Bartłomiej Barszczak
WEAIiIB Automatyka i Robotyka
Rok II semestr IV grupa 3

Zadanie 1

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

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


std::pair<std::vector<std::pair<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::vector<std::pair<int, int>> result = {}; // A;
    std::map<int, int> alfa = {}; // alfa
    std::map<int, int> beta = {}; // beta
    std::vector<int> queue = {}; // kolejka
    int new_vertex = 0; // u*
    int min_value = INF;


    for (const auto &item: graph) { // ustawianie warunkow poczatkowych
        alfa[item.first] = 0; // ustawianie wszystkich elementow w alfie na 0
        beta[item.first] = INF; // ustawienie wszystkich elementow w beta na 0
        queue.push_back(item.first); // dodanie do kolejki wszystkich wierzcholkow
    }

    beta[start] = 0;

    auto index = std::find(queue.cbegin(), queue.cend(), start); // usuniecie
poczatkowego wierzcholka z kolejki
    queue.erase(index);

    new_vertex = start; // wybranie nowego u*

    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()) { // warunek sprawdzajacy
czy u nalezy do sasiadow u*
                if (wages.find({u, new_vertex})->second < beta[u]) { // warunek
sprawdzajacy czy alfa od u i u* jest mniejsza niz beta od u
                    alfa[u] = new_vertex;
                    beta[u] = wages.find({u, new_vertex})->second;
                }
            }
        }

        min_value = INF; // ustawienie minimum na nieskonczonosc
        for (auto u: queue) {
```

```
            if (beta[u] < min_value) { // wybieranie nowej minimalnej wartosci w
beta
                min_value = beta[u];
            }
        }
        for (auto u: queue) {
            if (beta[u] == min_value) { // wybieranie wierzcholka ktorego wartosc w
beta jest minimalna
                new_vertex = u;
                break;
            }
        }

        index = std::find(queue.cbegin(), queue.cend(), new_vertex);
        queue.erase(index); // usuwanie nowego wierzcholka z kolejki
        result.emplace_back(alfa[new_vertex], new_vertex); // dodanie do wektora
wynikowego danego polaczenia
        sum += wages.find({alfa[new_vertex], new_vertex})->second; // dodanie do
sumy wagi krawedzi danej pray wierzcholkow
    }
    return {result, sum}; // zwracanie wynikow
}


int main() {
    std::map<int, std::vector<int>> g1 = { // tworzenie grafu
            {0, {1, 3}},
            {1, {0, 2}},
            {2, {1, 3, 5, 9}},
            {3, {0, 2, 4}},
            {4, {3, 5, 6}},
            {5, {2, 4, 8}},
            {6, {4, 7}},
            {7, {6, 8}},
            {8, {5, 7, 9}},
            {9, {2, 8}}
    };

    std::map<std::pair<int, int>, int> w1 = { // tworzenie wag grafu
            {{0, 1}, 2}, {{0, 3}, 1},
            {{1, 0}, 2}, {{1, 2}, 1},
            {{2, 1}, 1}, {{2, 3}, 4}, {{2, 5}, 2}, {{2, 9}, 8},
            {{3, 0}, 1}, {{3, 2}, 4}, {{3, 4}, 3},
            {{4, 3}, 3}, {{4, 5}, 1}, {{4, 6}, 2},
            {{5, 2}, 2}, {{5, 4}, 1}, {{5, 8}, 6},
            {{6, 4}, 2}, {{6, 7}, 5},
            {{7, 6}, 5}, {{7, 8}, 1},
            {{8, 5}, 6}, {{8, 7}, 1}, {{8, 9}, 3},
            {{9, 2}, 8}, {{9, 8}, 3}
    };

    auto result = DPA(g1, w1, 0); // wywolanie funckji DPA

    // prezentowanie danych
    for (const auto &item: result.first) {
        std::cout << item.first << " <-> " << item.second << "\n";
    }

    std::cout << "Sum: " << result.second << "\n";

    return 0;
}
```

Zadanie 2

Ważna jest spójność grafu oraz czy graf jest skierowany czy nie.

```
0 <-> 3
0 <-> 1
1 <-> 2
2 <-> 5
5 <-> 4
4 <-> 6
6 <-> 7
7 <-> 8
8 <-> 9
Sum: 18
```

Zrzut ekranu 1 Prezentacja wyników

Zadanie 3

Algorytm Kruskala polega na połączeniu wielu poddrzew w jedno za pomocą krawędzi o najmniejszej wadzie co w rezultacie skutkuje powstaniem drzewa, które będzie minimalne. Na początku należy posortować wszystkie krawędzie w porządku niemalejącym, następnie należy tworzyć drzewo.

Zadanie 4

Problemy rzeczywiste wyrażane za pomocą wag grafu mogą być np. przepustowością Internetu albo wody w rurociągach, lub w gdy określamy relacje pomiędzy ludźmi i chcemy znaleźć potencjalnych grup przyjaciół (jak bardzo dana osobę lubimy to waga wynosi 10, a jak nienawidzimy to waga wynosi 0). Żeby lepiej opisać problem możemy zamiast wag w postaci liczb całkowitych lub rzeczywistych użyć wielowymiarowych wektorów które lepiej, dokładniej opisują danych problem. Modyfikacja algorytmu jest konieczna ponieważ w niektórych sytuacjach może się okazać ze jedna osoba przyjaźni się z kilkoma osobami (z kilkoma osobami krawędź ma wagę 10) i zamiast wybierać losowo jedna z nich trzeba dodać obie lub więcej w przypadku gdy przyjaźni się z jeszcze większą ilością osób.