Team Notebook

December 29, 2020

Contents

1 Data Structure

1.1 Persistent segment tree

```
// ask: find a[l] + a[l+1] + a[l+2] +...+ a[r] After the i
     th update query online
// update: a[p]+= v

// ir = 0 which is its index in the initial segment tree
     Also you should have a NEXT_FREE_INDEX = 1 which is
     always the next free index for a node.
void build(int id = ir,int l = 0,int r = n){
 if(r - l < 2){
  s[id] = a[l];
  return ;
 }
 int mid = (l+r)/2;
 L[id] = NEXT_FREE_INDEX ++;
 R[id] = NEXT_FREE_INDEX ++;
 build(L[id], l, mid);
 build(R[id], mid, r);
 s[id] = s[L[id]] + s[R[id]];
}

// Update function : (its return value, is the index of the
     interval in the new version of segment tree and id is
     the index of old one)
int upd(int p, int v,int id,int l = 0,int r = n){
 int ID = NEXT_FREE_INDEX ++; // index of the node in new
      version of segment tree
 if(r - l < 2){
  s[ID] = (a[p] += v);
  return ID;
 }
 int mid = (l+r)/2;
 L[ID] = L[id], R[ID] = R[id]; // in case of not updating
      the interval of left child or right child
 if(p < mid)
  L[ID] = upd(p, v, L[ID], l, mid);
 else
  R[ID] = upd(p, v, R[ID], mid, r);
 return ID;
}

// (For the first query (with index 0) we should run root[0]
      = upd(p, v, ir)
// and for the rest of them, for j - th query se should run
     root[j] = upd(p, v, root[j - 1]) )
int sum(int x,int y,int id,int l = 0,int r = n){
 if(x >= r or l >= y) return 0;
 if(x <= l && r <= y) return s[id];
 int mid = (l+r)/2;
 return sum(x, y, L[id], l, mid) +
        sum(x, y, R[id], mid, r);
}

// (So, we should print the value of sum(x, y, root[i]) )
```

1.2 Segment Tree Lazy Propagation

```
#include <iostream>

#include <vector>

using namespace std;

#define Long long long int

const int N = 200 * 1000;
const Long inf = (Long)1000 * 1000 * 1000 * 1000;
int n;
int ary[N];
Long tree[N << 2];
Long lazy[N << 2];
vector <Long> ans;

void getArray();
void build(int, int, int);
inline int lc(int);
inline int rc(int);
vector <string> split(string, char);
int toInteger(string);
Long minimum(int, int, int, int, int);
void add(int, int, int, int, int, int);
void propagate(int, int, int);
void update(int, int);
void print();

int main(){
    getArray();
    build(1, 0, n - 1);
    int q;
    string line;
    cin >> q;
    getline(cin, line);
    for(int i = 0; i < q; i++){
        getline(cin, line);
        vector <string> command = split(line, ' ');
        int lf = toInteger(command[0]);
        int rg = toInteger(command[1]);
        int l1, l2, r1, r2;
        if(lf > rg){
            l1 = lf;
            r1 = n - 1;
            l2 = 0;
            r2 = rg;
        }
        else{
            l1 = lf;
            r1 = rg;
            l2 = -1;
            r2 = -1;
        }
        if(command.size() == 2){
            Long min1 = minimum(1, 0, n - 1, l1, r1);
            Long min2 = minimum(1, 0, n - 1, l2, r2);
            ans.push_back(min(min1, min2));
        }
        else{
            int val = toInteger(command[2]);
            add(1, 0, n - 1, l1, r1, val);
            add(1, 0, n - 1, l2, r2, val);
        }
    }
    print();
    return 0;
}

void getArray(){
    ios_base :: sync_with_stdio(false);
    cin.tie(0);
    cin >> n;
    for(int i = 0; i < n; i++)
        cin >> ary[i];
}

void build(int node, int l, int r){
    if(l == r)
        tree[node] = ary[l];
    else{
        int mid = (l + r) >> 1;
        build(lc(node), l, mid);
        build(rc(node), mid + 1, r);
        tree[node] = min(tree[lc(node)], tree[rc(node)]);
    }
}

inline int lc(int node){
    return node << 1;
```

```
}

inline int rc(int node){
    return node << 1 | 1;
}

vector <string> split(string str, char character){
    vector <string> res;
    string s = "";
    for(int i = 0; i < str.size(); i++){
        char c = str[i];
        if(c == character){
            res.push_back(s);
            s = "";
        }
        else
            s += c;
    }
    res.push_back(s);
    return res;
}

int toInteger(string str){
    int res = 0;
    bool positive = true;
    char zero = '0';
    for(int i = 0; i < str.size(); i++){
        char c = str[i];
        if(c == '-')
            positive = false;
        else{
            int d = int(c) - int(zero);
            res = res * 10 + d;
        }
    }
    if(!positive)
        res *= -1;
    return res;
}

Long minimum(int node, int l, int r, int beg, int end){
    if(l > end || r < beg)
        return inf;
    else if(l >= beg && r <= end)
        return tree[node];
    else{
        propagate(node, l, r);
        int mid = (l + r) >> 1;
        Long min1 = minimum(lc(node), l, mid, beg, end);
        Long min2 = minimum(rc(node), mid + 1, r, beg, end);
        return min(min1, min2);
    }
}

void add(int node, int l, int r, int beg, int end, int val){
    if(l > end || r < beg)
        return;
    else if(l >= beg && r <= end)
        update(node, val);
    else{
        propagate(node, l, r);
        int mid = (l + r) >> 1;
        add(lc(node), l, mid, beg, end, val);
        add(rc(node), mid + 1, r, beg, end, val);
        tree[node] = min(tree[lc(node)], tree[rc(node)]);
    }
}

void propagate(int node, int l, int r){
    if(l < r){
        int mid = (l + r) >> 1;
        update(lc(node), lazy[node]);
        update(rc(node), lazy[node]);
    }
    lazy[node] = 0;
}

void update(int node, int val){
    lazy[node] += val;
    tree[node] += val;
}

void print(){
    for(int i = 0; i < ans.size(); i++)
        cout << ans[i] << endl;
}
```

1.3 heavy light decomposition

```
void dfs_sz(int v = 0) {
    sz[v] = 1;
    for(auto &u: g[v]) {
        dfs_sz(u);
        sz[v] += sz[u];
        if(sz[u] > sz[g[v][0]]) {
            swap(u, g[v][0]);
        }
    }
}

void dfs_hld(int v = 0) {
    in[v] = t++;
    for(auto u: g[v]) {
        nxt[u] = (u == g[v][0] ? nxt[v] : u);
        dfs_hld(u);
    }
    out[v] = t;
}

/*
Then you will have such array that subtree of V correspond
    to segment [in(v), out(v))
and the path from V to the last vertex in ascending heavy
    path from V(which is nxt(v))
will be [in(nxt(v)), in(v)] subsegment
which gives you the opportunity to process queries on pathes
and subtrees simultaneously in the same segment tree.
*/
```

1.4 mo$_Complexity_improve$

```
// Complexity
// Sorting all queries will take O(QlogQ).

// How about the other operations? How many times will the
    add and remove be called?

// Let's say the block size is S.

// If we only look at all queries having the left index in
    the same block,
// the queries are sorted by the right index.
// Therefore we will call add(cur_r) and remove(cur_r) only
    O(N) times for all these queries combined.
// This gives O((N/S)*N) calls for all blocks.

// The value of cur_l can change by at most O(S) during
    between two queries.
// Therefore we have an additional O(SQ) calls of add(cur_l)
     and remove(cur_l).

// For   SN   this gives O((N+Q) N ) operations in total.
// Thus the complexity is O((N+Q)FN) where O(F) is the
    complexity of add and remove function.

// Tips for improving runtime
// Block size of precisely N doesn't always offer the best
    runtime.
```

```
// For example, if N=750 then it may happen that block size
    of 700 or 800 may run better.
// More importantly, don't compute the block size at runtime
    - make it const.
// Division by constants is well optimized by compilers.
// In odd blocks sort the right index in ascending order and
    in even blocks sort it in descending order.
// This will minimize the movement of right pointer,
// as the normal sorting will move the right pointer from
    the end back to the beginning at the start of every
    block.
// With the improved version this resetting is no more
    necessary.

bool cmp(pair<int, int> p, pair<int, int> q) {
    if (p.first / BLOCK_SIZE != q.first / BLOCK_SIZE)
        return p < q;
    return (p.first / BLOCK_SIZE & 1) ? (p.second < q.second)
        : (p.second > q.second);
}
```

1.5 mo$_a lgorithm$

```
void remove(idx); // TODO: remove value at idx from data
    structure
void add(idx);   // TODO: add value at idx from data
    structure
int get_answer(); // TODO: extract the current answer of the
    data structure

int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the
        range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

1.6 order$_s et$

```
// C++ program to demonstrate the
// ordered set in GNU C++
#include <iostream>
using namespace std;

// Header files, namespaces,
// macros as defined above
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define ordered_set tree<int, null_type,less<int>,
    rb_tree_tag,tree_order_statistics_node_update>

// Driver program to test above functions
int main()
{
    // Ordered set declared with name o_set
    ordered_set o_set;

    // insert function to insert in
    // ordered set same as SET STL
    o_set.insert(5);
    o_set.insert(1);
    o_set.insert(2);

    // Finding the second smallest element
    // in the set using * because
    // find_by_order returns an iterator
    cout << *(o_set.find_by_order(1))
        << endl;

    // Finding the number of elements
    // strictly less than k=4
    cout << o_set.order_of_key(4)
        << endl;

    // Finding the count of elements less
    // than or equal to 4 i.e. strictly less
    // than 5 if integers are present
    cout << o_set.order_of_key(5)
        << endl;

    // Deleting 2 from the set if it exists
    if (o_set.find(2) != o_set.end())
        o_set.erase(o_set.find(2));

    // Now after deleting 2 from the set
    // Finding the second smallest element in the set
    cout << *(o_set.find_by_order(1))
        << endl;

    // Finding the number of
    // elements strictly less than k=4
    cout << o_set.order_of_key(4)
        << endl;

    return 0;
}
```

2 Dynamic Programming

2.1 Convex$_T rick$

```
struct Line {
    ll k, b;

    Line() {
        k = b = 0ll;
    }
```

```
    Line(ll k, ll b) : k(k), b(b) {}

    ll get(ll x) {
        return k * x + b;
    }
};

ld interLine(Line a, Line b) {
    return (ld)(a.b - b.b) / (ld)(b.k - a.k);
}

struct CHT {
    vector<Line> V;

    CHT() {
        V.clear();
    }

    void addLine(Line l) {
        while (V.size() >= 2 && interLine(V[V.size() - 2], l)
              < interLine(V[V.size() - 2], V.back())) {
            V.pop_back();
        }

        V.push_back(l);
    }

    ll get(ll x) {
        int l = 0, r = (int)V.size() - 2, idx = (int)V.size()
              - 1;

        while (l <= r) {
            int mid = (l + r) >> 1;

            if (interLine(V[mid], V[mid + 1]) <= x) {
                l = mid + 1;
            }
            else {
                r = mid - 1;
                idx = mid;
            }
        }

        return V[idx].get(x);
    }
};

struct Hull_Static{
    /**
    all m need to be decreasing order
    if m is in increasing order then negate the m ( like
        , add_line(-m,c) ),
       remember in query you have to negate the x also
**/
const ll inf = 1000000000000000000;
int min_or_max; ///if min then 0 otherwise 1
int pointer; /// keep track for the best line for
     previous query, requires all insert first;
vector < ll > M, C; ///y = m * x + c;

inline void clear(){
    min_or_max = 0; ///initially with minimum trick
    pointer = 0;
    M.clear();
    C.clear();
}

Hull_Static(){
    clear();
}
Hull_Static(int _min_or_max){
    clear();
    this->min_or_max = _min_or_max;
}
bool bad_min(int idx1, int idx2, int idx3){
    //return (C[idx3] - C[idx1]) * (M[idx1] - M[idx2]) <
        (C[idx2] - C[idx1]) * (M[idx1] - M[idx3]);
    return 1.0 * (C[idx3] - C[idx1]) * (M[idx1] - M[idx2
        ]) <= 1.0 *(C[idx2] - C[idx1]) * (M[idx1] - M[
        idx3]); /// for overflow
}

bool bad_max(int idx1, int idx2, int idx3){
    //return (C[idx3] - C[idx1]) * (M[idx1] - M[idx2]) >
        (C[idx2] - C[idx1]) * (M[idx1] - M[idx3]);
    return 1.0 * (C[idx3] - C[idx1]) * (M[idx1] - M[idx2
        ]) >= 1.0 *(C[idx2] - C[idx1]) * (M[idx1] - M[
        idx3]); /// for overflow
}

bool bad(int idx1, int idx2, int idx3){ /// for removing
     line, which isn't necessary
    if(!min_or_max) return bad_min(idx1, idx2, idx3);
    else return bad_max(idx1, idx2, idx3);
}

void add_line(ll m, ll c){ /// add line where m is given
     in decreasing order
    //if(M.size() > 0 and M.back() == m) return; /// same
         gradient, no need to add
    M.push_back(m);
    C.push_back(c);

    while(M.size() >= 3 and bad((int)M.size() - 3, (int)M
         .size() - 2, (int)M.size() - 1)){
        M.erase(M.end() - 2);
        C.erase(C.end() - 2);
    }
}
ll getval(ll idx, ll x){ /// get the y coordinate of a
     specific line
    return M[idx] * x + C[idx];
}

ll getminval(ll x){ /// if queries are sorted, make sure
     all insertion first.
    while(pointer < (int)M.size() - 1 and getval(pointer
         + 1, x) < getval(pointer, x)) pointer++;
    return M[pointer] * x + C[pointer];
}
ll getmaxval(ll x){ /// if queries are sorted, make sure
     all insertion first.
    while(pointer < (int)M.size() - 1 and getval(pointer
         + 1, x) > getval(pointer, x)) pointer++;
    return M[pointer] * x + C[pointer];
}
ll getminvalternary(ll x){ /// minimum value with ternary
      search
    ll lo = 0;
    ll hi = (ll)M.size() - 1;
    ll ans = inf;
    while(lo <= hi){
        ll mid1 = lo + (hi - lo) / 3;
        ll mid2 = hi - (hi - lo) / 3;
        ll val1 = getval(mid1, x);
        ll val2 = getval(mid2, x);
        if(val1 < val2){
            ans = min(ans, val2);
            hi = mid2 - 1;
        }
        else{
            ans = min(ans, val1);
            lo = mid1 + 1;
        }
    }
    return ans;
}
```

```
    ll getmaxvalternary(ll x){ /// maximum value with ternary
         search
//       cout<<M.size()<<endl;
        ll lo = 0;
        ll hi = (ll)M.size() - 1;
        ll ans = -inf;
        while(lo <= hi){
            ll mid1 = lo + (hi - lo) / 3;
            ll mid2 = hi - (hi - lo) / 3;
            ll val1 = getval(mid1, x);
            ll val2 = getval(mid2, x);
            if(val1 < val2){
                ans = max(ans, val2);
                lo = mid1 + 1;
            }
            else{
                ans = max(ans, val1);
                hi = mid2 - 1;
            }
        }
        return ans;
    }

};
```

2.2 Divide and Conquer Optimization

```
// Divide and Conquer Optimization
// dp[i][j]  =  mink  < j {dp[ i - 1 ][k]   + C  [k][j]} A[i
     ][j]   A   [i][ j + 1 ]
// A[i][j]    the smallest k that gives optimal answer, for
     example in dp[i][j]=dp[ i - 1 ][k]   + C  [k][j]
// C[i][j]    some given cost function

int n , k , sum[N][N] , dp[K][N] ;

void calc(int ind,int l,int r,int opl,int opr)
{
 if(l>r) return;
 int m=(l+r)/2,op,res=1e9;
 for(int i=min(opr,m);i>=opl;i--)
 {
  int ret=dp[ind-1][i-1]+sum[m][m]+sum[i-1][i-1]-sum[i-1][m
      ]-sum[m][i-1];
  if(res>=ret)
   res=ret,op=i;
 }
 dp[ind][m]=res;
 if(l==r)
  return;
 calc(ind,l,m-1,opl,op);
 calc(ind,m+1,r,op,opr);
}
```

2.3 Knuth Optimization

```
for (int s = 0; s<=k; s++)             //s - length(size)
     of substring
    for (int L = 0; L+s<=k; L++) {          //L - left point
      int R = L + s;                        //R - right point
      if (s < 2) {
        res[L][R] = 0;                      //DP base -
           nothing to break
        mid[L][R] = l;                      //mid is equal to
            left border
        continue;
      }
      int mleft = mid[L][R-1];              //Knuth's trick:
          getting bounds on M
      int mright = mid[L+1][R];
      res[L][R] = 10000000000000000000LL;
      for (int M = mleft; M<=mright; M++) { //iterating for M
           in the bounds only
        int64 tres = res[L][M] + res[M][R] + (x[R]-x[L]);
        if (res[L][R] > tres) {             //relax current
            solution
          res[L][R] = tres;
          mid[L][R] = M;
        }
      }
    }
  int64 answer = res[0][k];
```

3 Geometry

3.1 3D Rotation

```
//From "You Know Izad?" team cheat sheet
Where c = cos (theta), s = sin(theta), t = 1-cos(theta), and
      <X,Y,Z> is the unit vector representing the arbitary
     axis
1. Left handed about arbitrary axis:
 tX^2+c  tXY-sZ  tXZ+sY 0
 tXY+sZ  tY^2+c  tYZ-sX 0
 tXZ-sY  tYZ+sX  tZ^2+c 0
 0  0  0 1

2. Right handed about arbitrary axis:
 tX^2+c  tXY+sZ  tXZ-sY 0
 tXY-sZ  tY^2+c  tYZ+sX 0
 tXZ+sY  tYZ-sX  tZ^2+c 0
 0  0  0 1

3. About X Axis
 1 0 0 0
 0 c -s 0
 0 s c 0
 0 0 0 1


4. About Y Axis
 c 0 s 0
 0 1 0 0
 -s 0 c 0
 0 0 0 1

5. About Z Axis
 c -s 0 0
 s c 0 0
 0 0 1 0
 0 0 0 1
```

3.2 Angle Bisector

```
// angle bisector
int bcenter( PT p1, PT p2, PT p3, PT& r ){
 if( triarea( p1, p2, p3 ) < EPS ) return -1;
 double s1, s2, s3;
 s1 = dist( p2, p3 );
 s2 = dist( p1, p3 );
 s3 = dist( p1, p2 );
 double rt = s2/(s2+s3);
 PT a1,a2;
 a1 = p2*rt+p3*(1.0-rt);
 rt = s1/(s1+s3);
 a2 = p1*rt+p3*(1.0-rt);
 intersection( a1,p1, a2,p2, r );
 return 0;
}
```

3.3 Circle Circle Intersection

```
// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p)  { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// compute intersection of circle centered at a with radius
    r
// with circle centered at b with radius R
vector<PT> CircleCircleIntersection(PT a, PT b, double r,
     double R) {
  vector<PT> ret;
  double d = sqrt(dist2(a, b));
  if (d > r + R || d + min(r, R) < max(r, R)) return ret;
  double x = (d * d - R * R + r * r) / (2 * d);
  double y = sqrt(r * r - x * x);
  PT v = (b - a) / d;
  ret.push_back(a + v * x + RotateCCW90(v) * y);
  if (y > 0)
    ret.push_back(a + v * x - RotateCCW90(v) * y);
  return ret;
}
```

3.4 Circle Line Intersection

```
// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r
     ) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}
```

3.5 Circle from Three Points

```
Point center_from(double bx, double by, double cx, double cy
     ) {
  double B=bx*bx+by*by, C=cx*cx+cy*cy, D=bx*cy-by*cx;
  return Point((cy*B-by*C)/(2*D), (bx*C-cx*B)/(2*D));
}

Point circle_from(Point A, Point B, Point C) {
  Point I = center_from(B.X-A.X, B.Y-A.Y, C.X-A.X, C.Y-A.Y);
  return Point(I.X + A.X, I.Y + A.Y);
}
```

3.6 Closest Pair of Points

```
struct point {
  double x, y;
  int id;
  point() {}
  point (double a, double b) : x(a), y(b) {}
};

double dist(const point &o, const point &p) {
  double a = p.x - o.x, b = p.y - o.y;
  return sqrt(a * a + b * b);
}

double cp(vector<point> &p, vector<point> &x, vector<point>
     &y) {
  if (p.size() < 4) {
    double best = 1e100;
    for (int i = 0; i < p.size(); ++i)
      for (int j = i + 1; j < p.size(); ++j)
        best = min(best, dist(p[i], p[j]));
    return best;
  }

  int ls = (p.size() + 1) >> 1;
  double l = (p[ls - 1].x + p[ls].x) * 0.5;
  vector<point> xl(ls), xr(p.size() - ls);
  unordered_set<int> left;
  for (int i = 0; i < ls; ++i) {
    xl[i] = x[i];
    left.insert(x[i].id);
  }
  for (int i = ls; i < p.size(); ++i) {
    xr[i - ls] = x[i];
  }

  vector<point> yl, yr;
  vector<point> pl, pr;
  yl.reserve(ls); yr.reserve(p.size() - ls);
  pl.reserve(ls); pr.reserve(p.size() - ls);
  for (int i = 0; i < p.size(); ++i) {
    if (left.count(y[i].id))
      yl.push_back(y[i]);
    else
      yr.push_back(y[i]);

    if (left.count(p[i].id))
      pl.push_back(p[i]);
    else
      pr.push_back(p[i]);
  }

  double dl = cp(pl, xl, yl);
  double dr = cp(pr, xr, yr);
  double d = min(dl, dr);
  vector<point> yp; yp.reserve(p.size());
  for (int i = 0; i < p.size(); ++i) {
    if (fabs(y[i].x - l) < d)
      yp.push_back(y[i]);
  }
  for (int i = 0; i < yp.size(); ++i) {
    for (int j = i + 1; j < yp.size() && j < i + 7; ++j) {
      d = min(d, dist(yp[i], yp[j]));
    }
  }
  return d;
}

double closest_pair(vector<point> &p) {
  vector<point> x(p.begin(), p.end());
  sort(x.begin(), x.end(), [](const point &a, const point &b
      ) {
    return a.x < b.x;
  });
  vector<point> y(p.begin(), p.end());
  sort(y.begin(), y.end(), [](const point &a, const point &b
      ) {
    return a.y < b.y;
  });
  return cp(p, x, y);
}
```

3.7 Closest Point on Line

```
//From In 1010101 We Trust cheatsheet:
//the closest point on the line p1->p2 to p3
void closestpt( PT p1, PT p2, PT p3, PT &r ){
```

```
 if(fabs(triarea(p1, p2, p3)) < EPS){ r = p3; return; }
 PT v = p2-p1; v.normalize();
 double pr; // inner product
 pr = (p3.y-p1.y)*v.y + (p3.x-p1.x)*v.x;
 r = p1+v*pr;
}
```

3.8 Convexhull

```
ll cross(Point a , Point b){
    return a.x * b.y - a.y *b.x;
}

void convex(){
 sort(points.begin(), points.end());
 int m = 0;
 fore(i,0,points.size()-1){
        while (m > 1 && cross(CH[m-1] - CH[m-2] , points[i] -
               CH[m-2]) <= 0){
            CH.pop_back();
            m--;
        }
        CH.push_back(points[i]);
        m++;
    }

    int k = m;
    forn(i,points.size()-2 , 0){
        while (m > k && cross(CH[m-1] - CH[m-2] , points[i] -
               CH[m-2]) <= 0){
            CH.pop_back();
            m--;
        }
        CH.push_back(points[i]);
        m++;
    }
}

ld area() {
    ld sum = 0;
    int i;
    fore(i,0,CH.size()-2){
        sum += (CH[i].x*CH[i+1].y - CH[i].y*CH[i+1].x);
    }
    return fabs(sum/2);
}
```

3.9 Delaunay Triangulation

```
// Slow but simple Delaunay triangulation. Does not handle
// degenerate cases (from O'Rourke, Computational Geometry
     in C)
//
// Running time: O(n^4)
//
// INPUT:   x[] = x-coordinates
//          y[] = y-coordinates
//
// OUTPUT:  triples = a vector containing m triples of
     indices
//                  corresponding to triangle vertices

typedef double T;

struct triple {
    int i, j, k;
    triple() {}
    triple(int i, int j, int k) : i(i), j(j), k(k) {}
};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T
     >& y) {
 int n = x.size();
 vector<T> z(n);
 vector<triple> ret;

 for (int i = 0; i < n; i++)
     z[i] = x[i] * x[i] + y[i] * y[i];

 for (int i = 0; i < n-2; i++) {
     for (int j = i+1; j < n; j++) {
  for (int k = i+1; k < n; k++) {
      if (j == k) continue;
      double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j
          ]-z[i]);
      double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k
          ]-z[i]);
      double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j
          ]-y[i]);
      bool flag = zn < 0;
      for (int m = 0; flag && m < n; m++)
   flag = flag && ((x[m]-x[i])*xn +
     (y[m]-y[i])*yn +
     (z[m]-z[i])*zn <= 0);
      if (flag) ret.push_back(triple(i, j, k));
  }
     }
 }
 return ret;
}

int main()
{
    T xs[]={0, 0, 1, 0.9};
    T ys[]={0, 1, 0, 0.9};
    vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
    vector<triple> tri = delaunayTriangulation(x, y);

    //expected: 0 1 3
    //          0 3 2

    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

3.10 Latitude and Longitude

```
/*
Converts from rectangular coordinates to latitude/longitude
     and vice
versa. Uses degrees (not radians).
*/

using namespace std;

struct ll
{
  double r, lat, lon;
};

struct rect
{
  double x, y, z;
};

ll convert(rect& P)
{
  ll Q;
  Q.r = sqrt(P.x*P.x+P.y*P.y+P.z*P.z);
  Q.lat = 180/M_PI*asin(P.z/Q.r);
  Q.lon = 180/M_PI*acos(P.x/sqrt(P.x*P.x+P.y*P.y));

  return Q;
}
```

```
rect convert(ll& Q)
{
  rect P;
  P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
  P.z = Q.r*sin(Q.lat*M_PI/180);

  return P;
}

int main()
{
  rect A;
  ll B;

  A.x = -1.0; A.y = 2.0; A.z = -3.0;

  B = convert(A);
  cout << B.r << " " << B.lat << " " << B.lon << endl;

  A = convert(B);
  cout << A.x << " " << A.y << " " << A.z << endl;
}
```

3.11 Line Intersection

```
// Ax + By = C
A = y2 - y1
B = x1 - x2
C = A*x1 + B*y1
double det = A1*B2 - A2*B1
double x = (B2*C1 - B1*C2)/det
double y = (A1*C2 - A2*C1)/det

typedef pair<double, double> pointd;
#define X first
#define Y second
bool eqf(double a, double b) {
    return fabs(b - a) < 1e-6;
}
int crossVecs(pointd a, pointd b) {
    return a.X * b.Y - a.Y*b.X;
}
int cross(pointd o, pointd a, pointd b){
    return crossVecs(make_pair(a.X - o.X, a.Y - o.Y),
        make_pair(b.X - o.X, b.Y - o.Y));
}
int dotVecs(pointd a, pointd b) {
    return a.X * b.X + a.Y * b.Y;
}
int dot(pointd o, pointd a, pointd b) {
    return dotVecs(make_pair(a.X - o.X, a.Y - o.Y), make_pair
        (b.X - o.X, b.Y - o.Y));
}
bool onTheLine(const pointd& a, const pointd& p, const
    pointd& b) {
    return eqf(cross(p, a, b), 0) && dot(p, a, b) < 0 ;
}
class LineSegment {
    public:
    double A, B, C;
    pointd from, to;
    LineSegment(const pointd& a, const pointd& b) {
        A = b.Y - a.Y;
        B = a.X - b.X;
        C = A*a.X + B*a.Y;
        from = a;
        to = b;
    }

    bool between(double l, double a, double r) const {
        if(l > r) {
            swap(l, r);
        }
        return l <= a && a <= r;
    }

    bool pointOnSegment(const pointd& p) const {
        return eqf(A*p.X + B*p.Y, C) && between(from.X, p.X,
            to.X) && between(from.Y, p.Y, to.Y);
    }

    pair<bool, pointd> segmentsIntersect(const LineSegment& l
        ) const {
        double det = A * l.B - B * l.A;
        pair<bool, pointd> ret;
        ret.first = false;
        if(det != 0) {
            pointd inter((l.B*C - B*l.C)/det, (A*l.C - l.A*C)
                /det);
            if(l.pointOnSegment(inter) && pointOnSegment(
                inter)) {
                ret.first = true;
                ret.second = inter;
            }
        }
        return ret;
    }
};
```

3.12 Point in Polygon

```
// determine if point is in a possibly non-convex polygon (
    by William
// Randolph Franklin); returns 1 for strictly interior
    points, 0 for
// strictly exterior points, and 0 or 1 for the remaining
    points.
// Note that it is possible to convert this into an *exact*
    test using
// integer arithmetic by taking care of the division
    appropriately
// (making sure to deal with signs properly) and then by
    writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[
          j].y - p[i].y))
      c = !c;
  }
  return c;
}
```

3.13 Polygon Centroid

```
// This code computes the area or centroid of a (possibly
    nonconvex)
// polygon, assuming that the coordinates are listed in a
    clockwise or
// counterclockwise fashion. Note that the centroid is often
     known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}
```

```
double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}
```

3.14 Rotation Around Origin by t

```
 x   = x.Cos(t) - y.Sin(t)
 y   = x.Sin(t) + y.Cos(t)
```

3.15 Two Point and Radius Circle

```
vector<point> find_center(point a, point b, long double r) {
  point d = (a - b) * 0.5;
  if (d.dot(d) > r * r) {
    return vector<point> ();
  }
  point e = b + d;
  long double fac = sqrt(r * r - d.dot(d));
  vector<point> ans;
  point x = point(-d.y, d.x);
  long double l = sqrt(x.dot(x));
  x = x * (fac / l);
  ans.push_back(e + x);
  x =  point(d.y, -d.x);
  x = x * (fac / l);
  ans.push_back(e + x);
  return ans;
}
```

3.16 geometry algorithms

```
Line(Point p1 , Point p2){
 a = p2.y - p1.y;
 b = p1.x - p2.x;
 c = a * p1.x + b * p1.y;
 c = -c;

}

Point intersection(Line l1 , Line l2){
 ld a1 = l1.a;
 ld b1 = l1.b;
 ld c1 = -l1.c;
 ld a2 = l2.a;
 ld b2 = l2.b;
 ld c2 = -l2.c;
 ld determinant = a1*b2 - a2*b1;
 ld x = (b2*c1 - b1*c2)/determinant;
 ld y = (a1*c2 - a2*c1)/determinant;
 return Point(x, y);
}

Point mirrorImage(Point p , Line l)
{
 ld a = l.a;
 ld b = l.b;
 ld c = l.c;
 ld x1 = p.x;
 ld y1 = p.y;
 ld temp = -2 * (a * x1 + b * y1 + c) /
 (a * a + b * b);
 ld x = temp * a + x1;
 ld y = temp * b + y1;
 return Point(x, y);
}

ld pointToLine(Point p0, Point p1, Point p2){
 //p0 to (p1 , p2)
 ll x0 = p0.x;
 ll y0 = p0.y;
 ll x1 = p1.x;
 ll y1 = p1.y;
 ll x2 = p2.x;
 ll y2 = p2.y;
 ld a = ((y2 - y1)*x0 - (x2 - x1)*y0 + x2 * y1 - y2 * x1);
 ld b = (y2 - y1)*(y2 - y1) + (x2 - x1)*(x2 - x1);
 return a * a / b;
}

inline p3d rotate(const p3d& p /*pt*/, const p3d& u /*axis*/
     , const ld& angle) {
//p center u
ld c = cos(angle), s = sin(angle), t = 1 - cos(angle);
     return {
 p.x*(t*u.x*u.x + c) + p.y*(t*u.x*u.y - s*u.z) + p.z*(t*u.x*
      u.z + s*u.y),
 p.x*(t*u.x*u.y + s*u.z) + p.y*(t*u.y*u.y + c) + p.z*(t*u.y*
      u.z - s*u.x),
 p.x*(t*u.x*u.z - s*u.y) + p.y*(t*u.y*u.z + s*u.x) + p.z*(t*
      u.z*u.z + c) };
}


int cmp(ld x){
 if (fabs(x) < eps)
  return 0;
 return ((x < 0 ) ? -1 : 1);
}

ld Dot( const Vec2& a, const Vec2& b )
{
    return a.x * b.x + a.y * b.y;
}

int orientation(Point p, Point q, Point r)
{
 ld val = (q.y - p.y) * (r.x - q.x) -
 (q.x - p.x) * (r.y - q.y);

 if (cmp(val) == 0) return 0;
 return (cmp(val) > 0)? 1: 2;
}

bool onSegment(Point p, Point q, Point r)
{
 // (p , r) point q
    if (cmp(q.x - max(p.x, r.x)) >= 0 && cmp(q.x - min(p.x, r
        .x)) <= 0 &&
           cmp(q.y - max(p.y, r.y)) >= 0 && cmp(q.y - min(p.
                y, r.y)) <= 0 )
        return true;
    return false;
}

bool doIntersect(Point p1, Point q1, Point p2, Point q2)
{
 // (p1 , q1) intersect (p2 , q2)
 int o1 = orientation(p1, q1, p2);
 int o2 = orientation(p1, q1, q2);
 int o3 = orientation(p2, q2, p1);
 int o4 = orientation(p2, q2, q1);
 if (o1 != o2 && o3 != o4)
  return true;
 if (o1 == 0 && onSegment(p1, p2, q1)) return true;
```

```
 if (o2 == 0 && onSegment(p1, q2, q1)) return true;
 if (o3 == 0 && onSegment(p2, p1, q2)) return true;
 if (o4 == 0 && onSegment(p2, q1, q2)) return true;
    return false; // Doesn't fall in any of the above cases
}

bool isInside(Point p)
{
 if (n < 3) return false;
 Point extreme = {1e18, p.y};
 int count = 0, i = 0;
 do
 {
  int next = (i+1)%n;
  if (doIntersect(polygon[i], polygon[next], p, extreme))
  {
   if (orientation(polygon[i], p, polygon[next]) == 0)
    return onSegment(polygon[i], p, polygon[next]);

   count++;
  }
  i = next;
 } while (i != 0);
 return count&1;
}
ld cross(Vec2 a , Vec2 b){
 return a.x * b.y - a.y * b.x;
}
ld len(Vec2 a){
 return hypotl(a.x , a.y);
}
ld SqDistancePtSegment( Vec2 a, Vec2 b, Vec2 p )
{
 Vec2 v1 = b - a;
 Vec2 v2 = p - a;
 Vec2 v3 = p - b;
 if (cmp(Dot(v1 , v2)) < 0)return len(v2);
 if (cmp(Dot(v1 , v3)) > 0) return len(v3);
 return fabs(cross(v1 , v2)) /len(v1);
}
Point F( int i ,int j , int k){
 Vec2 a , b , c;
 a.x = polygon[i].x;
 a.y = polygon[i].y;
 b.x = polygon[j].x;
 b.y = polygon[j].y;
 c.x = polygon[k].x;
 c.y = polygon[k].y;
 Vec2 v1 = b - a;
 Vec2 v2 = c - a;
 Vec2 nimsaz = (v2 * len(v1)) + (v1 * len(v2)) ;
 ld sz = len(nimsaz);
 nimsaz.x /= sz;
 nimsaz.y /= sz;
 ld costeta = Dot(nimsaz , v2) / (len(nimsaz) * len(v2));
 ld sinteta = sqrt(1.0 - costeta * costeta);
 ld d = R / sinteta;
 Vec2 point = (nimsaz * d) + a;
 return {point.x , point.y};

}

Point rotate(Point c , Point p , ld angle)
{
    ld sn = sin(angle);
    ld cs = cos(angle);
    Point q(cs*(p.x-c.x) - sn *(p.y-c.y) + c.x , sn *(p.x-c.x
        ) + cs * (p.y-c.y) + c.y);
    return q;
}
```

4 Graph

4.1 2-SAT

```
//From "You Know Izad?" team cheat sheet
//fill the v array
//e.g. to push (p v !q) use the following code:
//  v[VAR(p)].push_back( NOT( VAR(q) ) )
//  v[NOT( VAR(q) )].push_back( VAR(p) )
//the result will be in color array
#define VAR(X) (X << 1)
#define NOT(X) (X ^ 1)
#define CVAR(X,Y) (VAR(X) | (Y))z
#define COL(X) (X & 1)
#define NVAR 400
int n;
vector<int> v[2 * NVAR];
int color[2 * NVAR];
int bc[2 * NVAR];
bool dfs( int a, int col ) {
    color[a] = col;
    int num = CVAR( a, col );
    for( int i = 0; i < v[num].size(); i++ ) {
        int adj = v[num][i] >> 1;
        int ncol = NOT( COL( v[num][i] ) );
        if( ( color[adj] == -1 && !dfs( adj, ncol ) ) ||
            ( color[adj] != -1 && color[adj] != ncol ) ) {
            color[a] = -1;
            return false;
        }
    }
    return true;
}
bool twosat() {
    memset( color, -1, sizeof color );
    for( int i = 0; i < n; i++ ){
        if( color[i] == -1 ){
            memcpy(bc, color, sizeof color);
            if( !dfs( i, 0 )){
                memcpy(color, bc, sizeof color);
                if(!dfs( i, 1 ))
                    return false;
            }
        }
    }
    return true;
}
```

4.2 Bridge and Articulate Point Finding

```
typedef struct {
  int deg;
  int adj[MAX_N];
} Node;

Node alist[MAX_N];
bool art[MAX_N];
int df_num[MAX_N], low[MAX_N], father[MAX_N], count;
int bridge[MAX_N*MAX_N][2], bridges;

void add_bridge(int v1, int v2) {
  bridge[bridges][0] = v1;
  bridge[bridges][1] = v2;
  ++bridges;
}

void search(int v, bool root) {
  int w, child = 0;

  low[v] = df_num[v] = count++;

  for (int i = 0; i < alist[v].deg; ++i) {
    w = alist[v].adj[i];

    if (df_num[w] == -1) {
      father[w] = v;
```

```
      ++child;
      search(w, false);
      if (low[w] > df_num[v]) add_bridge(v, w);
      if (low[w] >= df_num[v] && !root) art[v] = true;
      low[v] = min(low[v], low[w]);
    }
    else if (w != father[v]) {
      low[v] = min(low[v], df_num[w]);
    }
  }

  if (root && child > 1) art[v] = true;
}

void articulate(int n) {
  int child = 0;

  for (int i = 0; i < n; ++i) {
    art[i] = false;
    df_num[i] = -1;
    father[i] = -1;
  }

  count = bridges = 0;

  search(0, true);
}
```

4.3 Count Triangles

```
vector <int> adj[maxn], Adj[maxn];

int ord[maxn], f[maxn], fi[maxn], se[maxn], ans[maxn];

bool get(int v,int u) {
 int idx = lower_bound(adj[v].begin(), adj[v].end(), u) -
     adj[v].begin();
 if (idx != adj[v].size() && adj[v][idx] == u)
  return true;
 return false;
}

bool cmp(int v,int u) {
 if (adj[v].size() < adj[u].size())
  return true;
 if (adj[v].size() > adj[u].size())
  return false;
 return (v < u);
}

int main() {
 int n, m, q;
 cin >> n >> m >> q;
 for (int i = 0; i < m; i++) {
  cin >> fi[i] >> se[i];
  fi[i]--, se[i]--;
  adj[fi[i]].push_back(se[i]);
  adj[se[i]].push_back(fi[i]);
  Adj[fi[i]].push_back(se[i]);
  Adj[se[i]].push_back(fi[i]);
 }
 for (int i = 0; i < n; i++)
  sort(adj[i].begin(), adj[i].end()),
  sort(Adj[i].begin(), Adj[i].end(), cmp);
 for (int i = 0; i < n; i++)
  ord[i] = i;
 sort (ord, ord + n, cmp);
 for (int i = 0; i < n; i++)
  f[ord[i]] = i;
 for (int v = 0; v < n; v++) {
  int idx = -1;
  for (int j = 0; j < adj[v].size(); j++) {
   int u = Adj[v][j];
   if (f[u] > f[v])
    break;
   idx = j;
  }
  for (int i = 0; i <= idx; i++)
   for (int j = 0; j < i; j++) {
    int u = Adj[v][i];
    int w = Adj[v][j];
    if (get(u,w))
     ans[v]++, ans[u]++, ans[w]++;
   }
 }
 for (int i = 0; i < q; i++) {
  int v;
  cin >> v;
  v--;
  cout << ans[v] << '\n';
 }
 return 0;
}
```

4.4 Eulerian Path

```
// Taken from https://github.com/lbv/pc-code/blob/master/
     code/graph.cpp
// Eulerian Trail

struct Euler {
  ELV adj; IV t;
  Euler(ELV Adj) : adj(Adj) {}
  void build(int u) {
    while(! adj[u].empty()) {
      int v = adj[u].front().v;
      adj[u].erase(adj[u].begin());
      build(v);
    }
    t.push_back(u);
  }
};
bool eulerian_trail(IV &trail) {
  Euler e(adj);
  int odd = 0, s = 0;
  /*
     for (int v = 0; v < n; v++) {
     int diff = abs(in[v] - out[v]);
     if (diff > 1) return false;
     if (diff == 1) {
     if (++odd > 2) return false;
     if (out[v] > in[v]) start = v;
     }
     }
     */
  e.build(s);
  reverse(e.t.begin(), e.t.end());
  trail = e.t;
  return true;
}
```

4.5 Euler Tour

```
//  DirectedEulerTourO  (  E  )
void visit (Graph& g, int a , vector<int>& path) {
 while (!g[a].empty()){
  int b = g[a].back().dst;
  g[a].pop_back();
  visit (g, b, path);
 }
 path.push_back (a);
}
bool eulerPath (Graph g, int s , vector<int> &path) {
 int n = g.size(), m = 0;
 vector<int> deg (n);
 REP (u , n) {
  m += g[u].size();
```

```
  FOR (e , g[u]) --deg[e->dst]; // in-deg
  deg[u] += g[u].size();   // out-deg
}
int k = n - count (ALL (deg), 0);
if (k == 0 || (k == 2 && deg[s] == 1)) {
 path.clear();
 visit (g, s , path);
 reverse (ALL (path));
 return path.size () == m + 1;
}
return false;
}
//  UndirectedEulerTourO  ( E  )
void visit(const Graph &g, vector< vector<int> > &adj, int s
     , vector<int> &path) {
 FOR (e , g[s])
 if (adj[e->src][e->dst]) {
  --adj[e->src][e->dst];
  --adj[e->dst][e->src];
  visit(g, adj, e->dst , path);
 }
 path.push_back(s);
}
bool eulerPath (const Graph &g, int s , vector<int> &path)
{
 int n = g.size();
 int odd = 0, m = 0;
 REP (i, n) {
  if (g[i].size() % 2 == 1)
   ++odd;
  m += g[i].size();
 }
 m/= 2;
 if (odd == 0 || (odd == 2 && g[s].size() % 2 == 0))
 {
  vector< vector<int> > adj (n , vector<int> (n));

  REP (u , n) FOR (e , g[u]) ++adj[e->src][e->dst];
  path.clear ();
  visit (g, adj, s, path);
  reverse (ALL (path));
  return path.size() == m + 1;
 }
 return false;
}
```

4.6 LCA

```
void dfsLCA(int u , int p){
    tin[u] = ++timer;
    up[u][0] = p;
    fore(i,1,l){
        up[u][i] = up[up[u][i-1]][i-1];
    }
    for(int v : tree[u]){
        if (v == p)
            continue;
        dfsLCA(v,u);
    }
    tout[u]=++timer;
}
bool isAnsector(int u , int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u , int v){
    if (isAnsector(u , v))
        return u;
    if (isAnsector(v , u))
        return v;
    forn(i , l , 0){
        if (!isAnsector(up[u][i] , v))
            u = up[u][i];
    }
    return up[u][0];
}

void findLca(){
    memset(visited , false ,sizeof visited);
    memset(lev , 0 , sizeof lev);
    dfs(1 , 0);
    memset(tin , 0 , sizeof tin);
    memset(tout , 0 ,sizeof tout);
    timer = 0;
    l = ceil(log2(n));
    memset(up , 0 ,sizeof up);
    dfsLCA(1,1);
}
```

4.7 Weighted Min Cut

```
// Maximum number of vertices in the graph
#define NN 256

// Maximum edge weight (MAXW * NN * NN must fit into an int)
#define MAXW 1000

// Adjacency matrix and some internal arrays
int g[NN][NN], v[NN], w[NN], na[NN];
bool a[NN];

int minCut( int n )
{
    // init the remaining vertex set
    for( int i = 0; i < n; i++ ) v[i] = i;

    // run Stoer-Wagner
    int best = MAXW * n * n;
    while( n > 1 )
    {
        // initialize the set A and vertex weights
        a[v[0]] = true;
        for( int i = 1; i < n; i++ )
        {
            a[v[i]] = false;
            na[i - 1] = i;
            w[i] = g[v[0]][v[i]];
        }

        // add the other vertices
        int prev = v[0];
        for( int i = 1; i < n; i++ )
        {
            // find the most tightly connected non-A vertex
            int zj = -1;
            for( int j = 1; j < n; j++ )
                if( !a[v[j]] && ( zj < 0 || w[j] > w[zj] ) )
                    zj = j;

            // add it to A
            a[v[zj]] = true;

            // last vertex?
            if( i == n - 1 )
            {
                // remember the cut weight
                best <?= w[zj];

                // merge prev and v[zj]
                for( int j = 0; j < n; j++ )
                    g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][
                        v[j]];
                v[zj] = v[--n];
                break;
            }
            prev = v[zj];
```

```
            // update the weights of its neighbours
            for( int j = 1; j < n; j++ ) if( !a[v[j]] )
                w[j] += g[v[zj]][v[j]];
        }
    }
    return best;
}

int main()
{
    // read the graph's adjacency matrix into g[][]
    // and set n to equal the number of vertices
    int n, answer = minCut( n );
    return 0;
}
```

4.8 assignment Problem

```
int assignment() {
    int n = a.size();
    int m = n * 2 + 2;
    vector<vector<int>> f(m, vector<int>(m));
    int s = m - 2, t = m - 1;
    int cost = 0;
    while (true) {
        vector<int> dist(m, INF);
        vector<int> p(m);
        vector<int> type(m, 2);
        deque<int> q;
        dist[s] = 0;
        p[s] = -1;
        type[s] = 1;
        q.push_back(s);
        while (!q.empty()) {
            int v = q.front();
            q.pop_front();
            type[v] = 0;
            if (v == s) {
                for (int i = 0; i < n; ++i) {
                    if (f[s][i] == 0) {
                        dist[i] = 0;
                        p[i] = s;
                        type[i] = 1;
                        q.push_back(i);
                    }
                }
            } else {
                if (v < n) {
                    for (int j = n; j < n + n; ++j) {
                        if (f[v][j] < 1 && dist[j] > dist[v] +
                            a[v][j - n]) {
                            dist[j] = dist[v] + a[v][j - n];
                            p[j] = v;
                            if (type[j] == 0)
                                q.push_front(j);
                            else if (type[j] == 2)
                                q.push_back(j);
                            type[j] = 1;
                        }
                    }
                } else {
                    for (int j = 0; j < n; ++j) {
                        if (f[v][j] < 0 && dist[j] > dist[v] -
                            a[j][v - n]) {
                            dist[j] = dist[v] - a[j][v - n];
                            p[j] = v;
                            if (type[j] == 0)
                                q.push_front(j);
                            else if (type[j] == 2)
                                q.push_back(j);
                            type[j] = 1;
                        }
                    }
                }
            }
        }

        int curcost = INF;
        for (int i = n; i < n + n; ++i) {
            if (f[i][t] == 0 && dist[i] < curcost) {
                curcost = dist[i];
                p[t] = i;
            }
        }
        if (curcost == INF)
            break;
        cost += curcost;
        for (int cur = t; cur != -1; cur = p[cur]) {
            int prev = p[cur];
            if (prev != -1)
                f[cur][prev] = -(f[prev][cur] = 1);
        }
    }

    // vector<int> answer(n);
    int answer = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (f[i][j + n] == 1)
                answer+=a[i][j];
        }
    }
    return answer;
}
```

4.9 bipartie$_{m}cmf$

```
vector<edge> g[maxn];
int h[maxn], dst[maxn], prevv[maxn ], preve[maxn];
inline void add_edge(int f, int t, int cap, int cost)
{
 g[f].emplace_back(t, cap, cost, g[t].size());
 g[t].emplace_back(f, 0, -cost, g[f].size() - 1);
}

int mcmf(int s, int t , int maxFlow)
{
 int res = 0;
 int c = INT_MAX;
 memset(h, 0, sizeof(h));
  int f = 0;
 while (f < maxFlow) {
  priority_queue<ii, vector<ii>, greater<ii> > que;
  fill(dst, dst + n , inf);
  dst[s] = 0;
  que.push(mp(0, s));
  while (!que.empty()) {
   ii p = que.top(); que.pop();
   int v = p.second;
   if (dst[v] < p.first) continue;
   fore(i , 0 , g[v].size() - 1) {
    edge &e = g[v][i];
    int nd = dst[v] + e.cost + h[v] - h[e.to];
    if (e.cap > 0 && dst[e.to] > nd){
     dst[e.to] = nd;
     prevv[e.to] = v;
     preve[e.to] = i;
     que.push(mp(dst[e.to], e.to));
    }
   }
  }

  if (dst[t] == inf) return c;
  fore(i, 0 , n - 1) h[i] += dst[i];

  int d = inf;
  for(int v = t; v != s; v = prevv[v])
```

```
    d = min(d, g[prevv[v]][preve[v]].cap);
   f += d;
   res += d * h[t];
   c = min(c, res);
   if (res >= 0) break;

   for(int v = t; v != s; v = prevv[v]){
    edge &e = g[prevv[v]][preve[v]];
    e.cap -= d;
    g[v][e.rev].cap += d;
   }
  }

  return c;
 }
```

4.10 flow

```
struct FlowEdge {
    int v, u;
    long long cap, flow = 0;
    FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(
        cap) {}
};

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

    Dinic(int n, int s, int t) : n(n), s(s), t(t) {
        adj.resize(n);
        level.resize(n);
        ptr.resize(n);
    }

    void add_edge(int v, int u, long long cap) {
        // TRACE(v _ u _ cap);
        edges.emplace_back(v, u, cap);
        edges.emplace_back(u, v, 0);
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs() {
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int id : adj[v]) {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q.push(edges[id].u);
            }
        }
        return level[t] != -1;
    }

    long long dfs(int v, long long pushed) {
        if (pushed == 0)
            return 0;
        if (v == t)
            return pushed;
        for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid
            ++) {
            int id = adj[v][cid];
            int u = edges[id].u;
            if (level[v] + 1 != level[u] || edges[id].cap -
                edges[id].flow < 1)
                continue;
            long long tr = dfs(u, min(pushed, edges[id].cap -
                edges[id].flow));
            if (tr == 0)
                continue;
            edges[id].flow += tr;
            edges[id ^ 1].flow -= tr;
            return tr;
        }
        return 0;
    }

    long long flow() {
        long long f = 0;
        while (true) {
            fill(level.begin(), level.end(), -1);
            level[s] = 0;
            q.push(s);
            if (!bfs())
                break;
            fill(ptr.begin(), ptr.end(), 0);
            while (long long pushed = dfs(s, flow_inf)) {
                f += pushed;
            }
        }
        return f;
    }
};
```

4.11 hungarian

```
const int64_t INF64 = int64_t(2e18) + 5;

vector<int> assignment;

template<typename T>
int64_t hungarian(vector<vector<T>> costs) {
    int n = int(costs.size());
    int m = costs.empty() ? 0 : int(costs[0].size());

    if (n > m) {
        vector<vector<T>> new_costs(m, vector<T>(n));

        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                new_costs[j][i] = costs[i][j];

        swap(costs, new_costs);
        swap(n, m);
    }

    vector<int64_t> u(n + 1), v(m + 1);
    vector<int> p(m + 1), way(m + 1);

    for (int i = 1; i <= n; i++) {
        vector<int64_t> min_v(m + 1, INF64);
        vector<bool> used(m + 1, false);
        p[0] = i;
        int j0 = 0;

        do {
            used[j0] = true;
            int i0 = p[j0], j1 = 0;
            int64_t delta = INF64;

            for (int j = 1; j <= m; j++)
                if (!used[j]) {
                    int64_t cur = costs[i0 - 1][j - 1] - u[i0]
                        - v[j];

                    if (cur < min_v[j]) {
                        min_v[j] = cur;
```

```
                way[j] = j0;
            }

            if (min_v[j] < delta) {
                delta = min_v[j];
                j1 = j;
            }
        }

        for (int j = 0; j <= m; j++)
            if (used[j]) {
                u[p[j]] += delta;
                v[j] -= delta;
            } else {
                min_v[j] -= delta;
            }

        j0 = j1;
    } while (p[j0] != 0);

    do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    } while (j0 != 0);
  }

  // Note that p[j] is the row assignment of column j (both
      1-based). If p[j] = 0, the column is unassigned.
  assignment = p;
  return -v[0];
}
```

5 Math

5.1 Binary Gaussian Elimination

```
//Amin Anvari's solution to Shortest XOR Path problem
#include <bits/stdc++.h>
using namespace std;
typedef pair <int,int> pii;
#define L first
#define R second
const int maxn = 1e5, maxl = 31;
bool mark[maxn];
vector <pii> adj[maxn];
vector <int> all;
int n, s, w[maxn], pat[maxn], b[maxn];
void dfs(int v,int par = -1) {
    mark[v] = true;
    for (int i = 0; i < adj[v].size(); i++) {
        int u = adj[v][i].L, e = adj[v][i].R, W = w[e];
        if (!mark[u]) {
            pat[u] = pat[v] ^ W;
            dfs(u, e);
        }
        else if (e != par)
            all.push_back(pat[v] ^ pat[u] ^ W);
    }
}
int get(int x) {
    for (int i = maxl - 1; i >= 0; i--)
        if (x & (1 << i))
            return i;
    return -1;
}
void add(int x) {
    for (int i = 0; i < s; i++)
        if (get(b[i]) != -1 && (x & (1 << get(b[i]))))
            x ^= b[i];
    if (x == 0)
        return;
    for (int i = 0; i < s; i++)
        if (b[i] < x)
            swap(x, b[i]);
    b[s++] = x;
}
int GET(int x) {
    for (int i = 0; i < s; i++)
        if (get(b[i]) != -1 && (x & (1 << get(b[i]))))
            x ^= b[i];
    return x;
}
int main() {
    ios_base::sync_with_stdio(false);
    int m;
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int v, u;
        cin >> v >> u >> w[i];
        v--, u--;
        adj[v].push_back(pii(u, i));
        adj[u].push_back(pii(v, i));
    }
    dfs(0);
    for (int i = 0; i < all.size(); i++)
        add(all[i]);
    cout << GET(pat[n - 1]) << endl;
    return 0;
}
```

5.2 Discrete Logarithm Solver

```
// discrete-logarithm, finding y for equation k = x^y % mod
int discrete_logarithm(int x, int mod, int k) {
  if (mod == 1) return 0;
  int s = 1, g;
  for (int i = 0; i < 64; ++i) {
    if (s == k) return i;
    s = (1ll * s * x) % mod;
  }
  while ((g = gcd(x, mod)) != 1) {
    if (k % g) return -1;
    mod /= g;
  }
  static unordered_map<int, int> M; M.clear();
  int q = int(sqrt(double(euler(mod)))) + 1; // mod-1 is
      also okay
  for (int i = 0, b = 1; i < q; ++i) {
    if (M.find(b) == M.end()) M[b] = i;
    b = (1ll * b * x) % mod;
  }
  int p = fpow(x, q, mod);
  for (int i = 0, b = 1; i <= q; ++i) {
    int v = (1ll * k * inverse(b, mod)) % mod;
    if (M.find(v) != M.end()) {
      int y = i * q + M[v];
      if (y >= 64) return y;
    }
    b = (1ll * b * p) % mod;
  }
  return -1;
}
```

5.3 Euler Totient Function

```
/* Returns the number of positive integers that are
 * relatively prime to n. As efficient as factor().
 * REQUIRES: factor()
 * REQUIRES: sqrt() must work on Int.
 * REQUIRES: the constructor Int::Int( double ).
 **/
int phi( int n ) {
 vector< int > p;
 factor( n, p );
```

```
 for( int i = 0; i < ( int )p.size(); i++ ) {
  if( i && p[i] == p[i - 1] ) continue;
  n /= p[i];
  n *= p[i] - 1;
 }
 return n;
}
```

5.4 Extended GCD

```
template< class Int >
struct Triple
{
 Int d, x, y;
 Triple( Int q, Int w, Int e ) : d( q ), x( w ), y( e ) {}
};

/* Given nonnegative a and b, computes d = gcd( a, b )
 * along with integers x and y, such that d = ax + by
 * and returns the triple (d, x, y).
 * WARNING: needs a small modification to work on
 * negative integers (operator% fails).
 **/

template< class Int >
Triple< Int > egcd( Int a, Int b )
{
 if( !b ) return Triple< Int >( a, Int( 1 ), Int( 0 ) );
 Triple< Int > q = egcd( b, a % b );
 return Triple< Int >( q.d, q.y, q.x - a / b * q.y );
}
```

5.5 Fibonacci Numbers Properties

Let A, B and n be integer numbers.

$$k = A - B \tag{1}$$

$$F_A F_B = F_{k+1} F_A^2 + F_k F_A F_{A-1} \tag{2}$$

$$\sum_{i=0}^{n} F_i^2 = F_{n+1} F_n \tag{3}$$

$ev(n)$ = returns 1 if n is even.

$$\sum_{i=0}^{n} F_i F_{i+1} = F_{n+1}^2 - ev(n) \tag{4}$$

$$\sum_{i=0}^{n} F_i F_{i-1} = \sum_{i=0}^{n-1} F_i F_{i+1} \tag{5}$$

5.6 Linear Diophantine Equation Solver

```
/* Solves integer equations of the form ax + by = c
 * for integers x and y. Returns a triple containing
 * the answer (in .x and .y) and a flag (in .d).
 * If the returned flag is zero, then there are no
 * solutions. Otherwise, there is an infinite number
 * of solutions of the form
 * x = t.x + k * b / t.d,
 * y = t.y - k * a / t.d;
 * where t is the returned triple, and k is any
 * integer.
 * REQUIRES: struct Triple, egcd
 **/
template< class Int >
Triple< Int > ldioph( Int a, Int b, Int c ) {
 Triple< Int > t = egcd( a, b );
 if( c % t.d ) return Triple< Int >( 0, 0, 0 );
 t.x *= c / t.d; t.y *= c / t.d;
 return t;
}
```

5.7 Maximum XOR (SGU 275)

```
int n;
long long x, ans;
vector<long long> st;
int main() {
 cin >> n;
 for (int i = 0; i < n; i++) {
  cin >> x;
  st.push_back(x);
 }
 for (int k = 0; k < n; k++)
  for (int i = 0; i < st.size(); i++)
   for (int j = i + 1; j < st.size(); j++)
    if (__builtin_clzll(st[j]) == __builtin_clzll(st[i]))
     st[j] ^= st[i];
 sort(st.begin(), st.end());
 reverse(st.begin(), st.end());
 for (auto e: st)
  ans = max(ans, ans ^ e);
 cout << ans << endl;
 return 0;
}
```

5.8 Modular Linear Equation Solver

```
/* Given a, b and n, solves the equation ax = b (mod n)
 * for x. Returns the vector of solutions, all smaller
 * than n and sorted in increasing order. The vector is
 * empty if there are no solutions.
 * REQUIRES: struct Triple, egcd
 **/
template< class Int >
vector< Int > msolve( Int a, Int b, Int n ) {
 if( n < 0 ) n = -n;
 Triple< Int > t = egcd( a, n );
 vector< Int > r;
 if( b % t.d ) return r;
 Int x = ( b / t.d * t.x ) % n;
 if( x < Int( 0 ) ) x += n;
 for( Int i = 0; i < t.d; i++ )
 r.push_back( ( x + i * n / t.d ) % n );
 return r;
}
```

5.9 Number of Divisors

```
/* Returns the number of positive divisors of n.
 * Complexity: about O(sqrt(n)).
 * REQUIRES: factor()
 * REQUIRES: sqrt() must work on Int.
 * REQUIRES: the constructor Int::Int( double ).
 **/
template< class Int >
Int divisors( Int n ) {
 vector< Int > f;
 factor( n, f );
 int k = f.size();
 vector< Int > table( k + 1, Int( 0 ) );
 table[k] = Int( 1 );

 for( int i = k - 1; i >= 0; i-- ) {
  table[i] = table[i + 1];
  for( int j = i + 1; ; j++ )
```

```
  if( j == k || f[j] != f[i] )
  { table[i] += table[j]; break; }
 }

 return table[0];
}
```

5.10 Prime Factors in n Factorial

```
using namespace std;
typedef long long ll;
typedef pair<ll ,int> pii;
vector <pii> v;
//////////// bozorgtarin i b shekli k N!%k^i==0
void fact(ll n) {
 ll x = 2;
 while (x * x <= n)
 {
  ll num = 0;
  while (n % x == 0) {
   num++;
   n /= x;
  }
  if (num) v.push_back(MP(x, num));
  x++;
  if (n == 1ll) break;
 }
 if(n > 1) v.push_back(MP(n, 1));
}

ll getfact(ll n) {
 ll ret = n;
 Rep(i, v.size()) {
  ll k = v[i].first;
  ll cnt = 0;
  ll t = n;
  while (k <= n) {
  cnt += n / k;
  n /= k;
  }
  n = t;
  ret = min(ret, cnt / v[i].second);
 }
 return ret;
}

int main() {
 int tc;
 ll n, k;
 cin >> tc;
 while (tc--) {
  v.clear();
  cin >> n >> k;
  fact(k);
  cout << getfact(n) << endl;
 }
 return 0;
}
```

5.11 Reduced Row Echelon Form

```
// Reduced row echelon form via Gauss-Jordan elimination
// with partial pivoting. This can be used for computing
// the rank of a matrix.
//
// Running time: O(n^3)
//
// INPUT:   a[][] = an nxm matrix
//
// OUTPUT:  rref[][] = an nxm matrix (stored in a[][])
//          returns rank of a[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPSILON = 1e-10;

typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

int rref(VVT &a) {
  int n = a.size();
  int m = a[0].size();
  int r = 0;
  for (int c = 0; c < m && r < n; c++) {
    int j = r;
    for (int i = r + 1; i < n; i++)
      if (fabs(a[i][c]) > fabs(a[j][c])) j = i;
    if (fabs(a[j][c]) < EPSILON) continue;
    swap(a[j], a[r]);

    T s = 1.0 / a[r][c];
    for (int j = 0; j < m; j++) a[r][j] *= s;
    for (int i = 0; i < n; i++) if (i != r) {
      T t = a[i][c];
      for (int j = 0; j < m; j++) a[i][j] -= t * a[r][j];
    }
    r++;
  }
  return r;
}

int main() {
  const int n = 5, m = 4;
  double A[n][m] = {
    {16,  2,  3, 13},
    { 5, 11, 10,  8},
    { 9,  7,  6, 12},
    { 4, 14, 15,  1},
    {13, 21, 21, 13}};
  VVT a(n);
  for (int i = 0; i < n; i++)
    a[i] = VT(A[i], A[i] + m);

  int rank = rref(a);

  // expected: 3
  cout << "Rank: " << rank << endl;

  // expected: 1 0 0 1
  //           0 1 0 3
  //           0 0 1 -3
  //           0 0 0 3.10862e-15
  //           0 0 0 2.22045e-15
  cout << "rref: " << endl;
  for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 4; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }
}
```

5.12 Solving Recursive Functions

```
//From "You Know Izad?" team cheat sheet
/*
a[i] = b[i] (for i <= k)
a[i] = c[1]*a[i-1] + c[2]a[i-2] + ... + c[k]a[i-k] (for i >
    k)
Given:
b[1], b[2], ..., b[k]
c[1], c[2], ..., c[k]
a[N]=?
```

```
*/
typedef vector<vector<ll> > matrix;
int K;
matrix mul(matrix A, matrix B){
    matrix C(K+1, vector<ll>(K+1));
    REP(i, K) REP(j, K) REP(k, K)
        C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % INF32;
    return C;
}
matrix pow(matrix A, ll p){
    if (p == 1) return A;
    if (p % 2) return mul(A, pow(A, p-1));
    matrix X = pow(A, p/2);
    return mul(X, X);
}
ll solve() {
    // base (initial) values
    vector<ll> F1(K+1);
    REP (i, K)
        cin >> F1[i];
    matrix T(K+1, vector<ll>(K+1));
    REP(i, K) {
        REP(j, K) {
            if(j == i + 1) T[i][j] = 1;
            else if(i == K) cin >> T[i][K - j + 1]; //
                multipliers
            else T[i][j] = 0;
        }
    }
    ll N;
    cin >> N;
    if (N == 1) return 1;
    T = pow(T, N-1);
    ll res = 0;
    REP(i, K)
        res = (res + T[1][i] * F1[i]) % INF32; // Mod Value
    return res;
}
int main() {
    cin >> K;
    cout << solve() << endl;
}
```

6 Others

6.1 FFT and Multiplication

```
#define base complex<double>
void fft (vector<base> & a, bool invert){
    if (L(a) == 1) return;
    int n = L(a);
    vector <base> a0(n / 2), a1(n / 2);
    for (int i = 0, j = 0; i < n; i += 2, ++j){
        a0[j] = a[i];
        a1[j] = a[i + 1];
    }
    fft (a0, invert);
    fft (a1, invert);
    double ang = 2 * PI / n * (invert ? -1 : 1);
    base w(1), wn(cos(ang), sin(ang));
    fore(i, 0, n / 2) {
        a[i] = a0[i] + w * a1[i];
        3
        a[i + n / 2] = a0[i] - w * a1[i];
        if (invert)
            a[i] /= 2, a[i + n / 2] /= 2;
        w *= wn;
    }
}
void multiply (const vector<int> &a, const vector<int> & b,
     vector<int> &res){
    vector <base> fa(all(a)), fb(all(b));
    size_t n = 1;
    while (n < max(L(a), (L(b)))) n <<= 1;
    n <<= 1;
    fa.resize(n), fb.resize(n);
    fft(fa, false), fft(fb, false);
    fore(i, 0, n)
    fa[i] *= fb[i];
    fft (fa, true);
    res.resize (n);
    fore(i, 0, n)
    res[i] = int (fa[i].real() + 0.5);
}
```

6.2 Fermat's Theory

if a is a natural number and p is a prime number then (a^p)

6.3 Miller-Rabin primality test

```
bool miillerTest(int d, int n)
{
    // Pick a random number in [2..n-2]
    // Corner cases make sure that n > 4
    int a = 2 + rand() % (n - 4);

    // Compute a^d % n
    int x = power(a, d, n);

    if (x == 1 || x == n-1)
       return true;

    // Keep squaring x while one of the following doesn't
    // happen
    // (i)  d does not reach n-1
    // (ii) (x^2) % n is not 1
    // (iii) (x^2) % n is not n-1
    while (d != n-1)
    {
        x = (x * x) % n;
        d *= 2;

        if (x == 1)     return false;
        if (x == n-1)   return true;
    }

    // Return composite
    return false;
}

// k is an input parameter that determines
// accuracy level. Higher value of k indicates more accuracy
    .
bool isPrime(int n, int k)
{
    // Corner cases
    if (n <= 1 || n == 4) return false;
    if (n <= 3) return true;

    // Find r such that n = 2^d * r + 1 for some r >= 1
    int d = n - 1;
    while (d % 2 == 0)
        d /= 2;

    // Iterate given nber of 'k' times
    for (int i = 0; i < k; i++)
         if (!miillerTest(d, n))
              return false;

    return true;
}
```

6.4 Uniform Random Number Generator

```
using namespace std;
//seed:
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<> dis(0, n - 1);
//generate:
int r = dis(gen);
```

6.5 faster FFT

```
const double PI = acos(-1);
#define base complex<double>
int lg_n;
int rev [maxn * 20];
vector<base> polies[maxn];
int reverse(int num ,int lll) {
    return rev[num];
}

void fft(vector<base> & a, bool invert) {
    int n = a.size();


    for (int i = 0; i < n; i++) {
        if (i < reverse(i, lg_n))
            swap(a[i], a[reverse(i, lg_n)]);
    }

    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 (base & x : a)
            x /= n;
    }
}

void multiply (int u , int v){
 int n = 1;
   while (n < max(L(a) , L(b))) {
    n <<= 1;
   }
   n <<= 1;
   lg_n = 0;
   while ((1 << lg_n) < n)
       lg_n++;

   for (int i=0; i<n; ++i) {
  rev[i] = 0;
  for (int j=0; j<lg_n; ++j)
   if (i & (1<<j))
    rev[i] |= 1<<(lg_n-1-j);
 }
 a.resize(n);
 b.resize(n);
   fft(a, false), fft(b, false);
   fore(i, 0, n - 1){
    a[i] *= b[i];
   }
   fft (a, true);
   res.resize (n);

   fore(i, 0, n - 1) {
    res[i] = round(a[i].real());
   }
}
```

7 String

7.1 Aho Corasick

```
#include <bits/stdc++.h>
#define FOR(i, n) for (int i = 0; i < (n); ++i)
#define REP(i, n) for (int i = 1; i <= (n); ++i)
using namespace std;

struct AC_trie {
  int N, P;
  vector<map<char, int>> next; // trie
  vector<int> link, out_link;
  vector<vector<int>> out;
  AC_trie(): N(0), P(0) { node(); }
  int node() {
    next.emplace_back(); // trie
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++;
  }
  int add_pattern(const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][c]) next[u][c] = node();
      u = next[u][c];
    }
    out[u].push_back(P);
    return P++;
  }
  void compute() {
    queue<int> q;
    for (q.push(0); !q.empty(); ) {
      int u = q.front(); q.pop();
      // trie:
      for (auto e : next[u]) {
        int v = e.second;
        link[v] = u ? advance(link[u], e.first) : 0;
        out_link[v] = out[link[v]].empty() ? out_link[link[v
            ]] : link[v];
        q.push(e.second);
      }
    }
  }
  int advance(int u, char c) {
    // trie:
    while (u && next[u].find(c) == next[u].end())
      u = link[u];
    if (next[u].find(c) != next[u].end())
      u = next[u][c];
    return u;
  }
  void match(const string S) {
    int u = 0;
    for (auto c : S) {
      u = advance(u, c);
      for (int v = u; v; v = out_link[v])
        for (auto p : out[v])
          cout << "match " << p << endl;
    }
  }
};
struct AC_automaton {
  int N, P;
  vector<vector<int>> next; // automaton
```

```
vector<int> link, out_link;
vector<vector<int>> out;
AC_automaton(): N(0), P(0) { node(); }
int node() {
  next.emplace_back(26, 0); // automaton
  link.emplace_back();
  out_link.emplace_back();
  out.emplace_back();
  return N++;
}
int add_pattern(const string T) {
  int u = 0;
  for (auto c : T) {
    if (!next[u][c - 'a']) next[u][c - 'a'] = node();
    u = next[u][c - 'a'];
  }
  out[u].push_back(P);
  return P++;
}
void compute() {
  queue<int> q;
  for (q.push(0); !q.empty(); ) {
    int u = q.front(); q.pop();
    // automaton:
    for (int c = 0; c < 26; ++c) {
      int v = next[u][c];
      if (!v) next[u][c] = next[link[u]][c];
      else {
        link[v] = u ? next[link[u]][c] : 0;
        out_link[v] = out[link[v]].empty() ? out_link[link[
            v]] : link[v];
        q.push(v);
      }
    }
  }
}
int advance(int u, char c) {
  // automaton:
  while (u && !next[u][c - 'a']) u = link[u];
  u = next[u][c - 'a'];
  return u;
}
void match(const string S) {
  int u = 0;
  for (auto c : S) {
    u = advance(u, c);
    for (int v = u; v; v = out_link[v])
      for (auto p : out[v])
        cout << "match " << p << endl;
  }
  }
};
int main() {
  int P;
  string T;
  cin >> P;

  AC_trie match1;
  AC_automaton match2;
  REP (i, P) {
    cin >> T;
    match1.add_pattern(T); match2.add_pattern(T);
  }
  match1.compute();
  match2.compute();
  cin >> T;
  match1.match(T);
  match2.match(T);
  return 0;
}
```

7.2 Aho corasick 1

```
struct Node
{
    char c;
    int parent;
    int isWord;
    int suffLink;
    vi children;
    int len;
    Node(){
        parent = -1;
        isWord = false;
        suffLink = -1;
        children.clear();
        len = 0;
    }
};
struct Aho
{
    vector<Node> nodes;

    Aho(){
        nodes.pub(Node());
        nodes[0].suffLink=0;
    }
    void addString(string s){
        int cur = 0;
        fore(i, 0 , s.size()-1){
            int nxt = -1;
            if (cur < nodes.size())
                fore(j , 0 , nodes[cur].children.size()-1){
                    if (nodes[nodes[cur].children[j]].c == s[i
                        ])
                    {
                        nxt = nodes[cur].children[j];
                        break;
                    }
                }
            if (~nxt)
            {
                cur = nxt;
                continue;
            }
            nodes[cur].children.pub(nodes.size());
            nodes.pub(Node());
            nodes[nodes.size()-1].parent = cur;
            nodes[nodes.size()-1].c = s[i];

            cur = nodes.size()-1;
        }
        nodes[cur].isWord = true;
        nodes[cur].len = s.size();
    }

    int calc(int cur){
        if (nodes[cur].suffLink == -1)
        {
            if (nodes[cur].parent == 0) return 0;
            return nodes[cur].suffLink = trans(calc(nodes[cur
                ].parent) , nodes[cur].c);
        }
        return nodes[cur].suffLink;
    }
    int res = inf;
    int pos;
    int trans(int cur , char c ){
        if (nodes[cur].isWord){
            res = min(res , pos - nodes[cur].len + 1);
        }
        fore(i,0 , nodes[cur].children.size()-1){
            if (nodes[nodes[cur].children[i]].c == c)
                return nodes[cur].children[i];
        }
        if (cur == 0) return 0;
        return trans(calc(cur) , c);
    }
    int find(string s){
```

```
        int cur = 0;
        fore(i , 0 , s.size()-1){
            pos = i;
            cur = trans(cur , s[i]);
            if (nodes[cur].isWord){
                res = min(res , i - nodes[cur].len +2);
            }
            calc(cur);
        }
        return res;
    }
};
```

7.3 KMP

```
//From "You Know Izad?" team cheat sheet
int fail[100005];
void build(const string &key){
    fail[0] = 0;
    fail[1] = 0;
    fore(i, 2, L(key)) {
        int j = fail[i - 1];
        while (true) {
            if (key[j] == key[i - 1]) {
                fail[i] = j + 1;
                break;
            }
            else if (j == 0) break;
            j = fail[j];
        }
    }
}
int KMP(const string &text, const string &key) {
    build(key);
    int i = 0, j = 0;
    while (true) {
        if (j == L(text)) return -1;
        if (text[j] == key[i]) {
            i++;
            j++;
            if (i == L(key)) return j - i;
        }
        else if (i > 0) i = fail[i];
        else j++;
    }
}
```

7.4 SuffixTree

```
#define pci pair< char, int >
#define NV N[v]
string s;
struct node {
    int p, b, e, link;
    /*map< char, int > children;*/
    vector< pci > children;
    node( int _p, int _b, int _e ) { p = _p, b = _b, e = _e,
         link = -1; }
    void addChild( pci a ) {
        children.push_back( a );
    }
    void changeChild( pci a ) {
 //children[a.first] = a.second;
        for( int i = 0; i < children.size(); i++ ) {
            if( children[i].first == a.first ) {
                children[i].second = a.second;
                return;
            }
        }
    }
    int length() { return e - b + 1; }
    bool gotoNext( char c, int &nv, int &nd) {
        if( nd < e - b ) {
            if( s[b + nd + 1] == c ) {
                nd++;
                return true;
            }
        } else {
            for( int i = 0; i < children.size(); i++ ) {
                if( children[i].first == c ) {
                    nv = children[i].second, nd = 0;
                    return true;
                }
            }
        }
        return false;
    }
};
vector< node > N;
void add2Tree( ) {
    N.clear();
    N.push_back( node( -1, -1, -1 ) );
    N[0].link = 0;
    int j = 0, pp = -1, v = 0, d = 0;
    for( int i = 0; i < s.length(); i++ ) {
        pp = -1;
        for( ; j <= i; j++ ) {
            if( NV.gotoNext( s[i], v, d ) ) {
                if( pp != -1 ) N[pp].link = NV.p;
                break;
            } else {
                int id = N.size();
                if( d < NV.e - NV.b ) {
                    if( pp != -1 ) N[pp].link = id;
                    N.push_back( node( NV.p, NV.b, NV.b + d )
                        );
                    N[NV.p].changeChild( pci( s[NV.b], id ) );
                    NV.b += d + 1;
                    NV.p = pp = id;
                    N[id].addChild( pci( s[NV.b], v ) );
                    int len = N[id].p ? d + 1 : d;
                    v = N[N[id].p].link;
                    d = NV.length() - 1;
                    while( len ) {
                        int temp = v;
                        N[temp].gotoNext( s[i - len] , v, d );
                        int l = NV.length();
                        if( len <= l ) {
                            d = len - 1;
                            break;
                        }
                        d = l - 1;
                        len -= l;
                    }
                    id++;
                } else {
                    if( pp != -1 ) N[pp].link = v;
                    pp = v;
                    v = NV.link;
                    d = NV.length() - 1;
                }
                N[pp].addChild( pci( s[i], id ) );
                N.push_back( node( pp, i, s.length() - 1 ) );
            }
        }
    }
}
```

7.5 Trietree

```
int ind(char c)
{
 return c - 'a';
}
int n, m;
int node[maxn][26];
```

```
int cnt = 1;
vector<int> edges[maxn];
int val [maxn];
string s;
vector<int> v;
void insert(){
 int nd = 0;
 int idx = 0;
 int last = -1;
 while(~node[nd][ind(s[idx])] && idx<s.size()){
  last = nd;
  nd = node[nd][ind(s[idx])];
  idx++;
 }
 v.clear();
 while(idx < s.size())
 {
  v.push_back(nd);
  node[nd][ind(s[idx])] = cnt;
  idx++;
  nd = cnt++;
 }
 v.push_back(nd);
 if (v.size() < 2)
  return;
 fore(i,0,v.size()-2){
  edges[v[i]].push_back(v[i+1]);
  edges[v[i+1]].push_back(v[i]);
 }
 if(v.size()>2)
  edges[v[1]].push_back(nd);
}
```

7.6 rope

```
#include <bits/stdc++.h>
#include <ext/rope> //header with rope
using namespace std;
using namespace __gnu_cxx; //namespace with rope and some
     additional stuff
int main()
{

    rope <int> v; //use as usual STL container
    int n, m;
    cin >> n >> m;
    for(int i = 1; i <= n; ++i)
        v.push_back(i); //initialization
    string p ;
    int idx;
    cin>>p>>idx;
    fore(i,1,p.size()){
        s.insert(i + idx -1 , p[i-1]);
    }
    int l, r;
    for(int i = 0; i < m; ++i)
    {
        cin >> l >> r;
        --l, --r;
        rope <int> cur = v.substr(l, r - l + 1);
        v.erase(l, r - l + 1);
        v.insert(v.mutable_begin(), cur);

    }
    for(rope <int>::iterator it = v.mutable_begin(); it != v.
         mutable_end(); ++it)
        cout << *it << " ";
    return 0;
}
```

8 Tips, Tricks and Theorems

8.1 Burnside's lemma

In the following, let G be a finite group that acts on a set X. For each g in G let Xg denote the set of elements in X that are fixed by g (also said to be left invariant by g), i.e. $X^g = \{x \in X | g.x = x\}$.

Burnside's lemma asserts the following formula for the number of orbits, denoted $|X/G|$:

$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$

Thus the number of orbits (a natural number or +infinity) is equal to the average number of points fixed by an element of G (which is also a natural number or infinity). If G is infinite, the division by G may not be well-defined; in this case the following statement in cardinal arithmetic holds:

$|G||X/G| = \sum_{g \in G} |X^g|.$

Example application:

The number of rotationally distinct colourings of the faces of a cube using three colours can be determined from this formula as follows.

Let X be the set of 3*3*3*3*3*3 possible face colour combinations that can be applied to a cube in one particular orientation, and let the rotation group G of the cube act on X in the natural manner. Then two elements of X belong to the same orbit precisely when one is simply a rotation of the other. The number of rotationally distinct colourings is thus the same as the number of orbits and can be found by counting the sizes of the fixed sets for the 24 elements of G.

- one identity element which leaves all 3*3*3*3*3*3 elements of X unchanged

- six 90-degree face rotations, each of which leaves 3*3*3 of the elements of X unchanged

- three 180-degree face rotations, each of which leaves 3*3*3*3 of the elements of X unchanged

- eight 120-degree vertex rotations, each of which leaves 3*3 of the elements of X unchanged

- six 180-degree edge rotations, each of which leaves 3*3*3 of the elements of X unchanged

The average fix size is thus

$\frac{1}{24}\left(3^6 + 6 \cdot 3^3 + 3 \cdot 3^4 + 8 \cdot 3^2 + 6 \cdot 3^3\right) = 57.$

Hence there are 57 rotationally distinct colourings of the faces of a cube in three colours. In general, the number of rotationally distinct colorings of the faces of a cube in n colors is given by

$\frac{1}{24}\left(n^6 + 3n^4 + 12n^3 + 8n^2\right).$

8.2 C++ Ordered Set

```
typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;

ordered_set X;
X.insert(1);
X.insert(2);
```

```
X.insert(4);
X.insert(8);
X.insert(16);

cout<<*X.find_by_order(1)<<endl; // 2
cout<<*X.find_by_order(2)<<endl; // 4
cout<<*X.find_by_order(4)<<endl; // 16
cout<<(end(X)==X.find_by_order(6))<<endl; // true

cout<<X.order_of_key(-5)<<endl; // 0
cout<<X.order_of_key(1)<<endl; // 0
cout<<X.order_of_key(3)<<endl; // 2
cout<<X.order_of_key(4)<<endl; // 2
cout<<X.order_of_key(400)<<endl; // 5
```

8.3 C++ Tricks

```
cout << fixed << setprecision(7) << M_PI << endl; //
    3.1415927
cout << scientific << M_PI << endl; // 3.1415927e+000
int x=15, y=12094;
cout << setbase(10) << x << " " << y << endl; // 15 12094
cout << setbase(8) << x << " " << y << endl; // 17 27476
cout << setbase(16) << x << " " << y << endl; // f 2f3e
x=5; y=9;
cout<<setfill('0')<<setw(2)<<x<< ":" << setw(2) << y << endl
    ; // 05:09
printf ("%10d\n", 111); // 111
printf ("%010d\n", 111); //0000000111
printf ("%d %x %X %o\n", 200, 200, 200, 200); //200 c8 C8
    310
printf ("%010.2f %e %E\n", 1213.1416, 3.1416, 3.1416); //
    0001213.14 3.141600e+00 3.141600E+00
printf ("%*.*d\n",10, 5, 111); // 00111
printf ("%-*.*d\n",10, 5, 111); //00111
printf ("%+*.*d\n",10, 5, 111); // +00111
char in[20]; int d;
scanf ("%s %*s %d",in,&d); //<- it's number 5
printf ("%s %d \n", in,d); //it's 5
```

8.4 Contest Tips

```
READ THE STATEMENT AGAIN. TELL YOUR TEAMMATE IF
 NECESSARY
Double check spell of literals
Graph: Multiple components, Multiple edges, Loops
Geometry: Be careful about +pi,-pi
Initialization: Use memset/clear(). Dont expect global
 variables to be zero. Care about multiple tests
Precision and Range: Use long long if necessary. Use
 BigInteger/BigDecimal
Derive recursive formulas that use sum instead of
 multiplication to avoid overflow.
Small cases (n=0,1,negative)
0-based <=> 1-based
Division by zero. Integer division a/(double)b
Stack overflow (DFS on 1e5)
Infinite loop?
array bound check. maxn or x*maxn
Dont use .size()-1 !
    (int)-3 < (unsigned int) 2 is false!
Check copy-pasted codes!
Be careful about -0.0
Remove debug info!
Output format: Spaces at the end of line. Blank lines.
 View the output in VIM if necessary
Add eps to double before getting floor or round
Convex Hull: Check if points are collinear
Geometry: Distance may not overflow, but its square may
 does
```

8.5 Dilworth Theorem

```
Let S be a finite partially ordered set. The size of a
     maximal antichain equals the size of a minimal chain
     cover of S. This is called the Dilworths theorem.

The width of a finite partially ordered set S is the maximum
      size of an antichain in S. In other words, the width
     of a finite partially ordered set S is the minimum
     number of chains needed to cover S, i.e. the minimum
     number of chains such that any element of S is in at
     least one of the chains.

Definition of chain : A chain in a partially ordered set is
     a subset of elements which are all comparable to each
     other.
Definition of antichain : An antichain is a subset of
     elements, no two of which are comparable to each other.
```

8.6 Gallai Theorem

```
a(G) := max{|C| | C is a stable set},
b(G) := min{|W| | W is a vertex cover},
c(G) := max{|M| | M is a matching},
d(G) := min{|F| | F is an edge cover}.
 Gallais  theorem: If G = (V, E) is a graph without isolated
      vertices, then
a(G) + b(G) = |V| = c(G) + d(G).
```

8.7 Konig Theorem

```
 Knig  theorem can be proven in a way that provides
     additional useful information beyond just its truth:
     the proof provides a way of constructing a minimum
     vertex cover from a maximum matching. Let {G=(V,E)} be
     a bipartite graph, and let the vertex set { V} be
     partitioned into left set { L} and right set { R}.
     Suppose that { M} is a maximum matching for { G}. No
     vertex in a vertex cover can cover more than one edge
     of { M} (because the edge half-overlap would prevent {
     M} from being a matching in the first place), so if a
     vertex cover with { |M|} vertices can be constructed,
     it must be a minimum cover.
To construct such a cover, let { U} be the set of unmatched
     vertices in { L} (possibly empty), and let { Z} be the
     set of vertices that are either in { U} or are
     connected to { U} by alternating paths (paths that
     alternate between edges that are in the matching and
     edges that are not in the matching). Let
{ K=(L - Z) Union (R Intersect Z).}
Every edge { e} in { E} either belongs to an alternating
     path (and has a right endpoint in { K}), or it has a
     left endpoint in { K}. For, if { e} is matched but not
     in an alternating path, then its left endpoint cannot
     be in an alternating path (for such a path could only
     end at { e}) and thus belongs to { L- Z}. Alternatively
     , if { e} is unmatched but not in an alternating path,
     then its left endpoint cannot be in an alternating path
     , for such a path could be extended by adding { e} to
     it. Thus, { K} forms a vertex cover.
Additionally, every vertex in { K} is an endpoint of a
     matched edge. For, every vertex in { L - Z} is matched
     because Z is a superset of U, the set of unmatched left
      vertices. And every vertex in { R intersect Z} must
     also be matched, for if there existed an alternating
     path to an unmatched vertex then changing the matching
     by removing the matched edges from this path and adding
      the unmatched edges in their place would increase the
     size of the matching. However, no matched edge can have
      both of its endpoints in { K}. Thus, { K} is a vertex
     cover of cardinality equal to { M}, and must be a
     minimum vertex cover.
```

8.8 Lucas Theorem

For non-negative integers m and n and a prime p, the following congruence relation holds: :

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod p,$$

where :

$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0,$$

and :

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$$

are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$ if $m \leq n$.

8.9 Minimum Path Cover in DAG

Given a directed acyclic graph $G = (V, E)$, we are to find the minimum number of vertex-disjoint paths to cover each vertex in V.

We can construct a bipartite graph $G' = (Vout \cup Vin, E')$ from G, where :

$$Vout = \{v \in V : v \ has \ positive \ out - degree\}$$

$$Vin = \{v \in V : v \ has \ positive \ in - degree\}$$

$$E' = \{(u, v) \in Vout \times Vin : (u, v) \in E\}$$

Then it can be shown, via König's theorem, that G' has a matching of size m if and only if there exists $n - m$ vertex-disjoint paths that cover each vertex in G, where n is the number of vertices in G and m is the maximum cardinality bipartite mathching in G'.

Therefore, the problem can be solved by finding the maximum cardinality matching in G' instead.

NOTE: If the paths are note necesarily disjoints, find the transitive closure and solve the problem for disjoint paths.

8.10 Planar Graph (Euler)

Euler's formula states that if a finite, connected, planar graph is drawn in the plane without any edge intersections, and v is the number of vertices, e is the number of edges and f is the number of faces (regions bounded by edges, including the outer, infinitely large region), then:

$$f + v = e + 2$$

It can be extended to non connected planar graphs with c connected components:

$$f + v = e + c + 1$$

8.11 Triangles

Let a, b, c be length of the three sides of a triangle.

$$p = (a + b + c) * 0.5$$

The inradius is defined by:

$$iR = \sqrt{\frac{(p-a)(p-b)(p-c)}{p}}$$

The radius of its circumcircle is given by the formula:

$$cR = \frac{abc}{\sqrt{(a+b+c)(a+b-c)(a+c-b)(b+c-a)}}$$

8.12 Uniform Random Number Generator

```cpp
using namespace std;
//seed:
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<> dis(0, n - 1);
//generate:
int r = dis(gen);
```