Team Notebook

Doggy Sweat - Shahid Beheshti University

December 11, 2018

Contents

1 DP

1.1 Knuth Optimization

```cpp
for (int s = 0; s<=k; s++)              //s - length(size)
        of substring
    for (int L = 0; L+s<=k; L++) {       //L - left point
      int R = L + s;                     //R - right point
      if (s < 2) {
        res[L][R] = 0;                   //DP base -
              nothing to break
        mid[L][R] = l;                   //mid is equal to
              left border
        continue;
      }
      int mleft = mid[L][R-1];           //Knuth's trick:
              getting bounds on M
      int mright = mid[L+1][R];
      res[L][R] = 1000000000000000000LL;
      for (int M = mleft; M<=mright; M++) { //iterating for M
              in the bounds only
        int64 tres = res[L][M] + res[M][R] + (x[R]-x[L]);
        if (res[L][R] > tres) {          //relax current
              solution
          res[L][R] = tres;
          mid[L][R] = M;
        }
      }
    }
  int64 answer = res[0][k];
```

2 Data Structures

2.1 Fenwick Tree

```cpp
#define LSOne(S) (S & (-S))
vector<int> fen;
void ft_create(int n) { fen.assign(n + 1, 0); }
// initially n + 1 zeroes
int ft_rsq(int b)
{ // returns RSQ(1, b)
 int sum = 0;
 for (; b; b -= LSOne(b)) sum += fen[b];
 return sum;
}
int ft_rsq(int a, int b)
{ // returns RSQ(a, b)
 return ft_rsq(b) - (a == 1 ? 0 : ft_rsq(a - 1));
}
// adjusts value of the k-th element by v (v can be +ve/inc
    or -ve/dec).
void ft_adjust(int k, int v)
{
 for (; k < (int)fen.size(); k += LSOne(k)) fen[k] += v;
}
```

2.2 Ranged Fenwick Tree

```cpp
//not tested
#define LSOne(S) (S & (-S))
vector<ll> fen[2];
void ft_create(int n) { fen[0].assign(n + 1, 0), fen[1].
    assign(n + 1, 0); }
ll ft_rsq(int b, bool bl)
{
 ll sum = 0;
 for (; b; b -= LSOne(b)) sum += fen[bl][b];
 return sum;
}
void ft_adjust(ll k, ll v, bool bl)
{
 for (; k <= (int)fen[bl].size(); k += LSOne(k)) fen[bl][k]
     += v;
}
void range_adjust(ll l, ll r, ll v)
{
 ft_adjust(l, v, 0), ft_adjust(r+1, -v, 0);
 ft_adjust(l, v*(l-1), 1), ft_adjust(r+1, -v*r, 1);
}
ll range_rsq(ll l, ll r)
{
 ll x=0, a, b;
 if(l)
 {
  a=ft_rsq(l-1, 0), b=ft_rsq(l-1, 1);
  x=(l-1)*a-b;
 }
 a=ft_rsq(r, 0), b=ft_rsq(r, 1);
 ll y=r*a-b;
 return y-x;
}
```

2.3 Sparse Table

```cpp
int n, arr[MAXN], table[MAXN][20], table2[MAXN][20];
int Log[MAXN];
vector<int> tmp, best;
int GCD(int a, int b)
{
 if(b==0) return a;
 return GCD(b, a%b);
}
void init(int n)
{
 int mx=0, l=0, r=n+1, lim=2, lg=0;
 Rep(i, MAXN)
 {
  if(i==lim)
  {
   lg++;
   lim*=2;
  }
  Log[i]=lg;
 }
 Rep(i, n) table[i][0]=table2[i][0]=arr[i];
 For(i, 1, Log[n]+1) for(int j=0; j+(1<<i)-1< n; j++)
 {
  table[j][i]=GCD(table[j][i-1], table[j+(1<<(i-1))][i-1]);
  table2[j][i]=min(table2[j][i-1], table2[j+(1<<(i-1))][i
      -1]);
 }
}
int get_gcd(int l, int r)
{
 int lg=Log[r-l+1];
 return GCD(table[l][lg], table[r-(1<<lg)+1][lg]);
}
```

3 Games

3.1 Declare Winner

```cpp
void declareWinner(int whoseTurn, int piles[], int Grundy[],
     int n) {
    int xorValue = Grundy[piles[0]];
    for (int i=1; i<=n-1; i++)
        xorValue = xorValue ^ Grundy[piles[i]];
    if (xorValue != 0) {
        if (whoseTurn == PLAYER1)
            printf("Player 1 will win\n");
        else
```

```
        printf("Player 2 will win\n");
    }
    else {
        if (whoseTurn == PLAYER1)
            printf("Player 2 will win\n");
        else
            printf("Player 1 will win\n");
    }
    return;
}
```

3.2 Grundy

```
// A function to Compute Grundy Number of 'n'
// Only this function varies according to the game
int calculateGrundy(int n) {
    if (n == 0)
        return (0);
    unordered_set<int> Set; // A Hash Table
    for (int i=0; i<=n-1; i++)
        Set.insert(calculateGrundy(i));
    return (calculateMex(Set));
}
```

3.3 Mex

```
int calculateMex(unordered_set<int> Set) {
    int Mex = 0;
    while (Set.find(Mex) != Set.end())
        Mex++;
    return (Mex);
}
```

4 Geometry

4.1 Angle Bisector

```
// angle bisector
int bcenter( PT p1, PT p2, PT p3, PT& r ){
 if( triarea( p1, p2, p3 ) < EPS ) return -1;
 double s1, s2, s3;
 s1 = dist( p2, p3 );
 s2 = dist( p1, p3 );
 s3 = dist( p1, p2 );
```

```
 double rt = s2/(s2+s3);
 PT a1,a2;
 a1 = p2*rt+p3*(1.0-rt);
 rt = s1/(s1+s3);
 a2 = p1*rt+p3*(1.0-rt);
 intersection( a1,p1, a2,p2, r );
 return 0;
}
```

4.2 Circle Circle Intersection

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

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

4.3 Circle Line Intersection

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

```
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}
```

4.4 Circle from Three Points

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

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

4.5 Closest Point on Line

```
//From In 1010101 We Trust cheatsheet:
//the closest point on the line p1->p2 to p3
void closestpt( PT p1, PT p2, PT p3, PT &r ){
 if(fabs(triarea(p1, p2, p3)) < EPS){ r = p3; return; }
 PT v = p2-p1; v.normalize();
 double pr; // inner product
 pr = (p3.y-p1.y)*v.y + (p3.x-p1.x)*v.x;
 r = p1+v*pr;
}
```

4.6 Delaunay Triangulation

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

```cpp
//                    corresponding to triangle vertices

typedef double T;

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

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

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

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

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

    //expected: 0 1 3
    //          0 3 2
    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

4.7   Latitude and Longitude

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

using namespace std;

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

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

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

  return Q;
}

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

  return P;
}

int main()
{
  rect A;
  ll B;

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

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

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

4.8   Line Intersection

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

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

```cpp
    pointd from, to;
    LineSegment(const pointd& a, const pointd& b) {
        A = b.Y - a.Y;
        B = a.X - b.X;
        C = A*a.X + B*a.Y;
        from = a;
        to = b;
    }

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

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

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

4.9   Point in Polygon

```cpp
// determine if point is in a possibly non-convex polygon (
    by William
// Randolph Franklin); returns 1 for strictly interior
    points, 0 for
// strictly exterior points, and 0 or 1 for the remaining
    points.
// Note that it is possible to convert this into an *exact*
    test using
```

```cpp
// integer arithmetic by taking care of the division
    appropriately
// (making sure to deal with signs properly) and then by
    writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
    p[j].y <= q.y && q.y < p[i].y) &&
    q.x < p[j].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[
        j].y - p[i].y))
      c = !c;
  }
  return c;
}
```

4.10   Polygon Centroid

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

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

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

4.11   Rotation Around Origin by t

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

5   Graph

5.1   2-SAT

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

```
            memcpy(color, bc, sizeof color);
            if(!dfs( i, 1 ))
                return false;
        }
    }
}
    return true;
}
```

5.2   Bidirectional Min Cost

```
define MAX_V 1+2*100
#define MAX_E 2*10001
typedef long long edge_type;
struct edge
{
 int start, to;
 ll cap, cost;
 edge(int _s, int _d, ll _c, ll _co)
 {
   start=_s, to=_d, cost=_co, cap=_c;
 }
 edge(){}
};
const edge_type INF = 1ll<<60;
int V,E,prevee[MAX_V],last[MAX_V];
edge_type flowVal, flowCost, pot[MAX_V], dist[MAX_V];
vector<int> adj[MAX_V];
vector<edge> yal;
void add(edge b)
{
 yal.push_back(b);
 adj[b.start].push_back(yal.size()-1);
 swap(b.start, b.to);
 b.cost*=-1, b.cap=0;
 yal.push_back(b);
 adj[b.start].push_back(yal.size()-1);
}
bool Bellman_Ford (int s)
{
 bool f;
 Rep(i, V+1) pot[i]=INF;
 pot[s]=0;
 for(int i=1;i<V;i++)
 {
  f=0;
  for(int j=0;j<yal.size();j++)
  {
   int k1=yal[j].start, k2=yal[j].to,w=yal[j].cost;
```

```
  if(pot[k2]>w+pot[k1])
  {
   pot[k2]=w+pot[k1];
   f=1;
  }
 }
}
 if(f==0)
  break;
}
 for(int i=0;i<yal.size();i++)
 {
  int k1=yal[i].start,k2=yal[i].to,w=yal[i].cost;
  if(pot[k2]>w+pot[k1])
   return 0;
 }
 return 1;
}
void mcmf(int s, int t){
 flowVal = flowCost = 0;
 memset(pot,0,sizeof(pot));
 Bellman_Ford(s);
 while(true){
  for(int i = 0;i<V;++i) dist[i] = INF, prevee[i]=-1;
  priority_queue<pair<ll, ll> > q;
  q.push(MP(0, s));
  dist[s] = prevee[s] = 0;

  while(!q.empty()){
   int aux = q.top().second; q.pop();

   for(int i = 0;i<adj[aux].size(); i++){
    int e=adj[aux][i];
    if(yal[e].cap<=0) continue;

    edge_type new_dist = dist[aux]+yal[e].cost+pot[aux]-pot[
        yal[e].to];

    if(new_dist<dist[yal[e].to]){
     dist[yal[e].to] = new_dist;
     prevee[yal[e].to] = e;
     q.push(MP(-1*new_dist, yal[e].to));
    }
   }
  }

  if (prevee[t]==-1) break;

  edge_type f = INF;

  for(int i = t;i!=s;i = yal[prevee[i]^1].to)
```

```
   f = min(f,yal[prevee[i]].cap);

  for(int i = t;i!=s;i = yal[prevee[i]^1].to){
   yal[prevee[i]].cap -= f;
   yal[prevee[i]^1].cap += f;
  }

  flowVal += f;
  flowCost += f*(dist[t]-pot[s]+pot[t]);
  for(int i = 0;i<V;++i) if (prevee[i]!=-1) pot[i] += dist[i
      ];
 }
}

int main(){
 int N,M,u[5000],v[5000];
 long long cst[5000],D,K;
 freopen("a.in", "r", stdin);
 while(scanf("%d %d",&N,&M)==2){
  yal.clear();
  V = 2*N+1;
  for(int i=0; i<=V; i++) adj[i].clear();
  for(int i = 0;i<M;++i){
   scanf("%d %d %lld",&u[i],&v[i],&cst[i]);
   --u[i]; --v[i];
  }
  scanf("%lld %lld",&D,&K);
  E=0;
  add(edge(0, 1, D, 0));
  for(int i = 0;i<N;++i) add(edge(1+2*i,1+2*i+1,INF,0));

  for(int i = 0;i<M;++i){
   add(edge(1+2*u[i]+1,1+2*v[i],K,cst[i]));
   add(edge(1+2*v[i]+1,1+2*u[i],K,cst[i]));
  }

  mcmf(0,2*N-1);

  if(flowVal!=D) printf("Impossible.\n");
  else printf("%lld\n",flowCost);
 }

 return 0;
}
```

5.3   Bipartite Matching and Vertex Cover

```
//Bipartite Matching is O(M * N)
#define M 128
```

```cpp
#define N 128
bool graph[M][N];
bool seen[N];
int matchL[M], matchR[N];
int n, m;
bool bpm( int u )
{
    for( int v = 0; v < n; v++ ) if( graph[u][v] )
    {
        if( seen[v] ) continue;
        seen[v] = true;

        if( matchR[v] < 0 || bpm( matchR[v] ) )
        {
            matchL[u] = v;
            matchR[v] = u;
            return true;
        }
    }
    return false;
}
vector<int> vertex_cover()
{
// Comment : Vertices on the left side (n side) are labeled
        like this : m+i where i is the index
 set<int> s, t, um; // um = UnMarked
 vector<int> vc;
 for(int i = 0; i < m; i++)
  if(matchL[i]==-1)
   s.insert(i), um.insert(i);
 while( um.size() )
 {
  int v = *(um.begin());
  for(int i = 0; i < n; i++)
   if( graph[v][i] && matchL[v]!=i)
   {
    t.insert(i);
    if( s.find(matchR[i]) == s.end())
     s.insert(matchR[i]), um.insert(matchR[i]);
   }
  um.erase(v);
 }
 for(int i = 0; i < m; i++)
  if( s.find(i) == s.end() )
   vc.push_back(i);
 for(set<int>::iterator i = t.begin(); i != t.end(); i++)
  vc.push_back((*i) + m);
 return vc;
}
int main()
```

```cpp
{
    // Read input and populate graph[][]
    // Set m, n
    memset( matchL, -1, sizeof( matchL ) );
    memset( matchR, -1, sizeof( matchR ) );
    int cnt = 0;
    for( int i = 0; i < m; i++ )
    {
        memset( seen, 0, sizeof( seen ) );
        if( bpm( i ) ) cnt++;
    }
    vector<int> vc = vertex_cover();
    // cnt contains the number of happy pigeons
    // matchL[i] contains the hole of pigeon i or -1 if
        pigeon i is unhappy
    // matchR[j] contains the pigeon in hole j or -1 if hole
        j is empty
    // vc contains the Vertex Cover
    return 0;
}
```

5.4   Bridge and Articulate Point Finding

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

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

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

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

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

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

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

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

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

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

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

  count = bridges = 0;

  search(0, true);
}
```

5.5   Center of Tree

```cpp
struct node
{
    char ch;
    int col, big, sz;
    vector<int> adj;
}nd[MAXN];
int n, col;
vector<int> vec;
void DFS(int pos, int col)
{
    nd[pos].sz=1;
    nd[pos].col=col;
    int k;
    nd[pos].big=0;
    Rep(i, nd[pos].adj.size())
    {
        k=nd[pos].adj[i];
        if(nd[k].col==col || nd[k].col==-1) continue;
        DFS(k, col);
```

```cpp
            nd[pos].sz+=nd[k].sz;
            nd[pos].big=max(nd[pos].big, nd[k].sz);
        }
        vec.push_back(pos);
}
void div(int r, char ch,int col)
{
    vec.clear();
    DFS(r, col);
    r=vec[0];
    int sz=vec.size();
    Rep(i, vec.size())
    {
        nd[vec[i]].big=max(nd[vec[i]].big, sz-nd[vec[i]].sz);
        if(nd[vec[i]].big<nd[r].big) r=vec[i];
    }
    nd[r].col=-1;
    nd[r].ch=ch;
    Rep(i, nd[r].adj.size()) if(nd[nd[r].adj[i]].col==col)
        div(nd[r].adj[i], ch+1, col+1);
}
```

5.6   Count Triangles

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

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

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

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

int main() {
 int n, m, q;
 cin >> n >> m >> q;
 for (int i = 0; i < m; i++) {
  cin >> fi[i] >> se[i];
```

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

5.7   DFS on Complement Graph

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

using namespace std;

const int maxn = 5e5 + 10;
int nxt[maxn], cmp, n;
vector <int> adj[maxn], ver[maxn];
```

```cpp
bool con(int v,int u) {
    int idx = lower_bound(adj[v].begin(), adj[v].end(), u) -
        adj[v].begin();
    return (idx != adj[v].size() && adj[v][idx] == u);
}

int get(int v) {
    if (nxt[v] == v)
        return v;
    return (nxt[v] = get(nxt[v]));
}

void dfs(int v) {
    nxt[v] = get(v + 1);
    ver[cmp].push_back(v);
    for (int u = get(0); u < n; u = get(u + 1)) {
        if (!con(u, v))
            dfs(u);
    }
}

int main() {
    int m;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; i++) {
        int v, u;
        scanf("%d%d", &v, &u);
        v--, u--;
        adj[v].push_back(u);
        adj[u].push_back(v);
    }
    for (int i = 0; i <= n; i++)
        sort (adj[i].begin(), adj[i].end());
    for (int i = 0; i < maxn; i++)
        nxt[i] = i;

    for (int i = 0; i < n; i++)
        if (get(i) == i)
            dfs(i), cmp++;
    printf("%d\n", cmp);
    for (int i = 0; i < cmp; i++) {
        printf("%d ", (int)ver[i].size());
        for (int j = 0; j < ver[i].size(); j++)
            printf("%d ", ver[i][j] + 1);
        printf("\n");
    }
    return 0;
}
```

5.8 DSU on Tree

```cpp
// How many vertices in subtree of vertice v has some
    property in O(n lg n) time (for all of the queries).
// Approach 1

//sz[i] = size of subtree of node i

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

5.9 Dinic Max Flow

```cpp
// adjacency matrix (fill this up)
// If you fill adj[][] yourself, make sure to include both u
    ->v and v->u.
int cap[NN][NN], deg[NN], adj[NN][NN];

// BFS stuff
int q[NN], prev[NN];

int dinic( int n, int s, int t )
{
//////////////
 memset( deg, 0, sizeof( deg ) );
    for( int u = 0; u < n; u++ )
        for( int v = 0; v < n; v++ ) if( cap[u][v] || cap[v][
            u] )
            adj[u][deg[u]++] = v;
//////////////
    int flow = 0;

    while( true )
    {
        // find an augmenting path
        memset( prev, -1, sizeof( prev ) );
        int qf = 0, qb = 0;
        prev[q[qb++] = s] = -2;
        while( qb > qf && prev[t] == -1 )
            for( int u = q[qf++], i = 0, v; i < deg[u]; i++ )
                if( prev[v = adj[u][i]] == -1 && cap[u][v] )
                    prev[q[qb++] = v] = u;

        // see if we're done
        if( prev[t] == -1 ) break;

        // try finding more paths
        for( int z = 0; z < n; z++ ) if( cap[z][t] && prev[z]
            != -1 )
        {
            int bot = cap[z][t];
            for( int v = z, u = prev[v]; u >= 0; v = u, u =
                prev[v] )
                bot <?= cap[u][v];
            if( !bot ) continue;

            cap[z][t] -= bot;
            cap[t][z] += bot;
            for( int v = z, u = prev[v]; u >= 0; v = u, u =
                prev[v] )
            {
                cap[u][v] -= bot;
                cap[v][u] += bot;
            }
            flow += bot;
        }
    }

    return flow;
}
```

5.10 Floyd Cycle Finding

```cpp
ll a, b, c;
vector<ll> vec;
ll f(ll x)
{
 return (a*x+(x%b))%c;
}
pii floydCycleFinding(ll x0)
{
 // mu : start of cycle , lambda : lenght of cycle, cnt for
     seting limits
 // 1st part: finding k*mu, hares speed is 2x tortoises
 ll tortoise = f(x0), hare = f(f(x0)); // f(x0) is the
     element/node next to x0
 int cnt=0;
 while (tortoise != hare && cnt<=20000000)
 { tortoise = f(tortoise); hare = f(f(hare)); cnt++;}
 if(cnt>20000000) return MP(-1, -1);
 // 2nd part: finding mu, hare and tortoise move at the same
     speed
 ll mu = 0; hare = x0;
 while (tortoise != hare) { tortoise = f(tortoise); hare = f
     (hare); mu++; }

 // 3rd part: finding lambda, hare moves, tortoise stays
 ll lambda = 1; hare = f(tortoise);
 while (tortoise != hare)
 { hare = f(hare); lambda++; }
 return pii(mu, lambda);
}
int main(){
 cin>>a>>b>>c;
 pii ans=floydCycleFinding(1);
 if(ans.first!=-1) cout<<ans.first+ans.second<<endl;
 else cout<<-1<<endl;
 return 0;
}
```

5.11 Hopcroft Karp Max Flow

```cpp
/// e*sqrt(v)
vector<int> adj[MAXN];
int dis1[MAXN], dis2[MAXN], g1[MAXN], g2[MAXN], n, inf
    =1<<30, n1, n2;
queue<int> q;
bool BFS()
{
 for(int i=0; i<n1; i++) dis1[i]=0;
```

```cpp
for(int i=0; i<n2; i++) dis2[i]=0;
for(int i=0; i<n1; i++) if(g1[i]==-1) q.push(i);
bool f=0;
int v, u;
while (!q.empty())
{
 v=q.front(), q.pop();
 for(int i=0; i<adj[v].size(); i++)
 {
  u=adj[v][i];
  if(dis2[u]==0)
  {
   dis2[u]=dis1[v]+1;
   if(g2[u]==-1) f=1;
   else
   {
    dis1[g2[u]]=dis2[u]+1;
     q.push(g2[u]);
   }
  }
 }
}
 return f;
}
bool DFS(int v)
{
 int u;
 for(int i=0; i<adj[v].size(); i++)
 {
  u=adj[v][i];
  if(dis2[u]==dis1[v]+1)
  {
   dis2[u]=0;
   if(g2[u]==-1 || DFS(g2[u]))
   {
    g1[v]=u;
    g2[u]=v;
    return 1;
   }
  }
 }
 return 0;
}
int Hopcroft_Karp()
{
 for(int i=0; i<n1; i++) g1[i]=-1;
 for(int i=0; i<n2; i++) g2[i]=-1;
 int matching=0;
 while (BFS()) for(int i=0; i<n1; i++)
  if(g1[i]==-1 && DFS(i)) matching++;
```

```cpp
 return matching;
}
int col[MAXN];
int ind[MAXN];

void paint_geraph()
{
 memset(col, -1, sizeof col);
 int c, k;
 ind[0]=0;
 col[n1++]=1;
 q.push(0);
 while (!q.empty())
 {
  c=q.front(), q.pop();
  for(int i=0; i<adj[c].size(); i++)
  {
   k=adj[c][i];
   if(col[k]!=-1) continue;
   col[k]=!col[c];
   if(col[k]) ind[k]=n1++;
   else ind[k]=n2++;
   q.push(k);
  }
 }
}
int main(){
 int m, u, v;
 vector<pii> ed;
 cin>>n>>m;
 for(int i=0; i<m; i++)
 {
  cin>>u>>v;
  ed.push_back(make_pair(u-1, v-1));
  adj[u-1].push_back(v-1);
  adj[v-1].push_back(u-1);
 }
 n1=n2=0;
 paint_geraph();
 for(int i=0; i<n; i++) adj[i].clear();
 for(int i=0; i<m; i++)
 {
  u=ind[ed[i].first], v=ind[ed[i].second];
  if(col[ed[i].first]) adj[u].push_back(v);
  else adj[v].push_back(u);
 }
 int ans=Hopcroft_Karp();
 cout<<ans<<endl;
 bool f;
 for(int i=0; i<m; i++)
```

```cpp
 {
  u=ind[ed[i].first], v=ind[ed[i].second];
  f=0;
  if(col[ed[i].first])
  {
   if(g1[u]==v)
    f=1;
  }
  else if(g2[u]==v) f=1;
  if(f) cout<<ed[i].first+1<<' '<<ed[i].second+1<<endl;
 }
 return 0;
}
```

5.12 Hungarian Algorithm

```cpp
#define N 55 //max number of vertices in one part
#define INF 100000000 //just infinity

int cost[N][N]; //cost matrix
int n, max_match; //n workers and n jobs
int lx[N], ly[N]; //labels of X and Y parts
int xy[N]; //xy[x] - vertex that is matched with x,
int yx[N]; //yx[y] - vertex that is matched with y
bool S[N], T[N]; //sets S and T in algorithm
int slack[N]; //as in the algorithm description
int slackx[N]; //slackx[y] such a vertex, that
// l(slackx[y]) + l(y) - w(slackx[y],y) = slack[y]
int prv[N]; //array for memorizing alternating paths


void init_labels()
{
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));
    for (int x = 0; x < n; x++)
        for (int y = 0; y < n; y++)
            lx[x] = max(lx[x], cost[x][y]);
}

void update_labels()
{
    int x, y, delta = INF; //init delta as infinity
    for (y = 0; y < n; y++) //calculate delta using slack
        if (!T[y])
            delta = min(delta, slack[y]);
    for (x = 0; x < n; x++) //update X labels
        if (S[x]) lx[x] -= delta;
    for (y = 0; y < n; y++) //update Y labels
```

```cpp
        if (T[y]) ly[y] += delta;
    for (y = 0; y < n; y++) //update slack array
        if (!T[y])
            slack[y] -= delta;
}


void add_to_tree(int x, int prevx)
//x - current vertex,prevx - vertex from X before x in the
    alternating path,
//so we add edges (prevx, xy[x]), (xy[x], x)
{
    S[x] = true; //add x to S
    prv[x] = prevx; //we need this when augmenting
    for (int y = 0; y < n; y++) //update slacks, because we
        add new vertex to S
        if (lx[x] + ly[y] - cost[x][y] < slack[y])
        {
            slack[y] = lx[x] + ly[y] - cost[x][y];
            slackx[y] = x;
        }
}


void augment() //main function of the algorithm
{
    if (max_match == n) return; //check wether matching is
        already perfect
    int x, y, root; //just counters and root vertex
    int q[N], wr = 0, rd = 0; //q - queue for bfs, wr,rd -
        write and read
    //pos in queue
    memset(S, false, sizeof(S)); //init set S
    memset(T, false, sizeof(T)); //init set T
    memset(prv, -1, sizeof(prv)); //init set prev - for the
        alternating tree
    for (x = 0; x < n; x++) //finding root of the tree
        if (xy[x] == -1)
        {
            q[wr++] = root = x;
            prv[x] = -2;
            S[x] = true;
            break;
        }

    for (y = 0; y < n; y++) //initializing slack array
    {
        slack[y] = lx[root] + ly[y] - cost[root][y];
        slackx[y] = root;
    }
    //second part of augment() function
```

```cpp
while (true) //main cycle
{
    while (rd < wr) //building tree with bfs cycle
    {
        x = q[rd++]; //current vertex from X part
        for (y = 0; y < n; y++) //iterate through all
            edges in equality graph
            if (cost[x][y] == lx[x] + ly[y] && !T[y])
            {
                if (yx[y] == -1) break; //an exposed
                    vertex in Y found, so
                //augmenting path exists!
                T[y] = true; //else just add y to T,
                q[wr++] = yx[y]; //add vertex yx[y], which
                    is matched
                //with y, to the queue
                add_to_tree(yx[y], x); //add edges (x,y)
                    and (y,yx[y]) to the tree
            }
        if (y < n) break; //augmenting path found!
    }
    if (y < n) break; //augmenting path found!

    update_labels(); //augmenting path not found, so
        improve labeling
    wr = rd = 0;
    for (y = 0; y < n; y++)
        //in this cycle we add edges that were added to
            the equality graph as a
        //result of improving the labeling, we add edge (
            slackx[y], y) to the tree if
        //and only if !T[y] && slack[y] == 0, also with
            this edge we add another one
        //(y, yx[y]) or augment the matching, if y was
            exposed
        if (!T[y] && slack[y] == 0)
        {
            if (yx[y] == -1) //exposed vertex in Y found -
                augmenting path exists!
            {
                x = slackx[y];
                break;
            }
            else
            {
                T[y] = true; //else just add y to T,
                if (!S[yx[y]])
                {
                    q[wr++] = yx[y]; //add vertex yx[y],
                        which is matched with
```

```cpp
                //y, to the queue
                add_to_tree(yx[y], slackx[y]); //and
                    add edges (x,y) and (y,
                //yx[y]) to the tree
            }
        }
    }
    if (y < n) break; //augmenting path found!
}

if (y < n) //we found augmenting path!
{
    max_match++; //increment matching
    //in this cycle we inverse edges along augmenting
        path
    for (int cx = x, cy = y, ty; cx != -2; cx = prv[cx],
        cy = ty)
    {
        ty = xy[cx];
        yx[cy] = cx;
        xy[cx] = cy;
    }
    augment(); //recall function, go to step 1 of the
        algorithm
}
}//end of augment() function

int hungarian()
{
    int ret = 0; //weight of the optimal matching
    max_match = 0; //number of vertices in current matching
    memset(xy, -1, sizeof(xy));
    memset(yx, -1, sizeof(yx));
    init_labels(); //step 0
    augment(); //steps 1-3
    for (int x = 0; x < n; x++) //forming answer there
        ret += cost[x][xy[x]];
    return ret;
}
```

5.13   List Dinic

```cpp
//From "You Know Izad?" team cheat sheet
const int MAXN = 300;
struct Edge
{
    int a, b, cap, flow;
};
int n, s, t, d[MAXN], ptr[MAXN];
```

```cpp
vector<Edge> e;
vi adj[MAXN];
void init(){
    e.clear();
    fore(i, 0, MAXN)
        adj[i].clear();
}
void add_edge (int a, int b, int cap) {
    Edge e1 = { a, b, cap, 0 };
    Edge e2 = { b, a, 0, 0 };
    adj[a].push_back ((int) e.size());
    e.push_back (e1);
    adj[b].push_back ((int) e.size());
    e.push_back (e2);
}
bool bfs() {
    queue <int> q;
    q.push(s);
    memset(d, -1, sizeof d);
    d[s] = 0;
    while (!q.empty() && d[t] == -1){
        int v = q.front();
        q.pop();
        for (int i = 0; i < L(adj[v]); ++i)
        {
            int id = adj[v][i],
                to = e[id].b;
            if (d[to] == -1 && e[id].flow < e[id].cap)
            {
                q.push(to);
                d[to] = d[v] + 1;
            }
        }
    }
    return d[t] != -1;
}
int dfs (int v, int flow){
    if (!flow) return 0;
    if (v == t) return flow;
    for (; ptr[v] < L(adj[v]); ++ptr[v]){
        int id = adj[v][ptr[v]],
            to = e[id].b;
        if (d[to] != d[v] + 1)
            continue;
        int pushed = dfs (to, min (flow, e[id].cap - e[id].
            flow));
        if (pushed) {
            e[id].flow += pushed;
            e[id ^ 1].flow -= pushed;
            return pushed;
```

```cpp
        }
    }
    return 0;
}
int dinic(){
    int flow = 0;
    while(true){
        if (!bfs())
            break;
        memset(ptr, 0, sizeof ptr);
        // overflow?
        while (int pushed = dfs (s, INF32))
            flow += pushed;
    }
    return flow;
}
int main()
{
    init();
    // set n, s, t
    // add edges using add_edge (directed edge)
    int result = dinic();
}
```

5.14   Matrix Dinic

```cpp
//From "You Know Izad?" team cheat sheet
#define MAXN 400
struct Edge
{
    int a, b;
    ll cap, flow;
};
int n, c[MAXN][MAXN], f[MAXN][MAXN], s, t, d[MAXN], ptr[MAXN
    ];
bool bfs()
{
    queue <int> q;
    q.push(s);
    memset (d, -1, sizeof d);
    d[s] = 0;
    while(!q.empty()){
        int v = q.front();
        q.pop();
        for (int to=0; to<n; ++to){
            if (d[to] == -1 && f[v][to] < c[v][to]){
                q.push(to);
                d[to] = d[v] + 1;
            }
        }
```

```cpp
        }
    }
    return d[t] != -1;
}
int dfs (int v, int flow)
{
    if (!flow) return 0;
    if (v == t) return flow;
    for (int & to=ptr[v]; to<n; ++to){
        if (d[to] != d[v] + 1) continue;
        int pushed = dfs (to, min (flow, c[v][to] - f[v][to])
            );
        if (pushed){
            f[v][to] += pushed;
            f[to][v] -= pushed;
            return pushed;
        }
    }
    return 0;
}
int dinic() {
    int flow = 0;
    // flow between any two vertices is initially zero
    memset(f, 0, sizeof f);
    while(true){
        if (!bfs()) break;
        memset(ptr, 0, sizeof ptr);
        // overflow?
        while (int pushed = dfs (s, INF32))
            flow += pushed;
    }
    return flow;
}
int main()
{
    // set s (source) , t (sink) , n (nodes)
    memset(c, 0, sizeof c);
    // add edges in capacity (c) matrix
    // call dinic function to get Maxflow
}
```

5.15   Max Flow

```cpp
int cap[MAXN][MAXN],n;
vector<int> adj[MAXN];
int BFS(int s, int e)
{
 bool v[MAXN]={0};
 int p[MAXN], i, t, k, c;
```

```
for(i=1;i<=n;i++)
 p[i] = -1;
queue <int> q;
q.push(s);
v[s]=1;
while(!q.empty())
{
 t=q.front();
 q.pop();
 for(i=0;i<adj[t].size();i++)
 {
  k=adj[t][i];
  if(v[k]==0 && cap[t][k]>0)
  {
   q.push(k);
   v[k]=1;
   p[k]=t;
   if(k==e)
    break;
  }
 }
 if(i<adj[t].size())
  break;
}
k=e,c=1<<28;
while(p[k]>-1)
{
 c=min(c,cap[p[k]][k]);
 k=p[k];
}
k=e;
while(p[k]>-1)
{
 cap[p[k]][k]-=c;
 cap[k][p[k]]+=c;
 k=p[k];
}
if(c==1<<28)
 return 0;
 return c;
}
int max_flow(int s, int e)
{
 int ans=0, c;
 while(1)
 {
  c=BFS(s, e);
  if(c==0)
   break;
  ans+=c;
```

```
 }
 return ans;
}
void add_edge(int u, int v, int c)
{
 adj[u].push_back(v), adj[v].push_back(u);
 if(c!=MOD) cap[u][v]+=c;
 else cap[u][v]=MOD;
}
```

5.16 Min Cost Bipartite Matching 2

```
//From "You Know Izad?" team cheat sheet
vi u (n+1), v (m+1), p (m+1), way (m+1);
for (int i=1; i<=n; ++i) {
    p[0] = i;
    int j0 = 0;
    vi minv (m+1, INF);
    vector<char> used (m+1, false);
    do {
        used[j0] = true;
        int i0 = p[j0], delta = INF, j1;
        for (int j=1; j<=m; ++j)
            if (!used[j]) {
                int cur = a[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;
            }
        for (int j=0; j<=m; ++j)
            if (used[j])
                u[p[j]] += delta, v[j] -= delta;
            else
                minv[j] -= delta;
        j0 = j1;
    } while (p[j0] != 0);
    do {
        int j1 = way[j0];
        p[j0] = p[j1];
        j0 = j1;
    } while (j0);
}
int cost = -v[0]; // minimum cost
// ans -> printable matching result
vi ans (n+1);
for (int j=1; j<=m; ++j)
    ans[p[j]] = j;
```

5.17 Min Cost Bipartite Matching

```
//
    ///////////////////////////////////////////////////////

// Min cost bipartite matching via shortest augmenting paths
//
// This is an O(n^3) implementation of a shortest augmenting
        path
// algorithm for finding min cost perfect matchings in dense
// graphs. In practice, it solves 1000x1000 problems in
    around 1
// second.
//
//   cost[i][j] = cost for pairing left node i with right
    node j
//   Lmate[i] = index of right node that left node i pairs
    with
//   Rmate[j] = index of left node that right node j pairs
    with
//
// The values in cost[i][j] may be positive or negative. To
    perform
// maximization, simply negate the cost[][] matrix.
//
    ///////////////////////////////////////////////////////


#include <algorithm>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

double MinCostMatching(const VVD &cost, VI &Lmate, VI &Rmate
    ) {
  int n = int(cost.size());

  // construct dual feasible solution
  VD u(n);
  VD v(n);
  for (int i = 0; i < n; i++) {
    u[i] = cost[i][0];
    for (int j = 1; j < n; j++) u[i] = min(u[i], cost[i][j]);
  }
```

```cpp
    for (int j = 0; j < n; j++) {
      v[j] = cost[0][j] - u[0];
      for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j] -
          u[i]);
    }

    // construct primal solution satisfying complementary
        slackness
    Lmate = VI(n, -1);
    Rmate = VI(n, -1);
    int mated = 0;
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        if (Rmate[j] != -1) continue;
        if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
    Lmate[i] = j;
    Rmate[j] = i;
    mated++;
    break;
        }
      }
    }

    VD dist(n);
    VI dad(n);
    VI seen(n);

    // repeat until primal solution is feasible
    while (mated < n) {

      // find an unmatched left node
      int s = 0;
      while (Lmate[s] != -1) s++;

      // initialize Dijkstra
      fill(dad.begin(), dad.end(), -1);
      fill(seen.begin(), seen.end(), 0);
      for (int k = 0; k < n; k++)
        dist[k] = cost[s][k] - u[s] - v[k];

      int j = 0;
      while (true) {

        // find closest
        j = -1;
        for (int k = 0; k < n; k++) {
    if (seen[k]) continue;
    if (j == -1 || dist[k] < dist[j]) j = k;
        }
        seen[j] = 1;
```

```cpp
        // termination condition
        if (Rmate[j] == -1) break;

        // relax neighbors
        const int i = Rmate[j];
        for (int k = 0; k < n; k++) {
    if (seen[k]) continue;
    const double new_dist = dist[j] + cost[i][k] - u[i] - v[k];
    if (dist[k] > new_dist) {
      dist[k] = new_dist;
      dad[k] = j;
    }
        }
      }

      // update dual variables
      for (int k = 0; k < n; k++) {
        if (k == j || !seen[k]) continue;
        const int i = Rmate[k];
        v[k] += dist[k] - dist[j];
        u[i] -= dist[k] - dist[j];
      }
      u[s] += dist[j];

      // augment along path
      while (dad[j] >= 0) {
        const int d = dad[j];
        Rmate[j] = Rmate[d];
        Lmate[Rmate[j]] = j;
        j = d;
      }
      Rmate[j] = s;
      Lmate[s] = j;

      mated++;
    }

    double value = 0;
    for (int i = 0; i < n; i++)
      value += cost[i][Lmate[i]];

    return value;
}
```

5.18   Min Cost Max Flow

```cpp
//From "You Know Izad?" team cheat sheet
struct rib {
```

```cpp
    int b, u, c, f;
    size_t back;
};
void add_rib (vector < vector<rib> > & g, int a, int b, int
    u, int c) {
    // u = capacity
    // c = cost per flow (maybe double)
    // add edge between a and b
    rib r1 = { b, u, c, 0, g[b].size() };
    rib r2 = { a, 0, -c, 0, g[a].size() };
    g[a].push_back (r1);
    g[b].push_back (r2);
}
int main() {
    // k = the exact amount of flow (cost is calculated
        according to this)
    // set k to infinity --> gives maxFlow (in flow variable)
    int n, m, k;
    vector < vector<rib> > g (n);
    int s, t;
    //reading the graph
    int flow = 0, cost = 0;
    while (flow < k) {
        vector<int> id (n, 0);
        vector<int> d (n, INF);
        vector<int> q (n);
        vector<int> p (n);
        vector<size_t> p_rib (n);
        int qh=0, qt=0;
        q[qt++] = s;
        d[s] = 0;
        while (qh != qt) {
            int v = q[qh++];
            id[v] = 2;
            if (qh == n) qh = 0;
            for (size_t i=0; i<g[v].size(); ++i) {
                rib & r = g[v][i];
                if (r.f < r.u && d[v] + r.c < d[r.b]) {
                    d[r.b] = d[v] + r.c;
                    if (id[r.b] == 0) {
                        q[qt++] = r.b;
                        if (qt == n) qt = 0;
                    }
                    else if (id[r.b] == 2) {
                        if (--qh == -1) qh = n-1;
                        q[qh] = r.b;
                    }
                    id[r.b] = 1;
                    p[r.b] = v;
                    p_rib[r.b] = i;
```

```cpp
                }
            }
        }
        if (d[t] == INF) break;
        int addflow = k - flow;
        for (int v=t; v!=s; v=p[v]) {
            int pv = p[v]; size_t pr = p_rib[v];
            addflow = min (addflow, g[pv][pr].u - g[pv][pr].f
                );
        }
        for (int v=t; v!=s; v=p[v]) {
            int pv = p[v]; size_t pr = p_rib[v], r = g[pv][pr
                ].back;
            g[pv][pr].f += addflow;
            g[v][r].f -= addflow;
            cost += g[pv][pr].c * addflow;
        }
        flow += addflow;
    }
    // output the result
}
```

5.19   Tarjan SCC

```cpp
int n, low_link[MAXN], index[MAXN], ind, group, gr[MAXN];
stack<int> st;
vector<int> adj[MAXN];
bool instack[MAXN];
void tarjan(int c)
{
 index[c]=low_link[c]=ind++;
 instack[c]=1;
 st.push(c);
 int k;
 Rep(i, (int)adj[c].size())
 {
  k=adj[c][i];
  if(index[k]==-1)
  {
   tarjan(k);
   low_link[c]=min(low_link[c], low_link[k]);
  }
  else if(instack[k]) low_link[c]=min(low_link[c], index[k])
      ;
 }
 if(low_link[c]==index[c])
 {
  group++;
  do
```

```cpp
  {
   k=st.top(), st.pop();
   gr[k]=group;
   instack[k]=0;
  }while (k!=c);
 }
}
int main(){
 Set(index, -1), Set(instack, 0);
 ind=group=0;
 Rep(i, n) if(index[i]==-1) tarjan(i);
 cout<<group<<endl;
 return 0;
}
```

5.20   Weighted Min Cut

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

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

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

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

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

        // add the other vertices
        int prev = v[0];
        for( int i = 1; i < n; i++ )
        {
```

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

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

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

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

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

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

6   Math

6.1   Discrete Logarithm Solver

```cpp
// discrete-logarithm, finding y for equation k = x^y % mod
int discrete_logarithm(int x, int mod, int k) {
  if (mod == 1) return 0;
  int s = 1, g;
```

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

6.2   Euler Totient Function

```
/**************************
 * Euler totient function *
 **************************
 * Returns the number of positive integers that are
 * relatively prime to n. As efficient as factor().
 * #include <vector>
 * REQUIRES: factor()
 * REQUIRES: sqrt() must work on Int.
 * REQUIRES: the constructor Int::Int( double ).
 **/
int phi( int n ) {
vector< int > p;
factor( n, p );
for( int i = 0; i < ( int )p.size(); i++ ) {
 if( i && p[i] == p[i - 1] ) continue;
 n /= p[i];
 n *= p[i] - 1;
}
return n;
}
```

6.3   Extended GCD

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

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

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

6.4   Gaussian Elimination

```
//Amin Anvari's solution to Shortest XOR Path problem
const int maxl = 31;
int s, w[maxn], pat[maxn], b[maxn];

int get(int x) {
    for (int i = maxl - 1; i >= 0; i--)
        if (x & (1 << i))
            return i;
    return -1;
}

void add(int x) {
    for (int i = 0; i < s; i++)
        if (get(b[i]) != -1 && (x & (1 << get(b[i]))))
            x ^= b[i];
    if (x == 0)
        return;
    for (int i = 0; i < s; i++)
        if (b[i] < x)
            swap(x, b[i]);
    b[s++] = x;
}

int GET(int x) {
    for (int i = 0; i < s; i++)
        if (get(b[i]) != -1 && (x & (1 << get(b[i]))))
            x ^= b[i];
    return x;
}
```

6.5   Linear Diophantine Equation Solver

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

6.6   Modular Linear Equation Solver

```
/********************************
 * Modular Linear Equation Solver *
 ********************************
 * Given a, b and n, solves the equation ax = b (mod n)
```

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

6.7   Number of Divisors

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

  for( int i = k - 1; i >= 0; i-- ) {
    table[i] = table[i + 1];
    for( int j = i + 1; ; j++ )
      if( j == k || f[j] != f[i] )
      { table[i] += table[j]; break; }
  }

  return table[0];
}
```

6.8   Prime Factors in n Factorial

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

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

int main() {
  int tc;
  ll n, k;
  cin >> tc;
  while (tc--) {
    v.clear();
    cin >> n >> k;
    fact(k);
```

```
  cout << getfact(n) << endl;
  }
  return 0;
}
```

6.9   Reduced Row Echelon Form (Gaussian Elimination)

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

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

using namespace std;

const double EPSILON = 1e-10;

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

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

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

```cpp
        r++;
    }
    return r;
}

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

    int rank = rref(a);

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

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

6.10   Solving Recursive Functions

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

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

7   Sequences

7.1   FFT and Multiply

```cpp
//From "You Know Izad?" team cheat sheet
# definebasecomplex <double>
  voidfft (vector<base> & a , boolinvert ){
    if    (L(a) == 1 ) return ;
```

```cpp
    intn    = L (a);
    vector    <base> a0 ( n / 2 ), a1 ( n / 2 );
    for    ( inti = 0 ,  j  = 0 ;  i  < n ;  i  += 2 ,  ++j){

        a0    [j]  = a [i];
        a1    [j]  = a [ i + 1 ];
    }
    fft    (a0, invert );
    fft    (a1, invert );
    doubleang    = 2 * PI / n * ( invert ?  -1  : 1 );

    basew    (1), wn (cos(ang), sin(ang));
    fore    (i, 0 ,  n  / 2 ) {
        a    [i]  = a0 [i]  +  w  * a1 [i];
        a    [ i +  n  / 2 ]  = a0 [i]  -  w  * a1 [i];
        if    (invert)
            a    [i]  /= 2 , a [ i +  n  / 2 ]  /= 2 ;
        w    *= wn ;
    }
}
voidmultiply (constvector<int> &a, constvector <int> & b ,
    vector <int> &res){
    vector    <base> fa (all(a)), fb (all(b));
    size_tn    = 1 ;
    while    ( n < max (L(a), (L(b)))) n  <<= 1 ;
    n    <<= 1 ;
    fa    .resize(n), fb.resize(n);
    fft    (fa, false ), fft (fb, false );
    fore    (i, 0 , n )
    fa[i]  *= fb [i];
    fft    (fa, true );
    res    . resize (n);
    fore    (i, 0 , n )
    res[i]  =  int  (fa[i].real() + 0 .5);
}
```

7.2   LIS

```cpp
void reconstruct_print(int end, int a[], int p[]) {
    int x = end;
    stack<int> s;
    for (; p[x] >= 0; x = p[x]) s.push(a[x]);
    printf("[%d", a[x]);
    for (; !s.empty(); s.pop()) printf(", %d", s.top());
    printf("]\n");
}

int main() {
    int n = 11, A[] = {-7, 10, 9, 2, 3, 8, 8, 1, 2, 3, 4};
```

```cpp
int L[MAX_N], L_id[MAX_N], P[MAX_N];

int lis = 0, lis_end = 0;
for (int i = 0; i < n; ++i) {
  int pos = lower_bound(L, L + lis, A[i]) - L;
  L[pos] = A[i];
  L_id[pos] = i;
  P[i] = pos ? L_id[pos - 1] : -1;
  if (pos + 1 > lis) {
    lis = pos + 1;
    lis_end = i;
  }
  printf("LIS ending at A[%d] is of length %d: ", i, pos +
      1);
  reconstruct_print(i, A, P);
  printf("\n");
}

printf("Final LIS is of length %d: ", lis);
reconstruct_print(lis_end, A, P);
return 0;
}
```

7.3   Manacher Longest Palindrome

```cpp
string preProcess(string s) {
  int n = s.length();
  if (n == 0) return "^$";
  string ret = "^";
  for (int i = 0; i < n; i++)
    ret += "#" + s.substr(i, 1);

  ret += "#$";
  return ret;
}

string longestPalindrome(string s) {
  string T = preProcess(s);
  int n = T.length();
  int *P = new int[n];
  int C = 0, R = 0;
  for (int i = 1; i < n-1; i++) {
    int i_mirror = 2*C-i; // equals to i' = C - (i-C)
    P[i] = (R > i) ? min(R-i, P[i_mirror]) : 0;
    // Attempt to expand palindrome centered at i
    while (T[i + 1 + P[i]] == T[i - 1 - P[i]])
      P[i]++;
    // If palindrome centered at i expand past R,
    // adjust center based on expanded palindrome.
```

```cpp
    if (i + P[i] > R) {
      C = i;
      R = i + P[i];
    }
  }
}
// Find the maximum element in P.
int maxLen = 0;
int centerIndex = 0;
for (int i = 1; i < n-1; i++) {
  if (P[i] > maxLen) {
    maxLen = P[i];
    centerIndex = i;
  }
}
delete[] P;

return s.substr((centerIndex - 1 - maxLen)/2, maxLen);
}
```

8   Strings

8.1   Aho Corasick

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

struct AC_trie {
  int N, P;
  vector<map<char, int>> next; // trie
  vector<int> link, out_link;
  vector<vector<int>> out;
  AC_trie(): N(0), P(0) { node(); }
  int node() {
    next.emplace_back(); // trie
    link.emplace_back(0);
    out_link.emplace_back(0);
    out.emplace_back(0);
    return N++;
  }
  int add_pattern(const string T) {
    int u = 0;
    for (auto c : T) {
      if (!next[u][c]) next[u][c] = node();
      u = next[u][c];
    }
    out[u].push_back(P);
```

```cpp
    return P++;
  }
  void compute() {
    queue<int> q;
    for (q.push(0); !q.empty(); ) {
      int u = q.front(); q.pop();
      // trie:
      for (auto e : next[u]) {
        int v = e.second;
        link[v] = u ? advance(link[u], e.first) : 0;
        out_link[v] = out[link[v]].empty() ? out_link[link[v
            ]] : link[v];
        q.push(e.second);
      }
    }
  }
  int advance(int u, char c) {
    // trie:
    while (u && next[u].find(c) == next[u].end())
      u = link[u];
    if (next[u].find(c) != next[u].end())
      u = next[u][c];
    return u;
  }
  void match(const string S) {
    int u = 0;
    for (auto c : S) {
      u = advance(u, c);
      for (int v = u; v; v = out_link[v])
        for (auto p : out[v])
          cout << "match " << p << endl;
    }
  }
};
struct AC_automaton {
  int N, P;
  vector<vector<int>> next; // automaton
  vector<int> link, out_link;
  vector<vector<int>> out;
  AC_automaton(): N(0), P(0) { node(); }
  int node() {
    next.emplace_back(26, 0); // automaton
    link.emplace_back();
    out_link.emplace_back();
    out.emplace_back();
    return N++;
  }
  int add_pattern(const string T) {
    int u = 0;
    for (auto c : T) {
```

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

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

```cpp
  match1.compute();
  match2.compute();
  cin >> T;
  match1.match(T);
  match2.match(T);
  return 0;
}
```

8.2   KMP

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

8.3   Suffix Array and LCP

```cpp
//From "You Know Izad?" team cheat sheet
const int MAXLG = 15;
const int MAXN = 3000;
```

```cpp
int P[MAXLG][MAXN], stp;
string s;
struct entry {
    int nr[2], p;
} L[MAXN];
int cmp(entry a, entry b) {
    return a.nr[0] == b.nr[0] ? (a.nr[1] < b.nr[1] ? 1 : 0) :
        (a.nr[0] < b.nr[0] ? 1 : 0);
}
void init() {
    memset(P, 0, sizeof P);
    stp = 1;
    fore(i, 0, MAXN) {
        L[i].nr[0] = 0;
        L[i].nr[1] = 0;
        L[i].p = 0;
    }
}
int rangeComp(int idx, const string &t, int len){
    fore(i, 0, len) {
        if(i >= L(t) && i + idx >= L(s)) return 0;
        else if(i + idx >= L(s)) return 1;
        else if(i >= L(t)) return -1;
        if(s[i + idx] == t[i]) continue;
        if(s[i + idx] > t[i]) return 1;
        return -1;
    }
    return 0;
}
void construct() {
    init();
    for (int i = 0; i < L(s); i++)
        P[0][i] = s[i] - 'a';
    for (int cnt = 1; (cnt >> 1) < L(s); stp++, cnt <<= 1){
        for (int i = 0; i < L(s); i++){
            L[i].nr[0] = P[stp - 1][i];
            L[i].nr[1] = i + cnt < L(s) ? P[stp-1][i+cnt] :
                -1;
            L[i].p = i;
        }
        sort(L, L + L(s), cmp);
        for (int i = 0; i < L(s); i ++)
            P[stp][L[i].p] = i > 0 && L[i].nr[0] == L[i - 1].
                nr[0] && L[i].nr[1] == L[i - 1].nr[1] ? P[stp
                ][L[i - 1].p] : i;
    }
}
ii stringMatching(const string &t){
    int low = 0, high = L(s) - 1, mid = low;
    while (low < high){
```

```
        mid = (low + high) / 2;
        int res = rangeComp(L[mid].p , t, L(t));
        if (res >= 0) high = mid;
        else low = mid + 1;
    }
    if (rangeComp(L[low].p , t, L(t)) != 0) return ii(-1, -1)
        ;
    ii ans;
    ans.first = low;
    low = 0; high = L(s) - 1; mid = low;
    while (low < high) {
        mid = (low + high) / 2;
        int res = rangeComp(L[mid].p, t, L(t));
        if (res > 0) high = mid;
        else low = mid + 1;
    }
    if (rangeComp(L[high].p, t, L(t)) != 0) high--;
    ans.second = high;
    return ans;
}
int lcp(int x, int y) {
    int ret = 0;
    if (x == y) return L(s) - x;
    for (int k = stp - 1; k >= 0 && x < L(s) && y < L(s); k
        --) {
        if (P[k][x] == P[k][y])
            x += (1 << k), y += (1 << k), ret += (1 << k);
    }
    return ret;
}
int main() {
    cin >> s;
    construct();
    string t;
// rangeComp and stringMatching are optional
    while(cin >> t) {
        ii ans = stringMatching(t);
        cout << ans.first << " " << ans.second << endl;
    }
}
```

8.4   Suffix Array

```
char T[MAX_N];                   // the input string, up to 100
    K characters
int n;                           // the length of
    input string
```

```
int RA[MAX_N], tempRA[MAX_N];       // rank array and temporary
    rank array
int SA[MAX_N], tempSA[MAX_N]; // suffix array and temporary
    suffix array
int c[MAX_N];                           // for counting/
    radix sort

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

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

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

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

void constructSA() {       // this version can go up to
    100000 characters
    int i, k, r;
    for (i = 0; i < n; i++) RA[i] = T[i];           // initial
        rankings
    for (i = 0; i < n; i++) SA[i] = i; // initial SA: {0, 1,
        2, ..., n-1}
```

```
    for (k = 1; k < n; k <<= 1) {    // repeat sorting process
        log n times
        countingSort(k); // actually radix sort: sort based on
            the second item
        countingSort(0);        // then (stable) sort based on
            the first item
        tempRA[SA[0]] = r = 0;          // re-ranking; start from
            rank r = 0
        for (i = 1; i < n; i++)             // compare
            adjacent suffixes
            tempRA[SA[i]] = // if same pair => same rank r;
                otherwise, increase r
            (RA[SA[i]] == RA[SA[i-1]] && RA[SA[i]+k] == RA[SA[i-1]+
                k]) ? r : ++r;
        for (i = 0; i < n; i++)                 // update the
            rank array RA
            RA[i] = tempRA[i];
        if (RA[SA[n-1]] == n-1) break;          // nice
            optimization trick
    }
}

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

ii stringMatching() {                   // string matching in
    O(m log n)
    int lo = 0, hi = n-1, mid = lo;       // valid matching
        = [0..n-1]
```

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

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

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

ii LCS() {              // returns a pair (the LCS length
    and its index)
  int i, idx = 0, maxLCP = -1;
  for (i = 1; i < n; i++)                  // O(n), start
      from i = 1
```

```cpp
    if (owner(SA[i]) != owner(SA[i-1]) && LCP[i] > maxLCP) //
        LCP[i]>0 ro bayad ezafe kard
      maxLCP = LCP[i], idx = i;
  return ii(maxLCP, idx);
}

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

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

  computeLCP();                                        // O(n)

  // LRS demo
  ii ans = LRS();                // find the LRS of the first
      input string
  char lrsans[MAX_N];
  strncpy(lrsans, T + SA[ans.second], ans.first);
  lrsans[ans.first]=0;// man ezafe kardam
  printf("\nThe LRS is '%s' with length = %d\n\n", lrsans,
      ans.first);

  // stringMatching demo
  //printf("\nNow, enter a string P below, we will try to
      find P in T:\n");
  strcpy(P, "A");
  m = (int)strlen(P);
  // if '\n' is read, uncomment the next line
  //P[m-1] = 0; m--;
  ii pos = stringMatching();
  if (pos.first != -1 && pos.second != -1) {
    printf("%s is found SA[%d..%d] of %s\n", P, pos.first,
        pos.second, T);
    printf("They are:\n");
    for (int i = pos.first; i <= pos.second; i++)
      printf(" %s\n", T + SA[i]);
  } else printf("%s is not found in %s\n", P, T);
```

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

  // reconstruct SA of the combined strings
  constructSA();                                      // O
      (n log n)
  computeLCP();

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

  ans = LCS();        // find the longest common substring
      between T and P
  char lcsans[MAX_N];
  strncpy(lcsans, T + SA[ans.second], ans.first);
  lcsans[ans.first]=0;// man ezafe kardam
  printf("\nThe LCS is '%s' with length = %d\n", lcsans, ans
      .first);

  return 0;
}
```

8.5   Z Algorithm

```cpp
// Z[i] => max len perfixi az s k az khuneye i e S shoru
    mishe

int L = 0, R = 0;
n=s.size();
for (int i = 1; i < n; i++)
{
  if (i > R) {
    L = R = i;
```

```
 while (R < n && s[R-L] == s[R]) R++;
 z[i] = R-L; R--;
}
else
{
 int k = i-L;
 if (z[k] < R-i+1) z[i] = z[k];
 else
 {
  L = i;
  while (R < n && s[R-L] == s[R]) R++;
  z[i] = R-L; R--;
 }
}
}
```

9   Theorems

9.1   Gallai

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

9.2   Konig

```
 Knig  theorem can be proven in a way that provides
     additional useful information beyond just its truth:
     the proof provides a way of constructing a minimum
     vertex cover from a maximum matching. Let {G=(V,E)} be
     a bipartite graph, and let the vertex set { V} be
     partitioned into left set { L} and right set { R}.
     Suppose that { M} is a maximum matching for { G}. No
     vertex in a vertex cover can cover more than one edge
     of { M} (because the edge half-overlap would prevent {
     M} from being a matching in the first place), so if a
     vertex cover with { |M|} vertices can be constructed,
     it must be a minimum cover.[10]
To construct such a cover, let { U} be the set of unmatched
     vertices in { L} (possibly empty), and let { Z} be the
     set of vertices that are either in { U} or are
     connected to { U} by alternating paths (paths that
     alternate between edges that are in the matching and
```

```
     edges that are not in the matching). Let
{ K=(L - Z) Union (R Intersect Z).}
Every edge { e} in { E} either belongs to an alternating
     path (and has a right endpoint in { K}), or it has a
     left endpoint in { K}. For, if { e} is matched but not
     in an alternating path, then its left endpoint cannot
     be in an alternating path (for such a path could only
     end at { e}) and thus belongs to { L- Z}. Alternatively
     , if { e} is unmatched but not in an alternating path,
     then its left endpoint cannot be in an alternating path
     , for such a path could be extended by adding { e} to
     it. Thus, { K} forms a vertex cover.[11]
Additionally, every vertex in { K} is an endpoint of a
     matched edge. For, every vertex in { L - Z} is matched
     because Z is a superset of U, the set of unmatched left
      vertices. And every vertex in { R intersect Z} must
     also be matched, for if there existed an alternating
     path to an unmatched vertex then changing the matching
     by removing the matched edges from this path and adding
      the unmatched edges in their place would increase the
     size of the matching. However, no matched edge can have
      both of its endpoints in { K}. Thus, { K} is a vertex
     cover of cardinality equal to { M}, and must be a
     minimum vertex cover.[11]
```