Universidade Federal de Goiás
Team Reference Material

2019 South America/Brazil Regional Contest

Contents

Augmenting Path

```
//Augmenting Path for MCBM
#include <bits/stdc++.h>
using namespace std;

const int N = 1000;
vector<int> g[N];

int augment(int v, vector<int> &match, vector<bool> &visited) {
      if (visited[v]) return 0;
      visited[v] = true;

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

            if (match[u] == -1 || augment(u, match, visited)) {
                  match[u] = v;
                  return 1;
            }
      }

      return 0;
}

int matching(int left_sz, int right_sz) {
      vector<int> match(left_sz + right_sz, -1);
      vector<bool> visited;

      int mcmb = 0;

      for (int i = 0; i < left_sz; i++) {
            visited.assign(left_sz, false);
            mcmb += augment(i, match, visited);
      }

      return mcmb;
}

int main() {
      ios::sync_with_stdio(false);

      return 0;
}
```

Combinations

```
//Combination Algorithm
#include <bits/stdc++.h>
using namespace std;

const int N = 120;
int comb[N][N];
int combination(int n, int m) {
   return comb[n][m];
}

void pre_compute(int n) {
   assert(n < N-1);
   for (int i = 0; i <= n; i++) {
      comb[i][0] = comb[i][i] = 1;
      for (int j = 0; j < i; j++) {
         comb[i][j] = comb[i-1][j-1] + comb[i-1][j];
      }
   }
}

int main() {
   ios::sync_with_stdio(false);

   return 0;
}
```

Debug Msg

```
//Lib - Debug message for CF
#include <bits/stdc++.h>
using namespace std;

template<typename T>
void debug(T value) {
#ifndef ONLINE_JUDGE
   cout << value << endl;
#endif
}

template <typename T, typename... Args>
void debug(T value, Args... args) {
#ifndef ONLINE_JUDGE
   cout << value;
   debug(args...);
#endif
}

int main() {
   debug("this␣message␣do␣not␣will␣appear␣in␣CF.");
   return 0;
}
```

Dinic (Max Flow)

```cpp
#include <bits/stdc++.h>
using namespace std;

struct FlowEdge {
   int v, u;
   long long cap, flow = 0;
   FlowEdge(int v, int u, long long cap) : v(v), u(u), cap(cap) {}
};

struct Dinic {
   const long long flow_inf = 1e18;
   vector<FlowEdge> edges;
   vector<vector<int>> adj;
   int n, m = 0;
   int s, t;
   vector<int> level, ptr;
   queue<int> q;

   Dinic(int n, int s, int t) : n(n), s(s), t(t) {
      adj.resize(n);
      level.resize(n);
      ptr.resize(n);
   }

   void add_edge(int v, int u, long long cap) {
      edges.push_back(FlowEdge(v, u, cap));
      edges.push_back(FlowEdge(u, v, 0));
      adj[v].push_back(m);
      adj[u].push_back(m + 1);
      m += 2;
   }

   bool bfs() {
      while (!q.empty()) {
         int v = q.front();
         q.pop();
         for (int id : adj[v]) {
            if (edges[id].cap - edges[id].flow < 1)
               continue;
            if (level[edges[id].u] != -1)
               continue;
            level[edges[id].u] = level[v] + 1;
            q.push(edges[id].u);
         }
      }
      return level[t] != -1;
   }

   long long dfs(int v, long long pushed) {
      if (pushed == 0)
         return 0;
      if (v == t)
         return pushed;
      for (int& cid = ptr[v]; cid < (int)adj[v].size(); cid++) {
         int id = adj[v][cid];
         int u = edges[id].u;
         if (level[v] + 1 != level[u] || edges[id].cap - edges[id].flow < 1)
            continue;
         long long tr = dfs(u, min(pushed, edges[id].cap - edges[id].flow));
         if (tr == 0)
            continue;
         edges[id].flow += tr;
         edges[id ^ 1].flow -= tr;
         return tr;
      }
      return 0;
   }

   long long flow() {
      long long f = 0;
      while (true) {
         fill(level.begin(), level.end(), -1);
         level[s] = 0;
         q.push(s);
         if (!bfs())
            break;
         fill(ptr.begin(), ptr.end(), 0);
         while (long long pushed = dfs(s, flow_inf)) {
            f += pushed;
         }
      }
      return f;
   }
};

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

Edmonds-Karp (Max Flow)

```cpp
// Lib - Edmonds-Karp Algorithm
// UVA - 820 - Network Bandwidth
#include <bits/stdc++.h>
using namespace std;

struct Edmonds {
   static const int64_t inf = 1e18;
   static const int V = 150;
   int64_t adj[V][V];
   vector<int> g[V];
   int p[V]; // parent vector
   int64_t dist[V];
   int n, s, t;

   Edmonds(int n, int s, int t) : n(n), s(s), t(t) {
      clear();
   }

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

```
      g[i].clear();
   memset(adj, 0, sizeof(adj));
}

int64_t augmented() {
   int u = t;
   int64_t minimum = inf;

   // find minimum flow in augmented path
   while (p[u] != -1) {
      minimum = min(adj[p[u]][u], minimum);
      u = p[u];
   }

   // walk in augment path updating flow
   u = t;
   while (p[u] != -1) {
      adj[p[u]][u] -= minimum;
      adj[u][p[u]] += minimum;
      u = p[u];
   }

   return minimum; // return minimum flow in augmented path
}

int64_t flow() {
   int64_t mf = 0, // max flow answer
         f = 1;
   int u;

   while (f > 0) {
      f = 0;
      for (int i = 0; i < n; i++)
         dist[i] = inf, p[i] = -1;
      dist[s] = 0;

      queue<int> q;
      q.push(s);

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

         // stop if reach sink t
         if (u == t) break;

         for (const auto &v : g[u]) {
            if (adj[u][v] > 0 && dist[v] == inf) {
               dist[v] = dist[u] + 1;
               q.push(v);
               p[v] = u;
            }
         }
      }

      // verify if bfs stop when reach sink t
      if (u == t) {
         // find minimum flow in augmented path
         f = augmented();

         // update max flow of network
         mf += f;
      }
   }

   return mf;
}

void add_edge(int v, int u, int64_t cap) {
   adj[v][u] = cap;
   adj[u][v] = 0;
   g[v].push_back(u);
   g[u].push_back(v);
}

void add_bi_edge(int v, int u, int64_t cap) {
   adj[v][u] = cap;
   adj[u][v] = cap;
   g[v].push_back(u);
   g[u].push_back(v);
}

void inc_edge(int v, int u, int64_t cap) {
   if (adj[v][u] == 0)
      g[v].push_back(u);
   adj[v][u] += cap;
}
};

int main() {
      ios::sync_with_stdio(false);

      int n, m, k;
      int a, b, source, sink;
      int64_t c;

      k = 1;
      while (cin >> n, n) {
            cin >> source >> sink >> m;
            source--;
            sink--;

            Edmonds edmonds(n, source, sink);

            for (int i = 0; i < m; i++) {
                  cin >> a >> b >> c;
                  a--;
                  b--;
                  edmonds.inc_edge(a, b, c);
                  edmonds.inc_edge(b, a, c);
            }

            c = edmonds.flow();
            cout << "Network␣" << k << "\nThe␣bandwidth␣is␣" << c << ".\n\n";
            k++;
      }

      return 0;
}
```

Euler's Tour

```cpp
// Euler Tour
#include <bits/stdc++.h>
using namespace std;

const int N = 100 * 1000 + 10;
vector<int> g[N];
int sz[N], ini[N], cnt = 0;
int64_t path[N];

int dfs(int v, int p) {
   ini[v] = cnt;
   path[cnt] = v;
   cnt++;
   sz[v] = 1;

   for (const auto &u : g[v]) {
      if (u == p) continue;

      sz[v] += dfs(u, v);
   }

   return sz[v];
}

int main() {
   int n;

   cin >> n;

   for (int i = 0, x, y; i < n-1; i++) {
      cin >> x >> y;
      g[x].push_back(y);
      g[y].push_back(x);
   }

   dfs(0, -1);

   for (int i = 0; i < cnt; i++) {
      cout << path[i] << "␣";
   }
   cout << endl;

   int v = 1;
   // subarvore de v
   for (int i = ini[v]; i < ini[v]+sz[v]; i++) {
      cout << path[i] << "␣";
   }
   cout << endl;

   return 0;
}
```

Extended Euclids

```cpp
// Euclids Algorithm and Modular Multiplicative Inverse
#include <bits/stdc++.h>
using namespace std;

pair<int, int> extendedEuclids(int a, int b) {
   if (b == 0) {
      return {1, 0};
   } else {
      auto p = extendedEuclids(b, a % b);
      return {p.second,
         p.first - floor((double)a / b) * p.second};
   }
}

int inverseMod(int a, int n) {
   auto p = extendedEuclids(a, n);
   return (p.first % n + n) % n;
}

int main() {
   ios::sync_with_stdio(false);

   return 0;
}
```

Fenwick Tree

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 2*1e5+10;

int bit[N];

// execute the query [1, x]
int query(int x) {
      int s = 0;

      while (x) {
            s += bit[x];
            x -= (x & -x);
      }

      return s;
}

// execute the update [1,x]
void update(int x, int v) {
      while (x <= N) {
            bit[x] += v;
            x += (x & -x);
      }
}
```

```
int main() {
      ios::sync_with_stdio(false);
      return 0;
}
```

Fenwick Tree 2D

```
//Lib - Fenwick Tree 2D
#include <bits/stdc++.h>
using namespace std;

const int N = 1010;
int ft[N][N];

int last(int v) {
      return v & -v;
}

void update(int x, int y, int val) {
      for (int i = x; i < N; i += last(i)) {
            for (int j = y; j < N; j += last(j)) {
                  ft[i][j] += val;
            }
      }
}

int query(int x, int y) {
      int s = 0;
      for (int i = x; i > 0; i -= last(i)) {
            for (int j = y; j > 0; j -= last(j)) {
                  s += ft[i][j];
            }
      }
      return s;
}

int sum(int x1, int y1, int x2, int y2) {
      return query(x2, y2) - query(x2, y1 - 1)
            - query(x1 - 1, y2) + query(x1 - 1, y1 - 1);
}

int main() {
   ios::sync_with_stdio(false);
   return 0;
}
```

Fenwick Tree 2D of XOR

```
//Lib - Fenwick Tree 2D XOR
#include <bits/stdc++.h>
using namespace std;

const int N = 1010;
int64_t dp[4][N][N];

int parity(int x, int y) {
      int res = 0;
      if (x % 2) res++;
      if (y % 2) res += 2;
      return res;
}

int64_t query(int x, int y) {
      int64_t res = 0;
      x++;
      y++;
      int whichSquare = parity(x, y);
      for (int i = x; i > 0; i -= (i & (-i))) {
            for (int j = y; j > 0; j -= (j & (-j))) {
                  res ^= dp[whichSquare][i][j];
            }
      }
      return res;
}

void update(int x, int y, int64_t val) {
      x++;
      y++;
      int whichSquare = parity(x, y);
      for (int i = x; i <= N; i += (i & (-i))) {
            for (int j = y; j <= N; j += (j & (-j))) {
                  dp[whichSquare][i][j] ^= val;
            }
      }
}

int query2d(int x1, int y1, int x2, int y2) {
   int r = query(x2, y2);
   r ^= query(x2, y1 - 1);
   r ^= query(x1 - 1, y2);
   r ^= query(x1 - 1, y1 - 1);
   return r;
}

void update2d(int x1, int y1, int x2, int y2, int val) {
   update(x1, y1, val);
   update(x1, y2 + 1, val);
   update(x2 + 1, y1, val);
   update(x2 + 1, y2 + 1, val);
}

int main() {
   ios::sync_with_stdio(false);
```

```
   int n, m, p;
   int a, b, c, d;
   cin >> n >> m;
   for (int i = 0; i < m; i++) {
      cin >> p >> a >> b >> c >> d;

      update2d(a, b, c, d, p);
   }

   for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
         cout << query2d(i, j, i, j);
         if (j < n)
            cout << "␣";
      }
      cout << endl;
   }
   return 0;
}
```

Floyd Warshall

```
// Lib - Floyd Warshall
#include <bits/stdc++.h>
using namespace std;

const int N = 110, inf = 1e9;
int64_t dist[N][N];

void floyd_warshall(int n) {
   for (int v = 1; v <= n; v++) {
      for (int u = 1; u <= n; u++) {
         for (int q = 1; q <= n; q++) {
            dist[u][q] = min(dist[u][q], dist[u][v] + dist[v][q]);
         }
      }
   }
}

int main() {
   int t, n, m;

   cin >> t;

   while (t--) {
      cin >> n >> m;

      for (int v = 1; v <= n; v++) {
         for (int u = 1; u <= n; u++) {
            if (u == v)
               dist[v][u] = 0;
            else
               dist[v][u] = inf;
         }
      }

      int64_t d;
      for (int i = 0, x, y; i < m; i++) {
         cin >> x >> y >> d;
         dist[x][y] = min(dist[x][y], d);
         dist[y][x] = min(dist[y][x], d);
      }

      floyd_warshall(n);

      for (int v = 1; v <= n; v++) {
         for (int u = 1; u <= n; u++) {
            cout << dist[v][u] << "␣";
         }
         cout << "\n";
      }
   }
   return 0;
}
```

Gaussian Elimination

```
// Lib - Gaussian Elimination
// UVA 11319 - Stupid Sequence
#include <bits/stdc++.h>
using namespace std;

const int N = 7, M = 7; // N variables, M equations
const int INF = 1e9;
const double EPS = 1e-9;

struct Matrix {
   double m[M][N+1];
   double* operator[](size_t i) {
      return m[i];
   };
};

struct Vector {
   double v[N];
   double& operator[](size_t i) {
      return v[i];
   };
};

// n variables, m equations
// note: for reducing error, implicit pivoting can be used
int elimination(Matrix &aug, int n, int m, Vector &ans) {
   int where[N];
   memset(where, -1, sizeof(where));

   for (int col = 0, row = 0, lgst; col < n && row < m; col++) {
```

```
      lgst = row;
      for (int i = row+1; i < m; i++) {
         if (fabs(aug.m[i][col]) > fabs(aug.m[lgst][col])) {
            lgst = i;
         }
      }

      if (fabs(aug.m[lgst][col]) < EPS) // independent variable or impossible system
         continue;

      swap(aug.m[row], aug.m[lgst]);

      where[col] = row; // assign a row for variable of column col

      // zero elements in column col (except in row)
      for (int i = 0; i < m; i++) {
         if (i == row) continue;

         double c = aug.m[i][col] / aug.m[row][col];

         for (int j = col; j <= n; j++) {
            aug.m[i][j] -= aug.m[row][j] * c;
         }
      }

      row++;
   }

   // assign answer to variables
   memset(ans.v, 0, sizeof(ans.v));
   for (int j = 0; j < n; j++) {
      if (where[j] != -1) {
         ans.v[j] = aug.m[where[j]][n] / aug.m[where[j]][j];
      }
   }

   // verify if system is impossible
   for (int i = 0; i < m; i++) {
      double sum = 0;
      for (int j = 0; j < n; j++) {
         sum += (ans.v[j] * aug.m[i][j]);
      }

      if (fabs(sum - aug.m[i][n]) > EPS) {
         return 0; // impossible system
      }
   }

   for (int j = 0; j < n; j++) {
      if (where[j] == -1) // infinite solutions
         return INF;
   }

   return 1;
}

int main() {
   int t;
   int n = 7;
   const int Z = 1500;
   uint64_t v, ans2[N], y[Z];

   Matrix aug;

   Vector ans;

   cin >> t;
   while (t--) {
      for (int i = 0; i < n; i++) {
         v = 1;
         for (int j = 0; j < n; j++) {
            aug[i][j] = v;
            v *= (i+1);
         }
      }

      for (int i = 0; i < Z; i++) {
         cin >> y[i];
      }

      for (int i = 0; i < n; i++) {
         aug[i][n] = y[i];
      }

      bool has_solution = true;

      if (elimination(aug, n, n, ans) != 1) {
         has_solution = false;
      } else {
         for (int j = 0; j < n; j++) {
            if (ans[j] < -EPS) {
               // it is negative
               has_solution = false;
               break;
            }

            ans2[j] = static_cast<uint64_t>(ans[j] + EPS);

            if (ans2[j] > 1000) {
               // greater than 1000
               has_solution = false;
               break;
            }
         }

         // verify if ans2 vector generate the results of function of input
         for (int i = 0; i < 1500; i++) {
            uint64_t f = 0;
            v = 1;
            for (int j = 0; j < n; j++) {
               f += ans2[j] * v;
               v *= (i+1);
            }

            if (f != y[i]) {
               has_solution = false;
               break;
            }
         }
      }

      if (has_solution) {
         for (int j = 0; j < n; j++) {
            cout << ans2[j] << "␣\n"[j == n-1];
         }
      } else {
         cout << "This␣is␣a␣smart␣sequence!" << endl;
```

```
      }
   }
   return 0;
```

Gaussian Elimination Xor

```
// Lib - Gaussian Elimination XOR
// For linear equation systems in modulo 2
#include <bits/stdc++.h>
using namespace std;

const int N = 10, M = 10; // N variables, M equations
const int INF = 1e9;

struct Matrix {
   bitset<N+1> m[M];
   bitset<N+1>& operator[](size_t i) {
      return m[i];
   };
};

struct Vector {
   bitset<N> v;
   int operator[](size_t i) {
      return v[i];
   };
};

// n variables, m equations
int elimination(Matrix &aug, int n, int m, Vector &ans) {
   int where[N];
   memset(where, -1, sizeof(where));

   for (int col = 0, row = 0; col < n && row < m; col++) {
      for (int i = row; i < m; i++) {
         if (aug.m[i][col]) {
            swap(aug.m[i], aug.m[row]);
            break;
         }
      }

      if (!aug.m[row][col]) // independent variable or impossible system
         continue;

      where[col] = row; // assign a row for variable of column col

      // zero elements in column col (except in row)
      for (int i = 0; i < m; i++) {
         if (i != row && aug.m[i][col]) {
            aug.m[i] ^= aug.m[row];
         }
      }

      row++;
   }

   // assign answer to variables
   for (int j = 0; j < n; j++) {
      if (where[j] != -1) {
         ans.v[j] = aug.m[where[j]][n];
      }
   }

   // verify if system is impossible
   for (int i = 0; i < m; i++) {
      int sum = 0;
      for (int j = 0; j < n; j++) {
         if (where[j] != -1)
            sum ^= (ans.v[j] & aug.m[i][j]);
      }

      if (sum != aug.m[i][n]) {
         cout << aug.m[i] << endl;
         return 0; // impossible system
      }
   }

   for (int j = 0; j < n; j++) {
      if (where[j] == -1) // infinite solutions
         return INF;
   }

   return 1;
}

void printSol(Matrix aug, int n, int m) {
   // print system
   for (int i = 0; i < m; i++) {
      for (int j = 0; j <= n; j++) {
         cout << aug.m[i][j];
      }
      cout << endl;
   }

   Vector ans;
   cout << "number␣of␣solutions:␣" << elimination(aug, n, m, ans) << endl;
   cout << "solution␣(x0,␣...,␣x" << n-1 << "):␣";
   for (int j = 0; j < n; j++) {
      cout << ans[j] << "␣";
   }
   cout << endl;
}

int main() {
   Matrix aug;
   // inverse order of values of matrix (bitset details)
   aug.m[0] = bitset<N+1>("11101");
   aug.m[1] = bitset<N+1>("01110");
   aug.m[2] = bitset<N+1>("01011");
   aug.m[3] = bitset<N+1>("10111");

   cout << "System␣1:␣" << endl;
   printSol(aug, 4, 4);

   aug.m[0] = bitset<N+1>("1111");
   aug.m[1] = bitset<N+1>("0001");
}
```

```
   aug.m[2] = bitset<N+1>("0111");
   aug.m[3] = bitset<N+1>("0110");

   cout << "System␣2:␣" << endl;
   printSol(aug, 3, 4);

   aug.m[0] = bitset<N+1>("0000001");
   aug.m[1] = bitset<N+1>("1001011");
   aug.m[2] = bitset<N+1>("1010101");
   aug.m[3] = bitset<N+1>("0010110");
   aug.m[4] = bitset<N+1>("0001110");

   cout << "System␣3:␣" << endl;
   printSol(aug, 6, 5);

      return 0;
}
```

Generate Combinations

```
#include <bits/stdc++.h>
using namespace std;

void gen_comb(vector<int> a, int r) {
   assert(r <= (int) a.size());

   vector<bool> v(a.size());
   fill(v.begin(), v.begin() + r, true);

   do {
      for (size_t i = 0; i < v.size(); i++) {
         if (v[i]) {
            cout << a[i] << "␣";
         }
      }
      cout << "\n";
   } while (prev_permutation(v.begin(), v.end()));
}

int main() {
   const int N = 50, R = 4;
   vector<int> a(N);

   for (int i = 0; i < N; i++) {
      a[i] = rand();
   }

   gen_comb(a, R);
   return 0;
}
```

Geometry

```
// Lib - Computational Geometry
#include <bits/stdc++.h>
using namespace std;

const double PI = 3.14159265358979323846;
const double EPS = 1e-9;

struct point {
   int x, y;
   bool operator==(point &other) {
      return this->x == other.x && this->y == other.y;
   }
};

struct vec {
   double x, y;
   vec(double _x, double _y) : x(_x), y(_y) {}
};

// Note about polygons: vector<point> with
// the first and last positions equals (same point)
// is a polygon

vec to_vec(point a, point b) {
   return vec(b.x - a.x, b.y - a.y);
}

double dot(vec a, vec b) {
   return (a.x * b.x + a.y * b.y);
}

double norm_sq(vec v) {
   return v.x * v.x + v.y * v.y;
}

double cross(vec a, vec b) {
   return a.x * b.y - a.y * b.x;
}

bool collinear(point p, point q, point r) {
   return fabs(cross(to_vec(p, q), to_vec(p, r))) < EPS;
}

double angle(point a, point o, point b) {
   vec oa = to_vec(o, a), ob = to_vec(o, b);
   return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob)));
}

bool ccw(point p, point q, point r) {
   return cross(to_vec(p, q), to_vec(p, r)) > 0;
}

// ccw with support to collinear points
bool coccw(point p, point q, point r) {
```

```
   if (collinear(p, q, r))
      return true;

   return cross(to_vec(p, q), to_vec(p, r)) > 0;
}

double dist(point p1, point p2) {
   return hypot(p1.x - p2.x, p1.y - p2.y);
}

double area(const vector<point> &q) {
   double result = 0.0, x1, y1, x2, y2;
   for (int i = 0; i < (int)q.size()-1; i++) {
      x1 = q[i].x;
      x2 = q[i+1].x;
      y1 = q[i].y;
      y2 = q[i+1].y;
      result += (x1 * y2 - x2 * y1);
   }

   return fabs(result) / 2.0;
}

// check if point p is inside polygon p
// restrictions: p can not be collinear
// with points of q
bool inner(point p, vector<point> q) {
   if (q.size() == 0) return false;

   double sum = 0;
   for (size_t i = 0; i < q.size()-1; i++) {
      if (ccw(p, q[i], q[i+1])) {
         sum += angle(q[i], p, q[i+1]);
      } else {
         sum -= angle(q[i], p, q[i+1]);
      }
   }

   // fabs(sum) == 2 * PI
   return fabs(fabs(sum) - 2 * PI) < EPS;
}

// check if point p is inside polygon q
// p can be collinear to edges of polygon q
bool inner2(point p, vector<point> q) {
   double triangles_area = 0, total_area = area(q);

   for (size_t i = 0; i < q.size()-1; i++) {
      vector<point> t = {p, q[i], q[i+1], p};
      triangles_area += area(t);
   }

   return fabs(triangles_area - total_area) < EPS;
}

int main() {
   int n;
   vector<point> poly;

   cin >> n;
   for (int i = 0, x, y; i < n; i++) {
      cin >> x >> y;
      poly.push_back({x, y});
   }

   cout << "area:␣" << area(poly) << endl;
   return 0;
}
```

Geometry CP3

```
#include <bits/stdc++.h>
using namespace std;

#define INF 1e9
#define EPS 1e-9
#define PI acos(-1.0) // important constant; alternative #define PI (2.0 * acos(0.0))

double DEG_to_RAD(double d) {
   return d * PI / 180.0;
}

double RAD_to_DEG(double r) {
   return r * 180.0 / PI;
}

// struct point_i { int x, y; }; // basic raw form, minimalist mode
struct point_i {
   int x, y; // whenever possible, work with point_i
   point_i() {
      x = y = 0; // default constructor
   }
   point_i(int _x, int _y) : x(_x), y(_y) {}
}; // user-defined

struct point {
   double x, y; // only used if more precision is needed
   point() {
      x = y = 0.0; // default constructor
   }
   point(double _x, double _y) : x(_x), y(_y) {} // user-defined
   bool operator < (point other) const { // override less than operator
      if (fabs(x - other.x) > EPS) // useful for sorting
         return x < other.x; // first criteria , by x-coordinate
      return y < other.y;
   } // second criteria, by y-coordinate
   // use EPS (1e-9) when testing equality of two floating points
   bool operator == (point other) const {
      return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS));
   }
};

double dist(point p1, point p2) { // Euclidean distance
   // hypot(dx, dy) returns sqrt(dx * dx + dy * dy)
   return hypot(p1.x - p2.x, p1.y - p2.y);
} // return double
```

```
// rotate p by theta degrees CCW w.r.t origin (0, 0)
point rotate(point p, double theta) {
   double rad = DEG_to_RAD(theta); // multiply theta with PI / 180.0
   return point(p.x * cos(rad) - p.y * sin(rad),
            p.x * sin(rad) + p.y * cos(rad));
}

struct line {
   double a, b, c;
}; // a way to represent a line

// the answer is stored in the third parameter (pass by reference)
void pointsToLine(point p1, point p2, line &l) {
   if (fabs(p1.x - p2.x) < EPS) { // vertical line is fine
      l.a = 1.0;
      l.b = 0.0;
      l.c = -p1.x; // default values
   } else {
      l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
      l.b = 1.0; // IMPORTANT: we fix the value of b to 1.0
      l.c = -(double)(l.a * p1.x) - p1.y;
   }
}

// not needed since we will use the more robust form: ax + by + c = 0 (see above)
struct line2 {
   double m, c;
}; // another way to represent a line

int pointsToLine2(point p1, point p2, line2 &l) {
   if (abs(p1.x - p2.x) < EPS) { // special case: vertical line
      l.m = INF; // l contains m = INF and c = x_value
      l.c = p1.x; // to denote vertical line x = x_value
      return 0; // we need this return variable to differentiate result
   }
   else {
      l.m = (double)(p1.y - p2.y) / (p1.x - p2.x);
      l.c = p1.y - l.m * p1.x;
      return 1; // l contains m and c of the line equation y = mx + c
   }
}

bool areParallel(line l1, line l2) { // check coefficients a & b
   return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EPS);
}

bool areSame(line l1, line l2) { // also check coefficient c
   return areParallel(l1,l2) && (fabs(l1.c - l2.c) < EPS);
}

// returns true (+ intersection point) if two lines are intersect
bool areIntersect(line l1, line l2, point &p) {
   if (areParallel(l1, l2)) return false; // no intersection
   // solve system of 2 linear algebraic equations with 2 unknowns
   p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);
   // special case: test for vertical line to avoid division by zero
   if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
   else p.y = -(l2.a * p.x + l2.c);
   return true;
}

struct vec {
   double x, y; // name: 'vec' is different from STL vector
   vec(double _x, double _y) : x(_x), y(_y) {}
};

vec toVec(point a, point b) { // convert 2 points to vector a->b
   return vec(b.x - a.x, b.y - a.y);
}

vec scale(vec v, double s) { // nonnegative s = [<1 .. 1 .. >1]
   return vec(v.x * s, v.y * s);
} // shorter.same.longer

point translate(point p, vec v) { // translate p according to v
   return point(p.x + v.x, p.y + v.y);
}

// convert point and gradient/slope to line
void pointSlopeToLine(point p, double m, line &l) {
   l.a = -m; // always -m
   l.b = 1; // always 1
   l.c = -((l.a * p.x) + (l.b * p.y));
} // compute this

void closestPoint(line l, point p, point &ans) {
   line perpendicular; // perpendicular to l and pass through p
   if (fabs(l.b) < EPS) { // special case 1: vertical line
      ans.x = -(l.c);
      ans.y = p.y;
      return;
   }

   if (fabs(l.a) < EPS) { // special case 2: horizontal line
      ans.x = p.x;
      ans.y = -(l.c);
      return;
   }

   pointSlopeToLine(p, 1 / l.a, perpendicular); // normal line
   // intersect line l with this perpendicular line
   // the intersection point is the closest point
   areIntersect(l, perpendicular, ans);
}

// returns the reflection of point on a line
void reflectionPoint(line l, point p, point &ans) {
   point b;
   closestPoint(l, p, b); // similar to distToLine
   vec v = toVec(p, b); // create a vector
   ans = translate(translate(p, v), v);
} // translate p twice

double dot(vec a, vec b) {
   return (a.x * b.x + a.y * b.y);
}

double norm_sq(vec v) {
   return v.x * v.x + v.y * v.y;
}

// returns the distance from p to the line defined by
// two points a and b (a and b must be different)
// the closest point is stored in the 4th parameter (byref)
double distToLine(point p, point a, point b, point &c) {
   // formula: c = a + u * ab
```

```
   vec ap = toVec(a, p), ab = toVec(a, b);
   double u = dot(ap, ab) / norm_sq(ab);
   c = translate(a, scale(ab, u)); // translate a to c
   return dist(p, c);
} // Euclidean distance between p and c

// returns the distance from p to the line segment ab defined by
// two points a and b (still OK if a == b)
// the closest point is stored in the 4th parameter (byref)
double distToLineSegment(point p, point a, point b, point &c) {
   vec ap = toVec(a, p), ab = toVec(a, b);
   double u = dot(ap, ab) / norm_sq(ab);
   if (u < 0.0) {
      c = point(a.x, a.y); // closer to a
      return dist(p, a);
   } // Euclidean distance between p and a
   if (u > 1.0) {
      c = point(b.x, b.y); // closer to b
      return dist(p, b);
   } // Euclidean distance between p and b
   return distToLine(p, a, b, c);
} // run distToLine as above

double angle(point a, point o, point b) { // returns angle aob in rad
   vec oa = toVec(o, a), ob = toVec(o, b);
   return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob)));
}

double cross(vec a, vec b) {
   return a.x * b.y - a.y * b.x;
}

//// another variant
//int area2(point p, point q, point r) { // returns 'twice' the area of this triangle A-B-c
// return p.x * q.y - p.y * q.x +
// q.x * r.y - q.y * r.x +
// r.x * p.y - r.y * p.x;
//}

// note: to accept collinear points, we have to change the '> 0'
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
   return cross(toVec(p, q), toVec(p, r)) > 0;
}

// returns true if point r is on the same line as the line pq
bool collinear(point p, point q, point r) {
   return fabs(cross(toVec(p, q), toVec(p, r))) < EPS;
}

// circles

int insideCircle(point_i p, point_i c, int r) { // all integer version
   int dx = p.x - c.x, dy = p.y - c.y;
   int Euc = dx * dx + dy * dy, rSq = r * r; // all integer
   return Euc < rSq ? 0 : Euc == rSq ? 1 : 2;
} //inside/border/outside

bool circle2PtsRad(point p1, point p2, double r, point &c) {
   double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
               (p1.y - p2.y) * (p1.y - p2.y);
   double det = r * r / d2 - 0.25;
   if (det < 0.0) return false;
   double h = sqrt(det);
   c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
   c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
   return true;
} // to get the other center, reverse p1 and p2

double perimeter(double ab, double bc, double ca) {
   return ab + bc + ca;
}

double perimeter(point a, point b, point c) {
   return dist(a, b) + dist(b, c) + dist(c, a);
}

double area(double ab, double bc, double ca) {
   // Heron's formula, split sqrt(a * b) into sqrt(a) * sqrt(b); in implementation
   double s = 0.5 * perimeter(ab, bc, ca);
   return sqrt(s) * sqrt(s - ab) * sqrt(s - bc) * sqrt(s - ca);
}

double area(point a, point b, point c) {
   return area(dist(a, b), dist(b, c), dist(c, a));
}

double rInCircle(double ab, double bc, double ca) {
   return area(ab, bc, ca) / (0.5 * perimeter(ab, bc, ca));
}

double rInCircle(point a, point b, point c) {
   return rInCircle(dist(a, b), dist(b, c), dist(c, a));
}

// assumption: the required points/lines functions have been written
// returns 1 if there is an inCircle center, returns 0 otherwise
// if this function returns 1, ctr will be the inCircle center
// and r is the same as rInCircle
int inCircle(point p1, point p2, point p3, point &ctr, double &r) {
   r = rInCircle(p1, p2, p3);
   if (fabs(r) < EPS) return 0; // no inCircle center

   line l1, l2; // compute these two angle bisectors
   double ratio = dist(p1, p2) / dist(p1, p3);
   point p = translate(p2, scale(toVec(p2, p3), ratio / (1 + ratio)));
   pointsToLine(p1, p, l1);

   ratio = dist(p2, p1) / dist(p2, p3);
   p = translate(p1, scale(toVec(p1, p3), ratio / (1 + ratio)));
   pointsToLine(p2, p, l2);

   areIntersect(l1, l2, ctr); // get their intersection point
   return 1;
}

double rCircumCircle(double ab, double bc, double ca) {
   return ab * bc * ca / (4.0 * area(ab, bc, ca));
}

double rCircumCircle(point a, point b, point c) {
   return rCircumCircle(dist(a, b), dist(b, c), dist(c, a));
}

// assumption: the required points/lines functions have been written
// returns 1 if there is a circumCenter center, returns 0 otherwise
```

```
// if this function returns 1, ctr will be the circumCircle center
// and r is the same as rCircumCircle
int circumCircle(point p1, point p2, point p3, point &ctr, double &r) {
   double a = p2.x - p1.x, b = p2.y - p1.y;
   double c = p3.x - p1.x, d = p3.y - p1.y;
   double e = a * (p1.x + p2.x) + b * (p1.y + p2.y);
   double f = c * (p1.x + p3.x) + d * (p1.y + p3.y);
   double g = 2.0 * (a * (p3.y - p2.y) - b * (p3.x - p2.x));
   if (fabs(g) < EPS) return 0;

   ctr.x = (d*e - b*f) / g;
   ctr.y = (a*f - c*e) / g;
   r = dist(p1, ctr); // r = distance from center to 1 of the 3 points
   return 1;
}

// returns true if point d is inside the circumCircle defined by a,b,c
int inCircumCircle(point a, point b, point c, point d) {
return (a.x - d.x) * (b.y - d.y) * ((c.x - d.x) * (c.x - d.x) +
      (c.y - d.y) * (c.y - d.y)) +
(a.y - d.y) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y) * (b.y - d.y))
      * (c.x - d.x) +
((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)) * (b.x - d.x)
      * (c.y - d.y) -
((a.x - d.x) * (a.x - d.x) + (a.y - d.y) * (a.y - d.y)) * (b.y - d.y)
      * (c.x - d.x) -
(a.y - d.y) * (b.x - d.x) * ((c.x - d.x) * (c.x - d.x) + (c.y - d.y)
      * (c.y - d.y)) -
(a.x - d.x) * ((b.x - d.x) * (b.x - d.x) + (b.y - d.y) * (b.y - d.y))
      * (c.y - d.y) > 0 ? 1 : 0;
}

bool canFormTriangle(double a, double b, double c) {
   return (a + b > c) && (a + c > b) && (b + c > a);
}

// returns the perimeter, which is the sum of Euclidian distances
// of consecutive line segments (polygon edges)
double perimeter(const vector<point> &P) {
   double result = 0.0;
   for (int i = 0; i < (int)P.size()-1; i++) // remember that P[0] = P[n-1]
      result += dist(P[i], P[i+1]);
   return result;
}

// returns the area, which is half the determinant
double area(const vector<point> &P) {
   double result = 0.0, x1, y1, x2, y2;
   for (int i = 0; i < (int)P.size()-1; i++) {
      x1 = P[i].x;
      x2 = P[i+1].x;
      y1 = P[i].y;
      y2 = P[i+1].y;
      result += (x1 * y2 - x2 * y1);
   }
   return fabs(result) / 2.0;
}

// returns true if we always make the same turn while examining
// all the edges of the polygon one by one
bool isConvex(const vector<point> &P) {
   int sz = (int)P.size();
   if (sz <= 3) return false; // a point/sz=2 or a line/sz=3 is not convex
   bool isLeft = ccw(P[0], P[1], P[2]); // remember one result
   for (int i = 1; i < sz-1; i++) // then compare with the others
      if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
         return false; // different sign -> this polygon is concave
   return true;
} // this polygon is convex

// returns true if point p is in either convex/concave polygon P
bool inPolygon(point pt, const vector<point> &P) {
   if ((int)P.size() == 0) return false;
   double sum = 0; // assume the first vertex is equal to the last vertex
   for (int i = 0; i < (int)P.size()-1; i++) {
      if (ccw(pt, P[i], P[i+1]))
         sum += angle(P[i], pt, P[i+1]); // left turn/ccw
      else sum -= angle(P[i], pt, P[i+1]);
   } // right turn/cw
   return fabs(fabs(sum) - 2*PI) < EPS;
}

// line segment p-q intersect with line A-B.
point lineIntersectSeg(point p, point q, point A, point B) {
   double a = B.y - A.y;
   double b = A.x - B.x;
   double c = B.x * A.y - A.x * B.y;
   double u = fabs(a * p.x + b * p.y + c);
   double v = fabs(a * q.x + b * q.y + c);
   return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u+v));
}

// cuts polygon Q along the line formed by point a -> point b
// (note: the last point must be the same as the first point)
vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
   vector<point> P;
   for (int i = 0; i < (int)Q.size(); i++) {
      double left1 = cross(toVec(a, b), toVec(a, Q[i])), left2 = 0;
      if (i != (int)Q.size()-1) left2 = cross(toVec(a, b), toVec(a, Q[i+1]));
      if (left1 > -EPS) P.push_back(Q[i]); // Q[i] is on the left of ab
      if (left1 * left2 < -EPS) // edge (Q[i], Q[i+1]) crosses line ab
         P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
   }
   if (!P.empty() && !(P.back() == P.front()))
      P.push_back(P.front()); // make P's first point = P's last point
   return P;
}

point pivot;
bool angleCmp(point a, point b) { // angle-sorting function
   if (collinear(pivot, a, b)) // special case
      return dist(pivot, a) < dist(pivot, b); // check which one is closer
   double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
   double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
   return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0;
} // compare two angles

vector<point> CH(vector<point> P) { // the content of P may be reshuffled
   int i, j, n = (int)P.size();
   if (n <= 3) {
      if (!(P[0] == P[n-1])) P.push_back(P[0]); // safeguard from corner case
      return P; // special case, the CH is P itself
   }

   // first, find P0 = point with lowest Y and if tie: rightmost X
   int P0 = 0;
```

```
   for (i = 1; i < n; i++)
      if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
         P0 = i;

   point temp = P[0];
   P[0] = P[P0];
   P[P0] = temp; // swap P[P0] with P[0]

   // second, sort points by angle w.r.t. pivot P0
   pivot = P[0]; // use this global variable as reference
   sort(++P.begin(), P.end(), angleCmp); // we do not sort P[0]

   // third, the ccw tests
   vector<point> S;
   S.push_back(P[n-1]);
   S.push_back(P[0]);
   S.push_back(P[1]); // initial S
   i = 2; // then, we check the rest
   while (i < n) { // note: N must be >= 3 for this method to work
      j = (int)S.size()-1;
      if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]); // left turn, accept
      else S.pop_back();
   } // or pop the top of S until we have a left turn
   return S;
} // return the result

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

Getchar Unlocked

```
#include <bits/stdc++.h>
using namespace std;

bool readChar(char &c) {
      c = getchar_unlocked();
      return c != EOF;
}

inline void writeChar(char c) {
      putchar_unlocked(c);
}

template<typename T>
bool readInt( T &n ) {
   n = 0;
   register bool neg = false;
   register char c = getchar_unlocked();
   if( c == EOF) { n = -1; return false; }
   while (!('0' <= c && c <= '9')) {
      if( c == '-' ) neg = true;
      c = getchar_unlocked();
   }
   while ('0' <= c && c <= '9') {
      n = n * 10 + c - '0';
      c = getchar_unlocked();
   }
   n = (neg ? (-n) : (n));
   return true;
}

template<typename T>
inline void writeInt(T n){
   register int idx = 20;
   if( n < 0 ) putchar_unlocked('-');
   n = abs(n);
   char out[21];
   out[20] = '␣';
   do{
      idx--;
      out[idx] = n % 10 + '0';
      n/= 10;
   }while(n);
   do{ putchar_unlocked(out[idx++]); } while (out[idx] != '␣');
}

int main() {
      ios::sync_with_stdio(false);

      int x;
      while (readInt(x)) {
            writeInt(x);
      }
      return 0;
}
```

Grundy Number

```
// Codeforces - Permutation Game - 1003C
// Grundy Number
// Game description:
/* A token is placed in one of the cells. They take
 * alternating turns of moving the token around the board,
 * with Alice moving first. The current player can move from
 * cell i to cell j only if the following two conditions are satisfied:
    ** the number in the new cell j must be strictly larger than
    ** the number in the old cell i (i.e. a[j] > a[i]), and
    ** the distance that the token travels during this turn
    ** must be a multiple of the number in the old cell
    (i.e. |i - j| mod a[i] == 0).
 * Whoever is unable to make a move, loses.
 */
/* Note: for combining various games, make nim-sum (xor-sum) of
 * grundy numbers of each game. If this value is 0, first player
 * loses, otherwise first player win.
 */
```

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 100 * 1000 + 10;
int n;
int a[N];
int memo[N];

int calculateMex(unordered_set<int> set) {
   int mex = 0;

   while (set.find(mex) != set.end())
      mex++;

   return mex;
}

int calculateGrundy(int i) {
   if (memo[i] != -1)
      return memo[i];

   if (a[i] == n)
      return 0;

   unordered_set<int> set;

   for (int j = i + a[i]; j < n; j += a[i]) {
      if (a[j] > a[i])
         set.insert(calculateGrundy(j));
   }

   for (int j = i - a[i]; j >= 0; j -= a[i]) {
      if (a[j] > a[i]) {
         set.insert(calculateGrundy(j));
      }
   }

   return memo[i] = calculateMex(set);
}

int main() {
   cin >> n;

   for (int i = 0; i < n; i++) {
      cin >> a[i];
   }

   memset(memo, -1, sizeof(memo));
   for (int i = 0; i < n; i++) {
      cout << (calculateGrundy(i) > 0 ? "A" : "B");
   }
   cout << "\n";
   return 0;
}
```

Kruskal (Minimum Spanning Tree)

```cpp
// UVA - Bond - 11354
// similar to problem "Caminhoes"
// Kruskal and queries response inside kruskal
#include <bits/stdc++.h>
using namespace std;

namespace UF {
      const int N = 100 * 1000 + 10;
      int parent[N];
      int sz[N];
      void init(int size) {
         assert(size < N);
            for (int i = 0; i < size; i++) {
                  parent[i] = i;
                  sz[i] = 1;
      }
      }

      int find(int x) {
            if (parent[x] == x) return x;
            else return parent[x] = find(parent[x]);
      }

      void join(int x, int y) {
            x = find(x);
            y = find(y);

            if (x == y) return;

      if (sz[x] < sz[y]) {
            parent[x] = y;
            sz[y] += sz[x];
      } else {
            parent[y] = x;
            sz[x] += sz[y];
      }
      }
}

struct edge {
   int x, y;
   int64_t w;
   bool operator< (const edge &other) {
      return this->w < other.w || (this->w == other.w && this->x < other.x);
   }
};

struct query {
   int x, y, ans = -1;
};

const int N = 100 * 1000 + 10;
edge edges[N];
query queries[N];
vector<int> comp_queries[N];

int main() {
      ios::sync_with_stdio(false);
      cin.tie(nullptr);
```

```
   int n, m, x, y, q;
   int64_t w;
   bool first = true;

   while (cin >> n >> m) {
      if (!first) {
         cout << "\n";
      }
      first = false;

      for (int i = 0; i < m; i++) {
         cin >> edges[i].x >> edges[i].y >> edges[i].w;
      }

      for (int i = 1; i <= n; i++) {
         comp_queries[i].clear();
      }

      cin >> q;

      for (int i = 0; i < q; i++) {
         cin >> x >> y;
         queries[i].x = x;
         queries[i].y = y;
         queries[i].ans = -1;

         comp_queries[x].push_back(i);
         comp_queries[y].push_back(i);
      }

      sort(edges, edges+m);

      UF::init(n+1);

      for (int i = 0; i < m; i++) {
         x = edges[i].x;
         y = edges[i].y;
         w = edges[i].w;

         if (UF::find(x) != UF::find(y)) {
            int small = UF::find(x),
               large = UF::find(y);

            if (UF::sz[small] >= UF::sz[large]) {
               swap(small, large);
            }

            // note: should choose small and large in the same
            // manner that union-find do

            UF::join(x, y);

            for (const auto &id : comp_queries[small]) {
               if (queries[id].ans == -1 &&
                     UF::find(queries[id].x) == UF::find(queries[id].y)) {
                  queries[id].ans = w;
               } else {
                  comp_queries[large].push_back(id);
               }
            }

            // store queries in large will maintain
            // access to queries, because UF::find return the large

            comp_queries[small].clear();
         }
      }

      for (int i = 0; i < q; i++) {
         cout << queries[i].ans << "\n";
      }
   }

   return 0;
}
```

Lowest Common Ancestor (LCA)

```
//Lib - LCA
#include <bits/stdc++.h>
using namespace std;

const int N = 100100, L = 20;
vector<pair<int, int64_t>> g[N];
int64_t dist[N];
int parent[N][L], lvl[N];

void dfs(int v, int p) {
      parent[v][0] = p;
      lvl[v] = lvl[p] + 1;

      int64_t w;
      for (int j = 0, u; j < (int)g[v].size(); j++) {
            u = g[v][j].first;
            w = g[v][j].second;

            if (u == p) continue;

            dist[u] = dist[v] + w;
            dfs(u, v);
      }
}

void pre_lca(int n) {
      for (int i = 1; i < L; i++) {
            for (int v = 0; v < n; v++) {
                  parent[v][i] = parent[parent[v][i-1]][i-1];
            }
      }
}

int lca(int a, int b) {
      if (lvl[a] < lvl[b]) {
            swap(a, b);
      }

      int d = lvl[a] - lvl[b];
```

```
      for (int i = L-1; i >= 0; i--) {
            if (d & (1 << i)) {
                  a = parent[a][i];
            }
      }

      if (a == b) return a;

      for (int i = L-1; i >= 0; i--) {
            if (parent[a][i] != parent[b][i]) {
                  a = parent[a][i];
                  b = parent[b][i];
            }
      }
      return parent[a][0];
}


int main() {
      ios::sync_with_stdio(false);

      int n;
      cin >> n;
      dfs(0, 0);
      pre_lca(n);

      return 0;
}
```

Longest Increasing Sequence

```
#include <bits/stdc++.h>
using namespace std;

int main() {
      ios::sync_with_stdio(false);
      const int N = 1000;
   int n;
   int64_t a[N];

   cin >> n;
   for (int i = 0; i < n; i++) {
      cin >> a[i];
   }

      set<int64_t> st;

      auto it = st.begin();
   for (int i = 0; i < n; i++) {
            if (st.find(a[i]) != st.end()) continue;

            st.insert(a[i]);
            it = st.find(a[i]);
            if (next(it) != st.end())
                  st.erase(next(it));
      }

      cout << st.size() << endl;

      return 0;
}
```

Longest Increasing Sequence with Fenwick Tree

```
// LIS using Fenwick Tree
#include <bits/stdc++.h>
using namespace std;

const int M = 310;
int ft[M];

void update(int x, int v) {
   x += 3;
   while (x < M) {
      ft[x] = max(ft[x], v);
      x += (x & -x);
   }
}

int query(int x) {
   x += 3;
   int ans = 0;
   while (x > 0) {
      ans = max(ans, ft[x]);
      x -= (x & -x);
   }
   return ans;
}

int dp[1000100];

int main() {
   const int N = 110;
   int n;
   int64_t a[N];

   cin >> n;
   for (int i = 0; i < n; i++) {
      cin >> a[i];
   }

   int lis = 1;

   for (int i = 0; i < n; i++) {
      dp[i] = query(a[i%n]-1) + 1;
      update(a[i%n], dp[i]);
      lis = max(lis, dp[i]);
   }
```

```
   cout << lis << endl;
   return 0;
```

Longest Increasing Sequence (Print elements)

```
// LIS algorithm with function to print the sequence
// UVA - What Goes Up - 481
#include <bits/stdc++.h>
using namespace std;

namespace LIS {
   const int N = 100 * 1000 + 10;
   int parent[N], l[N], l_id[N], l_end;
   void lis(int a[], const int n) {
      l_end = 0;
      for (int i = 0; i < n; i++) {
         int pos = lower_bound(l, l+l_end, a[i]) - l;

         l[pos] = a[i];
         l_id[pos] = i;
         parent[i] = pos ? l_id[pos-1] : -1;

         if (pos == l_end) {
            l_end++;
         }
      }
   }

   vector<int> get_lis(int a[]) {
      stack<int> st;
      for (int x = l_id[l_end-1]; x != -1; x = parent[x]) {
         st.push(a[x]);
      }
      vector<int> ans;
      while (!st.empty()) {
         ans.push_back(st.top());
         st.pop();
      }
      return ans;
   }
}

int main() {
   const int N = 100 * 1000 + 10;
   int x, n = 0;
   int a[N];

   while (cin >> x) {
      a[n++] = x;
   }

   LIS::lis(a, n);

   vector<int> ans = LIS::get_lis(a);
   cout << ans.size() << "\n-\n";
   for (const auto &x : ans) {
      cout << x << "\n";
   }

   return 0;
}
```

Matrix Power

```
// Codeforces - Gym 101845 - Univ. Nacional de Colombia PC
// Apple Trees
#include <bits/stdc++.h>
using namespace std;

const int MAXN = 5;

struct matrix {
   int64_t m[MAXN][MAXN];
};

matrix mult(matrix a, matrix b, int64_t mod) {
   matrix ans;
   for (int i = 0; i < MAXN; i++) {
      for (int j = 0; j < MAXN; j++) {
         ans.m[i][j] = 0;
         for (int k = 0; k < MAXN; k++) {
            ans.m[i][j] += (a.m[i][k] * b.m[k][j]) % mod;
         }
         ans.m[i][j] %= mod;
      }
   }

   return ans;
}

matrix power(matrix base, int64_t exp, int64_t mod) {
   matrix ans;
   for (int i = 0; i < MAXN; i++) {
      for (int j = 0; j < MAXN; j++) {
         ans.m[i][j] = (i == j) ? 1 : 0;
         base.m[i][j] %= mod;
      }
   }

   while (exp) {
      if (exp & 1) {
         ans = mult(ans, base, mod);
      }
```

```
      base = mult(base, base, mod);

      exp >>= 1;
   }

   return ans;
}

int main() {
   const int64_t mod = 1000000007;
   int64_t n;

   cin >> n;

   if (n < 10) {
      cout << "1\n";
      return 0;
   }

   matrix a = {{{16, 9, 4, 1, 0},
               {1, 0, 0, 0, 0},
               {0, 1, 0, 0, 0},
               {0, 0, 1, 0, 0},
               {0, 0, 0, 1, 0} }};

   a = power(a, n/10, mod);

   int64_t ans = 0;
   for (int i = 0; i < 4; i++) {
      ans += a.m[i][0];
      ans %= mod;
   }

   if (n % 10 < 5) {
      ans += a.m[4][0];
      ans %= mod;
   }

   cout << ans << "\n";
   return 0;
}
```

Minimum Cost Maximum Flow

```
// Min cost max flow
// UVA 10594 - Data Flow

#include <bits/stdc++.h>
using namespace std;

const int INF = numeric_limits<int>::max();
const int64_t LINF = 1000LL * 1000 * 1000 * 1000 * 1000 * 1000LL;
const int N = 150; // number of vertex
const int M = 4*5002; // number of edges

struct edge {
   // v - from vertex
   // u - to vertex
   int v, u, next;
   int64_t cap, cost;
} edges[M];

int first[N], edgenum = 0;

// initialize algorithm structs
void init(int sz = N) {
   for (int i = 0; i < sz; i++) {
      first[i] = -1;
   }
   edgenum = 0;
}

// add a directed edge v -> u and residual edge v <- u
void add_edge(int v, int u, int64_t cap, int64_t cost) {
   edge &e = edges[edgenum];
   e.v = v;
   e.u = u;
   e.cap = cap;
   e.cost = cost;
   e.next = first[v];
   first[v] = edgenum;
   edgenum++;

   // residual edge
   edge &e2 = edges[edgenum];
   e2.v = u;
   e2.u = v;
   e2.cap = 0;
   e2.cost = -cost;
   e2.next = first[u];
   first[u] = edgenum;
   edgenum++;
}

int64_t dist[N];
int in_queue[N], p[N];

// augment - walk in augmented path and update flow
// u - sink or final of path
int64_t augment(int u) {
      int64_t minimum = LINF;

      // find minimum flow in augmented path
   for (int k = p[u]; k != -1; k = p[edges[k].v]) {
      minimum = min(edges[k].cap, minimum);
      }

      // walk in augment path updating flow
   for (int k = p[u]; k != -1; k = p[edges[k].v]) {
      // forward edge
      edges[k].cap -= minimum;
      // residual edge
      edges[k^1].cap += minimum;
      }

      return minimum; // return minimum flow in augmented path
}
```

```
// s - source, t - sink, n - number of vertex [0, ..., n-1]
pair<int64_t, int64_t> mcmf(const int s, const int t, const int n) {
      int64_t mf = 0, // max flow answer
            min_cost = 0, // min cost answer
                  f = 1; // current min cost

      while (f > 0) {
            f = 0;
            for (int i = 0; i < n; i++) {
               dist[i] = LINF;
               p[i] = -1;
               in_queue[i] = 0;
      }

            dist[s] = 0;

            queue<int> q;
            q.push(s);
      in_queue[s] = 1;
            while (!q.empty()) {
                  int u = q.front();
                  q.pop();
                  in_queue[u] = 0;

         for (int k = first[u], v; k != -1; k = edges[k].next) {
            v = edges[k].u;
                        if (edges[k].cap > 0 && dist[v] > dist[u] + edges[k].cost) {
                              dist[v] = dist[u] + edges[k].cost;
                              p[v] = k;
                              if (in_queue[v] == 0) {
                                 q.push(v);
                                 in_queue[v] = 1;
                              }
                        }
                  }

            }

            // verify if bfs stop when reach sink t
      if (dist[t] != LINF) {
                  // find minimum flow in augmented path
                  f = augment(t);

                  // update max flow of network
                  min_cost += f * dist[t];
                  mf += f;
            }
      }

      return {mf, min_cost};
}


int main() {
   int a[M], b[M];
   int64_t c[M];
      int n, m;
      int64_t d, k;
      int source, sink;

      while (cin >> n >> m) {
      init(n+10);

            for (int i = 0; i < m; i++) {
               cin >> a[i] >> b[i] >> c[i];
            }

      cin >> d >> k;

            for (int i = 0; i < m; i++) {
               add_edge(a[i], b[i], k, c[i]);
               add_edge(b[i], a[i], k, c[i]);
      }

      source = 0;
      sink = n;

      add_edge(source, 1, d, 0);

            auto p = mcmf(source, sink, n+1);

            if (p.first != d) {
               printf("Impossible.\n");
      } else {
         printf("%jd\n", p.second);
      }
      }

      return 0;
}
```

Mo's Algorithm

```
// Mo's Algorithm
// Problem H - Wine Production
#include <bits/stdc++.h>
using namespace std;

// Variables, that hold current "state" of computation
long long current_answer;

// Array to store answers (because the order we achieve them is messed up)
const int N = 100500;
long long answers[N];
int BLOCK_SIZE;

int arr[N];

// We will represent each query as three numbers: L, R, idx. Idx is
// the position (in original order) of this query.
pair< pair<int, int>, int> queries[N];

unordered_map<int, int > cnt, caras;
map<int, int> ok;

// Essential part of Mo's algorithm: comparator, which we will
// use with std::sort. It is a function, which must return True
// if query x must come earlier than query y, and False otherwise.
```

```
inline bool mo_cmp(const pair< pair<int, int>, int> &x,
      const pair< pair<int, int>, int> &y) {
   int block_x = x.first.first / BLOCK_SIZE;
   int block_y = y.first.first / BLOCK_SIZE;
   if(block_x != block_y)
      return block_x < block_y;
   return x.first.second < y.first.second;
}

// When adding a number, we first nullify it's effect on current
// answer, then update cnt array, then account for it's effect again.
inline void add(int pos) {
   cnt[arr[pos]]++;
   int q = cnt[arr[pos]];
   caras[q]++;

   if (caras[q] >= q) {
      ok[q]++;
   }
   if(!ok.empty()){
      current_answer = (*(ok.end()--)).first;
   }
   else current_answer = 1;
}

// Removing is much like adding.
inline void remove(int pos) {
   int q = cnt[arr[pos]];
   cnt[arr[pos]]--;
   caras[q]--;

   if (ok.count(q)){
      ok[q]--;
      if (ok[q] == 0) {
         ok.erase(q);
      }
   }
   if(!ok.empty()){
      current_answer = (*(ok.end()--)).first;
   }
   else current_answer = 1;
}

int main() {
   cin.sync_with_stdio(false);
   cin.tie(NULL);

   int n, q;
   cin >> n >> q;
   BLOCK_SIZE = static_cast<int>(sqrt(n));

   for(int i = 0; i < n; i++)
      cin >> arr[i];

   for(int i = 0; i < q; i++) {
      cin >> queries[i].first.first >> queries[i].first.second;
      queries[i].first.first--;
      queries[i].first.second--;

      queries[i].second = i;
   }

   // Sort queries using Mo's special comparator we defined.
   sort(queries, queries + q, mo_cmp);

   // Set up current segment [mo_left, mo_right].
   int mo_left = 0, mo_right = -1;

   for(int i = 0; i < q; i++) {
      // [left, right] is what query we must answer now.
      int left = queries[i].first.first;
      int right = queries[i].first.second;

      // Usual part of applying Mo's algorithm: moving mo_left
      // and mo_right.
      while(mo_right < right) {
         mo_right++;
         add(mo_right);
      }
      while(mo_right > right) {
         remove(mo_right);
         mo_right--;
      }

      while(mo_left < left) {
         remove(mo_left);
         mo_left++;
      }
      while(mo_left > left) {
         mo_left--;
         add(mo_left);
      }

      // Store the answer into required position.
      answers[queries[i].second] = current_answer;
   }

   // We output answers *after* we process all queries.
   for(int i = 0; i < q; i++)
      cout << answers[i] << "\n";
   return 0;
}
```

Ordered Set

```
// Lib - Ordered Set - Policy-based Set
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;

typedef tree<
int ,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>pbd_set;
```

```
// map: tA -> tB com comparador less<tA>
// pode usar como um map normalmente
// s.find_by_order(k) :: retorna iterador para o k-esimo elemento (0-index) (ou s.end())
// s.order_of_key(x) :: retorna a qtd de elementos estritamente menores que x
int main(){
   pbd_set s;
   int t;
   scanf("%d␣",&t);
   while(t--){
      char typ;
      int n;
      scanf("␣%c%d",&typ , &n);
      if(typ == 'I')
         s.insert(n);
      else if (typ == 'D')
         s.erase(n);
      else if (typ == 'K'){
         int ans;
         n--;
         if(s.find_by_order(n) != s.end()){
            ans = *s.find_by_order(n);
            printf("%d\n",ans);
         }
         else
            printf("invalid\n");
      }
      else{
         int ans = s.order_of_key(n);
         printf("%d\n",ans);
      }
   }
   return 0;
}
```

Random Numbers

```
#include <algorithm>
#include <chrono>
#include <iostream>
#include <random>
#include <vector>
using namespace std;

const int N = 3000000;

double average_distance(const vector<int> &permutation) {
   double distance_sum = 0;

   for (int i = 0; i < N; i++)
      distance_sum += abs(permutation[i] - i);

   return distance_sum / N;
}

int main() {
   mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
   vector<int> permutation(N);

   for (int i = 0; i < N; i++)
      permutation[i] = i;

   shuffle(permutation.begin(), permutation.end(), rng);
   cout << average_distance(permutation) << '\n';

   for (int i = 0; i < N; i++)
      permutation[i] = i;

   for (int i = 1; i < N; i++)
      swap(permutation[i], permutation[uniform_int_distribution<int>(0, i)(rng)]);

   cout << average_distance(permutation) << '\n';
}
```

Rotating Calipers (Diameter of a polygon)

```
#include <bits/stdc++.h>
using namespace std;

typedef pair<int, int> Point;

int cross(Point a, Point b, Point c) {
   return a.first * (b.second - c.second)
         + b.first * (c.second - a.second)
         + c.first * (a.second - b.second);
}

const int N = 100100;
pair<int, int> p[N], p2[N], upper[N], lower[N];
int uppersize = 0, lowersize = 0;

void ch(int n) {
   sort(p, p+n);
   uppersize = 0;
   lowersize = 0;

   for (int i = 0; i < n; i++) {
      while (uppersize >= 2 && cross(upper[uppersize-2], upper[uppersize-1], p[i]) >= 0) {
         uppersize--;
      }

      upper[uppersize++] = p[i];
   }

   for (int i = 0; i < n; i++) {
      while (lowersize >= 2 && cross(lower[lowersize-2], lower[lowersize-1], p[i]) <= 0) {
         lowersize--;
      }
```

```
      lower[lowersize++] = p[i];
   }
}

double dist(Point a, Point b) {
   return hypot(a.first - b.first, a.second - b.second);
}

int main() {
   ios::sync_with_stdio(false);
   cin.tie(nullptr);

   int c;

   cin >> c;

   for (int i = 0; i < c; i++) {
      cin >> p2[i].first >> p2[i].second;
   }

   sort(p2, p2+c);

   int n = 0;

   p[n++] = p2[0];

   for (int i = 1; i < c; i++) {
      if (p2[i] != p2[i-1]) {
         p[n++] = p2[i];
      }
   }

   ch(n);

   int i = 0, j = lowersize-1;
   double ans = 0;

   while (i < uppersize-1 || j > 0) {
      ans = max(ans, dist(upper[i], lower[j]));

      if (i == uppersize-1) {
         j--;
      } else if (j == 0) {
         i++;
      } else {
         if ((upper[i+1].second - upper[i].second) * (lower[j].first - lower[j-1].first)
            > (lower[j].second - lower[j-1].second) * (upper[i+1].first - upper[i].first)) {
            i++;
         } else {
            j--;
         }
      }
   }

   cout << setprecision(10) << fixed;
   cout << ans << "\n";

   return 0;
}
```

Strongly Connected Components (SCC)

```
// Strongly Connected Components - Kosaraju and Sharir Algorithm
#include <bits/stdc++.h>
using namespace std;

const int N = 100100;
vector<int> g[N], gr[N];
bool used[N];
vector<int> order, component;

void dfs1(int v) {
   used[v] = true;
   for (const auto &u : g[v]) {
      if (!used[u]) {
         dfs1(u);
      }
   }
   order.push_back(v);
}

void dfs2(int v) {
   used[v] = true;
   component.push_back(v);
   for (const auto &u : gr[v]) {
      if (!used[u]) {
         dfs2(u);
      }
   }
}

void kosaraju(int n) {
   memset(used, false, sizeof(used));
   for (int v = 0; v < n; v++) {
      if (!used[v]) {
         dfs1(v);
      }
   }

   memset(used, false, sizeof(used));
   component.clear();

   for (int i = 0; i < n; i++) {
      int v = order[n - 1 - i];
      if (!used[v]) {
         dfs2(v);
         cout << "Component:";
         for (const auto &u : component) {
            cout << "␣" << u;
         }
         cout << endl;
         component.clear();
      }
   }
}
```

```cpp
int main() {
   int n, m;

   cin >> n >> m;

   for (int i = 0, x, y; i < m; i++) {
      cin >> x >> y; // x -> y
      g[x].push_back(y);
      gr[y].push_back(x); // reverse graph
   }

   kosaraju(n);
   return 0;
}
```

Sieve

```cpp
#include <bits/stdc++.h>
using namespace std;

const int M = 10000010;
bool notprime[M];
int fp[M]; // smallest first prime factor
vector<int> primes;

void sieve(int n) {
   assert(n < M);
   notprime[0] = notprime[1] = true;

   for (int i = 2; i <= n; i++) {
      if (!notprime[i]) {
         fp[i] = i;
         for (int j = i+i; j <= n; j += i) {
            notprime[j] = true;
            fp[j] = i;
         }
         primes.push_back(i);
      }
   }
}

bool isPrime(vector<int> &primes, int64_t n) {
   if (n < M) return !notprime[n];

   for (const auto &p : primes) {
      if (n % p == 0)
         return false;
   }

   return true;
}

int main() {
   int n;

   cin >> n;

   sieve(n);

   return 0;
}
```

String Hashing

```cpp
// String Hash - Polynomial rolling hash
#include <bits/stdc++.h>
using namespace std;

const int N = 200100;

struct Hash {
   uint64_t p, mod, ppow[N], ppow2[N], p2, mod2;
   pair<uint64_t, uint64_t> h[N];

   Hash() : p(33), mod(1000000007), p2(73), mod2(1000000009) {
      ppow[0] = 1;
      ppow2[0] = 1;
      for (int i = 1; i < N; i++) {
         ppow[i] = (ppow[i-1] * p) % mod;
         ppow2[i] = (ppow2[i-1] * p2) % mod2;
      }
   }

   void init(string &s) {
      h[0] = {5389ULL, 5389ULL};
      for (size_t i = 0; i < s.size(); i++) {
         int code = s[i];
         h[i+1].first = (h[i].first * p + code) % mod;
         h[i+1].second = (h[i].second * p2 + code) % mod2;
      }
   }

   pair<uint64_t, uint64_t> get_hash(int i, int j) {
      pair<uint64_t, uint64_t> r;
      r.first = (h[j+1].first - (h[i].first * ppow[j-i+1]) % mod + mod) % mod;
      r.second = (h[j+1].second - (h[i].second * ppow2[j-i+1]) % mod2 + mod2) % mod2;
      return r;
   }
};

template<typename T>
ostream& operator<<(ostream& os, pair<T, T> p) {
   os << "(" << p.first << ", " << p.second << ")";
   return os;
}
```

```
Hash h1, h2;
string s1, s2;

int main() {
   ios::sync_with_stdio(false);
   cin.tie(nullptr);

   s1 = "abghABCDE";
   s2 = "CD";
   h1.init(s1);
   h2.init(s2);

   cout << s1.substr(6, 2) << "␣==␣" << s2.substr(0, 2) << endl;
   cout << h1.get_hash(6, 7) << "␣==␣" << h2.get_hash(0, 1) << endl;

   return 0;
}
```

Suffix Array

```
// Lib - Suffix Array - Linear Time (O(n)) construction

#include <bits/stdc++.h>
using namespace std;

const int N = 400100, M = 400;

template <typename T>
using min_heap = priority_queue<T, vector<T>, greater<T>>;

// lexicographic order for pairs
inline bool leq(int a1, int a2, int b1, int b2) {
   return a1 < b1 || (a1 == b1 && a2 <= b2);
}

// and triples
inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3) {
   return a1 < b1 || (a1 == b1 && leq(a2,a3, b2,b3));
} // and triples

// stably sort a[0..n-1] to b[0..n-1] with keys in 0..K from r
static void radixPass(int* a, int* b, int* r, int n, int K) {// count occurrences
   int* c = new int[K + 1]; // counter array
   for (int i = 0; i <= K; i++) c[i] = 0; // reset counters
   for (int i = 0; i < n; i++) c[r[a[i]]]++; // count occurrences
   for (int i = 0, sum = 0; i <= K; i++) // exclusive prefix sums
   {
      int t = c[i];
      c[i] = sum;
      sum += t;
   }
   for (int i = 0; i < n; i++) b[c[r[a[i]]]++] = a[i]; // sort
   delete [] c;
}

// find the suffix array SA of s[0..n-1] in {1..K}^n
// require s[n]=s[n+1]=s[n+2]=0, n>=2
void suffixArray(int* s, int* SA, int n, int K) {
   s[n] = s[n+1] = s[n+2] = 0;

   int n0 = (n+2)/3, n1 = (n+1)/3, n2 = n/3, n02 = n0+n2;
   int* s12 = new int[n02+3]; s12[n02] = s12[n02+1] = s12[n02+2] = 0;
   int* SA12 = new int[n02+3]; SA12[n02] = SA12[n02+1] = SA12[n02+2] = 0;
   int* s0 = new int[n0];
   int* SA0 = new int[n0];
   // generate positions of mod 1 and mod 2 suffixes
   // the "+(n0-n1)" adds a dummy mod 1 suffix if n%3 == 1
   for (int i=0, j=0; i < n + (n0-n1); i++)
      if (i%3 != 0) s12[j++] = i;
   // lsb radix sort the mod 1 and mod 2 triples
   radixPass(s12 , SA12, s+2, n02, K);
   radixPass(SA12, s12 , s+1, n02, K);
   radixPass(s12 , SA12, s , n02, K);
   // find lexicographic names of triples
   int name = 0, c0 = -1, c1 = -1, c2 = -1;
   for (int i = 0; i < n02; i++) {
      if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2) {
         name++;
         c0 = s[SA12[i]];
         c1 = s[SA12[i]+1];
         c2 = s[SA12[i]+2];
      }
      if (SA12[i]%3 == 1) s12[SA12[i]/3] = name; // left half
      else s12[SA12[i]/3 + n0] = name; // right half
   }
   // recurse if names are not yet unique
   if (name < n02) {
      suffixArray(s12, SA12, n02, name);
      // store unique names in s12 using the suffix array
      for (int i = 0; i < n02; i++) s12[SA12[i]] = i + 1;
   } else // generate the suffix array of s12 directly
      for (int i = 0; i < n02; i++) SA12[s12[i] - 1] = i;
   // stably sort the mod 0 suffixes from SA12 by their first character
   for (int i = 0, j = 0; i < n02; i++)
      if (SA12[i] < n0) s0[j++] = 3*SA12[i];
   radixPass(s0, SA0, s, n0, K);
   // merge sorted SA0 suffixes and sorted SA12 suffixes
   for (int p = 0, t = n0-n1, k = 0; k < n; k++) {
      #define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
      int i = GetI(); // pos of current offset 12 suffix
      int j = SA0[p]; // pos of current offset 0 suffix
      if (SA12[t] < n0 ? // different compares for mod 1 and mod 2 suffixes
         leq(s[i], s12[SA12[t] + n0], s[j], s12[j/3]) :
         leq(s[i],s[i+1],s12[SA12[t]-n0+1], s[j],s[j+1],s12[j/3+n0]))
      {// suffix from SA12 is smaller
         SA[k] = i; t++;
         if (t == n02) // done --- only SA0 suffixes left
         for (k++; p < n0; p++, k++) SA[k] = SA0[p];
      } else {// suffix from SA0 is smaller
         SA[k] = j; p++;
         if (p == n0) // done --- only SA12 suffixes left
         for (k++; t < n02; t++, k++) SA[k] = GetI();
      }
   }
   delete [] s12; delete [] SA12; delete [] SA0; delete [] s0;
}

void lcp_construction(int *s, int n, int *sa, int *rnk, int *lcp) {
```

```
      for (int i = 0; i < n; i++) {
            rnk[sa[i]] = i;
      }

      int k = 0;
      for (int i = 0; i < n-1; i++) {
            lcp[i] = 0;
      }

      for (int i = 0; i < n; i++) {
            if (rnk[i] == n - 1) {
                  k = 0;
                  continue;
            }

            int j = sa[rnk[i] + 1];
            while (i + k < n && j + k < n && s[i+k] == s[j+k])
                  k++;
            lcp[rnk[i]] = k;
            if (k) k--;
      }
}

int sa[N], t[N], revsa[N], lcp[N];
char s[N];

int main() {
   int n;

   scanf("%d", &n);

   for (int i = 0; i < n; i++) {
      int64_t m = 0;
      scanf("%s", s);
      m = strlen(s);

      for (int i = 0; i < m; i++) {
         t[i] = s[i];
      }

      suffixArray(t, sa, m, M);
      lcp_construction(t, m, sa, revsa, lcp);

      /*
      for (size_t i = 0; i < strlen(s); i++) {
         cout << sa[i] << "\n";
      }
      */

      int64_t ans = (m * m + m) / 2;

      for (int i = 0; i < m-1; i++) {
         ans -= lcp[i];
      }

      cout << ans << "\n";
   }
   return 0;
}
```

Suffix Automata

```
// Suffix Automaton
// LCS problem
#include <bits/stdc++.h>
using namespace std;

const int N = 200100, E = 270;

struct SuffixAutomaton {
   int edges[N][E];
   int link[N];
   int lenght[N];
   bool term[N];
   int last, sz;
   int states;

   SuffixAutomaton() {
      memset(edges, -1, sizeof(edges));
      // add the initial node
      states = 1;
      link[0] = -1;
      lenght[0] = 0;
      term[0] = false;
      last = 0; // initiate with index of first state
      sz = 0; // lenght of longest suffix added to automaton
   }

   SuffixAutomaton(string s) : SuffixAutomaton() {
      for (const auto &ch : s) {
         extend(ch);
      }
      find_terminals();
   }

   void extend(char ch) {
      // create new state for new equivalence (end-points) class
      sz++;
      lenght[states] = sz;
      link[states] = 0;
      term[states] = false;
      states++;

      int r = states - 1;
      int p = last;

      while (p >= 0 && edges[p][(int)ch] == -1) {
         edges[p][(int)ch] = r;
         p = link[p];
      }

      if (p != -1) {
         int q = edges[p][(int)ch];
         if (lenght[p] + 1 == lenght[q]) {
            link[r] = q;
         } else {
            for (int e = 0; e < E; e++) {
               edges[states][e] = edges[q][e];
```

```
            }
            lenght[states] = lenght[p] + 1;
            link[states] = link[q];
            term[states] = false;
            states++;

            int qq = states - 1;
            link[q] = qq;
            link[r] = qq;

            while (p >= 0 && edges[p][(int)ch] == q) {
               edges[p][(int)ch] = qq;
               p = link[p];
            }
         }
      }

      last = r;
   }

   void find_terminals() {
      memset(term, 0, sizeof(term));
      int p = last;
      while (p >= 0) {
         term[p] = true;
         p = link[p];
      }
   }
};

void dfs(SuffixAutomaton &sa, int occur[], int words[], int st) {
   if (occur[st] > 0)
      return;

   int64_t occ = 0, wrd = 0;

   if (sa.term[st]) {
      occ++;
      wrd++;
   }

   for (int e = 0; e < E; e++) {
      if (sa.edges[st][e] != -1) {
         int newst = sa.edges[st][e];
         dfs(sa, occur, words, newst);
         occ += occur[newst];
         wrd += words[newst] + occur[newst];
      }
   }

   occur[st] = occ;
   words[st] = wrd;
}

int64_t lcs(SuffixAutomaton &sa, string t) {
   int st = 0, len = 0, best = 0, best_i = 0;

   for (int i = 0, sz = t.size(); i < sz; i++) {
      char ch = t[i];

      while (st != 0 && sa.edges[st][(int)ch] == -1) {
         st = sa.link[st];
         len = sa.lenght[st];
      }

      if (sa.edges[st][(int)ch] != -1) {
         st = sa.edges[st][(int)ch];
         len++;
      }

      if (best < len) {
         best = len;
         best_i = i;
      }
   }

   cout << t.substr(best_i-best+1, best) << endl;

   return best;
}

string find_kth_substr(SuffixAutomaton &sa, int k) {
   int occur[N], words[N];
   dfs(sa, occur, words, 0);

   // find kth substring

   int st = 0;
   string t = "";

   int64_t prev_k = k;
   while (k > 0) {
      int64_t acc = 0, tmp;

      for (int e = 0; e < E; e++) {
         if (sa.edges[st][e] != -1) {
            int newst = sa.edges[st][e];
            tmp = acc;
            acc += words[newst];
            if (acc >= k) {
               st = newst;
               k -= tmp + occur[newst];
               t += static_cast<char>(e);
               break;
            }
         }
      }

      if (k == prev_k) {
         t = "No such line.";
         break;
      }

      prev_k = k;
   }

   return t;
}

int main() {
   ios::sync_with_stdio(false);
   string s, t;

   cin >> s >> t;

   SuffixAutomaton am(s);
```

```
   cout << lcs(am, t) << "\n";

   // find kth substring in lexicographical order
   cout << find_kth_substr(am, 1) << endl;
   return 0;
}
```

Topological Sort

```
// Topological Sort
#include <bits/stdc++.h>
using namespace std;

const int N = 100100;
vector<int> g[N];
bool vis[N];
vector<int> ans;

void dfs(int v) {
   vis[v] = true;
   for (const auto &u : g[v]) {
      if (!vis[u]) {
         dfs(u);
      }
   }
   ans.push_back(v);
}

void topological_sort(int n) {
   memset(vis, false, sizeof(vis));
   for (int v = 0; v < n; v++) {
      if (!vis[v]) {
         dfs(v);
      }
   }

   reverse(ans.begin(), ans.end());
   // Topological sort is on vector ans
}

int main() {
   int n, m;
   cin >> n >> m;
   for (int i = 0, x, y; i < m; i++) {
      cin >> x >> y; // x -> y
      g[x].push_back(y);
   }

   topological_sort(n);

   cout << "Topological Order:";
   for (int i = 0; i < n; i++) {
      cout << " " << ans[i];
   }
   cout << endl;
   return 0;
}
```

Union Find

```
#include <bits/stdc++.h>
using namespace std;

// versao com namespace
namespace UF {
      const int N = 100 * 1000 + 10;
      int parent[N];
      int sz[N];
      void init(int size) {
         assert(size < N);
            for (int i = 0; i < size; i++) {
                  parent[i] = i;
                  sz[i] = 1;
      }
      }
      int find(int x) {
            if (parent[x] == x) return x;
            else return parent[x] = find(parent[x]);
      }
      void join(int x, int y) {
            x = find(x);
            y = find(y);
            if (x == y) return;

         if (sz[x] < sz[y]) {
            parent[x] = y;
            sz[y] += sz[x];
         } else {
            parent[y] = x;
            sz[x] += sz[y];
      }
      }
}

// versao usando um struct
const int N = 100 * 1000 + 10;
struct union_find {
   int parent[N];
   int sz[N];

   union_find() {
   }

   void init(int size) {
      assert(size < N);
```

```
      for (int i = 0; i < size; i++) {
         parent[i] = i;
         sz[i] = 1;
      }
   }

   int find(int x) {
      if (parent[x] == x)
         return x;
      else
         return parent[x] = find(parent[x]);
   }

   void join(int x, int y) {
            x = find(x);
            y = find(y);

            if (x == y) return;

      if (sz[x] < sz[y]) {
         parent[x] = y;
         sz[y] += sz[x];
      } else {
         parent[y] = x;
         sz[x] += sz[y];
      }
   }
};

int main() {
      ios::sync_with_stdio(false);

      int n;
      cin >> n;

   UF::init(n);

   union_find uf;
   uf.init(n);

      return 0;
}
```

Suffix Automata's Applications

Problem: Find whether a given string w is a substring of s.
Solution: Simply run the automaton.

```
SuffixAutomaton a(s);
bool fail = false;
int n = 0;
for(int i=0;i<w.size();i++) {
  if(a.edges[n].find(w[i]) == a.edges[n].end()) {
    fail = true;
    break;
  }
  n = a.edges[n][w[i]];
}
if(!fail) cout << w << " is a substring of " << s << "\n";
```

Problem: Find whether a given string w is a suffix of s.
Solution: Construct the list of terminal states, run the automaton as above and check in the end if the n is among the terminal states. Let's now look at the dp problems.

Problem: Count the number of distinct substrings in s.
Solution: The number of distinct substrings is the number of different paths in the automaton. These can be calculated recursively by calculating for each node the number of different paths starting from that node. The number of different paths starting from a node is the sum of the corresponding numbers of its direct successors, plus 1 corresponding to the path that does not leave the node.

Problem: Count the number of times a given word w occurs in s.
Solution: Similar to the previous problem. Let p be the node in the automaton that we end up while running it for w. This time the number of times a given word occurs is the number of paths starting from p and ending in a terminal node, so one can calculate recursively the number of paths from each node ending in a terminal node.

Problem: Find where a given word w occurs for the first time in s.
Solution: This is equivalent to calculating the longest path in the automaton after reaching the node p (defined as in the previous solution).

Finally let's consider the following problem where the suffix links come handy.

Problem: Find all the positions where a given word w occurs in s.

Solution: Prepend the string with some symbol '$' that does not occur in the string and construct the suffix automaton. Let's then add to each node of the suffix automaton its children in the suffix tree:

```
children=vector<vector<int>>(link.size());
for(int i=0;i<link.size();i++) {
  if(link[i] >= 0) children[link[i]].push_back(i);
}
```

Now find the node p corresponding to the node w as has been done in the previous problems. We can then dfs through the subtree of the suffix tree rooted at p by using the children vector. Once we reach a leaf, we know that we have found a prefix of s that ends in w, and the length of the leaf can be used to calculate the position of w. All of the dfs branches correspond to different prefixes, so no unnecessary work is done and the complexity is $O(|s| + |w| + sizeofoutput)$.

Combinatorics

Binomial Coefficients

Number of ways to pick a multiset of sike k from n elements: $\binom{n+k-1}{k}$
Number of n-tuples of non-negative integers with sum s: $\binom{s+n-1}{n-1}$, at most s: $\binom{s+n}{n}$
Number of n-tuples of positive integers with sum s: $\binom{s-1}{n-1}$
Number of lattice paths from $(0,0)$ to (a,b), restricted to east and north steps: $\binom{a+b}{a}$
$v_{r,c} = v_{r,c-1}\frac{r+1-c}{c}$
$x = r_0$; $y = 1$; $a_{r,c} = a_{r-1,c-1}\frac{++x}{++y}, r \geq r_0 + 2, c \geq 2$

Catalan numbers

$C_n = \frac{1}{n+1}\binom{2n}{n}$. $C_0 = 1$, $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$. $C_{n+1} = C_n\frac{4n+2}{n+2}$.
$C_0, C_1, ... = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, ...$
C_n is the number of: properly nested sequences of n pais of parentheses; rooted ordered binary trees with $n+1$ leaves; triangulations of a convex $(n+2)$-gon.

Derangements

Number of permutations of $n = 0, 1, 2, ...$ elements without fixed points is 1, 0, 1, 2, 9, 44, 265, 1854, 14833, ... Recurrence: $D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$.
Corollary: number of permutations with exactly k fixed points is $\binom{n}{k}D_{n-k}$.

Bell numbers

B_n is the number of partitions of n elements. $B_0, ... = 1, 1, 2, 5, 15, 52, 203, ...$
$B_{n+1} = \sum_{k=0}^{n}\binom{n}{k}B_k = \sum_{k=1}^{n} S_{n,k}$. Bell triangle: $B_r = a_{r,1} = a_{r-1,r-1}, a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

Eulerian numbers

$E(n,k)$ is the number of permutations with exactly k descents ($i : \pi_i < \pi_{i+1}$) / ascents ($\pi_i > \pi_{i+1}$) / excedances ($\pi_i > i$) / $k+1$ weak excedances ($\pi_i \geq i$).
Formula: $E(n,k) = (k+1)E(n-1,k) + (n-k)E(n-1,k-1)$.

Burnside's lemma

The number of orbits under G's action on set X : $|X/G| = \frac{1}{|G|}\sum_{g\in G}|X_g|$, where $X_g = \{x \in X : g(x) = x\}$. ("Average number of fixed points.") Let $w(x)$ be weight of x's orbit. Sum of all orbits' weights: $\sum_{o\in X/G} w(o) = \frac{1}{|G|}\sum_{g\in G}\sum_{x\in X_g} w(x)$.

Number Theory

Linear diophantine equation

$ax + by = c$. Let $d = gcd(a, b)$. A solution exists iff $d|c$. If (x_0, y_0) is any solution, then all solutions are given by $(x, y) = (x_0 + \frac{b}{d}t, y_0 - \frac{a}{d}t), t \in \mathbb{Z}$. To find some solution (x_0, y_0), use extended GCD to solve $ax_0 + by_0 = d = gcd(a, b)$, and multiply its solutions by $\frac{c}{d}$.
Linear diophantine equation in n variables: $a_1x_1 + ... + a_nx_n = c$ has solutions iff $gcd(a_1, ..., a_n)|c$. To find some solution, let $b = gcd(a_2, ..., a_n)$, solve $a_1x_1 + by = c$, and iterate with $a_2x_2 + ... = y$.

$a^{-1}(mod\ n) = a^{\phi(n)-1}$

$n\ prime \rightarrow a^{-1}(mod\ n) = a^{n-2}$

Chinese Remainder Theorem

System $x \equiv a_i(mod\ m_i)$ for $i = 1, ..., n$ with pairwise relatively prime m_i has a unique solution modulo $M = m_1m_2...m_n : x = a_1b_1\frac{M}{m_1} + ... + a_nb_n\frac{M}{m_n}(mod\ M)$, where b_i is modular inverse of $\frac{M}{M_i}$ modulo m_i.
System $x \equiv a(mod\ m), x \equiv b(mod\ n)$ has solutions iff $a \equiv b(mod\ g)$, where $g = gcd(m, n)$. The solution is unique modulo $L = \frac{mn}{g}$, and equals: $x \equiv a + T(b - a)m/g \equiv b + S(a - b)n/g(mod\ L)$, where S and T are integer solutions of $mT + nS = gcd(m, n)$.

Prime-counting function

$\pi(n) = |\{p \le n : p\ is\ prime\}|$. $n/ln(n) < \pi(n) < 1.3n/ln(n)$. $\pi(1000) = 168, \pi(10^6) = 78498, \pi(10^9) = 50847534$. n-th prime $\approx nln(n)$.

Miller-Rabin's primality test

Given $n = 2^rs + 1$ with odd s, and a random integer $1 < a < n$. If $a^s \equiv 1(mod\ n)$ or $a^{2^js} \equiv -1(mod\ n)$ for some $0 \le j \le r - 1$, then n is a probable prime.

Pollard-ρ

Choose random x_1, and let $x_{i+1} = x_i^2(mod\ n)$. Test $gcd(n, x_{2^k+i} - x_{2^k})$ as possible n's factors for $k = 0, 1, ...$ Expected time to find a factor: $O(\sqrt{m})$, where m is smallest prime power in n's factorization. That's $O(n^{1/4})$ if you check $n = p^k$ as a special case before factorization.

Fermat primes

A Fermat prime is a prime of form $2^{2^n} + 1$. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form $2^n + 1$ is prime only if it is a Fermat prime.

Perfect numbers

$n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

Carmichael numbers

A positive composite n is a Carmichael number ($a^{n-1} \equiv 1(mod\ n)$ for all $a : gcd(a, n) = 1$), iff n is square-free, and for all prime divisors p of n, $p - 1$ divides $n - 1$.

Number/sum of divisors

$\tau(p_1^{a_1}...p_k^{a_k}) = \prod_{j=1}^{k}(a_j + 1)$.
$\sigma(p_1^{a_1}...p_k^{a_k}) = \prod_{j=1}^{k} \frac{p_j^{a_j+1}-1}{p_j-1}$.

Euler's phi function

$\phi(n) = |\{m \in \mathbb{N}, m \leq n, gcd(m, n) = 1\}|$.
$\phi(mn) = \frac{\phi(m)\phi(n)gcd(m,n)}{\phi(gcd(m,n))}$.
$\phi(p^a) = p^{a-1}(p - 1)$.
$\sum_{d|n} \phi(d) = \sum_{d|n} \phi(\frac{n}{d}) = n$.

Euler's Theorem

$a^{\phi(n)} \equiv 1(mod\ n)$, if $gcd(a, n) = 1$.

Wilson's Theorem

p is prime iff $(p - 1)! \equiv -1(mod\ p)$

Mobius function

$\mu(1) = 1$.
$\mu(n) = 0$, if n is not square free.
$\mu(n) = (-1)^s$, if n is the product of s distinct primes.
Let f, F be functions on positive integers. If for all $n \in \mathbb{N}$, $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d)F(\frac{n}{d})$, and vice versa.
$\phi(n) = \sum_{d|n} \mu(d)\frac{n}{d}$.
$\sum_{d|n} \mu(d) = 1$.
If f is multiplicative, then $\sum_{d|n} \mu(d)f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n}(1 + f(p))$
f[i] = 1, f[p] = -1, f[j] *= (j%(i*i)==0)? 0 : -1;

Legendre symbol

If p is an odd prime, $a \in \mathbb{Z}$, then $(\frac{a}{p})$ equals 0, if $p|a$; 1 if a is a quadratic residue modulo p; and -1 otherwise. Euler's criterion: $(\frac{a}{p}) = a^{(\frac{p-1}{2})}(mod\ p)$.

Jacobi symbol

If $n = p_1^{a_1}...p_k^{a_k}$ is odd, then $(\frac{a}{n}) = \prod_{i=1}^{k}(\frac{a}{p_i})^{k_i}$.

Primitive roots

If the order of g modulo m ($\min n > 0 : g^n \equiv 1 (mod\ m)$) is $\phi(m)$, then g is called a primitive root. If $\mathbb{Z}_m$ has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. $\mathbb{Z}_m$ has a primitive root iff m is one of $2, 4, p^k, 2p^k$, where p is an odd prime. If $\mathbb{Z}_m$ has a primitive root g, then for all a coprime to m, there exists unique integer $i = ind_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a (mod\ m)$. $ind_g(a)$ has logarithm-like properties: $ind(1) = 0$, $ind(ab) = ind(a) + ind(b)$.
If p is prime and a is not divisible by p, then congruence $x^n \equiv a (mod\ p)$ has $gcd(n, p-1)$ solutions if $a^{(p-1)/gcd(n,p-1)} \equiv 1 (mod\ p)$, and no solutions otherwise.

Discrete log

Find x from $a^x \equiv b (mod\ m)$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n = \lceil \sqrt{m} \rceil$, and $x = ny - z$. Equation becomes $a^{ny} \equiv ba^z (mod\ m)$. Precompute all values that the RHS can take for z = 0,1,...,n-1, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

Pythagorean triples

Integer solutions of $x^2 + y^2 = z^2$. All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n$, $gcd(m, n) = 1$ and $m \not\equiv n (mod\ 2)$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

Postage stamps/McNuggets problem

Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax + by (x, y \geq 0)$, and the largest is $(a-1)(b-1) - 1 = ab - a - b$.

Fermat's two-squares theorem

Odd prime p can be represented as a sum of two squares iff $p \equiv 1 (mod\ 4)$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k + 3$ occurs an even number of times in n's factorization.

Graphs

Euler's theorem

For any planar graph, $V - E + F = 1 + C$, where V is the number of graph's vertices, E is the number of edges, F is the number of faces in graph's planar drawing, and C is the number of connected components. Corollary: $V - E + F = 2$ for a 3D polyhedron.

Vertex covers and independent sets

Let M, C, I be a max matching, a min vertex cover, and a max independent set. Then $|M| \leq |C| = N - |I|$, with equality for bipartite graphs. Complement of an MVC is always a MIS, and vice versa. Given a bipartite graph with partitions (A, B), build a network: connect source to A, and B to sink with edges of capacities, equal to the corresponding nodes' weights, or 1 in the unweighted case. Set capacities of the original graph's edges to the infinity. Let (S, T) be a minimum $s - t$ cut. Then a maximum(-weighted) independent set is $I = (A \bigcap S) \bigcup (B \bigcap T)$, and a minimum(-weighted) vertex cover is $C = (A \bigcap T) \bigcup (B \bigcap S)$.

Matrix-tree theorem

Let matrix $T = [t_{ij}]$, where t_{ij} is the number of multiedges between i and j, for $i \neq j$, and $t_{ii} = -deg_i$. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T.

Prufer code of a tree

Label vertices with integers 1 to n. Repeatedly remove the leaf with the smallest label, and output its only neighbor's label, until only one edge remains. The sequence has length $n-2$. Two isomorphic trees have the same sequence, and every sequence of integers from 1 and n corresponds to a tree. Corollary: the number of labelled trees with n vertices is n^{n-2}.

Euler tours

Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected.

Stable marriage problem

While there is a free man m: let w be the most-preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

Stoer-Wagner

```
/* Stoer-Wagner Min Cut on undirected graph */
#define MAXV 101
#define INF 0x3F3F3F3F

int grafo[MAXV][MAXV];

// v[i] representa o vertice original do grafo correspondente
// ao i-esimo vertice do grafo da fase atual do minCut e w[i]
// tem o peso do vertice v[i]..
int v[MAXV], w[MAXV];
int A[MAXV];



int minCut(int n) {
      if (n == 1) return 0;

      int i, u, x, s, t;
      int minimo;

      for (u = 1; u <= n; ++u) { v[u] = u; }

      w[0] = -1;
      minimo = INF;




      while (n > 1) {

      for (u = 1; u <= n; ++u) {
                  A[v[u]] = 0;
                  w[u] = grafo[v[1]][v[u]];
            }

            A[v[1]] = 1;
            s = v[1];

            for (u = 2; u <= n; ++u) {
                  // Encontra o mais fortemente conetado a A
                  t = 0;
                  for (x = 2; x <= n; ++x)
                        if (!A[v[x]] && (w[x] > w[t]))
                              t = x;

                  // adiciona ele a A
                  A[v[t]] = 1;




                  if (u == n) {
                        if (w[t] < minimo)
                              minimo = w[t];

                        // Une s e t
                        for (x = 1; x <= n; ++x) {
                              grafo[s][v[x]] += grafo[v[t]][v[x]];
                              grafo[v[x]][s] = grafo[s][v[x]];
                  }

                        v[t] = v[n--];
                        break;
            }
            s = v[t];

                  // Atualiza os pesos
                  for (x = 1; x <= n; ++x)
                        w[x] += grafo[v[t]][v[x]];
            }
      }
      return minimo;
}
```

Start from a set A containing an arbitrary vertex. While $A \neq V$, add to A the most tightly connected vertex ($z \notin A$ such that $\sum_{x \in A} w(x, z)$ is maximized.) Store cut-of-the-phase (the cut between the last added vertex and rest of the graph), and merge the two vertices added last. Repeat until the graph is contracted to a single vertex. Minimum cut is one of the cuts-of-the-phase.

Erdos-Gallai theorem

A sequence of integers $\{d_1, d_2, ..., d_n\}$, with $n - 1 \geq d_1 \geq d_2 \geq ... \geq d_n \geq 0$ is a degree sequence of some undirected simple graph iff $\sum d_i$ is even and $d_1 + ... + dk \leq k(k-1) + \sum_{i=k+1}^{n} min(k, d_i)$ for all $k = 1, 2, ..., n - 1$.

Dilworth's theorem

In any finite partially ordered set, the maximum number of elements in any antichain equals the minimum number of chains in any partition of the set into chains.

Dilworth's theorem characterizes the width of any finite partially ordered set in terms of a partition of the order into a minimum number of chains.
An antichain in a partially ordered set is a set of elements no two of which are comparable to each other, and a chain is a set of elements every two of which are comparable. Dilworth's theorem states that there exists an antichain A, and a partition of the order into a family P of chains, such that the number of chains in the partition equals the cardinality of A. When this occurs, A must be the largest antichain in the order, for any antichain can have at most one element from each member of P. Similarly, P must be the smallest family of chains into which the order can be partitioned, for any partition into chains must have at least one chain per element of A. The width of the partial order is defined as the common size of A and P.

Games

Grundy numbers

For a two-player, normal-play (last to move wins) game on a graph (V, E): $G(x) = mex(\{G(y) : (x, y) \in E\})$, where $mex(S) = min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

Sums of games

- Player chooses a game and makes a move in it. Grundy number of a position is xor of grundy numbers of positions in summed games.
- Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them. A position is losing iff each game is in a losing position.
- Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones. A position is losing iff grundy numbers of all games are equal.
- Player must move in all games, and loses if can't move in some game. A position is losing if any of the games is in a losing position.

Misère Nim

A position with pile sizes $a_1, a_2, ..., a_n \geq 1$, not all equal to 1, is losing iff $a_1 \bigoplus a_2 \bigoplus ... \bigoplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is odd.

Geometry

Pick's theorem

$I = A - B/2 + 1$, where A is the area of a lattice polygon, I is the number of lattice points inside it, and B is the number of lattice points on the boundary. Number of lattice points minus one on a line segment from (0, 0) and (x, y) is gcd(x, y).

$a.b = a_x b_x + a_y b_y = |a|.|b|.cos(\theta)$
$a \times b = a_x b_y - a_y b_x = |a|.|b|.sin(\theta)$
3D: $a \times b = (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)$

Line

Line $ax + by = c$ through $A(x_1, y_1)$ and $B(x_2, y_2)$: $a = y_1 - y_2$, $c = x_2 - x_1$, $c = ax_1 + by_1$.
Half-plane to the left of the directed segment AB: $ax + by \geq c$.
Normal vector: (a, b). Direction vector: $(b, -a)$. Perpendicular line: $-bx + ay = d$.
Point of intersection of $a_1x + b_1y = c_1$ and $a_2x + b_2y = c_2$ is $\frac{1}{a_1b_2 - a_2b_1}(c_1b_2 - c_2b_1, a_1c_2 - a_2c_1)$.
Distance from line $ax + by + c = 0$ to point (x_0, y_0) is $|ax_0 + by_0 + c|/\sqrt{a^2 + b^2}$.
Distance from line AB to P (for any dimension): $\frac{|(A-P)\times(B-P)|}{|A-B|}$.
Point-line segment distance:
if $(dot(B - A, P - A) < 0)$ return $dist(A, P)$;
if $(dot(A - B, P - B) < 0)$ return $dist(B, P)$;
return $fabs(cross(P, A, B)/dist(A, B))$;

Projection

Projection of point C onto line AB is $\frac{AB.AC}{AB.AB}AB$.
Projection of (x_0, y_0) onto line $ax + by = c$ is $(x_0, y_0) + \frac{1}{a^2+b^2}(ad, bd)$, where $d = c - ax_0 - by_0$.
Projection of the origin is $\frac{1}{a^2+b^2}(ac, bc)$.

Segment-segment intersection

Two line segments intersect if one of them contains an endpoint of the other segment, or each segment straddles the line, containing the other segment (AB straddles line l if A and B are on the opposite sides of l.)

Circle-circle and circle-line intersection

```
a = x2 - x1;   b = y2 - y1;   c = [(r1^2 - x1^2 - y1^2) - (r2^2 - x2^2 - y2^2)] / 2;
d = sqrt(a^2 + b^2);
if not |r1 - r2| <= d <= |r1 + r2|, return "no solution"
if d == 0, circles are concentric, a special case
// Now intersecting circle (x1,y1,r1) with line ax+by=c
Normalize line: a /= d; b /= d; c /= d;     // d=sqrt(a^2+b^2)
e = c - a*x1 - b*y1;
h = sqrt(r1^2 - e^2);                       // check if r1<e for circle-line test
return (x1, y1) + (a*e, b*e) +/- h*(-b, a);
```

Circle from 3 points (circumcircle)

Intersect two perpendicular bisectors. Line perpendicular to $ax + by = c$ has the form $-bx + ay = d$. Find d by substituting midpoint's coordinates.

Angular bisector

Angular bisector of angle ABC is line BD, where $D = \frac{BA}{|BA|} + \frac{BC}{|BC|}$.
Center of incircle of triangle ABC is at the intersection of angular bisectors, and is $\frac{a}{a+b+c}A + \frac{b}{a+b+c}B + \frac{c}{a+b+c}C$ where a, b, c are lengths of sides, opposite to vertices A, B, C. Radius $= \frac{2S}{a+b+c}$

Counter-clockwise rotation around the origin

$(x, y) \to (x\cos\phi - y\sin\phi, x\sin\phi + y\cos\phi)$.
90-degrees counter-clockwise rotation: $(x, y) \to (-y, x)$. Clockwise: $(x, y) \to (y, -x)$.

3D rotation

3D rotation by ccw angle ϕ around axis n: $r' = r\cos\phi + n(n \cdot r)(1 - \cos\phi) + (n \times r)\sin\phi$

Plane equation from 3 points

$N \cdot (x, y, z) = N \cdot A$, where N is normal: $N = (B - A) \times (C - A)$.

3D figures

Sphere: Volume $V = \frac{4}{3}\pi r^3$, surface area $S = 4\pi r^2$
$x = \rho\sin\theta\cos\phi$, $y = \rho\sin\theta\sin\phi$, $z = \rho\cos\theta$, $\phi \in [-\pi, \pi], \theta \in [0, \pi]$
Spherical section: Volume $V = \pi h^2(r - h/3)$, surface area $S = 2\pi rh$
Pyramid: Volume $V = \frac{1}{3}hS_{base}$
Cone: Volume $V = \frac{1}{3}\pi r^2 h$, lateral surface area $S = \pi r\sqrt{r^2 + h^2}$

Area of a simple polygon

$\frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1}y_i)$, where $x_n = x_0$, $y_n = y_0$. Area is negative if the boundary is oriented clockwise.

Winding number

Shoot a ray from given point in an arbitrary direction. For each intersection of ray with polygon's side, add +1 if the side crosses it counterclockwise, and -1 if clockwise.