Міністерство освіти України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

ЗВІТ

до практикуму № 2

з дисципліни

“Інфомаційні технології підтримки прийняття рішень”

на тему: “ Оптимізація за бінарним відношенням”

Варіант 11

Виконала:

Студентка групи ІП-71

Каспрук Анастасія Андріївна

Київ 2021

1. Результати.

Завдання 1

Відно-шення	Клас, до якого належить БВ	Опт. альтернативиза принципом домінування	Опт. альтернативиза принципом блокування
R1	-	$X^*_R = \emptyset$. $X^{**}_R = \emptyset$.	$X^0_R = \{c, f\}$. $X^{00}_R = \{f\}$.
R2	квазіпорядку	$X^*_R = \{f\}$. $X^{**}_R = \emptyset$.	$X^0_R = \{c, f\}$. $X^{00}_R = \emptyset$.
R3	еквівалентності	$X^*_R = \emptyset$. $X^{**}_R = \emptyset$.	$X^0_R = \{a,b,c,d,e,f,g\}$. $X^{00}_R = \{e\}$.
R4	строгого порядку	$X^*_P = \{b\}$.	$X^0_P = \{b\}$.
R5	-	$X^*_R = \emptyset$. $X^{**}_R = \emptyset$.	$X^0_R = \{f\}$. $X^{00}_R = \emptyset$.
R6	строгого порядку	$X^*_P = \{b\}$.	$X^0_P = \{b\}$.
R7	нестрогого порядку	$X^*_R = \{b\}$. $X^{**}_R = \{b\}$.	$X^0_R = \{b\}$. $X^{00}_R = \{b\}$.
R8	строгого порядку	$X^*_P = \{c\}$.	$X^0_P = \{c\}$.

Завдання 2 (нумерація вершин у відношеннях починається з нуля - 0)

Відно-шення	ациклічне/ неациклічне	Розв'язок Неймана-Моргенштерна	Опт. альтернативиза принципом К-оптимізації
R1	-		1-max: {2, 4, 9, 11, 13} 1-opt : {2, 4, 9, 11, 13} 2-max : {2, 4, 13} 3-max : ∅ 4-max : {2, 4, 13}
R2	+	$X^{НМ}=\{0, 4, 10\}$	
R3	-		1-max: {5} 1-opt : {5} 2-max : {5} 3-max : {5} 4-max : {5}
R4	-		1-max: {6, 7, 9, 10, 13, 14} 1-opt : {6, 7, 9, 10, 13, 14} 2-max : {6, 7, 9, 10, 13, 14} 3-max : {6, 7, 9, 10, 13, 14} 4-max : {6, 7, 9, 10, 13, 14}
R5	-		1-max: ∅ 1-opt : ∅ 2-max : ∅ 3-max : ∅ 4-max : ∅
R6	+	$X^{НМ}=\{1, 2, 8\}$	
R7	+	$X^{НМ}=\{1, 3, 6, 8, 10\}$	

R8	-		1-max: ∅ 1-opt : ∅ 2-max : ∅ 3-max : ∅ 4-max : ∅
R9	+	X^{HM}={0, 4, 8}	
R10	+	X^{HM}={0, 1, 6, 12}	

2. Постановка задачі.

Завдання 1

Для кожного з бінарних відношень R1-R8 (із завдання 1 практикуму 1) визначити множину найкращих альтернатив за принципами домінування та блокування.

Завдання 2

На множині із 15 альтернатив задано 10 бінарних відношень R1-R10 матрицями відношень. Для кожного із БВ Ri необхідно:

1) перевірити наявність властивості ациклічності;
2) а) якщо відношення Ri є ациклічним - знайти множину Неймана- Моргенштерна (отриманий результат обгрунтувати – показати, що отримана множина відповідає означенню розв'язка Неймана- Моргенштерна);
   б) якщо відношення Ri не ациклічне - знайти множини оптимальних альтернатив за принципом К-оптимізації (k=1, k=2, k=3, k=4).

3. Розв'язок.

Завдання 1

№1 ----------

	A	B	c	d	E	f
a	1	1	1	0	1	0
b	0	1	1	0	1	0
c	1	1	0	0	1	0
d	0	0	0	0	1	0
e	0	0	1	1	0	0
f	1	0	0	1	0	1

	a	B	c	d	e	f
a	I	P	I	N	P	0
b	0	I	I	N	P	N
c	I	I	N	N	I	N
d	N	N	N	N	I	0
e	0	0	I	I	N	N
f	P	N	N	P	N	I

Оптимізація за домінуванням:

I ≠ ∅, тому шукаємо X^{*}_{R} та X^{**}_{R}.
X^{*}_{R} = ∅, оскільки немає рядка зі всіма одиницями.
X^{**}_{R} = ∅, оскільки немає рядка зі всіма одиницями, X^{*}_{R} = ∅.

Оптимізація за блокуванням:

I ≠ ∅, тому шукаємо X^{0}_{R} та X^{00}_{R}.
X^{0}_{R} = {c, f }, оскільки у стовпцях c, f присутні тільки нулі та I.
X^{00}_{R} = { f }, оскільки у стовпці f всі нулі, окрім клітинки, що характеризує пару (f, f)

№2 ----------

	A	B	c	D	E	f
a	1	1	0	1	0	0
b	1	1	0	1	0	0
c	1	1	1	1	1	1
d	1	1	0	1	0	0
e	1	1	0	1	1	0
f	1	1	1	1	1	1

	a	B	c	d	e	F
a	I	I	0	I	0	0
b	I	I	0	I	0	0
c	P	P	I	P	P	I
d	I	I	0	I	0	0
e	P	P	0	P	I	0
f	P	P	I	P	P	I

Оптимізація за домінуванням:

I ≠ ∅, тому шукаємо X^{*}_{R} та X^{**}_{R}.
X^{*}_{R} = {f }, оскільки у рядку f всі одиниці.
X^{**}_{R} = ∅, оскільки у клітинці, що характеризує пару (c, f) стоїть одиниця.

Оптимізація за блокуванням:

I ≠ ∅, тому шукаємо X^{0}_{R} та X^{00}_{R}.
X^{0}_{R} = {c, f }, оскільки у стовпцях c, f присутні тільки нулі та I.
X^{00}_{R} = ∅, оскільки між (c, f) існує двонаправлений зв'язок.

№3 ----------

	A	**B**	**c**	**D**	**E**	**f**
a	1	1	0	0	0	0
b	1	1	0	0	0	0
c	0	0	1	1	0	1
d	0	0	1	1	0	1
e	0	0	0	0	1	0
f	0	0	1	1	0	1

	a	**b**	**c**	**d**	**e**	**F**
a	I	I	N	N	N	N
b	I	I	N	N	N	N
c	N	N	I	I	N	I
d	N	N	I	I	N	I
e	N	N	N	N	I	N
f	N	N	I	I	N	I

Оптимізація за домінуванням:

I ≠ ∅, тому шукаємо X^{*}_{R} та X^{**}_{R}.
X^{*}_{R} = ∅, оскільки немає рядка зі всіма одиницями.
X^{**}_{R} = ∅, оскільки немає рядка зі всіма одиницями, X^{*}_{R} = ∅.

Оптимізація за блокуванням:

I ≠ ∅, тому шукаємо X^{0}_{R} та X^{00}_{R}.
X^{0}_{R} = {a, b, c, d, e, f, g }, оскільки у стовпцях a,b,c,d,e,f,g присутні тільки нулі та I.
X^{00}_{R} = {e}, оскільки у стовпці e всі нулі, окрім клітинки, що характеризує пару (e, e)

№4 ----------

	A	**B**	**c**	**d**	**E**	**f**
a	0	0	1	1	0	1
b	1	0	1	1	1	1
c	0	0	0	0	0	0
d	0	0	0	0	0	0
e	0	0	1	1	0	1
f	0	0	0	0	0	0

	a	**B**	**c**	**d**	**e**	**F**
a	N	0	P	P	N	P
b	P	N	P	P	P	P
c	0	0	N	N	0	N
d	0	0	N	N	0	N
e	N	0	P	P	N	P
F	0	0	N	N	0	N

Оптимізація за домінуванням:

I = ∅, тому шукаємо X^{*}_{P}.
X^{*}_{P} ={b}, оскільки у рядку b всі одиниці, окрім клітинки, що характеризує пару (b,b).

Оптимізація за блокуванням:

I = ∅, тому шукаємо X^{0}_{P}.
X^{0}_{P} = {b}, оскільки у стовпці b всі нулі.

№5 ----------

	a	b	c	d	e	f
a	0	1	1	0	0	0
b	0	1	0	0	0	1
c	0	1	1	0	0	1
d	0	1	1	0	0	0
e	1	0	0	0	0	0
f	0	1	1	1	1	0

	a	b	c	d	e	F
A	N	P	P	N	0	N
B	0	I	0	0	N	I
C	0	P	I	0	N	I
D	N	P	P	N	N	0
E	P	N	N	N	N	0
F	N	I	I	P	P	N

Оптимізація за домінуванням:

I ≠ ∅, тому шукаємо X^{*}_{R} та X^{**}_{R}.
X^{*}_{R} = ∅, оскільки немає рядка зі всіма одиницями.
X^{**}_{R} = ∅, оскільки немає рядка зі всіма одиницями, X^{*}_{R} = ∅.

Оптимізація за блокуванням:

I ≠ ∅, тому шукаємо X^{0}_{R} та X^{00}_{R}.
X^{0}_{R} = { f }, оскільки у стовпці f присутні тільки нулі та I.
X^{00}_{R} = ∅, оскільки між (b, f), (c, f) існують двонаправлені зв'язки, а між (f, f) зв'язку нема.

№6 ----------

	A	B	c	d	e	f
A	0	0	0	0	0	0
B	1	0	1	1	1	1
C	1	0	0	0	0	0
D	1	0	1	0	1	1
E	1	0	1	0	0	0
F	1	0	1	0	1	0

	a	B	c	d	e	F
A	N	0	0	0	0	0
B	P	N	P	P	P	P
C	P	0	N	0	0	0
D	P	0	P	N	P	P
E	P	0	P	0	N	0
F	P	0	P	0	P	N

Оптимізація за домінуванням:

I = ∅, тому шукаємо X^{*}_{P}.
X^{*}_{P} ={b}, оскільки у рядку b всі одиниці, окрім клітинки, що характеризує пару (b,b).

Оптимізація за блокуванням:

I = ∅, тому шукаємо X^{0}_{P}.
X^{0}_{P} = {b}, оскільки у стовпці b всі нулі.

№7 ----------

	A	B	c	d	e	f
A	1	0	0	0	0	1
B	1	1	1	1	1	1
C	1	0	1	1	1	1
D	1	0	0	1	0	1
E	1	0	0	1	1	1
F	0	0	0	0	0	1

	a	b	c	d	e	F
a	I	0	0	0	0	P
b	P	I	P	P	P	P
c	P	0	I	P	P	P
d	P	0	0	I	0	P
e	P	0	0	P	I	P
f	0	0	0	0	0	I

Оптимізація за домінуванням:

I ≠ ∅, тому шукаємо X^{*}_{R} та X^{**}_{R}.
X^{*}_{R} = {b}, оскільки у рядку b всі одиниці.
X^{**}_{R} = {b}, оскільки у рядку b всі одиниці, а у стовпці b всі нулі, окрім клітинки, що характеризує пару (b, b).

Оптимізація за блокуванням:

I ≠ ∅, тому шукаємо X^{0}_{R} та X^{00}_{R}.
X^{0}_{R} = { b }, оскільки у стовпці b присутні тільки нулі та I.
X^{00}_{R} ={b}, оскільки у стовпці b всі нулі, окрім клітинки, що характеризує пару (b, b)

№8 ----------

	A	b	c	d	e	f
a	0	0	0	0	0	0
b	0	0	0	0	0	0
c	1	1	0	1	1	1
d	1	1	0	0	1	0
e	0	0	0	0	0	0
f	1	1	0	0	1	0

	a	b	c	d	e	F
a	N	N	0	0	N	0
b	N	N	0	0	N	0
c	P	P	N	P	P	P
d	P	P	0	N	P	N
e	N	N	0	0	N	0
f	P	P	0	N	P	N

Оптимізація за домінуванням:

I = ∅, тому шукаємо X^{*}_{P}.
X^{*}_{P} ={c}, оскільки у рядку c всі одиниці, окрім клітинки, що характеризує пару (c,c).

Оптимізація за блокуванням:

I = ∅, тому шукаємо X^{0}_{P}.
X^{0}_{P} = {c}, оскільки у стовпці c всі нулі.

Завдання 2

Для пошуку циклу в орієнтованому графі найчастіше використовують обходи графа в глибину (DFS - *depth-first search*) або в ширину (BFS - *breadth-first search*). У лекції був наведений приклад з обходом графа в ширину, тому в даній лабораторній роботі для перевірки графа на ациклічність використовується BFS.

Щодо алгоритмічної складності, то обидва алгоритми мають однакову складність O(|V| + |E|), де |V| - число вершин, |E| - число ребер.

Опис алгоритму для пошуку циклу з використанням BFS

Повторити для всіх вершин:

1. Помістити всі сусідні для цільової та ще не пройдені вершини у порожню чергу.
2. Витягти з початку черги вершину ***u***.
    - Якщо вершина ***u*** є цільовою вершиною, то завершити пошук з результатом «цикл знайдено».
    - В іншому випадку, в кінець черги додаються всі ще не пройдені наступники вершини ***u***.
3. Якщо черга порожня, то всі вершини орієнтованого графа були переглянуті, отже, цільова вершина недосяжна і циклу для неї не знайдено.
4. Якщо черга не порожня, повернутися до п. 2.

Якщо після перевірки відношення було ідентифіковано як ациклічне, то для знаходження множини найкращих альтернатив застосовувався алгоритм Неймана-Моргенштерна. Якщо ж цикл був знайдений, пошук множини найкращих альтернатив здійснювався за алгоритмом К-оптимізації.

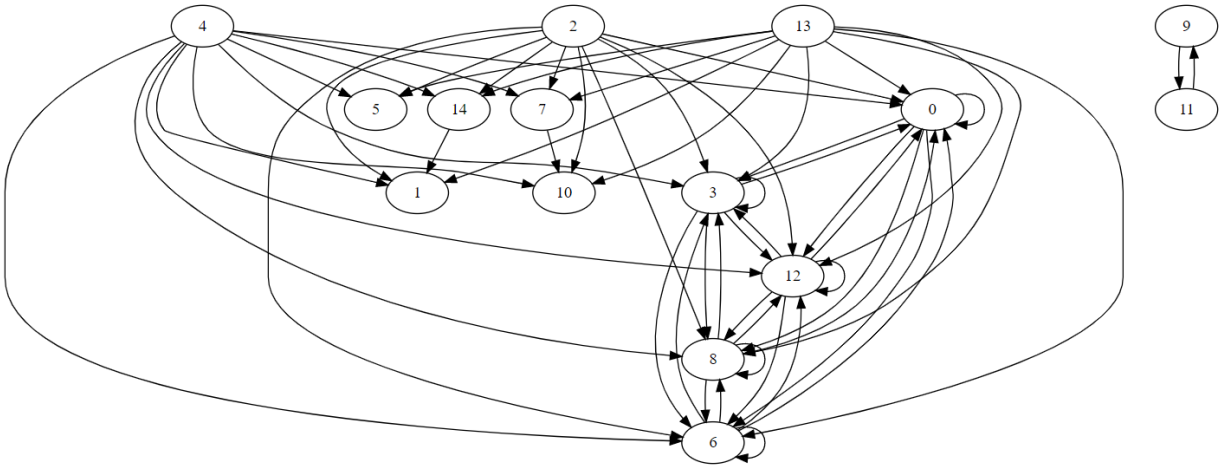
Робота алгоритму:

```
R1-----------------------------------------------
   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
0  1  0  0  1  0  0  1  0  1  0  0  0  1  0  0
1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
2  1  1  0  1  0  1  1  1  1  0  1  0  1  0  1
3  1  0  0  1  0  0  1  0  1  0  0  0  1  0  0
4  1  1  0  1  0  1  1  1  1  0  1  0  1  0  1
5  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
6  1  0  0  1  0  0  1  0  1  0  0  0  1  0  0
7  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0
8  1  0  0  1  0  0  1  0  1  0  0  0  1  0  0
9  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
10 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
11 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0
12 1  0  0  1  0  0  1  0  1  0  0  0  1  0  0
13 1  1  0  1  0  1  1  1  1  0  1  0  1  0  1
14 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0

   0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
0  I  N  0  I  0  N  I  N  I  N  N  N  I  0  N
1  N  N  0  N  0  N  N  N  N  N  N  N  N  0  0
2  P  P  N  P  N  P  P  P  P  N  P  N  P  N  P
3  I  N  0  I  0  N  I  N  I  N  N  N  I  0  N
4  P  P  N  P  N  P  P  P  P  N  P  N  P  N  P
5  N  N  0  N  0  N  N  N  N  N  N  N  N  0  N
6  I  N  0  I  0  N  I  N  I  N  N  N  I  0  N
7  N  N  0  N  0  N  N  N  N  N  P  N  N  0  N
8  I  N  0  I  0  N  I  N  I  N  N  N  I  0  N
9  N  N  N  N  N  N  N  N  N  N  N  I  N  N  N
10 N  N  0  N  0  N  N  0  N  N  N  N  N  0  N
11 N  N  N  N  N  N  N  N  N  I  N  N  N  N  N
12 I  N  0  I  0  N  I  N  I  N  N  N  I  0  N
13 P  P  N  P  N  P  P  P  P  N  P  N  P  N  P
14 N  P  0  N  0  N  N  N  N  N  N  N  N  0  N

Є ациклічним: False
Опт. альтернативи за принципом К - оптимізації
К = 1: 2 4 9 11 13
К = 1 (opt): 2 4 9 11 13
К = 2: 2 4 13
К = 3:
К = 4: 2 4 13
```

Граф:

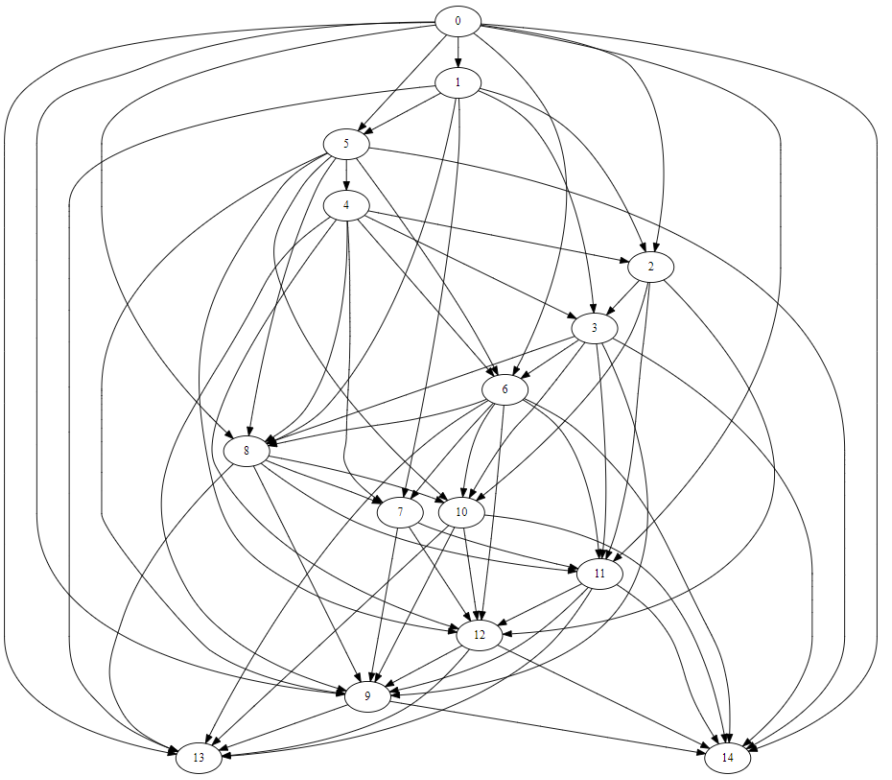


```
R2----------------------------------------
    0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0   0  1  1  0  0  1  1  0  1  1   0   1   0   1   1
1   0  0  1  1  0  1  0  1  1  0   0   0   0   1   0
2   0  0  0  1  0  0  0  0  0  0   1   1   1   0   0
3   0  0  0  0  0  0  1  0  1  1   1   1   0   0   1
4   0  0  1  1  0  0  1  1  1  1   0   0   1   0   0
5   0  0  0  0  1  0  1  0  1  1   1   0   1   0   1
6   0  0  0  0  0  0  0  1  1  0   1   1   1   1   1
7   0  0  0  0  0  0  0  0  0  1   0   1   1   0   0
8   0  0  0  0  0  0  0  1  0  1   1   1   0   1   0
9   0  0  0  0  0  0  0  0  0  0   0   0   0   1   1
10  0  0  0  0  0  0  0  0  0  1   0   0   1   1   1
11  0  0  0  0  0  0  0  0  0  1   0   0   1   1   1
12  0  0  0  0  0  0  0  0  0  1   0   0   0   1   1
13  0  0  0  0  0  0  0  0  0  0   0   0   0   0   0
14  0  0  0  0  0  0  0  0  0  0   0   0   0   0   0

    0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0   N  P  P  N  N  P  P  N  P  P   N   P   N   P   P
1   0  N  P  P  N  P  N  P  P  N   N   N   N   P   N
2   0  0  N  P  0  N  N  N  N  N   P   P   P   N   N
3   N  0  0  N  0  N  P  N  P  P   P   P   N   N   P
4   N  N  P  P  N  0  P  P  P  P   N   N   P   N   N
5   0  0  N  N  P  N  P  N  P  P   P   N   P   N   P
6   0  N  N  0  0  0  N  P  P  N   P   P   P   P   P
7   N  0  N  N  0  N  0  N  0  P   N   P   P   N   N
8   0  0  N  0  0  0  0  P  N  P   P   P   N   P   N
9   0  N  N  0  0  0  N  0  0  N   0   0   0   P   P
10  N  N  0  0  N  0  0  N  0  P   N   N   P   P   P
11  0  N  0  0  N  N  0  0  0  P   N   N   P   P   P
12  N  N  0  N  0  0  0  0  N  P   0   0   N   P   P
13  0  0  N  N  N  N  0  N  0  0   0   0   0   N   N
14  0  N  N  0  N  0  0  N  N  0   0   0   0   N   N

Є ациклічним: True
Розв'язок Неймана-Моргенштерна: 0 4 10
Чи є розв'язок внутрішньо стійким: True
Чи є розв'язок зовнішньо стійким: True
```

Граф:

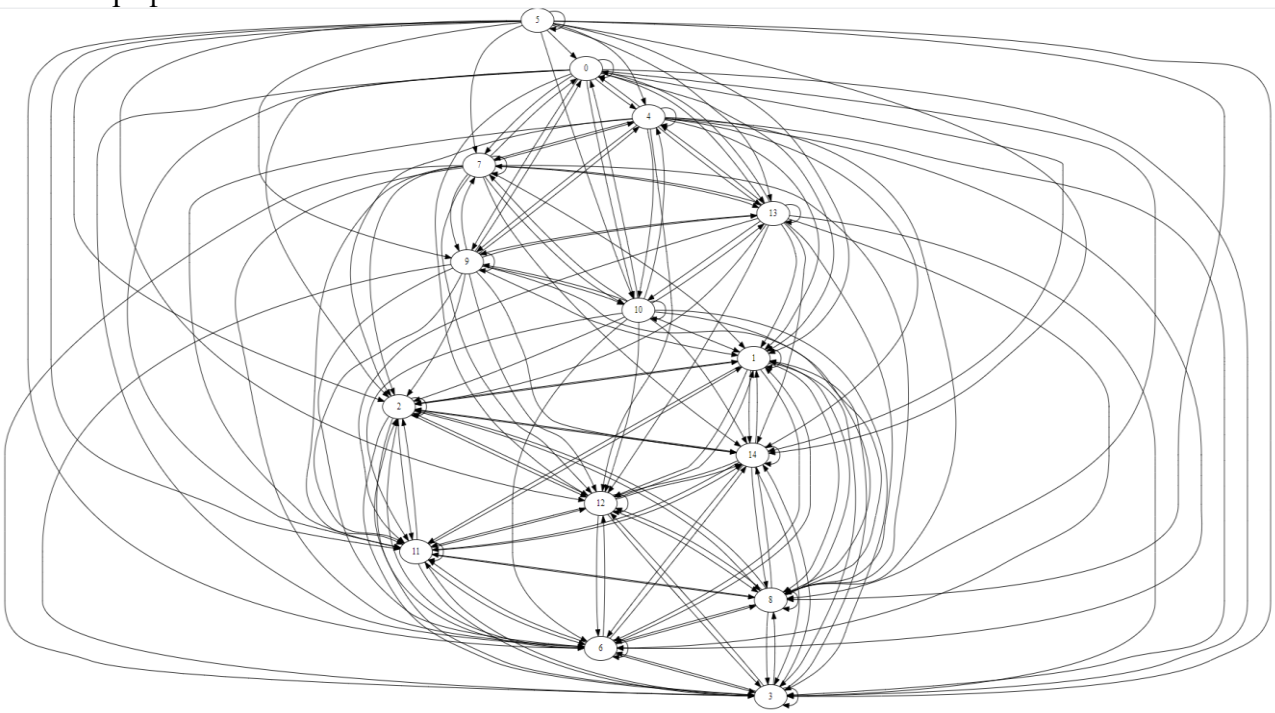


```
R3----------------------------------------------------
    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0   1  1  1  1  1  0  1  1  1  1   1  1  1  1  1
1   0  1  1  1  0  0  1  0  1  0   0  1  1  0  1
2   0  1  1  1  0  0  1  0  1  0   0  1  1  0  1
3   0  1  1  1  0  0  1  0  1  0   0  1  1  0  1
4   1  1  1  1  1  0  1  1  1  1   1  1  1  1  1
5   1  1  1  1  1  1  1  1  1  1   1  1  1  1  1
6   0  1  1  1  0  0  1  0  1  0   0  1  1  0  1
7   1  1  1  1  1  0  1  1  1  1   1  1  1  1  1
8   0  1  1  1  0  0  1  0  1  0   0  1  1  0  1
9   1  1  1  1  1  0  1  1  1  1   1  1  1  1  1
10  1  1  1  1  1  0  1  1  1  1   1  1  1  1  1
11  0  1  1  1  0  0  1  0  1  0   0  1  1  0  1
12  0  1  1  1  0  0  1  0  1  0   0  1  1  0  1
13  1  1  1  1  1  0  1  1  1  1   1  1  1  1  1
14  0  1  1  1  0  0  1  0  1  0   0  1  1  0  1

    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0   I  P  P  P  I  0  P  I  P  I   I  P  P  I  P
1   0  I  I  I  0  0  I  0  I  0   0  I  I  0  I
2   0  I  I  I  0  0  I  0  I  0   0  I  I  0  I
3   0  I  I  I  0  0  I  0  I  0   0  I  I  0  I
4   I  P  P  P  I  0  P  I  P  I   I  P  P  I  P
5   P  P  P  P  P  I  P  P  P  P   P  P  P  P  P
6   0  I  I  I  0  0  I  0  I  0   0  I  I  0  I
7   I  P  P  P  I  0  P  I  P  I   I  P  P  I  P
8   0  I  I  I  0  0  I  0  I  0   0  I  I  0  I
9   I  P  P  P  I  0  P  I  P  I   I  P  P  I  P
10  I  P  P  P  I  0  P  I  P  I   I  P  P  I  P
11  0  I  I  I  0  0  I  0  I  0   0  I  I  0  I
12  0  I  I  I  0  0  I  0  I  0   0  I  I  0  I
13  I  P  P  P  I  0  P  I  P  I   I  P  P  I  P
14  0  I  I  I  0  0  I  0  I  0   0  I  I  0  I

Є ациклічним: False
Опт. альтернативи за принципом К - оптимізації
К = 1: 5
К = 1 (opt): 5
К = 2: 5
К = 3: 5
К = 4: 5
```

Граф:

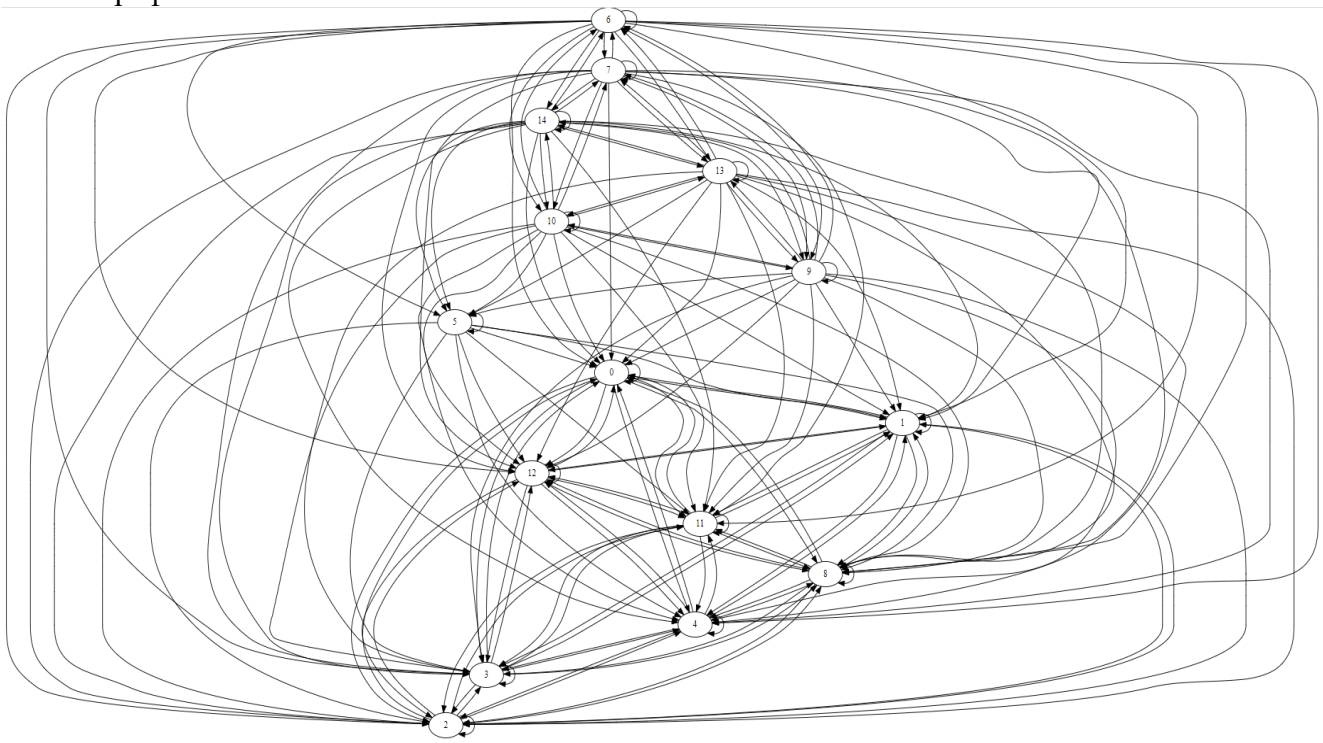


```
R4----------------------------------------------------
   0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0  1  1  1  1  1  0  0  0  1  0   0   1   1   0   0
1  1  1  1  1  1  0  0  0  1  0   0   1   1   0   0
2  1  1  1  1  1  0  0  0  1  0   0   1   1   0   0
3  1  1  1  1  1  0  0  0  1  0   0   1   1   0   0
4  1  1  1  1  1  0  0  0  1  0   0   1   1   0   0
5  1  1  1  1  1  1  0  0  1  0   0   1   1   0   0
6  1  1  1  1  1  1  1  1  1  1   1   1   1   1   1
7  1  1  1  1  1  1  1  1  1  1   1   1   1   1   1
8  1  1  1  1  1  0  0  0  1  0   0   1   1   0   0
9  1  1  1  1  1  1  1  1  1  1   1   1   1   1   1
10 1  1  1  1  1  1  1  1  1  1   1   1   1   1   1
11 1  1  1  1  1  0  0  0  1  0   0   1   1   0   0
12 1  1  1  1  1  0  0  0  1  0   0   1   1   0   0
13 1  1  1  1  1  1  1  1  1  1   1   1   1   1   1
14 1  1  1  1  1  1  1  1  1  1   1   1   1   1   1

   0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0  I  I  I  I  I  0  0  0  I  0   0   I   I   0   0
1  I  I  I  I  I  0  0  0  I  0   0   I   I   0   0
2  I  I  I  I  I  0  0  0  I  0   0   I   I   0   0
3  I  I  I  I  I  0  0  0  I  0   0   I   I   0   0
4  I  I  I  I  I  0  0  0  I  0   0   I   I   0   0
5  P  P  P  P  P  I  0  0  P  0   0   P   P   0   0
6  P  P  P  P  P  P  I  I  P  I   I   P   P   I   I
7  P  P  P  P  P  P  I  I  P  I   I   P   P   I   I
8  I  I  I  I  I  0  0  0  I  0   0   I   I   0   0
9  P  P  P  P  P  P  I  I  P  I   I   P   P   I   I
10 P  P  P  P  P  P  I  I  P  I   I   P   P   I   I
11 I  I  I  I  I  0  0  0  I  0   0   I   I   0   0
12 I  I  I  I  I  0  0  0  I  0   0   I   I   0   0
13 P  P  P  P  P  P  I  I  P  I   I   P   P   I   I
14 P  P  P  P  P  P  I  I  P  I   I   P   P   I   I

Є ациклічним: False
Опт. альтернативи за принципом K - оптимізації
K = 1: 6 7 9 10 13 14
K = 1 (opt): 6 7 9 10 13 14
K = 2: 6 7 9 10 13 14
K = 3: 6 7 9 10 13 14
K = 4: 6 7 9 10 13 14
```

Граф:

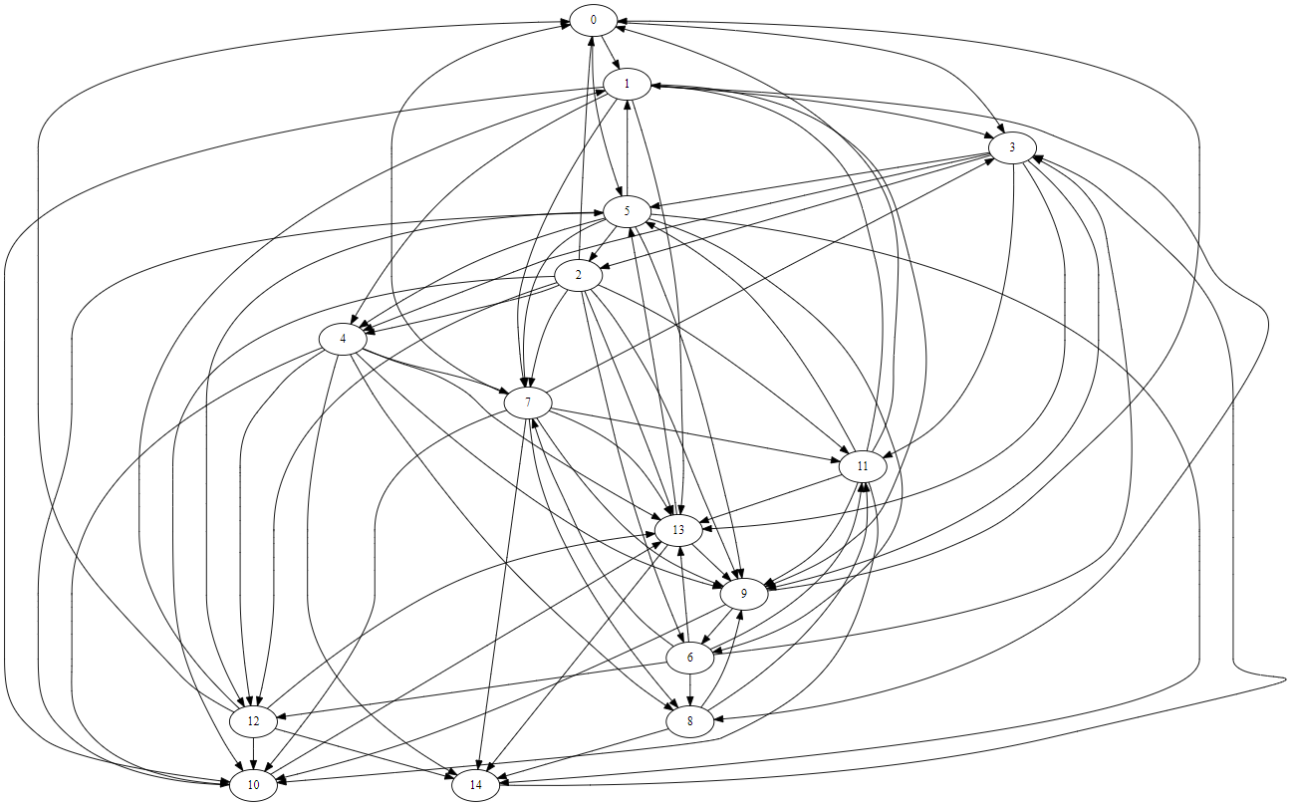


```
R5----------------------------------------------
   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0  0 1 0 1 0 1 0 0 0 0  0  0  0  0  0
1  0 0 0 1 1 0 0 1 1 1  1  0  0  1  0
2  1 0 0 0 1 0 1 1 0 1  1  1  1  1  0
3  0 0 1 0 1 1 0 0 0 1  0  1  0  1  0
4  0 0 0 0 0 0 0 1 1 1  1  0  1  1  1
5  0 1 1 0 1 0 1 1 0 1  0  0  1  0  1
6  0 0 0 1 0 0 0 1 1 0  0  1  1  1  0
7  1 0 0 1 0 0 0 0 1 1  1  1  0  1  1
8  0 0 0 0 0 0 0 0 0 1  0  1  0  0  1
9  1 0 0 0 0 0 0 1 0 0  0  1  0  0  0
10 0 0 0 0 0 0 1 0 0 0  0  0  0  1  0
11 1 1 0 0 0 1 0 0 0 1  1  0  0  1  0
12 1 1 0 0 0 0 0 0 0 0  1  0  0  1  1
13 0 0 0 0 0 0 1 0 0 0  1  0  0  0  1
14 0 0 0 1 0 0 0 0 0 0  0  0  0  0  0

   0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
0  N P 0 P N P N 0 N 0  N  0  0  N  N
1  0 N N P P 0 N P P P  P  0  0  P  N
2  P N N 0 P 0 P P N P  P  P  P  P  N
3  0 0 P N P P 0 0 N P  N  P  N  P  0
4  N 0 0 0 N 0 N P P P  P  N  P  P  P
5  0 P P 0 P N P P N P  0  0  P  0  P
6  N N 0 P N 0 N P P 0  N  P  P  P  N
7  P 0 0 P 0 0 0 N P P  P  P  N  P  P
8  N 0 N N 0 N 0 0 N P  N  P  N  N  P
9  P 0 0 0 0 0 P 0 0 N  P  0  N  0  N
10 N 0 0 N 0 P N 0 N 0  N  0  0  P  N
11 P P 0 0 N P 0 0 0 P  P  N  N  P  N
12 P P 0 N 0 0 0 N N N  P  N  N  P  P
13 N 0 0 0 0 P 0 0 N P  0  0  0  N  P
14 N N N P 0 0 N 0 0 N  N  N  0  0  N

Є ациклічним: False
Опт. альтернативи за принципом К - оптимізації
К = 1:
К = 1 (opt):
К = 2:
К = 3:
К = 4:
```

Граф:

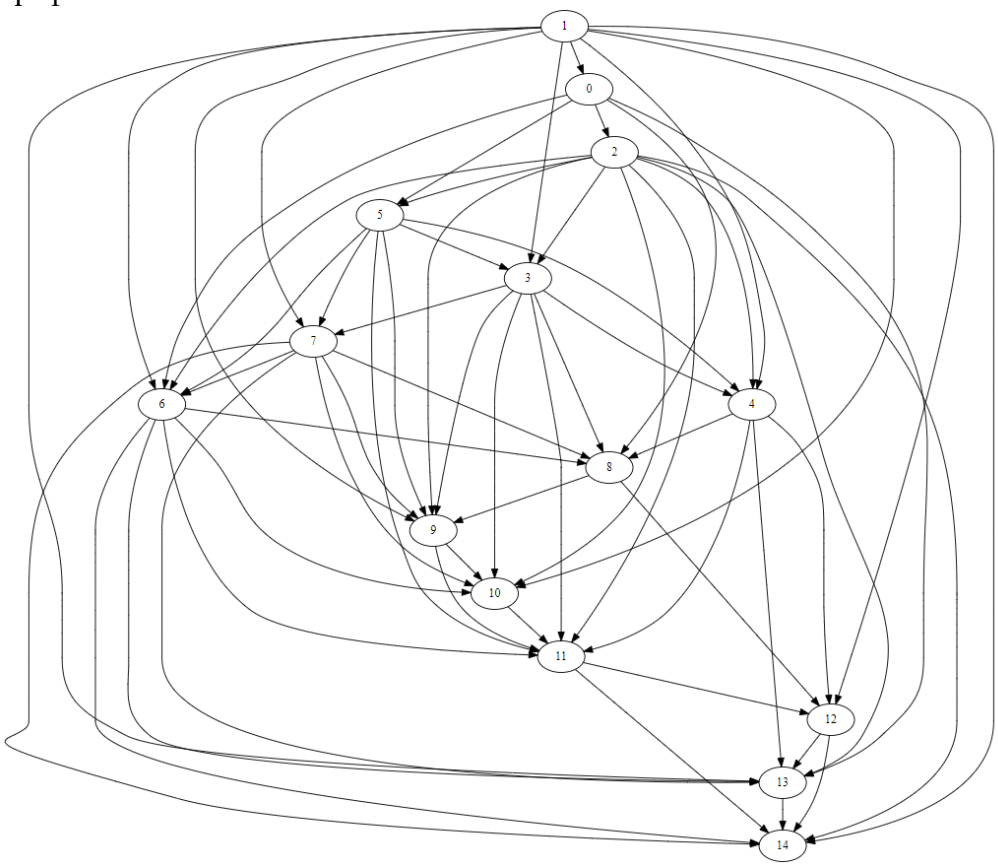


```
R6----------------------------------
    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0   0  0  1  0  0  1  1  0  1  0  0  0  0  1  0
1   1  0  0  1  1  0  1  1  0  1  1  0  1  1  1
2   0  0  0  1  1  1  1  0  0  1  1  1  0  1  1
3   0  0  0  0  1  0  0  1  1  1  1  1  0  0  0
4   0  0  0  0  0  0  0  0  1  0  0  1  1  1  0
5   0  0  0  1  1  0  1  1  0  1  0  1  0  0  0
6   0  0  0  0  0  0  0  0  1  0  1  1  0  1  1
7   0  0  0  0  0  0  1  0  1  1  1  0  0  1  1
8   0  0  0  0  0  0  0  0  0  1  0  0  1  0  0
9   0  0  0  0  0  0  0  0  0  0  1  1  0  0  0
10  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0
11  0  0  0  0  0  0  0  0  0  0  0  0  1  0  1
12  0  0  0  0  0  0  0  0  0  0  0  0  0  1  1
13  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
14  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0

    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0   N  0  P  N  N  P  P  N  P  N  N  N  N  P  N
1   P  N  N  P  P  N  P  P  N  P  P  N  P  P  P
2   0  N  N  P  P  P  P  N  N  P  P  P  N  P  P
3   N  0  0  N  P  0  N  P  P  P  P  P  N  N  N
4   N  0  0  0  N  0  N  N  P  N  N  P  P  P  N
5   0  N  0  P  P  N  P  P  N  P  N  P  N  N  N
6   0  0  0  N  N  0  N  0  P  N  P  P  N  P  P
7   N  0  N  0  N  0  P  N  P  P  P  N  N  P  P
8   0  N  N  0  0  N  0  0  N  P  N  N  P  N  N
9   N  0  0  0  N  0  N  0  0  N  P  P  N  N  N
10  N  0  0  0  N  N  0  0  N  0  N  P  N  N  N
11  N  N  0  0  0  0  0  N  N  0  0  N  P  N  P
12  N  0  N  N  0  N  N  N  0  N  N  0  N  P  P
13  0  0  0  N  0  N  0  0  N  N  N  N  0  N  P
14  N  0  0  N  N  N  0  0  N  N  N  0  0  0  N

Є ациклічним: True
Розв'язок Неймана-Моргенштерна: 1 2 8
Чи є розв'язок внутрішньо стійким: True
Чи є розв'язок зовнішньо стійким: True
```

Граф:

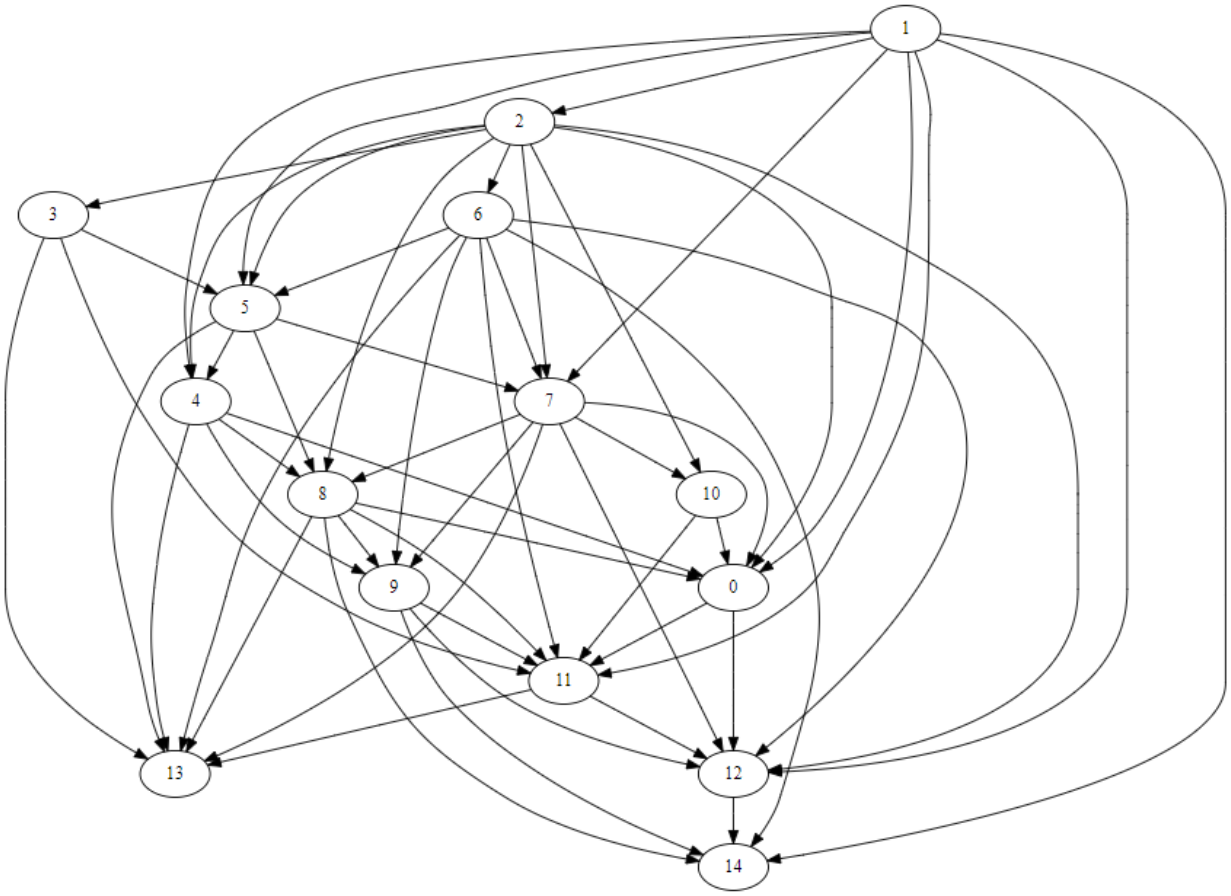


```
R7----------------------------------------------
    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0   0  0  0  0  0  0  0  0  0  0   0  1  1  0  0
1   1  0  1  0  1  1  0  1  0  0   0  1  1  0  1
2   1  0  0  1  1  1  1  1  1  0   1  0  1  0  0
3   0  0  0  0  0  1  0  0  0  0   0  1  0  1  0
4   1  0  0  0  0  0  0  0  0  1   1  0  0  0  1  0
5   0  0  0  0  1  0  0  1  1  0   0  0  0  0  1  0
6   0  0  0  0  0  1  0  1  0  1   0  1  1  1  1
7   1  0  0  0  0  0  0  0  0  1   1  1  0  1  1  0
8   1  0  0  0  0  0  0  0  0  0   1  0  1  0  1  1
9   0  0  0  0  0  0  0  0  0  0   0  1  1  0  1  1
10  1  0  0  0  0  0  0  0  0  0   0  1  0  0  0  0
11  0  0  0  0  0  0  0  0  0  0   0  0  1  1  0  0
12  0  0  0  0  0  0  0  0  0  0   0  0  0  0  1  1
13  0  0  0  0  0  0  0  0  0  0   0  0  0  0  0  0
14  0  0  0  0  0  0  0  0  0  0   0  0  0  0  0  0

    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0   N  0  0  N  0  N  N  0  0  N   0  P  P  N  N
1   P  N  P  N  P  P  N  P  N  N   N  P  P  N  P
2   P  0  N  P  P  P  P  P  P  N   P  N  P  N  N
3   N  N  0  N  N  P  N  N  N  N   N  P  N  P  N
4   P  0  0  N  N  0  N  N  P  P   N  N  N  P  N
5   N  0  0  0  P  N  0  P  P  N   N  N  N  P  N
6   N  N  0  N  N  P  N  P  N  P   N  P  P  P  P
7   P  0  0  N  N  0  0  N  P  P   P  N  P  P  N
8   P  N  0  N  0  0  N  0  N  P   N  P  N  P  P
9   N  N  N  N  0  N  0  0  0  N   N  P  P  N  P
10  P  N  0  N  N  N  N  0  N  N   N  P  N  N  N
11  0  0  N  0  N  N  0  N  0  0   0  N  P  P  N
12  0  0  0  N  N  N  0  0  N  0   N  0  N  N  P
13  N  N  N  0  0  0  0  0  0  N   N  0  N  N  N
14  N  0  N  N  N  N  0  N  0  0   N  N  0  N  N

Є ациклічним: True
Розв'язок Неймана-Моргенштерна: 1 3 6 8 10
Чи є розв'язок внутрішньо стійким: True
Чи є розв'язок зовнішньо стійким: True
```

Граф:



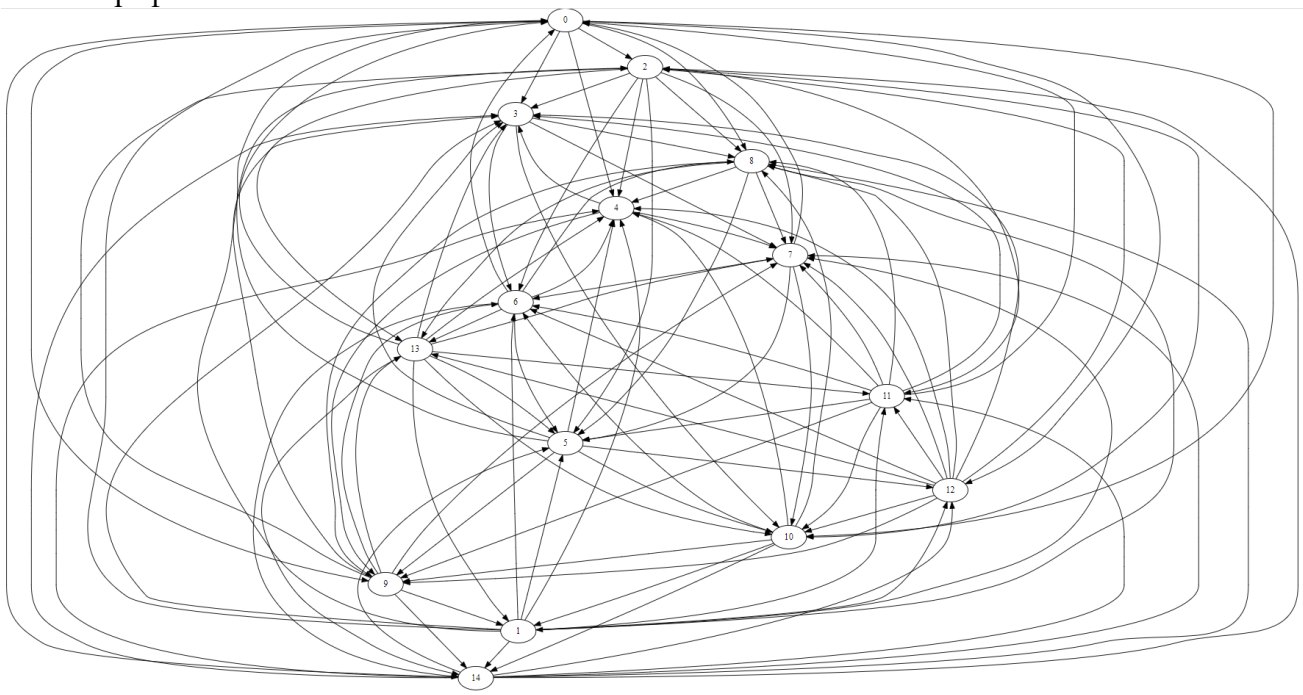
R8----------

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	1	1	1	0	0	0	1	1	1	0	1	0	1
1	1	0	1	1	1	1	1	1	0	0	0	1	1	0	1
2	0	0	0	1	1	1	1	1	1	0	1	1	0	1	0
3	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0
4	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	1	1	0	1	0	0
6	1	0	0	0	1	1	0	0	1	0	0	0	0	1	1
7	1	0	0	0	0	1	1	0	0	0	1	0	0	0	0
8	0	1	0	0	1	1	0	1	0	1	0	0	0	1	0
9	0	1	1	1	0	0	1	1	0	0	0	0	0	1	1
10	0	1	0	0	1	0	1	0	1	1	0	0	0	0	1
11	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0
12	0	0	1	1	1	0	1	1	1	1	1	1	0	1	0
13	1	1	0	1	1	1	0	1	0	0	1	1	0	0	1
14	0	0	1	1	1	1	0	1	1	0	0	1	1	0	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	N	0	P	P	P	0	0	0	P	P	P	0	P	0	P
1	P	N	P	P	P	P	P	P	0	0	0	P	P	0	P
2	0	0	N	P	P	P	P	P	P	0	P	P	0	P	0
3	0	0	0	N	0	0	P	P	P	0	P	0	0	0	0
4	0	0	0	P	N	0	0	P	0	P	0	0	0	0	0
5	P	0	0	P	P	N	0	0	0	P	P	0	P	0	0
6	P	0	0	0	P	P	N	0	P	0	0	0	0	P	P
7	P	0	0	0	0	P	P	N	0	0	P	0	0	0	0
8	0	P	0	0	P	P	0	P	N	P	0	0	0	P	0
9	0	P	P	P	0	0	P	P	0	N	0	0	0	P	P
10	0	P	0	0	P	0	P	0	P	P	N	0	0	0	P
11	P	0	0	P	P	P	P	P	P	P	P	N	0	0	0
12	0	0	P	P	P	0	P	P	P	P	P	P	N	P	0
13	P	P	0	P	P	P	0	P	0	0	P	P	0	N	P
14	0	0	P	P	P	P	0	P	P	0	0	P	P	0	N

```
Є ациклічним: False
Опт. альтернативи за принципом К - оптимізації
К = 1:
К = 1 (opt):
К = 2:
К = 3:
К = 4:
```

Граф:

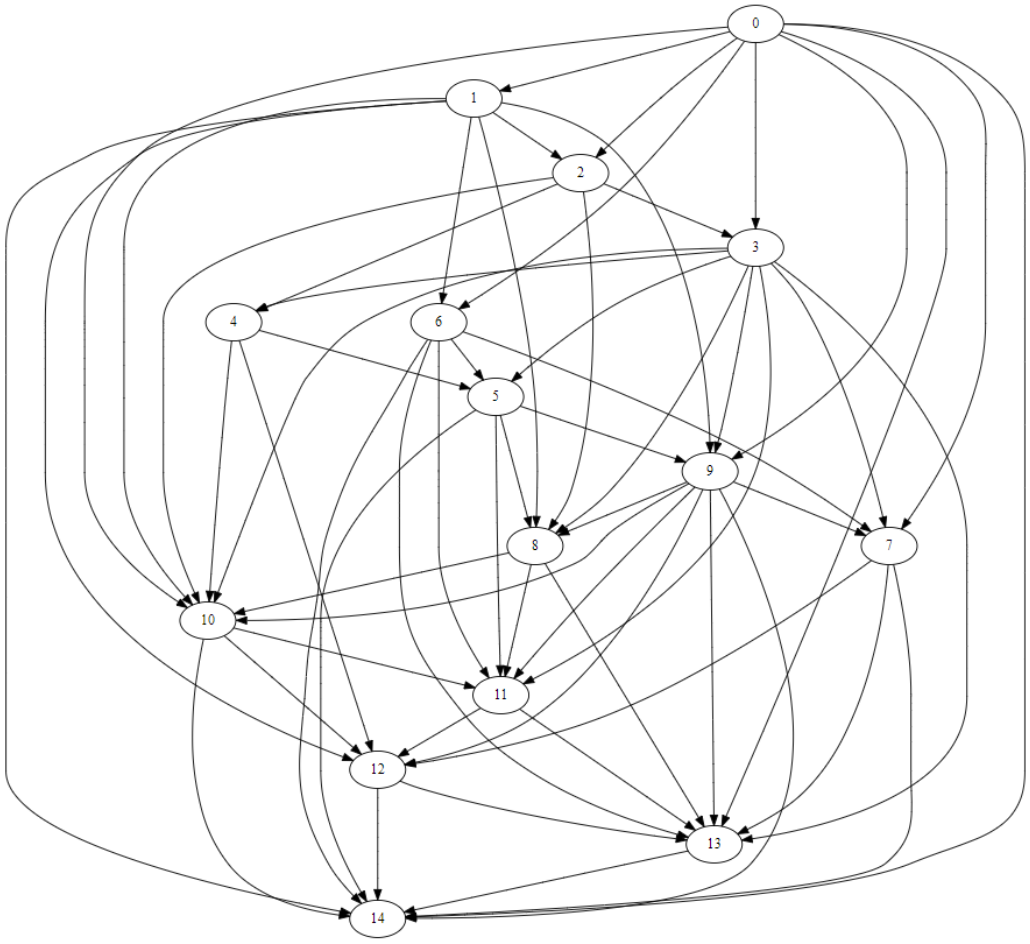


```
R9-----------------------------------
    0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0   0  1  1  1  0  0  1  1  0  1   1   0   0   1   1
1   0  0  1  0  0  0  1  0  1  1   1   0   1   0   1
2   0  0  0  1  1  0  0  0  1  0   1   0   0   0   0
3   0  0  0  0  1  1  0  1  1  1   1   1   0   1   0
4   0  0  0  0  0  1  0  0  0  0   1   0   1   0   0
5   0  0  0  0  0  0  0  0  1  1   0   1   0   0   1
6   0  0  0  0  0  1  0  1  0  0   0   1   0   1   1
7   0  0  0  0  0  0  0  0  0  0   0   0   1   1   1
8   0  0  0  0  0  0  0  0  0  0   1   1   0   1   0
9   0  0  0  0  0  0  0  1  1  0   1   1   1   1   1
10  0  0  0  0  0  0  0  0  0  0   0   1   1   0   1
11  0  0  0  0  0  0  0  0  0  0   0   0   1   1   0
12  0  0  0  0  0  0  0  0  0  0   0   0   0   1   1
13  0  0  0  0  0  0  0  0  0  0   0   0   0   0   1
14  0  0  0  0  0  0  0  0  0  0   0   0   0   0   0

    0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0   N  P  P  P  N  N  P  P  N  P   P   N   N   P   P
1   0  N  P  N  N  N  P  N  P  P   P   N   P   N   P
2   0  0  N  P  P  N  N  N  P  N   P   N   N   N   N
3   0  N  0  N  P  P  N  P  P  P   P   P   N   P   N
4   N  N  0  0  N  P  N  N  N  N   P   N   P   N   N
5   N  N  N  0  0  N  0  N  P  P   N   P   N   N   P
6   0  0  N  N  N  P  N  P  N  N   N   P   N   P   P
7   0  N  N  0  N  N  0  N  N  0   N   N   P   P   P
8   N  0  0  0  N  0  N  N  N  0   P   P   N   P   N
9   0  0  N  0  N  0  N  P  P  N   P   P   P   P   P
10  0  0  0  0  0  N  N  N  0  0   N   P   P   N   P
11  N  N  N  0  N  0  0  N  0  0   0   N   P   P   N
12  N  0  N  N  0  N  N  0  N  0   0   0   N   P   P
13  0  N  N  0  N  N  0  0  0  0   N   0   0   N   P
14  0  0  N  N  N  0  0  0  N  0   0   N   0   0   N

Є ациклічним: True
Розв'язок Неймана-Моргенштерна: 0 4 8
Чи є розв'язок внутрішньо стійким: True
Чи є розв'язок зовнішньо стійким: True
```

Граф:

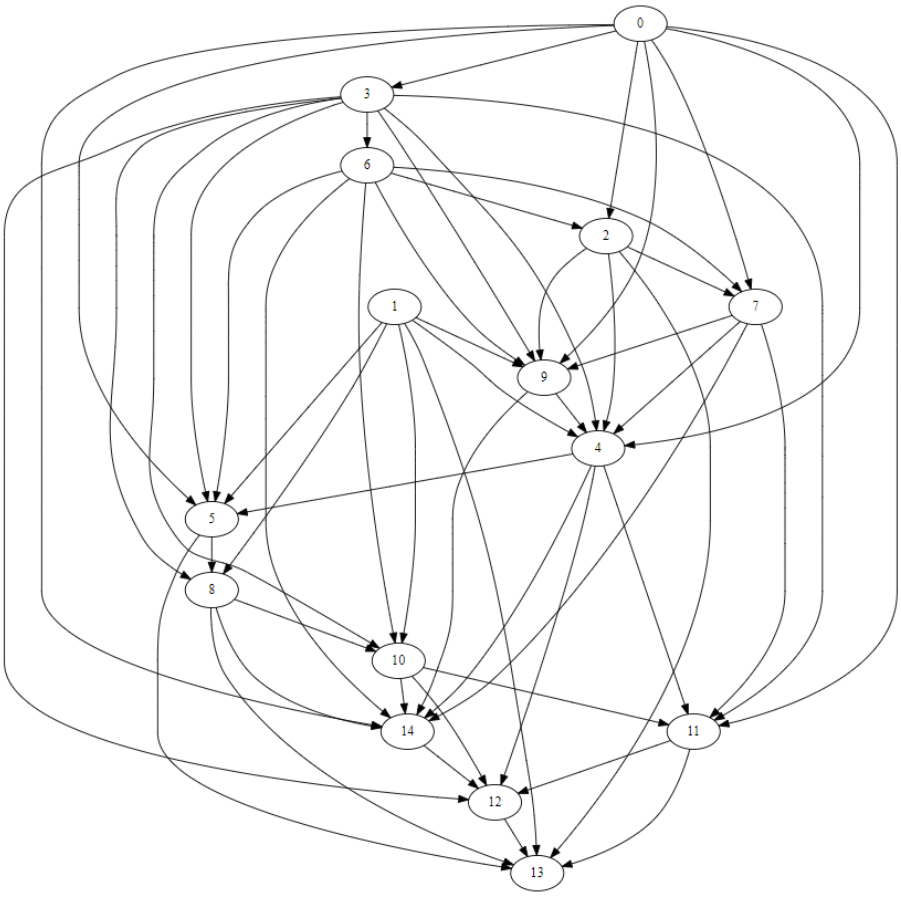


```
R10----------------------------------------
    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0   0  0  1  1  1  1  0  1  0  1   0  1  0  0  1
1   0  0  0  0  1  1  0  0  1  1   1  0  0  1  0
2   0  0  0  0  1  0  0  1  0  1   0  0  0  1  0
3   0  0  0  0  1  1  1  0  1  1   1  1  1  0  0
4   0  0  0  0  0  1  0  0  0  0   0  1  1  0  1
5   0  0  0  0  0  0  0  0  1  0   0  0  0  1  0
6   0  0  1  0  0  1  0  1  0  1   1  0  0  0  1
7   0  0  0  0  1  0  0  0  0  1   0  1  0  0  1
8   0  0  0  0  0  0  0  0  0  0   1  0  0  1  1
9   0  0  0  0  0  1  0  0  0  0   0  0  0  0  1
10  0  0  0  0  0  0  0  0  0  0   0  1  1  0  1
11  0  0  0  0  0  0  0  0  0  0   0  0  1  1  0
12  0  0  0  0  0  0  0  0  0  0   0  0  0  1  0
13  0  0  0  0  0  0  0  0  0  0   0  0  0  0  0
14  0  0  0  0  0  0  0  0  0  0   0  0  1  0  0

    0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
0   N  N  P  P  P  P  N  P  N  P   N  P  N  N  P
1   N  N  N  N  P  P  N  N  P  P   P  N  N  P  N
2   0  N  N  N  P  N  0  P  N  P   N  N  N  P  N
3   0  N  N  N  P  P  P  N  P  P   P  P  P  N  N
4   0  0  0  0  N  P  N  0  N  0   N  P  P  N  P
5   0  0  N  0  0  N  0  N  P  N   N  N  N  P  N
6   N  N  P  0  N  P  N  P  N  P   P  N  N  N  P
7   0  N  0  N  P  N  0  N  N  P   N  P  N  N  P
8   N  0  N  0  N  0  N  N  N  N   P  N  N  P  P
9   0  0  0  0  P  N  0  0  N  N   N  N  N  N  P
10  N  0  N  0  N  N  0  N  0  N   N  P  P  N  P
11  0  N  N  0  0  N  N  0  N  N   0  N  P  P  N
12  N  N  N  0  0  N  N  N  N  N   0  0  N  P  0
13  N  0  0  N  N  0  N  N  0  N   N  0  0  N  N
14  0  N  N  N  0  N  0  0  0  0   0  N  P  N  N

Є ациклічним: True
Розв'язок Неймана-Моргенштерна: 0 1 6 12
Чи є розв'язок внутрішньо стійким: True
Чи є розв'язок зовнішньо стійким: True
```

Граф:



4. Лістинг програми.

Посилання на github-репозиторій з кодом: https://github.com/KasprukNastia/decisions/

Клас **BfsCycleFinder**

```
public class BfsCycleFinder
  {
    // Пошук циклу у графі за допомогою алгоритму BFS
    public bool HasCycle(Relation relation)
    {
      // visited - множина для вже пройдених вершин, щоб не повертатися у них
      HashSet<int> visited = new HashSet<int>();
      // queue - черга для BFS
      Queue<int> queue;
      // currentVertex - номер вершини, витягнутої з черги BFS
      int currentVertex;
      // Проходимося по всім вершинам
      for(int vertex = 0; vertex < relation.Dimension; vertex++)
      {
        queue = new Queue<int>();

        // Додаємо всі сусідні для поточної (vertex) ще не пройдені вершини у чергу
        for (int index = 0; index < relation.Dimension; index++)
        {
          if (relation.Connections[vertex][index] == 1 && !visited.Contains(index))
            queue.Enqueue(index);
        }

        // Поки черга не спорожніє
        while (queue.Count > 0)
        {
          currentVertex = queue.Dequeue();

          // Якщо у черзі наткнулися на вершину, що співпадає з поточною, маємо цикл
          if (currentVertex == vertex)
            return true;

          // Додаємо всі сусідні для витягнутої (currentVertex) ще не пройдені вершини у чергу
          for (int index = 0; index < relation.Dimension; index++)
          {
            if (relation.Connections[currentVertex][index] == 1 && !visited.Contains(index))
              queue.Enqueue(index);
          }
        }

        // Додаємо поточну вершину у множену пройдених
        visited.Add(vertex);
      }

      return false;
    }
  }
```

Клас **KOptimization**

```
/// <summary>
/// Клас, що реалізує алгоритм К-оптимізації
/// </summary>
public class KOptimization
{
    // Маркери множин для К1-оптимізації
    private static readonly List<char> k1SuitableSetsMarkers = new List<char> { 'I', 'P', 'N' };
    // Маркери множин для К2-оптимізації
    private static readonly List<char> k2SuitableSetsMarkers = new List<char> { 'P', 'N' };
    // Маркери множин для К3-оптимізації
    private static readonly List<char> k3SuitableSetsMarkers = new List<char> { 'I', 'P' };
    // Маркери множин для К4-оптимізації
    private static readonly List<char> k4SuitableSetsMarkers = new List<char> { 'P' };

    /// <summary>
    /// Пошук К1-максимальних елементів
    /// </summary>
    public HashSet<int> GetK1BestAlternatives(Relation relation)
    {
        return GetBestAlternatives(relation, k1SuitableSetsMarkers);
    }

    /// <summary>
    /// Пошук К2-максимальних елементів
    /// </summary>
    public HashSet<int> GetK2BestAlternatives(Relation relation)
    {
        return GetBestAlternatives(relation, k2SuitableSetsMarkers);
    }

    /// <summary>
    /// Пошук К3-максимальних елементів
    /// </summary>
    public HashSet<int> GetK3BestAlternatives(Relation relation)
    {
        return GetBestAlternatives(relation, k3SuitableSetsMarkers);
    }

    /// <summary>
    /// Пошук К4-максимальних елементів
    /// </summary>
    public HashSet<int> GetK4BestAlternatives(Relation relation)
    {
        return GetBestAlternatives(relation, k4SuitableSetsMarkers);
    }

    /// <summary>
    /// Перевірка К1-оптимальних елементів
    /// </summary>
    public HashSet<int> CheckK1OptAlternatives(HashSet<int> k1BestAlternatives, Relation relation)
    {
        HashSet<int> optAlternatives = new HashSet<int>();

        // Перевіряємо всі К1-максимальні вершини
        foreach (int alternative in k1BestAlternatives)
        {
            // Якщо нижній переріз К1-максимальної вершини містить всі елементи відношення, то дана вершина є К1-
оптимальною
            if (relation.Characteristic[alternative].Where(c => k1SuitableSetsMarkers.Contains(c)).Count() == relation.Dimension)
                optAlternatives.Add(alternative);
        }
```

```
        return optAlternatives;
    }

    /// <summary>
    /// Реалізація алгоритму Kn-оптимізації
    /// </summary>
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    private HashSet<int> GetBestAlternatives(
        Relation relation,
        List<char> suitableSetsMarkers)
    {
        // Будуємо множину Sn
        int[][] characteristicSet = new int[relation.Dimension][];

        Dictionary<int, int> verticesRelationsCounter = new Dictionary<int, int>();
        for (int i = 0; i < relation.Dimension; i++)
        {
            characteristicSet[i] = new int[relation.Dimension];
            verticesRelationsCounter[i] = 0;
            for (int j = 0; j < relation.Dimension; j++)
            {
                if (suitableSetsMarkers.Contains(relation.Characteristic[i][j]))
                {
                    characteristicSet[i][j] = 1;
                    verticesRelationsCounter[i]++;
                }
            }
        }

        Relation helperSetRelation = new Relation(characteristicSet);

        // Відбираємо максимальні за включенням множини Sn
        int maxForBetter = verticesRelationsCounter.Values.Max();
        HashSet<int> bestAlternatives =
            verticesRelationsCounter.Where(elem => elem.Value == maxForBetter)
            .Select(elem => elem.Key)
            .ToHashSet();

        Dictionary<int, HashSet<int>> bestAlternativesLowerSections =
            bestAlternatives.ToDictionary(elem => elem, elem => helperSetRelation.GetLowerSection(elem));

        HashSet<int> vertexLowerSection;
        // Перевіряємо, що інші рядки не включають найкращі як власні підмножини
        for (int vertex = 0; vertex < helperSetRelation.Dimension; vertex++)
        {
            vertexLowerSection = helperSetRelation.GetLowerSection(vertex);
            foreach (var alternative in bestAlternativesLowerSections)
            {
                if (vertexLowerSection.Except(alternative.Value).Count() > 0)
                    bestAlternatives.Remove(alternative.Key);
            }
        }

        return bestAlternatives;
    }
}
```

Клас **NMOptimization**

```
/// <summary>
/// Клас, що реалізує алгоритм Неймана-Моргенштерна
/// </summary>
public class NMOptimization
{
   /// <summary>
   /// Пошук найкращих альтернатив за алгоритмом Неймана-Моргенштерна
   /// </summary>
   public HashSet<int> GetBestAlternatives(Relation relation)
   {
      HashSet<int> allVertices = Enumerable.Range(0, relation.Dimension).ToHashSet();
      List<HashSet<int>> sSets = new List<HashSet<int>>();

      HashSet<int> prev = new HashSet<int>();
      HashSet<int> next;
      // Побудова множин Sn
      while (allVertices.Count > 0)
      {
         next = new HashSet<int>(prev);

         foreach (int vertex in allVertices)
         {
            if (relation.GetUpperSection(vertex).Except(prev).Count() == 0)
               next.Add(vertex);
         }

         sSets.Add(next);
         prev = next;

         allVertices = allVertices.Except(next).ToHashSet();
      }

      HashSet<int> bestAlternatives = new HashSet<int>(sSets[0]);
      if (sSets.Count == 1)
         return bestAlternatives;
      // Побудова множин Qn
      for (int i = 1; i < sSets.Count; i++)
      {
         next = sSets[i];
         foreach(int vertex in next)
         {
            if (relation.GetUpperSection(vertex).Intersect(bestAlternatives).Count() == 0)
               bestAlternatives.Add(vertex);
         }
      }

      return bestAlternatives;
   }

   /// <summary>
   /// Перевірка найкращих альтернатив на внутрішню стійкість
   /// </summary>
   public bool IsBestAlternativesInternallyStable(HashSet<int> bestAlternatives, Relation relation)
   {
      // Якщо верхній переріз якої-небудь з найкращих альтернатив містить
      // у собі іншу найкращу альтернативу, то множина розв'язків не є внутрішньо стійкою
      foreach (int alternative in bestAlternatives)
      {
         if (relation.GetUpperSection(alternative).Intersect(bestAlternatives).Count() > 0)
            return false;
      }

      return true;
```

```
    }

    /// <summary>
    /// Перевірка найкращих альтернатив на зовнішню стійкість
    /// </summary>
    public bool IsBestAlternativesExternallyStable(HashSet<int> bestAlternatives, Relation relation)
    {
        HashSet<int> notBestAlternatives =
            Enumerable.Range(0, relation.Dimension)
            .Except(bestAlternatives)
            .ToHashSet();

        // Якщо верхній переріз якої-небудь з альтернатив, що не входять до найкращих, не містить
        // у собі жодної з найкращих альтернатив, то множина розв'язків не є зовнішньо стійкою
        foreach (int alternative in notBestAlternatives)
        {
            if (relation.GetUpperSection(alternative).Intersect(bestAlternatives).Count() == 0)
                return false;
        }

        return true;
    }
}
```

Клас **Relation**

```
/// <summary>
/// Клас, що описує відношення
/// </summary>
public class Relation
{
   /// <summary>
   /// Зв'язки відношення
   /// </summary>
   public int[][] Connections { get; }

   /// <summary>
   /// Розмірність відношення
   /// </summary>
   public int Dimension { get; }


   private char[][] _characteristic;

   /// <summary>
   /// Характеристика відношення у множинах 'I', 'P', 'N'
   /// </summary>
   public char[][] Characteristic
   {
      get
      {
         if (_characteristic != null)
            return _characteristic;

         _characteristic = new char[Dimension][];
         for (int i = 0; i < Dimension; i++)
            _characteristic[i] = new char[Dimension];

         for (int i = 0; i < Dimension; i++)
         {
            for(int j = i; j < Dimension; j++)
            {
               if(Connections[i][j] == 1 && Connections[j][i] == 1)
                  _characteristic[i][j] = _characteristic[j][i] = 'I';
               else if(Connections[i][j] == 0 && Connections[j][i] == 0)
                  _characteristic[i][j] = _characteristic[j][i] = 'N';
               else if (Connections[i][j] == 1 && Connections[j][i] == 0)
               {
                  _characteristic[i][j] = 'P';
                  _characteristic[j][i] = '0';
               }
               else if (Connections[i][j] == 0 && Connections[j][i] == 1)
               {
                  _characteristic[j][i] = 'P';
                  _characteristic[i][j] = '0';
               }
            }
         }

         return _characteristic;
      }
   }

   public Relation(int[][] connections)
   {
      Connections = connections ?? throw new ArgumentNullException(nameof(connections));

      Dimension = connections.Length;
```

```
        for(int i = 0; i < Dimension; i++)
        {
            if (connections[i].Length != Dimension)
                throw new ArgumentException($"{nameof(connections)} must be represented as a square matrix");

            for (int j = 0; j < Dimension; j++)
            {
                if (connections[i][j] != 0 && connections[i][j] != 1)
                    throw new ArgumentException($"{nameof(connections)} must be represented only as 0 or 1 digits");
            }
        }
    }

    /// <summary>
    /// Отримання верхнього перерізу для вершини vertex
    /// </summary>
    public HashSet<int> GetUpperSection(int vertex)
    {
        if (vertex < 0 || vertex >= Dimension)
            throw new ArgumentException($"The vertex {vertex} does not belong to the relation");

        HashSet<int> upperSection = new HashSet<int>();
        for (int i = 0; i < Dimension; i++)
        {
            if (Connections[i][vertex] == 1)
                upperSection.Add(i);
        }

        return upperSection;
    }

    /// <summary>
    /// Отримання нижнього перерізу для вершини vertex
    /// </summary>
    public HashSet<int> GetLowerSection(int vertex)
    {
        if (vertex < 0 || vertex >= Dimension)
            throw new ArgumentException($"The vertex {vertex} does not belong to the relation");

        HashSet<int> lowerSection = new HashSet<int>();
        for (int i = 0; i < Dimension; i++)
        {
            if (Connections[vertex][i] == 1)
                lowerSection.Add(i);
        }

        return lowerSection;
    }

    /// <summary>
    /// Приведення відношення до рядка
    /// </summary>
    public override string ToString() =>
        string.Join(Environment.NewLine, Connections.Select(arr => string.Join(' ', arr)));

    /// <summary>
    /// Приведення характеристики відношення до рядка
    /// </summary>
    public string CharateristicToString() =>
        string.Join(Environment.NewLine, Characteristic.Select(arr => string.Join(' ', arr)));
}
```

Клас **Program**

```
class Program
  {
    static void Main(string[] args)
    {
      List<Relation> relations = ReadAllRelations();

      var bfsCycleFinder = new BfsCycleFinder();
      var nmOptimization = new NMOptimization();
      var kOptimization = new KOptimization();

      Relation relation;
      bool hasCycle;
      HashSet<int> bestAlternatives;
      for (int relationNum = 1; relationNum <= relations.Count; relationNum++)
      {
        relation = relations[relationNum - 1];
        Console.WriteLine($"R{relationNum}{string.Concat(Enumerable.Repeat('-', 50))}");
        PrintRelation(relation, () => relation.Connections);
        PrintRelation(relation, () => relation.Characteristic);

        hasCycle = bfsCycleFinder.HasCycle(relation);
        Console.WriteLine($"Є ациклічним: {!hasCycle}");

        if (!hasCycle)
        {
          bestAlternatives = nmOptimization.GetBestAlternatives(relation);
          Console.WriteLine($"Розв’язок Неймана-Морґенштерна: {string.Join(" ", bestAlternatives)}");
          Console.WriteLine($"Чи є розв’язок внутрішньо стійким:
{nmOptimization.IsBestAlternativesInternallyStable(bestAlternatives, relation)}");
          Console.WriteLine($"Чи є розв’язок зовнішньо стійким:
{nmOptimization.IsBestAlternativesExternallyStable(bestAlternatives, relation)}");
        }
        else
        {
          Console.WriteLine("Опт. альтернативи за принципом K - оптимізації");
          bestAlternatives = kOptimization.GetK1BestAlternatives(relation);
          Console.WriteLine($"K = 1: {string.Join(" ", bestAlternatives)}");
          bestAlternatives = kOptimization.CheckK1OptAlternatives(bestAlternatives, relation);
          Console.WriteLine($"K = 1 (opt): {string.Join(" ", bestAlternatives)}");
          bestAlternatives = kOptimization.GetK2BestAlternatives(relation);
          Console.WriteLine($"K = 2: {string.Join(" ", bestAlternatives)}");
          bestAlternatives = kOptimization.GetK3BestAlternatives(relation);
          Console.WriteLine($"K = 3: {string.Join(" ", bestAlternatives)}");
          bestAlternatives = kOptimization.GetK4BestAlternatives(relation);
          Console.WriteLine($"K = 4: {string.Join(" ", bestAlternatives)}");
        }
        Console.WriteLine();
        Console.WriteLine();
      }
    }

    public static List<Relation> ReadAllRelations()
    {
      string directoryPath = Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.FullName;
      string fileName = $"{directoryPath}\\relations_var11.txt";
      string[] allFileLines = File.ReadAllLines(fileName);

      List<Relation> relations = new List<Relation>(10);
      int[][] relation = new int[15][];
      for (int i = 1; i < allFileLines.Length; i++)
      {
        if (i % 16 == 0)
        {
```

```
            relations.Add(new Relation(relation));
            relation = new int[15][];
            continue;
        }

        relation[(i - 1 - relations.Count) % 15] = allFileLines[i].Split(' ')
            .Where(s => !string.IsNullOrWhiteSpace(s))
            .Select(s => int.Parse(s))
            .ToArray();
    }
    relations.Add(new Relation(relation));

    return relations;
  }

  public static void PrintRelation<T>(Relation relation, Func<T[][]> printingSelector)
  {
    T[][] toPrint = printingSelector();

    Console.ForegroundColor = ConsoleColor.Green;
    Console.WriteLine($"   {string.Join(' ', Enumerable.Range(0, 15))}");
    for(int i = 0; i < relation.Dimension; i++)
    {
      Console.Write($"{i}{string.Concat(Enumerable.Repeat(' ', 3 - i.ToString().Length))}");
      Console.ForegroundColor = ConsoleColor.White;
      for(int j = 0; j < relation.Dimension; j++)
      {
        Console.Write($"{toPrint[i][j]}{string.Concat(Enumerable.Repeat(' ', (j + 1).ToString().Length))}");
      }
      Console.WriteLine();
      Console.ForegroundColor = ConsoleColor.Green;
    }
    Console.ForegroundColor = ConsoleColor.White;
    Console.WriteLine();
  }

  public static void PrintRelationGraph(Relation relation)
  {
    for (int i = 0; i < relation.Dimension; i++)
    {
      for (int j = 0; j < relation.Dimension; j++)
      {
        if(relation.Connections[i][j] == 1)
          Console.WriteLine($"{i} -> {j}");
      }
    }
    Console.WriteLine();
  }
}
```

5. Опис використаних бібліотек і методів.

У даній лабораторній роботі використовувалися наступні методи-операції над множинами із простору імен System.Linq

Назва методу	Опис
Except	Повертає різницю множин.
Intersect	Повертає переріз множин.

6. Опис класів. Перелік розроблених функцій на методів.

Клас	Ф-ція/Метод	Параметри	Опис	Значення, що повертає
BfsCycleFinder	HasCycle	Relation relation – відношення, для якого шукається цикл	Пошук циклу у графі за допомогою алгоритму BFS	bool – булеве значення, чи має відношення цикл
KOptimization	GetK1BestAlternatives	Relation relation – відношення, для якого шукаються максимальні елементи	Пошук K1-максимальних елементів	HashSet<int> - множина K1-максимальних елементів
	GetK2BestAlternatives	Relation relation – відношення, для якого шукаються максимальні елементи	Пошук K2-максимальних елементів	HashSet<int> - множина K2-максимальних елементів
	GetK3BestAlternatives	Relation relation – відношення, для якого шукаються максимальні елементи	Пошук K3-максимальних елементів	HashSet<int> - множина K3-максимальних елементів
	GetK4BestAlternatives	Relation relation – відношення, для якого шукаються максимальні елементи	Пошук K4-максимальних елементів	HashSet<int> - множина K4-максимальних елементів
	CheckK1OptAlternatives	HashSet<int> k1BestAlternatives - множина K1-максимальних альтернатив Relation relation – відношення, для якого перевіряються K1-максимальні елементи	Перевірка K1-оптимальних елементів	HashSet<int> - множина K1-оптимальних елементів
	GetBestAlternatives	Relation relation – відношення, для якого шукаються максимальні елементи List<char> suitableSetsMarkers - маркери множин для Kn-оптимізації (I', 'P', 'N')	Реалізація алгоритму Kn-оптимізації	HashSet<int> - множина Kn-максимальних елементів
NMOptimization	GetBestAlternatives	Relation relation – відношення, для якого шукаються найкращі альтернативи	Пошук найкращих альтернатив за алгоритмом Неймана-Моргенштерна	HashSet<int> - множина найкращих альтернатив за Нейманом-Моргенштерном
	IsBestAlternativesInternallyStable	HashSet<int> bestAlternatives – множина найкращих альтернатив для перевірки Relation relation – задане відношення	Перевірка найкращих альтернатив на внутрішню стійкість	bool – булеве значення, чи є найкращі альтернативи внутрішньо стійкими
	IsBestAlternativesExternallyStable	HashSet<int> bestAlternatives – множина найкращих альтернатив для перевірки Relation relation – задане відношення	Перевірка найкращих альтернатив на зовнішню стійкість	bool – булеве значення, чи є найкращі альтернативи зовнішньо стійкими

Relation	GetUpperSection	int vertex – номер вершини	Отримання верхнього перерізу для вершини	HashSet<int> - верхній переріз
	GetLowerSection	int vertex – номер вершини	Отримання нижнього перерізу для вершини	HashSet<int> - нижній переріз

7. Висновки.

Оптимізація за бінарним відношенням (БВ) – пошук підмножини альтернатив, які переважають інші альтернативи за цим бінарним відношенням (або не «гірші» за інші альтернативи цієї множини за цим БВ).

У даній лабораторній роботі ми мали можливість ознайомитися з наступними методами оптимізації бінарних відношень:

- принцип домінування;
- принцип блокування;
- Неймана-Моргенштерна;
- К-оптимізація.

Також ми мали можливість реалізувати алгоритм для пошуку циклу в графі.

Оптимізація за Нейманом-Моргенштерном підходить тільки для ациклічних відношень. К-оптимізацію застосовують для відношень, що мають цикл.

У результаті ми навчилися використовувати наведенні алгоритми для пошуку найкращих рішень, змогли знайти найкращі альтернативи для ряду відношень, що наведені в завданнях 1 та 2.