BINARY SEARCH TREE AND ITS BALANCED VERSION

HAOCHEN

1. Motivation example

In an airport with only one runway, we need to schedule the reservation of landing.

- Specify the request landing time t.
- To add t, there must be no other landing scheduled within k minutes.
- After landing, it will be removed from system.

Example 1. Example needed.

Our goal is to run the system in $O(log(n))$ time. If we take a close look at the problem, we know we need to deal with some operations such as searching, comparing, inserting, deleting. The latter two involve the whole list and therefore is quite different and need to notice.

2. Simple data structure.

Definition 2. A **data structure** is a collection of algorithms for storing and retrieving information.

- The operation for storing information is called **updates**, like insert, delete.
- The operation for retrieving info is called **queries,** like search, find_max, find_next.

The salient property of data structure is its **representation invariant**. the queries must be correct if this property holds, and updates should preserve this property. Often, we use check when debugging a data structure, it may cost $O(n)$ time, more than operations to be checked, usually $O(1)$.

Now let's introduce some simple data structures. Some thought before is that these invented form is usually borned for practical uses, like goals in computer tasks. So they are functionalized just like the art form from the primative society. The conflict between function and form is much less conflict than those in the architecture context. The form is itself devised to function properily.

Then how it can be related to culture? If can't, then what's the internal good?

Enough.

2.1. Stack.

like stack in reality, do operation insert and delete from tail only.

2.2. Queue.

Like queue in reality, do operation insert from tail and delete from start.

3. Binary search tree

Enough basics, back to our previous scheduling problem. Notice this is a sorted list, so queries can be achieved in $O(log(n))$, heigth of the tree, but insert may take $O(n)$ because of the duplication. For other easy insert data structure like dictionary and min_heap, the check of k min takes $O(n)$.

But a new invention comes: binary search tree (definition omitted).

Claim 3. In (ideal) binary search tree, insert only take $O(log(n))$ time.
property:

- each node has: key, left pointer, right pointer, parent pointer.
- BST property: left is small, right is large.

Claim 4. proof.

But the problem is that tree in practice is not always has height $log(n)$, in another word, balanced. Here is a counterexample:

Example 5. suspended.

The first problem is to define what is actually 'balanced', then how to achieve balance.
One approach from predessor's work is called **AVL trees** devised in 1962. Basically, they invent an operation which preserve RI, but head to balance is known as rotation.

4. AVL trees

Clarify the problem:

- To define the notion balanced, we need to detect the height in local place. Specifically, each node.
- we put the constraint : only allow one difference.
- when this produce the worst case, still in O(log(n)).

tackle the problem:
Rotation, 1,just change pointers, so O(1), 2, satisfy property
case1
case2,zigzia
Algoth:
1, BST insert
2, fix AVL constraint, change node up.
Also sort O(nlog(n))