World Final Template

Mingyu Deng, Qipeng Kuang, Lihui Xie
Sun Yat-Sen University

1 Polynomials

```cpp
namespace polynomial {

// dft.h
template <typename T>
class dft {
 public:
  static const bool use_fast_trans = true;
  static void trans(std::vector<T>& p) {
    assert(__builtin_popcount(p.size()) == 1);
    if constexpr (use_fast_trans) {
      dif(p);
    } else {
      bit_reverse(p);
      dit(p);
    }
  }

  static void inv_trans(std::vector<T>& p) {
    assert(__builtin_popcount(p.size()) == 1);
    if constexpr (use_fast_trans) {
      dit(p);
    } else {
      trans(p);
    }
    reverse(p.begin() + 1, p.end());
    T inv = T(p.size()).inv();
    for (T& x : p) x *= inv;
  }

  // should call dit after dif
  static void dit(std::vector<T>& p) {
    for (int len = 1; len < p.size(); len <<= 1) {
      auto sub_w = get_subw(len * 2);
      for (auto sub_p = p.begin(); sub_p != p.end(); sub_p += 2 * len)
        for (int i = 0; i < len; ++i) {
          T u = sub_p[i], v = sub_p[i + len] * sub_w[i];
          sub_p[i] = u + v;
          sub_p[i + len] = u - v;
        }
    }
  }

  static void dif(std::vector<T>& p) {
    for (int len = p.size() / 2; len >= 1; len >>= 1) {
      auto sub_w = get_subw(len * 2);
      for (auto sub_p = p.begin(); sub_p != p.end(); sub_p += 2 * len)
        for (int i = 0; i < len; ++i) {
          T _sub_pi = sub_p[i];
          sub_p[i] += sub_p[i + len];
          sub_p[i + len] = (_sub_pi - sub_p[i + len]) * sub_w[i];
        }
    }
  }

 private:
  typename std::vector<T>::iterator static get_subw(int len) {
    static std::vector<T> w = {0, 1};
    static const T primitive_root = T::primitive_root();
    while (w.size() <= len) {
      T e[] = {1, primitive_root.pow((T::modulus() - 1) / w.size())};
      w.resize(w.size() * 2);
      for (int i = w.size() / 2; i < w.size(); ++i) w[i] = w[i / 2] * e[i & 1];
    }
    return w.begin() + len;
  }
};

// poly.h
template <typename T>
class poly : public std::vector<T> {
 public:
  using std::vector<T>::vector;

  poly(std::string s) {
    for (int i = 0; i < s.size();) {
      auto scan_num = [&]() -> long long {
        int sgn = 1;
        if (s[i] == '-') sgn = -1, ++i;
        if (s[i] == '+') sgn = 1, ++i;
        if (i == s.size() || !std::isdigit(s[i])) return sgn;
        long long num = 0;
        while (i < s.size() && std::isdigit(s[i]))
          num = num * 10 + s[i++] - '0';
        return sgn * num;
```

```cpp
      };
      auto add_item = [&](size_t exponent, T coeff) {
        if (exponent >= this->size()) this->resize(exponent + 1);
        this->at(exponent) = coeff;
      };
      T coeff = scan_num();
      if (i == s.size() || s[i] != 'x')
        add_item(0, coeff);
      else {
        size_t exponent = 1;
        if (s[++i] == '^') {
          ++i;
          exponent = scan_num();
        }
        add_item(exponent, coeff);
      }
    }
  }

  int deg() const { return this->size() - 1; }
  poly operator-() const {
    poly ans = *this;
    for (auto& x : ans) x = -x;
    return ans;
  }
  T operator()(const T& x) const {
    T ans = 0;
    for (int i = this->size() - 1; i >= 0; --i) ans = ans * x + this->at(i);
    return ans;
  }
  T operator[](int idx) const {
    if (0 <= idx && idx < this->size()) return this->at(idx);
    return 0;
  }
  T& operator[](int idx) {
    if (idx >= this->size()) this->resize(idx + 1);
    return this->at(idx);
  }

  poly rev() const {
    poly res(*this);
    std::reverse(res.begin(), res.end());
    return res;
  }

  poly mulxk(size_t k) const {
    poly res = *this;
    res.insert(res.begin(), k, 0);
    return res;
  }
  poly divxk(size_t k) const {
    if (this->size() <= k) return {};
    return poly(this->begin() + k, this->end());
  }
  poly modxk(size_t k) const {
    k = std::min(k, this->size());
    return poly(this->begin(), this->begin() + k);
  }

  poly& operator*=(poly p) {
    if (this->empty() || p.empty()) return *this = {};
    constexpr int small_size = 128;
    if (this->size() < small_size || p.size() < small_size) {
      poly<T> t(this->size() + p.size() - 1);
      for (int i = 0; i < this->size(); i++)
        for (int j = 0; j < p.size(); j++) t[i + j] += this->at(i) * p[j];
      return *this = t;
    }
    int len = 1 << (std::__lg(this->deg() + p.deg()) + 1);
    this->resize(len);
    p.resize(len);
    dft<T>::trans(*this);
    dft<T>::trans(p);
    for (int i = 0; i < len; ++i) this->at(i) *= p[i];
    dft<T>::inv_trans(*this);
    return this->normalize();
  }

  poly& operator+=(const poly& p) {
    this->resize(std::max(this->size(), p.size()));
    for (int i = 0; i < this->size(); ++i) this->at(i) += p[i];
    return this->normalize();
  }
  poly& operator-=(const poly& p) {
    this->resize(std::max(this->size(), p.size()));
    for (int i = 0; i < this->size(); ++i) this->at(i) -= p[i];
    return this->normalize();
  }
  poly& operator/=(const poly& p) {
    if (this->size() < p.size()) return *this = {};
    int len = this->size() - p.size() + 1;
    return *this = (this->rev().modxk(len) * p.rev().inv(len))
                       .modxk(len)
                       .rev()
                       .normalize();
  }
  poly& operator%=(const poly& p) {
    return *this = (*this - (*this / p) * p).normalize();
  }
  poly& operator*=(const T& x) {
    for (int i = 0; i < this->size(); ++i) this->at(i) *= x;
    return *this;
  }
  poly& operator/=(const T& x) { return *this *= x.inv(); }
```

```cpp
  poly operator*(const poly& p) const { return poly(*this) *= p; }
  poly operator+(const poly& p) const { return poly(*this) += p; }
  poly operator-(const poly& p) const { return poly(*this) -= p; }
  poly operator/(const poly& p) const { return poly(*this) /= p; }
  poly operator%(const poly& p) const { return poly(*this) %= p; }
  poly operator*(const T& x) const { return poly(*this) *= x; }
  poly operator/(const T& x) const { return poly(*this) /= x; }

  // (quotient, remainder)
  std::pair<poly, poly> divmod(const poly& p) const {
    poly d = *this / p;
    return std::make_pair(d, (*this - d * p).normalize());
  }

  poly deriv() const {
    if (this->empty()) return {};
    poly res(this->size() - 1);
    for (int i = 0; i < this->size() - 1; ++i) {
      res[i] = this->at(i + 1) * (i + 1);
    }
    return res;
  }
  poly integr(T c = 0) const {
    poly res(this->size() + 1);
    for (int i = 0; i < this->size(); ++i) {
      res[i + 1] = this->at(i) / (i + 1);
    }
    res[0] = c;
    return res;
  }

  // mod x^k
  poly inv(int k = -1) const {
    if (!~k) k = this->size();
    poly res = {this->front().inv()};
    for (int len = 2; len < k * 2; len <<= 1) {
      res = (res * (poly{2} - this->modxk(len) * res)).modxk(len);
    }
    return res.modxk(k);
  }

  // mod x^k
  poly sqrt(int k = -1) const {
    if (!~k) k = this->size();
    poly res = {this->at(0).sqrt()};
    for (int len = 2; len < k * 2; len <<= 1) {
      res = (res + (this->modxk(len) * res.inv(len)).modxk(len)) / 2;
    }
    return res.modxk(k);
  }

  // mod x^k, a0=1 should hold
  poly log(int k = -1) const {
    assert(this->at(0) == 1);
    if (!~k) k = this->size();
    return (this->deriv() * this->inv(k)).integr().modxk(k);
  }
  // mod x^k, a0=0 should hold
  poly exp(int k = -1) const {
    assert(this->at(0) == 0);
    if (!~k) k = this->size();
    poly res = {1};
    for (int len = 2; len < k * 2; len <<= 1) {
      res = (res * (poly{1} - res.log(len) + this->modxk(len))).modxk(len);
    }
    return res.modxk(k);
  }
  // p^c mod x^k
  poly pow(int c, int k = -1) const {
    if (!~k) k = this->size();
    int i = 0;
    while (i < this->size() && !this->at(i)) ++i;
    if (i == this->size() || 1LL * i * c >= k) return {};
    T ai = this->at(i);
    poly f = this->divxk(i) * ai.inv();
    return (f.log(k - i * c) * c).exp(k - i * c).mulxk(i * c) * ai.pow(c);
  }

  // evaluate and interpolate
  struct product_tree {
    int l, r;
    std::unique_ptr<product_tree> lson = nullptr, rson = nullptr;
    poly product;
    product_tree(int l, int r) : l(l), r(r) {}

    static std::unique_ptr<product_tree> build(
        const std::vector<T>& xs, std::function<poly(T)> get_poly) {
      std::function<std::unique_ptr<product_tree>(int, int)> build =
          [&](int l, int r) {
            auto rt = std::make_unique<product_tree>(l, r);
            if (l == r) {
              rt->product = get_poly(xs[l]);
            } else {
              int mid = (l + r) >> 1;
              rt->lson = build(l, mid);
              rt->rson = build(mid + 1, r);
              rt->product = rt->lson->product * rt->rson->product;
            }
            return rt;
          };
      return build(0, xs.size() - 1);
    }
  };
```

```cpp
  poly mulT(poly p) const {
    if (p.empty()) return {};
    return ((*this) * p.rev()).divxk(p.size() - 1);
  }

  std::vector<T> evaluate(std::vector<T> xs) const {
    if (this->empty()) return std::vector<T>(xs.size());
    std::unique_ptr<product_tree> rt = product_tree::build(xs, [&](T x) {
      return poly{1, -x};
    });
    return evaluate_internal(xs, rt);
  }

  static poly interpolate(std::vector<T> xs, std::vector<T> ys) {
    assert(xs.size() == ys.size());
    if (xs.empty()) return {};
    std::unique_ptr<product_tree> rt = product_tree::build(xs, [&](T x) {
      return poly{1, -x};
    });
    std::vector<T> coef = rt->product.rev().deriv().evaluate_internal(xs, rt);
    for (int i = 0; i < ys.size(); ++i) coef[i] = ys[i] * coef[i].inv();
    std::function<poly(product_tree*)> solve = [&](product_tree* rt) {
      if (rt->l == rt->r) {
        return poly{coef[rt->l]};
      } else {
        return solve(rt->lson.get()) * rt->rson->product.rev() +
               solve(rt->rson.get()) * rt->lson->product.rev();
      }
    };
    return solve(rt.get());
  }

  // a0=0 must hold
  poly cos(int k = -1) const {
    assert(this->at(0) == 0);
    if (!~k) k = this->size();
    T i = T::root().pow((T::modulus() - 1) / 4);
    poly x = *this * i;
    return (x.exp(k) + (-x).exp(k)) / 2;
  }

  // a0=0 must hold
  poly sin(int k = -1) const {
    assert(this->at(0) == 0);
    if (!~k) k = this->size();
    T i = T::root().pow((T::modulus() - 1) / 4);
    poly x = *this * i;
    return (x.exp(k) - (-x).exp(k)) / (i * 2);
  }

  // a0=0 must hold
  poly tan(int k = -1) const { return this->sin(k) / this->cos(k); }

  poly acos(int k = -1) const {
    const poly& x = *this;
    return (-x.deriv() * (poly{1} - x * x).sqrt().inv()).integr();
  };
  poly asin(int k = -1) const {
    const poly& x = *this;
    return (x.deriv() * (poly{1} - x * x).sqrt().inv()).integr();
  };
  poly atan(int k = -1) const {
    const poly& x = *this;
    return (x.deriv() * (poly{1} + x * x).inv()).integr();
  };

  friend std::ostream& operator<<(std::ostream& os, poly p) {
    os << "{";
    for (auto x : p) os << x() << " ";
    os << "}";
    return os;
  }

 private:
  poly& normalize() {
    while (this->size() && !this->back()) this->pop_back();
    return *this;
  }

  std::vector<T> evaluate_internal(std::vector<T>& xs,
                                   std::unique_ptr<product_tree>& rt) const {
    std::vector<T> res(xs.size());
    xs.resize(std::max(xs.size(), this->size()));
    std::function<void(product_tree*, poly)> solve = [&](product_tree* rt,
                                                         poly p) {
      p = p.modxk(rt->r - rt->l + 1);
      if (rt->l == rt->r) {
        if (rt->l < res.size()) res[rt->l] = p.front();
      } else {
        solve(rt->lson.get(), p.mulT(rt->rson->product));
        solve(rt->rson.get(), p.mulT(rt->lson->product));
      }
    };
    solve(rt.get(), this->mulT(rt->product.inv(xs.size())));
    return res;
  }
};

}  // namespace polynomial

using poly = polynomial::poly<zint>;
```

1.1 多项式技巧

1. 若 $f(x)=\prod_i(1+x+x^2+\cdots)^{g_i}=\prod_i(\frac{1}{1-x^i})^{g_i}$.

   两边取对数：$\ln f(x)=-\sum_i g_i\cdot ln(1-x^i)=\sum_i g_i\cdot\sum_{j=1}^{\infty}\frac{x^{ij}}{j}$.

   两边取导数：$\frac{\sum_{i=1}^{\infty}(n-1)f_n x^{n-2}}{\sum_{i=1}^{\infty}f_n x^{n-1}}=\sum_i ig_i\cdot\sum_{j=1}^{\infty}x^{ij-1}$.

   即：$\sum_{i=1}^{\infty}(n-1)f_n x^{n-2}=(\sum_{i=1}^{\infty}f_n x^{n-1})\cdot(\sum_i ig_i\cdot\sum_{j=1}^{\infty}x^{ij-1})$.

   乘以 x^2：$\sum_{i=1}^{\infty}(n-1)f_n x^{n}=(\sum_{i=1}^{\infty}f_n x^{n})\cdot(\sum_i ig_i\cdot\sum_{j=1}^{\infty}x^{ij})$

   令 $t=ij$ 并交换求和顺序：$\sum_i ig_i\cdot\sum_{j=1}^{\infty}x^{ij}=\sum_{t=1}^{\infty}x^t\cdot\sum_{i|t}ig_i$

   即：$\sum_{i=1}^{\infty}(n-1)f_n x^{n}=(\sum_{i=1}^{\infty}f_n x^{n})\cdot(\sum_{t=1}^{\infty}x^t\cdot\sum_{i|t}ig_i)$

   对应项系数相等：$(n-1)f_n=\sum_i(\sum_{j|i}jg_j)f_{n-i}$，即

   $$f_n=\frac{1}{n-1}\sum_i(\sum_{j|i}jg_j)f_{n-i}$$

   若令 $ft_n=\sum_{j|i}jg_j$，则 $f_n=\frac{1}{n-1}\sum_i ft_i f_{n-i}$.

2. 若 $F(x)=\sum_{i=0}^{m}f_i\cdot x^i$，那么 $G(x)=F^k(x)$ 有

   $$g_n=\frac{1}{f_0}\cdot\left(\frac{k+1}{n}\sum_{i=1}^{m}if_i\cdot g_{n-i}-\sum_{i=1}^{m}f_i\cdot g_{n-i}\right)$$

2 Strings

```cpp
//****************************** Lyndon ************************

namespace lyndon {

std::vector<int> getfactorization(const std::string &s) {
  std::vector<int> right_ends;
  for (int i = 0; i < s.length();) {
    int j = i, k = i + 1;
    for (; k < s.length() && s[j] <= s[k]; j++, k++)
      if (s[j] < s[k]) j = i - 1;
    while (i <= j) i += k - j, right_ends.push_back(i);
  }
  return right_ends;
}

} // namespace lyndon

//****************************** Pam ************************

template <size_t alphabet_size = 26>
class palindrome_automaton {
 public:
  struct node {
    std::array<int, alphabet_size> to;
    int link, len, count;

    explicit node(int len = 0) : len(len), link(-1), count(0) { to.fill(0); }
    explicit node(int len, int link) : len(len), link(link), count(0) {
      to.fill(0);
    }
  };

  palindrome_automaton() {
    int even_rt = newnode(0);
    int odd_rt = newnode(-1);
    nodes[even_rt].link = odd_rt;
    nodes[odd_rt].link = even_rt;
    last = even_rt;
  }

  void extend(char c) {
    text.push_back(c);
    int i = text.size() - 1;
    auto getlink = [&](int u) {
      while (i - nodes[u].len - 1 < 0 || text[i - nodes[u].len - 1] != c) {
        u = nodes[u].link;
      }
      return u;
    };
    int w = c - 'a';
    int u = getlink(last);
    if (!nodes[u].to[w]) {
      int v = newnode(nodes[u].len + 2, nodes[getlink(nodes[u].link)].to[w]);
      nodes[u].to[w] = v;
    }
    last = nodes[u].to[w];
    ++nodes[last].count;  // should be accmulated later from fail link tree
  }

  std::string to_string() const {
    std::ostringstream os;
    std::function<void(int, std::string)> travel = [&](int k, std::string s) {
```

```cpp
      if (!k) return;
      os << k << ": " << s << " - " << nodes[k].count << "\n";
      for (int c = 0; c < alphabet_size; ++c)
        travel(nodes[k].to[c], std::string(1, c + 'a') + s + (char)(c + 'a'));
    };
    for (int c = 0; c < alphabet_size; ++c)
      travel(nodes[0].to[c], std::string(2, c + 'a'));
    for (int c = 0; c < alphabet_size; ++c)
      travel(nodes[1].to[c], std::string(1, c + 'a'));
    return os.str();
  }

 private:
  int last;
  std::string text;
  std::vector<node> nodes;

  template <typename... Args>
  int newnode(Args... args) {
    int res = nodes.size();
    nodes.push_back(node{args...});
    return res;
  }
};

//***************************** Suffix Array ***********************

template <typename Container = std::vector<int>>
struct SuffixArray {
  int n;
  Container s;
  // lc[0]=0 is meaningless
  std::vector<int> sa, rk, lc;
  SuffixArray(const Container& s)
      : s(s), n(s.size()), sa(s.size()), rk(s.size()), lc(s.size()) {
    std::iota(sa.begin(), sa.end(), 0);
    std::sort(sa.begin(), sa.end(), [&](int a, int b) { return s[a] < s[b]; });
    rk[sa[0]] = 0;
    for (int i = 1; i < n; ++i)
      rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
    std::vector<int> tmp, cnt(n);
    tmp.reserve(n);
    for (int k = 1; rk[sa[n - 1]] < n - 1; k *= 2) {
      tmp.clear();
      for (int i = 0; i < k; ++i) tmp.push_back(n - k + i);
      for (auto i : sa)
        if (i >= k) tmp.push_back(i - k);
      cnt.assign(n, 0);
      for (int i = 0; i < n; ++i) ++cnt[rk[i]];
      for (int i = 1; i < n; ++i) cnt[i] += cnt[i - 1];
      for (int i = n - 1; i >= 0; --i) sa[--cnt[rk[tmp[i]]]] = tmp[i];
      std::swap(rk, tmp);
      rk[sa[0]] = 0;
      for (int i = 1; i < n; ++i) {
        rk[sa[i]] = rk[sa[i - 1]];
        if (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k == n ||
            tmp[sa[i - 1] + k] < tmp[sa[i] + k])
          ++rk[sa[i]];
      }
    }
    for (int i = 0, j = 0; i < n; ++i) {
      if (!rk[i]) {
        j = 0;
      } else {
        if (j) --j;
        int k = sa[rk[i] - 1];
        while (i + j < n && k + j < n && s[i + j] == s[k + j]) ++j;
        lc[rk[i]] = j;
      }
    }
  }
};

template <typename Container = std::vector<int>>
class LongestCommonPrefix {
 public:
  LongestCommonPrefix(SuffixArray<Container>* sa) : sa(sa), st(sa->lc) {}

  int lcp(int i, int j) {
    assert(0 <= i && i <= sa->n);
    assert(0 <= j && j <= sa->n);
    if (i == sa->n || j == sa->n) return 0;
    if (i == j) return sa->n - i;
    int l = sa->rk[i], r = sa->rk[j];
    if (l > r) std::swap(l, r);
    return st.queryMin(l + 1, r);
  }

 private:
  SuffixArray<Container>* sa;
  SparseTable<int> st;
};

//***************************** SAM ***********************

template <size_t alphabet_size = 26>
class suffix_automaton {
 public:
  struct node {
    std::array<int, alphabet_size> to;
    int link, len, count;

    explicit node(int len = 0) : len(len), link(-1), count(0) { to.fill(-1); }
    explicit node(int len, int link, const std::array<int, alphabet_size>& to)
```

```cpp
        : len(len), link(link), to(to), count(0) {}
  };

  suffix_automaton() : nodes() { newnode(); }

  explicit suffix_automaton(const std::string& s) : suffix_automaton() {
    insert(s);
  }

  void insert(const std::string& s) {
    nodes.reserve(size() + s.size() * 2);
    int last = 0;
    for (int i = 0; i < s.size(); ++i) {
      last = extend(last, s[i] - 'a');
    }
  }

  int extend(int k, int c) {
    if (~nodes[k].to[c] && nodes[nodes[k].to[c]].len == nodes[k].len + 1) {
      return nodes[k].to[c];
    }
    int leaf = newnode(nodes[k].len + 1);
    for (; ~k && !~nodes[k].to[c]; k = nodes[k].link) nodes[k].to[c] = leaf;
    if (!~k) {
      nodes[leaf].link = 0;
    } else {
      int p = nodes[k].to[c];
      if (nodes[k].len + 1 == nodes[p].len) {
        nodes[leaf].link = p;
      } else {
        int np = newnode(nodes[k].len + 1, nodes[p].link, nodes[p].to);
        nodes[p].link = nodes[leaf].link = np;
        for (; ~k && nodes[k].to[c] == p; k = nodes[k].link)
          nodes[k].to[c] = np;
      }
    }
    return leaf;
  }

  void build_ancestors() {
    for (int i = 1; i < size(); ++i) ancestors[i] = {nodes[i].link};
    for (int j = 1; (1 << j) < size(); ++j) {
      for (int i = 0; i < size(); ++i)
        if (~ancestors[i][j - 1]) {
          ancestors[i][j] = ancestors[ancestors[i][j - 1]][j - 1];
        } else {
          ancestors[i][j] = -1;
        }
    }
  }

  std::vector<int> mark_count(const std::string& s) {
    std::vector<int> ends;
    int k = 0;
    for (char c : s) {
      k = nodes[k].to[c - 'a'];
      assert(~k);
      ends.push_back(k);
      ++nodes[k].count;
    }
    return ends;
  }

  void addup_count() {
    std::vector<int> ids(size()), bucket(size());
    for (int i = 0; i < size(); ++i) ++bucket[nodes[i].len];
    for (int i = 1; i < bucket.size(); ++i) bucket[i] += bucket[i - 1];
    for (int i = 0; i < size(); ++i) ids[--bucket[nodes[i].len]] = i;
    for (int i = size() - 1; i; --i)
      nodes[nodes[ids[i]].link].count += nodes[ids[i]].count;
  }

  const node& operator[](int v) const { return nodes[v]; }

  int maxlen(int v) const { return nodes[v].len; }

  int minlen(int v) const { return v ? nodes[nodes[v].link].len + 1 : 0; }

  int size() const { return nodes.size(); }

  std::string to_string() const {
    std::ostringstream os;
    std::function<void(int, std::string)> travel = [&](int k, std::string s) {
      if (!~k) return;
      os << k << ": " << s << " - " << nodes[k].count << "\n";
      for (int c = 0; c < alphabet_size; ++c)
        travel(nodes[k].to[c], s + (char)(c + 'a'));
    };
    travel(0, "");
    return os.str();
  }

 private:
  std::vector<node> nodes;
  std::vector<std::vector<int>> ancestors;

  template <typename... Args>
  int newnode(Args... args) {
    int res = nodes.size();
    nodes.push_back(node{args...});
    return res;
  }
};
```

```
//****************************** ACauto ***********************************

int go[maxtri][26],sum,count[maxtri],d[maxtri],fail[maxtri];
void make_tri(int k) {
    fo(i,0,nb-1) {
        int index=sb[i]-'a';
        if (!go[k][index]) go[k][index]=++sum;
        k=go[k][index];
    }
    count[k]++;
}
void make_fail() {
    int i=0, j=0;
    fo(p,0,25) if (go[0][p]) d[++j]=go[0][p];
    while (i++<j) {
        int now=d[i];
        fo(p,0,25) if (go[now][p]) {
            int son=go[now][p];
            fail[son]=go[fail[now]][p];
            count[son]+=count[fail[son]];
            d[++j]=son;
        } else go[now][p]=go[fail[now]][p];
    }
}
void find(int k) { //sa中出现了多少次sb
    fo(i,1,n) ans+=count[k=go[k][sa[i]-'a']];
}

//****************************** manacher **********************************

int f[2*maxn];
void manacher()
{
    int lim=0, mid=0;
    fo(i,1,m) // m=2*n+1
    {
        f[i]= (i<=lim) ?min(f[mid*2-i],lim-i+1) :1 ;
        while (i-f[i]>0 && i+f[i]<=m && s[i-f[i]]==s[i+f[i]]) f[i]++;
        if (i+f[i]-1>lim) lim=i+f[i]-1, mid=i;
    }
}

//****************************** exkmp ************************************

int next[maxn],ex[maxn];
void exkmp() {
    next[1]=nb;
    int k=0;
    fo(i,2,nb) {
        int lim=k+next[k]-1, L=next[i-k+1];
        if (i+L<=lim) next[i]=L; else {
            next[i]=max(lim-i+1,0);
            while (i+next[i]<=nb && sb[i+next[i]]==sb[1+next[i]]) next[i]++;
            k=i;
        }
    }

    k=1;
    fo(i,1,na) {
        int lim=k+ex[k]-1, L=next[i-k+1];
        if (i+L<=lim) ex[i]=L; else {
            ex[i]=max(lim-i+1,0);
            while (i+ex[i]<=na && 1+ex[i]<=nb && sa[i+ex[i]]==sb[1+ex[i]]) ex[i]++;
            k=i;
        }
    }
}

//****************************** min_representation ******************************

int s[maxn];
int min_representation(int *s,int len) {    // index from 0
    int i=0, j=1;
    while (i<len && j<len) {
        int k=0;
        for(; k<len && s[(i+k)%len]==s[(j+k)%len]; k++);
        if (k==len) break;
        (s[(i+k)%len]<s[(j+k)%len]) ?j+=k+1 :i+=k+1;
        i+=(i==j);
    }
    return min(i,j);
}
```

3 Geometry

```
    namespace geometry2d {

constexpr long double eps = 1e-7;
constexpr long double pi = std::acos(-1);

int fsign(long double x) { return (x > eps) - (x < -eps); }
long double sqr(long double x) { return x * x; }

// Ax^2+Bx+c=0
std::vector<long double> solveEquationP2(long double A, long double B,
                                          long double C) {
  long double delta = B * B - 4 * A * C;
  if (fsign(delta) < 0) return {};
  if (fsign(delta) == 0) return {-0.5 * B / A};
  long double sqrt_delta = std::sqrt(delta);
```

```cpp
  long double x1 = -0.5 * (B - sqrt_delta) / A,
              x2 = -0.5 * (B + sqrt_delta) / A;
  if (fsign(x1 - x2) > 0) std::swap(x1, x2);
  return {x1, x2};
}

struct Point;
struct Point3D {
  long double x, y, z;

  explicit Point3D(long double x = 0, long double y = 0, long double z = 0)
      : x(x), y(y), z(z) {}

  explicit Point3D(const Point& p);

  Point3D operator-(const Point3D& p) const {
    return Point3D(x - p.x, y - p.y, z - p.z);
  }
  long double innerProd(const Point3D& p) const {
    return x * p.x + y * p.y + z * p.z;
  }
  Point3D crossProd(const Point3D& p) const {
    return Point3D(y * p.z - z * p.y, -x * p.z + z * p.x, x * p.y - y * p.x);
  }
};

struct Point {
  long double x, y;
  explicit Point(long double x = 0, long double y = 0) : x(x), y(y) {}
  Point operator+(const Point& p) const { return Point(x + p.x, y + p.y); }
  Point operator-(const Point& p) const { return Point(x - p.x, y - p.y); }
  Point operator/(const long double& l) const { return Point(x / l, y / l); }
  Point operator*(const long double& l) const { return Point(x * l, y * l); }
  Point unit() const {
    long double l = len();
    assert(fsign(l) > 0);
    return *this / l;
  }
  long double crossProd(const Point& p) const { return x * p.y - y * p.x; }
  long double innerProd(const Point& p) const { return x * p.x + y * p.y; }
  long double lenSqr() const { return sqr(x) + sqr(y); }
  long double len() const { return std::sqrt(lenSqr()); }
  long double distanceSqr(const Point& p) const {
    return sqr(x - p.x) + sqr(y - p.y);
  }
  long double distance(const Point& p) const {
    return std::sqrt(distanceSqr(p));
  }
  long double angleWith(const Point& p) const {
    return std::atan2(crossProd(p), innerProd(p));
  }
  // return (-pi,pi]
  long double angle() const { return std::atan2(y, x); }
  int angleSign() const {
    if (!y) return fsign(x) < 0;
    return fsign(y);
  }
  Point rotate90() const { return Point(-y, x); }

  static long double crossProd(const Point& o, const Point& a, const Point& b) {
    return (a - o).crossProd(b - o);
  }
  static long double innerProd(const Point& o, const Point& a, const Point& b) {
    return (a - o).innerProd(b - o);
  }

  friend std::ostream& operator<<(std::ostream& os, Point p) {
    return os << "(" << p.x << "," << p.y << ")";
  }

  friend bool operator==(const Point& lhs, const Point& rhs) {
    return !fsign(lhs.x - rhs.x) && !fsign(lhs.y - rhs.y);
  }

  struct HorizontalComparer {
    bool operator()(const Point& lhs, const Point& rhs) const {
      return lhs.x < rhs.x || (lhs.x == rhs.x && lhs.y < rhs.y);
    }
  };

  struct PolarComparer {
    bool operator()(const Point& lhs, const Point& rhs) const {
      int angleSign_diff = lhs.angleSign() - rhs.angleSign();
      if (angleSign_diff) return angleSign_diff < 0;
      int crossProd_sign = fsign(lhs.crossProd(rhs));
      if (crossProd_sign) return crossProd_sign > 0;
      return lhs.lenSqr() < rhs.lenSqr();
    }
  };
};

Point3D::Point3D(const Point& p) : Point3D(p.x, p.y, p.x * p.x + p.y * p.y) {}

struct Line {
  Point pivot, unit_direction;
  Line(Point pivot, Point direction)
      : pivot(pivot), unit_direction(direction.unit()) {}
  long double angle() const { return unit_direction.angle(); }
  // 1: on the left
  // 0: in the line
  // -1: on the right
  int side(const Point& p) {
    return fsign(unit_direction.crossProd(p - pivot));
  }
```

```cpp
  bool isParallel(const Line& l) {
    return fsign(unit_direction.crossProd(l.unit_direction)) == 0;
  }
  bool isSame(const Line& l) {
    return isParallel(l) &&
           fsign(unit_direction.crossProd(l.pivot - pivot)) == 0;
  }
  bool isSameDirection(const Line& l) {
    return isParallel(l) &&
           fsign(unit_direction.innerProd(l.unit_direction)) > 0;
  }
  Point intersection(const Line& l) {
    assert(!isParallel(l));
    return pivot + unit_direction *
                       (l.unit_direction.crossProd(l.pivot - pivot)) /
                       l.unit_direction.crossProd(unit_direction);
  }

  friend std::ostream& operator<<(std::ostream& os, Line l) {
    return os << "(" << l.pivot << "," << l.unit_direction << ")";
  }
};

struct Segment {
  Point a, b;
  Segment(Point a, Point b) : a(a), b(b) {}
  bool hasIntersection(const Segment& s) const {
    return fsign(Point::crossProd(a, b, s.a)) *
                   fsign(Point::crossProd(a, b, s.b)) <
               0 &&
           fsign(Point::crossProd(s.a, s.b, a)) *
                   fsign(Point::crossProd(s.a, s.b, b)) <
               0;
  }
  bool isPointIn(const Point& p) const {
    return fsign(Point::innerProd(p, a, b)) <= 0 &&
           fsign(Point::crossProd(p, a, b)) == 0;
  }

  Point lerp(const long double& ratio) const { return a + (b - a) * ratio; }

  friend std::ostream& operator<<(std::ostream& os, Segment s) {
    return os << "(" << s.a << "," << s.b << ")";
  }
};

struct Polygon {
  static bool isConvexInCCW(const std::vector<Point>& p) {
    for (int i = 0; i < p.size(); ++i) {
      int l = (i + p.size() - 1) % p.size();
      int r = (i + 1) % p.size();
      if (Point::crossProd(p[i], p[l], p[r]) > 0) return false;
    }
    return true;
  }
  static std::vector<int> convexHullId(const std::vector<Point>& p) {
    if (p.size() == 0) return {};
    if (p.size() == 1) return {0};
    std::vector<int> ids(p.size());
    std::iota(ids.begin(), ids.end(), 0);
    sort(ids.begin(), ids.end(),
         [&, comp = Point::HorizontalComparer()](int i, int j) {
           return comp(p[i], p[j]);
         });
    std::vector<int> res;
    for (int i : ids) {
      while (res.size() > 1 &&
             fsign(Point::crossProd(p[res.end()[-2]], p[res.end()[-1]],
                                    p[i])) <= 0)
        res.pop_back();
      res.push_back(i);
    }
    ids.pop_back();
    std::reverse(ids.begin(), ids.end());
    int lower_size = res.size();
    for (int i : ids) {
      while (res.size() > lower_size &&
             fsign(Point::crossProd(p[res.end()[-2]], p[res.end()[-1]],
                                    p[i])) <= 0)
        res.pop_back();
      res.push_back(i);
    }
    res.pop_back();
    return res;
  }
  static std::vector<Point> convexHullPoint(const std::vector<Point>& p) {
    std::vector<int> ids = convexHullId(p);
    std::vector<Point> res;
    for (int i : ids) res.push_back(p[i]);
    return res;
  }

  // should be guaranteed that convex is a convex hull in ccw
  static bool isPointInConvexCCW(const Point& p,
                                 const std::vector<Point>& convex) {
    assert(Polygon::isConvexInCCW(convex));
    for (int i = 0; i < convex.size(); ++i) {
      Point a = convex[i];
      Point b = convex[(i + 1) % convex.size()];
      if (Point::crossProd(a, b, p) < 0) return false;
    }
    return true;
  }
};
```

```cpp

struct Circle {
  Point o;
  long double r;
  Circle() : Circle(Point(0, 0), 0) {}
  Circle(Point o, long double r) : o(o), r(r) {}
  Point pointInDirection(long double angle) {
    return Point(o.x + r * std::cos(angle), o.y + r * std::sin(angle));
  }

  std::vector<Point> intersection(const Line& l) const {
    long double A = 1;
    long double B = l.unit_direction.innerProd(l.pivot - o) * 2;
    long double C = o.distanceSqr(l.pivot) - sqr(r);
    std::vector<long double> roots = solveEquationP2(A, B, C);
    std::vector<Point> intersects;
    for (long double x : roots)
      intersects.push_back(l.pivot + l.unit_direction * x);
    return intersects;
  }
  std::vector<Point> intersection(const Segment& s) const {
    std::vector<Point> line_intersects = intersection(Line(s.a, s.b - s.a));
    std::vector<Point> intersects;
    for (const Point& p : line_intersects)
      if (s.isPointIn(p)) {
        intersects.push_back(p);
      }
    return intersects;
  }

  // triangle oab
  long double overlapAreaWithTriangle(const Point& a, const Point& b) const {
    if (!fsign(Point::crossProd(o, a, b))) return 0;
    std::vector<Point> key_points;
    key_points.push_back(a);
    for (const Point& p : intersection(Segment(a, b))) key_points.push_back(p);
    key_points.push_back(b);
    long double res = 0;
    for (int i = 1; i < key_points.size(); ++i) {
      Point mid_point = (key_points[i - 1] + key_points[i]) / 2;
      Point ray1 = key_points[i - 1] - o;
      Point ray2 = key_points[i] - o;
      if (o.distanceSqr(mid_point) <= sqr(r)) {
        res += std::abs(ray1.crossProd(ray2));
      } else {
        res += sqr(r) * std::abs(ray1.angleWith(ray2));
      }
    }
    dbg(a, b, key_points, res);
    return 0.5 * res;
  }

  long double overlapAreaWithPolygon(const std::vector<Point>& p) const {
    long double res = 0;
    for (int i = 0; i < p.size(); ++i) {
      int j = (i + 1) % p.size();
      res += overlapAreaWithTriangle(p[i], p[j]) *
             fsign(Point::crossProd(o, p[i], p[j]));
    }
    dbg(res);
    return res;
  }

  // 1: outside
  // 0: edge
  // -1: inside
  // equal to solving following determinant (a,b,c is counter-clockwise)
  // | ax, ay, ax^2+ay^2, 1 |
  // | bx, by, bx^2+by^2, 1 |
  // | cx, cy, cx^2+cy^2, 1 |
  // | px, py, px^2+py^2, 1 |
  static int side(const Point& a, Point b, Point c, const Point& p) {
    if (fsign(Point::crossProd(a, b, c)) < 0) std::swap(b, c);
    Point3D a3(a), b3(b), c3(c), p3(p);
    b3 = b3 - a3;
    c3 = c3 - a3;
    p3 = p3 - a3;
    Point3D f = b3.crossProd(c3);
    return fsign(p3.innerProd(f));
  }

  friend std::ostream& operator<<(std::ostream& os, const Circle& c) {
    return os << "(" << c.o << "," << c.r << ")";
  }
};

// return the intersection convex in ccw, should be guaranteed that the
// intersection is finite.
struct HalfPlaneIntersection {
  static std::vector<Point> solve(std::vector<Line> lines) {
    sort(lines.begin(), lines.end(),
         [comp = Point::PolarComparer()](auto l1, auto l2) {
           if (l1.isSameDirection(l2)) {
             return l1.side(l2.pivot) < 0;
           } else {
             return comp(l1.unit_direction, l2.unit_direction);
           }
         });

    std::deque<Line> key_lines;
    std::deque<Point> key_points;
    for (int i = 0; i < lines.size(); ++i) {
      if (i > 0 && lines[i - 1].isSameDirection(lines[i])) continue;
      while (key_points.size() && lines[i].side(key_points.back()) <= 0) {
```

```
        key_lines.pop_back();
        key_points.pop_back();
      }
      while (key_points.size() && lines[i].side(key_points.front()) <= 0) {
        key_lines.pop_front();
        key_points.pop_front();
      }
      if (key_lines.size()) {
        // since it's guaranteed that the intersection is finite, therefore must
        // be empty.
        if (lines[i].isParallel(key_lines.back())) return {};
        key_points.push_back(lines[i].intersection(key_lines.back()));
      }
      key_lines.push_back(lines[i]);
    }

    while (key_points.size() &&
           key_lines.front().side(key_points.back()) <= 0) {
      key_lines.pop_back();
      key_points.pop_back();
    }

    if (key_lines.size() <= 2) return {};

    std::vector<Point> convex;
    for (int i = 0; i < key_lines.size(); ++i)
      convex.emplace_back(
          key_lines[i].intersection(key_lines[(i + 1) % key_lines.size()]));
    return convex;
  }
};

struct Triangulation {
  struct Edge {
    int v;
    std::list<Edge>::iterator rev;
    Edge(int v = 0) : v(v) {}
  };
  // delaunay triangulation
  // should be guaranteed that all points are pairwise distinct
  static std::vector<std::pair<int, int>> nearest(const std::vector<Point>& p) {
    std::vector<std::list<Edge>> neighbor(p.size());
    std::vector<int> id(p.size());
    std::iota(id.begin(), id.end(), 0);
    std::sort(id.begin(), id.end(),
              [&, comp = Point::HorizontalComparer()](int i, int j) {
                return comp(p[i], p[j]);
              });

    auto addedge = [&](int u, int v) {
      neighbor[u].push_front(v);
      neighbor[v].push_front(u);
      neighbor[u].front().rev = neighbor[v].begin();
      neighbor[v].front().rev = neighbor[u].begin();
    };
    std::function<void(int, int)> divide = [&](int l, int r) {
      if (r - l + 1 <= 3) {
        for (int i = l; i <= r; ++i)
          for (int j = l; j < i; ++j) addedge(id[i], id[j]);
        return;
      }

      int mid = (l + r) >> 1;
      divide(l, mid);
      divide(mid + 1, r);

      auto get_base_LR_edge = [&]() {
        std::vector<int> stk;
        for (int i = l; i <= r; ++i) {
          while (stk.size() >= 2 &&
                 fsign(Point::crossProd(p[id[stk.end()[-2]]],
                                        p[id[stk.end()[-1]]], p[id[i]])) < 0)
            stk.pop_back();
          stk.push_back(i);
        }
        for (int i = 1; i < stk.size(); ++i)
          if (stk[i - 1] <= mid && stk[i] > mid)
            return std::make_pair(id[stk[i - 1]], id[stk[i]]);
      };

      auto [ld, rd] = get_base_LR_edge();

      while (true) {
        addedge(ld, rd);
        Point ptL = p[ld], ptR = p[rd];
        int ch = -1, side = -1;
        for (auto it = neighbor[ld].begin(); it != neighbor[ld].end(); ++it) {
          if (fsign(Point::crossProd(ptL, ptR, p[it->v])) > 0 &&
              (!~ch || Circle::side(ptL, ptR, p[ch], p[it->v]) < 0)) {
            ch = it->v;
            side = 0;
          }
        }
        for (auto it = neighbor[rd].begin(); it != neighbor[rd].end(); ++it) {
          if (fsign(Point::crossProd(ptR, p[it->v], ptL)) > 0 &&
              (!~ch || Circle::side(ptL, ptR, p[ch], p[it->v]) < 0)) {
            ch = it->v;
            side = 1;
          }
        }
        if (!~ch) break;
        assert(side == 0 || side == 1);
        if (!side) {
          for (auto it = neighbor[ld].begin(); it != neighbor[ld].end();) {
```

```cpp
            if (Segment(ptL, p[it->v]).hasIntersection(Segment(ptR, p[ch]))) {
              neighbor[it->v].erase(it->rev);
              neighbor[ld].erase(it++);
            } else {
              ++it;
            }
          }
          ld = ch;
        } else {
          for (auto it = neighbor[rd].begin(); it != neighbor[rd].end();) {
            if (Segment(ptR, p[it->v]).hasIntersection(Segment(ptL, p[ch]))) {
              neighbor[it->v].erase(it->rev);
              neighbor[rd].erase(it++);
            } else {
              ++it;
            }
          }
          rd = ch;
        }
      }
    };

    divide(0, p.size() - 1);

    std::vector<std::pair<int, int>> edges;
    for (int u = 0; u < p.size(); ++u)
      for (auto e : neighbor[u])
        if (u < e.v) edges.emplace_back(u, e.v);
    return edges;
  }

  // should be guaranteed that p is strictly convex
  static std::vector<std::pair<int, int>> furthest(
      const std::vector<Point>& p) {
    assert(Polygon::isConvexInCCW(p));
    std::vector<std::pair<int, int>> edges;
    if (p.size() < 3) {
      for (int i = 0; i < p.size(); ++i)
        for (int j = 0; j < i; ++j) {
          edges.emplace_back(i, j);
        }
      return edges;
    }

    std::vector<std::list<Edge>> neighbor(p.size());
    std::vector<int> ids(p.size());
    std::iota(ids.begin(), ids.end(), 0);

    // calculate cw, ccw
    std::vector<int> cw(p.size()), ccw(p.size());
    for (int i = 0; i < p.size(); ++i) {
      cw[i] = (i + p.size() - 1) % p.size();
      ccw[i] = (i + 1) % p.size();
    }
    std::random_shuffle(ids.begin(), ids.end());
    for (int i = ids.size() - 1; i >= 2; --i) {
      int u = ids[i];
      std::tie(ccw[cw[u]], cw[ccw[u]]) = std::make_pair(ccw[u], cw[u]);
    }

    std::vector<std::list<Edge>> lines(p.size());
    auto bind_rev_edge = [&](std::list<Edge>::iterator lhs,
                             std::list<Edge>::iterator rhs) {
      lhs->rev = rhs;
      rhs->rev = lhs;
    };

    bind_rev_edge(neighbor[ids[0]].emplace(neighbor[ids[0]].begin(), ids[1]),
                  neighbor[ids[1]].emplace(neighbor[ids[1]].begin(), ids[0]));

    for (int i = 2; i < ids.size(); ++i) {
      int u = ids[i];
      int cur = ccw[u];
      auto cur_iter = neighbor[cur].begin();
      while (1) {
        while (cur_iter != neighbor[cur].end()) {
          auto next_iter = std::next(cur_iter);
          if (next_iter == neighbor[cur].end()) {
            if (cur != cw[u]) break;
          } else {
            if (Circle::side(p[cur], p[cur_iter->v], p[next_iter->v], p[u]) < 0)
              break;
          }
          neighbor[cur].erase(cur_iter++);
        }
        bind_rev_edge(neighbor[u].emplace(neighbor[u].begin(), cur),
                      neighbor[cur].emplace(cur_iter, u));
        if (cur == cw[u]) break;
        std::tie(cur, cur_iter) =
            std::make_pair(cur_iter->v, std::next(cur_iter->rev));
      }
    }

    for (int u = 0; u < p.size(); ++u)
      for (auto e : neighbor[u])
        if (u < e.v) edges.emplace_back(u, e.v);
    return edges;
  }
};

struct PlanarGraphDuality {
  struct DirectionalEdge {
    int v, id = -1;
    Point direction;
```

```
    int rev;
    DirectionalEdge(int v, Point direction) : v(v), direction(direction) {}
  };

  // return all points' id in each faces and the edges between faces
  static std::pair<std::vector<std::vector<int>>,
                   std::vector<std::pair<int, int>>>
  solve(const std::vector<Point>& p,
        const std::vector<std::pair<int, int>>& edges) {
    std::vector<DirectionalEdge> directional_edges;
    directional_edges.reserve(edges.size() * 2);
    std::vector<std::vector<int>> out_edges(p.size());
    for (auto [u, v] : edges) {
      out_edges[u].push_back(directional_edges.size());
      directional_edges.emplace_back(v, p[v] - p[u]);
      out_edges[v].push_back(directional_edges.size());
      directional_edges.emplace_back(u, p[u] - p[v]);
      directional_edges.end()[-1].rev = out_edges[u].back();
      directional_edges.end()[-2].rev = out_edges[v].back();
    }
    const auto comp = [&, t_comp = Point::PolarComparer()](int lhs, int rhs) {
      return t_comp(directional_edges[lhs].direction,
                    directional_edges[rhs].direction);
    };
    for (int u = 0; u < p.size(); ++u)
      std::sort(out_edges[u].begin(), out_edges[u].end(), comp);
    std::vector<std::vector<int>> faces;
    for (int u = 0; u < p.size(); ++u) {
      for (int e_id : out_edges[u]) {
        if (~directional_edges[e_id].id) continue;
        std::vector<int> pids;
        for (int cur_e_id = e_id;;) {
          if (~directional_edges[cur_e_id].id) break;
          directional_edges[cur_e_id].id = faces.size();
          int v = directional_edges[cur_e_id].v;
          pids.push_back(v);
          auto it = std::lower_bound(out_edges[v].begin(), out_edges[v].end(),
                                     directional_edges[cur_e_id].rev, comp);
          assert(*it == directional_edges[cur_e_id].rev);
          if (it == out_edges[v].begin())
            cur_e_id = out_edges[v].back();
          else
            cur_e_id = *std::prev(it);
        }
        faces.push_back(pids);
      }
    }
    std::vector<std::pair<int, int>> face_edges;
    for (int u = 0; u < p.size(); ++u)
      for (int e_id : out_edges[u]) {
        int rev_id = directional_edges[e_id].rev;
        if (e_id < rev_id) continue;
```

```
        face_edges.emplace_back(directional_edges[e_id].id,
                                directional_edges[rev_id].id);
      }
    return std::make_pair(faces, face_edges);
  }
};

struct Voronoi {
  static constexpr long double kBoundaryInf = 50000;

  // should be guaranteed that
  // 1. all points are pairwise distinct
  // 2. boundary had better to be a convex
  // 3. all points are inside boundary
  static std::vector<std::vector<Point>> nearest(
      const std::vector<Point>& p,
      const std::vector<Line>& boundary = {
          Line(Point(-kBoundaryInf, -kBoundaryInf), Point(1, 0)),
          Line(Point(kBoundaryInf, -kBoundaryInf), Point(0, 1)),
          Line(Point(kBoundaryInf, kBoundaryInf), Point(-1, 0)),
          Line(Point(-kBoundaryInf, kBoundaryInf), Point(0, -1)),
      }) {
    // p0 in the left
    auto bisector = [&](const Point& p0, const Point& p1) {
      auto dir = (p1 - p0).rotate90();
      auto mid = (p0 + p1) / 2;
      return Line(mid, dir);
    };
    auto edges = Triangulation::nearest(p);
    std::vector<std::vector<Line>> limit(p.size(), boundary);
    for (auto [i, j] : edges) {
      limit[i].push_back(bisector(p[i], p[j]));
      limit[j].push_back(bisector(p[j], p[i]));
    }
    std::vector<std::vector<Point>> regions(p.size());
    for (int i = 0; i < p.size(); ++i) {
      regions[i] = HalfPlaneIntersection::solve(limit[i]);
    }
    return regions;
  }

  // should be guaranteed that
  // 1. p is strictly convex
  // 2. boundary had better to be a convex
  // 3. all points are inside boundary
  static std::vector<std::vector<Point>> furthest(
      const std::vector<Point>& p,
      const std::vector<Line>& boundary = {
          Line(Point(-kBoundaryInf, -kBoundaryInf), Point(1, 0)),
          Line(Point(kBoundaryInf, -kBoundaryInf), Point(0, 1)),
          Line(Point(kBoundaryInf, kBoundaryInf), Point(-1, 0)),
          Line(Point(-kBoundaryInf, kBoundaryInf), Point(0, -1)),
```

```cpp
      }) {
    // p0 in the right
    auto bisector = [&](const Point& p0, const Point& p1) {
      auto dir = (p0 - p1).rotate90();
      auto mid = (p0 + p1) / 2;
      return Line(mid, dir);
    };
    auto edges = Triangulation::furthest(p);
    std::vector<std::vector<Line>> limit(p.size(), boundary);
    for (auto [i, j] : edges) {
      limit[i].push_back(bisector(p[i], p[j]));
      limit[j].push_back(bisector(p[j], p[i]));
    }
    std::vector<std::vector<Point>> regions(p.size());
    for (int i = 0; i < p.size(); ++i) {
      regions[i] = HalfPlaneIntersection::solve(limit[i]);
    }
    return regions;
  }
};

}  // namespace geometry2d

//****************************** 圆的面积并 ******************

  // Area[i] 表示覆盖次数大于等于i的面积, 复杂度 O(n^2logn)
struct P {
  ouble x, y;
  (){}(double _x, double _y) { x = _x, y = _y; }
  operator+(const P& b) const { return P(x + b.x, y + b.y); }
  operator-(const P& b) const { return P(x - b.x, y - b.y); }
  operator*(double b) const { return P(x * b, y * b); }
  operator/(double b) const { return P(x / b, y / b); }
  double det(const P& b) const { return x * b.y - y * b.x; }
  P rot90() const { return P(-y, x); }
  P unit() { return *this / abs(); }
  double abs() { return hypot(x, y); }
};
struct Circle {
  P o;
  double r;
  bool contain(const Circle& v, const int& c) const { return sgn(r - (o - v.o).abs() - v.r) > c; }
  bool disjuct(const Circle& v, const int& c) const {  // 0严格, -1不严格
    return sgn((o - v.o).abs() - r - v.r) > c;
  }
};
//求圆与圆的交点, 包含相切, 假设无重圆
bool isCC(Circle a, Circle b, P& p1, P& p2) {
  if (a.contain(b, 0) || b.contain(a, 0) || a.disjuct(b, 0)) return 0;
  double s1 = (a.o - b.o).abs();
  double s2 = (a.r * a.r - b.r * b.r) / s1;
  double aa = (s1 + s2) / 2, bb = (s1 - s2) / 2;
  P mm = (b.o - a.o) * (aa / (aa + bb)) + a.o;
  double h = sqrt(max(0.0, a.r * a.r - aa * aa));
  P vv = (b.o - a.o).unit().rot90() * h;
  p1 = mm + vv, p2 = mm - vv;
  return 1;
}
struct EV {
  P p;
  double ang;
  int add;
  EV() {}
  EV(const P& _p, double _ang, int _add) { p = _p, ang = _ang, add = _add; }
  bool operator<(const EV& a) const { return ang < a.ang; }
} eve[N * 2];
int E, cnt, C, i, j;
Circle c[N];
bool g[N][N], overlap[N][N];
double Area[N];
int cX[N], cY[N], cR[N];
bool contain(int i, int j) {
  return (sgn(c[i].r - c[j].r) > 0 || sgn(c[i].r - c[j].r) == 0 && i < j) && c[i].contain(c[j], -1);
}
int main() {
  scanf("%d", &C);
  for (i = 0; i < C; i++) {
    scanf("%d%d%d", &cX[i], &cY[i], &cR[i]);
    c[i].o = P(cX[i], cY[i]);
    c[i].r = cR[i];
  }
  for (i = 0; i <= C; i++) Area[i] = 0;
  for (i = 0; i < C; i++)
    for (j = 0; j < C; j++) overlap[i][j] = contain(i, j);
  for (i = 0; i < C; i++)
    for (j = 0; j < C; j++) g[i][j] = !(overlap[i][j] || overlap[j][i] || c[i].disjuct(c[j], -1));
  for (i = 0; i < C; i++) {
    E = 0;
    cnt = 1;
    for (j = 0; j < C; j++)
      if (j != i && overlap[j][i]) cnt++;
    for (j = 0; j < C; j++)
      if (i != j && g[i][j]) {
        P aa, bb;
        isCC(c[i], c[j], aa, bb);
        double A = atan2(aa.y - c[i].o.y, aa.x - c[i].o.x);
        double B = atan2(bb.y - c[i].o.y, bb.x - c[i].o.x);
        eve[E++] = EV(bb, B, 1);
        eve[E++] = EV(aa, A, -1);
        if (B > A) cnt++;
      }
    if (E == 0)
      Area[cnt] += PI * c[i].r * c[i].r;
    else {
```

```
        sort(eve, eve + E);
        eve[E] = eve[0];
        for (j = 0; j < E; j++) {
          cnt += eve[j].add;
          Area[cnt] += eve[j].p.det(eve[j + 1].p) * 0.5;
          double theta = eve[j + 1].ang - eve[j].ang;
          if (theta < 0) theta += PI * 2;
          Area[cnt] += theta * c[i].r * c[i].r * 0.5 - sin(theta) * c[i].r * c[i].r * 0.5;
        }
      }
    }
    for (i = 1; i <= C; i++) printf("%d %.3f\n", i, Area[i] - Area[i + 1]);
}
```

4 Math and Number Theory

杨表 标准杨表钩长公式：$\frac{n!}{\prod hook(i,j)}$，其中 $hook(i,j) = |(i,j)\text{下方}| + |(i,j)\text{右方}| + \text{自己}$；

半标准杨表（非严格递增）钩长公式：$\prod_{(i,j)} \frac{n+j-i}{hook(i,j)}$

不交的 k 个上升子序列长度之和最大 $\Leftrightarrow$ 上升杨表前 k 层长度和（不降子序列 $\Leftrightarrow$ 半标准杨表）。序列依次插入杨表，若能放第 1 行末尾则放，否则取第 1 行大于当前数的最小的一个，取出，插入第 2 行。

约数个数表

n	10	1e2	1e3	1e4	1e5	1e6
$d(n)$	4	12	32	64	128	240
n	1e7	1e8	1e9	1e10	1e11	1e12
$d(n)$	448	768	1344	2304	4032	6720
n	1e13	1e14	1e15	1e16	1e17	1e18
$d(n)$	10752	17280	26880	41472	64512	103680

整数拆分数

n	10	20	30	40	50
$p(n)$	42	627	5604	37338	204226
n	60	70	80	90	100
$p(n)$	966467	4087968	15796476	56634173	190569292

整数拆分 五边形数：$\phi(x) = \prod_{i=1}^{\infty}(1-x^i) = 1 + \sum_{i=1}^{\infty}(-1)^i x^{\frac{i(3i\pm1)}{2}}$

整数拆分：$F(x) = \prod_{i=1}^{\infty}(1 + x^i + x^{2i} + \cdots) = \prod_{i=1}^{\infty} \frac{1}{1-x^i}$

$\phi(x)$ 与 $F(x)$ 互为逆多项式

递推：$F(n) = F(n-1) + F(n-2) - F(n-5) - F(n-7) \cdots$

单位根反演 $[i \bmod k = 0] = \frac{1}{k}\sum_{j=0}^{k-1}\omega^{ij}$，$\omega$ 为 k 次单位根

二项式反演 $a_n = \sum_{i=0}^{n}\binom{n}{i}b_i \Rightarrow b_n = \sum_{i=0}^{n}\binom{n}{i}(-1)^{n-i}a_i$

Fibonacci 通项：$f_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$

1. 相邻两项互质
2. $f_{m+n} = f_{m-1}f_n + f_m f_{n+1}$
3. $(f_{m+n}, f_n) = (f_m, f_n)$，$(f_n, f_m) = f_{(n,m)}$
4. $\sum_{i=0}^{n} f_i = f_{n+2} - 1$，$\sum_{i=0}^{n} f_i^2 = f_n f_{n+1}$
5. $f_n^2 = (-1)^{n-1} + f_{n-1}f_{n+1}$

斯特林数 第一类：将 p 个物品排成 k 个非空循环排列

$S(p,k) = (p-1)S(p-1,k) + S(p-1,k-1)$

$S(p,0) = 0,\ S(p,p) = 1,\ S(0,0) = 1$

第二类：将 p 个物品划分成 k 个非空集合（无编号盒子）

$S(p,k) = kS(p-1,k) + S(p-1,k-1)$

$S(p,0) = 0,\ S(p,p) = 1,\ S(0,0) = 1$

$$S(n,m) = \frac{1}{m!}\sum_{k=0}^{m}(-1)^k\binom{m}{k}(m-k)^n = \sum_{k=0}^{m}\frac{(-1)^k(m-k)^n}{k!(m-k)!}$$

$$x^k = \sum_{i=1}^{k} S(k,i)\cdot i! \cdot C(x,i)$$

二次剩余 ① $x^2 \equiv n \pmod p$

有解 iff $n^{\frac{p-1}{2}} \equiv 1 \pmod p$

解：随机 a 使得 $w = a^2 - n$ 非二次剩余，则 $x = \pm(a + \sqrt{w})^{\frac{p+1}{2}}$

② $ax^2 + bx \equiv c \pmod p$

解：配方得 $(2ax+b)^2 \equiv b^2 - 4ac \pmod p$，解 +exgcd

线性逆元 $i^{-1} = -\lfloor \frac{p}{i} \rfloor \cdot (p \pmod i)^{-1}$

powerful number 定义：所含质因子次数全部大于 1 的数，只有 $\sqrt{n}$ 个，可以暴力求。

积性函数求和 $\sum_{i=1}^{n} F(i)$，找一个积性函数 G 使得 $G(p) = F(p)$，则 $H = \frac{F}{G}$ 只在 powerful number 下有非 0 值。

$$\sum_{i=1}^{n} F(i) = \sum_{i=1}^{n} H(i) \sum_{j=1}^{\lfloor \frac{n}{i} \rfloor} G(j) = \sum_{i \in \text{powerful num}} H(i) S_G(\lfloor \frac{n}{i} \rfloor)$$

min25 数组 $g(n,j) = \sum_{i=1}^{n} [i \in P \vee minp_i > p_j] f(i)$

$S(n,j) = \sum_{i=1}^{n} [minp_i \geq p_j] f(i)$

Sherman-Morrison

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

$$|A| \cdot A^{-1} = adjA, (adjA\text{是代数余子式矩阵的转置})$$

新数论分块 $(\lfloor \frac{n}{i} \rfloor, \lceil \frac{i}{n\%i} \rceil)$ 的有效取值只有 $O(\sqrt{n} \ln n)$

```
for (LL i=1,a,j;i<=n;i=j+1)     {
j=n/(a=n/i);
  for (LL l=1,b,r;l<=j;l=r+1)   {
    if (n%l==0) {
      assert(l==j);//当x/n时一定出现在整除分块的末尾处。
      //....
    } else      {
      // cell(r/(n-ar))<=b -> r/(n-ar)<=b -> r<=b*(n-ar) -> r<=bn/(ab+1)
      b=l/(n%l);
        r=b*n/(a*b+1);
        //.....
      }
    }
  }
```

```
//***************************** combinatorics mod (exLucas) *************************

LL p[maxp],p0,pk[maxp],num[maxp],fac[maxp];
void Prime(LL P) {} // P = p[1]^num[1] * ... * p[p0]^num[p0] = pk[1] * ... * pk[p0]
void Pre() {
    fo(j,1,p0) {
        fac[j]=1;
        fo(i,1,pk[j]-1) if (i%p[j]) fac[j]=fac[j]*i%pk[j];
    }
}

LL mo;
LL Pow(LL x,LL y) {} // mod mo
LL count(LL n,LL p) {return (n) ?(n/p+count(n/p,p)) :0 ;}
LL Fc(LL n,LL j) {
    if (!n) return 1;
    LL re=Fc(n/p[j],j) *mi(fac[j],n/mo)%mo;
    fo(i,1,n%mo) if (i%p[j]) re=re*i%mo;
    return re;
}

LL C(LL n,LL m,LL p) {          // compute C(n+m,n). M=n+m.
    // preprocess: Prime(P); Pre();
    LL ans=0;
    fo(j,1,p0) {
        LL nump=count(M,p[j])-count(m,p[j])-count(n,p[j]);
        LL phi=pk[j]-pk[j]/p[j];
        mo=pk[j];
        LL a=(nump>=num[j]) ?0 :Fc(M,j)*Pow(Fc(n,j),phi-1)%mo*Pow(Fc(m,j),phi-1)%mo*Pow(p[j],nump)%mo
            ;
        (ans+=a *(P/pk[j])%P *Pow(P/pk[j],phi-1)%P)%=P;
    }

    return ans;
}

//***************************** Miller_Rabin ******************************

int pr[9]={2,3,5,7,11,13,17,19,23};
LL mul(LL x,LL y,LL mo) {
    LL re=0;
    for(; y; y>>=1, x=(x+x)%mo) if (y&1) re=(re+x)%mo;
    return re;
}
LL Pow(LL x,LL y,LL mo) {
```

```cpp
    LL re=1;
    for(; y; y>>=1, x=mul(x,x,mo)) if (y&1) re=mul(re,x,mo);
    return re;
}
bool Miller_Rabin(int d,LL s,LL a,LL n) {
    a=Pow(a,s,n);
    if (a==1) return 1;
    fo(i,1,d) {
        if (a==n-1) return 1;
        if (a==1) return 0;
        a=mul(a,a,n);
    }
    return 0;
}
bool isprime(LL n) {
    if (n<2) return 0;
    fo(i,0,8) {
        if (n==pr[i]) return 1;
        if (n%pr[i]==0) return 0;
    }
    int d=0; LL s=n-1;
    for(; !(s&1); s>>=1, d++);
    fo(i,0,8) if (!Miller_Rabin(d,s,pr[i],n)) return 0;
    return 1;
}

//****************************** pollard_rho ******************************

inline LL ran_f(LL x,LL c,LL n) {return (mul(x,x,n)+c)%n;}
LL pollard_rho(LL n) {
    for(LL c=rand()*rand()%n; ; c=rand()*rand()%n) {
        LL x=rand()*rand()%n, y=x;
        for(LL i=0, k=1; ; i++) {
            x=ran_f(x,c,n);
            LL t=__gcd(abs(x-y),n);
            if (t==n) break;
                else if (t>1) return t;
            if (i==k) y=x, k<<=1;
        }
    }
}

//****************************** similar gcd ******************************

struct FGH{
    LL f,g,h;
};  // f=\sum_{i=0}^n (a*i+b)/c, g=\sum_{i=0}^n i*(a*i+b)/c, h=\sum_{i=0}^n ((a*i+b)/c)^2

FGH calc(LL a,LL b,LL c,LL n) {
    LL ac=a/c, bc=b/c, sum1=n*(n+1)%mo*inv2%mo, sum2=n*(n+1)%mo*(2*n+1)%mo*inv6%mo;
    if (!a) return (FGH){(n+1)*bc%mo, sum1*bc%mo, (n+1)*bc%mo*bc%mo};
    if (a>=c || b>=c) {
        FGH nxt=calc(a%c,b%c,c,n);
        LL f=(nxt.f+ac*sum1+(n+1)*bc)%mo;
        LL g=(nxt.g+ac*sum2+sum1*bc)%mo;
        LL h=(nxt.h+sum2*ac%mo*ac%mo+(n+1)*bc%mo*bc%mo+2*ac*nxt.g%mo+2*bc*nxt.f%mo+n*(n+1)%mo*ac%mo*
            bc%mo)%mo;
        return (FGH){f,g,h};
    } else {
        LL m=(a*n+b)/c;
        FGH nxt=calc(c,c-b-1,a,m-1);
        m%=mo;
        LL f=(m*n-nxt.f+mo)%mo;
        LL g=((n+1)*n%mo*m-nxt.f-nxt.h+mo+mo)%mo*inv2%mo;
        LL h=((m+1)*n%mo*m-nxt.g-nxt.g-f-nxt.f-nxt.f+mo*5)%mo;
        return (FGH){f,g,h};
    }
}


//****************************** 筛 ****************************************

// f(n,k) 表示把 n 拆成 k 个数的积的方案数
LL mw[2*maxsqrtn],g[2*maxsqrtn];
int w0,id1[maxsqrtn],id2[maxsqrtn];
LL min25_g(LL n) {
    w0=0;
    for(LL i=1, j; i<=n; i=j+1) {
        j=n/(n/i);
        mw[++w0]=n/i;
        if (mw[w0]<=sqrtn) id1[mw[w0]]=w0; else id2[j]=w0;
        g[w0]=mw[w0]-1;
    }
    fo(j,1,Np[sqrtn])
        for(int i=1; i<=w0 && (LL)p[j]*p[j]<=mw[i]; i++) {
            int id=(mw[i]/p[j]<=sqrtn) ?id1[mw[i]/p[j]] :id2[n/(mw[i]/p[j])];
            (g[i]-=g[id]-(j-1))%=mo;
        }
}
LL min25_S(LL x,int j,int k) {
    if (x<=1 || p[j]>x) return 0;
    int id=(x<=sqrtn) ?id1[x] :id2[n/x];
    LL re=(g[id]-(j-1))*k;
    for(int i=j; i<=Np[sqrtn] && (LL)p[i]*p[i]<=x; i++) {
        LL pe=p[i];
        for(int e=1; pe*p[i]<=x; e++, pe*=p[i])
            (re+=min25_S(x/pe,i+1,k)*C[e+k-1][k-1]+C[e+k][k-1])%=mo;
    }
    return re;
}
```

5 Others

有向图最大费用循环流 思路：先让正权边全部流满，再调整做法：

1. 先强制流掉所有正权边（并加入答案）
2. 然后流量盈余的点连向汇点，源点连向流量亏损的点
3. 答案减去 MCMF

无源汇上下界网络流 记 d_i 为点 i 的入流减出流，附加网络如下：

1. 原图每条边 (u, v, l, r)，连边 $(u, v, r - l)$
2. 若 $d_i > 0$，则 (ss, i, d_i)
3. 若 $d_i < 0$，则 $(i, tt, -d_i)$
4. 当且仅当 ss, tt 满流时有可行解

最大权闭合子图 答案 = 所有正点权的和 − 如下最小割：

1. 原图每条边 (u, v)，连边 (u, v, inf)
2. 超级源连向正点权，负点权连向超级汇

Segment Tree Beats 维护区间最小值、区间次小值以及最小值出现的次数；将操作 (x, y) 抽象为 $a_i := min(a_i + x, y)$；若 TL 允许，写成矩阵形式转移最简单。

BEST 定理 图 G 从点 s 出发的欧拉路径数 $= ts(G) \prod (deg(v) - 1)!$，其中 $ts(G)$ 表示以 s 为根的外向树个数（入度矩阵 − 邻接矩阵，求 det）。前提：每个点出度入度相同。

```
//***************************** Blossom ***********************************

int ga[maxp];
int get(int x) {return (ga[x]==x) ?x :ga[x]=get(ga[x]) ;}

int d[maxp],di,dj,sum,pt[maxp],pf[maxp],clr[maxp],nowT,bz[maxp];
int lca(int x,int y) {
	for(nowT++, x=get(x), y=get(y); bz[x]!=nowT; ) {
		bz[x]=nowT;
		x=get(pf[pt[x]]);
		swap(x,y);
	}
	return x;
}
void shrink(int x,int y,int rt) {
	for(; get(x)!=rt; x=pf[y]) {
		pf[x]=y;
		y=pt[x];
		if (clr[y]==1) clr[ d[++dj]=y ]=0;
		ga[x]=ga[y]=rt;
	}
}
bool Blossom(int st) {
	fo(i,1,sum) ga[i]=i;
	memset(clr,255,sizeof(clr)); clr[st]=0;
	d[1]=st;
	for(di=1, dj=1; di<=dj; di++) {
		int now=d[di];
		for(int p=f1[now]; p; p=next[p]) if (clr[go[p]]==-1) {
			clr[go[p]]=1;
			pf[go[p]]=now;
			if (!pt[go[p]]) {
				for(int x=now, y=go[p], t; x; y=t, x=pf[y]) {
					t=pt[x];
					pt[x]=y, pt[y]=x;
				}
				return 1;
			} else {
				d[++dj]=pt[go[p]];
				clr[pt[go[p]]]=0;
			}
		} else if (clr[go[p]]==0 && get(go[p])!=get(now)) {
			int rt=lca(go[p],now);
			shrink(go[p],now,rt);
			shrink(now,go[p],rt);
		}
	}
	return 0;
}

//***************************** dominator tree (from CTL) *****************

const int maxn = 310000;
const int maxm = 1050000;

int n,m,s;
int sdom[maxn],idom[maxn];
```

```
vector<int>V[maxn];
vector<int>g[maxn],e[maxn];

int dfn[maxn],To[maxn],id,par[maxn];
void build(int u){
    To[dfn[u]=++id]=u;
    for (auto v:g[u]) if (!dfn[v]) par[v]=u,build(v);
}

int fa[maxn],fas[maxn];
void find(int x)
{
    if(fa[x]==x) return;
    find(fa[x]);
    if(dfn[sdom[fas[fa[x]]]]<dfn[sdom[fas[x]]]) fas[x]=fas[fa[x]];
    fa[x]=fa[fa[x]];
}
int ans[maxn];

int main(){
    scanf("%d%d%d",&n,&m,&s);
    int nn=n;
    rep(i,1,m) {
        int x,y;
        scanf("%d%d",&x,&y);
        e[n+i].push_back(x); e[y].push_back(n+i);
        g[x].push_back(n+i); g[n+i].push_back(y);
    }
    n+=m;
    build(s);
    rep(i,1,n) fa[i]=i,fas[i]=i,sdom[i]=idom[i]=i;
    per(i,id,1){
        int x=To[i],&semi=sdom[x];
        for(auto y:e[x]) if (dfn[y]){
            find(y);
            if(dfn[semi]>dfn[sdom[fas[y]]]) semi=sdom[fas[y]];
        }
        for(auto y:V[x]){
            find(y);
            if(dfn[sdom[fas[y]]]<i) idom[y]=fas[y];
            else idom[y]=x;
        }
        V[semi].push_back(x);
        for (auto y:g[x]) if (par[y]==x) fa[y]=x;
    }
    rep(i,1,id){
        int x=To[i];
        if(idom[x]!=sdom[x]) idom[x]=idom[idom[x]];
    }
    per(i,id,2){
        int x=To[i];
        if (1<=x&&x<=n&&idom[x]>nn) ans[idom[x]-nn]=1;
    }
    int cnt=0;
    rep(i,1,m) if (!ans[i]) cnt++;
    printf("%d\n",cnt);
    rep(i,1,m) if (!ans[i]) printf("%d ",i);
    return 0;
}

//****************************** dq_mincut *********************************

int num,st[maxn];
bool vis[maxn];
void find(int k)
{
    vis[k]=1;
    st[++num]=k;
    for(int p=f1[k]; p; p=nxt[p]) if (!vis[go[p]]) find(go[p]);
}

int bj[maxn],sum,nowh[maxn],d[maxn],bz[maxn],bzcnt;
// maxflow here, 稀疏图 dinic, else isap
void bfs(int s)
{
    bz[ d[1]=s ]=++bzcnt;
    for(int i=1, j=1; i<=j; i++)
    {
        for(int p=f1[d[i]]; p; p=nxt[p]) if (val2[p] && bz[go[p]]!=bzcnt)
            bz[ d[++j]=go[p] ]=bzcnt;
    }
}

int st1[maxn];
vector<pair<int,int>> e[maxn];
void Mincut(int l,int r)
{
    memcpy(val2,val,sizeof(val));
    sum=st[r];

    int flow=0;
    while (Dinic_bfs(st[l])) flow+=Dinic_dfs(st[l],inf);
    e[st[l]].push_back(make_pair(sum,flow)), e[sum].push_back(make_pair(st[l],flow));

    bfs(st[l]);
    int newr=l-1, newl=r+1;
    fo(i,l,r) if (bz[st[i]]==bzcnt) st1[++newr]=st[i]; else st1[--newl]=st[i];
    fo(i,l,r) st[i]=st1[i];

    if (l<newr) Mincut(l,newr);
    if (newl<r) Mincut(newl,r);
}

int main()
```

```
{
    fo(i,1,n) if (!vis[i] && f1[i])
    {
        num=0;
        find(i);
        Mincut(1,num);
    }
}

//****************************** KM ****************************************

LL lx[maxn],ly[maxn],slack[maxn];
int f[maxn],pre[maxn];
bool vis[maxn];
LL KM(int nl,int nr)
{
    fo(i,1,nl)
        fo(j,1,nr) lx[i]=max(lx[i],mp[i][j]);
    fo(i,1,nl)
    {
        memset(slack,127,sizeof(LL)*(nr+1));
        memset(vis,0,sizeof(bool)*(nr+1));
        f[0]=i;
        int py=0, nextpy;
        for(; f[py]; py=nextpy)
        {
            int px=f[py];
            LL d=inf;
            vis[py]=1;
            fo(j,1,nr) if (!vis[j])
            {
                if (lx[px]+ly[j]-mp[px][j]<slack[j]) slack[j]=lx[px]+ly[j]-mp[px][j], pre[j]=py;
                if (slack[j]<d) d=slack[j], nextpy=j;
            }
            fo(j,0,nr) if (vis[j]) lx[f[j]]-=d, ly[j]+=d;
                else slack[j]-=d;
        }
        for(; py; py=pre[py]) f[py]=f[pre[py]];
    }
    LL re=0;
    fo(i,1,nl) re+=lx[i];
    fo(j,1,nr) re+=ly[j];
    return re;
}

//****************************** LCA ****************************************

//倍增
int deep[maxn],fa[maxn][MX+1];
int lca(int x,int y) {
    if (deep[x]<deep[y]) swap(x,y);
    fd(i,MX,0)
        while (deep[fa[x][i]]>=deep[y]) x=fa[x][i];
    if (x==y) return x;
    fd(i,MX,0)
        while (fa[x][i]!=fa[y][i]) x=fa[x][i], y=fa[y][i];
    return fa[x][0];
}

//tarjan
int totq,goq[2*maxm],num[2*maxm],nextq[2*maxm],fq[maxn];
void inq(int x,int y,int z) {
    goq[++totq]=y;
    num[totq]=z;
    nextq[totq]=fq[x];
    fq[x]=totq;
}
int lca[maxm],fa[maxn];
bool bz[maxn];
int get(int x) {
    if (fa[x]==x) return x;
    return fa[x]=get(fa[x]);
}
void tarjan(int k,int last) { //ordinary
    fa[k]=k;
    for(int p=f1[k]; p; p=next[p]) if (go[p]!=last) {
        tarjan(go[p],k);
        fa[go[p]]=k;
    }
    bz[k]=1;
    for(int p=fq[k]; p; p=nextq[p])
        if (bz[goq[p]]) lca[num[p]]=get(goq[p]); else inq(goq[p],k,num[p]);
}

void tarjan(int k,int last) { //支持维护值
    fa[k]=k;
    for(int p=f1[k]; p; p=next[p]) if (go[p]!=last) {
        tarjan(go[p],k);
        f[go[p]]+=val[p];
        fa[go[p]]=k;
    }
    bz[k]=1;
    for(int p=fq[k]; p; p=nextq[p]) if (bz[goq[p]]) {
        int t=get(goq[p]);
        ans[num[p]]=valq[p]+f[goq[p]];
        if (t!=k) inq(t,k,f[goq[p]],num[p]); else lca[num[p]]=t;
    } else inq(goq[p],k,0,num[p]);
}

//rmq
int fa[2*maxn][MX+5],deep[maxn],ap[2*maxn],fir[2*maxn],Log[2*maxn],er[MX+5];
void rmq_pre() {
    fo(i,1,ap[0]) fa[i][0]=ap[i], Log[i]=log(i)/log(2);
    fo(i,0,MX) er[i]=1<<i;
```

```
    fo(j,1,MX)
        fo(i,1,ap[0]) {
            fa[i][j]=fa[i][j-1];
            if (i+er[j-1]<=ap[0] && deep[fa[i+er[j-1]][j-1]]<deep[fa[i][j]])
                fa[i][j]=fa[i+er[j-1]][j-1];
        }
}
int lca(int x,int y) {
    x=fir[x], y=fir[y];
    if (x>y) swap(x,y);
    int t=Log[y-x+1];
    return (deep[fa[x][t]]<deep[fa[y-er[t]+1][t]]) ?fa[x][t] :fa[y-er[t]+1][t] ;
}

void dfs_pre(int k,int last) {
    deep[k]=deep[last]+1;
    ap[++ap[0]]=k, fir[k]=ap[0];
    for(int p=f1[k]; p; p=next[p]) if (go[p]!=last) {
        dfs_pre(go[p],k);
        ap[++ap[0]]=k;
    }
}

//****************************** LeftTree **********************************

struct node{
    int val,l,r,fa,dis;
};

node lt[maxn];
int tot,ga[maxn];
int New(int val=0)
{
    lt[++tot]=(node){val,0,0,0,0};
    ga[tot]=tot;
    return tot;
}
int merge(int a,int b)
{
    if (!a) return b;
    if (!b) return a;
    if (lt[a].val>lt[b].val || lt[a].val==lt[b].val && a>b) swap(a,b);
    lt[a].r=merge(lt[a].r,b);
    lt[lt[a].r].fa=a;
    ga[lt[a].r]=a;
    if (lt[lt[a].r].dis>lt[lt[a].l].dis) swap(lt[a].l,lt[a].r);
    lt[a].dis=(lt[a].r==0) ?0 :lt[lt[a].r].dis+1;
    return a;
}
int top(int x) {return (ga[x]==x) ?x :ga[x]=top(ga[x]) ;}
void pop(int x)
{
```

```
    int t=merge(lt[x].l,lt[x].r);
    lt[t].fa=lt[x].fa;
    ga[x]=(top(x)==x) ?t :top(x);
    ga[t]=ga[x];
    for(int i=lt[x].fa; i; i=lt[i].fa) if (lt[lt[i].l].dis<lt[lt[i].r].dis)
    {
        swap(lt[i].l,lt[i].r);
        lt[i].dis=lt[lt[i].r].dis+1;
    } else break;
    lt[x].fa=-1;
}
void push(int x,int val)
{
    merge(top(x),New(val));
}

// init : lt[0].dis=-1;

//****************************** maxflow **********************************

//isap+gap+当前弧
int bj[maxsum],gap[maxsum],sum,nowh[maxsum],d[maxsum];
void init_Maxflow() {
    memset(bj,0,sizeof(bj));
    memset(gap,0,sizeof(gap)); gap[0]=sum+1;
    memcpy(nowh,f1,sizeof(nowh));
    d[1]=sum;
    for(int i=1, j=1; i<=j; i++) {
        for(int p=f1[d[i]]; p; p=e[p].nxt) if (e[p].go!=sum && !bj[e[p].go]) {
            bj[e[p].go]=bj[d[i]]+1;
            gap[0]--, gap[bj[e[p].go]]++;
            d[++j]=e[p].go;
        }
    }
}
int Maxflow(int k,int flow) {
    if (k==sum) return flow;
    int re=0;
    for(int &p=nowh[k]; p; p=e[p].nxt) if (e[p].val && bj[k]==bj[e[p].go]+1) {
        int fl=Maxflow(e[p].go, (flow-re<e[p].val) ?(flow-re) :e[p].val);
        e[p].val-=fl;
        e[(p&1) ?p+1 :p-1 ].val+=fl;
        re+=fl;
        if (re==flow || bj[0]>sum) return re;
    }
    nowh[k]=f1[k];
    if ((--gap[bj[k]])==0) bj[0]=sum+1; else bj[k]++;
    gap[bj[k]]++;
    return re;
}
//dinic+当前弧
bool Dinic_bfs(int s) {
```

```cpp
    memset(bj,255,sizeof(bj));
    memcpy(nowh,f1,sizeof(nowh));
    bj[ d[1]=s ]=0;
    for(int i=1, j=1; i<=j; i++) {
        for(int p=f1[d[i]]; p; p=e[p].nxt) if (e[p].val && bj[e[p].go]==-1) {
            bj[e[p].go]=bj[d[i]]+1;
            d[++j]=e[p].go;
        }
    }
    return bj[sum]!=-1;
}
int Dinic_dfs(int k,int flow) {
    if (k==sum) return flow;
    int re=0;
    for(int &p=nowh[k]; p; p=e[p].nxt) if (e[p].val && bj[k]+1==bj[e[p].go]) {
        int fl=Dinic_dfs(e[p].go, (flow-re<e[p].val) ?(flow-re) :e[p].val);
        e[p].val-=fl;
        e[(p&1) ?p+1 :p-1 ].val+=fl;
        re+=fl;
        if (re==flow) return re;
    }
    return re;
}

//****************************** random Hung ************************

int bz[maxn],tim,pt[maxn];
bool Hung(int x,int tim) {
    if (bz[x]==tim) return 0;
    bz[x]=tim;
    random_shuffle(e[x].begin(),e[x].end());
    for(int go:e[x]) {
        int k=pt[go];
        pt[k]=0, pt[x]=go, pt[go]=x;
        if (!k || Hung(k,tim)) return 1;
        pt[k]=go, pt[go]=k, pt[x]=0;
    }
    return 0;
}
int main() {
    fo(i,1,n) pmt[i]=i;
    random_shuffle(pmt+1,pmt+1+n);
    int tim=0, cnt=0;
    fo(j,1,5)
        fo(i,1,n) if (!pt[pmt[i]]) Hung(pmt[i],++tim);
}

//****************************** round square tree ************************

//广义（任意路径都是圆方交替）
int sum,dfn[maxn],low[maxn],z[maxn],z0,num[2*maxn],nn;
vector<int> e[2*maxn];
void tarjan(int k,int last) {
    dfn[k]=low[k]=++sum;
    z[++z0]=k;
    for(int p=f1[k]; p; p=nxt[p]) if (bh[p]!=last) {
        if (!dfn[go[p]]) {
            tarjan(go[p],bh[p]);
            low[k]=min(low[k],low[go[p]]);

            if (low[go[p]]>=dfn[k]) {
                num[++nn]=1;
                e[nn].push_back(k), e[k].push_back(nn);
                do {
                    num[nn]++;
                    e[nn].push_back(z[z0]), e[z[z0]].push_back(nn);
                } while (z[z0--]!=go[p]);
            }
        } else low[k]=min(low[k],dfn[go[p]]);
    }
}

//****************************** virtual tree ******************************

int p0,p[2*maxn],z[maxn],z0;
bool cmpP(const int &a,const int &b) {return dfn[a]<dfn[b];}
void make_vtree() {
    tot=0;
    sort(p+1,p+1+p0,cmpP);
    int t=p0;
    fo(i,1,t-1) p[++p0]=lca(p[i],p[i+1]);
    sort(p+1,p+1+p0,cmpP);
    f1[ z[z0=1]=1 ]=0;
    p[0]=1;
    fo(i,1,p0) if (p[i]!=p[i-1]) {
        for(; z0 && (dfn[p[i]]<dfn[z[z0]] || en[z[z0]]<dfn[p[i]]); z0--) ins(z[z0-1],z[z0]);
        f1[ z[++z0]=p[i] ]=0;
    }
    fo(i,1,z0-1) ins(z[i],z[i+1]);
}

//****************************** Tools ************************************

//乘法取模黑科技 Claris
LL mul(LL a,LL b,LL n){return(a*b-(LL)(a/(long double)n*b+1e-3)*n+n)%n;}

//split a string by whitespace
vector<string> split_str(string str) {
    vector<string> result;
    istringstream iss(str);
    string s;
    while ( getline( iss, s, ' ' ) ) result.push_back(s);
    return result;
}
```

```cpp

//***************************** RARMQ 2D *************************************
// 操作 (l1,r1,l2,r2,v) 可视为在时间 l1, 对 [l2,r2]+=v, 在时间 l2+1, 对 [l2,r2]-=v,
// 询问 (l1,r1,l2,r2) 可视为将只考虑时间 [l1,l2], [l2,r2] 上的历史最大值
// 每个询问套在线段树分治区间 [l,r], 分成 [l,mid],[mid+1,r] 两段询问
// 总复杂度 O(nlog^2n+qlogn)
#include<bits/stdc++.h>
#define maxn 500050
using namespace std;
typedef long long LL;

const LL N=65536,M=1e10,inf=1e18;

struct node {
    LL mx,hmx;
    LL tag,htag;
} T[N<<1];
#define mx(k) T[k].mx
#define tag(k) T[k].tag
#define hmx(k) T[k].hmx
#define htag(k) T[k].htag

void build(int k,int l,int r)   {
    mx(k)=hmx(k)=0;
    tag(k)=htag(k)=0;
    if (l==r) return ;
    int mid=(l+r)>>1;
    build(k<<1,l,mid);
    build(k<<1|1,mid+1,r);
}

void renew(int k,LL h,LL d)   {
    h=max(h,0LL);
    htag(k)=max(htag(k),tag(k)+h);
    tag(k)+=d;
    hmx(k)=max(hmx(k),mx(k)+h);
    mx(k)+=d;
}
void godown(int k)  {
    renew(k<<1,htag(k),tag(k));
    renew(k<<1|1,htag(k),tag(k));
    htag(k)=tag(k)=0;
}
void update(int k)  {
    mx(k)=max(mx(k<<1),mx(k<<1|1));
    hmx(k)=max(hmx(k<<1),hmx(k<<1|1));
}

void change(int k,int l,int r,int a,int b,LL d) {
    if (a<=l&&r<=b)
        renew(k,d,d);
    else    {
        godown(k);
        int mid=(l+r)>>1;
        if (a<=mid)
            change(k<<1,l,mid,a,b,d);
        if (b>mid)
            change(k<<1|1,mid+1,r,a,b,d);
        update(k);
    }
}

LL query(int k,int l,int r,int a,int b) {
    if (a>r||l>b) return -inf;
    if (a<=l&&r<=b) return hmx(k);
    godown(k);
    int mid=(l+r)>>1;
    return max(query(k<<1,l,mid,a,b),query(k<<1|1,mid+1,r,a,b));
}

struct seg  {
    int l,r,x;
    bool operator < (const seg& s) const {
        return x<s.x;
    }
};
vector<seg> L[maxn],R[maxn];

int n,m,q;
int Log2(int x) {   return !x?-1:Log2(x>>1)+1;    }
LL ans[maxn];
vector<pair<seg,int> > LQ[maxn],RQ[maxn];
void Max(LL &a,LL b)    {    a=max(a,b); }

int main()  {
    scanf("%d%d%d",&n,&m,&q);
    while (m--) {
        int l1,l2,r1,r2,x;
        scanf("%d%d%d%d%d",&l1,&l2,&r1,&r2,&x);
        L[l1].push_back(seg{l2,r2,x});
        R[r1].push_back(seg{l2,r2,x});
    }
    for (int i=0;i<q;++i)   {
        int l1,l2,r1,r2;
        scanf("%d%d%d%d",&l1,&l2,&r1,&r2);
        int t=max(Log2(l1^r1),0);
        LQ[t].emplace_back(seg{l2,r2,r1},i);
        RQ[t].emplace_back(seg{l2,r2,l1},i);
    }

    for (int t=0;(1<<t)<=N;++t) {
        build(1,0,N-1);
        LL cnt=0,sum=0,ALL=(1<<t)-1;

```

```
        sort(LQ[t].begin(),LQ[t].end());
        for (int i=0,j=0;i<N;++i)   {
            for (seg s:L[i])
                change(1,0,N-1,s.l,s.r,s.x),++cnt;
            for (;j<LQ[t].size()&&LQ[t][j].first.x==i;++j)
                Max(ans[LQ[t][j].second],query(1,0,N-1,LQ[t][j].first.l,LQ[t][j].first.r)-sum*M);
            for (seg s:R[i])
                change(1,0,N-1,s.l,s.r,-s.x),++cnt;
            if (ALL&~i) continue;
            change(1,0,N-1,0,N-1,cnt*M);
            sum+=cnt,cnt=0;
        }

        build(1,0,N-1),sum=cnt=0;
        sort(RQ[t].rbegin(),RQ[t].rend());
        for (int i=N-1,j=0;i>=0;--i)   {
            for (seg s:R[i])
                change(1,0,N-1,s.l,s.r,s.x),++cnt;
            for (;j<RQ[t].size()&&RQ[t][j].first.x==i;++j)
                Max(ans[RQ[t][j].second],query(1,0,N-1,RQ[t][j].first.l,RQ[t][j].first.r)-sum*M);
            for (seg s:L[i])
                change(1,0,N-1,s.l,s.r,-s.x),++cnt;
            if (ALL&i) continue;
            change(1,0,N-1,0,N-1,cnt*M);
            sum+=cnt,cnt=0;
        }
    }
    for (int i=0;i<q;++i) printf("%lld\n",ans[i]);
    return 0;
}
```