

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2
по курсу «Алгоритмы и структуры данных»

Тема: Графы

Вариант 3

Выполнил:

Бунос М.В.

К3141

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №3. Циклы [5 s, 512 Mb, 1 балл]	3
Задача №8. Стоимость полета [10 s, 512 Mb, 1.5 балла]	7
Задача №14. Автобусы [1 s, 16 Mb, 3 балла]	10
Дополнительные задачи	14
Задача №1. Лабиринт [5 s, 512 Mb, 1 балл]	14
Задача №2. Компоненты [5 s, 512 Mb, 1 балл]	19
Задача №5. Город с односторонним движением [5 s, 512 Mb, 1.5 балла]	22
Задача №6. Количество пересадок [10 s, 512 Mb, 1 балл]	26
Задача №7. Двудольный граф [10 s, 512 Mb, 1.5 балла]	30
Задача №10. Оптимальный обмен валюты [10 s, 512 Mb, 2 балла]	34
Задача №18. Построение дорог [10 s, 512 Mb, 4 балла]	38
Вывод	42

Задачи по варианту

Задача №3. Циклы [5 s, 512 Mb, 1 балл]

Учебная программа по инфокоммуникационным технологиям определяет пререквизиты для каждого курса в виде списка курсов, которые необходимо пройти перед тем, как начать этот курс. Вы хотите выполнить проверку согласованности учебного плана, то есть проверить отсутствие циклических зависимостей. Для этого строится следующий ориентированный граф: вершины соответствуют курсам, есть направленное ребро (u, v) – курс u следует пройти перед курсом v . Затем достаточно проверить, содержит ли полученный граф цикл. Проверьте, содержит ли данный граф циклы.

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}

void getFirstTime() {
```

```

        start = clock();
    }

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
    " sec\n";
}

//-----

vector<vector<int>> g;
vector<int> color;
int cycle_start;

bool dfs(int v) {
    color[v] = 1;
    for (int u : g[v]) {
        if (color[u] == 0) {
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

int main() {
    getFirstTime();

    // ---- code starts here ----

    int n, m;
    fin >> n >> m;
    g.assign(n, vector<int> ());
    cycle_start = -1;

    for(int i = 0; i < m; ++ i) {
        int a, b;
        fin >> a >> b;

        g[a - 1].push_back(b - 1);
    }

    color.assign(n, 0);

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        fout << "0";
    } else {
        fout << "1";
    }

    // ---- code ends here ----

```

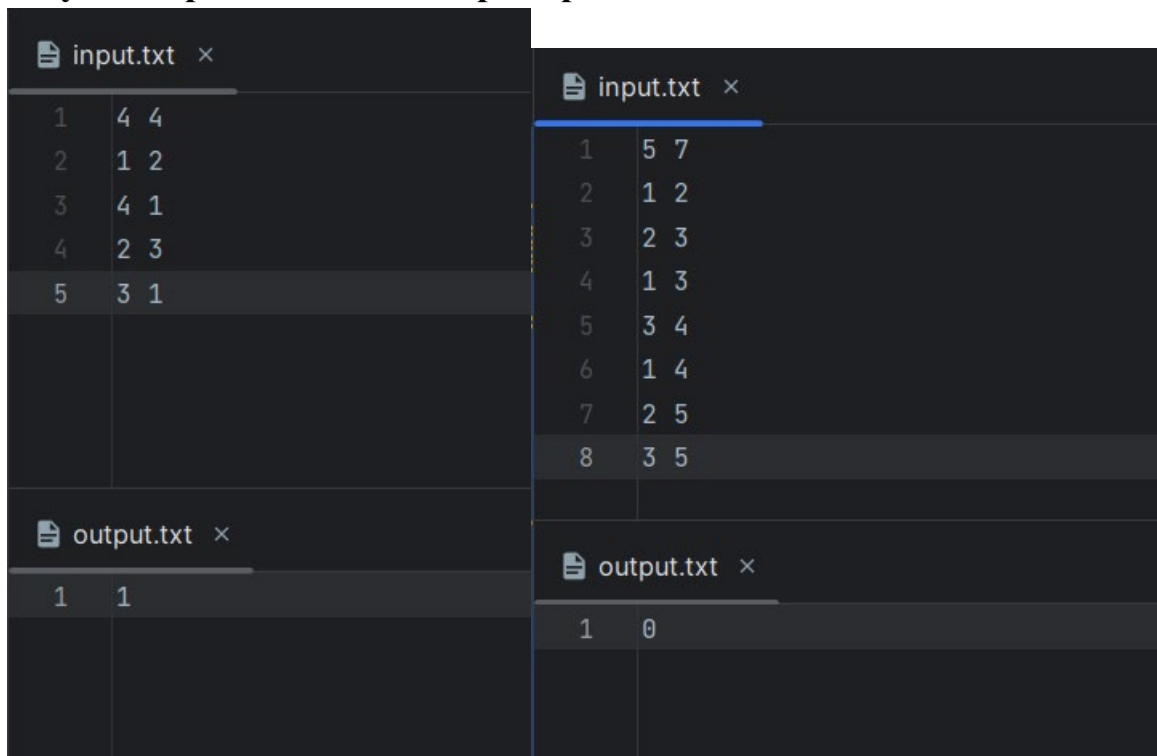
```
printTimeUse();
printMemoryUse();

fin.close();
fout.close();
return 0;
}
```

Текстовое объяснение решения:

Мы запустим серию DFS на графе. Изначально все вершины окрашены в белый цвет (0). Из каждой не посещенной (белой) вершины запустим DFS, отметим ее серым (1) при входе и черным (2) при выходе. Если DFS перемещается в серую вершину, значит, мы нашли цикл.

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.812500 MB
Пример из задачи	0.000000 sec	0.792969 MB

Пример из задачи	0.000000 sec	0.812500 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.804688 MB

Вывод по задаче:

Задача просто на нахождение циклов.

Задача №8. Стоимость полета [10 s, 512 Mb, 1.5 балла]

Теперь вас интересует минимизация не количества пересадок, а общей стоимости полета. Для этого строится взвешенный граф: вес ребра из одного города в другой – это стоимость соответствующего перелета.

Дан ориентированный граф с положительными весами ребер, n - количество вершин и m - количество ребер, а

также даны две вершины u и v . Вычислить вес кратчайшего пути между u и v (то есть минимальный общий вес пути из u в v).

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
```

```

" sec\n";
}

//-----
vector<vector<pair <int, int> >> g;

int main() {
    getFirstTime();

    // ---- code starts here -----

    int n, m;
    fin >> n >> m;
    g.assign(n, vector <pair <int, int> > ());

    for(int i = 0; i < m; ++ i) {
        int a, b, c;
        fin >> a >> b >> c;

        g[a - 1].emplace_back(b - 1, c);
    }

    int u, v;
    fin >> u >> v;

    vector<int> d, p;
    int s = u - 1;
    d.assign(n, INT_MAX);
    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin()->second;
        q.erase(q.begin());

        for (auto edge : g[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }

    fout << (d[v - 1] == INT_MAX ? -1 : d[v - 1]);

    // ---- code ends here -----

    printTimeUse();
    printMemoryUse();

    fin.close();
    fout.close();
}

```



```

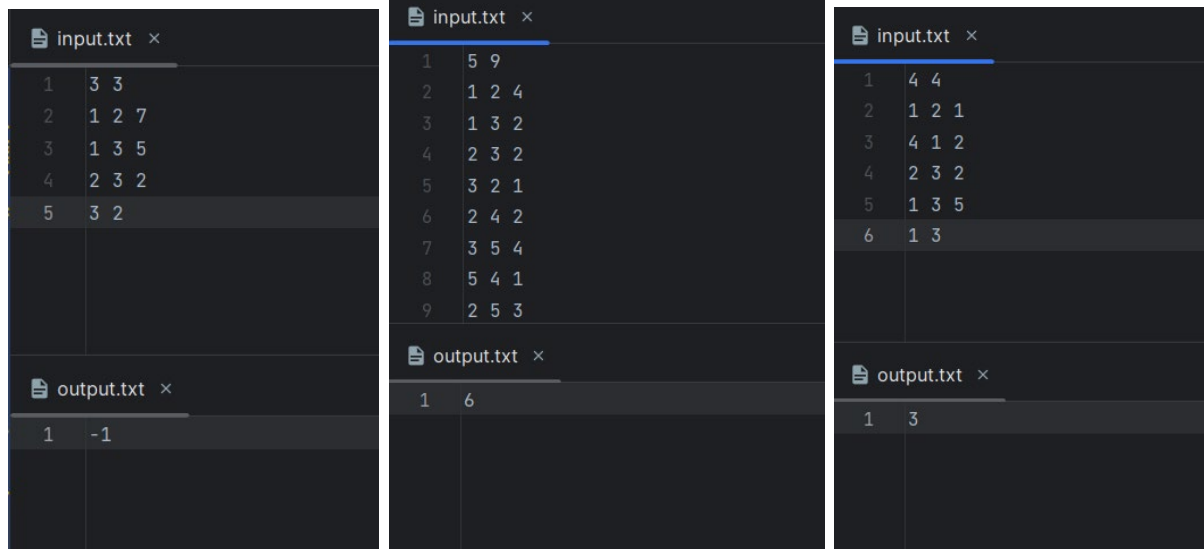
    return 0;
}

```

Текстовое объяснение решения:

Просто запустим алгоритм Дейкстры и получим ответ.

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.812500 MB
Пример из задачи	0.000000 sec	0.792969 MB
Пример из задачи	0.000000 sec	0.800781 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB

Вывод по задаче:

Обычная задача на алгоритм Дейкстры.

Задача №14. Автобусы [1 s, 16 Mb, 3 балла]

Между некоторыми деревнями края Власюки ходят автобусы. Поскольку пассажиропотоки здесь не очень большие, то автобусы ходят всего несколько раз в день.

Марии Ивановне требуется добраться из деревни d в деревню v как можно быстрее (считается, что в момент времени 0 она находится в деревне d).

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <queue>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
    " sec\n";
}
```

```

//-----
vector<vector<pair <int, int> >> g;

int main() {
    getFirstTime();

    // ---- code starts here -----

    int n, s, f, m;
    fin >> n >> s >> f >> m;

    struct Info {
        int a, b, c, d;
    };

    vector <Info> g(m);

    for (int i = 0; i < m; i++) {
        int s, t, e, l;
        fin >> s >> t >> e >> l;
        g[i] = {s, t, e, l};
    }

    vector <int> d (n, INT_MAX);

    d[s - 1] = 0;
    for(int i = 0; i < n - 1; ++ i)
        for(int j = 0; j < m; ++ j)
            if(d[g[j].c - 1] > g[j].d && d[g[j].a - 1] != INT_MAX && g[j].b
>= d[g[j].a - 1])
                d[g[j].c - 1] = g[j].d;

    fout << (d[f - 1] == INT_MAX ? -1 : d[f - 1]);

    // ---- code ends here -----

    printTimeUse();
    printMemoryUse();

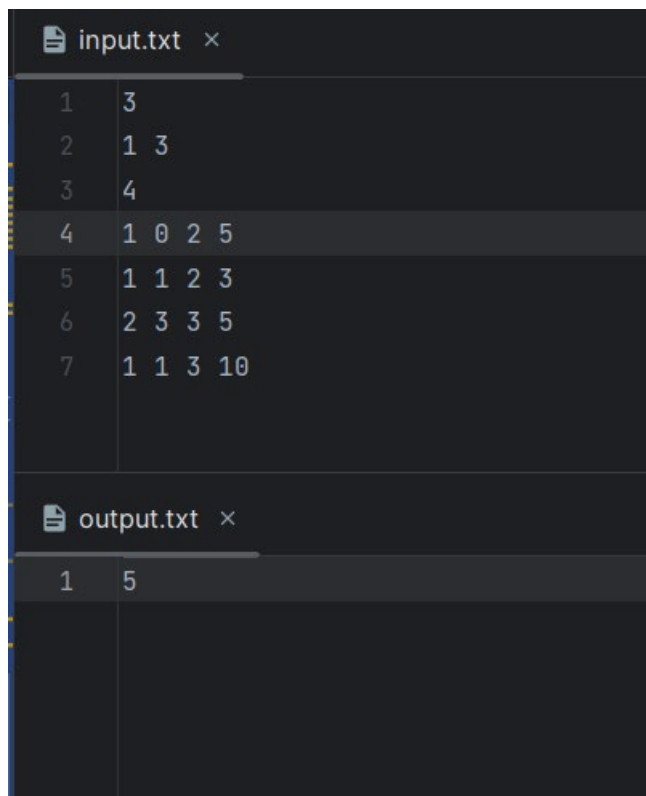
    fin.close();
    fout.close();
    return 0;
}

```

Текстовое объяснение решения:

Просто запустим алгоритм Беллмана-Форда и получим ответ.

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, астр и тд при наличии в задаче):

```

11 //-----
12 #include <ext/pb_ds/assoc_container.hpp>
13 #include <ext/pb_ds/tree_policy.hpp>
14 #include <queue>
15
16 using namespace std;
17 using namespace __gnu_pbds;
18
19 //-----
20 vector<vector<pair<int, int>>> g;
21
22 int main() {
23     int n, s, f, m;
24     cin >> n >> s >> f >> m;
25
26     struct Info {
27         int a, b, c, d;
28     };
29     vector<Info> g(m);
30
31     for (int i = 0; i < m; i++) {
32         int s, t, e, l;
33         cin >> s >> t >> e >> l;
34         g[i] = {s, t, e, l};
35     }
36
37     vector<int> d (n, INT_MAX);
38
39     d[s - 1] = 0;
40     for (int i = 0; i < n - 1; ++ i)
41         for (int j = 0; j < m; ++ j)
42             if (d[g[j].c - 1] > g[j].d && d[g[j].a - 1] != INT_MAX && g[j].b >= d[g[j].a - 1])
43                 d[g[j].c - 1] = g[j].d;
44
45     cout << (d[f - 1] == INT_MAX ? -1 : d[f - 1]);
46     return 0;
47 }
48

```

ID	Дата	Язык	Результат	Тест	Время	Память
18962902	06.03.2023 19:32:44	C++	Accepted		0,124	576 Kб

Размер кода: 717

Посылки решений:

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.812500 MB

Пример из задачи	0.000000 sec	0.808594 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.092000 sec	0.812500 MB

Вывод по задаче:

Аналогично простая задачка

Дополнительные задачи

Задача №1. Лабиринт [5 s, 512 Mb, 1 балл]

Лабиринт представляет собой прямоугольную сетку ячеек со стенками между некоторыми соседними ячейками.

Вы хотите проверить, существует ли путь от данной ячейки к данному выходу из лабиринта, где выходом также является ячейка, лежащая на границе лабиринта (в примере, показанном на рисунке, есть два выхода: один на левой границе и один на правой границе). Для этого вы представляете лабиринт в виде неориентированного графа: вершины графа являются ячейками лабиринта, две вершины соединены неориентированным ребром, если они смежные и между ними нет стены. Тогда, чтобы проверить, существует ли путь между двумя заданными ячейками лабиринта, достаточно проверить, что существует путь между соответствующими двумя вершинами в графе.

Вам дан неориентированный граф и две различные вершины u и v . Проверьте, есть ли путь между u и v .

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <queue>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
```

```

PROCESS_MEMORY_COUNTERS_EX pmc;
GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
&pmc, sizeof(pmc));
SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

cerr << fixed << setprecision(6);
cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
<< " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
" sec\n";
}

//-----
vector<vector<int>> g;
vector<bool> visited;

void dfs(int v, int p = -1) {
    if(visited[v])
        return;

    visited[v] = true;
    for(int u : g[v]) {
        if(u == p)
            continue;

        dfs(u, v);
    }
}

int main() {
    getFirstTime();

    // ---- code starts here ----

    int n, m;
    fin >> n >> m;

    g.assign(n, vector<int> ());
    visited.assign(n, false);
    for(int i = 0; i < m; ++ i) {
        int a, b;
        fin >> a >> b;

        -- a; -- b;
        g[a].push_back(b);
        g[b].push_back(a);
    }

    int u, v;
    fin >> u >> v;
    -- u; -- v;
    dfs(u);

```

```

fout << visited[v];

// ---- code ends here -----

printTimeUse();
printMemoryUse();

fin.close();
fout.close();
return 0;
}

```

Текстовое объяснение решения:

Просто запустим DFS, будем пометать посещенные вершины, если вершина v окажется помеченной, значит сможем туда пройти

Результат работы кода на примерах из текста задачи:

The image shows two side-by-side screenshots of a code editor. Each screenshot has two tabs: 'input.txt' and 'output.txt'.

Left Screenshot:

- input.txt:**

```

1 4 2
2 1 2
3 3 2
4 1 4

```
- output.txt:**

```

1 0

```

Right Screenshot:

- input.txt:**

```

1 4 4
2 1 2
3 3 2
4 4 3
5 1 4
6 1 4

```
- output.txt:**

```

1 1

```

Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB
Пример из задачи	0.000000 sec	0.812500 MB
Пример из задачи	0.000000 sec	0.800781 MB

Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.796875 MB
---	--------------	-------------

Вывод по задаче:

Обычная задача, поэтому она и первая (хотя разницы с последними не чувствую).

Задача №2. Компоненты [5 s, 512 Mb, 1 балл]

Теперь вы решаете сделать так, чтобы в лабиринте не было мертвых зон, то есть чтобы из каждой клетки был доступен хотя бы один выход. Для этого вы находите связные компоненты соответствующего неориентированного графа и следите за тем, чтобы каждый компонент содержал выходную ячейку.

Дан неориентированный граф с n вершинами и m ребрами. Нужно посчитать количество компонент связности в нем.

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <queue>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
```

```

    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
    " sec\n";
}

//-----
vector<vector<int>> g;
vector<bool> visited;

void dfs(int v) {
    if(visited[v])
        return;

    visited[v] = true;
    for(int u : g[v]) {
        dfs(u);
    }
}

int main() {
    getFirstTime();

    // ---- code starts here -----

    int n, m;
    fin >> n >> m;

    g.assign(n, vector<int> ());
    visited.assign(n, false);
    for(int i = 0; i < m; ++ i) {
        int a, b;
        fin >> a >> b;

        -- a; -- b;
        g[a].push_back(b);
        g[b].push_back(a);
    }

    int cnt = 0;

    for(int i = 0; i < n; ++ i) {
        if(!visited[i]) {
            dfs(i);
            ++cnt;
        }
    }

    fout << cnt;

    // ---- code ends here -----

    printTimeUse();
    printMemoryUse();

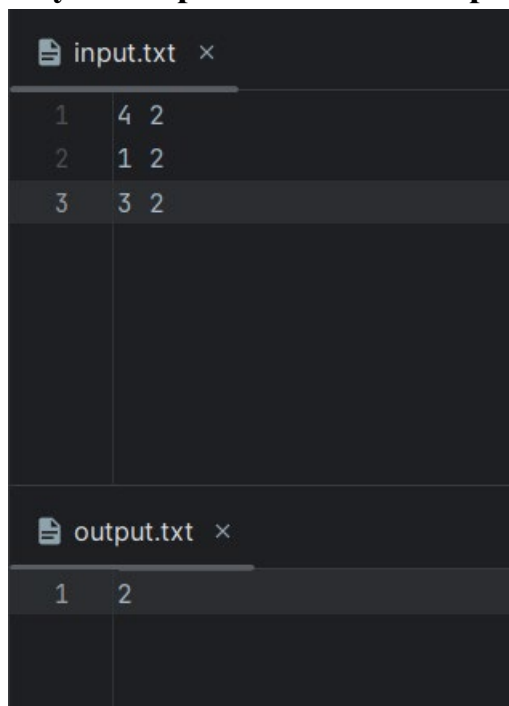
    fin.close();
    fout.close();
    return 0;
}

```

Текстовое объяснение решения:

Просто запустим DFS, будем помечать посещенные вершины, далее каждая непомеченная вершина будет давать на одну компоненту связности больше.

Результат работы кода на примерах из текста задачи:



The screenshot shows a code editor with two files open: `input.txt` and `output.txt`. The `input.txt` file contains the following text:

```
1 4 2
2 1 2
3 3 2
```

The `output.txt` file contains the following text:

```
1 2
```

Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.796875 MB
Пример из задачи	0.000000 sec	0.808594 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB

Вывод по задаче:

Легкая задача.

Задача №5. Город с односторонним движением [5 s, 512 Mb, 1.5 балла]

Департамент полиции города сделал все улицы односторонними. Вы хотели бы проверить, можно ли законно

проехать с любого перекрестка на какой-либо другой перекресток. Для этого строится ориентированный граф: вершины

– это перекрестки, существует ребро (u, v) всякий раз, когда в городе есть улица (с односторонним движением) из u в

v . Тогда достаточно проверить, все ли вершины графа лежат в одном компоненте сильной связности.

Нужно вычислить количество компонентов сильной связности заданного ориентированного графа с n вершинами

и m ребрами.

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <queue>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}

void getFirstTime() {
    start = clock();
```

```

}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
    " sec\n";
}

//-----
vector<vector<int>> g, g_rev;
vector<bool> used;
vector<int> order, component;

void dfs1(int v) {
    used[v] = true;

    for (auto u : g[v])
        if (!used[u])
            dfs1(u);

    order.push_back(v);
}

void dfs2(int v) {
    used[v] = true;
    component.push_back(v);

    for (auto u : g_rev[v])
        if (!used[u])
            dfs2(u);
}

int main() {
    getFirstTime();

    // ---- code starts here ----

    int n, m;
    fin >> n >> m;

    g.assign(n, vector<int> ());
    g_rev.assign(n, vector<int> ());
    used.assign(n, false);
    for(int i = 0; i < m; ++ i) {
        int a, b;
        fin >> a >> b;

        -- a; -- b;
        g[a].push_back(b);
        g_rev[b].push_back(a);
    }

    int cnt = 0;

    for (int i = 0; i < n; i++)
        if (!used[i])
            dfs1(i);

    used.assign(n, false);
    reverse(order.begin(), order.end());
}

```

```

for (auto v : order)
    if (!used[v]) {
        dfs2 (v);
        ++ cnt;
        component.clear();
    }

fout << cnt;

// ---- code ends here ----

printTimeUse();
printMemoryUse();

fin.close();
fout.close();
return 0;
}

```

Текстовое объяснение решения:

Запускаем два DFS, на обычных и обратных ребрах, далее идем по тем которые нашли в первом DFS, на основе их добавляем компоненты по обратным ребрам.

Результат работы кода на примерах из текста задачи:

<div>input.txt ×</div> <div>1 5 7</div> <div>2 2 1</div> <div>3 3 2</div> <div>4 3 1</div> <div>5 4 3</div> <div>6 4 1</div> <div>7 5 2</div> <div>8 5 3</div>	<div>input.txt ×</div> <div>1 4 4</div> <div>2 1 2</div> <div>3 4 1</div> <div>4 2 3</div> <div>5 3 1</div>
<div>output.txt ×</div> <div>1 5</div>	<div>output.txt ×</div> <div>1 2</div>

Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
--	------------------	----------------

Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.796875 MB
Пример из задачи	0.000000 sec	0.808594 MB
Пример из задачи	0.000000 sec	0.800781 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB

Вывод по задаче:

Аналогично простая задача.

Задача №6. Количество пересадок [10 s, 512 Mb, 1 балл]

Вы хотите вычислить минимальное количество сегментов полета, чтобы добраться из одного города в другой. Для этого вы строите следующий неориентированный граф: вершины представляют города, между двумя вершинами есть ребро всякий раз, когда между соответствующими двумя городами есть перелет. Тогда достаточно найти кратчайший путь из одного из заданных городов в другой. Дан неориентированный граф с n вершинами и m ребрами, а также две вершины u и v , нужно посчитать длину кратчайшего пути между u и v (то есть, минимальное количество ребер в пути из u в v).

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}

void getFirstTime() {
    start = clock();
}
```

```

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
    " sec\n";
}

//-----
vector<vector<pair <int, int> >> g;

int main() {
    getFirstTime();

    // ---- code starts here -----

    int n, m;
    fin >> n >> m;
    g.assign(n, vector <pair <int, int> > ());

    for(int i = 0; i < m; ++ i) {
        int a, b;
        fin >> a >> b;

        g[a - 1].emplace_back(b - 1, 1);
        g[b - 1].emplace_back(a - 1, 1);
    }

    int u, v;
    fin >> u >> v;

    vector<int> d, p;
    int s = u - 1;
    d.assign(n, INT_MAX);
    p.assign(n, -1);

    d[s] = 0;
    set<pair<int, int>> q;
    q.insert({0, s});
    while (!q.empty()) {
        int v = q.begin()->second;
        q.erase(q.begin());

        for (auto edge : g[v]) {
            int to = edge.first;
            int len = edge.second;

            if (d[v] + len < d[to]) {
                q.erase({d[to], to});
                d[to] = d[v] + len;
                p[to] = v;
                q.insert({d[to], to});
            }
        }
    }

    fout << (d[v - 1] == INT_MAX ? -1 : d[v - 1]);

    // ---- code ends here -----

    printTimeUse();
}

```

```

printMemoryUse();

fin.close();
fout.close();
return 0;
}

```

Текстовое объяснение решения:

Запустим Дейкстру и получим ответ...

Результат работы кода на примерах из текста задачи:

The screenshot shows a code editor with three files open:

- input.txt**: A 6x2 grid of numbers:

1	5 4
2	5 2
3	1 3
4	3 4
5	1 4
6	3 5
- output.txt**: A single line with the text "1 -1".
- input.txt**: A 6x2 grid of numbers:

1	4 4
2	1 2
3	4 1
4	2 3
5	3 1
6	2 4

Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB
Пример из задачи	0.000000 sec	0.796875 MB
Пример из задачи	0.000000 sec	0.796875 MB
Верхняя граница	0.000000 sec	0.808594 MB

диапазона значений входных данных из текста задачи		
--	--	--

Вывод по задаче:

Тоже простенькая задача.

Задача №7. Двудольный граф [10 s, 512 Mb, 1.5 балла]

Неориентированный граф называется двудольным, если его вершины можно разбить на две части так, что каждое ребро графа соединяет вершины из разных частей, то есть не существует рёбер между вершинами одной и той же части графа. Двудольные графы естественным образом возникают в задачах, где граф используется для моделирования связей между объектами двух разных типов (например, мальчиками и девочками, или студентами и общежитиями).

Альтернативное определение таково: граф двудольный, если его вершины можно раскрасить двумя цветами (например, черным и белым) так, что концы каждого ребра окрашены в разные цвета. Дан неориентированный граф с n вершинами и m ребрами, проверьте, является ли он двудольным.

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <queue>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}
```

```

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
    " sec\n";
}

//-----
vector<vector<int>>> g;

int main() {
    getFirstTime();

    // ---- code starts here -----

    int n, m;
    fin >> n >> m;
    g.assign(n, vector<int> ());

    for(int i = 0; i < m; ++i) {
        int a, b;
        fin >> a >> b;

        g[a - 1].push_back(b - 1);
        g[b - 1].push_back(a - 1);
    }

    vector<int> side(n, -1);
    bool is_bipartite = true;
    queue<int> q;
    for (int st = 0; st < n; ++st) {
        if (side[st] == -1) {
            q.push(st);
            side[st] = 0;
            while (!q.empty()) {
                int v = q.front();
                q.pop();
                for (int u : g[v]) {
                    if (side[u] == -1) {
                        side[u] = side[v] ^ 1;
                        q.push(u);
                    } else {
                        is_bipartite &= side[u] != side[v];
                    }
                }
            }
        }
    }

    fout << (is_bipartite ? "YES" : "NO") << endl;

    // ---- code ends here -----

    printTimeUse();
    printMemoryUse();
}

```

```

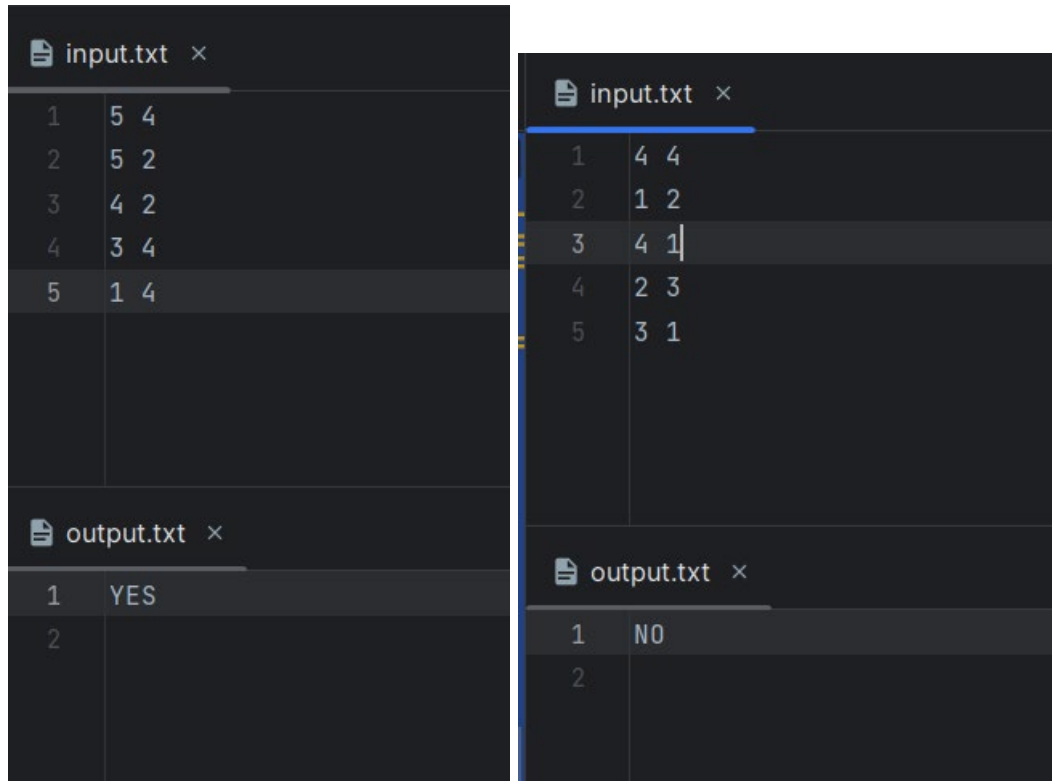
    fin.close();
    fout.close();
    return 0;
}

```

Текстовое объяснение решения:

Адаптируем BFS, чтобы у нас каждый раз проверялось на какой стороне мы находимся и запустим его.

Результат работы кода на примерах из текста задачи:



Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB
Пример из задачи	0.000000 sec	0.792969 MB
Пример из задачи	0.000000 sec	0.796875 MB
Верхняя граница диапазона значений входных данных из	0.000000 sec	0.804688 MB

текста задачи		
---------------	--	--

Вывод по задаче:

Аналогично мегасуперпростая задача.

Задача №10. Оптимальный обмен валюты [10 s, 512 Mb, 2 балла]

Теперь вы хотите вычислить оптимальный способ обмена данной вам валюты s_i на все другие валюты. Для этого вы находите кратчайшие пути из вершины s_i во все остальные вершины. Дан ориентированный граф с возможными отрицательными весами ребер, у которого n вершин и m ребер, а также задана одна его вершина s . Вычислите длину кратчайших путей из s во все остальные вершины графа.

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
    " sec\n";
}
```

```

//-----
const int INF = numeric_limits<int>::max();

struct Edge {
    int a, b, cost;
};

vector<int> bellman_ford(int n, int s, const vector<Edge> &edges) {
    vector<int> d(n, INF);
    d[s] = 0;
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (int j = 0; j < edges.size(); ++j)
            if (d[edges[j].a] < INF)
                if (d[edges[j].b] > d[edges[j].a] + edges[j].cost) {
                    d[edges[j].b] = max(-INF, d[edges[j].a] +
edges[j].cost);
                    p[edges[j].b] = edges[j].a;
                    x = edges[j].b;
                }
    }

    if (x != -1) {
        int y = x;
        for (int i = 0; i < n; ++i)
            y = p[y];

        vector<int> path;
        for (int cur = y;; cur = p[cur]) {
            path.push_back(cur);
            if (cur == y && path.size() > 1)
                break;
        }
        reverse(path.begin(), path.end());

        for (size_t i = 0; i < path.size(); ++i)
            d[path[i]] = -INF;
    }

    return d;
}

int main() {
    getFirstTime();

    // ---- code starts here ----

    int n, m, s;
    fin >> n >> m;

    vector<Edge> edges(m);
    for (int i = 0; i < m; i++) {
        int u, v, w;
        fin >> u >> v >> w;
        edges[i] = {u - 1, v - 1, w}; // 0-based indexing
    }

    fin >> s;

```

```

vector<int> dist = bellman_ford(n, s - 1, edges);

for (int i = 0; i < n; i++) {
    if (dist[i] == INF) {
        fout << "*" << endl;
    } else if (dist[i] == -INF) {
        fout << "-" << endl;
    } else {
        fout << dist[i] << endl;
    }
}

// ---- code ends here ----

printTimeUse();
printMemoryUse();

fin.close();
fout.close();
return 0;
}

```

Текстовое объяснение решения:

Используем алгоритм Форда-Беллмана, чтобы находить негативные циклы. Негативный цикл = значит выводим на нем -, иначе выводим расстояние или *, если путь отсутствует.

Результат работы кода на примерах из текста задачи:

input.txt	
1	6 7
2	1 2 10
3	2 3 5
4	1 3 100
5	3 5 7
6	5 4 10
7	4 3 -18
8	6 1 -1
9	1

output.txt	
1	0
2	10
3	-
4	-
5	-
6	*
7	

input.txt	
1	5 4
2	1 2 1
3	4 1 2
4	2 3 2
5	3 1 -5
6	4

output.txt	
1	-
2	-
3	-
4	0
5	*
6	

Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB
Пример из задачи	0.000000 sec	0.808594 MB
Пример из задачи	0.000000 sec	0.796875 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.800781 MB

Вывод по задаче:

Тоже простенькая задача.

Задача №18. Построение дорог [10 s, 512 Mb, 4 балла]

В этой задаче цель состоит в том, чтобы построить дороги между некоторыми парами заданных городов так, чтобы между любыми двумя городами существовал путь и общая длина дорог была минимальна.

Даны n точек на плоскости, соедините их отрезками минимальной общей длины так, чтобы между любыми двумя точками был путь. Напомним, что длина отрезка с концами (x_1, y_1) и (x_2, y_2) равна

p

$$(x_1 - x_2)^2 + (y_1 - y_2)^2.$$

Листинг кода:

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <math.h>
#include "windows.h"
#include "psapi.h"
#include <time.h>
#include <stdlib.h>
#include <algorithm>
#include <numeric>
#include <type_traits>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

//-----

ifstream fin("input.txt");
ofstream fout("output.txt");

clock_t start;

void printMemoryUse() {
    PROCESS_MEMORY_COUNTERS_EX pmc;
    GetProcessMemoryInfo(GetCurrentProcess(), (PROCESS_MEMORY_COUNTERS *)
    &pmc, sizeof(pmc));
    SIZE_T virtualMemUsedByMe = pmc.PrivateUsage;

    cerr << fixed << setprecision(6);
    cerr << "Memory used: " << double(virtualMemUsedByMe) / (1024. * 1024)
    << " MB\n";
}

void getFirstTime() {
    start = clock();
}

void printTimeUse() {
```

```

    cerr << fixed << setprecision(6);
    cerr << "Time used: " << (double) (clock() - start) / CLOCKS_PER_SEC <<
" sec\n";
}

//-----
const int INF = 1000000000;

struct Edge {
    double w;
    int to;
    Edge() : w(INF), to(-1) {}
    Edge(double w, int to) : w(w), to(to) {}
    bool operator<(Edge const& other) const {
        return make_pair(w, to) < make_pair(other.w, other.to);
    }
};

int n;
vector<vector<Edge>> g;

int main() {
    getFirstTime();

    // ---- code starts here ----

    fin >> n;

    vector <pair <int, int>> coords(n);
    for(int i = 0; i < n; ++ i)
        fin >> coords[i].first >> coords[i].second;

    g.assign(n, vector <Edge> ());

    for(int i = 0; i < n; ++ i)
        for(int j = 0; j < n; ++ j) {
            if(i == j)
                continue;

            g[i].emplace_back(sqrt((coords[i].first - coords[j].first) *
(coords[i].first - coords[j].first) + (coords[i].second - coords[j].second)
* (coords[i].second - coords[j].second)), j);
        }

    double total_weight = 0;
    vector<Edge> min_e(n);
    min_e[0].w = 0;
    set<Edge> q;
    q.insert({0, 0});
    vector<bool> selected(n, false);
    for (int i = 0; i < n; ++i) {
        if (q.empty()) {
            exit(0);
        }

        int v = q.begin()->to;
        selected[v] = true;
        total_weight += q.begin()->w;
        q.erase(q.begin());
    }
}

```

```

        for (Edge e : g[v]) {
            if (!selected[e.to] && e.w < min_e[e.to].w) {
                q.erase({min_e[e.to].w, e.to});
                min_e[e.to] = {e.w, v};
                q.insert({e.w, e.to});
            }
        }
    }

    fout << total_weight;

    // ---- code ends here ----

    printTimeUse();
    printMemoryUse();

    fin.close();
    fout.close();
    return 0;
}

```

Текстовое объяснение решения:

Используем алгоритм Прима для решения данной задачи, который заключается просто в добавлении наименьшего ребра по весу, который еще не выбрали.

Результат работы кода на примерах из текста задачи:

input.txt	
1	5
2	0 0
3	0 2
4	1 1
5	3 0
6	3 2

output.txt	
1	7.0645

input.txt	
1	4
2	0 0
3	0 1
4	1 0
5	1 1

output.txt	
1	3

Проверка задачи на (openedu, астр и тд при наличии в задаче):

	Время выполнения	Затраты памяти
--	------------------	----------------

Нижняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.804688 MB
Пример из задачи	0.000000 sec	0.789062 MB
Пример из задачи	0.000000 sec	0.808594 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.000000 sec	0.804688 MB

Вывод по задаче:

Аналогично простая задача.

Вывод

Легчайшая лабораторная работа.