Алгоритмы и структуры данных. Домашнее задание №7

Выполнил студент Наседкин Дмитрий Сергеевич (группа 242)

Письменная часть

№ 1

Для начала докажем, что если граф изначально был связным, то алгоритм построит остовное дерево, для этого просто заметим, что после каждого удаления ребра граф остается связным(иначе ребро не удаляется по алгоритму), при этом если в конце алгоритма получили не дерево, то есть существует цикл, а значит получаем противоречие, так как из этого цикла можно было удалить 1 ребро, а алгоритм этого не сделал.

Теперь докажем минимальность, для этого удобно доказать следующее утверждение:

После каждого шага алгоритма среди оставшихся(то есть тех, которые алгоритм еще не рассмотрел) и неудаленных ребер точно есть хотя бы одно MST исходного графа.

По индкуции:

- База: так как граф связен, то в нем точно есть хотя бы один остов, выберем среди них минимальный, и так как мы еще не удаляли ребра, то утверждение верно.
- Переход: Пусть мы рассматриваем очередное ребро e, а для всех прошлых шагов утверждение верно:
  - Если удаление e нарушает связность, то мы его не удаляем, а значит множество оставшихся и неудаленных ребер не изменилось, следовательно и для текущего шага утверждение верно
  - Иначе удаление e не нарушает связность, то есть среди оставшихся и неудаленных ребер есть цикл C содержащий e. Пусть также T - какое-то MST, которое было до шага алгоритма(оно есть из предположения индукции).
    * Если $e \notin T$, то его удаление оставляет хотя бы одно MST исходного графа, а значит утверждение верно на нашем шаге.
    * Иначе $e \in T$, тогда рассмотрим такое ребро цикла C, которое соединяет 2 компоненты связности, образующиеся при удалении e(оно точно есть, так как C цикл), назовем его f. Заметим, что $f \leq e$, то есть оно лежало среди оставшихся(нерасмотренных) ребер, так как иначе алгоритм должен был бы на прошлых шагах удалить его(оно не нарушало связности так как лежит в том же цикле C что и e). А значит удалив e и заменив его на f получим остов не большего веса, то есть существует хотя бы одно MST исходного графа, утверждение верно.

Таким образом мы доказали, что после всех шагов алгоритма получится остов, а среди неудаленных ребер есть MST, следовательно это остов минимального веса.

№ 2

Сведем нашу задачу к **2-SAT**, и тогда если у нас будет $O(n)$ переменных и $O(m)$ скобок, то алгоритм будет работать за $O(n + m)$. Алгоритм 2-SAT хорошо обсуждался на семинарах, так что именно его я описывать не буду.

Пусть вершина v покрашена в цвет c_v, тогда ее можно перекрасить в 1 из 2 оставшихся цветов, назовем их a_v и b_v ($a_v \neq b_v$). Введем

$$x_v = \begin{cases} 1, & \text{если новый цвет } v \text{ равен } a_v \\ 0, & \text{если новый цвет } v \text{ равен } b_v \end{cases}$$

Чтобы расскраска была верной требуется, чтобы цвета вершин на любом ребре были различны, то есть для ребра (v, u) появляются ограничения вида:

$$(x_u = \alpha \ \wedge \ x_v = \beta)$$

(Где α и β обозначают один цвет, просто они имеют разные значения для x_v и x_u).

То есть должно выполняться логическое **И** следующих выражений:

$$\neg(x_u = \alpha) \ \vee \ \neg(x_v = \beta)$$

Заметим, что для каждого ребра появится 1 скобка, если цвета концов были различны, и 2 скобки, если цвета были одинаковые, то есть у нас действительно $O(n)$ переменных и $O(m)$ скобок.

Тогда запустив 2-SAT получим значения x_v, по которым получим новые цвета. (Вообще я не очень понял, по видимому гарантируется, что существует корректная расскраска, тогда 2-SAT точно найдет решение, но даже если не гарантируется, то 2-SAT скажет, что решения нет, так что и так все ок)

Устная и письменная часть

P.S. Мини конспекты для себя, сдача устно:

№ 3

Построим слоистую сеть запустив поиск в ширину из s, и аналогично для t на обратных ребрах, таким образом найдем `dist(s, v)` и `dist(v, t)` - кратчайшие расстояния от s до всех и от всех до t.

Пусть мы взяли вершинку v и u, заметим, что тогда новое расстояние проходящее через ребро vu станет `dist(s, v) + 1 + dist(u, t)`, причем нам нужно чтобы `dist(s, t)` уменьшилось, а значит зафиксировав v легко выбрать количество нужных вершинок u с помощью префиксных сумм.

№ 4

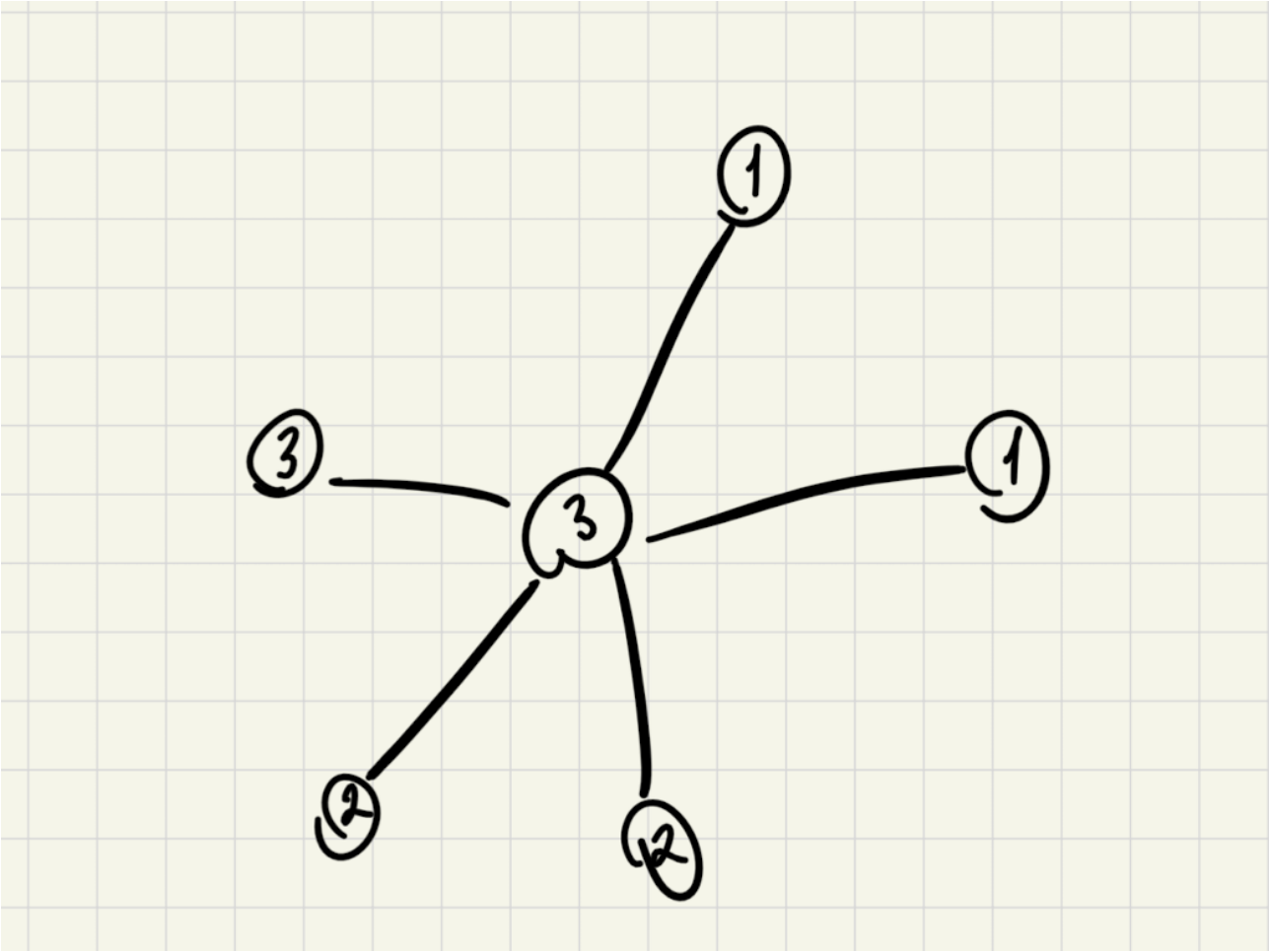
Давайте вначале присвоим всем вершинам цвет 1, после чего "исправлять"плохие ребра(плохими называем ребра у которого концы одного цвета).

Алгоритм следующий:

- Пусть существует вершина v, у которой больше 1 плохого ребра, иначе задача решена и можно завершиться
- Найдем среди соседей v цвет который встречается меньше всего, заметим, что будет цвет, который встречается не больше 1 раза. ($cnt_{c_0} \geq 2 \Rightarrow cnt_{c_1} + cnt_{c_2} = deg(v) - cnt_{c_0} \leq 5 - 2 = 3 \Rightarrow \min(cnt_{c_1}, cnt_{c_2}) \leq 1$).
- Покрасим v в этот цвет. Заметим, что плохих ребер уменьшилось хотя бы на 1. Следовательно алгоритм закончится не более чем за m итераций.

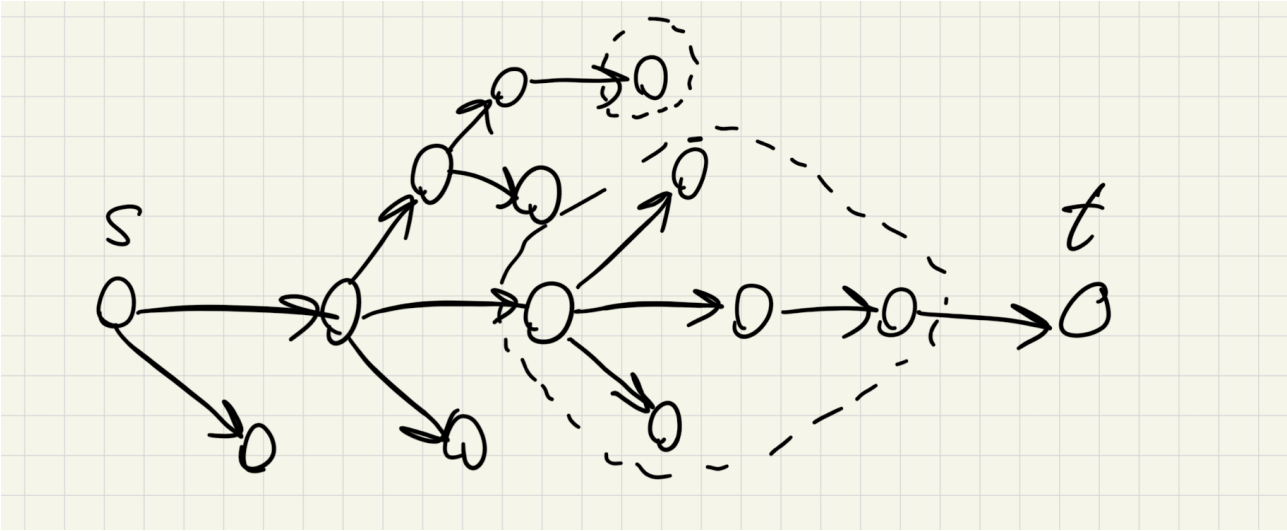
Второе и третье действие делается очевидно за $O(1)$, так как степень вершины не больше 5, значит осталось научится находить плохие вершины, однако для этого можно просто поддерживать количество цветов соседей при действиях,

Вершины можно поддерживать, например, в хеш-таблице



№ 5

Посчитаем кратчайшие расстояния от s до всех остальных и от всех до t Дейкстрой с помощью кучи за $O((n + m)\log n). Будем перебирать все вершинки $u \in A$, тогда какая длина пути будет от s до t? Очевидно это будет либо старый кратчайший путь от s до t либо он будет проходить через ребро новое ребро uv веса c, но важно чтобы его не было в исходном графе. Если его нет, то в этом случае его легко посчитать, это будет `dist(s, u) + c + dist(v, t)`, заметим, что левая часть выражения фиксирована, а значит чтобы минимизировать ответ нужно минимизировать `dist(v, t)`, такое, что не существует ребра uv. Для этого отсортируем все $d(v, t)$, такие что $v \in A$. Тогда будем перебирать по отсортированному списку явно проверяя, существует ли такое ребро, заметим, что неподходящих ребер суммарно будет не больше m, а значит все чик-пук.



№ 6

Найти точку, минимизирующую максимальное расстояние от нее до всех остальных вершин.

Найти минимум $e(\gamma) = \max\limits_{w \in V} d(\gamma, w) = \max\limits_{w \in V}[\min(\alpha + d(u, w), w_{uv} - \alpha + d(v, w))]$

а) Тернарник, либо все точки интереса ($O(n^3 + n^2 + n^3 m)$)

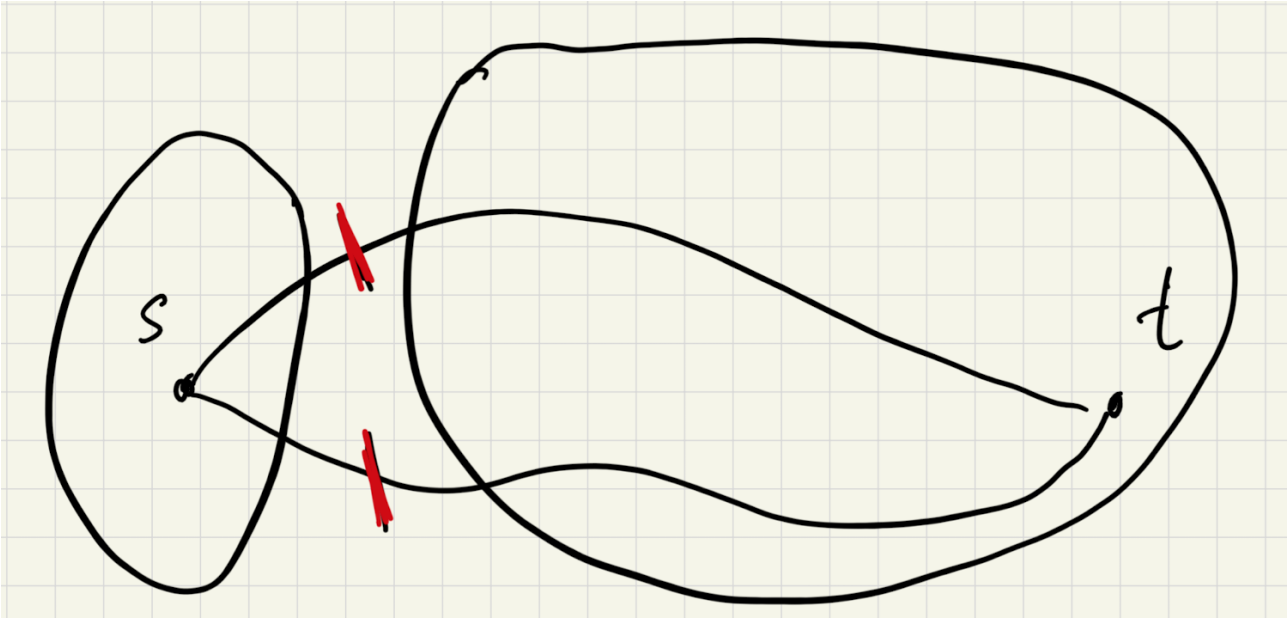
б) Минимум линейных функций (Построить все прямые).

№ 7

а) просто перебираем одно удаляемое ребро из графа, после чего удаляем его явно за $O(n + m)$ и ищем мосты за $O(n + m)$, то есть суммарно получится $O((n + m) * m) = O(m^2)$

б) Изначально проверим, если ли в графе мосты за $O(n + m)$, если да, то 2-мост очевидно есть, инчае построим сеть следующим образом, истоком будет произвольная вершина s, а сток t будем перебирать($n - 1$ вариант), пропускная способность каждого ребра равна 1. Тогда если найдется минимальный разрез пропускной способности 2, то очевидно существует 2 мост(как раз ребра разреза). При этом заметим, что достаточно лишь запустить Форда-Фалкерсона за $O(m)$ не более 3 раз. Докажем, что если алгоритм не нашел такой разрез, то 2-моста нет:

- от противного: пусть 2-мост существует, тогда рассмотрим вершины s, t в разных компонентах связности, заметим, что запустив поток с таким истоком и стоком мы найдем нужный нам разрез. См. картинку

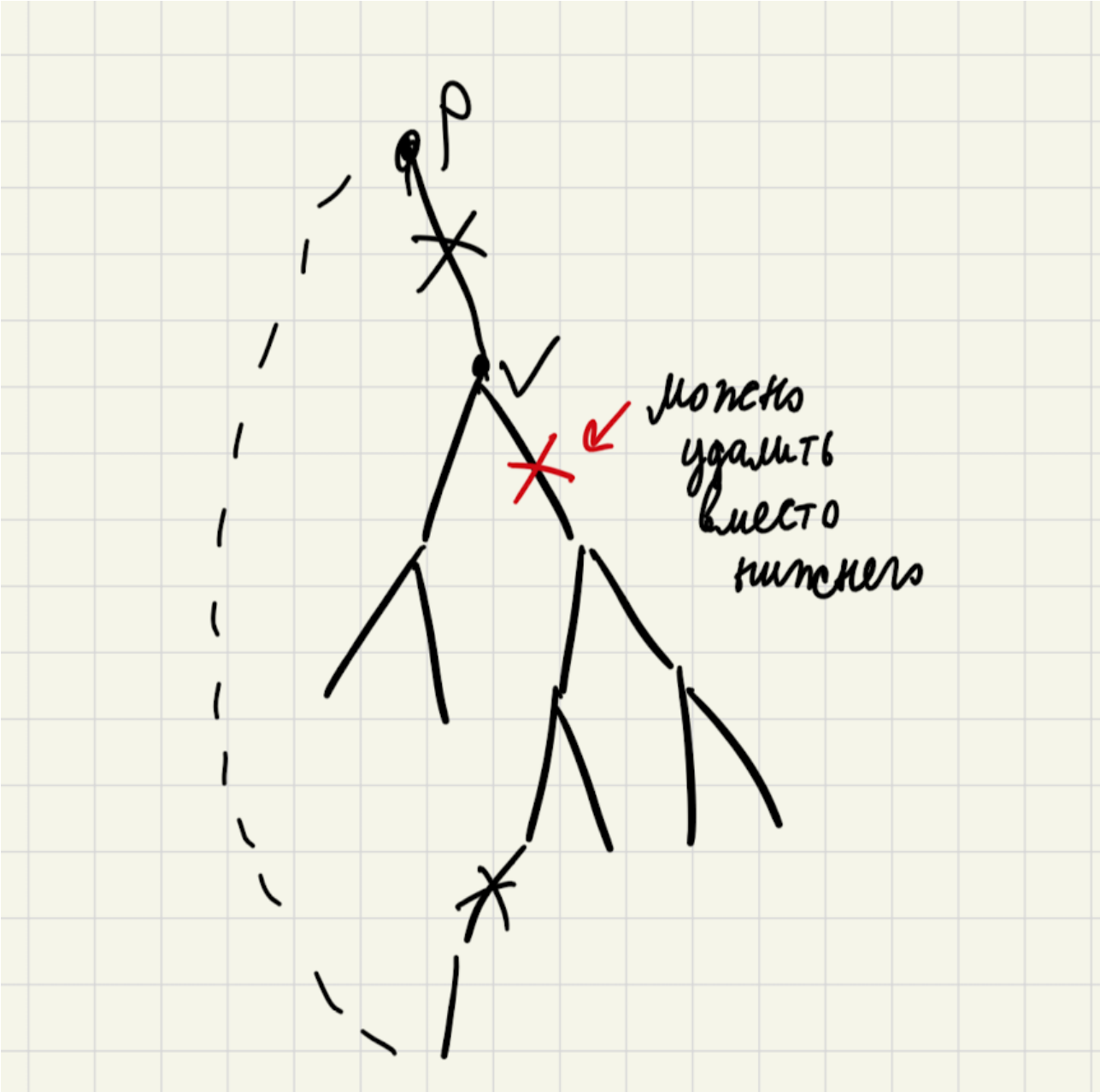


Алгоритм очевидно работает за $O(nm)$.

с) Построим дерево `DFS`, очевидно, что 2-мост не может состоять из 2 пунктирных ребер, а значит это обязательно либо жирное-пунктирное, либо жирное-жирное.

Пусть мы хоти удалить жирное ребро, заметим, что имеет смысл удалять только жирные и пунктирные ребра из нашего поддерева. Поддерживая 2 вершины ведующие максимально вверх можем легко проверить случай жирное-пунктирное.

Утверждение: Пусть 2-мост это жирное-жирное, тогда существует 2-мост из жирного-жирного, что они идут подряд. Док-во через случаи. (См. картинку)



Тогда построив преф, суф максимумы можем проверять наличие 2-жирного моста.