Jul 9, 2022

Question Name From Leetcode

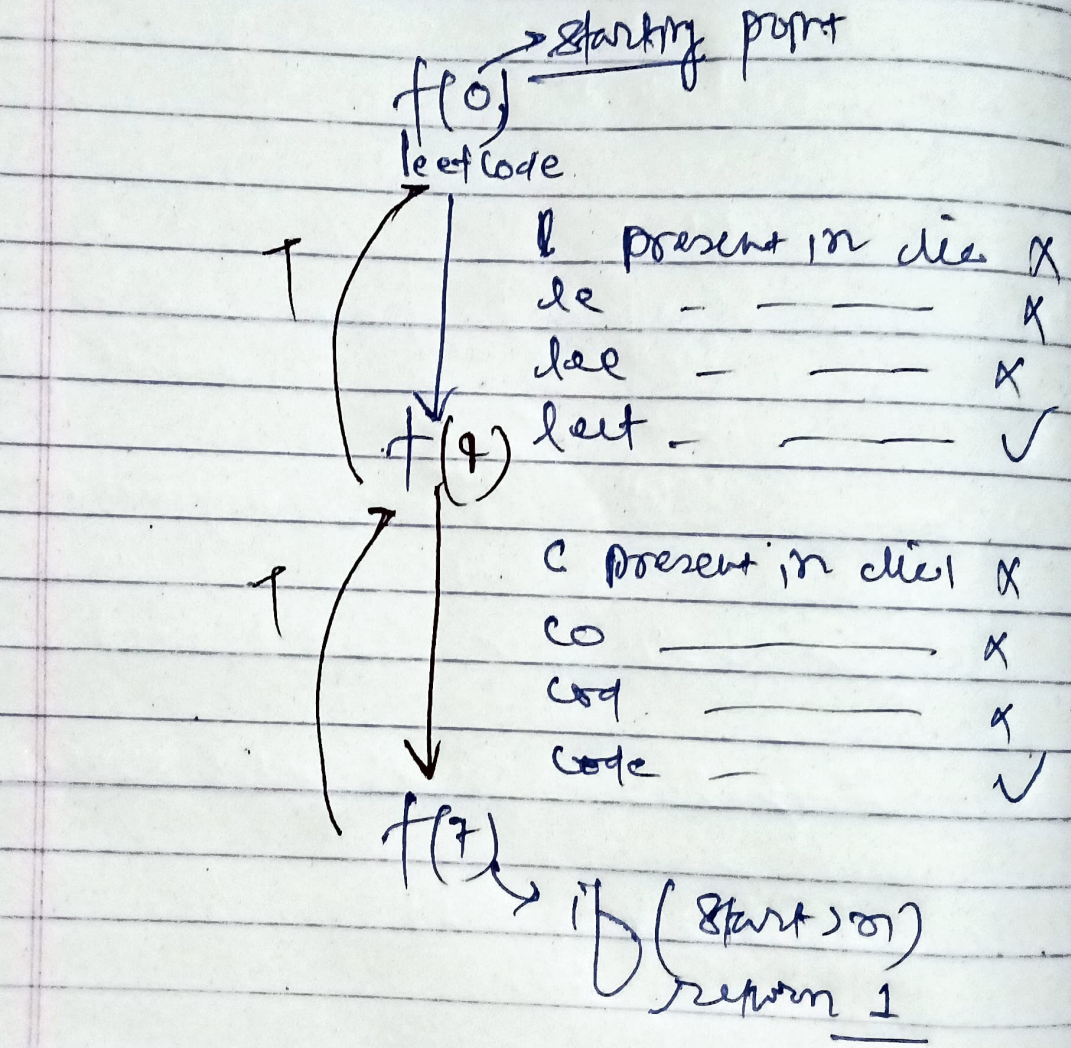
1. Trapping Rain water
2. LRU Cache
3. Longest Palindromic substring
4. Median of two sorted array
5. Critical connection in network
6. Integer to english word
7. Partition Labels
8. Minimum Remove to make valid parentheses
9. Container with most water
10. Alien Dictionary
11. Decode Ways
12. Decode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

Word Breaking

→ leet code → dict

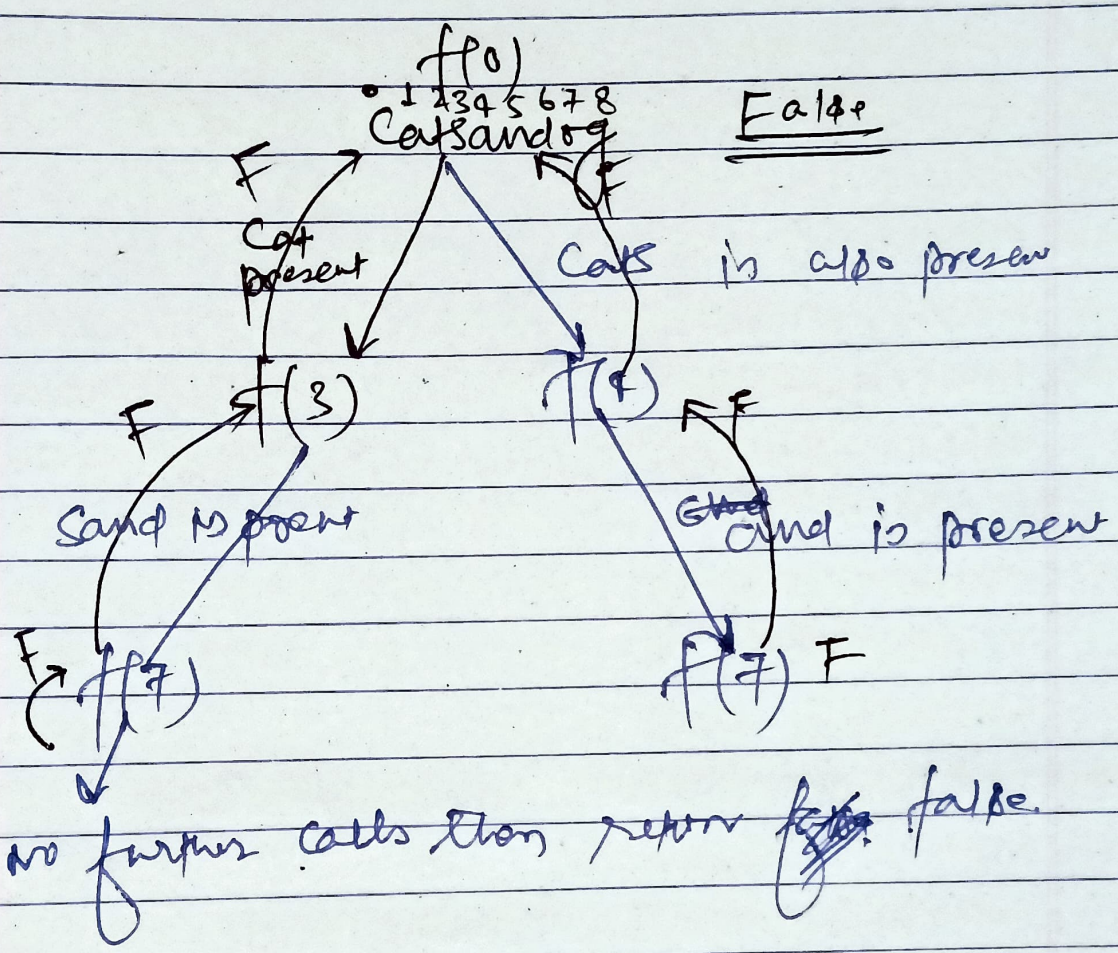
0 1 2 3 4 5 6

→ word ..leetCode



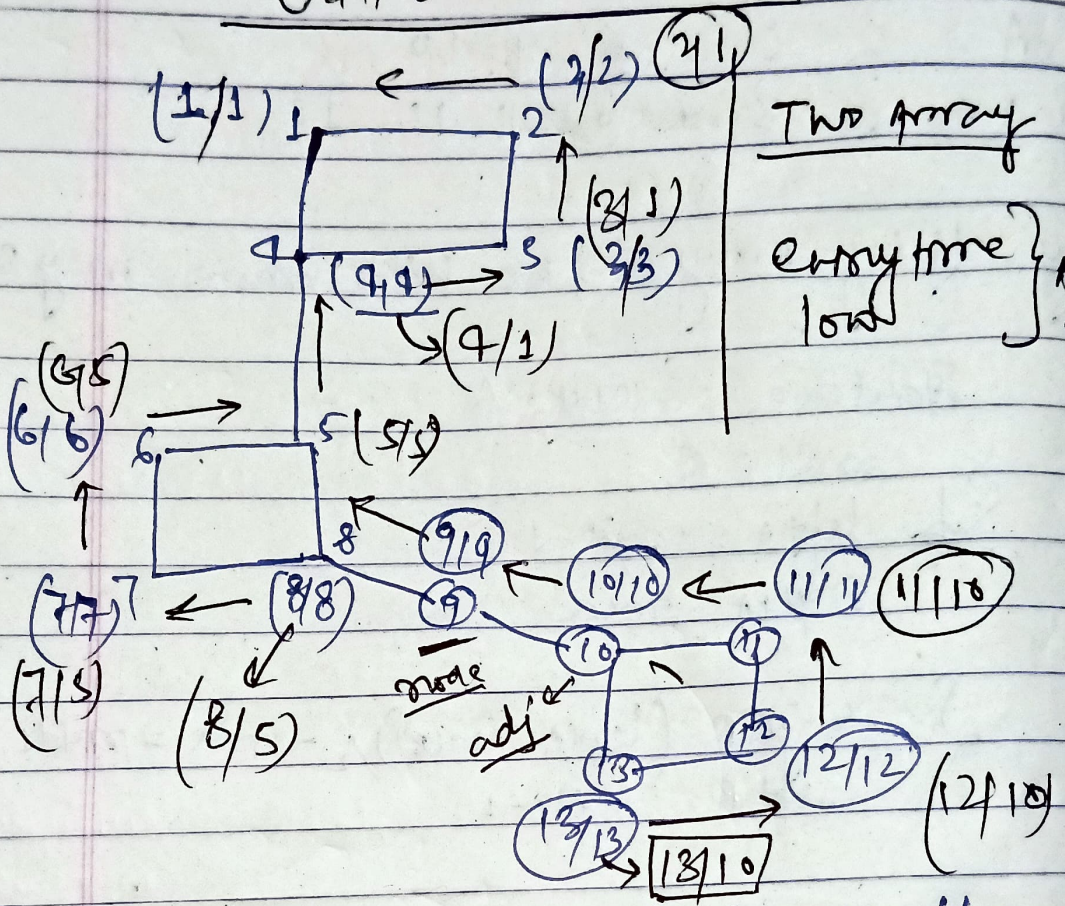
dict = Cats dog Sand and Cat

Word = Catsandog



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

bool helper(const string &s, int start, const unordered_set<string> &d, vector<int> &memo)
{
    if (start == s.size())
    {
        return true;
    }

    if (memo[start] != -1)
        return memo[start];

    for (int i = start; i <= s.size(); ++i)
    {
        const string sub = s.substr(start, i - start + 1);
        // cout<<sub<<endl;
        if (d.count(sub) == 1)
        {
            if (helper(s, i + 1, d, memo))
            {
                memo[start] = 1;
                return true;
            }
        }
    }

    memo[start] = 0;
    return false;
}

bool wordBreak(string s, vector<string> &wordDict)
{
    unordered_set<string> d(wordDict.begin(), wordDict.end());

    vector<int> memo(s.length(), -1);
    return helper(s, 0, d, memo);
}

int main()
{
    string s = "leetcode";
    vector<string> wordDict = {"leet", "code"};
    cout << wordBreak(s, wordDict) << endl;
    return 0;
}
```

Critical Connection



low → It's actually, By ignoring the actual path, what is the lowest node can reach.

→ If we found any correct edge low that edge is updated than

low[node] = min(low[node], entryTime

→ Otherwise call if next node and similarly update the low value

low[node] = min (low[node], low[adj]

Concept for Critical Connection

if(entrytime of node < low of adj)
    result.push_back({node, adj});

→ low of adj (low[adj]) = lowest can be reached by ignoring the actual path

→ entrytime[node] = i/p current entrytime.

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

vector<vector<int>> result;

void dfs(int node, int parent, vector<bool> &vis, vector<int> &entrytime, vector<int> &low, vector<int> graph[], int &timer)
{
    entrytime[node] = low[node] = timer++;

    vis[node] = true;

    for (auto temp : graph[node])
    {
        int adj = temp;
        if (adj == parent)
        {
            continue;
        }

        else if (!vis[adj])
        {
            dfs(adj, node, vis, entrytime, low, graph, timer);
            low[node] = min(low[node], low[adj]);
            if (low[adj] > entrytime[node])
            {
                result.push_back({node, adj});
            }
        }
        else if (vis[adj])
        {
            low[node] = min(low[node], entrytime[adj]);
        }
    }
}

vector<vector<int>> criticalConnections(int n, vector<vector<int>> &connections)
{

    vector<int> graph[n];
    int m = connections.size();
    for (int i = 0; i < m; i++)
    {
        graph[connections[i][0]].push_back(connections[i][1]);
        graph[connections[i][1]].push_back(connections[i][0]);
    }

    vector<bool> vis(n, false);
    vector<int> entrytime(n, -1);
    vector<int> low(n, -1);
    result.clear();
    int timer = 0;
    for (int i = 0; i < n; i++)
    {
        if (!vis[i])
        {
            dfs(i, -1, vis, entrytime, low, graph, timer);
        }
    }
```

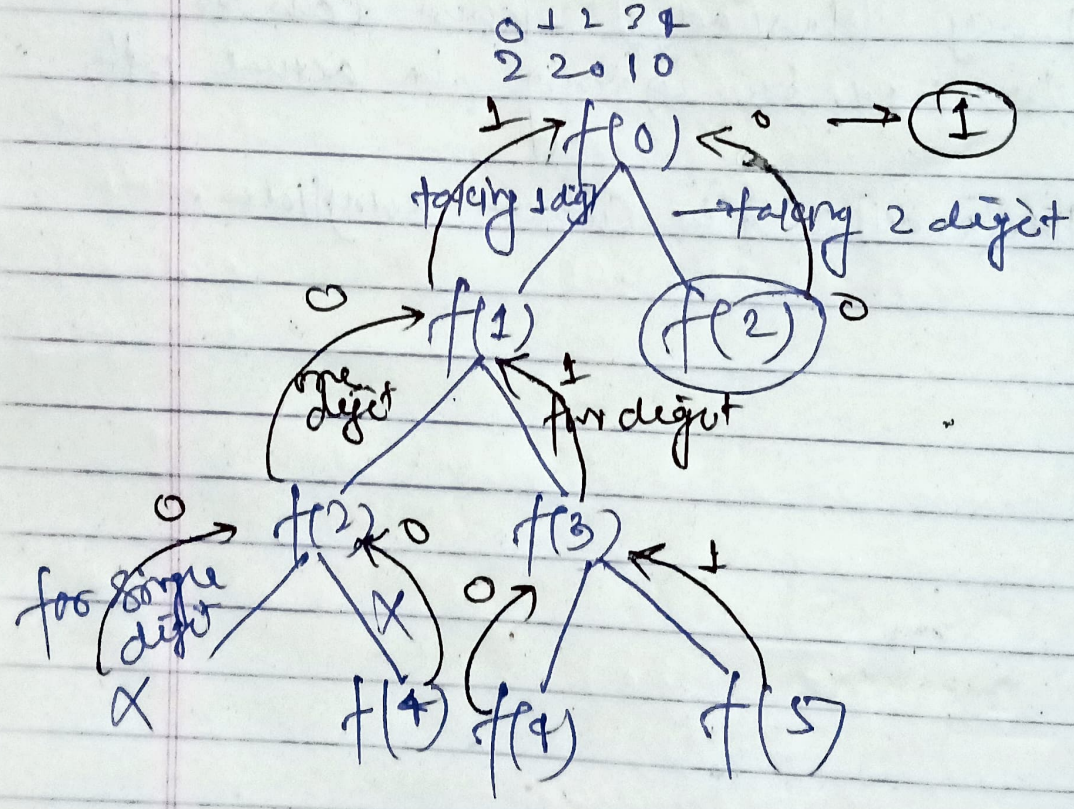
```
59
60      return result;
61  }
62
63  int main()
64  {
65      int n = 4;
66      vector<vector<int>> connections = {{0, 1}, {1, 2}, {2, 0}, {1, 3}};
67      vector<vector<int>> result = criticalConnections(n, connections);
68      for (auto temp : result)
69      {
70          cout << temp[0] 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

Decode String

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

① Taking only one digit

② Taking two digit a time.



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

class Solution
{
public:
    int tabulation(string &input)
    {
        int n = input.size();
        vector<int> dp(n + 1, 0);
        dp[0] = 1;

        for (int start = 1; start <= n; start++)
        {
            int takingOne = 0;
            int takingtwo = 0;

            if (input[start - 1] != '0')
            {
                takingOne = dp[start - 1];
            }
            if (start - 2 >= 0)
            {
                int number = (input[start - 2] - '0') * 10 + (input[start - 1] - '0');
                if (number >= 10 and number <= 26)
                {
                    takingtwo = dp[start - 2];
                }
            }
            dp[start] = (takingOne + takingtwo);
        }

        return dp[n];
    }

    int result(string &input, int start)
    {
        if (start < 0)
        {
            return 1;
        }

        int takingOne = 0;
        int takingtwo = 0;

        if (input[start] != '0')
        {
            takingOne = result(input, start - 1);
        }
        if (start - 1 >= 0)
        {
            int number = (input[start - 1] - '0') * 10 + (input[start] - '0');
            if (number >= 10 and number <= 26)
            {
                takingtwo = result(input, start - 2);
            }
        }
        return (takingOne + takingtwo);
```

```
    }
    int numDecodings(string s)
    {

        // return result(s,s.size()-1);
        return tabulation(s);
    }
};


int main()
{
    Solution s;
    string input = "12";
    cout << s.numDecodings(input) << endl;
    return 0;
}
```

Median of two Sorted Array

Arr1 = 2, 5, 7, 8, 9, 10
Arr2 = 1, 3, 4, 6, 11, 12, 13

→ if length of both array is odd then median will be max(Arr1[left1], Arr2[left2])

→ if array length is even then median will be

$$\frac{\max(Arr_1[left_1], Arr_2[left_2]) + \min(Arr_1[right_1], Arr_2[right_2])}{2}$$

→ 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 = 6.5 (Median)

	0	1	2	3	4	5
A =	2	5	7	8	9	10
B =	1	3	4	6	11	12	13

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

start = 0 end = 6

mid = 3

{ leftA = mid - 1
  rightA = mid

{ rightB = (len(A) + len(B))/2 - mid = mid1
  leftB = mid1 - 1

For iteration we need to take care of 3 cases.

① if (leftB > rightA)
    s = mid1 + 1;
else if (leftA > rightB)
    e = mid1 - 1;

else;

int total = (m+n);
if (total % 2 != 0)
    return max(leftA, leftB);
else

return (max(leftA, leftB) + min(rightA, rightB))/2

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

double findMedianSortedArrays(vector<int> &A, vector<int> &B)
{

    int n = A.size();
    int m = B.size();

    if (n > m)
        return findMedianSortedArrays(B, A);

    int totalLeft = (n + m + 1) / 2;

    int s = 0, e = n;

    while (s <= e)
    {
        int mid = (s + e) / 2;
        int countA = mid;
        int countB = totalLeft - countA;

        double leftA = (countA == 0) ? INT_MIN : A[countA - 1];
        double rightA = (countA == n) ? INT_MAX : A[countA];

        double leftB = (countB == 0) ? INT_MIN : B[countB - 1];
        double rightB = (countB == m) ? INT_MAX : B[countB];

        if (leftB > rightA)
            s = mid + 1;
        else if (leftA > rightB)
            e = mid - 1;
        else
        {
            int total = (n + m);
            if (total % 2 != 0)
                return double(max(leftA, leftB));
            else
                return (double(max(leftA, leftB)) + double(min(rightA, rightB))) / 2;
        }
    }
    return 0;
}

int main()
{
    vector<int> A={2,5,7,8,9,10};
    vector<int> B={1,3,4,6,11,12,13};
    double res=findMedianSortedArrays(A,B);
    cout<<res<<endl;
}
```

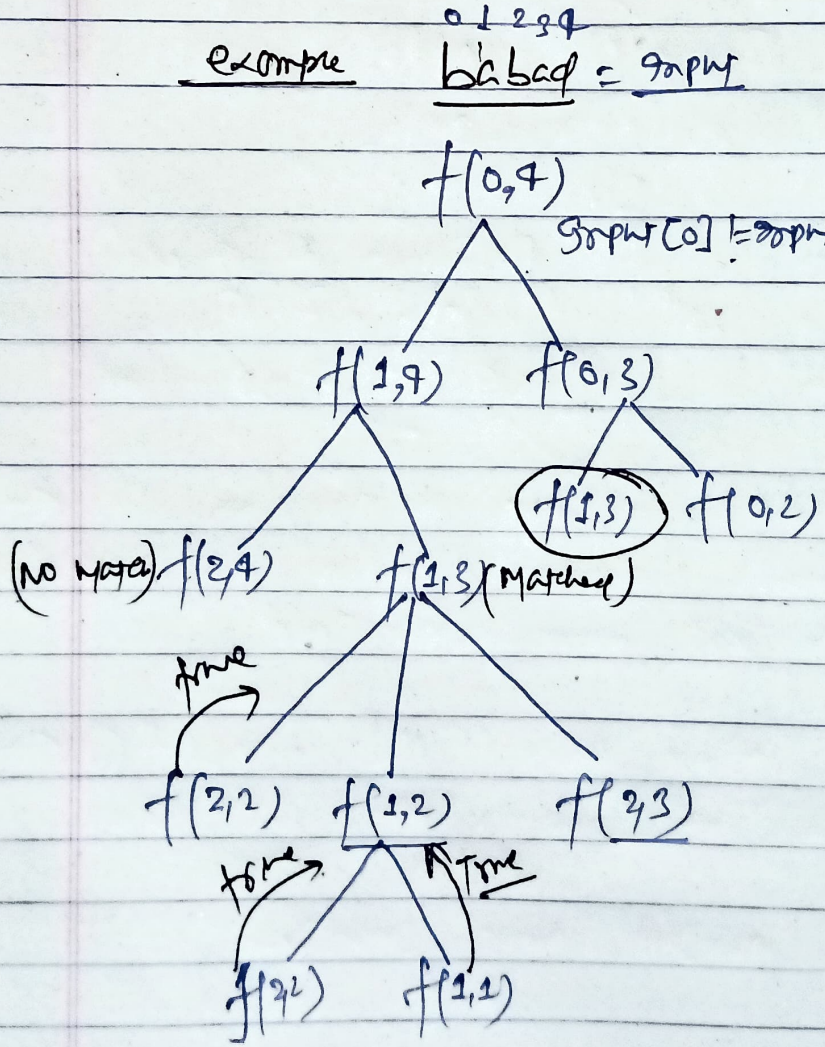
Longest palindromic Substring

→ If corner matched then go for three calls

→ If corner not matched then go 2 calls on

And at every point call the palindrome length if that segment is palindrom

example babad = ans



```cpp
#include<bits/stdc++.h>
using namespace std;
string tabulation(string s)
{
            int n= s.size();

            vector<vector<int>>dp(n,vector<int>(n,0));
            int start =0;
            int end=0;
            int maxlen=0;
            for(int g=0;g<n;g++)
            {
                for(int i=0,j=g;j<n;j++,i++)
                {
                    if(g==0)
                    {
                        dp[i][j]=true;
                    }
                    else if(g==1)
                    {
                        if(s[i]==s[j])
                        {
                            dp[i][j]=true;
                        }
                        else
                        {
                            dp[i][j]=false;
                        }
                    }
                    else
                    {
                        if(s[i]==s[j])
                        {
                            dp[i][j]=dp[i+1][j-1];
                        }
                        else
                        {
                            dp[i][j]=0;
                        }
                    }
                    if(dp[i][j])
                    {
                        int length= (j-i+1);
                        if(length>maxlen)
                        {
                            maxlen= length;
                            start= i;
                            end=j;
                        }

                    }
                }
            }
            //  cout<<start<<" "<<end<<" "<<maxlen<<endl;
            //  for(int i=0;i<n;i++)
            //  {
            //   for(int j=0;j<n;j++)
            //   {
            //       cout<<dp[i][j]<<" ";
```

```
            //   }
            //    cout<<endl;
            // }
            string result="";
            for(int i=start;i<=end;i++)
            {
               result+=s[i];
            }
             result= s.substr(start,end+1);
            return result;
}

bool helper(int l, int r, string &s, int memo[][1001], vector<int> &answer)
{
    // Check if [ L ... R ] is already calculated before
    if (memo[l][r] != -1)
    {
        return memo[l][r];
    }

    // Base case, every single letter or empty string is also a palindrome
    if (l >= r)
    {
        return true;
    }

    // Possible palindrome beginning
    bool found = false;
    if (s[l] == s[r])
    {
        found = helper(l + 1, r - 1, s, memo, answer);
    }

    // Try other possibilities
    helper(l + 1, r, s, memo, answer);
    helper(l, r - 1, s, memo, answer);

    // Update longest palindrome position and width
    if (found && (r - l + 1) > answer[2])
    {
        answer = {l, r, (r - l + 1)};
    }

    // Memoization
    memo[l][r] = found;

    return found;
}

string longestPalindrome(string s)
{
    int memo[1001][1001];
    memset(memo, -1, sizeof(memo));
    vector<int> answer = {0, 0, 1};
    helper(0, s.size() - 1, s, memo, answer);
    // cout<<answer[0] << " " << answer[1] << " " << answer[2] << endl;
    return s.substr(answer[0], answer[2]);
}


```

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

int maxArea(vector<int> &height)
{

    int ans = 0;
    int left = 0;
    int right = height.size() - 1;

    while (left < right)
    {
        int leftelement = height[left];

        int rightelement = height[right];

        int mine = min(rightelement, leftelement);

        ans = max(ans, (right - left) * mine);

        if (leftelement < rightelement)
        {
            left++;
        }
        else
        {
            right--;
        }
    }

    return ans;
}

int main()
{
    vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    cout << maxArea(height) << endl;
    return 0;

}
```

```cpp
int main()
{
    string s = "babad";
    cout << longestPalindrome(s) << endl;
    // cout<<"raushan kumar"<<endl;
    return 0;
}
```

```cpp
// 1st approach  by taking 2 vector
// 2nd approach by taking two pointer and two variable to track the max height and the max index

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

// 1st approach
/*
     1st vector is maxfromleft side   and second vector will be maxfromrightside
     and result will be
     rest+= min(maxfromleft, maxfromright)-height[i]
 */
int trap1(int A[],int n)
{
    int result=0;
    vector<int> maxfromleft(n,0);
    vector<int> maxfromright(n,0);
    maxfromleft[0]=A[0];
    maxfromright[n-1]=A[n-1];
    for(int i=1;i<n;i++)
    {
        maxfromleft[i]=max(maxfromleft[i-1],A[i]);
    }
    for(int i=n-2;i>=0;i--)
    {
        maxfromright[i]=max(maxfromright[i+1],A[i]);
    }
    for(int i=0;i<n;i++)
    {
        result+=min(maxfromleft[i],maxfromright[i])-A[i];
    }
    return result;
}


// 2nd approach
int trap(int A[], int n)
{
    int left = 0;
    int right = n - 1;
    int res = 0;
    int maxleft = 0, maxright = 0;
    while (left <= right)
    {
        if (A[left] <= A[right])
        {
            if (A[left] >= maxleft)
                maxleft = A[left];
            else
                res += maxleft - A[left];
            left++;
        }
        else
        {
            if (A[right] >= maxright)
                maxright = A[right];
            else
                res += maxright - A[right];
```

```
            right--;
        }
    }
    return res;
}

int main()
{
    int A[] = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};
    int n = sizeof(A) / sizeof(A[0]);
    cout << trap(A, n);
    return 0;
}
/*
time complexity: O(n)
*/


```

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

vector<char>calculate(string input[],int n)
{
   vector<int>indegree(26,-1);
   for(int i=0;i<n;i++)
   {
         for(int j=0;j<input[i].size();j++)
         {
              indegree[input[i][j]-'a']=0;
         }
   }

//    making the graph
    unordered_map<char,vector<char>>graph;

    for(int i=0;i<n-1;i++)
    {
        string first= input[i];
        string second= input[i+1];
        for(int j=0;j<min(first.size(),second.size());j++)
        {
            if(first[j]!=second[j])
            {
                graph[first[j]].push_back(second[j]);
                break;
            }
        }
    }
    vector<char>result;
    // now calculate the indegree of the graph
    for(auto i=graph.begin();i!=graph.end();i++)
    {
        for(int j=0;j<i->second.size();j++)
        {
            indegree[i->second[j]-'a']++;
        }
    }
    // print the indegree of the graph
    queue<char>q;
    for(int i=0;i<26;i++)
    {
        if(indegree[i]==0)
        {
            result.push_back(i+'a');
            q.push(i+'a');
        }
    }

    while(!q.empty())
    {
        char curr=q.front();
        q.pop();
        for(int i=0;i<graph[curr].size();i++)
        {
            indegree[graph[curr][i]-'a']--;
            if(indegree[graph[curr][i]-'a']==0)
            {
```

```
                result.push_back(graph[curr][i]);
                q.push(graph[curr][i]);
            }
        }
    }
    return result;


}



int main()
{
   string words[] = { "baa",
                "abcd",
                "abca",
                "cab",
                "cad" };

    int n = 5;
    vector<char>order=calculate(words, n);
    for(int i=0;i<order.size();i++)
    {
        cout<<order[i]<<" ";
    }

    return 0;
}
```

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

    string minWindow(string s, string t)
    {

        unordered_map<char, int> store;
        int nt = t.size();
        for (int i = 0; i < nt; i++)
        {
            store[t[i]]++;
        }

        int count = store.size();

        int ns = s.size();
        int start = -1;
        int end = -1;
        int minlen = INT_MAX;
        int left = 0;
        for (int right = 0; right < ns; right++)
        {
            if (store.find(s[right]) == store.end())
            {
                continue;
            }
            store[s[right]]--;
            if (store[s[right]] == 0)
            {
                count--;
            }

            while (count == 0 and left <= right)
            {
                if (count == 0)
                {
                    // cout<<count<<" "<<left<<" "<<right<<endl;
                    int lent = (right - left + 1);
                    if (minlen > lent)
                    {
                        minlen = lent;
                        start = left;
                        end = right;
                    }
                }
                if (store.find(s[left]) == store.end())
                {
                    left++;
                    continue;
                }
                store[s[left]]++;
                if (store[s[left]] > 0)
                {
                    count++;
                }
                left++;
            }
        }
        // cout<<minlen<<" "<<start<<" "<<end;
```

```
60
61          // return s.substr(start,minlen);
62          if (minlen == INT_MAX)
63          {
64              return "";
65          }
66          return s.substr(start, minlen);
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  }
```

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

int largestRectangleArea1(vector<int> &heights)
{
    int n = heights.size();
    vector<int> left(n, 0), right(n, n);
    stack<int> s;
    for (int i = 0; i < n; i++)
    {
        while (!s.empty() && heights[i] <= heights[s.top()])
            s.pop();
        if (s.empty())
            left[i] = -1; // minimum element
        else
            left[i] = s.top();
        s.push(i);
    }
    while (!s.empty())
        s.pop();
    for (int i = n - 1; i >= 0; i--)
    {
        while (!s.empty() && heights[i] <= heights[s.top()])
            s.pop();
        if (s.empty())
            right[i] = n; // minimum element
        else
            right[i] = s.top();
        s.push(i);
    }
    int max_area = 0;
    for (int i = 0; i < n; i++)
    {
        max_area = max(max_area, heights[i] * (right[i] - left[i] -1));
    }
    return max_area;
}

int largestRectangleArea(vector<int> &histo)
{
    stack<int> st;
    int maxA = 0;
    int n = histo.size();
    for (int i = 0; i <= n; i++)
    {
        while (!st.empty() and (i == n or histo[st.top()] >= histo[i]))
        {
            int height = histo[st.top()];
            st.pop();
            int width = 0;
            if (st.empty())
                width = i;
            else
                width = i - st.top() - 1;

            maxA = max(maxA, width * height);
        }
        st.push(i);
    }
```

```cpp
    return maxA;
}

int maximalRectangle(vector<vector<char>> &matrix)
{
    int maxe = 0;

    int n = matrix.size();
    if (n == 0)
        return 0;
    int m = matrix[0].size();
    vector<int> vct(m, 0);

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (matrix[i][j] == '0')
            {
                vct[j] = 0;
            }
            else
            {
                vct[j] += 1;
            }
        }

        maxe = max(maxe, largestRectangleArea(vct));
    }

    return maxe;
}

int main()
{
    vector<vector<char>> matrix = {
        {'1', '0', '1', '0', '0'},
        {'1', '0', '1', '1', '1'},
        {'1', '1', '1', '1', '1'},
        {'1', '0', '0', '1', '0'}};
    cout << maximalRectangle(matrix);
    return 0;
}
```

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

int firstMissingPositive(vector<int> &nums)
{
    // swap sort
    int n = nums.size();
    for (int i = 0; i < n; i++)
    {
        while (nums[i] > 0 and nums[i] <= n and nums[nums[i] - 1] != nums[i])
        {
            swap(nums[i], nums[nums[i] - 1]);
        }
    }
    for (int i = 0; i < n; i++)
    {
        if (nums[i] != i + 1)
        {
            return i + 1;
        }
    }
    return n + 1;
}


int main()
{
    vector<int> nums = {1, 2, 0};
    cout << firstMissingPositive(nums) << endl;
    return 0;
}
```

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

// vector<int> result;

/*
we are putting count for source code in this function .  For articulation point we need to handle source differently .

              if ((low[adj] >= entrytime[node]))
                    result[node] = true;
                    and


            if(parent==-1)
            {
                  if(count>=2)
                  {
                    result[node]=true;
                  }
            }

 if we remove both of the above mention condition  then it will convert to bridge question or ciritical connection problems.

*/

void dfs(int node, int parent, vector<bool> &vis, vector<int> &entrytime, vector<int> &low, vector<int> graph[], int &timer,vector<int>&result)
{
    entrytime[node] = low[node] = timer++;

    vis[node] = true;
    int count=0;

    for (auto temp : graph[node])
    {
        int adj = temp;
        if (adj == parent)
        {
            continue;
        }

        else if (!vis[adj])
        {
            count++;
            dfs(adj, node, vis, entrytime, low, graph, timer,result);
            low[node] = min(low[node], low[adj]);
            if(parent==-1)
            {
                  if(count>=2)
                  {
                    result[node]=true;
                  }
            }
            else
            {
                // result.push_back({node, adj});
                if ((low[adj] >= entrytime[node]))
```

```
                    result[node] = true;
            }
        }
        else if (vis[adj])
        {
            low[node] = min(low[node], entrytime[adj]);
        }
    }
}

vector<int> articulationPoint(int n, vector<vector<int>> &connections)
{

    vector<int> graph[n];
    int m = connections.size();
    for (int i = 0; i < m; i++)
    {
        graph[connections[i][0]].push_back(connections[i][1]);
        graph[connections[i][1]].push_back(connections[i][0]);
    }

    vector<bool> vis(n, false);
    vector<int> entrytime(n, -1);
    vector<int> low(n, -1);
    // result.clear();
    vector<int>result(n,0);
    int timer = 0;
    for (int i = 0; i < n; i++)
    {
        if (!vis[i])
        {
            dfs(i, -1, vis, entrytime, low, graph, timer,result);
        }
    }

    return result;
}

int main()
{
    int n = 4;
    vector<vector<int>> connections = {{0, 1}, {1, 2}, {2, 0}, {1, 3}};
    vector<int> result = articulationPoint(n, connections);
    for (auto i : result)
    {
        cout << i << " ";
    }
    return 0;
}
```

```cpp
/*
Design a data structure that follows the constraints of a Least Recently Used (LRU) cache.

Implement the LRUCache class:

LRUCache(int capacity) Initialize the LRU cache with positive size capacity.
int get(int key) Return the value of the key if the key exists, otherwise return -1.
void put(int key, int value) Update the value of the key if the key exists. Otherwise, add the key-value pair to the cache. If the number of keys exceeds the capacity from this operation, evict the least recently used key.
The functions get and put must each run in O(1) average time complexity.
*/
#include<bits/stdc++.h>
using namespace std;

class LRUCache
{
public:
    LRUCache(int capacity) : _capacity(capacity) {}

    int get(int key)
    {
        auto it = cache.find(key);
        if (it == cache.end())
            return -1;
        touch(it);
        return it->second.first;
    }

    void put(int key, int value)
    {
        auto it = cache.find(key);
        if (it != cache.end())
            touch(it);
        else
        {
            if (cache.size() == _capacity)
            {
                cache.erase(used.back());
                used.pop_back();
            }
            used.push_front(key);
        }
        cache[key] = {value, used.begin()};
    }

private:
    typedef list<int> LI;
    typedef pair<int, LI::iterator> PII;
    typedef unordered_map<int, PII> HIPII;

    void touch(HIPII::iterator it)
    {
        int key = it->first;
        used.erase(it->second.second);
        used.push_front(key);
        it->second.second = used.begin();
    }
```

```
57
58      HIPII 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  }
```

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

vector<int> partitionLabels(string s)
{
    // vector for keeping the track of  last occurence of every character
    vector<int> end_idx(26, 0);

    for (int i = 0; i < s.length(); ++i)
        end_idx[s[i] - 'a'] = i;

    vector<int> res;

    int start = 0, end = 0;
    // scanning string character by character
    for (int i = 0; i < s.length(); ++i)
    {
        // whenever we get an character we check,
        // last index of that character
        end = max(end, end_idx[s[i] - 'a']);

        // when current i.e i == end
        // add it to result
        if (i == end)
        {
            // all the characters of current partition included
            res.push_back(i - start + 1);
            // update the start pointer for fresh start
            start = i + 1;
        }
    }

    return res;
}

int main()
{
    string s = "ababcbacadefegdehijhklij";
    vector<int> res = partitionLabels(s);
    for (int i = 0; i < res.size(); ++i)
        cout << res[i] << " ";
    cout << endl;
    return 0;
}
```

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

string minRemoveToMakeValid(string s)
{
    stack<int> st;
    for (int i = 0; i < s.size(); i++)
    {
        if (s[i] == ')')
        {
            if (st.size() > 0)
            {
                if (s[st.top()] == '(')
                {
                    st.pop();
                }
                else
                {
                    st.push(i);
                }
            }
            else
            {
                st.push(i);
            }
        }
        else if (s[i] == '(')
        {
            st.push(i);
        }
    }
    while (st.size() != 0)
    {
        // cout<<st.top()<<" ";
        s[st.top()] = '.';
        st.pop();
    }
    string temp = "";
    for (int i = 0; i < s.size(); i++)
    {
        if (s[i] != '.')
        {
            temp += s[i];
        }
    }

    return temp;
}

int main()
{
     string s = "lee(t(c)o)de)";
    cout<<minRemoveToMakeValid(s)<<endl;
}
```

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

int maxArea(vector<int> &height)
{

    int ans = 0;
    int left = 0;
    int right = height.size() - 1;

    while (left < right)
    {
        int leftelement = height[left];

        int rightelement = height[right];

        int mine = min(rightelement, leftelement);

        ans = max(ans, (right - left) * mine);

        if (leftelement < rightelement)
        {
            left++;
        }
        else
        {
            right--;
        }
    }

    return ans;
}

int main()
{
    vector<int> height = {1, 8, 6, 2, 5, 4, 8, 3, 7};
    cout << maxArea(height) << endl;
    return 0;

}
```