Table of Contents

Scripts

run

```
#!/bin/bash
clear; clear
g++ $1 -g -std=c++0x -Wall -Wconversion -Wshadow -o sol || exit

for i in *.in; do
    echo --- $i
    ./sol < $i > o && (diff -y o ${i%.in}.[ao]?? > t || cat t) || cat o
done
```

sol

```
#!/bin/bash
mkdir $1
ln -st $1 ../run
cp acm.cc $1/$1.cc
```

vimrc

```
nnoremap j gj
nnoremap k gk
nnoremap B ^
```

```
nnoremap E $
nnoremap ^ <nop>
nnoremap $ <nop>
set nowrap
set autoindent
set expandtab
set tabstop=2
set softtabstop=2
set shiftwidth=2
set nojoinspaces
set number
```

Data structure

ClosestPair

```
typedef long long llong;
const int maxn = 200010;
const double INF = 4e18;

struct Point {
    double x, y;
};

class closestPair {
public:
  vector<Point> s;
  vector<Point> tmp;
  int size;

  void init(vector<Point>& points) {
    size = (int)points.size();
    s = vector<Point>(points);
    tmp.resize(size);
    auto cmpx = [](Point a, Point b) { return a.x < b.x; };
    sort(s.begin(), s.end(), cmpx);
  }

  bool cmpY(Point a, Point b) { return a.y < b.y; }

  void merge(int l, int m, int r) {
    int tm = m, tl = l, idx = 0;
    while (l < tm  && m < r) {
        if (s[l].y <= s[m].y) tmp[idx++] = s[l++];
        else tmp[idx++] = s[m++];
    }
    while (l < tm) tmp[idx++] = s[l++];
    while (m < r) tmp[idx++] = s[m++];
    for (int i=0; i<idx; i++) s[i+tl] = tmp[i];
  }

  double calc(int l, int r) {
    double d = INF;
    if (r == l) return d;
    int mid = (l + r) >> 1;
    double X = s[mid].x; // store X before recursive process, cuz they will
rearrange Points.
    d = min(calc(l, mid), calc(mid+1, r));
    merge(l, mid+1, r+1);

    int idx = 0;
    for (int i=l; i<=r; i++) {
```

```
        if (fabs(s[i].x - X) >= d) continue;
        for (int j=0; j<idx; j++) {
            double dx = s[i].x - tmp[idx - 1 - j].x;
            double dy = s[i].y - tmp[idx - 1 - j].y;
            if (dy >= d) break;
            d = min(d, sqrt(dx*dx + dy*dy));
        }
        tmp[idx++] = s[i];
    }
    return d;
  }

  double run() { return calc(0, size-1); }
} sol;

int main() {
    int n;
    scanf("%d", &n);
    vector<Point> ps(n);
    for (int i=0; i<n; i++) {
        double x, y;
        scanf("%lf%lf", &x, &y);
        ps[i] = Point{x, y};
    }
    sol.init(ps);
    printf("%lf\n",sol.run());
    return 0;
}
```

FenwickTree

```
#define N 1000010
#define lowb(x) (x & (-x))
int a[N], n, m;

class FenwickTree {
    private:
        int t[N];

    public:
        void add(int p, int v) {
            while (p <= n) {
                t[p] += v;
                p += lowb(p);
            }
        }

        int prefix(int p) {
            int ans = 0;
            while (p) {
                ans += t[p];
                p -= lowb(p);
            }
            return ans;
        }
```

```
        int range(int l, int r) {return prefix(r) - prefix(l-1);}
}t;

int main() {
    scanf("%d", &n);
    for (int i=1; i<=n; i++) scanf("%d", a+i);
    for (int i=1; i<=n; i++) t.add(i, a[i]);

    scanf("%d", &m);
    for (int i=1; i<=m; i++) {
        int o, x, y;
        scanf("%d%d%d", &o, &x, &y);
        if (o == 1) {
            printf("%d\n", i);
            t.add(x, y);
        } else {
            int ans = t.range(x, y);
            printf("%d %d\n", i, ans);
        }
    }
    return 0;
}
```

HeavyLightDecomposition

```
#define maxn 10010

struct Edge { int u, v, c; } es[maxn];

class SegmentTree {
// need test on https://www.codechef.com/problems/QTREE
    private:
        int t[maxn<<1], size, height;
    public:

        void init(int N) {
            // height of range n segment tree
            height = sizeof(int) * 8 - __builtin_clz(N);
            size = N;
            memset(t, 0, sizeof(t));
        }

        void modify(int p, int val) {
            for ( t[p += size] = val; p > 1; p >>= 1) {
                t[p >> 1] = max(t[p], t[p^1]);
            }
        }

        int query(int l, int r) {// [l, r)
            int ans = 0;
            for (l += size, r += size; l < r; l >>= 1, r >>= 1) {
                if (l & 1) ans = max(ans, t[l++]);
                if (r & 1) ans = max(ans, t[--r]);
            }
            return ans;
        }
};

class HLD {
    #define INF 1000000
    public:
        SegmentTree t;
```

```
// heavy light decomposition
int dep[maxn], heavy[maxn], fa[maxn], sz[maxn],
    top[maxn], pos[maxn], pCnt, n;
// tree
vector<int> g[maxn];

void init(int N, Edge (&es)[maxn]) {
    pCnt = 0;
    n = N;
    for (int i=1; i<=n; i++) g[i].clear();
    t.init(n);
    memset(dep, 0, sizeof(dep));
    memset(heavy, 0, sizeof(heavy));
    for (int i=1; i<n; i++) {
        g[es[i].u].push_back(es[i].v);
        g[es[i].v].push_back(es[i].u);
    }
    int root = 1;
    dfs1(root);
    dfs2(root, root);
    for (int i=1; i<n; i++) {
        int& u = es[i].u;
        int& v = es[i].v;
        if (dep[u] > dep[v]) swap(u, v);
        t.modify(pos[v], es[i].c);
    }
}

void dfs1(int rt) {
    sz[rt] = 1;
    for (auto v: g[rt]) if (v != fa[rt]) {
        fa[v] = rt;
        dep[v] = dep[rt] + 1;
        dfs1(v);
        sz[rt] += sz[v];
        if (sz[v] > sz[heavy[rt]]) heavy[rt] = v;
    }
}

void dfs2(int rt, int head) {
    pos[rt] = pCnt++;
    top[rt] = head;
    if (heavy[rt]) dfs2(heavy[rt], head);
    for (auto v: g[rt]) {
        if (v != fa[rt] && v != heavy[rt]) dfs2(v, v);
    }
}

void changeEdgeCost(int eid, int c) {
    t.modify(pos[es[eid].v], c);
}

int maxLenEdge(int u, int v) {
    int ans = -INF;
    while (top[u] != top[v]) {
        if (dep[top[u]] > dep[top[v]]) swap(u, v);
        ans = max(ans, t.query(pos[top[v]], pos[v]+1));
        v = fa[top[v]];
    }
    if (u == v) return ans;
    if (dep[u] > dep[v]) swap(u, v);
    ans = max(ans, t.query(pos[heavy[u]], pos[v]+1));
```

```
            return ans;
        }
} hld;

int main() {
    int cas;
//    freopen("test.in", "r", stdin);
    for (scanf("%d", &cas); cas; cas--) {
        int n;
        scanf("%d", &n);
        for (int i=1; i<n; i++) {
            int a, b, c;
            scanf("%d%d%d", &a, &b, &c);
            es[i].u = a, es[i].v = b, es[i].c = c;
        }
        hld.init(n, es);
        char op[10];
        int x, y;
        while (scanf("%s", op) && *op != 'D') {
            scanf("%d%d", &x, &y);
            if (*op == 'C')  hld.changeEdgeCost(x, y);
            else printf("%d\n", hld.maxLenEdge(x, y));
        }
    }
    return 0;
}
```

SegmentTree

```
#define maxn 100000
class SegmentTree {
    private:

        int t[maxn<<1], size, height;

        void push_up(int l, int r) {// [l, r)
            for (l += size, r += size-1; l > 1;) {
                l >>= 1, r >>= 1;
                for (int i=l; i<=r; i++) t[i] = max(t[i<<1], t[i<<1|1]);
            }
        }

        void push_down(int l, int r) {// [l, r)
            int s = height;
            for (l += size, r += size-1; s > 0; --s) {
                for (int i = (l>>s); i <= (r>>s); i++) {
                    // update
                }
            }
        }

    public:
        SegmentTree(int N) { init(N); }
        void init(int N) {
            size = N;
            // height of range n segment tree
            height = sizeof(int) * 8 - __builtin_clz(N);
            memset(t, 0, sizeof(t));
        }

        void modify(int p, int val) {
            for ( t[p += size] = val; p > 1; p >>= 1) {
```

```
                t[p >> 1] = max(t[p], t[p^1]);
            }
        }

        int query(int l, int r) {// [l, r)
            int ans = 0;
            for (l += size, r += size; l < r; l >>= 1, r >>= 1) {
                if (l & 1) ans = max(ans, t[l++]);
                if (r & 1) ans = max(ans, t[--r]);
            }
            return ans;
        }
};
```

Geometry

header

```
const double EPS = 1e-10;

struct P {
    double x, y;
    P() {};
    P(double a, double b) : x(a), y(b) {}

    double add(double a, double b) { // add operation consider eps
        if (fabs(a + b) < EPS * (abs(a) + abs(b))) return 0;
        return a + b;
    }
    P operator + (P p) { return P(add(x, p.x), add(y, p.y)); }
    P operator - (P p) { return P(add(x, -p.x), add(y, -p.y)); }
    P operator * (double d) { return P(x * d, y * d); }
    double dot(P p) { return add(x * p.x, y * p.y); }
    double det(P p) { return add(x * p.y, -y * p.x); }
};

bool on_seg(P p1, P p2, P q) { // detect whether q is on segment p1-p2
    return (p1 - q).det(p2 - q) == 0 && (p1 - q).dot(p2 - q) <= 0;
}

P intersection(P p1, P p2, P q1, P q2) {
    return p1 + (p2 - p1) * ((q2 - q1).det(q1 - p1) / (q2 - q1).det(p2 -
p1));
}

bool is_intersect(P p1, P p2, P q1, P q2) {
  return (p1 - p2).det(q1 - q2) != 0;
}
class ConvexHull {
  public:

  static bool cmp_x(const P& p, const P& q) {
    if (p.x != q.x) return p.x < q.x;
    return p.y < q.y;
  }

  vector<P> convex_hull(vector<P> ps, int n) {
    sort(ps.begin(), ps.end(), cmp_x);
    int k = 0;
    vector<P> qs(n*2);
    // construct upside
```

```
    for (int i=0; i<n; i++) {
      while (k > 1 && (qs[k-1] - qs[k-2]).det(ps[i] - qs[k-1]) <= 0) k--;
      qs[k++] = ps[i];
    }

    // custruct downside
    for (int i=n-2, t=k; i>=0; i--) {
      while (k > t && (qs[k-1] - qs[k-2]).det(ps[i] - qs[k-1]) <= 0) k--;
      qs[k++] = ps[i];
    }
    qs.resize(k-1);
    return qs;
  }
};
```

Graph

Dinic

```
#define MAX_V 1010
#define INF 1e8

class Dinic {
  public:

  struct edge { int to, cap, rev; };
  vector<edge> G[MAX_V];
  int level[MAX_V]; // distance to source
  int iter[MAX_V]; // current edge

  void add_edge(int from, int to, int cap) {
    G[from].push_back((edge){to, cap, (int)G[to].size()});
    G[to].push_back((edge){from, 0, (int)G[from].size()-1});
  }

  void bfs(int s) {
    memset(level, -1, sizeof(level));
    queue<int> que;
    level[s] = 0;
    que.push(s);
    while (!que.empty()) {
      int v = que.front(); que.pop();
      for (int i=0; i<G[v].size(); i++) {
        edge &e = G[v][i];
        if (e.cap > 0 && level[e.to] < 0) {
          level[e.to] = level[v] + 1;
          que.push(e.to);
        }
      }
    }
  }

  int dfs(int v, int t, int f) {
    if (v == t) return f;
    for (int& i = iter[v]; i<G[v].size(); i++) {
      edge &e = G[v][i];
      if (e.cap > 0 && level[v] < level[e.to]) {
        int d = dfs(e.to, t, min(f, e.cap));
        if (d > 0) {
          e.cap -= d;
          G[e.to][e.rev].cap += d;
          return d;
```

```
        }
      }
    }
    return 0;
  }

  int max_flow(int s, int t) {
    int flow = 0;
    for (;;) {
      bfs(s);
      if (level[t] < 0) return flow;
      memset(iter, 0, sizeof(iter));
      int f;
      while ((f = dfs(s, t, INF)) > 0) flow += f;
    }
  }
};
```

MinCostMaxFlow

```
using namespace std;
#define MAX_V 100010
#define INF 1e8
typedef pair<int, int> pii;

class MinCostMaxFlow {
  struct edge {int to, cap, cost, rev; };
  vector<edge> G[MAX_V];
  int V;
  int h[MAX_V];
  int dist[MAX_V];
  int prevv[MAX_V], preve[MAX_V];

  void add_edge(int from, int to, int cap, int cost) {
    G[from].push_back((edge){to, cap, cost, (int)G[to].size()});
    G[to].push_back((edge){from, 0, -cost, (int)G[from].size() - 1});
  }

  int min_cost_flow(int s, int t, int f) {
    int res = 0;
    fill(h, h + V, 0);
    while (f > 0) {
      priority_queue<pii, vector<pii>, greater<pii> > que;
      fill(dist, dist + V, INF);
      dist[s] = 0;
      que.push(pii(0, s));
      while (!que.empty()) {
        pii p = que.top(); que.pop();
        int v = p.second;
        if (dist[v] < p.first) continue;
        for (int i = 0; i < G[v].size(); i++) {
          edge &e = G[v][i];
          if (e.cap > 0 &&
              dist[e.to] > dist[v] + e.cost + h[v] - h[e.to]) {
            dist[e.to] = dist[v] + e.cost + h[v] - h[e.to];
            prevv[e.to] = v;
            preve[e.to] = i;
            que.push({dist[e.to], e.to});
          }
        }
      }
      if (dist[t] == INF) return -1;
```

```
      for (int v=0; v<V; v++) h[v] += dist[v];
      int d = f;
      for (int v=t; v!=s; v=prevv[v]) {
        d = min(d, G[prevv[v]][preve[v]].cap);
      }
      f -= d;
      res += d * h[t];
      for (int v=t; v!=s; v=prevv[v]) {
        edge &e = G[prevv[v]][preve[v]];
        e.cap -=d;
        G[v][e.rev].cap += d;
      }
    }
    return res;
  }
};

// remember reset residual network after compute max flow
void solve() {
  /*
   * int flow = max_flow(s, t);
   * int cost = min_cost_flow(s, t, flow);
   */
}
```

PushRelabel

```
using namespace std;
typedef long long ll;
#define N 10010
#define M 2000200
const ll INF = 1e10;

class MaxFlow {
    public:

    struct Edge {
        int to, c, f, rev;
    } e[M << 1];

    vector<int> g[N];
    queue<int> q;
    ll ex[N];
    bool inq[N];
    int n, m, s, t, tote, h[N];
    int cnth[N<<1];

    void initialize(int n, int m, int s, int t) {
        this->n = n; this->m = m;
        this->s = s; this->t = t; tote = 0;
        for (int i=0; i<n; i++) ex[i] = 0;
        for (int i=0; i<n; i++) h[i] = 0;
        for (int i=0; i<n; i++) g[i].clear();
        for (int i=0; i<=2*n; i++) cnth[i] = 0;
        for (int i=0; i<n; i++) inq[i] = 0;
        while (!q.empty()) q.pop();
    }

    void add_edge(int from, int to, int cap) {
        g[from].push_back(tote);
        e[tote] = Edge{to, cap, 0, tote^1};
        tote++;
```

```
}

void push(int from, Edge& edge) {
    int flow = min(ex[from], (ll)(edge.c - edge.f));
    ex[from] -= flow;
    ex[edge.to] += flow;
    edge.f += flow;
    e[edge.rev].f -= flow;
}

void gap_heuristic(int gap) {
    for (int u=0; u<n; u++) {
        if (u == s || u == t || h[u] < gap) continue;
        cnth[h[u]]--;
        h[u] = 2 * n;
        cnth[h[u]]++;
    }
}

void discharge(int u) {
    while (ex[u]) {
        int nexth = 2 * n;
        for (auto eid: g[u]) {
            if (e[eid].c - e[eid].f == 0) continue;
            int v = e[eid].to;
            if (h[u] == h[v] + 1) {
                push(u, e[eid]);
                if (v != s && v != t && !inq[v]) {
                    inq[v] = true;
                    q.push(v);
                }
            }
            if (h[u] < h[v] + 1) nexth = min(nexth, h[v] + 1);
            if (!ex[u]) break;
        }
        if (ex[u]) {
            if (cnth[h[u]] == 1) gap_heuristic(h[u]);
            else {
                cnth[h[u]] --;
                h[u] = nexth;
                cnth[h[u]] ++;
            }
        }
        if (h[u] >= n) break;
    }
    inq[u] = false;
}

ll run() {
    h[s] = n;
    cnth[0] = n - 1;
    cnth[n] = 1;
    for (auto eid: g[s]) {
        ex[s] = INF;
        if (!e[eid].c) continue;
        push(s, e[eid]);
        if (e[eid].to != t && !inq[e[eid].to]) {
            inq[e[eid].to] = true;
            q.push(e[eid].to);
        }
    }
```

```
        while (!q.empty()) {
            int u = q.front(); q.pop();
            discharge(u);
        }
        return ex[t];
    }
} mf;

int main() {
//    freopen("test.in", "r", stdin);
    int n, m, s, t;
    scanf("%d%d%d%d", &n, &m, &s, &t);

    mf.initialize(n, m, s-1, t-1);
    for (int i=0; i<m; i++) {
        int u, v, c;
        scanf("%d%d%d", &u, &v, &c);
        mf.add_edge(u-1, v-1, c);
        mf.add_edge(v-1, u-1, 0);
    }
    cout << mf.run() << endl;
    return 0;
}
```

Tarjan

```
#define MAX 10000
stack<int> S;
int disc = 0;
int in_stack[MAX];
void tarjan(int u,vector< vector<int> > g,int index[],int low_index[]){
  index[u] = low_index[u] = ++disc;
  in_stack[u] = 1;
  S.push(u);
  for(int i = 0;i < g[u].size();i++) {
  int v = g[u][i];
  if(index[v] == -1) {
      tarjan(v,g,index,low_index);
      low_index[u] = min(low_index[v],low_index[u]);
    }
    else if(in_stack[v] == 1)
      low_index[u] = min(low_index[u],index[v]);
  }
  if(index[u] == low_index[u]) {
    while(S.empty() == false && index[u] == low_index[S.top()] ) {
      cout<<S.top()<<" ";
      in_stack[S.top()] = 0;
      S.pop();
    }
    cout<<endl;
      }
}
```

ToplogicalSort

```
#define N 200010
int n, m, ind[N], vis[N], stk[N], top;
vector<int> g[N], ans;
priority_queue<int, vector<int>, greater<int> > q;

bool dfs0(int p, int flag) {
    vis[p] = flag;
    for (auto i: g[p]) {
```

```
        if (!vis[i]) {
            bool ans = dfs0(i, flag);
            if (ans) return ans;
        }
        else {
            if (vis[i] == flag) return true;
            else return false;
        }
    }
    return false;
}

bool checkCyc() {
    int flag = 0;
    for (int i=0; i<n; i++) if (!vis[i]) {
        if (dfs0(i, ++flag)) return true;
    }
    return false;
}

void sol1() {
    for (int i=0; i<n; i++) if (!ind[i]) q.push(i);
    while (!q.empty()) {
        int cur = q.top();
        q.pop();
        ans.push_back(cur);
        for (auto i: g[cur]) {
            ind[i]--;
            if (!ind[i]) {
                q.push(i);
            }
        }
    }
    if ((int)ans.size() != n) {
        assert(checkCyc() == true);
        printf("-1\n");
    }
    else {
        assert((int)ans.size() == n);
        for (int i=0; i<n; i++) printf("%d%c", ans[i], i==n-1?'\n': ' ');
    }
}

void dfs(int p) {
    vis[p] = 1;
    for (auto i: g[p]) {
        if (!vis[i]) dfs(i);
    }
    stk[top++] = p;
}

bool cmp(int a, int b) { return a > b; }

int idx[N];
bool checkValid() {
    for (int i=top-1; i>=0; i--) idx[stk[i]] = top-1-i;
    for (int i=0; i<n; i++) {
        for (auto j: g[i]) if (idx[j] <= idx[i]) return false;
    }
    return true;
}
```

```
void sol2() {
    memset(vis, 0, sizeof(vis));
    for (int i=0; i<n; i++) sort(g[i].begin(), g[i].end(), cmp);
    for (int i=n-1; i>=0; i--) if (!vis[i]) {
        dfs(i);
    }
    if(checkValid()) {
        for (int i=top-1; i>=0; i--) printf("%d%c", stk[i], i==0?'\n':' ');
    } else printf("-1\n");
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i=0; i<m; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        g[u].push_back(v);
        ind[v]++;
    }
    sol1();
}
```

BipartiteMatch

```
#define N 1010
typedef vector<int> vi;
vi e[N];
int match[N], n;
bool vis[N];

int dfs(int cur) {
  vis[cur] = true;
  for (int i: e[cur]) if (match[i] == -1 || (!vis[match[i]] &&
dfs(match[i]))) {
    match[cur] = i;
    match[i] = cur;
    return true;
  }
  return false;
}

int hungary() {
  memset(match, -1, sizeof(match));
  int res = 0;
  for (int i=0; i<n; i++) if (match[i] == -1) {
    memset(vis, false, sizeof(vis));
    res += dfs(i);
  }
  return res;
}
```

Math

Math

```
#define MAX_P 100010
typedef pair<int, int> pii;
typedef vector<int> vi;

// solve a*x + b*y = gcd(a, b), if a < 0,
// transform to |a|*(-x) + b*y = gcd(|a|, b)
int extgcd(int a, int b, int& x, int& y) {
  int d = a;
```

```
  if (b != 0) {
    d = extgcd(b, a%b, y, x);
    y -= (a/b) * x;
  } else {
    x = 1; y = 0;
  }
  return d;
}

int mod_inverse(int a, int m) {
  int x, y;
  extgcd(a, m, x, y);
  return (m + x % m) % m;
}

// it's CRT when A = {1,...1}
pii linear_congruence(const vi& A, const vi& B, const vi& M) {
  int x = 0, m = 1;
  for (int i=0; i<(int)A.size(); i++) {
    int a = A[i] * m, b = B[i] - A[i] * x, d = __gcd(M[i], a);
    if (b % d != 0) return {0, -1}; // no solution
    int t = b / d * mod_inverse(a / d, M[i] / d) % (M[i] / d);
    x = x + m * t;
    m *= M[i] / d;
  }
  return {x % m, m};
}

int fact[MAX_P];
// factorize n! => a * p^e, return a % p. O(log_p(n))
int mod_fact(int n, int p, int& e) {
  e = 0;
  if (n == 0) return 1;
  // calculate p, 2p, 3p, ...
  int res = mod_fact(n / p, p , e);
  e += n / p;

  // (p-1)! = -1 % mod p
  if (n / p % 2 != 0) return res * (p - fact[n % p]) % p;
  return res * fact[n % p] % p;
}

int mod_comb(int n, int k, int p) {
  if (n < 0 || k < 0 || n < k) return 0;
  int e1, e2, e3;
  int a1 = mod_fact(n, p, e1);
  int a2 = mod_fact(k, p, e2);
  int a3 = mod_fact(n-k, p, e3);
  if (e1 > e2 + e3) return 0;
  return a1 * mod_inverse(a2 * a3 % p, p) % p;
}
```

Matrix

```
const double EPS = 1E-8;
typedef vector<double> vec;
typedef vector<vec> mat;

// solve Ax = b, A is a n*n matrix
vec gauss_jordan(const mat& A, const vec& b) {
  int n = A.size();
  mat B(n, vec(n+1));
```

```
  for (int i=0; i<n; i++)
    for (int j=0; j<n; j++) B[i][j] = A[i][j];
  // augmented matrices
  for (int i=0; i<n; i++) B[i][n] = b[i];

  for (int i=0; i<n; i++) {
    int pivot = i;
    for (int j=i; j<n; j++) {
      if (abs(B[j][i]) > abs(B[pivot][i])) pivot = j;
    }
    swap(B[i], B[pivot]);

    // no solution, or infinite solution
    if (abs(B[i][i]) < EPS) return vec();

    for (int j=i+1; j<=n; j++) B[i][j] /= B[i][i];
    for (int j=0; j<n; j++) {
      if (i != j) {
        for (int k = i+1; k<=n; k++) B[j][k] -= B[j][i] * B[i][k];
      }
    }
  }
  vec x(n);
  for (int i=0; i<n; i++) x[i] = B[i][n];
  return x;
}
```