Мини-задача #28 (2 балла)

Реализовать фильтр Блума для IP-адресов (через битсет).

Для конфигурации структуры данных на вход подается прогнозируемое общее количество запросов и желаемая вероятность ошибки.

В качестве хеш-функций использовать случайные функции из универсального семейства хеш-функций, описанного в главе об универсальном хешировании.

*понятно, что гарантии на их свойства слабые, но для примера будем использовать их.

Алгоритмы и структуры данных

Фильтр Блума, алгоритм Каргера поиска наименьшего разреза в графе

Хеш-множества

Операции:

1. find(value) -> O(1)

---

7. insert(value) -> O(1)
8. remove(value) -> O(1)

Спойлер: хеш-таблицы лучше здесь (при правильной реализации дадут O(1))

Хеш-множества

Можем ли мы лучше?

Операции:

1. find(value) -> O(1)

---

7. insert(value) -> O(1)
8. remove(value) -> O(1)

Спойлер: хеш-таблицы лучше здесь (при правильной реализации дадут O(1))

Хеш-множества

Можем ли мы лучше?

В каком-то смысле да.

Операции:

1. find(value) -> O(1)

---

7. insert(value) -> O(1)
8. remove(value) -> O(1)

Спойлер: хеш-таблицы лучше здесь (при правильной реализации дадут O(1))

Множества

Операции:

1. `lookup(value): V -> {True, False}: -> O(1)`
2. `insert(value):                     -> O(1)`

Фильтр Блума

Операции:

1. lookup(value): V -> {True, False}: -> O(1)
2. insert(value): -> O(1)

При этом:

1. Структура данных будет намного эффективнее по памяти, чем хеш-таблица/хеш-множество

Фильтр Блума

Операции:

1. lookup(value): V -> {True, False}: -> O(1)
2. insert(value): -> O(1)

При этом:

1. Структура данных будет намного эффективнее по памяти, чем хеш-таблица/хеш-множество

2. Но функции lookup может давать ложноположительный ответ.

Фильтр Блума

Операции:

1. lookup(value): V -> {True, False}: -> O(1)
2. insert(value): -> O(1)

При этом:

1. Структура данных будет намного эффективнее по памяти, чем хеш-таблица/хеш-множество

2. Но функции lookup может давать ложноположительный ответ. Т.е. вы никогда не звали insert от некоторого значения, а lookup от него при этом возвращает True.

Фильтр Блума

Применения:

1. Исторически использовали для проверки орфографии.
    - Правильные слова добавлялись в фильтр Блума
    - Редактор текста проверял каждое слово: есть оно в фильтре или нет.

Фильтр Блума

Применения:

1. Исторически использовали для проверки орфографии.
    - Правильные слова добавлялись в фильтр Блума
    - Редактор текста проверял каждое слово: есть оно в фильтре или нет.

    Сильная экономия памяти и времени
    (важно для старых машин)

Фильтр Блума

Применения:

1. Исторически использовали для проверки орфографии.
    - Правильные слова добавлялись в фильтр Блума
    - Редактор текста проверял каждое слово: есть оно в фильтре или нет.

    Сильная экономия памяти и времени
    (важно для старых машин)

    Ложноположительное срабатывание?
    Ну что ж, это не самый надежный spellchecker.

Фильтр Блума

Применения:

1. Исторически использовали для проверки орфографии,
2. Проверка надежности пароля

Фильтр Блума

Применения:

1. Исторически использовали для проверки орфографии,
2. Проверка надежности пароля,
3. Первый этап реализации кэшей, особенно в сетевых приложениях, особенно на железе;

   http://wiki.squid-cache.org/SquidFaq/CacheDigests

Фильтр Блума

Применения:

1. Исторически использовали для проверки орфографии,
2. Проверка надежности пароля,
3. Первый этап реализации кэшей, особенно в сетевых приложениях, особенно на железе;

   http://wiki.squid-cache.org/SquidFaq/CacheDigests

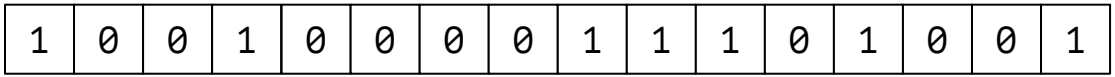
4. Первая проверка перед долго операцией доступа к диску в базе данных

   https://ru.wikipedia.org/wiki/BigTable

Фильтр Блума: реализация

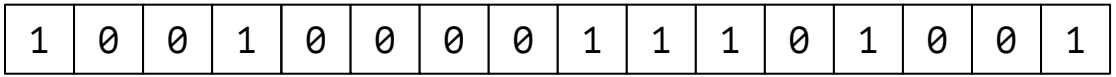
Массив

Фильтр Блума: реализация



Массив битов!

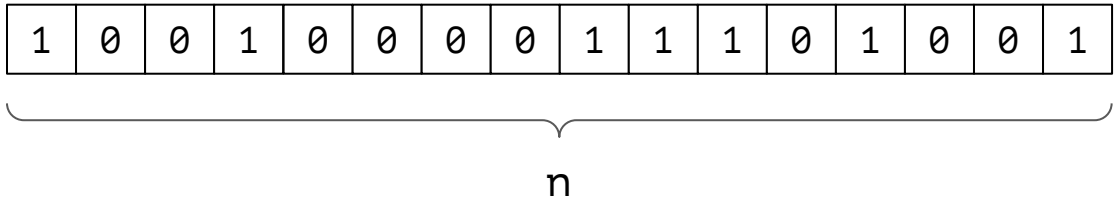
Фильтр Блума: реализация



Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

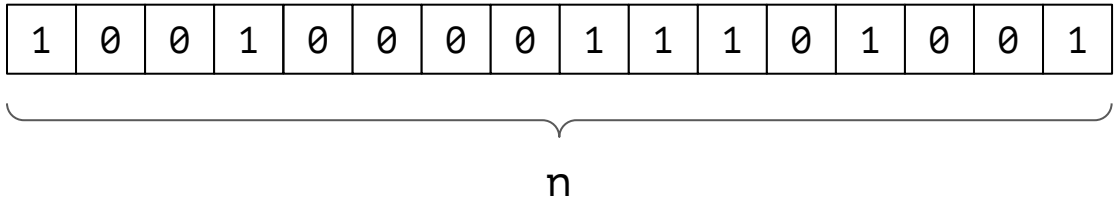
Фильтр Блума: реализация



Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Фильтр Блума: реализация

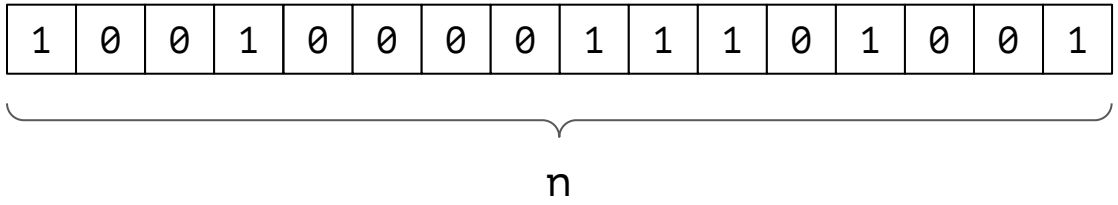


При этом говорят, что если в фильтре блума S всего |S| объектов, то мы используем n / |S| битов на каждый объект.

Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Фильтр Блума: реализация



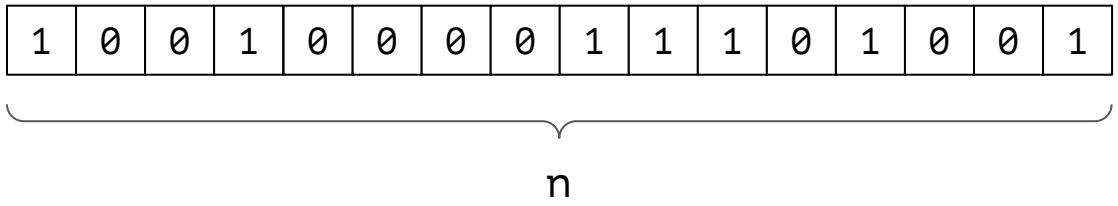
При этом говорят, что если в фильтре блума S всего |S| объектов, то мы используем n / |S| битов на каждый объект.

Что обычно НАМНОГО меньше, чем если бы мы хранили сами объекты или указатели на них.

Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Фильтр Блума: реализация



При этом говорят, что если в фильтре блума S всего |S| объектов, то мы используем n / |S| битов на каждый объект.

Что обычно НАМНОГО меньше, чем если бы мы хранили сами объекты или указатели на них.

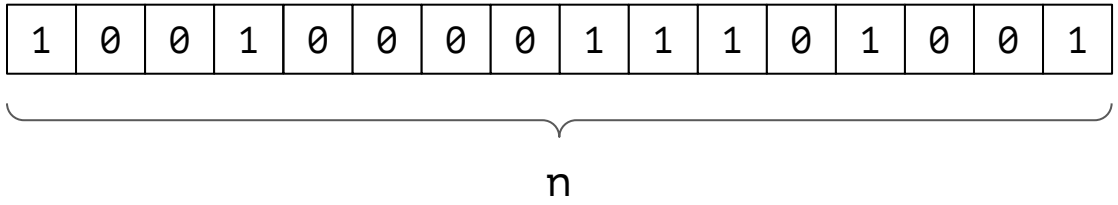
Пример: фильтр для IP адресов. Если бы хранили их плоско, то каждая запись - 4 байта.

В фильтре блума часто хватает 8 бит.

Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Фильтр Блума: реализация



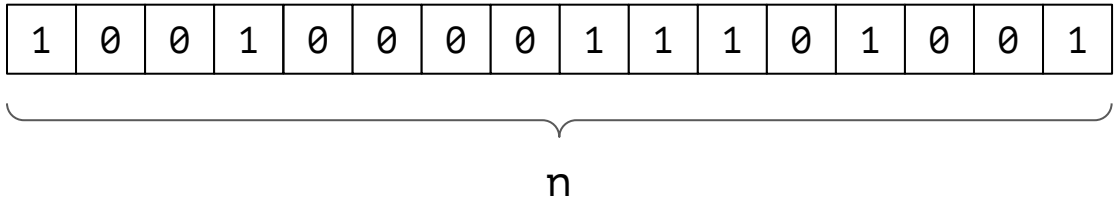
При этом говорят, что если в фильтре блума S всего |S| объектов, то мы используем n / |S| битов на каждый объект.

Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

И при этом есть набор хеш-функций $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

Фильтр Блума: реализация



При этом говорят, что если в фильтре блума S всего |S| объектов, то мы используем n / |S| битов на каждый объект.

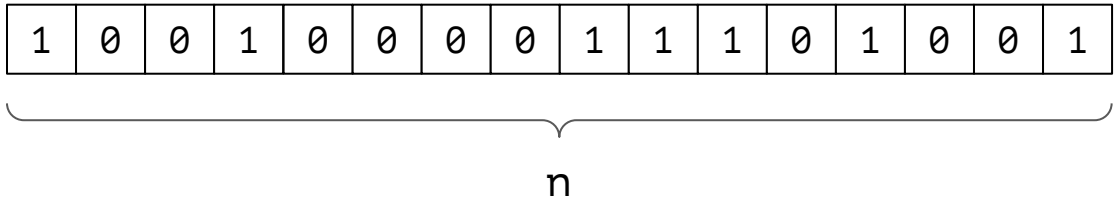
Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

И при этом есть набор хеш-функций $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1) insert(v): безусловно ставим k единиц: $A[h_1(v)] = 1, A[h_2(v)] = 1, \ldots, A[h_k(v)] = 1.$

Фильтр Блума: реализация



При этом говорят, что если в фильтре блума S всего |S| объектов, то мы используем n / |S| битов на каждый объект.

Массив битов!

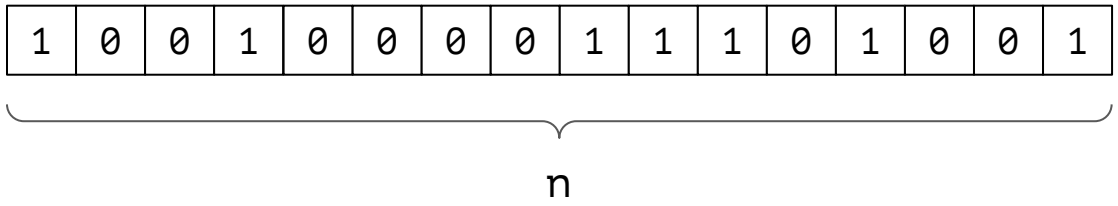
BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

И при этом есть набор хеш-функций $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1) insert(v): безусловно ставим k единиц: $A[h_1(v)] = 1, A[h_2(v)] = 1, \ldots, A[h_k(v)] = 1.$

2) lookup(v)?

Фильтр Блума: реализация



При этом говорят, что если в фильтре блума S всего |S| объектов, то мы используем n / |S| битов на каждый объект.

Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

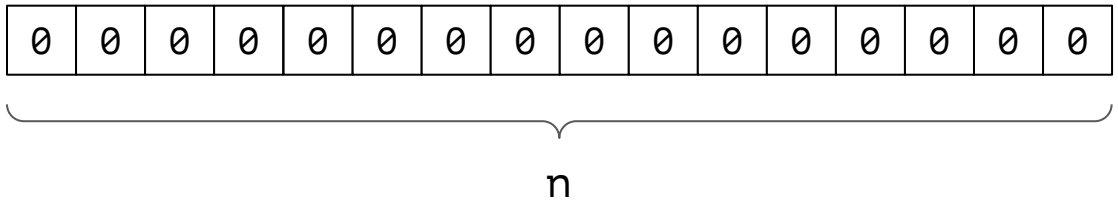
И при этом есть набор хеш-функций $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1) insert(v): безусловно ставим k единиц: $A[h_1(v)] = 1, A[h_2(v)] = 1, \ldots, A[h_k(v)] = 1.$

2) lookup(v): бездумно проверяем k бит на единички:

$if\ A[h_1(v)] == 1,\ \ldots,\ A[h_k(v)] == 1 \Rightarrow v \in S$

Фильтр Блума: реализация



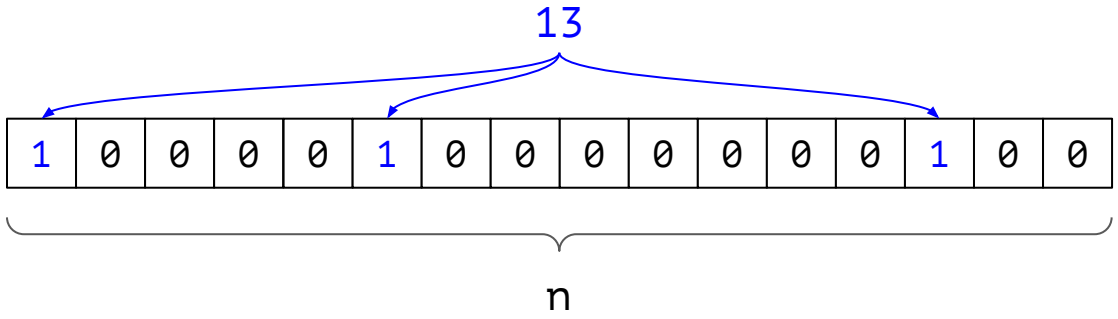
Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Пусть есть три хеш-функции h1, h2, h3.

insert(13)

Фильтр Блума: реализация



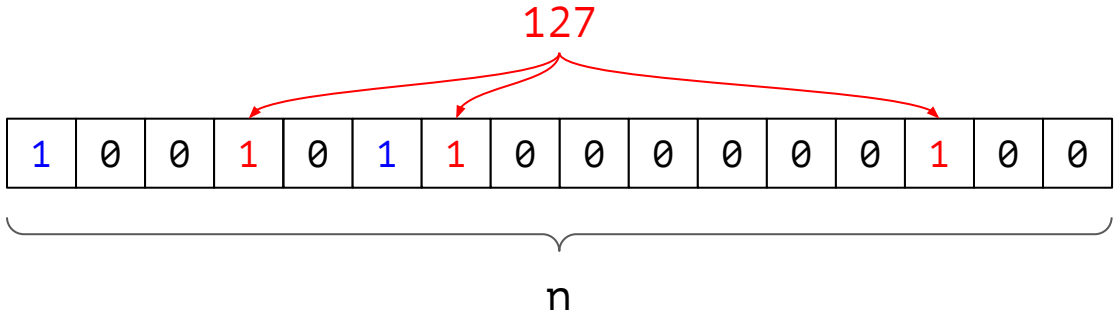
Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Пусть есть три хеш-функции h1, h2, h3.

insert(13) => h1(13) = 5, h2(13) = 0, h3(13) = 13

Фильтр Блума: реализация



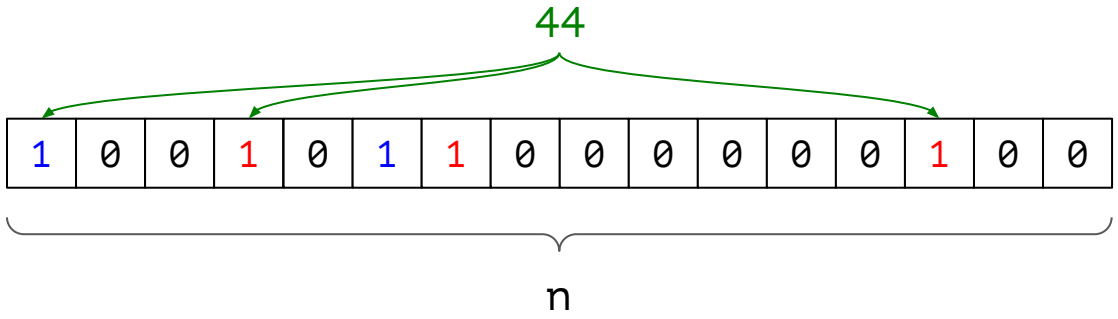
Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Пусть есть три хеш-функции h1, h2, h3.

insert(13) => h1(13) = 5, h2(13) = 0, h3(13) = 13
insert(127) => h1(127) = 3, h2(127) = 6, h3(127) = 13

Фильтр Блума: реализация



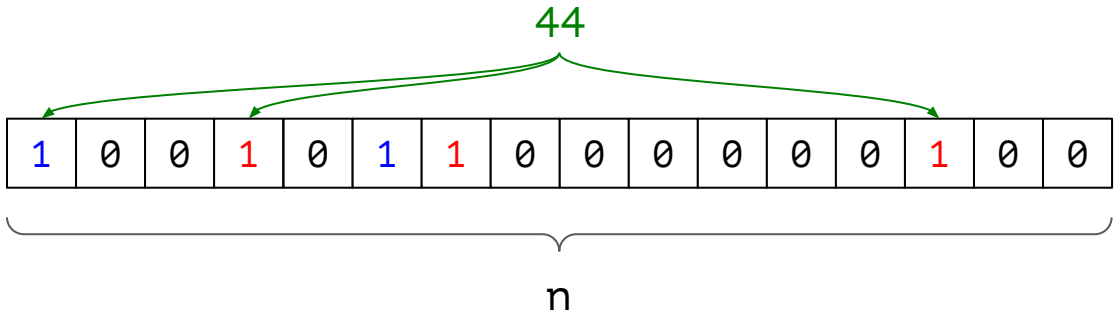
Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Пусть есть три хеш-функции h1, h2, h3.

```
insert(13) => h1(13) = 5, h2(13) = 0, h3(13) = 13
insert(127) => h1(127) = 3, h2(127) = 6, h3(127) = 13

lookup(44) => h1(44) = 0, h2(44) = 13, h3(44) = 3
```

Фильтр Блума: реализация



Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

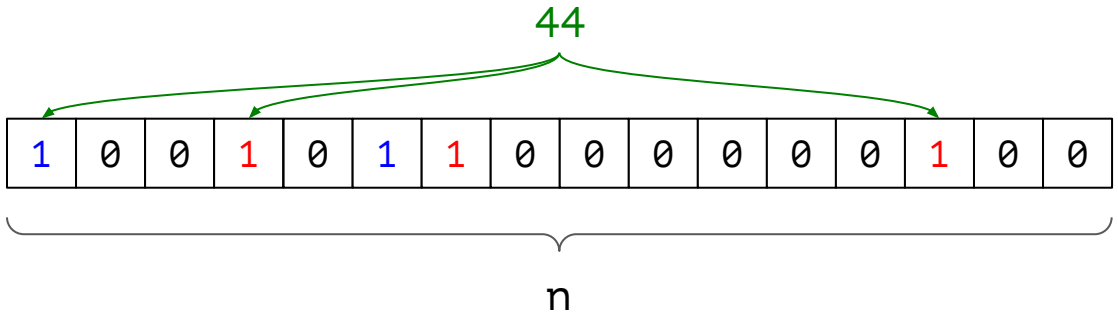
Пусть есть три хеш-функции h1, h2, h3.

insert(13) => h1(13) = 5, h2(13) = 0, h3(13) = 13
insert(127) => h1(127) = 3, h2(127) = 6, h3(127) = 13

lookup(44) => h1(44) = 0, h2(44) = 13, h3(44) = 3

Это и есть пример false positive. Не добавляли 44, но фильтр думает иначе.

Фильтр Блума: реализация



Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Пусть есть три хеш-функции h1, h2, h3.

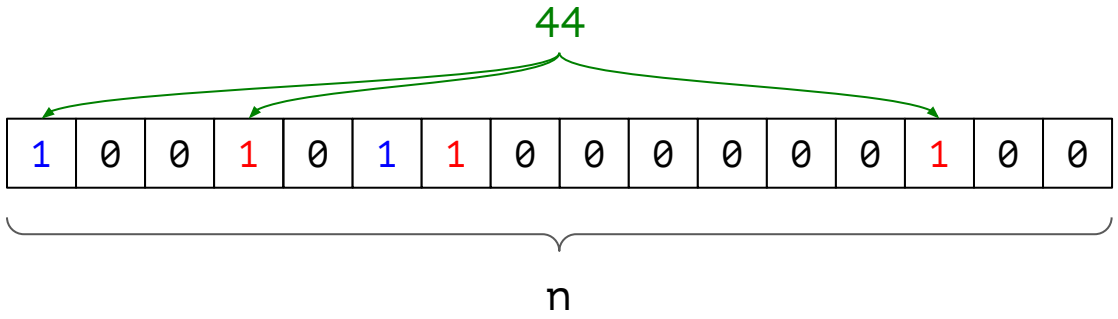
```
insert(13) => h1(13) = 5, h2(13) = 0, h3(13) = 13
insert(127) => h1(127) = 3, h2(127) = 6, h3(127) = 13

lookup(44) => h1(44) = 0, h2(44) = 13, h3(44) = 3
```

O(1)

O(1)

Это и есть пример false positive. Не добавляли 44, но фильтр думает иначе.

Фильтр Блума: реализация



Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Пусть есть три хеш-функции h1, h2, h3.

insert(13) => h1(13) = 5, h2(13) = 0, h3(13) = 13
insert(127) => h1(127) = 3, h2(127) = 6, h3(127) = 13

O(1)

lookup(44) => h1(44) = 0, h2(44) = 13, h3(44) = 3

O(1)

Это и есть пример false positive. Не добавляли 44, но фильтр думает иначе.

Насколько часто такое бывает?

Немного теории вероятностей #1

Ω – множество элементарных событий (все, что может произойти)

$p : \Omega \to [0, 1]$ – отображает элементарные события на их вероятности

$$\sum_{i \in \Omega} p(i) = 1$$

Немного теории вероятностей #2

Событие — подмножество $S \subseteq \Omega$

Вероятность события S: $Pr[S] = \sum_{i \in S} p(i)$

Немного теории вероятностей #2

Событие – подмножество $S \subseteq \Omega$

Вероятность события S: $Pr[S] = \sum_{i \in S} p(i)$

Пример #4: Событие – при броске двух игральных костей выпала сумма 7

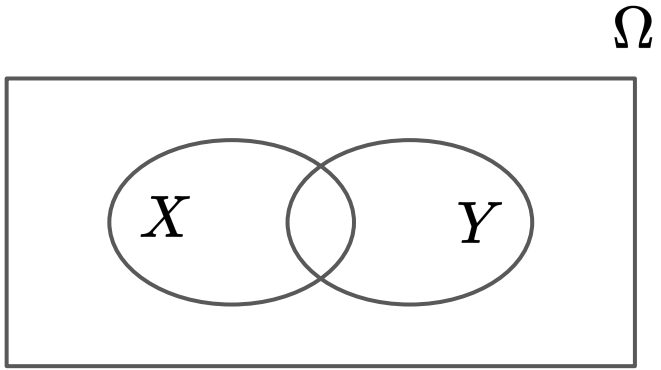
Назовите вероятность такого события?

$$S = \{(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)\}$$

$$Pr[S] = \sum_{i \in S} p(i) = \frac{1}{6}$$

Немного теории вероятностей #5

Пусть есть два события $X, Y \subseteq \Omega$

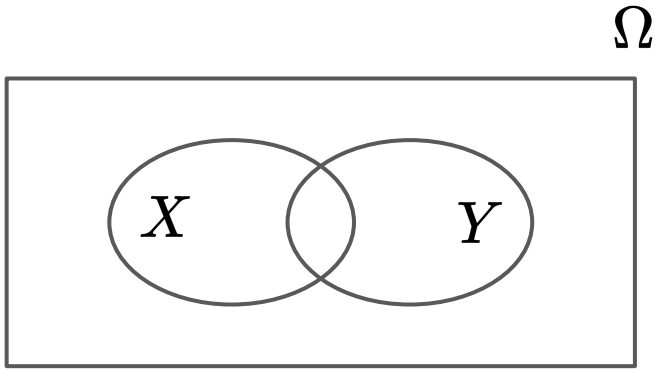


Немного теории вероятностей #5

Пусть есть два события $X, Y \subseteq \Omega$

Тогда условная вероятность срабатывания X при условии события осуществления Y:

$$Pr[X|Y]$$

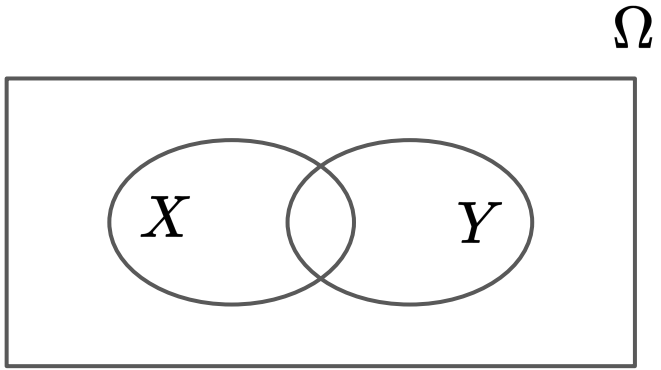


Немного теории вероятностей #5

Пусть есть два события $X, Y \subseteq \Omega$

Тогда условная вероятность срабатывания X при условии события осуществления Y:

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]}$$



Немного теории вероятностей #2

Событие – подмножество $S \subseteq \Omega$

Вероятность события S: $Pr[S] = \sum_{i \in S} p(i)$

Пример #4: Событие – при броске двух игральных костей выпала сумма 7

Назовите вероятность такого события?

$$S = \{(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)\}$$

$$Pr[S] = \sum_{i \in S} p(i) = \frac{1}{6}$$

Немного теории вероятностей #5

Пример #13: Чему равна вероятность того, что вам выпала хотя бы одна единица, если вы бросили две игральные кости на сумму 7?

Немного теории вероятностей #5

Пример #13: Чему равна вероятность того, что вам выпала хотя бы одна единица, если вы бросили две игральные кости на сумму 7?

Решение: X - событие, что у вас есть хотя бы одна единица. Y - событие, что вы выкинули на 7.

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]}$$

Немного теории вероятностей #5

Пример #13: Чему равна вероятность того, что вам выпала хотя бы одна единица, если вы бросили две игральные кости на сумму 7?

Решение: X - событие, что у вас есть хотя бы одна единица. Y - событие, что вы выкинули на 7.

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]} = 1/6$$

Немного теории вероятностей #5

Пример #13: Чему равна вероятность того, что вам выпала хотя бы одна единица, если вы бросили две игральные кости на сумму 7?

Решение: X - событие, что у вас есть хотя бы одна единица. Y - событие, что вы выкинули на 7.

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]}$$ = 1/6

$$Y = \{(1,6),(2,5),(3,4),(4,3),(5,2),(6,1)\}$$

$$Pr[Y] = \sum_{i \in S} p(i) = \frac{6}{36} = \frac{1}{6}$$

Немного теории вероятностей #5

Пример #13: Чему равна вероятность того, что вам выпала хотя бы одна единица, если вы бросили две игральные кости на сумму 7?

Решение: X - событие, что у вас есть хотя бы одна единица. Y - событие, что вы выкинули на 7.

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]}$$

$= 1/6$

$$Y = \{\boxed{(1,6)}, (2,5), (3,4), (4,3), (5,2), \boxed{(6,1)}\}$$

$$Pr[Y] = \sum_{i \in S} p(i) = \frac{6}{36} = \frac{1}{6}$$

Немного теории вероятностей #5

Пример #13: Чему равна вероятность того, что вам выпала хотя бы одна единица, если вы бросили две игральные кости на сумму 7?

Решение: X - событие, что у вас есть хотя бы одна единица. Y - событие, что вы выкинули на 7.

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]}$$

= 1/6

$$Y = \{(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)\} \qquad Pr[X \cap Y] = \frac{2}{36}$$

$$Pr[Y] = \sum_{i \in S} p(i) = \frac{6}{36} = \frac{1}{6}$$

Немного теории вероятностей #5

Пример #13: Чему равна вероятность того, что вам выпала хотя бы одна единица, если вы бросили две игральные кости на сумму 7?

Решение: X - событие, что у вас есть хотя бы одна единица. Y - событие, что вы выкинули на 7.

$$Pr[X|Y] = \frac{Pr[X \cap Y]}{Pr[Y]} = \frac{12}{36} = \frac{1}{3}$$

$= 1/6$

$$Y = \{(1,6), (2,5), (3,4), (4,3), (5,2), (6,1)\} \qquad Pr[X \cap Y] = \frac{2}{36}$$

$$Pr[Y] = \sum_{i \in S} p(i) = \frac{6}{36} = \frac{1}{6}$$

Немного теории вероятностей #6

Пусть есть два события $X, Y \subseteq \Omega$

Такие события называются независимыми, если:

$$Pr[X \cap Y] = Pr[X] * Pr[Y]$$

Немного теории вероятностей #6

Пусть есть два события $X, Y \subseteq \Omega$

Такие события называются независимыми, если:

$$Pr[X \cap Y] = Pr[X] * Pr[Y]$$

Тогда: $Pr[X|Y] = Pr[X]$ и $Pr[Y|X] = Pr[Y]$
(по определению условной вероятности)

Немного теории вероятностей #6

Пусть есть два события $X, Y \subseteq \Omega$

Такие события называются независимыми, если:

$$Pr[X \cap Y] = Pr[X] * Pr[Y]$$

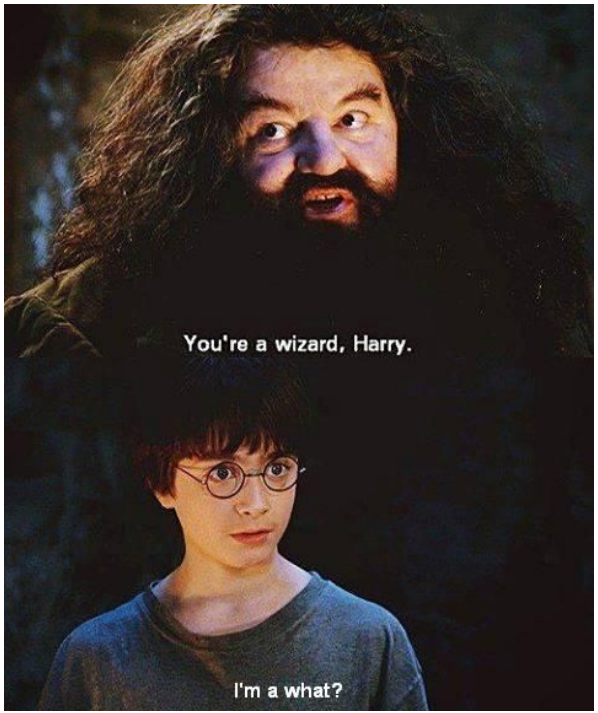
Тогда: $Pr[X|Y] = Pr[X]$ и $Pr[Y|X] = Pr[Y]$
(по определению условной вероятности)

И тогда если X и Y независимые случайные величины,
то: $\mathbb{E}[X * Y] = \mathbb{E}[X] * \mathbb{E}[Y]$ (док-во - упражнение).

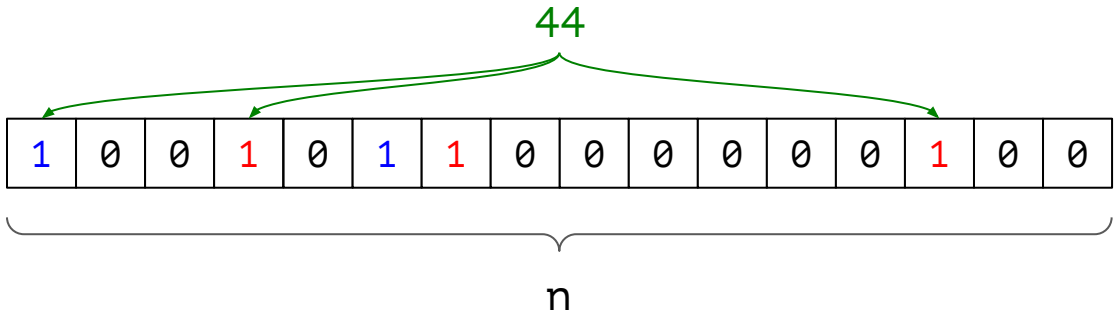
Немного теории вероятностей

Вот и все, что нам нужно для анализа вероятности ложноположительного срабатывания фильтра Блума!

(и этой лекции в целом)



Фильтр Блума: реализация



Массив битов!

BitSet, обычно реализуется как массив из int-ов, в каждом из которых используются биты.

Пусть есть три хеш-функции h1, h2, h3.

```
insert(13) => h1(13) = 5, h2(13) = 0, h3(13) = 13
insert(127) => h1(127) = 3, h2(127) = 6, h3(127) = 13

lookup(44) => h1(44) = 0, h2(44) = 13, h3(44) = 3
```

O(1)

O(1)

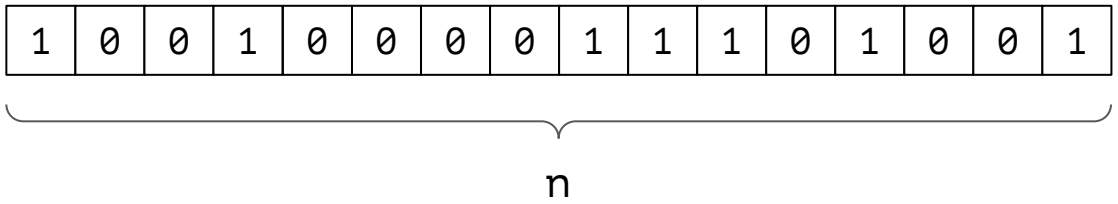
Это и есть приме false positive. Не добавляли 44, но фильтр думает иначе.

Насколько часто такое бывает?

Фильтр Блума: анализ

Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

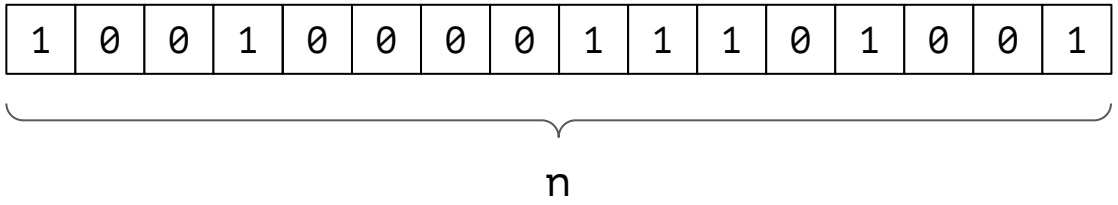


Фильтр Блума: анализ

Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).



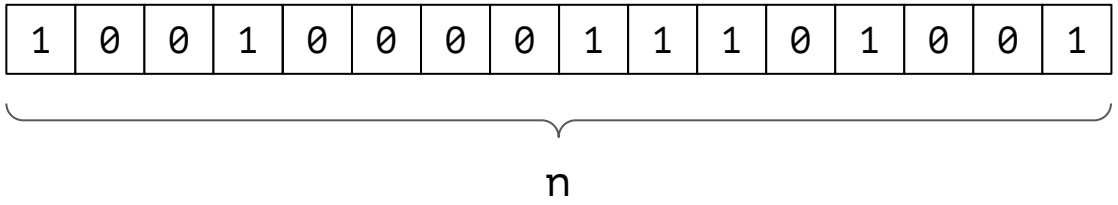
Фильтр Блума: анализ

Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).

Тогда чему равна вероятность того, что в i-ый бит не будет поставлена 1 при вызове некоторой хеш-функции h от некоторого значения v?



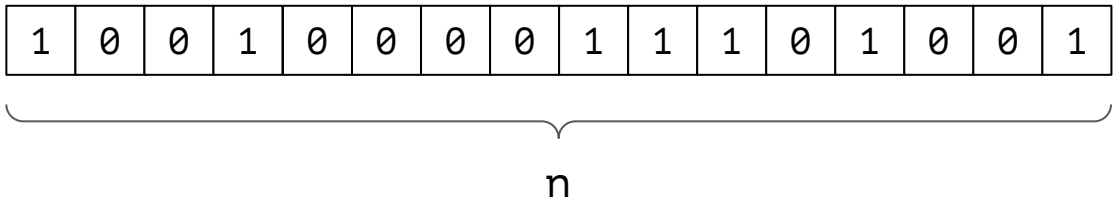
Фильтр Блума: анализ

Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).

Тогда чему равна вероятность того, что в i-ый бит не будет поставлена 1 при вызове некоторой хеш-функции h от некоторого значения v? $1 - \frac{1}{n}$



Фильтр Блума: анализ

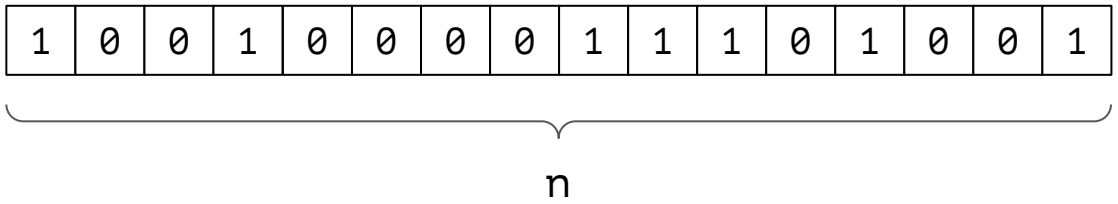
Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).

Тогда чему равна вероятность того, что в i-ый бит не будет поставлена 1 при вызове некоторой хеш-функции h от некоторого значения v? $1 - \frac{1}{n}$

Тогда чему равна вероятность того, что после работы всех k хеш-функций v в i-ый бит не будет поставлена 1?



Фильтр Блума: анализ

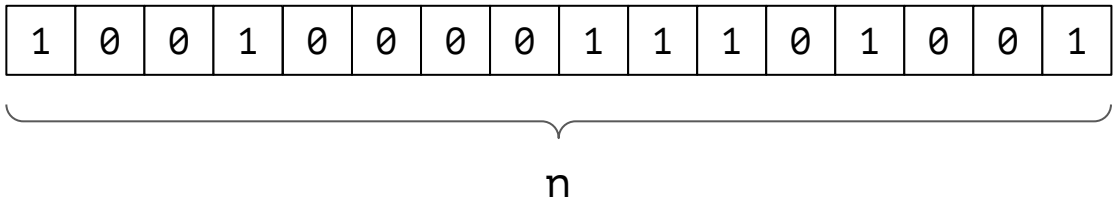
Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).

Тогда чему равна вероятность того, что в i-ый бит не будет поставлена 1 при вызове некоторой хеш-функции h от некоторого значения v? $1 - \frac{1}{n}$

Тогда чему равна вероятность того, что после работы всех k хеш-функций v в i-ый бит не будет поставлена 1? $(1 - \frac{1}{n})^k$



Фильтр Блума: анализ

Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).

Тогда чему равна вероятность того, что в i-ый бит не будет поставлена 1 при вызове некоторой хеш-функции h от некоторого значения v? $1 - \frac{1}{n}$

Тогда чему равна вероятность того, что после работы всех k хеш-функций v в i-ый бит не будет поставлена 1? $(1 - \frac{1}{n})^k$

Теперь пусть мы вставили уже |S| значений в фильтр Блума.
Какова вероятность того, что i-ый бит при это остался равен 0?

Фильтр Блума: анализ

Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).

Тогда чему равна вероятность того, что в i-ый бит не будет поставлена 1 при вызове некоторой хеш-функции h от некоторого значения v? $1 - \frac{1}{n}$

Тогда чему равна вероятность того, что после работы всех k хеш-функций v в i-ый бит не будет поставлена 1? $(1 - \frac{1}{n})^k$

Теперь пусть мы вставили уже |S| значений в фильтр Блума.
Какова вероятность того, что i-ый бит при это остался равен 0? $(1 - \frac{1}{n})^{k|S|}$

Фильтр Блума: анализ

Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).

Тогда чему равна вероятность того, что в i-ый бит не будет поставлена 1 при вызове некоторой хеш-функции h от некоторого значения v? $1 - \frac{1}{n}$

Тогда чему равна вероятность того, что после работы всех k хеш-функций v в i-ый бит не будет поставлена 1? $(1 - \frac{1}{n})^k$

Теперь пусть мы вставили уже |S| значений в фильтр Блума.
Какова вероятность того, что i-ый бит при это остался равен 0? $(1 - \frac{1}{n})^{k|S|}$

Инвертируем: вероятность того, что после вставки |S| элементов i-ый бит стал 1-ой?

Фильтр Блума: анализ

Пусть: $h_1, h_2, \ldots, h_k : values \to \{0, .., n-1\}$

1. Равномерно распределяют любые ключи по элементам нашего битсета (т. е. вероятность попасть в любой бит - 1/n)

2. Независимы (как друг от друга, так и для вызова разных ключей).

Тогда чему равна вероятность того, что в i-ый бит не будет поставлена 1 при вызове некоторой хеш-функции h от некоторого значения v? $1 - \frac{1}{n}$

Тогда чему равна вероятность того, что после работы всех k хеш-функций v в i-ый бит не будет поставлена 1? $(1 - \frac{1}{n})^k$

Теперь пусть мы вставили уже |S| значений в фильтр Блума. Какова вероятность того, что i-ый бит при это остался равен 0? $(1 - \frac{1}{n})^{k|S|}$

Инвертируем: вероятность того, что после вставки |S| элементов i-ый бит стал 1-ой? $1 - (1 - \frac{1}{n})^{k|S|}$

Фильтр Блума: анализ

Поиссследуем выражение $(1 - \frac{1}{n})^{k|S|}$

Фильтр Блума: анализ

Поиссследуем выражение $(1 - \frac{1}{n})^{k|S|}$

$$(1 - \frac{1}{n})^{k|S|} = ((1 - \frac{1}{n})^{n})^{\frac{k|S|}{n}}$$

Фильтр Блума: анализ

Поиссследуем выражение $(1 - \frac{1}{n})^{k|S|}$

$$(1 - \frac{1}{n})^{k|S|} = ((1 - \frac{1}{n})^{n})^{\frac{k|S|}{n}}$$

$\lim\limits_{x \to \infty} (1 - \frac{1}{x})^{x} = e^{-1}$ — следствие из второго замечательного предела

Фильтр Блума: анализ

Поиссследуем выражение $(1 - \frac{1}{n})^{k|S|}$

$$(1 - \frac{1}{n})^{k|S|} = ((1 - \frac{1}{n})^{n})^{\frac{k|S|}{n}} \approx (e^{-1})^{\frac{k|S|}{n}}$$

при достаточно больших n

$\lim_{x \to \infty} (1 - \frac{1}{x})^{x} = e^{-1}$ — следствие из второго замечательного предела

Фильтр Блума: анализ

Поиссследуем выражение $(1 - \frac{1}{n})^{k|S|}$

$$(1 - \frac{1}{n})^{k|S|} = ((1 - \frac{1}{n})^{n})^{\frac{k|S|}{n}} \approx (e^{-1})^{\frac{k|S|}{n}} = e^{-\frac{k|S|}{n}}$$

при достаточно больших n

$\lim_{x \to \infty} (1 - \frac{1}{x})^{x} = e^{-1}$ — следствие из второго замечательного предела

Фильтр Блума: анализ

Пусть теперь $x \notin S$

Что значит, что произошло ложноположительное срабатывание фильтра Блума для элемента x?

Фильтр Блума: анализ

Пусть теперь $x \notin S$

Что значит, что произошло ложноположительное срабатывание фильтра Блума для элемента x? Нужно, чтобы все k битов, в которые его отображают хеш-функции уже были заняты.

Фильтр Блума: анализ

Пусть теперь $x \notin S$

Что значит, что произошло ложноположительное срабатывание фильтра Блума для элемента x? Нужно, чтобы все k битов, в которые его отображают хеш-функции уже были заняты.

Шанс того, что 1 бит выставлен (после вставки |S| элементов) $\approx 1 - e^{-\frac{k|S|}{n}}$

Шанс того, что все k битов выставлены?

Фильтр Блума: анализ

Пусть теперь $x \notin S$

Что значит, что произошло ложноположительное срабатывание фильтра Блума для элемента x? Нужно, чтобы все k битов, в которые его отображают хеш-функции уже были заняты.

Шанс того, что 1 бит выставлен (после вставки |S| элементов) $\approx 1 - e^{-\frac{k|S|}{n}}$

Шанс того, что все k битов выставлены? $\approx \left(1 - e^{-\frac{k|S|}{n}}\right)^k$

Фильтр Блума: анализ

Пусть теперь $x \notin S$

Что значит, что произошло ложноположительное срабатывание фильтра Блума для элемента x? Нужно, чтобы все k битов, в которые его отображают хеш-функции уже были заняты.

Шанс того, что 1 бит выставлен (после вставки |S| элементов) $\approx 1 - e^{-\frac{k|S|}{n}}$

Шанс того, что все k битов выставлены? $\approx \left(1 - e^{-\frac{k|S|}{n}}\right)^k$

Теперь пусть $b = \frac{n}{|S|}$ фиксированная величина (кол-во битов на объект)

Фильтр Блума: анализ

Пусть теперь $x \notin S$

Что значит, что произошло ложноположительное срабатывание фильтра Блума для элемента x? Нужно, чтобы все k битов, в которые его отображают хеш-функции уже были заняты.

Шанс того, что 1 бит выставлен (после вставки |S| элементов) $\approx 1 - e^{-\frac{k|S|}{n}}$

Шанс того, что все k битов выставлены? $\approx \left(1 - e^{-\frac{k|S|}{n}}\right)^k$

Теперь пусть $b = \frac{n}{|S|}$ фиксированная величина (кол-во битов на объект)

Тогда $\left(1 - e^{-\frac{k}{b}}\right)^k$ достигает минимума в точке $k = ln(2) * b$

А вероятность ошибки будет: $\epsilon = \left(\frac{1}{2}\right)^{ln(2)*b}$

Фильтр Блума: анализ

Теперь пусть $b = \frac{n}{|S|}$ фиксированная величина (кол-во битов на объект)

Тогда $(1 - e^{-\frac{k}{b}})^k$ достигает минимума в точке $k = ln(2) * b$

А вероятность ошибки будет: $\epsilon = (\frac{1}{2})^{ln(2)*b}$

Вспоминаем: n - сколько битов используем, |S| - сколько объектов добавляли, k - сколько хеш-функций взяли.

Фильтр Блума: анализ

Теперь пусть $b = \frac{n}{|S|}$ фиксированная величина (кол-во битов на объект)

Тогда $(1 - e^{-\frac{k}{b}})^k$ достигает минимума в точке $k = ln(2) * b$

А вероятность ошибки будет: $\epsilon = (\frac{1}{2})^{ln(2)*b}$

Вспоминаем: n - сколько битов используем, |S| - сколько объектов добавляли, k - сколько хеш-функций взяли.

Допустим: поддерживаем b равное 8, т.е. используем на каждый объект всего 8 битов.

Фильтр Блума: анализ

Теперь пусть $b = \frac{n}{|S|}$ фиксированная величина (кол-во битов на объект)

Тогда $(1 - e^{-\frac{k}{b}})^k$ достигает минимума в точке $k = ln(2) * b$

А вероятность ошибки будет: $\epsilon = (\frac{1}{2})^{ln(2)*b}$

Вспоминаем: n - сколько битов используем, |S| - сколько объектов добавляли, k - сколько хеш-функций взяли.

Допустим: поддерживаем b равное 8, т.е. используем на каждый объект всего 8 битов.

Тогда $k = ln(2) * b \approx 0.69 * 8 < 6$

И $\epsilon \approx (\frac{1}{2})^{0.69*8} \approx 0.02$

Фильтр Блума: анализ

Теперь пусть $b = \frac{n}{|S|}$ фиксированная величина (кол-во битов на объект)

Тогда $(1 - e^{-\frac{k}{b}})^k$ достигает минимума в точке $k = ln(2) * b$

А вероятность ошибки будет: $\epsilon = (\frac{1}{2})^{ln(2)*b}$

Вспоминаем: n - сколько битов используем, |S| - сколько объектов добавляли, k - сколько хеш-функций взяли.

Допустим: поддерживаем b равное 8, т.е. используем на каждый объект всего 8 битов.

Тогда $k = ln(2) * b \approx 0.69 * 8 < 6$

И $\epsilon \approx (\frac{1}{2})^{0.69*8} \approx 0.02$

Т.е. при 8 битах на объект и 6 хеш-функциях получаем ошибку всего 2%!

Фильтр Блума: выводы

Как обычно работают с фильтром Блума:

1. Оцениваете, сколько элементов максимально будет в фильтре
2. Решаете, какая вероятность ошибки вас устроит

Фильтр Блума: выводы

Как обычно работают с фильтром Блума:

1. Оцениваете, сколько элементов максимально будет в фильтре
2. Решаете, какая вероятность ошибки вас устроит
3. Используя эти две величины получаете размер битсета и количество хеш-функций

Фильтр Блума: выводы

Как обычно работают с фильтром Блума:

1. Оцениваете, сколько элементов максимально будет в фильтре
2. Решаете, какая вероятность ошибки вас устроит
3. Используя эти две величины получаете размер битсета и количество хеш-функций
4. Получаете структуру данных с гарантированными свойствами

Мини-задача #28 (2 балла)

Реализовать фильтр Блума для IP-адресов (через битсет).

Для конфигурации структуры данных на вход подается прогнозируемое общее количество запросов и желаемая вероятность ошибки.

В качестве хеш-функций использовать случайные функции из универсального семейства хеш-функций, описанного в главе об универсальном хешировании.

Мини-задача #28 (2 балла)

Реализовать фильтр Блума для IP-адресов (через битсет).

Для конфигурации структуры данных на вход подается прогнозируемое общее количество запросов и желаемая вероятность ошибки.

В качестве хеш-функций использовать случайные функции из универсального семейства хеш-функций, описанного в главе об универсальном хешировании.

*понятно, что гарантии на их свойства слабые, но для примера будем использовать их.

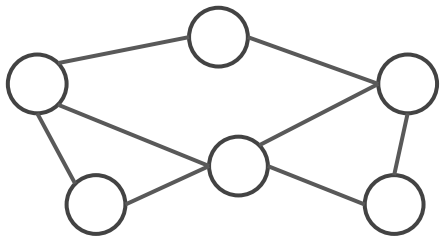
Что еще можно решить, применив рандомизированный алгоритм?

Что еще можно решить, применив рандомизированный алгоритм?

Много чего! Например, задачи на графы.

Задача о минимальном разрезе графа

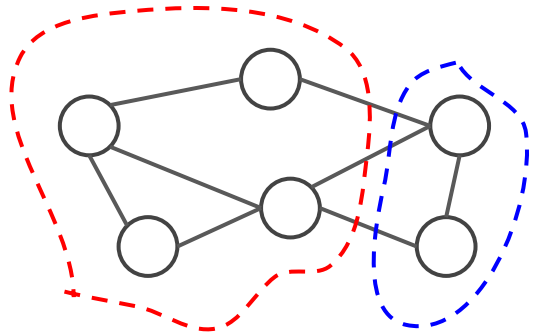
Пусть есть неориентированный граф $G = (V, E)$



Задача о минимальном разрезе графа

Пусть есть неориентированный граф $G = (V, E)$

Разрез графа - разбиение V на два непересекающихся непустых множества A и B.

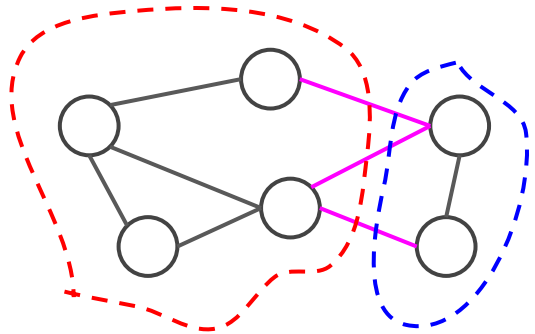


Задача о минимальном разрезе графа

Пусть есть неориентированный граф $G = (V, E)$

Разрез графа - разбиение V на два непересекающихся непустых множества A и B.

Пересекающие ребра разреза (A,B) - ребра, начинающиеся в A и заканчивающиеся в B.



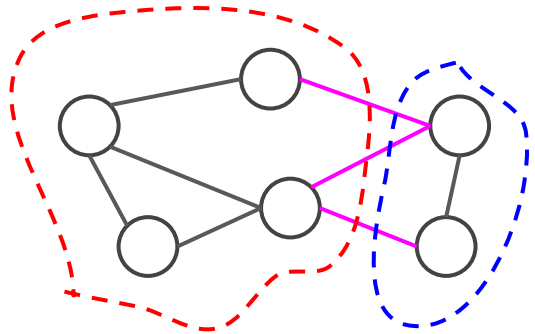
Задача о минимальном разрезе графа

Пусть есть неориентированный граф $G = (V, E)$

Разрез графа - разбиение V на два непересекающихся непустых множества A и B.

Пересекающие ребра разреза (A,B) - ребра, начинающиеся в A и заканчивающиеся в B.

Минимальный разрез в графе - разрез с минимальным количеством пересекающих ребер.



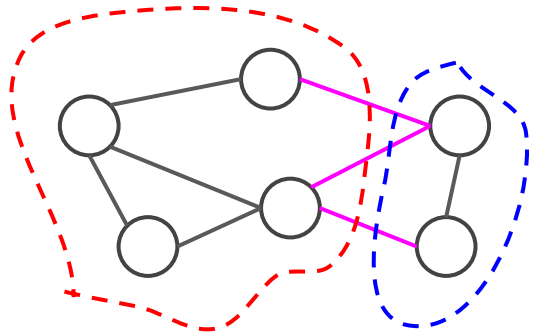
Задача о минимальном разрезе графа

Пусть есть неориентированный граф $G = (V, E)$

Разрез графа - разбиение V на два непересекающихся непустых множества A и B.

Пересекающие ребра разреза (A,B) - ребра, начинающиеся в A и заканчивающиеся в B.

Минимальный разрез в графе - разрез с минимальным количеством пересекающих ребер.



Это минимальный разрез?

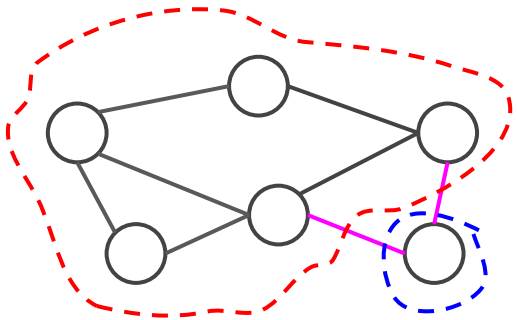
Задача о минимальном разрезе графа

Пусть есть неориентированный граф $G = (V, E)$

Разрез графа - разбиение V на два непересекающихся непустых множества A и B.

Пересекающие ребра разреза (A,B) - ребра, начинающиеся в A и заканчивающиеся в B.

Минимальный разрез в графе - разрез с минимальным количеством пересекающих ребер.



Это минимальный разрез?

Нет, а вот один из минимальных.

Задача о минимальном разрезе графа

Пусть есть неориентированный граф $G = (V, E)$

Разрез графа - разбиение V на два непересекающихся непустых множества A и B.

Пересекающие ребра разреза (A,B) - ребра, начинающиеся в A и заканчивающиеся в B.

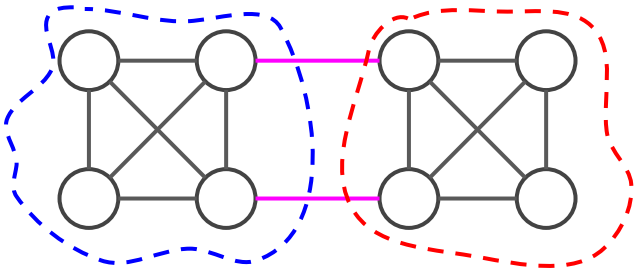
Минимальный разрез в графе - разрез с минимальным количеством пересекающих ребер.

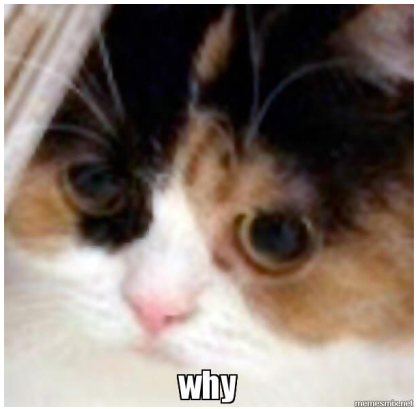
Задача о минимальном разрезе: по заданному неориентированному графу (возможно с кратными ребрами) найти его минимальный разрез.

Задача о минимальном разрезе графа

Задача о минимальном разрезе: по заданному неориентированному графу (возможно с кратными ребрами) найти его минимальный разрез.

А зачем?



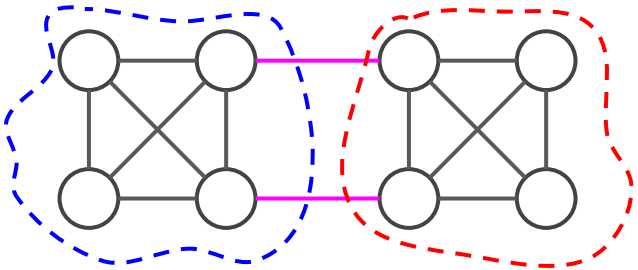


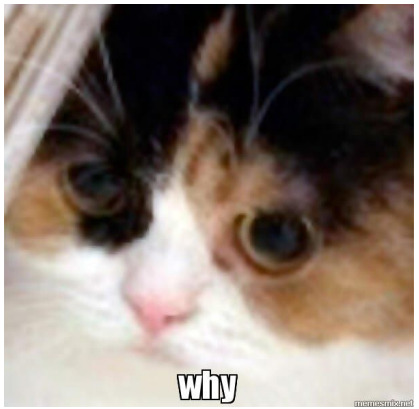
Задача о минимальном разрезе графа

Задача о минимальном разрезе: по заданному неориентированному графу (возможно с кратными ребрами) найти его минимальный разрез.

А зачем?

1. Искать слабые места в транспортной инфраструктуре (Бердское шоссе, например)





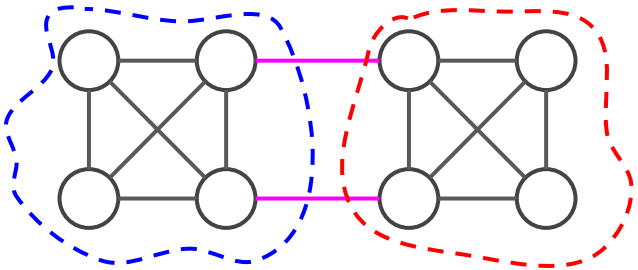
Задача о минимальном разрезе графа

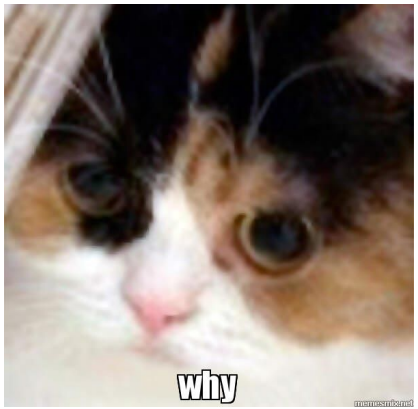
Задача о минимальном разрезе: по заданному неориентированному графу (возможно с кратными ребрами) найти его минимальный разрез.

А зачем?

1. Искать слабые места в транспортной инфраструктуре (Бердское шоссе, например)

2. Искать кластеры в социальных сетях



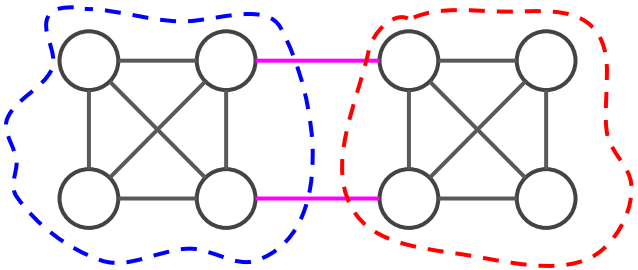


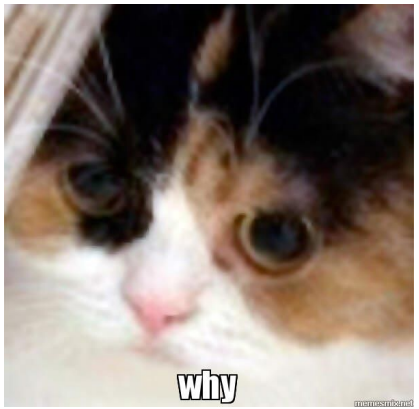
Задача о минимальном разрезе графа

Задача о минимальном разрезе: по заданному неориентированному графу (возможно с кратными ребрами) найти его минимальный разрез.

А зачем?

1. Искать слабые места в транспортной инфраструктуре (Бердское шоссе, например)
2. Искать кластеры в социальных сетях
3. Искать одинаковые объекты на изображениях

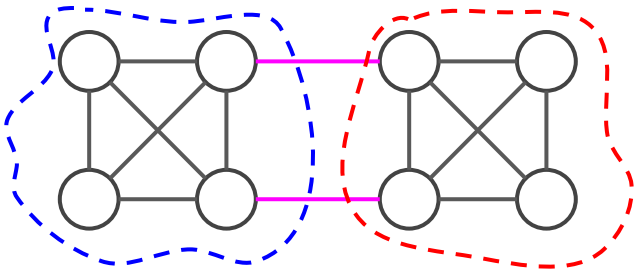




Задача о минимальном разрезе графа

Задача о минимальном разрезе: по заданному неориентированному графу (возможно с кратными ребрами) найти его минимальный разрез.

А сколько таких разрезов вообще существует?



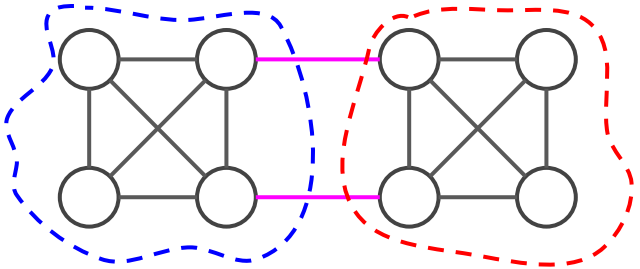
Задача о минимальном разрезе графа

Задача о минимальном разрезе: по заданному неориентированному графу (возможно с кратными ребрами) найти его минимальный разрез.

А сколько таких разрезов вообще существует?

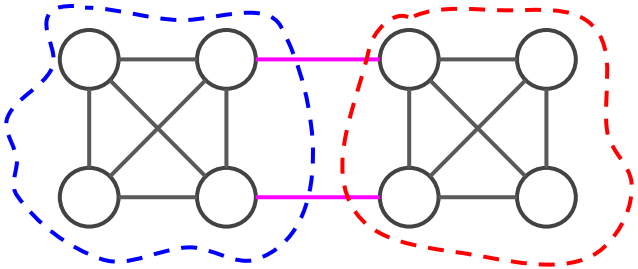
$2^{|V|} - 2$ т.к. это просто множество всех подмножеств вершин (кроме пустого и самого множества вершин)

Так что брут-форс - не вариант в достаточно больших графах.



Алгоритм Каргера

"Новый" алгоритм: опубликован в 1993 году!

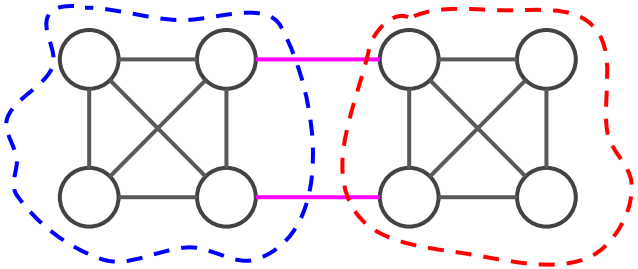


Алгоритм Каргера

"Новый" алгоритм: опубликован в 1993 году!

Алгоритм:

    Пока в графе больше двух вершин:



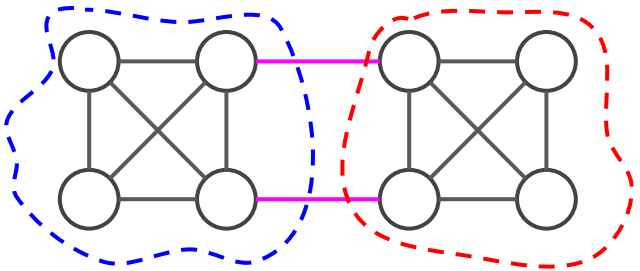
Алгоритм Каргера

"Новый" алгоритм: опубликован в 1993 году!

Алгоритм:

Пока в графе больше двух вершин:

1. Выбираем случайное ребро (u,v) из оставшихся,



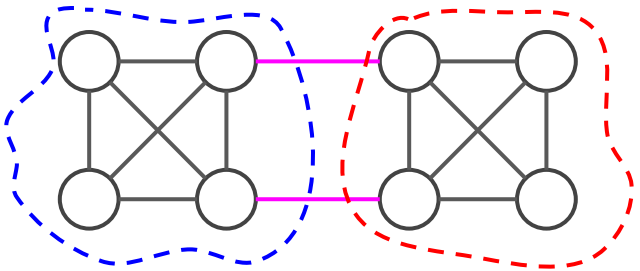
Алгоритм Каргера

"Новый" алгоритм: опубликован в 1993 году!

Алгоритм:

Пока в графе больше двух вершин:

1. Выбираем случайное ребро (u,v) из оставшихся,
2. Удаляем (стягиваем) это ребро, сливая u и v в одну вершину



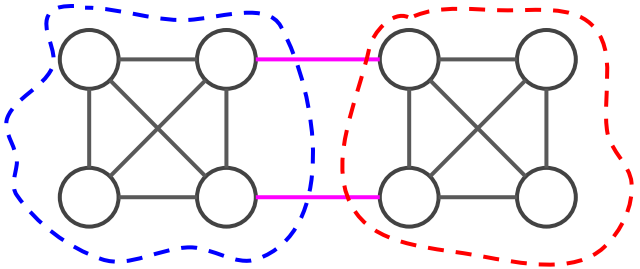
Алгоритм Каргера

"Новый" алгоритм: опубликован в 1993 году!

Алгоритм:

Пока в графе больше двух вершин:

1. Выбираем случайное ребро (u,v) из оставшихся,
2. Удаляем (стягиваем) это ребро, сливая u и v в одну вершину,
3. Если получились петли (ребра из вершины в себя) - удаляем.



Алгоритм Каргера

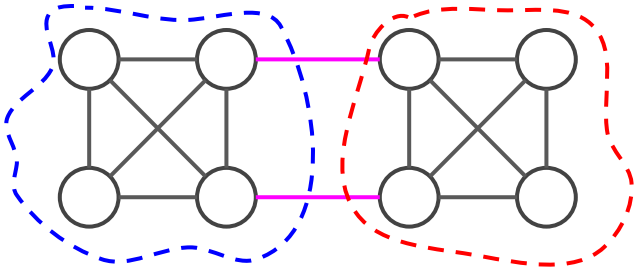
Да, это все*.

"Новый" алгоритм: опубликован в 1993 году!

Алгоритм:

Пока в графе больше двух вершин:

1. Выбираем случайное ребро (u,v) из оставшихся,
2. Удаляем (стягиваем) это ребро, сливая u и v в одну вершину,
3. Если получились петли (ребра из вершины в себя) - удаляем.



Алгоритм Каргера

Да, это все*.

"Новый" алгоритм: опубликован в 1993 году!

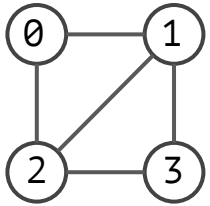
Алгоритм:

Пока в графе больше двух вершин:

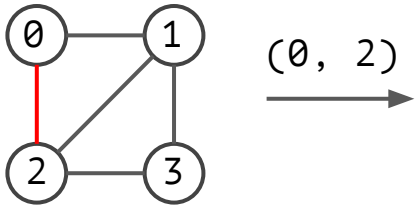
1. Выбираем случайное ребро (u,v) из оставшихся,
2. Удаляем (стягиваем) это ребро, сливая u и v в одну вершину,
3. Если получились петли (ребра из вершины в себя) - удаляем.

*Каждый раз при слиянии u и v запоминали, из каких старых новая получившаяся вершина состоит. Ответ - множества, из которых состоят последние две вершины.

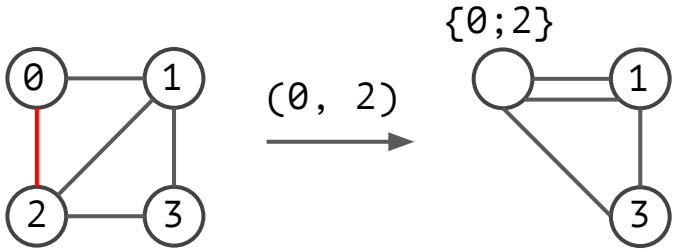
Алгоритм Каргера: пример



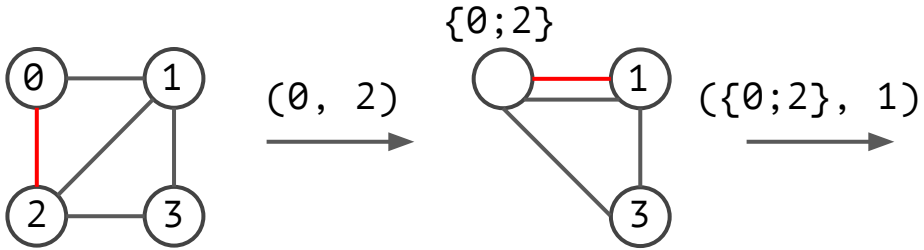
Алгоритм Каргера: пример



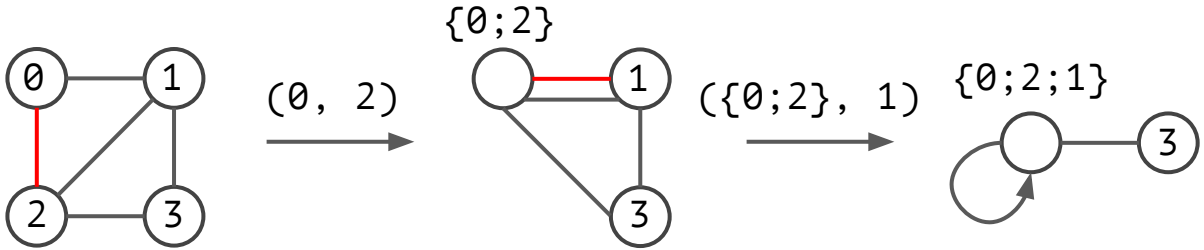
Алгоритм Каргера: пример



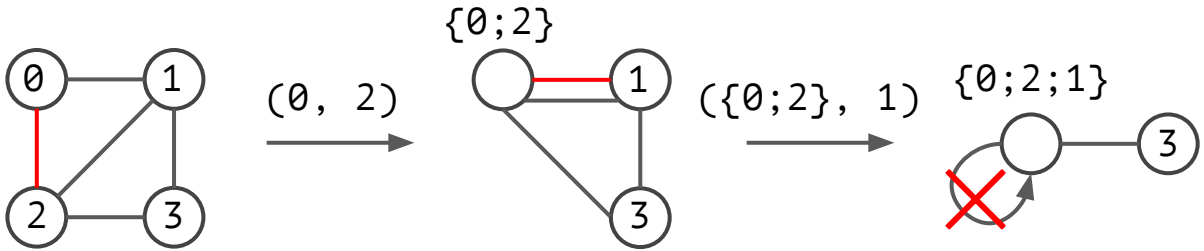
Алгоритм Каргера: пример



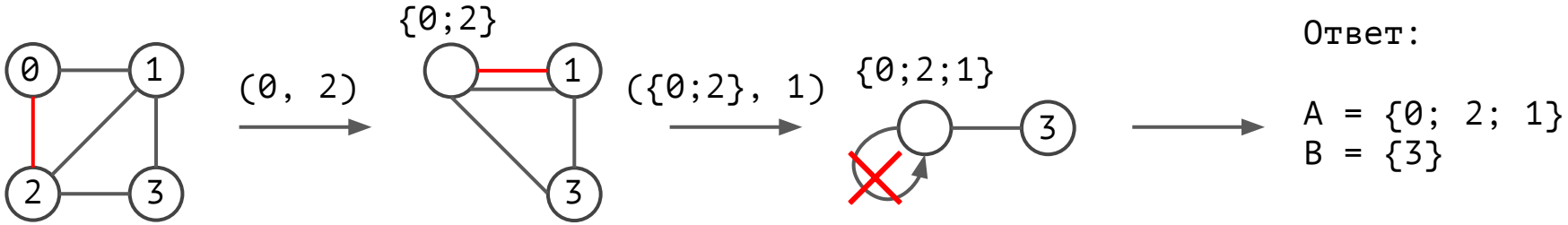
Алгоритм Каргера: пример



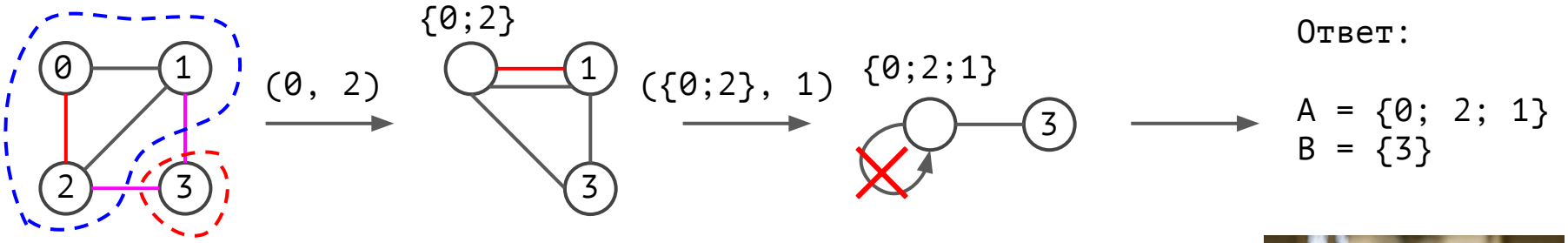
Алгоритм Каргера: пример



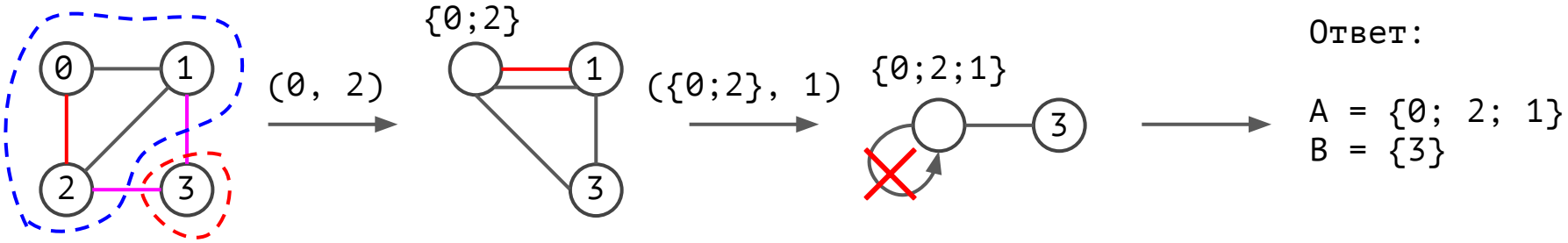
Алгоритм Каргера: пример



Алгоритм Каргера: пример

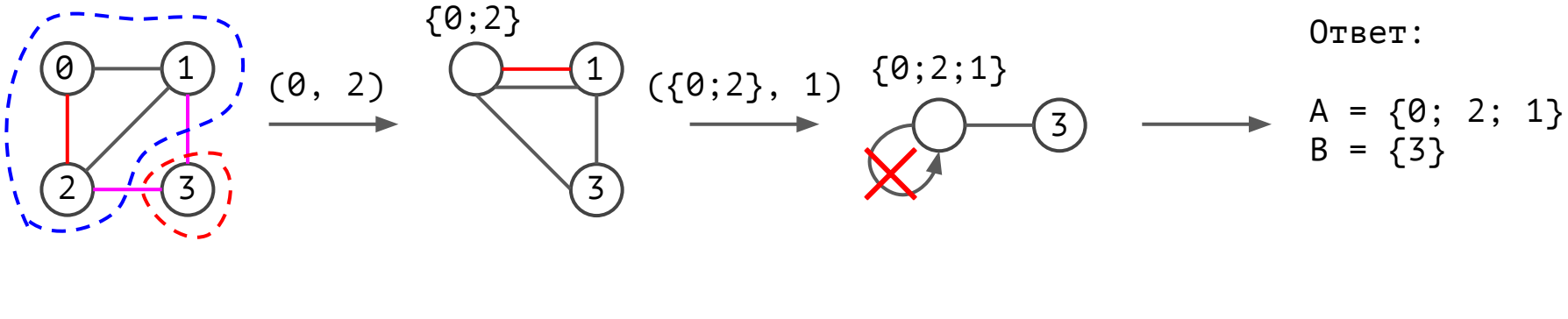


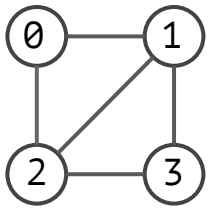
Алгоритм Каргера: пример



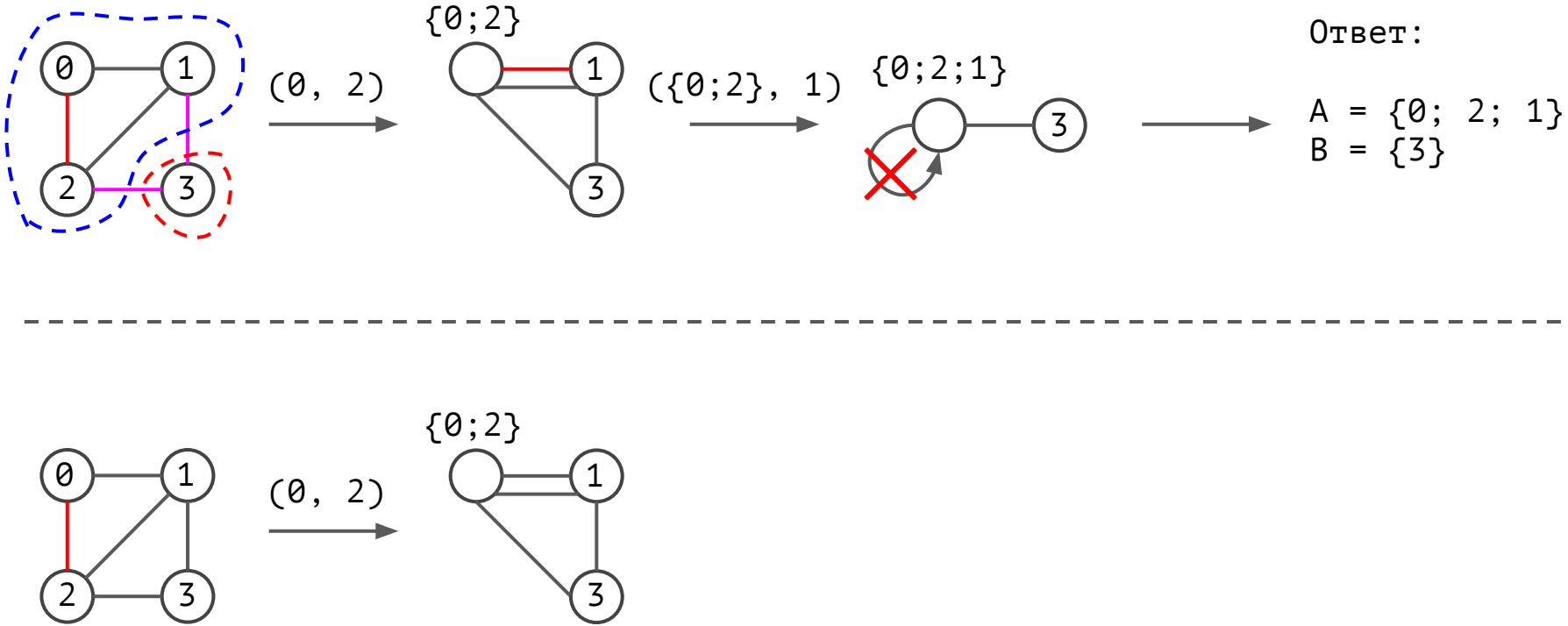
Попробуем еще раз!

Алгоритм Каргера: пример

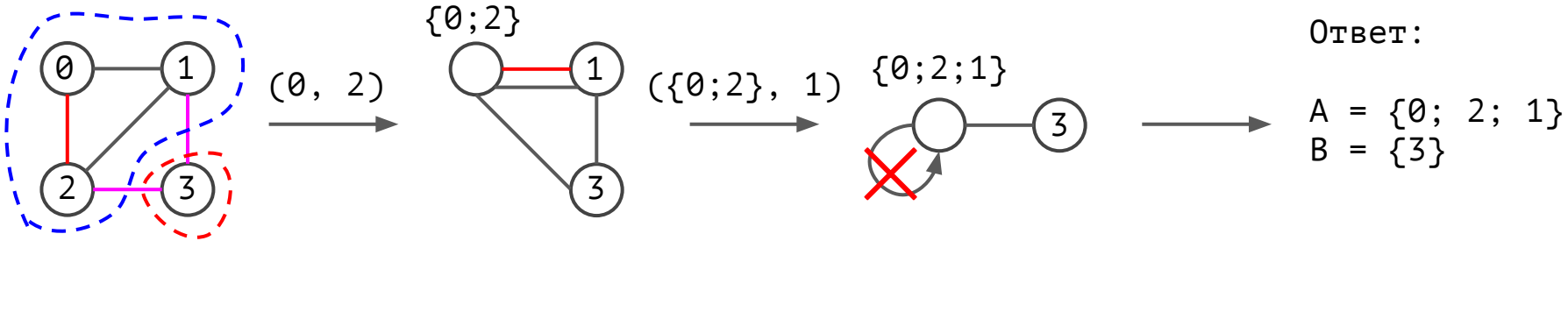


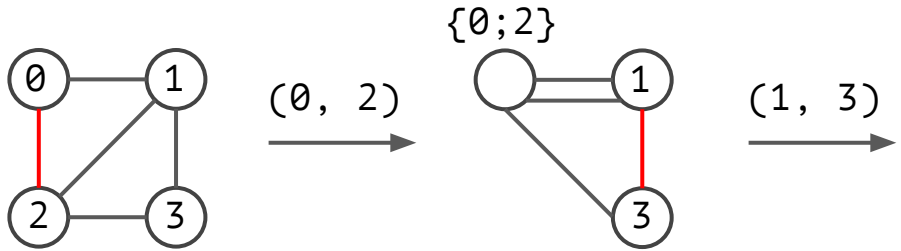


Алгоритм Каргера: пример



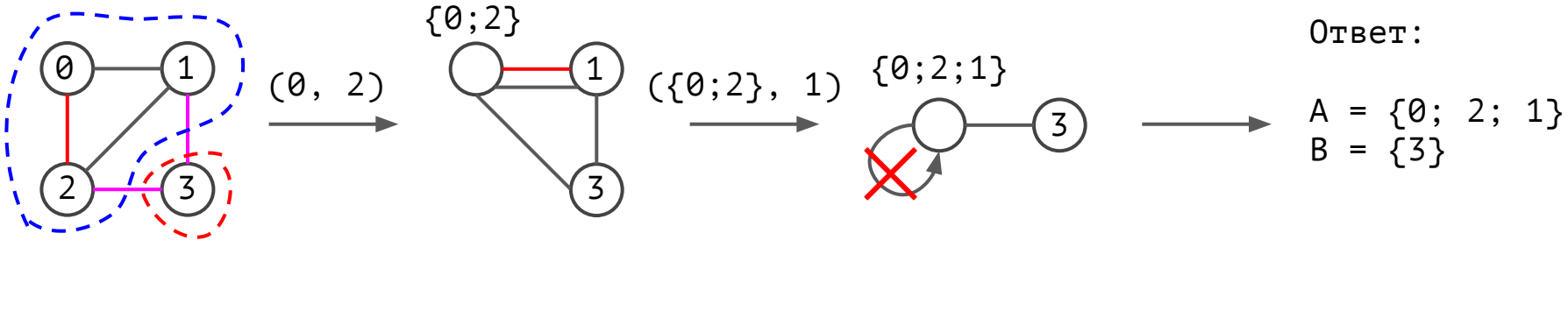
Алгоритм Каргера: пример

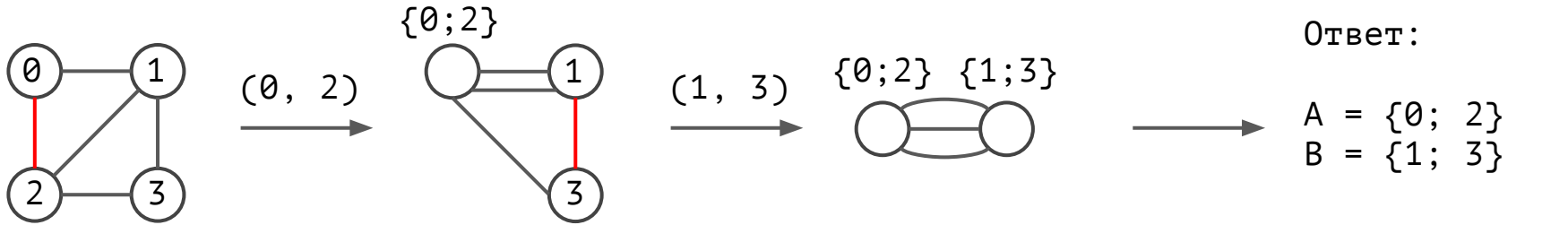




выбор ребра - случайный

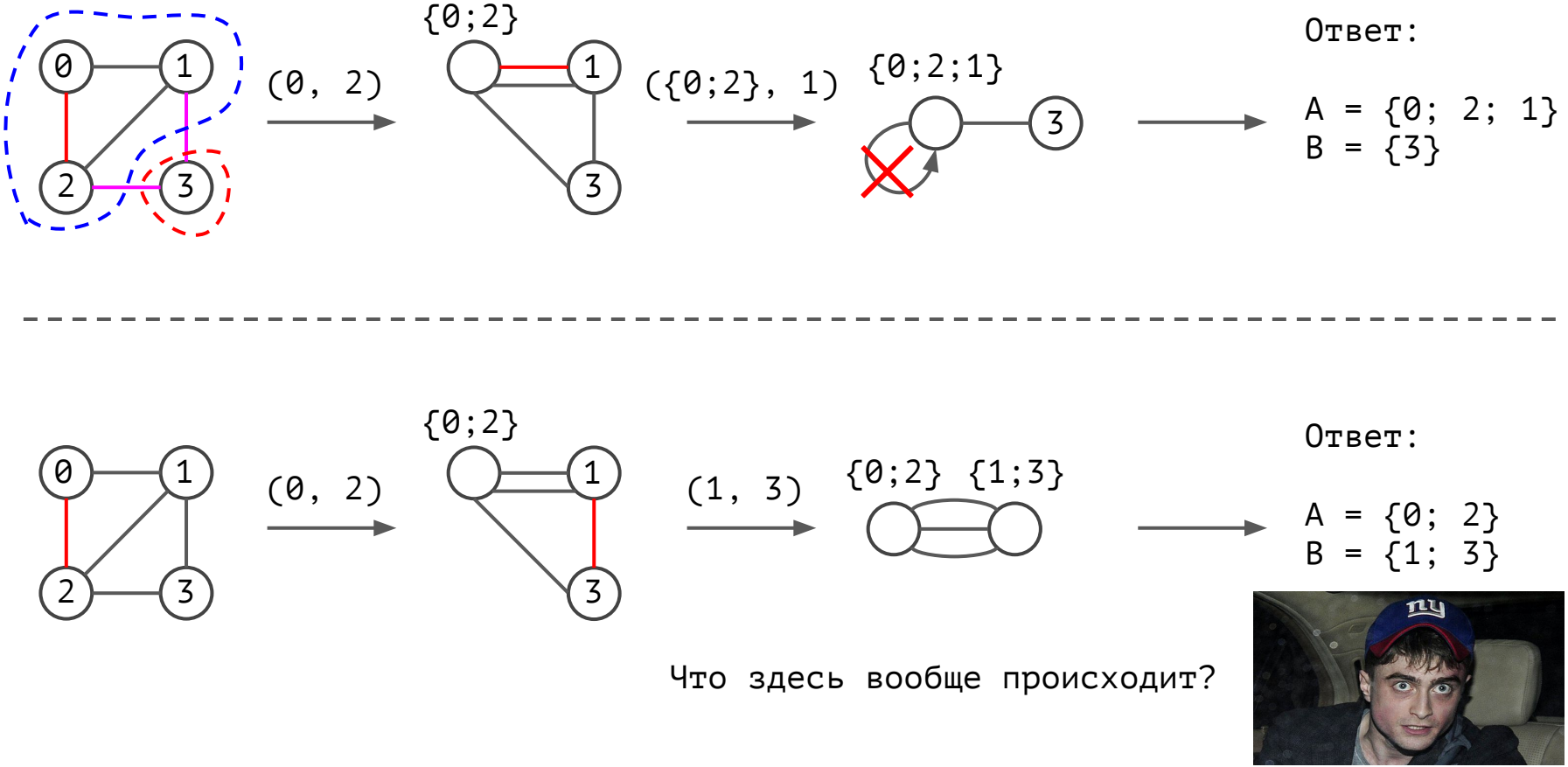
Алгоритм Каргера: пример





выбор ребра - случайный

Алгоритм Каргера: пример



Алгоритм Каргера: пример

Действительно, алгоритм Каргера может находить минимальные разрезы в графе.

Алгоритм Каргера: пример

Действительно, алгоритм Каргера может находить минимальные разрезы в графе. А может и не находить.

Алгоритм Каргера: пример

Действительно, алгоритм Каргера может находить минимальные разрезы в графе. А может и не находить.

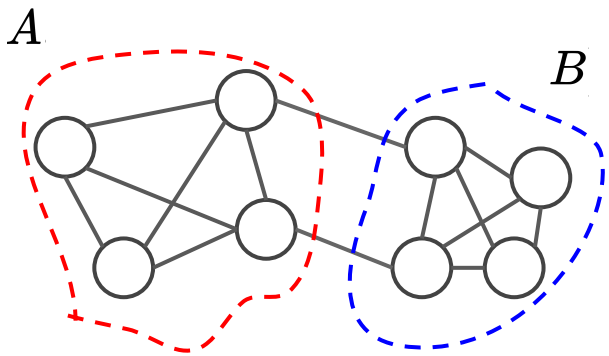
И весь вопрос в том, с какой вероятностью он найдет правильный ответ?

Алгоритм Каргера: пример

Действительно, алгоритм Каргера может находить минимальные разрезы в графе. А может и не находить.

И весь вопрос в том, с какой вероятностью он найдет правильный ответ?

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)

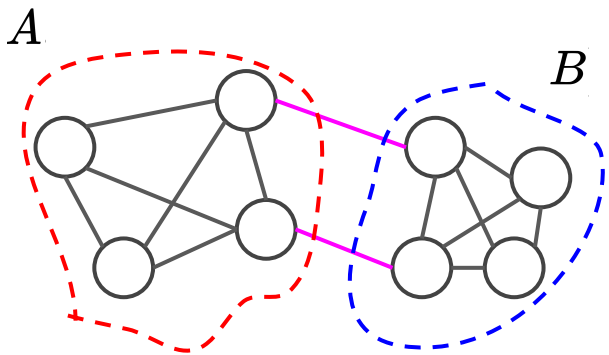


Алгоритм Каргера: пример

Действительно, алгоритм Каргера может находить минимальные разрезы в графе. А может и не находить.

И весь вопрос в том, с какой вероятностью он найдет правильный ответ?

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.



Алгоритм Каргера: пример

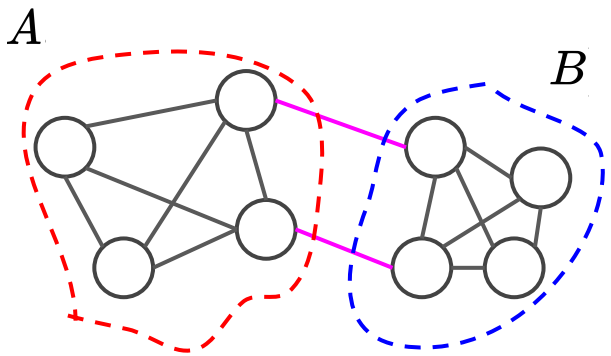
Действительно, алгоритм Каргера может находить минимальные разрезы в графе. А может и не находить.

И весь вопрос в том, с какой вероятностью он найдет правильный ответ?

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

Наблюдения:

1. Если мы хоть на одном шаге алгоритма взяли ребро из F - мы ошиблись, в ответе не получится (A,B).



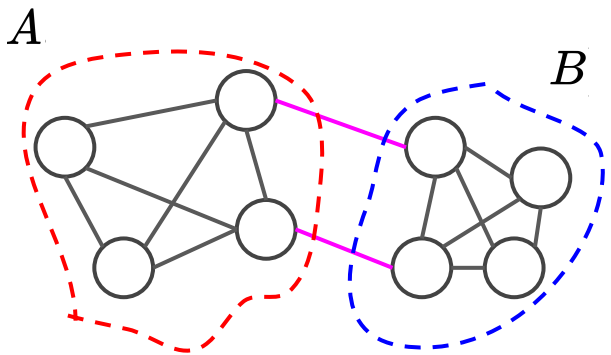
Алгоритм Каргера: пример

Действительно, алгоритм Каргера может находить минимальные разрезы в графе. А может и не находить.

И весь вопрос в том, с какой вероятностью он найдет правильный ответ?

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

Наблюдения:

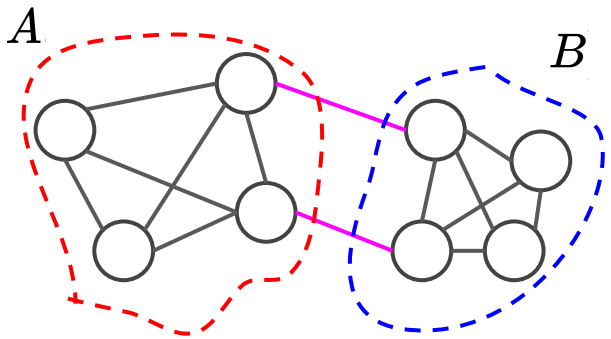


1. Если мы хоть на одном шаге алгоритма взяли ребро из F - мы ошиблись, в ответе не получится (A,B).

2. И наоборот: если на каждом шаге мы брали ребра из подграфа A или B => в ответе получим (A,B)

Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.



Наблюдения:

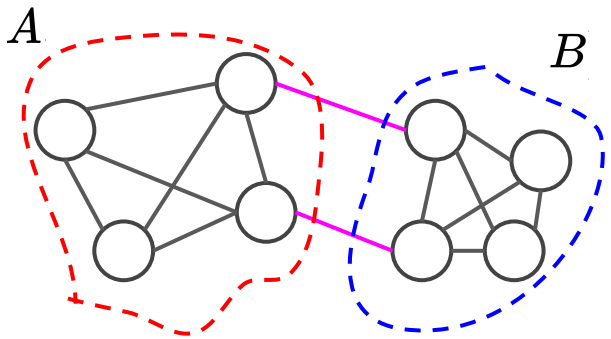
1. Если мы хоть на одном шаге алгоритма взяли ребро из F - мы ошиблись, в ответе не получится (A,B).

2. И наоборот: если на каждом шаге мы брали ребра из подграфа A или B => в ответе получим (A,B)

Тогда $Pr[\text{успех}] = Pr[\text{никогда не брали ребро из } F]$

Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.



Наблюдения:

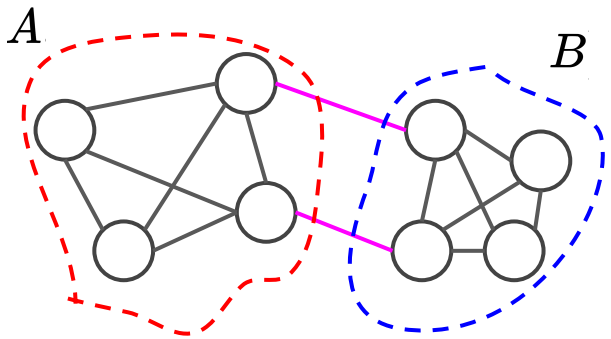
1. Если мы хоть на одном шаге алгоритма взяли ребро из F - мы ошиблись, в ответе не получится (A,B).

2. И наоборот: если на каждом шаге мы брали ребра из подграфа A или B => в ответе получим (A,B)

Тогда $Pr[\text{успех}] = Pr[\text{никогда не брали ребро из } F]$
Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.



Наблюдения:

1. Если мы хоть на одном шаге алгоритма взяли ребро из F - мы ошиблись, в ответе не получится (A,B).

2. И наоборот: если на каждом шаге мы брали ребра из подграфа A или B => в ответе получим (A,B)

Тогда $Pr[\text{успех}] = Pr[\text{никогда не брали ребро из } F]$
Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

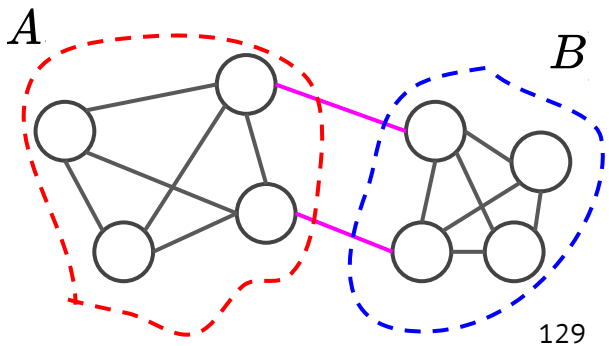
Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Начнем с простого, чем равно $Pr[S_1]$?



Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

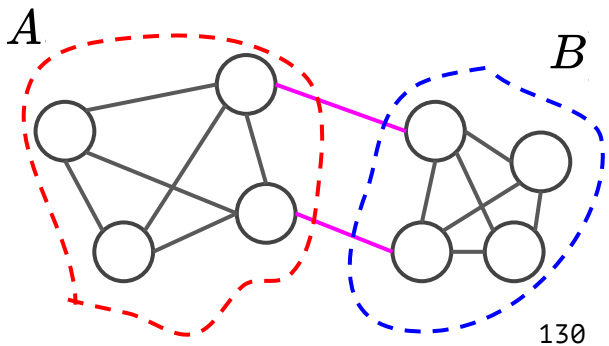
Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Начнем с простого, чем равно $Pr[S_1]$?

Пусть $|F| = k, |E| = m$

Тогда $Pr[S_1] = \frac{k}{m}$



Алгоритм Каргера: пример

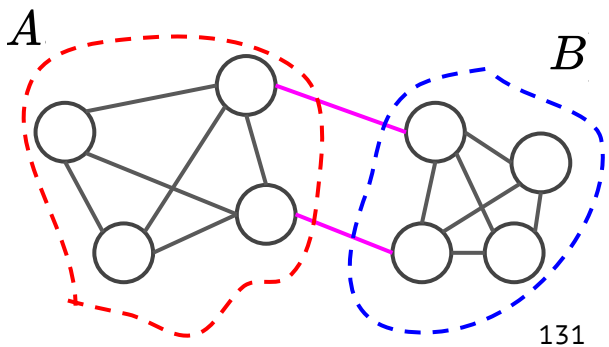
Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$

Заметим, что для любой вершины в графе
количество смежных вершин (степень) $\geq k$



Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

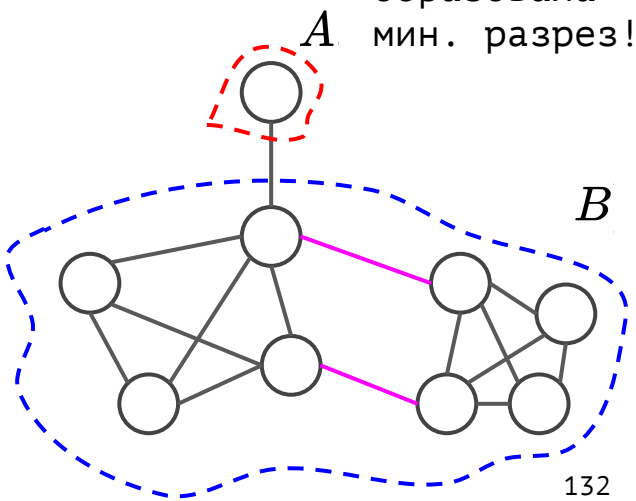
Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m,\ Pr[S_1] = \frac{k}{m}$

Заметим, что для любой вершины в графе
количество смежных вершин (степень) $\geq k$

Иначе бы она образовала мин. разрез!



Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

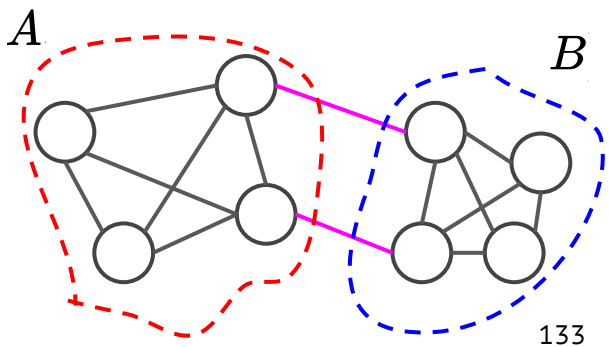
Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m,\ Pr[S_1] = \frac{k}{m}$

Заметим, что для любой вершины в графе
количество смежных вершин (степень) $\geq k$

А еще заметим, что сумма степеней всех вершин
$\sum_{v \in V} degree(v) = 2m$



Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

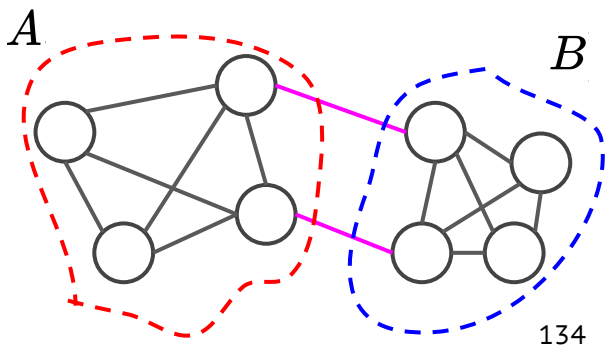
Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$

Заметим, что для любой вершины в графе
количество смежных вершин (степень) $\geq k$

А еще заметим, что сумма степеней всех вершин

$$\sum_{v \in V} degree(v) = 2m$$

$$\sum_{v \in V} degree(v) \geq k * n$$



Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

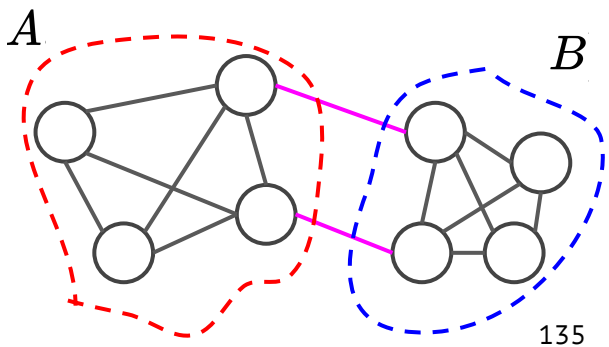
Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$

Заметим, что для любой вершины в графе
количество смежных вершин (степень) $\geq k$

А еще заметим, что сумма степеней всех вершин

$$\left.\begin{array}{l} \sum\limits_{v \in V} degree(v) = 2m \\ \sum\limits_{v \in V} degree(v) \geq k * n \end{array}\right\} \Rightarrow m \geq \frac{k*n}{2}$$



Алгоритм Каргера: пример

Зафиксируем граф $G = (V, E)$ и его минимальный разрез (A, B)
F - множество пересекающих ребер.

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

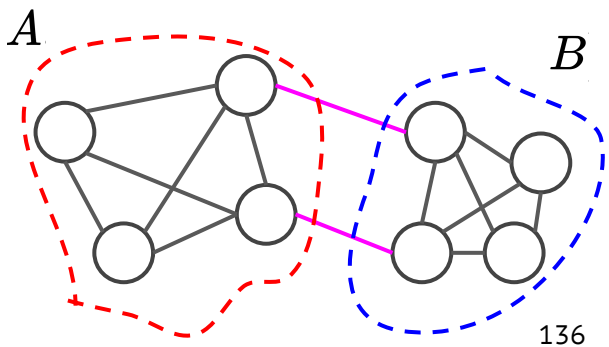
Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$

Заметим, что для любой вершины в графе
количество смежных вершин (степень) $\geq k$

А еще заметим, что сумма степеней всех вершин

$$\left.\begin{array}{l} \sum\limits_{v \in V} degree(v) = 2m \\ \sum\limits_{v \in V} degree(v) \geq k * n \end{array}\right\} \Rightarrow m \geq \frac{k*n}{2} \Rightarrow Pr[S_1] \leq \frac{2}{n}$$



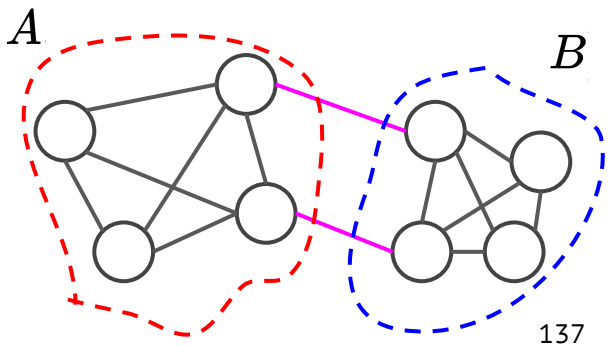
Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = ?$



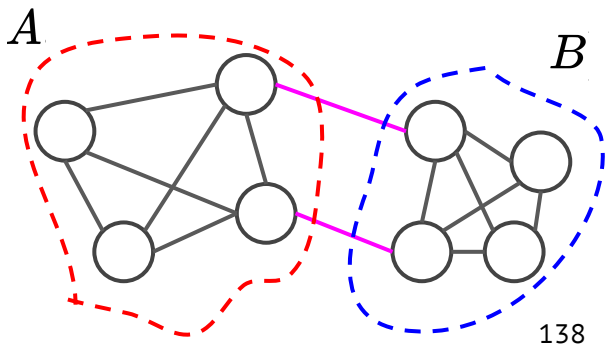
Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$



Алгоритм Каргера: вторая итерация

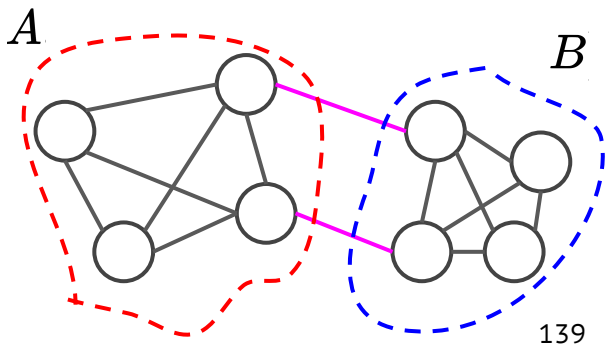
Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $\boxed{Pr[S_1] \leq \frac{2}{n}}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

При этом: $Pr[\neg S_1] \geq (1 - \frac{2}{n})$



Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

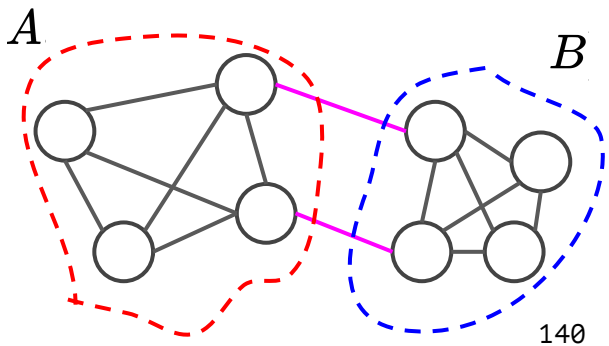
Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $\boxed{Pr[S_1] \leq \frac{2}{n}}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

При этом: $Pr[\neg S_1] \geq (1 - \frac{2}{n})$,

$$Pr[\neg S_2 | \neg S_1] = 1 - \frac{k}{\#(\text{оставшиеся ребра})}$$



Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

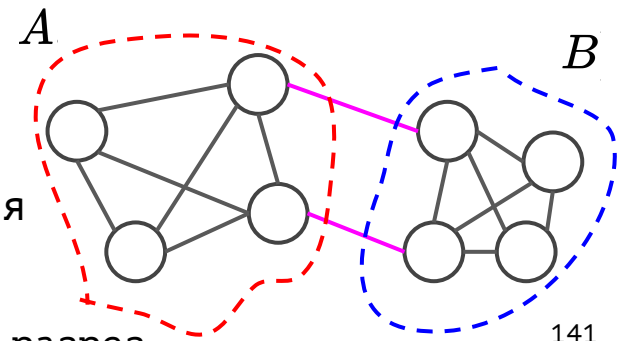
Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $\boxed{Pr[S_1] \leq \frac{2}{n}}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

При этом: $Pr[\neg S_1] \geq (1 - \frac{2}{n})$,

$$Pr[\neg S_2 | \neg S_1] = 1 - \frac{k}{\#(\text{оставшиеся ребра})}$$



Повторим трюк с первой итерации: в получившемся графе (в котором уже n-1 вершин) тоже верно:

$\#(\text{оставшиеся ребра}) \geq \frac{k(n-1)}{2}$ т.к. каждая вершина задает разрез

Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

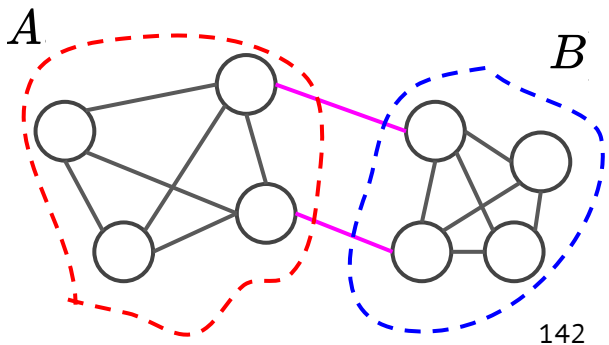
Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $\boxed{Pr[S_1] \leq \frac{2}{n}}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

При этом: $Pr[\neg S_1] \geq (1 - \frac{2}{n})$,

$Pr[\neg S_2 | \neg S_1] \geq (1 - \frac{2}{n-1})$



Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

При этом: $\left.\begin{array}{l} Pr[\neg S_1] \geq (1 - \frac{2}{n}), \\ Pr[\neg S_2 | \neg S_1] \geq (1 - \frac{2}{n-1}) \end{array}\right\} \Rightarrow Pr[\neg S_1 \cap \neg S_2] \geq (1 - \frac{2}{n})(1 - \frac{2}{n-1})$

Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

При этом: $\left.\begin{array}{l} Pr[\neg S_1] \geq (1 - \frac{2}{n}), \\ Pr[\neg S_2 | \neg S_1] \geq (1 - \frac{2}{n-1}) \end{array}\right\} \Rightarrow Pr[\neg S_1 \cap \neg S_2] \geq (1 - \frac{2}{n})(1 - \frac{2}{n-1})$

Продолжая рассуждения, получим: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}] =$

$= Pr[\neg S_1] Pr[\neg S_2 | \neg S_1] Pr[\neg S_3 | \neg S_2 \cap \neg S_1] \ldots Pr[\neg S_{n-2} | \neg S_{n-3} \cap \ldots \cap \neg S_1] \geq$

Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

При этом: $\left.\begin{array}{l} Pr[\neg S_1] \geq (1 - \frac{2}{n}), \\ Pr[\neg S_2 | \neg S_1] \geq (1 - \frac{2}{n-1}) \end{array}\right\} \Rightarrow Pr[\neg S_1 \cap \neg S_2] \geq (1 - \frac{2}{n})(1 - \frac{2}{n-1})$

Продолжая рассуждения, получим: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}] =$

$= Pr[\neg S_1] Pr[\neg S_2 | \neg S_1] Pr[\neg S_3 | \neg S_2 \cap \neg S_1] \ldots Pr[\neg S_{n-2} | \neg S_{n-3} \cap \ldots \cap \neg S_1] \geq$

$\geq (1 - \frac{2}{n})(1 - \frac{2}{n-1}) \ldots (1 - \frac{2}{n-(n-4)})(1 - \frac{2}{n-(n-3)})$

Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

Продолжая рассуждения, получим: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}] =$

$= Pr[\neg S_1] Pr[\neg S_2 | \neg S_1] Pr[\neg S_3 | \neg S_2 \cap \neg S_1] \ldots Pr[\neg S_{n-2} | \neg S_{n-3} \cap \ldots \cap \neg S_1] \geq$

$\geq (1 - \frac{2}{n})(1 - \frac{2}{n-1}) \ldots (1 - \frac{2}{n-(n-4)})(1 - \frac{2}{n-(n-3)}) =$

$\frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} * \ldots * \frac{2}{4} * \frac{1}{3} =$

Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

Продолжая рассуждения, получим: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}] =$

$= Pr[\neg S_1] Pr[\neg S_2 | \neg S_1] Pr[\neg S_3 | \neg S_2 \cap \neg S_1] \ldots Pr[\neg S_{n-2} | \neg S_{n-3} \cap \ldots \cap \neg S_1] \geq$

$\geq (1 - \frac{2}{n})(1 - \frac{2}{n-1}) \ldots (1 - \frac{2}{n-(n-4)})(1 - \frac{2}{n-(n-3)}) =$

$\frac{\cancel{n-2}}{n} * \frac{\cancel{n-3}}{n-1} * \frac{\cancel{n-4}}{\cancel{n-2}} * \ldots * \frac{2}{\cancel{4}} * \frac{1}{\cancel{3}} = \frac{2}{n(n-1)} \geq \frac{1}{n^2}$

Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

Продолжая рассуждения, получим: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}] =$

$= Pr[\neg S_1] Pr[\neg S_2 | \neg S_1] Pr[\neg S_3 | \neg S_2 \cap \neg S_1] \ldots Pr[\neg S_{n-2} | \neg S_{n-3} \cap \ldots \cap \neg S_1] \geq$

$\geq (1 - \frac{2}{n})(1 - \frac{2}{n-1}) \ldots (1 - \frac{2}{n-(n-4)})(1 - \frac{2}{n-(n-3)}) =$

$\frac{\cancel{n-2}}{n} * \frac{\cancel{n-3}}{n-1} * \frac{\cancel{n-4}}{\cancel{n-2}} * \ldots * \frac{2}{\cancel{4}} * \frac{1}{\cancel{3}} = \frac{2}{n(n-1)} \geq \frac{1}{n^2}$

т.е. в самом худшем случае алгоритм найдет мин. разрез с вероятностью всего то $\frac{1}{n^2}$

Алгоритм Каргера: вторая итерация

Пусть S_i - событие, что мы взяли ребро из F на i-ой итерации алгоритма.

Тогда нужно найти: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}]$, где $n = |V|$

Пусть $|F| = k, |E| = m$, $Pr[S_1] = \frac{k}{m}$, $Pr[S_1] \leq \frac{2}{n}$

Теперь вторая итерация: $Pr[\neg S_1 \cap \neg S_2] = Pr[\neg S_2 | \neg S_1] * Pr[\neg S_1]$

Продолжая рассуждения, получим: $Pr[\neg S_1 \cap \neg S_2 \cap \ldots \cap \neg S_{n-2}] =$

$= Pr[\neg S_1] Pr[\neg S_2 | \neg S_1] Pr[\neg S_3 | \neg S_2 \cap \neg S_1] \ldots Pr[\neg S_{n-2} | \neg S_{n-3} \cap \ldots \cap \neg S_1] \geq$

$\geq (1 - \frac{2}{n})(1 - \frac{2}{n-1}) \ldots (1 - \frac{2}{n-(n-4)})(1 - \frac{2}{n-(n-3)}) =$

$\frac{\cancel{n-2}}{n} * \frac{\cancel{n-3}}{n-1} * \frac{\cancel{n-4}}{\cancel{n-2}} * \ldots * \frac{2}{\cancel{4}} * \frac{1}{\cancel{3}} = \frac{2}{n(n-1)} \geq \frac{1}{n^2}$

т.е. в самом худшем случае алгоритм найдет мин. разрез с вероятностью всего то $\frac{1}{n^2}$

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

$$Pr[\text{за } N \text{ запусков не нашли}] =$$

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

$$Pr[\text{за } N \text{ запусков не нашли}] = Pr[\neg T_1 \cap \neg T_2 \cap \ldots \cap \neg T_N]$$

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

$$Pr[\text{за } N \text{ запусков не нашли}] = Pr[\neg T_1 \cap \neg T_2 \cap \ldots \cap \neg T_N] = \prod_{i=1}^{N} Pr[\neg T_i] \leq (1 - \tfrac{1}{n^2})^N$$

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

$$Pr[\text{за } N \text{ запусков не нашли}] = Pr[\neg T_1 \cap \neg T_2 \cap \ldots \cap \neg T_N] = \prod_{i=1}^{N} Pr[\neg T_i] \leq (1 - \tfrac{1}{n^2})^N$$

Теперь можем поподбирать N! Например, возьмем $N = n^2$

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

$$Pr[\text{за } N \text{ запусков не нашли}] = Pr[\neg T_1 \cap \neg T_2 \cap \ldots \cap \neg T_N] = \prod_{i=1}^{N} Pr[\neg T_i] \leq (1 - \tfrac{1}{n^2})^N$$

Теперь можем поподбирать N! Например, возьмем $N = n^2$
Снова вспоминаем про замечательный предел, получаем

$$Pr[\text{не нашли за } n^2 \text{ запусков}] \leq (e^{-\frac{1}{n^2}})^{n^2} = \frac{1}{e}$$

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

$$Pr[\text{за } N \text{ запусков не нашли}] = Pr[\neg T_1 \cap \neg T_2 \cap \ldots \cap \neg T_N] = \prod_{i=1}^{N} Pr[\neg T_i] \leq (1 - \tfrac{1}{n^2})^N$$

Теперь можем поподбирать N! Например, возьмем $N = n^2$
Снова вспоминаем про замечательный предел, получаем

$$Pr[\text{не нашли за } n^2 \text{ запусков}] \leq (e^{-\frac{1}{n^2}})^{n^2} = \frac{1}{e} \approx 37\%!$$

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

$$Pr[\text{за } N \text{ запусков не нашли}] = Pr[\neg T_1 \cap \neg T_2 \cap \ldots \cap \neg T_N] = \prod_{i=1}^{N} Pr[\neg T_i] \leq (1 - \tfrac{1}{n^2})^N$$

Теперь можем поподбирать N! Например, возьмем $N = n^2$
Снова вспоминаем про замечательный предел, получаем

$Pr[\text{не нашли за } n^2 \text{ запусков}] \leq (e^{-\frac{1}{n^2}})^{n^2} = \frac{1}{e} \approx 37\%!$ Сложность при этом $\Omega(n^2 * m)$

Алгоритм Каргера: развязка

В самом худшем случае алгоритм найдет мин. разрез с вероятностью $\frac{1}{n^2}$.

Но не будем отчаиваться!
А что, если позапускать алгоритм несколько раз?

Пусть запускаем наш алгоритм N раз.

Введем T_i - событие, что мы нашли минимальный разрез на i-ом шаге.
Все T_i независимы друг от друга по построению.

$$Pr[\text{за } N \text{ запусков не нашли}] = Pr[\neg T_1 \cap \neg T_2 \cap \ldots \cap \neg T_N] = \prod_{i=1}^{N} Pr[\neg T_i] \leq (1 - \tfrac{1}{n^2})^N$$

Теперь можем поподбирать N! Например, возьмем $N = n^2 * ln(n)$
Снова вспоминаем про замечательный предел, получаем

$Pr[\text{не нашли за } n^2 * log(n) \text{ запусков}] \leq (\frac{1}{e})^{ln(n)} = \frac{1}{n}$; Сложность: $\Omega(n^2 * m * ln(n))$

Алгоритм Каргера: выводы

- Алгоритм Каргера позволяет находить минимальный разрез с заданной точностью (варьируется количеством запусков).

Алгоритм Каргера: выводы

- Алгоритм Каргера позволяет находить минимальный разрез с заданной точностью (варьируется количеством запусков).

- При этом в угоду точности приходится жертвовать производительностью

Алгоритм Каргера: выводы

- Алгоритм Каргера позволяет находить минимальный разрез с заданной точностью (варьируется количеством запусков).
- При этом в угоду точности приходится жертвовать производительностью
- Есть улучшения, например, алгоритм Каргера-Штайна, который дает производительность $O(n^2 * ln(n))$, при вероятности ошибки $\Omega(\frac{1}{ln(n)})$

Takeaways

- Рандомизированные алгоритмы зачастую контринтуитивны, но дают потрясающее преимущество на практике.

Takeaways

- Рандомизированные алгоритмы зачастую контринтуитивны, но дают потрясающее преимущество на практике.

- Фильтр Блума и алгоритм Каргена стоит использовать там, где ошибки не являются фатальными.