Hashing implementation

Hashing implementation

→ One up side a dictionary, we have to it number of cells, among cells $1 \le i \le n$ and

perform following operations

→ Insert/Search/Delete.

	Insert	Search	Delete
Unsorted Array	$O(1)$	$O(n)$	$O(n)$
Sorted Array	$O(n)$ ← Shifting to make an space	$O(\log n)$ ← Binary Search	$O(n)$
Sorted Singly LL	$O(n)$ ← find spot + Shift	$O(n)$	$O(n)$
Sorted Doubly LL	$O(n)$	$O(n)$	$O(n)$

Binary Search Tree : $O(h)$ ← Best : $\log N$; Worst : N → BST

BST ← Balanced Binary Search Tree / AVL Trees / Red Black Trees

→ Sorted Tree?

Ex: $1 \to 2 \to 3$ → Right Skewed

B.B.S.T : $\log N$; $\log N$; $\log N$

$\log(N!)$; ↓ ; ↓ ; ↓

DAT : Direct Access Table

Idea with Ignore : There is a set of finite universe

→ If convert to non negative integer data, to an counter index value. Then can use it.

Ex: Index : $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$; $\{23, 32, 45, 14, 12, 77, 108\}$ → All value should be mapped to $\{0-9\}$

DAT (U) :

0	1	2	3	4	5	6	7	8	9
	23	32	45	14	12	77		108	

Ref : $O(1)$

A very simple way to map data

→ Hash Table

→ Hash Table Algo

→ Hash Function

Mapping a value to a counter index value. Using a counter function to convert hashing. The place where we are store this data is called Hash Table.

→ Simple Hash Function

$h = d \bmod m$; $[1, m]$

→ We should always need to $m \ge$ total number

→ We can prime number to avoid collision

Ex: Time Table

→ Collision : $\{E$ different keys → Same Hash Value$\}$ → Hash Collisional by Instruction Index.

Collision Resolving Techniques

1) Separate Chaining

2) Open Addressing
   1) Linear Probing
   2) Quadratic Probing
   3) Double Hashing

Separate Chaining :

data = $\{25, 32, 45, 22, 15, 23, 43\}$

S=10

0	1	2	3	4	5	6	7	8	9
		32			45				
		22			15				
		32			25				
		22			23				
		32			43				
		22							

At every hash index we have a balanced Binary Tree. // Assume we have 7 keys so, a BST

Insert	Search	Delete
$\log(7)$	$\log(7)$	$\log(7)$

→ Every index to hold cells in separate list

Insert	Search	Delete
$O(1)$	$O(7)$	$O(7)$

// P.S. cell → Keys

at the hashing patterns

Concept : Open Addressing

when ever there is collision occur in some available data space.

Step to follow : find available data space, that is should be hashable.

data = $\{10, 20, 30, 55, 16, 15\}$ → Hashing many method

Table S = 10

h(k) = (k%10)

0	1	2	3	4	5	6	7	8	9
10	20	30	55		15				

We are finding to some available slot keys : Linear Probing

// multiple hashfunction

$h(k) = (k + i) \% S$; Initially $i = 0$ If there is a collision we increase i.

$i = 0$: $h(k) \% S = 1$

1 $(h(k)) \% S = 2$

2 $(h(k)) \% S = 3$

3 $(h(k)) \% S = 4$

4 $(h(k)) \% S = 5$

5 $(h(k)) \% S = 6$

Insert : We keep on probing until we get an Free/Deleted

Search : data = $\{11, 21, 25\}$

S=10

0	1	2	3	4	5	6	7	8	9
	11	21	25						

Step to search : 21 → 1 → 2

until we find element or we find an Free slot.

Delete : data = $\{12, 22, 42, 52\}$

0	1	2	3	4	5	6	7	8	9
	12	22	42						

// Delete 22

Insert : 42

2 → 3 → 4

Empty / Deleted Lie Step

Search is not working

Then what you mean of you → {Available / Deleted / Deleted}

To solve above below we use Visited → Make Allow Indexing

Visited[i] → 0 : Not in Insert
→ 1 : Occupied
→ -1 : Deleted

→ Mark Visited Make a 1

Initially Visited[i] = 0

	Insert	Search	Delete
Linear Probing :	$O(1)$	$O(1)$	$O(1)$

P.S. only Probes

Collisions

data = $\{101, 201, 301, 401, 501, 601, 701, 801, 901, 1001\}$

S = 100

h(k) = (k%S)

0	1	2	3	4	5	6	7	8	9
101	201	301	401	501	601	701	801	901	1001

Total Cell : $\frac{S}{2}$

collisions reduced

Quadratic Probing :

1 → $h(k) = (h(k) \% S)$

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
101	201	301						401	501						
		3	2	1											

Total Cell : $\frac{S}{2}$

cell :

Double Hashing :

$h_1(k) = (h_1(k) + h_2(k) * i) \% S$ → collisions with further reduce

// If hash function to accidentally data access unavailable

Ex: If any collisions will reduce

→ If $N = 1000$; $S = 1000$ → Cannot even implement open addressing

N ≥ S : Keys

HLT

$S = 1000$: 45% Collisions Very High

$S = 2000$: Collisions will decrease

50% prior of

$S = 10000$: collisions = 1/5 → 20%

Half percent

→ If N threshold $\frac{S}{2}$

$\frac{S}{2} \Rightarrow O(n)$

Hash Map

Separate Chaining
: Hash Table
: Hash Function
: BBST

Open Addressing
: Based Hash Table Size
: Very Good Hash Function
: Quadratic Probing

→ $h(k) = (h(k) \% S$; $TC : O(1)$

Good Function :

→ Easy to compute also more than $O(1)$

→ It should be complete data. There collisions are less

Ex: Easy $10 \to$ Table^1, Table^2, Table^3

$H_1(k) = (k \% S) \% S$

$H_2(k) = (k \% S) \% S$

P.S. Keys : Hash Function for Easy to Access

Always consider key Counter. (k)

Hash Table

$S = 10$

$T = 6$

// Implementation

```
for int [int] ht;
for int [int] ht;
for int [int] ht;
for S = 100;

for hash (int x, int t) {
    return (x + t*t) % S
}

void insert(int x, int v) {
    for i = 0; i < 100; i++
    while ( ---- ) {
        for int hash = hash(x, i);
        if ( ht[hash] == -1 ) {
            break;
        }
        if ( visited[hash] == -1 ) {
            break;
        }
        i++;
    }
    ht[hash] = k, visited[hash] = 1
}

bool search (int k) {
    for i = 0; i < 100; i++
    while ( ---- ) {
        for int hash = hash(x, i);
        if ( ht[hash] == k ) {
            return true;
        }
        if ( visited[hash] == -1 ) {
            return false;
        }
        i++;
    }
```

Clear :

Separate Chaining

Step : Key

Value : Value

Step : Occupied → -1 : Deleted
→ 0 : Available
→ 1 : Occupied

→ int [100]

D D D D D D D D

→ Key
→ Value
→ Occupied