helpers.hpp

```
#ifndef HELPERS_HPP
#define HELPERS_HPP

#include <vector>
#include "matrix.h"

using std::vector;

class Graph_display {
public:
   Graph_display ();
    ~Graph_display();
public:
    template <int N>
    void operator() (int (&matrix)[N][2]);
    void operator() ();
private:
    vector<int> t1; // մուտքային տարրեր
    vector<int> t2; // միջանկյալ տարրեր
    vector<int> t3; // ելքային տարրեր
    vector<int> t4; // յուրաքանչյուր մատրիցի զրոյական սյուների թիվը
    vector<int> t5; // զուտ 2 միջանկյալ տարրերի միջև կապերի քանակ
    vector<int> t6; // ելքային տարրերի միջև կապերի թիվ
    int t7{}; // ձևավորվող տարրերի թիվ
    vector<double> Kmo_all_values; // ամեն մատրիցի միջանկյալ տարրերի օգտագործման գործակից
    double Km{}; // միջանկյալ տարրերի գործակից
    double Knk{}; // ներքին կապերի գործակից
    double Kmo_average{}; // միջանկյալ տարրերի օգտագործման միջին գործակից
    double Kkrk{}; // ելքային տարրերի
    int unit_connections_size{}; // միավոր կապերի քանակ
    int top_size{}; // գագաթների քանակ
private:
//---------------------------------------for dynamic matrix-----------------------
    void _find_t2 (int find, int **matrix, int N);
    void _find_t3 (int **matrix, int N);
    void _find_t1_t2_t3 (int **matrix, int N);
    Matrix _creating_A (int **matrix, int N);
    // for finding t5 (both static and dynamic matrixes)
    bool _is_t2 (int num);
    // for finding t6 (both static and dynamic matrixes)
    bool _is_t3 (int num);
    void _find_t5 (int **matrix, int N);
    void _find_t6 (int **matrix, int N);
//-----------------------------------------for static matrix--------------------
    template <int N>
    void _find_t2 (int find, int (&matrix)[N][2]);
    template <int N>
    void _find_t3 (int (&matrix)[N][2]);
    template <int N>
    void _find_t1_t2_t3 (int (&matrix)[N][2]);
    template <int N>
    void _find_t5 (int (&matrix)[N][2]);
    template <int N>
    void _find_t6 (int (&matrix)[N][2]);
    template <int N>
```

```
    Matrix _creating_A (int (&matrix)[N][2]);
    // for both static and dynamic matrixes
    void _find_t4 (const Matrix& m);
    void _print_elements () const;
    void _print_t4 ();
    void _print_t7 (int count);
    bool _major_diagonal_zero_check (const Matrix& m) const;
    int **_create_direct_flow_matrix_by_input (int row) const;
    void _console_output (const Matrix& m, int count);
    void print_coefficient ();
    void _display (const Matrix& m1);
};

Graph_display::Graph_display () = default;
Graph_display::~Graph_display() = default;

template <int N>
void Graph_display::operator() (int (&matrix)[N][2]) {
    unit_connections_size = N;
    Matrix m1 = _creating_A(matrix);
    top_size = m1.get_row();
    _display(m1);
}

void Graph_display::operator() () {
    int row = 0;
    std::cout << "Input the direct-flow matrixs row: ";
    std::cin >> row;
    if (row <= 0 || row > 100)
        throw std::out_of_range("Input the corrent row (0 < row < 100)");

    unit_connections_size = row;
    int **matrix = _create_direct_flow_matrix_by_input(row);

    Matrix m1 = _creating_A(matrix, row);
    top_size = m1.get_row();

    for (int i = 0; i < row; ++i) {
        delete matrix[i];
    }
    delete matrix;
    _display(m1);
}
//---------------------------------------for dynamic matrix-----------------------
------------------------
void Graph_display::_find_t2 (int find, int **matrix, int N) {
    bool flag = false;
    for (int i = 0; i < N; ++i) {
        if (find == matrix[i][1]) {
            flag = true;
            break;
        }
    }
    if (flag) {
        for (int i = 0; i < N; ++i) {
            if (find == matrix[i][0] && matrix[i][1] != 0) {
                t2.push_back(find);
                break;
            }
        }
```

```
115     }
116 }
117
118 void Graph_display::_find_t3 (int **matrix, int N) {
119     for (int i = 0; i < N; ++i) {
120         if (matrix[i][1] == 0)
121             t3.push_back(matrix[i][0]);
122     }
123 }
124
125 void Graph_display::_find_t1_t2_t3 (int **matrix, int N) {
126     int check = 0;
127     bool flag = true;
128     for (int i = 0; i < N; ++i) {
129         flag = true;
130         if (check != matrix[i][0]) {
131             check = matrix[i][0];
132             for (int j = 0; j < N; ++j) {
133                 if (check == matrix[j][1]) {
134                     flag = false;
135                     break;
136                 }
137             }
138             if (flag)
139                 t1.push_back(check);
140             else {
141                 _find_t2(check, matrix, N);
142             }
143         }
144     }
145     _find_t3(matrix, N);
146 }
147
148 Matrix Graph_display::_creating_A (int **matrix, int N) {
149     _find_t1_t2_t3(matrix, N);
150     _find_t5(matrix, N);
151     _find_t6(matrix, N);
152     int max = matrix[0][0];
153     for (int i = 1; i < N; ++i) {
154         if (max < matrix[i][1])
155             max = matrix[i][1];
156     }
157     Matrix goal(max, max);
158     for (int i = 0; i < N; ++i) {
159         goal(matrix[i][0] - 1, matrix[i][1] - 1) = 1;
160     }
161
162     return goal;
163 }
164
165
166 // for finding t5 (both static and dynamic matrixes)
167 bool Graph_display::_is_t2 (int num) {
168     for (int i : t2) {
169         if (i == num)
170             return true;
171     }
172     return false;
173 }
174
```

```
175 // for finding t6 (both static and dynamic matrixes)
176 bool Graph_display::_is_t3 (int num) {
177     for (int i : t3) {
178         if (i == num)
179             return true;
180     }
181     return false;
182 }
183 
184 void Graph_display::_find_t5 (int **matrix, int N) {
185     for (int i = 0; i < N; ++i) {
186         if (_is_t2(matrix[i][0])) {
187             if (_is_t2(matrix[i][1])) {
188                 t5.push_back(matrix[i][0]);
189             }
190         }
191         else if (_is_t3(matrix[i][0])) {
192             if (_is_t3(matrix[i][1])) {
193                 t6.push_back(matrix[i][0]);
194             }
195         }
196     }
197 }
198 
199 void Graph_display::_find_t6 (int **matrix, int N) {
200     for (int i = 0; i < N; ++i) {
201         if (_is_t3(matrix[i][0])) {
202             if (_is_t3(matrix[i][1])) {
203                 t6.push_back(matrix[i][0]);
204             }
205         }
206     }
207 }
208 
209 //-----------------------------------------for static matrix-------------------
    ---------------------------------
210 template <int N>
211 void Graph_display::_find_t2 (int find, int (&matrix)[N][2]) {
212     bool flag = false;
213     for (int i = 0; i < N; ++i) {
214         if (find == matrix[i][1]) {
215             flag = true;
216             break;
217         }
218     }
219     if (flag) {
220         for (int i = 0; i < N; ++i) {
221             if (find == matrix[i][0] && matrix[i][1] != 0) {
222                 t2.push_back(find);
223                 break;
224             }
225         }
226     }
227 }
228 
229 template <int N>
230 void Graph_display::_find_t3 (int (&matrix)[N][2]) {
231     for (int i = 0; i < N; ++i) {
232         if (matrix[i][1] == 0)
233             t3.push_back(matrix[i][0]);
```

```
234     }
235 }
236 
237 template <int N>
238 void Graph_display::_find_t1_t2_t3 (int (&matrix)[N][2]) {
239     int check = 0;
240     bool flag = true;
241     for (int i = 0; i < N; ++i) {
242         flag = true;
243         if (check != matrix[i][0]) {
244             check = matrix[i][0];
245             for (int j = 0; j < N; ++j) {
246                 if (check == matrix[j][1]) {
247                     flag = false;
248                     break;
249                 }
250             }
251             if (flag)
252                 t1.push_back(check);
253             else {
254                 _find_t2(check, matrix);
255             }
256         }
257     }
258     _find_t3(matrix);
259 }
260 
261 template <int N>
262 void Graph_display::_find_t5 (int (&matrix)[N][2]) {
263     for (int i = 0; i < N; ++i) {
264         if (_is_t2(matrix[i][0])) {
265             if (_is_t2(matrix[i][1])) {
266                 t5.push_back(matrix[i][0]);
267             }
268         }
269     }
270 }
271 
272 template <int N>
273 void Graph_display::_find_t6 (int (&matrix)[N][2]) {
274     for (int i = 0; i < N; ++i) {
275         if (_is_t3(matrix[i][0])) {
276             if (_is_t3(matrix[i][1])) {
277                 t6.push_back(matrix[i][0]);
278             }
279         }
280     }
281 }
282 
283 template <int N>
284 Matrix Graph_display::_creating_A (int (&matrix)[N][2]) {
285     _find_t1_t2_t3(matrix);
286     _find_t5(matrix);
287     _find_t6(matrix);
288     int max = matrix[0][0];
289     for (int i = 1; i < N; ++i) {
290         if (max < matrix[i][1])
291             max = matrix[i][1];
292     }
293     Matrix goal(max, max);
```

```
294     for (int i = 0; i < N; ++i) {
295         goal(matrix[i][0] - 1, matrix[i][1] - 1) = 1;
296     }
297 
298     return goal;
299 }
300 
301 // for both static and dynamic matrixes
302 void Graph_display::_find_t4 (const Matrix& m) {
303     bool flag = true;
304     int i = 0;
305     int j = 0;
306     while (i < m.get_row()) {
307         while (j < m.get_col()) {
308             if (m(j, i) != 0) {
309                 flag = false;
310                 break;
311             }
312             ++j;
313         }
314         if (flag)
315             t4.push_back(i + 1);
316         ++i;
317         j = 0;
318         flag = true;
319     }
320 }
321 
322 void Graph_display::_print_elements () const {
323     std::cout << "t1 = " << t1.size() << "-> { ";
324     for (int i : t1) {
325         std::cout << i << ' ';
326     }
327     std::cout << '}' << std::endl;
328     std::cout << "t2 = " << t2.size() << "-> { ";
329     for (int i : t2) {
330         std::cout << i << ' ';
331     }
332     std::cout << '}' << std::endl;
333     std::cout << "t3 = " << t3.size() << "-> { ";
334     for (int i : t3) {
335         std::cout << i << ' ';
336     }
337     std::cout << '}' << std::endl;
338     std::cout << "t5 = " << t5.size() << "-> { ";
339     for (int i : t5) {
340         std::cout << i << ' ';
341     }
342     std::cout << '}' << std::endl;
343     std::cout << "t6 = " << t6.size() << "-> { ";
344     for (int i : t6) {
345         std::cout << i << ' ';
346     }
347     std::cout << '}' << std::endl;
348 }
349 
350 void Graph_display::_print_t4 () {
351     std::cout << "t4 = " << t4.size() << "-> { ";
352     for (int i : t4) {
353         std::cout << i << ' ';
```

```
354     }
355     std::cout << '}' << std::endl;
356     t4.clear();
357 }
358 
359 void Graph_display::_print_t7 (int count) {
360     t7 = t4.size() - count;
361     std::cout << "t7 = " << t7 << std::endl;
362 }
363 
364 bool Graph_display::_major_diagonal_zero_check (const Matrix& m) const {
365     for (int i = 0; i < m.get_row(); ++i) {
366         if (m(i, i) != 0)
367             return true;
368     }
369     return false;
370 }
371 
372 int** Graph_display::_create_direct_flow_matrix_by_input (int row) const {
373     int **matrix = nullptr;
374 
375     matrix = new int*[row];
376     for (int i = 0; i < row; ++i) {
377         matrix[i] = new int[2];
378     }
379 
380     for (int i = 0; i < row; ++i) {
381         for (int j = 0; j < 2; ++j) {
382             std::cout << "matrix[" << i << "][" << j << "] = ";
383             std::cin >> matrix[i][j];
384         }
385     }
386 
387     return matrix;
388 }
389 
390 void Graph_display::_console_output (const Matrix& m, int count) {
391     std::cout << "------------------------------------------------------------
------------------" << std::endl;
392     std::cout << "\n A^" << count << std::endl;
393     m.display();
394     _find_t4(m);
395     _print_t7(count);
396     double Kmo = static_cast<double>(t7) / static_cast<double>(t4.size());
397     Kmo_all_values.push_back(Kmo);
398     _print_t4();
399     std::cout << "Kmo = " << Kmo << '\n';
400     std::cout << "------------------------------------------------------------
------------------" << std::endl;
401 }
402 
403 void Graph_display::print_coefficient () {
404 
405     double size = Kmo_all_values.size();
406     for (int i = 0; i < size; ++i) {
407         Kmo_average += Kmo_all_values[i];
408     }
409 
410     Kmo_average /= size;
411     Km = static_cast<double>(t2.size()) / static_cast<double>(top_size);
```

```
412     Knk = static_cast<double>(t5.size()) / static_cast<double>
(unit_connections_size);
413     Kkrk = 2 * static_cast<double>(t6.size()) / (static_cast<double>(t3.size()) *
(static_cast<double>(t3.size()) - 1));
414     std::cout << "Km = " << Km << '\n';
415     std::cout << "Knk = " << Knk << '\n';
416     std::cout << "Kmo_average = " << Kmo_average << std::endl;
417     std::cout << "Kkrk = " << Kkrk << std::endl;
418 }
419
420 void Graph_display::_display (const Matrix& m1) {
421     char check{};
422     short count{1};
423     Matrix m2 = m1;
424     Matrix m3;
425     Matrix delta = m1;
426     system("clear");
427     _console_output(m1, count++);
428
429     do {
430         if (m3.all_is_zero())
431             break;
432         m3 = m2 * m1;
433         delta += m3;
434
435         _console_output(m3, count);
436
437
438         if (_major_diagonal_zero_check(m3))
439             throw std::invalid_argument("There is a circuit in matrix (check the
diagonal)");
440
441         ++count;
442         if (count <= m1.get_row() && check != 'd') {
443             std::cout << "Do you want to get A^" << count << " (if you want to get
all, type (d))(y, n)? ";
444             std::cin >> check;
445         }
446         m2 = m3;
447     } while (check != 'n' && count <= m1.get_row());
448     if (check != 'n') {
449         std::cout << "\nDelta matrix: " << std::endl;
450         delta.display();
451         _find_t4(delta);
452         _print_elements();
453         _print_t4();
454         print_coefficient();
455     }
456 }
457
458 #endif // HELPERS_HPP
```