27/12/2023

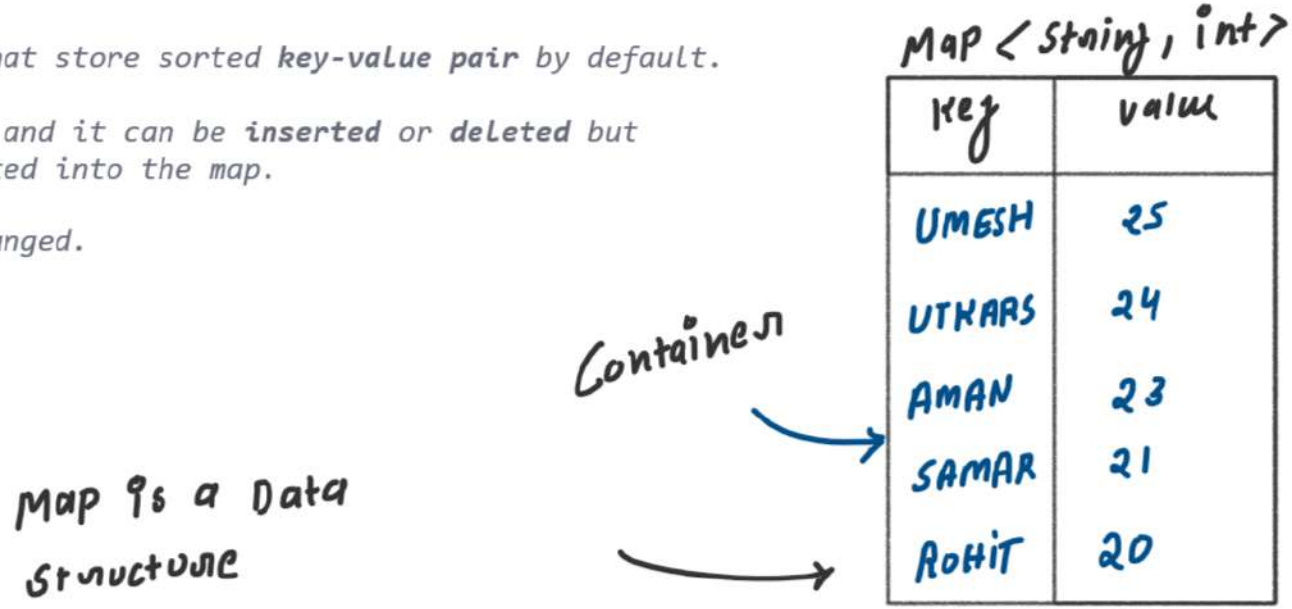
HASHMAPS & TRIES
CLASS - 1

📂 1. What is Maps?

*Maps are the associative containers that store sorted **key-value pair** by default.*

*In key-value pair, each key is **unique** and it can be **inserted** or **deleted** but **cannot be changed** once they are inserted into the map.*

Values associated with keys can be changed.



📂 2. C++ STL Maps Type

I. Ordered Map:
Time Complexity: O(Log N)
Ordered Map is implemented by a ***Balanced Binary Search Tree****.*

II. Unordered Map:
Time Complexity: O(1)
Unordered Map is implemented by ***Array/Hash table/Bucket array***

Note: *Maps is ordered map by default*

3. Implement C++ STL Unordered Map

```cpp
// 3. Implement C++ STL Unordered Map
#include<iostream>
#include<unordered_map>
using namespace std;

int main(){
    // Creation of unsorted map
    unordered_map<string, int> umapping;

    // Creation of pairs to map
    pair<string, int> p = make_pair("Umesh", 25);
    pair<string,int> q("Utkars", 24);
    pair<string,int> r;
    r.first = "Aman";
    r.second = 23;

    // Insertion of pairs into map
    umapping.insert(p);
    umapping.insert(q);
    umapping.insert(r);
    umapping["Samar"] = 22;

    // Size of map
    cout << "Map size: " << umapping.size() << endl;

    // Access value of key using at(key) method and square bracket[key]
    cout << "Value of Umesh: " << umapping.at("Umesh") << endl;
    cout << "Value of Umesh: " << umapping["Umesh"] << endl;

    // Way 1: Searching of key using count(key) method:
    // return 1 when key is found and return -1 when key is not found.
    cout << "Umesh is present: " << umapping.count("Umesh") << endl;

    // Way 2: Searching of key using find(key) method:
    if(umapping.find("Umesh") != umapping.end()){
        cout << "Found" << endl;
    }
    else{
        cout << "Not Found" << endl;
    }

    // Note 1: Jab hum "Using [key]" koi aise key value ko access karte hai jo
    // actual me map ke andar hai hi nhi hai to uss case me ek new entry create
    // ho jati hai jo 0 se belong karti hai like
    cout << "Temp is present: " << umapping["Temp"] << endl;

    cout << "Map size: " << umapping.size() << endl;

    return 0;
}
```

Creation map

Creation of pairs

Insertion

Size = 4

Access

Searching

Access key ⇒ jo Hai nahi Hai

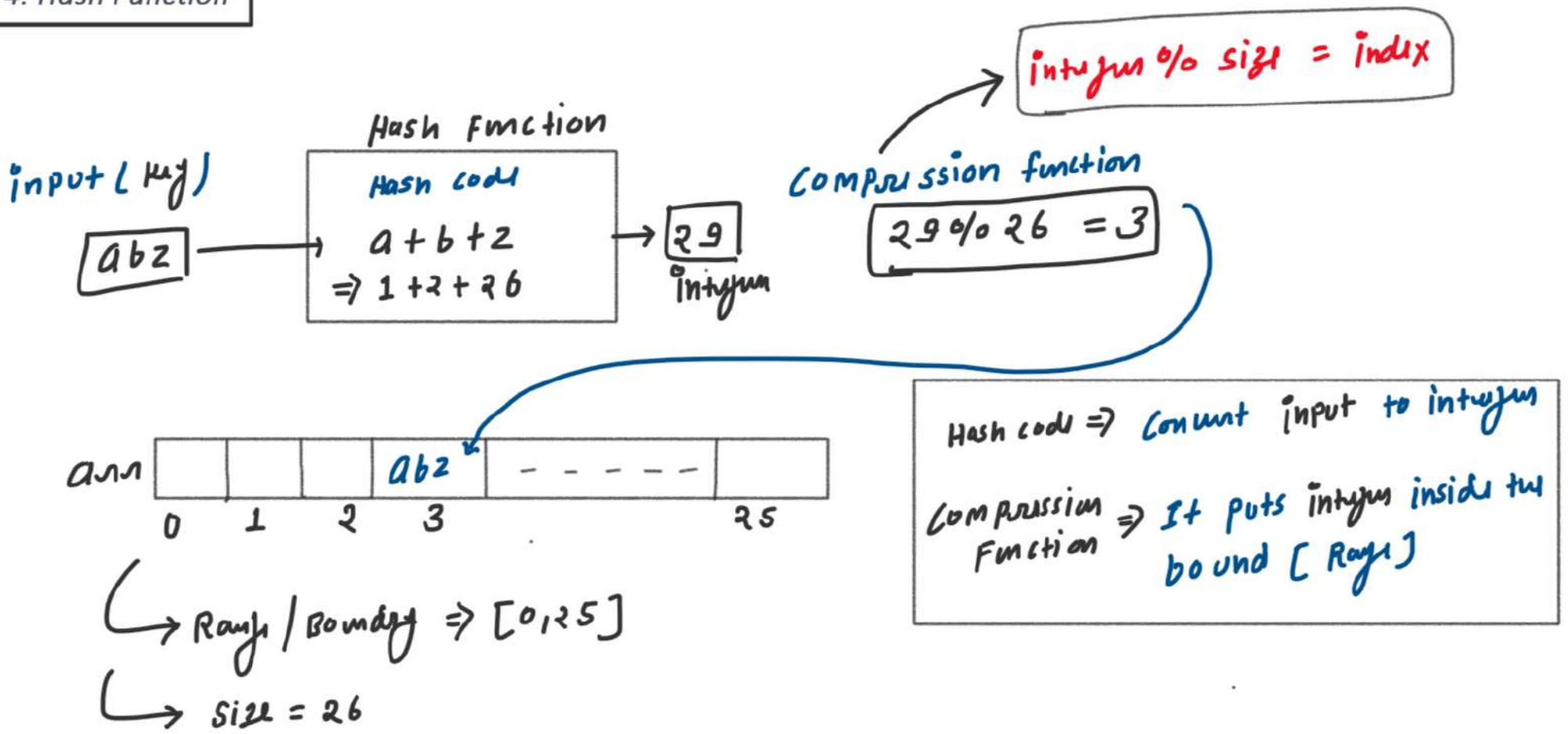
Map < String, int >

key	value
UMESH	25
UTKARS	24
AMAN	23
SAMAR	21
Temp	0

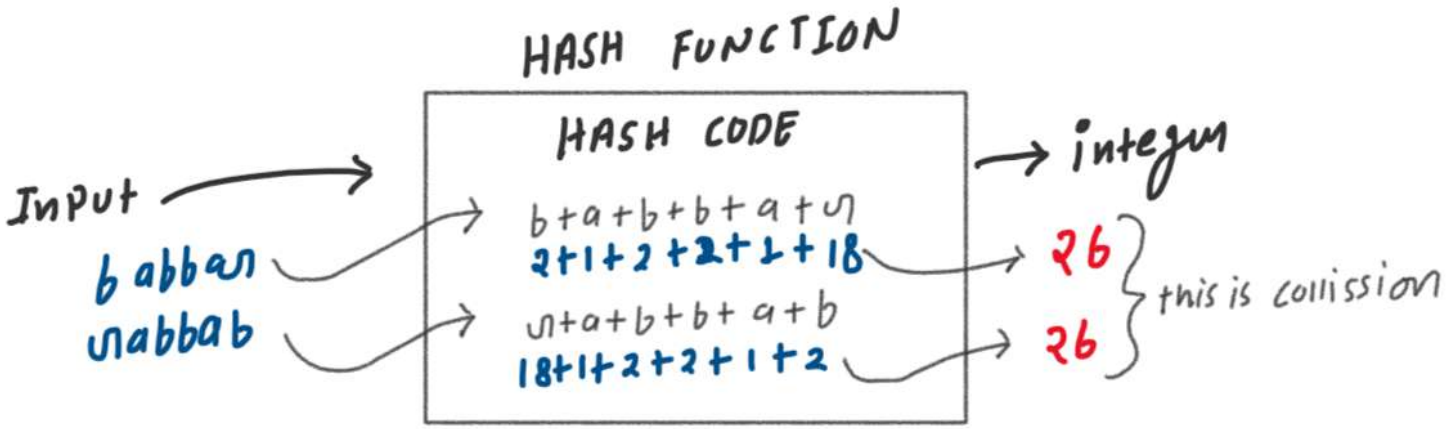
Size = 4

Size = 5

4. Hash Function



Collision



Ex

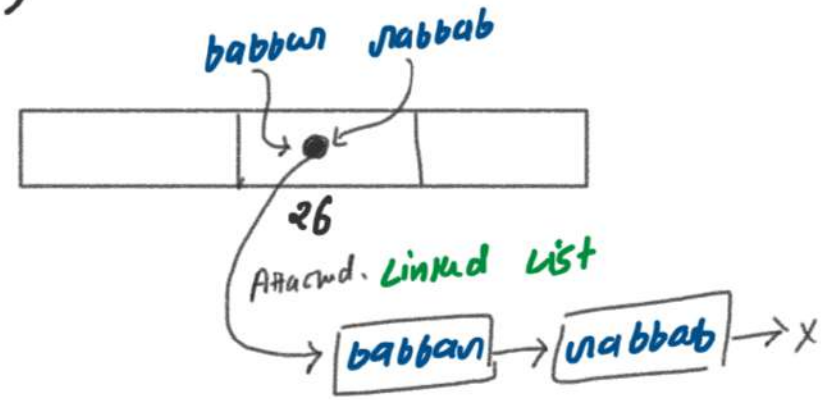
map["AB"] = 3

map["BA"] = 3

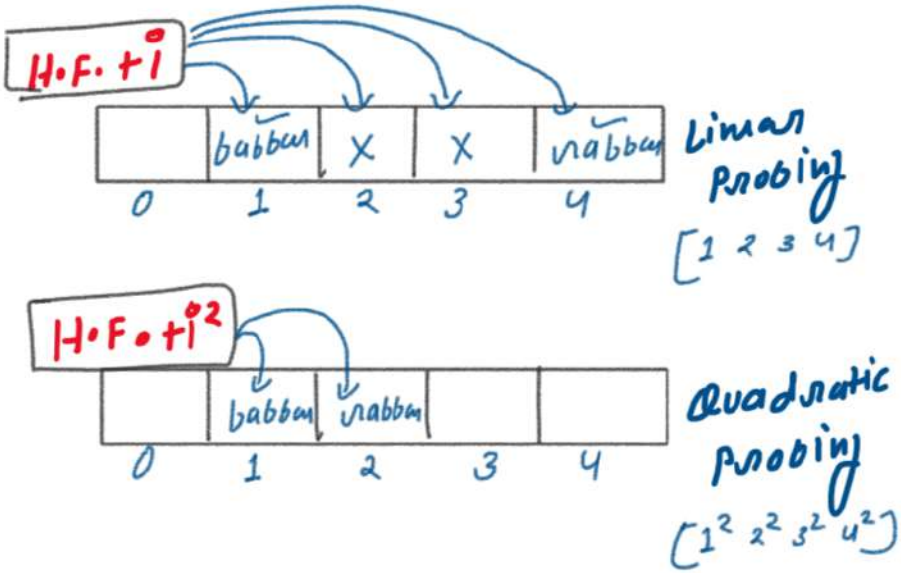
} this is collission

Collission Handling Techniques

a.) OPEN HASHING



b.) Closed Addressing



How and Why the Time Complexity of Insertion, Deletion, and Searching is O(1)

Time complexity

insertion, deletion, searching } O(K) ≡ O(1)

Usually

N >>> K

where

N = Largest size

K = 26

HASH Function

babbar → Hash code → 26

arr[26] = babbar

Insert with T.C. O(K)

Load Factor

$$\text{Load Factor Ratio} = \frac{N}{b}$$

Where

N = Number of elements

b = Free boxes

[ratio $< 0.7 \Rightarrow$ According to case study Good H.F. Aj lo.]

📂 5. Basic Problem on Map

Input string str = "lovebabbar";

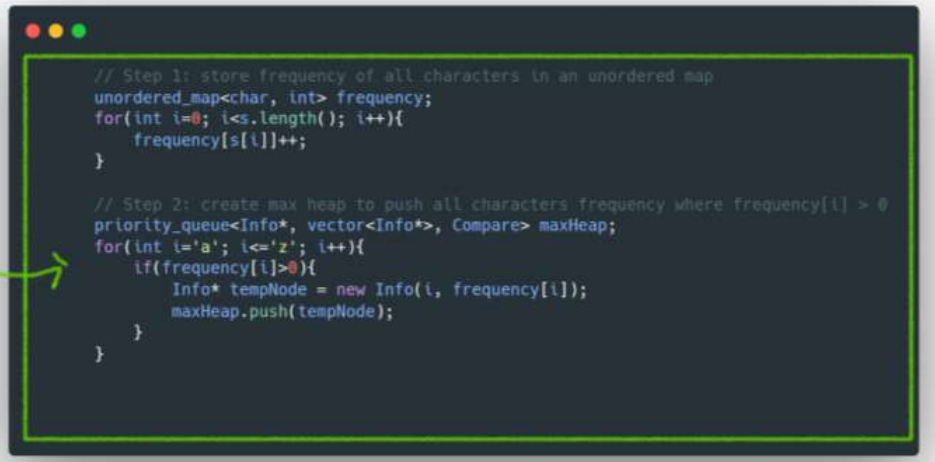
Output

l	- 1
o	- 1
v	- 1
e	- 1
b	- 3
a	- 2
r	- 1

```cpp
// Basic problems on maps

#include<iostream>
#include<unordered_map>
using namespace std;

void countCharacter(unordered_map<char, int> &mapping, string str){
    for(int i=0; i<str.length(); i++){
        char ch = str[i];
        mapping[ch]++;
    }
}

int main(){
    string str = "lovebabbar";
    unordered_map<char, int> mapping;
    countCharacter(mapping, str);

    for(auto i: mapping){
        cout << i.first << " -> " << i.second << endl;
    }

    return 0;
}
```

Problem 2

```cpp
// Problem 2: Reorganize String (Leetcode-767)
(Understand Logic Building in Class No: 47)

class Solution {
public:
    string reorganizeString(string s) {

        // Step 1 and 2

        // Step 3: reorganize the string
        string ans = "";
        while(maxHeap.size() > 1){

            Info* first = maxHeap.top();
            maxHeap.pop();
            Info* second = maxHeap.top();
            maxHeap.pop();

            ans.push_back(first->ch);
            first->count--;
            ans.push_back(second->ch);
            second->count--;

            if(first->count > 0){
                maxHeap.push(first);
            }
            if(second->count > 0){
                maxHeap.push(second);
            }
        }

        if(maxHeap.size() == 1){
            Info* last = maxHeap.top();
            maxHeap.pop();

            ans.push_back(last->ch);
            last->count--;

            if(last->count > 0){
                return "";
            }
        }

        return ans;

    }
};
```

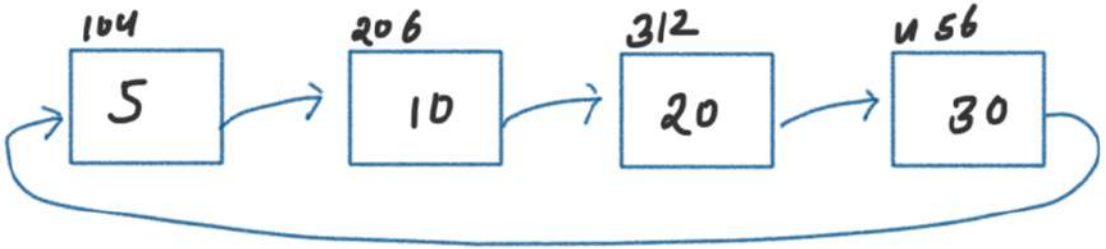


```cpp
// Step 1: store frequency of all characters in an unordered map
unordered_map<char, int> frequency;
for(int i=0; i<s.length(); i++){
    frequency[s[i]]++;
}

// Step 2: create max heap to push all characters frequency where frequency[i] > 0
priority_queue<Info*, vector<Info*>, Compare> maxHeap;
for(int i='a'; i<='z'; i++){
    if(frequency[i]>0){
        Info* tempNode = new Info(i, frequency[i]);
        maxHeap.push(tempNode);
    }
}
```

Problem3 Leetcode-141

Linked List is circular or not?

Ex1



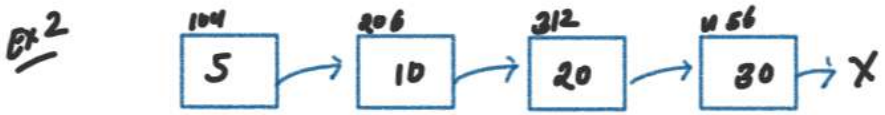
Output TRUE

MAP <Node*, bool>

Address	bool
104	True
206	True
312	True
456	True

Cycle present

```cpp
// PROBLEM 3: Linked List Cycle (Leetcode-141)

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {

        unordered_map<ListNode*, bool> mapping;
        ListNode* temp = head;

        while(temp != NULL){
            // Visited Address
            if(mapping.find(temp) != mapping.end()){
                // Visited address found-> cycle present hai
                return true;
            }
            // Not Visited Address
            else{
                mapping[temp] = true;
            }

            // Update temp
            temp = temp->next;
        }

        // Cycle is not present
        return false;
    }
};
```



Output Fail

MAP <Node*, bool>

Address	bool
104	True
206	True
312	True
456	True

No cycle

Present