GeeksMan
Data Structure
Lesson 3
Introduction to Vector

Trilok Kaushik

Founder of GeeksMan

Chirag Soni

Nupur Pahuja

Jessica Mishra

Team Coordinators

Vectors in C++

Vectors are sequence containers representing arrays that can change in size.

Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

Internally, vectors use a dynamically allocated array to store their elements. This array may need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it. This is a relatively expensive task in terms of processing time, and thus, vectors do not reallocate each time an element is added to the container.

Instead, vector containers may allocate some extra storage to accommodate for possible growth, and thus the container may have an actual capacity greater than the storage strictly needed to contain its elements (i.e., its size).

Therefore, compared to arrays, vectors consume more memory in exchange for the ability to manage storage and grow dynamically in an efficient way.

How to declare vectors?

First we have to include the vector header file
#include <vector>
Then for declaring,
vector<data_type> vector_name;
To access any specific element from a vector we can write(just like array)
vector_name[index]

Functions that we can use:

1. size() – Returns the number of elements in the vector.
2. max_size() – Returns the maximum number of elements that the vector can hold.
3. capacity() – Returns the size of the storage space currently allocated to the vector expressed as number of elements.
4. resize(n) – Resizes the container so that it contains ‘n’ elements.
5. empty() – Returns whether the container is empty.
6. reserve() – Requests that the vector capacity be at least enough to contain n elements.

For adding an element in vector:

1. assign() – It assigns new value to the vector elements by replacing old ones
2. push_back() – It push the elements into a vector from the back
3. pop_back() – It is used to pop or remove elements from a vector from the back.
4. insert() – It inserts new elements before the element at the specified position

We can also swap the contents of one vector with another vector of the same type(sizes may differ), by using the swap() function.

By using erase() function we can remove a particular element by specifying the position.
And by the clear() function, we can remove all the elements from the vector.

Nearest Greatest To Right(NGR)

For every integer in an array , you have to find the nearest largest element to the right of the array .
Let's take an example ,

1 3 0 0 1 2 4 → 3 4 1 1 2 4 -1

In this array , for 1 , nearest largest for 1 is 3 , for 3 is 4 , for 0 is 1 and so on ,
If in any case , you come across , any element for which there is no greater element on its right , then print -1.

ALGORITHM:

1. We will traverse this array from last to first element.
2. Create a stack and a vector .
3. If the stack is empty , then push -1 , in the vector and push a[i] in the stack.
4. Else if a[i] < st.top() , then push st..top() in vector.
5. If a[i] >= st.top() , while(a[i] >= st.top()) then st.top() . if(st.size() == 0) , push -1 in vector , else push st.top() in vector .
6. Print vector.

Similarly , if we traverse the array from left to right , we get Nearest Greatest to left.

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n,i,l;
    cin>>n;
    int A[n];
    for(i=0;i<n;i++)
      {
          cin>>A[i];
      }
      vector<int> v;
      stack<int> st;
      for(i=n-1;i>=0;i--)
```

```
16.  {
17.      if(st.size()==0)
18.      {
19.          v.push_back(-1);
20.      }
21.      else if(A[i]<st.top())
22.      {
23.           v.push_back(st.top());
24.      }
25.      else
26.      {
27.          while(st.size()>0&&A[i]>=st.top())
28.              st.pop();
29.          if(st.size()==0 )
30.              v.push_back(-1);
31.          else
32.              v.push_back(st.top());
33.      }
34.      st.push(A[i]);
35.  }
36.      l=v.size();
37.      for(i=n-1;i>=0;i--)
38.      {
39.          cout<<v[i]<<" ";
40.      }
41.  }
```

Link for the code : https://sapphireengine.com/@/4es1b5

Nearest Smallest To Left(NSL)

For every integer in an array , you have to find the nearest smallest element to the left of the array .
Let's take an example ,

1 3 0 0 1 2 4 → -1 1 -1 -1 0 1 2 -1

\If in any case , you come across any element for which there is no greater element on its right , then print -1.

ALGORITHM:

1. We will traverse this array from first to last element.
2. Create a stack and a vector .
3. If the stack is empty , then push -1 , in the vector and push a[i] in the stack.
4. Else if a[i] > st.top() , then push st..top() in vector.
5. If a[i] <= st.top() , while(a[i] <= st.top()) then st.top() . if(st.size() == 0) , push -1 in vector , else push st.top() in vector .
6. Print vector.

Similarly , if we traverse the array from left to right , we get Nearest Smallest to right.

```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int t;
    cin >> t;
    while(t--)
    {
       int n,i,l;
       cin>>n;
       int A[n];
       for(i=0;i<n;i++)
       {
           cin>>A[i];
       }
```

```
    vector<int> v;
    stack<int> st;
    for(i=0;i<n;i++)
{
    if(st.size()==0)
    {
        v.push_back(-1);
    }
    else if(A[i]>st.top())
    {
         v.push_back(st.top());
    }
    else
    {
        while(st.size()>0&&A[i]<=st.top())
            st.pop();
        if(st.size()==0 )
            v.push_back(-1);
        else
            v.push_back(st.top());
    }
    st.push(A[i]);
}
    l=v.size();
    for(i=0;i<l;i++)
    {
        cout<<v[i]<<" ";
    }
        cout << endl;
    }
}
```

Link for the code : https://sapphireengine.com/@/ibmdrt

MAXIMUM AREA OF HISTOGRAM

Find the largest rectangular area possible in a given histogram where the largest rectangle can be made of a number of contiguous bars. For simplicity, assume that all bars have the same width and the width is 1 unit.

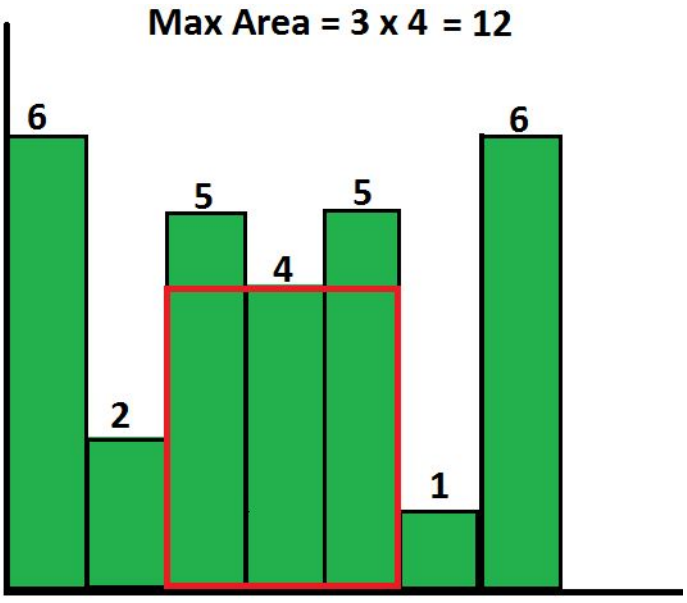
Input:

The first line contains an integer 'T' denoting the total number of test cases. T test-cases follow. In each test case, the first line contains an integer 'N' denoting the size of the array. The second line contains N space-separated integers A_1, A_2, ..., A_N denoting the elements of the array. The elements of the array represent the height of the bars.

Output:

For each test-case, in a separate line, the maximum rectangular area possible from the contiguous bars.

Question link :
https://practice.geeksforgeeks.org/problems/maximum-rectangular-area-in-a-histogram/0



ALGORITHM :

1. Create a stack and push all the elements into the stack
2. For each element , find nearest smallest to left , and nearest smallest to right .
3. Find the area considering the length of the rectangle as ngr - ngl -1 and width as the element(height of that bar).
4. Find the max area.

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

#define siz 100000
vector<int> ngl(int a[siz] , int n)
{
   int i;
   vector<int> v;
     stack<int> st;
     for(i=0;i<n;i++)
    {
         if(st.size()==0)
         {
             v.push_back(-1);
         }
         else if(a[i]>a[st.top()])
         {
             v.push_back(st.top());
         }
         else
         {
             while(st.size()>0&&a[i]<=a[st.top()])
                 st.pop();
             if(st.size()==0 )
```

```
                v.push_back(-1);
            else
                v.push_back(st.top());
        }
        st.push(i);
    }
    return v;
}
vector<int>  ngr(int a[siz] , int n)
{
    int i;
    vector<int> vr;
    stack<int> st;
    for(i=n-1;i>=0;i--)
    {
        if(st.size()==0)
        {
            vr.push_back(n);
        }
        else if(a[i]>a[st.top()])
        {
            vr.push_back(st.top());
        }
        else
        {
            while(st.size()>0 &&
 a[i]<=a[st.top()])
                st.pop();
            if(st.size()==0 )
                vr.push_back(n);
            else
                vr.push_back(st.top());
```

```
57.         }
58.         st.push(i);
59.    }
60.     vector<int>v3;
61.     for(i=n-1;i>=0;i--)
62.     {
63.         v3.push_back(vr[i]);
64.     }
65.     return v3;
66. }
67. int myfun(int a[siz] , int n)
68. {
69.     int i;
70.     vector<int>v1;
71.     vector<int>v2;
72.    v1=ngl(a,n);
73.    v2= ngr(a,n);
74.     int max_ar = 0,max;
75.     int c,m=0;
76.     for(i=0;i<n;i++)
77.     {
78.         int p= v2[i] - v1[i] - 1;
79.         p = a[i] * p;
80.         if(p > max_ar)
81.             max_ar = p;
82.     }
83.     return max_ar;
84. }
85. int main()
86. {
87.    int t;
88.     cin >> t;
```

```
    while(t--)
    {
        int n,i,a[siz];
        cin >> n;
        for(i = 0 ; i< n;i++)
        {
            cin >> a[i];
        }
        cout << myfun(a,n) << endl;
    }
   return 0;
}
```

Link for the code : https://sapphireengine.com/@/72fgtq

Question for practice=

1. https://practice.geeksforgeeks.org/problems/stock-span-problem/0
2. https://practice.geeksforgeeks.org/problems/smallest-number-on-left/0
3. https://practice.geeksforgeeks.org/problems/save-gotham/0
4. https://practice.geeksforgeeks.org/problems/next-larger-element/0
5. https://practice.geeksforgeeks.org/problems/delete-array-elements-which-are-smaller-than-next-or-become-smaller/0