Competitive STL Extensions

Meeting C++ 2018

Fedor Alekseev

Moscow Institute of Physics and Technology, My pity

November 16, 2018

Outline

Competitive Programming

Kool tricks
- Standard library
- g++ builtins
- SGI STL extensions
- Policy-Based Data Structures

Lacking utilities

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 some known 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 some known 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 some known 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}`

Standard library

- Algorithms: `sort`, `lower_bound`, `unique`, `next_permutation`, etc
- Data structures: `{unordered_,}{set,map}`
- libstdc++ specifics:

Standard library

- Algorithms: `sort`, `lower_bound`, `unique`, `next_permutation`, etc
- Data structures: `{unordered_,}{set,map}`
- libstdc++ specifics:
  - `#include <algorithm>` includes `std::__gcd`, even in pre-C++17 mode

Standard library

- Algorithms: `sort`, `lower_bound`, `unique`, `next_permutation`, etc
- Data structures: `{unordered_,}{set,map}`
- libstdc++ specifics:
  - `#include <algorithm>` includes `std::__gcd`, even in pre-C++17 mode
  - `#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
```

ctz: Count Trailing Zeros

```
1  int main(int argc, const char* argv[]) {
2      static_assert(32 == __builtin_ctz(0));
3      static_assert(0 == __builtin_ctz(0b1111));
4      static_assert(2 == __builtin_ctz(0b10100));
5      return __builtin_ctz(argc);
6  }
```

godbolts to

```
1  main:
2          xor     eax, eax
3          tzcnt   eax, edi
4          ret
```

Similarly, `__builtin_clz(int)` counts leading zeros

SGI STL extensions: power

```
1  #include <bits/extc++.h>
2
3  constexpr int64_t Modulo = 1000000007;  // a prime number
4  auto multiply_modulo = [](int64_t a, int64_t b) {
5    return a * b % Modulo;
6  };
7  int64_t identity_element(decltype(multiply_modulo)) {
8    return 1;
9  }
10
11 bool fermat_little_theorem_holds(int64_t x) {  // x^p = x  mod p
12   return __gnu_cxx::power(x, Modulo, multiply_modulo) == x % Modulo;
13 }
```

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 libstdc++ as an extension within namespace `__gnu_pbds`

PBDS: order statistics tree

```cpp
#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++ and libstdc++ are great tools for competitive programming, but

Lacking utilities

- C++ and libstdc++ are great tools for competitive programming, but
- There are some lacking utilities that still tamper its dominance

Lacking utilities

- C++ and libstdc++ are great tools 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 libstdc++ manual: `https://goo.gl/PmR86Z`