Competitive Programming Notebook

Daniel Santiago Lopez Daza
dlopezd@unal.edu.co

Tomás David Rodriguez Agudelo
trodrigueza@unal.edu.co

Eduard Joel Ostos Castro
eostos@unal.edu.co

Last updated: July 13, 2024

Contents

1 Snippets

2 Template

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

#define int long long
#define pb push_back
#define pii pair<int, int>
#define vi vector<int>
#define vb vector<bool>
#define vpi vector<pair<int, int>>
#define vvi vector<vector<int>>
#define ff first
#define ss second
#define all(container) container.begin(), container.end()

void solve()
{
}

int32_t main()
{
#ifndef ONLINE_JUDGE
  freopen("input.txt", "r", stdin);
  freopen("out.txt", "w", stdout);
  freopen("error.txt", "w", stderr);
#endif
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  int t; cin >> t;
  while(t--) solve();
}
```

3 Macros

1. **Debugging:**

   (a) `assert(x)`: If x is false, the program will terminate. ej:

   ```cpp
   assert(1 == 2);
   // Will terminate the program

   ```

2. **Container related Macros:**

   (a) **all(x)**

   ```cpp
   #define all(x) x.begin(), x.end()

   ```

   (b) **pb**

   ```cpp
   #define pb push_back

   ```

3. **Types:**

   (a) **int**

   ```cpp
   #define int long long

   ```

   (b) **vi, vb and vpi**

   ```cpp
   #define vi vector<int>
   #define vb vector<bool>
   #define vpi vector<pair<int, int>>

   ```

   (c) **pi**

   ```cpp
   #define pi pair<int, int>

   ```

   (d) **ff and ss**

   ```cpp
   #define ff first
   #define ss second

   ```

4 Container Classes and Initialization

1. **Vector:**

   (a) Class:

   ```cpp
   template < class T,
   class Alloc = allocator<T>> class vector;

   ```

   (b) Initialization:

   i. `vector<int> v`: Declares a vector of integers.

   ii. `vector<int> v(n)`: Declares a vector of integers of size n.

   iii. `vector<int> v(n, x)`: Declares a vector of integers of size n, with all elements initialized to x.

   iv. `vector<int> v = {1, 2, 3, 4}`: Declares a vector of integers with the elements 1, 2, 3 and 4.

   v. `vector<int> v {1, 2, 3, 4}`: Declares a vector of integers with the elements 1, 2, 3 and 4.

2. **Set:**

   (a) Class:

```
template < class T,
class Compare = less<T>, \\ View Interesting Classes Chapter
  for more information
class Alloc = allocator<T>> class set;
```

(b) Initialization:

i. `set<int> s`: Declares a set of integers.

ii. `set<int> s {1, 2, 3, 4}`: Declares a set of integers with the elements 1, 2, 3 and 4.

iii. `set<int> s = {1, 2, 3, 4}`: Declares a set of integers with the elements 1, 2, 3 and 4.

(c) **Map:**

i. Class:

```
    template <class Key, class T, class Compare = less<Key
>, class Alloc = allocator<pair<const Key, T>>> class map
;
```

ii. Initialization:

A. `map<int, int> m`: Declares a map with integer keys and values.

B. `map<int, int> m {{1, 2}, {3, 4}}`: Declares a map with the key-value pairs {1, 2} and {3, 4}.

C. `map<int, int> m = {{1, 2}, {3, 4}}`: Declares a map with the key-value pairs {1, 2} and {3, 4}.

(d) **Multiset:**

i. Class:

```
    template <class T, class Compare = less<T>, class Alloc
 = allocator<T>> class multiset;
```

ii. Initialization:

A. `multiset<int> ms`: Declares a multiset of integers.

B. `multiset<int> ms {1, 2, 2, 3, 4}`: Declares a multiset of integers with the elements 1, 2, 2, 3, and 4.

C. `multiset<int> ms = {1, 2, 2, 3, 4}`: Declares a multiset of integers with the elements 1, 2, 2, 3, and 4.

(e) **Unordered Set:**

i. Class:

```
    template <class Key, class Hash = hash<Key>, class Pred
 = equal_to<Key>, class Alloc = allocator<Key>> class
unordered_set;
```

ii. Initialization:

A. `unordered_set<int> us`: Declares an unordered set of integers.

B. `unordered_set<int> us {1, 2, 3, 4}`: Declares an unordered set of integers with the elements 1, 2, 3, and 4.

C. `unordered_set<int> us = {1, 2, 3, 4}`: Declares an unordered set of integers with the elements 1, 2, 3, and 4.

(f) **Unordered Map:**

i. Class:

```
    template <class Key, class T, class Hash = hash<Key>,
class Pred = equal_to<Key>, class Alloc = allocator<pair<
const Key, T>>> class unordered_map;
```

ii. Initialization:

A. `unordered_map<int, int> um`: Declares an unordered map with integer keys and values.

B. `unordered_map<int, int> um {{1, 2}, {3, 4}}`: Declares an unordered map with the key-value pairs {1, 2} and {3, 4}.

C. `unordered_map<int, int> um = {{1, 2}, {3, 4}}`: Declares an unordered map with the key-value pairs {1, 2} and {3, 4}.

(g) **Order Statistics Tree (using `tree` from `<ext/pb_ds/assoc_container.hpp>`):**

i. Class:

```
    #include <ext/pb_ds/assoc_container.hpp>
    using namespace __gnu_pbds;
    template <class Key, class Mapped, class Cmp_Fn = std::
less<Key>, class Tag = rb_tree_tag, class Node_Update =
null_node_update, class Alloc = std::allocator<char>>
class tree;
```

ii. Initialization:

A. `tree<int, null_type, less<int>, rb_tree_tag, tree_order_statisti`
`ost`: Declares an order statistics tree of integers.

B. `tree<int, null_type, less<int>, rb_tree_tag, tree_order_statisti`
`ost {1, 2, 3, 4}`: Declares an order statistics tree of integers with the elements 1, 2, 3, and 4.

C. `tree<int, null_type, less<int>, rb_tree_tag, tree_order_statisti`
`ost = {1, 2, 3, 4}`: Declares an order statistics tree of integers with the elements 1, 2, 3, and 4.

5 Functions

Containers Functions

1. **Vector:**

   (a) `push_back(x)`: Insert element x at the end of the vector. ej:

   ```
   vector<int> v;
   v.push_back(5);
   // New value of v = {5}
   ```

   Note: Amortized complexity of O(1).

   (b) `pop_back()`: Remove the last element of the vector. ej:

   ```
   vector<int> v ({5, 6, 7});
   v.pop_back();
   // New value of v = {5, 6}
   ```

   Note: Complexity of O(1).

   (c) `size()`: Return the number of elements in the vector. ej:

   ```
   vector<int> v ({5, 6, 7});
   v.size();
   // Will return 3
   ```

   Note: Complexity of O(1).

   (d) `at(i)`: Access the element at index i. ej:

   ```
   vector<int> v ({5, 6, 7});
   v.at(1);
   // Will return 6
   ```

   Note: Complexity of O(1).

   (e) `clear()`: Remove all elements from the vector. ej:

   ```
   vector<int> v ({5, 6, 7});
   v.clear();
   // New value of v = {}
   ```

   Note: Complexity of O(n).

2. **Map:**

   (a) `insert({key, value})`: Insert key-value pair in the map. ej:

   ```
   map<int, int> m;
   m.insert({5, 10});
   // New value of m = { {5, 10} }
   ```

   Note: Complexity of O($log_2 n$).

   (b) `erase(key)`: Erase element with the given key from the map. ej:

   ```
   map<int, int> m ({ {5, 10}, {6, 20} });
   m.erase(5);
   // New value of m = { {6, 20} }
   ```

   Note: Complexity of O($log_2 n$).

   (c) `find(key)`: Find element with the given key in the map. ej:

   ```
   map<int, int> m;
   m.find(5);
   // Will return m.end() if key is not in the map
   ```

   Note: Complexity of O($log_2 n$).

   (d) `lower_bound(key)`: Find the first element that is not less than the given key. ej:

   ```
   map<int, int> m ({ {5, 10}, {6, 20} });
   m.lower_bound(6);
   // Will return an iterator to the element {6, 20}
   ```

   Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

   (e) `upper_bound(key)`: Find the first element that is greater than the given key. ej:

   ```
   map<int, int> m ({ {5, 10}, {6, 20} });
   m.upper_bound(6);
   // Will return an iterator to the end of the map
   ```

   Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

3. **Multiset:**

   (a) `insert(x)`: Insert element x in the multiset. ej:

   ```
   multiset<int> ms;
   ms.insert(5);
   // New value of ms = {5}
   ```

   Note: Complexity of O($log_2 n$).

   (b) `erase(x)`: Erase all instances of element x from the multiset. ej:

   ```
   multiset<int> ms ({5, 6, 7, 5});
   ms.erase(5);
   // New value of ms = {6, 7}
   ```

Note: Complexity of O($log_2 n$).

(c) `find(x)`: Find an instance of element x in the multiset. ej:

```
multiset<int> ms;
ms.find(5);
// Will return ms.end() if x is not in the multiset

```

Note: Complexity of O($log_2 n$).

(d) `lower_bound(x)`: Find the first element that is not less than x. ej:

```
multiset<int> ms ({5, 6, 7});
ms.lower_bound(6);
// Will return an iterator to the element 6

```

Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

(e) `upper_bound(x)`: Find the first element that is greater than x. ej:

```
multiset<int> ms ({5, 6, 7});
ms.upper_bound(6);
// Will return an iterator to the element 7

```

Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

4. **Unordered Set:**

(a) `insert(x)`: Insert element x in the unordered set. ej:

```
unordered_set<int> us;
us.insert(5);
// New value of us = {5}

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(b) `erase(x)`: Erase element x from the unordered set. ej:

```
unordered_set<int> us ({5, 6, 7});
us.erase(5);
// New value of us = {6, 7}

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(c) `find(x)`: Find element x in the unordered set. ej:

```
unordered_set<int> us;
us.find(5);
// Will return us.end() if x is not in the unordered set

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(d) `bucket_count()`: Return the number of buckets in the unordered set. ej:

```
unordered_set<int> us ({5, 6, 7});
us.bucket_count();
// Will return the number of buckets

```

Note: Complexity of O(1).

(e) `load_factor()`: Return the load factor of the unordered set. ej:

```
unordered_set<int> us ({5, 6, 7});
us.load_factor();
// Will return the load factor

```

Note: Complexity of O(1).

5. **Unordered Map:**

(a) `insert({key, value})`: Insert key-value pair in the unordered map. ej:

```
unordered_map<int, int> um;
um.insert({5, 10});
// New value of um = { {5, 10} }

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(b) `erase(key)`: Erase element with the given key from the unordered map. ej:

```
unordered_map<int, int> um ({ {5, 10}, {6, 20} });
um.erase(5);
// New value of um = { {6, 20} }

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(c) `find(key)`: Find element with the given key in the unordered map. ej:

```
unordered_map<int, int> um;
um.find(5);
// Will return um.end() if key is not in the unordered map

```

Note: Average complexity of O(1), worst-case complexity of O(n).

(d) `bucket_count()`: Return the number of buckets in the unordered map. ej:

```
unordered_map<int, int> um ({ {5, 10}, {6, 20} });
um.bucket_count();
// Will return the number of buckets

```

Note: Complexity of O(1).

(e) `load_factor()`: Return the load factor of the unordered map. ej:

```
unordered_map<int, int> um ({ {5, 10}, {6, 20} });
um.load_factor();
// Will return the load factor

```

Note: Complexity of O(1).

6. **Order Statistics Tree (using `tree` from `<ext/pb_ds/assoc_container.hpp>`):**

   (a) `insert(x)`: Insert element x in the order statistics tree. ej:

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<int, null_type, less<int>, rb_tree_tag,
  tree_order_statistics_node_update> ost;
ost.insert(5);
// New value of ost = {5}
```

   Note: Complexity of O($log_2 n$).

   (b) `erase(x)`: Erase element x from the order statistics tree. ej:

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<int, null_type, less<int>, rb_tree_tag,
  tree_order_statistics_node_update> ost ({5, 6, 7});
ost.erase(5);
// New value of ost = {6, 7}
```

   Note: Complexity of O($log_2 n$).

   (c) `find_by_order(k)`: Find the k-th smallest element in the order statistics tree. ej:

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<int, null_type, less<int>, rb_tree_tag,
  tree_order_statistics_node_update> ost ({5, 6, 7});
ost.find_by_order(1);
// Will return an iterator to the element 6
```

   Note: Complexity of O($log_2 n$).

   (d) `order_of_key(x)`: Find the number of elements strictly less than x in the order statistics tree. ej:

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<int, null_type, less<int>, rb_tree_tag,
  tree_order_statistics_node_update> ost ({5, 6, 7});
ost.order_of_key(6);
// Will return 1
```

   Note: Complexity of O($log_2 n$).

   (e) `clear()`: Remove all elements from the order statistics tree. ej:

```
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
tree<int, null_type, less<int>, rb_tree_tag,
  tree_order_statistics_node_update> ost ({5, 6, 7});
ost.clear();
// New value of ost = {}
```

   Note: Complexity of O(n).

7. **Set:**

   (a) `insert(x)`: Insert element x in the set. ej:

```
set<int> s;
s.insert(5);
// New value of s = {5}
```

   Note: Complexity of O($log_2 n$).

   (b) `erase(x)`: Erase element x from the set. ej:

```
set<int> s ({5, 6, 7});
s.erase(5);
// New value of s = {6, 7}
```

   Note: Complexity of O($log_2 n$).

   (c) `find(x)`: Find element x in the set. ej:

```
set<int> s;
s.find(5);
// Will return s.end() if x is not in the set
```

   Note: Complexity of O($log_2 n$).

   (d) `lower_bound(x)`: Find the first element that is not less than x. ej:

```
set<int> s ({5, 6, 7});
s.lower_bound(6);
// Will return an iterator to the element 6
```

   Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

   (e) `upper_bound(x)`: Find the first element that is greater than x. ej:

```
set<int> s ({5, 6, 7});
s.upper_bound(6);
// Will return an iterator to the element 7
```

   Note: Complexity of O($log_2 n$). **Note:** Uses Binary Search under the hood.

6 Greedy

7 Two Pointers

8 Search

Binary Search

Ternary Search

9 Sorting

10 Array and Range Sums

11 Dynamic Programming

12 Number Theory

13 Graph Theory