Table of Contents

Maximum bipartite matching with augmenting paths:

```
findMaximumMatching(bipartite graph)
  for all unmatched vertices v
    set level of all vertices to 0;
    set parent of all vertices to null;
    level(v) = 1;
    last = null;
    clear queue;
    enqueue(v);

    while queue is not empty and last is null
      v = dequeue();

      if level(v) is an odd number
        for all vertices u adjacent to v such that level(u) is 0
          if u is unmatched
            parent(u) = v;
            last = u;
            break;
          else if u is matched but not with v
            parent(u) = v;
            level(u) = level(v) + 1;
            enqueue(u);
          end if
        end for
      else
         enqueue(vertex u matched with v);
         parent(u) = v;
         level(u) = level(v) +1;
      end if
    end while

    if last is not null
      for u = last ; u is not null ; u = parent(parent(u))
        matchedWith(u) = parent(u);
        matchedWith(parent(u)) = u;
      end for
    end if
  end for
end findMaximumMatching
```

Clipping a subject polygon with a clipper polygon   CohenSutherland algorithm. Polygons must be convex:

```
#include <stdio.h>
#include <math.h>
#define SIZE 2001

typedef struct{
  double x,y;
}Point;

enum { LEFT=0,RIGHT,BEHIND,BEYOND,ORIGIN,DESTINATION,BETWEEN};
enum { COLLINEAR=0,PARALLEL,SKEW,SKEW_CROSS,SKEW_NO_CROSS};

/* p2 on which side of p1-p0 */
int classify(Point p0,Point p1,Point p2) {
  Point a,b;
  double t;
  a.x=p1.x-p0.x;
  a.y=p1.y-p0.y;
  b.x=p2.x-p0.x;
  b.y=p2.y-p0.y;
  t=a.x*b.y-a.y*b.x;
  if(t>0.0)
    return LEFT;
  if(t<0.0)
    return RIGHT;
  if((a.x*a.x+a.y*a.y)<(b.x*b.x+b.y*b.y))
    return BEYOND;
  if((a.x>0.0 && b.x<0.0) || (a.x<0.0 && b.x>0.0))
    return BEHIND;
  if((a.y>0.0 && b.y<0.0) || (a.y<0.0 && b.y>0.0))
    return BEHIND;
  if(p0.x==p2.x && p0.y==p2.y)
    return ORIGIN;
  if(p1.x==p2.x && p1.y==p2.y)
    return DESTINATION;
  return BETWEEN;
}

double dotProduct(Point a,Point b){
  return a.x*b.x+a.y*b.y;
}

/* intersection of b-a and d-c : t of b-a */
int intersect(Point a,Point b,Point c,Point d,double *t){
  double denom,num;

  int aclass;
  Point n,ba,ac;
  n.x=d.y-c.y;
  n.y=c.x-d.x;
  ba.x=b.x-a.x;
  ba.y=b.y-a.y;
  ac.x=a.x-c.x;
  ac.y=a.y-c.y;
  denom=dotProduct(n,ba);
  if(denom==0.0){
    aclass=classify(c,d,a);
    if(aclass==LEFT || aclass==RIGHT)
      return PARALLEL;
    else
      return COLLINEAR;
  }
  num=dotProduct(n,ac);
  *t=-num/denom;
  return SKEW;

}

/* clip polygon p[] using edge b-a */
int lineClip(Point a,Point b,Point p[],int n){
  Point r[SIZE];
  Point org,dest,crosspt;
  int orgIsInside,destIsInside;
  double t;
  int i,j;
  p[n]=p[0];
  for(i=j=0;i<n;i++){
    org=p[i];
    dest=p[i+1];
    orgIsInside=(classify(a,b,org)!=LEFT);
    destIsInside=(classify(a,b,dest)!=LEFT);
    if(orgIsInside!=destIsInside){
      intersect(a,b,org,dest,&t);
      crosspt.x=a.x+t*(b.x-a.x);
      crosspt.y=a.y+t*(b.y-a.y); }
    if(orgIsInside && destIsInside)
      r[j++]=dest;
    else if(orgIsInside && !destIsInside){
      if(org.x!=crosspt.x || org.y!=crosspt.y)
        r[j++]=crosspt;
    }
    else if(!orgIsInside && !destIsInside)
      ;
    else{
      r[j++]=crosspt;
      if(dest.x!=crosspt.x || dest.y!=crosspt.y)
        r[j++]=dest;
    }
  }
  for(i=0;i<j;i++)
    p[i]=r[i];
  return j;
```

```
}

int polygonClip(Point subject[],int m,Point clipper[],int n) {
  int tm,i;
  clipper[n]=clipper[0];
  for(i=0;i<n;i++){
    tm=lineClip(clipper[i],clipper[i+1],subject,m);
    m=tm;
  }
  return m;
}
```

Optimal Binary Search Tree with Knuth's Optimization:

```
Optimal-BST(p,q,n)
  for i=1 to n+1 do
    e[i,i-1] = q[i-1];
    w[i,i-1] = q[i-1];
  end for

  for k=1 to n do
    for i=1 to n-k+1 do
      j = i+k-1;e[i,j] =  ;
      w[i,j] = w[i,j-1] + p[j] + q[j];

      for r=root[i,j-1] to root[i+1,j] do
        t = e[i,r-1] + e[r+1,j] + w[i,j];
        if t < e[i,j] then
          e[i,j] = t;
          root[i,j] = r;
        end if
      end for
    end for
  end for

  return e and root
end Optimal-BST
```

Here, p[i] = probability of a query ending in i-th node, 1   i   n
       q[i] = probability of a failed query, 0   i   n

Convex Hull Code

```
long p[MAX][2],h[MAX][2];

void Swap(long &x,long &y){
  long t;
  t = x; x = y; y = t;
}

long Dis(long i,long j){
  return ((p[i][0]-p[j][0])*(p[i][0]-p[j][0]) + (p[i][1]-p[j][1])*(p[i][1]-p[j][1]));
}

long Area(long i,long j,long k){
  return ( p[i][0]*(p[j][1]-p[k][1]) + p[j][0]*(p[k][1]-p[i][1])
     + p[k][0]*(p[i][1]-p[j][1]));
}

void SetMinimum(long n) {
  long index,i;
  index = 0 ;
  for(i=1;i<n;i++){
    if( p[i][1] > p[index][1] ) continue;
    else if( p[i][1] < p[index][1] )
      index = i;
    else{
      if( p[i][0] < p[index][0] )
        index = i;
    }
  }
  Swap(p[0][0],p[index][0]);
  Swap(p[0][1],p[index][1]);
}

long ConvexHull(long n){
  long i,j,count,prev,next,area,dis1,dis2,index;
  SetMinimum(n);
  h[0][0] = p[0][0];
  h[0][1] = p[0][1];
  index = 1;
  prev = 0;
  while(1){
    next = (prev+1) % n;
    for(j=next,count=1; count<=n-2; count++ ){
      i = (j+count) % n;
      area = Area(prev,next,i);
      if( area > 0 ) continue;
      else if ( area < 0 )
        next = i;
      else{
        dis1 = Dis(prev,next);
        dis2 = Dis(prev,i);
        if( dis2 > dis1 ) next = i;
      }
    }
    if(next==0) break;
    else{
      h[index][0] = p[next][0];
      h[index][1] = p[next][1];
      index++;
      prev = next ;
    }
  }
  return index;
}

void main(){
  long i,j,k,l,n,num,index,Case;
  scanf("%ld",&Case);
  printf("%ld\n",Case);
  for( num=0; num<Case ; ){
    scanf("%ld",&n );
    if( n==-1) continue;
    for( i=0; i<n ; i++)
      scanf("%ld%ld",&p[i][0],&p[i][1]);
    index = ConvexHull(n-1);
    printf("%ld\n",index+1);
    for(i=0; i<index; i++)
      printf("%ld %ld\n",h[i][0],h[i][1]);
    printf("%ld %ld\n",h[0][0],h[0][1]);
    if(num!=Case-1)
      printf("-1\n"); num++;
  }
}
```

DioPhantine Equation and Euclid's extended algorithm

```
typedef long int lint;

void Euclid(lint u, lint v, lint &f,lint &g){
  lint up[4] = {0, 1, 0, 0}, vp[4] = {0,0,1,0};

  lint q, t[4], i;
  double ft;
  up[3] = u;
  vp[3] = v;
  while(vp[3] != 0){
    ft = (double)up[3] / vp[3];
    ft = floor(ft);
    q = (lint)ft;
    for(i = 1; i <= 3; i++){
      t[i] = up[i] - vp[i]*q;
      up[i] = vp[i];
      vp[i] = t[i];

    }
  }
  f = up[1];
  g = up[2];
  return;
}

lint gcd(lint a,lint b){
  lint t;
  if(a<b){
    t=a;
    a=b;
    b=t;
  }
  while(b){
    t=a%b;
    a=b;
    b=t;
  }
  return a;
}

int main() {
  lint n,n1,n2,gd,u,v,temp,d,e,f,minx,miny;
  long double total,min,c1,c2,t,x,y,x0,y0,start,end,start1,end1;
  while(scanf("%ld",&n) == 1 && n){
    scanf("%Lf %ld %Lf %ld",&c1,&n1,&c2,&n2);
    gd = gcd(n1,n2);
    if(n % gd != 0 ){
      printf("failed\n");
      continue;
    }
    else{
      d = n1 / gd;
      e = n2 / gd;
      f = n / gd;
      Euclid(d,e,u,v);
      x0 = u * ((long double)f);
      y0 = v * ((long double)f);
      /*x = x0-e*t, y = y0+d*t*/
      start = ceil((-y0)/((long double)d));
      end = floor((x0)/((long double)e));
      min = INT_MAX;
      if(end < start)
        printf("failed\n");
      else{
        min = INT_MAX;
        t = start;
        x = x0 - e*t;
        y = y0 + d*t;
        total = x*c1 + y*c2;
        minx = x;
        miny = y;
        min = total;
        t = end;
        x = x0 - e*t;
        y = y0 + d*t;
        total = x*c1 + y*c2;
        if(total < min){
          minx = x;
          miny = y;
          min = total;
        }
        printf("%ld %ld\n",minx, miny);
      }
    }
  }
  return 0;
}
```

Determining if a point lies on the interior of a polygon

```
#define MIN(x,y) (x < y ? x : y)
#define MAX(x,y) (x > y ? x : y)

#define INSIDE 0
#define OUTSIDE 1

typedef struct{
  double x,y;
}Point;

int InsidePolygon(Point *polygon,int N,Point p) {
  int i,counter = 0;
  double xinters;
  Point p1,p2;
  p1 = polygon[0];
  for(i=1;i<=N;i++){
    p2 = polygon[i % N];
    if(p.y > MIN(p1.y,p2.y)){
      if(p.y <= MAX(p1.y,p2.y)){
        if(p.x <= MAX(p1.x,p2.x)){
          if (p1.y != p2.y){
            xinters = (p.y-p1.y)*(p2.x-p1.x)
              /(p2.y-p1.y)+p1.x;
            if (p1.x == p2.x || p.x <= xinters) counter++;
          }
        }
      }
    }
    p1 = p2;
  }
  if(counter % 2 == 0) return(OUTSIDE);
  else return(INSIDE);
}
```

Solution 2 (2D)

```
typedef struct{
  int h,v;
} Point;

int InsidePolygon(Point *polygon,int n,Point p) {
  int i;
  double angle=0;
  Point p1,p2;
  for (i=0;i<n;i++){
    p1.h = polygon[i].h - p.h;
    p1.v = polygon[i].v - p.v;
    p2.h = polygon[(i+1)%n].h - p.h;
    p2.v = polygon[(i+1)%n].v - p.v;
    angle += Angle2D(p1.h,p1.v,p2.h,p2.v);
  }
  if (ABS(angle) < PI) return(FALSE);
  else return(TRUE);
}

/* Return the angle between two vectors on a plane
The angle is from vector 1 to vector 2, positive anticlockwise
The result is between -pi -> pi */
```

```
double Angle2D(double x1, double y1, double x2, double y2){
  double dtheta,theta1,theta2;
  theta1 = atan2(y1,x1);
  theta2 = atan2(y2,x2);
  dtheta = theta2 - theta1;
  while (dtheta > PI)
    dtheta -= TWOPI;
  while (dtheta < -PI) dtheta += TWOPI;
  return (dtheta);
}
```

Solution 4 (3D)

To determine whether a point is on the interior of a convex polygon in 3D one might be tempted to first determine whether the point is on the plane, then determine it's interior status. Both of these can be accomplished at once by computing the sum of the angles between the test point (q below) and every pair of edge points p[i]->p[i+1]. This sum will only be twopi if both the point is on the plane of the polygon AND on the interior. The angle sum will tend to 0 the further away from the polygon point q becomes.

The following code snippet returns the angle sum between the test point q and all the vertex pairs. Note that the angle sum is returned in radians.

```
typedef struct{
  double x,y,z;
} XYZ;
#define EPSILON 0.0000001
#define MODULUS(p) (sqrt(p.x*p.x + p.y*p.y + p.z*p.z))
#define TWOPI 6.283185307179586476925287
#define RTOD 57.2957795

double CalcAngleSum(XYZ q,XYZ *p,int n) {
  int i;
  double m1,m2,anglesum=0,costheta;
  XYZ p1,p2;
  for(i=0;i<n;i++){
    p1.x = p[i].x - q.x;
    p1.y = p[i].y - q.y;
    p1.z = p[i].z - q.z;
    p2.x = p[(i+1)%n].x - q.x;
    p2.y = p[(i+1)%n].y - q.y;
    p2.z = p[(i+1)%n].z - q.z;
    m1 = MODULUS(p1);
    m2 = MODULUS(p2);
    if(m1*m2 <= EPSILON)
      return(TWOPI); /* We are on a node, consider this inside */
    else
      costheta = (p1.x*p2.x + p1.y*p2.y + p1.z*p2.z) / (m1*m2);
    anglesum += acos(costheta);
  }
  return(anglesum);
}
```

Centroid

As in the calculation of the area above, xN is assumed to be x0, in other words the polygon is closed.

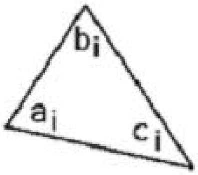
$$c_x = \frac{1}{6A} \sum_{i=0}^{N-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i) \qquad c_y = \frac{1}{6A} \sum_{i=0}^{N-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$$

Centroid of a 3D shell described by 3 vertex facets

The centroid C of a 3D object made up of a collection of N triangular faces with vertices (ai,bi,ci) is given below. Ri is the average of the vertices of the i'th face and Ai is twice the area of the i'th face. Note the faces are assumed to be thin sheets of uniform mass, they need not be connected or form

a solid object.
This reduces to the equations above for a 2D 3 vertex polygon.

$$C = \frac{\sum_{i=0}^{N-1} A_i R_i}{\sum_{i=0}^{N-1} A_i}$$



$$R_i = (a_i + b_i + c_i) / 3$$

$$A_i = \|(b_i - a_i) \otimes (c_i - a_i)\|$$

Strongly Connected Component

1 Call DFS(G) to compute finishing time f[u] for each vertex u
2 Compute G^T
3 Call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing f[u].
4 Output the vertices of each tree in the depth-first forest of step 3 as a separate strongly connected component.

Graham Scan(Q)

Q is a set of points representing polygon vertices in arbitrary order, |Q| >= 3. When the algorithm terminates, stack S contains exactly the vertices of CH(Q), in counter-clockwise order.

```
Let P0 the point in Q with the minimum y-coordinate, or the leftmost such point in the case of
a tie.
Let < P1, P2, …….,Pm > be the remaining points in Q, sorted by polar angle in counterclockwise
order around P0 ( If more than one point has the same angle, remove all but the one that is
farthest from P0)
top[S] = 0
Push(P0,S)
Push(P1,S)
Push(P2,S)
For i = 3 to m
   do while the angle formed by points Next-To-Top(S),Top(S) and Pi
        makes a nonleft turn
          do Pop(S)
    Push(S,Pi)
return S
```

Extended-Euclid(a,b)

```
If b = 0
     then return (a,1,0)
(d', x', y') = Extended-Euclid(b, a mod b)
(d, x, y) = (d', y', x' ⌊a/b⌋y')
return (d, x, y)
```

Modular Linear Equation Solver(a,b,n)

Obtains the solutions of ax = b mod n

```
(d, x', y') = Extended-Euclid(a,n)
if d | b then
   x0 = x'(b/d) mod n
   for i = 0 to d  1
     do print (x0 + i (n/d)) mod n
   end for
else
   print "no solutions"
end if
```

Modular-Exponentiation(a,b,n)

```
c = 0
d = 1
let < bk,bk-1,….,b0 > be the binary representation of b
for i = k downto 0
   c = 2c
   d = (d · d) mod n
   if bi = 1 then
      c = c + 1
      d = (d · a) mod n
   end if
end for
return d
```

Knuth-Morris-Pratt Algorithm

KMP-Matcher(T,P)

```
n = length[T]
m = length[P]
p = Compute-Prefix-Function(P)
q = 0
for i = 1 to n
   while q > 0 and P[q+1] != T[i]
     q = p[q]
      if P[q+1] = T[i] then
         q = q + 1
        if q = m then
              print "Pattern occurs with shift" i   m
         q = p[q]
       end if
     end if
   end while
end for
```

Compute-Prefix-Function(P)

```
m = length[P]
p[1] = 0
k = 0
for q = 2 to m
   while k > 0 and P[k+1] != P[q]
     k = p[k]
   end while
   if P[k+1] = P[q] then
      k = k + 1
   end if

   p[q] = k
end for
return p
```

JAVA Selected material

Taking Input:

```
static BufferedReader stdin = new BufferedReader(new InputStreamReader(System.in));
String s = stdin.readLine();
Stringtokenizer tokens = new StringTokenizer(s);
S = tokens.nextToken();
```

Alternate way of taking input using StreamTokenizer:

```
static StreamTokenizer input = null;
static int getInt() throws IOException {
        if (input == null){
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        input = new StreamTokenizer(br);
}
input.nextToken();
  return (int)(input.nval+0.01);
}
```

finding whether symmetry line exists for a non self-intersecting polygon (convex/concave)

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

#define MAX 200
#define PREC 1e-8

struct Point{
  int x,y;
} p[MAX];

int N;

int equals(double x,double y,double prec){
  return fabs(x-y) < prec;
}

bool isSymLine(double x1,double y1,double x2,double y2,
       int prev,int next,Point p[],int N) {
  double a,b,c,mx,my;
  int j,k,l;
  a = y1-y2;
  b  = x2-x1;
  c  = y1*(x1-x2) - x1*(y1-y2);
  for(k=0;k<N/2;k++){
    j = (next+k)%N;
    l = (prev-k+N)%N;
    // Now find mirror of alpha = p[j].x and beta = p[j].y
    mx = -(p[j].x*(a*a-b*b) + 2*a*(c+b*p[j].y))/(a*a+b*b);
    my = (p[j].y*(a*a-b*b) - 2*b*(c+a*p[j].x))/(a*a+b*b);
    if(!equals(mx,p[l].x,PREC) || !equals(my,p[l].y,PREC))
      return false;
  }
  return true;
}

bool oddCase(void) {
  int i,j,k;
  double x1,y1,x2,y2;
  for(i=0;i<N;i++){
    x1 = p[i].x;
    y1 = p[i].y;
    j = (i+N/2)%N;
    k = (j+1)%N;
    x2 = (p[j].x + p[k].x)/2.0;
    y2 = (p[j].y + p[k].y)/2.0;
    if(isSymLine(x1,y1,x2,y2,(i-1+N)%N,(i+1)%N,p,N)) break;
  }
  return i<N;
}

bool evenCase(void) {
  int i,j,k,l;
  double x1,y1,x2,y2;
  for(i=0;i<N;i++){
    j = (i+1)%N;
    k = (i+N/2)%N;
    l = (k+1)%N;
    x1 = (p[i].x+p[j].x)/2.0;
    y1 = (p[i].y+p[j].y)/2.0;
    x2 = (p[k].x+p[l].x)/2.0;
    y2 = (p[k].y+p[l].y)/2.0;
    if(isSymLine(x1,y1,x2,y2,i,j,p,N)) break;
  }
  if(i<N) return true;
  for(i=0;i<N;i++){
    x1 = p[i].x; y1 = p[i].y;
    j = (i+N/2)%N;
    x2 = p[j].x;
    y2 = p[j].y;
    if(isSymLine(x1,y1,x2,y2,i,i,p,N)) break;
  }
  return i<N;
}

int main(void) {
  int testCase,dataSet,i;
  bool t;
  scanf("%d",&dataSet);
  for(testCase=0;testCase<dataSet;testCase++) {
    scanf("%d",&N);
    for(i=0;i<N;i++)
      scanf("%d %d",&p[i].x,&p[i].y);
    if(N%2) t = oddCase();
    else t = evenCase();
    if(t) printf("Yes\n");
    else printf("No\n");
  }
  return 0;
}
```

Vector

- If 3 points A(**a**),B(**b**) & R(**r**) with common origin are collinear, scalars a,ß & ? are such that a+ß+? = 0 and **a**a+**b**ß+**r**? = 0.
- If position vectors **a**,**b**,**c**,**d** of four points A,B,C,D, no three of which are collinear and the non-zero scalars a,ß,?,d are such that a+ß+?+ d = 0 and **a**a+**b**ß+**c**?+**d**d = 0, then the four points are coplanar.
- [**a** **b** **c**]=[**b** **c** **a**]=[**c** **a** **b**]=**a**.(**b**×**c**). If vectors **a**,**b**,**c** represent the 3 sides of a parallelopiped, then its volume V = |[**abc**]|.

- Let A(**a**),B(**b**),C(**c**),D(**d**) be the 4 vertices of a tetrahedron.
  Then its volume V= [(**a d) (b d) (c d)**]/6
- Let a,b,c,d are 4 vectors. Then **d**   [**d b c**]**a** [**d c a**]**b**   [**d a b**]**c) / [a b c]**, [**a b c**] ? 0
- Let 2 non-coplanar straight lines be: **r** = **a**+t**b, r = a'**+s**b'**. Then, shortest distance between these lines = [**bb**'**a**' **a**] / **b**   **b**' .
- Equation of a plane: *p*   **r**.**n** = 0. *p* is the perpendicular distance from origin. **n** is unit normal of the plane.
- Equation of the bisectors of the angles between 2 planes *p*   **r**.**n1** = 0, *p*   **r**.**n2** = 0: If the origin & the points on the bisector lie on the same side of the given planes, the 2 perpendicular distances have the same sign. Then, the equation is *p*   **r**.**n1** = *p'*   **r**.**n2** or *p*   *p'* = **r**.(**n1-n2)**. When the origin & points on the other bisector are on the opposite sides of one of the given planes, then the perpendicular distances will be of opposite signs. Hence the equation is *p* + *p'* = **r.**(**n1+n2**).

Translation, Scaling & Rotation Matrices

Translation Scaling Rotation about Z

$$\begin{vmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \begin{vmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad \begin{vmatrix} cost & -sint & 0 & 0 \\ sint & cost & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Numerical Methods

Descartes' rule of sign

- In a polynomial equation P(x)= 0, the terms being written in the order of power of x, the number of positive roots **cannot exceed** the number of changes of sign (+ to - & vice verse) in the coefficients of P(x), and the negative roots **cannot exceed** the number of changes of sign P(-x).
- If P(x) is continuous in [a,b], then if P(a) & P(b) have opposite signs, there exists an odd number of real roots of the equation P(x) = 0 in (a,b). If P(a) & P(b) have same sign, then there exists no real root or an even number of real roots in (a,b).

Derivative & factor of a polynomial

Let P(x) = $a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n$. Let $b_0 = a_0$, $b_r = b_{r-1}X + a_r$,[r=1,2..,n]$_/$ & Q(x) = $b_0x^{n-1} + b_1x^{n-2} + ... + b_{n-2}x + b_{n-1}$. Then P(x) = (x-X)Q(x) + b_n. Therefore derivative at x = X, P (X) = Q(X). Also if x-X is a factor of P(x) then P(x) = (x-X)Q(x).

Lagrange's Interpolation Formula

$$f(x)=\frac{(x-x_1)(x-x_2)...(x-x_n)}{(x_0-x_1)(x_0-x_2)...(x_0-x_n)}f(x_0)+\frac{(x-x_0)(x-x_2)...(x-x_n)}{(x_1-x_0)(x_1-x_2)...(x_1-x_n)}f(x_1)+...+\frac{(x-x_0)(x-x_1)...(x-x_{n-1})}{(x_n-x_0)(x_n-x_2)...(x_n-x_{n-1})}f(x_n)$$

, f(x) is a polynomial in n] (x_n x_0)(x_n x_1)...(x_n $x_{n\ 1}$)

Assignment Problem

```
# include <stdio.h>
# define SIZE 5

void matching(int c[2*SIZE][2*SIZE],int n,int m[2*SIZE])
{
  int qsize=n+1,q[2*SIZE+1],rear,front,par[2*SIZE],level[2*SIZE],i,j,k,last;
  for(i=0;i<n;i++)
    m[i]=-1;
  for(k=0;k<n;k++){
    if(m[k]==-1){
      last=-1;
      for(i=0;i<n;i++){
        level[i]=0;
        par[i]=-1;
      }
      rear=front=0;
      level[k]=1;
      q[rear++]=k;
      if(rear==qsize) rear=0;
      while(rear!=front){
        if(last!=-1) break;
        i=q[front++];
        if(front==qsize) front=0;
        if(level[i]%2){
          for(j=0;j<n;j++) {
            if(c[i][j] && level[j]==0){
              level[j]=level[i]+1;
              par[j]=i;
              if(m[j]==-1){
                last=j;
                break;
              }
              else{
                q[rear++]=j;
                if(rear==qsize) rear=0;
              }
            }
          }
        }
        else{
          j=m[i];
          level[j]=level[i]+1;
          par[j]=i;
          q[rear++]=j;
          if(rear==qsize) rear=0;
        }
      }
      while(last!=-1){
        m[last]=par[last];
        m[par[last]]=last;
        last=par[par[last]];
      }
    }
  }
}

int isperfect(int m[],int n) {
  int i;
  for(i=0;i<n;i++)
    if(m[i]==-1) return 0;
    return 1;
}

void dfs(int k,int i,int mark[2][SIZE],int e[SIZE][SIZE],int n,int m[],int depth){
  int j;
  if(depth) mark[k][i]=1;
  if(k==0){
    for(j=0;j<n;j++){
      if(e[i][j]==0 && m[i]!=j+n){
        mark[k][i]=1;
        if(mark[1][j]==0) dfs(1,j,mark,e,n,m,depth+1);
      }
    }
  }
  else{
    for(j=0;j<n;j++){
      if(e[j][i]==0 && m[i+n]==j){
        mark[k][i]=1;
        if(mark[0][j]==0) dfs(0,j,mark,e,n,m,depth+1);
      }
    }
  }
}

void OptimalAssignment(int w[SIZE][SIZE],int n,int m[]){
  int u[SIZE],v[SIZE],e[SIZE][SIZE],c[2*SIZE][2*SIZE];
  int mark[2][SIZE],i,j,k,eta;
  for(i=0;i<n;i++){
    u[i]=v[i]=0;
    for(j=0;j<n;j++)
      if(w[i][j]>u[i]) u[i]=w[i][j];
  }
  for(i=0;i<n;i++){
    for(j=0;j<n;j++)
      e[i][j]=u[i]+v[j]-w[i][j];
  }
  for(i=0;i<2*n;i++){
    for(j=0;j<2*n;j++)
      c[i][j]=0;
  }
  for(i=0;i<n;i++){
    for(j=0;j<n;j++)
      if(e[i][j]==0)
        c[i][n+j]=c[n+j][i]=1;
  }
  matching(c,2*n,m);
  while(!isperfect(m,2*n)){
    for(i=0;i<n;i++)
      mark[0][i]=mark[1][i]=0;
    for(i=0;i<n;i++){
      if(m[i]==-1 && mark[0][i]==0)
        dfs(0,i,mark,e,n,m,0);
    }
    eta=-1;
    for(i=0;i<n;i++){
      for(j=0;j<n;j++) {
        if(mark[0][i] && mark[1][j]==0){
          if(eta==-1 || eta>u[i]+v[j]-w[i][j])
            eta=u[i]+v[j]-w[i][j];
        }
      }
    }
    for(i=0;i<n;i++){
      if(mark[0][i]) u[i]-=eta;
      if(mark[1][i]) v[i]+=eta;
    }
    for(i=0;i<n;i++){
      for(j=0;j<n;j++)
        e[i][j]=u[i]+v[j]-w[i][j];
    }
    for(i=0;i<2*n;i++){
      for(j=0;j<2*n;j++)
        c[i][j]=0;
    }
    for(i=0;i<n;i++){
      for(j=0;j<n;j++){
        if(e[i][j]==0)
          c[i][n+j]=c[n+j][i]=1;
      }
    }
    matching(c,2*n,m);
  }
}

int main(void){
  int n=5,w[SIZE][SIZE]={{4,1,6,2,3},{5,0,3,7,6},{2,3,4,5,8},{3,4,6,3,4},{4,6,5,8,6}};
  int m[2*SIZE],i,total;
  OptimalAssignment(w,n,m);total=0;
  for(i=0;i<n;i++)
    total+=w[i][m[i]-n];
  printf("%d\n",total);
  return 0;
}
```

Map

```
#include <stdio.h>
#include <map>
#include <malloc.h>

struct species {
  char s[81];
} buf;

bool operator==(const species &a, const species &b){
  return !strcmp(a.s, b.s);
}

bool operator<(const species &a, const species &b){
  return strcmp(a.s, b.s) < 0;
}

map<species,int> cnt;

main() {
  int tot = 0;
  while(gets(buf.s)){
    cnt[buf]++;
    tot++;
  }
  for (map<species,int>::iterator i = cnt.begin(); i!=cnt.end(); *i++){
    printf("%s %0.4lf\n",i->first.s, 100.0*i->second/tot);
  }
}
```

Stable Matching

Let there be *n* men and *n* women. A perfect matching is stable if and only if there is no man *x* and woman *a* such that *x* prefers *a* to his current partner and *a* prefers *x* to her current partner. A stable matching always exists.

Algorithm

Input: Preference rankings by each of *n* men and *n* women

Iteration: Each unmatched man proposes to the highest woman on his preference list who has not previously rejected him and is not yet matched. If each woman receives exactly one proposal, stop with these being the remaining confirmed matches. Otherwise, at least one woman receives at least two proposals. Every woman receiving more that one proposal rejects all but the highest on her list. Every woman receiving a proposal says “maybe” to the most attractive proposal received.

Note: The same algorithm can be used with women instead of men proposing. Of all stable matchings, every man is happiest under the male-proposal algorithm, and every woman is happiest under the female-proposal algorithm.

Eular cycles

Fleury’s Algorithm - constructing Eulerian trails

Input: A graph G with one nontrivial component and at most two odd vertices.

Initialization: Start at a vertex that has odd degree unless G is even, in which case start at any vertex.

Iteration: From the current vertex, traverse any remaining egde whose deletion from the remaining graph does not leave a graph with two non-trivial components. Stop when all edges have been traversed.

Directed Eulerian circuit

Input: A digraph *G* that is an orientation of a connected graph and has $d^{+}(u) = d^{-}(u)$for all $u \in V(G)$.
Step 1: Choose a vertex $v \in V(G)$. Let *G’* be the digraph obtained from *G* by reversing direction on each edge. Search *G’* to construct a tree *T’* consisting of paths from *v* to all other vertices (use BFS or DFS).
Step 2: Let *T* be the reversal of *T’*; *T* contains a *u,v*-path in *G* for each *u* *V*(*G*). Specify an arbitrary ordering of the edges that leave each vertex *u*, except that for *u* ? *v* the edge leaving *u* in *T*

must come last. (The tree edge is last in order).**Step 3:** Construct an Eulerian circuit from *v* as follows: whenever *u* is the current vertex, exit along the next unused edge in the ordering specified for edges leaving *u.*

Pick's Theorem

I = area + 1   B/2,
where I = number of points inside
B = number of points on the border.

Misc Geometric Formula

triangle	Circum Radius = a*b*c/(4*area) In Radius = area/s, where s = (a+b+b)/2 length of median to side c = sqrt(2*(a*a+b*b)-c*c)/2 length of bisector of angle C = sqrt(ab[(a+b)*(a+b)-c*c])/(a+b)
Ellipse	Area = PI*a*b Circumference = 4a *int(0,PI/2){sqrt(1-(k*sint)*(k*sint))}dt = 2*PI*sqrt((a*a+b*b)/2) approx where k = sqrt((a*a-b*b)/a) = PI*(3*(r1+r2)-sqrt[(r1+3*r2)*(3*r1+r2)])
Spherical cap	V = (1/3)*PI*h*h*(3*r-h) Surface Area = 2*PI*r*h
Spherical Sector	V = (2/3)*PI*r*r*h
Spherical Segment	V = (1/6)*PI*h*(3*a*a+3*b*b+h*h)
Torus	V = 2*PI*PI*R*r*r
Truncated Conic	V = (1/3)*PI*h*(a*a+a*b+b*b) Surface Area = PI*(a+b)*sqrt(h*h+(b-a)*(b-a)) = PI*(a+b)*l
Pyramidal frustum	(1/3)*h*(A1+A2+sqrt(A1*A2))

Misc Trigonometric Functions

Tan A/2 = ±sqrt((1-cos A)/(1+cos A))
= sin A / (1+cos A)
= (1-cos A) / sin A
= cosec A   cot A
sin 3A = 3*sin A   4*sincube A
cos 3A = 4*coscube A   3*cos A
tan 3A = (3*tan A-tancube A)/(1-3*tansq A)
sin 4A = 4*sin A*cos A   8*sincube A*cos A
cos 4A = 8*$\cos^4$ A   8*cossq A + 1
$[r*(cost+i*sint)]^p$ = r^p*(cos pt+i*sin pt)

Misc Integration Formula

a^x => a^x/ln(a)
1/sqrt(x*x+a*a) => ln(x+sqrt(x*x+a*a))
1/sqrt(x*x-a*a) => ln(x+sqrt(x*x-a*a))
1/(x*sqrt(x*x+a*a) => -(1/a)*ln([a+sqrt(x*x+a*a)]/x)
1/(x*sqrt(a*a-x*x) => -(1/a)*ln([a+sqrt(a*a-x*x)]/x)

Misc Differentiation Formula

asin x => 1/sqrt(1-x*x)
acos x => -1/sqrt(1-x*x)
atan x => 1/(1+x*x)
acot x => -1/(1+x*x)
asec x => 1/[x*sqrt(x*x-1)]
acosec x => -1/[x*sqrt(x*x-1)]
a^x => a^x*ln(x)
cot x => -cosecsq x
sec x => sec x * tan x
cosec x => -cosec x * cot x

Weighted Bipartite Matching - Hungerian Algorithm

```
/*
Problem  : Weighted Matching
Approach  : Algorithm by Kuhn and Munkres
Reference  : Photocopy Book

Input    : Given an N/N by matrix. If edgeList form is given then
      Other matrix entry is ZERO
Output    : A mathcing which maximizes the sum of Weight

Remark    : Intially all vertices of U/X has at least one edge in the
      equality subGraph but not for vertices in Y/V. So after
      initial matching each node of Y/V along with an edge in
      equality subGraph should have a match

x#S      : x is element of S
y$T      : y is not element of T
*/
#include <stdio.h>
#include <string.h>

#define MAX 50
#define INF 32000
#define SIZE (2*MAX+1)

int mat[SIZE][SIZE];    /* Weighted Matrix */
int matchedWith[SIZE];
int level[SIZE];
int parent[SIZE];
int N;

int weight[SIZE];   /* Weight/Label of each Node */
int S[SIZE];    /* Set of vertices of X/U in Hungarian Tree */
int T[SIZE];    /* Vertices in Y/V and present in the current Matching */

void init(void)
{
  int i;
  for(i=0;i<SIZE;i++)
    memset(mat[i],0,sizeof(int)*SIZE);
  return;
}

/*
Given an Unmatched Vertex 'u' from U/X. It will try to find an augmenting
path using Hungarian Tree. On success(Tree finds an unmatched vertex 'v' in
V/Y) it will return 1 and also modify the matchedWith[]
*/
int augmentPath(int u)
{
  int v,w,last;
  int q[SIZE+1],head,tail;

  /* Initialize part. Make level of all vertices_0 to indicate not Visited*/
  memset(level,0,sizeof(int)*(2*N+1));
  memset(parent,0,sizeof(int)*(2*N+1)); /* Necessary for Last part of function */

  /* Queue initialization part */
  head   = 0;   tail   = 1;
  q[1]   = u;  level[u]= 1;
  last  = 0;

  /* Start building Hungarian Tree */
  while((head!=tail) && !last)
  {
    head++;
    v = q[head];

    if(level[v]%2)  /* level[v] is odd. So it is in X/U part */
    {
      /* Adjacent vertex in V/Y */
      for(w=N+1;w<=2*N;w++)
      {
        /* If there is a edge with 'w' and already not visited */
        if(!level[w] && (mat[v][w]==weight[v]+weight[w]))
        {
          if(!matchedWith[w])
            /* 'w' is unmatched. Augmenting path found */
          {
            parent[w] = v;
            last      = w;
            break;
          }
          else if(matchedWith[w]!=v)/* Unmatched Edge(v,w) */
          {
            parent[w] = v;
            level[w]  = level[v]+1;
            q[++tail] = w;
          }
        }
      }
    }
    else
    {
      w        = matchedWith[v];/* Matching partner of v */
      parent[w]  = v;
      q[++tail]  = w;      /* Enque matched Edge (v,w) */
      level[w]  = level[v]+1;
    }
  }

  /* Modify Matching Part */
  if(last)    /* Augmenting Path found */
  {
    /* For this looping initialization of parent is a must */
    /* Take all unmathced edge and change current Matching */
    for(w=last; w; w=parent[parent[w]])
    {
      matchedWith[w]     = parent[w];
      matchedWith[parent[w]]  = w;
    }
    return 1;
  }
  else
  {
    /* As evidence Mark all x of U in S and all y of V in T */
    for(v=1;v<=N;v++)
    {
      if(level[v])  /* Vertices of X */
        S[v]   = 1;
      if(level[N+v])  /* Vertices of Y */
        T[N+v]   = 1;
    }
    return 0;
  }
}

void optimalMatching(void)
{
  int i,j,max,slack,u,v;
  int nMatch,eps;

  /* Intialization Part */

  /* Intially no Matching */
  memset(matchedWith,0,sizeof(int)*(2*N+1));
  /* Intially give u,v proper level. */
  for(i=1;i<=N;i++)
  {
    max = 0;
    for(j=N+1;j<=2*N;j++)
    {
      if(mat[i][j] > max)
        max = mat[i][j];
    }
    weight[i]   = max;
    weight[i+N] = 0;
  }
  /* Initally Matching is ZERO */
  nMatch   = 0;
  while(1)
  {
    memset(S,0,sizeof(int)*(2*N+1));
    memset(T,0,sizeof(int)*(2*N+1));
    for(i=1;i<=N;i++)
    {
      if(!matchedWith[i])  /* Unmatched vertex belongs to U */
      {
        if(augmentPath(i))
          nMatch++;
      }
    }
    if(nMatch == N)    /* Perfect Matching Found */
      break;
    /* Now S[i] is all vertices of U/X visited in Hungarian Tree from
       all unmatched vertices in X and same to T for Y/V. Find eps
       defined by Min [ l(x)+l(y) -wt(x,y) ] where x#S and y$T  */
    eps = INF;
    for(i=1;i<=N;i++)
    {
      if(S[i])            /* i shoould be in S */
      {
        for(j=N+1;j<=2*N;j++)
        {
          if(!T[j])    /* j must not be in T */
          {
            slack = weight[i]+weight[j]-mat[i][j];
            if(slack < eps)
              eps = slack;
          }
        }
      }
    }
    /* Now reLabel each Vertex */
    for(i=1;i<=N;i++)
    {
      if(S[i])
        weight[i]    -= eps;
      if(T[i+N])
        weight[i+N]  += eps;
    }
  }
  return ;
}

void main(void)
{
  int i,j,sum;

  freopen("input.txt","rt",stdin);
  freopen("output.out","wt",stdout);
  while(scanf("%d",&N)==1)
  {
    init();
    for(i=1;i<=N;i++)
    {
      for(j=N+1;j<=2*N;j++)
        scanf("%d",&mat[i][j]);
    }
    optimalMatching();

    for(i=1,sum=0;i<=N;i++)
    {
      j    = matchedWith[i];
      sum += mat[i][j];
    }
    printf("%d\n",sum);
  }
  return;
}
```

Common Errors

1. integer = {-ve, 0, +ve}
2. Overflow … intermediate overflow. use long long if not sure
   a. sqrt(a*b*c*d) instead of sqrt(a*b)*sqrt(c*d)
   b. Intermediate value overflow
   c. double is reliable for upto 15 digits, long long 18 digits.
   d. use BigInteger/Java for big number calculation
   e. if only small add, div by int, sub etc needed, may use C++
3. print “Case X:” … or Case #X:
4. Replacing j for i
   a. Swapping dimensions erroneously. Input as [r][c], use as [c][r]
   b. all pairs order of loops: k,I,j
5. Domain Errors
   a. Div by zero
   b. Sqrt(-ve)
   c. domain error on acos(), asin() and atan(). use myatan() etc.
   d. slope can be infinity.
6. Comparing double number using ‘==’
   a. floating point calculations must use EPS
7. Using ‘=’ instead of ‘==’ and vise versa
8. ve array indexing => use assert to check
   a. do not step into an invalid state in bfs or other searching alg.
9. Global local
   a. Using same variable twice...avoid this kind of optimization shortcuts
   b. using same looping variable as inner looping variable
10. Initialization
   a. input multiple test cases
   b. input the same test case multiple times
   c. use a function ‘input()’ for complex inputs
   d. Failure to clean up during backtrack
   e. Local variables are not initialized to zero
   f. clear array[n] if used. for example in gauss elimination where n+1 elements are needed per row.
11. 0-indexing and 1-indexing
   a. in input, process or output.
   b. Looping i <= n and looping i < n as needed
12. complex if-condition check mistakes
13. taking input using “%d” for long long, upper bytes may contain garbage
   a. %d for double values
   b. not using or using &
   c. difference between byref and byval
14. not using exhaustive BFS
15. not testing the input for boundary cases.
   a. check for small values in input, n=1, n=0. try to define correctly.
16. same point twice? same edge twice? same node?
17. Not reading the problem properly
18. Using scanf(“%s”,s) and gets()
19. remembering the difference between multiples and divisibles
20. using i when dist[i] is needed in dijkstra priority queuing.
21. Strategies
   a. Use binary search to guess input size / time limit / other limit
   b. Generate a table using inefficient code and send the table.
   c. optimize the most critical point
   d. in dp, use sum table to lower complexity by O(n)
   e. remember IDS, backtracking.
   f. solve easy ones first. check stats
   g. leave the WA, try that later.
22. termination
   a. whether all conditions should match or any one
   b. better check after input. not in the while satatement.
   c. EOF or n=0
23. Mark any critical constraints that make the problem easiler. read though the problem carefully.