

Jul 9, 2022

Question Name From Leetcode

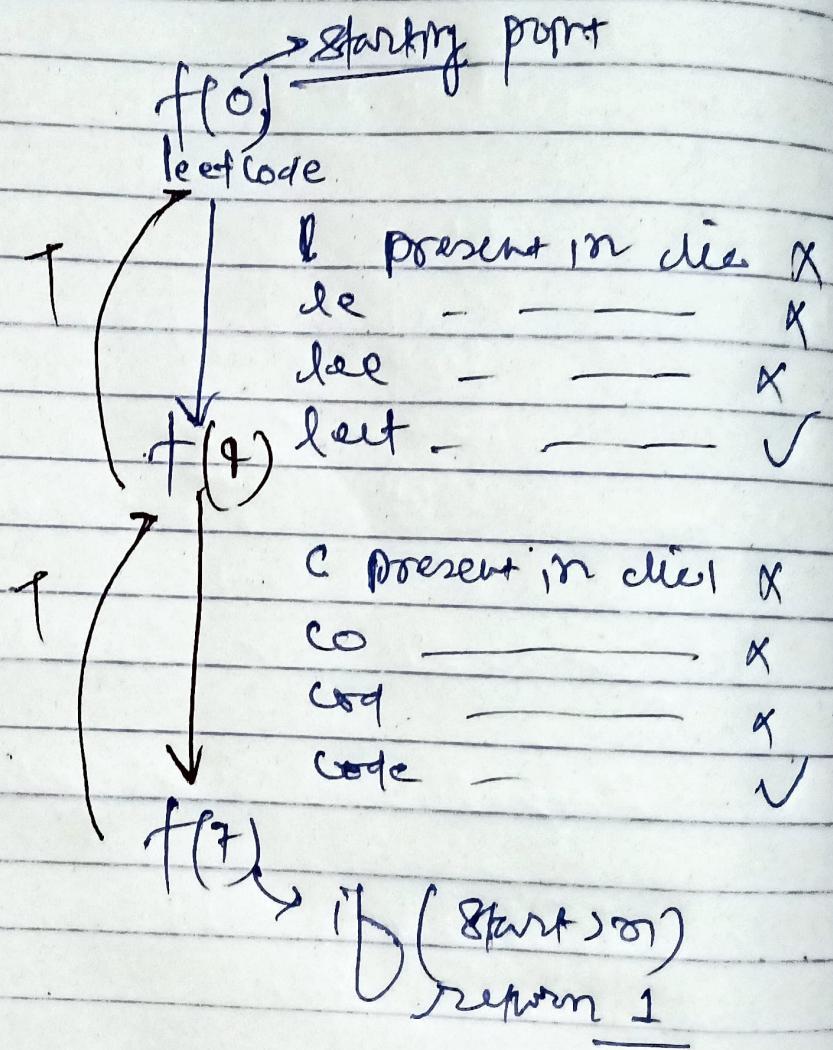
1. Trapping Rain water
2. LRU Cache
3. Longest Palindromic substring
4. Median of two sorted array
5. Critical connection in network
6. Integer to english word
7. Partition Labels
8. Minimum Remove to make valid parentheses
9. Container with most water
10. Alien Dictionary
11. Decode Ways
12. Decode String
13. Minimum window substring
14. Text justification
15. Maximal Rectangle
16. Word Break
17. First missing Positive number

Worst Breaking

→ leet code → dict

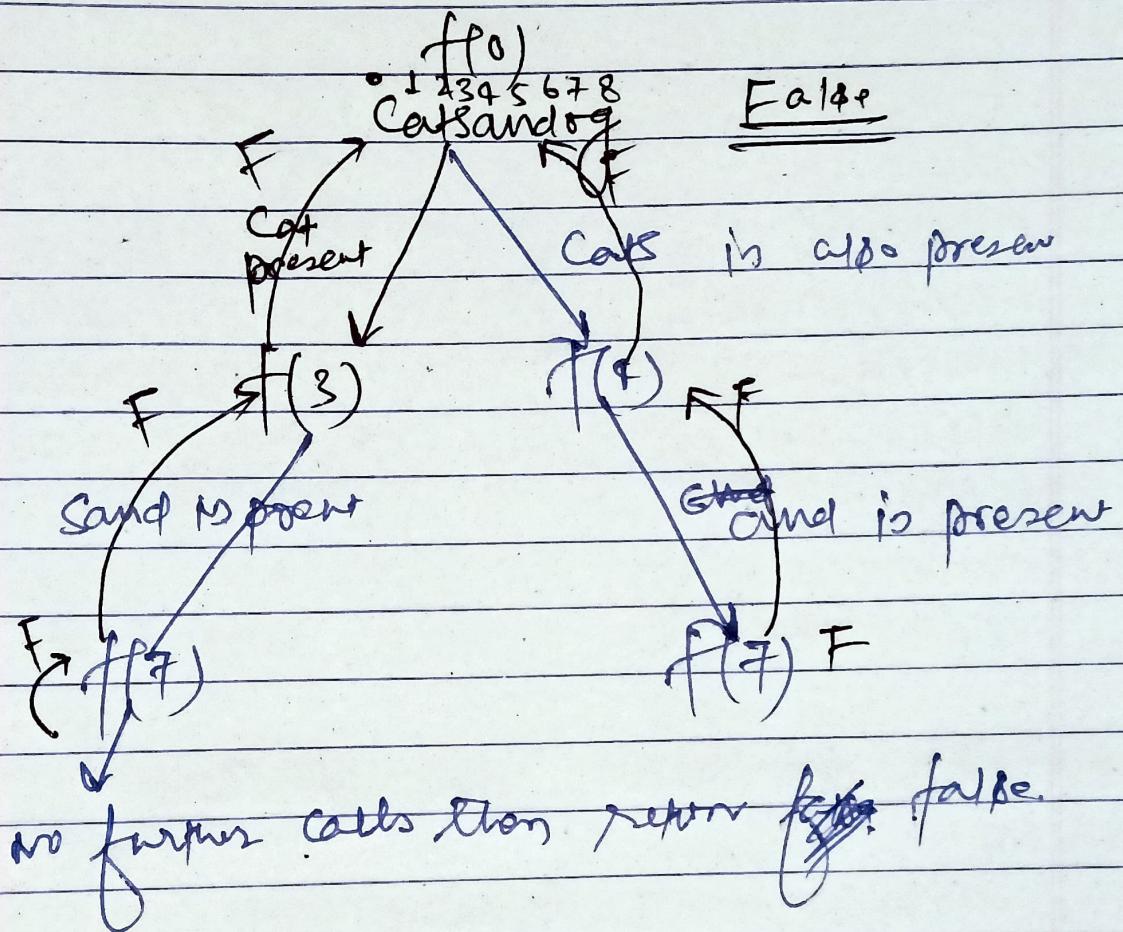
0 1 2 3 4 5 6

→ Worst .leetcode



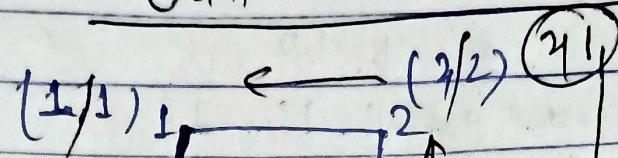
dict = {Cat, dog, Sand, and Cat}

Word 2 Catsanddog



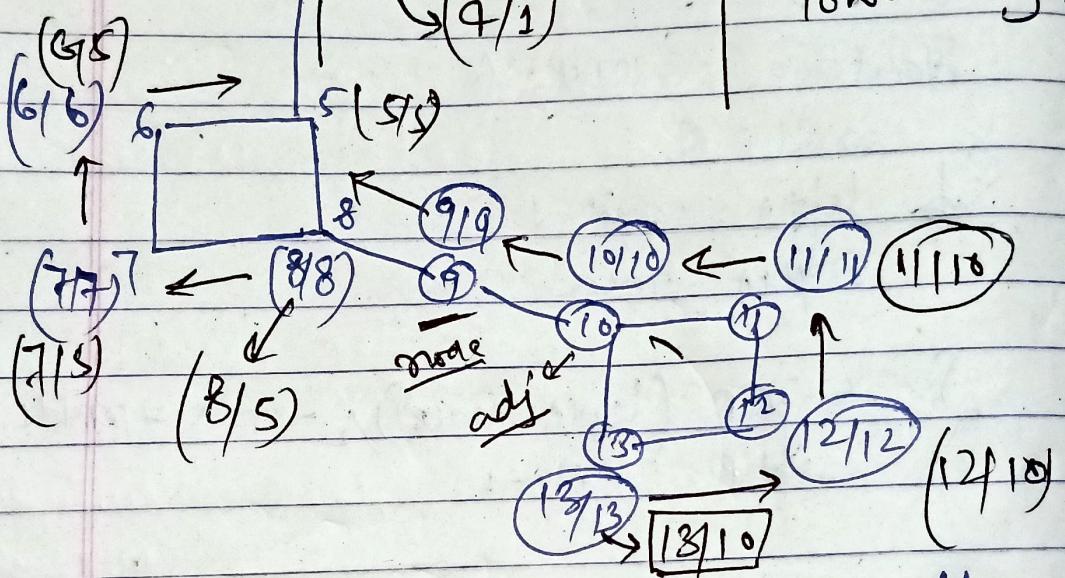
```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 bool helper(const string &s, int start, const unordered_set<string> &d, vector<int>
&memo)
5 {
6     if (start == s.size())
7     {
8         return true;
9     }
10
11    if (memo[start] != -1)
12        return memo[start];
13
14    for (int i = start; i <= s.size(); ++i)
15    {
16        const string sub = s.substr(start, i - start + 1);
17        // cout<<sub<<endl;
18        if (d.count(sub) == 1)
19        {
20            if (helper(s, i + 1, d, memo))
21            {
22                memo[start] = 1;
23                return true;
24            }
25        }
26    }
27
28    memo[start] = 0;
29    return false;
30 }
31
32 bool wordBreak(string s, vector<string> &wordDict)
33 {
34     unordered_set<string> d(wordDict.begin(), wordDict.end());
35
36     vector<int> memo(s.length(), -1);
37     return helper(s, 0, d, memo);
38 }
39
40 int main()
41 {
42     string s = "leetcode";
43     vector<string> wordDict = {"leet", "code"};
44     cout << wordBreak(s, wordDict) << endl;
45     return 0;
46 }
```

Critical Connection



Two array

everytime
low



low \rightarrow JI6 actually. By ignoring the external path, what is the lowest node can reach.

\rightarrow If we found any direct edge low
that edge is visited than
 $low[\text{node}] = \min(\text{low}[\text{node}], \text{everytime})$

\rightarrow Otherwise call it next node
and similarly update the low value

$low[\text{node}] = \min(\text{low}[\text{node}], \text{everytime}, \text{low}[\text{adj}])$

Condition for Critical Connection

if everytime of adj node $<$ low of adj
result push back ({node, adj}).

- low of adj ($\text{low}[\text{adj}]$) = lowest can be reached by ignoring the actual path
- Entertime[node] = 9'th current entertime.

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 vector<vector<int>> result;
5
6 void dfs(int node, int parent, vector<bool> &vis, vector<int> &entrytime, vector<int>
7 &low, vector<int> graph[], int &timer)
8 {
9     entrytime[node] = low[node] = timer++;
10    vis[node] = true;
11
12    for (auto temp : graph[node])
13    {
14        int adj = temp;
15        if (adj == parent)
16        {
17            continue;
18        }
19
20        else if (!vis[adj])
21        {
22            dfs(adj, node, vis, entrytime, low, graph, timer);
23            low[node] = min(low[node], low[adj]);
24            if (low[adj] > entrytime[node])
25            {
26                result.push_back({node, adj});
27            }
28        }
29        else if (vis[adj])
30        {
31            low[node] = min(low[node], entrytime[adj]);
32        }
33    }
34 }
35
36 vector<vector<int>> criticalConnections(int n, vector<vector<int>> &connections)
37 {
38
39     vector<int> graph[n];
40     int m = connections.size();
41     for (int i = 0; i < m; i++)
42     {
43         graph[connections[i][0]].push_back(connections[i][1]);
44         graph[connections[i][1]].push_back(connections[i][0]);
45     }
46
47     vector<bool> vis(n, false);
48     vector<int> entrytime(n, -1);
49     vector<int> low(n, -1);
50     result.clear();
51     int timer = 0;
52     for (int i = 0; i < n; i++)
53     {
54         if (!vis[i])
55         {
56             dfs(i, -1, vis, entrytime, low, graph, timer);
57         }
58     }
59 }
```

```
59     return result;
60 }
61 }
62
63 int main()
64 {
65     int n = 4;
66     vector<vector<int>> connections = {{0, 1}, {1, 2}, {2, 0}, {1, 3}};
67     vector<vector<int>> result = criticalConnections(n, connections);
68     for (auto temp : result)
69     {
70         cout << temp[0] << " " << temp[1] << endl;
71     }
72     return 0;
73 }
```

Decode String

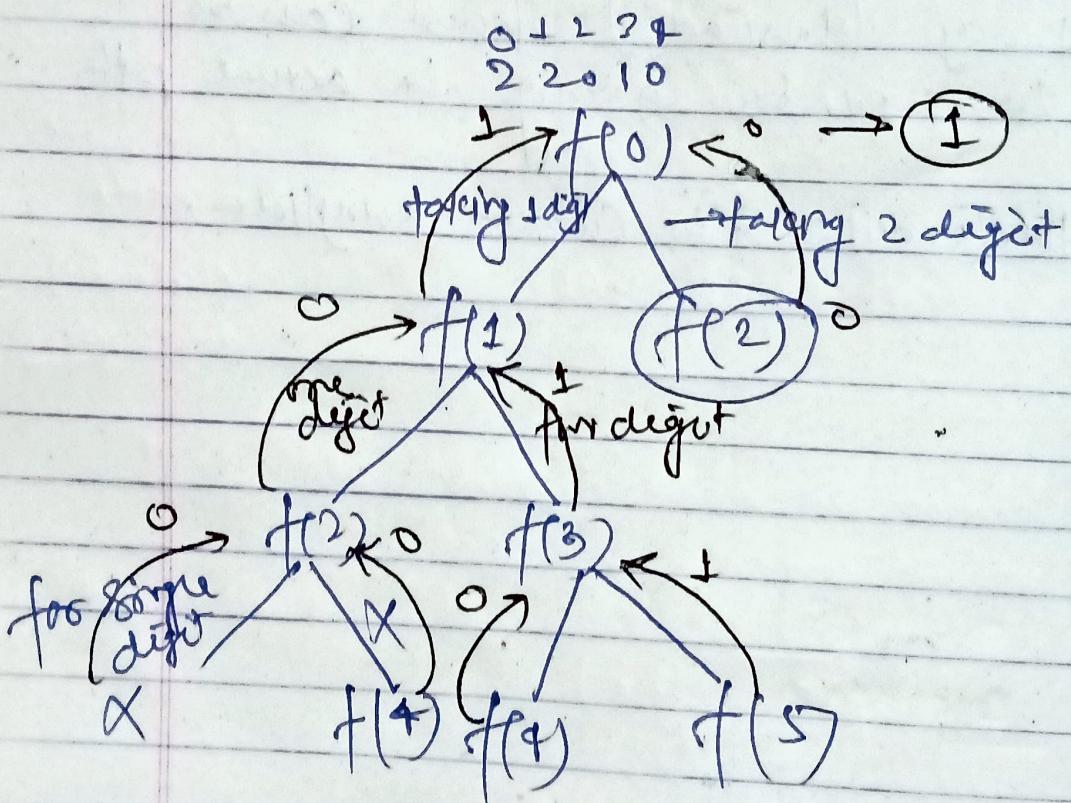
Page No.:

Date: / /

→ In this question we need to take care of two things

① Taking only one digit

② Taking two digit a from



```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 class Solution
5 {
6 public:
7     int tabulation(string &input)
8     {
9         int n = input.size();
10        vector<int> dp(n + 1, 0);
11        dp[0] = 1;
12
13        for (int start = 1; start <= n; start++)
14        {
15            int takingOne = 0;
16            int takingtwo = 0;
17
18            if (input[start - 1] != '0')
19            {
20                takingOne = dp[start - 1];
21            }
22            if (start - 2 >= 0)
23            {
24                int number = (input[start - 2] - '0') * 10 + (input[start - 1] -
25 '0');
26                if (number >= 10 and number <= 26)
27                {
28                    takingtwo = dp[start - 2];
29                }
30            }
31            dp[start] = (takingOne + takingtwo);
32        }
33
34        return dp[n];
35    }
36
37    int result(string &input, int start)
38    {
39        if (start < 0)
40        {
41            return 1;
42        }
43
44        int takingOne = 0;
45        int takingtwo = 0;
46
47        if (input[start] != '0')
48        {
49            takingOne = result(input, start - 1);
50        }
51        if (start - 1 >= 0)
52        {
53            int number = (input[start - 1] - '0') * 10 + (input[start] - '0');
54            if (number >= 10 and number <= 26)
55            {
56                takingtwo = result(input, start - 2);
57            }
58        }
59        return (takingOne + takingtwo);
60    }
61}
```

```
59     }
60     int numDecodings(string s)
61     {
62         // return result(s,s.size()-1);
63         return tabulation(s);
64     }
65 };
66
67
68
69 int main()
70 {
71     Solution s;
72     string input = "12";
73     cout << s.numDecodings(input) << endl;
74     return 0;
75 }
```

Median of two Sorted array

$$\text{arr}_1 = 2, 5, 7, 8, 9, 10$$

$$\text{arr}_2 = 1, 3, 6, 9, 11, 12, 13$$

→ if length of both array is odd then
median will be $\text{max}(\text{arr}_1[\text{left}], \text{arr}_2[\text{left}])$

→ If array length is even then median will be

$$\frac{\text{max}(\text{arr}_1[\text{left}], \text{arr}_1[\text{left}+1]) + \text{min}(\text{arr}_2[\text{right}], \text{arr}_2[\text{right}-1])}{2}$$

→ In normal array

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 length is odd

median

if Length is even

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

$$(6+7)/2 = \underline{6.5} \text{ (Median)}$$

	0	1	2	3	4	5
A =	2	5	7	8	9	10
B =	1	3	4	6	11	12

→ By fixing array A, we will move array B.

$$\text{start} = 0 \quad \text{end} = 6$$

$$\text{mid} = 3$$

$$\left\{ \begin{array}{l} \text{left } A = \text{mid} - 1 \\ \text{right } A = \text{mid} \end{array} \right.$$

$$\left\{ \begin{array}{l} \text{right } B = (\text{len}(A) + \text{len}(B)) / 2 - \text{mid} = \text{mid} + 1 \\ \text{left } B = \text{mid} + 1 - 1 \end{array} \right.$$

For iteration we need to take care of 3 cases.

① if ($\text{left } B > \text{right } A$)
 $\left\{ \begin{array}{l} \text{left } B = \text{mid} + 1; \\ \text{right } A = \text{mid} - 1; \end{array} \right.$

else if ($\text{left } A > \text{right } B$)
 $\left\{ \begin{array}{l} \text{left } A = \text{mid} + 1; \\ \text{right } B = \text{mid} - 1; \end{array} \right.$

else

$$\text{int total} = (\text{start} + \text{end});$$

if ($\text{total} \% 2 == 0$)

return $\max(\text{left } A, \text{left } B)$;

else

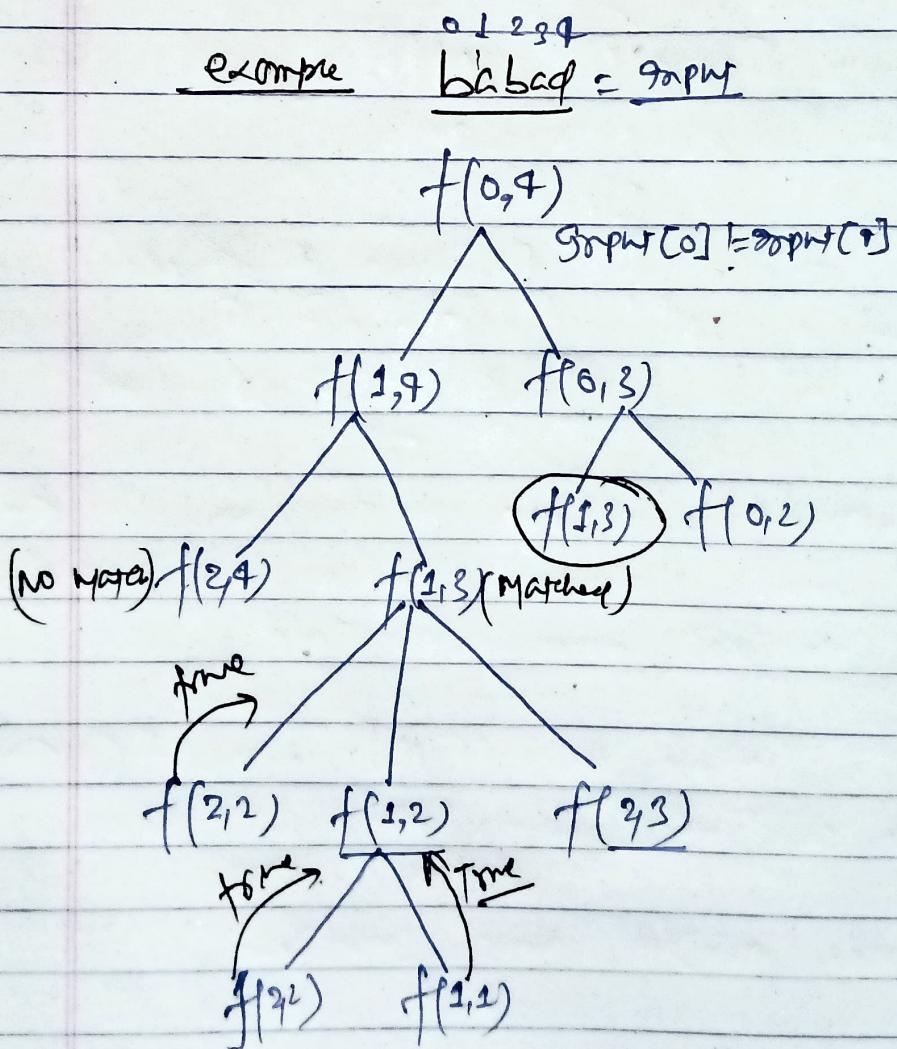
return $(\max(\text{left } A, \text{left } B) + \min(\text{right } A, \text{right } B)) / 2$

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 double findMedianSortedArrays(vector<int> &A, vector<int> &B)
5 {
6
7     int n = A.size();
8     int m = B.size();
9
10    if (n > m)
11        return findMedianSortedArrays(B, A);
12
13    int totalLeft = (n + m + 1) / 2;
14
15    int s = 0, e = n;
16
17    while (s <= e)
18    {
19        int mid = (s + e) / 2;
20        int countA = mid;
21        int countB = totalLeft - countA;
22
23        double leftA = (countA == 0) ? INT_MIN : A[countA - 1];
24        double rightA = (countA == n) ? INT_MAX : A[countA];
25
26        double leftB = (countB == 0) ? INT_MIN : B[countB - 1];
27        double rightB = (countB == m) ? INT_MAX : B[countB];
28
29        if (leftB > rightA)
30            s = mid + 1;
31        else if (leftA > rightB)
32            e = mid - 1;
33        else
34        {
35            int total = (n + m);
36            if (total % 2 != 0)
37                return double(max(leftA, leftB));
38            else
39                return (double(max(leftA, leftB)) + double(min(rightA, rightB))) / 2;
40        }
41    }
42    return 0;
43 }
44
45 int main()
46 {
47     vector<int> A={2,5,7,8,9,10};
48     vector<int> B={1,3,4,6,11,12,13};
49     double res=findMedianSortedArrays(A,B);
50     cout<<res<<endl;
51 }
```

Page No. _____

Longest Palindromic Substring

- If Corrⁿ matched then go for three calls
- If Corrⁿ not matched then go 2 cells and at every point Cal the palindrome length if that segment is palindrome



```
1 #include<bits/stdc++.h>
2 using namespace std;
3 string tabulation(string s)
4 {
5     int n= s.size();
6
7     vector<vector<int>>dp(n, vector<int>(n, 0));
8     int start = 0;
9     int end=0;
10    int maxlen=0;
11    for(int g=0;g<n;g++)
12    {
13        for(int i=0,j=g;j<n;j++,i++)
14        {
15            if(g==0)
16            {
17                dp[i][j]=true;
18            }
19            else if(g==1)
20            {
21                if(s[i]==s[j])
22                {
23                    dp[i][j]=true;
24                }
25                else
26                {
27                    dp[i][j]=false;
28                }
29            }
30            else
31            {
32                if(s[i]==s[j])
33                {
34                    dp[i][j]=dp[i+1][j-1];
35                }
36                else
37                {
38                    dp[i][j]=0;
39                }
40            }
41            if(dp[i][j])
42            {
43                int length= (j-i+1);
44                if(length>maxlen)
45                {
46                    maxlen= length;
47                    start= i;
48                    end=j;
49                }
50            }
51        }
52    }
53 }
54 // cout<<start<<" "<<end<<" "<<maxlen<<endl;
55 // for(int i=0;i<n;i++)
56 // {
57 //     for(int j=0;j<n;j++)
58 //     {
59 //         cout<<dp[i][j]<<" ";
```

```

60     // }
61     // cout<<endl;
62     // }
63     string result="";
64     for(int i=start;i<=end;i++)
65     {
66         result+=s[i];
67     }
68     result= s.substr(start,end+1);
69     return result;
70 }
71
72 bool helper(int l, int r, string &s, int memo[][1001], vector<int> &answer)
73 {
74     // Check if [ L ... R ] is already calculated before
75     if (memo[l][r] != -1)
76     {
77         return memo[l][r];
78     }
79
80     // Base case, every single letter or empty string is also a palindrome
81     if (l >= r)
82     {
83         return true;
84     }
85
86     // Possible palindrome beginning
87     bool found = false;
88     if (s[l] == s[r])
89     {
90         found = helper(l + 1, r - 1, s, memo, answer);
91     }
92
93     // Try other possibilities
94     helper(l + 1, r, s, memo, answer);
95     helper(l, r - 1, s, memo, answer);
96
97     // Update longest palindrome position and width
98     if (found && (r - l + 1) > answer[2])
99     {
100         answer = {l, r, (r - l + 1)};
101     }
102
103     // Memoization
104     memo[l][r] = found;
105
106     return found;
107 }
108
109 string longestPalindrome(string s)
110 {
111     int memo[1001][1001];
112     memset(memo, -1, sizeof(memo));
113     vector<int> answer = {0, 0, 1};
114     helper(0, s.size() - 1, s, memo, answer);
115     // cout<<answer[0] << " " << answer[1] << " " << answer[2] << endl;
116     return s.substr(answer[0], answer[2]);
117 }
118
119

```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int maxArea(vector<int> &height)
5 {
6
7     int ans = 0;
8     int left = 0;
9     int right = height.size() - 1;
10
11    while (left < right)
12    {
13        int leftelement = height[left];
14
15        int rightelement = height[right];
16
17        int mine = min(rightelement, leftelement);
18
19        ans = max(ans, (right - left) * mine);
20
21        if (leftelement < rightelement)
22        {
23            left++;
24        }
25        else
26        {
27            right--;
28        }
29    }
30
31    return ans;
32 }
33
34 int main()
35 {
36     vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
37     cout << maxArea(height) << endl;
38     return 0;
39 }
40 }
```

```
120 int main()
121 {
122     string s = "babad";
123     cout << longestPalindrome(s) << endl;
124     // cout<<"raushan kumar"<<endl;
125     return 0;
126 }
```

```

1 // 1st approach by taking 2 vector
2 // 2nd approach by taking two pointer and two variable to track the max height and
3 // the max index
4
5 #include<bits/stdc++.h>
6 using namespace std;
7
8 // 1st approach
9 /*
10    1st vector is maxfromleft side and second vector will be maxfromrightside
11    and result will be
12    rest+= min(maxfromleft, maxfromright)-height[i]
13 */
14 int trap1(int A[],int n)
15 {
16     int result=0;
17     vector<int> maxfromleft(n,0);
18     vector<int> maxfromright(n,0);
19     maxfromleft[0]=A[0];
20     maxfromright[n-1]=A[n-1];
21     for(int i=1;i<n;i++)
22     {
23         maxfromleft[i]=max(maxfromleft[i-1],A[i]);
24     }
25     for(int i=n-2;i>=0;i--)
26     {
27         maxfromright[i]=max(maxfromright[i+1],A[i]);
28     }
29     for(int i=0;i<n;i++)
30     {
31         result+=min(maxfromleft[i],maxfromright[i])-A[i];
32     }
33     return result;
34 }
35
36 // 2nd approach
37 int trap(int A[], int n)
38 {
39     int left = 0;
40     int right = n - 1;
41     int res = 0;
42     int maxleft = 0, maxright = 0;
43     while (left <= right)
44     {
45         if (A[left] <= A[right])
46         {
47             if (A[left] >= maxleft)
48                 maxleft = A[left];
49             else
50                 res += maxleft - A[left];
51             left++;
52         }
53         else
54         {
55             if (A[right] >= maxright)
56                 maxright = A[right];
57             else
58                 res += maxright - A[right];
59     }
60     return res;
61 }
```

```
59         right--;
60     }
61 }
62 return res;
63 }
64
65 int main()
66 {
67     int A[] = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
68     int n = sizeof(A) / sizeof(A[0]);
69     cout << trap(A, n);
70     return 0;
71 }
72 /*
73 time complexity: O(n)
74 */
75
76
```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 vector<char> calculate(string input[], int n)
5 {
6     vector<int> indegree(26, -1);
7     for(int i=0; i<n; i++)
8     {
9         for(int j=0; j<input[i].size(); j++)
10        {
11            indegree[input[i][j] - 'a'] = 0;
12        }
13    }
14
15 // making the graph
16 unordered_map<char, vector<char>> graph;
17
18 for(int i=0; i<n-1; i++)
19 {
20     string first = input[i];
21     string second = input[i+1];
22     for(int j=0; j<min(first.size(), second.size()); j++)
23     {
24         if(first[j] != second[j])
25         {
26             graph[first[j]].push_back(second[j]);
27             break;
28         }
29     }
30 }
31 vector<char> result;
32 // now calculate the indegree of the graph
33 for(auto i=graph.begin(); i!=graph.end(); i++)
34 {
35     for(int j=0; j<i->second.size(); j++)
36     {
37         indegree[i->second[j] - 'a']++;
38     }
39 }
40 // print the indegree of the graph
41 queue<char> q;
42 for(int i=0; i<26; i++)
43 {
44     if(indegree[i] == 0)
45     {
46         result.push_back(i + 'a');
47         q.push(i + 'a');
48     }
49 }
50
51 while(!q.empty())
52 {
53     char curr = q.front();
54     q.pop();
55     for(int i=0; i<graph[curr].size(); i++)
56     {
57         indegree[graph[curr][i] - 'a']--;
58         if(indegree[graph[curr][i] - 'a'] == 0)
59         {
```

```
60         result.push_back(graph[curr][i]);
61         q.push(graph[curr][i]);
62     }
63 }
64 return result;
65
66
67 }
68
69
70
71
72 int main()
73 {
74     string words[] = { "baa",
75                         "abcd",
76                         "abca",
77                         "cab",
78                         "cad" };
79
80     int n = 5;
81     vector<char>order=calculate(words, n);
82     for(int i=0;i<order.size();i++)
83     {
84         cout<<order[i]<<" ";
85     }
86
87     return 0;
88 }
```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4     string minWindow(string s, string t)
5     {
6
7         unordered_map<char, int> store;
8         int nt = t.size();
9         for (int i = 0; i < nt; i++)
10        {
11            store[t[i]]++;
12        }
13
14        int count = store.size();
15
16        int ns = s.size();
17        int start = -1;
18        int end = -1;
19        int minlen = INT_MAX;
20        int left = 0;
21        for (int right = 0; right < ns; right++)
22        {
23            if (store.find(s[right]) == store.end())
24            {
25                continue;
26            }
27            store[s[right]]--;
28            if (store[s[right]] == 0)
29            {
30                count--;
31            }
32
33            while (count == 0 and left <= right)
34            {
35                if (count == 0)
36                {
37                    // cout<<count<<" "<<left<<" "<<right<<endl;
38                    int lent = (right - left + 1);
39                    if (minlen > lent)
40                    {
41                        minlen = lent;
42                        start = left;
43                        end = right;
44                    }
45                }
46                if (store.find(s[left]) == store.end())
47                {
48                    left++;
49                    continue;
50                }
51                store[s[left]]++;
52                if (store[s[left]] > 0)
53                {
54                    count++;
55                }
56                left++;
57            }
58        }
59        // cout<<minlen<<" "<<start<<" "<<end;
```

```
60     // return s.substr(start,minlen);
61     if (minlen == INT_MAX)
62     {
63         return "";
64     }
65     return s.substr(start, minlen);
66 }
67
68
69
70 int main()
71 {
72     string s = "ADOBECODEBANC";
73     string t = "ABC";
74     cout << minWindow(s, t) << endl;
75     return 0;
76 }
```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int largestRectangleArea1(vector<int> &heights)
5 {
6     int n = heights.size();
7     vector<int> left(n, 0), right(n, n);
8     stack<int> s;
9     for (int i = 0; i < n; i++)
10    {
11        while (!s.empty() && heights[i] <= heights[s.top()])
12            s.pop();
13        if (s.empty())
14            left[i] = -1; // minimum element
15        else
16            left[i] = s.top();
17        s.push(i);
18    }
19    while (!s.empty())
20        s.pop();
21    for (int i = n - 1; i >= 0; i--)
22    {
23        while (!s.empty() && heights[i] <= heights[s.top()])
24            s.pop();
25        if (s.empty())
26            right[i] = n; // minimum element
27        else
28            right[i] = s.top();
29        s.push(i);
30    }
31    int max_area = 0;
32    for (int i = 0; i < n; i++)
33    {
34        max_area = max(max_area, heights[i] * (right[i] - left[i] - 1));
35    }
36    return max_area;
37 }
38
39 int largestRectangleArea(vector<int> &histo)
40 {
41     stack<int> st;
42     int maxA = 0;
43     int n = histo.size();
44     for (int i = 0; i <= n; i++)
45     {
46         while (!st.empty() and (i == n or histo[st.top()] >= histo[i]))
47         {
48             int height = histo[st.top()];
49             st.pop();
50             int width = 0;
51             if (st.empty())
52                 width = i;
53             else
54                 width = i - st.top() - 1;
55
56             maxA = max(maxA, width * height);
57         }
58         st.push(i);
59     }
```

```
60     return maxA;
61 }
62
63 int maximalRectangle(vector<vector<char>> &matrix)
64 {
65     int maxe = 0;
66
67     int n = matrix.size();
68     if (n == 0)
69         return 0;
70     int m = matrix[0].size();
71     vector<int> vct(m, 0);
72
73     for (int i = 0; i < n; i++)
74     {
75         for (int j = 0; j < m; j++)
76         {
77             if (matrix[i][j] == '0')
78             {
79                 vct[j] = 0;
80             }
81             else
82             {
83                 vct[j] += 1;
84             }
85         }
86
87         maxe = max(maxe, largestRectangleArea(vct));
88     }
89
90     return maxe;
91 }
92
93 int main()
94 {
95     vector<vector<char>> matrix = {
96         {'1', '0', '1', '0', '0'},
97         {'1', '0', '1', '1', '1'},
98         {'1', '1', '1', '1', '1'},
99         {'1', '0', '0', '1', '0'}};
100    cout << maximalRectangle(matrix);
101    return 0;
102 }
```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int firstMissingPositive(vector<int> &nums)
5 {
6     // swap sort
7     int n = nums.size();
8     for (int i = 0; i < n; i++)
9     {
10         while (nums[i] > 0 and nums[i] <= n and nums[nums[i] - 1] != nums[i])
11         {
12             swap(nums[i], nums[nums[i] - 1]);
13         }
14     }
15     for (int i = 0; i < n; i++)
16     {
17         if (nums[i] != i + 1)
18         {
19             return i + 1;
20         }
21     }
22     return n + 1;
23 }
24
25
26 int main()
27 {
28     vector<int> nums = {1, 2, 0};
29     cout << firstMissingPositive(nums) << endl;
30     return 0;
31 }
```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 // vector<int> result;
5
6 /*
7 we are putting count for source code in this function . For articulation point we
8 need to handle source differently .
9
10         if ((low[adj] >= entrytime[node]))
11             result[node] = true;
12             and
13
14         if(parent==-1)
15         {
16             if(count>=2)
17             {
18                 result[node]=true;
19             }
20         }
21
22 if we remove both of the above mention condition then it will convert to bridge
23 question or critical connection problems.
24 */
25
26 void dfs(int node, int parent, vector<bool> &vis, vector<int> &entrytime,
27           vector<int> &low, vector<int> graph[], int &timer,vector<int>&result)
28 {
29     entrytime[node] = low[node] = timer++;
30
31     vis[node] = true;
32     int count=0;
33
34     for (auto temp : graph[node])
35     {
36         int adj = temp;
37         if (adj == parent)
38         {
39             continue;
40         }
41
42         else if (!vis[adj])
43         {
44             count++;
45             dfs(adj, node, vis, entrytime, low, graph, timer,result);
46             low[node] = min(low[node], low[adj]);
47             if(parent== -1)
48             {
49                 if(count>=2)
50                 {
51                     result[node]=true;
52                 }
53             }
54         }
55         else
56         {
57             // result.push_back({node, adj});
58             if ((low[adj] >= entrytime[node]))
```

```
57         result[node] = true;
58     }
59 }
60 else if (vis[adj])
61 {
62     low[node] = min(low[node], entrytime[adj]);
63 }
64 }
65 }
66
67 vector<int> articulationPoint(int n, vector<vector<int>> &connections)
68 {
69
70     vector<int> graph[n];
71     int m = connections.size();
72     for (int i = 0; i < m; i++)
73     {
74         graph[connections[i][0]].push_back(connections[i][1]);
75         graph[connections[i][1]].push_back(connections[i][0]);
76     }
77
78     vector<bool> vis(n, false);
79     vector<int> entrytime(n, -1);
80     vector<int> low(n, -1);
81     // result.clear();
82     vector<int> result(n, 0);
83     int timer = 0;
84     for (int i = 0; i < n; i++)
85     {
86         if (!vis[i])
87         {
88             dfs(i, -1, vis, entrytime, low, graph, timer, result);
89         }
90     }
91
92     return result;
93 }
94
95 int main()
96 {
97     int n = 4;
98     vector<vector<int>> connections = {{0, 1}, {1, 2}, {2, 0}, {1, 3}};
99     vector<int> result = articulationPoint(n, connections);
100    for (auto i : result)
101    {
102        cout << i << " ";
103    }
104    return 0;
105 }
```

```
1 /*  
2 Design a data structure that follows the constraints of a Least Recently Used (LRU)  
3 cache.  
4  
5 Implement the LRUCache class:  
6  
7 LRUCache(int capacity) Initialize the LRU cache with positive size capacity.  
8 int get(int key) Return the value of the key if the key exists, otherwise return -1.  
9 void put(int key, int value) Update the value of the key if the key exists.  
10 Otherwise, add the key-value pair to the cache. If the number of keys exceeds the  
11 capacity from this operation, evict the least recently used key.  
12 The functions get and put must each run in O(1) average time complexity.  
13 */  
14 #include<bits/stdc++.h>  
15 using namespace std;  
16  
17 class LRUCache  
18 {  
19 public:  
20     LRUCache(int capacity) : _capacity(capacity) {}  
21  
22     int get(int key)  
23     {  
24         auto it = cache.find(key);  
25         if (it == cache.end())  
26             return -1;  
27         touch(it);  
28         return it->second.first;  
29     }  
30  
31     void put(int key, int value)  
32     {  
33         auto it = cache.find(key);  
34         if (it != cache.end())  
35             touch(it);  
36         else  
37             {  
38                 if (cache.size() == _capacity)  
39                 {  
40                     cache.erase(used.back());  
41                     used.pop_back();  
42                 }  
43                 used.push_front(key);  
44             }  
45             cache[key] = {value, used.begin()};  
46     }  
47  
48 private:  
49     typedef list<int> LI;  
50     typedef pair<int, LI::iterator> PII;  
51     typedef unordered_map<int, PII> HPIII;  
52  
53     void touch(HPIII::iterator it)  
54     {  
55         int key = it->first;  
56         used.erase(it->second.second);  
57         used.push_front(key);  
58         it->second.second = used.begin();  
59     }  
60 }
```

```
57
58     HIPPII cache;
59     LI used;
60     int _capacity;
61 };
62
63 int main()
64 {
65     LRUCache lru(2);
66     lru.put(1, 1);
67     lru.put(2, 2);
68     cout << lru.get(1) << endl;
69     lru.put(3, 3);
70     cout << lru.get(2) << endl;
71     lru.put(4, 4);
72     cout << lru.get(1) << endl;
73     cout << lru.get(3) << endl;
74     cout << lru.get(4) << endl;
75     return 0;
76 }
```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 vector<int> partitionLabels(string s)
5 {
6     // vector for keeping the track of last occurrence of every character
7     vector<int> end_idx(26, 0);
8
9     for (int i = 0; i < s.length(); ++i)
10        end_idx[s[i] - 'a'] = i;
11
12    vector<int> res;
13
14    int start = 0, end = 0;
15    // scanning string character by character
16    for (int i = 0; i < s.length(); ++i)
17    {
18        // whenever we get an character we check,
19        // last index of that character
20        end = max(end, end_idx[s[i] - 'a']);
21
22        // when current i.e i == end
23        // add it to result
24        if (i == end)
25        {
26            // all the characters of current partition included
27            res.push_back(i - start + 1);
28            // update the start pointer for fresh start
29            start = i + 1;
30        }
31    }
32
33    return res;
34 }
35
36 int main()
37 {
38     string s = "ababcbacadefegdehijhklij";
39     vector<int> res = partitionLabels(s);
40     for (int i = 0; i < res.size(); ++i)
41         cout << res[i] << " ";
42     cout << endl;
43     return 0;
44 }
```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 string minRemoveToMakeValid(string s)
5 {
6     stack<int> st;
7     for (int i = 0; i < s.size(); i++)
8     {
9         if (s[i] == ')')
10        {
11            if (st.size() > 0)
12            {
13                if (s[st.top()] == '(')
14                {
15                    st.pop();
16                }
17                else
18                {
19                    st.push(i);
20                }
21            }
22            else
23            {
24                st.push(i);
25            }
26        }
27        else if (s[i] == '(')
28        {
29            st.push(i);
30        }
31    }
32    while (st.size() != 0)
33    {
34        // cout<<st.top()<<" ";
35        s[st.top()] = '.';
36        st.pop();
37    }
38    string temp = "";
39    for (int i = 0; i < s.size(); i++)
40    {
41        if (s[i] != '.')
42        {
43            temp += s[i];
44        }
45    }
46
47    return temp;
48 }
49
50 int main()
51 {
52     string s = "lee(t(c)o)de";
53     cout<<minRemoveToMakeValid(s)<<endl;
54 }
```

```
1 #include<bits/stdc++.h>
2 using namespace std;
3
4 int maxArea(vector<int> &height)
5 {
6
7     int ans = 0;
8     int left = 0;
9     int right = height.size() - 1;
10
11    while (left < right)
12    {
13        int leftelement = height[left];
14
15        int rightelement = height[right];
16
17        int mine = min(rightelement, leftelement);
18
19        ans = max(ans, (right - left) * mine);
20
21        if (leftelement < rightelement)
22        {
23            left++;
24        }
25        else
26        {
27            right--;
28        }
29    }
30
31    return ans;
32 }
33
34 int main()
35 {
36     vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
37     cout << maxArea(height) << endl;
38     return 0;
39 }
40 }
```