

Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего профессионального образования  
Кубанский государственный технологический университет  
(ФГБОУ ВПО КубГТУ)

Факультет Компьютерных технологий и автоматизированных систем  
Кафедра Информационных систем и программирования

## КУРСОВАЯ РАБОТА

По дисциплине Дискретная математика

На тему Задачи и алгоритмы дискретной математики

Выполнил студент группы 12-КБ-ПР1

Сологуб Денис Сергеевич

Допущен к защите:

Руководитель работы: \_\_\_\_\_ Симоненко Е.А.

(подпись, дата, расшифровка подписи)

Защищен \_\_\_\_\_ Оценка \_\_\_\_\_

(дата)

Члены комиссии Симоненко Е. А.

Волик А. Г.

Краснодар  
2013



## РЕФЕРАТ

### ПЛАНАРНЫЙ ГРАФ, ОТРИЦАТЕЛЬНЫЙ ЦИКЛ, ФОРМУЛА ЭЙЛЕРА, АЛГОРИТМ ФЛОЙДА, ВЗВЕШЕННЫЙ ГРАФ

Стр. 17 , рис. 9 , ист. 3 .

В данной курсовой работе были рассмотрены такие понятия, как планарный, взвешенный ориентированный и неориентированный граф, изучен алгоритм проверки графа на планарность по формуле Эйлера, алгоритм Флойда получения матрицы кратчайших путей.

Также были изучены понятия графа с отрицательными весами и циклами, и реализован алгоритм нахождения Флойда получения матрицы кратчайших путей.

При выполнении обеих частей работы, важно было оценить вычислительную трудоемкость выбранного алгоритма и, соответственно, его приемлемость для решения рассматриваемой задачи.

Таким образом, данная работа помогла оценить и проанализировать целесообразность применения рассмотренных алгоритмов с учетом поставленной задачи.

## Содержание

Введение.....	5
1 Планарность графа.....	6
1.1 Формулировка задачи.....	6
1.2 Планарность графа.....	6
1.2.1 Формула Эйлера.....	6
1.3 Описание программы 1.....	7
1.4 Результаты тестирования.....	7
2 Кратчайшие пути: алгоритм Флойда.....	10
2.1 Формулировка задания.....	10
2.2 Матрица кратчайших путей.....	10
2.2.1 Алгоритм Флойда.....	10
2.3 Описание программы 2.....	10
2.4 Результаты тестирования:.....	11
Список использованных источников.....	13
Приложение А.....	14
Приложение Б.....	16

## **Введение**

В данной курсовой работе был реализован алгоритм по формуле Эйлера и алгоритм Флойда. На первом этапе было проведено исследование основного теоретического материала по данным темам, а также изучение основных алгоритмов. Эти исследования проводились с помощью информации, найденной в источниках. Для проверки графа на планарность использовали формулу Эйлера, по алгоритму Флойда, для нахождения матрицы кратчайших путей, использовали рекомендованный источник Окулов С. М. «Дискретная математика».

Для демонстрации работы алгоритмов были написаны необходимые программы на языке программирования C++ в среде Qt Creator. После написания необходимых программ было проведено тестирование.

## 1 Планарность графа

### 1.1 Формулировка задачи

Изучить тему «Планарность графа». Исследовать алгоритм проверки графа на планарность на планарность. Исследовать алгоритм укладки графа на плоскость. Разработать программу демонстрирующую работу алгоритма, провести тестирование.

Для выполнения задания были предложены рекомендованные источники для поиска информации.

### 1.2 Планарность графа

Планарным называется такой граф, который обладает двухмерной вершинно-реберной метрикой, то есть такой граф может быть уложен без наложения вершин или ребер друг на друга на любую действительную поверхность, обладающую двухмерной метрикой и без «петель».

Ниже представлен пример планарного графа (см. рисунок 1.1).

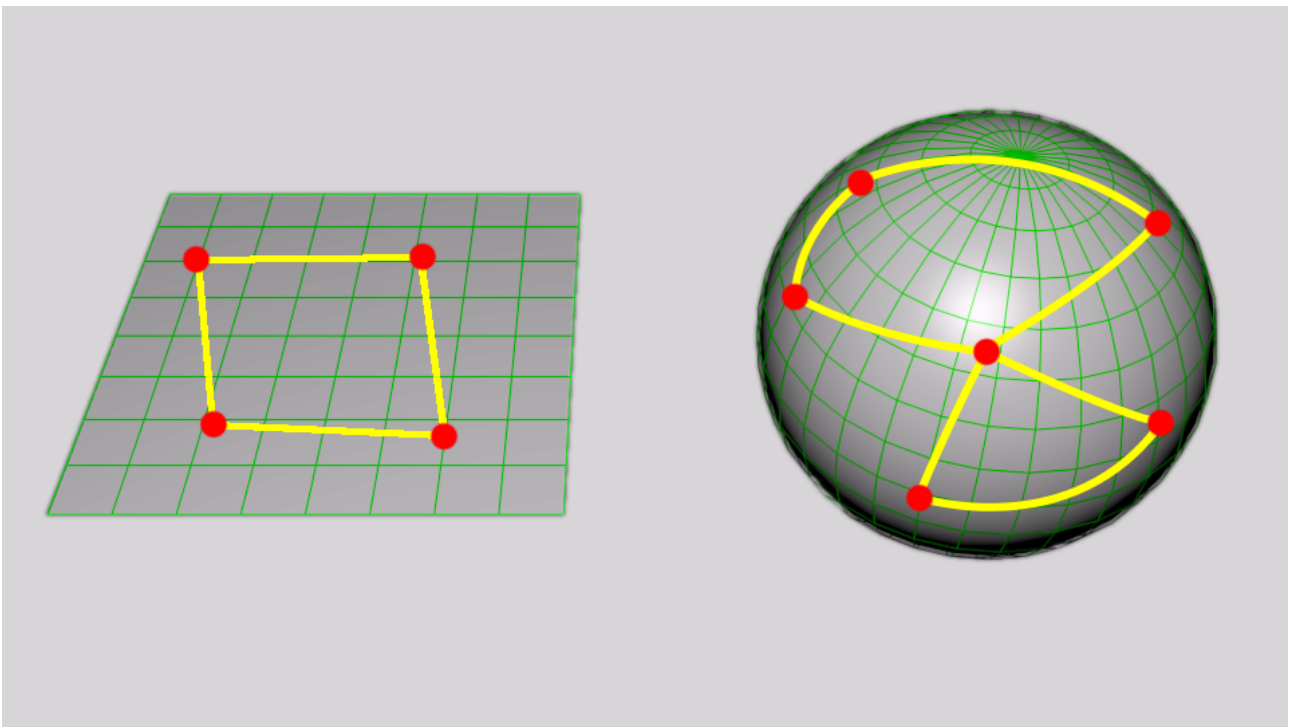


Рисунок 1.1 — Пример планарных графов, уложенных на поверхность с 2-метрикой

#### 1.2.1 Формула Эйлера

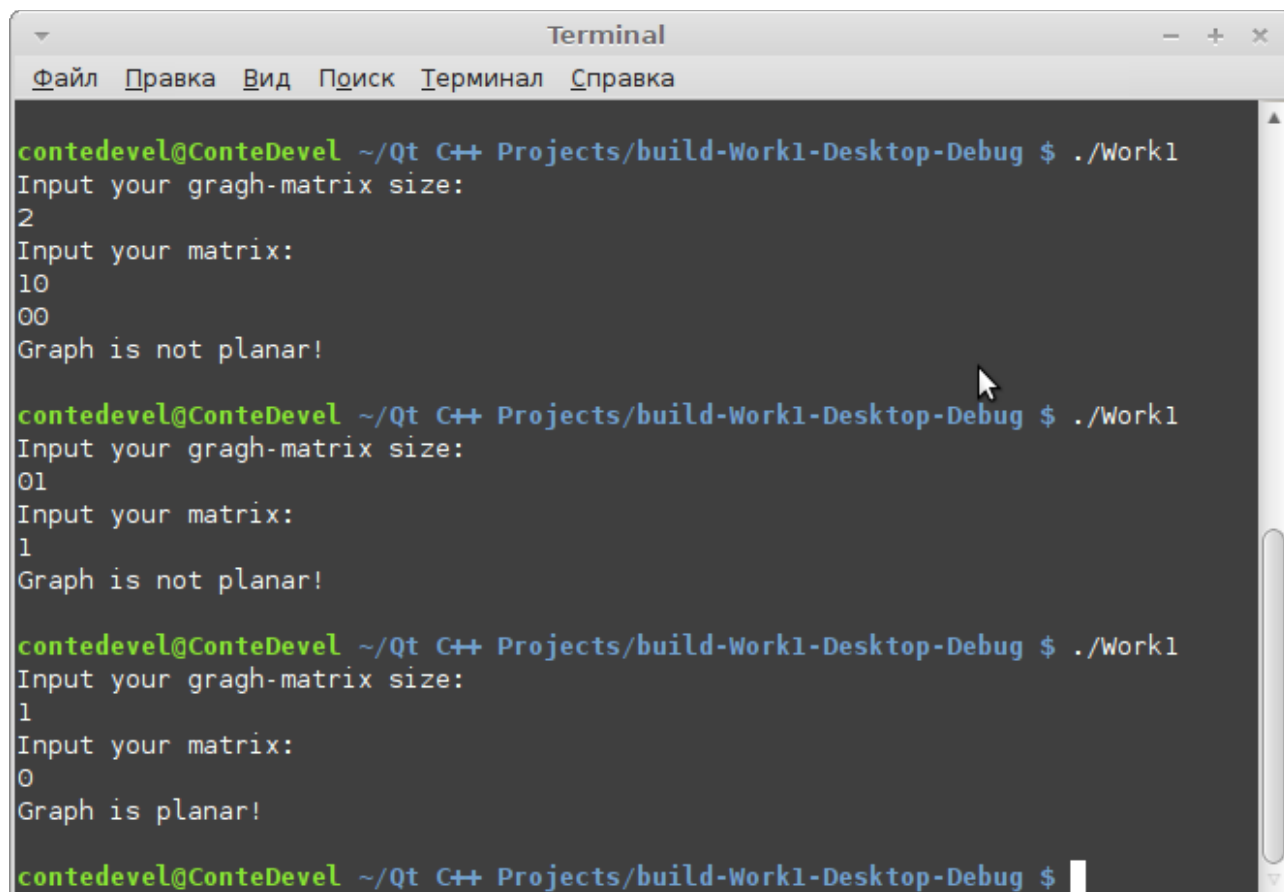
$|V(G)| - |E(G)| + |F(G)| = 2$ , где  $|V(G)|$  - количество вершин,  $|E(G)|$  - количество рёбер,  $|F(G)|$  - количество граней.

Для проверки графа на планарность мы будем пользоваться одним из ее

следствий:  $|E(G)| \leq 3|V(G)| - 6$ , так как данное неравенство позволяет сравнительно просто выполнить проверку графа по его матрице смежности, но накладывает ограничение на минимальный размер графа (наличие хотя бы одной внутренней грани).

### 1.3 Описание программы 1

Программа имеет консольный интерфейс (см. рисунок 1.2) и работает под управлением системы Linux. На вход подается размер графа и его матрица смежности, на выходе — результат проверки.



```
contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $ ./Work1
Input your graph-matrix size:
2
Input your matrix:
10
00
Graph is not planar!

contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $ ./Work1
Input your graph-matrix size:
01
Input your matrix:
1
Graph is not planar!

contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $ ./Work1
Input your graph-matrix size:
1
Input your matrix:
0
Graph is planar!

contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $
```

Рисунок 1.2 — Интерфейс программы 1

## 1.4 Результаты тестирования

```
contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $ ./Work1
Input your graph-matrix size:
1
Input your matrix:
0
Graph is planar!

contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $ ./Work1
Input your graph-matrix size:
1
Input your matrix:
1
Graph is not planar!
```

Рисунок 1.3 — Результат 1

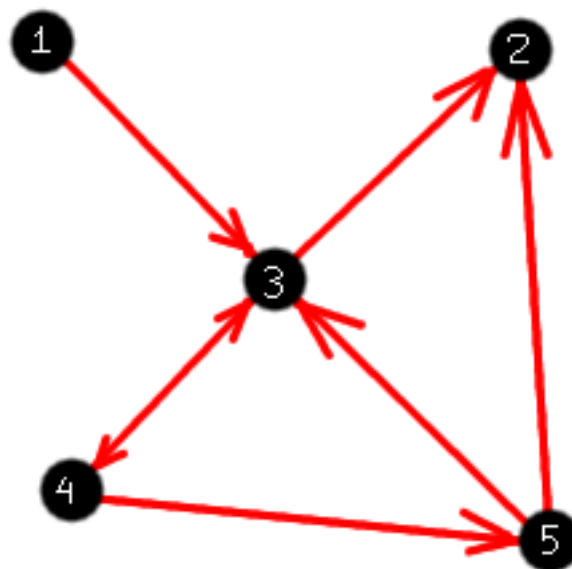
```
Terminal
Файл Правка Вид Поиск Терминал Справка

contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $ ./Work1
Input your graph-matrix size:
2
Input your matrix:
01
10
Graph is planar!

contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $ ./Work1
Input your graph-matrix size:
2
Input your matrix:
11
10
Graph is not planar!
```

Рисунок 1.3 — Результат 2

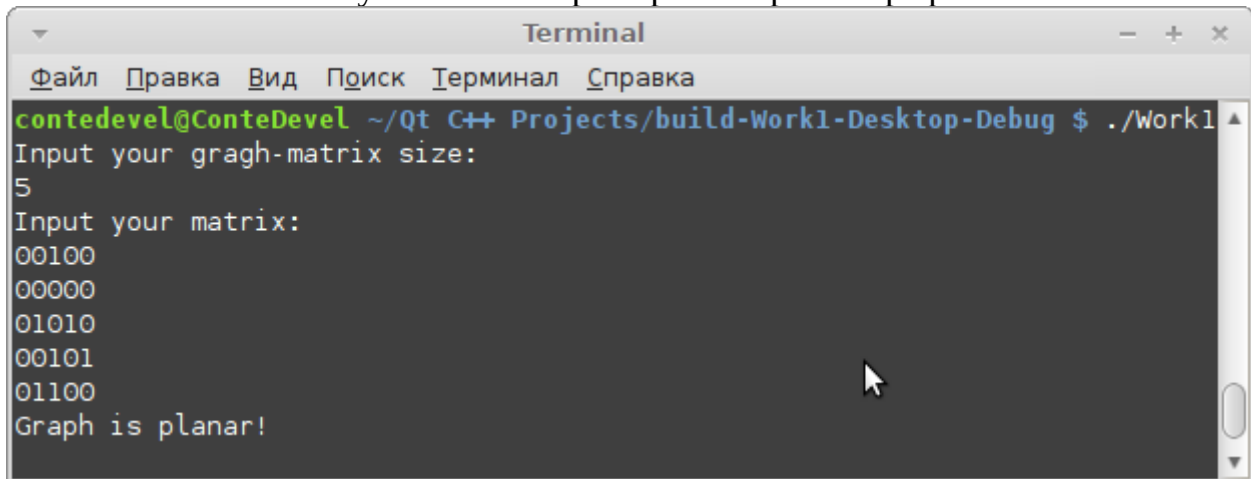
Пример  
5-ю вершинами



планарного графа с  
(см. рисунок 1.4):



Рисунок 1.4 — Пример планарного графа



```
contedevel@ConteDevel ~/Qt C++ Projects/build-Work1-Desktop-Debug $ ./Work1
Input your graph-matrix size:
5
Input your matrix:
00100
00000
01010
00101
01100
Graph is planar!
```

Рисунок 1.5 — Результат 3

## 2 Кратчайшие пути: алгоритм Флойда

### 2.1 Формулировка задания

Изучить тему «Кратчайшие пути: алгоритм Флойда». Исследовать алгоритм Флойда для нахождения кратчайших путей (если таковы имеются) между всеми вершинами графа. Исследовать алгоритм укладки графа на плоскость. Разработать программу демонстрирующую работу алгоритма, провести тестирование.

Для выполнения задания были предложены рекомендованные источники для поиска информации.

### 2.2 Матрица кратчайших путей

**Матрица кратчайших путей** — это матрица, у которой по строкам и столбцам расположены значения кратчайших путей, т. е. элемент  $a[i][j]$  означает, что  $a$  содержит кратчайшее расстояние от вершины  $i$  до вершины  $j$ .

#### 2.2.1 Алгоритм Флойда

**Алгоритм Флойда-Уоршелла** — это динамический алгоритм для нахождения кратчайших расстояний между всеми вершинами взвешенного ориентированного графа.

Пусть у нас имеется взвешенный граф  $G(V, E)$ , где  $|V| = n$  и все вершины пронумерованы. Обозначим кратчайшее расстояние из вершины  $i$  в вершину  $j$  через  $d_{ij}^k$ . Очевидно, что  $d_{jk}^0$  — вес ребра (если ребро отсутствует, то необходимо, чтобы  $d_{jk}^0 = \infty$ ), отсюда следует, что чтобы найти кратчайшие расстояния между всеми вершинами графа необходимо сравнить вес ребра с возможными «обходными путями» и выбрать наименьшее из них. Для этого в алгоритме Флойда используется три вложенных цикла:

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
       $W[i][j] = \min(W[i][j], W[i][k] + W[k][j])$ 
```

То есть мы выбираем  $d_{ij}^k = \min(d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1})$ , где  $k = 1, \dots, n$ , повторяя данную операцию для каждой ячейки матрицы, рекурсивно.

После выполнения цикла на выход мы получим искомую матрицу кратчайших расстояний.

Данный алгоритм сравнительно прост в реализации и, в отличие, от алгоритма Дейкстры, позволяет работать с отрицательными весами.

К недостаткам алгоритма можно отнести то, что алгоритм не умеет работать с отрицательными циклами в графе и с петлями (хотя это исправляется его модернизацией).

## 2.3 Описание программы 2

Программа имеет консольный интерфейс (см. рисунок 2.1) и работает под управлением системы Linux. На вход подается размер графа и его матрица смежности с весами ребер, на выходе — матрица кратчайших расстояний (x — означает отсутствие ребра). Для указания отсутствия ребра между вершинами можно указывать любое число, которое будет являться наибольшим в матрице.

```
Terminal
Файл Правка Вид Поиск Терминал Справка
contedevel@ConteDevel ~/Qt C++ Projects/build-Work2-Desktop-Debug $ ./Work2
Input matrix size:
2
Input matrix:
Input 1 1
03
Input 1 2
2
Input 2 1
1
Input 2 2
0
Path:
      1      2
-----
1|      x      2
2|      1      x
contedevel@ConteDevel ~/Qt C++ Projects/build-Work2-Desktop-Debug $
```

Рисунок 2.1 — Интерфейс программы

## 2.4 Результаты тестирования:

Рассмотрим граф из рисунка 1.4, но присвоим вес его ребрам (см. рисунок 2.2).

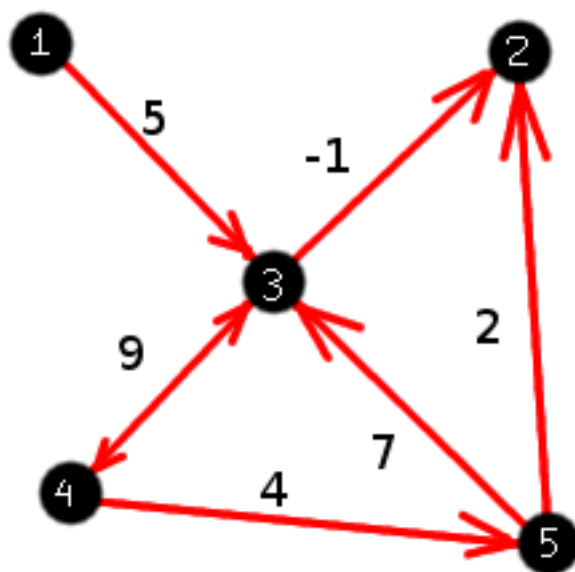
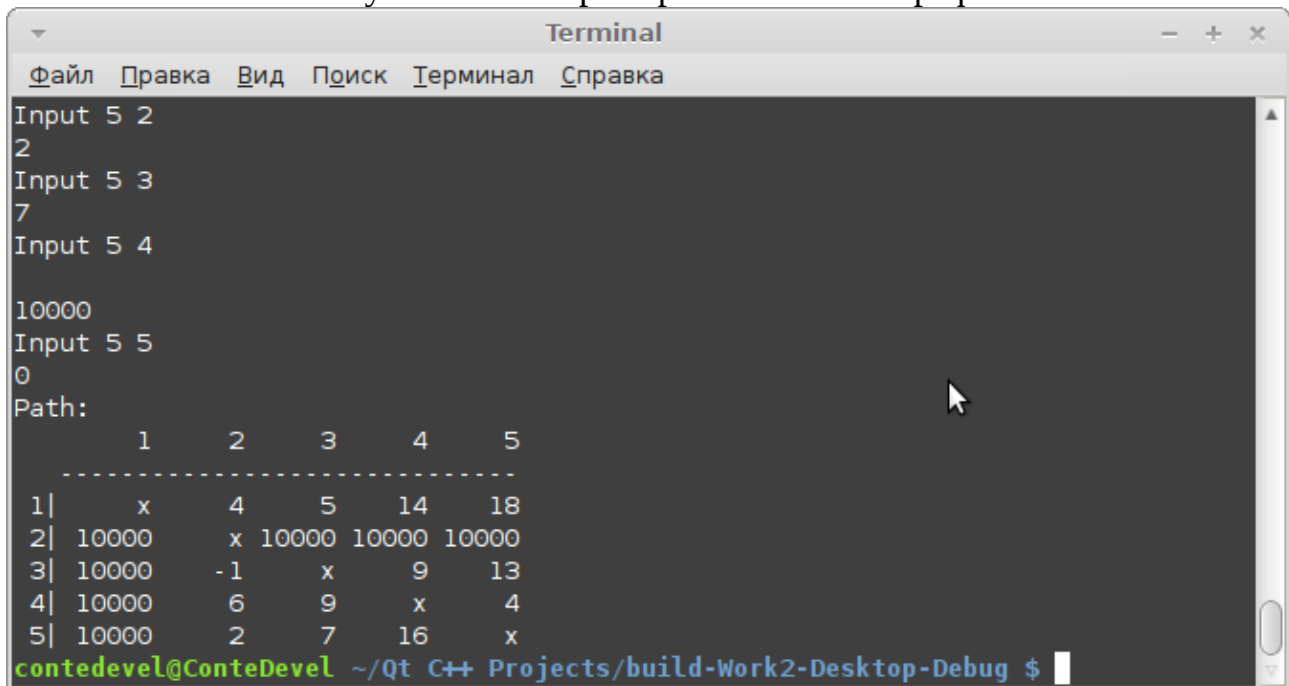


Рисунок 2.2 — Пример взвешенного графа



```
Terminal
Файл  Правка  Вид  Поиск  Терминал  Справка
Input 5 2
2
Input 5 3
7
Input 5 4
10000
Input 5 5
0
Path:
      1      2      3      4      5
-----
1|      x      4      5     14     18
2| 10000      x 10000 10000 10000
3| 10000     -1      x      9     13
4| 10000      6      9      x      4
5| 10000      2      7     16      x
contedevel@ConteDevel ~/Qt C++ Projects/build-Work2-Desktop-Debug $
```

Рисунок 2.3 — Результат выполнения

### **Список использованных источников**

- 1) Скиена С. Алгоритмы. Руководство по разработке. – 2-е изд.: пер. с англ. – СПб.: БХВ-Петербург, 2011. – 720 с. Скиена «Алгоритмы».
- 2) Седжвик Р. Алгоритмы на C++. – Пер. с англ. – М.: Вильямс, 2011. – 1056 с.
- 3) Окулов С.М. Дискретная математика. Теория и практика решения задач по информатике: учебное пособие. – М.: БИНОМ. Лаборатория знаний, 2008. – 422 с.

## Приложение А

```
#include <stdio.h>
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    printf("Input your graph-matrix size:\n");
    int n;
    cin >> n;

    printf("Input your matrix:\n");

    vector<vector<bool> > m(n, vector<bool>(n, false));

    for(int i = 0; i < n; i++) {
        getchar();
        for(int j = 0; j < n; j++) {
            char tmp = getchar();
            if(tmp != '0') {
                m[i][j] = true;
            }
        }
    }

    if(n < 3) {
        switch(n) {
            case 1:
                if(m[0][0]) {
                    cout << "Graph is not planar!\n" << endl;
                    return 0;
                }
                cout << "Graph is planar!\n" << endl;
                return 0;
            case 2:
                if((m[0][1] || m[1][0]) && (!m[0][0] && !m[1][1])) {
                    cout << "Graph is planar!\n" << endl;
                    return 0;
                }
                else {
```

```

        cout << "Graph is not planar!\n" << endl;
        return 0;
    }
}

int e = 0;

bool contin = true;

for(int i = 0; i < n; i++) {
    if(!contin) {
        break;
    }

    for(int j = i; j < n; j++) {
        if((i == j) && (m[i][j])) {
            cout << "Graph is not planar!\n" << endl;
            return 0;
        }

        if(m[i][j] || m[j][i]) {
            e++;
        }
    }
}

if(e <= (3*n - 6)) {
    cout << "Graph is planar!\n" << endl;
}

return 0;
}

```

## Приложение Б

```
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
int main()
{
    cout << "ConteDevel Corporation\n\nInput matrix size:\n";
    int n;
    cin >> n;
    vector <vector <long int> > m(n, vector <long int> (n, 0));
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            cout << "Input " << i + 1 << " -> " << j + 1 << endl;
            cin >> m[i][j];
            if(m[i][j] == 0) {
                m[i][j] = 1073741823;
            }
        }
    }
    for(int k = 0; k < n; k++)
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++) {
                if(m[i][j] > (m[i][k] + m[k][j])) {
                    m[i][j] = m[i][k] + m[k][j];
                }
            }
    cout << "Path:" << endl;
    cout << " ";
    for(int i = 0; i < n; i++) {
        cout << setw(6) << i+1;
    }
    cout << endl << " ";
    for(int i = 0; i < n; i++) {
        cout << "-----";
    }
    cout << endl;
    for (int i = 0; i < n; i++) {
        cout << setw(2) << i+1 << "|";
        for (int j = 0; j < n; j++) {
            if(i == j || m[i][j] > 1003741823) {
                cout << setw(6) << "x";
            }
        }
    }
}
```



```
        continue;
    }
    cout << setw(6) << m[i][j];
}
cout << endl;
}
return 0;
}
```