Задание A1

Алгоритм ALG_1

1. В данном задаче будем хранить графа в виде списка рёбер. Позже, при анализе, обоснуем этот выбор
2. Для сортировки рёбер по весу необходимо $\Theta(|E| \log |E|)$ операций
(в частном случае, если веса - ограниченное подмножество целых чисел фиксированного размера (в битах), можно применить radix-sort за $\Theta(|E| + K)$)
3. Внешний цикл будет выполняться за $\Theta(|E|)$, проверка на связность - обходом в глубину / ширину (dfs / bfs) (корректно, т.к. граф неориентированный)
На каждой итерации обход делается за $\Theta(|V| + |E'|)$, где E' - текущее количество рёбер в графе
На каждой итерации в худшем случае $|E'| = |E|$, если ни одно ребро на предыдущих итерациях не удалялось
Удаление ребра - за $O(|E|)$, добавление ребра обратно - за $O(1)$
Таким образом, цикл работает за $O(|E|\,(|E| + (|E| + |V|))) = O(|E|\,(|E| + |V|))$
3. В данной задаче не важно - хранить граф в виде списка рёбер или списка смежности, т.к. на каждой итерации самая дорогая операция - обход за $O(|V| + |E|)$.
Если хранить граф в виде матрицы смежности/сопряжённости, то операции удаления станут работать за $O(1)$, но обход станет работать за $O(|V|^2)$, что не лучше изначальной асимптотики.
4. Тогда весь алгоритм работает за $O\left(|E| \cdot (|E| + |V|) + |E| \log |E|\right) = O\left(|E| \cdot (|E| + |V|)\right)$

Алгоритм ALG_2

1. В данном задаче будем хранить графа в виде списка рёбер. Позже, при анализе, обоснуем этот выбор
2. Случайную последовательность ребёр графа $G = (V, E)$ можно сгенерировать за $\Theta(|E|)$, сгенерировав последовательность индексов от 0 до $|E| - 1$ за $\Theta(|E|)$ и перемешав её при помощи std::shuffle за $\Theta(|E|)$. Тогда рёбра можно будет выбирать по индексам в списке рёбер.
3. Внешний цикл выполняется за $\Theta(|E|)$
Проверка на наличие циклов в неориентированном графе - dfs за $O(|V| + |E|)$
Добавление ребра делается за $O(1)$, удаление ребра - за $O(|E|)$
Аналогично рассуждениям в предыдущей задаче, хранение графа в виде матрицы смежности/сопряжённости не улучшит асимптотику самой дорогой операции в цикле, но может ухудшить её (например, если $|E| = O(|V|)$).
4. Тогда весь алгоритм работает за $= O\left(|E| + |E| \cdot (|E| + |V|)\right) = O\left(|E| \cdot (|E| + |V|)\right)$

Алгоритм ALG_3

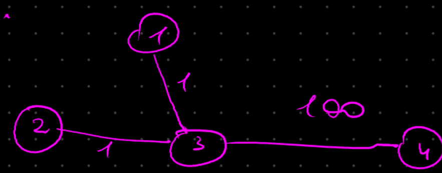
1. В данном задаче будем хранить графа в виде списка рёбер. Позже, при анализе, обоснуем этот выбор
2. Добавление ребра делается за $O(1)$, удаление ребра - за $O(|E|)$
3. Случайную последовательность ребёр графа $G = (V, E)$ можно сгенерировать за $\Theta(|E|)$,
сгенерировав последовательность индексов от 0 до $|E| - 1$ за $\Theta(|E|)$ и перемешав её
при помощи std::shuffle за $\Theta(|E|)$. Тогда рёбра можно будет выбирать по индексам в списке рёбер.
3. Внешний цикл выполняется за $\Theta(|E|)$
Поиск цикла в неориентированном графе - dfs за $O(|V| + |E|)$
Поиск ребра с максимальным весом в найденном цикле - за $O(V)$,
т.к. # рёбер в цикле = (# вершин в цикле $-$ 1) $< |V|$
Аналогично рассуждениям в предыдущей задаче, хранение графа в виде матрицы
смежности/сопряжённости не улучшит асимптотику самой дорогой операции в цикле, но может
ухудшить её (например, если $|E| = O(|V|)$).
4. Тогда весь алгоритм работает за $= O\left(|E| + |E| \cdot ((|E| + |V|) + |E| + |E|)\right) = O\left(|E| \cdot (|E| + |V|)\right)$

Исходный код приведён в файле *a1.cpp*

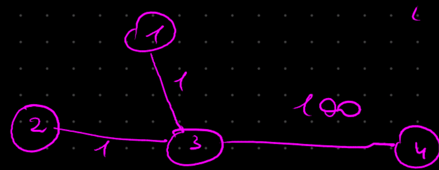
Корректность алгоритмов

Приведём контрпример, показывающий, что алгоритм ALG_1 не строит mst:

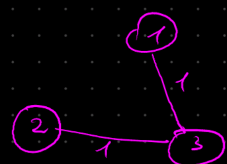
Пусть $G = (V, E)$:



1) $T = E = \{\{1,3\}, \{2,3\}, \{3,4\}\}$



2) $T = \{\{1,3\}, \{2,3\}\}$

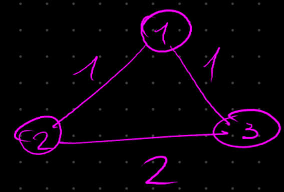


— связный граф, у которого больше нельзя удалять рёбра.

При этом T не mst, т.к. T не остовное дерево, т.к. вершина $4 \notin V_T$

Приведём контрпример, показывающий, что алгоритм ALG_2 не строит mst:

Пусть $G = (V, E)$:



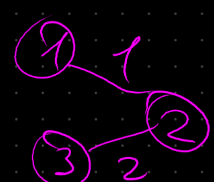
1) Изначально $T = \varnothing$

Выберем ребро $(2,3)$, далее ребро $(1,2)$, а потом ребро $(1,3)$

Тогда алгоритм построит такое множество рёбер:

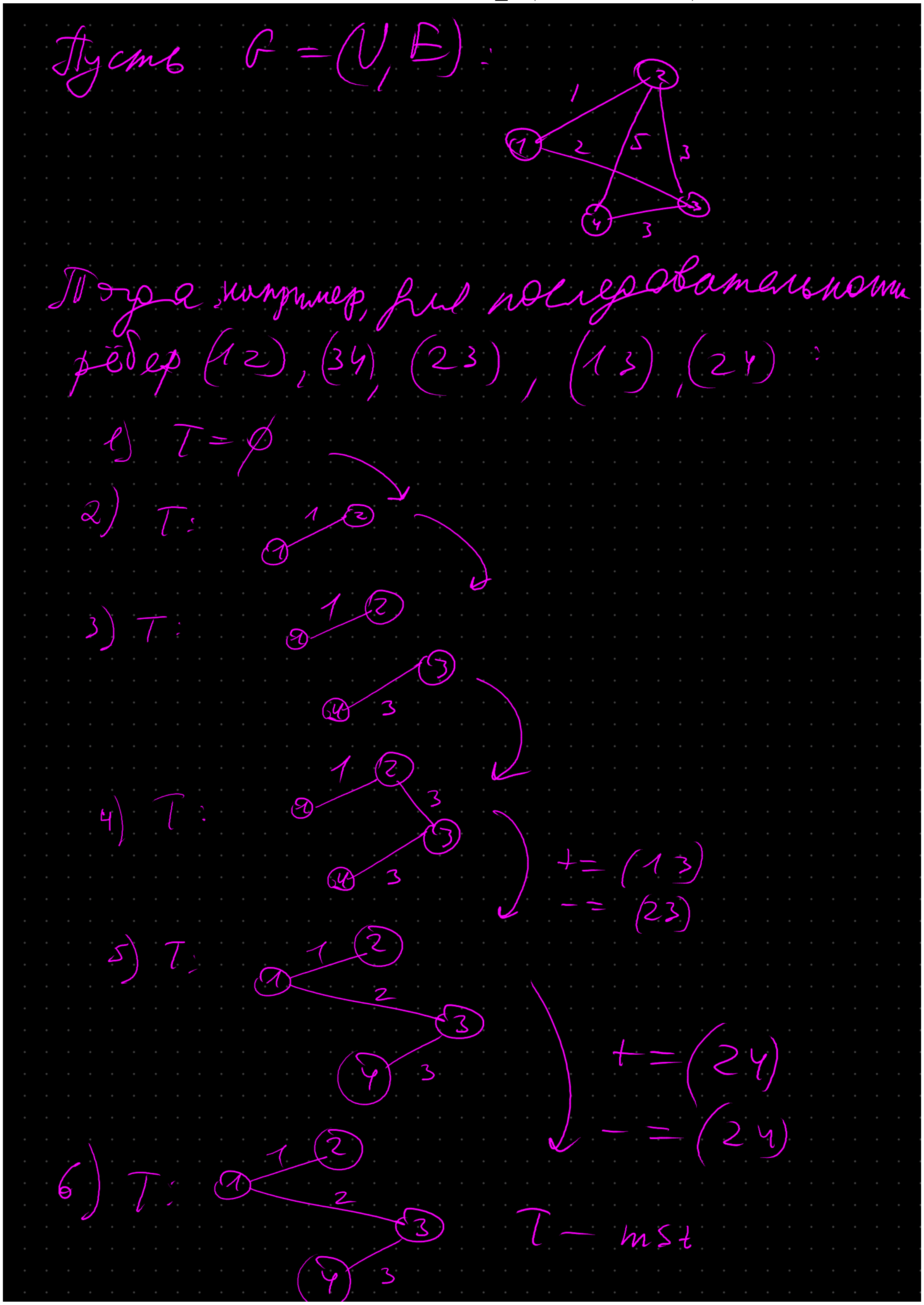
$T = \{(2,3), (1,2)\}$

Граф T:



— не mst, т.к. вес рёбер равен $3 > 2$

Приведём контр пример работы алгоритма ALG_3 (построение mst):



Lemma Лемма 1 об алгоритме ALG_3

Алгоритм ALG_3 строит остовное дерево данного графа $G = (V, E)$

Proof:

1. Здесь и далее будем обозначать через T граф, который построен на рёбрах,
которые получил алгоритм ALG_3 после окончания работы.
2. До начала итераций цикла T - пустое множество, далее, на первых двух итерациях
(или одной, если $|E| = 1$) в T добавляются 2 ребра, цикл образоваться не может
3. На каждой итерации из графа T удаляется ребро e_{max} после добавления
некоторого ребра e тогда и только тогда, когда после добавления
ребра e образовался цикл c, и при этом $e_{max} \in c$.
Пусть в T есть цикл. Тогда, на какой-то итерации алгоритма образовалось сразу хотя бы 2 цикла
(если на каждой итерации образовывается не более 1 цикла, то они сразу удаляются)
Рассмотрим первую среди таких итераций, на которой образовалось сразу 2 цикла.
Пусть на этой итерации было добавлено ребро $e = \{u, v\}$, тогда до добавления ребра
в графе не было циклов
(иначе, это не первая итерация, на которой появилось сразу хотя бы 2 цикла)
Т.к. на данной итерации появилось сразу хотя 2 цикла, то между вершинами
u и v было хотя бы 2 различных пути:
если между ними не было пути, то не образовался бы ни один цикл, а если был только один
путь, то образовался бы 1 цикл
Но раз между ними уже было хотя бы 2 различных пути, то в графе уже был цикл, что неверно,
т.е. пришли к противоречию, предположив, что в T есть цикл.
4. Т.к. в T рёбра удаляются только из циклов и только по-одному ребру, то граф связен
При этом, каждое ребро исходного графа добавлялось в T, тогда множество вершин графа T
совпадает с множеством вершин графа G (рёбра удаляются, только если они в цикле)
5. Доказали, что T - связный граф без циклов $\implies$ T - дерево.
Тогда $T = (V, E_T) \implies$ T - остовное дерево графа G.

■

Lemma Лемма 2 об алгоритме ALG_3

Алгоритм ALG_3 строит минимальное остовное дерево данного графа $G = (V, E)$

Proof.

1. По лемме 1 ALG_3 построил некоторое остовное дерево $T = (V, E_T)$ графа G.

В данной лемме через "mst" будем обозначать фразу "минимальное остовное дерево"

Рассмотрим некоторые крайние случаи:

$|V| = 0 \implies E_T = \varnothing \implies T - mst$ графа G

$|V| = 1 \implies E_T = \varnothing \implies T - mst$ графа G

$|V| = 2 \implies V = \{u, v\} \implies E_T = \{\{u, v\}\} \implies T - mst$ графа G

Показали, что при $|V| \leq 2$ лемма верна. Пусть $|V| > 2 \implies |V| \geq 3$

3. Граф G связен $\implies \exists$ хотя бы одно mst графа G.

Возьмём произвольное mst графа G, обозначив его как T', и покажем, что при помощи преобразований над рёбрами графа T', которые сохраняют его свойство быть минимальным остовным деревом графа G, из T' можно получить T.

4. Если $T = T'$, то лемма доказана. Пусть $T \neq T'$.

T и T' - остовные деревья графа G $\implies |E_T| = |E_{T'}| = |V| - 1 \geq 2$

$T = (V, E_T) \wedge T' = (V, E_{T'}) \wedge T \neq T' \implies E_T \neq E_{T'}$

$E_T \neq E_{T'} \wedge |E_T| \geq 2 \wedge |E_{T'}| \geq 2 \implies \exists (u, v) \in V^2 : \{u, v\} \in E_T \wedge \{u, v\} \notin E_{T'}$

T' - связный граф $\implies$ в графе T' есть путь из вершину u в вершину v.

Обозначим вершины этого пути как $u, p_1, p_2, ..., p_k, v$ (быть может, k = 1)

На картинке это можно изобразить так:

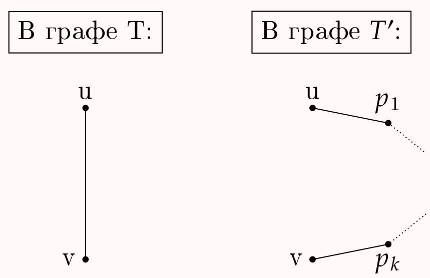


Figure 1

При этом, вершины $p_1, p_2, ..., p_k \in$ графу $T' \implies \{p_1, p_2, ..., p_k\} \subseteq V \implies$

$\implies$ эти вершины есть и в графе T (т.к. T - остовное дерево графа G)

5. Любое ребро из E_T и $E_{T'}$ есть в $E \implies$ в исходном графе G есть цикл $(u, p_1, ..., p_k, v, u)$

I. Если в графе T нет ребра $\{v, p_k\}$, тогда:

$\boxed{\text{В графе T:}}$ $\boxed{\text{В графе } T'\text{:}}$

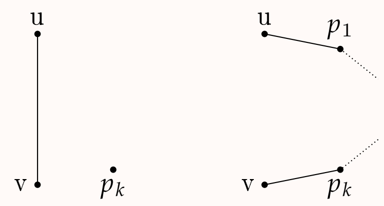


Figure 2

I.1 Ребра $\{v, p_k\}$ нет в T $\implies$ оно было удалено во время работы алгоритма.
Оба ребра $\{u, v\}$ и $\{v, p_k\}$ есть в графе G и принадлежат одному циклу, поэтому алгоритм мог удалить любое из них, не нарушив связность T.
Из того, что алгоритм удалил $\{v, p_k\}$, следует, что $w(v, p_k) \geq w(u, v)$
Это верно, т.к. если $\{v, p_k\}$ удалил из-за образования цикла после вставки $\{u, v\}$, то $w(v, p_k) \geq w(u, v)$
А если $\{v, p_k\}$ удалили из-за другого цикла c, когда в графе ещё не было $\{u, v\}$, то $w(v, p_k) \geq$ максимального веса рёбер в цикле c, из-за которого его удалили, но это цикл c точно проходил через вершины $\{v, p_k\}$ $\implies$ из-за того, что в G есть цикл $(u, p_1, ..., p_k, v, u)$, ребро $\{u, v\}$ также попадало в цикл с рёбрами из c, но осталось в графе $\implies$
$\implies$ его вес не больше максимального веса рёбер в c $\implies$ $w(u, v) \leq w(v, p_k)$
I.2 Если $w(u, v) < w(v, p_k)$, то в T' можно удалить ребро $\{v, p_k\}$ и добавить ребро $\{u, v\}$
Множество вершин графа T' при этом не изменится, граф останется связным и в нём не появится цикл, т.е. граф останется остовным деревом графа G.
Но при этом его (графа T') вес уменьшится, что противоречит тому, что $T' - mst$ $\implies \perp \implies$
$\implies$ предположение, что $w(u, v) < w(v, p_k)$, неверено $\implies$ $w(u, v) = w(v, p_k)$
I.3 Тогда построим граф T'' так: уберём из $E_{T'}$ ребро $\{v, p_k\}$ и добавим ребро $\{u, v\}$, т.е.
$E_{T''} := \big(E_{T'} \setminus \{\{v, p_k\}\}\big) \cup \{\{u, v\}\}$
В графе T'' не появится цикл, он останется связным, и его вес не изменится, т.е. останется равным весу графа T' $\implies$ $T'' - mst$ графа G.
И при этом его множество рёбер $E_{T''}$ станет на 1 ребро "ближе" к множеству вершин E_T
Формально, $|E_T \cap E_{T''}| = |E_T \cap E_{T'}| + 1$
На картинке это можно изобразить так:

$\boxed{\text{В графе T:}}$ $\boxed{\text{В графе } T''\text{:}}$

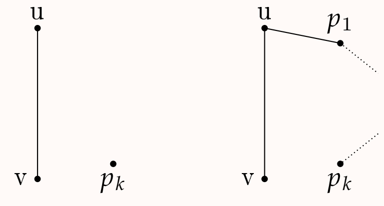


Figure 3

II. Если в графе T есть ребро $\{v, p_k\}$, тогда:

В графе T: В графе T':

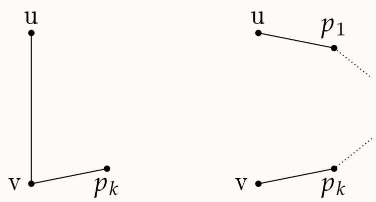


Figure 4

II.1. Рассмотрим рёбра $\{u, p_1\}, ..., \{p_k, v\}$ графа T' (это множество не пусто, т.к. $k \geq 1$)
Если они все $\in E_T$, то в T есть цикл $(u, p_1, ..., p_k, v, u) \implies$ T - не дерево $\implies \perp$
Пусть ребра $\{p_i, p_{i+1}\}$ (здесь $0 \leq i \leq k-1, p_0 := u$) нет в графе T
На картинке это можно изобразить так:

В графе T: В графе T':

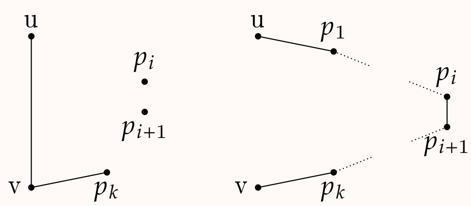


Figure 5

II.2 Ребра $\{p_i, p_{i+1}\}$ нет в T $\implies$ оно было удалено во время работы алгоритма.
По аналогии с пунктом I.1 $w(p_i, p_{i+1}) \geq w(u, v)$
II.3 Если $w(u, v) < w(p_i, p_{i+1})$, то в T' можно удалить ребро $\{p_i, p_{i+1}\}$ и добавить ребро $\{u, v\}$
Множество вершин графа T' при этом не изменится, граф останется связным и в нём
не появится цикл, т.е. граф останется остовным деревом графа G.
Но при этом его (графа T') вес уменьшится, что противоречит тому, что $T' - mst \implies \perp \implies$
$\implies$ предположение, что $w(u, v) < w(p_i, p_{i+1})$, неверено $\implies w(u, v) = w(p_i, p_{i+1})$
II.4 Тогда построим граф T'' так: уберём из $E_{T'}$ ребро $\{p_i, p_{i+1}\}$ и добавим ребро $\{u, v\}$, т.е.
$E_{T''} = \left(E_{T'} \setminus \{\{p_i, p_{i+1}\}\}\right) \cup \{\{u, v\}\}$
В графе T'' не появится цикл, он останется связным, и его вес не изменится, т.е. $T'' - mst$ графа G.
И при этом его множество рёбер $E_{T''}$ станет на 1 ребро "ближе" к множеству рёбер E_T
Формально, $|E_T \cap E_{T''}| = |E_T \cap E_{T'}| + 1$

На картинке это можно изобразить так:

$\boxed{\text{В графе T:}}$ $\boxed{\text{В графе } T'':}$

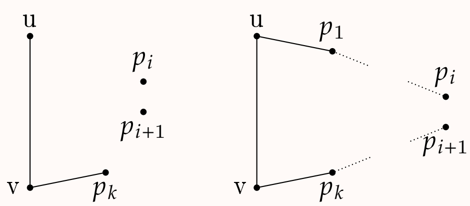


Figure 6

6. Таким образом, привели пример итерационного алгоритма, который на каждой итерации преобразует $mstT^j$ в $mstT^{j+1}$ путём удаления одного ребра из E_{T^j} и добавления
одного ребра из E_T, которого ранее не было в E_{T^j}, так, что множества вершин графов T^j и T^{j+1} совпадают и равны V, и при этом выполняется равенство $|E_T \cap E_{T^{j+1}}| = |E_T \cap E_{T^j}| + 1$

$\forall j : |E_{T^j}| = |V| - 1 \implies \forall j : |E_T \cap E_{T^j}| \leq |V| - 1$

Тогда через конечное число итераций алгоритма получим $mstT^l$, такое что: $|E_T \cap E_{T^l}| = |V| - 1$

$(|E_T| = |V| - 1) \wedge (|E_{T^l}| = |V| - 1) \wedge (|E_T \cap E_{T^l}| = |V| - 1) \implies E_T = E_{T^l}$

У графов T и T^l множества вершин совпадают $\implies T = T^l$

Таким образом, через конечное число итераций получим $mstT^l$, равное T $\implies T - mst$ графа G

■