Algoritmos

Sumário

Combinação

binomialCoefficient

```
#include<iostream>
#define MAXN 100
using namespace std;
long binomial_coefficient(int n,int m)
{
   int i,j;               //Counters
   long bc[MAXN][MAXN];     //Table of binomial coefficients
   for(i=0; i<=n; i++)
      bc[i][0] = 1;
   for(j=0; j<=n; j++)
      bc[j][j] = 1;
   for(i=1; i<=n; i++)
      for(j=1; j<i; j++)
         bc[i][j] = bc[i-1][j-1] + bc[i-1][j];
   return(bc[n][m]);
}
int main() {
  cout << binomial_coefficient(10, 5);
  return 0;
}
```

Combination

```
#include<iostream>
#include<string.h>
using namespace std;
void DoCombine(char in[], char out[], int length, int recursLev, int start) {
  for(int i=start; i<length; i++)
  {
    out[recursLev] = in[i];    //Select the current letter
    out[recursLev + 1] = '\0';  //tack on NULL for printf
    printf("%s\n", out);
    if(i<(length-1))
      DoCombine(in, out, length, recursLev + 1, i+1);
  }
}
int combine(char inString[]) {
   int length;
   char *out;
   length = strlen(inString);
   out = (char *)malloc(length+1);
   if(!out)
```

```
    return 0;
  DoCombine(inString, out, length, 0, 0);
  free(out);
  return 1;
}
int main() {
  combine("abcd");
  return 0;
}
```

Permutation

```
#include<iostream>
#include<string.h>
using namespace std;
void DoPermute(char in[], char out[], int used[], int length, int recursLev)
{
  //base case
  if(recursLev == length) {
    printf("%s\n", out);
    return;
  }
  //Recursive Case
  for(int i=0; i<length; i++) {
    if(used[i])  //If used, skip to next letter
      continue;
    out[recursLev] = in[i];  //Put current letter in output
    used[i] = 1;            //Mark this letter as used
    DoPermute(in, out, used, length, recursLev+1);
    used[i] = 0;           //Unmark this letter
  }
}
int Permute(char inString[]) {
  int length, *used;
  char *out;
  length = strlen(inString);
  out = (char *)malloc(length+1);
  if(!out)
    return 0; //Failed
  //so printf doesn't run past the end of the buffer
  out[length] = '\0';
  used = (int *)malloc(sizeof(int) * length);
  if(!used)
    return 0; //Failed
  //Start with no letters used, so zero array
  for(int i=0; i<length; i++)
    used[i] = 0;
  DoPermute(inString, out, used, length, 0);
  free(out);
  free(used);
  return 1; //Success
```

```
}
int main(){
  Permute("abcdefghijklmnopqrstuvwxyz");
  return 0;
}
```

Diversos

Roman

```
#include<iostream>
#include<string>
using namespace std;
string fill( char c, int n )
{
   string s;
   while( n-- ) s += c;
   return s;
}
/* Converts an integer in the range [1, 4000) to a lower case Roman numeral*/
string toRoman( int n )
{
   if( n < 4 ) return fill( 'i', n );
   if( n < 6 ) return fill( 'i', 5 - n ) + "v";
   if( n < 9 ) return string( "v" ) + fill( 'i', n - 5 );
   if( n < 11 ) return fill( 'i', 10 - n ) + "x";
   if( n < 40 ) return fill( 'x', n / 10 ) + toRoman( n % 10 );
   if( n < 60 ) return fill( 'x', 5 - n / 10 ) + 'l' + toRoman( n % 10 );
   if( n < 90 ) return string( "l" ) + fill( 'x', n / 10 - 5 ) + toRoman( n % 10 );
   if( n < 110 ) return fill( 'x', 10 - n / 10 ) + "c" + toRoman( n % 10 );
   if( n < 400 ) return fill( 'c', n / 100 ) + toRoman( n % 100 );
   if( n < 600 ) return fill( 'c', 5 - n / 100 ) + 'd' + toRoman( n % 100 );
   if( n < 900 ) return string( "d" ) + fill( 'c', n / 100 - 5 ) + toRoman( n % 100 );
   if( n < 1100 ) return fill( 'c', 10 - n / 100 ) + "m" + toRoman( n % 100 );
   if( n < 4000 ) return fill( 'm', n / 1000 ) + toRoman( n % 1000 );
   return "?";
}
int main() {
  cout << toRoman(3) << endl;
  cout << toRoman(300) << endl;
  cout << toRoman(3999) << endl;
  cout << toRoman(1) << endl;
  cout << toRoman(50) << endl;
  return 0;
}
```

Estrutura de dados

BST

```
#include<iostream>
using namespace std;
struct BST {
  int val;
  struct BST *left , *right;
};
BST* create(int val)
{
    BST *node = new BST;
    node->val = val;
    node->left = NULL;
    node->right = NULL;
    return node;
}
void insert(BST **root, int val)
{
  while(*root != NULL) {
    if((*root)->val < val)
        root = &((* root)->left);
    else
        root = &((* root)->right);
  }
  *root = create(val);

}
void list(BST *root)
{
  if(root != NULL) {
    list(root->right);
    cout << root->val << " ";
    list(root->left);
  }
}
int find(BST **root, int val)
{
   int tval;
   while(*root != NULL) {
     tval = (* root)->val;
     if(tval < val)
       root = &((* root)->left);
     else if(tval > val)
       root = &((* root)->right);
     else
       return 1;
   }
   return 0;
```

```
}
int main()
{
  BST *tree = new BST;
  tree = NULL;
  insert(&tree, 1);
  insert(&tree, 4);
  insert(&tree, 10);insert(&tree, 20);
  list(tree);
  cout << "Searching " << endl;
  printf("%d ", find(&tree, 10));
  printf("%d ", find(&tree, 20));
  printf("%d ", find(&tree, 7));
  return 0;
}
```

Geometria

ClosestFarthestPair

```
#include<iostream>
#include<math.h>
#include<vector>
using namespace std;

/*  Watch out the coordinates are integers! */
struct Point{
 int x,y; };

/*  Distance between two points */
double Distance(Point p1, Point p2){
 return sqrt(pow(p1.x-p2.x,2) + pow(p1.y-p2.y,2));
}

istream &operator>>(istream &is, Point &p){
 is>>p.x>>p.y;
 return is;
}

ostream &operator<<(ostream &os, Point &p){
 os<<"("<<p.x<<","<<p.y<<")";
 return os;
}

void ClosestFarthestPairs(vector<Point *> points)
{
 int max1, max2, min1, min2;
 double maxy=0, miny=0, maxt, mint;
```

```
  for(int i=0; i<points.size(); i++)
    for(int j=i+1; j<points.size(); j++)
    {
      if(maxy < (maxt=Distance(*points[i], *points[j])))
      {
        maxy = maxt;
        max1 = i;
        max2 = j;
      }
    }
  miny = maxy;
  for(int i=0; i<points.size(); i++)
    for(int j=i+1; j<points.size(); j++)
    {
      if(miny > (mint=Distance(*points[i], *points[j])))
      {
        miny = mint;
        min1 = i;
        min2 = j;
      }
    }
  cout<<"Closest pair is: "<<(*points[min1])<<" and "<<(*points[min2])<<endl;
  cout<<"Farthest pair is: "<<(*points[max1])<<" and "<<(*points[max2])<<endl;
}

int main(){
  vector<Point *> points;
  int i,nPoints;
  while(!cin.eof()){
    cin>>nPoints;
    if(!nPoints) break;
    points.clear();
    Point *p;
    for(i=0;i<nPoints;i++){
      p = new Point;
      cin>>*p;
      points.push_back(p);
    }
    ClosestFarthestPairs(points);
  }
  return 0;
}
```

LargestSetOfPointsOverALine

```
#include<iostream>
#include<math.h>
#include<stdlib.h>
using namespace std;

/*  Watch out the coordinates are integers! */
```

```
struct Point{
  int x,y; };

/*  The cross product: (p1-p) X (p2-p): Zero if parallel */
int CrossProduct(Point p1, Point p2, Point p)
{ return ((p1.x-p.x)*(p2.y-p.y) - (p2.x-p.x)*(p1.y-p.y)); }

istream &operator>>(istream &is, Point &p){
  is>>p.x>>p.y;
  return is;
}

ostream &operator<<(ostream &os, Point &p){
  os<<"("<<p.x<<","<<p.y<<")";
  return os;
}

/* -1 => _p1<_p2; 1 => _p1>_p2; 0 => _p1=_p2; */
int comparePoints(const void *_p1, const void *_p2)
{
  Point p1 = *((Point *)_p1);
  Point p2 = *((Point *)_p2);
  if(p1.x<p2.x)
    if(p1.y<p2.y)
      return 1;
    else
      return -1;
  else if(p1.x>p2.x)
    if(p1.y<=p2.y)
      return 1;
    else
      return -1;
  else
    if(p1.y<p2.y)
      return 1;
    else if(p1.y>p2.y)
      return -1;
    else
      return 0;
}

void LargestSetOfPointsOverALine(Point **points, int nPoints)
{
  Point tempAlignedPoints[nPoints];
  Point alignedPoints[nPoints];
  int max = 0, count;
  for(int i=0; i<nPoints; i++)
    for(int j=i+1; j<nPoints; j++)
    {
      count = 0;
      for(int k=j+1; k<nPoints; k++)
      {
```

```
//       if( k == i || k == j ) continue;
       if( CrossProduct((*points[i]),(*points[j]),(*points[k])) == 0 )
         tempAlignedPoints[count++] = *points[k];
     }
     if(count > max)
     {
       for(int x = 0; x<count; x++)
         alignedPoints[x] = tempAlignedPoints[x];
       alignedPoints[count++] = *points[i];
       alignedPoints[count++] = *points[j];
       max = count;
     }
   }
  cout<<"The largest set of points over a line has "<<max<<" elements. The points are:"<<endl;
  qsort(alignedPoints, max, sizeof(Point), comparePoints);
  for(int i=0; i<max; i++)
    cout<<(alignedPoints[i])<<endl;
}

int main(){
  int i,nPoints;
  while(!cin.eof()){
    cin>>nPoints;
    if(!nPoints) break;
    Point *points[nPoints];
    Point *p;
    for(i=0;i<nPoints;i++){
      p = new Point;
      cin>>(*p);
      points[i] = p;
    }
    LargestSetOfPointsOverALine(points, nPoints);
  }
  return 0;
}
```

LineIntersection

```
#include<iostream>
using namespace std;

/*  Watch out the coordinates are integers! */
struct Point{
  int x,y; };

struct Segment{
  Point p1,p2; };


istream &operator>>(istream &is, Point &p){
  is>>p.x>>p.y;
```

```
  return is;
}

istream &operator>>(istream &is, Segment &s){
  is>>s.p1>>s.p2;
  return is;
}

Point operator-(Point p1, Point p2){
  Point temp;
  temp.x = p1.x - p2.x;
  temp.y = p1.y - p2.y;
  return temp;
}

/*  The cross product for vectors:
u X v: Is zero if parallel segments*/
int VectorCrossProduct(Point u, Point v){
  return (u.x*v.y - u.y*v.x);
}

bool isIntersection(Segment s1, Segment s2){
  Point vector1, vector2;
  vector1 = s1.p1-s1.p2;
  vector2 = s2.p1-s2.p2;
  if( VectorCrossProduct(vector1,vector2) == 0 )
    return false;
  return true;
}

int main(){
  int c;
  c = 0;
  Segment s1,s2;
  while(!cin.eof()){
    cin>>s1; // using operator oeverload!  :)
    if(cin.eof()) break;
    cin>>s2; // using operator oeverload!  :)
    cout<<"Case #"<<++c<<": Do they intersect?"<<endl;
    if(isIntersection(s1,s2))
      cout<<"Yes.";
    else
      cout<<"No.";
    cout<<endl;
  }
  return 0;
}
```

PointInsidePolygon

```
#include<iostream>
#include<vector>
using namespace std;

/*  Watch out the point are integers! */
struct Point{
  int x,y;
};

int maxi(int x, int y){
  return ((x>y)?x:y);
}

int mini(int x, int y){
  return ((x<y)?x:y);
}

/* Checks if p is inside the fucking bounding box
  of the shitting segment formed by ps1 and ps2 */
bool isInBoundingBox(Point ps1, Point ps2, Point p){
  if((mini(ps1.x,ps2.x)<=p.x && p.x<=maxi(ps1.x,ps2.x)) &&
     (mini(ps1.y,ps2.y)<=p.y && p.y<=maxi(ps1.y,ps2.y)))
    return true;
  return false; }

/*  The cross product: (p1-p) X (p2-p): Zero if parallel */
int CrossProduct(Point p1, Point p2, Point p)
{ return ((p1.x-p.x)*(p2.y-p.y) - (p2.x-p.x)*(p1.y-p.y)); }

/* Watch out that if p touch the borders
is considered that p inside the polygon. */
bool isPointInside(vector<Point *>polygon, Point *p){
  int d=0,prevD;
  for(int i=0;i<polygon.size();i++)
  {
    prevD = CrossProduct((*polygon[i]),
                         (*polygon[((i+1)%polygon.size())]),
                         (*p));
                      if(d==0 && prevD!=0) d = prevD;
    if((prevD>0 && d<0) || (prevD<0 && d>0) )
      return false;
    if((prevD==0 && isInBoundingBox((*polygon[i]),
                                    (*polygon[((i+1)%polygon.size())]),
                                    (*p)))) return true;
  }
  return true;
}

int main(){
```

```
 vector<Point *> polygon;
 Point *pointInDiscordia;

 int i,c,nPoints;
 c = 1;
 while(!cin.eof()){
  cin>>nPoints;
  if(!nPoints) break;
  polygon.clear();
  Point *p;
  for(i=0;i<nPoints;i++){
   p = new Point;
   cin>>p->x>>p->y;
   polygon.push_back(p);
  }
  pointInDiscordia = new Point;
  cin>>pointInDiscordia->x>>pointInDiscordia->y;
  cout<<"Case #"<<(c++)<<":"<<endl;
  if(isPointInside(polygon,pointInDiscordia)){
   cout<<"The Discordia Point is INSIDE of ME.";
  } else {
   cout<<"The Discordia Point is OUTSIDE of ME.";
  }
  cout<<endl;
 }
 return 0;
}
```

PointOverSegment

```
#include<iostream>
using namespace std;

/*  Watch out the coordinates are integers! */
int maxi(int x, int y){
 return ((x>y)?x:y); }

int mini(int x, int y){
 return ((x<y)?x:y); }

struct Point{
 int x,y; };

struct Segment{
 Point p1,p2; };

bool isInBoundingBox(Segment s, Point p){
 if((mini(s.p1.x,s.p2.x)<=p.x && p.x<=maxi(s.p1.x,s.p2.x)) &&
   (mini(s.p1.y,s.p2.y)<=p.y && p.y<=maxi(s.p1.y,s.p2.y)))
  return true;
 return false;
```

```
}

/*  The cross product: (pk-pi) X (pj-pi): Is zero if
parallel segments*/
int CrossProduct(Point pi, Point pj, Point pk){
  return ((pk.x-pi.x)*(pj.y-pi.y) - (pj.x-pi.x)*(pk.y-pi.y));
}

bool isPointOverSegment(Segment s1, Point p)
{
  if( (CrossProduct(s1.p1, s1.p2, p)==0
                        && isInBoundingBox(s1,p)) ) return true;
  return false;
}

istream &operator>>(istream &is, Point &p){
  is>>p.x>>p.y;
  return is;
}

istream &operator>>(istream &is, Segment &s){
  is>>s.p1>>s.p2;
  return is;
}

int main(){
  int c;
  c = 0;
  Segment s1;
  Point p;
  while(!cin.eof()){
    cin>>s1; // using operator oeverload!  :)
    if(cin.eof()) break;
    cin>>p; // using operator oeverload!  :)
    cout<<"Case #"<<++c<<":"<<endl;
    if(isPointOverSegment(s1,p))
      cout<<"Yes.";
    else
      cout<<"No.";
    cout<<endl;
  }
  return 0;
}
```

SegmentIntersection

```
#include<iostream>
using namespace std;

/*  Watch out the coordinates are integers! */
int maxi(int x, int y){
```

```
  return ((x>y)?x:y); }

int mini(int x, int y){
  return ((x<y)?x:y); }

struct Point{
  int x,y; };

struct Segment{
  Point p1,p2; };

bool isInBoundingBox(Segment s, Point p){
  if((mini(s.p1.x,s.p2.x)<=p.x && p.x<=maxi(s.p1.x,s.p2.x)) &&
     (mini(s.p1.y,s.p2.y)<=p.y && p.y<=maxi(s.p1.y,s.p2.y)))
    return true;
  return false;
}

/*  The cross product: (pk-pi) X (pj-pi): Is zero if
parallel segments*/
int CrossProduct(Point pi, Point pj, Point pk){
  return ((pk.x-pi.x)*(pj.y-pi.y) - (pj.x-pi.x)*(pk.y-pi.y));
}

bool isIntersection(Segment s1, Segment s2){
  int d1,d2,d3,d4;
  d1 = CrossProduct(s1.p1, s1.p2, s2.p1);
  d2 = CrossProduct(s1.p1, s1.p2, s2.p2);
  d3 = CrossProduct(s2.p1, s2.p2, s1.p1);
  d4 = CrossProduct(s2.p1, s2.p2, s1.p2);

  if( ((d1>0 && d2<0) || (d1<0 && d2>0)) &&
      ((d3>0 && d4<0) || (d3<0 && d4>0)) )
    return true;

  if((d1==0 && isInBoundingBox(s1,s2.p1)) ||
     (d2==0 && isInBoundingBox(s1,s2.p2)) ||
     (d3==0 && isInBoundingBox(s2,s1.p1)) ||
     (d4==0 && isInBoundingBox(s2,s1.p2)) )
    return true;

  return false;
}

istream &operator>>(istream &is, Point &p){
  is>>p.x>>p.y;
  return is;
}

istream &operator>>(istream &is, Segment &s){
  is>>s.p1>>s.p2;
  return is;
```

```
}

int main(){
  int c;
  c = 0;
  Segment s1,s2;
  while(!cin.eof()){
    cin>>s1; // using operator oeverload!  :)
    if(cin.eof()) break;
    cin>>s2; // using operator oeverload!  :)
    cout<<"Case #"<<++c<<":"<<endl;
    if(isIntersection(s1,s2))
      cout<<"Yes.";
    else
      cout<<"No.";
    cout<<endl;
  }
  return 0;
}
```

ThreePointPerpendicularTest

```
#include<iostream>
using namespace std;

/*  Watch out the coordinates are integers! */
struct Point{
  int x,y; };

struct Segment{
  Point p1,p2; };

/*  The point product (pk-pi)(pj-pi): Is zero if
perpendicular segments*/
int PointProduct(Point pi, Point pj, Point pk){
  return ((pk.x-pi.x)*(pj.x-pi.x) + (pk.y-pi.y)*(pj.y-pi.y));
}

bool ThreePointPerpendicularTest(Point p1, Point p2, Point p3, Segment *sr1, Segment *sr2){
  Segment s1, s2, s3;
  s1.p1 = p1;
  s1.p2 = p2;
  s2.p1 = p2;
  s2.p2 = p3;
  s3.p1 = p3;
  s3.p2 = p1;

  if(PointProduct(p2,p1,p3) == 0){
                        (*sr1)=s1;
                        (*sr2)=s2;
                        return true;
```

```
  }else if(PointProduct(p3,p2,p1) == 0){
                        (*sr1)=s2;
                        (*sr2)=s3;
                        return true;
  }else if(PointProduct(p1,p3,p2) == 0){
                        (*sr1)=s3;
                        (*sr2)=s1;
                        return true;
  }
  return false;
}

istream &operator>>(istream &is, Point &p){
  is>>p.x>>p.y;
  return is;
}

istream &operator>>(istream &is, Segment &s){
  is>>s.p1>>s.p2;
  return is;
}

ostream &operator<<(ostream &os, Point &p){
  os<<"("<<p.x<<","<<p.y<<")";
  return os;
}

ostream &operator<<(ostream &os, Segment &s){
  os<<s.p1<<"-"<<s.p2;
  return os;
}
int main(){
  int c;
  c = 0;
  Segment s1,s2;
  Point p1,p2,p3;
  while(!cin.eof()){
    cin>>p1; // using operator overload!  :)
    if(cin.eof()) break;
    cin>>p2; // using operator overload!  :)
    if(cin.eof()) break;
    cin>>p3; // using operator overload!  :)
    cout<<"Case #"<<++c<<":"<<endl;
    if(ThreePointPerpendicularTest(p1,p2,p3,&s1,&s2))
      cout<<"Yes. Segments are "<<s1<<" y "<<s2;
    else
      cout<<"No.";
    cout<<endl;
  }
  return 0;
}
```

Grafos

BFS

```
#include<iostream>
#include<stack>
#define MAX 100

using namespace std;

queue<int> myStack;
int G[MAX][MAX];
int visit[MAX];
int V, E;


void bfs(int s) {
    int i, j, node;
    memset(visit, 0, sizeof(visit));
    myStack.push(s);

    while(!myStack.empty())
    {
        node = myStack.top();
        myStack.pop();
        if(visit[node]) continue;
        visit[node] = 1;
        cout << node << " ";

        for(i=0; i<V; i++)
            if(G[node][i]) myStack.push(i);
    }

}

int main() {
   memset(visit, 0, sizeof(visit));
   bfs(0);
   return 0;
}
```

BPM

```
/**
 *  ////////////////////////
 *  // BIPARTITE MATCHING //
 *  ////////////////////////
 *
 * This file is part of my library of algorithms found here:
```

```
 *     http://www.palmcommander.com:8081/tools/
 * LICENSE:
 *     http://www.palmcommander.com:8081/tools/LICENSE.html
 * Copyright (c) 2003
 * Contact author:
 *     igor at cs.ubc.ca
 **/

/**********************
 * Bipartite matching * O(m*n^2))
 **********************
 * Given a bipartite graph represented as an m-by-n matrix, where graph[i][j]
 * is true iff there is an edge from pigeon i to hole j, computes the maximum
 * number of pigeons that can find a hole (one per pigeon) and an optimal
 * assignment.
 *     Formally, this is a stripped down version of Ford-Fulkerson with
 * DFS used to find an augmenting path. DFS does the job quickly because
 * capacities can only be 0 or 1.
 *
 * PARAMETERS:
 *     - graph: adjacency matrix as above.
 * RETURNS:
 *     - an integer corresponding to the number of assignments
 *     - matchL[m]: for each pigeon, a hole or -1
 *     - matchR[n]: for each hole, a pigeon or -1
 * DETAILS:
 *     - graph[m][n], matchL[n], matchR[m] and seen[m] are global arrays
 *     - main() initializes matchL[] and matchR[] to all -1.
 *     - main() does a loop over all pigeons i and in each iteration
 *         - clears seen[] to all 0
 *         - calls bpm(i) and increments the maxflow counter
 *     - bpm(i) returns true iff pigeon i can be assigned a hole
 * ACKNOWLEDGEMENTS:
 *     - Thanks to tjq from TopCoder for a reference implementation.
 * FIELD TESTING:
 *     - Valladolid 10080: Gopher (II)
 *     - Valladolid 10092: The Problem with the Problem Setter
 *     - Valladolid 670:   The dog task
 *     - Valladolid 259:   Software Allocation
 * #include <string.h>
 **/

#include <string.h>

#define M 128
#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;
}

int main()
{
   // 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++;
   }

   // 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

   return 0;
}
```

DFS

```
#include<iostream>
#include<queue>
#define MAX 100

using namespace std;

queue<int> myQueue;
int G[MAX][MAX];
int visit[MAX];
int V, E;
```

```
void dfs(int s) {
    int i, j, node;
    memset(visit, 0, sizeof(visit));
    myQueue.push(s);

    while(!myQueue.empty())
    {
        node = myQueue.front();
        myQueue.pop();
        if(visit[node]) continue;
        visit[node] = 1;
        cout << node << " ";

        for(i=0; i<V; i++)
            if(G[i][node]) myQueue.push(i);
        for(j=0; j<V; j++)
            if(G[node][j]) myQueue.push(j);
    }

}

int main() {
   memset(visit, 0, sizeof(visit));
   dfs(0);
   return 0;
}
```

Dijkstra

```
#include<queue>
#include<iostream>
#include<math.h>
using namespace std;

#define MAXINT (int)(pow(2,31)-1)
#define MAX 100


int graph[MAX][MAX];
int total;


int distances[MAX];
int father[MAX];
bool visit[MAX];

void dijkstra(int start)
{
  priority_queue<pair<int,int> > queue;
  pair <int,int> nodotmp;
```

```
  int i, j;

  for (int i=1; i<=total; i++) {
    distances[i] = MAXINT;
    father[i] = -1;
    visit[i] = false;
  }

  distances[start] = 0;
  queue.push(pair <int,int> (distances[start], start));

  while(!queue.empty()) {
    nodotmp = queue.top();
    queue.pop();
    i = nodotmp.second;
    if (!visit[i]) {
      visit[i] = true;
      for (j = 1; j<=total; j++)
        if (!visit[j] && graph[i][j] > 0 && distances[i] + graph[i][j] < distances[j]) {
          distances[j] = distances[i] + graph[i][j];
          father[j] = i;
          queue.push(pair <int,int>(-distances[j], j));
        }
    }
  }
}


void getPath(int end) {
  cout << end << " ";
  while (father[end]!= -1) {
    cout << father[end] << " ";
    end = father[end];
  }
  cout << endl;
}

int main()
{
   int a, b, c;
   int tedges;
   memset(graph, 0, sizeof(graph));
   cin >> total >> tedges;
   for (int i=0; i<tedges; i++) {
     cin >> a >> b >> c;
     graph[a][b] = c;
   }
   for(int i=1; i<=total; i++) {
      for(int j=1; j<=total; j++)
         printf("%d ", graph[i][j]);
      printf("\n");
   }
```

```
  dijkstra(1);
  getPath(3);

  /*for (int i=1; i<=total; i++) {
    dijkstra(i);
    for(int i=1; i<=total; i++)
      cout << distances[i] << " ";
    cout << endl;
    for(int i=1; i<=total; i++)
      cout << father[i] << " ";
    cout << endl;
    getPath(5);
  }*/
  return 0;
}
```

Floyd

```
#include<iostream>
#define MAX 100

using namespace std;

int G[MAX][MAX];
int predecessor[MAX];
int V, E;

void floyd()
{
  for(int i=0; i<=V; i++)
    for(int j=0; j<=V; j++)
      if(G[j][i])
        for(int k=1; k<=V; k++)
          if(G[i][k] > 0)
            if(!G[j][k] || (G[j][i] + G[i][k] < G[j][k])) {
              G[j][k] = G[j][i] + G[i][k];
              predecessor[j][k] = predecessor[i][k];
            }
}

void print_path (int i, int j) {
  if (i!=j)
    print_path(i,p[i][j]);
  cout << j << " ";
}

int main() {
  read();
  return 0;
}
```

Fordfulkerson

```
/**
 *  //////////////////
 *  // MAXIMUM FLOW //
 *  //////////////////
 *
 * This file is part of my library of algorithms found here:
 *     http://www.palmcommander.com:8081/tools/
 * LICENSE:
 *     http://www.palmcommander.com:8081/tools/LICENSE.html
 * Copyright (c) 2004
 * Contact author:
 *     igor at cs.ubc.ca
 **/

/****************
 * Maximum flow * (Ford-Fulkerson on an adjacency matrix)
 ****************
 * Takes a weighted directed graph of edge capacities as an adjacency
 * matrix 'cap' and returns the maximum flow from s to t.
 *
 * PARAMETERS:
 *     - cap (global): adjacency matrix where cap[u][v] is the capacity
 *         of the edge u->v. cap[u][v] is 0 for non-existent edges.
 *     - n: the number of vertices ([0, n-1] are considered as vertices).
 *     - s: source vertex.
 *     - t: sink.
 * RETURNS:
 *     - the flow
 *     - fnet contains the flow network. Careful: both fnet[u][v] and
 *         fnet[v][u] could be positive. Take the difference.
 *     - prev contains the minimum cut. If prev[v] == -1, then v is not
 *         reachable from s; otherwise, it is reachable.
 * DETAILS:
 * FIELD TESTING:
 *     - Valladolid 10330: Power Transmission
 *     - Valladolid 653:   Crimewave
 *     - Valladolid 753:   A Plug for UNIX
 *     - Valladolid 10511: Councilling
 *     - Valladolid 820:   Internet Bandwidth
 *     - Valladolid 10779: Collector's Problem
 * #include <string.h>
 * #include <queue>
 **/

#include <string.h>

// the maximum number of vertices
#define NN 1024
```

```
// adjacency matrix (fill this up)
int cap[NN][NN];

// flow network
int fnet[NN][NN];

// BFS
int q[NN], qf, qb, prev[NN];

int fordFulkerson( int n, int s, int t )
{
   // init the flow network
   memset( fnet, 0, sizeof( fnet ) );

   int flow = 0;

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

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

      // get the bottleneck capacity
      int bot = 0x7FFFFFFF;
      for( int v = t, u = prev[v]; u >= 0; v = u, u = prev[v] )
         bot <?= cap[u][v] - fnet[u][v] + fnet[v][u];

      // update the flow network
      for( int v = t, u = prev[v]; u >= 0; v = u, u = prev[v] )
         fnet[u][v] += bot;

      flow += bot;
   }

   return flow;
}

//----------------- EXAMPLE USAGE -----------------
int main()
{
   memset( cap, 0, sizeof( cap ) );
   int numVertices = 100;

   // ... fill up cap with existing edges.
```

```
    // if the edge u->v has capacity 6, set cap[u][v] = 6.

    cout << fordFulkerson( numVertices, s, t ) << endl;

    return 0;
}
```

Kruskal

```
#include<iostream>
#include<vector>
#include<algorithm>
#define max 100

using namespace std;

struct Edge{
  int vi,vj,w;
};

bool lequal(Edge const* t1, Edge const* t2){
  return (t1->w<t2->w);
}

int V,E;
vector <Edge*>edges;
vector <Edge*>mst;
int cicles[max];

int main(){

  int i,W,number,c;
  Edge *e;

  c=1;
  while(true){
    edges.clear();
    mst.clear();
    cin>>V>>E;
    if(!V && !E) break;

    for(i=0;i<E;i++){
      e = new Edge;
      cin>>e->vi>>e->vj>>e->w;
      edges.push_back(e);
    }

    sort(edges.begin(),edges.end(),lequal);
    for(i=0;i<V;i++) cicles[i] = i;
```

```
  while(mst.size()<(V-1) && edges.size()){
    if(cicles[edges[0]->vi]!=cicles[edges[0]->vj]){
          number = cicles[edges[0]->vj];
          for(i=0;i<V;i++) {
            if(cicles[i]==number)
              cicles[i] = cicles[edges[0]->vi];
          }
          mst.push_back(edges[0]);
    }
    edges.erase(edges.begin(),edges.begin()+1);
  }
  W = 0;
  for(i=0;i<mst.size();i++) {
    W+=mst[i]->w;
  }
  cout<<"Case "<<c++<<":"<<endl;
  cout<<"The Minimun Spanning Tree has cost: "<<W<<endl;
 }

 return 0;
}
```

Maxflow

```
#include<iostream>
#define MAX 100

using namespace std;

int graph[MAX][MAX];
int queue[MAX];
int head, tail;
int parent[MAX];
int V, E;
int s, t, fTotal;
int F[MAX][MAX];


//Breadth First Search
bool reachable(int s, int t) {
  bool found = false;
  int vq;
  head = tail = 0;
  memset(parent, 255, sizeof(parent));
  queue[tail++] = s;
  parent[s] = s;

  while(head < tail && !found) {
    vq = queue[head++];
    for(int i=0; i<V; i++) {
      //Parents also made the function as visit vector
```

```
      if(graph[vq][i] && parent[i] == -1) {
        queue[tail++] = i;
        parent[i] = vq;

        if(i == t) {
          found = true;
          break;
        }
      }
    }
  }
  return found;
}

void maxflow() {
  int vj, min;
  fTotal = 0;
  while(reachable(s, t)) {
    //Gets the minimum possible capacity in edges of the path s to t
    min = graph[parent[t]][t];
    vj = t;
    while(parent[vj] != vj) {
      if(graph[parent[vj]][vj] < min)
        min = graph[parent[vj]][vj];
        vj = parent[vj];
    }

    vj = t;
    while(parent[vj] != vj) {
      graph[parent[vj]][vj] -= min;
      graph[vj][parent[vj]] += min;
      F[parent[vj]][vj] += min;
      vj = parent[vj];
    }
    fTotal += min;
  }
}

bool read() {
  cin >> V >> E >> s >> t;
  if(!V && !E) return false;
  memset(graph, 0, sizeof(graph));
  //memset(parent, -1, sizeof(parent));
  memset(queue, 0, sizeof(queue));
  memset(F, 0, sizeof(F));


  int v1, v2, val;
  for(int i=0; i<E; i++) {
    cin >> v1 >> v2 >> val;
    graph[v1][v2] = val;
  }
```

```
  return true;
}

int main() {

  while(read()) {
    maxflow();
    cout << "The max flow from s to t is : " << fTotal << endl;
  }
}
```

Minimum coloring

```
#include<iostream>
#define MAX 100

using namespace std;

int graph[MAX][MAX];
int n;              //Number of vertices
int color[MAX];

int colorea(int v, int nc) {
   int res, i, j, c;

   if(v == n)   return 1;

   for(c = 0; c<nc; c++) {
      for(i=0; i<n; i++)
         if(graph[v][i] && color[i] == c) break;
      if(i==n) {
         color[v] = c;
         if(colorea(v+1, nc)) return 1;
      }
   }
}

int main() {
   int nc = 4;
   memset(color, 255, sizeof(color));
   colorea(0, nc);
   return 0;
}
```

Minimum Cut

```
/**
 *  ////////////////////////////
 *  // Stoer-Wagner Minimum Cut //
 *  ////////////////////////////
```

```
*
* MAIN FUNCTION: minCut( n )
*     Takes an undirected, weighted graph and returns the weight
*     of the minimum cut in it. A cut is a set of edges that,
*     when removed, disconnects the graph. A minimum cut is a
*     cut of minimum total weight.
* ALGORITHM:
*     This is a O(n^3) implementation of the Stoer-Wagner
*     deterministic algorithm (no randomization is required).
* FIELD TESTING:
*     - UVa 10989: Bomb, Divide and Conquer
*
* LAST MODIFIED:
*     January 31, 2006
*
* This file is part of my library of algorithms found here:
*     http://shygypsy.com/tools/
* LICENSE:
*     http://shygypsy.com/tools/LICENSE.html
* Copyright (c) 2006
**/

// 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++ )
      {
         // 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 i = 0; i < n; i++ )
               g[v[i]][prev] = g[prev][v[i]] += g[v[zj]][v[i]];
            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;
}
```

StronglyConnectedComponents

```
#include<iostream>
#define MAX 100

using namespace std;

int graph[MAX][MAX];
int visit[MAX];
int V, E;
```

```
int orden[MAX];
int component[MAX];
int T;
int componentId;

void dfs(int v) {
  visit[v] = 1;
  for(int i=0; i<V; i++)
    if(graph[v][i] && !visit[i])
      dfs(i);
  orden[T--] = v;
}

void dfsInverse(int v, int componentId) {
  visit[v] = 1;
  component[v] = componentId;
  for(int i=0; i<V; i++)
    if(graph[i][v] && !visit[i])
      dfsInverse(i, componentId);
}

void findStronglyConnectedComponents() {
  T = V-1;
  for(int i=0; i<V; i++)
    if(!visit[i])
      dfs(i);
  memset(visit,0, sizeof(visit));
  componentId = 0;
  for(int i=0; i<V; i++)
    if(!visit[orden[i]])
      dfsInverse(orden[i], componentId++);
}

bool read() {
  cin >> V >> E;
  if(!V && !E) return false;
  memset(graph, 0, sizeof(graph));
  memset(visit, 0, sizeof(visit));
  memset(orden, 0, sizeof(orden));
  memset(component, 0, sizeof(component));

  int v1, v2;
  for(int i=0; i<E; i++) {
    cin >> v1 >> v2;
    graph[v1][v2] = 1;
  }
  return true;
}

int main() {
  while(read()) {
```

```
    findStronglyConnectedComponents();
    int j=0;
    while(j<componentId) {
      cout << " component " << (j+1) << endl << " ";
      for(int i=0; i<V; i++) {
        if(j==component[i])
          cout << i << " ";
      }
      cout << endl;
      j++;
    }
  }
}
```

Topsort

```
#include<iostream>
#include<queue>
#define MAX 100

using namespace std;

int graph[MAX][MAX];

int indegree[MAX];  //Indegree of each vertex
int sorted[MAX];
int V, E;

void computeInDegrees() {
  memset(indegree, 0, sizeof(indegree));
  for(int i=0; i<V; i++)
    for(int j=0; j<V; j++)
      if(graph[j][i])
        indegree[i]++;
}

void topsort() {
  queue<int> zeroin;   //Vertices of indegree 0
  int current, next;   //Current and next vertex

  computeInDegrees();

  for(int i=0; i<V; i++)
    if(indegree[i] == 0)
      zeroin.push(i);

  int j=0;
  while(!zeroin.empty()) {
    current = zeroin.front();
    zeroin.pop();
```

```
    sorted[j] = current;
    for(int i=0; i<V; i++) {
      cout << current << " " << i << " " << graph[current][i] << endl;
      if(graph[current][i]) {
        next = i;
        indegree[next]--;
        if(indegree[next] == 0) {
          zeroin.push(next);
        }
      }
    }
    j += 1;
  }

  if( j != V) {
    cout << "Not a DAG -- only " << j << " vertices found" << endl;
    for(int i=0; i<V; i++)
      cout << sorted[i] << " ";
    cout << endl;
  }
  else {
    for(int i=0; i<V; i++)
      cout << sorted[i] << " ";
    cout << endl;
  }
}

void read()
{
  cin >> V >> E;
  for(int i=0; i<E; i++) {
    int a, b;
    cin >> a >> b;
    graph[a][b] = 1;
  }
}
int main() {
  read();
  topsort();
  return 0;
}
```

Grafo templete (Dudu)

```
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<iomanip>
#include<string>
#include<map>
#include<list>
#include<queue>
```

```
#define MAX 101

using namespace std;

struct dist{
	int dest;
	double valor;

};

class nodo{
	public:
	double custo;
	int indice;
	list<dist> adj;

	int anterior; //usado no bellman ford

	nodo (){
		custo = 0;
	}

	void clean(){
		custo = 0;
		adj.clear();
	}

	void zeracusto(){
		custo = 0;
	}

	void infinitocusto(){
		custo = 1000000000;
	}

	bool operator<(const nodo &e) const{
		return e.custo < custo; //pega o nó com menor custo primeiro
		//return e.custo > custo; //pega o nó com maior custo primeiro
	}

	void print (){
		cout << "(" << indice <<  ") " << "=>  custo= " << custo << endl;

	}
};

nodo grafo[MAX];
bool visitado[MAX];

void criaaresta(nodo grafo[], int orig, int dest, double p){
	dist d;
	d.dest = dest;
```

```
	d.valor = p;
	grafo[orig].adj.push_back(d);
}

double dijkstra(nodo g[], int o, int d){
	priority_queue<nodo> q;
	q.push(g[o]);
	while (!q.empty()){
		nodo n = q.top();
		q.pop();

		if (n.indice == d){
			return n.custo;
		}

		visitado[n.indice] = true;

		list<dist>::iterator it;
		for (it = n.adj.begin(); it != n.adj.end(); it++){
			nodo prox = g[(*it).dest];
			if (!visitado[prox.indice]){
				prox.custo = n.custo + (*it).valor;
				q.push(prox);
			}


		}

	}

}

double bellman_ford(nodo g[], int o, int d, int nv){
	list<dist>::iterator it;
	for (int i = 1; i<=nv; i++){
		for (it = g[i].adj.begin(); it != g[i].adj.end(); it++){
			nodo *prox = &g[(*it).dest];
			if (prox->custo > g[i].custo * (*it).valor){
				prox->custo = g[i].custo * (*it).valor;
				//cout << prox->indice << " : custo =" << prox->custo;
				//cout << " anterior = " << i << endl;
				prox->anterior = i;
			}
		}
	}
	return g[d].custo;
}

double prim(nodo g[]){
		priority_queue<nodo> q;

		q.push(grafo[1]);
```

```
		double total = 0;

		while (!q.empty()){
			nodo n = q.top();

			q.pop();
			if (visitado[n.indice]) continue;
			n.print();
			total += n.custo;
			visitado[n.indice] = true;
			//cout << "Filhos : " << endl;
			list<dist>::iterator it;
			for (it = n.adj.begin(); it != n.adj.end(); it++){
				nodo prox = g[(*it).dest];
				if (!visitado[prox.indice]){
					prox.custo = (*it).valor;
					q.push(prox);
				}
			}
			//cout << endl;

		}
		return total;
}

int main ()
{
	int nv, na;
	int scen = 0;

	cin >> nv >> na;

	while (nv || na){
		for (int i = 1; i<MAX; i++){
			grafo[i].indice = i;
			visitado[i] = false;
			grafo[i].clean();
		}
		for (int i = 0 ; i<na; i++){
			int o,d;
			double p;
			cin >> o;
			cin >> d;
			cin >> p;
			criaaresta(grafo,o,d,p);
			criaaresta(grafo,d,o,p);
		}
		int origem, destino, total;
		cin >> origem >> destino;

		/* Bellman Ford
```

```
                for (int i = 1; i<=nv; i++){
                        if (i == origem) grafo[i].custo = 1;
                        else {
                                grafo[i].infinitocusto();
                        }
                        grafo[i].anterior = 0;
                }


                cout << "Scenario #" << ++scen << endl;
                cout << "Minimum Path (BF) = " ;
                cout << bellman_ford(grafo,origem,destino,nv) ;
                cout << endl << endl;

                */
                for (int i = 1; i<=nv; i++){
                        if (i == origem) grafo[i].custo = 1;
                        else {
                                grafo[i].infinitocusto();
                        }
                        grafo[i].anterior = 0;
                }
                cout << "Scenario #" << ++scen << endl;
                cout << "Minimum Path = " ;
                cout << dijkstra(grafo,origem,destino) ;
                cout << endl;
                /*for (int i = 1; i<MAX; i++){
                        grafo[i].zeracusto();
                        visitado[i] = false;
                }
                cout << "Minimun Tree = ";
                prim(grafo);*/
                cin >> nv >> na;
                cout << endl;
        }
   return 0;
}
```

Ford-Fukerson (Rafael)

```
/**
        Algoritmo implementado por Rafael Azevedo
        Problema do matching máximo em um grafo bipartido.
        Atribuição de cavalos à soldados
**/
#include <iostream>
#include <queue>
#include <stack>
#include <vector>
#include "limits.h"

#define MAX 300
```

```
using namespace std;

// Inicializa arrays de int com o valor dado
void init(int a[], int tam, int valor){
	for(int i=0; i<tam; ++i){
		a[i] = valor;
		}
}

//Inicializa arrays de bool com o valor dado
void init(bool a[], int tam, bool valor){
	for(int i=0; i<tam; ++i){
		a[i] = valor;
		}
}

//Imprime array de inteiros
void print(int a[], int tam){
	for(int i=0; i<tam; ++i)
		cout << a[i] << " ";
	cout << endl;
}

//Imprime elementos de um vector
void print(vector<int> v){
	for(int i=0; i<v.size(); ++i)
		cout << v[i] << " ";
	cout << endl;
}

//Classe que representa o grafo
class Grafo{
	public:
		int adj[MAX][MAX];
		int nVertices;

		Grafo(int n){
			nVertices = n;

			for(int i=0; i<nVertices; ++i){
				for(int j=0; j<nVertices; ++j){
					adj[i][j] = 0;
				}
			}
		}
		Grafo copy(){
			Grafo aux(nVertices);
			for(int i=0; i<nVertices; ++i){
				for(int j=0; j<nVertices; ++j){
					aux.adj[i][j] =adj[i][j];
				}
			}
```

```
                return aux;
        }
        void print(){
                for(int i=0; i<nVertices; ++i){
                        for(int j=0; j<nVertices; ++j){
                                cout << adj[i][j] << " ";
                        }
                        cout << endl;
                }
        }
};

//classe que representa os nos, e informacoes a respeito do mesmo
class estado{
        public:
                int city;
                int fMax;
                int pai;
           bool operator<(const estado &e)const{
         return fMax<e.fMax;
     }
};

//Busca em profundidade, usada para encontrar caminhos de aumento
int leva_fluxo(Grafo &g, int ini, int fim){
        //Inicializa arrays
        int *pai = new int[g.nVertices];
        init(pai, g.nVertices, -1);
        bool *visitado   = new bool[g.nVertices];
        init(visitado, g.nVertices, false);
        estado inicial = {ini, INT_MAX, -1};
        stack<estado> pilha;
        pilha.push(inicial);
        while(!pilha.empty()){
                estado top = pilha.top();
                pilha.pop();
                pai[top.city] = top.pai;
                visitado[top.city] = true;
                if(top.city == fim){
                        int ant = pai[fim];
                        while(ant!=-1){
                                g.adj[ant][fim]-=top.fMax;
                                g.adj[fim][ant]+=top.fMax;
                                fim = ant;
                                ant = pai[fim];
                        }
                        return top.fMax;
                }
                for(int i=0; i<g.nVertices; ++i)
                        if(g.adj[top.city][i] > 0 && !visitado[i])
                                pilha.push((estado){i, g.adj[top.city][i] < top.fMax ?
g.adj[top.city][i] : top.fMax, top.city});
```

```
        }
        return 0;
}

int maxFlow(Grafo &g, int ini, int fim){
        int allFlow = 0;
        int flow = leva_fluxo(g,ini,fim);
        while(flow!=0){
                allFlow+=flow;
                flow = leva_fluxo(g,ini,fim);
        }
        return allFlow;

}

// Observacao: Ao utilizar o algoritmo tomar cuidado para nao representar ausencia de aresta
// com o zero(0). Isso é representado por -1.
// Os vertices do grafo sao numerados de 0 a v-1

int main(){
        int n;
        int caso = 0;
        while(cin >> n){
            int m, k;
            cin >> m >> k;
            Grafo g(m+n+2);
            int aux;
            for(int i=0; i<n; ++i){
        cin >> aux;
        g.adj[0][i+1] = aux;
            }
            int u,v;
            for(int i=0; i<k; ++i){
        cin >> u >> v;
        g.adj[u][n+v] = 1;
            }
            for(int i=1; i<=m; ++i){
          g.adj[n+i][n+m+1] = 1;
            }
            cout << "Instancia " << ++caso << endl;
            cout << maxFlow(g,0, n+m+1) << endl << endl;
        }
}
```

Dijkstra (Rafael)

```
#include<iostream>
#include<string>
#include<vector>
#include<map>
#include<queue>
```

```
using namespace std;

//Nível do nó
int erdos[101];
//Recebe uma lista de adjacencia e o tamanho
void buscalargura(vector<int> v[], int tam){
		for(int i=0; i<101; ++i)
			erdos[i] = -1;


	queue<estado> fila;
	estado ini = (estado){1, 0};
	int nvisit = 0;

	fila.push(ini);

	while(!fila.empty()){
		ini = fila.front();
		fila.pop();
		//cout << "Desenfila " << ini.no <<" " <<  ini.custo << endl;
		if(erdos[ini.no] == -1){
			erdos[ini.no] = ini.custo;
			nvisit++;
		}

		if(nvisit==tam)
			return;

		for(int i=0; i<v[ini.no].size(); ++i)
			if(erdos[v[ini.no][i]] == -1){
				estado emp = (estado){v[ini.no][i], ini.custo+1};
				//emp.print();
				fila.push(emp);

			}

	}

}
```

Busca em largura (Rafael)

```
#include<iostream>
#include<string>
#include<vector>
#include<map>
#include<queue>

using namespace std;
```

```
struct estado{
	int no;
	int custo;

	void print(){
		cout << " Nó " << no << " Custo " << custo << endl;
	}
};

//Nível do nó
int erdos[101];
//Recebe uma lista de adjacencia e o tamanho
void buscalargura(vector<int> v[], int tam){
		for(int i=0; i<101; ++i)
			erdos[i] = -1;
   queue<estado> fila;
	estado ini = (estado){1, 0};
	int nvisit = 0;

	fila.push(ini);

	while(!fila.empty()){
		ini = fila.front();
		fila.pop();

		if(erdos[ini.no] == -1){
			erdos[ini.no] = ini.custo;
			nvisit++;
		}
		if(nvisit==tam)
			return;

		for(int i=0; i<v[ini.no].size(); ++i)
			if(erdos[v[ini.no][i]] == -1){
				estado emp = (estado){v[ini.no][i], ini.custo+1};
				//emp.print();
				fila.push(emp);

			}
	}

}
```

Numerico

BaseConverter

```
/**
 * BaseConverter.c - Convert decimal to any base, and any base to decimal
 *                                   numbers. Only for positive integers, and bases
between
 *                                   2 and 36.
 */
#include<stdio.h>
#include<string.h>
#include<math.h>

char digits[] = {'0','1', '2','3','4','5','6','7','8','9','A','B','C','D',
            'E','F','G','H','I','J','K','L','M','N','O','P','Q','R',
            'S','T','U','V','W','X','Y','Z'};

char * decimalToAny(int number, int base)
{
          int index;
          int digitValue;
          char *result = new char[40];
          int counter = 0;

          if((number > 0) && (base >= 2 && base <= 36))
          while(number) {
          digitValue = number % base;
         number /= base;
         result[counter++] = digits[digitValue];
     }
          result[strlen(result)] = '\0';
          return result;
}

int anyToDecimal(char * number, int base)
{
          int n = strlen(number);
          int index;
          int counter = 0;
          int result = 0;
          for(index = n-1; index>=0; index--)  {
                    result += (number[index]>='A')?(number[index]-'A'+10):(number[index]-
'0')*(int)pow(base,counter++);
          }
          return result;
}

int main() {
```

```
	char * ea = decimalToAny(255, 2);
	printf("%s\n", ea);
	printf("%d\n", anyToDecimal(ea, 2));
	return 0;
}
```

Bigmod

```
/**
 * BigMod.cpp - Calculate the remainder or modulus of a big number b^p using
 *                                  a divide and conquer approach based on the formula:
 *
 *                                  (A*B*C) mod n = (A mod n) * (B mod n) * (C mod n).
 */
#include<math.h>
#include<iostream>

using namespace std;

long square(long n) {
  return n*n;
}

long bigMod(long b, long p, long m) {
	if(p == 0)
		return 1;
	else if(p%2 == 0)
		return square(bigMod(b, p/2, m)) %m;
	else
		return ((b%m) * bigMod(b, p-1, m)) % m;

}

int main() {
	long b, p, m;

	b = 3;
	p = 18132;
	m = 17;
	cout << bigMod(b, p, m) << endl;

	b = 17;
	p = 1765;
	m = 3;
	cout << bigMod(b, p, m) << endl;

	b = 2374859;
	p = 3029382;
	m = 36123;
	cout << bigMod(b, p, m) << endl;
}
```

Bignum

```
#include<stdio.h>
#include<string.h>
#define MAXDIGITS       1001

#define PLUS            1
#define MINUS           -1
#define abs(a) a>=0?a:-a

typedef struct {
  char digits[MAXDIGITS];
  int signbit;
  int lastdigit;
} bignum;

void print_bignum(bignum *);
void int_to_bignum(int, bignum *);
void char_to_bignum(char *, bignum *);
void initialize_bignum(bignum *);
int max(int, int);
void subtract_bignum(bignum *, bignum *, bignum *);
void add_bignum(bignum *, bignum *, bignum *);
int compare_bignum(bignum *, bignum *);
void zero_justify(bignum *);
void divide_bignum(bignum *, bignum *, bignum *);
void multiply_bignum(bignum *, bignum *, bignum *);

void print_bignum(bignum *n)
{
  int i;
  if(n->signbit == MINUS) printf("- ");
  for(i=n->lastdigit; i>=0; i--)
    printf("%c",'0'+ n->digits[i]);
  printf("\n");
}

void int_to_bignum(int s, bignum *n)
{
  int i;
  int t;
  if (s >= 0)
    n->signbit = PLUS;
  else
    n->signbit = MINUS;
  for(i=0; i<MAXDIGITS; i++)
    n->digits[i] = (char) 0;
  n->lastdigit = -1;
  t = abs(s);
  while(t > 0) {
```

```
    n->lastdigit ++;
    n->digits[ n->lastdigit ] = (t % 10);
    t = t / 10;
  }
  if(s == 0)
    n->lastdigit = 0;
}

void char_to_bignum(char * s, bignum *n)
{
  int i;
  int j = 0;
  n->lastdigit = -1;
  for(i=0; i<MAXDIGITS; i++)
    n->digits[i] = (char) 0;
  if(s[0] == '-') {
    n->signbit = MINUS;
    for(i=strlen(s)-1; i>=1; i--) {
      n->lastdigit++;
      n->digits[j++] = s[i]-'0';
    }
    n->lastdigit = strlen(s)-2;
  }
  else {
    n->signbit = PLUS;
    for(i=strlen(s)-1; i>=0; i--) {
      n->lastdigit++;
      n->digits[j++] = s[i]-'0';
    }
    n->lastdigit = strlen(s)-1;
  }
}

void initialize_bignum(bignum *n)
{
  int_to_bignum(0,n);
}

int max(int a, int b)
{
  if(a > b) return(a); else return(b);
}

void add_bignum(bignum *a, bignum *b, bignum *c)
{
  int carry;
  int i;
  initialize_bignum(c);
  if(a->signbit == b->signbit) c->signbit = a->signbit;
  else {
    if(a->signbit == MINUS) {
      a->signbit = PLUS;
```

```
      subtract_bignum(b,a,c);
      a->signbit = MINUS;
    }
    else {
      b->signbit = PLUS;
      subtract_bignum(a,b,c);
      b->signbit = MINUS;
    }
    return;
  }
  c->lastdigit = max(a->lastdigit,b->lastdigit)+1;
  carry = 0;

  for(i=0; i<=(c->lastdigit); i++) {
    c->digits[i] = (char) (carry+a->digits[i]+b->digits[i]) % 10;
    carry = (carry + a->digits[i] + b->digits[i]) / 10;
  }
  zero_justify(c);
}

void subtract_bignum(bignum *a, bignum *b, bignum *c)
{
  int borrow;
  int v;
  int i;
  initialize_bignum(c);
  if((a->signbit == MINUS) || (b->signbit == MINUS)) {
    b->signbit = -1 * b->signbit;
    add_bignum(a,b,c);
    b->signbit = -1 * b->signbit;
    return;
  }
  if(compare_bignum(a,b) == PLUS) {
    subtract_bignum(b,a,c);
    c->signbit = MINUS;
    return;
  }
  c->lastdigit = max(a->lastdigit,b->lastdigit);
  borrow = 0;

  for(i=0; i<=(c->lastdigit); i++) {
    v = (a->digits[i] - borrow - b->digits[i]);
    if (a->digits[i] > 0)
      borrow = 0;
    if (v < 0) {
      v = v + 10;
      borrow = 1;
    }
    c->digits[i] = (char) v % 10;
  }
  zero_justify(c);
}
```

```
int compare_bignum(bignum *a, bignum *b)
{
  int i;

  if((a->signbit == MINUS) && (b->signbit == PLUS)) return(PLUS);
  if((a->signbit == PLUS) && (b->signbit == MINUS)) return(MINUS);

  if(b->lastdigit > a->lastdigit) return (PLUS * a->signbit);
  if(a->lastdigit > b->lastdigit) return (MINUS * a->signbit);

  for(i = a->lastdigit; i>=0; i--) {
    if(a->digits[i] > b->digits[i]) return(MINUS * a->signbit);
    if(b->digits[i] > a->digits[i]) return(PLUS * a->signbit);
  }
  return 0;
}

void zero_justify(bignum *n)
{
  while((n->lastdigit > 0) && (n->digits[ n->lastdigit ] == 0))
    n->lastdigit --;

  if((n->lastdigit == 0) && (n->digits[0] == 0))
    n->signbit = PLUS;
}

void digit_shift(bignum *n, int d)
{
  int i;
  if((n->lastdigit == 0) && (n->digits[0] == 0)) return;
  for(i=n->lastdigit; i>=0; i--)
    n->digits[i+d] = n->digits[i];
  for (i=0; i<d; i++) n->digits[i] = 0;
  n->lastdigit = n->lastdigit + d;
}

void multiply_bignum(bignum *a, bignum *b, bignum *c)
{
  bignum row;
  bignum tmp;
  int i,j;

  initialize_bignum(c);
  row = *a;
  for(i=0; i<=b->lastdigit; i++) {
    for(j=1; j<=b->digits[i]; j++) {
      add_bignum(c,&row,&tmp);
      *c = tmp;
    }
    digit_shift(&row,1);
  }
```

```
  c->signbit = a->signbit * b->signbit;
  zero_justify(c);
}

void divide_bignum(bignum *a, bignum *b, bignum *c)
{
  bignum row;
  bignum tmp;
  int asign, bsign;
  int i,j;

  initialize_bignum(c);
  c->signbit = a->signbit * b->signbit;
  asign = a->signbit;
  bsign = b->signbit;
  a->signbit = PLUS;
  b->signbit = PLUS;

  initialize_bignum(&row);
  initialize_bignum(&tmp);

  c->lastdigit = a->lastdigit;

  for(i=a->lastdigit; i>=0; i--) {
    digit_shift(&row,1);
    row.digits[0] = a->digits[i];
    c->digits[i] = 0;
    while (compare_bignum(&row,b) != PLUS) {
      c->digits[i] ++;
      subtract_bignum(&row,b,&tmp);
      row = tmp;
    }
  }
  zero_justify(c);
  a->signbit = asign;
  b->signbit = bsign;
}

int main()
{
        bignum n1,n2,n3,zero;
        char a[100], b[100];
        while (scanf("%s %s\n",&a,&b) != EOF) {
                printf("a = %s    b = %s\n",a,b);
                char_to_bignum(a,&n1);
                char_to_bignum(b,&n2);

                add_bignum(&n1,&n2,&n3);
                printf("addition -- ");
                print_bignum(&n3);

                printf("compare_bignum a ? b = %d\n",compare_bignum(&n1, &n2));
```

```
                subtract_bignum(&n1,&n2,&n3);
                printf("subtraction -- ");
                print_bignum(&n3);

        multiply_bignum(&n1,&n2,&n3);
                printf("multiplication -- ");
        print_bignum(&n3);

                int_to_bignum(0,&zero);
                if (compare_bignum(&zero, &n2) == 0)
                        printf("division -- NaN \n");
        else {
                        divide_bignum(&n1,&n2,&n3);
                        printf("division -- ");
                print_bignum(&n3);
                }
                printf("-------------------------\n");
        }
  return 0;
}
```

DivisorsCalculator

```
#include<iostream>
#include<math.h>
#define MAX 100
using namespace std;

using namespace std;

int factors[MAX];
int times[MAX];
int total = 0;
int divisors[MAX];
int totalDivisors = 0;

long insertFactor(long lastFact, long actualFact)
{
  if(lastFact != actualFact) {
    factors[total] = actualFact;
    times[total]  = 1;
  }
  else
    times[total]++;
  total += 1;
  return actualFact;
}

void primeFactorization(long num)
```

```
{
    long i; //Counter
    long c; //Remaining product to factor
    long last = -1;

    c = num;

    while((c%2) == 0) {
      last = insertFactor(last, 2);
      c = c/2;
    }
    i = 3;
    while(i <= (sqrt(c)+1))
      if((c%i) == 0) {
        last = insertFactor(last, i);
        c = c/i;
      }
      else
        i = i+2;
    if(c > 1)
      last = insertFactor(last, c);
}

void calcDivisor(int value, int level, int n)
{
  if(level == n) {
    if(value > 0)
      divisors[totalDivisors++] = value;
    return;
  }
  int aux = 1;
  for(int i=0; i<=times[level]; i++) {
    calcDivisor(value * aux, level+1, n);
    aux *= factors[level];
  }
}

void divisorsCalculator(int n)
{
 primeFactorization(n);
 calcDivisor(1, 0, n);
}

int main() {
  divisorsCalculator(7);
  for(int i=0; i<totalDivisors; i++)
    cout << divisors[i] << " ";
  cout << endl;
  return 0;
}
```

FastestCountingCombinations

```
//     N!
// ----------    compute the values of C(N,K) when N and/or K is big but the
//   (N-K)!*K!   result is guaranteed to fit in 32-bit integer

#include<iostream>

using namespace std;

int GCD(int x, int y)
{
   while(y>0) {
      x = x%y;
      x^=y^=x^=y;  //swap
   }
   return x;
}

void divByGCD(int &x, int &y)
{
  int g = GCD(x, y);
  x /= g;
  y /= g;
}

int fastestCountingCombinations(int n, int k)
{
   int numerator = 1;
   int denominator = 1;
   int toMul, toDiv;

   if(k > n/2)
     k = n-k; //Smaller k

   for(int i=k; i; i--) {
     toMul = n-k+i;
     toDiv = i;
     divByGCD(toMul, toDiv);
     divByGCD(numerator, toDiv);
     divByGCD(toMul, denominator);
     numerator *= toMul;
     denominator *= toDiv;
   }
   return numerator/denominator;
}

int main() {
  cout << fastestCountingCombinations(10, 3);
  return 0;
}
```

FastestExponentiation

```
#include<iostream>

using namespace std;

int fastest_exponentiation(int base, int exponent) {
  int half;
  if(exponent == 0)
    return 1;
  else if(exponent % 2 == 0) {
    half = fastest_exponentiation(base, exponent/2);
    return half*half;
  }
  else {
    half = fastest_exponentiation(base, exponent-1);
    return base * half;
  }
}

int main() {
  cout << fastest_exponentiation(3, 2) << endl;
  cout << fastest_exponentiation(30, 5) << endl;
  cout << fastest_exponentiation(2, 5) << endl;
  return 0;
}
```

GCD

```
#include<iostream>

using namespace std;

int GCD(int x, int y)
{
   while(y>0) {
      x = x%y;
      x^=y^=x^=y;  //swap
   }
   return x;
}

int main() {
   cout << GCD(10, 20) << endl;
   cout << GCD(12, 57) << endl;
   cout << GCD(13, 20) << endl;
   cout << GCD(1, 20) << endl;
  return 0;
}
```

LCM

```
#include<iostream>

using namespace std;

int GCD(int x, int y)
{
   while(y>0) {
      x = x%y;
      x^=y^=x^=y;  //swap
   }
   return x;
}

int LCM(int x, int y)
{
  return (x * y)/GCD(x, y);
}

int main() {
  cout << LCM(8, 16) << endl;
  cout << LCM(3, 7) << endl;
  cout << LCM(100, 200) << endl;
  return 0;
}
```

NumberOfDivisors

```
#include<iostream>
#include<math.h>
#define MAX 100
using namespace std;

using namespace std;

int factors[MAX];
int times[MAX];
int total = 0;

long insertFactor(long lastFact, long actualFact)
{
  if(lastFact != actualFact) {
    factors[total] = actualFact;
    times[total]  = 1;
  }
  else
    times[total]++;
  total += 1;
  return actualFact;
```

```
}

void primeFactorization(long num)
{
    long i; //Counter
    long c; //Remaining product to factor
    long last = -1;

    c = num;

    while((c%2) == 0) {
      last = insertFactor(last, 2);
      c = c/2;
    }
    i = 3;
    while(i <= (sqrt(c)+1))
      if((c%i) == 0) {
        last = insertFactor(last, i);
        c = c/i;
      }
      else
        i = i+2;
    if(c > 1)
      last = insertFactor(last, c);
}


int totalDivisors(long n)
{
 int divisors = 1;

 primeFactorization(n);
 for(int i=0; i<total; i++)
   divisors *= times[i]+1;
 return divisors;
}

int main() {
  cout << totalDivisors(820);
  return 0;
}
```

PrimeFactorization

```
#include<iostream>
#include<math.h>
#define MAX 100
using namespace std;

int factors[MAX];
int total = 0;
```

```
void primeFactorization(long num)
{
    long i; //Counter
    long c; //Remaining product to factor
    c = num;

    while((c%2) == 0) {
      factors[total++] = 2;
      c = c/2;
    }
    i = 3;
    while(i <= (sqrt(c)+1))
      if((c%i) == 0) {
        factors[total++] = i;
        c = c/i;
      }
      else
        i = i+2;
    if(c > 1)
      factors[total++] = c;
}

int main()
{
  primeFactorization(70);
  for(int i=0; i<total; i++)
    cout << factors[i] << " ";
  cout << endl;
  return 0;
}
```

PrimeFactorizationCompressed

```
#include<iostream>
#include<math.h>
#define MAX 100
using namespace std;

int factors[MAX];
int times[MAX];
int total = 0;

long insertFactor(long lastFact, long actualFact)
{
  if(lastFact != actualFact) {
    factors[total] = actualFact;
    times[total]  = 1;
  }
  else
    times[total]++;
```

```
  total += 1;
  return actualFact;
}

void prime_factorization(long num)
{
    long i; //Counter
    long c; //Remaining product to factor
    long last = -1;

    c = num;

    while((c%2) == 0) {
      last = insertFactor(last, 2);
      c = c/2;
    }
    i = 3;
    while(i <= (sqrt(c)+1))
      if((c%i) == 0) {
        last = insertFactor(last, i);
        c = c/i;
      }
      else
        i = i+2;
    if(c > 1)
      last = insertFactor(last, c);
}

int main()
{
  primeFactorization(70);
  for(int i=0; i<total; i++)
    cout << factors[i] << " " << times[i] << " " << endl;
  cout << endl;
  return 0;
}
```

Ordenação

Metodos de ordenção

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#define NELEM 15

void insertion_sort(int s[], int n)
{
  for(int i=0; i<n; i++) {
```

```
    int j=i;
    while((j>0) && (s[j] < s[j-1])) {
      int tmp = s[j-1];
      s[j-1] = s[j];
      s[j] = tmp;
      j = j-1;
    }
  }
}

void selectionSort(int data[], int count)
{
  int min, temp;
  for(int i = 0; i < count - 1; i++) {
    min = i;
    for(int j = i+1; j < count; j++)
      if(data[j] < data[min])
        min = j;
    temp = data[i];
    data[i] = data[min];
    data[min] = temp;
  }
}

void q_sort(int numbers[], int left, int right)
{
  int pivot, l_hold, r_hold;
  l_hold = left;
  r_hold = right;
  pivot = numbers[left];
  while (left < right) {
    while ((numbers[right] >= pivot) && (left < right))
      right--;
    if(left != right) {
      numbers[left] = numbers[right];
      left++;
    }
    while((numbers[left] <= pivot) && (left < right))
      left++;
    if(left != right) {
      numbers[right] = numbers[left];
      right--;
    }
  }
  numbers[left] = pivot;
  pivot = left;
  left = l_hold;
  right = r_hold;
  if(left < pivot)
    q_sort(numbers, left, pivot-1);
  if(right > pivot)
    q_sort(numbers, pivot+1, right);
```

```
}



void quickSort(int numbers[], int array_size)
{
  q_sort(numbers, 0, array_size - 1);
}

bool  merge (int list1[], int size1, int list2[], int size2, int list3[])
{
  int i1, i2, i3;

  if (size1+size2 > NELEM) {
    return false;
  }
  i1 = 0; i2 = 0; i3 = 0;
  while (i1 < size1 && i2 < size2) {
    if (list1[i1] < list2[i2]) {
      list3[i3++] = list1[i1++];
    } else {
      list3[i3++] = list2[i2++];
    }
  }
  while (i1 < size1) {
    list3[i3++] = list1[i1++];
  }
  while (i2 < size2) {
    list3[i3++] = list2[i2++];
  }
}


void mergeSort (int array[], int size)
{
  int temp[NELEM];
  int mid, i;

  if (size < 2) {
    return;
  } else {
    mid = size / 2;
    mergeSort(array, mid);
    mergeSort(array + mid, size - mid);
    merge (array, mid, array + mid, size - mid, temp);
    for (i = 0; i < size; i++) {
      array[i] = temp[i];
    }
  }
}

int main()
```

```
{
  int s[NELEM+2] = {2,4,5,10,19,28,1,2,3,4,5,31,32,34,100};

  //insertion_sort(s,NELEM);
  //selectionSort(s,NELEM);
  //quickSort(s,NELEM-1);
  mergeSort(s, NELEM);
  for(int i=0; i<NELEM; i++) printf("%d ",s[i]);
    printf("\n");
  return 0;
}
```

Scramble Sort

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string>
#include <cstring>
#include <cctype>
#include <vector>
#include <sstream>
#include <algorithm>

using namespace std;

class word{
   public:
      string s;
      word(string c) {s = c;}
      bool operator < (const word &c) const {
         string xx = s;
         string yy = c.s;

         for(long long i = 0; i<xx.size(); ++i){
            if (isupper(xx[i])) xx[i] = tolower(xx[i]);
         }
         for(long long i = 0; i<yy.size(); ++i){
            if (isupper(yy[i])) yy[i] = tolower(yy[i]);
         }

         if (xx > yy) return false;
         if (xx < yy) return true;

      }

};

int main(){
   string t;
   char c;
```

```
while((c = cin.get()) != '.'){
  string t;
  cin.unget();
  getline(cin, t, '.');
  cin.get();

  for(int i=0; i<t.size(); ++i)
      if(t[i] == ',') t[i] = ' ';

  stringstream texto(stringstream::in | stringstream::out);
  texto << t;

  bool num[1000000] = {0};
  string t2;

  vector<word> palavra;
  vector<long long> numero;
  long long cont = 0;

  while (texto >> t2){
     stringstream t3(stringstream::in | stringstream::out);
     if(isdigit(t2[0]) || ((t2[0] == '-' || t2[0] == '+') && isdigit(t2[1]))){
         t3 << t2;
         long long x;
         t3 >> x;
         numero.push_back(x);
         num[cont] = true;
     }
     else {
        palavra.push_back(t2);
        num[cont] = false;
     }
     cont++;
  }

  sort(numero.begin(), numero.end());
  sort(palavra.begin(), palavra.end());

  long long n = 0, p = 0;
  for(int i=0; i<cont-1; ++i){
     if(num[i]){
        cout << numero[n] << ", ";
        n++;
     }
     else {
        cout << palavra[p].s << ", ";
        p++;
     }
  }

  if(num[cont-1]){
     cout << numero[n] << ".\n";
```

```
      }
      else {
         cout << palavra[p].s << ".\n";
      }

   }

   return 0;
}
```

Primos

isPrime

```
#include<iostream>

using namespace std;

bool isPrime(int number) {
  if(number == 1 || (number%2 == 0)) return false;
  if(number == 2) return true;

  for(int i=3; i*i<=number; i+=2) {
    if(number%i == 0)
      return 0;
  }
  return 1;
}

int main() {
  for(int i=0; i<100; i++) {
    if(isPrime(i))
      cout << i << " is Prime" << endl;
  }
  return 0;
}
```

PrimeTable

```
#include<iostream>
#include<math.h>
#define MAX 1000

using namespace std;

int table[MAX];
```

```
int offset;

void generatePrimeTable(int total) {
  bool isPrime;
  int sq, counter;
  offset = counter = 0;

  table[offset++] = 2;
  counter++;

  for(int i=3;;i+=2) {
    isPrime = true;
    sq = (int)sqrt(i) + 1;
    for(int j=0;j<offset && table[j]<=sq;j++) {
      if(!(i%table[j])) {
        isPrime = false;
                                break;
      }
    }
    if(isPrime){
      table[offset++] = i;
      counter++;
    }
    if(total==counter)
      break;
  }
}

int main() {
  generatePrimeTable(100);
  for(int i=0; i<offset; i++) {
    cout << table[i] << endl;
  }
  return 0;
}
```

Progração Dinamica

countingChange

```
#include<iostream>
#define MAXN 2000
#define MAXK 1000

using namespace std;

int matrix[MAXN][MAXK+1];
```

```
int cost[MAXN];
int profit[MAXN];

int knapsack(int n,int k) {
  int c;

  for(int i=0; i<n; i++)
    for (int j=0;j<=k;j++)
      matrix[i][j] = 0;
  for(int i=cost[0]; i<=k; i++)
    matrix[0][i] = profit[0];

  for(int i=1;i<n;i++)
    for(int j=0;j<=k;j++) {
      if(j-cost[i] >= 0)
                c = matrix[i-1][j-cost[i]]+profit[i];
      else
                c = 0;
      if(c < matrix[i-1][j])
                c = matrix[i-1][j];
       matrix[i][j] = c;
    }
  return matrix[n-1][k];
}

int main()
{
   return 0;
}
```

Knapsack

```
#include<iostream>
#define MAXN 2000
#define MAXK 1000

using namespace std;

int matrix[MAXN][MAXK+1];
int cost[MAXN];
int profit[MAXN];

int knapsack(int n,int k) {
  int c;

  for(int i=0; i<n; i++)
    for (int j=0;j<=k;j++)
      matrix[i][j] = 0;
  for(int i=cost[0]; i<=k; i++)
    matrix[0][i] = profit[0];
```

```
  for(int i=1;i<n;i++)
    for(int j=0;j<=k;j++) {
      if(j-cost[i] >= 0)
                c = matrix[i-1][j-cost[i]]+profit[i];
      else
                c = 0;
      if(c < matrix[i-1][j])
                c = matrix[i-1][j];
       matrix[i][j] = c;
    }
  return matrix[n-1][k];
}

int main()
{
   return 0;
}
```

String

StringMatching

```
#include<iostream>
#include<string.h>
using namespace std;

int findMatch(char *p, char*t)
{
   int plen, tlen;

   plen = strlen(p);
   tlen = strlen(t);

   for(int i=0; i<=(tlen-plen); i++) {
      int j=0;

      while((j<plen) && (t[i+j]==p[j]))
        j++;
      if(j == plen)
        return i;
   }
   return -1;
}

int main()
{
   int position = findMatch("jair", "Cazarin jai villanueva jair");
```

```
   cout << position << endl;
   return 0;
}
```

Stringstream

```
#include <iostream>
#include <cmath>
#include <vector>
#include <algorithm>
#include <cctype>
#include <string>
#include <cstring>
#include <map>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <sstream>
using namespace std;
int main(){
        int t;
        int teste = 1;
        cin >> t;
        int n;
        int d, i;
        for(int k=0; k < t; ++k){
                cin >> n;
                string robos[30];
                for(i=0; i<n; ++i) cin >> robos[i];
                cin.get();
                cin >> d;
                bool presente[30] = {0};
                cin.get();
                for(i=0; i<d; ++i){
                   string t;
                   getline(cin, t);
                   stringstream texto(stringstream::in | stringstream::out);
                        texto << t;
                        string t2;
                        while (texto >> t2){
                            for(int j=0; j<n; ++j)   if(t2 == robos[j]) presente[j] = true;
                        }
                }

                cout << "Test set " << teste++ << ":\n";
                for(i=0; i<n; ++i){
                        if(presente[i]) cout << robos[i] << " is present" << endl;
```

```
                else cout << robos[i] << " is absent" << endl;
            }
            cout << endl;
        }

        return 0;
}
```

Problemas feitos

110501 - Primary Arithmetic

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string>
#include <cstring>
#include <cctype>
#include <vector>
#include <sstream>
#include <algorithm>

#define ll long long

using namespace std;


ll calcula (ll x, ll y){
   ll vaium = 0;

   int t1;

   int a[10] = {0};
   int b[10] = {0};

   int i = 9;
   while(x>0){
      a[i] = x%10;
      i--;
      x/=10;
   }
   i = 9;
   while(y>0){
      b[i] = y%10;
      i--;
      y/=10;
   }
   int v1 = 0;
   for(int j = 9; j>=0; --j){
```

```
        t1 = a[j] + b[j] + v1;
        if(t1 >= 10) {
            vaium++;
            v1 = 1;
        }
        else v1 = 0;
    }

    return vaium;
}

int main(){
    ll x, y;
    cin >> x >> y;
    while(x || y){
        ll v1 = calcula(x, y);
        if(v1 == 0) printf("No carry operation.\n");
        else if( v1 > 1 ) printf("%d carry operations.\n", v1);
        else printf("%d carry operation.\n", v1);

        cin >> x >> y;
    }
    return 0;
}
```

110502 - Reverse and Add

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string>
#include <cstring>
#include <cctype>
#include <vector>
#include <sstream>
#include <algorithm>

#define ll long long

using namespace std;

vector<ll> a;
vector<ll> aux;

bool palindrome(){
    ll i = 0, j = a.size()-1, k;
    int tam = a.size();
```

```
   for(k = 0; k<tam/2; ++k){
      if(a[i] != a[j]) return false;
      i++;
      j--;
   }
   return true;
}


void converte(int n){
    vector<ll>::iterator it;
    it = a.begin();
    while(n>0){
      it = a.insert(it, n%10);
      aux.push_back(n%10);
      n/=10;

   }
}

void soma(){
   ll t1;
   vector<ll> c;
   vector<ll>::iterator it;
   it = c.begin();
   int v1 = 0;
   for(int j = a.size()-1; j>=0; --j){
      t1 = a[j] + aux[j] + v1;
      if(t1 >= 10) {
         v1 = 1;
      }
      else v1 = 0;
      it = c.insert(it, t1%10);
   }
   if(v1 == 1)
     it = c.insert(it, 1);

   a.clear();
   aux.clear();

   a = c;
   for(ll i=a.size()-1; i>=0; --i)
      aux.push_back(a[i]);
}
```

```
int main(){
   int n;
   cin >> n;
   while(n--){
      ll x;
      cin >> x;
      a.clear();
      aux.clear();
      ll num = 0;
      converte(x);
      while( !palindrome() ){
         soma();
         num++;
      }
      cout << num << " ";
      for(int i=0; i<a.size(); ++i)
        cout << a[i];
      cout << endl;
   }

   return 0;
}
```

110506 - Polynomial Coefficients

```
/******
Feito com a seguinte ajuda:

Here's a quick overview:
Let n(k) be the power of the kth term in the monomial we're asked to examine.
Given the polynomial (x1 + x2 + ... + xl)^n, we can break it up using the binomial theorem:
((x1 + ... + xk-1) + xk)^n
[sum from i = 1 to k] (n C i) * (x1 + ... + xk-1)^n-i * xk^i
The only term we care about is where i = n(xk), so we can through away the rest. We keep
breaking this down recursively until we end up with a product of Chooses,
which we can then simplify into a simple final form:
n! / (n(1)! * n(2)! * ... * n(k)!)

e

http://mathworld.wolfram.com/MultinomialCoefficient.html

(p+q+r)^2 = {p + (q+r)}^2;
******/

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>
#include <string>
#include <cstring>
#include <cctype>
#include <vector>
#include <sstream>
#include <algorithm>

using namespace std;

double fatorial(double n){
	double f = 1;
	if(n <= 1)
		return 1;
	for(int i=1; i<=n; i++)
		f *= i;
	return f;
}
int f[13] = {1,1,2,6,24,120,720,5040,40320,362880,3628800,39916800,479001600};

int main(){
	int c;
	double prod, r;
	int n, k;
	while(cin >> n >> k){
		prod = 1;
		for(int i=0; i<k; i++){
			cin >> c;
			prod *= f[c];
		}
		cout << (int)(f[n]/prod) << endl;
	}

	return 0;
}
```

110703 - Euclid Problem

```
#include<stdio.h>
void expand_gcd(long a,long b,long &d,long &x,long &y){
  if(b==0){
    d=a;x=1;y=0;
  }
  expand_gcd(b,a%b,d,y,x);
  y-=x*(a/b);
}
int main(){
  long a,b,d,x,y;
  while(scanf("%ld%ld",&a,&b)==2)  {
    expand_gcd(a,b,d,x,y);
    printf("%ld %ld %ld\n",x,y,d);
```

```
   }
return 0;
}
```

110901 – Bicoloring

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string>
#include <cstring>
#include <cctype>
#include <vector>
#include <sstream>
#include <algorithm>

#define MAX 201
#define COR1 1
#define COR2 2

using namespace std;

int grafo[MAX][MAX];
int cor[MAX];
bool colorido[MAX];

bool colore(int ini, int n) //Busca em profundidade
{
   for(int i = 0; i < n; i++)
   {
      if(grafo[ini][i] == 1 && !colorido[i])
      {
         colorido[ini] = true;
         if(cor[ini] == COR1)
         {
            if(cor[i] == 0)
               cor[i] = COR2;
            else if(cor[i] == COR1)
               return false;
         }
         else
         {
            if(cor[i] == 0)
               cor[i] = COR1;
            else if(cor[i] == COR2)
               return false;
         }
         colore(i, n);
      }
   }
   return true;
```

```
}

int main()
{
   int n, m, a, b;
   cin >> n;
   while(n)
   {
      for(int i = 0; i < n; i++)
      {
         for(int j = 0; j < n; j++)
            grafo[i][j] = 0;

         cor[i] = 0;
         colorido[i] = false;
      }

      cin >> m;
      for(int i = 0; i < m; i++)
      {
         cin >> a >> b;
         grafo[a][b] = 1;
         grafo[b][a] = 1;
      }

      cor[0] = COR1;
      if(colore(0, n))
         cout << "BICOLORABLE.\n";
      else
         cout << "NOT BICOLORABLE.\n";

      cin >> n;
   }
   return 0;
}
```

111101 - Is Bigger Smarter?

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string>
#include <cstring>
#include <cctype>
#include <vector>
#include <sstream>
#include <algorithm>
#include <vector>
#include <stack>

#define ll long long int
```

```
#define MAX 1001
#define INF 1000000000

using namespace std;

struct elefante{
   int p, qi, n;
};

bool compara(const elefante &a, const elefante &b)
{
   if(a.qi == b.qi)
      return a.p < b.p;
   return a.qi > b.qi;
}

int tam[MAX]; //tamanho da maior (todos = 1)
int pr[MAX];//quem vem antes (todos = -1)

int lis(elefante e[], int n)
{
   for (int i=0; i<n; i++)
      for (int j=i+1; j<n; j++)
         if (e[j].p > e[i].p && e[j].qi < e[i].qi) //se for maior
            if (tam[i] + 1 > tam[j])   //e aumentar a
            {
               tam[j] = tam[i] + 1; //sequencia
               pr[j] = i;
            }
//retornar a posição do maior valor do vetor tam
}

int main(){
   int p, qi, i=0, n=0;
   elefante e[MAX];
   while(cin >> p >> qi){
      e[i].p = p;
      e[i].qi = qi;
      e[i].n = i+1;
      i++;
      n++;
   }
   sort(e, e+n, compara);

   for(i = 0; i<n; ++i){
      tam[i] = 1;
      pr[i] = -1;
   }

   lis(e, n);
   int posM = 0;
```

```
   for(i = 0; i<n; ++i)
      if( tam[i] >= tam[posM] ) posM = i;

   p = posM;
   int cont = 1;
   stack<int> pilha;
   while(pr[p] != -1){
      pilha.push(e[p].n);
      p = pr[p];
      cont++;
   }
   pilha.push(e[p].n);
   cout << cont << endl;
   while(!pilha.empty()){
      cout << pilha.top() << endl;
      pilha.pop();
   }
   return 0;
}
```

111103 - Weights and Measures

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string>
#include <cstring>
#include <cctype>
#include <vector>
#include <sstream>
#include <algorithm>
#include <vector>
#include <stack>

#define ll long long int
#define MAX 6000
#define INF 1000000000

using namespace std;

struct tartaruga{
   int p, s, n, r;
};

bool compara(const tartaruga &a, const tartaruga &b)
{
   if (a.s != b.s)
      return a.s < b.s;
   return a.p > b.p;
}

int tam[MAX]; //tamanho da maior (todos = 1)
```

```
int pr[MAX];//quem vem antes (todos = -1)

int lis(tartaruga t[], int n)
{
   for (int i=0; i<n; i++)
      for (int j=i+1; j<n; j++)
         if (t[j].r + t[i].p <= t[j].s) //se for maior
            if (tam[i] + 1 > tam[j])   //e aumentar a
            {
               tam[j] = tam[i] + 1; //sequencia
               pr[j] = i;
               t[j].r+=t[i].p;
            }
//retornar a posição do maior valor do vetor tam
}

int main(){
   int p, s, i=0, n=0;
   tartaruga t[MAX];
   while(cin >> p >> s){
      t[i].p = p;
      t[i].s = s;
      t[i].n = i+1;
      t[i].r = p;
      i++;
      n++;
   }
   sort(t, t+n, compara);

   for(i = 0; i<n; ++i){
      tam[i] = 1;
      pr[i] = -1;
   }
   lis(t, n);
   int posM = 0;
   for(i = 0; i<n; ++i)
      if( tam[i] >= tam[posM] ) posM = i;

   p = posM;
   int cont = 1;
   while(pr[p] != -1){
      p = pr[p];
      cont++;
   }
   cout << cont << endl;

   return 0;

}
```

110901 – Bicoloring (Dudu)

```
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<iomanip>
#include<string>
#include<list>

using namespace std;
/* Bicoloring */
class grafo{
	public:
	list<int> adjacentes;
	void clean(){
		adjacentes.clear();
	}


};

void verificaCiclo (int v, grafo g[], int b[], bool &t, int veiode, int coratual){
	if (b[v] == 0){
		b[v] = coratual;
		if (coratual == 1) coratual = 2;
		else coratual = 1;
		list<int>::iterator it;
		for ( it=g[v].adjacentes.begin() ; it != g[v].adjacentes.end(); it++ ){
			if (*it != veiode && b[*it] == 0)
				verificaCiclo(*it, g, b, t, v, coratual);
			if (b[*it] != coratual) t = true;
			if (t) break;
		}
	}

}

int main ()
{
	int n;

	cin >> n;

	while (n){
		int a;
		grafo v[200];
		cin >> a;
		for (int i = 0; i < a; i++){
			int v1, v2;
			cin >> v1 >> v2;
			v[v1].adjacentes.push_back(v2);
			v[v2].adjacentes.push_back(v1);
```

```
                }
                int b[n];
                bool temciclo = false;
                for (int i = 0; i < n; i++){
                        b[i] = 0;
                }
    //for (int i = 0; i < n; i++){
                        verificaCiclo(0,v,b,temciclo,0,1);
                //      for (int j = 0; j < n; j++){
                //              b[j] = 0;
                //      }
                //}
                if (temciclo){
                        cout << "NOT BICOLORABLE.\n";
                }
                else
                        cout << "BICOLORABLE.\n";

                for (int i = 0; i < 200; i++){
                        v[i].adjacentes.clear();
                }
                cin >> n;
        }
  return 0;
}
```

110903 - The Tourist Guide

```
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<iomanip>
#include<string>
#include<map>
#include<list>
#include<queue>

/****  The Tourist Guide  ****/
/*
```

Mr. G. works as a tourist guide in Bangladesh. His current assignment is to show a group of tourists a distant city.
As in all countries, certain pairs of cities are connected by two-way roads. Each pair of neighboring cities has a bus service
that runs only between those two cities and uses the road that directly connects them. Each bus service has a particular limit
on the maximum number of passengers it can carry. Mr. G. has a map showing the cities and the roads connecting them, as well
as the service limit for each each bus service.

It is not always possible for him to take all the tourists to the destination city in a single trip.

For example, consider the following road map of seven cities, where the edges represent roads and the number written
on each edge indicates the passenger limit of the associated bus service.

*/

```
using namespace std;

struct dist{
	int dest;
	int valor;

};

class nodo{
	public:
	int custo;
	int indice;
	list<dist> adj;
	nodo (){
		custo = 100000000;
	}
	void clean(){
		custo = 100000000;
		adj.clear();
	}
	bool operator<(const nodo &e) const{
		return e.custo > custo;
	}
};

nodo grafo[101];
bool visitado[101];
void criaaresta(nodo grafo[], int orig, int dest, int p){
	dist d;
	d.dest = dest;
	d.valor = p;
	grafo[orig].adj.push_back(d);
}

int achacusto(nodo g[], int o, int d, bool v[]){
	priority_queue<nodo> q;
	q.push(g[o]);
	while (!q.empty()){
		nodo n = q.top();
		q.pop();

		if (n.indice == d){
			return n.custo;
		}

		v[n.indice] = true;
```

```
                list<dist>::iterator it;
                for (it = n.adj.begin(); it != n.adj.end(); it++){
                        nodo prox = g[(*it).dest];
                        if (!v[prox.indice]){
                                prox.custo = (n.custo < (*it).valor) ? n.custo : (*it).valor;
                                q.push(prox);
                        }
                }
        }
}

int main ()
{
        int nv, na;
        int scen = 0;
        cin >> nv >> na;
        while (nv || na){
                for (int i = 1; i<101; i++){
                        grafo[i].indice = i;
                        visitado[i] = false;
                        grafo[i].clean();
                }
                for (int i = 0 ; i<na; i++){
                        int o,d,p;
                        cin >> o;
                        cin >> d;
                        cin >> p;
                        criaaresta(grafo,o,d,p-1);
                        criaaresta(grafo,d,o,p-1);
                }
                int origem, destino, total;
                cin >> origem >> destino >> total;
                cout << "Scenario #" << ++scen << endl;
                cout << "Minimum Number of Trips = " ;
                int npass =  achacusto(grafo,origem,destino,visitado) ;
                int numt = total / npass;
                if (total % npass > 0)
                        numt++;
                cout << numt << endl;
                cin >> nv >> na;
                cout << endl;
        }
   return 0;
}
```

111001 – Freckels

```
#include<iostream>
#include<cmath>
#include<cstdlib>
#include<iomanip>
```

```
#include<string>
#include<map>
#include<list>
#include<queue>
#include<cstdio>

using namespace std;

struct aresta;

class nodo{
	public:
	int indice;
	double x;
	double y;
	double custo;

	list<aresta> adj;
	nodo (){
		custo = 0;
	}
	void clean(){
		adj.clear();
		custo = 0;
	}

	bool operator<(const nodo &e) const{
		return e.custo < custo;
	}
	void print (){
		cout << "(" << x << ", " << y << ") " << "=>  custo= " << custo << endl;

	}
};

struct aresta {
	nodo n;
	double v;
};

nodo grafo[101];
bool visitado[101];

double calculadist(nodo a, nodo b){
	return sqrt(pow(a.x-b.x,2.0)+ pow(a.y-b.y,2.0));


}

int main ()
{
	int ntest;
```

```
cin >> ntest;

for (int e = 0; e < ntest ; e++){
	int nvert;
	cin >> nvert;
	for (int i = 1 ; i<=nvert; i++){
		visitado[i] = false;
		grafo[i].clean();
		grafo[i].indice = i;
		cin >> grafo[i].x >> grafo[i].y;
	}

	/*for (int i = 1 ; i<=nvert; i++){
		for (int j = 1; j<= nvert; j++)
			if (i != j){
				aresta a;
				a.n = grafo[j];
				a.v = calculadist(grafo[i],grafo[j]);
				grafo[i].adj.push_back(a);
			}
	}*/

	priority_queue<nodo> q;

	q.push(grafo[1]);
	double total = 0;

	while (!q.empty()){
		nodo n = q.top();
		//n.print();
		q.pop();
		if (visitado[n.indice]) continue;

		total += n.custo;
		visitado[n.indice] = true;
		//cout << "Filhos : " << endl;
		list<aresta>::iterator it;
		for (int i = 1 ; i<=nvert; i++){
			if (i!=n.indice && !visitado[i]){
				nodo prox = grafo[i];
				prox.custo = calculadist(grafo[n.indice],prox);
				//prox.print();
				q.push(prox);

			}
		}
		//cout << endl;
	}

	printf("%.2lf\n", total);
	if (e != ntest-1) cout << endl;
```

```
                //cout << total;
                /*for (int i = 1; i<=nvert; i++){
                        cout << "Vertice " <<  i << endl;
                        list<aresta>::iterator it;
                        for (it = grafo[i].adj.begin(); it != grafo[i].adj.end(); it++){
                                cout << (*it).n.indice << " : " << (*it).v << "    " ;
                        }
                        cout << endl << endl;
                }
*/
        }
   return 0;
}
```

110601 - How Many Fibs

```
import java.math.*;
import java.util.*;
public class Main {
   static BigInteger fb[];
   public static void main(String[] args) {
      BigInteger a, b;
      Scanner sc = new Scanner(System.in);
      fb = new BigInteger[501];
      fb[0] = BigInteger.ONE;
      fb[1] = fb[0].add(BigInteger.ONE);
      for (int n = 2; n <= 500; n++) {
         fb[n] = fb[n - 1].add(fb[n - 2]); // fb(n) = fb(n-1) + fb(n-2)
      }
      while (true) {
         a = sc.nextBigInteger();
         b = sc.nextBigInteger();
         if (a.compareTo(BigInteger.ZERO) == 0 && b.compareTo(BigInteger.ZERO) == 0) {
            break;
         }
         int cont = 0;
         int i = 0;
         while (fb[i].compareTo(a) < 0) {
            ++i;
         }
         for (; i <= 500; ++i) {
            if (fb[i].compareTo(b) <= 0) {
               cont++;
            } else {
               break;
            }
         }
         System.out.println(cont);
      }
   }
}
```