GEMP - UECE - ICPC Library

Contents

1 Geometry 2
- 1.1 Distance Between Nearest Pair Of Points . . . 2
- 1.2 Convex Hull Trick . . . 2
- 1.3 Check If A Point Is Inside A Convex Polygon . . . 3
- 1.4 Geometry Stan . . . 4
- 1.5 Dynamic Convex Hull Trick . . . 6
- 1.6 Build Two Lines That Go Through All Points Of A Set . . . 7
- 1.7 Graham Scan . . . 8
- 1.8 Radial Sort . . . 8
- 1.9 Enclosing Circle R2 . . . 9
- 1.10 Enclosing Circle R3 . . . 9
- 1.11 Andrew Algorithm Convex Hull . . . 9
- 1.12 Segment Intersection . . . 10
- 1.13 Maximum Dot Product . . . 10

2 String 11
- 2.1 Trie Static . . . 11
- 2.2 Suffix Array . . . 12
- 2.3 LIS LDS . . . 13
- 2.4 SA . . . 14
- 2.5 Longest Common Substring . . . 16
- 2.6 Dynamic Trie . . . 16
- 2.7 Z Function . . . 17
- 2.8 Suffix Array And Applications . . . 17
- 2.9 Aho Corasick . . . 19
- 2.10 Manacher . . . 21
- 2.11 Trie With Vector . . . 21
- 2.12 KMP . . . 22
- 2.13 Trie . . . 22

3 Miscellaneous 23
- 3.1 Maximum Subarray Xor . . . 23
- 3.2 Rectangles Union Area . . . 24
- 3.3 Count Divisors . . . 25
- 3.4 Counting Different Elements In A Path With Mo . . . 26
- 3.5 Gen Random Tree . . . 27
- 3.6 Index Compression . . . 27
- 3.7 Odd Rectangles Area . . . 27
- 3.8 Karp Rabin . . . 29
- 3.9 Quick Sort And Select . . . 30
- 3.10 Histogram . . . 30
- 3.11 Divide Conquer Optimization . . . 30
- 3.12 Inclusion Exclusion . . . 31
- 3.13 Custom Hash Function Unordered Map Or Set . . . 32
- 3.14 Square Root Decomposition . . . 32
- 3.15 Big Num Product . . . 33
- 3.16 Fence Problem With Max Flow . . . 33
- 3.17 Mo . . . 34
- 3.18 Knuth Optimization . . . 37
- 3.19 Count Sort . . . 37
- 3.20 Lontest Substring That Is A Correct Bracket Sequence . . . 38
- 3.21 Knapsack With Backtraking . . . 38
- 3.22 Small To Large . . . 39
- 3.23 FastIO . . . 41
- 3.24 String Matching Hash Sqrtdecomp . . . 42

4 Dynamic Programming 43
- 4.1 Traveling Salesman Problem Bottom Up Dp . . . 43
- 4.2 Knapsack Zero One Without Value . . . 43
- 4.3 KnapsackErrichto . . . 44
- 4.4 Edit Distance With DP . . . 44
- 4.5 Digit DP Sum Of Digits In Range . . . 44
- 4.6 Coin Problem Topdown Dp . . . 45
- 4.7 Kadane 3D . . . 45
- 4.8 KnapsackWithCopies . . . 45
- 4.9 Knapsack0-kSemValor . . . 46
- 4.10 KnapsackwithPDtopdown . . . 46
- 4.11 Subset Sum . . . 47
- 4.12 Kadane 2D . . . 47
- 4.13 Traveling Salesman Problem Topdown Dp . . . 47
- 4.14 Longest Increasing Subsequence . . . 48
- 4.15 Longest Common Subsequece And Edit Distance . . . 48
- 4.16 Knapsack With Copies SqrtN Memory . . . 48

5 Math 49
- 5.1 Pollard Rho . . . 49
- 5.2 Mod Gaussian Elimination . . . 50
- 5.3 Catalan Numbers . . . 51
- 5.4 Matrix Exponentiation . . . 51
- 5.5 Baby Step Giant Step . . . 52
- 5.6 Gaussian Elimination For Max Subset Xor . . . 52
- 5.7 Counting Number Of Times That A Digit Appears Until N . . . 53
- 5.8 Chinese Remainder Theorem . . . 53
- 5.9 Modular Arithmetic . . . 53
- 5.10 Karatsuba . . . 54
- 5.11 Fast Fourier Transform . . . 54
- 5.12 Mulmod Trick . . . 59
- 5.13 Miller Rabin . . . 59
- 5.14 Mobius . . . 60
- 5.15 Conversion Base . . . 60

6 Useful Scripts 61
- 6.1 Stress.sh . . . 61

7 Graph 61
- 7.1 Boruvka MST . . . 61
- 7.2 2SAT . . . 62
- 7.3 Longest And Shortest Path In DAG . . . 63
- 7.4 Knapsack Dijkstra . . . 63
- 7.5 Eulirian Path . . . 64
- 7.6 Tree Isomorfism . . . 65
- 7.7 K Short Paths . . . 66
- 7.8 Hopcroft Karp . . . 66
- 7.9 Diameter And Center Of A Tree . . . 67
- 7.10 Dijkstra . . . 67
- 7.11 BFS Zero One . . . 68
- 7.12 MPC MinimumPathCover . . . 68
- 7.13 Binary Lifting . . . 69
- 7.14 Lca With Square Root Decomposition . . . 70
- 7.15 Fully Dynamic Connectivity Check If Two Vertices Are In The Same Component . . . 71
- 7.16 Floyd Sucessor Graph . . . 72
- 7.17 MCE MinimumEdgeCover . . . 73
- 7.18 Maximum Clique . . . 73
- 7.19 MVC MinimumVertexCover . . . 74
- 7.20 Lca With Tree Linearization And Sparse Table . . . 74
- 7.21 Erdos Gallai Theorem . . . 75
- 7.22 Kuhn MCBM . . . 75
- 7.23 Lca With Tree Linearization And Segment Tree . . . 76
- 7.24 Min Cost Max Flow . . . 77
- 7.25 Fully Dynamic Connectivity Count Conected Components . . . 77
- 7.26 Dinic . . . 79
- 7.27 Prim . . . 80
- 7.28 Ford Fulkerson . . . 80
- 7.29 Center Of A Tree . . . 81

8 Data Structures 81
- 8.1 Two Stacks Trick . . . 81
- 8.2 Segment Tree Tree 2D . . . 82
- 8.3 Merge Sort Tree With Set . . . 83
- 8.4 Segment Tree With Lazy Propagation . . . 83
- 8.5 Color Update . . . 84
- 8.6 Segment Tree Iterative . . . 85
- 8.7 DSU With Partial Persistence . . . 86

8.8 BIT 1D . . . 86
8.9 Dynamic Segment Tree With Lazy Propagation . . . 86
8.10 Merge Sort Tree . . . 87
8.11 Treap . . . 88
8.12 Max Queue . . . 90
8.13 BIT 2D . . . 90
8.14 Ordered Set With BIT . . . 90
8.15 Dynamic Segment Tree With Vector . . . 91
8.16 Implicit Treap . . . 92
8.17 Sparse Table RMQ . . . 93
8.18 LiChao Tree . . . 93
8.19 Wavelet Tree . . . 94
8.20 PBDS . . . 95
8.21 TreeIsomorfismWithPolynomialHashing . . . 96
8.22 Merge Sort Tree Iterative . . . 97
8.23 TreeIsomorfismWithMap . . . 98
8.24 Centroid Decomposition . . . 99
8.25 Heavy Light Decomposition . . . 99
8.26 BIT Range Sum And Range Update . . . 101
8.27 Persistent Segment Tree . . . 102
8.28 Merge Sort Tree Range Order Statistics Queries . . . 103
8.29 Heavy Light Decomposition Path And Subtree Queries . . . 103

1 Geometry

1.1 Distance Between Nearest Pair Of Points

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e4;
const double EPS = 1e7;
typedef pair<double, double> ii;

vector<ii> v;
set<ii> sy, sx;
int n;

int main()
{
    double x, y;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> x >> y;
        v.push_back({x, y});
    }
    sort(v.begin(), v.end());
    double d = 0x3f3f3f3f;
    for(int i = 0; i < n; i++)
    {
        x = v[i].first, y = v[i].second;
        while(!sx.empty())
        {
            ii p = *(sx.begin());
            if(p.first + d < x)
            {
                sy.erase({p.second, p.first});
                sx.erase(p);
            }
            else
                break;
        }
        auto it = sy.lower_bound({int(floor(y-d))-1, 0});
        while(it != sy.end() and it->first < y + d + 1)
        {
            d = min(d, hypot(x - it->second, y - it->first));
            it++;
        }
        sy.insert({y, x});
        sx.insert({x, y});
    }
    cout << d << '\n';

    return 0;
}
```

1.2 Convex Hull Trick

```
#include <bits/stdc++.h>
using namespace std;
#define type int
const int MAX = 1e5;
const int OO = 0x3f3f3f3f;

struct line
{
  type m, b;
  line(type _m, type _b){ m = _m, b = _b; }
};

int pointer;//Keeps track of the best line from previous query
vector<line> hull;//store hull

//Returns true if line l3 is always better than line l2
bool bad(int l1, int l2, int l3)
{
  /*
    intersection(l1,l2) has x-coordinate (b1-b2)/(m2-m1)
  intersection(l1,l3) has x-coordinate (b1-b3)/(m3-m1)
  set the former greater than the latter, and cross-multiply to
    eliminate division
    */
  line L1 = hull[l1], L2 = hull[l2], L3 = hull[l3];
  return (L3.b-L1.b)*(L1.m-L2.m) < (L2.b-L1.b)*(L1.m-L3.m);
}

//Adds a new line
void add(type m, type b)
{
    if(hull.size() > 0 and hull.back().m == m) return;
     //First, let's add it to the end
    hull.emplace_back(m, b);
    //If the penultimate is now made irrelevant between the
        antepenultimate
    //and the ultimate, remove it. Repeat as many times as necessary
    while(hull.size()>=3 and bad(hull.size()-3,hull.size()-2,hull.size
        ()-1))
        hull.erase(hull.end()-2);
}

//Returns y value of a function i
type eval(int i, type x)
```

```
{
  return hull[i].m * x + hull[i].b;
}

//Returns the minimum y-coordinate of any intersection
//between a given vertical line and the lower envelope
//O(N) for all queries (queries are in ascending order of x)
type query(type x)
{
    if(pointer >= hull.size())
        pointer = hull.size() - 1;
    while(pointer < hull.size()-1 and eval(pointer+1, x) < eval(
        pointer, x))
    pointer++;
    return eval(pointer, x);
}

//Returns the minimum y-coordinate of any intersection
//between a given vertical line and the lower envelope
//O(LogN) time (queries are in any order of x)
type binarySearch(type x)
{
  int b = 0, e = hull.size() - 1;
  while(b < e)
  {
    int mid = (b + e) / 2;
    if(eval(mid+1, x) < eval(mid, x)) b = mid + 1;
    else e = mid;
  }
  return eval(b, x);
}

/*
    Maximum Y coordenate query, we have two options:
    1) Maximum Y-coordenate query: multiply m and b by -1 and
    make minimum Y-corrdenate query...

    2) Order lines by increasing m if m is not equal, otherwise by
        decreasing b
       in the function query and binary Search change < to >
            eval(pointer+1, x) < eval(pointer, x)
            to,
            eval(pointer+1, x) > eval(pointer, x)
*/

int main()
{
  int n;
    cin >> n;
    //Order lines by decreasing m if m is not equal, otherwise by
        increasing b
    for(int i = 0; i < n; i++)
    {
        int m, b;
        cin >> m >> b;
        add(m, b);
    }
    int q;
    cin >> q;
    vector<int> queries(q);//queries are in ascending order of x - run
         in O(N)
    for(int &w : queries)
    cin >> w;
  //processing queries in ascending order of x
  for(int &w : queries)
    cout << query(w) << '\n';

  int x;
  while(cin >> x)//queries are in any order of x - run in O(logN)
    cout << binarySearch(x) << '\n';

    return 0;
}
```

1.3 Check If A Point Is Inside A Convex Polygon

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define ii pair<int, int>
#define fi first
#define se second

int n;
vector<ii> P;

ii operator-(ii a, ii b)
{
    return {a.fi - b.fi, a.se - b.se};
}

int operator*(ii a, ii b)
{
    return a.fi * b.se - a.se * b.fi;
}

void setFirstPoint()
{
    int pos = 0;
    for(int i = 0; i < n; i++)
        if(P[i].fi < P[pos].fi or P[i].fi == P[pos].fi and P[i].se < P
            [pos].se)
            pos = i;
    rotate(P.begin(), P.begin() + pos, P.end());
}

bool pointInTriangle(ii a, ii b, ii c, ii p)
{
    int s1 = abs((a - c) * (b - c));
    int s2 = abs((a - p) * (b - p)) + abs((b - p) * (c - p)) +
       abs((c - p) * (a - p));
    return s1 == s2;//mesma area
}

int dist(ii a, ii b)
{
    return (a.fi - b.fi) * (a.fi - b.fi) + (a.se - b.se) * (a.se - b.
        se);
}

// O(logN) per query
```

```
bool pointInConvexPolygon(ii p)
{
    //adicionar = desconsidera pontos na borda
    if((P[1] - P[0]) * (p - P[0]) < 0)
        return false;
    //adicionar = desconsidera pontos na borda
    if((p - P[0]) * (P[n - 1] - P[0]) < 0)
        return false;
    //o ponto esta em cima do segento P[0], P[n-1]
    if((p - P[0]) * (P[n - 1] - P[0]) == 0)
        return  dist(P[0], p) <= dist(P[0], P[n - 1]) and dist(P[n -
            1], p) <= dist(P[0], P[n - 1]);
    //o ponto esta em cima do segento P[0], P[1]
    if((P[1] - P[0]) * (p - P[0]) == 0)
        return  dist(P[0], p) <= dist(P[0], P[1]) and dist(P[1], p) <=
             dist(P[0], P[1]);
    // se o ponto esta entre os segmentos P[0], P[n]
    int l = 0, e = n - 1, ans = 0;
    while(l <= e)
    {
        int m = l + (e - l) / 2;
        if((P[m] - P[0]) * (p - P[0]) >= 0) l = m + 1, ans = m;
        else e = m - 1;
    }
    return pointInTriangle(P[ans], P[ans + 1], P[0], p);
}

int32_t main()
{
    int q, x, y;
    cin >> n >> q;
    for(int i = 0; i < n; i++)
    {//poligono no sentido anti-horario
        cin >> x >> y;
        P.push_back({x, y});
    }
    setFirstPoint();
    while(q--)
    {
        cin >> x >> y;
        cout << (pointInConvexPolygon({x, y}) ? "Dentro" : "Fora") <<
            '\n';
    }

    return 0;
}
```

1.4 Geometry Stan

```
struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
  PT operator + (const PT &p) const { return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p) const { return PT(x-p.x, y-p.y); }
  PT operator * (double c) const { return PT(x*c, y*c ); }
  PT operator / (double c) const { return PT(x/c, y/c ); }
  bool operator == (PT p) const {
    return (fabs(x-p.x) < EPS && (fabs(y-p.y) < EPS)); }
  bool operator < (PT p) const {
    if(fabs(x-p.x) > EPS) return x<p.x; return y<p.y; };
};
// dot(p,q) = length(p)*length(q)*cos(angle between p and q)
double dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double dist(PT p, PT q) { return sqrt(dist2(p,q)); }
double mdist(PT p, PT q) { return fabs(p.x-q.x)+fabs(p.y-q.y); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {return os<<"("<<p.x<<",
    "<<p.y<<")";}
// 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));
}
// returns angle aob in rad
double angle(PT a, PT o, PT b){
  return acos(dot(a-o,b-o)/sqrt(dot(a-o,a-o)*dot(b-o,b-o)));
}
// returns true if point r is on the left side of line pq
bool ccw(PT p, PT q, PT r) {
  return cross(p,q)+cross(q,r)+cross(r,p) > 0;
}
// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}
// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}
// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}
// compute distance between point (x,y,z) and plane ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
double a, double b, double c, double d){
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}
// determine if lines from a to b and c to d are parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
    && fabs(cross(a-b, a-c)) < EPS
    && fabs(cross(c-d, c-a)) < EPS;
}
// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
```

```
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}
// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}
// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+RotateCW90
      (a-c));
}
// 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;
}
// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q), q) <
        EPS)
      return true;
  return false;
}
// 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;
}
// 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;
}
// 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));
}
// gravity center
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;
}
// tests whether or not a given polygon (in CW or CCW order) is simple
// segments do not intersect
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}
```

```
// compute two center's of circle given two points a and b
// and a radius r
pair<PT, PT> CumputeTwoCircleCenter(PT a, PT b, double R)
{
  if(dist(a, b) < EPS)
    return {a, b};

  PT middle = (a + b) / 2;
  PT v = RotateCCW90(middle - a);
  v = v / sqrt( dot(v, v) );

  double l = 0, r = 100, escalar;

  for(int i = 0; i < 100; i++)
  {
    double mid = (l + r) / 2;
    PT u = v * mid + middle;
    if(dist(a, u) <= R + EPS)
      l = mid, escalar = mid;
    else
      r = mid;
  }

  PT c1 = v * escalar + middle;
  PT c2 = v * (-escalar) + middle;

  return {c1, c2};
}
```

1.5 Dynamic Convex Hull Trick

```
#include <bits/stdc++.h>
using namespace std;
#define type __int128
#define int __int128
#define gc getchar
#define pc putchar
#define Min(a, b) (a > b ? b : a)

inline void scanint(int &k)
{
  bool sinal = true;
    register char c;
    k = 0;
    for(c = gc(); sinal and (c < '0' or c > '9'); c = gc())
      if(c == '-')
        sinal = false;
    for(; c >= '0' and c <= '9'; c = gc())
        k = (k << 3) + (k << 1) + c - '0';
  if(!sinal) k = -k;
}

inline void printint(int n)
{
  if(n < 0) pc('-');
  n = abs(n);
  int rev = n, cnt = 0;
  if(!n)
  {
    pc('0');
    pc('\n');
    return;
  }
  while(!(rev % 10))
    cnt++, rev /= 10;
  rev = 0;
  while(n)
    rev = (rev << 3) + (rev << 1) + n % 10, n /= 10;
  while(rev)
    pc(rev % 10 + '0'), rev /= 10;
  while(cnt--)
    pc('0');
  pc('\n');
}


struct line
{
  type m, b;
  line(type _m, type _b){ m = _m, b = _b; }
    line(){ m = 0, b = 0; }
};

bool bad(int l1, int l2, int l3, vector<line> &hull)
{
  line L1 = hull[l1], L2 = hull[l2], L3 = hull[l3];
  return (L3.b-L1.b)*(L1.m-L2.m) < (L2.b-L1.b)*(L1.m-L3.m);
}

void add(type m, type b, vector<line> &hull)
{
    if(hull.size() > 0 and hull.back().m == m) return;
    hull.emplace_back(m, b);
    while(hull.size()>=3 and bad(hull.size()-3,hull.size()-2,hull.size
        ()-1, hull))
        hull.erase(hull.end()-2);
}

type eval(int i, type x, vector<line> &hull)
{
  return hull[i].m * x + hull[i].b;
}

type binarySearch(type x, vector<line> &hull)
{
  int b = 0, e = hull.size() - 1;
  while(b < e)
  {
    int mid = (b + e) / 2;
    if(eval(mid+1, x, hull) < eval(mid, x, hull)) b = mid + 1;
    else e = mid;
  }
  return eval(b, x, hull);
}

//##########DAQUI PRA BAIXO EH O SUCESSO################

vector<line> merge(vector<line> a, vector<line> b)
{
    if(a.size() < b.size()) swap(a, b);
    for(int i = 0; i < b.size(); i++)
```

```
        a.push_back(b[i]);
    sort(a.begin(), a.end(), [](line c, line d)
      { return c.m == d.m ? c.b < d.b : c.m > d.m;  });
    b.clear();
    for(int i = 0; i < a.size(); i++)
        add(a[i].m, a[i].b, b);
    return b;
}

vector<vector<line>> groups;

void add(line l)
{
    vector<line> g = {l};
    while(!groups.empty() and groups.back().size() <= g.size())
    {
        g = merge(g, groups.back());
        groups.pop_back();
    }
    groups.push_back(g);
}

type query(int x)
{
    int ans = 0;
    for(int i = 0; i < groups.size(); i++)
        ans = Min(ans, binarySearch(x, groups[i]));
    return -ans;
}

int32_t main()
{
  int n, q;
  scanint(n);
  scanint(q);

    vector<line> cyc(n + 1);

    while(q--)
    {
        int t, T;
        scanint(t);
        scanint(T);
        if(t % 2 == 1)
        {
            int N, C;
            scanint(C);
            scanint(N);
            int b = - N * T + cyc[C].m * T + cyc[C].b;
            //cout << N << ' ' << b << '\n';
            add(line(-N, -b));
            cyc[C] = line(N, b);
        }
        else
            printint(query(T));
    }

    return 0;
}
```

1.6 Build Two Lines That Go Through All Points Of A Set

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e6 + 10;
typedef long long ll;

struct Point
{
    ll x, y;
    Point(ll _x, ll _y) : x(_x), y(_y) {}
    Point() {}
    Point operator-(const Point& p)
    {
        return Point(x - p.x, y - p.y);
    }
    ll operator*(const Point& p)
    {
        return (x * p.y) - (y * p.x);
    }
};

int n;
int visit[MAX];
Point point[MAX];

bool inLine(int a, int b, int c)
{
    return ((point[a] - point[c]) * (point[b] - point[c])) == 0LL;
}

bool check(int a, int b)// traca a reta AB e verifica se todos os
//pontos que nao estao em AB estao contidos em uma mesma reta
{
    memset(visit, 0, sizeof(visit));
    visit[a] = visit[b] = 1;
    for(int i = 0; i < n; i++)
        if(!visit[i] and inLine(a, b, i))
            visit[i] = 1;// marco todos os pontos que estao na reta AB
    vector<int> c;
    for(int i = 0; i < n and c.size() < 2; i++)
        if(!visit[i])
            c.push_back(i);// procuro dois pontos que nao estao na
                reta AB
    if(c.size() < 2) return true;
    visit[c[0]] = visit[c[1]] = 1;
    for(int i = 0; i < n; i++)
        if(!visit[i])
        {   // checo se o ponto que nao esta na reta AB esta na reta
            C0C1
            if(inLine(c[0], c[1], i))
                visit[i] = 1;
            else
                return false;
        }
    return true;
}

int main()
```

```
{
    cin >> n;
    for(int i = 0; i < n; i++)
        cin >> point[i].x >> point[i].y;
    if(n <= 2) return cout << "YES\n", 0;
    int k = 2;
    while(k < n and inLine(0, 1, k)) k++;
    if(k == n) return cout << "YES\n", 0;
    cout << ((check(0, 1) or check(0, k)
      or check(1, k)) ? "YES\n" : "NO\n");

    return 0;
}
```

1.7 Graham Scan

```
#include <bits/stdc++.h>
using namespace std;
#define ii pair<int, int>
#define fi first
#define se second

vector<ii> P;

ii operator-(ii a, ii b)
{
    return ii(a.fi - b.fi, a.se - b.se);
}

ii operator+(ii a, ii b)
{
    return ii(a.fi + b.fi, a.se + b.se);
}

int  operator*(ii a, ii b)
{
    return a.fi * b.se - a.se * b.fi;
}

int dist(ii a, ii b)
{
    return (a.fi - b.fi) *  (a.fi - b.fi) + (a.se - b.se) *  (a.se - b
        .se);
}

bool cmp(ii a, ii b)
{
    int cross = (a - P[0]) * (b - P[0]);
    if(!cross) return dist(P[0], a) > dist(P[0], b);
    return cross > 0;
}

void setFirstPoint()
{
    for(int i = 1; i < P.size(); i++)
        if(P[i].fi < P[0].fi or P[i].fi == P[0].fi and P[i].se < P[0].
            se)
            swap(P[0], P[i]);
}

vector<ii> GrahamScan()
{
    setFirstPoint();
    sort(P.begin() + 1, P.end(),cmp);
    vector<ii> H(P.size() * 2);
    int k = 0;
    for(int i = 0; i < P.size(); i++)
    {   //crsso <= 0 para remover os pontos colineares
        while(k > 2 and (H[k - 1] - H[k - 2]) * (P[i] - H[k - 1]) < 0)
             k--;
        H[k++] = P[i];
    }
    H.resize(k);
    return H;
}

int main()
{
    int n;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        ii p;
        cin >> p.fi >> p.se;
        P.push_back(p);
    }
    vector<ii> H = GrahamScan();
    for(int i = 0; i < H.size(); i++)
        cout << H[i].fi << ' ' << H[i].se << '\n';
    return 0;
}
```

1.8 Radial Sort

```
#include <bits/stdc++.h>
using namespace std;
#define type int
#define point pair<type, type>
#define X first
#define Y second

point operator-(point a, point b)
{
    return {a.X - b.X, a.Y - b.Y};
}

type operator*(point a, point b)
{
    return a.X * b.Y - a.Y * b.X;
}

int n;
vector<point> P;
point R;

int dist(point a, point b)
{
    return (a.X - b.X) * (a.X - b.X) + (a.Y - b.Y) * (a.Y - b.Y);
}
```

```
bool cmp(point a, point b)
{
    if((a - R).Y * (b - R).Y <= 0) return a.Y > R.Y;
    int c = (a - R) * (b - R);
    if(c == 0) return dist(R, a) <= dist(R, b);
    return c > 0;
}

int main()
{
    cin >> n >> R.X >> R.Y;
    for(int i = 0; i < n; i++)
    {
        type x, y;
        cin >> x >> y;
        P.push_back({x, y});
    }
    sort(P.begin(), P.end(), cmp);
    for(point p : P) cout << p.X << ' ' << p.Y << '\n';

    return 0;
}
```

1.9 Enclosing Circle R2

```
#include <cstdio>
#include <cmath>
int n;
double x[1005], y[1005], X, Y, d, e;
double dist(double a, double b) {
  return a*a + b*b;
}
int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    scanf("%lf%lf", &x[i], &y[i]);
    X += x[i]; Y += y[i];
  }
  X /= n; Y /= n;
  double P = 0.1;
  for (int i = 0; i < 30000; i++) {
    int f = 0;
    d = dist(X - x[0], Y - y[0]);
    for (int j = 1; j < n; j++) {
      e = dist(X - x[j], Y - y[j]);
      if (d < e) { d = e; f = j; }
    }
    X += (x[f] - X)*P;
    Y += (y[f] - Y)*P;
    P *= 0.999;
  }
  printf("%.3lf %.3lf\n%.3lf", X, Y, sqrt(d));
}
```

1.10 Enclosing Circle R3

```
#include <cstdio>
#include <cmath>
int n;
double x[105], y[105], z[105], X, Y, Z, d, e;
double dist(double a, double b, double c) {
  return a*a + b*b + c*c;
}
int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; i++) {
    scanf("%lf%lf%lf", &x[i], &y[i], &z[i]);
    X += x[i];
    Y += y[i];
    Z += z[i];
  }
  X /= n; Y /= n; Z /= n;
  double P = 0.1;
  for (int i = 0; i < 70000; i++) {
    int f = 0;
    d = dist(X - x[0], Y - y[0], Z - z[0]);
    for (int j = 1; j < n; j++) {
      e = dist(X - x[j], Y - y[j], Z - z[j]);
      if (d < e) {
        d = e;
        f = j;
      }
    }
    X += (x[f] - X)*P;
    Y += (y[f] - Y)*P;
    Z += (z[f] - Z)*P;
    P *= 0.998;
  }
  printf("%.10lf %.10lf %.10lf", X, Y, Z);
}
```

1.11 Andrew Algorithm Convex Hull

```
#include <bits/stdc++.h>
using namespace std;
#define X first
#define Y second
typedef pair<int, int> ii;

int cross(ii O, ii A, ii B)
{
    return (((A.X - O.X) * (B.Y - O.Y)) - ((A.Y - O.Y) * (B.X - O.X)))
        ;
}

vector<ii> ConvexHull(vector<ii> P)
{
  if(P.size() <= 1) return P;
    vector<ii> H(2*P.size());
    int k = 0;
    sort(P.begin(), P.end());
    //lower hull
    for(int i = 0; i < P.size(); i++)
    {
        while(k >= 2 and cross(H[k-2], H[k-1], P[i]) < 0) k--;
        H[k++] = P[i];
    }
    //upper hull
```

```
    for(int i = P.size()-2, l = k + 1; i >= 0; i--)
    {
        while(k >= l and cross(H[k-2], H[k-1], P[i]) < 0) k--;
        H[k++] = P[i];
    }
    H.resize(k-1);
    return H;
}

int main()
{
    int n, x, y;
    vector<ii> P;

    cin >> n;
    while(n--)
    {
        cin >> x >> y;
        P.push_back({x, y});
    }

    vector<ii> H = ConvexHull(P);

    for(int i = 0; i < H.size(); i++)
        cout << H[i].X << ' ' << H[i].Y << '\n';

  return 0;
}
```

1.12 Segment Intersection

```
#define ll long long
#define Pll complex<ll>
#define Y imag()
#define X real()

bool intersec(Pll a, Pll b, Pll c, Pll d)
{
  if(cross(c - a, c - b) * cross(d - a, d - b) <= 0  and
    cross(a - c, a - d) * cross(b - c, b - d) <= 0)
    return true;
  return false;
}
```

1.13 Maximum Dot Product

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define X first
#define Y second
const int OO = 0x3f3f3f3f3f3f3f3f;
typedef pair<int, int> ii;

int cross(ii O, ii A, ii B)
{
    return (((A.X - O.X) * (B.Y - O.Y)) - ((A.Y - O.Y) * (B.X - O.X)))
        ;
}

int dot(ii a, ii b)
{
    return a.X * b.X + a.Y * b.Y;
}

vector<ii> ConvexHull(vector<ii> P)
{
    if(P.size() <= 1) return P;
    vector<ii> H(2*P.size());
    int k = 0;
    sort(P.begin(), P.end());
    for(int i = 0; i < P.size(); i++)
    {
        while(k >= 2 and cross(H[k-2], H[k-1], P[i]) <= 0) k--;
        H[k++] = P[i];
    }
    for(int i = P.size()-2, l = k + 1; i >= 0; i--)
    {
        while(k >= l and cross(H[k-2], H[k-1], P[i]) <= 0) k--;
        H[k++] = P[i];
    }
    H.resize(k-1);
    return H;
}

vector<ii> merge(vector<ii> H1, vector<ii> H2)
{
    for(auto &it : H2) H1.push_back(it);
    return ConvexHull(H1);
}

int maxConcavityUp(int b, int e, vector<ii> &H, ii p)
{
    if(b > e) return -OO;
    return max(dot(H[b], p), dot(H[e], p));
}

int maxConcavityDown(int b, int e, vector<ii> &H, ii p)
{
    if(b > e) return -OO;
    b--;
    while(e - b > 1)
    {
      int m = b + (e - b) / 2;
      if(dot(H[m], p) > dot(H[m + 1], p))
          e = m;
      else
          b = m;
    }
    return dot(H[e], p);
}

int maximumDot(vector<ii> &H, ii p)
{
    bool growing = dot(H[0], p) <= dot(H[1], p);
    if(growing)
    {
      int b = 0, e = H.size() - 1, w = -1;
      while(b <= e)
```

```
        {
            int m = (b + e) / 2;
            if(dot(H[0], p) <= dot(H[m], p))
                b = m + 1, w = m;
            else
                e = m - 1;
        }
        return max(maxConcavityUp(w, H.size() - 1, H, p),
           maxConcavityDown(0, w, H, p));
        //cout << "caso #1\n0 " << w << " concavidade para baixo\n"
        //     << w + 1 << ' ' << H.size() - 1 << " concavidade para
            cima\n";
    }
    else
    {
      int b = 0, e = H.size() - 1, w = -1;
      while(b <= e)
      {
          int m = (b + e) / 2;
          if(dot(H[0], p) >= dot(H[m], p))
              b = m + 1, w = m;
          else
              e = m - 1;
      }
      //cout << "caso #2\n0 " << w << " concavidade para cima\n"
      //     << w + 1 << ' ' << H.size() - 1 << " cocavidade para
          baixo\n";
      return max(maxConcavityUp(0, w, H, p),
        maxConcavityDown(w, H.size() - 1, H, p));
    }
}

vector<vector<ii>> st;

void add(ii p)
{
    vector<ii> g = {p};
    while(!st.empty() and st.back().size() <= g.size())
    {
        g = merge(g, st.back());
        st.pop_back();
    }
    st.push_back(g);
}

int query(ii p)
{
    int ans = -OO;
    for(int i = 0; i < st.size(); i++)
        ans = max(ans, maximumDot(st[i], p));
    return ans;
}

int32_t main()
{
    int n, q;
    scanf(" %lld", &n);
    for(int i = 0; i < n; i++)
    {
        int x, y;
        scanf(" %lld %lld", &x, &y);
        add({x, y});
    }
    scanf(" %lld", &q);
    while(q--)
    {
        char s[10];
        int x, y;
        scanf(" %s %lld %lld", s, &x, &y);
        if(s[0] == 'a') add({x, y});
        else printf("%lld\n", query({x, y}));
    }

    return 0;
}
```

2 String

2.1 Trie Static

```
#include <bits/stdc++.h>
using namespace std;

// as posicoes de 0 ate 25 representam as letras
// de a ate z do alfabeto.
// a posicao 26 armazena quantas strings terminam
// nesse vertice.
// a posicao 27 armazena quantas strings passam
// nesse vertice.

int trie[8000000][30], CUR = 1;

// fl eh zero se for uma operacao de inserir
// fl eh um se for uma operacao de buscar
int add(string &s, int fl)
{
  int root = 0;
  for(char &c : s)
  {
    if(trie[root][c - 'a'] == 0)
    {
      if(fl) return 0;
      trie[root][c - 'a'] = CUR++;
    }
    if(!fl) trie[root][27]++;
    root = trie[root][c - 'a'];
  }
  if(fl) return trie[root][26];
  trie[root][26]++;
  return 1;
}

void sub(string &s)
{
  int root = 0;
  for(char &c : s)
    if(trie[root][c - 'a'] and trie[root][27])
    {
      trie[root][27]--;
```

```
      root = trie[root][c - 'a'];
    }
  trie[root][26]--;
}

int main()
{
  int q;

  cin >> q;

  while(q--)
  {
    int o;
    string s;
    cin >> o >> s;
    if(o == 1) add(s, 0);
    else if(o == 2) puts(add(s, 1) ? "existe" : "nao existe");
    else sub(s);
  }

  return 0;
}
```

2.2 Suffix Array

```
#include <bits/stdc++.h>
using namespace std;

string txt; // texto
string pat; // padrao
int n; // tamanho do texto
int chave[100]; // chave para comparacao dos sufixos
int vs[100]; // vetor de sufixos
int ord[100]; // ordem de um sufixo (qual classe ele pertence)
int lcp[100]; // maior prefixo comum

bool comp(int a, int b)
{
    return chave[a] < chave[b];
}

void constroi()// O(N*Log(N)*Log(N))
{
    for(int i = 0; i < n; i++)
    {
        vs[i] = i;
        chave[i] = txt[i] - 'a' + 1;
    }
    sort(vs, vs+n, comp);
    for(int i = 0; ; i++)
    {
        int classes = 0;
        for(int j = 0; j < n; j++)
            ord[vs[j]] = j > 0 and chave[vs[j]]
              == chave[vs[j-1]] ? ord[vs[j-1]] : ++classes;
        if(classes == n) break;
        for(int j = 0; j < n; j++)
        {
            chave[j] = ord[j]*(classes+1);
            chave[j] += j+(1<<i) < n ? ord[j+(1<<i)] : 0;
        }
        sort(vs, vs+n, comp);
    }
}

int strcompara(int pos)
{// retorna 0 se o padrao esta no sufixo, maior que zero se o padrao
    eh
//lexicograficamente maior que o sufixo, e menor que 0 se o padrao
//eh lexicograficamente menor que o sufixo
    for(int i = 0; i < pat.size(); i++)
        if(i+pos >= n)
            return 1;
        else if(pat[i] != txt[i+pos])
            return pat[i] - txt[i+pos];
    return 0;
}

bool search()// O(Size(Pat)*Log(N))
{
    int b = 0, e = n - 1, m, aux;
    while(b <= e)
    {
        m = (b + e) / 2;
        aux = strcompara(vs[m]);
        if(aux == 0)
            return true;
        else if(aux > 0)
            b = m + 1;
        else
            e = m - 1;
    }
    return false;
}

// numero de vezes que o padrao aparece no texto. O(Size(Pat)*Log(N))
int numberOfOcur()
{
    int b = 0, e = n - 1, m, aux, l = INT_MAX, r = INT_MIN;
    while(b <= e)
    {
        m = (b + e) / 2;
        aux = strcompara(vs[m]);
        if(aux == 0)
        {
            l = min(l, m);
            e = m - 1;
        }
        else if(aux > 0)
            b = m + 1;
        else
            e = m - 1;
    }
    b = 0, e = n - 1;
    while(b <= e)
    {
        m = (b + e) / 2;
        aux = strcompara(vs[m]);
        if(aux == 0)
        {
```

```
            r = max(r, m);
            b = m + 1;
        }
        else if(aux > 0)
            b = m + 1;
        else
            e = m - 1;
    }
    return abs(r-l+1);
}

// kasai em O(NlogN) pra construir o LCP (longest common prefix)
void kasai()
{
    vector<int> invSuff(n, 0);
    for(int i = 0; i < n; i++)
        invSuff[vs[i]] = i;
    int k = 0;
    for(int i = 0; i < n; i++)
    {
        if(invSuff[i] == n-1)
        {
            k = 0;
            continue;
        }
        int j = vs[invSuff[i]+1];
        while(i + k < n and j + k < n and txt[i + k] == txt[j + k])
            k++;
        lcp[invSuff[i]] = k;
        if(k > 0)
            k--;
    }
}

void printAll()
{
    cout << "Vetor de sufixos:\n";
    for(int i = 0; i < n; i++)
        cout << vs[i] << ' ';
    cout << '\n';
    cout << "Sufixos em ordem:\n";
    for(int i = 0; i < n; i++)
        cout << txt.substr(vs[i]) << '\n';
}

/*
  Dado um array LCP onde LCP[i] armazena o tamanho do maior prefixo
   em comum entre os sufixos i e i + 1 da suffix array
  , entao para achar o maior prefixo em comum entre dois sufixos que
   estao nas posicoes a e b da suffix array, corresponde a achar
    o menor valor no intervalo [a, b-1] no LCP array.

  Outra aplicacao eh dada uma string, para contar quantas substrings
   diferentes ela tem basta contar quantas tem no total ( ((n + 1) * n
       )/2
    possiveis substrings) e remover todos os valores de LCP do total.
*/

int main()
{
    cin >> txt;
    n = txt.size();
    constroi();
    printAll();

    cout << '\n';

    cin >> pat;
    cout << (search() ? "found " : "not found ") <<  numberOfOcur()
        << " vez(es)\n\n";

    kasai();
    cout << "LCP\n";
    for(int i = 0; i < n; i++)
      cout << lcp[i] << ' ';
    cout << '\n';

    return 0;
}
```

2.3 LIS LDS

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e4;

int n;
int LI[MAX], LD[MAX];
vector<int> arr;

int LIS() {
    for(int i = 0; i < n; i++)
        LI[i] = 1;
    for(int i = n - 1; i >= 0; i--)
        for(int j = 0; j < i; j++)
            if(arr[j] < arr[i])
                LI[j] = max(LI[j], LI[i] + 1);
}

int LDS() {
    reverse(arr.begin(), arr.end());
    for(int i = 0; i < n; i++)
        LD[i] = 1;
    vector<int> pilha;
    for(int i = 0; i < n; i++) {
        int p = (int)(lower_bound(pilha.begin(),
          pilha.end(), arr[i]) - pilha.begin());
        if(p == pilha.size())
            pilha.push_back(arr[i]);
        else
            pilha[p] = arr[i];
        LD[i] = p + 1;
    }

}

int main() {
    cin >> n; arr.resize(n);
    for(int i = 0; i < n; i++) cin >> arr[i];

    LIS();
```

```
    LDS();

    for(int i = 0; i < n; i++)
      cout << LI[i] << ' '; puts("");
    for(int i = 0; i < n; i++)
      cout << LD[n - i - 1] << ' '; puts("");

  return 0;
}
```

2.4 SA

```
/*
 * Code frin Competitive Programming 3
 */

#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;

typedef pair<int, int> ii;

#define MAX_N 100010                          // second approach: O(n log n)
char T[MAX_N];                    // the input string, up to 100K characters
int n;                                        // the length of input string
int RA[MAX_N], tempRA[MAX_N];         // rank array and temporary rank array
int SA[MAX_N], tempSA[MAX_N];     // suffix array and temporary suffix array
int c[MAX_N];                                 // for counting/radix sort

char P[MAX_N];                    // the pattern string (for string matching)
int m;                                        // the length of pattern string

int Phi[MAX_N];                       // for computing longest common prefix
int PLCP[MAX_N];
int LCP[MAX_N];  // LCP[i] stores the LCP between previous suffix T+SA[i-1]
                                              // and current suffix T+SA[i]

bool cmp(int a, int b) { return strcmp(T + a, T + b) < 0; }      // compare

void constructSA_slow() {                // cannot go beyond 1000 characters
  for (int i = 0; i < n; i++) SA[i] = i; // initial SA: {0, 1, 2, ..., n-1}
  sort(SA, SA + n, cmp); // sort: O(n log n) * compare: O(n) = O(n^2 log n)
}
```

```
void countingSort(int k) {                                              // O(n)
  int i, sum, maxi = max(300, n);   // up to 255 ASCII chars or length of n
  memset(c, 0, sizeof c);                           // clear frequency table
  for (i = 0; i < n; i++)       // count the frequency of each integer rank
    c[i + k < n ? RA[i + k] : 0]++;
  for (i = sum = 0; i < maxi; i++) {
    int t = c[i]; c[i] = sum; sum += t;
  }
  for (i = 0; i < n; i++)           // shuffle the suffix array if necessary
    tempSA[c[SA[i]+k < n ? RA[SA[i]+k] : 0]++] = SA[i];
  for (i = 0; i < n; i++)                       // update the suffix array SA
    SA[i] = tempSA[i];
}

void constructSA() {          // this version can go up to 100000 characters
  int i, k, r;
  for (i = 0; i < n; i++) RA[i] = T[i];                 // initial rankings
  for (i = 0; i < n; i++) SA[i] = i;      // initial SA: {0, 1, 2, ..., n-1}
  for (k = 1; k < n; k <<= 1) {       // repeat sorting process log n times
    countingSort(k);  // actually radix sort: sort based on the second item
    countingSort(0);           // then (stable) sort based on the first item
    tempRA[SA[0]] = r = 0;              // re-ranking; start from rank r = 0
    for (i = 1; i < n; i++)                     // compare adjacent suffixes
      tempRA[SA[i]] = // if same pair => same rank r; otherwise, increase r
      (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+k]) ? r : ++r;
    for (i = 0; i < n; i++)                      // update the rank array RA
      RA[i] = tempRA[i];
    if (RA[SA[n-1]] == n-1) break;               // nice optimization trick
} }

void computeLCP_slow() {
  LCP[0] = 0;                                                 // default value
  for (int i = 1; i < n; i++) {                 // compute LCP by definition
    int L = 0;                                              // always reset L to 0
    while (T[SA[i] + L] == T[SA[i-1] + L]) L++;      // same L-th char, L++
    LCP[i] = L;
} }

void computeLCP() {
```

```
  int i, L;
  Phi[SA[0]] = -1;                                    // default
      value
  for (i = 1; i < n; i++)                             // compute Phi in
      O(n)
    Phi[SA[i]] = SA[i-1];    // remember which suffix is behind this
        suffix
  for (i = L = 0; i < n; i++) {                       // compute Permuted LCP in
      O(n)
    if (Phi[i] == -1) { PLCP[i] = 0; continue; }      // special
        case
    while (T[i + L] == T[Phi[i] + L]) L++;            // L increased max n
        times
    PLCP[i] = L;
    L = max(L-1, 0);                                  // L decreased max n
        times
  }
  for (i = 0; i < n; i++)                             // compute LCP in
      O(n)
    LCP[i] = PLCP[SA[i]];   // put the permuted LCP to the correct
        position
}

ii stringMatching() {                                 // string matching in O(m
    log n)
  int lo = 0, hi = n-1, mid = lo;                     // valid matching =
      [0..n-1]
  while (lo < hi) {                                   // find lower
      bound
    mid = (lo + hi) / 2;                              // this is round
        down
    int res = strncmp(T + SA[mid], P, m);  // try to find P in suffix
        'mid'
    if (res >= 0) hi = mid;         // prune upper half (notice the >=
        sign)
    else          lo = mid + 1;           // prune lower half
        including mid
  }                                   // observe '=' in "res >= 0"
      above
  if (strncmp(T + SA[lo], P, m) != 0) return ii(-1, -1);    // if not
      found
  ii ans; ans.first = lo;
  lo = 0; hi = n - 1; mid = lo;
  while (lo < hi) {             // if lower bound is found, find upper
      bound
    mid = (lo + hi) / 2;
    int res = strncmp(T + SA[mid], P, m);
    if (res > 0) hi = mid;                            // prune upper
        half
    else         lo = mid + 1;            // prune lower half
        including mid
  }                           // (notice the selected branch when res
      == 0)
  if (strncmp(T + SA[hi], P, m) != 0) hi--;                 // special
      case
  ans.second = hi;
  return ans;
} // return lower/upperbound as first/second item of the pair,
    respectively

ii LRS() {                  // returns a pair (the LRS length and its
    index)
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)                             // O(n), start from
      i = 1
    if (LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}

int owner(int idx) { return (idx < n-m-1) ? 1 : 2; }

ii LCS() {                  // returns a pair (the LCS length and its
    index)
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)                             // O(n), start from
      i = 1
    if (owner(SA[i]) != owner(SA[i-1]) && LCP[i] > maxLCP)
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}

int main() {
  //printf("Enter a string T below, we will compute its Suffix Array:\
      n");
  strcpy(T, "GATAGACA");
  n = (int)strlen(T);
  T[n++] = '$';
  // if '\n' is read, uncomment the next line
  //T[n-1] = '$'; T[n] = 0;

  constructSA_slow();                                                 // O(n^2
      log n)
  printf("The Suffix Array of string T = '%s' is
    shown below (O(n^2 log n) version):\n", T);
  printf("i\tSA[i]\tSuffix\n");
  for (int i = 0; i < n; i++) printf("%2d\t%2d\t%s\n", i, SA[i], T +
      SA[i]);

  constructSA();                                                       // O(n
      log n)
  printf("\nThe Suffix Array of string T = '%s' is
     shown below (O(n log n) version):\n", T);
  printf("i\tSA[i]\tSuffix\n");
  for (int i = 0; i < n; i++) printf("%2d\t%2d\t%s\n", i, SA[i], T +
      SA[i]);

  computeLCP();                                                                //
       O(n)

  // LRS demo
  ii ans = LRS();                  // find the LRS of the first input
      string
  char lrsans[MAX_N];
  strncpy(lrsans, T + SA[ans.second], ans.first);
  printf("\nThe LRS is '%s' with length = %d\n\n", lrsans, ans.first);

  // stringMatching demo
  //printf("\nNow, enter a string P below, we will try to find P in T
      :\n");
  strcpy(P, "A");
  m = (int)strlen(P);
```

```
  // if '\n' is read, uncomment the next line
  //P[m-1] = 0; m--;
  ii pos = stringMatching();
  if (pos.first != -1 && pos.second != -1) {
    printf("%s is found SA[%d..%d] of %s\n", P, pos.first, pos.second,
         T);
    printf("They are:\n");
    for (int i = pos.first; i <= pos.second; i++)
      printf("  %s\n", T + SA[i]);
  } else printf("%s is not found in %s\n", P, T);

  // LCS demo
  //printf("\nRemember, T = '%s'\nNow, enter another string P:\n", T);
  // T already has '$' at the back
  strcpy(P, "CATA");
  m = (int)strlen(P);
  // if '\n' is read, uncomment the next line
  //P[m-1] = 0; m--;
  strcat(T, P);                                                    //
      append P
  strcat(T, "#");                                        // add '$' at the
       back
  n = (int)strlen(T);                                              //
      update n

  // reconstruct SA of the combined strings
  constructSA();                                                 // O(n
      log n)
  computeLCP();                                                        //
       O(n)
  printf("\nThe LCP information of 'T+P' = '%s':\n", T);
  printf("i\tSA[i]\tLCP[i]\tOwner\tSuffix\n");
  for (int i = 0; i < n; i++)
    printf("%2d\t%2d\t%2d\t%2d\t%s\n", i, SA[i], LCP[i], owner(SA[i]),
         T + SA[i]);

  ans = LCS();          // find the longest common substring between T
      and P
  char lcsans[MAX_N];
  strncpy(lcsans, T + SA[ans.second], ans.first);
  printf("\nThe LCS is '%s' with length = %d\n", lcsans, ans.first);

  return 0;
}
```

2.5 Longest Common Substring

```
#include <stdio.h>
#include <string.h>

int pd[10000][10000];

int max(int a, int b)
{
  return a > b ? a : b;
}

int solve(char *a, char *b)
{
  int i, j, ans = 0;
  int t1 = strlen(a), t2 = strlen(b);
  for(i = 1; i < t1; i++)
  for(j = 1; j < t2; j++)
    if(a[i-1] == b[j-1])
      pd[i][j] = pd[i-1][j-1] + 1,
        ans = max(ans, pd[i][j]);
    else
      pd[i][j] = 0;
  return ans;
}

int main()
{
  char s1[55], s2[55];

  while(fgets(s1, 54, stdin) != NULL
    && fgets(s2, 54, stdin) != NULL)
    printf("%d\n", solve(s1, s2));

  return 0;
}
```

2.6 Dynamic Trie

```
#include <bits/stdc++.h>
using namespace std;

struct TrieNode
{
  map<int, TrieNode*> childreen;
  bool isLeaf;
  TrieNode()
  {
    isLeaf = false;
  }
};

void inserir(TrieNode *root, string s)
{
  TrieNode *node = root;
  for(int i = 0; i < s.size(); i++)
  {
    int index = s[i] - 'a';
    if(node->children.find(index) == node->children.end())
      node->children[index] = new TrieNode();
    node = node->children[index];
  }
  node->isLeaf = true;
}

bool buscar(TrieNode *root, string s)
{
  TrieNode *node = root;
  for(int i = 0; i < s.size(); i++)
  {
    int index = s[i] - 'a';
    if(node->children.find(index) == node->children.end())
      return false;
    node = node->children[index];
  }
```

```
  return node->isLeaf;
}

bool remover(TrieNode *node, string s, int level)
{
  if(node != nullptr)
    if(s.size() == level)
    {
      if(node->isLeaf)
      {
        node->isLeaf = false;
        return !node->childreen.size();
      }
    }
    else
    {
      int index = s[level] - 'a';
      if(remover(node->childreen[index], s, level+1))
      {
        delete node->childreen[index];
        node->childreen.erase(index);
        return !node->childreen.size();
      }
    }
  return false;
}

int main()
{
  TrieNode *root = new TrieNode();

  inserir(root, "abc");
  inserir(root, "abd");
  inserir(root, "cfa");
  remover(root, "abc", 0);
  printf(buscar(root, "abc") ? "yes\n" : "no\n");
  printf(buscar(root, "abd") ? "yes\n" : "no\n");

  return 0;
}
```

2.7 Z Function

```
#include "bits/stdc++.h"
#define Max(a,  b) (a > b ? a : b)
#define Min(a,  b) (a < b ? a : b)
using namespace std;
const int MAX = 1e5;

vector<int> Z(string &s)
{
  int n = s.size(), x = 0, y = 0;
  vector<int> z(n);
  for(int i = 1; i < n; i++)
  {
    z[i] = Max(0, Min(z[i - x], y - i + 1));
    while(i + z[i] < n and s[z[i]] == s[i + z[i]])
      x = i, y = i + z[i], z[i]++;
  }
  return z;
}

int main()
{
  string txt, pattern;

  cin >> txt >> pattern;
  string s = pattern + "#" + txt;
  vector<int> z = Z(s);
  for(int &w : z)
    cout << w << ' ';
  cout << '\n';

  return 0;
}
```

2.8 Suffix Array And Applications

```
#include <bits/stdc++.h>
using namespace std;

string s;
vector<int> sa, c, lcp;

// O(n)
void countSort()
{
  int n = sa.size();
  vector<int> buc(n), new_sa(n);
  for(int &w : sa)
    buc[c[w]]++;
  for(int i = 1; i < n; i++)
    buc[i] += buc[i - 1];
  for(int i = n - 1; i >= 0; i--)
    new_sa[ --buc[ c[sa[i]] ] ] = sa[i];
  sa = new_sa;
}

// O(|s| * log|s|)
void buildSuffixArray()
{
  int n = s.size();
  sa.resize(n);
  c.resize(n);
  for(int i = 0; i < n; i++)
    sa[i] = i, c[i] = s[i];
  sort(sa.begin(), sa.end(), [&](int a, int b)
    {
      return c[a] < c[b];
    });
  c[sa[0]] = 0;
  for(int i = 1; i < n; i++)
    if(s[sa[i - 1]] == s[sa[i]])
      c[sa[i]] = c[sa[i - 1]];
    else
      c[sa[i]] = c[sa[i - 1]] + 1;
  int k = 0;
  while((1 << k) < n)
  {
    for(int i = 0; i < n; i++)
```

```
      sa[i] = (sa[i] - (1 << k) + n) % n;
    countSort();
    vector<int> new_c(n);
    new_c[sa[0]] = 0;
    for(int i = 1; i < n; i++)
    {
      pair<int, int> prev = {c[sa[i - 1]], c[(sa[i - 1] + (1 << k)) %
          n]};
      pair<int, int> cur = {c[sa[i]], c[(sa[i] + (1 << k)) % n]};
      if(prev == cur) new_c[sa[i]] = new_c[sa[i - 1]];
      else new_c[sa[i]] = new_c[sa[i - 1]] + 1;
    }
    c = new_c;
    k++;
  }
}

// 0: padrao esta no sufixo
// 1: o padrao eh lexicograficamente maior que o sufixo k
//-1: o padrao eh lexicograficamente menor que o sufixo k
// O(|p|)
int cmp(int k, string &p)
{
  for(int i = 0; i < p.size(); i++)
  {
    if(i + k >= s.size()) return 1;
    if(s[i + k] < p[i]) return 1;
    if(s[i + k] > p[i]) return -1;
  }
  return 0;
}

// posicao no suffix array do sufixo mais
// a esquerda que contem p como prefixo
// O(|p| * log|s|)
int lower_bound(string &p)
{
  int b = 0, e = (int)sa.size() - 1, ans = 0;
  while(b <= e)
  {
    int mid = (b + e) / 2;
    int r = cmp(sa[mid], p);
    if(!r)
      e = mid - 1, ans = mid;
    else if(r == -1)
      e = mid - 1;
    else
      b = mid + 1;
  }
  if(s.substr(sa[ans], p.size()) != p)
    return -1;
  return ans;
}

// posicao no suffix array do sufixo mais
// a direita que contem p como prefixo
// O(|p| * log|s|)
int upper_bound(string &p)
{
  int b = 0, e = (int)sa.size() - 1, ans = 0;
  while(b <= e)
  {
    int mid = (b + e) / 2;
    int r = cmp(sa[mid], p);
    if(!r)
      b = mid + 1, ans = mid;
    else if(r == -1)
      e = mid - 1;
    else
      b = mid + 1;
  }
  if(s.substr(sa[ans], p.size()) != p)
    return -1;
  return ans;
}

// numero de ocorrencias da string p
// como substring de s
// O(|p| * log|s|)
int count(string &p)
{
  int l = lower_bound(p);
  int u = upper_bound(p);
  if(l == -1 or u == -1)
    return 0;
  return u - l + 1;
}

// construcao do array lcp
// lcp[i] eh o maior prefixo comum
// aos sufixos i e i - 1 do suffix array
// O(n)
void buildLcp()
{
  int n = s.size();
  lcp.resize(n);
  int k = 0;
  for(int i = 0; i < n - 1; i++)
  {
  // pi eh a posicao no suffix array do
  // sufixo que comeca na posicao i da strig
    int pi = c[i];
    int j = sa[pi - 1];
    while(s[i + k] == s[j + k]) k++;
    lcp[pi] = k;
    k = max(k - 1, 0);
  }
}

// conta a quantidade de substrings
// diferentes na string s
// O(|s|)
long long numberOfDifSubStr()
{
  long long n = s.size();
  long long ans = n * (n - 1) / 2;
  for(int i = 0; i < n; i++)
    ans -= lcp[i];
  return ans;
}

// encontra a maior substring comum a s e p
```

```
// O(|s + p|) depois de construir suffix array
void longestCommonSubstring(string p)
{
  int n = s.size();
  int m = p.size();
  int ans = -1, j = 1;
  s = s + "$" + p + "#";

  buildSuffixArray();
  buildLcp();

  for(int i = 1; i < sa.size(); i++)
  {
    int p = sa[i - 1] < n ? 1 : -1;
    int q = sa[i] < n ? 1 : -1;
    if(p * q < 0 and ans < lcp[i])
      ans = lcp[i], j = i;
  }

  int a = sa[j];
  int b = sa[j - 1];
  string lcs;

  while(a < s.size() and b < s.size() and s[a] == s[b])
  {
    lcs.push_back(s[a]);
    a++, b++;
  }

  cout << ans << '\n';
  cout << lcs << '\n';
}

int32_t main()
{
  string p;
  cin >> s >> p;

    // o tamanho do maior prefixo comum entre dois
  // sufixos que estao nas posicoes a e b do
  // suffix array eh igual ao menor valor no
  // intervalo [a+1, b] do array lcp

  // descomentar as linhas abaixo para todas as
  // funcoes, exceto para longestCommonSubstring

  // s.push_back('$');
  // buildSuffixArray();
  // buildLcp();

  longestCommonSubstring(p);

  return 0;
}
```

2.9 Aho Corasick

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e4;

int p[MAX], f[MAX], nxt[MAX][26], ch[MAX];
int tsz = 1; // size of the trie

int cnt[MAX]; // used to know number of matches

const int S = 2e3+5;
bitset<MAX> elem[S];
// S eh tamanho da maior das N strings que sao
// pradroes para buscar no texto

void init()
{
    tsz = 1;
    memset(f, 0, sizeof(f));
    memset(nxt, 0, sizeof(nxt));
    memset(cnt, 0, sizeof(cnt));
    for (int i = 0; i < MAX; i++)
      elem[i].reset();
}

void add(const string &s, int x)
{
  // the first element of the trie is the root
    int cur = 1;
    for(int i = 0; s[i]; ++i)
    {
        int j = s[i] - 'a';
        if(!nxt[cur][j])
        {
            tsz++;
            p[tsz] = cur;
            ch[tsz] = j;
            nxt[cur][j] = tsz;
        }
        cur = nxt[cur][j];
    }
    cnt[cur]++;
    elem[cur].set(x);
}

void build()
{
    queue<int> q;
    for(int i = 0; i < 26; ++i)
    {
        nxt[0][i] = 1;
        if(nxt[1][i])
            q.push(nxt[1][i]);
    }
    while(!q.empty())
    {
        int v = q.front(); q.pop();
        int u = f[p[v]];
        while(u and !nxt[u][ch[v]]) u = f[u];
        f[v] = nxt[u][ch[v]];
        cnt[v] += cnt[f[v]];
        for(int i = 0; i < 26; ++i)
            if(nxt[v][i])
                q.push(nxt[v][i]);
    }
```

```
}

bitset<MAX> match(const string &s)
{
    int ans = 0;
    // Numero de matches
    bitset<MAX> found;
    // Usado pra saber quais strings matches
    int x = 1;
    for(int i = 0; i < s.size(); ++i)
    {
        int t = s[i] - 'a';
        while(x and !nxt[x][t])
            x = f[x];
        x = nxt[x][t];
        ans += cnt[x];
        found |= elem[x];
    }
    return found;
}

int main()
{
    int n;
    string s;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> s;
        add(s, i);
    }
    build();
    cin >> s;
    bitset<MAX> ans = match(s);
    for(int i = 0; i < n; i++)
        cout << ans[i] << '\n';
        // 1 se a i-esima string lida
        // aparece no texto, 0 cc
  return 0;
}

//
    ##########################################################################################################################


#include <bits/stdc++.h>
using namespace std;

const int K = 60;

struct Vertex {
   int next[K];
   bool leaf = false;
   int p = -1;
   char c;
   int link = -1;
   int go[K];
   bitset<1005> S;
   Vertex(int _p=-1, char _c = '$') : p(_p), c(_c) {
      fill(begin(next), end(next), -1);
      fill(begin(go), end(go), -1);
   }
};

vector<Vertex> t;

void init() {
   t.clear();
   t.resize(1);
}

void add(string &s, int i) {
   int v = 0;
   for(char ch : s) {
      int c = ch - 'A';
      if(t[v].next[c] == -1) {
         t[v].next[c] =  t.size();
         t.push_back(Vertex(v, ch));
      }
      v = t[v].next[c];
   }
   t[v].leaf = true;
   t[v].S[i] = 1;
}

int go(int v, char ch);

int get_link(int v) {
   if(t[v].link == -1) {
      if(v == 0 or t[v].p == 0)
         t[v].link = 0;
      else
         t[v].link = go(get_link(t[v].p), t[v].c);
   }
   return t[v].link;
}

int go(int v, char ch) {
   int c = ch - 'A';
   if(t[v].go[c] == -1) {
      if(t[v].next[c] != -1)
         t[v].go[c] = t[v].next[c];
      else
         t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
   }
   return t[v].go[c];
}

int32_t main() {
  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

   int caso;
   cin >> caso;
   while(caso--) {
      init();
      string s;
      int n;
      cin >> s >> n;
      bitset<1005> S;
      for(int i = 0; i < n; i++) {
         string a;
```

```
            cin >> a;
            add(a, i);
        }
        int v = 0;
        for(char &c : s) {
            v = go(v, c);
            S |= t[v].S;
        }
        for(int i = 0; i < n; i++)
            cout << (S[i] ? 'y' : 'n') << '\n';
    }

    return 0;
}
```

2.10 Manacher

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 100;

int lps[2*MAX+5];
char s[MAX];

int manacher()
{
    int n = strlen(s);
    string p(2*n+3, '#');
    p[0] = '^';
    for(int i = 0; i < n; i++)
        p[2*(i+1)] = s[i];
    p[2*n+2] = '$';
    int k = 0, r = 0, m = 0;
    int l = p.length();
    for(int i = 1; i < l; i++)
    {
        int o = 2*k - i;
        lps[i] = (r > i) ? min(r-i, lps[o]) : 0;
        while(p[i + 1 + lps[i]] == p[i - 1 - lps[i]])
            lps[i]++;
        if(i + lps[i] > r) k = i, r = i + lps[i];
        m = max(m, lps[i]);
    }
    /*for(int i = 1; i <= 2 * n + 1; i++)
        cout << lps[i] << ' ';
    puts("");*/
    return m;
}

int main()
{
    cin >> s;
    cout << manacher() << '\n';
  return 0;
}
```

2.11 Trie With Vector

```
#include <bits/stdc++.h>
using namespace std;

struct TrieNode
{
  int child[26], size, cnt;
  TrieNode()
  {
    memset(child, 0, sizeof(child));
    size = cnt = 0;
  }
};

vector<TrieNode> trie;

void init()
{
  trie.clear();
  trie.push_back(TrieNode());
}

void add(string s)
{
  int root = 0;
  for(int i = 0; i < (int)s.size(); i++)
  {
    int index = s[i] - 'a';
    if(trie[root].child[index] == 0)
    {
      trie[root].child[index] = trie.size();
      trie.push_back(TrieNode());
    }
    root = trie[root].child[index];
    trie[root].size++;
  }
  trie[root].cnt++;
}

void sub(string s)
{
  int root = 0;
  for(int i = 0; i < (int)s.size(); i++)
  {
    int index = s[i] - 'a';
    root = trie[root].child[index];
    trie[root].size--;
  }
  trie[root].cnt--;
}

int query(string s)
{
  int root = 0;
  for(int i = 0; i < (int)s.size(); i++)
  {
    int index = s[i] - 'a';
    if(!trie[trie[root].child[index]].size)
      return false;
    root = trie[root].child[index];
  }
  return trie[root].cnt;
```

```
}

int main()
{
  string s;
  int o;
  init();
  while(cin >> o >> s)
  {
    if(o == 1) add(s);
    else if(o == 2) sub(s);
    else cout << query(s) << '\n';
  }

  return 0;
}
```

2.12 KMP

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e6;

int n, m;
// n eh o tamanho do texto e m eh o
// tamanho do padrao
int arr[MAX];
// array que guarda o tamanho do maior
// prefixo proprio que tambem eh sufixo
string t, p; // t eh o texto e p eh o padrao

void build() // kMP Preprocess
{
    int i = 0, j = 1;
    while(j < m)
    {
        if(p[i] == p[j])
            arr[j] = ++i;
        else
        {
            i = 0;
            if(p[i] == p[j])
                arr[j] = ++i;
        }
        j++;
    }
}

int matching() // KMP search
{
    int i = 0, j = 0;
    while(j < n)
    {
        if(p[i] == t[j]) i++, j++;
        else if(i) i = arr[i - 1];
        else j++;
        if(i == m)
            return j - m;
    // a substring P inicia na posicao j - m em T
    }
    return -1; // P nao eh substring de T
}

int main()
{
    cin >> t >> p;
    n = (int)t.size();
    m = (int)p.size();
    build();
    cout << matching() << '\n';

    return 0;
}
```

2.13 Trie

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e6;

int trie[MAX][26], cnt[MAX], tsz = 1;
bool leaf[MAX];

void insert(string s)
{
    int cur = 1;
    cnt[cur]++;
    for(int i = 0; i < s.size(); i++)
    {
        int a = s[i] - 'a';
        if(!trie[cur][a]) trie[cur][a] = ++tsz;
        cur = trie[cur][a];
        cnt[cur]++;
    }
    leaf[cur] = true;
}

bool find(string s)
{
    int cur = 1;
    for(int i = 0; i < s.size(); i++)
    {
        int a = s[i] - 'a';
        if(!trie[cur][a] or !cnt[cur])
            return false;
        cur = trie[cur][a];
    }
    return leaf[cur] and cnt[cur];
}

int remove(string s)
{
    int cur = 1;
    for(int i = 0; i < s.size(); i++)
    {
        int a = s[i] - 'a';
        cnt[cur]--;
        cur = trie[cur][a];
    }
    leaf[cur] = false;
```

```
    cnt[cur]--;
}

int main()
{
    string s;
    int n, o;
    while(cin >> o >> s)
    {
        if(o == 1)
            cout << (find(s) ? "found\n" : "not found\n");
        else if(o == 2)
            insert(s);
        else
            remove(s);
    }

  return 0;
}
```

3 Miscellaneous

3.1 Maximum Subarray Xor

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e4 + 10;
const int OO = 0x3f3f3f3f;
const double EPS = 1e-9;

#define bug(x) cout << #x << " = " << x << '\n'
#define FOR(i, a, n) for(int i = a; i < n; i++)
#define REP(i, n) FOR(i, 0, n)
#define fi first
#define se second
#define pb push_back
#define mt make_tuple
#define all(vetor) vetor.begin(), vetor.end()
#define X real()
#define Y imag()
//#define gc getchar_unlocked

typedef long long ll;
typedef long double ld;
typedef pair<ll, ll> ii;
typedef pair<int, ii> iii;
typedef complex<ll> Pll;
typedef complex<ld> Pld;

struct TrieNode
{
    int value;
    TrieNode *children[2];
    TrieNode()
    {
        value = 0;
        children[0] = children[1] = nullptr;
    }
};

void insert(TrieNode *root, int n)
{
    TrieNode *aux = root;
    for(int i = 31; i >= 0; i--)
    {
        bool b = (n & (1 << i));
        if(aux->children[b] == nullptr)
            aux->children[b] = new TrieNode();
        aux = aux->children[b];
    }
    aux->value = n;
}

int query(TrieNode *root, int n)
{
    TrieNode *aux = root;
    for(int i = 31; i >= 0; i--)
    {
        bool b = (n & (1 << i));
        if(aux->children[1-b] != nullptr)
            aux = aux->children[1-b];
        else
            aux = aux->children[b];
    }
    return n ^ aux->value;
}

void maxSubArrayXor(int *arr, int n)
{
    TrieNode *root = new TrieNode();
    insert(root, 0);
    int px = 0, ans = INT_MIN, r = -1;
    for(int i = 0; i < n; i++)
    {
        px = px ^ arr[i];
        insert(root, px);
        int num = max(ans, query(root, px));
        if(num > ans)
            ans = num, r = i;
    }
    int l = r, xo = 0;
    for(; l >= 0; l--)
    {
        xo ^= arr[l];
        if(xo == ans)
            break;
    }
    cout << "O Xor maximo eh: " << ans << '\n';
    while(l <= r)
        cout << arr[l++] << ' ';
    cout << '\n';
}

int main()
{
    int arr[MAX], n;

    cin >> n;
```

```
    REP(i, n) cin >> arr[i];
    maxSubArrayXor(arr, n);

  return 0;
}
```

3.2 Rectangles Union Area

```
#include <bits/stdc++.h>
using namespace std;
#define ii pair<int, int>
#define fi first
#define se second

struct Event
{
    int x1, x2, y, t;
    Event(int _x1, int _x2, int _y, int _t)
    {
        x1 = _x1, x2 = _x2, y = _y, t = _t;
    }
    Event(){}
};

ii tree[500800];
int lazy[500800];

int n;
vector<pair<ii, ii>> segments, rect;
int X1, Y1, X2, Y2, P;

ii calc(ii a, ii b)
{
  if(a.fi > b.fi) return b;
  else if(a.fi < b.fi) return a;
  return {a.fi, a.se + b.se};
}

void build(int node, int start, int end)
{
  if(start == end)
    tree[node] = {0, 1}, lazy[node] = 0;
  else
  {
    int mid = (start + end) / 2;
    build(2 * node, start, mid);
    build(2 * node + 1, mid + 1, end);
    tree[node] = calc(tree[2 * node], tree[2 * node + 1]);
    lazy[node] = 0;
  }
}

void push(int node, int start, int end)
{
  tree[node].fi += lazy[node];
  if(start != end)
  {
    lazy[2 * node] += lazy[node];
    lazy[2 * node + 1] += lazy[node];
  }
  lazy[node] = 0;
}

void update(int node, int start, int end, int l, int r, int v)
{
  if(lazy[node]) push(node, start, end);
  if(start > r or end < l) return;
  if(l <= start and end <= r)
  {
    lazy[node] += v;
    push(node, start, end);
    return;
  }
  int mid = (start + end) / 2;
  update(2 * node, start, mid, l, r, v);
  update(2 * node + 1, mid + 1, end, l, r, v);
  tree[node] = calc(tree[2 * node], tree[2 * node + 1]);
}

int query(int node, int start, int end)
{
  if(lazy[node]) push(node, start, end);
  return end - start + 1 - tree[node].se;
}

void mount(int r)
{
  rect.clear();
  for(auto &it : segments)
  {
    int x1 = max(min(it.fi.fi, it.se.fi) - r, X1);
    int y1 = max(min(it.fi.se, it.se.se) - r, Y1);
    int x2 = min(max(it.fi.fi, it.se.fi) + r, X2);
    int y2 = min(max(it.fi.se, it.se.se) + r, Y2);
    x2--;
    rect.push_back({{x1, y1}, {x2, y2}});
  }
}

bool cmp(Event a, Event b)
{
    if(a.y != b.y) return a.y < b.y;
    return a.t > b.t;
}

long long area(int r)
{
  mount(r);
  vector<Event> eve;
  for(auto &it : rect)
  {
    eve.emplace_back(it.fi.fi, it.se.fi, it.fi.se, 1);
        eve.emplace_back(it.fi.fi, it.se.fi, it.se.se, -1);
  }
  sort(eve.begin(), eve.end(), cmp);
  build(1, 0, 100001);
  long long Y = 0, ans = 0;
    for(int i = 0; i < eve.size(); i++)
    {
        long long s = query(1, 0, 100001);
        long long aux = s * 1LL * (eve[i].y - Y);
```

```
        update(1, 0, 100001, eve[i].x1, eve[i].x2, eve[i].t);
        Y = eve[i].y;
        ans += aux;
    }
    return ans;
}

int32_t main()
{
  scanf(" %d", &n);
  for(int i = 0; i < n; i++)
  {
    int x1, y1, x2, y2;
    scanf(" %d %d %d %d", &x1, &y1, &x2, &y2);
    segments.push_back({{x1, y1}, {x2, y2}});
  }
  scanf(" %d %d %d %d %d", &P, &X1, &Y1, &X2, &Y2);

  long long tot = (X2 - X1) * 1LL * (Y2 - Y1);

  int b = 0, e = 100000, ans = 0;
  while(b <= e)
  {
    int mid = (b + e) / 2;
    long long A = area(mid);
    if(P * 1LL * tot <= 100LL * A) ans = mid, e = mid - 1;
    else b = mid + 1;
  }
  cout << ans << '\n';

    return 0;
}
```

3.3 Count Divisors

```
#include <bits/stdc++.h>
using namespace std;

long long add(long long a, long long b, long long c)
{
  long long ans = (a + b) % c;
    if(ans < 0) ans += c;
    return ans;
}

long long mulmod(long long a, long long b, long long c)
{
  long long ans = 0;
  while(b)
  {
    if(b & 1) ans = add(ans, a, c);
    a = add(a, a, c);
    b /= 2;
  }
  return ans;
}

long long fexp(long long a, long long b, long long c)
{
  long long ans = 1;
  while(b)
  {
    if(b & 1) ans = mulmod(ans, a, c);
    a = mulmod(a, a, c);
    b /= 2;
  }
  return ans;
}

bool miller(long long a, long long n)
{
    if (a >= n) return true;
    long long s = 0, d = n - 1;
    while(d%2 == 0 and d) d >>= 1, s++;
    long long x = fexp(a, d, n);
    if(x == 1 or x == n - 1) return true;
    for(int r = 0; r < s; r++, x = mulmod(x, x, n))
    {
        if (x == 1) return false;
        if (x == n-1) return true;
    }
    return false;
}

bool isprime(long long n)
{
  if(n < 2) return false;
    int base[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};
    for(int i = 0; i < 12; i++)
        if(!miller(base[i], n))
            return false;
    return true;
}

vector<int> prime;
bitset<10000000> composite;

void sieve()
{
  for(int i = 2; i < 10000000; i++)
    if(!composite[i])
    {
      prime.push_back(i);
      for(int j = 2; i * j < 10000000; j++)
        composite[i * j] = 1;
    }
}

long long countDivisors(long long n)
{
  int idx = 1;
  long long ans = 1, p = prime[0];
  while(p * p * p <= n)
  {
    int cnt = 1;
    while(n % p == 0)
      n /= p, cnt++;
    ans *= cnt;
    p = prime[idx++];
  }
  if(n == 1) return ans;
```

```
  if(isprime(n)) ans *= 2;
  else
  {
    long long sq = sqrt(n);
    if(sq * sq == n)
      ans *= 3;
    else if(n != 1)
      ans *= 4;
  }
  return ans;
}

int main()
{
    long long n;
    cin >> n;
    sieve();
  cout << countDivisors(n) << '\n';

    return 0;
}
```

3.4 Counting Different Elements In A Path With Mo

```
//COT - Count on a tree (SPOJ)
//Em cada vertice existe um valor
//A resposta para uma query eh quantos valores
//distintos existem no caminho de u a v
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e6;

typedef long long ll;

struct Query
{
    int x, y, l, r, lc, res;
};

int n, q, max_log, tempo, blk, ans;
vector<int> G[MAX];
int value[MAX], pos[MAX], anc[MAX][25], depth[MAX];
int tl[MAX], ST[MAX], EN[MAX], freq[MAX], node[MAX];
Query Q[MAX];
ll arr[MAX];

void dfs(int v, int p, int d)
{
    anc[v][0] = p;
    depth[v] = d;
    tl[tempo] = v;
    ST[v] = tempo++;
    if(d) max_log = max(max_log, (int)log2(d));
    for(const int &u : G[v])
        if(u != p)
            dfs(u, v, d + 1);
    tl[tempo] = v;
    EN[v] = tempo++;
}

int walk(int v, int k)
{
    while(k) v = anc[v][(int)log2(k&-k)], k -= k&-k;
    return v;
}

int lca(int u, int v)
{
    if(depth[u] > depth[v]) u = walk(u, depth[u] - depth[v]);
    if(depth[u] < depth[v]) v = walk(v, depth[v] - depth[u]);
    if(u == v) return u;
    for(int i = max_log; i >= 0; i--)
        if(anc[u][i] != anc[v][i])
        {
            u = anc[u][i];
            v = anc[v][i];
        }
    return anc[u][0];
}

void build()
{
    memset(anc, -1, sizeof(anc));
    dfs(0, -1, 0);
    for(int j = 1; j <= max_log; j++)
        for(int i = 0; i < n; i++)
            if(anc[i][j-1] != -1)
                anc[i][j] = anc[anc[i][j-1]][j-1];
}

inline void mo(int i)
{
    int u = tl[i];
    if(node[u] and --freq[value[u]] == 0) ans--;
    else if(!node[u] and ++freq[value[u]] == 1) ans++;
    node[u] ^= 1;
}

bool compare(int a, int b)
{
    if(Q[a].l/blk != Q[b].l/blk)
        return Q[a].l < Q[b].l;
    return Q[a].r > Q[b].r;
}

int main()
{
    scanf("%d %d", &n, &q);
    for(int i = 0; i < n; i++)//values
        scanf("%d", &arr[i]), pos[i] = i;

    sort(pos, pos + n, [](ll a, ll b){return arr[a] < arr[b];});
    for(int i = 0, j = 1; i < n; i++)
        if(!i)
            value[pos[i]] = j++;
        else if(arr[pos[i]] != arr[pos[i-1]])
            value[pos[i]] = j++;
        else value[pos[i]] = value[pos[i-1]];

    for(int i = 0; i < n-1; i++)
    {
```

```
        int u, v;
        scanf("%d %d", &u, &v); u--; v--;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    build();

    for(int i = 0; i < q; i++)
    {
        int u, v;
        scanf("%d %d", &u, &v); u--; v--;
        if(ST[u] > ST[v]) swap(u, v);
        Q[i].lc = lca(u, v);
        Q[i].x = u, Q[i].y = v;
        if(u == Q[i].lc)
            Q[i].l = ST[u], Q[i].r = ST[v];
        else
            Q[i].l = EN[u], Q[i].r = ST[v];
        pos[i] = i;
    }

    blk = sqrt(tempo);
    sort(pos, pos + q, compare);

    int curL = 0, curR = 0;
    for(int i = 0; i < q; i++)
    {
        int L = Q[pos[i]].l, R = Q[pos[i]].r;
        while(curL < L)
            mo(curL++);
        while(curL > L)
            mo(--curL);
        while(curR < R + 1)
            mo(curR++);
        while(curR > R + 1)
            mo(--curR);

        if(Q[pos[i]].x != Q[pos[i]].lc)
            mo(ST[Q[pos[i]].lc]);
        Q[pos[i]].res = ans;
        if(Q[pos[i]].x != Q[pos[i]].lc)
            mo(ST[Q[pos[i]].lc]);
    }
    for(int i = 0; i < q; i++)
        printf("%d\n", Q[i].res);

    return 0;
}
```

3.5 Gen Random Tree

```
#include <bits/stdc++.h>
using namespace std;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

int rand(int a, int b) {
    return a + rng() % (b - a + 1);
}

int main() {
    int n = rand(4, 20);
  cout << n << endl;
    vector<pair<int,int>> edges;
    for(int i = 2; i <= n; i++)
        edges.emplace_back(rand(1, i - 1), i);

    // re-naming vertices
    vector<int> perm(n + 1); // re-naming vertices
    for(int i = 1; i <= n; ++i)
        perm[i] = i;

    // random order of labels
  shuffle(perm.begin() + 1, perm.end(), rng);
    // random order of edges
  shuffle(edges.begin(), edges.end(), rng);

    for(auto [u, v] : edges) {
    // random order of two vertices
        if(rng() % 2) swap(u, v);
        cout << perm[u] << ' ' << perm[v] << endl;
    }

  return 0;
}
```

3.6 Index Compression

```
#include <bits/stdc++.h>
const int MAX = 1e6 + 10;
using namespace std;

int n, arr[MAX], pos[MAX], newArr[MAX], realValue[MAX];

int main()
{
    cin >> n;
    for(int i = 0; i < n; i++)
        cin >> arr[i], pos[i] = i;
    sort(pos, pos + n, [](int i, int j){ return arr[i] < arr[j]; });
    int id = 1;
    for(int i = 0; i < n; i++)
    {
    if(!i) newArr[pos[i]] = id, realValue[id] = arr[pos[i]];
    else if(arr[pos[i-1]] == arr[pos[i]]) newArr[pos[i]] = newArr[pos[
        i-1]];
    else newArr[pos[i]] = ++id, realValue[id] = arr[pos[i]];
    }
    for(int i = 0; i < n; i++)
        cout << arr[i] << ' ' << newArr[i] <<
           ' ' << realValue[newArr[i]] << '\n';


    return 0;
}
```

3.7 Odd Rectangles Area

```
/*
You are given several axis-aligned rectangles. Compute the sum
of the area of the regions that are covered by an odd number of
    rectangles.

input: The first line of input contains a single integer n (1<=n
    <=10^5), representing
the number of rectangles. Each of the next n lines contains four space
    -separated
integers x1, y1, x2, and y2, each between 0 and 109, describing the
    coordinates of
a rectangle.

Output: Print, on one line, the total area covered by an odd number of
     rectangles
as an exact integer.
*/

#include <bits/stdc++.h>
using namespace std;

struct Event
{
    int x1, x2, y, t;
    Event(int _x1, int _x2, int _y, int _t)
    {
        x1 = _x1, x2 = _x2, y = _y, t = _t;
    }
    Event(){}
};

struct Node
{
  int l, r, value;
};

int n;
vector<Node> tree;
vector<int> lazy;
vector<Event> arr;

int init()
{
  tree.clear();
  lazy.clear();
  tree.emplace_back();
  lazy.push_back(0);
}

void createL(int node)
{
  tree[node].l = tree.size();
  tree.emplace_back();
  lazy.push_back(0);
}

void createR(int node)
{
  tree[node].r = tree.size();
  tree.emplace_back();
  lazy.push_back(0);
}

void calc(int node)
{
  tree[node].value = 0;
  if(tree[node].l) tree[node].value += tree[tree[node].l].value;
  if(tree[node].r) tree[node].value += tree[tree[node].r].value;
}

void push(int node, int start, int end)
{
  tree[node].value = (end - start + 1) - tree[node].value;
  if(start != end)
  {
    if(tree[node].l == 0) createL(node);
    if(tree[node].r == 0) createR(node);
    lazy[tree[node].l] ^= 1;
    lazy[tree[node].r] ^= 1;
  }
  lazy[node] = 0;
}

void update(int node, int start, int end, int l, int r)
{
  if(lazy[node])
    push(node, start, end);
  if(start > r or l > end) return;
  if(l <= start and end <= r)
  {
    push(node, start, end);
  }
  else
  {
    int mid = (start + end) / 2;
    if(tree[node].l == 0) createL(node);
    update(tree[node].l, start, mid, l, r);
    if(tree[node].r == 0) createR(node);
    update(tree[node].r, mid + 1, end, l, r);
    calc(node);
  }
}

int query(int node, int start, int end, int l, int r)
{
  if(lazy[node])
    push(node, start, end);

  if(start > r or l > end) return 0;

  if(l <= start and end <= r) return tree[node].value;

  int mid = (start + end) / 2, q1 = 0, q2 = 0;
  if(tree[node].l) q1 = query(tree[node].l, start, mid, l, r);
  if(tree[node].r) q2 = query(tree[node].r, mid + 1, end, l, r);
  return q1 + q2;
}

bool cmp(Event a, Event b)
{
    if(a.y != b.y) return a.y < b.y;
    return a.t < b.t;
```

```
}

int32_t main()
{
    scanf(" %d", &n);
    for(int i = 0; i < n; i++)
    {
        int x1, y1, x2, y2;
        scanf(" %d %d %d %d", &x1, &y1, &x2, &y2);
        int xmi = min(x1, x2);
        int ymi = min(y1, y2);
        int xma = max(x1, x2);
        int yma = max(y1, y2);
        xma--;
        //yma--;
        arr.emplace_back(xmi, xma, ymi, -1);
        arr.emplace_back(xmi, xma, yma, 1);
    }
    sort(arr.begin(), arr.end(), cmp);
    long long Y = 0, ans = 0;
    init();
    for(int i = 0; i < arr.size(); i++)
    {
        //cout << "op " << arr[i].t << '\n';
        long long s = query(0, 0, 1000000008, 0, 1000000007);
        long long aux = s * 1LL * (arr[i].y - Y);
        //cout << "this        " << aux << ' ' << arr[i].y1 << ' ' <<
            Y << ' ' << s << '\n';
        update(0, 0, 1000000008, arr[i].x1, arr[i].x2);
        Y = arr[i].y;
        ans += aux;
    }
    cout << ans << '\n';

    return 0;
}
```

3.8 Karp Rabin

```
/*
    Funcao de Hash:

    Tome uma string S[0 ... n-1] e deois inteiros A e B

    (S[0]*A^n-1 + S[1]*A^n-2 + S[2]*A^n-3 + S[3]*A^n-4
       ... S[N-1]*A^0) mod B
*/

#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

typedef long long ll;

ll A = 911382323, B = 972663749;
ll h1[MAX], h2[MAX], p[MAX];
int arr[MAX];
string s;

ll buildH(ll *H, int k)
{
    if(k == 0)
        return H[0] = s[0];
    return H[k] = (buildH(H, k - 1)*A + s[k]) % B;
}

ll buildP(int k)
{
    if(k == 0)
        return p[0] = 1;
    return p[k] = (buildP(k - 1)*A) % B;
}

ll vhash(ll *H, int a, int b)
{
    if(a == 0)
        return H[b];
    ll ans = (H[b] - H[a - 1] * p[b - a + 1]) % B;
    if(ans < 0)
        ans += B;
    return ans;
}

bool slidingWindow(int k)
{
    if(k < 0 or k > s.size()) return false;
    for(int i = 0; i + k - 1 < s.size(); i++)
        if(vhash(h1, i, i + k - 1) == vhash(h2, s.size()
           - (i + k - 1) - 1, s.size() - (i + k - 1) - 2 + k))
            return true; // A substring [i, i + k - 1] eh palindromo
    return false;
}

int buscab()
{
    int tam = 0;
    for(int i = 0; i < s.size(); i++)
        arr[i] = 2*i + 1, tam++;
    int b = 0, e = tam, m, ans = 0;
    while(b <= e)
    {
        m = (b + e) / 2;
        slidingWindow(arr[m]) ? b = m + 1, ans = arr[m] : e = m - 1;
    }
    tam = 0;
    for(int i = 0; i < s.size(); i++)
        arr[i] = 2*i, tam++;
    b = 0, e = tam;
    while(b <= e)
    {
        m = (b + e) / 2;
        slidingWindow(arr[m]) ? b = m + 1,
          ans = max(arr[m], ans) : e = m - 1;
    }
    return ans;
}

int main()
{
    cin >> s;
    buildH(h1, s.size()-1);
```

```
    reverse(s.begin(), s.end());
    buildH(h2, s.size()-1);
    buildP(s.size()-1);
    reverse(s.begin(), s.end());
    cout << buscab() << '\n';

    return 0;
}
```

3.9 Quick Sort And Select

```
#include <bits/stdc++.h>
using namespace std;

int n, arr[10000];

int quickselect(int l, int r, int k)
{
    int j = l - 1;
    for(int i = l; i < r; i++)
        if(arr[i] <= arr[r])
            swap(arr[++j], arr[i]);
    swap(arr[j+1], arr[r]);
    if(j+1 < k) return quickselect(j+2, r, k);
    else if(j+1 > k) return quickselect(l, j, k);
    return arr[j+1];
}

void quicksort(int l, int r)
{
    int j = l - 1;
    for(int i = l; i < r; i++)
        if(arr[i] <= arr[r])
            swap(arr[++j], arr[i]);
    swap(arr[j+1], arr[r]);
    if(l < j)
        quicksort(l, j);
    if(j+2 < r)
        quicksort(j+2, r);
}

int main()
{
    int k;
    cin >> n >> k;
    for(int i = 0; i < n; i++) cin >> arr[i];
    cout << quickselect(0, n-1, k-1) << '\n';

    return 0;
}
```

3.10 Histogram

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
int n, vet[1000000];

ll histogram()
{
  stack<ll> s;
  ll ans = 0, tp, cur;
  int i = 0;
  while(i < n or !s.empty())
  {
    if(i < n and (s.empty() or vet[s.top()] <= vet[i]))
      s.push(i++);
    else
    {
      tp = s.top();
      s.pop();
      cur = vet[tp] * (s.empty() ? i : i - s.top() - 1);
      if(ans < cur)
        ans = cur;
    }
  }
  return ans;
}

int main()
{
  while(cin >> n and n)
  {
    for(int i = 0; i < n; i++)
      cin >> vet[i];
    cout << histogram() << '\n';
  }
    return 0;
}
```

3.11 Divide Conquer Optimization

```
#include <bits/stdc++.h>
using namespace std;

#define maxn 20005
#define maxnlog 22
const long long OO = 0x3f3f3f3f3f3f3f3f;

struct SparseTableDS
{
  int Sparse_Table[maxnlog][maxn];
  bool maxi;
  int n;

  void build()
  {
    for(int i = 1; (1 << i) <= n; i++)
      for(int j = 0; j + (1 << i) <= n; j++)
      {
        if(maxi)
          Sparse_Table[i][j] =  max(Sparse_Table[i-1][j],
            Sparse_Table[i-1][j+(1 << (i-1))]);
        else
          Sparse_Table[i][j] =  min(Sparse_Table[i-1][j],
              Sparse_Table[i-1][j+(1 << (i-1))]);
      }
```

```
  }

  int query(int i, int j)
  {
    int sz = log2(j-i+1);
    if(maxi)
      return max(Sparse_Table[sz][i],Sparse_Table[sz][j+1-(1 << sz)]);
    return min(Sparse_Table[sz][i],Sparse_Table[sz][j+1-(1 << sz)]);
  }


  void init(bool fl, vector<int> &arr)
  {
    maxi = fl;
    if(!maxi) memset(Sparse_Table, 63, sizeof(Sparse_Table));
    n = arr.size();
    for(int i = 0; i < n; i++)
      Sparse_Table[0][i] = arr[i];
    build();
  }
};

int n, k;
SparseTableDS maxi, mini;

long long dp_before[maxn];
long long dp_cur[maxn];

int get(int l, int r)
{
  int a = maxi.query(l, r);
  int b = mini.query(l, r);
  return abs(a - b);
}

void compute(int l, int r, int optl, int optr)
{
  if(l > r) return;

  int mid = (l + r) >> 1;
  int best = 0;
  int opt = optl;

  for(int k = optl; k < min(mid, optr + 1); k++)
    if(best < dp_before[k] + get(k + 1, mid))
    {
      best = dp_before[k] + get(k + 1, mid);
      opt = k;
    }
  dp_cur[mid] = best;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}

int32_t main()
{
  cin >> n >> k;
  vector<int> arr(n);
  for(int &w : arr) scanf(" %d", &w);
  maxi.init(true, arr);
  mini.init(false, arr);

  for(int i = 0; i < n; i++)
    dp_cur[i] = get(0, i);

  for(int i = 2; i <= k; i++)
  {
    for(int j = 0; j < n; j++)
    {
      dp_before[j] = dp_cur[j];
      dp_cur[j] = 0;
    }
    compute(i - 2, n - 1, i - 2, n - 1);

  }

  cout << dp_cur[n - 1] << endl;

  return 0;
}
```

3.12 Inclusion Exclusion

```
/*
 contar a quantidade de numeros na range [1, b]
 que sao multiplos de pelo menos um numero na range [1, a]
*/

#include <bits/stdc++.h>
using namespace std;
#define bug(x) cout << #x << "  >>>>>>>  " << x << '\n'
#define _ << " , " <<
#define int long long
#define Max(a, b) (a > b ? a : b)
#define Min(a, b) (a < b ? a : b)
#define ii pair<int, int>
#define fi first
#define se second
#define SZ(v) (int)v.size()
#define UNTIL(t) while (clock() < (t) * CLOCKS_PER_SEC)
const long long MAX = (long long)1e15; //2 * 10^5
const int MOD = 1000000007; //10^9 + 7
const int OO = 0x3f3f3f3f;//3f3f3f3f;
const double EPS = 1e-9;  //10^-9
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

vector<int> prime = {3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43,
    47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
    113, 127};
vector<vector<ii>> lista;

void add(int id)
{
  vector<ii> aux;
  for(int k = 0; k < id; k++)
  {
    int i = SZ(lista[k]) - 1;
    while(i >= 0 and MAX / lista[k][i].fi / prime[id] == 0) i--;
    for(int j = 0; j <= i; j++)
```

```
        aux.push_back({lista[k][j].fi * prime[id], !lista[k][j].se});
    }
    aux.push_back({prime[id], 1});
    sort(aux.begin(), aux.end());
    lista.push_back(aux);
}

int32_t main()
{
    for(int i = 0; i < SZ(prime); i++)
        add(i);

    int t;

    scanf(" %lld", &t);

    while(t--)
    {
        int a, b;

        scanf(" %lld %lld", &a, &b);

        int ans = b / 2;
        int cnt_p = 0;

        for(int &w : prime) cnt_p += (w <= a);

        for(int i = 0; i < cnt_p; i++)
        {
            for(int j = 0; j < SZ(lista[i]); j++)
            {
                if(lista[i][j].fi > b) break;
                if(lista[i][j].se) ans += (b / lista[i][j].fi + 1) / 2;
                else ans -= (b / lista[i][j].fi + 1) / 2;
            }
        }
        printf("%lld\n", ans);
    }

    return 0;
}
```

3.13 Custom Hash Function Unordered Map Or Set

```
#include <bits/stdc++.h>
using namespace std;

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM = chrono::steady_clock::now
            ().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};

//teste
const int N = 2e5;

void insert_numbers(long long x)
{
    clock_t begin = clock();
    unordered_map<long long, int, custom_hash> numbers;

    for (int i = 1; i <= N; i++)
        numbers[i * x] = i;

    long long sum = 0;

    for (auto &entry : numbers)
        sum += (entry.first / x) * entry.second;

    printf("x = %lld: %.3lf seconds, sum = %lld\n", x, (double) (clock
        () - begin) / CLOCKS_PER_SEC, sum);
}

int main()
{
    insert_numbers(107897);
    insert_numbers(126271);

  return 0;
}
```

3.14 Square Root Decomposition

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

int n, blk_sz;
int arr[MAX], block[MAX];

void update(int idx, int val)
{
    int blockNumber = idx/blk_sz;
    block[blockNumber] += val - arr[idx];
    arr[idx] = val;
}

int query(int l, int r)
{
    int sum = 0;
    while(l < r and l%blk_sz != 0 and l != 0)
        sum += arr[l], l++;
    while(l+blk_sz <= r)
        sum += block[l/blk_sz], l += blk_sz;
    while(l <= r)
        sum += arr[l], l++;
    return sum;
}

void build()
```

```
{
    int blk_idx = -1;
    blk_sz = sqrt(n);
    for(int i = 0; i < n; i++)
    {
        if(i%blk_sz == 0)
            blk_idx++;
        block[blk_idx] += arr[i];
    }
}

int main()
{
    cin >> n;
    for(int i = 0; i < n; i++)
        cin >> arr[i];
    build();
    int o, l, r;
    while(cin >> o >> l >> r and o)
        if(o == 1)
            update(l-1, r);
        else
            cout << query(l-1, r-1) << '\n';

    return 0;
}
```

3.15 Big Num Product

```
string mul(string a, string b)
{
  while(a.size() > b.size()) b = "0" + b;
  while(a.size() < b.size()) a = "0" + a;
  a = "00" + a;
  b = "00" + b;
  int ans = 0, n = a.size(), carry = 0;
  vector<int> num(2 * n, 0);
  for(int i = n - 1; i >= 0; i--)
    for(int j = n - 1; j >= 0; j--)
    {
      int di = a[i] - '0';
      int dj = b[j] - '0';
      num[i + j + 1] += (di * dj) + carry;
      carry = (num[i + j + 1] / 10);
      num[i + j + 1] %= 10;
    }
  string r;
  for(int i = 0, fl = 0; i < 2 * n; i++)
  {
    if(num[i]) fl = 1;
    if(fl) r.push_back(num[i] + '0');
  }
  return r;
```

3.16 Fence Problem With Max Flow

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e4;
const int OO = 0x3f3f3f3f;
int SOURCE, SINK;

struct edge
{
    int v, f, c;
    edge(){}
    edge(int _v, int _f, int _c)
    {
        v = _v, f = _f, c = _c;
    }
};

vector<edge> edges;
vector<int> G[MAX];
int dist[MAX], work[MAX];

void add_edge(int u, int v, int cp, int rc){
  edges.push_back(edge(v, 0, cp));
  G[u].push_back(edges.size()-1);
  edges.push_back(edge(u, 0, rc));
  G[v].push_back(edges.size()-1);
}

bool bfs(int s, int t)
{
    memset(dist, -1, sizeof(dist));
    dist[s] = 0;
    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        for(int e : G[u])
            if(dist[edges[e].v] == -1 and edges[e].c-edges[e].f > 0)
            {
                q.push(edges[e].v);
                dist[edges[e].v] = dist[u] + 1;
            }
    }
    return dist[t] != -1;
}

int dfs(int s, int t, int f)
{
    if(s == t) return f;
    for(int &i = work[s]; i < G[s].size(); i++)
    {
      int e = G[s][i];
        if(dist[edges[e].v] == dist[s] + 1 and edges[e].c-edges[e].f >
             0)
            if(int a = dfs(edges[e].v, t, min(f, edges[e].c-edges[e].f
                )))
            {
                edges[e].f += a;
                edges[e^1].f -= a;
                return a;
            }
    }
```

```
    return 0;
}

int MaxFlow(int s, int t)
{
    int mf = 0;
    while(bfs(s, t))
  {
      memset(work, 0, sizeof(work));
        while(int a = dfs(s, t, OO))
            mf += a;
    }
    return mf;
}

int n, m, a, b;
int dx[] = {1, 0, -1, 0};
int dy[] = {0, -1, 0, 1};
char ANS[60][60];
bool cor[MAX];

bool check(int x, int y)
{
  return x >= 0 and x < n and y >= 0 and y < m;
}

int vertexIn(int i, int j)
{
  return i * m + j;
}

int vertexOut(int i, int j)
{
  return i * m + j + n * m + 1;
}

void mountANS(int v)
{
  cor[v] = true;
  for(int &e : G[v])
  {
    if(cor[edges[e].v]) continue;
    if(edges[e].c - edges[e].f > 0)
      mountANS(edges[e].v);
  }
}

int main()
{
  memset(ANS, '.', sizeof(ANS));
  cin >> n >> m >> a >> b; a--; b--;
  SOURCE = 2 * n * m + 2;
  SINK = 2 * n * m + 3;
  for(int i = 0; i < n; i++)
    for(int j = 0; j < m; j++)
    {
      int cost;
      cin >> cost;
      if(a == i and b == j) cost = OO;
      add_edge(vertexIn(i, j), vertexOut(i, j), cost, 0);
      if(cost == OO) add_edge(vertexOut(i, j), SINK, OO, 0);
      for(int k = 0; k < 4; k++)
      {
        int x = i + dx[k], y = j + dy[k];
        if(check(x, y))
          add_edge(vertexOut(i, j),
            vertexIn(x, y), OO, 0);
      }
      if(!i or !j or i == n - 1 or j == m - 1)
        add_edge(SOURCE, vertexIn(i, j), OO, 0);
    }
  cout << MaxFlow(SOURCE, SINK) << '\n';
  mountANS(SOURCE);
  for(int i = 0; i < n * m; i++)
    for(int &e : G[i])
      if(!(e & 1) and cor[i] and !cor[edges[e].v])
        ANS[i / m][i % m] = 'X';
  for(int i = 0; i < n; i++)
  {
    for(int j = 0; j < m; j++)
      cout << ANS[i][j];
    puts("");
  }
    return 0;
}
```

3.17 Mo

```
//com hash no braco

#include <bits/stdc++.h>
using namespace std;
const int MAX = 3 * 1e5;

struct Query
{
    int l, r, qt;
    vector<int> el;
};

Query q[MAX];
int n, m, blk, ans;
int arr[MAX], freq[MAX], qtd[MAX], pos[MAX];
vector<vector<int>> tab;

void add(int i)
{
    if(freq[arr[i]])
        qtd[freq[arr[i]]]--;
    freq[arr[i]]++;
    qtd[freq[arr[i]]]++;
    if(ans <= freq[arr[i]])
    {
        ans = freq[arr[i]];
        tab[ans].push_back(arr[i]);
    }
}

void sub(int i)
{
    qtd[freq[arr[i]]]--;
```

```
    if(!qtd[ans])
    {
        for(int j = 0; j < tab[ans].size(); j++)
            if(tab[ans][j] == arr[i])
            {
                tab[ans].erase(tab[ans].begin() + j);
                break;
            }
        ans--;
        tab[ans].push_back(arr[i]);
    }
    freq[arr[i]]--;
    if(freq[arr[i]])
        qtd[freq[arr[i]]]++;
}

bool compare(int a, int b)
{
    if(q[a].l/blk != q[b].l/blk)
        return q[a].l < q[b].l;
    return q[a].r < q[b].r;
}

int main()
{
    while(cin >> n >> m and n)
    {
        blk = sqrt(n);
        for(int i = 0; i < n; i++)
        {
            cin >> arr[i];
            arr[i] += 100000;
        }
        for(int i = 0; i < m; i++)
            cin >> q[i].l >> q[i].r, pos[i] = i;

        sort(pos, pos + m, compare);

        ans = 0;
        int curL = 0, curR = 0;
        int L, R;

        memset(freq, 0, sizeof(freq));
        memset(qtd, 0, sizeof(qtd));
        tab.clear();
        tab.resize(MAX);

        for(int j = 0; j < m; j++)
        {
            L = q[pos[j]].l - 1;
            R = q[pos[j]].r - 1;
            while(curL < L)
                sub(curL++);
            while(curL > L)
                add(--curL);
            while(curR < R + 1)
                add(curR++);
            while(curR > R + 1)
                sub(--curR);
            q[pos[j]].qt = ans;
            q[pos[j]].el = tab[ans];
        }
        for(int j = 0; j < m; j++)
            for(int i = 0; i < q[j].el.size(); i++)
                cout << q[j].el[i]-100000 << '\n';
    }

    return 0;
}

////////////////////////////////////////////
//com unordered_multimap

#include <bits/stdc++.h>
using namespace std;
const int MAX = 3 * 1e5;

struct Query
{
    int l, r, qt, morefrequent;
};

Query q[MAX];
int n, m, blk, ans;
int arr[MAX], freq[MAX], qtd[MAX], pos[MAX];
unordered_multimap<int, int> tab;

void add(int i)
{
    if(freq[arr[i]])
        qtd[freq[arr[i]]]--;
    freq[arr[i]]++;
    qtd[freq[arr[i]]]++;
    if(ans <= freq[arr[i]])
    {
        ans = freq[arr[i]];
        tab.insert({ans, arr[i]});
    }
}

void sub(int i)
{
    qtd[freq[arr[i]]]--;
    if(!qtd[ans])
    {
        int k = 0, sz = tab.bucket(ans);
        auto it = tab.find(ans);
        for(int j = 0; j < sz; j++)
            if(it->second == arr[i])
            {
                tab.erase(it);
                j = sz;
            }
            else it++;

        ans--;
        tab.insert({ans, arr[i]});
    }
    freq[arr[i]]--;
    if(freq[arr[i]])
        qtd[freq[arr[i]]]++;
}
```

```
bool compare(int a, int b)
{
    if(q[a].l/blk != q[b].l/blk)
        return q[a].l < q[b].l;
    return q[a].r < q[b].r;
}

int main()
{
    while(cin >> n >> m and n)
    {
        blk = sqrt(n);
        for(int i = 0; i < n; i++)
        {
            cin >> arr[i];
            arr[i] += 100000;
        }
        for(int i = 0; i < m; i++)
            cin >> q[i].l >> q[i].r, pos[i] = i;

        sort(pos, pos + m, compare);

        ans = 0;
        int curL = 0, curR = 0;
        int L, R;

        memset(freq, 0, sizeof(freq));
        memset(qtd, 0, sizeof(qtd));
        tab.clear();

        for(int j = 0; j < m; j++)
        {
            L = q[pos[j]].l - 1;
            R = q[pos[j]].r - 1;
            while(curL < L)
                sub(curL++);
            while(curL > L)
                add(--curL);
            while(curR < R + 1)
                add(curR++);
            while(curR > R + 1)
                sub(--curR);
            q[pos[j]].qt = ans;
            q[pos[j]].morefrequent = tab.find(ans)->second;
        }
        for(int j = 0; j < m; j++)
                cout << q[j].morefrequent-100000 << '\n';
    }

    return 0;
}

//////////////////////////////////////
Mo em arvore parte 1

#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5 + 10;

struct Query
{
    int l, r, v;
};

int n, q;
int freq[MAX], inicio[MAX], fim[MAX], pos[MAX], value[MAX];
vector<int> tree_linearization, G[MAX];
Query Q[MAX];
int ans, blk;

void TreeLinearization(int v, int p)
{
    inicio[v] = tree_linearization.size();
    tree_linearization.push_back(v);
    for(const int &u : G[v])
        if(u != p)
            TreeLinearization(u, v);
    fim[v] = tree_linearization.size() - 1;
}

void add(int i)
{
    if(!freq[value[tree_linearization[i]]])
        ans++;
    freq[value[tree_linearization[i]]]++;
}

void sub(int i)
{
    freq[value[tree_linearization[i]]]--;
    if(!freq[value[tree_linearization[i]]])
        ans--;
}

bool compare(int a, int b)
{
    if(Q[a].l/blk != Q[b].l/blk)
        return Q[a].l/blk < Q[b].l/blk;
    return Q[a].r < Q[b].r;
}

int main()
{
    int u, v;
    cin >> n >> q;
    for(int i = 0; i < n; i++)
        cin >> value[i];
    for(int i = 0; i < n-1; i++)
    {
        cin >> u >> v;
        u--; v--;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    TreeLinearization(0, -1);
    for(int i = 0; i < q; i++)
    {
        cin >> u;
        u--;
        Q[i].l = inicio[u];
        Q[i].r = fim[u];
```

```
        pos[i] = i;
    }
    blk = sqrt(n);
    sort(pos, pos+q, compare);
    int curL = 0, curR = 0;
    for(int i = 0; i < q; i++)
    {
        int L = Q[pos[i]].l, R = Q[pos[i]].r;
        while(curL < L)
            sub(curL++);
        while(curL > L)
            add(--curL);
        while(curR < R + 1)
            add(curR++);
        while(curR > R + 1)
            sub(--curR);
        Q[pos[i]].v = ans;
    }
    for(int i = 0; i < q; i++)
        cout << Q[i].v << '\n';

    return 0;
}
```

3.18 Knuth Optimization

```
#include <bits/stdc++.h>
using namespace std;

// Knuth Optimization

int pf[6000], n;
int dp[6000][6000];

int sum(int l, int r)
{
  return pf[r] - pf[l - 1];
}

int solve(int l, int r)
{
  if(l > r) return 0;
  if(dp[l][r] != -1) return dp[l][r];
  int ans = (1 << 30);
  for(int i = l; i <= r; i++)
    ans = min(ans, sum(l, r) + solve(l, i - 1) + solve(i + 1, r));
  return dp[l][r] = ans;
}



#define ii pair<int, int>
#define fi first
#define se second

ii DP[6000][6000];

//Point(l, r - 1) <= Point(l, r) <= Point(l + 1, r)
ii knuth(int l, int r)
{
  if(l == r) return {sum(l, r), l};
  if(DP[l][r] != ii(-1, -1)) return DP[l][r];
  int lef = knuth(l, r - 1).se;
  int rig = knuth(l + 1, r).se;
  int point = l, ans = (1 << 30);
  for(int i = lef; i <= rig; i++)
  {
    int cur = sum(l, r);
    if(i - 1 >= l) cur += knuth(l, i - 1).fi;
    if(i + 1 <= r) cur += knuth(i + 1, r).fi;
    if(cur < ans) ans = cur, point = i;
  }
  return DP[l][r] = {ans, point};
}

int main()
{
  memset(dp, -1, sizeof(dp));
  cin >> n;
  //for(int i = 1; i <= n; i++)
    //cin >> pf[i], pf[i] += pf[i - 1];
  //cout << solve(1, n) << endl;

  memset(DP, -1, sizeof(DP));
  cout << knuth(1, n).fi << endl;

  return 0;
}
```

3.19 Count Sort

```
#include <bits/stdc++.h>
using namespace std;

int n, m;
int arr[100];
int cnt[10000];
int aux[100];

void count_sort()
{
    for(int i = 0; i < n; i++)
        cnt[arr[i]]++;
    for(int i = 1; i <= m; i++)
        cnt[i] += cnt[i-1];
    for(int i = 0; i < n; i++)
        aux[--cnt[arr[i]]] = arr[i];
    memcpy(arr, aux, n*sizeof(int));
}

int main()
{
    cin >> n;
    for(int i = 0; i < n; i++)
        cin >> arr[i], m = max(arr[i], m);

    count_sort();

    for(int i = 0; i < n; i++)
      cout << arr[i] << ' '; cout << '\n';
```

```
    return 0;
}
```

3.20 Lontest Substring That Is A Correct Bracket Sequence

```
#include <bits/stdc++.h>
using namespace std;
#define OO 0x3f3f3f3f
#define gc getchar
#define pc putchar
#define offset 1000000

string str;
int Sparse_Table[22][1000002], n;
vector<int> forest[2000002];
int pf[1000002], cnt[1000002];

inline void build()
{
  for(int i = 1; (1 << i) < n; i++)
    for(int j = 0; j + (1 << i) < n; j++)
      Sparse_Table[i][j] = min(Sparse_Table[i-1][j],
        Sparse_Table[i-1][j+(1 << (i-1))]);
}

inline int range_query(int i, int j)
{
  int sz = log2(j-i+1);
  return min(Sparse_Table[sz][i],Sparse_Table[sz][j+1-(1 << sz)]);
}

inline int countNumberOfElementEqualToXinLR(int l, int r, int x)
{
  int p1 = lower_bound(forest[x + offset].begin(), forest[x + offset].
      end(), l) - forest[x + offset].begin();
  int p2 = upper_bound(forest[x + offset].begin(), forest[x + offset].
      end(), r) - forest[x + offset].begin() - 1;
  if(p1 > p2) return -1;
  return p2 - p1 + 1;
}

inline int nxt(int i, int x)
{
  int b = i, e = n - 1, ans = -1;
  while(b <= e)
  {
    int m = (b + e) >> 1;
    int v = range_query(i, m);
    if(v >= x) ans = m, b = m + 1;
    else e = m - 1;
  }
  return ans;
}

inline int queryIndex(int l, int r, int x)
{
  int b = l, e = r, ans = -1;
  while(b <= e)
  {
    int m = (b + e) >> 1;
    if(countNumberOfElementEqualToXinLR(m, r, x) > 0) ans = m, b = m +
        1;
    else e = m - 1;
  }
  return ans;
}

inline void scanstr(string &k)
{
    register char c;
  k = "";
    for(c = gc(); c != '(' and c != ')'; c = gc());
    for(; c >= '(' and c <= ')'; c = gc()) k.push_back(c);
}

int main()
{
  scanstr(str);
  n = str.size() + 1;
  for(int i = 1; i < n; i++)
  {
    pf[i] = pf[i - 1] + (str[i - 1] == '(' ? 1 : -1);
    Sparse_Table[0][i] = pf[i];
    forest[pf[i] + offset].push_back(i);
  }
  build();
  int ans = 0;
  cnt[0] = 1;
  for(int i = 1; i < n; i++)
  {
    if(str[i - 1] != '(') continue;
    int e = nxt(i, pf[i] - 1);
    if(e < i) continue;
    int p = queryIndex(i, e, pf[i] - 1);
    if(p < i) continue;
    int l = p - i + 1;
    cnt[l]++;
    if(l > ans) ans = l;
  }
  printf("%d %d\n", ans, cnt[ans]);

  return 0;
}
```

3.21 Knapsack With Backtraking

```
#include <bits/stdc++.h>
using namespace std;
#define pii pair<int, int>
#define fi first
#define se second
#define pb push_back

int n,c;
vector<pii> v;
int res,aux;
double c2,aux2;
```

```
void bt(int i){
  if(i == n) return;

  aux2 = 0; c2 = c;
  for(int j=i; j<n && c2; j++){
    if(v[j].fi <= c2){
      c2 -= v[j].fi; aux2 += v[j].se;
    } else {
      aux2 += (v[j].se*c2)/v[j].fi;
      c2 = 0;
    }
  }
  if(aux2 + aux <= res) return;

  if(v[i].fi <= c){
    c -= v[i].fi;
    aux += v[i].se;
    if(aux > res) res = aux;
    bt(i+1);
    aux -= v[i].se;
    c += v[i].fi;
  }
  bt(i+1);
}

int32_t main(){
  ios::sync_with_stdio(false);cin.tie(0);

  cin>>n>>c;
  for(int i = 0; i < n; i++)
  {
    int wei,value; cin>>wei>>value;
    v.pb({wei,value});
  }

  sort(v.begin(), v.end(), [](pii a, pii b){
    return (a.se+0.0)/a.fi > (b.se+0.0)/b.fi;
  });

  bt(0);
  cout<<res<<endl;

  return 0;
}
```

3.22 Small To Large

```
https://codeforces.com/blog/entry/44351

// Small To Large (using map)
// Given a tree, every vertex has color. Query is
//how many vertices in subtree of vertex v are
// colored with color c?
// O(N*logN*logN), (we are using map)

#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e4 + 10;

vector<int> g[MAX];
int sz[MAX], col[MAX];
map<int, int> *cnt[MAX];

void getsz(int v, int p)
{
    sz[v] = 1;
    for(auto u : g[v])
        if(u != p){
            getsz(u, v); sz[v] += sz[u]; } }

void dfs(int v, int p)
{
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p)
       {
           dfs(u, v);
           if(sz[u] > mx)
               mx = sz[u], bigChild = u;
       }
    if(bigChild != -1)
        cnt[v] = cnt[bigChild];
    else
        cnt[v] = new map<int, int> ();
    (*cnt[v])[ col[v] ] ++;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           for(auto x : *cnt[u])
               (*cnt[v])[x.first] += x.second;
    //now (*cnt[v])[c] is the number of vertices in
    //subtree of vertex v that has color c. You can
    //answer the queries easily.
}

int32_t main()
{
    int n, m;

    cin >> n >> m;
    for(int i = 0; i < n; i++)
        cin >> col[i];
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v; u--; v--;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    getsz(0, -1);
    dfs(0, -1);

  return 0;
}


//////////////////////////////////////////////////


// dsu on tree (using vector)
```

```cpp
// Given a tree, every vertex has color. Query is
//how many vertices in subtree of vertex v are
// colored with color c?
// O(N*logN)

#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e4 + 10;

vector<int> g[MAX];
int sz[MAX], col[MAX];
vector<int> *vec[MAX];
int cnt[MAX];

void getsz(int v, int p)
{
    sz[v] = 1;
    for(auto u : g[v])
        if(u != p){
            getsz(u, v); sz[v] += sz[u]; } }


void dfs(int v, int p, bool keep)
{
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
           mx = sz[u], bigChild = u;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           dfs(u, v, 0);
    if(bigChild != -1)
        dfs(bigChild, v, 1), vec[v] = vec[bigChild];
    else
        vec[v] = new vector<int> ();
    vec[v]->push_back(v);
    cnt[ col[v] ]++;
    for(auto u : g[v])
       if(u != p && u != bigChild)
           for(auto x : *vec[u]){
               cnt[ col[x] ]++;
               vec[v] -> push_back(x);
           }
//now (*cnt[v])[c] is the number of vertices in subtree
//of vertex v that has color c. You can answer the queries
//easily. note that in this step *vec[v] contains all of
//the subtree of vertex v.
    if(keep == 0)
        for(auto u : *vec[v])
            cnt[ col[u] ]--;
}

int32_t main()
{
    int n, m;

    cin >> n >> m;
    for(int i = 0; i < n; i++)
        cin >> col[i];
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v; u--; v--;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    getsz(0, -1);
    dfs(0, -1, 0);

  return 0;
}

// Small To Large (heavy-light decomposition style)
// Given a tree, every vertex has color. Query is
//how many vertices in subtree of vertex v are
// colored with color c?
// O(N*logN)

#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e4 + 10;

vector<int> g[MAX];
int sz[MAX], col[MAX];
bool big[MAX];
int cnt[MAX];

void getsz(int v, int p)
{
    sz[v] = 1;
    for(auto u : g[v])
        if(u != p){
            getsz(u, v); sz[v] += sz[u]; } }

void add(int v, int p, int x)
{
    cnt[ col[v] ] += x;
    for(auto u: g[v])
        if(u != p && !big[u])
            add(u, v, x);
}

void dfs(int v, int p, bool keep)
{
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
       if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
        // run a dfs on small childs and clear them from cnt
            dfs(u, v, 0);
    if(bigChild != -1)
        // bigChild marked as big and not cleared from cnt
        dfs(bigChild, v, 1), big[bigChild] = 1;
    add(v, p, 1);
//now cnt[c] is the number of vertices in subtree of
//vertex v that has color c. You can answer the queries easily.
    if(bigChild != -1)
        big[bigChild] = 0;
```

```
    if(keep == 0)
        add(v, p, -1);
}

int32_t main()
{
    int n, m;

    cin >> n >> m;
    for(int i = 0; i < n; i++)
        cin >> col[i];
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v; u--; v--;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    getsz(0, -1);
    dfs(0, -1, 0);

  return 0;
}

// Small To Large (using nesting intervals)
// Given a tree, every vertex has color. Query is
//how many vertices in subtree of vertex v are
// colored with color c?
// O(N*logN)

#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e4 + 10;

vector<int> g[MAX];
int sz[MAX], col[MAX];
int st[MAX], ft[MAX];
int cnt[MAX], ver[MAX];
int tempo = 0;

void getsz(int v, int p)
{
    sz[v] = 1;
    ver[tempo] = v;
    st[v] = tempo++;
    for(auto u : g[v])
        if(u != p){
            getsz(u, v); sz[v] += sz[u]; }
    ft[v] = tempo++;
}

void dfs(int v, int p, bool keep)
{
    int mx = -1, bigChild = -1;
    for(auto u : g[v])
        if(u != p && sz[u] > mx)
          mx = sz[u], bigChild = u;
    for(auto u : g[v])
        if(u != p && u != bigChild)
            // run a dfs on small childs and
            // clear them from cnt
            dfs(u, v, 0);
    if(bigChild != -1)
        // bigChild marked as big and not cleared from cnt
        dfs(bigChild, v, 1);
    for(auto u : g[v])
  if(u != p && u != bigChild)
      for(int p = st[u]; p < ft[u]; p++)
        cnt[ col[ ver[p] ] ]++;
    cnt[ col[v] ]++;
//now cnt[c] is the number of vertices in subtree of vertex
//v that has color c. You can answer the queries easily.
    if(v == 1) cout << cnt[2] << '\n';
    if(keep == 0)
        for(int p = st[v]; p < ft[v]; p++)
          cnt[ col[ ver[p] ] ]--;
}

int32_t main()
{
    int n, m;

    cin >> n >> m;
    for(int i = 0; i < n; i++)
        cin >> col[i];
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v; u--; v--;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    getsz(0, -1);
    dfs(0, -1, 0);

  return 0;
}

/*
But why it is ? You know that why dsu has time
(for q queries); the code uses the same method.
Merge smaller to greater.

If you have heard heavy-light decomposition you
will see that function add will go light edges
only, because of this, code works in time.

*/
```

3.23 FastIO

```
#include <bits/stdc++.h>
using namespace std;

#define gc getchar_unlocked
#define pc putchar_unlocked

inline void scanint(int &k)
{
  bool sinal = true;
```

```
    register char c;
    k = 0;
    for(c = gc(); sinal and (c < '0' or c > '9'); c = gc())
      if(c == '-')
        sinal = false;
    for(; c >= '0' and c <= '9'; c = gc())
        k = (k << 3) + (k << 1) + c - '0';
  if(!sinal) k = -k;
}

inline void printint(int n)
{
  if(n < 0) pc('-');
  n = abs(n);
  int rev = n, cnt = 0;
  if(!n)
  {
    pc('0');
    pc('\n');
    return;
  }
  while(!(rev % 10))
    cnt++, rev /= 10;
  rev = 0;
  while(n)
    rev = (rev << 3) + (rev << 1) + n % 10, n /= 10;
  while(rev)
    pc(rev % 10 + '0'), rev /= 10;
  while(cnt--)
    pc('0');
  pc('\n');
}

inline void scanstr(string &k)
{
    register char c;
  k = "";
    for(c = gc(); c < 'a' or c > 'z'; c = gc());
    for(; c >= 'a' and c <= 'z'; c = gc()) k.push_back(c);
}

inline void printstr(string &k)
{
  for(char &c : k) putchar(c);
  putchar('\n');
}

int main()
{
  int k;
  scanint(k);
  printint(k);

    return 0;
}
```

3.24 String Matching Hash Sqrtdecomp

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e5 + 100;

typedef long long ll;

#define ii pair<int, long long>
#define fi first
#define se second

ll A = 911382323, B = 972663749;
ll h[MAX], p[MAX];
string s;

ll buildP(int k)
{
    if(k == 0)
        return p[0] = 1;
    return p[k] = (buildP(k - 1)*A) % B;
}

int main()
{
    ios_base::sync_with_stdio(false);
  cin.tie(NULL);

    cin >> s;
  buildP(s.size() + 10);

    int n = s.size();
    int sz = sqrt(n);

    unordered_set<ll> AAAA;
    vector<ii> BBBB;

  string aux;

    while(cin >> aux)
  {
        long long value = 0, j = 0;
        while(j < aux.size())
        {
            if(j == 0) value = aux[j];
            else value = (value * A + aux[j]) % B;
            j++;
        }
        if(aux.size() > sz)
            BBBB.push_back({aux.size(), value});
        else
            AAAA.insert(value);
  }
    sort(BBBB.begin(), BBBB.end());

  string ans;

  int j = 0, i = 0;

  while(i < s.size())
  {
      ans.push_back(s[i]);

      if(j == 0) h[j] = s[i];
      else h[j] = (h[j - 1] * A + s[i]) % B;
```

```
    int leng = -1;
    long long vh;

      for(int k = j; k >= 0 and k >= j - sz - 1 and leng < 0; k--)
      {
          if(k == 0) vh = h[j];
          else
          {
              vh = (h[j] - h[k - 1] * p[j - k + 1]) % B;
              if(vh < 0) vh += B;
          }
          if(AAAA.count(vh))
              leng = j - k + 1;
          //length j - k + 1
      }

    if(leng == -1)
    {
        for(int k = 0; k < BBBB.size(); k++)
        {
            int a = j - BBBB[k].first + 1;

              if(a < 0) break;

            if(a == 0) vh = h[j];
              else
              {
                  vh = (h[j] - h[a - 1] * p[j - a + 1]) % B;
                  if(vh < 0) vh += B;
              }

              if(vh == BBBB[k].se)
                  leng = BBBB[k].fi;

        }
    }
    if(leng != -1)
    {
          j -= leng;
          while(leng--) ans.pop_back();
    }
      ++j;
    i++;
  }
    cout << ans << '\n';


    return 0;
}
```

4 Dynamic Programming

4.1 Traveling Salesman Problem Bottom Up Dp

```
#include <bits/stdc++.h>
using namespace std;
const int OO = 0x3f3f3f3f;
int n;
double dist[20][20];
double pd[1 << 17][20];

int tsp(int ori)
{
  memset(pd, 63, sizeof(pd));
  for(int i = 0; i < n; i++)
    if(i != ori)
      pd[1 << i][i] = dist[ori][i];
  for(int k = 0; k < (1 << n); k++)
  for(int i = 0; i < n; i++)
  if(k & (1 << i))
    for(int j = 0; j < n; j++)
      if((k & (1 << j)) and i != j)
      pd[k][j] = min(pd[k][j], pd[k ^ (1 << j)][i] + dist[i][j]);
  return pd[(1 << n) - 1][ori];
}

int main()
{
  // inicializar dist, dist[i][j] guarda a distancia de i para j no
      grafo
  // chamar tsp

  return 0;
}
```

4.2 Knapsack Zero One Without Value

```
// Knapsack 0 - 1 sem valor em O((N*W) / word)
#include <bits/stdc++.h>
using namespace std;

int n, W, weight[10000];
bitset<10000> T[100];

bool knapsack()
{
  T[0][0] = 1;
  for(int i = 1; i <= n; i++)
    T[i] = ((T[i - 1] << weight[i - 1]) | T[i - 1]);
  return T[n][W];
}

void retrieve()
{
  vector<int> ans;
  for(int i = n; i > 0; i--)
    if(W >= weight[i - 1] and T[i - 1][W - weight[i - 1]])
    {
      ans.push_back(i - 1);
      W -= weight[i - 1];
    }
  for(int &w : ans) cout << weight[w] << ' '; puts("");
}

int main()
{
```

```
  cin >> n;
  for(int i = 0; i < n; i++)
    cin >> weight[i];
  cin >> W;
  cout << knapsack() << '\n';
  retrieve();

  return 0;
}
```

4.3 KnapsackErrichto

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int OO = 0x3f3f3f3f3f3f3f3f;

int n, w, maxi;
int value[120], weight[120];
int dp[100040];

int32_t main()
{
  cin >> n >> w;
  for(int i = 0; i < n; i++)
    cin >> weight[i] >> value[i];
  // dp[i] = maximum total value of itens with total weight exactly
  for(int item = 0; item < n; item++)
    for(int cur_wei = w - weight[item]; cur_wei >= 0; cur_wei--)
      dp[cur_wei + weight[item]] = max(dp[cur_wei + weight[item]],
                                       dp[cur_wei] + value[item]);
  cout << *max_element(dp, dp + w + 1) << '\n';

  return 0;
}
```

4.4 Edit Distance With DP

```
#include <bits/stdc++.h>
using namespace std;

string a, b;
int PD[2008][2008];

int solve(int i, int j)
{
  if(!i) return j;
  if(!j) return i;
  if(PD[i][j] != -1)
    return PD[i][j];
  // substituir um caracter se for preciso
  int ans1 = solve(i - 1, j - 1) + (a[i] != b[j]);
  //apagar o caracter da string i
  int ans2 = solve(i - 1, j) + 1;
  //apagar o caracter da string j
  int ans3 = solve(i, j - 1) + 1;
  return PD[i][j] = min(ans1, min(ans2, ans3));
}

int main()
{
  int q;
  cin >> q;
  while(q--)
  {
    memset(PD, -1, sizeof(PD));
    cin >> a >> b;
    a = "#" + a;
    b = "#" + b;
    cout << solve(a.size()-1, b.size()-1) << '\n';
  }

  return 0;
}
```

4.5 Digit DP Sum Of Digits In Range

```
#include "bits/stdc++.h"
using namespace std;

int dp[20][200][2];

int digitDP(int idx, int sum, int can, vector<int> &digit)
{// idx eh o indice atual, sum a soma dos digitos ate idx,
//e can uma flag para indicar se pode colocar
//qualquer valor a partir daqui
  if(idx == (int)digit.size())
    return sum;
  if(dp[idx][sum][can] != -1)
    return dp[idx][sum][can];
  int ans = 0;
  for(int i = 0; i < 10; i++)
    if(can or i <= digit[idx])
      ans += digitDP(idx + 1, sum + i,
        can or i < digit[idx], digit);
  return dp[idx][sum][can] = ans;
}

int query(int x) // responde a consulta de 0 ate x
{
  memset(dp, -1, sizeof(dp));
    vector<int> digit;
    while(x)
    {
        digit.push_back(x%10);
        x /= 10;
    }
    reverse(digit.begin(), digit.end());
    return digitDP(0, 0, 0, digit);
}

int main()
{
    int q, a, b;
    cin >> q;
    while(q--)
    {
    cin >> a >> b;
```

```
    cout << query(b) - query(a - 1) << '\n';
  }
    return 0;
}
```

4.6 Coin Problem Topdown Dp

```
#include <bits/stdc++.h>
using namespace std;
using namespace std;

vector<int> coin;
int memo[1000000];

int solve(int troco)
{
  if(troco < 0)
    return (1 << 25);
  if(memo[troco] != -1)
    return memo[troco];
  if(troco == 0)
    return 0;
  int ans = (1 << 25);
  for(int i = 0; i < coin.size(); i++)
    ans = min(ans, 1 + solve(troco - coin[i]));
  return memo[troco] = ans;
}

void ans(int troco)
{
  if(troco < 0)
    return;
  if(troco == 0)
    return;
  for(int i = 0; i < coin.size(); i++)
    if(solve(troco - coin[i]) + 1 == memo[troco])
    {
      cout << coin[i] << ' ';
      ans(troco - coin[i]);
      break;
    }
}

int main()
{
  memset(memo, -1, sizeof(memo));
  int n, troco;
  cin >> n >> troco;
  coin.resize(n);
  for(int &w : coin)
    cin >> w;
  cout << solve(troco) << '\n';
  ans(troco);
  puts("");

  return 0;
}
```

4.7 Kadane 3D

```
#include <bits/stdc++.h>
using namespace std;

int A, B, C;
int par[22][22][22], pd[22][22][22];

int main()
{
  cin >> A >> B >> C;
  for(int i = 1; i <= A; i++)
    for(int j = 1; j <= B; j++)
      for(int k = 1; k <= C; k++)
        cin >> par[i][j][k];

  for(int i = 1; i <= A; i++)
    for(int j = 1; j <= B; j++)
      for(int k = 1; k <= C; k++)
        pd[i][j][k] = pd[i][j - 1][k] + pd[i][j][k - 1]
           - pd[i][j - 1][k - 1] + par[i][j][k];

  int ans = -(1 << 25);
  for(int h1 = 1; h1 <= C; h1++)
  for(int h2 = h1; h2 <= C; h2++)
  for(int l1 = 1; l1 <= B; l1++)
    for(int l2 = l1; l2 <= B; l2++)
    {
      int sum = -(1 << 25);
      for(int i = 1; i <= A; i++)
      {
        int s = pd[i][l2][h2] - pd[i][l1 - 1][h2]
        - pd[i][l2][h1 - 1] + pd[i][l1 - 1][h1 - 1];
        sum = max(sum + s, s);
        ans = max(ans, sum);
      }
    }
  cout << ans << '\n';

  return 0;
}
```

4.8 KnapsackWithCopies

```
// O( S * sqrt( SumKi ) )

#include <bits/stdc++.h>
using namespace std;
const int OO = 0x3f3f3f3f;

int freq[50];
vector<int> value, weight;
int memo[5000][5000];

// Decompor o numero em uma soma de potencias
//de 2 de tal forma que qualquer numero entre 0 e k
// pode ser formado usando os numeros da decomposicao.
void decomp(int k, int w, int v)
```

```
{
  int i = 1;
  freq[1] = k;
  while(true)
  {
    int m = (freq[i] - 1) / 2;
    if(freq[i] - m * 2 == 0) break;
    freq[i] -= 2 * m;
    freq[2 * i] += m;
    i++;
  }
  for(int i = 0; i < 32; i++)
  {
    while(freq[i]--)
    {
      value.push_back(i * v);
      weight.push_back(i * w);
    }
    freq[i] = 0;
  }
}

int solve(int id, int W)
{
  if(memo[id][W] != -1)
    return memo[id][W];
  if(id == value.size() or !W)
    return memo[id][W] = 0;
  int ans = 0;
  if(weight[id] > W)
    ans = solve(id + 1, W);
  else
    ans = max(value[id] + solve(id + 1, W
      - weight[id]), solve(id + 1, W));
  return memo[id][W] = ans;
}

int main()
{
  int n, w, v, k, S;
  memset(memo, -1, sizeof(memo));
  cin >> n >> S;
  for(int i = 0; i < n; i++)
  {
    cin >> v >> w >> k;
    decomp(k, w, v);
  }
  cout << solve(0, S) << '\n';

  return 0;
}
```

4.9 Knapsack0-kSemValor

```
// Knapsack 0 - k sem valor em O((N*W*LogK) / word)

#include <bits/stdc++.h>
using namespace std;

int n, W, weight[10000], K[10000];
bitset<10000> T[100];

bool knapsack()
{
  T[0][0] = 1;
  for(int i = 1; i <= n; i++)
  {
    int s = K[i - 1];
    T[i] = T[i - 1];
    for(int p = 1; p <= s; s -= p, p *= 2)
      T[i] |= ((T[i] << (weight[i - 1] * p)) | T[i]);
    if(s)
      T[i] |= ((T[i] << (weight[i - 1] * s)) | T[i]);
  }
  return T[n][W];
}

void retrieve()
{
  vector<pair<int, int>> ans;
  for(int i = n; i > 0; i--)
  {
    int s = K[i - 1], qtd = 0;
    for(int p = 1; p <= s; p *= 2)
    if(W >= weight[i - 1] * p and T[i - 1][W - weight[i - 1] * p])
      W -= weight[i - 1] * p, qtd += p, s -= p;
    if(W >= weight[i - 1] * s and T[i - 1][W - weight[i - 1] * s])
      W -= s * weight[i - 1], qtd += s;
    if(qtd) ans.push_back({qtd, i - 1});
  }//first eh q quantidade de pesos i - 1
  for(pair<int, int> &w : ans)
    cout << w.first << ' ' << weight[w.second] << '\n';
}

int main()
{
  cin >> n;
  for(int i = 0; i < n; i++)
    cin >> weight[i];
  for(int i = 0; i < n; i++)
    cin >> K[i];
  cin >> W;
  cout << (knapsack() ? "possible\n" : "impossible\n");
  retrieve();

  return 0;
}
```

4.10 KnapsackwithPDtopdown

```
#include <bits/stdc++.h>
using namespace std;

int n, W, weight[2005], value[2005];
int memo[2005][2005];

int solve(int id, int W)
{
  if(memo[id][W] != -1)
    return memo[id][W];
```

```
  if(id == n or !W)
    return memo[id][W] = 0;
  int ans = 0;
  if(weight[id] > W)
    ans = solve(id + 1, W);
  else
    ans = max(value[id] + solve(id + 1,
      W - weight[id]), solve(id + 1, W));
  return memo[id][W] = ans;
}

void ans(int id, int W)
{
  if(id == n or !W)
    return;
  if(solve(id + 1, W) == memo[id][W])
    ans(id + 1, W);
  else
  {
    cout << id << ' ';
    ans(id + 1, W - weight[id]);
  }
}

int main()
{
  memset(memo, -1, sizeof(memo));
  cin >> n >> W;
  for(int i = 0; i < n; i++)
    cin >> weight[i] >> value[i];
  cout << solve(0, W) << '\n';
  cout << "Objetos escolhidos 0 - indexdos\n";
  ans(0, W);
  puts("");

  return 0;
}
```

4.11 Subset Sum

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e6 + 10;

int n, x, weight[1005];
bool pd[MAX];
int ans[MAX];

void printAns(int m)
{
  cout << ans[m] << ' ';
  if(m - ans[m] > 0)
    printAns(m - ans[m]);
}

int main()
{
  cin >> n >> x;
  int sum = 0;
  for(int i = 0; i < n; i++) cin >> weight[i], sum += weight[i];
  pd[0] = 1;
  for(int j = 0; j < n; j++)
    for(int i = sum; i >= 0; i--)
      if(pd[i] and !pd[i + weight[j]])
      {
        ans[i + weight[j]] = weight[j];
        pd[i + weight[j]] = 1;
      }

  printAns(x);
  puts("");

  return 0;
}
```

4.12 Kadane 2D

```
#include <bits/stdc++.h>
using namespace std;

int pd[100][100], A[100][100];

int main()
{
  int n, m;
  cin >> n >> m;
  for(int i = 1; i <= n; i++)
    for(int j = 1; j <= m; j++)
      cin >> A[i][j], pd[i][j] = pd[i][j - 1] + A[i][j];
  int ans = 0;
  for(int i = 1; i <= n; i++)
    for(int j = i + 1; j <= m; j++)
    {
      int sum = 0;
      for(int k = 1; k <= n; k++)
      {
        sum += pd[k][j] - pd[k][i - 1];
        if(sum < 0) sum = 0;
        ans = max(ans, sum);
      }
    }
  cout << ans << '\n';

  return 0;
}
```

4.13 Traveling Salesman Problem Topdown Dp

```
#include <bits/stdc++.h>
using namespace std;

int dist[22][22], m;
int memo[20][1 << 20];

int solve(int id, int mask) {
  if(((1 << m) - 1) == mask)
    return dist[id][0];
```

```
  if(memo[id][mask] != -1)
    return memo[id][mask];
  int ans = INT_MAX;
  for(int i = 0; i < m; i++)
    if((mask & (1 << i)) == 0)
      ans = min(ans, dist[id][i] + solve(i, mask | (1 << i)));
  return memo[id][mask] = ans;
}

int main() {
  memset(memo, -1, sizeof(memo));
  //inicializa a matriz dist com as distancias
  //de todo mundo pra todo mundo..
  cout << solve(0, 1) << '\n';
  return 0;
}
```

4.14 Longest Increasing Subsequence

```
#include <bits/stdc++.h>
using namespace std;

void lis(vector<int> &arr)
{
  vector<int> pilha;
  int pai[1000], pos[1000];
  for(int i = 0; i < arr.size(); i++)
  {
    int p = int(upper_bound(pilha.begin(),
       pilha.end(), arr[i]) - pilha.begin());
    if(p == pilha.size())
      pilha.push_back(arr[i]);
    else
      pilha[p] = arr[i];
    pos[p] = i;
    if(!p)
      pai[i] = -1;
    else
      pai[i] = pos[p - 1];
  }
  vector<int> L;
  int aux = pos[pilha.size() - 1];
  cout << pilha.size() << '\n';
  while(aux != -1)
  {
    L.push_back(arr[aux]);
    aux = pai[aux];
  }
  reverse(L.begin(), L.end());
  for(const int &w : L)
    cout << w << ' ';
  cout << '\n';
}

int main()
{
  int n;
  cin >> n;
  vector<int> arr(n);
  for(int &w : arr)
    cin >> w;
  lis(arr);

  return 0;
}
```

4.15 Longest Common Subsequece And Edit Distance

```
#include <bits/stdc++.h>
using namespace std;

// longest common substring

int pd[1000][1000];

int LCS(string a, string b)
{
    for(int i = 1; i <= a.size(); i++)
        for(int j = 1; j <= b.size(); j++)
            if(a[i-1] == b[j-1])
                pd[i][j] = pd[i-1][j-1] + 1;
            else
                pd[i][j] = max(pd[i][j-1], pd[i-1][j]);
    return pd[a.size()][b.size()];
}

int main()
{
    string a, b;

    cin >> a >> b;
    int lcs = LCS(a, b);

    cout << lcs << '\n';
    cout << "Edit Distance: " << a.size()+b.size()-2*lcs << '\n';

    return 0;
}
```

4.16 Knapsack With Copies SqrtN Memory

```
#include <bits/stdc++.h>
using namespace std;
#define bug(x) cout << #x << "  >>>>>>>  " << x << '\n'
#define _ << " , " <<
//#define int long long
#define Max(a, b) (a > b ? a : b)
#define Min(a, b) (a < b ? a : b)
#define ii pair<int, int>
#define fi first
#define se second
#define UNTIL(t) while (clock() < (t) * CLOCKS_PER_SEC)
const int MAX = 20002; //2 * 10^5
const int MOD = 1000000007; //10^9 + 7
const int OO = 0x3f3f3f3f; // 0x3f3f3f3f;
const double EPS = 1e-9;  //10^-9
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
```

```
int n, S;
int B[205];
int C[205];
int V[3500];
int W[3500];
int ID[3500];
int id = 1;
int dp[MAX];
int linha[70][MAX], T[70][MAX];
int L[70];

void add(int j)
{
  int sum = 0;
  for(int k = 0; sum + (1 << k) <= C[j]; sum += (1 << k), k++)
  {
    V[id] = B[j] * (1 << k);
    W[id] = (1 << k);
    ID[id] = j;
    id++;
  }
  int r = C[j] - sum;
  if(r > 0)
  {
    V[id] = B[j] * r;
    W[id] = r;
    ID[id] = j;
    id++;
  }
}

int32_t main()
{
  cin >> n;
  for(int i = 1; i <= n; i++) cin >> B[i];
  for(int i = 1; i <= n; i++) cin >> C[i];
  for(int i = 1; i <= n; i++) add(i);
  cin >> S;

  for(int j = 1; j <= S; j++)
    dp[j] = OO;

  int cnt = 0, k = -1, sq = max(10, (int)sqrt(id * 1.));
  for(int i = 1; i < id; i++)
  {
    if(cnt % sq == 0)
    {
      cnt = 0;
      k++;
      for(int j = 0; j <= S; j++)
        linha[k][j] = dp[j];
      L[k] = i - 1;
    }
    for(int j = S; j >= V[i]; j--)
      dp[j] = Min(dp[j - V[i]] + W[i], dp[j]);
    cnt++;
  }

  int last_raw = id - 1, s = S;

  vector<int> note(n + 1);

  while(last_raw >= 1)
  {
    int first_raw = last_raw - 1;
    while(first_raw > L[k])
      first_raw--;

    for(int j = 0; j <= S; j++)
      T[0][j] = linha[k][j];

    for(int i = 1; i <= last_raw - first_raw; i++)
      for(int j = 0; j <= S; j++)
        if(j >= V[i + first_raw])
          T[i][j] = Min(T[i - 1][j], T[i - 1][j - V[i + first_raw]] +
              W[i + first_raw]);
        else
          T[i][j] = T[i - 1][j];

    for(int i = last_raw - first_raw; i > 0; i--)
      if(T[i][s] != T[i - 1][s])
      {
        note[ID[i + first_raw]] += W[i + first_raw];
        s -= V[i + first_raw];
      }

    last_raw = first_raw;
    k--;
  }

  int number_of_notes = 0;
  for(int &w : note)
    number_of_notes += w;

  cout << number_of_notes << '\n';

  for(int i = 1; i <= n; i++)
    cout << note[i] << ' ';
  puts("");


  return 0;
}
```

5 Math

5.1 Pollard Rho

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long

ll llrand()
{
  ll tmp = rand();
  return (tmp << 31) | rand();
}

ll add(ll a, ll b, ll c)
```

```
{
  ll ans = (a + b) % c;
    if(ans < 0) ans += c;
    return ans;
}

ll mulmod(ll a, ll b, ll c)
{
  ll ans = 0;
  while(b)
  {
    if(b & 1) ans = add(ans, a, c);
    a = add(a, a, c);
    b /= 2;
  }
  return ans;
}

ll rho(ll n)
{
    if(n % 2 == 0) return 2;
    ll d = n;
    while(d == n)
    {
        ll c = llrand() % n, x = llrand() % n, y = x;
        do
        {
            x = add(mulmod(x, x, n), c, n);
            y = add(mulmod(y, y, n), c, n);
            y = add(mulmod(y, y, n), c, n);
            d = __gcd(abs(x - y), n);
        }while(d == 1);
    }
    return d;
}

// Miller-Rabin AQUI

vector<ll> fac;

void factors(ll n) // encontrar os fatores primos de N
{// Usar Miller-Rabin para testar se N eh primo
    if(n == 1) return;
    if(isprime(n)) { fac.push_back(n); return; }
    ll d = rho(n);
    factors(d);
    factors(n / d);
}

int32_t main()
{
    srand(time(NULL));
    ll n;
    cin >> n;
    cout << rho(n) << '\n';

    return 0;
}
```

5.2 Mod Gaussian Elimination

```
#include<bits/stdc++.h>
using namespace std;
//#define int long long
#define pb push_back
#define inf 0x3f3f3f3f

int MOD = 1000000007LL;

inline int prod(int a, int b)
{
  return ((((a % MOD) * 1LL * (b % MOD)) % MOD) + MOD) % MOD;
}

inline int sub(int a, int b)
{
  return ((((a % MOD) - (b % MOD)) % MOD) + MOD) % MOD;
}

inline int expMod(int x, int e)
{
  int ans = 1;
  while(e > 0)
  {
    if(e & 1LL) ans = prod(ans, x), e--;
    else x = prod(x, x), e /= 2;
  }
  return ans;
}

inline int inv(int x)
{
  return expMod(x, MOD - 2);
}

inline int gauss (vector<vector<int>> a, int mod)
{
  MOD = mod;

    int n = (int) a.size();
    int m = (int) a[0].size();

    vector<int> where (m, -1);
    for(int col = 0, row = 0; col < m and 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(a[sel][col] == 0)
            continue;
        for(int i = col; i < m; ++i)
            swap(a[sel][i], a[row][i]);
        where[col] = row;
        for(int i = row + 1; i < n; ++i)
    {
            int c = prod(a[i][col], inv(a[row][col]));
            for(int j = col; j < m; ++j)
            a[i][j] = sub(a[i][j], prod(a[row][j], c));
```

```
        }
        ++row;
    }
  int ans = 0;
    for(int i = 0; i < m; ++i)
        if(where[i] != -1)
            ans++;
    return n - ans;
}

int32_t main()
{
  int n, m, a, k, t, caso = 1;

  cin >> t;
  while(t--)
  {
    scanf(" %d %d %d", &n, &m, &k);
    vector<vector<int>> A(n, vector<int>(n));
    while(m--)
    {
      int u, v;
      scanf(" %d %d", &u, &v), u--; v--;
      A[u][v] = A[v][u] = 1;
      if(u != v) A[u][v] = A[v][u] = k - 1;
    }
    for(int i = 0; i < n; i++) A[i][i] = 1;
    int ans = gauss(A, k);
    MOD = 1000000007LL;
    printf("Case %d: %d\n", caso++, expMod(k, ans));
  }

  return 0;
}
```

5.3 Catalan Numbers

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1e5 + 10;
const long long MOD = 1000000000;

int catalan[MAX];

void init()
{
    catalan[0] = catalan[1] = 1;
    for(int i = 2; i <= 1000; i++)
        for(int j = 0; j < i; j++)
        {
            catalan[i] += (catalan[j] * catalan[i-j-1]) % MOD;
            if(catalan[i] >= MOD)
                catalan[i] -= MOD;
    }
}

int main()
{
  init();
  int n;

  while(cin >> n)
    printf("%d\n", catalan[n]);

  return 0;
}
```

5.4 Matrix Exponentiation

```
#include <bits/stdc++.h>
using namespace std;
#define matrix vector<vector<int>>

matrix init(int n, int m, int value = 0)
{
    return vector<vector<int>>(n, vector<int>(m, value));
}

void printtt(const matrix &M)
{
    for(int i = 0; i < M.size(); i++)
    {
        for(int j = 0; j < M[0].size(); j++)
            cout << M[i][j] << ' ';
        puts("");
    }
}

matrix multiply(const matrix &A, const matrix &B)
{
    matrix C = init(A.size(), B[0].size());
    for(int i = 0; i < A.size(); i++)
        for(int j = 0; j < B[i].size(); j++)
            for(int k = 0; k < B.size(); k++)
                C[i][j] += A[i][k] * B[k][j];
    return C;
}

matrix exp(matrix M, int k)
{
    matrix I = init(M.size(), M[0].size());
    for(int i = 0; i < M.size(); i++) I[i][i] = 1;
    while(k)
        if(k & 1) I = multiply(I, M), k--;
        else M = multiply(M, M), k /= 2;
    return I;
}


int determinantOfMatrix(matrix mat)
{
  int n = mat.size();
    int num1, num2, det = 1, index, total = 1;
    int temp[n + 1];
    for(int i = 0; i < n; i++)
    {
        index = i;
        while(mat[index][i] == 0 and index < n)
            index++;
```

```
        if(index == n)
            continue;
    if(index != i)
        {
            for(int j = 0; j < n; j++)
                swap(mat[index][j], mat[i][j]);
            det = det*pow(-1,index-i);
    }
    for(int j = 0; j < n; j++)
      temp[j] = mat[i][j];
    for(int j = i+1; j < n; j++)
    {
      num1 = temp[i];
      num2 = mat[j][i];
      for(int k = 0; k < n; k++)
        mat[j][k] = (num1 * mat[j][k]) - (num2 * temp[k]);
      total = total * num1;
    }
  }
    for(int i = 0; i < n; i++)
        det = det * mat[i][i];
    return (det/total);
}


int32_t main()
{
    int n, m;

    cin >> n >> m;
    matrix A = init(n, m);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            cin >> A[i][j];
    matrix C = exp(A, 7);
    printtt(C);

    return 0;
}
```

5.5 Baby Step Giant Step

```
// a ^ kcongb mod m

int value[1000008];
int cor[1000008], tempo = 1;

// com vetor o modulo deve ser <= 10^7 fica O(sqrt(m))
inline int discreteLogarithm(int a, int b, int m) {
    tempo++;
  a %= m; b %= m;
    int n = (int)sqrt(m + .0) + 1, an = 1;
  for(int i = 1; i <= n; i++) an = (an * 1LL * a) % m;
    for(int i = 1, cur = an; i <= n; i++) {
      if(cor[cur] < tempo) value[cur] = i, cor[cur] = tempo;
        cur = (cur * 1LL * an) % m;
    }
    for(int j = 0, cur = b; j <= n; j++) {
        if(cor[cur] == tempo) {
            int ans = value[cur] * n - j;
            if(ans < m)
                return ans;
        }
    cur = (cur * 1LL * a) % m;
    }
  return -1;
}

// com mapa o modulo pode ser ateh <= 10^12 fica O(sqrt(m) * log(m))
int discreteLogarithm(int a, int b, int m)
{
  a %= m; b %= m;
    int n = (int)sqrt(m + .0) + 1, an = 1;
  for(int i = 1; i <= n; i++) an = (an * a) % m;
    unordered_map<int, int> value;
    for(int i = 1, cur = an; i <= n; i++) {
      if(!value.count(cur)) value[cur] = i;
        cur = (cur * an) % m;
    }
    for(int j = 0, cur = b; j <= n; j++) {
        if(value[cur]) {
            int ans = value[cur] * n - j;
            if(ans < m)
                return ans;
        }
    cur = (cur * a) % m;
    }
    return -1;
}
```

5.6 Gaussian Elimination For Max Subset Xor

```
#include <bits/stdc++.h>
using namespace std;

#define ull unsigned long long

int MSB(ull n)
{
    int cnt = 0;
    while(n)
    {
        cnt++;
        n >>= 1;
    }
    return cnt;
}

int main()
{
    int n;
    cin >> n;
    ull a[n];
    for(int i = 0; i < n; i++)
        cin >> a[i];
    int lengths[n];
    for(int i = 0; i < n; i++)
        lengths[i] = MSB(a[i]);
    //eh um array que armazena os coeficientes
    //das equacoes
```

```
    vector<ull> buckets[65];
    //para a Gaussian Elimination, semelhante
    //a linha da matriz em algebra linear
    for(int i = 0; i < n; i++)
        buckets[lengths[i]].push_back(a[i]);
    ull modified_array[100], m_index = 0;

    // Gaussian Elimination
    for(int i = 64; i > 0; i--)
        if(buckets[i].size())
        {
            modified_array[m_index++] = buckets[i][0];
            for(int j = 1; j < buckets[i].size(); j++)
            {
                ull temp = buckets[i][0] ^ buckets[i][j];
                int len = MSB(temp);
                buckets[len].push_back(temp);
            }
        }
    ull ans = 0;
    for(int i = 0; i < m_index; i++)
        if(ans < (ans ^ modified_array[i]))
            ans = (ans ^ modified_array[i]);
    cout << ans << '\n';

    return 0;
}
```

5.7 Counting Number Of Times That A Digit Appears Until N

```
ll digits(int n, int d)
{
  ll res = 0, pot = 1, rem = 0;
  while (n)
  {
    int x = n%10;
    n /= 10;
    if (x > d) res += (n+1)*pot;
    else res += n*pot;
    if (x == d) res += rem+1;
    if (d == 0) res -= pot;
    rem += pot * x;
    pot *= 10;
  }
  return res;
}
```

5.8 Chinese Remainder Theorem

```
//codar em Python para evitar problemas de overflow
// O(Tlog(lcm(n1*n2*..)))
//https://codeforces.com/blog/entry/61290

#include<bits/stdc++.h>
using namespace std;
const int MAX = 20;
#define ll long long

ll GCD(ll a, ll b) { return (b == 0) ? a : GCD(b, a % b); }

inline ll LCM(ll a, ll b) { return a / GCD(a, b) * b; }

inline ll normalize(ll x, ll mod) { x %= mod; if (x < 0) x += mod;
    return x; }

struct GCD_type { ll x, y, d; };

GCD_type ex_GCD(ll a, ll b)
{
    if (b == 0) return {1, 0, a};
    GCD_type pom = ex_GCD(b, a % b);
    return {pom.y, pom.x - a / b * pom.y, pom.d};
}

int t;
ll a[MAX], n[MAX], ans, lcm;

int main()
{
    cin >> t;
    for(int i = 1; i <= t; i++)
        cin >> a[i] >> n[i], normalize(a[i], n[i]);
    ans = a[1];
    lcm = n[1];
    for(int i = 2; i <= t; i++)
    {
        auto pom = ex_GCD(lcm, n[i]);
        ll x1 = pom.x;
        ll d = pom.d;
        if((a[i] - ans) % d != 0) return cerr << "No solutions" <<
            endl, 0;
        ans = normalize(ans + x1 * (a[i] - ans) / d % (n[i] / d) * lcm
            ,
      lcm * n[i] / d);
        lcm = LCM(lcm, n[i]);
        // you can save time by replacing above lcm * n[i] /d
        // by lcm = lcm * n[i] / d
    }
    cout << ans << " " << lcm << endl;

    return 0;
}
```

5.9 Modular Arithmetic

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int MOD = 1000000007LL;

int normalize(int x)
{
  x = x % MOD;
  if(x < 0) x += MOD;
  return x;
}
```

```
int add(int a, int b)
{
  return normalize(normalize(a) + normalize(b));
}

int prod(int a, int b)
{
  return normalize(normalize(a) * normalize(b));
}

int sub(int a, int b)
{
  return normalize(normalize(a) - normalize(b));
}

int expMod(int x, int e)
{
  int ans = 1;
  while(e > 0)
  {
    if(e & 1LL) ans = prod(ans, x), e--;
    else x = prod(x, x), e /= 2;
  }
  return normalize(ans);
}

int inv(int x)
{
  return expMod(x, MOD - 2);
}


int extended_euclidean(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = extended_euclidean(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

int inv(int a, int m) {
  int x, y;
  int g = extended_euclidean(a, m, x, y);
  if (g != 1) return -1; // nao tem inverso
  return ((x % m) + m) % m;
}
```

5.10 Karatsuba

```
#include <bits/stdc++.h>
using namespace std;

typedef vector<long long> vll;

vll karatsubaMultiply(const vll &a, const vll &b) {
  int n = a.size();
  vll res(n + n);
  if (n <= 32) {
      for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
          res[i + j] += a[i] * b[j];
      return res;
  }

  int k = n >> 1;
  vll a1(a.begin(), a.begin() + k);
  vll a2(a.begin() + k, a.end());
  vll b1(b.begin(), b.begin() + k);
  vll b2(b.begin() + k, b.end());

  vll a1b1 = karatsubaMultiply(a1, b1);
  vll a2b2 = karatsubaMultiply(a2, b2);

  for(int i = 0; i < k; i++)
      a2[i] += a1[i];
  for(int i = 0; i < k; i++)
      b2[i] += b1[i];

  vll r = karatsubaMultiply(a2, b2);

  for(int i = 0; i < (int) a1b1.size(); i++)
      r[i] -= a1b1[i];
  for(int i = 0; i < (int) a2b2.size(); i++)
      r[i] -= a2b2[i];

  for(int i = 0; i < (int) r.size(); i++)
      res[i + k] += r[i];
  for(int i = 0; i < (int) a1b1.size(); i++)
      res[i] += a1b1[i];
  for(int i = 0; i < (int) a2b2.size(); i++)
      res[i + n] += a2b2[i];
  return res;
}

int main()
{
  vll a = {8, 7, 5};
  vll b = {12};

  vll c = karatsubaMultiply(a, b);

  for(auto it : c) cout << it << ' '; puts("");

  return 0;
}
```

5.11 Fast Fourier Transform

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

using namespace std;

typedef complex<double> ftype;
const double pi = acos(-1);
```

```
const int maxn = 1 << 22;
ftype w[maxn];

void init()
{
    for(int i = 0; i < maxn; i++)
        w[i] = polar(1., 2 * pi / maxn * i);
}

template<typename T>
void fft(T *in, ftype *out, int n, int k = 1)
{
    if(n == 1)
    {
        *out = *in;
        return;
    }
    int t = maxn / n;
    n >>= 1;
    fft(in, out, n, 2 * k);
    fft(in + k, out + n, n, 2 * k);
    for(int i = 0, j = 0; i < n; i++, j += t)
    {
        ftype t = w[j] * out[i + n];
        out[i + n] = out[i] - t;
        out[i] += t;
    }
}

vector<ftype> evaluate(vector<int> p)
{
    while(__builtin_popcount(p.size()) != 1)
        p.push_back(0);
    vector<ftype> res(p.size());
    fft(p.data(), res.data(), p.size());
    return res;
}

vector<int> interpolate(vector<ftype> p)
{
    int n = p.size();
    vector<ftype> inv(n);
    fft(p.data(), inv.data(), n);
    vector<int> res(n);
    for(int i = 0; i < n; i++)
        res[i] = round(real(inv[i]) / n);
    reverse(begin(res) + 1, end(res));
    return res;
}

void align(vector<int> &a, vector<int> &b)
{
    int n = a.size() + b.size() - 1;
    while(a.size() < n)
        a.push_back(0);
    while(b.size() < n)
        b.push_back(0);
}

vector<int> poly_multiply(vector<int> a, vector<int> b)
{
    align(a, b);
    auto A = evaluate(a);
    auto B = evaluate(b);
    for(int i = 0; i < A.size(); i++)
        A[i] *= B[i];
    return interpolate(A);
}

const int base = 10;
vector<int> normalize(vector<int> c)
{
    int carry = 0;
    for(auto &it: c)
    {
        it += carry;
        carry = it / base;
        it %= base;
    }
    while(carry)
    {
        c.push_back(carry % base);
        carry /= base;
    }
    return c;
}

vector<int> multiply(vector<int> a, vector<int> b)
{
    return normalize(poly_multiply(a, b));
}

vector<int> faz(string s)
{
  vector<int> ans;
  for(char &c : s)
    ans.push_back(c-'0');
  return ans;
}

string multAB(string s1, string s2)
{
  if(s1 == "0" or s2 == "0")
    return "0";
  bool sinall;
  if(s1[0] == '-' and s2[0] == '-' or s1[0] != '-' and s2[0] != '-')
    sinall = true;
  else
    sinall = false;
  if(s1[0] == '-') s1[0] = '0';
  if(s2[0] == '-') s2[0] = '0';
  vector<int> A = faz(s1), B = faz(s2);
  A = normalize(A);
  B = normalize(B);
  reverse(A.begin(), A.end());
  reverse(B.begin(), B.end());
  auto C = multiply(A, B);
  while(C.back() == 0)
      C.pop_back();
  reverse(C.begin(), C.end());
  string ans;
  ans += (!sinall ? "-" : "");
```

```
  for(int &c: C)
    ans += char(c + '0');
  return ans;
}

int main()
{
  int t;
    init();

    cin >> t;
    while(t--)
    {
    string s1, s2;
    cin >> s1 >> s2;// le os dois numeros como strings
    if(s1 == "0" or s2 == "0")
    {
      puts("0");
      continue;
    }
    vector<int> A = faz(s1), B = faz(s2);
    A = normalize(A);
    B = normalize(B);
    reverse(A.begin(), A.end());
    reverse(B.begin(), B.end());

    auto C = multiply(A, B);

    while(C.back() == 0)
          C.pop_back();
    reverse(C.begin(), C.end());
    for(int &c: C)
      cout << c;
    puts("");
    }

  /*
    init();
      vector<int> a = {3, 4}, b = {2, 3};
    auto C = poly_multiply(a, b);
    int k = int(a.size() + b.size()) - 1;
    for(int i = 0; i < k; i++)
        cout << C[i] << "X^" << k-i-1 << (i < k-1 ? " + " : "\n");
  */

    return 0;
}



////////////////////////////////////////////////////////


#include <bits/stdc++.h>

using namespace std;

typedef long double ld;
const double PI = acos(-1);

struct T
{
  ld x, y;
  T() : x(0), y(0) {}
  T(ld a, ld b=0) : x(a), y(b) {}

  T operator/=(ld k) { x/=k; y/=k; return (*this); }
  T operator*(T a) const { return T(x*a.x - y*a.y, x*a.y + y*a.x); }
  T operator+(T a) const { return T(x+a.x, y+a.y); }
  T operator-(T a) const { return T(x-a.x, y-a.y); }
} a[1 << 23], b[1 << 23];

void fft(T* a, int n, int s)
{
  for (int i=0, j=0; i<n; i++)
  {
    if (i>j) swap(a[i], a[j]);
    for (int l=n/2; (j^=l) < l; l>>=1);
  }
  for(int i = 1; (1<<i) <= n; i++)
  {
    int M = 1 << i;
    int K = M >> 1;
    T wn = T(cos(s*2*PI/M), sin(s*2*PI/M));
    for(int j = 0; j < n; j += M)
    {
      T w = T(1, 0);
      for(int l = j; l < K + j; ++l)
      {
        T t = w*a[l + K];
        a[l + K] = a[l]-t;
        a[l] = a[l] + t;
        w = wn*w;
      }
    }
  }
}

void multiply(T* a, T* b, int n)
{
  fft(a,n,1);
  fft(b,n,1);
  for (int i = 0; i < n; i++)
    a[i] = a[i]*b[i];
  fft(a,n,-1);
  for (int i = 0; i < n; i++)
    a[i] /= n;
}

int main()
{
  int n, na, nb, c;
  cin >> na >> nb;
  n = na + nb;
  while(n&(n-1))
    n++;
  for(int i = n - na; i < n; i++)
  {
    cin >> c;
    a[i] = T(c);
  }
```

```
  for(int i = n - nb; i < n; i++)
  {
    cin >> c;
    b[i] = T(c);
  }
  multiply(a, b, n);
  for(int i = 0; i < n - 1; i++)
    cout << int(a[i].x + 0.5) << "X^"
     << n - 2 - i << (i < n - 2 ? " + " : "");
  puts("");

  /*
    3 2
    1 0 0
    2 3
    0X^6 + 0X^5 + 0X^4 + 2X^3 + 3X^2 + 0X^1 + 0X^0
  */

  return 0;
}


////////////////////////////////////////////////////////////////
//contar quantos subarrays de soma diferentes existem usando FFT

#include <bits/stdc++.h>

using namespace std;

typedef long double ld;
const long double PI = acos(-1);

struct T
{
  ld x, y;
  T() : x(0), y(0) {}
  T(ld a, ld b=0) : x(a), y(b) {}

  T operator/=(ld k) { x/=k; y/=k; return (*this); }
  T operator*(T a) const { return T(x*a.x - y*a.y, x*a.y + y*a.x); }
  T operator+(T a) const { return T(x+a.x, y+a.y); }
  T operator-(T a) const { return T(x-a.x, y-a.y); }
} a[16777219], b[16777219];

int pd[16777219];

void fft(T* a, int n, int s)
{
  for(int i=0, j=0; i<n; i++)
  {
    if (i>j) swap(a[i], a[j]);
    for (int l=n/2; (j^=l) < l; l>>=1);
  }
  for(int i = 1; (1<<i) <= n; i++)
  {
    int M = 1 << i;
    int K = M >> 1;
    T wn = T(cos(s*2*PI/M), sin(s*2*PI/M));
    for(int j = 0; j < n; j += M)
    {
      T w = T(1, 0);
      for(int l = j; l < K + j; ++l)
      {
        T t = w*a[l + K];
        a[l + K] = a[l]-t;
        a[l] = a[l] + t;
        w = wn*w;
      }
    }
  }
}

void multiply(T* a, T* b, int n)
{
  fft(a,n,1);
  fft(b,n,1);
  for (int i = 0; i < n; i++)
    a[i] = a[i]*b[i];
  fft(a,n,-1);
  for (int i = 0; i < n; i++)
    a[i] /= n;
}

int main()
{
  int k;
  cin >> k;
  for(int i = 1; i <= k; i++)
  {
    int aux;
    cin >> aux;
    pd[i] = pd[i - 1] + aux;
  }
  if(k >= 10000)
  {
    for(int i = 0; i <= k; i++)
    {
      a[pd[i] + pd[k]].x = 1;
      b[pd[k] - pd[i]].x = 1;
    }
    int n = pd[k] + pd[k];
    n = 2 * n;
    while(n & (n - 1))
      n++;
    multiply(a, b, n);
    int ans = 0;
    for(int i = 0; i <= n; i++)
      if(int(a[i].x + 0.5) > 0 and (i - 2 * pd[k]) > 0)
        ans++;
    cout << ans << '\n';
  }
  else
  {
    int cnt = 0;
    unordered_set<int> ans1;
    for(int i = 1; i <= k; i++)
      for(int j = i; j <= k; j++)
        if(ans1.find(pd[j] - pd[i - 1]) == ans1.end())
        {
          ans1.insert(pd[j] - pd[i - 1]);
          cnt++;
        }
```

```
    cout << cnt << '\n';
  }
  // quantidade de subarrays com soma diferente

  return 0;
}


//
    ////////////////////////////////////////////////////////////////////////////////////


#include <bits/stdc++.h>

using namespace std;

typedef long double ld;
const double PI = acos(-1);

struct T
{
  ld x, y;
  T() : x(0), y(0) {}
  T(ld a, ld b=0) : x(a), y(b) {}

  T operator/=(ld k) { x/=k; y/=k; return (*this); }
  T operator*(T a) const { return T(x*a.x - y*a.y, x*a.y + y*a.x); }
  T operator+(T a) const { return T(x+a.x, y+a.y); }
  T operator-(T a) const { return T(x-a.x, y-a.y); }
} a[1 << 20], b[1 << 20];

void fft(T* a, int n, int s)
{
  for (int i=0, j=0; i<n; i++)
  {
    if (i>j) swap(a[i], a[j]);
    for (int l=n/2; (j^=l) < l; l>>=1);
  }
  for(int i = 1; (1<<i) <= n; i++)
  {
    int M = 1 << i;
    int K = M >> 1;
    T wn = T(cos(s*2*PI/M), sin(s*2*PI/M));
    for(int j = 0; j < n; j += M)
    {
      T w = T(1, 0);
      for(int l = j; l < K + j; ++l)
      {
        T t = w*a[l + K];
        a[l + K] = a[l]-t;
        a[l] = a[l] + t;
        w = wn*w;
      }
    }
  }
}

void multiply(T* a, T* b, int n)
{
  fft(a,n,1);
  fft(b,n,1);
  for (int i = 0; i < n; i++)
    a[i] = a[i]*b[i];
  fft(a,n,-1);
  for (int i = 0; i < n; i++)
    a[i] /= n;
}

const int base = 10;
vector<int> normalize(vector<int> c)
{
    int carry = 0;
    for(auto &it: c)
    {
        it += carry;
        carry = it / base;
        it %= base;
    }
    while(carry)
    {
        c.push_back(carry % base);
        carry /= base;
    }
    return c;
}

vector<int> faz(string s)
{
  vector<int> ans;
  for(char &c : s)
    ans.push_back(c-'0');
  return ans;
}

string mul(string s1, string s2)
{
  vector<int> A = normalize(faz(s1));
  vector<int> B = normalize(faz(s2));

  int na = A.size(), nb = B.size();
  int n = na + nb;

  while(n&(n-1))
    n++;

  reverse(A.begin(), A.end());
  reverse(B.begin(), B.end());

  while(A.size() < n) A.push_back(0);
  while(B.size() < n) B.push_back(0);

  reverse(A.begin(), A.end());
  reverse(B.begin(), B.end());

  for(int i = 0; i < n; i++)
    a[i] = T(A[i]);
  for(int i = 0; i < n; i++)
    b[i] = T(B[i]);

  multiply(a, b, n);
```

```
  vector<int> r;
  for(int i = 0; i < n - 1; i++)
    r.push_back(a[i].x + 0.5);

  reverse(r.begin(), r.end());

  r = normalize(r);

  while(r.back() == 0)
          r.pop_back();

  reverse(r.begin(), r.end());

  string ans;

  for(int &c: r)
    ans.push_back(c + '0');

  return ans;
}

int main()
{


  return 0;
}
```

5.12 Mulmod Trick

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;

ll mulmod(ll a, ll b, ll m)
{
  ll q = ll((long double)a*b/m);
  ll r = a * b - m * q;
  while(r < 0) r += m;
  while(r >= m) r -= m;
  return r;
}

int main()
{
  ll a, b, c;

  cin >> a >> b >> c;
  cout << mulmod(a, b, c) << '\n';

    return 0;
}
```

5.13 Miller Rabin

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long

ll add(ll a, ll b, ll c)
{
  ll ans = (a + b) % c;
    if(ans < 0) ans += c;
    return ans;
}

ll mulmod(ll a, ll b, ll c)
{
  ll ans = 0;
  while(b)
  {
    if(b & 1) ans = add(ans, a, c);
    a = add(a, a, c);
    b /= 2;
  }
  return ans;
}

ll fexp(ll a, ll b, ll c)
{
  ll ans = 1;
  while(b)
  {
    if(b & 1) ans = mulmod(ans, a, c);
    a = mulmod(a, a, c);
    b /= 2;
  }
  return ans;
}

bool miller(ll a, ll n)
{
    if (a >= n) return true;
    ll s = 0, d = n - 1;
    while(d%2 == 0 and d) d >>= 1, s++;
    ll x = fexp(a, d, n);
    if(x == 1 or x == n - 1) return true;
    for(int r = 0; r < s; r++, x = mulmod(x, x, n))
    {
        if (x == 1) return false;
        if (x == n-1) return true;
    }
    return false;
}

bool isprime(ll n)
{
    int base[] = {2, 3, 5, 7, 11,
       13, 17, 19, 23, 29, 31, 37};
    for(int i = 0; i < 12; i++)
        if(!miller(base[i], n))
            return false;
    return true;
}

int32_t main()
{
    ll n;
    cin >> n;
```

```
    cout << (isprime(n) ? "PRIME\n"
       : "NOT PRIME\n");

    return 0;
}
```

5.14 Mobius

```
/*
- mi(n) = 0 se n tem como divisor um outro numero
natural ao quadrado
- mi(n) = 1 se n nao tem como divisor um outro
numero natural ao quadrado
e eh decomposto em uma quantidade par de
numeros primos
- mi(n) = -1 se n nao tem como divisor um outro
numero natural ao quadrado
e eh decomposto em uma quantidade impar de
numeros primos
*/

#include "bits/stdc++.h"
using namespace std;
const int MAX = 1e6;

bool np[MAX];
int mob[MAX];

void mobius()
{
  for(int i = 1; i < MAX; i++)
    mob[i] = 1;
  for(int i = 2; i < MAX; i++)
  {
    if(np[i]) continue;
    for(int j = i; j < MAX; j += i)
    {
      np[j] = true;
      mob[j] *= -1;
      if((j / i) % i == 0)
        mob[j] = 0;
    }
  }
}

int main()
{
  mobius();

    for(int i = 2; i <= 10; i++)
      cout << i << ' ' << mob[i] << '\n';
  puts("");

  return 0;
}
```

5.15 Conversion Base

```
#include <bits/stdc++.h>
using namespace std;

int a,b; char sa[10000]; char sb[10000];

void rev(char s[])
{
  int l = strlen(s);
  for(int i = 0; i < l - 1 - i; i++)
    swap(s[i], s[l - 1 - i]);
}

void multi(char s[], int k)
{
  int i, c = 0, d;
  for(i=0;s[i];i++)
  {
    d = (s[i] - '0') * k + c;
    c = d / b; d %= b;
    s[i] = '0' + d;
  }
  while(c)
  {
    s[i] = '0' + (c % b); i++;
    c /= b;
  }
  s[i] = '\0';
}

void add(char s[], int k)
{
  int i, c = k, d;
  for(i = 0; s[i]; i++)
  {
    d = (s[i] - '0') + c;
    c = d / b; d %= b;
    s[i] = '0' + d;
  }
  while(c)
  {
    s[i] = '0' + (c % b); i++;
    c /= b;
  }
  s[i]='\0';
}

void trans(char s[])
{
  for(int i = 0; s[i]; i++)
  {
    char& c = s[i];
    if(c >= 'A' && c <= 'Z') c = '0' + 10 + (c - 'A');
    if(c >= 'a' && c <= 'z') c = '0' + 36 + (c - 'a');
  }
}

void itrans(char s[])
{
  for(int i = 0; s[i]; i++)
  {
    char& c = s[i]; int d = c - '0';
```

```
    if(d >= 10 && d <= 35) c = 'A' + (d - 10);
    if(d >= 36) c = 'a' + (d - 36);
  }
}

int main()
{
//digitos {0-9,A-Z,a-z}
  int q; cin>>q;
  int i,j;
  while(q)
  {
    q--;

    cin >> a >> b >> sa; sb[0] = '0'; sb[1] = '\0';
    // a e b sao dados na base 10
    // sa eh dado na base a
    // converter sa da base a pra base b
    cout << a << " " << sa << '\n';
    trans(sa);
    for(i = 0; sa[i]; i++)
    {
      multi(sb, a);
      add(sb, sa[i] - '0');
    }
    rev(sb);
    itrans(sb);
    // sb eh a na base b
    cout << b << " " << sb << '\n';
    puts("");
  }
  return 0;
}
```

6 Useful Scripts

6.1 Stress.sh

```
make sol brute gen

for ((i = 1; ; i++)) do
        ./gen $i > in
        ./sol < in > out
        ./brute < in > out2
        if (! cmp -s out out2) then
                echo "--> entrada:"
                cat in
                echo "--> saida sol"
                cat out
                echo "--> saida2 brute"
                cat out2
                break;
        fi
        echo $i
done
```

7 Graph

7.1 Boruvka MST

```
#include <bits/stdc++.h>
using namespace std;

int n, m;
vector<array<int, 3>> edge;
int pai[100100], sz[100100];

int find(int x)
{
    return pai[x] == x ? x : pai[x] = find(pai[x]);
}

void join(int x, int y)
{
    x = find(x);
    y = find(y);
    if(x == y) return;
    if(sz[x] > sz[y]) swap(x, y);
    pai[x] = y;
    sz[y] += sz[x];
}

int main()
{
    scanf(" %d %d", &n, &m);
    for(int i = 0; i < m; i++)
    {
        int u, v, w;
        scanf(" %d %d %d", &u, &v, &w); u--; v--;
        edge.push_back({w, v, u});
    }
    for(int i = 0; i < n; i++)
        pai[i] = i, sz[i] = 1;
    int mst_cost = 0;
    bool fl = true;
    while(fl)
    {
        fl = false;
        vector<int> aux(n, -1);
        for(int i = 0; i < m; i++)
        {
            int u = find(edge[i][1]), v = find(edge[i][2]), w = edge[i
                ][0];
            if(u == v) continue;
            if(aux[u] == -1) aux[u] = i;
            else if(edge[aux[u]][0] > w) aux[u] = i;
            if(aux[v] == -1) aux[v] = i;
            else if(edge[aux[v]][0] > w) aux[v] = i;
        }
        for(int i = 0; i < n; i++)
        {
            if(aux[i] == -1) continue;
            int u = find(edge[aux[i]][1]), v = find(edge[aux[i]][2]);
            if(u == v) continue;
```

```
            // add_edge edge[aux[i]][1] --- edge[aux[i]][2] in the MST
            join(u, v);
            mst_cost += edge[aux[i]][0];
            fl = true;
        }
    }
    cout << mst_cost << '\n';

    return 0;
}
```

7.2 2SAT

```
// Os vertices pares indicam as proposicoes falsas
// Os vertices impares indicam as proposicoes verdadeiras
// Achar qual proposicao relativa a cada vertice, eh so dividiar
    vertice/2
// tamG = quantidade_proposicoes*2

#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e3;

int n, m, tamG;
vector<int> G[MAX], G_t[MAX], C[MAX];
stack<int> sta;
bool cor[MAX];
int componente[MAX], comp;

void preenche(int v)
{
    cor[v] = true;
    for(const int &u : G_t[v])
        if(!cor[u])
            preenche(u);
    sta.push(v);
}

void dfs(int v, int comp)
{
    componente[v] = comp;
    C[comp].push_back(v);
    for(const int &u : G[v])
        if(!componente[u])
            dfs(u, comp);
}

void kosaraju()
{
    memset(cor, false, sizeof(cor));
    for(int i = 0; i < tamG; i++)
        if(!cor[i])
            preenche(i);
    memset(cor, false, sizeof(cor));
    comp = 1;
    while(!sta.empty())
    {
        int u = sta.top();
        sta.pop();
        if(componente[u]) continue;
        dfs(u, comp);
        comp++;
    }
}



// Id no grafo que representa a proposicao de numero P como verdadeira
    .
int idTrue(int p)
{
    return (p << 1) + 1;
}

// Id no grafo que representa a proposicao de numero P como falsa.
int idFalse(int p)
{
    return (p << 1);
}

bool twoSat()
{
  kosaraju();
  for(int i = 0; i < tamG; i+=2)
  {
  // Todo par de proposicoes(proposicao falsa, proposicao verdadeira)
  // Nao podem estar no mesmo componente
    if(componente[i] == componente[i + 1])
        return false;
  }
  return true;
}

int addEdge(int u, int v)
{
    G[idFalse(u)].push_back(idTrue(v));
    G[idFalse(v)].push_back(idTrue(u));
    G[idTrue(u)].push_back(idFalse(v));
    G[idTrue(v)].push_back(idFalse(u));
    // montar grafo transposto para kosaraju nessa
    // aplicacao o grafo G sera igual ao transposto
    G_t[idFalse(u)].push_back(idTrue(v));
    G_t[idFalse(v)].push_back(idTrue(u));
    G_t[idTrue(u)].push_back(idFalse(v));
    G_t[idTrue(v)].push_back(idFalse(u));
}



vector<int> g[MAX];
vector<int> ts;
int value[MAX];

void topSort(int v)
{
    cor[v] = true;
    for(int &u : G[v])
        if(!cor[u])
            topSort(u);
    ts.push_back(v);
}
```

```
void mountDAG()
{
    for(int v = 0; v < tamG; v++)
        for(int &u : G[v])
            if(componente[v] != componente[u])
                g[componente[v]].push_back(componente[u]);
    memset(cor, false, sizeof(cor));
    for(int v = 1; v < comp; v++)
        if(!cor[v])
            topSort(v);
    // nao inverter ts, pois precisamos da ordenacao
    // topologica ao contrario
}

// encontrar uma atribuicao (TREU ou FALSE) para as proposicoes
void assignment()
{
    if(!twoSat()) return;
    mountDAG();
    memset(value, -1, sizeof(value));
    for(int &v : ts)
        for(int &u : C[v])
            if(value[u >> 1] == -1) //   u / 2 eh a proposicao
                value[u >> 1] = (u & 1 ? 1 : 0);
    for(int i = 0; i < (tamG >> 1); i++)
        cout << value[i] << ' ';
    puts("");
}

int main()
{
    cin >> n >> m;
    tamG = 2 * n;
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v; u--; v--;
        addEdge(u, v);
    }
    cout << twoSat() << '\n';
    assignment();

    return 0;
}
```

7.3 Longest And Shortest Path In DAG

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

const int OO = 0x3f3f3f3f;
const int MAX = 1e6;

using namespace std;

int n, m;
vector<pair<int, int>> G[MAX];
int dist1[MAX], dist2[MAX];
vector<int> ts;
bool cor[MAX];

void dfs(int v)
{
  cor[v] = true;
  for(pair<int, int> &w : G[v])
    if(!cor[w.first])
      dfs(w.first);
  ts.push_back(v);
}

// caminho de 0 a n-1
pair<int, int> longestAndShortestPathInDAG()
{
  for(int i = 0; i <= n; i++)
    dist1[i] = -OO, dist2[i] = OO;
  dist1[0] = dist2[0] = 0;
  int p = 0;
  while(p < (int)ts.size())
  {
    int v = ts[p++];
    if(dist1[v] != -OO)
      for(int i = 0; i < (int)G[v].size(); i++)
      {
        int u = G[v][i].first, d = G[v][i].second;
        if(dist1[u] < dist1[v] + d)
          dist1[u] = dist1[v] + d;
      }
    if(dist2[v] != OO)
      for(int i = 0; i < (int)G[v].size(); i++)
      {
        int u = G[v][i].first, d = G[v][i].second;
        if(dist2[u] > dist2[v] + d)
          dist2[u] = dist2[v] + d;
      }
  }
  return {dist1[n-1], dist2[n-1]};
}

int main()
{
  cin >> n >> m;
  for(int i = 0; i < m; i++)
  {
    int u, v, w;
    cin >> u >> v >> w; u--; v--;
    G[u].push_back({v, w});
  }
  for(int i = 0; i < n; i++)
    if(!cor[i])
      dfs(i);
  reverse(ts.begin(), ts.end());
  pair<int, int> ans = longestAndShortestPathInDAG();
  cout << "Longest Path " << ans.first << '\n';
  cout << "Shortest Path " << ans.second << '\n';

  return 0;
}
```

7.4 Knapsack Dijkstra

```cpp
#include <bits/stdc++.h>
using namespace std;
const int OO = 0x3f3f3f3f;
#define ii pair<int, int>
#define fi first
#define se second

vector<ii> G[105];
int dist[100000008];
vector<int> peso;

void dijkstra()
{
  memset(dist, 63, sizeof(dist));
  dist[0] = 0;
  priority_queue<ii> pq;
  pq.push({0, 0});
  while(!pq.empty())
  {
    int u = pq.top().se;
    int d = -pq.top().fi;
    pq.pop();
    if(d > dist[u]) continue;
    for(int i = 0; i < G[u].size(); i++)
    {
      int w = G[u][i].fi, dd = G[u][i].se;
      if(dist[w] > dist[u] + dd)
      {
        dist[w] = dist[u] + dd;
        pq.push({-dist[w], w});
      }
    }
  }
  dist[0] = peso[0];
}

int32_t main()
{
  int n, e, d;
  cin >> n >> d >> e;
  // ler pesos
  peso = vector<int>{d, 2 * d, 5 * d, 10 * d, 20 * d, 50 * d, 100 * d,
    5 * e, 10 * e, 20 * e, 50 * e, 100 * e, 200 * e};
  // ordena pra pegar o menor valor
  sort(peso.begin(), peso.end());
  //montar grafo
  for(int i = 0; i < peso[0]; i++)
    for(int j = 0; j < peso.size(); j++)
    {
      int x = (i + peso[j]) % peso[0];
      G[i].push_back({x, peso[j]});
    }
  /*
    dist[i] eh o menor numero que eu consigo formar usando
    os meus objetos tal que dist[i] % peso[0] == i
  */
  dijkstra();

  /*
    se dist[X % peso[0]] <= X eh possivel gerar um valor X
    utilizando os valores do array peso
    OBS: cada valor pode ser usado infinitas vezes
  */
  return 0;
}
```

7.5 Eulirian Path

```cpp
#include <bits/stdc++.h>
using namespace std;

int32_t main() {

   int n, m;

   cin >> n >> m;
   vector<vector<int>> g(n);
   vector<int> deg_in(n), deg_out(n);

   for(int i = 0; i < m; i++) {
      int u, v;
      cin >> u >> v; u--; v--;
      g[u].push_back(v);
      deg_in[v]++;
      deg_out[u]++;
   }

   int s = -1, f = -1;
   for(int i = 0; i < n; ++i) {
      if(deg_in[i] - deg_out[i] == 0) continue;

      if(s == -1 and deg_out[i] - deg_in[i] == 1) s = i;
      else if(f == -1 and deg_in[i] - deg_out[i] == 1) f = i;
      else return cout << "NO\n", 0;
   }

   if(s == -1 and f == -1) s = 0;
   else if(s != -1 and f == -1 or s == -1 and f != -1) return cout <<
        "NO\n", 0;

   stack<int> st;
   st.push(s);
   vector<int> res;

   while(!st.empty()) {
      int v = st.top();
      if(g[v].empty()) {
         res.push_back(v);
         st.pop();
      } else {
         int u = g[v].back();
         g[v].pop_back();
         st.push(u);
      }
   }

   for(int i = 0; i < n; i++)
      if(g[i].empty() == false)
         return cout << "NO\n", 0;

   reverse(res.begin(), res.end());
```

```
    for(int w : res)
       cout << w + 1 << ' ';
    cout << endl;

  return 0;
}
```

7.6 Tree Isomorfism

```
#include <bits/stdc++.h>
using namespace std;

const int ms = 100100;

int degree[ms], vis[ms];
int size[ms];
int n;

bool cmp(int a, int b)
{
  return size[a] < size[b];
}

void pre(vector<vector<int>> &edges, int on = 0)
{
  size[on] = 1;
  for(auto to : edges[on])
  {
    pre(edges, to);
    size[on] += size[to];
  }
  sort(edges[on].begin(), edges[on].end(), cmp);
}

void solve(vector<vector<int>> &edges, string &str, int on = 0)
{
  str += 'D';
  for(int l = 0, r = 0; l < edges[on].size(); l = r) {
    while(r < edges[on].size() &&
      size[edges[on][l]] == size[edges[on][r]]) r++;
    if(r == l + 1)
      solve(edges, str, edges[on][l]);
    else
    {
      priority_queue<string> hp;
      for(int i = l; i < r; i++) {
        string temp;
        solve(edges, temp, edges[on][i]);
        hp.push(temp);
      }
      while(!hp.empty())
      {
        str += hp.top();
        hp.pop();
      }
    }
  }
  str += 'U';
}

// enraizar arvore
void mount(vector<vector<int>> &graph,
  vector<vector<int>> &G, int v = 0, int p = -1)
{
  for(int &u : G[v])
    if(u != p)
    {
      graph[v].push_back(u);
      mount(graph, G, u, v);
    }
}

// achar centro da arvore e enraizar no centro
void findCenterAndComputeStr(vector<vector<int>> &graph,
  vector<vector<int>> &G, string *str)
{
  memset(vis, 0, sizeof(vis));
  queue<int> fila[2];
  for(int i = 0; i < n; i++)
    if(degree[i] == 1)
      fila[0].push(i);
  int cnt = 0, turn = 0;
  while(cnt + 2 < n)
  {
    while(!fila[turn].empty())
    {
      int u = fila[turn].front(); fila[turn].pop();
      vis[u] = true;
      cnt++;
      for(int i = 0; i < G[u].size(); i++)
        if(!vis[G[u][i]])
        {
          degree[G[u][i]]--;
          if(degree[G[u][i]] == 1)
            fila[1-turn].push(G[u][i]);
        }
    }
    turn ^= 1;
  }
  int k = 0;
  for(int i = 0; i < n; i++)
  {
    if(vis[i]) continue;
    graph.clear();
    graph.resize(n + 1);
    mount(graph, G, i);
    pre(graph, i);
    solve(graph, str[k], i);
    k++;
  }
}

int main()
{
  while(cin >> n)
  {
    string str[2][2];
    for(int i = 0; i < 2; i++)
    {
      vector<vector<int>> graph, G;
```

```
      G.resize(n + 1);
      memset(degree, 0, sizeof(degree));
      for(int j = 1; j < n; j++)
      {
        int u, v;
        scanf(" %d %d", &u, &v); v--; u--;

        G[v].push_back(u);
        G[u].push_back(v);
        degree[v]++;
        degree[u]++;
      }
      findCenterAndComputeStr(graph, G, str[i]);
    }
    bool fl = (str[0][0] == str[1][0]) or (str[0][0] == str[1][1]);
    fl |= ((str[0][1] == str[1][0]) or (str[0][0] == str[1][1]));
    puts(fl ? "S" : "N");
  }

  return 0;
}
```

7.7 K Short Paths

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
const int OO = 0x3f3f3f3f3f3f3f3f;
const int MAX = 2000000;

typedef pair<int, int> ii;

int n, m, k;
vector<ii> G[MAX];
int cnt[MAX];

void dijkstra(int v) {
  priority_queue<ii> pq;
  pq.push({0, v});
  int c = 0;
  while(!pq.empty()) {
    int u = -pq.top().second;
    int d = -pq.top().first;
    pq.pop();
    cnt[u]++;
    if(cnt[u] > k) continue;
      if(u == n - 1) {
        cout << d << ' ';
      if(++c == k) { cout << '\n'; return; }
    }
    for(auto [_d, w] : G[u])
      if(cnt[w] < k)
        pq.push({-(d + _d), -w});
    }
}

int32_t main() {
  cin >> n >> m >> k;
  while(m--) {
    int u, v, w;
    cin >> u >> v >> w; u--; v--;
    G[u].push_back({w, v});
  }
  dijkstra(0);

  return 0;
}
```

7.8 Hopcroft Karp

```
#include <bits/stdc++.h>
using namespace std;
const int OO = 0x3f3f3f3f;

int n, m;
vector<int> G[10000];
queue<int> q;
int pairU[10000], pairV[10000], dist[10000];

bool bfs()
{
    for(int u = 1; u <= m; u++)
        if(!pairU[u])
        {
            dist[u] = 0;
            q.push(u);
        }
        else dist[u] = OO;
    dist[0] = OO;
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        if(dist[u] < dist[0])
            for(const int &v : G[u])
                if(dist[pairV[v]] == OO)
                {
                    dist[pairV[v]] = dist[u] + 1;
                    q.push(pairV[v]);
                }
    }
    return (dist[0] != OO);
}

bool dfs(int u)
{
    if(u)
    {
    for(const int &v : G[u])
            if(dist[pairV[v]] == dist[u]+1)
                if (dfs(pairV[v]))
                {
                    pairV[v] = u;
                    pairU[u] = v;
                    return true;
                }
        dist[u] = OO;
        return false;
    }
    return true;
```

```
}

int hopcroftKarp()
{
    memset(pairU, 0, sizeof(pairU));
    memset(pairV, 0, sizeof(pairV));
    int result = 0;
    while(bfs())
        for(int u = 1; u <= m; u++)
            if(!pairU[u] and dfs(u))
                result++;
    return result;
}

int main()
{
  n = m = 4;

  G[1].push_back(2);
  G[2].push_back(1);
  G[1].push_back(3);
  G[3].push_back(1);
  G[2].push_back(1);
  G[1].push_back(2);
  G[3].push_back(2);
  G[2].push_back(3);
  G[4].push_back(2);
  G[2].push_back(4);
  G[4].push_back(4);
  G[4].push_back(4);

  cout << hopcroftKarp() << '\n';

  return 0;
}
```

7.9 Diameter And Center Of A Tree

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

int n;
vector<int> G[MAX];

int bfs(int v, vector<int> &dist)
{
    queue<int> q;
    q.push(v);
    dist[v] = 0;
    int lgv = -1;
    while(!q.empty())
    {
        int u = q.front(); q.pop();
        lgv = u;
        for(int &w : G[u])
            if(dist[w] == -1)
            {
                dist[w] = dist[u] + 1;
                q.push(w);
            }
    }
    return lgv;
}

void findCenterAndDiameter(int w)
{
    vector<int> dist1(n + 1, -1);
    vector<int> dist2(n + 1, -1);
  int v = bfs(w, dist1);
  int u = bfs(v, dist2);
  int d = dist2[u];
    dist1.assign(n + 1, -1);
    u = bfs(u, dist1);
    cout << "center ";
    for(int i = 0; i < n; i++)
    {
        int d1 = dist1[i], d2 = dist2[i];
        if(d1 == d / 2 and d2 == d - d / 2 or d2 == d / 2 and d1 == d
            - d / 2)
        cout << i + 1 << ' ';
    }
    cout << "\ndiameter " << d << '\n';
}

int main()
{
    cin >> n;
    for(int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v; u--; v--;
        G[u].push_back(v);
    G[v].push_back(u);
    }
    findCenterAndDiameter(0);

    return 0;
}
```

7.10 Dijkstra

```
#include <bits/stdc++.h>
using namespace std;
const int OO = 0x3f3f3f3f;
const int MAX = 1e5;

typedef pair<int, int> ii;

int n, m;
int dist[MAX];
vector<ii> G[MAX];

int dijkstra(int v, int z)
{
    memset(dist, 63, sizeof(dist));
    dist[v] = 0;
    priority_queue<ii> pq;
    pq.push({0, v});
    while(!pq.empty())
```

```
    {
        int u = pq.top().second;
        int d = -pq.top().first;
        pq.pop();
        if(d > dist[u]) continue;
        if(u == z) return d;
        for(int i = 0; i < G[u].size(); i++)
        {
            int w = G[u][i].second, _d = G[u][i].first;
            if(dist[w] > d + _d)
            {
                dist[w] = d + _d;
                pq.push({-dist[w], w});
            }
        }
    }
    return OO;
}

int main()
{
    int u, v, w;

    cin >> n >> m;
    while(m--)
    {
        cin >> u >> v >> w;
        u--; v--;
        G[u].push_back({w, v});
        G[v].push_back({w, u});
    }
    cin >> u >> v;
    cout << dijkstra(u-1, v-1) << '\n';

    return 0;
}
```

7.11 BFS Zero One

```
// o peso das arestas eh 0 ou 1

#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;
const int OO = 0x3f3f3f3f;
typedef pair<int, int> ii;

int n, m;
vector<ii> G[MAX];
int dist[MAX];
deque<int> dq;

void zeroOneBfs(int v)
{
    memset(dist, 63, sizeof(dist));
    dist[v] = 0;
    dq.push_back(v);
    while(!dq.empty())
    {
        int u = dq.front();
        dq.pop_front();
        for(int i = 0; i < G[u].size(); i++)
        {
            int w = G[u][i].first, d = G[u][i].second;
            if(dist[w] > dist[u] + d)
            {
                dist[w] = dist[u] + d;
                if(!d) dq.push_front(w);
                else dq.push_back(w);
            }
        }
    }
    for(int i = 0; i < n; i++)
        cout << dist[i] << ' ';
    puts("");
}

int main()
{
    cin >> n >> m;
    while(m--)
    {
        int u, v, w;
        cin >> u >> v >> w; u--; v--;
        G[u].push_back({v, w});
        G[v].push_back({u, w});
    }
    zeroOneBfs(0);

    return 0;
}
```

7.12 MPC MinimumPathCover

```
#include <bits/stdc++.h>
using namespace std;

int n, m;
vector<int> G[1000], bip[1000], ts;
int vis[1000], b[1000], go[1000], tempo = 1;

bool kuhn(int v)
{
  if(vis[v] == tempo)
    return 0;
  vis[v] = tempo;
  for(const int &u : bip[v])
    if(!b[u] or kuhn(b[u]))
    {
      go[v] = u - n;
      return b[u] = v;
    }
  return 0;
}

void topological_sort(int v)
{
  vis[v] = tempo;
  for(const int &u : G[v])
    if(vis[u] != tempo)
```

```
      topological_sort(u);
  ts.push_back(v);
}

int main()
{
  cin >> n >> m;
  while(m--)
  {
    int u, v;
    cin >> u >> v;
    G[u].push_back(v);
    bip[u].push_back(v + n);
  }
  int ans = 0;
  for(int i = 1; i <= n; i++)
    ans += kuhn(i), tempo++;
  for(int i = 1; i <= n; i++)
    if(vis[i] != tempo)
      topological_sort(i);
  reverse(ts.begin(), ts.end());
  tempo++;
  cout << n - ans << '\n';
  for(int i = 0; i < n; i++)
  {
    int u = ts[i];
    if(vis[u] != tempo)
    {
      while(u)
      {
        vis[u] = tempo;
        cout << u << ' ';
        u = go[u];
      }
      puts("");
    }
  }

    return 0;
}
```

7.13 Binary Lifting

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

int n, m, nivel[MAX], anc[MAX][30], MAX_LOG;
vector<int> G[MAX];

void dfs(int v, int p, int d)
{
    anc[v][0] = p;
    nivel[v] = d;
    if(d) MAX_LOG = max(MAX_LOG, (int)log2(d));
    for(const int &u : G[v])
        if(u != p)
            dfs(u, v, d + 1);
}

int walk(int v, int k)
{
    while(k) v = anc[v][(int)log2(k&-k)], k -= k&-k;;
    return v;
}

int lca(int u, int v)
{
    if(nivel[u] < nivel[v]) v = walk(v, nivel[v]-nivel[u]);
    if(nivel[u] > nivel[v]) u = walk(u, nivel[u]-nivel[v]);
    if(u == v) return u;
    for(int i = MAX_LOG; i >= 0; i--)
        if(anc[u][i] != anc[v][i])
        {
            u = anc[u][i];
            v = anc[v][i];
        }
    return anc[u][0];
}

void build()
{
    memset(anc, -1, sizeof anc);
    nivel[0] = 0;
    dfs(0, -1, 0);
    for(int j = 1; j <= MAX_LOG; j++)
        for(int i = 1; i <= n; i++)
            if(anc[i][j-1] != -1)
                anc[i][j] = anc[anc[i][j-1]][j-1];
}

int main()
{
    int u, v;

    cin >> n >> m;
    while(m--)
    {
        cin >> u >> v;
        u--; v--;
        G[u].push_back(v);
        G[v].push_back(u);
    }
    build();
    cin >> u >> v;
    cout << lca(u-1, v-1)+1 << '\n';

    return 0;
}


//////////////////////////////////////////


#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;
typedef pair<int, int> ii;

int n, m, max_log;
vector<ii> G[MAX];
```

```
int anc[MAX][30], min_edge[MAX][30], depth[MAX];

void dfs(int v, int d, int p, int a)
{
    anc[v][0] = p;
    depth[v] = d;
    if(d) max_log = max(max_log, (int)log2(d));
    if(p != -1) min_edge[v][0] = a;
    for(int i = 0; i < G[v].size(); i++)
    {
        int u = G[v][i].second, w = G[v][i].first;
        if(u != p)
            dfs(u, d + 1, v, w);
    }
}

void build()
{
    memset(anc, -1, sizeof(anc));
    memset(min_edge, 63, sizeof(min_edge));
    dfs(0, 0, -1, -1);
    for(int j = 1; j <= max_log; j++)
        for(int i = 0; i < n; i++)
        {
            if(anc[i][j-1] != -1)
            {
                anc[i][j] = anc[anc[i][j-1]][j-1];
                min_edge[i][j] = min(min_edge[i][j-1], min_edge[anc[i
                    ][j-1]][j-1]);
            }
        }
}

int walk(int v, int k)
{
    while(k) v = anc[v][(int)log2(k&-k)], k -= k&-k;
    return v;
}

int lca(int u, int v)
{
    if(depth[u] > depth[v]) u = walk(u, depth[u]-depth[v]);
    if(depth[u] < depth[v]) v = walk(v, depth[v]-depth[u]);
    if(u == v) return u;
    for(int i = max_log; i >= 0; i--)
        if(anc[u][i] != anc[v][i])
        {
            u = anc[u][i];
            v = anc[v][i];
        }
    return anc[u][0];
}

int queryMinEdge(int u, int v)
{
    int LCA = lca(u, v);
    int ans = INT_MAX;
    int k = depth[u]-depth[LCA];
    while(k)
    {
        ans = min(ans, min_edge[u][(int)log2(k&-k)]);
        u = walk(u, k&-k);
        k -= k&-k;
    }
    k = depth[v]-depth[LCA];
    while(k)
    {
        ans = min(ans, min_edge[v][(int)log2(k&-k)]);
        v = walk(v, k&-k);
        k -= k&-k;
    }
    return ans;
}

int main()
{
    int u, v, w;

    cin >> n;
    for(int i = 0; i < n-1; i++)
    {
        cin >> u >> v >> w;
        u--; v--;
        G[u].push_back({w, v});
        G[v].push_back({w, u});
    }
    cin >> u >> v;
    build();
    cout << lca(u-1, v-1)+1 << '\n';
    cout << queryMinEdge(u-1, v-1) << '\n';

    return 0;
}
```

7.14 Lca With Square Root Decomposition

```
#include <bits/stdc++.h>
const int MAX = 50500;

using namespace std;

vector<int> G[MAX];
int nivel[MAX], pai[MAX], jump[MAX], n, blk_sz;

void dfs(int v, int d, int p)
{
    pai[v] = p;
    nivel[v] = d;
    (nivel[v]%blk_sz == 0) ? jump[v] = pai[v] : jump[v] = jump[p];
    for(const int &u : G[v])
        if(u != p)
            dfs(u, d + 1, v);
}

int lcaTrivial(int u, int v)
{
    while(u != v)
        (nivel[u] > nivel[v]) ? u = pai[u] : v = pai[v];
    return u;
}
```

```
int lca(int u, int v)
{
    while(jump[u] != jump[v])
        (nivel[u] > nivel[v]) ? u = jump[u] : v = jump[v];
    return lcaTrivial(u, v);
}

void build()
{
    blk_sz = sqrt(n);
    dfs(0, 0, 0);
}

int main()
{
    int x, y;
    cin >> n;
    for(int i = 0; i < n-1; i++)
    {
        cin >> x >> y;
        G[x-1].push_back(y-1);
        G[y-1].push_back(x-1);
    }
    build();
    cin >> x >> y;
    cout << lca(x-1, y-1) + 1 << '\n';

    return 0;
}
```

7.15 Fully Dynamic Connectivity Check If Two Vertices Are In The Same Component

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 131072;
const int OO = 0x3f3f3f3f;
const double EPS = 1e-9;

#define bug(x) cout << #x << " = " << x << '\n'
#define FOR(i, a, n) for(int i = a; i < n; i++)
#define REP(i, n) FOR(i, 0, n)
#define fi first
#define se second
#define pb push_back
#define mt make_tuple
#define mp make_pair
#define all(vetor) vetor.begin(), vetor.end()
#define X real()
#define Y imag()
//#define gc getchar_unlocked

typedef long long ll;
typedef long double ld;
//typedef pair<int, int> ii;
//typedef pair<int, ii> iii;
typedef complex<ll> Pll;
typedef complex<ld> Pld;


typedef pair<int, int> edge;

vector<edge> tree[4 * MAX], query[4 * MAX];;
int pai[MAX], sz[MAX];
stack<pair<int, int>> stk, size;
vector<int> ans;

int find(int x)
{
    if(pai[x] == x)
        return x;
    stk.push(mp(x, pai[x]));
    size.push(mp(x, sz[x]));
    return pai[x] = find(pai[x]);
}

void join(int x, int y)
{
    x = find(x);
    y = find(y);
    if(x == y) return;
    if(sz[x] > sz[y])
        swap(x, y);
    stk.push(mp(x, pai[x]));
    size.push(mp(y, sz[y]));
    sz[y] += sz[x];
    pai[x] = y;
}

void rollback(int rollback_to)//desfaz todas as alteracaes no DSU,
{//O(k) onde k eh a quantidade de operacoes realizadas
  while(rollback_to < stk.size())
  {
    pai[stk.top().fi] = stk.top().se;
    stk.pop();
        sz[size.top().fi] = size.top().se;
        size.pop();
  }
}

void add_edge(int node, int start, int end, int l, int r, edge e)
{
    if(start == l and end == r)
    {
        tree[node].push_back(e);
        return;
    }
    if(l >= r)
        return;
    int mid = (start + end) / 2;
    add_edge(2*node, start, mid, l, min(mid, r), e);
    add_edge(2*node + 1, mid + 1, end, max(l, mid + 1), r, e);
}

void add_query(int node, int start, int end, int idx, edge e)
{
    if(start == end)
        query[node].push_back(e);
```

```
    else
    {
        int mid = (start + end) / 2;
        if(idx <= mid)
            add_query(2*node, start, mid, idx, e);
        else
            add_query(2*node + 1, mid + 1, end, idx, e);
    }
}

void processar(int node)
{
    for(auto it : tree[node])
        join(it.first, it.second);
}

void dfs(int node, int start, int end)
{
    int rollback_to = stk.size();
    processar(node);
    if(start == end)
    {
        for(auto v : query[node])
        {
            bool rep = (find(v.first) == find(v.second));
            ans.push_back(rep);
        }
    }
    else
    {
        int mid = (start + end) / 2;
        dfs(2*node, start, mid);
        dfs(2*node + 1, mid + 1, end);
    }
    rollback(rollback_to);
}

int main()
{
    int n, q, o, u, v;

    cin >> n >> q;
    for(int i = 0; i <= n; i++)
        sz[i] = 1, pai[i] = i;
    int cur = 0;
    map<pair<int, int>, int> mapa;
    while(q--)
    {
        cin >> o >> u >> v; u--; v--;
    if(u > v) swap(u, v);
        if(o == 1)// adicionar aresta
            mapa[mp(u, v)] = cur++;
        else if(o == 2)// remover aresta
        {
            add_edge(1, 0, MAX-1, mapa[mp(u, v)], cur++, mp(u, v));
            mapa.erase(mp(u, v));
        }
        else // verificar se dois vertices estao na mesma componente
            add_query(1, 0, MAX-1, cur++, mp(u, v));
    }
    cur++;
    for(auto it : mapa)
        add_edge(1, 0, MAX-1, it.second, cur, it.first);
    dfs(1, 0, MAX-1);
    for(int i = 0; i < ans.size(); i++)
        cout << (ans[i] ? "Yes\n" : "No\n");

    return 0;
}
```

7.16 Floyd Sucessor Graph

```
#include <bits/stdc++.h>
using namespace std;

int n;
int table[10000][20];

//table[i][j] armazena o sucessor de distancia 2^j do vertice i
void build()
{
  for(int j = 1; (1 << j) <= n; j++)
    for(int i = 0; i < n; i++)
        if(table[i][j-1] != -1)
          table[i][j] = table[table[i][j-1]][j-1];
}

int succ(int u, int k)
{
  while(k)
  {
    u = table[u][(int)log2(k&-k)];
    if(u == -1)
        return -1;// nao existe
    k -= k&-k;
  }
  return u;
}
////////////////////////////////////////////////////


//algoritmo de Floyd para encontrar o tamanho de um ciclo
//alcancado a partir de um vertice u em um grafo sucessor
int Floyd(int u)
{
  int a = succ(u, 1);
  int b = succ(u, 2);

  //encontra um vertice no ciclo
  while(a != b)
  {
    a = succ(a, 1);
    b = succ(b, 2);
    if(a == -1 or b == -1)
        return -1;// nao existe ciclo
  }

  //a e b vao ficar posicionados no inicio do ciclo
  a = u;
  while(a != b)
  {
```

```
      a = succ(a, 1);
      b = succ(b, 1);
  }

  //percorre todo o ciclo contando o seu tamanho
  b = succ(a, 1);
  int lenght = 1;
  while(a != b)
  {
    b = succ(b, 1);
    lenght++;
  }
  return lenght;
}
////////////////////////////////////////////////////


int main()
{
  int u, v, m;

  cin >> n >> m;
  memset(table, -1, sizeof(table));
  for(int i = 0; i < m; i++)
  {
    cin >> u >> v; u--; v--;
    table[u][0] = v;
  }
  build();
  cin >> u >> v;
  cout << "O sucessor de " << u << " com " << v <<
  " unidades a frente eh " << succ(u-1, v)+1 << '\n';

  cout << '\n';

  cin >> u;
  cout << "tamanho do ciclo iniciando em " << u <<
   ": " << Floyd(u-1) << '\n';

  return 0;
}
```

7.17 MCE MinimumEdgeCover

```
#include <bits/stdc++.h>
using namespace std;

vector<int> G[1000];
int b[1000], vis[1000], tempo;

bool kuhn(int v)
{
  if(vis[v] == tempo)
    return 0;
  vis[v] = tempo;
  for(const int &u : G[v])
    if(!b[u] or kuhn(b[u]))
      return b[u] = v;
  return 0;
}

int main()
{
  int n, m, e;
  cin >> n >> m >> e;
  while(e--)
  {
    int u, v;
    cin >> u >> v;
    G[u].push_back(v + n);
  }
  int ans = 0;
  tempo = 1;
  for(int i = 1; i <= n; i++)
    ans += kuhn(i), tempo++;


  //encontrar as arestas do Minimum Edge Cover
  vector<bool> covered(n + m + 10, false);
    vector<pair<int, int>> cover;
  for(int i = n + 1; i <= n + m; i++)
    if(b[i])
    {
      covered[b[i]] = covered[i] = true;
      cover.push_back({b[i], i - n});
    }
  for(int i = 1; i <= n; i++)
  {
    bool is_covered = covered[i];
    for(const int &u : G[i])
      if(!covered[u])
      {
        is_covered = true;
        cover.push_back({i, u - n});
        covered[i] = covered[u] = true;
      }
    if(!is_covered and !G[i].empty())
      cover.push_back({i, G[i].front() - n});
  }
  cout << "MEC = " << cover.size() << '\n';
  for(int i = 0; i < cover.size(); i++)
    cout << cover[i].first << ' ' << cover[i].second << '\n';

    return 0;
}
```

7.18 Maximum Clique

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAX = 43, C = 20;

int n, m, dp[1 << C];
ll G[MAX];

int maxClique()
{
  for(int i = 1; i < (1 << max(0, n - C)); i++)
  {
```

```
    int x = i;
    for(int j = 0; j < max(0, n - C); j++)
      if((i >> j) & 1)
        x &= G[j + C] >> C;
    if(x == i) dp[i] = __builtin_popcount(i);
  }
  for(int i = 1; i < (1 << max(0, n - C)); i++)
    for(int j = 0; j < max(0, n - C); j++)
      if((i >> j) & 1)
        dp[i] = max(dp[i], dp[i ^ (1 << j)]);
  int ans = 0;
  for(int i = 0; i < (1 << min(C, n)); i++){
    int x = i, y = (1 << max(0, n - C)) - 1;
    for(int j = 0; j < min(C, n); j++)
      if((i >> j) & 1)
        x &= G[j], y &= G[j] >> C;
    if(x == i)
        ans = max(ans, __builtin_popcount(i) + dp[y]);
  }
  return ans;
}

int main()
{
    cin >> n >> m;
    while(m--)
    {
        int u, v;
        cin >> u >> v; u--; v--;
        G[u] |= (1LL << v);
        G[v] |= (1LL << u);
    }
    for(int i = 0; i < n; i++)
        G[i] |= (1LL << i);
    cout << maxClique() << '\n';

  return 0;
}
```

7.19 MVC MinimumVertexCover

```
#include <bits/stdc++.h>
using namespace std;

vector<int> G[1000];
int b[1000], vis[1000], tempo;
bool be[1000];
set<int> r0, r1;

bool kuhn(int v)
{
  if(vis[v] == tempo)
    return 0;
  vis[v] = tempo;
  for(const int &u : G[v])
    if(!b[u] or kuhn(b[u]))
      return b[u] = v;
  return 0;
}

void MVC(int v)
{
  if(vis[v] == tempo)
    return;
  vis[v] = tempo;
  for(const int u : G[v])
    if(b[u] != v and b[u])
    {
      r1.insert(u);
      vis[b[u]] = tempo;
    }
}

int main()
{
  int n, m, e;
  cin >> n >> m >> e;
  while(e--)
  {
    int u, v;
    cin >> u >> v;
    G[u].push_back(v);
  }
  int ans = 0;
  tempo = 1;
  for(int i = 1; i <= n; i++)
    ans += kuhn(i), tempo++;
  for(int i = n + 1; i <= n + m; i++)
    if(b[i])
      be[i - n] = be[b[i]] = true;
  for(int i = 1; i <= n; i++)
    if(!be[i])
      MVC(i);
  for(int i = 1 ; i <= n; i++)
    if(vis[i] < tempo)
      r0.insert(i);
  cout << "MVC = "<< ans << '\n';
  cout << "tamanho lado esquerdo " << r0.size() << '\n';
  for(auto it = r0.begin(); it != r0.end(); it++)
    cout << *it << ' ';
  puts("");
  cout << "tamanho lado direito " << r1.size() << '\n';
  for(auto it = r1.begin(); it != r1.end(); it++)
    cout << *it - n << ' ';
  puts("");

    return 0;
}
```

7.20 Lca With Tree Linearization And Sparse Table

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5 + 10;

int n, m;
vector<int> G[MAX], tl;
int deft[MAX], SpT[27][MAX], pos[MAX];

void tree_linearization(int v, int p, int d)
```

```
{
  deft[v] = d;
  pos[v] = tl.size();
  tl.push_back(v);
  for(int &u : G[v])
  {
    if(u != p)
    {
      tree_linearization(u, v, d + 1);
      tl.push_back(v);
    }
  }
}

void build(int tam)
{
  for(int i = 0; (1 << i) <= tam; i++)
  {
    for(int j = 0; j + (1 << i) <= tam; j++)
      if(!i)
        SpT[i][j] = tl[j];
      else if(deft[SpT[i-1][j]] < deft[SpT[i-1][j+(1<<(i-1))]])
        SpT[i][j] = SpT[i-1][j];
      else
        SpT[i][j] = SpT[i-1][j+(1<<(i-1))];
  }
}

int lca(int i, int j)
{
  int k = log2(j-i+1);
  if(deft[SpT[k][i]] < deft[SpT[k][j+1-(1<<k)]])
    return SpT[k][i];
  else
    return SpT[k][j+1-(1<<k)];
}

int main()
{
    int u, v, q;

  cin >> n >> m;
  for(int i = 0; i < m; i++)
  {
    cin >> u >> v;
    G[u].push_back(v);
    G[v].push_back(u);
  }

  tree_linearization(1, -1, 0);
  build(tl.size());

  cin >> q;
  while(q--)
  {
    cin >> u >> v;
    cout <<  lca(min(pos[u], pos[v]), max(pos[u], pos[v])) << '\n';
  }

  return 0;
}
```

7.21 Erdos Gallai Theorem

```
#include <bits/stdc++.h>
using namespace std;
#define int long long

int32_t main()
{
  int n;
  while(~scanf(" %lld", &n))
  {
      vector<int> degree(n), pref(n + 2);
        for(int &i : degree)
            scanf (" %lld", &i);
        sort(degree.begin(), degree.end(), greater<int>());
        for(int i = 0; i < n; i++)
    pref[i + 1] = pref[i] + degree[i];
        bool fl = true;
        if(pref[n] & 1)
            fl = false;
        int j = n;
        for(int k = 1; k <= n and fl; k++)
        {
            int L = pref[k];
            int R = k * (k - 1);
            while(j > 0 and degree[j - 1] < k)
                j--;
            int pos = max(j, k);
            R += pref[n] - pref[pos] + (pos - k) * k;
            if(L > R) fl = false;
        }
        puts(fl ? "possivel" : "impossivel");
  }
  return 0;
}
```

7.22 Kuhn MCBM

```
#include <bits/stdc++.h>
using namespace std;

int na, nb, m, tempo = 1;
int b[105];
int cor[105];
vector<int> G[105];

bool kuhn(int u)
{
  if(cor[u] == tempo)
    return 0;
  cor[u] = tempo;
//random_shuffle(G[u].begin(), G[u].end(), [](int x){ return rand() %
    x; });
  for(const int &v : G[u])
    if(!b[v] or kuhn(b[v]))
      return b[v] = u;
  return 0;
}
```

```
int main()
{
  //srand(time(NULL));
  cin >> na >> nb >> m;
  while(m--)
  {
    int u, v;
    cin >> u >> v;
    G[u].push_back(v + na);
  }
  tempo = 1;
  int ans = 0;
  for(int i = 1; i <= na; i++)
    ans += kuhn(i), tempo++;
  cout << "MCBM = " << ans << '\n';
  for(int i = nb + 1; i <= na + nb; i++)
    if(b[i])
      cout << b[i] << ' ' << i - na << '\n';

    return 0;
}
```

7.23 Lca With Tree Linearization And Segment Tree

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

int pos[MAX];
int deft[MAX];
int segtree[5*MAX];
vector<int> tl;
vector<int> G[MAX];

void tree_linearization(int v, int p, int d)
{
  deft[v] = d;
  pos[v] = tl.size();
  tl.push_back(v);
  for(const int &u : G[v])
    if(u != p)
    {
      tree_linearization(u, v, d + 1);
      tl.push_back(v);
    }
}

void build(int node, int start, int end)
{
  if(start == end)
    segtree[node] = tl[start];
  else
  {
    int mid = (start+end)/2;
    build(2*node, start, mid);
    build(2*node+1, mid+1, end);
    if(deft[segtree[2*node]] < deft[segtree[2*node+1]])
      segtree[node] = segtree[2*node];
    else
      segtree[node] = segtree[2*node+1];
  }
}

int lca(int node, int start, int end, int l, int r)
{
  if(l > end or r < start)
    return -1;
  if(l <= start and end <= r)
    return segtree[node];
  int mid = (start+end)/2;
  int p1 = lca(2*node, start, mid, l, r);
  int p2 = lca(2*node+1, mid+1, end, l, r);
  if(p1 == -1) return p2;
  if(p2 == -1) return p1;
  return deft[p1] < deft[p2] ? p1 : p2;
}

/*int _lca(int a, int b)
{
  int ancestor = a, nivel = 0x3f3f3f3f;
  for(int i = pos[a]; i <= pos[b]; i++)
    if(deft[tl[i]] < nivel)
    {
      ancestor = tl[i];
      nivel = deft[tl[i]];
    }
  return ancestor;
}*/


int main()
{
  int n, u, v;

  cin >> n;
  for(int i = 0; i < n-1; i++)
  {
    cin >> u >> v;
    G[u].push_back(v);
    G[v].push_back(u);
  }


  tree_linearization(1, -1, 0);
  build(1, 0, tl.size()-1);

  /*
  for(int i = 1; i <= n; i++)
    cout << deft[i] << ' ';
  cout << '\n';
  for(int i = 1; i <= n; i++)
    cout << pos[i] << ' ';
  cout << '\n';
  for(const int &p : tl)
    cout << p << ' ';
  cout << '\n';
  for(int i = 1; i <= 4*n; i++)
    cout << segtree[i] << ' ';
  cout << '\n';*/
```

```
  while(cin >> u >> v)
    cout << /*_lca(u, v) << ' ' <<*/
     lca(1, 0, tl.size()-1, pos[u], pos[v]) << '\n';

  return 0;
}
```

7.24 Min Cost Max Flow

```
/*
 *  from IME Library
 */

#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e3;
const int OO = 0x3f3f3f3f;

struct edge {int v, f, w, c; };

// flw_lmt eh a quantidade de de fluxo que posso passar
// no maximo, alterar se necessario
// node_count eh o valor do maior vertice no grafo...
// inicializar node_count com numero de vertices no inicio...
int node_count, flw_lmt = OO, p[MAX];
vector<edge> edges;
vector<int> G[MAX];

// u--->v, custo w e capacidade c
void add_edge(int u, int v, int w, int c)
{
  int k = edges.size();
  node_count = max(node_count, u+1);
  node_count = max(node_count, v+1);
  G[u].push_back(k);
  G[v].push_back(k+1);
  edges.push_back({ v, 0,  w, c });
  edges.push_back({ u, 0, -w, 0 });
}

void clear()
{
  flw_lmt = OO;
  for(int i = 0; i < node_count; ++i) G[i].clear();
  edges.clear();
  node_count = 0;
}

bool SPFA(int s, int t)
{
  vector<int> dist(node_count, OO);
  vector<int> et(node_count, 0);
  deque<int> q;
  q.push_back(s), dist[s] = 0;
  while (!q.empty())
  {
    int u = q.front(); q.pop_front();
    et[u] = 2;
    for(int i : G[u])
    {
      edge &e = edges[i];
      int v = e.v;
      if (e.f < e.c and dist[v] > dist[u] + e.w)
      {
        dist[v] = dist[u] + e.w;
        if (et[v] == 0) q.push_back(v);
        else if (et[v] == 2) q.push_front(v);
        et[v] = 1;
        p[v] = i;
      }
          }
  }
  return dist[t] != OO;
}

int min_cost_max_flow(int s, int t)
{
  int mf = 0, cost = 0;
    while(SPFA(s, t) and mf < flw_lmt)
    {
    int inc = flw_lmt - mf;
    for (int u = t; u != s; u = edges[p[u]^1].v)
    {
      edge &e = edges[p[u]];
      inc = min(inc, e.c - e.f);
    }
    for (int u = t; u != s; u = edges[p[u]^1].v)
    {
      edge &e = edges[p[u]], &rev = edges[p[u]^1];
      e.f += inc;
      rev.f -= inc;
      cost += inc * e.w;
    }
        if (!inc) break;
        mf += inc;
  }
  cout << "Max Flow " << mf << '\n';
  cout << "Min Cost " << cost << '\n';
  return cost;
}

int main()
{


  return 0;
}
```

7.25 Fully Dynamic Connectivity Count Conected Components

```
#include <bits/stdc++.h>
using namespace std;

const int MAX = 131072;
const int OO = 0x3f3f3f3f;
const double EPS = 1e-9;

#define bug(x) cout << #x << " = " << x << '\n'
```

```cpp
#define FOR(i, a, n) for(int i = a; i < n; i++)
#define REP(i, n) FOR(i, 0, n)
#define fi first
#define se second
#define pb push_back
#define mt make_tuple
#define mp make_pair
#define all(vetor) vetor.begin(), vetor.end()
#define X real()
#define Y imag()
//#define gc getchar_unlocked

typedef long long ll;
typedef long double ld;
//typedef pair<int, int> ii;
//typedef pair<int, ii> iii;
typedef complex<ll> Pll;
typedef complex<ld> Pld;


typedef pair<int, int> edge;

vector<edge> tree[4 * MAX];
vector<int> query[4 * MAX];
int pai[MAX], sz[MAX];
stack<pair<int, int>> stk, size;
stack<int> qtd;
vector<int> ans;
int rep;

int find(int x)
{
    if(pai[x] == x)
        return x;
    stk.push(mp(x, pai[x]));
    size.push(mp(x, sz[x]));
    qtd.push(rep);
    return pai[x] = find(pai[x]);
}

void join(int x, int y)
{
    x = find(x);
    y = find(y);
    if(x == y) return;
    if(sz[x] > sz[y])
        swap(x, y);
    qtd.push(rep);
    stk.push(mp(x, pai[x]));
    size.push(mp(y, sz[y]));
    sz[y] += sz[x];
    pai[x] = y;
    rep--;
}

void rollback(int rollback_to)//desfaz todas as alteracoes no DSU,
{//O(k) onde k eh a quantidade de operacoes realizadas
  while(rollback_to < stk.size())
  {
    pai[stk.top().fi] = stk.top().se;
    stk.pop();
    sz[size.top().fi] = size.top().se;
    size.pop();
    rep = qtd.top();
    qtd.pop();
  }
}

void add_edge(int node, int start, int end, int l, int r, edge e)
{
    if(start == l and end == r)
    {
        tree[node].push_back(e);
        return;
    }
    if(l >= r)
        return;
    int mid = (start + end) / 2;
    add_edge(2*node, start, mid, l, min(mid, r), e);
    add_edge(2*node + 1, mid + 1, end, max(l, mid + 1), r, e);
}

void add_query(int node, int start, int end, int idx, int e)
{
    if(start == end)
        query[node].push_back(e);
    else
    {
        int mid = (start + end) / 2;
        if(idx <= mid)
            add_query(2*node, start, mid, idx, e);
        else
            add_query(2*node + 1, mid + 1, end, idx, e);
    }
}

void processar(int node)
{
    for(auto it : tree[node])
        join(it.first, it.second);
}

void dfs(int node, int start, int end)
{
    int rollback_to = stk.size();
    processar(node);
    if(start == end)
    {
        for(auto v : query[node])
            ans.push_back(rep);
    }
    else
    {
        int mid = (start + end) / 2;
        dfs(2*node, start, mid);
        dfs(2*node + 1, mid + 1, end);
    }
    rollback(rollback_to);
}

int main()
{
```

```
    int n, q, u, v;
    cin >> n >> q;
    rep = n;
    for(int i = 0; i <= n; i++)
        sz[i] = 1, pai[i] = i;
    int cur = 0;
    map<pair<int, int>, int> mapa;
    while(q--)
    {
        char o;
        cin >> o;
        if(o != '?')
        {
            cin >> u >> v; u--; v--;
      if(u > v) swap(u, v);
            if(o == '+')// adicionar aresta
                mapa[mp(u, v)] = cur++;
            else if(o == '-')// remover aresta
            {
                add_edge(1, 0, MAX-1, mapa[mp(u, v)], cur++, mp(u, v))
                    ;
                mapa.erase(mp(u, v));
            }
        }
        else // verificar se dois vertices estao na mesma componente
            add_query(1, 0, MAX-1, cur++, 1);
    }
    cur++;
    for(auto it : mapa)
        add_edge(1, 0, MAX-1, it.second, cur, it.first);
    dfs(1, 0, MAX-1);
    for(int i = 0; i < ans.size(); i++)
        cout << ans[i] << '\n';

    return 0;
}
```

7.26 Dinic

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e4;
const int OO = 0x3f3f3f3f;

struct edge
{
    int v, f, c;
    edge(){}
    edge(int _v, int _f, int _c)
    {
        v = _v, f = _f, c = _c;
    }
};

vector<edge> edges;
vector<int> G[MAX];
int dist[MAX], work[MAX];

void add_edge(int u, int v, int cp, int rc){
  edges.push_back(edge(v, 0, cp));
  G[u].push_back(edges.size()-1);
  edges.push_back(edge(u, 0, rc));
  G[v].push_back(edges.size()-1);
}

bool bfs(int s, int t)
{
    memset(dist, -1, sizeof(dist));
    dist[s] = 0;
    queue<int> q;
    q.push(s);
    while(!q.empty())
    {
        int u = q.front();
        q.pop();
        for(int e : G[u])
            if(dist[edges[e].v] == -1 and edges[e].c-edges[e].f > 0)
            {
                q.push(edges[e].v);
                dist[edges[e].v] = dist[u] + 1;
            }
    }
    return dist[t] != -1;
}

int dfs(int s, int t, int f)
{
    if(s == t) return f;
    for(int &i = work[s]; i < G[s].size(); i++)
    {
      int e = G[s][i];
        if(dist[edges[e].v] == dist[s] + 1 and edges[e].c-edges[e].f >
            0)
            if(int a = dfs(edges[e].v, t, min(f, edges[e].c-edges[e].f
                )))
            {
                edges[e].f += a;
                edges[e^1].f -= a;
                return a;
            }
    }
    return 0;
}

int MaxFlow(int s, int t)
{
    int mf = 0;
    while(bfs(s, t))
  {
      memset(work, 0, sizeof(work));
        while(int a = dfs(s, t, OO))
            mf += a;
    }
    return mf;
}

int main()
{
    int n, m, u, v, w;

    cin >> n >> m;
```

```
    while(m--)
    {
        cin >> u >> v >> w;
        add_edge(u-1, v-1, w, 0);
    }
    cin >> u >> v;
    cout << MaxFlow(u-1, v-1) << '\n';

    return 0;
}
```

7.27 Prim

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;
const int OO = 0x3f3f3f3f;
typedef pair<int, int> ii;
typedef pair<int, ii> iii;

int n, m;
vector<ii> G[MAX];
int dist[MAX], edge[MAX];
bool visit[MAX];

void prim(int s)
{
    memset(visit, 0, sizeof(visit));
    memset(dist, 63, sizeof(dist));
    dist[s] = 0;
    priority_queue<ii> pq;
    pq.push({0, s});
    while(!pq.empty())
    {
        int u = pq.top().second;
        pq.pop();
        if(visit[u]) continue;
        for(int i = 0; i < G[u].size(); i++)
        {
            int v = G[u][i].second, d = G[u][i].first;
            if(!visit[v] and dist[v] > d)
            {
                dist[v] = d;
                edge[v] = u;
                pq.push({-d, v});
            }
        }
        visit[u] = true;
    }
    int ans = 0;
    edge[s] = -2;
    for(int i = 0; i < n; i++)
    {
        cout << edge[i]+1 << ' ';
        ans += dist[i];
    }
    cout << '\n';
    cout << ans << '\n';
}

int main()
{
    int u, v, w;

    cin >> n >> m;
    while(m--)
    {
        cin >> u >> v >> w; u--; v--;
        G[u].push_back({w, v});
        G[v].push_back({w, u});
    }
    prim(0);

    return 0;
}
```

7.28 Ford Fulkerson

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> ii;
const int OO = 0x3f3f3f3f;
const int MAX = 1e4;

struct edge
{
    int v, f, c;
    edge(){}
    edge(int _v, int _f, int _c)
    {
        v = _v, f = _f, c = _c;
    }
};

vector<edge> edges;
vector<int> G[MAX];
int tempo = 1, cor[MAX];

void add_edge(int u, int v, int cp, int rc)
{
    edges.push_back(edge(v, 0, cp));
    G[u].push_back(edges.size()-1);
    edges.push_back(edge(u, 0, rc));
    G[v].push_back(edges.size()-1);
}

int dfs(int s, int t, int f)
{
    if(s == t) return f;
    cor[s] = tempo;
    for(int e : G[s])
        if(cor[edges[e].v] < tempo and edges[e].c-edges[e].f > 0)
            if(int a = dfs(edges[e].v, t, min(f, edges[e].c-edges[e].f
                )))
            {
                edges[e].f += a;
                edges[e^1].f -= a;
                return a;
            }
    return 0;
```

```
}

int MaxFlow(int s, int t)
{
    int mf = 0;
    while(int a = dfs(s, t, OO))
        mf += a, tempo++;
    return mf;
}

int main()
{
    int n, m, w, u, v;

    cin >> n >> m;
    while(m--)
    {
        cin >> u >> v >> w;
        add_edge(u-1, v-1, w, 0);
    }
    cin >> u >> v;
    cout << MaxFlow(u-1, v-1) << '\n';

    return 0;
}
```

7.29 Center Of A Tree

```
#include <bits/stdc++.h>
const int MAX = 1e5;
using namespace std;

int n, degree[MAX];
vector<int> G[MAX];
bool vis[MAX];

int findCenter()
{
  queue<int> fila[2];
  for(int i = 0; i < n; i++)
    if(degree[i] == 1)
      fila[0].push(i);
  int cnt = 0, turn = 0;
  while(cnt + 2 < n)
  {
    while(!fila[turn].empty())
    {
      int u = fila[turn].front(); fila[turn].pop();
      vis[u] = true;
      cnt++;
      for(int i = 0; i < G[u].size(); i++)
        if(!vis[G[u][i]])
        {
          degree[G[u][i]]--;
          if(degree[G[u][i]] == 1)
            fila[1-turn].push(G[u][i]);
        }
    }
    turn ^= 1;
  }
  cout << "the set of central vertices\n";
  for(int i = 0; i < n; i++)
    if(!vis[i])
        cout << i + 1 << '\n';
}

int main()
{
    cin >> n;
  for(int i = 1; i < n; i++)
  {
    int u, v;
    scanf("%d %d", &u, &v); u--; v--;
    G[u].push_back(v);
    G[v].push_back(u);
    degree[u]++;
    degree[v]++;
  }
  findCenter();

  return 0;
}
```

8 Data Structures

8.1 Two Stacks Trick

```
const int OO = 0x3f3f3f3f;

struct Stack {
    vector<int> s, smax = {-OO}, smin = {OO};
    void push(int x) {
        s.push_back(x);
        smax.push_back(max(smax.back(), x));
        smin.push_back(min(smin.back(), x));
    }
    int pop() {
        int x = s.back();
        s.pop_back();
        smax.pop_back();
        smin.pop_back();
        return x;
    }
    int min_() {
        return smin.back();
    }
    int max_() {
        return smax.back();
    }
    bool empty() {
        return s.empty();
    }
};

Stack s1, s2;

void push(int x) {
    s2.push(x);
```

```
}

void pop() {
    if(s1.empty()) {
        while(!s2.empty())
            s1.push(s2.pop());
    }
    s1.pop();
}

int min_() {
    return min(s1.min_(), s2.min_());
}

int max_() {
    return max(s1.max_(), s2.max_());
}
```

8.2 Segment Tree Tree 2D

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e3;

int n, leaf;
int arr[MAX][MAX];
int ST[4*MAX][4*MAX];

void buildLeaf(int k, int node, int l, int r)
{
  if(l == r)
    ST[k][node] = arr[leaf][l];
  else
  {
    int mid = (l + r) / 2;
    buildLeaf(k, 2*node, l, mid);
    buildLeaf(k, 2*node + 1, mid + 1, r);
    ST[k][node] = ST[k][2*node] + ST[k][2*node+1];
  }
}

void build(int node, int l, int r)
{
  if(l == r)
    buildLeaf(node, 1, 0, n-1), leaf++;
  else
  {
    int mid = (l + r) / 2;
    build(2*node, l, mid);
    build(2*node + 1, mid + 1, r);
    for(int i = 1; i < 4*n; i++)
      ST[node][i] = ST[2*node][i] + ST[2*node+1][i];
  }
}

int queryNode(int k, int node, int l, int r, int cx, int cy)
{
  if(l > cy or r < cx)
    return 0;
  if(cx <= l and r <= cy)
    return ST[k][node];
  int mid = (l + r) / 2;
  int ans = queryNode(k, 2*node, l, mid, cx, cy);
  ans += queryNode(k, 2*node + 1, mid + 1, r, cx, cy);
  return ans;
}

int query(int node, int l, int r, int lx, int ly, int cx, int cy)
{
  if(l > ly or r < lx)
    return 0;
  if(lx <= l and r <= ly)
    return queryNode(node, 1, 0, n-1, cx, cy);
  int mid = (l + r) / 2;
  int ans = query(2*node, l, mid, lx, ly, cx, cy);
  ans += query(2*node + 1, mid + 1, r, lx, ly, cx, cy);
  return ans;
}

void updateNode(int k, int node, int l, int r, int x, int y, int value
    )
{
  if(l == r)
    ST[k][node] = arr[x][y] = value;
  else
  {
    int mid = (l + r) / 2;
    if(l <= y and y <= mid)
      updateNode(k, 2*node, l, mid, x, y, value);
    else
      updateNode(k, 2*node + 1, mid + 1, r, x, y, value);
    ST[k][node] = ST[k][2*node] + ST[k][2*node + 1];
  }
}

void update(int node, int l, int r, int x, int y, int value)
{
  if(l == r)
    updateNode(node, 1, 0, n-1, x, y, value);
  else
  {
    int mid = (l + r) / 2;
    if(l <= x and x <= mid)
      update(2*node, l, mid, x, y, value);
    else
      update(2*node + 1, mid + 1, r, x, y, value);
    for(int i = 1; i < 4*n; i++)
      ST[node][i] = ST[2*node][i] + ST[2*node+1][i];
  }
}

int main()
{
  cin >> n;
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      cin >> arr[i][j];
  build(1, 0, n-1);
  int o, a, b, c, d;
  while(cin >> o)
  {
```

```
    cin >> a >> b >> c;
    if(o == 1)
      update(1, 0, n-1, a-1, b-1, c);
    else
    {
      cin >> d;
      cout << query(1, 0, n-1, a-1, b-1, c-1, d-1) << '\n';
    }
  }

  return 0;
}
```

8.3 Merge Sort Tree With Set

```
/*
encontra o menor numero na range [L, R] que eh maior
ou igual a K. build eh O(NlogNlogN) e query eh O(logNlogN)
e o erasee eh O(logNlogN).
*/


#define ii pair<int, int>
#define value first
#define index second
const int MAX = 1e6;

int n, m;
vector<array<int, 3>> B, A;

set<ii> tree[MAX];

void build(int node, int start, int end)
{
    if(start == end)
        tree[node] = set<ii>{ii(A[start][1], start)};
    else
    {
        int mid = (start + end) / 2;
        build(2*node, start, mid);
        build(2*node + 1, mid + 1, end);
      for(auto &it : tree[2 * node])
        tree[node].insert(it);
      for(auto &it : tree[2 * node + 1])
        tree[node].insert(it);
    }
}

ii query(int node, int start, int end, int l, int r, int k)
{
    if(start > r or end < l)
        return ii(-1, -1);
    if(l <= start and end <= r)
    {
      auto it = tree[node].upper_bound({k, n + 1});
      ii q = {-1, -1};
        if(it != tree[node].begin()) q = *--it;
        return q;
    }
    int mid = (start + end) / 2;
    ii p1 = query(2 * node, start, mid, l, r, k);
    ii p2 = query(2 * node + 1, mid + 1, end, l, r, k);
  return p1.value <= p2.value ? p2 : p1;
}

void erasee(int node, int start, int end, ii p)
{
  if(tree[node].count(p) == 0) return;
  if(start == end)
  {
    tree[node].erase(p);
    return;
  }
  int mid = (start + end) / 2;
  erasee(2 * node, start, mid, p);
  erasee(2 * node + 1, mid + 1, end, p);
  tree[node].erase(p);
}
```

8.4 Segment Tree With Lazy Propagation

```
#include <bits/stdc++.h>
using namespace std;

#define int long long

#define esq node << 1LL
#define dir (node << 1) | 1LL

struct SegmentTree {
  vector<int> tree, lazy;
  int size;

  void build(int node, int start, int end, vector<int> &a) {
    if(start == end) {
      tree[node] = a[start];
      return;
    }
    int mid = (start + end) >> 1;
    build(esq, start, mid, a);
    build(dir, mid + 1, end, a);
    tree[node] = tree[esq] + tree[dir];
  }

  SegmentTree() {}

  SegmentTree(int n) {
    size = n;
    tree.resize(size << 2);
    lazy.resize(size << 2);
  }

  void init(vector<int> &a) {
    build(1, 0, size - 1, a);
  }

  // += add in the interval
  void push(int node, int start, int end) {
    tree[node] += lazy[node] * (end - start + 1);
    if(start != end) {
```

```
      lazy[esq] += lazy[node];
      lazy[dir] += lazy[node];
    }
    lazy[node] = 0;
  }

  int query(int node, int start, int end, int l, int r) {
    if(lazy[node]) push(node, start, end);
    if(l > end or start > r or l > r) return 0;
    if(l <= start and end <= r) return tree[node];
    int mid = (start + end) >> 1;
    int q1 = query(esq, start, mid, l, r);
    int q2 = query(dir, mid + 1, end, l, r);
    return q1 + q2;
  }

  int query(int l, int r) {
    return query(1, 0, size - 1, l, r);
  }

  void update(int node, int start, int end, int l, int r, int v) {
    if(lazy[node]) push(node, start, end);
    if(l > end or start > r or l > r) return;
    if(l <= start and end <= r)
    {
      lazy[node] += v;
      push(node, start, end);
      return;
    }
    int mid = (start + end) >> 1;
    update(esq, start, mid, l, r, v);
    update(dir, mid + 1, end, l, r, v);
    tree[node] = tree[esq] + tree[dir];
  }

  void update(int l, int r, int v) {
    update(1, 0, size - 1, l, r, v);
  }
};

int32_t main() {

  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  int n;
  cin >> n;
  SegmentTree T(n);

  return 0;
}
```

8.5 Color Update

```
#include <bits/stdc++.h>
using namespace std;

#define bug(x) cout << #x << "  >>>>>>>  " << x << '\n'
#define _ << " , " <<
#define INF 0x3f3f3f3f

#define ii pair<int, int>
#define fi first
#define se second

struct Color {
  int x;
  Color(int _x) : x(_x) {}
  Color() : x(-1) {}
  bool operator<(const Color &c) const {
    return x < c.x;
  }
};

#define iic pair<pair<int, int>, Color>
#define defaut_color Color(-1)

struct ColorUpdate {
  set<iic> intervals;

  ColorUpdate(int begin = -INF, int end = INF) {
    intervals.insert( { {begin, end}, defaut_color } );
  }

  void paint(int l, int r, Color c = defaut_color) {
    if(l > r) return;

    auto a = prev(intervals.upper_bound({{l, INF}, INF}));
    auto b = prev(intervals.upper_bound({{r, INF}, INF}));

    int l1 = a->fi.fi, r1 = l - 1;
    Color c1 = a->se;
    int l2 = r + 1, r2 = b->fi.se;
    Color c2 = b->se;

    intervals.erase(a, next(b));

    if(l1 <= r1) intervals.insert({{l1, r1}, c1});
    if(l2 <= r2) intervals.insert({{l2, r2}, c2});
    if(l <= r) intervals.insert({{l, r}, c});

    // printall();
  }

  Color get_color_of(int x) {
    return prev(intervals.upper_bound({{x, INF}, INF}))->se;
  }

  // true if x is in some interval
  bool find(int x) {
    auto a = intervals.upper_bound({{x, INF}, INF});
    if(a == intervals.begin()) return false;
    a--;
    if(a->fi.fi > x or a->fi.se < x) return false;
    return true;
  }

  ii get_interval_of(int x) {
    if(!find(x)) return {-INF, INF};
    return prev(intervals.upper_bound({{x, INF}, INF}))->fi;
  }
```

```
  ii get_interval_of(int l, int r) {
    if(!find(l)) return {-INF, INF};
    ii i = get_interval_of(l);
    if(i.fi <= r and r <= i.se) return i;
    return {-INF, INF};
  }

  // x will be on the left side
  void cut_at(int x) {
    if(!find(x)) return;

    auto a = prev(intervals.upper_bound({{x, INF}, INF}));

    Color c = a->se;

    int l1 = a->fi.fi, r1 = x;
    int l2 = x + 1, r2 = a->fi.se;

    intervals.erase(a);

    if(l1 <= r1) intervals.insert({{l1, r1}, c});
    if(l2 <= r2) intervals.insert({{l2, r2}, c});
  }

  void remove_interval(int l, int r) {
    cut_at(l - 1);
    cut_at(r);

    auto a = prev(intervals.upper_bound({{l, INF}, INF}));
    auto b = prev(intervals.upper_bound({{r, INF}, INF}));

    intervals.erase(a, next(b));
  }

  void remove_at(int x) {
    remove_interval(x, x);
  }

  void p_interval(iic i) {
    cout << "elements from " << i.fi.fi << " to " << i.fi.se;
    cout << " have color " << i.se.x << endl;
  }

  void printall() {
    cout << "\n\n\nColor Of The Elements:\n";
    for(auto it : intervals)
      p_interval(it);
    cout << endl << endl;
  }
};

int32_t main() {

  ios_base::sync_with_stdio(false);
  cin.tie(nullptr);

  ColorUpdate C(1, 100);

  for(int i = 3; i <= 50; i += 10)
    C.cut_at(i);

  C.remove_interval(16, 57);

  C.remove_at(100);
  C.remove_at(1);
  C.remove_at(2);
  C.remove_at(3);

  C.cut_at(20);

  C.printall();

  return 0;
}
```

8.6 Segment Tree Iterative

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5 + 10;

int tree[MAX << 1], n;

void build()
{
  for(int i = n - 1; i > 0; --i) tree[i] = tree[i << 1] + tree[i << 1
      | 1];
}

void update(int p, int value)
{
  for(tree[p += n] = value; p > 1; p >>= 1) tree[p >> 1] = tree[p] +
      tree[p ^ 1];
}

int query(int l, int r)
{
  int res = 0;
  for(l += n, r += n; l < r; l >>= 1, r >>= 1)
  {
    if(l & 1) res += tree[l++];
    if(r & 1) res += tree[--r];
  }
  return res;
}

int main()
{
  cin >> n;
  for(int i = 0; i < n; i++)
    cin >> tree[i + n];
  build();
  int q, l, r, o;
  cin >> q;
  while(q--)
  {
    cin >> o >> l >> r;
    if(o == 1)
      cout << query(l - 1, r) << '\n'; // soma de [l, r)
    else
```

```
        update(l - 1, r); // atualiza a posicoo l pra r
    }

    return 0;
}
```

8.7 DSU With Partial Persistence

```
int n, pai[MAX], sz[MAX], his[MAX], tempo;

void init()
{
  tempo = 0;
  for(int i = 0; i < n; i++)
    pai[i] = i, sz[i] = 1, his[i] = 0;
}

int find(int x, int t)
{
  if(pai[x] == x) return x;
  if(his[x] > t) return x;
  return find(pai[x], t);
}

void join(int u, int v)
{
  tempo++;
  u = find(u, tempo);
  v = find(v, tempo);
  if(sz[u] > sz[v]) swap(u, v);
  pai[u] = v;
  his[u] = tempo;
  sz[v] += sz[u];
}
```

8.8 BIT 1D

```
#include <bits/stdc++.h>
using namespace std;

int aux, n, arr[1000], BIT[1000];

// construir uma BIT a partir de um array em O(N)
void build()  {
  for(int i = 1; i <= n; i++) {
    BIT[i] += arr[i];
    if(i + (i & -i) <= n)
      BIT[i + (i & -i)] += BIT[i];
  }
}

// construir o array que gera a BIT a partir de uma BIT em O(N)
void buildArray()  {
  for(int i = n; i >= 1; i--)
    if(i + (i & -i) <= n)
      BIT[i + (i & -i)] -= BIT[i];
}

int sum(int x) {
    int s = 0;
    while(x) s += BIT[x], x -= x&-x;
    return s;
}

void update(int x, int value) {
    while(x <= n) BIT[x] += value, x += x&-x;
}

int main() {
    cin >> n;
    for(int i = 1; i <= n; i++) {
        cin >> aux;
        update(i, aux);
    }
    int a, b;
    cin >> a >> b;
    cout << sum(b)-sum(a-1) << '\n';

    return 0;
}
```

8.9 Dynamic Segment Tree With Lazy Propagation

```
#include <bits/stdc++.h>
using namespace std;

struct Node
{
  int l, r, value;
};

vector<Node> tree;
vector<int> lazy;

int init()
{
  tree.clear();
  lazy.clear();
  tree.emplace_back();
  lazy.push_back(0);
}

void createL(int node)
{
  tree[node].l = tree.size();
  tree.emplace_back();
  lazy.push_back(0);
}

void createR(int node)
{
  tree[node].r = tree.size();
  tree.emplace_back();
  lazy.push_back(0);
}

void calc(int node)
{
```

```
  tree[node].value = 0;
  if(tree[node].l) tree[node].value += tree[tree[node].l].value;
  if(tree[node].r) tree[node].value += tree[tree[node].r].value;
}

void push(int node, int start, int end)
{
  // +=
  tree[node].value = lazy[node] * (end - start + 1);
  if(start != end)
  {
    if(tree[node].l == 0) createL(node);
    if(tree[node].r == 0) createR(node);
    lazy[tree[node].l] = lazy[node]; // +=
    lazy[tree[node].r] = lazy[node]; // +=
  }
  lazy[node] = 0;
}

void update(int node, int start, int end, int l, int r, int value)
{
  if(lazy[node])
    push(node, start, end);

  if(start > r or l > end) return;

  if(l <= start and end <= r)
  {
    tree[node].value = value * (end - start + 1); // +=
    if(start != end)
    {
      if(tree[node].l == 0) createL(node);
      if(tree[node].r == 0) createR(node);
      lazy[tree[node].l] = value; // +=
      lazy[tree[node].r] = value; // +=
    }
  }
  else
  {
    int mid = (start + end) / 2;
    if(tree[node].l == 0) createL(node);
    update(tree[node].l, start, mid, l, r, value);
    if(tree[node].r == 0) createR(node);
    update(tree[node].r, mid + 1, end, l, r, value);
    calc(node);
  }
}

int query(int node, int start, int end, int l, int r)
{
  if(lazy[node])
    push(node, start, end);

  if(start > r or l > end) return 0;

  if(l <= start and end <= r) return tree[node].value;

  int mid = (start + end) / 2, q1 = 0, q2 = 0;
  if(tree[node].l) q1 = query(tree[node].l, start, mid, l, r);
  if(tree[node].r) q2 = query(tree[node].r, mid + 1, end, l, r);
  return q1 + q2;
}

int main()
{
  int n, q;

  cin >> n >> q;
  init();
  for(int i = 0; i < n; i++)
  {
    int x;
    cin >> x;
    update(0, 0, n - 1, i, i, x);
  }
  while(q--)
  {
    int o, l, r, x;
    cin >> o >> l >> r;
    if(o == 1)
    {
      cin >> x;
      update(0, 0, n - 1, l - 1, r - 1, x);
    }
    else
      cout << query(0, 0, n - 1, l - 1, r - 1) << '\n';
  }

  return 0;
}
```

8.10 Merge Sort Tree

```
/*
O nome eh Merge Sort Tree pois o array armazenado em
 um Node da Segment Tree eh igual ao gerado pelo algoritmo
 de ordenacao Merge Sort no Node correspondente da
 arvore de recursao.
*/

#include <bits/stdc++.h>
using namespace std;

const int MAX = 131072;

vector<int> a[MAX], tree[MAX];

void build(int node, int start, int end)
{
    if(start == end)
        tree[node] = a[start];
    else
    {
        int mid = (start + end) / 2;
        build(2*node, start, mid);
        build(2*node + 1, mid + 1, end);
        merge(tree[2*node].begin(), tree[2*node].end(),
            tree[2*node + 1].begin(), tree[2*node + 1].end()
            , back_inserter(tree[node]));
    }
}
```

```
int query(int node, int start, int end, int l, int r, int k)
{
    if(start > r or end < l)
        return 0;
    if(l <= start and end <= r)
        return upper_bound(tree[node].begin(), tree[node].end(), k)
               - tree[node].begin();
    int mid = (start + end) / 2;
    int p1 = query(2*node, start, mid, l, r, k);
    int p2 = query(2*node + 1, mid + 1, end, l, r, k);
    return p1 + p2;
}

int main()
{
    int n, aux;

    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> aux;
        a[i].push_back(aux);
    }
    build(1, 0, n-1);
    int l, r, k;
    cin >> l >> r >> k;
//quantidade de elementos menores ou iguais a k na range [l - r].
    cout << query(1, 0, n-1, l-1, r-1, k) << '\n';

    return 0;
}
```

8.11 Treap

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

using namespace std;

struct Node
{
  int valor, priority, size, maior;
  Node *l, *r;

  Node(int _valor) : valor(_valor), priority((rand() << 16)
   ^ rand()), size(1), l(nullptr), r(nullptr), maior(_valor) {}
  ~Node() { delete l; delete r; }

  void recalc()
  {
    size = 1;
    maior = valor;
    if (l) size += l->size, maior = max(maior, l->maior);
    if (r) size += r->size, maior = max(maior, r->maior);
  }
};

struct Treap
{
  Node* merge(Node *l, Node *r)
  {
    if(!l or !r) return l ? l : r;
    // Se a prioridade esquerda eh menor.
    if(l->priority < r->priority)
    {
      l->r = merge(l->r, r);
      l->recalc();
      return l;
      // Se a prioridade direita eh maior ou igual.
    }
    else
    {
      r->l = merge(l, r->l);
      r->recalc();
      return r;
    }
  }

// Valores maiores ou iguais a "valor" ficarao no r, e os demais no l.
  void split(Node *v, int valor, Node *&l, Node *&r)
  {
    l = r = nullptr;
    if(!v) return;
    // Se o valor for maior, ir para direita
    if(v->valor < valor)
    {
      split(v->r, valor, v->r, r);
      l = v;
      // Se o valor for menor ou igual ir para esquerda.
    }else
    {
      split(v->l, valor, l, v->l);
      r = v;
    }
    v->recalc();
  }

  bool find(Node *v, int valor)
  {
    if(!v) return false;
    if( v->valor == valor ) return true;
    if( v->valor < valor ) return find(v->r, valor);
    if( v->valor > valor ) return find(v->l, valor);
  }

  int smallestCount(Node *v, int valor)
  {
    if(!v) return 0;
    // Se for menor ou igual adicionar + 1.
    if(v->valor == valor) return (v->l ? v->l->size : 0);
    if(v->valor < valor) return 1 + (v->l ? v->l->size : 0)
      + smallestCount(v->r, valor);
    if(v->valor > valor) return smallestCount(v->l, valor);
  }

  Node* kth(Node *v, int posicao)
  {
    if(!v) return nullptr;
    int esquerda = (v->l? v->l->size: 0);
    if(posicao-esquerda == 1) return v;
```

```
  if(posicao-esquerda > 1) return kth(v->r, posicao-esquerda-1);
  if(posicao-esquerda < 1) return kth(v->l, posicao);
}

// Sendo i e j os indices no array ordenado
// Talvez deh problemas de i e j estiverem fora do range.
int query(int i, int j)
{
  Node *l, *q, *r;
  split(root, kth(root, i+1)->valor, l, q);
  split(q, kth(q, j+1-i)->valor+1, q, r);
  int x = q->maior;
  root = merge(l, merge(q,r));
  return x;
}

Node * root;
Treap() : root(nullptr) {}
~Treap() { delete root; }

// Se existe um elemento com o valor
bool find(int valor)
{
  return find(root, valor);
}
// Quantidade de elementos menores que o valor
int smallestCount(int valor)
{
  return smallestCount(root, valor);
}
// Retorna o k-th menor elemento
Node * kth(int posicao){
  return kth(root, posicao);
}
// Insere o valor mesmo se ja exista outro com valor igual
void insert(int valor)
{
  Node * l, * r;
  split(root, valor, l, r);
  root = merge(merge(l, new Node(valor)), r);
}
// Apaga todos os elementos que possuem o valor.
void erase(int valor)
{
  Node * l, * m, * r;
  split(root, valor, l, m);
  split(m, valor + 1, m, r);
  delete m;
  root = merge(l, r);
}

// Quantos valores existem menor que "valor"
int menoresQue(int valor)
{
  Node * l, * r;
  split(root, valor, l, r);
  int res = (l? l->size: 0);
  root = merge(l,r);
  return res;
}

  // splitSmallest eh uma funcao que esta na implicit treap


  // Retorna a consulta dos primeiros "quantidade" valor
  int top(int quantidade)
  {
    Node *l, *r;
    splitSmallest(root, quantidade, l, r);
    int valor = (l ? l->maior : 0);
    root = merge(l,r);
    return valor;
  }
  // Remover os d menores
  void removeSmallest(int d)
  {
    Node *l, *r;
    splitSmallest(root, d, l, r);
    root = r;
    if(l) delete l;
  }
  // Remover todos menos os d menores
  void limit(int d)
  {
    Node * l, * r;
    splitSmallest(root, d, l, r);
    root = l;
    if(r) delete r;
  }
  int size() const { return root ? root->size : 0; }
}treap;

int n, a;
char op;

int main()
{
    srand(time(0));

  cin >> n;
  for(int i = 0; i < n; i++)
  {
    cin >> op >> a;
    if(op == 'I')
    {
      if(!treap.find(a))
        treap.insert(a);
    }
    else if(op == 'D' )
      treap.erase(a);
    else if(op == 'C' )
      cout << treap.smallestCount(a) << '\n';
    else if(op == 'K' )
    {
      Node *v = treap.kth(a);
      if(v == nullptr) cout << "invalid" << '\n';
      else cout << v->valor << '\n';
    }
  }

  return 0;
}
```

8.12 Max Queue

```
#include <bits/stdc++.h>
using namespace std;

struct MaxQueue
{
  int plus = 0;
  deque<pair<int, int>> dq;

  bool empty()
  {
    return (int)dq.size() == 0;
  }

  void clear()
  {
    plus = 0;
    dq.clear();
  }

  void add(int x)
  { // somar x em cada elemento da fila
    plus += x;
  }
  int max()
  {
    return dq.begin()->first + plus;
  }

  void push(int x)
  {
    x -= plus;
    int amt = 0;
    while (dq.size() and dq.back().first <= x)
      amt += dq.back().second + 1, dq.pop_back();
    dq.push_back({ x, amt });
  }

  void pop()
  {
    if (dq.empty()) return;
    if (!dq.front().second) dq.pop_front();
    else dq.front().second--;
  }
};

int main()
{
  int n, aux;
  MaxQueue Q;
  cin >> n;
  for(int i = 0; i < n; i++)
  {
    int aux;
    cin >> aux;
    Q.push(aux);
    cout << "max " << Q.max() << '\n';
  }
  return 0;
}
```

8.13 BIT 2D

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e3;

int n, aux, BIT[MAX][MAX];

void update(int x, int y, int value)
{
  for(int i = x; i <= n; i += i&-i)
    for(int j = y; j <= n; j += j&-j)
      BIT[i][j] += value;
}

int query(int x, int y)
{
  int sum = 0;
  for(int  i = x; i > 0; i -= i&-i)
    for(int j = y; j > 0; j -= j&-j)
      sum += BIT[i][j];
  return sum;
}

int queryInRectangle(int x1, int y1, int x2, int y2)
{
  int sum = 0;
  sum += query(max(x1, x2), max(y1, y2));
  sum -= query(max(x1, x2), min(y1, y2) - 1);
  sum -= query(min(x1, x2) - 1, max(y1, y2));
  sum += query(min(x1, x2) - 1, min(y1, y2) - 1);
  return sum;
}

int main()
{
  cin >> n;
  for(int i = 1; i <= n; i++)
    for(int j = 1; j <= n; j++)
      cin >> aux, update(i, j, aux);
  int x1, y1, x2, y2;
  while(cin >> x1 >> y1 >> x2 >> y2)
    cout << queryInRectangle(x1, y1, x2, y2) << '\n';

  return 0;
}
```

8.14 Ordered Set With BIT

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5+10;

int bit[MAX], arr[MAX];

int bitSearch(int v)
```

```
{
  int sum = 0, pos = 0, LOGN = log2(MAX - 2);
  for(int i = LOGN; i >= 0; i--)
    if(pos + (1 << i) < MAX and sum + bit[pos + (1 << i)] < v)
    {
      sum += bit[pos + (1 << i)];
      pos += (1 << i);
    }
  return pos + 1;
  // pos + 1, pq pos eh a maior posicao cuja soma do prefixo ate
  // ela eh menor que V
}
// essa funcao retorna o indice J no array em que a soma do
// prefixo [1, J] eh o lower_bound para V
// inserir os elemento na BIT com add(i, arr[i]), para todo i em [1, n
    ]

int query(int idx)// soma de um prefixo
{
  int sum = 0;
  for(; idx > 0; idx -= idx&-idx) sum += bit[idx];
  return sum;
}

void add(int idx, int k)
{
  for(int i = idx; i < MAX; i += i&-i) bit[i] += k;
}

int smallerCount(int v)
{
  return query(v);
}

int count(int v)
{
  return query(v) - query(v - 1);
}

int greaterCount(int v)
{
  return query(MAX - 3) - query(v - 1);
}

int orderOfKey(int v)
{
  return smallerCount(v);
}

int kth(int k)
{
  return bitSearch(k);
}

int main()
{
  int n;

  cin >> n;
  for(int i = 1; i <= n; i++)
  {
    cin >> arr[i];
    add(arr[i], 1);
  }
  cout << smallerCount(3) << '\n';
  cout << count(3) << '\n';
  cout << greaterCount(3) << '\n';
  cout << kth(2) << '\n';
  cout << orderOfKey(4) << '\n';

  return 0;
}
```

8.15 Dynamic Segment Tree With Vector

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

struct Node
{
  int l, r, value;
};

vector<Node> tree;

int init()
{
  tree.emplace_back();
}

void calc(int node)
{
  tree[node].value = 0;
  if(tree[node].l) tree[node].value += tree[tree[node].l].value;
  if(tree[node].r) tree[node].value += tree[tree[node].r].value;
}

void update(int node, int start, int end, int idx, int value)
{
  if(start == end)
    tree[node].value = value;
  else
  {
    int mid = (start + end) / 2;
    if(start <= idx and idx <= mid)
    {
      if(tree[node].l == 0)
      {
        tree[node].l = tree.size();
        tree.emplace_back();
      }
      update(tree[node].l, start, mid, idx, value);
    }
    else
    {
      if(tree[node].r == 0)
      {
        tree[node].r = tree.size();
        tree.emplace_back();
      }
```

```
      update(tree[node].r, mid + 1, end, idx, value);
    }
    calc(node);
  }
}

int query(int node, int start, int end, int l, int r)
{
  if(l > end or r < start) return 0;
  if(l <= start and end <= r) return tree[node].value;
  int mid = (start + end) / 2, q1 = 0, q2 = 0;
  if(tree[node].l) q1 = query(tree[node].l, start, mid, l, r);
  if(tree[node].r) q2 = query(tree[node].r, mid + 1, end, l, r);
  return q1 + q2;
}

int main()
{
  int n, q;

  cin >> n >> q;
  init();
  while(q--)
  {
    int o, l, r;
    cin >> o >> l >> r;
    if(o == 1) update(0, 0, n - 1, l - 1, r);
    else cout << query(0, 0, n - 1, l - 1, r - 1) << '\n';
  }

  return 0;
}
```

8.16 Implicit Treap

```
#include<bits/stdc++.h>
using namespace std;

struct Node
{
  int valor, priority, size, sum;
  Node *l, *r;
  bool rev;
  Node(int _valor) : rev(false), sum(_valor), valor(_valor),
    priority((rand() << 16) ^ rand()), size(1), l(nullptr), r(nullptr)
        {}
  ~Node() { delete l; delete r; }

  void recalc()
  {
    size = 1;
    sum = valor;
    if(l) size += l->size, sum += l->sum;
    if(r) size += r->size, sum += r->sum;
  }
};

struct Treap
{
  int size(Node* t) { return t ? t->size : 0; }

  int size() const { return root ? root->size : 0; }

  Node* propagate(Node* t)
  {
    if(t == nullptr) return t;
    if(t->rev)
    {
      swap(t->l, t->r);
      if(t->l != nullptr) t->l->rev ^= 1;
      if(t->r != nullptr) t->r->rev ^= 1;
      t->rev = 0;
    }
    t->recalc();
    return t;
  }

  int position(Node *t, int n)
  {
    //nao esta na treap, botar valor que noa esta no array...
    if(t == nullptr) return -1;
    propagate(t);
    if(n == size(t->l) + 1) return t->valor;
    else if(n <= size(t->l)) return position(t->l, n);
    else return position(t->r, n - size(t->l) - 1);
  }

  int at(int n)
  {
    return position(root, n);
  }

  Node* merge(Node *l, Node *r)
  {
    l = propagate(l);
    r = propagate(r);
    if(!l or !r) return l ? l : r;
    if(l->priority < r->priority)
    {
      l->r = merge(l->r, r);
      l->recalc();
      return l;
    }
    else
    {
      r->l = merge(l, r->l);
      r->recalc();
      return r;
    }
  }

  void split(Node *v, int valor, Node *&l, Node *&r)
  {
    v = propagate(v);
    l = r = nullptr;
    if(!v) return;
    if(size(v->l) < valor)
    {
      split(v->r, valor - size(v->l) - 1, v->r, r);
      l = v;
    }else
```

```
    {
      split(v->l, valor, l, v->l);
      r = v;
    }
    v->recalc();
  }

  Node * root;
  Treap() : root(nullptr) {}
  ~Treap() { delete root; }

  void insert(int valor, int pos)
  {
    Node * l, * r;
    split(root, pos - 1, l, r);
    root = merge(merge(l, new Node(valor)), r);
  }

  void erase(int valor)
  {
    Node * l, * m, * r;
    split(root, valor - 1, l, m);
    split(m, 1, m, r);
    delete m;
    root = merge(l, r);
  }

  void reverse(int l, int r)
  {
    l--; r--;
    if(l > r) swap(l, r);
    Node *a, *b, *c, *d;
    split(root, l, a, d);
    split(d, r - l + 1, b, c);
    if(b) b->rev ^= 1;
    root = merge(a, merge(b, c));
  }

  int query(int l, int r)
  {
    Node *a, *b, *c, *d;
    split(root, l - 1, a, b);
    split(b, r - l + 1, c, d);
    int ans = c->sum;
    root = merge(a, merge(c, d));
    return ans;
  }

  /*void emOrdem(Node *node)
  {
    if(node == nullptr) return;
    emOrdem(node->l);
    printf("%d ", node->valor);
    emOrdem(node->r);
  }*/

}treap;

int main()
{
  srand(time(0));
  for(int i = 1; i <= 6; i++)
  {
    int x; cin >> x;
    cout << x << ' ';
    treap.insert(x, i);
  }
  return 0;
}
```

8.17 Sparse Table RMQ

```
#include <bits/stdc++.h>
#define maxn 100000
#define maxnlog 20

using namespace std;

const double EPS = 1e-6;

int n, q, Sparse_Table[maxnlog][maxn];

void build()
{
  for(int i = 1; (1 << i) <= n; i++)
    for(int j = 0; j + (1 << i) <= n; j++)
      Sparse_Table[i][j] =  max(Sparse_Table[i-1][j],
        Sparse_Table[i-1][j+(1 << (i-1))]);
}

int range_query(int i, int j)
{
  int sz = log2(j-i+1);
  return max(Sparse_Table[sz][i],Sparse_Table[sz][j+1-(1 << sz)]);
}

int main()
{
  scanf("%d %d", &n, &q);
  for(int i = 0; i < n; i++)
    scanf("%d", &Sparse_Table[0][i]);
  build();
  for(int i = 0; i < q ; i++)
  {
    int a, b;
    scanf("%d %d", &a, &b);
    cout << range_query(a,b) << endl;
  }

  return 0;
}
```

8.18 LiChao Tree

```
#include <bits/stdc++.h>
using namespace std;
#define x real
#define y imag
```

```
typedef int ftype;
typedef complex<ftype> point;
const int OO = 0x3f3f3f3f;
const int maxn = 2e5;

point line[4 * maxn];

void init()
{
    for(int i = 0; i < 4 * maxn; i++)
        line[i] = point(0, OO);
}

ftype dot(point a, point b)
{
    return (conj(a) * b).x();
}

ftype f(point a,  ftype x)
{
    return dot(a, {x, 1});
}

void add_line(point nw, int v = 1, int l = 0, int r = maxn)
{
    int m = (l + r) / 2;
    bool lef = f(nw, l) < f(line[v], l);
    bool mid = f(nw, m) < f(line[v], m);
    if(mid)
        swap(line[v], nw);
    if(r - l == 1)
        return;
    else if(lef != mid)
        add_line(nw, 2 * v, l, m);
    else
        add_line(nw, 2 * v + 1, m, r);
}

int get(int x, int v = 1, int l = 0, int r = maxn)
{
    int m = (l + r) / 2;
    if(r - l == 1)
        return f(line[v], x);
    else if(x < m)
        return min(f(line[v], x), get(x, 2 * v, l, m));
    else
        return min(f(line[v], x), get(x, 2 * v + 1, m, r));
}

int main()
{
    init();

    point a(2, 4);
    point b(1, 3);

    add_line(a);
    add_line(b);

    cout << get(2) << '\n';

    return 0;
}
```

8.19 Wavelet Tree

```
#include <bits/stdc++.h>
using namespace std;
const int N = 100100;
const int MAX = 30 * N;
// MAX = N * log(maxX - minX)
// Queries in O(log(maxX - minX))

struct WaveletTree
{
  int arr[N], aux[N];
  int lo[MAX], hi[MAX];
  vector<int> freq[MAX];
  int lef[MAX], rig[MAX];
  int nextNode;

  WaveletTree(vector<int> a, int minX, int maxX)
  {
    int sz = a.size();
    for(int i = 0; i < sz; i++)
      arr[i] = a[i];
    nextNode = 1;
    build(0, 0, sz, minX, maxX);
  }

  int stable_partition(int s, int e, int mid)
  {
    int pivot = 0;
    for(int i = s; i < e; i++)
      aux[i] = arr[i], pivot += (arr[i] <= mid);
    int l = s, r = s + pivot;
    for(int i = s; i < e; i++)
      if(aux[i] <= mid)
        arr[l++] = aux[i];
      else
        arr[r++] = aux[i];
    return l;
  }

  void build(int node, int s, int e, int minX, int maxX)
  {
    lo[node] = minX, hi[node] = maxX;
    if(lo[node] == hi[node] or s >= e) return;

    int mid = (minX + maxX - 1) / 2;

    freq[node].resize(e - s + 1);
    freq[node][0] = 0;

    for(int i = s; i < e; i++)
      freq[node][i - s + 1] = freq[node][i - s] + (arr[i] <= mid);

    int pivot = stable_partition(s, e, mid);

    lef[node] = nextNode++, rig[node] = nextNode++;
```

```
  build(lef[node], s, pivot, minX, mid);
  build(rig[node], pivot, e, mid + 1, maxX);
}

int went_right(int node, int i)
{
  return i - freq[node][i];
}

// less than ou equal to x in range [l, r]
int lte(int l, int r, int x, int node = 0)
{
  if(l > r or x < lo[node]) return 0;
  if(hi[node] <= x) return r - l + 1;

  int l1 = freq[node][l - 1] + 1, r1 = freq[node][r];
  int l2 = went_right(node, l - 1) + 1, r2 = went_right(node, r);

  return lte(l1, r1, x, lef[node]) + lte(l2, r2, x, rig[node]);
}

// greater than ou equal to x in range [l, r]
int gte(int l, int r, int x, int node = 0)
{
  if(l > r or x > hi[node]) return 0;
  if(lo[node] >= x) return r - l + 1;

  int l1 = freq[node][l - 1] + 1, r1 = freq[node][r];
  int l2 = went_right(node, l - 1) + 1, r2 = went_right(node, r);

  return gte(l1, r1, x, lef[node]) + gte(l2, r2, x, rig[node]);
}

// counting numbers equal to x in range [l, r]
int count(int l, int r, int x, int node = 0)
{
  if(l > r or lo[node] > x or hi[node] < x) return 0;

  if(lo[node] == hi[node] and lo[node] == x) return r - l + 1;

  int l1 = freq[node][l - 1] + 1, r1 = freq[node][r];
  int l2 = went_right(node, l - 1) + 1, r2 = went_right(node, r);

  return count(l1, r1, x, lef[node]) + count(l2, r2, x, rig[node]);
}

// find kth number in range [l, r]
int kth(int l, int r, int k, int node = 0)
{
  if(l > r) return 0;
  if(lo[node] == hi[node]) return lo[node];

  int inLeft = freq[node][r] - freq[node][l - 1];
  int l1 = freq[node][l - 1] + 1, r1 = freq[node][r];

  if(k <= inLeft) return kth(l1, r1, k, lef[node]);

  int l2 = went_right(node, l - 1) + 1, r2 = went_right(node, r);

  return kth(l2, r2, k - inLeft, rig[node]);
}
};

int main()
{
  vector<int> a = {2, 5, 3, 2, 4, 2};

  WaveletTree T(a, 0, 9);

  cout << T.lte(3, 5, 3) << '\n';
  cout << T.gte(3, 5, 3) << '\n';
  cout << T.count(1, 6, 2) << '\n';
  cout << T.kth(1, 6, 5) << '\n';

  return 0;
}
```

8.20 PBDS

```
#include <bits/stdc++.h>
// Common file
#include <ext/pb_ds/assoc_container.hpp>
// Including tree_order_statistics_node_update
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

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

int main()
{
    ordered_set X;
    X.insert(2);
    X.insert(13);
    X.insert(5);
    X.insert(2);
    cout << *X.find_by_order(0) << '\n';
    cout << X.order_of_key(1) << '\n';

    return 0;
}


////////////////////////////////////////////////////


#include <bits/stdc++.h>
// Common file
#include <ext/pb_ds/assoc_container.hpp>
// Including tree_order_statistics_node_update
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

#define vi vector<int>
#define var pair<int,int>
#define ordered_multiset tree<var, null_type, less<var>
  , rb_tree_tag, tree_order_statistics_node_update>

int id = 0; map<int,vi> ids;
```

```
void insere(ordered_multiset &s, int x)
{
    s.insert({x, ++id});
    ids[x].push_back(id);
}

void apaga(ordered_multiset &s, int x)
{
  if(ids[x].empty()) return;
    s.erase({x, ids[x].back()});
    ids[x].pop_back();
}

int kth(ordered_multiset &s, int x)
{
    return s.find_by_order(x)->first;
}

int smallerCount(ordered_multiset &s, int x)
{
    return s.order_of_key({x, 0});
}

int count(ordered_multiset &s, int x)
{
    return smallerCount(s, x + 1) - smallerCount(s, x);
}

ordered_multiset::iterator find(ordered_multiset &s, int x)
{
  if(ids[x].empty())
    return s.end();
    return s.find({x, ids[x].back()});
}

int main()
{
    ordered_multiset X;

    // usar funcoes ...

    return 0;
}
```

8.21 TreeIsomorfismWithPolynomialHashing

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
const int MAX = 1e5 + 10;
const ll A = 911382323;
const ll B = 972663749;

int n, ID, degree[MAX];
map<ll, ll> formato;
bool vis[MAX];

inline ll norm(ll a)
{
    return a > B ? a = a % B : a;
}

inline ll add(ll a, ll b)
{
    a = norm(a); b = norm(b);
    return norm(a + b);
}

inline ll prod(ll a, ll b)
{
    a = norm(a); b = norm(b);
    return norm(a * b);
}

inline ll pol_hash(vector<ll> &v)
{
    ll p = 1, ans = 0;
    for(ll &w : v)
  {
    ans = add(ans, prod(p, w));
    p = prod(p, A);
    }
  return norm(ans);
}

ll dfs(int v, int p, vector<vector<int>> &G)
{
    if((int)G[v].size() == 1)
        return 1;
    vector<ll> ids;
    for(int &u : G[v])
    {
        if(u == p) continue;
        ll x = dfs(u, v, G);
        ids.push_back(x);
    }
    sort(ids.begin(), ids.end());
  ll ph = pol_hash(ids);
  if(formato.count(ph) <= 0) formato[ph] = ++ID;
    return formato[ph];
}

inline void findCenterAndComputeID(vector<vector<int>> &G, vector<ll>
    &val)
{
    memset(vis, 0, sizeof(vis));
    queue<int> fila[2];
    for(int i = 0; i < n; i++)
        if(degree[i] == 1)
            fila[0].push(i);
    int cnt = 0, turn = 0;
    while(cnt + 2 < n)
    {
        while(!fila[turn].empty())
        {
            int u = fila[turn].front(); fila[turn].pop();
            vis[u] = true;
            cnt++;
            for(int i = 0; i < G[u].size(); i++)
                if(!vis[G[u][i]])
```

```
                {
                    degree[G[u][i]]--;
                    if(degree[G[u][i]] == 1) fila[1-turn].push(G[u][i
                        ]);
                }
            }
            turn ^= 1;
        }
        for(int i = 0; i < n; i++)
        {
            if(vis[i]) continue;
            val.push_back(dfs(i, -1, G));
        }
    }

    int32_t main()
    {
        while(cin >> n)
        {
        formato.clear();
        ID = 1;
            vector<ll> val[2];
            for(int j = 0; j < 2; j++)
            {
                memset(degree, 0, sizeof(degree));
                vector<vector<int>> G(n + 1);
                for(int i = 1; i < n; i++)
                {
                    int u, v;
                    scanf(" %d %d", &u, &v); u--; v--;
                    G[u].push_back(v);
                    G[v].push_back(u);
                    degree[v]++;
                    degree[u]++;
                }
                findCenterAndComputeID(G, val[j]);
            }
            bool fl = false;
            for(ll &v0 : val[0])
                for(ll &v1 : val[1])
                    if(v0 == v1)
                        fl = true;
            puts(fl ? "S" : "N");
        }

        return 0;
    }
```

8.22 Merge Sort Tree Iterative

```
#include <bits/stdc++.h>
using namespace std;
#define OO 0x3f3f3f3f

struct MergeSortTree
{
  int n;
  vector<vector<int>> tree;

  MergeSortTree(vector<int> &a)
  {
    n = a.size();
    tree.resize(n << 1);
    for(int i = 0; i < n; i++)
      tree[i + n] = vector<int>{a[i]};
    build();
  }

  void build()
  {
    for(int i = n - 1; i > 0; --i)
    {
      int L = i << 1;
      int R = (i << 1) | 1;
      int l = 0, r = 0, sz = tree[L].size() + tree[R].size();

      tree[i].resize(sz);

      tree[L].push_back(OO);
      tree[R].push_back(OO);

      for(int j = 0; j < sz; j++)
        if(tree[L][l] < tree[R][r])
          tree[i][j] = tree[L][l++];
        else
          tree[i][j] = tree[R][r++];

      tree[L].pop_back();
      tree[R].pop_back();
    }
  }

  int queryMax(int l, int r, int x)
  {
    if(l >= r) return 0;
    int res = 0;
    for(l += n, r += n; l < r; l >>= 1, r >>= 1)
    {
      if(l & 1)
      {
        auto it = upper_bound(tree[l].begin(), tree[l].end(), x);
        int p = it - tree[l].begin();
        if(it != tree[l].end())
        {
          int p = it - tree[l].begin();
          res += (int)tree[l].size() - p;
        }
        l++;
      }
      if(r & 1)
      {
        r--;
        auto it = upper_bound(tree[r].begin(), tree[r].end(), x);
        if(it != tree[r].end())
        {
          int p = it - tree[r].begin();
          res += (int)tree[r].size() - p;
        }
      }
    }
    return res;
```

```
  }

  int queryMin(int l, int r, int x)
  {
    if(l >= r) return 0;
    int res = 0;
    for(l += n, r += n; l < r; l >>= 1, r >>= 1)
    {
      if(l & 1)
      {
        auto it = lower_bound(tree[l].begin(), tree[l].end(), x);
        if(it == tree[l].end()) res += tree[l].size();
        else res += it - tree[l].begin();
        l++;
      }
      if(r & 1)
      {
        r--;
        auto it = lower_bound(tree[r].begin(), tree[r].end(), x);
        if(it == tree[r].end()) res += tree[r].size();
        else res += it - tree[r].begin();
      }
    }
    return res;
  }
};

int32_t main()
{
  int n;
  scanf(" %d", &n);
  vector<int> v(n);
  for(int &w : v) cin >> w;
  MergeSortTree T(v);
  // query(l, r, x)      [l, r)

  return 0;
}
```

8.23 TreeIsomorfismWithMap

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5 + 10;

int n, ID, degree[MAX];
map<map<int, int>, int> formato;
bool vis[MAX];

void init()
{
    formato.clear();
    ID = 1;
}

int dfs(int v, int p, vector<vector<int>> &G)
{
    if((int)G[v].size() == 1)
        return 1;
    map<int, int> ids;
    for(int &u : G[v])
    {
        if(u == p) continue;
        int x = dfs(u, v, G);
        ids[x]++;
    }
    if(formato.count(ids) <= 0)
        formato[ids] = ++ID;
    return formato[ids];
}

inline void findCenterAndComputeID(vector<vector<int>> &G, vector<int>
     &val)
{
    memset(vis, 0, sizeof(vis));
    queue<int> fila[2];
    for(int i = 0; i < n; i++)
        if(degree[i] == 1)
            fila[0].push(i);
    int cnt = 0, turn = 0;
    while(cnt + 2 < n)
    {
        while(!fila[turn].empty())
        {
            int u = fila[turn].front(); fila[turn].pop();
            vis[u] = true;
            cnt++;
            for(int i = 0; i < G[u].size(); i++)
                if(!vis[G[u][i]])
                {
                    degree[G[u][i]]--;
                    if(degree[G[u][i]] == 1) fila[1-turn].push(G[u][i
                        ]);
                }
        }
        turn ^= 1;
    }
    for(int i = 0; i < n; i++)
    {
        if(vis[i]) continue;
        val.push_back(dfs(i, -1, G));
    }
}

int32_t main()
{
    while(cin >> n)
    {
        init();
        vector<int> val[2];
        for(int j = 0; j < 2; j++)
        {
            memset(degree, 0, sizeof(degree));
            vector<vector<int>> G(n + 1);
            for(int i = 1; i < n; i++)
            {
                int u, v;
                scanf(" %d %d", &u, &v); u--; v--;
                G[u].push_back(v);
                G[v].push_back(u);
                degree[v]++;
```

```
                degree[u]++;
            }
            findCenterAndComputeID(G, val[j]);
        }
        bool fl = false;
        for(int &v0 : val[0])
            for(int &v1 : val[1])
                if(v0 == v1)
                    fl = true;
        puts(fl ? "S" : "N");
    }

    return 0;
}
```

8.24 Centroid Decomposition

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

int n, m, Centroid_Tree[MAX], Size[MAX];
vector<int> G[MAX], cTree[MAX];
bool cut[MAX];

int dfs(int v, int p)
{
  int s = 1;
  for(const int &u : G[v])
    if(!cut[u] and u != p)
      s += dfs(u, v);
  return Size[v] = s;
}

int Find_Centroid(int v, int p, int tot)
{
  int next, cnt = 0;
  for(const int &u : G[v])
    if(!cut[u] and u != p and cnt < Size[u])
    {
      cnt = Size[u];
      next = u;
    }
  if(cnt > tot/2) return Find_Centroid(next, v, tot);
  return v;
}

void build(int v, int p)
{
  dfs(v, -1);
  int u = Find_Centroid(v, -1, Size[v]);
  cut[u] = true;
  Centroid_Tree[u] = p;
  if(p != -1)
  {
      cTree[u].push_back(p);
      cTree[p].push_back(u);
  }
  for(const int &w : G[u])
    if(!cut[w])
      build(w, u);
}

int main()
{
    int u, v;
  memset(Centroid_Tree,-1, sizeof Centroid_Tree);
    cin >> n >> m;
    while(m--)
  {
      cin >> u >> v;
      u--; v--;
    G[u].push_back(v);
    G[v].push_back(u);
  }
  build(0, -1);
    for(int i = 0; i < n; i++)
    {
        cout << i+1 << ": ";
        for(int &w : cTree[i])
            cout << w+1 << ' ';
        cout << '\n';
    }

  return 0;
}
```

8.25 Heavy Light Decomposition

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

// Heavy-Light Decomposition
vector<int> adj[MAX];
int par[MAX], h[MAX];

int chainno, chain[MAX], head[MAX], chainpos[MAX],;
int chainsz[MAX], pos[MAX], arrsz;
int sc[MAX], sz[MAX];

void dfs(int u)
{
    sz[u] = 1, sc[u] = 0; // nodes 1-indexed (0-ind: sc[u]=-1)
    for(int v : adj[u])
        if(v != par[u])
        {
            par[v] = u, h[v] = h[u]+1, dfs(v);
            sz[u]+=sz[v];
            if(sz[sc[u]] < sz[v]) sc[u] = v; // 1-indexed (0-ind: sc[u
                ]<0 or ...)
        }
}

void hld(int u)
{
    if(!head[chainno]) head[chainno] = u; // 1-indexed
    chain[u] = chainno;
    chainpos[u] = chainsz[chainno];
    chainsz[chainno]++;
```

```
    pos[u] = ++arrsz;
    if(sc[u]) hld(sc[u]);
    for(int v : adj[u]) if(v != par[u] and v != sc[u]) chainno++, hld(
        v);
}

int lca(int u, int v)
{
    while(chain[u] != chain[v])
    {
        if(h[head[chain[u]]] < h[head[chain[v]]]) swap(u, v);
        u = par[head[chain[u]]];
    }
    if(h[u] > h[v]) swap(u, v);
    return u;
}

/*int query_up(int u, int v)
{
    if(u == v) return 0;
    int ans = -1;
    while(true)
    {
        if(chain[u] == chain[v])
        {
            if (u == v) break;
            ans = max(ans, query(1, 1, n, chainpos[v]+1, chainpos[u]))
                ;
            break;
        }
        ans = max(ans, query(1, 1, n, chainpos[head[chain[u]]],
            chainpos[u]));
        u = par[head[chain[u]]];
    }
    return ans;
}

int query(int u, int v)
{
    int l = lca(u, v);
    return max(query_up(u, l), query_up(v, l));
}*/

int main()
{
    int n;
    cin >> n;
    for(int i = 1; i < n; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    dfs(1);
    hld(1);
    for(int i = 1; i <= n; i++)
        cout << chain[i] << ' ';
    puts("");
    for(int i = 1; i <= n; i++)
        cout << chainpos[i] << ' ';
    puts("");

    return 0;
}

////////////////////////////////////////////////
//Heavy Light Decomposition para encontrar a maior aresta no
// caminho de u para v em uma arvore

#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

// Heavy-Light Decomposition
vector<int> adj[MAX], W[MAX];
int par[MAX], h[MAX];
map<pair<int, int>, int> number_edge;
int chainno, chain[MAX], head[MAX], A[MAX], pos[MAX];
int sc[MAX], sz[MAX], weight[MAX], st[MAX], edge_counted, n;

void dfs(int u)
{
    sz[u] = 1, sc[u] = 0; // nodes 1-indexed (0-ind: sc[u]=-1)
    for(int i = 0; i < adj[u].size(); i++)
    {
        int v = adj[u][i], w = W[u][i];
        if(v != par[u])
        {
            weight[v] = w, par[v] = u, h[v] = h[u] + 1, dfs(v);
            sz[u]+=sz[v];
            if(sz[sc[u]] < sz[v]) sc[u] = v; // 1-indexed (0-ind: sc[u
                ]<0 or ...)
        }
    }
}

void hld(int u)
{
    if(!head[chainno]) head[chainno] = u; // 1-indexed
    chain[u] = chainno;
    pos[u] = edge_counted;
    A[edge_counted++] = weight[u];
    if(sc[u]) hld(sc[u]);
    for(int v : adj[u])
        if(v != par[u] and v != sc[u])
        {
            number_edge[{u, v}] = edge_counted;
            chainno++, hld(v);
        }
}

int lca(int u, int v)
{
    while(chain[u] != chain[v])
    {
        if(h[head[chain[u]]] < h[head[chain[v]]]) swap(u, v);
        u = par[head[chain[u]]];
    }
    if(h[u] > h[v]) swap(u, v);
    return u;
}
```

```
void build(int node, int start, int end)
{
    if(start == end)
        st[node] = A[start];
    else
    {
        int mid = (start + end) / 2;
        build(2 * node, start, mid);
        build(2 * node + 1, mid + 1, end);
        st[node] = max(st[2 * node], st[2 * node + 1]);
    }
}

int query(int node, int start, int end, int l, int r)
{
    if(l > end or start > r)
        return -1;
    if(l <= start and end <= r)
        return st[node];
    int mid = (start + end) / 2;
    int q1 = query(2 * node, start, mid, l, r);
    int q2 = query(2 * node + 1, mid + 1, end, l, r);
    return max(q1, q2);
}

void update(int node, int start, int end, int idx, int value)
{
    if(start == end)
        st[node] = A[idx] = value;
    else
    {
        int mid = (start + end) / 2;
        if(start <= idx and idx <= mid)
            update(2 * node, start, mid, idx, value);
        else
            update(2 * node + 1, mid + 1, end, idx, value);
        st[node] = max(st[2 * node], st[2 * node + 1]);
    }
}

int query_up(int u, int v)
{
    if(u == v) return 0;
    int ans = -1;
    while(true)
    {
        if(chain[u] == chain[v])
        {
            if(u == v) break;
            ans = max(ans, query(1, 0, n-1, pos[v] + 1, pos[u]));
            break;
        }
        ans = max(ans, query(1, 0, n-1, pos[head[chain[u]]], pos[u]));
        u = par[head[chain[u]]];
    }
    return ans;
}

int queryMaxEdge(int u, int v)
{
    int l = lca(u, v);
    return max(query_up(u, l), query_up(v, l));
}

void updateEdge(int u, int v, int value)
{
    int idx;
    if(number_edge.find({u, v}) != number_edge.end())
        idx = number_edge[{u, v}];
    else
        idx = number_edge[{v, u}];
    update(1, 0, n-1, idx, value);
}

int main()
{
    cin >> n;
    for(int i = 1; i < n; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u].push_back(v);
        adj[v].push_back(u);
        W[u].push_back(w);
        W[v].push_back(w);
    }
    weight[1] = -1;
    dfs(1);
    hld(1);
    build(1, 0, n-1);
    int x, y, o;
    while(cin >> o >> x >> y)
    {
        if(o == 1)
            cout << queryMaxEdge(x, y) << '\n';
        else
        {
            int w; cin >> w;
            updateEdge(x, y, w);
        }
    }
    return 0;
}
```

8.26 BIT Range Sum And Range Update

```
#include <bits/stdc++.h>
const int MAX = 1e5;

using namespace std;

struct BIT {

  int N;
  int BIT1[MAX];
  int BIT2[MAX];

  BIT(int M) {
    N = M;
  }
```

```
  void add(int *b, int pos, int x) {
    while(pos <= N) b[pos] += x, pos += pos&-pos;
  }

  void range_add(int l, int r, int x) {
    add(BIT1, l, x);
    add(BIT1, r + 1, -x);
    add(BIT2, l, x * (l - 1));
    add(BIT2, r + 1, -x * r);
  }

  int sum(int *b, int pos) {
    int s = 0;
    while(pos) s += b[pos], pos -= pos&-pos;
    return s;
  }

  int prefix_sum(int pos) {
    return sum(BIT1, pos) * pos - sum(BIT2, pos);
  }

  int range_sum(int l, int r) {
    return prefix_sum(r) - prefix_sum(l - 1);
  }
};

int main() {

  int n, q;
  cin >> n >> q;

  BIT B(n);

  while(q--) {
    int o, l, r, x;
    cin >> o >> l >> r;
    if(o == 1) {
      cout << B.range_sum(l, r) << '\n';
    } else {
      cin >> x;
        B.range_add(l, r, x);
    }
  }

  return 0;
}
```

8.27 Persistent Segment Tree

```
#include <bits/stdc++.h>
using namespace std;
const int MAX = 1e5;

struct Node
{
  int l, r, value;
};

vector<Node> tree;
vector<int> root;

void init()
{
  tree.emplace_back();
  root.push_back(0);
}

void calc(int node)
{
  tree[node].value = 0;
  if(tree[node].l) tree[node].value += tree[tree[node].l].value;
  if(tree[node].r) tree[node].value += tree[tree[node].r].value;
}

void update(int prev, int node, int start, int end, int idx, int value
    )
{
  if(start == end)
    tree[node].value = value;
  else
  {
    int mid = (start + end) / 2;
    if(start <= idx and idx <= mid)
    {
      tree[node].r = tree[prev].r;
      if(tree[node].l == 0)
      {
        tree[node].l = tree.size();
        tree.emplace_back();
      }
      update(tree[prev].l, tree[node].l, start, mid, idx, value);
    }
    else
    {
      tree[node].l = tree[prev].l;
      if(tree[node].r == 0)
      {
        tree[node].r = tree.size();
        tree.emplace_back();
      }
      update(tree[prev].r, tree[node].r, mid + 1, end, idx, value);
    }
    calc(node);
  }
}

int query(int node, int start, int end, int l, int r)
{
  if(l > end or r < start) return 0;
  if(l <= start and end <= r) return tree[node].value;
  int mid = (start + end) / 2, q1 = 0, q2 = 0;
  if(tree[node].l) q1 = query(tree[node].l, start, mid, l, r);
  if(tree[node].r) q2 = query(tree[node].r, mid + 1, end, l, r);
  return q1 + q2;
}

int main()
{
  int n, q;
```

```
  cin >> n >> q;
  init();
  while(q--)
  {
    int o, l, r;
    cin >> o >> l >> r;
    if(o == 1)
    {
      int prev = root.back();
      root.push_back(tree.size());
      tree.emplace_back();
      update(prev, root.back(), 0, n - 1, l - 1, r);
    }
    else
    {
      int version;
      scanf(" %d", &version);
      cout << query(root[version], 0, n - 1, l - 1, r - 1) << '\n';
    }
  }

  return 0;
}
```

8.28 Merge Sort Tree Range Order Statistics Queries

```
#include <bits/stdc++.h>
using namespace std;

int n;
vector<int> tree[100000];
vector<pair<int, int>> arr;

void build(int node, int start, int end)
{
    if(start == end)
        tree[node].push_back(arr[start].second);
    else
    {
        int mid = (start + end) / 2;
        build(2 * node, start, mid);
        build(2 * node + 1, mid + 1, end);
        merge(tree[2 * node].begin(), tree[2 * node].end(),
            tree[2 * node + 1].begin(), tree[2 * node + 1].end(),
            back_inserter(tree[node]));
    }
}

int query(int node, int start, int end, int l, int r, int k)
{
    if(start == end)
        return arr[start].first;
    int M = upper_bound(tree[2 * node].begin(), tree[2 * node].end(),
        r)
       - lower_bound(tree[2 * node].begin(), tree[2 * node].end(), l);
    int mid = (start + end) / 2;
    if(M >= k)
        return query(2 * node, start, mid, l, r, k);
    else
        return query(2 * node + 1, mid + 1, end, l, r, k - M);
}

int main()
{
    cin >> n;
    int aux;
    for(int i = 0; i < n; i++)
    {
        cin >> aux;
        arr.push_back({aux, i});
    }
    sort(arr.begin(), arr.end());
    build(1, 0, n-1);

    int l, r, k;
    while(cin >> l >> r >> k)
        cout << query(1, 0, n-1, l-1, r-1, k) << '\n';

    return 0;
}
```

8.29 Heavy Light Decomposition Path And Subtree Queries

```
/*
no intervalo [ in[v], out[v] ) do array A
temos a subarvore de v. Para fazer consultas
basta usar a segtree.

no intervalo [ in[nxt[v]], in[v] ] temos os vertices no
caminho de nxt[v] ate v, Em que nxt[v] esta no inicio da
cadeia da HLD. e o caminho de nxt[v] ate v faz parte da
cadeia que comeca em nxt[v].

Assim, podemos processar queries rapidamente em caminhos
e subarvores usando a mesma segment tree.

Bonus: para uma query de mudanca de raiz: se a raiz atual
for v e a consulta for na subarvore de u, entao, se u for
ancestral de v a resposta eh a consulta da arvore
total menos a consulta da subarvore enraizada pelo filho
de u que eh ancestral de v, caso contrario a consulta eh
normal como se a raiz da arvore nunca tivesse mudado.
obs: Para encontrar o filho de u que eh ancestral de v
podemos usar binary lifting, da mesma forma que usamos
para calcular lca.
*/

#include <bits/stdc++.h>
using namespace std;
const int MAX = 5 * 1e5;

int n;
vector<int> Adj[MAX], G[MAX], A;
int in[MAX], out[MAX], rin[MAX], sz[MAX], nxt[MAX], arr[MAX], t = 0;
int st[MAX], lazy[MAX], val_vertex[MAX], depth[MAX], father[MAX];

void mount(int v = 0, int p = -1)
{
```

```
  for(int &u : Adj[v])
    if(u != p)
    {
      G[v].push_back(u);
      mount(u, v);
    }
}

void dfs_sz(int v = 0, int p = 0, int d = 0)
{
    sz[v] = 1;
    depth[v] = d;
    father[v] = p;
    for(int &u: G[v])
    {
      if(u == p) continue;
        dfs_sz(u, v, d + 1);
        sz[v] += sz[u];
        if(sz[u] > sz[G[v][0]])
            swap(u, G[v][0]);
    }
}

void dfs_hld(int v = 0, int p = -1)
{
    in[v] = t++;
  rin[in[v]] = v;
    A.push_back(val_vertex[v]);
    for(int u: G[v])
    {
      if(u == p) continue;
        nxt[u] = (u == G[v][0] ? nxt[v] : u);
        dfs_hld(u, v);
    }
    out[v] = t;
}

void build(int node, int start, int end)
{
  if(start == end)
    st[node] = A[start];
  else
  {
    int mid = (start + end) / 2;
    build(2 * node, start, mid);
    build(2 * node + 1, mid + 1, end);
    st[node] = st[2 * node] + st[2 * node + 1];
  }
}

void update(int node, int start, int end, int l, int r, int value)
{
  if(lazy[node])
  {
    st[node] += (end - start + 1) * lazy[node];
    if(start != end)
    {
      lazy[2 * node] += lazy[node];
      lazy[2 * node + 1] += lazy[node];
    }
    lazy[node] = 0;
  }
  if(l > end or start > r)
    return;
  if(l <= start and end <= r)
  {
    st[node] += value * (end - start + 1);
    if(start != end)
    {
      lazy[2 * node] += value;
      lazy[2 * node + 1] += value;
    }
    return;
  }
  int mid = (start + end) / 2;
  update(2 * node, start, mid, l, r, value);
  update(2 * node + 1, mid + 1, end, l, r, value);
  st[node] = st[2 * node] + st[2 * node + 1];
}

int query(int node, int start, int end, int l, int r)
{
  if(lazy[node])
  {
    st[node] += (end - start + 1) * lazy[node];
    if(start != end)
    {
      lazy[2 * node] += lazy[node];
      lazy[2 * node + 1] += lazy[node];
    }
    lazy[node] = 0;
  }
  if(l > end or start > r)
    return 0;
  if(l <= start and end <= r)
    return st[node];
  int mid = (start + end) / 2;
  int q1 = query(2 * node, start, mid, l, r);
  int q2 = query(2 * node + 1, mid + 1, end, l, r);
  return q1 + q2;
}

int lca(int u, int v)
{
  while(nxt[u] != nxt[v])
  {
    if(depth[nxt[u]] < depth[nxt[v]])
      v = father[nxt[v]];
    else
      u = father[nxt[u]];
  }
  return depth[u] < depth[v] ? u : v;
}

void update_up(int u, int l, int value)
{
  while(true)
  {
    if(nxt[u] == nxt[l])
    {
      update(1, 0, n - 1, in[l], in[u], value);
      break;
```

```
    }
    update(1, 0, n - 1, in[nxt[u]], in[u], value);
    u = father[nxt[u]];
  }
}

// atualiza o valor de cada vertice no caminho de
void updatePath(int u, int v, int value)//u para v.
{
  int l = lca(u, v);
  update_up(u, l, value);
  update_up(v, l, value);
  update(1, 0, n - 1, in[l], in[l], -value);
}

int query_up(int u, int l)
{
  int ans = 0;
  while(true)
  {
    if(nxt[u] == nxt[l])
    {
      ans += query(1, 0, n - 1, in[l], in[u]);
      break;
    }
    ans += query(1, 0, n - 1, in[nxt[u]], in[u]);
    u = father[nxt[u]];
  }
  return ans;
}

//consulta a soma do valor de cada vertice no caminho
int queryPath(int u, int v)// de u para v.
{
  int l = lca(u, v), ans = 0;
  ans += query_up(u, l);
  ans += query_up(v, l);
  ans -= query(1, 0, n - 1, in[l], in[l]);
  return ans;
}

int main()
{
  int q;
  scanf(" %d %d", &n, &q);
  for(int i = 1; i < n; i++)
  {// ler a arvore em qualquer ordem
    int u, v;
    scanf(" %d %d", &u, &v); u--; v--;
    Adj[u].push_back(v);
    Adj[v].push_back(u);
  }
  mount();// montar a arvore direcionada

  dfs_sz();
  dfs_hld();

  // realizar consultas

  return 0;
}
```