Game Over Team Reference - Contents

NUCES Fast Karachi

Huzaifa Rashid, Abdullah Gohar, M. Shaheer Luqman

Contents

1. Misc

1.1. Template.

```
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

#define int long long int
#define ld long double
#define nl cout << "\n";
#define yesno(b) cout << ((b) ? "YES" : "NO");
#define pii pair<int, int>

#define forn(a, b) for (int i = a; i < b; i++)
#define getunique(v){sort(v.begin(), v.end()); v.erase(unique(v.begin(), v.end()),

template <typename T>
using ordered_set = tree<T, null_type, less_equal<T>, rb_tree_tag, tree_order_stat

#define __builtin_popcountll __builtin_popcountll
#define __builtin_clzll __builtin_clzll
#define __builtin_ctzll __builtin_ctzll
```

1.2. istringstream.

```
int main() {
 int test;
 scanf("%d", &test);
 getchar();
 string line;
 for (int i = 0; i < test; i++) {
  getline(cin, line);
  istringstream in(line);

  while (in >> line)
   cout << line << endl;
 }
 return 0;
}
```

1.3. printf scanf.

```
char s[100];

scanf("%[aeiou]", s); // solo lee las vocales

scanf("%[^aeiou]", s); // solo lee las letras

scanf("%[^\n]", z); // funciona igual q gets()

// eliminar rayita de la fecha (5-29-2014) o (5/29/2014)
scanf("%d %*c %d %*c %d", &m, &d, &y);

printf("%09d\n", f); // imprime el entero f y rellena con 9 ceros

printf("%G\n", c); // imprime c sin ceros finales (convierte a E)

printf("%g\n", c); // imprime c sin ceros finales (convierte a e)

printf("%x\n", x); // imprime x en hexadecimal (Letras minusculas)

printf("%X\n", x); // imprime x en hexadecimal (Letras mayusculas)

printf("%o\n", o); // imprime o como octal unsigned

printf("%e\n", cient); // imprime el # en notacion cientifica (e minuscula)

printf("%E\n", cient); // imprime el # en notacion cientifica (E mayuscula)

// muestra un valor de apuntador en forma de puesta en marcha definida
```

```
printf("El␣valor␣de␣Ptr␣es␣%p\n", ptr);

// Almacena el # char almacenados en el printf.
printf("Total␣de␣char␣impresos␣en␣esta␣linea␣es:%n", &cant);
printf("␣%d\n\n", cant);

printf("%%\n"); // muestra el caracter de porciento

printf("\\\n"); //muestra el caracter \

printf("\'\n"); // muestra el caracter '

printf("\"\n"); // muestra el caracter "

printf("\?\n"); // muestra el caracter '

printf("\\n\n"); // muestra el caracter \n

printf("%11d\n", 123); // justifica a la derecha en 11
// 7:ancho del campo 2:preicision, valor 98.74 justificado derecha
printf("%*.*f\n", 7, 2, 98.736);
// if precision < 0 ---> justificado izquierda

/*sprintf*/
char numstr[100];
int num = 1200;

sprintf(numstr, "%d", num); // a decimal
printf("%s\n", numstr);

sprintf(numstr, "%X", num); // a hexadecimal en mayuscula
printf("%s\n", numstr);
//-----------------------------------------------

int base = 8;
cout << setbase(base) << 8 << endl; // pone a cout a imprimir en base
                            // (0,8,10,16)

/* istringstream
```

1.4. Cube.

```
template <class T> struct cube {
 T F, U, D, L, R, B;

 void rotX() {
  swap(D, B);
  swap(B, U);
  swap(U, F);
 } // FUBD -> DFUB

 void rotY() {
  swap(D, R);
  swap(R, U);
  swap(U, L);
 } // LURD -> DLUR

 void rotZ() {
  swap(B, R);
  swap(R, F);
  swap(F, L);
 } // LFRB -> BLFR
};
```

1.5. Josephus.

```
/*
     Tested: ??????
*/

// n-cantidad de personas, m es la longitud del salto.
// comienza en la k-esima persona.
ll josephus(ll n, ll m, ll k) {
 ll x = -1;
 for (ll i = n - k + 1; i <= n; ++i)
  x = (x + m) % i;
 return x;
}

ll josephus_inv(ll n, ll m, ll x) {
 for (ll i = n;; i--) {
  if (x == i)
```

```
    return n - i;
  x = (x - m % i + i) % i;
 }
  return -1;
}
```

1.6. Partition.

```
typedef long long ll;

ll partition(ll n) {
 vector<ll> dp(n + 1);
 dp[0] = 1;
 for (int i = 1; i <= n; i++)
   for (int j = 1, r = 1; i - (3 * j * j - j) / 2 >= 0; j++, r *= -1) {
    dp[i] += dp[i - (3 * j * j - j) / 2] * r;
    if (i - (3 * j * j + j) / 2 >= 0)
     dp[i] += dp[i - (3 * j * j + j) / 2] * r;
   }
 return dp[n];
}
```

1.7. Random.

```
std::default_random_engine generator;
std::uniform_real_distribution<double> distribution(0.0, 1.0);
```

1.8. Useful.

```
// TIME
for (int a = 0;; ++a) {
 if (clock() >= 2.5 * CLOCKS_PER_SEC)
  break;
 // It will stop when 2.5 seconds have passed
}

// LAMBDA
function<bool(int, int)> add_edge = [&](int u, int v) {
 // code here...
 return true;
};

// RANDOM DISTRIBUTIONS
std::default_random_engine generator;
std::uniform_real_distribution<double> distribution(0.0, 1.0);
```

2. BitMask

2.1. Amount of Hamiltonian Walks.

```
/*
      task: Finding the number of Hamiltonian walks in the
                  unweighted and directed graph G = (V, E).
      complexity: O(2^n * n^2)
      notes: Let dp[msk][v] be the amount of Hamiltonian walks
                  on the subgraph generated by vertices in msk that
                  end in the vertex v.
*/
#define BIT(n) (1 << n)
const int MAXN = 20;

int n, m, u, v, g[MAXN], dp[BIT(MAXN)][MAXN], ans;

int main() {
 cin >> n >> m;

 for (int i = 0; i < m; ++i) {
  cin >> u >> v;
  g[u] |= BIT(v);
 }

 for (int i = 0; i < n; ++i)
  dp[BIT(i)][i] = 1;

 for (int msk = 1; msk < BIT(n); ++msk) {
  for (int i = 0; i < n; ++i)
   if (msk & BIT(i)) {
    int tmsk = msk ^ BIT(i);

    for (int j = 0; tmsk && j < n; ++j) {
     if (g[j] & BIT(i))
      dp[msk][i] += dp[tmsk][j];
    }
   }
 }
 for (int i = 0; i < n; ++i)
  ans += dp[BIT(n) - 1][i];
 cout << ans << endl;
 return 0;
}
```

2.2. Existence of Hamiltonian Cycle.

```
/*
      task: Check for existence of Hamiltonian cycle in a
                directed graph G = <V, E>.
      complexity: O(2^n * n)
      notes: Let dp[msk] be the mask of the subset consisting
                of those vertices j such that exist a
  Hamiltonian walk over the subset msk beginning in vertex 0 and ending in j.
*/
#define BIT(n) (1 << n)
const int MAXN = 20;
int n, m, u, v, g[MAXN], dp[BIT(MAXN)];

int main() {
 cin >> n >> m;

 for (int i = 0; i < m; ++i) {
  cin >> u >> v;
  g[v] |= BIT(u);
 }

 dp[1] = 1;

 for (int msk = 2; msk < BIT(n); ++msk) {
  for (int i = 0; i < n; ++i) {
   if ((msk & BIT(i)) && (dp[msk ^ BIT(i)] & g[i]))
    dp[msk] |= BIT(i);
  }
 }

 cout << ((dp[BIT(n) - 1] & g[0]) != 0) << endl;
 return 0;
}
```

2.3. Existence of Hamiltonian Walk.

```
/*
      task: Check for existence of Hamiltonian walk in the
                  directed graph G = <V, E>.

      complexity: O(2^n * n)

      notes: Let dp[msk] be the mask of the subset consisting
                  of those vertices v for which exist a
  Hamiltonian walk over the subset msk ending in v.
*/

#define BIT(n) (1 << n)
const int MAXN = 20;

int n, m, u, v, g[MAXN], dp[BIT(MAXN)];

int main() {
  cin >> n >> m;
  for (int i = 0; i < m; ++i) {
    cin >> u >> v;
    g[v] |= BIT(u);
  }

  for (int i = 0; i < n; ++i)
    dp[BIT(i)] = BIT(i);

  for (int msk = 1; msk < BIT(n); ++msk) {
    for (int i = 0; i < n; ++i) {
      if ((msk & BIT(i)) && (dp[msk ^ BIT(i)] & g[i]))
        dp[msk] |= BIT(i);
    }
  }

  cout << (dp[BIT(n) - 1] != 0) << endl;
  return 0;
}
```

2.4. Finding the Number of Simple Paths.

```
/*
      task: Finding the number of simple paths in the
                  directed graph G = <V, E>.

      complexity: O(f)

      notes: Let dp[msk][v] be the number of Hamiltonian
                  walks in the subgraph generated by
                  vertices in msk that end in v.
*/

#define BIT(n) (1 << n)
const int MAXN = 20;

int n, m, u, v, ans, g[MAXN], dp[BIT(MAXN)][MAXN];

int main() {
  cin >> n >> m;

  for (int i = 0; i < m; ++i) {
    cin >> u >> v;
    g[u] |= BIT(v);
  }

  for (int i = 0; i < n; ++i)
    dp[BIT(i)][i] = 1;

  for (int msk = 1; msk < BIT(n); ++msk) {
    for (int i = 0; i < n; ++i)
      if (BIT(i) & msk) {
        int tmsk = msk ^ BIT(i);

        for (int j = 0; tmsk && j < n; ++j)
          if (g[j] & BIT(i))
            dp[msk][i] += dp[tmsk][j];

        ans += dp[msk][i];
      }
  }
  cout << ans - n << endl;
  return 0;
}
```

2.5. Finding the Shortest Hamiltonian Cycle.

```
/*
      task: Search for the shortest Hamiltonian cycle.
                  Let the directed graph G = (V,
                  E) have n vertices, and each
                  edge have weight d(i, j). We
                  want to find a Hamiltonian cycle for which the sum of
                  weights of its edges is minimal.

      complexity: O(2^n * n^2)

      notes: Let dp[msk][v] be the length of the shortest Hamiltonian
                  walk on the subgraph generated
                  by vertices in msk beginning in
   verex 0 and ending in vertex v.
*/

#define BIT(n) (1 << n)

using namespace std;

const int MAXN = 20, INF = 0x1fffffff;

int n, m, u, v, w, g[MAXN][MAXN], dp[BIT(MAXN)][MAXN], ans = INF;

int main() {
  cin >> n >> m;

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
      g[i][j] = INF;
  }

  for (int i = 0; i < BIT(n); ++i) {
    for (int j = 0; j < n; ++j)
      dp[i][j] = INF;
  }

  for (int i = 0; i < m; ++i) {
    cin >> u >> v;
    cin >> g[u][v];
  }

  dp[1][0] = 0;

  for (int msk = 2; msk < BIT(n); ++msk) {
    for (int i = 0; i < n; ++i)
      if (msk & BIT(i)) {
        int tmsk = msk ^ BIT(i);

        for (int j = 0; tmsk && j < n; ++j)
          dp[msk][i] = min(dp[msk][i], dp[tmsk][j] + g[j][i]);
      }
  }

  for (int i = 1; i < n; ++i)
    ans = min(ans, dp[BIT(n) - 1][i] + g[i][0]);

  cout << ans << endl;
  return 0;
}
```

2.6. Number of Hamiltonian Cycles.

```
/*
      task: Finding the number of Hamiltonian cycles in
                  the unweighted and directed graph G = (V, E).

      complexity: O(2^n * n^2)

      notes: Let dp[msk][v] be the amount of Hamiltonian walks
                  on the subgraph generated by vertices in msk that
                  begin in vertex 0 and end in vertex v.
*/

#define BIT(n) (1 << n)
const int MAXN = 20;

int n, m, u, v, ans, g[MAXN], dp[BIT(MAXN)][MAXN];

int main() {
  cin >> n >> m;

  for (int i = 0; i < m; ++i) {
```

```
    cin >> u >> v;
    g[u] |= (1 << v);
  }

  dp[1][0] = 1;

  for (int msk = 2; msk < BIT(n); ++msk) {
    for (int i = 0; i < n; ++i)
      if (msk & BIT(i)) {
        int tmsk = msk ^ BIT(i);

        for (int j = 0; tmsk && j < n; ++j)
          if (g[j] & BIT(i))
            dp[msk][i] += dp[tmsk][j];
      }
  }

  for (int i = 1; i < n; ++i)
    if (g[i] & 1)
      ans += dp[BIT(n) - 1][i];

  cout << ans << endl;
  return 0;
}
```

2.7. Number of Simple Cycles.

```
/*
      task: Finding the number of simple cycles in a
                directed graph G = <V, E>.

      complexity: O(2^n * n^2)

      notes: Let dp[msk][v] be the number of Hamiltonian
                walks in the subgraph generated by vertices
                in msk that begin in the lowest vertex in
                msk and end in vertex v.
*/

#define BIT(n) (1 << n)
#define ONES(n) __builtin_popcount(n)

const int MAXN = 20;

int n, m, u, v, g[MAXN];
long long dp[BIT(MAXN)][MAXN], ans;

int main() {
  cin >> n >> m;

  for (int i = 0; i < m; ++i) {
    cin >> u >> v;
    g[u] |= BIT(v);
  }

  for (int i = 0; i < n; ++i)
    dp[BIT(i)][i] = 1;

  for (int msk = 1; msk < BIT(n); ++msk) {
    for (int i = 0; i < n; ++i) {
      if ((msk & BIT(i)) && !(msk & -msk & BIT(i))) {
        int tmsk = msk ^ BIT(i);

        for (int j = 0; tmsk && j < n; ++j)
          if (g[j] & BIT(i))
            dp[msk][i] += dp[tmsk][j];

        if (ONES(msk) > 2 && (g[i] & msk & -msk))
          ans += dp[msk][i];
      }
    }
  }
  cout << ans << endl;
  return 0;
}
```

2.8. Shortest Hamiltonian Walk.

```
/*
      task: Search for the shortest Hamiltonian walk.
                Let the directed graph G = (V, E) have n
                vertices, and each edge have weight d(i, j).
                We want to find a Hamiltonian walk for which
                the sum of weights of its edges is minimal.
```

```
    complexity: O(2^n * n^2)

    notes: Let dp[msk][v] be the length of the shortest
            Hamiltonian walk on the subgraph generated by
            vertices in msk that end in vertex v.
*/

#define MAXN 20
#define INF 0x1fffffff
#define BIT(n) (1 << n)

using namespace std;

int n, m, ans = INF, d[MAXN][MAXN], u, v, w, dp[1 << MAXN][MAXN];

int main() {
 cin >> n >> m;

 for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j)
   d[i][j] = INF;
 }

 for (int i = 0; i < BIT(n); ++i) {
  for (int j = 0; j < n; ++j)
   dp[i][j] = INF;
 }

 for (int i = 0; i < m; ++i) {
  cin >> u >> v >> w;
  d[u][v] = w;
 }

 for (int i = 0; i < n; ++i)
  dp[1 << i][i] = 0;

 for (int msk = 1; msk < (1 << n); ++msk) {
  for (int i = 0; i < n; ++i)
   if (msk & BIT(i)) {
    int tmsk = msk ^ BIT(i);

    for (int j = 0; tmsk && j < n; ++j)
     dp[msk][i] = min(dp[tmsk][j] + d[j][i], dp[msk][i]);
   }
 }

 for (int i = 0; i < n; ++i)
  ans = min(ans, dp[BIT(n) - 1][i]);

 cout << ans << endl;
 return 0;
}
```

2.9. Subset Subset (3^n).

```
/*
  Computing all subset of subset.
  Time: 3^n
*/
#include <bits/stdc++.h>
using namespace std;

int main() {
 int N = 4;
 for (int i = 0; i < (1 << N); ++i) {
  bitset<8> n(i);
  cout << "MASK: " << n << endl;
  cout << "SUBMASK:" << endl;
  for (int j = i; j; j = (j - 1) & i) {
   bitset<8> p(j);
   cout << p << endl;
  }
  cout << endl;
 }
 return 0;
}
```

3. Data Structures

3.1. AVL Tree.

```
/*
      Coding an AVL Tree

      Remarks: Assuming keys are integers. The data structure does
             not allows duplicate keys.
      Performance:
         Insert: O(log n)
         Erase: O(log n)
         Contains: O(log n)
         Find minimum: O(log n)
         Find maximum: O(log n)
         Find k-th: O(log n)
*/

struct AVL_Tree {
  struct node {
    int key;
    int size, height;
    node *ch[2];

    int balance_factor() { return ch[1]->height - ch[0]->height; }
    void update() {
      height = 1 + max(ch[0]->height, ch[1]->height);
      size = ch[0]->size + ch[1]->size + 1;
    }
  } * root, *null;

  int key;

  node *new_node(const int &key) {
    node *x = new node();
    x->key = key;
    x->height = x->size = 1;
    x->ch[0] = x->ch[1] = null;
    return x;
  }

  node *rotate(node *x, bool b) {

    if (x == null || x->ch[!b] == null)
      return x;

    node *y = x->ch[!b];
    x->ch[!b] = y->ch[b];
    y->ch[b] = x;

    x->update();
    y->update();

    return y;
  }

  node *balance(node *x) {

    x->update();

    if (x->balance_factor() > 1) {
      if (x->ch[1]->balance_factor() <= 0)
        x->ch[1] = rotate(x->ch[1], 1);
      x = rotate(x, 0);
    } else if (x->balance_factor() < -1) {
      if (x->ch[0]->balance_factor() >= 0)
        x->ch[0] = rotate(x->ch[0], 0);
      x = rotate(x, 1);
    }

    x->update();
    return x;
  }

  node *insert(node *x, const int &key) {

    if (x == null)
      x = new_node(key);
    else {

      if (key == x->key)
        return x;

      bool b = !(key < x->key);
      x->ch[b] = insert(x->ch[b], key);

      x = balance(x);
    }

    return x = balance(x);
```

```
}

node *erase(node *x, int key) {

  if (x == null)
    return x;

  int tmp = x->key;
  if (tmp == x->key) {
    if (x->ch[0] == null || x->ch[1] == null)
      return x->ch[x->ch[0] == null];
    else {
      node *p = x->ch[0];
      while (p->ch[1] != null)
        p = p->ch[1];
      x->key = p->key;
      key = p->key;
    }
  }

  bool b = !(key < tmp);
  x->ch[b] = erase(x->ch[b], key);

  return x = balance(x);
}

bool contains(node *root, const int &key) {
  node *x = root;
  for (;;) {
    if (x == null)
      return 0;
    if (key == x->key)
      return 1;
    x = x->ch[!(key < x->key)];
  }
}

int get_extreme(bool b) {
  assert(root != null);
  node *x = root;
  while (x->ch[b] != null)
    x = x->ch[b];
  return x->key;
}

int find_kth(node *root, int k) {

  assert(root->size >= k);

  node *x = root;
  for (;;) {
    int rank = x->ch[0]->size + 1;
    if (rank == k)
      return x->key;
    if (k < rank)
      x = x->ch[0];
    else
      x = x->ch[1], k -= rank;
  }
}

/* "Public" methods */

void insert(int x) { root = insert(root, key = x); }
void erase(int x) { root = erase(root, x); }
bool contains(int x) { return contains(root, key = x); }
int get_min() { return get_extreme(0); }
int get_max() { return get_extreme(1); }
int find_kth(int k) { return find_kth(root, k); }

AVL_Tree() {
  null = new node();
  null->height = null->size = 0;
  null->ch[0] = null->ch[1] = 0;
  root = null;
}
};
```

3.2. Big Integer.

```
typedef long long Int;
const Int B = 10; // base (power of 10)
const int BW = 1; // log B
const int MAXDIGIT = 100; // it can represent 4 * MAXDIGIT digits (in base 10)

struct BigNum {
  Int digit[MAXDIGIT];
  int size;
```

```
  BigNum(int size = 1, Int a = 0) : size(size) {
    memset(digit, 0, sizeof(digit));
    digit[0] = a;
  }
};
const BigNum ZERO(1, 0), ONE(1, 1);

// Comparators
bool operator<(BigNum x, BigNum y) {
  if (x.size != y.size)
    return x.size < y.size;
  for (int i = x.size - 1; i >= 0; --i)
    if (x.digit[i] != y.digit[i])
      return x.digit[i] < y.digit[i];
  return false;
}
bool operator>(BigNum x, BigNum y) { return y < x; }
bool operator<=(BigNum x, BigNum y) { return !(y < x); }
bool operator>=(BigNum x, BigNum y) { return !(x < y); }
bool operator!=(BigNum x, BigNum y) { return x < y || y < x; }
bool operator==(BigNum x, BigNum y) { return !(x < y) && !(y < x); }

BigNum normal(BigNum x) {
  Int c = 0;
  for (int i = 0; i < x.size; ++i) {
    while (x.digit[i] < 0)
      x.digit[i + 1] -= 1, x.digit[i] += B;
    Int a = x.digit[i] + c;
    x.digit[i] = a % B;
    c = a / B;
  }
  for (; c > 0; c /= B)
    x.digit[x.size++] = c % B;
  while (x.size > 1 && x.digit[x.size - 1] == 0)
    --x.size;
  return x;
}

BigNum convert(Int a) { return normal(BigNum(1, a)); }

BigNum convert(const string &s) {
  BigNum x;
  int i = s.size() % BW;
  if (i > 0)
    i -= BW;
  for (; i < (int)s.size(); i += BW) {
    Int a = 0;
    for (int j = 0; j < BW; ++j)
      a = 10 * a + (i + j >= 0 ? s[i + j] - '0' : 0);
    x.digit[x.size++] = a;
  }
  reverse(x.digit, x.digit + x.size);
  return normal(x);
}

// Input / Output
ostream &operator<<(ostream &os, BigNum x) {
  os << x.digit[x.size - 1];
  for (int i = x.size - 2; i >= 0; --i)
    os << setw(BW) << setfill('0') << x.digit[i];
  return os;
}
istream &operator>>(istream &is, BigNum &x) {
  string s;
  is >> s;
  x = convert(s);
  return is;
}

// Basic Operations
BigNum operator+(BigNum x, BigNum y) {
  if (x.size < y.size)
    x.size = y.size;
  for (int i = 0; i < y.size; ++i)
    x.digit[i] += y.digit[i];
  return normal(x);
}

BigNum operator-(BigNum x, BigNum y) {
  assert(x >= y);
  for (int i = 0; i < y.size; ++i)
    x.digit[i] -= y.digit[i];
  return normal(x);
}

BigNum operator*(BigNum x, BigNum y) {
  BigNum z(x.size + y.size);
  for (int i = 0; i < x.size; ++i)
    for (int j = 0; j < y.size; ++j)
      z.digit[i + j] += x.digit[i] * y.digit[j];
  return normal(z);
}

BigNum operator*(BigNum x, Int a) {
```

```
  for (int i = 0; i < x.size; ++i)
    x.digit[i] *= a;
  return normal(x);
}

pair<BigNum, Int> divmod(BigNum x, Int a) {
  Int c = 0, t;
  for (int i = x.size - 1; i >= 0; --i) {
    t = B * c + x.digit[i];
    x.digit[i] = t / a;
    c = t % a;
  }
  return pair<BigNum, Int>(normal(x), c);
}

BigNum operator/(BigNum x, Int a) { return divmod(x, a).first; }

Int operator%(BigNum x, Int a) { return divmod(x, a).second; }

pair<BigNum, BigNum> divmod(BigNum x, BigNum y) {
  if (x.size < y.size)
    return pair<BigNum, BigNum>(ZERO, x);
  int F = B / (y.digit[y.size - 1] + 1); // multiplying good-factor
  x = x * F;
  y = y * F;
  BigNum z(x.size - y.size + 1);
  for (int k = z.size - 1, i = x.size - 1; k >= 0; --k, --i) {
    z.digit[k] = (i + 1 < x.size ? x.digit[i + 1] : 0) * B + x.digit[i];
    z.digit[k] /= y.digit[y.size - 1];
    BigNum t(k + y.size);
    for (int m = 0; m < y.size; ++m)
      t.digit[k + m] = z.digit[k] * y.digit[m];
    t = normal(t);
    while (x < t) {
      z.digit[k] -= 1;
      for (int m = 0; m < y.size; ++m)
        t.digit[k + m] -= y.digit[m];
      t = normal(t);
    }
    x = x - t;
  }
  return pair<BigNum, BigNum>(normal(z), x / F);
}

BigNum operator/(BigNum x, BigNum y) { return divmod(x, y).first; }

BigNum operator%(BigNum x, BigNum y) { return divmod(x, y).second; }

// Advanced Operations
BigNum shift(BigNum x, int k) {
  if (x.size == 1 && x.digit[0] == 0)
    return x;
  x.size += k;
  for (int i = x.size - 1; i >= k; --i)
    x.digit[i] = x.digit[i + k];
  for (int i = k - 1; i >= 0; --i)
    x.digit[i] = 0;
  return x;
}
BigNum sqrt(BigNum x) { // verified UVA 10023
  const BigNum _20 = convert(2 * B);
  BigNum odd = ZERO;
  BigNum rem(2, 0);
  BigNum ans = ZERO;
  for (int i = 2 * ((x.size - 1) / 2); i >= 0; i -= 2) {
    int group = (i + 1 < x.size ? x.digit[i + 1] : 0) * B + x.digit[i];
    odd = _20 * ans + ONE;
    rem = shift(rem, 2) + convert(group);
    int count = 0;
    while (rem >= odd) {
      count = count + 1;
      rem = rem - odd;
      odd.digit[0] += 2;
      odd = normal(odd);
    }
    ans = shift(ans, 1) + convert(count);
  }
  return ans;
}
```

3.3. Binary Heap.

```
int oo = (1 << 30);
int N, heap_size;

// O(log n)
void max_heapyfi(int *A, int i) {
  int l, r, largest = i;
```

```
  do {
    i = largest;
    l = (i << 1) + 1;
    r = (i << 1) + 2;

    if (l < heap_size && A[l] > A[largest])
      largest = l;
    if (r < heap_size && A[r] > A[largest])
      largest = r;

    swap(A[largest], A[i]);
  } while (largest != i);
}

// O(1)
int parent(int i) { return (i - 1) / 2; }

// O(log n)
void max_heapyfiUp(int *A, int i) {
  while (i >= 0 && A[i] > A[parent(i)]) {
    swap(A[i], A[parent(i)]);
    i = parent(i);
  }
}

// O(n)
void build_max_heap(int *A) {
  heap_size = N;
  for (int i = N / 2; i >= 0; --i)
    max_heapyfi(A, i);
}

// O(1)
int max_heap(int *A) { return A[0]; }

// O(log n)
int heap_extract_max(int *A) {
  if (heap_size < 1)
    return oo;

  int max = A[0];

  swap(A[0], A[heap_size - 1]);
  --heap_size;

  max_heapyfi(A, 0);

  return max;
}

// O(log n)
void heap_increase_key(int *A, int i, int key) {
  if (key <= A[i])
    return;

  A[i] = key;
  max_heapyfiUp(A, i);
}
```

3.4. Disjoint Set.

```
int N;
int parent[N], cont[N];

void initSet() {
  for (int i = 0; i < N; ++i) {
    parent[i] = i;
    cont[i] = 1;
  }
}

int SetOf(int x) { return (x == parent[x]) ? x : parent[x] = SetOf(parent[x]); }

void Merge(int x, int y) {
  x = SetOf(x);
  y = SetOf(y);

  if (x == y)
    return;

  if (cont[x] < cont[y])
    swap(x, y);

  parent[y] = x;
  cont[x] += cont[y];
}
```

3.5. Fenwick Tree 1D.

```
/*
 * Performance:
 * 0-based
 * To start the index on 1
 * lowbit --> O(1)
 * query --> O(log N)
 * update --> O(log N)
 */
template <class T> struct abi {

  vector<T> ft;
  abi(int n) : ft(n + 1, 0) {}

  int lowbit(int x) { return x & -x; }

  // item[pos] += val
  void update(int pos, T val) {
    for (; pos < (int)ft.size(); pos += lowbit(pos))
      ft[pos] += val;
  }

  // Give sum[0...pos]
  T query(int pos) {
    T sum = 0;
    for (; pos > 0; pos -= lowbit(pos))
      sum += ft[pos];
    return sum;
  }

  // Give sum[l...r]
  T query(int l, int r) {
    l = (l > 0) ? l - 1 : 0;
    return query(r) - query(l);
  }

  int highestOneBit(int n) {
    int shift = 31 - (__builtin_clz(n));
    int ans = 1;
    ans <<= shift;
    return ans;
  }

  // Return min(p|sum[0,p]>=sum)
  int lower_bound(int sum) {
    --sum;
    int pos = 0;
    for (int blockSize = highestOneBit(ft.size()); blockSize; blockSize >>= 1) {
      int nextPos = pos + blockSize;
      if (nextPos < (int)ft.size() && sum >= ft[nextPos]) {
        sum -= ft[nextPos];
        pos = nextPos;
      }
    }
    return pos + 1;
  }

  // number of free places in [0, x]
  int getZeros(int x) { return x < 0 ? 0 : x + 1 - query(x); }

  int getZeros(int x1, int x2) {
    int s = getZeros(x2) - getZeros(x1 - 1);
    return x1 <= x2 ? s : s + getZeros(ft.size() - 1);
  }
};
```

3.6. Fenwick Tree 2D.

```
/*
 * Performance:
 * 0-based
 * To start the index on 1
 * lowbit --> O(1)
 * query --> O( log (N+M) )
 * update --> O( log (N+M) )
 */

// Tested 1904 - Again Making Queries III COJ
#define MOD 10000
#define MaxN 4005
int N, U, Q;
int ft[MaxN][MaxN];

int lowbit(int x) { return x & -x; }
```

```
bool Valid(int r, int c) {
 if (r < 1 || r > N)
  return false;
 if (c < 1 || c > N)
  return false;
 return true;
}
void update(int r, int c, int val) {
 if (!Valid(r, c))
  return;
 for (int i = r; i <= N; i += lowbit(i))
  for (int j = c; j <= N; j += lowbit(j))
   ft[i][j] += val;
}

int query(int r, int c) {
 if (!Valid(r, c))
  return 0;
 int sum = 0;
 for (int i = r; i > 0; i -= lowbit(i))
  for (int j = c; j > 0; j -= lowbit(j))
   sum += ft[i][j];
 return sum;
}
int query(int r, int c, int R, int C) {
 return query(R, C) - query(R, c - 1) - query(r - 1, C) + query(r - 1, c - 1);
}
```

3.7. Fraction.

```
template <class T> struct fraction {
 T n, d;

 fraction() {
  n = 0;
  d = 1;
 }

 fraction(T _n, T _d) {
  n = _n;
  d = _d;
 }
};

template <class T> fraction<T> operator+(fraction<T> &a, fraction<T> &b) {
 T mcm = a.d * b.d;
 return fraction<T>(mcm / a.d * a.n + mcm / b.d * b.n, mcm);
}

template <class T> fraction<T> operator*(fraction<T> &a, fraction<T> &b) {
 return fraction<T>(a.n * b.n, a.d * b.d);
}

template <class T> istream &operator>>(istream &in, fraction<T> &frac) {
 in >> frac.n >> frac.d;
 return in;
}

template <class T> ostream &operator<<(ostream &out, fraction<T> &frac) {
 out << frac.n << "/" << frac.d;
 return out;
}

template <class T> bool operator<(fraction<T> a, fraction<T> b) {
 return a.n * b.d < a.d * b.n;
}
```

3.8. Kd-Tree.

```
/*
 TASK : Coding a kd-tree

 Remarks: The data structure is used in this code to
        answer 2D range queries on a set of n 2D
        points of the type "report all points inside
        a rectangle [a,b]x[c,d]". The points' coordinates
        are assumed to be integers.
 Performance:
        Build kd-tree: O(n log n)*
        Query: O(sqrt(n) + k)
        k: number of points inside query region

        * expected
```

```
*/
#define MAXN 10000
#define oo 1000000000

struct point {
 int x, y;
};
struct region {
 int xlo, xhi, ylo, yhi;
};

struct node {
 point p;
 node *l, *r;
 region R;
 node(point p, node *l, node *r, int xlo, int xhi, int ylo, int yhi)
    : p(p), l(l), r(r) {
  R = (region){xlo, xhi, ylo, yhi};
 }
} * root;

int N, Q;
int xlo, ylo;
int xhi, yhi;
region R;

point p[MAXN];

inline bool leaf(node *x) { return !x->l && !x->r; }
inline bool less_than(const point &a, const point &b, bool byX) {
 return byX ? a.x < b.x : a.y < b.y;
}

void partition(point a[], int lo, int hi, const int &k, bool byX) {

 int l = lo, r = hi - 1, mid = (lo + hi) >> 1;

 if (less_than(a[mid], a[lo], byX))
  swap(a[mid], a[lo]);
 if (less_than(a[hi], a[lo], byX))
  swap(a[hi], a[lo]);
 if (less_than(a[hi], a[mid], byX))
  swap(a[hi], a[mid]);

 if (hi - lo + 1 <= 3)
  return;

 swap(a[mid], a[hi - 1]);
 point pivot = a[hi - 1];

 for (;;) {
  while (less_than(a[++l], pivot, byX))
   ;
  while (less_than(pivot, a[--r], byX))
   ;
  if (l < r)
   swap(a[l], a[r]);
  else
   break;
 }

 swap(a[l], a[hi - 1]);

 if (k < l)
  partition(a, lo, l - 1, k, byX);
 if (k > l)
  partition(a, l + 1, hi, k, byX);
}

node *build_kd_tree(point p[], int len, int depth, int xlo, int xhi, int ylo,
               int yhi) {

 if (len == 1)
  return new node(p[0], 0, 0, p[0].x, p[0].x, p[0].y, p[0].y);

 int mid = (len - 1) / 2;
 partition(p, 0, len - 1, mid, !(depth & 1));

 int c1 = 0, c2 = 0;
 point p1[MAXN], p2[MAXN];
 for (int i = 0; i <= mid; i++)
  p1[c1++] = p[i];
 for (int i = mid + 1; i < len; i++)
  p2[c2++] = p[i];

 int xlo1 = xlo, xhi1 = xhi, ylo1 = ylo, yhi1 = yhi, xlo2 = xlo, xhi2 = xhi,
   ylo2 = ylo, yhi2 = yhi;

 if (!(depth & 1))
  xhi1 = p[mid].x, xlo2 = p[mid].x + 1;
 else
  yhi1 = p[mid].y, yhi2 = p[mid].y + 1;

 node *left = build_kd_tree(p1, mid + 1, depth + 1, xlo1, xhi1, ylo1, yhi1);
```

```
  node *right =
      build_kd_tree(p2, len - mid - 1, depth + 1, xlo2, xhi2, ylo2, yhi2);

  return new node(p[mid], left, right, xlo, xhi, ylo, yhi);
}

void report(node *t) {
  if (!t)
    return;
  if (leaf(t))
    printf("(%d,%d) ", t->p.x, t->p.y);
  else {
    report(t->l);
    report(t->r);
  }
}

region make_region(node *t) {
  return (region){t->R.xlo, t->R.xhi, t->R.ylo, t->R.yhi};
}

bool contained(const region &a, const region &b) {
  return (b.xlo <= a.xlo && a.xlo <= b.xhi && b.xlo <= a.xhi &&
          a.xhi <= b.xhi && b.ylo <= a.ylo && a.ylo <= b.yhi &&
          b.ylo <= a.yhi && a.yhi <= b.yhi);
}

bool intersect(const region &a, const region &b) {
  bool okX = ((a.xlo <= b.xlo && b.xlo <= a.xhi) ||
              (a.xlo <= b.xhi && b.xhi <= a.xhi));
  bool okY = ((a.ylo <= b.ylo && b.ylo <= a.yhi) ||
              (a.ylo <= b.yhi && b.yhi <= a.yhi));
  return okX && okY;
}

void query(node *t, const region &R) {
  if (leaf(t)) {
    if (contained(t->R, R))
      report(t);
  } else {

    region lc = make_region(t->l);
    if (contained(lc, R))
      report(t->l);
    else if (intersect(lc, R))
      query(t->l, R);

    region rc = make_region(t->r);
    if (contained(rc, R))
      report(t->r);
    else if (intersect(rc, R))
      query(t->r, R);
  }
}

int main() {

  scanf("%d", &N);
  for (int i = 0; i < N; i++)
    scanf("%d %d", &p[i].x, &p[i].y);

  root = build_kd_tree(p, N, 0, -oo, oo, -oo, oo);

  for (scanf("%d", &Q); Q--;) {
    scanf("%d %d %d %d", &xlo, &ylo, &xhi, &yhi);
    R = (region){xlo, xhi, ylo, yhi};
    query(root, R);
    printf("\n");
  }

  return 0;
}
```

3.9. Longest Common Ancestor. Sparse Table.

```
/*
        TASK : LCA Problem using DP
        Performance:
              Preprocess logarithms --> O(V)
              Build tree --> O(V)
              buildSparseTable --> O(V log V)
              queryLCA --> O(log V)
*/
#define LOGV 16
#define MAXV 1 << LOGV

using namespace std;
```

```
struct Node {
 int v, next;
} L[MAXV];

int V;
int P[MAXV];
int level[MAXV], parent[MAXV];
int LCA[MAXV][LOGV];

void readTree() {
 for (int i = 0; i < V - 1; ++i) {
  int u, v;
  cin >> u >> v;
  --u;
  --v;

  L[2 * i] = (Node){v, P[u]};
  P[u] = 2 * i;

  L[2 * i + 1] = (Node){u, P[v]};
  P[v] = 2 * i + 1;
 }
}

void buildSparseTable() {
 queue<int> Q;
 level[0] = 0;
 parent[0] = -1;

 for (Q.push(0); !Q.empty(); Q.pop()) {
  int u = Q.front();
  for (int i = P[u]; i != -1; i = L[i].next) {
   int v = L[i].v;
   if (v == parent[u])
    continue;
    parent[v] = u;
    level[v] = level[u] + 1;
    Q.push(v);

    // DP
    LCA[v][0] = u;
    for (int j = 1; j <= __lg(level[v]); ++j)
     LCA[v][j] = LCA[LCA[v][j - 1]][j - 1];
  }
 }
}

int queryLCA(int u, int v) {
 if (level[u] < level[v])
  swap(u, v);

 if (level[u] != level[v])
  for (int i = __lg(level[u]); i >= 0; --i)
   if (level[u] - (1 << i) >= level[v])
    u = LCA[u][i];

 if (u == v)
  return u;

 for (int i = __lg(level[u]); i >= 0; --i)
  if (level[u] - (1 << i) >= 0 && LCA[u][i] != LCA[v][i]) {
   u = LCA[u][i];
   v = LCA[v][i];
  }
 return parent[u];
}

void init() {
 memset(P, -1, sizeof(P));
 readTree();
 buildSparseTable();
}
```

3.10. Polynomial.

```
template <class T> struct polynomial {
 int deg;
 vector<T> coef;

 polynomial() {}

 polynomial(int _deg) {
  deg = _deg;
  coef = vector<T>(deg + 1, 0);
 }

 polynomial(int _deg, vector<T> _coef) {
```

```
    deg = _deg;
    coef = _coef;
  }

  T eval(double x) {
    T y = 0;
    double pow = 1;
    for (int i = 0; i <= deg; ++i) {
      y = y + coef[i] * pow;
      pow = pow * x;
    }
    return y;
  }
};

template <class T> istream &operator>>(istream &in, polynomial<T> &pol) {
  in >> pol.deg;
  pol.coef = vector<T>(pol.deg + 1);

  for (int i = 0; i <= pol.deg; ++i)
    in >> pol.coef[i];
  return in;
}

void literal(ostream &out, int i) {
  if (i == 0)
    return;
  if (i == 1) {
    out << "x";
    return;
  }
  out << "x^" << i;
}

template <class T> ostream &operator<<(ostream &out, polynomial<T> pol) {
  bool first = true;
  for (int i = pol.deg; i > 0; --i) {
    if (pol.coef[i] != 0) {
      if (first) {
        if (pol.coef[i] != 1 && pol.coef[i] != -1)
          out << pol.coef[i];
        else if (pol.coef[i] == -1)
          out << "-";
      } else {
        if (pol.coef[i] == 1)
          out << "+";
        else if (pol.coef[i] == -1)
          out << "-";
        else if (pol.coef[i] > 0)
          out << "+" << pol.coef[i];
        else
          out << pol.coef[i];
      }

      if (i == 1)
        out << "x";
      else if (i > 1)
        out << "x^" << i;

      first = false;
    }
  }

  if (first) {
    out << pol.coef[0];
    return out;
  } else {
    if (pol.coef[0] != 0) {
      if (pol.coef[0] > 0)
        out << "+";
      out << pol.coef[0];
    }
  }

  return out;
}

template <class T> polynomial<T> operator+(polynomial<T> &a, polynomial<T> &b) {
  polynomial<T> sum;

  if (a.deg >= b.deg)
    sum = a;
  else
    sum = b;

  for (int i = 0; i <= min(a.deg, b.deg); ++i)
    sum.coef[i] = a.coef[i] + b.coef[i];
  return sum;
}

template <class T>
polynomial<T> operator*(polynomial<T> &p1, polynomial<T> &p2) {
  polynomial<T> mult(p1.deg + p2.deg);
  for (int i = 0; i <= p1.deg; ++i)
```

```
  for (int j = 0; j <= p2.deg; ++j)
   mult.coef[i + j] = mult.coef[i + j] + p1.coef[i] * p2.coef[j];
  return mult;
}
```

3.11. Range Minimum Query Fast.

```
struct RMQ {
 vector<int> rmq;
 int n;
 RMQ(vector<int> &a) {
  n = a.size();
  buildRMQ(a);
 }
 void buildRMQ(vector<int> &a) {
  int logn = 1;
  for (int k = 1; k < n; k <<= 1)
   ++logn;
  rmq = vector<int>(n * logn);
  vector<int>::iterator b = rmq.begin();
  copy(ALL(a), b);
  for (int k = 1; k < n; k <<= 1) {
   copy(b, b + n, b + n);
   b += n;
   REP(i, n - k) b[i] = min(b[i], b[i + k]);
  }
 }
 int minimum(int x, int y) {
  int z = y - x, k = 0, e = 1, s; // y-x>=e=2^k k up to a
  s = ((z & 0xffff0000) != 0) << 4;
  z >>= s;
  e <<= s;
  k |= s;
  s = ((z & 0x0000ff00) != 0) << 3;
  z >>= s;
  e <<= s;
  k |= s;
  s = ((z & 0x000000f0) != 0) << 2;
  z >>= s;
  e <<= s;
  k |= s;
  s = ((z & 0x0000000c) != 0) << 1;
  z >>= s;
  e <<= s;
  k |= s;
  s = ((z & 0x00000002) != 0) << 0;
  z >>= s;
  e <<= s;
  k |= s;
  return min(rmq[x + n * k], rmq[y + n * k - e + 1]);
 }
};
```

3.12. Range Minimum Query.

```
/*
      Start in 0.
      TASK : Range Minimum Query Problem: Given a sequence S of real numbers,
            RMQ(i,j) returns the index of element in S[i...j] with
            smallest value.

      Preprocess Sparse Table --> O(N log N)
      Answer query --> O(1)
*/

// Tested 1651 - Finding Minimum COJ
//  1082 - Array Queries Lightoj

const int Max = 10005, MaxLog = 15;

int N;
int rmq[Max][MaxLog], array[Max];

void build() {
 for (int i = 0; i < N; ++i)
  rmq[i][0] = i;
 for (int i = 1; (1 << i) <= N; ++i)
  for (int j = 0; j + (1 << i) <= N; ++j) {
   if (array[rmq[j][i - 1]] < array[rmq[j + (1 << (i - 1))][i - 1]])
    rmq[j][i] = rmq[j][i - 1];
   else
    rmq[j][i] = rmq[j + (1 << (i - 1))][i - 1];
  }
```

```
}

int query(int l, int r) {
 int k = __lg(r - l + 1);
 return array[rmq[l][k]] < array[rmq[r - (1 << k) + 1][k]]
        ? rmq[l][k]
        : rmq[r - (1 << k) + 1][k];
}
```

3.13. Range Minimum Sum Segment Query.

```
/*
      TASK : Range Minimum-Sum Segment Query Problem
           With two intervals too.

      Compute arrays C, P and M --> O(N)
      Preprocess RMQ --> O(N log N)
      Answer RMSQ queries --> O(1)
*/
#define MAXN 50005
#define LGN 16

int A[MAXN];
int C[MAXN], P[MAXN], M[MAXN], L[MAXN];
int RMQ[MAXN][LGN][2];

// for two intervals
int rmqMAXC[MAXN][LGN];

int N;

// Compute arrays C, P, L and M
// C[i] = sum(A[1]...A[i])
// L[i] = max{k | C[k] >= C[i] k[1, i-1]}
//   {0    otherwise }
// P[i] = max{k | k[L[i]+1, i] and C[k-1] <= C[l] forall l[L[i], i-1]}
// M[i] = sum(P[i], i)
void buildCLPM() {
 for (int i = 1; i <= N; ++i) {
  C[i] = C[i - 1] + A[i];
  L[i] = i - 1;
  P[i] = i;
  while (C[L[i]] < C[i] && L[i]) {
   if (C[P[L[i]] - 1] < C[P[i] - 1])
    P[i] = P[L[i]];
   L[i] = L[L[i]];
  }
  M[i] = C[i] - C[P[i] - 1];
 }
}

// Preprocess array C for RMQmin and array M for RMQmax
// RMQ[i][j][0] holds the minimum, while RMQ[i][j][1] holds
// the maximum
void buildRMQ() {
 for (int i = 0; i <= N; ++i)
  RMQ[i][0][0] = RMQ[i][0][1] = i;

 for (int j = 1; j <= __lg(N + 1); ++j)
  for (int i = 0; i + (1 << j) - 1 <= N + 1; ++i) {
   if (C[RMQ[i][j - 1][0]] <= C[RMQ[i + (1 << (j - 1))][j - 1][0]])
    RMQ[i][j][0] = RMQ[i][j - 1][0];
   else
    RMQ[i][j][0] = RMQ[i + (1 << (j - 1))][j - 1][0];
   if (M[RMQ[i][j - 1][1]] >= M[RMQ[i + (1 << (j - 1))][j - 1][1]])
    RMQ[i][j][1] = RMQ[i][j - 1][1];
   else
    RMQ[i][j][1] = RMQ[i + (1 << (j - 1))][j - 1][1];
  }
}

int queryRMQ(int l, int r, int b) {

 int k = __lg(r - l + 1);

 // For two Intervals
 if (b == 2)
  return max(rmqMAXC[l][k], rmqMAXC[r - (1 << k) + 1][k]);

 if (!b)
  return C[RMQ[l][k][b]] <= C[RMQ[r - (1 << k) + 1][k][b]]
         ? RMQ[l][k][b]
         : RMQ[r - (1 << k) + 1][k][b];
 else
  return M[RMQ[l][k][b]] >= M[RMQ[r - (1 << k) + 1][k][b]]
         ? RMQ[l][k][b]
         : RMQ[r - (1 << k) + 1][k][b];
}
```

```
pair<int, int> queryRMSQ(int l, int r) {
 int x = queryRMQ(l, r, 1);
 if (P[x] < l) {
  int y = queryRMQ(x + 1, r, 1);
  int z = queryRMQ(l - 1, x - 1, 0) + 1;
  if (C[x] - C[z - 1] < M[y])
   return pair<int, int>(P[y], y);
  return pair<int, int>(z, x);
 }
 return pair<int, int>(P[x], x);
}

// RMSQ with two intervals
// Return i <= x <= j, k <= y <= l
// max{ Sum(x, y) }

void buildRMSQ2() {
 // Apply RMSQ preprocessing to A
 // Apply RMQmin and RMQmax preprocessing to C[]
 for (int i = 0; i <= N; ++i)
  rmqMAXC[i][0] = i;
 for (int j = 1; j <= __lg(N + 1); ++j)
  for (int i = 0; i + (1 << j) - 1 <= N + 1; ++i) {
   if (C[rmqMAXC[i][j - 1]] >= C[rmqMAXC[i + (1 << (j - 1))][j - 1]])
    rmqMAXC[i][j] = rmqMAXC[i][j - 1];
   else
    rmqMAXC[i][j] = rmqMAXC[i + (1 << (j - 1))][j - 1];
  }
}

pair<int, int> queryRMSQ(int i, int j, int k, int l) {
 if (j <= k)
  return pair<int, int>(queryRMQ(i - 1, j - 1, 0) + 1, queryRMQ(k, l, 2));

 int x[4], y[4];

 x[1] = queryRMQ(i - 1, k - 1, 0) + 1;
 y[1] = queryRMQ(k, l, 2);

 x[2] = queryRMQ(k, j - 1, 0) + 1;
 y[2] = queryRMQ(j, l, 2);

 pair<int, int> tmp = queryRMSQ(k, j);
 x[3] = tmp.first;
 y[3] = tmp.second;

 int maxSum = max(C[x[1]] - C[y[1] - 1],
          max(C[x[2]] - C[y[2] - 1], C[x[3]] - C[y[3] - 1]));

 if (C[x[1]] - C[y[1] - 1] == maxSum)
  return pair<int, int>(x[1], y[1]);
 if (C[x[2]] - C[y[2] - 1] == maxSum)
  return pair<int, int>(x[2], y[2]);
 return pair<int, int>(x[3], y[3]);
}

int main() {

 cin >> N;
 for (int i = 1; i <= N; ++i)
  cin >> A[i];

 buildCLPM();
 buildRMQ();
 buildRMSQ2();

 int q;
 cin >> q;
 for (int i = 0; i < q; ++i) {
  /*int l, r; cin >> l >> r;
  pair<int, int> ans = queryRMSQ(l, r);
  cout << ans.first << " " << ans.second << endl;*/
  int a, b, c, d;
  cin >> a >> b >> c >> d;
  pair<int, int> ans = queryRMSQ(a, b, c, d);
  cout << ans.first << "␣" << ans.second << endl;
 }
 return 0;
}
```

3.14. Segment Tree Lazy Propagation.

```
/*
    In this example:
    update item[l...r] + val
    query sum(item[l...r])
*/
```

```
#define MaxN 1000
#define Left(x) ((x << 1) + 1)
#define Right(x) ((x << 1) + 2)

int st[4 * MaxN], lazy[4 * MaxN];

void push(int node, int nodeL, int nodeR) {
  int m = (nodeL + nodeR) / 2;

  lazy[Left(node)] += lazy[node];
  lazy[Right(node)] += lazy[node];

  st[Left(node)] += (m - nodeL + 1) * lazy[node];
  st[Right(node)] += (nodeR - m) * lazy[node];

  lazy[node] = 0;
}

void update(int node, int nodeL, int nodeR, int l, int r, int val) {
  if (l > nodeR || r < nodeL)
    return;
  if (nodeL >= l && nodeR <= r) {
    st[node] += (nodeR - nodeL + 1) * val;
    lazy[node] += val;
    return;
  }
  push(node, nodeL, nodeR);

  int m = (nodeL + nodeR) / 2;
  update(Left(node), nodeL, m, l, r, val);
  update(Right(node), m + 1, nodeR, l, r, val);
  st[node] = st[Left(node)] + st[Right(node)];
}

int query(int node, int nodeL, int nodeR, int l, int r) {
  if (l > nodeR || r < nodeL)
    return 0;
  if (nodeL >= l && nodeR <= r)
    return st[node];
  push(node, nodeL, nodeR);
  int m = (nodeL + nodeR) / 2;
  return query(Left(node), nodeL, m, l, r) +
       query(Right(node), m + 1, nodeR, l, r);
}
```

3.15. Segment Tree-1D Query.

```
/*
      In this example update is in a position and the query is
      the sum of interval. item[N], st[4*N]
*/
#define Left(x) ((x << 1) + 1)
#define Right(x) ((x << 1) + 2)
#define MaxN 1000

int item[MaxN];

void build(int *st, int node, int nodeL, int nodeR) {
  if (nodeL == nodeR) {
    st[node] = item[nodeL];
    return;
  }
  int m = (nodeL + nodeR) / 2;
  build(st, Left(node), nodeL, m);
  build(st, Right(node), m + 1, nodeR);
  st[node] = st[Left(node)] + st[Right(node)];
}

void update(int *st, int node, int nodeL, int nodeR, int pos, int val) {
  if (nodeL == nodeR) {
    st[node] = val;
    return;
  }
  int m = (nodeL + nodeR) / 2;
  if (pos <= m)
    update(st, Left(node), nodeL, m, pos, val);
  else
    update(st, Right(node), m + 1, nodeR, pos, val);
  st[node] = st[Left(node)] + st[Right(node)];
}

int query(int *st, int node, int nodeL, int nodeR, int l, int r) {
  if (nodeL == l && nodeR == r)
    return st[node];
  int m = (nodeL + nodeR) / 2;
  if (r <= m)
    return query(st, Left(node), nodeL, m, l, r);
  if (l > m)
    return query(st, Right(node), m + 1, nodeR, l, r);
```

```
  return query(st, Left(node), nodeL, m, l, m) +
      query(st, Right(node), m + 1, nodeR, m + 1, r);
}
```

3.16. Segment Tree-2D.

```
/*
        TASK : Range Minimum-Sum Segment Query Problem
               With two intervals too.

        Compute arrays C, P and M --> O(N)
        Preprocess RMQ --> O(N log N)
        Answer RMSQ queries --> O(1)
*/
#define MAXN 50005
#define LGN 16

int A[MAXN];
int C[MAXN], P[MAXN], M[MAXN], L[MAXN];
int RMQ[MAXN][LGN][2];

// for two intervals
int rmqMAXC[MAXN][LGN];

int N;

// Compute arrays C, P, L and M
// C[i] = sum(A[1]...A[i])
// L[i] = max{k | C[k] >= C[i] k[1, i-1]}
//   {0    otherwise }
// P[i] = max{k | k[L[i]+1, i] and C[k-1] <= C[l] forall l[L[i], i-1]}
// M[i] = sum(P[i], i)
void buildCLPM() {
  for (int i = 1; i <= N; ++i) {
    C[i] = C[i - 1] + A[i];
    L[i] = i - 1;
    P[i] = i;
    while (C[L[i]] < C[i] && L[i]) {
      if (C[P[L[i]] - 1] < C[P[i] - 1])
        P[i] = P[L[i]];
      L[i] = L[L[i]];
    }
    M[i] = C[i] - C[P[i] - 1];
  }
}

// Preprocess array C for RMQmin and array M for RMQmax
// RMQ[i][j][0] holds the minimum, while RMQ[i][j][1] holds
// the maximum
void buildRMQ() {
  for (int i = 0; i <= N; ++i)
    RMQ[i][0][0] = RMQ[i][0][1] = i;

  for (int j = 1; j <= __lg(N + 1); ++j)
    for (int i = 0; i + (1 << j) - 1 <= N + 1; ++i) {
      if (C[RMQ[i][j - 1][0]] <= C[RMQ[i + (1 << (j - 1))][j - 1][0]])
        RMQ[i][j][0] = RMQ[i][j - 1][0];
      else
        RMQ[i][j][0] = RMQ[i + (1 << (j - 1))][j - 1][0];
      if (M[RMQ[i][j - 1][1]] >= M[RMQ[i + (1 << (j - 1))][j - 1][1]])
        RMQ[i][j][1] = RMQ[i][j - 1][1];
      else
        RMQ[i][j][1] = RMQ[i + (1 << (j - 1))][j - 1][1];
    }
}

int queryRMQ(int l, int r, int b) {

  int k = __lg(r - l + 1);

  // For two Intervals
  if (b == 2)
    return max(rmqMAXC[l][k], rmqMAXC[r - (1 << k) + 1][k]);

  if (!b)
    return C[RMQ[l][k][b]] <= C[RMQ[r - (1 << k) + 1][k][b]]
           ? RMQ[l][k][b]
           : RMQ[r - (1 << k) + 1][k][b];
  else
    return M[RMQ[l][k][b]] >= M[RMQ[r - (1 << k) + 1][k][b]]
           ? RMQ[l][k][b]
           : RMQ[r - (1 << k) + 1][k][b];
}

pair<int, int> queryRMSQ(int l, int r) {
  int x = queryRMQ(l, r, 1);
  if (P[x] < l) {
    int y = queryRMQ(x + 1, r, 1);
```

```
    int z = queryRMQ(l - 1, x - 1, 0) + 1;
    if (C[x] - C[z - 1] < M[y])
      return pair<int, int>(P[y], y);
    return pair<int, int>(z, x);
  }
  return pair<int, int>(P[x], x);
}

// RMSQ with two intervals
// Return i <= x <= j, k <= y <= l
// max{ Sum(x, y) }

void buildRMSQ2() {
  // Apply RMSQ preprocessing to A
  // Apply RMQmin and RMQmax preprocessing to C[]
  for (int i = 0; i <= N; ++i)
    rmqMAXC[i][0] = i;
  for (int j = 1; j <= __lg(N + 1); ++j)
    for (int i = 0; i + (1 << j) - 1 <= N + 1; ++i) {
      if (C[rmqMAXC[i][j - 1]] >= C[rmqMAXC[i + (1 << (j - 1))][j - 1]])
        rmqMAXC[i][j] = rmqMAXC[i][j - 1];
      else
        rmqMAXC[i][j] = rmqMAXC[i + (1 << (j - 1))][j - 1];
    }
}

pair<int, int> queryRMSQ(int i, int j, int k, int l) {
  if (j <= k)
    return pair<int, int>(queryRMQ(i - 1, j - 1, 0) + 1, queryRMQ(k, l, 2));

  int x[4], y[4];

  x[1] = queryRMQ(i - 1, k - 1, 0) + 1;
  y[1] = queryRMQ(k, l, 2);

  x[2] = queryRMQ(k, j - 1, 0) + 1;
  y[2] = queryRMQ(j, l, 2);

  pair<int, int> tmp = queryRMSQ(k, j);
  x[3] = tmp.first;
  y[3] = tmp.second;

  int maxSum = max(C[x[1]] - C[y[1] - 1],
                   max(C[x[2]] - C[y[2] - 1], C[x[3]] - C[y[3] - 1]));

  if (C[x[1]] - C[y[1] - 1] == maxSum)
    return pair<int, int>(x[1], y[1]);
  if (C[x[2]] - C[y[2] - 1] == maxSum)
    return pair<int, int>(x[2], y[2]);
  return pair<int, int>(x[3], y[3]);
}

int main() {

  cin >> N;
  for (int i = 1; i <= N; ++i)
    cin >> A[i];

  buildCLPM();
  buildRMQ();
  buildRMSQ2();

  int q;
  cin >> q;
  for (int i = 0; i < q; ++i) {
    /*int l, r; cin >> l >> r;
    pair<int, int> ans = queryRMSQ(l, r);
    cout << ans.first << " " << ans.second << endl;*/
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    pair<int, int> ans = queryRMSQ(a, b, c, d);
    cout << ans.first << "␣" << ans.second << endl;
  }
  return 0;
}
```

3.17. Treap.

```
/*
  TASK : Coding a treap

  Remarks: Assuming keys are integers. Using Max Heap
  Performance:
    Insert: O(log n)*
    Erase: O(log n)*
    Find: O(log n)*
    Find k-th: O(log n)*
```

```
  * expected
*/

struct generator {

  static const int A = 48271;
  static const int M = 2147483647;
  static const int Q = M / A;
  static const int R = M % A;

  int state;

  generator() {
    srand(time(0));
    state = rand() + 1;
  }
  int pseudo_random() {
    state = A * (state % Q) - R * (state / Q);
    return state > 0 ? state : state += M;
  }
} g;

struct treap {

#define SIZE(x) ((x) ? (x)->size : 0)
#define RESIZE(x) (SIZE((x)->ch[0]) + SIZE((x)->ch[1]) + (x)->cnt)

  struct node {
    int key, p, size, cnt;
    node *ch[2];
    node(int key) : key(key), p(g.pseudo_random()), size(1), cnt(1) {
      ch[0] = ch[1] = 0;
    }
  } * root;

  int key;

  node *rotate(node *x, bool b) {
    node *y = x->ch[!b];
    x->ch[!b] = y->ch[b];
    y->ch[b] = x;
    x->size = RESIZE(x);
    y->size = RESIZE(y);
    return y;
  }

  node *insert(node *t, const int &key) {

    if (!t)
      return new node(key);

    if (key == t->key)
      t->cnt++, t->size++;
    else {
      bool b = !(key < t->key);
      t->ch[b] = insert(t->ch[b], key);
      t->size = RESIZE(t);
      if (t->ch[b]->p > t->p)
        t = rotate(t, !b);
    }
    return t;
  }

  node *erase(node *t, const int &key) {

    if (!t)
      return 0;

    if (key != t->key) {
      bool b = !(key < t->key);
      t->ch[b] = erase(t->ch[b], key);
      t->size = RESIZE(t);
    } else {
      if (t->cnt > 1)
        t->cnt--, t->size--;
      else {
        if (!t->ch[0] && !t->ch[1]) {
          delete t;
          return 0;
        } else if (!t->ch[0])
          t = rotate(t, 0);
        else if (!t->ch[1])
          t = rotate(t, 1);
        else
          t = rotate(t, t->ch[0]->p > t->ch[1]->p);
        t = erase(t, key);
      }
    }

    return t;
  }

  /* "Public" methods */
```

```
  void insert(int x) { root = insert(root, key = x); }
  void erase(int x) { root = erase(root, key = x); }
  int size() { return SIZE(root); }
  bool find(int x) {
    node *t = root;
    while (t) {
      if (x == t->key)
        return 1;
      t = t->ch[!(x < t->key)];
    }
    return 0;
  }
  int find_kth(int k) { /* assuming k <=
                         SIZE( root ) */
    node *t = root;
    while (1) {
      int lo_rank = SIZE(t->ch[0]) + 1, hi_rank = SIZE(t->ch[0]) + t->cnt;
      if (lo_rank <= k && k <= hi_rank)
        return t->key;
      else if (k < lo_rank)
        t = t->ch[0];
      else {
        k -= hi_rank;
        t = t->ch[1];
      }
    }
  }

  treap() : root(0) {}
};
```

3.18. Treap Implicit Key.

```
typedef long long ptype;

ptype seed = 47;

ptype my_rand() {
  seed = (seed * 279470273) % 4294967291LL;
  return seed;
}

struct ImplicitTreap {

  struct item {
    int value;
    ptype prior;
    item *l, *r;
    int sons;

    bool rev;
    long long sum;

    item() {}
    item(int value) : value(value), l(0), r(0), sons(0), rev(0), sum(value) {
      prior = my_rand();
    }
  } * root;

  void fix(item *t) {
    if (!t)
      return;
    t->sons = (t->l ? t->l->sons + 1 : 0) + (t->r ? t->r->sons + 1 : 0);
    t->sum = (t->l ? t->l->sum : 0) + (t->r ? t->r->sum : 0) + t->value;
  }

  void push(item *it) {
    if (it && it->rev) {
      it->rev = 0;
      swap(it->l, it->r);
      if (it->l)
        it->l->rev ^= 1;
      if (it->r)
        it->r->rev ^= 1;
    }
  }

  void merge(item *&t, item *l, item *r) {
    push(l);
    push(r);

    if (!l || !r)
      t = l ? l : r;
    else if (l->prior > r->prior)
      merge(l->r, l->r, r), t = l;
    else
      merge(r->l, l, r->l), t = r;

    fix(t);
```

```
}

void split(item *t, item *&l, item *&r, int pos, int add = 0) {
 if (!t)
  l = r = NULL;
 else {
  push(t);
  int cur_pos = add + (t->l ? 1 + t->l->sons : 0);

  if (pos <= cur_pos)
   split(t->l, l, t->l, pos, add), r = t;
  else
   split(t->r, t->r, r, pos, cur_pos + 1), l = t;

  fix(t);
 }
}

void insert(item *&t, item *&it, int pos, int add = 0) {
 if (!t)
  t = it;
 else {
  push(t);
  int cur_pos = add + (t->l ? 1 + t->l->sons : 0);

  if (it->prior > t->prior) {
   split(t, it->l, it->r, pos, add), t = it;
  } else {
   if (pos <= cur_pos)
    insert(t->l, it, pos, add);
   else
    insert(t->r, it, pos, cur_pos + 1);
  }
  fix(t);
 }
}

void remove(item *&t, int pos, int add = 0) {
 int cur_pos = add + (t->l ? 1 + t->l->sons : 0);

 if (cur_pos == pos)
  merge(t, t->l, t->r);
 else if (pos < cur_pos)
  remove(t->l, pos, add);
 else
  remove(t->r, pos, cur_pos + 1);
  fix(t);
}

void reverse(item *t, int l, int r) {
 item *t1, *t2, *t3;

 split(t, t1, t2, l);
 split(t2, t2, t3, r - l + 1);

 t2->rev ^= 1;
 merge(t, t1, t2);
 merge(t, t, t3);
}

long long sum(item *&t, int lo, int hi, int a, int b, int add = 0) {
 if (!t || lo > b || hi < a)
  return 0;

 if (a <= lo && hi <= b)
  return t->sum;

 if (t->rev)
  push(t);
 int cur_key = add + (t->l ? 1 + t->l->sons : 0);

 long long ret = (a <= cur_key && cur_key <= b ? t->value : 0);

 ret += sum(t->l, lo, cur_key - 1, a, b, add);
 ret += sum(t->r, cur_key + 1, hi, a, b, cur_key + 1);

 return ret;
}

void print(item *t) {
 if (!t)
  return;
 push(t);
 print(t->l);
 printf("%d␣", t->value);
 print(t->r);
}

void clear(item *t) {
 if (!t)
  return;
 clear(t->l);
 delete t;
```

```
    clear(t->r);
  }

  ImplicitTreap() { root = 0; }

  /*Public Methods*/

  void print() { print(root); }

  int size() { return root->sons + 1; }

  long long sum(int l, int r) { return sum(root, 0, this->size() - 1, l, r); }

  void reverse(int l, int r) { reverse(root, l, r); }

  void remove(int pos) { remove(root, pos); }

  void insert(int pos, int val) {
    item *node = new item(val);
    insert(root, node, pos);
  }

  void clear() { clear(root); }
};
```

3.19. **Trie.**

```
/*
    TASK : Given a set P of strings and a string S, count how many
           elements of P contain S as a prefix, and how many p(i), for
           some i, have |p(i)|<|S|.

    Remarks: Using English alphabet (|S|=26)
    Performance:
      Insert: O(|p|)
      Count: O(|p|)
      p: string processed
*/

#define MAXLEN 20000

struct Trie {

  struct node {
    int partial, full;
    node *edge[26];
    node() : partial(0), full(0) { memset(edge, 0, sizeof(edge)); }
  } * root;

  Trie() { root = new node(); }

  void insert(char s[], int len) {

    node *t = root;
    for (int i = 0; i < len; i++) {
      char c = s[i] - 'a';
      if (!t->edge[c])
        t->edge[c] = new node();
      t = t->edge[c];
      t->partial++;
    }
    t->full++;
  }

  int count(char s[], int len) {

    if (!root)
      return 0;

    node *t = root;
    int ret = 0;

    for (int i = 0; i < len; i++) {
      if (!t)
        break;
      ret += t->full;
      t = t->edge[s[i] - 'a'];
    }
    if (t)
      ret += t->partial;

    return ret;
  }
};
```

4. Dynamic Programming

4.1. Convex Hull Trick.

```
typedef pair<int, int> pii;
typedef long long ll;

struct line {
 ll m, b;
 line(ll m, ll b) : m(m), b(b) {}
};

struct ConvexHullTrick {

 int len, ptr;
 vector<line> r;
 ConvexHullTrick(int n) {
  r.assign(n, line(0, 0));
  ptr = len = 0;
 }

 bool bad(line l1, line l2, line l3) {
  return (l3.b - l1.b) * (l1.m - l2.m) < (l2.b - l1.b) * (l1.m - l3.m);
 }

 void add(line x) {
  while (len >= 2 && bad(r[len - 2], r[len - 1], x))
   --len;
  r[len++] = x;
 }

 ll query(int x) {
  ptr = min(ptr, len - 1);
  while (ptr + 1 < len &&
      r[ptr + 1].m * x + r[ptr + 1].b < r[ptr].m * x + r[ptr].b)
   ++ptr;
  return r[ptr].m * x + r[ptr].b;
 }
};
```

4.2. Longest Increasing Subsequence.

```
const int oo = 99999999;

#define index_of(as, x) \
 distance(as.begin(), lower_bound(as.begin(), as.end(), x))

/*
   Tested: LISTA
   Contest 3 COCI 2006-2007
*/
vector<int> lis_fast(const vector<int> &a) {
 const int n = a.size();
 vector<int> A(n, oo), id(n);

 for (int i = 0; i < n; ++i) {
  id[i] = index_of(A, a[i]);
  A[id[i]] = a[i];
 }

 int m = *max_element(id.begin(), id.end());
 vector<int> b(m + 1);

 for (int i = n - 1; i >= 0; --i)
  if (id[i] == m)
   b[m--] = a[i];

 return b;
}
```

4.3. Matrix Chain.

```
const int oo = 1 << 30;

int matrix_chain(const vector<int> &p) {
 int n = p.size() - 1;
```

```
  int dp[n + 1][n + 1];

  for (int i = 1; i <= n; ++i)
    dp[i][i] = 0;

  for (int len = 2; len <= n; ++len) {
    for (int i = 1, j = i + len - 1; j <= n; ++i, ++j) {
      dp[i][j] = oo;
      for (int k = i; k < j; ++k)
        dp[i][j] =
            min(dp[i][j], dp[i][k] + dp[k + 1][j] + p[i - 1] * p[k] * p[j]);
    }
  }

  return dp[1][n];
}
```

5. Geometry

5.1. Basic Operation.

```
#define x(c) real(c)
#define y(c) imag(c)
#define NEXT(i) (((i) + 1) % n)

const double EPS = 1e-7;
const int oo = (1 << 30);
typedef complex<double> point;

int cmp_double(double x, double y = 0) {
 return (x <= y + EPS) ? (x + EPS < y) ? -1 : 0 : 1;
}

bool cmp_point(const point &a, const point &b) {
 return (a.x() != b.x()) ? (cmp_double(a.x(), b.x()) == -1)
                   : (cmp_double(a.y(), b.y()) == -1);
}

bool operator<(const point &a, const point &b) { return cmp_point(a, b); }

// a1*b2 - a2*b1 = axb = |a||b|*sin()
double cross(const point &a, const point &b) { return imag(conj(a) * b); }

// a1*b1 + a2*b2 = a.b = |a||b|*cos(a,b)
double dot(const point &a, const point &b) { return real(conj(a) * b); }

int ccw(point a, point b, point c) {
 b -= a;
 c -= a;
 if (cross(b, c) > 0)
  return +1; // counter clockwise
 if (cross(b, c) < 0)
  return -1; // clockwise
 if (dot(b, c) < 0)
  return +2; // c - a - b on line
 if (cmp_double(norm(b), norm(c)) == -1)
  return -2; // a - b - c on line
 return 0; // a - c - b on line;
}

int cw(point a, point b, point c) { return -ccw(a, b, c); }

double sq(double x) { return x * x; }

double dist2(const point &a, const point &b) {
 return sq(a.x() - b.x()) + sq(a.y() - b.y());
}

double dist(const point &a, const point &b) { return abs(a - b); }

/*
Compares to 2D points by angle
Angle -90 is the first
Tested: LightOJ 1292
*/
bool polar_cmp(point a, point b) {
 if (a.x() >= 0 && b.x() < 0)
  return true;
 if (a.x() < 0 && b.x() >= 0)
  return false;
 if (a.x() == 0 && b.x() == 0) {
  if (a.y() > 0 && b.y() < 0)
   return false;
  if (a.y() < 0 && b.y() > 0)
   return true;
 }
 return cross(a, b) > 0;
}

// p-q-r: clockwise
double angle(point p, point q, point r) {
 point u = p - q, v = r - q;
 return atan2(cross(u, v), dot(u, v));
}

point rotateCCW90(point p) { return point(-p.imag(), p.real()); }

point rotate_by(const point &p, const point &about, double radians) {
 return (p - about) * exp(point(0, radians)) + about;
}
```

5.2. Circles.

```
struct circle {
 point center;
 double ratio;

 circle(point center, double ratio) : center(center), ratio(ratio) {}
};

// Tested [BAPC 2010 Clocks]
vector<point> circles_intersection(const circle &c1, const circle &c2) {
 vector<point> ret;
 double d = dist(c1.center, c2.center);
 if (d > c1.ratio + c2.ratio ||
    d + min(c1.ratio, c2.ratio) < max(c1.ratio, c2.ratio))
  return ret;
 double x = (d * d - c2.ratio * c2.ratio + c1.ratio * c1.ratio) / (2 * d);
 double y = sqrt(c1.ratio * c1.ratio - x * x);
 point v = (c2.center - c1.center) / d;
 ret.push_back(c1.center + v * x + rotateCCW90(v) * y);
 if (y > 0)
  ret.push_back(c1.center + v * x - rotateCCW90(v) * y);
 return ret;
}

// Interseccion Linea-Circulo
vector<point> intersectLC(line l, circle c) {
 point a = l[0], b = l[1];
 vector<point> ret;
 b = b - a;
 a = a - c.center;
 double A = dot(b, b);
 double B = dot(a, b);
 double C = dot(a, a) - c.ratio * c.ratio;
 double D = B * B - A * C;

 if (cmp(D) < 0)
  return ret;
 ret.push_back(c.center + a + b * (-B + sqrt(D + EPS)) / A);
 if (cmp(D) > 0)
  ret.push_back(c.center + a + b * (-B - sqrt(D)) / A);

 return ret;
}

/*
    Area of the intersection of a circle with a polygon
    Circle's center lies in (0,0)
    Polygon must be given counterclockwise
    Tested [Light OJ 1358]
*/

#define xx(_t) (xa + (_t)*a)
#define yy(_t) (ya + (_t)*b)

double radian(double xa, double ya, double xb, double yb) {
 return atan2(xa * yb - xb * ya, xa * xb + ya * yb);
}

double part(double xa, double ya, double xb, double yb, double r) {
 double l = sqrt((xa - xb) * (xa - xb) + (ya - yb) * (ya - yb));
 double a = (xb - xa) / l, b = (yb - ya) / l, c = a * xa + b * ya;
 double d = 4.0 * (c * c - xa * xa - ya * ya + r * r);
 if (d < EPS)
  return radian(xa, ya, xb, yb) * r * r * 0.5;
 else {
  d = sqrt(d) * 0.5;
  double s = -c - d, t = -c + d;
  if (s < 0.0)
   s = 0.0;
  else if (s > l)
   s = l;
  if (t < 0.0)
   t = 0.0;
  else if (t > l)
   t = l;
  return (xx(s) * yy(t) - xx(t) * yy(s) +
      (radian(xa, ya, xx(s), yy(s)) + radian(xx(t), yy(t), xb, yb)) * r *
        r) *
     0.5;
 }
}

double area_intersectionPC(polygon P, double r) {
 double s = 0.0;
 int n = (int)P.size();
 P.push_back(P[0]);
 for (int i = 0; i < n; ++i)
  s += part(P[i].x(), P[i].y(), P[NEXT(i)].x(), P[NEXT(i)].y(), r);
 return fabs(s);
}
```

```cpp
// circle tangents through point
vector<point> tangent(point p, circle C) {
 // not tested enough

 double D = abs(p - C.p);

 if (D + eps < C.r)
  return {};
 point t = C.p - p;

 double theta = asin(C.r / D);
 double d = cos(theta) * D;

 t = t / abs(t) * d;
 if (abs(D - C.r) < eps)
  return {p + t};
 point rot(cos(theta), sin(theta));
 return {p + t * rot, p + t * conj(rot)};
}

bool incircle(point a, point b, point c, point p) {
 a -= p;
 b -= p;
 c -= p;
 return norm(a) * cross(b, c) + norm(b) * cross(c, a) +
        norm(c) * cross(a, b) >=
      0;
 // < : inside, = cocircular, > outside
}

point three_point_circle(point a, point b, point c) {
 point x = 1.0 / conj(b - a), y = 1.0 / conj(c - a);
 return (y - x) / (conj(x) * y - x * conj(y)) + a;
}

/*
  Get the center of the circles that pass through p0 and p1
  and has ratio r.

  Be careful with epsilon.
*/
vector<point> two_point_ratio_circle(point p0, point p1, double r) {
 if (abs(p1 - p0) > 2 * r + eps) // Points are too far.
  return {};

 point pm = (p1 + p0) / 2.0l;
 point pv = p1 - p0;

 pv = point(-pv.imag(), pv.real());

 double x1 = p1.real(), y1 = p1.imag();
 double xm = pm.real(), ym = pm.imag();
 double xv = pv.real(), yv = pv.imag();

 double A = (sqr(xv) + sqr(yv));
 double C = sqr(xm - x1) + sqr(ym - y1) - sqr(r);
 double D = sqrt(-4 * A * C);
 double t = D / 2.0 / A;

 if (abs(t) <= eps)
  return {pm};

 return {c1, c2};
}
```

5.3. Closest Pair Points.

```cpp
/*
     Compute distance between closest points.

     Tested: AIZU(judge.u-aizu.ac.jp) CGL.5A
     Complexity: O(n log n)
*/

double closest_pair_points(vector<point> &P) {
 auto cmp = [](point a, point b) {
  return make_pair(a.imag(), a.real()) < make_pair(b.imag(), b.real());
 };

 int n = P.size();
 sort(P.begin(), P.end(), cmp_point);

 set<point, decltype(cmp)> S(cmp);
 const double oo = 1e9; // adjust
 double ans = oo;

 for (int i = 0, ptr = 0; i < n; ++i) {
  while (ptr < i && abs(P[i].real() - P[ptr].real()) >= ans)
   S.erase(P[ptr++]);
```

```
    auto lo = S.lower_bound(point(-oo, P[i].imag() - ans - eps));
    auto hi = S.upper_bound(point(-oo, P[i].imag() + ans + eps));

    for (decltype(lo) it = lo; it != hi; ++it)
      ans = min(ans, abs(P[i] - *it));
    S.insert(P[i]);
  }

  return ans;
}
```

5.4. Convex Cut.

```
/*
    Cut a convex polygon by a line and
    return the part to the left of the line

    Tested: AIZU(judge.u-aizu.ac.jp) CGL.4C
    Complexity: O(n)
*/

polygon convex_cut(const polygon &P, const line &l) {
  polygon Q;
  for (int i = 0, n = P.size(); i < n; ++i) {
    point A = P[i], B = P[(i + 1) % n];
    if (ccw(l.p, l.q, A) != -1)
      Q.push_back(A);
    if (ccw(l.p, l.q, A) * ccw(l.p, l.q, B) < 0)
      Q.push_back(crosspoint((line){A, B}, l));
  }
  return Q;
}
```

5.5. Convex Hull 3D.

```
// TODO: Change vec3 to use point3d from team reference

template <typename vtype> struct vec3 {
  vec3() { X[0] = X[1] = X[2] = 0; }
  vec3(vtype x, vtype y, vtype z) {
    X[0] = x;
    X[1] = y;
    X[2] = z;
  }

  /* 3D cross product */
  vec3 operator*(const vec3 &v) const {
    return vec3(X[1] * v.X[2] - X[2] * v.X[1], X[2] * v.X[0] - X[0] * v.X[2],
                X[0] * v.X[1] - X[1] * v.X[0]);
  }

  vec3 operator-(const vec3 &v) const {
    return vec3(X[0] - v.X[0], X[1] - v.X[1], X[2] - v.X[2]);
  }

  vec3 operator+(const vec3 &v) const {
    return vec3(X[0] + v.X[0], X[1] + v.X[1], X[2] + v.X[2]);
  }

  vec3 operator-() const { return vec3(-X[0], -X[1], -X[2]); }

  vec3 operator*(vtype d) const { return vec3(X[0] * d, X[1] * d, X[2] * d); }

  vtype dot(const vec3 &v) const {
    return X[0] * v.X[0] + X[1] * v.X[1] + X[2] * v.X[2];
  }

  bool operator!=(const vec3 v) {
    return X[0] != v.X[0] || X[1] != v.X[1] || X[2] != v.X[2];
  }

  void print() { cout << X[0] << "␣" << X[1] << "␣" << X[2] << endl; }

  bool zero() { return abs(X[0]) < eps && abs(X[1]) < eps && abs(X[2]) < eps; }

  bool notZero() {
    return abs(X[0]) > eps || abs(X[1]) > eps || abs(X[2]) > eps;
  }

  vtype X[3];
```

```
};

typedef vec3<double> point;

struct face {
 int idx[3];

 face() {}
 face(int i, int j, int k) { idx[0] = i, idx[1] = j, idx[2] = k; }

 int &operator[](int u) { return idx[u]; }
};

vector<point> read() {
 int n;
 cin >> n;
 vector<point> P(n);

 for (int i = 0; i < n; ++i) {
  double x, y, z;
  cin >> x >> y >> z;
  P[i] = point(x, y, z);
 }

 return P;
}

vector<face> convex_hull(vector<point> &cloud) {
 // bad

 int n = (int)cloud.size();

 point a = cloud[0], b = cloud[1];

 for (int i = 2; i < n; ++i) {
  point nr = (b - a) * (cloud[i] - a);

  if (nr.notZero()) {
   swap(cloud[i], cloud[2]);
   break;
  }
 }

 point c = (b - a) * (cloud[2] - a);

 for (int i = 3; i < n; ++i) {
  if (abs(c.dot(cloud[i] - a)) > eps) {
   swap(cloud[i], cloud[3]);
   break;
  }
 }

 vector<face> faces;

 function<point(face &)> normal = [&](face &f) {
  point a = cloud[f[1]] - cloud[f[0]];
  point b = cloud[f[2]] - cloud[f[0]];
  return a * b;
 };

 function<void(int, int, int)> add_face = [&](int x, int y, int z) {
  point a = cloud[x] * n, b = cloud[y] * n, c = cloud[z] * n;

  point nr = (b - a) * (c - a);

  for (int i = 0; i < n; ++i) {
   point d = cloud[i] - a;

   auto value = d.dot(nr);

   if (abs(value) > eps) {
    if (value > 0)
     swap(y, z);
    break;
   }
  }

  faces.push_back(face(x, y, z));
 };

 for (int i = 0; i < 4; ++i)
  for (int j = i + 1; j < 4; ++j)
   for (int k = j + 1; k < 4; ++k)
    add_face(i, j, k);

 for (int i = 4; i < n; ++i) {
  point x = cloud[i];

  vector<vi> seen(n, vi(n));
  vector<face> next_faces;

  for (auto f : faces) {
   if ((x - cloud[f[0]]).dot(normal(f)) > eps) {
    for (int u = 0; u < 3; ++u)
```

```
        for (int v = 0; v < 3; ++v)
          seen[f[u]][f[v]]++;
      } else
        next_faces.push_back(f);
    }

    faces.swap(next_faces);

    for (int j = 0; j < i; ++j)
      for (int k = j + 1; k < i; ++k) {
        if (seen[j][k] == 1)
          add_face(i, j, k);
      }
  }

  return faces;
}

int L[100];

vector<face> convex_hull_slow(vector<point> &cloud) {
  // good O(n^4)

  int n = (int)cloud.size();

  vector<face> faces;

  for (int i = 0; i < n; ++i)
    for (int j = i + 1; j < n; ++j)
      for (int k = j + 1; k < n; ++k) {

        point a = cloud[i], b = cloud[j], c = cloud[k];
        point nr = (b - a) * (c - a);

        int pnt = 0;
        L[pnt++] = j;
        L[pnt++] = k;

        bool proc = true;

        int v = 0, V = 0;

        for (int l = 0; l < n && proc; ++l) {
          if (l == i || l == j || l == k)
            continue;

          double t = nr.dot(cloud[l] - a);
          if (abs(t) < eps) {
            if (l < k)
              proc = false;
            else
              L[pnt++] = l;
          } else {
            if (t < 0)
              v = -1;
            else
              V = +1;
          }
        }

        if (!proc || v * V == -1)
          continue;

        //    cout << "tri: " << i << " " << j << " " <<
        //k
        //<< endl;     for (int l = 0; l < pnt; ++l)
        //cout << L[ l ] << " ";     cout << endl;

        function<bool(int, int)> compare = [&](int u, int v) {
          return nr.dot((cloud[u] - a) * (cloud[v] - a)) > 0;
        };

        sort(L, L + pnt, compare);

        for (int l = 0; l + 1 < pnt; ++l)
          faces.push_back(face(i, L[l], L[l + 1]));
      }

  return faces;
}

void mass_center(vector<point> &cloud, vector<face> &faces) {
  point pivot = cloud[0];

  double x = 0, y = 0, z = 0, v = 0;

  for (auto f : faces) {
    auto value = (cloud[f[0]] - pivot)
                 .dot((cloud[f[1]] - pivot) * (cloud[f[2]] - pivot));

    point sum = cloud[f[0]] + cloud[f[1]] + cloud[f[2]] + pivot;
    double cvol = abs(1. * value / 6);

    v += cvol;
```

```
    cvol /= 4;

    x += cvol * sum.X[0];
    y += cvol * sum.X[1];
    z += cvol * sum.X[2];
  }

  x /= v, y /= v, z /= v;

  // Mass center of a polyhedron at (x, y, z)
}
```

5.6. Lines.

```
struct line : public vector<point> {
  line(const point &a, const point &b) {
    if (a < b) {
      push_back(a);
      push_back(b);
    } else {
      push_back(b);
      push_back(a);
    }
  }
};

bool intersectLL(const line &l, const line &m) {
  return abs(cross(l[1] - l[0], m[1] - m[0])) > EPS || // non-parallel
       abs(cross(l[1] - l[0], m[0] - l[0])) < EPS; // same line
}

bool intersectLP(const line &l, const point &p) {
  return abs(cross(l[1] - p, l[0] - p)) < EPS;
}

point projectionPL(const point &p, const line &l) {
  double t = dot(p - l[0], l[0] - l[1]) / norm(l[0] - l[1]);
  return l[0] + t * (l[0] - l[1]);
}

point reflectPL(const point &p, const line &l) {
  point z = p - l[0];
  point w = l[1] - l[0];
  return conj(z / w) * w + l[0];
}

double distancePL(const point &p, const line &l) {
  return abs(p - projectionPL(p, l));
}

double distanceLL(const line &l, const line &m) {
  return intersectLL(l, m) ? 0 : distancePL(m[0], l);
}

// Punto interseccion recta recta
point crosspoint(const line &l, const line &m) {
  double A = cross(l[1] - l[0], m[1] - m[0]);
  double B = cross(l[1] - l[0], l[1] - m[0]);
  if (abs(A) < EPS && abs(B) < EPS)
    return m[0]; // Same line
  if (abs(A) < EPS)
    return point(0, 0); // parallels
  return m[0] + B / A * (m[1] - m[0]);
}

bool parallelLL(const line &l, const line &m) {
  return !cmp_double(cross(l[1] - l[0], m[0] - m[1]));
}

bool collinearLL(const line &l, const line &m) {
  return parallelLL(l, m) && !cmp_double(cross(l[0] - l[1], l[0] - m[0])) &&
       !cmp_double(cross(m[0] - m[1], m[0] - l[0]));
}
```

5.7. Minkowski.

```
/*
    Minkowski sum of two convex polygons. O(n + m)
    Note: Polygons MUST be counterclockwise
*/
```

```
polygon minkowski(polygon &A, polygon &B) {
 int na = (int)A.size(), nb = (int)B.size();

 if (A.empty() || B.empty())
  return polygon();

 rotate(A.begin(), min_element(A.begin(), A.end()), A.end());
 rotate(B.begin(), min_element(B.begin(), B.end()), B.end());

 int pa = 0, pb = 0;

 polygon M;

 while (pa < na && pb < nb) {
  M.push_back(A[pa] + B[pb]);
  double x = cross(A[(pa + 1) % na] - A[pa], B[(pb + 1) % nb] - B[pb]);
  if (x <= eps)
   pb++;
  if (-eps <= x)
   pa++;
 }

 while (pa < na)
  M.push_back(A[pa++] + B[0]);
 while (pb < nb)
  M.push_back(B[pb++] + A[0]);

 return M;
}
```

5.8. Point 3D.

```
const double pi = acos(-1.0);

// Construct a point on a sphere with center on the origin and radius R
// TESTED [COJ-1436]
struct point3d {
 double x, y, z;

 point3d(double x = 0, double y = 0, double z = 0) : x(x), y(y), z(z) {}

 double operator*(const point3d &p) const {
  return x * p.x + y * p.y + z * p.z;
 }

 point3d operator-(const point3d &p) const {
  return point3d(x - p.x, y - p.y, z - p.z);
 }
};

double abs(point3d p) { return sqrt(p.x * p.x + p.y * p.y + p.z * p.z); }

point3d from_polar(double lat, double lon, double R) {
 lat = lat / 180.0 * pi;
 lon = lon / 180.0 * pi;
 return point3d(R * cos(lat) * sin(lon), R * cos(lat) * cos(lon),
          R * sin(lat));
}

struct plane {
 double A, B, C, D;
};

double euclideanDistance(point3d p, point3d q) { return abs(p - q); }

/*
 Geodisic distance between points in a sphere
 R is the radius of the sphere
 */
double geodesic_distance(point3d p, point3d q, double r) {
 return r * acos(p * q / r / r);
}

const double eps = 1e-9;

// Find the rect of intersection of two planes on the space
// The rect is given parametrical
// TESTED [TIMUS 1239]
void planePlaneIntersection(plane p, plane q) {
 if (abs(p.C * q.B - q.C * p.B) < eps)
  return; // Planes are parallel

 double mz = (q.A * p.B - p.A * q.B) / (p.C * q.B - q.C * p.B);
 double nz = (q.D * p.B - p.D * q.B) / (p.C * q.B - q.C * p.B);

 double my = (q.A * p.C - p.A * q.C) / (p.B * q.C - p.C * q.B);
 double ny = (q.D * p.C - p.D * q.C) / (p.B * q.C - p.C * q.B);
```

```
  // parametric rect: (x, my * x + ny, mz * x * nz)
}
```

5.9. Polygon Triangulation.

```
typedef vector<point> triangle;

triangle make_triangle(const point &a, const point &b, const point &c) {
  triangle ret(3);
  ret[0] = a;
  ret[1] = b;
  ret[2] = c;
  return ret;
}

bool triangle_contains(const triangle &tri, const point &p) {
  return ccw(tri[0], tri[1], p) >= 0 && ccw(tri[1], tri[2], p) >= 0 &&
         ccw(tri[2], tri[0], p) >= 0;
}

bool ear_Q(int i, int j, int k, const polygon &P) {
  triangle tri = make_triangle(P[i], P[j], P[k]);
  if (ccw(tri[0], tri[1], tri[2]) <= 0)
    return false;
  for (int m = 0; m < (int)P.size(); ++m)
    if (m != i && m != j && m != k)
      if (triangle_contains(tri, P[m]))
        return false;
  return true;
}

void triangulate(const polygon &P, vector<triangle> &t) {
  const int n = P.size();
  vector<int> l, r;

  for (int i = 0; i < n; ++i) {
    l.push_back((i - 1 + n) % n);
    r.push_back((i + 1 + n) % n);
  }

  int i = n - 1;
  while ((int)t.size() < n - 2) {
    i = r[i];
    if (ear_Q(l[i], i, r[i], P)) {
      t.push_back(make_triangle(P[l[i]], P[i], P[r[i]]));
      l[r[i]] = l[i];
      r[l[i]] = r[i];
    }
  }
}

/*
        Perturbative deformation of a polygon.
        Each side of the polygon in counterclockwise
        polygon len making just the right translation.
*/
#define curr(P, i) P[i]
#define prev(P, i) P[((i - 1) + P.size()) % P.size()]
#define next(P, i) P[(i + 1) % P.size()]

polygon shrink_polygon(const polygon &P, double len) {
  polygon res;
  for (int i = 0; i < (int)P.size(); ++i) {
    point a = prev(P, i), b = curr(P, i), c = next(P, i);
    point u = (b - a) / abs(b - a);
    double th = arg((c - b) / u) * 0.5;
    point tmp(-sin(th), cos(th));
    res.push_back(b + u * tmp * len / cos(th));
  }
  return res;
}
```

5.10. Rectilinear MST.

```
/*
        Tested: USACO OPEN08 (Cow Neighborhoods)
        Complexity: O(n log n)
*/

typedef long long ll;
typedef complex<ll> point;
```

```
ll rectilinear_mst(vector<point> ps) {
 vector<int> id(ps.size());
 iota(id.begin(), id.end(), 0);

 struct edge {
  int src, dst;
  ll weight;
 };

 vector<edge> edges;
 for (int s = 0; s < 2; ++s) {
  for (int t = 0; t < 2; ++t) {
   sort(id.begin(), id.end(), [&](int i, int j) {
    return real(ps[i] - ps[j]) < imag(ps[j] - ps[i]);
   });

   map<ll, int> sweep;

   for (int i : id) {
    for (auto it = sweep.lower_bound(-imag(ps[i])); it != sweep.end();
        sweep.erase(it++)) {
     int j = it->second;
     if (imag(ps[j] - ps[i]) < real(ps[j] - ps[i]))
      break;
     ll d = abs(real(ps[i] - ps[j])) + abs(imag(ps[i] - ps[j]));
     edges.push_back({i, j, d});
    }
    sweep[-imag(ps[i])] = i;
   }

   for (auto &p : ps)
    p = point(imag(p), real(p));
  }

  for (auto &p : ps)
   p = point(-real(p), imag(p));
 }

 ll cost = 0;
 sort(edges.begin(), edges.end(),
    [](edge a, edge b) { return a.weight < b.weight; });

 union_find uf(ps.size());
 for (edge e : edges)
  if (uf.join(e.src, e.dst))
   cost += e.weight;

 return cost;
}
```

5.11. Rotating Calipers.

```
/*
      Gets all the antipodal pair of points
      Time: O(n)
*/
#define NEXT(i) (((i) + 1) % n)
double area (point a, point b, point c) //2 * area
{
      return abs(cross(b - a, c - a));
}

vector<pair<int, int> > antipodal_pairs (polygon &P)
{
      vector<pair<int, int> > ans;
      int n = P.size();

      if (P.size() == 2)
            ans.push_back(make_pair(0, 1));

      if (P.size() < 3)
            return ans;

      int q0 = 0;

      while (area(P[n - 1], P[0], P[NEXT(q0)]) >
                  area(P[n - 1], P[0], P[q0]))
            ++q0;

      for (int q = q0, p = 0; q != 0 && p <= q0; ++p)
      {
            ans.push_back(make_pair(p, q));

            while (area(P[p], P[NEXT(p)], P[NEXT(q)]) >
                        area(P[p], P[NEXT(p)], P[q]))
            {
                  q = NEXT(q);
```

```
            if (p == q0 && q == 0)
                return ans;

            ans.push_back(make_pair(p, q));
        }

        if (area(P[p], P[NEXT(p)], P[NEXT(q)]) ==
                area(P[p], P[NEXT(p)], P[q]))
        {
            if (p != q0 || q != n - 1)
                ans.push_back(make_pair(p, NEXT(q)));
            else
                ans.push_back(make_pair(NEXT(p), q));
        }
    }

    return ans;
}


/*
    Gets the farthest pair of points of the given points.
    (maybe TLE using double)
    TESTED [POJ 2187]
*/
pair<point, point> farthest_pair (polygon &P)
{
    P = convex_hull(P);
    vector<pair<int, int> > pairs = antipodal_pairs(P);

    double best = 0;
    pair<point, point> ans;

    for (int i = 0; i < (int)pairs.size(); ++i)
    {
        point p1 = P[pairs[i].first];
        point p2 = P[pairs[i].second];

        double dist = norm(p1-p2);

        if (dist > best)
        {
            best = dist;
            ans = make_pair(p1, p2);
        }
    }

    return ans;
}


/*
    Gets the minimum distance between parallel lines of
    support of the convex polygon P
    Time: O(n)
*/

double check (int a, int b, int c, int d, polygon &P)
{
    for (int i = 0; i < 4 && a != c; ++i)
    {
        if (i == 1)
            swap(a, b);
        else
            swap(c, d);
    }
    if (a == c) //a admits a support line parallel to bd
    {
        //assert(b != d)
        double A = area(P[a], P[b], P[d]); //double of the triangle area
        double base = abs(P[b] - P[d]); //base of the triangle abd
        return A / base;
    }
    return oo;
}

double polygon_width (polygon &P)
{
    if (P.size() < 3)
        return 0;

    vector<pair<int, int> > pairs = antipodal_pairs(P);
    double best = oo;
    int n = pairs.size();

    for (int i = 0; i < n; ++i)
    {
        double tmp = check(pairs[i].first, pairs[i].second,
                    pairs[NEXT(i)].first,
                    pairs[NEXT(i)].second, P);
        best = min(best, tmp);
    }
    return best;
}
```

5.12. Segment Intersect.

```
#define _GLIBCXX_DEBUG
#include <algorithm>
#include <complex>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <stack>
#include <stdio.h>
#include <string.h>
#include <string>
#include <vector>
using namespace std;
#define REP(i, n) for (int i = 0; i < (int)n; ++i)
#define FOR(i, c) \
 for (__typeof((c).begin()) i = (c).begin(); i != (c).end(); ++i)
#define ALL(c) (c).begin(), (c).end()
#define Y(c) imag(c)
#define X(c) real(c)
#define INF 100000000
// Graph Only
typedef int Weight;
struct Edge {
 int src, dst;
 Weight weight;
 Edge(int src, int dst, Weight weight) : src(src), dst(dst), weight(weight) {}
};
bool operator<(const Edge &e, const Edge &f) {
 return e.weight != f.weight ? e.weight > f.weight
      : e.src != f.src ? e.src < f.src
                      : e.dst < f.dst;
}
typedef vector<Edge> Edges;
typedef vector<Edges> Graph;
typedef vector<Weight> Array;
typedef vector<Array> Matrix;
#define P complex<double>
typedef vector<P> Pol;
bool operator<(const P &a, const P &b) {
 return X(a) != X(b) ? X(a) < X(b) : Y(a) < Y(b);
}
struct L : public vector<P> {
 L(const P &a, const P &b) {
  if (a < b) {
   push_back(a);
   push_back(b);
  } else {
   push_back(b);
   push_back(a);
  }
 }
};
const double EPS = 1e-8, oo = 1e12;

bool op_min(const P &a, const P &b) {
 return X(a) != X(b) ? X(a) < X(b) : Y(a) < Y(b);
}

double cross(P a, P b) { return Y(conj(a) * b); }
double dot(P a, P b) { return X(conj(a) * b); }

int ccw(P a, P b, P c) { // Orientacion de 3 puntos
 b -= a;
 c -= a;
 if (cross(b, c) > 0)
  return +1; // counter clockwise
 if (cross(b, c) < 0)
  return -1; // clockwise
 if (dot(b, c) < 0)
  return +2; // c - a - b line
 if (norm(b) < norm(c))
  return -2; // a - b - c line
 return 0;
}

bool intersectSS(L s, L t) { // Inters de 2 segm
 if (abs(s[0] - t[0]) < EPS || abs(s[0] - t[1]) < EPS ||
   abs(s[1] - t[0]) < EPS || abs(s[1] - t[1]) < EPS)
  return 1; // Puntos Iguales
 return ccw(s[0], s[1], t[0]) * ccw(s[0], s[1], t[1]) <= 0 &&
     ccw(t[0], t[1], s[0]) * ccw(t[0], t[1], s[1]) <= 0;
}

P crosspoint(L l, L m) { // Punto inters /2 rectas
 double A = cross(l[1] - l[0], m[1] - m[0]);
 double B = cross(l[1] - l[0], l[1] - m[0]);
 if (abs(A) < EPS && abs(B) < EPS)
  return m[0]; // Same L
 if (abs(A) < EPS)
  return P(0, 0); // parallels
```

```
  return m[0] + B / A * (m[1] - m[0]);
}

struct event {
  double x;
  int type;
  L seg;
  event(double x, int type, const L &seg) : x(x), type(type), seg(seg) {}
  bool operator<(const event &e) const {
    return x != e.x ? x > e.x : type > e.type;
  }
};
struct segComp {
  bool operator()(const L &a, const L &b) {
    if (a[0] < b[0])
      return true;
    if (a[1] < b[1])
      return true;
    return false;
  }
};
int segment_intersects(const vector<L> &segs, vector<P> &out) {
  priority_queue<event> Q;
  for (int i = 0; i < segs.size(); ++i) {
    double x1 = real(segs[i][0]), x2 = real(segs[i][1]);
    Q.push(event(min(x1, x2), 0, segs[i]));
    Q.push(event(max(x1, x2), 1, segs[i]));
  }
  int count = 0;
  set<L, segComp> T;
  while (!Q.empty()) {
    event e = Q.top();
    Q.pop();
    if (e.type == 0) {
      for (set<L, segComp>::iterator itr = T.begin(); itr != T.end(); ++itr)
        if (intersectSS(*itr, e.seg)) {
          out.push_back(crosspoint(*itr, e.seg));
          ++count;
        }
      T.insert(e.seg);
    } else
      T.erase(e.seg);
  }
  return count;
}

bool merge_if_able(L &s, L t) {
  if (abs(cross(s[1] - s[0], t[1] - t[0])) > EPS)
    return false;
  if (ccw(s[0], t[0], s[1]) == +1 || ccw(s[0], t[0], s[1]) == -1)
    return false; // not on the same line
  if (ccw(s[0], s[1], t[0]) == -2 || ccw(t[0], t[1], s[0]) == -2)
    return false; // separated
  s = L(min(s[0], t[0], op_min), max(s[1], t[1], op_min));
  return true;
}
void merge_segments(vector<L> &segs) {
  for (int i = 0; i < segs.size(); ++i)
    for (int j = i + 1; j < segs.size(); ++j)
      if (merge_if_able(segs[i], segs[j]))
        segs[j--] = segs.back(), segs.pop_back();
}

pair<P, P> closestPair(vector<P> &p) {
  int n = p.size(), s = 0, t = 1, m = 2, S[n];
  S[0] = 0, S[1] = 1;
  sort(ALL(p), op_min); //"p<q"<=>"px<qx"
  double d = norm(p[s] - p[t]);
  for (int i = 2; i < n; S[m++] = i++)
    REP(j, m) {
      if (norm(p[S[j]] - p[i]) < d)
        d = norm(p[s = S[j]] - p[t = i]);
      if (real(p[S[j]]) < real(p[i]) - d)
        S[j--] = S[--m];
    }
  return make_pair(p[s], p[t]);
}

int main() {
  P p1(0, 1);
  P p2(-1, 0);
  P p3(0, 2);
  P p4(4, 2);
  P p5(1, 0);
  P p6(-1, 0);
  vector<L> segs;
  vector<P> out;
  segs.push_back(L(p1, p2));
  segs.push_back(L(p3, p4));
  segs.push_back(L(p5, p6));
  cout << segment_intersects(segs, out) << endl;
  FOR(i, out) cout << *i << endl;
  return 0;
}
```

5.13. Segments.

```
// Interseccion recta y segmento
bool intersectLS(const line &l, const line &s) {
 return cross(l[1] - l[0], s[0] - l[0]) * // s[0] is left of l
          cross(l[1] - l[0], s[1] - l[0]) <
      EPS; // s[1] is right of l
}

bool intersectSS(const line &s, const line &t) {
 return ccw(s[0], s[1], t[0]) * ccw(s[0], s[1], t[1]) <= 0 &&
      ccw(t[0], t[1], s[0]) * ccw(t[0], t[1], s[1]) <= 0;
}

bool intersectPS(const point &p, const line &s) {
 return abs(s[0] - p) + abs(s[1] - p) - abs(s[1] - s[0]) <
      EPS; // triangle inequality
}

double distanceLS(const line &l, const line &s) {
 if (intersectLS(l, s))
  return 0;
 return min(distancePL(s[0], l), distancePL(s[1], l));
}

double distancePS(const point &p, const line &s) {
 const point r = projectionPL(p, s);
 if (intersectPS(r, s))
  return abs(r - p);
 return min(abs(s[0] - p), abs(s[1] - p));
}

double distanceSS(const line &s, const line &t) {
 if (intersectSS(s, t))
  return 0;
 return min(min(distancePS(t[0], s), distancePS(t[1], s)),
          min(distancePS(s[0], t), distancePS(s[1], t)));
}

point projectionPS(const point &p, const line &l) {
 double r = dot(l[1] - l[0], l[1] - l[0]);
 if (cmp_double(r, 0) == 0)
  return l[0];
 r = dot(p - l[0], l[1] - l[0]) / r;
 if (r < 0)
  return l[0];
 if (r > 1)
  return l[1];
 return l[0] + (l[1] - l[0]) * r;
}

bool merge_if_able(line &s, line t) {
 if (abs(cross(s[1] - s[0], t[1] - t[0])) > EPS)
  return false;

 if (ccw(s[0], t[0], s[1]) == +1 || ccw(s[0], t[0], s[1]) == -1)
  return false; // nsame line

 if (ccw(s[0], s[1], t[0]) == -2 || ccw(t[0], t[1], s[0]) == -2)
  return false; // separated

 s = line(min(s[0], t[0], cmp_point), max(s[1], t[1], cmp_point));

 return true;
}

/*
    Tested: STRAZA
    Contest 2 - COCI 2006-2007
*/
void merge_segments(vector<line> &segs) {
 bool changed = true;
 while (changed) {
  changed = false;
  for (int i = 0; i < (int)segs.size(); ++i)
   for (int j = i + 1; j < (int)segs.size(); ++j) {
    line a = segs[i], b = segs[j];
    if (merge_if_able(segs[i], segs[j])) {
     changed = true;
     segs.erase(segs.begin() + j);
     break;
    }
   }
 }
}
```

5.14. Semiplane Intersection.

```
/*
      Check wether there is a point in the intersection of
      several semi-planes. if p lies in the border of some
      semiplane it is considered to belong to the semiplane.

      Expected Running time: linear

      Tested on Triathlon [Cuban Campament Contest]
*/

bool intersect(vector<line> semiplane) {

  function<bool(line &, point &)> side = [](line &l, point &p) {
    // IMPORTANT: point p belongs to semiplane defined by l
    // iff p it's clockwise respect to segment < l.p, l.q >
    // i.e. (non negative cross product)

    return cross(l.q - l.p, p - l.p) >= 0;
  };

  function<bool(line &, line &, point &)> crosspoint =
      [](const line &l, const line &m, point &x) {
        double A = cross(l.q - l.p, m.q - m.p);
        double B = cross(l.q - l.p, l.q - m.p);
        if (abs(A) < eps)
          return false;
        x = m.p + B / A * (m.q - m.p);
        return true;
      };

  int n = (int)semiplane.size();

  random_shuffle(semiplane.begin(), semiplane.end());

  point cent(0, 1e9);

  for (int i = 0; i < n; ++i) {
    line &S = semiplane[i];

    if (side(S, cent))
      continue;

    point d = S.q - S.p;
    d /= abs(d);

    point A = S.p - d * 1e8, B = S.p + d * 1e8;

    for (int j = 0; j < i; ++j) {
      point x;
      line &T = semiplane[j];

      if (crosspoint(T, S, x)) {
        int cnt = 0;

        if (!side(T, A)) {
          A = x;
          cnt++;
        }

        if (!side(T, B)) {
          B = x;
          cnt++;
        }

        if (cnt == 2)
          return false;
      } else {
        if (!side(T, A))
          return false;
      }
    }

    if (imag(B) > imag(A))
      swap(A, B);
    cent = A;
  }

  return true;
}
```

5.15. Triangles.

```
double area_heron(double const &a, double const &b, double const &c) {
  double s = (a + b + c) / 2;
  return sqrt(s * (s - a) * (s - b) * (s - c));
}

double circumradius(const double &a, const double &b, const double &c) {
  return a * b * c / 4 / area_heron(a, b, c);
}

double inradius(const double &a, const double &b, const double &c) {
  return 2 * area_heron(a, b, c) / (a + b + c);
}

/*
Center of the circumference of a triangle
[Tested COJ 1572 - Joining the Centers]
*/
point circunference_center(point a, point b, point c) {
  point x = 1.0 / conj(b - a), y = 1.0 / conj(c - a);
  return (y - x) / (conj(x) * y - x * conj(y)) + a;
}

bool circunference_center(point &a, point &b, point &c, point &r) {
  double d = (a.x() * (b.y() - c.y()) + b.x() * (c.y() - a.y()) +
            c.x() * (a.y() - b.y())) *
           2.0;
  if (fabs(d) < EPS)
    return false;
  r.x() = ((a.x() * a.x() + a.y() * a.y()) * (b.y() - c.y()) +
          (b.x() * b.x() + b.y() * b.y()) * (c.y() - a.y()) +
          (c.x() * c.x() + c.y() * c.y()) * (a.y() - b.y())) /
         d;
  r.y() = -((a.x() * a.x() + a.y() * a.y()) * (b.x() - c.x()) +
           (b.x() * b.x() + b.y() * b.y()) * (c.x() - a.x()) +
           (c.x() * c.x() + c.y() * c.y()) * (a.x() - b.x())) /
          d;
  return true;
}

/*
//Interseccion de las bisectrices
double incenter(vect &a,vect &b,vect &c,vect &r)
{
	double u=(b-c).length(),v=(c-a).length(),w=(a-b).length(),s=u+v+w;
	if(s<EPS) {r=a;return 0.0;}
	r.x=(a.x*u+b.x*v+c.x*w)/s;
	r.y=(a.y*u+b.y*v+c.y*w)/s;
	return sqrt((v+w-u)*(w+u-v)*(u+v-w)/s)*0.5;
}

//Interseccion de las alturas
bool orthocenter(vect &a,vect &b,vect &c,vect &r)
{
	double d=a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
	if(fabs(d)<EPS) return false;
	r.x=((c.x*b.x+c.y*b.y)*(c.y-b.y)+(a.x*c.x+a.y*c.y)*(a.y-c.y)
		+(b.x*a.x+b.y*a.y)*(b.y-a.y))/d;
	r.y=-((c.x*b.x+c.y*b.y)*(c.x-b.x)+(a.x*c.x+a.y*c.y)*(a.x-c.x)
		+(b.x*a.x+b.y*a.y)*(b.x-a.x))/d;
	return true;
}
*/

double signed_area(const point &p1, const point &p2, const point &p3) {
  return cross(p2 - p1, p3 - p1);
}
double triangle_area(const point &a, const point &b, const point &c) {
  return 0.5 * abs(cross(b - a, c - a));
}
```

6. Graphs

6.1. Articulation Point And Bridge.

```
const int MaxV = 10005;

enum { White, Gray, Black };

vi g[MaxV];
int d[MaxV], low[MaxV], pi[MaxV];
int step = 0;
bool puntoArticulacion[MaxV];
set<pii> aristaPuente;
int dfsRoot, rootChildren;

int n, m;

void DFS(int u) {
  low[u] = d[u] = ++step;
  REP(i, g[u].size()) {
    int v = g[u][i];
    if (d[v] == White) {
      pi[v] = u;
      if (u == dfsRoot)
        ++rootChildren;

      DFS(v);
      if (low[v] >= d[u]) // for articulation point
        puntoArticulacion[u] = true;
      if (low[v] > d[u]) // for bridge
        aristaPuente.insert(pii(u, v));

      low[u] = min(low[u], low[v]);
    } else if (v != pi[u])
      low[u] = min(low[u], d[v]);
  }
}

void articulationPointAndBridge() {
  step = 0;
  REP(i, n - 1) {
    if (d[i] == White) {
      dfsRoot = i;
      rootChildren = 0;
      DFS(i);
      puntoArticulacion[dfsRoot] = (rootChildren > 1);
    }
  }
}
```

6.2. Bellman-Ford.

```
int n, m;

const int MaxN = 1000;

vector<pii> g[MaxN];

struct Edge {

  int src, dst, weight;
  Edge(int a, int b, int c) : src(a), dst(b), weight(c) {}
};

vector<Edge> edges;
vector<int> dist;

bool Bellman_Ford(int s) {
  dist = vector<int>(n, oo);
  dist[s] = 0;

  for (int i = 0; i < n - 1; ++i) {
    foreach (e, edges) {
      int u = e->src;
      int v = e->dst;
      int w = e->weight;

      if (dist[u] + w < dist[v])
        dist[v] = dist[u] + w;
    }
  }
```

```
  foreach (e, edges) {
    int u = e->src;
    int v = e->dst;
    int w = e->weight;
      if (dist[u] + w < dist[v])
        return false; // negative_cycle_exist
    }
    return true;
}
```

6.3. Biconnected Components.

```
const int MaxN = 10000;

int n, m;
vector<int> g[MaxN];
int d[MaxN], low[MaxN], pi[MaxN];
int step;

stack<pii> bicon;

void BiconComp(int u) {
  d[u] = low[u] = ++step;
  REP(i, g[u].size()) {
    int w = g[u][i];
    if (w != pi[u] && d[w] < d[u]) // foward edge
    {
      bicon.push(pii(u, w));
      if (d[w] == 0) {
        BiconComp(w);
        low[u] = min(low[u], low[w]);
        if (low[w] >= d[u]) {
          printf("New Biconnected Component:\n");
          pii tmp;
          do {
            tmp = bicon.top();
            bicon.pop();
            printf("%d %d\n", tmp.F, tmp.S);
          } while (!(tmp.F == u && tmp.S == w));
          printf("\n");
        }
      } else if (w != pi[u]) // back edge
        low[u] = min(low[u], d[w]);
    }
  }
}

void init() {
  step = 0;
  REP(i, n) {
    d[i] = low[i] = 0;
    pi[i] = -1;
  }
}
```

6.4. Bipartite Matching.

```
/*
    Tested: AIZU(judge.u-aizu.ac.jp) GRL_7_A
    Complexity: O(nm)
*/

struct graph {
  int L, R;
  vector<vector<int>> adj;

  graph(int L, int R) : L(L), R(R), adj(L + R) {}

  void add_edge(int u, int v) {
    adj[u].push_back(v + L);
    adj[v + L].push_back(u);
  }

  int maximum_matching() {
    vector<int> visited(L), mate(L + R, -1);
    function<bool(int)> augment = [&](int u) {
      if (visited[u])
        return false;
      visited[u] = true;
      for (int w : adj[u]) {
        int v = mate[w];
        if (v < 0 || augment(v)) {
          mate[u] = w;
```

```
        mate[w] = u;
        return true;
      }
    }
    return false;
  };
  int match = 0;
  for (int u = 0; u < L; ++u) {
    fill(visited.begin(), visited.end(), 0);
    if (augment(u))
      ++match;
  }
  return match;
 }
};
```

6.5. Centroid Decomposition.

```
/*
  Centroid decomposition of a tree.
  Find the centroid of the subtree that contains node c.

  Nodes availables are those which aren't marked, i.e mk[u] == False
*/

vi adj[maxn];
bool mk[maxn];
int q[maxn], p[maxn], sz[maxn], mc[maxn];

int centroid(int c) {
 int b = 0, e = 0;
 q[e++] = c, p[c] = -1, sz[c] = 1, mc[c] = 0;

 while (b < e) {
  int u = q[b++];
  for (auto v : adj[u])
   if (v != p[u] && !mk[v])
    p[v] = u, sz[v] = 1, mc[v] = 0, q[e++] = v;
 }

 for (int i = e - 1; ~i; --i) {
  int u = q[i];
  int bc = max(e - sz[u], mc[u]);
  if (2 * bc <= e)
   return u;
  sz[p[u]] += sz[u], mc[p[u]] = max(mc[p[u]], sz[u]);
 }

 assert(false);
 return -1;
}
```

6.6. Dijkstra.

```
const int MaxN = 1000;

int n, m;

vector<pii> g[MaxN];
int pi[MaxN];

priority_queue<pii, vector<pii>, greater<pii>> pq;

vector<int> Dijkstra(int s) {
 vector<int> d(n, oo);
 pq = priority_queue<ii, vector<ii>, greater<ii>>();

 d[s] = 0;
 pq.push(ii(0, s));

 while (!pq.empty()) {
  int dist = pq.top().F, u = pq.top().S;
  pq.pop();

  if (dist == d[u]) {
   for (int i = 0; i < (int)g[u].size(); ++i) {
    int v = g[u][i].S;
    int w = g[u][i].F;
    if (d[u] + w < d[v]) {
     d[v] = d[u] + w;
     pq.push(ii(d[v], v));
    }
```

```
      }
    }
  }
```

```
  return d;
}
```

6.7. Dominator Tree.

```
/*
    Dominator Tree (Lengauer-Tarjan)

    Tested: SPOJ EN
    Complexity: O(m log n)
*/

struct graph {
  int n;
  vector<vector<int>> adj, radj;

  graph(int n) : n(n), adj(n), radj(n) {}

  void add_edge(int src, int dst) {
    adj[src].push_back(dst);
    radj[dst].push_back(src);
  }

  vector<int> rank, semi, low, anc;

  int eval(int v) {
    if (anc[v] < n && anc[anc[v]] < n) {
      int x = eval(anc[v]);
      if (rank[semi[low[v]]] > rank[semi[x]])
        low[v] = x;
      anc[v] = anc[anc[v]];
    }
    return low[v];
  }

  vector<int> prev, ord;

  void dfs(int u) {
    rank[u] = ord.size();
    ord.push_back(u);
    for (auto v : adj[u]) {
      if (rank[v] < n)
        continue;
      dfs(v);
      prev[v] = u;
    }
  }

  vector<int> idom; // idom[u] is an immediate dominator of u

  void dominator_tree(int r) {
    idom.assign(n, n);
    prev = rank = anc = idom;
    semi.resize(n);
    iota(semi.begin(), semi.end(), 0);
    low = semi;
    ord.clear();
    dfs(r);

    vector<vector<int>> dom(n);
    for (int i = (int)ord.size() - 1; i >= 1; --i) {
      int w = ord[i];
      for (auto v : radj[w]) {
        int u = eval(v);
        if (rank[semi[w]] > rank[semi[u]])
          semi[w] = semi[u];
      }
      dom[semi[w]].push_back(w);
      anc[w] = prev[w];
      for (int v : dom[prev[w]]) {
        int u = eval(v);
        idom[v] = (rank[prev[w]] > rank[semi[u]] ? u : prev[w]);
      }
      dom[prev[w]].clear();
    }

    for (int i = 1; i < (int)ord.size(); ++i) {
      int w = ord[i];
      if (idom[w] != semi[w])
        idom[w] = idom[idom[w]];
    }
  }

  vector<int> dominators(int u) {
    vector<int> S;
```

```
  for (; u < n; u = idom[u])
   S.push_back(u);
  return S;
 }
};
```

6.8. Flow with Lower Bound.

```
/*
    Flow with lower bound

    Tested: ZOJ 3229
    Complexity: O(n^2 m)
*/

template <typename T> struct dinic {
 struct edge {
  int src, dst;
  T low, cap, flow;
  int rev;
 };

 int n;
 vector<vector<edge>> adj;

 dinic(int n) : n(n), adj(n + 2) {}

 void add_edge(int src, int dst, T low, T cap) {
  adj[src].push_back({src, dst, low, cap, 0, (int)adj[dst].size()});
  if (src == dst)
   adj[src].back().rev++;
  adj[dst].push_back({dst, src, 0, 0, 0, (int)adj[src].size() - 1});
 }

 vector<int> level, iter;

 T augment(int u, int t, T cur) {
  if (u == t)
   return cur;
  for (int &i = iter[u]; i < (int)adj[u].size(); ++i) {
   edge &e = adj[u][i];
   if (e.cap - e.flow > 0 && level[u] > level[e.dst]) {
    T f = augment(e.dst, t, min(cur, e.cap - e.flow));
    if (f > 0) {
     e.flow += f;
     adj[e.dst][e.rev].flow -= f;
     return f;
    }
   }
  }
  return 0;
 }

 int bfs(int s, int t) {
  level.assign(n + 2, n + 2);
  level[t] = 0;
  queue<int> Q;
  for (Q.push(t); !Q.empty(); Q.pop()) {
   int u = Q.front();
   if (u == s)
    break;
   for (edge &e : adj[u]) {
    edge &erev = adj[e.dst][e.rev];
    if (erev.cap - erev.flow > 0 && level[e.dst] > level[u] + 1) {
     Q.push(e.dst);
     level[e.dst] = level[u] + 1;
    }
   }
  }
  return level[s];
 }

 const T oo = numeric_limits<T>::max();

 T max_flow(int source, int sink) {
  vector<T> delta(n + 2);

  for (int u = 0; u < n; ++u) // initialize
   for (auto &e : adj[u]) {
    delta[e.src] -= e.low;
    delta[e.dst] += e.low;
    e.cap -= e.low;
    e.flow = 0;
   }

  T sum = 0;
  int s = n, t = n + 1;
```

```
    for (int u = 0; u < n; ++u) {
      if (delta[u] > 0) {
        add_edge(s, u, 0, delta[u]);
        sum += delta[u];
      } else if (delta[u] < 0)
        add_edge(u, t, 0, -delta[u]);
    }

    add_edge(sink, source, 0, oo);
    T flow = 0;

    while (bfs(s, t) < n + 2) {
      iter.assign(n + 2, 0);
      for (T f; (f = augment(s, t, oo)) > 0;)
        flow += f;
    }

    if (flow != sum)
      return -1; // no solution

    for (int u = 0; u < n; ++u)
      for (auto &e : adj[u]) {
        e.cap += e.low;
        e.flow += e.low;
        edge &erev = adj[e.dst][e.rev];
        erev.cap -= e.low;
        erev.flow -= e.low;
      }

    adj[sink].pop_back();
    adj[source].pop_back();

    while (bfs(source, sink) < n + 2) {
      iter.assign(n + 2, 0);
      for (T f; (f = augment(source, sink, oo)) > 0;)
        flow += f;
    } // level[u] == n + 2 ==> s-side

    return flow;
  }
};
```

6.9. Floyd Warshall.

```
const int
      MaxN = 10000;

int n, m;
int g[MaxN][MaxN];
int dist[MaxN][MaxN];

void Floyd_Warshall()
{
      for(int k = 0; k < n; ++k)
            for(int i = 0; i < n; ++i)
                  for(int j = 0; j < n; ++j)
                        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}

int init()
{
      REP(i, n) REP(j, n)
      {
            if(dist[i][j] == 0)
            {
                  dist[i][j] = oo;
                  g[i][j] = oo;
            }
      }

      for(int i = 0; i < n; ++i)
            dist[i][i] = 0;
}
```

6.10. Gabow Edmonds.

```
/*
      Tested: Timus 1099
      Complexity: O(n^3)
*/

struct graph {
```

```
  int n;
  vector<vector<int>> adj;

  graph(int n) : n(n), adj(n) {}

  void add_edge(int u, int v) {
   adj[u].push_back(v);
   adj[v].push_back(u);
  }

  queue<int> q;
  vector<int> label, mate, cycle;

  void rematch(int x, int y) {
   int m = mate[x];
   mate[x] = y;
   if (mate[m] == x) {
    if (label[x] < n)
     rematch(mate[m] = label[x], m);
    else {
     int s = (label[x] - n) / n, t = (label[x] - n) % n;
     rematch(s, t);
     rematch(t, s);
    }
   }
  }

  void traverse(int x) {
   vector<int> save = mate;
   rematch(x, x);
   for (int u = 0; u < n; ++u)
    if (mate[u] != save[u])
     cycle[u] ^= 1;
   save.swap(mate);
  }

  void relabel(int x, int y) {
   cycle = vector<int>(n, 0);
   traverse(x);
   traverse(y);
   for (int u = 0; u < n; ++u) {
    if (!cycle[u] || label[u] >= 0)
     continue;
    label[u] = n + x + y * n;
    q.push(u);
   }
  }

  int augment(int r) {
   label.assign(n, -2);
   label[r] = -1;
   q = queue<int>();
   for (q.push(r); !q.empty(); q.pop()) {
    int x = q.front();
    for (int y : adj[x]) {
     if (mate[y] < 0 && r != y) {
      rematch(mate[y] = x, y);
      return 1;
     } else if (label[y] >= -1)
      relabel(x, y);
     else if (label[mate[y]] < -1) {
      label[mate[y]] = x;
      q.push(mate[y]);
     }
    }
   }
   return 0;
  }

  int maximum_matching() {
   mate.assign(n, -2);
   int matching = 0;
   for (int u = 0; u < n; ++u)
    if (mate[u] < 0)
     matching += augment(u);
   return matching;
  }
};
```

6.11. Gomory hu Tree.

```
/*
    Gomory-Hu tree

    Tested: SPOj MCQUERY

    Complexity: O(n-1) max-flow call
*/

template <typename flow_type> struct edge {
```

```
  int src, dst;
  flow_type cap;
};

template <typename flow_type>
vector<edge<flow_type>> gomory_hu(dinic<flow_type> &adj) {
  int n = adj.n;

  vector<edge<flow_type>> tree;
  vector<int> parent(n);
  for (int u = 1; u < n; ++u) {
    tree.push_back({u, parent[u], adj.max_flow(u, parent[u])});
    for (int v = u + 1; v < n; ++v)
      if (adj.level[v] == -1 && parent[v] == parent[u])
        parent[v] = u;
  }

  return tree;
}
```

6.12. **Hopcroft Karp.**

```
/*
      Tested: SPOJ MATCHING
      Complexity: O(m n^0.5)
*/

struct graph {
  int L, R;
  vector<vector<int>> adj;

  graph(int L, int R) : L(L), R(R), adj(L + R) {}

  void add_edge(int u, int v) {
    adj[u].push_back(v + L);
    adj[v + L].push_back(u);
  }

  int maximum_matching() {
    vector<int> level(L), mate(L + R, -1);

    function<bool(void)> levelize = [&]() {
      queue<int> Q;
      for (int u = 0; u < L; ++u) {
        level[u] = -1;
        if (mate[u] < 0) {
          level[u] = 0;
          Q.push(u);
        }
      }
      while (!Q.empty()) {
        int u = Q.front();
        Q.pop();
        for (int w : adj[u]) {
          int v = mate[w];
          if (v < 0)
            return true;
          if (level[v] < 0) {
            level[v] = level[u] + 1;
            Q.push(v);
          }
        }
      }
      return false;
    };

    function<bool(int)> augment = [&](int u) {
      for (int w : adj[u]) {
        int v = mate[w];
        if (v < 0 || (level[v] > level[u] && augment(v))) {
          mate[u] = w;
          mate[w] = u;
          return true;
        }
      }
      return false;
    };
    int match = 0;
    while (levelize())
      for (int u = 0; u < L; ++u)
        if (mate[u] < 0 && augment(u))
          ++match;
    return match;
  }
};
```

6.13. Hungarian.

```
/*
      Maximum assignment (Kuhn-Munkres)

      Description:
      - We are given a cost table of size n times m with n <= m.
      - It finds a maximum cost assignment, i.e.,
                        max sum_{ij} c(i,j) x(i,j)
            where sum_{i in [n]} x(i,j) = 1,
                        sum_{j in [n]} x(i,j) <= 1.

      Complexity: O(n^3)

      Tested: http://www.spoj.com/problems/SCITIES/
*/

template <typename T> T max_assignment(const vector<vector<T>> &a) {
 int n = a.size(), m = a[0].size();
 assert(n <= m);

 vector<int> x(n, -1), y(m, -1);
 vector<T> px(n, numeric_limits<T>::min()), py(m, 0);

 for (int u = 0; u < n; ++u)
  for (int v = 0; v < m; ++v)
   px[u] = max(px[u], a[u][v]);

 for (int u = 0, p, q; u < n;) {
  vector<int> s(n + 1, u), t(m, -1);

  for (p = q = 0; p <= q && x[u] < 0; ++p)
   for (int k = s[p], v = 0; v < m && x[u] < 0; ++v)
    if (px[k] + py[v] == a[k][v] && t[v] < 0) {
     s[++q] = y[v], t[v] = k;
     if (s[q] < 0)
      for (p = v; p >= 0; v = p)
       y[v] = k = t[v], p = x[k], x[k] = v;
    }

  if (x[u] < 0) {
   T delta = numeric_limits<T>::max();

   for (int i = 0; i <= q; ++i)
    for (int v = 0; v < m; ++v)
     if (t[v] < 0)
      delta = min(delta, px[s[i]] + py[v] - a[s[i]][v]);

   for (int i = 0; i <= q; ++i)
    px[s[i]] -= delta;

   for (int v = 0; v < m; ++v)
    py[v] += (t[v] < 0 ? 0 : delta);
  } else
   ++u;
 }

 T cost = 0;

 for (int u = 0; u < n; ++u)
  cost += a[u][x[u]];

 return cost;
}
```

6.14. Kruskal.

```
struct Edge {
 int src, dst, weight;
 Edge(int a, int b, int c) : src(a), dst(b), weight(c) {}
};

const int MaxN = 10000;

vector<Edge> mst;
vector<Edge> edge;

bool cmp(Edge x, Edge y) { return x.weight < y.weight; }

int cost = 0;
void Kruskal() {
 mst.clear();
 initDisjointSet();

 sort(ALL(edge), cmp);
```

```
  for (int i = 0; i < (int)edge.size(); ++i) {
    int u = edge[i].src;
    int v = edge[i].dst;
    if (SetOf(u) != SetOf(v)) {
      cost += edge[i].weight;
      Merge(u, v);
    }
  }
}
```

6.15. Max Flow Dinic.

```
/*
    Maximum Flow (Dinitz)

    Complexity: O(n^2 m) but very fast in practice

    Tested: http://www.spoj.com/problems/FASTFLOW/
*/

template <typename flow_type> struct dinic {
  struct edge {
    size_t src, dst, rev;
    flow_type flow, cap;
  };

  int n;
  vector<vector<edge>> adj;

  dinic(int n) : n(n), adj(n), level(n), q(n), it(n) {}

  void add_edge(size_t src, size_t dst, flow_type cap, flow_type rcap = 0) {
    adj[src].push_back({src, dst, adj[dst].size(), 0, cap});
    if (src == dst)
      adj[src].back().rev++;
    adj[dst].push_back({dst, src, adj[src].size() - 1, 0, rcap});
  }

  vector<int> level, q, it;

  bool bfs(int source, int sink) {
    fill(level.begin(), level.end(), -1);
    for (int qf = level[q[0] = sink] = 0, qb = 1; qf < qb; ++qf) {
      sink = q[qf];
      for (edge &e : adj[sink]) {
        edge &r = adj[e.dst][e.rev];
        if (r.flow < r.cap && level[e.dst] == -1)
          level[q[qb++] = e.dst] = 1 + level[sink];
      }
    }
    return level[source] != -1;
  }

  flow_type augment(int source, int sink, flow_type flow) {
    if (source == sink)
      return flow;
    for (; it[source] != adj[source].size(); ++it[source]) {
      edge &e = adj[source][it[source]];
      if (e.flow < e.cap && level[e.dst] + 1 == level[source]) {
        flow_type delta = augment(e.dst, sink, min(flow, e.cap - e.flow));
        if (delta > 0) {
          e.flow += delta;
          adj[e.dst][e.rev].flow -= delta;
          return delta;
        }
      }
    }
    return 0;
  }

  flow_type max_flow(int source, int sink) {
    for (int u = 0; u < n; ++u)
      for (edge &e : adj[u])
        e.flow = 0;
    flow_type flow = 0;
    flow_type oo = numeric_limits<flow_type>::max();

    while (bfs(source, sink)) {
      fill(it.begin(), it.end(), 0);
      for (flow_type f; (f = augment(source, sink, oo)) > 0;)
        flow += f;

    } // level[u] = -1 => source side of min cut
    return flow;
  }
};
```

6.16. Max Flow push relabel.

```
/*
	Maximum Flow (Goldberg-Tarjan)

	Complexity: O(n^3) faster than Dinic in most cases

	Tested: http://www.spoj.com/problems/FASTFLOW/
*/

template<typename flow_type>
struct goldberg_tarjan
{
	struct edge
	{
		size_t src, dst, rev;
		flow_type flow, cap;
	};

	int n;
	vector<vector<edge>> adj;

	goldberg_tarjan(int n) : n(n), adj(n) {}

	void add_edge(size_t src, size_t dst, flow_type cap, flow_type rcap = 0)
	{
		adj[src].push_back({ src, dst, adj[dst].size(), 0, cap });
		if (src == dst) adj[src].back().rev++;
		adj[dst].push_back({ dst, src, adj[src].size() - 1, 0, rcap });
	}

	flow_type max_flow(int source, int sink)
	{
		vector<flow_type> excess(n);
		vector<int> dist(n), active(n), count(2 * n);
		queue<int> q;
		auto enqueue = [&](int v)
		{
			if (!active[v] && excess[v] > 0)
			{
				active[v] = true;
				q.push(v);
			}
		};
		auto push = [&](edge &e)
		{
			flow_type f = min(excess[e.src], e.cap - e.flow);
			if (dist[e.src] <= dist[e.dst] || f == 0) return;
			e.flow += f;
			adj[e.dst][e.rev].flow -= f;
			excess[e.dst] += f;
			excess[e.src] -= f;
			enqueue(e.dst);
		};
		dist[source] = n;
		active[source] = active[sink] = true;
		count[0] = n - 1;
		count[n] = 1;
		for (int u = 0; u < n; ++u)
			for (edge &e : adj[u]) e.flow = 0;
		for (edge &e : adj[source])
		{
			excess[source] += e.cap;
			push(e);
		}
		for (int u; !q.empty(); q.pop())
		{
			active[u = q.front()] = false;
			for (auto &e : adj[u]) push(e);
			if (excess[u] > 0)
			{
				if (count[dist[u]] == 1)
				{
					int k = dist[u]; // Gap Heuristics
					for (int v = 0; v < n; v++)
					{
						if (dist[v] < k)
							continue;
						count[dist[v]]--;
						dist[v] = max(dist[v], n + 1);
						count[dist[v]]++;
						enqueue(v);
					}
				}
				else
				{
					count[dist[u]]--; // Relabel
					dist[u] = 2 * n;
					for (edge &e : adj[u])
						if (e.cap > e.flow)
							dist[u] = min(dist[u],
								dist[e.dst] + 1);
```

```
                        count[dist[u]]++;
                        enqueue(u);
                    }
                }
            }
            flow_type flow = 0;
            for (edge e : adj[source])
                flow += e.flow;
            return flow;
        }
};
```

6.17. Min Cost Max Flow.

```
/*
    Minimum Cost Flow (Tomizawa, Edmonds-Karp)

    Complexity: O(F m log n), where F is the amount of maximum flow

    Tested: Codeforces [http://codeforces.com/problemset/problem/717/G]
*/

template <typename flow_type, typename cost_type> struct min_cost_max_flow {
 struct edge {
  size_t src, dst, rev;
  flow_type flow, cap;
  cost_type cost;
 };

 int n;
 vector<vector<edge>> adj;

 min_cost_max_flow(int n) : n(n), adj(n), potential(n), dist(n), back(n) {}

 void add_edge(size_t src, size_t dst, flow_type cap, cost_type cost) {
  adj[src].push_back({src, dst, adj[dst].size(), 0, cap, cost});
  if (src == dst)
   adj[src].back().rev++;
  adj[dst].push_back({dst, src, adj[src].size() - 1, 0, 0, -cost});
 }

 vector<cost_type> potential;

 inline cost_type rcost(const edge &e) {
  return e.cost + potential[e.src] - potential[e.dst];
 }

 void bellman_ford(int source) {
  for (int k = 0; k < n; ++k)
   for (int u = 0; u < n; ++u)
    for (edge &e : adj[u])
       if (e.cap > 0 && rcost(e) < 0)
        potential[e.dst] += rcost(e);
 }

 const cost_type oo = numeric_limits<cost_type>::max();

 vector<cost_type> dist;
 vector<edge *> back;

 cost_type dijkstra(int source, int sink) {
  fill(dist.begin(), dist.end(), oo);

  typedef pair<cost_type, int> node;
  priority_queue<node, vector<node>, greater<node>> pq;

  for (pq.push({dist[source] = 0, source}); !pq.empty();) {
   node p = pq.top();
   pq.pop();

   if (dist[p.second] < p.first)
    continue;
   if (p.second == sink)
    break;

   for (edge &e : adj[p.second])
    if (e.flow < e.cap && dist[e.dst] > dist[e.src] + rcost(e)) {
     back[e.dst] = &e;
     pq.push({dist[e.dst] = dist[e.src] + rcost(e), e.dst});
    }
  }

  return dist[sink];
 }

 pair<flow_type, cost_type> max_flow(int source, int sink) {
  flow_type flow = 0;
  cost_type cost = 0;
```

```
    for (int u = 0; u < n; ++u)
      for (edge &e : adj[u])
        e.flow = 0;

    potential.assign(n, 0);
    dist.assign(n, 0);
    back.assign(n, nullptr);

    bellman_ford(source); // remove negative costs

    while (dijkstra(source, sink) < oo) {
      for (int u = 0; u < n; ++u)
        if (dist[u] < dist[sink])
          potential[u] += dist[u] - dist[sink];

      flow_type f = numeric_limits<flow_type>::max();

      for (edge *e = back[sink]; e; e = back[e->src])
        f = min(f, e->cap - e->flow);
      for (edge *e = back[sink]; e; e = back[e->src])
        e->flow += f, adj[e->dst][e->rev].flow -= f;

      flow += f;
      cost += f * (potential[sink] - potential[source]);
    }
    return {flow, cost};
  }
};
```

6.18. Prim.

```
const int MaxN = 10000;

int n, m;
typedef pair<int, pii> par;
priority_queue<par, vector<par>, greater<par>> pq;
vi taken;
vector<pii> g[MaxN];
int mstCost;
vector<pii> mstEdge;

void process(int u) {
  taken[u] = 1;
  for (int i = 0; i < (int)g[u].size(); ++i) {
    pii v = g[u][i];
    if (!taken[v.S])
      pq.push(par(v.F, pii(u, v.S)));
  }
}

void Prim(int s) {
  taken.assign(n, 0);
  pq = priority_queue<par, vector<par>, greater<par>>();

  process(s);
  mstCost = 0;

  while (!pq.empty()) {
    par top = pq.top();
    pq.pop();
    pii node = top.S;

    int w = top.F;
    int u = node.F;
    int v = node.S;

    if (!taken[v]) {
      mstCost += w;
      mstEdge.pb(pii(u, v));
      process(v);
    }
  }
}
```

6.19. Satisfiability Two SAT.

```
/*
        Two-Sat

        Complexity: O(n)

        Tested: POI (Gates)
```

```
*/

struct satisfiability_twosat {
 int n;
 vector<vector<int>> imp;

 satisfiability_twosat(int n) : n(n), imp(2 * n) {}

 void add_edge(int u, int v) { imp[u].push_back(v); }

 int neg(int u) { return (n << 1) - u - 1; }

 void implication(int u, int v) {
  add_edge(u, v);
  add_edge(neg(v), neg(u));
 }

 vector<bool> solve() {
  int size = 2 * n;
  vector<int> S, B, I(size);

  function<void(int)> dfs = [&](int u) {
   B.push_back(I[u] = S.size());
   S.push_back(u);

   for (int v : imp[u])
     if (!I[v])
      dfs(v);
     else
      while (I[v] < B.back())
       B.pop_back();

    if (I[u] == B.back())
     for (B.pop_back(), ++size; I[u] < S.size(); S.pop_back())
      I[S.back()] = size;
   };

   for (int u = 0; u < 2 * n; ++u)
    if (!I[u])
     dfs(u);

   vector<bool> values(n);

   for (int u = 0; u < n; ++u)
    if (I[u] == I[neg(u)])
     return {};
    else
     values[u] = I[u] < I[neg(u)];

   return values;
  }
 };
```

6.20. Strongly Connected Components.

```
const int MaxN = 10000;

struct edge {

 int src, dst, w;
 edge(int a, int b, int c) : src(a), dst(b), w(c) {}
};

typedef vector<edge> Graph;
int n, m;
Graph g[MaxN];
Graph gt[MaxN];
int order[MaxN], mk[MaxN];
int scc[MaxN];
int vcount[MaxN];
int cur;
int cur_scc;


void dfs(int u) {
 mk[u] = true;
 for (int i = 0; i < (int)g[u].size(); ++i) {
  int v = g[u][i].dst;
  if (!mk[v])
   dfs(v);
 }
 order[n - 1 - cur++] = u;
}

void dfs_rev(int u) {
 scc[u] = cur_scc;
 ++vcount[cur_scc];
 mk[u] = true;

 for (int i = 0; i < (int)gt[u].size(); ++i) {
```

```
    int v = gt[u][i].dst;
    if (!mk[v])
      dfs_rev(v);
  }
}

void make_scc() {
  cur = 0;
  memset(mk, 0, sizeof(mk));
  for (int i = 0; i < n; ++i)
    if (!mk[i])
      dfs(i);

  cur_scc = 0;
  memset(mk, 0, sizeof(mk));
  for (int i = 0; i < n; ++i) {
    int v = order[i];
    if (!mk[v]) {
      dfs_rev(v);
      ++cur_scc;
    }
  }
}

void init() {
  for (int i = 0; i < n; ++i) {
    g[i].clear();
    gt[i].clear();
    vcount[i] = 0;
  }
}
```

6.21. SCC Gabow.

```
/*
      Gabow's strongly connected component

      Complexity: O(n + m)

      Tested: http://www.spoj.com/problems/CAPCITY/
*/

struct graph {
  int n;
  vector<vector<int>> adj;

  graph(int n) : n(n), adj(n) {}

  void add_edge(int u, int v) { adj[u].push_back(v); }

  vector<int> &operator[](int u) { return adj[u]; }
};

vector<vector<int>> scc_gabow(graph &adj) {
  int n = adj.n;

  vector<vector<int>> scc;
  vector<int> S, B, I(n);

  function<void(int)> dfs = [&](int u) {
    B.push_back(I[u] = S.size());
    S.push_back(u);

    for (int v : adj[u])
      if (!I[v])
        dfs(v);
      else
        while (I[v] < B.back())
          B.pop_back();

    if (I[u] == B.back()) {
      scc.push_back({});
      for (B.pop_back(); I[u] < S.size(); S.pop_back()) {
        scc.back().push_back(S.back());
        I[S.back()] = n + scc.size();
      }
    }
  };

  for (int u = 0; u < n; ++u)
    if (!I[u])
      dfs(u);

  return scc; // in reverse topological order
}
```

6.22. Stoer Wagner.

```
/*
      Tested: ZOJ 2753
      Complexity: O(n^3)
*/

template <typename T>
pair<T, vector<int>> stoer_wagner(vector<vector<T>> &weights) {
  int n = weights.size();
  vector<int> used(n), cut, best_cut;
  T best_weight = -1;

  for (int phase = n - 1; phase >= 0; --phase) {
    vector<T> w = weights[0];
    vector<int> added = used;
    int prev, last = 0;

    for (int i = 0; i < phase; ++i) {
      prev = last;
      last = -1;
      for (int j = 1; j < n; ++j)
        if (!added[j] && (last == -1 || w[j] > w[last]))
          last = j;
      if (i == phase - 1) {
        for (int j = 0; j < n; ++j)
          weights[prev][j] += weights[last][j];
        for (int j = 0; j < n; ++j)
          weights[j][prev] = weights[prev][j];

        used[last] = true;
        cut.push_back(last);

        if (best_weight == -1 || w[last] < best_weight) {
          best_cut = cut;
          best_weight = w[last];
        }
      } else {
        for (int j = 0; j < n; j++)
          w[j] += weights[last][j];
        added[last] = true;
      }
    }
  }

  return make_pair(best_weight, best_cut);
}
```

6.23. Tree Isomorphism.

```
/*
      Tested: SPOJ TREEISO
      Complexity: O(n log n)
*/

#define all(c) (c).begin(), (c).end()

struct tree {
  int n;
  vector<vector<int>> adj;

  tree(int n) : n(n), adj(n) {}

  void add_edge(int src, int dst) {
    adj[src].push_back(dst);
    adj[dst].push_back(src);
  }

  vector<int> centers() {
    vector<int> prev;
    int u = 0;
    for (int k = 0; k < 2; ++k) {
      queue<int> q;
      prev.assign(n, -1);
      for (q.push(prev[u] = u); !q.empty(); q.pop()) {
        u = q.front();
        for (auto v : adj[u]) {
          if (prev[v] >= 0)
            continue;
          q.push(v);
          prev[v] = u;
        }
      }
    }
```

```
    vector<int> path = {u};
    while (u != prev[u])
      path.push_back(u = prev[u]);

    int m = path.size();
    if (m % 2 == 0)
      return {path[m / 2 - 1], path[m / 2]};
    else
      return {path[m / 2]};
  }

  vector<vector<int>> layer;
  vector<int> prev;

  int levelize(int r) {
    prev.assign(n, -1);
    prev[r] = n;
    layer = {{r}};
    while (1) {
      vector<int> next;
      for (int u : layer.back())
        for (int v : adj[u]) {
          if (prev[v] >= 0)
            continue;
          prev[v] = u;
          next.push_back(v);
        }

      if (next.empty())
        break;
      layer.push_back(next);
    }
    return layer.size();
  }
};

bool isomorphic(tree S, int s, tree T, int t) {
  if (S.n != T.n)
    return false;
  if (S.levelize(s) != T.levelize(t))
    return false;

  vector<vector<int>> longcodeS(S.n + 1), longcodeT(T.n + 1);
  vector<int> codeS(S.n), codeT(T.n);
  for (int h = (int)S.layer.size() - 1; h >= 0; --h) {
    map<vector<int>, int> bucket;
    for (int u : S.layer[h]) {
      sort(all(longcodeS[u]));
      bucket[longcodeS[u]] = 0;
    }
    for (int u : T.layer[h]) {
      sort(all(longcodeT[u]));
      bucket[longcodeT[u]] = 0;
    }

    int id = 0;
    for (auto &p : bucket)
      p.second = id++;
    for (int u : S.layer[h]) {
      codeS[u] = bucket[longcodeS[u]];
      longcodeS[S.prev[u]].push_back(codeS[u]);
    }
    for (int u : T.layer[h]) {
      codeT[u] = bucket[longcodeT[u]];
      longcodeT[T.prev[u]].push_back(codeT[u]);
    }
  }

  return codeS[s] == codeT[t];
}

bool isomorphic(tree S, tree T) {
  auto x = S.centers(), y = T.centers();
  if (x.size() != y.size())
    return false;
  if (isomorphic(S, x[0], T, y[0]))
    return true;
  return x.size() > 1 && isomorphic(S, x[1], T, y[0]);
}
```

7. Matrix

7.1. Gauss.

```
/*
[TESTED COJ 2536 05/11/2014]
*/
const int MAXN = 110;
const int oo = (1 << 30);
const double EPS = 1e-6;

double a[MAXN][MAXN];
double ans[MAXN];

int n; // ecuations
int m; // variables

void init(int _n, int _m) {
 n = _n;
 m = _m;
 memset(a, 0, sizeof a);
 memset(ans, 0, sizeof ans);
}

int solve() {
 vector<int> where(m, -1);
 for (int col = 0, row = 0; col < m && row < n; ++col) {
  int sel = row;
  for (int i = row; i < n; ++i)
   if (abs(a[i][col]) > abs(a[sel][col]))
    sel = i;

  if (abs(a[sel][col]) < EPS)
   continue;
   for (int i = col; i <= m; ++i)
    swap(a[sel][i], a[row][i]);

   where[col] = row;

   for (int i = 0; i < n; ++i) {
    if (i != row) {
     double c = a[i][col] / a[row][col];
     for (int j = col; j <= m; ++j)
      a[i][j] -= a[row][j] * c;
    }
   }
   ++row;
  }

  for (int i = 0; i < m; ++i)
   if (where[i] != -1)
    ans[i] = a[where[i]][m] / a[where[i]][i];

  for (int i = 0; i < n; ++i) {
   double sum = 0;
   for (int j = 0; j < m; ++j)
    sum += ans[j] * a[i][j];
   if (abs(sum - a[i][m]) > EPS)
    return 0;
  }
  for (int i = 0; i < m; ++i)
   if (where[i] == -1)
    return oo;
  return 1;
}
```

7.2. Gauss Modulo 2.

```
/*
	[TESTED: SPOJ XMAX, LightOJ 1272,1288
	Matrix: (2) sol|x1 x2...xn
	Answer:  ans[vars-1...0]
*/
const int MAXN = 110;
const int MAXR = 70;

bitset<MAXN> row[MAXR];

int ans[MAXN];
int first[MAXR];
int vars;
int rows;
```

```cpp
void init(int _vars) {
 vars = _vars;
 rows = 0;
}

bool add(bitset<MAXN> cur) {
 for (int i = 0; i < rows; i++) {
  if (cur[first[i]] != 0) {
   cur ^= row[i];
  }
 }
 first[rows] = 0;
 while (first[rows] < vars && !cur[first[rows]])
  first[rows]++;

 /*remove if want to add always the equation*/
 if (first[rows] == vars && cur[vars])
  return false;
 row[rows++] = cur;
 return true;
}

void solve() {
 memset(ans, 0, sizeof ans);
 for (int i = rows - 1; i >= 0; i--) {
  int aux = row[i][vars];
  for (int j = first[i]; j < vars; j++)
   aux ^= (ans[j] * row[i][j]);
  ans[first[i]] = aux;
 }
}

int main() {

 init(3);

 bitset<MAXN> eq1(14), eq2(3), eq3(4);
 /*
      1|1 1 0
      0|0 1 1
      0|1 0 0
   ---------
   Ans:0 1 1
  */
 cout << add(eq1);
 cout << add(eq2);
 cout << add(eq3) << endl;

 solve();

 for (int i = vars - 1; i >= 0; --i)
  cout << ans[i] << " ";

 return 0;
}
```

7.3. Matrix Template.

```cpp
#define maxn 500

template <class T> struct Matrix {

 vector<vector<T>> data;
 int m, n;

 Matrix(int m, int n) {
  this->m = m;
  this->n = n;
  data = vector<vector<T>>(m);
  for (int i = 0; i < m; ++i)
   data[i] = vector<T>(n, 0);
 }

 void ident() {
  for (int i = 0; i < m; ++i)
   data[i][i] = 1;
 }

 Matrix<T> operator*(Matrix<T> &mtx) {
  Matrix<T> ans(m, mtx.n);
  for (int i = 0; i < ans.m; ++i)
   for (int j = 0; j < ans.n; ++j)
    for (int k = 0; k < n; ++k)
     ans.data[i][j] += data[i][k] * mtx.data[k][j];
  return ans;
 }

 Matrix<T> operator^(int exp) {
```

```
    Matrix<T> ret(m, n);
    Matrix<T> a = *this;

    ret.ident();

    if (exp == 0)
      return ret;
    if (exp == 1)
      return a;

    while (exp) {
      if (exp & 1)
        ret = ret * a;
      a = (a * a);
      exp >>= 1;
    }
    return ret;
  }
};

template <class T> istream &operator>>(istream &in, Matrix<T> &mtx) {
  for (int i = 0; i < mtx.m; ++i)
    for (int j = 0; j < mtx.n; ++j)
      in >> mtx.data[i][j];
  return in;
}

template <class T> ostream &operator<<(ostream &out, Matrix<T> &mtx) {
  for (int i = 0; i < mtx.m; ++i) {
    for (int j = 0; j < mtx.n; j++) {
      if (j)
        out << " ";
      out << mtx.data[i][j];
    }
    out << endl;
  }
  return out;
}
```

```
const double eps = 1e-7;

// Determinante
template <class T> double det(Matrix<T> M0) {
  double ans = 1;
  int size = M0.m;

  for (int i = 0, r = 0; i < size; ++i) {
    bool found = false;

    for (int j = r; j < size; ++j)
      if (fabs(M0.data[j][i]) > eps) {
        found = true;

        if (j > r)
          ans = -ans;
        else
          break;

        for (int k = 0; k < size; ++k)
          swap(M0.data[r][k], M0.data[j][k]);
        break;
      }
    if (found) {
      for (int j = r + 1; j < size; ++j) {
        double aux = M0.data[j][i] / M0.data[r][i];
        for (int k = i; k < size; ++k)
          M0.data[j][k] -= aux * M0.data[r][k];
      }
      r++;
    } else
      return 0;
  }

  for (int i = 0; i < size; ++i)
    ans *= M0.data[i][i];
  return ans;
}
```

8. Number Theory

8.1. Binomial Coefficient.

```
/*
   CALCULA COMBINATORIA DE n en k
   USANDO EL TRIANGULO DE PASCAL
*/
#include <cstdio>
#include <iostream>

#define MAX 10000

using namespace std;

int C[MAX][MAX];

void Pascal(int level) {
 for (int n = 0; n <= level; ++n) {
   C[n][0] = C[n][n] = 1;
   for (int k = 1; k < n; ++k)
    C[n][k] = C[n - 1][k] + C[n - 1][k - 1];
 }
}

int main() {
 int n, k;
 cin >> n >> k;
 Pascal(n);
 cout << C[n][k];

 return 0;
}
```

8.2. Divisibility.

```
pair<vector<int>, int> rmatrix(int base, int div) {
 vector<int> vis(div, -1);
 vector<int> res;
 res.push_back(1);
 vis[1] = 0;

 while (vis[(res[res.size() - 1] * base) % div] == -1) {
  vis[(res[res.size() - 1] * base) % div] = res.size();
  res.push_back((res[res.size() - 1] * base) % div);
 }
 return make_pair(res, vis[(res[res.size() - 1] * base) % div]);
}

bool div(int base, int div, vector<int> &num) // reverse num
{
 pair<vector<int>, int> r = rmatrix(base, div);
 int pp = 0, b = r.second;
 vector<int> a = r.first;
 for (int i = 0; i < num.size(); ++i) {
  int kk = num[i];
  if (i < b)
   pp += ((kk * a[i]) % div);
  else
   pp += ((kk * a[b + ((i - b) % (a.size() - b))]) % div);
 }
 return pp % div == 0;
}
```

8.3. ALL Number Theory.

```
/*
      Binary Multiplication
      [Tested Timus 1141,1204]**
*/
Int mod_mult(Int a, Int b, Int mod) {
 Int x = 0;
 while (b) {
  if (b & 1)
   x = (x + a) % mod;
  a = (a << 1) % mod;
```

```
    b >>= 1;
  }
  return x;
}

/*
        Binary Exponentiation
        [Tested Timus 1141,1204]**
*/
Int mod_pow(Int a, Int n, Int mod) {
  Int x = 1;
  while (n) {
    if (n & 1)
      x = mod_mult(x, a, mod);
    a = mod_mult(a, a, mod);
    n >>= 1;
  }
  return x;
}

/*
        Extended Euclidean algorithm
        Solve ax+by = (a,b)
        Works well even for negative numbers
        [Tested Timus 1141,1204]**
*/
int gcd(int a, int b, int &x, int &y) {
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  int r = gcd(b, a % b, y, x);
  y -= a / b * x;
  return r;
}

/*
Euler's function
phi(p^a) = p^a - p^(a-1)
(a,b) = 1 => phi(a*b) = phi(a)*phi(b)
[Tested Timus 1141]*
*/
int phi(int a) {
  int b = a;
  for (int i = 2; i * i <= a; ++i)
    if (a % i == 0) {
      b = b / i * (i - 1);
      do
        a /= i;
      while (a % i == 0);
    }
  if (a > 1)
    b = b / a * (a - 1);
  return b;
}

/*
Modular Inverse
        (a,m) = 1
        Solves a*x = 1 (m)
        [Tested Timus 1141, 1204]**
*/
int inverse(int a, int m) {
  int x, y;
  if (gcd(a, m, x, y) != 1)
    return 0;
  return (x % m + m) % m;
}

/*
        Baby-Step-Giant-Step Algorithm
        O(sqrt(m)log(m))
        Solve a^x = b(mod m)
        [TESTED LightOJ 1325 05/11/2014]
*/
Int discrete_log(Int a, Int b, Int m) {
  map<Int, Int> hash;
  Int n = phi(m), k = sqrt(n);

  for (Int i = 0, t = 1; i < k; i++) {
    hash[t] = i;
    t = (t * a) % m;
  }
  Int c = mod_pow(a, n - k, m);
  for (Int i = 0; i * k < n; i++) {
    if (hash.find(b) != hash.end())
      return (i * k + hash[b]) % n;

    b = (b * c) % m;
  }
  return -1;
}
```

```
/*
      Solves a*x = b (mod p)
      [Tested CodeChef Quadratic Equations]
*/
long solve_linear(long a, long b, int p) { return (b * inverse(a, p)) % p; }

/*
      Solve x=ai(mod mi)
      For any i and j, (mi,mj)|ai-aj.
      Return x0 in [0,[M]).
      M = m1m2..mn
      All solutions are x=x0+t[M].
*/
int linear_con(int a[], int m[], int n) {
 int u = a[0], v = m[0], p, q, r, t;
 for (int i = 1; i < n; i++) {
  r = gcd(v, m[i], p, q);
  t = v;
  v = v / r * m[i];
  u = ((a[i] - u) / r * p * t + u) % v;
 }
 if (u < 0)
  u += v;
 return u;
}

/*
      Solve x = ai(mod mi)
      For any i and j, (mi,mj)==1.
      Returns x0 in [0,M).
      M = m1m2..mn
      All solutions are x=x0 + tM.
*/
int chinese(int a[], int m[], int n) {
 int s = 1, t, ans = 0, p, q;
 for (int i = 0; i < n; i++)
  s *= m[i];
 for (int i = 0; i < n; i++) {
  t = s / m[i];
  gcd(t, m[i], p, q);
  ans = (ans + t * p * a[i]) % s;
 }
 if (ans < 0)
  ans += s;
 return ans;
}
```

```
/*
Kth discrete roots of a (mod n)
x^k = a (n)
When (k, phi(n)) = 1
[Tested Timus 1141]**
*/
int discrete_root(int k, int a, int n) {
 int _phi = phi(n);
 int s = (int)inverse(k, _phi);
 return (int)mod_pow(a, s, n);
}

/*
Tonelli Shank's algorithm
Solves x^2=a (mod p)
[Tested CodeChef Quadratic Equations, Timus 1132]
Warning: Precompute primes to avoid TLE
*/
int solve_quadratic(int a, int p) {
 if (a == 0)
  return 0;
 if (p == 2)
  return a;
 if (mod_pow(a, (p - 1) / 2, p) != 1)
  return -1;

 int phi = p - 1;
 int n = 0, k = 0;

 while (phi % 2 == 0) {
  phi /= 2;
  n++;
 }

 k = phi;
 int q = 0;

 for (int j = 2; j < p; j++)
  if (mod_pow(j, (p - 1) / 2, p) == p - 1) {
   q = j;
   break;
  }

 int t = mod_pow(a, (k + 1) / 2, p);
 int r = mod_pow(a, k, p);

 while (r != 1) {
```

```
    int i = 0, v = 1;
    while (mod_pow(r, v, p) != 1) {
      v *= 2;
      i++;
    }

    int e = mod_pow(2, n - i - 1, p);
    int u = mod_pow(q, k * e, p);

    t = (t * u) % p;
    r = (r * u * u) % p;
  }

  return t;
}

/*
Solves a*x^2 + b*x + c = 0 (mod p)
[Tested CodeChef Quadratic Equations]
*/
set<Int> solve_quadratic(Int a, Int b, Int c, int p) {
  set<Int> ans;
  if (c == 0)
    ans.insert(0L);
  if (a == 0)
    ans.insert(solve_linear((p - b) % p, c, p));
  else if (p == 2 && (a + b + c) % 2 == 0)
    ans.insert(1L);
  else {
    Int r = ((b * b) % p - (4 * a * c) % p + p) % p;
    Int x = solve_quadratic(r, p);
    if (x == -1)
      return ans;
    Int w = solve_linear((2 * a) % p, (x - b + p) % p, p);
    ans.insert(w);
    w = solve_linear((2 * a) % p, (p - x - b + p) % p, p);
    ans.insert(w);
  }
  return ans;
}

/*
Primitive roots
[Tested Timus 1268]
Warning: Precompute primes to avoid TLE
Only: m = 1, p^k , n = 2p^k (p prime > 2),
          m = 2, m = 4
*/
int primitive_root(int m, int p[]) {
  if (m == 1)
    return 0;
  if (m == 2)
    return 1;
  if (m == 4)
    return 3;

  int t = m;
  if ((t & 1) == 0)
    t >>= 1;

  for (int i = 0; p[i] * p[i] <= t; ++i) {
    if (t % p[i])
      continue;
    do
      t /= p[i];
    while (t % p[i] == 0);
    if (t > 1 || p[i] == 2)
      return 0;
  }

  int f[100];
  int x = phi(m), y = x, n = 0;

  for (int i = 0; p[i] * p[i] <= y; ++i) {
    if (y % p[i])
      continue;
    do
      y /= p[i];
    while (y % p[i] == 0);
    f[n++] = p[i];
  }

  if (y > 1)
    f[n++] = y;

  for (int i = 1; i < m; ++i) {
    if (__gcd(i, m) > 1)
      continue;
    bool flag = true;

    for (int j = 0; j < n; ++j)
      if (mod_pow(i, x / f[j], m) == 1) {
        flag = false;
        break;
```

```
    }

   if (flag)
    return i;
  }
 return 0;
}

typedef long long ll;

ll divisor_sigma(ll n) {
 ll sigma = 0, d = 1;
 for (; d * d < n; ++d)
  if (n % d == 0)
   sigma += d + n / d;
 if (d * d == n)
  sigma += d;
 return sigma;
}

// sigma(n) for all n in [lo, hi)
vector<ll> divisor_sigma(ll lo, ll hi) {
 vector<ll> ps = primes(sqrt(hi) + 1);
 vector<ll> res(hi - lo), sigma(hi - lo, 1);
 iota(res.begin(), res.end(), lo);
 for (ll p : ps)
  for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p) {
   ll b = 1;
   while (res[k - lo] > 1 && res[k - lo] % p == 0) {
    res[k - lo] /= p;
    b = 1 + b * p;
   }
   sigma[k - lo] *= b;
  }
 for (ll k = lo; k < hi; ++k)
  if (res[k - lo] > 1)
   sigma[k - lo] *= (1 + res[k - lo]);
 return sigma; // sigma[k-lo] = sigma(k)
}
```

```
typedef long long ll;

ll mobius_mu(ll n) {
 if (n == 0)
  return 0;
 ll mu = 1;
 for (ll x = 2; x * x <= n; ++x)
  if (n % x == 0) {
   mu = -mu;
   n /= x;
   if (n % x == 0)
    return 0;
  }
 return n > 1 ? -mu : mu;
}

// phi(n) for all n in [lo, hi)
vector<ll> mobius_mu(ll lo, ll hi) {
 vector<ll> ps = primes(sqrt(hi) + 1);
 vector<ll> res(hi - lo), mu(hi - lo, 1);
 iota(res.begin(), res.end(), lo);
 for (ll p : ps)
  for (ll k = ((lo + (p - 1)) / p) * p; k < hi; k += p) {
   mu[k - lo] = -mu[k - lo];
   if (res[k - lo] % p == 0) {
    res[k - lo] /= p;
    if (res[k - lo] % p == 0) {
     mu[k - lo] = 0;
     res[k - lo] = 1;
    }
   }
  }
 for (ll k = lo; k < hi; ++k)
  if (res[k - lo] > 1)
   mu[k - lo] = -mu[k - lo];
 return mu; // mu[k-lo] = mu(k)
}
```

8.4. Prime.

```
const int N = 16000000;
const int sqrtN = sqrt(N);
bool isP[N];
```

```
O(N log log N)
void sieve() {
 fill(isP, isP + N, true);
 isP[0] = isP[1] = false;
```

```
  for (int i = 4; i < N; i += 2)
    isP[i] = false;

  for (Int i = 3; i < sqrtN; i += 2)
    if (isP[i])
      for (Int j = i * i; j < N; j += 2ll * i)
        isP[j] = false;
}

/*
      Binary Multiplication
      [Tested Timus 1141,1204]**
*/
Int mod_mult(Int a, Int b, Int mod) {
  Int x = 0;
  while (b) {
    if (b & 1)
      x = (x + a) % mod;
    a = (a << 1) % mod;
    b >>= 1;
  }
  return x;
}

/*
      Binary Exponentiation
      [Tested Timus 1141,1204]**
*/
Int mod_pow(Int a, Int n, Int mod) {
  Int x = 1;
  while (n) {
    if (n & 1)
      x = mod_mult(x, a, mod);
    a = mod_mult(a, a, mod);
    n >>= 1;
  }
  return x;
}

/*
Miller Rabin
[Tested SPOJ PON]
*/
bool witness(Int a, Int s, Int d, Int n) {
  Int x = mod_pow(a, d, n);
  if (x == 1 || x == n - 1)
    return false;
  for (int i = 0; i < s - 1; i++) {
    x = mod_mult(x, x, n);
    if (x == 1)
      return true;
    if (x == n - 1)
      return false;
  }
  return true;
}

bool isPrime(Int n) {
  if (n < 2)
    return false;
  if (n == 2)
    return true;
  if (n % 2 == 0)
    return false;
  Int d = n - 1, s = 0;
  while (d % 2 == 0)
    ++s, d /= 2;
  Int test[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 0};
  for (int i = 0; test[i] && test[i] < n; ++i)
    if (witness(test[i], s, d, n))
      return false; // composite
  return true; // probably prime
}

/*
Integer Factorization Pollard's Rho
*/
uint64 pollar_rho(uint64 n) // n shouldn't be prime
{
  if (!(n & 1))
    return 2;

  while (true) {
    uint64 x = (uint64)rand() % n, y = x, c = rand() % n;

    if (c == 0 || c == 2)
      c = 1;

    for (int i = 1, k = 2;; i++) {
      x = mod_mult(x, x, n);
      if (x >= c)
        x -= c;
      else
```

```
        x += n - c;
      if (x == n)
        x = 0;
      if (x == 0)
        x = n - 1;
      else
        x--;

      uint64 d = __gcd(x > y ? x - y : y - x, n);

      if (d == n)
        break;
      if (d != 1)
        return d;
      if (i == k) {
        y = x;
        k <<= 1;
      }
    }
  }
}

// fact primos de n
vector<pair<Int, Int>> fact(Int n) {
  vector<pair<Int, Int>> fp;
  for (int i = 2; i <= n; ++i) {
    pair<Int, Int> pp = make_pair(i, 0);
    while (!(n % i)) {
      n /= i;
      pp.second++;
    }
    if (pp.second)
      fp.push_back(pp);
  }

  if (n > 1)
    fp.push_back(make_pair(n, 1));

  return fp;
}

vector<Int> primes;

// fact primos de n!
vector<pair<Int, Int>> factF(Int n) {
  vector<pair<Int, Int>> fp;
  Int p;
  for (int i = 0; i < (int)primes.size(); ++i) {
    p = primes[i];
    if (p > n)
      break;

    Int k = n;
    pair<Int, Int> pp = make_pair(p, 0);
    while (k) {
      pp.second += k / p;
      k /= p;
    }
    fp.push_back(pp);
  }

  return fp;
}

/*
    Tested: SPOJ PRIME1, ETFS
    Complexity: O(n log log n)
*/

typedef long long ll;

// primes in [lo, hi)
vector<ll> primes(ll lo, ll hi) {
  const ll M = 1 << 14, SQR = 1 << 16;
  vector<bool> composite(M), small_composite(SQR);
  vector<pair<ll, ll>> sieve;
  for (ll i = 3; i < SQR; i += 2)
    if (!small_composite[i]) {
      ll k = i * i + 2 * i * max(0.0, ceil((lo - i * i) / (2.0 * i)));
      sieve.push_back({2 * i, k});
      for (ll j = i * i; j < SQR; j += 2 * i)
        small_composite[j] = 1;
    }
  vector<ll> ps;
  if (lo <= 2) {
    ps.push_back(2);
    lo = 3;
  }
  for (ll k = lo | 1, low = lo; low < hi; low += M) {
    ll high = min(low + M, hi);
    fill(composite.begin(), composite.end(), 0);
    for (auto &z : sieve)
      for (; z.second < high; z.second += z.first)
        composite[z.second - low] = 1;
```

```
  for (; k < high; k += 2)
    if (!composite[k - low])
      ps.push_back(k);
 }
  return ps;
}

vector<ll> primes(ll hi) { return primes(0, hi); }
```

8.5. Tree Stern-Brocot.

```
/*
Stern-Brocot Tree for enumerating rationals
Enumerating all irreducible rationals ascending order,
Whose sum of N and D is atmost B
*/
void sternBrocot(Int B, Int pl = 0, Int ql = 1, Int pr = 1, Int qr = 0) {
  Int pm = pl + pr, qm = ql + qr;
  if (pm + qm > B)
    return;
  sternBrocot(B, pl, ql, pm, qm); // [pl / ql, pm / qm]
  cout << pm << "/" << qm << endl;
  sternBrocot(B, pm, qm, pr, qr); // [pm / qm, pr / qr]
}
```

9. Numeric Methods

9.1. Fast Fourier Transform.

```
typedef complex<double> base;

// y[i] = A(w^(dir*i)),
// w = exp(2pi/N) is N-th complex principal root of unity,
// A(x) = a[0] + a[1] x + ... + a[n-1] x^{n-1},
// * N must be a power of 2,
long double PI = 2 * acos(0.0L);

void fft(vector<base> &a, bool invert) {
 int n = (int)a.size();

 for (int i = 1, j = 0; i < n; ++i) {
  int bit = n >> 1;
  for (; j >= bit; bit >>= 1)
   j -= bit;
  j += bit;
  if (i < j)
   swap(a[i], a[j]);
 }
 for (int len = 2; len <= n; len <<= 1) {
  double ang = 2 * PI / len * (invert ? -1 : 1);
  base wlen(cos(ang), sin(ang));

  for (int i = 0; i < n; i += len) {
   base w(1);
   for (int j = 0; j < len / 2; ++j) {
    base u = a[i + j], v = a[i + j + len / 2] * w;
    a[i + j] = u + v;
    a[i + j + len / 2] = u - v;
    w *= wlen;
   }
  }
 }
 if (invert)
  for (int i = 0; i < n; ++i)
   a[i] /= n;
}

void convolve(const vector<int> &a, const vector<int> &b, vector<int> &res) {
 vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());
 size_t n = 1;
 while (n < max(a.size(), b.size()))
  n <<= 1;
 n <<= 1;
 fa.resize(n), fb.resize(n);
 fft(fa, false), fft(fb, false);
 for (size_t i = 0; i < n; ++i)
  fa[i] *= fb[i];
 fft(fa, true);
 res.resize(n);
 for (size_t i = 0; i < n; ++i)
  res[i] = int(fa[i].real() + 0.5);
}

void print(vector<int> a) {
 cout << a.size() << endl;
 for (int i = 0; i < (int)a.size(); ++i)
  cout << a[i] << " ";
 cout << endl;
}
```

9.2. Goldsection Search.

```
/*
    Minimum of unimodal function (goldsection search)

    Tested: COJ 2890 :(
*/

template <class F> double find_min(F f, double a, double d, double eps = 1e-9) {
 const int iter = 150;
 const double r = 2 / (3 + sqrt(5.));
 double b = a + r * (d - a), c = d - r * (d - a), fb = f(b), fc = f(c);
 for (int it = 0; it < iter && d - a > eps; ++it) {
  // '<': maximum, '>': minimum
  if (fb > fc) {
   a = b;
   b = c;
   c = d - r * (d - a);
```

```
      fb = fc;
      fc = f(c);
    } else {
      d = c;
      c = b;
      b = a + r * (d - a);
      fc = fb;
      fb = f(b);
    }
  }
  return c;
}
```

9.3. Linear Recursion.

```
/*
    Linear Recurrence Solver

    Description: Consider
    x[i+n] = a[0] x[i] + a[1] x[i+1] + ... + a[n-1] x[i+n-1]
    with initial solution x[0], x[1], ..., x[n-1]
    We compute k-th term of x in O(n^2 log k) time.

    Tested: SPOJ REC
    Complexity: O(n^2 log k) time, O(n log k) space
*/

typedef long long ll;

ll linear_recurrence(vector<ll> a, vector<ll> x, ll k) {
  int n = a.size();
  vector<ll> t(2 * n + 1);
  function<vector<ll>(ll)> rec = [&](ll k) {
    vector<ll> c(n);
    if (k < n)
      c[k] = 1;
    else {
      vector<ll> b = rec(k / 2);
      fill(t.begin(), t.end(), 0);
      for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
          t[i + j + (k & 1)] += b[i] * b[j];
      for (int i = 2 * n - 1; i >= n; --i)
        for (int j = 0; j < n; ++j)
          t[i - n + j] += a[j] * t[i];
      for (int i = 0; i < n; ++i)
        c[i] = t[i];
    }
    return c;
  };
  vector<ll> c = rec(k);
  ll ans = 0;
  for (int i = 0; i < x.size(); ++i)
    ans += c[i] * x[i];
  return ans;
}
```

9.4. Romberg.

```
const double EPS = 1e-6;

// Romberg
// Assume F' = f
// input: interval [a,b] and a function f
// ouput: F(b)-F(a)

inline int cmp(double x, double y = 0) {
  return (x <= y + EPS) ? (x + EPS < y) ? -1 : 0 : 1;
}

int pow(int a, int n) {
  int x = 1;
  while (n) {
    if (n & 1)
      x *= a;
    n >>= 1;
    a *= a;
  }
  return x;
}

long double romberg(int a, int b, double (*func)(double)) {
  long double approx[2][50];
```

```
  long double *cur = approx[1], *prev = approx[0];

  prev[0] = 1 / 2.0 * (b - a) * (func(a) + func(b));

  for (int it = 1; it < 25; ++it, swap(cur, prev)) {
    if (it > 1 && cmp(prev[it - 1], prev[it - 2]) == 0)
      return prev[it - 1];

    cur[0] = 1 / 2.0 * prev[0];
    long double div = (b - a) / pow(2, it);
    for (long double sample = a + div; sample < b; sample += 2 * div)
      cur[0] += div * func(a + sample);

    for (int j = 1; j <= it; ++j)
      cur[j] = cur[j - 1] + 1 / (pow(4, it) - 1) * (cur[j - 1] + prev[j - 1]);
  }

  return prev[24];
}
```

9.5. Roots Newton.

```
template <class F, class G> double find_root(F f, G df, double x) {
  for (int iter = 0; iter < 100; ++iter) {
    double fx = f(x), dfx = df(x);
    x -= fx / dfx;
    if (fabs(fx) < 1e-12)
      break;
  }
  return x;
}
```

9.6. Simplex.

```
/*
    Parametric Self-Dual Simplex method

    Description:
    - Solve a canonical LP:
        min. c x
      s.t. A x <= b
        x >= 0

    Complexity: O(n+m) iterations on average

    Tested: http://codeforces.com/contest/375/problem/E
*/

const double eps = 1e-9, oo = numeric_limits<double>::infinity();

typedef vector<double> vec;
typedef vector<vec> mat;

double simplexMethodPD(mat &A, vec &b, vec &c) {
  int n = c.size(), m = b.size();
  mat T(m + 1, vec(n + m + 1));
  vector<int> base(n + m), row(m);

  for (int j = 0; j < m; ++j) {
    for (int i = 0; i < n; ++i)
      T[j][i] = A[j][i];
    T[j][n + j] = 1;
    base[row[j] = n + j] = 1;
    T[j][n + m] = b[j];
  }

  for (int i = 0; i < n; ++i)
    T[m][i] = c[i];

  while (1) {
    int p = 0, q = 0;
    for (int i = 0; i < n + m; ++i)
      if (T[m][i] <= T[m][p])
        p = i;

    for (int j = 0; j < m; ++j)
      if (T[j][n + m] <= T[q][n + m])
        q = j;

    double t = min(T[m][p], T[q][n + m]);

    if (t >= -eps) {
```

```
    vec x(n);
    for (int i = 0; i < m; ++i)
      if (row[i] < n)
        x[row[i]] = T[i][n + m];
    // x is the solution
    return -T[m][n + m]; // optimal
  }

  if (t < T[q][n + m]) {
    // tight on c -> primal update
    for (int j = 0; j < m; ++j)
      if (T[j][p] >= eps)
        if (T[j][p] * (T[q][n + m] - t) >= T[q][p] * (T[j][n + m] - t))
          q = j;

    if (T[q][p] <= eps)
      return oo; // primal infeasible
  } else {
    // tight on b -> dual update
    for (int i = 0; i < n + m + 1; ++i)
      T[q][i] = -T[q][i];

    for (int i = 0; i < n + m; ++i)
      if (T[q][i] >= eps)
        if (T[q][i] * (T[m][p] - t) >= T[q][p] * (T[m][i] - t))
          p = i;

    if (T[q][p] <= eps)
      return -oo; // dual infeasible
  }

  for (int i = 0; i < m + n + 1; ++i)
    if (i != p)
      T[q][i] /= T[q][p];

  T[q][p] = 1; // pivot(q, p)
  base[p] = 1;
  base[row[q]] = 0;
  row[q] = p;

  for (int j = 0; j < m + 1; ++j)
    if (j != q) {
      double alpha = T[j][p];
      for (int i = 0; i < n + m + 1; ++i)
        T[j][i] -= T[q][i] * alpha;
    }
  }

  return oo;
}
```

9.7. Simpson.

```
/*
    Tested: COJ
    2121 - Environment Protection
*/
// METODO DE SIMPSON 1/3 Compuesta
// a,b: intervalo de integracion
// n = 10000: numero de pasos (ya multiplicado por 2)
double Simpson(int n, double a, double b, double (*f)(double)) {
  double s = 0;
  double h = (double)(b - a) / n;
  for (int i = 0; i <= n; ++i) {
    double x = a + h * i;
    s += f(x) * ((i == 0 || i == n) ? 1 : ((i & 1) == 0) ? 2 : 4);
  }
  return s * (h / 3);
}
```

10. Parsing

10.1. Shunting Yard.

```
enum type { op, value, obracket, cbracket }; // types
struct token {
 string text;
 type ttype;
};

template <typename T> struct operation {
 int precedence;
 function<void(stack<T> &s)> operate;
};

void mul(stack<string> &s); // operator
void pluss(stack<string> &s);
void poww(stack<string> &s);

unordered_map<string, operation<string>> operations;

bool rpn(const vector<token> &tokens, queue<token> &rpn) {
 stack<token> operators;
 for (auto &token : tokens) {
  if (token.ttype == value)
   rpn.push(token);
  else if (token.ttype == op) {
   while (operators.size() > 0 && operators.top().ttype != obracket &&
        operations[token.text].precedence >
           operations[operators.top().text].precedence) {
    rpn.push(operators.top());
    operators.pop();
   }
   operators.push(token);
  } else if (token.ttype == obracket)
   operators.push(token);
  else if (token.ttype == cbracket) {
   while (operators.top().ttype != obracket) {
    rpn.push(operators.top());
    operators.pop();
    if (operators.size() == 0)
     return false;
   }
   operators.pop();
  }
 }
 while (operators.size() > 0) {
  if (operators.top().ttype == obracket)
   return false;
  rpn.push(operators.top());
  operators.pop();
 }
 return true;
}

template <typename T> T eval(queue<token> &rpn, bool &ok) {
 stack<T> result;
 while (rpn.size() > 0) {
  auto t = rpn.front();
  rpn.pop();
  if (t.ttype == value)
   result.push(t.text); // parsear t.text
  if (t.ttype == op)
   operations[t.text].operate(result);
 }
 ok = result.size() == 1;
 return result.top();
}

vector<token> lex(const string &str); // lexer

int main() {
 operations["*"] = {1, poww};
 operations["."] = {2, mul};
 operations["|"] = {3, pluss};
 string str;
 auto toks = lex(str);
 queue<token> q;
 rpn(toks, q);
 bool ok;
 auto result = eval<string>(q, ok);
 cout << result << '\n';
 return 0;
}
```

11. Sorting-Searching

11.1. Ternary Searh.

```
double TernarySearchMin(double l, double r) {
 while (r - l > EPS) {
  double m1 = (2 * l + r) / 3.0;
  double m2 = (l + 2 * r) / 3.0;

  if (f(m1) < f(m2))
   r = m2;
  else
   l = m1;
 }
 return (l + r) / 2.0;
}

double TernarySearchMax(double l, double r) {
 while (r - l > EPS) {
  double m1 = (2 * l + r) / 3.0;
  double m2 = (l + 2 * r) / 3.0;

  if (f1(m1) < f1(m2))
   l = m1;
  else
   r = m2;
 }
 return (l + r) / 2.0;
}

// Discrette
int SearchMin(vector<int> &y) {
 int l = 0, r = y.size() - 1;
 while (r - l < 3) {
  int m1 = (2 * l + r) / 3;
  int m2 = (l + 2 * r) / 3;

  if (y[m1] < y[m2])
   r = m2;
  else
   l = m1;
 }
 return min_element(y.begin() + l, y.begin() + r) - y.begin();
}
```

12. String

12.1. Aho Corasick.

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

using namespace std;

#define endl '\n'
#define DB(x) cout << #x << " = " << x << endl;

const int size = 505;
const int MAXS = size * size + 10;
const int MAXC = 26;

struct aho_corasick {
 vector<string> key;
 vector<bitset<505>> output;
 vector<int> failure;
 vector<vector<int>> gto;

 int buildMachine() {
  int states = 1;
  for (int i = 0; i < key.size(); ++i) {
   const string &word = key[i];
   int currentState = 0;

   for (int j = 0; j < word.size(); ++j) {
    int ch = word[j] - 'a';

    if (gto[currentState][ch] == -1)
     gto[currentState][ch] = states++;

    currentState = gto[currentState][ch];
   }
   output[currentState].set(i);
  }

  for (int ch = 0; ch < MAXC; ++ch)
   if (gto[0][ch] == -1)
    gto[0][ch] = 0;

  queue<int> q;
  for (int ch = 0; ch < MAXC; ++ch) {
   if (gto[0][ch] != 0) {
    failure[gto[0][ch]] = 0;
    q.push(gto[0][ch]);
   }
  }

  while (!q.empty()) {
   int state = q.front();
   q.pop();

   for (int ch = 0; ch < MAXC; ++ch) {
    if (gto[state][ch] != -1) {
     int f = failure[state];
     while (gto[f][ch] == -1)
      f = failure[f];

     f = gto[f][ch];
     failure[gto[state][ch]] = f;
     output[gto[state][ch]] |= output[f];
     q.push(gto[state][ch]);
    }
   }
  }
  return states;
 }

 aho_corasick(const vector<string> &k) : key(k) {
  failure = vector<int>(MAXS, -1);
  gto = vector<vector<int>>(MAXS, vector<int>(MAXC, -1));
  output = vector<bitset<505>>(MAXS);

  buildMachine();
 }

 int nextState(int currentState, char nextInput) {
  int state = currentState;
  int ch = nextInput - 'a';
  while (gto[state][ch] == -1)
   state = failure[state];
  return gto[state][ch];
 }

 vector<int> match(const string &text) {
  vector<int> ans(key.size());
  int currentState = 0;
```

```
    for (int i = 0; i < text.size(); ++i) {
     currentState = nextState(currentState, text[i]);

     if (output[currentState].any())
      for (int j = 0; j < key.size(); ++j)
       if (output[currentState].test(j))
        ans[j]++;
    }
    return ans;
  }
};

int main() {
  int nc;
  cin >> nc;
  for (int tc = 1; tc <= nc; ++tc) {
    int n;
    cin >> n;
    string t;
    cin >> t;
    vector<string> key(n);
    for (int i = 0; i < n; ++i)
     cin >> key[i];

    aho_corasick aho(key);

    cout << "Case␣" << tc << ":\n";
    vector<int> ans = aho.match(t);
    for (int i = 0; i < ans.size(); ++i)
     cout << ans[i] << endl;
  }

  return 0;
}
```

12.2. Knuth-Morris-Pratt.

```
// pi[1...m]
vector<int> buildFail(string p) {
  int m = p.size();
  vector<int> pi(m + 1, 0);

  int j = pi[0] = -1;

  for (int i = 1; i <= m; ++i) {
    while (j >= 0 && p[j] != p[i - 1])
      j = pi[j];
    pi[i] = ++j;
  }
  return pi;
}
// KMP Cuenta la cantidad de veces que aparece una
// sub-cadena (p) en la cadena (t)
int match(string t, string p, vector<int> &pi) {
  int n = t.size(), m = p.size();
  int count = 0;

  for (int i = 0, k = 0; i < n; ++i) {
    while (k >= 0 && p[k] != t[i])
      k = pi[k];
    if (++k >= m) {
      ++count;
      k = pi[k];
    }
  }
  return count;
}
```

12.3. Longest Common Subsequence.

```
#define MAX 100
char X[MAX], Y[MAX];
int i, j, m, n, c[MAX][MAX], b[MAX][MAX];

int LCSlength() {
  m = strlen(X);
  n = strlen(Y);

  for (i = 1; i <= m; i++)
    c[i][0] = 0;
  for (j = 0; j <= n; j++)
    c[0][j] = 0;

  for (i = 1; i <= m; i++)
```

```
  for (j = 1; j <= n; j++) {
   if (X[i - 1] == Y[j - 1]) {
    c[i][j] = c[i - 1][j - 1] + 1;
    b[i][j] = 1; /* from north west */
   } else if (c[i - 1][j] >= c[i][j - 1]) {
    c[i][j] = c[i - 1][j];
    b[i][j] = 2; /* from north */
   } else {
    c[i][j] = c[i][j - 1];
    b[i][j] = 3; /* from west */
   }
  }

 return c[m][n];
}

void printLCS(int i, int j) {
 if (i == 0 || j == 0)
  return;

 if (b[i][j] == 1) {
  printLCS(i - 1, j - 1);
  printf("%c", X[i - 1]);
 } else if (b[i][j] == 2)
  printLCS(i - 1, j);
 else
  printLCS(i, j - 1);
}

int main() {
 while (1) {
  gets(X);
  if (feof(stdin))
   break; /* press ctrl+z to terminate */
  gets(Y);
  printf("LCS␣length␣->␣%d\n", LCSlength()); /* count length */
  printLCS(m, n); /* reconstruct LCS */
  printf("\n");
 }
 return 0;
}
```

12.4. Longest Palindrome Substring.

```
// Transform S into T.
// For example, S = "abba", T = "^#a#b#b#a#$".
// ^ and $ signs are sentinels appended to each end to avoid bounds checking
string preProcess(string s) {
 int n = s.length();
 if (n == 0)
  return "^$";
 string ret = "^";
 for (int i = 0; i < n; i++)
  ret += "#" + s.substr(i, 1);

 ret += "#$";
 return ret;
}

// Time: O(n)
string longestPalindrome(string s) {
 string T = preProcess(s);
 int n = T.length();
 int *P = new int[n];
 int C = 0, R = 0;
 for (int i = 1; i < n - 1; i++) {
  int i_mirror = 2 * C - i; // equals to i' = C - (i-C)

  P[i] = (R > i) ? min(R - i, P[i_mirror]) : 0;

  // Attempt to expand palindrome centered at i
  while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
   P[i]++;

  // If palindrome centered at i expand past R,
  // adjust center based on expanded palindrome.
  if (i + P[i] > R) {
   C = i;
   R = i + P[i];
  }
 }

 // Find the maximum element in P.
 int maxLen = 0;
 int centerIndex = 0;
 for (int i = 1; i < n - 1; i++) {
  if (P[i] > maxLen) {
```

```
      maxLen = P[i];
      centerIndex = i;
    }
  }
  delete[] P;

  return s.substr((centerIndex - 1 - maxLen) / 2, maxLen);
}
```

12.5. Manacher.

```
#define MAX 100
int rank[MAX], LCP[MAX];

// ""[ (i-d)/2 , (i+d)/2 )"" l[i] = d
vector<int> manacher(string text) {
  int n = text.size(), i, j, k = 0;
  vector<int> rad(n << 1);

  for (i = 0, j = 0; i < (n << 1); i += k, j = max(j - k, 0)) {
    while (i - j >= 0 && i + j + 1 < (n << 1) &&
        text[(i - j) >> 1] == text[(i + j + 1) >> 1])
      ++j;
    rad[i] = j;
    for (k = 1; i - k >= 0 && rad[i] - k >= 0 && rad[i - k] != rad[i] - k; ++k)
      rad[i + k] = min(rad[i - k], rad[i] - k);
  }
  rad.insert(rad.begin(), 0);
  return rad;
}
```

12.6. Maximal Suffix.

```
/*
    Complexity: O(n)
*/

int maximal_suffix(const string &s) {
  int n = s.length(), i = 0, j = 1;

  for (int k = 0; j < n - 1; k = 0) {
    while (j + k < n - 1 && s[i + k] == s[j + k])
      ++k;

    if (s[i + k] < s[j + k]) {
      i += (k / (j - i) + 1) * (j - i);
      j = i + 1;
    } else
      j += k + 1;
  }

  return i;
}
```

12.7. Minimum Rotation.

```
/*
    Complexity: O(n)
*/

int minimum_rotation(const string &s) {
  int n = s.length(), i = 0, j = 1, k = 0;

  while (i + k < 2 * n && j + k < 2 * n) {
    char a = i + k < n ? s[i + k] : s[i + k - n];
    char b = j + k < n ? s[j + k] : s[j + k - n];

    if (a > b) {
      i += k + 1;
      k = 0;
      if (i <= j)
        i = j + 1;
    } else if (a < b) {
      j += k + 1;
```

```
    k = 0;
    if (j <= i)
      j = i + 1;
  } else
    ++k;
  }

  return min(i, j);
}
```

12.8. Palindromic Tree.

```
/*
        Palindromic Tree

        Complexity: O(n)

        Tested: ??
*/

template <size_t maxlen, size_t alpha> struct PalindromicTree {
  int go[maxlen + 2][alpha], slink[maxlen + 2], length[maxlen + 2];
  int s[maxlen], slength, size, last;

  int new_node() {
    memset(go[size], 0, sizeof go[size]);
    slink[size] = length[size] = 0;
    return size++;
  }

  PalindromicTree() { reset(); }

  void reset() {
    size = slength = 0;
    length[new_node()] = -1;
    last = new_node();
  }

  int get_link(int p) {
    for (int i = slength - 1;
        i - 1 - length[p] < 0 || s[i - 1 - length[p]] != s[i];)
      p = slink[p];
    return p;
  }

  int _extend(int c) {
    s[slength++] = c;
    int p = get_link(last), np;
    if (go[p][c])
      return go[p][c];
    length[np = new_node()] = 2 + length[p];
    go[p][c] = np;
    if (length[np] == 1)
      return slink[np] = 1, np;
    p = slink[p];
    slink[np] = go[get_link(p)][c];
    return np;
  }

  void extend(int c) { last = _extend(c); }
};
```

12.9. Substring Palindrome.

```
using System;
namespace hash {
class Program {
  static int MAXN = 100000 + 10;
  static long[] fh, bh, prime;
  static long mod = 1000000009;
  static long x = 1223;
  static string s;
  static int n;

  static void prime_power(int n) {
    prime[0] = 1;
    for (int i = 1; i <= n + 5; i++)
      prime[i] = (prime[i - 1] * x) % mod;
  }

  static void compute_hash(string s) {
    for (int i = 1, j = n; i <= n; j--, i++) {
```

```
    fh[i] = (fh[i - 1] + s[i - 1] * prime[i]) % mod;
    bh[j] = (bh[j + 1] + s[j - 1] * prime[i]) % mod;
  }
}

static bool subtring_palindrome(int l, int r) {
  ++l;
  ++r;
  long h1 = (fh[r] - fh[l - 1] + mod) % mod;
  long h2 = (bh[l] - bh[r + 1] + mod) % mod;

  if (l <= n - r + 1) {
    int pow = (n - r + 1) - l;
    h1 = (h1 * prime[pow]) % mod;
  } else {
    int pow = l - (n - r + 1);
    h2 = (h2 * prime[pow]) % mod;
  }
  return h1 == h2;
}

  static void Main(string[] args) {
    fh = new long[MAXN];
    bh = new long[MAXN];
    prime = new long[MAXN];

    string s = Console.ReadLine();
    n = s.Length;
    prime_power(s.Length);
    compute_hash(s);

    int q = int.Parse(Console.ReadLine());
    for (int i = 0; i < q; ++i) {
      int[] query = Array.ConvertAll(Console.ReadLine().Split(), int.Parse);
      Console.WriteLine("{0}",
                  subtring_palindrome(query[0], query[1]) ? "YES" : "NO");
    }
  }
}
} // namespace hash
```

12.10. Suffix Array.

```
/*
    Suffix array + lcp

    Complexity: O(n log n)

    Tested:
    - http://www.spoj.com/problems/SARRAY/
    - http://acm.timus.ru/problem.aspx?space=1&num=1393
    - http://wcipeg.com/problem/coci092p6
    - http://www.spoj.com/problems/LCS/

    Note: lcp[i] = lcp(s[sa[i-1]...], s[sa[i]...])
*/

template <typename charT> struct SuffixArray {
  int n;
  vector<int> sa, rank, lcp;

  SuffixArray(const basic_string<charT> &s)
      : n(s.length() + 1), sa(n), rank(n), lcp(n) {
    vector<int> _sa(n), bucket(n);

    iota(sa.rbegin(), sa.rend(), 0);

    sort(next(sa.begin()), sa.end(), [&](int i, int j) { return s[i] < s[j]; });

    for (int i = 1, j = 0; i < n; ++i) {
      rank[sa[i]] = rank[sa[i - 1]] + (i == 1 || s[sa[i - 1]] < s[sa[i]]);
      if (rank[sa[i]] != rank[sa[i - 1]])
        bucket[++j] = i;
    }

    for (int len = 1; len <= n; len += len) {
      for (int i = 0, j; i < n; ++i) {
        if ((j = sa[i] - len) < 0)
          j += n;
        _sa[bucket[rank[j]]++] = j;
      }

      sa[_sa[bucket[0] = 0]] = 0;

      for (int i = 1, j = 0; i < n; ++i) {
        if (rank[_sa[i]] != rank[_sa[i - 1]] ||
            rank[_sa[i] + len] != rank[_sa[i - 1] + len])
          bucket[++j] = i;

        sa[_sa[i]] = j;
```

```
    }

    copy(sa.begin(), sa.end(), rank.begin());
    sa.swap(_sa);

    if (rank[sa[n - 1]] == n - 1)
     break;
```

```
   }

   for (int i = 0, j = rank[lcp[0] = 0], k = 0; i < n - 1; ++i, ++k)
    while (k >= 0 && s[i] != s[sa[j - 1] + k])
     lcp[j] = k--, j = rank[sa[j] + 1];
  }
};
```

12.11. **Suffix Automaton.**

```
/*
      Generalized Suffix Automaton

      Complexity: O(n)

      Tested:
      - http://codeforces.com/contest/616/problem/F
      - http://codeforces.com/contest/452/problem/E
      - http://codeforces.com/contest/204/problem/E
*/


template<size_t maxlen, size_t alpha>
struct SuffixAutomaton
{
      int go[2 * maxlen][alpha], slink[2 * maxlen], length[2 * maxlen];
      int size, last;

      int new_node()
      {
            memset(go[size], 0, sizeof go[size]);
            slink[size] = length[size] = 0;
            return size++;
      }

      SuffixAutomaton() { reset(); }

      void reset()
      {
            size = last = 0;
            new_node();
            slink[0] = -1;
      }

      int _extend(int c)
      {
            int p, q, np, nq;
            if (q = go[last][c])
            {
                  if (length[q] == 1 + length[last]) return q;
                  int nq = new_node();
                  length[nq] = 1 + length[last];
                  memcpy(go[nq], go[q], sizeof go[q]);
                  slink[nq] = slink[q];
                  slink[q] = nq;
                  for (p = last; p != -1 && go[p][c] == q; p = slink[p])
                        go[p][c] = nq;
                  return nq;
            }
            np = new_node();
            length[np] = 1 + length[last];
            for (p = last; p != -1 && !go[p][c]; p = slink[p])
                  go[p][c] = np;
            if (p == -1) return slink[np] = 0, np;
            if (length[q = go[p][c]] == 1 + length[p]) return slink[np] = q, np;
            nq = new_node();
            length[nq] = 1 + length[p];
            memcpy(go[nq], go[q], sizeof go[q]);
            slink[nq] = slink[q];
            slink[q] = slink[np] = nq;
            for (; p != -1 && go[p][c] == q; p = slink[p])
                  go[p][c] = nq;
            return np;
      }

      void extend(int c) { last = _extend(c); }

      int bucket[maxlen + 1], order[2 * maxlen];

      void top_sort()
      {
            int maxl = 0;
```

```
        for (int e = 0; e < size; ++e)
            maxl = max(maxl, length[e]);
        for (int l = 0; l <= maxl; ++l)
            bucket[l] = 0;
        for (int e = 0; e < size; ++e)
            ++bucket[length[e]];
        for (int l = 1; l <= maxl; ++l)
            bucket[l] += bucket[l - 1];
        for (int e = 0; e < size; ++e)
            order[--bucket[length[e]]] = e;
    }
};
```

12.12. **Z Function.**

```
// Z[i] is the length of the longest substring
// starting from S[i] which is also a prefix of S.
vector<int> z_function(string s) {
 int n = (int)s.length();
 vector<int> z(n);

 for (int i = 1, l = 0, r = 0; i < n; ++i) {
  if (i <= r)
   z[i] = min(r - i + 1, z[i - 1]);
  while (i + z[i] < n && s[z[i]] == s[i + z[i]])
   ++z[i];
  if (i + z[i] - 1 > r)
   l = i, r = i + z[i] - 1;
 }
 return z;
}

// suff[i] = length of the longest common suffix of s and s[0..i]
vector<int> suffixes(const string &s) {
 int n = s.length();

 vector<int> suff(n, n);

 for (int i = n - 2, g = n - 1, f; i >= 0; --i) {
  if (i > g && suff[i + n - 1 - f] != i - g)
   suff[i] = min(suff[i + n - 1 - f], i - g);
  else {
   for (g = min(g, f = i); g >= 0 && s[g] == s[g + n - 1 - f]; --g)
    ;
   suff[i] = f - g;
  }
 }

 return suff;
}
```

13. Mathematical facts

13.1. **Números de Catalán.** están definidos por la recurrencia:

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

13.2. **Números de Stirling de primera clase.** son el número de permutaciones de n elementos con exactamente k ciclos disjuntos.

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

13.3. **Números de Stirling de segunda clase.** son el número de particionar un conjunto de n elementos en k subconjuntos no vacíos.

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = k\begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$$

Además:

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

13.4. **Números de Bell.** cuentan el número de formas de dividir n elementos en subconjuntos.

$$\mathcal{B}_{n+1} = \sum_{k=0}^{n}\binom{n}{k}\mathcal{B}_k$$

x	0	1	2	3	4	5	6	7	8	9	10
$\mathcal{B}_x$	1	1	2	5	15	52	203	877	4.140	21.147	115.975

13.5. **Derangement.** permutación que no deja ningún elemento en su lugar original

$$!n = (n-1)(!(n-1)+!(n-2)); !1 = 0, !2 = 1$$

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

13.6. **Números armónicos.**

$$H_n = \sum_{k=1}^{n}\frac{1}{k}$$

$$\frac{1}{2n+1} < H_n - \ln n - \gamma < \frac{1}{2n}$$

$$\gamma = 0.577215664901532860606512090082402431042159335\ldots$$

13.7. **Número de Fibonacci.** $f_0 = 0,\ f_1 = 1$:

$$f_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$$

$$f_{n+1} = f_n * 2 - f_{n-2}$$

$$f_0 + f_1 + f_2 + \cdots + f_n = f_{n+2} - 1$$

$$f_0 - f_1 + f_2 - \cdots + (-1)^n f_n = (-1)^n f_{n-1} - 1$$

$$f_1 + f_3 + f_5 + \cdots + f_{2n-1} = f_{2n}$$

$$f_0 + f_2 + f_4 + \cdots + f_{2n} = f_{2n+1} - 1$$

$$f_0^2 + f_1^2 + f_2^2 + \cdots + f_n^2 = f_n f_{n+1}$$

$$f_1f_2 + f_2f_3 + f_3f_4 + \cdots + f_{2n-1}f_n = f_{2n}^2$$

$$f_1f_2 + f_2f_3 + f_3f_4 + \cdots + f_{2n}f_{2n+1} = f_{2n+1}^2 - 1$$

$$k \geq 1 \Rightarrow f_{n+k} = f_kf_{n+1} + f_{k-1}f_n \forall n \geq 0$$

Identidad de Cassini: $f_{n+1}fn - 1 - f_n^2 = (-1)^n$

$$f_{n+1}^2 + f_n^2 = f_{2n+1}$$

$$f_{n+2}^2 - f_n^2 = f_{2n+2}$$

$$f_{n+2}^2 - f_{n+1}^2 = f_nf_{n+3}$$

$$f_{n+2}^3 - f_{n+1}^3 - f_n^3 = f_{3n+3}$$

$$mcd(f_n, f_m) = f_{mcd(n,m)}$$

$$f_{n+1} = \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-j}{j}$$

$$f_{3n} = \sum_{j=0}^{n} \binom{n}{j} 2^j f_j$$

El último dígito de cada número se repite periódicamente cada 60 números. Los dos últimos, cada 300; a partir de ahí, se repiten cada $15 * 10^{n-1}$ números.

13.8. **Sumas de combinatorios.**

$$\sum_{i=n}^{m} \binom{i}{n} = \binom{m+1}{n+1}$$

$$\sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$$

13.9. **Funciones generatrices.** Una lista de funciones generatrices para secuencias útiles:

$(1, 1, 1, 1, 1, 1, \ldots)$	$\frac{1}{1-z}$
$(1, -1, 1, -1, 1, -1, \ldots)$	$\frac{1}{1+z}$
$(1, 0, 1, 0, 1, 0, \ldots)$	$\frac{1}{1-z^2}$
$(1, 0, \ldots, 0, 1, 0, 1, 0, \ldots, 0, 1, 0, \ldots)$	$\frac{1}{1-z^2}$
$(1, 2, 3, 4, 5, 6, \ldots)$	$\frac{1}{(1-z)^2}$
$(1, \binom{m+1}{m}, \binom{m+2}{m}, \binom{m+3}{m}, \ldots)$	$\frac{1}{(1-z)^{m+1}}$
$(1, c, \binom{c+1}{2}, \binom{c+2}{3}, \ldots)$	$\frac{1}{(1-z)^c}$
$(1, c, c^2, c^3, \ldots)$	$\frac{1}{1-cz}$
$(0, 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots)$	$\ln \frac{1}{1-z}$

Truco de manipulación:

$$\frac{1}{1-z}G(z) = \sum_{n}\sum_{k \leq n} g_k z^n$$

13.10. **The twelvefold way.** ¿Cuántas funciones $f\colon N \to X$ hay?

N	X	Any f	Injective	Surjective
dist.	dist.	x^n	$(x)_n$	$x!\left\{ n \atop x \right\}$
indist.	dist.	$\binom{x+n-1}{n}$	$\binom{x}{n}$	$\binom{n-1}{n-x}$
dist.	indist.	$\left\{ n \atop 1 \right\} + \ldots + \left\{ n \atop x \right\}$	$[n \leq x]$	$\left\{ n \atop k \right\}$
indist.	indist.	$p_1(n) + \ldots p_x(n)$	$[n \leq x]$	$p_x(n)$

Where $\binom{a}{b} = \frac{1}{b!}(a)_b$ and $p_x(n)$ is the number of ways to partition the integer n using x summands.

13.11. **Teorema de Euler.** si un grafo conexo, plano es dibujado sobre un plano sin intersección de aristas, y siendo v el número de vértices, e el de aristas y f la cantidad de caras (regiones conectadas por aristas, incluyendo la región externa e infinita), entonces

$$v - e + f = 2$$

13.12. **Burnside's Lemma.** Si X es un conjunto finito y G es un grupo de permutaciones que actúa sobre X, sean $S_x = \{g \in G : g * x = x\}$ y $Fix(g) = \{x \in X : g * x = x\}$. Entonces el número de órbitas está dado por:

$$N = \frac{1}{|G|} \sum_{x \in X} |S_x| = \frac{1}{|G|} \sum_{g \in G} |Fix(g)|$$

13.13. **Ángulo entre dos vectores.** Sea α el ángulo entre $\vec{a}$ y $\vec{b}$:

$$\cos \alpha = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

13.14. **Proyección de un vector.** Proyección de $\vec{a}$ sobre $\vec{b}$:

$$\text{proy}_{\vec{b}} \vec{a} = (\frac{\vec{a} \cdot \vec{b}}{\vec{b} \cdot \vec{b}}) \vec{b}$$