Team notebook

Mauricio Cari Leal

May 25, 2024

Contents

1 DP **1**
1.1 1D2DMaxSum-Multiplication . . . 1
1.2 CoinProblem . . . 2
1.3 ConvexHullTrick . . . 2
1.4 DivideConquerDP . . . 3
1.5 Knapsack . . . 3
1.6 LongestIncreasingSubsequence . . . 4

2 Geometry **5**
2.1 2DAlgorithms . . . 5
2.2 3DAlgorithms . . . 8
2.3 Polygons . . . 8
2.4 SegmentIntersection . . . 11
2.5 TrianglesCircles . . . 12

3 Graphs **15**
3.1 BellmanFord . . . 15
3.2 BiconnectedComponents . . . 15
3.3 CentroidDecomposition . . . 16
3.4 DFS . . . 16
3.5 Dijkstra . . . 17
3.6 D´Esopo-Pape . . . 18
3.7 EulerTour . . . 18
3.8 FloydWarshall . . . 18
3.9 HeavyLightDecomposition . . . 19
3.10 Hungarian . . . 20
3.11 LCA-SP . . . 21
3.12 LaplacianMatrix . . . 23
3.13 LinkCutTree . . . 24
3.14 MaxBipartiteMatching . . . 25
3.15 MaxFlowDinic . . . 25
3.16 MaxFlowFordFulkerson . . . 26
3.17 MinCostMaxFlow . . . 27
3.18 SCC . . . 28
3.19 StableMatching . . . 29
3.20 ToposortDFS . . . 30
3.21 ToposortKhan . . . 30

4 Header **31**

5 Math **31**
5.1 ArithmeticEval . . . 31
5.2 CRT . . . 32
5.3 Combinatory . . . 33
5.4 ErathostenesSieve . . . 34
5.5 FFT . . . 34
5.6 Functions . . . 36
5.7 GaussianElimination . . . 38
5.8 MathFunctions . . . 39
5.9 Matrices . . . 40
5.10 PrimeFactorization . . . 41
5.11 Simplex . . . 42

6 Other **43**
6.1 AdHoc . . . 43
6.2 Line input . . . 44
6.3 NextGreaterLower . . . 44

7 String **45**
7.1 AhoCorasick 45
7.2 Hashing 45
7.3 KMP 45
7.4 LongestCommonSubsequence 46
7.5 LongestCommonSubstring 46
7.6 Manacher 47
7.7 PalindromicTree 48
7.8 SuffixArray 49
7.9 Trie 50

1 DP

1.1 1D2DMaxSum-Multiplication

```
#include "../Header.cpp"

int main()
{
    // 1D Max Array Sum
    int n = 9, A[] = { 4, -5, 4, -3, 4, 4, -4, 4, -5 }; //Allow all negative
         numbers
    int sum = A[0], ans = A[0];
    for (int i = 1; i < n; i++)
    {
        sum = max(A[i] + sum, A[i]); // Ignores sum if prev sum is worse than A[i]
        ans = max(ans, sum);
    }
    cout << ans << "\n";

    // 2D Max Array Sum
    int B [100][100];
    ans = -INF;
    cin>>n;
    for (int i = 0; i < n; i++) for (int j = 0; j < n; j++)
    {
        cin >> B[i][j];
        if (j > 0) B[i][j] += B[i][j-1]; // Acum sum per Row
    }
    for (int l = 0; l < n; l++) for (int r = l; r < n; r++)
    {
        sum = B[0][r];
        int SubAns = B[0][r];
        if (l > 0){ sum -= B[0][l-1]; SubAns -= B[0][l-1]; }
        for (int row = 1; row < n; row++)
        {
            int aux = B[row][r];
            if(l > 0) aux -= B[row][l-1];
            sum = max(sum + aux, aux);
            SubAns = max (SubAns, sum);
        }
        ans = max(ans, SubAns);
    }
    cout << ans << "\n";

    // Max Array Multiplication
    vl c;
    bool o = 0;
    ans = 1;
    ll miniend = 1, maxiend = 1;
    for (int i = 0; i < c.size(); i++)
    {
        if(c[i] > 0)
        {
            o = 1;
            if(miniend < 0)miniend *= c[i];
            maxiend *= c[i];
        }
        else if(c[i] == 0)
        {
            miniend = 1;
            maxiend = 1;
        }
        else
        {
            int aux = maxiend;
            maxiend = max(1LL, miniend * c[i]);
            miniend = aux*c[i];
        }
        ans = max(ans, maxiend);
    }
    if(ans == 1 && !o) cout << "0\n";
    else
    {
        cout << ans << "\n";
    }


    // n dimension acumulative sum
    for dim = 0 to 4
        for a = 0 to na-1
            for b = 0 to nb-1
                for c = 0 to nc - 1
                    for d = 0 to nd - 1
                        pa = a - (dim==0); pb = b - (dim==1); pc = c - (dim==2);
                        pd = d - (dim==3);
                        if (pa >= 0 && pb >= 0 && pc >= 0 && pd >= 0)
                            dp(a, b, c, d) += dp (pa, pb, pc, pd)
    // o por cada celda
    for x in S //(celda de menor a mayor tal que todas las anteriores estan
         procesadas)


}
```

1.2 CoinProblem

```cpp
#include "../Header.cpp"

// Number of ways of reaching a quantity n from a set of coins c

int main()
{

	int c[5] = {1, 5, 10, 25, 50};
	int n;
	while(cin >> n)
	{
		int m[n+1];
		m[0] = 1;
		for(int i = 1; i <= n+1; i++) m[i] = 0;
		for(int j = 0; j < 5; j++)
		{
			for(int i = 1; i <= n+1; i++)
			{
				if(i - c[j] >= 0)
				{
					m[i] += m[i - c[j]];
					//m[i]=min(m[i],m[i-c[j]]+1); for minimum
					    coins
				}
			}

		}
		cout << m[n] << "\n";
	}

	return 0;
}
```

1.3 ConvexHullTrick

```cpp
#include "../Header.cpp"

//https://github.com/mhunicken/icpc-team-notebook-el-vasito/blob/master/data_structures/convexhull_trick.cpp

typedef ll tc;
struct Line{tc m,h;};
struct CHT { // for minimum (for maximum just change the sign of lines)
	vector<Line> c;
	int pos=0;
	tc in(Line a, Line b){
		tc x=b.h-a.h,y=a.m-b.m;
		return x/y+(x%y?!((x>0)^(y>0)):0); // ==ceil(x/y)
	}
	void add(tc m, tc h){ // m's should be non increasing
		Line l=(Line){m,h};
		if(c.size()&&m==c.back().m){
			l.h=min(h,c.back().h);c.pop_back();if(pos)pos--;
		}
		while(c.size()>1&&in(c.back(),l)<=in(c[c.size()-2],c.back())){
			c.pop_back();if(pos)pos--;
		}
		c.pb(l);
	}
	inline bool fbin(tc x, int m){return in(c[m],c[m+1])>x;}
	tc eval(tc x){
		// O(log n) query:
		int s=0,e=c.size();
		while(e-s>1){int m=(s+e)/2;
			if(fbin(x,m-1))e=m;
			else s=m;
		}
		return c[s].m*x+c[s].h;
		// O(1) query (for ordered x's):
		while(pos>0&&fbin(x,pos-1))pos--;
		while(pos<c.size()-1&&!fbin(x,pos))pos++;
		return c[pos].m*x+c[pos].h;
	}
};


struct Line {
    ll m, c, id;
    ll calc(ll x) {

        return m * x + c;
    }
};
bool obsolete(Line a, Line b, Line c){
    return (c.c - a.c) * (a.m - b.m) < (a.m - c.m) * (b.c - a.c);
}

vector<Line>lines;
void insert(Line l) {
    while(lines.size() > 1) {
        ll sz = lines.size();
        if(obsolete(lines[sz-2], lines[sz-1], l)){
            lines.pop_back();
        } else break;
    }
    lines.push_back(l);
}
```

1.4 DivideConquerDP

```cpp
#include "../Header.cpp"

// dp(i, j) = min dp(i-1,k-1) + C(k,j) for all k in [0, j]
```

```
// C(a,c) + C(b, d) <= C(a,d) + C(b,c) for all a <= b <= c <= d

vp c;
vl acum1, acum2;

ll cost(ll i, ll j)
{
  return c[j].first * (acum1[j+1] - acum1[i]) - (acum2[j+1] - acum2[i]);
}

vector<ll> last, now;

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

    int mid = (l + r) / 2;
    pair<ll, int> best = {cost(0, mid), -1};

    for(int k = max(1, optl); k < min(mid, optr) + 1; k++)
        best = min(best, {last[k - 1] + cost(k, mid), k});

    now[mid] = best.first;

    compute(l, mid - 1, optl, best.second);
    compute(mid + 1, r, best.second, optr);
}

int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0);


  ll n, k, x, w;
  while(cin >> n >> k){

    c.clear();
    for(int i = 0; i < n; i++){

      cin >> x >> w;
      c.push_back({x, w});
    }

    acum1.clear();
    acum2.clear();
    acum1.push_back(0);
    acum2.push_back(0);

    for(int i = 0; i < n; i++){
      acum1.push_back(c[i].second);
      acum2.push_back(c[i].first * c[i].second);
      acum1.back() += acum1[i];
      acum2.back() += acum2[i];
    }

    last.assign(n, INF);
    now.resize(n);

    for(int i = 0; i < k; i++) { compute(0, n - 1, 0, n - 1); swap(last, now); }

    cout<< last[n-1]<<"\n";

  }
```

1.5 Knapsack

```
#include "../Header.cpp"

int V[10000], W[10000], M[102][10202];

// index, capacity
int DP(int i, int c)
{
    if(i==-1)
    {
        return 0;
    }
    if(c==0)return 0;
        if(M[i][c] != -1) return M[i][c];

        M[i][c] = DP(i-1, c);
        if(W[i] <= c){
        M[i][c] = max(M[i][c], DP(i-1, c - W[i]) + V[i]);
        }
        return M[i][c];
}

// Variation

int usados=0,espacio_usado;
int knapSack(int W, int wt[], int val[], int n)
{
   int i, w;
   int K[n+1][W+1][3];


   for (i = 0; i <= n; i++)
   {
       for (w = 0; w <= W; w++)
       {
           if (i==0 || w==0){
               K[i][w][0]= 0;
               K[i][w][1]= 0;
               K[i][w][2]= 0;
           }
           else if (wt[i-1] <= w){
```

```
                K[i][w][0] = max(val[i-1] + K[i-1][w-wt[i-1]][0],K[i-1][w][0]);
                if(K[i-1][w][0]>val[i-1] + K[i-1][w-wt[i-1]][0])
                {
                    K[i][w][1]=K[i-1][w][1];
                    K[i][w][2]=K[i-1][w][2];
                }
                else{
                   K[i][w][1]=K[i-1][w-wt[i-1]][1]+wt[i-1];
                   K[i][w][2]=K[i-1][w-wt[i-1]][2]+1;
                }
        }
            else
            {
                K[i][w][0] = K[i-1][w][0];
                K[i][w][1] = K[i-1][w][1];
                K[i][w][2] = K[i-1][w][2];
            }
        }
    }
     usados=K[n][W][2];
     espacio_usado=K[n][W][1];
     return K[n][W][0];
}

int main()
{
    int v,W,t;
    cin>>t;
    for(int o=0;o<t;o++)
    {
    W=50;
    usados=0;
    cin>>v;
    int val[v];
    int wt[v];
    for(int i=0;i<v;i++)
    {
        cin>>val[i];
        cin>>wt[i];
    }
    int n = sizeof(val)/sizeof(val[0]);
    cout<<knapSack(W, wt, val, n)<<" brinquedos"<<endl;
    cout<<"Peso: "<<espacio_usado<<" kg"<<endl;
    cout<<"sobra(m) "<<v-usados<<" pacote(s)"<<endl<<endl;
    }
    return 0;
}
```

1.6 LongestIncreasingSubsequence

```
#include "../Header.cpp"

vl A, p;
void print_LIS(int i) {                     // backtracking routine
  if (p[i] == -1) { printf("%d", A[i]); return; }// base case
  print_LIS(p[i]);                          // backtrack
  printf(" %d", A[i]);
}

//O(nlogn)
int lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        int j = upper_bound(d.begin(), d.end(), a[i]) - d.begin();
        if (d[j-1] < a[i] && a[i] < d[j])
            d[j] = a[i];
    }

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

int main()
{
    ll t,n;
    cin>>t;
    while(t--)
    {
        int x;
        cin>>n;
        for(int i=0;i<n;i++)
        {
            cin>>x;
            A.push_back(x);
        }


        ll LIS[100][100] // LIS for any (i, j)
        for(int z = 0; z < n; z++){
            int k = z, lis_end = z;
            vl L(n, 0), L_id(n, 0);
            p.assign (n, -1)

            for (int i = z; i < n; ++i) {
                int pos = lower_bound(L.begin() + z, L.begin()+k, c[i]) - L.begin();
                if(A[i]==L[pos])pos++;//For non strickly increasing subsequence
                L[pos] = c[i];
                L_id[pos] = i;
                p[i] = pos ? L_id[pos-1] : -1;
```

```
            if (pos == k) {
            k = pos+1;
            lis_end = i;
            }
        }
        for(int i = z; i < n; i++)
        {
            if(p[i] == -1) LIS[z][i] = 1;
            else LIS[z][i] = 1 + LIS[z][p[i]];
        }
    }
    cout<<"Final LIS is of length: "<< k<<"\n";
    print_LIS(lis_end);cout<<"\n";



    //DP
    vl LI(n, 0), LD(n,0);
    ll in=0,dec=0;
    for(int i=0;i<n;i++)
    {
        LI[i]=1;
        LD[i]=1;
        for(int j=0;j<i;j++)
        {
            if(A[j]<A[i])
                LI[i]=max(LI[i],LI[j]+1);
            if(A[j]>A[i])
                LD[i]=max(LD[i],LD[j]+1);
        }
        in=max(in,LI[i]);
        dec=max(dec,LD[i]);

    }
}

    return 0;
}
```

2 Geometry

2.1 2DAlgorithms

```
#include "../Header.cpp"

double DEG_to_RAD(double d) { return d*PI / 180.0; }
double RAD_to_DEG(double r) { return r*180.0 / PI; }

struct point { db x, y;
    point() { x = y = 0.0; }
    point(db _x, db _y) : x(_x), y(_y) {}
    bool operator <(const point& p) const { return (x < p.x ? true : (x == p.x &&
        y < p.y)); }
    bool operator == (const point& p) const { return abs(p.x - x) < EPS &&
        abs(p.y - y) < EPS; }
    point operator + (const point& p) const { return point(x + p.x, y + p.y); }
    point operator - (const point& p) const { return point(x - p.x, y - p.y); }
    point operator * (db p) const { return point(x * p, y * p); }
    point operator / (db p) const { return point(x / p, y / p); }
    db operator^(const point &p) const {return x * p.y - y * p.x; }
    db operator*(const point &p) const {return x * p.x + y * p.y; }
    db norm_sq() const{ return x*x + y*y; }
    point rot(){ return point(-y, x); }
    point rot45(){ return point(x + y, y - x); }

    // by angles but with cross
    bool half() const { return y > 0 || (y == 0 && x > 0); }
    bool operator<(const point &p) const
    {
        int h1 = half(), h2 = p.half();
        return h1 != h2 ? h1 > h2 : ((*this) ^ p) > 0;
    }

    db ang()
    {
        double a = atan2(y, x);
        if (a < 0) a += 2.0 * PI;
        return a;
    }

};

db dist(const point& p1,const point& p2) {
  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+ (p1.y-p2.y)*(p1.y-p2.y)); }
db dist_sq(point p1, point p2) {
  return (p1.x - p2.x)*(p1.x - p2.x)+(p1.y - p2.y)*(p1.y - p2.y); }


point rotate(point p, db rad) {
  return point(p.x * cos(rad) - p.y*sin(rad),
               p.x * sin(rad) + p.y*cos(rad)); }


struct line { db a, b, c; };


void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x-p2.x) < EPS)              // vertical line is fine
    l = {1.0, 0.0, -p1.x};                      // default values
  else {
    db a = -(db)(p1.y-p2.y) / (p1.x-p2.x);
    l = {a,
         1.0,             // IMPORTANT: we fix the value of b to 1.0
         -(db)(a*p1.x) - p1.y}; }
  }
```

```
// for integers, normalized
void pointsToLine(point& p1, point p2, line &l) {
  l.a = p1.y - p2.y;
  l.b = p2.x - p1.x;
  l.c = p1.x * (p2.y - p1.y) - p1.y * (p2.x - p1.x);
  ll g = __gcd(abs(l.a), __gcd(abs(l.b), abs(l.c)));
  ll sgn = 1;
  if(l.a < 0 || (l.a == 0 && l.b < 0))sgn = -1;
  l.a /= g * sgn; l.b /= g * sgn; l.c /= g * sgn;
}
// not needed since we will use the more robust form: ax + by + c = 0
struct line2 { db m, c; };   // another way to represent a line

int pointsToLine2(point p1, point p2, line2 &l) {
 if (abs(p1.x-p2.x) < EPS) {        // special case: vertical line
   l.m = INF;                  // l contains m = INF and c = x_value
   l.c = p1.x;                // to denote vertical line x = x_value
   return 0;  // we need this return variable to differentiate result
 }
 else {
   l.m = (db)(p1.y-p2.y) / (p1.x-p2.x);
   l.c = p1.y - l.m*p1.x;
   return 1;    // l contains m and c of the line equation y = mx + c
} }

bool areParallel(line l1, line l2) {   // check coefficients a & b
  return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS); }

bool areSame(line l1, line l2) {       // also check coefficient c
  return areParallel(l1 ,l2) && (fabs(l1.c-l2.c) < EPS); }

// returns true (+ intersection point) if two lines are intersect
bool areIntersect(line l1, line l2, point &p) {
  if (areParallel(l1, l2)) return false;       // no intersection
  // solve system of 2 linear algebraic equations with 2 unknowns
  p.x = (l2.b*l1.c - l1.b*l2.c) / (l2.a*l1.b - l1.a*l2.b);
  // special case: test for vertical line to avoid division by zero
  if (fabs(l1.b) > EPS) p.y = -(l1.a*p.x + l1.c);
  else                  p.y = -(l2.a*p.x + l2.c);
  return true; }

// Or use pointsToSlope, Revisar, mejor con 2 puntos
void perpendicular_line(point a, line l, line& ans)
{
    point b((-l.b*a.y-l.c)/l.a,a.y+1);
    b.x-=a.x;
    b.y-=a.y;
    b = rotate(b,90);
    b.x+=a.x;
    b.y+=a.y;
    pointsToLine(a,b, ans);
}


//Scalar projection of vector a onto vector b
// if s < -EPS or s > |b| + EPS then the projection is not on the segment
db sproject(point a, point b)
{
  return a*b/sqrt(b.norm_sq());
}

bool onSegment(const point& p, const point& p1, const point& p2)
{
    bool x = (abs(p1.x - p2.x) < EPS && abs(p.x - p2.x) < EPS) || (p.x <=
         max(p1.x, p2.x) && p.x >= min(p1.x, p2.x));
    bool y = (abs(p1.y - p2.y) < EPS && abs(p.y - p2.y) < EPS) || (p.y <=
         max(p1.y, p2.y) && p.y >= min(p1.y, p2.y));
    return x && y;
}

// convert point and gradient/slope to line, A PARTIR DE UNA DIRECCION M
// usar 1/l.a para calcular perpendicular
void pointSlopeToLine(point p, db m, line &l) {
  l.a = -m;                                          // always -m
  l.b = 1;                                            // always 1
  l.c = -((l.a*p.x) + (l.b*p.y)); }                // compute this

void closestPoint(line l, point p, point &ans) {
  line perpendicular;      // perpendicular to l and pass through p
  if (fabs(l.b) < EPS) {           // special case 1: vertical line
    ans.x = -(l.c); ans.y = p.y;    return; }

  if (fabs(l.a) < EPS) {         // special case 2: horizontal line
    ans.x = p.x;    ans.y = -(l.c); return; }

  pointSlopeToLine(p, 1/l.a, perpendicular);       // normal line
  // intersect line l with this perpendicular line
  // the intersection point is the closest point
  areIntersect(l, perpendicular, ans); }

// returns the reflection of point on a line
void reflectionPoint(line l, point p, point &ans) {
  point b;
  closestPoint(l, p, b);                 // similar to distToLine
  point v = (b - p);                        // create a vector
  ans = p + v + v; }      // translate p twice

// returns the distance from p to the line defined by
// two points a and b (a and b must be different)
// the closest point is stored in the 4th parameter (byref)
db distToLine(point p, point a, point b, point &c) {
  // formula: c = a + u*ab
  point ap = (p - a), ab = (b - a);
  db u = ap * ab / ab.norm_sq();
  c = a + ab * u;               // translate a to c
  return dist(p, c); }        // Euclidean distance between p and c

// returns the distance from p to the line segment ab defined by
// two points a and b (still OK if a == b)
// the closest point is stored in the 4th parameter (byref)
```

```
db distToLineSegment(point p, point a, point b, point &c) {
  point ap = (p - a), ab = (b - a);
  db u = ap * ab / ab.norm_sq();
  if (u < 0.0) { c = point(a.x, a.y);             // closer to a
    return dist(p, a); }      // Euclidean distance between p and a
  if (u > 1.0) { c = point(b.x, b.y);             // closer to b
    return dist(p, b); }      // Euclidean distance between p and b
  return distToLine(p, a, b, c); }      // run distToLine as above

bool ccw(point p, point q, point r) {
  return ((q - p)^(r - p)) > -EPS; }

// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
  return fabs(((q - p)^(r - p))) < EPS; }


// angle from 0 to 2*PI
db anglet(point a, point o, point b) { // returns angle aob in rad
  point oa = (a - o), ob = (b - o);
  db ang = acos(oa * ob / sqrt(oa.norm_sq()*ob.norm_sq()));
  if(ang!=0&&!collinear(a,o,b)&&ccw(a,o,b))ang = 2*PI - ang;
  return ang; } // better

db angle(point a, point o, point b) { // returns angle aob in rad
  point oa = (a - o), ob = (b - o);
  return acos(oa * ob / sqrt(oa.norm_sq()*ob.norm_sq())); }

point min(point a,point b)
{
    if(a<b)return a;
    return b;
}
point max(point a, point b)
{
    if(!(a<b))return a;
    return b;
}

// 0 -> No intersection, 1 -> Point intersection, 2 -> segment intersection
int SegmentIntersection(point a1, point a2, point b1, point b2, point& ans,
     point& ans2)
{
    line A,B;
    point I;
    pointsToLine(a1,a2,A);
    pointsToLine(b1,b2,B);
    if(areSame(A,B)&&!(a1==a2)&&!(b1==b2))
    {
        ans=max(min(a1,a2),min(b1,b2));
        ans2=min(max(a1,a2),max(b1,b2));
        if(ans2<ans)return 0;
        else if(ans == ans2)return 1;
        return 2;
    }
    if (a1==a2&&b1==b2)
    {
        if(a1==b1)
        {
            ans=a1;
            return 1;
        }
        return 0;
    }
    if(a1==a2)
    {
        if(fabs(distToLineSegment(a1, b1, b2, ans)-0.0) < EPS)
        {
            ans=a1;
            return 1;
        }
        return 0;
    }
    if(b1==b2)
    {
        if(fabs(distToLineSegment(b1, a1, a2, ans)-0.0) < EPS)
        {
            ans=b1;
            return 1;
        }
        return 0;
    }
    if (areIntersect(A,B,I) && fabs(distToLineSegment(I, a1, a2, ans)-0.0) < EPS
         && fabs(distToLineSegment(I, b1, b2, ans)-0 < EPS))
    {
        return 1;
    }
    return 0;
}
```

2.2 3DAlgorithms

```
#include "../Header.cpp"


struct point { db x, y, z;
    point() { x = y = z = 0.0; }
    //point(db r, db u, db v) : x(r*cos(u)*cos(v)), y(r*cos(u)*sin(v)),
         z(r*sin(u)) {}
        point(db _x, db _y, db _z) : x(_x), y(_y), z(_z) {}
        point operator^(const point &p) const {
               return { y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y * p.x};
        }
        db dot(point& p) { return x*p.x + y*p.y + z*p.z; }
    db norm() { return sqrt(x*x + y*y + z*z); }

        bool operator == (const point& p) const
```

```
    {
        return abs(p.x - x) < EPS && abs(p.y - y) < EPS && abs(p.z - z) < EPS;
    }
    point operator + (const point& p) const
    {
        return point(x + p.x, y + p.y, z + p.z);
    }
    point operator - (const point& p) const
    {
        return point(x - p.x, y - p.y, z - p.z);
    }
    point operator * (db a) const
    {
        return point(x * a, y * a, z * a);
    }
    point operator / (db a) const
    {
        return point(x / a, y / a, z / a);
    }
        point unit() {
        db d = norm();
        return {x/d,y/d,z/d};
    }

};
db angle2(point& x, point& y)
{
        return acos(x.dot(y) / (R*R));
}

bool in_arc(point& p1, point& p2, point& n, point& inter)
{
        db ab = angle2(p1, p2);
        db ap = angle2(p1, inter);
        point d = (p1 * cos(ap) + (n ^ p1) * sin(ap));
        return ab > ap && inter == d;
}
bool do_intersect_circles()
{
    point a1 = g[j][z], a2 = g[j][0];
    if(z < g[j].size() - 1)
        a2 = g[j][z+1];
    point p1 = route[i], p2 = route[i+1];
    point n1 = (p1^p2).unit(), n2 = (a1^a2).unit();
    point inter = n1^n2;
    if(inter.norm() < EPS)continue;
    inter = inter.unit() * R;

    if(in_arc(p1, p2, n1, inter) && in_arc(a1, a2, n2, inter))
    {
        ag.push_back(angle2(p1, inter));
        continue;
    }
    inter = inter * -1.0;
    if(in_arc(p1, p2, n1, inter) && in_arc(a1, a2, n2, inter))
    {
        ag.push_back(angle2(p1, inter));
        continue;
    }
}
```

2.3 Polygons

```
#include "../Header.cpp"

db DEG_to_RAD(db d) { return d*PI / 180.0; }

db RAD_to_DEG(db r) { return r*180.0 / PI; }

struct point { db x, y;
    point() { x = y = 0.0; }
    point(db _x, db _y) : x(_x), y(_y) {}
    bool operator <(const point& p) const { return (x < p.x ? true : (x == p.x &&
         y < p.y)); }
    bool operator == (const point& p) const { return abs(p.x - x) < EPS &&
         abs(p.y - y) < EPS; }
    point operator + (const point& p) const { return point(x + p.x, y + p.y); }
    point operator - (const point& p) const { return point(x - p.x, y - p.y); }
    point operator * (db p) const { return point(x * p, y * p); }
    point operator / (db p) const { return point(x / p, y / p); }
    db operator^(const point &p) const {return x * p.y - y * p.x; }
    db operator*(const point &p) const {return x * p.x + y * p.y; }
    db norm_sq() const{ return x*x + y*y; }
    point rot(){ return point(-y, x); }

    // by angles but with cross
    bool half() const { return y > 0 || (y == 0 && x > 0); }
    bool operator<(const point &p) const
    {
        int h1 = half(), h2 = p.half();
        return h1 != h2 ? h1 > h2 : ((*this) ^ p) > 0;
    }

    db ang()
    {
        double a = atan2(y, x);
        if (a < 0) a += 2.0 * PI;
        return a;
    }

  };


db dist(point& p1, point& p2) {
  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+ (p1.y-p2.y)*(p1.y-p2.y)); }
db dist_sq(point p1, point p2) {
  return (p1.x - p2.x)*(p1.x - p2.x)+(p1.y - p2.y)*(p1.y - p2.y); }
```

```
// returns the perimeter, which is the sum of Euclidian distances
// of consecutive line segments (polygon edges)
db perimeter(vector<point> &P) {
  db result = 0.0;
  for (ll i = 0; i < (ll)P.size()-1; i++) // remember that P[0] = P[n-1]
    result += dist(P[i], P[i+1]);
  return result; }

// returns the area
db area(const vector<point> &P) {
  db result = 0.0;
  for (ll i = 0; i < (ll)P.size()-1; i++)        // Shoelace formula
    result += P[i]^P[i+1]; // if all points are ll
  return fabs(result)/2.0; }  // result can be ll(eger) until last step

db seg_integrate(point& a, point& b, db t1, db t2)
{
  // area
  point p1 = a + (b-a) * t1;
  point p2 = a + (b-a) * t2;
  return (p1^p2) / 2.0;
}

db param(point p1, point p2, point a)
{
  if(p1.x != p2.x)
  {
    db sgn = 1;
    if(p1.x > p2.x)sgn = -1;
    return (a.x - p1.x) / abs(p2.x - p1.x) * sgn;
  }
  db sgn = 1;
  if(p1.y > p2.y)sgn = -1;
  return (a.y - p1.y) / abs(p2.y - p1.y) * sgn;
}

// note: to accept collinear points, we have to change the '> 0'
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
  return ((q - p)^(r - p)) > 0; }

int orientation(point p, point q, point r) {
    ll tmp = ((q - p)^(r - p));
    return tmp < 0 ? -1 : tmp == 0 ? 0 : 1; // sign
}
/*bool do_rectangles_intersect(point dl1, point ur1, point dl2, point ur2) {
    return max(dl1.x, dl2.x) <= min(ur1.x, ur2.x) && max(dl1.y, dl2.y) <=
        min(ur1.y, ur2.y);
}*/

bool do_segments_intersect(point p1, point q1, point p2, point q2) {
    int o11 = orientation(p1, q1, p2);
    int o12 = orientation(p1, q1, q2);
    int o21 = orientation(p2, q2, p1);
    int o22 = orientation(p2, q2, q1);
    // oxx != 0 means cross intersection, no T intersection
    if (o11 != o12 && o21 != o22 && o11 != 0 && o12 != 0 && o21 != 0 && o22 != 0)
        // general case -> non-collinear intersection
       return true;
    return false;
}

// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
  return fabs(((q - p)^(r - p))) < EPS; }

// angle from 0 to 2*PI
db anglet(point a, point o, point b) { // returns angle aob in rad
  point oa = (a - o), ob = (b - o);
  db ang = acos(oa * ob / sqrt(oa.norm_sq()*ob.norm_sq()));
  if(ang!=0&&!collinear(a,o,b)&&ccw(a,o,b))ang = 2*PI - ang;
  return ang; } // better

db angle(point a, point o, point b) { // returns angle aob in rad
  point oa = (a - o), ob = (b - o);
  return acos(oa * ob / sqrt(oa.norm_sq()*ob.norm_sq())); }
// returns true if we always make the same turn while examining
// all the edges of the polygon one by one
bool isConvex(const vector<point> &P) {
  ll sz = (ll)P.size();
  if (sz <= 3) return false; // a point/sz=2 or a line/sz=3 is not convex
  bool firstTurn = ccw(P[0], P[1], P[2]);       // remember one result
  for (ll i = 1; i < sz-1; i++)        // then compare with the others
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != firstTurn)
      return false;         // different sign -> this polygon is concave
  return true; }                              // this polygon is convex

// returns true if point p is in either convex/concave polygon P
bool inPolygon(point pt, const vector<point> &P) {
  if ((ll)P.size() < 3) return false;          // avoid point or line
  db sum = 0;  // assume the first vertex is equal to the last vertex
  for (ll i = 0; i < (ll)P.size()-1; i++) {
    if (((P[i] - pt)^(P[i+1] - pt)) > 0) //CCW check collinear
        sum += angle(P[i], pt, P[i+1]);             // left turn/ccw
    else sum -= angle(P[i], pt, P[i+1]); }          // right turn/cw
  return fabs(sum) > PI; } // 360d -> in, 0d -> out, we have large margin

// line segment p-q intersect with line A-B.
point lineIntersectSeg(point p, point q, point A, point B) {
  db a = B.y - A.y;
  db b = A.x - B.x;
  db c = B.x * A.y - A.x * B.y;
  db u = fabs(a * p.x + b * p.y + c);
  db 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 formed by point a -> point b
// (note: the last point must be the same as the first point)
```

```
// to cut the other side, swap (a,b)
vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
  vector<point> P;
  for (ll i = 0; i < (ll)Q.size(); i++) {
    db left1 = (b - a)^(Q[i] - a), left2 = 0;
    if (i != (ll)Q.size()-1) left2 = (b - a)^(Q[i+1] - a);
    if (left1 > -EPS) P.push_back(Q[i]);   // Q[i] is on the left of ab
    if (left1 * left2 < -EPS)     // edge (Q[i], Q[i+1]) crosses line ab
      P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
  }
  if (!P.empty() && !(P.back() == P.front()))
    P.push_back(P.front());     // make P's first point = P's last point
  return P; }

vector<point> CH_Andrew(vector<point> &Pts) {
  ll n = Pts.size(), k = 0;
  vector<point> H(2*n);
  sort(Pts.begin(), Pts.end());     // sort the points lexicographically
  for (ll i = 0; i < n; i++) {                    // build lower hull
    while (k >= 2 && ccw(H[k-2], H[k-1], Pts[i]) <= 0) k--;
    H[k++] = Pts[i];
  }
  for (ll i = n-2, t = k+1; i >= 0; i--) {        // build upper hull
    while (k >= t && ccw(H[k-2], H[k-1], Pts[i]) <= 0) k--;
    H[k++] = Pts[i];
  }
  H.resize(k);
  return H;
}

point pivot(0, 0);
vector<point> CH_Graham(vector<point> &Pts) {
  vector<point> P(Pts);    // copy all points so that Pts is not affected
  ll i, j, n = (ll)P.size();
  if (n <= 3) {        // corner cases: n=1=point, n=2=line, n=3=triangle
    if (!(P[0] == P[n-1])) P.push_back(P[0]); // safeguard from corner case
    return P; }                                  // the CH is P itself

  // first, find P0 = point with lowest Y and if tie: rightmost X
  ll P0 = 0;
  for (i = 1; i < n; i++)                                    // O(n)
    if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
      P0 = i;
  swap(P[0], P[P0]);                           // swap P[P0] with P[0]

  // second, sort points by angle w.r.t. pivot P0, O(n log n) for this sort
  pivot = P[0];                 // use this global variable as reference
  sort(++P.begin(), P.end(), [](point a, point b) { // we do not sort P[0]
    if (collinear(pivot, a, b))                     // special case
      return dist(pivot, a) < dist(pivot, b); // check which one is closer
    db d1x = a.x-pivot.x, d1y = a.y-pivot.y;
    db d2x = b.x-pivot.x, d2y = b.y-pivot.y;
    return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0; }); // compare 2 angles

  // third, the ccw tests, although complex, it is just O(n)
  vector<point> S;
  S.push_back(P[n-1]); S.push_back(P[0]); S.push_back(P[1]); // initial S
  i = 2;                                       // then, we check the rest
  while (i < n) {   // note: n must be >= 3 for this method to work, O(n)
    j = (ll)S.size()-1;
    if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]); // left turn, accept
    else S.pop_back(); } // or pop the top of S until we have a left turn
  return S; } // return the result, overall O(n log n) due to angle sorting

point center_of_mass(vector <point>& Q)
{
    point ctr(0,0);
    for (ll i=0;i<Q.size()-1;i++)
    {
        ctr = ctr + Q[i];
    }
    ctr.x/=Q.size()-1;
    ctr.y/=Q.size()-1;
    return ctr;
}
// Pick's theorem
// A = i + b/2 -1
// A: Area poligon with integer coords
// i: Interior points, b: points in the segments

// with vector form of integer segment
// (x0,y0) + t(dx,dy)
ll points_in_segment(point a, point b)
{
  ll absx=abs(a.x-b.x),absy=abs(a.y-b.y);
  return __gcd(absx,absy) + 1;

}

ll memo[101][101][101];

// Dp that pass all possible convex poligons
// from the shortest in(in.y < p.y)
// p si counter cw from in, p
ll all_convex(ll in, ll p, ll q, vector<point>& Q)
{
    if(memo[in][p][q] != -1) return memo[in][p][q];
    ll ans = 0;

    for(int i = in + 1; i < Q.size(); i++)
    {
        if(i != p && i != q && ccw(Q[in], Q[q], Q[i]) && ccw(Q[p], Q[q], Q[i]))
        {
            ans += (all_convex(in, q, i, Q) + 1);
        }
    }
    return memo[in][p][q] = ans;
}
bool comp(point& a, point& b)
{
```

```
    return a.y < b.y;
}
//..
vector<point>Q;
sort(ALL(Q), comp);
ll ans = 0;
for(int i = 0; i < n; i++)
{
    for(int p = i + 1;p < n; p++) for(int q = p + 1; q < n; q++)
    {
        if(ccw(Q[i], Q[p], Q[q]))
            ans += all_convex(i, p, q, Q) + 1;
        else
            ans += all_convex(i, q, p, Q) + 1;
        ans %= m;
    }
}
```

2.4 SegmentIntersection

```
#include "../Header.cpp"

struct point { db x, y;
    point() { x = y = 0.0; }
    point(db _x, db _y) : x(_x), y(_y) {}
    bool operator <(const point& p) const { return (x < p.x ? true : (x == p.x &&
         y < p.y)); }
    bool operator == (const point& p) const { return abs(p.x - x) < EPS &&
         abs(p.y - y) < EPS; }
    point operator + (const point& p) const { return point(x + p.x, y + p.y); }
    point operator - (const point& p) const { return point(x - p.x, y - p.y); }
    point operator * (db p) const { return point(x * p, y * p); }
    point operator / (db p) const { return point(x / p, y / p); }
    db operator^(const point &p) const {return x * p.y - y * p.x; }
    db operator*(const point &p) const {return x * p.x + y * p.y; }
    db norm_sq() const{ return x*x + y*y; }
    point rot(){ return point(-y, x); }
    db ang()
    {
        double a = atan2(y, x);
        if (a < 0) a += 2.0 * PI;
        return a;
    }

  };

db dist(const point& p1,const point& p2) {
  return sqrt((p1.x-p2.x)*(p1.x-p2.x)+ (p1.y-p2.y)*(p1.y-p2.y)); }


//Constant values to be returned
constexpr int Colinear = -1, NoIntersect = 0, Intersect = 1;
constexpr int CW = 2, CCW = 3;

int orientation(point& p, point& q, point& r) {
    ll tmp = (q - p)^(r - p);
    return tmp < 0 ? CW : tmp == 0 ? Colinear : CCW; // sign
}

struct segment { point p1, p2;
  segment(point _p1, point _p2) : p1(_p1), p2(_p2) {}
};

//Returns of list of intersection points between segments s1, and s2
//If they do not intersect, the result is an empty vector
//If they intersect at exactly 1 point, the result contains that point
//If they overlap for non-0 distance, the left and right points of that
     intersection
//  are returned
bool onSegment(const point& p, const segment& s)
{
    bool x = (abs(s.p1.x - s.p2.x) < EPS && abs(p.x - s.p2.x) < EPS) || (p.x <=
         max(s.p1.x, s.p2.x) && p.x >= min(s.p1.x, s.p2.x));
    bool y = (abs(s.p1.y - s.p2.y) < EPS && abs(p.y - s.p2.y) < EPS) || (p.y <=
         max(s.p1.y, s.p2.y) && p.y >= min(s.p1.y, s.p2.y));
    return x && y;
}

vector<point> intersect(const segment& s1, const segment& s2)
{

    point a = s1.p1, b = s1.p2, c = s2.p1, d = s2.p2;

    if(orientation(a, b, c) == Colinear && orientation(a, b, d) == Colinear &&
        orientation(c, d, a) == Colinear && orientation(c, d, b) == Colinear)
    {
        point min_s1 = min(a, b), max_s1 = max(a, b);
        point min_s2 = min(c, d), max_s2 = max(c, d);

        if(max_s1 < min_s2 || max_s2 < min_s1) return {};

        point start = max(min_s1, min_s2), end = min(max_s1, max_s2);
        if(start == end)
            return {start};
        else
            return {min(start, end), max(start, end)};
    }

    db a1 = b.y - a.y, a2 = d.y - c.y;
    db b1 = a.x - b.x, b2 = c.x - d.x;
    db c1 = a1*a.x + b1*a.y, c2 = a2*c.x + b2*c.y;
    db det = a1*b2 - a2*b1;
    if(abs(det) > EPS)
    {
        point inter((b2*c1 - b1*c2)/det, (a1*c2 - a2*c1)/det), aux;
        //if(distToLineSegment(inter, s1.p1, s1.p2, aux) <= EPS &&
             distToLineSegment(inter, s2.p1, s2.p2, aux) <= EPS)
```

```
        if(onSegment(inter, s1) && onSegment(inter, s2))
            return {inter};
    }
    return {};
}
```

2.5 TrianglesCircles

```
#include "../Header.cpp"

//#define double long long //Para usar enteros

db DEG_to_RAD(db d) { return d * PI / 180.0; }
db RAD_to_DEG(db r) { return r * 180.0 / PI; }

//sweepline rotating a circle around a point
// how many points are in circle radius r
// alpha = atan2(point - center) +- acos(dist/2r)

struct point { db x, y;
    point() { x = y = 0.0; }
    point(db _x, db _y) : x(_x), y(_y) {}
    bool operator <(const point& p) const { return (x < p.x ? true : (x == p.x &&
         y < p.y)); }
    bool operator == (const point& p) const { return abs(p.x - x) < EPS &&
         abs(p.y - y) < EPS; }
    point operator + (const point& p) const { return point(x + p.x, y + p.y); }
    point operator - (const point& p) const { return point(x - p.x, y - p.y); }
    point operator * (db p) const { return point(x * p, y * p); }
    point operator / (db p) const { return point(x / p, y / p); }
    db operator^(const point &p) const {return x * p.y - y * p.x; }
    db operator*(const point &p) const {return x * p.x + y * p.y; }
    db norm_sq() const{ return x*x + y*y; }
    point rot(){ return point(-y, x); }

    // by angles but with cross
    bool half() const { return y > 0 || (y == 0 && x > 0); }
    bool operator<(const point &p) const
    {
        int h1 = half(), h2 = p.half();
        return h1 != h2 ? h1 > h2 : ((*this) ^ p) > 0;
    }

    db ang()
    {
        double a = atan2(y, x);
        if (a < 0) a += 2.0 * PI;
        return a;
    }

};
```

```
ll insideCircle(point p, point c, ll r) { // all integer version
    ll dx = p.x - c.x, dy = p.y - c.y;
    ll Euc = dx * dx + dy * dy, rSq = r * r;        // all integer
    return Euc < rSq ? 0 : Euc == rSq ? 1 : 2; } //inside/border/outside

// P1 an d P2 intersections of circles and radius r -> pos of centers of circles
     of intersection
bool circle2PtsRad(point p1, point p2, db r, point &c) {
    db d2 = (p1.x - p2.x) * (p1.x - p2.x) +
            (p1.y - p2.y) * (p1.y - p2.y);
    db det = r * r / d2 - 0.25;
    if (det < 0.0) return false;
    db h = sqrt(det);
    c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
    c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
    return true; }       // to get the other center, reverse p1 and p2

db dist(point& p1, point& p2) {           // Euclidean distance
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+ (p1.y-p2.y)*(p1.y-p2.y)); }
db dist_sq(point p1, point p2) {
    return (p1.x - p2.x)*(p1.x - p2.x)+(p1.y - p2.y)*(p1.y - p2.y); }


// a = max x, b = max y from the center, AREA
db A_elipse(db a,db b)
{
    return a*b*PI;
}

// Length of segment with two points on the circunference
// separated by an angle
db chord(db r, db angle)
{
    return sqrt(2*r*r*(1-cos(angle)));
}

//Triangles
db perimeter(db ab, db bc, db ca) {
    return ab + bc + ca; }

db perimeter(point a, point b, point c) {
    return dist(a, b) + dist(b, c) + dist(c, a); }

db area(db ab, db bc, db ca) {
  // Heron's formula, split sqrt(a * b) into sqrt(a) * sqrt(b); in implementation
    db s = 0.5 * perimeter(ab + bc + ca);
    return sqrt(s) * sqrt(s - ab) * sqrt(s - bc) * sqrt(s - ca); }

db area(point a, point b, point c) {
    return area(dist(a, b), dist(b, c), dist(c, a)); }


// Area of the circle enclosed by an arc and a chord defined by an angle
db segment(db r, db angle)
{
```

```
    return angle/2.0*r*r-area(chord(r,angle),r,r);
}

// And overlaping rectangle area > 0
bool rectangles_intersect(point a1,point a2,point b1,point b2,point& ans1, point&
     ans2)
{
    if(b1<a1)
    {
        swap(a1,b1);
        swap(a2,b2);
    }
    if(b1.x>=a2.x||b1.y>=a2.y||b2.y<=a1.y)return 0;
    ans1.x=b1.x;
    ans1.y=max(b1.y,a1.y);
    ans2.x=min(b2.x,a2.x);
    ans2.y=min(b2.y,a2.y);
    return 1;

}

struct line { db a, b, c; };

void pointsToLine(point p1, point p2, line &l) {
    if (fabs(p1.x - p2.x) < EPS) {         // vertical line is fine
        l.a = 1.0;  l.b = 0.0;  l.c = -p1.x;        // default values
    } else {
        l.a = -(db)(p1.y - p2.y) / (p1.x - p2.x);
        l.b = 1.0;              // IMPORTANT: we fix the value of b to 1.0
        l.c = -(db)(l.a * p1.x) - p1.y;
} }

bool areParallel(line l1, line l2) {    // check coefficient a + b
    return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS); }

bool areSame(line l1, line l2) {       // also check coefficient c
    return areParallel(l1 ,l2) && (fabs(l1.c-l2.c) < EPS); }

// returns true (+ intersection point) if two lines are intersect
bool areIntersect(line l1, line l2, point &p) {
    if (areParallel(l1, l2)) return false;       // no intersection
    // solve system of 2 linear algebraic equations with 2 unknowns
    p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
    // special case: test for vertical line to avoid division by zero
    if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
    else                  p.y = -(l2.a * p.x + l2.c);
    return true; }

db rInCircle(db ab, db bc, db ca) {
    return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca)); }

db rInCircle(point a, point b, point c) {
    return rInCircle(dist(a, b), dist(b, c), dist(c, a)); }

// assumption: the required points/lines functions have been written
// returns 1 if there is an inCircle center, returns 0 otherwise
// if this function returns 1, ctr will be the inCircle center
// and r is the same as rInCircle
ll inCircle(point p1, point p2, point p3, point &ctr, db &r) {
    r = rInCircle(p1, p2, p3);
    if (fabs(r) < EPS) return 0;              // no inCircle center

    line l1, l2;                  // compute these two angle bisectors
    db ratio = dist(p1, p2) / dist(p1, p3);
    point p = p2 + (p3 - p2) * (ratio / (1 + ratio));
    pointsToLine(p1, p, l1);

    ratio = dist(p2, p1) / dist(p2, p3);
    p = p1 + (p3 - p1) * (ratio / (1 + ratio));
    pointsToLine(p2, p, l2);

    areIntersect(l1, l2, ctr);       // get their intersection point
    return 1; }

db rCircumCircle(db ab, db bc, db ca) {
    return ab * bc * ca / (4.0 * area(ab, bc, ca)); }

db rCircumCircle(point a, point b, point c) {
    return rCircumCircle(dist(a, b), dist(b, c), dist(c, a)); }

// assumption: the required points/lines functions have been written
// returns 1 if there is a circumCenter center, returns 0 otherwise
// if this function returns 1, ctr will be the circumCircle center
// and r is the same as rCircumCircle
ll circumCircle(point p1, point p2, point p3, point &ctr, db &r){
    db a = p2.x - p1.x, b = p2.y - p1.y;
    db c = p3.x - p1.x, d = p3.y - p1.y;
    db e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
    db f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
    db g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
    if (fabs(g) < EPS) return 0;

    ctr.x = (d*e - b*f) / g;
    ctr.y = (a*f - c*e) / g;
    r = dist(p1, ctr); // r = distance from center to 1 of the 3 points
    return 1; }

//
     https://www.nayuki.io/res/smallest-enclosing-circle/computational-geometry-lecture-6.
// O(N) expected time

void smallest_enclosing_circle(vector<point>& pts, point& center, db& r) {
    random_shuffle(pts.begin(), pts.end());
    center = pts[0]; r = 0;
    ll N = pts.size();
    for(ll i=1;i<N;i++) {
        if (dist(pts[i] , center) > r + EPS) {
            center = pts[i];
            r = 0;
            for(ll j=0;j<i;j++) {
```

```
            if (dist(pts[j] , center) > r + EPS) {
                center = (pts[i] + pts[j]) * 0.5;
                r = dist(pts[i] , center);
                for(ll k=0;k<j;k++) {
                    if (dist(pts[k] , center) > r + EPS) {
                        db rr;
                        circumCircle(pts[i], pts[j], pts[k],center,rr);
                        r = dist(pts[k] , center);
                    }
                }
            }
        }
    }
}

// returns true if point d is inside the circumCircle defined by a,b,c
ll inCircumCircle(point a, point b, point c, point d) {
    return (a.x - d.x) * (b.y - d.y) * ((c.x - d.x) * (c.x - d.x) + (c.y - d.y) *
         (c.y - d.y)) +
        (a.y - d.y) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y) * (b.y - d.y)) *
            (c.x - d.x) +
        ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)) * (b.x - d.x) *
            (c.y - d.y) -
        ((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)) * (b.y - d.y) *
            (c.x - d.x) -
        (a.y - d.y) * (b.x - d.x) * ((c.x - d.x) * (c.x - d.x) + (c.y - d.y) *
            (c.y - d.y)) -
        (a.x - d.x) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y) * (b.y - d.y)) *
            (c.y - d.y) > 0 ? 1 : 0;
}

bool canFormTriangle(db a, db b, db c) {
     return (a + b > c) && (a + c > b) && (b + c > a); }


/*
Si un punto tiene un ngulo >= 60, el lado opuesto no es el menor del triangulo
*/

// Function to find the circle on
// which the given three points lie
// better CIRCUMCENTER
tuple<db, db, db> findCircle(db x1, db y1, db x2, db y2, db x3, db y3)
{
    db x12 = x1 - x2;
    db x13 = x1 - x3;

    db y12 = y1 - y2;
    db y13 = y1 - y3;

    db y31 = y3 - y1;
    db y21 = y2 - y1;

    db x31 = x3 - x1;

    db x21 = x2 - x1;

    db sx13 = x1*x1 - x3*x3;
    db sy13 = y1*y1 - y3*y3;

    db sx21 = x2*x2 - x1*x1;
    db sy21 = y2*y2 - y1*y1;

    db f = ((sx13) * (x12)
            + (sy13) * (x12)
            + (sx21) * (x13)
            + (sy21) * (x13))
           / (2 * ((y31) * (x12) - (y21) * (x13)));
    db g = ((sx13) * (y12)
            + (sy13) * (y12)
            + (sx21) * (y13)
            + (sy21) * (y13))
           / (2 * ((x31) * (y12) - (x21) * (y13)));

    db c = -x1*x1 - y1*y1 - 2 * g * x1 - 2 * f * y1;

    // eqn of circle be x^2 + y^2 + 2*g*x + 2*f*y + c = 0
    // where centre is (h = -g, k = -f) and radius r
    // as r^2 = h^2 + k^2 - c
    db h = -g;
    db k = -f;
    db sqr_of_r = h * h + k * k - c;

    // r is the radius
    db r = sqrt(sqr_of_r);

    //cout << "Centre = (" << h << ", " << k << ")" << endl;
    //cout << "Radius = " << r;
    return make_tuple(h, k, r);
}
```

3 Graphs

3.1 BellmanFord

```
#include "../Header.cpp"

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int v,e,x,y,w,r;
    cin >> v >> e >> r;
    pll h;
    vl d(v, INF);
    d[r] = 0;
    vector<vector<pll> > g(v, vector<pll> (0));
```

```
    for(int i = 0 ; i < e; i++)
{
    cin >> x >> y >> w;
    h.first = y;
    h.second = w;
    g[x].push_back(h);
}
/*
inequations solver
v - u <= p
g[u].push_back({v, p});
g[v].push_back({u, -1});
d[s] = 0
for i in v: g[s].push_back({i, 0})
*/

    rep(i, v-1)
{
    bool mod = 0;
    rep(j, v)
        if(d[j] != INF)
            for(auto it : g[j])
            {
                d[it.first] = min(d[it.first], d[j] + it.second);
                mod = 1;
            }
    if(mod == 0)
        break;
}
bool cyc = 0;
rep(j, v)
    for(auto it : g[j])
        if(d[j] < INF && d[it.first] > d[j] + it.second)
            cyc = 1;


// From
    :https://github.com/stevenhalim/cpbook-code/blob/master/ch4/sssp/bellman_ford_moore.cpp
// Faster but doesnt support negative cycles
// SPFA from source S
vl dist(v, INF); dist[s] = 0;              // INF = 1e9 here
queue<int> q; q.push(s);                   // like BFS queue
vl in_queue(v, 0); in_queue[s] = 1;        // unique to SPFA
while (!q.empty()) {
    int u = q.front(); q.pop(); in_queue[u] = 0; // pop from queue
    for (auto it : g[u]) {              // C++17 style
        if (dist[u]+it.first >= dist[it.second]) continue;  // not improving,
             skip
        dist[it.second] = dist[u]+it.first;                 // relax operation
        if (!in_queue[it.second]) {                    // add to the queue
            q.push(it.second);                           // only if v is not
            in_queue[it.second] = 1;                     // already in the queue
        }
    }
    }
}
```

3.2 BiconnectedComponents

```
#include "../Template.cpp"
//https://github.com/BenjaminRubio/CompetitiveProgramming/blob/master/Material/Graphs/Tar
vector<vl> G;
vl D, L;
// p: -1, L: 0, D: -1
void dfs(int u, int p, int d)
{
    D[u] = L[u] = d;
    for(int v : G[u]) if (v != p)
    {
        if (D[v] == -1)
        {
            dfs(v, u, d + 1);
            if (L[v] > D[u]) {} // (u - v) cut edge
            L[u] = min(L[u], L[v]);
        }
        else L[u] = min(L[u], D[v]);
    }
}

int rc = 0; // Articulation Point
stack<pll> S; // BCC
void dfs(int u, int p, int d)
{
    D[u] = L[u] = d;
    for (int v : G[u]) if (v != p)
    {
        if (D[v] == -1)
        {
            S.emplace(u, v); dfs(v, u, d + 1);
            if ((p == -1 && ++rc == 2) || (p != -1 && L[v] >= d)) {} // u is AP
            if (p == -1 or L[v] >= d) while (1) // BCC found
            {
                pll e = S.top(); S.pop();
                if (e == pll(u, v)) break;
            }
            L[u] = min(L[u], L[v]);
        }
        else if (D[v] < d) { S.emplace(u, v); L[u] = min(L[u], D[v]); }
    }
}
```

3.3 CentroidDecomposition

```
#include "../Header.cpp"
```

```
// all tree diameters pass through the centroid

//https://github.com/mhunicken/icpc-team-notebook-el-vasito/blob/master/graphs/centroid.cpp
const int MAXN = 1e5 + 5;
vector<int> g[MAXN];int n;
bool tk[MAXN];
int fat[MAXN]; // father in centroid decomposition
int szt[MAXN]; // size of subtree
int calcsz(int x, int f){
       szt[x]=1;
       for(auto y:g[x])if(y!=f&&!tk[y])szt[x]+=calcsz(y,x);
       return szt[x];
}
void cdfs(int x=0, int f=-1, int sz=-1){ // O(nlogn)
       if(sz<0)sz=calcsz(x,-1);
       for(auto y:g[x])if(!tk[y]&&szt[y]*2>=sz){
              szt[x]=0;cdfs(y,f,sz);return;
       }
       tk[x]=true;fat[x]=f;
       for(auto y:g[x])if(!tk[y])cdfs(y,x);
}
void centroid(){memset(tk,false,sizeof(tk));cdfs();}

int main(){
  ll t; cin >> t;
  for(int T = 1; T <= t; T++) {
    memset(memo, -1, sizeof(memo));
    ll x;
    cin >> N >> K;
    B.clear(); ac.clear(); ac.push_back(0);
    for(int i = 0; i < N; i++){
      cin >> x;
      B.push_back(x); ac.push_back(x);
      ac[i+1] += ac[i];
    }
    ll acum = 0;
    for(int i = 0; i < N; i++){
        acum += B[i];
        if(acum == ac[i+1])cout<<"1\n";
        else cout <<"0\n";
    }
    ll ans = INF;
    for(int i = 0; i < N; i++){
      ans = min(ans, dp(i, 0, 0));
    }
    if(ans >= INF) ans = -1;
    cout << "Case #" << T << ": ";
    cout << ans << "\n";
  }
}
```

3.4 DFS

```
#include "../Header.cpp"

vl depht, v, nodes;
vl ind;
vector <vl>g;


ll dp[100000];

void dfs(int t)
{
    if(!v[t]){
        nodes.push_back(t);
        v[t]=1;
        for(auto it : g[t]){

            if(!v[it])
            {
              depht[it]=depht[t]+1;
              dfs(it);
            }
        }
        nodes.pop_back();
    }
}


vector<ll> tam();

void dp(int t)
{
    if(!v[t]){
        nodes.push_back(t);
        v[t]=1;
        for(auto it : g[t]){

            if(!v[it])
            {
              depht[it]=depht[t]+1;
              dfs(it);
            }
        }
        nodes.pop_back();
    }
}

int main()
{
       ll n, x, y, e, a, b, i n =0;
       cin >> n >> e;
       vector<vl> g(n, vl(0));
       rep(i, e)
    {
        cin >> x >> y;
```

```
    g[x].push_back(y);
}
    vl v(n, 0);
vl d(n,INF);
d[inicio]=0;
stack<ll> q;
q.push(inicio);
while(!q.empty()){

    ll t = q.top();
    q.pop();
    if(v[t]) continue;

    v[t] = 1;
    for(auto it : g[t]){

        if(!v[it])
        {
            d[it] = d[t]+1;
            q.push(it);
        }
    }

}


vector<vl>g(n, vl(m)), v(n, vl(m, 0)), d(n, vl(m, 0));
rep(i, n)
    rep(j, m)
    {
        cin >> g[i][j];
    }


// implicit
vl dx = {1, -1, 0, 0};
vl dy = {0, 0, 1, -1};

queue<pll> q;
q.push({0, 0});
while(!q.empty())
{
    ll x, y;
    tie(x, y) = q.front();
    q.pop();

    if(vis[x][y])continue;
    rep(z, 4)
    {
        ll nx = x + dx[z], ny = y + dy[z];

        if(nx < 0 || ny < 0 || nx >= n || ny >= m) continue;
        q.push({nx, ny});
    }
```

```
    }

        return 0;
}
```

3.5 Dijkstra

```
#include "../Header.cpp"

int main()
{
        vl d(v, INF);
        priority_queue<pll, vp, greater<pll> > q;
        ll s, t;
        q.push({0, s});
        d[s] = 0;
        while(!q.empty()){
                        ll w, u;
                        tie(w, u) = q.top();
                        q.pop();
                        if(w > d[u]) continue;
                        for(auto it : g[u]){
                                        if(d[it.second] > w + it.first){
                                                        d[it.second] = w + it.first;
                                                        q.push({d[it.second],
                                                            it.second});
                                        }
                        }
        }
}
```

3.6 D´Esopo-Pape

```
#include "../Header.cpp"

// From: https://cp-algorithms.com/graph/desopo_pape.html

int main()
{
        ll v,x,y,e;
        pll h;
        vector<ll>b;
        cin >> v >> e;
        vector<vp > g(v, vp (0));
        vl peso(v, INF);

    int w;
        for(ll i = 0; i < e; i++)
        {
                cin >> x >> y >> w;
```

```
            g[x-1].push_back({w, y-1});
            g[y-1].push_back({w, x-1});
        }
        ll s, t;

    vl d(v, INF);
    d[s] = 0;
    vl m(v, 2);
    deque<ll> q;
    q.push_back(s);
    p.assign(v, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop_front();
        m[u] = 0;
        for (auto it : g[u]) {
            if (d[it.second] > d[u] + it.first) {
                d[it.second] = d[u] + it.first;
                p[it.second] = u;
                if (m[it.second] == 2) {
                    m[it.second] = 1;
                    q.push_back(it.second);
                } else if (m[it.second] == 0) {
                    m[it.second] = 1;
                    q.push_front(it.second);
                }
            }
        }
    }

        return 0;
}
```

3.7 EulerTour

```
#include "../Header.cpp"

//Euler Tour
vl L, R, d, c;
ll num = -1;
vector<vl>g;
void dfs(ll in, ll p)
{
    num++;
    L[in] = num;
    d.push_back(c[in]);
    for(auto it : g[in])
    {
        if(p != it)
            dfs(it, in);
    }
    R[in] = num;
}
```

3.8 FloydWarshall

```
#include "../Header.cpp"

// From Competitive Programing 4 book
ll p[500][500];
void printPath(int i, int j)
{
    if(i != j) printPath(i, p[i][j]);
    cout << j+1 << " ";
}

int main()
{
    int n, m, q, x ,y ,w;


    vector<vl > g(n, vl(n, INF));
    rep(i, n)
    {
        g[i][i] = 0;
        //g[i][i] = INF; Detect cheapest positive cycle for each i
    }
    rep(i, m)
    {
        cin >> x >> y >> w;
        g[x][y] = min(g[x][y], w); // handle repeats
    }


    rep(i, n)
        rep(j, n)
            p[i][j] = i;


    rep(k, n)
        rep(i, n)
            rep(j, n)
                //g[i][j] |= (g[i][k] & g[k][j]); to find i is connected with j
                // if at the end g[i][j] & g[j][i], i and j are in the same SCC

                // To find minimal max edge in path from i to j
                //g[i][j] = min(g[i][j], max(g[i][k], g[k][j]));

                if(g[i][k] + g[k][j] < g[i][j])
                {
                    g[i][j] = g[i][k] + g[k][j];
                    p[i][j] = p[k][j];
```

```
            }

    rep(k, n)
        rep(i, n)
            rep(j, n)
                if(g[i][k] != INF && g[k][j] !=INF
                    && g[k][k] < 0)
                        g[i][j] = -INF;
   rep(i, q)
    {
        cin >> x >> y;
        if(g[x][y] == INF)
            cout << "Impossible\n";
        else if (g[x][y] == -INF)
            cout << "-Infinity\n";
        else
            cout << g[x][y] << "\n";
    }


        return 0;
}
```

3.9 HeavyLightDecomposition

```
//https://github.com/BenjaminRubio/CompetitiveProgramming/blob/master/Material/Graphs/HeavyLightDecomposition.cpp

#include "../Header.cpp"

class HLD
{
    ST st;
    // ancestor, heavy, depht, component parent, index in ST
    vi A, H, D, R, P;

    int dfs(vector<vi> &G, int u)
    {
        int ans = 1, M = 0, s;
        for (int v : G[u]) if (v != A[u])
        {
            A[v] = u, D[v] = D[u] + 1;
            s = dfs(G, v), ans += s;
            if (s > M) H[u] = v, M = s;
        }
        return ans;
    }
    template <class OP>
    void path(int u, int v, OP op)
    {
        for (; R[u] != R[v]; v = A[R[v]])
        {
            if (D[R[u]] > D[R[v]]) swap(u, v);
            op(P[R[v]], P[v] );
        }
        if (D[u] > D[v]) swap(u, v);
        op(P[u], P[v]);             // VALUES ON VERTEX
        // op(P[u] + 1, P[v]);      // VALUES ON EDGE
    }

public:
    HLD(vector<vi> &G, int n) : A(n), D(n), R(n), P(n)
    {
        st = SegmentTree(n);
        H.assign(n, -1); A[0] = -1, D[0] = 0; dfs(G, 0); int p = 0;
        rep(i, n) if (A[i] == -1 || H[A[i]] != i)
            for (int j = i; j != -1; j = H[j]) R[j] = i, P[j] = p++;
    }
    void set(int v, const node &x) { st.set(P[v], x); } // VALUES ON VERTEX
    // void set(int u, int v, const node &x)       // VALUES ON EDGE
    // {
    //     if (D[u] > D[v]) swap(u, v);
    //     st.set(P[v], x);
    // }
    void update(int u, int v, const node &x)            // OPTIONAL FOR RANGE
         UPDATES
    { path(u, v, [this, &x](int l, int r) { st.update(l, r, x); }); }
    node query(int u, int v)
    {
        node ans = node();
        path(u, v, [this, &ans](int l, int r) { ans = node(ans, st.query(l, r));
            });
        return ans;
    }
};

// USAGE: HLD<ST<Node>, Node> hld(G, N);

//// NON COMMUTATIVE QUERIES :

class HLD
{
    ST st;
    vi A, H, D, R, P;

    int dfs(vector<vi> &G, int u)
    {
        int ans = 1, M = 0, s;
        for (int v : G[u]) if (v != A[u])
        {
            A[v] = u, D[v] = D[u] + 1;
            s = dfs(G, v), ans += s;
            if (s > M) H[u] = v, M = s;
        }
        return ans;
    }
```

```
public:
    node path(int u, int v)
    {
        node ans1, ans2; bool d = 0;
        for (; R[u] != R[v]; v = A[R[v]])
        {
            if (D[R[u]] > D[R[v]]) swap(u, v), d = !d;
            if (d) ans1 = node(st.query(P[R[v]], P[v] ), ans1);
            else ans2 = node(st.query(P[R[v]], P[v] ), ans2);
        }
        if (D[u] > D[v]) swap(u, v), d = !d;
        if (d) ans1 = node(st.query(P[u], P[v] ), ans1);
        else ans2 = node(st.query(P[u], P[v] ), ans2);
        ans1.sw(); return node(ans1, ans2);
    }
    HLD(vector<vi> &G, int n) : A(n), st(n), D(n), R(n), P(n)
    {
        st = SegmentTree(n);
        H.assign(n, -1); A[0] = -1, D[0] = 0; dfs(G, 0); int p = 0;
        rep(i, n) if (A[i] == -1 || H[A[i]] != i)
            for (int j = i; j != -1; j = H[j]) R[j] = i, P[j] = p++;
    }
    void set(int v, const node &x) { st.set(P[v], x); }
};
```

3.10 Hungarian

```
#include "../Header.cpp"

// From
     https://docs.google.com/document/d/1rcex_saP4tExbbU62qGUjR3eenxOh-50i9Y45WtHkc4/edit

#define rep(i, n) for (int i = 0; i < (int)n; i++)
#define repx(i, a, b) for (int i = (int)a; i < (int)b; i++)

// Minimum/Maximum cost of a perfect matching in complete graph
// O(n^3)
template<class T>
class Hungarian
{
    T inf = numeric_limits<T>::max() / 2;
    bool maxi, swapped = false;
    vector<vector<T>> cost;
    vector<T> u, v;
    vl p, way;
    int l, r;

public:
    // left/right == partition sizes
    Hungarian(int left, int right, bool maximizing)
    {
        l = left, r = right, maxi = maximizing;
        if (swapped = l > r) swap(l, r);
        cost.assign(l + 1, vector<T>(r + 1, 0));
        u.assign(l + 1, 0); v.assign(r + 1, 0);
        p.assign(r + 1, 0); way.assign(r + 1, 0);
    }

    void add_edge(int l, int r, T w)
    {
        assert(l and r); // indices start from 1 !!
        if (swapped) swap(l, r);
        cost[l][r] = maxi ? -w : w;
    }

    // execute after all edges were added
    void calculate()
    {
        repx(i, 1, l + 1)
        {
            vector<bool> used(r+1, false);
            vector<T> minv(r+1, inf);
            int j0 = 0; p[0] = i;

            while (p[j0])
            {
                int j1, i0 = p[j0]; used[j0] = true;
                T delta = inf;
                repx(j, 1, r + 1) if (not used[j])
                {
                    T cur = cost[i0][j] - u[i0] - v[j];
                    if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                    if(minv[j] < delta) delta = minv[j], j1 = j;
                }
                rep(j, r + 1)
                {
                    if (used[j]) u[p[j]] += delta, v[j] -= delta;
                    else minv[j] -= delta;
                }
                j0 = j1;
            }

            while (j0) p[j0] = p[way[j0]], j0 = way[j0];
        }
    }

    // execute after executing calculate()
    T answer() { return maxi ? v[0] : -v[0]; }

    bool are_matched(int l, int r)
    {
        if (swapped) swap(l, r);
        return p[r] == l;
    }
};

int main(){
```

```
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    ll n;
    cin >> n;
    ll d[n][n], pos = (n+1)/2;

    Hungarian<ll> h(pos, pos, 0);

    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            cin >> d[i][j];


    for(int i = 0; i < n; i += 2)
    {
        for(int j = 0; j < n; j+= 2){
            ll cost = 0;
            if(j > 0) cost += d[i][j-1];
            if(j < n-1) cost += d[i][j+1];
            h.add_edge(i/2+1, j/2+1, cost);

        }
    }
    h.calculate();
    cout << h.answer() << "\n";
}
```

3.11 LCA-SP

```
#include "../Header.cpp"

ll maxlog2(ll x){
    return (63 - __builtin_clzll(x));
}

// To minimize diameter, connect the center of the diameter of two trees
// min(diam1, diam2, dia1+dia2+1) dia1 = diam1/2 (if diameter odd)+ 1

struct SparseTableLCA
{
    int maxlg;
    vector<vl >SP;
    vector<vl >MN;
    vl D;
    SparseTableLCA(vector<vl>& g, ll ini=0)
    {
        ll n = g.size();
        vl vis(n,0), parent(n,-1);
        D.resize(n,INF);D[ini]=0;
        queue<ll> q;
        q.emplace(ini);
        while(!q.empty()){
            ll k=q.front();q.pop();
            if(!vis[k]){
                vis[k]=1;
                for(auto it : g[k])
                    if(!vis[it])
                    {
                        parent[it]=k;
                        D[it]=D[k]+1;
                        q.push(it);
                    }

            }
        }

        SP.clear();
        SP.push_back(parent);
        maxlg = maxlog2(n);

        repx(i, 1 , maxlg+1)
        {
            vl aux;
            rep(j, n)
            {
                if(SP[i-1][j]!=-1)
                aux.push_back(SP[i-1][SP[i-1][j]]);
                else aux.push_back(-1);
            }
            SP.push_back(aux);
        }
    }

    ll maxL(ll u,ll v)//arista largo maximo
    {
        ll a,b,x=LCA(u, v);
        if(u==x)a =- 1;
        else a = query(D[x], u);
        if(v==x)b =- 1;
        else b = query(D[x], v);
        return max(a, b);
    }
    ll query(ll a,ll n)
    {
        ll maxi=-1;
        while(D[n]!=a)
        {
            maxi = max(maxi, MN[maxlog2(D[n]-a)][n]);
            n=SP[maxlog2(D[n]-a)][n];
        }
        return maxi;
    }
    ll level(ll a, ll n) // up a to depth n
    {
        while(D[n] != a)
            n = SP[maxlog2(D[n]-a)][n];
```

```
        return n;
    }
    ll LCA(ll x,ll y)
    {
        if(D[x] <= D[y]) swap(x, y);

        if(D[x] != D[y])
            x = level(min(D[x], D[y]), x);

        if(x == y) return x;

        for(ll i = maxlg; i>=0; i--)
        {
            if(SP[i][x] != SP[i][y] && SP[i][x] != -1)
            {
                x = SP[i][x];
                y = SP[i][y];
            }
        }
        return SP[0][x];
    }
    ll Dist(ll u,ll v)
    {
        return D[u] + D[v] - 2*D[LCA(u, v)];
    }
    ll kth_fartest_node(ll u, ll v, ll d)
    {
        if(Dist(u, LCA(u, v)) < d)
            return level(D[v] - (Dist(u, v) - d), v);

        else
            return level(D[u] - d, u);
    }

    // move u k steps in path to v
    ll next_path(ll u, ll v, ll k){

      if(D[u] - D[LCA(u, v)] >= k) return level(D[u] - k, u);
      else return level(D[LCA(u, v)] + k - (D[u] - D[LCA(u, v)]), v);
    }
};


/* edge weight queries
SparseTableLCA(vector<vector<pll>>& g, ll ini)
    {
        ll n = g.size();
        vl vis(n,0), parent(n,-1), b(n,-1);
        D.resize(n,INF);D[ini]=0;
        queue<ll> q;
        q.emplace(ini);
        while(!q.empty()){
            ll k=q.front();q.pop();
            if(!vis[k]){
                vis[k]=1;
                for(auto it : g[k])
                    if(!vis[it.second])
                    {
                        b[it.second]=it.first;
                        parent[it.second]=k;
                        D[it.second]=D[k]+1;
                        q.push(it.second);
                    }
            }
        }

        SP.clear();
        SP.push_back(parent);
        maxlg= 63 - __builtin_clzll(n);
        for(ll i = 1; i <= maxlg; i++)
        {
            vl c;
            for(ll j=0;j<n;j++)
            {
                if(SP[i-1][j]!=-1)
                c.push_back(SP[i-1][SP[i-1][j]]);
                else c.push_back(-1);
            }
            SP.push_back(c);
        }
        MN.clear();
        MN.push_back(b);

        for(ll i=1;i<=maxlg;i++)
        {
            vl c;
            for(ll j=0;j<n;j++)
            {
                if(MN[i-1][j]!=-1)
                c.push_back(max(MN[i-1][SP[i-1][j]],MN[i-1][j]));
                else c.push_back(-1);
            }
            MN.push_back(c);
        }
    }
*/
```

3.12 LaplacianMatrix

```
#include "../Header.cpp"

//https://www.geeksforgeeks.org/determinant-of-a-matrix/

// Dimension of input square matrix
#define N 4

// Function to get cofactor of mat[p][q] in temp[][]. n is
```

```
// current dimension of mat[][]
void getCofactor(int mat[N][N], int temp[N][N], int p,
                 int q, int n)
{
    int i = 0, j = 0;

    // Looping for each element of the matrix
    for (int row = 0; row < n; row++)
    {
        for (int col = 0; col < n; col++)
        {
            //  Copying into temporary matrix only those
            //  element which are not in given row and
            //  column
            if (row != p && col != q)
            {
                temp[i][j++] = mat[row][col];

                // Row is filled, so increase row index and
                // reset col index
                if (j == n - 1)
                {
                    j = 0;
                    i++;
                }
            }
        }
    }
}

/* Recursive function for finding determinant of matrix.
   n is current dimension of mat[][]. */
int determinantOfMatrix(int mat[N][N], int n)
{
    int D = 0; // Initialize result

    //  Base case : if matrix contains single element
    if (n == 1)
        return mat[0][0];

    int temp[N][N]; // To store cofactors

    int sign = 1; // To store sign multiplier

    // Iterate for each element of first row
    for (int f = 0; f < n; f++)
    {
        // Getting Cofactor of mat[0][f]
        getCofactor(mat, temp, 0, f, n);
        D += sign * mat[0][f]
             * determinantOfMatrix(temp, n - 1);

        // terms are to be added with alternate sign
        sign = -sign;
    }

    return D;
}

/* function for displaying the matrix */
void display(int mat[N][N], int row, int col)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
            cout <<" " << mat[i][j];
        cout <<"n";
    }
}

// Driver program to test above functions
int main()
{
    /* int mat[N][N] = {{6, 1, 1},
                        {4, -2, 5},
                        {2, 8, 7}}; */

    int mat[N][N] = { { 1, 0, 2, -1 },
                      { 3, 0, 0, 5 },
                      { 2, 1, 4, -3 },
                      { 1, 0, 5, 0 } };

    int T[N][N]; //Tutte matrix
    //range of matrix(number of pivots) is # different matching maximum
    ll n, m;
    for(int i = 0; i < m; i++)
    {
        ll x, y, w;
        cin >> x >> y >> w;
        mat[x][y] = -1;//-w;
        mat[y][x] = -1;//-w;
        mat[x][x] += 1;//w;
        mat[y][y] += 1;//w;

        T[x][x] = 0;
        T[x][y] = rand() % 1e9+7;
        T[y][x] = -T[x][y];

    }

    // O(N^3)
    cout <<"Number of spanning trees/weight sum of spanning tree : " <<
         determinantOfMatrix(mat, N-1);

    cout <<"Number of spanning forest with i in one component and j in the other
         : " << determinantOfMatrix(mat, N-i -j);//delete i/t row and column

    if(determinantOfMatrix(T, N) == 0)
        cout <<"Does not exist a maximum matching in the general graph (need more
             verify)";
```

```
    else
        cout <<"Does exist perfect matching";
    cout <<"Number of spanning trees/weight sum of spanning tree : " <<
         determinantOfMatrix(T, N);
    return 0;
}
```

3.13 LinkCutTree

```
//https://github.com/mhunicken/icpc-team-notebook-el-vasito/blob/master/data_structures/linkcut2.cpp
//https://tc-arg.tk/pdfs/2020/linkcut.pdf
// query de aristas: crear nodos en cada aristas con valor
const int N_DEL = 0, N_VAL = 0; //delta, value
inline int mOp(int x, int y){return x+y;}//modify
inline int qOp(int lval, int rval){return lval + rval;}//query
inline int dOnSeg(int d, int len){return d==N_DEL ? N_DEL : d*len;}
//mostly generic
inline int joinD(int d1, int d2){
  if(d1==N_DEL)return d2;if(d2==N_DEL)return d1;return mOp(d1, d2);}
inline int joinVD(int v, int d){return d==N_DEL ? v : mOp(v, d);}
struct Node_t{
  int sz, nVal, tVal, d;
  bool rev;
  Node_t *c[2], *p;
  Node_t(int v) : sz(1), nVal(v), tVal(v), d(N_DEL), rev(0), p(0){
    c[0]=c[1]=0;
  }
  bool isRoot(){return !p || (p->c[0] != this && p->c[1] != this);}
  void push(){
    if(rev){
      rev=0; swap(c[0], c[1]);
      fore(x,0,2)if(c[x])c[x]->revˆ=1;
    }
    nVal=joinVD(nVal, d); tVal=joinVD(tVal, dOnSeg(d, sz));
    fore(x,0,2)if(c[x])c[x]->d=joinD(c[x]->d, d);
    d=N_DEL;
  }
  void upd();
};
typedef Node_t* Node;
int getSize(Node r){return r ? r->sz : 0;}
int getPV(Node r){
  return r ? joinVD(r->tVal, dOnSeg(r->d,r->sz)) : N_VAL;}
void Node_t::upd(){
  tVal = qOp(qOp(getPV(c[0]), joinVD(nVal, d)), getPV(c[1]));
  sz = 1 + getSize(c[0]) + getSize(c[1]);
}
void conn(Node c, Node p, int il){if(c)c->p=p;if(il>=0)p->c[!il]=c;}
void rotate(Node x){
  Node p = x->p, g = p->p;
  bool gCh=p->isRoot(), isl = x==p->c[0];
  conn(x->c[isl],p,isl); conn(p,x,!isl);
  conn(x,g,gCh?-1:(p==g->c[0])); p->upd();
}
void spa(Node x){//splay
  while(!x->isRoot()){
    Node p = x->p, g = p->p;
    if(!p->isRoot())g->push();
    p->push(); x->push();
    if(!p->isRoot())rotate((x==p->c[0])==(p==g->c[0])? p : x);
    rotate(x);
  }
  x->push(); x->upd();
}
Node exv(Node x){//expose
  Node last=0;
  for(Node y=x; y; y=y->p)spa(y),y->c[0]=last,y->upd(),last=y;
  spa(x);
  return last;
}
void mkR(Node x){exv(x);x->revˆ=1;}//makeRoot
Node getR(Node x){exv(x);while(x->c[1])x=x->c[1];spa(x);return x;}
Node lca(Node x, Node y){exv(x); return exv(y);}
bool connected(Node x, Node y){exv(x);exv(y); return x==y?1:x->p!=0;}
void link(Node x, Node y){mkR(x); x->p=y;}
void cut(Node x, Node y){mkR(x); exv(y); y->c[1]->p=0; y->c[1]=0;}
Node father(Node x){
       exv(x);
       Node r=x->c[1];
       if(!r)return 0;
       while(r->c[0])r=r->c[0];
       return r;
}
void cut(Node x){ // cuts x from father keeping tree root
       exv(father(x));x->p=0;}
int query(Node x, Node y){mkR(x); exv(y); return getPV(y);}
void modify(Node x, Node y, int d){mkR(x);exv(y);y->d=joinD(y->d,d);}
Node lift_rec(Node x, int t){
       if(!x)return 0;
       if(t==getSize(x->c[0]))return x;
       if(t<getSize(x->c[0]))return lift_rec(x->c[0],t);
       return lift_rec(x->c[1],t-getSize(x->c[0])-1);
}
Node lift(Node x, int t){ // t-th ancestor of x (lift(x,1) is x's father)
       exv(x);return lift_rec(x,t);}
int depth(Node x){ // distance from x to its tree root
       exv(x);return getSize(x)-1;}
```

3.14 MaxBipartiteMatching

```
#include "../Header.cpp"

// From https://github.com/stevenhalim/cpbook-code/blob/master/ch4/mcbm.cpp
```

```cpp
// Works with 3-pairing or more (Perfect MCBM)

vl match, vis;                              // global variables
vector<vl> g;

int Aug(int L) {
  if (vis[L]) return 0;                     // L visited, return 0
  vis[L] = 1;
  for (auto it : g[L])
    if ((match[it] == -1) || Aug(match[it])) {
      match[it] = L;                         // flip status
      return 1;                             // found 1 matching
    }
  return 0;                                 // no matching
}



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

    ll V, Vleft;
    // VLeft and VRight can have common vertices names
    // match[R] -> L
    match.assign(V, -1);
    ll MCBM = 0;
    for(int L = 0; L < Vleft; L++)
    {
        vis.assign(Vleft, 0);
        MCBM += Aug(L);
    }
    cout << "Found " << MCBM << " matchings\n"; // the answer is 2 for Figure 4.38
    cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
    return 0;
}
```

3.15 MaxFlowDinic

```cpp
// Time Complexity:
// - general worst case: O (|E| * |V|^2)
// O (|E||max_flow|) ford fulkenson
// - unit capacities: O( min( V^(2/3), sqrt(E) ) )
// - Bipartite graph (unit capacities) + source & sink (any capacities): O(E sqrt
    V)

// minimo corte:
// separa en 2 colores el grafo y el minimo corte
// es la separacin con menos aristas entre distinto color
// mandar infinito a nodos que estan obligados a ser de un colors
// min_cut == max_flow

// max matching = max_flow (grafo bipartito)
// minimo cubrimiento nodos = nodos - max_flow


// maximo matching m*sqrt(n)
#include "../Header.cpp"

struct Dinic {
    struct edge {
        int to, rev;
        ll f, cap;
    };

    vector<vector<edge>> g;
    vector<ll> dist;
    vector<int> q, work;
    int n, sink;

    bool bfs(int start, int finish) {
        dist.assign(n, -1);
        dist[start] = 0;
        int head = 0, tail = 0;
        q[tail++] = start;
        while (head < tail) {
            int u = q[head++];
            for (const edge &e : g[u]) {
                int v = e.to;
                if (dist[v] == -1 and e.f < e.cap) {
                    dist[v] = dist[u] + 1;
                    q[tail++] = v;
                }
            }
        }
        return dist[finish] != -1;
    }

    ll dfs(int u, ll f) {
        if (u == sink)
            return f;
        for (int &i = work[u]; i < (int)g[u].size(); ++i) {
            edge &e = g[u][i];
            int v = e.to;
            if (e.cap <= e.f or dist[v] != dist[u] + 1)
                continue;
            ll df = dfs(v, min(f, e.cap - e.f));
            if (df > 0) {
                e.f += df;
                g[v][e.rev].f -= df;
                return df;
            }
        }
        return 0;
    }
```

```
    Dinic(int n) {
        this->n = n;
        g.resize(n);
        dist.resize(n);
        q.resize(n);
    }

    void add_edge(int u, int v, ll cap) {
        edge a = {v, (int)g[v].size(), 0, cap};
        edge b = {u, (int)g[u].size(), 0, 0}; //Poner cap en vez de 0 si la arista
             es bidireccional
        g[u].push_back(a);
        g[v].push_back(b);
    }

    ll max_flow(int source, int dest) {
        sink = dest;
        ll ans = 0;
        while (bfs(source, dest)) {
            work.assign(n, 0);
            while (ll delta = dfs(source, LLONG_MAX))
                ans += delta;
        }
        return ans;
    }
};

// usage
int main() {
    Dinic din(2);
    din.add_edge(0, 1, 10);
    ll mf = din.max_flow(0,1);
}
```

3.16 MaxFlowFordFulkerson

```
#include "../Header.cpp"

// minimo cubrimiento aristas = min-cut
// extremos de aristas cortadas que no son s ni t
// max conj, ind = nodos - minimo cubrimiento aristas

// O(VE^2)
// O(EF)

// Number of vertices in given graph
#define V 6

/* Returns true if there is a path from source 's' to sink 't' in
  residual graph. Also fills parent[] to store the path */
int bfs(int rGraph[V][V], int s, int t, int parent[])
{
    // Create a visited array and mark all vertices as not visited
    bool visited[V];
    memset(visited, 0, sizeof(visited));

    // Create a queue, enqueue source vertex and mark source vertex
    // as visited
    queue <int> q;
    q.push(s);
    visited[s] = true;
    parent[s] = -1;

    // Standard BFS Loop
    while (!q.empty())
    {
        int u = q.front();
        q.pop();

        for (int v=0; v<V; v++)
        {
            if (visited[v]==false && rGraph[u][v] > 0)
            {
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }

    // If we reached sink in BFS starting from source, then return
    // true, else false
    return (visited[t] == true);
}

// A DFS based function to find all reachable vertices from s. The function
// marks visited[i] as true if i is reachable from s. The initial values in
// visited[] must be false. We can also use BFS to find reachable vertices
void dfs(int rGraph[V][V], int s, bool visited[])
{
    visited[s] = true;
    for (int i = 0; i < V; i++)
       if (rGraph[s][i] && !visited[i])
           dfs(rGraph, i, visited);
}

// Prints the minimum s-t cut
void minCut(int graph[V][V], int s, int t)
{
    int u, v;

    // Create a residual graph and fill the residual graph with
    // given capacities in the original graph as residual capacities
    // in residual graph
    int rGraph[V][V]; // rGraph[i][j] indicates residual capacity of edge i-j
    for (u = 0; u < V; u++)
        for (v = 0; v < V; v++)
```

```
            rGraph[u][v] = graph[u][v];

    int parent[V]; // This array is filled by BFS and to store path
    int max_flow = 0; // There is no flow initially
    // Augment the flow while there is a path from source to sink
    while (bfs(rGraph, s, t, parent))
    {
        // Find minimum residual capacity of the edhes along the
        // path filled by BFS. Or we can say find the maximum flow
        // through the path found.
        int path_flow = INT_MAX;
        for (v=t; v!=s; v=parent[v])
        {
            u = parent[v];
            path_flow = min(path_flow, rGraph[u][v]);
        }

        // update residual capacities of the edges and reverse edges
        // along the path
        for (v=t; v != s; v=parent[v])
        {
            u = parent[v];
            rGraph[u][v] -= path_flow;
            rGraph[v][u] += path_flow;
        }
        max_flow += path_flow;
    }

    // Flow is maximum now, find vertices reachable from s
    bool visited[V];
    memset(visited, false, sizeof(visited));
    dfs(rGraph, s, visited);

    // Print all edges that are from a reachable vertex to
    // non-reachable vertex in the original graph
    for (int i = 0; i < V; i++)
      for (int j = 0; j < V; j++)
         if (visited[i] && !visited[j] && graph[i][j])
              cout << i << " - " << j << endl;

    return;
}

// Driver program to test above functions
int main()
{
    // Let us create a graph shown in the above example
    int graph[V][V] = { {0, 16, 13, 0, 0, 0},
                        {0, 0, 10, 12, 0, 0},
                        {0, 4, 0, 0, 14, 0},
                        {0, 0, 9, 0, 0, 20},
                        {0, 0, 0, 7, 0, 4},
                        {0, 0, 0, 0, 0, 0}
                      };
    minCut(graph, 0, 5);

    return 0;
}
```

3.17 MinCostMaxFlow

```
#include "../Header.cpp"

struct Edge
{
    int from, to, capacity, cost;
};

vector<vector<int>> adj, cost, capacity;

const int INF = 1e9;

void shortest_paths(int n, int v0, vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);

    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] > d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
        }
    }
}

 // flow, source, to;
int min_cost_flow(int N, vector<Edge> edges, int K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
```

```
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }

    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;

        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f, capacity[p[cur]][cur]);
            cur = p[cur];
        }

        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}

int sup[55], inf[55];

int main(){
    int n, q;

    vector<Edge>ee;
    Edge E;


    E.from = i;
    E.to = id;
    E.capacity = 1;
    E.cost = 2 * j + 1;
    ee.push_back(E);
    E.from = id;
    E.to = 1;
    E.capacity = 1;
    E.cost = 0;
    ee.push_back(E);
    id++;

    ans = min_cost_flow(2 * n + n*n + 10, ee, n, 0, 1);
    cout << ans << "\n";


    cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
}
```

3.18 SCC

```
#include "../Header.cpp"

// From CP4:
     https://github.com/stevenhalim/cpbook-code/blob/master/ch4/traversal/UVa11838.cpp

vl dfs_num, dfs_low, visited;
vector<vl> g, invg;
void Kosaraju(int u, int pass, vl& S) { // pass = 1 (original), 2 (transpose)
  dfs_num[u] = 1;
  vl &neighbor = (pass == 1) ? g[u] : invg[u];
  for (auto it : neighbor)
    if (dfs_num[it] == -1)
      Kosaraju(it, pass, S);
  S.push_back(u);
}
//


// ------------------------------------------
// implementation of Tarjan's SCC algorithm
stack<int> St;
int cont, numSCC;

void tarjanSCC(int u) {
  dfs_low[u] = dfs_num[u] = cont;
  cont++;
  St.push(u);
  visited[u] = 1;
  for (auto v : g[u]) {
    if (dfs_num[v] == -1)
      tarjanSCC(v);
    if (visited[v])
      dfs_low[u] = min(dfs_low[u], dfs_low[v]);
  }

  if (dfs_low[u] == dfs_num[u]) {
    while (1) {
      int v = St.top(); St.pop(); visited[v] = 0;

      if (u == v) break;
```

```
    }
    ++numSCC;
  }
}


int main() {

    int n, m; cin >> n >> m;
    vector<vl> g(n);
    while(m--) {
        int u, v; cin >> u >> v; u--, v--;
        g[u].push_back(v);
    }
    // run Tarjan's SCC
    dfs_num.assign(n, -1); dfs_low.assign(n, 0); visited.assign(n, 0);
    while (!St.empty()) St.pop();
    cont = numSCC = 0;
    for (int u = 0; u < n; ++u)
     if (dfs_num[u] == -1)
       tarjanSCC(u);


    //Kosaraju's SCC

    vl S;
    dfs_num.assign(n, -1);
    for (int u = 0; u < n; ++u)
        if (dfs_num[u] == -1)
            Kosaraju(u, 1, S);
    int numSCC = 0;
    dfs_num.assign(n, -1);
    for (int i = n-1; i >= 0; --i){
        vl comp;
        if (dfs_num[S[i]] == -1)
            numSCC++, Kosaraju(S[i], 2, comp);        // on transposed graph
    }

    return 0;
}
```

3.19 StableMatching

```
#include "../Header.cpp"
// Match the preferences of N clients and M restaurants that no side prefer
     another
// restaurant or client, respectively.

// hospital-residents variation of stable matching
int main(){

       ios_base::sync_with_stdio(0);

       cin.tie(0);
       ll n, m, k, x;
       cin >> n >> m;

       vl cap(m, 1), match(n, -1);

       vector<queue<ll>>pref(n);
       queue<ll> q;
       for(auto &it : cap) cin >> it; // capacity

       for(int i = 0; i < n; i++)
       {
              q.push(i);
              cin >> k;
              while(k--)
              {
                     cin >> x;
                     pref[i].push(x-1); //client preference list
              }
       }

       vector<unordered_map<ll, ll>> res_pref(m);
       for(int i = 0; i < m; i++)
       {
              ll id = 0;
              cin >> k;
              while(k--)
              {
                     cin >> x;
                     res_pref[i][x-1] = -id; // restaurant preference list
                     id++;
              }
       }

       vector<set<pll>> in_res(m);
       while(!q.empty())
       {
              ll cl = q.front();
              q.pop();

              ll rest = pref[cl].front(); // actual preference restaurant

              in_res[rest].insert({res_pref[rest][cl], cl});// add client to
                   restaurant
              match[cl] = rest;
              pref[cl].pop(); // remove client preference

              if(cap[rest] == 0)
              {
                     ll cl_new = (*in_res[rest].begin()).second; // erase client
                          with less preference
                     in_res[rest].erase(in_res[rest].begin());

                     match[cl_new] = -1;
```

```
                if(!pref[cl_new].empty()) // add client to queue if he has
                    more preferences
                    q.push(cl_new);
            }
            else
                cap[rest]--;
        }
        for(int i = 0; i < n; i++)
            if(match[i] != -1) cout << i+1 <<" "<<match[i]+1 <<"\n";

}
```

3.20 ToposortDFS

```
#include "../Header.cpp"

vl s, v;
vector<vl > g;
void dfs(int t)
{
    v[t] = 1;
    for(auto it : g[t])
        if(!v[it])
            dfs(it);

    s.pb(t);
}
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
        ll n;
        g.assign(n, vl());
    v.assign(n, 0);

    rep(i, n)
        if(!v[i])
            dfs(i);

    reverse(ALL(s));
        return 0;
}
```

3.21 ToposortKhan

```
#include "../Header.cpp"

// All toposort orderings
vector<vl> g;
ll in;
vl indegree, vis;
vl sorted;
vector<char> ans;
bool possible;
void toposort()
{
    bool flag = 0;

    for(int i = 0; i < in; i++)
    {
        if(indegree[i] == 0 && !vis[i])
        {
            sorted.push_back(i);


            for(auto it : g[i])
                indegree[it]--;
            vis[i] = 1;
            toposort();
            vis[i] = 0;
            sorted.pop_back();

            for(auto it : g[i])
                indegree[it]++;
            flag = 1;
        }
    }
    if(!flag && sorted.size() == in)
    {
        possible = 1;
        for(int i = 0; i < in; i++)
        {
            if(i > 0) cout << " ";
            cout << ans[sorted[i]];
        }
        cout << "\n";
    }
}

int main()
{
        ll v,x,y,e,a,b,in=0;
        pll h;
        cin>>v >> e;
        vector<vl > g(v, vl(0));
        for(int i = 0; i < e; i++)
    {
        cin >> x >> y;
        g[x].push_back(y);
    }

    //set indegrees
    vl indegree(v, 0);
    vl sorted;
    for(int i=0; i < v; i++)
```

```
    {
        for(auto it : g[i])
            indegree[it]++;
    }
    queue<ll> q;
    //agregar par a cola para tener los paquetes o niveles de profundidad, sumar
    for(int i = 0; i < v; i++)
        if(indegree[i] == 0)
            q.push(i);

    while(!q.empty()){

        ll t = q.front();
        q.pop();

        for(auto it : g[t]){

            if(--indegree[it] == 0)
                q.push(it);
        }
    }
}
```

4 Header

```
#include<bits/stdc++.h>
#pragma GCC optimize("Ofast")
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef vector<ll> vl;
typedef vector<int> vi;
typedef pair<ll,ll> pll;
typedef vector<pll> vp;
typedef double db;
#define INF 1e17
#define INF32 INT_MAX
#define EPS 1e-7
#define ALL(x) x.begin() , x.end()
#define ALLR(x) x.rbegin() , x.rend()
#define UNIQUE(c) (c).resize(unique(ALL(c)) - (c).begin())
#define PI acos(-1.0)
#define pb push_back
#define rep(i, n) for (int i = 0; i < (int)n; i++)
#define repx(i, a, b) for (int i = (int)a; i < (int)b; i++)

#define DBG 1

#define cerr \
    if (DBG) cerr

int main() {
        ios::sync_with_stdio(false);
        cin.tie(0);
        cout.tie(0);
        srand((unsigned int) time(0));

        // Code here

        // Compile:
        // g++ Code1.cpp && ./a.out < in > out
        // ulimit -s 1048576 more stack size 1gb
        // g++ -std=c++11 Code1.cpp && a.exe < in > out
        cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
        return 0;
}
```

5 Math

5.1 ArithmeticEval

```
#include "../Header.cpp"


ll k, n;
string s;

ll eval(ll l, ll r){
        ll open = 0;
        stack<ll> vals;
        stack<char> ops;

        if(s[l] == '+' || s[l] == '*' || s[r] == '+' || s[r] == '*')return -1;
        if(s[l] == '0' && r-l >= 1 && isdigit(s[l+1]))return -1;

        repx(i, l, r+1){
                if(isdigit(s[i])){
                        vals.push((s[i] - '0') % k);
                        while(i < r && isdigit(s[i+1])){
                                i++;
                                vals.top() = (vals.top() * 10 + (s[i] - '0')) % k;


                        }
                }
                else if(s[i] == '('
                {
                        open++;
                        ops.push(s[i]);
                }
                else if(s[i] == ')')
                {
                        open--;
```

```
                if(open < 0) return -1;
                ll in = i-1;
                while(ops.top() != '('){
                        ll aux = vals.top();
                        vals.pop();
                        if(ops.top() == '+'){
                                vals.top() = (vals.top() + aux) % k;
                        }
                        else{

                                vals.top() = (vals.top() * aux) % k;
                        }
                        ops.pop();
                }
                ops.pop();
        }
        else{
                // higher precedence first
                while(!ops.empty() && ops.top() == '*'){
                        ll aux = vals.top();
                        vals.pop();
                        vals.top() = (vals.top() * aux) % k;
                        ops.pop();
                }
                ops.push(s[i]);
        }
    }
    if(open != 0) return -1;

    while(!ops.empty()){
            ll aux = vals.top();
            vals.pop();
            if(ops.top() == '+'){
                    vals.top() = (vals.top() + aux) % k;
            }
            else{
                    vals.top() = (vals.top() * aux) % k;
            }
            ops.pop();
    }

    return vals.top();
}

int main(){

    ios_base::sync_with_stdio(0);
    cin.tie(0);
    srand((unsigned int) time(0));


    cin >> k >> n;
    cin >> s;

    //cout << eval(0, n-1)<<"\n";
    ll ans = 0;
    rep(i, n){
            repx(j, i, n){
                    if(eval(i, j) == 0){
                            ans++;
                            //cout << i << " " << j <<" "<<eval(i,j)<< "\n";
                    }

            }
    }
    cout << ans << "\n";
}
```

5.2 CRT

```
#include "../Header.cpp"

//https://github.com/PabloMessina/Competitive-Programming-Material/blob/master/Mathematics

inline ll mod(ll x, ll m) { return ((x %= m) < 0) ? x+m : x; }
inline ll mul(ll x, ll y, ll m) { return (x * y) % m; }
inline ll add(ll x, ll y, ll m) { return (x + y) % m; }

// extended euclidean algorithm
// finds g, x, y such that
//   a * x + b * y = g = GCD(a,b)
ll gcdext(ll a, ll b, ll& x, ll& y) {
    ll r2, x2, y2, r1, x1, y1, r0, x0, y0, q;
    r2 = a, x2 = 1, y2 = 0;
    r1 = b, x1 = 0, y1 = 1;
    while (r1) {
        q = r2 / r1;
        r0 = r2 % r1;
        x0 = x2 - q * x1;
        y0 = y2 - q * y1;
        r2 = r1, x2 = x1, y2 = y1;
        r1 = r0, x1 = x0, y1 = y0;
    }
    ll g = r2; x = x2, y = y2;
    if (g < 0) g = -g, x = -x, y = -y; // make sure g > 0
    // for debugging (in case you think you might have bugs)
    // assert (g == a * x + b * y);
    // assert (g == __gcd(abs(a),abs(b)));
    return g;
}

void modInverse(int a, int m)
{
    int x, y;
    int g = gcdext(a, m, &x, &y);
    if (g != 1)
        cout << "Inverse doesn't exist";
```

```
    else
    {
        // m is added to handle negative x
        int res = (x%m + m) % m;
        cout << "Modular multiplicative inverse is " << res;
    }
}

// ============================================
// CRT for a system of 2 modular linear equations
// ============================================
// We want to find X such that:
//   1) x = r1 (mod m1)
//   2) x = r2 (mod m2)
// The solution is given by:
//    sol = r1 + m1 * (r2-r1)/g * x' (mod LCM(m1,m2))
// where x' comes from
//   m1 * x' + m2 * y' = g = GCD(m1,m2)
//   where x' and y' are the values found by extended euclidean algorithm (gcdext)
// Useful references:
//   https://codeforces.com/blog/entry/61290
//   https://forthright48.com/chinese-remainder-theorem-part-1-coprime-moduli
//   https://forthright48.com/chinese-remainder-theorem-part-2-non-coprime-moduli
// ** Note: this solution works if lcm(m1,m2) fits in a long long (64 bits)
pair<ll,ll> CRT(ll r1, ll m1, ll r2, ll m2) {
    ll g, x, y; g = gcdext(m1, m2, x, y);
    if ((r1 - r2) % g != 0) return {-1, -1}; // no solution
    ll z = m2/g;
    ll lcm = m1 * z;
    ll sol = add(mod(r1, lcm), m1*mul(mod(x,z),mod((r2-r1)/g,z),z), lcm);
    // for debugging (in case you think you might have bugs)
    // assert (0 <= sol and sol < lcm);
    // assert (sol % m1 == r1 % m1);
    // assert (sol % m2 == r2 % m2);
    return {sol, lcm}; // solution + lcm(m1,m2)
}

// ============================================
// CRT for a system of N modular linear equations
// ============================================
//  Args:
//      r = array of remainders
//      m = array of modules
//      n = length of both arrays
//  Output:
//      a pair {X, lcm} where X is the solution of the sytemm
//          X = r[i] (mod m[i]) for i = 0 ... n-1
//      and lcm = LCM(m[0], m[1], ..., m[n-1])
//      if there is no solution, the output is {-1, -1}
// ** Note: this solution works if LCM(m[0],...,m[n-1]) fits in a long long (64
    bits)
#define rep(i,a,b) for (int i=a; i<b; ++i)
pair<ll,ll> CRT(ll* r, ll* m, int n) {
    ll r1 = r[0], m1 = m[0];
    rep(i,1,n) {
        ll r2 = r[i], m2 = m[i];
        ll g, x, y; g = gcdext(m1, m2, x, y);
        if ((r1 - r2) % g != 0) return {-1, -1}; // no solution
        ll z = m2/g;
        ll lcm = m1 * z;
        ll sol = add(mod(r1, lcm), m1*mul(mod(x,z),mod((r2-r1)/g,z),z), lcm);
        r1 = sol;
        m1 = lcm;
    }
    // for debugging (in case you think you might have bugs)
    // assert (0 <= r1 and r1 < m1);
    // rep(i,0,n-1) assert (r1 % m[i] == r[i]);
    return {r1, m1};
}
```

5.3 Combinatory

```
#include "../Header.cpp"

const int M = 1e9+7;
// binary exponent
ll expmod(ll b, ll e){
    ll ans = 1;
    while(e){
        if(e&1) ans = ans*b %M;
        b = b*b %M; e >>= 1;
    }
    return ans;
}

// When M is prime
ll invmod(ll a){ return expmod(a, M-2); }

//inv modular factoriales
const ll MAXN = 1e5 + 1;
ll F[MAXN], INV[MAXN], FI[MAXN];
// ...

F[0] = 1; repx(i, 1, MAXN) F[i] = F[i-1]*i %M;
INV[1] = 1; repx(i, 2, MAXN) INV[i] = M - (ll)(M/i)*INV[M%i]%M;
FI[0] = 1; repx(i, 1, MAXN) FI[i] = FI[i-1]*INV[i] %M;


// combinatory
ll Comb(ll n, ll k){
    if(n < k) return 0;
    return F[n]*FI[k] %M *FI[n-k] %M;
}

// M balls in N spaces Comb(M + N-1, N-1)

// combinatury precalc
```

```
ll C[MAXN][MAXK];
// ...
rep(i, MAXN){
    C[i][0] = 1; if(i < MAXN) C[i][i] = 1;
    repx(j, 1, min(i, (int)MAXK))
        C[i][j] = (C[i-1][j-1] + C[i-1][j])%M;
}


// divide a elements into b segments = C[a-1][b-1]
// each segment has at least 1 element

//Wilson theorem
//(p-1)! mod p = -1 if p prime

//https://cp-algorithms.com/algebra/factorial-modulo.html#multiplicity-of-p

// n! mod p with p prime and ignore multiples of p
int factmod(int n, int p) {
    vector<int> f(p);
    f[0] = 1;
    for (int i = 1; i < p; i++)
        f[i] = f[i-1] * i % p;

    int res = 1;
    while (n > 1) {
        if ((n/p) % 2)
            res = p - res;
        res = res * f[n%p] % p;
        n /= p;
    }
    return res;
}

// number of times p divides n! = v_p, n! % p*v_p = 0
int multiplicity_factorial(int n, int p) {
    int count = 0;
    do {
        n /= p;
        count += n;
    } while (n);
    return count;
}

// combinatoria n = 1e18, primo chico
// lucas
const int M = 3005;
int C[M][M];
// ...
ll lucas(ll n, ll k, int p){
    ll ans = 1;
    while(n + k){
        ans = (ans * C[n%M][k%M]) % M;
        n /= M; k /= M;
    }
    return ans;
}
```

5.4 ErathostenesSieve

```
#include "../Header.cpp"

vector <bool> crib;
void criba(ll b)
{
    crib.assign(b, 1);
    crib[0] = 0;
    crib[1] = 0;
    repx(k, 2, sqrt(b+1) + 1)
        if(crib[k])
            for(int j=2; (j * k) < b + 1; j++)//optimization j=k
                crib[k*j] = 0;
}
```

5.5 FFT

```
#include "../Header.cpp"

#define M_PIl 3.1415926535897932384626433832795028841L

typedef complex<double> C;
typedef vector<double> vd;

void fft(vector<C> &a)
{
    int n = a.size(), L = 31 - __builtin_clz(n);
    static vector<complex<long double>> R(2, 1);
    static vector<C> rt(2, 1);
    for (static int k = 2; k < n; k *= 2)
    {
        R.resize(n);
        rt.resize(n);
        auto x = polar(1.0L, M_PIl / k);
        repx(i, k, 2 * k) rt[i] = R[i] = i & 1 ? R[i / 2] * x : R[i / 2];
    }
    vector<int> rev(n);
    rep(i, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2) for (int i = 0; i < n; i += 2 * k) rep(j, k)
    {
        auto x = (double *)&rt[j + k], y = (double *)&a[i + j + k];
        C z(x[0] * y[0] - x[1] * y[1], x[0] * y[1] + x[1] * y[0]);
        a[i + j + k] = a[i + j] - z;
        a[i + j] += z;
    }
```

```
}

vd conv(const vl &a, const vl &b)
{
    if (a.empty() || b.empty()) return {};
    vd res(a.size() + b.size() - 1);
    int L = 32 - __builtin_clz(res.size()), n = 1 << L;
    vector<C> in(n), out(n);
    copy(a.begin(), a.end(), in.begin());
    rep(i, b.size()) in[i].imag(b[i]);
    fft(in);
    for (auto &x : in) x *= x;
    rep(i, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i, res.size()) res[i] = imag(out[i]) / (4 * n);
    return res;
}

//slower
vl convMod(const vl &a, const vl &b, int M)
{
    if (a.empty() || b.empty()) return {};
    vl res(a.size() + b.size() - 1);
    int B = 32 - __builtin_clz(res.size()), n = 1 << B, cut = int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i, a.size()) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i, b.size()) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R);
    rep(i, n)
    {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i, res.size())
    {
        ll av = ll(real(outl[i]) + .5), cv = ll(imag(outs[i]) + .5);
        ll bv = ll(imag(outl[i]) + .5) + ll(real(outs[i]) + .5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cout.tie(0);
    int n, m;
    cin >> n >> m;
    string s, t;
    cin >> s >> t;
    // SAME SIZE
    vl an(n, 0), am(n ,0);
    vl bn(n, 0), bm(n ,0);
    for (int k = 0; k < n; ++k) {
        if (s[k] == 'a') an[k] = 1;
        else bn[k] = 1;
    }
    for (int k = 0; k < m; ++k) {
        if (t[k] == 'a') am[k] = 1;
        else bm[k] = 1;
    }
    reverse(am.begin(), am.end());
    reverse(bm.begin(), bm.end());
    vd resA= conv(an,am);
    vd resB = conv(bn,bm);
    vector<vector<int>> ans;
    ans.assign(m+1, vector<int>());

    //n > m
    // All complete count mathces
    for (int i = n-1; i < 2*n - m; ++i) {
        ans[m - round(resA[i]) - round(resB[i])].push_back(i-n+2);
    }
    // or these ranges for an and bm with original legnths
    for (int j = m-1; j < n; ++j) {
    for (int j = 0; j <= m; ++j) {
        cout << j << ":";
        for (int u: ans[j]) cout << " " << u;
        cout << "\n";
    }
    cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
}
```

5.6 Functions

```
#include "../Header.cpp"

using namespace chrono;
auto start1 = high_resolution_clock::now();
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start1);
//cerr << duration.count()/1000 << "ms" << endl;

default_random_engine generator;
uniform_real_distribution<double> distribution(0,LLONG_MAX);
ll num = distribution(generator);


// files
ifstream input;
input.open("divisibility-tree.in");
input >> n;
ofstream output;
output.open("divisibility-tree.out");
output<<" ";
```

```
output.close();

// subset sum
for i = 0 to n-1
    for mask = 0 to (1n) - 1
        if (mask & (1i))
            dp(mask) += dp(mask - (1i))

// divisor sum
for p    P
    for x    S // Increasing order
        if (p divide a x)
            dp(x) += dp(x / p)

// hash pairs unrderedmap<pll,ll,hash_pair>
struct hash_pair {
    template <class T1, class T2>
    size_t operator()(const pair<T1, T2>& p) const
    {
        auto hash1 = hash<T1>{}(p.first);
        auto hash2 = hash<T2>{}(p.second);
        return hash1 ^ hash2;
    }
};

//propagate val in mask to all its submask
for (int i = 0; i < p; i++)
{
    for(int mask = 0; mask < (1 << p); mask++)
    {
        if((mask & (1 << i)) == 0)
            f[mask] += f[mask | (1 << i)];
        if(mask & (1 << i)) // to propagate from submasks to mask
            dp[mask] += dp[mask - (1 << i)];
    }
}

// Iterate over non empty subsets of bitmask
for(int s=m;s;s=(s-1)&m) // Decreasing order
for (int s=0;s=s-m&m;)  // Increasing order

// O(3^n)
rep(m, (1 << n)){
    // 2^k k: number of bits in m
    // iterates over al submasks of m in descending order of value
    for(int s = m; ; s = (s-1) & m){
        cout << s << endl;
        if(s == 0) break;
    }
}

int bit_opst(ll N,ll S)
{

    // Return the numbers the numbers of 1-bit in x
    int __builtin_popcount (unsigned int x)
    // Returns the number of trailing 0-bits in x. x=0 is undefined.
    int __builtin_ctz (unsigned int x)
    // Returns the number of leading 0-bits in x. x=0 is undefined.
    int __builtin_clz (unsigned int x)

    //Obtain the remainder (modulo) of S when it is divided by N (N is a power of
        2)
    return S &(N -1);
    //Determine if S is a power of 2.
    return (S &( S -1)) == 0;
    //Turn  o  the last bit in S, e.g.S = (40)10 = (101000)2 S = (32)10 =
        (100000)2.
    return S &( S -1);
    // Turn on the last zero in S, e.g.S = (41)10 = (101001)2 S = (43)10 =
        (101011)2.
    return S|| (S + 1);
    // Turn  o  the last consecutive run of ones in S
    return S &( S + 1);
    // Turn on the last consecutive run of zeroes in S
    return S|| (S -1);
    // Turn on all bits
    return S = (1 << 62)-1;
    // multiply by 2^N
    return S<<=N;
    // Divide by 2^N
    return S>>=N;
    // Turn on the N-th bit
    return S = S||(1<<N);
    // Check if N-th bit is on
    return (S & (1 << N));
    // Turn off the N-th bit
    return S &= ~(1 << N);
    // Alternate satatus of N-th bir
    return S ^= (1 << N);
    //Value of the least significant bit on from the right
    return N = (S&(-S));
}
//count numbers with i bit set [1, n-1]
ll kol(ll n, ll i)
{
    return (n / (1ll << (i + 1))) * (1ll << i) + max((n % (1ll << (i + 1))) -
        (1ll << i), 0ll);
}
kol(r+1, i) - kol(l, i);

// Gradien descent
db find(db x, int dir)
{
    rep(i, 10)
    {
        db x2 = x + dir * 0.01 * der(x);
        x = x2;
    }
    while(abs(der(x)) > 0.001 )
```

```
    {
        db x2 = x + dir * 0.0001 * der(x);
        x = x2;
    }

    return x;
}



// old implemented algorithms:

int A[10000];//Set con reset O(1),Tambien con Map
int t=1;
bool fin(int x)
{
    return A[x]==t;
}
void borrar()
{
    t++;
}
void inse(int x)
{
    A[x]=t;
}

int gcd(int a, int b)
{
    if (b == 0) return a;
    return gcd(b, a % b);
}
int extn_gcd(int m, int n)
{
    stack<tuple<int,int,int,int> >ext_gcd;
    ext_gcd.push(make_tuple(max(m,n),min(m,n),-1,-1));
    while(get<1>(ext_gcd.top())!=0)
    {
        ext_gcd.push(make_tuple(get<1>(ext_gcd.top()),get<0>(ext_gcd.top())%get<1>(ext_gcd.top()),-1,-1));

    }
    ext_gcd.pop();
    float a,b;
    int x=0,y=1;
    while(!ext_gcd.empty())
    {
        b=get<1>(ext_gcd.top());
        a=get<0>(ext_gcd.top());
        int aux=x;
        x=y-floor(a/b)*x;
        y=aux;
        ext_gcd.pop();
    }
    return max(x,y);
}
```

```
vector<pair<int,int> >equations;
int CRT()//x=b (mod m), x=a (mod n)
{
    int sol;
    if(equations.size()==1)
    {
        sol=equations[0].first%equations[0].second;
    }
    else{
    int n1=equations[0].second,n2=equations[1].second;
    int a1=equations[0].first,a2=equations[1].first;
    int s=extn_gcd(n1,n2),lcm;
    int gc=gcd(n1,n2);
    if((a1-a2)%gc!=0)//gc=gcd(n1,n2)
    {
        cout<<"no solution\n";
        return 0;
    }
    else
    {
        lcm=n1*n2/gc;
        sol=(a1+s*(a2-a1)/gc%(n2/gc)*n1)%lcm;
        if(sol<0)sol+=lcm;
        sol%=lcm;
    }
    for(int j=2;j<equations.size();j++)
    {
        a1=sol%lcm;
        n1=lcm;
        a2=equations[j].first;
        n2=equations[j].second;
        s=extn_gcd(n1,n2);
        lcm=lcm*equations[j].second/gc;
        if((a1-a2)%gc!=0)
        {
            cout<<"no solution\n";
            continue;
        }
        else
        {
            int lcm=n1*n2/gc;
            sol=(a1+s*(a2-a1)/gc%(n2/gc)*n1)%lcm;
            if(sol<0)sol+=lcm;
            sol%=lcm;
        }
    }
    }
    return sol;
}
```

5.7 GaussianElimination

```cpp
#include "../Header.cpp"

const double EPS = 1e-9;
const int INF = 2; // it doesn't actually have to be infinity or a big number

int gauss (vector < vector<double> > a, vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}

int main(){
        ios_base::sync_with_stdio(0);
    cin.tie(0);

        ll n = 2;

        vector<vl>g (2, vl(3, 0));

        g[0][0] = 1;
        g[1][1] = 1;
        g[0][2] = 1;
        g[1][2] = 2;

    // g: rows: equations, columns: x_1 * p_1 + x_2 * p_2 + x_3 * p_3 = y
        for(int i = 0; i < n-2; i++)
        {

                for(int z = i+1; z < n-1; z++)
                {
                        db mul = g[z][i] / g[i][i];
                        for(int j = 0; j < n; j++)
                                g[z][j] -= mul * g[i][j];

                }

        }
        vector<db> vals(n, 0);

        for(int i = n-2; i >= 0; i--)
        {
                db sum = g[i][n-1];
                for(int j = i+1; j < n-1; j++)

                        sum -= g[i][j] * vals[j];

                sum /= g[i][i];
                vals[i] = sum;
        }

        for(int i = 0; i < n-1; i++)
                cout << vals[i] << " ";

        cout << endl;

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

                for(int j = 0; j < n+1; j++)
                        cout << g[i][j] << " ";
                cout << endl;

        }

}
```

5.8 MathFunctions

```cpp
#include "../Header.cpp"
```

```cpp
// pre overflow
ll mul(ll x, ll y) { if (x > MX / y) return MX; return x * y; }
ll sums(ll x, ll y) { if (MX - x < y) return MX; return x + y; }

const int N = 1e5 + 10, LOG_A = 31;
ll basis[LOG_A], sz;

// O(N * LOG), base that produces the maximum
void insertVector(int mask) {
       for (ll i = LOG_A - 1; i >= 0; i--) {
              if ((mask & 1 << i) == 0) continue;

              if (!basis[i]) {
                     basis[i] = mask;
          sz++;
                     return;
              }

              mask ^= basis[i];
       }
}


struct Cientific{
  ll val;
  ll exp;
  Cientific(ll val, ll exp)
  {
    this->val = val;
    this->exp = exp;
  }

  Cientific()
  {
    val = 1;
    exp = 0;
  }

  Cientific operator*(Cientific &b)
  {
    Cientific p(val, exp);

    p.val = (p.val * b.val);
    p.exp += b.exp;

    if(p.val >= 1e9)
    {
      p.exp++;
      p.val /= 1000000000;
    }
    return p;
  }
  bool operator<(const Cientific& rhs) const {
    if(exp == rhs.exp)
      return val < rhs.val;
    return exp < rhs.exp;
  }

  bool operator==(const Cientific& rhs) const {
    return exp == rhs.exp && val == rhs.val;
  }
};

Cientific expmod2(Cientific b, ll e){
    Cientific ans(1, 0);
    while(e){
        if(e&1) ans = (ans*b);
        b = (b*b); e >>= 1;
    }
    return ans;
}


// inclusion, exclusion
ll ans = 0;
forr(bitmask, 1, (1<<n)){
    // bitmask srepresenta la interseccion actual
    bool resta = __builtin_popcount(bitmask)%2;
    ans = (ans + (resta ? 1 : M-1)*cuenta(bitmask) %M) %M;
}

// phi

repx(i,2,MAXN) if(!spf[i]) for(int j=i;j<MAXN;j+=i) if(!spf[j])spf[j]=i;

repx(i,2,MAXN){
    int x=i,res=i;
    while(x>1){
        int now=spf[x];
        while(spf[x]==now)x/=now;
        res/=now;
        res*=now-1;
    }
    phi[i]=res;
}

// Catalan number

/*
    Number of ways to place   pairs of parentheses correctly.
    Number of binary trees with   nodes.
    Number of full binary trees with  +     leaves.
    Number of ways to triangulate a convex  +     sided polygon.
*/

ll CAT[MAXN];
// ...
CAT[0] = CAT[1] = 1;
repx(i, 2, MAXN){
    CAT[i] = 0;
```

```
    rep(k, i)
        CAT[i]=(CAT[i]+CAT[k]*CAT[i-1-k]%M)%M;
}
ll F[MAXN], INV[MAXN], FI[MAXN];
ll Cat(int n){
    return F[2*n] *FI[n+1]%M *FI[n]%M;
}
// Stirling numbers

// number of ways to partition a set of n elements into k nonempty subsets

ll Stirling[MAXN][MAXN];
// ...
repx(i, 1, MAXN)Stirling[i][1] = 1;
repx(i, 2, MAXN)Stirling[1][i] = 0;
repx(i, 2, MAXN)forr(j, 2, MAXN){
    Stirling[i][j] =
    (Stirling[i-1][j-1] + j*Stirling[i-1][j]%MOD) %MOD;
}

// Bell numbers

// Number of partitions of set of n elements

// a deck of n cards is shuffled by repeatedly removing the top card and
     reinserting it anywhere in the deck (including its original position at the
     top of the deck), with exactly n repetitions
// stays the same B_n ways
// Probability B_n / n^n

// nth Bell number equals the number of permutations on n items in which no three
     values that are in sorted order have the last two of these three consecutive
ll Stirling[MAXN][MAXN], Bell[MAXN];
// ...
forn(i, MAXN){
    Bell[i] = 0;
    forn(j, MAXN)
        Bell[i] = (Bell[i] + Stirling[i][j]) %MOD;
}

//grundy
int tag[n*n];
int mex(int id)
{
    int ans = 0;
    while(tag[ans] == id) ++ans;
    return ans;
}

ll cn = 0;
for(int i = 0; i < n; i++)
{
    for(int j = 0; j < n; j++)
    {
        ll id = ++cn;

        //abajo
        for(int k = i - 1; k >= 0; k--)
            tag[grundy[k][j]] ] = id;

        //izquierda
        for(int k = j - 1; k >= 0; k--)
            tag[grundy[i][k]] ] = id;

        // diagonal
        for(int k = 1; k <= min(i, j); k++)
            tag[grundy[i-k][j-k]] ] = id;
        grundy[i][j] = mex(id);
    }
}

// fibonacci numbers
f_i = 1 / sqrt(5 * ((1 + sqrt(5)) / 2) ^ n - ((1 - sqrt(5)) / 2) ^ n);

// catalan numbers
a_n = 1 / (n+1) * comb(2n, n);
```

5.9 Matrices

```
#include "../Header.cpp"

/*
matrix A: transitions Axb
vector b(rows, 1): base case of dp
Represents last |b| states of dp

F_n
...
F2
F1
*/

// a^p = a*p mod P
// if a % p == 0 return 0

// to calculate p, can use p mod (P-1)
struct Mat {
    vector<vl> vec;
    Mat(): vec(1, vl(1, 0)) {}
    Mat(int n): vec(n, vl(n) ) {}
    Mat(int n, int m): vec(n, vl(m, 0) ) {}
    vl &operator[](int f){ return vec[f]; }
    const vl &operator[](int f) const { return vec[f]; }
    int size() const { return vec.size(); }
};
```

```
Mat operator *(Mat A, Mat B) {
    int n = A.size(), m = A[0].size(), t = B[0].size();
    Mat ans(n, t);
    rep(i, n) rep(j, t) rep(k, m)
        ans[i][j] = (ans[i][j] + A[i][k] * B[k][j] % MOD) % MOD;
    return ans;
}

Mat expmat(Mat A, ll e){
    int n = A.size();
    Mat Ans(n); rep(i, n) Ans[i][i] = 1;
    while(e){
        if(e&1) Ans = Ans*A;
        A = A*A; e >>= 1;
    }
    return Ans;
}

ll Fibo(ll n) {
    Mat V0(1, 2), T(2);
    V0[0] = {1, 1};
    T[0] = {0, 1}; T[1] = {1, 1};
    Mat V = V0*expmat(T, n);
    return V[0][0];
}
```

5.10 PrimeFactorization

```
#include "../Header.cpp"

// stores smallest prime factor for every number
int spf[MAXN];

// Calculating SPF (Smallest Prime Factor) for every
// number till MAXN.
// Time Complexity : O(nloglogn)
void sieve()
{
    spf[1] = 1;
    for (int i=2; i<MAXN; i++)

        // marking smallest prime factor for every
        // number to be itself.
        spf[i] = i;

    // separately marking spf for every even
    // number as 2
    for (int i=4; i<MAXN; i+=2)
        spf[i] = 2;

    for (int i=3; i*i<MAXN; i++)
    {
        // checking if i is prime
        if (spf[i] == i)
        {
            // marking SPF for all numbers divisible by i
            for (int j=i*i; j<MAXN; j+=i)

                // marking spf[j] if it is not
                // previously marked
                if (spf[j]==j)
                    spf[j] = i;
        }
    }
}

// A O(log n) function returning primefactorization
// by dividing by smallest prime factor at every step
vector<int> getFactorization(int x)
{
    vector<int> ret;
    while (x != 1)
    {
        ret.push_back(spf[x]);
        x = x / spf[x];
    }
    return ret;
}

void primeFactors(ll n)
{
    while (n % 2 == 0)
    {
        cout << 2 << " ";
        n = n/2;
    }

    for (int i = 3; i <= sqrt(n); i = i + 2)
    {
        while (n % i == 0)
        {
            cout << i << " ";
            n = n/i;
        }
    }

    if (n > 2)
        cout << n << " ";
}
```

5.11 Simplex

```
#include "../Header.cpp"
```

```cpp
#define fore(i,a,b) for(int i=a,ThxDem=b;i<ThxDem;++i)

namespace Simplex {
vector<int> X,Y;
vector<vector<db> > A;
vector<db> b,c;
db z;
int n,m;
void pivot(int x,int y){
	swap(X[y],Y[x]);
	b[x]/=A[x][y];
	fore(i,0,m)if(i!=y)A[x][i]/=A[x][y];
	A[x][y]=1/A[x][y];
	fore(i,0,n)if(i!=x&&abs(A[i][y])>EPS){
		b[i]-=A[i][y]*b[x];
		fore(j,0,m)if(j!=y)A[i][j]-=A[i][y]*A[x][j];
		A[i][y]=-A[i][y]*A[x][y];
	}
	z+=c[y]*b[x];
	fore(i,0,m)if(i!=y)c[i]-=c[y]*A[x][i];
	c[y]=-c[y]*A[x][y];
}
pair<db,vector<db> > simplex( // maximize c^T x s.t. Ax<=b, x>=0
		vector<vector<db> > _A, vector<db> _b, vector<db> _c){
	// returns pair (maximum value, solution vector)
	A=_A;b=_b;c=_c;
	n=b.size();m=c.size();z=0.;
	X=vector<int>(m);Y=vector<int>(n);
	fore(i,0,m)X[i]=i;
	fore(i,0,n)Y[i]=i+m;
	while(1){
		int x=-1,y=-1;
		db mn=-EPS;
		fore(i,0,n)if(b[i]<mn)mn=b[i],x=i;
		if(x<0)break;
		fore(i,0,m)if(A[x][i]<-EPS){y=i;break;}
	if(y<0) return(make_pair(-1, b));
		assert(y>=0); // no solution to Ax<=b
		pivot(x,y);
	}
	while(1){
		db mx=EPS;
		int x=-1,y=-1;
		fore(i,0,m)if(c[i]>mx)mx=c[i],y=i;
		if(y<0)break;
		db mn=INF;
		fore(i,0,n)if(A[i][y]>EPS&&b[i]/A[i][y]<mn)mn=b[i]/A[i][y],x=i;
		assert(x>=0); // c^T x is unbounded
		pivot(x,y);
	}
	vector<db> r(m);
	fore(i,0,n)if(Y[i]<m)r[Y[i]]=b[i];
	return {z,r};
}
}
```

```cpp
int main(){
	ios_base::sync_with_stdio(0);
    cin.tie(0);

    ll n, k, x;
    db y;
    cin >> n >> k >> x;

    vector<db> b, c;
    vector<vector<db> > A;

    for(int i = 0; i < n; i++)
    {
        cin >> y;
        c.push_back(y);
    }

    vector<db>aux(n, 0);
    for(int i = 0; i < k; i++)aux[i] = -1;
    A.push_back(aux);
    b.push_back(-1.);

    for(int i = k; i < n; i++)
    {
        aux[i - k] = 0;
        aux[i] = -1;
        A.push_back(aux);
        b.push_back(-1);
    }
    aux.assign(n, 0);
    for(int i = 0; i < n; i++)
    {
        aux[i] = 1;
        A.push_back(aux);
        b.push_back(1);
        aux[i] = 0;
    }
    aux.assign(n, 1);
    A.push_back(aux);
    b.push_back(x);
    ll in = 0;
    /*for(auto it : A)
    {
        for(auto it2: it)cout<<it2<<" ";
        cout<<" "<<b[in];
        in++;
        cout<<"*\n";
    }*/

    db mx = Simplex::simplex(A,b,c).first;
    cout<<(long long)mx<<"\n";

    cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
    return 0;
```

```
}
```

6 Other

6.1 AdHoc

```
#include "../Header.cpp"

int joseph(int n,int m){
    int Result=0;
    for(int i=1;i<=n;i++){
        Result=(Result+m-1)%i+1;
    }
    return(Result);
}
int joseph(int n,int m){
    vl a(n+1, 0);
    //see eliminated
    bool o = 1;
    for(int i = 0; i < n/2; i++){
        a[i+1] = (a[i] + m-1)%(n-i);
        if(a[i+1] < n/2){
            o = 0;
            break;
        }
    }
}
// if k = 2
// move first significant bit to right
int joseph(ll n){
    ll bit = 62;
    while(!(n & (1 << bit))){
        bit--;
    }
    n &= ~(1 << bit);
    return 1 + (n << 1);
}

// matching in DAG gives min number of paths to cover all nodes
// Dilword
// matching in transitive DAG gives max independent set

// primes in a n size range n / log(n)

// nim game
// a_1 ^ a_2 ^ ... ^ a_n = 0: player 1 lose

// nim variation: remove stones from [0, k] piles
// a_1 ^_k+1 a_2 ^_k+1 ... ^_k+1 a_n = 0: player 1 lose
// ^_k+1 = xor mod (k+1) k+1 bits = 0 mod (k+1)
// sum of pairs
// a*b + b*c + c*a

//(a + b + c + d)^2 - (a^2 + b^2 + c^2)

// valor{x} = (x, 0)
//combinar((s_1, p_1), (s_2, p_2)) = ((s_1 + s_2),(p_1 + p_2 + s_1 * s_2))

// sum of subconj
// 1 + a + b + c + a*b + b*c + c*a + a*b*c
//(1 + a)*(1 + b)*(1 + c)

// valor{x} = 1 + x
//combinar(a, b) = a*b

// x >= y -> x mod y < x/2 counting decimals

/*Para un arbol de tamao N, solo hay un arbol para cada divisor(N)
de tamao divisor(N) que lo puede armar solo consigo mismo

Para hashear un arbol se usan parentesis, el hash es distinto para cada root,
hay que ordenar los hijos antes de hashear



everyone loses their hats all at once, and each person puts on a random hat;
in expectation, how many people get their own hats back?
The probability that the each person gets their own hat is 1/N,
and then by linearity of expectation,
the total number of instances of someone getting their own hat is 1/N*N=1.

expeted value to two people will get their original hat : 1/2
for 3: 1/3

*/

// Modular sum optimization
if (R >= MOD) R -= MOD;


/*
euler cycle
all vertex with even degree


hamiltonian cycle
d(v) >= n/2 vertex degree


exact partition O(3^(m/3)) O(2^(m/2))
m(4) sets and n(3) objects
101 -
010 -
110
011

for each i in n:
    choose a row with bit i on
```

```
    erase all rows with bit i on
    continue
*/

// convex hull of max max(X, Y)^(2/3) points in recatangle (0, 0) (X, Y)
```

6.2 Line input

```
#include "Header.cpp"

int main()
{
    // save strings separated by space in a line
    string line, token;
    getline(cin, line);
    stringstream ss(line);
    while(ss >> token)
    {
        cout << token << "\n";
    }
        return 0;
}
```

6.3 NextGreaterLower

```
#include "../Header.cpp"
int main(){

        ios_base::sync_with_stdio(0);
        cin.tie(0);
  ll n;
        cin >> n;

  vl c(n);
  // next value with lower/grater value
  // right greater, left greater, right lower, left lower
  vl Rg(n, n), Lg(n, -1), Rl(n, n), Ll(n, -1);

  rep(i, n){
    cin >> c[i];
  }

  stack<ll> Sg, Sl;
  rep(i, n){
    while(!Sg.empty() && c[Sg.top()] < c[i]){
      Rg[Sg.top()] = i;
      Sg.pop();
    }
    Sg.push(i);
    while(!Sl.empty() && c[Sl.top()] > c[i]){
      Rl[Sl.top()] = i;
      Sl.pop();
    }
    Sl.push(i);
  }
  while(!Sg.empty()) Sg.pop();
  while(!Sl.empty()) Sl.pop();

  for(int i = n-1; i >= 0; i--){
    while(!Sg.empty() && c[Sg.top()] <= c[i]){
      Lg[Sg.top()] = i;
      Sg.pop();
    }
    Sg.push(i);

    while(!Sl.empty() && c[Sl.top()] > c[i]){
      Ll[Sl.top()] = i;
      Sl.pop();
    }
    Sl.push(i);
  }
  cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
  return 0;
}
```

7 String

7.1 AhoCorasick

```
#include "../Header.cpp"

//https://github.com/mhunicken/icpc-team-notebook-el-vasito/blob/master/strings/aho_coras
struct vertex {
        map<char,int> next,go;
        int p,link;
        char pch;
        vector<int> leaf;
        vertex(int p=-1, char pch=-1):p(p),pch(pch),link(-1){}
};
vector<vertex> t;
void aho_init(){ //do not forget!!
        t.clear();t.pb(vertex());
}
void add_string(string s, int id){
        int v=0;
        for(char c:s){
                if(!t[v].next.count(c)){
                        t[v].next[c]=t.size();
                        t.pb(vertex(v,c));
                }
```

```
                v=t[v].next[c];
        }
        t[v].leaf.pb(id);
}
int go(int v, char c);
int get_link(int v){
        if(t[v].link<0)
                if(!v||!t[v].p)t[v].link=0;
                else t[v].link=go(get_link(t[v].p),t[v].pch);
        return t[v].link;
}
int go(int v, char c){
        if(!t[v].go.count(c))
                if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
                else t[v].go[c]=v==0?0:go(get_link(v),c);
        return t[v].go[c];
}
```

7.2 Hashing

```
#include "../Header.cpp"
//https://github.com/BenjaminRubio/CompetitiveProgramming/blob/master/Material/Strings/Hash.cpp
struct RH
{
    // choose base B random to avoid hacks 33 37 41
    // randomize V(s[i])
    int B = 1777771, M[2] = {999727999, 1070777777}, P[2] = {325255434, 10018302};
    vl H[2], I[2];
    RH(string &s)
    {
        int N = s.size(); rep(k, 2)
        {
            H[k].resize(N + 1), I[k].resize(N + 1);
            H[k][0] = 0, I[k][0] = 1; ll b = 1;
            rep(i, N + 1) if (i)
            {
                H[k][i] = (H[k][i - 1] + b * s[i - 1]) % M[k];
                I[k][i] = (1LL * I[k][i - 1] * P[k]) % M[k];
                b = (b * B) % M[k];
            }
        }
    }
    ll get(int l, int r) // inclusive - exclusive
    {
        ll h0 = (H[0][r] - H[0][l] + M[0]) % M[0];
        h0 = (1LL * h0 * I[0][l]) % M[0];
        ll h1 = (H[1][r] - H[1][r] + M[1]) % M[1];
        h1 = (1LL * h1 * I[1][l]) % M[1];
        return (h0 << 32) | h1;
    }
};
bool compare(int a, int b)
{

    ll l = 0, r = n-1, p, res = -1;
    while(l <= r)
    {
        p = (l + r) / 2;
        if(rhs[a].get(0, p) == rhs[b].get(0, p))l = p+1;
        else {
            res = p;
            r = p-1;
        }
    }
    if(res == -1)return a < b;
    return s[a][res] < s[b][res];

}

//Suffix Array O(N log^2 N)
rep(n) sa[i] = i;
sort(ALL(sa), compare)
```

7.3 KMP

```
#include "../Header.cpp"


// FROM:
//https://github.com/PabloMessina/Competitive-Programming-Material/blob/master/Strings/KMP

// Build longest proper prefix/suffix array (lps) for pattern
// lps[i] = length of the longest proper prefix which is also suffix in pattern[0
    .. i]
void init_lps(string& s, int lps[]) {
    int n = s.size();
    lps[0] = 0; // base case: no proper prefix/suffix for pattern[0 .. 0] (length
        1)
    repx(j, 1, n) { // for each s[0 .. j]
        int i = lps[j-1]; // i points to the char next to lps of previous iteration
        while (s[i] != s[j] and i > 0) i = lps[i-1];
        lps[j] = s[i] == s[j] ? i+1 : 0;

        //optimization to reutilice the lps in O(n)
        if(i > 0 && s[i] == s[lps[i-1]] && lps[i-1] != 0) lps[i-1] =
            lps[lps[i-1]-1];
    }
}

// Count number of matches of pattern string in target string using KMP algorithm
int kmp(string& s, string& t) {
    int n = s.size(), m = t.size();
    int lps[n];
    init_lps(s, lps); // build lps array
```

```
    int matches = 0;
    int i = 0; // i tracks current char in pattern to compare
    rep(j, m) { // j tracks each char in target to compare
        // try to keep prefix before i as long as possible while ensuring i
            matches j
        while (s[i] != t[j] && i > 0) i = lps[i-1];
        if (s[i] == t[j]) {
            if (++i == n) { // we matched the whole pattern
                i = lps[n-1]; // shift the pattern so that the longest proper
                    prefix/suffix pair is aligned

                                matches++;

            }
        }
    }
    return matches;
}

int main() { // usage
    string target, pattern;
    while (true) {
        cin >> target >> pattern;
        cout << kmp(pattern, target) << " matches\n";
    }
    return 0;
}
```

7.4 LongestCommonSubsequence

```
//https://www.geeksforgeeks.org/
#include "../Header.cpp"


int lcs(strint& X, string& Y, int m, int n)
{
   if (m == 0 || n == 0)
     return 0;
   if (X[m-1] == Y[n-1])
     return 1 + lcs(X, Y, m-1, n-1);
   else
     return max(lcs(X, Y, m, n-1), lcs(X, Y, m-1, n));
}


int main()
{
  string X = "AGGTAB", Y = "GXTXAYB";

  int m = X.size();
  int n = Y.size();

  cout << "Length of LCS is" << lcs(X, Y, m, n) << "\n";

  return 0;
}
```

7.5 LongestCommonSubstring

```
//https://www.geeksforgeeks.org/
#include "../Header.cpp"



int LCSubStr(string& X, string& Y, int m, int n)
{
    // Create a table to store lengths of longest common suffixes of
    // substrings. Notethat LCSuff[i][j] contains length of longest
    // common suffix of X[0..i-1] and Y[0..j-1]. The first row and
    // first column entries have no logical meaning, they are used only
    // for simplicity of program
    int LCSuff[m+1][n+1];
    int result = 0; // To store length of the longest common substring

    /* Following steps build LCSuff[m+1][n+1] in bottom up fashion. */
    for (int i=0; i<=m; i++)
    {
        for (int j=0; j<=n; j++)
        {
            if (i == 0 || j == 0)
                LCSuff[i][j] = 0;

            else if (X[i-1] == Y[j-1])
            {
                LCSuff[i][j] = LCSuff[i-1][j-1] + 1;
                result = max(result, LCSuff[i][j]);
            }
            else LCSuff[i][j] = 0;
        }
    }
    return result;
}

int main()
{
    string X = "OldSite:GeeksforGeeks.org", Y = "NewSite:GeeksQuiz.com";


    int m = X.size();
    int n = Y.size();

    cout << "Length of Longest Common Substring is " << LCSubStr(X, Y, m, n) <<
         "\n";
    return 0;
```

```
}
```

7.6 Manacher

```
#include "../Header.cpp"

vl manacher(string& s)
{
    int n = s.size();
    // string con # entre medio (2n - 1)
    vl lps(n);
    int l = 0, r = 0, c = 0;
    rep(i, n)
    {
        int j = l+(r-i);
        lps[i] = min(r-i, (int)lps[j]);
        while(i - lps[i] >= 0 && i+lps[i] < n &&
               s[i-lps[i]] == s[i+lps[i]]) lps[i]++;

        // acutalizar l, r
        if(r < i + lps[i])
        {
            l = i - lps[i];
            r = i + lps[i];
        }
    }
    // returns total size for each index
    return lps;
}

int main()
{
  ios::sync_with_stdio(0);
  cin.tie(0);
  ll n = 5;
  string s = "aaaaa", s2 = "a#a#a#a#a";

1
2
3
2
1

0
1
2
2
1

  // d1 -> number of expansions
  vector<int> d1(n);
  for (int i = 0, l = 0, r = -1; i < n; i++) {
      int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
      while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
          k++;
      }
      d1[i] = k--;
      if (i + k > r) {
          l = i - k;
          r = i + k;
      }
  }
  // evens, start at index 1, right -> aaAa
  vector<int> d2(n);
  for (int i = 0, l = 0, r = -1; i < n; i++) {
      int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
      while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
          k++;
      }
      d2[i] = k--;
      if (i + k > r) {
          l = i - k - 1;
          r = i + k ;
      }
  }
  for(auto it : d1)cout<<it<<"\n";
  cout<<"\n";
  for(auto it : d2)cout<<it<<"\n";
  cout<<"\n";
  //vl t = manacher(s2);
  //for(auto it : t)cout<<it<<"\n";
  cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
    return 0;
}
```

7.7 PalindromicTree

```
#include "../Header.cpp"

#define MAXN 2000005
ll M = 51123987;
struct Node
{
    // store start and end indexes of current
    // Node inclusively
    // only for first ocurrence
    ll start, end;

    // stores length of substring
    ll length;

    // stores insertion Node for all characters a-z
    ll insertEdg[26];
```

```
    // stores the Maximum Palindromic Suffix Node for
    // the current Node
    ll suffixEdg;

    ll depht;
};

// two special dummy Nodes as explained above
Node root1, root2;

// stores Node information for constant time access
Node tree[MAXN];

// Keeps track the current Node while insertion
ll currNode;
string s;
ll ptr;

void insert(ll idx)
{
//STEP 1//

    /* search for Node X such that s[idx] X S[idx]
       is maximum palindrome ending at position idx
       iterate down the suffix link of currNode to
       find X */
    ll tmp = currNode;
    while (true)
    {
        ll curLength = tree[tmp].length;
        if (idx - curLength >= 1 and s[idx] == s[idx-curLength-1])
            break;
        tmp = tree[tmp].suffixEdg;
    }

    /* Now we have found X ....
     * X = string at Node tmp
     * Check : if s[idx] X s[idx] already exists or not*/
    if(tree[tmp].insertEdg[s[idx]-'a'] != 0)
    {
        // s[idx] X s[idx] already exists in the tree
        currNode = tree[tmp].insertEdg[s[idx]-'a'];
        return;
    }

    // creating new Node
    ptr++;

    // making new Node as child of X with
    // weight as s[idx]
    tree[tmp].insertEdg[s[idx]-'a'] = ptr;

    // calculating length of new Node
    tree[ptr].length = tree[tmp].length + 2;

    // updating end point for new Node
    tree[ptr].end = idx;

    // updating the start for new Node
    tree[ptr].start = idx - tree[ptr].length + 1;



//STEP 2//

    /* Setting the suffix edge for the newly created
       Node tree[ptr]. Finding some String Y such that
       s[idx] + Y + s[idx] is longest possible
       palindromic suffix for newly created Node.*/

    tmp = tree[tmp].suffixEdg;

    // making new Node as current Node
    currNode = ptr;
    if (tree[currNode].length == 1)
    {
        // if new palindrome's length is 1
        // making its suffix link to be null string
        tree[currNode].suffixEdg = 2;
        tree[currNode].depht = 1;
        return;
    }
    while (true)
    {
        ll curLength = tree[tmp].length;
        if (idx-curLength >= 1 and s[idx] == s[idx-curLength-1])
            break;
        tmp = tree[tmp].suffixEdg;
    }

    // Now we have found string Y
    // linking current Nodes suffix link with s[idx]+Y+s[idx]
    tree[currNode].suffixEdg = tree[tmp].insertEdg[s[idx]-'a'];
    tree[currNode].depht = tree[tree[tmp].insertEdg[s[idx]-'a']].depht + 1;

}
// para ir al revez currNode = 1, reverse(s)
// para un string nuevo, devolver currNode o currNode = 1

// driver program
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    // initializing the tree
    root1.length = -1;
    root1.suffixEdg = 1;
    root2.length = 0;
    root2.suffixEdg = 1;
    root1.depht = 0;
```

```
    root2.depht = 0;

    tree[1] = root1;
    tree[2] = root2;
    ptr = 2;
    currNode = 1;
    ll l;
    cin >> l;
    cin >> s;
    l = s.length();
    cout<<l<<endl;
    vl sums(l+1, 0);
    ll ans = 0;

    for (ll i=0; i<l; i++){
        insert(i);
        ll nod = currNode, depht = 0;
        while(tree[nod].length > 0)
        {
          nod = tree[nod].suffixEdg;
          depht++;
        }
        nod = currNode;
        sums[i+1]=(depht + sums[i])%M;
        depht--;
        while(tree[nod].length > 1)
        {
          ans += (depht + sums[i] - sums[i - tree[nod].length + 1])%M;
          ans %= M;

          nod = tree[nod].suffixEdg;
          depht--;
        }

    }
    cout<<ans<<"\n";

    // printing all of its distinct palindromic
    // substring
    cout << "All distinct palindromic substring for "
         << s << " : \n";
    for (int i=3; i<=ptr; i++)
    {
        cout << i-2 << ") ";
        for (int j=tree[i].start; j<=tree[i].end; j++)
            cout << s[j];
        cout << endl;
    }


  cerr << "\nTime elapsed: " << 1000 * clock() / CLOCKS_PER_SEC << "ms\n";
  return 0;
}
```

7.8 SuffixArray

```
// ========================================================
// Suffix Array Construction : Prefix Doubling + Radix Sort
// ========================================================
// Complexity: O(N*log(N))
// references: https://www.cs.helsinki.fi/u/tpkarkka/opetus/10s/spa/lecture11.pdf
// https://youtu.be/_TUeAdu-U_k
#include "../Header.cpp"
#define invrep(i,b,a) for(int i = b; i >= a; --i)


struct SA {
    int n; vl counts, rank, rank_, sa, sa_, lcp; // lcp is optional
    inline int gr(int i) { return i < n ? rank[i]: 0; }
    void csort(int maxv, int k) {
        counts.assign(maxv+1, 0);
        repx(i,0,n) counts[gr(i+k)]++;
        repx(i,1,maxv+1) counts[i] += counts[i-1];
        invrep(i,n-1,0) sa_[--counts[gr(sa[i]+k)]] = sa[i];
        sa.swap(sa_);
    }
    void get_sa(vl& s) {
        repx(i,0,n) sa[i] = i;
        sort(sa.begin(), sa.end(), [&s](int i, int j) { return s[i] < s[j]; });
        int r = rank[sa[0]] = 1;
        repx(i,1,n) rank[sa[i]] = (s[sa[i]] != s[sa[i-1]]) ? ++r : r;
        for (int h=1; h < n and r < n; h <<= 1) {
            csort(r, h); csort(r, 0); r = rank_[sa[0]] = 1;
            repx(i,1,n) {
                if (rank[sa[i]] != rank[sa[i-1]] or
                    gr(sa[i]+h) != gr(sa[i-1]+h)) ++r;
                rank_[sa[i]] = r;
            } rank.swap(rank_);
        }
    }
    // LCP construction in O(N) using Kasai's algorithm
    // reference: https://codeforces.com/blog/entry/12796?#comment-175287
    void get_lcp(vl& s) { // lcp is optional
        lcp.assign(n, 0); int k = 0;
        repx(i,0,n) {
            int r = rank[i]-1;
            if (r == n-1) { k = 0; continue; }
            int j = sa[r+1];
            while (i+k<n and j+k<n and s[i+k] == s[j+k]) k++;
            lcp[r] = k;
            if (k) k--;
        }
    }
    SA(vl& s) {
        n = s.size();
        rank.resize(n); rank_.resize(n);
        sa.resize(n); sa_.resize(n);
        get_sa(s); get_lcp(s); // lcp is optional
```

```
    }
};

int main() { // how to use
    string test; cin >> test;
    vl s;
    for (char c : test) s.push_back(c);
    SA sa(s);
    for (int i : sa.sa) cout << i << ":\t" << test.substr(i) << '\n';
    repx(i,0,s.size()) {
        printf("LCP between %d and %d is %d\n", i, i+1, sa.lcp[i]);
    }
}
```

7.9 Trie

```
#include "../Header.cpp"

//https://github.com/BenjaminRubio/CompetitiveProgramming/blob/master/Material/Strings/Trie.cpp

const int MAX = 2e5;
struct Trie
{

  int c = 0;
  vector<vector<int>> N;
  vector<int> S;

  Trie()
  {
    N.assign(MAX, vector<int>(26, 0));
    S.assign(MAX, 0);
    c = 0;
  }

  void add(string s, int a = 1)
  {
    int p = 0; S[p] += a;
    for (char l : s)
    {
      int t = l - 'a';
      if(!N[p][t]) N[p][t] = ++c;
      S[p = N[p][t]] += a;
    }
  }
};

struct TrieXOR
{
    static const int MAX = 1e6;
    int N[MAX][2] = {0}, S[MAX] = {0}, c = 0;
    void add(int x, int a = 1)
    {
        int p = 0; S[p] += a;
        rep(i, 31)
        {
            int t = (x >> (30 - i)) & 1;
            if (!N[p][t]) N[p][t] = ++c;
            S[p = N[p][t]] += a;
        }
    }
    int get(int x)
    {
        if (!S[0]) return -1;
        int p = 0; rep(i, 31)
        {
            int t = ((x >> (30 - i)) & 1) ^ 1;
            if (!N[p][t] || !S[N[p][t]]) t ^= 1;
            p = N[p][t]; if (t) x ^= (1 << (30 - i));
        }
        return x;
    }
};

struct Trie {
    vector<vector<int>> g;
    vector<int> count;
    int vocab;
    Trie(int vocab, int maxdepth = 10000) : vocab(vocab) {
        g.reserve(maxdepth);
        g.emplace_back(vocab, -1);
        count.reserve(maxdepth);
        count.push_back(0);
    }
    int move_to(int u, int c) {
        assert (0 <= c and c < vocab);
        int& v = g[u][c];
        if (v == -1) {
            v = g.size();
            g.emplace_back(vocab, -1);
            count.push_back(0);
        }
        count[v]++;
        return v;
    }
    void insert(const string& s, char ref = 'a') { // insert string
        int u = 0; for (char c : s) u = move_to(u, c - ref);
    }
    void insert(vector<int>& s) { // insert vector<int>
        int u = 0; for (int c : s) u = move_to(u, c);
    }
    db query(const string& s, char ref = 'a')
    {
      int u = 0;
      db cost = 0;
      for (char c : s){
        ll co = 0;
```

```
        for(auto it : g[u]) if(it != -1)co++;

        ll nex = move_to(u, c - ref);
        if(u == 0 || co > 1 || count[u] != count[nex]) cost++;
        u = nex;

      }
      return cost;
    }
    ll dfs(int u, int depht)
    {
        ll ans = INF;
        if(count1[u] == 1 && count2[u] == 1)ans = depht;
        for(int i = 0; i < 26; i++)
        {
            if(g[u][i] != -1) ans = min(ans, dfs(g[u][i], depht + 1));
        }
        return ans;
    }
    int size() { return g.size(); }
};

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    ll n;
    while(cin >> n){
      string s;
      vector<string > c;
      Trie trie(26);
      for(int i = 0; i < n; i++)
      {
        cin >> s;
        c.push_back(s);
        trie.insert(s);
      }
      db sum = 0;
      for(int i = 0; i < n; i++)
      {
        sum += trie.query(c[i]);
      }



      cout<<fixed<<setprecision(2)<< sum / db(n) << "\n";
    }
}
```