Система непересекающихся множеств (англ. Disjoint Set Union)

В некоторых задачах необходимо хранить **разбиение какого-то набора уникальных объектов на непересекающиеся динамические множества.**

Предположим, что набор объектов, для которого выполнялось разбиение на непересекающиеся множества, — целые числа от **1** до n.

$$\{1, \mathbf{2}, 3, 4, 8\}, \{5, \mathbf{6}\}, \{\mathbf{7}\}$$

В каждом множестве выделен один из его элементов, который называют **представителем** (англ. *representative*), который будет определять данное множество. В литературе часто вместо слова представитель говорят **лидер**.

Пусть изначально каждый объект находится в собственном одноэлементном множестве.

БАЗОВЫЕ ОПЕРАЦИИ	$\boldsymbol{FindSet}(\boldsymbol{x})$ — выдать указатель на представителя множества, которому принадлежит элемент $\boldsymbol{x}$; $\boldsymbol{Union}(\boldsymbol{x}, \boldsymbol{y})$ — объединить два непересекающихся множества, которые содержат элементы $\boldsymbol{x}$ и $\boldsymbol{y}$.

Структуру данных, поддерживающую такой интерфейс, называют **системой непересекающихся множеств** (**СНМ**)
(англ. *disjoint set union,* **DSU**).

Реализация DSU

1. на массиве
2. на связном списке с указателем на представителя
3. с помощью семейства корневых деревьев

1. **Реализация DSU с использованием структуры данных массив** предполагает, что элемент массива $\boldsymbol{array[i]}$ содержит представителя множества, которому принадлежит элемент $\boldsymbol{i}$. Нумерация в массиве `array` начинается с 1.

Для системы непересекающихся множеств: $\{1, 2, 3, 4, 8\}, \{5, 6\}, \{7\}$.

Массив будет иметь следующий вид:

	1	2	3	4	5	6	7	8
array	**2**	**2**	**2**	**2**	**6**	**6**	**7**	**2**

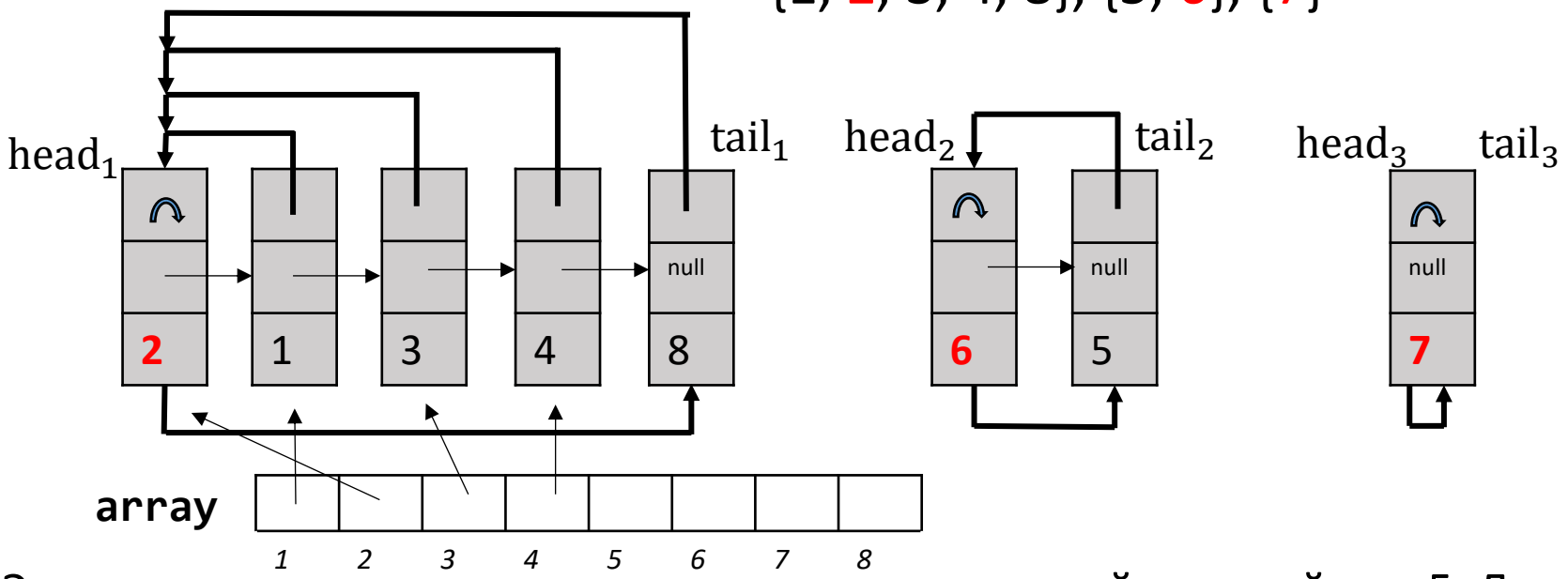
БАЗОВЫЕ ОПЕРАЦИИ

$\boldsymbol{FindSet(x)} - \quad \mathbf{O(1)}$

$\boldsymbol{Union(x, y)} - \quad \mathbf{O(n)}$

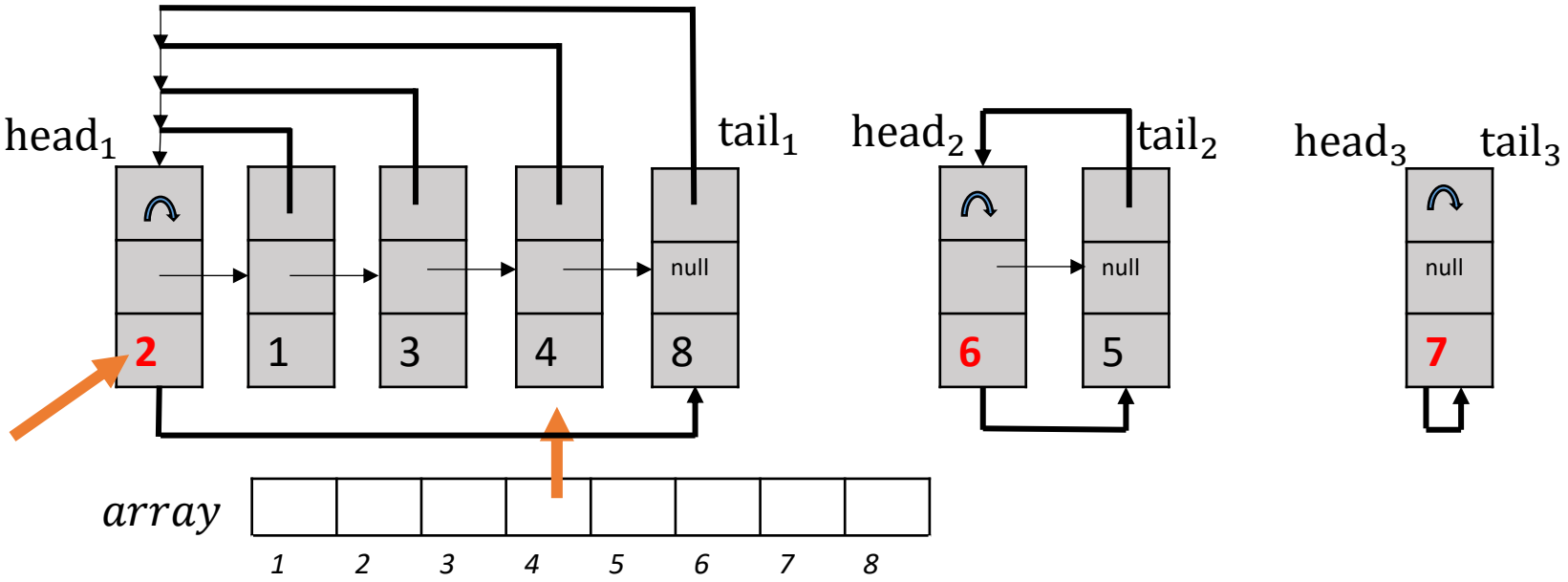
2. Реализация на связном списке с указателем на представителя

{1, 2, 3, 4, 8}, {5, 6}, {7}



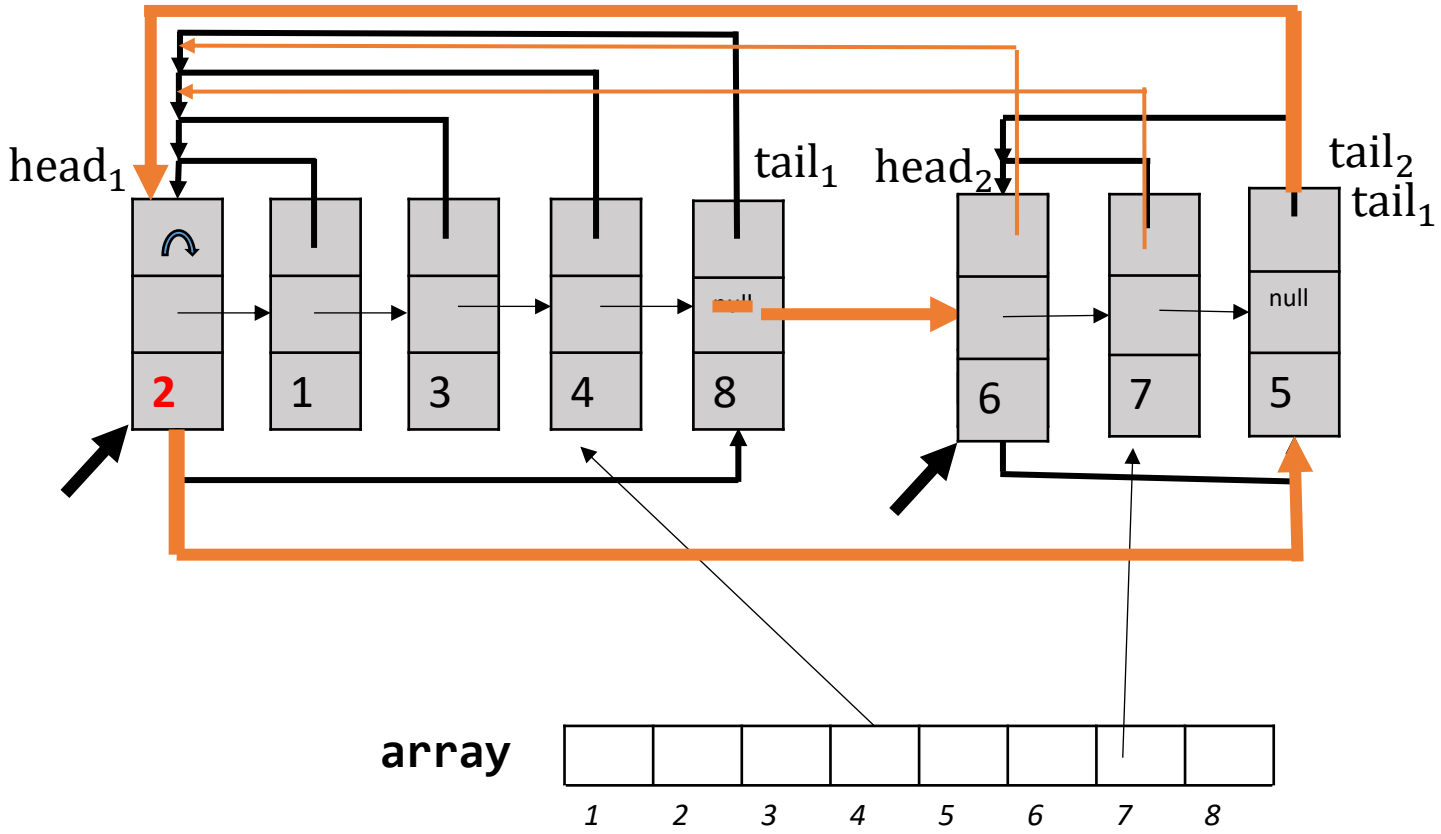
1. Элементы каждого множества связаны в отдельный связный список.
2. Представителем множества является первый элемент списка .
3. Каждый элемент списка содержит:
   - указатель на представителя;
   - указатель на следующий за ним элемент.
4. Для каждого списка поддерживают два указателя: на первый и последний элементы списка.
5. Дополнительно: массив **`array`**, где $array[i]$ - указатель на элемент `i` в связном списке.

$$FindSet(4)$$



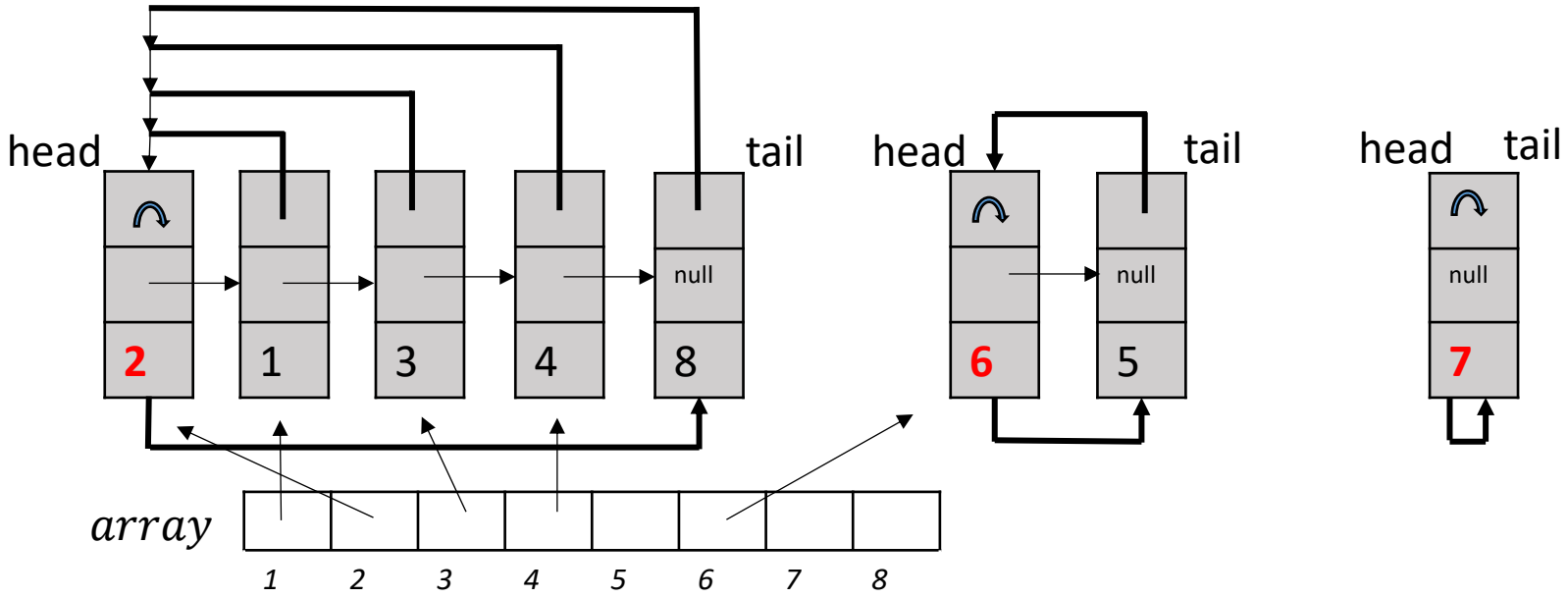
$$FindSet(x) - \mathbf{O(1)}$$

$Union(4, 7)$



$Union(x, y)$ — $O(n)$

{1, 2, 3, 4, 8}, {5, 6}, {7}

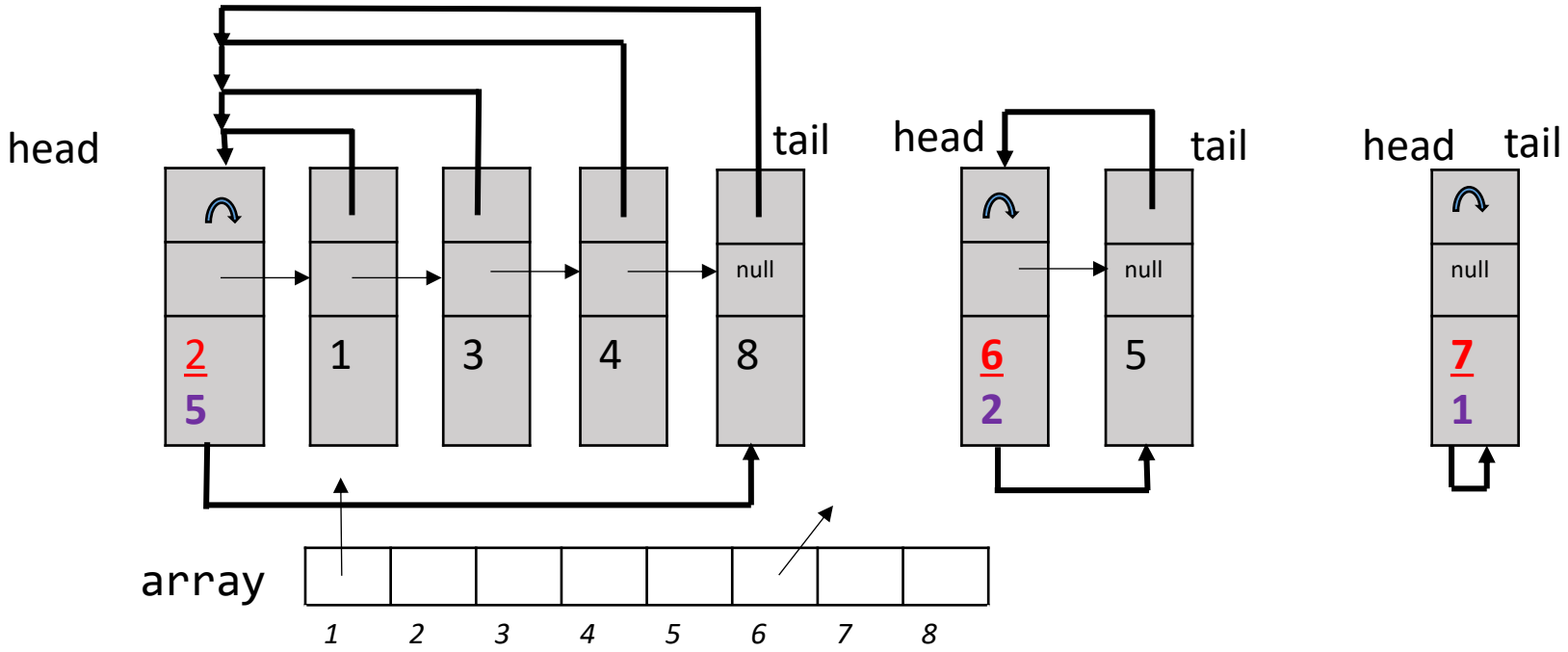


$$\boldsymbol{FindSet(x)} \ - \quad \mathbf{O(1)}$$

$$\boldsymbol{Union(x, y)} \ - \quad \mathbf{O(n)}$$

Правило **меньшее к большему:**

при выполнении операции **`Union(x,y)`** ссылки на нового представителя изменяются у всех элементов меньшего множества.



Правило **меньшее к большему**

при выполнении операции **`Union(x,y)`** ссылки на нового представителя изменяются у всех элементов меньшего множества.

Теорема

Пусть есть $\boldsymbol{n}$ одноэлементных множеств, тогда последовательность из $\boldsymbol{m}$ операций $\boldsymbol{FindSet(x)}$ и $\boldsymbol{Union(x, y)}$ потребует времени $\mathbf{O}(\boldsymbol{m} + \boldsymbol{n}\log \boldsymbol{n})$.

Доказательство

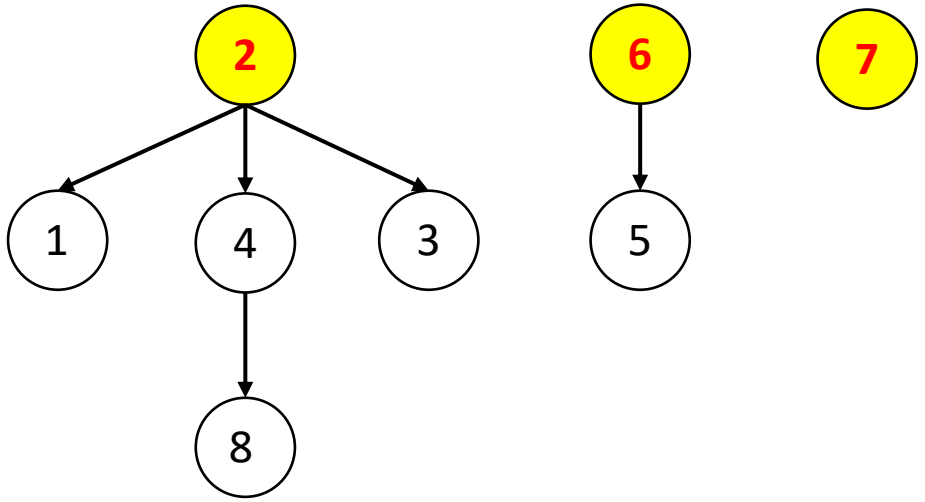
Каждый раз, когда какой-то элемент перемещается из одного множества в другое с изменением представителя, размер множества, содержащего этот элемент, **увеличивается не менее чем вдвое**.

Так как множество может вырасти лишь до размера n, то каждый элемент может подвергнуться перемещению не более чем $\log_2 n$ раз.

3. Реализация DSU с помощью семейства корневых деревьев

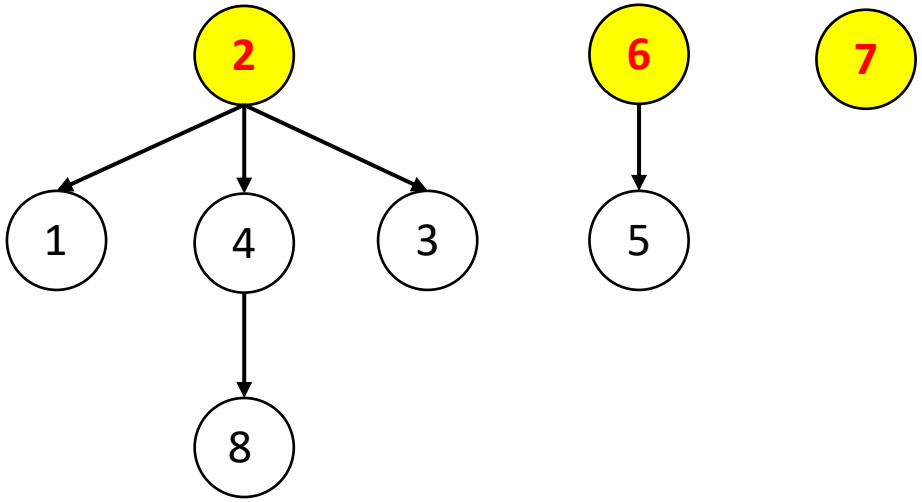
1) Каждому множеству поставим в соответствие своё корневое дерево.
2) Каждой вершине дерева соответствует ровно один элемент множества.
3) Корень дерева содержит представителя множества.

{1, 2, 3, 4, 8}
{5, 6}
{7}



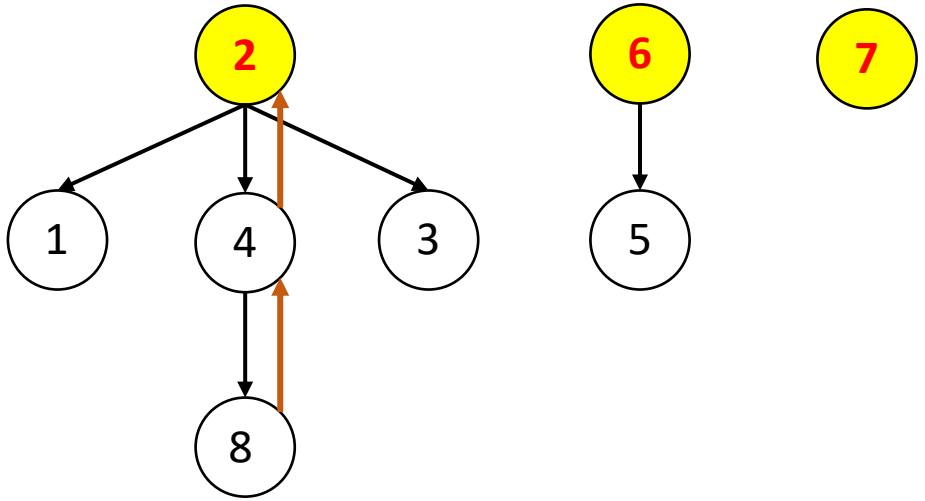
В памяти компьютера семейство корневых деревьев хранится в каноническом виде (массив **parent):**

- для каждой вершины дерева **i** указан её предок **parent[i];**
- для корня дерева верно равенство **i==parent[i]**.



i	1	2	3	4	5	6	7	8
Parent[i]	2	2	2	2	6	6	7	4

$FindSet\ (8)$

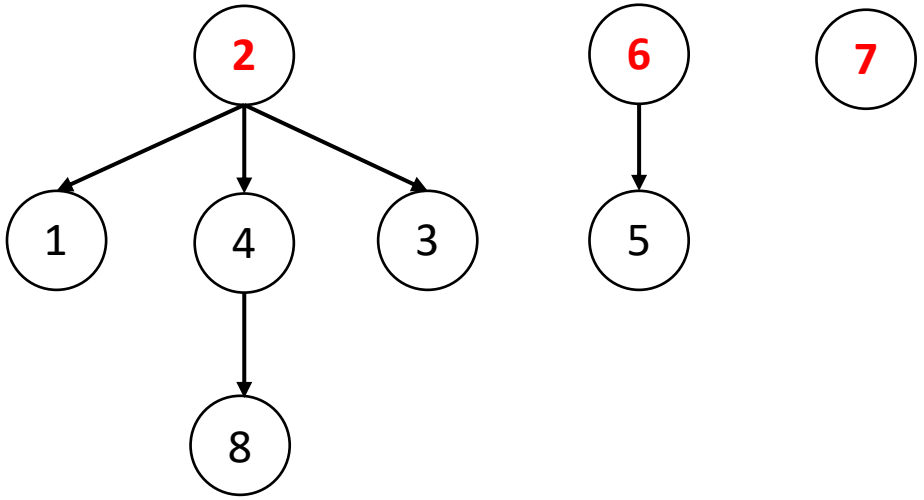


i	1	2	3	4	5	6	7	8
parent[i]	2	2	2	2	6	6	7	4

$FindSet(x)$ — $O(h)$

$Union(x, y)$

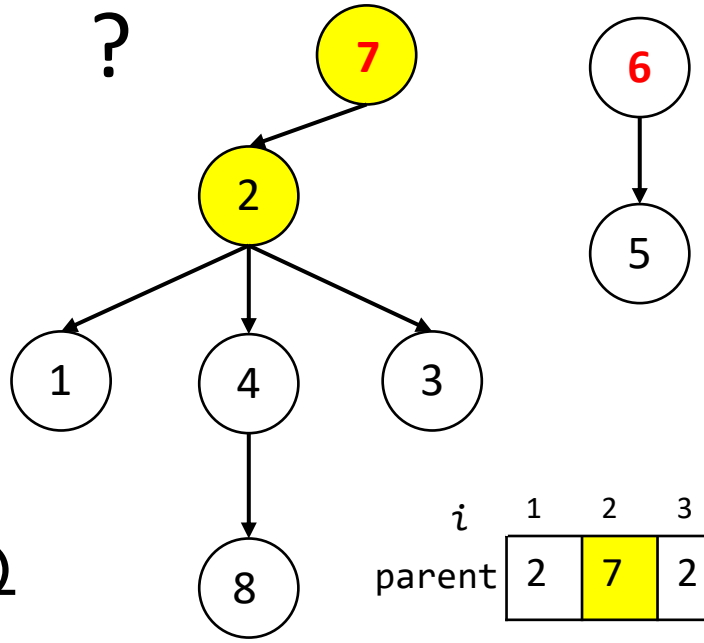
- ✓ определить представителей множеств, которым принадлежат x и y;
- ✓ представителя одного из множеств сделать сыном представителя другого множества;



i	1	2	3	4	5	6	7	8
parent	2	2	2	2	6	6	7	4

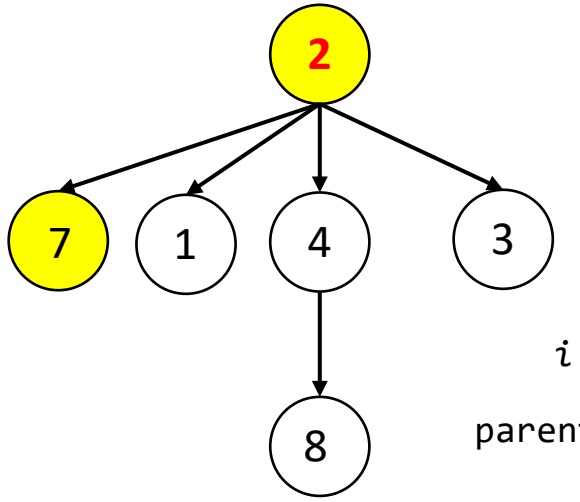
Union(7,8)

?



i	1	2	3	4	5	6	7	8
parent	2	7	2	2	6	6	7	4

?



i	1	2	3	4	5	6	7	8
parent	2	2	2	2	6	6	2	4

БАЗОВЫЕ ОПЕРАЦИИ

$\boldsymbol{FindSet}(\boldsymbol{x})$ — $O(h)$

$\boldsymbol{Union}(\boldsymbol{x}, \boldsymbol{y})$ — $O(h)$

$$h = \Theta(n)$$

Усовершенствование 1

Объединение по размеру (или рангу)

Если у каждого корневого дерева поддерживать весовую характеристику, в качестве которой может выступать или

число вершин в дереве (размер),

или

высота дерева (ранг),

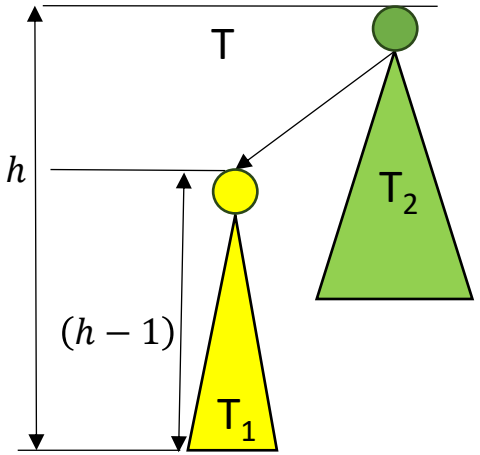
то тогда при выполнении операции $\boldsymbol{Union}$ корень дерева с меньшей весовой характеристикой будет присоединяться в качестве сына к корню дерева с большей весовой характеристикой (т.е. **присоединяем меньшее к большему**).

Теорема

Если при выполнении каждой операции ***Union*** корень дерева с меньшим числом вершин преобразуется в сына корня дерева с большим числом вершин (**эвристика объединения по размеру**), то дерево в семействе сможет достичь высоты $\boldsymbol{h}$, только, если оно имеет не менее $\boldsymbol{2^h}$ вершин ($\boldsymbol{n_{\mathrm{T}} \geq 2^h}$).

Доказательство

Проведём доказательство по индукции.



Для $h = 0$ утверждение верно, так как каждое дерево имеет, по крайней мере, один узел.

Предположим, что утверждение верно для всех значений параметра, меньших h (≥ 1).

Пусть T – дерево высоты h с минимальным числом вершин.

Предположим, что дерево T получилось в результате слияния двух деревьев T_1 и T_2 и

$$n(\mathrm{T}_2) \geq n(\mathrm{T}_1).$$

Тогда должно выполняться:

- ✓ $h(\mathrm{T}_2) < h$ (если предположить, что $h(\mathrm{T}_2) = h$, то придём к противоречию, что T – дерево высоты h с минимальным числом вершин;
- ✓ $h(\mathrm{T1}) = h - 1$ (иначе у дерева T не будет достигаться высота h).

По индукционному предположению $n(\mathrm{T1}) \geq 2^{h-1}$ поэтому получаем, что

$$n(\mathrm{T}) = n(\mathrm{T}_1) + n(\mathrm{T}_2) \geq n(\mathrm{T}_1) + n(\mathrm{T}_1) \geq 2^{h-1} + 2^{h-1} = 2^h.$$

Следствие

Никакое дерево $\boldsymbol{T}$ при объединении по размеру не может иметь высоту $\boldsymbol{h(T)}$ большую, чем $\boldsymbol{log_2 n}$.

Доказательство

По теореме для любого дерева $\boldsymbol{T}$ в семействе выполняется неравенство:

$$\boldsymbol{n(T) \geq 2^{h(T)}}.$$

Так как $\boldsymbol{n(T) \leq n}$, то для любого дерева $\boldsymbol{T}$ выполняется

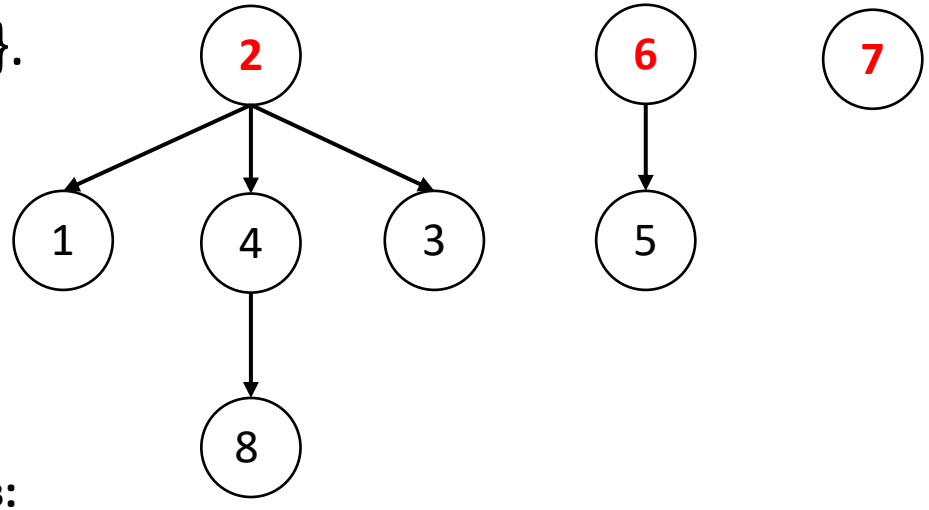
$$\boldsymbol{h(T) \leq log_2 n(T) \leq log_2 n}.$$

БАЗОВЫЕ ОПЕРАЦИИ

$\boldsymbol{FindSet(x)}$ —	$O(\log n)$
$\boldsymbol{Union(x, y)}$ —	$O(\log n)$

Пример.

Система непересекающихся множеств: {1, 2, 3, 4, 8}, {5, 6}, {7}.



Интерфейс реализуется на семействе корневых деревьев с эвристикой объединения по размеру.

В памяти компьютера для хранения DSU можно использовать один из двух способов:

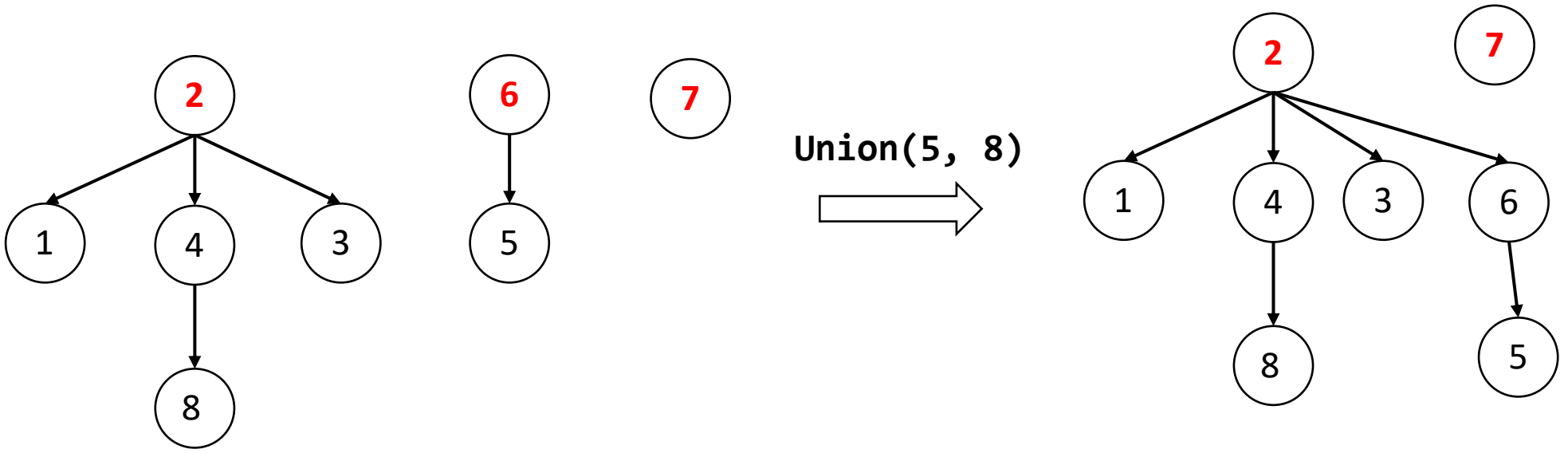
Способ 1

	1	2	3	4	5	6	7	8
parent	2	2	2	2	6	6	7	4
size	1	5	1	2	1	2	1	1

Способ 2

	1	2	3	4	5	6	7	8
parent	2	-5	2	2	6	-2	-1	4

Пример (продолжение).



Способ 1

	1	2	3	4	5	6	7	8
parent	2	2	2	2	6	6	7	4
size	1	5	1	2	1	2	1	1

	1	2	3	4	5	6	7	8
parent	2	2	2	2	6	2	7	4
size	1	7	1	2	1	2	1	1

Способ 2

parent	1	2	3	4	5	6	7	8
	2	-5	2	2	6	-2	-1	4

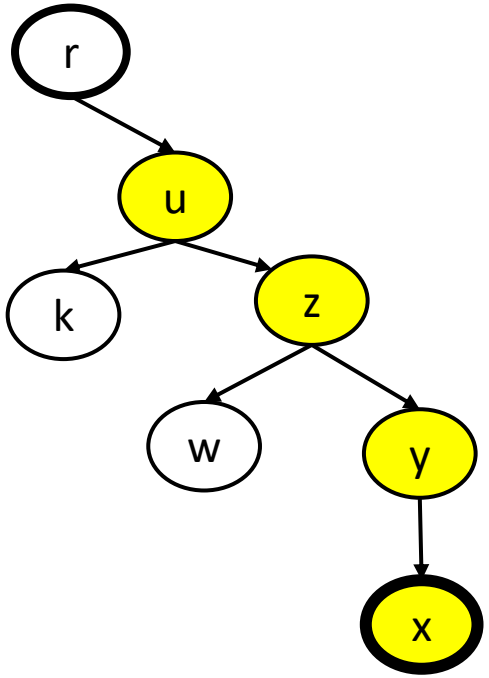
parent	1	2	3	4	5	6	7	8
	2	-7	2	2	6	2	-1	4

Усовершенствование 2.

Сжатие пути (англ. *path compression*)

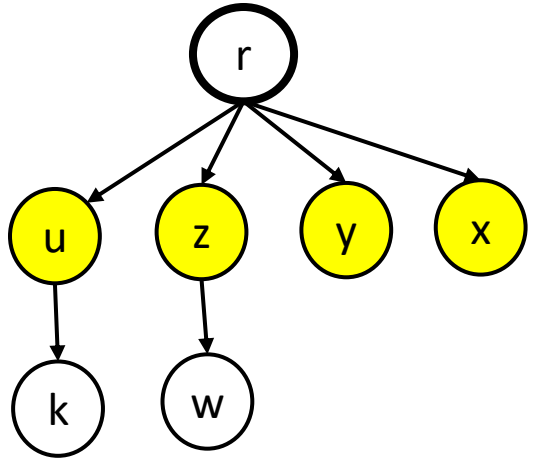
Позволяет получить практически линейное время работы серии операций $FindSet$ и $Union$.

Предположим, что выполняется операция $FindSet(\boldsymbol{x})$, которая выполняется простым подъёмом от вершины $\boldsymbol{x}$ к корню дерева $\boldsymbol{r}$, используя массив предков $parent$.
Все посещённые при этом подъёме вершины составляют путь поиска.

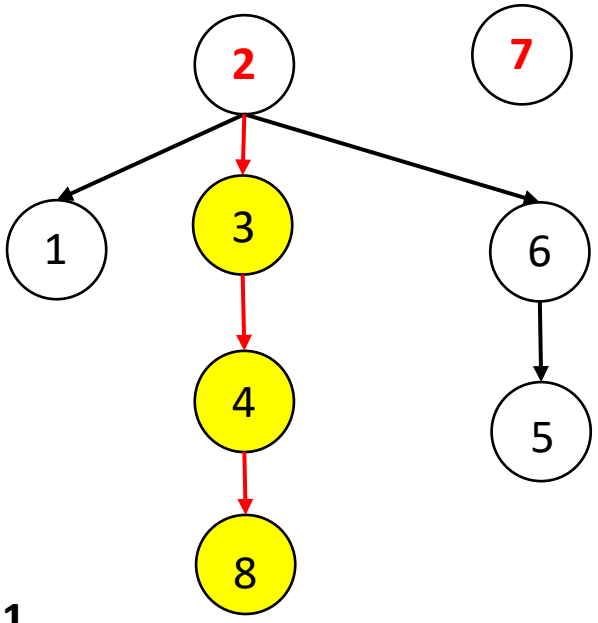


Процедура сжатия пути

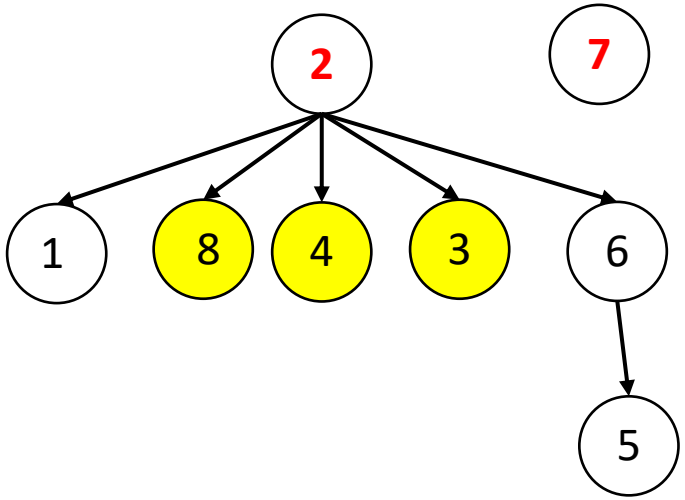
> всем вершинам, лежащим **на пути поиска**, в качестве предка присваивает ссылку на корень данного дерева (сжатие пути не изменяет ранги вершин).



Пример.



$FindSet(8)$



Способ 1

	1	2	3	4	5	6	7	8
parent	2	2	2	3	6	2	7	4
size	1	7	3	2	1	2	1	1

	1	2	3	4	5	6	7	8
parent	2	2	2	2	6	2	7	2
size	1	7	3	2	1	2	1	1

Способ 2

	1	2	3	4	5	6	7	8
parent	2	−7	2	3	6	2	−1	4

	1	2	3	4	5	6	7	8
parent	2	−7	2	2	6	2	−1	2

Теорема

Последовательность из m операций $\boldsymbol{FindSet(x)}$ и $\boldsymbol{Union(x, y)}$ может быть выполнена с **использованием эвристик объединения по размеру и сжатия пути** за время $\boldsymbol{O(m \cdot \alpha(n))}$ в худшем случае.

В формулировке теоремы функция $\boldsymbol{\alpha(n)}$ — обратная функция для функции Аккермана.
Функция $\alpha(n)$ растёт очень медленно, она становится больше числа 4 только для очень больших значений $n \gg 10^{80}$.
Поэтому для практических приложений полагают, что $\alpha(n) \leq 4$.

Функция Аккермана:

$$A(m,n) = \begin{cases} n+1, & m = 0; \\ A(m-1,1), & m > 0, n = 0; \\ A(m-1, A(m,n-1)), & m > 0, n > 0. \end{cases}$$

Обратная функция для функции Аккермана определяется следующим образом:

$$\alpha(n) = \min\{k \geqslant 1 : A(k,k) \geqslant n\}.$$

Вильгельм Аккерман
Wilhelm Ackermann
1886-1962
Германия

ФПМИ БГУ

```
class DisjointSetUnion:

def __init__(self, n):
  self.size = [1 for i in range(n)]
  self.parent = [i for i in range(n)]

def FindSet(self, x):
  if x == self.parent[x]:
  return x
 # Path compression
 self.parent[x] = self.FindSet(self.parent[x])
 return self.parent[x]

def Union(self, x, y):
  x = self.FindSet(x)
  y = self.FindSet(y)
  if x != y:
    # Union by size
    if self.size[x] < self.size[y]:
    # Swap x and y
    x, y = y, x
    # Now size[x] >= size[y]
   self.parent[y] = x
   self.size[x] += self.size[y]
```

ФПМИ БГУ

Общие задачи в iRunner для закрепления навыков реализации DSU

0.5 Строительство дорог

0.6 Разрушение дорог (простая версия)

0.7 Разрушение дорог (сложная версия)

Задача 0.5. Строительство дорог

Имя входного файла: input.txt
Имя выходного файла: output.txt
Ограничение по времени: 1 с
Ограничение по памяти: 256 МБ

Берляндия состоит из n городов. Изначально все города изолированы, то есть между городами нету дорог.

По очереди будут добавляться дороги между парами городов. Необходимо после каждой добавленной дороги узнать, какое количество компонент связности из городов получилось.

Формат входных данных

Первая строка входного файла содержит два целых числа n и q ($1 \leqslant n, q \leqslant 500000$) — количество городов и запросов соответственно.

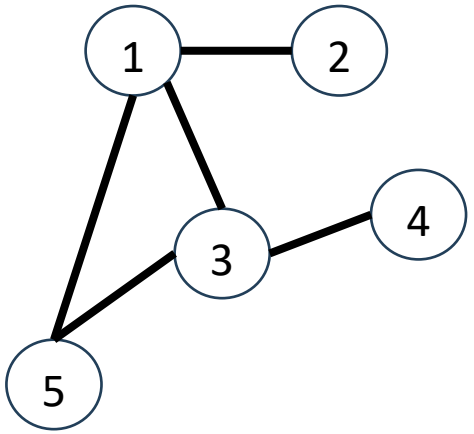
Каждая из следующих q строк содержит два целых числа u, v — между какой парой городов будет построена дорога. В данной задаче между любой парой городов строится не более одной дороги, а для любого запроса справедливо $u \neq v$.

Формат выходных данных

На каждый запрос второго типа необходимо вывести одно число — количество компонент связности в графе из городов.

Пример

input.txt	output.txt
5 5 1 2 3 4 1 3 3 5 1 5	4 3 2 1 1



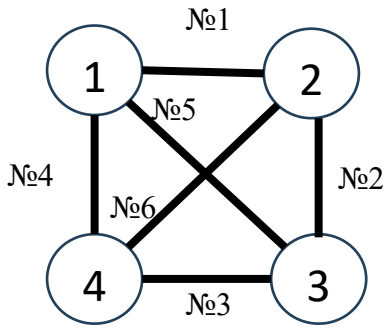
ФПМИ БГУ

Задача 0.6. Разрушение дорог (простая версия)

Имя входного файла: input.txt
Имя выходного файла: output.txt
Ограничение по времени: 1 с
Ограничение по памяти: 256 МБ

В Берляндии n городов, связанных m дорогами. Гарантируется, что изначально граф из городов связный, т.е. существует путь между любой парой вершин.

В Берляндии происходит q землетрясений, в ходе каждого из них разрушается ровно одна дорога. Необходимо после каждого землетрясения узнать, является ли полученный граф из городов связным. После очередного землетрясений дорога не перестраивается, то есть разрушается навсегда.



Формат входных данных

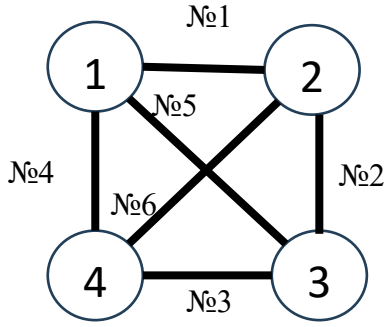
Первая строка входного файла содержит три числа n, m, q ($1 \le n, m, q \le 100000, 2 \le n$) - количество городов, дорог и землетрясений в Берляндии соответственно.

Следующие m строк содержат по два натуральных числа u и v ($1 \le u, v \le n$) - номера городов, связанных дорогой. Гарантируется, что $u \ne v$ и между каждой парой городов существует не больше одной дороги.

Следующие q строк содержат по одному числу x_i ($1 \le x_i \le m$) - номер дороги, которая пострадает во время i-ого землетрясения. Гарантируется, что два различных землетрясения не разрушают одну и ту же дорогу.

Формат выходных данных

Выведите строку длины q из нулей и единиц. i-ый символ равен 1, если после i-ого землетрясения граф оказался связным, в противном случае выведите 0.



Пример

input.txt	output.txt
4 6 6 1 2 2 3 3 4 4 1 3 1 4 2 1 6 2 5 4 3	110000

ФПМИ БГУ

Задача 0.7. Разрушение дорог (сложная версия)

Имя входного файла: `input.txt`
Имя выходного файла: `output.txt`
Ограничение по времени: 1,25 с
Ограничение по памяти: 256 МБ

В Берляндии n городов, связанных m дорогами. Гарантируется, что изначально граф из городов связный, т.е. существует путь между любой парой вершин.

В Берляндии происходит q землетрясений, в ходе каждого из них разрушается ровно одна дорога. Необходимо после каждого землетрясения узнать, является ли полученный граф из городов связным. После очередного землетрясений дорога не перестраивается, то есть разрушается навсегда.

Формат входных данных

Первая строка входного файла содержит три числа n, m, q $(1 \le n, m, q \le 750000)$ - количество городов, дорог и землетрясений в Берляндии соответственно.

Следующие m строк содержат по два натуральных числа u и v $(1 \le u, v \le n)$ - номера городов, связанных дорогой. Гарантируется, что $u \neq v$ и между каждой парой городов существует не больше одной дороги.

Следующие q строк содержат по одному числу x_i $(1 \le x_i \le m)$ - номер дороги, которая пострадает во время i-ого землетрясения. Гарантируется, что два различных землетрясения не разрушают одну и ту же дорогу.

Формат выходных данных

Выведите строку длины q из нулей и единиц. i-ый символ равен 1, если после i-ого землетрясения граф оказался связным, в противном случае выведите 0.

Пример

input.txt	output.txt
4 6 6 1 2 2 3 3 4 4 1 3 1 4 2 1 6 2 5 4 3	110000

Спасибо за внимание!