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

Zadanie 1

```cpp
#include <iostream>
#include <vector>
#include <map>
#include <set>
#include <cmath>


struct Node {
    int name;
    int time = -1;
    int ES = -1;
    int EF = -1;
    int LS = -1;
    int LF = -1;
    int slack = -1;
};

struct NodePlus {
    Node node;
    int lower = 0;
    int mean = 0;
    int upper = 0;
    double expected_time = -1;
    double variance = 0;
};

/**
 * Funkcja prezentuje uzyskane wyniki
 * @param cpm_path  Sciezka krytyczna
 * @param pert_result   Prawopodobienstwo
 */
void present_results(const std::vector<Node> &cpm_path, double pert_result) {
    std::cout << "--- Results ---\nCritical path: ";
    for (int i = 0; i < cpm_path.size(); i++) {
        if (i == cpm_path.size() - 1)
            std::cout << cpm_path[i].name << "\n";
        else
            std::cout << cpm_path[i].name << " -> ";
    }
    std::cout << "Possibility: " << pert_result << std::endl;
}

/**
 * Funkcja tworzy kontener Node i wyplenia go ustalona wartoscia
 * @param times     Kontener czasu przetwarzania
 * @return  Nowo utworzony kontener obiektow typu "Node"
 */
std::vector<Node> create_nodes(const std::vector<int> &times) {
    std::vector<Node> nodes = {};
    for (int i = 0; i < times.size(); i++) {
```

```
        Node node{};
        node.name = i;
        node.time = times[i];
        nodes.push_back(node);
    }
    return nodes;
}

/**
 * Funkcja towrzy kontener NodePlus i wyplenia go ustalonimi wartosciami
 * @param nodes     Kontener wezlow
 * @param lowers    Kontener dolnych granic
 * @param means     Kontener wartosci oczekiwanych
 * @param uppers    Kontener gornych granic
 * @return  Nowo utworony kontener obiektow typu "NodePlus"
 */
std::vector<NodePlus>
create_nodes_plus(const std::vector<Node> &nodes, const std::vector<int> &lowers,
const std::vector<int> &means,
                  const std::vector<int> &uppers) {
    std::vector<NodePlus> result;
    for (int i = 0; i < nodes.size(); i++) {
        NodePlus node_plus{};
        node_plus.node = nodes[i];
        node_plus.lower = lowers[i];
        node_plus.mean = means[i];
        node_plus.upper = uppers[i];
        result.push_back(node_plus);
    }
    return result;
}

/**
 * Funckja realizuje algorytm CPM
 * @param graph Graf w postaci listy sasiedztwa
 * @param nodes Kontener obiektow typu Node
 * @return Nowo utworzony kontener zawierajacy sciezke krytyczna
 */
std::vector<Node> CPM(const std::map<int, std::vector<int>> &graph,
std::vector<Node> &nodes) {
    std::map<int, std::vector<int>> predecessors = {};
    std::set<int> set1 = {}, set2 = {}, starts = {};
    std::vector<int> helper = {};
    std::vector<Node> result = {};
    int total_time;

    // wyznaczanie poprzednikow
    for (int i = (int) nodes.size() - 1; i >= 0; i--) {
        helper.clear();
        for (auto item: graph) {
            if (std::find(item.second.begin(), item.second.end(), i) !=
item.second.cend()) {
                helper.push_back(item.first);
            }
        }
        predecessors[i] = helper;
    }

    // wyznaczanie poczatkowych wezlow
    for (auto &item: graph) {
        set1.insert(item.first);
        for (auto value: item.second) {
            set2.insert(value);
```

```
        }
    }
    std::set_difference(set1.cbegin(), set1.cend(), set2.cbegin(), set2.cend(),
std::inserter(starts, starts.cend()));

    // ustawianie na poczatkowych wezlach wartosci najwczesniejszego staru i konca
    for (auto item: starts) {
        nodes[item].ES = 0;
        nodes[item].EF = nodes[item].ES + nodes[item].time;
    }

    // ustawianie na wezlach wartosci najwczesniejszego staru i konca
    for (int i = 1; i < nodes.size(); i++) {
        int index = *std::max_element(predecessors[i].begin(),
predecessors[i].end(),
                                      [&nodes](auto a, auto b) {
                                          if (nodes[a].EF < nodes[b].EF)
                                              return true;
                                          else return false;
                                      });
        nodes[i].ES = nodes[index].EF;
        nodes[i].EF = nodes[i].ES + nodes[i].time;
    }

    // tworzenie kontenera zawierajacego liscie grafu
    helper.clear();
    for (auto &item: graph) {
        if (item.second.empty()) {
            helper.push_back(item.first);
        }
    }
    // ustawienie calkowitego czasu
    total_time = nodes[*std::max_element(helper.cbegin(), helper.cend(),
[&nodes](int a, int b) {
        if (nodes[a].EF < nodes[b].EF)
            return true;
        else return false;
    })].EF;

    // przypisane calkowitego czasu do najpozniejszych poczatkow i koncow lisci
grafu oraz opoznienie
    for (auto item: helper) {
        nodes[item].LF = total_time;
        nodes[item].LS = nodes[item].LF - nodes[item].time;
        nodes[item].slack = nodes[item].LF - nodes[item].EF;
    }

    // przypisane wartosci do najpozniejszych poczatkow i koncow dla wezla oraz
jego opoznienie
    for (int i = (int) (nodes.size() - helper.size() - 1); i >= 0; i--) {
        int index = *std::min_element(graph.at(i).cbegin(), graph.at(i).cend(),
                                      [&nodes](int a, int b) {
                                          if (nodes[a].LS < nodes[b].LS)
                                              return true;
                                          else return false;
                                      });
        nodes[i].LF = nodes[index].LS;
        nodes[i].LS = nodes[i].LF - nodes[i].time;
        nodes[i].slack = nodes[i].LF - nodes[i].EF;
    }

    // tworzenie listy wynikowej, sciezki krytycznej
    for (auto &item: nodes) {
```

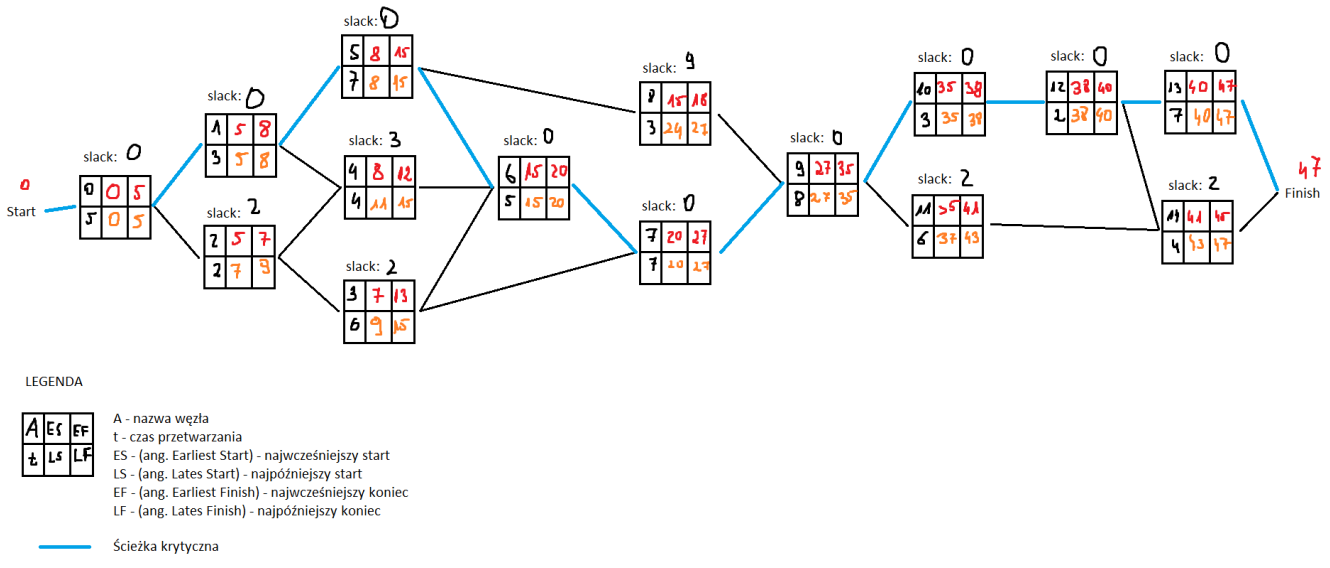
```
        if (item.slack == 0)
            result.push_back(item);
    }

    return result;
}

/**
 * Funckja wylicza prawdopodobienstwo metoda PERT
 * @param nodes     Kontener obiektów typu NodePlus
 * @param inv_p     Prawdopodobienstwo (odczytnana wartość)
 * @return Termin realizacji zgodny z prawodpodobienstwem
 */
double PERT(std::vector<NodePlus> &nodes, double inv_p) {
    std::vector<NodePlus *> critical_path = {};
    double total_variance = 0;
    double sigma;
    double mean = std::max_element(nodes.cbegin(), nodes.cend(), [](NodePlus a,
NodePlus b) {
        if (a.node.EF > b.node.EF)
            return false;
        else return true;})->node.EF;
    double result;

    for (auto &item: nodes) {
        if (item.node.slack == 0)
            critical_path.push_back(&item);
        item.expected_time = (double) (item.lower + 4 * item.mean + item.upper) /
6;
    }

    for (auto &item: critical_path) {
        item->variance = ((double) (item->upper - item->lower) / 6) * ((double)
(item->upper - item->lower) / 6);
        total_variance += item->variance;
    }
    sigma = sqrt(total_variance);
    result = mean + inv_p * sigma;
    return result;
}


int main() {
    // graf
    std::map<int, std::vector<int>> g1 = {
            {0,  {1, 2}},
            {1,  {4, 5}},
            {2,  {3, 4}},
            {3,  {6, 7}},
            {4,  {6}},
            {5,  {6, 8}},
            {6,  {7}},
            {7,  {9}},
            {8,  {9}},
            {9,  {10, 11}},
            {10, {12}},
            {11, {14}},
            {12, {13, 14}},
            {13, {}},
            {14, {}}
    };

    std::vector<int> t1 = {5, 3, 2, 6, 4, 7, 5, 7, 3, 8, 3, 6, 2, 7, 4}; // czasy
```

```
przetwarzania zadania
    std::vector<int> a1 = {3, 1, 2, 2, 3, 5, 2, 6, 2, 4, 1, 5, 2, 5, 1}; // dolne
granice
    std::vector<int> b1 = {6, 6, 3, 10, 7, 9, 6, 13, 5, 9, 5, 7, 3, 9, 9}; // gorne
granice
    std::vector<int> m1 = {4, 2, 2, 6, 4, 7, 4, 9, 4, 7, 3, 5, 2, 8, 6}; //
wartosci oczekiwane

    auto n1 = create_nodes(t1);
    auto result1 = CPM(g1, n1);

    auto np1 = create_nodes_plus(n1, a1, m1, b1);
    double result2 = PERT(np1, 1.28);

    present_results(result1, result2);

    return 0;
}
```

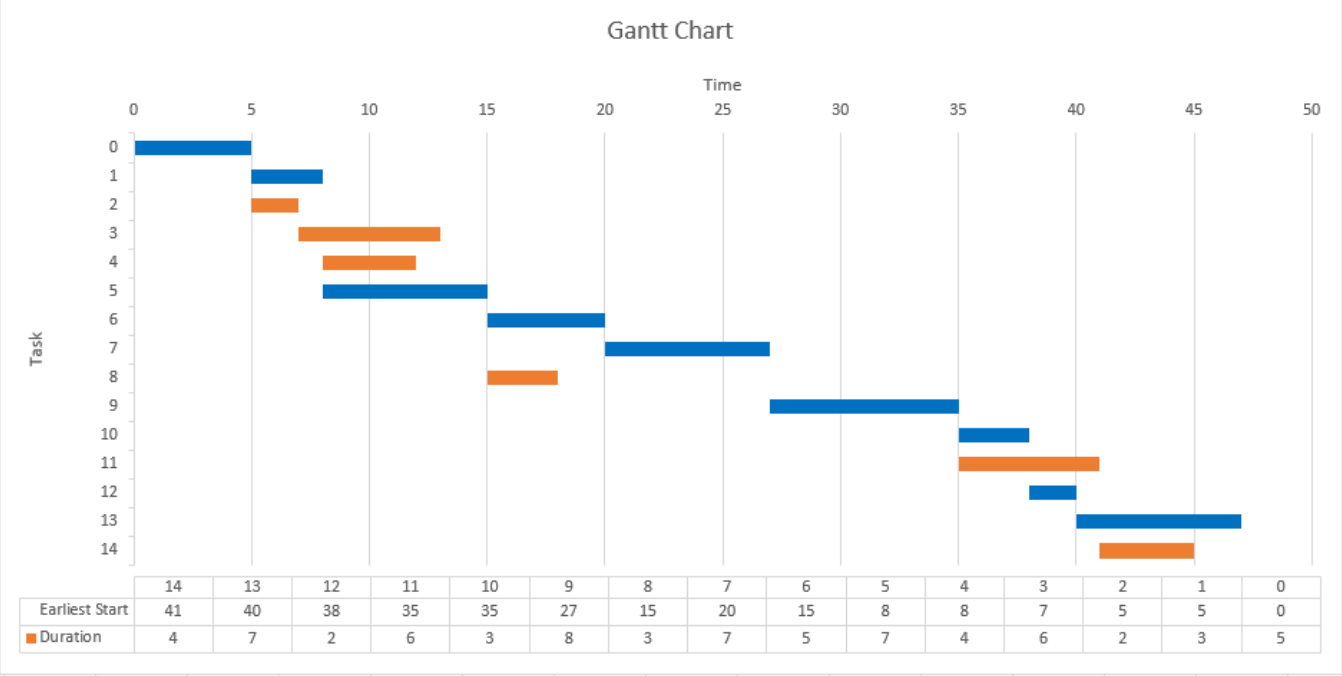
Zadanie 2

```
--- Results ---
Critical path: 0 -> 1 -> 5 -> 6 -> 7 -> 9 -> 10 -> 12 -> 13
Possibility: 49.806
```

Rysunek 1 Prezentacja wyników



Rysunek 2 Rysunek poglądowy przedstawiający rozwiązanie metodą CPM



	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Earliest Start	41	40	38	35	35	27	15	20	15	8	8	7	5	5	0
Duration	4	7	2	6	3	8	3	7	5	7	4	6	2	3	5

Rysunek 3 Wykres Gantta wykonany w programie Microsoft Excel

Zadanie 3

Ścieżka krytyczna na wykresie Gantt'a jest zaznaczona kolorem niebieskim, rezerwa interpretowana jest jako kolor pomarańczowy.