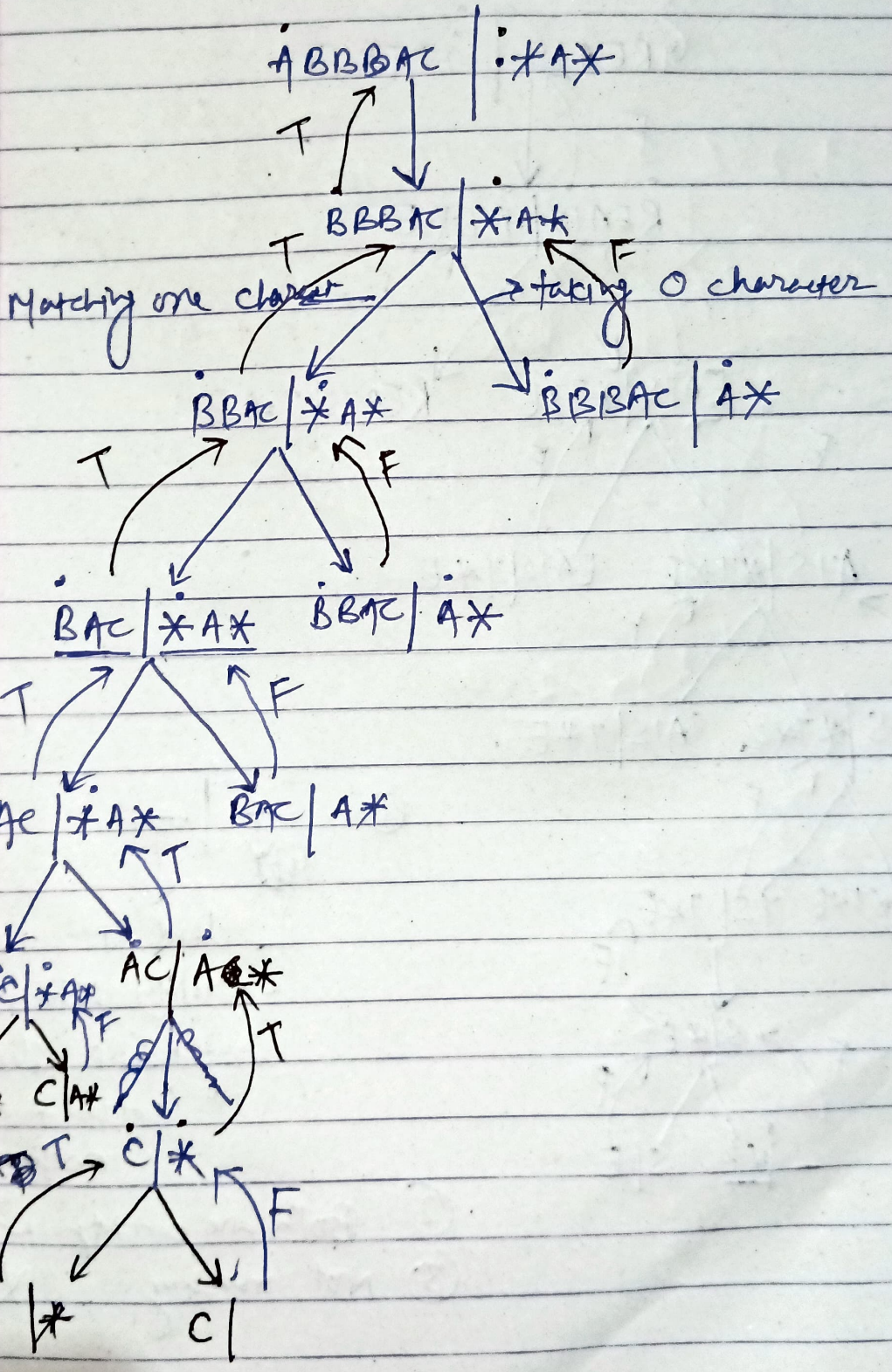
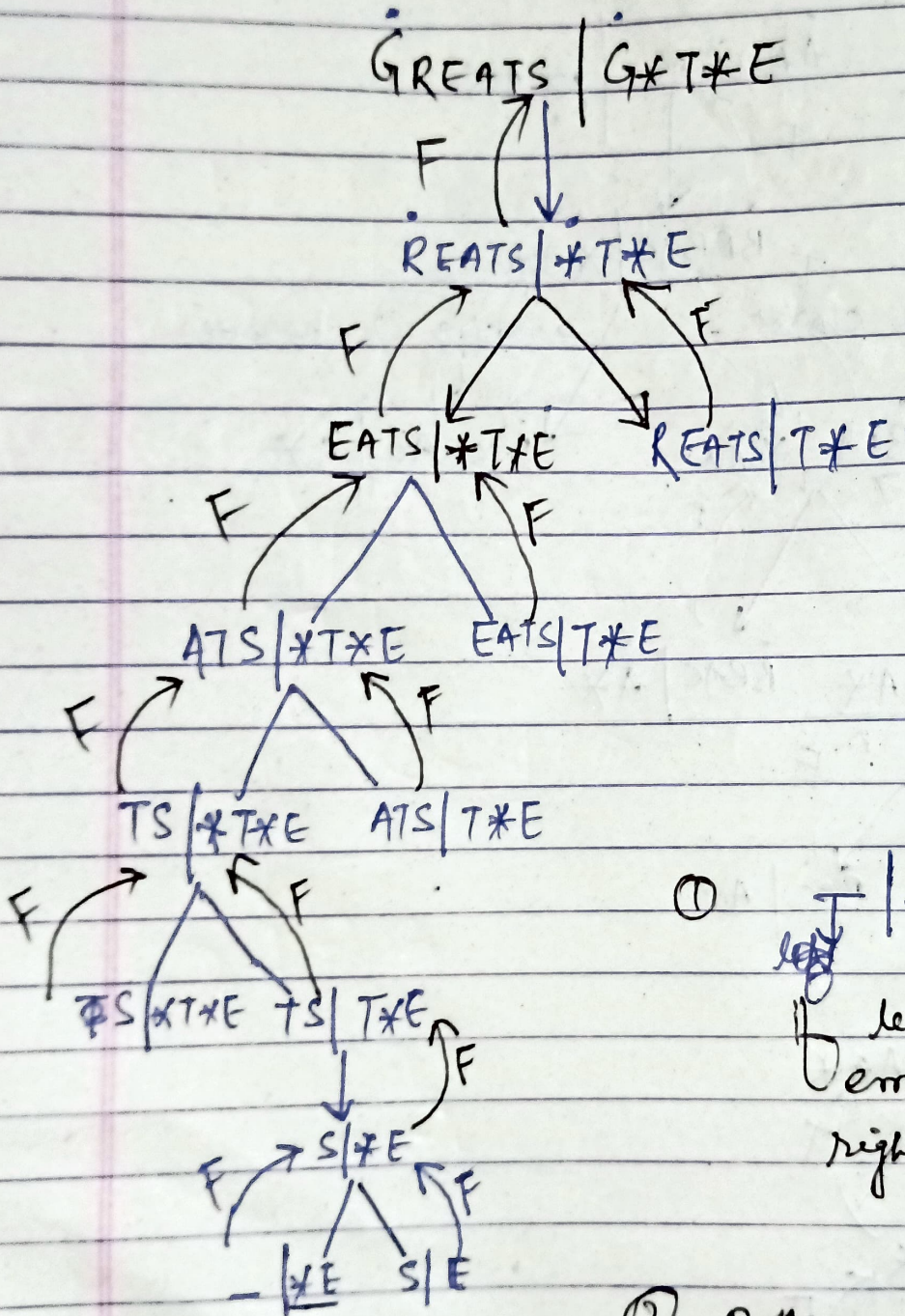
Tree diagram for Regular expression matching.

String = ABBBAC pattern : ~~*A*~~ .*A*



→ String = GREATS
pattern = G*T*E



① If left side is empty then right side should contain * or nothing

② Both are empty ✓

③ Not matching ✗

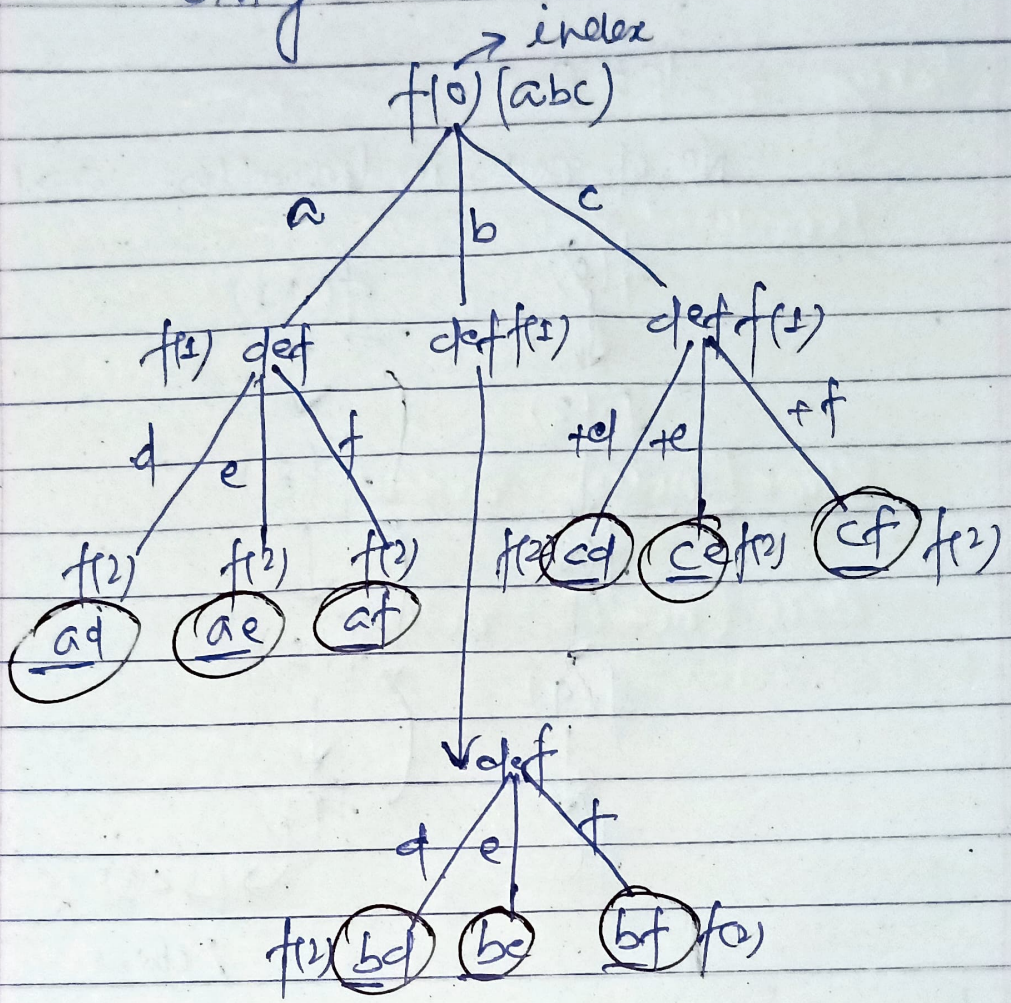
```cpp
#include <bits/stdc++.h>
using namespace std;

int recursiveWildCard(string input, string required, int n, int m,
vector<vector<int>> &dp)
{
    if (n < 0 and m < 0)
        return 1;
    if (m >= 0 and n < 0)
    {
        return 0;
    }
    if (n >= 0 and m < 0)
    {
        bool flag = true;
        for (int i = 0; i <= m; i++)
        {
            if (input[i] != '*')
            {
                flag = false;
                break;
            }
        }
        if (flag)
            return 1;
        else
            return 0;
    }
    if (dp[n][m] != -1)
        return dp[n][m];

    if (input[n] == required[m] or input[n] == '?')
    {
        dp[n][m] = recursiveWildCard(input, required, n - 1, m - 1, dp);
    }

    else if (input[n] == '*')
    {
        dp[n][m] = recursiveWildCard(input, required, n - 1, m, dp) ||
recursiveWildCard(input, required, n, m - 1, dp);
    }
    else
    {
        dp[n][m] = 0;
    }
    return dp[n][m];
}
int tabulationWildCard(string input, string required)
{
    int n = input.length();
    int m = required.length();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, 0));
    for (int i = 0; i <= m; i++)
    {
        dp[0][i] = 0;
    }
    for (int i = 0; i <= n; i++)
    {
        int flag = 1;
```

```cpp
        for (int j = 0; j <= i; j++)
        {
            if (input[j] != '*')
            {
                flag = 0;
                break;
            }
        }
        if (flag)
            dp[i][0] = 1;
        else
            dp[i][0] = 0;
    }
    dp[0][0] = 1;
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
        {
            if (input[i] == required[j] or input[i] == '?')
            {
                dp[i][j] = dp[i - 1][j - 1];
            }

            else if (input[i] == '*')
            {
                dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
            }
            else
            {
                dp[i][j] = 0;
            }
        }
    }
    return dp[n][m];
}
int main()
{
    string input;
    string required;
    cin >> input;
    cin >> required;
    int n = input.length();
    int m = required.length();
    vector<vector<int>> dp(n + 1, vector<int>(m + 1, -1));
    int ans = recursiveWildCard(input, required, n - 1, m - 1, dp);
    int ans1 = tabulationWildCard(input, required);
    if (ans1 and ans)
        cout << "YES";
    else
        cout << "NO";
}
```

2 → abc
3 → def

(Letter Combination of phone Number)

String = "123"




```
if( index == String.length() )
{
    res.push_back(output); return;
}
```

→ [OUTPUT :- ad, ae, af, bd, be, bf, cd, ce, cf]

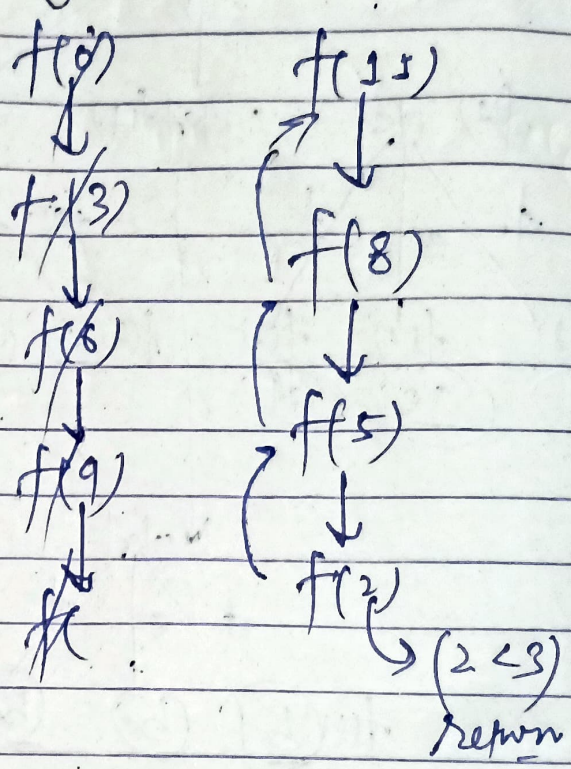
```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    vector<string> res;
    string getcode(char n)
    {

        string temp = "";
        if (n == '2')
        {
            temp = "abc";
        }
        else if (n == '3')
        {
            temp = "def";
        }
        else if (n == '4')
        {
            temp = "ghi";
        }
        else if (n == '5')
        {
            temp = "jkl";
        }
        else if (n == '6')
        {
            temp = "mno";
        }
        else if (n == '7')
        {
            temp = "pqrs";
        }
        else if (n == '8')
        {
            temp = "tuv";
        }
        else if (n == '9')
        {
            temp = "wxyz";
        }

        return temp;
    }

    void result(string digits, string output, int tochar)
    {
        if (tochar == digits.size())
        {
            res.push_back(output);
            output.pop_back();
            return;
        }

        string input = getcode(digits[tochar]);

        for (int i = 0; i < input.size(); i++)
```

```
60          {
61              output.push_back(input[i]);
62              result(digits, output, tochar + 1);
63              output.pop_back();
64          }
65          return;
66      }
67
68      vector<string> letterCombinations(string digits)
69      {
70
71          res.clear();
72          string output;
73          if (digits.size() == 0)
74          {
75              return {};
76          }
77
78          result(digits, output, 0);
79
80          return res;
81      }
82  };
83
84  int main()
85  {
86      Solution obj;
87      vector<string> res = obj.letterCombinations("23");
88      for (int i = 0; i < res.size(); i++)
89      {
90          cout << res[i] << endl;
91      }
92      return 0;
93  }
```

Reverse Node in K Group

k = 3

No of nodes in linked list = 11



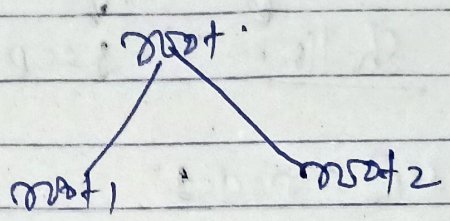
Rest in code

```cpp
#include<bits/stdc++.h>
using namespace std;

struct ListNode
{
    int val;
    ListNode *next;
    ListNode() : val(0), next(nullptr){}  ListNode(int x) : val(x), next(nullptr){}
 ListNode(int x, ListNode *next) : val(x), next(next){}

};

class Solution
{
public:
    ListNode *reverseKGroup(ListNode *head, int k)
    {

        if (head == nullptr)
        {
            return head;
        }
        int count = 1;

        ListNode *head1 = head;
        while (head1 and count <= k)
        {
            count++;
            head1 = head1->next;
        }

        if (count <= k)
        {
            return head;
        }

        ListNode *prev = NULL;
        ListNode *curr = head;
        ListNode *next = head;

        count = 1;
        while (curr and count <= k)
        {
            count++;
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }

        if (curr)
        {
            head->next = reverseKGroup(curr, k);
        }
        return prev;
    }
};

int main()
```

```
59 {
60     // make 5 test for the problem
61     ListNode *head = new ListNode(1);
62     head->next = new ListNode(2);
63     head->next->next = new ListNode(3);
64     head->next->next->next = new ListNode(4);
65     head->next->next->next->next = new ListNode(5);
66     Solution obj;
67     ListNode *head1 = obj.reverseKGroup(head, 2);
68     while (head1)
69     {
70         cout << head1->val << " ";
71         head1 = head1->next;
72     }
73 }
74
75
76
```

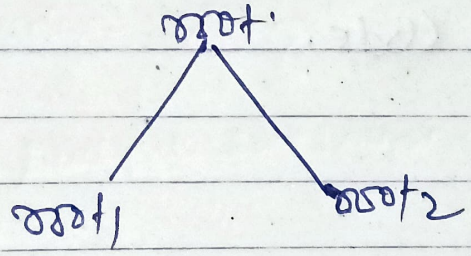
Copy With Random pointer

Shallow Copy :- Data change in every node.



if we change in root, root1 or root2 then change will reflect in all the root.

Deep copy :- Change not reflect



Here root1, root2 or root are independent of each other.

Approach

taking map with node, node

unordered_map <node*, node*> store

Shallow Deep Copy

store [node1] = node1'

store [node2] = node2'

.

.

.

Left in Code.

```cpp
#include<bits/stdc++.h>
using namespace std;

class Node
{
public:
    int val;
    Node *next;
    Node *random;

    Node(int _val)
    {
        val = _val;
        next = NULL;
        random = NULL;
    }
};
class Solution
{
public:
    Node *copyRandomList(Node *head)
    {
        map<Node *, Node *> m;
        int i = 0;
        Node *ptr = head;
        while (ptr)
        {
            m[ptr] = new Node(ptr->val);
            ptr = ptr->next;
        }
        ptr = head;
        while (ptr)
        {
            m[ptr]->next = m[ptr->next];
            m[ptr]->random = m[ptr->random];
            ptr = ptr->next;
        }
        return m[head];
    }
};

int main()
{
    // make 5 test for the problem
    Node *head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
    head->next->next->next->next = new Node(5);
    head->next->next->next->next->next = new Node(6);
    head->next->next->next->next->next->next = new Node(7);
    Solution obj;
    Node *head1 = obj.copyRandomList(head);




    return 0;
}
```

Time based key value

```cpp
#include<bits/stdc++.h>
using namespace std;
/*
 map<string,vector<pair<string,int>>>store;
    TimeMap() {

    }

    void set(string key, string value, int timestamp) {

        store[key].push_back({value,timestamp});
    }

    string get(string keys, int timestamp) {

        if(store.find(keys)==store.end())
        {
           return "";
        }

        int key =timestamp;

        int left=0;
        // int right=store.find(keys).second.size()-1;

        vector<pair<string,int>>&vct=store[keys];
        // cout<<store[keys][0].first<<endl;
        int right= vct.size()-1;

        while(left<=right)
        {
            int mid=(left+right)>>1;

            if(vct[mid].second==key)
            {
              return vct[mid].first;
            }

            if(vct[mid].second>key)
            {
                right =mid-1;
            }
            else
            {
                left= mid+1;
            }
        }
        // cout<<right<<" "<<left<<endl;

        return left!=0?vct[left-1].first:"";

    }
*/

unordered_map<string, vector<pair<int, string>>> m;
void set1(string key, string value, int timestamp)
{
    m[key].push_back({timestamp, value});
}
```

first approach

Second Approach

```cpp
string get(string key, int timestamp)
{
    auto it = upper_bound(begin(m[key]), end(m[key]), pair<int, string>(timestamp, ""), [](const pair<int, string> &a, const pair<int, string> &b)
                          { return a.first < b.first; });
    return it == m[key].begin() ? "" : prev(it)->second;
}


int main()
{
//   create 5 test case
    set1("foo", "bar", 1);
    set1("foo", "bar", 3);
    set1("foo", "baz", 5);
    set1("foo", "bar", 7);
    set1("foo", "bar", 9);
    get("foo", 5);
    get("foo", 10);
    get("foo", 11);
    get("foo", 12);
    return 0;
}
```

```cpp

#include<bits/stdc++.h>
using namespace std;

struct TreeNode
{
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL){}

};
class Codec
{
public:
    // Encodes a tree to a single string.
    string serialize(TreeNode *root)
    {
        string res="";

        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty())
        {
            TreeNode* curr=q.front();
            q.pop();
            if(curr==NULL)
            {
                res+="null,";
            }
            else
            {
                res+=to_string(curr->val)+",";
                q.push(curr->left);
                q.push(curr->right);
            }
        }
        return res;
    }

    // Decodes your encoded data to tree.
    TreeNode *deserialize(string data)
    {
        stringstream ss(data);
        string curr;
        getline(ss,curr,',');
        if(curr=="null")
        {
            return NULL;
        }
        TreeNode* root=new TreeNode(stoi(curr));
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty())
        {
            int n= q.size();
            for(int i=0;i<n;i++)
            {
                TreeNode* node=q.front();
```

Serialized and Deserialized

```
                q.pop();
                getline(ss,curr,',');
                if(curr=="null")
                {
                    node->left=NULL;
                }
                else
                {
                    TreeNode* left=new TreeNode(stoi(curr));
                    node->left=left;
                    q.push(left);
                }
                getline(ss,curr,',');
                if(curr=="null")
                {
                     node->right=NULL;
                }
                else
                {
                    TreeNode *right = new TreeNode(stoi(curr));
                    node->right = right;
                    q.push(right);
                }
            }
        }
        return root;
    }
};


int main()
{
   Codec obj;
    TreeNode *root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);
    string s = obj.serialize(root);
    TreeNode *root1 = obj.deserialize(s);
//    print root1




    return 0;
}


```

```cpp
/*

“Meeting Rooms II” states that you are given an array of meeting time intervals “intervals” where “intervals[i] = [ start[i], end[i] ]”, return the minimum number of conference rooms required.

*/

#include <bits/stdc++.h>
using namespace std;
int minMeetingRooms(vector<vector<int>> &intervals)
{
    vector<int> start;
    vector<int> end;
    for (int i = 0; i < intervals.size(); i++)
    {
        start.push_back(intervals[i][0]);
        end.push_back(intervals[i][1]);
    }
    sort(start.begin(), start.end());
    sort(end.begin(), end.end());
    int i = 1, j = 0, curr = 1;
    int answer = 1;
    while (i < start.size() && j < end.size())
    {
        if (start[i] < end[j])
        {
            curr++;
            i++;
        }
        else
        {
            curr--;
            j++;
        }
        answer = max(answer, curr);
    }
    return answer;
}
int main()
{
    vector<vector<int>> intervals = {{0, 30}, {5, 10}, {15, 20}};
    cout << minMeetingRooms(intervals);
    return 0;
}
```

Meeting Room II

```cpp
/*
Implement the RandomizedSet class:

RandomizedSet() Initializes the RandomizedSet object.
bool insert(int val) Inserts an item val into the set if not present. Returns true if the item was not present, false otherwise.
bool remove(int val) Removes an item val from the set if present. Returns true if the item was present, false otherwise.
int getRandom() Returns a random element from the current set of elements (it's guaranteed that at least one element exists when this method is called). Each element must have the same probability of being returned.
You must implement the functions of the class such that each function works in average O(1) time complexity.
*/
#include<bits/stdc++.h>
using namespace std;

class RandomizedSet
{
public:
    unordered_map<int, int> store;
    vector<int> value;
    RandomizedSet()
    {
    }

    bool insert(int val)
    {
        if (store.find(val) != store.end())
        {
            return false;
        }
        value.push_back(val);
        store[val] = value.size() - 1;

        return true;
    }

    bool remove(int val)
    {
        if (store.find(val) == store.end())
        {
            return false;
        }

        int index = store[val];
        int lastelement = value.back();
        store[lastelement] = index;
        value[index] = lastelement;
        value.pop_back();
        store.erase(val);

        return true;
    }

    int getRandom()
    {

        return value[rand() % (value.size())];
```

Insert Delete Get Random

```
    }
};

int main()
{
 RandomizedSet obj;
    obj.insert(1);
    obj.insert(2);
    obj.insert(3);
    obj.insert(4);
    obj.insert(5);
    obj.remove(3);
    obj.remove(4);
    obj.insert(6);
    cout << obj.getRandom();
    return 0;

}
```

Find Median From Data Stream

```cpp
#include<bits/stdc++.h>
using namespace std;

class MedianFinder
{
public:
    //     max priority queue
    priority_queue<int> maxpq;
    priority_queue<int, vector<int>, greater<int>> minpq;
    MedianFinder()
    {
    }

    void addNum(int num)
    {
        if (maxpq.size() == 0 or num < maxpq.top())
        {
            maxpq.push(num);
        }
        else
        {
            minpq.push(num);
        }
        //         balancing part
        if (maxpq.size() > minpq.size() + 1)
        {
            minpq.push(maxpq.top());
            maxpq.pop();
        }
        else if (minpq.size() > maxpq.size() + 1)
        {
            maxpq.push(minpq.top());
            minpq.pop();
        }
    }

    double findMedian()
    {

        if (minpq.size() == maxpq.size())
        {
            if (minpq.size() == 0)
            {
                return 0;
            }
            return ((minpq.top() + maxpq.top()) * 1.0) / 2;
        }
        else
        {
            return (minpq.size() > maxpq.size() ? minpq.top() : maxpq.top());
        }
    }
};


int main()
{
    MedianFinder obj;
    obj.addNum(1);
```

```
60      obj.addNum(2);
61      obj.addNum(3);
62      cout << obj.findMedian() << endl;
63      obj.addNum(4);
64      cout << obj.findMedian() << endl;
65      return 0;
66  }
```