Contents

1 Misc

1.1 Makefile

```
.PRECIOUS: ./p%

%: p%
  ulimit -s unlimited && ./$<

p%: p%.cpp
  g++ -o $@ $< -std=gnu++17 -Wall -Wextra -Wshadow \
  -fsanitize=address -fsanitize=undefined

init:
  for i in a b c d e f g h; do \
    cp default.cpp "p$$i.cpp"; \
  done
```

1.2 Bump Allocator

```
// 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 buf[256 << 20];
size_t idx = sizeof buf;
template<typename T> struct bump {
  typedef T value_type;
  bump() {}
  template<typename U> bump(U,...) {}
  T* allocate(size_t n) {
    idx -= n * sizeof(T);
    idx &= 0 - alignof(T);
    return (T*)(buf + idx);
  }
  void deallocate(T*, size_t n) {}
};
```

1.3 SplitMix64

```
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);
}
```

2 Data Structures

2.1 GNU PBDS

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/rope>
using namespace __gnu_pbds;

// most of std::map + order_of_key, find_by_order
template<typename T, typename U = null_type>
using ordered_map = tree<T, U, std::less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
// rb_tree_tag can be changed to splay_tree_tag

template<typename T> struct myhash {
  size_t operator()(T x) const;  // splitmix64, etc.
};
// mostly the same as std::unordered_map
template<typename T, typename U = null_type>
using hash_table = gp_hash_table<T, U, myhash<T>>;

// most of std::priority_queue + merge
using heap = priority_queue<int, std::less<int>>;
// the third template parameter is tag, useful ones are
// pairing_heap_tag, binary_heap_tag, binomial_heap_tag

// similar to treap, has insert/delete range, merge, etc.
using __gnu_cxx::rope;
```

3 Math

3.1 Number Theory

3.1.1 Modular

```
template<typename T> struct M {
  static T MOD;
  T v;
  M() : v(0) {}
  M(T x) {
    v = (-MOD <= x && x < MOD) ? x : x % MOD;
    if (v < 0) v += MOD;
  }
  explicit operator T() const { return v; }
  bool operator==(const M& b) const { return v == b.v; }
  bool operator!=(const M& b) const { return v != b.v; }
  M operator-() { return M(-v); }
  M operator+(M b) { return M(v + b.v); }
  M operator-(M b) { return M(v - b.v); }
  M operator*(M b) { return M((__int128)v * b.v % MOD); }
  M operator/(M b) { return *this * (b ^ (MOD - 2)); }
  friend M operator^(M a, ll b) {
    M ans(1);
    for (; b; b >>= 1, a *= a) if (b & 1) ans *= a;
    return ans;
  }
  friend M& operator+=(M& a, M b) { return a = a + b; }
  friend M& operator-=(M& a, M b) { return a = a - b; }
  friend M& operator*=(M& a, M b) { return a = a * b; }
  friend M& operator/=(M& a, M b) { return a = a / b; }
};
using Mod = M<ll>;
template<>ll Mod::MOD = 1000000007;
ll &MOD = Mod::MOD;

/* Safe primes
 * 21673, 26497, 22621, 21817, 28393, 26821, 30181, 22093
 * 977680993, 971939533, 970479637, 910870273, 1041012121
 * 741266610070171837, 1110995545625882557
 * NTT prime             | p - 1        | primitive root
 * 65537                 | (2^16)       | 3
 * 998244353             | (2^23)*119   | 3
 * 2748779069441         | (2^39)*5     | 3
 * 1945555039024054273 | (2^56)*27    | 5                 */
```

3.1.2 Extended GCD

```
tuple<ll, ll, ll> extgcd(ll a, ll b) {
  if (b == 0) return { 1, 0, a };
  else {
    auto [p, q, g] = extgcd(b, a % b);
    return { q, p - q * (a / b), g };
  }
}
```

3.1.3 Chinese Remainder

```
ll crt(ll a, ll m, ll b, ll n) {
  if (n > m) swap(a, b), swap(m, n);
  auto [x, y, g] = extgcd(m, n);
  assert((a - b) % g == 0);  // no solution
  x = ((b - a) / g * x) % (n / g) * m + a;
  return x < 0 ? x + m / g * n : x;
}
```

3.1.4 Miller-Rabin

```
// checks if Mod::MOD is prime
bool is_prime() {
  if (MOD <= 1 || MOD % 2 == 0) return MOD == 2;
  // Mod A[] = {2, 7, 61};
  Mod A[] = {2,325,9375,28178,450775,9780504,1795265022};
  int s = __builtin_ctz(MOD - 1), i;
  for (Mod a : A) {
    Mod x = a ^ (MOD >> s);
    for (i = 0; i < s && ll(x + 1) > 2; i++, x *= x);
    if (i && x != -1) return 0;
  }
  return 1;
}
```

3.1.5 Tonelli-Shanks

```
int legendre(Mod a) {
  if (a == 0) return 0;
  return (a ^ ((MOD - 1) / 2)) == 1 ? 1 : -1;
}
Mod sqrt(Mod a) {
  assert(legendre(a) != -1);  // no solution
  ll p = MOD, s = p - 1;
  if (a == 0) return 0;
  if (p == 2) return 1;
  if (p % 4 == 3) return a ^ ((p + 1) / 4);
  int r, m;
  for (r = 0; !(s & 1); r++) s >>= 1;
  Mod n = 2;
  while (legendre(n) != -1) n += 1;
  Mod x = a ^ ((s + 1) / 2), b = a ^ s, g = n ^ s;
  while (b != 1) {
    Mod t = b;
    for (m = 0; t != 1; m++) t *= t;
    Mod gs = g ^ (1LL << (r - m - 1));
    g = gs * gs, x *= gs, b *= g, r = m;
  }
  return x;
}
```

3.1.6 Baby-Step Giant-Step

```
// returns x such that a ^ x = b where x \in [l, r)
ll bsgs(Mod a, Mod b, ll l = 0, ll r = MOD - 1) {
  int m = sqrt(r - l) + 1, i;
  unordered_map<ll, ll> tb;
  Mod d = (a ^ l) / b;
  for (i = 0, d = (a ^ l) / b; i < m; i++, d *= a)
    if (d == 1) return l + i;
    else tb[(ll)d] = l + i;
  Mod c = Mod(1) / (a ^ m);
  for (i = 0, d = 1; i < m; i++, d *= c)
    if (auto j = tb.find((ll)d); j != tb.end())
      return j->second + i * m;
  return assert(0), -1;  // no solution
}
```

4 Numeric

4.1 long long Multiplication

```
using ull = unsigned long long;
using ll = long long;
using ld = long double;
// returns a * b % M where a, b < M < 2**63
ull mult(ull a, ull b, ull M) {
  ll ret = a * b - M * ull(ld(a) * ld(b) / ld(M));
  return ret + M * (ret < 0) - M * (ret >= (ll)M);
}
```

4.2 Barrett Reduction

```
using ull = unsigned long long;
using uL = __uint128_t;
// very fast calculation of a % m
struct reduction {
  const ull m, d;
  reduction(ull m) : m(m), d(((uL)1 << 64) / m) {}
  inline ull operator()(ull a) const {
    ull q = (ull)(((uL)d * a) >> 64);
    return (a -= q * m) >= m ? a - m : a;
  }
};
```

4.3 Fast Fourier Transform

```
template<typename T>
void work(int n, vector<T>& a, vector<T>& rt, bool inv) {
  for (int i = 1, r = 0; i < n; i++) {
    for (int bit = n; !(r & bit); bit >>= 1, r ^= bit);
    if (r > i) swap(a[i], a[r]);
  }
  for (int len = 2; len <= n; len <<= 1)
    for (int i = 0; i < n; i += len)
      for (int j = 0; j < len / 2; j++) {
        int pos = n / len * (inv ? len - j : j);
        T u = a[i + j], v = a[i + j + len / 2] * rt[pos];
        a[i + j] = u + v, a[i + j + len / 2] = u - v;
      }
  if (inv) {
    T minv = T(1) / T(n);
    for (T& x : a) x *= minv;
  }
}
void FFT(vector<complex<double>>& a, bool inv) {
  int n = a.size();
  vector<complex<double>> rt(n + 1);
  double arg = acos(-1) * 2 / n;
  for (int i = 0; i <= n; i++)
    rt[i] = { cos(arg * i), sin(arg * i) };
  work(n, a, rt, inv);
}
void NTT(vector<Mod>& a, bool inv, Mod p_root) {
  int n = a.size();
  Mod root = p_root ^ (MOD - 1) / n;
  vector<Mod> rt(n + 1, 1);
  for (int i = 0; i < n; i++) rt[i + 1] = rt[i] * root;
  work(n, a, rt, inv);
}
```

5 Graph

5.1 Flow

5.1.1 Dinic

```
struct Dinic {
  struct edge { int to, cap, flow, rev; };
  static constexpr int MAXN = 1000, MAXF = 1e9;
  vector<edge> v[MAXN];
  int top[MAXN], deep[MAXN], side[MAXN], s, t;
  void make_edge(int s, int t, int cap) {
    v[s].push_back({t, cap, 0, (int)v[t].size()});
    v[t].push_back({s, 0, 0, (int)v[s].size() - 1});
  }
  int dfs(int a, int flow) {
```

```
    if (a == t || !flow) return flow;
    for (int &i = top[a]; i < v[a].size(); i++) {
      edge &e = v[a][i];
      if (deep[a] + 1 == deep[e.to] && e.cap - e.flow) {
        int x = dfs(e.to, min(e.cap - e.flow, flow));
        if (x) {
          e.flow += x, v[e.to][e.rev].flow -= x;
          return x;
        }
      }
    }
    deep[a] = -1;
    return 0;
  }
  bool bfs() {
    queue<int> q;
    fill_n(deep, MAXN, 0);
    q.push(s), deep[s] = 1;
    int tmp;
    while (!q.empty()) {
      tmp = q.front(), q.pop();
      for (edge e : v[tmp])
        if (!deep[e.to] && e.cap != e.flow)
          deep[e.to] = deep[tmp] + 1, q.push(e.to);
    }
    return deep[t];
  }
  int max_flow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, tflow;
    while (bfs()) {
      fill_n(top, MAXN, 0);
      while ((tflow = dfs(s, MAXF)))
        flow += tflow;
    }
    return flow;
  }
  void reset() {
    fill_n(side, MAXN, 0);
    for (auto &i : v) i.clear();
  }
};
```