Contents

1 Combinatorial optimization

1.1 Dinic's

```
struct Dinic {
struct Edge {
  int u, v;
  long long cap, flow;
  Edge() {}
  Edge(int u, int v, long long cap): u(u), v(v), cap(
      cap), flow(0) {}
};
int N;
vector<Edge> E;
vector<vector<int>> g;
vector<int> d, pt;

Dinic(int N): N(N), E(0), g(N), d(N), pt(N) {}

void AddEdge(int u, int v, long long cap) {
  if (u != v) {
    E.emplace_back(Edge(u, v, cap));
    g[u].emplace_back(E.size() - 1);
    E.emplace_back(Edge(v, u, 0));
    g[v].emplace_back(E.size() - 1);
  }
}

bool BFS(int S, int T) {
  queue<int> q({S});
  fill(d.begin(), d.end(), N + 1);
  d[S] = 0;
  while(!q.empty()) {
    int u = q.front(); q.pop();
    if (u == T) break;
    for (int k: g[u]) {
      Edge &e = E[k];
      if (e.flow < e.cap && d[e.v] > d[e.u] + 1) {
        d[e.v] = d[e.u] + 1;
        q.emplace(e.v);
      }
    }
  }
  return d[T] != N + 1;
}

long long DFS(int u, int T, long long flow = -1) {
  if (u == T || flow == 0) return flow;
  for (int &i = pt[u]; i < g[u].size(); ++i) {
    Edge &e = E[g[u][i]];
    Edge &oe = E[g[u][i]^1];
    if (d[e.v] == d[e.u] + 1) {
      long long amt = e.cap - e.flow;
      if (flow != -1 && amt > flow) amt = flow;
      if (long long pushed = DFS(e.v, T, amt)) {
        e.flow += pushed;
        oe.flow -= pushed;
        return pushed;
      }
    }
  }
  return 0;
}

long long MaxFlow(int S, int T) {
  long long total = 0;
  while (BFS(S, T)) {
    fill(pt.begin(), pt.end(), 0);
```

```
      while (long long flow = DFS(S, T))
        total += flow;
    }
    return total;
  }
};
```

1.2 Min-cost max-flow

```
// 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
```

1.3 Edmonds Max Matching

```
/*
Input:
V->number of vertices
E->number of edges
pair of vertices as edges (vertices are 1..V)

Output:
output of edmonds() is the maximum matching
match[i] is matched pair of i (-1 if there isn't a
    matched pair)
 */

#include <bits/stdc++.h>
using namespace std;
const int M=505;
struct struct_edge{int v;struct_edge* n;};
typedef struct_edge* edge;
struct_edge pool[M*M*2];
edge top=pool,adj[M];
int V,E,match[M],qh,qt,q[M],father[M],base[M];
bool inq[M],inb[M],ed[M][M];
void add_edge(int u,int v)
{
  top->v=v,top->n=adj[u],adj[u]=top++;
  top->v=u,top->n=adj[v],adj[v]=top++;
}
int LCA(int root,int u,int v)
{
  static bool inp[M];
  memset(inp,0,sizeof(inp));
  while(1)
    {
      inp[u=base[u]]=true;
      if (u==root) break;
      u=father[match[u]];
    }
  while(1)
    {
      if (inp[v=base[v]]) return v;
      else v=father[match[v]];
    }
}
void mark_blossom(int lca,int u)
{
  while (base[u]!=lca)
    {
      int v=match[u];
      inb[base[u]]=inb[base[v]]=true;
      u=father[v];
      if (base[u]!=lca) father[u]=v;
    }
}
void blossom_contraction(int s,int u,int v)
{
  int lca=LCA(s,u,v);
  memset(inb,0,sizeof(inb));
  mark_blossom(lca,u);
  mark_blossom(lca,v);
  if (base[u]!=lca)
    father[u]=v;
  if (base[v]!=lca)
    father[v]=u;
  for (int u=0;u<V;u++)
    if (inb[base[u]])
      {
  base[u]=lca;
  if (!inq[u])
    inq[q[++qt]=u]=true;
      }
}
int find_augmenting_path(int s)
{
  memset(inq,0,sizeof(inq));
  memset(father,-1,sizeof(father));
  for (int i=0;i<V;i++) base[i]=i;
  inq[q[qh=qt=0]=s]=true;
  while (qh<=qt)
    {
      int u=q[qh++];
      for (edge e=adj[u];e;e=e->n)
        {
    int v=e->v;
    if (base[u]!=base[v]&&match[u]!=v)
      if ((v==s)||(match[v]!=-1 && father[match[v]]!=-1)
        )
        blossom_contraction(s,u,v);
      else if (father[v]==-1)
        {
    father[v]=u;
    if (match[v]==-1)
      return v;
    else if (!inq[match[v]])
      inq[q[++qt]=match[v]]=true;
        }
        }
    }
```

```
  return -1;
}
int augment_path(int s,int t)
{
  int u=t,v,w;
  while (u!=-1)
    {
      v=father[u];
      w=match[v];
      match[v]=u;
      match[u]=v;
      u=w;
    }
  return t!=-1;
}
int edmonds()
{
  int matchc=0;
  memset(match,-1,sizeof(match));
  for (int u=0;u<V;u++)
    if (match[u]==-1)
      matchc+=augment_path(u,find_augmenting_path(u));
  return matchc;
}
int main()
{
  int u,v;
  cin>>V>>E;
  while(E--)
    {
      cin>>u>>v;
      if (!ed[u-1][v-1])
  {
    add_edge(u-1,v-1);
    ed[u-1][v-1]=ed[v-1][u-1]=true;
  }
    }
  cout<<edmonds()<<endl;
  for (int i=0;i<V;i++)
    if (i<match[i])
      cout<<i+1<<" "<<match[i]+1<<endl;
}
```

2 Geometry

2.1 Convex hull

```
typedef pair<long long, long long> PT;
long double dist(PT a, PT b){
  return sqrt(pow(a.first-b.first,2)+pow(a.second-b.
      second,2));
}

long long cross(PT o, PT a, PT b){
  PT OA = {a.first-o.first,a.second-o.second};
  PT OB = {b.first-o.first,b.second-o.second};
  return OA.first*OB.second - OA.second*OB.first;
}
vector<PT> convexhull(){
  vector<PT> hull;
  sort(a,a+n,[](PT i, PT j){
    if(i.second!=j.second)
      return i.second < j.second;
    return i.first < j.first;
  });
  for(int i=0;i<n;++i){
    while(hull.size()>1 && cross(hull[hull.size()-2],
        hull.back(),a[i])<=0)
      hull.pop_back();
    hull.push_back(a[i]);
  }
  for(int i=n-1, siz = hull.size();i--;){
    while(hull.size()>siz && cross(hull[hull.size()-2],
        hull.back(),a[i])<=0)
      hull.pop_back();
    hull.push_back(a[i]);
  }
  return hull;
}
```

2.2 Miscellaneous geometry

```
double INF = 1e100,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;
}
```

2.3 3D geometry

```
public class Geom3D {
// distance from point (x, y, z) to plane aX + bY + cZ + d = 0
public static double ptPlaneDist(double x, double y, double z,
    double a, double b, double c, double d) {
  return Math.abs(a*x + b*y + c*z + d) / Math.sqrt(a*a + b*b + c*c);
}

// distance between parallel planes aX + bY + cZ + d1 = 0 and
// aX + bY + cZ + d2 = 0
public static double planePlaneDist(double a, double b, double c,
    double d1, double d2) {
  return Math.abs(d1 - d2) / Math.sqrt(a*a + b*b + c*c);
}

// distance from point (px, py, pz) to line (x1, y1, z1)-(x2, y2, z2)
// (or ray, or segment; in the case of the ray, the endpoint is the
// first point)
public static final int LINE = 0;
public static final int SEGMENT = 1;
public static final int RAY = 2;
public static double ptLineDistSq(double x1, double y1, double z1,
    double x2, double y2, double z2, double px, double py, double pz,
    int type) {
  double pd2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) + (z1-z2)*(z1-z2);

  double x, y, z;
  if (pd2 == 0) {
    x = x1;
    y = y1;
    z = z1;
  } else {
    double u = ((px-x1)*(x2-x1) + (py-y1)*(y2-y1) + (pz-z1)*(z2-z1)) / pd2;
    x = x1 + u * (x2 - x1);
    y = y1 + u * (y2 - y1);
    z = z1 + u * (z2 - z1);
    if (type != LINE && u < 0) {
      x = x1;
      y = y1;
      z = z1;
    }
    if (type == SEGMENT && u > 1.0) {
      x = x2;
      y = y2;
      z = z2;
    }
  }

  return (x-px)*(x-px) + (y-py)*(y-py) + (z-pz)*(z-pz);
}

public static double ptLineDist(double x1, double y1, double z1,
```

```
    double x2, double y2, double z2, double px, double py, double pz,
    int type) {
  return Math.sqrt(ptLineDistSq(x1, y1, z1, x2, y2, z2, px, py, pz, type));
 }
}
```

3 Numerical algorithms

3.1 Number theory (modular, Chinese remainder, linear Diophantine)

```
// All algorithms described here work on nonnegative integers.

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

// 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){
  return b?powermod(a*a%m,b/2,m)*(b%2?a:1)%m:1;
}

// returns g = gcd(a, b); finds x, y such that d = 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;
}
```

3.2 Systems of linear equations, matrix inverse, determinant

```cpp
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
// OUTPUT:   X      = an nxm matrix (stored in b[][])
//           A^{-1} = an nxn matrix (stored in a[][])
//           returns determinant of a[][]

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

using namespace std;

const double EPS = 1e-10;

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

T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;

  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
      for (int k = 0; k < n; k++) if (!ipiv[k])
        if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk]))
          { pj = j; pk = k; }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular." << endl; exit(0); }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj;
    icol[i] = pk;

    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p] *= c;
    for (int p = 0; p < m; p++) b[pk][p] *= c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
    }
  }

  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol[p]]);
  }

  return det;
}

int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = {
    {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6} };
  double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
  VVT a(n), b(n);
  for (int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }

  double det = GaussJordan(a, b);

  // expected: 60
  cout << "Determinant: " << det << endl;

  // expected: -0.233333 0.166667 0.133333 0.0666667
  //           0.166667 0.166667 0.333333 -0.333333
  //           0.233333 0.833333 -0.133333 -0.0666667
  //           0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

  // expected: 1.63333 1.3
  //           -0.166667 0.5
  //           2.36667 1.7
  //           -1.85 -1.35
  cout << "Solution: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

3.3 Reduced row echelon form, matrix rank

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

```
// OUTPUT:   rref[][] = an nxm matrix (stored in a[][])
//           returns rank of a[][]

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

using namespace std;

const double EPSILON = 1e-10;

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

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

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

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

  int rank = rref(a);

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

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

3.4 Simplex algorithm

```
// Two-phase simplex algorithm for solving linear
   programs of the form
//
//     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).

#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

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;
}
```

3.5 Fast Fourier transform

```
auto FFT = [](vector<long double>a,vector<long double>b)
    {
  auto DFT = [](vector<complex<long double>>&a, bool inv
      ){
    int L=31-__builtin_clz(a.size()), n=1<<L;
    vector<complex<long double>> A(n);
    for(int k=0,r,i;k<n;A[r]=a[k++])
      for(i=r=0;i<L;(r<<=1)|=(k>>i++)&1);
    complex<long double> w,wm,t;
    for(int m=2,j,k;m<=n;m<<=1)
      for(w={0,2*acos(-1)/m},wm=exp(inv?-w:w),k=0;k<n;k
          +=m)
        for(j=0,w=1;j<m/2;++j,w*=wm)
          t=w*A[k+j+m/2],A[k+j+m/2]=A[k+j]-t,A[k+j]+=t;
    return A;
  };
  int n=4<<31-__builtin_clz(max(a.size(),b.size()));
  vector<complex<long double>> A(n), B(n), CC(n);
  for(int i=0;i<n;++i)
    A[i]=i<a.size()?a[i]:0,B[i]=i<b.size()?b[i]:0;
  vector<complex<long double>> AA=DFT(A,0), BB=DFT(B,0);
  for(int i=0;i<n;++i) CC[i]=AA[i]*BB[i];
  vector<long double> c;
  for(auto i:DFT(CC,1)) if(c.size()<a.size()+b.size()-1)
    c.push_back(i.real()/n+1e-5);
  return c;
};
```

3.6 BigInt library

```
struct bigint {
  const int base = 1000000000, base_digits = 9;
  vector<int> a;
  int sign;
  bigint() : sign(1) {}
  bigint(long long v) {
    *this = v;
  }
```

```
bigint(const string &s) {
  read(s);
}
void operator=(const bigint &v) {
  sign = v.sign;
  a = v.a;
}
void operator=(long long v) {
  sign = 1;
  if (v < 0) sign = -1, v = -v;
  for (; v > 0; v = v / base)
    a.push_back(v % base);
}
bigint operator+(const bigint &v) const {
  if (sign == v.sign) {
    bigint res = v;

    for (int i = 0, carry = 0; i < (int) max(a.size(),
        v.a.size()) || carry; ++i) {
      if (i == (int) res.a.size())
        res.a.push_back(0);
      res.a[i] += carry + (i < (int) a.size() ? a[i] :
          0);
      carry = res.a[i] >= base;
      if (carry)
        res.a[i] -= base;
    }
    return res;
  }
  return *this - (-v);
}
bigint operator-(const bigint &v) const {
  if (sign == v.sign) {
    if (abs() >= v.abs()) {
      bigint res = *this;
      for (int i = 0, carry = 0; i < (int) v.a.size()
          || carry; ++i) {
        res.a[i] -= carry + (i < (int) v.a.size() ? v.
            a[i] : 0);
        carry = res.a[i] < 0;
        if (carry)
          res.a[i] += base;
      }
      res.trim();
      return res;
    }
    return -(v - *this);
  }
  return *this + (-v);
}
void operator*=(int v) {
  if (v < 0)
    sign = -sign, v = -v;
  for (int i = 0, carry = 0; i < (int) a.size() ||
      carry; ++i) {
    if (i == (int) a.size())
      a.push_back(0);
    long long cur = a[i] * (long long) v + carry;
    carry = (int) (cur / base);
    a[i] = (int) (cur % base);
    //asm("divl %%ecx" : "=a"(carry), "=d"(a[i]) : "A
         "(cur), "c"(base));
  }
  trim();
}
bigint operator*(int v) const {
  bigint res = *this;
  res *= v;
  return res;
}
friend pair<bigint, bigint> divmod(const bigint &a1,
    const bigint &b1) {
  int norm = a1.base / (b1.a.back() + 1);
  bigint a = a1.abs() * norm;
  bigint b = b1.abs() * norm;
  bigint q, r;
  q.a.resize(a.a.size());
  for (int i = a.a.size() - 1; i >= 0; i--) {
    r *= a1.base;
    r += a.a[i];
    int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.
        size()];
    int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.
        a.size() - 1];
    int d = ((long long) a1.base * s1 + s2) / b.a.back
        ();
    r -= b * d;
    while (r < 0)
      r += b, --d;
    q.a[i] = d;
  }
  q.sign = a1.sign * b1.sign;
  r.sign = a1.sign;
  q.trim();
  r.trim();
  return make_pair(q, r / norm);
}
bigint operator/(const bigint &v) const {
  return divmod(*this, v).first;
}
bigint operator%(const bigint &v) const {
  return divmod(*this, v).second;
}
void operator/=(int v) {
  if (v < 0) sign = -sign, v = -v;
  for (int i = (int) a.size() - 1, rem = 0; i >= 0; --
      i) {
    long long cur = a[i] + rem * (long long) base;
    a[i] = (int) (cur / v);
    rem = (int) (cur % v);
  }
  trim();
}
bigint operator/(int v) const {
  bigint res = *this;
  res /= v;
  return res;
}
int operator%(int v) const {
  if (v < 0)
    v = -v;
  int m = 0;
  for (int i = a.size() - 1; i >= 0; --i)
```

```cpp
      m = (a[i] + m * (long long) base) % v;
    return m * sign;
  }
  void operator+=(const bigint &v) {
    *this = *this + v;
  }
  void operator-=(const bigint &v) {
    *this = *this - v;
  }
  void operator*=(const bigint &v) {
    *this = *this * v;
  }
  void operator/=(const bigint &v) {
    *this = *this / v;
  }
  bool operator<(const bigint &v) const {
    if (sign != v.sign)
      return sign < v.sign;
    if (a.size() != v.a.size())
      return a.size() * sign < v.a.size() * v.sign;
    for (int i = a.size() - 1; i >= 0; i--)
      if (a[i] != v.a[i])
        return a[i] * sign < v.a[i] * sign;
    return false;
  }
  bool operator>(const bigint &v) const {
    return v < *this;
  }
  bool operator<=(const bigint &v) const {
    return !(v < *this);
  }
  bool operator>=(const bigint &v) const {
    return !(*this < v);
  }
  bool operator==(const bigint &v) const {
    return !(*this < v) && !(v < *this);
  }
  bool operator!=(const bigint &v) const {
    return *this < v || v < *this;
  }
  void trim() {
    while (!a.empty() && !a.back())
      a.pop_back();
    if (a.empty())
      sign = 1;
  }
  bool isZero() const {
    return a.empty() || (a.size() == 1 && !a[0]);
  }
  bigint operator-() const {
    bigint res = *this;
    res.sign = -sign;
    return res;
  }
  bigint abs() const {
    bigint res = *this;
    res.sign *= res.sign;
    return res;
  }
  long long longValue() const {
    long long res = 0;
    for (int i = a.size() - 1; i >= 0; i--)
      res = res * base + a[i];
    return res * sign;
  }
  friend bigint gcd(const bigint &a, const bigint &b) {
    return b.isZero() ? a : gcd(b, a % b);
  }
  friend bigint lcm(const bigint &a, const bigint &b) {
    return a / gcd(a, b) * b;
  }
  void read(const string &s) {
    sign = 1;
    a.clear();
    int pos = 0;
    while (pos < (int) s.size() && (s[pos] == '-' || s[
        pos] == '+')) {
      if (s[pos] == '-') sign = -sign;
      ++pos;
    }
    for (int i = s.size() - 1; i >= pos; i -=
        base_digits) {
      int x = 0;
      for (int j = max(pos, i - base_digits + 1); j <= i
          ; j++)
        x = x * 10 + s[j] - '0';
      a.push_back(x);
    }
    trim();
  }
  friend istream& operator>>(istream &stream, bigint &v)
      {
    string s;
    stream >> s;
    v.read(s);
    return stream;
  }
  friend ostream& operator<<(ostream &stream, const
      bigint &v) {
    if (v.sign == -1) stream << '-';
    stream << (v.a.empty() ? 0 : v.a.back());
    for (int i = (int) v.a.size() - 2; i >= 0; --i)
      stream << setw(v.base_digits) << setfill('0') << v
          .a[i];
    return stream;
  }
  static vector<int> convert_base(const vector<int> &a,
      int old_digits, int new_digits) {
    vector<long long> p(max(old_digits, new_digits) + 1)
        ;
    p[0] = 1;
    for (int i = 1; i < (int) p.size(); i++)
      p[i] = p[i - 1] * 10;
    vector<int> res;
    long long cur = 0;
    int cur_digits = 0;
    for (int i = 0; i < (int) a.size(); i++) {
      cur += a[i] * p[cur_digits];
      cur_digits += old_digits;
      while (cur_digits >= new_digits) {
        res.push_back(int(cur % p[new_digits]));
        cur /= p[new_digits];
```

```
      cur_digits -= new_digits;
    }
  }
  res.push_back((int) cur);
  while (!res.empty() && !res.back())
    res.pop_back();
  return res;
}
typedef vector<long long> vll;
static vll karatsubaMultiply(const vll &a, const vll &
   b) {
  int n = a.size();
  vll res(n + n);
  if (n <= 32) {
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        res[i + j] += a[i] * b[j];
    return res;
  }
  int k = n >> 1;
  vll a1(a.begin(), a.begin() + k);
  vll a2(a.begin() + k, a.end());
  vll b1(b.begin(), b.begin() + k);
  vll b2(b.begin() + k, b.end());
  vll a1b1 = karatsubaMultiply(a1, b1);
  vll a2b2 = karatsubaMultiply(a2, b2);
  for (int i = 0; i < k; i++)
    a2[i] += a1[i];
  for (int i = 0; i < k; i++)
    b2[i] += b1[i];
  vll r = karatsubaMultiply(a2, b2);
  for (int i = 0; i < (int) a1b1.size(); i++)
    r[i] -= a1b1[i];
  for (int i = 0; i < (int) a2b2.size(); i++)
    r[i] -= a2b2[i];
  for (int i = 0; i < (int) r.size(); i++)
    res[i + k] += r[i];
  for (int i = 0; i < (int) a1b1.size(); i++)
    res[i] += a1b1[i];
  for (int i = 0; i < (int) a2b2.size(); i++)
    res[i + n] += a2b2[i];
  return res;
}
bigint operator*(const bigint &v) const {
  vector<int> a6 = convert_base(this->a, base_digits,
     6);
  vector<int> b6 = convert_base(v.a, base_digits, 6);
  vll a(a6.begin(), a6.end());
  vll b(b6.begin(), b6.end());
  while (a.size() < b.size())
    a.push_back(0);
  while (b.size() < a.size())
    b.push_back(0);
  while (a.size() & (a.size() - 1))
    a.push_back(0), b.push_back(0);
  vll c = karatsubaMultiply(a, b);
  bigint res;
  res.sign = sign * v.sign;
  for (int i = 0, carry = 0; i < (int) c.size(); i++)
     {
    long long cur = c[i] + carry;
      res.a.push_back((int) (cur % 1000000));
      carry = (int) (cur / 1000000);
    }
    res.a = convert_base(res.a, 6, base_digits);
    res.trim();
    return res;
  }
};
```

4 Graph algorithms

4.1 Bellman-Ford shortest paths with negative edge weights

```
// Single source shortest paths with negative edge
   weights.
// Returns false if a negative weight cycle is detected.
// 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
//            dad[i] = prevector<int>ous node on the
   best path from the
//                      start node
vector<int> dad;
vector<double> dist;
bool BellmanFord(int start, vector<vector<double>> &w){
  int n = w.size();
  dad = vector<int>(n, -1);
  dist = vector<double>(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;
          else dist[j] = dist[i] + w[i][j], dad[j] = i;
        }

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

4.2 Eulerian path

```
struct Edge;
typedef list<Edge>::iterator iter;

struct Edge
{
        int next_vertex;
        iter reverse_edge;

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

const int max_vertices = ;
int num_vertices;
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;
}
```

4.3 Minimum spanning trees

```
// This function runs Prim's algorithm for constructing minimum
// weight spanning trees.
//
// Running time: O(|V|^2)
//
//   INPUT:   w[i][j] = cost of edge from i to j
//
//            NOTE: Make sure that w[i][j] is nonnegative and
//            symmetric.  Missing edges should be given -1
//            weight.
//
//   OUTPUT:  edges = list of pair<int,int> in minimum spanning tree
//            return total weight of tree

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

T Prim (const VVT &w, VPII &edges){
  int n = w.size();
  VI found (n);
  VI prev (n, -1);
  VT dist (n, 1000000000);
  int here = 0;
  dist[here] = 0;

  while (here != -1){
    found[here] = true;
    int best = -1;
    for (int k = 0; k < n; k++) if (!found[k]){
      if (w[here][k] != -1 && dist[k] > w[here][k]){
        dist[k] = w[here][k];
        prev[k] = here;
      }
      if (best == -1 || dist[k] < dist[best]) best = k;
    }
    here = best;
  }

  T tot_weight = 0;
  for (int i = 0; i < n; i++) if (prev[i] != -1){
    edges.push_back (make_pair (prev[i], i));
    tot_weight += w[prev[i]][i];
  }
  return tot_weight;
}

int main(){
  int ww[5][5] = {
    {0, 400, 400, 300, 600},
    {400, 0, 3, -1, 7},
    {400, 3, 0, 2, 0},
    {300, -1, 2, 0, 5},
    {600, 7, 0, 5, 0}
  };
  VVT w(5, VT(5));
  for (int i = 0; i < 5; i++)
    for (int j = 0; j < 5; j++)
      w[i][j] = ww[i][j];

  // expected: 305
  //           2 1
  //           3 2
  //           0 3
  //           2 4

  VPII edges;
  cout << Prim (w, edges) << endl;
  for (int i = 0; i < edges.size(); i++)
    cout << edges[i].first << " " << edges[i].second << endl;
}
```

4.4 Centroid decomposition

```
set<int> v[100005];
map<int,int> mp[100005];
int n,up[100005][17],lvl[100005],par[100005],CNT,siz[100005],tin[100005],tout[100005];
void dfspre(int u, int dad=1, int depth = 0){
  static int clk = 0;
  tin[u]=clk++;
  up[u][0] = dad;
  lvl[u] = depth;
```

```
    for(int i=1;i<17;++i)
      up[u][i] = up[up[u][i-1]][i-1];
    for(int i:v[u]) if(i!=dad)
      dfspre(i,u,depth+1);
    tout[u]=clk++;
  }
  int dfs(int u, int dad){
    siz[u] = 1;
    for(int i:v[u]) if(i!=dad)
      siz[u] += dfs(i,u);
    return siz[u];
  }
  int centroid(int u, int dad){
    for(int i:v[u]) if(i!=dad && siz[i]>CNT)
      return centroid(i,u);
    return u;
  }
  void decompose(int u, int dad){
    CNT = dfs(u,dad)/2;
    int centre = centroid(u,dad);
    par[centre] = dad;
    for(int i:v[centre]) if(i!=dad){
      v[i].erase(centre);
      decompose(i,centre);
    }
    v[centre].clear();
  }
  int lca(int u, int v){
    if(lvl[u]>lvl[v]) swap(u,v);
    if(tin[u]<=tin[v] && tout[v]<=tout[u]) return u;
    for(int i=17;i--;)
      if(!(tin[up[u][i]]<=tin[v] && tout[v]<=tout[up[u][i
          ]]))
        u = up[u][i];
    return up[u][0];
  }
  void update(int u){
    for(int node = u;u;u = par[u])
      ++mp[u][lvl[node]+lvl[u] - 2*lvl[lca(u,node)]];
  }
  int get(int u){
    int ans = INT_MAX;
    for(int node = u; u; u = par[u])
      ans = min(ans,lvl[u]+lvl[node]-2*lvl[lca(u,node)]+(*
          mp[u].begin()).first);
    return ans;
  }
```

4.5 Heavy-Light decomposition

```
  #include <bits/stdc++.h>
  using namespace std;
  int a[500005],sz[500005],lvl[500005];
  int seg_id[500005],pos_id[500005],parent[500005];
  int up[500005][19],tin[500005],tout[500005],clk,CNT;
  vector<int> chain[500005];
  template <typename T>
  class SegmentTree{
    vector<T> segtree,lazy;
    public:
    SegmentTree(int size){
      segtree.resize(4*size,0);
      lazy.resize(4*size,0);
    }
    void update(int u, int a, int b, int i, int j, T x){
      if(lazy[u]){
        segtree[u]+=(b-a+1)*lazy[u];
        if(a!=b)
          lazy[u*2]+=lazy[u],lazy[2*u+1]+=lazy[u];
        lazy[u]=0;
      }
      if(j<a || i>b || a>b) return;
      if(j>=b && i<=a){
        segtree[u]+=(b-a+1)*x;
        if(a!=b) lazy[u*2]+=x,lazy[2*u+1]+=x;
        return;
      }
      update(u*2,a,(a+b)/2,i,j,x); update(u*2+1,(a+b)/2+1,
          b,i,j,x);
      segtree[u]=segtree[u*2]+segtree[u*2+1];
    }
    void update(int i, T x){
      update(1, 0, segtree.size()/4-1, i, i, x);
    }
    void update(int i, int j, T x){
      update(1, 0, segtree.size()/4-1, i, j, x);
    }
    T query(int u, int a, int b, int i, int j){
      if(j<a || i>b || a>b) return 0;
      if(lazy[u]){
        segtree[u]+=(b-a+1)*lazy[u];
        if(a!=b)
          lazy[u*2]+=lazy[u],lazy[2*u+1]+=lazy[u];
        lazy[u]=0;
      }
      if(j>=b && i<=a) return segtree[u];
      return query(u*2,a,(a+b)/2,i,j)+query(u*2+1,1+(a+b)
          /2,b,i,j);
    }
    T query(int i, int j){
      return query(1,0,segtree.size()/4-1,i,j);
    }
  };
  map<int,int> v[500005];
  int dfs(int u, int dad=1, int depth=1, int last_edge =
      0){
    tin[u]=clk++;
    up[u][0] = dad;
    lvl[u] = depth;
    a[u] = last_edge;
    for(int i=1;i<19;++i)
      up[u][i] = up[up[u][i-1]][i-1];
    lvl[u] = depth, sz[u] = 1;
    for(auto i:v[u]) if(i.first!=dad)
      sz[u] += dfs(i.first,u,depth+1, i.second);
    tout[u] = clk++;
    return sz[u];
  }
  void hld(int u, int dad = 1, int chain_no = 0, int
      chain_parent = 0){
    seg_id[u] = chain_no;
    pos_id[u] = chain[chain_no].size();
```

```
  parent[u] = chain_parent;
  chain[chain_no].push_back(u);
  int max_sz = 0, heavy_child = -1;
  for(auto i:v[u]) if(i.first!=dad && max_sz<sz[i.first])
    tie(max_sz,heavy_child)=make_pair(sz[i.first],i.first);
  if(heavy_child!=-1)
    hld(heavy_child, u, chain_no, chain_parent);
  for(auto i:v[u]) if(i.first!=dad && i.first!=heavy_child)
    hld(i.first,u,++CNT, u);
}
int lca(int u, int v){
  if(lvl[u]>lvl[v]) swap(u,v);
  if(tin[u]<=tin[v] && tout[v]<=tout[u]) return u;
  for(int i=19;i--;)
    if(!(tin[up[u][i]]<=tin[v]&&tout[v]<=tout[up[u][i]]))
      u = up[u][i];
  return up[u][0];
}
vector<SegmentTree> ST;
bool get(int x, int y){
  if(seg_id[x] == seg_id[y]){
    if(pos_id[x]>pos_id[y]) swap(x,y);
    return ST[seg_id[x]].query(pos_id[x],pos_id[y]);
  }
  if(seg_id[x]>seg_id[y]) swap(x,y);
  return get(x,parent[y]) ^ ST[seg_id[y]].query(0,pos_id[y]);
}
int u[100005],T,q,n;
vector<pair<pair<int,int>,int>> edges(1);
int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0); cout.tie(0);
  cin>>n;
  for(int i=1,x,y,p;i<n;++i){
    cin>>x>>y>>p;
    p%=2;
    v[x][y] = p;
    v[y][x] = p;
    edges.push_back({{x,y},p});
  }
  dfs(1); hld(1);
  ST.clear();
  for(int i=0;i<=CNT;++i){
    ST.push_back(SegmentTree(chain[i].size()));
    for(auto u:chain[i])
      ST[i].update(pos_id[u],a[u]);
  }
  for(cin>>q;q--;){
    int type,x,y;
    cin>>type>>x>>y;
    if(type==1){
      int l = lca(x,y);
      bool sum = get(x,l) ^ get(y,l);
      cout<<(sum?"WE NEED BLACK PANDA\n":"JAKANDA FOREVER\n");
    } else {
      if(lvl[edges[x].first.first]>lvl[edges[x].first.second])
        x = edges[x].first.first;
      else
        x = edges[x].first.second;
      y %= 2;
      ST[seg_id[x]].update(pos_id[x],y);
    }
  }
}
```

5 Data structures

5.1 Suffix array

```
#include <bits/stdc++.h>
using namespace std;
vector<int> suffix_array(string &s){
  int n = s.size();
  vector<int> sa(n), buckets(n);
  for(int i=0;i<n;++i) sa[i] = n-i-1;
  stable_sort(sa.begin(),sa.end(),[&](int i, int j){
      return s[i]<s[j];});
  for(int i=0;i<n;++i) buckets[i]=s[i];
  for(int len=1;len<n;len*=2){
    vector<int> b(buckets), cnt(n), s(sa);
    for(int i=0;i<n;++i)
      buckets[sa[i]]=i&&b[sa[i-1]]==b[sa[i]]&&sa[i-1]+len<n&&b[sa[i-1]+len/2]==b[sa[i]+len/2]?buckets[sa[i-1]]:i;
    iota(cnt.begin(), cnt.end(),0);
    for(int i=0;i<n;++i) if(s[i]>=len)
      sa[cnt[buckets[s[i]-len]]++]=s[i]-len;
  }
  return sa;
}
vector<int> kasai(string &s, vector<int> &sa){
  int n = s.size();
  vector<int> lcp(n),inv(n);
  for(int i=0;i<n;++i) inv[sa[i]] = i;
  for(int i=0,k=0;i<n;++i){
    if(k<0) k = 0;
    if(inv[i]==n-1){ k=0; continue; }
    for(int j=sa[inv[i]+1];max(i,j)+k<n&&s[i+k]==s[j+k];++k);
    lcp[inv[i]] = k--;
  }
  return lcp;
}
int main(){
  ios_base::sync_with_stdio(0);
  cin.tie(0);
  string a,s;
  int K = 0;
  for(;cin>>a;++K) s += a + char(4+K);
  vector<int> color(s.size()), col(s.size());
  for(int i=0,cnt=0;i<s.size();++i)
    col[i]=cnt, cnt+=s[i]<20;
  auto sa = suffix_array(s);
  auto lcp = kasai(s,sa);
```

```
  for(int i=0;i<lcp.size();++i) color[i]=col[sa[i]];
  int freq[11] = {};
  deque<int> v, mq;
  multiset<int> ms;
  int ans=0;
  for(int i=1,COL=0;i<lcp.size();++i){
    if(++freq[color[i]]==1) ++COL;
    while(v.size() && freq[color[v[0]]]>1){
      if(mq[0]==v[0])
        mq.pop_front();
      --freq[color[v[0]]];
      v.pop_front();
    }
    if(COL==K) ans = max(ans,lcp[mq[0]]);
    v.push_back(i);
    while(mq.size() && lcp[mq[0]]>lcp[i])
      mq.pop_front();
    mq.push_back(i);
  }
  cout<<ans<<'\n';
  return 0;
}
```

5.2 KD-tree

```
// A straightforward, but probably sub-optimal KD-tree implmentation
// that's probably good enough for most things (current it's a
// 2D-tree)
//
//  - constructs from n points in O(n lg^2 n) time
//  - handles nearest-neighbor query in O(lg n) if points are well
//    distributed
//  - worst case for nearest-neighbor may be linear in pathological
//    case
//
// Sonny Chan, Stanford University, April 2009

#include <iostream>
#include <vector>
#include <limits>
#include <cstdlib>
using namespace std;

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

bool operator==(const point &a, const point &b){
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b){
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b){
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b){
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox{
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry) {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < v.size(); ++i) {
            x0 = min(x0, v[i].x);   x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y);   y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)       return pdist2(point(x0, y0), p);
            else if (p.y > y1)  return pdist2(point(x0, y1), p);
            else                return pdist2(point(x0, p.y), p);
        }
        else if (p.x > x1) {
            if (p.y < y0)       return pdist2(point(x1, y0), p);
            else if (p.y > y1)  return pdist2(point(x1, y1), p);
            else                return pdist2(point(x1, p.y), p);
        }
        else {
            if (p.y < y0)       return pdist2(point(p.x, y0), p);
            else if (p.y > y1)  return pdist2(point(p.x, y1), p);
            else                return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or leaf
struct kdnode {
    bool leaf;      // true if this is a leaf node (has one point)
```

```
    point pt;       // the single point of this is a leaf
    bbox bound;     // bounding box for set of points in children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete second; }

    // intersect a point with this node (returns squared distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of points
    void construct(vector<point> &vp){
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
            pt = vp[0];
        }
        else {
            // split on x if the bbox is wider than high (not best heuristic...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode();   first->construct(vl);
            second = new kdnode();  second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree{
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to nearest point
    ntype search(kdnode *node, const point &p)
    {
        if (node->leaf) {
            // commented special case tells a point not to find itself
//          if (p == node->pt) return sentry;
//          else
                return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to search first
        // (note that the other side is also searched if needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        }
        else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};

// some basic test code here

int main(){
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i) {
        vp.push_back(point(rand()%100000, rand()%100000));
    }
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x
             << ", " << q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }
}
```

5.3 Merge Sort Tree

```
#include <bits/stdc++.h>
using namespace std;
int A[1000005];
vector<int> segtree[400005];
void build(int u, int a, int b){
        if(a==b){
                segtree[u].push_back(A[a]);
                return;
        }
        build(u*2,a,(a+b)/2); build(u*2+1,(a+b)/2+1,b);
        segtree[u].resize(b-a+1);
        merge(segtree[u*2].begin(),segtree[u*2].end(),
           segtree[u*2+1].begin(),segtree[u*2+1].end(),
           segtree[u].begin());
}
int query(int u, int a, int b, int i, int j, int k){
        if(b<a || j<a || i>b) return 0;
        if(i<=a && b<=j) return lower_bound(segtree[u].
           begin(),segtree[u].end(),k) - segtree[u].
           begin();
        return query(u*2,a,(a+b)/2,i,j,k)+query(u*2+1,(a
           +b)/2+1,b,i,j,k);
}
int main(){
        ios_base::sync_with_stdio(0);
        int n,q,low,high,mid,x,y,k;
        for(cin>>n>>q,x=0;x<n;cin>>A[x++]);
        for(build(1,0,n-1);q--;cout<<low-1<<'\n')
                for(cin>>x>>y>>k,low=-1e9,high=1e9;low<
                   high;)
                        if(query(1,0,n-1,x-1,y-1,mid=low
                           +high>>1)<k) low=mid+1;
                        else high=mid;
}
```

6 Miscellaneous

6.1 Miller-Rabin Primality Test

```
//  Error rate: 2^(-TRIAL)
//  Almost constant time. srand is needed

int64_t ModMul(int64_t a, int64_t b, int64_t m){
  int64_t ret=0, c=a;
  for(;b;b>>=1, c=(c+c)%m)
    if(b&1) ret=(ret+c)%m;
  return ret;
}
int64_t ModExp(int64_t a, int64_t n, int64_t m){
  return n?ModMul(ModExp(ModMul(a,a,m),n/2,m),(n%2?a:1),
     m):1;
}
bool Witness(int64_t a, int64_t n){
  int64_t u=n-1;
    int t=0;
  while(!(u&1)){u>>=1; t++;}
  int64_t x0=ModExp(a, u, n), x1;
  for(int i=1;i<=t;i++) {
    x1=ModMul(x0, x0, n);
    if(x1==1&&x0!=1&&x0!=n-1) return true;
    x0=x1;
  }
  if(x0!=1) return true;
  return false;
}

bool IsPrimeFast(int64_t n, int TRIAL=15){
  if(n<=2) return (n==2);
  static random_device rd;
  static mt19937_64 g(rd());
  while(TRIAL--)
    if(Witness(g()/2%(n-2)+1, n))
      return false;
  return true;
}
```

6.2 Pollard-Rho factorization

```
typedef long long unsigned int llui;
typedef long long int lli;
typedef long double float64;

llui mul_mod(llui a, llui b, llui m){
  llui y = (llui)((float64)a*(float64)b/m+(float64)1/2);
  y = y * m;
  llui x = a * b;
  llui r = x - y;
  if ( (lli)r < 0 ){
    r = r + m; y = y - 1;
  }
  return r;
}

llui C,a,b;
llui gcd(){
  llui c;
  if(a>b){
    c = a; a = b; b = c;
  }
  while(1){
    if(a == 1LL) return 1LL;
    if(a == 0 || a == b) return b;
    c = a; a = b%a;
    b = c;
  }
}

llui f(llui a, llui b){
  llui tmp;
  tmp = mul_mod(a,a,b);
  tmp+=C; tmp%=b;
  return tmp;
}

llui pollard(llui n){
  if(!(n&1)) return 2;
  C=0;
  llui iteracoes = 0;
  while(iteracoes <= 1000){
    llui x,y,d;
    x = y = 2; d = 1;
    while(d == 1){
```

```
      x = f(x,n);
      y = f(f(y,n),n);
      llui m = (x>y)?(x-y):(y-x);
      a = m; b = n; d = gcd();
    }
    if(d != n)
      return d;
    iteracoes++; C = rand();
  }
  return 1;
}
llui pot(llui a, llui b, llui c){
  if(b == 0) return 1;
  if(b == 1) return a%c;
  llui resp = pot(a,b>>1,c);
  resp = mul_mod(resp,resp,c);
  if(b&1)
    resp = mul_mod(resp,a,c);
  return resp;
}
// Rabin-Miller primality testing algorithm
bool isPrime(llui n){
  llui d = n-1;
  llui s = 0;
  if(n <=3 || n == 5) return true;
  if(!(n&1)) return false;
  while(!(d&1)){ s++; d>>=1; }
  for(llui i = 0;i<32;i++){
    llui a = rand();
    a <<=32;
    a+=rand();
    a%=(n-3); a+=2;
    llui x = pot(a,d,n);
    if(x == 1 || x == n-1) continue;
    for(llui j = 1;j<= s-1;j++){
      x = mul_mod(x,x,n);
      if(x == 1) return false;
      if(x == n-1)break;
    }
    if(x != n-1) return false;
  }
  return true;
}
map<llui,int> factors;
// Precondition: factors is an empty map, n is a
    positive integer
// Postcondition: factors[p] is the exponent of p in
    prime factorization of n
void fact(llui n){
  if(!isPrime(n)){
    llui fac = pollard(n);
    fact(n/fac); fact(fac);
  }else{
    map<llui,int>::iterator it;
    it = factors.find(n);
    if(it != factors.end()){
      (*it).second++;
    }else{
      factors[n] = 1;
    }
  }
}
```

6.3 Manachers algorithm

```
// Maximal palindrome lengths centered around each
// position in a string (including positions between
    characters) and returns
// them in left-to-right order of centres. Linear time.
// Ex: "opposes" -> [0, 1, 0, 1, 4, 1, 0, 1, 0, 1, 0, 3,
     0, 1, 0]
vector<int> fastLongestPalindromes(string str) {
  int i=0,j,d,s,e,lLen,palLen=0;
  vector<int> res;
  while (i < str.length()) {
    if (i > palLen && str[i-palLen-1] == str[i]) {
      palLen += 2; i++; continue;
    }
    res.push_back(palLen);
    s = res.size()-2;
    e = s-palLen;
    bool b = true;
    for (j=s; j>e; j--) {
      d = j-e-1;
      if (res[j] == d) { palLen = d; b = false; break; }
      res.push_back(min(d, res[j]));
    }
    if (b) { palLen = 1; i++; }
  }
  res.push_back(palLen);
  lLen = res.size();
  s = lLen-2;
  e = s-(2*str.length()+1-lLen);
  for (i=s; i>e; i--) { d = i-e-1; res.push_back(min(d,
      res[i])); }
  return res;
}
```

6.4 Convex Hull Trick

```
struct Line {
  long long m, b;
  mutable function<const Line*()> succ;
  bool operator<(const Line& rhs) const{
    if (rhs.b != -(1ll<<62)) return m > rhs.m; // < for
        max
    const Line* s = succ();
    if (!s) return 0;
    return b-s->b > (s->m -m)*rhs.m; // < for max
  }
};
struct HullDynamic : public multiset<Line> {
  bool bad(iterator y) {
    auto z = next(y);
    if(y==begin()){
      if(z==end())return 0;
      return y->m == z->m && y->b >= z->b; // <= for max
    }
    auto x = prev(y);
    if (z == end()) return y->m == x->m && y->b >= x->b;
        // <= for max
```

```
      return (x->b - y->b)*1.0*(z->m - y->m) >= (y->b - z
          ->b)*1.0*(y->m - x->m);
    }
    void insert_line(long long m, long long b) {
      auto y = insert({ m, b });
      y->succ = [=] { return next(y) == end() ? 0 : &*next
          (y); };
      if (bad(y)) { erase(y); return; }
      while (next(y) != end() && bad(next(y))) erase(next(
          y));
      while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    long long eval(long long x) {
      auto l = *lower_bound((Line){x,-(1ll<<62)});
      return l.m * x + l.b;
    }
};
```

6.5 Dynamic Programming(DnC)

```
long long dp[21][100005];
void cost(int x, int y);
void computeDP(int idx,int jleft,int jright,int kleft,
    int kright){
  if(jleft>jright) return;
  int jmid=(jleft+jright)/2;
  int bestk=jmid;
  for(int k=kleft;k<=min(kright,jmid);++k){
    cost(k,jmid);
    if(dp[idx-1][k-1]+tot<dp[idx][jmid])
      dp[idx][jmid]=dp[idx-1][k-1]+tot,bestk=k;
  }
  computeDP(idx,jleft,jmid-1,kleft,bestk);
  computeDP(idx,jmid+1,jright,bestk,kright);
}
int main(){
  for(int i=0;i<=k;++i)
    for(int j=0;j<=n;dp[i][j++]=1e17);
  dp[0][0]=0;
  for(int i=1;i<=k;++i)
    computeDP(i,1,n,1,n);
  cout<<dp[k][n];
}
```

6.6 Longest increasing subsequence

```
// 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

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] = it == best.begin() ? -1 : prev(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;
}
```

6.7 Dates

```
// 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];
}
```

6.8 Knuth-Morris-Pratt

```
typedef vector<int> VI;

void buildPi(string& p, VI& pi){
  pi = VI(p.length());
  int k = -2;
  for(int i = 0; i < p.length(); i++) {
    while(k >= -1 && p[k+1] != p[i])
      k = (k == -1) ? -2 : pi[k];
    pi[i] = ++k;
  }
}
int KMP(string& t, string& p){
  VI pi;
  buildPi(p, pi);
  int k = -1;
  for(int i = 0; i < t.length(); i++) {
    while(k >= -1 && p[k+1] != t[i])
      k = (k == -1) ? -2 : pi[k];
    k++;
    if(k == p.length() - 1) {
      // p matches t[i-m+1, ..., i]
      cout << "matched at index " << i-k << ": ";
      cout << t.substr(i-k, p.length()) << endl;
      k = (k == -1) ? -2 : pi[k];
    }
  }
}

int main(){
  KMP("AABAACAADAABAABA", "AABA"); //Mtches at: 0,9,12
}
```

6.9 2-SAT

```
struct TwoSat {
  int n;
  vector<vector<int> > adj, radj, scc;
  vector<int> sid, vis, val;
  stack<int> stk;
  int scnt;
  // n: number of variables, including negations
  TwoSat(int n): n(n), adj(n), radj(n), sid(n), vis(n),
      val(n, -1) {}
  // adds an implication
  void impl(int x, int y) { adj[x].push_back(y); radj[y
      ].push_back(x); }
  // adds a disjunction
  void vee(int x, int y) { impl(x^1, y); impl(y^1, x); }
  // forces variables to be equal
  void eq(int x, int y) { impl(x, y); impl(y, x); impl(x
      ^1, y^1); impl(y^1, x^1); }
  // forces variable to be true
  void tru(int x) { impl(x^1, x); }

  void dfs1(int x) {
    if (vis[x]++) return;
    for (int i = 0; i < adj[x].size(); i++)
      dfs1(adj[x][i]);
    stk.push(x);
  }
  void dfs2(int x) {
    if (!vis[x]) return; vis[x] = 0;
    sid[x] = scnt; scc.back().push_back(x);
    for (int i = 0; i < radj[x].size(); i++)
      dfs2(radj[x][i]);
  }

  // returns true if satisfiable, false otherwise
  // on completion, val[x] is the assigned value of
      variable x (Note: val[x] = 0 implies val[x^1] = 1)
  bool two_sat() {
    scnt = 0;
    for (int i = 0; i < n; dfs1(i++));
    while (!stk.empty()) {
      int v = stk.top(); stk.pop();
      if (vis[v]) {
        scc.push_back(vector<int>());
        dfs2(v);
        scnt++;
      }
    }
    for (int i = 0; i < n; i += 2)
      if (sid[i] == sid[i+1]) return false;
    vector<int> must(scnt);
    for (int i = 0; i < scnt; i++)
      for (int j = 0; j < scc[i].size(); j++){
        val[scc[i][j]] = must[i];
        must[sid[scc[i][j]^1]] = !must[i];
      }
    return true;
  }
};
```