Todays Content : Strength 21

1. Lexicographical smallest string
2. Implement queue using stack
3. Amortized Analysis

{ Need another monotonic stack
Instead of this for future problems }

18 Remove Duplicate Letters

Given a string S, remove duplicate letters so that every letter appears only once and only once.
You must make sure that your result is smallest in lexicographical order among all possible results.

Ex:

S : b c a b c

~~b~~ ~~c~~ a b c : a b c #ans

~~b~~ c a b ~~c~~ : c a b

b ~~c~~ a ~~b~~ c : b a c

b c a ~~b~~ ~~c~~ : b c a

S : c b a c d c b c

0 1 2 3 4 5 6 7

~~c~~ ~~b~~ a c d ~~c~~ b ~~c~~ = a c d b #ans

0 1 2 3 4 5 6 7

~~c~~ ~~b~~ a ~~c~~ d ~~c~~ b c = a d b c

S : c d f b e f a f c e a d

Dry Run:

S : c d f b e f a f c e a d

✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ * ✓

ans = ~~c~~ ~~d~~ ~~f~~ b ~~e~~ ~~f~~ a f c e d

obs1: When to delete last_char;

if last_char > new_char and

if last_char exists in future → : Hashmap

obs2: If a char is already present in ans → : HashSet

skip it

Operations:

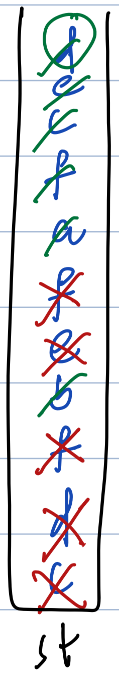
1. Access at last / Delete last / Insert at last in ans = stack

Dry Run:

↑

S : c d f b e f a f c e a d

*



get st → string ans

ans = d e c f a b

reverse ans

ans = b a f c e d

HS: Contains ans char

~~c~~ ~~d~~ ~~f~~ b ~~e~~ ~~f~~ a f c e

HM: Frequency of future

c : ~~2~~ ~~1~~ 0 a : ~~2~~ ~~1~~ 0

d : ~~2~~ 1 e : ~~2~~ ~~1~~ 0

f : ~~3~~ ~~2~~ ~~1~~ 0 b : ~~1~~ 0

```
string lexicographical(string s){   TC: O(N)   SC: O(N)
    unordered_map<char, int> hm;
    int N = s.size();
    for(int i=0; i<N; i++){
        hm[s[i]]++;
    }

    stack<char> st;
    unordered_set<char> hs;
    for(int i=0; i<N; i++){
        #New char s[i];
        if( hs.find(s[i]) != hs.end()){
        }   hm[s[i]]--;
        else{
            while( st.size()>0  &&  st.top() > s[i]  &&  hm[st.top()] >0) {
                hs.erase(st.top());
                st.pop();
            }

            st.push(s[i]);
            hs.insert(s[i]);
            hm[s[i]]--;
        }
    }

    string ans="";
    while(st.size()>0){
        ans += st.top();
    }   st.pop();
    reverse & return ans;
}
```

Total Iterations:

1. Frequency: N
2. Nested: Outer + Inner:
   N   N
3. Ans & reverse: N

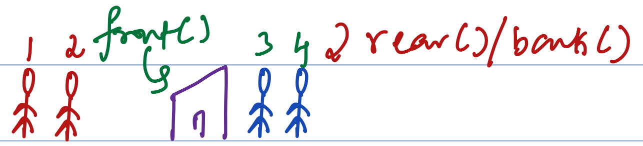
1 pop = 1 iteration

Total push = N iterations

Total pop = N

Total iteration = N

Queue:



Queue is datastructure, where you enter at back/rear & exit at front()

Property: FIFO : First In First Out

Functions:

Enque(x) : Insert x at rear/back end aka enque
deque() : delete ele at front end
front() : Return ele at front end
size() : Return no: of ele in queue.

Any que operation O(1)

Examples:

8	14	9	20	↑	30	front()	↑	front()	front()	↑	60
✓	✓	✓	✓	8	✓	14	14	9	9	9	

front rear()

~~8~~ ~~14~~ ~~9~~ 20 30 60

C++

queue<type> que;

que.push( ) Insert x at rear/back end aka enque
que.pop() delete ele at front end
que.front() Return ele at front end
que.size() Return no: of ele in queue.

28 Implement Queue using Stacks

Queue operations:

a. Enque()
b. Deque()
c. Front()
d. size()

} Every queue function should be implemented using stack function only

Enqueued TC: O(1)

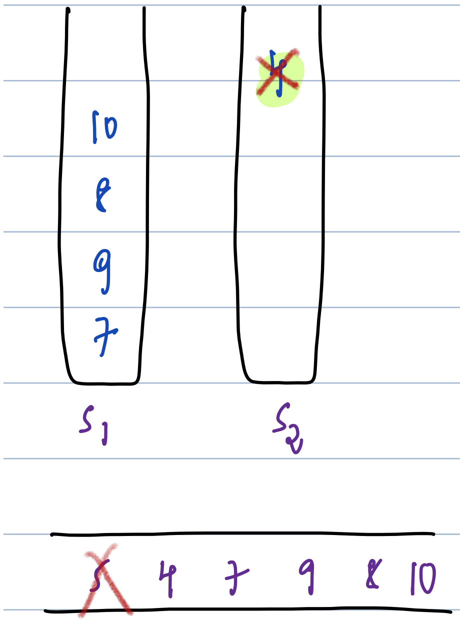
Stack:

a. push()
b. pop()
c. top()
d. size()

Ex: 5 4 7 9 deq() 8 fro() 10 deq() deq() 14 deq() deq() deq()

✓ ✓ ✓ ✓ 5 ✓ 4 ✓

Idea:



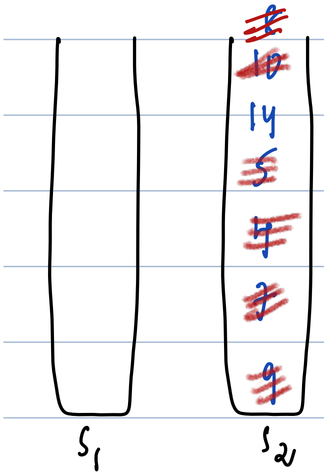
Pseudo Code:

```
stack<int> S1, S2;
void enque(int x){          TC: O(1)
    S1.push(x);
}
void deque( ){              TC: O(N)
    Transfer all ele S1 → S2
    S2.pop();
    Transfer all ele S2 → S1
}
int front( ){               TC: O(N)
    Transfer all ele S1 → S2
    int ans = S2.top();
    Transfer all ele S2 → S1
    return ans;
}
int size(){                 TC: O(1)
    return S1.size();
}
```

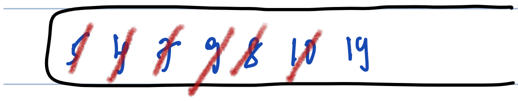
Ex: 5 ✓ 4 ✓ 7 ✓ 9 ✓ deq() 8 ✓ fro() 10 ✓ deq() ✓ deq() 14 ✓ deq() deq() deq()

5 4 4 7 9 8 10

Ideas:



Queue for Understanding:



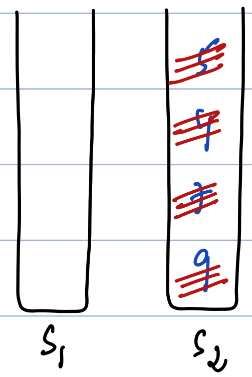
Pseudo Code:

```
stack<int> s1, s2;
void enque(int x){        TC: O(1)
    s1.push(x);
}
void deque( ){            TC: O(N)
    if( s2.size() == 0){
    }  Transfer all ele s1 -> s2
    s2.pop();
}
int front( ){             TC: O(N)
    if( s2.size() == 0){
    }  Transfer all ele s1 -> s2
    return s2.top();
}
int size(){               TC: O(1)
    return s1.size() + s2.size();
}
```

Estimating TC:

When worst case occurs, little rare, not regular, we estimate complexity based on avg: #Amortized Analysis.

Ex:	5	4	7	9	deq()	deq()	deq()	deq()
	✓	✓	✓	✓	5	4	7	9



		Iterations
1st deque:	Transfer S1 → S2 :	4
	S2.pop() :	1
2nd deque:	S2.pop() :	1
3rd deque:	S2.pop() :	1
4th deque:	S2.pop() :	1

Total 4 deque : 8 iterations

Avg 1 deque = 2 iteration ≈ O(1) Iterations

Remove All Duplicates in String:

You are given a string s and an integer k, if k continous characters are exactly same, delete them in string s.
Repeat above process in string return string which should not contain k continous similar characters

Ex1: s: a ~~b~~ ~~c~~ ~~c~~ ~~b~~ e a   # aea
k = 2

Ex2 s: ~~d~~ ~~e~~ ~~e~~ ~~e~~ ~~d~~ ~~b~~ ~~b~~ ~~c~~ ~~c~~ ~~c~~ ~~b~~ ~~d~~ a a   # aa
k = 3

Ans k=2

s = a b c d d e f f e c k

ans = a b ~~c~~ ~~d~~ ~~e~~ ~~f~~ k   # a b k

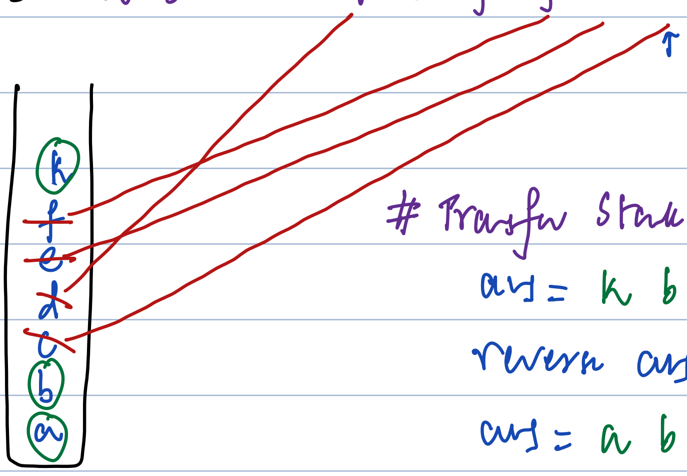
Operations:

a. Compare with last_char
b. Delete last_char
c. Insert at last

} # Can be done with stack

Dry Run with Stack.

s = a b c d d e f f e c k



Transfer Stack to string

ans = k b a

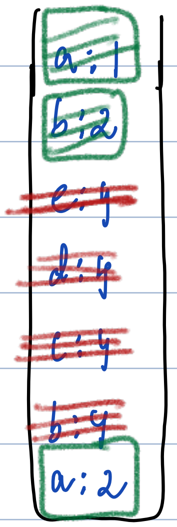
reverse ans

ans = a b k

Ex: a a ~~b~~ ~~c~~ ~~d d d d~~ ~~c c c~~ ~~b b~~ ~~e e e e~~ ~~b~~ b b a

k = 4

Idea:



ans = a b b a a   #reverse ans = a a b b a

1 2 2

stack<pair<char, int>> st;

Each ele in st is a pair

```
string krepeat(string s, int k){

    stack< pair<char, int>> st;
    for(int i=0; i<s.size(); i++){
        # s[i]
        if( st.size()==0 || st.top().first != s[i]){
            st.push({s[i],1});
        }
        else{
            st.top().second++;
        }

        if( st.top().second==k){
            st.pop();
        }
    }

    string ans;
    while(st.size()>0){
        char ch = st.top().first;
        int f = st.top().second;
        for(int i=1; i<=f; i++){
            ans += ch;
        }
        st.pop();
    }

    #reverse ans & return it.

}
```