SWERC 2017 Universidad Rey Juan Carlos - RaspuTeam

Matemáticas y Combinatoria

Grafos

Estructura de Datos

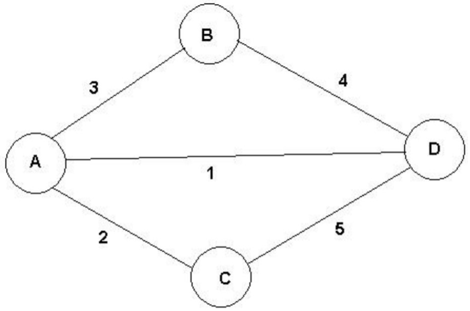
Mix

Algoritmo de Euclides (MCM,LCM) (C++)

```cpp
#include <iostream>
using namespace std;
int gcd(int a, int b) {
while (b > 0) {
int temp = b; b = a % b;   a = temp;   }
    return a; }
int lcm(int a, int b){ return a*(b/gcd(a,b));}
int main() { int numero;
while(1) {
cin>>numero; if(numero!=0) {
int array[numero];
for(int p=0; p<numero;p++)   cin>>array[p];
int final=array[0];
     for(int i=1;i<numero;i++)
     final=lcm(final,array[i]);
cout<<final<<"\n"; }
else    break;    }   return 0;   }
```

Min Cost Max Flow (C++)

```
// IN- graph, constructed using AddEdge(),source, sink
// OUTPUT:(maximum flow value, minimum cost value), to
obtain the actual Flow, look at positive values only.
```



```
4 5
1 4 1
1 3 3
3 4 4
1 2 2
2 4 5
```

```
#include <bits/stdc++.h>
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 = 0x3F3F3F3F;
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); } };
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; }
```

Push Relabel Max Flow (C++)

```
Given a graph undirected, weighted, compute max
Flow/min cut from 1 to N. It solves random problems
with 10000 vertices and 1000000 edges in a few seconds,
though it is possible to construct test cases that
achieve the worst-case. 4 6 | 1 2 3 | 2 3 4 | 3 1 2 | 2
2 5 | 3 4 3 | 4 3 3 -> 5
// IN: graph, constructed using AddEdge() source sink
// OUTPUT: maximum flow value, To obtain the actual
flow values, look at all edges with capacity > 0 (zero
capacity edges are residual edges).
#include <bits/stdc++.h>
using namespace std;
typedef long long LL;
struct Edge {
int from, to, cap, flow, index;
Edge(int from, int to, int cap, int flow, int index) :
from(from), to(to), cap(cap), flow(flow), index(index)
{} };
struct PushRelabel { int N;
vector<vector<Edge> > G; vector<LL> excess;
vector<int> dist, active, count; queue<int> Q;
PushRelabel(int N) : N(N), G(N), excess(N), dist(N),
active(N), count(2*N) {}
void AddEdge(int from, int to, int cap) {
G[from].push_back(Edge(from, to, cap, 0,
G[to].size()));
if (from == to) G[from].back().index++;
G[to].push_back(Edge(to, from, 0, 0, G[from].size() -
1)); }
void Enqueue(int v) {
if (!active[v] && excess[v] > 0) { active[v] = true;
Q.push(v); } }
void Push(Edge &e) {
int amt = int(min(excess[e.from], LL(e.cap - e.flow)));
if (dist[e.from] <= dist[e.to] || amt == 0) return;
e.flow += amt;
G[e.to][e.index].flow -= amt;
excess[e.to] += amt; excess[e.from] -= amt;
Enqueue(e.to); }
void Gap(int k) {
for (int v = 0; v < N; v++) {
if (dist[v] < k) continue;
count[dist[v]]--;
dist[v] = max(dist[v], N+1); count[dist[v]]++;
Enqueue(v); } }
void Relabel(int v) {
count[dist[v]]--; dist[v] = 2*N;
for (int i = 0; i < G[v].size(); i++)
if (G[v][i].cap - G[v][i].flow > 0)
dist[v] = min(dist[v], dist[G[v][i].to] + 1);
count[dist[v]]++; Enqueue(v); }
void Discharge(int v) {
for (int i = 0; excess[v]>0&&i<G[v].size();i++)
Push(G[v][i]);
if (excess[v] > 0) {
if (count[dist[v]] == 1) Gap(dist[v]);
else Relabel(v);} }
LL GetMaxFlow(int s, int t) {
count[0] = N-1; count[N] = 1; dist[s] = N;
active[s] = active[t] = true;
for (int i = 0; i < G[s].size(); i++) {
excess[s] += G[s][i].cap;
Push(G[s][i]); }
while (!Q.empty()) {
int v = Q.front();
Q.pop(); active[v] = false;
Discharge(v); }
LL totflow = 0;
for (int i = 0; i < G[s].size(); i++) totflow +=
G[s][i].flow; return totflow; } };
// SPOJ problem #4110: Fast Maximum Flow
int main() { int n, m;
scanf("%d%d", &n, &m); PushRelabel pr(n);
for (int i = 0; i < m; i++) {
int a, b, c; scanf("%d%d%d", &a, &b, &c);
if (a == b) continue;
pr.AddEdge(a-1, b-1, c); pr.AddEdge(b-1, a-1, c);}
printf("%Ld\n", pr.GetMaxFlow(0, n-1)); return 0;}
```

Min Cost Bipartite Matching (C++)

```cpp
// Algorithm for finding min cost perfect matchings in
dense graphs. In practice, it solves 1000x1000 problems
in around 1 second. cost[i][j] = cost for pairing left
node i with right node j. Lmate[i] = index of right
node that left node i pairs with. Rmate[j] = index of
left node that right node j pairs with
// The values in cost[i][j] may be positive or
negative. To perform
// maximization, simply negate the cost[][] matrix.
//Find a minimum cost matching between S and T among
all those with maximum cardinality.
#include <bits/stdc++.h>
using namespace std;
typedef vector<double> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;
double MinCostMatching(const VVD &cost, VI &Lmate, VI
&Rmate) {
int n = int(cost.size());
VD u(n); VD v(n);
for (int i = 0; i < n; i++) {u[i] = cost[i][0];
for (int j = 1; j < n; j++) u[i] = min(u[i],
cost[i][j]); }
for (int j = 0; j < n; j++) {
v[j] = cost[0][j] - u[0];
for (int i = 1; i < n; i++) v[j] = min(v[j], cost[i][j]
- u[i]); }
Lmate = VI(n, -1); Rmate = VI(n, -1);
int mated = 0;
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
if (Rmate[j] != -1) continue;
if (fabs(cost[i][j] - u[i] - v[j]) < 1e-10) {
Lmate[i] = j; Rmate[j] = i; mated++;
break; } } }
VD dist(n); VI dad(n); VI seen(n);
while (mated < n) { int s = 0;
while (Lmate[s] != -1) s++;
fill(dad.begin(), dad.end(), -1);
fill(seen.begin(), seen.end(), 0);
for (int k = 0; k < n; k++)
dist[k] = cost[s][k] - u[s] - v[k];
int j = 0;
while (true) { j = -1;
for (int k = 0; k < n; k++) {
if (seen[k]) continue;
if (j == -1 || dist[k] < dist[j]) j = k; }
seen[j] = 1;
if (Rmate[j] == -1) break;
const int i = Rmate[j];
for (int k = 0; k < n; k++) {
if (seen[k]) continue;
const double new_dist = dist[j] +
cost[i][k] - u[i] - v[k];
if (dist[k] > new_dist) {
dist[k] = new_dist;
dad[k] = j; } } }
for (int k = 0; k < n; k++) {
if (k == j || !seen[k]) continue;
const int i = Rmate[k];
v[k] += dist[k] - dist[j];
u[i] -= dist[k] - dist[j]; }
u[s] += dist[j];
// augment along path
while (dad[j] >= 0) {
const int d = dad[j];
Rmate[j] = Rmate[d]; Lmate[Rmate[j]] = j;
j = d; }
Rmate[j] = s; Lmate[s] = j; mated++; }
double value = 0;
for (int i = 0; i < n; i++)
value += cost[i][Lmate[i]];
return value;
}
```

Max bipartite Matching (C++)

```
//IN: w[i][j]=edge between row node i and column node j
//OP: mr[i]=assignment for row node i, -1 if unassigned
//mc[j] = assignment for column node j, -1 if
unassigned function returns number of matches made
#include <vector>
using namespace std;
typedef vector<int> VI;
typedef vector<VI> VVI;
bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI
&seen) {
for (int j = 0; j < w[i].size(); j++) {
if (w[i][j] && !seen[j]) {
seen[j] = true;
if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
mr[i] = j; mc[j] = i;
return true; } } }
return false; }
int BipartiteMatching(const VVI &w, VI &mr, VI &mc) {
mr = VI(w.size(), -1);
mc = VI(w[0].size(), -1);
int ct = 0;
for (int i = 0; i < w.size(); i++) {
VI seen(w[0].size());
if (FindMatch(i, w, mr, mc, seen)) ct++; }
return ct; }
```

Módulos, Teorema Chino del Resto, etc (C++)

```
#include <bits/stdc++.h>
using namespace std;
typedef vector<int> VI;
typedef pair<int, int> PII;
// return a % b (positive value)
int mod(int a, int b) {
return ((a%b) + b) % b; }
int gcd(int a, int b) {
while (b) { int t = a%b; a = b; b = t; }
return a; }
int lcm(int a, int b) {
return a / gcd(a, b)*b; }
// (a^b) mod m via successive squaring
int powermod(int a, int b, int m) {
int ret = 1;
while (b) {
if (b & 1) ret = mod(ret*a, m);
a = mod(a*a, m); b >>= 1; }
return ret; }
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; }
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); }
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); }
PII chinese_remainder_theorem
(const VI &m, const VI &r) {
PII ret = make_pair(r[0], m[0]);
for (int i = 1; i < m.size(); i++) {
```

```
ret = chinese_remainder_theorem(ret.second,
ret.first, m[i], r[i]);
if (ret.second == -1) break; }
return ret; }
// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x,
int &y) {
if (!a && !b) {
if (c) return false; x = 0; y = 0; return true; }
if (!a) { if (c % b) return false;
x = 0; y = c / b; return true;}
if (!b) { if (c % a) return false;
x = c / a; y = 0; return true; }
int g = gcd(a, b);
if (c % g) return false;
x = c / g * mod_inverse(a / g, b / g);
y = (c - a*x) / b; return true; }
int main() {
cout << gcd(14, 30) << endl;
// expected: 2 -2 1 int x, y;
int g = extended_euclid(14, 30, x, y);
cout << g << " " << x << " " << y << endl;
// expected: 95 451
VI sols = modular_linear_equation_solver(14, 30, 100);
for (int i = 0; i < sols.size(); i++) cout << sols[i]
<< " ";
cout << endl;
// expected: 8
cout << mod_inverse(8, 9) << endl;
// expected: 23 105
// 11 12
PII ret = chinese_remainder_theorem(VI({ 3, 5, 7 }),
VI({ 2, 3, 2 }));
cout << ret.first << " " << ret.second << endl;
ret = chinese_remainder_theorem(VI({4,6}), VI({3,5 }));
cout << ret.first << " " << ret.second << endl;// 5 -15
if (!linear_diophantine(7,2,5, x, y)) cout << "ERROR\n"
cout << x << " " << y << endl;
return 0; }
```

Sistemas de ecuaciones lineales, inversa, etc (C++)

```
// Gauss-Jordan elimination with full pivoting.
// (1) solving systems of linear equations (AX=B)
// (2) inverting matrices (AX=I)
// (3) computing determinants of square matrices
// 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 <bits/stdc++.h>
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; }}
```

Simplex Algorithm (C++)

```
// 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
// arguments. Then, call Solve(x).
#include <bits/stdc++.h>
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; }
```

Números Catalanes (C++)

```
// 1 1 2 5 14 42 132 429 1430 4862 16796 58786
#include<iostream>
using namespace std;
unsigned long int catalanDP(unsigned int n){
    unsigned long int catalan[n+1];
     catalan[0] = catalan[1] = 1;
    for (int i=2; i<=n; i++){
        catalan[i] = 0;
        for (int j=0; j<i; j++)
            catalan[i]+=catalan[j]*catalan[i-j-1];
    } return catalan[n]; }
int main() {
    for (int i = 0; i < 10; i++)
     cout << catalanDP(i) << " "; return 0; }
```

Números Eulerianos (C++)

```
// C++ program to  print all increasing sequences of
// length 'k' such that the elements in every
sequence are from first 'n' natural numbers.
// Input: k = 2, n = 3 Output: 1 2 | 1 3 | 2 3
#include<iostream>
using namespace std;
void printSeqUtil(int n, int k, int &len, int
arr[]){
    if (len == k) {
        printArr(arr, k); return;  }
    int i = (len == 0)? 1 : arr[len-1] + 1;
    len++;
    while (i<=n)  {
        arr[len-1] = i;
        printSeqUtil(n, k, len, arr);
        i++; } len--; }
void printSeq(int n, int k) {
    int arr[k];   int len = 0;
    printSeqUtil(n, k, len, arr); }
int main() {
    int k = 3, n = 7; printSeq(n, k);  return 0; }
```

Karatsuba (C++)

```
#include<bits/stdc++.h>
using namespace std;
int makeEqualLength(string &str1, string &str2){
    int len1 = str1.size();
    int len2 = str2.size();
    if (len1 < len2){
        for (int i = 0 ; i < len2 - len1 ; i++)
            str1 = '0' + str1;
        return len2;}
    else if (len1 > len2) {
        for (int i = 0 ; i < len1 - len2 ; i++)
            str2 = '0' + str2; }
    return len1;} // If len1 >= len2
string addBitStrings( string first, string second ){
    string result;  // To store the sum bits
    // make the lengths same before adding
    int length = makeEqualLength(first, second);
    int carry = 0;  // Initialize carry
    for (int i = length-1 ; i >= 0 ; i--) {
        int firstBit = first.at(i) - '0';
        int secondBit = second.at(i) - '0';
        int sum = (firstBit ^ secondBit ^ carry)+'0';
        result = (char)sum + result;
        carry = (firstBit&secondBit) |
(secondBit&carry) | (firstBit&carry);  }
    if (carry)  result = '1' + result;
    return result; }
int multiplyiSingleBit(string a, string b)
{  return  (a[0] - '0')*(b[0] - '0');   }
long int multiply(string X, string Y) {
    int n = makeEqualLength(X, Y);
    if (n == 0) return 0;
    if (n == 1) return multiplyiSingleBit(X, Y);
    int fh = n/2; int sh = (n-fh);
    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);
    string Yl = Y.substr(0, fh);
    string Yr = Y.substr(fh, sh);
    long int P1 = multiply(Xl, Yl);
    long int P2 = multiply(Xr, Yr);
    long int P3 = multiply(addBitStrings(Xl, Xr),
addBitStrings(Yl, Yr));
    return P1*(1<<(2*sh)) +
(P3 - P1 - P2)*(1<<sh) + P2; }
// Driver program to test aboev functions - 120
int main() {
    printf ("%ld\n", multiply("1100", "1010")); }
```

Binomial Coeficiente (C++)

```
//n! mod p
#include <bits/stdc++.h>
using namespace std;
// Utility function to do modular exponentiation.
// It returns (x^y) % p
int power(int x, unsigned int y, int p)
{ int res = 1;      // Initialize result
    x = x % p;     while (y > 0)  {
   // If y is odd, multiply x with result
    if (y & 1)
     res = (res*x) % p;
        y = y>>1; // y = y/2
        x = (x*x) % p; }
    return res; }
int modInverse(int a, int p) {
   return power(a, p-2, p); }
// Returns n! % p using Wilson's Theorem
int modFact(int n, int p) {
    if (p <= n)
        return 0;     int res = (p-1);
    for (int i=n+1; i<p; i++)
       res  = (res * modInverse(i, p)) % p;
    return res; }
int main() {
    int n = 25, p = 29;
    cout << modFact(n, p);
    return 0; }
```

Integración por Simpson (C++)

$$\int_a^b f(x)dx$$

```
#include <bits/stdc++.h>
double Simpson(double a, double b, int k, double
(*f)(double)){
  double dx, x, t=0; int i;
  dx = (b-a)/(2.0*k);
  for( i=0; i<k; i++ ) {
    t += (i==0 ? 1.0 : 2.0) * (*f)(a+2.0*i*dx);
    t += 4.0 * (*f)(a+(2.0*i+1.0)*dx);}
  t += (*f)(b);
  return t * (b-a)/6.0/k; }
double example_function(double x) {
  return x*x; }
int main(void){
  int k;  double a = 0, b=5.0;
  printf("Integral from %f to %f is:\n",a,b);
  for( k=1; k<=40; k++ ) {
    printf("  k = %3d:
%f\n",k,Simpson(a,b,k,example_function)); }
  return 0; }
```

Módulo en factorial (C++)

```
// n! % p using Wilson's Theorem
#include <bits/stdc++.h>
using namespace std;
int power(int x, unsigned int y, int p) {
    int res = 1;  x = x % p;
    while (y > 0) {
        if (y & 1) res = (res*x) % p;
        y = y>>1; x = (x*x) % p; }
    return res;}
Assumption: p is prime
int modInverse(int a, int p) {
   return power(a, p-2, p); }
int modFact(int n, int p) {
    // n! % p is 0 if n >= p
    if (p <= n)   return 0;
    int res = (p-1);
    for (int i=n+1; i<p; i++)
       res  = (res * modInverse(i, p)) % p;
    return res; }
int main() {
    int n = 25, p = 29;
    cout << modFact(n, p);
    return 0; }
```

Fubini Numbers (C++)

//0, 1, 1, 3, 13, 75, 541, 4683, 47293, 545835

```
#include<bits/stdc++.h>
using namespace std;
#define LL long long #define nmax 3100
#define nnum 46337
LL num[nmax][nmax], fac[nmax];
void init() {    int i, j;
    for (i = 1, fac[0] = 1; i < nmax; i++) {
        fac[i] = fac[i - 1] * i % nnum;    }
    for (i = 1; i < nmax; i++) {
        num[i][1] = 1; num[i][0] = 0; }
    for (i = 2; i < nmax; i++) {
        for (j = 1; j < nmax; j++) {
            if (i == j) {
                num[i][i] = 1;
            } else {
 num[i][j] = (num[i - 1][j - 1] + num[i - 1]
[j] * j) % nnum;   } }  } }
int main() { ofstream myfile;
    myfile.open ("example.txt");
    init(); int  N=0, i; LL res;
    while (N!=3050) {
      for (i = 1, res = 0; i <= N; i++) {
       res += num[N][i] * fac[i];   res %= nnum;}
     myfile<<res<<","; printf("%I64d\n", res);
          N++; } myfile.close();  return 0; }
```

Dijkstra (C++)

Pesos no negativos

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
struct edge{
  int from, to, weight;
  edge(){}
  edge(int a, int b, int c){
    from = a;   to = b; weight = c; }};
struct state{
  int node, dist; state(){}
  state(int a, int b){
    node = a; dist = b;   }
  bool operator<(const state &other)const{
    return other.dist < other.dist; } };
vector<edge> graph[MAXN];
int dist[MAXN]; int a=1, b=3; int N,E;
int dijkstra(int start, int end){
  dist[start] = 0;
  priority_queue<state> pq;
  pq.push(state(start, 0));
  while(!pq.empty()){
    state cur = pq.top(); pq.pop();
    if(dist[cur.node] < cur.dist) continue;
    if(cur.node == end) return cur.dist;
    for(int i=0;i<graph[cur.node].size();i++){
      int dest = graph[cur.node][i].to;
      int wht = graph[cur.node][i].weight + cur.dist;
      if(dist[dest] <= wht) continue;
      dist[dest] = wht;
      pq.push(state(dest, wht));
    } } return -1; }
int main(){
  freopen("dijkstra.in","r",stdin);
        scanf("%d %d",&N,&E);
  memset(dist,0x3f,sizeof(dist));
  for(int i=1;i<=N;i++) graph[i].clear();
        for(int i=0;i<E;i++){
    int from, to, weight; scanf("%d %d %d",&from,
&to, &weight);
    graph[from].push_back(edge(from,to,weight));
    graph[to].push_back(edge(to, from, weight));
    // borrar linea si es dirigido}
  printf("Dijkstra d%d a %d es %d\n",a,b,
dijkstra(a,b));
  return 0; }}
```

Kruskal (C++)

```
// El algoritmo de Kruskal calcula el tamaño
minimo de un bosque es decir union de arboles
cada uno conectado a una componente, posibilitando
una matriz de adyacencia dada una matriz con peso
en los nodos donde -1 es si no existe el vertice.
Devuelve el minimo peso del bosque calculando los
vertices guardados en T, usa un arbol disjunto para
amortizar la efectividad en un tiempo contante
siendo la complejidad O(E*log(E))
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
struct edge{
  int from, to, weight; edge(){}
  edge(int a, int b, int c){
    from = a;  to = b;  weight = c;}
  bool operator<(const edge &other)const{
    return weight < other.weight;} };
struct UF{
    int parents[MAXN]; int sz[MAXN];
    int components;   int mst_sum;
    UF(int n){
        for(int i=0;i<n;i++){
            parents[i] = i; sz[i] = 1;   }
        components = n; mst_sum = 0; }
    int find(int n){
        return n==parents[n] ? n : find(parents[n]);
    }
    bool isConnected(int a, int b){
        return find(a) == find(b); }
    void connect(int a, int b, int weight){
        if(isConnected(a, b)) return;
        int A,B; A = find(a); B = find(b);
        if(sz[A] > sz[B]){
            parents[B] = A; sz[A] += sz[B];   }
        else{ parents[A] = B; sz[B] += sz[A];
        } mst_sum += weight; components--; } };
int a=1; int N,E;
int main(){
  freopen("mst.in","r",stdin);
        scanf("%d %d",&N,&E);
  vector<edge> edges;
  UF uf = UF(N);
        for(int i=0;i<E;i++){
    int from, to, weight; scanf("%d %d %d",&from, &to,
&weight);
    edges.push_back(edge(from,to,weight));}
  sort(edges.begin(), edges.end());
  for(int i=0;i<E;i++) uf.connect(edges[i].from,
edges[i].to, edges[i].weight);
  printf("Kruskal de %d es %d\n",a,uf.mst_sum);
  return 0; }
```

DFS (Componentes Conexas, Ciclos) y BFS (C++)

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
vector<int> graph[MAXN];
bool visited[MAXN];
int a=1, b=6; int N,E;
int DFS(int node, int target){
  if(node == target) return 0;
  if(visited[node]) return INF;
  visited[node] = true;
  int best_result = INF;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    best_result = min(
        best_result,
        DFS(dest, target)+1  );}
  return best_result; }
int BFS(int start, int target){
  queue<pair<int,int> > q;
  q.push(make_pair(start,0));
  visited[start] = true;
  while(!q.empty()){
    pair<int,int> current = q.front(); q.pop();
    if(current.first == target) return
current.second;
    for(int i=0;i<graph[current.first].size();i++){
      int dest = graph[current.first][i];
      if(visited[dest]) continue;
      visited[dest] = true;
      q.push(make_pair(dest,current.second+1));
    } }
  return -1; }

int main(){
  freopen("recorridos.in","r",stdin);
        scanf("%d %d",&N,&E);
  for(int i=1;i<=N;i++) graph[i].clear();
        for(int i=0;i<E;i++){
    int from, to; scanf("%d %d",&from, &to);
    graph[from].push_back(to);
    graph[to].push_back(from);
// borrar linea si es dirigido }
  memset(visited,0,sizeof(visited));
  printf("BFS de %d a %d es %d\n",a,b,BFS(a,b));
  memset(visited,0,sizeof(visited));
  printf("DFS de %d a %d es %d\n",a,b,DFS(a,b));
  return 0; }
```

```
// Contar ciclos
bool DFS(int node, int parent){
  if(visited[node])   return false;
  visited[node] = true;  bool res = true;
  for(unsigned int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    if(dest==parent && visited[dest])
        return true;
    if(visited[dest])  continue;
    res = DFS(dest,node); }
  return res; }

// Componentes Conexas
void DFS(int node){
  if(visited[node]) return;
  visited[node] = true;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i];
    DFS(dest);   } }
int main(){
  memset(visited,0,sizeof(visited));
  int comps = 0;
  for(int i=1;i<=N;i++){
    if(!visited[i]){ DFS(i); comps++; } }
  printf("Hay %d componentes\n", comps);
  return 0; }
```

Eulerian Path (C++)

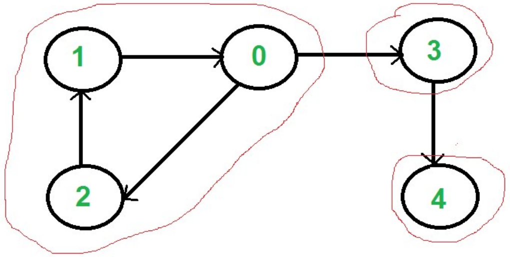
```
#include<bits/stdc++.h>
using namespace std;
class Graph {
    int V; list<int> *adj;
public:
    Graph(int V)   {this->V = V; adj = new
list<int>[V]; }
    ~Graph() { delete [] adj; }
    void addEdge(int v, int w);
    int isEulerian();
    bool isConnected();
    void DFSUtil(int v, bool visited[]); };

void Graph::addEdge(int v, int w) {
    adj[v].push_back(w);
    adj[w].push_back(v); }
void Graph::DFSUtil(int v, bool visited[]) {
    visited[v] = true;
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFSUtil(*i, visited); }
bool Graph::isConnected() {
    bool visited[V];   int i;
    for (i = 0; i < V; i++)
        visited[i] = false;
    for (i = 0; i < V; i++)
        if (adj[i].size() != 0)  break;
    if (i == V)   return true;
    DFSUtil(i, visited);
    for (i = 0; i < V; i++)
       if (visited[i] == false && adj[i].size() > 0)
            return false;
    return true; }
// The function returns one of the following values
// 0 --> If graph is not Eulerian
// 1 --> If graph has an Euler path (Semi-Eulerian)
// 2 --> If graph has an Euler Circuit (Eulerian)
int Graph::isEulerian() {
    if (isConnected() == false) return 0;
    int odd = 0;
    for (int i = 0; i < V; i++)
        if (adj[i].size() & 1)
            odd++;
    if (odd > 2)
        return 0;
    // If odd count is 2, then semi-eulerian.
    // If odd count is 0, then eulerian
    return (odd)? 1 : 2; }
void test(Graph &g) {
    int res = g.isEulerian();
```

```
  if (res == 0)
      cout << "graph is not Eulerian\n";
  else if (res == 1)
      cout << "graph has a Euler path\n";
  else
      cout << "graph has a Euler cycle\n"; }
int main() {
  Graph g1(5); g1.addEdge(1, 0); g1.addEdge(0, 2);
  g1.addEdge(2, 1); g1.addEdge(0, 3);
  g1.addEdge(3, 4); test(g1);

  Graph g3(5); g3.addEdge(1, 0);  g3.addEdge(0, 2);
  g3.addEdge(2, 1); g3.addEdge(0, 3);
  g3.addEdge(3, 4); g3.addEdge(1, 3);  test(g3);
  // connected in the form of cycle
  Graph g5(3);    test(g5);
return 0; }
/* graph has a Euler path | graph is not Eulerian
graph has a Euler cycle */
```

Strongly Connected Components (C++)



```
/* 4 4 | 1 2 | 3 2 | 4 3 | 2 1
Output: 2 |  1 2 */
#include <bits/stdc++.h>
using namespace std;
const int MAX = 100005; int N, M;
int componentCount[2];
int componentID[2][MAX]; int degree[MAX];
bool visited[2][MAX];
```

```
vector<int> orders;
vector<int> Graph[MAX], reverseGraph[MAX];
  void dfs1 (int node) {visited[0][node] = true;
    componentID[0][node] = componentCount[0];
    for (int i : reverseGraph[node]) {
      if (!visited[0][i])  dfs1(i);  }
    orders.push_back(node); }
  void dfs2 (int node) {
    visited[1][node] = true;
    componentID[1][node] = componentCount[1];
    for (int i : Graph[node])
      if (!visited[1][i])   dfs2(i); }
  int main() {  scanf("%d%d", &N, &M);
    for (int i = 0; i < M; i ++) {int u, v;
      scanf("%d%d", &u, &v);
      Graph[u].push_back(v);
      reverseGraph[v].push_back(u); }
    for (int i = 1; i <= N; i ++) {
      if (!visited[0][i]) {
        componentCount[0] ++; dfs1(i); } }
    reverse(orders.begin(), orders.end());
    for (int i : orders) {
      if (!visited[1][i]) {
        componentCount[1] ++; dfs2(i); } }
    for (int i = 1; i <= N; i ++)
      for (int j : reverseGraph[i])
 if (componentID[1][i] != componentID[1][j])
  degree[componentID[1][j]] ++; int startings = 0;
   for (int i = 1; i <= componentCount[1]; i ++)
      if (!degree[i]) startings ++;
    if (startings > 1) puts("0");
    else {  vector<int> output;
      for (int i = 1; i <= N; i ++)
        if (!degree[componentID[1][i]])
          output.push_back(i);
      printf("%d\n", (int)output.size());
   for (int i = 0; i < (int)output.size(); i ++)
    {          printf("%d", output[i]);
   if (i < (int)output.size() - 1) putchar(' ');
   } putchar('\n');} return 0;}
```

Floyd Warshall (C++)

```
#include<bits/stdc++.h>
using namespace std; //No tiene porque ser simétrica!
#define INF 0x3F3F3F3F
   int dist[4][4] = { {  0, 5  ,INF,10},
                      {INF, 0  ,3  ,INF},
                      {INF, INF,0  ,1},
                      {INF, INF,INF,0}};
void floydWarshall ();
{
    int dist[4][4], i, j, k;
    for (i = 0; i < 4; i++)
        for (j = 0; j < 4; j++)
            dist[i][j] = graph[i][j];
    for (k = 0; k < 4; k++)
        for (i = 0; i < 4; i++)
            for (j = 0; j < 4; j++)
    dist[i][j]=min(dist[i][j],dist[i][k]+dist[k][j]);}
int main(){ floydWarshall();return 0;}
```

Prim (C++)

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
struct edge{
  int from, to, weight;
  edge(){}
  edge(int a, int b, int c){
    from = a; to = b; weight = c; } };
struct state{
  int node, dist;
  state(){}
  state(int a, int b){
    node = a; dist = b; }
  bool operator<(const state &other)const{
    return other.dist < dist; } };

vector<edge> graph[MAXN];
bool visited[MAXN];
int a=1; int N,E;
int prim(int start){
  priority_queue<state> pq;
  pq.push(state(start, 0));
  int sum = 0;
  while(!pq.empty()){
    state cur = pq.top(); pq.pop();
    if(visited[cur.node]) continue;
    sum += cur.dist;
    visited[cur.node] = true;
    for(int i=0;i<graph[cur.node].size();i++){
      int dest = graph[cur.node][i].to;
      int wht = graph[cur.node][i].weight;
      if(visited[dest]) continue;
      pq.push(state(dest, wht));
    } } return sum; }
int main(){
  freopen("mst.in","r",stdin);
        scanf("%d %d",&N,&E);
  memset(visited,0,sizeof(visited));
  for(int i=1;i<=N;i++) graph[i].clear();
        for(int i=0;i<E;i++){
    int from,to,weight; scanf("%d %d %d",&from,&to,
 &weight);
    graph[from].push_back(edge(from,to,weight));
    graph[to].push_back(edge(to, from, weight));
// borrar linea si es dirigido }
  printf("Prim de %d es %d\n",a,prim(a));
  return 0;}
```

BellMan Ford (C++)

```
#include<bits/stdc++.h>
#define N 2001
#define MAX 100000000
using namespace std;
int a[N], b[N], t[N];
bool BellmanFord(int n, int m) {
    int d[N];
    fill(d, d + n, MAX);   d[0] = 0;
    //bellman ford
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < m; j++)
            if (d[a[j]] != MAX)
                if (d[a[j]] + t[j] < d[b[j]])
                    d[b[j]] = d[a[j]] + t[j];
    //negative cycle check
    for (int j = 0; j < m; j++)
        if (d[a[j]] + t[j] < d[b[j]])
            return true;  return false; }
int main() {
    int Case, n, m;
    scanf("%d", &Case);
    while (Case--)   { int i;   scanf("%d%d", &n, &m);
        for (i = 0; i < m; i++)
            scanf("%d%d%d", &a[i], &b[i], &t[i]);
        puts(BellmanFord(n, m) ? "possible" : "not
possible");   }  return 0; }
//Sample Input 2 - 3 3 | 0 1 1000 | 1 2 15 | 2 1 -42 |
4 4 |0 1 10 |1 2 20 |2 3 30 |3 0 -60 Sample Output
posible | not possible
```

Topological Sort (C++)

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 100100;
const int INF = 0x3f3f3f3f;
vector<int> graph[MAXN];
bool visited[MAXN];
stack<int> topological_order;
int N,E;
void DFS(int node){
  if(visited[node]) return;
  visited[node] = true;
  for(int i=0;i<graph[node].size();i++){
    int dest = graph[node][i]; DFS(dest); }
  topological_order.push(node); }
int main(){
  freopen("topological.in","r",stdin);
        scanf("%d %d",&N,&E);
  for(int i=1;i<=N;i++) graph[i].clear();
        for(int i=0;i<E;i++){
    int from, to; scanf("%d %d",&from, &to);
    graph[from].push_back(to); }
  printf("Ordenamiento topologico:");
  //Asumo que el nodo 1 no es dependiente de nadie
  DFS(1);
  while(!topological_order.empty()){
    printf(" %d",topological_order.top());
    topological_order.pop(); }
  printf("\n"); return 0; }
```

Puntos de articulación (C++)

```
#include <bits/stdc++.h>
    #define oo 1000
    int link[100][100], n;
    int depth[100], low[100];
    int used[100], cut;
    int DFS(int node, int d, int parent) {
        int i, back = oo, son = 0, tmp, flag = 0;
        depth[node] = d;
        for(i = 1; i <= n; i++) {
            if(link[node][i] == 1) {
                if(used[i] == 0) {
                    used[i] = 1;
                    tmp = DFS(i, d+1, node);
                    if(tmp >= d)      flag = 1;
                back = back < tmp ? back : tmp;
                    son++;
                } else {
                    if(i != parent)
      back = back < depth[i] ? back : depth[i];
                } } }
        low[node] = back;
        if(node == 1)  if(son > 1) cut++;
        else  cut += flag;
        return low[node]; }
    int main() {
        int x, y; char c;
while(scanf("%d", &n) == 1 && n) {
    memset(link, 0, sizeof(link));
    memset(depth, 0, sizeof(depth));
    memset(low, 0, sizeof(low));
    memset(used, 0, sizeof(used));
    while(scanf("%d", &x) == 1 && x) {
        while(scanf("%d%c", &y, &c) == 2) {
            link[x][y] = 1;
            link[y][x] = 1;
            if(c == '\n') break; } }
    used[1] = 1; cut = 0;
    DFS(1, 1, 0); printf("%d\n", cut); }
return 0; }
```

Priority Queue (C++)

```
#include <bits/stdc++.h>
using namespace std;
struct cosa{
 int weight; int modificado; int inicial;
 bool operator<(const cosa &other)const{ return
weight < other.weight; } };
int main() { priority_queue<cosa> pq;
        for(int i=0;i<cases;i++) {
            cin>>numer;
            pq.push(cosa{numer,2,numer}); }
        cosa t1 = pq.top();  pq.pop();
    return 0; }
```

Lazy Segment Tree (C++)

```
#include <bits/stdc++.h>
#define MAX 100000
using namespace std;
struct sum {
    long long int msum;
    long long int m;};
int array1[ MAX + 1 ];
sum tree[ 4 * MAX + 1 ];
void init( int node, int i, int j ) {
    if ( i == j )
        tree[node] = ((sum) {array1[i],array1[i]});
    else {
        init( node * 2, i, ( i + j ) / 2 );
        init( node * 2 + 1, ( i + j ) / 2 + 1, j );
        sum left = tree[ node * 2 ],
right = tree[ node*2+1 ];
        tree[ node ].msum = max( left.msum,
 max( right.msum, left.m + right.m ) );
        tree[ node ].m = max( left.m, right.m );}}
sum query( int node, int a, int b, int i, int j ) {
    if ( a > b || a > j || b < i )
        return ( ( sum ) { 0, 0 } );
    if ( a >= i && b <= j )
        return tree[ node ];
    sum left= query( node*2,a, (a+b)/2, i, j );
   sum right= query( node*2+1, (a+b)/2+1, b, i, j );
    return ( ( sum ) {
                max(left.msum, max( right.msum,
left.m + right.m ) ),
                max( left.m, right.m ) } ); }
void update(int node,int a,int b,int pos,int val)
{    if ( a == b && a == pos ) {
        tree[ node ] = ( ( sum ) { val, val } );
        return;  }
    if ( pos <= ( a + b ) / 2 ) {
        update( node * 2, a, (a+b)/2, pos, val );
    }
    if ( pos > (a+b)/2) {
        update( node*2+1, (a+b)/2+1,b,pos,val);}
    sum left = tree[node*2], right=tree[node*2+1];
    tree[node].msum =
max(left.msum, max(right.msum,left.m+right.m ) );
    tree[ node ].m = max( left.m, right.m ); }

int main() {
    int N, Q, l, r, i;
    char c;
    scanf( "%d", &N );
    for ( i = 0; i < N; ++i ) {
        scanf( "%d", array1 + i );
    }
    init( 1, 0, N - 1 );
    scanf( "%d", &Q );
    for ( i = 0; i < Q; ++i ) {
        scanf( "%*c%c%d%d", &c, &l, &r );
        if ( c == 'U' )
            update( 1, 0, N - 1, l - 1, r );
        else
            printf( "%lld\n", query( 1, 0, N - 1, l -
1, r - 1 ).msum );    } return 0; }
/* Input:
5
1 2 3 4 5
6
Q 2 4 | Q 2 5 | U 1 6 | Q 1 5 | U 1 7 | Q 1 5
Output: 7, 9, 11, 12
```

Last Common acestor (C++)

```
#include <bits/stdc++.h>
#define FOR(i, a, b) for (int i=a; i<=b; i++)
#define REP(i, n) for (unsigned int i=0; i<n; i++)
#define Fill(ar, val) memset(ar, val, sizeof(ar))
#define pb push_back
#define bit(n) (1<<(n))
#define maxN 5005
using namespace std;
int n, T[maxN], L[maxN], P[maxN][20];
vector<int> adj[maxN];
void build_tree(int u, int p, int lvl) {
    if (T[u]) return;
    T[u] = p;
    L[u] = lvl;
    REP(i, adj[u].size())
        build_tree(adj[u][i], u, lvl + 1); }
void build_array() {
    FOR (i, 1, n) {
        P[i][0] = T[i];
        for (int j = 1; bit(j) < n; j++)
            P[i][j] = -1;
    }
    P[1][0] = -1;
    for (int j = 1; bit(j) < n; j++)
        FOR (i, 1, n)
            if (P[i][j - 1] != -1)
                P[i][j] = P[ P[i][j - 1] ][j - 1];}
int LCA(int u, int v) {
    int Log = 0;
    for (; bit(Log) <= L[u]; Log++);
    Log--;
    for (int i = Log; i >= 0; i--)
        if (L[u] - bit(i) >= L[v])
            u = P[u][i];
    if (u == v) return u;
    for (int i = Log; i >= 0; i--)
        if (P[u][i]!= -1 && P[u][i] != P[v][i])
            u = P[u][i], v = P[v][i];
    return T[u]; }
int ancester(int u, int length) {
    int Log = 0;
    for (; bit(Log) <= L[u]; Log++);
    Log--;
    for (int i = Log; i >= 0; i--)
        if (length - bit(i) >= 0)
            u = P[u][i], length -= bit(i);
    return u; }

void solve(int u, int v) {
    if (L[u] < L[v]) swap(u, v);
    int lca_level = L[LCA(u, v)];
    int length = L[u] + L[v] - (lca_level << 1);
    if (length % 2) {
        u = ancester(u, length >> 1);
        v = T[u];
        if (u > v) swap(u, v);
        printf("The fleas jump forever
between %d and %d.\n", u, v);
    }
    else printf("The fleas meet at %d.\n",
ancester(u, length >> 1)); }
```

```
int main() {
    int m, u, v;
    while (scanf("%d", &n) && n) {
        FOR (i, 1, n) {
            adj[i].clear();
            T[i] = 0;
        }
        FOR (i, 2, n) {
            scanf("%d %d", &u, &v);
            adj[u].pb(v);
            adj[v].pb(u);
        }
        build_tree(1, n + 1, 0);
        build_array();
        scanf("%d", &m);
        while (m--) {
            scanf("%d %d", &u, &v);
            solve(u, v);
        }  } }
8 | 1 2 | 1 3 | 2 4 | 2 5 | 3 6 | 3 7 | 5 8
5 | 5 1 | 7 4 | 1 8 | 4 7 | 7 8
The fleas meet at 2. The fleas meet at 1.
The fleas jump forever between 2 and 5.
The fleas meet at 1.
The fleas jump forever between 1 and 2
```

Trie (C++)

```
//PHONE LST TRIE
#include <bits/stdc++.h>
using namespace std;
struct trie { trie *next[10];  bool end;
    trie() {
        for(int i=0; i<10; i++) next[i] = NULL;
        end = false; } };
int main() {
    int t, n, i, p;
    char str[15]; bool flag; cin>>t;
    while(t--) {
        cin>>n; trie *head, *tail;
        head = new trie;  flag = true;
        while(n--) { cin>>str;
            if(flag) {
                tail = head;
                for(i=0; str[i]; i++) {
                    if(tail->end) {
                        flag = false;
                        break; }
                    p = str[i]-48;
if(tail->next[p]==NULL) tail->next[p] = new trie;
                    tail = tail->next[p]; }
                tail->end = true;
                for(i=0; i<10; i++) {
                    if(tail->next[i]) {
                    flag = false;   break;   } } }}
        if(flag) printf("YES\n");
        else printf("NO\n"); } return 0;}
```

Next Permutation (C++)

```
#include <bits/stdc++.h>
using namespace std;
int main() { int casos; scanf("%d", &casos);
    while (casos--) {
        int C, V, A[16] = {};
        char s[16], mm[3] = "CV";
        scanf("%d %d", &C, &V);
        for (int i = C; i < C+V; i++)
            A[i] = 1; int f = 0;
        do { for (int i = 0; i < C+V; i++)
                s[i] = mm[A[i]];
            s[C+V] = '\0';
            if (f)     putchar(' ');
            printf("%s", s), f = 1;
        } while (next_permutation(A, A+C+V));
        puts(""); }return 0; }
```

Hojas de un árbol balanceado (C++)

```
#include <iostream>
using namespace std;
int main() { int n=1; long long int sum=0;
while(n!=0) {
   cin >> n; int arr[n]; arr[0]=n;
    if(n!=0) {
     for(int i = 1 ; i <= n ; i++) {
       cin >> arr[i]; if((2 * i) > n)
       sum += arr[i]; } cout<<sum<<"\n";
  sum=0; } }return 0; }
```

Sets (C++)

```
#include <bits/stdc++.h>
using namespace std;
int main(){
 int n; cin>>n;
 while(n--) {
     int p; cin>>p;
     map<string,int> a; cin.ignore();
     for(int i=0; i<p;i++) {
         string m;  getline(cin,m);
   if(a.insert(pair<string,int>(m,1)).second==false)
{ int k = a.find(m)->second;   k++;
         map<string, int>::iterator it = a.find(m);
              if (it != a.end())
                  it->second = k;  } }
     cout<<endl;
      map<string,int>::iterator it = a.begin();
     for (it=a.begin(); it!=a.end(); ++it)
        std::cout << it->first << it->second << '\n'; }
return 0; }
```

Longest Increasing Subsequence (C++)

// INPUT: a vector of integers **X= 0, 8, 4, 12, 2, 10, 6 || LCS= 0,8,12**

//OUTPUT: a vector containing the longest increasing subsequence

```
#include <bits/stdc++.h>
int Ceil(vector<int> &v, int l, int r, int key) {
while (r-l > 1) { int m = l + (r-l)/2;
if (v[m] >= key) r = m;
else l = m; } return r; } int LIS(vector<int> &v) {
if (v.size() == 0)   return 0;
vector<int> tail(v.size(), 0);
int length = 1;  tail[0] = v[0];
for (size_t i = 1; i < v.size(); i++) {
if (v[i] < tail[0]) tail[0] = v[i];
else if (v[i] > tail[length-1])tail[length++]=v[i];
else tail[Ceil(tail, -1, length-1, v[i])] = v[i];
  } return length;}
```

Knuth-Morris-Pratt (C++)

```
#include <bits/stdc++.h>
using namespace std;
void kmp(const string &needle,const string
&haystack){ int m = needle.size();
  vector<int> border(m + 1); border[0] = -1;
  for (int i = 0; i < m; ++i) {
    border[i+1] = border[i];
while (border[i+1]>-1 and needle[border[i+1]] !=
needle[i]) { border[i+1] = border[border[i+1]]; }
    border[i+1]++; }
  int n = haystack.size(); int seen = 0;
  for (int i = 0; i < n; ++i){
 while (seen > -1 and needle[seen] != haystack[i]) {
     seen = border[seen]; }
    if (++seen == m) { printf("%d\n", i - m + 1);
    seen = border[m];  } } }
int main(){  int m; bool first = true;
  while (scanf("%d",&m)==1) {
    if (!first) puts("");  first = false;
    string needle; getline(cin, needle);
    getline(cin, needle);
    string haystack; getline(cin, haystack);
    kmp(needle, haystack); } return 0;}
```

Búsqueda binaria y lineal (C++)

```
#include <bits/stdc++.h>
using namespace std;
int lineal_search(int *array, int searched, int
arraySize) {
    for (int i = 0; i< arraySize; i++) {
        if (searched == array[i]) { return array[i]; }
}    return 0;}
int binary_search(int *array, int searched, int
arraySize) {  int first = 0, middle, last = arraySize -
1;
    while (first<=last) { middle = (first + last) / 2;
if (searched == array[middle])  return array[middle];
else {
if (array[middle] > searched) last = middle - 1;
else      first = middle + 1;
}    } return -1; }
int main() {
    int arraySize, searched; cin >> arraySize;
    int array[arraySize]; cin >> searched;
    lineal_search(array, searched, arraySize);
    binary_search(array, searched, arraySize);
return 0; }
```

QuickSort (C++)

```
#include <bits/stdc++.h>
int values[] = { 40, 10, 100, 90, 20, 25 };
int compare (const void * a, const void * b) {
  return ( *(int*)a - *(int*)b ); }
int main (){ int n;
qsort (values, 6, sizeof(int), compare);
for (n=0; n<6; n++)  printf ("%d ",values[n]);
return 0; }
```

Latitud/Longitud (C++)

```
//Converts from rectangular coordinates to latitude
/longitude and vice versa. Uses degrees (not
radians).
#include <bits/stdc++.h>
using namespace std;
struct ll { double r, lat, lon; };
struct rect { double x, y, z; };
ll convert(rect& P) { ll Q;
Q.r = sqrt(P.x*P.x+P.y*P.y+P.z*P.z);
Q.lat = 180/M_PI*asin(P.z/Q.r);
Q.lon = 180/M_PI*acos(P.x/sqrt(P.x*P.x+P.y*P.y));
return Q; }
rect convert(ll& Q) { rect P;
P.x = Q.r*cos(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
P.y = Q.r*sin(Q.lon*M_PI/180)*cos(Q.lat*M_PI/180);
P.z = Q.r*sin(Q.lat*M_PI/180);
return P; }
int main() { rect A; ll B;
A.x = -1.0; A.y = 2.0; A.z = -3.0;
B = convert(A);
cout << B.r << " " << B.lat << " " << B.lon <<
endl; A = convert(B);
cout << A.x << " " << A.y << " " << A.z << endl; }
```

Números Primos (C++)

```
//2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,41,
43, 47, 53, 59, 61, 67, 71,73, 79, 83, 89, 97
#include <bits/stdc++.h>
using namespace std;
bool prime(int n) {
if (n<2) return false; if (n<=3) return true;
if (!(n%2) || !(n%3)) return false;
for (int i=5;i*i<=n;i+=6)
if (!(n%i) || !(n%(i+2))) return false;
return true; }
```

Constantes (C++)

PI: 4 * atan(1)
Distancia Eucledeiana: sqrt(pow(q[i].x-actual.x,2.0)+ pow(q[i].y-actual.y,2.0)+ pow(q[i].z-actual.z,2.0));

Distancia Manhattan: abs(x-x)+abs(y-y)

INF: 0x3F3F3F3F

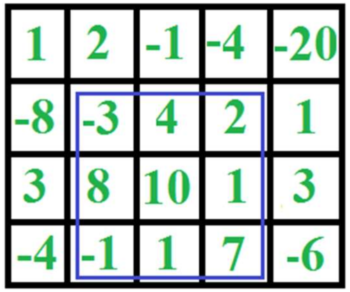
```
//number too large. use powl instead of pow.
powl(a, b)
(int)round(p, (1.0/n))
printf("%.1f\n", (a * b)/2);
void copy(first, last, result);
void swap(a,b);
void swap(first1, last1, first2);
void replace(first, last, old_value, new_value);
void replace_if(first, last, pred, new_value);
void reverse(first, last);
void reverse_copy(first, last, result);
// Searching
int find(const string &s2, int pos1 = 0);
int rfind(const string &s2, int pos1 = end);
int find_first_of(const string &s2, int pos1 = 0);
int find_last_of(const string &s2, int pos1 = end);
int find_first_not_of(const string &s2,int pos1=0);
int find_last_not_of(const string &s2,int pos1=end);
// Insert, Erase, Replace
string& insert(int pos1,const string &s2);
string& insert(int pos1,int repetitions, char c);
string& erase(int pos = 0,int len = npos);
string& replace(int pos1,int len1, const string &s2);
string& replace(int pos1, int len1,int
repetitions, char c); int i = 22;
s1 << "Hello world! " << i;
cout << s1.str() << endl;
```

Maneras de Leer en ficheros (C++)

```
if(scanf(“%d”,&cases!=EOF) || ==1
while(getline(cin,str))
```

Maximum submatrix (C++)

```
#include<iostream>
using namespace std;
int a[150][150]={0};
int c[200]={0};
int maxarray(int n) {
int b=0, sum=-100000000;
for (int i=1;i<=n;i++) {
if (b>0) b+=c[i]; else b=c[i];
if (b>sum) sum=b; } return sum; }
int maxmatrix(int n) {
int sum=-100000000, max=0;
for (int i=1;i<=n;i++) {
for (int j=1;j<=n;j++) c[j]=0;
for (int j=i;j<=n;j++) {
for (int k=1;k<=n;k++) c[k]+=a[j][k];
max=maxarray(n);
if (max>sum) sum=max; } } return sum; }
int main(void) { int n=0; cin >> n;
for (int i=1;i<=n;i++)
for (int j=1;j<=n;j++)
cin >> a[i][j]; cout << maxmatrix(n); return 0;}
```



1	2	-1	-4	-20
-8	-3	4	2	1
3	8	10	1	3
-4	-1	1	7	-6

Partitions Integer (C++)

```
     //4 = 3 + 1 , 2 + 2 ,
     2 + 1 + 1 ,
     1 + 1 +1 + 1
    void printAllUniqueParts(int n) {
    int p[n]; int k = 0;  p[k] = n;
    while (true) {
        printArray(p, k+1);
        int rem_val = 0;
        while (k >= 0 && p[k] == 1) {
            rem_val += p[k];  k--; }
        if (k < 0)  return;
        p[k]--;
        rem_val++;
        while (rem_val > p[k]) {
            p[k+1] = p[k];
            rem_val = rem_val - p[k];
            k++;   }
        p[k+1] = rem_val; k++; } }
```

Longest Common Subsequence(C++)

```
#include<bits/stdc++.h>
using namespace std;
string X;
string Y;
int memo[1005][1005];
bool mark[1005][1005];
int lcs(int m, int n )
{
   if (m == 0 || n == 0) return 0;
   int &best= memo[m][n];
   if(mark[m][n]) return best;
   mark[m][n]=true;
   if (X[m-1] == Y[n-1])  return best=1+lcs(m-1, n-1);
   return best=max(lcs(m, n-1),lcs(m-1, n));
}
int main()
{
  while(getline(cin,X))
  {
      memset(mark,false,sizeof(mark));
      memset(memo,0,sizeof(memo));
  getline(cin,Y);
  printf("%d\n", lcs(X.length(),Y.length()));
  }
  return 0; }
```

Longest Increasing Common Subsequence (C++)

```
    // 2 3 1 6 5 4 6 AND 1 3 5 6 the LCIS is 3 5 6.
//Dada una lista de numberos de longitud n,
extrae a que es la mayor subsecuencia de aumento
O(nlogn)
//INPUT: a vector of integers // Posible solucion
//X= 0, 8, 4, 12, 2, 10, 6 || LCS= 0,8,12
//OUTPUT: a vector containing the longest increasing
subsequence
#include <bits/stdc++.h>
using namespace std;
int Ceil(vector<int> &v, int l, int r, int key) {
while (r-l > 1) { int m = l + (r-l)/2;
if (v[m] >= key) r = m;
else l = m; } return r; }
int LIS(vector<int> &v) {
if (v.size() == 0)   return 0;
vector<int> tail(v.size(), 0);
int length = 1;
tail[0] = v[0];
for (size_t i = 1; i < v.size(); i++) {
if (v[i] < tail[0]) tail[0] = v[i];
else if (v[i] > tail[length-1])tail[length++]=v[i];
else tail[Ceil(tail, -1, length-1, v[i])] = v[i];
  } return length;}
```

Partitions of sets – Bell Numbers (C++)

```
#include<iostream>
using namespace std;
int countP(int n, int k) {
  if (n == 0 || k == 0 || k > n) return 0;
  if (k == 1 || k == n) return 1;
  return  k*countP(n-1, k) + countP(n-1, k-1); }
int main() {
    int a=0;
   for(int i=0; i<5;i++)
//Sin for solo devuelve en X subsets
de esa cantidad
   a+=countP(5, i);  a++;
   cout<<a<<endl;  return 0; }
//1,1,2,5,15,52 Bell Numbers
```

FastInput (C++)

```
inline int getchar_unlocked() { return getchar();}

inline void fastInput(int &n){
    char ch;
    int sign = 1;
    while(ch = getchar_unlocked(), isspace(ch)) {

    };
    n = 0;
    if(ch == '-')
        sign = -1;
    else n = ch - '0';
    while(ch = getchar_unlocked(), isdigit(ch))
        n = (n << 3) + (n << 1) + ch - '0';
    n *= sign; }
```

```
#include <bits/stdc++.h>
using namespace std;
struct point{
    double x,y;
    point(){}
    point(double a,double b){
        x=a;y=b;
    }
    bool operator<(const point &other)const{
        return x < other.x || (x==other.x && y <
other.y); } };
int L,S;
point points[MAXN];
int orientation(point a,point b,point c){
    int v = (b.y-a.y) * (c.x-b.x) - (b.x-a.x) *
(c.y-b.y);
    if(!v) return 0; //colinear
    return v>0?1:2; // clock or counterclock wise }
double cross(const point &O, const point &A, const
point &B)
{    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y)
* (B.x - O.x); }
vector<point> getConvexHull(){
    int n = L, k = 0;
    vector<point> H(2*n);
    // Sort points lexicographically
    sort(points, points+n);
    // Build lower hull
    for (int i = 0; i < n; ++i) {
        while (k >= 2 && cross(H[k-2], H[k-1],
points[i]) <= 0) k--;
        H[k++] = points[i];  }
    for (int i = n-2, t = k+1; i >= 0; i--) {
        while (k >= t && cross(H[k-2], H[k-1],
points[i])
 <= 0) k--;
        H[k++] = points[i]; }
    H.resize(k-1);
    return H;}
//Comprueba si el punto q pertenece al segmento pr
bool onSegment(point p, point q, point r) {
 if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&
        q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))
        return true;
    return false; }
bool doIntersect(point p1, point q1, point p2, point
q2) {
    // Find the four orientations needed for general
and special cases
    int o1 = orientation(p1, q1, p2);
    int o2 = orientation(p1, q1, q2);
    int o3 = orientation(p2, q2, p1);
    int o4 = orientation(p2, q2, q1);
    if (o1 != o2 && o3 != o4)
        return true;
    if (o1 == 0 && onSegment(p1, p2, q1)) return true;
    if (o2 == 0 && onSegment(p1, q2, q1)) return true;
    if (o3 == 0 && onSegment(p2, p1, q2)) return true;
    if (o4 == 0 && onSegment(p2, q1, q2)) return true;
    return false;  }
int pointInsidePolygon(point p, vector<point> polygon){
    int n = (int)polygon.size();
    if (polygon.size() < 3)
        return false;
    if (cross(polygon[0], p, polygon[1]) > 1e-10)
        return false;
    if (cross(polygon[0], p, polygon[n-1]) < -1e-10)
        return false;
    int l = 2, r = n-1;
    int line = -1;
    while (l <= r) {
        int mid = (l + r)>>1;
     if (cross(polygon[0],p, polygon[mid]) > -1e-10) {
            line = mid;
            r = mid - 1;
        } else l = mid + 1; }
    return cross(polygon[line-1], p, polygon[line]) <
1e-10; }
/*bool pointInsidePolygon(point P, vector<point> poly){
    int n = poly.size();
    bool in = 0;
    for(int i = 0,j = n - 1;i < n;j = i++){
        double dx = poly[j].x - poly[i].x;
        double dy = poly[j].y - poly[i].y;

  if((poly[i].y <= P.y + 1e-10 && P.y < poly[j].y) ||
       (poly[j].y <= P.y + 1e-10 && P.y < poly[i].y))
    if(P.x - 1e-10 < dx * (P.y-poly[i].y)/dy+poly[i].x)
                 in ^= 1; }
    return in; }*/
double signed_area(vector<point> &poly){
  int n = poly.size();
  if(n < 3) return 0.0;

  double S = 0.0;

  for(int i = 1;i <= n;++i)
    S += poly[i % n].x * (poly[(i + 1) % n].y - poly[i -
1].y);
  return S / 2; }
int main(){
    while(scanf("%d",&L)!=EOF){
        FOR(i,0,L){
            RF(points[i].x);   RF(points[i].y);
        }
        vector<point> convex_hull = getConvexHull();
        int cnt = 0;
        RI(S);
        FOR(i,0,S){
            int x,y; RI(x); RI(y);
            cnt += pointInsidePolygon(point(x,y),
convex_hull);       }
        printf("%d\n",cnt);  }   return 0; }
```

Among oldest and most well-studied problems in
 computational geometry problems.Robot motion planning.
Shortest perimeter fence enclosing P.
Smallest area polygon enclosing P.
Unique convex polygon whose vertices are points in P that
encloses P.
Smallest convex set containing all N points (i.e.,
intersection of all convex sets containing the N points).