ARTICULATION POINTS + BRIDGES O(V+E) 2

TOPOLOGICAL SORT O( V+E) 2

ARTICULATION POINTS + BRIDGES O(V+E) 2

EULER TOUR O(V+E) 3

FLOYD WARSHALL (cnt paths + classic + minimax ) 3

STRONGLY CONNECTED COMPONENTS O (V+E) 3

LOWEST COMMON ANCESTOR 3

BELLMAN FORD O(VE) 4

PRIM O( (V+E )LOG(E) ) 4

KRUSKAL ( E LOG ( E ) ) 4

DIJKSTRA O( (V+E)LOG ( E ) ) 4

GREEDY GRAPH COLORING 4

MAX FLOW – EDMONDS KARP O(min( E^2\*V, E\*FLOW )) 5

MAX MATCHING FOR BI.GRAPH – KUHN’S O(EV) 5

MCMF USING BELLMAN FORD O(min(E^2\*V^2, E\*V\*FLOW)) 5

HUNGERIAN WEIGHTED MATCHING O (V^3) 6

MCMF USING DIJKSTRA (O(E\*LOG(E)\*MAXFLOW) ) 7

MIN CUT- STOER WAGNER ( UNDIR. GRAPH ) O(V^3) 7

MAXFLOW- DINIC O(V^3) 8

BIP MATCHING – VERTEX COVER O(VE) 8

SEG TREE ( LAZY UPDATE ) 9

SEG TREE ( NORMAL UPDT + MIDSET RANGE CALC ) 10

DISJOINT SET 11

FENWICK TREE 11

COMBINATION 11

SPRAGUE-GRUNDY 11

INPUT READER 12

LOWER/UPPER BOUND 12

LONGEST COMMON INC. SUBSEQ. O(N LOG(K)) 12

LIS O(N LOG(N)) 12

LONGEST COMMON SUBSEQUENCE O(MN) 13

LONGEST COMMON SUBSTRING (MN) 13

LONGEST PALINDROME O(N) 13

Z-ALGORITHM O(N) 13

TRIE O(N) 14

SUFFIX TREE O(N) 14

SUFFIX TREE O(N\*LOG^2(N)) 14

EXTENDED GCD O(LOG(A+B)) 15

CHEBYSHEV’S ALGO 15

FIBONACCI O( LOG(N)) 15

SEIVE OF YARIN 16

STRONG COMBINATION 16

NEXT PERMUTATION 16

NEWTON’S ROOT FINDING METHOD 16

GEOMETRY BASIC DEFINITIONS 16

LINE INTERSECTION 16

LINE TANGENT ANGLE 16

SEGMENT INTERSECTION 16

POINT + SLOPE -> LINE 17

TWO POINTS -> LINE 17

CLOSEST POINT ON LINE 17

TRIANGLE CIRCUM CIRCLE 17

POLYGON AREA 17

TRIANGLE INCIRCLE 17

POINT IN TRIANGLE 18

CONVEX HULL 18

BISECTOR 18

POINT + SLOPE -> LINE2 18

POINT ROTATE AROUND POINT( IN LDBL ) 18

POINT MIRROR TO LINE( IN LDBL ) 18

POLYGON AREA ( ORDERED POINTS ) 18

PREPENDICULAR LINE TO LINE FROM GIVEN POINT 18

LINE DISTANCE TO POINT 18

X OF GIVEN Y 18

Y OF GIVEN X 18

LINE1 TO LINE2 ( TWO POINTS -> AX+BX+C = 0 ) 19

DIST TO POINT 19

CLOSEST POINT ON LINE TO POINT 19

LINE INTERSECTS TO LINE ( BOOLEAN ) 19

LINE – LINE INTERSECTION POINT ( LDBL ) 19

FOOT OF PREPENDICULAR 19

PREPENDICULAR LINE 19

LENGTH OF ARC 19

LENGTH OF CHORD 19

3 POINTS TO CIRCLE( turn(A,B,C)!=0 noncolinear) 19

TRIANGLE AREA 19

GREAT CIRCLE DISTANCE 19

IS POINT INSIDE POLYGON( ORDERED POINTS) 19

BISECTOR 19

### ARTICULATION POINTS + BRIDGES O(V+E)

vector<vector<int>> AdjList;

vector<int> dfs\_num;

vector<int> dfs\_parent;// distinguish backedge

vector<int> dfs\_low;//loop head

vector<bool> articulation\_vertex;

int dfsNumberCounter, dfsRoot, rootChildren;

void init(int n) {

dfsNumberCounter = 0;

dfs\_low.assign(n, 0);

dfs\_num.assign(n, DFS\_WHITE);

dfs\_parent.assign(n, -1);

articulation\_vertex.assign(n, false);

}

void articulationPointAndBridge(int u) {

dfs\_low[u] = dfs\_num[u] = dfsNumberCounter++; for(int j = 0; j < (int)AdjList[u].size(); j++) {

int v = AdjList[u][j];

if (dfs\_num[v] == DFS\_WHITE) {

dfs\_parent[v] = u;

//if rootChildren <= 1 -> root is an art. point

if (u == dfsRoot) rootChildren++;

articulationPointAndBridge(v);

if (dfs\_low[v] >= dfs\_num[u])

articulation\_vertex[u] = true;

if (dfs\_low[v] > dfs\_num[u])

printf("Edge(%d,%d)is a bridge\n",u,v);

dfs\_low[u] = min(dfs\_low[u], dfs\_low[v]);

}

else if (v != dfs\_parent[u])

dfs\_low[u] = min(dfs\_low[u], dfs\_num[v]);

}

}

### TOPOLOGICAL SORT O( V+E)

vector<vector<int>> AdjList;

bool visit (intv, vector<int>&order, vector<int>&color) {

color[v] = 1;// in stack

for (int j = 0; j < AdjList[v].size(); j++) {

int dst = AdjList[v][j];

if(color[dst] == 2) continue;//cross edge( no matter)

if(color[dst] == 1) return false;// loop found!

if (!visit(dst, order, color)) return false;

}

order.push\_back(v); color[v] = 2;// visited

return true;

}

bool topologicalSort(vector<int>&order) {//returns true if could be sorted and the answer is set in "order"

int n = AdjList.size();

vector<int> color(n);// open vertex->color is set to zero

for (int u = 0; u < n; u++)

if (!color[u] && !visit(u, order, color)) returnfalse;

reverse (order.begin(), order.end());

returntrue;

}

### ARTICULATION POINTS + BRIDGES O(V+E)

static ArrayList<Integer>[] w;

static int[] nums;// fill -1

static int[] low;

static int[] parent;

static boolean[] ap;

static int counter;

static int rChildren;// 0

static int root;

static void APB(int start) {

low[start] = nums[start] = counter++;

for (int i : w[start]) {

if (nums[i] == -1) {

parent[i] = start;

if (start == root)

rChildren++;

APB(i);

if (low[i] >= nums[start]) {//for Art. points

ap[start] = true;

}

// if(low[i]>nums[start]){

// found bridge

// }

low[start] = Math.min(low[start], low[i]);

} else if (i != parent[start]) {

low[start] = Math.min(low[start], nums[i]);

}

}

}

//////// IN MAIN:

Arrays.fill(nums, -1);

for (int i = 1; i <= n; i++)

if (nums[i] == -1) {

rChildren = 0;

root = i;

APB(i);

ap[i] = (rChildren > 1); if true -> i is an art. point

}

### EULER TOUR O(V+E)

vector<vector<int>> AdjList;

// for directed graph

void visit(int a , vector<int>& path) {

while (!AdjList[a].empty()){ // for every outgoing edge

int b = AdjList[a].back(); AdjList[a].pop\_back();

visit (b, path);

}

path.push\_back (a);

}

// undirected edges, every edge has to be crossed

exactly once PUSH EACH EDGE TWICE

void visit(vector<vector<int>>&adj, ints,vector<int>&path) {

for (int j = 0; j < AdjList[s].size(); j++)

{

intv = AdjList[s][j];

if (adj[s][v]) {

--adj[s][v]; --adj[v][s]; // crossed the edge

visit(adj, v , path);

} }

path.push\_back(s);

}

bool eulerPath (int s , vector<int>&path) {

int n = AdjList.size(), m = 0;

vector<int> deg (n); // deg[n] = out-deg[n] - in-deg[n]

for (int u = 0; u < n; u++) { // compute deg[u]

m += AdjList[u].size();

for (int j = 0; j < AdjList[u].size(); j++)

--deg[AdjList[u][j]]; // in-deg

deg[u] += AdjList[u].size(); // out-deg

}

int k = n - count (deg.begin(), deg.end(), 0);

if (k == 0 || (k == 2 && deg[s] == 1)) {

path.clear();

visit (s , path); reverse (path.begin(), path.end());

return path.size () == m + 1;

}

returnfalse;

}

### FLOYD WARSHALL (counting paths + classic + minimax )

for s from 2 to n

for i from 1 to n for j from 1 to n for k from 1 to n

a[i][j][s] = max(a[i][j][s], a[i][j][s-1]+a[j][k][1])

REP (k, 0, V - 1) REP (i, 0, V - 1) REP (j, 0, V - 1)

graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j]);

// finding the minimum of maximum edge weight amongall possible paths from i to j.

d[i][j] = min(d[i][j], max(d[i][k], d[k][j]));

### STRONGLY CONNECTED COMPONENTS O (V+E)

static ArrayList<Integer>[] w;

static int[] low;

static int[] nums;

static boolean[] instack;

static Stack<Integer> stack = new Stack<Integer>();

static int counter;//dar main 0 she, ghable seda zadane scc

static void scc(int start) {

low[start] = nums[start] = counter++;

instack[start] = true;

stack.push(start);

for (int i : w[start]) {

if (nums[i] == -1) scc(i);

if (instack[i])

low[start] = Math.min(low[i], low[start]);

}

if (nums[start] == low[start]) {

// yek scc peyda shode ke dar stack hast

}

}

### LOWEST COMMON ANCESTOR

void process(intN, intT[MAXN], intP[MAXN][LOGMAXN]) {

int i, j;

//we initialize every element in P with -1

for (i = 0; i <N; i++)

for (j = 0; 1 << j <N; j++) P[i][j] = -1;

//the first ancestor of every node i is T[i]

for (i = 0; i <N; i++) P[i][0] = T[i];

//bottom up dynamic programing

for (j = 1; 1 << j <N; j++) for (i = 0; i <N; i++)

if (P[i][j - 1] != -1) P[i][j] = P[P[i][j - 1]][j - 1];

}

int query(intN, intP[MAXN][LOGMAXN], intT[MAXN],intL[MAXN], intp, intq) {

int tmp, log, i;

//if p is situated on a higher level than q swap them

if (L[p] <L[q]) tmp = p, p = q, q = tmp;

//we compute the value of [log(L[p)]

for (log = 1; 1 << log <= L[p]; log++); log--;

/ find the ancestor of node p situated on the same level

//with q using the values in P

for (i = log; i >= 0; i--)

if (L[p] - (1 << i) >= L[q]) p = P[p][i];

if (p == q) return p;

// compute LCA(p, q) using the values in P

for (i = log; i >= 0; i--)

if (P[p][i] != -1 &&P[p][i] != P[q][i])

p = P[p][i], q = P[q][i];

returnT[p];

}

### BELLMAN FORD O(VE)

static int[] dist;

static ArrayList<Edge> edges;

static boolean bellmanFord(int s,int v) {

dist = new int[V]; Arrays.fill(dist, INF);

dist[s] = 0;

for (int i = 0; i < V - 1; i++)

for (Edge e : edges)

dist[e.v] = Math.min(dist[e.v], dist[e.u] + e.w);

for (Edge e : edges)

if (dist[e.v] > dist[e.u] + e.w)

return false;

return true;

}

### PRIM O( (V+E )LOG(E) )

typedefpair<int, int>ii;

vector<vector<ii>> AdjList;

vector<bool> taken;

priority\_queue<ii> pq;

void process(int vtx) {

taken[vtx] = 1;

for (int j = 0; j < AdjList[vtx].size(); j++) {

ii v = AdjList[vtx][j];

if (!taken[v.first]) pq.push(ii(-v.second, -v.first));

} }

int prim(int n)

{

taken.assign(n, 0);

process(0);

int mst\_cost = 0;

while (!pq.empty()) {

ii front = pq.top(); pq.pop();

int u = -front.second, w = -front.first;

if (!taken[u])

mst\_cost += w, process(u);

}

return mst\_cost;

}

### KRUSKAL ( E LOG ( E ) )

int kruskal(int n, list<Edge>& E) {

E.sort(); DisjointSet ds(n); int mst = 0;

for (int m = 0; m < n-1; ) {

Edge e = E.front(); E.pop\_front();

int i = ds.find(e.u), j = ds.find(e.v);

if (i != j) { ds.merge(i,j); mst += e.w; m++; }

}

return mst;

}

### DIJKSTRA O( (V+E)LOG ( E ) )

static void dijktra(int s) {

dist = new int[V]; Arrays.fill(dist, INF);

parent = new int[V]; Arrays.fill(parent, -1);

checked = new boolean[V];

Queue<Pair> q = new PriorityQueue<Pair>();

dist[s] = 0;

q.add(new Pair(0, s));

while (!q.isEmpty()) {

Pair front = q.remove();

int u = front.vertex;

if (!checked[u]) {

checked[u] = true;

for (Pair p : list.get(u)) {

int v = p.vertex, w = p.weight;

if (dist[u] + w < dist[v]) {

dist[v] = dist[u] + w;

q.add(new Pair(dist[v], v));

parent[v] = u;

} } } }}

### GREEDY GRAPH COLORING

public class GraphColoringGreedy {

public static int[] color(List<Integer>[] graph) {

int n = graph.length;

BitSet[] used = new BitSet[n];

int[] colors = new int[n];

PriorityQueue<Long> q = new PriorityQueue<Long>(n);

for (int i = 0; i < n; i++) {

used[i] = new BitSet();

colors[i] = -1;

q.add((long) i);

}

for (int i = 0; i < n; i++) {

int bestu;

while (true) {

bestu = q.remove().intValue();

if (colors[bestu] == -1) break;

}

int c = used[bestu].nextClearBit(0);

colors[bestu] = c;

for (int v : graph[bestu]) {

if (!used[v].get(c)) {

used[v].set(c);

if (colors[v] == -1)

q.add(v - ((long) used[v].cardinality() << 32)); } } }

return colors;

}

public static void main(String[] args) {

int n = 5; List<Integer>[] g = new List[n];

for (int i = 0; i < n; i++) g[i] = new ArrayList<Integer>();

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

g[i].add((i + 1) % n);

g[(i + 1) % n].add(i);

}

}

System.out.println(Arrays.toString(color(g))); } }

### MAX FLOW – EDMONDS KARP O(min( E^2\*V, E\*FLOW ))

public static int edmondsKarp(ArrayList<Integer>[] E, int[][] C, int s,

int t) {

// Residual capacity from u to v is C[u][v] - F[u][v]

int[][] F = new int[n][n];

while (true) {

int[] P = new int[n]; // Parent table

Arrays.fill(P, -1);

P[s] = s;

int[] M = new int[n]; // Capacity of path to node

M[s] = Integer.MAX\_VALUE;

// BFS queue

Queue<Integer> Q = new LinkedList<Integer>();

Q.offer(s);

LOOP: while (!Q.isEmpty()) {

int u = Q.poll();

for (int v : E[u]) {

// There is available capacity,

// and v is not seen before in search

if (C[u][v] - F[u][v] > 0 && P[v] == -1) {

P[v] = u;

M[v] = Math.min(M[u], C[u][v] - F[u][v]);

if (v != t)

Q.offer(v);

else {

// Backtrack search, and write flow

while (P[v] != v) {

u = P[v];

F[u][v] += M[t];

F[v][u] -= M[t];

v = u;

}

break LOOP;

} } } }

if (P[t] == -1) { // We did not find a path to t

int sum = 0;

for (int x : F[s])

sum += x;

return sum;

} } }

### MAX MATCHING FOR BI.GRAPH – KUHN’S O(EV)

public class MaxMatching2 {

static boolean findPath

(List<Integer>[] g, int u1, int[] matching, boolean[] vis) {

vis[u1] = true;

for (int v : g[u1]) {

int u2 = matching[v];

if (u2 == -1|| !vis[u2] && findPath(g, u2, matching, vis))

{

matching[v] = u1;

return true;

}

}

return false;

}

public static int maxMatching(List<Integer>[] g, int n2) {

int n1 = g.length;

int[] matching = new int[n2];

Arrays.fill(matching, -1);

int matches = 0;

for (int u = 0; u < n1; u++)

if (findPath(g, u, matching, new boolean[n1]))

++matches;

return matches;

}

public static void main(String[] args) {

int n1 = 2;

int n2 = 3;

List<Integer>[] g = new List[n1];

for (int i = 0; i < n1; i++)

g[i] = new ArrayList<Integer>();

//input g as in : g[0].add(2);

System.out.println(2 == maxMatching(g, n2));

}

}

### MCMF USING BELLMAN FORD O(min(E^2\*V^2, E\*V\*FLOW))

public class MinCostFlowBF {

static class Edge {

int to, f, cap, cost, rev;

Edge(int v, int cap, int cost, int rev) {

this.to = v;

this.cap = cap;

this.cost = cost;

this.rev = rev;

}

}

public static List<Edge>[] createGraph(int n) {

List<Edge>[] graph = new List[n];

for (int i = 0; i < n; i++)

graph[i] = new ArrayList<Edge>();

return graph;

}

public static void addEdge(List<Edge>[] graph, int s, int t, int cap, int cost) {

graph[s].add(new Edge(t, cap, cost, graph[t].size()));

graph[t].add(new Edge(s, 0, -cost, graph[s].size() - 1));

}

static void bellmanFord(List<Edge>[] graph, int s, int[] dist, int[] prevnode, int[] prevedge, int[] curflow) {

int n = graph.length;

Arrays.fill(dist, 0, n, Integer.MAX\_VALUE);

dist[s] = 0;

curflow[s] = Integer.MAX\_VALUE;

boolean[] inqueue = new boolean[n];

int[] q = new int[n];

int qt = 0;

q[qt++] = s;

for (int qh = 0; (qh - qt) % n != 0; qh++) {

int u = q[qh % n];

inqueue[u] = false;

for (int i = 0; i < (int) graph[u].size(); i++) {

Edge e = graph[u].get(i);

if (e.cap <= e.f)

continue;

int v = e.to;

int ndist = dist[u] + e.cost;

if (dist[v] > ndist) {

dist[v] = ndist;

prevnode[v] = u;

prevedge[v] = i;

curflow[v] = Math.min(curflow[u], e.cap - e.f);

if (!inqueue[v]) {

inqueue[v] = true;

q[qt++ % n] = v;

}

}

}

}

}

public static int[] minCostFlow(List<Edge>[] graph, int s, int t, int maxf) {

int n = graph.length;

int[] dist = new int[n];

int[] curflow = new int[n];

int[] prevedge = new int[n];

int[] prevnode = new int[n];

int flow = 0;

int flowCost = 0;

while (flow < maxf) {

bellmanFord

(graph, s, dist, prevnode, prevedge, curflow);

if (dist[t] == Integer.MAX\_VALUE)

break;

int df = Math.min(curflow[t], maxf - flow);

flow += df;

for (int v = t; v != s; v = prevnode[v]) {

Edge e = graph[prevnode[v]].get(prevedge[v]);

e.f += df;

graph[v].get(e.rev).f -= df;

flowCost += df \* e.cost;

}

}

return new int[]{flow, flowCost};

}

public static void main(String[] args) {

List<Edge>[] graph = createGraph(3);

// input g as in: addEdge(graph, 0, 1, 3, 1);

int[] res = minCostFlow

(graph, 0, 2, Integer.MAX\_VALUE);

int flow = res[0];

int flowCost = res[1];

}

}

### HUNGERIAN WEIGHTED MATCHING O (V^3)

#defineinf 1000000000

#defineNN 200

int n, m, weight[NN][NN];

int x[NN], y[NN];// X[i] = j means i-th vertex of left matched to j-th vertex of rigth

int hungarian() {

int p, q;

vector<int> fx(n, inf), fy(n, 0);

memset(x, -1, sizeof x); memset(y, -1, sizeof y);

for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j)

fx[i] = max(fx[i], weight[i][j]);

for (int i = 0; i < n; ) {

vector<int> t(n, -1), s(n+1, i);

for (p = q = 0; p <= q && x[i] < 0; ++p)

for (int k = s[p], j = 0; j < n && x[i] < 0; ++j)

if (fx[k] + fy[j] == weight[k][j] && t[j] < 0) {

s[++q] = y[j], t[j] = k;

if (s[q] < 0) for (p = j; p >= 0; j = p)

y[j] = k = t[j], p = x[k], x[k] = j;

}

if (x[i] < 0) {

int d = inf;

for (int k = 0; k <= q; ++k) for (int j = 0; j < n; ++j)

if (t[j] < 0) d = min(d, fx [s[k]] + fy[j] - weight[s[k]][j]);

for (int j = 0; j < n; ++j) fy[j] += (t[j] < 0? 0: d);

for (int k = 0; k <= q; ++k) fx[s[k]] -= d;

} else ++i;

}

int ret = 0;

for (int i = 0; i < n; ++i) ret += weight[i][x[i]];

return ret;}

### MCMF USING DIJKSTRA (O(E\*LOG(E)\*MAXFLOW) )

/\*fnet contains the flow network. Careful: both fnet[u][v] andfnet[v][u] could be positive.\*\*/

#defineNN 1024

int cap[NN][NN], cost[NN][NN], fnet[NN][NN], adj[NN][NN], deg[NN];

int par[NN], d[NN], q[NN], inq[NN], qs, pi[NN];

#defineCLR(a, x) memset( a, x, sizeof( a ) )

#defineInf (INT\_MAX/2)

#defineBUBL { t = q[i]; q[i] = q[j]; q[j] = t; t = inq[q[i]]; inq[q[i]] = inq[q[j]]; inq[q[j]] = t; }

#definePot(u,v) (d[u] + pi[u] - pi[v])

bool dijkstra( intn, ints, intt ){

CLR( d, 0x3F ); CLR( par, -1 ); CLR( inq, -1 );

d[s] = qs = 0; inq[q[qs++] = s] = 0; par[s] = n;

while( qs ){

int u = q[0]; inq[u] = -1; q[0] = q[--qs];

if( qs ) inq[q[0]] = 0;

for( int i = 0, j = 2\*i + 1, t; j < qs; i = j, j = 2\*i + 1 ){

if( j + 1 < qs && d[q[j + 1]] < d[q[j]] ) j++;

if( d[q[j]] >= d[q[i]] ) break;

BUBL;

}

for(int k = 0, v =adj[u][k]; k<deg[u]; v = adj[u][++k] ){

if( fnet[v][u] && d[v] >Pot(u,v) - cost[v][u] )

d[v] = Pot(u,v) - cost[v][par[v] = u];

if( fnet[u][v] < cap[u][v] && d[v] >Pot(u,v) + cost[u][v] )

d[v] = Pot(u,v) + cost[par[v] = u][v];

if( par[v] == u ){

if( inq[v] < 0 ) { inq[q[qs] = v] = qs; qs++; }

for( int i = inq[v], j = ( i - 1 )/2, t;

d[q[i]] < d[q[j]]; i = j, j = ( i - 1 )/2 )

BUBL;

} } }

for( int i = 0; i <n; i++ ) if( pi[i] <Inf ) pi[i] += d[i];

return par[t] >= 0; }

int mcmf4( intn, ints, intt, int&fcost ){

CLR( deg, 0 );

for( int i = 0; i <n; i++ ) for( int j = 0; j <n; j++ )

if( cap[i][j] || cap[j][i] ) adj[i][deg[i]++] = j;

CLR( fnet, 0 ); CLR( pi, 0 );

int flow = fcost = 0;

while( dijkstra( n, s, t ) ){

int bot = INT\_MAX;

for( int v = t, u = par[v]; v != s; u = par[v = u] )

bot <= fnet[v][u] ? fnet[v][u] : ( cap[u][v] - fnet[u][v] );

for( int v = t, u = par[v]; v != s; u = par[v = u] )

if( fnet[v][u] ) { fnet[v][u] -= bot; fcost -= bot \* cost[v][u]; }

else { fnet[u][v] += bot; fcost += bot \* cost[u][v]; }

flow += bot;

}

return flow;

}

int main(){

int numV, m, a, b, c, cp, s, t;

memset( cap, 0, sizeof( cap ) );

cin >> numV; cin >> m>> s >> t;

for (int i=0; i<m; i++) {

cin >> a >> b >> cp >> c;

cost[a][b] = c; // cost[b][a] = c;

cap[a][b] = cp; // cap[b][a] = cp;

}

int fcost;

int flow = mcmf4( numV, s, t, fcost );

cout <<"flow: "<< flow <<" cost: "<< fcost << endl;

}

### MIN CUT- STOER WAGNER ( UNDIR. GRAPH ) O(V^3)

#define NN 256 // Maximum number of vertices in the graph

#define MAXW 1000 // Maximum edge weight (MAXW \* NN \* NN must fit into an int)

int g[NN][NN], v[NN], w[NN], na[NN]; // Adjacency matrix and some internal arrays

bool a[NN];

int minCut( int n ) {

for( int i = 0; i < n; i++ ) v[i] = i;// init the remaining v set

int best = MAXW \* n \* n;

while( n > 1 ) {

a[v[0]] = true;

for( int i = 1; i < n; i++ ) {

a[v[i]] = false; na[i - 1] = i;

w[i] = g[v[0]][v[i]];

}

int prev = v[0];

for( int i = 1; i < n; i++ ) {

int zj = -1;

for( int j = 1; j < n; j++ )

if( !a[v[j]] && ( zj < 0 || w[j] > w[zj] ) ) zj = j;

a[v[zj]] = true; // add it to A

if( i == n - 1 ) { // last vertex?

best = min(best, w[zj]);//remember the cut w

for( int j = 0; j < n; j++ )

g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][v[j]];

v[zj] = v[--n]; break;

}

prev = v[zj];

for( int j = 1; j < n; j++ ) if( !a[v[j]] )

w[j] += g[v[zj]][v[j]];

} }

return best;}

### MAXFLOW- DINIC O(V^3)

#defineNN 105 // the maximum number of vertices

int cap[NN][NN], deg[NN], adj[NN][NN]; //cap[u][v] is the capacity of the edge u->v

int q[NN], prev[NN]; // BFS stuff

int dinic( int n, int s, int t )

{

int flow = 0;

while( true ){ // find an augmenting path

memset( prev, -1, sizeof( prev ) );

int qf = 0, qb = 0; prev[q[qb++] = s] = -2;

while( qb > qf && prev[t] == -1 )

for( int u = q[qf++], i = 0, v; i < deg[u]; i++ )

if( prev[v = adj[u][i]] == -1 && cap[u][v] )

prev[q[qb++] = v] = u;

if( prev[t] == -1 ) break; // we’re done finding paths

for( int z = 0; z <n; z++ )

if( cap[z][t] && prev[z] != -1 ){

int bot = cap[z][t];

for( int v = z,u = prev[v]; u >= 0;v = u,u = prev[v] )

bot = min(bot,cap[u][v]);

if( !bot ) continue;

cap[z][t] -= bot; cap[t][z] += bot;

for( int v =z,u = prev[v]; u >= 0;v = u, u =prev[v] )

cap[u][v] -= bot; cap[v][u] += bot;

flow += bot;

} }

return flow;

}

int main()

{

memset( cap, 0, sizeof( cap ) );

int n, s, t, m;

scanf( " %d %d %d %d", &n, &s, &t, &m );

while( m-- ) {

int u, v, c; scanf( " %d %d %d", &u, &v, &c );

cap[u][v] = c;

}

memset( deg, 0, sizeof( deg ) );

for( int u = 0; u < n; u++ )

for( int v = 0; v < n; v++ )

if( cap[u][v] || cap[v][u] ) adj[u][deg[u]++] = v;

printf( "%d\n", dinic( n, s, t ) );}

### BIP MATCHING – VERTEX COVER O(VE)

#define M 128

#define N 128

bool graph[M][N], seen[N];

int matchL[M], matchR[N];

int n, m;

bool bpm( int u ){

for( int v = 0; v < n; v++ ) if( graph[u][v] ){

if( seen[v] ) continue;

seen[v] = true;

if( matchR[v] < 0 || bpm( matchR[v] ) ){

matchL[u] = v; matchR[v] = u;

returntrue;

} }

returnfalse; }

vector<int> vertex\_cover(){ //Vertices on the left side (n side) are labeled like this: m+i where i is the index

set<int> s, t, um; // um = UnMarked

vector<int> vc;

for(int i = 0; i < m; i++)

if(matchL[i]==-1) s.insert(i), um.insert(i);

while( um.size() ){

int v = \*(um.begin());

for(int i = 0; i < n; i++)

if( graph[v][i] && matchL[v]!=i){

t.insert(i);

if( s.find(matchR[i]) == s.end())

s.insert(matchR[i]), um.insert(matchR[i]);

}

um.erase(v);

}

for(int i = 0; i < m; i++)

if( s.find(i) == s.end() ) vc.push\_back(i);

for(set<int>::iterator i = t.begin(); i != t.end(); i++)

vc.push\_back((\*i) + m);

return vc; }

int main(){// Read input and populate graph[][]

memset( matchL, -1, sizeof( matchL ) );

memset( matchR, -1, sizeof( matchR ) );

int cnt = 0;

for( int i = 0; i < m; i++ ){

memset( seen, 0, sizeof( seen ) );

if( bpm( i ) ) cnt++;

}

vector<int> vc = vertex\_cover();

/\*cnt contains the number of happy pigeons

matchL[i] :the hole of pigeon i or -1 if pigeon i is unhappy

matchR[j] :the pigeon in hole j or -1 if hole j is empty

vc contains the Vertex Cover\*/ }

### SEG TREE ( LAZY UPDATE )

#define maxn 1024000 // more than complete pow 2

#define K 21+1 // ceil( logN ) + 1 mahze ehtiat

#define maxl 1 << ( K+1) + 4 // mahze ehtiat

int seg\_tree[maxl];//n

char updateData[maxl];//u

int left[maxl];//a

int right[maxl];//b

bool pirates[maxn];

char tmp [52];

int ind;

void init\_seg\_tree( int a , int b , int n ){

left[n] = a;

right[n] = b;

updateData[n] = ' ';

if( a == b )

seg\_tree[n] = pirates[a];

else if( a < b )

{

int mid = (a+b)>>1;

init\_seg\_tree( a , mid , 2\*n );

init\_seg\_tree( mid+1 , b , 2\*n + 1 );

seg\_tree[n] = seg\_tree[2\*n] + seg\_tree[2\*n+1];

}

}

void setNodeUpdateType( int n , char t ){

char & u = updateData[n];

if( t == 'I' && u != ' ' )

{

if( u == 'E' ) u = 'F';

else if( u == 'F' ) u = 'E';

else u = ' ';

}

else u = t;

}

void updateNode( int n )

{

char & u = updateData[n];

if( u == ' ' ) return ;

else if( u == 'E' ) seg\_tree[n] = 0;

else if( u == 'F' ) seg\_tree[n] = right[n] - left[n] +1;

else seg\_tree[n] = right[n]-left[n]+1-seg\_tree[n];

setNodeUpdateType( 2\*n , u );

setNodeUpdateType( 2\*n+1 , u );

u = ' ';

}

void update\_range( int n , int qa , int qb , char t ){

if( left[n] >= qa && right[n] <= qb )

setNodeUpdateType( n , t );

updateNode( n );

if( left[n] > qb || right[n] < qa )//inja ham update mikone! chon parentesh badan mikhad az etelaAte segTreesh estefade kone

return;

if( !( left[n] >= qa && right[n] <= qb ) )

{

update\_range( 2\*n , qa , qb , t );

update\_range( 2\*n+1 , qa , qb , t );

seg\_tree[n] = seg\_tree[2\*n]+seg\_tree[2\*n+1];

}

}

// manteghesh shabihe balayi as har node ke besh resdidim update harvakh bacheha update shodan khodetam update kon

int query\_range( int n , int qa , int qb )

{

updateNode( n );// har masiri ke mirim payin update mikonim! mese udpate range chon momkene parent bekhadesh

if( left[n] >= qa && right[n] <= qb )

return seg\_tree[n];

else if( right[n] < qa || left[n] > qb )

return 0;

else

{

int ans = query\_range( 2\*n , qa , qb ) + query\_range( 2\*n+1 , qa , qb );

seg\_tree[n] = seg\_tree[2\*n]+seg\_tree[2\*n+1];

return ans;

}

}

int main()

{

int T; scanf("%d" , &T );

for( int t = 1 ; t <= T ; t++ )

{

printf("Case %d:\n" , t);

//memset( seg\_tree , 0 , sizeof seg\_tree );

//memset( updateData , ' ' , sizeof updateData );

ind = 0;

int m;scanf("%d" , &m );

while( m-- )

{

int c;scanf("%d" , &c );

scanf( "%s" , tmp );

while( c-- )

{

char \* p = tmp;

while( \*p )

pirates[ind++] = (\*p++) - '0';

}

}

init\_seg\_tree( 0 , ind-1 , 1 );

int q;scanf("%d" , &q );

int gods = 0;

while( q-- )

{

char c; scanf("%c" , &c );scanf("%c" , &c );

int a , b;scanf("%d%d" , &a , &b );

if( c == 'S' )

printf("Q%d: %d\n" , ++gods , query\_range( 1 , a , b ));

else

update\_range( 1 , a , b , c );

}

}

return 0;}

### SEG TREE ( NORMAL UPDT + MIDSET RANGE CALC )

#define offset 100000 //to handle neg numbers

#define maxn 100004 // yadet nare az tavan 2 o inaye taraf o ina :D bara hamin runtime error mikhori!!! :D

#define k 18+1 // ceil( logN ) + 1 mahze ehtiat

#define maxl 1 << ( k+1) + 4// mahze ehtiat

int seg\_tree[maxl];// frequency

int nums[maxn];

int segBeg[2\*maxn];

int segEnd[2\*maxn];

int n;

void init\_seg\_tree( int b , int e , int node ){

if( b == e )seg\_tree[node] = 1;

else{

//init both halves

int m = ( b + e )/2;

init\_seg\_tree( b , m , 2\*node );

init\_seg\_tree( m+1 , e , 2\*node + 1 );

// calculate midset range frequnecy

seg\_tree[node] = min( segEnd[ nums[m] ] , e ) - max( segBeg[ nums[m] ] , b ) + 1;

seg\_tree[node] = max( seg\_tree[node] , max( seg\_tree[2\*node] , seg\_tree[2\*node+1] ) );

}

}

int query\_range( int a , int b , int node , int qa , int qb ){

if( qb < a || qa > b ) return 0; //doesn't intersect

// is inside query range

if( a >= qa && b <= qb ) return seg\_tree[node];

int m = (a+b)/2;

int first = query\_range( a , m , 2\*node , qa , qb );

int second=query\_range( m+1 , b , 2\*node+1 , qa , qb );

// calculate midset range frequnecy

int ans = min( segEnd[ nums[m] ] , min( qb , b ) )

- max( segBeg[ nums[m] ] , max( qa , a ) ) + 1;

return max( ans , max( first , second ) );

}

int main(){

while( scanf("%d" , &n ) != (-1) &&n ) {

int q;scanf("%d" , &q );

int numBef = 2\*offset + 1;

for( int i = 0 ; i < n ; i++ ) {

scanf("%d" , &nums[i] );

nums[i] += offset;

if(nums[i] != numBef){

segEnd[numBef] = i - 1;

segBeg[nums[i]] = i;

}

numBef = nums[i];

}

segEnd[nums[n - 1]] = n - 1;

init\_seg\_tree( 0 , n-1 , 1 );

while( q-- )

{

int a , b; scanf("%d%d" , &a , &b ); a--; b--;

printf("%d\n" , query\_range( 0 , n-1 , 1 , a , b ));

} } }

### DISJOINT SET

class DisjointSets {

int size;

int[] parent;

int[] depth;

public DisjointSets(int n) {

size = n;

parent = new int[n];

depth = new int[n];

for (int i = 0; i < n; i++) {

parent[i] = i; depth[i] = 0;

}

}

int find(int x) {

while (x != parent[x]) x = parent[x];

return x;

}

boolean sameSet( int i , int j ){

return find(i) == find(j);

}

boolean merge(int i, int j) {

i = find(i);

j = find(j);

if (i == j) return false;

if (depth[i] < depth[j])

parent[i] = j;

else {

parent[j] = i;

if (depth[i] == depth[j])

++depth[i];

}

--size;

return true; }}

### FENWICK TREE

int tree1D[ maxval+1 ] = {};

void update1D( int idx , int val ){

while( idx <= maxval ) {

tree1D[idx] += val;

idx += idx & -idx;

}

}

int cumulative1D( int idx ){

int sum = 0;

while( idx ){

sum += tree1D[idx];

idx -= idx & -idx;

}

return sum;

}

// 2D example:

void update(int i, int j, int val){

while (i <= max\_i){

int j1 = j;

while (j1 <= max\_j){

tree[i][j1] += val;

j1 += (j1 & -j1);

}

i += (i & -i);

} }

### COMBINATION

public static long entekhab(int n,int k){

if(k>n-k) k=n-k;

long t=1;

for(int i=1;i<=k;i++) t=(n-i+1)\*t/i;

return t;

}

### SPRAGUE-GRUNDY

// given the adjacency matrix it return the colors of each node. ATTENTION: graph must be acyclic

#define MAX 100

int sg[MAX], graph[MAX][MAX];

int s\_g(int i,int Node){

char used[MAX];

memset(used,0,sizeof(char)\*MAX);

int k=0, fr; //follower

if(sg[i]!=-1) return sg[i];

for(int j=0;j<Node;j++)

if(graph[i][j]){

if(sg[j]==-1){

fr = s\_g(j,Node); sg[j]=fr;

}

used[sg[j]]=1;

}

k=0;

while(used[k]) k++;

return k;

}

void clear(){

for(int i=0;i<MAX;i++)sg[i]=-1;

}

int main(){

int Node=0;clear();cin>>Node;

for(int i=0;i<Node;i++)

for(int j=0;j<Node;j++) cin>>graph[i][j];

for(int i=0;i<Node;i++) sg[i] = s\_g(i,Node);

for(int i=0;i<Node;i++) cout<<sg[i]<<" ";

}

### INPUT READER

static class InputReader {

private BufferedReader reader;

private StringTokenizer tokenizer;

public InputReader(InputStream stream) {

reader = new BufferedReader(new InputStreamReader(stream));

tokenizer = null;

}

public String nextLine() {

try { return reader.readLine();

} catch (IOException e) {

e.printStackTrace();

return null;

}

}

public String next() {

while (tokenizer == null || !tokenizer.hasMoreTokens()) {

try {

tokenizer = new StringTokenizer(reader.readLine());

} catch (IOException e) {

throw new RuntimeException(e);

}

}

return tokenizer.nextToken();

}

public int nextInt() {

return Integer.parseInt(next());

}

public long nextLong() {

return Long.parseLong(next());

}

public double nextDouble() {

return Double.parseDouble(next());

}

}

### LOWER/UPPER BOUND

//{1,4,4,7,9} upper(0)=lower(0)=0|upper(10)=lower(10)=5

//upper(4)=3 lower(4)=1

Integer[] memo = { 1, 4, 4, 7, 9 };

int index1 = Arrays.binarySearch(memo, target,

new Comparator<Integer>() {

// upper-bound

public int compare(Integer a, Integer b) {

if (a <= b) return -1;

return 1;

}

});

index1 = -index1 - 1;//CAREFUL !!!!!

int index2 = Arrays.binarySearch(memo, target,

new Comparator<Integer>() {

// lower-bound

public int compare(Integer a, Integer b) {

if (a < b) return -1;

return 1;

}

});

index2 = -index2 - 1;//CAREFUL !!!!!

static int compare(double a, double b) {

if (a - b > 1e-9) return 1;

if (b - a > 1e-9) return -1;

return 0; }

### LONGEST COMMON INC. SUBSEQUENCE O(N LOG(K))

vector<int> LCIS(vector<int> A, vector<int> B){

int N=A.size(),M=B.size(),i,j;

vector<int> C(M,0), prev(M,0), res;

for (i=0;i<N;i++){

int cur=0,last=-1;

for (j=0;j<M;j++){

if (A[i]==B[j] && cur+1>C[j])

C[j]=cur+1; prev[j]=last;

if (B[j]<A[i] && cur<C[j])

cur=C[j]; last=j;

} }

int length=0,index=-1;

for (i=0;i<M;i++)

if (C[i]>length)

length=C[i]; index=i;

if (length>0){

while (index!=-1){

res.push\_back(B[index]); index=prev[index];

}

reverse(res.begin(),res.end());

}

return res;}

### LIS O(N LOG(N))

static int lis(ArrayList<Integer> array) {

Integer[] memo = new Integer[array.size()];

Arrays.fill(memo, Integer.MAX\_VALUE / 2);

for (int a : array) {

int index = Arrays.binarySearch

(memo, a, new Comparator<Integer>() {

@Override

public int compare(Integer q0, Integer q1) {

if (q0 == q1) return 1;

return q0 - q1;

}

}); index = -1 - index; memo[index] = a; }

for (int i = memo.length - 1;; i--)

if (memo[i] != Integer.MAX\_VALUE / 2)

return i + 1;

}

### LONGEST COMMON SUBSEQUENCE O(MN)

string X, Y;

int L[1005][1005]; // for LCS

int LCS(){

int m = X.length(),n=Y.length(),i,j;

for (i=0;i<=m;i++)

for (j=0;j<=n;j++)

if (i==0 || j==0)L[i][j]=0;

else {

if (X[i-1]==Y[j-1]) L[i][j]=L[i-1][j-1]+1;

else L[i][j]=max(L[i][j-1],L[i-1][j]);

}

return (L[m][n]);

}

void printLCS(int i,int j){ // must run LCS before

if (i==0 || j==0) return;

if (X[i-1]==Y[j-1]){

printLCS(i-1,j-1); cout<<X[i-1];

} elseif (L[i][j]==L[i-1][j]) printLCS(i-1,j);

else printLCS(i,j-1);

}

### LONGEST COMMON SUBSTRING (MN)

int LongestCommonSubstring(const string& str1, const string& str2){

if(str1.empty() || str2.empty()) return 0;

int maxSubstr = 0, m = str1.length(), n = str2.length();

vector<int> curr(n), prev(n); // both n

for(int i = 0; i < m; ++i){

for(int j = 0; j < n; ++j)

if(str1[i] != str2[j]) curr[j] = 0;

else {

if(i == 0 || j == 0) curr[j] = 1;

else curr[j] = 1 + prev[j-1];

maxSubstr = max(maxSubstr, curr[j]);

}

curr.swap(prev);

}

return maxSubstr;}

### LONGEST PALINDROME O(N)

int longest\_palindrome (char \*text, intn) {

int rad [2\*n], i, j , k;

for (i = 0, j = 0; i < 2\*n; i += k , j = max (j-k, 0)) {

while (i-j >= 0 && i+j+1 < 2\*n&&text [(i-j) /2] == text [(i+j+1) /2]) ++j;

rad [i] = j;

for (k = 1; i-k >= 0 && rad [i] - k >= 0 && rad [i-k] != rad [i] - k; ++k)

rad [i+k] = min (rad [i-k], rad [i] - k);

}

return \*max\_element (rad, rad+2\*n); // ret. centre of the longest palindrome

}

### Z-ALGORITHM O(N)

public static int[] zAlgo(char[] s) {

int n = s.length;

int[] z = new int[n];

int R = 0, L = 0;

for (int i = 1; i < n; i++)

if (i > R) {

R = L = i;

while (R < n && s[R - L] == s[R])R++;

z[i] = R - L;

R--;

} else {

int k = i - L;

if (z[k] < R - i + 1)

z[i] = z[k];

else {

L = i;

while (R < n && s[R - L] == s[R])R++;

z[i] = R - L;

R--;

}

}

return z;

}

int indexOf(string& text, string& pattern){

int n = pattern.length();

string str = pattern + '$' + text;

vector <int> z = zAlgorithm(str);

for (int i = 0; i < z.size(); i++) if (z[i] == n) return i-n-1;

return -1;

}

### TRIE O(N)

structTrie {

int value;

Trie \*next [0x100];

Trie () {fill (next , next+0x100, (Trie\*) 0); }

};

Trie\* find (char \*t, Trie \*r) {

for (int i = 0; t[i]; ++i) {

char c = t[i];

if (!r->next[c]) r->next[c] = newTrie;

r = r->next[c];

}

return r;

}

### SUFFIX TREE O(N)

#definepcipair<char, int>

#defineNV N[v]

string s;

structnode {

int p, b, e, link;

vector<pci> children;

node( int\_p, int\_b, int\_e ) { p = \_p, b = \_b, e = \_e, link = -1; }

void addChild( pcia ) { children.push\_back( a ); }

void changeChild( pcia ) {

for( int i = 0; i < children.size(); i++ )

if( children[i].first == a.first ) {

children[i].second = a.second; return;

} }

int length() { return e - b + 1; }

bool gotoNext( charc, int&nv, int&nd) {

if( nd< e - b ) {

if( s[b + nd + 1] == c ) { nd++; returntrue; }

} else {

for( int i = 0; i < children.size(); i++ )

if( children[i].first == c ) {

nv = children[i].second, nd = 0; returntrue;

} }

returnfalse;

} };

vector<node> N;

void add2Tree( ) {

N.clear(); N.push\_back( node( -1, -1, -1 ) ); N[0].link = 0;

int j = 0, pp = -1, v = 0, d = 0;

for( int i = 0; i < s.length(); i++ ) {

pp = -1;

for( ; j <= i; j++ )

if( NV.gotoNext( s[i], v, d ) ) {

if( pp != -1 ) N[pp].link = NV.p; break;

} else {

int id = N.size();

if( d <NV.e - NV.b ) {

if( pp != -1 ) N[pp].link = id;

N.push\_back(node( NV.p, NV.b, NV.b+d ));

N[NV.p].changeChild( pci( s[NV.b], id ) );

NV.b += d + 1; NV.p = pp = id;

N[id].addChild( pci( s[NV.b], v ) );

int len = N[id].p ? d + 1 : d;

v = N[N[id].p].link; d = NV.length() - 1;

while( len ) {

int temp = v;

N[temp].gotoNext( s[i - len] , v, d );

int l = NV.length();

if( len <= l ) { d = len - 1; break; }

d = l - 1; len -= l;

}

id++;

} else {

if( pp != -1 ) N[pp].link = v;

pp = v; v = NV.link; d = NV.length() - 1;

}

N[pp].addChild( pci( s[i], id ) );

N.push\_back( node( pp, i, s.length() - 1 ) );

} } }

void dfs( inta, intdepth = 0 ) {

for( int i = 0; i < N[a].children.size(); i++ )

dfs( N[a].children[i].second, depth + 1 );

}

### SUFFIX TREE O(N\*LOG^2(N))

#define MAXN 200010

int RA[MAXN], SA[MAXN], LCP[MAXN], \*FC, \*SC, step;

char S[MAXN], Q[MAXN];

bool cmp(int a, int b) {

if (step==-1 || FC[a]!=FC[b]) return FC[a] < FC[b];

return FC[a+(1<<step)] < FC[b+(1<<step)];

}

void suffix\_array(char \*S, int n) { // O(n log^2(n))

for (int i=0; i<n; i++) RA[i] = S[SA[i] = i];

for (FC=RA, SC=LCP, step=-1; (1<<step)<n; step++){

sort(SA, SA+n, cmp);

int cnt = 0;

for (int i=0; i<n; i++){

if (i>0 && cmp(SA[i-1],SA[i])) cnt++;

SC[SA[i]] = cnt;

}

if (cnt==n-1) break; // all distinct, no need to continue

swap(FC,SC);

}

for (int i=0; i<n; i++) RA[SA[i]] = i;

}

pair<int, int> range(int n, char \*Q) {

int lo = 1, hi = n, m = strlen(Q), mid = lo; // index 0 - null, valid range = [1..n]

while (lo <= hi) {

mid = (lo + hi) / 2;

int cmp = strncmp(S + SA[mid], Q, m);

if (cmp == 0) break; // found

elseif (cmp > 0) hi = mid - 1;

else lo = mid + 1;

}

if (lo > hi) return make\_pair(-1, -1); // not found

for (lo = mid; lo >= 1 && strncmp(S + SA[lo], Q, m) == 0; lo--); lo++;

for (hi = mid; hi <= n && strncmp(S + SA[hi], Q, m) == 0; hi++); hi--;

return make\_pair(lo, hi);

}

int main(){

int n = strlen(gets(S));

suffix\_array(S, n + 1); // NULL is included!

for (int i = 1; i <= n; i++) printf("%d %s\n", SA[i], S + SA[i]); // SA[0] is the NULL

gets(Q);

pair<int, int> pos = range(n, Q);

if (pos.first != -1 && pos.second != -1) {

printf("%s is found in index [%d .. %d] in SA of %s\n", Q, pos.first, pos.second, S);

printf("They are:\n");

for (int i = pos.first; i <= pos.second; i++) printf(" %s\n", S + SA[i]);

}

else printf("%s is not found in %s\n", Q, S);

} // return 0;

EULER PHI FUNCTION O(N LOG(LOG(N))

int phi[1000000]; // phi[i] = number of co-prime numbers to i less than i. note that: phi[m\*n] = phi[m]\*phi[n]

void all\_phi(int n) {

phi [1] = 1;

for(int i = 2; i <= n; i ++) if (! phi[i]) // prime

for(int j=i; j <=n; j+=i ) { // for each multiple

if (! phi[j]) phi[j] = j;// first time , initialize

phi[j] = phi[j] / i \* (i -1); // multiply

} };

### EXTENDED GCD O(LOG(A+B))

// a x + b y = gcd (a and b)

int extgcd(int a, int b, int&x, int&y) {

int g = a; x = 1; y = 0;

if (b != 0) g = extgcd(b, a % b, y, x), y -= (a/b) \* x;

return g;

}

### CHEBYSHEV’S ALGO

unsignedlonglong p2[30], p5[30];

unsignedlonglong cheb2(unsignedlonglong x){

int res=0, i=1;

while(p2[i]<=x){

res+=x/p2[i]; i++;

}

return res;

}

unsignedlonglong cheb5(unsignedlonglong x){

int res=0, i=1;

while(p5[i]<=x){

res+=x/p5[i]; i++;

}

return res;

}

int main(){

p2[0]=p5[0]=1;

for(int i=1;i<=30;i++) {

p2[i]=p2[i-1]\*2;

p5[i]=p5[i-1]\*5;

}

unsignedlonglong t, n; cin>>t;

while(t--){

cin>>n; cout<<min(cheb2(n),cheb5(n))<<endl;

} }

### FIBONACCI O( LOG(N))

long conquer\_fibonacci(long n){

long i, h, j, k, t;

i = h = 1;

j = k = 0;

while (n > 0) {

if (n % 2 == 1){

t = j \* h;

j = i \* h + j \* k + t;

i = i \* k + t;

}

t = h \* h;

h = 2 \* k \* h + t;

k = k \* k + t;

n = (long) n / 2;

}

return j;

}

int main(){

long n, res;

while (scanf("%ld", &n) == 1) {

res = conquer\_fibonacci(n);

printf("%ld\n", res);

} }

### SEIVE OF YARIN

#define MAXSIEVE 100000000// All primes up to this

#define MAXSIEVEHALF (MAXSIEVE/2)

#define MAXSQRT 5000 // sqrt(MAXSIEVE)/2

char a[MAXSIEVE/16+2];

#define isprime(n) (a[(n)>>4]&(1<<(((n)>>1)&7))) // Works when n is odd have to check for even numbers

void sieve(){

int i,j;

memset(a,255,sizeof(a)); a[0]=0xFE;

for(i=1;i<MAXSQRT;i++)

if (a[i>>3]&(1<<(i&7)))

for(j=i+i+i+1;j<MAXSIEVEHALF;j+=i+i+1) a[j>>3]&=~(1<<(j&7));

}

### STRONG COMBINATION

int main(){

int n,m; cin>>n>>m;

vector <unsignedlonglong> factor;

vector <unsignedint> div;

m=m<(n-m)?m:(n-m);

for(int i=n-m+1;i<=n;i++) factor.push\_back(i);

for(int i=1;i<=m;i++){ int r=1;

for(unsignedint j=0;j<factor.size();j++){

if(factor[j]%i==0){

r=0; factor[j]/=i; break;

} }

if(r) div.push\_back(i);

}

unsignedlonglong ret=1;

for(unsignedint j=0;j<factor.size();j++) ret\*=factor[j];

for(unsignedint j=0;j<div.size();j++) ret/=div[j];

cout<<ret<<endl;

}

### NEXT PERMUTATION

static boolean next\_permutation() {

for (int a = p.length - 2; a >= 0; --a)

if (p[a] < p[a + 1])

for (int b = p.length - 1;; --b)

if (p[b] > p[a]) {

int t = p[a]; p[a] = p[b]; p[b] = t;

for (++a, b = p.length - 1; a < b; ++a, --b) {

t = p[a]; p[a] = p[b]; p[b] = t;

}

returntrue;

}

returnfalse; }

### NEWTON’S ROOT FINDING METHOD

publicstaticfinaldouble EPSILON = 1e-14;

// to find x\* such that f(x\*) = 0, starting at x

publicstaticdouble root(Function f, double x) { // FORGOT TO CHECK IF |F.DERIV(X)| < EPS

while (Math.abs(f.eval(x) / f.deriv(x)) > EPSILON) {

x = x - f.eval(x) / f.deriv(x);

}

return x; }

### GEOMETRY BASIC DEFINITIONS

struct point {double x,y;};

struct line{ double a; double b;double c; } ; //ax+by+c=0

struct circle { point o; double r;};

struct Tri { point p[3];};

struct polygon {point p[1000];int ct;};

double dist(point p1, point p2){

return sqrt((double)((p1.x-p2.x)\*(p1.x-p2.x))+(double)((p1.y-p2.y)\*(p1.y-p2.y)));};

double cross(point p1, point p2, point p0){return (p1.x-p0.x)\*(p2.y-p0.y)-(p2.x-p0.x)\*(p1.y-p0.y);};

const long double PI = (2\*acosl(0.0));

### LINE INTERSECTION

point intersection(line e1, line e2){

point p;

p.x = (e2.b\*e1.c - e1.b\*e2.c) / (e2.a\*e1.b - e1.a\*e2.b);

if (fabs(e1.b) > EPS) /\* test for vertical line \*/

p.y = - (e1.a \* (p.x) + e1.c) / e1.b;

elsep.y = - (e2.a \* (p.x) + e2.c) / e2.b;

return p;}

### LINE TANGENT ANGLE

double tan\_lines(line e1,line e2){// returns the tan of the angel between two lines

double result;

double a = e1.a\*e2.b - e2.a\*e1.b;

double b = e1.a\*e2.a - e1.b\*e2.b;

result = a/b; return result; }

### SEGMENT INTERSECTION

bool segIntersection(point a, point b, point c, point d){

double denominator = (d.y-c.y)\*(b.x-a.x) - (d.x-c.x)\*(b.y-a.y);

double k = ((d.x-c.x)\*(a.y-c.y) - (d.y-c.y)\*(a.x-c.x) ) / denominator;

if( (k<0.0) || (k>1.0) ) returnfalse;

k = ( (b.x-a.x)\*(a.y-c.y) - (b.y-a.y)\*(a.x-c.x) ) / denominator;

if( (k<0.0) || (k>1.0) ) returnfalse;

returntrue;}

### POINT + SLOPE -> LINE

line point\_and\_slope\_to\_line(point p, double m){

line e;

e.a = -m; e.b = 1; e.c = -((e.a\*p.x) + (e.b\*p.y));

return e;

}

### TWO POINTS -> LINE

line LineOfPoint(point p1, point p2){

line ans;

ans.a=(double)(p2.y-p1.y);

ans.b=(double)(p1.x-p2.x);

ans.c=(double)(2\*p1.x\*p1.y-p1.x\*p2.y+p2.x\*p1.y);

return ans; };

### CLOSEST POINT ON LINE

point closest\_point(point p\_in, line l){ needs: point\_and\_slope\_to\_line - intersection

point p\_c;

line perp; /\* HAVE TO BE FILLED perpendicular to l through (x,y) \*/

if (fabs(l.b) <= EPS){ /\* vertical line \*/

p\_c.x = -(l.c); p\_c.y = p\_in.y;

return p\_c;

}

if (fabs(l.a) <= EPS){ /\* horizontal line \*/

p\_c.x = p\_in.x; p\_c.y = -(l.c);

return p\_c;

}

perp = point\_and\_slope\_to\_line(p\_in,1/l.a); /\* normal case \*/

p\_c = intersection(l,perp);

return p\_c;

}

### TRIANGLE CIRCUM CIRCLE

circle circumcircle(Tri t){//the circumcircle of the triangle. needs: dist - Area(Tri)

circle ans;

double a, b, c, c1, c2, xa, ya, xb, yb, xc, yc;

a=dist(t.p[0],t.p[1]);b=dist(t.p[1],t.p[2]); c=dist(t.p[2],t.p[0]); ans.r=.25\*a\*b\*c/Area(t);

xa=t.p[0].x; ya=t.p[0].y; xb=t.p[1].x; yb=t.p[1].y; xc=t.p[2].x; yc=t.p[2].y;

c1=.5\*(xa\*xa+ya\*ya-xb\*xb-yb\*yb); c2=.5\*(xa\*xa+ya\*ya-xc\*xc-yc\*yc);

ans.o.x=(c1\*(ya-yc)-c2\*(ya-yb))/((xa-xb)\*(ya-yc)-(xa-xc)\*(ya-yb));

ans.o.y=(c1\*(xa-xc)-c2\*(xa-xb))/((ya-yb)\*(xa-xc)-(ya-yc)\*(xa-xb));

return ans; };

### POLYGON AREA

double Area(double a, double b, double c){//Area of triangle(Heron)

double p=(a+b+c)/2.0;

return sqrt(p\*(p-a)\*(p-b)\*(p-c));

};

double Area(Tri t){ //Area of triangle with points

point p1=t.p[0], p2=t.p[1], p3=t.p[2];

return .5\*abs((double)(-p2.x\*p1.y+p2.x\*p1.y+p1.x\*p2.y-p3.x\*p2.y-p1.x\*p3.y+p2.x\*p3.y));

};

double signed\_triangle\_area(point a, point b, point c)

{ return( (a.x\*b.y - a.y\*b.x + a.y\*c.x - a.x\*c.y + b.x\*c.y - c.x\*b.y) / 2.0 );}

double Area(polygon p) { // POINTS HAVE TO BE ORDERED

double ans=0; int i;

for (i=0;i<p.ct-1;i++) ans+=(double)(p.p[i].x\*p.p[i+1].y-p.p[i+1].x\*p.p[i].y);

return .5\*abs(ans); };

### TRIANGLE INCIRCLE

circle incircle(Tri t){ //the incircle of the triangle. needs: dist - Area(Tri)

circle ans;

double a, b, c, aa, ab, ac, p, p2, p3, f1, f2;

double xa, ya, xb, yb, xc, yc;

a=dist(t.p[0], t.p[1]); b=dist(t.p[1], t.p[2]); c=dist(t.p[2], t.p[0]);

ans.r=2.0\*Area(t)/(a+b+c);

aa=acos((b\*b+c\*c-a\*a)/(2\*b\*c)); ab=acos((a\*a+c\*c-b\*b)/(2\*a\*c)); ac=acos((a\*a+b\*b-c\*c)/(2\*a\*b));

p=sin(.5\*aa); p2=sin(.5\*ab); p3=sin(.5\*ac);

xa=t.p[0].x; ya=t.p[0].y; xb=t.p[1].x; yb=t.p[1].y; xc=t.p[2].x; yc=t.p[2].y;

f1=.5\*(ans.r\*ans.r/(p2\*p2)-ans.r\*ans.r/(p\*p)+xa\*xa-xb\*xb+ya\*ya-yb\*yb);

f2=.5\*(ans.r\*ans.r/(p3\*p3)-ans.r\*ans.r/(p\*p)+xa\*xa-xc\*xc+ya\*ya-yc\*yc);

ans.o.x=(f1\*(ya-yc)-f2\*(ya-yb))/((xa-xb)\*(ya-yc)-(xa-xc)\*(ya-yb));

ans.o.y=(f1\*(xa-xc)-f2\*(xa-xb))/((ya-yb)\*(xa-xc)-(ya-yc)\*(xa-xb));

return ans;

};

### POINT IN TRIANGLE

bool isInTriangle(point p,Tri t){ //returns true if p is in t, determine by vector product. needs:cross

double k1,k2,k3;

k1=cross(t.p[0],t.p[1],p);

k2=cross(t.p[1],t.p[2],p);

k3=cross(t.p[2],t.p[0],p);

if (k1\*k2\*k3<=EPS){

if (k1\*k2<=EPS) return 0;

if (k1\*k2<=EPS) return 0;

}

return 1;

};

### CONVEX HULL

typedeflonglong int64; // must be big enough to hold 2\*max(|coordinate|)^2

// 2D cross product of OA and OB vectors, i.e. z-component of their 3D cross product.Returns a positive value, if OAB makes a counter-clockwise turn,negative for clockwise turn, and zero if the points are collinear.

int64 cross(const Point &O, const Point &A, const Point &B){

return (A.x - O.x) \* (int64)(B.y - O.y) - (A.y - O.y) \* (int64)(B.x - O.x);

}

// Returns a list of points on the convex hull in counter-clockwise order. the last point in the returned list is the same as the first one. to include on edge point, change <= 0 in both while conditions to < 0.

vector<Point> convex\_hull(vector<Point> P){

int n = P.size(), k = 0;

vector<Point> H(2\*n);

sort(P.begin(), P.end()); // Sort points lexicographically

// Build lower hull

for (int i = 0; i < n; i++) {

while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--; //while points already on hull don't make a left turn with p, pop them out of the hull

H[k++] = P[i]; // push the current point to the hull

}

// Build upper hull

for (int i = n-2, t = k+1; i >= 0; i--) {

while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--;

H[k++] = P[i];

}

H.resize(k);

return H;

}

### BISECTOR

line bisector(line l, line m){

line res;

res.a= sqrt( l.a \* l.a + l.b \* l.b ) \* m.a + sqrt( m.a \* m.a + m.b \* m.b ) \* l.a ;

res.b= sqrt( l.a \* l.a + l.b \* l.b ) \* m.b + sqrt( m.a \* m.a + m.b \* m.b ) \* l.b ;

res.c= sqrt( l.a \* l.a + l.b \* l.b ) \* m.c + sqrt( m.a \* m.a + m.b \* m.b ) \* l.c ;

return res;

}

### POINT + SLOPE -> LINE2

line2( point<T> a , long double s ) {

A = -s; B = 1; C = -( A\*a.x + B\*a.y ); }

### POINT ROTATE AROUND POINT( IN LDBL )

point rotateAround( point<T> o , long double degree ) {

degree = toRadian( degree );

point ans ( x-o.x , y-o.y );

ans=point( ans.x\*cosl( degree ) - ans.y\*sinl( degree ) , ans.x\*sinl( degree ) + ans.y\*cosl( degree ) );

return point( ans.x + o.x , ans.y + o.y );

}

### POINT MIRROR TO LINE( IN LDBL )

point mirrorToLine( line2<T> a ){

point b =footOfPrependicular( a , \*this );

return point( 2\*b.x - x , 2\*b.y - y );//b - ( \*this - b ); }

### POLYGON AREA ( ORDERED POINTS )

long double area( vector< point<T> > a ){

T ans = 0; // if negative, points are clockwise

fr( i , 0 , a.size() ) ans += a[i].cross( a[(i+1)%a.size()] );

return fabs( (long double)ans/2 );// half of an integer

}

### PREPENDICULAR LINE TO LINE FROM GIVEN POINT

line2 prependicularLine( point<T> a ) {

line2( -B , A , -B\*a.x + A\*a.y ); }

### LINE DISTANCE TO POINT

long double distToPoint( point<T> a ) {

return fabls(A\*a.x + B\*a.y + C)/sqrtl( A\*A + B\*B ); }

### X OF GIVEN Y

long double xOfGivenY( T y ) {

return -(long double)( B\*y + C )/A; }

### Y OF GIVEN X

long double yOfGivenX( T x ) {

return -(long double)( A\*x + C )/B; }

### LINE1 TO LINE2 ( TWO POINTS -> AX+BX+C = 0 )

line2 toLine2() {

return line2( p2.y - p1.y, p1.x - p2.x , A\*p1.x + B\*p1.y );}

### DIST TO POINT

long double distToPoint( point<T> a ) {

if( isSegment ){

if( (a-p2).cross(p2-p1) > 0 ) return (p2-a).norm();

if( (a-p1).cross(p1-p2) > 0 ) return (p1-a).norm();

}

return dist = fabsl(((p2 - p1 )^( a - p1))/(p2 - p1).norm() ); }

### CLOSEST POINT ON LINE TO POINT

point<T> closestPointToPoint( point<T> a ) {

if( isSegment ) {

if( (a-p2).cross(p2-p1) > 0 ) return p2;

if( (a-p1).cross(p1-p2) > 0 ) return p1;

}

return footOfPrependicular( toLine2 , a ); }

### LINE INTERSECTS TO LINE ( BOOLEAN )

bool intersects( line1<T> a , line1<T> b ){

return b.p1.turn( b.p2 , a.p1 ) >= 0 && b.p1.turn( b.p2 , a.p2 ) >= 0 && a.p1.turn( a.p2 , b.p1 ) >= 0 && a.p1.turn( a.p2 , b.p2 ) >= 0 ;}

### LINE – LINE INTERSECTION POINT ( LDBL )

point intersectionPoint( line2<T> a , line2<T> b ){

long double det = b.A\*a.B - b.B\*a.A;

if( det == 0 )//infinite/no intersection (parallel lines)

return point( -inf , -inf );

point ans((a.B\*b.C-b.B\*a.C)/det,

(b.A\*a.C -a.A\*b.C)/det);

if( b.isSegment && ! b.isInSegment( ans ) )

return point( -inf , -inf );

if( a.isSegment && ! a.isInSegment( ans ) )

return point<long double>( -inf , -inf );

return ans;}

### FOOT OF PREPENDICULAR

point footOfPrependicular( line2<T> a , point<T> c ){

return intersectionPoint( a , a.prependicularLine( c ) );}

### PREPENDICULAR LINE

line1 prependicularLine( line2<T> a , point<T> c ){

return line1( footOfPrependicular( a , c ) , c );}

### LENGTH OF ARC

long double lengthOfArc( long double degree ) {

return degree/360 \* perimeter; }

### LENGTH OF CHORD

long double lengthOfChord( long double degree ) {

return 2\*r\*r( 1 - cosl(toRadian(degree)) ); }

### 3 POINTS TO CIRCLE( turn(A,B,C)!=0 noncolinear)

circle( point<T> A , point<T> B , point<T> C ) {

line2<T> prep1 = line1<T>( A , B ).toLine2()

.prependicularLine( (A+B)/2.0 );

line2<T> prep2 = line1<T>( B , C ).toLine2()

.prependicularLine( (B+C)/2.0 );

o = intersectionPoint( prep1 , prep2 );

r = ( o - A ).norm();

}

### TRIANGLE AREA

long double s = perimeter/2.0; area = s;

fr( i , 0 , 3 ) area \*= s - sides[i]; area = sqrtl( area );

TRIANGLE IN/OUT CIRCLE RADIUS

radOfInnerCircle = area/s;

radOfOuterCircle = 1;

fr( i , 0 , 3 ) radOfOuterCircle \*= sides[i]; radOfOuterCircle/= 4\*area;

### GREAT CIRCLE DISTANCE

long double greatCircleDistance( long double plat , long double plong , long double qlat , long double qlong , long double r ){

plat = toRadian(plat); plong = toRadian(plong);

qlat = toRadian(qlat); qlong = toRadian(qlong);

return r\* acosl(cosl(plat)\*cosl(plong)\*cosl(qlat)\*cosl(qlong) +

cosl(plat)\*sinl(plong)\*cosl(qlat)\*sinl(qlong) + sinl(plat)\*sin(qlat) ); }

### IS POINT INSIDE POLYGON( ORDERED POINTS)

bool pointInsidePoly(vector<point<T>> poly , point<T> a ){

bool ok = false;

fr( i , 0 , poly.size() ) {

int j = (i+1)%poly.size();

if( min( poly[i].y , poly[j].y ) <= a.y && max( poly[i].y , poly[j].y ) > a.y ) // [ , ) points shouldn’t be repeated

if( line1<T>( poly[i] , poly[j] ).xOfGivenY( a.y ) <= a.x)

ok = !ok;

}

return ok; }

### BISECTOR

line2< long double> bisector(line<T> a, line<T> b){

long double asqr = sqrtl( a.A\*a.A + a.B\*a.B );

long double bsqr = sqrtl( b.A\*b.A + b.B\*b.B );

return line2< long double>( asqr \* b.A + bsqr \* a.A , asqr \* b.B + bsqr \* a.B , asqr \* b.C + bsqr \* a.C ); }