, следует использовать DSU — это позволит выполнить проверку за время, которое можно описать как $O(\alpha(n))$.
- **Поиск цикла и максимального ребра**: Если цикл был обнаружен, необходимо найти его и ребро с максимальным весом. DSU позволяет легко найти сам цикл, но для поиска максимального ребра необходимо хранить веса рёбер и проверять их. Это приведет к худшему случаю временной сложности O(m) для поиска цикла и O(1) для выбора максимального веса, если веса уже хранятся в структуре.
- **Работа алгоритма**: В целом, алгоритм функционирует за время, пропорциональное O(m*m).

Часть 2

Алгоритм 1

- Сортировка по убыванию весов и удаление рёбер, сохраняя связность, означает, что ALG_1 удаляет рёбра с максимальными весами, которые не являются необходимыми для связности. Это эквивалентно выбору минимальных рёбер для сохранения связности, что соответствует построению минимального остовного дерева (МОД).
  Ответ: **Да, всегда формируется**

Алгоритм 2

- Возьмем граф с вершинами A, B и C и рёбрами A-B (вес 1), B-C (вес 100) и A-C (вес 2). Минимальное остовное дерево (МОД) — это ребро A-C, вес которого равен 3. Если алгоритм ALG_2 сначала выберет ребро B-C, то в связном графе не возникнет цикл, и набор T={A-B, B-C} с весом 101 не будет минимальным.
  Ответ: **Нет, формируется не всегда**

Алгоритм 3

- Этот алгоритм последовательно добавляет ребра в граф до тех пор, пока не образуется цикл. Затем он удаляет ребро с наибольшим весом из этого цикла. Это можно рассматривать как выбор минимального ребра на каждом шаге, подобно алгоритму Краскала. После удаления максимального ребра в цикле остаются только те, что имеют меньший вес. Если граф связный и все рёбра имеют уникальные веса, или если алгоритм корректно обрабатывает равные веса, то случайный порядок добавления ребер не повлияет на конечный результат.
- Ответ: **Да, всегда формируется**