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 2
  - 1.4.1 Bit Hacks 2
  - 1.4.2 Aliens Trick 2
  - 1.4.3 Hilbert Curve 2

1. Misc

1.1. Contest

1.1.1. Makefile

```
.PRECIOUS: ./p%

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

1.1.2. Default Code

```cpp
#include <bits/stdc++.h>

#define pb        push_back
#define eb        emplace_back
#define F         first
#define S         second
#define SZ(v)     ((int)(v).size())
#define ALL(v)    (v).begin(), (v).end()
#define MEM(a, b) memset(a, b, sizeof a)
#define unpair(p) (p).F][(p).S

using namespace std;
using ll = long long;
using ld = long double;
using LL = __int128;
using pii = pair<int, int>;
using pll = pair<ll, ll>;

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.

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

1.2.2. constexpr

```cpp
constexpr array<int, 10> fibonacci{[] {
  array<int, 10> a{};
  a[0] = a[1] = 1;
  for (int i = 2; i < 10; i++) a[i] = a[i - 1] + a[i - 2];
  return a;
}()};
static_assert(fibonacci[9] == 55, "CE");

template <typename F, typename INT, INT... S>
constexpr void for_constexpr(integer_sequence<INT, S...>,
                             F &&func) {
  int _[] = {(func(integral_constant<INT, S>{}), 0)...};
}
// example
template <typename... T> void print_tuple(tuple<T...> t) {
  for_constexpr(make_index_sequence<sizeof...(T)>{},
                [&](auto i) { cout << get<i>(t) << '\n'; });
}
// some default limits in g++ (7.x - trunk):
// constexpr recursion depth: 512
// constexpr loop iteration (per function): 262144
// constexpr operation count (per function): 33554432
// template recursion depth: 900 (g++ might segfault)
```

1.2.3. Bump Allocator

```cpp
// global bump allocator
char mem[256 << 20]; // 256 MB
size_t rsp = sizeof mem;
void *operator new(size_t s) {
  assert(s < rsp); // MLE
  return (void *)&mem[rsp -= s];
}
void operator delete(void *) {}

// bump allocator for STL / pbds containers
char mem[256 << 20];
size_t rsp = sizeof mem;
template <typename T> struct bump {
  typedef T value_type;
  bump() {}
  template <typename U> bump(U, ...) {}
  T *allocate(size_t n) {
    rsp -= n * sizeof(T);
    rsp &= 0 - alignof(T);
    return (T *)(mem + rsp);
  }
  void deallocate(T *, size_t n) {}
};
```

1.3. Tools

1.3.1. Floating Point Binary Search

```cpp
union di {
  double d;
  ull i;
};
bool check(double);
// binary search in [L, R) with relative error 2^-eps
double binary_search(double L, double R, int eps) {
  di l = {L}, r = {R}, m;
  while (r.i - l.i > 1LL << (52 - eps)) {
    m.i = (l.i + r.i) >> 1;
    if (check(m.d)) r = m;
    else l = m;
  }
  return l.d;
}
```

1.3.2. SplitMix64

```cpp
using ull = unsigned long long;
inline ull splitmix64(ull x) {
  // static ull x = seed;
  ull z = (x += 0x9E3779B97F4A7C15);
  z = (z ^ (z >> 30)) * 0xBF58476D1CE4E5B9;
  z = (z ^ (z >> 27)) * 0x94D049BB133111EB;
  return z ^ (z >> 31);
}
```

1.3.3. <random>

```cpp
#ifdef __unix__
random_device rd;
mt19937_64 RNG(rd());
#else
const auto SEED = chrono::high_resolution_clock::now()
                  .time_since_epoch()
                  .count();
mt19937_64 RNG(SEED);
#endif
// random long long: RNG();
// uniformly random type T (int, double, ...) in [l, r]:
// uniform_int_distribution<T> dist(l, r); dist(RNG);
```

1.4. Algorithms

1.4.1. Bit Hacks

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

1.4.2. Aliens Trick

```cpp
// min dp[i] value and its i (smallest one)
pll get_dp(int n);
ll aliens(int n) {
  int l = 0, r = 1000000;
  while (l != r) {
    int m = (l + r) / 2;
    auto [f, s] = get_dp(m);
    if (s == n) return f - m * n;
    if (s < n) r = m;
    else l = m + 1;
  }
  return get_dp(--l).first - l * n;
}
```

1.4.3. Hilbert Curve

```cpp
ll hilbert(ll n, int x, int y) {
  ll res = 0;
  for (ll s = n / 2; s; s >>= 1) {
    int rx = (x & s) > 0;
    int ry = (y & s) > 0;
    res += s * s * ((3 * rx) ^ ry);
    if (ry == 0) {
      if (rx == 1) x = s - 1 - x, y = s - 1 - y;
      swap(x, y);
    }
  }
  return res;
}
```