Code Template for ACM-ICPC

PseudoCode, IIT Goa

April 8, 2023

Contents

1 DP

1.1 LCS

```
/*
Calculates the length of the longest common subsequence
    of two vectors.
Backtracks to find a single subsequence or all
    subsequences. Runs in
O(m*n) time except for finding all longest common
    subsequences, which
may be slow depending on how many there are.
*/

#include <iostream>
#include <vector>
#include <set>
#include <algorithm>

using namespace std;

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

typedef vector<int> VI;
typedef vector<VI> VVI;

void backtrack(VVI& dp, VT& res, VT& A, VT& B, int i,
    int j)
{
  if(!i || !j) return;
  if(A[i-1] == B[j-1]) { res.push_back(A[i-1]);
      backtrack(dp, res, A, B, i-1, j-1); }
  else
  {
    if(dp[i][j-1] >= dp[i-1][j]) backtrack(dp, res, A,
        B, i, j-1);
    else backtrack(dp, res, A, B, i-1, j);
  }
}

void backtrackall(VVI& dp, set<VT>& res, VT& A, VT& B,
    int i, int j)
{
  if(!i || !j) { res.insert(VI()); return; }
  if(A[i-1] == B[j-1])
  {
    set<VT> tempres;
    backtrackall(dp, tempres, A, B, i-1, j-1);
    for(set<VT>::iterator it=tempres.begin();
        it!=tempres.end(); it++)
    {
      VT temp = *it;
      temp.push_back(A[i-1]);
      res.insert(temp);
    }
  }
  else
  {
    if(dp[i][j-1] >= dp[i-1][j]) backtrackall(dp, res,
        A, B, i, j-1);
    if(dp[i][j-1] <= dp[i-1][j]) backtrackall(dp, res,
        A, B, i-1, j);
  }
}

VT LCS(VT& A, VT& B)
{
  VVI dp;
  int n = A.size(), m = B.size();
  dp.resize(n+1);
  for(int i=0; i<=n; i++) dp[i].resize(m+1, 0);

  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
    {
      if(A[i-1] == B[j-1]) dp[i][j] = dp[i-1][j-1]+1;
      else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
    }

  VT res;
  backtrack(dp, res, A, B, n, m);
  reverse(res.begin(), res.end());
  return res;
}

set<VT> LCSall(VT& A, VT& B)
{
  VVI dp;
  int n = A.size(), m = B.size();
  dp.resize(n+1);
  for(int i=0; i<=n; i++) dp[i].resize(m+1, 0);
  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++)
    {
      if(A[i-1] == B[j-1]) dp[i][j] = dp[i-1][j-1]+1;
      else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
    }
  set<VT> res;
  backtrackall(dp, res, A, B, n, m);
  return res;
}

int main()
{
  int a[] = { 0, 5, 5, 2, 1, 4, 2, 3 }, b[] = { 5, 2,
      4, 3, 2, 1, 2, 1, 3 };
  VI A = VI(a, a+8), B = VI(b, b+9);
  VI C = LCS(A, B);

  for(int i=0; i<C.size(); i++) cout << C[i] << " ";
  cout << endl << endl;

  set <VI> D = LCSall(A, B);
  for(set<VI>::iterator it = D.begin(); it != D.end();
      it++)
  {
    for(int i=0; i<(*it).size(); i++) cout << (*it)[i]
        << " ";
    cout << endl;
  }
}
```

1.2 LongestIncreasingSubsequence

```
// Given a list of numbers of length n, this routine
    extracts a
// longest increasing subsequence.
//
// Running time: O(n log n)
//
```

```
//   INPUT: a vector of integers
//   OUTPUT: a vector containing the longest increasing
    subsequence

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> VI;
typedef pair<int,int> PII;
typedef vector<PII> VPII;

#define STRICTLY_INCREASNG

VI LongestIncreasingSubsequence(VI v) {
  VPII best;
  VI dad(v.size(), -1);

  for (int i = 0; i < v.size(); i++) {
#ifdef STRICTLY_INCREASNG
    PII item = make_pair(v[i], 0);
    VPII::iterator it = lower_bound(best.begin(),
        best.end(), item);
    item.second = i;
#else
    PII item = make_pair(v[i], i);
    VPII::iterator it = upper_bound(best.begin(),
        best.end(), item);
#endif
    if (it == best.end()) {
      dad[i] = (best.size() == 0 ? -1 :
          best.back().second);
      best.push_back(item);
    } else {
      dad[i] = dad[it->second];
      *it = item;
    }
  }

  VI ret;
  for (int i = best.back().second; i >= 0; i = dad[i])
    ret.push_back(v[i]);
  reverse(ret.begin(), ret.end());
  return ret;
}
```

2 Geometry

2.1 Geometry

```
// C++ routines for computational geometry.

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

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

struct PT {
  double x, y;
  PT() {}
  PT(double x, double y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
  PT operator + (const PT &p) const { return PT(x+p.x,
      y+p.y); }
  PT operator - (const PT &p) const { return PT(x-p.x,
      y-p.y); }
  PT operator * (double c)  const { return PT(x*c, y*c
      ); }
  PT operator / (double c)  const { return PT(x/c, y/c
      ); }
};

double dot(PT p, PT q)  { return p.x*q.x+p.y*q.y; }
double dist2(PT p, PT q) { return dot(p-q,p-q); }
double cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ostream &operator<<(ostream &os, const PT &p) {
  os << "(" << p.x << "," << p.y << ")";
}

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

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane
    ax+by+cz=d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c,
                              double d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are
    parallel or collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

bool LinesCollinear(PT a, PT b, PT c, PT d) {
  return LinesParallel(a, b, c, d)
```

```
      && fabs(cross(a-b, a-c)) < EPS
      && fabs(cross(c-d, c-a)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return
          true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 &&
        dot(c-b, d-b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return
       false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return
       false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that
    unique
// intersection exists; for segment intersection, check
    if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b),
       c, c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex
    polygon (by William
// Randolph Franklin); returns 1 for strictly interior
    points, 0 for
// strictly exterior points, and 0 or 1 for the
    remaining points.
// Note that it is possible to convert this into an
    *exact* test using
// integer arithmetic by taking care of the division
    appropriately
// (making sure to deal with signs properly) and then
    by writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
      q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y)
          / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i],
        p[(i+1)%p.size()], q), q) < EPS)
      return true;
    return false;
}

// compute intersection of line through points a and b
    with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c,
    double r) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

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

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

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

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

// tests whether or not a given polygon (in CW or CCW
    order) is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}

int main() {

  // expected: (-5,2)
  cerr << RotateCCW90(PT(2,5)) << endl;

  // expected: (5,-2)
  cerr << RotateCW90(PT(2,5)) << endl;

  // expected: (-5,2)
  cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

  // expected: (5,2)
  cerr << ProjectPointLine(PT(-5,-2), PT(10,4),
      PT(3,7)) << endl;

  // expected: (5,2) (7.5,3) (2.5,1)
  cerr << ProjectPointSegment(PT(-5,-2), PT(10,4),
      PT(3,7)) << " "
       << ProjectPointSegment(PT(7.5,3), PT(10,4),
          PT(3,7)) << " "
       << ProjectPointSegment(PT(-5,-2), PT(2.5,1),
          PT(3,7)) << endl;

  // expected: 6.78903
  cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

  // expected: 1 0 1
  cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1),
      PT(4,5)) << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(2,0),
          PT(4,5)) << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(5,9),
          PT(7,13)) << endl;

  // expected: 0 0 1
  cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1),
      PT(4,5)) << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(2,0),
          PT(4,5)) << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(5,9),
          PT(7,13)) << endl;

  // expected: 1 1 1 0
  cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1),
      PT(-1,3)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3),
          PT(0,5)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1),
          PT(-2,1)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5),
          PT(1,7)) << endl;

  // expected: (1,2)
  cerr << ComputeLineIntersection(PT(0,0), PT(2,4),
      PT(3,1), PT(-1,3)) << endl;

  // expected: (1,1)
  cerr << ComputeCircleCenter(PT(-3,4), PT(6,1),
      PT(4,5)) << endl;

  vector<PT> v;
  v.push_back(PT(0,0));
  v.push_back(PT(5,0));
  v.push_back(PT(5,5));
  v.push_back(PT(0,5));

  // expected: 1 1 1 0 0
  cerr << PointInPolygon(v, PT(2,2)) << " "
       << PointInPolygon(v, PT(2,0)) << " "
       << PointInPolygon(v, PT(0,2)) << " "
       << PointInPolygon(v, PT(5,2)) << " "
       << PointInPolygon(v, PT(2,5)) << endl;

  // expected: 0 1 1 1 1
  cerr << PointOnPolygon(v, PT(2,2)) << " "
       << PointOnPolygon(v, PT(2,0)) << " "
       << PointOnPolygon(v, PT(0,2)) << " "
       << PointOnPolygon(v, PT(5,2)) << " "
       << PointOnPolygon(v, PT(2,5)) << endl;

  // expected: (1,6)
  //           (5,4) (4,5)
  //           blank line
  //           (4,5) (5,4)
  //           blank line
  //           (4,5) (5,4)
  vector<PT> u = CircleLineIntersection(PT(0,6),
      PT(2,6), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << "
      "; cerr << endl;
  u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1),
      5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << "
      "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(10,10), 5,
      5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << "
      "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << "
      "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5),
      10, sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << "
      "; cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5,
      sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << "
      "; cerr << endl;
```

```
  // area should be 5.0
  // centroid should be (1.1666666, 1.166666)
  PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
  vector<PT> p(pa, pa+4);
  PT c = ComputeCentroid(p);
  cerr << "Area: " << ComputeArea(p) << endl;
  cerr << "Centroid: " << c << endl;

  return 0;
}
```

3 Graphs

3.1 ArticulationOrBridge

```
// Array u acts as visited bool array, d stores DFN
    No., low
// stores lowest DFN no reachable, par stores parent
    node's DFN no.

int gl = 0;
const int N = 10010;
int u[N],d[N],low[N],par[N];
vi G[N];
void dfs1(int node,int dep){ //find dfs_num and dfs_low
  u[node]=1;
  d[node]=dep;low[node]=dep;
  for(int i = 0; i < G[node].size(); i++){
    int it = G[node][i];
    if(!u[it]){
      par[it]=node;
      dfs1(it,dep+1);
      low[node]=min(low[node],low[it]);
      /*if(low[it] > d[node] ){
          node-it is cut edge/bridge
      }*/
      /*
      if(low[it] >= d[node] && (par[node]!=-1 ||
          sz(G[node]) > 2)){
          node is cut vertex/articulation point
      }
      */
    }else if(par[node]!=it)
        low[node]=min(low[node],low[it]);
    else par[node]=-1;
  }
}
int main(){
  return 0;
}
```

3.2 BellmanFord

```
// This function runs the Bellman-Ford algorithm for
    single source
// shortest paths with negative edge weights. The
    function returns
// false if a negative weight cycle is detected.
    Otherwise, the
// function returns true and dist[i] is the length of
    the shortest
// path from start to i.
//
// Running time: O(|V|^3)
//
//   INPUT:  start, w[i][j] = cost of edge from i to j
//   OUTPUT: dist[i] = min weight path from start to i
//           prev[i] = previous node on the best path
    from the
//                     start node

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

using namespace std;

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

typedef vector<int> VI;
typedef vector<VI> VVI;

bool BellmanFord (const VVT &w, VT &dist, VI &prev, int
    start){
  int n = w.size();
  prev = VI(n, -1);
  dist = VT(n, 1000000000);
  dist[start] = 0;

  for (int k = 0; k < n; k++){
    for (int i = 0; i < n; i++){
      for (int j = 0; j < n; j++){
        if (dist[j] > dist[i] + w[i][j]){
          if (k == n-1) return false;
          dist[j] = dist[i] + w[i][j];
          prev[j] = i;
        }
      }
    }
  }

  return true;
}
```

3.3 Cycles

```
// 4.1 Cycle Detection in Undirected Graph
// using DFS

bool _detectCycle(int node, int parent, vector<int>
    &vis, vector<vector<int>> &adj) {
    vis[node] = 1;
    for (auto child : adj[node]) {
        if (!vis[child])
            if (_detectCycle(child, node, vis, adj))
                return true;
        if (vis[child] && parent != child) // child
             already visited and parent is not child
            return true;
    }
```

```
    return false;
}

bool detectCycle(vector<vector<int>> &adj, int n) {
    vector<int> vis(n + 1, 0);
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) {
            if (_detectCycle(i, -1, vis, adj))
                return true;
        }
    }
    return false;
}

//! 4.2 Length of shortest Cycle in Undirected Graph
    using BFS
class Solution {
public:
    int ans = 1e9;
    int findShortestCycle(int n, vector<vector<int>>
        &edges) {
        vector<int> graph[n];
        for (auto &e : edges) {
            graph[e[0]].push_back(e[1]);
            graph[e[1]].push_back(e[0]);
        }
        for (int i = 0; i < n; i++) { // for all
            components
            vector<int> dist(n, 1e9);
            queue<pair<int, int>> q; // {node,par}
            q.push({i, -1});
            dist[i] = 0;
            while (!q.empty()) {
                auto [u, par] = q.front();
                q.pop();
                for (auto v : graph[u]) {
                    if (dist[v] == 1e9) { // not visited
                        dist[v] = dist[u] + 1;
                        q.push({v, u});
                    } else if (par != v) {
                                            // visited and
                        not parent
                        ans = min(ans, dist[u] + dist[v]
                            + 1); // left sum + right sum
                            + 1 (edge between u and v)
                    }
                }
            }
        }
        return ans == 1e9 ? -1 : ans;
    }
};


// 6. Cycle Detection in Directed Graph using DFS
// Idea - if a node is already visited and is in the
    current path then there is a cycle
//
    https://practice.geeksforgeeks.org/problems/detect-cycle-in-a-directed-graph/1

class Solution {
public:
    bool dfs(int node, vector<vector<int>> &adj,
        vector<int> &vis, vector<int> &pathVis) {
        vis[node] = 1;
        pathVis[node] = 1;
        for (auto child : adj[node]) {
            if (!vis[child]) {
                if (dfs(child, adj, vis, pathVis))
                    return true;
            }
            if (vis[child] && pathVis[child])
                return true;
        }
        // backtracking step
        pathVis[node] = 0;
        return false;
    }

    bool CycleDetectionDirected(int n,
        vector<vector<int>> &adj) {
        vector<int> vis(n + 1);
        vector<int> pathVis(n + 1);
        for (int i = 1; i <= n; i++) {
            if (!vis[i])
                if (dfs(i, adj, vis, pathVis))
                    return true;
        }
        return false;
    }
};
```

3.4 Diameter

```
// 9.2 Diameter of a tree
// Finding the longest path in a tree using 2 dfs

int _dia(int node, int par, vector<vector<int>> &adj,
    vector<int> &dist) {
    int farthestNode = node;
    for (auto child : adj[node]) {
        if (child != par) {
            dist[child] = dist[node] + 1;
            int farthestChild = _dia(child, node, adj,
                dist);
            if (dist[farthestChild] > dist[farthestNode])
                farthestNode = farthestChild;
        }
    }
    return farthestNode;
}

int diameter(vector<vector<int>> &adj) {
    int n = adj.size();
    vector<int> dist(n + 1, 0);
    int farthestNode = _dia(1, -1, adj, dist);
    dist.assign(n + 1, 0);
    return _dia(farthestNode, -1, adj, dist);
}
```

3.5 DSU

```
class DSU {
private:
    vector<int> parent, rank;

public:
    DSU(int size) {
        parent.resize(size);
        rank.resize(size, 0);
```

```
        for (int i = 0; i < size; i++) {
            parent[i] = i;
        }
    }
    int find(int x) {
        if (parent[x] != x)
            parent[x] = find(parent[x]);
        return parent[x];
    }
    void union_set(int x, int y) {
        int xset = find(x), yset = find(y);
        if (xset == yset) {
            return;
        } else if (rank[xset] < rank[yset]) {
            parent[xset] = yset;
        } else if (rank[xset] > rank[yset]) {
            parent[yset] = xset;
        } else {
            parent[yset] = xset;
            rank[xset]++;
        }
    }
};
```

3.6 EulerianPath

```
struct Edge;

struct Edge {
  int next_vertex;
  iter reverse_edge;

  Edge(int next_vertex) :next_vertex(next_vertex)
    { }
};

const int max_vertices = 10;
// int num_vertices = 6;
list<Edge> adj[max_vertices]; // adjacency list

vector<int> path;

void find_path(int v) {
  while(adj[v].size() > 0) {
    int vn = adj[v].front().next_vertex;
    adj[vn].erase(adj[v].front().reverse_edge);
    adj[v].pop_front();
    find_path(vn);
  }
  path.push_back(v);
}

void add_edge(int a, int b) {
  adj[a].push_front(Edge(b));
  iter ita = adj[a].begin();
  adj[b].push_front(Edge(a));
  iter itb = adj[b].begin();
  ita->reverse_edge = itb;
  itb->reverse_edge = ita;
}

int main() {
  int total=0, start_vertex = 0;
  rep(i, max_vertices)
    if(adj[i].size()&1)                    // if the
        size is odd then increment 'total'
      total++, start_vertex=i;       // put the starting
          vertex as an odd degree vertex
  if(total==0||total==2) {                 //
      necessary and sufficient condition to check the
      existence of an EC
    find_path(start_vertex);
    rep(i, path.size()) cout << path[i] << " ";
  }
  else
    cout << "No Eulerian Circuit\n";
  return 0;
}
```

3.7 GraphColoring

```
// Graph Colouring (Main topic)
// 5.1 Bipartite Graphs - should not have odd length
    cycle, can be coloured using 2 adjacent colors

class Solution {
public:
    bool dfs(int node, vector<int> &vis,
        vector<vector<int>> &graph, int color) {
        vis[node] = color;
        for (auto child : graph[node]) {
            if (vis[child] == -1) { // unvisited call dfs
                if (!dfs(child, vis, graph, 1 - color))
                    return false;
            } else if (vis[child] == color) // already
                visited and same color
                return false;
        }
        return true;
    }
    bool isBipartite(vector<vector<int>> &graph) {
        int n = graph.size();
        int UNVISITED = -1, RED = 0, BLUE = 1;
        vector<int> vis(n, UNVISITED);
        for (int i = 0; i < n; i++) {
            if (vis[i] == UNVISITED) {
                if (!dfs(i, vis, graph, RED))
                    return false;
            }
        }
        return true;
    }
};

// 5.2 Bipartite coloring using BFS
class Solution {
public:
    bool isBipartite(vector<vector<int>> &graph) {
        int n = graph.size();
        int UNVISITED = -1, RED = 0, BLUE = 1;
        vector<int> color(n, UNVISITED);
        for (int i = 0; i < n; i++) { // running for
            all components
            queue<int> q;
            q.push(i);
            color[i] = RED;
            while (!q.empty()) {
                int node = q.front();
```

```
                q.pop();
                for (auto child : graph[node]) {
                    if (color[child] == UNVISITED) {
                        color[child] = 1 - color[node];
                        q.push(child);
                    } else if (color[child] ==
                         color[node])
                        return false;
                }
            }
        }
        return true;
    }
};
```

3.8 LCA

```
const int max_nodes, log_max_nodes;
int num_nodes, log_num_nodes, root;

vector<int> children[max_nodes]; // children[i]
    contains the children of node i
int A[max_nodes][log_max_nodes+1]; // A[i][j] is the
    2^j-th ancestor of node i, or -1 if that ancestor
    does not exist
int L[max_nodes];   // L[i] is the distance between
    node i and the root

// floor of the binary logarithm of n
int lb(unsigned int n) {
  if(n==0)
  return -1;
  int p = 0;
  if (n >= 1<<16) { n >>= 16; p += 16; }
  if (n >= 1<< 8) { n >>= 8; p += 8; }
  if (n >= 1<< 4) { n >>= 4; p += 4; }
  if (n >= 1<< 2) { n >>= 2; p += 2; }
  if (n >= 1<< 1) {         p += 1; }
  return p;
}

void DFS(int i, int l) {
  L[i] = l;
  for(int j = 0; j < children[i].size(); j++)
  DFS(children[i][j], l+1);
}

int LCA(int p, int q) {
  // ensure node p is at least as deep as node q
  if(L[p] < L[q])
  swap(p, q);

  // "binary search" for the ancestor of node p
     situated on the same level as q
  for(int i = log_num_nodes; i >= 0; i--)
  if(L[p] - (1<<i) >= L[q])
    p = A[p][i];

  if(p == q)
  return p;

  // "binary search" for the LCA
  for(int i = log_num_nodes; i >= 0; i--)
  if(A[p][i] != -1 && A[p][i] != A[q][i]) {
    p = A[p][i];
    q = A[q][i];
  }

  return A[p][0];
}

int main(int argc,char* argv[]) {
  // read num_nodes, the total number of nodes
  log_num_nodes=lb(num_nodes);

  for(int i = 0; i < num_nodes; i++) {
    int p;
    // read p, the parent of node i or -1 if node i is
        the root

    A[i][0] = p;
    if(p != -1)
      children[p].push_back(i);
    else
      root = i;
  }

  // precompute A using dynamic programming
  for(int j = 1; j <= log_num_nodes; j++)
    for(int i = 0; i < num_nodes; i++)
      if(A[i][j-1] != -1)
        A[i][j] = A[A[i][j-1]][j-1];
      else
        A[i][j] = -1;

  // precompute L
  DFS(root, 0);

  return 0;
}
```

3.9 MaxBipartiteMatching

```
// This code performs maximum bipartite matching.
//
// Running time: O(|E| |V|) -- often much faster in
    practice
//
//   INPUT: w[i][j] = edge between row node i and
    column node j
//   OUTPUT: mr[i] = assignment for row node i, -1 if
    unassigned
//           mc[j] = assignment for column node j, -1 if
    unassigned
//           function returns number of matches made

#include <vector>

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI
    &seen) {
  for (int j = 0; j < w[i].size(); j++) {
    if (w[i][j] && !seen[j]) {
      seen[j] = true;
      if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc,
          seen)) {
```

```
        mr[i] = j;
        mc[j] = i;
        return true;
      }
    }
  }
  return false;
}

int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
  mr = VI(w.size(), -1);
  mc = VI(w[0].size(), -1);

  int ct = 0;
  for (int i = 0; i < w.size(); i++) {
    VI seen(w[0].size());
    if (FindMatch(i, w, mr, mc, seen)) ct++;
  }
  return ct;
}
```

3.10 MinCostMaxFlow

```
// Implementation of min cost max flow algorithm using
    adjacency
// matrix (Edmonds and Karp 1972). This implementation
    keeps track of
// forward and reverse edges separately (so you can set
    cap[i][j] !=
// cap[j][i]). For a regular max flow, set all edge
    costs to 0.
//
// Running time, O(|V|^2) cost per augmentation
//     max flow:           O(|V|^3) augmentations
//     min cost max flow: O(|V|^4 * MAX_EDGE_COST)
    augmentations
//
// INPUT:
//     - graph, constructed using AddEdge()
//     - source
//     - sink
//
// OUTPUT:
//     - (maximum flow value, minimum cost value)
//     - To obtain the actual flow, look at positive
    values only.

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

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef long long L;
typedef vector<L> VL;
typedef vector<VL> VVL;
typedef pair<int, int> PII;
typedef vector<PII> VPII;

const L INF = numeric_limits<L>::max() / 4;

struct MinCostMaxFlow {
  int N;
  VVL cap, flow, cost;
  VI found;
  VL dist, pi, width;
  VPII dad;

  MinCostMaxFlow(int N) :
    N(N), cap(N, VL(N)), flow(N, VL(N)), cost(N, VL(N)),
    found(N), dist(N), pi(N), width(N), dad(N) {}

  void AddEdge(int from, int to, L cap, L cost) {
    this->cap[from][to] = cap;
    this->cost[from][to] = cost;
  }

  void Relax(int s, int k, L cap, L cost, int dir) {
    L val = dist[s] + pi[s] - pi[k] + cost;
    if (cap && val < dist[k]) {
      dist[k] = val;
      dad[k] = make_pair(s, dir);
      width[k] = min(cap, width[s]);
    }
  }

  L Dijkstra(int s, int t) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    fill(width.begin(), width.end(), 0);
    dist[s] = 0;
    width[s] = INF;

    while (s != -1) {
      int best = -1;
      found[s] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        Relax(s, k, cap[s][k] - flow[s][k], cost[s][k],
            1);
        Relax(s, k, flow[k][s], -cost[k][s], -1);
        if (best == -1 || dist[k] < dist[best]) best =
            k;
      }
      s = best;
    }

    for (int k = 0; k < N; k++)
      pi[k] = min(pi[k] + dist[k], INF);
    return width[t];
  }

  pair<L, L> GetMaxFlow(int s, int t) {
    L totflow = 0, totcost = 0;
    while (L amt = Dijkstra(s, t)) {
      totflow += amt;
      for (int x = t; x != s; x = dad[x].first) {
        if (dad[x].second == 1) {
          flow[dad[x].first][x] += amt;
          totcost += amt * cost[dad[x].first][x];
        } else {
          flow[x][dad[x].first] -= amt;
          totcost -= amt * cost[x][dad[x].first];
        }
      }
    }
    return make_pair(totflow, totcost);
  }
};
```

```
// BEGIN CUT
// The following code solves UVA problem #10594: Data
    Flow

int main() {
  int N, M;

  while (scanf("%d%d", &N, &M) == 2) {
    VVL v(M, VL(3));
    for (int i = 0; i < M; i++)
      scanf("%Ld%Ld%Ld", &v[i][0], &v[i][1], &v[i][2]);
    L D, K;
    scanf("%Ld%Ld", &D, &K);

    MinCostMaxFlow mcmf(N+1);
    for (int i = 0; i < M; i++) {
      mcmf.AddEdge(int(v[i][0]), int(v[i][1]), K,
          v[i][2]);
      mcmf.AddEdge(int(v[i][1]), int(v[i][0]), K,
          v[i][2]);
    }
    mcmf.AddEdge(0, 1, D, 0);

    pair<L, L> res = mcmf.GetMaxFlow(0, N);

    if (res.first == D) {
      printf("%Ld\n", res.second);
    } else {
      printf("Impossible.\n");
    }
  }

  return 0;
}

// END CUT
```

3.11 MinCut

```
// Adjacency matrix implementation of Stoer-Wagner min
    cut algorithm.
//
// Running time:
//     O(|V|^3)
//
// INPUT:
//     - graph, constructed using AddEdge()
//
// OUTPUT:
//     - (min cut value, nodes in half of min cut)



typedef vector<vi> vvi;

const int INF = 1000000000;

pair<int, vi> GetMinCut(vvi &weights) {
  int N = weights.size();
  vi used(N), cut, best_cut;
  int best_weight = -1;

  for (int phase = N-1; phase >= 0; phase--) {
    vi w = weights[0];
    vi added = used;
    int prev, last = 0;
    for (int i = 0; i < phase; i++) {
      prev = last;
      last = -1;
      for (int j = 1; j < N; j++)
        if (!added[j] && (last == -1 || w[j] >
            w[last])) last = j;
      if (i == phase-1) {
        for (int j = 0; j < N; j++) weights[prev][j] +=
            weights[last][j];
        for (int j = 0; j < N; j++) weights[j][prev] =
            weights[prev][j];
        used[last] = true;
        cut.push_back(last);
        if (best_weight == -1 || w[last] < best_weight)
            {
          best_cut = cut;
          best_weight = w[last];
        }
      } else {
        for (int j = 0; j < N; j++)
          w[j] += weights[last][j];
        added[last] = true;
      }
    }
  }
  return make_pair(best_weight, best_cut);
}

// BEGIN CUT
// The following code solves UVA problem #10989: Bomb,
    Divide and Conquer
int main() {
  int N;
  cin >> N;
  for (int i = 0; i < N; i++) {
    int n, m;
    cin >> n >> m;
    vvi weights(n, vi(n));
    for (int j = 0; j < m; j++) {
      int a, b, c;
      cin >> a >> b >> c;
      weights[a-1][b-1] = weights[b-1][a-1] = c;
    }
    pair<int, vi> res = GetMinCut(weights);
    cout << "Case #" << i+1 << ": " << res.first <<
        endl;
  }
}
// END CUT
```

3.12 MST

```
// Kruskal's Algorithm

int kruksal(vector<vector<int>> graph) {
    int n = graph.size();
    DSU dsu(n);
    sort(graph.begin(), graph.end(), [](vector<int> &a,
        vector<int> &b) {
        return a[2] < b[2]; // sort edges in increasing
            order of weight
    });
    int ans = 0;
    for (auto edge : graph) {
```

```
        int u = edge[0], v = edge[1], wt = edge[2];
        if (dsu.find(u) != dsu.find(v)) { // u, v in
            different components
            dsu.union_set(u, v);
            ans += wt;
        }
    }
    return ans; // Return the weight of the MST
}

// Prims
int prim(vector<vector<int>> graph) {
    int n = graph.size();
    vector<int> dist(n, 1e9);
    vector<bool> vis(n, false);
    dist[0] = 0;
    int src = 0;
    int ans = 0;
    priority_queue<pair<int, int>, vector<pair<int,
        int>>, greater<pair<int, int>>> pq;
    pq.push({0, src}); // {dist, node}
    while (!pq.empty()) {
        auto [d, node] = pq.top();
        pq.pop();
        if (vis[node])
            continue;
        vis[node] = true;
        ans += d;
        for (int i = 0; i < n; i++) {
            if (graph[node][i] != -1 && !vis[i]) {
                if (graph[node][i] < dist[i]) {
                    dist[i] = graph[node][i];
                    pq.push({dist[i], i});
                }
            }
        }
    }
    return ans; // Return the weight of the MST
}
```

3.13 SCC

```
#define MAXE 1000000
#define MAXV 100000
struct edge{int e, nxt;};
int V, E;
edge e[MAXE], er[MAXE];
int sp[MAXV], spr[MAXV];
int group_cnt, group_num[MAXV];
bool v[MAXV];
int stk[MAXV];  // Stack, stk[0] stores size
void fill_forward(int x) {
  int i;
  v[x]=true;
  for(i=sp[x];i;i=e[i].nxt) if(!v[e[i].e])
      fill_forward(e[i].e);
  stk[++stk[0]]=x;
}

void fill_backward(int x) {
  int i;
  v[x]=false;
  group_num[x]=group_cnt;
  for(i=spr[x];i;i=er[i].nxt) if(v[er[i].e])
      fill_backward(er[i].e);
}

void add_edge(int v1, int v2) { //add edge v1->v2
  e [++E].e=v2; e [E].nxt=sp [v1]; sp [v1]=E;
  er[ E].e=v1; er[E].nxt=spr[v2]; spr[v2]=E;
}

void SCC() {
  int i;
  stk[0]=0;
  memset(v, false, sizeof(v));
  for(i=1;i<=V;i++) if(!v[i]) fill_forward(i);
  group_cnt=0;
  for(i=stk[0];i>=1;i--) if(v[stk[i]]){group_cnt++;
      fill_backward(stk[i]);}
}

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

3.14 ShortestPaths

```
// BFS in undirected Graph (Edge weight = 1)

int shortestDist(int source, int dest,
     vector<vector<int>> adj) {
    int n = adj.size();
    vector<int> dist(n + 1, 1e9);
    queue<int> q;

    q.push(source);
    dist[source] = 0;

    while (!q.empty()) {
        int curr = q.front();
        q.pop();

        for (auto child : adj[curr]) {
            if (dist[child] == 1e9) {
                dist[child] = dist[curr] + 1;
                q.push(child);
            }
        }
    }
    return dist[dest];
}

// 1. Dijsktra - Shortest Path in Undirected Weighted
    Graphs
//
int dijkstra(int source, int dest,
     vector<vector<pair<int, int>>> adj) {
    int n = adj.size();
    vector<int> dist(n + 1, 1e9);
    priority_queue<pair<int, int>, vector<pair<int,
        int>>, greater<pair<int, int>>> pq;
    pq.push({0, source}); // {dist, node}
    dist[source] = 0;
    while (!pq.empty()) {
        auto [node, wt] = pq.top();
        pq.pop();
        for (auto child : adj[node]) {
            auto [childNode, childWt] = child;
```

```
            if (dist[childNode] > dist[node] + childWt)
                { // Relaxation
                dist[childNode] = dist[node] + childWt;
                pq.push({childNode, dist[childNode]});
            }
        }
    }
    return dist[dest];
    // vector<int> path;
    // int curr = dest;
    // while (curr != -1) {
    //     path.push_back(curr);
    //     curr = parent[curr];
    // }
    // reverse(path.begin(), path.end());
    // return path;
} // O(nlogn)

// 3
vector<vector<int>> floydWarshall(vector<vector<int>>
    adj) {
    int n = adj.size();
    vector<vector<int>> dist(n, vector<int>(n, 1e9));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j)
                dist[i][j] = 0;
            else if (adj[i][j] != -1)
                dist[i][j] = adj[i][j];
        }
    }
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    return dist;
}
```

3.15 TopoSort

```
stack<int> st;
void _TopoSort(int node, vector<int> &vis,
    vector<vector<int>> adj) {
    vis[node] = 1;
    for (auto child : adj[node]) {
        if (!vis[child]) {
            vis[child] = 1;
            _TopoSort(child, vis, adj);
        }
    }
    // Node has been processed
    st.push(node); // ? Simple Change
    return;
}

vector<int> TopoSort(int n, vector<vector<int>> adj) {
    vector<int> vis(n + 1, 0);
    for (int i = 1; i <= n; i++) {
        if (!vis[i]) {
            _TopoSort(i, vis, adj);
        }
    }
    vector<int> res;
    while (!st.empty()) {
        res.push_back(st.top());
        st.pop();
    }
    return res;
}

// 7.2 Kahn Algo Topological Sort (BFS) Iterative
vector<int> Kahn(int n, vector<vector<int>> adj) {
    vector<int> in(n + 1, 0);
    for (int i = 1; i <= n; i++) {
        for (auto child : adj[i]) { // go to all
            children of node i
            in[child]++;
        }
    }
    queue<int> q;
    // Initialize queue with all vertices with indegree
        0
    for (int i = 1; i <= n; i++) {
        if (in[i] == 0)
            q.push(i);
    }
    vector<int> res;
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        res.push_back(node);
        for (auto child : adj[node]) {
            in[child]--;
            if (in[child] == 0) // Push to queue if
                indegree is 0
                q.push(child);
        }
    }
    return res;
}
```

4 NumberTheory

4.1 Euclid

```
// This is a collection of useful code for solving
    problems that
// involve modular linear equations. Note that all of
    the
// algorithms described here work on nonnegative
    integers.


// return a % b (positive value)
int mod(int a, int b) {
  return ((a%b) + b) % b;
}

// computes gcd(a,b)
int gcd(int a, int b) {
  while (b) { int t = a%b; a = b; b = t; }
  return a;
}

// computes lcm(a,b)
int lcm(int a, int b) {
```

```
  return a / gcd(a, b)*b;
}

// (a^b) mod m via successive squaring
int powermod(int a, int b, int m) {
  int ret = 1;
  while (b)  {
    if (b & 1) ret = mod(ret*a, m);
    a = mod(a*a, m);
    b >>= 1;
  }
  return ret;
}

// returns g = gcd(a, b); finds x, y such that g = ax +
    by
int extended_euclid(int a, int b, int &x, int &y) {
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
    int q = a / b;
    int t = b; b = a%b; a = t;
    t = xx; xx = x - q*xx; x = t;
    t = yy; yy = y - q*yy; y = t;
  }
  return a;
}

// finds all solutions to ax = b (mod n)
vi modular_linear_equation_solver(int a, int b, int n) {
  int x, y;
  vi ret;
  int g = extended_euclid(a, n, x, y);
  if (!(b%g)) {
    x = mod(x*(b / g), n);
    for (int i = 0; i < g; i++)
      ret.push_back(mod(x + i*(n / g), n));
  }
  return ret;
}

// computes b such that ab = 1 (mod n), returns -1 on
    failure
int mod_inverse(int a, int n) {
  int x, y;
  int g = extended_euclid(a, n, x, y);
  if (g > 1) return -1;
  return mod(x, n);
}

// Chinese remainder theorem (special case): find z
    such that
// z % m1 = r1, z % m2 = r2. Here, z is unique modulo M
    = lcm(m1, m2).
// Return (z, M). On failure, M = -1.
pii chinese_remainder_theorem(int m1, int r1, int m2,
    int r2) {
  int s, t;
  int g = extended_euclid(m1, m2, s, t);
  if (r1%g != r2%g) return make_pair(0, -1);
  return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2) / g,
      m1*m2 / g);
}

// Chinese remainder theorem: find z such that
// z % m[i] = r[i] for all i. Note that the solution is
// unique modulo M = lcm_i (m[i]). Return (z, M). On
// failure, M = -1. Note that we do not require the
    a[i]'s
// to be relatively prime.
pii chinese_remainder_theorem(const vi &m, const vi &r)
    {
  pii ret = make_pair(r[0], m[0]);
  for (int i = 1; i < m.size(); i++) {
    ret = chinese_remainder_theorem(ret.second,
        ret.first, m[i], r[i]);
    if (ret.second == -1) break;
  }
  return ret;
}

// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x,
    int &y) {
  if (!a && !b) {
    if (c) return false;
    x = 0; y = 0;
    return true;
  }
  if (!a) {
    if (c % b) return false;
    x = 0; y = c / b;
    return true;
  }
  if (!b) {
    if (c % a) return false;
    x = c / a; y = 0;
    return true;
  }
  int g = gcd(a, b);
  if (c % g) return false;
  x = c / g * mod_inverse(a / g, b / g);
  y = (c - a*x) / b;
  return true;
}

int main() {
  // expected: 2
  cout << gcd(14, 30) << endl;

  // expected: 2 -2 1
  int x, y;
  int g = extended_euclid(14, 30, x, y);
  cout << g << " " << x << " " << y << endl;

  // expected: 95 45
  vi sols = modular_linear_equation_solver(14, 30, 100);
  for (int i = 0; i < sols.size(); i++) cout << sols[i]
      << " ";
  cout << endl;

  // expected: 8
  cout << mod_inverse(8, 9) << endl;

  // expected: 23 105
  //           11 12
  int v1[3]={3,5,7}, v2[3]={2,3,2};
  pii ret = chinese_remainder_theorem(vi(v1, v1+3),
      vi(v2, v2+3));
  cout << ret.first << " " << ret.second << endl;
  int v3[2]={4,6}, v4[2]={3,5};
  ret = chinese_remainder_theorem(vi(v3, v3+2), vi(v4,
      v4+2));
```

```
  cout << ret.first << " " << ret.second << endl;

  // expected: 5 -15
  if (!linear_diophantine(7, 2, 5, x, y)) cout <<
      "ERROR" << endl;
  cout << x << " " << y << endl;
  return 0;
}
```

4.2 NumberT

```
// ------ NUMBER THEORY ------
string bin(int n) { return bitset<32>(n).to_string(); }
// binary of n

int isPrime(int a) {
    if (a == 1)
        return 0;
    for (int i = 2; i * i <= a; i++) {
        if (a % i == 0)
            return 0;
    }
    return 1;
}
// O(sqrt(n))

int GCD_extended(int a, int b, int &x, int &y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
// return gcd(a, b), and x, y such that ax + by =
    gcd(a, b)

int binaryExponentiation(int x, int n) {
    if (n == 0) {
        return 1;
    } else if (n % 2 == 0) {
        return binaryExponentiation(x * x, n / 2);
    } else {
        return x * binaryExponentiation(x * x, (n - 1)
            / 2);
    }
}
// x^n in O(log n)

int binaryExponentiation_mod(int r, int y, int p) {
    int res = 1;
    r = r % p;
    while (y > 0) {
        if (y & 1) {
            res = (res * r) % p;
        }
        y = y >> 1;
        r = (r * r) % p;
    }
    return res;
}
// r^y mod p in O(log y)

vector<int> primeFactorisation(int n) {
    vector<int> v;
    while (n % 2 == 0) {
        v.push_back(2);
        n = n / 2;
    }
    for (int i = 3; i <= sqrt(n); i = i + 2) {
        while (n % i == 0) {
            v.push_back(i);
            n = n / i;
        }
    }
    if (n > 2)
        v.push_back(n);
    return v;
} // O(sqrt(N)) , 24 = 2^3 * 3^1

vector<int> All_divisors(int n) {
    vector<int> v;
    for (int i = 1; i * i <= n; ++i) {
        if (n % i == 0) {
            v.push_back(i);
        }
    }
    for (int i = (int)sqrt(n); i >= 1; --i) {
        if (n % i == 0) {
            if (n / i != i) {
                v.push_back(n / i);
            }
        }
    }
    return v;
} // O(sqrtN), 24 = {1,2,3,4,6,8,12,24}

vector<int> SieveOfEratosthenes(int n) {
    bool prime[n + 1];
    memset(prime, true, sizeof(prime));
    for (int p = 2; p * p <= n; p++) {
        if (prime[p] == true) {
            for (int i = p * p; i <= n; i += p)
                prime[i] = false;
        }
    }
    vector<int> v;
    for (int p = 2; p <= n; p++)
        if (prime[p]) {
            v.push_back(p);
        };
    return v;
} // O(Nlog(logN)) , generate all prime numbers till n

vector<int> primeFactors(int n) {
    vector<int> v;
    for (int i = 2; i * i <= n; i++) {
        while (n % i == 0) {
            v.push_back(i);
            n = n / i;
        }
    }
    if (n > 1)
        v.push_back(n);
    v.erase(unique(v.begin(), v.end()), v.end());
    return v;
}
// O(sqrt(N)), 24 = {2,3}
```

5 Others

5.1 CSP

```
// Constraint satisfaction problems
// TODO doesn't compiles



#define DONE -1
#define FAILED -2

typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<vvi> vvvi;

typedef set<int> SI;

// Lists of assigned/unassigned variables.
vi assigned_vars;
SI unassigned_vars;

// For each variable, a list of reductions (each of
    which a list of eliminated
// variables)
vvvi reductions;

// For each variable, a list of the variables whose
    domains it reduced in
// forward-checking.
vvi forward_mods;

// need to implement ----------------------------
int Value(int var);

void SetValue(int var, int value);
void ClearValue(int var);

int DomainSize(int var);
void ResetDomain(int var);
void AddValue(int var, int value);
void RemoveValue(int var, int value);

int NextVar() {
  if ( unassigned_vars.empty() ) return DONE;

  // could also do most constrained...
  int var = *unassigned_vars.begin();
  return var;
}

int Initialize() {
  // setup here
  return NextVar();
}
// ------------------------- end -- need to implement


void UpdateCurrentDomain(int var) {
  ResetDomain(var);
  for (int i = 0; i < reductions[var].size(); i++) {
    vector<int>& red = reductions[var][i];
    for (int j = 0; j < red.size(); j++) {
      RemoveValue(var, red[j]);
    }
  }
}

void UndoReductions(int var) {
  for (int i = 0; i < forward_mods[var].size(); i++) {
    int other_var = forward_mods[var][i];
    vi& red = reductions[other_var].back();
    for (int j = 0; j < red.size(); j++) {
      AddValue(other_var, red[j]);
    }
    reductions[other_var].pop_back();
  }
  forward_mods[var].clear();
}


bool ForwardCheck(int var, int other_var) {
  vector<int> red;

  foreach value in current_domain(other_var) {
    SetValue(other_var, value);
    if ( !Consistent(var, other_var) ) {
      red.push_back(value);
      RemoveValue(other_var, value);
    }
    ClearValue(other_var);
  }
  if ( !red.empty() ) {
    reductions[other_var].push_back(red);
    forward_mods[var].push_back(other_var);
  }

  return DomainSize(other_var) != 0;
}


pair<int, bool> Unlabel(int var) {
  assigned_vars.pop_back();
  unassigned_vars.insert(var);

  UndoReductions(var);
  UpdateCurrentDomain(var);

  if ( assigned_vars.empty() ) return make_pair(FAILED,
      true);

  int prev_var = assigned_vars.back();
  RemoveValue(prev_var, Value(prev_var));
  ClearValue(prev_var);
  if ( DomainSize(prev_var) == 0 ) {
    return make_pair(prev_var, false);
  } else {
    return make_pair(prev_var, true);
  }
}


pair<int, bool> Label(int var) {
  unassigned_vars.erase(var);
  assigned_vars.push_back(var);

  bool consistent;
  foreach value in current_domain(var) {
    SetValue(var, value);
    consistent = true;
    for (int j=0; j<unassigned_vars.size(); j++) {
      int other_var = unassigned_vars[j];
      if ( !ForwardCheck(var, other_var) ) {
```

```
        RemoveValue(var, value);
        consistent = false;
        UndoReductions(var);
        ClearValue(var);
        break;
      }
    }
    if ( consistent ) return (NextVar(), true);
  }
  return make_pair(var, false);
}



void BacktrackSearch(int num_var) {
  // (next variable to mess with, whether current state
      is consistent)
  pair<int, bool> var_consistent =
      make_pair(Initialize(), true);
  while ( true ) {
    if ( var_consistent.second ) var_consistent =
        Label(var_consistent.first);
    else var_consistent = Unlabel(var_consistent.first);

    if ( var_consistent.first == DONE ) return; //
        solution found
    if ( var_consistent.first == FAILED ) return; // no
        solution
  }
}
```

5.2 Dates

```
// Routines for performing computations on dates. In
    these routines,
// months are expressed as integers from 1 to 12, days
    are expressed
// as integers from 1 to 31, and years are expressed as
    4-digit
// integers.

string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu",
    "Fri", "Sat", "Sun"};

// converts Gregorian date to integer (Julian day
    number)
int dateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian
    date: month/day/year
void intToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;
  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string intToDay (int jd){ return dayOfWeek[jd % 7]; }
```

5.3 Simplex

```
// Two-phase simplex algorithm for solving linear
    programs of
// the form (c^T is c Transpose)
//
//     maximize     c^T x
//     subject to   Ax <= b
//                  x >= 0
//
// INPUT: A -- an m x n matrix
//        b -- an m-dimensional vector
//        c -- an n-dimensional vector
//        x -- a vector where the optimal solution will
    be stored
//
// OUTPUT: value of the optimal solution (infinity if
    unbounded
//         above, nan if infeasible)
//
// To use this code, create an LPSolver object with A,
    b, and c as
// arguments. Then, call Solve(x).



typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> vi;

const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  vi B, N;
  VVD D;

  LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n + 1), B(m), D(m + 2,
        VD(n + 2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n;
        j++) D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n + i; D[i][n]
        = -1; D[i][n + 1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] =
        -c[j]; }
    N[n] = -1; D[m + 1][n] = 1;
  }

  void Pivot(int r, int s) {
    double inv = 1.0 / D[r][s];
    for (int i = 0; i < m + 2; i++) if (i != r)
      for (int j = 0; j < n + 2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] * inv;
    for (int j = 0; j < n + 2; j++) if (j != s) D[r][j]
        *= inv;
```

```
    for (int i = 0; i < m + 2; i++) if (i != r) D[i][s]
        *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
  }

  bool Simplex(int phase) {
    int x = phase == 1 ? m + 1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] ==
            D[x][s] && N[j] < N[s]) s = j;
      }
      if (D[x][s] > -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] < EPS) continue;
        if (r == -1 || D[i][n + 1] / D[i][s] < D[r][n +
            1] / D[r][s] ||
          (D[i][n + 1] / D[i][s]) == (D[r][n + 1] /
              D[r][s]) && B[i] < B[r]) r = i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }

  DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n + 1] <
        D[r][n + 1]) r = i;
    if (D[r][n + 1] < -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m + 1][n + 1] < -EPS) return
          -numeric_limits<DOUBLE>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] ==
              D[i][s] && N[j] < N[s]) s = j;
        Pivot(i, s);
      }
    }
    if (!Simplex(2)) return
        numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] =
        D[i][n + 1];
    return D[m][n + 1];
  }
};

int main() {

  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 }, { -1, -5, 0 },
    { 1, 5, 1 }, { -1, -5, -1 }
  };
  DOUBLE _b[m] = { 10, -4, 5, -5 };
  DOUBLE _c[n] = { 1, -1, 0 };

  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] +
      n);

  LPSolver solver(A, b, c);
  VD x;
  DOUBLE value = solver.Solve(x);

  cerr << "VALUE: " << value << endl; // VALUE: 1.29032
  cerr << "SOLUTION:"; // SOLUTION: 1.74194 0.451613 1
  for (size_t i = 0; i < x.size(); i++) cerr << " " <<
      x[i];
  cerr << endl;
  return 0;
}
```

5.4 template

```
/* Adarsh Anand */
/* This too shall pass */
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define ll long long
const int
    INF=1e18,N=1e5+5,MOD1=1e9+7,MOD2=998244353,EPS=1e-9;
void __print(int x) { cerr << x; } void __print(long x)
    { cerr << x; } void __print(unsigned x) { cerr <<
    x; } void __print(unsigned long x) { cerr << x; }
    void __print(unsigned long long x) { cerr << x; }
    void __print(float x) { cerr << x; } void
    __print(double x) { cerr << x; } void __print(long
    double x) { cerr << x; } void __print(char x) {
    cerr << '\'' << x << '\''; } void __print(const
    char *x) { cerr << '\"' << x << '\"'; } void
    __print(const string &x) { cerr << '\"' << x <<
    '\"'; } void __print(bool x) { cerr << (x ? "true"
    : "false"); } template <typename T, typename V>
    void __print(const pair<T, V> &x) { cerr << '{';
    __print(x.first); cerr << ','; __print(x.second);
    cerr << '}'; } template <typename T> void
    __print(const T &x) { int f = 0; cerr << '{'; for
    (auto &i : x) cerr << (f++ ? "," : ""),
    __print(i); cerr << "}"; } void _print() { cerr <<
    "]\n"; } template <typename T, typename... V> void
    _print(T t, V... v) { __print(t); if
    (sizeof...(v)) cerr << ", "; _print(v...); }
template <typename T1, typename T2> istream&
    operator>>(istream& istream, pair<T1, T2>& p) {
    return (istream >> p.first >> p.second); }
    template <typename T> istream& operator>>(istream&
    istream, vector<T>& v) { for (auto& it : v) cin >>
    it; return istream; } template <typename T1,
    typename T2> ostream& operator<<(ostream& ostream,
    const pair<T1, T2>& p) { return (ostream <<
    p.first << " " << p.second); } template <typename
    T> ostream& operator<<(ostream& ostream, const
    vector<T>& c) { for (auto& it : c) cout << it << "
    "; return ostream; }
#ifndef ONLINE_JUDGE
#define debug(x...) cerr<<"["<<#x<<"] = ["; _print(x)
#else
#define debug(x...)
#endif
```

```
void solve(){

}

signed main() {
    ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);
    cout << setprecision(12) << fixed; int t=1;
    cin >> t;
    while (t--) { solve();
         cerr<<"------------------"<<endl; }
}
```

6 SegTree

6.1 lazyProp

```
/*
  A lazy tree implementation of Range Updation & Range
      Query
*/

ll Arr[Lim], Tree[4*Lim], lazy[4*Lim];

void build_tree(int Node, int a, int b) {
  // Do not forget to clear lazy Array before calling
      build

  if(a == b) {
    Tree[Node] = Arr[a];
  } else if (a < b) {
    int mid = (a+b)>>1, left=Node<<1, right=left|1;
    build_tree(left, a, mid); build_tree(right, mid+1,
        b);
    Tree[Node] = Tree[left]+Tree[right];
  }
}

void Propogate(int Node, int a, int b) {
  int left=Node<<1, right=left|1;
  Tree[Node]+=lazy[Node]*(b-a+1);
  if(a != b) {
    lazy[left]+=lazy[Node];
    lazy[right]+=lazy[Node];
  }
  lazy[Node] = 0;
}

void update_tree (int Node, int start, int end, ll
    value, int a, int b) {
  int mid=(a+b)>>1, left=Node<<1, right=left|1;
  if(lazy[Node] != 0)
    Propogate(Node, a, b);

  if(a > b || a > end || b < start) {
    return;
  } else {
    if(start <= a && b <= end) {
      if (a != b) {
        lazy[left] += value;
        lazy[right] += value;
      }
      Tree[Node] += value * (b - a + 1);
    } else {
      update_tree(left, start, end, value, a, mid);
      update_tree(right, start, end, value, mid+1, b);
      Tree[Node]=Tree[left]+Tree[right];
    }
  }

ll query(int Node, int start, int end, int a, int b) {
  int mid=(a+b)>>1, left=Node<<1, right=left|1;
  if(lazy[Node] != 0)
    Propogate(Node, a, b);

  if (a > b || a > end || b < start) {
    return 0;
  } else {
    ll Sum1, Sum2;
    if (start <= a && b <= end) {
      return Tree[Node];
    } else {
      Sum1 = query(left, start, end, a, mid);
      Sum2 = query(right, start, end, mid + 1, b);
      return Sum1+Sum2;
    }
  }
}
```

6.2 RMQ

```
/* matrix structure for finding the range minima in
    O(1) time using O(n) log(n)) space */

#define better(a, b) A[a] < A[b] ? (a) : (b)
int A[100100], H[1100][1100]; // A is the Array and H
    is the lookup matrix
int make_dp(int n) {       // N log N
    rep(i, n) H[i][0] = i;
    for (int l = 0, k; (k = 1 << l) < n; l++)
        for (int i = 0; i + k < n; i++)
            H[i][l + 1] = better(H[i][l], H[i + k][l]);
}
int query_dp(int a, int b) {
    int l = __lg(b - a);
    return better(H[a][l], H[b - (1 << l) + 1][l]);
}
```

6.3 segmentTree

```
/*
  Segment Tree for Range Minima Query, Can be modified
      easily for
  other cases.

  Deal Everything in one based indexing
*/

ll Arr[Lim], Tree[4*Lim];

void buildTree(int Node, int a, int b) {
  if(a == b) {
    Tree[Node]=Arr[a];
  } else if (a < b) {
    int mid=(a+b)>>1, left=Node<<1;
```

```
    int right=left|1;
    buildTree(left, a, mid);
    buildTree(right, mid+1, b);
    Tree[Node] = min(Tree[left], Tree[right]);
  }
}

void updateTree(int Node, ll value, int a, int b, int
     index) {
  if (a > index || b < index) {

  } else if (a == b) {
    Tree[Node] = value;
    Arr[index] = value;
  } else if (a <= index && b >= index) {
    int mid=(a+b)>>1, left=Node<<1;
    int right=left|1;
    updateTree(left, index, value, a, mid);
    updateTree(right, index, value, mid+1, b);
    Tree[Node]=min(Tree[left], Tree[right]);
  }
 }

ll queryTree(int Node, int start, int end, int a, int
     b) {
  int mid=(a+b)>>1, left=Node<<1;
  int right=left|1;
  ll Ans = Inf;
  if (start <= a && b <= end) {
    return Tree[Node];
  } else {
    if(mid >= start)
      Ans = queryTree(left, start, end, a, mid);

    if(mid < end)
      Ans = min(Ans, queryTree(right, start, end,
           mid+1, b));

    return Ans;
  }
}
```

6.4 segTreeN

```
template<typename T>
struct segTree {
  T Tree[4*Lim];
  T combine(int l, int r) {
    T ret;
    ret=min(l, r); // TODO
    return ret;
  }
  void buildST(int Node, int a, int b) {
    if (a==b)
      Tree[Node]=0; // TODO
    else if (a<b) {
      int left=Node<<1, right=(Node<<1)|1, mid=(a+b)>>1;
      buildST(left, a, mid); buildST(right, mid+1, b);
      Tree[Node]=combine(Tree[left], Tree[right]);
    }
  }
  void buildST(int Node, int a, int b, vi Arr) {
    if (a==b)
      Tree[Node]=Arr[a];
    else if (a<b) {
      int left=Node<<1, mid=(a+b)>>1, right=(Node<<1)|1;
      buildST(left, a, mid, Arr); buildST(right, mid+1,
           b, Arr);
      Tree[Node]=combine(Tree[left], Tree[right]);
    }
  }
  T query(int Node, int a, int b, int S, int E) {
    if (E < a || b < S) return 0; // TODO
    else if (a==b) return Tree[Node];
    int left=Node<<1, mid=(a+b)>>1, right=(Node<<1)|1;
    if (S <= a && b <= E) return Tree[Node];
    return combine(query(left, a, mid, S, E),
         query(right, mid+1, b, S, E));
  }
  void update(int Node, int a, int b, int val, int I1,
       int I2) {
    if (I2 < a || b < I1) return;
    if (I1 <=a && b <= I2) return void(Tree[Node]=val);
         // TODO
    int left=Node<<1, mid=(a+b)>>1, right=(Node<<1)|1;
    update(left, a, mid, val, I1, I2), update(right,
         mid+1, b, val, I1, I2);
    Tree[Node]=combine(Tree[left], Tree[right]);
  }
};

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

7 Snippet

7.1 Snippet

```
{
"cmd": ["g++.exe","-std=c++14", "${file}", "-o",
    "${file_base_name}.exe", "&&" ,
    "${file_base_name}.exe<inputf.in>outputf.in"],
"selector":"source.cpp",
"shell":true,
"working_dir":"$file_path"
}
```

8 STL

8.1 next-permutation

```
// Example for Array:
int a[N];
sort(a, a+N);
next_permutation(a, a+N);
// Example for vector:
vector<int> ivec;
sort(ivec.begin(), ivec.end());
next_permutation(ivec.begin(), ivec.end());
//Example for loop:
vector<int> myVec;
sort(myVec.begin(),myVec.end());
do{
    for (i = 0 ;i < size;i ++ ) cout << myVec[i] << "
         \t " ;
    cout << endl;
```

```
}while (next_permutation(myVec.begin(), myVec.end()));
```

8.2 nth-element

```
struct node
{
    int val,pos;
}A[10];

int cmp(node a,node b)
{
    return a.pos < b.pos;
}

int main()
{
    int a[10] = {-1,3,9,1,4,5,8,7,6,2};
    int i;
    while(0){
        cin >> i;
        nth_element(a+1,a+i,a+9+1);
        cout << a[i];
    }
    for(int i=1;i<=9;i++) A[i].pos = 10-i,A[i].val=i;
    nth_element(A+1,A+4,A+9+1,cmp);
    printf("%d\n",A[4].pos);
}
```

8.3 priority-queue

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

struct cmp{
    bool operator()(node a,node b)
    {
        if(a.x==b.x) return a.y > b.y;
        return a.x < b.x;
    }
};

priority_queue<int,vector<int>,greater<int> > q;
priority_queue<node,vector<node>,cmp > qq;
```

9 String

9.1 KMP

```
/*
Searches for the string w in the string s (of length
    k). Returns the
0-based index of the first match (k if no match is
    found). Algorithm
runs in O(k) time.
*/

#include <iostream>
#include <string>
#include <vector>

using namespace std;

typedef vector<int> VI;

void buildTable(string& w, VI& t)
{
  t = VI(w.length());
  int i = 2, j = 0;
  t[0] = -1; t[1] = 0;

  while(i < w.length())
  {
    if(w[i-1] == w[j]) { t[i] = j+1; i++; j++; }
    else if(j > 0) j = t[j];
    else { t[i] = 0; i++; }
  }
}

int KMP(string& s, string& w)
{
  int m = 0, i = 0;
  VI t;

  buildTable(w, t);
  while(m+i < s.length())
  {
    if(w[i] == s[m+i])
    {
      i++;
      if(i == w.length()) return m;
    }
    else
    {
      m += i-t[i];
      if(i > 0) i = t[i];
    }
  }
  return s.length();
}

int main()
{
  string a = (string) "The example above illustrates
      the general technique for assembling "+
    "the table with a minimum of fuss. The principle is
        that of the overall search: "+
    "most of the work was already done in getting to
        the current position, so very "+
    "little needs to be done in leaving it. The only
        minor complication is that the "+
    "logic which is correct late in the string
        erroneously gives non-proper "+
    "substrings at the beginning. This necessitates
        some initialization code.";

  string b = "table";

  int p = KMP(a, b);
  cout << p << ": " << a.substr(p, b.length()) << " "
      << b << endl;
}
```

9.2 zAlgo

```
/* Z algorithm for matching substrings. KMP's Brother */

vector<int> zfunction(char *s) {
  int N = strlen(s), a=0, b=0;
  vector<int> z(N, N);
  for (int i = 1; i < N; i++) {
    int k = i<b ? min(b-i, z[i-a]) : 0;
    while (i+k < N && s[i+k]==s[k]) ++k;
    z[i] = k;
    if (i+k > b) { a=i; b=i+k; }
  }
  return z;
}

Definition:
z[i] = max {k: s[i..i+k-1]=s[0..k-1]}
```

10 Tries

10.1 Tries

```
struct Node {
    Node *links[26];
    bool flag = false;

    bool hasKey(char c) {
        return links[c - 'a'] != NULL;
    }
    void insertKey(char c, Node *node) {
        links[c - 'a'] = node;
    }
    Node *next(char c) {
        return links[c - 'a'];
    }
    void setTrue() {
        flag = true;
    }
};

class Trie {
private:
    Node *root;

public:
    Trie() {
        root = new Node();
    }

    void insert(string word) {
        Node *node = root;
        for (char c : word) {
            if (!(node->hasKey(c))) {
                node->insertKey(c, new Node());
            }
            // go next node
            node = node->next(c);
        }
        node->setTrue();
    }

    bool search(string word) {
        Node *node = root;
        for (char c : word) {
            if (!(node->hasKey(c))) {
                return false;
            }
            // go next node
            node = node->next(c);
        }
        return node->flag;
    }

    bool startsWith(string prefix) {
        Node *node = root;
        for (char c : prefix) {
            if (!(node->hasKey(c))) {
                return false;
            }
            // go next node
            node = node->next(c);
        }
        return true;
    }
};

/**
 * Your Trie object will be instantiated and called as
     such:
 * Trie* obj = new Trie();
 * obj->insert(word);
 * bool param_2 = obj->search(word);
 * bool param_3 = obj->startsWith(prefix);
 */
```