ЖАДНЫЕ АЛГОРИТМЫ

Идеи и применимость жадных алгоритмов. Алгоритм Радо-Эдмондса

Распределяем работу

- Каждый заказ занимает один день и имеет дедлайн и стоимость
- Одновременно можно делать только один

JobId	**A**	**B**	**C**	**D**	**E**
Deadline	2	1	2	1	3
Profit	40	25	30	15	20

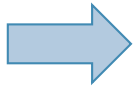
- Задача: распределить заказы так, чтобы максимизировать прибыль
- Например в приведённом примере разумно заниматься в первый день заказом **A**, во второй **C** и в третий день заказом **E**

Problem JA: распределение работы

- Формально (для контеста) у нас есть n заказов и d дней

```
struct order_t { int number; int cost; int deadline; };
struct answer_t { int norders; int *numbers; };
```

JobId	A	B	C	D	E
Deadline	2	1	2	1	3
Profit	40	25	30	15	20



C	A	E

- Вы должны выбрать наилучшее по суммарной стоимости число заказов

```
struct answer_t
betforjobs(struct order_t * orders, int n, int d);
```

Обсуждение

- Попробуйте найти наилучшую стратегию
- Попробуйте также понять как вы рассуждаете в поисках наилучшей стратегии

Жадные алгоритмы

- Распределяя работы, вы, вероятно, прибегали к следующему жадному алгоритму:
  1. Выбираем наибольший из имеющихся заказов и ставим его в очередь
  2. Если заказы ещё остались, переходим к шагу 1.
- Это главные ингредиенты жадного алгоритма: безопасный шаг выбора следующего элемента и подобие в структуре оставшейся подзадачи
- Каждый раз безопасный шаг соответствует локально оптимальному выбору

Размен монет

- Иногда жадный шаг не является оптимальным
- Разменять сумму X монетами с номиналами $x_1, x_2, \ldots, x_n$ выбрав наименьшее число монет
- Пример: разменять 6 рублей монетами с номиналами 4, 3 и 1
- Жадное решение: взять 4, потом 1, потом опять 1
- Оптимальное решение: взять 3 и 3
- В этом случае говорят, что локально-оптимальный шаг не является безопасным шагом и нужно пользоваться другими методами

Обсуждение

- В принципе для размена жадный алгоритм даёт какое-то решение. Не оптимальное, но вообще-то далёкое от наихудшего
- Может ли быть такое, что, для определённого набора монет, жадный алгоритм даст оптимальное решение?

Перевод в двоичную систему

- Перевод в двоичную систему является частным случаем задачи размена монет. Только теперь вы должны оптимальным способом разменять число X «монетами» с номиналами $1, 2, 4, 8, 16, \ldots$
- Например $25 = 16 + 8 + 1 = 8 + 8 + 4 + 4 + 2 + 1$
- Алгоритм для оптимального размена прост: берём в качестве следующего разряда $N \div 2$ и переходим к $N/2$
- `25` $\rightarrow$ `12` $\rightarrow$ `6` $\rightarrow$ `3` $\rightarrow$ `1`
- `1, 0, 0, 1, 1` $\rightarrow$ `11001`
- Является ли это жадным алгоритмом? Какой тут безопасный шаг? Есть ли тут подобие подзадач?

Обсуждение

- Интересно, что жадный алгоритм будет работать оптимально для размена любой суммы монетами с номиналами 10, 5, 2 и 1
- Есть ли у вас гипотеза в чём принципиальная разница между множествами монет для которых жадный алгоритм работает оптимально и не оптимально?
- Не оптимально: {4, 3, 1}, {10, 7, 2, 1}
- Оптимально: {4, 2, 1}, {10, 5, 2, 1}

Обсуждение

- Интересно, что жадный алгоритм будет работать оптимально для размена любой суммы монетами с номиналами 10, 5, 2 и 1
- Есть ли у вас гипотеза в чём принципиальная разница между множествами монет для которых жадный алгоритм работает оптимально и не оптимально?
- Не оптимально: {4, 3, 1}, {10, 7, 2, 1}, {9, 4, 1}, {25, 15, 1}
- Оптимально: {4, 2, 1}, {10, 5, 2, 1}, {9, 5, 1}, {25, 15, 10, 5, 1}
- Добавленные примеры показывают, что эта задача гораздо сложнее, чем кажется. Отложим её пока что

Problem EV: степени четверки

- Напишите функцию, которая оптимально раскладывает число n по ограниченному количеству степеней четверки 4^0, 4^1, и т. д. до 4^m включительно
- Каждая степень считается одной монетой.
  Например `27 = 1*16 + 2*4 + 3*1` использует 6 монет.
  В то же время `27 = 1*16 + 1*4 + 7*1` использует 9 монет

```
unsigned powers_four(unsigned n, unsigned m);
```

- Функция должна возвращать минимальное количество монет

```
unsigned res = powers_of_four(13, 1);
assert(res == 4); // 3*4 + 1*1
```

Problem EG: египетские дроби

- Египетской дробью называется дробь, имеющая в числителе 1
- Каждое число может быть разложено на египетские дроби жадным алгоритмом, который выбирает наибольшую дробь на каждом шаге
- Например: $\frac{39}{50} = \frac{1}{2} + \frac{1}{4} + \frac{1}{34} + \frac{1}{1700}$
- Напишите функцию, которая берёт числитель и знаменатель и возвращает выделенный в динамической памяти массив знаменателей и его размер

```
struct denom_array_t { unsigned *arr; unsigned sz; };

struct denom_array_t
egyptian_fractions(unsigned num, unsigned den);
```

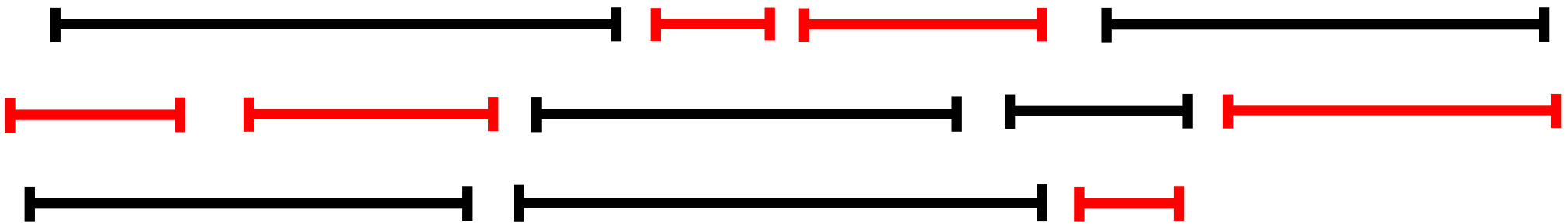
Обсуждение

- В случае египетских дробей, даёт ли жадный алгоритм оптимальное разложение или просто какое-то?
- Иногда это не очевидно
- Иногда, особенно когда вариантов жадных шагов несколько, не очевидно существует ли жадный алгоритм, дающий оптимальное решение
- Но если он существует, обычно он прост и эффективен, поэтому попытаться его найти часто стоит потраченных усилий

Алгоритм поиска алгоритмов

- Переход к следующему пункту только если недостаточно эффективно сработал предыдущий
  1. Попробовать самое наивное решение
  2. Поискать жадный алгоритм
  3. Поискать алгоритм с разбиением на подзадачи (divide & conquer)
  4. Попробовать дискретное динамическое программирование
  5. Начать думать
- Ну и разумеется нулевой шаг: провести обзор литературы и поиск в интернете

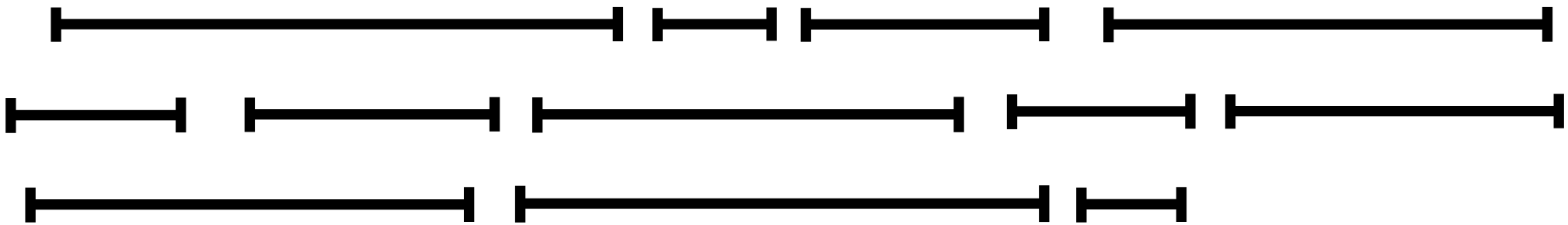
Выбор подмножества интервалов



- Необходимо выбрать максимальное подмножество непересекающихся интервалов на прямой (показано красным)
- Какой жадный шаг вы бы тут попробовали?

Неочевидные жадные шаги

- Структура жадного алгоритма не всегда очевидна



- Какие жадные шаги можно сделать для выбора множества интервалов?
- Выбирать первый слева?
- Выбирать самый короткий?
- Выбирать самый длинный?

Контрпримеры

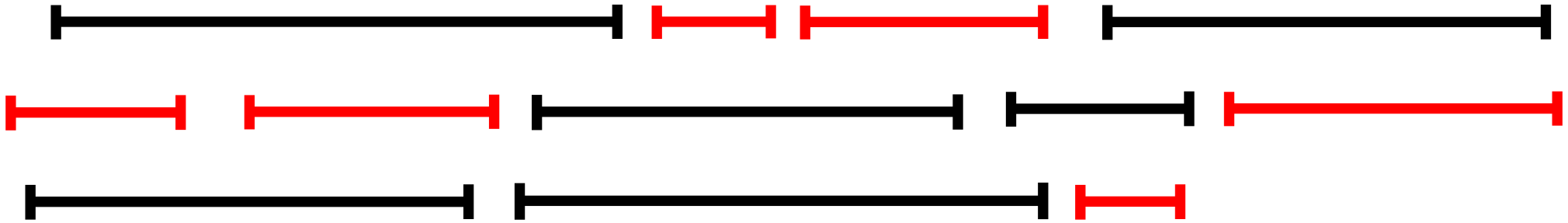
- Первый слева

- Самый короткий

- Самый длинный (см. контрпример первому слева)
- Первый справа (см. контрпример первому слева)
- У вас есть ещё идеи?

Правильный жадный шаг

- Нужно первым брать тот, который первым заканчивается



- Этот шаг является безопасным и локально оптимальным: если мы возьмём иной интервал, то он не пересечёт меньшего числа интервалов
- Получающаяся подазадача после этого шага подобна исходной

Problem IC: расписание аудитории

- Для конкретной аудитории есть ряд заявок с временем начала окончания
- Необходимо вернуть максимальное количество допустимых заявок

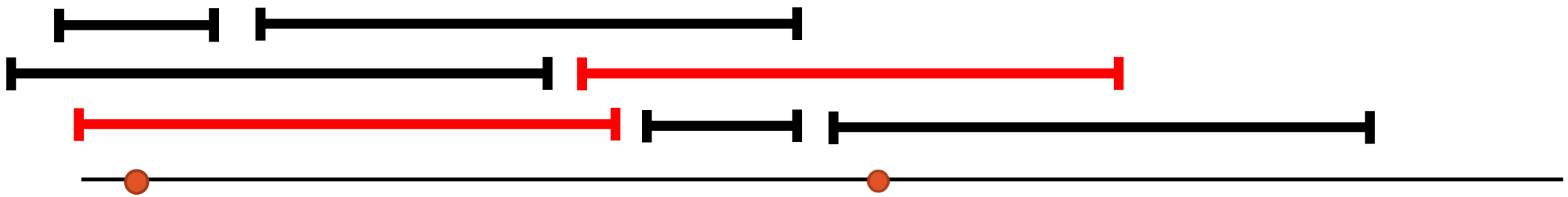
```
struct intvl_t { int number; int start; int fin; };

int schedulemax(struct intvl_t *reqs, int nreqs) {
  // TODO: ваш код здесь
}
```

- Это та самая задача выбора подмножества интервалов, которая рассматривалась выше

Problem RU: ларьки под общую крышу

- На прямой расположены отрезки $[l_i, r_i]$ которые полностью или даже с избытком покрывают интервал $[L, R]$

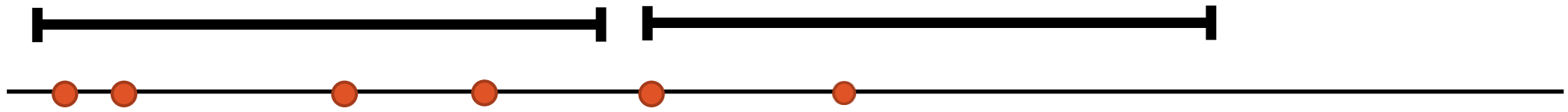


- Ваша задача обратна проблеме IC: теперь вам нужно выбрать наименьшее множество отрезков так, чтобы они всё ещё покрывали интервал

```
int covermin(int L, int R, struct intvl_t *intervs, int ncovs) {
  // TODO: ваш код здесь
}
```

Problem PC: покрытие точек отрезками

- У вас есть n точек с целыми координатами на прямой. Их необходимо покрыть отрезками длины `unitlen`, используя минимальное количество отрезков (ваш запас отрезков в этой задаче неограничен, но все они одинаковые)
- Нужно вернуть количество использованных отрезков



```
int coverpoints(int *pts, int n, int unitlen) {
  // TODO: ваш код здесь
}
```

Problem FP: организация утренника

- Вам нужно организовать детский утренник на которых пришло `nkids` детей. У каждого ребёнка есть номер `id` и возраст `age`
- Вы хотели бы получить минимальное количество групп, но при этом возраст детей в каждой группе не должен отличаться больше, чем на 2 года

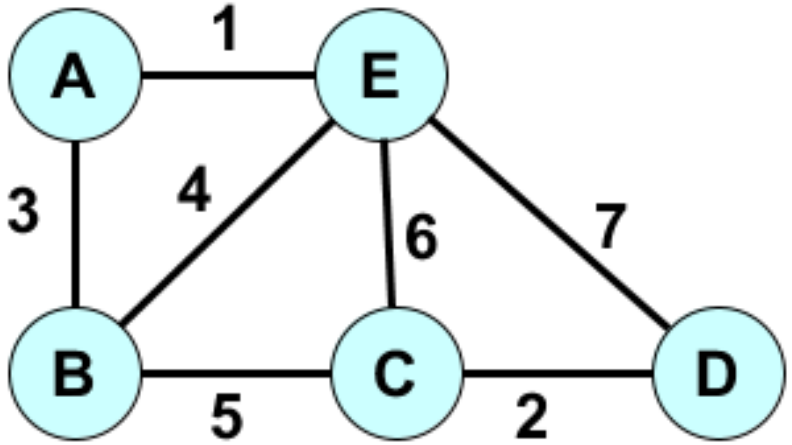
```
struct kid_t { int id; int age; };
struct kidgroup_t { int nkid; int ngroup; };
struct answer_t { int ngroups; struct kidgroup_t *mapping; };

struct answer_t
funparty(struct kid_t *kids, int nkids) {
  // TODO: ваш код здесь
}
```

- Сможете ли вы свести эту задачу к `Problem PC`?

Графы

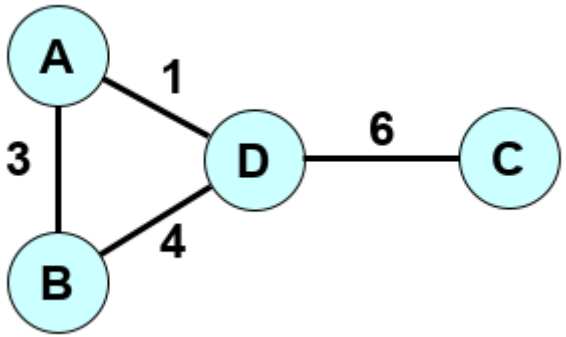
- Графом (V, E) называется множество вершин V, соединённых множеством рёбер E
- Можно думать о вершинах как о городах, а о рёбрах как о дорогах
- Или о вершинах как о числах, а о рёбрах, как о парах чисел
- Или о вершинах как о котиках, а о рёбрах как о признаке совместного появления двух котиков на одной фотке



- У вершин и рёбер могут быть дополнительные признаки, например здесь у каждого ребра показан его вес

Представления графов

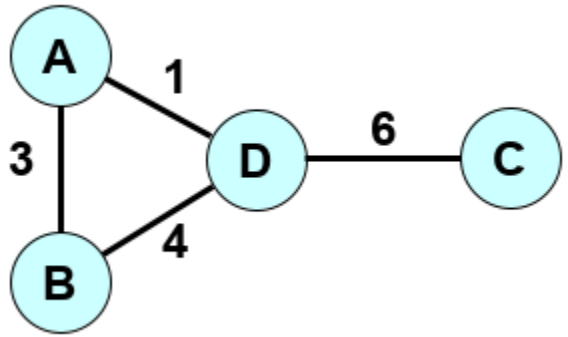
- Самое простое представление графа это просто список его рёбер
- Такое представление хорошо для сброса в человеко-читаемый файл но непрактично для операций над графом



4		
0	1	3
0	3	1
1	3	4
2	3	6

Представления графов

- Граф может быть представлен двумерным массивом который называется матрицей смежности
- Если у рёбер есть веса, то в матрице смежности легко ставить веса вместо единиц

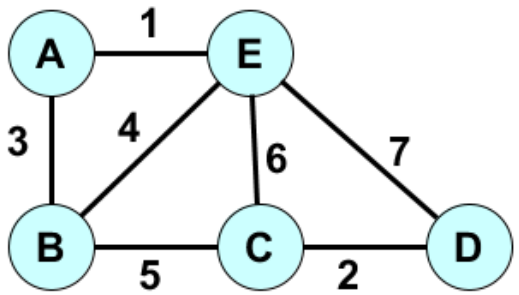


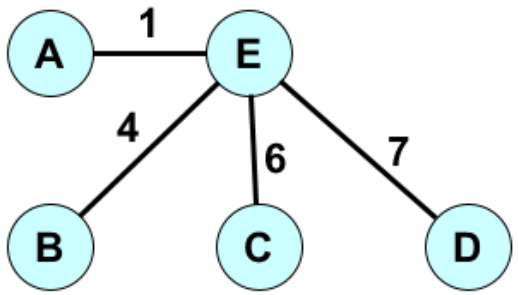
4		
0	1	3
0	3	1
1	3	4
2	3	6

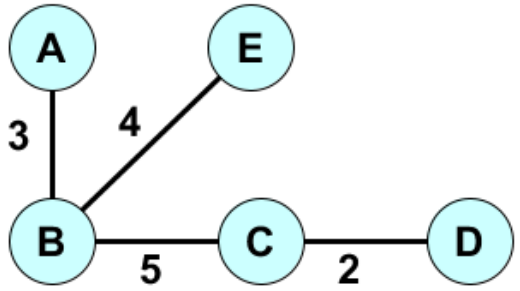
$$\begin{bmatrix} 0 & 3 & 0 & 1 \\ 3 & 0 & 0 & 4 \\ 0 & 0 & 0 & 6 \\ 1 & 4 & 6 & 0 \end{bmatrix}$$

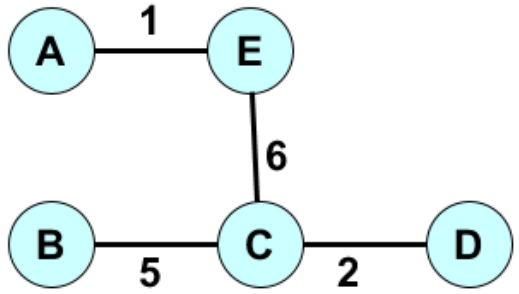
Остовные деревья

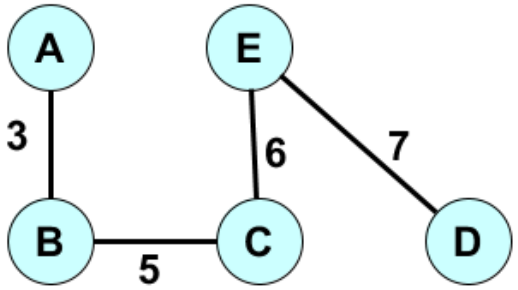
- Деревом в теории графов называется ненаправленный ациклический граф
- Ниже изображён граф со взвешенными рёбрами и некоторые его остовные деревья

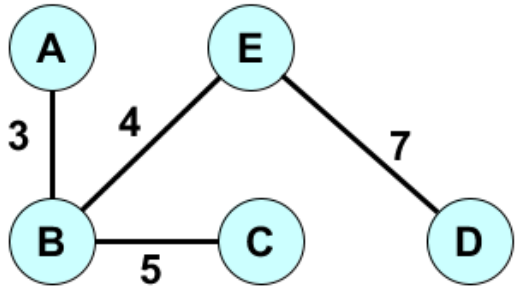












Обсуждение

- Как можно понять, что некий неориентированный граф является деревом?

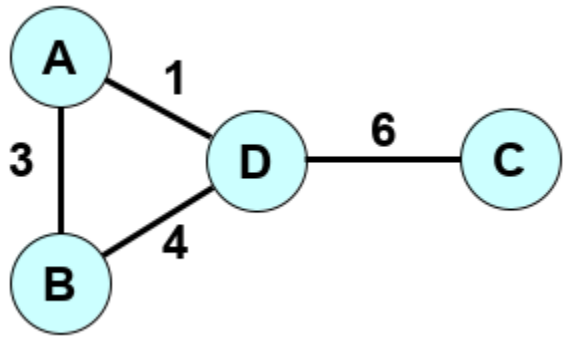
Обсуждение

- Как можно понять, что некий неориентированный граф является деревом?
- Самый простой способ это поиск в ширину, использующий массив родителей
- Изначально для всех вершин устанавливаем $P(v) = -1$
- Берём любую вершину w и для всех её соседей устанавливаем $P(v) = w$
- Рекурсивно вызываем ту же функцию для каждого из соседей отмечая w как текущего родителя
- Как только найдётся сосед v не совпадающий с её текущим родителем и имеющий родителя, не совпадающего с v, цикл найден и это не дерево

Problem GL – петля в графе

- Вам задан неориентированный граф как количество вершин и потом список рёбер с весами

```
4
0 1 3
0 3 1
1 3 4
2 3 6
```



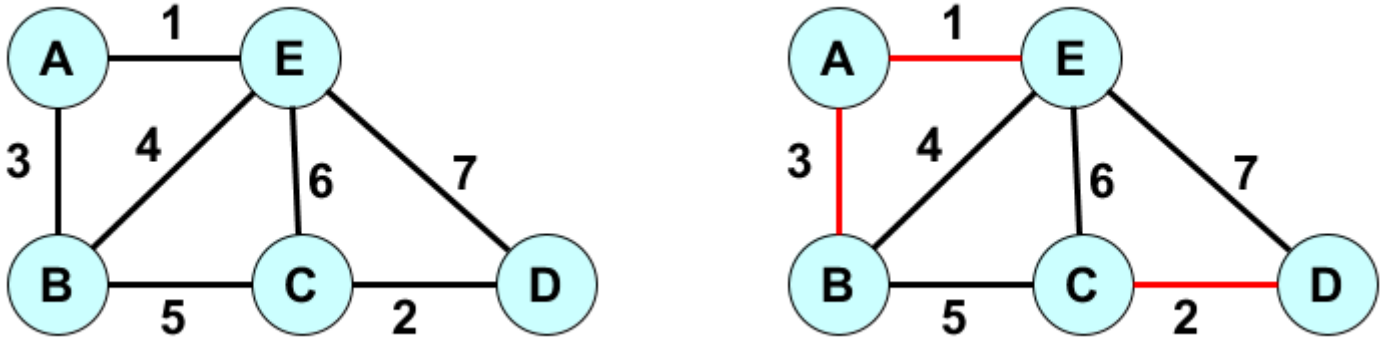
- Выдайте на выход 1 если цикл любой длины есть и 0 если его нет
- В данном случае петля есть

Обсуждение

- Каждый раз, когда мы проверяем таким образом цикличность, мы строим дерево, которой, при успешном построении будет остовным
- Как найти минимальное остовное дерево?
- Удивительно, но для этого есть жадный алгоритм, целых два

Жадные алгоритмы на графах

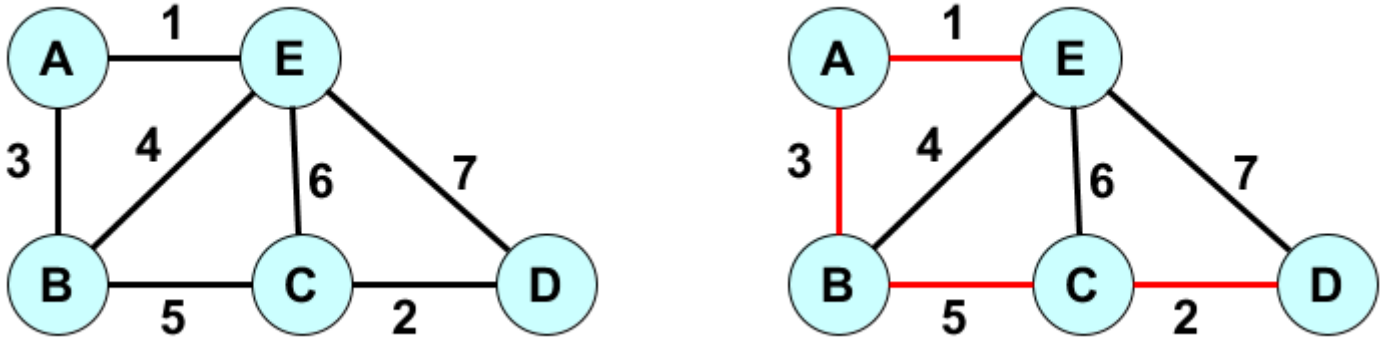
- Самый известный из жадных алгоритмов на графах это поиск минимального остовного дерева во взвешенном графе (алгоритм Краскала)



- Жадный шаг: берётся ребро с минимальным весом и добавляется в остовное дерево, если при этом не создаёт там цикла
- На рисунке выбраны рёбра с весами 1, 2 и 3. Далее ребро с весом 4 добавить нельзя, так как образуется цикл

Жадные алгоритмы на графах

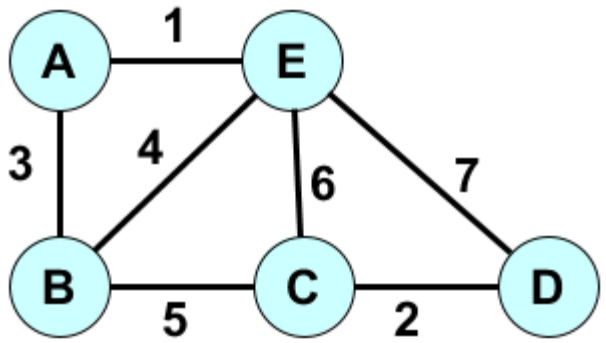
- Самый известный из жадных алгоритмов на графах это поиск минимального остовного дерева во взвешенном графе (алгоритм Краскала)



- Жадный шаг: берётся ребро с минимальным весом и добавляется в остовное дерево, если при этом не создаёт там цикла
- Поэтому берётся ребро с весом 5, после чего остовное дерево готово

Домашняя работа HWG

- Считайте со стандартного ввода имя файла, в котором граф задан как список рёбер. Например граф с прошлого слайда

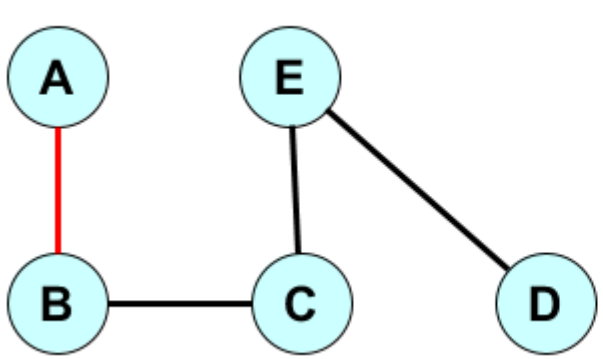


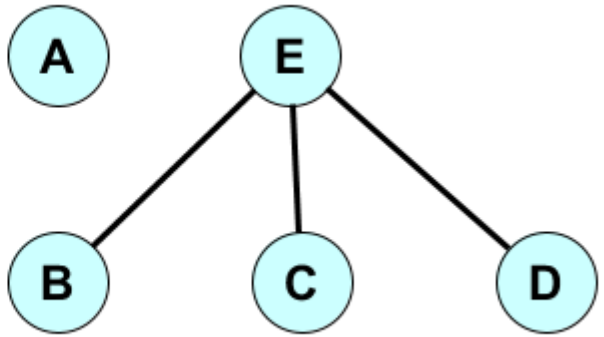

```
5
0 1 3 0 4 1
1 4 4 1 2 5
2 3 2 2 4 6
3 4 7
```

- Реализуйте на языке C алгоритм Краскала для поиска минимального остовного дерева. Напечатайте в `stdout` вес найденного дерева, в данном случае результат это 11

Жадные алгоритмы: скрытая структура

- Обратим внимание на интересное свойство ациклических подграфов

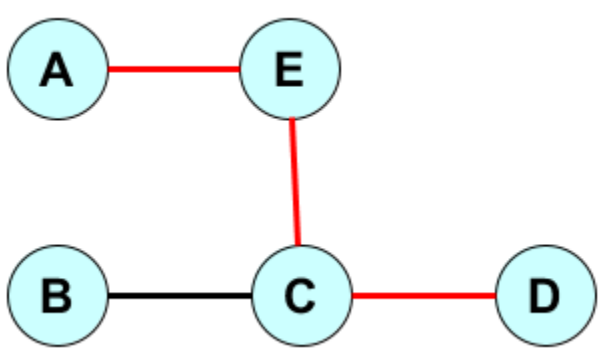


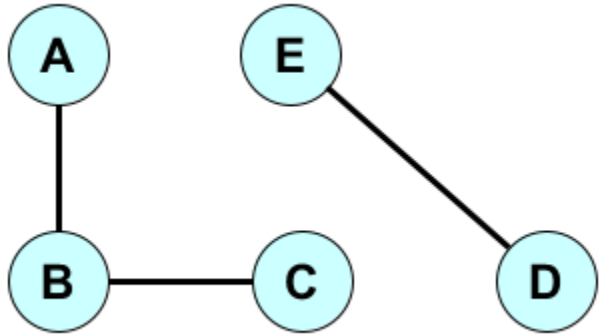


- Если в одном ациклическом подграфе меньше рёбер, чем в другом, то в большем всегда можно найти ребро, чтобы добавить в меньший, сохранив его ацикличность

Жадные алгоритмы: скрытая структура

- Обратим внимание на интересное свойство ациклических подграфов





- Если в одном ациклическом подграфе меньше рёбер, чем в другом, то в большем всегда можно найти ребро, чтобы добавить в меньший, сохранив его ацикличность. Иногда даже не одно

Матроиды

- Матроидом I называется система множеств, в которой

1. Пустое множество принадлежит I
2. Если множество принадлежит I, то все его подмножества принадлежат I
3. Если одно из принадлежащих I множеств больше другого по мощности, в большем всегда найдётся такой элемент, который можно добавить в меньшее так, чтобы меньшее множество с добавленным элементом тоже принадлежало I

- Все множества, входящие матроид состоят из элементов носителя матроида
- Базой матроида называется максимальное по включение множество в нём

Алгоритм Радо-Эдмондса

- Пусть дан матроид I, с носителем X, в котором каждому элементу x_i сопоставлен вес w_i, а каждое входящее в матроид множество имеет вес равный сумме весов своих элементов
- Алгоритм Радо-Эдмондса ищет в матроиде базу минимального веса, делая каждый раз жадный шаг, выбирая следующий элемент с наименьшим весом

$$A_0 = \emptyset$$

$$A_i = A_{i-1} + \{x\}, \text{ где } x = \min_{w_j}\{y_j \in X \setminus A_{i-1} \mid A_{i-1} + \{y_j\} \in I\}$$

- Любой жадный алгоритм на конкретном матроиде, например алгоритм Краскала, это всего лишь специализация этого общего алгоритма

Вернёмся к проблеме JA

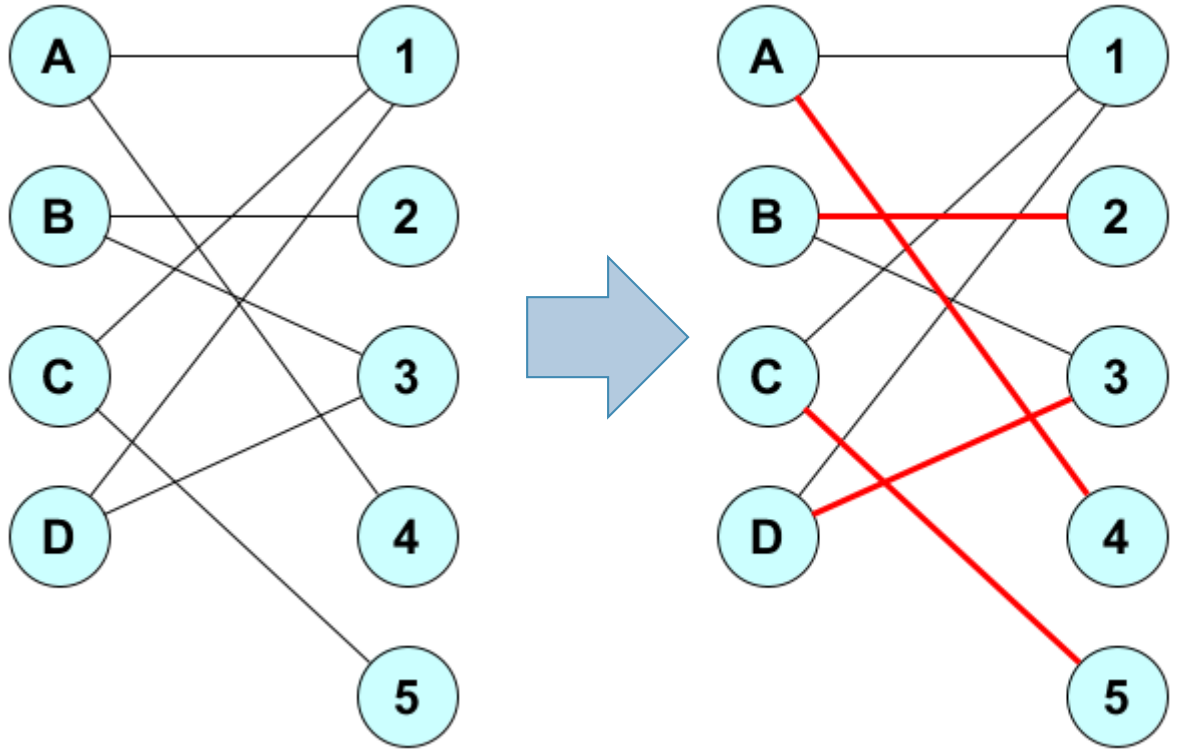
- У Кормена $[Cormen]$ есть замечательный анализ проблемы JA (см. ранее)

JobId	A	B	C	D	E
Deadline	2	1	2	1	3
Profit	100	27	25	19	25

- Назовём выполнимыми множества работ, которые можно сделать без просрочки дедлайна. Например {B, A, E} или {D, C} выполнимы
- Тогда можно проверить, что условия матроида выполняются для систем выполнимых множеств
- Тот жадный алгоритм, которым вы решали эту задачу, это тот же алгоритм, который ищет остовные деревья в графе (фантастика)

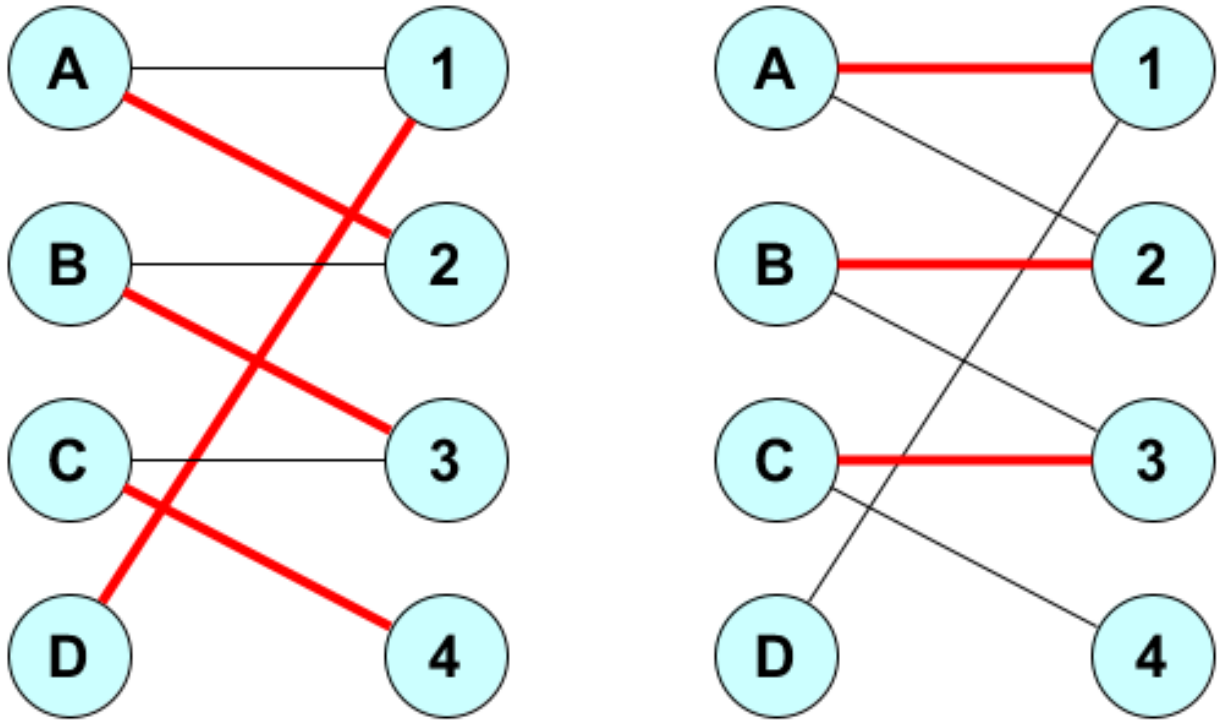
Распределение работников

- Интересная задача, которая тоже имеет структуру матроида это распределение работников по работам с целью максимизации прибыли
- Слева буквами заданы работники
- Справа цифрами стоимость работ
- Показано оптимальное решение
- Видите ли вы структуру матроида?
- Что является множествами?
- Как будет выглядеть жадный алгоритм решения этой задачи?



Матроид не образуют рёбра сочетания

- Тривиальный контрпример
- Здесь нарушается [aug]
- Тем не менее, жадный алгоритм возможен
- В чём же дело?

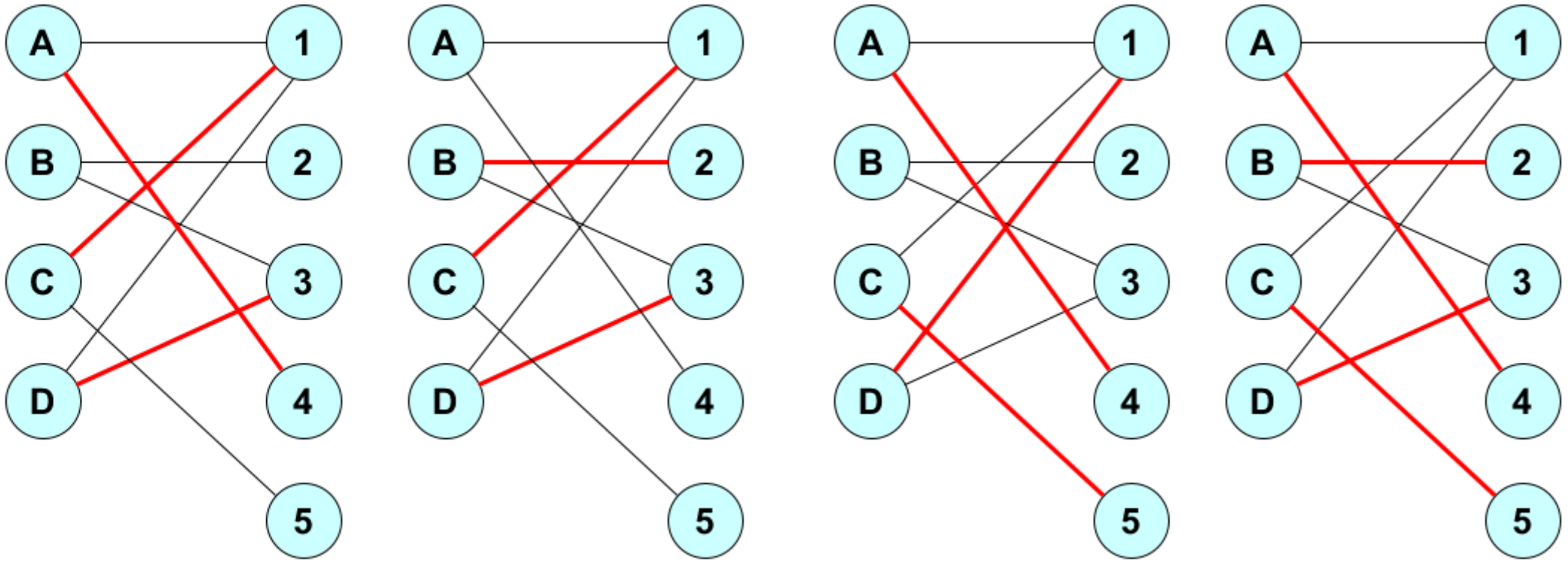


Обсуждение

- Если бы рёбра сочетания образовали матроид, то жадный алгоритм существовал бы для взвешенного паросочетания
- Понимаете ли вы почему это так?

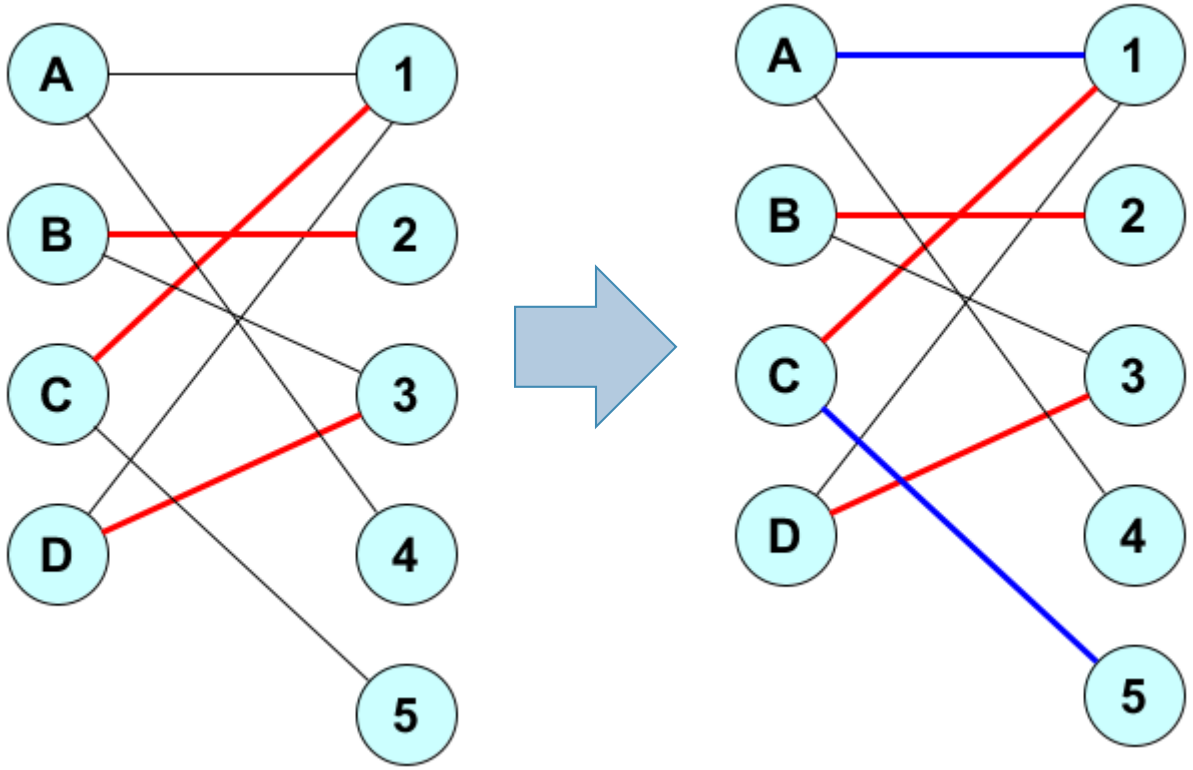
Выполнимые множества

- Назовём множество работ выполнимым, если для него есть удовлетворяющее его паросочетание



Идея увеличивающего пути

- Увеличивающим путём для выбранного паросочетания называется путь с чередующимися рёбрами и с обоими концами в свободных вершинах
- Если два множества выполнимы и одно больше другого, то всегда найдётся увеличивающий путь в меньшем сочетании с одним из концов в большем сочетании
- Это задаёт секущий матроид и подсказывает жадный алгоритм



$\{1, 2, 3\} \rightarrow \{1, 2, 3, 5\}$

Обсуждение

- И какой же жадный алгоритм поиска максимального паросочетания это вам подсказывает?

Вернёмся к монетам

- Выполним следующие операции:
  1. Разменяем жадным алгоритмом и посмотрим сколько получилось монет
  2. Рассмотрим все частичные размены не более чем по столько монет
- Например для 6 монет и разменного множества {4, 3, 1}
- Имеем частичные размены не более чем по 3 монеты:
  {4, 1, 1}, {3, 3, 1}, {3, 1, 1},{1, 1, 1},
  {4, 1}, {3, 1}, {3, 3}, {1, 1}, {4}, {3}, {1}
- Они не образуют матроид ([aug] нарушается для {4, 1, 1} и {3, 3})
- Значит жадный алгоритм для этого разменного множества не работает

Вернёмся к монетам

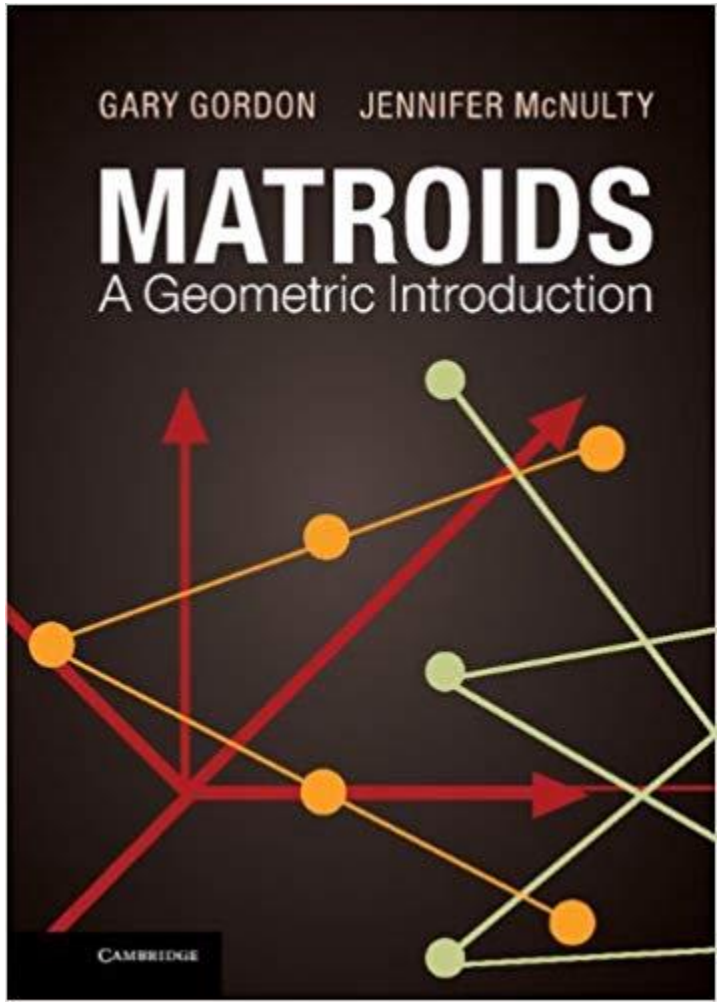
- Выполним следующие операции:
  1. Разменяем жадным алгоритмом и посмотрим сколько получилось монет
  2. Рассмотрим все частичные размены не более чем по столько монет
- Например для 6 монет и разменного множества $\{4, 2, 1\}$
- Имеем частичные размены не более чем по 3 монеты:
  $\{4, 2, 1\}, \{2, 2, 1\}, \{4, 1, 1\}, \{2, 1, 1\}, \{1, 1, 1\},$
  $\{4, 1\}, \{2, 1\}, \{2, 2\}, \{1, 1\}, \{4\}, \{2\}, \{1\}$
- Они образуют матроид
- Значит жадный алгоритм для этого разменного множества работает

Больше о матроидах

- На самом деле матроиды это удивительные комбинаторные объекты
- Они имеют странную связь с графами, в частности с планарными графами
- У них масса обобщений, разной степени обобщённости
- Через матроиды устанавливаются криптоморфизмы между различными дискретными структурами
- Подробнее можно почитать в $[GM]$ (для математически настроенной части аудитории)

Литература

- [C11] ISO/IEC – "Information technology – Programming languages – C", 2011
- [$Cormen$] Thomas H. Cormen – Introduction to Algorithms, 2009
- [GM] Gordon, McNulty – Matroids, A Geometric Introduction, 1993
- [KS] Klappenecker – Theory of Greedy Algorithms
- [NP] Новиков, Поздняков «Жадные алгоритмы»
- [AT] Алексеев, Таланов , «Графы и алгоритмы», онлайн-курс, Intuit



СЕКРЕТНЫЙ УРОВЕНЬ

Графические матроиды

Матроиды без украшений

- Матроид это просто система множеств $I \subseteq 2^E$ над своим носителем
- Ниже считаем, что ab это сокращение для $\{a, b\}$

$E = \{a, b, c\}, \qquad I = \{\emptyset, a, c, ab, ac\}$

$E = \{a, b, c\}, \qquad I = \{\emptyset, a, b, c, ab\}$

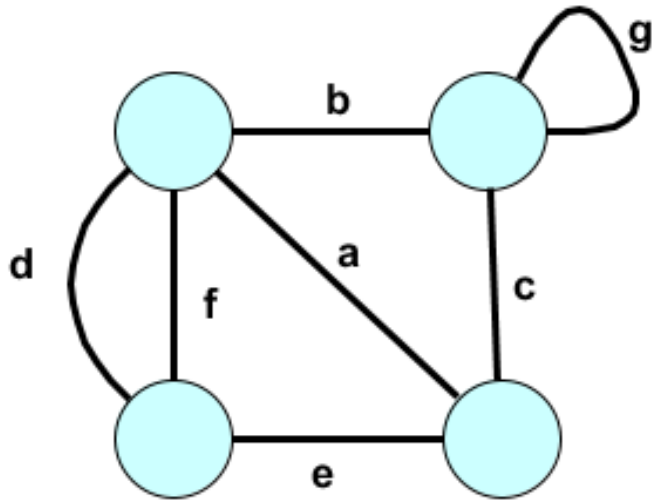
$E = \{a, b, c, d\}, \qquad I = \{\emptyset, a, b, c, ab, ac, abc\}$

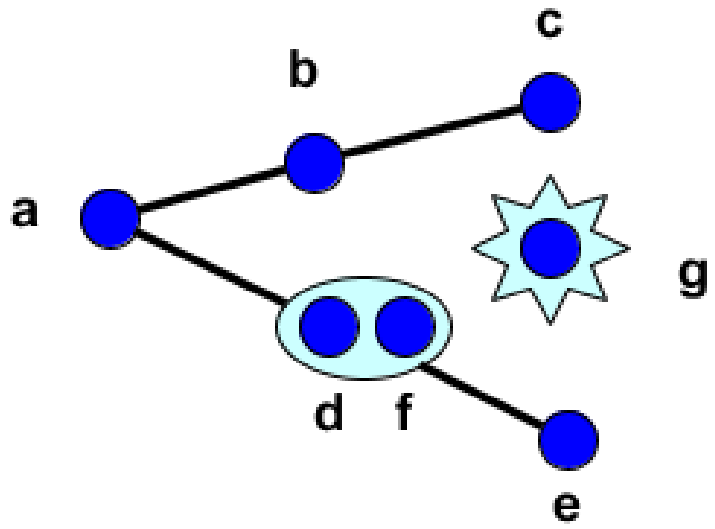
$E = \{a, b, c, d, e, f\}, \qquad I = E + \{xy \mid x, y \in E\} + \{xyz \mid x, y, z \in E\} - \{abc, bcd, cde, def\}$

$E = \{a, b, c, d, e, f\}, \qquad I = E + \{xy \mid x, y \in E\} + \{abc, bcd, cde, def\}$

- Какие из перечисленных систем множеств являются матроидами?
- Как вы думаете над этой проблемой?

Матроид из графа



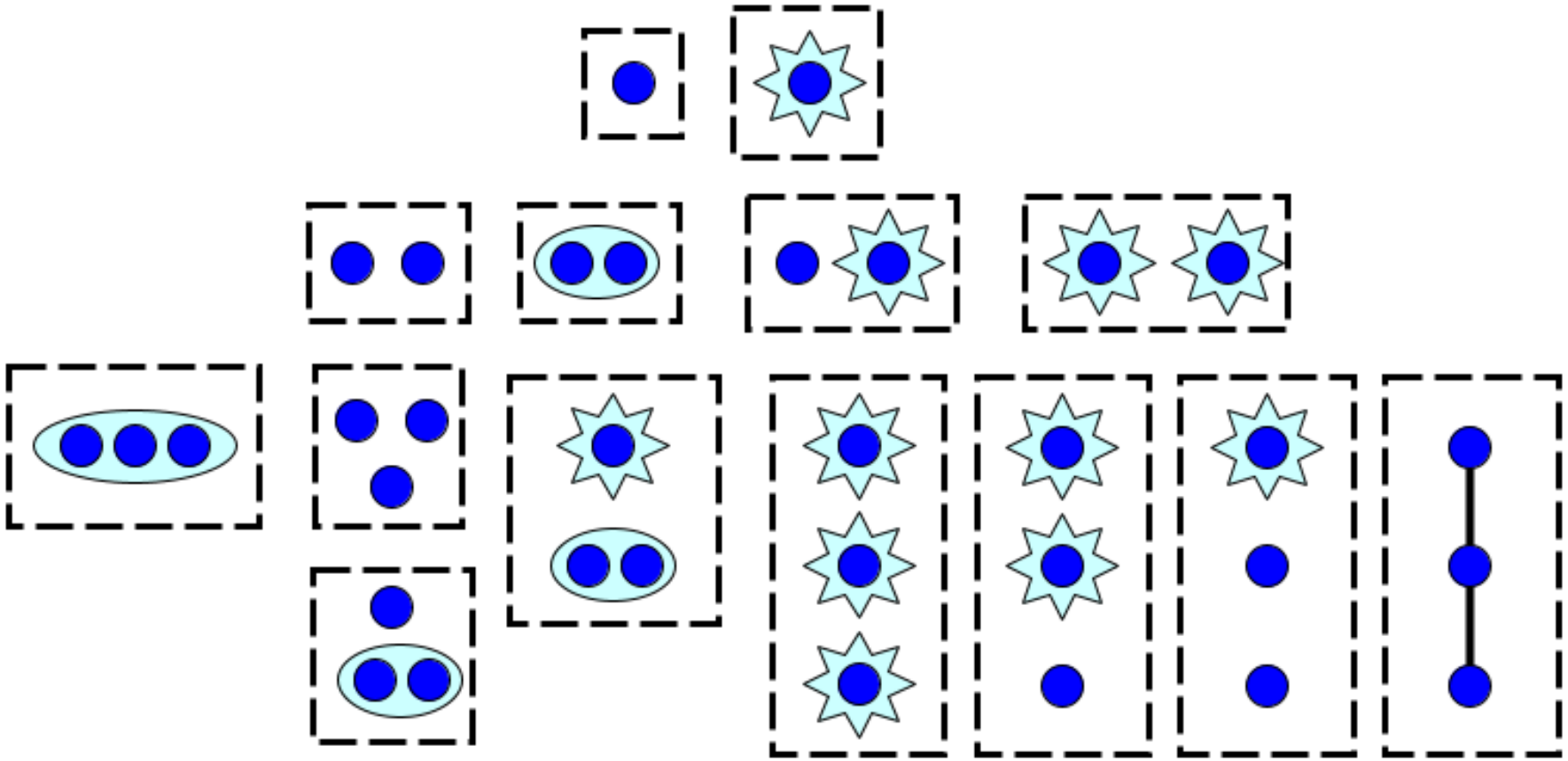


- Матроид для этого графа, состоящий из его остовных деревьев (циклический или графический матроид)

$$E = \{a, b, c, d, e, f, g\}, F = E - \{g\}$$

$$I = F + \{xy \mid x, y \in F\} - df + \{xyz \mid x, y, z \in F\} - \{abc, ade, afe\}$$

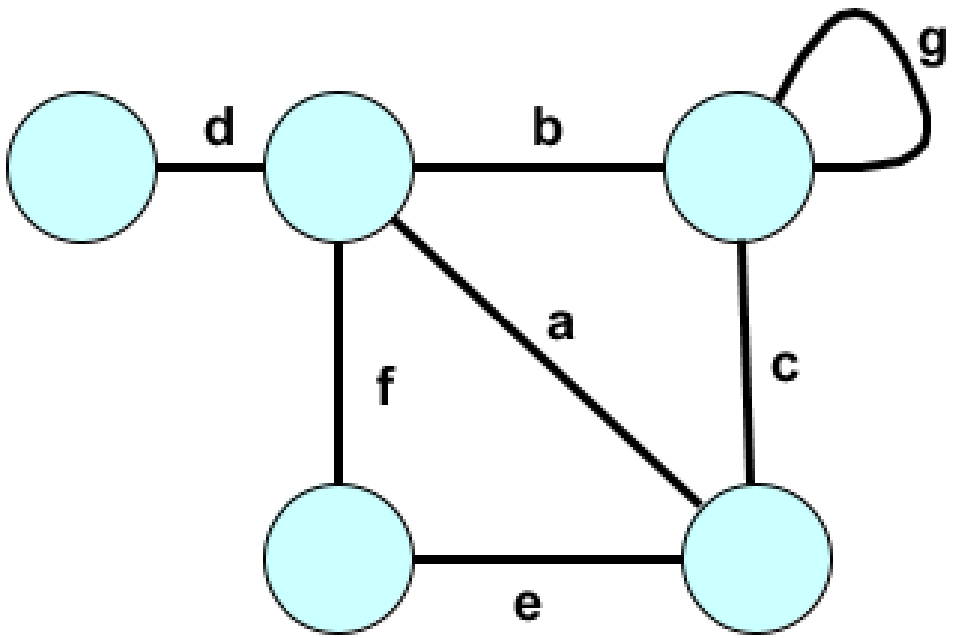
Подсчёт матроидов

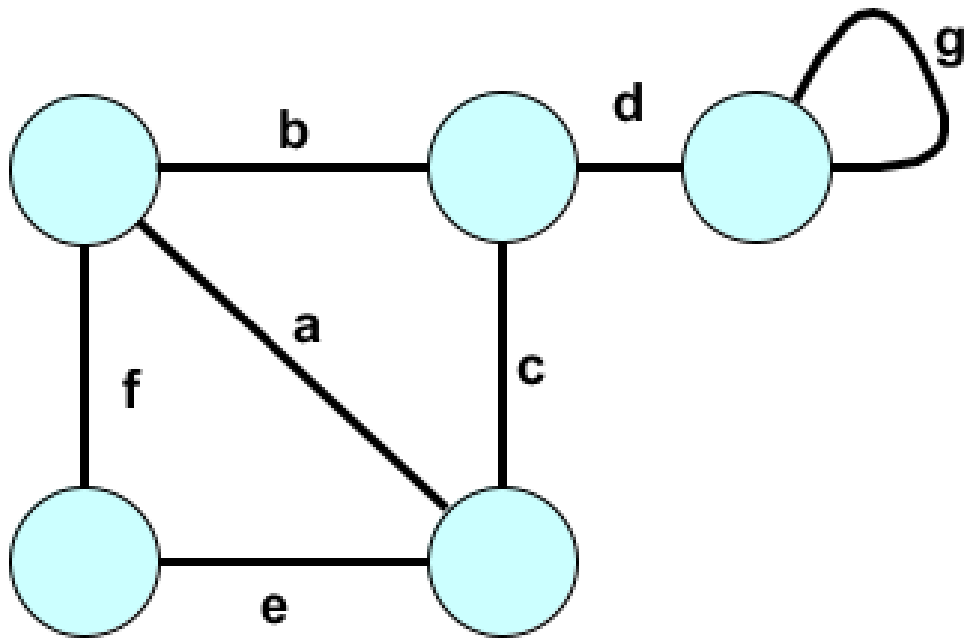


- Количество матроидов: `2, 4, 8, 17, 38, 98, 306, 1724, ...` (см. $[MR]$)

Графические матроиды

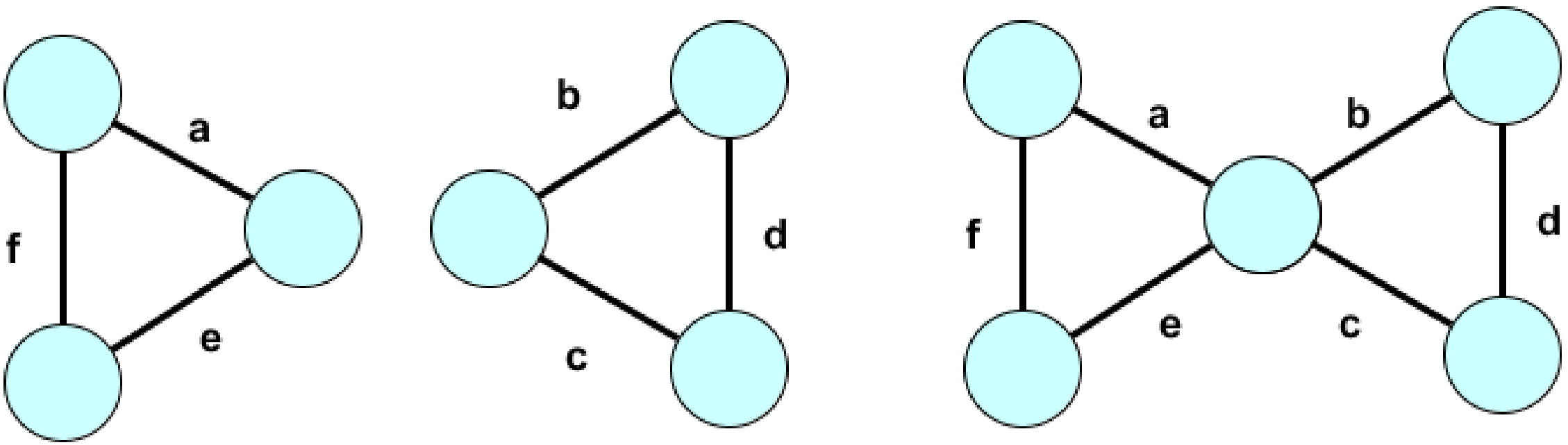
- Графический матроид иногда получается одинаковым для разных графов





Графические матроиды

- Графический матроид иногда получается одинаковым для разных графов



Обсуждение

- Каждый ли матроид является графическим?

Обсуждение

- Каждый ли матроид является графическим?
- Нет. И пример привести очень просто

$E = \{a, b, c, d\}$

$I = E + \{xy \mid x, y \in E\}$

- Можно ли истолковать этот матроид как-то иначе?