

Мини-задача #36 (2 балла)

Реализовать алгоритм решения описанной в лекции задачи планирования с использованием структуры union-find.

Достигнуть при этом сложности $O(N * \alpha(N))$.

Проверить решение на собственном наборе тестов, который бы демонстрировал некорректность наивного жадного алгоритма.



Алгоритмы и структуры данных

Single-link кластеризация, жадные алгоритмы на
матроидах.



Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов)

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Задача кластеризации

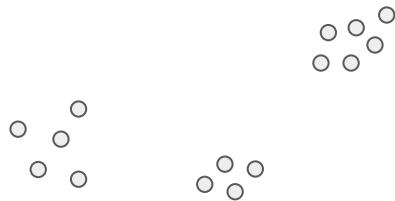
Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Задача неформально: разбить множество точек на группы "похожих" между собой

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

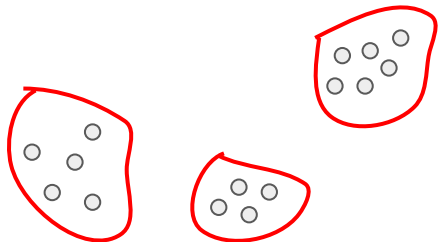
Задача неформально: разбить множество точек на группы "похожих" между собой



Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Задача неформально: разбить множество точек на группы "похожих" между собой



Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

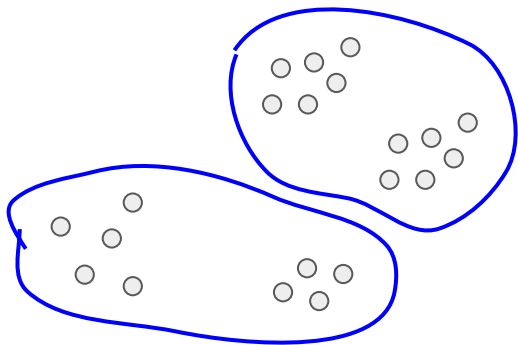
Как понять, что одно разделение на кластеры лучше, чем другое?

Задача кластеризации

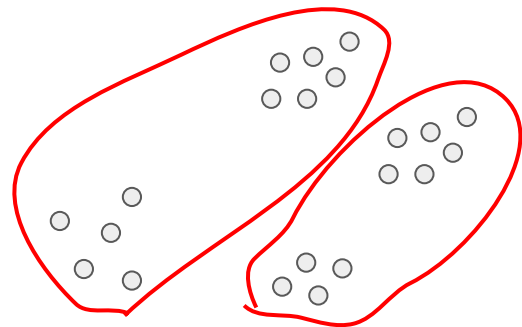
Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

Как понять, что одно разделение на кластеры лучше, чем другое?



или



?

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

Идея: лучше бы, чтобы получившиеся кластеры были "далеко" друг от друга.

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

Идея: лучше бы, чтобы получившиеся кластеры были "далеко" друг от друга. Расстояние между кластерами = минимальное расстояние между элементами этих кластеров (вот это и называется **single-link**).

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

Идея: лучше бы, чтобы получившиеся кластеры были "далеко" друг от друга. Расстояние между кластерами = минимальное расстояние между элементами этих кластеров (вот это и называется **single-link**).

Хотим **максимизировать** расстояние между кластерами.

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

Расстояние между кластерами: $\min_{p \in A, q \in B} d(p, q)$, где A и B — различные кластеры

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

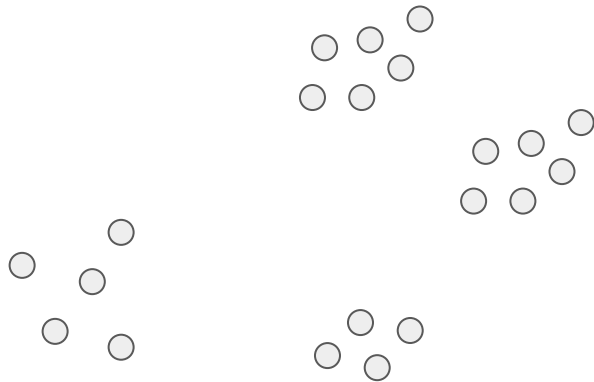
Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

Расстояние между кластерами: $\min_{p \in A, q \in B} d(p, q)$, где A и B — различные кластеры

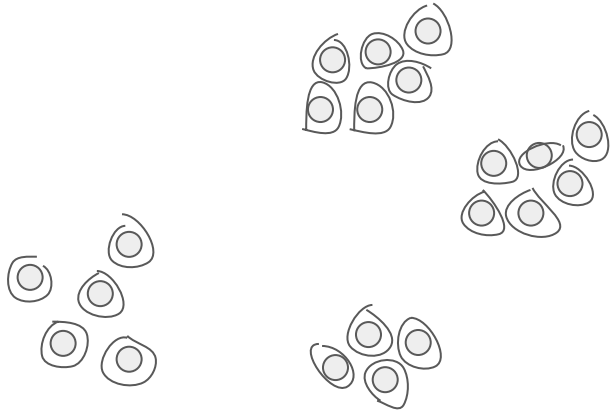
Задача: максимизировать эту величину.

Алгоритм кластеризации

Пусть $K = 2$



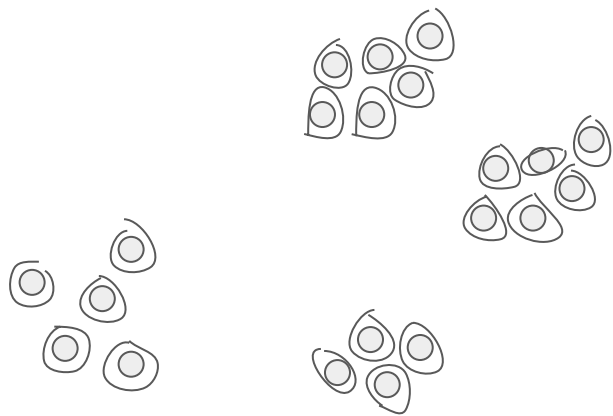
Алгоритм кластеризации



Пусть $K = 2$

Изначально каждая точка в
своем кластере

Алгоритм кластеризации

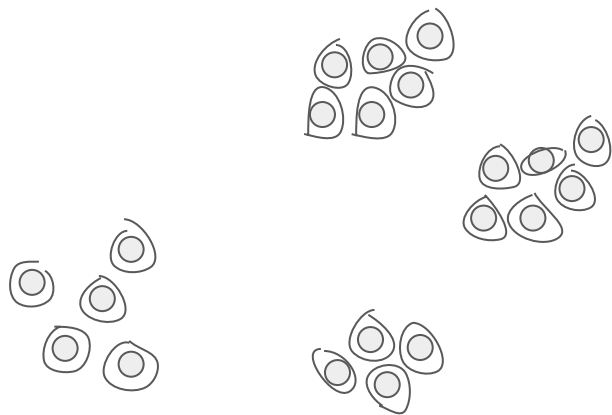


Пусть $K = 2$

Изначально каждая точка в своем кластере

Если мы хотим сократить количество кластеров, какой **жадный** шаг нам стоит сделать?

Алгоритм кластеризации



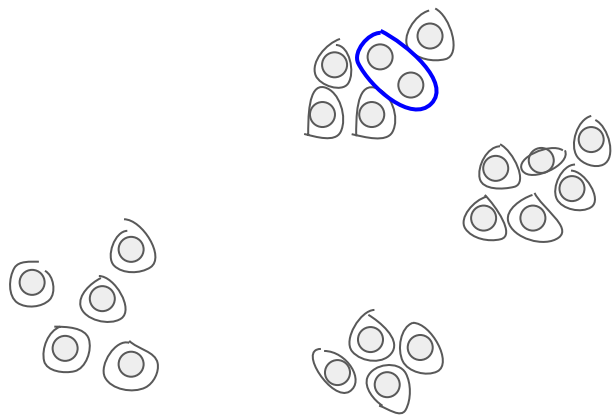
Пусть $K = 2$

Изначально каждая точка в своем кластере

Если мы хотим сократить количество кластеров, какой **жадный** шаг нам стоит сделать?

Найти две ближайшие точки из разных кластеров и слить их кластеры!

Алгоритм кластеризации



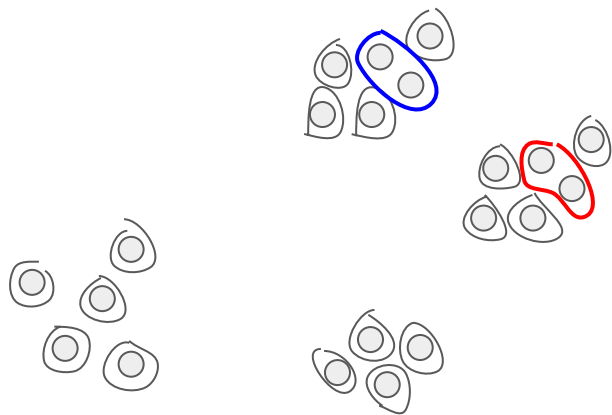
Пусть $K = 2$

Изначально каждая точка в своем кластере

Если мы хотим сократить количество кластеров, какой **жадный** шаг нам стоит сделать?

Найти две ближайшие точки из разных кластеров и слить их кластеры!

Алгоритм кластеризации



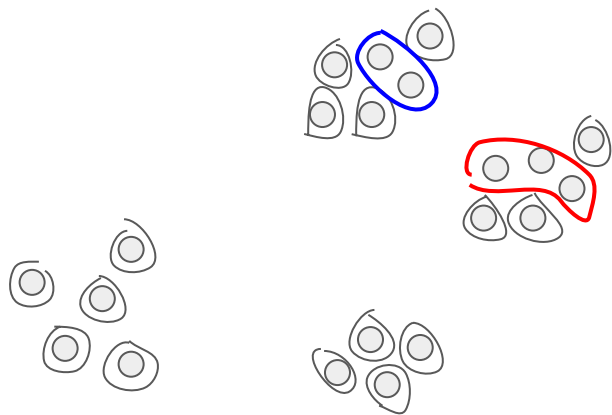
Пусть $K = 2$

Изначально каждая точка в своем кластере

Если мы хотим сократить количество кластеров, какой **жадный** шаг нам стоит сделать?

Найти две ближайшие точки из разных кластеров и слить их кластеры!

Алгоритм кластеризации



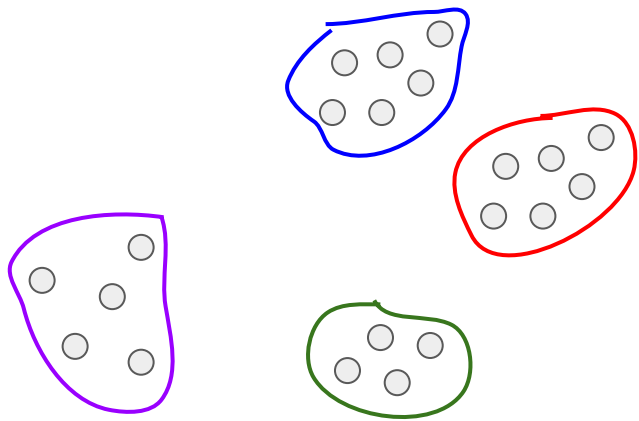
Пусть $K = 2$

Изначально каждая точка в своем кластере

Если мы хотим сократить количество кластеров, какой **жадный** шаг нам стоит сделать?

Найти две ближайшие точки из разных кластеров и слить их кластеры!

Алгоритм кластеризации



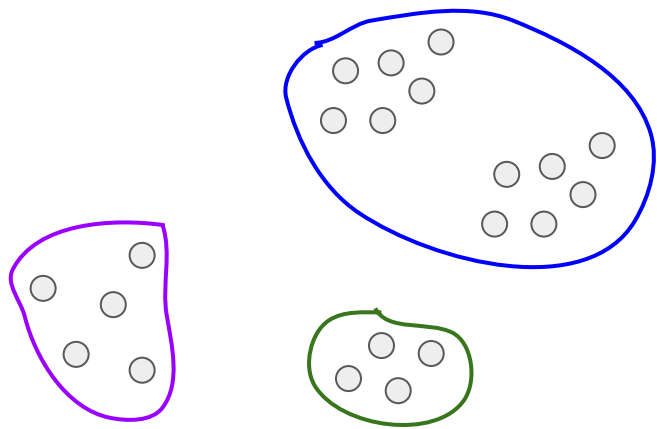
Пусть $K = 2$

Изначально каждая точка в своем кластере

Если мы хотим сократить количество кластеров, какой **жадный** шаг нам стоит сделать?

Найти две ближайшие точки из разных кластеров и слить их кластеры!

Алгоритм кластеризации



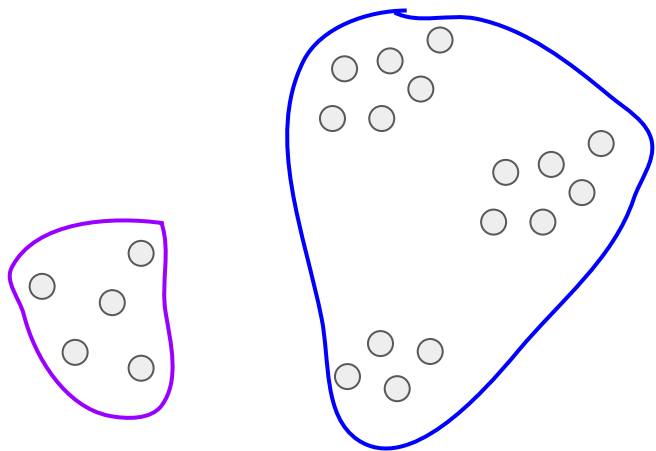
Пусть $K = 2$

Изначально каждая точка в своем кластере

Если мы хотим сократить количество кластеров, какой **жадный** шаг нам стоит сделать?

Найти две ближайшие точки из разных кластеров и слить их кластеры!

Алгоритм кластеризации



Останавливаемся, когда
осталось K кластеров

Пусть $K = 2$

Изначально каждая точка в
своем кластере

Если мы хотим сократить
количество кластеров, какой
жадный шаг нам стоит сделать?

Найти две ближайшие точки из
разных кластеров и слить их
кластеры!

Алгоритм кластеризации

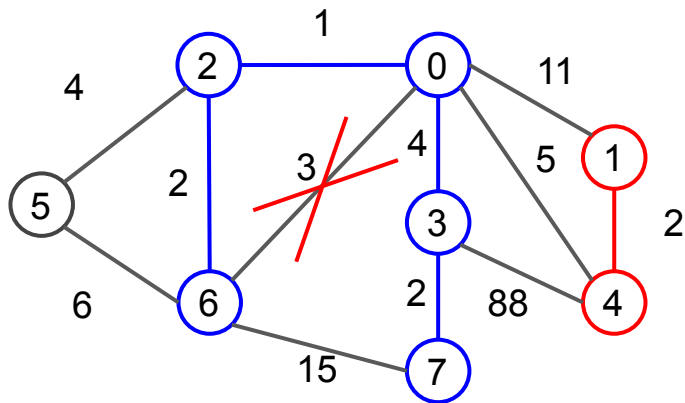
- 0) Изначально каждая точка в своем кластере
- 1) Идем по возрастанию расстояния между точками, если пара точек оказывается в **разных** кластерах, то объединяем их
- 2) Продолжаем, пока кластеров не останется **k** штук

Алгоритм кластеризации

- 0) Изначально каждая точка в своем кластере
- 1) Идем по возрастанию расстояния между точками, если пара точек оказывается в **разных** кластерах, то объединяем их
- 2) Продолжаем, пока кластеров не останется **k** штук

На что безумно похоже?

Алгоритм Краскала



- 0) Сортируем все ребра в порядке возрастания веса
- 1) Идем по получившемуся массиву ребер и добавляем ребро в результат T , но только, если это не создаст **ЦИКЛОВ**!

Алгоритм кластеризации

- 0) Изначально каждая точка в своем кластере
- 1) Идем по возрастанию расстояния между точками, если пара точек оказывается в **разных** кластерах, то объединяем их
- 2) Продолжаем, пока кластеров не останется **k** штук

На что безумно похоже? На алгоритм **Краскала**, который заканчивается раньше (когда у вас еще не остовное дерево, а остовный лес из k деревьев)



Алгоритм кластеризации: корректность

Зачем вообще доказывать корректность, мы же знаем, что Краскал строит минимальное остовное дерево (доказано)?

Алгоритм кластеризации: корректность

Зачем вообще доказывать корректность, мы же знаем, что Краскал строит минимальное остовное дерево (доказано)?

Но этого **недостаточно**. Наш алгоритм другой – он прерывается до того, как построит остовное дерево.

Алгоритм кластеризации: корректность

Зачем вообще доказывать корректность, мы же знаем, что Краскал строит минимальное остовное дерево (доказано)?

Но этого **недостаточно**. Наш алгоритм другой – он прерывается до того, как построит остовное дерево.

Да и в Краскале ничего не говорилось про "расстояние" между остовными деревьями.

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

S — расстояние между кластерами: $\min_{p \in A, q \in B} d(p, q)$, где A и B — различные кластеры

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$.

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) p и q на каком-то шаге алгоритма стали **ближайшими** и из-за них были **слиты** их кластеры.

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

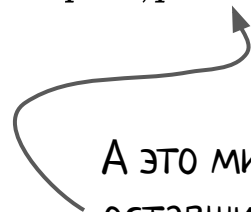
1) p и q на каком-то шаге алгоритма стали **ближайшими** и из-за них были **слиты** их кластеры. Тогда $d(p, q) \leq S$, т.к. алгоритм выбирал точки по возрастанию расстояния между ними.

Задача кластеризации

Дано: пусть есть набор "точек" (на самом деле — любых элементов) и функция $d(p, q)$ — "расстояние" между точками (на самом деле — любая **симметричная** функция)

Уточним: пусть мы знаем, что хотим получить именно **k** кластеров

Расстояние между кластерами: $\min_{p \in A, q \in B} d(p, q)$, где A и B — различные кластеры



А это минимум из оставшихся, которых еще не взяли!

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) p и q на каком-то шаге алгоритма стали **ближайшими** и из-за них были **слиты** их кластеры. Тогда $d(p, q) \leq S$, т.к. алгоритм выбирал точки по возрастанию расстояния между ними.

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) p и q на каком-то шаге алгоритма стали **ближайшими** и из-за них были **слиты** их кластеры. Тогда $d(p, q) \leq S$, т.к. алгоритм выбирал точки по возрастанию расстояния между ними.

А тогда $S \geq d(p, q) \geq \hat{S}$

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) p и q на каком-то шаге алгоритма стали **ближайшими** и из-за них были **слиты** их кластеры. Тогда $d(p, q) \leq S$, т.к. алгоритм выбирал точки по возрастанию расстояния между ними.

А тогда $S \geq d(p, q) \geq \hat{S}$, ведь это просто две какие-то точки из **разных** кластеров альтернативного разбиения, а $\hat{S} = \min_{x \in \hat{C}_i, y \in \hat{C}_j} d(x, y)$

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.

Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.



Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

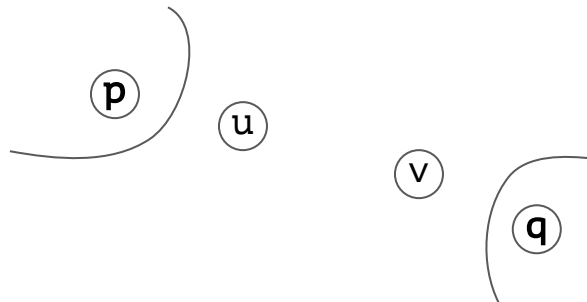
Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.



Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

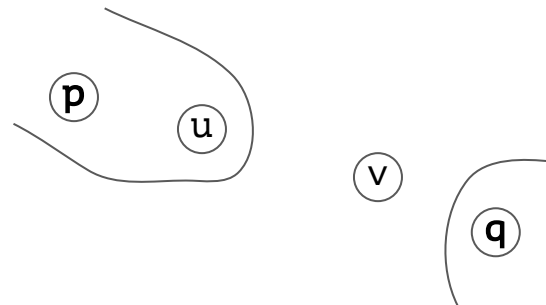
Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.



Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

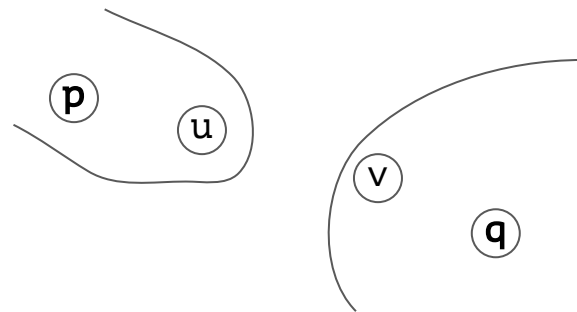
Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.



Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

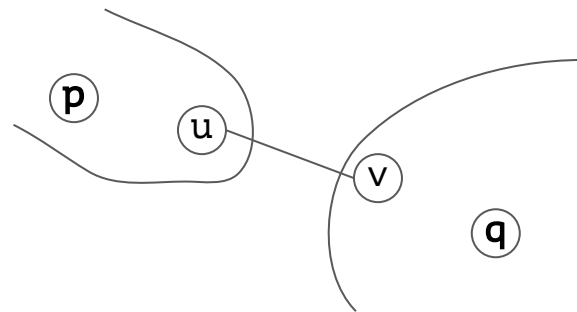
Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.



Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

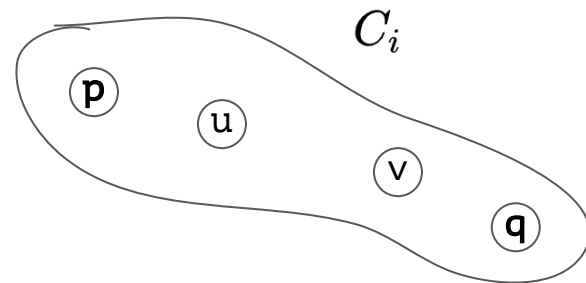
Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.



Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

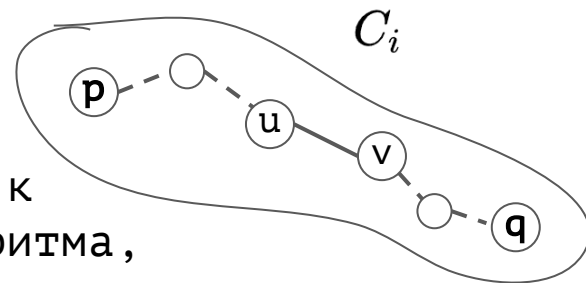
Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

- 1) ...
- 2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.

Значит есть последовательность точек:
 $p, p_1, \dots, u, v, \dots, q_1, q$, такая что каждые две соседние из последовательности приводили к слиянию кластеров на очередном шаге алгоритма,



Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

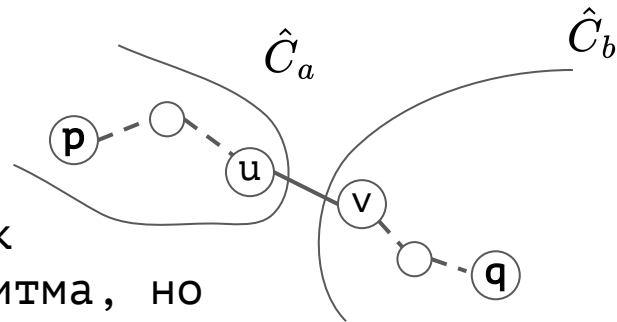
Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.

Значит есть последовательность точек:

$p, p_1, \dots, u, v, \dots, q_1, q$, такая что каждые две соседние из последовательности приводили к слиянию кластеров на очередном шаге алгоритма, но при этом $u \in \hat{C}_a$, а $v \in \hat{C}_b$



Теорема: описанный алгоритм single-link кластеризации дает оптимальное решение, т.е. кластеризацию с максимальным расстоянием.

Доказательство: пусть C_1, C_2, \dots, C_k кластеризация, полученная жадным алгоритмом, а S - его расстояние.

Пусть есть другая кластеризация: $\hat{C}_1, \hat{C}_2, \dots, \hat{C}_k$. Докажем, что его расстояние будет $\hat{S} \leq S$

Разбиения различные $\Rightarrow \exists p, q : p, q \in C_i$, но $p \in \hat{C}_a, q \in \hat{C}_b$. Тогда либо:

1) ...

2) либо p и q **никогда** не становились причиной слияния кластера, но все равно попали в один кластер.

Значит есть последовательность точек: $p, p_1, \dots, u, v, \dots, q_1, q$, такая что каждые две соседние из последовательности приводили к слиянию кластеров на очередном шаге алгоритма, но при этом $u \in \hat{C}_a$, а $v \in \hat{C}_b$

А значит вернулись к случаю 1) и снова: $S \geq d(u, v) \geq \hat{S} \quad \square$

Другие алгоритмы кластеризации

Графовые алгоритмы для кластеризации довольно **примитивны** на фоне современных и реальных аналогов, используются нами для примера **жадного алгоритма** (и решения большой задачи).

Другие алгоритмы кластеризации

Графовые алгоритмы для кластеризации довольно **примитивны** на фоне современных и реальных аналогов, используются нами для примера **жадного алгоритма** (и решения большой задачи).

Если вас заинтересовала тема, прочитайте про:

- EM-алгоритм,
- K-means,
- DBSCAN,
- ...

Когда вообще жадный алгоритм сработает?



Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$
 - Разменять 6 рублей монетами $\{1, 2, 5, 10\}$.
Как решать?

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$
 - Разменять 6 рублей монетами $\{1, 2, 5, 10\}$.
Как решать? Берем самую большую подходящую: $5 \rightarrow 1$

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$
 - Разменять 6 рублей монетами $\{1, 2, 5, 10\}$ -> **жадность**

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$
 - Разменять 6 рублей монетами $\{1, 2, 5, 10\}$ -> **жадность**
 - Разменять 6 рублей монетами $\{1, 3, 4\}$?

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$
 - Разменять 6 рублей монетами $\{1, 2, 5, 10\}$ -> **жадность**
 - Разменять 6 рублей монетами $\{1, 3, 4\}$?
Жадное решение даст $4 \rightarrow 1 \rightarrow 1$.

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$
 - Разменять 6 рублей монетами $\{1, 2, 5, 10\}$ -> **жадность**
 - Разменять 6 рублей монетами $\{1, 3, 4\}$?
Жадное решение даст $4 \rightarrow 1 \rightarrow 1$. Но **оптимальное**: $3 \rightarrow 3$

Применимость жадных алгоритмов

Знаем разные примеры:

- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$
 - Разменять 6 рублей монетами $\{1, 2, 5, 10\}$ -> **жадность**
 - Разменять 6 рублей монетами $\{1, 3, 4\}$ -> **не жадность**

Применимость жадных алгоритмов

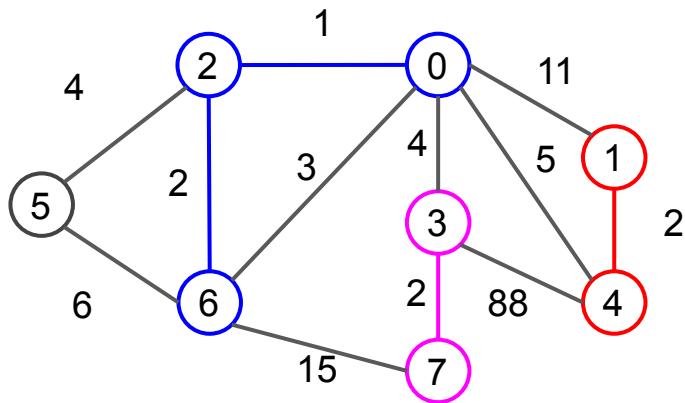
Знаем разные примеры:

Неужели каждый раз
придется проверять **заново**?



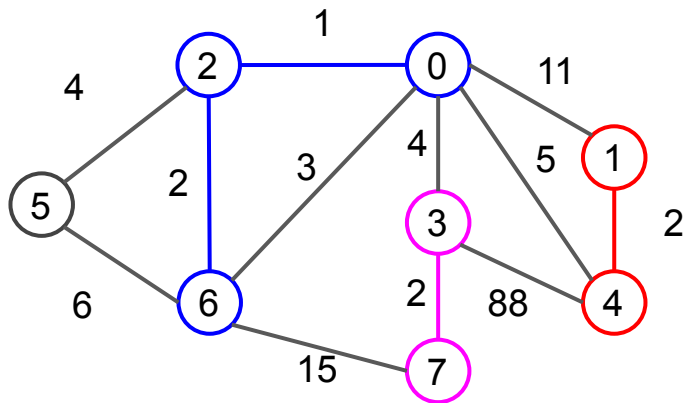
- Задача планирования с первой лекции:
один жадный алгоритм **подошел**, но второй - **нет**
- Поиск кратчайшего пути в графе: Дейкстра **работает**, но только без дуг **отрицательного** веса
- Задача размена монет: пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$
 - Разменять 6 рублей монетами $\{1, 2, 5, 10\}$ -> **жадность**
 - Разменять 6 рублей монетами $\{1, 3, 4\}$ -> **не жадность**

Алгоритм Краскала



- 0) Сортируем все ребра в порядке возрастания веса
- 1) Идем по получившемуся массиву ребер и добавляем ребро в результат T , но только, если это не создаст **циклов**!

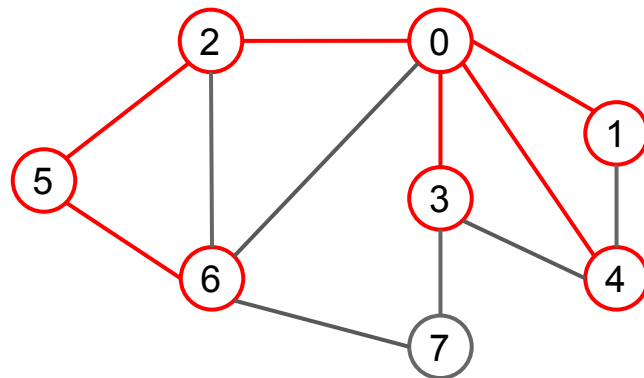
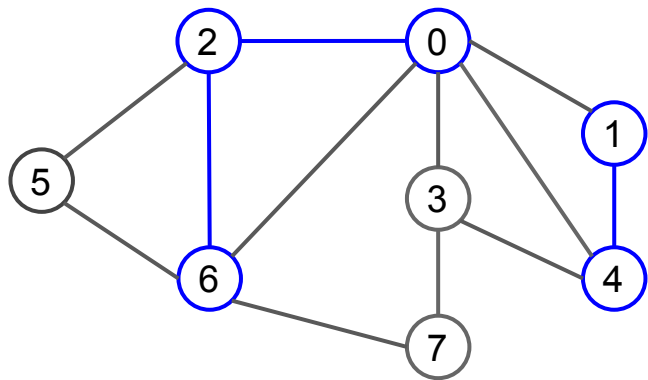
Алгоритм Краскала



- 0) Сортируем все ребра в порядке возрастания веса
- 1) Идем по получившемуся массиву ребер и добавляем ребро в результат T , но только, если это не создаст **циклов**!

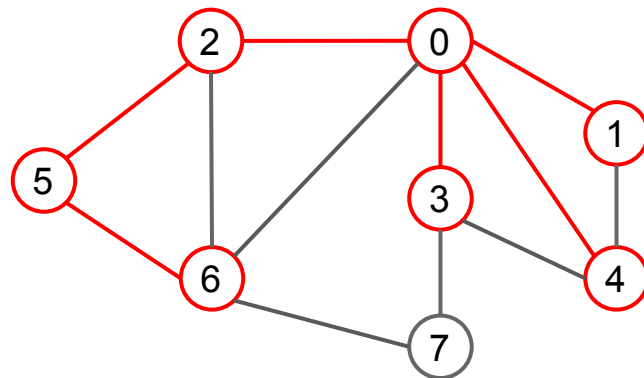
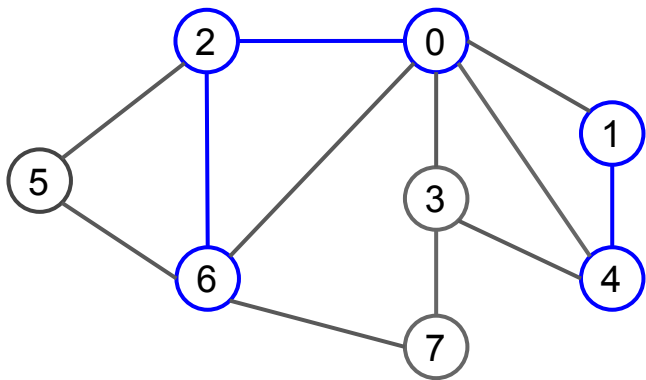
Идея: обобщить Краскала на более широкий класс задач.

Заметим



Пусть есть два **ациклических подграфа** (остовных леса) разного размера.

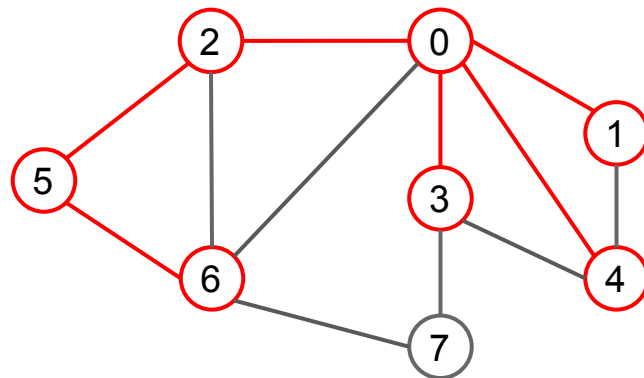
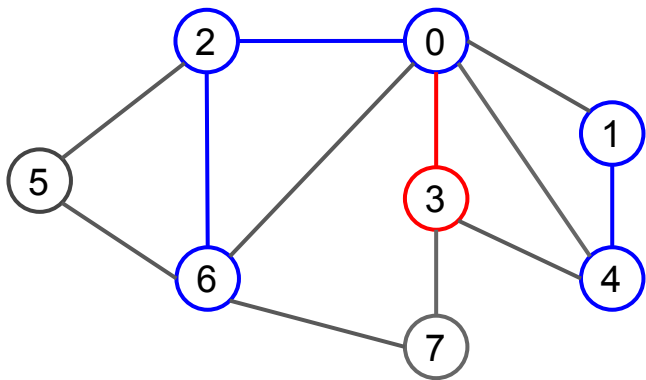
Заметим



Пусть есть два **ациклических подграфа** (остовных леса) разного размера.

Тогда в большем **всегда** можно выбрать ребро, которое добавить в меньший, чтобы тот остался **ациклическим подграфом**.

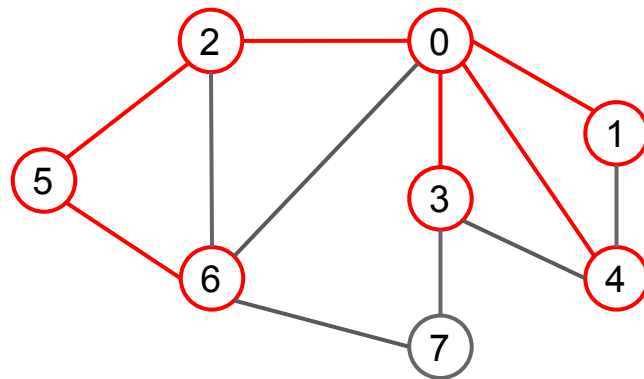
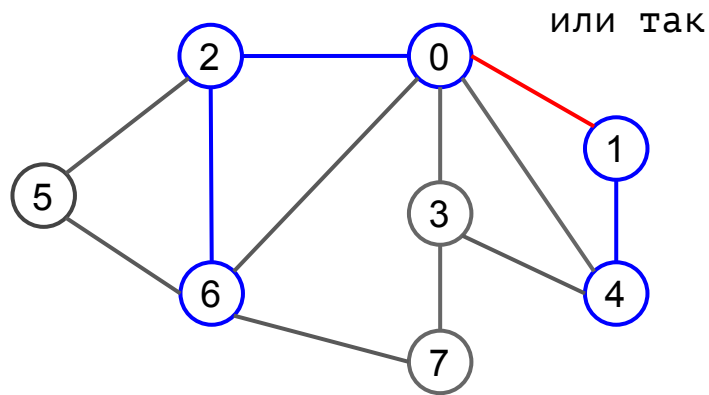
Заметим



Пусть есть два **ациклических подграфа** (остовных леса) разного размера.

Тогда в большем **всегда** можно выбрать ребро, которое добавить в меньший, чтобы тот остался **ациклическим подграфом**.

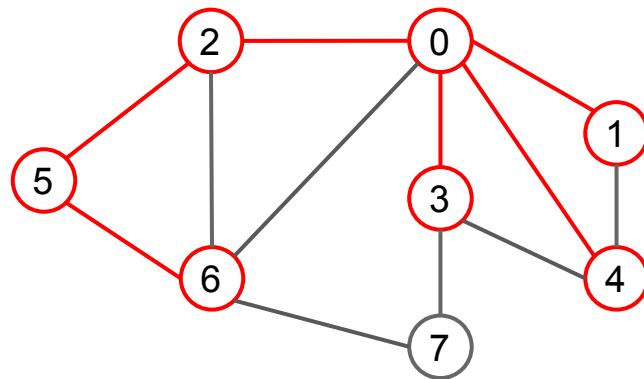
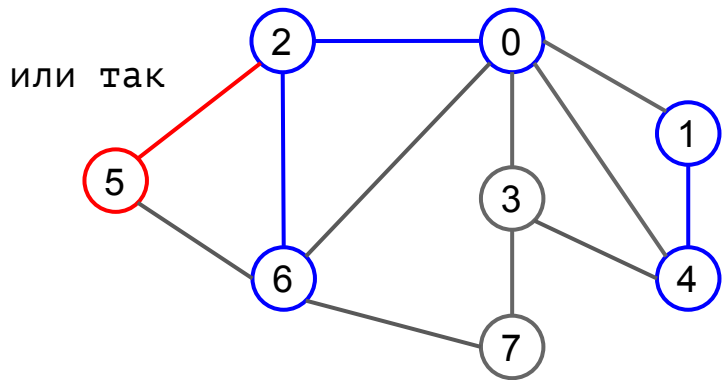
Заметим



Пусть есть два **ациклических подграфа** (остовных леса) разного размера.

Тогда в большем **всегда** можно выбрать ребро, которое добавить в меньший, чтобы тот остался **ациклическим подграфом**.

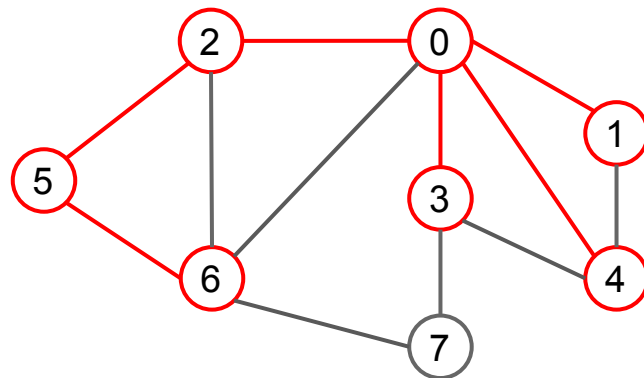
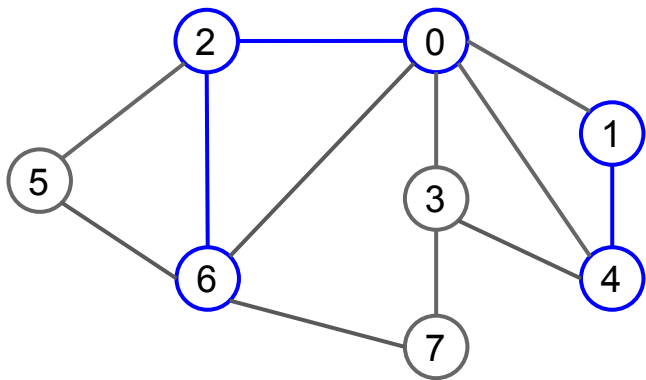
Заметим



Пусть есть два **ациклических подграфа** (остовных леса) разного размера.

Тогда в большем **всегда** можно выбрать ребро, которое добавить в меньший, чтобы тот остался **ациклическим подграфом**.

Заметим

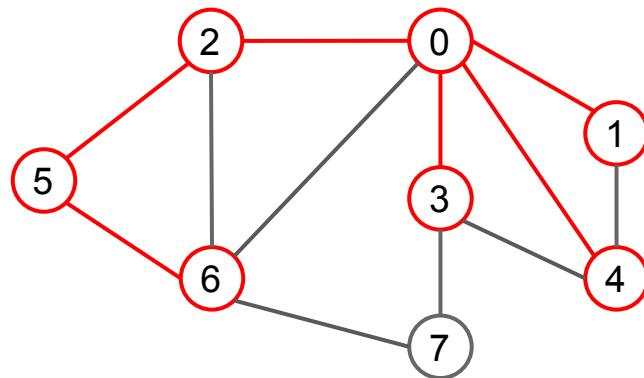
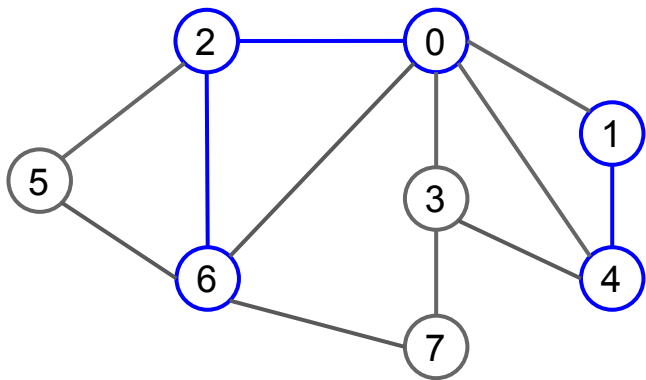


Пусть есть два **ациклических подграфа** (остовных леса) разного размера.

Тогда в большем **всегда** можно выбрать ребро, которое добавить в меньший, чтобы тот остался **ациклическим подграфом**.

Найти можно всегда, т.к. в меньшем подграфе больше компонент связности

Заметим



Пусть есть два **ациклических подграфа** (остовных леса) разного размера.

Тогда в большем **всегда** можно выбрать ребро, которое добавить в меньший, чтобы тот остался **ациклическим подграфом**.

Найти можно всегда, т.к. в меньшем подграфе больше компонент связности => добавлять нужно ребро между этими компонентами

Матроиды

Матроидом называется пара $M = (S, I)$, где S - конечное множество элементов (носитель), а I - система подмножеств S , такая что:

Матроиды

Матроидом называется пара $M = (S, I)$, где S - конечное множество элементов (носитель), а I - система подмножеств S , такая что:

- Пустое множество входит в I [nil]

Матроиды

Матроидом называется пара $M = (S, I)$, где S - конечное множество элементов (носитель), а I - система подмножеств S , такая что:

- Пустое множество входит в I [nil]
- Если множество принадлежит I , то и все его подмножества в I [sub]

Матроиды

Матроидом называется пара $M = (S, I)$, где S - конечное множество элементов (носитель), а I - система подмножеств S , такая что:

- Пустое множество входит в I [nil]
- Если множество принадлежит I , то и все его подмножества в I [sub]
- Если одно из подмножеств из I больше по мощности другого, [aug]
то в нем всегда **найдется** такой элемент, который можно добавить в меньшее, чтобы меньшее с этим элементом осталось в I

Матроиды

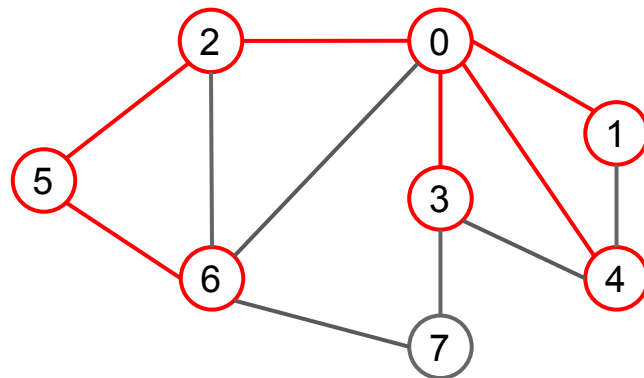
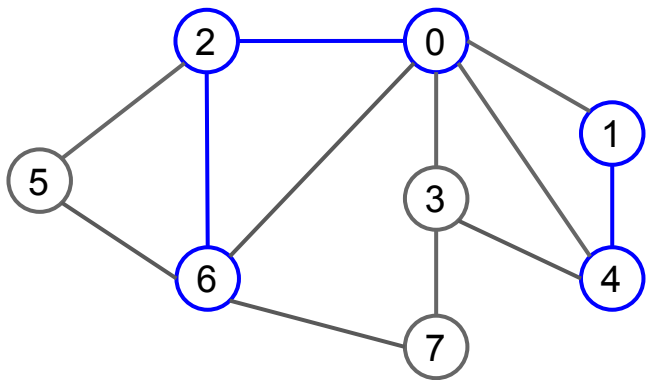
Матроидом называется пара $M = (S, I)$, где S - конечное множество элементов (носитель), а I - система подмножеств S , такая что:

- Пустое множество входит в I [nil]
- Если множество принадлежит I , то и все его подмножества в I [sub]
- Если одно из подмножеств из I больше по мощности другого, [aug]
то в нем всегда **найдется** такой элемент, который можно добавить в меньшее, чтобы меньшее с этим элементом осталось в I

Базой матроида назовем максимальное подмножество из I , т.е. то, к которому нельзя добавить новый элемент из S так, чтобы оно осталось в I .

Матроиды

Где здесь матроид?



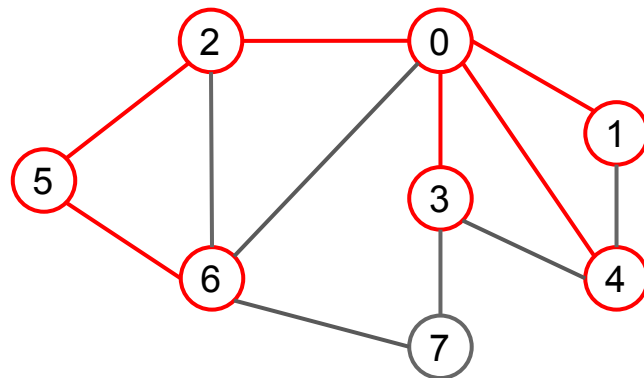
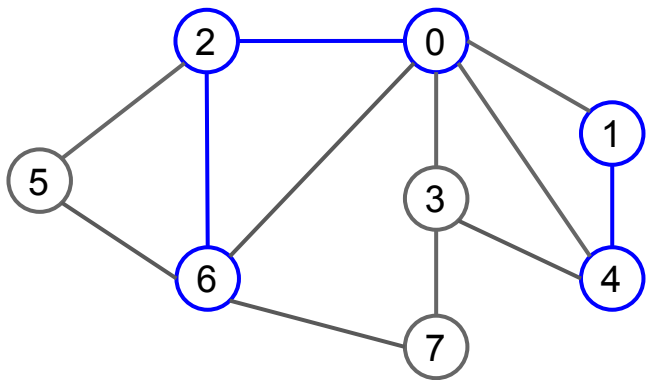
Пусть есть два **ациклических подграфа** (остовных леса) разного размера.

Тогда в большем **всегда** можно выбрать ребро, которое добавить в меньший, чтобы тот остался **ациклическим подграфом**.



Матроиды

Где здесь матроид?



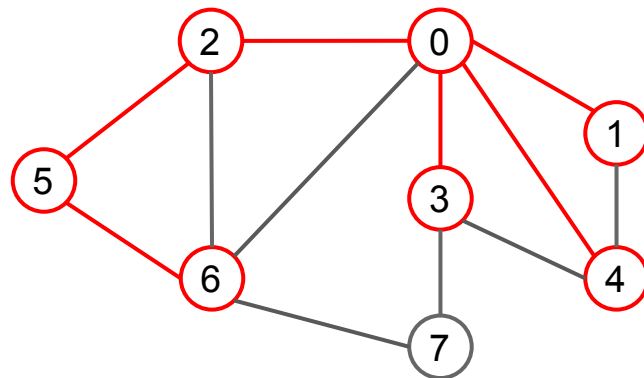
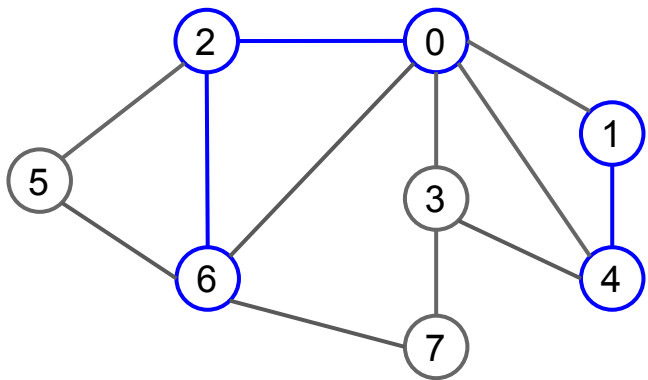
Если наш граф $G = (V, E)$, то $M_G = (S_G, I_G)$ - **графовый матроид**, где:

$S_G = E$ - множество всех ребер графа, а I_G - подмножества ребер, такие что: $U \in I_G \Rightarrow G' = (V, U)$ - это **остовный лес**



Матроиды

Где здесь матроид?



Если наш граф $G = (V, E)$, то $M_G = (S_G, I_G)$ - **графовый матроид**, где:

$S_G = E$ - множество всех ребер графа, а I_G - подмножества ребер, такие что: $U \in I_G \Rightarrow G' = (V, U)$ - это **остовный лес**

А что будет базой?



Матроиды

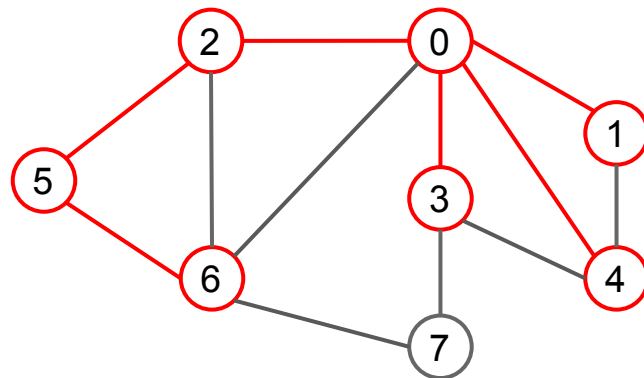
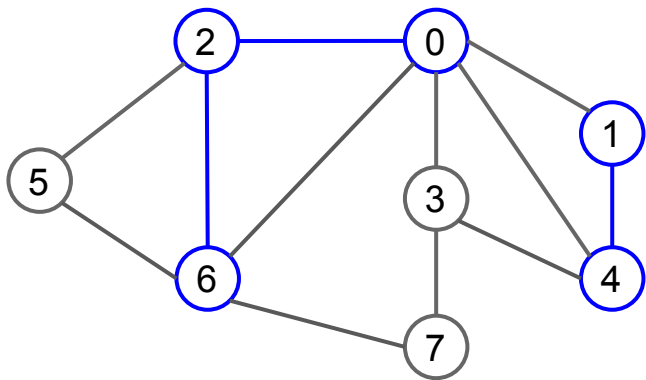
Матроидом называется пара $M = (S, I)$, где S - конечное множество элементов (носитель), а I - система подмножеств S , такая что:

- Пустое множество входит в I [nil]
- Если множество принадлежит I , то и все его подмножества в I [sub]
- Если одно из подмножеств из I больше по мощности другого, [aug]
то в нем всегда **найдется** такой элемент, который можно добавить в меньшее, чтобы меньшее с этим элементом осталось в I

Базой матроида назовем максимальное подмножество из I , т.е. то, к которому нельзя добавить новый элемент из S так, чтобы оно осталось в I .

Матроиды

Где здесь матроид?



Если наш граф $G = (V, E)$, то $M_G = (S_G, I_G)$ - **графовый матроид**, где:

$S_G = E$ - множество всех ребер графа, а I_G - подмножества ребер, такие что: $U \in I_G \Rightarrow G' = (V, U)$ - это **остовный лес**

А что будет базой? **Остовное дерево** графа!



Матроиды

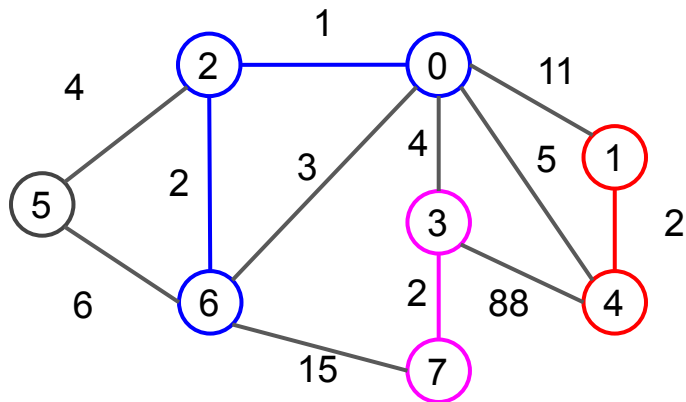
Матроид $M = (S, I)$ назовем взвешенным, если на носителе введена весовая функция w , такая что для $s_i \in S : w(s_i) > 0$.

Вес подмножества $A \subseteq S$ равен сумме весов элементов: $w(A) = \sum_{x \in A} w(x)$

Матроиды

Матроид $M = (S, I)$ назовем взвешенным, если на носителе введена весовая функция w , такая что для $s_i \in S : w(s_i) > 0$.

Вес подмножества $A \subseteq S$ равен сумме весов элементов: $w(A) = \sum_{x \in A} w(x)$



В Крускале имеем взвешенный графовый матроид. Весовую функцию возьмем $w'(e) = w_0 - w(e)$, где w_0 - значение, превышающее вес любого ребра

Матроиды

Матроид $M = (S, I)$ назовем взвешенным, если на носителе введена весовая функция w , такая что для $s_i \in S : w(s_i) > 0$.

Вес подмножества $A \subseteq S$ равен сумме весов элементов: $w(A) = \sum_{x \in A} w(x)$

Алгоритм **Радó-Эдмондса** (поиска базы максимального веса в матроиде):

Матроиды

Матроид $M = (S, I)$ назовем взвешенным, если на носителе введена весовая функция w , такая что для $s_i \in S : w(s_i) > 0$.

Вес подмножества $A \subseteq S$ равен сумме весов элементов: $w(A) = \sum_{x \in A} w(x)$

Алгоритм **Радó-Эдмондса** (поиска базы максимального веса в матроиде):

Итеративно приближаемся к решению, строя множества $A_i \in I$:

1. $A_0 = \emptyset$ (входит в I по [nil])

Матроиды

Матроид $M = (S, I)$ назовем взвешенным, если на носителе введена весовая функция w , такая что для $s_i \in S : w(s_i) > 0$.

Вес подмножества $A \subseteq S$ равен сумме весов элементов: $w(A) = \sum_{x \in A} w(x)$

Алгоритм **Радо-Эдмондса** (поиска базы максимального веса в матроиде):

Итеративно приближаемся к решению, строя множества $A_i \in I$:

1. $A_0 = \emptyset$ (входит в I по [nil])
2. $A_i = A_{i-1} + \{x\}$, где $x = \max_{w_j} \{y_j \in S \setminus A_{i-1} | A_{i-1} + \{y_j\} \in I\}$

Матроиды

Матроид $M = (S, I)$ назовем взвешенным, если на носителе введена весовая функция w , такая что для $s_i \in S : w(s_i) > 0$.

Вес подмножества $A \subseteq S$ равен сумме весов элементов: $w(A) = \sum_{x \in A} w(x)$

Алгоритм **Радо-Эдмондса** (поиска базы максимального веса в матроиде):

Итеративно приближаемся к решению, строя множества $A_i \in I$:

1. $A_0 = \emptyset$ (входит в I по [nil])
2. $A_i = A_{i-1} + \{x\}$, где $x = \max_{w_j} \{y_j \in S \setminus A_{i-1} | A_{i-1} + \{y_j\} \in I\}$

Теорема: алгоритм Радо-Эдмондса находит базу максимального веса

Матроиды

Матроид $M = (S, I)$ назовем взвешенным, если на носителе введена весовая функция w , такая что для $s_i \in S : w(s_i) > 0$.

Вес подмножества $A \subseteq S$ равен сумме весов элементов: $w(A) = \sum_{x \in A} w(x)$

Алгоритм **Радо-Эдмондса** (поиска базы максимального веса в матроиде):

Итеративно приближаемся к решению, строя множества $A_i \in I$:

1. $A_0 = \emptyset$ (входит в I по [nil])
2. $A_i = A_{i-1} + \{x\}$, где $x = \max_{w_j} \{y_j \in S \setminus A_{i-1} | A_{i-1} + \{y_j\} \in I\}$

Теорема: алгоритм Радо-Эдмондса находит базу максимального веса

Еще одна задача планирования

Снова есть один исполнитель и множество задач. Но теперь можно выполнять ровно **одну** задачу в день.

У каждой задачи j есть дедлайн d_j и штраф за его просрочку w_j

Построить расписание задач, **минимизирующее** суммарный штраф.

Еще одна задача планирования

Снова есть один исполнитель и множество задач. Но теперь можно выполнять ровно **одну** задачу в день.

У каждой задачи j есть дедлайн d_j и штраф за его просрочку w_j

Построить расписание задач, **минимизирующее** суммарный штраф.

Пример:

Задача	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Еще одна задача планирования

Предложите жадный алгоритм?
Попробуем просто начать с
самых дорогих задач

Снова есть один исполнитель и множество задач. Но теперь можно выполнять ровно **одну** задачу в день.

У каждой задачи j есть дедлайн d_j и штраф за его просрочку w_j

Построить расписание задач, **минимизирующее** суммарный штраф.

Пример: [D ->]

Задача	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Еще одна задача планирования

Предложите жадный алгоритм?
Попробуем просто начать с
самых дорогих задач

Снова есть один исполнитель и множество задач. Но теперь можно выполнять ровно **одну** задачу в день.

У каждой задачи j есть дедлайн d_j и штраф за его просрочку w_j

Построить расписание задач, **минимизирующее** суммарный штраф.

Пример: [D -> C ->]

Задача	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Еще одна задача планирования

Предложите жадный алгоритм?
Попробуем просто начать с
самых дорогих задач

Снова есть один исполнитель и множество задач. Но теперь можно выполнять ровно **одну** задачу в день.

У каждой задачи j есть дедлайн d_j и штраф за его просрочку w_j

Построить расписание задач, **минимизирующее** суммарный штраф.

Пример: [**D** -> **C** -> **A** ->]

Задача	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Еще одна задача планирования

Предложите жадный алгоритм?
Попробуем просто начать с
самых дорогих задач

Снова есть один исполнитель и множество задач. Но теперь можно выполнять ровно **одну** задачу в день.

У каждой задачи j есть дедлайн d_j и штраф за его просрочку w_j

Построить расписание задач, **минимизирующее** суммарный штраф.

Пример: [D -> C -> A -> E -> B]

Задача	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20



Все очень
плохо

Еще одна задача планирования

Снова есть один исполнитель и множество задач. Но теперь можно выполнять ровно **одну** задачу в день.

У каждой задачи j есть дедлайн d_j и штраф за его просрочку w_j

Построить расписание задач, **минимизирующее** суммарный штраф.

Наивный жадный алгоритм явно не работает, но не стоит расстраиваться!

Ведь здесь несложно заметить матроид.



Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Назовем **выполнимыми** множества работ, которые можно сделать без просрочки дедлайна. Пример: {C, D, B} или {A, E}

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Назовем **выполнимыми** множества работ, которые можно сделать без просрочки дедлайна. Пример: {C, D, B} или {A, E}

Заметим, что для $S = (J, I)$, где J - работы, а I - выполнимые подмножества работ выполняются требования матроида

Матроиды

Матроидом называется пара $M = (S, I)$, где S - конечное множество элементов (носитель), а I - система подмножеств S , такая что:

- Пустое множество входит в I [nil]
- Если множество принадлежит I , то и все его подмножества в I [sub]
- Если одно из подмножеств из I больше по мощности другого, то в нем всегда **найдется** такой элемент, который можно добавить в меньшее, чтобы меньшее с этим элементом осталось в I [aug]

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Назовем **выполнимыми** множества работ, которые можно сделать без просрочки дедлайна. Пример: {C, D, B} или {A, E}

Заметим, что для $S = (J, I)$, где J - работы, а I - выполнимые подмножества работ выполняются требования матроида

[nil] и [sub] - очевидно

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Назовем **выполнимыми** множества работ, которые можно сделать без просрочки дедлайна. Пример: {C, D, B} или {A, E}

Заметим, что для $S = (J, I)$, где J - работы, а I - выполнимые подмножества работ выполняются требования матроида

[nil] и [sub] - очевидно, [aug] - берем задачу, заканчивающуюся в день, который мы не использовали в меньшем

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Назовем **выполнимыми** множества работ, которые можно сделать без просрочки дедлайна. Пример: {C, D, B} или {A, E}

Заметим, что для $S = (J, I)$, где J - работы, а I - выполнимые подмножества работ выполняются требования матроида

[nil] и [sub] - очевидно, [aug] - берем задачу, заканчивающуюся в день, который мы не использовали в меньшем $\{1, 3, 5\}; \{5, 9\} \rightarrow \{1, 5, 9\}$

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Назовем **выполнимыми** множества работ, которые можно сделать без просрочки дедлайна. Пример: {C, D, B} или {A, E}

Заметим, что для $S = (J, I)$, где J - работы, а I - выполнимые подмножества работ выполняются требования матроида

А значит применим алгоритм **Радо-Эдмондса**!

Матроиды

Матроид $M = (S, I)$ назовем взвешенным, если на носителе введена весовая функция w , такая что для $s_i \in S : w(s_i) > 0$.

Вес подмножества $A \subseteq S$ равен сумме весов элементов: $w(A) = \sum_{x \in A} w(x)$

Алгоритм **Радо-Эдмондса** (поиска базы максимального веса в матроиде):

Итеративно приближаемся к решению, строя множества $A_i \in I$:

1. $A_0 = \emptyset$ (входит в I по [nil])

2. $A_i = A_{i-1} + \{x\}$, где $x = \max_{w_j} \{y_j \in S \setminus A_{i-1} | A_{i-1} + \{y_j\} \in I\}$

Теорема: алгоритм Радо-Эдмондса находит базу максимального веса

Еще одна задача планирования

Применяем алгоритм **Радон-Эдмондса**:

Наша задача - построить максимальное (и самое дорогое) выполнимое множество задач. Их будем успевать в срок, а остальные задачи - как-нибудь, с просрочками.

Еще одна задача планирования

Применяем алгоритм **Радон-Эдмондса**:

Наша задача - построить максимальное (и самое дорогое) выполнимое множество задач. Их будем успевать в срок, а остальные задачи - как-нибудь, с просрочками.

На каждом шаге алгоритма выбираем задачу:

1. С самым **большим** штрафом,
2. Так, чтобы получившееся множество задач оставалось **выполнимым**

Это даст нам оптимальное решение.

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

На первом шаге берем самую дорогую работу (т.к. множество из одной работы всегда будет выполнимо)

{D ->

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

На первом шаге берем самую дорогую работу (т.к. множество из одной работы всегда будет выполнимо)

{D -> C ->

Дальше: берем самое дорогое из оставшихся и совместных (про место в финальном расписании пока не думаем)

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

На первом шаге берем самую дорогую работу (т.к. множество из одной работы всегда будет выполнимо)

{D -> C -> A ->

Дальше: берем самое дорогое из оставшихся и совместных (про место в финальном расписании **пока не думаем**)

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

На первом шаге берем самую дорогую работу (т.к. множество из одной работы всегда будет выполнимо)

{D -> C -> A ->

Дальше: берем самое дорогое из оставшихся и совместных (про место в финальном расписании **пока не думаем**). Не можем теперь взять E, т.к. множество перестанет быть выполнимым!

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

На первом шаге берем самую дорогую работу (т.к. множество из одной работы всегда будет выполнимо)

{D -> C -> A -> B}

Дальше: берем самое дорогое из оставшихся и совместных (про место в финальном расписании **пока не думаем**). Не можем теперь взять E, т.к. множество перестанет быть выполнимым!

Еще одна задача планирования

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

На первом шаге берем самую дорогую работу (т.к. множество из одной работы всегда будет выполнимо)

{D -> C -> A -> B} => Ответ: C -> A -> D -> B -> E

Дальше: берем самое дорогое из оставшихся и совместных (про место в финальном расписании **пока не думаем**). Не можем теперь взять E, т.к. множество перестанет быть выполнимым! Собираем ответ.

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

		D		
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо.

C		D		
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо.

C		D		
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо.

C	A	D		
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо. Если не получилось ровно на день дедлайна поставить => ищем **левее** свободное место.

C	A	D		
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо. Если не получилось ровно на день дедлайна поставить => ищем **левее** свободное место. А если и левее места нет,

C	A	D		E
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо. Если не получилось ровно на день дедлайна поставить => ищем **левее** свободное место. А если и левее места нет, то ставим ее в самое **правое** свободное место.

Это финальный ответ



C	A	D	B	E
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо. Если не получилось ровно на день дедлайна поставить => ищем **левее** свободное место. А если и левее места нет, то ставим ее в самое **правое** свободное место.

Это финальный ответ



C	A	D	B	E
1	2	3	4	5

Практическая реализация


Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо. Если не получилось ровно на день дедлайна поставить => ищем **левее** свободное место. А если и левее места нет, то ставим ее в самое **правое** свободное место.

Упражнение: доказать, что корректно отправлять работы с просрочкой в конец массива.

Это финальный ответ



C	A	D	B	E
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо. Если не получилось ровно на день дедлайна поставить => ищем **левее** свободное место. А если и левее места нет, то ставим ее в самое **правое** свободное место.

А что по поводу сложности?

Это финальный ответ



C	A	D	B	E
1	2	3	4	5

Практическая реализация

Номер задачи	A	B	C	D	E
Дедлайн	3	4	1	3	3
Штраф	25	10	30	50	20

Выбираем самую дорогую из еще не выбранных задач. Пытаемся поставить ее исполнение в день ее **дедлайна**.

Если получилось - хорошо. Если не получилось ровно на день дедлайна поставить => ищем **левее** свободное место. А если и левее места нет, то ставим ее в самое **правое** свободное место.

А что по поводу сложности? $O(N^2)$ (для каждой работы можем бежать влево, чтобы найти место)

Мини-задача #36 (2 балла)

Реализовать алгоритм решения описанной в лекции задачи планирования с использованием структуры union-find.

Достигнуть при этом сложности $O(N * \alpha(N))$.

Проверить решение на собственном наборе тестов, который бы демонстрировал некорректность наивного жадного алгоритма.



Задача про монеты

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

- Разменять 6 рублей монетами $\{1, 2, 5, 10\}$ -> жадность
- Разменять 6 рублей монетами $\{1, 3, 4\}$?
Жадное решение даст $4 \rightarrow 1 \rightarrow 1$. Но оптимальное: $3 \rightarrow 3$

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

2) выписываем все частичные размены не более чем по N монет

3) если получился матроид, то жадное решение находит оптимальное решение

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 2, 4\} \rightarrow \{4, 2\}$

2) выписываем все частичные размены не более чем по N монет

$\{4, 2\}, \{4, 1\}, \{2, 2\}, \{1, 2\}, \{1, 1\}, \{4\}, \{2\}, \{1\}$

3) если получился матроид, то жадное решение находит оптимальное решение

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 2, 4\} \rightarrow \{4, 2\}$

2) выписываем все частичные размены не более чем по N монет

$\{4, 2\}, \{4, 1\}, \{2, 2\}, \{1, 2\}, \{1, 1\}, \{4\}, \{2\}, \{1\}$ ←

3) если получился матроид, то жадное решение находит оптимальное решение

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 2, 4\} \rightarrow \{4, 2\}$

2) выписываем все частичные размены не более чем по N монет

$\{4, 2\}, \{4, 1\}, \{2, 2\}, \{1, 2\}, \{1, 1\}, \{4\}, \{2\}, \{1\}$ ←

3) если получился матроид, то жадное решение находит оптимальное решение (просто по Радо-Эдмондсу!)

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 3, 4\}$

2) выписываем все частичные размены не более чем по N монет

3) если получился матроид, то жадное решение находит оптимальное решение. А если нет?

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 3, 4\} \rightarrow \{4, 1, 1\}$

2) выписываем все частичные размены не более чем по N монет

3) если получился матроид, то жадное решение находит оптимальное решение. А если нет?

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 3, 4\} \rightarrow \{4, 1, 1\}$

2) выписываем все частичные размены не более чем по N монет

$\{4, 1, 1\}, \{3, 1, 1\}, \{1, 1, 1\}, \{4, 1\}, \{3, 1\}, \{3, 3\}, \{1, 1\}, \{4\}, \{3\}, \{1\}$

3) если получился матроид, то жадное решение находит оптимальное решение. А если нет?

На паре $\{4,1,1\}$ и $\{3,3\}$
ломается условие [aug],
так что это **не матроид**

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 3, 4\} \rightarrow \{4, 1, 1\}$

2) выписываем все частичные размены не более чем по N монет

$\{4,1,1\}, \{3,1,1\}, \{1,1,1\}, \{4,1\}, \{3,1\}, \{3,3\}, \{1,1\}, \{4\}, \{3\}, \{1\}$

3) если получился матроид, то жадное решение находит оптимальное решение. А если нет?

На паре $\{4,1,1\}$ и $\{3,3\}$
ломается условие [aug],
так что это **не матроид**

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 3, 4\} \rightarrow \{4, 1, 1\}$

2) выписываем все частичные размены не более чем по N монет

$\{4, 1, 1\}, \{3, 1, 1\}, \{1, 1, 1\}, \{4, 1\}, \{3, 1\}, \{3, 3\}, \{1, 1\}, \{4\}, \{3\}, \{1\}$

3) если получился матроид, то жадное решение находит оптимальное решение. А если нет? Здесь жадность не работает.

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Разменять 6 через $\{1, 2, 5, 10\} \rightarrow \{5, 1\}$

2) выписываем все частичные размены не более чем по N монет

$\{5, 1\}, \{2, 2\}, \{1, 2\}, \{1, 1\}, \{5\}, \{2\}, \{1\}$

3) если получился матроид, то жадное решение находит оптимальное решение. А если нет?

Задача про монеты: матроиды

Пусть есть сумма X , представить ее минимальным набором доступных монет $x_1, x_2, x_3, \dots, x_n$

1) запускаем **жадный** алгоритм, смотрим на получившееся решение: пусть в нем N монет

Но жадность здесь **вполне работает!**

Разменять 6 через $\{1, 2, 5, 10\} \rightarrow \{5, 1\}$

2) выписываем все частичные размены не более чем по N монет

$\{5, 1\}, \{2, 2\}, \{1, 2\}, \{1, 1\}, \{5\}, \{2\}, \{1\}$

3) если получился матроид, то жадное решение находит оптимальное решение. А если нет, то ничего сказать нельзя.

Задача про монеты: матроиды

Понятно, что такой способ проверки не практичен - придется искать множество частичных разменов, наивное решение дает **экспоненциальную** сложность.



Задача про монеты: матроиды

Понятно, что такой способ проверки не практичен - придется искать множество частичных разменов, наивное решение дает **экспоненциальную** сложность.

Ценность матроидов здесь в понимании структуры задачи.



Задача про монеты: матроиды

Понятно, что такой способ проверки не практичен - придется искать множество частичных разменов, наивное решение дает **экспоненциальную** сложность.

Ценность матроидов здесь в понимании структуры задачи.

Для практики **эффективный алгоритм** проверки всего-то за $O(N^3)$.



Выводы по матроидам

1. Формулировка через матроиды - **достаточное**, но не необходимое условие оптимальности жадного алгоритма на нем

(т.е. есть жадные алгоритмы, которые работают не на матроиде, но при этом оптимальны)

Выводы по матроидам

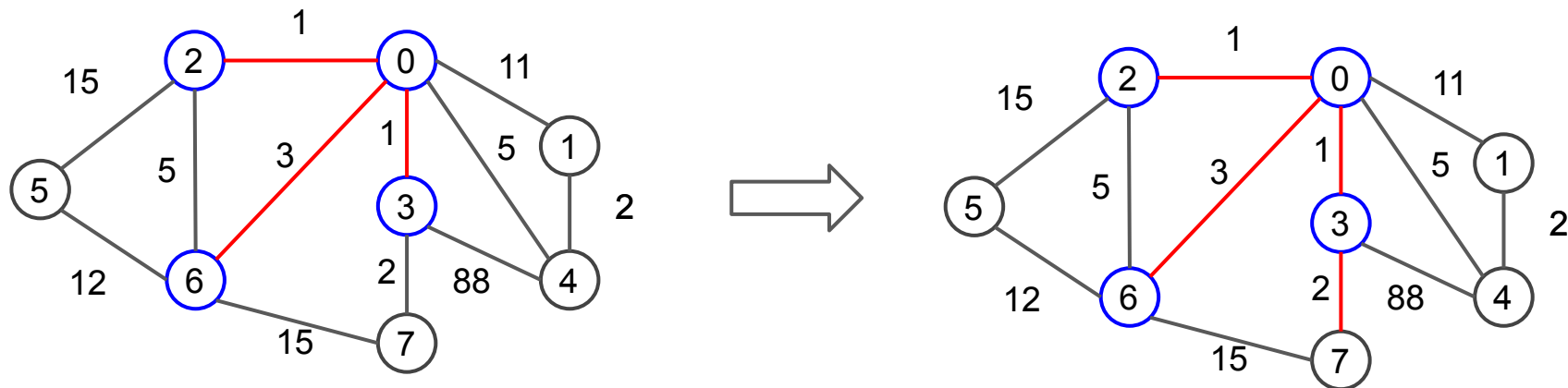
1. Формулировка через матроиды - **достаточное**, но не необходимое условие оптимальности жадного алгоритма на нем

(т.е. есть жадные алгоритмы, которые работают не на матроиде, но при этом оптимальны)
2. Решение через матроиды не всегда самое быстрое, зачастую оценки плохие, а алгоритмы требуют **доработки**

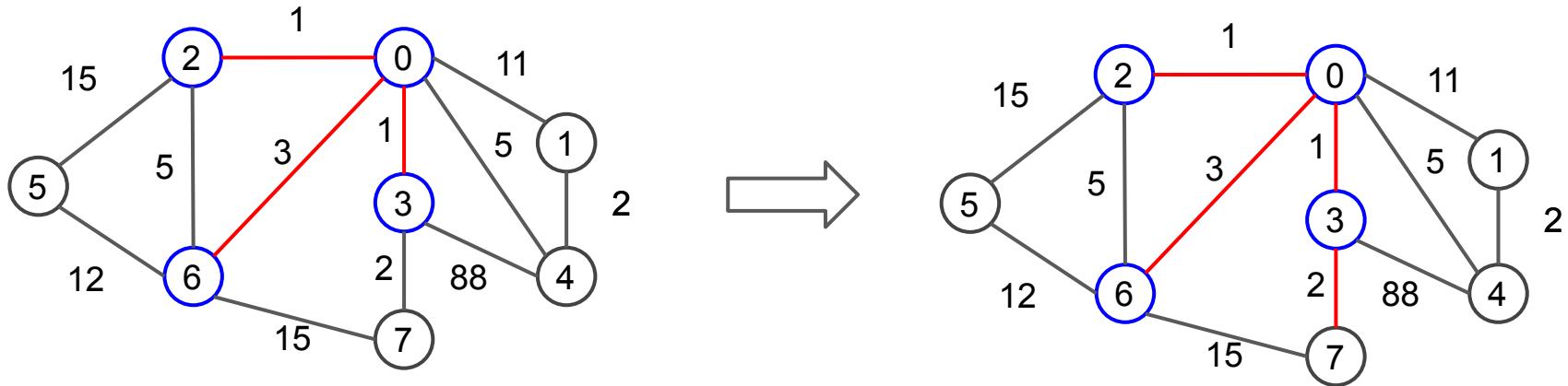
Выводы по матроидам

1. Формулировка через матроиды - **достаточное**, но не необходимое условие оптимальности жадного алгоритма на нем

(т.е. есть жадные алгоритмы, которые работают не на матроиде, но при этом оптимальны)
2. Решение через матроиды не всегда самое быстрое, зачастую оценки плохие, а алгоритмы требуют **доработки**
3. Но формулировка через матроиды - это хороший старт: вы точно получите **оптимальное** решение, это может привести вас на мысли о других алгоритмах



Можно ли сказать, что алгоритм Прима работает на матроиде?



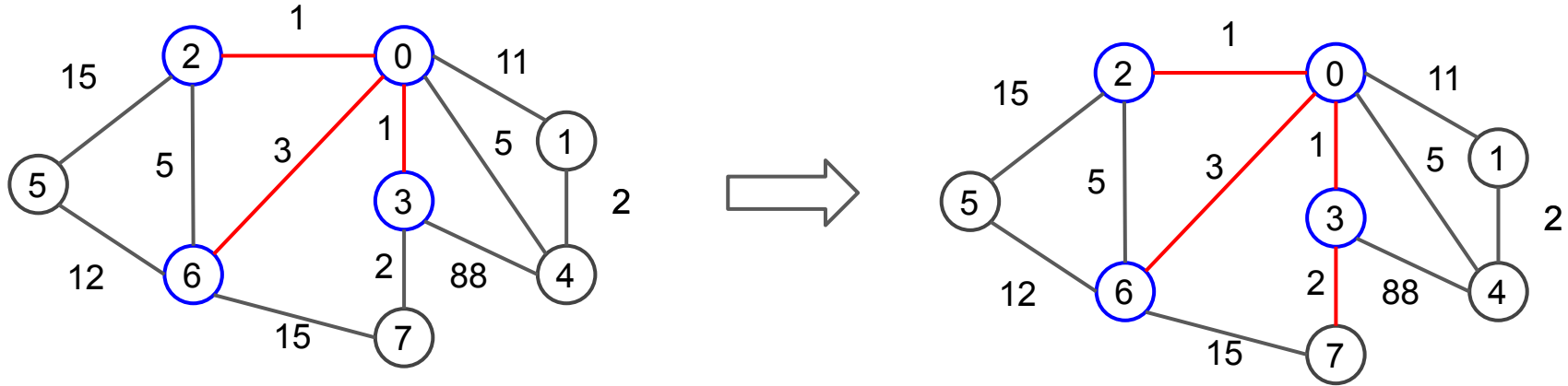
Можно ли сказать, что алгоритм Прима работает на матроиде?
На каждом шаге алгоритма имеем **поддерево** (не лес!)

Матроиды

Матроидом называется пара $M = (S, I)$, где S - конечное множество элементов (носитель), а I - система подмножеств S , такая что:

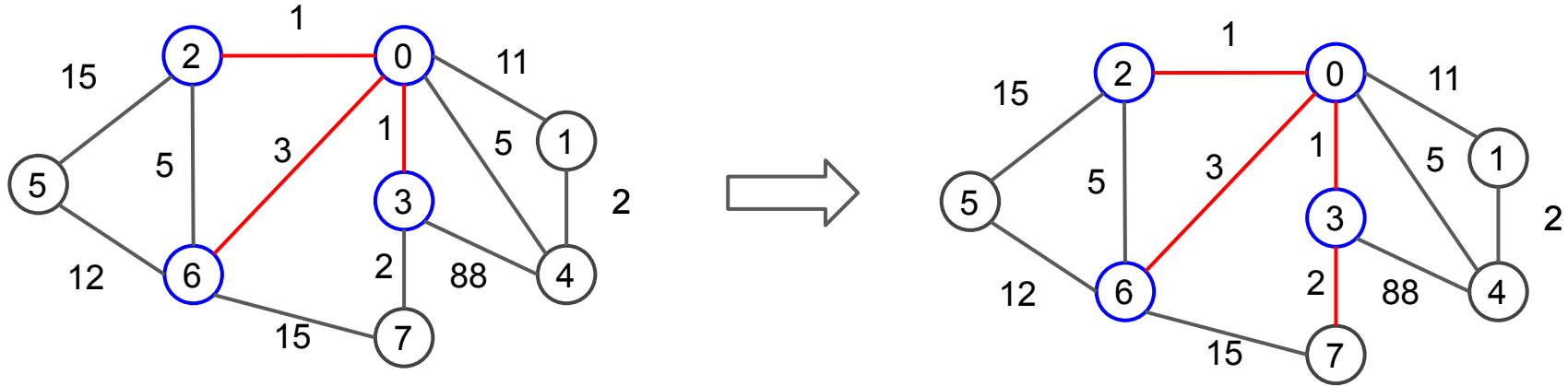
- Пустое множество входит в I [nil]
- Если множество принадлежит I , то и все его подмножества в I [sub]
- Если одно из подмножеств из I больше по мощности другого, [aug]
то в нем всегда **найдется** такой элемент, который можно добавить в меньшее, чтобы меньшее с этим элементом осталось в I

Базой матроида назовем максимальное подмножество из I , т.е. то, к которому нельзя добавить новый элемент из S так, чтобы оно осталось в I .



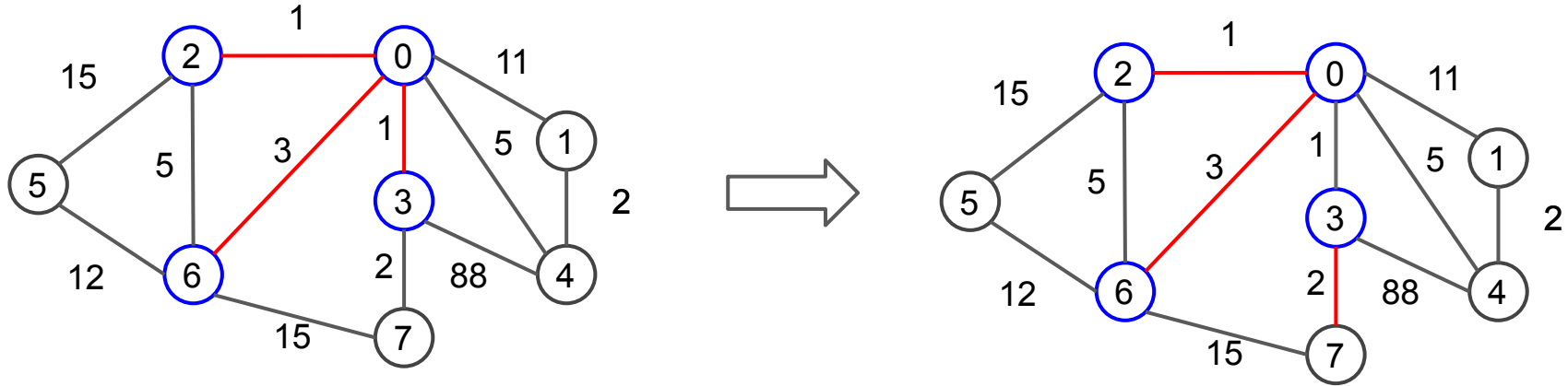
Можно ли сказать, что алгоритм Прима работает на матроиде?
На каждом шаге алгоритма имеем **поддерево** (не лес!)

А верно ли, что множество всех поддеревьев даст нам матроид?



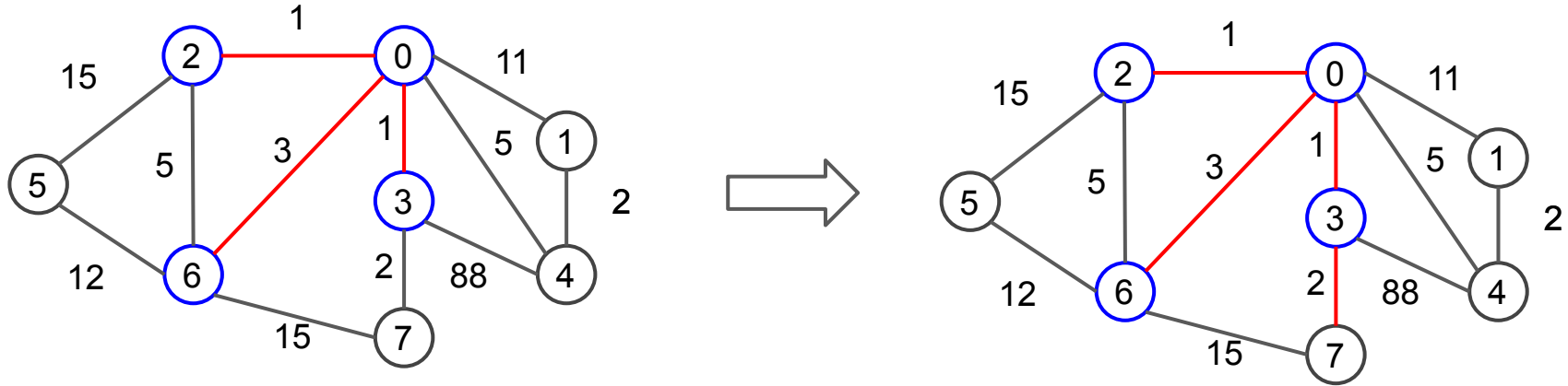
Можно ли сказать, что алгоритм Прима работает на матроиде?
 На каждом шаге алгоритма имеем **поддерево** (не лес!)

А верно ли, что множество всех поддеревьев даст нам матроид?
 Будет ли выполняться $[sub]$ - что любое подмножество тоже
 входит в I ?



Можно ли сказать, что алгоритм Прима работает на матроиде?
 На каждом шаге алгоритма имеем **поддерево** (не лес!)

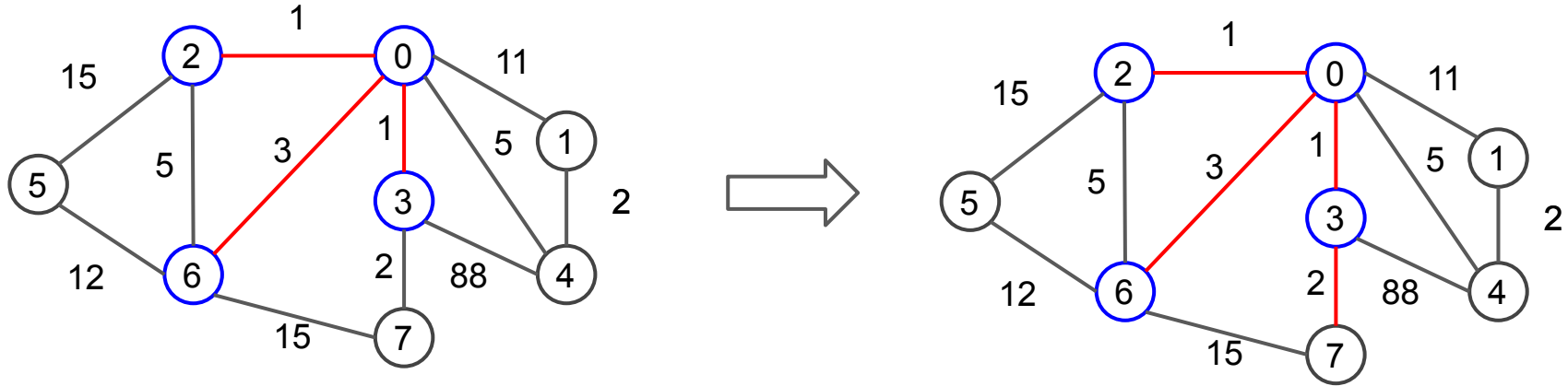
А верно ли, что множество всех поддеревьев даст нам матроид?
 Будет ли выполняться [sub] - что любое подмножество тоже
 входит в I ? Нет! Из-за удаления ребра дерево может перестать
 быть деревом. Поэтому Прима работает не на **матроиде** :(



Можно ли сказать, что алгоритм Прима работает на матроиде?
На каждом шаге алгоритма имеем **поддерево** (не лес!)

А верно ли, что множество всех поддеревьев даст нам матроид?

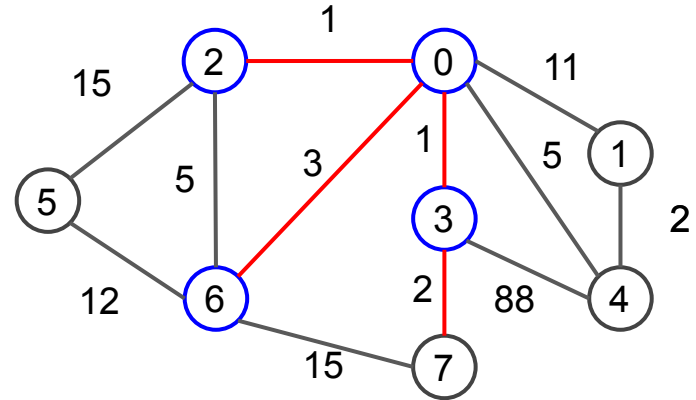
На самом деле Прима работает не на матроиде, а на **гридоиде**.



Можно ли сказать, что алгоритм Прима работает на матроиде?
На каждом шаге алгоритма имеем **поддерево** (не лес!)

А верно ли, что множество всех поддеревьев даст нам матроид?

На самом деле Прима работает не на матроиде, а на **гридоиде**.
При некоторых дополнительных условиях на таких структурах
продолжает работать алгоритм **Радос-Эдмондса**.



Можно ли сказать, что алгоритм Прима работает на матроиде?
На каждом шаге алгоритма имеем **поддерево** (не лес!)

А верно ли, что множество всех поддеревьев даст нам матроид?

На самом деле Прима работает не на матроиде, а на **гридоиде**.
При некоторых дополнительных условиях на таких структурах
продолжает работать алгоритм **Радос-Эдмондса**.

Что еще посмотреть и почитать

Лекция Константина Владимирова (МФТИ) про жадные алгоритмы и матроиды: <https://www.youtube.com/watch?v=roymyfCKgHg>

Что еще посмотреть и почитать

Лекция Константина Владимирова (МФТИ) про жадные алгоритмы и матроиды: <https://www.youtube.com/watch?v=roymyfCKgHg>

Доказательство алгоритма Радон-Эдмондса можно почитать в Кормене, глава 16.4

Что еще посмотреть и почитать

Лекция Константина Владимирова (МФТИ) про жадные алгоритмы и матроиды: <https://www.youtube.com/watch?v=roymyfCKgHg>

Доказательство алгоритма Радон-Эдмондса можно почитать в Кормене, глава 16.4

Про гридоиды почитать здесь:

<https://www.mi.fu-berlin.de/math/groups/discgeom/ziegler/P reprintfiles/006PREPRINT.pdf?1397057423>

Takeaways

- Задача кластеризации и ее решение через Краскала
- Жадные алгоритмы иногда можно обобщить на **матроидах** (и **гридоидах**)
- На решении на матроиде не стоит останавливаться, возможно есть алгоритм и лучше (но это хороший старт)