МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность (ИУ8)

ТЕОРИЯ ИГР И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ

Лабораторная работа № 5 на тему:

«Кооперативные игры. Вектор Шепли»

Вариант 4

Преподаватель:
Коннова Н.С.

Студент:
Куликова А.В.

Группа:
ИУ8-21М

Цель работы

Изучить постановку кооперативной игры и найти оптимальное распределение выигрыша (дележ) между игроками путем вычисления компонент вектора Шепли.

Постановка задачи

Для заданной характеристической функцией игры по варианту выполнить:

1) Проверить кооперативную игру на супераддитивность и выпуклость. Если игра по варианту не супераддитивна, изменить характеристическую функцию таким образом, чтобы игра была супераддитивна. Продемонстрировать изменения и повторную проверку.
2) Составить программу вычисления компонент вектора Шепли и, в зависимости от варианта, рассчитать его.
3) Проверить условия индивидуальной и групповой рационализации...

Вариант помечен цветом:

Варианты работы

	№ варианта																	
ХФ	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
$v(Æ)$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$v(\{1\})$	4	4	3	1	3	4	4	2	3	3	3	2	3	4	2	4	1	2
$v(\{2\})$	1	3	4	1	2	2	3	1	3	1	4	3	3	1	3	4	2	3
$v(\{3\})$	3	2	1	3	3	2	2	1	1	2	4	4	2	1	2	4	1	4
$v(\{4\})$	1	3	2	3	2	1	2	2	2	4	4	1	4	1	4	2	1	4
$v(\{1,2\})$	6	8	7	4	6	7	7	4	7	4	8	6	6	7	6	9	4	6
$v(\{1,3\})$	8	6	5	4	6	7	7	4	6	5	7	7	5	7	5	9	2	6
$v(\{1,4\})$	6	7	6	6	5	5	6	4	7	8	7	4	8	7	7	6	3	7
$v(\{2,3\})$	5	5	5	4	6	4	6	2	4	3	9	7	6	3	6	10	4	7
$v(\{2,4\})$	3	7	7	4	4	4	6	4	7	5	8	5	9	2	8	7	4	8
$v(\{3,4\})$	5	6	3	7	7	3	4	4	3	6	9	6	8	2	8	7	2	8
$v(\{1,2,3\})$	9	10	10	9	10	11	11	7	10	7	13	12	9	10	9	15	7	11

$v(\{1,2,4\})$	8	12	10	9	10	9	10	8	10	10	13	9	12	10	11	12	6	11
$v(\{1,3,4\})$	10	10	9	8	9	10	11	8	9	11	13	9	10	10	10	12	5	12
$v(\{2,3,4\})$	7	9	9	8	10	7	10	6	7	8	14	10	12	6	12	14	6	12
$v(I)$	11	13	12	11	12	13	13	10	12	13	16	14	14	12	14	17	9	14

Ход работы

1. Результат вычисления супераддитивности кооперативной игры:

Перевод результата с приложения А:

В игре нет: СУПЕРАДДИТИВНОСТЬ

обнаружено нарушение условий

v({ 3 } ИЛИ { 1 4 } = { 1 3 4 }) = 8 < v({ 3 }) = 3 + v({ 1 4 }) = 6

2. Результат вычисления выпуклости:

Перевод результата с приложения А:

В игре нет: ВЫПУКЛОСТИ

обнаружено нарушение условий

v({ 3 } OR { 1 4 } = { 1 3 4 }) = 8 + v({ 3 } SUM { 1 4 } = { }) = 0 < v({ 3 }) = 3 + v({ 1 4 }) = 6

3. Результат вычисления Шепли:

Перевод результата с приложения А:

Элементы вектора Шепли X: 2,5 2,16667 3 3,33333

Сумма элементов вектора Шепли:11

Значение характеристической функции для всей совокупности игроков: 11.

Условие групповой рационализации: ВЫПОЛНЕНО.

Условие индивидуальной рационализации [1 игрок]: ВЫПОЛНЕНО

Значение элемента вектора Шепли: 2,5.

Значение характера функции: 1

Условие индивидуальной рационализации [2 игрока]: ГОТОВО.

Значение элемента вектора Шепли: 2,16667.

Значение характера функции: 1

Условие индивидуальной рационализации [3 игрока]: ГОТОВО.

Значение элемента вектора Шепли: 3

Значение характера функции: 1

Условие индивидуальной рационализации [4 игрока]: ГОТОВО.

Значение элемента вектора Шепли: 3,33333.

Значение характера функции: 1

Условие индивидуальной рационализации: ВЫПОЛНЕНО

- **Условие групповой рационализации** *выполняется*, так как сумма элементов вектора Шепли равна значению характеристической функции для всего набора игроков.
- **Условие индивидуальной рационализации** *выполняется, но не сразу**.

** не выполняется до тех пор, пока игрок №3 не найдет наилучшую и оптимальную стратегию* из-за действий, который не выбирает оптимальную стратегию, которая не соответствует его предпочтениям или интересам

Результат программы представлен в приложении А.

Выводы

В ходе проделанной работы изучили постановку кооперативной игры и найдено оптимальное распределение выигрыша (дележ) между игроками путем вычисления компонент вектора Шепли

Контрольные вопросы

1. Дайте определение кооперативной игры и ее характеристической функции.

Кооперативная игра - это математическая модель, в которой группа игроков объединяется в коалиции для достижения определенной цели или выигрыша. В кооперативных играх игроки сотрудничают друг с другом, обмениваются ресурсами и принимают совместные решения для максимизации своего общего выигрыша.

Характеристическая функция в кооперативной игре определяет, сколько выигрыша может получить каждая возможная коалиция игроков. Формально, характеристическая функция v определена на множестве всех подмножеств игроков 2^N, где N - множество всех игроков в игре. Для любой коалиции игроков $S \subset N$, значение характеристической функции v(S) равно выигрышу, который может получить данная коалиция игроков.

2. Дайте определение выпуклой игры. Является ли любая выпуклая игра супераддитивной и наоборот?

Выпуклая игра - это такая кооперативная игра, в которой сумма выигрышей, полученных двумя коалициями игроков, больше или равна выигрышу, полученному объединением этих двух коалиций. Формально, игра называется выпуклой, если для любых двух коалиций S и T выполняется условие:

$$v(S \cup T) \geq v(S) + v(T)$$

Где v(S) - выигрыш, который может получить коалиция игроков S.

Не любая выпуклая игра является супераддитивной. Супераддитивная игра - это игра, в которой сумма выигрышей, полученных двумя коалициями игроков, равна выигрышу, полученному объединением этих двух коалиций. Формально, игра называется супераддитивной, если для любых двух коалиций S и T выполняется условие:

$$v(S \cup T) = v(S) + v(T)$$

3. Что означает дележ в кооперативной игре? Перечислите аксиомы рационального дележа.

Дележ в кооперативной игре означает распределение выигрыша между участниками коалиции. Рациональный дележ предполагает, что участники коалиции стремятся к максимизации своего индивидуального выигрыша при условии соблюдения некоторых основных принципов или аксиом.

Основные аксиомы рационального дележа в кооперативных играх:

1. Аксиома аддитивности: Если игроки объединяются в коалицию S, а затем делят свой общий выигрыш между собой, то сумма выигрышей каждого игрока должна быть равна общему выигрышу коалиции.

2. Аксиома индивидуальной рациональности: Каждый игрок должен получить не менее своего минимально приемлемого выигрыша, т.е. никто не должен быть хуже, чем если бы он играл в одиночку.

3. Аксиома симметрии: Если два игрока вносят одинаковый вклад в коалицию, то их выигрыши должны быть симметричными.

4. Аксиома связности: Если коалиция делится на две непересекающиеся подгруппы, то выигрыши этих подгрупп должны быть независимыми.

5. Аксиома стабильности: Дележ является стабильным, если никакая другая коалиция не может предложить участникам больший совокупный выигрыш.

4. На примере вариантов работы поясните аксиому линейности.

Аксиома линейности в теории кооперативных игр гласит, что сумма выигрышей, полученных от объединения двух непересекающихся коалиций, должна быть равна сумме выигрышей, полученных от каждой из этих коалиций по отдельности.

пример:?

5. Поясните смысл условий групповой и индивидуальной рационализации.

Условия групповой и индивидуальной рационализации являются ключевыми концепциями в теории кооперативных игр, которые помогают оценить справедливость и стабильность различных дележей выигрыша между участниками коалиций.

1. Групповая рационализация: Групповая рационализация означает, что дележ выигрыша между участниками коалиции должен быть таким, чтобы ни одна другая коалиция не могла предложить им лучший вариант.

2. Индивидуальная рационализация: Индивидуальная рационализация означает, что каждый участник коалиции должен получить не менее своего минимально приемлемого выигрыша. Это условие гарантирует, что ни один участник не будет хуже, чем если бы играл в одиночку или присоединился к другой коалиции. Индивидуальная рационализация обеспечивает защиту интересов каждого участника и предотвращает возможные ситуации, когда участники получают менее желаемый выигрыш.

6. Каким свойством обладает «нулевой игрок»?

Нулевой игрок" (или "игрок с нулевой суммой") - это игрок в теории игр, который не имеет собственных интересов в игре и действует исключительно

для того, чтобы максимизировать потери или выигрыши других игроков. Он не стремится к собственной выгоде, а является своего рода "нейтральным" участником, чьи действия направлены на изменение выигрышей других игроков.

Основное свойство "нулевого игрока" заключается в том, что он не влияет на собственные выигрыши или потери, а лишь изменяет результаты других участников. Его решения и стратегии могут быть направлены на создание равновесия между другими игроками или на увеличение или уменьшение общего выигрыша в игре.

7. Раскройте смысл вектора Шепли.

Вектор Шепли - это концепция, используемая в теории кооперативных игр для распределения прибыли или выигрыша между участниками игры. Он был разработан Ллойдом Шепли в 1953 году и представляет собой способ определения справедливого распределения выигрыша, учитывая вклад каждого игрока в общий результат.

Суть вектора Шепли заключается в следующем: предполагается, что каждый игрок приносит свой вклад в общий выигрыш, и это определяет его "вес" в процессе распределения. Вектор Шепли учитывает все возможные коалиции игроков и определяет, какой выигрыш должен быть распределен между ними с учетом их вклада.

Процесс определения вектора Шепли:

1. Рассмотрение всех возможных перестановок игроков (коалиций).

2. Определение, какой выигрыш достается каждой коалиции.

3. Расчет среднего выигрыша для каждого игрока, учитывая все возможные коалиции.

4. Получение окончательного вектора Шепли, который определяет справедливое распределение выигрыша между игроками.

Приложение А

gameTheoryAndOperationsResearch_superadditivity_l5.txt

v({ } OR { 1 } = { 1 }) = 1 >= v({ }) = 0 + v({ 1 }) = 1
v({ } OR { 2 } = { 2 }) = 1 >= v({ }) = 0 + v({ 2 }) = 1
v({ } OR { 1 2 } = { 1 2 }) = 4 >= v({ }) = 0 + v({ 1 2 }) = 4
v({ } OR { 3 } = { 3 }) = 3 >= v({ }) = 0 + v({ 3 }) = 3
v({ } OR { 1 3 } = { 1 3 }) = 4 >= v({ }) = 0 + v({ 1 3 }) = 4
v({ } OR { 2 3 } = { 2 3 }) = 4 >= v({ }) = 0 + v({ 2 3 }) = 4
v({ } OR { 1 2 3 } = { 1 2 3 }) = 9 >= v({ }) = 0 + v({ 1 2 3 }) = 9
v({ } OR { 4 } = { 4 }) = 3 >= v({ }) = 0 + v({ 4 }) = 3
v({ } OR { 1 4 } = { 1 4 }) = 6 >= v({ }) = 0 + v({ 1 4 }) = 6
v({ } OR { 2 4 } = { 2 4 }) = 4 >= v({ }) = 0 + v({ 2 4 }) = 4
v({ } OR { 1 2 4 } = { 1 2 4 }) = 9 >= v({ }) = 0 + v({ 1 2 4 }) = 9
v({ } OR { 3 4 } = { 3 4 }) = 7 >= v({ }) = 0 + v({ 3 4 }) = 7
v({ } OR { 1 3 4 } = { 1 3 4 }) = 8 >= v({ }) = 0 + v({ 1 3 4 }) = 8
v({ } OR { 2 3 4 } = { 2 3 4 }) = 8 >= v({ }) = 0 + v({ 2 3 4 }) = 8
v({ } OR { 1 2 3 4 } = { 1 2 3 4 }) = 11 >= v({ }) = 0 + v({ 1 2 3 4 }) = 11
v({ 1 } OR { 2 } = { 1 2 }) = 4 >= v({ 1 }) = 1 + v({ 2 }) = 1
v({ 1 } OR { 3 } = { 1 3 }) = 4 >= v({ 1 }) = 1 + v({ 3 }) = 3
v({ 1 } OR { 2 3 } = { 1 2 3 }) = 9 >= v({ 1 }) = 1 + v({ 2 3 }) = 4
v({ 1 } OR { 4 } = { 1 4 }) = 6 >= v({ 1 }) = 1 + v({ 4 }) = 3
v({ 1 } OR { 2 4 } = { 1 2 4 }) = 9 >= v({ 1 }) = 1 + v({ 2 4 }) = 4
v({ 1 } OR { 3 4 } = { 1 3 4 }) = 8 >= v({ 1 }) = 1 + v({ 3 4 }) = 7
v({ 1 } OR { 2 3 4 } = { 1 2 3 4 }) = 11 >= v({ 1 }) = 1 + v({ 2 3 4 }) = 8
v({ 2 } OR { 3 } = { 2 3 }) = 4 >= v({ 2 }) = 1 + v({ 3 }) = 3
v({ 2 } OR { 1 3 } = { 1 2 3 }) = 9 >= v({ 2 }) = 1 + v({ 1 3 }) = 4
v({ 2 } OR { 4 } = { 2 4 }) = 4 >= v({ 2 }) = 1 + v({ 4 }) = 3
v({ 2 } OR { 1 4 } = { 1 2 4 }) = 9 >= v({ 2 }) = 1 + v({ 1 4 }) = 6
v({ 2 } OR { 3 4 } = { 2 3 4 }) = 8 >= v({ 2 }) = 1 + v({ 3 4 }) = 7
v({ 2 } OR { 1 3 4 } = { 1 2 3 4 }) = 11 >= v({ 2 }) = 1 + v({ 1 3 4 }) = 8
v({ 1 2 } OR { 3 } = { 1 2 3 }) = 9 >= v({ 1 2 }) = 4 + v({ 3 }) = 3
v({ 1 2 } OR { 4 } = { 1 2 4 }) = 9 >= v({ 1 2 }) = 4 + v({ 4 }) = 3
v({ 1 2 } OR { 3 4 } = { 1 2 3 4 }) = 11 >= v({ 1 2 }) = 4 + v({ 3 4 }) = 7
v({ 3 } OR { 4 } = { 3 4 }) = 7 >= v({ 3 }) = 3 + v({ 4 }) = 3
The game is not: SUPERADDITIVITY
condition violation found
v({ 3 } OR { 1 4 } = { 1 3 4 }) = 8 < v({ 3 }) = 3 + v({ 1 4 }) = 6

Результат gameTheoryAndOperationsResearch_bulge_l5.txt

v({ } OR { 1 } = { 1 }) = 1 + v({ } SUM { 1 } = { }) = 0 >= v({ }) = 0 + v({ 1 }) = 1
v({ } OR { 2 } = { 2 }) = 1 + v({ } SUM { 2 } = { }) = 0 >= v({ }) = 0 + v({ 2 }) = 1
v({ } OR { 1 2 } = { 1 2 }) = 4 + v({ } SUM { 1 2 } = { }) = 0 >= v({ }) = 0 + v({ 1 2 }) = 4
v({ } OR { 3 } = { 3 }) = 3 + v({ } SUM { 3 } = { }) = 0 >= v({ }) = 0 + v({ 3 }) = 3
v({ } OR { 1 3 } = { 1 3 }) = 4 + v({ } SUM { 1 3 } = { }) = 0 >= v({ }) = 0 + v({ 1 3 }) = 4
v({ } OR { 2 3 } = { 2 3 }) = 4 + v({ } SUM { 2 3 } = { }) = 0 >= v({ }) = 0 + v({ 2 3 }) = 4
v({ } OR { 1 2 3 } = { 1 2 3 }) = 9 + v({ } SUM { 1 2 3 } = { }) = 0 >= v({ }) = 0 + v({ 1 2 3 }) = 9
v({ } OR { 4 } = { 4 }) = 3 + v({ } SUM { 4 } = { }) = 0 >= v({ }) = 0 + v({ 4 }) = 3
v({ } OR { 1 4 } = { 1 4 }) = 6 + v({ } SUM { 1 4 } = { }) = 0 >= v({ }) = 0 + v({ 1 4 }) = 6
v({ } OR { 2 4 } = { 2 4 }) = 4 + v({ } SUM { 2 4 } = { }) = 0 >= v({ }) = 0 + v({ 2 4 }) = 4
v({ } OR { 1 2 4 } = { 1 2 4 }) = 9 + v({ } SUM { 1 2 4 } = { }) = 0 >= v({ }) = 0 + v({ 1 2 4 }) = 9
v({ } OR { 3 4 } = { 3 4 }) = 7 + v({ } SUM { 3 4 } = { }) = 0 >= v({ }) = 0 + v({ 3 4 }) = 7
v({ } OR { 1 3 4 } = { 1 3 4 }) = 8 + v({ } SUM { 1 3 4 } = { }) = 0 >= v({ }) = 0 + v({ 1 3 4 }) = 8
v({ } OR { 2 3 4 } = { 2 3 4 }) = 8 + v({ } SUM { 2 3 4 } = { }) = 0 >= v({ }) = 0 + v({ 2 3 4 }) = 8
v({ } OR { 1 2 3 4 } = { 1 2 3 4 }) = 11 + v({ } SUM { 1 2 3 4 } = { }) = 0 >= v({ }) = 0 + v({ 1 2 3 4 }) = 11
v({ 1 } OR { 2 } = { 1 2 }) = 4 + v({ 1 } SUM { 2 } = { }) = 0 >= v({ 1 }) = 1 + v({ 2 }) = 1
v({ 1 } OR { 1 2 } = { 1 2 }) = 4 + v({ 1 } SUM { 1 2 } = { 1 }) = 1 >= v({ 1 }) = 1 + v({ 1 2 }) = 4
v({ 1 } OR { 3 } = { 1 3 }) = 4 + v({ 1 } SUM { 3 } = { }) = 0 >= v({ 1 }) = 1 + v({ 3 }) = 3
v({ 1 } OR { 1 3 } = { 1 3 }) = 4 + v({ 1 } SUM { 1 3 } = { 1 }) = 1 >= v({ 1 }) = 1 + v({ 1 3 }) = 4
v({ 1 } OR { 2 3 } = { 1 2 3 }) = 9 + v({ 1 } SUM { 2 3 } = { }) = 0 >= v({ 1 }) = 1 + v({ 2 3 }) = 4
v({ 1 } OR { 1 2 3 } = { 1 2 3 }) = 9 + v({ 1 } SUM { 1 2 3 } = { 1 }) = 1 >= v({ 1 }) = 1 + v({ 1 2 3 }) = 9
v({ 1 } OR { 4 } = { 1 4 }) = 6 + v({ 1 } SUM { 4 } = { }) = 0 >= v({ 1 }) = 1 + v({ 4 }) = 3
v({ 1 } OR { 1 4 } = { 1 4 }) = 6 + v({ 1 } SUM { 1 4 } = { 1 }) = 1 >= v({ 1 }) = 1 + v({ 1 4 }) = 6
v({ 1 } OR { 2 4 } = { 1 2 4 }) = 9 + v({ 1 } SUM { 2 4 } = { }) = 0 >= v({ 1 }) = 1 + v({ 2 4 }) = 4
v({ 1 } OR { 1 2 4 } = { 1 2 4 }) = 9 + v({ 1 } SUM { 1 2 4 } = { 1 }) = 1 >= v({ 1 }) = 1 + v({ 1 2 4 }) = 9
v({ 1 } OR { 3 4 } = { 1 3 4 }) = 8 + v({ 1 } SUM { 3 4 } = { }) = 0 >= v({ 1 }) = 1 + v({ 3 4 }) = 7
v({ 1 } OR { 1 3 4 } = { 1 3 4 }) = 8 + v({ 1 } SUM { 1 3 4 } = { 1 }) = 1 >= v({ 1 }) = 1 + v({ 1 3 4 }) = 8
v({ 1 } OR { 2 3 4 } = { 1 2 3 4 }) = 11 + v({ 1 } SUM { 2 3 4 } = { }) = 0 >= v({ 1 }) = 1 + v({ 2 3 4 }) = 8
v({ 1 } OR { 1 2 3 4 } = { 1 2 3 4 }) = 11 + v({ 1 } SUM { 1 2 3 4 } = { 1 }) = 1 >= v({ 1 }) = 1 + v({ 1 2 3 4 }) = 11
v({ 2 } OR { 1 2 } = { 1 2 }) = 4 + v({ 2 } SUM { 1 2 } = { 2 }) = 1 >= v({ 2 }) = 1 + v({ 1 2 }) = 4
v({ 2 } OR { 3 } = { 2 3 }) = 4 + v({ 2 } SUM { 3 } = { }) = 0 >= v({ 2 }) = 1 + v({ 3 }) = 3
v({ 2 } OR { 1 3 } = { 1 2 3 }) = 9 + v({ 2 } SUM { 1 3 } = { }) = 0 >= v({ 2 }) = 1 + v({ 1 3 }) = 4
v({ 2 } OR { 2 3 } = { 2 3 }) = 4 + v({ 2 } SUM { 2 3 } = { 2 }) = 1 >= v({ 2 }) = 1 + v({ 2 3 }) = 4
v({ 2 } OR { 1 2 3 } = { 1 2 3 }) = 9 + v({ 2 } SUM { 1 2 3 } = { 2 }) = 1 >= v({ 2 }) = 1 + v({ 1 2 3 }) = 9
v({ 2 } OR { 4 } = { 2 4 }) = 4 + v({ 2 } SUM { 4 } = { }) = 0 >= v({ 2 }) = 1 + v({ 4 }) = 3
v({ 2 } OR { 1 4 } = { 1 2 4 }) = 9 + v({ 2 } SUM { 1 4 } = { }) = 0 >= v({ 2 }) = 1 + v({ 1 4 }) = 6
v({ 2 } OR { 2 4 } = { 2 4 }) = 4 + v({ 2 } SUM { 2 4 } = { 2 }) = 1 >= v({ 2 }) = 1 + v({ 2 4 }) = 4
v({ 2 } OR { 1 2 4 } = { 1 2 4 }) = 9 + v({ 2 } SUM { 1 2 4 } = { 2 }) = 1 >= v({ 2 }) = 1 + v({ 1 2 4 }) = 9
v({ 2 } OR { 3 4 } = { 2 3 4 }) = 8 + v({ 2 } SUM { 3 4 } = { }) = 0 >= v({ 2 }) = 1 + v({ 3 4 }) = 7
v({ 2 } OR { 1 3 4 } = { 1 2 3 4 }) = 11 + v({ 2 } SUM { 1 3 4 } = { }) = 0 >= v({ 2 }) = 1 + v({ 1 3 4 }) = 8
v({ 2 } OR { 2 3 4 } = { 2 3 4 }) = 8 + v({ 2 } SUM { 2 3 4 } = { 2 }) = 1 >= v({ 2 }) = 1 + v({ 2 3 4 }) = 8
v({ 2 } OR { 1 2 3 4 } = { 1 2 3 4 }) = 11 + v({ 2 } SUM { 1 2 3 4 } = { 2 }) = 1 >= v({ 2 }) = 1 + v({ 1 2 3 4 }) = 11
v({ 1 2 } OR { 3 } = { 1 2 3 }) = 9 + v({ 1 2 } SUM { 3 } = { }) = 0 >= v({ 1 2 }) = 4 + v({ 3 }) = 3
v({ 1 2 } OR { 1 3 } = { 1 2 3 }) = 9 + v({ 1 2 } SUM { 1 3 } = { 1 }) = 1 >= v({ 1 2 }) = 4 + v({ 1 3 }) = 4
v({ 1 2 } OR { 2 3 } = { 1 2 3 }) = 9 + v({ 1 2 } SUM { 2 3 } = { 2 }) = 1 >= v({ 1 2 }) = 4 + v({ 2 3 }) = 4
v({ 1 2 } OR { 1 2 3 } = { 1 2 3 }) = 9 + v({ 1 2 } SUM { 1 2 3 } = { 1 2 }) = 4 >= v({ 1 2 }) = 4 + v({ 1 2 3 }) = 9
v({ 1 2 } OR { 4 } = { 1 2 4 }) = 9 + v({ 1 2 } SUM { 4 } = { }) = 0 >= v({ 1 2 }) = 4 + v({ 4 }) = 3
v({ 1 2 } OR { 1 4 } = { 1 2 4 }) = 9 + v({ 1 2 } SUM { 1 4 } = { 1 }) = 1 >= v({ 1 2 }) = 4 + v({ 1 4 }) = 6
v({ 1 2 } OR { 2 4 } = { 1 2 4 }) = 9 + v({ 1 2 } SUM { 2 4 } = { 2 }) = 1 >= v({ 1 2 }) = 4 + v({ 2 4 }) = 4
v({ 1 2 } OR { 1 2 4 } = { 1 2 4 }) = 9 + v({ 1 2 } SUM { 1 2 4 } = { 1 2 }) = 4 >= v({ 1 2 }) = 4 + v({ 1 2 4 }) = 9
v({ 1 2 } OR { 3 4 } = { 1 2 3 4 }) = 11 + v({ 1 2 } SUM { 3 4 } = { }) = 0 >= v({ 1 2 }) = 4 + v({ 3 4 }) = 7
v({ 1 2 } OR { 1 3 4 } = { 1 2 3 4 }) = 11 + v({ 1 2 } SUM { 1 3 4 } = { 1 }) = 1 >= v({ 1 2 }) = 4 + v({ 1 3 4 }) = 8
v({ 1 2 } OR { 2 3 4 } = { 1 2 3 4 }) = 11 + v({ 1 2 } SUM { 2 3 4 } = { 2 }) = 1 >= v({ 1 2 }) = 4 + v({ 2 3 4 }) = 8
v({ 1 2 } OR { 1 2 3 4 } = { 1 2 3 4 }) = 11 + v({ 1 2 } SUM { 1 2 3 4 } = { 1 2 }) = 4 >= v({ 1 2 }) = 4 + v({ 1 2 3 4 }) = 11
v({ 3 } OR { 1 3 } = { 1 3 }) = 4 + v({ 3 } SUM { 1 3 } = { 3 }) = 3 >= v({ 3 }) = 3 + v({ 1 3 }) = 4
v({ 3 } OR { 2 3 } = { 2 3 }) = 4 + v({ 3 } SUM { 2 3 } = { 3 }) = 3 >= v({ 3 }) = 3 + v({ 2 3 }) = 4
v({ 3 } OR { 1 2 3 } = { 1 2 3 }) = 9 + v({ 3 } SUM { 1 2 3 } = { 3 }) = 3 >= v({ 3 }) = 3 + v({ 1 2 3 }) = 9
v({ 3 } OR { 4 } = { 3 4 }) = 7 + v({ 3 } SUM { 4 } = { }) = 0 >= v({ 3 }) = 3 + v({ 4 }) = 3

```
The game is not: BULGE
condition violation found
v({ 3 } OR { 1 4 } = { 1 3 4 }) = 8 + v({ 3 } SUM { 1 4 } = { }) = 0 < v({ 3 }) = 3 + v({ 1 4 }) = 6
```

Результат gameTheoryAndOperationsResearch_l5.txt

```
Elements of the Shapley vector X: 2.5 2.16667 3 3.33333
The sum of the elements of the Shapley vector:11
The value of the characteristic function for the entire set of players: 11
The condition of group rationalization: DONE

The condition of individual rationalization [1 player]:DONE
The value of the element of the Shapley vector: 2.5
The meaning of the character of the function: 1

The condition of individual rationalization [2 player]:DONE
The value of the element of the Shapley vector: 2.16667
The meaning of the character of the function: 1

The condition of individual rationalization [3 player]:DONE
The value of the element of the Shapley vector: 3
The meaning of the character of the function: 1

The condition of individual rationalization [4 player]:DONE
The value of the element of the Shapley vector: 3.33333
The meaning of the character of the function: 1
The condition of individual rationalization: DONE
```

Приложение Б

исходный код CMakeLists.txt:

```
cmake_minimum_required(VERSION 2.8)

add_executable(main
        main.cpp main.define.h main.func.h
    )
```

исходный код main.cpp:

```
#include "main.define.h"
#include "main.func.h"

#include <string>

// Макрос для вывода сообщения "The game is: x" в файл
#define PRINT_YES(x, file) (file << "The game is: " << x)

// Макрос для вывода сообщения "The game is not: x" в файл
#define PRINT_NO(x, file) (file << "The game is not: " << x)

// Макрос для вывода произвольного комментария x в файл
#define PRINT_COMMENT(x, file) (file << x)

// Константа для сообщения "DONE"
#define PRINT_DONE "DONE"

// Константа для сообщения "NOT DONE"
#define PRINT_NOT_DONE "NOT DONE"

// Макрос для вывода сообщения о состоянии рационализации группы: "The
condition of group rationalization: x"
#define PRINT_RADIALIZ(x, file) (file << "The condition of group
rationalization: " << x)

// Макрос для вывода сообщений о состоянии индивидуальной рационализации
#define PRINT_INDIVID_RADIALIZ(num, x, vector, vfunc, file) \
    (file \
    << "\nThe condition of individual rationalization [" << num \
    << " player]:" << x \
    << "\nThe value of the element of the Shapley vector: " << vector \
    << "\nThe meaning of the character of the function: " << vfunc \
    )
#define PRINT_NOT_DONE_INDIVID_RADIALIZ(file)     (file \
    << "SEARCHING FOR THE BEST...")

namespace gameTheoryAndOperationsResearch
{

int _main(int argc, char* argv[])
{
    // Проверка игры на супераддитивность
    // Проверяем все возможные пары множеств игроков, чтобы убедиться, что их
пересечения равны 0

    std::ofstream  fout(gameTheoryAndOperationsResearch_filename);
// Поток для записи в файл
```

```
    std::ofstream
fout1(gameTheoryAndOperationsResearch_filename_superadditivity); // Поток для
записи результатов проверки на супераддитивность
    std::ofstream  fout2(gameTheoryAndOperationsResearch_filename_bulge);
// Поток для записи результатов проверки на выпуклость

    bool flagAdd = true;     // Флаг супераддитивности
    int i, j, k1, k2;        // Для задания индексов

    for (i = 0; i < N_KOOL - 1; i++)
    {
        // Индекс i задает первое множество игроков
        for (j = i + 1; j < N_KOOL; j++)
        {
            // Индекс j задает второе множество игроков
            k1 = i & j;

            // Побитовое И (пересечение множеств)
            if (k1 == 0)
            {
                k2 = i | j;

                // Побитовое ИЛИ (объединение множеств)
                if (!(v[k2] >= v[i] + v[j]))
                {
                    // Нет супераддитивности
                    flagAdd = false;
                    break;
                }

                fout1 << "v({ ";
                gameTheoryAndOperationsResearch_print(i, fout1);
                fout1 << "} OR { ";
                gameTheoryAndOperationsResearch_print(j, fout1);
                fout1 << "} = { ";
                gameTheoryAndOperationsResearch_print(k2, fout1);
                fout1 << "}) = " << v[k2] << " >= v({ ";
                gameTheoryAndOperationsResearch_print(i, fout1);
                fout1 << "}) = " << v[i] << " + v({ ";
                gameTheoryAndOperationsResearch_print(j, fout1);
                fout1 << "}) = " << v[j]<<std::endl;
            }
        }

        // Если нет супераддитивности
        if (!flagAdd)
            break;
    }


    // Есть супераддитивность
    if (flagAdd)
        PRINT_YES("SUPERADDITIVITY", fout1);
    else
    {
        PRINT_NO("SUPERADDITIVITY", fout1);
        PRINT_COMMENT("\ncondition violation found",fout1);

        fout1 << std::endl;
        fout1 << "v({ ";    gameTheoryAndOperationsResearch_print(i, fout1);
        fout1 << "} OR { ";  gameTheoryAndOperationsResearch_print(j, fout1);
        fout1 << "} = { ";  gameTheoryAndOperationsResearch_print(k2, fout1);
        fout1 << "}) = " << v[k2] << " < v({ ";
gameTheoryAndOperationsResearch_print(i, fout1);
```

```
        fout1 << "}) = " << v[i] << " + v({ ";
gameTheoryAndOperationsResearch_print(j, fout1);
        fout1 << "}) = " << v[j];
    }
    fout1 << "\n\n";

    // Проверка игра на выпуклость
    // Проверяем все возможные пары множеств игроков
    bool flagVyp = true; // Флаг выпуклости

    for (i = 0; i < N_KOOL - 1; i++) // Индекс i задает первое множество
игроков
    {
        for (j = i + 1; j < N_KOOL; j++) // Индекс j задает второе множество
игроков
        {
            k1 = i & j;  // Побитовое И (пересечение множеств)
            k2 = i | j; // Побитовое ИЛИ (объединение множеств)
            if (!(v[k1] + v[k2] >= v[i] + v[j]))  // Условие выпуклости не
выполняется
            {
                flagVyp = false; //  Нет выпуклости
                break;
            }
            fout2 << "v({ ";    gameTheoryAndOperationsResearch_print(i,
fout2);
            fout2 << "} OR { ";  gameTheoryAndOperationsResearch_print(j,
fout2);
            fout2 << "} = { ";  gameTheoryAndOperationsResearch_print(k2,
fout2);
            fout2 << "}) = " << v[k2];

            fout2 << " + v({ "; gameTheoryAndOperationsResearch_print(i,
fout2);
            fout2 << "} SUM { ";  gameTheoryAndOperationsResearch_print(j,
fout2);
            fout2 << "} = { ";  gameTheoryAndOperationsResearch_print(k1,
fout2);
            fout2 << "}) = " << v[k1];

            fout2 << " >= v({ ";
gameTheoryAndOperationsResearch_print(i, fout2);
            fout2 << "}) = " << v[i] << " + v({ ";
gameTheoryAndOperationsResearch_print(j, fout2);
            fout2 << "}) = " << v[j]<<std::endl;

        }
        if (!flagVyp) // Нет выпуклости
            break;
    }

    if (flagVyp) // Есть выпуклость
        PRINT_YES("BULGE", fout2);
    else
    {
        PRINT_NO("BULGE", fout2);
        PRINT_COMMENT("\ncondition violation found", fout2);

        fout2 << std::endl;
        fout2 << "v({ ";    gameTheoryAndOperationsResearch_print(i, fout2);
        fout2 << "} OR { ";  gameTheoryAndOperationsResearch_print(j, fout2);
        fout2 << "} = { ";  gameTheoryAndOperationsResearch_print(k2, fout2);
        fout2 << "}) = " << v[k2];
```

```
        fout2 << " + v({ ";     gameTheoryAndOperationsResearch_print(i,
fout2);
        fout2 << "} SUM { ";       gameTheoryAndOperationsResearch_print(j,
fout2);
        fout2 << "} = { ";      gameTheoryAndOperationsResearch_print(k1,
fout2);
        fout2 << "}) = " << v[k1];

        fout2 << " < v({ ";
gameTheoryAndOperationsResearch_print(i, fout2);
        fout2 << "}) = " << v[i] << " + v({ ";
gameTheoryAndOperationsResearch_print(j, fout2);
        fout2 << "}) = " << v[j];

    }
    fout2 << std::endl;

    // Для вектора Шепли
    double X[4];

    // Факториал
    int n_gameTheoryAndOperationsResearch_fact =
gameTheoryAndOperationsResearch_fact(4);
    double sum = 0;

    fout << "\nElements of the Shapley vector X: ";

    // В цикле значение i - номера игрока
    for (i = 1; i <= 4; i++)
    {
        X[i - 1] = 0;

        // В цикле j задает множество (коалицию) игроков
        for (j = 1; j < N_KOOL; j++) {
            // Игрок есть в множестве
            if
(gameTheoryAndOperationsResearch_checking_presence_player_set(i, j))
            {
                X[i - 1] +=

gameTheoryAndOperationsResearch_fact(gameTheoryAndOperationsResearch_length(j
) - 1) *
                        gameTheoryAndOperationsResearch_fact(4 -
gameTheoryAndOperationsResearch_length(j)) *
                        (
                            v[j]
                            -

v[gameTheoryAndOperationsResearch_delete_presence_player_set(i, j)]
                        );
            }
        }
        X[i - 1] /= n_gameTheoryAndOperationsResearch_fact;
        sum += X[i - 1]; // Считаем сумму
        fout << X[i - 1] << ' ';
    }


    // Проверка групповой рационализация
    // Сумма элементов вектора Шепли равна значению характеристической
функции для всего множества игроков

    fout << std::endl << "The sum of the elements of the Shapley vector:" <<
sum;
```

```
    fout << std::endl << "The value of the characteristic function for the
entire set of players: " << v[N_KOOL-1];
    fout << std::endl;

    if (sum == v[N_KOOL - 1])
        PRINT_RADIALIZ(PRINT_DONE, fout);
    else
        PRINT_RADIALIZ(PRINT_NOT_DONE, fout);
    fout << std::endl;
    /*
     * Проверка условия индивидуальной рационализации
     *
     * Значения элемента вектора Шепли для игрока больше,
     * чем значение характеристической функции для множества, состоящего из
этого игрока
     */

    bool flagIndRac = true; // условия индивидуальной рационализации
выполнены (первоначально)

    // 0001
    bool player1 = false, player2 = false, player3 = false, player4 = false;
    int NUMBER_MATRIX = 1;

    while (!(player1 && player2 && player3 && player4)) {
        // std::cout << "\n[run]";

        if (NUMBER_MATRIX == 11)
            NUMBER_MATRIX = 1;

        if (player1 && player2 && player3 && player4)
            break;

        // ------------------ run player1 ---------------------
        if (!player1) {
            // std::cout << "\n[run player1]";
            if (X[0] <= v[NUMBER_MATRIX]) {
                flagIndRac = false;
                PRINT_INDIVID_RADIALIZ("1", PRINT_NOT_DONE, nullptr, nullptr,
fout); // Условие для 1-го игрока не выполнено
                fout << std::endl;
                PRINT_NOT_DONE_INDIVID_RADIALIZ(fout);
            } else {
                PRINT_INDIVID_RADIALIZ("1", PRINT_DONE, X[0],
v[NUMBER_MATRIX], fout);
                player1 = true;
                flagIndRac = true;
                NUMBER_MATRIX = 1;
            }
            fout << std::endl;
        }

        // ------------------ run player2 ---------------------
        if (player1 && !player2) {
            // std::cout << "\n[run player2]";
            if (X[1] <= v[NUMBER_MATRIX]) {
                flagIndRac = false;
                PRINT_INDIVID_RADIALIZ("2", PRINT_NOT_DONE, nullptr, nullptr,
fout); // Условие для 2-го игрока не выполнено
                fout << std::endl;
                PRINT_NOT_DONE_INDIVID_RADIALIZ(fout);
            } else {
                PRINT_INDIVID_RADIALIZ("2", PRINT_DONE, X[1],
v[NUMBER_MATRIX], fout);
```

```
                player2 = true;
                flagIndRac = true;
                NUMBER_MATRIX = 1;
            }
            fout << std::endl;
        }

        // ------------------ run player3 ---------------------
        if (player1 && player2 && !player3) {
            // std::cout << "\n[run player3]";
            if (X[2] <= v[NUMBER_MATRIX]) {
                flagIndRac = false;
                PRINT_INDIVID_RADIALIZ("3", PRINT_NOT_DONE, nullptr, nullptr,
fout); // Условие для 3-го игрока не выполнено
                fout << std::endl;
                PRINT_NOT_DONE_INDIVID_RADIALIZ(fout);
            } else {
                PRINT_INDIVID_RADIALIZ("3", PRINT_DONE, X[2],
v[NUMBER_MATRIX], fout);
                player3 = true;
                flagIndRac = true;
                NUMBER_MATRIX = 1;
            }
            fout << std::endl;
        }

        // ------------------ run player4 ---------------------
        if (player1 && player2 && player3 && !player4) {
            // std::cout << "\n[run player4]";
            if (X[3] <= v[NUMBER_MATRIX]) {
                flagIndRac = false;
                PRINT_INDIVID_RADIALIZ("4", PRINT_NOT_DONE, nullptr, nullptr,
fout); // Условие для 4-го игрока не выполнено
                fout << std::endl;
                PRINT_NOT_DONE_INDIVID_RADIALIZ(fout);
            } else {
                PRINT_INDIVID_RADIALIZ("4", PRINT_DONE, X[3],
v[NUMBER_MATRIX], fout);
                player4 = true;
                flagIndRac = true;
                NUMBER_MATRIX = 1;
            }
            fout << std::endl;
        }

        NUMBER_MATRIX++;
    }



    (flagIndRac) ?
        PRINT_COMMENT("The condition of individual rationalization: DONE",
fout):
        PRINT_COMMENT("The condition of individual rationalization: NOT
DONE", fout);

    fout.close();
    fout1.close();
    fout2.close();

    _gameTheoryAndOperationsResearch_readfile();

    return 0;
}
```

```
}

int main(int argc, char* argv[])
{
    gameTheoryAndOperationsResearch::_main(argc, argv);
    return 0;
}
```

исходный код main.define.h:

```
#ifndef MAIN_DEFINE_H
#define MAIN_DEFINE_H

#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <fstream>
#include <locale.h>

#define gameTheoryAndOperationsResearch_filename
"gameTheoryAndOperationsResearch_l5.txt"
#define gameTheoryAndOperationsResearch_filename_superadditivity
"gameTheoryAndOperationsResearch_superadditivity_l5.txt"
#define gameTheoryAndOperationsResearch_filename_bulge
"gameTheoryAndOperationsResearch_bulge_l5.txt"

// Число различных множеств коолиций игроков
#define N_KOOL 16

/*
Массив значений характиристической функции
индекс массива в двоичном коде определяет состав коолиции
единица в разряде индекса определяет, что игрок с заданным номером,
входит в коолицию (игрок 1 соответсвует младшему разряду)
*/

int v[N_KOOL] =
{
    0,      // 0000 - пустое  множесто
    1,      // 0001 - { 1 }
    1,      // 0010 - { 2 }
    4,      // 0011 - { 1, 2 }
    3,      // 0100 - { 3 }
    4,      // 0101 - { 1, 3 }
    4,      // 0110 - { 2, 3 }
    9,      // 0111 - { 1, 2, 3 }
    3,      // 1000 - { 4 }
    6,      // 1001 - { 1, 4 }
    4,      // 1010 - { 2, 4 }
    9,      // 1011 - { 1, 2, 4 }
    7,      // 1100 - { 3, 4 }
    8,      // 1101 - { 1, 3, 4 }
    8,      // 1110 - { 2, 3, 4 }
    11,     // 1111 - { 1, 2, 3, 4 }
};


#endif // MAIN_DEFINE_H
```

исходный код main.func.h:

```
#ifndef MAIN_FUNC_H
#define MAIN_FUNC_H

#include "main.define.h"
namespace gameTheoryAndOperationsResearch {
    void _gameTheoryAndOperationsResearch_readfile() {
            std::cout << "\n[" <<
gameTheoryAndOperationsResearch_filename_superadditivity << "]" << std::endl;
            std::string line1;
            std::ifstream
in1(gameTheoryAndOperationsResearch_filename_superadditivity);
            while (std::getline(in1, line1))
                std::cout << line1 << std::endl;

            std::cout << "\n[" <<
gameTheoryAndOperationsResearch_filename_bulge << "]" << std::endl;
            std::string line2;
            std::ifstream
in2(gameTheoryAndOperationsResearch_filename_bulge);
            while (std::getline(in2, line2))
                std::cout << line2 << std::endl;

            std::cout << "\n[" << gameTheoryAndOperationsResearch_filename <<
"]" << std::endl;
            std::string line;
            std::ifstream in(gameTheoryAndOperationsResearch_filename);
            while (std::getline(in, line))
                std::cout << line << std::endl;

            line.clear();
            line1.clear();
            line2.clear();
    }

    /*
     * функция принимает index, представляющее четыре элемента множества
(1,2,3,4),
     * выводит номера элементов, соответствующих установленным битам в этом
числе
     */

    void gameTheoryAndOperationsResearch_print(int index, std::ofstream&
fout)
    {
        int mask = 1; // Инициализация битовой маски

        // Проверяем в цикле все возможные четыре элемента множества
        for (int i = 1; i <= 4; i++)
        {
            if (mask & index)           // Проверяем установку бита в 1
                fout << i << ' ';       // Печатаем номер элемента, если бит
установлен
            mask <<= 1;                 // Сдвигаем битовую маску на один бит
влево
        }
    }

    /*
     * определяет количество элементов в множестве index,
     * представленном целым двоичным числом (0,1) подсчетом установленных
битов
     */

    int gameTheoryAndOperationsResearch_length(int index)
```

```
    {
        int len = 0;         // Инициализация переменной для подсчета
количества элементов
        int maska = 1;       // Инициализация битовой маски

        // Проверяем в цикле все возможные четыре элемента множества
        for (int i = 1; i <= 4; i++)
        {
            if (maska & index)       // Проверяем установку бита в 1
                len++;               // Увеличиваем счетчик элементов
            maska <<= 1;             // Сдвигаем битовую маску на один бит
влево для проверки следующего элемента
        }
        return len; // Возвращаем количество элементов в множестве
    }

    /*
     * Функция для вычисления факториала числа n
     */

    int gameTheoryAndOperationsResearch_fact(int n)
    {
        if (n < 0)
            return 0;    // Если число отрицательное, возвращаем 0 (факториал
не определен для отрицательных чисел)
        if (n == 0)
            return 1;    // Факториал 0 равен 1

        int f = 1;       // Инициализация переменной для хранения результата
        for (int i = 2; i <= n; i++)
            f *= i;      // Вычисление факториала путем умножения на каждое
число от 2 до n
        return f;        // Возвращаем вычисленное значение факториала
    }

    // Проверка наличия игрока в множестве (коалиции)
    bool gameTheoryAndOperationsResearch_checking_presence_player_set(int
num/*номер игрока (1,2,3,4)*/, int set/*целое определяет множество*/)
    {
        int mask = 1; // Инициализация маски
        for (int i = 2; i <= num; i++)
            mask <<= 1; // Настройка маски на нужный бит

        if (set & mask) // Проверка, установлен ли нужный бит в множестве
            return true; // Если бит установлен, возвращаем true
        else
            return false; // Если бит не установлен, возвращаем false
    }


    // Удаление игрока из множества (коалиции), бит, соответствующий игроку,
устанавливаем в 0
    // Функция возвращает целое число, задающее полученное множество
    int gameTheoryAndOperationsResearch_delete_presence_player_set(int num
/*номер игрока (1,2,3,4)*/, int set /*целое определяет множество*/)
    {
        int mask = 1;
        for (int i = 2; i <= num; i++)
            mask <<= 1; // Настройка маски на нужный бит

        int mask2 = ~mask; // Побитовое инвертирование: нужный бит равен 0,
остальные равны 1
        return set & mask2 & 0xf; // Обнуляем требуемый бит, оставляем только
последние 4 бита в итоговом числе
```

```
    }

}
#endif // MAIN_FUNC_H
```