Contents

1 Basic

1.1 default

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

using namespace std;

#ifdef LOCAL
template<class A, class B>
ostream& operator<<(ostream &os, pair<A, B> p) { return
     os << "(" << p.first << ", " << p.second << ")"; }
template<class ...T> void dbg(T ...x) { char e{}; ((
    cerr << e << x, e = ' '), ...); }
template<class T> void org(T l, T r) { while (l != r)
    cerr << ' ' << *l++; cerr << '\n'; }
#define debug(args...) dbg("(", #args, ") =", args, '\n
    ')
#define orange(args...) dbg("[", #args, ") ="), org(
    args)
#else
#define debug(...) (0)
#define orange(...) (0)
#endif
template<class T> bool chmin(T &a, T b) { return b < a
    and (a = b, true); }
template<class T> bool chmax(T &a, T b) { return a < b
    and (a = b, true); }
#define all(v) (v).begin(), (v).end()
#define rall(v) (v).rbegin(), (v).rend()
#define ff first
#define ss second

using i64 = long long;
using u64 = unsigned long long;

constexpr int mod = 998244353;
template<class ...T> int add(T ...x) { int t{}; return
    (((t += x) %= mod), ...), t; }
template<class ...T> int mul(T ...x) { i64 t{1}; return
     (((t *= x) %= mod), ...), t; }
```

2 Matching & Flow

3 Graph

3.1 2-SAT

```cpp
struct TwoSAT {
  vector<vector<int>> G;
  int n;
  TwoSAT(int _n) : n(_n), G(_n * 2) {}
  int ne(int x) { return x < n ? x + n : x - n; }
  void add_edge(int u, int v) { // u or v
    G[ne(u)].push_back(v);
    G[ne(v)].push_back(u);
  }
  vector<int> solve() {
    vector<int> ans(n * 2, -1);
    vector<int> id(n * 2);
    vector<int> low(n * 2), dfn(n * 2), vis(n * 2);
    vector<int> stk;
    int _t = 0, scc_cnt = 0;
    function<void(int)> dfs = [&](int u) {
      dfn[u] = low[u] = _t++;
      stk.push_back(u);
      vis[u] = 1;
      for (int v : G[u]) {
        if (!vis[v]) {
          dfs(v);
          chmin(low[u], low[v]);
        } else if (vis[v] == 1) {
          chmin(low[u], dfn[v]);
        }
      }
      if (dfn[u] == low[u]) {
        for (int x = -1; x != u; ) {
          x = stk.back(); stk.pop_back();
          vis[x] = 2;
          id[x] = scc_cnt;
          if (ans[x] == -1) {
            ans[x] = 1;
            ans[ne(x)] = 0;
          }
        }
        scc_cnt++;
      }
    };
    for (int i = 0; i < n + n; i++)
      if (!vis[i]) dfs(i);
    for (int i = 0; i < n; i++)
      if (id[i] == id[ne(i)])
        return {};
    ans.resize(n);
    return ans;
  }
};
```

4 Data Structure

4.1 pbds tree

```cpp
#include <bits/extc++.h>
using namespace __gnu_pbds;
template<class T>
using BST = tree<T, null_type, less<T>, rb_tree_tag,
    tree_order_statistics_node_update>;
bst.insert((x << 20) + i);
bst.erase(bst.lower_bound(x << 20));
bst.order_of_key(x << 20) + 1;
*bst.find_by_order(x - 1) >> 20;
*--bst.lower_bound(x << 20) >> 20;
*bst.upper_bound((x + 1) << 20) >> 20;
```

4.2 Centroid Decomposition

```cpp
struct CenDec {
  vector<vector<pair<int, int>>> anc;
  vector<int> Mdis;
  CenDec(const vector<vector<int>> &G) : anc(G.size()),
     Mdis(G.size(), INF) {
    const int n = G.size();

    vector<int> siz(n);
    vector<bool> vis(n);

    function<int(int, int)> getsiz = [&](int u, int f)
    {
      siz[u] = 1;
      for (int v : G[u]) if (v != f and !vis[v])
        siz[u] += getsiz(v, u);
      return siz[u];
    };

    function<int(int, int, int)> find = [&](int u, int
    f, int s) {
      for (int v : G[u]) if (v != f and !vis[v])
        if (siz[v] * 2 >= s) return find(v, u, s);
      return u;
    };

    function<void(int, int, int, int)> caldis = [&](int
     u, int f, int a, int d) {
      anc[u].emplace_back(a, d);
      for (int v : G[u]) if (v != f and !vis[v])
        caldis(v, u, a, d + 1);
    };

    function<void(int)> build = [&](int u) {
      u = find(u, u, getsiz(u, u));
      vis[u] = 1;
      for (int v : G[u]) if (!vis[v]) {
        caldis(v, u, u, 1);
        build(v);
      }
      vis[u] = 0;
    };
    build(0);
  }
  void add(int p) {
    Mdis[p] = 0;
    for (auto [v, d] : anc[p])
      chmin(Mdis[v], d);
  }
  int que(int p) {
    int r = Mdis[p];
    for (auto [v, d] : anc[p])
      chmin(r, Mdis[v] + d);
    return r;
  }
};
```

5 Math

5.1 CRT

```
pair<i64, i64> exgcd(i64 a, i64 b) { // ax + by = 1
  if (b == 0) return {1, 0};
  auto [x, y] = exgcd(b, a % b);
  return {y, x - a / b * y};
};

i64 CRT(vector<pair<i64, i64>> E) {
  i128 R = 0, M = 1;
  for (auto [r, m] : E) {
    i128 d = r - R, g = gcd<i64>(M, m);
    if (d % g != 0) return -1;
    i128 x = exgcd(M / g, m / g).ff * d / g;
    R += M * x;
    M = M * m / g;
    R = (R % M + M) % M;
  }
  return R;
}
```

5.2 Factorize

```
struct Factorize {
  i64 fmul(i64 a, i64 b, i64 p) {
    return (i128)a * b % p;
  }
  i64 fpow(i64 a, i64 b, i64 p) {
    i64 res = 1;
    for (; b; b >>= 1, a = fmul(a, a, p))
      if (b & 1) res = fmul(res, a, p);
    return res;
  }
  bool MillerRabin(i64 n) {
    auto Check = [&](i64 a, i64 u, i64 n, int t) ->
    bool {
      a = fpow(a, u, n);
      if (a == 0 or a == 1 or a == n - 1) return true;
      for (int i = 0; i < t; i++) {
        a = fmul(a, a, n);
        if (a == 1) return false;
        if (a == n - 1) return true;
      }
      return false;
    };

    auto IsPrime = [&](i64 n) -> bool {
      constexpr array<i64, 7> kChk{2, 235, 9375, 28178,
     450775, 9780504, 1795265022};
      // for int: {2, 7, 61}
      if (n < 2) return false;
      if (n % 2 == 0) return n == 2;
      i64 u = n - 1;
      int t = 0;
      while (u % 2 == 0) u >>= 1, t++;
      for (auto v : kChk) if (!Check(v, u, n, t))
     return false;
      return true;
    };
    return IsPrime(n);
  }
  i64 PollardRho(i64 n) {
    if (n % 2 == 0) return 2;
    i64 x = 2, y = 2, d = 1, p = 1;
    auto f = [](i64 x, i64 n, i64 p) -> i64 {
      return ((i128)x * x % n + p) % n;
    };
    while (true) {
      x = f(x, n, p);
      y = f(f(y, n, p), n, p);
      d = __gcd(abs(x - y), n);
      if (d != n and d != 1) return d;
      if (d == n) ++p;
    }
  }
};
```

5.3 NTT

```
// 17 -> 3
// 97 -> 5
// 193 -> 5
// 998244353 -> 3
// 985661441 -> 3

i64 power(i64 a, i64 b, i64 M) {
  i64 ret = 1;
  for (; b; b >>= 1, a = a * a % M)
    if (b & 1) ret = ret * a % M;
  return ret;
};

template<i64 mod, i64 G>
vector<i64> convolution(vector<i64> f, vector<i64> g) {
  const i64 iG = power(G, mod - 2, mod);

  auto NTT = [&](vector<i64> &v, bool inv) {
    int n = v.size();
    for (int i = 0, j = 0; i < n; i++) {
      if (i < j) swap(v[i], v[j]);
      for (int k = n / 2; (j ^= k) < k; k /= 2);
    }
    for (int mid = 1; mid < n; mid *= 2) {
      i64 w = power((inv ? iG : G), (mod - 1) / (mid +
    mid), mod);
      for (int i = 0; i < n; i += mid * 2) {
        i64 now = 1;
        for (int j = i; j < i + mid; j++, now = now * w
     % mod) {
          i64 x = v[j], y = v[j + mid];
          v[j] = (x + y * now) % mod;
          v[j + mid] = (x - y * now) % mod;
        }
      }
    }
    if (inv) {
      i64 in = power(n, mod - 2, mod);
      for (int i = 0; i < n; i++) v[i] = v[i] * in %
    mod;
    }
  };

  int sum = f.size() + g.size() - 1, len = 1;
  while (len < sum) len <<= 1;
  f.resize(len); g.resize(len);
  NTT(f, 0), NTT(g, 0);
  for (int i = 0; i < len; i++) (f[i] *= g[i]) %= mod;
  NTT(f, 1);
  f.resize(sum);
  for (int i = 0; i < sum; i++) if (f[i] < 0) f[i] +=
    mod;

  return f;
}

vector<i64> mul(const vector<i64> &f, const vector<i64>
     &g) {
  constexpr int M1 = 998244353, G1 = 3;
  constexpr int M2 = 985661441, G2 = 3;
  const __int128_t M1M2 = (__int128_t)M1 * M2;
  const __int128_t M1m1 = (__int128_t)M2 * power(M2, M1
     - 2, M1);
  const __int128_t M2m2 = (__int128_t)M1 * power(M1, M2
     - 2, M2);

  auto p = convolution<M1, G1>(f, g);
  auto q = convolution<M2, G2>(f, g);
  auto cal = [&](i64 a, i64 b) -> i64 { return (a *
    M1m1 + b * M2m2) % M1M2; };
  for (int i = 0; i < p.size(); i++) p[i] = cal(p[i], q
     [i]);
  return p;
}
```

5.4 Lucas

```
i64 Lucas(i64 N, i64 M, i64 D) {
  auto Factor = [&](i64 x) -> vector<pair<i64, i64>> {
    vector<pair<i64, i64>> r;
    for (i64 i = 2; x > 1; i++)
      if (x % i == 0) {
        i64 c = 0;
        while (x % i == 0) x /= i, c++;
        r.emplace_back(i, c);
      }
    return r;
  };
  auto Pow = [&](i64 a, i64 b, i64 m) -> i64 {
    i64 r = 1;
    for (; b; b >>= 1, a = a * a % m)
      if (b & 1) r = r * a % m;
    return r;
  };
  vector<pair<i64, i64>> E;
  for (auto [p, q] : Factor(D)) {
    const i64 mod = Pow(p, q, 1 << 30);
    auto CountFact = [&](i64 x) -> i64 {
      i64 c = 0;
      while (x) c += (x /= p);
      return c;
    };
    auto CountBino = [&](i64 x, i64 y) { return
    CountFact(x) - CountFact(y) - CountFact(x - y); };
    auto Inv = [&](i64 x) -> i64 { return (exgcd(x, mod
    ).ff % mod + mod) % mod; };
    vector<i64> pre(mod + 1);
    pre[0] = pre[1] = 1;
    for (i64 i = 2; i <= mod; i++) pre[i] = (i % p == 0
     ? 1 : i) * pre[i - 1] % mod;
    function<i64(i64)> FactMod = [&](i64 n) -> i64 {
      if (n == 0) return 1;
      return FactMod(n / p) * Pow(pre[mod], n / mod,
    mod) % mod * pre[n % mod] % mod;
    };
    auto BinoMod = [&](i64 x, i64 y) -> i64 {
      return FactMod(x) * Inv(FactMod(y)) % mod * Inv(
    FactMod(x - y)) % mod;
    };
    i64 r = BinoMod(N, M) * Pow(p, CountBino(N, M), mod
    ) % mod;
    E.emplace_back(r, mod);
  };
  return CRT(E);
}
```

5.5 FloorSum

```
// sigma 0 ~ n-1: (a * i + b) / m
i64 floor_sum(i64 n, i64 m, i64 a, i64 b) {
  u64 ans = 0;
  if (a < 0) {
    u64 a2 = (a % m + m) % m;
    ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
    a = a2;
  }
  if (b < 0) {
    u64 b2 = (b % m + m) % m;
    ans -= 1ULL * n * ((b2 - b) / m);
    b = b2;
  }
  while (true) {
    if (a >= m) {
      ans += n * (n - 1) / 2 * (a / m);
      a %= m;
    }
    if (b >= m) {
      ans += n * (b / m);
      b %= m;
    }
    u64 y_max = a * n + b;
    if (y_max < m) break;
    n = y_max / m;
    b = y_max % m;
    swap(m, a);
  }
  return ans;
}
```

6 Geometry

6.1 Convex Hull

```
long double cro(Pt a, Pt b, Pt c) { return (b - a) ^ (c
     - a); }
int sig(long double x) { return (x > -eps) - (x < eps);
     }
Pt Interset(Pt a, Pt b, Pt c, Pt d) {
  long double s = cro(c, d, a), t = -cro(c, d, b);
  return (a * t + b * s) / (s + t);
}

template<class T>
struct Convex {
  int n;
  vector<T> A, V, L, U;
  Convex(const vector<T> &_A) : A(_A), n(_A.size()) {
    assert(n >= 3);
    auto it = max_element(all(A));
    L.assign(A.begin(), it + 1);
    U.assign(it, A.end()), U.push_back(A[0]);
    for (int i = 0; i < n; i++) {
      V.push_back(A[(i + 1) % n] - A[i]);
    }
  }
  int inside(T p, const vector<T> &h, auto f) { // 0:
    out, 1: on, 2: in
    auto it = lower_bound(all(h), p, f);
    if (it == h.end()) return 0;
    if (it == h.begin()) return p == *it;
    return 1 - sig(cro(*prev(it), p, *it));
  }
  int inside(T p) {
    return min(inside(p, L, less{}), inside(p, U,
    greater{}));
  }
  static bool cmp(T a, T b) { return sig(a ^ b) > 0; }
  int tangent(T v) {
    auto l = V.begin(), r = V.begin() + L.size() - 1;
    if (v < T()) l = r, r = V.end();
    return (lower_bound(l, r, v, cmp) - V.begin()) % n;
  }
  array<int, 2> tangent2(T p) {
    array<int, 2> t{-1, -1};
    if (inside(p)) return t;
    for (int i = 0; i != t[0]; i = tangent((A[t[0] = i]
     - p)));
    for (int i = 0; i != t[1]; i = tangent((p - A[t[1]
    = i])));
    return t;
  }
  T Find(int l, int r, T a, T b) {
    if (r < l) r += n;
    int s = sig(cro(a, b, A[l % n]));
    while (r - l > 1) {
      (sig(cro(a, b, A[(l + r) / 2 % n])) == s ? l : r)
     = (l + r) / 2;
    }
    return Interset(a, b, A[l % n], A[r % n]);
  };
  vector<T> LineIntersect(T a, T b) { // long double
    int l = tangent(a - b), r = tangent(b - a);
    if (sig(cro(a, b, A[l])) * sig(cro(a, b, A[r])) >=
    0) return {};
    return {Find(l, r, a, b), Find(r, l, a, b)};
  }
};

vector<Pt> Hull(vector<Pt> P) {
  sort(all(P));
  P.erase(unique(all(P)), P.end());
  P.insert(P.end(), rall(P));
  vector<Pt> stk;
  for (auto p : P) {
    while (stk.size() >= 2 and \
        cro(*++stk.rbegin(), stk.back(), p) <= 0 and \
        (*++stk.rbegin() < stk.back()) == (stk.back() <
     p)) {
      stk.pop_back();
    }
    stk.push_back(p);
  }
  stk.pop_back();
  return stk;
}
```

6.2 Dynamic Convex Hull

```
template<class T, class Comp = less<T>>
struct DynamicHull {
  set<T, Comp> H;
  DynamicHull() {}
  void insert(T p) {
    if (inside(p)) return;
    auto it = H.insert(p).ff;
    while (it != H.begin() and prev(it) != H.begin() \
        and cross(*prev(it, 2), *prev(it), *it) <= 0) {
      it = H.erase(--it);
    }
    while (it != --H.end() and next(it) != --H.end() \
        and cross(*it, *next(it), *next(it, 2)) <= 0) {
      it = --H.erase(++it);
    }
  }
  bool inside(T p) {
    auto it = H.lower_bound(p);
    if (it == H.end()) return false;
    if (it == H.begin()) return p == *it;
    return cross(*prev(it), p, *it) <= 0;
  }
};
```

6.3 Half Plane Intersection

```
struct Line {
  Pt a{}, b{};
  Line() {}
  Line(Pt _a, Pt _b) : a{_a}, b{_b} {}
};

Pt Interset(Line L, Line R) {
  double s = cro(R.a, R.b, L.a), t = -cro(R.a, R.b, L.b
    );
  return (L.a * t + L.b * s) / (s + t);
}

vector<Pt> HalfPlaneInter(vector<Line> P) {
  const int n = P.size();
  sort(all(P), [&](Line L, Line R) -> bool {
    Pt u = L.b - L.a, v = R.b - R.a;
    bool f = Pt(sig(u.ff), sig(u.ss)) < Pt{};
    bool g = Pt(sig(v.ff), sig(v.ss)) < Pt{};
    if (f != g) return f < g;
    return (sig(u ^ v) ? sig(u ^ v) : sig(cro(L.a, R.a,
     R.b))) > 0;
  });
  auto Same = [&](Line L, Line R) {
    Pt u = L.b - L.a, v = R.b - R.a;
    return sig(u ^ v) == 0 and sig(u * v) == 1;
  };
  deque <Pt> inter;
  deque <Line> seg;
  for (int i = 0; i < n; i++) if (i == 0 or !Same(P[i -
     1], P[i])) {
    while (seg.size() >= 2 and sig(cro(inter.back(), P[
    i].b, P[i].a)) == 1) {
      seg.pop_back(), inter.pop_back();
    }
    while (seg.size() >= 2 and sig(cro(inter.front(), P
    [i].b, P[i].a)) == 1) {
      seg.pop_front(), inter.pop_front();
    }
    if (!seg.empty()) inter.push_back(Interset(seg.back
    (), P[i]));
    seg.push_back(P[i]);
  }
  while (seg.size() >= 2 and sig(cro(inter.back(), seg.
    front().b, seg.front().a)) == 1) {
    seg.pop_back(), inter.pop_back();
  }
  inter.push_back(Interset(seg.front(), seg.back()));
  return vector<Pt>(all(inter));
}
```

6.4 Minimal Enclosing Circle

```
typedef pair<double, double> pdd;
pdd operator+(const pdd &a, const pdd &b) { return pdd(
    a.ff + b.ff, a.ss + b.ss); }
pdd operator-(const pdd &a, const pdd &b) { return pdd(
    a.ff - b.ff, a.ss - b.ss); }
pdd operator/(const pdd &a, double c) { return pdd(a.ff
     / c, a.ss / c); }
pdd operator*(const pdd &a, double c) { return pdd(a.ff
     * c, a.ss * c); }
double operator*(const pdd &a, const pdd &b) { return a
    .ff * b.ff + a.ss * b.ss; }
double abs(const pdd &x) { return sqrt(x.ff * x.ff + x.
    ss * x.ss); }
using circle = pair<pdd, double>;
struct Mes {
  Mes() {}
  bool inside(const circle &c, pdd p) {
    return abs(p - c.ff) <= c.ss;
  };
  circle get_cir(pdd a, pdd b) {
    return circle((a + b) / 2., abs(a - b) / 2.);
  }
  circle get_cir(pdd a, pdd b, pdd c) {
    pdd p = (b - a) / 2.;
    p = pdd(-p.ss, p.ff);
    double t = ((c - a) * (c - b)) / (2 * (p * (c - a))
        );
    p = ((a + b) / 2.) + (p * t);
    return circle(p, abs(p - a));
  }
  circle get_mes(vector<pdd> P) {
    if (P.empty()) return circle{pdd(0, 0), 0};
    mt19937 rng(random_device{}());
    shuffle(all(P), rng);
    circle C{P[0], 0};
    for (int i = 1; i < P.size(); i++) {
      if (inside(C, P[i])) continue;
      C = get_cir(P[i], P[0]);
      for (int j = 1; j < i; j++) {
        if (inside(C, P[j])) continue;
        C = get_cir(P[i], P[j]);
        for (int k = 0; k < j; k++) {
          if (inside(C, P[k])) continue;
          C = get_cir(P[i], P[j], P[k]);
        }
      }
    }
    return C;
  }
} mes;
```

6.5 Minkowski

```
vector<Pt> Minkowski(vector<Pt> P, vector<Pt> Q) {
  auto reorder = [&](auto &R) -> void {
    auto cmp = [&](Pt a, Pt b) -> bool { return Pt(a.ss
    , a.ff) < Pt(b.ss, b.ff); };
    rotate(R.begin(), min_element(all(R), cmp), R.end()
    );
    R.push_back(R[0]), R.push_back(R[1]);
  };
  const int n = P.size(), m = Q.size();
  reorder(P), reorder(Q);
  vector<Pt> R;
  for (int i = 0, j = 0, s; i < n or j < m; i += (s >=
    0), j += (s <= 0)) {
    R.push_back(P[i] + Q[j]);
    s = sig((P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]));
  }
  return R;
}
```

7 Stringology

7.1 Z-algorithm

```
vector<int> zalgo(string s) {
  if (s.empty()) return {};
  int len = s.size();
  vector<int> z(len);
  z[0] = len;
  for (int i = 1, l = 1, r = 1; i < len; i++) {
    z[i] = i < r ? min(z[i - l], r - i) : 0;
    while (i + z[i] < len and s[i + z[i]] == s[z[i]]) z
    [i]++;
    if (i + z[i] > r) l = i, r = i + z[i];
  }
  return z;
}
```

7.2 Manacher

```
vector<int> manacher(const string &s) {
  string p = "@#";
  for (char c : s) {
    p += c;
    p += '#';
  }
  p += '$';
  vector<int> dp(p.size());
  int mid = 0, r = 1;
  for (int i = 1; i < p.size() - 1; i++) {
    auto &k = dp[i];
    k = i < mid + r ? min(dp[mid * 2 - i], mid + r - i)
     : 0;
    while (p[i + k + 1] == p[i - k - 1]) k++;
    if (i + k > mid + r) {
      mid = i;
      r = k;
    }
  }
  return vector<int>(dp.begin() + 2, dp.end() - 2);
}
```

7.3 SuffixArray

```
namespace sfx {
  const int N = 5e5 + 5;
  bool _t[N * 2];
  int SA[N * 2], H[N], RA[N];
  int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2];
  void pre(int *sa, int *c, int n, int z) {
    fill_n(sa, n, 0), copy_n(c, z, x);
  }
  void induce(int *sa, int *c, int *s, bool *t, int n,
    int z) {
    copy_n(c, z - 1, x + 1);
    for (int i = 0; i < n; ++i) if (sa[i] && !t[sa[i] -
     1]) sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
    copy_n(c, z, x);
    for (int i = n - 1; i >= 0; --i) if (sa[i] && t[sa[
    i] - 1]) sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
  }
  void sais(int *s, int *sa, int *p, int *q, bool *t,
    int *c, int n, int z) {
    bool uniq = t[n - 1] = true;
    int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
    last = -1;
    fill_n(c, z, 0);
    for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
    partial_sum(c, c + z, c);
    if (uniq) {
      for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
      return;
    }
    for (int i = n - 2; i >= 0; --i)
      t[i] = (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i
    + 1]);
    pre(sa, c, n, z);
    for (int i = 1; i <= n - 1; ++i)
      if (t[i] && !t[i - 1])
        sa[--x[s[i]]] = p[q[i] = nn++] = i;
    induce(sa, c, s, t, n, z);
    for (int i = 0; i < n; ++i)
      if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
        bool neq = last < 0 || !equal(s + sa[i], s + p[
    q[sa[i]] + 1], s + last);
        ns[q[last = sa[i]]] = nmxz += neq;
      }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz
     + 1);
    pre(sa, c, n, z);
    for (int i = nn - 1; i >= 0; --i)
      sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
    induce(sa, c, s, t, n, z);
  }
  vector<int> build(vector<int> s) { // s[i] > 0 !!
    const int n = s.size();
    copy_n(begin(s), n, _s), _s[n] = 0;
    sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
    vector<int> sa(n);
    for (int i = 0; i < n; ++i)
      sa[i] = SA[i + 1];
    return sa;
  }
  vector<int> lcp_array(vector<int> &s, vector<int> &sa
    ) {
    int n = int(s.size());
    vector<int> rnk(n);
    for (int i = 0; i < n; i++) rnk[sa[i]] = i;
    vector<int> lcp(n - 1);
    int h = 0;
    for (int i = 0; i < n; i++) {
      if (h > 0) h--;
      if (rnk[i] == 0) continue;
      int j = sa[rnk[i] - 1];
      for (; j + h < n && i + h < n; h++)
        if (s[j + h] != s[i + h]) break;
      lcp[rnk[i] - 1] = h;
    }
    return lcp;
  }
}
```

7.4 PalindromicTree

```
struct PAM {
  struct Node {
    int fail, len, dep;
    array<int, 26> ch;
    Node(int _len) : len{_len}, fail{}, ch{}, dep{} {};
  };
  vector<Node> g;
  vector<int> id;
  int odd, even, lst;
  string S;
  int new_node(int len) {
    g.emplace_back(len);
    return g.size() - 1;
  }
  PAM() : odd(new_node(-1)), even(new_node(0)) {
    lst = g[even].fail = odd;
  }
  int up(int p) {
    while (S.rbegin()[g[p].len + 1] != S.back())
      p = g[p].fail;
    return p;
  }
  int add(char c) {
    S += c;
    lst = up(lst);
    c -= 'a';
    if (!g[lst].ch[c]) g[lst].ch[c] = new_node(g[lst].
    len + 2);
    int p = g[lst].ch[c];
    g[p].fail = (lst == odd ? even : g[up(g[lst].fail)
    ].ch[c]);
    lst = p;
    g[lst].dep = g[g[lst].fail].dep + 1;
    id.push_back(lst);
    return lst;
  }
  void del() {
    S.pop_back();
    id.pop_back();
    lst = id.empty() ? odd : id.back();
  }
};
```

8 Misc