МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №4.2

по дисциплине: Дискретная математика
тема: «Циклы»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
ст. пр. Рязанов Юрий Дмитриевич
ст. пр. Бондаренко Татьяна Владимировна

Белгород 2023 г.

Лабораторная работа №4.2

Циклы

Цель работы: изучить разновидности циклов в графах, научиться генерировать случайные графы, определять их принадлежность к множеству эйлеровых и гамильтоновых графов, находить все эйлеровы и гамильтоновы циклы в графах.

1. Разработать и реализовать алгоритм генерации случайного графа, содержащего n вершин и m ребер.

```
#include "../../libs/alg/alg.h"

#include <random>

typedef Node<int> IntNode;

Graph<Edge<IntNode>>* generateRandomGraph(int nodesCount, int edgeCount) {
    auto graph = new AdjacencyMatrixGraph<Edge<IntNode>>();

    for (int i = 1; i <= nodesCount; i++) {
        Node<int> n(i);

        graph->addNode(n);
    }

    std::vector<std::pair<int, int>> edges;
    for (int i = 1; i <= nodesCount; i++) {
        for (int j = 1; j <= nodesCount; j++) {
            if (i == j) continue;

            edges.push_back({i, j});
        }
    }

    std::random_device rd;
    std::mt19937 g(rd());

    std::shuffle(edges.begin(), edges.end(), g);

    for (int i = 0; i < edgeCount && i < edges.size(); i++)
        graph->addEdge({{(*graph)[edges[i].first],
        (*graph)[edges[i].second]}, false});

    return graph;
}
```

2. Написать программу, которая:

   а) в течение десяти секунд генерирует случайные графы, содержащие n вершин и m ребер;

б) для каждого полученного графа определяет, является ли он эйлеровым или гамильтоновым;

в) подсчитывает общее количество сгенерированных графов и количество графов каждого типа.

Результат работы программы представить в виде таблицы. Величину h подобрать такой, чтобы в таблице количество строк было в диапазоне от 20 до 30. *task1.tpp*

```
#include "../lab11/graph.tpp"

template <typename E, typename N>
bool AdjacencyMatrixGraph<E, N>::isHamiltonian() {
    if (this->nodes.size() < 3) return false;
    // От теоремы Оре и Диррака пришось отказаться, так как мы можем
    // получить ориентированный граф, теоремы расчитаны только для
    // неориентированных графов.
    std::vector<bool> cache(this->nodes.size(), false);

    for (int i = 0; i < this->nodes.size(); i++)
        if (hasHamiltonianCycle(i, i, cache)) return true;

    return false;
}

template <typename E, typename N>
bool AdjacencyMatrixGraph<E, N>::hasHamiltonianCycle(int originIndex, int startIndex, std::vector<bool>&
↪  takenNodes) {
    takenNodes[startIndex] = true;

    bool anyElementFound = false;
    for (int i = 0; i < this->nodes.size(); i++) {
        if (takenNodes[i] || this->edges[startIndex][i] == nullptr) continue;

        anyElementFound = true;

        if (hasHamiltonianCycle(originIndex, i, takenNodes)) return true;
    }

    if (!anyElementFound) {
        if (this->edges[startIndex][originIndex] == nullptr) {
            takenNodes[startIndex] = false;

            return false;
        }

        for (auto takenFlag : takenNodes)
            if (!takenFlag) {
                takenNodes[startIndex] = false;

                return false;
            }

        return true;
```

```
    }

    takenNodes[startIndex] = false;
    return false;
}

// Для данной задачи метод был упрощён в угоду скорости выполнения,
// мы игнорируем наличие мультиграфов, орграфов.
// TODO: Переделать перед использованием!
template <typename E, typename N>
bool AdjacencyMatrixGraph<E, N>::isEuler() {
    if (this->nodes.size() < 3) return false;

    int edgeCount = 0;
    for (int i = 0; i < this->nodes.size(); i++)
        for (int j = 0; j < this->nodes.size(); j++)
            if (this->edges[i][j] != nullptr) edgeCount++;

    for (int i = 0; i < this->nodes.size(); i++) {
        std::vector<std::vector<bool>> empty(this->nodes.size(),
        std::vector<bool>(this->nodes.size(), false));
        if (hasEulerCycle(i, i, empty, 0, edgeCount)) return true;
    }

    return false;
}

// Для данной задачи метод был упрощён в угоду скорости выполнения,
// мы игнорируем наличие мультиграфов, орграфов.
// TODO: Переделать перед использованием!
template <typename E, typename N>
bool AdjacencyMatrixGraph<E, N>::hasEulerCycle(int originIndex, int startIndex,
std::vector<std::vector<bool>>& visited, int visitedSize, int totalEdgesCount) {
    if (visitedSize == totalEdgesCount && originIndex == startIndex) return true;
    if (visitedSize > totalEdgesCount) return false;

    for (int i = 0; i < this->nodes.size(); i++) {
        if (this->edges[startIndex][i] == nullptr) continue;
        if (visited[startIndex][i]) continue;

        visited[startIndex][i] = true;
        visited[i][startIndex] = this->edges[i][startIndex] != nullptr;

        if (hasEulerCycle(originIndex, i, visited, visitedSize + 1 + (this->edges[i][startIndex] != nullptr),
        ↪  totalEdgesCount)) return true;

        visited[startIndex][i] = false;
        visited[i][startIndex] = false;
    }

    return false;
}
```

main.cpp

```cpp
#include "task1.h"

#include <thread>
#include <future>
#include <unistd.h>

struct Report {
    int n;
    double h;
    int edgesCount;
    int hamilthonCount;
    int eulerCount;
    int totalCount;
};

#define LOWER_N 8
#define UPPER_N 10
#define ELEMENTS_FOR_N 10

#define CORES_AMOUNT 10
std::atomic<int> takenCores;

int main() {
    std::vector<std::pair<std::chrono::_V2::system_clock::time_point, std::future<Report>>> pool;
    for (int n = LOWER_N; n <= UPPER_N; n++) {
        double h = ((n * (n - 1)) / 2 - 1.0 * n) / (ELEMENTS_FOR_N);

        for (double e = n; e <= ((n * (n - 1)) / 2); e += h) {
            while (takenCores >= CORES_AMOUNT) {
                sleep(1);
                std::cout << "Working on it..." << "\n";
            }

            takenCores++;
            pool.push_back({std::chrono::system_clock::now(), std::async(std::launch::async, [n, h, e] {
                int totalGraphsGenerated = 0;
                int hamilthonGraphs = 0;
                int eulerGraphs = 0;
                auto start = std::chrono::system_clock::now();
                while (std::chrono::system_clock::now() - start < std::chrono::seconds(10)) {
                    auto graph = generateRandomGraph(n, static_cast<int>(e));
                    totalGraphsGenerated++;

                    if (graph->isHamiltonian()) hamilthonGraphs++;
                    if (graph->isEuler()) eulerGraphs++;

                    delete graph;
                }

                takenCores--;

                return (Report){n, h, static_cast<int>(e), hamilthonGraphs, eulerGraphs, totalGraphsGenerated};
```

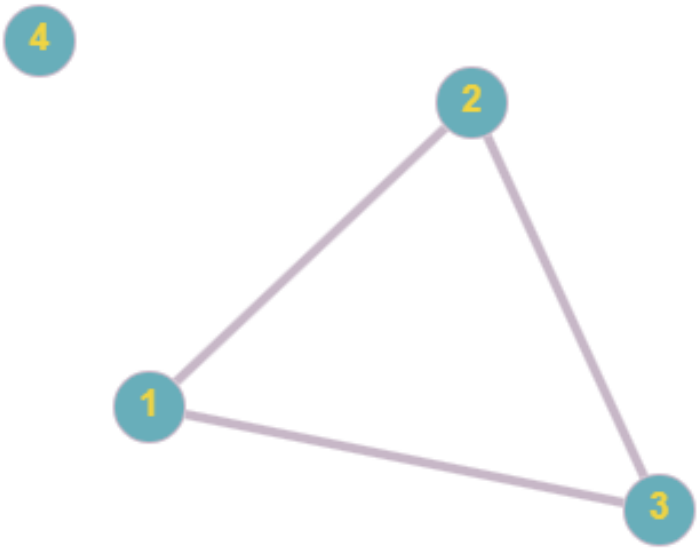
```
            }));
        }
    }



    std::cout << "Waiting for results...\n\n";
    for (auto& future : pool) {
        if (future.second.wait_until(future.first + std::chrono::seconds(60)) != std::future_status::ready) {
            std::cout << "====================\n";
            std::cout << "Thread is not responing, consider no result is present.\n";

            continue;
        }

        auto t = future.second.get();

        std::cout << "====================\n";
        std::cout << "n = " << t.n << "\n";
        std::cout << "h = " << t.h << "\n";
        std::cout << "edges = " << t.edgesCount << "\n";
        std::cout << "hamilthon = " << t.hamilthonCount << "\n";
        std::cout << "euler = " << t.eulerCount << "\n";
        std::cout << "total = " << t.totalCount << "\n\n";
    }

    getchar();

    return 0;
}
```

Результаты работы программы

Количество вершин	Количество рёбер	Количество графов		
		эйлеровых	гамильтоновых	всех
8	8	1810	125	241387
8	10	1244	2797	167884
8	12	676	9627	84592
8	14	220	8366	25491
8	16	37	2401	4170
8	18	5	419	568
8	20	0	68	75
8	22	0	25	27
8	24	0	1	1
8	26	0	1	1
8	28	0	0	0
9	9	940	42	250760
9	11	609	849	148928
9	14	145	3750	36355
9	17	11	997	2666
9	19	1	186	315
9	22	0	20	24
9	25	0	0	0
9	27	0	1	1
9	30	0	0	0
9	33	0	0	0
9	36	0	0	0
10	10	376	8	217316
10	13	153	576	78020
10	17	11	899	5239
10	20	2	81	204
10	24	0	4	6
10	27	0	5	5
10	31	0	0	0
10	34	0	0	0
10	38	0	0	0
10	41	0	0	0
10	45	0	0	0

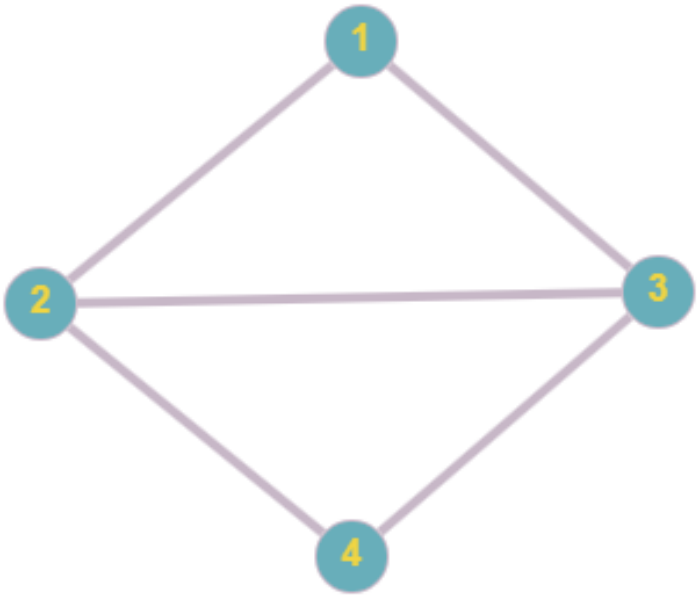
3. Выполнить программу при n = 8,9,10 и сделать выводы.
   Чем больше рёбер в графе, тем больше вероятность того, что он окажется гамильтоновым и тем меньше вероятность того, что граф окажется эйлеровым.

4. Привести пример диаграммы графа, который является эйлеровым, но не гамильтоновым. Найти в нем все эйлеровы циклы.



Эйлеровы циклы:
$\{1, 2, 3, 1\}$
$\{2, 3, 1, 2\}$
$\{3, 1, 2, 3\}$
$\{1, 3, 2, 1\}$
$\{3, 2, 1, 3\}$
$\{2, 1, 3, 2\}$

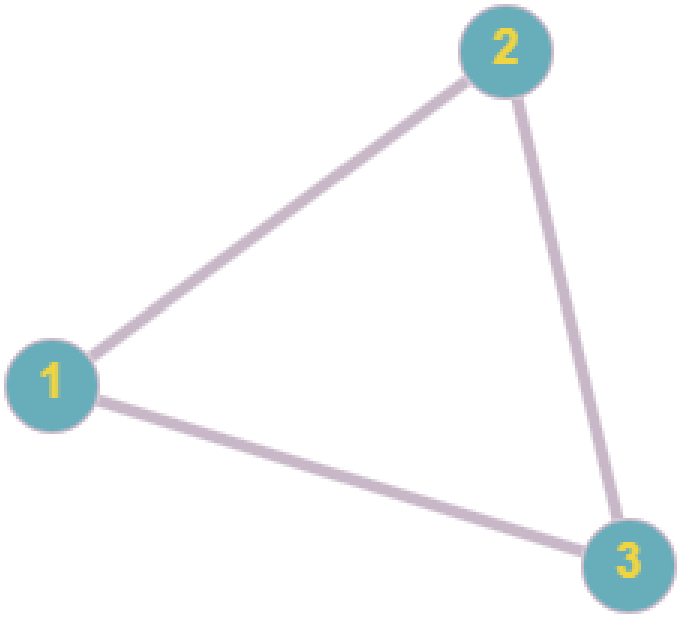
5. Привести пример диаграммы графа, который является гамильтоновым, но не эйлеровым. Найти в нем все гамильтоновы циклы.



Гамильтоновы циклы:
$\{1, 2, 4, 3, 1\}$
$\{2, 4, 3, 1, 2\}$
$\{4, 3, 1, 2, 4\}$
$\{3, 1, 2, 4, 3\}$

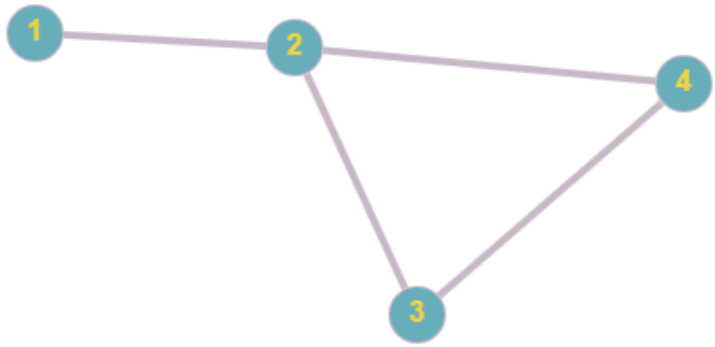
$\{1, 3, 4, 2, 1\}$
$\{3, 4, 2, 1, 3\}$
$\{4, 2, 1, 3, 4\}$
$\{2, 1, 3, 4, 2\}$

6. Привести пример диаграммы графа, который является эйлеровым и гамильтоновым. Найти в нем все эйлеровы и гамильтоновы циклы.



Эйлеровы циклы:
$\{1, 2, 3, 1\}$
$\{2, 3, 1, 2\}$
$\{3, 1, 2, 3\}$
$\{1, 3, 2, 1\}$
$\{3, 2, 1, 3\}$
$\{2, 1, 3, 2\}$
Гамильтоновы циклы:
$\{1, 2, 3, 1\}$
$\{2, 3, 1, 2\}$
$\{3, 1, 2, 3\}$
$\{1, 3, 2, 1\}$
$\{3, 2, 1, 3\}$
$\{2, 1, 3, 2\}$

7. Привести пример диаграммы графа, который не является ни эйлеровым, ни гамильтоновым.



Вывод: в ходе лабораторной работы изучили разновидности циклов в графах, научились генерировать случайные графы, определять их принадлежность к множеству эйлеровых и гамильтоновых графов, находить все эйлеровы и гамильтоновы циклы в графах.