Built-in Data Structures I

By: Andrew Qi Tang

Introduction

We normally put #include <bits/stdc++.h> at the top of our code in c++.

This includes “everything” in the c++ standard library.

This includes many data structures.

Part I will include some of the linear data structures.

What is a Data Structure?

A way to organize data.

Allows us to we can access, modify more efficiently.

Example of data structures: arrays.

Linear Data Structures

Linear Data Structures looks like the data is organized in a line.

Examples: Array, Queue, Deque, Stack, Vector

Array

Most fundamental data structure. (Not counting variable)

Most used as well. (Not counting variable)

Each array “index” has an associated value to it.

“The best data structure.”

To call an array: data_type array_name[array_size];

Operations With Array

Access one index - O(1) time

array_name[index]

Change one index - O(1) time

array_name[index] = new value

Vectors

Problem with array is that the size cannot change.

A vector can change the shape.

Vector Initialization

For an empty vector.

vector<data_type> vector_name;

For a vector with certain elements.

vector<data_type> vector_name = {first_element, second_element, … last_element};

For a vector with a certain element repeated multiple times.

vector<data_type> vector_name (number_of_elements, element_value);

Operations With Vectors

Access one index - O(1) time

Change one index - O(1) time

Add an element at the end - O(1) time

vector_name.push_back(element);

Remove the element at the end - O(1) time

vector_name.pop_back();

vector_name.back(); A shorter coding method to get the last element. O(1) time.

More Vector Operations

Removes one element in the vector with the value O(N) time.

vector_name.erase(element_value);

Removes the pointer from the vector. O(1) time.

vector_name.erase(pointer);

To change the size of the vector. O(N) time.

vector_name.resize(new_size);

More Vector Operators

To make the vector empty. O(1) or O(N) time

vector_name.clear();

To swap two vectors. O(1) time.

swap(first_vector_name, second_vector_name);

To get the size of the vector. O(1) time

vector_name.size();

Pointers and Vectors

Pointers are something exclusive to C++

To get the first pointer of a vector: v.begin()

To get the last pointer of a vector: v.end()

For Deque, Vector And Array. To get index from pointer.

pointer_of_current - first_pointer.

Pointer of the i-th element.

pointer_of_current+i.

Deque

A dynamic array that can access the front and the end.

Slower than vector and array (Constant)

Access the indices and change value by index in O(1). Syntax same as before.

Deque

Front Operations

d.push_front(element)

- Pushes element to the front.

d.front();

- Returns the first element

d.pop_front();

- Deletes the first element

Back Operations

d.push_back(element)

- Pushes element to the back.

d.back();

- Returns the last element

d.pop_back();

- Deletes the last element

More Complicated Operations For All 3

Sorting in O(N log N) time.

sort(begin_pointer, end_pointer+1). Sorts the subarray in increasing order.

Reversing an array in O(N) time.

reverse(begin_pointer, end_pointer+1). Reverses the subarray [L, R].

Returns the pointer of the first occurence of an element, O(N) time.

find(begin_pointer, end_pointer+1, element_value);

More Complicated Operations For All 3

To replace every element in a range with a new value. O(N) time

fill(begin_pointer, end_pointer+1, value);

Binary Searching For All 3

Works only if elements are non-decreasing.

lower_bound(begin_pointer, end_pointer+1, value); Pointer of first element >= value

upper_bound(begin_pointer, end_pointer+1, value); Pointer of first element > value

binary_search(begin_pointer, end_pointer+1, value); True if it exists, otherwise false.

Comparator

Needed to sort a set of elements.

It is to determine which element is smaller or larger.

Can be overwritten.

Comparator Template

```
//comparing to see if f goes after or before s
bool comparator_name(data_type f, data_type s){
 return is_f_smaller_than_s(f, s); //returns true if f is smaller than s
}

int main(){
 sort(begin_pointer, end_pointer+1, comparator_name); //sorting in that interval with that comparator
}
```

Example Comparator

We want to make a function to prioritize the larger last digit.

If the last digit are equal, we would then want to sort by smallest value first.

Example Comparator Code

```
#include <bits/stdc++.h>

using namespace std;

bool mycomparator(int a, int b){
   if(a%10 == b%10){
       return a < b;
   }
   else{
       return a%10 > b%10;
   }
}

vector<int> v = {10, 11, 2, 21};

int main(){
   sort(v.begin(), v.end(), mycomparator);
}
```

Warning With Comparator

Return true if the first element is smaller than the second element.

Do not return true if they’re equal, return false instead.

To read more:

https://codeforces.com/blog/entry/70237

https://codeforces.com/blog/entry/70236

https://stackoverflow.com/questions/45929474/why-must-stdsort-compare-function-return-false-when-arguments-are-equal

Homework

https://dmoj.ca/problem/ccc14s3

https://dmoj.ca/problem/mwc15c4p3

https://dmoj.ca/problem/pwindsor18p7

https://dmoj.ca/problem/mockccc15j5

https://dmoj.ca/problem/dmopc16c3p2

https://dmoj.ca/problem/year2017p1

https://dmoj.ca/problem/vmss7wc15c2p3