Todays Content

1. Min stack
2. Max Frequency stack

Implement a stack with following operations.

1. push() 3. peek() : Return top element
2. pop() : Delete top element 4. getMin() : Return min ele of stack

Note: Each operation should take O(1)

Ex: 10 6 getMin() 12 4 getMin() ↑ getMin()
6 4 6

Trace:

Idea1: Track min with variable & update it, for push & pop()

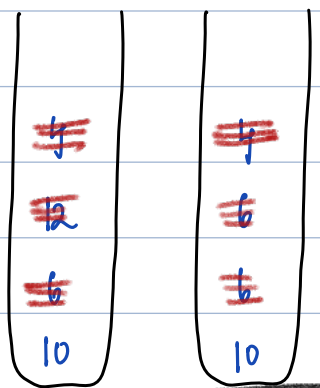
10 6 getMin() 12 4 getMin() ↑ getMin()
6 4 4

min = ~~∞~~ ~~10~~ ~~6~~ ~~4~~

#Issue = When we delete top() element, we cannot update min value.

Idea2: Create another stack, to keep track of min till that point.

10 6 getMin() 12 4 getMin() ↑ getMin() ↑ getMin() ↑
6 4 4 6 12 6 6



st1 st2 : Stores min value, till that point.

#Note: A stack where data inc or dec is called monotonic stacks

```
stack<int> st1, st2;
class MinStack {
    void push(int x) {
        s1.push(x);
        if (s2.size() == 0) {
            s2.push(x);
        else {
            int val = min(x, s2.top());
            s2.push(val);
        }
    }

    void pop() {
        if (s1.size() == 0) {
            return; # Depends on question.
        }
        s1.pop();
        s2.pop();
    }

    int peek() {
        if (s1.size() == 0) {
            return -1; # Depends on question.
        }
        return s1.top();
    }

    int getMin()
        if (s1.size() == 0) {
            return -1; # Depends on question.
        }
        return s2.top();
    }

    int size() {
        return s1.size();
    }
}
```

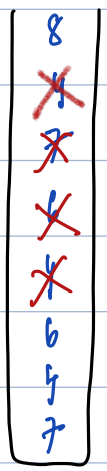
Implement a stack with following operations

1. push()
2. pop() : Removes & return most frequent element in stack.
   If there is tie for most frequent element, element closest to the top is removed and returned.

Note: Each operation should take O(1)

Ex: 7 4 6 4 6 7 4 8 T T T T T

4 7 6 4 8



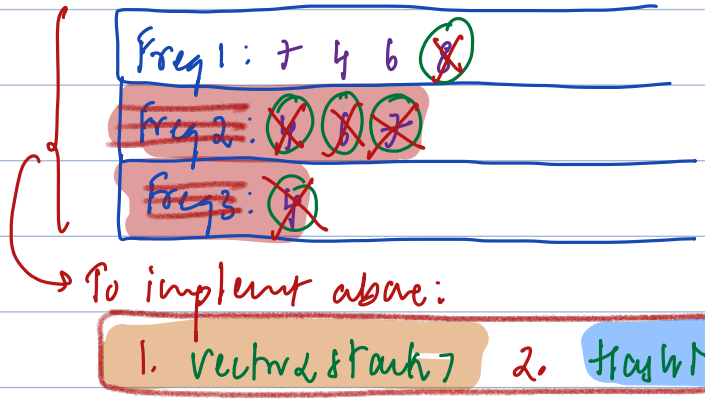
Hint: If there is tie for most frequent element, element closest to the top pos is removed and returned.

stack idea

Catch: For every frequency we need a stack.

Eg: 7 4 6 4 6 7 4 8 T T T T T

4 7 6 4 8



HM:

7 : x x 1

4 : x x x x 1

6 : x x 1

8 : x 0

To implement above:

Steps: 1. vector<stack> 2. HashMap<int, stack>

push(x):

```
hm[x]++; # The freq of x in hashmap;
int f = hm[x];
if freq f stack doesn't exist:
    Create freq f stack
goto freq f stack & Insert x
```

popC):

```
highest frequency hf = number of stacks.
from frequency hf stack
    int ele = get top from hf stack
    pop() from hf stack
    if( hf stack size == 0){
    } Delete hf stack
hm[ele]--; # Reduce freq of ele in hashmap by 1.
```

```
unordered_map<int, stack<int>> sm; #stack map
unordered_map<int, int> fm; #frequy map

class maxFreqStack {

    void push(int x){
        fm[x]++;
        int f = fm[x];
        if( sm.find(f) == sm.end() ){
            #frequency stack f is not present.
            stack<int> tmp;
            sm[f] = tmp; # Insert stack f in stack map
        }

        sm[f].push(x); # Insert ele x in stack f
    }

    int pop() {
        if(sm.size() == 0){ return -1; }
        int f = sm.size();
        int ele = sm[f].top();
        sm[f].pop();
        if( sm[f].size() == 0 ){ # stack f is empty
            sm.erase(f); #Removing stack f from hash map
        }

        fm[ele]--; #Reduce freq f ele in hashmap
        if( fm[ele] == 0 ){
            fm.erase(ele);
        }
    return ele;
    }
}
```