

Persistent Data Structure

Nattee Niparnan

What is?

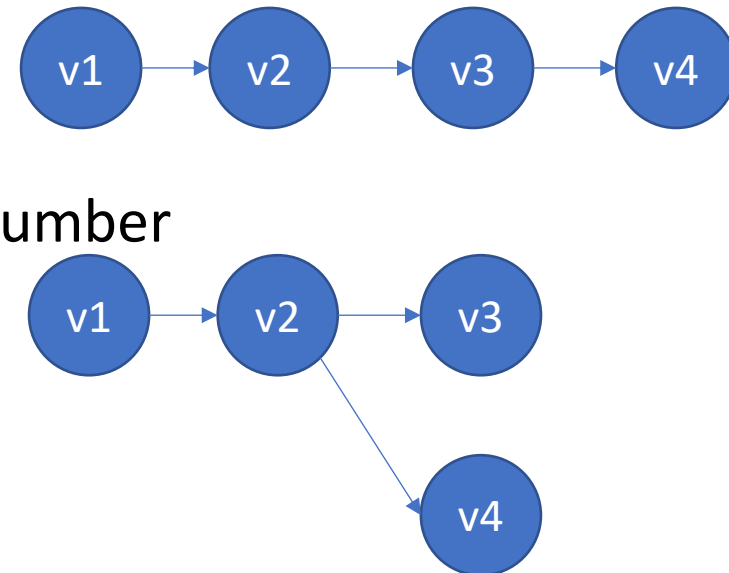
- Traditional data structure is ephemeral
 - When you modify it, it change.
 - The old data is lost, there is no obvious way to backtrack the edit (in most cases)
- Persistent Data Structure is a data structure that allow us to see back in time
 - Time is identified by version number
 - Partial persistent: can read back in time
 - Full persistent: can read/write back in time, creating fork of versions

When to use Persistent?

- Update – query style problem
- Where query is NOT known beforehand
 - E.g., interactive task
- If query is known, it is better to “sweep” through time
 - E.g., sort query, update by times
- Most of the time, it is not that obvious

Example Interface

- Consider array of size n, A
- Normal array of int
 - Read: `int get(idx)` (e.g., `cout << A[x]`)
 - Write: `void set(idx,value)` (e.g., `A[x] = 20`)
- Partial Persistent Array
 - Versioning: `int current_version()`
 - Read: `int get(idx, version)`
 - Write: `int set(idx, value) //return version number`
- Full persistent Array
 - Write: `int set(idx, value, version)`
 - Versioning: `int previous_version(version)`



Naïve Implementation

- Use 2D array to store 1D Array
- The other dimension is “version”
- Full-Copy of each version

operation	Ver	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
Init	0	0	0	0	0	0	0	0
Set(3,1)	1	0	0	0	1	0	0	0
Set(5,4)	2	0	0	0	1	0	4	0
Set(2,3)	3	0	0	3	1	0	4	0
Set(3,5)	4	0	0	3	5	0	4	0
Set(5,9)	5	0	0	3	5	0	9	0

Naïve Implementation

- Set = $O(n)$
- Get = $O(1)$
- Space $O(n)$ per operation
 - For m operation, it's $O(nm)$
- This approach can be done in most data structure

Today Topic

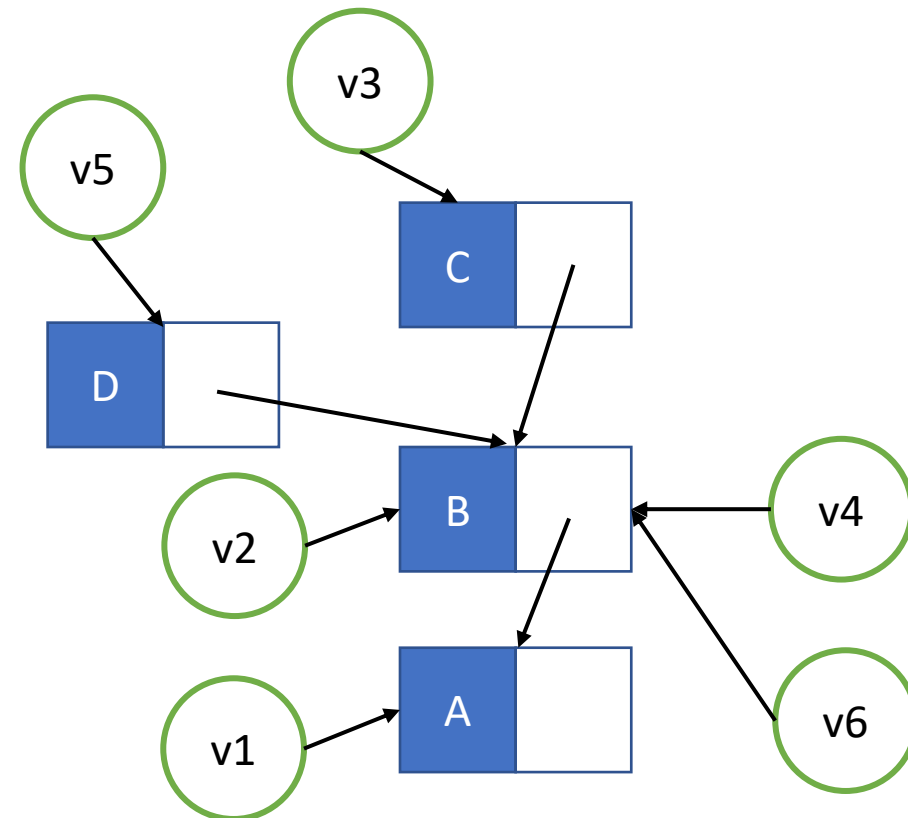
- How to make “faster” and “smaller” persistent data structure

Persistent Stack

- Idea: Pointer based (can use vector)
- Interface:
 - Create new version: push, pop
 - No change: top
- Each element in the stack contains a pair <data, next-to-top>
- Version = a pointer to the top

Example

- Sequence of operation
 - Start with version 0 = empty stack
 - Push(A) → version 1
 - Push(B) → version 2
 - Push(C) → version 3
 - Pop() → version 4
 - Push(D) → version 5
 - Pop → version 6
- $O(1)$ push, pop, top
- $O(1)$ space per push, pop



Code

Data:

```
vector<pair<T,int>> nodes;  
vector<int> tops = {-1};  
int current_version = 0;
```

top:

```
T& top(int version) {  
    return nodes[tops[version]].first;  
}
```

push:

```
int push(T value) {  
    nodes.push_back( {value, tops.size()-1 } );  
    tops.push_back(nodes.size()-1);  
    return ++current_version;  
}
```

pop:

```
int pop() {  
    int tos = tops[current_version];  
    tops.push_back(nodes[tos].second);  
    return ++current_version;  
}
```

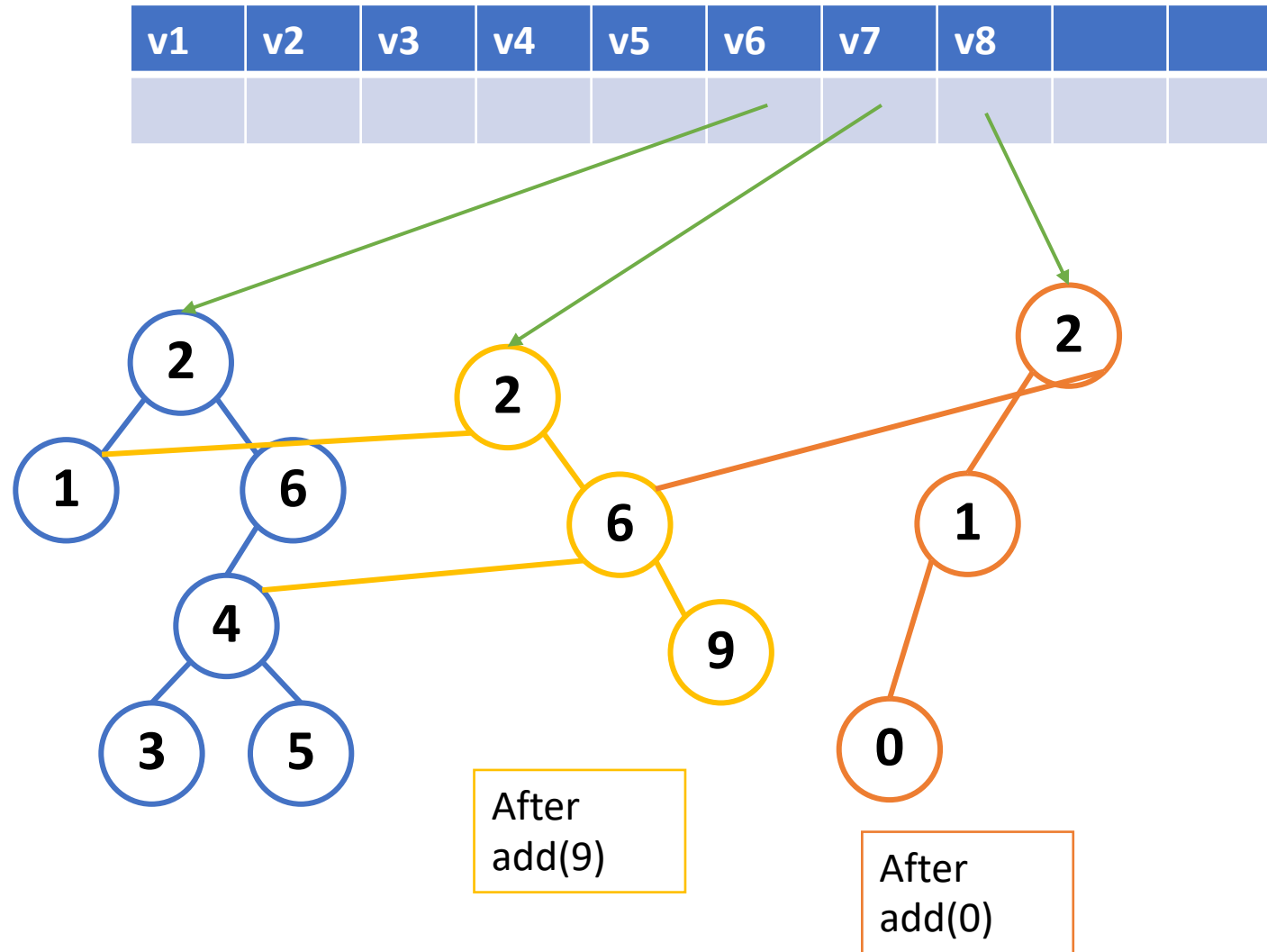
Persistent Queue

- Like a stack
- Each version keep <front,back>
- Enqueue, dequeue add new pairs of <front, back>

Persistent BS Tree

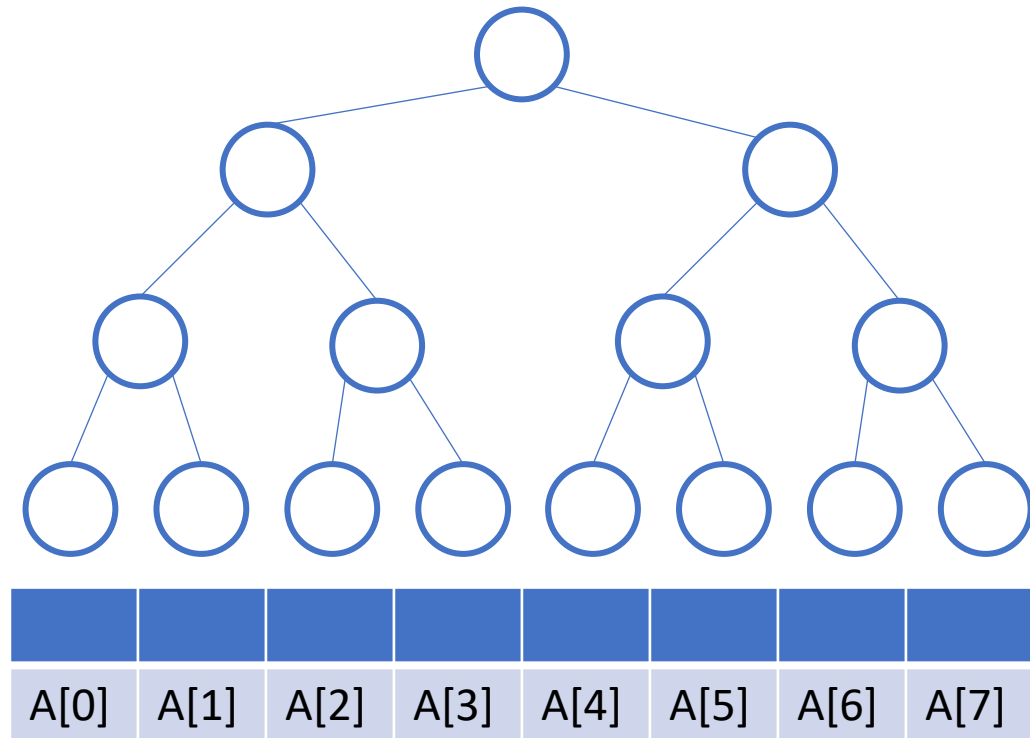
- Each version has different root
- When add, create a new version of nodes from the new leaf to the new root
- Each insert cost additional $O(h)$ (to the original $O(h)$) in times where h is the height of the tree
 - Also add $O(h)$ space

Example



Persistent Array

- Use tree to store array
- Internal nodes covers some part of array (in segment tree fashion)
- Update in the same way as BST



Implementation Summary

- Queue, Stack use pointer-based
- Use “Segtree Structure” (as in Persistent Array) if underlying data structure is Array
 - Such as Array, Fenwick Tree, Priority Queue, Open Addressing Hash
 - Also Disjoint Set, Queue, Stack can be done this way
 - Need additional $O(\log W)$ where W is the size of the array
- Use “Tree Structure” (as in Persistent BST) if underlying data structure is a Tree
 - BST
 - Generalized Seg Tree

Useful Link

- Link from GfG <https://www.geeksforgeeks.org/persistent-data-structures/>
- Good Intro <http://www.toves.org/books/persist/>
- Purely Functional Data Structure <https://www.cs.cmu.edu/~rwh/theses/okasaki.pdf>

Some Interesting Problem

- CodeForces Sign on Fence
<http://codeforces.com/problemset/problem/484/E>
- SPOJ K-th Number
- SPOJ Count on Tree
- CodeForces The Classic Problem
 - <https://codeforces.com/problemset/problem/464/E>