Disjoint-set union

(Система непересекающихся множеств)

СНМ –

структура данных, которая хранит и оперирует непересекающимися множествами.

По сути, она хранит разбиение множества в виде непересекающихся подмножеств, каждое из которых задается представителем.

В классическом варианте предоставляет 3 операции на добавление нового множества, объединения двух множеств и поиск элемента в множестве.

A little bit of history…

Впервые СНМ была описана Бернардом Галлером и Майклом Джоном Фишером в 1964 году. В 1973 году была доказана асимптотика $O(\log n)$.

В 1975 Роберт Тарьян показал, что на самом деле асимптотика равняется $O(a(n))$, где $a(n)$ – функция, обратная к функции Аккермана.

В 1989 году Майкл Фридман и Майкл Сакс доказали, что амортизированное время $\Omega(a(n))$ достигается в любой структуре данных СНМ за одну операцию.

Определение

Пусть S – конечное множество, разбитое на непересекающиеся подмножества (классы) X_i.

$$S = X_0 \cup X_1 \cup X_2 \cup \dots \cup X_n:$$
$$X_i \cap X_j = \emptyset \qquad \forall i, j \in \{0, \dots k\}, i \neq j$$

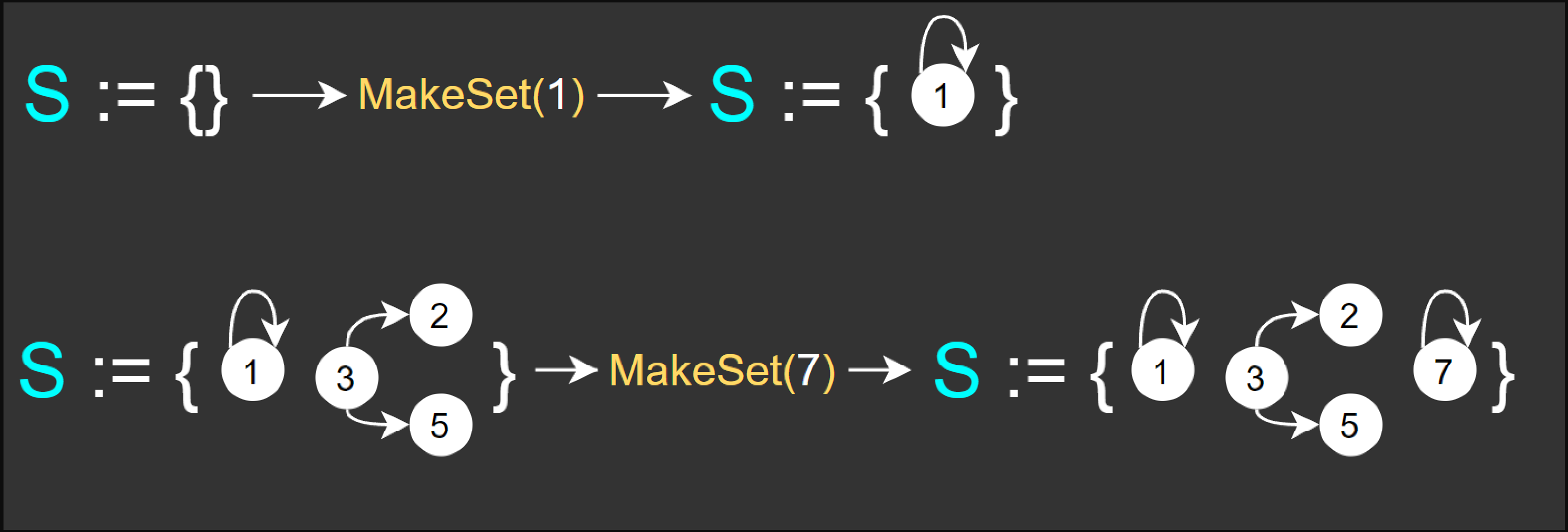
Для каждого подмножества X_i определен свой представитель $p_i \in X_i$.

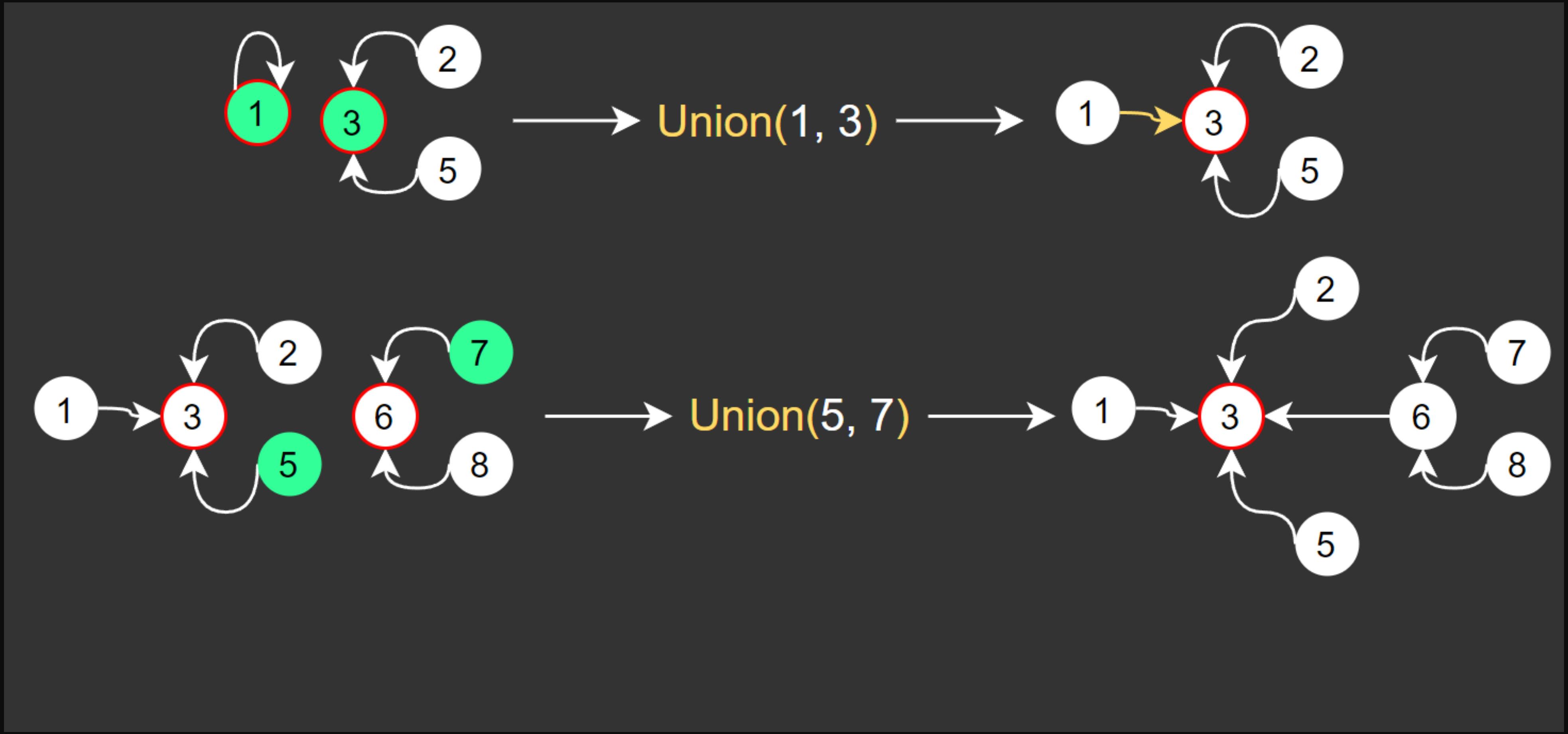
Операции

- $MakeSet(x)$: создаёт для элемента x новое подмножество. Назначает x представителем созданного подмножества.

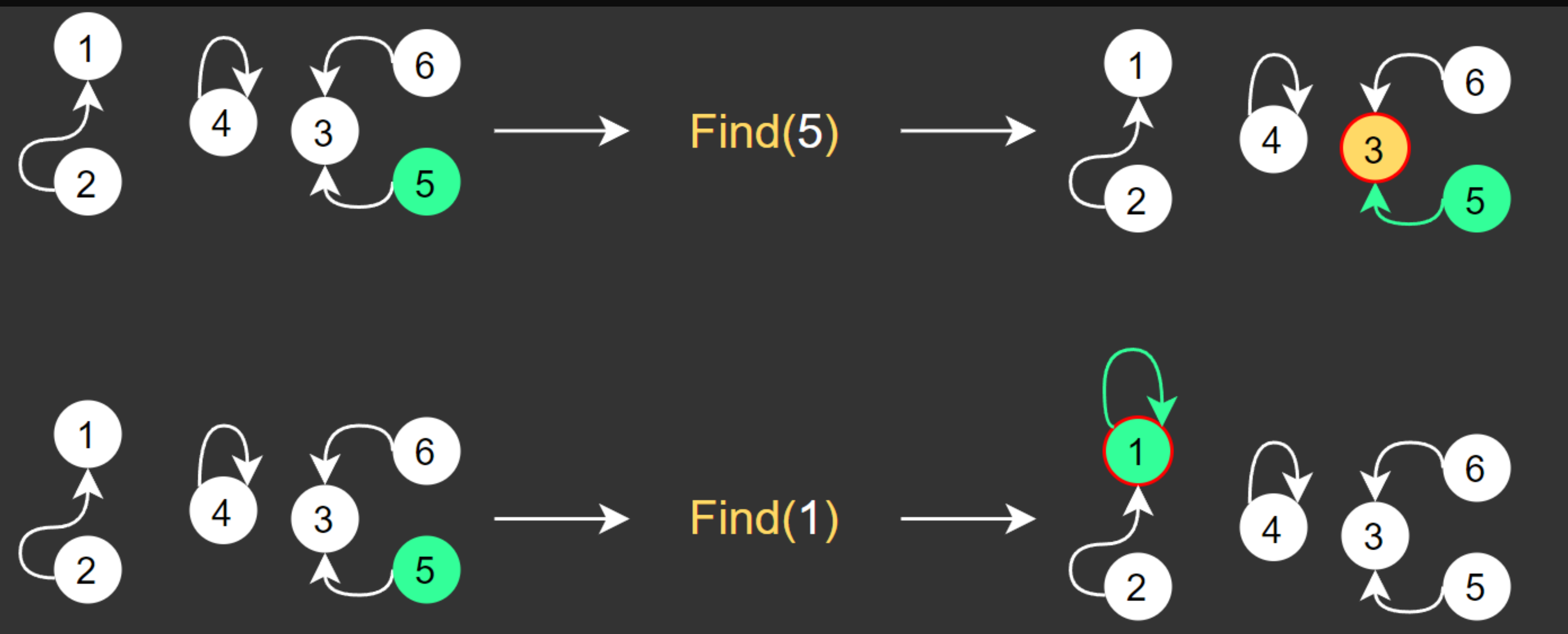
- $Union(r, s)$: объединяет оба подмножества, принадлежащие представителям r и s, и назначает представителем нового подмножества одного из них.

- $Find(x)$: определяет для $x \in S$ подмножество, к которому принадлежит элемент, и возвращает его представителя.


S := {}
MakeSet(1)
S := { 1 }
S := { 1 3 2 5 }
MakeSet(7)
S := { 1 3 2 5 7 }



1
3
2
5
Union(1, 3)
1
3
2
5
1
3
2
5
6
7
8
Union(5, 7)
2
1
3
6
7
8
5


1
2
4
3
6
5
Find(5)
1
2
4
3
6
5
1
2
4
3
6
5
Find(1)
1
2
4
3
6
5

Реализация

(наиболее практичная)

Каждое подмножество будет представлено в виде дерева, у которого узлами будут элементы этого подмножества, а корнем будет его представитель.

Ссылки на самих представителей будем где-то хранить, например, в динамическом массиве. Тогда $MakeSet(x)$ будет работать за $O(1)$ как добавление в конец этого массива.

$Find(x)$ будет работать за $O(n)$, поскольку в худшем случае деревья выродятся в список и потребуется $n-1$ итераций, чтобы отыскать представителя.

$Union(r, s)$ аналогично $Find(x)$ в худшем случае будет работать за $O(n)$, а для m запросов получаем $O(nm)$.

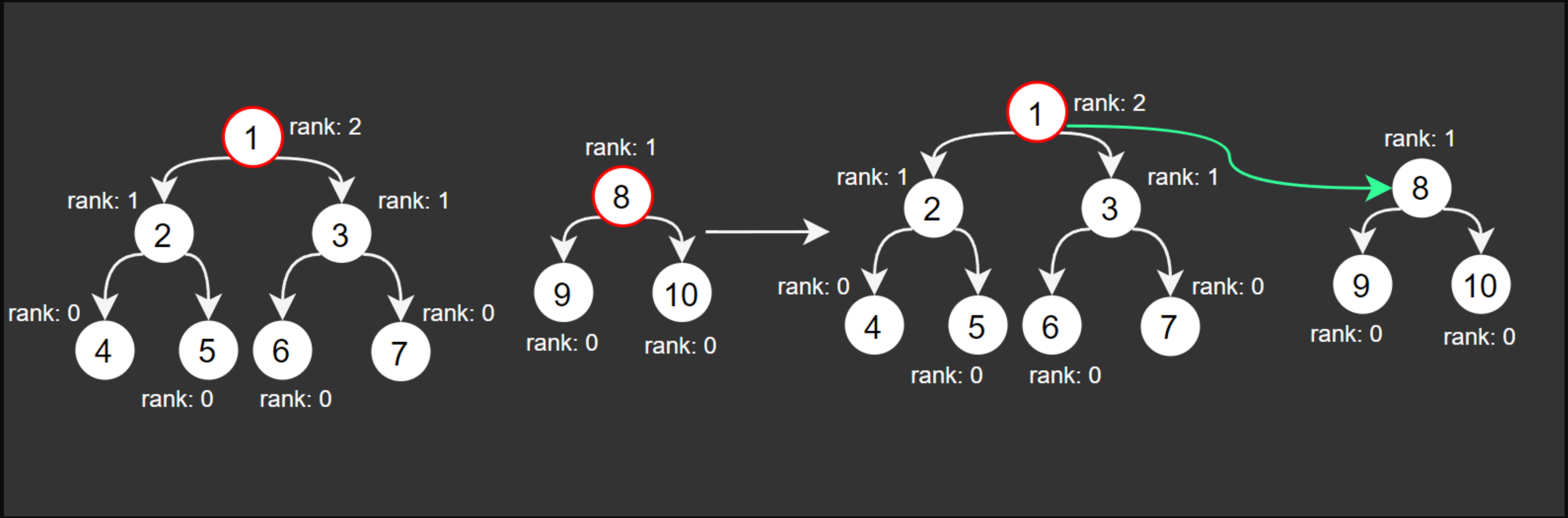
Можно лучше?

- Требуется улучшить асимптотику функций $Find(x)$ и $Union(r, s)$.

- Можем использовать эвристику $UnionByRank$. $Rank$ – для каждого элемента будем хранить специальное значение: верхнюю границу высоты данного узла. Будем всегда подвешивать дерево с меньшим рангом к дереву с большим.

- Не сложно показать, что тогда будет достигаться сбалансированность деревьев и у каждого дерева T высота в среднем будет $\log T$.

- Соответственно, ожидаемое время работы $Union(r, s)$ (также, как и для $Find(x)$), в худшем случае будет $O(\log T)$. Для m запросов получаем $O(m \log n)$.


1
rank: 2
rank: 1
2
rank: 1
3
rank: 0
4
5
rank: 0
6
rank: 0
7
rank: 0
rank: 1
8
9
rank: 0
10
rank: 0
1
rank: 2
rank: 1
2
rank: 1
3
rank: 0
4
5
rank: 0
6
rank: 0
7
rank: 0
rank: 1
8
9
rank: 0
10
rank: 0

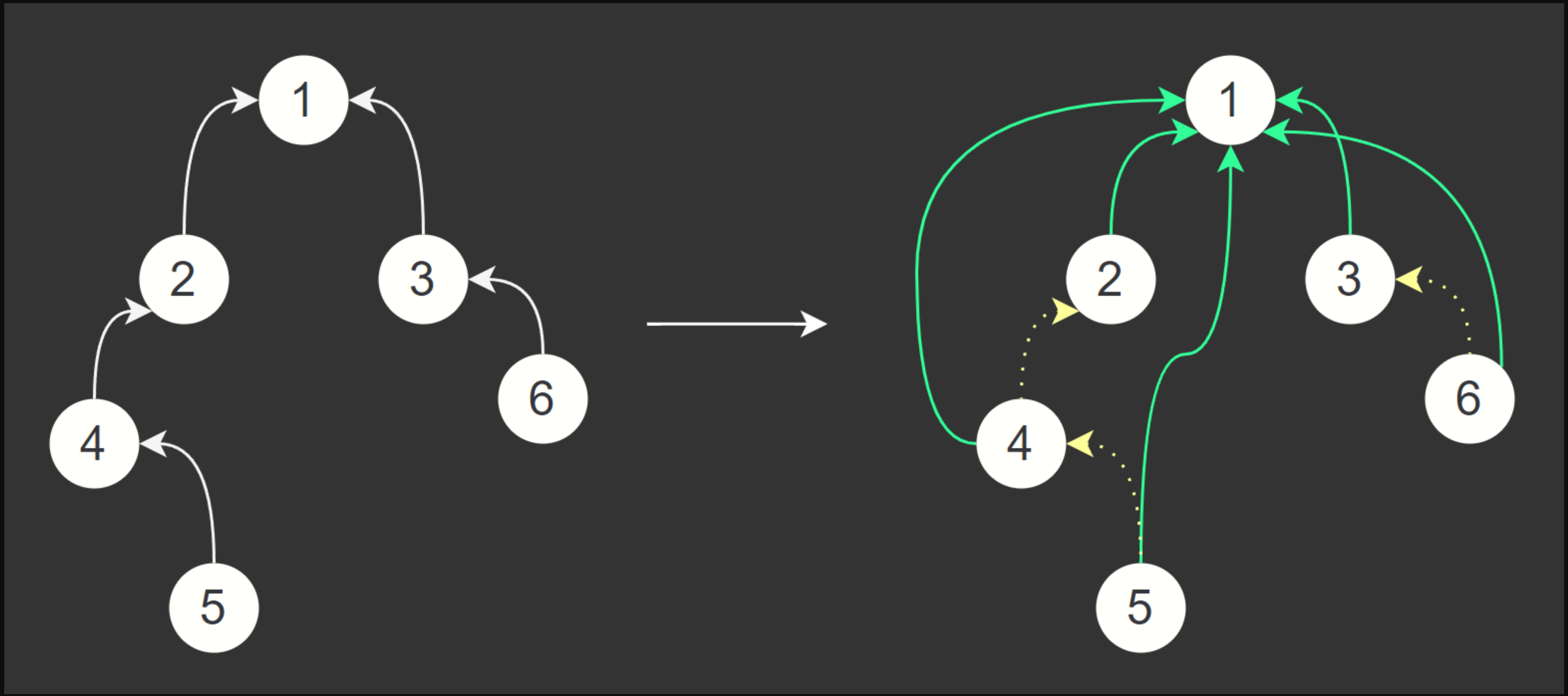
Boost more

Попробуем использовать еще одну эвристику – сжатие путей. При каждом новом поиске все элементы, находящиеся на пути от корня до искомого элемента, вешаются под корень дерева.

Тогда, ожидаемая асимптотика $Find(x)$ для m запросов, при использовании данной эвристики (вместе с предыдущей) будет $O\big(ma(n)\big)$. А тогда амортизированное время для одной операции $Find(x)$ будет $O\big(a(n)\big)$.

$a(n)$ – функция, обратная к функции Аккермана, которая растет настолько медленно, что при всех мыслимых практических ограничениях данную сложность можно считать константной.

Соответственно и $Union(r, s)$ имеет такую сложность $O\big(a(n)\big)$.


1
2
3
4
5
6
1
2
3
4
5
6

Точное формальное доказательство асимптотики $O\big(a(n)\big)$ является достаточно непростой и объемной задачей (с этим доказательством можно ознакомится в [1]).

Однако, существует гораздо более простое доказательство, которое показывает, что амортизированное время для любых m операций $Find(x)$ или $Union(r, s)$ равняется $O(m \log* n)$, где $\log*$ обозначает итерированный логарифм.

Данную асимптотику тоже можно считать почти константной, поскольку $\log*$ растет достаточно медленно.

Для сравнения: $\log* 2^{2^{2^{2^{2}}}} = 5$.

Доказательство

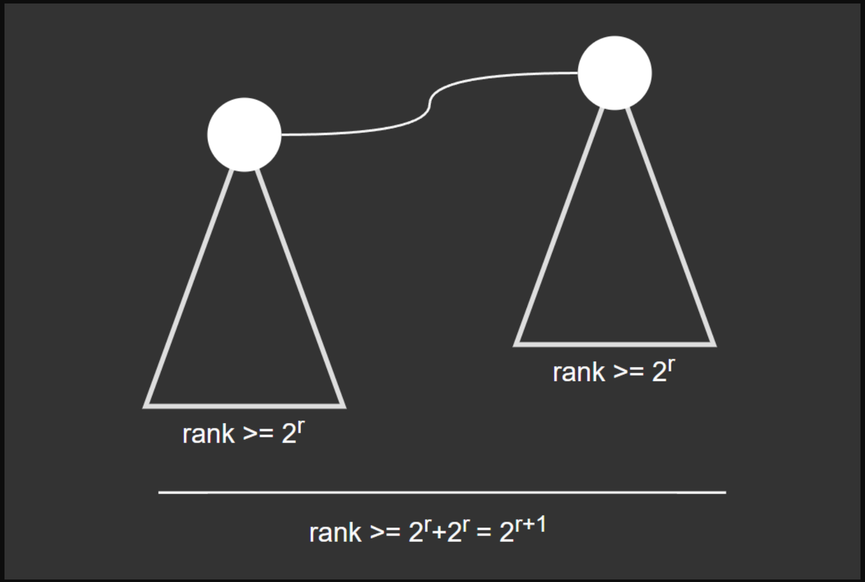
Докажем, что для m запросов типа $Find(x)$, при использовании эвристики $UnionByRank$, асимптотика будет $O(m \log* n)$. Можно будет сделать вывод, что амортизированное время для одной операции будет $O(\log* n)$.

Лемма 1. По мере того, как функция $Find(x)$ следует по пути к корню, ранг очередного встречаемого узла увеличивается.

Если каждый узел является представителем своего собственного дерева, то утверждение тривиальное.
Иначе дерево с меньшим рангом всегда будет присоединятся к дереву с большим рангом, а не наоборот. Тогда все узлы, посещенные по пути, будут привязаны к корню, который имеет больший ранг, чем его дочерние элементы.

Лемма 2. Узел u, который является корнем поддерева ранга r, имеет не менее 2^r узлов.

Если каждый узел является представителем своего собственного дерева, то утверждение тривиально верно. В ином случае, предположим, что узел u ранга r имеет не менее 2^r узлов. Затем, когда два дерева с рангом r объединяются с помощью $UnionByRank$, получается дерево с рангом $r + 1$, корень которого имеет не менее $2^r + 2^r = 2^{r+1}$ узлов.



Лемма 3. Максимальное количество узлов ранга r не превышает $\frac{n}{2^r}$.

Очевидное следствие из Леммы 2. Мы получим максимальное количество узлов ранга r тогда и только тогда, когда каждый узел ранга r является корнем дерева, которое имеет ровно 2^r узлов.

Введем понятие “блока”. Каждый блок хранит множество узлов с определенным рангом. Если $B-й$ блок содержит вершины с рангами из интервала $[r,\ 2^r - 1] = [r, R-1]$, то $(B + 1) - й$ блок будет содержать вершины с рангами из интервала $[R,\ 2^R - 1]$ для всех $B\ >=\ 0$.


0
1
[2, 3]
[4, 15]
[B, 2^B-1]
[2^B, 2^2^B-1]

Можем заметить два факта касаемо блоков:

1. Общее количество блоков не превышает $\log * n$ (когда мы переходим от $i - \text{го}$ блока к $(i + 1) - \text{ому}$, мы возводим еще одну двойку в уже существующую степень.

2. Максимальное количество элементов в блоке $[B, 2^B - 1]$ не превышает $\frac{2n}{2^B}$.

$$\frac{n}{2^B} + \frac{n}{2^{B+1}} + \frac{n}{2^{B+2}} + \cdots + \frac{n}{2^{2^B - 1}} \leq \frac{2n}{2^B}$$

Пусть F будет представлять список из произведенных m запросов $Find(x)$, а
$T_1 = \sum_F(\text{ссылка от узла к его представителю})$;
$T_2 = \sum_F(\text{число пройденных ссылок, если блоки разные})$;
$T_3 = \sum_F(\text{число пройденных ссылок, если блоки одинаковые})$.

Тогда все время выполнения этих m запросов будет равно $T = T_1 + T_2 + T_3$.

$T_1 = O(m)$, потому что $Find(x)$ делает только один переход от узла к представителю.
$T_2 = O(m \log * \, n)$, потому что в наблюдении 1 мы сказали, что количество блоков не превышает $\log * \, n$.

Для T_3 предположим, что мы проходим ребро от узла u до v, где ранги u и v находятся в блоке $[B, 2^B - 1]$ и v не является представителем (если v таки является им, то имеем случай с T_1).

Зафиксируем u и рассмотрим последовательность $v_1, v_2, v_3, \dots v_k$. Из-за сжатия пути и без учета ребра, ведущего к представителю, эта последовательность содержит только различные узлы. Из леммы 1 мы знаем, что ранги узлов в этой последовательности строго возрастают.

Так как оба узла находятся в одном блоке, мы можем заключить, что длина последовательности k (количество раз, когда узел u присоединяется к другому в одном и том же блоке) не превышает числа рангов в ведрах B, что не более чем $2^B - B - 1 < 2^B$.

Для T_3 предположим, что мы проходим ребро от узла u до v, где ранги u и v находятся в блоке $[B, 2^B - 1]$ и v не является представителем (если v таки является им, то имеем случай с T_1).

Зафиксируем u и рассмотрим последовательность $v_1, v_2, v_3, \dots v_k$. Из-за сжатия пути и без учета ребра, ведущего к представителю, эта последовательность содержит только различные узлы. Из леммы 1 мы знаем, что ранги узлов в этой последовательности строго возрастают.

Так как оба узла находятся в одном блоке, мы можем заключить, что длина последовательности k (количество раз, когда узел u присоединяется к другому в одном и том же блоке) не превышает числа рангов в ведрах B, что не более чем $2^B - B - 1 < 2^B$.

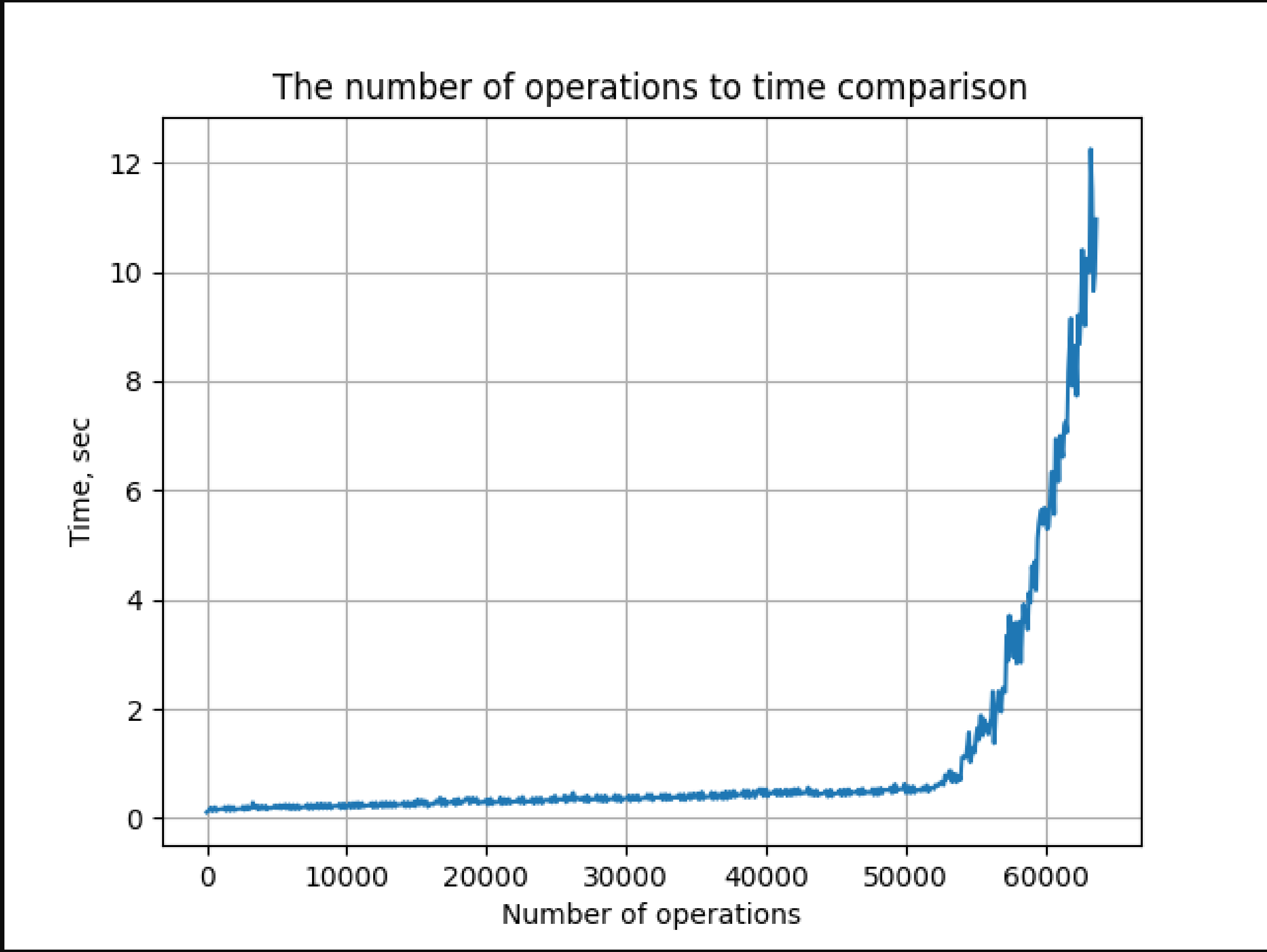
Следовательно, $T_3 \leq \sum_{[B, 2^B - 1]} \sum_u 2^B$.

Из наблюдений 1 и 2 можно сделать вывод, что $T_3 \leq \sum_B 2^B \frac{2n}{2^B} \leq 2n \log* n$.

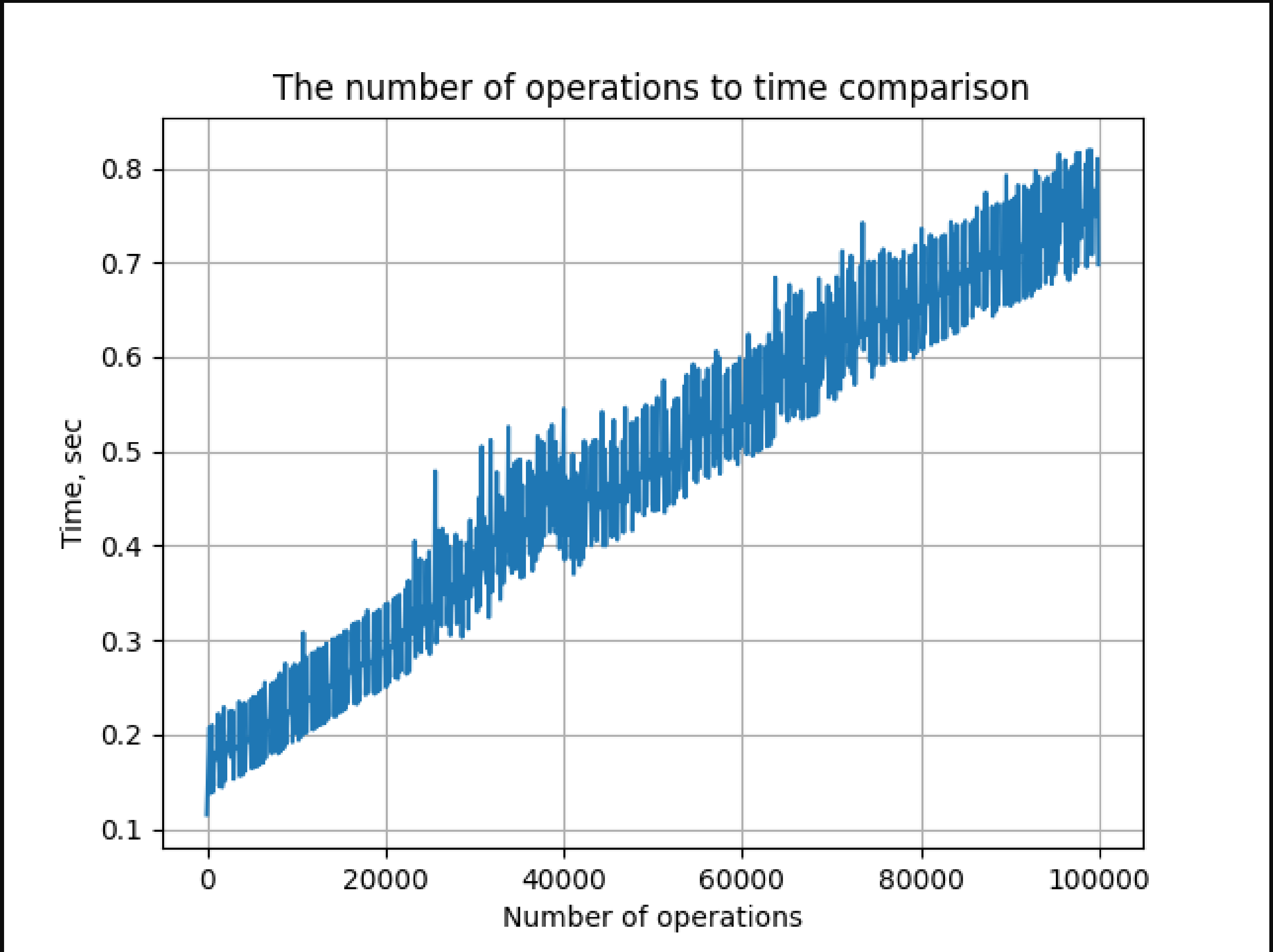
Тогда, общее время работы будет $T \ = \ T_1 + \ T_2 + T_3 = O(m \log* n)$.

∎

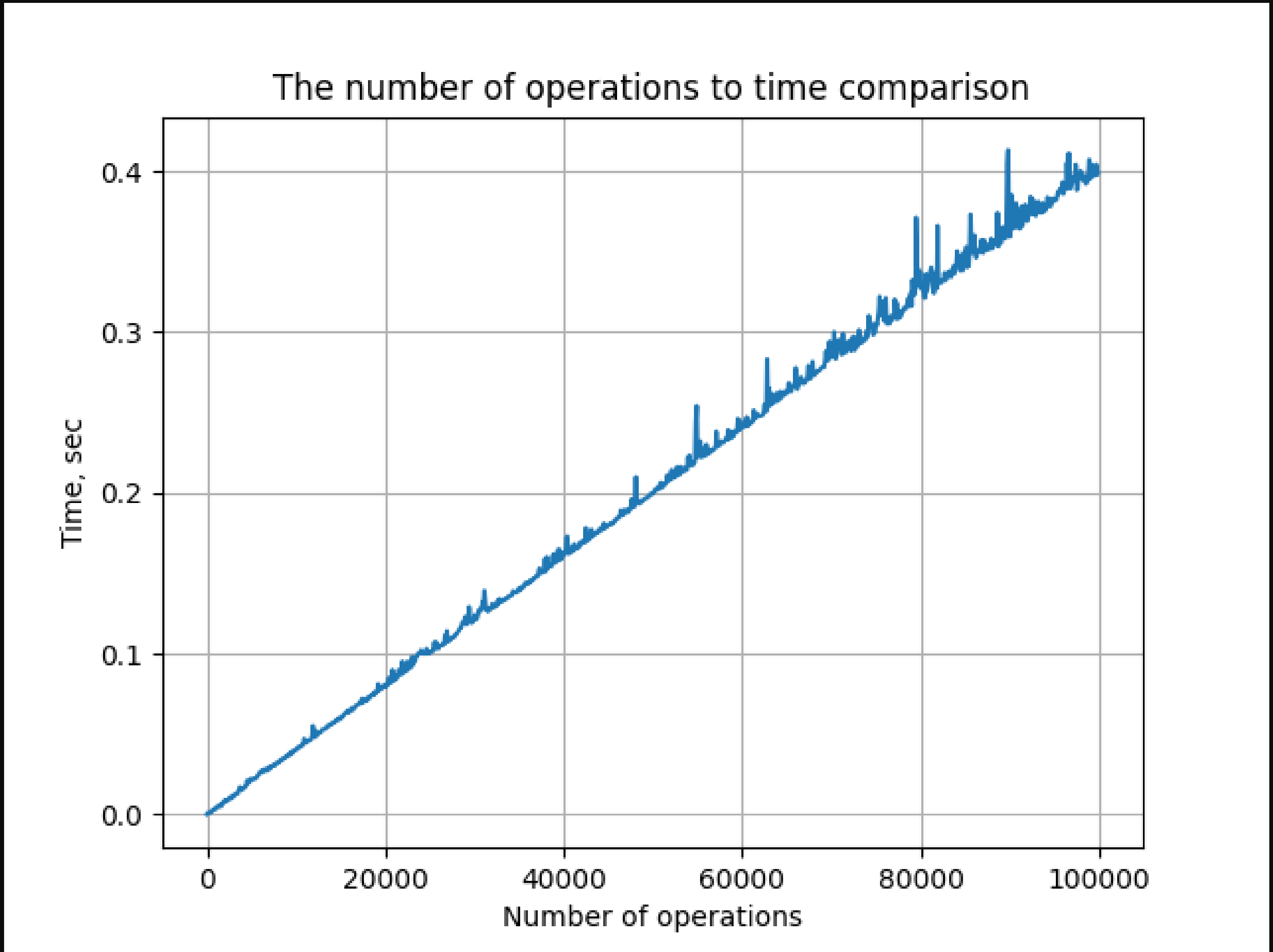
Несколько тестов



* без эвристик



* только с эвристикой *UnionByRank*



* с обеими эвристиками

I want more!

На самом деле есть еще несколько приемов, которые могут улучшить время выполнения операций $Find(x)$ и $Union(r, s)$.

$RandomUnion$ – можно выбирать представителя в множестве случайным образом. Если имеется много запросов типа $Union(\text{большое множество с маленьким})$, данная эвристика улучшает среднее время работы в два раза. На случайных запросах $RandomUnion$ может давать хороший результат сама по себе, однако рекомендуется ее использовать вместе с эвристикой сжатия путей. К тому же, тогда экономится дополнительные $O(n)$ памяти на сохранении количества узлов в дереве.

Существуют распараллеленные версии $Find(x)$ и $Union(r, s)$, которые не требуют блокировки потоков [4]. Параллельная обработка может еще больше ускорить алгоритм.

Персистентность?

В 2007 году Сильвен Коншон и Жан-Кристоф Филлиатр разработали персистентную версию СНМ, позволяющую эффективно сохранять предыдущие версии структуры, и доказали ее корректность [5].

Основная идея – использовать вместо обычных массивов персистентные. Итоговая сложность такой структуры будет $O(\log n)$.

Применения

Поиск MST (алгоритм Краскала)

Алгоритм Тарьяна для поиска LCA оффлайн (дано дерево и набор запросов вида: для данных вершин u и v вернуть их ближайшего общего предка)

Поиск компонент связности в мультиграфе

Сегментирование изображений

Генерация лабиринтов

Источники

1. Robert E. Tarjan, Robert Endre (1975). “Efficiency of a Good But Not Linear Set Union Algorithm”.

2. Robert E. Tarjan, Jan van Leeuwen (1984). “Worst-case analysis of set union algorithms”.

3. https://en.wikipedia.org/wiki/Disjoint-set_data_structure

4. Richard J. Anderson, Heather Woll (1994). “Wait-free Parallel Algorithms for the Union-Find Problem”.

5. Sylvain Conchon, Jean-Christophe Filliâtre (2007). “A Persistent Union-Find Data Structure”.