Team Notebook

Hakuna Matata

December 30, 2020

Contents

1 3dpsum

```
num a[23][23][23];
num psum[23][23][23];
int A, B, C;

num get(int i, int j, int k){
    if(i >= 0 && i < A && j >= 0 && j < B && k >= 0 && k < C)
        return psum[i][j][k];
    return 0;
}
// Inclusion-exclusion principle everywhere!
void build(){
    for(int i = 0; i < A; i++)
        for(int j = 0; j < B; j++)
            for(int k = 0; k < C; k++)
                psum[i][j][k] = a[i][j][k] + get(i-1, j, k) +
                     get(i, j-1, k) + get(i, j, k-1) - get(i
                     -1, j-1, k) - get(i-1, j, k-1) - get(i, j
                     -1, k-1) + get(i-1, j-1, k-1);
}

num cube(int i1, int j1, int k1, int i2, int j2, int k2){
    return get(i2, j2, k2) - get(i1-1, j2, k2) - get(i2, j1
         -1, k2) - get(i2, j2, k1-1) + get(i1-1, j1-1, k2) +
         get(i1-1, j2, k1-1) + get(i2, j1-1, k1-1) - get(i1
         -1, j1-1, k1-1);
}
```

2 bellmanford

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

// SSSP, -ve weight

// Bellman Ford's Algorithm

// O(V ^ 3)

#define inf INT_MAX // INT_MAX is important

int n, src;
struct edge{
    int u, v, w; // edge from u to v with weight w
} edges[103];
long long int dis[103]; // long long int is important

int main(){
    // read the graph
    fill_n(dis, n, inf);
    dis[src] = 0;
    for(int i = 0; i < n-1; i++)
        for(edge e : edges)
            if(dis[e.u] != inf)
                dis[e.v] = min(dis[e.v], dis[e.u] + e.w);
    for(int i = 0; i < n-1; i++)
        for(edge e : edges)
            if(dis[e.u] != inf && dis[e.u] + e.w < dis[e.v])
                dis[e.v] = -inf;
    // nodes with dis = inf are not reachable
    // nodes with dis = -inf are reachable from at least one
         negative cycle
}
```

3 bfs

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

// SSSP, BFS

// O(V + E)

#define inf INT_MAX

int n, src;
vector<int> adj[103];
int dis[103];
queue<int> bfs;

int main(){
    // read the graph
    fill_n(dis, n, inf);
    dis[src] = 0;
    bfs.push(src);
    while(!bfs.empty()){
        int u = bfs.front();
        bfs.pop();
        for(int v : adj[u])
            if(dis[v] == inf)
                dis[v] = dis[u] + 1, bfs.push(v);
    }
}
```

4 bicoloring

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

// Graph Bi-Coloring (Check)

// O(V + E)

const int maxn = 500003;

int n, m;
vector<int> adjs[maxn];
int color[maxn]; // 0 is red, 1 is blue and 2 is without
     color
bool bicolorable = true;

void dfs(int u, int c){
    color[u] = c;
    for(int x : adjs[u]){
        if(color[x] == 2)
            dfs(x, 1 - c); // swap the color!
        else if(color[x] == color[u])
            bicolorable = false;
    }
}

int main(){
    // read the graph
    fill_n(color, n, 2);
    for(int i = 0; i < n; i++)
        if(color[i] == 2)
            dfs(i, 1); // or dfs(i, 0)
}
```

5 bridge

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

// Cut Vertices/Bridges

// O(V + E)

const int maxn = 103;

int v, e;
```

```
vector<int> adjs[maxn];
int vis[maxn]; // 0 is for UNVISITED, 1 is for EXPLORED and
    2 is for VISITED
int dfs_low[maxn], dfs_num[maxn];
int par[maxn];
int cnt = 0;

set<int> articulation_points;
set<pair<int, int>> bridges;

void dfs(int u){
    vis[u] = 1;
    dfs_low[u] = dfs_num[u] = cnt++;
    for(int v : adjs[u]){
        if(!vis[v]){
            par[v] = u;
            dfs(v);
            if(dfs_low[v] >= dfs_num[u])
                articulation_points.insert(u);
            if(dfs_low[v] > dfs_num[u])
                bridges.insert({u, v});
   dfs_low[u] = min(dfs_low[u], dfs_low[v]);
        }
        else if(v != par[u])
            dfs_low[u] = min(dfs_low[u], dfs_num[v]);
    }
    vis[u] = 2;
}

int main(){
    // read the graph
    dfs(0);
    if(adjs[0].size() == 1) // root is not ap if it has at
        most one child
        articulation_points.erase(0);
    // print aps or bridges
}
```

6 bronkerbosch

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

// Finding All Maximal Cliques(Max Independent Sets)

// Bron Kerbosch's Algorithm

// O(3 ^ (V / 3))

const int maxn = 133;

int n, m, deg[maxn];
set<int> adj[maxn];
set<int> R, P, X;

void bron(){
    if(P.empty() && X.empty()){ // A Maximal Cilque found and
        is in Set R
        cout << "Clique found: ";
        for(int u : R)
            cout << u << " ";
        cout << endl;
        return;
    }
 if(P.empty())
  return;
    int u, degm = -1;
    for(int cand : P)
        if(deg[cand] > degm)
            u = cand, degm = deg[cand];
    for(int cand : X)
        if(deg[cand] > degm)
            u = cand, degm = deg[cand];
    set<int> PP = P;
    for(int nu : adj[u])
        PP.erase(nu);
    for(int v : PP){
        set<int> cR = R, cP = P, cX = X;
        R.insert(v);
        P.clear();
        X.clear();
        for(int nv : adj[v]){
            if(cP.find(nv) != cP.end())
                P.insert(nv);
            if(cX.find(nv) != cX.end())
                X.insert(nv);
        }
        bron();
        R = cR, P = cP, X = cX;
        P.erase(v);
        X.insert(v);
    }
}

int main(){
    // read the graph
    for(int i = 0; i < n; i++)
        deg[i] = adj[i].size(), P.insert(i);
    bron();
}
```

7 catalan

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

// Catalan Numbers using DP VS Formula

long long int dp[23], cat[23];

int main(){
    dp[0] = 1;
    for(int n = 1; n <= 20; n++)
        for(int k = 0; k < n; k++)
            dp[n] += dp[k] * dp[n - 1 - k];
    cat[0] = 1;
    for(int n = 1; n <= 20; n++)
        cat[n] = (cat[n - 1] * (4 * n - 2)) / (n + 1);
    for(int n = 0; n <= 20; n++)
        cout << n << " " << dp[n] << " " << cat[n] << endl;
}
```

8 circle

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

struct circle{
    point center;
    double radius;

    circle() = default;

    circle(point center, double radius){
        this->center = center;
        this->radius = radius;
    }

    circle& operator=(const circle& o) = default;

    int status(point& p){
        double dx = p.x - center.x, dy = p.y - center.y;
```

```
        double euc = dx * dx + dy * dy;
        if(fabs(euc - radius * radius) < EPS)
            return 0; // on circle
        if(euc < radius * radius)
            return -1; // inside circle
        return 1; // outside circle
    }

    double perimeter(){
        return 2 * M_PI * this->radius;
    }

    double area(){
        return M_PI * this->radius * this->radius;
    }

    double arc(double rad){
        return rad * this->radius;
    }

    double chord(double rad){
        return 2 * this->radius * sin(rad / 2);
    }

    double sector(double rad){
        return (rad / (2 * M_PI)) * this->area();
    }
};

// before using make sure that p1.dist(p2) <= 2 * r
pair<circle, circle> circles_with_2_points(point& p1, point&
      p2, double r){
    double l = p1.dist(p2);
    double a = sqrt(r * r - l * l / 4);
    return {circle(point((p1.x + p2.x) / 2 + (p1.y - p2.y) *
         a / l, (p1.y + p2.y) / 2 + (p2.x - p1.x) * a / l), r
        ),
           circle(point((p1.x + p2.x) / 2 + (p2.y - p1.y) *
                a / l, (p1.y + p2.y) / 2 + (p1.x - p2.x) * a
                / l), r)};
}
```

9 closestpair

```
inline double cdis(pair<double, double>& p1, pair<double,
     double>& p2){
    double deltax = p1.first - p2.first;
    double deltay = p1.second - p2.second;
    return sqrt(deltax * deltax + deltay * deltay);
}

bool byX(const pair<double, double>& p1, const pair<double,
     double>& p2){
    return p1.first < p2.first;
}

bool byY(const pair<double, double>& p1, const pair<double,
     double>& p2){
    return p1.second < p2.second;
}

int n;
pair<double, double> points[100003];
pair<double, double> tmp[100003];

struct closest{
    double dis;
    pair<double, double> p1, p2;
};

closest find_closest(int l, int r){
    closest ans;
    if(r-l == 1){
        ans.dis = cdis(points[l], points[r]);
        ans.p1 = points[l];
        ans.p2 = points[r];
    }
    else if(r-l == 2){
        ans.dis = cdis(points[r-1], points[r]);
        ans.p1 = points[r-1];
        ans.p2 = points[r];
        for(int k = r-1; k <= r; k++){
            double dis = cdis(points[l], points[k]);
            if(dis < ans.dis){
                ans.dis = dis;
                ans.p1 = points[l];
                ans.p2 = points[k];
            }
        }
    }
    else{
        int mid = (l + r) / 2;
        closest left = find_closest(l, mid);
        closest right = find_closest(mid+1, r);
        if(left.dis < right.dis){
            ans.dis = left.dis;
            ans.p1 = left.p1;
            ans.p2 = left.p2;
        }else{
            ans.dis = right.dis;
            ans.p1 = right.p1;
            ans.p2 = right.p2;
        }
        auto lptr = lower_bound(points+l, points+r+1,
             make_pair(points[mid].first - ans.dis, 0.0), byX
             );
        auto rptr = upper_bound(points+l, points+r+1,
             make_pair(points[mid+1].first + ans.dis, 0.0),
             byX);
        int size = rptr - lptr;
        copy(lptr, rptr, tmp);
        sort(tmp, tmp+size, byY);
        for(int i = 0; i < size; i++)
            for(int j = i+1; j < min(i+16, size); j++){
                double dis = cdis(tmp[i], tmp[j]);
                if(dis < ans.dis){
                    ans.dis = dis;
                    ans.p1 = tmp[i];
                    ans.p2 = tmp[j];
                }
            }
    }
    return ans;
}

closest solve(){
    sort(points, points+n);
    return find_closest(0, n-1);
}
```

10 dfs

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

// Finding Connected Components

// O(V + E)

int n, cc = 0;
vector<int> adj[103];
bool vis[103];
int id[103], ccsize[103];

void dfs(int u){
    vis[u] = true;
```

```
    id[u] = cc;
    ccsize[cc]++;
    for(int v : adj[u])
        if(!vis[v])
            dfs(v);
}

int main(){
    // read the graph
    fill_n(vis, n, false);
    for(int u = 0; u < n; u++)
        if(!vis[u])
            dfs(u), cc++;
}
```

11 diameter

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

// Tree

// Diameter = Longest Path of a Tree

// O(V)

#define MAX_N 100003

int n;
vector<int> adj[MAX_N];

int diameter, ind; // ind is start of longest path (farest
     node from root)

void dfs(int u, int p, int h){
    if(diameter < h)
        diameter = h, ind = u;
    for(auto v : adj[u])
        if(v != p)
            dfs(v, u, h + 1);
}

int par[MAX_N], ind2; // ind2 is end of longest path (farest
      node from ind)

void dfs2(int u, int h){
    if(diameter < h)
        diameter = h, ind2 = u;
    for(auto v : adj[u])
        if(v != par[u])
            par[v] = u, dfs2(v, h + 1);
}

vector<int> path;

int main(){
    // read the graph
    diameter = -1;
    dfs(0, -1, 0);
    diameter = -1;
    dfs2(ind, 0);
    cout << diameter << endl;
    int it = ind2;
    while(it != -1)
        path.push_back(it), it = par[it];
    for(int i = path.size() - 1; i >= 0; i--)
        cout << path[i] << " ";
    cout << endl;
}
```

12 dijkstra

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

// SSSP, Dijkstra

// Dijkstra's Algorithm

// O(V + E * logV)

#define inf INT_MAX

int n, src;
vector<pair<int, int>> adjs[103]; // {weight, dst}
priority_queue<pair<int, int>, vector<pair<int, int>>,
     greater<pair<int, int>>> pq;
int dis[103];

int main(){
    // read the graph
    fill_n(dis, n, inf);
    pq.push({0, src});
    dis[src] = 0;
    while(!pq.empty()){
        auto p = pq.top();
        pq.pop();
        int d = p.first, u = p.second;
        if(dis[u] < d) // lazy delete
            continue;
        for(auto x : adjs[u])
            if(dis[x.second] > dis[u] + x.first) //
                 relaxation
                dis[x.second] = dis[u] + x.first, pq.push({dis
                     [x.second], x.second});
    }
}
```

13 dinic

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

// Dinic's Network Flow

// O(V ^ 2 * E)

// O(sqrt(V) * E) for Bipartite Matching

const int maxn = 503;
const int inf = INT_MAX;

struct edge{
    int u, v, cap, flow = 0;
    edge(int u, int v, int cap){
        this->u = u, this->v = v, this->cap = cap;
    }
};

int n, m = 0, s, t, ptr[maxn], level[maxn];
vector<edge> edges;
vector<int> adjs[maxn];
queue<int> q;

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

bool bfs(){
    q.push(s);
```

```
    fill_n(level, n, -1);
    level[s] = 0;
    while(!q.empty()){
        int u = q.front();
        q.pop();
        for(int eid : adjs[u]){
            int v = edges[eid].v;
            if(edges[eid].cap - edges[eid].flow == 0)
                continue;
            if(level[v] == -1)
                q.push(v), level[v] = level[u] + 1;
        }
    }
    return level[t] != -1;
}

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

int max_flow(){
    int ans = 0;
    while(bfs()){
        fill_n(ptr, n, 0);
        while(1){
            int p = dfs(s, inf);
            if(!p)
                break;
            ans += p;
        }
    }
    return ans;
}

int main(){
    // read the graph using add_edge function
    // Note: do not change variable m!
    // set s(source) and t(sink)
    cout << max_flow() << endl;
}
```

14 euler

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

// Eulerian Graph

// Euler's Algorithm

// O(V + E)

// This code is for a directed graph and prints either
    eulerian tour or path

const int maxn = 1003;

int n;
vector<pair<string, int>> adjs[maxn]; // edges are labeled
    with string
int indeg[maxn], outdeg[maxn];

vector<int> eulerian_nodes;
vector<string> eulerian_edges;

void euler(int u){
    while(!adjs[u].empty()){
        auto x = *adjs[u].rbegin();
        adjs[u].pop_back();
        euler(x.second);
        eulerian_edges.push_back(x.first);
    }
    eulerian_nodes.push_back(u);
}

int check_eulerian(){ // -1 is for a non-eulerian graph,
    else start node is returned
    int oddcnt = 0;
    vector<int> oddv;
    for(int i = 0; i < n; i++)
        if(indeg[i] != outdeg[i])
            oddcnt++, oddv.push_back(i);
    if(oddcnt == 1 || oddcnt > 2)
        return -1;
    if(oddcnt == 0){
        for(int i = 0; i < n; i++)
            if(adjs[i].size() > 0)
                return i;
    }
    // oddcnt is 2:
    int u1 = oddv[0], u2 = oddv[1];
    if((indeg[u1] - outdeg[u1]) * (indeg[u2] - outdeg[u2]) !=
        -1)
        return -1;
    if(outdeg[u1] > indeg[u1])
        return u1;
    return u2;
}

int main(){
    // read the graph
    int s = check_eulerian();
    if(s == -1){
        cout << "impossible" << endl;
        return 0;
    }
    euler(s);
    for(int i = 0; i < n; i++)
        if(!adjs[i].empty()){
            cout << "impossible" << endl;
            return 0;
        }
    reverse(eulerian_nodes.begin(), eulerian_nodes.end());
    reverse(eulerian_edges.begin(), eulerian_edges.end());
    // print the path
}
```

15 eval

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

// encode and decode for unary - operator

string encode(string str){
    string en = "";
    en += (str[0] == '-') ? '#' : str[0];
    for(int i = 1; i < str.size(); i++){
        if(str[i] == '-' && (str[i+1] >= '0' && str[i+1] <= '
            9') && !(str[i-1] >= '0' && str[i-1] <= '9'))
            en += '#';
```

```
        else
            en += str[i];
    }
    return en;
}

string decode(string str){
    string de = "";
    for(char c : str){
        if(c == '#')
            de += '-';
        else de += c;
    }
    return de;
}

vector<string> tokenize(string str, char delim){
    stringstream ss(delim == '-' ? encode(str) : str);
    string token;
    vector<string> ans;
    while(getline(ss, token, delim))
        ans.push_back(delim == '-' ? decode(token) : token);
    return ans;
}

bool div_by_zero;

int eval1(string str){ // just with * and /
    int ans = 1;
    vector<string> t1 = tokenize(str, '*');
    for(string s : t1){
        vector<string> t2 = tokenize(s, '/');
        ans *= stoi(t2[0]);
        for(int i = 1; i < t2.size(); i++){
            int tmp = stoi(t2[i]);
            if(tmp)
                ans /= tmp;
            else div_by_zero = true;
        }
    }
    return ans;
}

int eval2(string str){ // with + - * /
    int ans = 0;
    vector<string> t1 = tokenize(str, '+');
    for(string s : t1){
        vector<string> t2 = tokenize(s, '-');
        if(t2[0] != "")
            ans += eval1(t2[0]);
        for(int i = 1; i < t2.size(); i++)
            ans -= eval2(t2[i]);
    }
    return ans;
}

int eval(string str){ // with () + - * /
    string ans = "";
    int i = 0;
    while(i < str.size()){
        if(str[i] == '('){
            int cnt = 1, j = i + 1;
            while(cnt){
                if(str[j] == '(')
                    cnt++;
                else if(str[j] == ')')
                    cnt--;
                j++;
            }
            ans += to_string(eval(str.substr(i+1, j-i-1)));
            i = j;
        }
        else
            ans += str[i], i++;
    }
    return eval2(ans);
}

int main(){
    string str;
    cin >> str;
    cout << eval(str) << endl;
}
```

16 explore

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

// Graph Properties Check

// DFS Spanning Tree

// O(V + E)

const int maxn = 50003;

int n;
vector<int> adj[maxn];
int vis[maxn]; // 0 is for UNVISITED, 1 is for EXPLORED and
    2 is for VISITED
int par[maxn];
bool cycle = false;

set<pair<int, int>> tree_edge, back_edge, cross_edge;

void dfs(int u){
    vis[u] = 1;
    for(int v : adj[u]){
        if(!vis[v]){
            tree_edge.insert({u, v});
            par[v] = u, dfs(v);
        }
        else if(par[u] != v){ // Not two way edge!
            if(vis[v] == 1)
                cycle = true, back_edge.insert({u, v});
            else // vis[v] == 2
                cross_edge.insert({u, v});
        }
    }
    vis[u] = 2;
}

int main(){
    // read the graph
    for(int i = 0; i < n; i++)
        if(!vis[i])
            dfs(i);
    // print edges
}
```

17 ExtendedEuclid+modInverse+modI

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

// Extended Euclid: Solving Linear Diophantine Equation
// ax + by = c has no integral solutions if gcd(a, b) | c is
     not true
// calculate x0, y0 by this algorithm
// then: x = x0 + (b/gcd(a, b))n, y = y0  (a/gcd(a, b))n
long long int x, y, d; // d = gcd(a, b)
void extendedEuclid(long long int a, long long int b) {
    if (b == 0){ // base case
        x = 1;
        y = 0;
```

```
        d = a;
        return;
    }
    extendedEuclid(b, a % b); // similar as the original gcd
    long long int x1 = y;
    long long int y1 = x - (a / b) * y;
    x = x1;
    y = y1;
}


// work to find x in: a.x  1 mod m
int modularInverse(long long int a, long long int m){
    extendedEuclid(a, m);
    if (d != 1) {
        return -1; // No Solution.
    }
    else {
        return (x % m + m) % m;
    }
}

int modDivide(long long int a, long long int b, long long
     int m)
{
    a = a % m;
    long long int inv = modularInverse(b, m);
    if (inv == -1)
       return -1; // No Solution!
    else
       return (inv * a) % m;
}


int main()
{
    int a= 23, b = 1337;
    cout << modularInverse(a, b) << endl; //ans = 872
    cout << modDivide(8, 4, 5) << endl; // ans = 2
    return 0;
}
```

18 fastpower

```
#define mod 1000000007

long long int fast_power(long long int a, long long int b){
    if(b == 0)
        return 1;
    if(b % 2 == 1)
        return (a * fast_power(a, b-1)) % mod;
    long long int half = fast_power(a, b / 2);
    return (half * half) % mod;
}
```

19 fenwicktree

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

class FenwickTree { // Notice: it is 1-based fenwick-tree.
    private:
    vector<long long int> ft;
    public:
    FenwickTree(int n){
        ft.assign(n + 1, 0); // init n + 1 zeroes
    }
    long long int rsq(long long int b) { // returns RSQ(1, b)
        long long int summ = 0;
        for (; b; b -= (b & (-b)))
            summ += ft[b];
        return summ;
    }
    long long int rsq(int a, int b) { // returns RSQ(a, b)
        return rsq(b) - (a == 1 ? 0 : rsq(a - 1));
    }
    // adjusts value of the k-th element by v (v can be +ve/
         inc or -ve/dec)
    void adjust(long long int k, long long int v) { // note:
         n = ft.size() - 1
        for (; k < ft.size(); k += (k & (-k)))
            ft[k] += v;
    }
};


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

    //  get input with fast io functions.
    //  it is better to save the result and output them at
         the end.

    return 0;
}
```

20 floyd

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

// APSP

// Floyd Warshall's Algorithm

// O(V ^ 3)

// Minimax path (max edge weight should be minimum): sp[i][j
     ] = min(sp[i][j], max(sp[i][k], sp[k][j]))
// Shortest cycle: Initially set sp[i][i] = inf then the
     answer is min(sp[i][i]) for all i
// Diameter (max sp): max(sp[i][j]) for all i and j
// SCC: if sp[i][j] != inf and sp[j][i] != inf then i and j
     are in the same scc

#define inf INT_MAX
#define num long long int

const int maxn = 153;

int n, e, q;
num sp[maxn][maxn];
int par[maxn][maxn];

void path(int i, int j){
    int k = par[i][j];
    if(k == -1)
        cout << i << " ";
    else
        path(i, k), path(k, j);
}

int main(){
    cin >> n >> e;
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++){
            if(i == j)
                sp[i][j] = 0;
            else
                sp[i][j] = inf;
            par[i][j] = -1;
        }
    for(int i = 0; i < e; i++){
        int u, v, w;
        cin >> u >> v >> w; // edges are directed
        sp[u][v] = min(sp[u][v], (num)w);
```

```
    }
    for(int k = 0; k < n; k++)
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                if(sp[i][k] != inf && sp[k][j] != inf && sp[i
                    ][k] + sp[k][j] < sp[i][j])
                    sp[i][j] = sp[i][k] + sp[k][j], par[i][j]
                        = k;
    // check for paths with negative cycle:
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            for(int k = 0; k < n; k++)
                if(sp[i][k] != inf && sp[k][j] != inf && sp[k
                    ][k] < 0)
                    sp[i][j] = -inf;
    int src, dst;
    cin >> src >> dst;
    cout << sp[src][dst] << endl;
    // print path:
    path(src, dst);
    cout << dst << endl;
}
```

21 fraction

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

#define num long long int

const num inf = LLONG_MAX;

num GCD(num a, num b) {
    if (a == 0)
        return b;
    return GCD(b % a, a);
}

struct fraction {

        num numerator, denominator;
        double real;

        fraction(){
            this->numerator = 0;
            this->denominator = 1;
            this->real = 0.0;
        }

        fraction(num numerator, num denominator){
            if(denominator == 0){
                this->numerator = inf;
                this->denominator = 1;
            }
            else if(numerator == 0){
                this->numerator = 0;
                this->denominator = 1;
            }
            else{
                num sign = (numerator > 0 ? 1 : -1) * (
                    denominator > 0 ? 1 : -1);
                numerator = abs(numerator), denominator = abs(
                    denominator);
                num gcd = GCD(min(numerator, denominator), max
                    (numerator, denominator));
                this->numerator = sign * numerator / gcd;
                this->denominator = denominator / gcd;
            }
            this->real = double(this->numerator) / this->
                denominator;
        }

        fraction& operator=(const fraction &other) = default;

        fraction(double real) {
            this->real = real;
            char str[50];
            sprintf(str, "%f", real);
            int str_len = strlen(str);
            double exp = pow(10, str_len - (find(str, str +
                str_len, '.') - str) - 1);
            numerator = real * exp;
            denominator = exp;
            num gcd = GCD(min(numerator, denominator), max(
                numerator, denominator));
            numerator = numerator / gcd;
            denominator = denominator / gcd;
        }

        bool operator<(const fraction &other) const {
            if(this->numerator == inf)
                return false;
            if(other.numerator == inf)
                return true;
            return this->numerator * other.denominator <
                other.numerator * this->denominator;
        }

        bool operator==(const fraction &other) const {
            return this->numerator == other.numerator && this
                ->denominator == other.denominator;
        }

        operator const char *() const {
            char *buffer = (char *) malloc(20);
            sprintf(buffer, "%lld/%lld", this->numerator,
                this->denominator);
            return buffer;
        }

        fraction negate(){
            if(this->numerator == inf)
                return fraction(1, 0);
            if(this->numerator == 0)
                return fraction(0, 1);
            return fraction(-this->numerator, this->
                denominator);
        }

        fraction reverse(){
            if(this->numerator == inf)
                return fraction(0, 1);
            if(this->numerator == 0)
                return fraction(1, 0);
            return fraction(this->denominator, this->
                numerator);
        }

        fraction operator+(const fraction &other) const {
            return fraction(this->numerator * other.
                denominator + other.numerator * this->
                denominator, this->denominator * other.
                denominator);
        }

        fraction operator-(const fraction &other) const {
            return fraction(this->numerator * other.
                denominator - other.numerator * this->
                denominator, this->denominator * other.
                denominator);
        }

        fraction operator*(const fraction &other) const {
            return fraction(this->numerator * other.numerator
                , this->denominator * other.denominator);
        }

        fraction operator/(const fraction &other) const {
```

```
        fraction p(this->numerator, this->denominator);
        fraction q(other.numerator, other.denominator);
        return p * q.reverse();
    }
};

int main(){
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    fraction p = fraction(a, b), q = fraction(c, d);
    cout << p + q << endl;
    cout << p * q << endl;
    cout << p - q << endl;
    cout << p / q << endl;
}
```

22 gaussianelimination

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

#define MAX_N 103 // adjust this value as needed

struct AugmentedMatrix{
    double mat[MAX_N][MAX_N + 1];
};

struct ColumnVector{
    double vec[MAX_N];
};

ColumnVector GaussianElimination(int N, AugmentedMatrix Aug)
     { // O(N^3)
    // input: N, Augmented Matrix Aug, output: Column vector
         X, the answer
    int i, j, k, l;
    double t;
    ColumnVector X;
    for(j = 0; j < N - 1; j++){ // the forward elimination
         phase
        l = j;
        for(i = j + 1; i < N; i++) // which row has largest
             column value
            if(fabs(Aug.mat[i][j]) > fabs(Aug.mat[l][j]))
                l = i; // remember this row l
        // swap this pivot row, reason: to minimize floating
             point error
        for(k = j; k <= N; k++) // t is a temporary double
             variable
            t = Aug.mat[j][k], Aug.mat[j][k] = Aug.mat[l][k],
                 Aug.mat[l][k] = t;
        for(i = j + 1; i < N; i++) // the actual forward
             elimination phase
            for(k = N; k >= j; k--)
                Aug.mat[i][k] -= Aug.mat[j][k] * Aug.mat[i][j]
                     / Aug.mat[j][j];
    }
    for(j = N - 1; j >= 0; j--){ // the back substitution
         phase
        for(t = 0.0, k = j + 1; k < N; k++)
            t += Aug.mat[j][k] * X.vec[k];
        X.vec[j] = (Aug.mat[j][N] - t) / Aug.mat[j][j]; //
             the answer is here
    }
    return X;
}
```

23 gcd

```
int gcd(int a, int b){
    int mx = max(abs(a), abs(b));
    int mi = min(abs(a), abs(b));
    if(mi == 0)
        return mx;
    return gcd(mi, mx % mi);
}
```

24 hamilton

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

// Hamiltonian Path in DAG

// Hamiltonian Path exists if and only if DAG has one unique
     Topological Order

// O(V + E)

int n;
int indeg[1003];
vector<int> adj[1003];
vector<int> path;
bool hamiltonian(){
 queue<int> candidate;
    for(int i = 0; i < n; i++)
        if(!indeg[i])
            candidate.push(i);
    while(!candidate.empty()){
        if(candidate.size() > 1)
            return false;
        int u = candidate.front();
        candidate.pop();
        path.push_back(u);
        for(int v : adj[u]){
            indeg[v]--;
            if(!indeg[v])
                candidate.push(v);
        }
    }
 return true;
}
```

25 inversionindex

```
int n;
int arr[1000003];
int tmp[1000003];

long long int swaps(int start, int end){
    if(start == end)
        return 0;
    int mid = (start + end) / 2;
    long long int cnt = swaps(start, mid) + swaps(mid+1, end)
         ;
    int lptr = start, rptr = mid+1;
    int output = start;
    while(lptr <= mid || rptr <= end){
        if(lptr == mid+1)
            tmp[output++] = arr[rptr++];
        else if(rptr == end+1)
            tmp[output++] = arr[lptr++];
        else{
            if(arr[lptr] > arr[rptr]){
                tmp[output++] = arr[rptr++];
                cnt += mid - lptr + 1;
            }
            else tmp[output++] = arr[lptr++];
        }
```

```
    }
    copy(tmp+start, tmp+output, arr+start);
    return cnt;
}
```

26 kahn

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

// Topological Sort

// Kahn's Algorithm

// O(V + E)

#define MAX_N 100003

int n, m, a, b;
vector<int> adj[MAX_N];
int indeg[MAX_N];
priority_queue<int, vector<int>, greater<int>> pq;
vector<int> topoList;

int main(){
    // read the graph
    for(int i = 0; i < n; i++)
        if(!indeg[i])
            pq.push(i);
    while(!pq.empty()){
        int cur = pq.top();
        pq.pop();
        topoList.push_back(cur);
        for(auto v: adj[cur]){
            indeg[v]--;
            if(!indeg[v])
                pq.push(v);
        }
    }
    if(topoList.size() != n)
        cout << "cycle" << endl;
    else for(int i = 0; i < topoList.size(); i++)
            cout << topoList[i] << " ";
}
```

27 kmp

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

// KMP String Matching

// Find All Occurrences of Pattern P in Text T

// O(n + m)

const int maxn = 1e8 + 5;

int n, m; // n = size of t, m = size of p
string t, p; // t is text, p is pattern
int lcp[maxn + 3];

int main(){
 // read input
 n = t.size(), m = p.size();
 int i = 0, j = 1;
 lcp[0] = 0;
 while(j < m){
  if(p[i] == p[j])
   lcp[j] = i + 1, i++, j++;
  else if(i)
   i = lcp[i - 1];
  else
   lcp[j] = 0, j++;
 }
 i = 0, j = 0;
 while(i < n){
  if(t[i] == p[j]){
   i++, j++;
   if(j == m){
    cout << i - j << endl; // T[i-m, i) == P
    j = lcp[j - 1];
   }
  }
  else if(j)
   j = lcp[j - 1];
  else
   i++;
 }
}
```

28 kruskal

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

// MST

// Kruskal's Algorithm

// O(E * logV)

// Second Best Spanning Tree: Remove each edge in MST and re
    -run kruskal, get min
// Minimax path: path that minimizes max-edge-weight: path
    in ms tree

#define MAX_N 1003

int n, m;
int a, b, w;
vector<pair<int, pair<int, int>>> edgeList;
int parent[MAX_N];
int raank[MAX_N];

void initSets(int n){
    for(int i = 0; i < n; i++){
        parent[i] = i;
        raank[i] = 0;
    }
}

int findSet(int i){
    if(parent[i] == i)
        return i;
    return parent[i] = findSet(parent[i]); // Path
        Compression
}

inline bool inSameSets(int i, int j){
    return findSet(i) == findSet(j);
}

void mergeSets(int i, int j){
    int a = findSet(i);
    int b = findSet(j);
    if(a == b)
        return;
    if(raank[a] >= raank[b])
        parent[b] = a;
    else
        parent[a] = b;
    if(raank[a] == raank[b])
```

```
        raank[a]++;
}

int main(){
    // read the graph
    sort(edgeList.begin(), edgeList.end());
    int mst = 0;
    int forests = n;
    initSets(n);
    for (int i = 0; i < edgeList.size(); i++){
        if(forests == 1)
            break;
        pair<int, pair<int, int>> front = edgeList[i];
        if(!inSameSets(front.second.first, front.second.
             second)){
            mst += front.first;
            forests--;
            mergeSets(front.second.first, front.second.second
                 );
        }
    }
    cout << mst << endl;
}
```

29 lazysegmenttree

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

// Lazy Segment Tree Implementation

// Example: Query = Obtaining minimum of a range
// Example: Update = Adding a constant value to a range

const int maxn = 1e5;
const int inf = INT_MAX;

int n, arr[maxn + 3];
int seg[4 * maxn + 3], lazy[4 * maxn + 3];

// O(n * log(n))
void build(int v, int tl , int tr){
    if(tl == tr)
        seg[v] = arr[tl];
    else{
        int mid = (tl + tr) / 2;
        build(2 * v, tl, mid);
        build(2 * v + 1, mid + 1, tr);
        seg[v] = min(seg[2 * v], seg[2 * v + 1]);
    }
}

// O(1)
void push(int v){
    seg[2 * v] += lazy[v];
    seg[2 * v + 1] += lazy[v];
    lazy[2 * v] += lazy[v];
    lazy[2 * v + 1] += lazy[v];
    lazy[v] = 0;
}

// O(log(n))
void update(int v, int tl, int tr, int l , int r, int val){
    if(l > r)
        return;
    if(l == tl && tr == r){
        seg[v] += val;
        lazy[v] += val;
        return;
    }
    push(v);
    int mid = (tr + tl) / 2;
    update(2 * v, tl, mid, l, min(r, mid), val);
    update(2 * v + 1, mid + 1, tr, max(l, mid + 1), r, val);
    seg[v] = min(seg[2 * v], seg[2 * v + 1]);
}

// O(log(n))
int query(int v, int tl, int tr, int l , int r){
    if(l > r)
        return inf;
    if(l == tl && tr == r)
        return seg[v];
    push(v);
    int mid = (tr + tl) / 2;
    return min(query(2 * v, tl, mid, l, min(r, mid)), query(2
         * v + 1, mid + 1, tr, max(l, mid + 1), r));
}

int main(){
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    build(1, 0, n - 1);
    // handle queries and updates, initially put v = 1, tl =
         0, tr = n - 1
}
```

30 lca

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

// Lowest Common Ancestor

// O(log(n)) per query

const int maxn = 1e5 + 3;
int n, height[maxn], first[maxn], t[8 * maxn];
vector<int> euler;
vector<int> adjs[maxn];
bool vis[maxn];

void build(int v, int tl, int tr){
    if(tl == tr)
        t[v] = euler[tl];
    else{
        int tm = (tl + tr) / 2;
        build(2 * v, tl, tm);
        build(2 * v + 1, tm + 1, tr);
        int l = t[2 * v], r = t[2 * v + 1];
        t[v] = (height[l] <= height[r]) ? l : r;
    }
}

void dfs(int u, int h){
    vis[u] = 1;
    first[u] = euler.size();
    height[u] = h;
    euler.push_back(u);
    for(int v : adjs[u])
        if(!vis[v]){
            dfs(v, h + 1);
            euler.push_back(u);
        }
}

int query(int v, int tl, int tr, int l, int r){
    if(l > r)
        return -1;
    if(tl == l && tr == r)
        return t[v];
    int tm = (tl + tr) / 2;
    int p1 = query(2 * v, tl, tm, l, min(tm, r));
    int p2 = query(2 * v + 1, tm + 1, tr, max(tm + 1, l), r);
    if(p1 == -1)
        return p2;
    if(p2 == -1)
```

```
        return p1;
    return (height[p1] <= height[p2]) ? p1 : p2;
}

int lca(int u, int v){
    if(first[u] > first[v])
        swap(u, v);
    return query(1, 0, euler.size() - 1, first[u], first[v]);
}

int main(){
    cin >> n;
    for(int i = 0; i < n - 1; i++){
        int u, v;
        cin >> u >> v;
        adjs[u].push_back(v), adjs[v].push_back(u);
    }
    dfs(0, 0);
    build(1, 0, euler.size() - 1);
    int q;
    cin >> q;
    while(q--){
        int u, v;
        cin >> u >> v;
        cout << lca(u, v) << endl;
    }
}
```

31 line

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

#define INF LLONG_MAX

struct line{
    double m, b;

 line() = default;

    line(point& p1, point& p2){
        if(fabs(p1.x - p2.x) < EPS)
            this->m = INF, this->b = p1.x;
        else if(fabs(p1.y - p2.y) < EPS)
            this->m = 0, this->b = p1.y;
        else{
            this->m = (p1.y - p2.y) / (p1.x - p2.x);
            this->b = p1.y - this->m * p1.x;
        }
    }

    line& operator=(const line& o) = default;

    bool parallel(const line& o) const{
        return fabs(this->m - o.m) < EPS;
    }

    bool operator==(const line& o) const{
        return parallel(o) && (fabs(this->b - o.b) < EPS);
    }

    point intersect(const line& o) const{
        if(this->m == INF)
            return point(this->b, o.m * this->b + o.b);
        if(o.m == INF)
            return point(o.b, this->m * o.b + this->b);
        double x = (o.b - this->b) / (this->m - o.m);
        return point(x, this->m * x + this->b);
    }
};

bool on_line_segment(point& a, point& b, point& c){
    return fabs(a.dist(b) - c.dist(a) - c.dist(b)) < EPS;
}
```

32 lis

```
vector<int> sequence;
vector<int> lis;
vector<int> lis_index;
vector<int> trace;
vector<int> path;

void clis(){
    int num;
    while(cin >> num)
        sequence.push_back(num);
    trace.resize(sequence.size(), -1);
    for(int i = 0; i < sequence.size(); i++){
        int num = sequence[i];
        int ptr = lower_bound(lis.begin(), lis.end(), num) -
             lis.begin();
        if(ptr == lis.size()){
            lis.push_back(num);
            lis_index.push_back(i);
        }
        else {
            lis[ptr] = num;
            lis_index[ptr] = i;
        }
        if(ptr > 0)
            trace[i] = lis_index[ptr-1];
    }
    cout << lis.size() << endl;
    cout << "-" << endl;
    int it = lis_index[lis.size()-1];
    while(it != -1){
        path.push_back(sequence[it]);
        it = trace[it];
    }
    for(int i = path.size() - 1; i >= 0; i--)
        cout << path[i] << endl;
}
```

33 maxflow

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

// Edmond Karp's Network Flow

// O(V * E ^ 2)

const int maxn = 503;
const int inf = INT_MAX;

// s: source, t: sink, c: capacity
int n, m, s, t, c[maxn][maxn], flow[maxn][maxn], par[maxn];
vector<int> adj[maxn];

int augment(){
    fill_n(par, n, -1);
    par[s] = -2;
    queue<pair<int, int>> bfs;
    bfs.push({s, inf});
    while(!bfs.empty()){
        int u = bfs.front().first, f = bfs.front().second;
        bfs.pop();
        if(u == t)
            return f;
        for(int v : adj[u])
            if(par[v] == -1 && c[u][v] - flow[u][v] > 0)
```

```
                par[v] = u, bfs.push({v, min(f, c[u][v] - flow
                    [u][v])});
    }
    return 0;
}

int max_flow(){
    int ans = 0;
    while(1){
        int f = augment();
        if(!f)
            break;
        ans += f;
        int it = t;
        while(it != s){
            flow[par[it]][it] += f;
            flow[it][par[it]] -= f;
            it = par[it];
        }
    }
    return ans;
}

// For Minimum Cut Problem:

bool vis[maxn];
vector<int> sCut, tCut;

void dfs(int u){
    vis[u] = true;
    for(int v : adj[u])
        if(!vis[v] && c[u][v] - flow[u][v] > 0)
            dfs(v);
}


int main(){
 // read the graph
    cout << max_flow() << endl;
    dfs(s);
    for(int i = 0; i < n; i++){
        if(vis[i])
            sCut.push_back(i);
        else
            tCut.push_back(i);
    }
}
```

34 mcbm

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

// Maximum Cardinality Bipartite Matching
// O(V * E)

const int maxn = 1003;
const int maxm = 1003;

// n is |left set|, m is |right set|, edges are from left
    set to right set
int n, m;
vector<int> adjs[maxn];
bool vis[maxn];
int match[maxm]; // if match[v] = u then u from left set is
    matched with v from right set

int aug(int u){
    if(vis[u])
        return 0;
    vis[u] = 1;
    for(int v : adjs[u])
        if(match[v] == -1 || aug(match[v])){
            match[v] = u;
            return 1;
        }
    return 0;
}

int main(){
    // read the graph
    int ans = 0;
    fill_n(match, m, -1);
    for(int i = 0; i < n; i++){
        fill_n(vis, n, 0);
        ans += aug(i);
    }
    cout << ans << endl;
}
```

35 mincostmaxflow

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

// Minimum Cost Maximum Flow

#define num long long int

const int maxn = 253;
const num inf = LLONG_MAX;

int N, M, src, sink;
bool found[maxn];
int cap[maxn][maxn], flow[maxn][maxn], par[maxn];
num picked[maxn], dis[maxn], cost[maxn][maxn];

bool aug(){
    fill_n(found, N, false);
    fill_n(dis, N + 1, inf);
    int it = src;
    dis[it] = 0;
    while(it != N){
        int best = N;
        found[it] = true;
        for(int k = 0; k < N; k++){
            if(found[k])
                continue;
            if(flow[k][it] != 0){
                num val = dis[it] + picked[it] - picked[k] -
                     cost[k][it];
                if(dis[k] > val)
                    dis[k] = val, par[k] = it;
            }
            if(flow[it][k] < cap[it][k]){
                num val = dis[it] + picked[it] - picked[k] +
                     cost[it][k];
                if(dis[k] > val)
                    dis[k] = val, par[k] = it;
            }
            if(dis[k] < dis[best])
                best = k;
        }
        it = best;
    }
    for(int k = 0; k < N; k++)
        if(dis[k] != inf)
            picked[k] += dis[k];
    return found[sink];
}

pair<int, num> mincost_maxflow(){
    int tf = 0;
    num tc = 0;
    while(aug()){
```

```
        int f = INT_MAX;
        for(int x = sink; x != src; x = par[x]){
            if(flow[x][par[x]] != 0)
                f = min(f, flow[x][par[x]]);
            else
                f = min(f, cap[par[x]][x] - flow[par[x]][x]);
        }
        tf += f;
        for(int x = sink; x != src; x = par[x]){
            if(flow[x][par[x]] != 0){
                flow[x][par[x]] -= f;
                tc -= cost[x][par[x]] * f;
            }
            else{
                flow[par[x]][x] += f;
                tc += cost[par[x]][x] * f;
            }
        }
    }
    return {tf, tc};
}

int main(){
    cin >> N >> M >> src >> sink;
    int x, y;
    while(M--)
        cin >> x >> y >> cap[x][y] >> cost[x][y];
    auto ans = mincost_maxflow();
    cout << ans.first << " " << ans.second << endl;
}
```

36 mincoverage

```
int n;
pair<int, int> intervals[10003];
vector<int> indices;

int min_coverage(int A, int B){ // -1 means impossible
    sort(intervals, intervals+n);
    indices.clear();
    int coverage = A, i = -1, cnt = 0;
    while(i < n){
        int cand = -1, maxl = INT_MIN;
        for(int j = i + 1; j < n; j++)
            if(intervals[j].first <= coverage && intervals[j
                ].second >= maxl)
                cand = j, maxl = intervals[j].second;
        if(cand == -1)
            break;
        indices.push_back(cand);
        coverage = maxl, i = cand, cnt++;
        if(coverage >= B)
            break;
    }
    return coverage >= B ? cnt : -1;
}
```

37 moore

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

// Moore Algorithm to minimize the number of late jobs.

// O(n * logn)

#define num long long int

const int maxn = 1e5 + 3;
int n, m;
pair<num, num> interval[maxn]; // first: needed time, second
    : deadline

bool cmp(pair<num, num> & p1, pair<num, num> & p2){
    return p1.second < p2.second;
}

num moore(){
    priority_queue<num> pq;
    num t = 0;
    for(int i = 0; i < n; i++){
        num needed_time = interval[i].first;
        pq.push(needed_time);
        if(t + needed_time <= interval[i].second)
            t += needed_time;
        else{
            num p = pq.top();
            pq.pop();
            t = t - p + needed_time;
        }
    }
    return n - (int)pq.size();
}

int main(){
    // read the input
    sort(interval, interval + n, cmp);
    cout << moore() << endl;
}
```

38 point

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

#define EPS 1e-8

struct point{
    double x, y;

    point() = default;

    point(double x, double y){
        this->x = x, this->y = y;
    }

 point& operator=(const point& o) = default;

    bool operator==(const point& o) const{
        return fabs(this->x - o.x) < EPS && fabs(this->y - o.
            y) < EPS;
    }

    bool operator<(const point& o) const{
        if(fabs(this->x - o.x) < EPS)
            return this->y < o.y;
        return this->x < o.x;
    }

    double dist(const point& o){
        return hypot(this->x - o.x, this->y - o.y);
    }

    point rotate(double rad){
        return point(this->x * cos(rad) - this->y * sin(rad),
            this->x * sin(rad) + this->y * cos(rad));
    }
};

inline double deg_to_rad(double deg){
    return (deg * M_PI) / 180;
}

inline double rad_to_dag(double rad){
```

```
    return (rad * 180) / M_PI;
}
```

39 prim

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

// MST

// Prim's Algorithm

// O(E * logV)

const int maxn = 1003;

int n;
vector<pair<int, int>> adjs[maxn];
priority_queue<pair<int, int>, vector<pair<int, int>>,
     greater<pair<int, int>>> pq;
bool taken[maxn];
set<int> srcs;

void prim(int u){
    taken[u] = 1;
    for(auto x : adjs[u]){
        int v = x.second, w = x.first; // w is the weight
        if(!taken[v])
            pq.push({w, v});
    }
}

int main(){
    // read the graph
    for(int s : src)
        prim(s);
    int mst = 0;
    while(!pq.empty()){
        auto p = pq.top();
        pq.pop();
        int u = p.second, w = p.first;
        if(taken[u])
            continue;
        mst += w;
        prim(u);
    }
    cout << mst << endl;
}
```

40 primesieve

```
#include <bits/stdc++.h>
using namespace std;
#define MAX_N 10000010
long long int _sieve_size;
bitset<MAX_N> pFlags;
long long int spf[MAX_N]; // smallest pf of numbers
vector<long long int> PF;

void sieve(long long int upperbound){
    _sieve_size = upperbound + 1;
    pFlags.set();
    pFlags[0] = pFlags[1] = 0;
    for(long long int q = 4; q <= _sieve_size; q += 2){
        pFlags[q] = 0;
        spf[q] = 2;
    }
    PF.push_back(2);
    for(long long int p = 3; p <= _sieve_size; p += 2){ //
         constraint can be p * p <= _sieve_size if we don't
         need to calc PF
        if(pFlags[p]){
            spf[p] = p;
            for(long long int q = p * p; q <= _sieve_size; q
                 += p){
 if(pFlags[q]){
                pFlags[q] = 0;
                spf[q] = p;
 }
            }
            PF.push_back(p);
        }
    }
}

bool isPrime(long long int N) {
    if (N <= _sieve_size)
        return pFlags[N];
    for (int i = 0; i < (int)PF.size(); i++){
        if (N % PF[i] == 0)
            return false;
    }
    return true; // it takes longer time if N is a large
         prime!
} // note: only work for N <= (last prime in PF)^2

int main()
{
    sieve(10000000);
    printf("%d\n", isPrime(2147483647));
    printf("%d\n", isPrime(136117223861*1LL));
    return 0;
}
```

41 primesievefunctions

```
vector<long long int> factors;
long long int primeFactors(long long int N) { // and also
     sumPF
    long long int PF_idx = 0, primefactor = PF[PF_idx], res =
         0;
    while (primefactor * primefactor <= N) { // stop at sqrt(
         N); N can get smaller
        while (N % primefactor == 0){
            N /= primefactor; // remove PF
            factors.push_back(primefactor);
            res += primefactor;
        }
        primefactor = PF[++PF_idx]; // only consider primes
    }
    if (N != 1)
        factors.push_back(N), res += N; // special case if N
             is a prime
    return res;
}

long long int numPF(long long int N) {
    long long int PF_idx = 0, primefactor = PF[PF_idx], ans =
         0;
    while (primefactor * primefactor <= N) {
        while (N % primefactor == 0){
            N /= primefactor;
            ans++;
        }
        primefactor = PF[++PF_idx];
    }
    if (N != 1)
        ans++;
    return ans;
}

long long int numDiv(long long int N) {
    long long int PF_idx = 0, primefactor = PF[PF_idx], ans =
          1; // start from ans = 1
    while (primefactor * primefactor <= N) {
        long long int power = 0;
```

```
        while (N % primefactor == 0) { N /= primefactor;
             power++; }
        ans *= (power + 1); // according to the formula
        primefactor = PF[++PF_idx];
    }
    if (N != 1)
        ans *= 2;// (last factor has pow = 1, we add 1 to it)
    return ans;
}

long long int sumDiv(long long int N) {
    long long int PF_idx = 0, primefactor = PF[PF_idx], ans =
         1; // start from ans = 1
    while (primefactor * primefactor <= N) {
        long long int power = 0;
        while (N % primefactor == 0){
            N /= primefactor;
            power++;
        }
        ans *= ((long long int)pow((double)primefactor, power
             + 1.0) - 1) / (primefactor - 1);
        primefactor = PF[++PF_idx];
    }
    if (N != 1)
        ans *= ((long long int)pow((double)N, 2.0) - 1) / (N
             - 1); // last
    return ans;
}

long long int EulerPhi(long long int N) { // Count the
     number of positive integers < N that are relatively
     prime to N
    long long int PF_idx = 0, primefactor = PF[PF_idx], ans =
         N; // start from ans = N
    while (primefactor * primefactor <= N) {
        if (N % primefactor == 0)
            ans -= ans / primefactor; // only count unique
                 factor
        while (N % primefactor == 0)
            N /= primefactor;
        primefactor = PF[++PF_idx];
    }
    if (N != 1)
        ans -= ans / N; // last factor
    return ans;
}

int numDiffPF[MAX_N];
void modified_sieve(){ // calculate number of diffferent PFs
    for (int i = 2; i < MAX_N; i++)
        if (numDiffPF[i] == 0) // i is a prime number
            for (int j = i; j < MAX_N; j += i)
                numDiffPF[j]++; // increase the values of
                     multiples of i
}
```

42 readlines

```
import sys
lines = sys.stdin.readlines()
for line in lines:
    for ch in line.strip():
        pass
```

43 scc

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

// Finding SCCs

// Tarjan's Algorithm

// O(V + E)

#define MAX_N 100003

int n, m, nums = 1, scc = 0;
int dfs_low[MAX_N], dfs_num[MAX_N];
int compo[MAX_N];
int compoSize[MAX_N];
vector<int> adj[MAX_N];
vector<int> s;
int id[MAX_N];
bool visited[MAX_N];

void dfs(int u){
    dfs_num[u] = dfs_low[u] = nums++;
    s.push_back(u);
    visited[u] = true;
    for(auto v: adj[u]){
        if(!dfs_num[v])
            dfs(v);
        if(visited[v])
            dfs_low[u] = min(dfs_low[u], dfs_low[v]);
    }
    if(dfs_low[u] == dfs_num[u]){
        scc++;
        while(true){
            int w = s.back();
            s.pop_back();
            visited[w] = false;
            compo[w] = scc;
            id[w] = compoSize[scc]++;
            if(w == u)
                break;
        }
    }
}

int main(){
    // read the graph
    for (int i = 0; i < n; i++)
        if(!dfs_num[i])
            dfs(i);
}
```

44 segmenttree

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

// Segment Tree Implementation

// Example: Bitmask of Distinct Characters on Intervals of a
     String

const int maxn = 1e5;

string s;
int t[4 * maxn + 3];

// O(n * log(n))
void build(int v, int tl, int tr){
    if(tl == tr)
        t[v] |= (1 << (s[tl] - 'a')); // Base Case
    else{
        int tm = (tl + tr) / 2;
        build(2 * v, tl, tm);
        build(2 * v + 1, tm + 1, tr);
        t[v] = t[2 * v] | t[2 * v + 1]; // Combine
    }
}
```

```
// O(log(n))
void update(int v, int tl, int tr, int pos, char c){
    if(tl == tr)
        t[v] = (1 << (c - 'a')); // Base Case
    else{
        int tm = (tl + tr) / 2;
        if(pos <= tm)
            update(2 * v, tl, tm, pos, c);
        else
            update(2 * v + 1, tm + 1, tr, pos, c);
        t[v] = t[2 * v] | t[2 * v + 1]; // Combine
    }
}

// O(log(n))
int query(int v, int tl, int tr, int l, int r){
    if(l > r)
        return 0; // Nill
    if(l == tl && r == tr)
        return t[v]; // Hit
    int tm = (tl + tr) / 2;
    int mask1 = query(2 * v, tl, tm, l, min(tm, r));
    int mask2 = query(2 * v + 1, tm + 1, tr, max(l, tm + 1),
         r);
    return mask1 | mask2; // Combine
}

int main(){
    cin >> s;
 int n = s.size();
    build(1, 0, n - 1);
 // handle queries and updates, initially put v = 1, tl = 0,
       tr = n - 1
}
```

45 split

```
vector<string> split(const string& str, char delim){
    vector<string> cont;
    stringstream ss(str);
    string token;
    while(getline(ss, token, delim))
        cont.push_back(token);
    return cont;
}
```

46 suffixarray

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

// Suffix Array Implementation

const int maxn = 1e5 + 3;

string s = "GATAGACA$", p = "GA";
int n = s.size(), m = p.size();
int sa[maxn], ra[maxn], cnt;
vector<pair<pair<int, int>, int>> tmp;
pair<int, int> prevy, nil = {-1, -1};

inline pair<int, int> find_match(){
    // finding lower bound
    int lo = 0, hi = n - 1;
    while(lo < hi){
        int mid = (lo + hi) / 2;
        if(strncmp(s.c_str() + sa[mid], p.c_str(), m) >= 0)
            hi = mid;
        else
            lo = mid + 1;
    }
    if(strncmp(s.c_str() + sa[lo], p.c_str(), m))
        return nil;
    pair<int, int> ans = {lo, -1};
    // finding (inclusive) upper bound
    lo = 0, hi = n-1;
    while(lo < hi){
        int mid = (lo + hi) / 2;
        if(strncmp(s.c_str() + sa[mid], p.c_str(), m) > 0)
            hi = mid;
        else
            lo = mid + 1;
    }
    if(strncmp(s.c_str() + sa[hi], p.c_str(), m))
        hi--;
    ans.second = hi;
    return ans;
}

int phi[maxn], plcp[maxn], lcp[maxn];

inline void computeLCP(){
    phi[sa[0]] = -1;
    for(int i = 1; i < n; i++)
        phi[sa[i]] = sa[i-1];
    int l = 0;
    for(int i = 0; i < n; i++){
        if(phi[i] == -1){
            plcp[i] = 0;
            continue;
        }
        while(s[i + l] == s[phi[i] + l])
            l++;
        plcp[i] = l;
        l = max(0, l-1);
    }
    for(int i = 0; i < n; i++)
        lcp[i] = plcp[sa[i]];
}

int main(){
    cin >> s;
    s += '$';
    n = s.size();
    cin >> p;
    m = p.size();
    for(int i = 0; i < n; i++)
        sa[i] = i, ra[i] = s[sa[i]];
    for(int k = 1; k < n; k *= 2){
        tmp.clear();
        for(int i = 0; i < n; i++)
            tmp.push_back({{ra[sa[i]], (sa[i] + k >= n) ? 0 :
                 ra[sa[i] + k]}, sa[i]});
        sort(tmp.begin(), tmp.end());
        cnt = -1;
        prevy = {-1, -1};
        for(int i = 0; i < n; i++){
            sa[i] = tmp[i].second;
            if(tmp[i].first != prevy)
                cnt++;
            ra[sa[i]] = cnt;
            prevy = tmp[i].first;
        }
        if(ra[sa[n-1]] == n-1)
            break;
    }
    computeLCP();
    for(int i = 0; i < n; i++)
        cout << sa[i] << " " << s.substr(sa[i]) << " " << lcp
             [i] << endl;
    pair<int, int> ans = find_match();
    if(ans == nil)
        cout << "not found" << endl;
    else for(int i = ans.first; i <= ans.second; i++)
        cout << sa[i] << endl;
}
```

47 topol

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

// Topological Sort

// O(V + E)

#define MAX_N 1000003
#define UNVISITED 0
#define EXPLORED 1
#define VISITED 2

vector<int> adj[MAX_N];
int visited[MAX_N];
int parent[MAX_N];
int n, m, a, b;
bool res = true;
vector<int> resv;

void dfs(int u){
    visited[u] = EXPLORED;
    for(auto v : adj[u]){
        if(!visited[v]){
            parent[v] = u;
            dfs(v);
        }
        else if(visited[v] == EXPLORED) // check cycle (DAG)
            res = false;
    }
    visited[u] = VISITED;
    resv.push_back(u);
}

int main(){
    // read the graph
    for(int i = 0; i < n; i++)
        if(!visited[i])
            dfs(i);
    if(!res)
        cout << "IMPOSSIBLE" << endl;
    else
        for(int i = resv.size() - 1; i >= 0; i--)
            cout << resv[i] << endl;
}
```

48 toruspsum

```cpp
int n;
int psum[78][78];

int get(int i, int j){
    if(i >= 0 && i < n && j >= 0 && j < n)
        return psum[i][j];
    return 0;
}

void build(){
 for(int i = 0; i < n; i++)
  for(int j = 0; j < n; j++){
   cin >> psum[i][j];
   psum[i][j] += get(i-1, j) + get(i, j-1) - get(i-1, j-1);
       }
}

int rect(int x1, int y1, int x2, int y2){
    return psum[x2][y2] - get(x1-1, y2) - get(x2, y1-1) + get
        (x1-1, y1-1);
}

int torus(){
 int max_sum = INT_MIN;
 for(int x1 = 0; x1 < n; x1++)
  for(int y1 = 0; y1 < n; y1++)
   for(int x2 = 0; x2 < n; x2++)
    for(int y2 = 0; y2 < n; y2++){
     int area;
     if(x2 >= x1 && y2 >= y1)
      area = rect(x1, y1, x2, y2);
     else if(x2 >= x1 && y2 < y1)
      area = rect(x1, 0, x2, y2) + rect(x1, y1, x2, n-1);
     else if(x2 < x1 && y2 >= y1)
      area = rect(0, y1, x2, y2) + rect(x1, y1, n-1, y2);
     else
      area = rect(0, 0, x2, y2) + rect(x1, y1, n-1, n-1) +
           rect(x1, 0, n-1, y2) + rect(0, y1, x2, n-1);
     max_sum = max(max_sum, area);
    }
 return max_sum;
}
```

49 trie

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

// Trie Implementation

class trie{
    public:
    trie* adj[26+3];
    trie* parent = nullptr;
    bool terminal = false;

    trie(){
        fill_n(adj, 26, nullptr);
    }

    trie(trie* parent){
        fill_n(adj, 26, nullptr);
        this->parent = parent;
    }
};

trie* root = new trie();

// O(|S|)
inline void trie_insert(string s){
    trie* t = root;
    for(int i = 0; i < s.size(); i++){
        int ind = s[i] - 'A'; // use 'a' for lowercase
            letters
        if(t->adj[ind] == nullptr)
            t->adj[ind] = new trie(t);
        t = t->adj[ind];
    }
    t->terminal = true;
}

// O(|S|)
inline bool trie_find(string s){
    trie* t = root;
    for(int i = 0; i < s.size(); i++){
        int ind = s[i] - 'A'; // use 'a' for lowercase
            letters
        if(t->adj[ind] == nullptr)
            return false;
        t = t->adj[ind];
    }
    return t->terminal;
}

int main(){
```

```
    // Example:
    trie_insert("ABC");
    trie_insert("KARIM");
    trie_insert("KERIM");
    trie_insert("ABCD");
    trie_insert("GHAZAL");
    string s;
    while(cin >> s)
        cout << trie_find(s) << endl;
}
```

50 vector

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

struct vec{
    double x, y;

 vec() = default;

    vec(double x, double y){
        this->x = x;
        this->y = y;
    }

    vec(point& p1, point& p2){
        this->x = p2.x - p1.x;
        this->y = p2.y - p1.y;
    }

    vec& operator=(const vec& o) = default;

    vec scale(double s){
        return vec(this->x * s, this->y * s);
    }

    point translate(point& p){
        return point(p.x + this->x, p.y + this->y);
    }

    double dot(vec& o){
        return this->x * o.x + this->y * o.y;
    }

    double norm_squared(){
        return this->x * this->x + this->y * this->y;
    }

    double norm(){
        return hypot(this->x, this->y);
    }

    double cross(vec& o){
        return this->x * o.y - this->y * o.x;
    }

 vec operator+(const vec& o) const{
        return vec(this->x + o.x, this->y + o.y);
    }
};

double dist_to_line(point& p, point& a, point& b){ // a, b
     are 2 different points on the given line.
    vec ap = vec(a, p);
    vec ab = vec(a, b);
    double u = ab.dot(ap) / ab.norm_squared();
    return ab.scale(u).translate(a).dist(p); // c = u * ab +
         a
}

double dist_to_line_segment(point& p, point& a, point& b){
    vec ap = vec(a, p);
    vec ab = vec(a, b);
    double u = ab.dot(ap) / ab.norm_squared();
    if(u < 0)
        return p.dist(a);
    if(u > 1)
        return p.dist(b);
    return dist_to_line(p, a, b);
}

double dist_of_line_segments(point& p1, point& p2, point& q1
     , point& q2){
    line l1(p1, p2), l2(q1, q2);
    if(!l1.parallel(l2)){
        point inter = l1.intersect(l2);
        if(on_line_segment(p1, p2, inter) && on_line_segment(
             q1, q2, inter))
            return 0.0;
    }
    double ans1 = min(dist_to_line_segment(q1, p1, p2),
         dist_to_line_segment(q2, p1, p2));
    if(p1 == p2)
        ans1 = min(p1.dist(q1), p1.dist(q2));
    double ans2 = min(dist_to_line_segment(p1, q1, q2),
         dist_to_line_segment(p2, q1, q2));
    if(q1 == q2)
        ans2 = min(q1.dist(p1), q1.dist(p2));
    return min(ans1, ans2);
}

double angle(point& a, point& o, point& b){
    vec oa = vec(o, a);
    vec ob = vec(o, b);
    return acos(oa.dot(ob) / (oa.norm() * ob.norm()));
}

// to determine whether point r is on the left/right side of
     the line.
bool ccw(point& p, point& q, point& r){
    vec v(p, r);
    return vec(p, q).cross(v) > -EPS;
}

bool collinear(point& p, point& q, point& r){
    vec v(p, r);
    return fabs(vec(p, q).cross(v)) < EPS;
}
```

51 zigzag

```
int n, a[1000003];

int zigzag(bool inc){ // true for decrease first
    int ans = 1;
    for(int i = 1; i < n; i++){
        if(inc && a[i] < a[i-1])
            inc = false, ans++;
        if(!inc && a[i] > a[i-1])
            inc = true, ans++;
    }
    return ans;
}
```