Team notebook

October 25, 2019

Contents

1 DP

1.1 MaxSubRectangle

```
#include <bits/stdc++.h>
#define fi first
#define se second
#define pb push_back
#define mp make_pair
#define MOD 1000000007
#define pii pair<int,int>
#define LL long long
using namespace std;

int main () {
    //clock_t start = clock();
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    int r,c;   // row and col of matrix
    cin >> r >> c;
    int a[r+5][c+5];
    memset (a,0,sizeof(a));
    for (int i=1;i<=r;i++) {
        for (int j=1;j<=c;j++) {
            cin >> a[i][j];
        }
    }

    // prefix sum all columns
    for (int i=1;i<=r;i++) {
        for (int j=1;j<=c;j++) {
            a[i][j]+=a[i-1][j];
```

```
        }
    }
    int ans=0;

    for (int i=1;i<=r;i++) {
        for (int j=i;j<=r;j++) {

            // This array contains the sum of
                all numbers from row i to j
            int sums[r+5];
            for (int k=1;k<=c;k++) {
                sums[k] = a[j][k]-a[i-1][k];
            }

            // kadane's algorithm
            int maxSum = 0, curSum=0;
            for (int k=1;k<=c;k++) {
                curSum += sums[k];
                if (curSum<0) curSum=0;
                maxSum = max(maxSum, curSum);
            }
            ans=max(ans,maxSum);
        }
    }

    cout << ans << endl;

    //cerr << fixed << setprecision(3) <<
        (clock()-start)*1./CLOCKS_PER_SEC <<
        endl;
    return 0;
}
```

1.2 coinchange$_{c}$*ombination*

```
typedef long long ll;
ll compressed(){
    ll table[v+1];
    MEM(table,0);
    table[0] = 1;
    for(int i=0;i<n;i++){
        for(int j=c[i];j<=v;j++){
            table[j] += table[j-c[i]];
        }
    }
    return table[v];
}
```

1.3 coinchange$_{m}$*inimum*

```
typedef long long ll;
const ll inf = 1e18;
ll minimum_coin(){
    dp[0] = 0; // if the value is 0 then
        include nothing
    for(int i=1;i<=v;i++){
        dp[i] = inf; // set dp[i] to infinite
            if it is not possible to have a
            value i
        for(int j=1;j<=n;j++){
            if(c[j-1] <= i){
                dp[i] = min(dp[i], 1+dp[i -
                    c[j-1]]);
            }
        }
    }
    if(dp[v] == inf){
        dp[v] = -1;
    }
    return dp[v];
}
```

1.4 cuttingrod

```
int cutting_rod(){
    int dp[n+3];
    for(int i=1;i<=n;i++){
        if(i == 1){
            dp[i] = a[i-1];
        }else{
            dp[i] = a[i-1];
            for(int j=1;j<=i/2;j++){
                dp[i] = max(dp[i],dp[j] +
                    dp[i-j]);
            }
        }
    }
    return dp[n];
}
```

1.5 editdistance

```
// (remove => dp[i-1][j], insert =>
    dp[i][j-1], replace => dp[i-1][j-1])
int iterative_edit_distance(){
    int dp[n+3][m+3];
    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            if(i == 0){
                dp[i][j] = j;
            }else if(j == 0){
                dp[i][j] = i;
            }else{
                if(str1[i-1] == str2[j-1]){
                    dp[i][j] = dp[i-1][j-1];
                }else{
                    dp[i][j] = 1 +
                        minimum(dp[i-1][j],dp[i][j-1],d
                }
            }
        }
    }
    return dp[n][m];
}
```

1.6 knapsack

```
int iterative_dp(){
    for(int i=0;i<=N;i++){
        for(int j=0;j<=W;j++){
            if(i == 0 || j == 0){
                dp[i][j] = 0;
            }else{
                dp[i][j] = dp[i-1][j];
                if(weight[i-1] <= j){
```

```
                dp[i][j] = max(dp[i][j],
                    value[i-1] +
                    dp[i-1][j-weight[i-1]]);
            }
        }
    }
    return dp[N][W];
}
```

1.7 lcs

```
int iter_lcs(int n, int m){
    for(int i=0;i<=n;i++){
        for(int j=0;j<=m;j++){
            if(i == 0 || j == 0){
                dp[i][j] = 0;
            }else{
                if(a[i-1] == b[j-1]){
                    dp[i][j] = 1+dp[i-1][j-1];
                }else{
                    dp[i][j] =
                        max(dp[i-1][j],dp[i][j-1]);
                }
            }
        }
    }
    return dp[n][m];
}
```

1.8 lis

```
// O(N^2) solution
int iterative_dp(){
    int dp[n+3];
    for(int i=1;i<=n;i++){
        dp[i] = 1;
        for(int j=1;j<i;j++){
            if(a[j-1] < a[i-1]){
                dp[i] = max(dp[i],dp[j]+1);
            }
        }
    }
    return *max_element(dp+1,dp+1+n);
}

// O(N Log N) solution
int lis(){
    int tail[n+3], len = 1;
    MEM(tail,0);

    tail[0] = a[0];
    for(int i=1;i<n;i++){
        if(a[i] > tail[len-1]){
            // if there is an element bigger
                than the tail
            // change the tail to that element
            len++;
            tail[len-1] = a[i];
        }else{
            // find wether a[i] is already in
                the subsequence or not
            auto it = find(tail,tail+len,a[i]);
            if(it != tail+len){
                // if found then continue
                continue;
            }
            // if there is no a[i], then change
                the tail element to a[i]
            it =
                upper_bound(tail,tail+len,a[i]);
            *it = a[i];
        }
    }
    return len;
}
```

1.9 subsetsum

```
bool iterative_is_subset(int n, int v){
    // The value of subset[i][j] will be true
        if there is a
    // subset of set[0..i-1] with sum equal to
        j
    bool dp[n+3][v+3];
    for(int i=0;i<=n;i++){
        for(int j=0;j<=v;j++){
            if(j == 0){
                dp[i][j] = true;
            }
            else if(i == 0){
                dp[i][j] = false;
            }else{
                if(sets[i-1] <= j){
                    dp[i][j] = (dp[i-1][j] ||
                        dp[i-1][j-sets[i-1]]);
                }else{
                    dp[i][j] = dp[i-1][j];
                }
            }
        }
    }
    return dp[n][v];
}
```

2 DS

2.1 bit$_{fenwicktree}$

```
int BIT[1000], a[1000], n;

void update(int x, int val){
    for(; x<=n ; x+=x&-x){
        BIT[x] += val;
    }
}

void update2(int x, int val){
    for(; x<0 ; x|=(x+1)){
        BIT[x] += val;
    }
}

int query(int x){
    int sum = 0;
```

```
    for(; x>0 ; x-=x&-x){
        sum += BIT[x];
    }
    return sum;
}

int query2(int x){
    int sum = 0;
    for(;x>=0;x=(x&(x+1))-1){
        sum += BIT[x];
    }
    return sum
}
```

2.2 mosalgorithm

```
// Variable to represent block size. This is
    made global
// so compare() of sort can use it.
int block;

// Structure to represent a query range
struct Query
{
    int L, R;
};

// Function used to sort all queries so that
    all queries
// of the same block are arranged together
    and within a block,
// queries are sorted in increasing order of
    R values.
bool compare(Query x, Query y)
{
    // Different blocks, sort by block.
    if (x.L/block != y.L/block)
        return x.L/block < y.L/block;

    // Same block, sort by R value
    return x.R < y.R;
}

// Prints sum of all query ranges. m is
    number of queries
// n is size of array a[].
void queryResults(int a[], int n, Query q[],
    int m)
{
    // Find block size
    block = (int)sqrt(n);
    cout << "block => " << block << "\n";
    // Sort all queries so that queries of
        same blocks
    // are arranged together.
    sort(q, q + m, compare);

    for(int i=0;i<m;i++){
        cout << q[i].L << " " << q[i].R <<
            "\n";
    }
    cout << "\n";
    // Initialize current L, current R and
        current sum
    int currL = 0, currR = 0;
    int currSum = 0;

    // Traverse through all queries
    for (int i=0; i<m; i++)
    {
        // L and R values of current range
        int L = q[i].L, R = q[i].R;
        cout << L << "," << R << "\n";
        // Remove extra elements of previous
            range. For
        // example if previous range is [0, 3]
            and current
        // range is [2, 5], then a[0] and a[1]
            are subtracted
        while (currL < L)
        {
            currSum -= a[currL];
            currL++;
        }

        // Add Elements of current Range
        while (currL > L)
        {
            currSum += a[currL-1];
            currL--;
        }
        while (currR <= R)
        {
            currSum += a[currR];
            currR++;
        }

        // Remove elements of previous range.
            For example
        // when previous range is [0, 10] and
            current range
        // is [3, 8], then a[9] and a[10] are
            subtracted
        while (currR > R+1)
        {
            currSum -= a[currR-1];
            currR--;
        }
        cout << currL << " " << currR << "\n";
        // Print sum of current range
        cout << "Sum of [" << L << ", " << R
            << "] is " << currSum << endl;
    }
}
```

2.3 segtree2d

```
const int N = 1024;
int n,tc;
int t[4*N+3][4*N+3];
int a[N+3][N+3];


void build_y(int vx, int sx, int ex, int vy,
    int sy, int ey){
    if(sy == ey){
        if(sx == ex){
            t[vx][vy] = a[sx][ey];
        }else{
            t[vx][vy] =
                t[2*vx][vy]+t[2*vx+1][vy];
```

```
        }
    }else{
        int my = (sy+ey)/2;
        build_y(vx,sx,ex,2*vy,sy,my);
        build_y(vx,sx,ex,2*vy+1,my+1,ey);
        t[vx][vy] = t[vx][2*vy]+t[vx][2*vy+1];
    }
}

void build_x(int vx, int sx, int ex){
    if(sx != ex){
        int mx = (sx+ex)/2;
        build_x(2*vx,sx,mx);
        build_x(2*vx+1,mx+1,ex);
    }
    build_y(vx,sx,ex,1,1,n);
}

void update_y(int vx, int sx, int ex, int vy,
     int sy, int ey, int x, int y, int val){
    if(sy == ey){
        if(sx == ex){
            t[vx][vy] = val;
        }else{
            t[vx][vy] =
                t[2*vx][vy]+t[2*vx+1][vy];
        }
    }else{
        int my = (sy+ey)/2;
        if(y<=my){
            update_y(vx,sx,ex,2*vy,sy,my,x,y,val);
        }else{
            update_y(vx,sx,ex,2*vy+1,my+1,ey,x,y,val);
        }
        t[vx][vy] = t[vx][2*vy]+t[vx][2*vy+1];
    }
}

void update_x(int vx, int sx, int ex, int x,
     int y, int val){
    if(sx!=ex){
        int mx = (sx+ex)/2;
        if(x <= mx){
            update_x(2*vx,sx,mx,x,y,val);
        }else{
            update_x(2*vx+1,mx+1,ex,x,y,val);
        }
    }
    update_y(vx,sx,ex,1,1,n,x,y,val);
}

int sum_y(int vx, int vy, int sy, int ey, int
     l, int r){
    //printf("**%d,%d,%d\n",vy,sy,ey);
    if(l > r || vy == 0 || l > ey || r < sy){
        return 0;
    }
    if(l <= sy && ey <= r){
        return t[vx][vy];
    }
    int my = (sy+ey)/2;
    int p1 = sum_y(vx,2*vy,sy,my,l,r);
    int p2 = sum_y(vx,2*vy+1,my+1,ey,l,r);
    return p1+p2;
}

int sum_x(int vx, int sx, int ex, int lx, int
     rx, int ly, int ry){
    //printf("%d,%d,%d\n",vx,sx,ex);
    if(lx > rx || vx == 0 || ex < lx || rx <
         sx){
        return 0;
    }
    if(lx <= sx && ex <= rx){
        return sum_y(vx,1,1,n,ly,ry);
    }
    int mx = (sx+ex)/2;
    int p1 = sum_x(2*vx,sx,mx,lx,rx,ly,ry);
    int p2 = sum_x(2*vx+1,mx+1,ex,lx,rx,ly,ry);
    return p1+p2;
}
```

2.4 segtreelazy

```
const int NMAX = 100*1000;
ll t[4*NMAX+3];
ll lazy[4*NMAX+3];
ll a[NMAX+3];
int n,q;

void build(int v, int s, int e){
    if(s==e){
        t[v] = a[s];
    }else{
        int m = (s+e)/2;
        build(2*v,s,m);
        build(2*v+1,m+1,e);
        t[v] = min(t[2*v],t[2*v+1]);
    }
}

void updateRange(int v, int s, int e, int l,
     int r, ll val){
    if(lazy[v] != 0){
        t[v] += lazy[v];
        if(s != e){
            lazy[2*v] += lazy[v];
            lazy[2*v+1] += lazy[v];
        }
        lazy[v] = 0;
    }
    if(s > e || s > r || l > e){
        return;
    }
    if(l<=s && e<=r){
        t[v] += val;
        if(s != e){
            lazy[2*v] += val;
            lazy[2*v+1] += val;
        }
        return;
    }
    int m = (s+e)/2;
    updateRange(2*v,s,m,l,r,val);
    updateRange(2*v+1,m+1,e,l,r,val);
    t[v] = min(t[2*v],t[2*v+1]);
}

ll queryRange(int v, int s, int e, int l, int
     r){
    if(s > e || s > r || l > e){
        return inf;
    }
```

```
    if(lazy[v]!=0){
        t[v] += lazy[v];
        if(s!=e){
            lazy[2*v] += lazy[v];
            lazy[2*v+1] += lazy[v];
        }
        lazy[v] = 0;
    }
    if(l <= s && e <= r){
        return t[v];
    }
    int m = (s+e)/2;
    ll p1 = queryRange(2*v,s,m,l,r);
    ll p2 = queryRange(2*v+1,m+1,e,l,r);
    return min(p1,p2);
}
```

2.5 segtreelazybit

```
const int N = 5e4+5;
int n,q;
long long bits[N][33], t[4*N][33],
     lazy[4*N][33];

void build(int v, int s, int e, int k){
    if(s == e){
        t[v][k] = bits[s][k];
    }else{
        int m = (s+e)>>1;
        build(v<<1,s,m,k);
        build(v<<1|1,m+1,e,k);
        t[v][k] = t[v<<1][k] + t[v<<1|1][k];
    }
}

void update(int v, int s, int e, int l, int
     r, int k){
    if(lazy[v][k]){
        t[v][k] = (e-s+1) - t[v][k];
        if(s != e){
            lazy[v<<1][k] ^= 1;
            lazy[v<<1|1][k] ^= 1;
        }
        lazy[v][k] = 0;
    }

    if(s > r || l > e){
        return;
    }

    if(l <= s && e <= r){
        t[v][k] = (e-s+1) - t[v][k];
        if(s != e){
            lazy[v<<1][k] ^= 1;
            lazy[v<<1|1][k] ^= 1;
        }
        lazy[v][k] = 0;
        return;
    }

    int m = (s+e)>>1;
    update(v<<1,s,m,l,r,k);
    update(v<<1|1,m+1,e,l,r,k);
    t[v][k] = t[v<<1][k] + t[v<<1|1][k];
}

long long query(int v, int s, int e, int l,
     int r, int k){
    if(lazy[v][k]){
        t[v][k] = (e-s+1) - t[v][k];
        if(s != e){
            lazy[v<<1][k] ^= 1;
            lazy[v<<1|1][k] ^= 1;
        }
        lazy[v][k] = 0;
    }


    if(s > r || e < l){
        return 0;
    }

    if(l <= s && e <= r){
        return t[v][k];
    }

    int m = (s+e)>>1;
    return query(v<<1,s,m,l,r,k) +
        query(v<<1|1,m+1,e,l,r,k);
}
```

2.6 trieminxor

```
struct Trie {
    struct Node {
        Node *child[2];
        int cnt;
        Node() {
            child[0] = child[1] = NULL;
            cnt = 0;
        }
    };
    Node *head;
    Trie() {
        head = new Node();
    }
    void insert(int val) {
        Node *cur = head;
        for (int i = 30; i >= 0; i--) {
            bool v = val & (1 << i);
            ++cur -> cnt;
            if (cur -> child[v] == NULL) {
                cur -> child[v] = new Node();
            }
            cur = cur -> child[v];
        }
        ++cur -> cnt;
    }
    void erase(int val) {
        Node *cur = head;
        for (int i = 30; i >= 0; i--) {
            bool v = val & (1 << i);
            --cur -> cnt;
            cur = cur -> child[v];
        }
        --cur -> cnt;
    }
    int getMinXOR(int val, int &id) {
        Node *cur = head;
        int res = 0, valz = val;
```

```
        for (int i = 30; i >= 0; i--) {
            bool v = val & (1 << i);
            if (cur -> child[v] != NULL && cur
                -> child[v] -> cnt != 0) {
                cur = cur -> child[v];
            } else {
                cur = cur -> child[v ^ 1];
                res += (1 << i);
                valz ^= (1 << i);
            }
        }
        id = lower_bound(p + 1, p + 1 + n,
            valz) - p;
        return res;
    }
};
```

3 Geometry

3.1 AllAboutGeometry

```
Proyeksi segitiga: BC^2 = AC^2 + AB^2  2AD.AC
#define EPS 1E-9
#define PI acos(-1)
// >>>> Constructor of point
struct point {
  double x,y;
  point() { x = y = 0.0; }
  point(double _x, double _y) : x(_x), y(_y) {}
  bool operator == (point other) const {
   return (fabs(x - other.x) < EPS && (fabs(y
       - other.y) < EPS));
  }
};
// >>>> Constructor of vector
struct vec {
  double x, y;
  vec(double _x, double _y) : x(_x), y(_y) {}
};
// >>>> Constructor of line (ax + by = c)
struct line {
  double a,b,c;
};
// Distance of two points
double dist(point p1, point p2) {
  return hypot(p1.x - p2.x, p1.y - p2.y);
}
double DEG_to_RAD(double theta) {
  return theta * PI / 180.0;
}
// Rotate a point THETA degrees
point rotate(point p, double theta) {
  double rad = DEG_to_RAD(theta);
  return point(p.x * cos(rad) - p.y * sin(rad),
      p.x * sin(rad) + p.y * cos(rad));
}
// Make a line l from 2 given points
void pointsToLine(point p1, point p2, line
    &l) {
  if (fabs(p1.x - p2.x) < EPS) {
   l.a = 1.0 ; l.b = 0.0 ; l.c = -p1.x;
  } else {
   l.a = -(double)(p1.y - p2.y) / (p1.x -
       p2.x);
   l.b = 1.0;
   l.c = -(double)(l.a * p1.x) - p1.y;
  }
}
// Check if two lines are parallel
bool areParallel(line l1, line l2) {
  return (fabs(l1.a-l2.a) < EPS) &&
      (fabs(l1.b-l2.b) < EPS);
}
// Check if two lines are same
bool areSame(line l1, line l2) {
  return areParallel(l1, l2) && (fabs(l1.c -
      l2.c) < EPS);
}
// Check if two lines are intersect (at point
    P)
bool areIntersect(line l1, line l2, point &p)
    {
  if (areParallel(l1, l2)) return false;
  p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a *
      l1.b - l1.a * l2.b);
  if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x +
      l1.c);
    else p.y = -(l2.a * p.x + l2.c); return
        true;
}
// Convert 2 points to vector A -> B
vec toVec(point a, point b) {
  return vec(b.x - a.x, b.y - a.y);
}
// Scale a vector
vec scale(vec v, double s) {
  return vec(v.x * s, v.y * s);
}
// Translate P according to v
point translate(point p, vec v) {
  return point(p.x + v.x, p.y + v.y);
}
// Dot product of two vectors
double dot(vec a, vec b) {
  return a.x * b.x + a.y * b.y;
}
// Cross product of two vectors
double cross(vec a, vec b) {
  return a.x * b.y - a.y * b.x;
}
double norm_sq(vec v) {
  return v.x * v.x + v.y * v.y;
}
// Get the minimum distance of point P and
    line AB
// Line PC is the minimum distance
double distToLine(point p, point a, point b,
    point &c) {
  vec ap = toVec(a, p), ab = toVec(a,b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u));
  return dist(p,c);
}
// Get the minimum distance of point P and
    line segment AB
// Line PC is the minimum distance
double distToLineSegment(point p, point a,
    point b, point &c) {
  vec ap = toVec(a, p), ab = toVec(a,b);
  double u = dot(ap, ab) / norm_sq(ab);
  if (u < 0.0) {
   c = point(a.x, a.y);
```

```
   return dist(p,a);
  }
  if (u > 1.0) {
    c = point(b.x, b.y);
    return dist(p, b);
  }
  return distToLine(p, a, b, c);
}
// Returns angle AOB in RADIANS
double angle(point a, point o, point b) {
  vec oa = toVec(o, a), ob = toVec(o, b);
  return acos(dot(oa,ob) / sqrt(norm_sq(oa) *
      norm_sq(ob)));
}
// Heron's Formula : Find the area of
    triangle double
heronsFormula(double a, double b, double c) {
  double s = perimeter(a, b, c) * 0.5;
  return sqrt(s * (s - a) * (s - b) * (s - c));
}
// Find the radius incircle of triangle ABC
    (lengths)
double rInCircle(double ab, double bc, double
    ca) {
  return heronsFormula(ab,bc,ca) / (0.5 *
      perimeter(ab, bc, ca));
}
// Find the radius incircle of triangle ABC
    (points)
double rInCircle(point a, point b, point c) {
  return rInCircle(dist(a, b), dist(b, c),
      dist (c, a));
}
// Returns 1 if there is an incircle center,
    return 0 otherwise
// ctr will be the incircle center
// r is the same as rInCircle
int inCircle(point p1, point p2, point p3,
    point &ctr, double &r) {
  r = rInCircle(p1, p2, p3);
  if (fabs(r) < EPS) return 0;

  line l1, l2;
  double ratio = dist(p1, p2) / dist(p1, p3);
  point p = translate(p2, scale(toVec(p2, p3),
      ratio / (1 + ratio)));
  pointsToLine(p1, p, l1);
  ratio = dist(p2, p1) / dist(p2 , p3);
  p = translate(p1, scale(toVec(p1, p3), ratio
      / (1 + ratio)));
  pointsToLine(p2, p, l2);

  areIntersect(l1, l2, ctr);
  return 1;
}
// Find the radius circumcircle of triangle
    ABC (lengths)
double rCircumCircle(double ab, double bc,
    double ca) {
  return ab * bc * ca / (4.0 *
      heronsFormula(ab, bc, ca));
}
// Find the radius circumcircle of triangle
    ABC (points)
double rCircumCircle(point a, point b, point
    c) {
  return rCircumCircle(dist(a, b), dist(b, c),
      dist(c , a));
}
// Polygon Representation :
// 4 points, entered in counter clockwise
    order, 0-based indexing
// vector<point> P;
// P.push_back(point(1,1)); // P[0]
// P.push_back(point(3,3)); // P[1]
// P.push_back(point(9,7)); // P[2]
// P.push_back(point(1,7)); // P[3]
// P.push_back(P[0]); // P[n-1] = P[0]
// Checks if a polygon is convex or not
bool isConvex(const vector<point> &P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;
    bool isLeft = ccw(P[0], P[1], P[2]);
  for (int i = 1; i > sz-1; i++)
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 :
        i+2]) != isLeft)
      return false;
  return true;
}
// Line segment PQ intersect with line AB at
    this point
point lineIntersectSeg(point p, point q,
    point A, point B) {
  double a = B.y - A.y;
  double b = A.x - B.x;
  double c = B.x * A.y - A.x * B.y;
  double u = fabs(a * p.x + b * p.y + c);
  double v = fabs(a * q.x + b * q.y + c);
  return point((p.x * v + q.x * u) / (u + v),
      (p.y * v + q.y * u) / (u + v));
}
// Cuts polygon Q along the line AB
vector<point> cutPolygon(point a, point b,
    const vector<point> &Q) {
  vector<point> P;
  for (int i = 0; i < (int)Q.size(); i++) {
   double left1 = cross(toVec(a,b), toVec(a,
       Q[i])), left2 = 0;
   if (i != (int)Q.size()-1) left2 =
       cross(toVec(a, b), toVec(a, Q[i+1]));
   // Q[i] is on the left of AB
   // edge(Q[i], Q[i+1]) crosses line AB
   if (left1 > -EPS) P.push_back(Q[i]);
   if (left1 * left2 < -EPS)
     P.push_back(lineIntersectSeg(Q[i],
         Q[i+1], a, b));
  }
  if (!P.empty() && !(P.back() == P.front()))
    P.push_back(P.front());
  return P;
}
//-- Line Segment Intersection
int pyt(PII a, PII b){
    int dx=a.x-b.x;
    int dy=a.y-b.y;
    return (dx*dx + dy*dy);
}
int det(PII a, PII b, PII c){
    return ((a.x*b.y)+(b.x*c.y)+(c.x*a.y)
          -(a.x*c.y)-(b.x*a.y)-(c.x*b.y));
}
bool insec(pair<PII,PII> t1, pair<PII,PII>
    t2){
    bool hsl;
```

```
    h1=det(t1.F,t1.S, t2.F);
    h2=det(t1.F,t1.S, t2.S);
    h3=det(t2.F,t2.S, t1.F);
    h4=det(t2.F,t2.S, t1.S);
    hsl=false;
    if ((h1*h2<=0) && (h3*h4<=0) && !((h1==0)
        && (h2==0) && (h3==0) && (h4==0))){
       hsl=true;
    }
    return hasil;
}
...
//sg1 dan sg2 adalah pair<PII,PII>
if (insec(sg1,sg2)){
   le=sqrt((double)pyt(sg2.x, sg2.y));
   r1=fabs(crosp(MP(sg2.x, sg1.x),sg2)/le);
   r2=fabs(crosp(MP(sg2.x, sg1.y),sg2)/le);
   r2=r1+r2;
   dix=sg1.x.x + (r1/r2)*(sg1.y.x - sg1.x.x);
   diy=sg1.x.y + (r1/r2)*(sg1.y.y - sg1.x.y);
   //intersect here
   return MP(dix,diy);
}
// returns the area, which is half the
    determinant
// works for both convex and concave polygons
double area(vector<point> P) {
       double result = 0.0, x1, y1, x2, y2;
       for (int i = 0; i < P.size() - 1; i++)
           {
              x1 = P[i].x;
              x2 = P[i + 1].x;
              y1 = P[i].y;
              y2 = P[i + 1].y;
              result += (x1 * y2 - x2 * y1);
       }
       return fabs(result) / 2.0;
}
// returns true if point p is in either
    convex/concave polygon P
bool inPolygon(point p, const vector<point>
    &P) {
       if ((int) P.size() == 0) return false;
       double sum = 0; // assume first vertex
           = last vertex
       for (int i = 0; i < (int) P.size() -
           1; i++) {
          if (ccw(p, P[i], P[i + 1]))
                 sum += angle(P[i], p,
                     P[i + 1]); // left
                     turn/ccw
          else
                 sum -= angle(P[i], p,
                     P[i + 1]);
       } // right turn/cw
       return fabs(fabs(sum) - 2 * PI) < EPS;
}
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;
} // 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);
}
//circle-circle intersect
for(int i = 1; i < n; i++) {
       for(int j = i + 1; j <= n; j++) {
              double d = dist(P[i], P[j]);
              double r0 = P[i].r, x0 =
                  P[i].x, y0 = P[i].y
              double r1 = P[j].r, x1 =
                  P[j].x, y1 = P[j].y;
              point center;
              if (d > r0 + r1) continue;
              if (d < fabs(r0 - r1) ||
                  fabs(d) < EPS) {
                 if (r0 < r1) center =
                     P[i];
                 else center = P[j];
              } else {
                     double a = (r0*r0 -
                         r1*r1 + d*d)/(2*d);
                     double h = sqrt(r0*r0 -
                         a*a);
                     double x2 = x0 + a*(x1 -
                         x0)/d;
                     double y2 = y0 + a*(y1 -
                         y0)/d;
                     double translationY =
                         h*(y1 - y0)/d;
                     double translationX =
                         h*(x1 - x0)/d;
                     center.x = x2 +
                         translationY;
                     center.y = y2 -
                         translationX;
                     ans = max(ans,
                         go(center));
                     center.x = x2 -
                         translationY;
                     center.y = y2 +
                         translationX;
              }
              ans = max(ans, go(center));
       }
}
// line segment with circle intersect
private int FindLineCircleIntersections(
    float cx, float cy, float radius,
    PointF point1, PointF point2,
    out PointF intersection1, out PointF
        intersection2)
{
    float dx, dy, A, B, C, det, t;
    dx = point2.X - point1.X;
    dy = point2.Y - point1.Y;
    A = dx * dx + dy * dy;
    B = 2 * (dx * (point1.X - cx) + dy *
        (point1.Y - cy));
    C = (point1.X - cx) * (point1.X - cx) +
       (point1.Y - cy) * (point1.Y - cy) -
       radius * radius;
    det = B * B - 4 * A * C;
    if ((A <= 0.0000001) || (det < 0)) {
```

```
        // No real solutions.
        intersection1 = new PointF(float.NaN,
            float.NaN);
        intersection2 = new PointF(float.NaN,
            float.NaN);
        return 0;
    }    else if (det == 0)  {
        // One solution.
        t = -B / (2 * A);
        intersection1 =
            new PointF(point1.X + t * dx,
                point1.Y + t * dy);
        intersection2 = new PointF(float.NaN,
            float.NaN);
        return 1;
    } else {
      // Two solutions.
      t = (float)((-B + Math.Sqrt(det)) / (2 *
          A));
      intersection1 = new PointF(point1.X + t
          * dx, point1.Y + t * dy);
      t = (float)((-B - Math.Sqrt(det)) / (2 *
          A));
      intersection2 = new PointF(point1.X + t
          * dx, point1.Y + t * dy);
      return 2;
    }
}
```

3.2 ConvexHull

```
#include <bits/stdc++.h>
#define x first
#define y second
#define pb push_back
#define mp make_pair
#define pdd pair<double,double>
#define LL long long
#define INF 1e8

using namespace std;

pdd p[100115],c[100115],P0;

double triangleArea(pdd a,pdd b, pdd c) {
    return (a.x*(b.y-c.y) + b.x*(c.y-a.y) +
        c.x*(a.y-b.y));
}

double sqDist(pdd a,pdd b) {
    return ((a.x-b.x)*(a.x-b.x) +
        (a.y-b.y)*(a.y-b.y));
}

bool cmp(pdd a,pdd b) {
    double d=triangleArea(P0,a,b);
    if (d<0) return 0;
    return !(d==0&&sqDist(P0,a)>sqDist(P0,b));
}

bool normal(pdd a,pdd b) {
    if (a.x==b.x) {
        return a.y<b.y;
    } else {
        return a.x<b.x;
    }
}

bool same(pdd a,pdd b) {
    return a.x==b.x && a.y==b.y;
}

void nosame(int &np) {
    sort (p,p+np,normal);
    np=unique(p,p+np,same)-p;
}

void convexhull(int &np, int &nc) {
    int pos=0,j;
    for (int i=1;i<np;i++) {
        if (p[i].y<p[pos].y ||
            (p[i].y==p[pos].y &&
            p[i].x<p[pos].x)) pos=i;
    }
    swap(p[0],p[pos]);
    P0 = p[0];
    sort(&p[1],&p[np],cmp);
    for (int i=0;i<3;i++) {
        c[i]=p[i];
    }
    for (int i=j=3;i<np;i++) {
        while
            (triangleArea(c[j-2],c[j-1],p[i])<0)
            j--;
        c[j++]=p[i];
    }
    nc=j;
}

int main () {
    int np,nc;
    cin >> np;
    for (int i=0;i<np;i++) cin >> p[i].x >>
        p[i].y;
    nosame(np);
    convexhull(np,nc);
    sort (c,c+nc,cmp);
    c[nc]=c[0];
    // c contains points that form the convex
        hull, sorted counterclockwise
    for (int i=0;i<nc;i++) cout << c[i].x << "
        " << c[i].y << endl;
    return 0;
}
```

3.3 PointInsidePolygon

```
bool pointInsidePolygon(pdd p, pdd c[], int
    nc) {
    // p point, c polygon points, nc number of
        polygon points
    pdd center = mp(0,0);
    for (int i=0;i<nc;i++) {
        center = mp(center.x+c[i].x,
            center.y+c[i].y);
    }
    center = mp(center.x*(1.0/nc),
        center.y*(1.0/nc));
    bool isInside = 1;
    for (int i=0;i<nc;i++) {
        pdd u;
```

```
        u=mp(c[i+1].x-c[i].x,c[i+1].y-c[i].y);
        pdd v1 = mp(center.x-c[i].x,
             center.y-c[i].y);
        pdd v2 = mp(p.x-c[i].x, p.y-c[i].y);
        if (u.x*v2.y-u.y*v2.x==0) {
            isInside = 1;
            break;
        }
        if ((u.x*v1.y-u.y*v1.x) *
             (u.x*v2.y-u.y*v2.x)<0) {
            isInside = 0;
            break;
        }
    }
    return isInside;
}
```

4 Graph

4.1 LCAbinarylifting

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

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >=
        tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}
```

4.2 LCArmq

```
struct LCA {
    vector<int> height, euler, first, segtree;
    vector<bool> visited;
    int n;

    LCA(vector<vector<int>> &adj, int root =
        0) {
        n = adj.size();
        height.resize(n);
        first.resize(n);
        euler.reserve(n * 2);
        visited.assign(n, false);
        dfs(adj, root);
        int m = euler.size();
        segtree.resize(m * 4);
        build(1, 0, m - 1);
    }

    void dfs(vector<vector<int>> &adj, int
        node, int h = 0) {
        visited[node] = true;
        height[node] = h;
        first[node] = euler.size();
        euler.push_back(node);
        for (auto to : adj[node]) {
            if (!visited[to]) {
                dfs(adj, to, h + 1);
                euler.push_back(node);
            }
        }
    }

    void build(int node, int b, int e) {
        if (b == e) {
            segtree[node] = euler[b];
        } else {
            int mid = (b + e) / 2;
            build(node << 1, b, mid);
            build(node << 1 | 1, mid + 1, e);
            int l = segtree[node << 1], r =
                segtree[node << 1 | 1];
            segtree[node] = (height[l] <
                height[r]) ? l : r;
        }
    }

    int query(int node, int b, int e, int L,
        int R) {
        if (b > R || e < L)
            return -1;
        if (b >= L && e <= R)
            return segtree[node];
        int mid = (b + e) >> 1;

        int left = query(node << 1, b, mid, L,
            R);
```

```
        int right = query(node << 1 | 1, mid +
            1, e, L, R);
        if (left == -1) return right;
        if (right == -1) return left;
        return height[left] < height[right] ?
            left : right;
    }

    int lca(int u, int v) {
        int left = first[u], right = first[v];
        if (left > right)
            swap(left, right);
        return query(1, 0, euler.size() - 1,
            left, right);
    }
};
```

4.3 MST$_{K}ruskal$

```
struct E{
    int u,v,w;
};

const int N = 100;
int T,n,m,a,b,c;
E temp;
vector<E> graph; // Store the inputted graph
    (u, v, w).
vector<E> selected_edges; // Store the edges
    which is selected for the MST from given
    graph.
int parent[N+2];

bool cstm(E a, E b){
    return a.w < b.w;
}

int findParent(int r){
    if(r == parent[r])return r;
    return parent[r] = findParent(parent[r]);
}

int Kruskal_MST(){

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

    sort(graph.begin(),graph.end(),cstm);

    /*for(int i=0;i<graph.size();i++){
        cout << graph[i].w << "\n";
    }*/

    int edgeCounter = 0, totalCost = 0;

    int len = graph.size();

    for(int i=0; i<len;i++){

        int parent_of_u =
            findParent(graph[i].u);
        int parent_of_v =
            findParent(graph[i].v);

        if(parent_of_u != parent_of_v){
            parent[parent_of_u] = parent_of_v;
            totalCost += graph[i].w;
            selected_edges.pb(graph[i]);

            edgeCounter++;
            if(edgeCounter == n-1)
                break;
        }

    }
    return totalCost;
}
```

4.4 MST$_{P}rim$

```
ll minSpanningTree(vector<pll> a[]){

    ll mst = 0;

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

// create a vector key and initialize all
    of it to inf
vector<ll> key(n+2,inf);

// to store a parent which in turn store
    MST (optional)
vector<ll> parent(n+2,-1);

// To keep track of vertices included in
    MST
vector<bool> inMST(n+2,false);

pq.push(mp(0,s));
key[s] = 0;
while(!pq.empty()){
    int u = pq.top().ss;
    pq.pop();

    // include vertex u to the MST
    inMST[u] = true;

    for(int j=0; j<a[u].size();j++){
        int v = a[u][j].ff;
        int weight = a[u][j].ss;

        // if v is not yet in MST and
            weight of (u,v) is smaller
        // than the current key of v
        if(inMST[v] == false && key[v] >
            weight){
            // update the key[v]
            key[v] = weight;
            pq.push(mp(key[v],v));
            parent[v] = u;
        }
    }
}

// print edges of MST using parent array
for (int i=1;i<=n;i++){
    cout << i << " => " << parent[i] << "
        => " << key[i] << "\n";
    mst += key[i];
}
return mst;
```

```
}
```

4.5 MaxBipartiteMatching

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

using namespace std;

bool __hopcroftkarp_bfs(int n, int m,
    vector<int>* adjlist, int *pair_u, int
    *pair_v, int *dist) {
       queue<int> que;
       for (int i = 0; i < n; i++)
              if (pair_u[i] < 0)
                     dist[i] = 0, que.push(i);
              else
                     dist[i] = n+m+1;
       dist[-1] = n+m+1;
       while (!que.empty()) {
              int t = que.front(); que.pop();
              if (dist[t] < dist[-1])
                     for (int i = 0; i <
                         (int)
                         adjlist[t].size();
                         i++)
                            if
                                (dist[pair_v[adjlist[t][i]]]
                                == n+m+1) {
                                   dist[pair_v[adjlist[t][i]]]
                                       =
                                       dist[t]
                                       + 1;
                                   que.push(pair_v[adjlist[t][i]]);
                                }
       }
       return dist[-1] != n+m+1;
}

bool __hopcroftkarp_dfs(int n, int m,
    vector<int>* adjlist, int *pair_u, int
    *pair_v, int *dist, int node) {
       if (node != -1) {
              for (int i = 0; i < (int)
                  adjlist[node].size(); i++)
                     if
                         (dist[pair_v[adjlist[node][i]]]
                         == dist[node] + 1 &&
                         __hopcroftkarp_dfs(n,
                             m, adjlist,
                             pair_u, pair_v,
                             dist,
                             pair_v[adjlist[node][i]]))
                             {
                            pair_v[adjlist[node][i]]
                                = node;
                            pair_u[node] =
                                adjlist[node][i];
                            return 1;
                     }
              dist[node] = n+m+1;
              return 0;
       }
       return 1;
}

int hopcroftkarp(int n, int m, vector<int>
    *adjlist) {
       int* pair_u = new int[n+1] + 1;
       int* pair_v = new int[m+1] + 1;
       int* dist = new int[n+1] + 1;
       for (int i = -1; i < n; i++)
              pair_u[i] = -1;
       for (int i = -1; i < m; i++)
              pair_v[i] = -1;

       int matching = 0;
       while (__hopcroftkarp_bfs(n, m,
           adjlist, pair_u, pair_v, dist))
              for (int i = 0; i < n; i++)
                     if (pair_u[i] == -1 &&
                         __hopcroftkarp_dfs(n,
                         m, adjlist, pair_u,
                         pair_v, dist, i))
                            matching++;

       delete [] (pair_u-1);
       delete [] (pair_v-1);
       delete [] (dist-1);
       return matching;
}

int main() {
    int n,m,p; // n = set A, m = set B, p =
        edges
    vector<int> adjlist[50000];
    cin >> n >> m >> p;
       for (int i=1;i<=p;i++) {
              int x,y; scanf("%d%d", &x, &y);
                  x--; y--;
              adjlist[x].push_back(y);
       }
       printf("%d\n",
           hopcroftkarp(n,m,adjlist));
}
```

4.6 MaxFlow

```
#include <bits/stdc++.h>
#define fi first
#define se second
#define pb push_back
#define mp make_pair
#define MOD 1000000007
#define pii pair<int,int>
#define LL long long
using namespace std;

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

struct Dinic {
    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<vector<int>> adj;
    int n, m = 0;
```

```
    int s, t;
    vector<int> level, ptr;
    queue<int> q;

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

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

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

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

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

int main () {
    int n,m;
    cin >> n >> m; // n = vertice, m =
        undirected edge
    Dinic d(m,1,n);
    for (int i=1;i<=m;i++) {
        int x,y,cap;
        cin >> x >> y >> cap;
        d.add_edge(x,y,cap);
        d.add_edge(y,x,cap);
    }
    cout << d.flow() << endl;
    return 0;
}
```

4.7 djikstra

```
void Djikstra(vector<pii> a[]){
    priority_queue<pii, vector<pii>,
        greater<pii> > pq;

    vector<int> dist(n+2,inf);
    dist[s] = 0;
    pq.push(mp(dist[s],s));
    while(!pq.empty()){
        int u = pq.top().ss;
        pq.pop();

        for(int j=0; j<a[u].size(); j++){
            int v = a[u][j].ff;
            int weight = a[u][j].ss;
            if(dist[v] > dist[u]+weight){
                //cout << v << " => " <<
                    dist[u]+weight << "\n";
                dist[v] = dist[u]+weight;
                pq.push(mp(dist[v],v));
            }
        }
    }

    for(int i=1;i<=n;i++){
        cout << i << " => " << dist[i] << "\n";
        // if dist[i] is still inf, then it is
            not connected to s
    }

}
```

4.8 eulerpath$_{h}ielholzer$

```
// stack St;
// put start vertex in St;
// until St is empty
//   let V be the value at the top of St;
//   if degree(V) = 0, then
//     add V to the answer;
//     remove V from the top of St;
//   otherwise
//     find any edge coming out of V;
//     remove it from the graph;
//     put the second end of this edge in St;
using namespace std;

int main() {
    std::ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<int> bp(n - 1), cp(n - 1);
    set<int> s;
    for (int i = 0; i < n - 1; ++i) {
        cin >> bp[i];
    }
    for (int i = 0; i < n - 1; ++i) {
        cin >> cp[i];
    }
    for (int i = 0; i < n - 1; ++i) {
        if (bp[i] > cp[i]) {
            cout << -1 << "\n";
            return 0;
        }
        s.insert(bp[i]);
        s.insert(cp[i]);
    }
    map<int, int> id;
    vector<int> inv;
    int cur = 0;
    for (int x : s) {
        inv.push_back(x);
        id[x] = cur++;
    }
    vector<multiset<int>> g(cur);
    for (int i = 0; i < n - 1; ++i) {
        int u = id[bp[i]], v = id[cp[i]];
        g[u].insert(v);
        g[v].insert(u);
    }
    int cur_v = 0, cnt_odd = 0;
    for (int u = 0; u < cur; ++u) {
        if ((int)g[u].size() % 2 == 1) {
            cur_v = u;
            ++cnt_odd;
        }
    }
    if (cnt_odd != 0 && cnt_odd != 2) {
        cout << -1 << "\n";
        return 0;
    }
    vector<int> eul_path;
    stack<int> cur_path;
    cur_path.push(cur_v);
    while (!cur_path.empty()) {
        if (!g[cur_v].empty()) {
            cur_path.push(cur_v);
            int nxt_v = *g[cur_v].begin();
            g[cur_v].erase(g[cur_v].begin());
            g[nxt_v].erase(g[nxt_v].find(cur_v));
            cur_v = nxt_v;
        } else {
            eul_path.push_back(cur_v);
            cur_v = cur_path.top();
            cur_path.pop();
        }
    }
    if ((int)eul_path.size() < n) {
        cout << -1 << "\n";
        return 0;
    }
    for (int u : eul_path) {
        cout << inv[u] << " ";
    }
    cout << "\n";

    return 0;
}
```

4.9 findSCC

```
vector < vector<int> > g, gr;
vector<bool> used;
vector<int> order, component;

void dfs1 (int v) {
    used[v] = true;
    for (size_t i=0; i<g[v].size(); ++i)
        if (!used[ g[v][i] ])
            dfs1 (g[v][i]);
    order.push_back (v);
}

void dfs2 (int v) {
    used[v] = true;
    component.push_back (v);
    for (size_t i=0; i<gr[v].size(); ++i)
        if (!used[ gr[v][i] ])
            dfs2 (gr[v][i]);
}

int main() {
    int n;
    ... reading n ...
    for (;;) {
        int a, b;
        ... reading next edge (a,b) ...
        g[a].push_back (b);
        gr[b].push_back (a);
    }

    used.assign (n, false);
    for (int i=0; i<n; ++i)
        if (!used[i])
            dfs1 (i);
    used.assign (n, false);
    for (int i=0; i<n; ++i) {
        int v = order[n-1-i];
        if (!used[v]) {
            dfs2 (v);
            ... printing next component ...
            component.clear();
        }
    }
```

```
}
```

4.10 findarticularpoint

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of
    graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p != -1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}
```

4.11 findbriges

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of
    graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                IS_BRIDGE(v, to);
        }
    }
}

void find_bridges() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
}
```

4.12 findcycle

```
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v !=
             cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
```

```
}
```

4.13 floydwarshall

```
void floydWarshall(){
    for(int k=0;k<N;k++){
        for(int i=0;i<N;i++){
            for(int j=0;j<N;j++){
                if(a[i][j] > a[i][k]+a[k][j]){
                    a[i][j] = a[i][k]+a[k][j];
                }
            }
        }
    }

    // checking negative cycle
    for(int i=0;i<N;i++){
        for(int j=0;j<N;j++){
            if(a[i][j] < 0){
                cout << "It contains negative
                    cycle!\n";
                return;
            }
        }
    }
}
```

4.14 negativecycle

```
struct Edge {
    int a, b, cost;
};

int n, m;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n,INF);
    vector<int> p(n, -1);
    int source = 1;
    d[source] = 0;
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if (d[e.a] + e.cost < d[e.b]) {
                d[e.b] = d[e.a] + e.cost;
                p[e.b] = e.a;
                x = e.b;
            }
        }
    }

    if (x == -1) {
        cout << "No negative cycle found.";
    } else {
        for (int i = 0; i < n; ++i)
            x = p[x];

        vector<int> cycle;
        for (int v = x;; v = p[v]) {
            cycle.push_back(v);
            if (v == x && cycle.size() > 1)
                break;
        }
        reverse(cycle.begin(), cycle.end());

        cout << "Negative cycle: ";
        for (int v : cycle)
            cout << v << ' ';
        cout << endl;
    }
}
```

4.15 tarjanSCC

```
int n, m;
struct Node
{
    vector<int> adj;
};
Node graf[MAX_N];
stack<int> Stack;
bool onStack[MAX_N];
int Indices, Index[MAX_N], LowLink[MAX_N],
    component[MAX_N], numComponents;

// Complexity : O(V + E)

void tarjanDFS(int i)
{
    Index[i] = ++Indices;
    LowLink[i] = Indices;
    Stack.push(i); onStack[i] = true;
    for (int j=0;j<graf[i].adj.size();j++)
    {
        int w = graf[i].adj[j];
        if (Index[w] == 0)
        {
            tarjanDFS(w);
            LowLink[i] = min(LowLink[i],
                LowLink[w]);
        }
        else if (onStack[w])
        {
            LowLink[i] = min(LowLink[i],
                Index[w]);
        }
    }
    if (LowLink[i] == Index[i])
    {
        int w = 0;
        do
        {
            w = Stack.top(); Stack.pop();
            component[w] = numComponents;
            onStack[w] = false;
        } while (i != w && !Stack.empty());
        numComponents++;
    }
}

void Tarjan()
{
    Indices = 0;
    while (!Stack.empty()) Stack.pop();
```

```
    for (int i=n;i>0;i--) onStack[i] =
        LowLink[i] = Index[i] = 0;
    numComponents = 0;
    for (int i=n;i>0;i--) if (Index[i] == 0)
        tarjanDFS(i);
}
```

4.16 toposort$_k$ahn

```
void Graph::topologicalSort()
{
    // Create a vector to store indegrees of
        all
    // vertices. Initialize all indegrees as 0.
    vector<int> in_degree(V, 0);

    // Traverse adjacency lists to fill
        indegrees of
    // vertices. This step takes O(V+E) time
    for (int u=0; u<V; u++)
    {
        list<int>::iterator itr;
        for (itr = adj[u].begin(); itr !=
            adj[u].end(); itr++)
            in_degree[*itr]++;
    }

    // Create an queue and enqueue all
        vertices with
    // indegree 0
    queue<int> q;
    for (int i = 0; i < V; i++)
        if (in_degree[i] == 0)
            q.push(i);

    // Initialize count of visited vertices
    int cnt = 0;

    // Create a vector to store result (A
        topological
    // ordering of the vertices)
    vector <int> top_order;

    // One by one dequeue vertices from queue
        and enqueue
    // adjacents if indegree of adjacent
        becomes 0
    while (!q.empty())
    {
        // Extract front of queue (or perform
            dequeue)
        // and add it to topological order
        int u = q.front();
        q.pop();
        top_order.push_back(u);

        // Iterate through all its
            neighbouring nodes
        // of dequeued node u and decrease
            their in-degree
        // by 1
        list<int>::iterator itr;
        for (itr = adj[u].begin(); itr !=
            adj[u].end(); itr++)

            // If in-degree becomes zero, add
                it to queue
            if (--in_degree[*itr] == 0)
                q.push(*itr);

        cnt++;
    }
    // Check if there was a cycle
    if (cnt != V)
    {
        cout << "There exists a cycle in the
            graph\n";
        return;
    }
    // Print topological order
    for (int i=0; i<top_order.size(); i++)
        cout << top_order[i] << " ";
    cout << endl;
}
```

5 Math

5.1 BigInteger

```
const int BASE_LENGTH = 2;
const int BASE = (int) pow(10, BASE_LENGTH);
const int MAX_LENGTH = 500;

string int_to_string(int i, int width, bool
    zero) {
    string res = "";
    while (width--) {
        if (!zero && i == 0) return res;
        res = (char)(i%10 + '0') + res;
        i /= 10;
    }
    return res;
}

struct bigint {
    int len, s[MAX_LENGTH];

    bigint() {
        memset(s, 0, sizeof(s));
        len = 1;
    }

    bigint(unsigned long long num) {
        len = 0;
        while (num >= BASE) {
            s[len] = num % BASE;
            num /= BASE;
            len ++;
        }
        s[len++] = num;
    }

    bigint(const char* num) {
        int l = strlen(num);
        len = l/BASE_LENGTH;
        if (l % BASE_LENGTH) len++;
        int index = 0;
        for (int i = l - 1; i >= 0; i -=
            BASE_LENGTH) {
```

```
        int tmp = 0;
        int k = i - BASE_LENGTH + 1;
        if (k < 0) k = 0;
        for (int j = k; j <= i; j++) {
            tmp = tmp*10 + num[j] - '0';
        }
        s[index++] = tmp;
    }
}

void clean() {
    while(len > 1 && !s[len-1]) len--;
}

string str() const {
    string ret = "";
    if (len == 1 && !s[0]) return "0";
    for(int i = 0; i < len; i++) {
        if (i == 0) {
            ret += int_to_string(s[len - i
                - 1], BASE_LENGTH, false);
        } else {
            ret += int_to_string(s[len - i
                - 1], BASE_LENGTH, true);
        }
    }
    return ret;
}

unsigned long long ll() const {
    unsigned long long ret = 0;
    for(int i = len-1; i >= 0; i--) {
        ret *= BASE;
        ret += s[i];
    }
    return ret;
}

bigint operator + (const bigint& b) const {
    bigint c = b;
    while (c.len < len) c.s[c.len++] = 0;
    c.s[c.len++] = 0;
    bool r = 0;
    for (int i = 0; i < len || r; i++) {
        c.s[i] += (i<len)*s[i] + r;
        r = c.s[i] >= BASE;
        if (r) c.s[i] -= BASE;
    }
    c.clean();
    return c;
}

bigint operator - (const bigint& b) const {
    if (operator < (b)) throw "cannot do
        subtract";
    bigint c = *this;
    bool r = 0;
    for (int i = 0; i < b.len || r; i++) {
        c.s[i] -= b.s[i];
        r = c.s[i] < 0;
        if (r) c.s[i] += BASE;
    }
    c.clean();
    return c;
}

bigint operator * (const bigint& b) const {
    bigint c;
    c.len = len + b.len;
    for(int i = 0; i < len; i++)
        for(int j = 0; j < b.len; j++)
            c.s[i+j] += s[i] * b.s[j];
    for(int i = 0; i < c.len-1; i++){
        c.s[i+1] += c.s[i] / BASE;
        c.s[i] %= BASE;
    }
    c.clean();
    return c;
}

bigint operator / (const int b) const {
    bigint ret;
    int down = 0;
    for (int i = len - 1; i >= 0; i--) {
        ret.s[i] = (s[i] + down * BASE) / b;
        down = s[i] + down * BASE -
            ret.s[i] * b;
    }
    ret.len = len;
    ret.clean();
    return ret;
}

    bool operator < (const bigint& b) const {
        if (len < b.len) return true;
        else if (len > b.len) return false;
        for (int i = 0; i < len; i++)
            if (s[i] < b.s[i]) return true;
            else if (s[i] > b.s[i]) return
                false;
        return false;
    }

    bool operator == (const bigint& b) const {
        return !(*this<b) && !(b<(*this));
    }

    bool operator > (const bigint& b) const {
        return b < *this;
    }
};
```

5.2 BitsetSieve

```
/**
* Description :Test Primality up to n in
    O(log(logn))
*/

const int SZ = 1e7;
bitset<SZ> bs;
vector<long long> primes;

void sieve(){
    bs.set();
    bs[0] = false; bs[1] = false;
    for (long long i = 2; i <= SZ; i++){
        if (bs[i]){
            primes.push_back(i);
            for (long long j = i * i; j <= SZ;
                j+=i)
                bs[j] = false;
        }
```

```
    }
}
```

5.3 CRT

```
#include <bits/stdc++.h>
using namespace std;
  // Returns modulo inverse of a with respect
      to m using extended
// Euclid Algorithm. Refer below post for
    details:
//
    https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/
int inv(int a, int m) {
    int m0 = m, t, q;
    int x0 = 0, x1 = 1;
    if (m == 1)
       return 0;
    // Apply extended Euclid Algorithm
    while (a > 1) {
        // q is quotient
        q = a / m;
        t = m;
        // m is remainder now, process same as
        // euclid's algo
        m = a % m, a = t;
        t = x0;
        x0 = x1 - q * x0;
        x1 = t;
    }
    // Make x1 positive
    if (x1 < 0)
       x1 += m0;
    return x1;
}
// k is size of num[] and rem[]. Returns the
    smallest
// number x such that:
//  x % num[0] = rem[0],
//  x % num[1] = rem[1],
//  ..................
//  x % num[k-2] = rem[k-1]
// Assumption: Numbers in num[] are pairwise
    coprime
// (gcd for every pair is 1)
int findMinX(int num[], int rem[], int k) {
    // Compute product of all numbers
    int prod = 1;
    for (int i = 0; i < k; i++)
        prod *= num[i];
    // Initialize result
    int result = 0;
    // Apply above formula
    for (int i = 0; i < k; i++) {
        int pp = prod / num[i];
        result += rem[i] * inv(pp, num[i]) *
            pp;
    }
    return result % prod;
}
```

5.4 InverseModulo

```
/**
* Description : find x such that ax = 1 mod m
*/
/* case 1 : when(gcd(a,m) = 1) */
/* use extended euclid : find x such that ax
    + my = 1 */
/* store x, y, and d as global variables */
/* d = gcd */
void extendedEuclid(int a, int b) {
  if (b == 0) { x = 1; y = 0; d = a; return; }
  /* base case */
  extendedEuclid(b, a % b);
  /* similar as the original gcd */
  int x1 = y;
  int y1 = x - (a / b) * y;
  x = x1;
  y = y1;
}
/* compute the first case inverse modulo*/
int firstInverseModulo(int a, int m){
  /* produces x and y, such that ax + my = 1 */
  /* return a^-1 mod m */
  extendedEuclid(a, m);
  return (x + m)%m;
}
/* case 2 : m is prime */
/* a^(m-1) = 1 mod m */
/* a^(m-2) = a^-1 mod m */
int power(int a,int b){
  int res = 1;
  while (b > 0){
    if (b%2 == 1)
      res *= a;
    b /= 2;
    a *= a;
  }
  return res;
}
int secondInverseModulo(int a,int m){
  return power(a, m-2);
}
```

5.5 MatrixExpo

```
/*
    This is an implementation of Matrix
        Exponentiation made in C++.
    In this case, the base cases are f(0) = 0,
        f(1) = 1, f(2) = 2.
    The reccurence relation is f(n) = f(n-1) +
        2f(n-2) + 3f(n-3).
    This code calculates f(n) % 1000000007.
*/

#include <bits/stdc++.h>
#define LL long long
#define MOD 1000000007

using namespace std;

const LL sz = 3; // size of matrix

// A utility function to multiply two
    matrices, result of multiplication stored
    in a.
```

```
void multiply(LL a[sz][sz], LL b[sz][sz]) {
    LL mul[sz][sz];
    for (LL i = 0; i < sz; i++) {
        for (LL j = 0; j < sz; j++) {
            mul[i][j] = 0;
            for (LL k = 0; k < sz; k++) {
                mul[i][j] +=
                    (a[i][k]%MOD)*(b[k][j]%MOD);
                mul[i][j] %= MOD;
            }
        }
    }
    for (LL i=0; i<sz; i++) {
        for (LL j=0; j<sz; j++) {
            a[i][j] = mul[i][j];
        }
    }
}

// Function to compute F raise to power of n.
void power(LL F[sz][sz], LL n) {
    LL res[sz][sz] = {{1,0,0},{0,1,0},{0,0,1}};
    while (n) {
        if (n & 1) {
            multiply(res,F);
        }
        n = n >> 1;
        multiply(F, F);
    }
    for (int i=0;i<sz;i++) {
        for (int j=0;j<sz;j++) {
            F[i][j] = res[i][j];
        }
    }
}

// Driver code
int main() {
    LL n;
    cin >> n;
    LL base[sz] = {0,1,2};
    for (int i=0;i<=n;i++) { // print f(i) up
        to n
        LL mat[sz][sz] =
            {{0,1,0},{0,0,1},{3,2,1}};
        power(mat, i);
        LL ans = 0;
        for (int j=0;j<sz;j++) ans +=
            ((mat[0][j]%MOD) * (base[j]%MOD))
            % MOD;
        cout << ans << endl;
    }
    return 0;
}
```

5.6 MillerRabin

```
struct Miller{
  const vector<long long> v = { 2 , 7 , 61 };
      // < 4,759,123,141
  // x^k (mod m)
  long long modpow( long long x, long long k,
      long long m ){
    long long res = 1;
    while( k ){
      if( k & 1 ){
        res = res * x % m;
      }
      k >>= 1;
      x = x * x % m;
    }
    return res;
  }
  // check if n is prime
  bool check( long long n ){
    if( n < 2 ){
      return false;
    }
    long long d = n - 1;
    long long s = 0;
    while( d % 2 == 0 ){
      d >>= 1;
      s++;
    }
    for( long long a : v ){
      if( a == n ){
        return true;
      }
      if( modpow( a , d , n ) != 1 ){
        bool ok = true;
        for( long long r = 0; r < s; r++ ){
          if( modpow( a, d * (1LL << r), n )
              == n-1 ){
            ok = false;
            break;
          }
        }
        if( ok ){
          return false;
        }
      }
    }
    return true;
  }
};

Miller miller;

int main () {
    int x;
    cin >> x;
    cout << miller.check(x);
}
```

5.7 PrimeFactor

```
/**
* Description : some function that have
    relation with prime factor
*/

/* find prime factor */
vector<long long> primefactor(long long N){
    vector<long long> factors;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
       while (N%PF == 0){
           N /= PF;
           factors.push_back(PF);
       }
```

```
        PF = primes[++idx];
    }
    if (N != 1) factors.push_back(N);
    return factors;
}

/* number of divisor */
long long numDiv(long long N){
    long long ans = 1;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
        long long power = 0;
        while (N%PF == 0){
            power++;
            N /= PF;
        }
        ans *= (power + 1);
        PF = primes[++idx];
    }
    if (N != 1) ans *= 2;
    return ans;
}

/* sum of divisor */
long long sumDiv(long long N){
    long long ans = 1;
    long long idx = 0;
    long long PF = primes[idx];
    while (PF <= (long long)sqrt(N)){
        long long power = 0;
        while (N%PF == 0){
            power++;
            N /= PF;
        }
        /* 1 + PF + PF^2 + PF^3 + ... + PF^pow
            = (a.r^n - 1) / (r-1) */
        ans *= ((long long)pow((double)PF,
            power + 1.0) - 1) / (PF - 1);
        PF = primes[++idx];
    }
    if (N != 1) ans *= ((long
        long)pow((double)N, 2.0) - 1) / (N -
        1);
    return ans;
}

/* Euler Phi */
long long eulerPhi(long long N){
    long long idx = 0;
    long long PF = primes[idx];
    long long ans = N;
    while (PF <= (long long)sqrt(N)){
        if (N%PF == 0) ans -= ans / PF;
        while (N%PF == 0) N /= PF;
        PF = primes[++idx];
    }
    if (N != 1) ans -= ans / N;
    return ans;
}
```

6 Others

6.1 others

```
// random
mt19937
    rng(chrono::steady_clock::now().time_since_epoch().count());
/* usage: rng() */

// fast io
ios_base::sync_with_stdio(false);
cin.tie(NULL);
cout.tie(NULL);

// magic
#pragma GCC optimize ("O3")

// baca file
ifstream fin("input.txt")
ofstream fout("output.txt")

// fast scan number
void fastscan(int &number) {
    //variable to indicate sign of input number
    bool negative = false;
    register int c;
    number = 0;
    c = getchar();
    if (c=='-') {
        negative = true;
        c = getchar();
    }
    for (; (c>47 && c<58); c=getchar())
        number = number *10 + c - 48;
    if (negative)
        number *= -1;
}
```

7 String

7.1 Hashing

```
long long compute_hash(string const& s) {
    const int p = 31; //another good option :
        p = 53
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' +
            1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

7.2 KMP

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

void computeLPSArray(char *pat, int M, int
    *lps);

void KMPSearch(char *pat, char *txt,
    vector<int> &ans)
```

```
{
        int M = strlen(pat);
        int N = strlen(txt);

    // create lps[] that will hold the longest
        prefix suffix values for pattern
        int *lps = (int
            *)malloc(sizeof(int)*M);
    int j = 0;  // index for pat[]

    // Preprocess the pattern (calculate lps[]
        array)
    computeLPSArray(pat, M, lps);

    int i = 0; // index for txt[]
    while (i < N)
    {
        if (pat[j] == txt[i])
        {
                j++;
                i++;
        }

        if (j == M)
        {
                ans.push_back(i-j);
        // printf("Found pattern at index %d
            \n", i-j);
                j = lps[j-1];
        }

      // mismatch after j matches
        else if (i < N && pat[j] != txt[i])
        {
        // Do not match lps[0..lps[j-1]]
            characters,
        // they will match anyway
                if (j != 0)
                        j = lps[j-1];
                else
                        i = i+1;
        }
    }
    free(lps); // to avoid memory leak
}

void computeLPSArray(char *pat, int M, int
    *lps)
{
    int len = 0; // lenght of the previous
        longest prefix suffix
    int i;

    lps[0] = 0; // lps[0] is always 0
    i = 1;

    // the loop calculates lps[i] for i = 1 to
        M-1
    while (i < M)
    {
        if (pat[i] == pat[len])
        {
                len++;
                lps[i] = len;
                i++;
        }
       else // (pat[i] != pat[len])
       {
            if (len != 0) {
                // This is tricky. Consider the
                    example AAACAAAA and i = 7.
                len = lps[len-1];
                // Also, note that we do not
                    increment i here
            } else // if (len == 0)
                {
                    lps[i] = 0;
                    i++;
                }
        }
    }
}

// Driver program to test above function
int main() {
        int t;
        scanf("%i", &t);
        while(t--){
                char str[1000020], pat[1000020];
                scanf("%s %s", str, pat);
                vector<int> ans;
                KMPSearch(pat, str, ans);
                if (ans.size() == 0){
                        printf("Not Found\n");
                        continue;
                }
                printf("%lu\n", ans.size());
                for(int i = 0; i < ans.size();
                    i++)
                        printf("%i ", ans[i]+1);
                printf("\n");
        }
        return 0;
}
```