|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Робототехники и комплексной автоматизации

КАФЕДРА Системы автоматизированного проектирования (РК-6)

**ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

Студент Никатов Владислав Алексеевич

Группа РК6-84б

Тип задания Лабораторная работа

Тема лабораторной работы Решение задачи размещения

Вариант 7

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Никатов В.А. \_**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Берчун Ю.В. \_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2020 г.*

**Оглавление**

[1 Задание 3](#_Toc41071635)

[2 Выбор наилучшего варианта платы 4](#_Toc41071636)

[3 Описание решения с выбранным размером платы 4](#_Toc41071637)

[3.1 Последовательный алгоритм размещения 4](#_Toc41071638)

[3.2 Итерационный алгоритм размещения 9](#_Toc41071639)

[Приложение 1 16](#_Toc41071640)

# 1 Задание

Решить задачу размещения. Рассматриваем платы 3х10 и 5х6. В отчёт включаем пошаговое описание решения для лучшего варианта платы.

Таблица 1.1. Матрица смежности для графа из условия.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 3 | 2 | 0 | 0 | 4 | 1 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 1 | 5 | 0 | 0 | 0 | 0 | 2 | 1 | 3 | 5 | 5 | 2 | 0 | 2 |
| 1 | 0 | 0 | 1 | 3 | 0 | 0 | 3 | 0 | 1 | 0 | 2 | 1 | 0 | 3 | 0 | 0 | 3 | 0 | 0 | 2 | 3 | 0 | 3 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 3 | 2 | 2 | 0 | 0 | 4 | 0 | 0 | 0 | 1 | 2 | 1 | 2 | 2 | 0 | 4 | 1 | 0 | 0 | 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 3 | 0 | 4 | 0 | 0 | 0 | 0 | 4 | 1 | 2 | 0 | 0 | 0 | 4 | 0 | 4 | 1 | 4 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 4 |
| 2 | 3 | 2 | 4 | 0 | 4 | 3 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 2 | 0 | 2 | 0 | 0 | 0 | 3 | 2 | 3 | 0 | 3 | 0 | 0 | 0 |
| 0 | 0 | 2 | 0 | 4 | 0 | 0 | 1 | 0 | 0 | 1 | 3 | 1 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 2 | 2 | 4 | 1 |
| 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 3 | 0 | 1 | 3 | 3 | 2 | 4 | 0 | 0 | 3 | 2 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 2 | 2 | 0 |
| 4 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 4 | 0 | 2 | 2 | 0 | 0 | 4 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 4 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 3 | 0 | 0 | 1 | 0 | 2 | 2 | 0 | 0 | 1 | 2 | 3 | 2 | 0 | 0 | 3 |
| 0 | 1 | 0 | 4 | 0 | 0 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 4 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 1 | 1 | 0 | 2 | 1 | 3 | 1 | 2 | 0 | 0 |
| 3 | 2 | 0 | 2 | 0 | 3 | 1 | 2 | 1 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 3 | 0 | 4 | 0 | 3 | 3 | 0 | 3 | 0 | 0 | 0 | 3 | 0 | 4 |
| 0 | 1 | 1 | 0 | 0 | 1 | 3 | 2 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 2 | 3 | 0 | 2 | 3 | 0 | 0 | 0 | 0 | 4 | 0 |
| 0 | 0 | 2 | 0 | 0 | 3 | 3 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 4 | 3 | 1 | 3 | 0 | 0 | 1 | 1 | 0 | 0 | 4 | 0 | 0 | 2 | 3 | 4 |
| 0 | 3 | 1 | 0 | 2 | 0 | 2 | 0 | 3 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 |
| 0 | 0 | 2 | 4 | 1 | 0 | 4 | 4 | 0 | 1 | 0 | 0 | 3 | 3 | 0 | 0 | 1 | 0 | 0 | 2 | 2 | 0 | 0 | 3 | 1 | 0 | 0 | 0 | 2 | 0 |
| 1 | 0 | 2 | 0 | 2 | 0 | 0 | 3 | 0 | 0 | 2 | 3 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 3 | 1 | 2 | 1 | 0 | 1 |
| 5 | 3 | 0 | 4 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 3 | 0 | 2 | 2 | 0 | 4 | 3 | 0 | 0 |
| 0 | 0 | 4 | 1 | 2 | 0 | 3 | 0 | 0 | 4 | 1 | 4 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 4 | 1 | 1 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 4 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 1 | 0 | 1 | 0 | 0 |
| 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 3 | 3 | 1 | 0 | 2 | 0 | 0 | 4 | 0 | 0 | 4 | 0 | 0 | 2 | 0 | 4 | 0 | 0 | 0 |
| 0 | 3 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 1 | 3 | 0 | 1 | 0 | 0 | 0 | 3 | 1 | 0 | 4 | 0 | 0 | 3 | 3 | 4 | 4 | 1 | 0 | 3 |
| 2 | 0 | 3 | 0 | 3 | 0 | 1 | 0 | 0 | 4 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 2 | 4 | 0 |
| 1 | 3 | 1 | 0 | 2 | 0 | 1 | 0 | 1 | 0 | 2 | 3 | 3 | 0 | 0 | 3 | 0 | 2 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 3 | 0 | 0 | 0 | 3 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 0 | 4 | 0 | 1 | 3 | 2 | 2 | 4 | 2 | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 4 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 4 | 0 | 0 | 1 | 0 | 0 | 4 | 1 | 0 |
| 5 | 1 | 0 | 2 | 3 | 2 | 0 | 0 | 2 | 0 | 1 | 0 | 0 | 0 | 4 | 0 | 2 | 4 | 0 | 0 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| 2 | 0 | 0 | 2 | 0 | 2 | 2 | 0 | 0 | 0 | 2 | 3 | 0 | 2 | 0 | 0 | 1 | 3 | 0 | 1 | 0 | 1 | 2 | 0 | 0 | 4 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 2 | 0 | 4 | 2 | 1 | 0 | 2 | 0 | 0 | 4 | 3 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 2 | 2 | 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 4 | 0 | 1 | 0 | 1 | 3 | 0 | 0 | 4 | 0 | 4 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 4 | 0 | 2 | 0 | 0 | 0 |

# 2 Выбор наилучшего варианта платы

В рамках задачи размещения необходимо минимизировать длину связей – размещать элементы, имеющие общие цепи, как можно ближе друг к другу.

Критерием оптимальности в данной задаче является суммарная длина связей. Ее можно записать в виде формулы

где – расстояние между вершинами, которое будем считать как «манхэттенское расстояние», согласно которому, в двумерной плоскости платы

.

Результаты решения задачи для рассматриваемых размеров платы представлены в таблице 1.2.

Таблица 2.1. Результаты решения задачи для различных размеров платы.

|  |  |
| --- | --- |
| Размер платы | Q |
| 3x10 | 1184 |
| 5x6 | 1076 |

Из таблицы 2.1 видно, что лучшее решение было получено с использованием платы 5x6.

# 3 Описание решения с выбранным размером платы

## 3.1 Последовательный алгоритм размещения

Последовательный алгоритм прост в реализации и используется для получения удовлетворительного размещения. Полученное решение будет использоваться в качестве опорного для итерационного алгоритма размещения.

Сначала необходимо получить опорное решение с помощью последовательного алгоритма, чтобы потом его улучшить с помощью итерационного.

Первым шагом необходимо вычислить суммарное количество связей для каждой из вершин графа. Эти значения представлены в таблице 3.1.2.

Таблица 3.1.2а. Суммарное количество связей для первой половины вершин.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|  | 20 | 31 | 19 | 31 | 18 | 24 | 32 | 33 | 26 | 18 | 30 | 13 | 32 | 33 | 20 |

Таблица 3.1.2б. Суммарное количество связей для второй половины вершин.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 |
|  | 26 | 25 | 23 | 33 | 29 | 14 | 27 | 23 | 20 | 26 | 34 | 25 | 23 | 28 | 24 |

Выбираем вершину с наименьшим числом внешних связей () и размещаем ее на плате. Если для нескольких вершин одинаково, то следует выбирать вершину, которая имеет связи с минимальным числом других вершин. Если и таких вершин несколько – любую из них. Полученное размещение представлено в таблице 3.1.3. Размещенная вершина подсвечена красным цветом, незанятый позиции обозначены символом «х».

Таблица 3.1.3. Размещение, итерация 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | х | х | х | х | х |
| х | х | х | х | х | х |
| х | х | х | х | х | х |
| х | х | х | х | х | х |
| х | х | х | х | х | х |

Рассмотрим вершины {}, имеющие общие связи размещенной вершиной . Приращение связей при размещении вершины считается по формуле где – количество связей с уже размещенными, а сам называют коэффициентом внешней связности вершины. Его значение равно разности между числом внешних (с вершинами, не входящими в соответствующее подмножество) связей вершины и внутренних (с вершинами подмножества, в котором находится рассматриваемая вершина). связей вершины.

Значения для вершин-кандидатов {} представлены в таблице 3.1.4.

Таблица 3.1.4. Приращение связей для вершин-кандидатов, итерация 2.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 5 | 6 | 16 | 20 | 22 | 24 |
|  | -16 | -28 | -23 | -12 | -21 | -18 |

Из них необходимо выбрать ту, при размещении которой будет минимальное приращение количеств связей. Таким образом, нам необходимо разместить вершину на плате. Размещать вершину необходимо в позицию, на которой суммарная длина связей вершины будет минимальна

Считать суммарную длину связей размещаемой вершины имеет смысл лишь в позициях, расположенных в непосредственной близости к занятым. Эти позиции имеют координаты [0;1] и [0;0] и в таблице 3.1.5 подсвечены серым цветом.

Таблица 3.1.5. Позиции-кандидаты на размещение, итерация 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | 1 | х | х | х | х |
| 1 | х | х | х | х | х |
| х | х | х | х | х | х |
| х | х | х | х | х | х |
| х | х | х | х | х | х |

В качестве значений ячеек позиций-кандидатов указана суммарная длина связей размещаемой вершины в этих позициях. В данном случае в обеих позициях длина связей равна, поэтому размесить вершину можно в любой из них.

Повторяем действия. Находим неразмещенные вершины, имеющую общую связь с размещенными, а так же приращение связей при их размещении, представленное в таблице 3.1.6.

Таблица 3.1.6. Приращение связей для вершин-кандидатов, итерация 2.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 5 | 6 | 12 | 13 | 15 | 16 | 18 | 19 | 22 | 24 |
|  | -16 | -24 | -28 | -25 | -24 | -23 | -27 | -27 | -21 | -18 |

Выбираем вершину с наибольшим по модулю приращение связей. Места на плате, куда ее можно разместить проиллюстрированы в таблице 3.1.7 вместе с указанием длины связей на этих местах.

Таблица 3.1.7. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | 8 | х | х | х |
| 4 | 8 | х | х | х | х |
| х | х | х | х | х | х |
| х | х | х | х | х | х |
| х | х | х | х | х | х |

Повторяем действия. В таблице 3.1.8 представлены приращения связей при размещении неразмещенных вершин, имеющих связи с уже размещенными.

Таблица 3.1.8. Приращение связей для вершин-кандидатов, итерация 3.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 3 | 4 | 6 | 10 | 12 | 13 | 15 | 16 | 17 | 18 | 19 | 22 | 24 | 25 | 29 |
|  | -25 | -14 | -24 | -26 | -28 | -25 | -22 | -21 | -19 | -25 | -23 | -21 | -14 | -32 | -18 |

Выбираем вершину с наибольшим по модулю приращение связей. Места на плате, куда ее можно разместить проиллюстрированы в таблице 3.1.9 вместе с указанием длины связей на этих местах.

Таблица 3.1.9. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | 6 | х | х | х |
| **5** | 2 | х | х | х | х |
| 2 | х | х | х | х | х |
| х | х | х | х | х | х |
| х | х | х | х | х | х |

Следующей вершиной, по тому же принципу, размещается :

Таблица 3.1.10. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | 22 | х | х | х |
| **5** | 10 | х | х | х | х |
| **4** | 8 | х | х | х | х |
| 8 | х | х | х | х | х |
| х | х | х | х | х | х |

Таблица 3.1.11. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | 41 | х | х | х |
| **5** | 21 | х | х | х | х |
| **4** | 17 | х | х | х | х |
| **17** | 19 | х | х | х | х |
| 19 | х | х | х | х | х |

Таблица 3.1.12. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | 27 | х | х | х |
| **5** | 15 | х | х | х | х |
| **4** | **3** | 19 | х | х | х |
| **17** | 13 | x | х | х | х |
| 15 | x | х | х | х | х |

Таблица 3.1.13. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | 29 | х | х | х |
| **5** | 15 | х | х | х | х |
| **4** | **3** | 15 | х | х | х |
| **17** | **15** | 12 | х | х | х |
| 13 | 12 | х | х | х | х |

Таблица 3.1.14. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | 25 | х | х | х |
| **5** | 19 | х | х | х | х |
| **4** | **3** | 27 | х | х | х |
| **17** | **15** | **27** | 39 | х | х |
| 29 | 34 | 39 | х | х | х |

Таблица 3.1.15. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | 27 | х | х | х |
| **5** | **24** | 18 | х | х | х |
| **4** | **3** | 17 | х | х | х |
| **17** | **15** | **27** | 25 | х | х |
| 27 | 24 | 25 | х | х | х |

Таблица 3.1.16. Позиции-кандидаты на размещение для вершины .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | х | х | х | х |
| **5** | **24** | х | х | х | х |
| **4** | **3** | **29** | х | х | х |
| **17** | **15** | **27** | х | х | х |
| х | х | х | х | х | х |

Продолжая выполнение алгоритма, пока все вершины не будут размещены на плате, в результате получим размещение, представленное в таблице 3.1.17.

Таблица 3.1.17. Размещение вершин в результате работы алгоритма.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | **1** | **21** | **28** | **0** |
| **5** | **24** | **22** | **2** | **16** | **13** |
| **4** | **3** | **29** | **8** | **25** | **18** |
| **17** | **15** | **27** | **26** | **12** | **9** |
| **23** | **19** | **10** | **14** | **7** | **6** |

Суммарная длина связей для данного размещения .

## 3.2 Итерационный алгоритм размещения

Для улучшения размещения использовался итерационный алгоритм направленного поиска. В данном алгоритме предварительно оценивается качество размещения каждой из вершин. Для оценки качества размещения вершины вычисляется ее средняя длина связей

Использовать суммарную длину нельзя, так как различны локальные степени вершин. Значение каждой вершины представлены в таблице 3.2.1 в подстрочном виде.

Таблица 3.2.1. Значение средней длины для каждой размещенной вершины.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1139** | **2074** | **185** | **2195** | **2887** | **075** |
| **563** | **2461** | **2254** | **245** | **1664** | **13109** |
| **441** | **387** | **2965** | **857** | **25107** | **18113** |
| **1754** | **1579** | **2749** | **2657** | **1279** | **954** |
| **23103** | **19114** | **10105** | **1449** | **7105** | **6139** |

Из таблицы 3.2.1 видно, что наихудшим образом расположена вершина с самой большой средней длиной связи – вершина .

Затем для вершины определяется ее оптимальные координаты – координаты точки, в которой суммарная, а значит, и средняя длина связей вершины будет минимальна.

Для вершины оптимальными координатами, иначе говоря «центром масс», являются координаты .

После чего определяются вершины, попадающие в некоторую заранее заданную область вокруг полученных координат. Вершины-кандидаты на перестановку, попадающие в область вокруг «центра масс», в таблице 3.2.2 отмечены серым цветом, а сама вершина , участвующая в перестановке – зеленым.

Таблица 3.2.2. Вершины-кандидаты на перестановку с вершиной .

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | **1** | **21** | **28** | **0** |
| **5** | **24** | **22** | **2** | **16** | **13** |
| **4** | **3** | **29** | **8** | **25** | **18** |
| **17** | **15** | **27** | **26** | **12** | **9** |
| **23** | **19** | **10** | **14** | **7** | **6** |

Выполняются пробные перестановки полученных вершин из указанной области с вершиной . Для каждой из них рассчитывается приращение суммарной длины связи – значение, на которое изменится суммарная длина связи при осуществлении перестановки. Приращение суммарной длины связи отображено на схеме размещения в таблице 3.2.3 в подстрочном виде.

Таблица 3.2.3. Значение приращения вершин-кандидатов на перестановку.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | **1** | **21** | **28** | **0** |
| **5** | **2480** | **2238** | **2-3** | **16** | **13** |
| **4** | **340** | **2918** | **846** | **25** | **18** |
| **17** | **1542** | **2746** | **2617** | **12** | **9** |
| **23** | **19** | **10** | **14** | **7** | **6** |

Закрепляется наилучшая перестановка, которой будет отрицательное и наибольшее по модулю. В данном случае только одна перестановка уменьшит значение суммарной длины связи – перестановка и . Размещение после перестановки представлено в таблице 3.2.4.

Таблица 3.2.4. Размещение после перестановки и

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | **1** | **21** | **28** | **0** |
| **5** | **24** | **22** | **6** | **16** | **13** |
| **4** | **3** | **29** | **8** | **25** | **18** |
| **17** | **15** | **27** | **26** | **12** | **9** |
| **23** | **19** | **10** | **14** | **7** | **2** |

Алгоритм повторяется до тех пор, пока для вершины, расположенной наихудшим образом (имеющей максимальное значение средней длины связи, ), есть хотя бы одна перестановка, дающая положительный результат (приращение суммарной длины связи которой отрицательно, ).

Таблица 3.2.5. Значение средней длины для каждой размещенной вершины.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1129** | **2064** | **185** | **2195** | **2878** | **070** |
| **563** | **2461** | **2254** | **6113** | **1670** | **13107** |
| **438** | **387** | **2971** | **866** | **25107** | **18113** |
| **1754** | **1576** | **2749** | **2660** | **1279** | **957** |
| **23106** | **19115** | **10102** | **1448** | **7117** | **268** |

Наихудшим образом расположена вершина Ее центр масс находится в точке с координатами . Точки, расположенные вокруг ее центра масс и приращение при перестановке с ними представлены в таблице 3.2.6.

Таблица 3.2.6. Значение приращения вершин-кандидатов на перестановку с.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | **1** | **21** | **28** | **0** |
| **5** | **2476** | **2224** | **618** | **16** | **13** |
| **4** | **348** | **2910** | **8-5** | **25** | **18** |
| **17** | **1530** | **2726** | **26-16** | **12** | **9** |
| **23** | **19** | **10** | **14** | **7** | **2** |

Вершина с наибольшим по модулю отрицательным приращением . Закрепляем перестановку вершин и и снова считаем для каждой вершины.

Таблица 3.2.7. Значение средней длины для каждой размещенной вершины.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1129** | **2064** | **185** | **2191** | **2878** | **070** |
| **563** | **2463** | **2260** | **6105** | **1670** | **13107** |
| **438** | **383** | **2965** | **862** | **25107** | **18113** |
| **1754** | **1584** | **2749** | **791** | **1279** | **957** |
| **23106** | **19115** | **10102** | **1448** | **2670** | **262** |

Наихудшим образом расположена вершина Ее центр масс находится в точке с координатами . Точки, расположенные вокруг ее центра масс и приращение при перестановке с ними представлены в таблице 3.2.8.

Таблица 3.2.8. Значение приращения вершин-кандидатов на перестановку с

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20** | **1** | **21** | **28** | **0** |
| **5** | **2439** | **222** | **635** | **16** | **13** |
| **4** | **316** | **291** | **8-6** | **25** | **18** |
| **17** | **151** | **270** | **724** | **12** | **9** |
| **23** | **19** | **10** | **14** | **26** | **2** |

Вершины для перестановки и .

Таблица 3.2.9. Значение средней длины для каждой размещенной вершины.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1129** | **2064** | **189** | **2191** | **2878** | **070** |
| **563** | **2463** | **2269** | **697** | **1678** | **1391** |
| **438** | **383** | **2965** | **1979** | **25115** | **18101** |
| **1754** | **1584** | **2749** | **791** | **1279** | **957** |
| **23106** | **8115** | **10102** | **1448** | **2670** | **262** |

Наихудшим образом расположена вершина Ее центр масс находится в точке с координатами Точки, расположенные вокруг ее центра масс и приращение при перестановке с ними представлены в таблице 3.2.10.

Таблица 3.2.10. Значение приращения вершин-кандидатов на перестановку с

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **20-28** | **134** | **2117** | **28** | **0** |
| **5** | **2430** | **225** | **66** | **16** | **13** |
| **4** | **317** | **29-14** | **19-7** | **25** | **18** |
| **17** | **15** | **27** | **7** | **12** | **9** |
| **23** | **8** | **10** | **14** | **26** | **2** |

Вершины для перестановки и .

Таблица 3.2.11. Значение средней длины для каждой размещенной вершины.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1134** | **25123** | **177** | **2188** | **2880** | **070** |
| **560** | **2463** | **2266** | **695** | **1678** | **1379** |
| **435** | **380** | **2965** | **1976** | **2028** | **18106** |
| **1754** | **1587** | **2749** | **793** | **1289** | **953** |
| **23102** | **890** | **10102** | **1448** | **2670** | **262** |

Наихудшим образом расположена вершина . Ее центр масс находится в точке с координатами . Точки, расположенные вокруг ее центра масс и приращение при перестановке с ними представлены в таблице 3.2.12.

Таблица 3.2.12. Значение приращения вершин-кандидатов на перестановку с.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **25** | **1** | **21** | **28** | **0** |
| **5** | **24-14** | **22-2** | **628** | **16** | **13** |
| **4** | **36** | **292** | **1940** | **20** | **18** |
| **17** | **154** | **274** | **737** | **12** | **9** |
| **23** | **8** | **10** | **14** | **26** | **2** |

Вершины для перестановки и .

Таблица 3.2.13. Значение средней длины для каждой размещенной вершины.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1130** | **2465** | **177** | **2188** | **2881** | **070** |
| **561** | **25107** | **2270** | **695** | **1678** | **1379** |
| **432** | **377** | **2966** | **1976** | **2028** | **18102** |
| **1753** | **1585** | **2751** | **793** | **1285** | **953** |
| **2398** | **890** | **10102** | **1449** | **2671** | **262** |

Наихудшим образом расположена вершина . Ее центр масс находится в точке с координатами . Точки, расположенные вокруг ее центра масс и приращение при перестановке с ними представлены в таблице 3.2.14.

Таблица 3.2.14. Значение приращения вершин-кандидатов на перестановку с.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **24** | **1** | **21** | **28** | **0** |
| **5** | **250** | **221** | **632** | **16** | **13** |
| **4** | **3-5** | **294** | **1927** | **20** | **18** |
| **17** | **150** | **27-1** | **730** | **12** | **9** |
| **23** | **8** | **10** | **14** | **26** | **2** |

Вершины для перестановки и .

Таблица 3.2.15. Значение средней длины для каждой размещенной вершины.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1130** | **2465** | **181** | **2189** | **2883** | **070** |
| **559** | **383** | **2268** | **695** | **1678** | **1375** |
| **432** | **2596** | **2966** | **1976** | **2028** | **1898** |
| **1755** | **1583** | **2753** | **795** | **1281** | **953** |
| **2394** | **891** | **10105** | **1449** | **2671** | **262** |

Наихудшим образом расположена вершина . Ее центр масс находится в точке с координатами . Точки, расположенные вокруг ее центра масс и приращение при перестановке с ними представлены в таблице 3.2.16.

Таблица 3.2.16. Значение приращения вершин-кандидатов на перестановку с.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **24** | **1** | **21** | **28** | **0** |
| **564** | **330** | **22-1** | **6** | **16** | **13** |
| **448** | **2524** | **29-6** | **19** | **20** | **18** |
| **1751** | **1510** | **27-1** | **7** | **12** | **9** |
| **23** | **8** | **10** | **14** | **26** | **2** |

Вершины для перестановки и .

Таблица 3.2.17. Значение средней длины для каждой размещенной вершины.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **1130** | **2467** | **177** | **2189** | **2883** | **070** |
| **561** | **377** | **2262** | **695** | **1676** | **1375** |
| **432** | **2596** | **1083** | **1976** | **2028** | **1896** |
| **1755** | **1583** | **2753** | **795** | **1281** | **953** |
| **2394** | **891** | **2982** | **1457** | **2671** | **264** |

Наихудшим образом расположена вершина . Ее центр масс находится в точке с координатами . Точки, расположенные вокруг ее центра масс и приращение при перестановке с ними представлены в таблице 3.2.18.

Таблица 3.2.18. Значение приращения вершин-кандидатов на перестановку с.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **2432** | **153** | **2130** | **28** | **0** |
| **5** | **35** | **228** | **636** | **16** | **13** |
| **4** | **250** | **1018** | **1930** | **20** | **18** |
| **17** | **15** | **27** | **7** | **12** | **9** |
| **23** | **8** | **29** | **14** | **26** | **2** |

Видно, что перестановок, приводящих к уменьшению суммарной длины связей больше нет. Итерационный алгоритм на этом завершается, итоговое размещение приедено в таблице 3.2.19.

Таблица 3.2.19. Итоговое размещение.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **11** | **24** | **1** | **21** | **28** | **0** |
| **5** | **3** | **22** | **6** | **16** | **13** |
| **4** | **25** | **10** | **19** | **20** | **18** |
| **17** | **15** | **27** | **7** | **12** | **9** |
| **23** | **8** | **29** | **14** | **26** | **2** |

Суммарная длина связей после всех перестановок .

Код программы, с реализацией последовательного и итерационного алгоритмов размещения, приведен в приложении 1.

# Приложение 1

**main.py**

from print import print\_plate  
from data import matrix  
from sequential\_solver import sequential\_algorithm  
from iterative\_solver import iterative\_algorithm  
from helper import optimal  
  
  
if \_\_name\_\_ == **'\_\_main\_\_'**:  
 plate\_size = (5, 6)  
 first\_plate = sequential\_algorithm(matrix, plate\_size, info=True)  
 second\_plate = iterative\_algorithm(matrix, first\_plate, info=True)  
  
 print\_plate(first\_plate)  
 print(**'Суммарная длина связей:'**, optimal(first\_plate, matrix))  
 print\_plate(second\_plate)  
 print(**'Суммарная длина связей:'**, optimal(second\_plate, matrix))

**sequential\_solver.py**

import numpy as np  
from print import print\_plate  
from helper import distance  
from print import print\_r\_matrix  
  
  
def sequential\_algorithm(matrix, plate\_size, info=False):  
 plate = [[None] \* plate\_size[1] for \_ in range(plate\_size[0])]  
 use\_list = []  
 element = first\_element(matrix, use\_list, info)  
 if info:  
 print(**'Первый элемент:'**, end=**' '**)  
 print(element)  
  
 use\_element(plate, element, (0, 0), use\_list)  
 if info:  
 print(**'Размещение:'**)  
 print\_plate(plate)

for \_ in range(len(matrix) - 1):  
 element = next\_element(use\_list, matrix, use\_list, info)  
 if info:  
 print(**'следующий элемент:'**, end=**' '**)  
 print(element)  
  
 candidate\_list = candidate\_place\_list(plate)  
 if info:  
 print(**'список кандидатов мест на плате:'**, end=**' '**)  
 print(candidate\_list)  
  
 sum\_cost\_list = [sum\_cost(matrix, plate, element, candidate\_place) for candidate\_place in candidate\_list]  
 min\_cost\_ind = np.array(sum\_cost\_list).argmin()  
 place = candidate\_list[min\_cost\_ind]  
 if info:  
 print(**"Значение дельта для выбранных вершин:"**)  
 print(\*candidate\_list, sep=**'**\t**'**)  
 print(\*sum\_cost\_list, sep=**'**\t\t**'**)  
 print(**'Место для размещения элемента:'**, end=**' '**)  
 print(place)  
  
 use\_element(plate, element, place, use\_list)  
 if info:  
 print(**'Размещение с новым элементом: '**)  
 print\_plate(plate)  
 print()  
 return plate  
  
  
def first\_element(matrix, use\_list, info=False):  
 *# создание словарей нераспределенных вершин:  
 # 1. количество связей с ненулевыми неиспользуемыми вершинами  
 # 2. сумма связей с ненулевыми неиспользуемыми вершинами* count\_dict = dict()  
 sum\_dict = dict()  
 *# строки матрицы, соответствующие неиспользуемым вершинам* for elem, line in [(i, line) for i, line in enumerate(matrix) if i not in use\_list]:  
 *# количество ненулевых связей с вершинами, отсутствующими в списке use\_list* count\_dict.update([(elem, len([v for el, v in enumerate(line) if v != 0 and el not in use\_list]))])  
 *# сумма связей с вершинами, отсутствующими в списке use\_list* sum\_dict.update([(elem, sum([v for el, v in enumerate(line) if el not in use\_list]))])  
 if info:  
 print(**'Сумма связей для нераспределенных вершин:'**)  
 print\_r\_matrix([list(sum\_dict.values())], [list(sum\_dict.keys())])  
 *# вершины с минимальной суммой* elem\_list = [i for i, v in sum\_dict.items() if v == min(sum\_dict.values())]  
 *# оставляем только вершины с минимальной суммой* for el in [elem for elem in range(len(matrix)) if elem not in elem\_list and elem not in use\_list]:  
 count\_dict.pop(el)  
 if info:  
 print(**'Вершины с минимальной суммой внешних связей:'**, list(count\_dict.keys()))  
 *# вершина с минимальным количеством связей* value = [el for el in elem\_list if count\_dict[el] == min(count\_dict.values())][0]  
 if info and len(count\_dict) > 1:  
 print(**'Количество внешних связей для них:'**)  
 print\_r\_matrix([list(count\_dict.values())], [list(count\_dict.keys())])  
 print(**'Вершина с минимальным количеством внешних связей:'**, value)  
 return value  
  
  
def next\_element(group, matrix, use\_list, info=False):  
 adjacent\_group = get\_adjacent\_group(group, matrix, use\_list)  
 if info:  
 print(**"Перечень неразмещенных вершин, имеющих общие связи с размещенными:"**, adjacent\_group)  
 if len(adjacent\_group) == 0:  
 return first\_element(matrix, use\_list, info)  
  
 p\_list = list(map(sum, matrix))  
 delta = dict([(i, 0) for i in adjacent\_group])  
 for el1 in adjacent\_group:  
 for el2 in group:  
 delta[el1] += matrix[el2][el1] \* 2  
 delta[el1] -= p\_list[el1]  
 value, delta = [(i, v) for i, v in delta.items() if v == max(delta.values())][0]  
 return value  
  
  
def get\_adjacent\_group(group, matrix, use\_list):  
 adjacent\_group = set()  
 for el1 in group:  
 adjacent\_group.update([i for i, el2 in enumerate(matrix[el1]) if el2 != 0 and i not in use\_list])  
 return list(adjacent\_group)  
def use\_element(plate, element, place, use\_list):  
 use\_list.append(element)  
 plate[place[0]][place[1]] = element  
  
  
def candidate\_place\_list(plate):  
 candidate\_set = set()  
 for i, p\_line in enumerate(plate):  
 for j in [ind for ind, place in enumerate(p\_line) if place is not None]:  
 if 0 < i and plate[i - 1][j] is None:  
 candidate\_set.add((i - 1, j))  
 if i < len(plate) - 1 and plate[i + 1][j] is None:  
 candidate\_set.add((i + 1, j))  
 if 0 < j and plate[i][j - 1] is None:  
 candidate\_set.add((i, j - 1))  
 if j < len(p\_line) - 1 and plate[i][j + 1] is None:  
 candidate\_set.add((i, j + 1))  
 return list(candidate\_set)  
  
  
def sum\_cost(matrix, plate, element, candidate\_place):  
 result = 0  
 for i, p\_line in enumerate(plate):  
 for j in [ind for ind, place in enumerate(p\_line) if place is not None]:  
 result += matrix[element][plate[i][j]] \* distance((i, j), candidate\_place)  
 return result

**iterative\_solver.py**

from helper import distance  
from helper import optimal  
from print import print\_plate  
import numpy as np  
import copy  
  
def iterative\_algorithm(matrix, plate, info=False):  
 plate = copy.deepcopy(plate)  
 if info:  
 print\_plate(plate)  
 while True:  
 *# Вычисляем среднюю длину связей каждой вершины* avg\_len\_list = [sum\_len(matrix, plate, (i, j)) for i in range(len(plate)) for j in range(len(plate[0]))]  
 *# Находим вершину с наибольшим значением средней длины связи* point = [(i, j) for i in range(len(plate)) for j in range(len(plate[0])) if sum\_len(matrix, plate, (i, j)) == max(avg\_len\_list)][0]  
 if info:  
 print(**'Средняя длина связей каждой вершины:'**)  
 for i in range(len(plate)):  
 for j in range(len(plate[i])):  
 print(sum\_len(matrix, plate, (i, j)), end=**'**\t**'**)  
 print()  
 *# Находим ее центр массы* mass\_center\_p = mass\_center(matrix, plate, point)  
 *# Находим вершины-кандидаты для перестановки* candidate\_list = candidate\_place\_list(plate, \*mass\_center\_p)  
 *# Совершаем пробные перестановки с вершинами-кандидатами* delta\_list = get\_delta\_list(matrix, plate, point, candidate\_list)  
 if info:  
 print(**'Наихудшим образом расположенная вершина:'**, plate[point[0]][point[1]])  
 print(**'Ее центр масс связей в точке:'**, mass\_center\_p)  
 print(**'Вершины вблизи ее центра масс, их координаты и величина,'  
 'на которую изменится сумма связей при перестановке:'**)  
 for p, v in zip(candidate\_list, delta\_list):  
 print(plate[p[0]][p[1]], p, v)  
 if min(delta\_list) >= 0:  
 if info:  
 print(**'Перестановок больше нет'**)  
 return plate  
 point\_to\_swap = candidate\_list[np.array(delta\_list).argmin()]  
 if info:  
 print(**'Вершина для перестановки:'**, point\_to\_swap)  
 print(plate[point[0]][point[1]], **' меняются с'**, plate[point\_to\_swap[0]][point\_to\_swap[1]])  
 swap\_element(plate, \*point, \*point\_to\_swap)  
 if info:  
 print(**'Q = '**, optimal(plate, matrix))  
 print(**'Новое размещение: '**)  
 print\_plate(plate)  
  
  
def candidate\_place\_list(plate, ind\_i, ind\_j):  
 place\_list = set()  
 ind\_i = int(ind\_i + (0.5 if ind\_i > 0 else -0.5)) *# округление* ind\_j = int(ind\_j + (0.5 if ind\_j > 0 else -0.5)) *# округление* for i in range(ind\_i - 1, ind\_i + 2):  
 for j in range(ind\_j - 1, ind\_j + 2):  
 *# координаты за пределами платы не рассматриваются* if i < 0 or j < 0 or i >= len(plate) or j >= len(plate[0]):  
 continue  
 place\_list.add((i, j))  
 return list(place\_list)  
  
  
def sum\_len(matrix, plate, point):  
 element = plate[point[0]][point[1]]  
 result = 0  
 for i, p\_line in enumerate(plate):  
 for j in [ind for ind, place in enumerate(p\_line)]:  
 result += matrix[element][plate[i][j]] \* distance((i, j), point)  
 return result  
  
  
def average\_len(matrix, plate, point):  
 element = plate[point[0]][point[1]]  
 return sum\_len(matrix, plate, point) / sum(matrix[element])  
  
  
def mass\_center(matrix, plate, point):  
 element = plate[point[0]][point[1]]  
 mass\_i = 0  
 mass\_j = 0  
 for i, p\_line in enumerate(plate):  
 for j in [ind for ind, place in enumerate(p\_line) if place is not None]:  
 mass\_i += matrix[element][plate[i][j]] \* np.abs(i - point[0])  
 mass\_j += matrix[element][plate[i][j]] \* np.abs(j - point[1])  
 return mass\_i / sum(matrix[element]), mass\_j / sum(matrix[element])  
  
  
def swap\_element(plate, i1, j1, i2, j2):  
 plate[i1][j1], plate[i2][j2] = plate[i2][j2], plate[i1][j1]  
  
  
def get\_delta\_list(matrix, plate, point, candidate\_list):  
 delta\_list = []  
 for candidate in candidate\_list:  
 s1 = sum\_len(matrix, plate, candidate) + sum\_len(matrix, plate, point)  
 swap\_element(plate, \*candidate, \*point)  
 s2 = sum\_len(matrix, plate, candidate) + sum\_len(matrix, plate, point)  
 swap\_element(plate, \*candidate, \*point)  
 delta\_list.append(s2 - s1)  
 return delta\_list

**helper.py**

import numpy as np  
  
  
def concat(ll):  
 return [el for lst in ll for el in lst]  
  
  
def distance(point1, point2):  
 return np.abs(point1[0] - point2[0]) + np.abs(point1[1] - point2[1])  
  
  
def optimal(plate, matrix):  
 result = 0  
 for i, line1 in enumerate(plate):  
 for j, el1 in enumerate(line1):  
 for i2, line2 in enumerate(plate):  
 for j2, el2 in enumerate(line2):  
 if el1 < el2:  
 result += matrix[el1][el2] \* distance((i, j), (i2, j2))  
 return result

**print.py**

from helper import concat  
  
  
def print\_r\_matrix(r\_matrix, group\_list=None):  
 if group\_list is None:  
 group\_list = list([range(len(r\_matrix[0]))])  
 elem\_list = concat(group\_list)  
 *# шапка* print(end=**' R '**)  
 for el in elem\_list:  
 if el < 10:  
 print(end=**' '**)  
 print(el, end=**' '**)  
 print()  
  
 *# столбец + матрица* for i, line in enumerate(r\_matrix):  
 *# столбец* if elem\_list[i] < 10:  
 print(end=**' '**)  
 print(elem\_list[i], end=**''**)  
 *# строка матрицы* for el in line:  
 if el <= -10:  
 print(el, end=**''**)  
 elif el < 0 or el >= 10:  
 print(**''**, el, end=**''**)  
 elif el < 10:  
 print(**' '**, el, end=**''**)  
 print()  
  
  
def print\_p\_matrix(p\_matrix, group1, group2):  
 *# шапка* print(end=**' '**)  
 for el in group2:  
 if el <= -10:  
 print(**''**, el, end=**''**)  
 elif el < 0 or el >= 10:  
 print(**' '**, el, end=**''**)  
 elif el < 10:  
 print(**' '**, el, end=**''**)  
 print()  
  
 *# столбец + матрица* for i, line in enumerate(p\_matrix):  
 *# столбец* if group1[i] < 10:  
 print(end=**' '**)  
 print(group1[i], end=**''**)  
 *# строка матрицы* for el in line:  
 if el <= -10:  
 print(**''**, el, end=**''**)  
 elif el < 0 or el >= 10:  
 print(**' '**, el, end=**''**)  
 elif el < 10:  
 print(**' '**, el, end=**''**)  
 print()  
  
  
def print\_plate(plate):  
 for line in plate:  
 for el in line:  
 if el is None:  
 print(**' X'**, end=**' '**)  
 continue  
 if el < 10:  
 print(end=**' '**)  
 print(el, end=**' '**)  
 print()

**data.py**

*# Вариант 7*matrix = [[0, 0, 0, 0, 0, 0, 5, 3, 0, 0, 0, 0, 5, 4, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 4, 0, 0, 0, 3, 0, 2, 0, 0, 0, 0, 4, 2, 0, 4, 4, 0, 0, 4, 2],  
 [0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 3, 0, 0, 2, 1, 0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 3, 0, 0, 2],  
 [0, 0, 0, 0, 3, 3, 0, 2, 3, 0, 3, 0, 0, 4, 0, 0, 0, 4, 0, 0, 0, 2, 2, 0, 0, 3, 0, 2, 0, 0],  
 [0, 0, 0, 3, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 4, 0, 2, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0],  
 [0, 0, 0, 3, 2, 0, 0, 0, 0, 0, 2, 4, 0, 0, 0, 1, 1, 2, 1, 2, 0, 0, 0, 0, 2, 1, 0, 0, 0, 3],  
 [5, 0, 0, 0, 1, 0, 0, 4, 1, 1, 0, 2, 0, 4, 0, 3, 0, 0, 0, 3, 2, 0, 0, 3, 0, 0, 0, 0, 3, 0],  
 [3, 0, 0, 2, 0, 0, 4, 0, 2, 2, 0, 0, 3, 2, 0, 0, 0, 0, 4, 1, 0, 2, 0, 4, 0, 0, 1, 0, 0, 3],  
 [0, 2, 4, 3, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 3, 1, 2, 2, 0, 4, 0, 0],  
 [0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4],  
 [0, 4, 3, 3, 0, 2, 0, 0, 0, 0, 0, 0, 4, 0, 4, 0, 2, 1, 1, 0, 0, 3, 3, 0, 0, 0, 0, 0, 0, 0],  
 [0, 0, 0, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 4, 0, 0, 0, 0, 0],  
 [5, 0, 0, 0, 0, 0, 0, 3, 0, 0, 4, 0, 0, 0, 3, 0, 4, 1, 0, 0, 2, 0, 1, 0, 0, 4, 4, 1, 0, 0],  
 [4, 0, 2, 4, 0, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 3, 4, 4, 1, 0, 0, 0, 0, 0, 0, 2, 0],  
 [0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 4, 0, 3, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 1, 1, 0, 3, 0, 0, 0],  
 [2, 0, 0, 0, 2, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 2, 2, 1, 1, 0, 0, 0, 2, 4, 2, 0, 1],  
 [1, 2, 2, 0, 0, 1, 0, 0, 2, 1, 2, 1, 4, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 1],  
 [0, 0, 0, 4, 4, 2, 0, 0, 0, 0, 1, 0, 1, 0, 0, 3, 0, 0, 0, 1, 0, 0, 0, 0, 1, 2, 0, 3, 1, 0],  
 [0, 0, 0, 0, 0, 1, 0, 4, 0, 4, 1, 0, 0, 3, 0, 2, 0, 0, 0, 3, 3, 0, 1, 2, 0, 4, 3, 0, 2, 0],  
 [0, 0, 2, 0, 2, 2, 3, 1, 0, 2, 0, 0, 0, 4, 3, 2, 0, 1, 3, 0, 1, 0, 0, 1, 0, 0, 0, 2, 0, 0],  
 [0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1, 2, 4, 0, 1, 0, 0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
 [0, 4, 0, 2, 0, 0, 0, 2, 0, 0, 3, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 0, 0, 2, 3],  
 [0, 2, 0, 2, 1, 0, 0, 0, 3, 0, 3, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 4, 0, 3, 0, 2, 0],  
 [0, 0, 0, 0, 0, 0, 3, 4, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 1, 0, 3, 0, 0, 0, 4, 1, 0, 0, 0],  
 [0, 4, 0, 0, 0, 2, 0, 0, 2, 0, 0, 4, 0, 0, 1, 0, 0, 1, 0, 0, 0, 3, 4, 0, 0, 0, 1, 2, 1, 1],  
 [0, 4, 0, 3, 3, 1, 0, 0, 2, 0, 0, 0, 4, 0, 0, 2, 0, 2, 4, 0, 0, 3, 0, 4, 0, 0, 0, 0, 2, 0],  
 [0, 0, 3, 0, 0, 0, 0, 1, 0, 0, 0, 0, 4, 0, 3, 4, 0, 0, 3, 0, 0, 0, 3, 1, 1, 0, 0, 0, 2, 0],  
 [0, 0, 0, 2, 0, 0, 0, 0, 4, 0, 0, 0, 1, 0, 0, 2, 2, 3, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 1, 4],  
 [0, 4, 0, 0, 0, 0, 3, 0, 0, 4, 0, 0, 0, 2, 0, 0, 2, 1, 2, 0, 0, 2, 2, 0, 1, 2, 2, 1, 0, 0],  
 [0, 2, 2, 0, 0, 3, 0, 3, 0, 4, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 3, 0, 0, 1, 0, 0, 4, 0, 0]]