```cpp
#include <iostream>
#include <vector>
#include <limits>
#include <map>


int INF = std::numeric_limits<int>::max();


std::pair<std::vector<int>, int> near_in(const std::map<int,
std::vector<int>> &graph, const std::map<std::pair<int, int>, int> &wages) {
   int current = graph.cbegin()->first;
   std::vector<int> result = {current, current};
   std::vector<int> nodes = {};
   int min_distance;
   int neighbour;
   int total_cost = 0;
   std::vector<int> costs = {};

   nodes.reserve(graph.size());
   for (const auto &item : graph) {
           nodes.push_back(item.first);
   }
   nodes.erase(nodes.cbegin());

   while (!nodes.empty()) {
       // szukanie najmniejszej odlegolosci
       min_distance = INF;
       for (auto node: nodes) {
           if (wages.find({current, node})->second < min_distance) {
               min_distance = wages.find({current, node})->second;
           }
       }
       for (auto node: nodes) {
           if (wages.find({current, node})->second == min_distance) {
               neighbour = node;
               break;
           }
       }

       costs.reserve(result.size());
       for (int i = 1; i < result.size(); i++) {
           costs.push_back(wages.find({current, neighbour})->second);
       }

       int min_cost = INF;
       for (auto cost : costs)
           if (cost < min_cost)
               min_cost = cost;

       for(int i = 0; i < costs.size(); i++) {
           if (costs[i] == min_cost) {
               result.insert(std::next(result.cbegin(), i), neighbour);
```

```
                break;
            }
        }

        nodes.erase(std::find(nodes.cbegin(), nodes.cend(), current));
        total_cost += wages.find({current, neighbour})->second;
        current = neighbour;
    }
    total_cost += wages.find({neighbour, result[0]})->second;

    return {result, total_cost};
}


int main() {
    std::map<int, std::vector<int>> g1 = {
            {0, {1}},
            {1, {2}},
            {2, {3}},
            {3, {4}},
            {4, {0}}
    };

    std::map<std::pair<int, int>, int> w1 = {
            {{0, 1}, 3},
            {{1, 2}, 2},
            {{2, 3}, 4},
            {{3, 4}, 2},
            {{4, 0}, 1}
    };

    auto result = near_in(g1, w1);

    for (auto value : result.first) {
        printf("%d -> ", value);
    }
    printf("\nCost: %d\n", result.second);


    return 0;
}
```