УЗБЕКСКОЕ АГЕНТСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

САМАРКАНДСКИЙ ФИЛИАЛ

Ходжаев Т.Т., Абдуллаева Н.И.,Ходжаев Ш.

ТЕОРИЯ ПРИНЯТИЯ РЕШЕНИЙ

Лабораторный практикум

САМАРКАНД – 2008

УЗБЕКСКОЕ АГЕНТСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ

ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

САМАРКАНДСКИЙ ФИЛИАЛ

Ходжаев Т.Т., Абдуллаева Н.И., Ходжаев Ш.

ТЕОРИЯ ПРИНЯТИЯ РЕШЕНИЙ

Лабораторный практикум

(Методическое указание)

РЕКОМЕНДОВАННО
Учебно методическим советом Самаркандкого филиала ТУИТ для студентов обучающихся по направлению 5521900 - «Информатика и информационные технологии»

САМАРКАНД - 2008

Ходжаев Т.Т., Абдуллаева Н.И., Ходжаев Ш.
Теория принятия решений. Лабораторный практикум
Методическое указание. – Самарканд: Изд. СамГУ, 2008. – стр.

Методическое указание «Теория принятия решений» (лабораторный практикум) написано на основе программы одноименного предмета. В работе проработаны и изложены: сведения о Теории принятия решений, классификация задач теории принятия решений, примеры, иллюстрирующие принципы принятия решений в условиях риска и в условиях неопределенности. Проработаны и формализованы пять лабораторных работ: общая постановка задачи теории игр и методы ее решения, решение матричных игр, задача теории расписаний с двумя обслуживающими устройствами, задача о кратчайшем маршруте, применение метода динамического программирования к задаче оптимального управления запасами

Рецензенты:
Зав. кафедрой ТУИТ СФ,
доцент Бекмурадов К. А.
Доцент СамГУ Порсаев Г.М.

СОДЕРЖАНИЕ

Предисловие.

Современный этап развития средств и систем информационных технологий характеризуется переходом к созданию и внедрению информационных систем различного масштаба и назначения, качественное обоснование и оценка которых зависят от принятия эффективного решения. В этом плане, методы теории принятия решений являются эффективными средствами исследования, анализа рассматриваемых задач для достижения поставленной цели.

Отметим, что учитывая характер задач, решаемых различного рода специалистами, методы теории принятия решений позволяют сконцентрировать внимание лица, принимающего решение на конкретных концептуальных подходах.

В настоящем методическом указании подобраны и формализованы такие лабораторные работы, задачи которых вполне соответствуют классу задач теории принятия решений.

Приведем классификацию задач теории принятия решений.

Классификация задач теории принятия решений.

В зависимости от условий внешней среды и степени информативности лица, принимающего решение (ЛПР) производится следующая классификация задач принятия решений:

а) в условиях риска;

б) в условиях неопределённости;

в) в условиях конфликта или противодействия (активного противника).

Задачи, в которых принятие решений осуществляется в условиях риска, связаны с использованием критерия ожидаемого значения**.**

Использование критерия ожидаемого значения обусловлено стремлением максимизировать ожидаемую прибыль (или минимизировать ожидаемые затраты). Использование ожидаемых величин предполагает возможность многократного решения одной и той же задачи, пока не будут получены достаточно точные расчётные формулы. Математически это выглядит так: пусть X– случайная величина с математическим ожиданием MX и дисперсией DX. Если $x_1,x_2,\ldots,x_n$ – значения случайной величины X, то среднее арифметическое их (выборочное среднее) значений $\overline{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$ имеет дисперсию $\frac{DX}{n}$. Таким образом, когда $n \to \infty$, $\frac{DX}{n} \to 0$ и

$$\overline{x} \to MX.$$

Другими словами при достаточно большом объёме выборки разница между средним арифметическим и математическим ожиданием стремится к нулю (так называемая предельная теорема теории вероятности). Следовательно, использование критерия «ожидаемое значение» справедливо только в случае, когда одно и тоже решение приходится применять достаточно большое число раз. Верно и обратное: ориентация на ожидания будет приводить к неверным результатам для решений, которые приходится принимать небольшое число раз.

Изложенное выше проиллюстрируем на одном примере.

Пример 1. Требуется принять решение о том, когда необходимо проводить профилактический ремонт ПЭВМ, чтобы минимизировать потери из-за неисправности. В случае, если ремонт будет производиться слишком часто, затраты на обслуживание будут большими при малых потерях из-за случайных поломок.

Так как заранее невозможно предсказать когда возникнет неисправность, то необходимо найти вероятность того, что ПЭВМ выйдет из строя в период времени *t*. В этом и состоит элемент ”риска”.

Математически это выглядит так: ПЭВМ ремонтируется индивидуально, если она остановилась из-за поломки. Через *T* интервалов времени выполняется профилактический ремонт всех *n* ПЭВМ. Необходимо определить оптимальное значение *T*, при котором минимизируются общие затраты на ремонт неисправных ПЭВМ и проведение профилактического ремонта в расчёте на один интервал времени.

Пусть p_t – вероятность выхода из строя одной ПЭВМ в момент *t*, а n_t – случайная величина, равная числу всех вышедших из строя ПЭВМ в тот же момент. Пусть далее C_1 – затраты на ремонт неисправной ПЭВМ и C_2 – затраты на профилактический ремонт одной машины.

Применение критерия ожидаемого значения в данном случае оправдано, если ПЭВМ работают в течение большого периода времени. При этом ожидаемые затраты на один интервал составят

$$O3 = \frac{C_1 \sum_{t=1}^{T-1} M(n_t) + C_2 n}{T},$$

где $M(n_t)$ – математическое ожидание числа вышедших из строя ПЭВМ в момент *t*. Так как n_t имеет биномиальное распределение с параметрами (n, p_t), то $M(n_t) = np_t$. Таким образом

$$O3 = \frac{n(C_1 \sum_{t=1}^{T-1} p_t + C_2)}{T}$$

Необходимые условия оптимальности T^* имеют вид:

$$O3\ (T^*-1) \geq O3\ (T^*),$$
$$O3\ (T^*+1) \geq O3\ (T^*).$$

Следовательно, начиная с малого значения *T*, вычисляют *O3(T)*, пока не будут удовлетворены необходимые условия оптимальности.

Пусть $C_1 = 100$; $C_2 = 10$; $n = 50$. Значения p_t имеют вид:

T	p_t	$\sum_{t=1}^{T-1} p_t$	$O3(T)$
1	0.05	0	$\frac{50\ (100 \cdot 0 + 10)}{1} = 500$
2	0.07	0.05	375
3	0.10	0.12	366.7
4	0.13	0.22	400
5	0.18	0.35	450

$$T^* \to 3\,, \quad O3(T^*) \to 366.7$$

Следовательно профилактический ремонт необходимо делать через $T^*=3$ интервала времени.

Принятие решений в условиях неопределённости.

В задачах, где требуется принимать решение в условиях неопределенности, будем предполагать, что лицу, принимающему решение не противостоит разумный противник.

Данные, необходимые для принятия решения в условиях неопределенности, обычно задаются в форме матрицы, строки которой соответствуют возможным действиям, а столбцы – возможным состояниям системы.

Рассмотрим следующий пример. Пусть, из некоторого материала требуется изготовить изделие, долговечность которого при допустимых затратах невозможно определить. Нагрузки

считаются известными. Требуется решить, какие размеры должно иметь изделие из данного материала.

Предлогаемые варианты решения задачи следующие. Вначале введем следующие обозначения:

E_1 – выбор размеров из соображений максимальной долговечности ;

E_m– выбор размеров из соображений минимальной долговечности ;

E_i – промежуточные решения.

Условия, требующие рассмотрения таковы :

F_1 – условия, обеспечивающие максимальную долговечность;

F_n – условия, обеспечивающие минимальную долговечность;

F_i – промежуточные условия.

Под результатом решения $e_{ij} = e(E_i ; F_j)$ здесь можно понимать оценку, соответствующую варианту E_i и условиям F_j , и, характеризующие прибыль, полезность или надёжность. Обычно мы будем называть такой результат полезностью решения.

Тогда семейство (матрица) решений $\| e_{ij} \|$ будет иметь вид :

	F_1	F_2	. . .	F_n
E_1	e_{11}	e_{12}	. . .	e_{1n}
E_2	e_{21}	e_{22}	. . .	e_{2n}
. . .	. . . . .			
E_m	e_{m1}	e_{m2}	. . .	e_{mn}

Чтобы придти к однозначному, и, по возможности, наивыгоднейшему варианту решения, необходимо ввести оценочную (целевую) функцию. При этом матрица решений $\| e_{ij} \|$

сводится к одному столбцу. Каждому варианту E_i приписывается, таким образом, некоторый результат e_{ir}, характеризующий, в целом, все последствия этого решения. Такой результат мы будем в дальнейшем обозначать тем же символом e_{ir}.

Список использованной литературы

1. Карнадская, Н.Л. Принятие управленческого решения [Текст] / Карнадская Н.Л. – М.: ЮНИТИ, 1999.-265с.
2. Фатхутдинов, Р.А. Управленческие решения [Текст] / - М.: ИНФРА-М, 2001.-324с.

2. Лабораторная работа №1. Задачи теории игр

2.1. Постановка общей задачи теории игр

Теория игр является математической теорией конфликтных ситуаций. Примерами конфликтных ситуаций могут служить спортивные встречи, боевые операции и другие явления, в которых просматриваются взаимоотношения определенных сторон. Простейшие модели конфликтных ситуаций – это спортивные игры.

В игре могут сталкиваться интересы двух противников (игра парная или игра двух лиц), интересы n ($n > 2$) противников (игра множественная или игра n лиц). Существуют игры с бесконечным множеством игроков.

В каждой игре участники, называемые игроками, придерживаются определенной стратегии. **Стратегией** игрока называется система правил, однозначно определяющих выбор поведения игрока при каждом ходе, в зависимости от ситуации, сложившейся в процессе игры. В зависимости от числа возможных стратегий, игры делятся на конечные и бесконечные.

Процесс игры состоит в выборе каждым игроком i одной своей стратегии $s_i \in S_i$. В результате сложившейся ситуации s игрок i получает выигрыш $H_i(s)$.

Игры, в которых целью каждого участника является получение, по возможности, большего индивидуального выигрыша, называются **бескоалиционными.**

В **коалиционных** играх действия игроков направлены на максимизацию выигрышей коллективов (коалиции) без дальнейшего разделения выигрыша между участниками.

Бескоалиционной игрой называется система

$$\Gamma =< I,\{S_i\}_{i\in I},\{H_i\}_{i\in I} >,$$

в которой I и $s_i (i \in I)$ – множества, H_i – функции на множестве $S = \prod_{i\in I} S_i$, принимающие вещественные значения.

Бескоалиционная игра называется игрой с **постоянной суммой**, если существует такое постоянное C, что $\sum_{i\in I} H_i(s) = C$ для всех ситуаций $s \in S$.

В любой игре игроки действуют согласно возникшей ситуаций. Ситуация s, в игре, называется *приемлемой* для игрока i, если этот игрок, изменяя в этой ситуации свою стратегию s_i, на какую-либо другую, например, s_i', не может увеличить свой выигрыш.

Ситуация s, приемлемая для всех игроков, называется *ситуацией равновесия.*

Процесс нахождения ситуации равновесия в бескоалиционной игре есть процесс решения игры.

Чистая стратегия – это гарантированный выигрыш игрока, применяющего свою *i*-ю стратегию, тогда:

$$\alpha_{ij} = \min_{1 \le j \le n} a_{ij}.$$

Число $\underline{v} = \max_{1 \le i \le m} \alpha_i = \max_{1 \le i \le m} \{\min_{1 \le j \le n} \alpha_{ij}\}$ называется нижним значением игры, а соответствующая чистая стратегия i_0, при которой достигается это нижнее значение $\underline{v}$, называется *максиминной* стратегией первого игрока.

Аналогично, $\bar{v} = \min_{1 \le j \le n} \{\max_{1 \le i \le m} \alpha_{ij}\}$ называется верхним значением игры, а j_0 – *минимаксной* стратегией второго игрока.

Всегда $\underline{v} \le \bar{v}$. Если $\underline{v} = \bar{v} = v$ то игра имеет *седловую точку* в чистых стратегиях; число v называется значением игры (или *ценой игры*).

Игра имеет седловую точку в чистых стратегиях тогда и только тогда, когда существует элемент матрицы $a_{i_0 j_0}$, минимальный в своей строке, и, в то же время, максимальный в столбце

$$a_{ij_0} \le a_{i_0 j_0} \le a_{i_0 j} \qquad (1)$$

Любая пара (i_0, j_0), обладающая свойством (1), называется *седловой точкой*.

2.2. Смешанные стратегии

Если обозначить через $x_1, x_2, \ldots, x_m$ вероятности (частоты), с которыми первый игрок выбирает соответственно первую, вторую, . . ., m-ю чистую стратегии, так что выполняется условие $x_i \geq 0, i = 1,2,\ldots,m, \sum_{i=1}^{m} x_i = 1$, а через $y_1, y_2, \ldots, y_n$ вероятности, с которыми второй игрок выбирает первую, вторую,…, n-ю чистые стратегии, причем $y_i \geq 0, i = 1,2,\ldots,n, \sum_{i=1}^{n} y_i = 1$, то наборы чисел $x = (x_1, x_2, \ldots, x_m)$ и $y = (y_1, y_2, \ldots, y_n)$ называются **смешанными стратегиями** первого и второго игроков соответственно.

Каждый игрок имеет бесчисленное множество смешанных стратегий. Множество смешанных стратегий первого игрока обозначим через s_1 и множество смешанных стратегий второго игрока – через s_2.

Задача первого игрока состоит в выборе такой стратеги $x^* \in s_1$, чтобы при отсутствии информации о выборе другого, максимизировать свой выигрыш. Задача второго игрока состоит в выборе такой стратегии $y^* \in s_2$, чтобы при отсутствии информации о поведении первого игрока, минимизировать выигрыш первого.

Если первый игрок применяет стратегию $x \in S_1$, а второй – стратегию $y \in S_2$, то средний выигрыш $M(x, y)$ первого игрока равен

$$M(x,y) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} x_i y_j$$

Выигрыш $M(x,y)$ называют функцией игры.

Например, в задаче с матрицей $A = \begin{pmatrix} 2 & 3 & -1 & 4 \\ 5 & -4 & 3 & 5 \end{pmatrix}$

первый игрок имеет две чистые стратегии $x^{(1)} = (1,0)$, $x^{(2)} = (0,1)$ и бесчисленное множество смешанных стратегий, таких, как $x^{(3)} = (1/2, 1/2)$, $x^{(4)} = (1/17, 16/17)$ $x^{(5)} = (12/25, 13/25)$ и т. д. все они являются элементами множества $S_1 = \{x : x = (x_1, x_2), x_1 \geq 0, x_2 \geq 0, x_1 + x_2 = 1\}$ второй игрок имеет четыре чистые стратегии $y^{(1)} = (1,0,0,0), y^{(2)} = (0,1,0,0), y^{(2)} = (0,0,1,0), y^{(4)} = (0,0,0,1)$ и бесчисленное множество смешанных стратегий, таких, как $y^{(5)} = (1/4, 0, 1/4, 1/2)$, $y^{(6)} = (1/3, 1/5, 1/4, 1/4)$, $y^{(7)} = (1/20, 1/5, 1/2, 1/4)$, являющихся элементами следующего множества:

$$S_2 = \{y : y = (y_1, y_2, y_3, y_4), y_1 \geq 0, y_2 \geq 0, y_3 \geq 0, y_4 \geq 0, y_1 + y_2 + y_3 + y_4 = 1\}$$

Если первый игрок применяет смешанную стратегию $x^{(0)} = (1/6, 5/6)$, а второй - применяет стратегию $y^{(0)} = (1/3, 0, 1/3, 1/3)$, то средний выигрыш первого игрока, определяемый функцией

игры, окажется равным $M(x^{(0)}, y^{(0)}) = \sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij} x_i^{(0)} y_j^{(0)} = 10/9$

Если же первый игрок применяет стратегию $x^{(0)} = (1/6, 5/6)$, а второй — стратегию $y^{(1)} = (1/3, 0, 1/2, 1/6)$, то $M(x^{(0)}, y^{(1)}) = 25/12$. Оптимальная стратегия первого игрока $x^* = (4/11, 7/11)$, а второго - $y^* = (0, 4/11, 7/11, 0)$; цена игры равна $5/11$.

2.3. Седловая точка в смешанных стратегиях

Пара смешанных стратегий (x^*, y^*) называется *седловой точкой* функции $M(x, y)$, если

$$M(x, y^*) \le M(x^*, y^*) \le M(x^*, y) \qquad (2)$$

Каждая матричная игра с нулевой суммой имеет решение в смешанных стратегиях, т.е. существуют такие смешанные стратегии x^* и y^* первого и второго игроков соответственно, что выполняется условие (2). Гарантированный выигрыш первого игрока, применяющего смешанную стратегию

$$v_1(x) = \min_{y \in S_2} M(x, y)$$

$$s_2 = \{y : y = (y_1, y_2, ..., y_n), y_j \ge 0, \sum_{j=1}^{n} y_j = 1\}$$

Стратегия x^*, при которой гарантированный выигрыш первого игрока достигает максимального значения, называется *оптимальной стратегией* первого игрока:

$$v_1(x^*) = \min_{x \in s_1} v_1(x) = \max_{x \in s_1} M(x, y) \min_{y \in s_2} M(x, y)$$

$$s_1 = \{x : x = (x_1, x_2, ..., x_m),\ x_i \geq 0,\ \sum_{i=1}^{m} y_i = 1\}$$

Гарантированный проигрыш второго игрока

$$u_2(y) = \max_{x \in s_1} M(x, y)$$

y* является оптимальной стратегией второго игрока, если

$$u_2(y^*) = \min_{y \in s_2} u_2(y) = \min_{y \in s_2} \max_{x \in s_1} M(x, y)$$

Гарантированный выигрыш первого игрока, применяющего свою оптимальную стратегию, равен гарантированному проигрышу второго игрока, применяющего свою оптимальную стратегию: $v_1(x^*) = u_2(y^*) = v^*$, v^* – цена игры.

2.4. Сведение задачи теория игр к задаче линейного программирования

Задача максимизации гарантированного выигрыша первого игрока и задача минимизации гарантированного проигрыша второго игрока сводятся к паре взаимно двойственных задач линейного программирования:

Задача первого игрока	Задача второго игрока
$F = v(\max),$ $\sum_{i=1}^{m} a_{ij} x_i \geq v,\ j = 1,2,...,n$ $\sum_{i=1}^{m} x_i = 1$ $x_i \geq 0, i = 1,2,...,m$	$F = u(\max),$ $\sum_{j=1}^{n} a_{ij} y_j \geq u,\ j = 1,2,...,m$ $\sum_{j=1}^{n} y_j = 1$ $y_j \geq 0,\ j = 1,2,...,n$

Процесс решения задач упрощается, если перейти к переменным $\xi_i = \frac{x_i}{v}(i = 1,2,...,m),\ \ \eta_j = \frac{y_j}{u}(j = 1,2,...,n)$. Это возможно, если выполняется условие $a_{ij} \geq 0$. Тогда имеем:

Задача первого игрока	Задача второго игрока
$f = \sum_{i=1}^{m} \xi_i (\min),$ $\sum_{i=1}^{m} a_{ij} \xi_i \geq 1,\ j = 1,2,...,n$ $\xi_i \geq 0, i = 1,2,...,m$	$f = \sum_{j=1}^{n} \eta_j (\max),$ $\sum_{j=1}^{n} a_{ij} \eta_j \geq 1, i = 1,2,...,m$ $\eta_j \geq 0,\ j = 1,2,...,n$

Оптимальные стратегии игроков не изменятся, если матрицу игры $A=\|a_{ij}\|$ заменить на $\tilde{A}=\|a_{ij}+c\|$. Цена игры при этом увеличивается на c.

2.5. Методы решения задач теории игр

Методы решения задач теории игр во многом зависят от условий задачи и от матрицы A выигрышей первого игрока.

1) Если матрица A имеет седловую точку, то решение игры сводится к нахождению седловой точки матрицы A. При этом, оптимальные стратегии игроков определяются координатами (i, j) седловой точки матрицы A, а цена игры элементом a_{ij} в седловой точке.

2.5.1.Пример 1. Найти оптимальные стратегии игроков и цену игры:

$$A=\begin{pmatrix} 2 & 10 & 3 & 14 & 5 \\ 8 & 9 & 5 & 6 & 7 \\ 10 & 8 & 4 & 8 & 12 \end{pmatrix}$$

Минимизируя элементы первой строки получим $\alpha_1=\min\{2,10,3,14,5\}=2$. Аналогично, $\alpha_2=\min\{8,9,5,6,7\}=5$, $\alpha_3=\min\{10,8,4,8,12\}=4$.

Максимизируя элементы по столбцам получим: $\beta_1=10,\ \beta_2=10,\ \beta_3=5,\ \beta_4=14,\ \beta_5=12$.

Нижняя цена игры $\underline{v}$ определяется путем максимизации α_i: $\underline{v} = \max\limits_i \alpha_i = \max\{2,5,4\} = 5$, а верхняя цена игры определяется минимизацией β_j: $\bar{v} = \min\limits_j \beta_j = \min\{10,10,5,14,12\} = 5$

Итак, имеем $\bar{v} = \underline{v}$, так, что $v = 5$. Игра, определяемая матрицей A, имеет седловую точку (2;3). Здесь задача разрешима в чистых стратегиях. Следовательно, оптимальный способ игры найден. Придерживаясь чистой второй стратегии, первый игрок обеспечивает себе выигрыш, не меньший 5 условных единиц, а второй игрок, применяя чистую третью стратегию, проигрывает не более 5 условных единиц. Обе стратегии i=2 и j=3 являются оптимальными для первого и второго игроков. Цена игры v=5.

2) Если матрица A имеет размерность $m \times 2$ или $2 \times n$, то решение задачи может быть получено графически.

Иногда процесс решения задачи можно упростить, если вычеркнуть из матрицы *доминирующие* (неполезные, заведомо невыгодные) стратегии. Вычеркнутым стратегиям соответствуют нулевые компоненты в смешанных стратегиях. Стратегия i_0 для первого игрока является *доминирующей*, если существует такая стратегия i_1 для первого игрока, что

$$a_{i_1 j} \geq a_{i_0 j},\ j = 1,2,\ldots,n$$

или если существуют такие числа $\mu_i \geq 0, (i \neq i_0),\ \sum_{i \neq i_0} \mu_i = 1$, что

$$\sum_{i \neq i_0} \mu_i a_{ij} \geq a_{i_0 j},\ j = 1,2,...,n$$

Стратегия j_0 для второго игрока является *доминирующей,* если существует такая стратегия j_1 второго игрока, что

$$a_{ij_1} \geq a_{ij_0},\ i = 1,2,...,m$$

или если существуют такие числа $\nu_j \geq 0, (j \neq j_0), \sum_{j \neq j_0} \nu_j = 1$, что

$$\sum_{j \neq j_0} \nu_j a_{ij} \geq a_{ij_0},\ i = 1,2,...,m$$

Пример 2. Найти оптимальные стратегии игроков и цену игры

$$A = \begin{pmatrix} 3 & 7 & 2 & 1 & 5 & 8 \\ 4 & 9 & 3 & 6 & 2 & 1 \\ 2 & 3 & 1 & 4 & 9 & 20 \end{pmatrix}$$

В чистых стратегиях решения игры нет, так как

$$\underline{\nu} = \max_i \min_j a_{ij} = 1,$$
$$\overline{\nu} = \min_i \max_j a_{ij} = 3,$$
$$\underline{\nu} < \overline{\nu}.$$

Упростим матрицу A, заметив, что

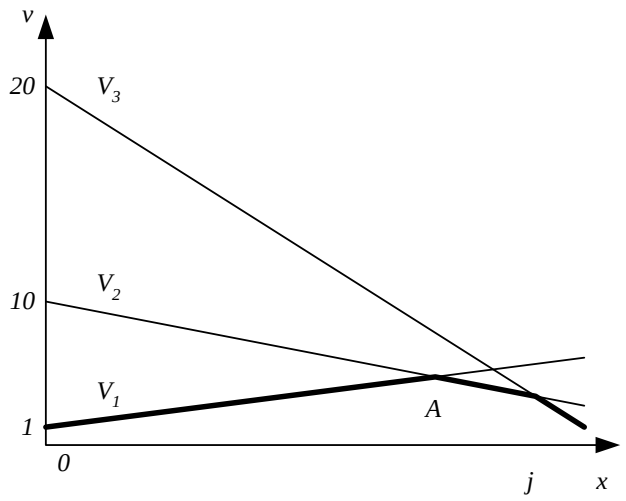
$$a_{i2} \geq a_{i1},\ i = 1,2,3,$$
$$a_{i1} \geq a_{i3},\ i = 1,2,3,$$
$$a_{1j} \geq \frac{1}{2} a_{2j} + \frac{1}{2} a_{3j},\ j = 3,4,5.$$
$$a_{i4} \geq a_{i3},\ i = 2,3.$$

Задачу, заданную матрицей $\overline{A}$ размерности 2x3, решим геометрически:

$$\overline{A} = \begin{pmatrix} 3 & 2 & 1 \\ 1 & 9 & 20 \end{pmatrix}$$

В плоскости переменных (x, v) построим – $v_j(x)$ ожидаемый средний выигрыш первого (рис.1) игрока, применяющего первую стратегию с вероятностью x при условии, что второй игрок отвечает чистой стратегией j, (j=1,2,3):

$$v_1(x) = 3x + 1(1-x) = 2x + 1$$
$$v_2(x) = 2x + 9(1-x) = -7x + 9,\ 0 \le x \le 1$$
$$v_3(x) = x + 20(1-x) = -19x + 20$$



В точке A (рис.1) пересечения прямых $v_1(x)$ и $v_2(x)$ гарантированный выигрыш первого игрока (изображенный жирной линией) достигает наибольшего значения

$$2x + 1 = -7x + 9,$$
$$9x = 8,$$
$$x = 8/9;$$
$$\overline{x^*} = (8/9, 1/9);$$
$$\overline{y^*} = (7/9, 2/9), \overline{v^*} = 25/9$$

Исходная задача с матрицей A имеет следующее решение:

$$v^* = 25/9; x^* = (0, 8/9, 1/9);$$

$$y^* = (0,0, 7/9, 0, 2/9, 0)$$

3) Если задача теории игр не имеет решения в чистых стратегиях и не может быть решена графически, то для получения точного решения игры используют методы линейного программирования.

Для решения задачи второго игрока более удобно воспользоваться симплекс-методом. В последней симплекс-таблице, содержащей оптимальное решение задачи второго игрока, можно найти и оптимальное решение двойственной задачи – задачи первого игрока.

3. **Лабораторная работа №2. Решение матричных игр**

3.1. Цель работы и методические указания к ее выполнению

Матричная игра представляет собой антагонистическую игру двух лиц с платежной матрицей $A_{m\times n}$, с m возможными стратегиями первого игрока и n стратегиями второго игрока. Первый игрок стремится максимизировать свой выигрыш, второй – минимизировать свой проигрыш. Если матрица игры имеет седловую точку, т.е. существует элемент, являющийся максимальным в столбце и минимальным в строке, то игра имеет решение в чистых стратегиях. В противном случае игра имеет решение в смешанных стратегиях, которые представляют собой вероятностные распределения на множестве чистых стратегий:

$\bar{x} = (x_1,...,x_m)^T$, – смешанная стратегия первого игрока, $\sum_{i=1}^{m} x_i = 1$;

$\bar{y} = (y_1,...,y_n)^T$, – смешанная стратегия второго игрока, $\sum_{i=1}^{n} y_i = 1$.

Использование в игре оптимальных смешанных стратегий обеспечивает первому игроку выигрыш, не меньший, чем при использовании им любой другой стратегии; второму игроку – проигрыш не больший, чем при использовании им любой другой стратегии.

С точки зрения 1-го игрока игру можно записать как задачу линейного программирования

$$V \to \max;$$

$$\sum_{i=1}^{m} x_i a_{ij} \geq V, \ j = 1,...,n;$$

$$\sum_{i=1}^{m} x_i = 1, x_i \geq 0; i = 1,...,m;$$

где значение игры V рассматривается как $m+1$-я переменная, неограниченная по знаку (свободная).

С точки зрения 2-го игрока игру задача имеет вид

$$V \to \min;$$

$$\sum_{j=1}^{n} a_{ij} y_j \leq V, \ i = 1,..., m;$$

$$\sum_{i=j}^{n} y_j = 1, \ y_j \geq 0; \ j = 1,..., n.$$

Причем, обе эти задачи представляют собой пару двойственных задач линейного программирования, и решив одну из них, можем найти смешанные стратегии обоих игроков.

Предлагается следующий алгоритм:

Шаг 1. В платежной матрице $A = \|a_{ij}\|$ находятся нижняя цена α и верхняя цена β игры. Если они равны между собой, то игра имеет седловую точку, и, следовательно, существует решение игры в чистых стратегиях. Если игра не имеет седловой точки, то переход к шагу 2,

Шаг 2. Игра решается в смешанных стратегиях. Если в матрице игры имеются отрицательные элементы, то ко всем

элементам матрицы *A* добавляется некоторое число μ для того, чтобы все элементы платежной матрицы стали положительными.

Шаг 3. Для преобразованной матрицы *A* формируются две задачи линейного программирования.

Шаг 4. Одним из методов линейного программирования производится решение полученных задач.

Шаг 5. По найденным решениям $x=(x_1,x_2,...,x_n)$ и $y=(y_1,y_2,...,y_n)$ задач линейного программирования определяются цена игры ν и вероятности p_i и q_j $i=\overline{1,m}$, $j=\overline{1,n}$.

Если к исходной платежной матрице добавлялось число μ, то вычисляется истинная цена игры, которая равна полученной цене игры минус добавляемое число μ.

Порядок выполнения работы

1. Дать формализованную постановку задачи**.**
2. Определить множество возможных стратегий игроков, при этом по возможности исключить эквивалентные стратегии.
3. Выписать матрицу игры.
4. Найти оптимальные стратегии игроков.

3.2. Постановка задачи и последовательность ее выполнения

Постановка задачи. Необходимо построить платежную матрицу для игры «Три пальца», которая характеризуется следующими правилами: два игрока одновременно выбрасывают

от одного до трех пальцев. В случае, если сумма выброшенных пальцев четная, то выигрывает игрок *A,* и его выигрыш количественно равен сумме выброшенных пальцев. В случае нечетной суммы - выигрывает игрок *B.*

Анализ правил позволяет установить, что для игрока *A* определены три стратегии: A_1 - выбрасывать один палец; A_2 - выбрасывать два пальца; *Аз* - выбрасывать три пальца. Аналогичные стратегии B_1, B_2 и B_3 формируются и для игрока *B.* Платежная матрица приведена в табл.1.

Таблица 1. Платежная матрица для игры «Три пальца»

Стратегия игрока *A*	Стратегия игрока *B*		
	B_1	B_2	B_3
A_1	2	-3	4
A_2	-3	4	-5
Аз	4	-5	6

Проигрыш игрока - это число, указанное со знаком минус,

Пример. Необходимо найти решение задачи «Три пальца», в которой для игрока *A* возможны три стратегии $A = \{A_1, A_2, A_3\}$ и для игрока *B* также выделены три стратегии $B = \{B_1, B_2, B_3\}$.

Шаг 1. Определяются нижняя цена игры $\alpha = -3$ и верхняя цена игры $\beta = 4$. Так как, $\alpha \neq \beta$, то седловая точка отсутствует, и рассматриваемая игра не имеет решения в чистых стратегиях.

Шаг 2. Производится решение игры в чистых стратегиях. Для этого в платежной матрице находится максимальный по модулю отрицательный элемент. Тогда, добавляемая константа $\mu = 5$ прибавляется ко всем элементам исходной платежной матрицы.

В результате формируется преобразованная платежная матрица (табл. 2).

Преобразованная платежная матрица Таблица 2.

Стратегия игрока *A*	Стратегия игрока *B*		
	B_1	B_2	B_3
A_1	7	2	9
A_2	2	9	0
A_3	9	0	11

Шаг 3. Для преобразованной платежной матрицы формируется задача линейного программирования для игрока:

$$w = x_1 + x_2 + x_3 \to \min,$$
$$7x_1 + 2x_2 + 9x_3 \geq 1,$$
$$2x_1 + 9x_2 \geq 1,$$
$$9x_1 + 11x_3 \geq 1,$$

$$x_1, x_2, x_3 \geq 0,\ x_1 = p_1 / \nu.$$

Шаг 4. Решение задачи линейного программирования позволяет найти оптимальные значения переменных $x_i,\ i = \overline{1,3}$

$$x_1 = 1/20,\ x_2 = 1/10,\ x_3 = 1/20$$

Шаг 5. По найденным значениям $x_i,\ i = \overline{1,3}$ определяется цена игры $\nu_1 = \dfrac{1}{1/20 + 1/10 + 1/20} = 5$

и вычисляются вероятности применения стратегий A_1, A_2, A_3 игроком A:

$$p_1 = 5\frac{1}{20} = \frac{1}{4},\quad p_2 = 5\frac{1}{10} = \frac{1}{2},\ p_3 = 5\frac{1}{20} = \frac{1}{4}$$

Аналогично формируется задача линейного программирования для нахождения вероятности применения игроком B стратегий B_1, B_2 и B_3;

$$\begin{aligned} L = y_1 + y_2 + y_3 &\to max, \\ 7x_1 + 2x_2 + 9x_3 &\leq 1, \\ 2x_1 + 9x_2 &\leq 1, \\ 9x_1 + 11x_3 &\leq 1, \\ y_1, y_2, y_3 &\geq 0. \end{aligned}$$

Решение задачи позволяет найти оптимальные значения вероятности $q_i,\ i = \overline{1,3}$, применения игроком B стратегии B_j.

$$y_1 = \frac{1}{20},\ \ y_2 = \frac{1}{10},\ \ y_3 = \frac{1}{20},$$

$$q_1 = \frac{1}{4}, \quad q_2 = \frac{1}{2}, \quad q_3 = \frac{1}{4},$$

$$\nu = \mu - 5$$

4. Лабораторная работа №3. Задача теории расписаний с двумя обслуживающими устройствами.

4.1. Постановка задачи и алгоритм ее решения

Имеются два станка *A* и *B,* а также даны *n* деталей. Для каждой *i*-й детали определено время обработки на станках *A* и *B:* $a_i,\ b_i, i=\overline{1,n}$. Все детали обрабатываются по единому технологическому маршруту: сначала на станке *A,* затем на станке *B.* Согласно алгоритма (алгоритм Джонсона) каждый станок одновременно должен обрабатывать только одну деталь, и каждая деталь в конкретный момент времени должна обрабатываться только одним станком. Требуется определить такую последовательность запуска деталей на обработку, при которой длительность производственного цикла была бы минимальной (общее время простоя обоих механизмов было бы минимальным).

Определение: для последовательности $\sigma_n^{'}=(i_1,...,i_n)$, длительностью производственного цикла называется промежуток времени между началом обработки первой по порядку детали i_n на станке *A* и окончанием обработки последней по порядку детали i_n на станке *B.*

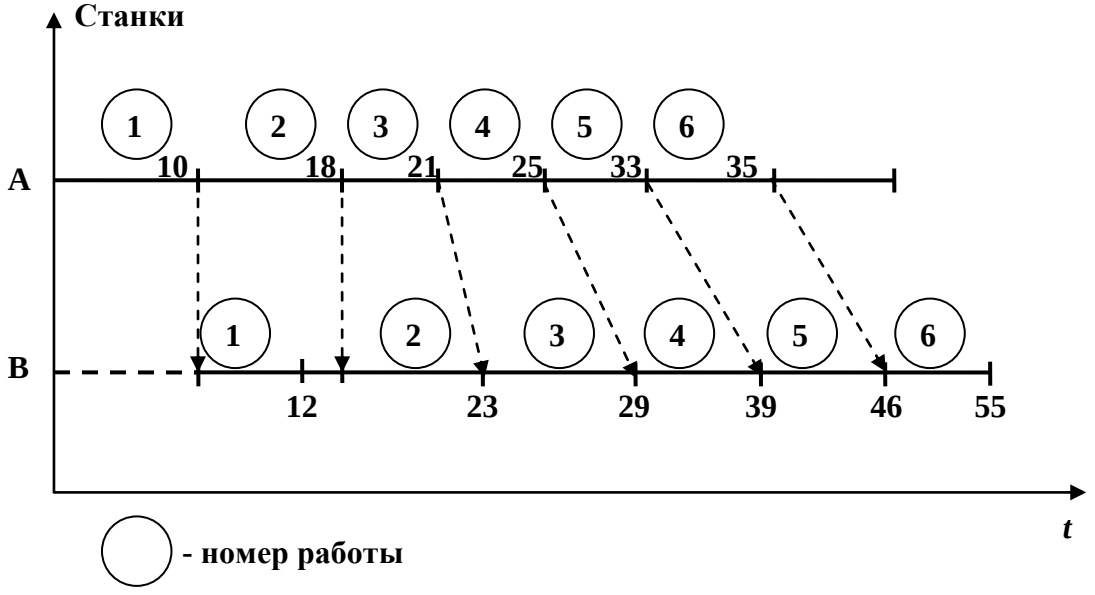
4.2 Пример.

Пусть, например, даны шесть деталей, т.е. $n = 6$ и известны времена их обработки на станках, представленные в табл.1.

Таблица 1. Исходные данные для задачи двух станков

Время обработки	Номера деталей					
	1	2	3	4	5	6
A	10	8	3	4	8	2
B	2	5	6	10	7	9

Предположим, что выбрана следующая последовательность запуска деталей па обработку: $\sigma_6^1 = (1,2,3,4,5,6)$. На рис.1 представлен график обработки деталей для последовательности σ_6^1



На рис 1. представлен график обработки деталей для неоптимальной последовательности

Решающее правило предполагает выполнение следующих шагов:

Шаг 1. Формируется множество номеров всех деталей: $I = \{1,2,...,i,...,n\}$

Шаг 2. Определяется множество *J* деталей, для которых время обработки на станке *A* меньше либо равно времени обработки на станке *B:*

$$J = \{ j / j \in I \text{ and } a_j \le b_j \}$$

Шаг 3, Определяется множество оставшихся деталей

$Q = l \setminus J$.

Шаг 4. Формируется кортеж 7* из номеров деталей, принадлежащих множеству J_n упорядоченных в порядке возрастания времени a_j :

$$J^* = (j_1, j_2, ..., j_k);\ a_{j_1} \le a_{j_2} \le ... \le a_{j_k}.$$

Шаг 5. Формируется кортеж $Q^* = (q_1, ..., q_z, ..., q_l)$ из элементов множества *Q* таких, что $b_{q_1} \ge b_{q_2} \ge ... \ge b_{q_l}$

Шаг б. Тогда оптимальная последовательность запуска деталей на обработку: $\sigma_n^0 = (j_1, j_2, ..., j_k, q_1, q_2, ..., q_l)$

Рассмотрим применение решающего правила для анализируемого численного примера.

Шаг 1. I={1,2,3,4,5,6}. Шаг 2. J={3,4,6}

Шаг 3. Q={1,2,5}. Шаг 4. J^*={6,3,4,}

Шаг 5. Q^*={5,2,1}. Шаг 6. σ^0_{n-6} *={6,3,4,5,2,1}*

На рис. 2 представлен график оптимальной последовательности обработки деталей.

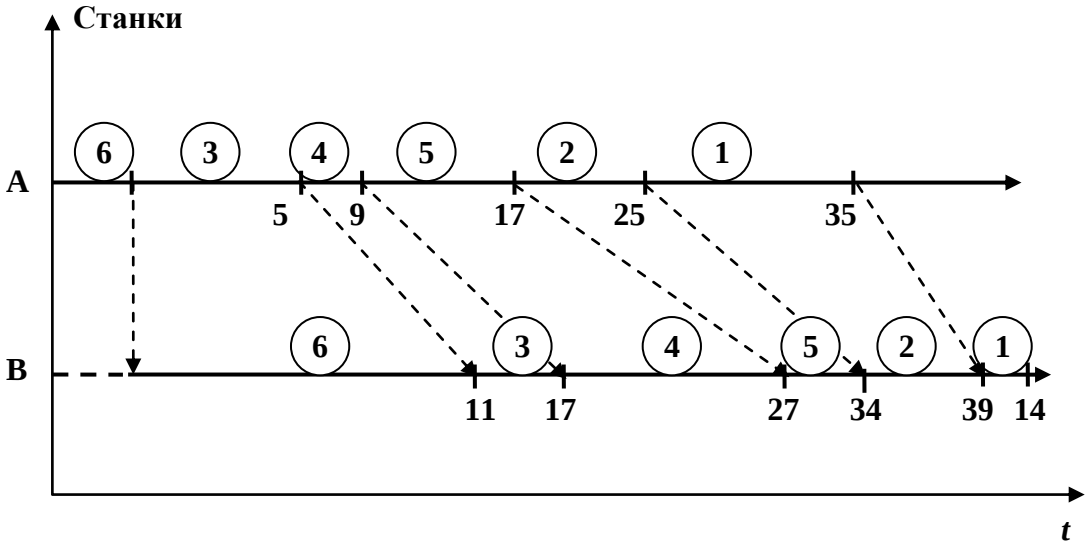


Рис.2

Для оптимальной последовательности обработки деталей длина производственного цикла равна сорок одной единице времени, и, она существенно меньше длины для неоптимальной последовательности, составляющей пятьдесят пять единиц времени.

5. Лабораторная работа №4. Решение задачи о кратчайшем маршруте. Метод Форда.

5.1. Постановка сетевой транспортной задачи.

Задача определения кратчайшего маршрута является одной из приоритетных задач транспортной системы. Это обусловлено тем, что определение кратчайшего пути, по маршруту следования приводит к желаемым результатам. Транспортная сеть может быть представлена в виде графа, представленного на рис.1, дуги которого - транспортные магистрали,

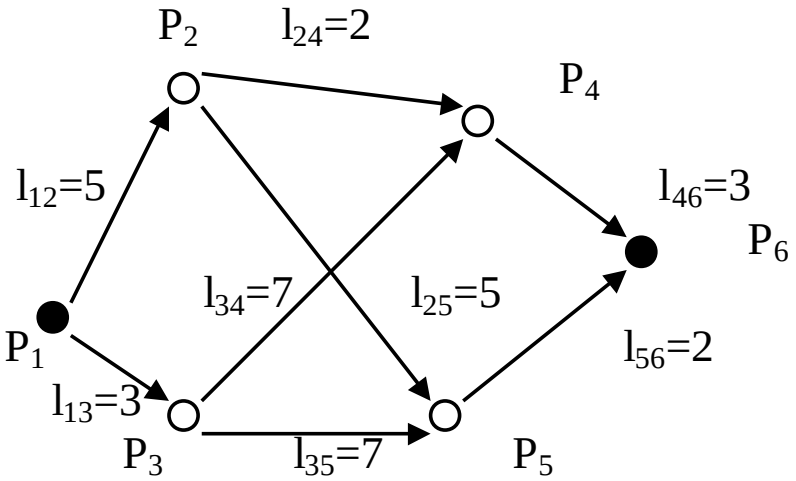


Рис.1

а узлы - пункты отправления и назначения. Графически транспортная сеть изображается в виде совокупности *n* пунктов $P_1,P_2,...,P_n$, причем некоторые упорядоченные пары (P_i,P_j) пунктов назначения соединены дугами, заданной

длинны $\rho(P_i,P_j)=l_{ij}$. Некоторые, или все дуги, могут быть ориентированы, т.е. по ним возможно движение только в указанном стрелками направлении.

На рис.1 построена ориентированная транспортная сеть, содержащая шесть пунктов $P_1,P_2,...,P_6$, которые связаны между собой восьмью транспортными путями.

Необходимо определить кратчайший маршрут из пункта P_1 в P_6. Определение кратчайшего пути состоит в указании последовательности прохождения маршрута через промежуточные пункты и суммарной длины маршрута.

Например, маршрут из пункта P_1 в пункт P_6: $P_1P_2P_4P_6$; $L=l_{12}+l_{24}+l_{46}=10$.

Постановка задачи приобретает смысл в том случае, если имеется несколько вариантов маршрута из начального пункта в конечный. В этом случае, физический смысл функции цели задачи, состоит в минимизации общей длинны маршрута, т.е. в определении кратчайшего пути из P_1 в P_n.

5.2. Описание метода и алгоритма решения.

Метод Форда является одним из эффективных методов решения сетевых транспортных задач, в основе которого заложен принцип оптимальности.

Алгоритм метода Форда содержит четыре этапа. На первом этапе производится заполнение исходной таблицы расстояний от любого *i*-го пункта в любой другой *j*-й пункт назначения. На втором этапе, определяются для каждого пункта некоторые параметры λ_i и λ_j по соответствующим формулам. Далее, на третьем этапе определяются кратчайшие расстояния. Наконец, на четвертом этапе, определяются кратчайшие маршруты из пункта отправления P_1 в любой другой пункт назначения P_j, $j=1,2,...,n$.

Остановимся подробнее на каждом из этих четырех этапов**.**

5.2.1 Первый этап: составление исходной таблицы расстояний.

Исходная таблица будет содержать $n+1$ строк и такое же количество столбцов; P_i - пункты отправления; P_j - пункты назначения. Во второй строке и втором столбце проставляются значения параметров λ_i и λ_j, определение значений которых производится на втором этапе решения задачи. В остальных клетках таблицы проставляются значения расстояний l_{ij} из *i*-го пункта в *j*-й пункт. Причем заполняются клетки таблицы, лежащие выше главной диагонали. Если пункт P_i не соединен отрезком пути с

пунктом P_j, то соответствующая клетка таблицы не заполняется.

5.2.2. Второй этап: определение λ_i и λ_j.

Значения параметров λ_i и λ_j определяются в соответствии с формулой:

$$\lambda_j = \min(\lambda_1 + l_{ij}); \quad i = 1,2,...,n; \quad j = 1,2,...,n, \qquad (1)$$

где $\lambda_i = 0$.

Эти значения заполняются во второй строке и во втором столбце таблицы.

5.2.3. Третий этап: определение длины кратчайших путей.

Возможны два случая определения длины кратчайших путей из пунктов P_i в пункты P_j, $i=1,2,...,n$; $j=1,2,...,n$.

В первом случае, если выполняется неравенство:

$$\lambda_j - \lambda_i \le l_{ij}; \ l_{ij} \ne 0; \ j=1,2,...,n; \ j=1,2,...,n, \qquad (2)$$

то значения параметров $\lambda_1,...,\lambda_n$ удовлетворяют условиям оптимальности. Каждое значение λ_j есть не что иное, как кратчайшее расстояние от пункта P_i до пункта P_j, $j=2,3,...,n$.

Во втором случае, если для некоторых клеток (i,j) таблицы имеет место неравенство:

$$\lambda_j - \lambda_i \succ l_{ij}; \ i = \overline{1,n}; \quad j = \overline{1,n}, \qquad (3)$$

то значения λ_j и λ_i могут быть уменьшены.

Если справедливо (3), тогда исправим значение λ_{j0}, пересчитав его по формуле:

$$\lambda'_{j0} = \lambda_{i0} + l_{i0\,j0}. \qquad (4)$$

5.2.4. Четвертый этап: нахождение кратчайшего пути.

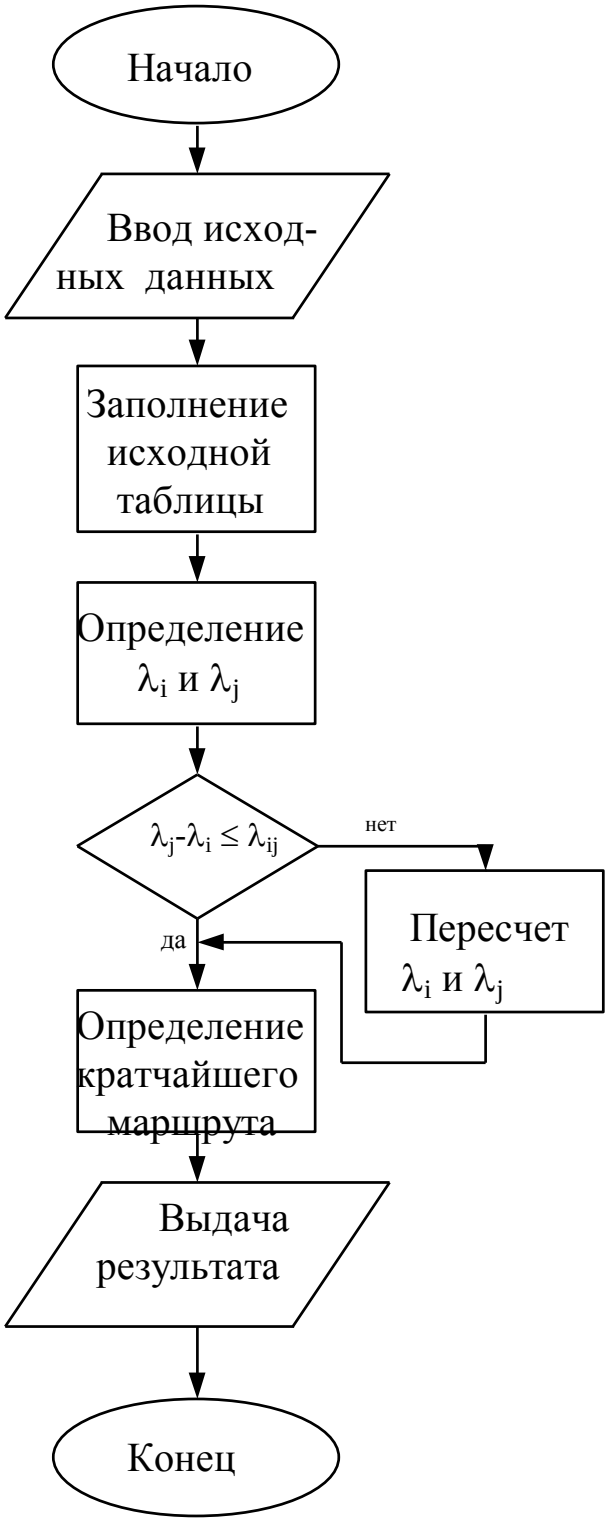
На этом этапе определяются последовательности пунктов кратчайшего маршрута. С этой целью для каждого столбца определяют величину:

$$l_{r1,j} = \lambda_j - \lambda_{r1}, \qquad (5)$$

где $l_{r1,j}$ берется из таблицы, причем λ_{r1} выбирается так, чтобы выполнилось равенство (5). Таким образом определим *r1*. Далее, продолжим ту же операцию. Однако, последней будем считать не P_n, а P_{r1}. Будем продолжать до тех пор, пока $r_n=1$.

Таким образом кратчайший маршрут проходит через $P_{r1},P_{r2},\ldots,P_{rn}$, а длина маршрута $L_{\min} = l_{r2,r1} + l_{r3,,r2} + \ldots + l_{m-1,m}$.

5.2.5. Блок-схема алгоритма расчёта



5.2.6. Информационное пояснение программы

Программа “FORD” написана на языке высокого уровня - Pascal, в интегрированной среде разработки “Turbo Pascal 7.0” фирмы Borland Inc.

Программа предназначена для нахождения кратчайшего пути в сетевом графе по методу Форда. Программа удобна в использовании, что достигается за счет использования дружественного интерфейса и иерархического меню. Вначале программы производится ввод данных, затем нахождение кратчайшего маршрута и вычисление его длинны, далее выводится результат. Вывод результатов возможен как в файл, так и на экран.

В программе предусмотрена возможность повторного решения задачи с другими исходными данными.

5.2.6 Описание подпрограмм и процедур.

Подпрограммы и функции.

ТИП	НАЗВАНИЕ	НАЗНАЧЕНИЕ
Function type : real	*min*;	Вычисляет минимальное значение вектора *k[i]*;
Procedure	*set_graph_mode*;	Устанавливает графический режим;
Procedure	*install_firewall*;	Инициализирует огонь;
Procedure	*fire*;	Процедура рисования огня;
Procedure	*ok*;	Выводит сообщение о корректности операции;
Procedure	*notok*;	Выводит сообщение о некорректности операции;
Procedure	*check_input_data*;	Проверяет корректность ввода данных;
Proced	*keybord_input*;	Ввод исходных данных с

ure		клавиатуры;
Proced *ure*	*ramka*;	Выводит рамку по краям экрана;
Proced *ure*	*save*;	Сохранение результатов в файл;
Proced *ure*	*about_progra m;*	Выводит информацию о программе;
Proced *ure*	*about_method*;	Выводит информацию о методе Форда;
Proced *ure*	*output_graph*;	Рисует вершины графа;
Proced *ure*	*draw_ways*;	Рисует дуги графа;
Proced *ure*	*draw_short_w ay;*	Рисует кратчайший маршрут;
Proced *ure*	*count_point_c oord;*	Вычисляет экранные координаты вершин графа;
Proced *ure*	*set_font*;	Инициализирует шрифт пользователя;
Proced *ure*	*calculate*;	Основное математическое ядро программы;
Proced *ure*	*draw_menu*;	Открытие меню;

Proced*ure*	*redraw_menu;*	Закрытие меню;
Proced*ure*	*main_menu;*	Основной механизм меню;
Proced*ure*	*pixel;*	Ставит точку;
Proced*ure*	*stars;*	Инициализирует массив со звездами;
Proced*ure*	*welcomescreen;*	Заставка;

Таблица идентификаторов.

ИМЯ	ТИП	НАЗНАЧЕНИЕ
Константы		
menu	*array of string*	Описывает меню программы
menuof	*array of byte*	Описывает меню программы
menugo	*array of byte*	Описывает меню программы

name1	*string*	Имя файла входных данных
name2	*string*	Имя файла выходных данных
xxx	*word*	Размер огня по x
yyy	*word*	Размер огня по y
xx1	*word*	Координата x огня
yy1	*word*	Координата y огня
messize	*byte*	Размер заглавия
title	*array of string*	Заглавие
Переменные		
mas	*array of real*	Основная матрица вычислений
coord_point	*array of real*	Координаты вершин графа
i	*integer*	Переменная для организации цикла
j	*integer*	Переменная для организации цикла
t	*integer*	Используется при расчете пути
m	*integer*	Счетчик количества вершин в кратчайшем пути

n	*integer*	Количество вершин в графе
z	*integer*	Код ошибки
x1	*integer*	Используется в процедуре вывода на экран
y1	*integer*	Используется в процедуре вывода на экран
x2	*integer*	Используется в процедуре вывода на экран
y2	*integer*	Используется в процедуре вывода на экран
kk	*integer*	Промежуточное значение
iii	*integer*	Промежуточное значение
x	*integer*	Координата *x* конца отрезка
y	*integer*	Координата *y* конца отрезка
lenth	*integer*	Количество вершин в кратчайшем маршруте
chrus	*integer*	Номер шрифта пользователя
z1	*integer*	Номер графического драйверв
z2	*integer*	Номер графического режима
k	*array of real*	Используется для нахождения минимума

result	*array of integer*	Номера вершин, которые входят в кратчайший маршрут
error_code	*array of byte*	Коды ошибок при вводе данных
fire1	*array of byte*	Хранит цвета огня
fire2	*array of byte*	Матрица промежуточных данных
aa	*real*	Используется при вычислении координат вершин графа
pi1	*real*	Используется при вычислении координат вершин графа
s	*real*	Хранит промежуточное значение
l	*boolean*	Используется. при определении кратчайшего маршрута
inputdata	*boolean*	*TRUE*, если данные вводились
calculated	*boolean*	*TRUE*, если данные били

ata		обработаны
mov	*boolean*	Используется в процедуре меню
o	*string*	Используется при вводе с клавиатуры
temp	*byte*	Хранит временное значение
cursor	*byte*	Координаты курсора меню
lastcursor	*byte*	Последние координаты курсора меню
menulevel	*byte*	Уровень меню
nline	*byte*	Количество строк в текушем уровне меню
pressed	*char*	Используется при вводе с клавиатуры
f1	*text*	Файловая переменная
f2	*text*	Файловая переменная

Примеры решения контрольных задач.

Исходная таблица расстояний для одного из вариантов ранжированного графа:

P_i/P_j	1	2	3	4	5	6
1	X	5	3			
2		X		2	5	
3			X	7	7	
4				X		3
5					X	2
6						X

После обработки таблицы с заданными исходными данными, программа выдает следующие результаты:

- кратчайший маршрут: 1-2-4-6

- длинна кратчайшего маршрута: 10

Исходная таблица расстояний для одного из вариантов не ранжированного графа:

P_i/P_j	1	2	3	4	5	6
1	X		1	6	2	
2		X				1
3		8	X			
4		2		X		5
5			1	3	X	9
6						X

После обработки таблицы с заданными исходными данными, программа выдает следующие результаты:

- кратчайший маршрут: 1-5-4-2-6

- длинна кратчайшего маршрута: 8

Программа работоспособна при любых других вариантах исходных данных.

5.2.7. Выводы.

Анализ алгоритма операций, необходимых при решении сетевой транспортной задачи методом Форда в заданной постановке подтверждает следующее:

Достижение конечного результата производится в четыре этапа.

Каждый этап описывается простыми математическими операциями и может быть записан на одном из языков программирования.

Составлена программа на алгоритмическом языке высокого уровня “Pascal”, позволяющая решать задачу в диалоговом режиме, удобном для пользователя не программиста.

Алгоритм решения транспортной задачи методом Форда является универсальным, что позволяет производить расчёты

как с ранжированными, так и с не ранжированными графами (примеры решения задачи приведены на странице 11).

Возможность реализаций для удобства работы пользователя в программе сервисной части.

Возможность неоднократного решения задачи методом Форда при различных исходных данных.

5.2.8 Текст программы расчёта

```
PROGRAM ford;
uses crt,graph;
const menu:array[0..4,1..6] of string =
 (('Ввод данных','Решение задачи','Вывод результата',
 'О методе','О программе','Выход'),
 ('Ввод данных','Просмотр данных','Назад','','',''),
 ('Экран','Файл','Назад','','',''),
 ('Клавиатура','Файл','Назад','','',''),
 ('Да','Нет','','','',''));
 menuof:array[0..4] of byte =(6,3,3,3,2);
 menugo:array[0..4,1..6] of byte = ((1,0,2,0,0,4),
(3,0,0,0,0,0), (0,0,0,0,0,0), (0,0,1,0,0,0), (0,0,0,0,0,0));
 name1='input.dat';
 name2='output.dat';
```

```
      xxx=140;
      yyy=20;
      xx1=10;
      yy1=140;
      messize=3;
      col:array[16..31] of
byte=(0,186,113,4,40,41,41,42,42,43,44,69,15,15,15,15);
      title:array[0..messize] of string =
('АЛГОРИТМИЧЕСКИЕ МЕТОДЫ',
     ' ИССЛЕДОВАНИЯ ОПЕРАЦИЙ ', ' ', ' Метод Форда
');

     type matr = array[0..20,0..20] of real;
      coord = array [1..20,1..2] of real;

     var mas:matr;
      coord_point:coord;
      i,j,t,m,n,z,x1,y1,x2,kk,iii,y2,x,y,lenth,chrus,z1,z2:integer;
      k:array[1..20] of real;
      result:array[1..20] of integer;
      error_code:array[1..5] of byte;
      fire1:array[1..yyy,1..xxx] of byte;
      fire2:array[1..yyy,1..xxx] of byte;
```

```
 mask:array[1..6] of byte;
 starx:array[1..500] of word;
 stary:array[1..500] of word;
 starc:array[1..500] of byte;
 aa,cc,pi1,s:real;
 l,inputdata,calculatedata,move:boolean;
 o:string;
 temp,cursor,lastcursor,menulevel,nline,step:byte;
 pressed:char;
 f1,f2:text;

FUNCTION min:real;
begin
 s:=0;
 for i:=1 to n do
 if (s=0) and (k[i]<>-1) then s:=k[i]
 else if(k[i]<s) and (k[i]<>-1)
 then s:=k[i];
 min:=s;
end;

PROCEDURE set_graph_mode;
begin
```

```
 z1:=installuserdriver('svga256',nil);
 initgraph(z1,z2,'');
 cleardevice;
end;

PROCEDURE pixel(x:word;y,col:byte);
begin
asm
mov bx,x
mov cl,y
mov dl,col
mov ax,0a000h
mov es,ax
mov al,0a0h
mul cl
add ax,ax
add bx,ax
mov [es:bx],dl
end;
end;
```

```
PROCEDURE install_firewall;
begin
for i:=1 to yyy do
 for j:=1 to xxx do
 begin
 fire1[i,j]:=0;
 fire2[i,j]:=0;
 end;
end;

PROCEDURE fire;
begin
for i:=1 to yyy-1 do
 for j:=1 to xxx do
 begin
 pixel(j*2+xx1,i*3+yy1,col[fire1[i,j]]);
 pixel(j*2+xx1,i*3+yy1-1,col[fire1[i,j]]);
 pixel(j*2+xx1,i*3+yy1-2,col[fire1[i,j]]);
 end;
for j:=1 to xxx do
 begin
 kk:=random(8);
 if kk<3 then fire1[yyy,j]:=16
```

```
 else fire1[yyy,j]:=round(31-kk);
 end;
for i:=yyy-1 downto 1 do
 for j:=2 to xxx-1 do
 begin
 fire2[i,j]:=round((fire1[i+1,j]+fire1[i+1,j-1]+fire1[i+1,j+1]-
random(4))/3);
 if (fire2[i,j]<16) or (fire2[i,j]>31) then fire2[i,j]:=16;
 end;
for i:=1 to yyy do
 for j:=1 to xxx do
 fire1[i,j]:=fire2[i,j];
end;

PROCEDURE ok;
begin
cleardevice;
setcolor(1);
rectangle(120,100,520,220);
rectangle(100,120,540,200);
setcolor(14);
outtextxy(180,130,'Операция произведена');
outtextxy(250,160,'корректно.');
```

```
repeat until keypressed;
end;

PROCEDURE notok;
begin
cleardevice;
setcolor(4);
rectangle(120,100,520,220);
rectangle(100,120,540,200);
setcolor(14);
outtextxy(180,130,'Операция произведена');
outtextxy(230,160,'не корректно.');
repeat until keypressed;
end;

PROCEDURE check_input_data;
begin
inputdata:=true;
for i:=1 to 5 do
 error_code[i]:=0;
for i:=0 to n do
 begin
 if mas[i,1]<>-1 then error_code[1]:=1;
```

```
 if mas[n,i]<>-1 then error_code[2]:=1;
 if mas[i,i]<>-1 then error_code[3]:=1;
 end;
for i:=1 to n do
 for j:=1 to n do
 begin
 if (mas[i,j]<>-1) and (mas[j,i]<>-1) then error_code[4]:=1;
 if (mas[i,j]<0) and (mas[i,j]<>-1) then error_code[5]:=1;
 end;
clrscr;
if error_code[1]<>0 then
 writeln('Ошибка: Не существует истока.');
if error_code[2]<>0 then
 writeln('Ошибка: Не существует стока.');
if error_code[3]<>0 then
 writeln('Ошибка: Существует дуга из одной вершины в ту же вершину.');
if error_code[4]<>0 then
 writeln('Ошибка: Существует две дуги из одной вершины в другую.');
if error_code[5]<>0 then
 writeln('Ошибка: Существует дуга с отрицительной нагрузкой.');
```

```
for i:=1 to 5 do
 if error_code[i]<>0 then inputdata:=false;
 if (z<>0) or (round(n)<>n) or (n<2) or (n>20) then inputdata:=false;
calculatedata:=false;
end;

PROCEDURE keyboard_input;
begin
z:=0;
closegraph;
clrscr;
write('Введите колличество пунктов(2-20): ');
readln(o);
val(o,n,z);
if (z<>0) or (round(n)<>n) or (n<2) or (n>20) then check_input_data;
writeln(' Введите нагрузку. Если дуга не существует, то нажмите Enter.');
writeln;
for i:=1 to n-1 do
 for j:=i to n do
 if i<>j then
```

```
 begin
 write(' Введите нагрузку от ',i,'-й вершины до ',j,'-й вершины:');
 readln(o);
 if o<>'' then val(o,mas[i,j],z)
 else mas[i,j]:=-1;
 if z<>0 then exit;
 end;
check_input_data;
set_graph_mode;
settextstyle(chrus,0,2);
if inputdata=true then ok
 else notok;
end;

PROCEDURE ramka;
begin
cleardevice;
setcolor(1);
rectangle(30,10,610,470);
rectangle(10,30,630,450);
end;
```

```
PROCEDURE save;
begin
assign(f2,name2);
rewrite(f2);
write(f2,'Кратчайший маршрут: ');
for i:=1 to lenth do
 write(f2,result[lenth-i+1]);
writeln(f2,'');
write(f2,'Длинна кратчайшего маршрута: ');
write(f2,round(mas[0,n]));
close(f2);
ok;
end;

PROCEDURE about_program;
begin
ramka;
settextstyle(chrus,0,5);
setcolor(14);
outtextxy(160,30,'О программе');
settextstyle(chrus,0,1);
setcolor(12);
outtextxy(40,100,'Программа: ');
```

```
outtextxy(40,150,'Версия: ');
outtextxy(40,175,'Назначение: ');
outtextxy(40,240,'Автор: ');
outtextxy(40,265,'Дата: ');
setcolor(8);
outtextxy(200,100,'Решение задачи о кратчайшем');
outtextxy(200,120,'маршруте методом Форда.');
outtextxy(200,150,'v1.0');
outtextxy(200,175,'Курсовой проект по дисциплине');
outtextxy(200,195,'"Алгоритмические методы иссле-');
outtextxy(200,215,'дования опираций"');
outtextxy(200,240,'');
outtextxy(200,265,'декабрь 1998 года');
setcolor(11);
outtextxy(50,395,'для большей информации смотрите README.TXT');
repeat until keypressed;
end;

PROCEDURE about_metod;
begin
ramka;
settextstyle(chrus,0,5);
```

```
setcolor(14);
outtextxy(130,30,'О методе Форда');
settextstyle(chrus,0,1);
setcolor(8);
outtextxy(40,90,'Метод Форда был разработан специально для');
outtextxy(50,110,'решения сетевых транспортных задач и осно-');
outtextxy(50,130,'ван, по существу на принципе оптимальности.');
outtextxy(40,150,'Алгоритм метода Форда содержит четыре этапа.');
outtextxy(50,170,'На первом этапе производится заполнение ис-');
outtextxy(50,190,'ходной таблицы расстояний от любого i-го');
outtextxy(50,210,'пункта в любой другой j-й пункт назначения');
outtextxy(50,230,'На втором этапе определяются для каждого');
outtextxy(50,250,'пункта некоторые параметры Ai и Aj по соот-');
```

```
outtextxy(50,270,'ветствующим формулам и правилам. Далее на');
outtextxy(50,290,'третьем этапе определяется кратчайшее рас-');
outtextxy(50,310,'стояние. Наконец, на четвертом этапе опре-');
outtextxy(50,330,'деляются кратчайшие маршруты из пункта');
outtextxy(50,350,'отправления P1 в любой пункт назначения Pj,');
outtextxy(50,370,'j=2,3,...,n.');
repeat until keypressed;
end;

PROCEDURE output_graph;
begin
settextstyle(chrus,0,1);
for i:=1 to n do
 begin
 setcolor(10);
fillellipse(round(coord_point[i,1]),round(coord_point[i,2]), 15,15);
 setcolor(15);
```

```
 str(i,o);
 if i>9 then outtextxy(round(coord_point[i,1]-12),
round(coord_point[i,2]-12),o)
else outtextxy(round(coord_point[i,1]-7),
round(coord_point[i,2]-12),o);
 end;
repeat until keypressed;
end;

PROCEDURE draw_ways;
begin
settextstyle(chrus,0,2);
for i:=1 to n do
 for j:=1 to n do
 if mas[i,j]<>-1 then
 begin
 x1:=round(coord_point[i,1]);
 y1:=round(coord_point[i,2]);
 x2:=round(coord_point[j,1]);
 y2:=round(coord_point[j,2]);
 setcolor(15);
 line(x1,y1,x2,y2);
 temp:=round(mas[i,j]);
```

```
 str(temp,o);
 setcolor(2);
outtextxy(round((x1+x2)/2+5),round((y1+y2)/2+5),o);
 end;
end;

PROCEDURE draw_short_way;
begin
for i:=1 to lenth-1 do
 begin
 setlinestyle(0,0,3);
 setcolor(red);
 x:=result[i];
 y:=result[i+1];
 x1:=round(coord_point[x,1]);
 y1:=round(coord_point[x,2]);
 x2:=round(coord_point[y,1]);
 y2:=round(coord_point[y,2]);
 line(x1,y1,x2,y2);
 end;
settextstyle(chrus,0,1);
setcolor(14);
outtextxy(50,370,'Кратчайший маршрут: ');
```

```
for i:=1 to lenth do
 begin
 str(result[lenth-i+1],o);
 outtextxy(300+i*15,370,o);
 end;
outtextxy(50,400,'Длинна кратчайшего маршрута: ');
str(round(mas[0,n]),o);
outtextxy(420,400,o);
end;

PROCEDURE count_point_coord;
begin
 pi1:=(2*pi)/n;
 m:=0;
 aa:=3*pi/2;
 for i:=1 to n do
 begin
 coord_point[i,1]:=(cos(aa)*150)+300;
 coord_point[i,2]:=(sin(aa)*150)+200;
 aa:=aa+pi1;
 end;
end;
```

```
PROCEDURE set_font;
begin
chrus:=installuserfont('fn03');
settextstyle(chrus,0,2);
end;

PROCEDURE calculate;
begin
for i:=1 to n do
 k[i]:=0;
clrscr;
mas[0,1]:=0;
mas[1,0]:=0;
{3}
 for j:=2 to n do
 begin
 for i:=1 to n do
 if (mas[0,i]<>-1) and (mas[i,j]<>-1)
 then k[i]:=mas[0,i]+mas[i,j]
 else k[i]:=-1;
 mas[0,j]:=min;
 mas[j,0]:=mas[0,j];
 end;
```

```
{4}
repeat
 l:=true;
 for i:=1 to n do
 for j:=1 to n do
 if (mas[0,j]-mas[0,i]>mas[i,j]) and (mas[i,j]<>-1) then
 begin
 l:=false;
 mas[0,j]:=mas[0,i]+mas[i,j];
 end;
until l;
{5}
j:=n;
m:=1;
t:=0;
for i:=1 to n do
 result[i]:=-1;
result[1]:=n;
repeat
 inc(m);
 for i:=1 to j do
 begin
```

```
      if (mas[i,j]<>-1) and (i<>j) and (mas[i,j]=mas[0,j]-
mas[0,i])
      then
      begin
      t:=i;
      break;
      end;
      end;
      result[m]:=t;
      j:=t;
      lenth:=m;
     until j=1;
     calculatedata:=true;
     ok;
     end;

     PROCEDURE stars;
     begin
     for i:=1 to 500 do
      begin
      starx[i]:=round(random(640));
      stary[i]:=round(random(480));
      starc[i]:=round(31-random(16));
```

```
 end;
end;

PROCEDURE draw_menu;
begin
cleardevice;
for i:=1 to 500 do
 putpixel(starx[i],stary[i],starc[i]);
cursor:=1;
lastcursor:=cursor;
 for i:=1 to 260 do
 begin
 setcolor(8);
 line(210+i,110,210+i,110);
 setcolor(4);
 line(200+i,100,200+i,100);
 end;
 for j:=1 to nline*30+10 do
 begin
 setcolor(8);
 line(210,110+j,470,110+j);
 setcolor(4);
 line(200,100+j,460,100+j);
```

```
 end;
setcolor(0);
for j:=1 to nline do
 outtextxy(220,110+(j-1)*25,menu[menulevel,j]);
end;

PROCEDURE redraw_menu;
begin
 for j:=nline*30+10 downto 1 do
 begin
 setcolor(0);
 line(210,110+j,470,110+j);
 line(200,100+j,210,100+j);
 setcolor(8);
 if j<10 then
 begin
 setcolor(0);
 line(210,100+j,470,100+j);
 end
 else
 line(210,100+j,470,100+j);
 end;
 for i:=260 downto 0 do
```

```
 begin
 putpixel(210+i,110,0);
 putpixel(200+i,100,0);
 end;
cleardevice;
end;

PROCEDURE main_menu;
begin
settextstyle(chrus,0,2);
draw_menu;
repeat
 setcolor(0);
 outtextxy(220,110+(lastcursor-
1)*25,menu[menulevel,lastcursor]);
 setcolor(7);
 outtextxy(220,110+(cursor-
1)*25,menu[menulevel,cursor]);
 pressed:=readkey;
 if pressed=#0 then
 begin
 pressed:=readkey;
 move:=false;
```

```
 if (pressed=#80) and (cursor=nline) then
 begin
lastcursor:=nline; cursor:=1;
move:=true;
 end;
 if (pressed=#72) and (cursor=1) then
 begin
lastcursor:=1;
cursor:=nline;
move:=true;
 end;

 if (pressed=#80) and (cursor<nline) and not(move) then
 begin
lastcursor:=cursor;
inc(cursor);
 end;
 if (pressed=#72) and (cursor>1) and not(move) then
 begin
lastcursor:=cursor;
dec(cursor);
 end;
 end;
```

```
      until pressed=#13;
      redraw_menu;
      if cursor=5 then about_program;
      if cursor=4 then about_metod;
      if (cursor=1) and (menulevel=3) then keyboard_input;
      if (cursor=1) and (menulevel=4) then
       begin
       closegraph;
       halt;
       end;
      if (cursor=2) and (menulevel=1) and (inputdata=false) then
notok;
      if (cursor=2) and (menulevel=1) and (inputdata=true) then
       begin
      count_point_coord;
      draw_ways;
      output_graph;
       end;
      if (cursor=2) and (menulevel=0) and (inputdata=true) then
calculate;
      if (cursor=2) and (menulevel=0) and (inputdata=false) then
notok;
```

if (cursor=1) and (menulevel=2) and (calculatedata=false) then notok;

if (cursor=1) and (menulevel=2) and (calculatedata=true) then

begin

count_point_coord;

draw_ways;

draw_short_way;

output_graph;

***end*;**

if (cursor=2) and (menulevel=2) and (calculatedata=true) then save;

if (cursor=2) and (menulevel=2) and (calculatedata=false) then notok;

if (cursor=2) and (menulevel=3) then notok;

menulevel:=menugo[menulevel,cursor];

nline:=menuof[menulevel];

main_menu;

***end*;**

PROCEDURE welcomescreen;

begin

settextstyle(chrus,0,1);

```
randomize;
install_firewall;
 for i:=0 to messize do
 begin
 setcolor(4);
 outtextxy(10,iii*step+i*30,title[i]);
 end;
repeat
 fire;
until keypressed;
end;

BEGIN
for i:=0 to 20 do
 for j:=0 to 20 do
 mas[i,j]:=-1;
stars;
inputdata:=false;
calculatedata:=false;
menulevel:=0;
nline:=menuof[menulevel];
z2:=0;
set_graph_mode;
```

set_font;

welcomescreen;

closegraph;

z2:=2;

set_graph_mode;

main_menu;

repeat until keypressed;

END.

6. Лабораторная работа № 5. Применение метода динамического программирования к задаче оптимального управления запасами

6.1 Постановка задачи.

Современный уровень развития научно-технического прогресса создает необходимые предпосылки для повышения качества управления исследуемых задач за счет использования вычислительной техники, математических методов, теории управления, автоматизации управления . Все это нашло конкретную реализацию в автоматизированных системах управления .

Управление заключается в сборе информации, ее переработке и выводе управляющей информации для изменения хода процесса .

Основным путем повышения качества управления является автоматизация управления производством, при которой данные задачи решаются средствами вычислительной техники.

Одной из задач управления предприятием является задача определения оптимального плана производства изделий, обеспечивающего заданный спрос продукции при минимизации затрат на производство и хранение продукции.

Это задача оптимального управления запасами. Теория управления запасами позволяет определять уровни запасов материалов, полуфабрикатов, производственных мощностей и других ресурсов в зависимости от спроса на них.
Проблема управления запасами является одной из наиболее важных в организационном правлении. Но, как правило, не существует типовых решений – условия на каждом предприятии или фирме уникальны и включают множество ограничений и различных особенностей. С этим связаны и проблемы, возникающие при разработке математической модели и определении оптимальной стратегии управления запасами.
В данной работе сделана попытка реализовать решение задачи управления запасами методом динамического программирования.

6.1 . Постановка задачи и ее формализованное описание

Предприятие разрабатывает календарный план выпуска изделий на плановый период, состоящий из нескольких этапов. Текущая деятельность предприятия характеризуется следующими параметрами:
1) длительность одного этапа планового периода;

2) выпуск продукции в течение этапа;

3) спрос на продукцию в конце этапа;

4) уровень запасов изделий на конец этапа;

5) максимально возможный выпуск изделий на одном этапе;

6) максимально возможный уровень запасов на одном этапе;

Известны затраты на каждом этапе планового периода, связанные с выпуском изделий и хранением запасов изделий. Так же известны затраты на формирование начального запаса.

Необходимо определить план производства изделий, чтобы обоеспечить заданный спрос продукции при минимальных затратах.

6.2 Математическое описание задачи

Введем следующие обозначения:

N – число календарных этапов из которых состоит плановый период;

При этом каждый n-й этап (n=1,N) характеризуется параметрами:

$i_n - 1$ – запас, оставшийся после окончания n-1-го этапа;

x_n – объем производства предприятия на n-м этапе;

d_n – величина спроса на продукцию предприятия на *n*-м этапе;

$x_{\max}$ – максимальный объем производства на одном этапе;

$i_{\max}$ – максимальный объем запасов на одном этапе;

$C_n(x_n, i_{n-1})$ – затраты на *n*-м этапе функционирования, связанные с выпуском xn деталей и хранением i_{n-1} запасов деталей.

Тогда критерий оптимизации имеет вид:

$$F=\sum C_n(x_n,i_n\text{-}1) \rightarrow min \qquad (1)$$

при ограничениях:

1) ограничение на удовлетворение спроса на каждом этапе:

$$d_n \text{ £ } i_n\text{-}1 + x_n \text{ , } n=1,\ldots,N \text{ ;} \qquad (2)$$

2) установление объема запаса в конце n-го периода:

$$i_n = i_{n-1} + x_n - d_n,\ n=1,\ldots,N,\quad x_n = 0.\ x_{\max},\ i_n = 0.\ i_{\max}; \qquad (3)$$

6.3. Алгоритмизация решения задачи управления запасами. Выбор метода решения

В изложенной задаче необходимо учитывать изменение моделируемого процесса во времени и влияние времени на критерий оптимальности. Для решения таких задач используется метод динамического планирования (динамическое программирование).

6.4. Описание метода динамического программирования

Динамическое программирование—это метод оптимизации многошаговых или многоэтапных процессов, критерий эффективности которых обладает свойством аддитивности (т.е. общий доход процесса равен сумме локальных доходов на отдельных этапах). В задачах динамического программирования критерий эффективности называется доходом. Данные процессы управляемые, и от правильного выбора управления зависит величина дохода.

Показатель эффективности задачи в целом обозначим через *W*, а показатели эффективности на отдельных шагах—через $\varphi_i, i = 1.m$. Если *W* обладает свойством аддитивности, т.е.

$$W = \sum \varphi_i , \qquad (4)$$

Переменная X_i от которой зависят выигрыши на *i*-м шаге и, следовательно, выигрыш в целом, называется шаговым управлением, *i=1..m.*

Управлением процесса в целом (*x*) называется последовательность шаговых управлений (вектор управлений) $x=(x_1, x_2, \ldots, x_i, \ldots, x_m)$.

Оптимальное управление *x*—это значение управления *x*, при котором значение *W(x*)* является максимальным (или минимальным, если требуется уменьшить проигрыш).

$W^*=W(x^*)=max\{W(x)\}, x \in X,$ (5)

где X—область допустимых управлений.

Оптимальное управление x^* определяется последовательностью оптимальных шаговых управлений $x^*=(x_2^*, x_2^*,\ldots, x_i^*,\ldots, x_m^*)$.

В основе метода динамического программирования лежит принцип оптимальности Беллмана, формулирующийся следующим образом: управление на каждом шаге надо выбрать так, чтобы оптимальной была сумма выигрышей на всех оставшихся до конца процесса шагах, включая выигрыш на данном шаге. Объясняется это правило так: при решении задачи динамического программирования на каждом шаге выбирается управление, которое должно привести к оптимальному выигрышу. Если считать все шаги независимыми друг от друга, то оптимальным шаговым управлением будет то управление, которое приносит максимальный выигрыш именно на данном шаге. Но, например, при покупке новой техники в замен устаревшей на ее приобретение затрачиваются определенные средства. Поэтому прибыль от ее эксплуатации вначале может быть небольшой. Однако в следующие годы новая техника будет приносить большую прибыль. И наоборот, если руководитель примет решение оставить старую технику для получения

прибыли в текущем году, то в дальнейшем это приведет к значительным убыткам. Данный пример демонстрирует следующий факт: в многошаговых процессах все шаги зависят друг от друга, и, следовательно, управление на каждом конкретном шаге надо выбирать с учетом его будущих воздействий на этот процесс.

Другой момент, который следует учитывать при выборе управления на данном шаге,—это возможные варианты окончания предыдущего шага. Эти варианты определяют состояние процесса. Например, при определении количества средств, вкладываемых в предприятие в *i*-м году, необходимо знать, какая прибыль получена в предыдущем *(i-1)*-м году. Таким образом, при выборе шагового управления необходимо учитывать:

1) возможные исходы предыдущего шага;

2) влияние управления на все оставшиеся до конца процесса шаги.

В задачах динамического программирования первый пункт учитывают, делая на каждом шаге условные предположения о возможных вариантах окончания предыдущего шага и приводя для каждого из вариантов условную оптимизацию. Выполнение второго пункта обеспечивается тем, что в задачах динамического программирования условная

оптимизация проводится от конца процесса к началу. Сперва оптимизируется последний *m*-й шаг, на котором не надо учитывать возможные воздействия выбранного управления xm на все последующие шаги, так как эти шаги просто отсутствуют. Делая предположения об условиях окончания *(m-1)*- го шага, делая предположения об исходах окончания *(m-2)*-го шага и определяя условное оптимальное управление на *(m-1)*-м шаге, приносящее оптимальный выигрыш на двух последних шагах—(*m-1*)-м и *m*-м. Так же действуют на всех остальных шагах до первого. На первом шаге, как правило, не надо делать условных предположений, так как состояние системы перед первым шагом обычно известно. Для этого состояния выбирают оптимальное шаговое управление, обеспечивающее оптимальный выигрыш на первом и всех последующих шагах. Это управление является безусловным оптимальным управлением на первом шаге и, зная его, определяются оптимальное значение выигрыша и безусловные оптимальные управления на всех шагах.

Понятия этапа(шага), состояния, управления, дохода(выигрыша) целиком зависит от предметной ориентации исследуемой системы. Для организационно-экономической системы (к которой и относится наша задача) эти понятия имеют вид:

1) этап – некий календарный интервал (месяц, квартал, год и т.д.);

2) состояние – наличие финансовых/производственных средств, свободной продукции и т. п.;

3) управление – возможные варианты использования имеющихся средств;

4) локальный доход – прибыль (или затраты) получаемая на отдельном этапе;

5) суммарный доход – прибыль (или затраты) получаемая по окончании планового периода.

Дополнительно введем следующие условные обозначения:

s—состояние процесса;

S_i—множество возможных состояний процесса перед i-м шагом;

W_i—выигрыш с i-го шага до конца процесса, $i=1..m$.

Можно определить следующие основные этапы составления математической модели задачи динамического программирования.

1) Разбиение задачи на шаги (этапы). Шаг не должен быть слишком мелким, чтобы не проводить лишних расчетов и не должен быть слишком большим, усложняющим процесс шаговой оптимизации.

2) Выбор переменных, характеризующих состояние s моделируемого процесса перед каждым шагом, и выявление налагаемых на них ограничений. В качестве таких переменных следует брать факторы, представляющие интерес для исследователя, например годовую прибыль при планировании деятельности предприятия.

3) Определение множества шаговых управлений x_i, $i=1..m$ и налагаемых на них ограничений, т.е. области допустимых управлений X.

4) Определение выигрыша $\varphi(s, x_i)$, который принесет на i-м шаге управление x_i, если система перед этим находилась в состоянии s.

5) Определение состояния s', в которое переходит система из состояния s под влиянием управления x_i : $s' = f_i(s, x_i,)$, где f_i— функция перехода на i-м шаге из состояния s в состояние s'.

6) Составление уравнения, определяющего условный оптимальный выигрыш на последнем шаге, для состояния s моделируемого процесса

$$W_m(S)=max\{\varphi_m(s, x_m)\}. \qquad (6)$$

$x_m \in X$

7) Составление основного функционального уравнения динамического программирования, определяющего условный оптимальный выигрыш для

данного состояния *s* с *i*-го шага и до конца процесса через уже известный оптимальный выигрыш с *(i+1)*-го шага и до конца:

$$W_i(S)=max\{\varphi_i(s, x_i)+W_i=1(f_i(s, x_i))\}. \quad (7)$$
$$x_m€X$$

После того как выполнены пункты 1—7, и математическая модель составлена, приступают к ее расчету. Укажем основные этапы решения задачи динамического программирования.

1) Определение множества возможных состояний S_m для последнего шага.

2) Проведение условной оптимизации для каждого состояния $s€S_m$ на последнем *m*-м шаге и определение условного оптимального управления $x(s), s € S_m$.

3) Определение множества возможных состояний S_i для *i*-го шага, $i=2, 3,…m-1$.

4) Проведение условной оптимизации для *i*-го шага, $i=2, 3,…m-1$ для каждого состояния $s€S_i$, по формуле (7) и определение условного оптимального управления $x_i(s), s € S_i$, $i=2, 3,…m-1$.

5) Определение начального состояния системы s_1, оптимального выигрыша $W_1(S_1)$ и оптимального управления

$x_1(S_1)$ по формуле (7) при $i=1$. Это есть оптимальный выигрыш для всей задачи $W^*=W_1(x_1^*)$.

6)Проведение безусловной оптимизации управления. Для проведения безусловной оптимизации необходимо найденное на первом шаге оптимальное управление $x_1^*=x_1(s_1)$ подставить в формулу (5) и определить следующее состояние системы $s_2=f_2(s_1, x_1)$. Для измененного состояния найти оптимальное управление $x_2^*=x_2(s_2)$, для i-го состояния s_i найти $s_{i+1}=f_{i+1}(s_i,x_i^*)$ и $x^*_{i+1}(s_{i+1})$ и т.д.

Определим основные компоненты:

1) этап - календарный период деятельности предприятия, $n=1,N$;

2) состояние – объем запасов in в конце n периода;

3) управление – планируемый объем производства xn на n-м периоде;

4) локальный доход – затраты на n-м этапе, связанные с хранением запасов и производством новой продукции $C_n(x_n,i_{n-1})$;

5) оператор перехода – устанавливает связь между объемом запасов в конце $n-1$-го и n-го этапов: $i_n = i_{n-1} + x_n - d_n$.

Введем функцию:

$f_n(i_n) = min\sum C_n(x_n,i_{n-1})$ (8) .

Функциональное уравнение Беллмана для такой задачи:

$f_n(i_n) = min(f_n(i_{n-1}) + C_n(x_n,i_{n-1}))$. (9)

Если

$C_n(x_n,i_{n-1}) = c_n(x_n) + h*i_{n-1}$ (10) ,

где $c_n(x_n)$ – затраты на производство продукции на *n*-ном этапе в x_n объеме;

$h*i_{n-1}$ – затраты на хранение продукции на *n*-ном этапе в объеме i_0.

Известно $c_0(x_0)$ - затраты на формирование начального запаса.

Тогда на шаге *1* принятия решения уравнение Беллмана (9) примет вид:

$f_1(i_1) = min(f_1(i_0) + C_1(x_1,i_0)) = min(f_1(i_0) + c_0(x_0) + h*i_0)$ (11)

все переменные в уравнении известны, а значит его можно решить.

На шаге *n* получим следующее уравнение:

$f_n(i_n) = min(f_n(i_n\text{-}1) + c_n(x) + h*i_{n-1})$ (12)

6.5. Алгоритм решения задачи

Для получения оптимального решения нам необходимо разработать алгоритм решения уравнения Беллмана (12) на произвольном шаге принятия решения *n.*

Для этого целесообразно воспользоваться *2*-мя таблицами. Заполнение таблицы 1 проводится так: столбцы – величина запаса с предыдущего шага, строки – объем производства на текущем этапе. Число столбцов ограничивается i_{max}, а число строк x_{max}. Клетка таблицы делится на 2 части. В одной части записываются значения состояния в конце текущего этапа $(i_n = i_{n-1} + x_n - d_n)$. Если $i_n < 0$,то это недопустимое состояние, клетка вычеркивается из рассмотрения. Во второй части клетки записывается значение функции

$$f_n(i_n) = c_{n-1}(x_{n-1}) + c_n(x_n) + h*i_{n-1} \qquad (13)$$

Среди допустимых клеток находятся клетки с одинаковыми значениями состояний, и выбирается клетка, для которой функция $f_n(i_n)$ минимальна, для нее фиксируется оптимальный объем производства. Эти результаты записываются в таблицу 2. Такие шаги повторяются *N* раз.

Для нахождения оптимальных объемов производства x_n и оптимальных уровней запасов i_n производится решение задачи в обратном порядке. На последнем этапе $(n = N)$ из таблицы 2 выбирается x_n и i_n , соответствующие оптимальной

(минимальной) функции затрат $f_n(i_n)$. На этапах $n < N$ из таблицы 2 выбираются строки для которых x_n и i_n такие, что бы $|d_n - x_{n+1}| = i_n$. Обратное решение задачи производится до $n = 1$ этапа.

6.6 Пример разработанной программы (Pascal)

```
type
  TData = record
  N: integer;              //кол-во интервалов планирования
  d: array of integer;     //спрос
  Xmax: integer;           //производственные мощности
  Imax: integer;           //предел запасов
  C1: integer;             //конст1 (формир. начальн. запасов)
  C2: integer;             //2
  C3: integer;             //3
  h: integer;              //зататы на хран
end;
type
  TTab1 = record
  f: integer;
  i: integer;
end;

type
  TTab2 = record
  i: integer;
  x: integer;
  f: integer;
end;

procedure TFormMain.ButtonRestartClick(Sender: TObject);
var f: textfile;
```

```
    i,x,step,ii: integer;
    str: string;
begin
///предполагается, что исходные данные занесены в текстовый
файл 001.TMP
AssignFile (f,'001.tmp');
Reset(f);
ReadLn(f, Data.N);
ReadLn(f, Data.H);
ReadLn(f, Data.Imax);
ReadLn(f, Data.Xmax);
ReadLn(f, Data.C1);
ReadLn(f, Data.C2);
ReadLn(f, Data.C3);
SetLength (Data.D, Data.N);
for i:=0 to Data.N-1 do ReadLn(f, Data.D[i]);
CloseFile(f);
SetLength (Tab1,Data.Xmax+1,Data.Imax+1);
SetLength (Tab2,Data.N+1,Data.Imax+1);
///результаты расчетов будем выводить в Memo список
MemoMain.Lines.Clear;
///выведем исходные данные:
MemoMain.Lines.Add('**********************************
********');
MemoMain.Lines.Add('Исходные данные: ');

MemoMain.Lines.Add('');
MemoMain.Lines.Add('Количество интервалов планирования: '
                   +floattostr(Data.N));
MemoMain.Lines.Add('Ограничение на производственные
мощности: '
                   +inttostr(Data.Xmax));
MemoMain.Lines.Add('Предельный уровень запасов: '
                   +inttostr(Data.Imax));
MemoMain.Lines.Add('Величина спроса: ');
```

```
MemoMain.Lines.Add('№ этапа | спрос ');
for i:=0 to Data.N-1 do
   MemoMain.Lines.Add(inttostr(i)+'
'+inttostr(Data.D[i]));
MemoMain.Lines.Add('Затраты на формирование начального
спроса: ');
MemoMain.Lines.Add(floattostr(Data.C1)+'*i');
MemoMain.Lines.Add('Затраты на производство и хранение: ');
MemoMain.Lines.Add(floattostr(Data.C2)+'+'+
                floattostr(Data.C3)+'*Xn+'+floattostr(Data.H)+'*i');
MemoMain.Lines.Add('**********************************
********');
///проверяем корректность входных данных
for i:=0 to Data.N-1 do
  if ((Data.Xmax+Data.Imax)<=Data.d[i]) then
     begin
     MemoMain.Lines.Add('Спрос превышает
производственные возможности.');
     MemoMain.Lines.Add('Расчет невозможен.');
     Exit;
     end;
///Формируем начальный запас
for i:=0 to Data.Imax do
       Tab2[0,i].f:=Data.C1*i;
///Пошли шагать
for step:=0 to Data.N-1 do
begin
MemoMain.Lines.Add(' ');
MemoMain.Lines.Add(' ');
MemoMain.Lines.Add('Этап № '+inttostr(step));
///Заполняем первую таблицу
MemoMain.Lines.Add('Таблица '+inttostr(step)+'.1');
MemoMain.Lines.Add('');
for i:=0 to Data.Imax do
```

```
    MemoMain.Lines.Text:=MemoMain.Lines.Text+'
i'+(inttostr(i));
for x:=0 to Data.Xmax do
  begin
  MemoMain.Lines.Add('x'+inttostr(x)+ '  ');
  for i:=0 to Data.Imax do
     begin
       Tab1[x,i].i:=i+x-Data.d[step];
       if (Tab1[x,i].i<=-1) or (Tab1[x,i].i>Data.Imax) then
          begin
            MemoMain.Lines.Text:=MemoMain.Lines.Text+('******
');
            continue;

          end
       else
          if x>0 then
             Tab1[x,i].f:=Tab2[step,i].f+Data.C2+Data.C3*x+Data.h*i
          else
             Tab1[x,i].f:=Tab2[step,i].f+Data.h*i;
MemoMain.Lines.Text:=MemoMain.Lines.Text+(inttostr(Tab1[x,
i].f)+'/'  +inttostr(Tab1[x,i].i)+'     ');
     end; ///end i tabl1
  end; ///end x tabl1
///Заполняем вторую таблицу
MemoMain.Lines.Add(' ');
MemoMain.Lines.Add('Таблица '+inttostr(step)+'.2');
for i:=0 to Data.Imax do
    Tab2[step+1,i].f:=10000000;
for ii:=0 to Data.Imax do
for x:=0 to Data.Xmax do
  begin
  for i:=0 to Data.Imax do
     begin
       if (ii=Tab1[x,i].i) then
```

```
          if Tab1[x,i].f<=Tab2[step+1,ii].f then
            begin
             Tab2[step+1,ii].f:=Tab1[x,i].f;
             Tab2[step+1,ii].i:=ii;
             Tab2[step+1,ii].x:=x;
            end;
    end;
  end;
for i:=0 to Data.Imax do
  MemoMain.Lines.Add('i='+inttostr(Tab2[step+1,i].i)+' '+
                  'x='+inttostr(Tab2[step+1,i].x)+' '+
                   'f='+inttostr(Tab2[step+1,i].f)+' ');
end; ///end step
///Обратный ход, во время его выводим результат
MemoMain.Lines.Add('');
MemoMain.Lines.Add('');
MemoMain.Lines.Add('Оптимальное решение: ');
MemoMain.Lines.Add('Общие затраты (F) = '+inttostr(ii));
for i:=0 to Data.Imax do
   if Tab2[Data.N,i].f=ii then
   begin
     MemoMain.Lines.Add('На '+inttostr(Data.N)+' этапе'+': запас
(i) = '+inttostr(Tab2[Data.N,i].i)+', '+
                      ' производство (x) =
'+inttostr(Tab2[Data.N,i].x)+';');
     x:=Tab2[Data.N,i].i;
   end;
for ii:=Data.N-1 downto 1 do
   begin
    for i:=0 to Data.Imax do

         begin
           if power(Data.d[ii]-
Tab2[ii+1,x].x,2)=power(Tab2[ii,i].i,2)
           then
```

```
            begin
              MemoMain.Lines.Add('На '+inttostr(ii)+
              ' этапе'+': запас (i) = '+
              inttostr(Tab2[ii,i].i)+', '+
              ' производство (x) = '+inttostr(Tab2[ii,i].x)+';');
              x:=Tab2[ii,i].i;
            end;
          end;
    end;
Tab1:=nil;
Tab2:=nil;
end;
```