#include <iostream>  
#include <vector>  
#include <map>  
#include <queue>  
#include <functional>  
#include <algorithm>  
#include <limits>  
  
int INF = std::numeric\_limits<int>::max();  
  
//  
std::pair<std::map<int, int>, int> DPA(const std::map<int, std::vector<int>> &graph, const std::map<std::pair<int, int>, int> &wages, int start) {  
 int sum = 0; // suma  
 std::map<int, int> result = {}; // A  
 std::map<int, int> alfa = {}; // alfa  
 std::map<int, int> beta = {}; // beta  
 std::vector<int> queue = {}; // kolejka  
 std::vector<int> all\_vertex = {}; // V  
 int new\_vertex = 0; // u\*  
  
  
 for (const auto &item : graph){  
 alfa[item.first] = 0;  
 beta[item.first] = INF;  
 queue.push\_back(item.first);  
 all\_vertex.push\_back(item.first);  
 }  
  
 beta[start] = 0;  
  
 auto tmp = std::find(queue.cbegin(), queue.cend(), start);  
 queue.erase(tmp);  
  
 new\_vertex = start;  
 while (!queue.empty()) {  
 for (auto u : queue) {  
 if (std::find(graph.find(new\_vertex)->second.begin(), graph.find(new\_vertex)->second.end(), u) != graph.find(new\_vertex)->second.cend()) {  
 if (wages.find({u, new\_vertex})->second < beta[u]) {  
 alfa[u] = new\_vertex;  
 beta[u] = wages.find({u, new\_vertex})->second;  
 }  
 }  
 }  
// for (auto u : queue) {  
// new\_vertex = ;  
// }  
 //*todo znalezc minimum w beta a nastepnie index wierzcholka dla jakiego to min wystepuje* }  
}  
  
  
  
int main() {  
 std::map<int, std::vector<int>> g1 = **{** {0, {2, 3}},  
 {1, {0, 2, 3}},  
 {2, {0, 1}},  
 {3, {0, 1}}  
 **}**;  
  
 std::map<std::pair<int, int>, int> w1 = {};  
  
 DPA(g1, w1, 0);  
  
 return 0;  
}