DOMAIN WINTER CAMP WORKSHEET

DAY-4 (23/12/2024)

Student Name :- Pratham Kapoor **University ID :- 22BCS10732**

Branch :- B.E. (C.S.E.) **Section/Group :- FL_ 603-A**

Problem-1 :- Design a stack that supports push, pop, top, and retrieving the minimum element in constant time. Implement MinStack class :- MinStack() initializes the stack object, void push(int val) pushes the element val onto the stack, void pop() removes the element on the top of the stack, int top() gets the top element of the stack, int getMin() retrieves the minimum element in the stack. You must implement a solution with O(1) time complexity for each function. **(Very Easy)**

Source Code

```
#include<iostream>
#include<stack>
#include<vector>
using namespace std;

class MinStack {
public:
    stack<int>stk;
    stack<int>minStk;

    MinStack() {}

    void push(int val) {
        stk.push(val);
        if(minStk.empty()||val<=minStk.top()) {
```

```
            minStk.push(val);
        }
    }

    void pop() {
        if(stk.top()==minStk.top()) {
            minStk.pop();
        }
        stk.pop();
    }

    int top() {
        return stk.top();
    }

    int getMin() {
        return minStk.top();
    }
};

int main() {
    MinStack minStack;
    vector<pair<int, int>> operations;
    int numOperations, choice, val;

    cout<<"Enter the Number of Operations :- ";
    cin>>numOperations;

    for(int i=0; i<numOperations; i++) {
        cout<<"\n============================\n";
        cout<<"        MAIN MENU\n";
        cout<<"============================\n";
```

```
    cout<<"1. Push\n";
    cout<<"2. Pop\n";
    cout<<"3. Top\n";
    cout<<"4. GetMin\n";
    cout<<"============================\n";
    cout<<"Enter the Choice :- ";
    cin>>choice;
    cout<<"============================\n";
    if(choice==1) {
        cout<<"Enter the Value to Push :- ";
        cin>>val;
        operations.push_back({choice, val});
    } else if(choice>=2 && choice<=4) {
        operations.push_back({choice, 0});
    } else {
        cout<<"Invalid Choice. Try Again !\n";
        i--;
    }
}

cout<<"\nThe MinStack Operations Result is [";
cout<<"Null,";
for(auto op : operations)
     {
    if(op.first==1) {
        minStack.push(op.second);
        cout<<"Null,";
    } else if(op.first==2) {
        minStack.pop();
        cout<<"Null,";
    } else if(op.first==3) {
        cout<<minStack.top()<<",";
```

```
        } else if(op.first==4) {
            cout<<minStack.getMin()<<",";
        }
    }
    cout<<"]"<<endl;
    return 0;
}
```

Output

```
Enter the Number of Operations :- 7

============================
         MAIN MENU
============================
1. Push
2. Pop
3. Top
4. GetMin
============================
Enter the Choice :- 1
============================
Enter the Value to Push :- -2

============================
         MAIN MENU
============================
1. Push
2. Pop
3. Top
4. GetMin
============================
Enter the Choice :- 1
============================
Enter the Value to Push :- 0

============================
         MAIN MENU
============================
1. Push
2. Pop
3. Top
4. GetMin
============================
Enter the Choice :- 1
============================
Enter the Value to Push :- -3
```

```
============================
         MAIN MENU
============================
1. Push
2. Pop
3. Top
4. GetMin
============================
Enter the Choice :- 4
============================

============================
         MAIN MENU
============================
1. Push
2. Pop
3. Top
4. GetMin
============================
Enter the Choice :- 2
============================

============================
         MAIN MENU
============================
1. Push
2. Pop
3. Top
4. GetMin
============================
Enter the Choice :- 3
============================
```

```
============================
         MAIN MENU
============================
1. Push
2. Pop
3. Top
4. GetMin
============================
Enter the Choice :- 4
============================

The MinStack Operations Result is [Null,Null,Null,Null,-3,Null,0,-2,]
```

Problem-2 **:-** The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwiches. The number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step :- If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue. Otherwise, they will leave it and go to the queue's end. This continues until none of the queue students want to take the top sandwich and are thus unable to eat. You are given two integer arrays students and sandwiches where sandwiches[i] is the type of the ith sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the jth student in the initial queue (j = 0 is the front of the queue). Return the number of students that are unable to eat. **(Easy)**

Source Code

```
#include<iostream>
#include<queue>
using namespace std;
int main()
{
   int n,Count=0;
   cout<<"Enter the Number of Students :- ";
   cin>>n;
   queue<int>Students;
   queue<int>Sandwiches;
   cout<<"Enter the Preferences of Students :- ";
   for(int i=0;i<n;i++) {
      int s;
      cin>>s;
      Students.push(s);
   }
```

```
    cout<<"Enter the Types of Sandwiches :- ";
    for(int i=0;i<n;i++) {
        int sw;
        cin>>sw;
        Sandwiches.push(sw);
    }
    while(!Sandwiches.empty()) {
        if(Students.front()==Sandwiches.front()) {
            Students.pop();
            Sandwiches.pop();
            Count=0;
        } else {
            int temp=Students.front();
            Students.pop();
            Students.push(temp);
            Count++;
            if(Count==Students.size()) {
                break;
            }
        }
    }
    cout<<"\nThe Number of Students Unable to Eat are "<<Students.size();
    return 0;
}
```

Output

```
Enter the Number of Students :- 4
Enter the Preferences of Students :- 1 1 0 0
Enter the Types of Sandwiches :- 0 1 0 1

The Number of Students Unable to Eat are 0
```

Problem-3 **:-** Given a circular integer array nums (i.e., the next element of nums[nums.length - 1] is nums[0]), return the next greater number for every element in nums. The next greater number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return -1 for this number. **(Medium)**

Source Code

```
#include<iostream>
#include<vector>
#include<stack>
using namespace std;

int main() {
   int n;
   cout<<"Enter the Number of Elements :- ";
   cin>>n;
   vector<int>Nums(n);
   vector<int>Result(n,-1);
   stack<int>Stack;
   cout<<"Enter the Elements of the Array :- ";
   for(int i=0;i<n;i++) {
      cin>>Nums[i];
   }
   for(int i=0;i<2*n;i++) {
      while(!Stack.empty()&&Nums[Stack.top()]<Nums[i%n]) {
         Result[Stack.top()]=Nums[i%n];
         Stack.pop();
      }
      if(i<n) {
         Stack.push(i);
```

```
        }
    }
    cout<<"\nThe Next Greater Numbers are ";
    for(int i=0;i<n;i++) {
        cout<<Result[i]<<" ";
    }
    return 0;
}
```

Output

```
Enter the Number of Elements :- 3
Enter the Elements of the Array :- 1 2 1

The Next Greater Numbers are 2 -1 2
```

Problem-4 :- You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position. Return the max sliding window. **(Hard)**

Source Code

```
#include<iostream>
#include<vector>
#include<deque>
using namespace std;

int main() {
    int n, k;
    cout << "Enter the Number of Elements :- ";
    cin >> n;
```

```
cout << "Enter the Size of Sliding Window :- ";
cin >> k;
vector<int> Nums(n);
cout << "Enter the Elements of the Array :- ";
for(int i = 0; i < n; i++) {
   cin >> Nums[i];
}
vector<int> Result;
deque<int> Deque;
for(int i = 0; i < n; i++) {
   if(!Deque.empty() && Deque.front() == i - k) {
      Deque.pop_front();
   }
   while(!Deque.empty() && Nums[Deque.back()] <= Nums[i]) {
      Deque.pop_back();
   }
   Deque.push_back(i);
   if(i >= k - 1) {
      Result.push_back(Nums[Deque.front()]); }
}
cout << "\nThe Max Sliding Window is ";
for(int i = 0; i < Result.size(); i++) {
   cout << Result[i] << " ";
}
return 0; }
```

Output

```
Enter the Number of Elements :- 8
Enter the Size of Sliding Window :- 3
Enter the Elements of the Array :- 1 3 -1 -3 5 3 6 7

The Max Sliding Window is 3 3 5 5 6 7
```

Problem-5 :- There are a number of plants in a garden. Each of the plants has been treated with some amount of pesticide. After each day, if any plant has more pesticide than the plant on its left, being weaker than the left one, it dies. You are given the initial values of the pesticide in each of the plants. Determine the number of days after which no plant dies, i.e. the time after which there is no plant with more pesticide content than the plant to its left. **(Very Hard)**

Source Code

```
#include<iostream>
#include<vector>
#include<stack>
using namespace std;

int poisonousPlants(vector<int>& p) {
    int n = p.size();
    vector<int> days(n, 0);
    stack<int> stk;
    int maxDays = 0;

    for (int i = 0; i < n; i++) {
        while (!stk.empty() && p[i] <= p[stk.top()]) {
            stk.pop();
        }

        if (!stk.empty()) {
            days[i] = days[stk.top()] + 1;
        }

        maxDays = max(maxDays, days[i]);
        stk.push(i);
    }
```

```
    return maxDays;
}

int main() {
    int n;
    cout << "Enter the Number of Plants :- ";
    cin >> n;
    vector<int> p(n);
    cout << "Enter the Pesticide Levels of the Plants :- ";
    for (int i = 0; i < n; i++) {
        cin >> p[i];
    }

    cout << "\nThe Number of Days Until no Plant Dies are " << poisonousPlants(p) << endl;
    return 0;
}
```

Output

```
Enter the Number of Plants :- 7
Enter the Pesticide Levels of the Plants :- 6 5 8 4 7 10 9

The Number of Days Until no Plant Dies are 2
```