

Персистентные структуры данных -

- такие структуры данных, которые при внесении в них изменений сохраняют все свои предыдущие состояния и доступ к ним.

Уровни персистентности [\[править\]](#)

Есть несколько уровней персистентности:

- частичная (англ. *partial*),
- полная (англ. *full*),
- конфлюэнтная (англ. *confluent*),
- функциональная (англ. *functional*).

В частично персистентных структурах данных к каждой версии можно делать запросы, но изменять можно только последнюю версию структуры данных.

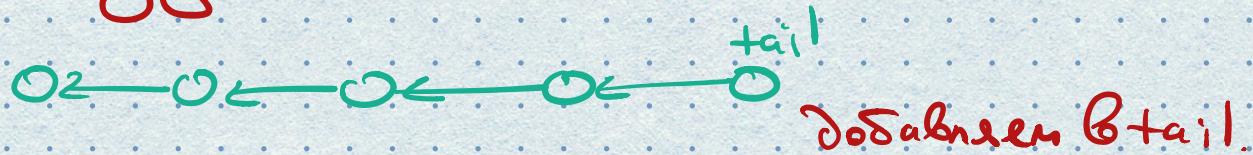
В полностью персистентных структурах данных можно менять не только последнюю, но и любую версию структур данных, также к любой версии можно делать запросы.

Конфлюэнтные структуры данных позволяют объединять две структуры данных в одну (деревья поиска, которые можно сливать).

Функциональные структуры данных полностью персистентны по определению, так как в них запрещаются уничтожающие присваивания, т.е. любой переменной значение может быть присвоено только один раз и изменять значения переменных нельзя. Если структура данных функциональна, то она и конфлюэнтна, если конфлюэнтна, то и полностью персистентна, если полностью персистентна, то и частично персистентна. Однако бывают структуры данных не функциональные, но конфлюэнтные.

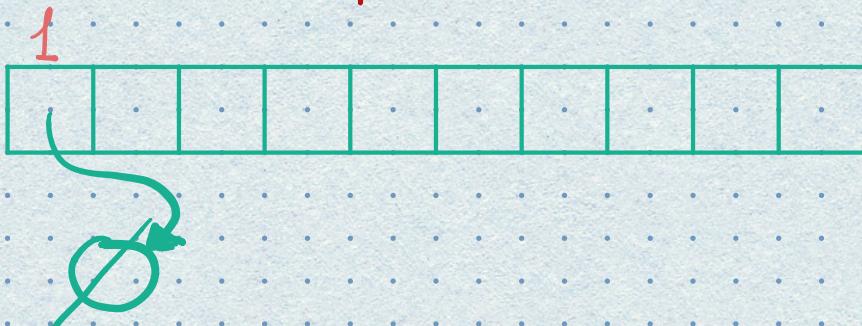
Stack

используем список

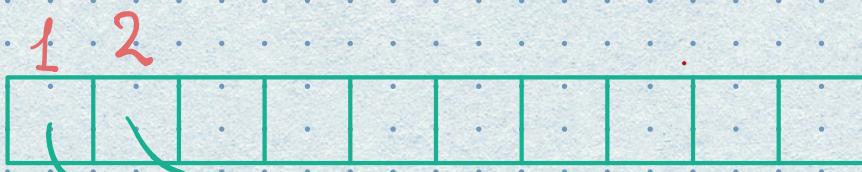


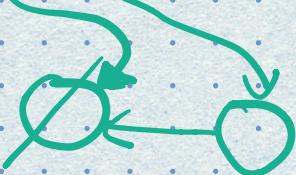
будем хранить список версий

1) пустой стэк



2) добавили элемент





1 2 3

3) добавили элемент.



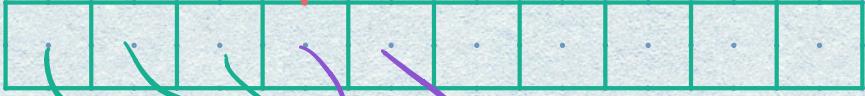
1 2 3 4

4) добавили элемент



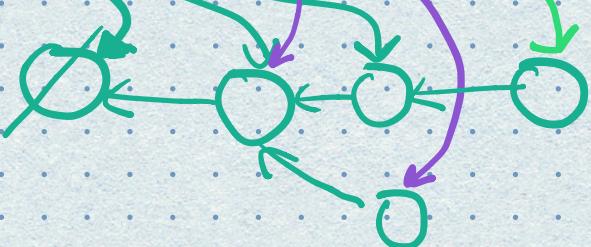
1 2 3 4 5

5) добавили к 4 варианту



1 2 3 4 5 6

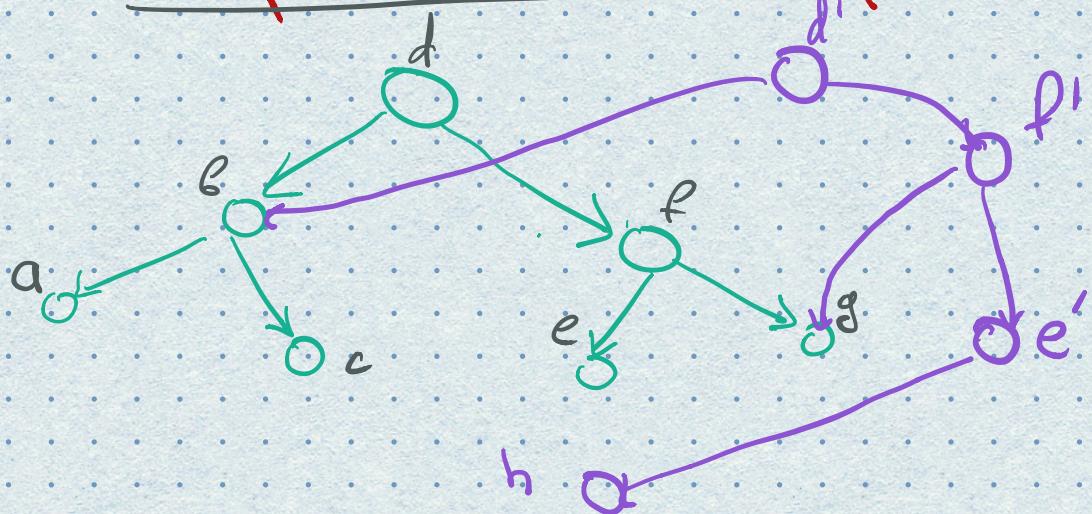
6) добавили элемент.
к 3 варианту



операции - $O(1)$

потреб. памяти - $O(\text{versions})$

Периодическое дин. дерево поиска.



Хотим добавить h как родителя e .

1. Создаем e' как конца e
2. Воссоздаем поддерево.

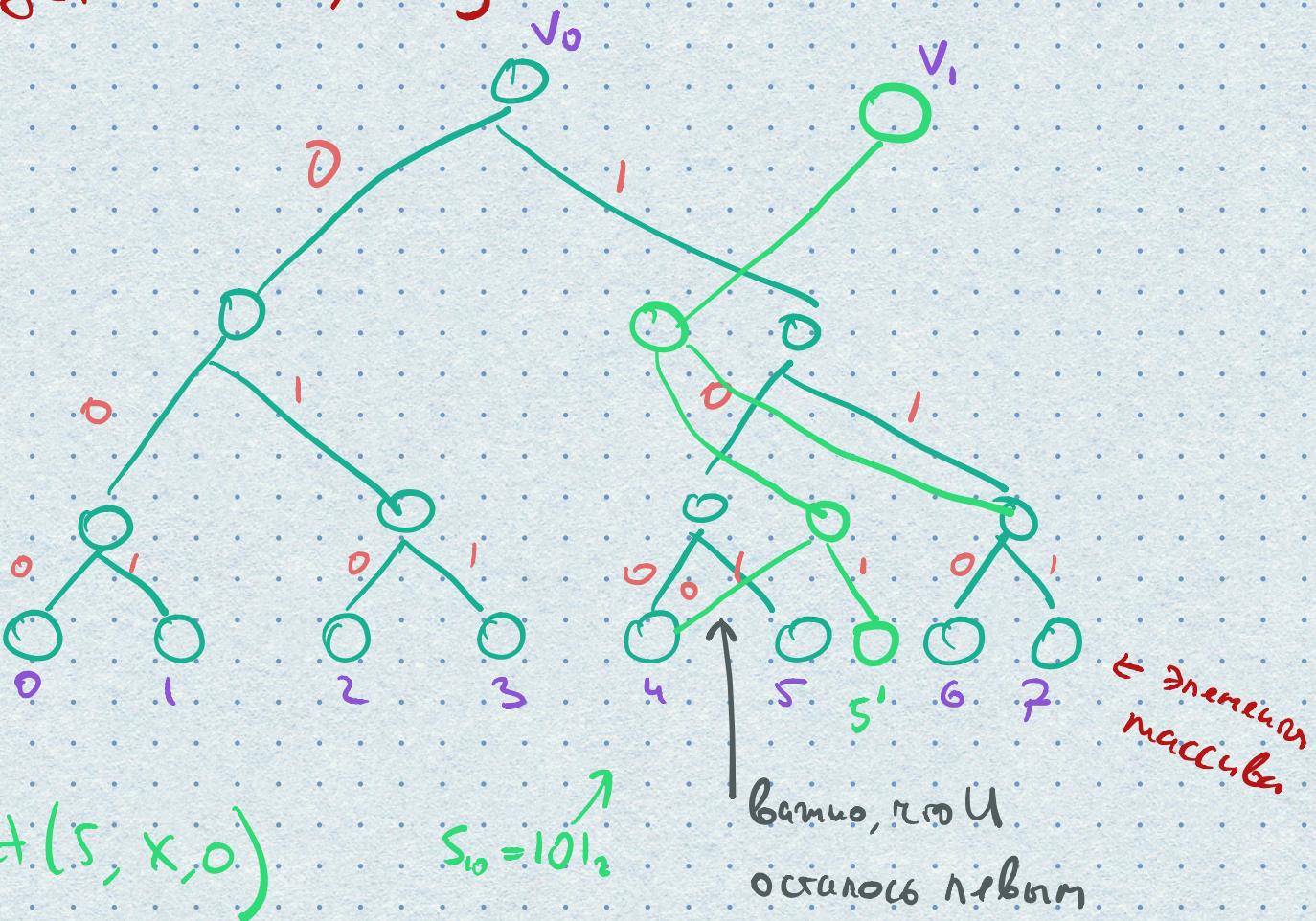
Причем т.к. b не меняется, то из d' мы должны находить b то же b . аналогично g .

создаём $\log(n)$ вершины для балансир. дерева и $\log(h)$ для поддерев.

Нерекурсивный массив

Хотим операции:

- set(index, x, ver)
- get(index, ver)



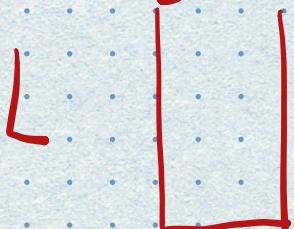
итерироваться — общее дерево

создание новых вершин $\log(n)$ — дерево сбалансировано

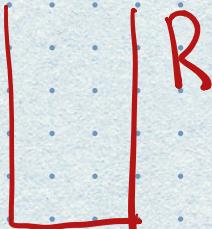
Очередь:

2 стека

push



pop.



когда R пустой, перекладываем

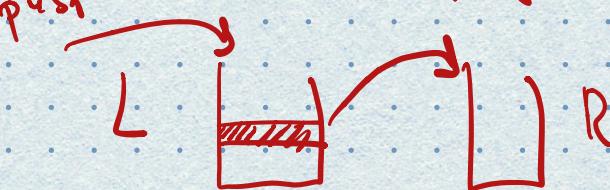
из L.

Аморт. сложность $O(1)$ т.к.
каждый элемент перекладывается не
больше 1 раза

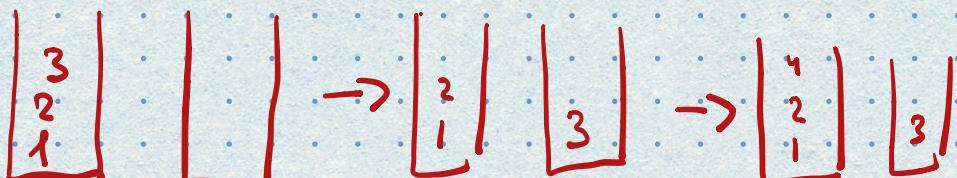
но в случае с пересечением это не подходит
это т.к. мы можем все время извлекать худшее
создавая очередь из сложности $O(n^2)$

Иdea: перекладывать из L в R непрерывно.

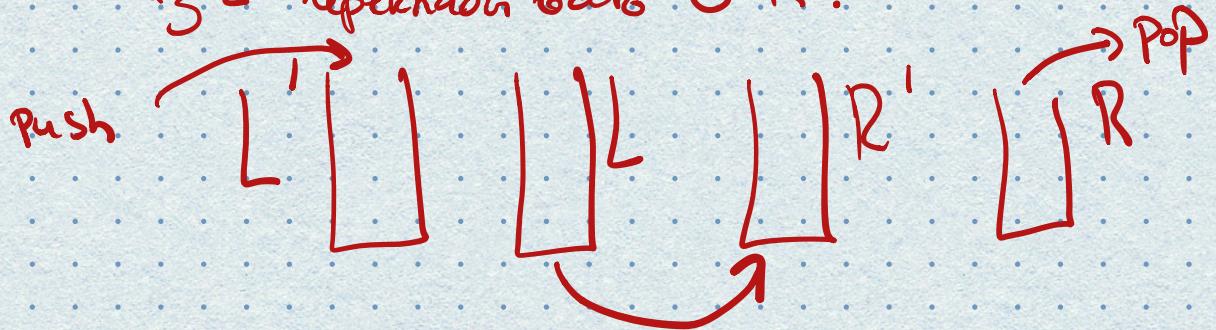
делать операцию — перекладка.



аналог т.к. можно сделать push и все упорядоч.



1. Добавим $R' L'$ в L' будем добавлять новые элементы из L перекладывая в $R R'$.



и меняем $L' c L$, $R' c R$

2. Если $B L$ то 0 элем. из R этим.

если придет 2 операции pop то мы ошиблись.
если наоборот, то все ок.

3. Решим перекопирование : $\text{size}(L) > \text{size}(R)$
с какими шагами перекладываем из L в R'
нося L не носим

общий решин: кладем в один, забираем из другого.

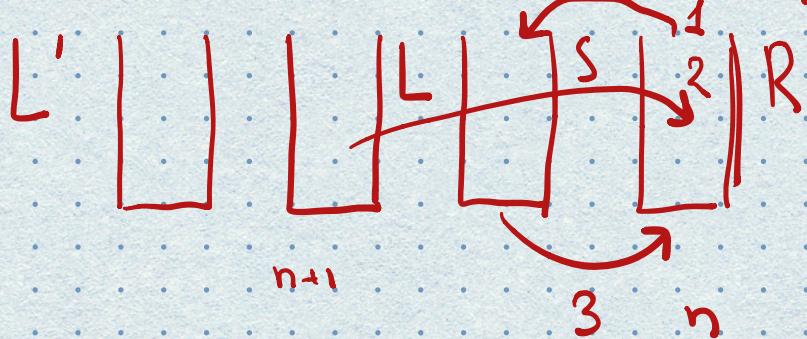
4. Рука L опустошилась, тогда "роблема".



Возможно, мы 2-го не достали из R и у нас 2 стека, откуда надо доставать.

Если нужно сдвинуть новое перекопирование, то понадобится новый R'' . — иначе,

5. Поменяем перекопирование.



Нужно 3-я операция.
Сделаем сделав №3
операцию 3-я раз.

6. Но как тогда удалять?

R_C — конец стека R на момент начала перекоп.

$R_C = R$ в базовом режиме: досчит 2nd-ую R
и из R_C токе.

В режиме перекоп. берешь из него эти адреса

R_C' — базовому режиме идёшь

В конце перекопирования $= R$ и в конце $\rightarrow R_C$

На самом деле R_C в конце перекоп. не пустой
и нужно сделать так, чтобы R_C начало
перекоп. Он становится пустым.

7. После перекопирования в R захватываются данные.

т.к. досчит из R_C

потому во время работы нужно удалять

записи из R токе

8. В R_c скончалась корова после перекопа. Поэтому при каждом операторе нужно так же указать R_c . Важно!, почему мы упомянули это описание до следующего перекопирования?

Нужно было временно хранить. Так что

x push и $n-x$ pop.
тогда после перекопа:



$$\text{size}(L) = x$$

$$\begin{aligned} \text{size}(R) &= 2n+1 - (n-x) = \\ &= n+1+x \end{aligned}$$

$$\begin{matrix} U & H_n \\ R'_c & R_c \end{matrix}$$

$$\text{size}(R_c) = n - (n-x) = x$$

следующий перекоп. Будет ему $\text{size}(L) > \text{size}(R)$

то есть минимальный разрез $n+1$ запас.

то есть R_c уменьшит описание ($x < n$)