

# Формулировка задачи

Реализовать библиотеку со следующими структурами данных в persistence-вариантах:

- массив (константное время доступа по индексу, переменная длина)
- двусвязный список
- ассоциативный массив (на основе hash-таблицы)

Требования:

- покрыть библиотеку компонентными тестами
- библиотека должна быть документирована и к ней должна быть автоматически сгенерирована документация

документация

- единый API для всех структур
- реализовать универсальный undo-redo механизм для перечисленных структур
- реализовать более эффективное по скорости доступа представление структур данных, чем fat-node

## Предполагаемое решение

Библиотека будет реализована на языке C++. Типизация будет осуществляться посредством template. Для сборки будет использоваться CMake. Генерировать документацию будем с помощью Doxygen.

### Общий API

Collection undo() const

Collection redo() const

операции, изменяющие коллекцию, возвращают новую версию коллекции

итераторы пользователь может получить при помощи методов begin() end()

### Undo/redo

При модификации в undo стек добавляется действие. Для оптимального хранения реализован персистентный стек.

### Лучше чем fat node

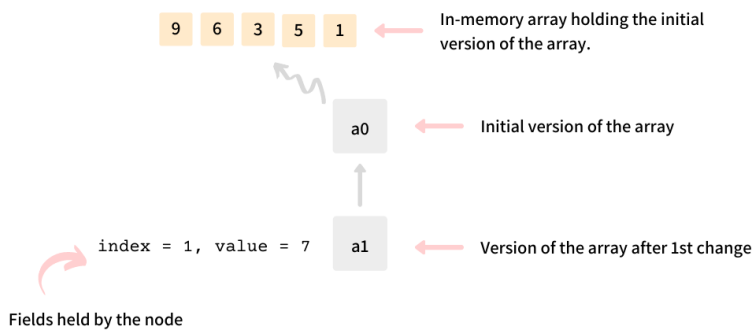
#### Массив

Для персистентного массива будет реализован алгоритм из статьи

<https://arpitbhayani.me/blogs/fully-persistent-arrays>.

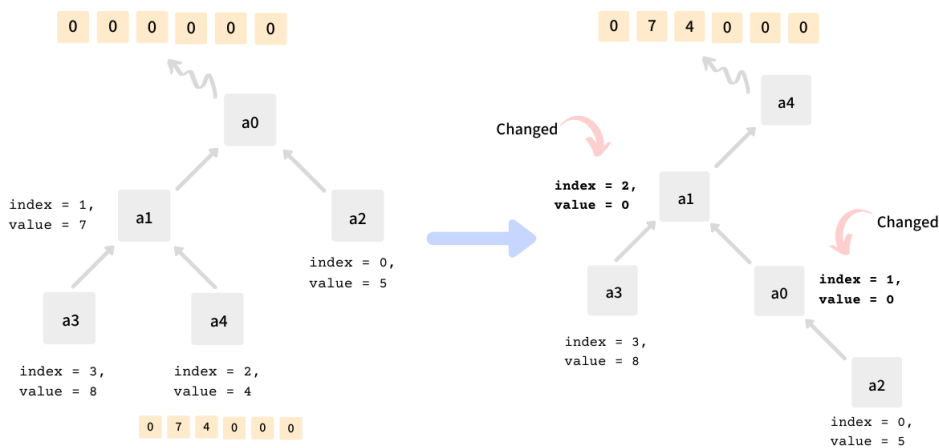
Будем хранить "корневой" массив root\_arr, дерево изменений change\_tree для него и массив версий ver\_arr. В массиве версий будут храниться указатель на узел дерева изменений, соответствующий данной версии. Узел дерева изменений хранит "изменение", которое было сделано в этой версии, и указатель на узел предшествующей версии.

При обновлении просто добавляем новый узел везде, при возврате значения пробегаем по change\_tree с нужного узла до тех пор, пока не встретим изменение элемента с нужным индексом или не дойдём до корня.



## Making first change to the array

Для того, чтобы get работал быстро, время от времени подвешиваем за новый корень.



## Rerooting

### Список

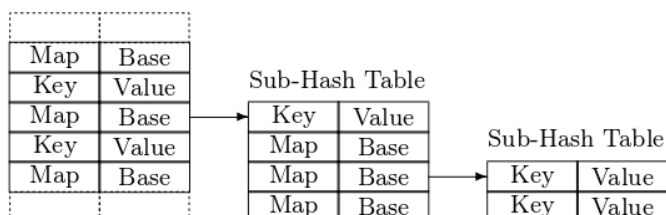
Используем объединение fat node и copy-on-write. В каждой ноде хранится ограниченное число элементов, если нужно добавить ещё, создаётся новая нода. Подробнее этот метод описан в лекции ComputeScienceCenter <https://www.youtube.com/watch?v=cSiiVofy2Tw>. Там же объясняется, почему это эффективнее, чем fat node.

### Ассоциативный массив

Для ассоциативного массива используется персистентная версия из статьи <http://lampwww.epfl.ch/papers/idealhashtrees.pdf>

Для поддержания персистентности в случае если при добавлении/удалении элемента структура была изменена, то создается копия для всей цепочки до самого корня (path copying). Поддержание персистентности реализовано по аналогии с реализацией HAMT в стандартной библиотеке Clojure.

Root Hash Table



# Календарный план

сроки	полученная функциональность	разделение ответственности
ноябрь	персистентные структуры данных	Галя - list Никита - array, map
12 декабря	undo/redo	Никита - общий механизм, встраивание его в array, map Галя - встраивание в list
18 декабря	компонентные тесты, документация	Галя - list Никита - array, map