SET 6. A1

Algorithms [SET 6](SET 6)

Задание:

`1. Для каждого из трех представленных алгоритмов обоснуйте его наиболее эффективную по временной сложности реализацию, в особенности, с точки зрения используемых структур данных и операций на них. Обоснуйте оценки сложности. Представьте исходный код на языке C++ для каждой из соответствующих реализаций, в которых используемые структуры данных достаточно отразить на уровне интерфейса – приводить полный код используемых структур данных не нужно.`
`2. Для каждого из трех представленных алгоритмов определите, формируется ли в множестве ребер T минимальное остовное дерево исходного графа G. Обоснуйте свой ответ и приведите (контр)примеры.`

Algorithm 1

Используем список рёбер с их длиной. Тогда их сортировка составит `O(ElogE)`.

Дальше проходимся по всем рёбрам и проверяем, что множество `T – {e}` образует связанный граф. Проверку будем производить через обход в глубину DFS, которая составит `O(V + E')`, где `E'` - количество рёбер на итерации. В худшем случае, при использовании всех рёбер на каждой итерации будет `O(V + E)`. Удаление в худшем случае выполнится за `O(E)`. Поэтому весь цикл займёт `O(E ∗ (V + 2 ∗ E))`.

В итоге алгоритм будет работать за `O(ElogE + E ∗ (V + 2 ∗ E)) = O(E ∗ (logE + V + 2 ∗ E)) = O(E ∗ (V + E))`.

```
#include <vector>
#include <algorithm>

struct Edge {
    int u;
    int v;
    int weight;

    bool operator<(const Edge& other) const {
        return weight > other.weight;
    }
};

bool isRelated(std::vector<Edge>& T, int V);

std::vector<Edge> ALG_1(std::vector<Edge> E, int V) {
    std::sort(E.begin(), E.end());
```

```
    std::vector<Edge> T = E;
    for (Edge& edge : T) {
        auto it = std::find(T.begin(), T.end(), edge);
        T.erase(it);
        if (!isRelated(T, V)) {
            T.insert(it, edge);
        }
    }
    return T;
}
```

Этот алгоритм формирует минимальное остовное дерево исходного графа.

Покажем это:

1. Алгоритм формирует остовное дерево, так как после его применения граф остаётся связанным и без циклом, потому что при нахождении цикла алгоритм бы нашёл в нём максимальное ребро и удалил.
2. Покажем, что алгоритм формирует минимальное остовное дерево, по индукции. Предположение: множество рёбер `S` содержит рёбра `T`, входящие в MST.
    1. В самом начале множество содержит все вершины, поэтому в нём точно есть рёбра из MST.
    2. Пусть условие выполнено для множества `S` - в нём есть рёбра `T`, образующие остовное дерево.
    3. Удалим ребро `e` из `S`, тогда в `S` должно остаться какое-то минимальное остовное дерево `T'`.
    4. Если `e` не принадлежало `T`, тогда `T' = T` и всё корректно.
    5. Если `e` принадлежало `T`, тогда остовное дерево `T` перестанет быть связанным, а `S` останется связанным. И получается, что остовное дерево `T` делится на 2 подграфа `T1` и `T2`, которые должны иметь общее ребро отличное от удалённого. Пусть этим ребром будет `e'`. Тогда `T' = T - e + e'` и граф `S` имеет цикл из подгафов `T1` и `T2` через рёбра `e` и `e'`.
    6. Так как `T` было остовным деревом, то `T'` также им будет, потому что мы просто вместо одного ребра взяли другое меньшего размера.
    7. Покажем, что `T'` - MST:
        1. Если `e'` < `e`, то `T > T'`, но по условию `T` было минимальным остовным деревом, поэтому меньше другого дерева оно быть не могло => противоречие.
        2. Если `e' > e`, то есть мы удалили меньшее ребро, но по алгоритму нам сначала должно было встретиться ребро `e'`, которое можно удалить, так как они образуют цикл. Противоречие.
        3. Если `e' = e`, тогда `T' = T` - минимальное остовное дерево.

    Следовательно, алгоритм находит MST.

Algorithm 2

Используем список рёбер с их длиной.

Цикл по рёбрам - `O(E)`. В произвольном порядке берём рёбра и добавляем их в множество T за амортизированную `O(1).` После этого проверяем, на наличие циклов через DFS за `O(V + E)`. В случае нахождения цикла, удаляем ребро за `O(1)`.

В итоге алгоритм будет работать за `O(E * (V + E))`.

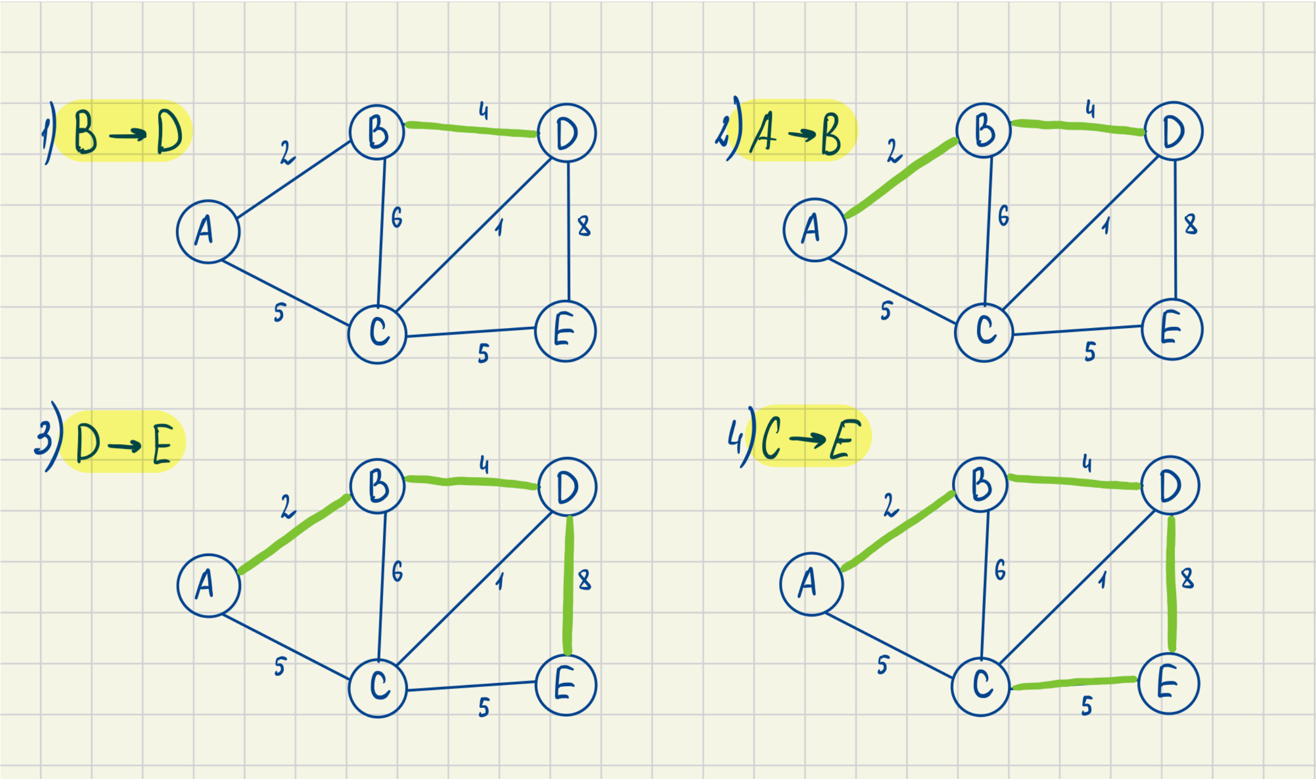
```
#include <vector>

struct Edge {
    int u;
    int v;
    int weight;
};

std::vector<Edge> ALG_2(std::vector<Edge> E, int V) {
    std::vector<Edge> T;
    UnionFind union_find(T);
    for (Edge& edge : E) {
        T.push_back(edge);
        if (!union_find.isCycle()) {
            T.pop_back();
        }
    }
    return T;
}
```

Алгоритм не строит минимальное остовное дерево. Контрпример:

По данному алгоритму выберем рёбра B-D, A-B, D-E, C-E.
Они образовали остовное дерево длиной 19, но есть другое дерево длиной 12 (A-B-D-C-E).



Algorithm 3

Используем список рёбер с их длиной.

Цикл по рёбрам - `O(E)`. В произвольном порядке берём рёбра и добавляем их в множество T за амортизированную `O(1).` После этого проверяем, на наличие циклов через DFS за `O(V + E)`. В случае нахождения цикла, ищем в этом цикле ребро с максимальным весом (за `O(E)`) и удаляем его за `O(1)`.

В итоге алгоритм будет работать за `O(E * (V + E + E)) = O(E * (V + E))`.

```
#include <vector>

struct Edge {
    int u;
    int v;
    int weight;
};

bool isCycle( std::vector<Edge> T);

Edge findMaxEdgeInCycle(std::vector<Edge> T);

std::vector<Edge> ALG_2(std::vector<Edge> E, int V) {
    std::vector<Edge> T;
    for (Edge& edge : E) {
        T.push_back(edge);
        if (isCycle(T)) {
            Edge e_max = findMaxEdgeInCycle(T);
            auto it = std::find(T.begin(), T.end(), e_max);
```

```
            T.erase(it);
        }
    }
    return T;
}
```

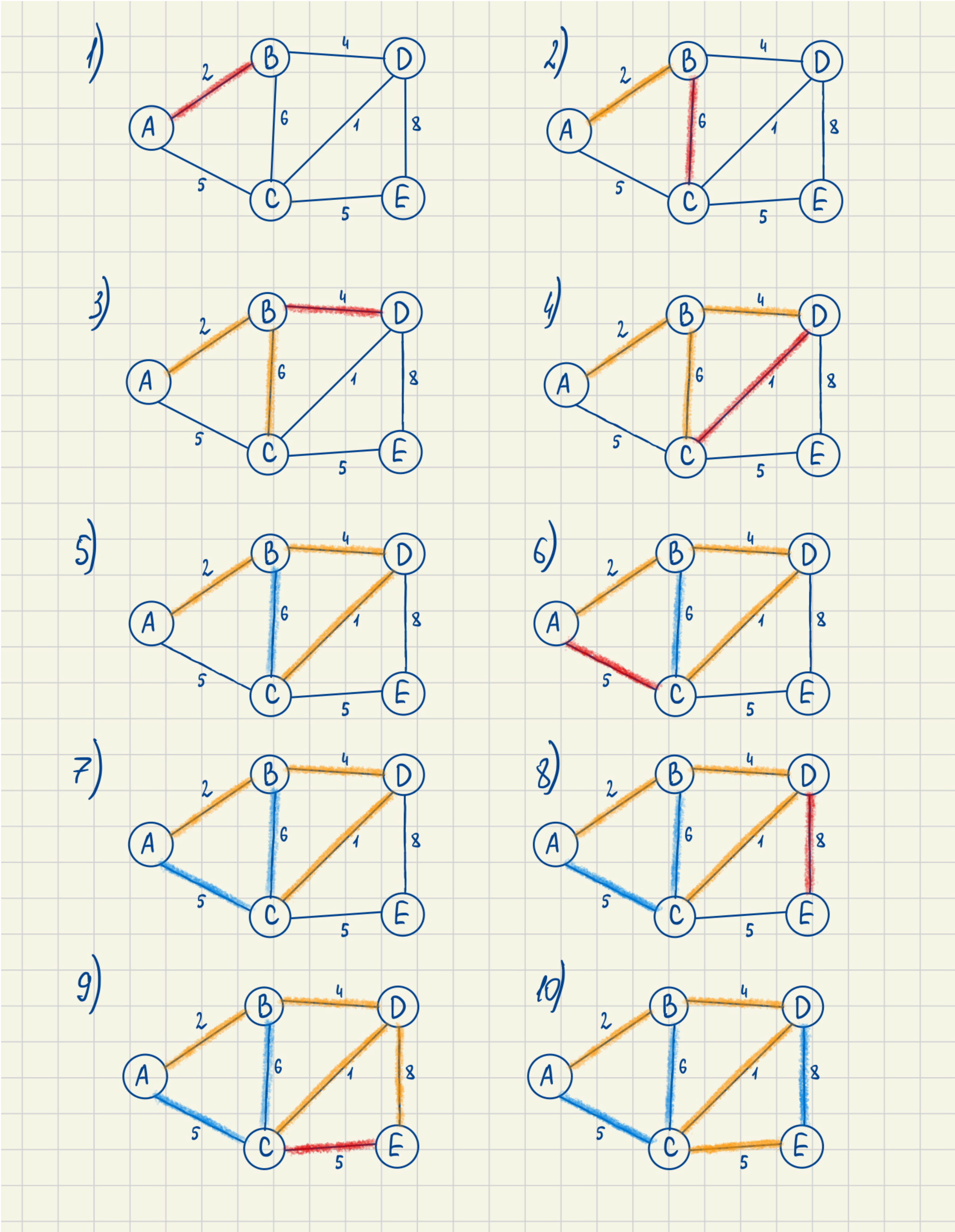
Данный алгоритм образует MST.

Алгоритм берёт рёбра в произвольном порядке. Затем при нахождении цикла ребро с максимальным весом будет удалён, то есть локально в данной компоненте, где образовался цикл, будет взят оптимальный вариант. И так как все такие циклы будут оптимизированы, то получится минимальное остовное дерево.

Пример работы алгоритма:

Уточнение:
Красное - ребро, которое добавляется.
Жёлтое - ребро, которое было добавлено в T на предыдущих шагах.
Синее - ребро, которое удалили.

4 - Образовался цикл `BCD`, поэтому берём ребро `CD` и удаляем `BC`.
6 - Образовался цикл `ABCD`, поэтому удаляем `AC`.
9 - Образовался цикл `CDE`, поэтому удаляем `DE` и берём `CE`.



Таким образом нашли минимальное остовное дерево `A–B–D–C–E` длиной 12.