Competitive STL Extensions

Meeting C++ 2018

Fedor Alekseev

Moscow Institute of Physics and Technology: My pity

November 16, 2018

Outline

Competitive Programming

A contest

- Participants receive a set of 4 to 12 problems, and they have 2 to 5 hours to solve them.

Competitive Programming

A contest

- Participants receive a set of 4 to 12 problems, and they have 2 to 5 hours to solve them.
- Problems are well-defined and usually have computer science nature

Competitive Programming

A contest

- Participants receive a set of 4 to 12 problems, and they have 2 to 5 hours to solve them.
- Problems are well-defined and usually have computer science nature
- Solving a problem means sending a program to the judging system. The program should pass all the secret test cases within time limit

Competitive Programming

A contest

- Participants receive a set of 4 to 12 problems, and they have 2 to 5 hours to solve them.
- Problems are well-defined and usually have computer science nature
- Solving a problem means sending a program to the judging system. The program should pass all the secret test cases within time limit
- Optimal algorithmic complexity is usually enough, especially for C++ solutions

Competitive Programming

A contest

- Participants receive a set of 4 to 12 problems, and they have 2 to 5 hours to solve them.
- Problems are well-defined and usually have computer science nature
- Solving a problem means sending a program to the judging system. The program should pass all the secret test cases within time limit
- Optimal algorithmic complexity is usually enough, especially for C++ solutions
- Solutions are compiled in a judging environment without any additional libraries, with just a vanilla compiler installation.

Standard library

- Algorithms: `sort`, `lower_bound`, `unique`, `next_permutation`, etc
- Data structures: `{unordered_,}{set,map}`, simpler containers

Standard library

- Algorithms: `sort`, `lower_bound`, `unique`, `next_permutation`, etc
- Data structures: {`unordered_`,}{`set`,`map`}, simpler containers
- GNU C++ specific: `#include <bits/stdc++.h>` includes everything!

popcount: number of set bits

```
int main(int argc, const char* argv[]) {
  static_assert(0 == __builtin_popcount(0));  // wow so constexpr
  static_assert(4 == __builtin_popcount(0b1111));
  static_assert(3 == __builtin_popcount(0b100101));
  return __builtin_popcount(argc);
}
```

godbolts under x86 to

```
main:
        xor     eax, eax
        popcnt  eax, edi
        ret
```

Similarly, `__builtin_clz` and `__builtin_ctz` count leading/trailing zeros

SGI STL extensions: power

- Sometimes you have an operation (a^n) that can be expressed as some other operation ($a \cdot a$) repeated n times

SGI STL extensions: power

- Sometimes you have an operation (a^n) that can be expressed as some other operation ($a \cdot a$) repeated n times
- This is usually called exponentiation

SGI STL extensions: power

- Sometimes you have an operation (a^n) that can be expressed as some other operation ($a \cdot a$) repeated n times
- This is usually called exponentiation
- Matrix exponentiation: $A^n = E \cdot \underbrace{A \cdot A \cdot \ldots \cdot A}_{n \text{ times}}$, where E is identity matrix

SGI STL extensions: power

- Sometimes you have an operation (a^n) that can be expressed as some other operation ($a \cdot a$) repeated n times
- This is usually called exponentiation
- Matrix exponentiation: $A^n = E \cdot \underbrace{A \cdot A \cdot \ldots \cdot A}_{n \text{ times}}$, where E is identity matrix
- Integer exponentiation with modulo:

$$a^n \mod p = 1 \cdot \underbrace{(((a \mod p) \cdot a \mod p) \cdot \ldots \cdot a \mod p)}_{n \text{ times } a}$$

SGI STL extensions: power

- Sometimes you have an operation (a^n) that can be expressed as some other operation ($a \cdot a$) repeated n times
- This is usually called exponentiation
- Matrix exponentiation: $A^n = E \cdot \underbrace{A \cdot A \cdot \ldots \cdot A}_{n \text{ times}}$, where E is identity matrix
- Integer exponentiation with modulo:

$$a^n \mod p = 1 \cdot \underbrace{(((a \mod p) \cdot a \mod p) \cdot \ldots \cdot a \mod p)}_{n \text{ times } a}$$

- Can be done in just $O(\log n)$ multiplications

SGI STL extensions: power

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

constexpr int64_t Modulo = 1000000007;  // a prime number
auto multiply_modulo = [](int64_t a, int64_t b) {
  return a * b % Modulo;
};
// this is required to fully define the operation
// will be called through ADL
int64_t identity_element(decltype(multiply_modulo)) {
  return 1;
}
bool fermat_little_theorem_holds(int64_t x) {  // x^p ≡ x (mod p)
  return __gnu_cxx::power(x, Modulo, multiply_modulo) == x % Modulo;
}
```

Policy-Based Data Structures

- There are data structures fundamentally similar to these in the standard library, but with different trade-offs and possibilities

Policy-Based Data Structures

- There are data structures fundamentally similar to these in the standard library, but with different trade-offs and possibilities
- Policy-Based Data Structures library is an attempt to express some of this variety

Policy-Based Data Structures

- There are data structures fundamentally similar to these in the standard library, but with different trade-offs and possibilities
- Policy-Based Data Structures library is an attempt to express some of this variety
- Shipped with GNU C++ library as an extension within namespace `__gnu_pbds`

PBDS: order statistics tree

```
#include <bits/extc++.h>
using namespace __gnu_pbds;

template<typename K, typename V, class Earlier = std::less<K>>
using RankedMap = tree<
  K, V, Earlier,
  rb_tree_tag,  // or splay_tree_tag
  tree_order_statistics_node_update  // extension policy
>;

template<typename K, class Earlier = std::less<K>>
using RankedSet = RankedMap<K, null_type, Earlier>;
```

Lacking utilities

- C++ is a great choice for competitive programming, but

Lacking utilities

- C++ is a great choice for competitive programming, but
- There are some lacking utilities that still tamper its dominance

Lacking utilities

- C++ is a great choice for competitive programming, but
- There are some lacking utilities that still tamper its dominance
- Most importantly, arbitrary precision arithmetics: although problems requiring it are quite rare, sometimes it is easier to switch to python or java just for big integers.

kthxbye

- Thanks!
- More examples are available on my github `https://github.com/moskupols/competitive-stl-extensions`
- For more info on PBDS see GNU C++ library manual: `https://goo.gl/PmR86Z`