Stacks

The basic stack problems

#1) Parenthesis Balance

Identification.

① Whenever we have a inner loop j that is dependent on i; we can think of using stacks.

```
for(j=i+1; j<n; j++)
```

like

Code

```
bool isValid(string x){
    stack<char> stk;
    for(int i=0; i<x.length(); i++){
        if(x[i]=='(' || x[i]=='{' || x[i]=='[')
            stk.push(x[i])
        else if(x[i]==')' || x[i]=='}' || x[i]==']'){
            if(stk.empty() ||
              (stk.top()=='(' && (x[i]=='}' || x[i]==']'))
            || (stk.top()=='{' && (x[i]==')' || x[i]==']'))
            || (stk.top()='[' && (x[i]==')' || x[i]=='}')))
                return false;
            else
                stk.pop();
        }
    }
    return stk.empty();
}
```

// If stack is empty or the top of the stack doesn't match with the incoming parenthesis; the parenthesis is not balanced.

// After iterating the exp. if stack is empty then true;

#2) Infix to Postfix Conversion

If possibility other than mentioned in question, take the following

Rule:- 1) Operator of lower precedence can't be pushed over operator of higher precedence. Vice versa can happen

2) * If ')' encountered; pop all operators from stack and push to string until '(' comes.

Code

```
int precedence(char x){
    if (x == '^') return 3;
    else if (x == '*' || x == '/') return 2;   //L-R
    else if (x == '+' || x == '-') return 1;
    else return 0;
}
string infix_to_postfix(string s){
    Stack <char> stk;
    string res;
    for(int i=0; i< s.length(); i++){
        if((s[i]>='a' && s[i] <='z') || (s[i]>='A' && s[i]<='Z'))
            stk.push(s[i]);
        else if (s[i] =='(')
            stk.push(s[i]);

        else if (s[i] == ')'){
            while(!stk.empty() && stk.top != '('){
                char x = stk.top;
                stk.pop();
                res += x;

            }
            if (stk.top() == '(')
                stk.pop;

    }
```

```
else {                    // Operators '*', '+' etc.
    while (!stk.empty() && precedence(s[i]) <=
                           precedence(stk.top()))
    {
        char x = stk.top()
        stk.pop();
        res += x;
    }
    stk.push(s[i])        // Precedence higher so;
                          pushed.
}
} // End of for loop;
while (!stk.empty()) {
    char x = stk.top()
    stk.pop()
    res += x;
}
return res;
}
```

#3) Evaluate Post fix expression

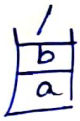
Code

```
int evaluatePostfix (String s).
    stack<int> stk;
    for (int i=0; i< s.length ; i++){.
        if (isdigit(s[i]))
            stk.push (int(s[i]) - 48)   // Ascii of 0 =
        else {      // For Operators
            int val1 = stk.top();  // Immediate top
            stk.pop();
            int val2 = stk.top();  // Second.
            stk.pop();
            switch (s[i]){
                case '+' ; stk.push(val2+val1) ; break;
                case '-' : stk.push (val2 - val1) ; break;
                case '*' : stk.push (val2 * val1) ; break;
                case '/' : stk.push (val2 / val1) ; break;
            }
        }
    }
    return stk.top();
}
```

a/b.
a+b

Postfix:
ab/ ; ab+.

Evaluation



So its always second popped (val 2) / (val 1).

For prefix evaluation:- last to first
For (i = s.length-1; i>=0; i--)

In operations.

val 1 + val 2;
val 1 - val 2;
val 1 * val 2;
val 1 / val 2;

4) Longest Valid Parenthesis (LeetCode). Approach 3.

Code

```
int longest(string st){
    stack<int> stk;
    int count = 0;
    stk.push(-1);
    for(int i=0; i< st.length(); i++){
        if(st[i]=='(')
        stk.push(i).
        else{
            stk.pop();
            if(stk.empty())
                stk.push(i);

            else
                count = max(count, i-stk.top());
        }
    }
    return count;
}.
```

Algorithm

① For every '(' push its index

② For every ')' encountered, we pop the topmost element and subtract current index from top element. and store the maximum.

If while popping stk becomes empty push its index to the stack.

#5) Simplify Directory path (Leetcode)

Input: /a/./b/../../c/.
Output: /c.

(.) represents current directory [Do nothing].
(..) go to parent directory ie, previous directory from now. [pop; so we are now in 2nd last directory]

Output the simplified path;
Add '/' to each.

Code (Python)

```
expr = input()
x = expr.split("/")

stk = []
for i in range(len(x)):
    if (x[i] >= "a" and x[i] <= "z") or (x[i] >= "A" and x[i] <= "Z"):
        stk.append(x[i])
    elif (x[i] == "."):
        continue
    elif (x[i] == ".."):
        stk.pop()
    else:
        continue

res = ""
for i in stk:
    res = res + "/" + i

print(res)
```

#6) Mass of Molecules

I/P = $(COOH)_2$ C, O, H are only symbols.
O/P = 90

Code

```
long long m_mass (string ex){
stack<int> stk;
for(int i=0; i<ex.length(); i++){
  if (ex[i]=='C') stk.push(12);
  else if (ex[i]=='O') stk.push(16);
  else if (ex[i]=='H') stk.push(1);
  else if (ex[i]=='C') stk.push('C');

  else if (ex[i]==')'){
     int int_mass = 0
     while(!stk.empty() && stk.top!='C'){
        int_mass += stk.top()
        stk.pop();
     }
     if (stk.top()='C')
        stk.pop();
     stk.push(int_mass);
  }
  else if (isdigit(ex[i])){
     for(int i=0; i<int(ex[i])-'0'-1; i++)
        stk.push(stk.top())
  }
  long long mass = 0;
  while(!stk..empty()){
     mass += stk.top()
     stk.pop();
  }
  return mass;
}
```

#87) Check for Redundant Brackets

(Algorithm 6 of 6)

Code

```
bool check_redundancy (string s){
    stack<char> stk;
    for(int i=0; i<s.length(); i++){
        if(s[i]==')'){
            char x=stk.top();
            stk.pop();
            bool flag=true;
            while(x!='('){
                if(x=='+'||x=='-'||x=='*'||x=='/')
                    flag=false;
                x=stk.top();
                stk.pop();
            }
            if(flag)
                return true;
        }else{
            stk.push(s[i]);
        }
    }
    return false
}
```

Without closing braces there has to be an operator for it to be non-redundant.

8) Given no. of opening and closing brackets '{' and '}' find out minimum number of reversals to make expression balanced;

— x —

Algorithm

balanced

1) Remove the valid parenthesis part from the expression

2) Now, we have 0 or more closing brackets (grp together) followed by 0 or more opening brackets (grp together).

} m times { n times.

Result = ceil(m/2) + ceil(n/2).

```
int numberOfReversals (string expr) {
    int len = expr.length();
    if (len % 2 != 0) return -1;   // Odd length can't be balanced
    Stack<char> stk;

    for (int i = 0; i < len; i++) {
        if (expr[i] == '}' && !stk.empty()) {
            if (stk.top == '{')
                stk.pop() ;
            else
                stk.push ('}');
        } else
            stk.push(expr[i]);
    }
    int count_open = 0;
    while (!stk.empty() && stk.top == '{') {
        stk.pop();
        count_open++
    }
    int count_close = stk.size()   // remaining close braces.
    return (ceil(count_open/2.00) + ceil(count_close/2.00));
}
```

**

#9) Nearest Greater to the Right.

I/P : 1 3 2 4

O/P : 3 4 4 -1

Code

```
vector<long long> nextLargerElement(vector<long long> arr, int
    vector<long long> res;
    stack<long long> stk;
    for(int i=n-1; i>=0; i--){   // Iterate from end of array. Finally reverse the result.
        if(stk.empty())
            res.push_back(-1);
        else if(!stk.empty() && stk.top() > arr[i])
            res.push_back(stk.top())
        else if(!stk.empty() && stk.top() <= arr[i]){
            while(!stk.empty() && stk.top() <= arr[i])
                stk.pop();          // Remove all elements that are smaller.
            if(stk.empty())
                res.push_back(-1);
            else
                res.push_back(stk.top());
        }
        stk.push(arr[i])
    }
    reverse(res.begin(), res.end());
    return res;
}
```

// For Nearest Greater to the left. } Only changes.

① for(int i=0; i<n; i++)

② Reversing the o/P not needed

10) Nearest Smaller on the Left

I/P - 1 5 0 3 4 5

O/P - -1 1 -1 0 3 4

Code

```
vector<int> leftSmaller(int n, int * arr){
    vector<int> res;
    stack<int> stk;
    for(int i=0; i<n; i++){
        if (stk.empty())
            res.push_back(-1);

        else if (!stk.empty() && stk.top() < arr[i])
            res.push_back(stk.top());

        else if (!stk.empty() && stk.top() >= arr[i]){
            while(!stk.empty() && stk.top() >= arr[i])
                stk.pop();

            if(stk.empty())
                res.push_back(-1);

            else
                res.pus_back(stk.top())

        }
        stk.push(arr[i])
    }

    return res
}
```

// For nearest Smaller to the right.

① for(int i=n-1; i>=0; i--)

② Reversing the O/P - ✓

} Only changes.

#11) Stock Span Problem

The span S_i of the stock's price on a given day i is defined as the max. number of consecutive days just before the given day for which the price of the stock on the current day is less than equal to price on day i.

```
          0   1   2   3   4   5   6
I/P = { 100, 80, 60, 70, 60, 75, 85 }
O/P =   1   1   1   2   1   4   6.
```

So here, we need to need to O/P no. of days

We need to find the nearest left greater and store its index with it.

O/P = index(i) - index of nearest greater to left

Code

```
vector<int> span(int* price, int n){
    vector<int> res;
    stack<pair<int, int>> stk;   // for storing left nearest Pnt & its index.
    for(int i=0; i<n; i++){
        if (stk.empty())
            res.push_back(-1);
        else if (!stk.empty() && stk.top().first > price[i])
            res.push_back(stk.top().second);
        else if (!stk.empty() && stk.top().first <= price[i])
            while(" ")
                stk.pop()
            if (stk.empty())
                res.push_back(-1)
            else
                res.push_back(stk.top().second).
    }
    stk.push({price[i], i})
}
for(int i=0; i<n; i++)
    res[i] = i - res[i]   // Stores the span
return res;
```

Null code to store index.

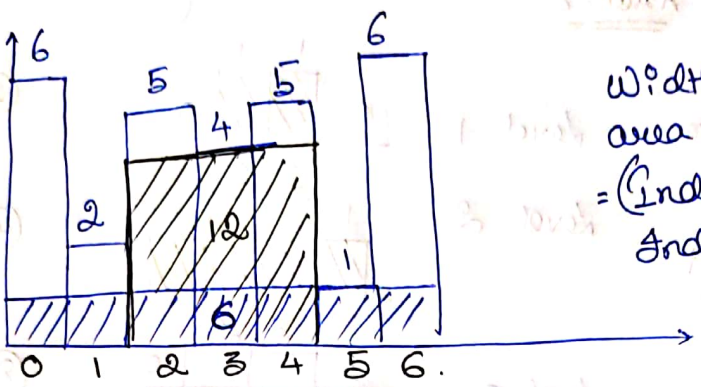
***

12) Maximum area Histogram

I/P = {6, 2, 5, 4, 5, 1, 6}

O/P = 12.

Width of each bar = 1 unit.



Width of rectangular area = (Index of NSL − Index of NSR − 1).

Area = (Index of NSL − Index of NSR − 1) * height[i]

Nearest left Smaller — Nearest Right Smaller

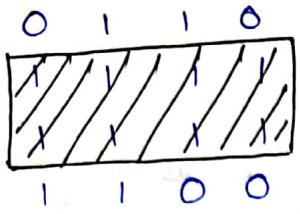
Code

(MAH)

```
long long maxArea(long long* height, int n){
    vector<int> left; right;
    left = NSL(height, n);     // same code no element (-1)
    right = NSR(height, n);    // No element push pseudoindex (n).

    vector<long long> width;          Do not push (-1)
    for(int i=0; i<n; i++){           Push (n)
        long long w = right[i] - left[i] - 1
        width.push_back(w);
    }

    vector<long long> area;
    for(int i=0; i<n; i++){
        long long ar = width[i] * height[i];
        area.push_back(ar)
    }

    return *max_element(area.begin(), area.end());
}
```

****

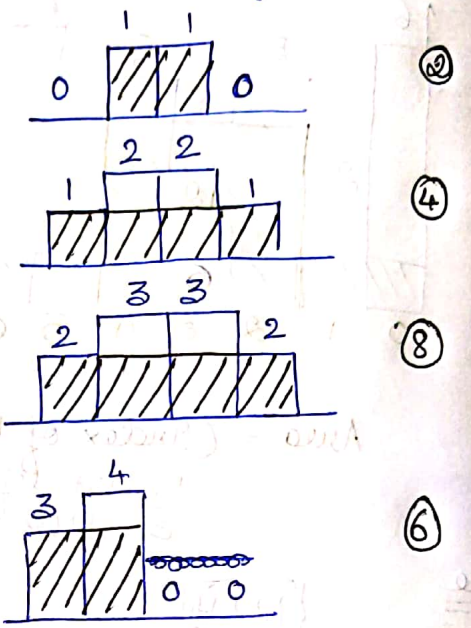
13) Maximum Area of Rectangle in a Binary Matrix



Area = 8.

n x m

We convert each row into a histogram.



O/P = 8

As we keep on getting (1) add to hgt. of histogram.

As soon as 0 is encountered; height of histogram is made 0. (Cannot float)

Max. area Histogram.

For each level call MAH function

Code

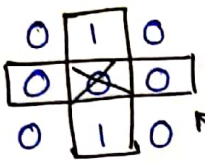
```
vector<int> areaMax(int** mat, int n, int m){
    vector<int> v;
    for(int j=0; j<m; j++)          // first row.
        v.push_back(mat[0][j])
    int max = MAH(v);               // MAH written last page.
    for(int i=1; i<n; i++){
        for(int j=0; j<m; j++){
            if(mat[i][j] == 1)
                v[j] = v[j] + mat[i][j];
            else if(mat[i][j] == 0)
                v[j] = 0;
        }
        max = max(max, MAH(v))      // finding area storing max for each level
    } return max;
```

***

14) Celebrity Problem

Defined as; a person who does not know any one but everyone knows him.

Algorithm:- Method of elimination followed



1) Push all IDs in the stack.
2) Pop top 2 elements till (stack.size() >= 2) and check each.
3) Find potential.
Run a final check over the pot's row and col.

Code

```
int celebrity(vector<vector<int>> & M, int n){
    stack<int> stk;
    for(int i=0; i<n; i++)
        stk.push(i)

    while(stk.size() >= 2){
        int A = stk.top(); stk.pop();
        int B = stk.top(); stk.pop();

        if(M[A][B] == 1)
            //If A knows B; A is not a celebrity (B can be)
            stk.push(B);
        else
            //If A doesn't B; B is not a celebrity
            stk.push(A);
    }

    int pot = stk.top(); // Only the potential candidate is left in the stack now.
    stk.pop();

    for(int i=0; i<n; i++) // Step 3 of algo
        if(i != pot){ // If i == pot will always be 0
            if(M[i][pot] == 0 || M[pot][i] == 1)
                //If i does not know potential (pot not celeb)
                //If pot knows i (pot not celeb)
                return -1; //No celeb. found.
        }
    }
    return pot;
}
```