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

Решить задачу размещения. Рассматриваем платы 3x10 и 5x6. 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	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

$$Q = \sum_{i}^{n-1} \sum_{j=i+1}^{n} d_{ij} r_{ij} ,$$

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

$$d_{ij} = |x_i - x_j| + |y_i - y_i|.$$

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.

v_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
s_i	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.

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

Выбираем вершину с наименьшим числом внешних связей (v_{11}) и размещаем ее на плате. Если для нескольких вершин s_i 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	x	x	x	x	x
x	x	x	x	x	x
x	x	x	x	x	x
x	x	x	x	x	x
x	x	x	x	x	x

Рассмотрим вершины $\{v_5, v_6, v_{16}, v_{20}, v_{22}, v_{24}\}$, имеющие общие связи размещенной вершиной v_{11}. Приращение связей при размещении вершины считается по формуле $\delta_i = 2 \cdot r_i - s_i$, где r_i – количество связей с уже размещенными, а сам δ_i 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 δ_i для вершин-кандидатов $\{v_5, v_6, v_{16}, v_{20}, v_{22}, v_{24}\}$ представлены в таблице 3.1.4.

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

v_i	5	6	16	20	22	24
δ_i	-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 v_{20} на плате. Размещать вершину необходимо в позицию, на которой суммарная длина связей вершины будет минимальна

$$L_i = \sum_{j=1}^{n} d_{ij} r_{ij} .$$

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	x	x	x	x
1	x	x	x	x	x
x	x	x	x	x	x
x	x	x	x	x	x
x	x	x	x	x	x

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 v_{20} 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.

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

Выбираем вершину v_5 с наибольшим по модулю приращение связей. Места на плате, куда ее можно разместить проиллюстрированы в таблице 3.1.7 вместе с указанием длины связей v_4 на этих местах.

Таблица 3.1.7. Позиции-кандидаты на размещение для вершины v_5.

11	**20**	8	x	x	x
4	8	x	x	x	x
x	x	x	x	x	x
x	x	x	x	x	x
x	x	x	x	x	x

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.

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

Выбираем вершину v_4 с наибольшим по модулю приращение связей. Места на плате, куда ее можно разместить проиллюстрированы в таблице 3.1.9 вместе с указанием длины связей v_4 на этих местах.

Таблица 3.1.9. Позиции-кандидаты на размещение для вершины v_4.

11	**20**	6	x	x	x
5	2	x	x	x	x
2	x	x	x	x	x
x	x	x	x	x	x
x	x	x	x	x	x

Следующей вершиной, по тому же принципу, размещается v_{17}:

Таблица 3.1.10. Позиции-кандидаты на размещение для вершины v_{17}.

11	**20**	22	x	x	x
5	10	x	x	x	x
4	8	x	x	x	x
8	x	x	x	x	x
x	x	x	x	x	x

Таблица 3.1.11. Позиции-кандидаты на размещение для вершины v_3.

11	**20**	41	x	x	x
5	21	x	x	x	x
4	17	x	x	x	x
17	19	x	x	x	x
19	x	x	x	x	x

Таблица 3.1.12. Позиции-кандидаты на размещение для вершины v_{15}.

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

Таблица 3.1.13. Позиции-кандидаты на размещение для вершины v_{27}.

11	**20**	29	x	x	x
5	15	x	x	x	x
4	**3**	15	x	x	x
17	**15**	12	x	x	x
13	12	x	x	x	x

Таблица 3.1.14. Позиции-кандидаты на размещение для вершины v_{24}.

11	**20**	25	x	x	x
5	19	x	x	x	x
4	**3**	27	x	x	x
17	**15**	**27**	39	x	x
29	34	39	x	x	x

Таблица 3.1.15. Позиции-кандидаты на размещение для вершины v_{29}.

11	**20**	27	x	x	x
5	**24**	18	x	x	x
4	**3**	17	x	x	x
17	**15**	**27**	25	x	x
27	24	25	x	x	x

Таблица 3.1.16. Позиции-кандидаты на размещение для вершины v_{22}.

11	**20**	x	x	x	x
5	**24**	x	x	x	x
4	**3**	**29**	x	x	x
17	**15**	**27**	x	x	x
x	x	x	x	x	x

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 $Q = 1154$.

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

$$L_i = \frac{\sum_{j=1}^{n} d_{ij} r_{ij}}{\rho_i}.$$

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

Таблица 3.2.1. Значение средней длины для каждой размещенной вершины.

11$_{39}$	**20**$_{74}$	**1**$_{85}$	**21**$_{95}$	**28**$_{87}$	**0**$_{75}$
5$_{63}$	**24**$_{61}$	**22**$_{54}$	**2**$_{45}$	**16**$_{64}$	**13**$_{109}$
4$_{41}$	**3**$_{87}$	**29**$_{65}$	**8**$_{57}$	**25**$_{107}$	**18**$_{113}$
17$_{54}$	**15**$_{79}$	**27**$_{49}$	**26**$_{57}$	**12**$_{79}$	**9**$_{54}$
23$_{103}$	**19**$_{114}$	**10**$_{105}$	**14**$_{49}$	**7**$_{105}$	**6**$_{139}$

Из таблицы 3.2.1 видно, что наихудшим образом расположена вершина с самой большой средней длиной связи – вершина v_6.

Затем для вершины v_6 определяется ее оптимальные координаты – координаты точки, в которой суммарная, а значит, и средняя длина связей вершины будет минимальна.

$$y_{k\ opt} = \frac{\sum_j y_j r_{kj}}{\rho_k};\ x_{k\ opt} = \frac{\sum_j x_j r_{kj}}{\rho_k}.$$

Для вершины v_6 оптимальными координатами, иначе говоря «центром масс», являются координаты $y_{6\ opt} = 2.125; x_{6\ opt} = 2.21875$.

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 v_6, участвующая в перестановке – зеленым.

Таблица 3.2.2. Вершины-кандидаты на перестановку с вершиной v_6.

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 v_6. Для каждой из них рассчитывается приращение суммарной длины связи $\Delta L_{ij} = (L'_i - L_i) + (L'_j - L_j)$ – значение, на которое изменится суммарная длина связи Q при осуществлении перестановки. Приращение суммарной длины связи ΔL_{ij} 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	24_{80}	22_{38}	2_{-3}	16	13
4	3_{40}	29_{18}	8_{46}	25	18
17	15_{42}	27_{46}	26_{17}	12	9
23	19	10	14	7	6

Закрепляется наилучшая перестановка, ΔL_{ij} которой будет отрицательное и наибольшее по модулю. В данном случае только одна перестановка уменьшит значение суммарной длины связи – перестановка v_6 и v_2. Размещение после перестановки представлено в таблице 3.2.4.

Таблица 3.2.4. Размещение после перестановки v_6 и v_2

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, $L_{i\,max}$), есть хотя бы одна перестановка, дающая положительный результат (приращение суммарной длины связи которой отрицательно, $\Delta L_{ij} < 0$).

Таблица 3.2.5. Значение средней длины для каждой размещенной вершины.

11$_{29}$	20$_{64}$	1$_{85}$	21$_{95}$	28$_{78}$	0$_{70}$
5$_{63}$	24$_{61}$	22$_{54}$	6$_{113}$	16$_{70}$	13$_{107}$
4$_{38}$	3$_{87}$	29$_{71}$	8$_{66}$	25$_{107}$	18$_{113}$
17$_{54}$	15$_{76}$	27$_{49}$	26$_{60}$	12$_{79}$	9$_{57}$
23$_{106}$	19$_{115}$	10$_{102}$	14$_{48}$	7$_{117}$	2$_{68}$

Наихудшим образом расположена вершина v_7. Ее центр масс находится в точке с координатами $y_{7\,opt} = 2.0; x_{7\,opt} = 1.5454$. Точки, расположенные вокруг ее центра масс и приращение ΔL_{ij} при перестановке с ними представлены в таблице 3.2.6.

Таблица 3.2.6. Значение приращения вершин-кандидатов на перестановку с v_7.

11	20	1	21	28	0
5	24$_{76}$	22$_{24}$	6$_{18}$	16	13
4	3$_{48}$	29$_{10}$	8$_{-5}$	25	18
17	15$_{30}$	27$_{26}$	26$_{-16}$	12	9
23	19	10	14	7	2

Вершина с наибольшим по модулю отрицательным приращением v_{26}. Закрепляем перестановку вершин v_7 и v_{26} и снова считаем L_i для каждой вершины.

Таблица 3.2.7. Значение средней длины для каждой размещенной вершины.

11$_{29}$	20$_{64}$	1$_{85}$	21$_{91}$	28$_{78}$	0$_{70}$
5$_{63}$	24$_{63}$	22$_{60}$	6$_{105}$	16$_{70}$	13$_{107}$
4$_{38}$	3$_{83}$	29$_{65}$	8$_{62}$	25$_{107}$	18$_{113}$
17$_{54}$	15$_{84}$	27$_{49}$	7$_{91}$	12$_{79}$	9$_{57}$
23$_{106}$	19$_{115}$	10$_{102}$	14$_{48}$	26$_{70}$	2$_{62}$

Наихудшим образом расположена вершина v_{19}. Ее центр масс находится в точке с координатами $y_{19\,opt} = 1.6897; x_{19\,opt} = 2.2759$. Точки, расположенные вокруг ее центра масс и приращение ΔL_{ij} при перестановке с ними представлены в таблице 3.2.8.

Таблица 3.2.8. Значение приращения вершин-кандидатов на перестановку с v_{19}.

11	20	1	21	28	0
5	24$_{39}$	22$_{2}$	6$_{35}$	16	13
4	3$_{16}$	29$_{1}$	8$_{-6}$	25	18
17	15$_{1}$	27$_{0}$	7$_{24}$	12	9
23	19	10	14	26	2

Вершины для перестановки v_8 и v_{19}.

Таблица 3.2.9. Значение средней длины для каждой размещенной вершины.

11$_{29}$	20$_{64}$	1$_{89}$	21$_{91}$	28$_{78}$	0$_{70}$
5$_{63}$	24$_{63}$	22$_{69}$	6$_{97}$	16$_{78}$	13$_{91}$
4$_{38}$	3$_{83}$	29$_{65}$	19$_{79}$	25$_{115}$	18$_{101}$
17$_{54}$	15$_{84}$	27$_{49}$	7$_{91}$	12$_{79}$	9$_{57}$
23$_{106}$	8$_{115}$	10$_{102}$	14$_{48}$	26$_{70}$	2$_{62}$

Наихудшим образом расположена вершина v_{25}. Ее центр масс находится в точке с координатами $y_{25\,opt} = 1.1471; x_{25\,opt} = 2.2353$ Точки, расположенные вокруг ее центра масс и приращение ΔL_{ij} при перестановке с ними представлены в таблице 3.2.10.

Таблица 3.2.10. Значение приращения вершин-кандидатов на перестановку с v_{25}.

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 v_{20} и v_{25}.

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 v_{25}. Ее центр масс находится в точке с координатами $y_{25\,opt} = 2.0294; x_{25\,opt} = 1.5882$. Точки, расположенные вокруг ее центра масс и приращение ΔL_{ij} при перестановке с ними представлены в таблице 3.2.12.

Таблица 3.2.12. Значение приращения вершин-кандидатов на перестановку с v_{25}.

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 v_{25} и v_{24}.

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 v_{25}. Ее центр масс находится в точке с координатами $y_{25\,opt} = 1.5588; x_{25\,opt} = 1.5882$. Точки, расположенные вокруг ее центра масс и приращение ΔL_{ij} при перестановке с ними представлены в таблице 3.2.14.

Таблица 3.2.14. Значение приращения вершин-кандидатов на перестановку с v_{25}.

11	24	1	21	28	0
5	25_{0}	22_{1}	6_{32}	16	13
4	3_{-5}	29_{4}	19_{27}	20	18
17	15_{0}	27_{-1}	7_{30}	12	9
23	8	10	14	26	2

Вершины для перестановки v_{25} и v_{3}.

Таблица 3.2.15. Значение средней длины для каждой размещенной вершины.

11_{30}	24_{65}	1_{81}	21_{89}	28_{83}	0_{70}
5_{59}	3_{83}	22_{68}	6_{95}	16_{78}	13_{75}
4_{32}	25_{96}	29_{66}	19_{76}	20_{28}	18_{98}
17_{55}	15_{83}	27_{53}	7_{95}	12_{81}	9_{53}
23_{94}	8_{91}	10_{105}	14_{49}	26_{71}	2_{62}

Наихудшим образом расположена вершина v_{10}. Ее центр масс находится в точке с координатами $y_{10\,opt} = 2.1667; x_{10\,opt} = 1.3333$. Точки, расположенные вокруг ее центра масс и приращение ΔL_{ij} при перестановке с ними представлены в таблице 3.2.16.

Таблица 3.2.16. Значение приращения вершин-кандидатов на перестановку с v_{10}.

11	24	1	21	28	0
5_{64}	3_{30}	22_{-1}	6	16	13
4_{48}	25_{24}	29_{-6}	19	20	18
17_{51}	15_{10}	27_{-1}	7	12	9
23	8	10	14	26	2

Вершины для перестановки v_{10} и v_{29}.

Таблица 3.2.17. Значение средней длины для каждой размещенной вершины.

1130	**24**67	**1**77	**21**89	**28**83	**0**70
561	**3**77	**22**62	**6**95	**16**76	**13**75
432	**25**96	**10**83	**19**76	**20**28	**18**96
1755	**15**83	**27**53	**7**95	**12**81	**9**53
2394	**8**91	**29**82	**14**57	**26**71	**2**64

Наихудшим образом расположена вершина v_{25}. Ее центр масс находится в точке с координатами $y_{25\,opt} = 1.2353; x_{25\,opt} = 1.5882$. Точки, расположенные вокруг ее центра масс и приращение ΔL_{ij} при перестановке с ними представлены в таблице 3.2.18.

Таблица 3.2.18. Значение приращения вершин-кандидатов на перестановку с v_{25}.

11	**24**32	**1**53	**21**30	**28**	**0**
5	**3**5	**22**8	**6**36	**16**	**13**
4	**25**0	**10**18	**19**30	**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 $Q = 1076$.

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, 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]]
```