МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)
01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)
Системное программирование и компьютерные технологии
(направленность (профиль)/специализация

БАКАЛАВРСКАЯ РАБОТА

на тему Решение оптимизационных задач методом динамического программирования

Студент С.Б. Ковяшев
(И.О. Фамилия) (личная подпись)
Руководитель О.В. Лелонд
(И.О. Фамилия) (личная подпись)
Консультанты М.А. Четаева
(И.О. Фамилия) (личная подпись)

Допустить к защите

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»_____20_____г.

Тольятти 2018

АННОТАЦИЯ

Тема выпускной квалификационной работы: «Решение оптимизационных задач методом динамического программирования»

Работа была выполнена студентом Тольяттинского государственного университета, института математики, физики и информационных технологий, группы ПМИб-1401, Ковяшевым Сергеем Борисовичем. Выпускная квалификационная работа посвящена созданию алгоритмов метода динамического программирования для решения оптимизационных задач, а также реализации одного из полученных алгоритмов.

Объект исследования – математические модели оптимизационных задач.

Предмет исследования – решение задач методом динамического программирования.

Целью выпускной квалификационной работы является использование метода динамического программирования для решения оптимизационных задач.

Задачи работы:

1. Изучение общих подходов динамического программирования.
2. Описание алгоритмов для решения оптимизационных задач.
3. Программная реализация одного из полученных алгоритмов.

Выпускная квалификационная работа состоит из введения, трёх глав, заключения, списка используемых источников. Во введении рассказывается об актуальности исследования по выбранному направлению, ставится проблема, цель и задачи исследования, определяются объект, предмет научных поисков, формулируется гипотеза, ставятся цель и задачи, указывается методологическая база исследования, его теоретическая, практическая значимость.

В главе 1 рассматриваются виды оптимизационных задач, демонстрируется применение к ним метода динамического программирования. Для каждой конкретной задачи выводятся рекуррентные

соотношения, которые определяют алгоритмы решения задач. В главе 2 приводится решение оптимизационных задач с конкретно заданными условиями. В главе 3 описывается программная реализация задачи отыскания кратчайших путей. В заключении представлены результаты и выводы о выполненной работе.

В работе использовано 6 таблиц, 3 рисунка, 2 блок-схемы и 1 приложение. Список литературы содержит 20 источников, в том числе 5 на иностранном языке. Общий объем выпускной квалификационной работы составляет 59 страниц.

ABSTRACT

The title of the bachelor's thesis is Solution of optimization problems with the dynamic programming method.

The aim of this work was the use of the method of dynamic programming for solving optimization problems.

The object of the study was mathematical models of optimization problems.

The subject of the study was optimizing the problems by dynamic programming.

The bachelor's thesis is devoted to the application of the basic approaches of the method of dynamic programming to mathematical models of optimization problems.

The first part of the thesis describes the types of optimization problems and applying the dynamic programming method to them. For each specific problem recurrence relations, which was determine the algorithms for solving problems, were derived. The second part provides solutions to optimization problems with specified conditions. The last part describes the software implementation of the task of finding the shortest paths.

Application of the dynamic programming method to optimization problems of small dimension allowed obtaining effective solution algorithms, in rare cases, inferior to alternative algorithms. However, with the increase in the dimension of the problem, the complexity of the solution and the amount of memory required to store the results of solving a large number of subproblems are greatly increased.

The problem of finding the shortest paths is a great example of applying dynamic programming method. On the basis of the obtained algorithm for solving this problem, a program was implemented in the Java programming language. The algorithm was supplemented by the definition of the sets of vertices through which the computed shortest paths passed.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

При управлении сложными дорогостоящими экономическими системами возникшие ошибки могут иметь крайне высокую цену. Чтобы исключить или, по крайней мере, снизить риск появления подобных ошибок, необходимо использовать математические модели. Математические модели могут выражать полный спектр существенных соотношений между параметрами систем управления и различными количественными характеристиками. Другими словами, математическая модель описывает в математической форме исследуемую систему, процессы или явления.

Оптимизационными называются экономико-математические задачи, цель которых заключается в нахождении оптимального (наилучшего) с точки зрения некоторого критерия или критериев варианта использования имеющихся ресурсов (труда, капитала и пр.).

Опирающиеся на хорошо построенные математические модели задачи управления приводят к качественным и пригодным для применения на практике результатам. Однако, такие задачи управления являются сложными, и простых формул для их решения не существует. Поэтому возникает объективная потребность в разработке специальных математических методов решения поставленных задач.

Таким методом решения задач управления многошаговыми процессами является рассматриваемый в данной работе метод динамического программирования.

Динамическое программирование — это один из математических методов нахождения оптимальных решений по управлению многошаговыми процессами. Состояние исследуемых систем изменяется поэтапно или во времени. В реальном мире подобные процессы и системы имеют широкое распространение. Важность и актуальность применения математических методов для решения соответствующих управленческих задач определяется необходимостью эффективного управления. Для метода динамического

программирования теоретической основой является принцип оптимальности, который получил широкую сферу приложений в технике, естествознании, экономике и военном деле.

Объект исследования: математические модели оптимизационных задач.

Предмет исследования: оптимизация задач методом динамического программирования.

Цель данной работы заключается в разработке алгоритмов решения оптимизационных задач методом динамического программирования и программной реализации одного из полученных алгоритмов.

Для достижения поставленной цели потребуется:

- Изучить виды оптимизационных задач.
- Проанализировать метод динамического программирования.
- Построить математические и оптимизационные модели для выбранных задач.
- Описать алгоритмы решения оптимизационных задач.
- Разработать программную реализацию одного из полученных алгоритмов для решения выбранной задачи.
- Протестировать программное обеспечение на реальных задачах.

ГЛАВА 1 ОСНОВНЫЕ ВИДЫ ОПТИМИЗАЦИОННЫХ ЗАДАЧ И МЕТОД ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ

1.1 Оптимизационные задачи

Оптимизационными называются экономико-математические задачи, цель которых состоит в нахождении с точки зрения некоторого критерия или критериев варианта использования имеющихся ресурсов (труда, капитала и пр.).

Оптимизационные задачи решаются с помощью оптимизационных моделей методами математического программирования.

Оптимизационная модель состоит из трех основных элементов: целевой функции, области допустимых решений и системы ограничений, определяющими эту область. В общем виде целевая функция включает в себя:

- управляемые переменные;
- неуправляемые переменные;
- форму функцию (вид зависимости между управляемыми и неуправляемыми переменными).

Область, в пределах которой осуществляется выбор решений называется областью допустимых решений. Эта область задается в виде системы уравнений и неравенств. В экономических задачах эти ограничения представляются наличными ресурсами, условиями.

Если система ограничений несовместима, то область допустимых решений является пустой. Ограничения подразделяются на:

а) линейные и нелинейные;

б) детерминированные и стохастические.

Несмотря на многообразие оптимизационных задач в различных областях человеческой деятельности к настоящему времени можно выделить некоторые основные типы:

- Задачи распределения ресурсов.

- Задачи ремонта и замены оборудования.
- Задачи составления оптимальных расписаний.
- Сетевые оптимизационные задачи.
- Задачи управления запасами.
- Задачи оптимизации систем обслуживания.
- Комбинированные задачи, объединяющие в себе черты задач разных типов.

Оптимизационные задачи решаются методами математического программирования, которые подразделяются на:

- исследование операций;
- нелинейное программирование;
- линейное программирование;
- динамическое программирование;
- выпуклое программирование;
- геометрическое программирование;
- целочисленное программирование и др.

1.2 Метод динамического программирования

Метод динамического программирования – это один из вычислительных методов математического программирования. Метод динамического программирования используется для решения специальных задач нелинейного программирования и оптимального управления, математические модели которых имеют характер многоэтапных и динамических процессов [3].

Основу метода динамического программирования составляет принцип оптимальности, который сформулировал Р. Беллман. «Каково бы ни было состояние системы перед очередным шагом, надо выбрать управление на этом шаге так, чтобы выигрыш на данном шаге плюс оптимальный выигрыш на всех последующих шагах был максимальным» [4]. Решение рекуррентных соотношений, которые получаются при использовании этого принципа,

позволяет последовательно получить оптимальное управление для исходной задачи оптимизации.

Суть метода динамического программирования заключается в том, что оптимальное управление для исходной задачи оптимизации конструируется постепенно, шаг за шагом. На каждом отдельном шаге выбирается оптимальное управление только для этого шага, но с учетом последствий. Управление, которое оптимизирует целевую функцию исключительно для данного шага, может привести к неоптимальному результату всего процесса. Поэтому управление оптимизируется с точки зрения всего процесса.

Принцип оптимальности непосредственно указывает процедуру нахождения оптимального решения и имеет конструктивный характер. В математической форме он имеет вид:

$W_i\ S = \min_{x_i} / \max_{x_i} \{f_i\ S, x_i + W_{i+1}\ S'\}$ – основное уравнение динамического программирования (уравнение Беллмана),

где S – текущее состояние управляемой системы;

$W_i = f_i\ S, x_i$ – функция выигрыша/стоимости при использовании x_i управления на *i*-м шаге;

$S' = \varphi_i(S, x_i)$ – следующее состояние, в которое переходит система под воздействием управления x_i.

Главными условиями применимости являются оптимальная подструктура решаемой задачи и аддитивность. Оптимальная подструктура задачи означает, что при разбиении исходной задачи на подзадачи решение подзадач может быть использовано при последующем конструировании решения исходной задачи. Подзадачи рекурсивно делятся на подзадачи меньшего размера до тех пор, пока их решение не станет тривиальным [1].

Как правило, решаемая задача содержит в себе большое количество перекрывающихся подзадач. Понятие перекрывающихся подзадач подразумевает использование решения малых подзадач в некотором количестве задач большего размера.

Основные этапы решения задачи:

1. Разбить задачу на подзадачи меньшего размера.

2. Составить рекурсивное решение для нахождения оптимального решения.

3. Вычислить значения, соответствующие оптимальному решению подзадач, методом восходящего/нисходящего анализа.

4. Использовать полученные решения подзадач для построения решения исходной задачи.

Динамическое программирование обычно придерживается двух подходов к решению задач:

- Нисходящее: задача рекурсивно разбивается на подзадачи меньшего размера до тех пор, пока их решение не будет тривиальным, и затем полученные решения комбинируются для решения исходной задачи.
- Восходящее: сначала просчитываются все подзадачи, решения которых потребуются для решения исходной задачи, затем на их основе строится решение исходной задачи.

Восходящее ДП лучше нисходящего в смысле размера необходимого стека и количества вызовов функций.

1.3 Анализ основных видов оптимизационных задач и применение метода динамического программирования

1.3.1 Задача коммивояжера

Прежде чем дать описание задачи коммивояжера, необходимо упомянуть несколько определений. Простым называется цикл в конечном ориентированном графе, который не имеет самопересечений. Гамильтоновым называется цикл, который проходит через все вершины графа, но только по одному разу. Полным называется граф G, в котором для каждой пары вершин i, j существует дуга (i, j). Если в графе G есть гамильтонов цикл, то такой граф именуется гамильтоновым.

Для взвешенного ориентированного графа G вес произвольного цикла – это сумма весов дуг, которые входят в состав этого цикла. Задачу коммивояжера сформулируем следующим образом: дан полный взвешенный ориентированный граф G без петель с множеством вершин N = {1, 2, …, n}; веса всех дуг графа неотрицательны; требуется найти гамильтонов цикл минимального веса.

В качестве исходной информации используется матрица $S=\{s_{ij}\}$ размерности n×n. В этой матрице s_{ij} – вес дуги (i, j) графа G, $i = \overline{1,n}$, j = $\overline{1,n}$, $i \neq j$; элементы находящиеся на главной диагонали матрицы S равны нулю.

Обычно вершины 1, 2, …, n графа G интерпретируются как города. Переходы между городами отображаются дугами графа. Находящемуся в городе 1 коммивояжеру требуется пройти ровно по одному разу каждый из остальных городов и вернуться обратно в город 1. Длины переходов между городами – это веса соответствующих дуг графа G. Цель задачи – найти удовлетворяющий описанным требованиям маршрут с минимальной длиной. Так как существуют иные возможные интерпретации задачи, на матрицу S не станем налагать требование симметричности и не обязательно выполнение неравенства треугольника.

Для получения алгоритма решения задачи применим метод динамического программирования. Будем придерживаться терминологии приведенной интерпретации задачи при описании алгоритма.

Пусть V – произвольное подмножество городов, в которое не входят город 1 и город i, где i∈N. Совокупность путей, берущих начало в городе i, завершающихся в городе 1 и проходящих по одному разу через промежуточные города из множества V, обозначим как M(i, V). Минимальной длины путь из совокупности M(i, V) обозначим через Z(i, V). Z(1, {2, 3,…,n}) – это минимальная длина замкнутого пути без самопересечений, который проходит через все города.

Совокупность M(i, V) будет содержать только один путь p = (i, j, 1), если V состоит только из одного города, V ={j}, где j ≠ 1 и j ≠ i. Поэтому

$$Z\ i,\ j\ \ = s_{ij} + s_{j1}, i \in N, j \in\ 2,3,\dots,n\ , j \neq i\ . \qquad (1)$$

Предположим, что для всех i∈ N \ {1} и для всех множеств V, состоящих из k элементов (k < n-1), значения Z(i, V) были вычислены. Тогда для V' – любого подмножества совокупности N \ {1, i}, состоящего из k+1 элемента, значение Z(i, V') будет вычисляться по формуле

$$Z\ i, V'\ = \min_{j \in V'}\ \ s_{ij} + Z\ j, V' \backslash \{j\}\ \ . \qquad (2)$$

Таким образом, для решения задачи коммивояжера будут использоваться функции (1)-(2), которые являются рекуррентными соотношениями динамического программирования.

1.3.2 Задача о ранце

По условию задачи имеются предметы P_1, P_2, …, P_n и ранец. У каждого предмета есть два показателя: стоимость c_i и вес v_i, где $i = 1, n$. Каждый из предметов можно положить в ранец только в одном экземпляре и только целиком. Цель задачи – отыскать такую совокупность предметов, при которой суммарная стоимость предметов будет максимальной, при

ограничении, что суммарный вес предметов не превысит определенной натуральной константы T.

Задача о ранце может быть решена разными методами:

- Перебор всех подмножеств набора P_1, P_2, …, P_n (сложность $O(2^N)$);
- Метод Meet-in-the-middle (сложность $O(2^{N/2}N)$);
- Метод динамического программирования (сложность — O(NT)).

Булеву задачу о ранце математически можно представить в виде:

$$\sum_{i=1}^{n} c_i x_i \to max \quad (3)$$

при условиях

$$\sum_{i=1}^{n} v_i x_i \le T \quad (4)$$

$$x_i \in \{0,1\}, i = 1,n \quad (5)$$

Задачу (3) - (5) назовем задачей Z.

Исходную задачу Z разобьем на совокупность частных задач Z(k, t), где $k \in \{1, 2, \ldots, n\}$; $t \in \{1, 2, \ldots, T\}$. Задачу Z(k, t) запишем следующим образом:

$$\sum_{i=1}^{k} c_i x_i \to max$$

при условиях

$$\sum_{i=1}^{k} v_i x_i \le t$$

$$x_i \in \{0,1\}, i = 1,k.$$

Для задачи Z(k, t) считается, что в наличии есть только первые t предметов из совокупности $\{P_1, P_2, \ldots, P_n\}$, а также суммарный вес ограничен константой t.

Обозначим оптимальное решение задачи Z(k, t) через B(k, t). Легко увидеть, что задача Z(n, T) – это исходная задача Z, значит её оптимальным решением будет B(n, T).

Оптимальное решение частной задачи Z(k, t) сводится к

$$B(1,t) = \begin{cases} 0 \text{ при } t \in \{0,1,2,\ldots,v_1 - 1\}; \\ c_1 \text{ при } t \in \{v_1, v_1 + 1, \ldots, T\}. \end{cases} \quad (6)$$

Допустим, что значения функции B(k, t) уже найдены для k∈{1, 2, …, n-1} и t∈{1, 2, …, *T*}. При вычислении B(k+1, t), если t∈{1, 2, …, v_{k+1}-1}, то ситуация идентична ситуации нахождения B(k, t), так как предмет P_{k+1} , появляющийся при переходе от задачи Z(k, t) к задаче Z(k+1, t) невозможно добавить в ранец. В ситуации, когда t∈{v_{k+1}, v_{k+1}+1, …, T}, при переходе от Z(k, t) к Z(k+1, t) появляется возможность добавить в ранец предмет P_{k+1}. Если не использовать эту возможность и не добавлять предмет P_{k+1}, то B(k+1, t) примет то же значение, что и B(k, t). В противном случае, когда предмет P_{k+1} (со стоимостью c_{k+1} и весом v_{k+1}) кладется в ранец, задача перейдет к задаче Z(k, t - v_{k+1}). Таким образом, получается:

$$B(k+1,t) = \begin{matrix} B\ k,t \ \text{если}\ t \in \{0,1,2,\dots,v_{k+1}-1\}; \\ max\ B\ k,t\ , c_{k+1} + B\ k,t-v_{k+1} \quad \text{при}\ t \in\ v_{k+1}, v_{k+1}+1,\dots,T\ ; \end{matrix} \qquad (7)$$

где k = 1, 2, …, n-1.

Для решения задачи о ранце используются полученные рекуррентные соотношения (6) – (7) динамического программирования. Решение строится последовательным вычислением B(1, t) для t∈{1, 2, …, *T*}, затем B(2, t) для t∈{1, 2, …, *T*} и т.д., пока не будет найдено оптимальное решение B(n, T) для исходной задачи Z.

Процесс нахождения значений B(k, t) можно реализовать в виде процесса последовательного заполнения таблицы стоимостей. Таблица заполняется по строкам, которым соответствуют возрастающие значения параметра k. Столбцам таблицы соответствуют возрастающие значения параметра t. Каждая клетка таблицы заполняется значением B(k, t) для соответствующих этой клетке строки k и столбца t. В клетки первой строки вносятся значения, полученные с помощью формулы (6). Формула (7) используется для заполнения остальных n-1 строк. В каждой клетке целесообразно также фиксировать множество M(k, t) добавленных в ранец предметов, при которых достигается оптимальное решение B(k, t). При t< v_1 M(1, t) = ∅, а при t≥ v_1 M(1, p) ={1}. Для остальных n-1 строк

$$M(k+1,t) = \begin{matrix} M\ k,t\ , \text{если}\ \boldsymbol{B}^*\ k+1,t\ = \boldsymbol{B}^*\ k,t\ , \\ M\ k+1,t\ = M\ k,t-v_{k+1}\ \cup\ \mathrm{k}+1\ \ \textbf{в противном случае}. \end{matrix}$$

Оптимальное решение исходной задачи Z будет находиться в клетке (n,T). Этому решению соответствует оптимальное значение критерия B(k, T) и множество добавленных в ранец предметов M(n, T).

Описанный алгоритм является алгоритмом решения задачи о ранце методом динамического программирования и имеет сложность O(NT), что показывает эффективность данного решения перед альтернативными алгоритмами.

1.3.3 Задача отыскания кратчайших путей на графе

Задача нахождения кратчайших путей задается конечным взвешенным ориентированным графом G с множеством вершин N = {1, 2, …, n}. Веса дуг трактуются как их длины, и эти значения неотрицательны. Путь из вершины i_1 в вершину i_k определяется последовательностью i_1, i_2, …, i_k, если в данном графе имеются дуги (i_t, i_t+1), где t =1, 2, …, k-1. Сумма весов дуг, образующих путь, является длинной этого пути. Цель задачи – найти пути минимальной длины из вершины 1 до каждой другой вершины графа G.

Из-за существования разных постановок задачи для решения могут использоваться разные алгоритмы:

- алгоритм Дейкстры;
- алгоритм Левита;
- алгоритм Ли (волновой алгоритм) и др.

Перечисленные алгоритмы имеют усредненную сложность $O(n^2)$.

Теперь применим метод динамического программирования и сравним сложность полученного алгоритма с вышеперечисленными.

Введем обозначение для веса дуги (i, j) графа G – c(i, j). Назовем длину кратчайшего пути из вершины 1 до вершины *x* расстоянием от 1 до *x*. Это расстояние обозначим как s(*x*).

Очевидно, что путь s(1) будет равен 0. Пусть множество H – это множество вершин, расстояния до которых уже вычислены. В начальном состоянии это множество состоит только из элемента {1}. Обозначим

минимальную из длин кратчайших путей от 1 до вершин множества $N \setminus H \subseteq \{2, 3, \ldots, n\}$ через $s(1, N \setminus H)$. Если в состав множества H уже входит вершина 1, то получаем

$$s\ 1, N\setminus H\ = \min_{i\in H\ , j\in N\setminus H}\ (s(i)\ +\ c(i,j)). \qquad (8)$$

Допустим, что минимум из правой части соотношения получается при значениях $i = a_0$ и $j = b_0$. Значит, кратчайший путь из вершины 1 до вершины b_0 определяется суммой расстояния от вершины 1 до вершины a_0 и дугой (a_0, b_0). Длина $s(b_0)$ этого пути равна $s(a_0) + c(a_0, b_0)$.

Уравнение (8) является рекуррентным соотношением динамического программирования для решения задачи отыскания кратчайших путей. Используя эту формулу, на первом шаге определяется ближайшая к вершине 1 вершина a_1 из совокупности $\{2, 3, \ldots, n\}$. На втором шаге определяется ближайшая к вершине 1 вершина a_2 из совокупности $\{2, 3, \ldots, n\}\setminus\{a_1\}$. На третьем шаге определяется ближайшая к вершине 1 вершина a_3 совокупности $\{2, 3, \ldots, n\}\setminus\{a_1, a_2\}$, и т.д. В ходе счета строится дерево кратчайших путей D из вершины 1 до остальных вершин. Вершина 1 является корнем дерева кратчайших путей D. Когда на произвольном шаге алгоритма минимум из правой части соотношения (8) получается на паре (a_0, b_0), ребро (a_0, b_0) добавляется к конструируемому дереву.

Оформление хода выполняемых вычислений может быть организовано как процесс заполнения таблицы. i-ая строка таблицы соответствуют i-ой вершине графа G, где $i = 1, n$. Заполнение таблицы происходит последовательно слева направо по столбцам и каждый столбец заполняется сверху вниз. Первый столбец получает название «1». На пересечении столбца «1» и j-ой строки в клетку вносится значение $c(1, j)$ при условии, что в графе G дуга (1, j) есть. В противном случае в клетку вносится $+\infty$. После заполнения всех строк этого столбца в нем отмечается символом * наименьшее значение (если таких значений несколько, выбирается любое из них). Если значение u, отмеченное символом *, располагается на пересечении

со строкой v, то расстояние от вершины 1 до вершины v считается найденным и равным s(v) = u. На этом первый шаг алгоритма завершается.

На каждом последующем шаге работы алгоритма заполняются два следующих столбца таблицы. На каждом шаге определяется кратчайший путь от вершины 1 до еще одной вершины. Когда определяется очередное значение s(x), строка x, соответствующая этому расстоянию, считается помеченной и остальные клетки этой строки не заполняются. В незаполненных клетках ставится символ «-». Так как значение s(1) изначально известно и равно 0, первая строка считается помеченной с самого начала.

На втором и на каждом последующем шаге четные столбцы получат название, соответствующее найденной на предыдущем шаге вершине x графа G, для которой алгоритм определил кратчайший путь. Для первого, второго и следующих четных столбцов в заголовках помимо «x» также указывается значение s(x). При заполнении второго и каждого следующего четного столбца "x" в каждую клетку, находящуюся в пересечении с неотмеченной строкой j, вносится сумма s(x) + c(x, j); если в графе G дуга (x, j) отсутствует, в клетку вносится символ +∞. Третий и далее каждый следующий нечетный столбец имеет наименование "min". В каждую клетку такого столбца, стоящую в пересечении с j-ой неотмеченной строкой, вносится минимальное из расстояний, записанных в двух соседних слева клетках данной строки. После заполнения каждого очередного столбца "min" в нем отмечается символом * наименьший элемент (если таких элементов несколько, отмечается любой из них). Если отмеченный символом * элемент последнего заполненного столбца "min" равен u и находится в строке номер v, то, согласно (8), считаем найденной длину кратчайшего пути от вершины 1 до вершины v, s(v)= u. С этого момента v-ая строка таблицы переходит в число отмеченных строк, дальнейшее заполнение ее клеток производиться не будет. Следующий подлежащий заполнению столбец четный, ему назначается имя "v", в заголовке этого столбца указывается также найденное значение s(v).

Далее заполняется очередной столбец "min". В результате заполнения каждой следующей пары столбцов определяется расстояние от вершины 1 до еще одной вершины графа, таким образом завершается реализация следующего шага алгоритма. Изначально известно, что s(1) = 0. Поэтому общее число заполняемых пар столбцов n-1. Заполнение каждого очередного столбца требует выполнения линейно зависящего от n числа элементарных операций.

Таким образом, реализация изложенного алгоритма требует выполнения квадратично зависящего от n числа элементарных операций. А значит, метод динамического программирования не уступает таким популярным алгоритмам, как алгоритм Дейкстры или волновому алгоритму.

1.3.4 Задача о назначениях

Задача о назначениях формулируется следующим образом. Дано множество исполнителей I = {1, 2, ..., n} и множество работ R = {r_1, r_2, ..., r_n}. Также имеется матрица n×n численных оценок (допустим, оценок производителей) A = {a_{ij}}, где a_{ij} – оценка закрепления исполнителя i за работой r_j, i=1,n, j =1,n. Назначения – это взаимно однозначные отображения множества {1, 2, ..., n} в себя. Назначение d предписывает исполнителю i работу $r_{d(i)}$. Тогда оценка этого назначения будет $a_{i\ d(i)}$, т.е. элемент из матрицы A, где i=1,2, ..., n. Каждое назначение d будет оцениваться по критерию $K\ d\ = \sum_{i=1}^{n} a_{i\ d(i)}$. В данной интерпретации задачи значение критерия $K\ d$ – это суммарная производительность исполнителей при назначении d. Цель задачи – найти такое назначение, при котором будет максимальной суммарная производительность исполнителей.

Классическая задача о назначениях имеет целевую функцию

$$\sum_{i=1}^{n} a_{id(i)}.$$

Обозначим записанную задачу символом Z и рассмотрим её решение методом динамического программирования. Для этого выделим

совокупность частных подзадач Z(i, W_i). Подзадачи формулируются аналогично исходной задаче Z: i∈{1,2, …, n}, W_i – это i-элементные подмножества совокупности {1,2, …,n}. В подзадаче Z(i, W_i) между исполнителями {1,2, …, i} распределяются работы, индексы которых перечисляются в W_i. Критерий у подзадач такой же, как у задачи Z – суммарная производительность распределенных исполнителей. Обозначим оптимальное значение критерия в подзадаче как Z*(i, W_i). Тогда Z*(n, {1,2, …, n}) станет оптимальным значением критерия для исходной задачи Z.

Очевидно, что

$$Z^*(1,\{j\}) = \mathbf{a_{1j}} \text{ для всех } j \in \{1,2,\dots,n\}. \quad (9)$$

Если i > 1, то

$$Z^*(i,W_i) = \max_{j\in W_i} (\mathbf{a}_{ij} + Z^*(i-1, W_i \setminus j)), i = 2,3,\dots,n; \quad (10)$$

где W_i – это произвольное i-элементное подмножество из совокупности {1,2, …, n}.

Записанные рекуррентные формулы (9) - (10) являются соотношениями динамического программирования для решения классической задачи о назначениях. Оценкой числа элементарных операций, требуемых для выполнения алгоритмом, основанным на рекуррентных формулах (9) - (10), является функция, экспоненциально зависящая от n.

В отличие от всех остальных рассматриваемых в данной главе задач, для классической задачи о назначениях существуют алгоритмы, которые обладают лучшим быстродействием в сравнении с рекуррентными соотношениями динамического программирования. Описанная задача может быть решена, в частности, алгоритмом Куна.

1.3.5 Задача о порядке произведения матриц

Даны прямоугольные матрицы M_1, M_2, …, M_n, таких размерностей, что их произведение будет равно:

$$M = M_1 x M_2 x \dots x M_n$$

Известно, что при умножении матрицы размерностью p×q на матрицу размерностью q×r потребуется выполнение f(p, q, r) элементарных операций. Функция f является монотонно возрастающей относительно всех своих аргументов. Цель задачи – определить такой порядок умножения матриц M_1, M_2, …, M_n, что умножение потребует минимального количества элементарных операций.

Возьмем в качестве примера матрицы M_1, M_2, M_3 и M_4, размерности которых 5×30, 30×60, 60×1 и 1×150 соответственно. При умножении матриц размерностей p×q и q×r потребуется 2pqr элементарных операций. Если выполнять умножение матриц в порядке $(M_1\times(M_2\times(M_3\times M_4)))$, то потребуется 603 тысячи элементарных операций. Однако, если порядок произведения будет задан записью $((M_1\times(M_2\times M_3))\times M_4)$, на вычисление потребуется только 5400 элементарных операций.

С целью нахождения экономной схемы может быть использован метод перебора, однако перебор всех вариантов порядка перемножения n матриц требует количества операций, экспоненциально зависящего от числа n. Если число перемножаемых матриц достаточно большое, то метод перебора на практике неприемлем.

Задачу нахождения оптимальной схемы умножения матриц M_1, M_2, …M_n, где каждая матриц M_i задана размерностью $r_i\times r_{i+1}$, для $i = 1, n$, назовем задачей B. Образуем совокупность частных подзадач B(i, j), где каждая подзадача заключается в минимизации числа элементарных операций в произведении $M_i\times M_{i+1}\times\ldots\times M_j$ при $i = 1, n$; $j = 1, n$; $i \le j$. Минимальное число операций, требующееся для вычисления $M_i\times M_{i+1}\times\ldots\times M_j$, обозначим через m_{ij}. Очевидно, что $m_{ii} = 0$.

Значения m_{ij} будем определять последовательно, по возрастанию разности j - i. Учитывая, что на перемножение матрицы размерностью p×q на матрицу размерностью q×r потребуется выполнение f(p, q, r) элементарных операций, получим:

$$m_{i\,i+1} = f(r_i, r_{i+1}, r_{i+2}), i = 1, 2, \ldots, n-1. \qquad (11)$$

Для j - i > 1 имеем:

$$m_{ij} = \min_{k\in\{i,i+1,\ldots,j-1\}} m_{ik} + m_{k+1\,j} + f(r_i, r_{k+1}, r_{j+1}) . \quad (12)$$

Рекуррентные соотношения (11) - (12) позволяют вычислять значения m_{ij} и определять соответствующие схемы умножения матриц последовательно, увеличивая разность j - i от 1 до n-1. В итоге будет вычислена величина m_{1n} и соответствующая ей оптимальная схема произведения матриц $M_1, M_2, \ldots M_n$.

Описанная процедура определения оптимальной схемы умножения n матриц имеет оценку, кубично зависящую от числа n выполняемых элементарных операций. Альтернативы решению методом динамического программирования, такие как перебор всех вариантов расстановки скобок, имеют экспоненциальную сложность и напрямую не применяются на практике.

1.3.6 Задача распределения ресурсов

Между n предприятиями P_1, P_2, ..., P_n необходимо распределить начальную сумму средств E_0. Выделенная предприятию сумма x_k принесет прибыль, равную $f_k(x_k)$ (k = l, 2, ..., n). Допустим, что выполняется несколько утверждений:

прибыль от вложения средств в предприятие P_1 не будет зависеть от вложения средств в остальные предприятия;

полученная от разных предприятий прибыль будет выражаться в одних и тех же единицах;

общая прибыль будет равна сумме доходов, которые получены при вложении средств по всем предприятиям.

Требуется определить количество средств, которое будет выделено каждому отдельному предприятию и которое даст максимальную суммарную прибыль.

$$Z = \sum_{k=1}^{n} f_k(x_k). \quad (13)$$

Переменные x_k обязаны удовлетворять двум условиям

$$x_1 + x_2 + \cdots + x_n = E_0 \; ; \; x_k \geq 0 \; (k = 1, \ldots, n). \qquad (14)$$

Цель задачи заключается в нахождении таких значений x_1, ..., x_n, удовлетворяющих ограничениям (14), при которых целевая функция (13) примет максимальное значение.

Построив математическую модель задачи можно перейти к описанию оптимизационной модели динамического программирования. Процесс распределения средств по n предприятиям позволяет рассматривать его как многошаговый процесс с количеством шагов, равным n. k-му шагу будет соответствовать номер предприятия, которому выделяются x_k средств. Значит, на первом шаге первому предприятию выделяем x_1 средств, на втором шаге – второму предприятию из оставшихся средств выделяется x_2 и т.д. Переменные x_k (k=1, 2, ..., n) будут рассматриваться как управляющие. Величина E_0 характеризует начальное состояние системы. Если выделить из этих средств величину x_1, то для распределения останется $E_1 = E_0 - x_1$ средств и т.д. Значения E_1, E_2, …, E_n являются остатками средств от распределения на предыдущих шагах. Они будут соответствовать параметрам состояния системы. Изменению состояния соответствует уравнение

$$E_k = E_{k-1} - x_k \quad (k = 1, \ldots, n).$$

Суммарная прибыль за n шагов распределения составляет

$$Z = F(E_0, U) = \sum_{k=1}^{n} f_k(x_k)$$

и выражает собой показатель оптимизации процесса. При этом видно, что показатель имеет аддитивную форму.

Если в начале шага k имеется остаток средств E_{k-1}, то можно за оставшиеся n-(k-1) шагов получить прибыль, равную $Z_k = \sum_{i=k}^{n} f_i(x_i)$, распределив средства между предприятиями P_k, P_{k+1}, ..., P_n.

Максимальная прибыль за n-(k-1) шагов будет зависеть от того, сколько осталось средств после распределения на предыдущих k-1 шагах, т.е. от значения E_{k-1}. Будем обозначать максимальную прибыль через $Z_k^*(E_{k-1})$.

При применении нисходящего подхода получим, что $Z_1^*(E_0) = Z_{max}$, т. е. $Z_1^*(E_0)$ – это суммарная прибыль за n шагов при оптимальном распределении E_0 средств между всеми предприятиями.

Рассмотрим детально k-ый шаг. Допустимым будет считаться выбор значения x_k, удовлетворяющего неравенству $0 \le x_k \le E_{k-1}$. Согласно принципу оптимальности в этом случае выделение x_k средств k-му предприятию принесет прибыль, равную $f_k(x_k)$, и оставшиеся средства $E_k = E_{k-1} - x_k$ нужно распределить между предприятиями P_{k+1}, ..., P_n так, чтобы получить максимальный доход $Z_{k+1}^*(E_k)$. Значит, величину x_k необходимо определять из максимальной суммы $f_k(x_k) + Z_{k+1}^*(E_k)$.

Таким образом, получаем рекуррентное соотношение

$$Z_k^*(E_{k-1}) = \max_{0 \le x_k \le E_{k+1}} f_k(x_k) + Z_{k+1}^*(E_k) \quad .$$

Шаги будут выполняться, начиная с конца процесса распределения к его началу. В результате будут получены две последовательности:

$$Z_n^*(E_{n-1}), Z_{n-1}^*(E_{n-2}), \ldots, Z_1^*(E_0)$$

(условные максимальные доходы) и

$$x_n^*(E_{n-1}), x_{n-1}^*(E_{n-2}), \ldots, x_1^*(E_0)$$

(условные оптимальные управления).

На этом первый и главный этап условной оптимизации завершается.

Второй и последний этап – безусловной оптимизации. После получения функции $Z_1^*(E_0)$, по значению E_0^* определяется $Z_{max} = Z_1^*(E_0)$. Теперь используем последовательность условных оптимальных управлений. Первому предприятию выделяем $x_1^* = x_1^*(E_0^*)$ средств; остаток средств для распределения будет равен $E_1^* = E_0^* - x_1^*$. По этому значению находим оптимальное значение $x_2^* = x_2^*(\xi_1^*)$, которое выделяется второму предприятию. Снова определяем $E_2^* = E_1^* - x_2^*$ и находим x_3^* и т.д., пока не будет найдено искомое управление (x_1^*, x_2^*, ..., x_n^*).

Изложенный алгоритм является алгоритмом решения задачи распределения ресурсов методом динамического программирования.

ГЛАВА 2 НЕКОТОРЫЕ ОПТИМИЗАЦИОННЫЕ ЗАДАЧИ И ИХ РЕШЕНИЕ

2.1 Решение задачи коммивояжера

В качестве примера возьмем полный взвешенный ориентированный граф без петель G, заданный матрицей

$$S = \begin{matrix} 0 & 6 & 2 & 5 \\ 3 & 0 & 4 & 6 \\ 2 & 5 & 0 & 3 \\ 4 & 6 & 5 & 0 \end{matrix}.$$

Цель задачи - отыскать гамильтонов цикл минимального веса.

Пусть i – произвольная вершина графа, а V – любое подмножество вершин, не включающее вершины 1 и i. M(i, V) – совокупность путей, которые берут начало в вершине i и, проходя через вершины множества V, оканчиваются в вершине 1. B(i, V) – длина кратчайшего пути множества M(i, V).

Во-первых, пользуясь формулой (1), вычислим значения длин путей для множества M(i, V), где V – одноэлементные множества {j}, j≠1 и j≠i:

$$B(2, 3) = s_{23} + s_{31} = 6;$$

$$B(2, 4) = s_{24} + s_{41} = 10;$$

$$B(3, 2) = s_{32} + s_{21} = 8;$$

$$B(3, 4) = s_{34} + s_{41} = 7;$$

$$B(4, 2) = s_{42} + s_{21} = 9;$$

$$B(4, 3) = s_{43} + s_{31} = 7.$$

Далее по формуле (2) находим значения длин путей для M(i, V), где V – двухэлементные множества {j,k}, j и k не равны 1 и i:

$$B(2, \underline{3}, 4) = \min(s_{23} + B(3, 4), s_{24} + B(4, 3)) = \min(11, 13) = 11;$$

$$B(3, 2, \underline{4}) = \min(s_{32} + B(2, 4), s_{34} + B(4, 2)) = \min(15, 12) = 12;$$

$$B(4, \underline{2}, 3) = \min(s_{42} + B(2, 3), s_{43} + B(3, 2)) = \min(12, 13) = 12.$$

Подчеркнутые в левой части значения j указывают на вершины, через которые проложены кратчайшие пути.

Последним находим значение длины кратчайшего пути для M(i, V), где V – множество всех вершин, за исключением вершины 1.

$$B\ 1,\ 2,\underline{3},4\ = \min\ s_{12} + B\ 2,\ \underline{3},4\ ,s_{13} + B\ 3,\ 2,\underline{4}\ ,s_{14} + B\ 4,\ \underline{2},3\ =$$

$$= \min\ 6 + 11,2 + 12,5 + 12\ = 14.$$

Таким образом, мы получаем оптимальное значение равное 14.

Чтобы определить оптимальный маршрут, используем сделанные подчеркивания:

$$1 \to 3 \to 4 \to 2 \to 1.$$

2.2 Решение задачи о ранце

Разберем пример задачи о ранце 0-1.

Даны 5 предметов и ранец вместимостью 9; стоимости предметов равны 4, 3, 5, 7 и 6; веса предметов соответственно 2, 1, 3, 4 и 3. Каждый предмет можно поместить в ранец только целиком и только в одном экземпляре. Требуется найти совокупность предметов, которые следует поместить в ранец, учитывая, что суммарная стоимость предметов в ранце должна быть максимальной при условии, что суммарный вес предметов в ранце не может превысить 9.

$$4x_1 + 3x_2 + 5x_3 + 7x_4 + 6x_5 \to max$$

при условиях

$$2x_1 + x_2 + 3x_3 + 4x_4 + 3x_5 \leq 9$$

$$x_i \in\ 0,1\ ,i = 1,5.$$

Как было описано в разделе 1.3.2, задача решается в виде процесса заполнения строк таблицы стоимостей сверху вниз. В таблице строки будут соответствовать значениям k – количество первых предметов из основной совокупности, которые можно положить в ранец. Столбцам соответствует значение p – суммарный вес, который взятые предметы не могут превысить. (Таблица 2.1).

В каждую клетку будет вноситься оптимальное значение стоимости выбранных предметов (они тоже указываются в скобках малыми цифрами) при соответствующих k и p.

Заполнение первой строки осуществляется согласно формуле (6). Если подставить данные задачи в формулу, то получим

$$B^*(1,p) = \begin{matrix} 0 \text{ при } p \in \{0,1,2,3\}; \\ 3 \text{ при } p \in \{4,5,\dots,9\}. \end{matrix}$$

Вторая и следующие строки заполняются в соответствии с формулой(7). Смысл формулы заключается в том, что для каждой клетки таблицы стоимостей совершается выбор. Если в ранце есть место для k+1-го предмета, то сравниваются два значения:

- значение в клетке строкой выше (k),
- сумма стоимости k+1-го предмета и значения в клетке, отстоящей влево на значение веса k+1-го предмета (оказываемся в строке k, столбце p-v_{k+1}).

Выбирается наибольшее из этих двух значений и записывается в ячейку таблицы на пересечении k+1 и p.

Таблица 2.1 – Таблица стоимостей

k \ p	1	2	3	4	5	6	7	8	9
1	0	$4_{(1)}$	$4_{(1)}$	$4_{(1)}$	$4_{(1)}$	$4_{(1)}$	$4_{(1)}$	$4_{(1)}$	$4_{(1)}$
2	$3_{(2)}$	$4_{(1)}$	$7_{(1,2)}$	$7_{(1,2)}$	$7_{(1,2)}$	$7_{(1,2)}$	$7_{(1,2)}$	$7_{(1,2)}$	$7_{(1,2)}$
3	$3_{(2)}$	$4_{(1)}$	$7_{(1,2)}$	$8_{(2,3)}$	$9_{(1,3)}$	$10_{(1,2,3)}$	$10_{(1,2,3)}$	$10_{(1,2,3)}$	$10_{(1,2,3)}$
4	$3_{(2)}$	$4_{(1)}$	$7_{(1,2)}$	$8_{(2,3)}$	$10_{(2,4)}$	$11_{(1,4)}$	$14_{(1,2,4)}$	$15_{(2,3,4)}$	$16_{(1,3,4)}$
5	$3_{(2)}$	$4_{(1)}$	$7_{(1,2)}$	$9_{(2,5)}$	$10_{(2,4)}$	$13_{(1,2,5)}$	$14_{(1,2,4)}$	$16_{(2,4,5)}$	$17_{(1,4,5)}$

Содержимое нижней правой клетки в заполненной таблице стоимостей – это оптимальное значение критерия в данной задаче. Таким образом, оптимальным выбором будет взять предметы 1, 4 и 5.

2.3 Решение задачи отыскания кратчайших путей на графе

Рассмотрим в качестве примера конечный взвешенный ориентированный граф G, состоящий из 7 вершин и заданный матрицей

$$M = \begin{matrix} 0 & 11 & 5 & \infty & 7 & 2 & 14 \\ \infty & 0 & 3 & \infty & 5 & 6 & 9 \\ 10 & 4 & 0 & 2 & \infty & \infty & 7 \\ \infty & 8 & \infty & 0 & 3 & 1 & 7 \\ \infty & 6 & 8 & 10 & 0 & 9 & \infty \\ \infty & 2 & 4 & 5 & \infty & 0 & 3 \\ 4 & 2 & 7 & 8 & \infty & 13 & 0 \end{matrix}$$

Числовой элемент m_{ij} этой матрицы равен весу дуги (i,j) графа G; если дуга в графе отсутствует, то $m_{ij} = \infty$. Веса дуг трактуются как их длины. Требуется найти расстояния от вершины 1 до остальных вершин графа

Как было описано в разделе 1.3.3, решение задачи будет оформлено в виде процесса заполнения таблицы. Строки этой таблицы соответствуют вершинам графа. Таблица будет заполняться по столбцам, слева направо.

На первом шаге заполняется столбец «1», в клетки строки j вносятся значения дуг c(1,j) из первой строки данной матрицы. Наименьшее значение в столбце выделяется символом *. Это значение является длиной наикратчайшего пути s(j) от вершины 1 до вершины j, где j номер строки выделенного значения. Когда s(j) получено, строка j считается помеченной и дальнейшие клетки этой строки не вычисляются, а заполняются символами «-». В данном примере s(6)=2 (минимальное в столбце) и строка 6 считается помеченной. Так как s(1)=0 (расстояние от вершины 1 до неё самой), то также первая строка считается отмеченной (Таблица 2.2).

Таблица 2.2 – Первый шаг алгоритма нахождения кратчайших путей

	«1» s(1)=0								
1	-	-	-	-	-	-	-	-	-
2	11								
3	5								
4	∞								
5	7								
6	2*	-	-	-	-	-	-	-	-
7	14								

Каждый следующий шаг заполняются по два столбца – четные и нечетные. Четные столбцы получают название, соответствующее вершине, для которой на предыдущем шаге был получен наикратчайший путь. Нечетные столбцы получают название «min».

На втором шаге второй столбец получает заголовок «6», так как на первом шаге получен наикратчайший путь s(6)=2 (что тоже указывается в заголовке). Второй столбец заполняется суммами s(6)+c(6,j), за исключением помеченных строк. В каждую клетку третьего столбца вносится значение, являющееся минимальным из предыдущих двух столбцов в той же строке. Как и на первом шаге, выбирается минимальное значение из третьего столбца и помечается символом *. В данном примере это значение s(2)=4, строка 2 становится помеченной. (Таблица 2.3)

Таблица 2.3 – Второй шаг алгоритма нахождения кратчайших путей

	«1» s(1)=0	«6» s(6)=2	min						
1	-	-	-	-	-	-	-	-	-
2	11	4	4*	-	-	-	-	-	-
3	5	6	5						
4	∞	7	7						
5	7	∞	7						
6	2*	-	-	-	-	-	-	-	-
7	14	5	5						

Следующие шаги выполняются аналогично второму шагу. На каждом шаге определяется кратчайшее расстояние от вершины 1 до еще одной вершины. (Таблица 2.4)

Таблица 2.4 – Результат алгоритма нахождения кратчайших путей

	«1» s(1)=0	«6» s(6)=2	min	«2» s(2)=4	min	«3» s(3)=5	min	«7» s(7)=5	min	«4» s(4)=7	min
1	-	-	-	-	-	-	-	-	-	-	-
2	11	4	4*	-	-	-	-	-	-	-	-
3	5	6	5	7	5*	-	-	-	-	-	-
4	∞	7	7	∞	7	7	7	14	7*	-	-
5	7	∞	7	9	7	∞	7	∞	7	10	7*
6	2*	-	-	-	-	-	-	-	-	-	-
7	14	5	5	13	5	12	5*	-	-	-	-

В результате работы алгоритма получаем, что от вершины 1 до вершин 2, 3, 4, 5, 6, 7 значения кратчайших путей равны соответственно 4, 5, 7, 7, 2 и 5.

2.4 Решение задачи о порядке произведения матриц

Рассмотрим пример, когда матрицы M_1, M_2, M_3 и M_4 имеют размеры 20x30, 30x60, 60x5 и 5x100 соответственно.

Обозначим через m_{ij} минимальное число элементарных операций, требуемое для расчета произведения $M_i \times M_{i+1} \times \ldots M_j$. Допустим, матрица M_i имеет размерность $q_i \times q_{i+1}$, тогда умножение матрицы M_i на матрицу M_{i+1} требует выполнения $2 \cdot q_i \cdot q_{i+1} \cdot q_{i+2}$ элементарных операций.

m_{ij} вычисляются последовательно, в порядке возрастания разности j-i.

Если j=i, то $m_{ij}=0$.

Пользуясь формулой (11), полученной в разделе 1.3.5, получаем:

$$m_{12} = 2q_1q_2q_3 = 2 \cdot 20 \cdot 30 \cdot 60 = 72000;$$

$$m_{23} = 2q_2q_3q_4 = 2 \cdot 30 \cdot 60 \cdot 5 = 18000;$$

$$m_{34} = 2q_3q_4q_5 = 2 \cdot 60 \cdot 5 \cdot 100 = 60000.$$

Для дальнейших вычислений используется формула (12):

$$m_{13} = \min\ m_{11} + m_{23} + 2q_1q_2q_4, m_{12} + m_{33} + 2q_1q_3q_4\ =$$
$$= \min\ 0 + 18000 + 6000, 72000 + 0 + 12000\ = 24000$$

$$m_{24} = \min\ m_{22} + m_{34} + 2q_2q_3q_5, m_{23} + m_{44} + 2q_2q_4q_5\ =$$
$$= \min\ 0 + 60000 + 360000, 18000 + 0 + 30000\ = 48000$$

$$m_{14} = \min\{m_{11} + m_{24} + 2q_1q_2q_5, m_{12} + m_{34} + 2q_1q_3q_5, m_{13} + m_{44} +$$
$$+2q_1q_4q_5\} = \min\{0 + 48000 + 120000, 72000 + 60000 +$$
$$+240000, 24000 + 0 + 20000\} = 44000\,.$$

Полученное значение m_{14} является оптимальным. Схему, соответствующую оптимальному перемножению, легко получить, просмотрев в обратном порядке выбранные оптимальные решения на каждом шаге:

$$M_1 \times\ M_2 \times M_3\ \ \times M_4\ .$$

2.5 Решение задачи распределения ресурсов

Рассмотрим задачу при конкретных заданных условиях. Необходимо распределить начальную сумму C = 200 млн.руб. Средства планируется распределить между четырьмя предприятиями $П_1$, $П_2$, $П_3$, $П_4$. Средства выделяются в размерах, кратных 40 млн.руб. Функции дохода заданы в таблице 2.5:

Таблица 2.5 – Значения функций дохода

f / u	$f_1(u)$	$f_2(u)$	$f_3(u)$	$f_4(u)$
40	8	6	3	4
80	10	9	4	6
120	11	11	7	8
160	12	13	11	13
200	18	15	18	16

Решение данного примера будет выполнено обратным методом Беллмана, описанным в разделе 1.3.6. Запишем необходимые рекуррентные соотношения:

$$B\ t,U\ = \begin{matrix} B\ n,U\ = 0, U \in\ 0,1,\dots,C\ , \\ \max_{u_{t+1} \in \{0,1,\dots,\min\ c_{t+1},U\ \}} f_{t+1}\ u_{t+1}\ + B\ t+1, U-u_{t+1}\ \end{matrix} . \quad (15)$$

В процессе вычисления найденные значения функции B(t, U) будем вносить в таблицу 2.6. Параметр t – момент дискретного времени – определяет столбец таблицы, а параметр U – нераспределенная к данному моменту сумма – определяет строку. В каждую клетку таблицы одновременно с B(t, U) вносится в квадратных скобках значение u_{t+1}, при котором достигается оптимальное B(t, U).

Как видно из описанной выше формулы (15), B(4, U)=0, поэтому процесс вычисления начинается с B(3, U). Также из формулы (15) следует, что B(3, U)=f_4(U).

Из соотношения (15) легко вывести формулы для заполнения столбца «t=2»:

$$B\ 2{,}40\ = \max_{u_3 \in\ 0{,}40}\ f_3\ u_3\ + B\ 3{,}40 - u_3\ \ = = \max\ f_3\ 0\ +$$

$$B\ 3{,}40\ , f_3\ 40\ + B\ 3{,}0\ \ = \max\ 4, 3\ = 4.$$

В этом случае, как и в последующих, прекрасно видно, при каком значении u_3 достигается максимальное значение:

$$B\ 2{,}80\ = \max_{u_3 \in\ 0{,}40{,}80}\ f_3\ u_3\ + B\ 3{,}80 - u_3\ \ =$$

$$= \max\ f_3\ 0\ + B\ 3{,}80\ , f_3\ 40\ + B\ 3{,}40\ , f_3\ 80\ + B\ 3{,}0$$

$$= \max\ 6, 7, 4\ = 7$$

$$B\ 2{,}120\ = \max\ f_3\ 0\ + B\ 3{,}120\ , f_3\ 40\ + B\ 3{,}80\ , f_3\ 80$$

$$+ B\ 3{,}40\ , f_3\ 120\ + B\ 3{,}0\ \ = \max\ 8, 9, 8, 7\ = 9$$

$$B\ 2{,}160\ = \max\ f_3\ 0\ + B\ 3{,}160\ , f_3\ 40\ + B\ 3{,}120\ , f_3\ 80$$

$$+ B\ 3{,}80\ , f_3\ 120\ + B\ 3{,}40\ , f_3\ 160\ + B\ 3{,}0$$

$$= \max\ 13, 11, 10, 11, 11\ = 13$$

$$B\ 2{,}200\ = \max\ f_3\ 0\ + B\ 3{,}200\ , f_3\ 40\ + B\ 3{,}160\ , f_3\ 80$$

$$+ B\ 3{,}120\ , f_3\ 120\ + B\ 3{,}80\ , f_3\ 160\ + B\ 3{,}40\ , f_3\ 200$$

$$+ B\ 3{,}0\ \ = \max\ 16, 16, 12, 15, 15, 18\ = 18$$

Формулы для вычисления столбцов «t=1» и «t=0» выглядят аналогично (меняются лишь значения U).

Таблица 2.6 – Значения функции B(t, U)

U \ t	t=0	t=1	t=2	t=3
0	0[0]	0[0]	0[0]	0[0]
40	8[40]	6[40]	4[0]	4[40]
80	14[40]	10[40]	7[40]	6[80]
120	18[40]	13[80]	9[40]	8[120]
160	21[40]	16[80]	13[0]	13[160]
200	24[40]	19[40]	18[200]	16[200]

Оптимальное решение выглядит следующим образом:

1) Максимальный доход равен 24, может быть достигнут при вложении в первое предприятие 40 млн.руб (значение в квадратных скобках).
2) Осталось 160 млн.руб. Пересечение строки с соответствующим значением и столбца «t=1» приводит нас к тому, что во второе предприятие вкладывается 80 млн.руб.
3) На пересечении строки «80» и столбца «t=2» виден вклад в третье предприятие 40 млн.руб.
4) В последнее – четвертое предприятие – также необходимо вложить 40 млн.руб.

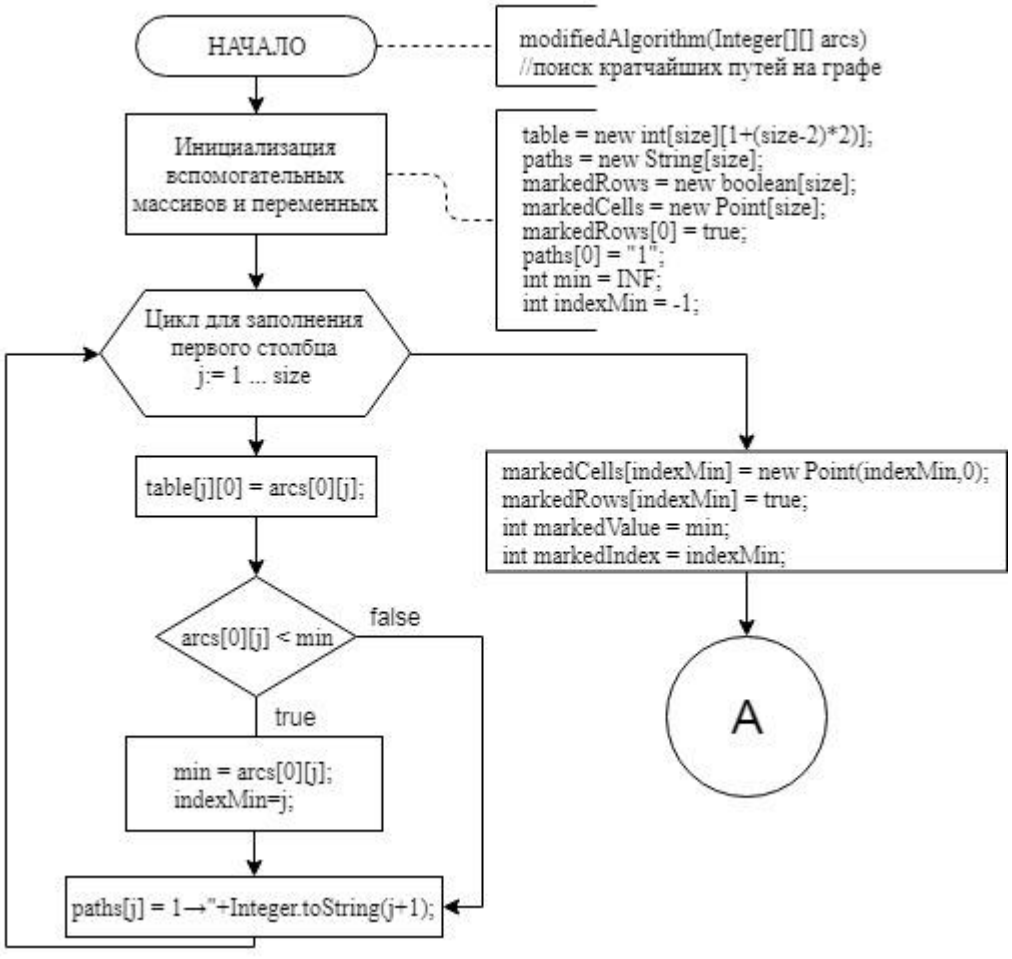
В результате вычислен максимальный доход, равный 24 млн.руб. Этот результат достигается, если первому предприятию выделить 40 млн.руб., второму – 80, третьему и четвертому по 40 млн.руб.

ГЛАВА 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА РЕШЕНИЯ ЗАДАЧИ ОТЫСКАНИЯ КРАТЧАЙШИХ ПУТЕЙ НА ГРАФЕ

3.1 Алгоритм решения задачи отыскания кратчайших путей на графе

В задаче отыскания кратчайших путей входными данными является матрица смежности графа *C*. Цель алгоритма – рассчитать все кратчайшие пути из вершины 1 до остальных вершин.

На первом шаге алгоритма определяется ближайшая вершина и длина пути до нее, равная соответствующему значению из матрицы смежности *C*. Полученная ближайшая вершина маркируется. Промежуточные значения длин путей от вершины 1 до остальных вершин сохраняются во вспомогательный массив.

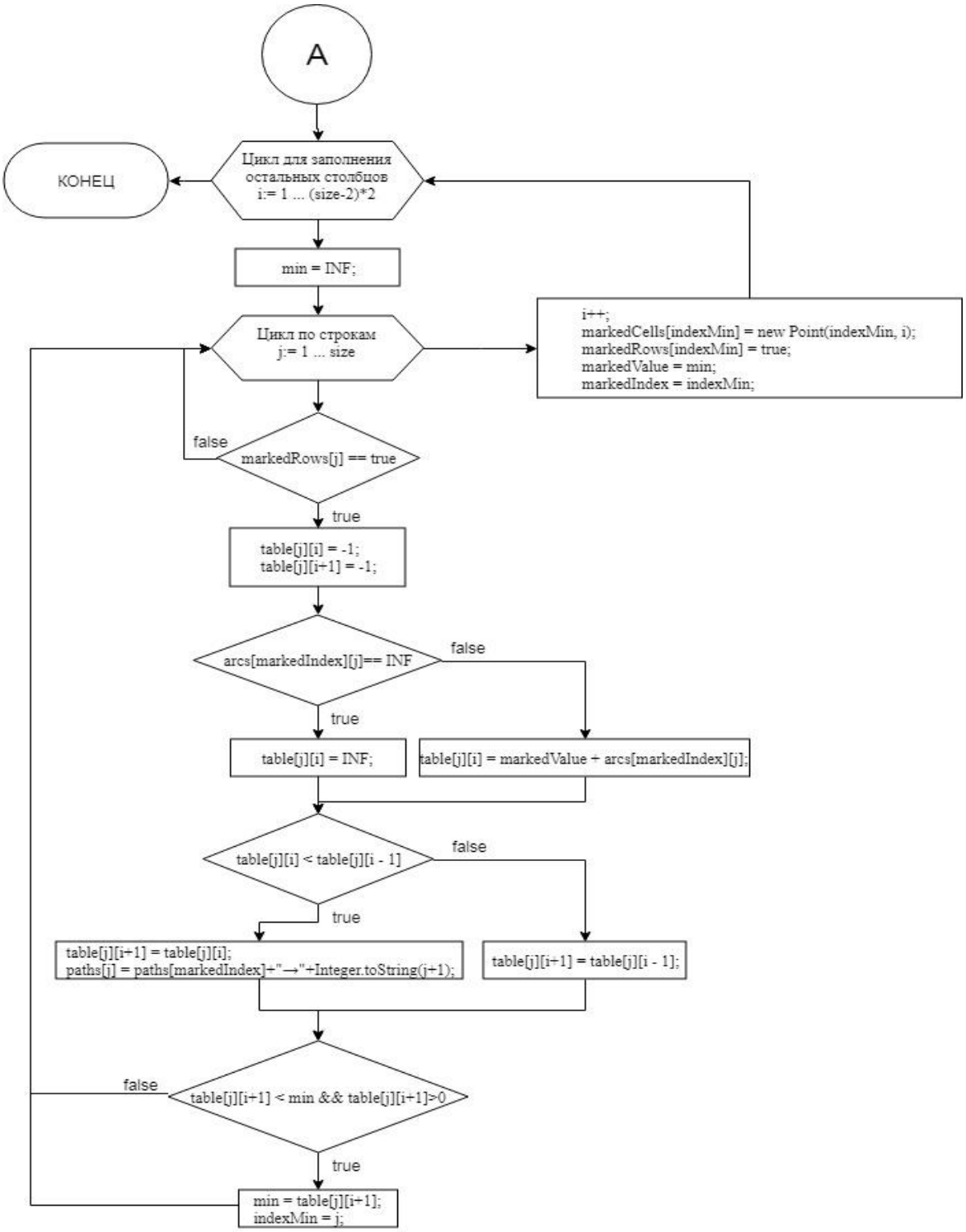


Блок-схема 1 Инициализация вспомогательных переменных и первый шаг алгоритма

На последующих шагах вычисляются значения длин путей от вершины 1 до остальных немаркированных вершин. Длины путей на этих шагах вычисляются в соответствии с рекуррентным соотношением

$$s\ 1, N\backslash H\ = \min_{i\in H\ , j\in N\backslash H} (s(i)\ +\ \mathrm{c}(i,j)).$$

Ближайшая вершина, полученная на каждом шаге, маркируется.



Блок-схема 2 Второй и последующие шаги алгоритма

Данный алгоритм также был дополнен определением последовательностей пройденных вершин, через которые проходят кратчайшие пути. Все промежуточные вычисления алгоритма сохраняются во вспомогательный массив, позволяя получить наглядную таблицу выполнения шагов алгоритма.

Алгоритм был реализован на языке Java. Исходный код алгоритма находится в классе **ShortestPaths**.java (Приложение А).

3.2 Описание интерфейса

В разработанной программе входная матрица смежности графа читается из выбранного файла. Матрица смежности в файле задается следующим образом:

- указывается размерность матрицы;
- записываются целочисленные веса ребер графа;
- в качестве разделителей между числами используется символ пробела.

На рисунках 1 и 2 представлен интерфейс стартового окна и окна выбора файла матрицы смежности.

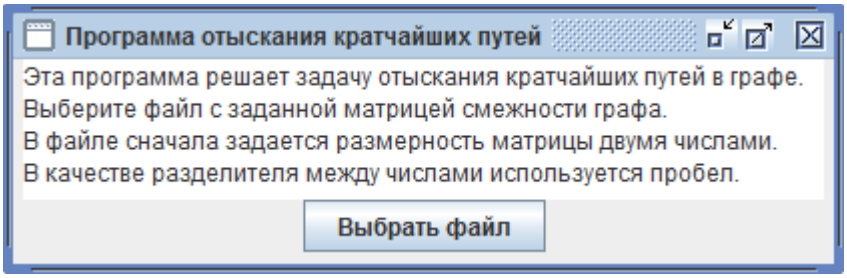


Рисунок 1 – Стартовое окно реализованной программы

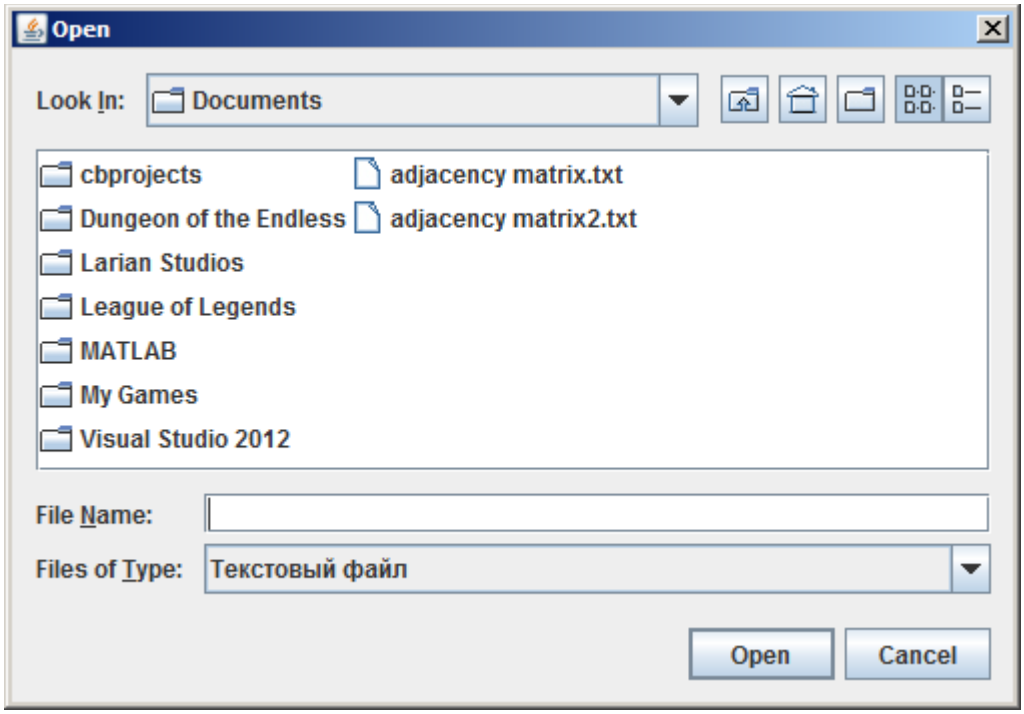
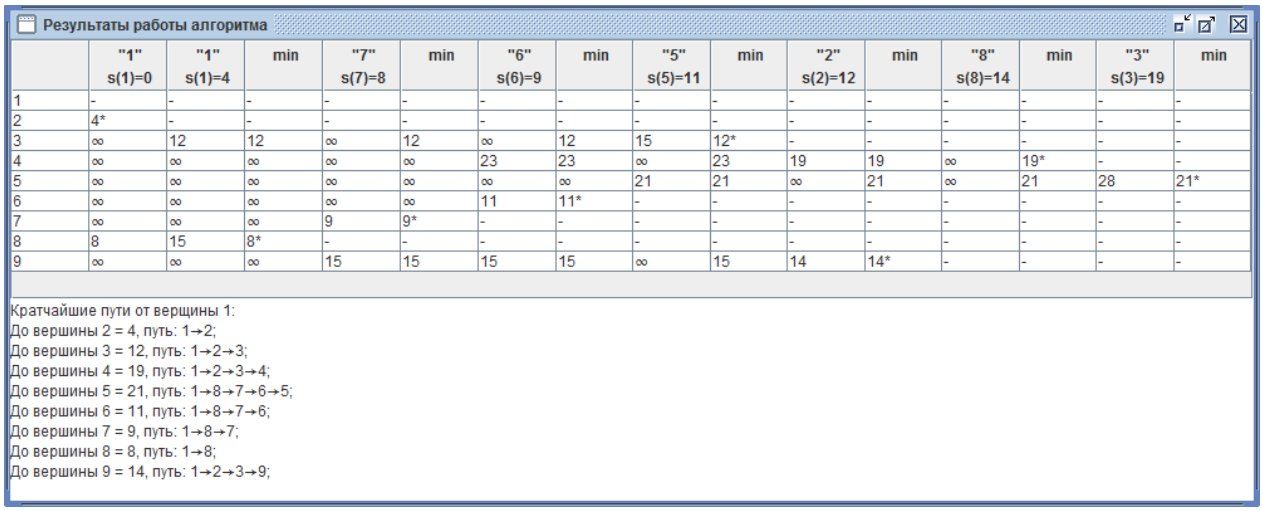


Рисунок 2 – Окно выбора файла матрицы смежности

При выборе корректного файла программа выполняет реализованный алгоритм и открывается окно результатов (Рисунок 3).



	"1" s(1)=0	"1" s(1)=4	min	"7" s(7)=8	min	"6" s(6)=9	min	"5" s(5)=11	min	"2" s(2)=12	min	"8" s(8)=14	min	"3" s(3)=19	min
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2	4*	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	∞	12	12	∞	12	∞	12	15	12*	-	-	-	-	-	-
4	∞	∞	∞	∞	∞	23	23	∞	23	19	19	∞	19*	-	-
5	∞	∞	∞	∞	∞	∞	∞	21	21	∞	21	∞	21	28	21*
6	∞	∞	∞	∞	∞	11	11*	-	-	-	-	-	-	-	-
7	∞	∞	∞	9	9*	-	-	-	-	-	-	-	-	-	-
8	8	15	8*	-	-	-	-	-	-	-	-	-	-	-	-
9	∞	∞	∞	15	15	15	15	∞	15	14	14*	-	-	-	-

Рисунок 3 – Окно результатов выполнения реализованного алгоритма

На окне результатов можно увидеть наглядную таблицу выполнения отдельных шагов алгоритма. Подробное описание таблицы было рассмотрено в разделе 2.3.

Под таблицей выведены длины кратчайших путей, а также последовательности вершин, через которые проходят соответствующие кратчайшие пути.

ЗАКЛЮЧЕНИЕ

В результате изучения и анализа метода динамического программирования показан широкий спектр задач, к которым данный метод применим. Представлены подходы использования метода динамического программирования.

В работе были подробно рассмотрены разные виды оптимизационных задач и особенности применения к ним метода динамического программирования для решения в общем виде.

В рамках данной работы описаны алгоритмы решения оптимизационных задач, полученные методом динамического программирования. Работа алгоритмов продемонстрирована на конкретных примерах оптимизационных задач.

Применение метода динамического программирования к оптимизационным задачам малой размерности позволило получить эффективные алгоритмы решения, в редких случаях уступающих альтернативным алгоритмам. Однако при увеличении размерности задачи сильно увеличивается трудоемкость решения и объём памяти, требуемый для хранения результатов решения большого количество подзадач.

Задача отыскания кратчайших путей является удачным примером применения метода динамического программирования. На основе полученного алгоритма решения этой задачи была реализована программа на языке программирования Java. Алгоритм дополнен определением совокупностей вершин, через которые проходят вычисленные кратчайшие пути.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

Научная и методическая литература

1. Акулич И.Л. Математическое программирование в примерах и задачах: учебное пособие/ И.Л. Акулич – СПб.: Издательство «Лань», 2011. – 352 с.
2. Алексеев О.Г. Комплексное применение методов дискретной оптимизации / О.Г. Алексеев – М.: Наука, 1987. – 247 с.
3. Банди Б. Основы линейного программирования / Б. Банди – М.: Радио и связь, 2009. – 176с.
4. Беллман Р. Динамическое программирование. /Р. Беллман – М.: Издательство иностранной литературы, 1960. – 400 с.
5. Беллман Р. Прикладные задачи динамического программирования / Р. Беллман, С. Дрейфус. – М.: Наука. Главная редакция Физико-математической литературы, 1965. – 460 с.
6. Вагнер Г. Основы исследования операций. Т. 1. / Г. Вагнер – М.: Мир, 1972. – 336 с.
7. Габасов Р. Основы динамического программирования / Р. Габасов, Ф.М. Кириллова – Мн.: Изд-во БГУ, 1975. – 264 с.
8. Интрилигатор М. Математические методы оптимизации и экономическая теория / М. Интрилигатор - М.: Прогресс, 2005. – 592с.
9. Карзаева, Н.Н. Математическое программирование в экономике: Учебное пособие / Н.Н. Карзаева. - М.: Финансы и статистика, 2010. - 240 с.
10. Коган Д.И. Динамическое программирование и дискретная многокритериальная оптимизация: учебное пособие / Д.И. Коган – М.: Издательство Нижегородского университета, 2004. - 150 с.
11. Корбут А.А. Дискретное программирование / А.А. Корбут, Ю.Ю. Финкельштейн. – М.: Наука, 1969. – 368 с.
12. Кристофидес Н. Теория графов. Алгоритмический подход / Н. Кристофидес – М.: Мир, 2008. – 432 с.

13. Морозов, В.В. Исследование операций в задачах и упражнениях / В.В. Морозов, А.Г. Сухарев, В.В. Федоров. - М.: КД Либроком, 2016. - 288 с.

14. Подиновский В.В. Парето - оптимальные решения многокритериальных задач / В.В. Подиновский, В.Д. Ногин – М.: Наука, 2002. – 256 с.

15. Соколов А.В. Методы оптимальных решений. Общие положения. Математическое программирование / А.В. Соколов, В.В. Токарев. - М.: Физматлит, 2011. - 564 с.

Литература на иностранном языке

16. Fletcher R. Practical Methods of Optimization. Vol. 1, Unconstrained Optimization, and Vol. 2, Constrained Optimization, John Wiley and Sons., 2000.

17. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C., Introduction to Algorithms (2nd ed.), MIT Press & McGraw–Hill, 2001, ISBN 0-262-03293-7 . pp. 344.

18. Gill P.E., W. Murray, M.A. Saunders, and M.H. Wright. Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints. ACM Trans. Math. Software, Vol. 10, pp. 282-298, 1984.

19. Sniedovich M., Dynamic Programming: Foundations and Principles, Taylor & Francis, 2010, ISBN 978-0-8247-4099-3

20. Zadeh L.A. Optimality and Nonscalar-valued Performance Criteria. IEEE Trans. Automat. Contr., Vol. AC-8, p. 1, 1963.

ПРИЛОЖЕНИЕ А

Листинг кода программной реализации задачи отыскания кратчайших путей на графе методом динамического программирования

MainFrame.java

```
package shortestpaths;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.Scanner;

import static shortestpaths.ShortestPaths.INF;

public class MainFrame extends JFrame implements ActionListener {
    JButton openButton;
    JFileChooser fc;
    Integer[][] adjMatrix;
    JPanel rootPanel;
    ResultsFrame rf;
    JLabel label;
    AdjMatrixTableModel model;
    JTable adJMatrixTable;
    public MainFrame() {
        super("Программа отыскания кратчайших путей");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
            rootPanel = new JPanel();
            rootPanel.setLayout(new
BoxLayout(rootPanel,BoxLayout.PAGE_AXIS));
            StringBuilder sb = new StringBuilder("Эта программа решает задачу
отыскания кратчайших путей в графе.\n");
            sb.append("Выберите файл с заданной матрицей смежности
графа.\n");
            sb.append("В файле сначала задается размерность матрицы двумя
числами.\n");
            sb.append("В качестве разделителя между числами используется
пробел.");
            JTextArea about = new JTextArea(sb.toString());
            about.setAlignmentX(Component.CENTER_ALIGNMENT);
            about.setMaximumSize(new Dimension(400,  70));
            rootPanel.add(about);
            fc = new JFileChooser();
            fc.setFileFilter(new FileNameExtensionFilter("Текстовый файл",
"txt"));
            openButton = new JButton("Выбрать файл");
            openButton.addActionListener(this);
            openButton.setAlignmentX(Component.CENTER_ALIGNMENT);
            rootPanel.add(openButton);

            getContentPane().add(rootPanel);
            setPreferredSize(new Dimension(420,  135));
            pack();
            setVisible(true);
        }
        public static void main(String[] args) {
            javax.swing.SwingUtilities.invokeLater(()  -> {
```

```
        JFrame.setDefaultLookAndFeelDecorated(true);
        new MainFrame();
    });
}
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == openButton) {
        int returnVal = fc.showOpenDialog(MainFrame.this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fc.getSelectedFile();
            readGraphFromFile(file);

            if (adjMatrix!=null) {
                if(label==null) {
                    label = new JLabel();
                    label.setText("Выбранная матрица смежности:");
                    label.setAlignmentX(Component.CENTER_ALIGNMENT);
                    rootPanel.add(label);
                }
                model = new AdjMatrixTableModel(adjMatrix);
                if(adJMatrixTable==null) {
                    adJMatrixTable = new JTable(model);
                    rootPanel.add(adJMatrixTable);
                    setPreferredSize(new Dimension(400, 300));
                }
                else
                    adJMatrixTable.setModel(model);
                pack();
                repaint();
                if (rf!=null)
```

```
                rf.dispose();
            rf = new ResultsFrame(adjMatrix);
            rf.setVisible(true);
        }
    }
  }
}

private void readGraphFromFile(File f) {
  adjMatrix = null;
  try(Scanner sc = new Scanner(f)) {
    String line = null;
    int rows = sc.nextInt();
    int columns = sc.nextInt();
    adjMatrix = new Integer[rows][columns];
    for (int i = 0; i < rows; i++) {
      for (int j = 0;j < columns; j++) {
        int tempInt = sc.nextInt();
        adjMatrix[i][j] = (tempInt==0)?INF:tempInt;
      }
    }
  } catch(Exception e) {
    e.printStackTrace();
  }
}

public class AdjMatrixTableModel extends DefaultTableModel {
  public AdjMatrixTableModel(Integer[][] data) {
    super(data,(new String[data.length]));
  }
```

```
        @Override
        public Object getValueAt(int row, int column) {
            if((Integer)super.getValueAt(row,column)==INF)
                return 0;
            else
                return super.getValueAt(row,column);
        }
    }
}
```

ResultFrame.java

```
package shortestpaths;
import javax.swing.*;
import javax.swing.event.TableModelListener;
import javax.swing.table.TableCellRenderer;
import javax.swing.table.TableColumn;
import javax.swing.table.TableModel;
import java.awt.*;
import java.util.Enumeration;
import java.util.HashSet;
import java.util.Set;
import static shortestpaths.ShortestPaths.INF;
public class ResultsFrame extends JFrame {
    private ShortestPaths dp;
    public ResultsFrame(Integer[][] adjMatrix)  {
        super("Результаты работы алгоритма");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        dp = new ShortestPaths(adjMatrix);
        JPanel rootPanel = new JPanel();
```

```
            rootPanel.setLayout(new
BoxLayout(rootPanel,BoxLayout.PAGE_AXIS));
            TableModel model = new ResultTableModel(dp.getTable());
            JTable table = new JTable(model);
            MultiLineHeaderRenderer  renderer = new MultiLineHeaderRenderer();
            Enumeration e = table.getColumnModel().getColumns();
            while (e.hasMoreElements()) {
               ((TableColumn) e.nextElement()).setHeaderRenderer(renderer);
            }
            rootPanel.add(new JScrollPane(table));
            StringBuilder results = new StringBuilder("Кратчайшие пути от
вершины 1:\n");
            for (int i = 1; i < dp.getTable().length; i++) {
               results.append("До вершины ");
               results.append(i+1);
               results.append(" = ");
               results.append(dp.getTable()[i][dp.getMarkedCells()[i].y()]);
               results.append(", путь: ");
               results.append(dp.getPaths()[i]);
               results.append(";\n");
            }
            JTextArea resultsTextArea = new JTextArea(results.toString());
            rootPanel.add(resultsTextArea);
            getContentPane().add(rootPanel);
            rootPanel.setMinimumSize(new  Dimension(500,  300));
            setPreferredSize(new Dimension(1000,  400));
            pack();
            setLocationRelativeTo(null);
            setVisible(true);
         }
```

```
public class ResultTableModel implements TableModel {
    private Set<TableModelListener> listeners = new
HashSet<TableModelListener>();
    private int[][] data;
    public ResultTableModel(int[][] data) {
        this.data = data;
    }
    public void addTableModelListener(TableModelListener listener) {
        listeners.add(listener);
    }
    public Class<?> getColumnClass(int columnIndex) {
        return String.class;
    }
    public int getColumnCount() {
        return data[0].length+1;
    }
    public String getColumnName(int columnIndex) {
        return dp.getTableColumns()[columnIndex];
    }
    public int getRowCount() {
        return data.length;
    }
    public Object getValueAt(int rowIndex, int columnIndex) {
        if (columnIndex==0)
            return rowIndex+1;
        else
        if (data[rowIndex][columnIndex-1] == INF)
            return "∞";
        else if (data[rowIndex][columnIndex-1] == -1)
```

```
                return "-";
            else if (dp.getMarkedCells()!=null &&
(dp.getMarkedCells())[rowIndex].y()==columnIndex-1)
                return data[rowIndex][columnIndex-1]+"*";
            else
                return data[rowIndex][columnIndex-1];
        }
        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return false;
        }
        public void removeTableModelListener(TableModelListener listener) {
            listeners.remove(listener);
        }
        public void setValueAt(Object value, int rowIndex, int columnIndex) {
        }
    }
    class MultiLineHeaderRenderer extends JList implements
TableCellRenderer {
        public MultiLineHeaderRenderer() {
            setOpaque(true);
            setForeground(UIManager.getColor("TableHeader.foreground"));
            setBackground(UIManager.getColor("TableHeader.background"));
            setBorder(UIManager.getBorder("TableHeader.cellBorder"));
            ListCellRenderer renderer = getCellRenderer();
            ((JLabel) renderer).setHorizontalAlignment(JLabel.CENTER);
            setCellRenderer(renderer);
        }
        @Override
        public Component getTableCellRendererComponent(JTable table,
Object value, boolean isSelected, boolean hasFocus, int row, int column) {
```

```
            int width = table.getColumnModel().getColumn(column).getWidth();
            String[] data = ((String)value).split(" ");
            this.setListData(data);
            setSize(width, getPreferredSize().height);
            return this;
        }
    }
}
```

ShortestPaths.java

```
package shortestpaths;

import java.util.Arrays;

public class ShortestPaths {
    final class Point {
        private final int x;
        private final int y;
        public Point(int x, int y) {
            this.x = x;
            this.y = y;
        }
        public int y() { return y; }
    }
    static int INF = Integer.MAX_VALUE;
    private int[][] table;
    private String[] tableColumns;
    private Point[] markedCells;
    private boolean[] markedRows;
```

```
private String[] paths;
public int[][] getTable() { return table; }
public String[] getTableColumns() { return tableColumns; }
public Point[] getMarkedCells() { return markedCells; }
public String[] getPaths() { return paths; }
public void modifiedAlgorithm(Integer[][] arcs) {
    int size = arcs.length;
    paths = new String[size];
    markedRows = new boolean[size];
    Arrays.fill(markedRows,false);
    markedCells = new Point[size];
    //первый шаг
    markedRows[0] = true;
    paths[0] = "1";
    int min=INF;
    int indexMin=-1;
    for (int j = 1; j < size; j++) {
        table[j][0] = arcs[0][j];
        if(arcs[0][j] < min) {
            min = arcs[0][j];
            indexMin=j;
        }
        paths[j] = "1→"+Integer.toString(j+1);
    }
    markedCells[indexMin] = new Point(indexMin,0);
    markedRows[indexMin] = true;
    int markedValue = min;
    int markedIndex = indexMin;
    tableColumns[1] = "\"1\" s(1)=0";
    //цикл для остальных шагов
```

```
            for (int i = 1; i < (size-2)*2; i++) {
                tableColumns[i+1]  =  "\""+markedIndex+"\"  s("+markedIndex+")
="+markedValue;
                tableColumns[i+2] = "min";
                min = INF;
                for (int j = 1; j < size; j++) {
                    if (markedRows[j]) {
                        table[j][i] = -1;
                        table[j][i+1] = -1;
                        continue;
                    }
                    //заполнение четных столбцов
                    if (arcs[markedIndex][j]== INF)
                        table[j][i] = INF;
                    else
                        table[j][i] = markedValue + arcs[markedIndex][j];
                    //заполнение нечетных столбцов
                    if (table[j][i] < table[j][i - 1]) {
                        table[j][i+1] = table[j][i];
                        paths[j] = paths[markedIndex]+"→"+Integer.toString(j+1);
                    } else {
                        table[j][i+1] = table[j][i - 1];
                    }

                    if (table[j][i+1] < min && table[j][i+1]>0) {
                        min = table[j][i+1];
                        indexMin = j;
                    }
                }
                i++;
```

```
            markedCells[indexMin] = new Point(indexMin,  i);
            markedRows[indexMin] = true;
            markedValue = min;
            markedIndex = indexMin;
        }
    }
    public ShortestPaths(Integer[][] arcsOfGraph){
        table = new int[arcsOfGraph.length][1+(arcsOfGraph.length-2)*2];
        tableColumns = new String[table[0].length+1];
        tableColumns[0] = "";
        for (int i = 0; i < table[0].length;  i++) {
            table[0][i] = -1;
        }
        modifiedAlgorithm(arcsOfGraph);
    }
    private void printArray(int[][]arr){
        for (int i = 0; i < arr.length;  i++) {
            System.out.print("[");
            for (int j = 0; j < arr[0].length; j++) {
                if (arr[i][j]== INF)
                    System.out.print("  ∞");
                else if (arr[i][j]==-1)
                    System.out.print("  -");
                else {
                    System.out.printf("%2d", arr[i][j]);
                    if (markedCells!=null && markedCells[i].y==j)
                        System.out.print("*");
                }
                if (j!=arr[0].length-1)
                    System.out.print(", ");
```

```
            }
            System.out.print("]\n");
        }
    }
}
```