

Contents

1	Misc	1
1.1	Contest	1
1.1.1	Makefile	1
1.1.2	Default Code	1
1.2	How Did We Get Here?	1
1.2.1	Macros	1
1.2.2	constexpr	1
1.2.3	Bump Allocator	1
1.3	Tools	1
1.3.1	Floating Point Binary Search	1
1.3.2	SplitMix64	1
1.3.3	<random>	1
1.4	Algorithms	1
1.4.1	Bit Hacks	1
1.4.2	Aliens Trick	2
1.4.3	Hilbert Curve	2

1. Misc

1.1. Contest

1.1.1. Makefile

```
1 .PRECIOUS: ./p%
3 %: p%
    ulimit -s unlimited && ./s$<
5 p%: p%.cpp
    g++ -o $@ $< -std=c++17 -Wall -Wextra -Wshadow \
7         -fsanitize=address,undefined
```

1.1.2. Default Code

```
1 #include <bits/stdc++.h>
3 #define pb          push_back
4 #define eb          emplace_back
5 #define F           first
6 #define S           second
7 #define SZ(v)       ((int)(v).size())
8 #define ALL(v)       (v).begin(), (v).end()
9 #define MEM(a, b)    memset(a, b, sizeof a)
10 #define unpair(p)    (p).F[(p).S
11
12 using namespace std;
13 using ll = long long;
14 using ld = long double;
15 using LL = __int128;
16 using pii = pair<int, int>;
17 using pll = pair<ll, ll>;
19 int main() { ios::sync_with_stdio(0), cin.tie(0); }
```

1.2. How Did We Get Here?

1.2.1. Macros

Use vectorizations and math optimizations at your own peril. For gcc>=9, there are `[[likely]]` and `[[unlikely]]` attributes. Call gcc with `-fopt-info-optimized-missed-optall` for optimization info.

```
1 #define _GLIBCXX_DEBUG      1 // for debug mode
2 #define _GLIBCXX_SANITIZE_VECTOR 1 // for asan on vectors
3 #pragma GCC optimize("O3", "unroll-loops")
4 #pragma GCC optimize("fast-math")
5 #pragma GCC target("avx,avx2,abm,bmi,bmi2") // tip: `lscpu`
6 // before a loop
7 #pragma GCC unroll 16 // 0 or 1 -> no unrolling
8 #pragma GCC ivdep
```

1.2.2. constexpr

Some default limits in g++ (7.x - trunk):

- constexpr recursion depth: 512
- constexpr loop iteration per function: 262144
- constexpr operation count per function: 3354432
- template recursion depth: 900 (g++ might segfault)

```
1 constexpr array<int, 10> fibonacci{[] {
2     array<int, 10> a{};
3     a[0] = a[1] = 1;
4     for (int i = 2; i < 10; i++) a[i] = a[i - 1] + a[i - 2];
5     return a;
6 }()};
7 static_assert(fibonacci[9] == 55, "CE");
9 template <typename F, typename INT, INT... S>
constexpr void for_constexpr(integer_sequence<INT, S...>
```

```
11 F &&func) {
12     int _[] = {(func(integral_constant<INT, S>{}), 0)...};
13 }
14 // example
15 template <typename... T> void print_tuple(tuple<T...> t) {
16     for_constexpr(make_index_sequence<sizeof...(T)>{}),
17         [&](auto i) { cout << get<i>(t) << '\n'; };
18 }
```

1.2.3. Bump Allocator

```
1 // global bump allocator
2 char mem[256 << 20]; // 256 MB
3 size_t rsp = sizeof mem;
4 void *operator new(size_t s) {
5     assert(s < rsp); // MLE
6     return (void *)&mem[rsp -= s];
7 }
8 void operator delete(void *) {}
9
10 // bump allocator for STL / pbds containers
11 char mem[256 << 20];
12 size_t rsp = sizeof mem;
13 template <typename T> struct bump {
14     typedef T value_type;
15     bump() {}
16     template <typename U> bump(U, ...) {}
17     T *allocate(size_t n) {
18         rsp -= n * sizeof(T);
19         rsp &= 0 - alignof(T);
20         return (T *)(&mem + rsp);
21     }
22     void deallocate(T *, size_t n) {}
23 };
```

1.3. Tools

1.3.1. Floating Point Binary Search

```
1 union di {
2     double d;
3     ull i;
4 };
5 bool check(double);
6 // binary search in [L, R] with relative error 2^-eps
7 double binary_search(double L, double R, int eps) {
8     di l = {L}, r = {R}, m;
9     while (r.i - l.i > 1LL << (52 - eps)) {
10         m.i = (l.i + r.i) >> 1;
11         if (check(m.d)) r = m;
12         else l = m;
13     }
14     return l.d;
15 }
```

1.3.2. SplitMix64

```
1 using ull = unsigned long long;
2 inline ull splitmix64(ull x) {
3     // static ull x = seed;
4     ull z = (x += 0x9E3779B97F4A7C15);
5     z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
6     z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
7     return z ^ (z >> 31);
8 }
```

1.3.3. <random>

```
1 #ifdef __unix__
2 random_device rd;
3 mt19937_64 RNG(rd());
4 #else
5 const auto SEED = chrono::high_resolution_clock::now()
6     .time_since_epoch()
7     .count();
8 mt19937_64 RNG(SEED);
9 #endif
10 // random long long: RNG();
11 // uniformly random type T (int, double, ...) in [l, r]:
12 // uniform_int_distribution<T> dist(l, r); dist(RNG);
```

1.4. Algorithms

1.4.1. Bit Hacks

```
1 ull next_permutation(ull x) {
2     ull c = __builtin_ctzll(x), r = x + (1 << c);
3     return (r ^ x) >> (c + 2) | r;
4 }
5 // iterate over all (proper) subsets of bitset s
6 void subsets(ull s) {
7     for (ull x = s; x;) { --x &= s; /* do stuff */ }
8 }
```

1.4.2. Aliens Trick

```
1 // min dp[i] value and its i (smallest one)
2 pll get_dp(int n);
3 ll aliens(int n) {
4     int l = 0, r = 1000000;
5     while (l != r) {
6         int m = (l + r) / 2;
7         auto [f, s] = get_dp(m);
8         if (s == n) return f - m * n;
9         if (s < n) r = m;
10        else l = m + 1;
11    }
12    return get_dp(--l).first - l * n;
13 }
```

1.4.3. Hilbert Curve

```
1 ll hilbert(ll n, int x, int y) {
2     ll res = 0;
3     for (ll s = n / 2; s; s >>= 1) {
4         int rx = (x & s) > 0;
5         int ry = (y & s) > 0;
6         res += s * s * ((3 * rx) ^ ry);
7         if (ry == 0) {
8             if (rx == 1) x = s - 1 - x, y = s - 1 - y;
9             swap(x, y);
10        }
11    }
12    return res;
13 }
```