UsingHashMap

From Algorithmist

It's often the case that you want to improve your submission time on a contest, and you are using a std::map to do lookups on large maps. std::map suffers from a $O(\lg n)$ search cost, since it maintains order. Often times you don't need the order. You can speed up your code by using the non-standard hash_map which has an expected $O(1)$ search cost. However getting the code to work is often problematic, as the judge uses an old version of gcc, and you will sometimes have to define your own hash function. Here is a template for making it all work portably.

```
#ifdef _WIN32
#include "stl_hash.h"
using std::hash_map;
#else
#if defined (__GNUC__) && (__GNUC__ <= 2)
#include <hash_map>
using std::hash_map;
#else
#include <ext/hash_map>
using __gnu_cxx::hash_map;
#endif
#endif
```

Then here is an example of a hash based on a 64 bit integer. The choosen hash of the 64 bit integer simply xors the least significant 32 bits with the most significant 32 bits.

```
struct hash_long_long {
  size_t operator()(const long long in)  const {
  long long ret = (in >> 32L) ^ (in & 0xFFFFFFFF);
  return (size_t) ret;
  }
};
```

Finally, declare your hash_map like this

```
hash_map<long long, int, hash_long_long> memo;
```

Note that hash_set exists and works just as you might expect when including the correct headers. The hash_set header names parallel the hash_map names, just replace map with set.

```
hash_set<long long, hash_long_long> set_of_ll;
```

For a more theoretical look at hash tables, including good hash functions, see Hash Table.

Categories: Cpp | Hashing