Отчет по расчетной работе по дисциплине ПиОИвИС

Тема: Графы

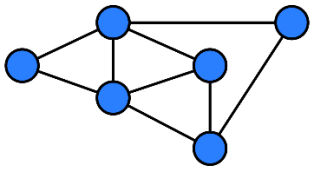
Цель: Найти циклы указанной длины в графе.

Задача: Создать алгоритм, который будет брать у пользователя информацию о ребрах и вершинах, также информацию о начальной и конечной вершины, и вес ребра, чтобы найти цикл, указанной длины.

Вариант: 5.17(внг, си)

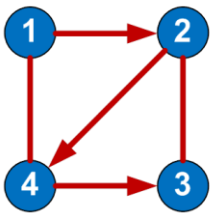
Список ключевых понятий:

- Граф – это совокупность непустого множества вершин и множества пар вершин.

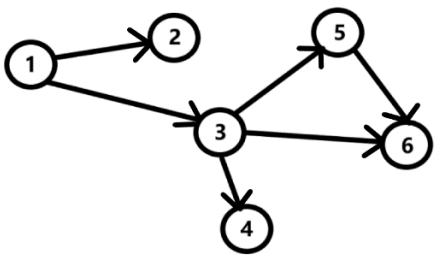


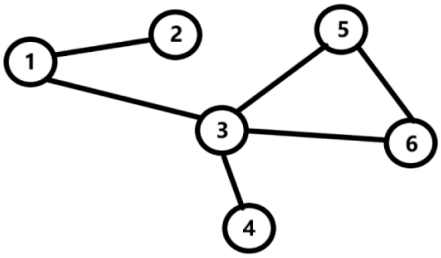
- Список смежности — один из способов представления графа в виде коллекции списков вершин. Каждой вершине графа соответствует список, состоящий из «соседей» этой вершины.

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

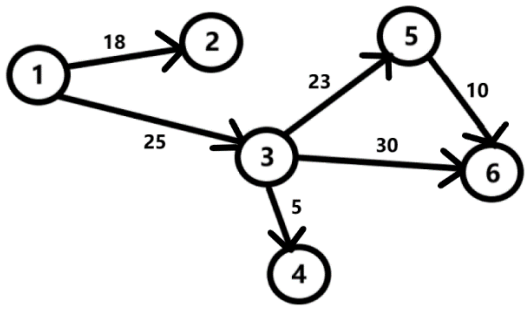


- В теории графов приняты следующие термины - графы с односторонними ребрами называют ориентированными, иначе - неориентированными.





- Также мы можем присвоить ребрам графа какой-либо параметр, например, стоимость проезда по дороге, тогда наш граф становится взвешенным.



Алгоритм:

- Создается класс Graph, который представляет граф. В конструкторе класса задается количество вершин графа, инициализируется список смежности для хранения ребер.
- Ребра добавляются в граф с помощью метода addEdge, который принимает исходную вершину (src), целевую вершину (dest) и вес ребра (weight).
- Для проверки наличия цикла указанной длины вызывается метод hasCycleOfLength, который принимает длину цикла (cycleLength). В этом методе выполняется обход графа по всем вершинам.
- Внутри метода hasCycleOfLength создается вектор visited для отслеживания посещенных вершин и множество path для отслеживания текущего пути.
- Для каждой вершины v выполняется рекурсивная вспомогательная функция hasCycleOfLengthUtil, которая проверяет наличие цикла указанной длины, начинающегося с вершины v.
- В функции hasCycleOfLengthUtil проверяется, является ли текущая длина пути больше указанной длины цикла. Если да, то цикл данной длины невозможен, и возвращается false.
- Если текущая длина пути равна указанной длине цикла и текущая вершина v присутствует в пути, то найден цикл указанной длины, и возвращается true.
- В противном случае, для текущей вершины v устанавливается флаг visited[v] = true и она добавляется в путь path.
- Затем происходит рекурсивный обход всех непосещенных соседних вершин next, и если один из них возвращает true из рекурсивного вызова функции hasCycleOfLengthUtil, то цикл указанной длины найден, и возвращается true.

- Если соседняя вершина next уже посещена и принадлежит пути, и текущая длина пути равна cycleLength - 1, то добавляется next в путь и возвращается true.
- По завершении рекурсии из пути удаляется текущая вершина v, и функция возвращает false.

Код(внг):

```
#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

// Структура для представления ребра в графе
struct Edge {
    int src, dest, weight;
};

// Класс для представления графа
class Graph {
    int V; // Количество вершин графа
    vector<vector<pair<int, int>>> adjList; // Список смежности для хранения ребер

public:
    Graph(int V) : V(V) {
        adjList.resize(V);
    }

    // Добавление ребра в граф
    void addEdge(int src, int dest, int weight) {
        adjList[src].push_back({ dest, weight });
        adjList[dest].push_back({ src, weight });
    }

    // Функция для проверки наличия цикла указанной длины с помощью DFS
    bool hasCycleOfLength(int cycleLength) {
        for (int v = 0; v < V; ++v) {
            vector<bool> visited(V, false); // Флаги посещения вершин
            unordered_set<int> path; // Сет для отслеживания пути
            if (hasCycleOfLengthUtil(v, cycleLength, visited, path))
                return true;
        }
```

```
        return false;
    }

private:
    // Вспомогательная функция для поиска цикла указанной длины с помощью DFS
    bool hasCycleOfLengthUtil(int v, int cycleLength, vector<bool>& visited, unordered_set<int>& path) {
        if (path.size() > cycleLength)
            return false;

        if (path.size() == cycleLength && path.find(v) != path.end())
            return true;

        visited[v] = true;
        path.insert(v);

        for (const auto& neighbor : adjList[v]) {
            int next = neighbor.first;
            if (!visited[next]) {
                if (hasCycleOfLengthUtil(next, cycleLength, visited, path))
                    return true;
            }
            else if (path.find(next) != path.end() && path.size() == cycleLength - 1) {
                // Если соседняя вершина уже посещена и принадлежит пути, проверяем, достигнута ли целевая длина цикла
                path.insert(next);
                return true;
            }
        }

        path.erase(v);
        return false;
    }
};

int main() {
    setlocale(LC_ALL, "Russian");
    int V, E; // Количество вершин и ребер графа
    cout << "Введите количество вершин графа: ";
    cin >> V;

    Graph g(V);

    cout << "Введите количество ребер графа: ";
```

```
    cin >> E;

    cout << "Введите ребра в формате <src> <dest> <weight>:" << endl;
    for (int i = 0; i < E; ++i) {
        int src, dest, weight;
        cin >> src >> dest >> weight;
        g.addEdge(src, dest, weight);
    }

    int cycleLength; // Длина цикла, которую нужно найти
    cout << "Введите длину цикла: ";
    cin >> cycleLength;

    if (g.hasCycleOfLength(cycleLength))
        cout << "Граф содержит цикл длины " << cycleLength << endl;
    else
        cout << "Граф не содержит цикл длины " << cycleLength << endl;

    return 0;
     }
```

Этот код подходит для взвешенного неориентированного графа, но для того, чтобы найти цикл указанной длины через список смежности, потребуется внести пару поправок.

Код:

```
#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

class Graph {
    int V; // Количество вершин графа
    vector<unordered_set<int>> adjList; // Список смежности для хранения вершин
public:
    Graph(int V) : V(V), adjList(V) {}

    void addEdge(int src, int dest) {
```

```
        adjList[src].insert(dest);
        adjList[dest].insert(src);
    }

    bool hasCycleOfLength(int cycleLength) {
        vector<bool> visited(V, false); // Флаги посещения вершин

        for (int v = 0; v < V; ++v) {
            if (dfs(v, cycleLength, visited, v, 1))
                return true;
        }

        return false;
    }

private:
    bool dfs(int v, int cycleLength, vector<bool>& visited, int startVertex, int
pathLength) {
        if (pathLength == cycleLength && adjList[v].count(startVertex))
            return true;

        visited[v] = true;

        for (const auto& neighbor : adjList[v]) {
            if (!visited[neighbor] && dfs(neighbor, cycleLength, visited, startVertex,
pathLength + 1))
                return true;
        }

        visited[v] = false;

        return false;
    }
};

int main() {
    setlocale(LC_ALL, "Russian");

    int V, E; // Количество вершин и ребер графа
    cout << "Введите количество вершин графа: ";
    cin >> V;

    Graph g(V);

    cout << "Введите количество ребер графа: ";
```

```
    cin >> E;

    cout << "Введите ребра в формате <src> <dest>:" << endl;
    for (int i = 0; i < E; ++i) {
        int src, dest;
        cin >> src >> dest;
        g.addEdge(src, dest);
    }

    int cycleLength; // Длина цикла, которую нужно найти
    cout << "Введите длину цикла: ";
    cin >> cycleLength;

    if (g.hasCycleOfLength(cycleLength))
        cout << "Граф содержит цикл длины " << cycleLength << endl;
    else
        cout << "Граф не содержит цикл длины " << cycleLength << endl;

    return 0;
}
```