AKADEMIA GÓRNICZO-HUTNICZA im. Stanisława Staszica w Krakowie

Programowanie sieciowe – algorytmy CPM, PERT

Stanisław Olech - 412023

Automatyka i Robotyka

EAIiIB

Zad. 1

Kod. 1 Zaimplementowany przez mnie algorytmu

```
#include <iostream>
#include <limits>
#include <map>
#include <list>
#include <algorithm>
#include <cmath>

int inf = std::numeric_limits<int>::max();
float float_inf = std::numeric_limits<float>::max();

struct path{
    int origin;
    int destination;
};

struct re_num{
    int org_num;
    int new_num;
};

template <size_t size>
void array(int(&tc)[size][size], int(&tm)[size][size],
int(&tp)[size][size], float(&tx)[size][size]){
    for (size_t x = 0; x < size; x++) {
        for (size_t y = 0; y < size; y++) {
            if (tc[x][y] != inf){
                tx[x][y] = 1.29 * pow(pow(tp[x][y] - tc[x][y], 2) / 36,
0.5) + (float(tc[x][y]) / 6 + 4.0 / 6 * float(tm[x][y]) + float(tp[x][y]) /
6);
            } else{
                tx[x][y] = float_inf;
            }
        }
    }
}

template <size_t size>
std::tuple<std::list<path>, float> PERT(float(&tc)[size][size]){

    // Deklaracja zmiennych
    std::list<path> path = {};
    std::list<re_num> new_name = {};
    std::list<size_t> nodes = {};
    float tw_x[size];
    float tp_x[size];
    float ans;



    // Przenumerowanie
    int i = 0;
    while (nodes.size() != size) {
        for (size_t x = 0; x < size; x++) {
            if(std::find(nodes.begin(), nodes.end(), x) != nodes.end()){
                continue;
            }
```

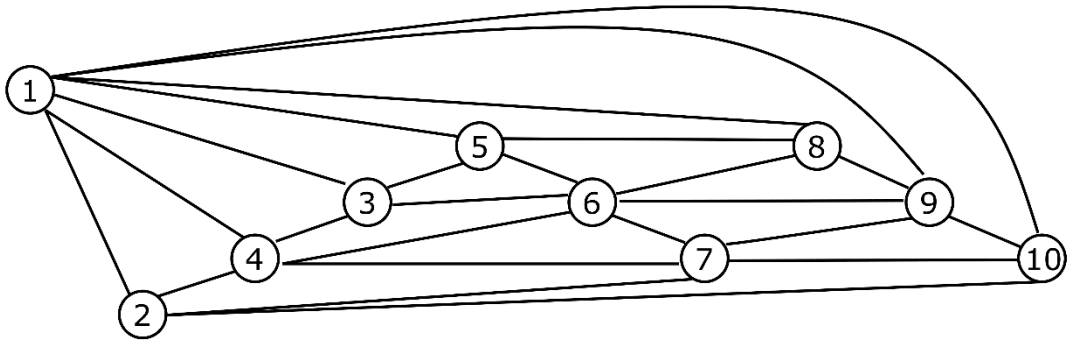
```
            bool flag = true;


            for (size_t y = 0; y < size; y++) {
                if(std::find(nodes.begin(), nodes.end(), y) !=
nodes.end()){
                    continue;
                }
                if (tc[y][x] != float_inf and y != x) {
                    flag = false;
                }
            }

            if (flag) {
                new_name.push_back({static_cast<int>(x), i});
                nodes.push_back(x);
                i++;
            }
        }
    }

    for(auto& x :new_name){
        float sum = 0;
        for (size_t y = 0; y < size; y++) {
            if (tc[y][x.org_num] != float_inf and tc[y][x.org_num] +
tw_x[y] > sum){
                sum = tc[y][x.org_num] + tw_x[y];
            }
        }
        tw_x[x.org_num] = sum;
        tp_x[x.org_num] = sum;
    }

    for(auto x = new_name.rbegin(); x != new_name.rend(); x++){
        re_num ele = x.operator*();
        if (ele.new_num == size -1){continue;}

        float sum = float_inf;
        for (size_t y = 0; y < size; y++) {
            if (tc[ele.org_num][y] != float_inf and tp_x[y] -
tc[ele.org_num][y] < sum){
                sum = tp_x[y] - tc[ele.org_num][y];
            }
        }
        tp_x[ele.org_num] = sum;
    }

    int last = 0;
    bool flag = false;
    for(auto& x :new_name){
        if (!flag){
            last = x.org_num;
            flag = true;
            continue;
        }
        if(std::abs(tw_x[x.org_num] - tp_x[x.org_num]) < 0.0001){
            path.push_back({last, x.org_num});
            last = x.org_num;
        }
        ans = tw_x[x.org_num];
    }
```

```
    return {path, ans};

}

template <size_t size>
std::tuple<std::list<path>, float> PERT(int(&tc)[size][size]){
    float tx[size][size];

    for (size_t x = 0; x < size; x++) {
        for (size_t y = 0; y < size; y++) {
            if (tc[x][y] != inf){
                tx[x][y] = float(tc[x][y]);
            } else{
                tx[x][y] = float_inf;
            }
        }
    }
    return (PERT(tx));
}

template <size_t size, typename type>
std::list<size_t> order(type(&tx)[size][size], const std::list<path>&
cri_path){

    std::list<size_t> nodes = {};

    for(auto& ele: cri_path){
        nodes.push_back(ele.origin);
        nodes.push_back(std::numeric_limits<size_t>::max());
        for (size_t x = 0; x < size; x++) {
            if (std::find(nodes.begin(), nodes.end(), x) != nodes.end() or
                x == ele.destination) {
                continue;
            }

            bool flag = true;


            for (size_t y = 0; y < size; y++) {
                if (std::find(nodes.begin(), nodes.end(), y) !=
nodes.end()) {
                    continue;
                }
                if (tx[y][x] != float_inf and y != x) {
                    flag = false;
                }
            }
            if (flag) {
                nodes.push_back(x);
            }
        }
    }
    for (int x = 0; x != size; x++){
        if (std::find(nodes.begin(), nodes.end(), x) == nodes.end()) {
            nodes.push_back(x);
        }
    }

    return nodes;
}
```

```
int main() {
    int tc[10][10] = {
            {inf,2,  4,  1,   3,   inf, inf, 3,   7,   3  },
            {inf,inf,3,  1,   inf, inf, 5,   inf, inf, 2  },
            {inf,inf,inf,inf, 1,   2,   inf, inf, inf, inf},
            {inf,inf,1,  inf, inf, 4,   4,   inf, inf, inf},
            {inf,inf,inf,inf, inf, 3,   inf, 5,   inf, inf},
            {inf,inf,inf,inf, inf, inf, 1,   4,   4,   inf},
            {inf,inf,inf,inf, inf, inf, inf, inf, 2,   1  },
            {inf,inf,inf,inf, inf, inf, inf, inf, 1,   inf},
            {inf,inf,inf,inf, inf, inf, inf, inf, inf, 3  },
            {inf,inf,inf,inf, inf, inf, inf, inf, inf, inf},
    };

    int tm[10][10] = {
            {inf,3,  6,  1,   8,   inf, inf, 5,   8,   8  },
            {inf,inf,3,  2,   inf, inf, 5,   inf, inf, 2  },
            {inf,inf,inf,inf, 3,   2,   inf, inf, inf, inf},
            {inf,inf,1,  inf, inf, 4,   4,   inf, inf, inf},
            {inf,inf,inf,inf, inf, 7,   inf, 5,   inf, inf},
            {inf,inf,inf,inf, inf, inf, 4,   6,   4,   inf},
            {inf,inf,inf,inf, inf, inf, inf, inf, 2,   3  },
            {inf,inf,inf,inf, inf, inf, inf, inf, 3,   inf},
            {inf,inf,inf,inf, inf, inf, inf, inf, inf, 3  },
            {inf,inf,inf,inf, inf, inf, inf, inf, inf, inf},
    };

    int tp[10][10] = {
            {inf,9,  6,  15,  8,   inf, inf, 7,   10,  10 },
            {inf,inf,5,  3,   inf, inf, 5,   inf, inf, 3  },
            {inf,inf,inf,inf, 5,   2,   inf, inf, inf, inf},
            {inf,inf,2,  inf, inf, 7,   4,   inf, inf, inf},
            {inf,inf,inf,inf, inf, 7,   inf, 5,   inf, inf},
            {inf,inf,inf,inf, inf, inf, 4,   6,   5,   inf},
            {inf,inf,inf,inf, inf, inf, inf, inf, 7,   4  },
            {inf,inf,inf,inf, inf, inf, inf, inf, 3,   inf},
            {inf,inf,inf,inf, inf, inf, inf, inf, inf, 7  },
            {inf,inf,inf,inf, inf, inf, inf, inf, inf, inf},
    };

    float tx[10][10];
    array(tc, tm, tp, tx);

    auto ans1 = PERT(tc);
    std::cout << "minimalna dlugosc: "<< std::get<1>(ans1) << std::endl;

    for(auto ele : std::get<0>(ans1)){
        std::cout << ele.origin + 1 << " -> " << ele.destination + 1 <<
std::endl;
    }
    std::cout << std::endl;


    auto ans2 = PERT(tm);
    std::cout << "przewidywana dlugosc: "<< std::get<1>(ans2) << std::endl;

    for(auto ele : std::get<0>(ans2)){
        std::cout << ele.origin + 1 << " -> " << ele.destination + 1 <<
std::endl;
    }
```

```
    std::cout << std::endl;


    auto ans3 = PERT(tx);
    std::cout << "z prawdopodobienstwem 90% nie przekroczy: "<<
std::get<1>(ans3) << std::endl;

    for(auto ele : std::get<0>(ans3)){
        std::cout << ele.origin + 1 << " -> " << ele.destination + 1 <<
std::endl;
    }
    std::cout << std::endl;

    auto ansprim = order(tx, std::get<0>(ans3));
    for(auto ele : ansprim){
        if (ele + 1 == 0){
            std::cout << "---------" << std::endl;
            continue;
        }
        std::cout << ele + 1 << std::endl;
    }
    std::cout << std::endl;



    auto ans4 = PERT(tp);
    std::cout << "maksymalna dlugosc: "<< std::get<1>(ans4) << std::endl;

    for(auto ele : std::get<0>(ans4)){
        std::cout << ele.origin + 1 << " -> " << ele.destination + 1 <<
std::endl;
    }
}
```

kod źródłowy metody PERT.



Rys. 1 Graf z 10 wierzchołkami i 20 krawędziami. Oczywiście wbrew temu co rysunek pokazuje jest to graf skierowany i możemy się poruszać jedynie w prawo.

Jako reprezentację przyjąłem tabelę z wartościami optymalnymi, przewidywanymi i pesymistycznymi.

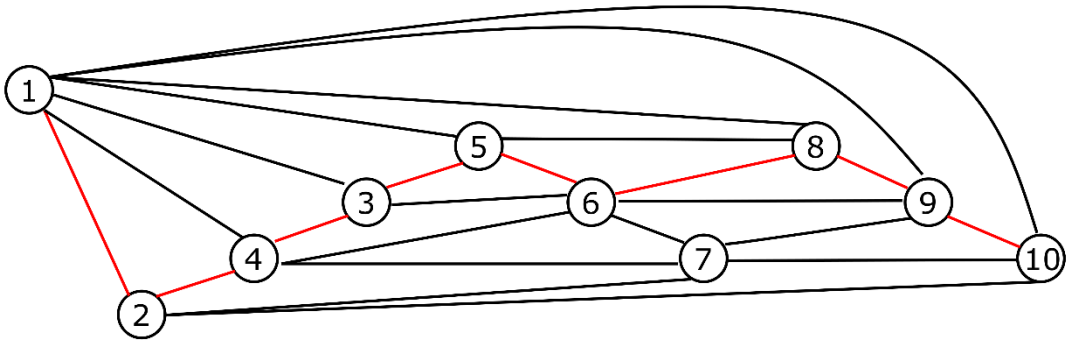
```
int tc[10][10] = {
        [0]: { [0]: inf, [1]: 2,   [2]: 4,   [3]: 1,   [4]: 3,   [5]: inf, [6]: inf, [7]: 3,   [8]: 7,   [9]: 3  },
        [1]: { [0]: inf, [1]: inf, [2]: 3,   [3]: 1,   [4]: inf, [5]: inf, [6]: 5,   [7]: inf, [8]: inf, [9]: 2  },
        [2]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: 1,   [5]: 2,   [6]: inf, [7]: inf, [8]: inf, [9]: inf},
        [3]: { [0]: inf, [1]: inf, [2]: 1,   [3]: inf, [4]: inf, [5]: 4,   [6]: 4,   [7]: inf, [8]: inf, [9]: inf},
        [4]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: 3,   [6]: inf, [7]: 5,   [8]: inf, [9]: inf},
        [5]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: 1,   [7]: 4,   [8]: 4,   [9]: inf},
        [6]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: 2,   [9]: 1  },
        [7]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: 1,   [9]: inf},
        [8]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: inf, [9]: 3  },
        [9]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: inf, [9]: inf},
};

int tm[10][10] = {
        [0]: { [0]: inf, [1]: 3,   [2]: 6,   [3]: 1,   [4]: 8,   [5]: inf, [6]: inf, [7]: 5,   [8]: 8,   [9]: 8  },
        [1]: { [0]: inf, [1]: inf, [2]: 3,   [3]: 2,   [4]: inf, [5]: inf, [6]: 5,   [7]: inf, [8]: inf, [9]: 2  },
        [2]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: 3,   [5]: 2,   [6]: inf, [7]: inf, [8]: inf, [9]: inf},
        [3]: { [0]: inf, [1]: inf, [2]: 1,   [3]: inf, [4]: inf, [5]: 4,   [6]: 4,   [7]: inf, [8]: inf, [9]: inf},
        [4]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: 7,   [6]: inf, [7]: 5,   [8]: inf, [9]: inf},
        [5]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: 4,   [7]: 6,   [8]: 4,   [9]: inf},
        [6]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: 2,   [9]: 3  },
        [7]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: 3,   [9]: inf},
        [8]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: inf, [9]: 3  },
        [9]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: inf, [9]: inf},
};
int tp[10][10] = {
        [0]: { [0]: inf, [1]: 9,   [2]: 6,   [3]: 15,  [4]: 8,   [5]: inf, [6]: inf, [7]: 7,   [8]: 10,  [9]: 10 },
        [1]: { [0]: inf, [1]: inf, [2]: 5,   [3]: 3,   [4]: inf, [5]: inf, [6]: 5,   [7]: inf, [8]: inf, [9]: 3  },
        [2]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: 5,   [5]: 2,   [6]: inf, [7]: inf, [8]: inf, [9]: inf},
        [3]: { [0]: inf, [1]: inf, [2]: 2,   [3]: inf, [4]: inf, [5]: 7,   [6]: 4,   [7]: inf, [8]: inf, [9]: inf},
        [4]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: 7,   [6]: inf, [7]: 5,   [8]: inf, [9]: inf},
        [5]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: 4,   [7]: 6,   [8]: 5,   [9]: inf},
        [6]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: 7,   [9]: 4  },
        [7]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: 3,   [9]: inf},
        [8]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: inf, [9]: 7  },
        [9]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: inf, [7]: inf, [8]: inf, [9]: inf},
};
```

Rys. 2 Reprezentacja.

```
minimalna dlugosc: 17
1 -> 2
2 -> 3
3 -> 5
5 -> 6
6 -> 8
8 -> 9
9 -> 10

przewidywana dlugosc: 28
1 -> 2
2 -> 4
4 -> 3
3 -> 5
5 -> 6
6 -> 8
8 -> 9
9 -> 10
```

```
z prawdopodobienstwem 90% nie przekroczy: 33.9233
1 -> 2
2 -> 4
4 -> 3
3 -> 5
5 -> 6
6 -> 8
8 -> 9
9 -> 10

maksymalna dlugosc: 47
1 -> 4
4 -> 3
3 -> 5
5 -> 6
6 -> 7
7 -> 9
9 -> 10
```

Rys. 3 Działanie algorytmu.

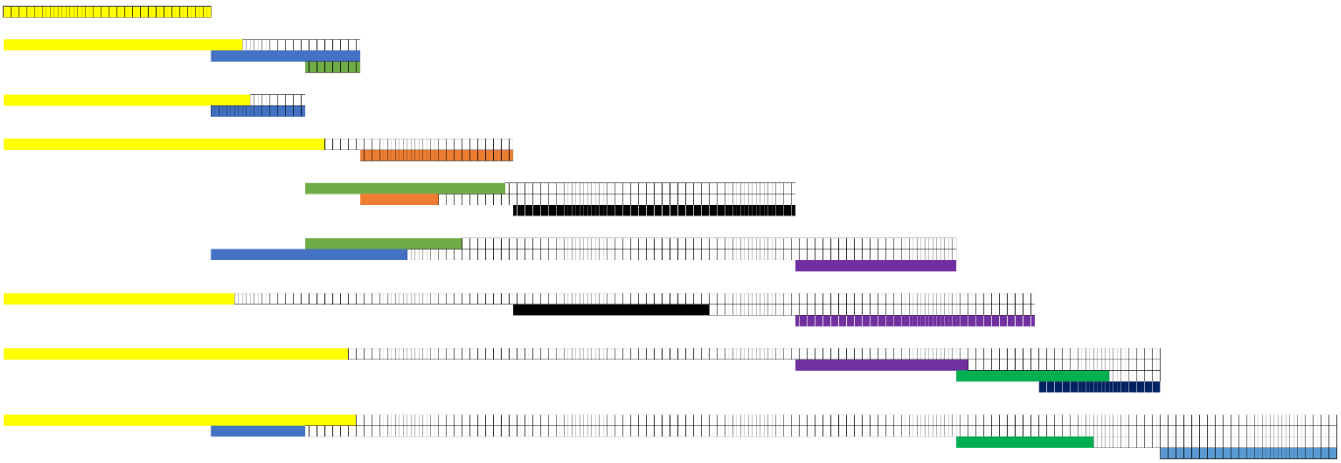


Rys. 4 Ścieżka krytyczna dla czasu liczonego z 90% pewnością

Zad. 2

```
1
---------
2
---------
4
---------
3
---------
5
---------
6
---------
7
8
---------
9
---------
10
```

Rys. 5 Kolejność wykonywania działań



Rys. 5 wykres Gantt'a. Kolorowe zakreskowane to ścieżki krytyczne a zakreskowane białe to zapas

Wnioski

Zadanie sprawiło mi trudnośći, w reprezentacji wykresu Gantt'a. oraz w samej implementacji .Jednak nauczyłem się podczas niego dużo o szablonach klas w c++ oraz różnych metodach zwracania tablicę przez funkcję.