#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

#include <climits>

using namespace std;

const int MAX = 1e4 + 5;

typedef pair<int, int> pii;

vector<pii> graph[MAX], originalGraph[MAX];

int dist[MAX];

bool visited[MAX];

void dijkstra(int s, vector<pii> graph[]) {

fill(dist, dist + MAX, INT\_MAX);

memset(visited, false, sizeof(visited));

priority\_queue<pii, vector<pii>, greater<pii>> pq;

pq.push({0, s});

dist[s] = 0;

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

if (visited[u]) continue;

visited[u] = true;

for (auto &edge : graph[u]) {

int v = edge.first;

int w = edge.second;

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

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

pq.push({dist[v], v});

}

}

}

}

int main() {

int n, m, s, t;

cin >> n >> m >> s >> t;

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

int u, v, w;

cin >> u >> v >> w;

originalGraph[u].push\_back({v, w});

}

dijkstra(s, originalGraph);

int originalDistance = dist[t];

int maxIncrease = 0;

pii criticalEdge;

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

for (auto &edge : originalGraph[u]) {

int v = edge.first;

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

graph[i] = originalGraph[i];

}

graph[u].erase(remove(graph[u].begin(), graph[u].end(), edge), graph[u].end());

dijkstra(s, graph);

int newDistance = dist[t];

if (newDistance > originalDistance && newDistance - originalDistance > maxIncrease) {

maxIncrease = newDistance - originalDistance;

criticalEdge = {u, v};

}

}

}

if (maxIncrease > 0)

cout << "Xóa cạnh (" << criticalEdge.first << ", " << criticalEdge.second << ") làm tăng đường đi từ " << s << " đến " << t << " lên " << maxIncrease << endl;

else

cout << "Không có cạnh nào làm tăng độ dài đường đi ngắn nhất từ " << s << " đến " << t << endl;

return 0;

}