⇒ factoring matrices in different ways can lead to new insights about how matrix multiplication works

Matrix vector multiplication

Suppose $A \in M_{m,n}$ has cols $a_1, a_2, \ldots a_n$ and $V \in R^n$ is a column vector. Then

$$AV = v_1 a_1 + v_2 a_2 + \ldots + v_n a_n$$

Temporal

Persistent Data structures → ① keep all versions of DS
→ retroactivity

→ we have update opeans and query operations. We are mainly concerned about updates here.

→ every time you do an update, you think of it as taking a version of the data structure and making a new one. And you never want to destroy old versions.

→ when you change the DS, we want to remember the past data as well.

② All ds operations are relative to a specified version.

③ so an update makes and returns a new version.

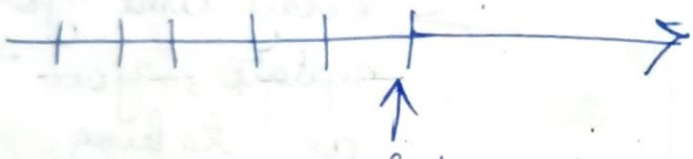
→ So when you do an insert you specify a version of your data structure and the thing you want to insert. And the output is new version. Then you could insert into that new version, keep going, may be go back to the old version, modify that

4 levels of persistence

① partial persistence

→ only allowed to update the latest version which means the versions are linearly ordered.

→ easiest to obtain



We have a time-line of various versions on it.

Suppose I have a new version

Then our latest is this

U can still ask qns about the one old version and able to search on any of these data structures. But you can't change them.

U can only change the latest ver

② full persistence :-

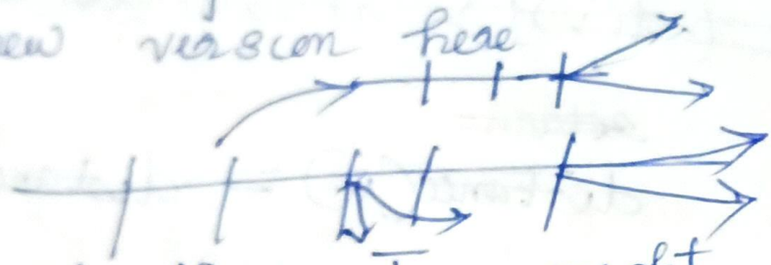
In full persistence, you can update anything you want

↳ update any version

Then the versions form a tree.

So in this model, may be u have nice line of versions

But now I if I go back to this version and update it, I branch, get new version here



and then I might keep modifying that versions some times. Any of the guys can branch.

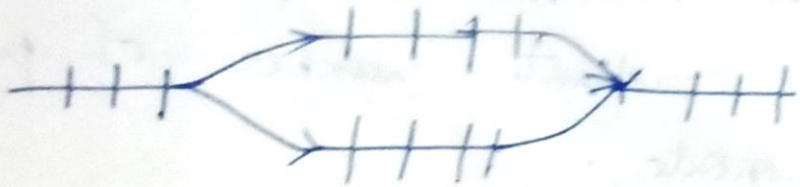
So it is called branching universe model.

③ confluent persistence :-

→ combine 2 versions to create a new version.

→ In this version u cannot destroy old versions.

→ the versions form a DAG, Directed Acyclic Graph.



eg:- we can combine a linked list & BST.

④ functional persistence:

→ difficult to incorporate in real world problems.

→ never modify nodes.

→ only make new nodes.

As long as we still represent the same data in the old versions, we don't have to represent it in the same way.

That let's do things more efficiently in other persistence.

In functional persistence, all the old versions in exactly the way you used to represent them.