**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский

технический университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

Институт компьютерных технологий и защиты информации

Кафедра систем автоматизированного проектирования

09.03.01. «Информатика и вычислительная техника»

Курсовая работа

по дисциплине: «Методы программирования систем автоматизированного проектирования»  
на тему: «Решение задачи о размещении вершин графа в линейке с использованием метода ветвей и границ»

Обучающийся 4215 \_\_\_\_\_\_\_\_\_\_\_\_ Закиров И.Р.

(номер группы) (подпись, дата) (Ф.И.О.)

Руководитель старший преподаватель Суздальцев И.В.

(должность) (Ф.И.О.)

Курсовая работа (проект) зачтена (зачтен) с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

Казань 2020

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc61669990)

[1. ПОСТАНОВКА ЗАДАЧИ 5](#_Toc61669991)

[1. 1. Содержательная постановка задачи 5](#_Toc61669992)

[1. 2. Математическая постановка задачи 5](#_Toc61669993)

[2. РАЗРАБОТКА АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ 8](#_Toc61669994)

[3. РЕШЕНИЕ ЗАДАЧИ НА КОНТРОЛЬНОМ ПРИМЕРЕ 12](#_Toc61669995)

[4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ РАЗРАБОТАННОГО АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ 33](#_Toc61669996)

[5. ОЦЕНКА ВРЕМЕННОЙ СЛОЖНОСТИ 34](#_Toc61669997)

[ЗАКЛЮЧЕНИЕ 37](#_Toc61669998)

[СПИСОК ИСТОЧНИКОВ 38](#_Toc61669999)

[ПРИЛОЖЕНИЕ A. ЭКРАННЫЕ ФОРМЫ 39](#_Toc61670000)

[ПРИЛОЖЕНИЕ Б. ЛИСТИНГ ПРОГРАММЫ 40](#_Toc61670001)

# **ВВЕДЕНИЕ**

В ходе работы будет рассматриваться задача размещения графа на линейке, имеющая множество практических применений. Самое явное – размещение элементов на печатных платах с минимально возможным количеством соединяющего материала. Нахождение новых способов решения этой задачи поможет уменьшить трудоёмкость проектирования систем самого широкого спектра, а это актуально и будет актуально всегда в связи с неуклонным усложнением техники.

Сама задача является NP-трудной, и до сих пор нет эффективных методов её решения, и поэтому для решения задачи часто применяют генетические алгоритмы. Мы же будем использовать метод ветвей и границ, являющийся более эффективной вариацией полного перебора, позволяющий отбраковывать некоторое количество бесперспективных вариантов.

Целью работы является изучить алгоритм решения задачи размещения – метода ветвей и границ, и осуществить его программную реализацию на языке C++ с использованием объектно-ориентированного программирования, алгоритмов и структур данных, изученных в ходе учебного курса.

Основными задачами работы является формирование постановки задачи, изучение заданного алгоритма решения этой задачи, программная реализация алгоритма и оценка временной сложности полученной реализации.

# **1. ПОСТАНОВКА ЗАДАЧИ**

## **1. 1. Содержательная постановка задачи**

В этой работе будет рассматриваться задача размещения графа на линейке. В самой задаче решение ищется для взвешенного графа. Граф G (V, U) – математический объект, являющийся совокупностью множеств вершин V и рёбер U. Каждое ребро может соединять только две вершины. Взвешенный граф — граф, каждому ребру которого поставлено в соответствие некое значение (вес ребра). Обычно вес – это число, и его интерпретируют как длину ребра.

Исходные данные для решения задачи: количество вершин графа, матрица смежности с указанием веса рёбер между вершинами. Матрица смежности – матрица элемент которого равен весу ребра, соединяющего вершины и . Далее, на рис. 1, представлен пример графа и его размещения на линейке:

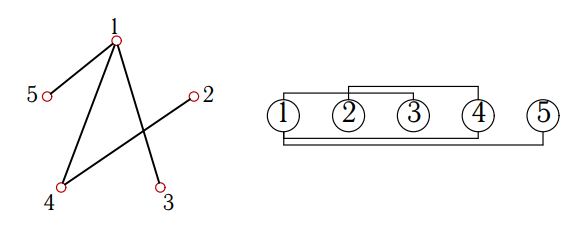


Рис. 1. Невзвешенный граф и его разложение на линейке

В результате мы должны получить оптимальные позиции вершин графа на линейке с минимальной суммарной длиной рёбер. Выводиться будет в виде n-ки вершин, расположенных в линейке (например, [3, 5, 2, 4, 1]).

Задача размещения графа на линейке (а так же на плоскости) имеет реальное практическое применение: размещение микросхем с использованием наименьшего количества соединительного материала. Это полезно и экономией материала, и улучшением пропускной способности соединений. Так же проблема размещения, исходя из названия, может возникнуть при размещении чего угодно, у чего есть межэлементные связи и рекомендации эти связи оптимизировать. Прецеденты можно найти на множестве предприятий, к примеру, при работе с конвейерами.

## **1. 2. Математическая постановка задачи**

Таблица 1. Обозначения элементов математической модели задачи

|  |  |  |
| --- | --- | --- |
| **Обозначения элементов модели** | **Описание элементов модели** | **Примечание** |
|  | Исходный граф |  |
|  | Множество вершин |  |
|  | Множество рёбер |  |
|  | Количество вершин графа |  |
|  | Количество позиций |  |
|  | Матрица смежности с указанием веса рёбер между вершинами |  |
|  | Вес ребра, инцидентного -й и -й вершинам графа | *,* , |
|  | Кортеж, определяющий одно из возможных решений |  |
|  | Вершина графа, размещённая в данной позиции |  |

Таблица 1 содержит все обозначения, используемые в математической постановке задачи.

Целевая функция:

Ограничения задачи:

В решении для каждой вершины назначается только одна позиция (m = n).

Количество потенциальных решений для задачи о размещении графа определяется по комбинаторной формуле (как количество перестановок вершин графа):

где – количество вершин графа, – общее число вариантов размещения вершин графа на линейке.

# **2. РАЗРАБОТКА АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ**

Метод ветвей и границ – метод, предназначенный для нахождения решений задач оптимизации. Метод является развитием метода полного перебора, он позволяет отсеивать часть возможных неоптимальных решений. Впервые был предложен в 1960-м году Алисой Лэнд и Элисон Дойг. Распространён для решения NP-сложных задач на оптимизацию.

Основа метода – сочетание двух действий: последовательное разбиение множества решений на подмножества (ветвление), оценка нижней границы и отбрасывание бесперспективных ветвей. На каждом шаге метода элементы разбиения подвергаются проверке – может ли здесь содержаться оптимальное решение или нет. Проверка осуществляется посредством вычисления оценки снизу для целевой функции на данном подмножестве. Если оценка снизу не меньше рекорда — наилучшего из найденных решений, то подмножество далее не рассматривается. Если значение целевой функции на найденном решении меньше рекорда, то происходит смена рекорда. По окончанию работы алгоритма на этой стадии находится самое перспективное подмножество (с наименьшим значением нижней оценки), и оно подвергается дальнейшему разбиению. Новые подмножества вновь подвергаются проверке и т.д. Основа алгоритма и его самая важная часть – способ вычисления нижней границы на каждом из этапов.

Далее перейдём к описанию шагов алгоритма.

R – матрица смежности графа.

D – матрица расстояний между позициями.

n – количество вершин в графе.

Fq – суммарная длина соединений между размещёнными элементами.

wP – длина соединений между размещёнными и неразмещёнными элементами.

vP – длина соединений между неразмещёнными элементам.

minF – минимальная нижняя граница целевой функции на данном этапе.

minP – позиция, с которой связана minF.

FP – нижняя оценка целевой функции.

1. Инициализация: создаём список Р из значений от 1 до n. Создаём пустой список Pp для занятых позиций из Р. Первое значение будет указывать, на какой позиции находится первая вершина, второе значение – местонахождение второй вершины и так далее. Создаём переменную Ei, изначально равную единице, обозначающую, какая вершина в данный момент размещается. Создаём переменные FP, Fq, wP, vP, minF, minP.

2. В minF записываем максимально возможное значение типа данных.

3. Начинаем проходится по списку P, перебирая все значения (шаги 4-8).

4. Обнуляем FP, Fq, wP, vP.

5. Если Pp не пуст, то:

5.1. От 1 до Ei включительно находим (по матрице R) соединённые рёбрами вершины и вычисляем Fq, складывая перемноженные веса рёбер между вершинами и расстояния между их размещениями (местоположения вершин от 1 до Ei-1 находятся в списке Pp, вершине Ei соответствует текущее значение из P).

Иначе:

5.2. Fq = 0.

6. Находим wP. Для этого перебираем вершины от 1 до Ei включительно, для каждой находя по матрице R расстояния от них самих до вершин от Ei+1 до n включительно. Значения эти записываем в вектор r в порядке не возрастания. Так же для каждой занятой, из списка Pp, (и одной занимаемой в данный момент) позиции находим по матрице D расстояния до незанятых позиций (перебираем для этого P) и записываем эти значения в вектор d в порядке не убывания. Скалярно перемножаем вектора r и d для нахождения wP.

7. Находим длину vP. Находим по матрице R пересечения вершин от Ei+1 до n включительно друг с другом, образуя невозрастающий вектор r, а по матрице D находим пересечения позиций из P друг с другом, образуя неубывающий вектор d. Скалярно перемножаем вектора r и d для нахождения vP.

Вычисляем FP, складывая Fq, wP и vP.

8. Если FP меньше minF, то перезаписываем minF=FP и minP = текущий элемент P (8.1).

9. Перебираем список P, убирая из него значение minP и добавляя его в конец списка Pp (9.2).

10. Увеличиваем Ei на 1.

11. Повторяем шаги 2-10, пока список Р не пуст.

12. Выводим искомое размещение, которое соответствует массиву Рр.

На рисунках 2 и 3 представлена блок-схема алгоритма.

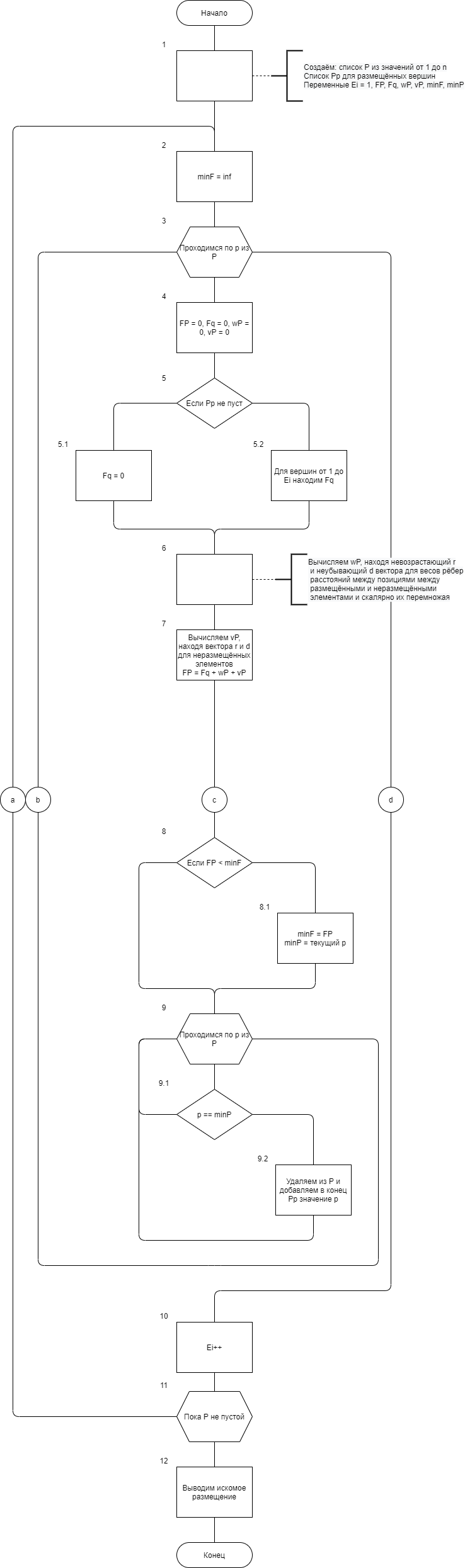


Рис. 2. Алгоритмическая схема, часть 1

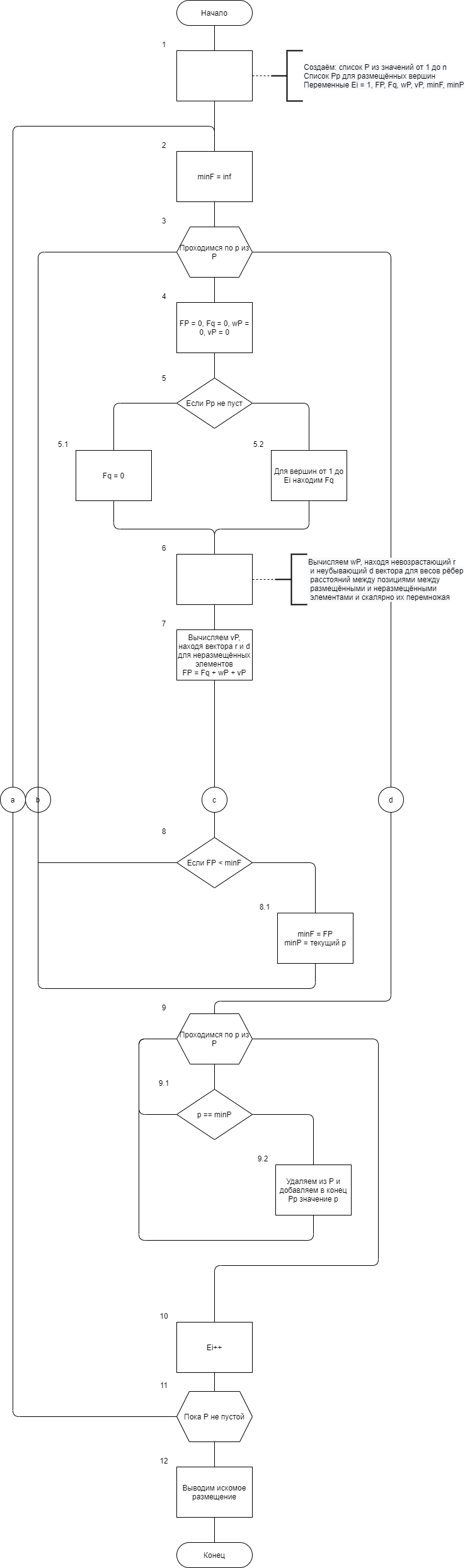


Рис. 3. Алгоритмическая схема, часть 2

# **3. РЕШЕНИЕ ЗАДАЧИ НА КОНТРОЛЬНОМ ПРИМЕРЕ**

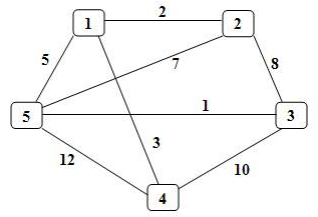


Рис. 4. Граф для решения задачи

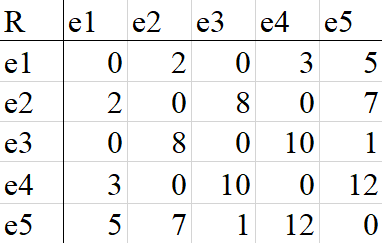


Рис. 5. Матрица смежности

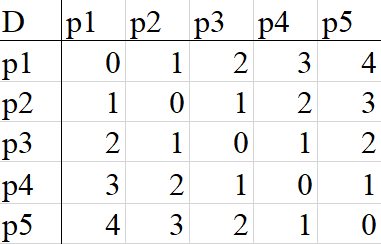


Рис. 6. Матрица расстояний

Шаг1. Инициализация алгоритма, показана на рисунке 7.

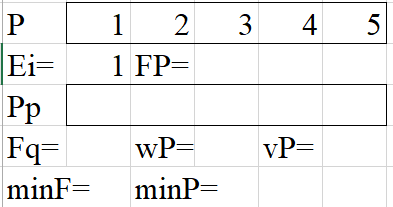


Рис. 7. Инициализация

Шаг 2. Задаём минимальное значение большим числом.

Шаг 3. “Устанавливаем” первую вершину в гнездо p1.

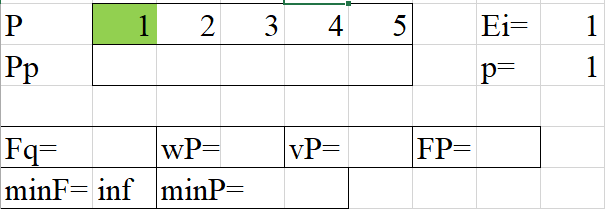


Рис. 8. Вершина 1.

Шаг 4. Обнуляем переменные перед работой тела цикла.

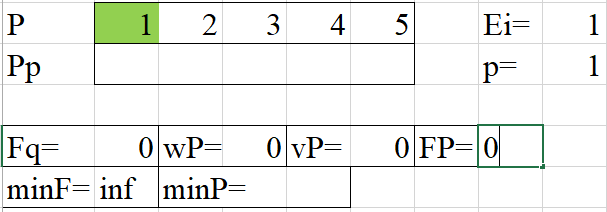


Рис. 9. Обнуление переменных.

Шаг 5. Рр пока что пуст, поэтому Fq оставляем равным нулю.

Шаг 6. Так как вершина у нас только одна, находим единственный вектор r (расстояния от первой вершины до всех остальных) и единственный же вектор d, отсортированные. После мы их скалярно перемножили, получив в итоге 17.

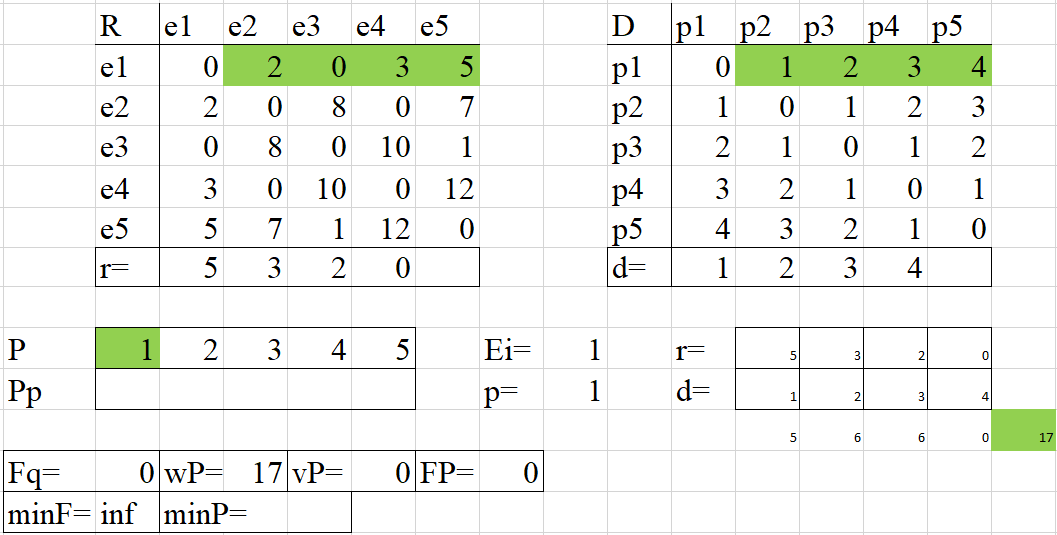


Рис. 10. Вычисление wP.

Шаг 7. Зачеркнув строки и столбцы, соответствующие первой вершине и первому гнезду, получаем вектора и их скалярное произведение – vP = 46. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

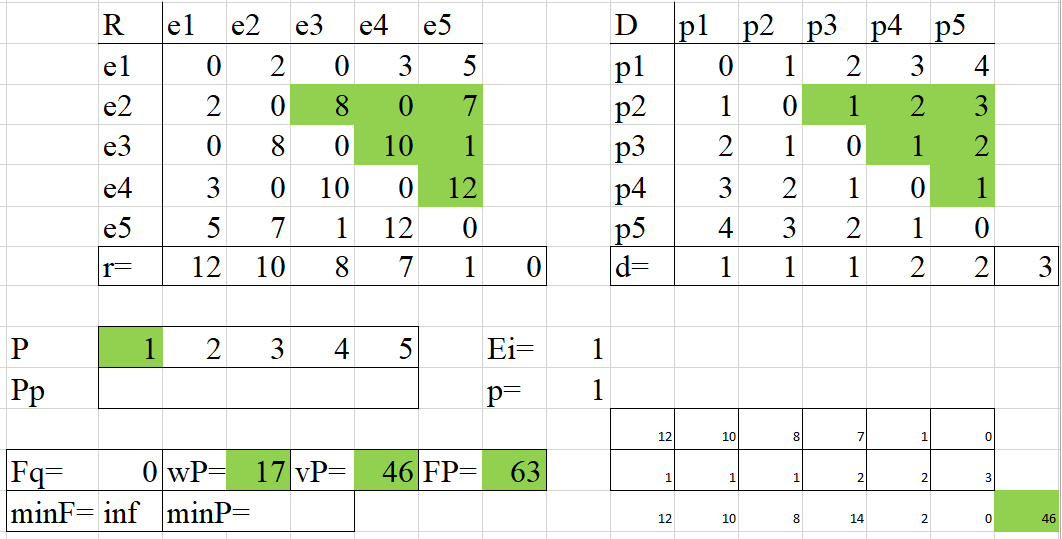


Рис. 11. Вычисление vP и нижней оценки.

Шаг 8. FP=63 – первое полученное значение, поэтому мы сразу записываем его в minF, а так же запишем, что значение мы нашли при установлении вершины в гнездо p1.

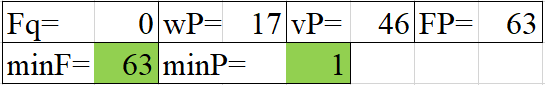


Рис. 12. Сравнение с рекордом.

Шаг 9 (3). Возвращаемся к шагу 3, продолжаем брать значения из P, следующее – 2. Вершина всё ещё первая.

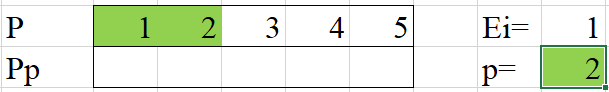


Рис. 13. Выбор следующего гнезда.

Шаг 10 (4). Обнулили значения.

Шаг 11 (5). Рр пока что пуст, поэтому Fq оставляем равным нулю.

Шаг 12 (6). Находим единственный вектор r (расстояния от первой вершины до всех остальных) и единственный же вектор d (расстояние уже от второго гнезда до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 12.

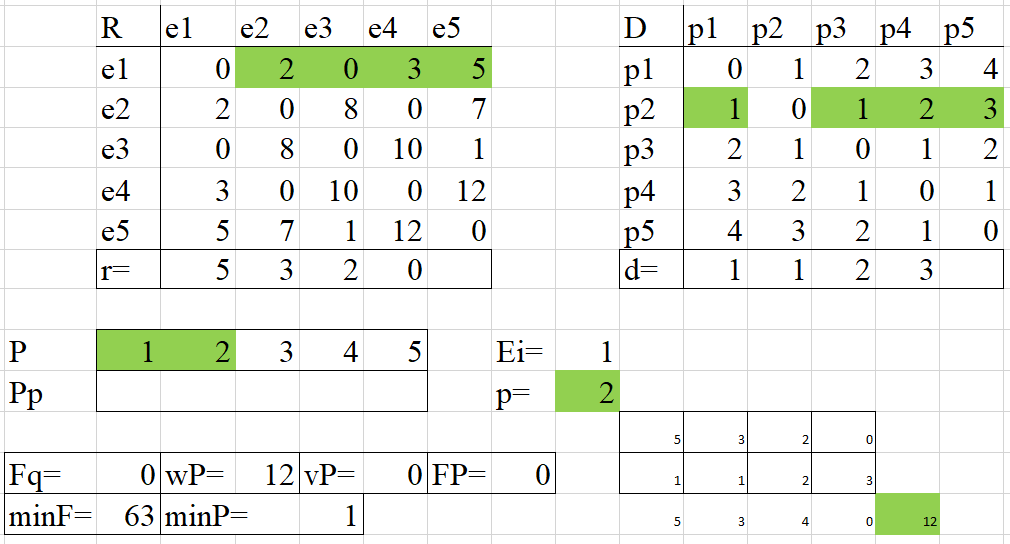


Рис. 14. Вычисление wP.

Шаг 13 (7). Зачеркнув строки и столбцы, соответствующие первой вершине и второму гнезду, получаем вектора и их скалярное произведение – vP = 55. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

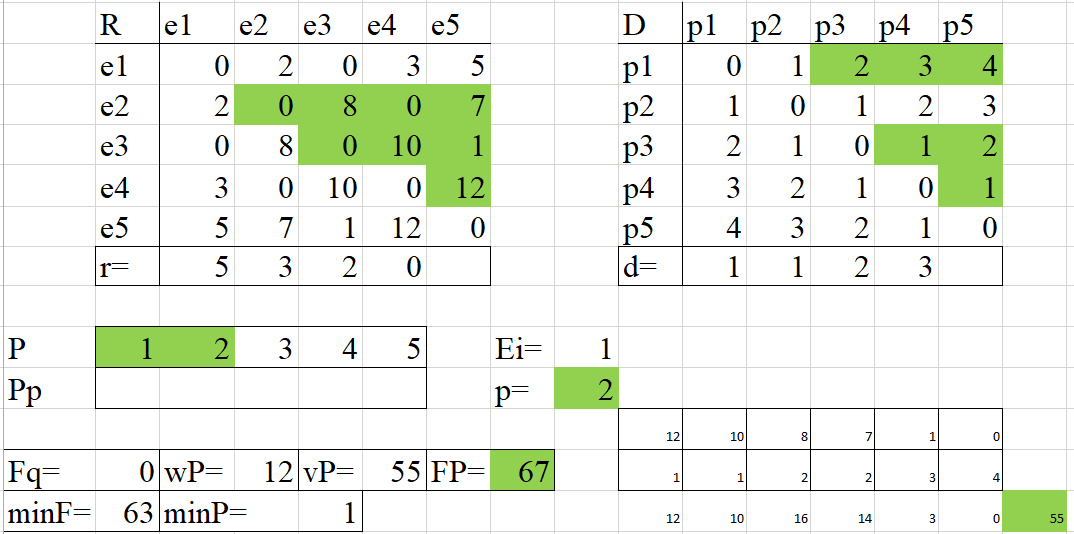


Рис. 15. Вычисление vP.

Шаг 14 (8). Так как 67 больше, чем 63, то мы оставляем прошлые значения неизменными.

Шаг 15 (3). Возвращаемся к шагу 3, продолжаем брать значения из P, следующее – 3. Вершина всё ещё первая.

Шаг 16 (4). Обнулили значения.

Шаг 17 (5). Рр пока что пуст, поэтому Fq оставляем равным нулю.

Шаг 18 (6). Находим единственный вектор r (расстояния от первой вершины до всех остальных) и единственный же вектор d (расстояние от третьего гнезда до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 12.

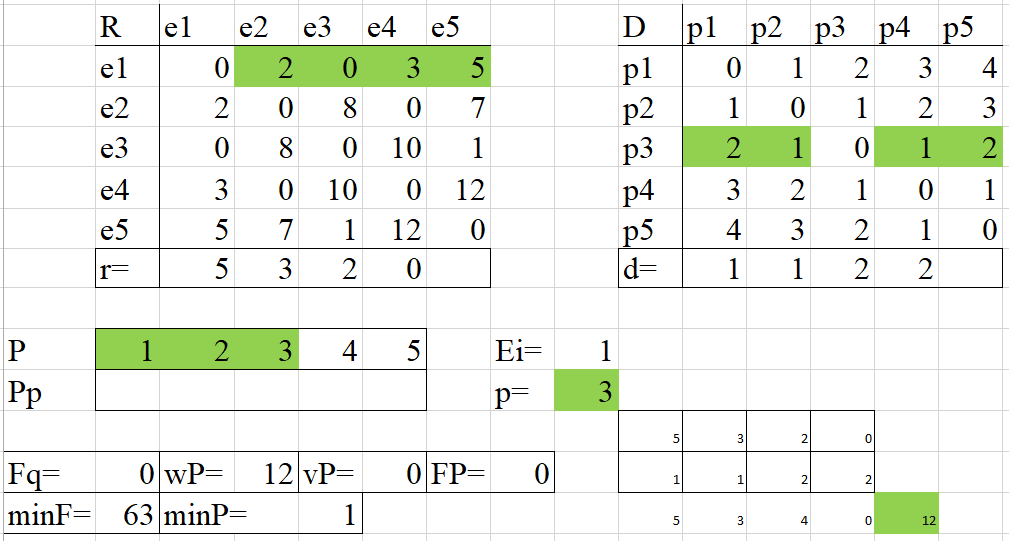


Рис. 16. Вычисление wP.

Шаг 19 (7). Зачеркнув строки и столбцы, соответствующие первой вершине и третьему гнезду, получаем вектора и их скалярное произведение – vP = 62. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

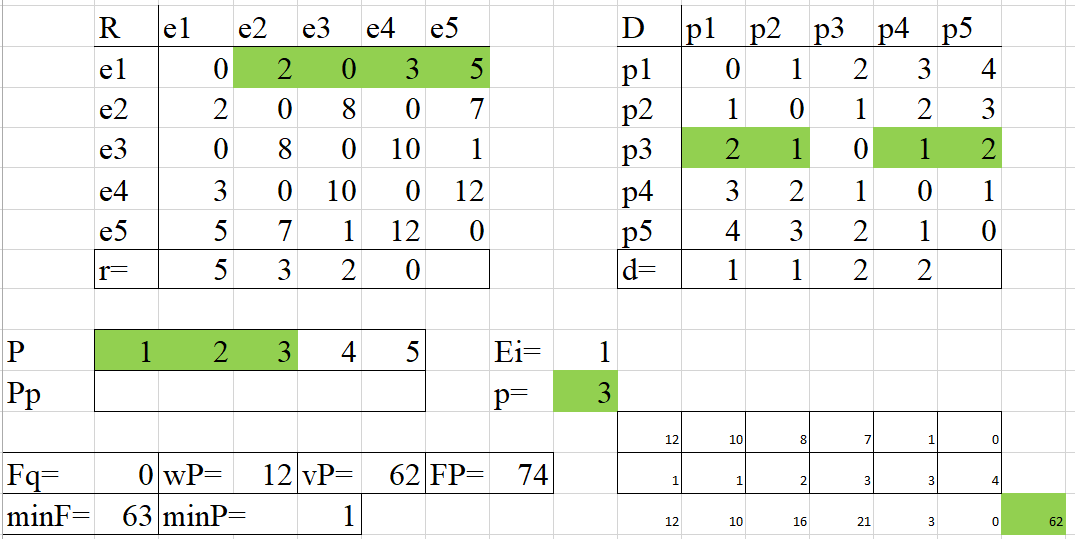


Рис. 17. Вычисление vP и нижней оценки.

Шаг 20 (8). Так как 74 больше, чем 63, то мы оставляем прошлые значения неизменными.

Примечание: Позиции 4 и 5 симметричны позициям 2 и 1 соответственно, следовательно значение нижней оценки получится абсолютно таким же, и эти шаги мы опустим, сразу перейдя к 9-му шагу основного алгоритма и продолжив уже с позицией номер два.

Шаг 33 (9). minP указывает на гнездо 1, гнездо с минимальной нижней границей. Есть ещё симметричное пятая, но мы по порядку возьмём гнездо 1. Итого, первая вершина располагается на позиции 1.

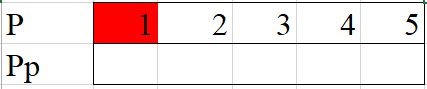
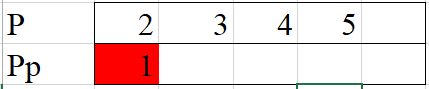
 

Рис. 18. Установка первой вершины в положение 1.

Шаг 34 (10). Перешли к вершине 2.

Шаг 35 (11). Р не пуст, идём по циклу дальше.

Шаг 36 (2). Обнулим minF для следующего уровня ветвления.

Шаг 37 (3). В Р остались гнёзда 2, 3, 4, 5. Берём гнездо 2.

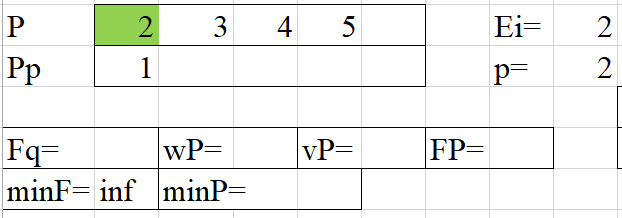


Рис. 19. Выбор следующего гнезда.

Шаг 38 (4). Обнулим значения.

Шаг 39 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 2 вершины в размещении, первая и вторая, и вес ребра между ними – 2. Расстояние между позициями – 1. Итого – 2.

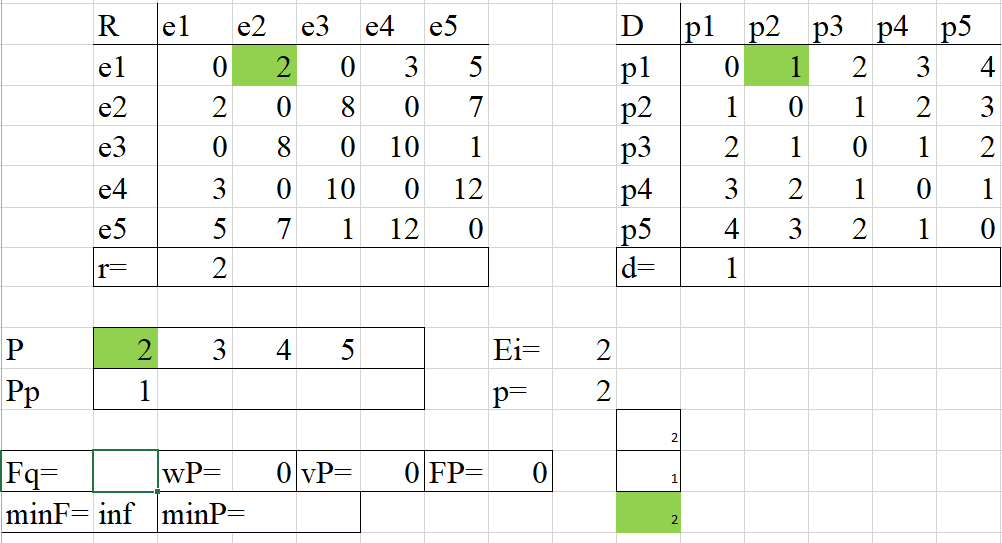


Рис. 20. Вычисление Fq.

Шаг 40 (6). Находим уже два вектора r (расстояния от первой и второй вершин до всех остальных) и два вектора d (расстояние от первого и второго гнёзд до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 41.

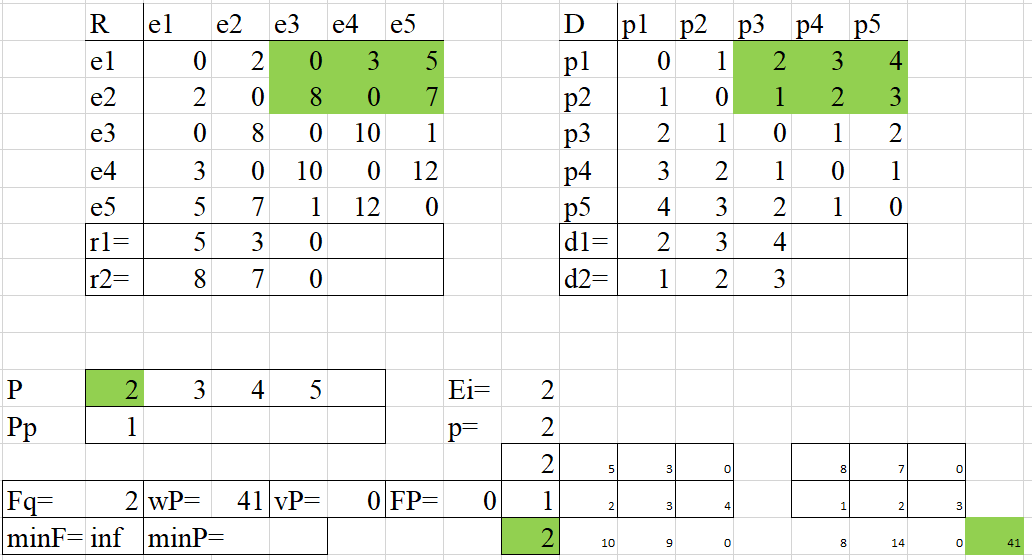


Рис. 21. Вычисление wP.

Шаг 41 (7). Зачеркнув строки и столбцы, соответствующие первой, второй вершине и первому, второму гнезду, получаем вектора и их скалярное произведение – vP = 24. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

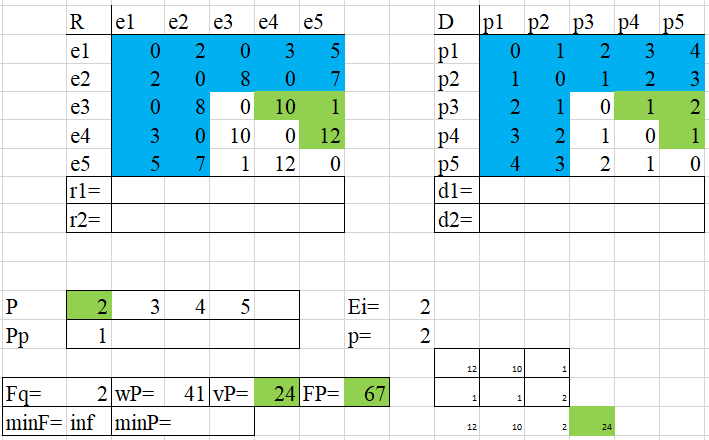


Рис. 22. Вычисление vP и нижней оценки.

Шаг 42 (8). 67 – первое значение, оно меньше максимума, его и берём. Также указываем позицию, в которой мы нашли эту оценку.



Рис. 23. Установка рекорда.

Шаг 43 (3). Возвращаемся к шагу 3, продолжаем брать значения из P, следующее – 3. Вершины – первая и вторая.

Шаг 44 (4). Обнуляем значения.

Шаг 45 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 2 вершины в размещении, первая и вторая, и вес ребра между ними – 2. Расстояние между позициями – уже 2. Итого – 4.

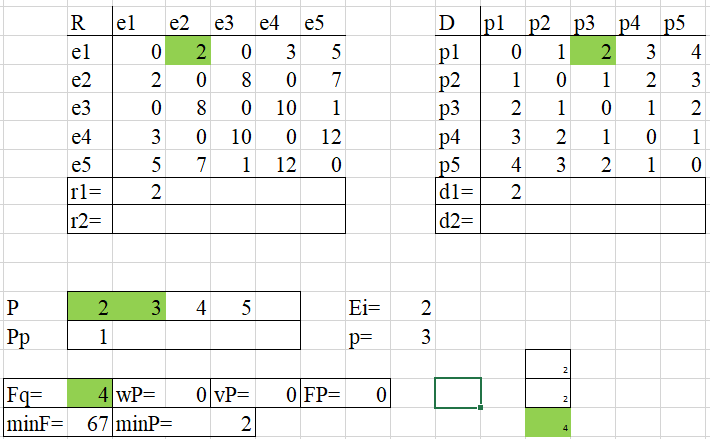


Рис. 24. Вычисление Fq.

Шаг 46 (6). Находим два вектора r (расстояния от первой и второй вершин до всех остальных) и два вектора d (расстояние от первого и третьего гнёзд до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 29.

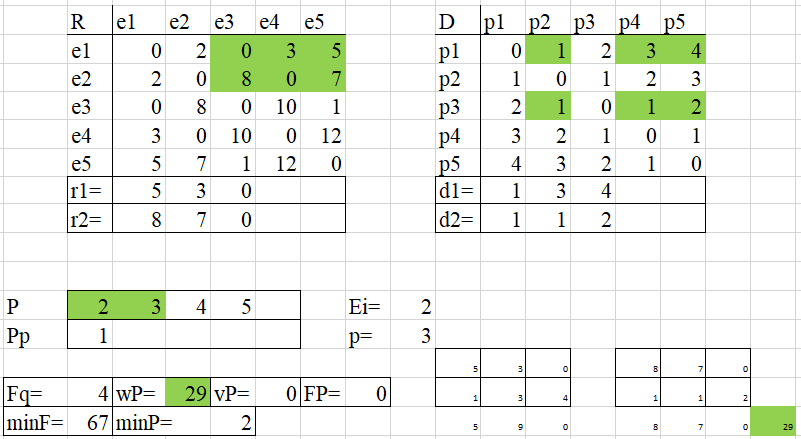


Рис. 25. Вычисление wP.

Шаг 47 (7). Зачеркнув строки и столбцы, соответствующие первой, второй вершине и первому, третьему гнезду, получаем вектора и их скалярное произведение – vP = 35. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

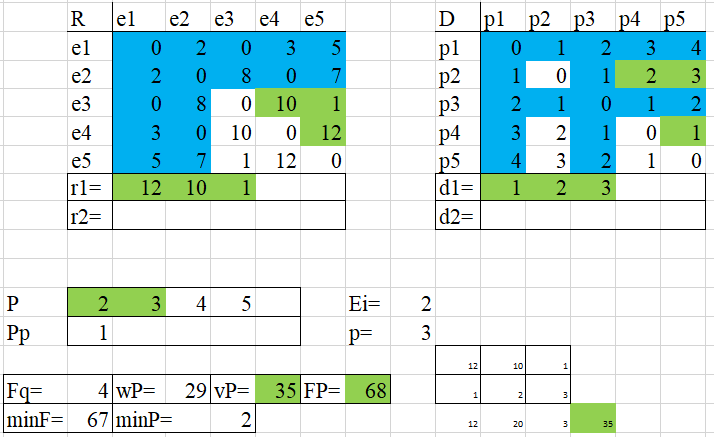


Рис. 26. Вычисление vP и нижней оценки.

Шаг 48 (8). Так как 68 больше, чем 67, то мы оставляем прошлые значения неизменными.

Шаг 49 (3). Возвращаемся к шагу 3, продолжаем брать значения из P, следующее – 4. Вершины – первая и вторая.

Шаг 50 (4). Обнуляем значения.

Шаг 51 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 2 вершины в размещении, первая и вторая, и вес ребра между ними – 2. Расстояние между позициями – уже 3. Итого – 6.

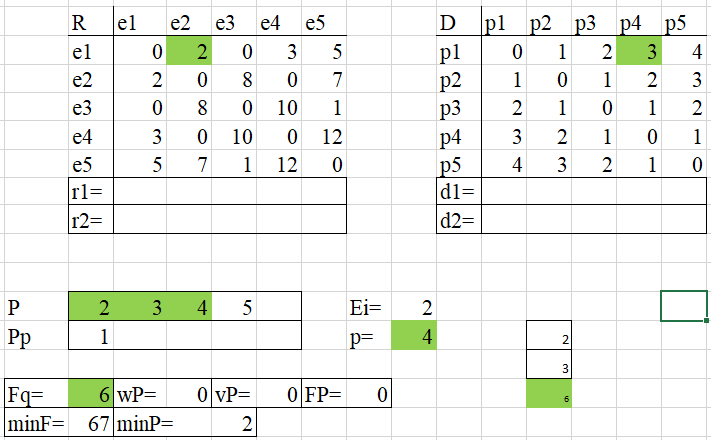


Рис. 27. Вычисление Fq.

Шаг 52 (6). Находим два вектора r (расстояния от первой и второй вершин до всех остальных) и два вектора d (расстояние от первого и четвёртого гнёзд до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 26.

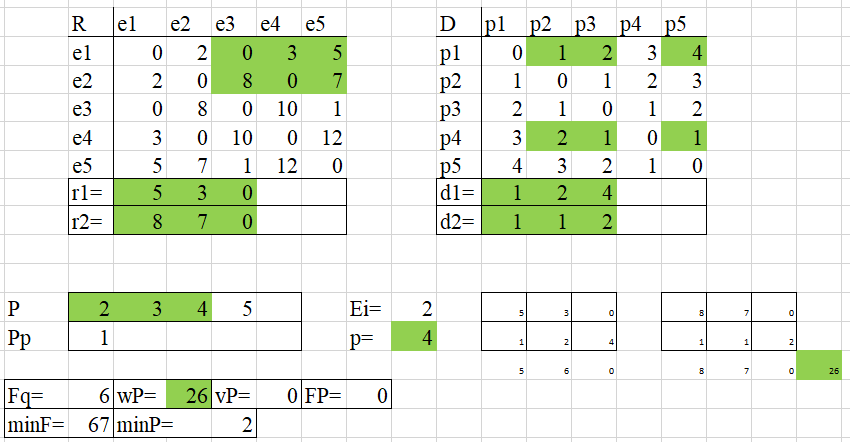


Рис. 28. Вычисление wP.

Шаг 53 (7). Зачеркнув строки и столбцы, соответствующие первой, второй вершине и первому, четвёртому гнезду, получаем вектора и их скалярное произведение – vP = 35. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

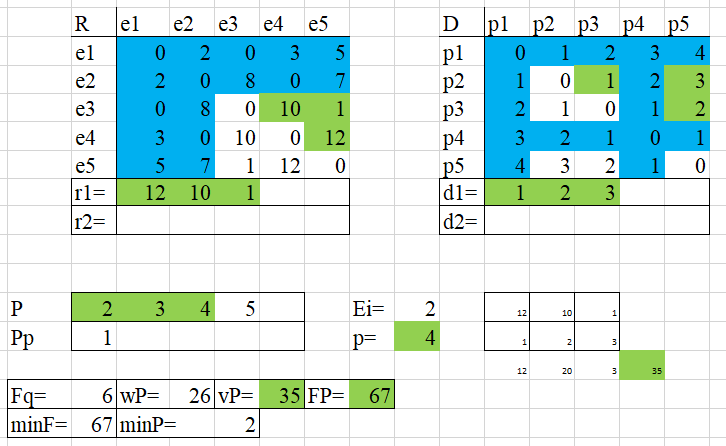


Рис. 29. Вычисление vP и нижней оценки.

Шаг 54 (8). Так как 67 равно, чем 67, то мы оставляем прошлые значения неизменными.

Шаг 55 (3). Возвращаемся к шагу 3, продолжаем брать значения из P, следующее – 5. Вершины – первая и вторая.

Шаг 56 (4). Обнуляем значения.

Шаг 57 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 2 вершины в размещении, первая и вторая, и вес ребра между ними – 2. Расстояние между позициями – уже 4. Итого – 8.

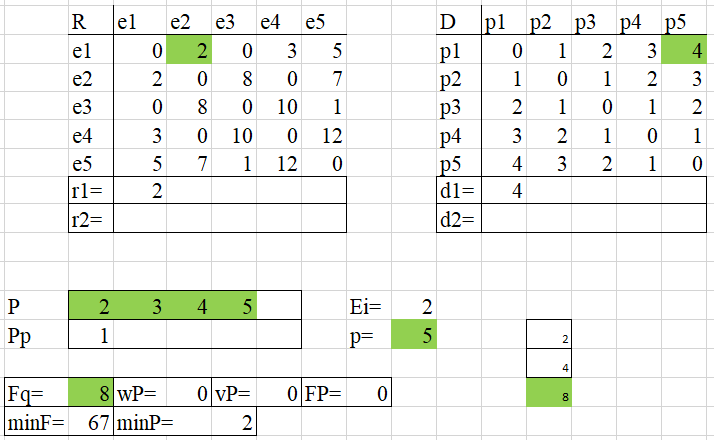


Рис. 30. Вычисление Fq.

Шаг 58 (6). Находим два вектора r (расстояния от первой и второй вершин до всех остальных) и два вектора d (расстояние от первого и четвёртого гнёзд до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 33.

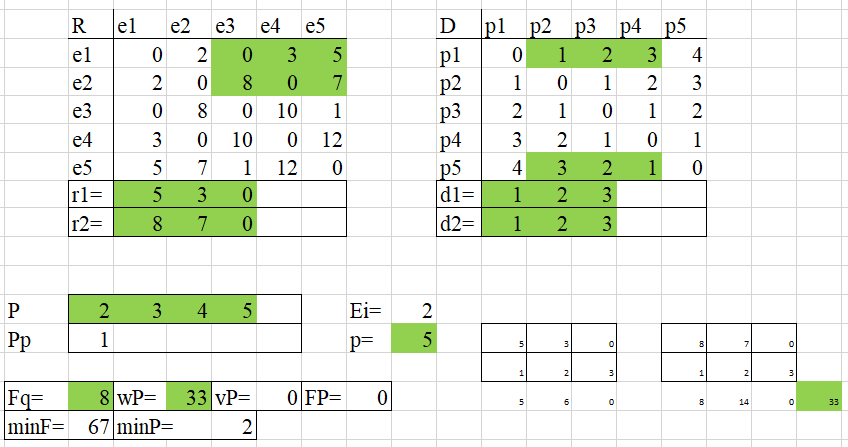


Рис. 31. Вычисление wP.

Шаг 59 (7). Зачеркнув строки и столбцы, соответствующие первой, второй вершине и первому, пятому гнезду, получаем вектора и их скалярное произведение – vP = 24. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

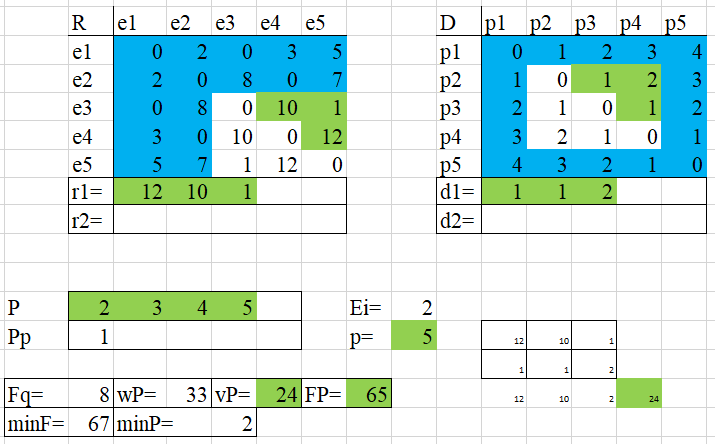


Рис. 32. Вычисление vP и нижней оценки.

Шаг 60 (8). Так как 65 меньше, чем 67, то заменяем значение функции и номер гнезда.

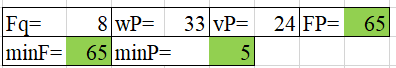


Рис. 33. Проверка рекорда.

Шаг 61 (9). minP указывает на гнездо 5, гнездо с минимальной нижней границей. Вторую вершину располагаем в гнездо 5.

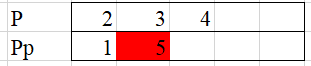
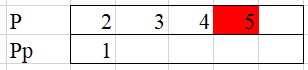


Рис. 34. Расположение второй вершины.

Шаг 62 (10). Перешли к вершине 3.

Шаг 63 (11). Р не пуст, идём по циклу дальше.

Шаг 64 (2). Обнулим minF для следующего уровня ветвления.

Шаг 65 (3). В Р остались гнёзда 2, 3, 4. Берём гнездо 2.

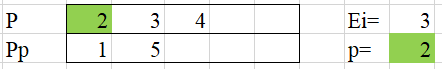


Рис. 35. Берём следующее гнездо.

Шаг 66 (4). Обнуляем значения.

Шаг 67 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 3 вершины в размещении, первая, вторая и третья, и вес ребра между первой и второй вершиной – 2, а между второй и третьей - 8. Между третьей и первой вершиной ребра нет. Расстояние между позициями размещения, соответственно – 4 и 3. Итого – 32.

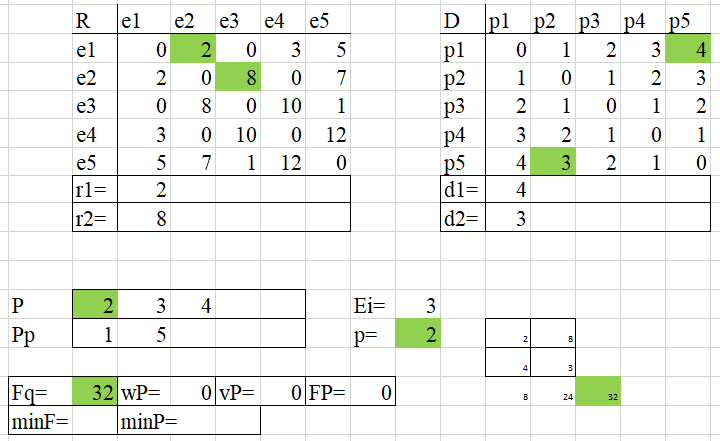


Рис. 36. Вычисление Fq.

Шаг 68 (6). Находим три вектора r (расстояния от первой, второй и третьей вершин до всех остальных) и три вектора d (расстояние от первого, пятого и второго гнёзд до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 38.

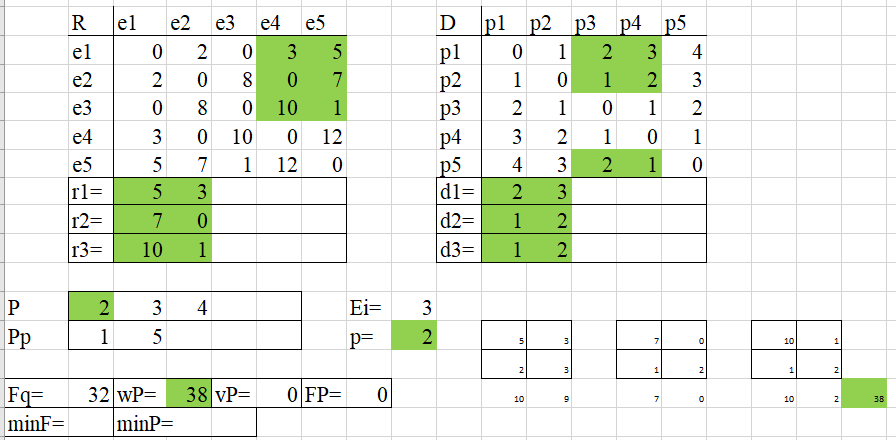


Рис. 37. Вычисление wP.

Шаг 69 (7). Зачеркнув строки и столбцы, соответствующие первой, второй и третьей вершине и первому, пятому и второму гнезду, получаем вектора и их скалярное произведение – vP = 12. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

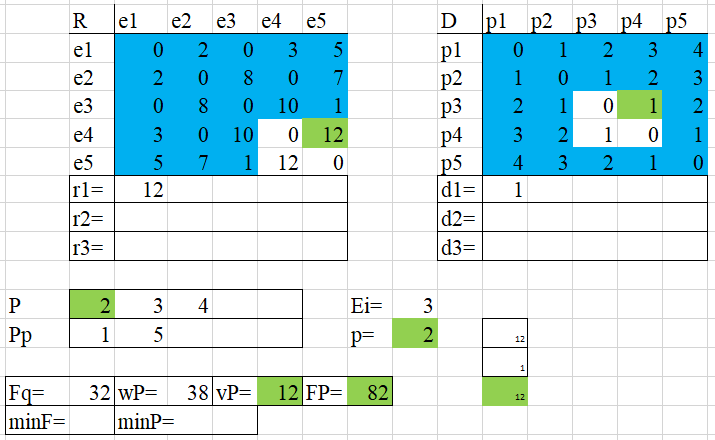


Рис. 38. Вычисление vP и нижней оценки.

Шаг 70 (8). 82 – первое значение, оно меньше максимума, его и берём. Также указываем позицию, в которой мы нашли эту оценку.

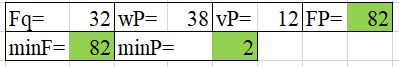


Рис. 39. Проверка рекорда.

Шаг 71 (3). Возвращаемся к шагу 3, продолжаем брать значения из P, следующее – 3. Вершины – первая, вторая и третья.

Шаг 72 (4). Обнуляем значения.

Шаг 73 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 3 вершины в размещении, первая, вторая и третья, и вес ребра между первой и второй вершиной – 2, а между второй и третьей - 8. Между третьей и первой вершиной ребра нет. Расстояние между позициями размещения, соответственно – 4 и 2. Итого – 24.

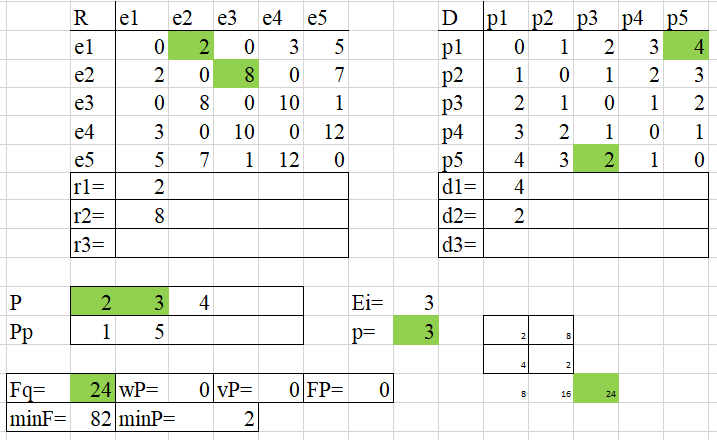


Рис. 40. Вычисление Fq.

Шаг 74 (6). Находим три вектора r (расстояния от первой, второй и третьей вершин до всех остальных) и три вектора d (расстояние от первого, пятого и третьего гнёзд до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 34.

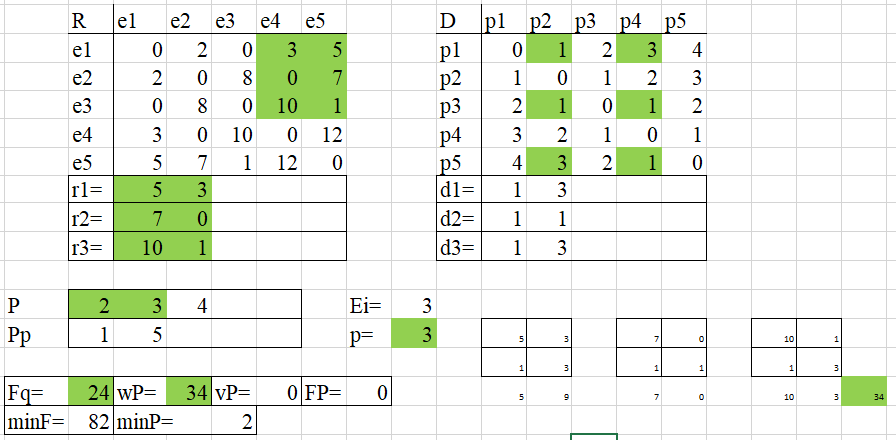


Рис. 41. Вычисление wP.

Шаг 75 (7). Зачеркнув строки и столбцы, соответствующие первой, второй и третьей вершине и первому, пятому и третьему гнезду, получаем вектора и их скалярное произведение – vP = 24. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

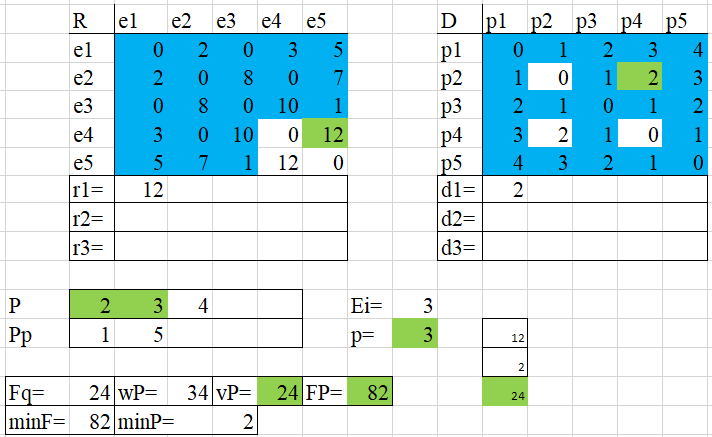


Рис. 42. Вычисление vP и нижней оценки.

Шаг 76 (8). 82 не меньше 82, поэтому оставляем прошлое значение.

Шаг 77 (3). Возвращаемся к шагу 3, продолжаем брать значения из P, следующее – 4. Вершины – первая, вторая и третья.

Шаг 78 (4). Обнуляем значения.

Шаг 79 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 3 вершины в размещении, первая, вторая и третья, и вес ребра между первой и второй вершиной – 2, а между второй и третьей - 8. Между третьей и первой вершиной ребра нет. Расстояние между позициями размещения, соответственно – 4 и 1. Итого – 16.

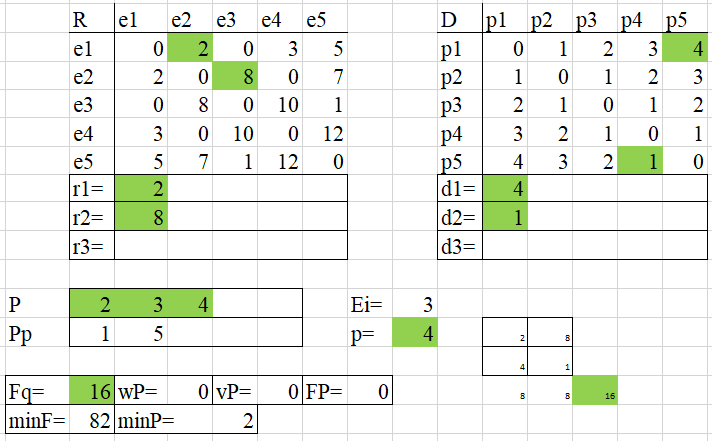


Рис. 43. Вычисление Fq.

Шаг 80 (6). Находим три вектора r (расстояния от первой, второй и третьей вершин до всех остальных) и три вектора d (расстояние от первого, пятого и четвёртого гнёзд до остальных), отсортированные. После мы их скалярно перемножили, получив в итоге 37.

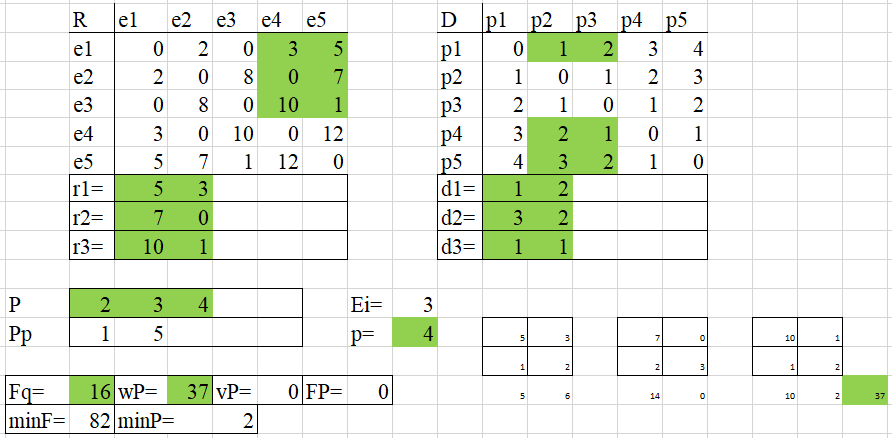


Рис. 44. Вычисление wP.

Шаг 81 (7). Зачеркнув строки и столбцы, соответствующие первой, второй и третьей вершине и первому, пятому и четвёртому гнезду, получаем вектора и их скалярное произведение – vP = 12. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

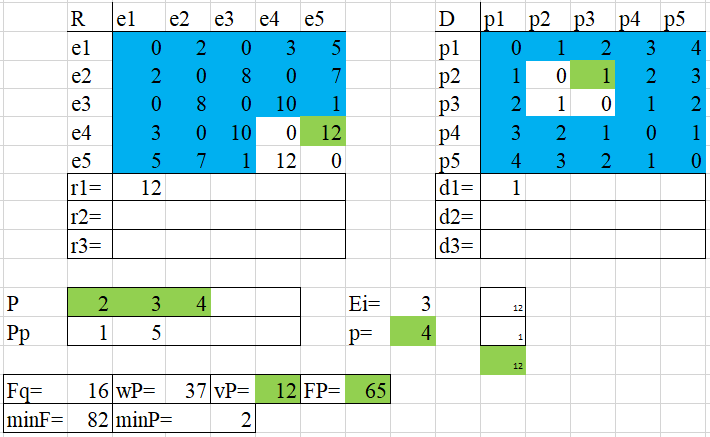


Рис. 45. Вычисление vP и нижней оценки.

Шаг 82 (8). 65 меньше 82, поэтому записываем новое значение нижней границы и номер гнезда.

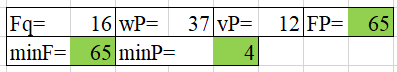


Рис. 46. Проверка рекорда.

Шаг 83 (9). minP указывает на гнездо 4, гнездо с минимальной нижней границей. Третью вершину располагаем в гнездо 4.

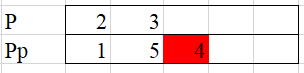
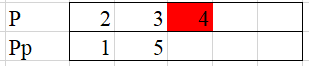


Рис. 47. Расположение третьей вершины.

Шаг 84 (10). Перешли к вершине 4.

Шаг 85 (11). Р не пуст, идём по циклу дальше.

Шаг 86 (2). Обнулим minF для следующего уровня ветвления.

Шаг 87 (3). В Р остались гнёзда 2, 3. Берём гнездо 2.

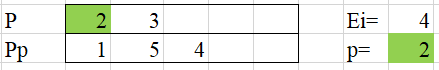


Рис. 48. Выбор следующего гнезда для установки.

Шаг 88 (4). Обнуляем значения.

Шаг 89 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 4 вершины в размещении, первая, вторая, третья и четвёртая, и вес ребра между первой и второй вершиной – 2, а между второй и третьей – 8, между третьей и четвёртой – 10, между четвёртой и первой - 3. Других рёбер нет. Расстояние между позициями размещения, соответственно – 4, 1, 2, 1. Итого – 39.

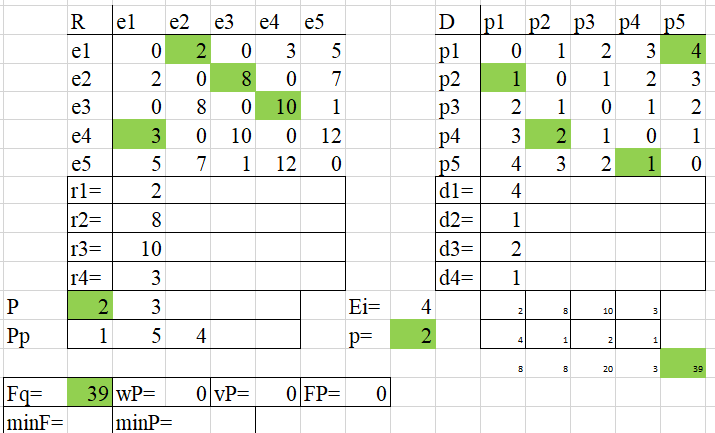


Рис. 49. Вычисление Fq.

Шаг 90 (6). Находим четыре вектора r (расстояния от первой, второй, третьей и четвёртой вершин до всех остальных) и четыре вектора d (расстояние от первого, пятого, четвёртого и второго гнёзд до остальных), отсортированные. Распределение векторов такое: вершина 1 находится в гнезде 1, поэтому ей соответствует вектор из гнезда 1, вершина 2 находится в гнезде 5, ей соответствует вектор из гнезда 5, и т.д. После мы их скалярно перемножили, получив в итоге 37.

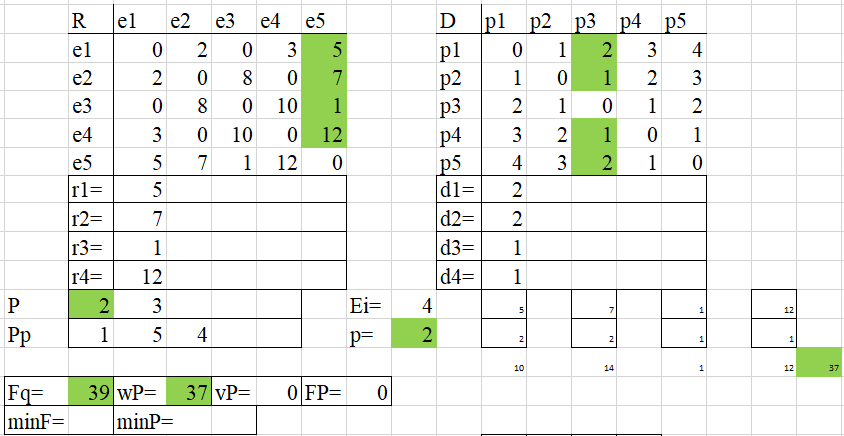


Рис. 50. Вычисление wP.

Шаг 91 (7). Зачеркнув строки и столбцы, соответствующие первой, второй, третьей и четвёртой вершине и первому, пятому, четвёртому и второму гнезду, не получаем векторов. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

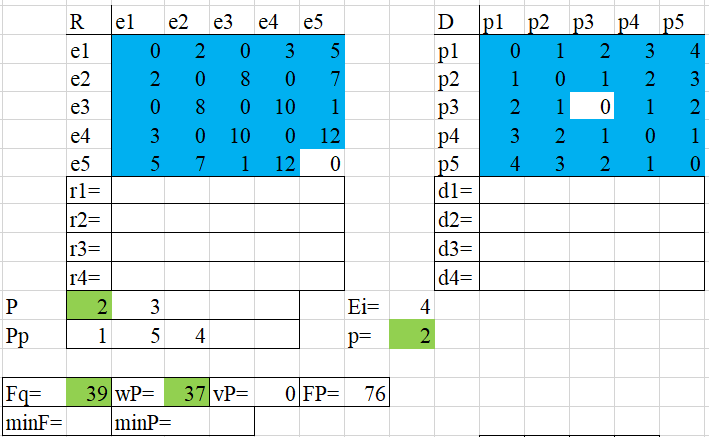


Рис. 51. Вычисление vP и нижней оценки.

Шаг 92 (8). 76 – первое значение, поэтому записываем его и вершину 2.

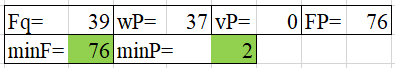


Рис. 52. Проверка рекорда.

Шаг 93 (3). Возвращаемся к шагу 3, продолжаем брать значения из P, следующее – 3. Вершины – первая, вторая, третья и четвёртая.

Шаг 88 (4). Обнуляем значения.

Шаг 89 (5). Так как Рр не пуст, будем высчитывать Fq. У нас 4 вершины в размещении, первая, вторая, третья и четвёртая, и вес ребра между первой и второй вершиной – 2, а между второй и третьей – 8, между третьей и четвёртой – 10, между четвёртой и первой - 3. Других рёбер нет. Расстояние между позициями размещения, соответственно – 4, 1, 1, 2. Итого – 32.

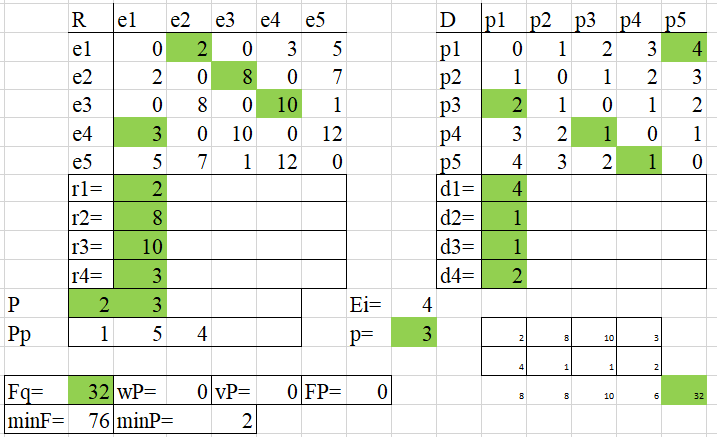


Рис. 53. Вычисление Fq.

Шаг 90 (6). Находим четыре вектора r (расстояния от первой, второй, третьей и четвёртой вершин до всех остальных) и четыре вектора d (расстояние от первого, пятого, четвёртого и третьего гнёзд до остальных), отсортированные. Распределение векторов такое: вершина 1 находится в гнезде 1, поэтому ей соответствует вектор из гнезда 1, вершина 2 находится в гнезде 5, ей соответствует вектор из гнезда 5, и т.д. После мы их скалярно перемножили, получив в итоге 37.

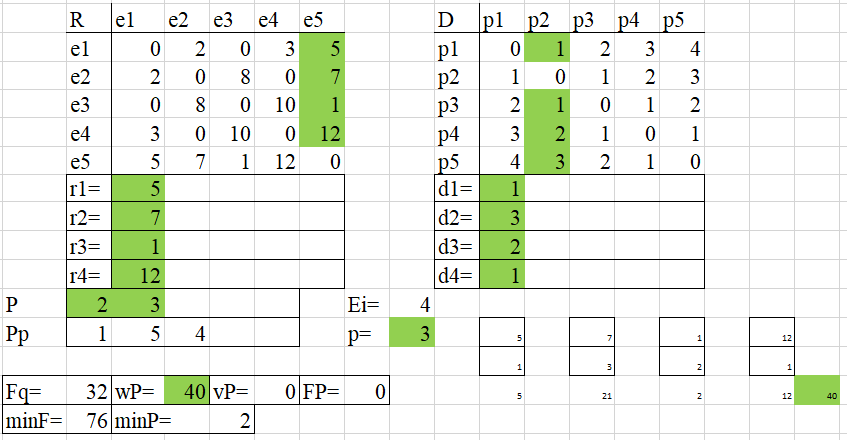


Рис. 54. Вычисление wP.

Шаг 91 (7). Зачеркнув строки и столбцы, соответствующие первой, второй, третьей и четвёртой вершине и первому, пятому, четвёртому и третьему гнезду, не получаем векторов. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

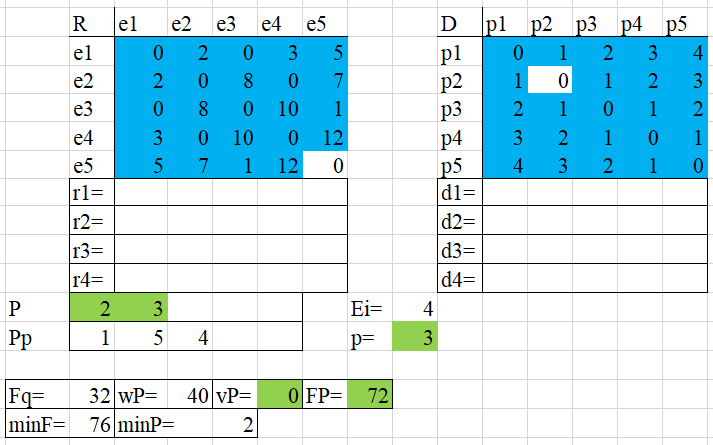


Рис. 55. Вычисление vP и нижней оценки.

Шаг 92 (8). 72 меньше чем 76, поэтому переписываем наименьшее значение функции и соответствующее гнездо.

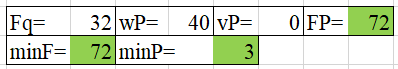


Рис. 56. Проверка рекорда.

Шаг 93 (9). minP указывает на гнездо 3, гнездо с минимальной нижней границей. четвёртую вершину располагаем в гнездо 3.

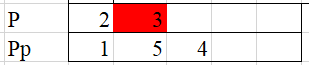
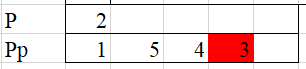
 

Рис. 57. Установка четвёртой вершины в гнездо 3.

Шаг 94 (10). Перешли к вершине 5.

Шаг 95 (11). Р не пуст, идём по циклу дальше.

Шаг 96 (2). Обнулим minF для следующего уровня ветвления.

Шаг 97 (3). В Р осталось гнездо 2. Берём гнездо 2.

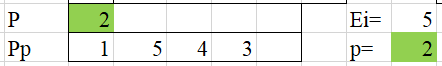


Рис. 58. Берём следующее гнездо для размещения.

Шаг 98 (4). Обнуляем значения.

Шаг 99 (5). Так как Рр уже не пуст, будем высчитывать Fq. У нас 5 вершины в размещении, первая, вторая, третья, четвёртая и пятая, и вес ребра между первой и второй вершиной – 2, а между второй и третьей – 8, между третьей и четвёртой – 10, между четвёртой и первой – 3, между четвёртой и пятой – 12, между пятой и первой – 5, между пятой и второй – 7, между пятой и третьей - 1. Это все рёбра графа, потому что размещены все вершины. Расстояние между позициями размещения, соответственно – 4, 1, 1, 2, 1, 1, 3, 2. Итого – 72.

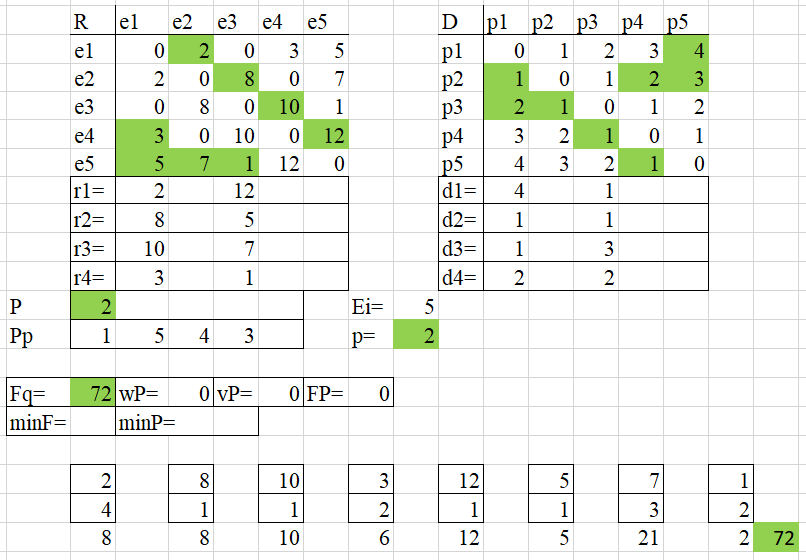


Рис. 59. Вычисление Fq.

Шаг 100 (6). Все вершины размещены, поэтому нижняя оценка расстояний между размещёнными и неразмещенными равна нулю.

Шаг 101 (7). Все вершины размещены, поэтому нижняя оценка расстояний между неразмещёнными равна нулю. Далее сложим полученные Fq, wP, vP, чтобы получить нижнюю оценку FP.

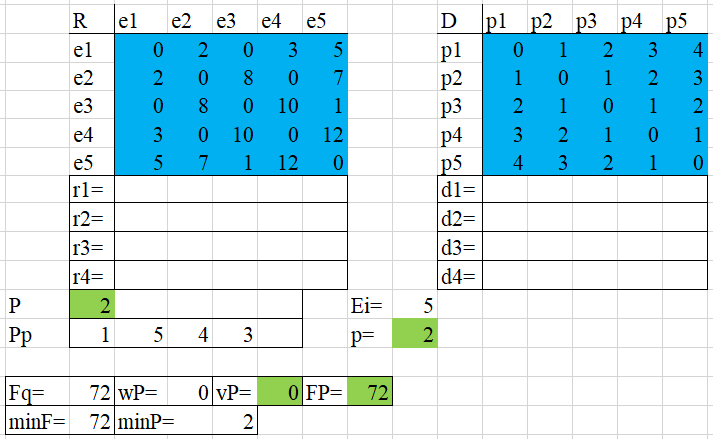


Рис. 60. Вычисление vP и нижней оценки.

Шаг 102 (8). 72 – первое значение, поэтому записываем его и вершину 2.



Рис. 61. Проверка рекорда.

Шаг 103 (9). minP указывает на гнездо 2, гнездо с минимальной нижней границей. четвёртую вершину располагаем в гнездо 3.

Рис. 62. Установка последней вершины во второе гнездо.

Шаг 104 (10). Перешли к вершине 6.

Шаг 105 (11). Р пуст.

Шаг 106 (12). Искомое размещение получено.



Рис. 63. Искомое размещение.

# **4. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ РАЗРАБОТАННОГО АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ**

На рисунке 4 представлена диаграмма классов разработанного приложения.

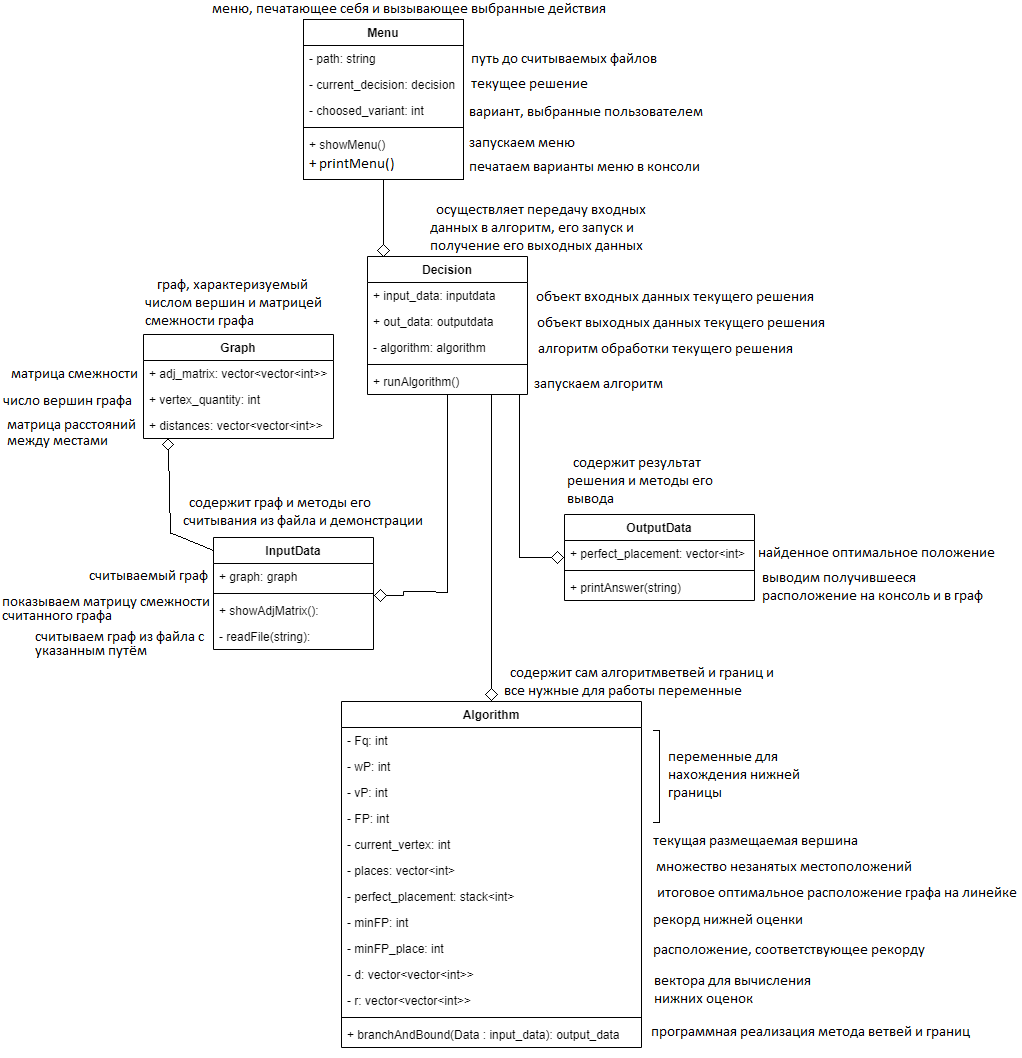


Рис. 64. Диаграмма классов

# **5. ОЦЕНКА ВРЕМЕННОЙ СЛОЖНОСТИ**

В таблице 2 приведены расчёты количества операций в программной реализации.

Таблица 2. Расчёт количества операций метода branchAndBound.

|  |  |  |
| --- | --- | --- |
| Т | Операторы | Количество выполнений |
| 1  2  3  4  5  6  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | for (int i = 1; i <= input\_data.graph.vertex\_quantity; i++){  places.push\_back(i);}  current\_vertex = 1;int d\_index;  do{  min\_FP = INT\_MAX;  for (auto current\_place = places.begin();  current\_place != places.end(); ++current\_place)  {perfect\_placement.push\_back(\*current\_place); {  FP = Fq = wP = vP = 0;  r.clear();d.clear();  if (perfect\_placement.empty()){  Fq = 0;}  Else{  for (int i = 0; i < perfect\_placement.size(); i++){  for (int j = 0; j < perfect\_placement.size(); j++){  if (j > i) {  Fq += input\_data.graph.adjacency\_matrix[i][j] \* input\_data.graph.distance\_matrix[perfect\_placement[i]-1]  [perfect\_placement[j] -1];}}}}  for (int i = 0; i < current\_vertex; i++){  r.push\_back(vector<int> ());  for (int j = current\_vertex; j < input\_data.graph.vertex\_quantity; j++){  r[i].push\_back(input\_data.graph.adjacency\_matrix[i] [j]);}}  d\_index = 0;  for (auto place1 = perfect\_placement.begin();  place1 != perfect\_placement.end(); ++place1){  d.push\_back(vector<int>());  for (auto place2 = places.begin();  place2 != places.end(); ++place2){  if (\*place2 != \*current\_place){  d[d\_index].push\_back(input\_data.graph.  distance\_matrix[\*place1-1][\*place2-1]);}}  d\_index++;}  for (int q = 0; q < r.size(); q++){  sort(r[q].begin(), r[q].end(), descendance);  sort(d[q].begin(), d[q].end());}  for (int q = 0; q < r.size(); q++){  for (int j = 0; j < r[0].size(); j++){  wP += r[q][j] \* d[q][j]; }}  r.clear();  d.clear();  r.push\_back(vector<int>());  for (int i = current\_vertex; i <  input\_data.graph.vertex\_quantity; i++){  for (int j = i + 1; j < input\_data.graph.vertex\_quantity;  j++){  r[0].push\_back(input\_data.graph.adjacency\_matrix  [i][j]); }}  d.push\_back(vector<int>());  for (auto place1 = places.begin();  place1 != places.end(); ++place1) {  for (auto place2 = places.begin();  place2 != places.end(); ++place2){  if (\*place1 != \*current\_place && \*place2 !=  \*current\_place && \*place2 > \*place1){  d[0].push\_back(input\_data.graph.distance\_matrix  [\*place1-1][\*place2-1]); }}}  sort(r[0].begin(), r[0].end(), descendance);  sort(d[0].begin(), d[0].end());  for (int j = 0; j < r[0].size(); j++){  vP += r[0][j] \* d[0][j]; }  FP = Fq + wP + vP;  if (FP < min\_FP) {  min\_FP = FP;  min\_FP\_place = \*current\_place;}  perfect\_placement.pop\_back();}  auto tmp = places.begin  for (auto current\_place = places.begin();  current\_place != places.end(); ++current\_place) {  if (\*current\_place == min\_FP\_place) {  tmp = current\_place; }}  perfect\_placement.push\_back(\*tmp);  places.erase(tmp);  current\_vertex++;  } while (!places.empty());  return perfect\_placement; | 1  N  1  N  N  (N+1)N/2~N^2  N^2  N^2  N^2  N  N^2  ~N^3  ~N^4  ~N^4  ~N^4  ~N^3  ~N^3  ~N^4  ~N^4  N^2  ~N^3  ~N^3  ~N^4  ~N^4  ~N^4  ~N^4  ~N^3  ~N^3  ~N^3  ~N^3  ~N^4  ~N^4  N^2  N^2  N^2  ~N^3  ~N^4  ~N^4  N^2  ~N^3  ~N^4  ~N^4  ~N^4  N^2  N^2  ~N^3  ~N^4  N^2  N^2  N^2  N^2  N^2  N  N^2  N^2  N  N  N  N  N  1 |

По найденному количеству выполнения операций составим функцию трудоёмкости:

T(n) = T1 + T2 + T3+ …. + T61 = 3 \* 1 + 10 \* N + 19 \* N^2 + 12 \* N^3 + 17 \* N^4 ~ 17 \* N^4 ~ N^4.

Таким образом, асимптотическая оценка временной сложности алгоритма – O(n4).

# **ЗАКЛЮЧЕНИЕ**

В процессе работы была изучена задача размещения вершин графа на линейке, на её основе составлены содержательная и математическая постановка задачи.

Также был изучен метод ветвей и границ, составлена его программная реализация. На её основе была получена оценка временной сложности равная O(n4), что не является эффективным. Само оформление алгоритма, реализация его и, упомяну отдельно, вычисление нижней границы оказались достаточно трудоёмкими.

Программная реализация была составлена на языке C++. В ходе написания активно использовалось объектно-ориентированное программирование, что очень помогло со структуризацией кода, его легкочитаемостью и последующими правками. Входными данными программы был текстовый файл с указанием числа вершин и всех рёбер с их весом и инцидентными вершинами, что упростило дальнейший тестинг программы.

# **СПИСОК ИСТОЧНИКОВ**

1. Магистерская диссертация на тему «Алгоритмизация труднорешаемой задачи размещения графа». В.А. Дудников, науч.рук. – О.В. Лелонд, Тольяттинский государственный университет, 2018 – 91 с.
2. Алгоритмы конструкторского проектирования ЭВМ. Учебное пособие по дисциплине «Конструкторско-технологическое обеспечение производства ЭВМ». Зыков А.Г., Поляков В.И. – СПб: Университет ИТМО, 2014. – 136 с.
3. ВУЗ Экспонента [Электронный ресурс]/ URL: <http://vuz.exponenta.ru/PDF/book/bm71.pdf> (дата обращения: 22.12.2020г.)
4. Генетический алгоритм для решения оптимизационной задачи размещение вершин графа на линейке [Электронный ресурс]/ IT factorial / URL: <https://it-factorial.ru/articles/geneticheskij-algoritm-dlya-resheniya-optimizatsionnoj-zadachi-razmeshhenie-vershin-grafa-na-linejke/> (дата обращения: 22.12.2020г.)
5. Метод ветвей и границ [Электронный ресурс]/ Википедия URL:https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4\_%D0%B2%D0%B5%D1%82%D0%B2%D0%B5%D0%B9\_%D0%B8\_%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%86 (дата обращения: 22.12.2020г.).
6. Руководство по C++ [Электронный ресурс]/ METANIT.COM URL: https://metanit.com/cpp/tutorial/1.1.php (дата обращения: 22.12.2020г.).

# **ПРИЛОЖЕНИЕ A. ЭКРАННЫЕ ФОРМЫ**

Далее, на рисунках, представлены варианты консольного меню.

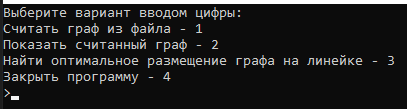


Рис. 65. Изначальный вид меню

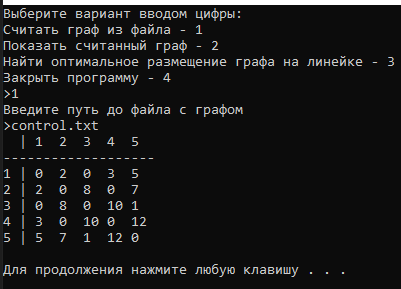


Рис. 66. Вид меню при считывании графа из файла

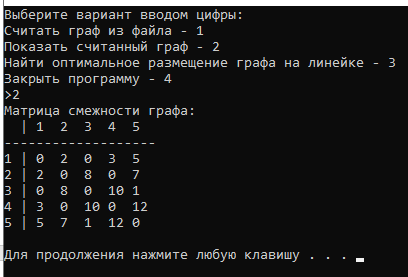


Рис. 67. Вид меню если запросить считанный граф

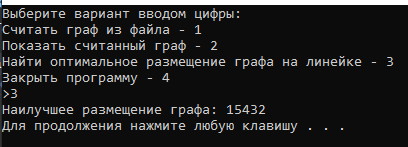


Рис. 68. Вид меню после осуществления решения задачи

# **ПРИЛОЖЕНИЕ Б. ЛИСТИНГ ПРОГРАММЫ**

#include <iostream>

#include <fstream> // для ввода-вывода файлов

#include <cmath> // берём отсюда функцию модуля числа

#include <string> // используем для обработки пути к файлу

#include <vector> // удобный динамический массив

#include <climits> // для получения максимального значения int

#include <algorithm> // содержит метод сортировки векторов

using namespace std;

// Примечание: для переменных используем нижнее подчёркивание, для функций -

// camelCase с маленькой буквы, а для классов - CamelCase с большой буквы

//прототипы, чтобы не было конфликтов с очередностью

class Graph;

class InputData;

class OutputData;

class Algorithm;

class Decision;

class Menu;

bool descendance(int a, int b) { return (a > b); } // для работы сортировки векторов

// граф, содержащит число вершин и матрицу смежности

class Graph

{

public:

int vertex\_quantity;

vector<vector<int>> adjacency\_matrix;

vector<vector<int>> distance\_matrix;

};

// содержит граф и методы по считыванию его из файла

class InputData

{

public:

Graph graph;

InputData(string path) // чтобы инициализировать сразу со считанным графом

{

readFile(path);

showAdjMatrix();

}

InputData() {}; // конструктор по умолчанию, для правильного функционирования

void showAdjMatrix() // выводим матрицу смежности для наглядности, с простой разметкой

{

cout << " | ";

for (int i = 0; i < graph.vertex\_quantity; i++) {

cout << i+1 << " ";

if (i / 9 == 0)

{ // делается для того, чтобы выровнять двуразрядные и одноразрядные числа

cout << " "; // пожалуйста, не вводите в файл трёхразрядные числа(

} // выводим номера вершин

}

cout << endl;

for (int i = 0; i < 3 \* graph.vertex\_quantity + 4; i++) cout << "-";

cout << endl; // горизонтальная черта

for (int i = 0; i < graph.vertex\_quantity; i++)

{

if (i / 9 == 0)

{

cout << i+1 << " | ";

}

else cout << i+1 << "| "; // выводим номера вершин

for (int j = 0; j < graph.vertex\_quantity; j++) {

cout << graph.adjacency\_matrix[i][j] << " ";

if (graph.adjacency\_matrix[i][j] / 9 == 0)

{

cout << " ";

}

}

cout << endl;

}

cout << endl;

}

private:

// считываем из файла данные о графе, генерируем матрицы смежности и расстояний

void readFile(string path)

{

/\*

1. создаём поток и переменные для работы с файлом

2. формируем матрицы нужного размера, так как создаются он безразмерным

3. в цикле, пока не закончится файл, считываем рёбра в виде инцидентных вершин и веса

\*/

ifstream in\_file(path); // удобный поток вывода из файла

if (in\_file.is\_open())

{

int vertex1, vertex2, weight;

in\_file >> graph.vertex\_quantity;

for (int i = 0; i < graph.vertex\_quantity; i++)

{

graph.adjacency\_matrix.push\_back(vector<int>(graph.vertex\_quantity, 0));

}

for (int i = 0; i < graph.vertex\_quantity; i++)

{

graph.distance\_matrix.push\_back(vector<int>(graph.vertex\_quantity));

for (int j = 0; j < graph.vertex\_quantity; j++)

{

graph.distance\_matrix[i][j] = abs(i - j);

// так как индексы обозначают вершины, расстояния легко найти модулем разности

}

}

while (in\_file >> vertex1 >> vertex2 >> weight)

{

if (vertex1 < 1 || vertex1 > graph.vertex\_quantity ||

vertex2 < 1 || vertex2 > graph.vertex\_quantity)

{ // если вершина с индексом меньше единицы или больше максимальной, то что-то не так

cout << "В файле есть вершины, которых быть не должно" << endl;

}

else

{

graph.adjacency\_matrix[vertex1 - 1][vertex2 - 1] =

graph.adjacency\_matrix[vertex2 - 1][vertex1 - 1] = weight;

// делаем так, потому что вершины в ребрах будут записаны в произвольном порядке

// а вычитаем единицу потому, что вершины графа начинаются c 1, что не удобно с 1

}

}

in\_file.close();

}

else throw exception(); // отправляем ошибку, если файл не считан,

} // потому что сама программа её не генерирует

};

// содержит результаты решения и методы для его вывода

class OutputData

{

public:

vector<int> perfect\_placement;

void printAnswer(string path) // выводит результат решения на консоль и в файл

{

/\*

1. создаём поток для работы с файлом

2. создаём изменённый вектор расположения, так как, что в perfect\_placement вершины - это

индексы, а элементы - места, а для наглядности нужно наоборот

3. выгружаем вектор в файл и в консоль

\*/

ofstream out\_file("output.txt", ios::app); // ios::app - чтобы запись шла в конец файла

out\_file << endl << "Наилучшее размещение графа из файла " << path << " :";

cout << "Наилучшее размещение графа: ";

vector<int> fixed\_placement(perfect\_placement.size());

int i = 1;

for (auto current\_place = perfect\_placement.begin();

current\_place != perfect\_placement.end(); ++current\_place)

{

fixed\_placement[\*current\_place - 1] = i; // не забываем про исправление индекса мест

i++;

}

for (auto current\_vertex = fixed\_placement.begin();

current\_vertex != fixed\_placement.end(); ++current\_vertex)

{

cout << \*current\_vertex;

out\_file << \*current\_vertex;

}

cout << endl;

out\_file << endl;

out\_file.close(); // закрываем файл

}

};

// содержит

class Algorithm

{

private:

int Fq, wP, vP, FP;

vector<int> places;

vector<int> perfect\_placement; // здесь элементы - это положения, а вершины, которые находятся

vector<vector<int>> d; // в этом положении - это соответствующий индекс + 1

vector<vector<int>> r;

int current\_vertex;

int min\_FP, min\_FP\_place;

public:

Algorithm() {}

vector<int> branchAndBound(InputData input\_data) // собственно, алгоритм ветвей и границ

{

/\*

Алгоритм представлен в пояснительной записке

\*/

for (int i = 1; i <= input\_data.graph.vertex\_quantity; i++)

{

places.push\_back(i);

}

current\_vertex = 1;

int d\_index; // переменная для функционирования вычисления векторов d

do

{

min\_FP = INT\_MAX;

for (auto current\_place = places.begin();

current\_place != places.end(); ++current\_place)

{

perfect\_placement.push\_back(\*current\_place); // добавляем текущую вершину в

// размещённые, чтобы итерировать

// вместе с ней

FP = Fq = wP = vP = 0;

r.clear(); // очищаем массивы векторов,

d.clear(); // чтобы на каждом витке цикла заполнять заново

if (perfect\_placement.empty())

{

Fq = 0;

}

else

{

// находим Fq

for (int i = 0; i < perfect\_placement.size(); i++)

{

for (int j = 0; j < perfect\_placement.size(); j++)

{

if (j > i) // чтобы учитывать только часть матрицы выше диагонали

{

Fq += input\_data.graph.adjacency\_matrix[i][j] \*

input\_data.graph.distance\_matrix[perfect\_placement[i]-1]

[perfect\_placement[j] -1];

}

}

}

}

// находим wP

// 1) находим r

for (int i = 0; i < current\_vertex; i++)

{ // находим на пересечении размещённых и неразмещённых вершин,

//а размещаем мы по порядку

r.push\_back(vector<int> ()); // будем создавать столько векторов, сколько

// размещено вершин

for (int j = current\_vertex; j < input\_data.graph.vertex\_quantity; j++)

{

r[i].push\_back(input\_data.graph.adjacency\_matrix[i][j]);

}

}

// 2) находим d

d\_index = 0; // мы итерируем по вектору вместо обычного for, поэтому нам нужен

//искуственный индекс

for (auto place1 = perfect\_placement.begin();

place1 != perfect\_placement.end(); ++place1)

{

d.push\_back(vector<int>());

for (auto place2 = places.begin();

place2 != places.end(); ++place2)

{

if (\*place2 != \*current\_place)

{

d[d\_index].push\_back(input\_data.graph.distance\_matrix

[\*place1-1][\*place2-1]);

}

}

d\_index++;

}

// 3) сортируем

for (int q = 0; q < r.size(); q++)

{

sort(r[q].begin(), r[q].end(), descendance); // по убыванию, нужна функция

sort(d[q].begin(), d[q].end()); // которую объявили вначале

}

// 4) скалярно перемножаем

for (int q = 0; q < r.size(); q++)

{

for (int j = 0; j < r[0].size(); j++)

{

wP += r[q][j] \* d[q][j];

}

}

// находим vP

r.clear();

d.clear();

// 1) находим r

r.push\_back(vector<int>()); // здесь будет только один вектор у d и у r

for (int i = current\_vertex; i < input\_data.graph.vertex\_quantity; i++)

{

for (int j = i + 1; j < input\_data.graph.vertex\_quantity; j++)

{

r[0].push\_back(input\_data.graph.adjacency\_matrix[i][j]);

}

}

// 2) находим d

d.push\_back(vector<int>());

for (auto place1 = places.begin();

place1 != places.end(); ++place1)

{

for (auto place2 = places.begin();

place2 != places.end(); ++place2)

{

if (\*place1 != \*current\_place && \*place2 != \*current\_place &&

\*place2 > \*place1)

{

d[0].push\_back(input\_data.graph.distance\_matrix[\*place1-1][\*place2-1]);

}

}

}

// 3) сортируем

sort(r[0].begin(), r[0].end(), descendance);

sort(d[0].begin(), d[0].end());

// 4) скалярно перемножаем

for (int j = 0; j < r[0].size(); j++)

{

vP += r[0][j] \* d[0][j];

}

FP = Fq + wP + vP;

if (FP < min\_FP)

{

min\_FP = FP;

min\_FP\_place = \*current\_place;

}

perfect\_placement.pop\_back(); // удаляем текущую вершину,

// так как она там только гостила

}

auto tmp = places.begin(); // ввожу временный итератор, потому что цикл сломется, если

// удалить элемент из вектора и продолжить цикл

for (auto current\_place = places.begin();

current\_place != places.end(); ++current\_place)

{

if (\*current\_place == min\_FP\_place)

{

tmp = current\_place;

}

}

perfect\_placement.push\_back(\*tmp);

places.erase(tmp);

current\_vertex++;

} while (!places.empty());

return perfect\_placement;

}

};

// класс берёт входные данные, помещает в алгоритм, ответ алгоритма записывает во выходные данные

class Decision

{

private:

Algorithm algorithm;

public:

InputData input\_data;

OutputData output\_data;

Decision() {}

void runAlgorithm()

{

output\_data.perfect\_placement = algorithm.branchAndBound(input\_data);

}

};

// меню, печатающее себя и вызывающее соответствующие действия

class Menu

{

private:

string path;

Decision current\_decision;

int choosed\_variant;

void printMenu() // выводит пользователю варианты

{

system("cls"); // очищаем экран, чтобы не копился лишний текст

cout << "Выберите вариант вводом цифры: " << endl;

cout << "Считать граф из файла - 1" << endl;

cout << "Показать считанный граф - 2" << endl;

cout << "Найти оптимальное размещение графа на линейке - 3" << endl;

cout << "Закрыть программу - 4" << endl;

cout << ">";

}

public:

Menu()

{

showMenu();

}

void showMenu()

{

int is\_inputed = 0; // чтобы алгоритм не запустился, пока не будет считан файл

do {

printMenu();

string inputed\_variant;

cin >> inputed\_variant;

while (sscanf\_s(inputed\_variant.c\_str(), "%d", &choosed\_variant) != 1 ||

choosed\_variant < 1 || choosed\_variant > 4)

// преобразуем

{ // пока пользователь не введёт правильный вариант, считываем новый

cout << "Вариант не найден, попробуйте снова" << endl;

cout << ">";

cin >> inputed\_variant;

}

switch (choosed\_variant)

{ // для выбора подзадачи

case 1:

cout << "Введите путь до файла с графом" << endl;

cout << ">";

cin >> path; // пользователь вводит путь к файлу

try { // нужно для того, чтобы отловить неверный ввод пути к файлу

current\_decision = Decision();

current\_decision.input\_data = InputData(path);

is\_inputed++; // если путь правильный, то укажем,

} // что граф мы уже считали и можно произвоить с ним вычисления

catch (...) {

cout << "С файлом какая-то ошибка, попробуйте снова" << endl;

}

break;

case 2:

if (is\_inputed)

{

cout << "Матрица смежности графа:" << endl;

current\_decision.input\_data.showAdjMatrix();

}

else cout << "Для начала считайте граф!" << endl;

break;

case 3:

if (is\_inputed)

{

current\_decision.runAlgorithm();

current\_decision.output\_data.printAnswer(path);

}

else cout << "Для начала считайте граф!" << endl;

break;

}

if (choosed\_variant != 4)

system("pause"); // если этого не сделать, пользователь не увидит результат

} while (choosed\_variant != 4); // выходим из программы, если пользователь выбрал выход

cout << "До свидания!" << endl;

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

Menu menu = Menu();

}