Stack and Queue Implementation

1614. Maximum Nesting Depth of the Parentheses

```
i C++

    Autocomplete

   1 •
         class Solution {
   2
         public:
   3 ▼
              int maxDepth(string s) {
   4
                  int i=0;
   5
                  int max=0;
   6
                  int len=0;
   7
                  string l,m,n;
                  m="(";
n=")";
   8
   9
  10
                  for(i=0;i<s.length();i++)</pre>
  11 ▼
  12
                       l=s[i];
  13
                       if(1.compare(m)==0)
  14
  15 ▼
                           len++;
  16
  17
                   if(1.compare(n)==0)
  18
  19 v
  20
                        len--;
  21
                   if(len>max)
  22
  23 ▼
                        max=len;
  24
  25
  26
  27
                  return max;
  28
  29
         }:
```

1021. Remove Outermost Parentheses

```
i C++

    Autocomplete

      class Solution {
 1 *
 2
      public:
           string removeOuterParentheses(string s) {
 3 *
               string str="";
 4
 5
               int count=0;
               for(char c:s)
 6
 7 +
               {if(c=='(')
               { if(count++)
 8 +
 9 *
10
                        str=str+c;
11
                if(c==')')
12
                {if(--count)
13 v
14 *
15
                        str=str+c;
16
17
                   }}
18
19
               return str;
20
21
      };
```

897. Increasing Order Search Tree

```
i C++

    Autocomplete

  10
  11
  12 v
        class Solution {
  13
        public:
             void pushall(TreeNode* root,stack<TreeNode*>& s)
  14
  15 *
             {while(root->left)
  16 v
                 {root=root->left;
                  cout<<root->val;
  17
  18
                     s.push(root);}
                 return;}
  19
  20 ₹
             TreeNode* increasingBST(TreeNode* root) {
                 if(root==NULL)
  21
  22
                     return NULL;
  23
                 stack<TreeNode*> s;
  24
                 s.push(root);
  25
                 pushall(root,s);
                 TreeNode* newroot=s.top();
  26
                 TreeNode* prev=NULL;
  27
                 while(!s.empty())
  28
  29 *
                 {TreeNode* temp=s.top();
  30
                  s.pop();
                  if(temp->right)
  31
                  {s.push(temp->right);
  32 ₹
                      pushall(temp->right,s);}
  33
  34
                  if(prev!=NULL)
  35 ₹
                  {prev->left=NULL;
  36
                      prev->right=temp;}
  37
                  prev=temp;}
  38
                 prev->left=NULL;
  39
                 prev->right=NULL;
  40
                 return newroot;
  41
  42
        };
  43
```

1475. Final Prices With a Special Discount in a Shop

```
i C++

    Autocomplete

        class Solution {
   1 *
   2
        public:
            vector<int> finalPrices(vector<int>& prices) {
   3 *
                 vector<int> ans;
   4
                 int b,i,j,cur,flag;
   5
                 for(i=0;i<prices.size();i++)</pre>
   6
   7 +
                 {cur=prices[i];
   8
   9
                  flag=0;
                  for(j=i+1;j<prices.size();j++)</pre>
 10
 11 *
                  {
                       if(cur>=prices[j])
 12
 13 *
                        ans.push_back(cur-prices[j]);
 14
 15
                           flag=1;
 16
                           break;
 17
 18
                         if(j==(prices.size()-1))
 19
                           ans.push_back(cur);
  20
                 ans.push_back(prices[prices.size()-1]);
 21
 22
                 return ans;
 23
             }
 24
        };
```

682. Baseball Game

```
i C++

    Autocomplete

        class Solution {
   1 *
   2
        public:
             int calPoints(vector<string>& ops) {
   3 ▼
   4
                 stack<int> s;
   5
                 for(auto it: ops)
   6 ▼
                      if(it=="+")
   7
                     {int a=s.top();
   8 🔻
   9
                       s.pop();
                       int b=s.top();
  10
  11
                       s.pop();
  12
                       s.push(b);
  13
                        s.push(a);
  14
                       s.push(a+b);
  15
                      else if(it=="D")
  16
  17
                          s.push(s.top()*2);
  18
                      else if(it=="C")
  19
                          s.pop();
  20
                     else
  21 ▼
                      {int a=stoi(it);
  22
                          s.push(a);}
  23
  24
                 int sum=0;
  25
                 while(!s.empty())
  26 ▼
  27
                      sum+=s.top();
  28
                      s.pop();
  29
  30
                 return sum;
  31
             }
  32
        };
```

1047. Remove All Adjacent Duplicates In String

```
i C++

    Autocomplete

   2 🔻
        class Solution {
        public:
   3
            string removeDuplicates(string s) {
  4 ▼
   5
        // stack<char> st;
   6
        // string ans="";
        // for(auto curr:s) {
   7
                if(st.empty()) st.push(curr);
  8
        //
                else if(st.top() == curr) st.pop();
  9
                else st.push(curr);
  10
        //
        // }while(!st.empty()) {
 11
                ans += st.top();
  12
        //
 13
                st.pop();
 14
        // reverse(ans.begin(), ans.end());
 15
  16
                   return ans;
 17
 18
 19
                stack<char> st;
                string ans="";
  20
                for(auto c:s)
 21
                {if(st.empty())
 22 ▼
                         st.push(c);
 23
                     else if(st.top()==c)
  24
  25
                     st.pop();
 26
                      else
  27
                         st.push(c); }
                while(!st.empty())
  28
  29 ▼
                   ans+=st.top();
  30
                     st.pop();
  31
         reverse(ans.begin(),ans.end());
  32
 33
               return ans;
  34
  35
        };
```

496. Next Greater Element I

```
i C++

    Autocomplete

        class Solution {
  1 +
        public:
  2
  3 *
            vector<int> nextGreaterElement(vector<int>& nums1, vector<int>& nums2) {
            unordered_map<int, int> mpp;
  4
  5
                 stack<int> s;
                 vector<int> ans;
  6
  7
                for(int i=nums2.size()-1;i>=0;i--)
  8 +
  9
        11
                    {if(s.size()==0)
        11
 10
 11
        11
 12
                     if(!s.empty() && s.top()>nums2[i])
 13
                     mpp[nums2[i]]=s.top();
 14
                 else if( !s.empty() && s.top()<nums2[i])</pre>
 15
 16 *
                      while(!s.empty() && s.top()<nums2[i] )</pre>
 17
 18 *
 19
                          s.pop();
 20
                      if(!s.empty())
 21
 22
                          mpp[nums2[i]]=s.top();
 23
 24
                 s.push(nums2[i]);
 25
                     for(auto it: nums1)
 26
 27 ▼
                     ans.push_back(mpp.count(it) ? mpp[it] : -1);
 28
 29
 30
 31
                 return ans;
 32
        };
```

1700. Number of Students Unable to Eat Lunch

```
i C++
                                                                            i {} 0

    Autocomplete

        class Solution {
   1 +
   2
        public:
             int countStudents(vector<int>& students, vector<int>& sandwiches) {
   3 *
   4
                 int student_count = students.size(), circular_stu=0, square_stu=0;
   5 +
            for(int i=0; i<students.size(); i++){</pre>
                 if(students[i] == 0){circular_stu++;}
   6
   7
                 else{square_stu++;}
   8
   9 +
             for(int i=0; i<sandwiches.size(); i++){</pre>
                 if(sandwiches[i] == 0){
  10 *
  11 *
                     if(circular_stu > 0){
                         student_count--;
  12
                         circular stu--;
  13
  14
                     else{return student count;}
  15
  16
                 else{
  17 v
                     if(square_stu > 0){
  18 *
  19
                         student count--;
  20
                         square_stu--;
  21
                     else{return student count;}
  22
  23
  24
  25
             return student count;
  26
  27
        };
```

1598. Crawler Log Folder

```
i C++

    Autocomplete

       class Solution {
 1 •
  2
       public:
           int minOperations(vector<string>& logs) {
  3 ₹
 4
                int x=0;
                for(auto it: logs)
  5
  6 ▼
                    if(it=="../")
  7
 8 🔻
                    {x++;
 9
                     if(x>0)
 10
                         x=0;
11
                    else if(it!="./")
12
13
                        X--;
14
15
                 return x \ge 0 ? 0 : -x;
16
17
       };
```

20. Valid Parentheses

```
C++

    Autocomplete

       class Solution {
  1 +
  2
       public:
  3 *
           bool isValid(string s) {
  4
                stack<char> st;
  5
                for(auto it: s)
  6 +
                    if(it=='(' || it=='[' || it=='{')
  7
  8
                    st.push(it);
                    else
  9
 10 +
                    { if(st.empty())
                        return false;
 11
                        else if(st.top()=='('
                                                 && it!=')')
 12
 13
                             return false;
                        else if(st.top()=='{' && it!='}')
 14
 15
                             return false;
 16
                             else if(st.top()=='[' && it!=']')
 17
                                 return false;
                        else
 18
 19
                             st.pop();
                    }
 20
 21
                if(st.empty()==0)
 22
 23
                    return false;
 24
                return true;
 25
           }
 26
       };
```

921. Minimum Add to Make Parentheses Valid

```
C++

    Autocomplete

       class Solution {
  1 •
  2
       public:
            int minAddToMakeValid(string s) {
  3 ▼
                stack<char> st;
  4
  5
                int ans=0;
  6
                for(auto it: s)
  7 🔻
                {if(it=='(')
                    st.push(it);
  8
                 else if(it==')')
  9
 10 •
                     if(st.empty())
 11
 12
                          ans++;
 13
                     else
 14
                          st.pop();
               }
 15
 16
 17
                ans=ans+st.size();
 18
                return ans;
 19
 20
            }
 21
       };
 22
```

1963. Minimum Number of Swaps to Make the String Balanced

```
i C++

    Autocomplete

        class Solution {
   1 *
         public:
   2
             int minSwaps(string s) {
   3 ▼
   4
                  stack<char> st;
   5
                  int count=0;
                  for(int i=0;i<s.size();i++)</pre>
   6
   7 🔻
                  {if(s[i]=='[')
                      st.push(s[i]);
   9
                   else
  10 •
                       if(!st.empty() )
  11
  12
                            st.pop();
  13
                       else
  14
                            count++;
  15
                   }
  16
  17
                  return (count+1)/2;
  18
  19
  20
         };
```

1111. Maximum Nesting Depth of Two Valid Parentheses Strings

```
i C++

    Autocomplete

   1 *
        class Solution {
        public:
   2
   3 ▼
             vector<int> maxDepthAfterSplit(string seq) {
                 vector<int> ans(seq.size());
   4
   5
                 int a=0;
                 for(int i=0;i<seq.size();i++)</pre>
   6
   7 🔻
                 {if(seq[i]=='(')
   8
                  ans[i]=a++;
   9
                  else
  10 •
                       ans[i]=--a;
  11
  12
  13
  14
  15
  16
                 return ans;
  17
             }
        };
  18
```

739. Daily Temperatures

```
i C++

    Autocomplete

  1 +
        class Solution {
  2
        public:
            vector<int> dailyTemperatures(vector<int>& t) {
  3 +
                stack<pair<int, int>> s;
  4
  5
                vector<int> ans(t.size());
  6
  7
                for(int i=t.size()-1;i>=0;i--)
                {if(s.size()==0)
  8 +
  9
                     ans[i]=0;
 10
                else if(s.size()!=0 && s.top().first>t[i])
 11
 12 *
 13
                     ans[i]=s.top().second-i;
 14
 15
                else if(s.size()!=0 && s.top().first<=t[i])
 16
 17 *
 18
                     while(s.size()!=0 && s.top().first<=t[i])
 19 +
                {
                     s.pop();
 20
 21
                 if(s.size()==0)
 22
 23
                     ans[i]=0;
 24
                else
                     ans[i]=s.top().second-i;
 25
 26
               s.push({t[i], i});
 27
 28
 29
                }
 30
 31
            return ans;}
 32
        };
```

2104. Sum of Subarray Ranges

```
i C++

    Autocomplete

        class Solution {
  1 •
  2
        public:
             long long subArrayRanges(vector<int>& nums) {
  3 ▼
  4
                 int maxi;
  5
                 int mini;
  6
             long long sum=0;
                 for(int i=0;i<nums.size();i++)</pre>
  7
  8 *
  9
                      maxi=nums[i];
 10
                      mini=nums[i];
                      for(int j=i+1; j<nums.size(); j++)</pre>
 11
 12 •
                          maxi=max(maxi, nums[j]);
 13
                          mini=min(mini, nums[j]);
 14
 15
                          sum+=maxi-mini;
 16
 17
 18
                 return sum;
 19
             }
 20
        };
```

1541. Minimum Insertions to Balance a Parentheses String

```
i C++
                    Autocomplete
        class Solution {
   1 •
        public:
   2
             int minInsertions(string s) {
   3 ₹
   4
                 stack<int> st;
   5
                 int count=0;
                 for(int i=0;i<s.size();i++)</pre>
   6
                 {if(s[i]=='(')
   7 🔻
                      {if(!st.empty() && st.top()==1)
   8 *
   9 🔻
                      {count++;
  10
                          st.pop();}
                          st.push(2);}
  11
  12
                      { if(st.empty())
  13 ▼
  14 🔻
                      {count++;
                          st.push(1); }
  15
  16
                           else if(st.top()==2)
  17 ▼
                          { st.top()--;
  18
  19
                          else if(st.top()==1)
  20
  21 ▼
  22
                              st.pop();
  23
                      }
  24
  25
                 while(!st.empty())
  26
  27 ▼
  28
                      count=count+st.top();
  29
                      st.pop();
  30
  31
                 return count;
  32
             }
        };
  33
```

853. Car Fleet

```
C++
                                                                           i {} 0

    Autocomplete

  1 *
        class Solution {
  2
        public:
  3 ₹
            int carFleet(int target, vector<int>& position, vector<int>& speed) {
            vector<pair<int, double>> car(position.size());
  4
  5
                for(int i=0;i<position.size();i++)</pre>
  6 *
                {
  7
                    car[i]={position[i], (double)(target-position[i])/speed[i]};
  8
  9
               sort(car.begin(), car.end());
                reverse(car.begin(), car.end());
 10
 11
                int ans=1;
 12
 13
                double tt=car[0].second;
 14
 15
                for(int i=1;i<car.size();i++)</pre>
 16
 17 v
                {
                    if(car[i].second>tt)
 18
 19 v
                    {
 20
                         ans++;
                         tt=car[i].second;
 21
 22
 23
 24
                return ans;
 25
            }
 26
        };
```

456. 132 Pattern

```
C++

    Autocomplete

  1 *
       class Solution {
       public:
  2
           bool find132pattern(vector<int>& nums) {
  3 .
  4
                 stack<int> s;
  5
                int prev = INT_MIN;
  6
  7 +
                for (auto it = nums.rbegin(); it != nums.rend(); it++) {
  8 +
                    while (!s.empty() && s.top() < *it) {
  9
                        if (prev > s.top())
 10
                            return true;
 11
                        prev = s.top();
 12
                        s.pop();
                    }
 13
 14
 15
                    s.push(*it);
 16
                }
 17
 18
                return !s.empty() && prev > s.top();
 19
            }
       };
 20
```

895. Maximum Frequency Stack

```
{}
C++

    Autocomplete

      class FreqStack {
 1 *
 2
      public:
 3
          int max frequency;
          unordered map<int, int> freq mp;
 4
          unordered_map<int, stack<int>> freq_stack_mp;
 5
          FreqStack() {
 6 +
 7
              max frequency=0;
8
 9
          void push(int x) {
10 ₹
               ++freq mp[x];
11
12
               if(max frequency<freq mp[x]) max frequency = freq mp[x];</pre>
              freq_stack_mp[freq_mp[x]].push(x);
13
          }
14
15
16 *
          int pop() {
17
               int curr top = freq stack mp[max frequency].top();
               freq_stack_mp[max_frequency].pop();
18
19
               --freq mp[curr top];
20 ₹
               if(freq_stack_mp[max_frequency].empty()) {
                   freq_stack_mp.erase(max_frequency);
21
22
                   --max frequency;
23
24
               return curr top;
25
26
      };
27
      /**
```