МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра компьютерных технологий и систем

КАНАШЕВИЧ
Кирилл Алексеевич

РЕАЛИЗАЦИЯ ИНДЕКСНЫХ И БАЗИСНЫХ МЕТОДОВ РЕШЕНИЯ ДВУХКРИТЕРИАЛЬНЫХ ЗАДАЧ ПОТОКОВОГО ПРОГРАММИРОВАНИЯ

Дипломная работа

Научный руководитель:
кандидат физ.-мат. наук,
доцент Л. А. Пилипчук

Допущен к защите
«__» _____2025 г.
Зав. кафедрой компьютерных технологий и систем
Доктор педагогических наук, профессор В.В. Казаченок

Минск, 2025

ОГЛАВЛЕНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

$\in$ - принадлежность множеству

$\subseteq$ - подмножество

$\bigcup$ - объединение множеств

$\setminus$ - разность множеств

$\square$ - конец доказательства

$|I|$ - количество элементов в I

$\mathbf{R}$ - множество рациональных чисел

$\mathbb{N}$ - множество натуральных чисел

РЕФЕРАТ

Дипломная работа, 50 с., 39 рис., 16 источников, 3 приложения.

Ключевые слова: ДВУХКРИТЕРИАЛЬНЫЕ ЗАДАЧИ, ГЕНЕРАТОР ГРАФОВ, ВИЗУАЛИЗАЦИЯ ГРАФА, ПОТОКОВОЕ ПРОГРАММИРОВАНИЕ, ПОСТРОЕНИЕ КРАТЧАЙШЕГО ПУТИ.

Объект исследования: двухкритериальные задачи, в частности поиск пути минимальной стоимости, максимальной ширины.

Цель работы: реализация индексных и базисных методов решения двухкритериальных задач, сравнительный анализ различных алгоритмов.

Методы исследования: математические модели, технологии визуализации.

Результаты: реализованы базисный и индексный алгоритмы решения двухкритериальных задач, реализован генератор графов, проведен анализ полученных решений, предложены варианты использования данных методов.

ABSTRACT

Thesis, 50 pages, 39 pictures, 16 sources, 3 tables,

Keywords: TWO-CRITERIA PROBLEMS, GRAPH GENERATOR, GRAPH VISUALIZATION, FLOW PROGRAMMING, SHORTEST PATH CONSTRUCTION.

Object of study: research: two-criteria problems, in particular, the search for a path of minimum cost, maximum width.

Purpose of work: realization of index and basis methods of solving two-criteria problems, a comparative analysis of different algorithms.

Research methods: mathematical models, visualization technologies.

Results: the basis and index algorithms for solving two-criteria problems are implemented, the graph generator is realized, the obtained solutions are analyzed, and variants of using these methods are proposed.

РЭФЕРАТ

Дыпломная праца, 50 с., 39 іл., 16 крыніц, 3 прыкладанні.

Ключавыя словы: ДВУХКРЫТЭРЫЯЛЬНЫЯ ЗАДАЧЫ, ГЕНЕРАТАР ГРАФАЎ, ВІЗУАЛІЗАЦЫЯ ГРАФА, ПОТОКОВОЕ ПРАГРАМАВАННЕ, ПАБУДАВАННЕ НАЙКАРАЦЕЙШАГА ШЛЯХУ.

Аб'ект даследавання: двухкрытэрыяльныя задачы, у прыватнасці пошук шляху мінімальнай кошту, максімальнай шырыні.

Мэта працы: рэалізацыя індэксных і базісных метадаў рашэння двухкрытэрыяльных задач, параўнальны аналіз розных алгарытмаў.

Метады даследавання: матэматычныя мадэлі, тэхналогіі візуалізацыі.

Вынікі: рэалізаваны базісны і індэксны алгарытмы рашэння двухкрытэрыяльных задач, рэалізаваны генератар графаў, праведзены аналіз атрыманых рашэнняў, прапанаваны варыянты выкарыстання дадзеных метадаў.

ВВЕДЕНИЕ

Современные задачи оптимизации всё чаще сталкиваются с необходимостью учета сразу нескольких критериев, что существенно усложняет их решение. Классические методы оптимизации, ориентированные преимущественно на решение задач с одним критерием, в данной ситуации оказываются не столь эффективными. Потоковое программирование, применяемое для моделирования процессов и потоков в транспортных, производственных и логистических сетях, выступает как перспективная область для решения данной проблемы. Методы решения двухкритериальных задач потокового программирования предназначены для поиска решений, которые сочетают в себе различные критерии, например, поиск пути минимальной стоимости, максимальной ширины.

Существует широкий спектр методов для решения задач потокового программирования. Среди них выделяются индексные и базисные методы, которые отличаются принципами построения решений, сложностью алгоритмов и применимостью к различным видам задач. В рамках данной работы рассматриваются индексные и базисные методы решения двухкритериальных задач. Эти подходы отличаются алгоритмической сложностью и применимостью к различным видам задач, что определяет необходимость их детального анализа и сравнения. Индексные методы основываются на использовании весовых коэффициентов для объединения критериев в единый целевой показатель, тогда как базисные методы предполагают поиск базовых оптимальных решений с учётом каждого критерия по отдельности.

Актуальность работы обусловлена широким применением двухкритериальных задач в реальных системах управления потоками, где требуется учёт множества факторов для принятия оптимальных решений.

Результаты исследования могут быть использованы для повышения эффективности решения задач в таких областях, как логистика и производство.

ГЛАВА 1
ДВУХКРИТЕРИАЛЬНЫЕ ЗАДАЧИ

1.1 Постановка двухкритериальных задач

«Зададим конечный ориентированный связный граф $G = (I, U)$ с множеством узлов I и множеством дуг U, $|I| < \infty$, $|U| < \infty$. Каждая дуга $(i, j) \in U$ характеризуется следующими параметрами: $x_{i,j}$ – величина дугового потока, $c_{i,j}$ – стоимость перемещения единицы дугового потока $x_{i,j}$ по дуге (i, j), $d_{i,j}$ – пропускная способность (ширина) дуги (i, j). Обозначим $x = (x_{i,j}, (i, j) \in U)$ – вектор дуговых потоков. Определим соответственно ширину $d(L_{s,t})$ и стоимость $c(L_{s,t})$ по пути $L_{s,t}$ из узла s в достижимый узел t следующим образом: » [3]

$$d\left(L_{s,t}\right) = \min\left\{d_{i,j}, (i,j) \in L_{s,t}\right\}, c\left(L_{s,t}\right) = \sum_{(i,j)\in L_{s,t}} c_{i,j}x_{i,j}. \quad (1.1)$$

Рассмотрим несколько различных двухкритериальных задач, для которых будем строить методы решения.

Задача 1. Построить пути максимальной ширины, из которых в последствии выбрать путь минимальной стоимости из вершины s в вершину t.

Задача 2. Необходимо найти путь из вершины s в достижимую вершину t который имеет минимальную стоимость (кратчайший), среди всех возможных путей, среди которых выбрать путь максимальной ширины.

Задача 3. Найти максимальный поток из s в t, имеющий при этом минимальную стоимость среди всех возможных максимальных потоков.

Все три задачи демонстрируют различные аспекты многокритериальной оптимизации и требуют разработки особых методов решения. Первые две задачи предполагают строгий приоритет между критериями, в то время как третья требует наилучшего результата сразу по обоим критериям.

1.2 Кратчайший путь в множестве путей максимальной ширины

Рассмотрим решение первой задачи.

«Математическая модель задачи поиска кратчайшего пути в графе $G = \{I, U\}$ между двумя узлами s и t имеет следующий вид:

$$\sum_{(i,j)\in U} c_{i,j}x_{i,j} \rightarrow min, \quad (1.2)$$

$$\sum_{j\in I_i^+(U)} x_{i,j} - \sum_{j\in I_i^-(U)} x_{j,i} = \begin{cases} 1,\ i = s, \\ -1,\ i = t \\ 0, i \in I\backslash\{s,t\}, \end{cases}, \quad (1.3)$$

$$x_{i,j} \in \{0,1\}, \{i,j\} \in U. \quad (1.4)$$

где $I_i^+(U) = \{j \in I: (i,j) \in U\}, I_i^-(U) = \{j \in I: (i,j) \in U\}$ » [3].

Для определенной задачи применим структурный подход связанный с поиском кратчайшего пути. Основу алгоритма составит формирование специальных графов, благодаря которым будут установлены ключевый свойства.

Рассмотрим более детальное описание алгоритма построения специальной структуры графа.

На начальном этапе осуществляется подготовка начальных данных. Для заданного графа G распределим дуги по множествам, относительно ширины, далее отсортируем множества по убыванию ширины.

Для каждого $i = 1, \dots, k$ сформируем множества дуг $\widetilde{U^i}$ следующим образом

$$\widetilde{U^i} = \bigcup_{j=1}^{i} U^j, \text{ где } i = 1, \dots, k. \quad (1.5)$$

«Теорема 1. Пусть $L_{s,t}^p$ – некоторый путь в графе $G^p = (I, \widetilde{U}^p)$ из узла s в достижимый узел t, где $p \in \{1, \dots, h\}$. Путь $L_{s,t}^p$ может быть не единственным в графе $G^p = (I, \widetilde{U}^p)$. Если для ширины каждого пути $L_{s,t}^1, \dots, L_{s,t}^p$ графа $G^p = (I, \widetilde{U}^p)$ выполняются условия

$$d\left(L_{s,t}^1\right) = d\left(L_{s,t}^2\right) =, \dots, = d\left(L_{s,t}^p\right) > d(L_{s,t}^{p+1}) \quad (1.6)$$

то $\{L_{s,t}^1, \dots, L_{s,t}^p\}$ – множество путей максимальной ширины из узла s в достижимый узел t в начальном графе $G = (I, U)$.

Доказательство. Строим графы $G^1 = (I, \widetilde{U}^1)$, $G^2 = (I, \widetilde{U}^2)$, ..., $G^p = (I, \widetilde{U}^p)$ до тех пор, пока не построим некоторый граф $G^p = (I, \widetilde{U}^p)$, для которого узел t достижим из узла s. Другими словами, пока не сформируем множество дуг $\widetilde{U}^p$ графа $G^p = (I, \widetilde{U}^p)$, в котором существует путь $L_{s,t}^p$ из узла s в узел t. По построению, путь $L_{s,t}^p$ в графе $G^p = (I, \widetilde{U}^p)$ имеет максимальную ширину $d(L_{s,t}^p)$ и может быть не единственным. Все пути в графе $G^p = (I, \widetilde{U}^p)$, из s в достижимый узел t имеют максимальную ширину, равную $d(L_{s,t}^p)$, и проходят только через те дуги, ширина которых не меньше ширины $d(L_{s,t}^p)$ самого широкого пути. Условия (1.6) определяют множество путей $\{L_{s,t}^1, \dots, L_{s,t}^p\}$

максимальной ширины $d(L_{s,t}^{p})$ из узла s в достижимый узел t в графе $G = (I, U)$. Следовательно, кратчайший путь $L_{s,t}^{0}$, построенный в графе $G^{p} = (I, \widetilde{U}^{p})$, имеет максимальную ширину и является кратчайшим путем максимальной ширины в начальном графе $G = (I, U)$. Теорема доказана » [3]. □

Формальная постановка задачи поиска кратчайшего пути $L_{s,t}^{0}$ может быть представлена, как задача бикритериальной оптимизации:

$$\sum_{(i,j)\in\widetilde{U}^{p}} \mathrm{c}_{i,j} x_{i,j} \to min, \tag{1.7}$$

$$\sum_{j\in I_i^{+}(\widetilde{U}^{p})} x_{i,j} - \sum_{j\in I_i^{-}(\widetilde{U}^{p})} x_{j,i} = \begin{cases} 1, & i = s, \\ -1, & i = t \\ 0, & i \in I\backslash\{s,t\}, \end{cases}, \tag{1.8}$$

$$x_{i,j} \in \{0,1\}, \{i,j\} \in \widetilde{U}^{p}. \tag{1.9}$$

где $I_i^{+}(\widetilde{U}^{p}) = \{j \in I: (i,j) \in \widetilde{U}^{p}\}$, $I_i^{-}(\widetilde{U}^{p}) = \{j \in I: (i,j) \in \widetilde{U}^{p}\}$.

Алгоритм поиска кратчайшего пути может быть реализован следующим образом.

На первом этапе инициализируем счетчик уровня, который будет отвечать за уровень множества. Затем, с помощью например, алгоритма Дейкстры попробуем построить путь из s в t для полученного графа. При нахождении пути, полагаем процесс завершенным, в ином случае увеличиваем значение счетчика на 1 и процесс проверки повторяется для следующего графа. Полученное при нахождении пути значение счетчика является максимальной шириной для пути минимальной стоимости, а построенный путь является решением поставленной задачи (1.2) - (1.4). Алгоритм гарантированно находит оптимальное решение при его существовании в графе, за конечное число шагов.

1.3 Путь максимальной ширины среди множества кратчайших путей

Для определения пути с максимальной шириной среди всех путей минимальной стоимости из узла s в узел t, рассмотрим следующий подход.

Проведем обработку исходного графа, при которой упорядочим все дуги по убыванию ширины. Это позволит последовательно анализировать граф по уровням ширины его ребер. Концептуально, происходит сужение множества допустимых решений, на каждом шагу формируется подграф, который и задает пространство поиска.

«Теорема 2. Пусть $L_{s,t}$ – кратчайший путь в графе $G^i = (I, \widetilde{U}^i)$ из узла s в достижимый узел t, где дуги множества $\widetilde{U}^i$ сформированы согласно (1.5), $i \in \{1, \dots, k\}$. Если выполняются условия

$$c\left(L_{s,t}^k\right) = c\left(L_{s,t}^{k-1}\right) = \cdots = c\left(L_{s,t}^l\right) < c(L_{s,t}^{l-1}) \quad (1.10)$$

то путь $L_{s,t}^l$ имеет наибольшую ширину среди всех кратчайших путей из узла s в достижимый узел t в графе $G = (I, U)$ » [3].

Распишем алгоритм для решения поставленной задачи. Во многом он будет похож на уже составленный для задачи 1.

Первоначально задаем счетчик $i = k$, что формально соответствует рассмотрению всего исходного графа G, со всеми его дугами. Далее применяем алгоритм Дейкстры для нахождения кратчайшего пути между 2 вершинами. Сохраняем значение стоимости полученного пути. После этого начинаем итерационный процесс. На каждой итерации уменьшаем счетчик на 1, и заново вычисляем кратчайший путь, сравнивая его с результатом предыдущего уровня. Продолжаем процесс пока стоимости совпадают. В момент, когда стоимость пути стала меньше, что соответствует теореме 2, фиксируем полученное значение i и завершаем итерационный процесс. Полученный путь является решением поставленной задачи. Данный подход обеспечивает систематический поиск оптимального решения, последовательно анализируя подграфы, постепенно переходя к меньшим, что гарантирует нахождение пути с наилучшими характеристиками. Основным преимуществом алгоритма является ясно определенная процедура поиска, которая гарантирует оптимальность полученного решения.

1.4 Максимальный поток минимальной стоимости

Рассмотрим задачу определения максимального потока минимальной стоимости.

Пусть задан граф G. Каждая дуга характеризуется 3 параметрами: величина потока по дуге, стоимость транспортировки одной единицы потока по дуге и пропускная способность. Так же определим 2 специальных узла, $s -$ источник и $t -$ сток. Причем исток не имеет входящих потоков, а сток не имеет исходящих. Зададим величину v, которая вычисляется как сумма всех потоков по всем дугам исходящим из истока.

«Математическая модель задачи определения максимального потока v в сети $G = \{I, U\}$ имеет следующий вид:

$$v \to max, \tag{1.11}$$

$$\sum_{j \in I_i^+(U)} x_{i,j} - \sum_{j \in I_i^-(U)} x_{j,i} = \begin{cases} v, & i = s, \\ -v, & i = t, \\ 0, & i \in I\{s,t\}, \end{cases} \tag{1.12}$$

$$0 \le x_{i,j} \le d_{i,j}, (i,j) \in U, \tag{1.13}$$

где $I_i^+(U) = \{j \in I: (i,j) \in U\}$, $I_i^-(U) = \{j \in I: (j,i) \in U\}$ » [3].

«Задача (1.11) – (1.13) определения максимального потока v в сети $G = \{I, U\}$ является частным случаем экстремальной задачи математического программирования вида

$$\sum_{(i,j) \in U} c_{i,j} x_{i,j} + \sum_{i \in I^*} c_i x_i \to min, \tag{1.14}$$

$$\sum_{j \in I_i^+(U)} x_{i,j} - \sum_{j \in I_i^-(U)} x_{j,i} = \begin{cases} x_i, & i \in I^*, \\ a_i, & i \in I \backslash I^*, \end{cases} \tag{1.15}$$

$$0 \le x_{i,j} \le d_{i,j}, (i,j) \in U, b_{*i} \le x_i \le b_i^*, i \in I^*, \tag{1.16}$$

при следующих предположениях:

$$I^* = \{s,t\} \subseteq I, c_{i,j} = 0, (i,j) \in U;\ c_s = -1, c_t = 0;$$
$$x_s = v, x_t = -v;\ a_i = 0, i \in I \backslash I^*. \text{» [3]} \tag{1.17}$$

Сформулируем начальное допустимое условие. В сети G выберем произвольный путь $L_{s,t}$ из источника в исток. Потоки на дугах этого пути устанавливаются равными его пропускной способности, определяемой как минимальная пропускная способность среди всех дуг пути. Остальные дуги имеют нулевой поток.

$$d\left(L_{s,t}\right) = \min\{d_{i,j}, (i,j) \in L_{s,t}\}, \tag{1.18}$$

Величина начального потока в сети v равна сумме потоков из источника.

$$v = \sum_{j \in I_s^+(U)} x_{s,j} = d(L_{s,t}) \tag{1.19}$$

Далее применяя методы декомпозиции, вычислительные алгоритмы и структурные решения, находим оптимальное решение.

$$b = \sum_{j \in I_s^+(U)} x_{s,j}^0 \;\; x^0 = (x_{i,j}^0, (i,j) \in U). \tag{1.20}$$

где $v^0 = b$ – величина максимального потока в сети $G = (I, U)$.

«Пусть известно численное значение b величины максимального потока в сети G. Математическая модель задачи нахождения потока минимальной стоимости максимальной величины b в сети $G = (I, U)$ имеет следующий вид:

$$f(x) = \sum_{(i,j) \in U} m_{i,j} x_{i,j}, \tag{1.21}$$

$$\sum_{j \in I_i^+(U)} x_{i,j} - \sum_{j \in I_i^-(U)} x_{j,i} = \begin{cases} b, & i = s, \\ -b, & i = t, \\ 0, i \in I \backslash \{s,t\}, \end{cases} \tag{1.22}$$

$$0 \le x_{i,j} \le d_{i,j}, (i,j) \in U, \tag{1.23}$$

где $\{s,t\} \subset I$, $x_{i,j}$ – дуговой поток дуги (i,j), $c_{i,j}$ – стоимость перемещения единицы дугового потока $x_{i,j}$ по дуге (i,j), $d_{i,j}$ – пропускная способность дуги (i,j) (максимальная величина дугового потока дуги (i,j)). Нижняя граница дугового потока каждой дуги $(i,j) \in U$ равна нулю, $x = (x_{i,j}, (i,j) \in U)$ – вектор дуговых потоков, b – численное значение величины максимального потока в сети G » [3].

«Теорема 3. Пусть $(x^0, v^0 = b)$ – оптимальное решение задачи (1.14) – (1.16), где $v^0 = b$ – численное значение (1.18) максимального потока в сети G. Если для любого допустимого решения $x = (x_{i,j}, (i,j) \in U)$ задачи (1.21) – (1.23) условия сохранения потока (1.21) для узлов $\{s,t\} \subset I$ выполняются для величины максимального потока b и для узлов $I\{s,t\}$ равны нулю, то оптимальное решение $\tilde{x}^0 = (\tilde{x}_{i,j}^0, (i,j) \in U)$ экстремальной задачи (1.21) – (1.23) является потоком минимальной стоимости $f(\tilde{x}^0)$ для максимального потока $v^0 = b$:

$$f(\tilde{x}^0) = \sum_{(i,j) \in U} c_{i,j} \tilde{x}_{i,j}^0 \qquad b = \sum_{j \in I_s^+(U)} \tilde{x}_{s,j}^0 \text{ » [3].} \tag{1.24}$$

Приведем алгоритм построения в сети $G = (I, U)$ максимального потока минимальной стоимости.

Первоначально производится инициализация параметров: задаются узлы источника и стока с потенциалами, устанавливаются начальные потоки, а остальные приравниваются к 0. Далее выбирается произвольный путь из s в t. Пропускная способность этого пути определяется как минимальная среди всех его дуг. Исходя из этого строится начальное допустимое решение. После этого задается величина начального потока и решается задача (1.14) – (1.16). Полученное решение используется как начальное приближение для следующей

задачи, где фиксируется права часть в соответствии с найденным максимальным потоком. В заключении находится оптимальный поток минимальной стоимости.

ГЛАВА 2
БАЗИСНЫЙ АЛГОРИТМ

2.1 Кратчайшие пути

Рассмотрим конечный ориентированный связный граф G с множеством узлов I и множеством дуг U, определенных на $I \times I$, $|I| < \infty$, $|U| < \infty$. Каждой дуге (v, w) орграфа G приписана стоимость (длина) $c_{v,w}$. Длина $c(L_{s,t})$ пути $L_{s,t} = \{s = v_0, v_1, \dots, v_{k-1}, v_k = t\}$ от узла $t \in I$ определяется следующим образом:

$$c\left(L_{s,t}\right) = \sum_{i=1}^{k} c_{v_{i-1}+v_i}. \quad (2.1)$$

«Математическая модель задачи о кратчайших путях из узла $s \in I$ в узлы $i \in I\backslash\{s\}$ графа G имеет вид:

$$\sum_{(i,j)\in U} c_{i,j}x_{i,j} \to min, \quad (2.2)$$

$$\sum_{j\in I_i^+(U)} x_{i,j} - \sum_{j\in I_i^-(U)} x_{j,i} = \begin{cases} n-1, i = s \\ -1, i \in I\backslash\{s\} \end{cases} \quad (2.3)$$

$$0 \le x_{i,j} \le n, (i,j) \in U, n = |I|, x_{i,j} \in \mathbb{N} \quad (2.4)$$

где $I_i^+(U) = \{j \in I: (i,j) \in U\}, I_i^-(U) = \{j \in I: (j,i) \in U\}$, $x_{i,j}$ – величина дугового потока дуги $(i,j), x = (x_{i,j}, (i,j) \in U)$ – вектор дуговых потоков, $c_{i,j}$ – стоимость перемещения по дуге единицы потока $x_{i,j}$ из узла i в узел j »[6].

Можно заметить, что решение задачи будет целочисленным, даже без явного указания условий целочисленности переменных. Исходя из этого можно упростить ограничения до неотрицательных потоков, и они примут вид

$$x_{i,j} \ge 0, (i,j) \in U, x_{i,j} \in \mathbb{N} \quad (2.5)$$

Данная модель гарантирует нахождение оптимальных кратчайших путей в графе.

2.2 Начальное допустимое решение

Построим начальное допустимое решение $x = (x_{i,j}, (i,j) \in U)$ экстремальной задачи (2.2), (2.4), (2.5), для которого выполняются ограничения (2.4), (2.5). Для дуг, не входящих в состав дерева $G_0 = (I, U_0)$ опорного потока $\{x, G_0\}$, величины дуговых потоков положим равными нулю: $x_{i,j} = 0, (i,j) \in U_N = U \backslash U_0$.

«Теорема 4. Пусть задано некоторое корневое дерево $G_0 = (I, U_0)$ с корнем в узле s и список $\{pred[i], i \in I\}$, который определяет для каждого узла $i \in I \backslash \{s\}$ значение предка $|pred[i]|$ (элемент биективного отображения узла i корневого дерева), $pred[s] = 0$. Для любого узла $i \in I \backslash s$ величина дугового потока $x_{pred[i]}$, каждой дуги $(pred[i], i) \in U_0$ равняется количеству узлов поддерева с корнем в узле i »[6].

Обозначим m_i – количество узлов поддерева с корнем в узле $i \in I$.

«Теорема 5. Теоретическая оценка сложности алгоритма нахождения последовательности узлов поддерева с корнем в узле i, основанная на применении структур данных: $\{thread[i], \forall i \in I\}$ и $\{depth[i], \forall i \in I\}$, равна $O(m_i), \forall i \in I$, где m_i – число узлов поддерева с корнем в узле i »[6].

Для построения начального решения первоначально выберем корневое дерево, для всех дуг, не входящих в состав этого дерева, значения потоков устанавливается равным нулю. Далее выполняется расчет потоков на дугах дерева.

Представленный подход гарантирует построение допустимого решения, удовлетворяющего всем ограничениям задачи.

2.3 Условия оптимальности

Рассматривая свойства опорного потока, важно отметить его невырожденный характер, который обусловлен выполнением строгих неравенств для потоков.

«Теорема 6. Для оптимальности опорного потока $\{x, G_0\}$ необходимо и достаточно выполнения следующих условий: »[6].

$$\Delta_{\tau,\rho} > 0, \text{ если } x_{\tau,\rho} = 0; \Delta_{\tau,\rho} = 0, \text{ если } x_{\tau,\rho} > 0, (\tau,\rho) \in U_N = U \backslash U_0 \text{ »[6].} \quad (2.6)$$

Теорема 6 устанавливает критерий оптимальности опорного потока, формулируя необходимые и достаточные условия через потенциалы. К примеру,

на практике потенциалы можно рассматривать как показатели эффективности использования дуг.

2.4 Алгоритмы структурных преобразований опорного потока

«Теорема 7. Дуга $(i_1, j_1 \in U_0)$, выводимая из корневого дерева G_0, определяется следующим образом: $(i_1, j_1) = (pred[j_0], j_0)$. Дуговые потоки дуг пути

$$i_1, pred[i_1], pred\big[pred[i_1]\big], \dots, s$$

уменьшаются на величину $y = x_{pred[j_0],j_0}$ дугового потока дуги $(pred[j_0], j_0)$ корневого дерева G_0. Новый дуговой поток дуги (i_1, j_1) равен нулю: $\bar{x}_{i_1,j_1} = 0$ и $pred[j_0] = i_0$. Дуговые потоки дуг пути i_0, $pred[i_0]$, $pred\big[pred[i_0]\big], \dots, s$ увеличиваются на значение y»[6].

Рассмотрим процедуру модификации опорного потока $\{x, G_0\}$. В ходе структурных изменений заменим $(pred[j_0], j_0)$ на (i_0, j_0). При это скорректируем потоковые значения на соответствующих дугах. Образовавшийся фундаментальный цикл $L(i_0, j_0)$ становится носителем предельного допустимого потока. Для поддержания древовидной структуры проведем обновление вспомогательных структур данных.

В первую очередь проведем поиск специальной вершины, которая удовлетворяет условию $thread = j_L$. Проводить поиск будем, обходя поддерево с корнем в j_L. Найденные же вершины станут основой для последующих преобразований, таких как: модификация списка предшественников, корректировка порядка обхода и пересчет глубин вершин с учетом поправочного коэффициента $r = depth[i_0] - depth[j_0] + 1$.

Особенностью данного алгоритма является сохранение первоначального корня на всех итерациях процесса. По итогу формируется новое допустимое решение, удовлетворяющее всем ограничениям исходной задачи.

2.5 Пути максимальной надежности

Для каждой дуги в графе G определим некоторую вероятность того, что это ребро останется в сети. Предполагается, что ребра могут выпадать из графа независимо друг от друга, тогда надежность пути из узла s в узел t можно описать как произведение вероятностей выпадения отдельных ребер.

$$R(W) = \prod_{(i,j)\in W} p_{i,j}. \quad (2.7)$$

Чем больше $R(W)$, тем выше вероятность того, что весь путь останется дееспособным.

Поскольку произведение вероятностей может быть неудобным для оптимизации, прологарифмируем $R(W)$ по основанию, большему единицы.

$$\log R(W) = \sum_{(i,j)\in W} \log p_{i,j}. \quad (2.8)$$

Чтобы перейти к стандартной задаче поиска пути, введем веса ребер

$$c_{i,j} = -logp_{i,j} \geq 0. \quad (2.9)$$

Теперь задача максимизации $R(W)$ эквивалентна минимизации суммы весов. Таким образом путь максимальной надежности, можно найти алгоритмом нахождения пути минимальной стоимости.

ГЛАВА 3
ГЕНЕРАТОР ГРАФОВ

3.1 Алгоритмы генерации графов

Существует несколько основных методов генерации графов. В данном разделе подробнее рассмотрим некоторые из наиболее популярных.

Самым простым методом является случайная генерация. В данном случае задается некоторое количество вершин, к которым случайным образом добавляются рёбра между парами вершин. При этом вероятность образования определенного ребра зависит от общей вероятности. Описанный метод в основном используется для изучения типичных свойств графов, используя теорию вероятности. На практике же таким образом можно создавать и исследовать модели сетевых и социальных сетей.

При генерации можно задать ряд правил, следую которым в итоге получается структурированный граф. Обычно за правила принимают степень связанности рёбер в графе. В отличие от случайных, структурированные графы обладают предсказуемой структурой и используются для моделирования более сложные и реальные системы.

Следующим методом является определение графа, путем задания $n-$ мерного пространства с добавление вершин, как точек определенного пространства. Ребрами же определяется расстояние между вершинами. Используются данные графы в основном для моделирования, где важно учитывать именно пространственные взаимоотношения, например, в сетях беспроводных сенсоров.

3.2 Реализация структурированного метода генерации графа

Для реализации структурированного метода был выбран язык программирования Python. Выбор был сделан в связи с его мощными возможностями и большим количеством библиотек, связанных с графами. В ходе написания приложения были использованы networkx, matplotlib, random, time. Так же, не менее важным аспектом, является наличие возможности организации визуализации.

Перейдем к описанию процесса генерации. Сперва пользователь задает количество вершин и ребер у ожидаемого графа рисунок 3.1.

```
num_nodes = 7  # Количество вершин
num_edges = 12  # Количество рёбер
```

Рисунок 3.1 — Параметры генератора

Так же в коде имеется проверка на возможность генерации графа рисунок 3.2, так как граф является ориентированным, то между двумя любыми вершинами может существовать одно единственное ребро.

```
if self.num_edges > self.num_nodes * (self.num_nodes - 1):
    raise ValueError("Слишком много рёбер для заданного количества узлов.")
```

Рисунок 3.2 — Проверка входных данных

Сам алгоритм заключается в случайной генерации пары чисел, меньших количества вершин, и задании между ними ребра, для которого в последующем будут определены ширина и стоимость. Направление задается порядком вершин. Так же заведем коллекцию для хранения уже существующих ребер. После генерации ребра, проверяем существует ли такое в графе и, если его нет, то добавляем в множество.

```
while edges_added < self.num_edges:
    u, v = random.sample(self.nodes, k: 2)
    if (u, v) not in existing_edges and (v, u) not in existing_edges:
        width = random.randint(*self.width_range)
        cost = random.randint(*self.cost_range)
        self.G.add_edge(u, v, cost=cost, width=width)
        existing_edges.add((u, v))
        edges_added += 1
```

Рисунок 3.3 — Генерация ребра для графа

В итоге получаем быстрый генератор направленных графов, которым воспользуемся для анализа алгоритмов.

```
Время выполнения: 0.000124 секунд
```

Русинок 3.4 — Время выполнения для графа $I = 7, U = 12$

3.3 Примеры генерации графов

Зададим начальные данные $I = 8, U = 15$. И сгенерируем для них граф рисунок 3.5.

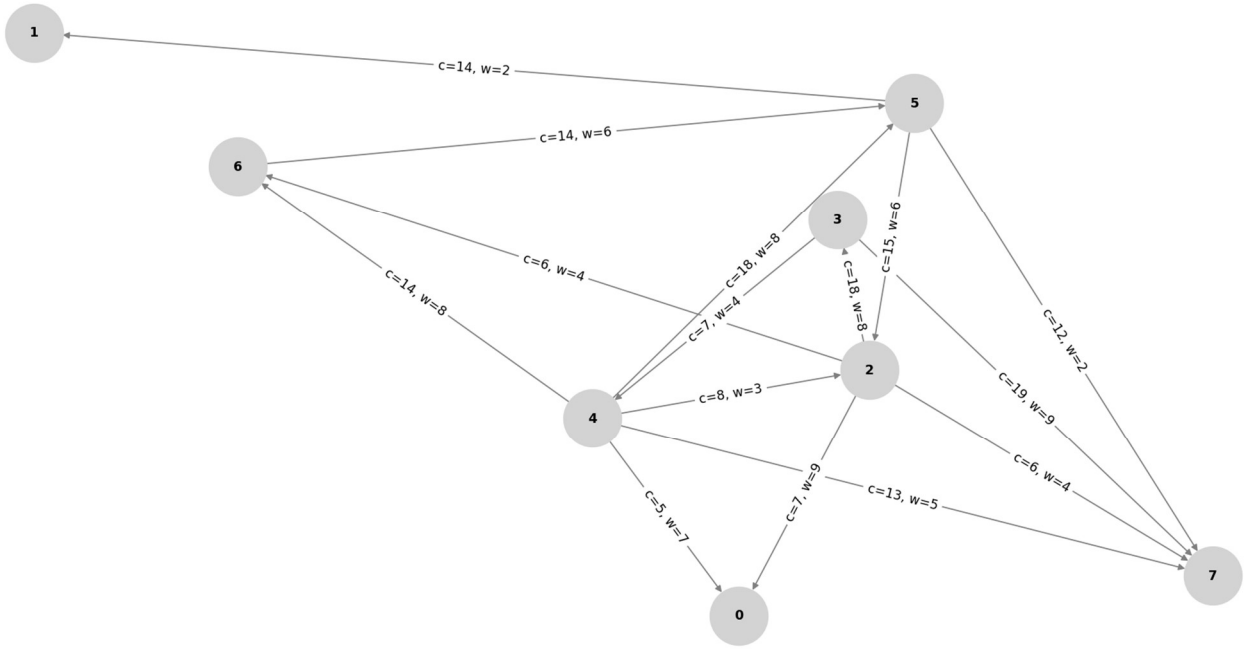

Рисунок 3.5 — Сгенерированный граф

```
Время выполнения: 0.001088 секунд
```

Рисунок 3.6 — Время генерации графа

Сгенерированные графы сохраняются в текстовый файл. Для нашего графа текстовый файл представлен на рисунке 3.7.

```
8 15
4 2 8 3
4 0 5 7
4 6 14 8
4 5 18 8
4 7 13 5
2 6 6 4
2 0 7 9
2 3 18 8
2 7 6 4
6 5 14 6
3 7 19 9
3 4 7 4
5 7 12 2
5 2 15 6
5 1 14 2
```

Рисунок 3.7 — Представление графа в файле

Теперь поменяем данные на $I = 1000$, $U = 5000$.

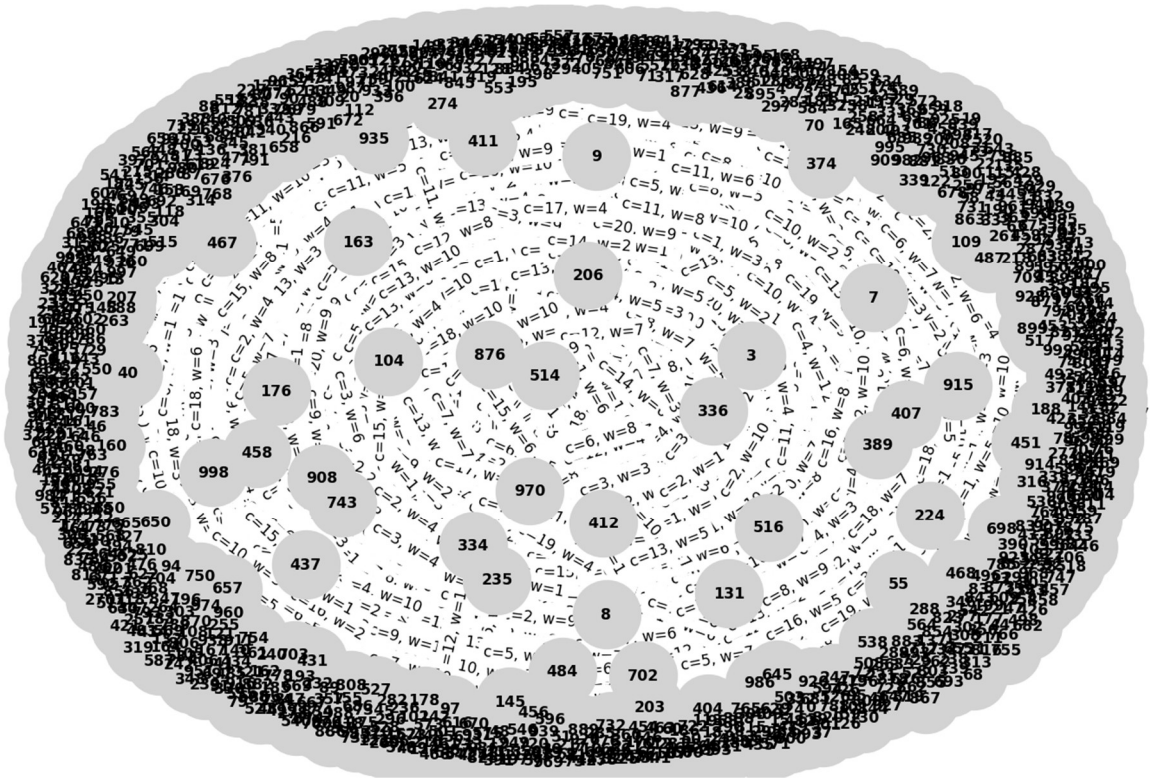

Рисунок 3.8 — Граф у которого $I = 1000$, $U = 5000$

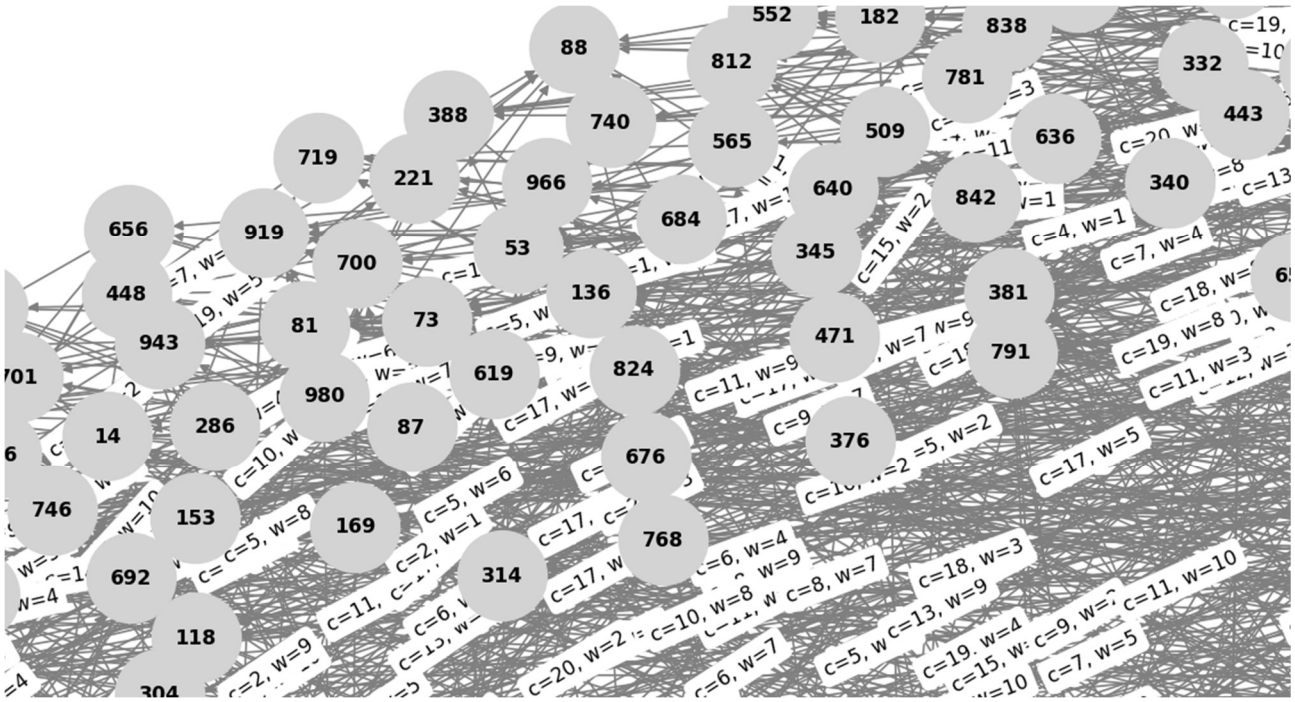

Рисунок 3.9 — Фрагмент графа

Для графа такого время выполнения программы заняло 0.019758 секунд, без учета временных затрат на визуализацию.

В дальнейшем с помощью данного генератора будем формировать графы и практически докажем эффективность и работоспособность описанных алгоритмов. При необходимости, код можно модифицировать под реализацию других типов графов.

ГЛАВА 4
ПРИМЕРЫ ПОСТРОЕНИЯ КРАТЧАЙШИХ ПУТЕЙ

4.1 Пример 1

Практический пример демонстрирует пошаговую реализацию алгоритма на конкретном графе с заранее заданными параметрами – узлами и ребрами.

Перейдем к реализации задачи. Задача имеет вид:

$$\sum_{(i,j)\in U} c_{i,j} x_{i,j} \to min,$$

$$\sum_{j\in I_i^+(U)} x_{i,j} - \sum_{j\in I_i^-(U)} x_{j,i} = \begin{cases} 1, \ i = s, \\ -1, \ i = t \\ 0, i \in I\backslash\{s,t\}, \end{cases},$$

$$x_{i,j} \in \{0,1\}, \{i,j\} \in U.$$

Пусть имеется граф с узлами s, a, b, c, t, e, f и ребра с различной стоимостью и шириной. На рисунке 4.1 представлена визуализация графа.

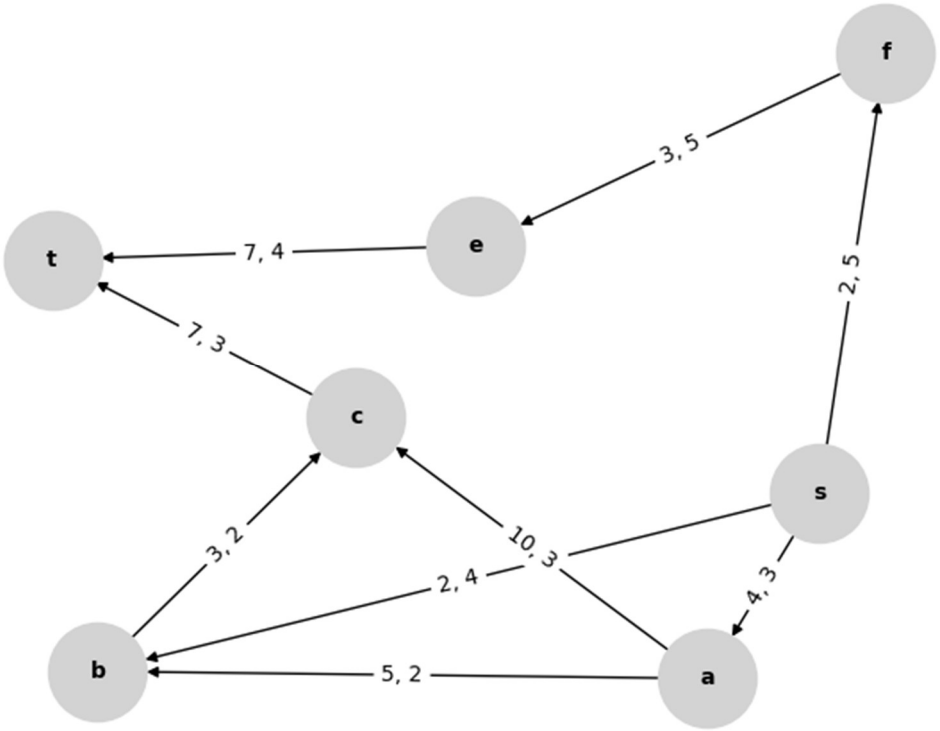

Рисунок 4.1 — Визуализация графа

Переходим к решению. Найдем кратчайший путь максимальной ширины из вершины s в t.

Алгоритм начинает работу с поиска максимальной ширины. Ширина $d_{s,f} = 5$. Но при такой ширине, не существует пути из s в t.

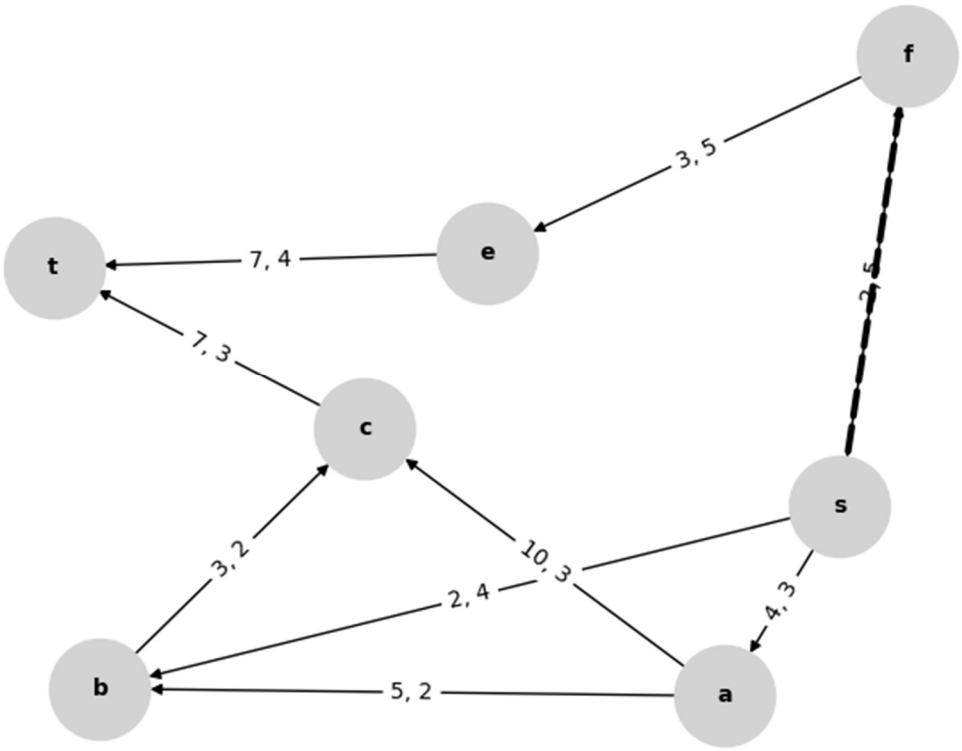

Рисунок 4.2 – Итерация 1

Сначала будет построен исходный граф, после чего алгоритм найдет путь из исходного узла s в целевой узел t. В ходе работы алгоритма:

1. Для разных значений минимальной ширины ребер будет проверяться достижимость узла t из узла s.

2. Будет найдено максимальное значение ширины p, при котором существует путь от s до t.

3. С использованием алгоритма Дейкстры(индексный) будет определен кратчайший путь при заданном ограничении на ширину ребер.

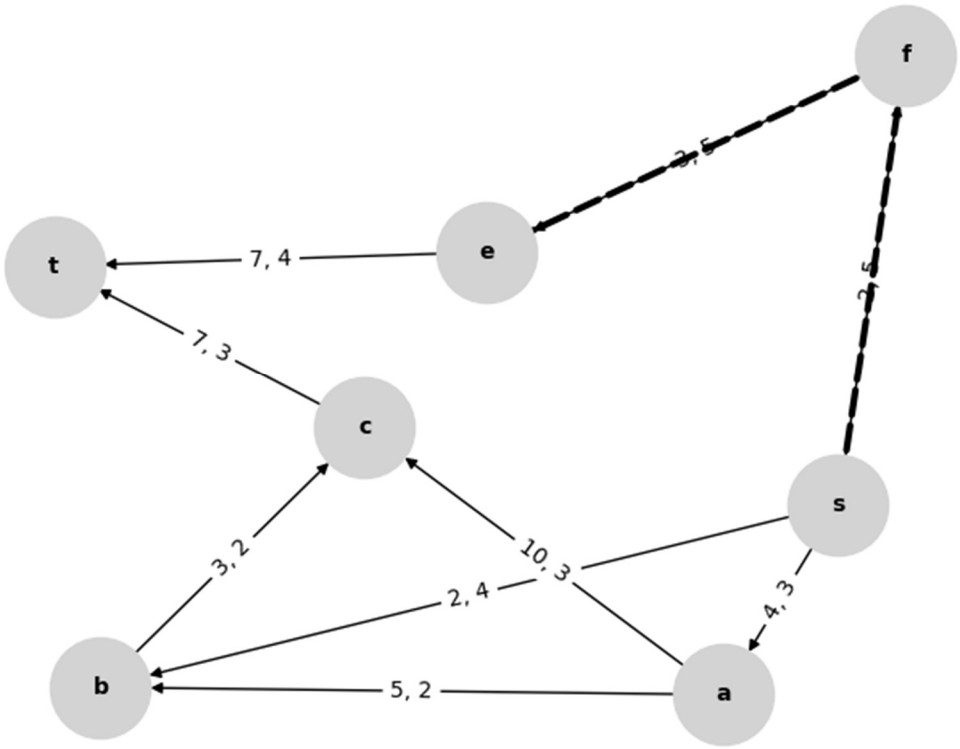

Рисунок 4.3 – Итерация 2

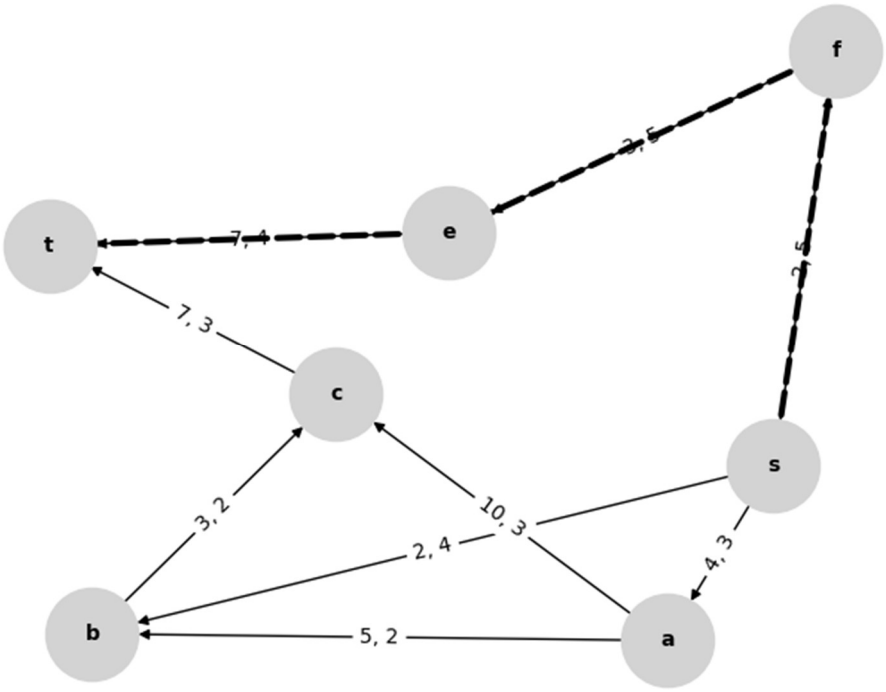

Рисунок 4.4 – Итерация 3 кратчайший путь из s в t

На рисунке 4.4 построен путь из s в t. Кратчайший путь: $\{s, f, e, t\}$. Стоимость пути – 12, ширина 4.

На основе примера наглядно показано, как работает алгоритм и каким образом удается учесть оба критерия (максимизация ширины и минимизация стоимости).

4.2 Пример 2

Рассмотрим ориентированный граф представленный на рисунке 4.5. Построим путь максимальной ширины из вершины $s = 3$ в $t = 2$.

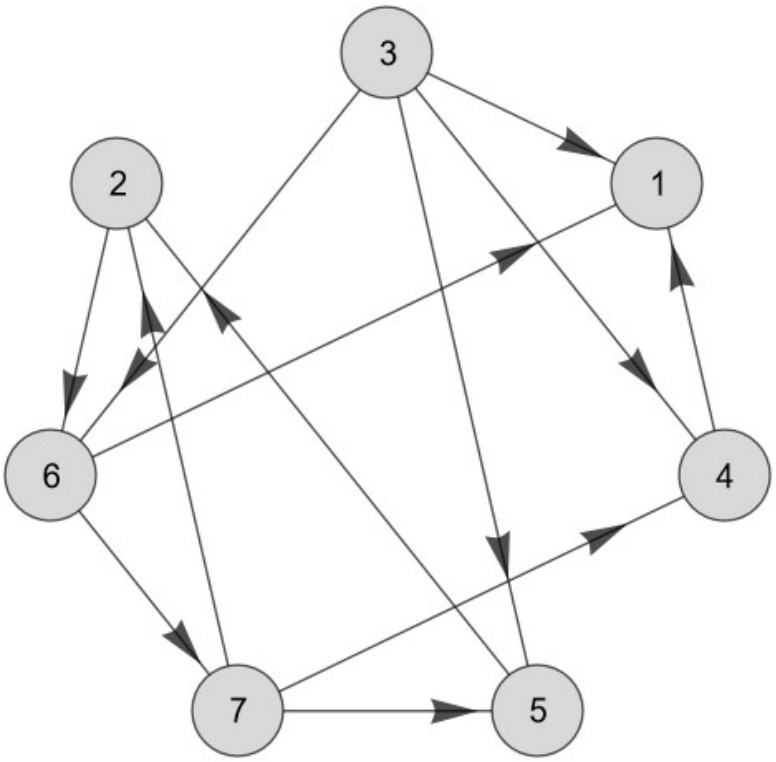

Рисунок 4.5 – Визуализация графа

Добавим к ребрам стоимости $c_{i,j}$ и веса $d_{i,j}$ (рисунок 4.6).

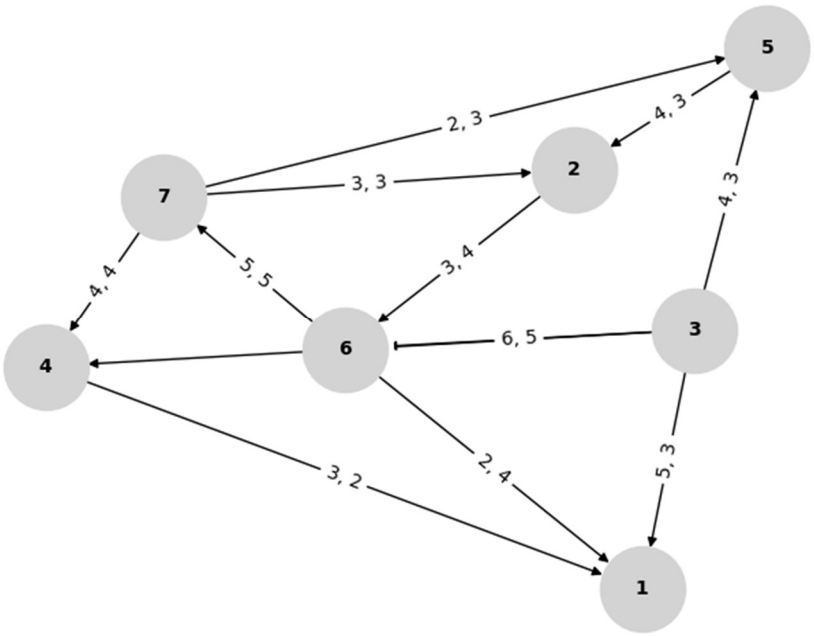

Рисунок 4.6 – Орграф со стоимостями и весами, указанными над дугами

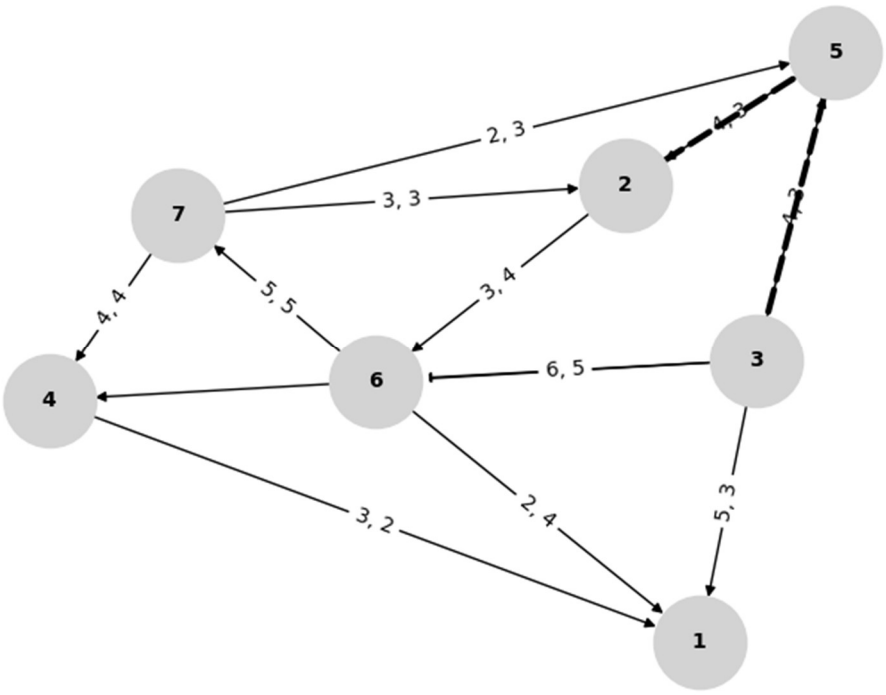

Рисунок 4.7 – Путь минимальной стоимости $\sum c_{i,j} = 8$, $(i,j) \in U$ при максимальной ширине $d_{3,2} = 3$

На графе можно увидеть, что из узла 3 существует несколько исходящих ребер, и алгоритм будет находить оптимальный путь с максимальной шириной и минимальной стоимостью, используя заданные критерии. На рисунке 4.7 выделен путь минимальной стоимости, максимальной ширины.

4.3 Пример 3

Сгенерируем граф с помощью написанного генератора. Покажем, как алгоритм работает изнутри на примере графа, представленного на рисунке 4.8. Путь будем искать путь из вершины 2 в 1.

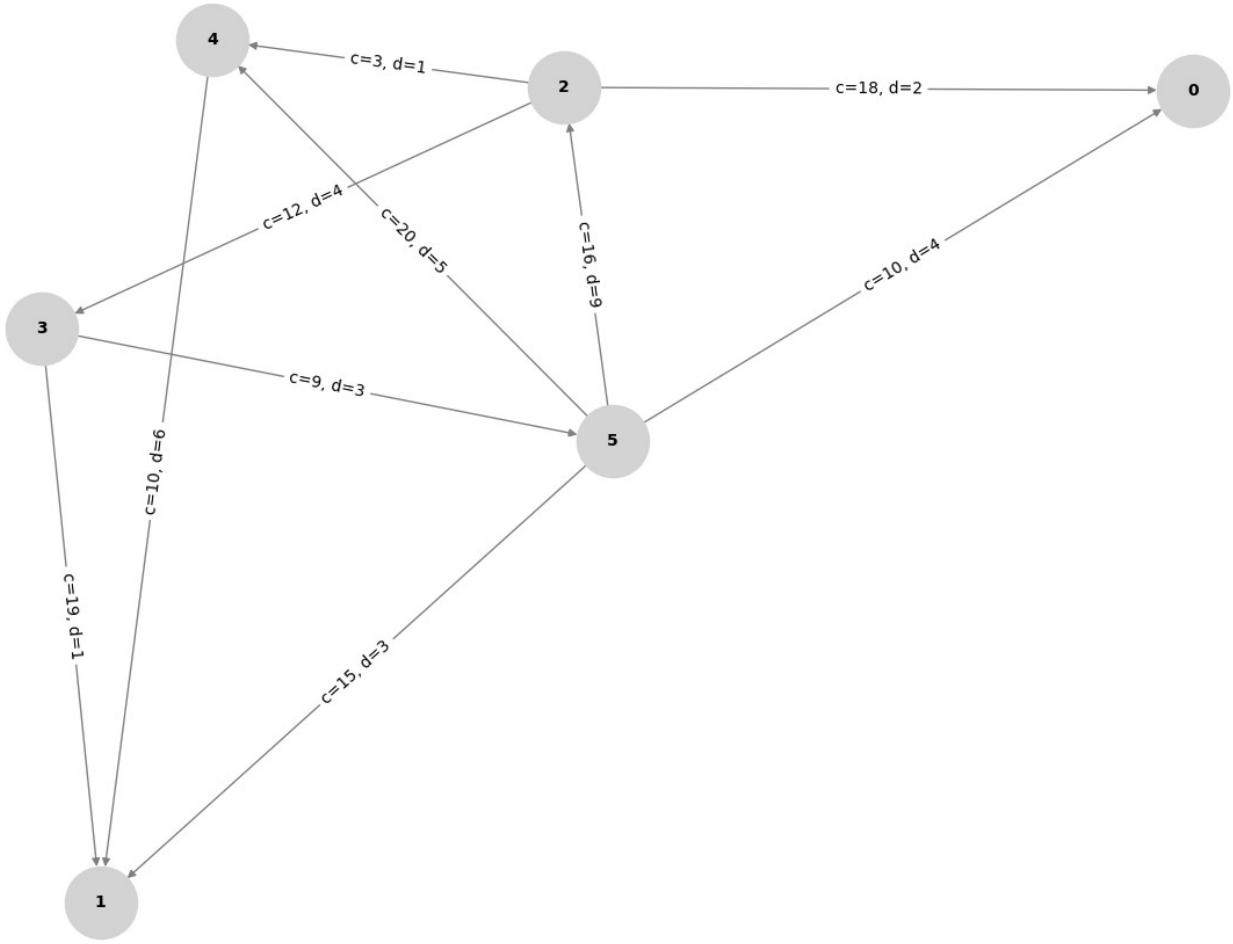

Рисунок 4.8 – Орграф со стоимостями и пропускными способностями дуг

Составим таблицу для данного графа предоставленную на рисунке 4.9.

(i,j)	Ширина	Стоимость
(5,2)	9	16
(4,1)	6	10
(5,4)	5	20
(2,3)	4	12
(5,0)	4	10
(5,1)	3	15
(3,5)	3	9
(2,0)	2	18
(2,4)	1	3
(3,1)	1	19

Рисунок 4.9 – Таблица рёбер для графа. Дуги упорядочены по убыванию пропускной способности

В полученной таблице занесены значения ширины и стоимости ребер данного графа. Дальнейшим шагом будет составление множества ребер максимальной ширины. Визуализация на рисунке 4.10.

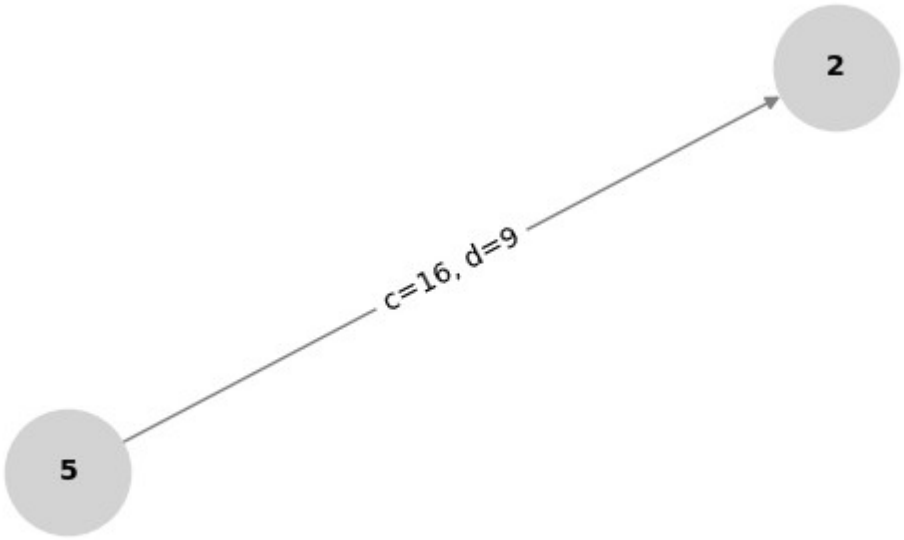

Рисунок 4.10 – Граф множества $d_{i,j} \geq 9$

Так как путь из вершины 2 в вершину 1 не существует, то продолжаем добавлять в множество ребра рисунок 3.11.

Рисунок 4.11 – Граф множества $d_{i,j} \geq 6$

Продолжаем итерации до тех пор, пока путь из вершины 2 в 1 не будет достигнут.

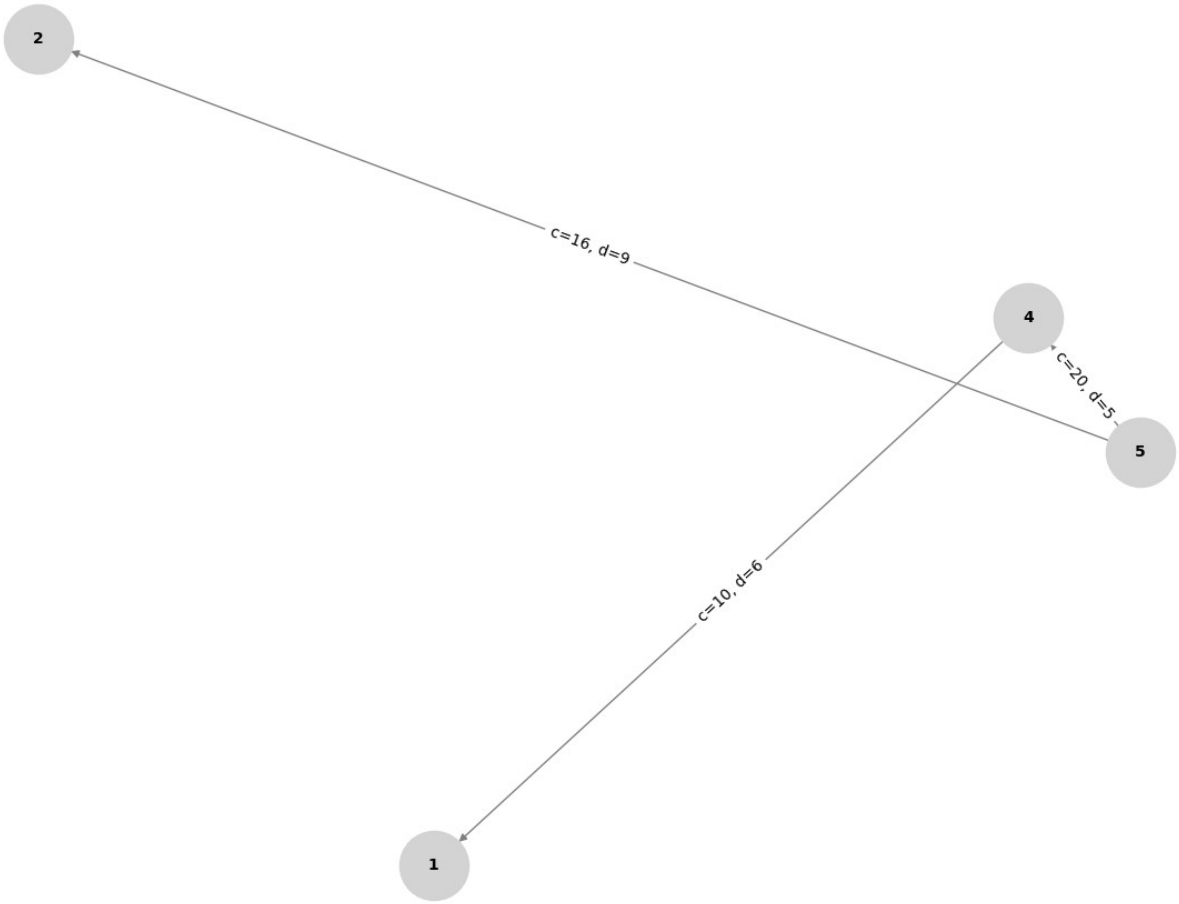

Рисунок 4.12 – Граф множества $d_{i,i} \geq 5$

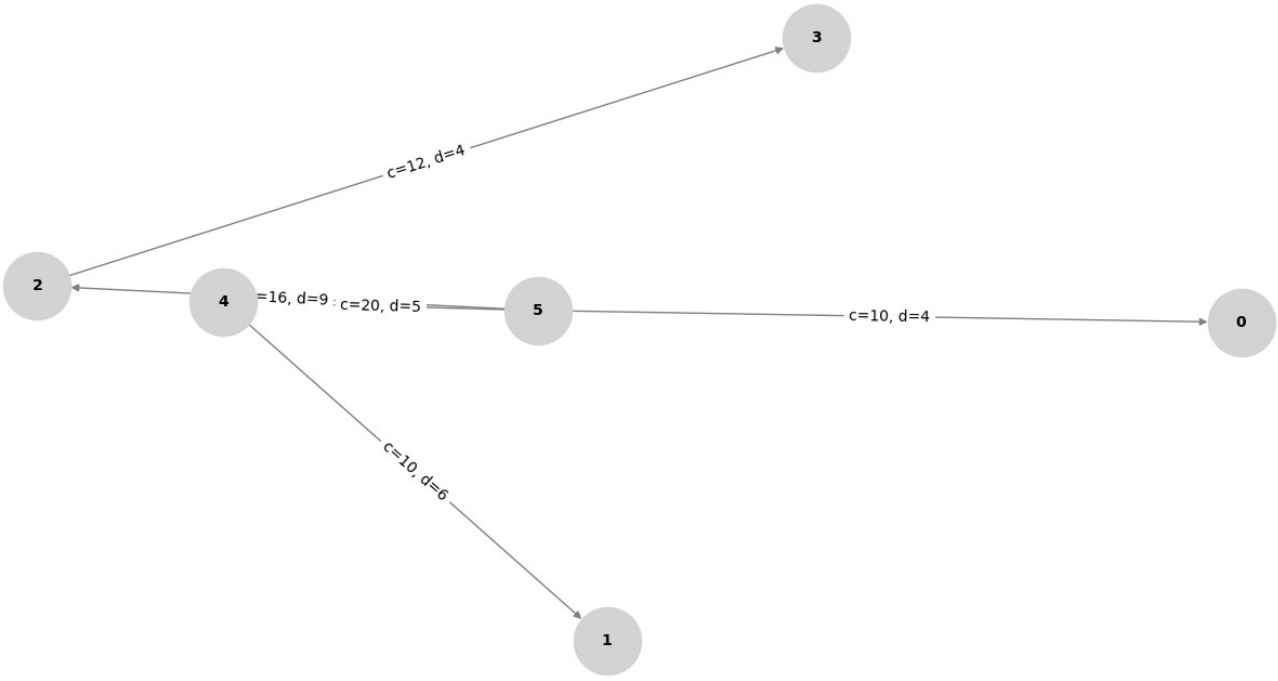

Рисунок 4.13 – Граф множества $d_{i,j} \geq 4$

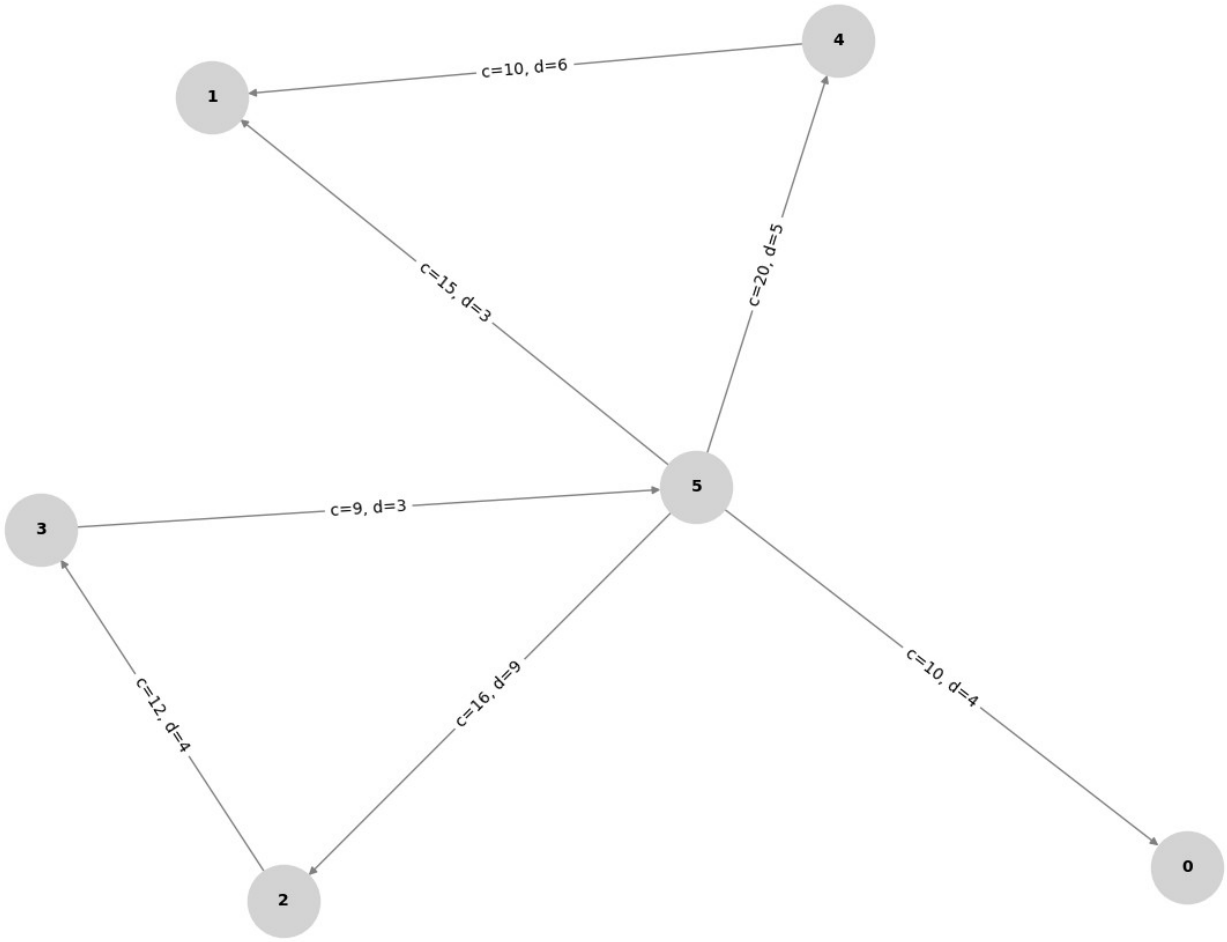

Рисунок 4.14 – Граф множества $d_{i,j} \geq 3$

В множестве $d_{i,j} \geq 3$ путь из вершины 2 в 1 наконец становится доступным. Применяем алгоритм Дейкстры и получаем итоговый маршрут на рисунке 3.15 максимальной ширины и минимальной стоимости.

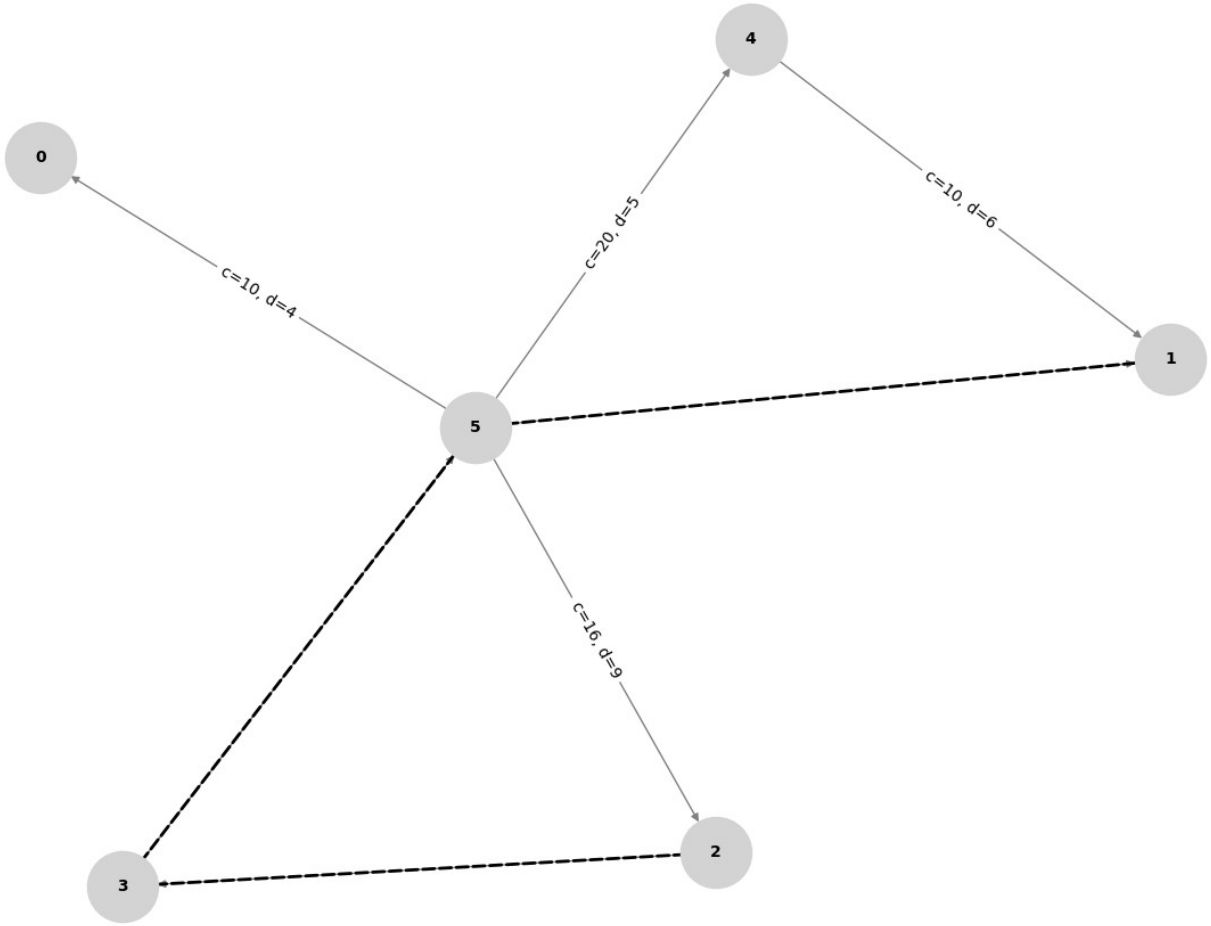

Рисунок 4.15 – Путь из вершины 2 в 1

```
Минимальная стоимость пути: 36
Путь: 2 -> 3 -> 5 -> 1
```

Рисунок 4.16 – Вывод программы

4.4 Пример 4

Проанализируем эффективность алгоритма для графа, в котором $I = 500$, $U = 1000$. Для этого сгенерируем граф рисунок 3.17.

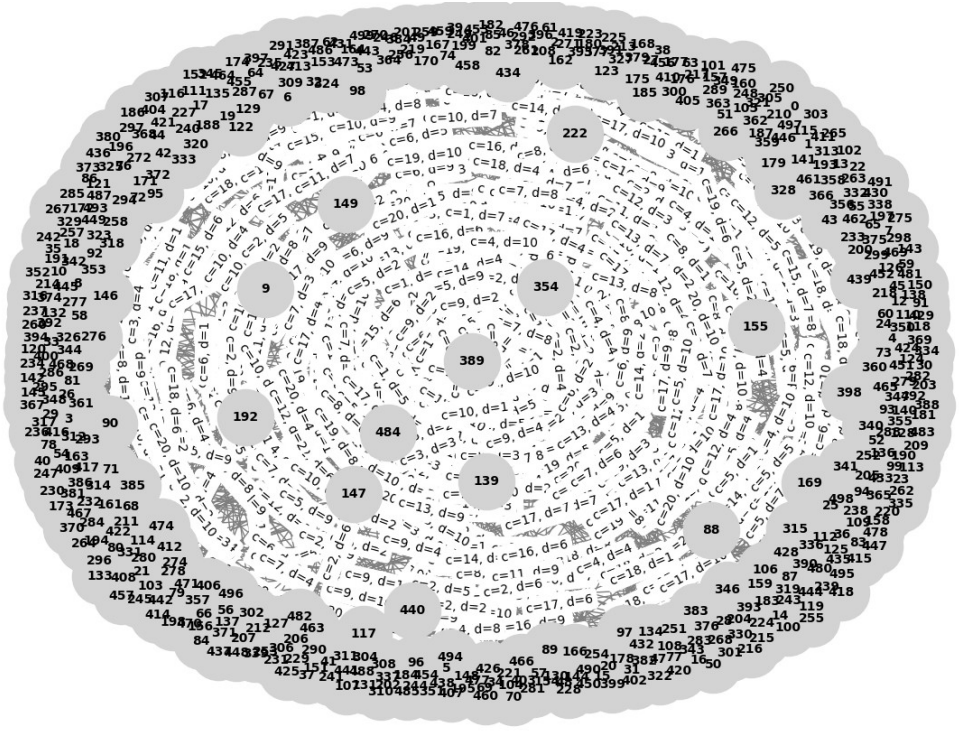

Рисунок 4.17 – Граф с $I = 500$ и $U = 1000$

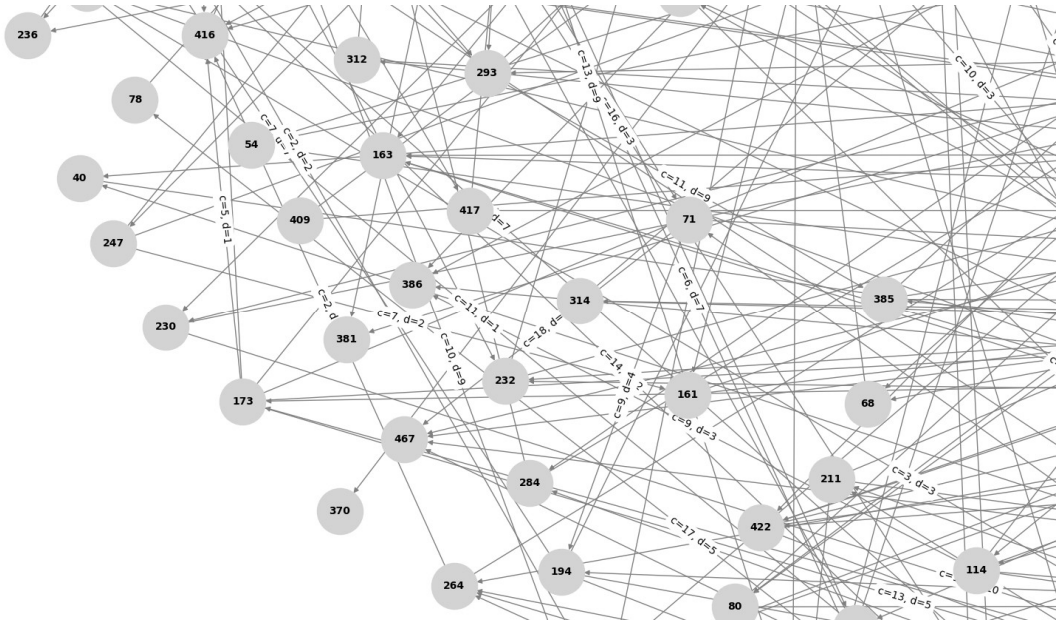

Рисунок 4.18 – Фрагмент графа

В итоге получаем.

```
Минимальная стоимость пути: 188
Путь: 2 -> 383 -> 67 -> 69 -> 459 -> 283 -> 418 -> 138 -> 450 -> 198 -> 457 -> 244 -> 173 -> 27 -> 263 -> 427 -> 147 -> 29 -> 119 -> 1
Время выполнения: 0.006726 секунд
```

Рисунок 4.19 – Выполнение программы для большого графа.

4.5 Построение пути с реальными длинами дуг

Перейдем к конкретной задаче. Для примера был выбран город Логойск. Необходимо создать граф, который будет максимально приближенным к этому городу. Для этого были использованы данные OpenStreetMap, в питоне они представлены библиотекой osmnx. Экспортируем данные и преобразуем их в граф.

```python
# Загрузка графа Логойска
place = "Логойск, Беларусь"
try:
    graph = ox.graph_from_place(place, network_type='drive', simplify=True)
except Exception as e:
    print(f"Ошибка загрузки графа: {e}")
    exit()
```

Рисунок 4.20 – Код для получения данных

Полученные данные содержат некоторые неточности, это связано с неравномерным покрытием, упрощенно геометрией и устаревшей информацией. Так же библиотека обладает собственным механизмом построения пути, который в дальнейшем будет использован для аналитического сравнения.

На рисунке 4.21 представлена карта Логойска с обработанной информацией.

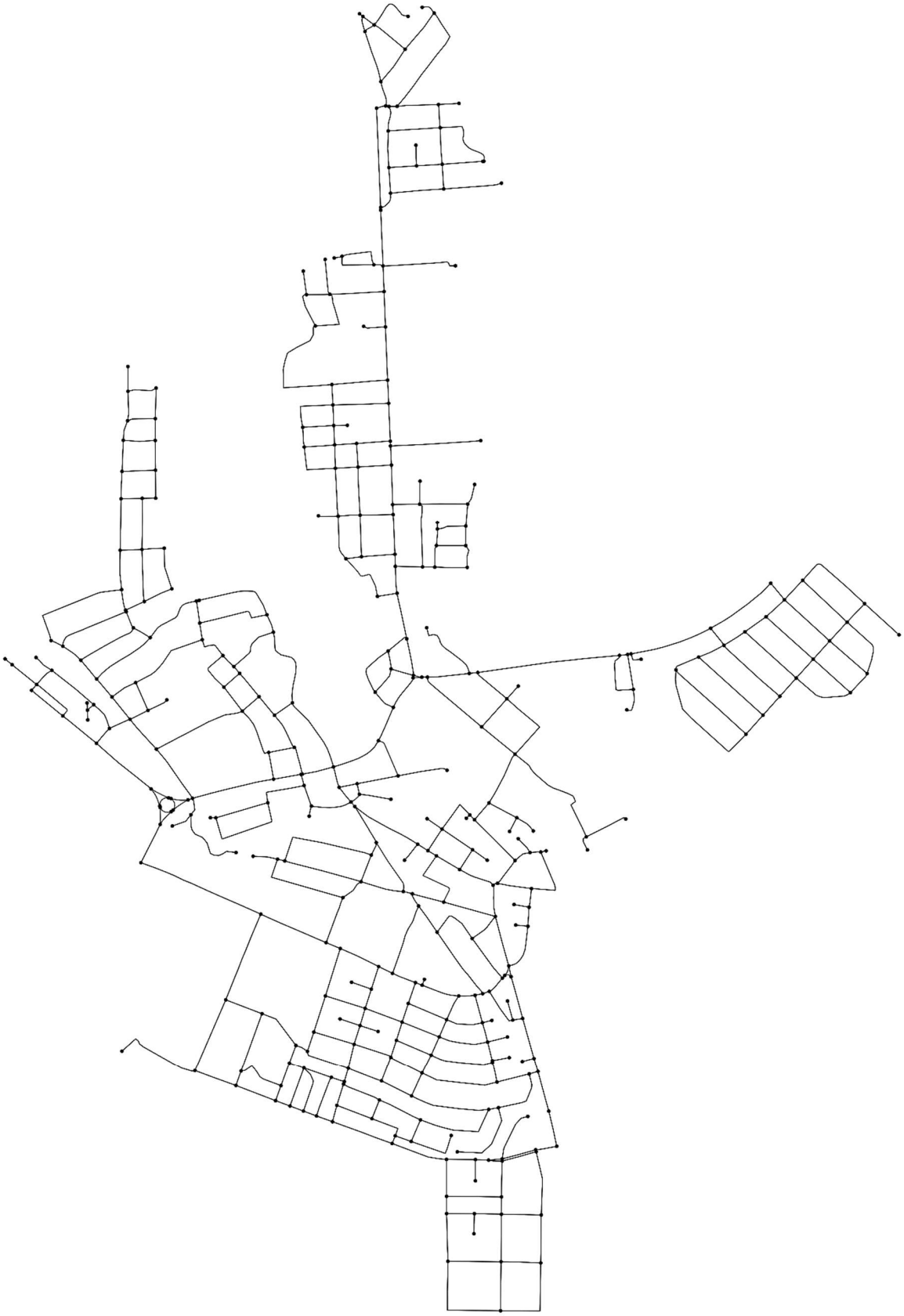

Рисунок 4.21 – Представление дорог Логойска в виде графа

На выходе мы получаем два текстовых файла с описание ребер и вершин, рисунки 4.22 и 4.23.

```
y,x,street_count,highway,geometry
54.219925,27.8485187,3,,POINT (27.8485187 54.219925)
54.219607,27.8505588,3,,POINT (27.8505588 54.219607)
54.19122,27.8594717,3,,POINT (27.8594717 54.19122)
54.1919339,27.8591353,1,,POINT (27.8591353 54.1919339)
54.1912768,27.8601097,3,,POINT (27.8601097 54.1912768)
54.1907685,27.8467134,3,,POINT (27.8467134 54.1907685)
54.1903099,27.849294,4,,POINT (27.849294 54.1903099)
54.1898127,27.8516484,4,,POINT (27.8516484 54.1898127)
54.1892115,27.8536525,4,,POINT (27.8536525 54.1892115)
54.1891729,27.8605243,3,,POINT (27.8605243 54.1891729)
54.1888703,27.8584598,3,,POINT (27.8584598 54.1888703)
```

Рисунок 4.22 – Представление вершин

```
osmid,highway,maxspeed,oneway,reversed,length,geometry,lanes,name,ref,bridge,width,junction
32680386,residential,60,False,False,179.18255622794706,"LINESTRING (27.8485187 54.219925, 27
32682635,residential,60,False,True,168.1456173063088,"LINESTRING (27.8485187 54.219925, 27.8
32680386,residential,60,False,True,33.54440485186007,"LINESTRING (27.8485187 54.219925, 27.8
32680386,residential,60,False,True,179.18255622794706,"LINESTRING (27.8505588 54.219607, 27.
```

Рисунок 4.23 – Представление рёбер

После успешной загрузки данных, преобразуем их для передачи в основную программу. Как видно из представленных файлов, библиотека osmnx, дает необходимую информацию для построения задачи. Важными будут являться такие данные, как oneway и reversed, отвечающие за направление ребер в строящемся графе. Значение length будет использована в ключе стоимости для ребра графа. Вместо ширины на практике можно было бы использовать загруженность дороги, но так как такие данные получить нет возможности, то просто сгенерируем их. Учитывая два эти фактора, была составлена двухкритериальная задача, которая постоянно встречается в повседневной жизни, например, при построении маршрута используя карты.

Далее преобразуем данные в подходящий для нас вид. И пропустим через программу поиска пути.

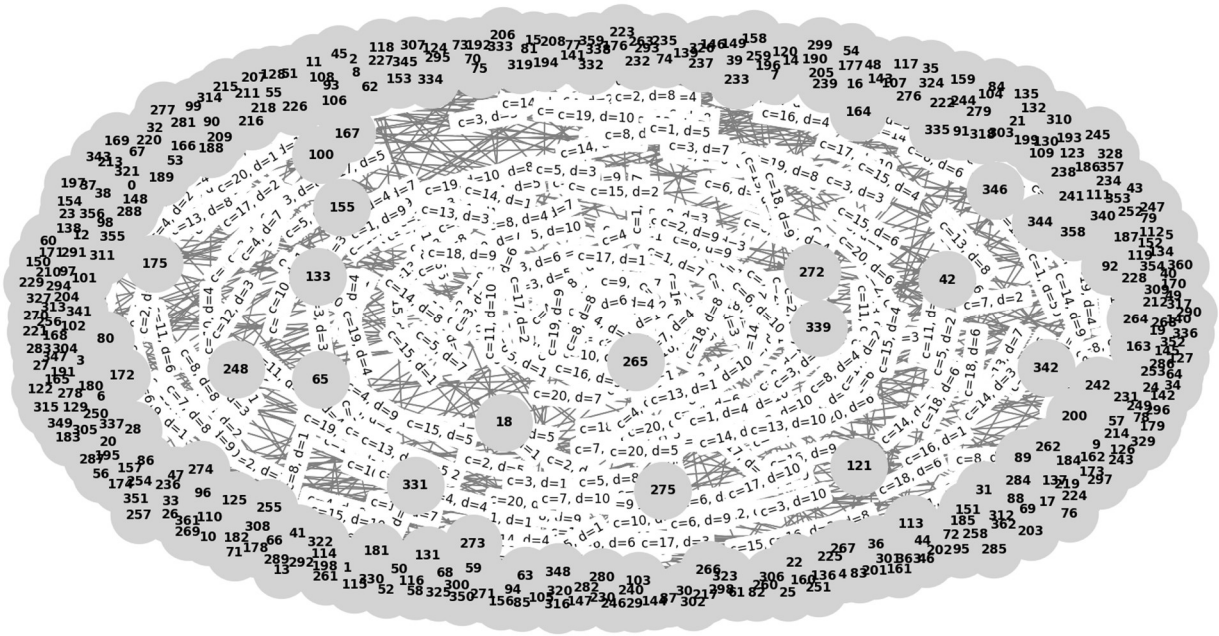

Рисунок 4.24 – Визуализация дорог

После работы программы был получен маршрут и рассчитана стоимость пути, для сравнения теперь воспользуемся встроенной функцией библиотеки.

```
Минимальная стоимость пути: 2073
Путь: 2 -> 4 -> 344 -> 242 -> 157 -> 151
Время выполнения: 0.007401
```

Рисунок 4.25 – Результат программы

Пути построенные обоими алгоритмами совпали. Но описанный базисный алгоритм вышел быстрее более чем в 100 раз.

```
Время выполнения: 1.265108 секунд
```

Рисунок 4.26 – Результат встроенной функции

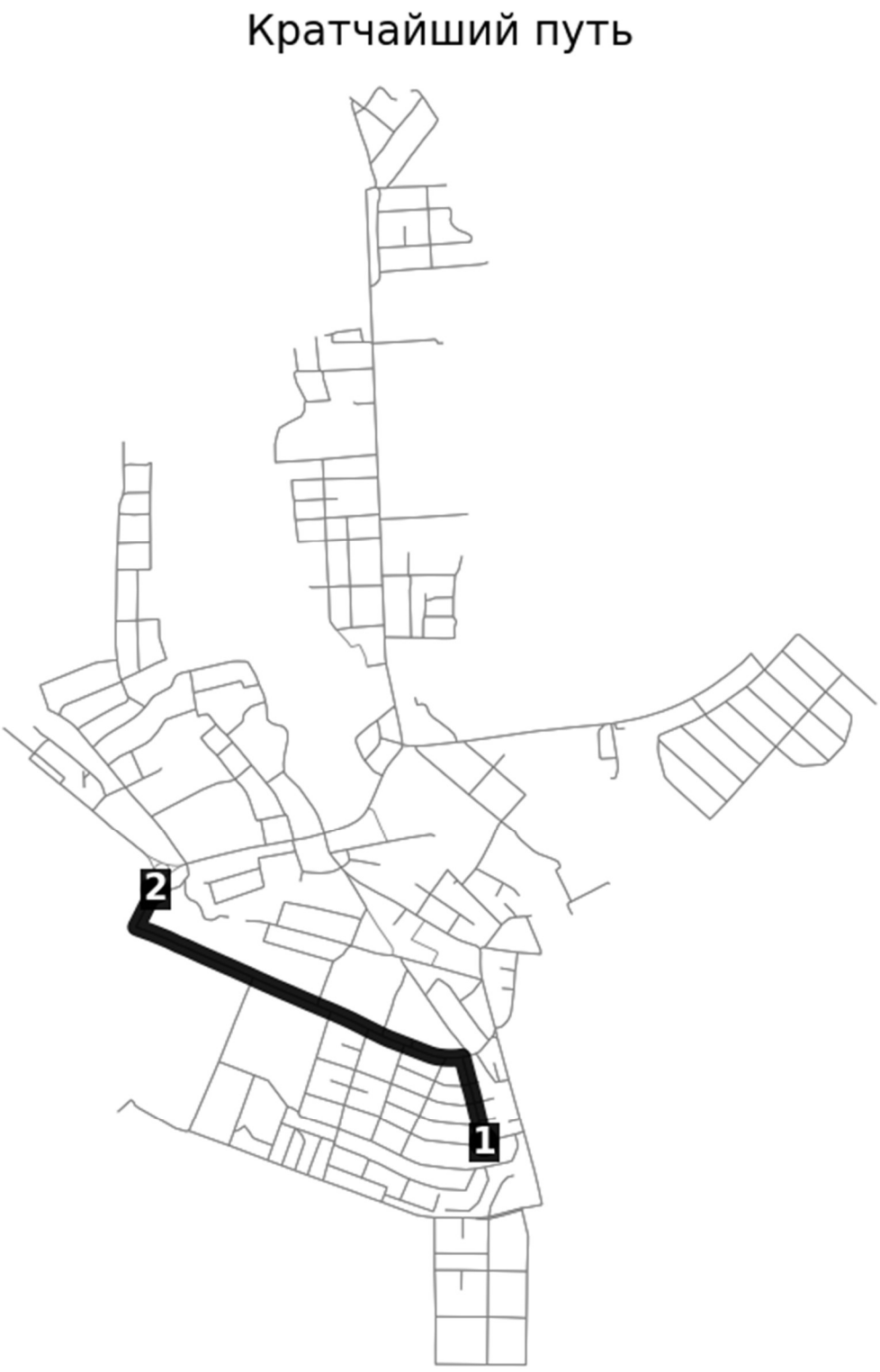

Рисунок 4.27 – Кратчайший путь из 2 в 1

4.6 Пример пути максимальной надежности

Перейдем к решению задачи о пути максимальной надежности. Как было доказана в теории, задачу о пути максимальной надежности можно решать так же, как и минимальной стоимости. Для примера воспользуемся графом, для которого уже был построен путь минимальной стоимости. Для правильного решения необходимо немного модифицировать входной файл, разделим все стоимости на 100. После небольших изменений в коде представленных на рисунке 4.29, получаем программу для нахождения путь максимальной надежности максимальной ширины.

```
for u, v, p, d in sorted_edges:
    if d >= width:
        current_graph.add_edge(u, v, weight=-math.log(p), reliability=p, width=d)
```

Рисунок 4.28 – Модификация программы

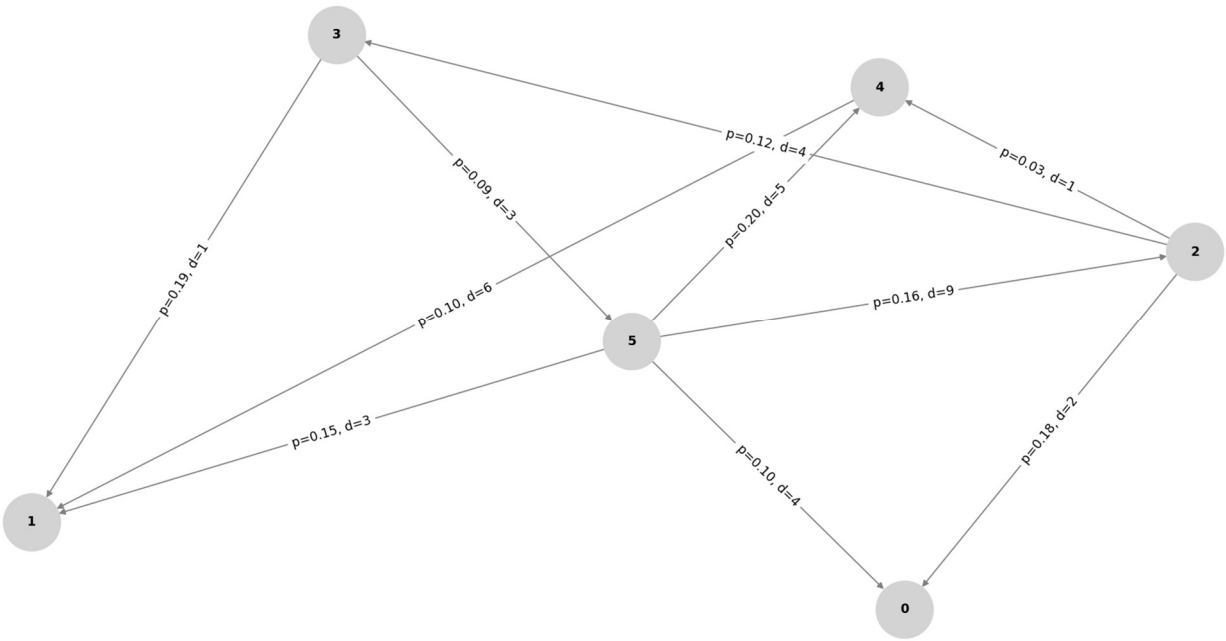

Рисунок 4.29 – Орграф с заданными вероятностями надежности дуг

Из нахождения пути максимальной надежности наглядно видно, что решение этой задачи совпадает с решением пути минимальной стоимости.

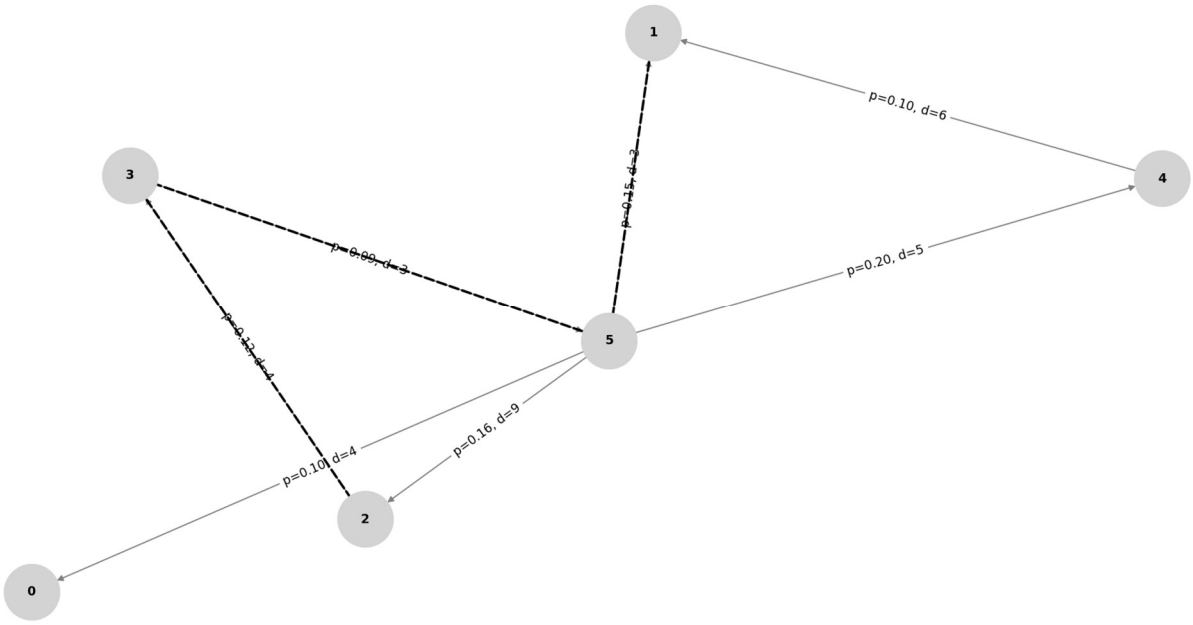

Рисунок 4.30 – Путь максимальной надежности

ЗАКЛЮЧЕНИЕ

В рамках данной дипломной работы были изучены и реализованы основные подходы к решению двухкритериальных задач потокового программирования на основе индексных и базисных методов. Проведенная работа позволила достичь следующих результатов:

Разработан и реализован алгоритм, который позволяет находить путь с максимальной шириной среди всех достижимых маршрутов в графе, одновременно минимизируя стоимость пути.

Было показано, что алгоритм эффективно решает задачу поиска оптимального маршрута, комбинируя два критерия:

1) Максимизация минимальной ширины рёбер на пути.
2) Минимизация суммарной стоимости пути.

Визуализация работы алгоритма на конкретных примерах позволила наглядно продемонстрировать, как происходит построение пути пошагово, что повышает понимание работы алгоритма и его корректности.

Была проведена модификация для решения задачи о пути максимальной надежности. В ходе которой было показано, что задачу о пути минимальной стоимости можно свести к задаче о пути максимальной надежности.

Так же было проведено тестирование, в ходе которого была наглядно показана эффективность описанного алгоритма. Была построена и решена реальная двухкритериальная задача, на примере построения кратчайшего пути в Логойске. При чем время, затраченное на построение пути, было более чем в 100 раз меньше чем при использовании встроенной функции.

Общая сложность алгоритма: сортировка ребер : $O(N * \log N)$, построение подграфа : $O(N)$, алгоритм Дейкстры : $O\big((M + N)\log M\big)$, где $M -$ количество вершин, $N -$ количество ребер. Если количество уровней ширины обозначить как W, общая сложность алгоритма составит $O(N \log N + W * ((N + M)\log M))$.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Габасов, Р. Методы линейного программирования: в 3 ч. / Р. Габасов, Ф. М. Кириллова. – Минск: БГУ, 1980. – Ч. 3: Специальные задачи. – 368 с.

2. Йенсен, П. Потоковое программирование / П. Йенсен, Д. Барнес. – М.: Радио и связь, 1984. – 392 с.

3. Кокотов Ю. Б. Методы оптимизации в многокритериальных задачах. Современные алгоритмы поиска компромиссных решений / Ю. Б. Кокотов. — СПб. : Лань, 2010. — 304 с.

4. Лотов А. В. Многокритериальные задачи принятия решений / А. В. Лотов, И. И. Поспелова. — М. : МАКС Пресс, 2008. — 192 с.

5. Пилипчук, Л.А. Двухкритериальные задачи потокового программирования / Л.А. Пилипчук, А.С. Пилипчук, Е.Н. Полячок // Известия Гомельского государственного университета имени Ф. Скорины.– 2020. – №6 (123). – Естественные науки. – С. 144–150.

6. Пилипчук Л.А. К двойственным методам решения двухпродуктовой транспортной сетевой задачи / Л.А Пилипчук // Вестн. Белорус. гос. ун-та. Серия 1. Физика. Математика. Інформатика. 1984. № 3. С. 52–54.

7. Пилипчук, Л.А. Линейные неоднородные задачи потокового программирования / Л.А. Пилипчук. – Минск : БГУ, 2009. – 222 с.

8. Пилипчук, Л.А. Оптимальные пути: алгоритмические, структурные и технологические решения/ Л.А. Пилипчук, А.С. Пилипчук, Е.Н. Полячок // Веснік Гродзенскага дзяржаўнага ўніверсітэта імя Янкі Купалы. Серыя 2. Матэматыка. Фізіка. Інфарматыка, вылічальная тэхніка і кіраванне. – 2020. –Том 10, № 3. –С. 143–151.

9. Пилипчук, Л.А. О численных методах решения двухкритериальных задач потокового программирования / Л.А. Пилипчук, Е.Н. Полячок // Материалы 10-го международного семинара «Аналитические методы анализа и дифференциальных уравнений – AMADE», Минск, 13 – 17 сентября 2021 г. – Институт математики НАН Беларуси. – Минск : ИВЦ Минфина, 2021. – С. 67 – 68.

10. Подкорытов В. А. Теория важности критериев в многокритериальных задачах принятия решений / В. А. Подкорытов, В. В. Подиновский. — М. : Физматлит, 2002. — 176 с.

11. Сборник тезисов 71-й научной конференции студентов Ин¬ститута бизнеса и менеджмента технологий Белорусского государ¬ственного университета : 15 мая 2014 г., Минск . - Минск, 2014. - С.195

12. Смирнов, А.Д. Алгоритмы решения двухкритериальных задач на графах / Иванова, Е.А. // Журнал вычислительной математики и математической физики. – 2015. – Т. 55, № 3. – С. 456–470.

13. Яндекс Карты: использование и возможности [Электронный ресурс] // Яндекс. – Режим доступа: https://yandex.ru/maps. – Дата доступа: 18.04.2025.

14. XX Международная научная конференция по дифференциальным уравнениям (ЕРУГИНСКИЕ ЧТЕНИЯ-2022): материалы Международной научной конференции. Новополоцк. 31 мая 03 июня 2022 г.: в 2 ч. — 4.2. — Новополоцк: Полоцкий государственный университет, 2022. — С.132-134.

15. Ahuja, R.K. Network Flows: Theory, Algorithms, and Applications / R.K. Ahuja, T.L.Magnanti, J.B. Orlin. – New Jersey: Prentice Hall, 1993. – 864 p.

16. Maros, I. Computational Techniques of the Simplex Method / I. Maros. Boston : Kluwer Academic Publishers, 2003. – 325 p.

ПРИЛОЖЕНИЕ А

ИНДЕКСНЫЙ АЛГОРИТМ НАХОЖДЕНИЯ КРАТЧАЙШЕГО ПУТИ

```
def read_graph(file_name):
    with open(file_name, 'r') as file:
        lines = file.readlines()

    nodes_count, edges_count = map(int, lines[0].split())
    edges = []
    for line in lines[1:]:
        u, v, cost, width = map(int, line.split())
        edges.append((u, v, cost, width))

    return nodes_count, edges


def build_and_print_table(edges):
    print("Таблица рёбер:")
    print(f"{'(i,j)':<15} {'Ширина':<15} {'Стоимость':<10}")
    print("-" * 40)
    edges.sort(key=lambda x: -x[3])
    for u, v, cost, width in edges:
        print(f"({u},{v}){' ' * 10}{width:<10}{cost:<10}")
    return edges


def visualize_graph(edges, path=None, title="Граф"):
    G = nx.DiGraph()
    for u, v, data in edges:
        G.add_edge(u, v, cost=data['cost'], width=data['width'])

    pos = nx.spring_layout(G, k=1.5, iterations=50)

    if path:
        path_edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]
        edge_labels = {
            (u, v): f"c={data['cost']}, d={data['width']}"
            for u, v, data in edges if (u, v) not in path_edges
        }
    else:
        edge_labels = {
            (u, v): f"c={data['cost']}, d={data['width']}"
            for u, v, data in edges
        }

    plt.figure(figsize=(12, 9))
    nx.draw(
        G,
        pos,
        with_labels=True,
```

```
        node_color='lightgray',
        node_size=2000,
        font_size=10,
        font_weight='bold',
        arrows=True,
        edge_color='gray',
        width=1.0
    )
    nx.draw_networkx_edge_labels(
        G,
        pos,
        edge_labels=edge_labels,
        font_size=10)

    if path:
        path_edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]
        nx.draw_networkx_edges(
            G,
            pos,
            edgelist=path_edges,
            edge_color='black',
            style='dashed',
            width=2.0
        )

    plt.title(title)
    plt.tight_layout()
    plt.show()


def dijkstra(graph, start, target):
    pq = [(0, start)]
    distances = {node: float('inf') for node in graph.nodes}
    distances[start] = 0
    visited = set()
    previous = {node: None for node in graph.nodes}

    while pq:
        current_cost, current_node = heapq.heappop(pq)

        if current_node == target:
            path = []
            node = target
            while node is not None:
                path.append(node)
                node = previous[node]
            path.reverse()
            return current_cost, path

        if current_node in visited:
            continue
        visited.add(current_node)

        for neighbor, data in graph[current_node].items():
            new_cost = current_cost + data['cost']
            if new_cost < distances[neighbor]:
```

```
                distances[neighbor] = new_cost
                previous[neighbor] = current_node
                heapq.heappush(pq, (new_cost, neighbor))

    return float('inf'), []


def main():
    file_name = "big.txt"
    nodes_count, edges = read_graph(file_name)
    visualize_graph([(u, v, {'cost': cost, 'width': width}) for u, v, cost,
width in edges], title="Исходный граф")
    sorted_edges = build_and_print_table(edges)

    start_node = 2
    target_node = 1

    subgraph = nx.DiGraph()
    max_width = sorted_edges[0][3]

    for i in range(max_width, 0, -1):
        for u, v, cost, width in sorted_edges:
            if width >= i:
                subgraph.add_edge(u, v, cost=cost, width=width)

        visualize_graph(subgraph.edges(data=True), title=f"Подграф с шириной >=
{i}")

        if start_node not in subgraph.nodes or target_node not in
subgraph.nodes:
            print(f"На уровне ширины >= {i}: Начальный или целевой узел
отсутствует в подграфе.")
            continue

        min_cost, path = dijkstra(subgraph, start_node, target_node)
        if min_cost != float('inf'):
            print(f"Минимальная стоимость пути: {min_cost}")
            print(f"Путь: {' -> '.join(map(str, path))}")

            visualize_graph(subgraph.edges(data=True), path=path, title=f"Граф
с выделенным путём (ширина >= {i})")
            break
    else:
        print("Путь не найден.")


if __name__ == "__main__":
    main()
```

ПРИЛОЖЕНИЕ Б

ГЕНЕРАТОР ОРИЕНТИРОВАННЫХ ГРАФОВ

```
class GraphGenerator:
    def __init__(self, num_nodes, num_edges, width_range=(1, 10),
cost_range=(1, 20)):
        self.num_nodes = num_nodes
        self.num_edges = num_edges
        self.width_range = width_range
        self.cost_range = cost_range
        self.nodes = list(range(num_nodes))
        self.G = nx.DiGraph()

    def generate_graph(self):
        if self.num_edges > self.num_nodes * (self.num_nodes - 1):
            raise ValueError("Слишком много рёбер для заданного количества
узлов.")

        edges_added = 0
        existing_edges = set()

        while edges_added < self.num_edges:
            u, v = random.sample(self.nodes, 2)
            if (u, v) not in existing_edges and (v, u) not in existing_edges:
                width = random.randint(*self.width_range)
                cost = random.randint(*self.cost_range)
                self.G.add_edge(u, v, cost=cost, width=width)
                existing_edges.add((u, v))
                edges_added += 1

    def visualize_graph(self):
        pos = nx.spring_layout(self.G, k=1.5, iterations=50)

        edge_labels = {
            (u, v): f'c={data["cost"]}, w={data["width"]}'
            for u, v, data in self.G.edges(data=True)
        }

        plt.figure(figsize=(12, 9))
        nx.draw(
            self.G,
            pos,
            with_labels=True,
            node_color='lightblue',
            node_size=2000,
```

```
            font_size=10,
            font_weight='bold',
            arrows=True,
            edge_color='gray',
            width=1.0
        )

        nx.draw_networkx_edge_labels(
            self.G,
            pos,
            edge_labels=edge_labels,
            font_color='black',
            label_pos=0.5
        )

        plt.title("Граф с шириной и стоимостью пути")
        plt.tight_layout()
        plt.show()

    def save_to_file(self, filename="big.txt"):
        with open(filename, "w") as file:
            file.write(f"{self.num_nodes} {self.num_edges}\n")

            for u, v, data in self.G.edges(data=True):
                file.write(f"{u} {v} {data['cost']} {data['width']}\n")

        print(f"Граф успешно сохранён в файл '{filename}'.")
```

ПРИЛОЖЕНИЕ В

ФОРМИРОВАНИЕ ТАБЛИЦЫ НА РЕАЛЬНЫХ ДАННЫХ

```
ox.settings.directed = True
ox.settings.log_console = True
ox.settings.use_cache = True
random.seed(13)

place = "Логойск, Беларусь"
try:
    graph = ox.graph_from_place(place, network_type='drive', simplify=True)
except Exception as e:
    print(f"Ошибка загрузки графа: {e}")
    exit()

fig, ax = plt.subplots(figsize=(15, 15), facecolor='white')

ox.plot_graph(
    graph,
    ax=ax,
    bgcolor='white',
    node_color='black',
    node_size=5,
    node_edgecolor='none',
    edge_color='black',
    edge_linewidth=0.5,
    show=False,
    close=False
)

ax.set_axis_off()

plt.savefig(
    "logoysk_black_white.png",
    dpi=600,
    bbox_inches='tight',
    pad_inches=0,
    facecolor='white'
)

print("Черно-белый граф сохранен в файл: logoysk_black_white.png")
```