**KMP**

**int** N, W, k, cnt;

**char** c;

**int** Q[MAXL];

**char** pattern[MAXL], word[MAXL];

**int** **main**() {

cin >> word >> pattern;

**memset**(Q, 0, **sizeof** Q);

N = **strlen**(pattern);

// Compute KMP automaton

// (the so-called failure function)

k = Q[0] = -1;

**for** (**int** i = 1; i < N; i++) {

**while** (k >= 0 && pattern[k + 1] != pattern[i])

k = Q[k];

**if** (pattern[k + 1] == pattern[i])

k++;

Q[i] = k;

}

// Find occurrences

k = -1;

W = **strlen**(word);

**int** sol = 0;

**for** (**int** i = 0; i < W; ++i) {

c = word[i];

**while** (k >= 0 && pattern[k + 1] != c)

k = Q[k];

**if** (pattern[k + 1] == c)

k++;

cnt++;

**if** (k + 1 == N) {

sol++;

k = Q[k];

}

}

cout << sol << "\n";

**return** 0;

}

**BFS**

**void** **bfs**(**int** s) {

queue<**int**> q;

q.push(s);

visited[s] = **true**;

**while** (!q.empty()) {

**int** u = q.front();

q.pop();

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

**int** v = graph[u][i];

**if** (!visited[v]) {

visited[v] = **true**;

q.push(v);

}

}

}

}

**Topological Sort**

**const** **int** MAXN = 1000000;

vector<**int**> adjList[MAXN];

**int** degrees[MAXN], order[MAXN], p[MAXN];

**int** m, n;

**int** **main**() {

ios\_base::**sync\_with\_stdio**(0);

cin.tie(0);

**freopen**("file", "r", stdin);

**while** (**true**) {

**memset**(degrees, 0, **sizeof**(degrees));

**memset**(order, 0, **sizeof**(order));

**memset**(p, -1, **sizeof**(p));

cin >> n >> m;

**for** (**int** i = 0; i < MAXN; ++i) {

adjList[i].clear();

}

**if** (m == 0 && n == 0) {

**break**;

}

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

**int** a, b;

cin >> a >> b;

degrees[b]++;

adjList[a].push\_back(b);

}

queue<**int**> q;

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

**if** (degrees[i] == 0) {

q.push(i);

}

}

**int** k = 0;

**while** (!q.empty()) {

**int** u = q.front();

q.pop();

order[k++] = u;

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

**int** v = adjList[u][i];

degrees[v]--;

**if** (degrees[v] == 0) {

q.push(v);

}

}

}

**if** (k != n) {

cout << "IMPOSSIBLE\n";

} **else** {

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

cout << order[i] << "\n";

}

}

}

**return** 0;

}

**Dijkstra con aristas doblemente ponderadas**

**struct** nodo {

**int** u, km, sol;

**bool** **operator<**(**const** nodo &n) **const** {

**return** (sol == n.sol) ? km < n.km : sol < n.sol;

}

};

**struct** comparator {

**bool** **operator()**(**const** nodo& a, **const** nodo& b) **const** {

**return** (a.sol == b.sol) ? a.km > b.km : a.sol > b.sol;

}

};

nodo d[MAXV];

vector<nodo> adjList[MAXV];

**int** **main**() {

**int** V, E, source, dest;

**memset**(d, 0, **sizeof**(d));

**for** (**int** i = 0; i < MAXV; ++i) {

adjList[i].clear();

}

cin >> V >> E >> source >> dest;

**for** (**int** i = 0; i < E; ++i) {

**int** u, v, w, sun;

cin >> u >> v >> w >> sun;

nodo n1, n2;

n1.u = u, n1.km = w, n1.sol = w - sun;

n2.u = v, n2.km = w, n2.sol = w - sun;

adjList[u].push\_back(n2);

adjList[v].push\_back(n1);

}

priority\_queue<nodo, vector<nodo>, comparator> Q;

**for** (**int** i = 0; i < V; i++)

d[i] = (nodo ) { i, oo, oo };

nodo s = (nodo ) { source, 0, 0 };

d[source] = s;

**for** (Q.push(s); !Q.empty(); Q.pop()) {

nodo top = Q.top();

**int** u = top.u;

**int** sol\_u = top.sol;

**if** (sol\_u <= d[u].sol)

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

**int** v = adjList[u][i].u;

**int** sol = adjList[u][i].sol;

**int** km = adjList[u][i].km;

**if** (d[u].sol + sol < d[v].sol) {

d[v].sol = d[u].sol + sol;

d[v].km = d[u].km + km;

Q.push(d[v]);

} **else** **if** (d[u].sol + sol == d[v].sol) {

**if** (d[u].km + km < d[v].km) {

d[v].km = d[u].km + km;

Q.push(d[v]);

}

}

}

}

cout << d[dest].km << " " << d[dest].sol << "\n";

**return** 0;

}

**Second MST**

**#include** <bits/stdc++.h>

**using** **namespace** std;

**typedef** pair<**int**, **int**> ii;

**const** **int** MAXV = 105;

**const** **int** MAXE = 5100;

**const** **int** oo = 10000000;

**struct** edge {

**int** u, v, w;

**bool** **operator<**(**const** edge& e) **const** {

**return** w < e.w;

}

**bool** **operator!=**(**const** edge& e) **const** {

**return** u != e.u || v != e.v || w != e.w;

}

};

**int** ite, v, e, a, b, c, p[MAXV], mst, mst2;

vector<edge> edges;

vector<**int**> mstEdges;

**int** **secondMST**(**int** off) {

**int** mstemp = 0;

**int** cont = 0;

initialize(v);

**for** (**int** i = 0; i < e && cont < v - 1; ++i) {

**int** u = edges[i].u;

**int** v = edges[i].v;

**int** w = edges[i].w;

**int** set1 = find\_set(u);

**int** set2 = find\_set(v);

**if** (set1 != set2 && i != off) {

mstemp += w;

cont++;

merge(set1, set2);

}

}

**if** (cont == v - 1) {

**return** mstemp;

} **else** {

**return** oo;

}

}

**int** **main**() {

// ... Normal MST ...

mst2 = oo;

**for** (**int** k = 0; k < v - 1; ++k) {

mst2 = min(mst2, secondMST(mstEdges[k]));

}

cout << mst << " " << mst2 << "\n";

}

**return** 0;

}

**Euler Path**

**#define** MAXI 500

**#define** MAXF 1200

**int** conn[MAXI][MAXI], deg[MAXI], nconn, path[MAXF], plen;

**void** **find\_path**(**int** loc) {

**int** lv;

**for** (lv = 0; lv < nconn; lv++)

**if** (conn[loc][lv]) {

conn[loc][lv]--;

conn[lv][loc]--;

deg[lv]--;

deg[loc]--;

find\_path(lv);

}

path[plen++] = loc;

}

**int** **main**() {

**freopen**("file", "r", stdin);

**int** nfen;

**int** lv;

**int** x, y;

**while** (**scanf**("%d", &nfen) != EOF) {

**memset**(conn, 0, **sizeof** conn);

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

**memset**(path, 0, **sizeof** path);

plen = 0;

**for** (lv = 0; lv < nfen; lv++) {

**scanf**("%d %d", &x, &y);

x--;

y--;

conn[x][y]++;

conn[y][x]++;

deg[x]++;

deg[y]++;

**if** (x >= nconn)

nconn = x + 1;

**if** (y >= nconn)

nconn = y + 1;

}

**for** (lv = 0; lv < nconn; lv++)

**if** (deg[lv] % 2 == 1)

**break**;

**if** (lv >= nconn)

**for** (lv = 0; lv < nconn; lv++)

**if** (deg[lv])

**break**;

find\_path(lv);

**for** (lv = plen - 1; lv >= 0; lv--)

**printf**("%i\n", path[lv] + 1);

}

}