LEC4 Hashing

https://github.com/GUMI-21/MIT6.006_note
3.17

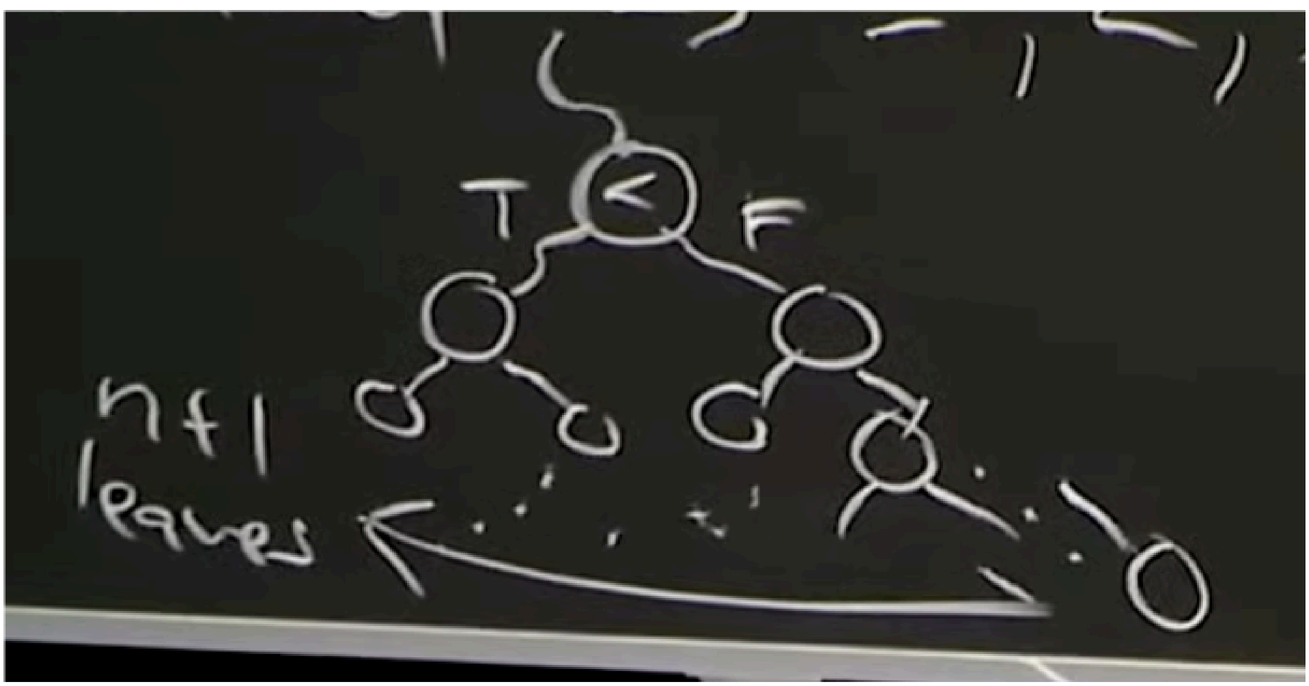
Last lec review

Data Structure	Container	Static	Dynamic	Order	
	build(A)	find(k)	insert(x) delete(k)	find_min() find_max()	find_prev(k) find_next(k)
Array	n	n	n	n	n
Sorted Array	$n \log n$	$\log n$	n	1	$\log n$

find(k) takes $\log(n)$ time for sorted array.

Comparison Model

$=, <, >, <=, >=, !=$

- *a **binary tree** can represent the comparisons done by an algorithm*
  it has been n+1 leaves in search alogrithm binary tree. (n is number of items in set).
  because it needs store n item in leaves and one false return case.
  

- *compare times of search algorithm*
  in the worsest case, the compare times of alogrithm is the height of this binary tree.

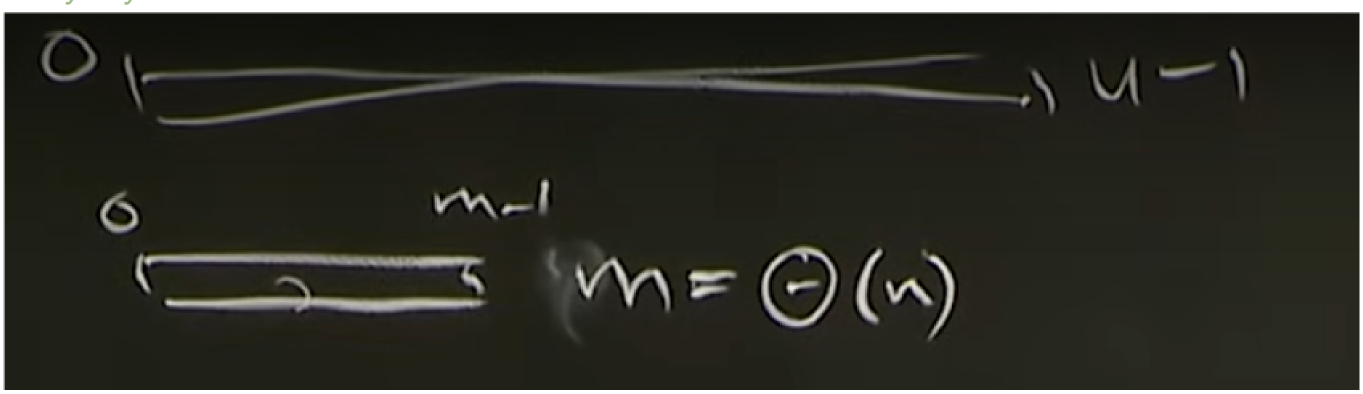
so the question beacomes how can we make the tree with n+1 leaves be *minimum height*.
min height is $\theta(\log(n))$

Direct Access Array

direct store item in one memory index place. *means store all items in a word length array, so RAM can random get every item in O(1), and the search runtime is O(1)*
then u-> largest key. $u < 2^w$,w is the word size of machine,
example: 64byte w: can random address 2^64 byte address in memory one time.
一个 64 位 CPU 可以使用 64 位地址来访问内存中的任何位置。
and only integer key, what means we can only store integers.
the search run time is $O(1)$, but use a log of space in one RAM process. How to use less space ? use hashing.

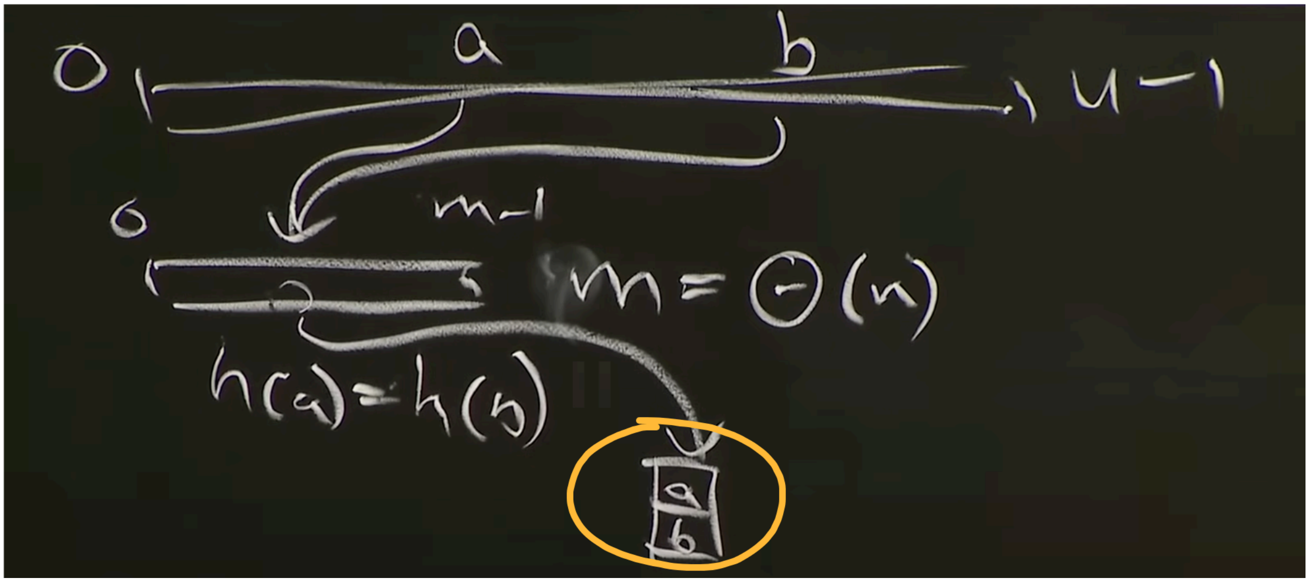
Hashing

for store n items, use m length space to store key, $m = \theta(n)$.
n items is store in memory, and m is very short so that make sure our cpu can random access every key in once.



$h : \{0, 1, \ldots., n-1\} -> \{0, \ldots, m-1\}$ *make the key space into a compressed space*

- *problem*
  there will be like $h(a) = h(b)$, one more values map to one key.
- *solution*
  when I have *collision*, go to the datastruct *chain* associated to the index, then do linear opeartion to see is the item I search in there.



The point of this lecture is to find a hash function that make sure the chain datastructs are very small.

hashing functions

Division hash function

$h(k) = k\, mod m$ this is essentially what Python does.
but this function will make chain datastruct going to be large at a single hash, but it is a deterministic has function (means every time the program run, it's going to do the same thing underneath)

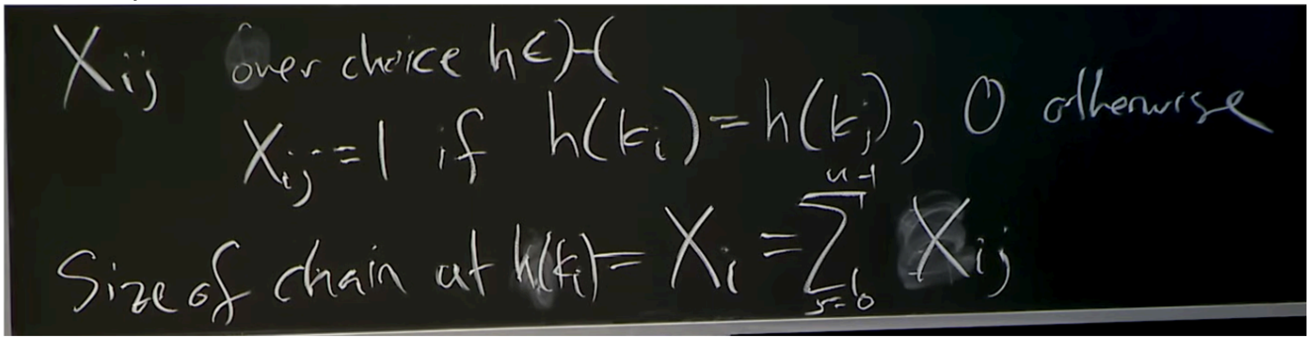
Using universal hash function

satisfy *universal hash property*
$h_{ab}(k) = (((ak + b) mod\ p) mod\ m)$
$H(p, m) = h_{ab}(k) | a, b \in \{o, \ldots, p-1\}\ and\ a! = 0$
I have a hash family. p is a large prime number I picked according to the length of hashtable m, and will be fixed when I make the hash table. And when I new a hash table will choice a hash function in family by random choice a & b.
-> every *Instantiate* a hash table, random a & b mod p to be a hash function. So it's really hard to give a bad example which goes to be a large chain datastruct.

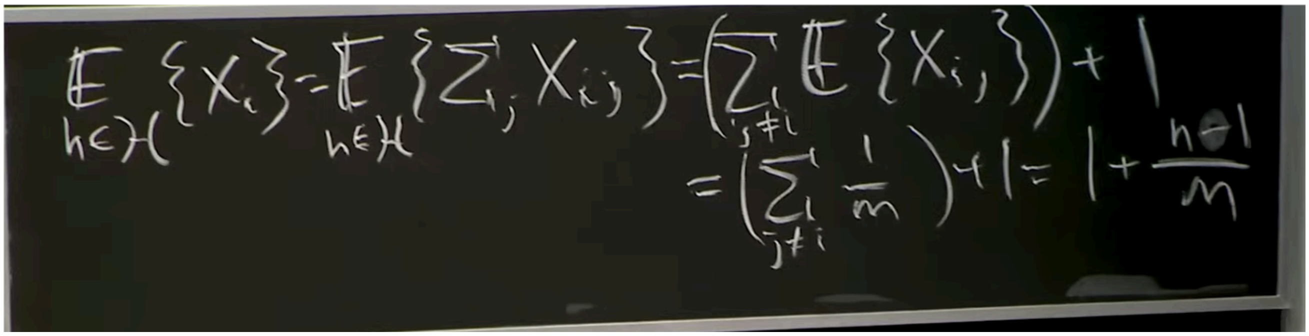
- *Universal property*
  $P_r(probility) : h\ \in H, \{h(k_i) = h(k_j)\} <= 1/m,\ \forall k_i\,! = k_j\ \in \{0 .. . n-1\}$
  means for any keys that I pick in my universe space, if I randomly choose a hash function, the probility that these things collide is less than 1/m.
  So the problem becomes we want to prove the hash family satify the probility upper.

- *proof of the chains is expected to be constant length*

  define X_{ij}



1/m from universal property.



So if we choose n large bigger than m, then Ex_i, the chain will be a constant.

Data Structure	Operations $O(\cdot)$				
	Container	Static	Dynamic	Order	
	`build(X)`	`find(k)`	`insert(x)` `delete(k)`	`find_min()` `find_max()`	`find_prev(k)` `find_next(k)`
Array	n	n	n	n	n
Sorted Array	$n \log n$	$\log n$	n	1	$\log n$
Direct Access Array	u	1	1	u	u
Hash Table	$n_{(e)}$	$1_{(e)}$	$1_{(a)(e)}$	n	n