Tugas 5 PBO C - 1

Brendan Timothy Mannuel

5025221177

Interesting Fun of Yura

At the lesson of computer science, Yura became very sad, so he invented a fun for himself.

In the beginning, he has an empty set. At each next turn, he comes up with a number and checks if it belongs to the set. If it belongs, then Yura screams "**Yes**". If not, he screams "**No**", and adds it to the set. Before making a new number, Yura screams the number of elements in the set.

The teacher was tired of Yura's screams, so he made him write a program that screams instead of a boy. But Yura does not know how to program, so he asked for help from you.

Untuk soal ini kita diperlukan untuk mencari apakah suatu bilangan telah tersimpan didalam suatu set yang berisi banyak angka, dimana angka akan diberikan oleh soal. Kita harus mengeluarkan “Yes” jika bilangan sudah terdapat di dalam set, dan “No” jika bilangan belum terdapat didalam set dan kita akan memasukannya, setelah itu kita juga harus mengeluarkan banyak bilangan yang tersimpan didalam set. Pada soal ini kita dapat menggunakan implementasi dari class SET di c++.

std::**set** <set>

```
template < class T,                        // set::key_type/value_type
           class Compare = less<T>,        // set::key_compare/value_compare
```

Set

Sets are containers that store unique elements following a specific order.

In a `set`, the value of an element also identifies it (the value is itself the ***key***, of type `T`), and each value must be unique. The value of the elements in a `set` cannot be modified once in the container (the elements are always const), but they can be inserted or removed from the container.

Internally, the elements in a `set` are always sorted following a specific ***strict weak ordering*** criterion indicated by its internal comparison object (of type `Compare`).

`set` containers are generally slower than unordered_set containers to access individual elements by their ***key***, but they allow the direct iteration on subsets based on their order.

Sets are typically implemented as ***binary search trees***.

Set termasuk sebagai non linear class di stl pada c++, dimana non linear itu berarti bahwa kita tidak harus melakukan pengecekan secara satu satu untuk menyampai target, salah satu contoh utama dari non linear adalah graph dimana dalam 1 node dapat berhubungan dengan banyak node yang lain. Kenapa kita menggunakan set karena kita akan memanfaatkan fungsi dari set yaitu count() dan size()

std::**set::count** <set>

```
size_type count (const value_type& val) const;
```

Count elements with a specific value

Searches the container for elements equivalent to ***val*** and returns the number of matches.

Because all elements in a set container are unique, the function can only return ***1*** (if the element is found) or zero (otherwise).

Two elements of a set are considered equivalent if the container's comparison object returns `false` reflexively (i.e., no matter the order in which the elements are passed as arguments).

count() yang akan mencari suatu value yang sesuai dan akan mereturn angka 1 ketika ketemu dan angka 0 jika tidak menemukan.

public member function

std::set::size

<set>

C++98 C++11

```
size_type size() const;
```

Return container size

Returns the number of elements in the set container.

serta size() yang akan mengembalikan besar ukuran set tersebut

```cpp
#include <set>
#include <cstdio>
using namespace std;
#define gc getchar_unlocked
set<int> set_int;

void Get ( int &ret ){
    ret = 0; char inp=gc(); int kl=1;
    while (inp<'0' || inp>'9'){
        if(inp=='-') kl=-1; inp=gc();}
    while('0'<=inp && inp <= '9')
        ret = (ret<<3) + (ret<<1) + (int)(inp-'0'), inp = gc();
        if (kl < 1) ret =-ret;
}

int main() {
    int tmp, tests;
    Get(tests);

    while(tests--) {
        Get(tmp);

        if (set_int.count(tmp) == 0) {
            printf("No ");
            set_int.insert(tmp);
            printf("%d\n", set_int.size());
        } else {
            printf("Yes ");
            printf("%d\n", set_int.size());
        }
    }

    return 0;
}
```

Kode akan mengecek apakah angka yang diinput ada dalam set, jika tidak ada maka kita akan mengeluarkan “No” dan memasukan angka tersebut kedalam set, serta mengeluarkan ukuran set. Jika terdapat maka kita akan mengeluarkan “Yes” dan besar dari set tanpa harus menginput angkanya.

Tugas 5 PBO C - 2

Brendan Timothy Mannuel

5025221177

Replacement

> Given a sequence of **n** positive integers. You must replace each element with the next nearest one (with a larger index) that is strictly larger than its value. If there is no larger element, replace this element with zero.

Pada soal ini kita akan mengganti suatu angka dengan angka selanjutnya yang lebih besar tetapi harus berada di kanan nya. Kita akan melakukan 2 pendekatan, yaitu secara linear dan non linear.

Untuk cara linear kita akan menggunakan class stack

std::stack <stack>

```
template <class T, class Container = deque<T> > class stack;
```

LIFO stack

Stacks are a type of container adaptor, specifically designed to operate in a LIFO context (last-in first-out), where elements are inserted and extracted only from one end of the container.

stacks are implemented as ***container adaptors***, which are classes that use an encapsulated object of a specific container class as its ***underlying container***, providing a specific set of member functions to access its elements. Elements are ***pushed***/***popped*** from the ***"back"*** of the specific container, which is known as the ***top*** of the stack.

The underlying container may be any of the standard container class templates or some other specifically designed container class. The container shall support the following operations:
`empty`
`size`
`back`
`push_back`
`pop_back`

The standard container classes `vector`, `deque` and `list` fulfill these requirements. By default, if no container class is specified for a particular `stack` class instantiation, the standard container `deque` is used.

Stack adalah sebuah class linear yang menyimpan suatu value secara LIFO atau last in first out.

```
vector<int> sequence(n);
vector<int> res(n, 0);
stack<int> s;
```

Selain menggunakan stack, kita akan menggunakan bantuan class vector sebagai tempat penyimpanan awal dan hasil akhir, kita akan mengisi vector hasil dengan angka 0 terlebih dahulu

```
for (int i = 0; i < n; i++) {
    Get(sequence[i]);
}
```

Kita akan menginputkan deretan angka kedalam vector sequence

```
for (int i = 0; i < n; i++) {
    while (!s.empty() && sequence[i] > sequence[s.top()]) {
        res[s.top()] = sequence[i];
        s.pop();
    }
    s.push(i);
}
```

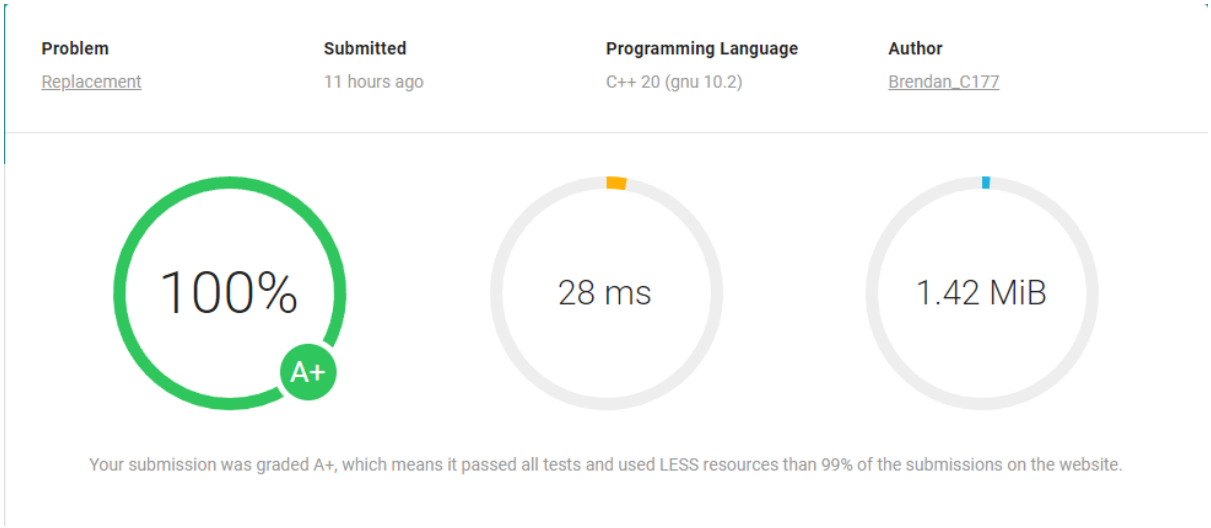
Loop ini adalah bagian penting dari algoritma yang menghitung elemen-elemen berikutnya yang lebih besar dari setiap elemen dalam urutan masukan dan menyimpan hasilnya dalam vektor res.

1. for (int i = 0; i < n; i++): Loop ini akan berjalan sebanyak n kali, di mana n adalah panjang urutan yang diinputkan.
2. while (!s.empty() && sequence[i] > sequence[s.top()]): Loop ini adalah bagian dari algoritma yang mencari elemen-elemen berikutnya yang lebih besar. Ini akan terus berjalan selama stack s tidak kosong dan elemen saat ini dalam urutan sequence[i] lebih besar daripada elemen yang terdapat di puncak stack.
3. res[s.top()] = sequence[i];: Ketika loop di atas terus berjalan, itu berarti elemen di puncak stack saat ini memiliki elemen berikutnya yang lebih besar. Oleh karena itu, nilai dari elemen ini dalam vektor res diisi dengan nilai sequence[i].
4. s.pop();: Setelah nilai res telah diisi, elemen di puncak stack saat ini dihapus dari stack.
5. s.push(i);: Elemen saat ini (indeks i) ditambahkan ke stack untuk mencari elemen yang lebih besar selanjutnya untuk elemen ini.

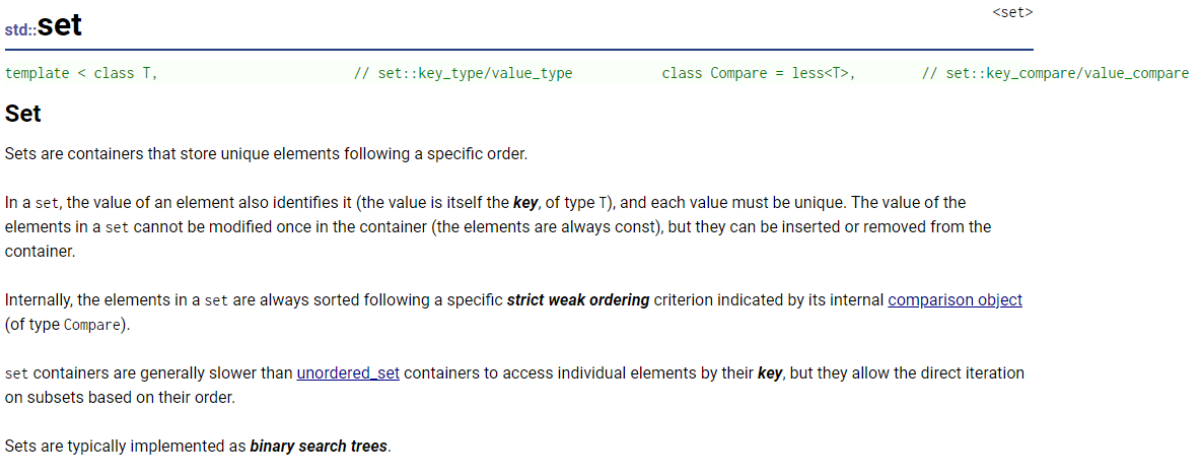
Dengan cara ini, algoritma ini akan mengisi vektor res dengan nilai-nilai yang merupakan elemen-elemen berikutnya yang lebih besar dari setiap elemen dalam urutan sequence.

```
for (int i = 0; i < n; i++) {
    printf("%d ", res[i]);
}
```

Setelah itu kita akan mengoutputkan hasilnya



Untuk pendekatan Kedua kita akan menggunakan pendekatan secara non linear dengan menggunakan set.



std::**set** <set>

```
template < class T,                        // set::key_type/value_type
           class Compare = less<T>,        // set::key_compare/value_compare
```

Set

Sets are containers that store unique elements following a specific order.

In a `set`, the value of an element also identifies it (the value is itself the ***key***, of type T), and each value must be unique. The value of the elements in a `set` cannot be modified once in the container (the elements are always const), but they can be inserted or removed from the container.

Internally, the elements in a `set` are always sorted following a specific ***strict weak ordering*** criterion indicated by its internal comparison object (of type `Compare`).

`set` containers are generally slower than unordered_set containers to access individual elements by their ***key***, but they allow the direct iteration on subsets based on their order.

Sets are typically implemented as ***binary search trees***.

Untuk yang pertama kita akan menggunakan 2 set, yang pertama sebagai iterator dan yang kedua sebagai tempat penyimpanan sementara, serta kita akan menggunakan 2 array

```
#define MAX 100001

set<int> s;
set<int>::iterator it;

int sequence[MAX], res[MAX];
```

Kemudian kita akan masuk kedalam loop utama

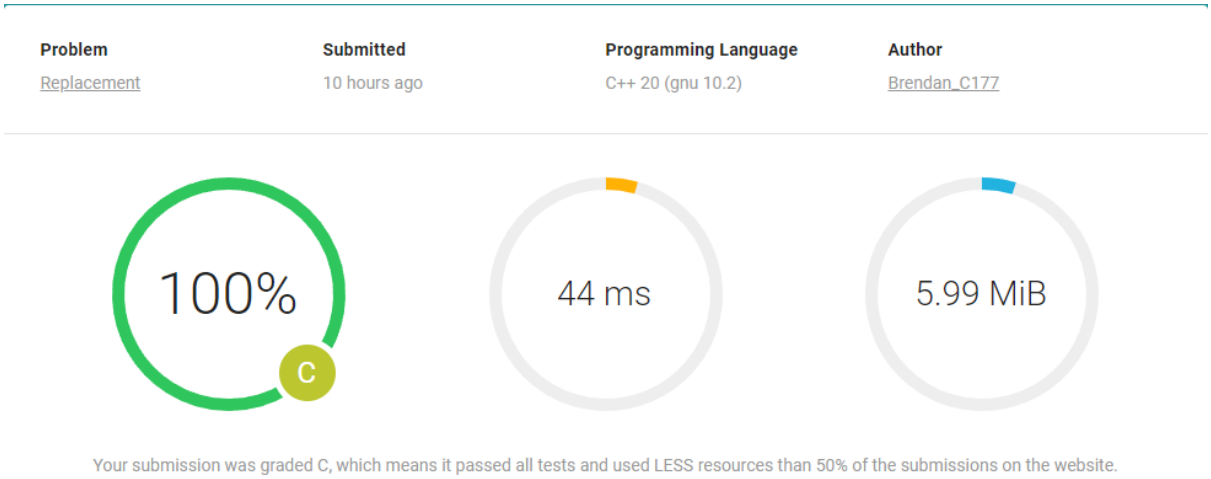
```
for (int i = 0; i < n; i++) {
    Get(sequence[i]);
}

for (int i = n - 1; i >= 0; i--) {
    s.insert(sequence[i]);
    it = s.find(sequence[i]);
    s.erase(s.begin(), it);
    it++;
    if (it == s.end())
        res[i] = 0;
    else
        res[i] = *it;
}

for (int i = 0; i < n; i++) {
    printf("%d ", res[i]);
}
```

Kita akan menginput deretan angka kedalam array sequence, kemudain kita akan memulai array dari kanan menuju kiri. Kemudaian kita akan memasukan angka sekarang kedalam set, kemudian kita akan membuat iterator menunjuk kepada angka sekarang dan kita akan menghapus semua angka yang lebih kecil dari sekarang. Kemudian kita akan memasukan elemen kedua (angka yang lebih besar

selanjutnya) kedalam array result, jika tidak ada maka kita akan memasukan angka 0. Kemudian kita akan mengoutput array result jika loop telah selesai.



KESIMPULAN

Dari kedua cara yang dipakai terlihat bahwa cara linear menggunakan stack lebih cepat dibandingkan dengan cara non linear menggunakan set. Hal ini disebabkan oleh berapa hal, salah satunya adalah tipe soal, dimana soal ini memrlukan adanya pengecekan secara satu persatu dimana tentunya cara linear pasti akan jauh lebih cepat karena cara native dari linear adalah melakukan checking satu satu, dimana non linear tidak melakukan checking satu satu menyebabkan pasti adanya step tambahan yang menyebabkan lebih banyak waktu yang terbuang. Hal ini membuktikan bahwa class yang dipakai juga harus mengikuti permasalahan dari soal, dibawah ini ada beberapa perbedaan antara linear dan non linear:

Linear	Non-Linear
Dalam struktur ini, elemen-elemen disusun secara berurutan atau linear dan melekat satu sama lain.	Dalam struktur ini, elemen-elemen disusun secara hirarkis atau non-linear.
Array, linked list, stack, dan queue	Trees dan graphs
Karena linear, mereka mudah untuk diimplementasikan	Karena organisasi non-linear, mereka sulit untuk diimplementasikan.
Karena struktur data linear memiliki 1 level, maka dibutuhkan satu kali jalan untuk menelusuri setiap item data.	Item data dalam struktur data non-linear tidak dapat diakses dalam sekali traverse. Diperlukan beberapa traversing untuk melintasinya.
Setiap item data terhubung dengan item sebelumnya dan berikutnya	Setiap item terhubung dengan banyak item lainnya.
Struktur data ini tidak mengandung hirarki apa pun, dan semua elemen data diorganisasi dalam satu tingkat.	Dalam hal ini, elemen-elemen data disusun dalam beberapa tingkat.
Kompleksitas waktu dari struktur data linear meningkat mengikuti ukuran input.	Kompleksitas waktu dari struktur data non-linear seringkali tetap sama dengan peningkatan ukuran input.

Dalam hal ini, penggunaan memori tidak efisien.	Dalam hal ini, memori digunakan dengan cara yang sangat efisien.