Міністерство освіти України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

ЗВІТ

до практикуму № 3

з дисципліни

“Інфомаційні технології підтримки прийняття рішень”

на тему: “ Відношення переваги при глобальній порівнюваності критеріїв”

Варіант 11

Виконала:

Студентка групи ІП-71

Каспрук Анастасія Андріївна

Київ 2021

1. Результати.

Відношення	Опт. альтернативи за принципом домінування	Опт. альтернативиза принципом блокування
Парето	$X^{*}_{R} = \emptyset$. $X^{**}_{R} = \emptyset$.	$X^{0}_{R} = \{0, 6, 8, 12, 14, 16, 19\ \}$. $X^{00}_{R} = \{0, 6, 8, 12, 14, 16, 19\ \}$.
Мажоритарне	$X^{*}_{P} = \{\ 16\ \}$.	$X^{0}_{P} = \{\ 16\ \}$.
Лексикографічне	$X^{*}_{P} = \{\ 16\ \}$.	$X^{0}_{P} = \{\ 16\ \}$.
Березовського	$X^{*}_{P} = \emptyset$.	$X^{0}_{P} = \{\ 6, 14, 16, 19\ \}$.
Подиновського	$X^{*}_{R} = \emptyset$. $X^{**}_{R} = \emptyset$.	$X^{0}_{R} = \{\ 6, 16, 19\ \}$. $X^{00}_{R} = \{\ 6, 16, 19\ \}$.

2. Постановка задачі.

Задано множину з 20 альтернатив, які оцінені за множиною критеріїв K = {k_i}, i = 1,…,12.

У вхідному файлі міститься інформація:

1) оцінки альтернатив за критеріями множини K (20 рядків, j-й рядок – це оцінки альтернативи j)
2) про порівнюваність критеріїв:
   - впорядкування критеріїв за спаданням важливості, яке відповідає відношенню строгогопорядку V1на множині K;
   - впорядкування класів рівноважливих критеріїв за зростанням важливості класів, яке відповідаєвідношенню квазіпорядку V2на множині K

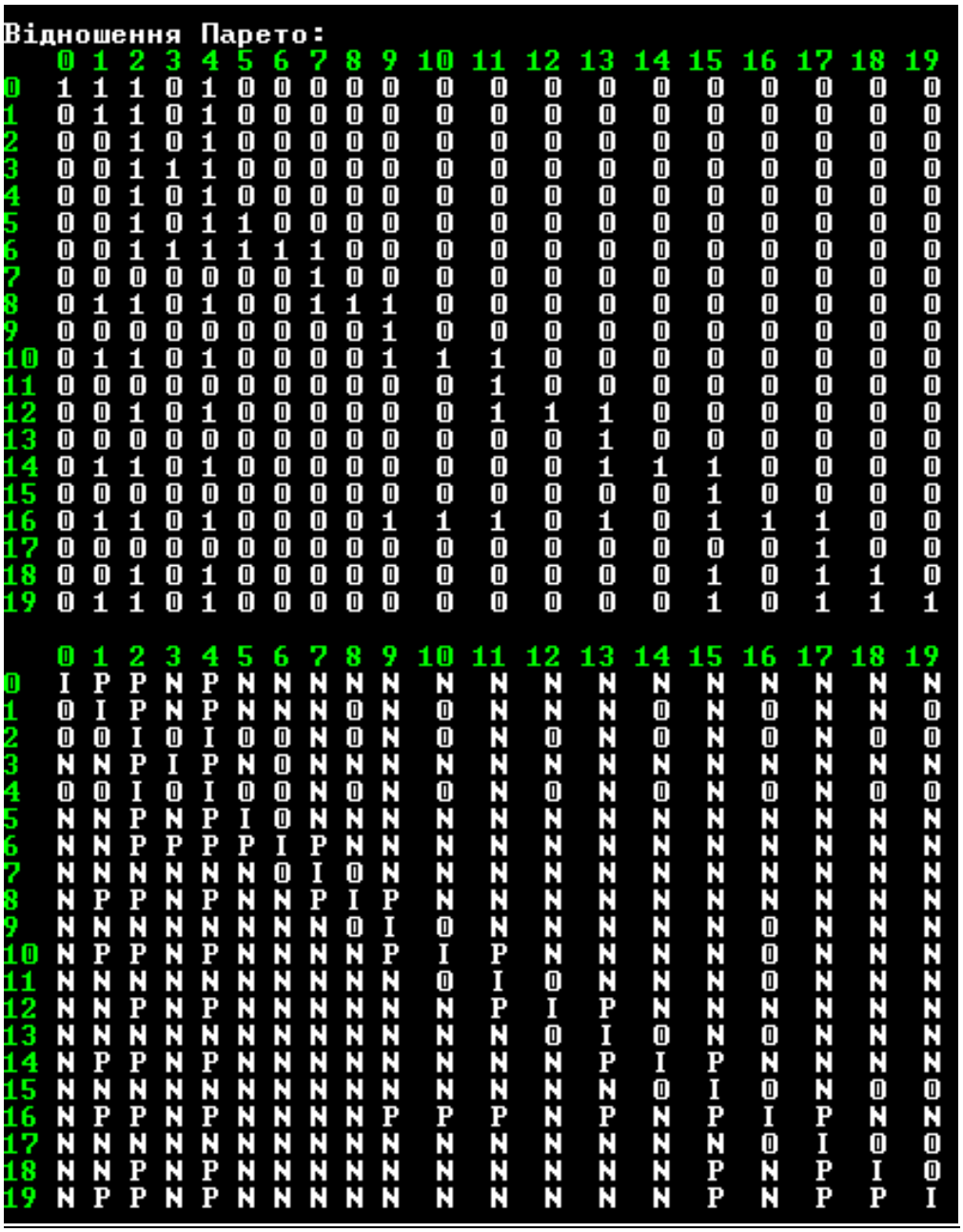
Необхідно за інформацією про оцінки альтернатив за критеріями к1-к12 та інформацією про порівнюваність критеріїв побудувати на множині альтернатив **відношенняпереваги** та визначити **оптимальні альтернативи**, якщо:

1) інформація про порівнюваність критеріїв несуттєва (відн. Парето);
2) критерії рівноважливі (мажоритарне в.);
3) на множині критеріїв задане віднош. строгого порядку V1 (лексикографічне в.);
4) на множині критеріїв задане відношення квазіпорядку V2 (відн. Березовського);
5) для випадку рівноважливих критеріїв побудувати на множині альтернатив відношення Подиновського.

Завдання для варіанту 11:

```
10  1  5  1  9  2  3  9  6  7  6  5
 1  1  5  1  5  2  1  9  6  7  6  4
 1  1  5  1  5  2  1  8  4  7  5  4
 3  4  5  3  7  6  7  9  8  7  5  4
 1  1  5  1  5  2  1  8  4  7  5  4
 8  8  5 10  5  8 10  8  7  7  5  7
 8  8  5 10  7  8 10  9  8  7  5 10
 8  2  5  3  7  8 10  3  5  3  2  6
 8  5  5  5  9  9 10  9  6  7  9  6
 2  5  5  3  4  9  6  4  6  5  8  1
10  5  5  3  6  9  9 10 10  7  8  8
 1  5  4  1  5  8  7  3 10  4  7  8
 5  8  6  2  7  8  7  8 10  7  7  8
 5  4  6  2  1  5  5  5  1  7  7  5
10  5 10 10  5  5  6  9  8  7  7  5
 5  5  6  4  3  2  1  5  3  7  1  5
10  8  6  6  6 10  9 10 10  7  8  8
 3  4  4  1  6 10  3  6  4  7  1  2
 8  9  6  5  6 10  3  9  4  7  8  5
 9 10  9  5  6 10  3  9 10 10  9  5
Відношення строгого порядку на мн-ні критеріїв
(впорядкування за спаданням важливості):
k1>k8>k4>k10>k12>k3>k11>k5>k2>k7>k6>k9
Відношення квазіпорядку на мн-ні критеріїв
(класи впорядковані за зростанням важливості):
{k6,k8,k12} < {k4,k5,k10,k11} < {k1,k2,k3,k7,k9}
```

3. Розв’язок.



Відношення Парето:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
8	0	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
10	0	1	1	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
12	0	0	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
14	0	1	1	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
16	0	1	1	0	1	0	0	0	0	1	1	1	0	1	0	1	1	1	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
18	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0
19	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	I	P	P	N	P	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
1	0	I	P	N	P	N	N	N	0	N	0	N	N	N	0	N	0	N	N	0
2	0	0	I	0	I	0	0	N	0	N	0	N	0	N	0	N	0	N	0	0
3	N	N	P	I	P	N	0	N	N	N	N	N	N	N	N	N	N	N	N	N
4	0	0	I	0	I	0	0	N	0	N	0	N	0	N	0	N	0	N	0	0
5	N	N	P	N	P	I	0	N	N	N	N	N	N	N	N	N	N	N	N	N
6	N	N	P	P	P	P	I	P	N	N	N	N	N	N	N	N	N	N	N	N
7	N	N	N	N	N	N	0	I	0	N	N	N	N	N	N	N	N	N	N	N
8	N	P	P	N	P	N	N	P	I	P	N	N	N	N	N	N	N	N	N	N
9	N	N	N	N	N	N	N	N	0	I	0	N	N	N	N	N	0	N	N	N
10	N	P	P	N	P	N	N	N	N	P	I	P	N	N	N	N	0	N	N	N
11	N	N	N	N	N	N	N	N	N	N	0	I	0	N	N	N	0	N	N	N
12	N	N	P	N	P	N	N	N	N	N	N	P	I	P	N	N	N	N	N	N
13	N	N	N	N	N	N	N	N	N	N	N	N	0	I	0	N	0	N	N	N
14	N	P	P	N	P	N	N	N	N	N	N	N	N	P	I	P	N	N	N	N
15	N	N	N	N	N	N	N	N	N	N	N	N	N	N	0	I	0	N	0	0
16	N	P	P	N	P	N	N	N	N	P	P	P	N	P	N	P	I	P	N	N
17	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	0	I	0	0	0
18	N	N	P	N	P	N	N	N	N	N	N	N	N	N	N	P	N	P	I	0
19	N	P	P	N	P	N	N	N	N	N	N	N	N	N	N	P	N	P	P	I

Оптимізація за домінуванням:

$I \neq \emptyset$, тому шукаємо X^{*}_{R} та X^{**}_{R}.

$X^{*}_{R} = \emptyset$, оскільки немає рядка зі всіма одиницями.

$X^{**}_{R} = \emptyset$, оскільки немає рядка зі всіма одиницями, $X^{*}_{R} = \emptyset$.

Оптимізація за блокуванням:

$I \neq \emptyset$, тому шукаємо X^{0}_{R} та X^{00}_{R}.

$X^{0}_{R} = \{0, 6, 8, 12, 14, 16, 19\}$, оскільки у відповідних стовпцях присутні тільки нулі та I.

$X^{00}_{R} = \{0, 6, 8, 12, 14, 16, 19\}$, оскільки у відповідних стовпцях всі нулі, окрім клітинок, що характеризують пару альтернативи з самою собою.

Мажоритарне відношення:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0
1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	0	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	0	0
6	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	0	1	1	0
7	0	1	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	1	0	0
8	1	1	1	1	1	0	0	1	0	1	0	1	1	1	1	1	0	1	1	0
9	0	1	1	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0
10	1	1	1	1	1	1	0	1	0	1	0	1	1	1	1	1	0	1	1	0
11	1	1	1	0	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
12	1	1	1	1	1	1	0	1	0	1	0	1	0	1	1	1	0	1	0	0
13	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
14	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0
15	0	1	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
17	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	0	1	0	0
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	N	P	P	0	P	0	0	P	0	N	0	0	0	0	0	P	0	P	0	0
1	0	N	P	0	P	0	0	0	0	0	0	0	0	0	0	0	0	N	0	0
2	0	0	N	0	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	P	P	P	N	P	0	0	P	0	P	0	P	0	P	0	P	0	P	0	0
4	0	0	N	0	N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	P	P	P	P	P	N	0	P	N	P	0	P	0	P	0	P	0	P	0	0
6	P	P	P	P	P	P	N	P	P	P	N	P	P	P	P	P	0	P	P	0
7	0	P	P	0	P	0	0	N	0	0	0	N	0	P	0	P	0	P	0	0
8	P	P	P	P	P	N	0	P	N	P	N	P	P	P	P	P	0	P	P	0
9	N	P	P	0	P	0	0	P	0	N	0	P	0	P	0	0	0	N	0	0
10	P	P	P	P	P	P	N	P	N	P	N	P	P	P	P	P	0	P	P	0
11	P	P	P	0	P	0	0	N	0	0	0	N	0	P	0	P	0	N	0	0
12	P	P	P	P	P	P	0	P	0	P	0	P	N	P	P	P	0	P	0	0
13	P	P	P	0	P	0	0	0	0	0	0	0	0	N	0	0	0	P	0	0
14	P	P	P	P	P	P	0	P	0	P	0	P	0	P	N	P	0	P	P	0
15	0	P	P	0	P	0	0	0	0	P	0	0	0	P	0	N	0	N	0	0
16	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	N	P	P	P
17	0	N	P	0	P	0	0	0	0	N	0	N	0	0	0	N	0	N	0	0
18	P	P	P	P	P	P	0	P	0	P	0	P	P	P	0	P	0	P	N	0
19	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	0	P	P	N

Оптимізація за домінуванням:

I = ∅, тому шукаємо X^*_P.
X^*_P ={16 }, оскільки у рядку 16 всі одиниці, окрім клітинки, що характеризує пару (16, 16).

Оптимізація за блокуванням:

I = ∅, тому шукаємо X^0_P.
X^0_P = {16}, оскільки у стовпці 16 всі нулі.

Лексикографічне відношення:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1
1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
3	0	1	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	0	1	1	1	1	0	0	1	0	1	0	1	1	1	0	1	0	1	0	0
6	0	1	1	1	1	1	0	1	1	1	0	1	1	1	0	1	0	1	1	0
7	0	1	1	1	1	0	0	0	0	1	0	1	1	1	0	1	0	1	0	0
8	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	0	1	1	0
9	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
10	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	1	1	1	1	0	0	0	0	1	0	1	0	1	0	1	0	1	0	0
13	0	1	1	1	1	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0
14	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	1	1
15	0	1	1	1	1	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
17	0	1	1	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0
18	0	1	1	1	1	1	0	1	0	1	0	1	1	1	0	1	0	1	0	0
19	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	0	1	1	0

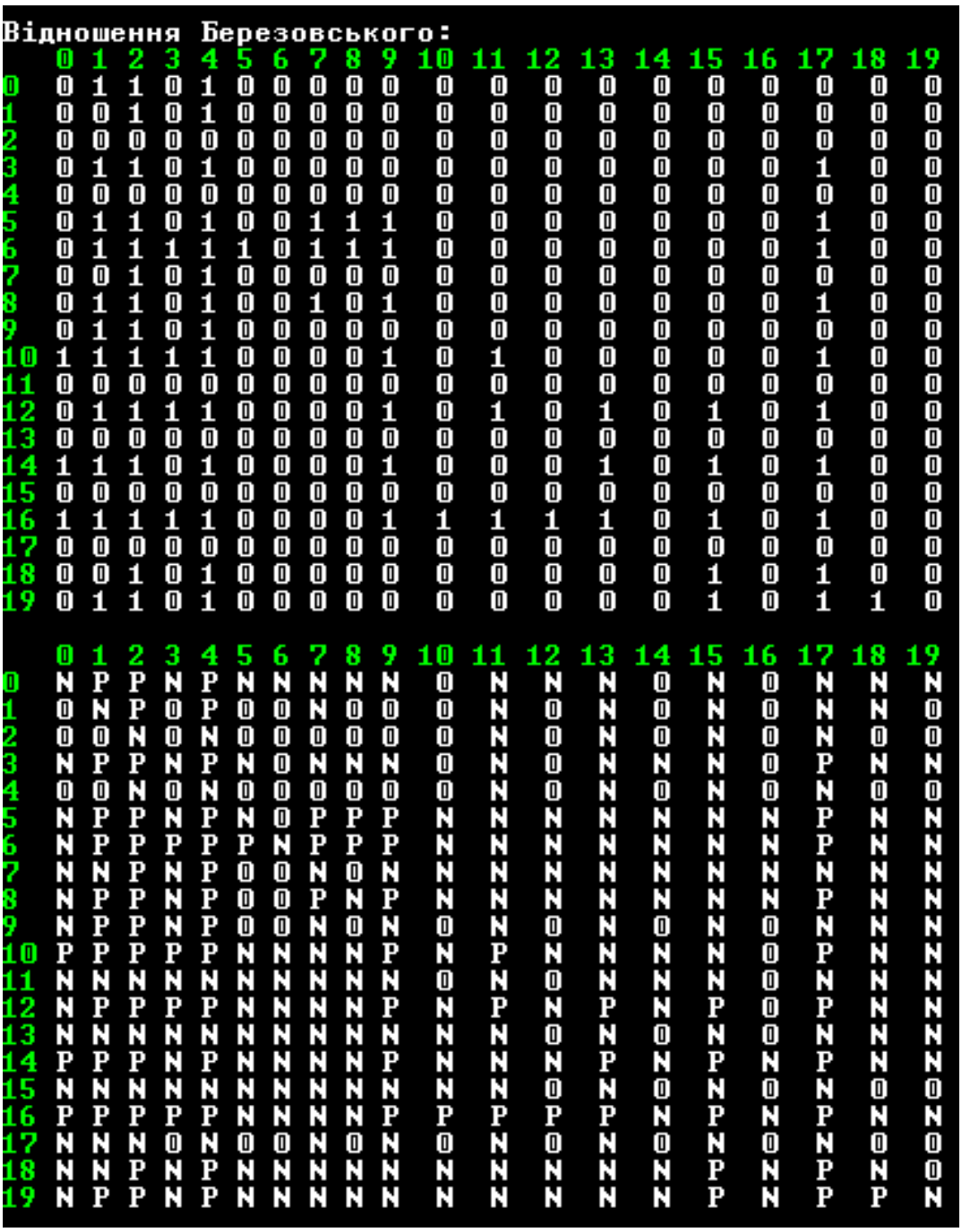
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	N	P	P	P	P	P	P	P	P	P	0	P	P	P	0	P	0	P	P	P
1	0	N	P	0	P	0	0	0	0	0	0	P	0	0	0	0	0	0	0	0
2	0	0	N	0	N	0	0	0	0	0	0	P	0	0	0	0	0	0	0	0
3	0	P	P	N	P	0	0	0	0	P	0	P	0	0	0	0	0	P	0	0
4	0	0	N	0	N	0	0	0	0	0	0	P	0	0	0	0	0	0	0	0
5	0	P	P	P	P	N	0	P	0	P	0	P	P	P	0	P	0	P	0	0
6	0	P	P	P	P	P	N	P	P	P	0	P	P	P	0	P	0	P	P	0
7	0	P	P	P	P	0	0	N	0	P	0	P	P	P	0	P	0	P	0	0
8	0	P	P	P	P	P	0	P	N	P	0	P	P	P	0	P	0	P	P	0
9	0	P	P	0	P	0	0	0	0	N	0	P	0	0	0	0	0	0	0	0
10	P	P	P	P	P	P	P	P	P	P	N	P	P	P	P	P	0	P	P	P
11	0	0	0	0	0	0	0	0	0	0	0	N	0	0	0	0	0	0	0	0
12	0	P	P	P	P	0	0	0	0	P	0	P	N	P	0	P	0	P	0	0
13	0	P	P	P	P	0	0	0	0	P	0	P	0	N	0	0	0	P	0	0
14	P	P	P	P	P	P	P	P	P	P	0	P	P	P	N	P	0	P	P	P
15	0	P	P	P	P	0	0	0	0	P	0	P	0	P	0	N	0	P	0	0
16	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	N	P	P	P
17	0	P	P	0	P	0	0	0	0	P	0	P	0	0	0	0	0	N	0	0
18	0	P	P	P	P	P	0	P	0	P	0	P	P	P	0	P	0	P	N	0
19	0	P	P	P	P	P	P	P	P	P	0	P	P	P	0	P	0	P	P	N

<u>Оптимізація за домінуванням:</u>

I = ∅, тому шукаємо X^*_P.
X^*_P ={ 16 }, оскільки у рядку 16 всі одиниці, окрім клітинки, що характеризує пару (16, 16).

<u>Оптимізація за блокуванням:</u>

I = ∅, тому шукаємо X^0_P.
X^0_P = { 16 }, оскільки у стовпці 16 всі нулі.



Відношення Березовського:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0
6	0	1	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0
7	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	1	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0
9	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	1	1	1	1	0	0	0	0	1	0	1	0	1	0	1	0	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	1	1	0	1	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	1	1	1	1	1	0	0	0	0	1	1	1	1	1	0	1	0	1	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
19	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0

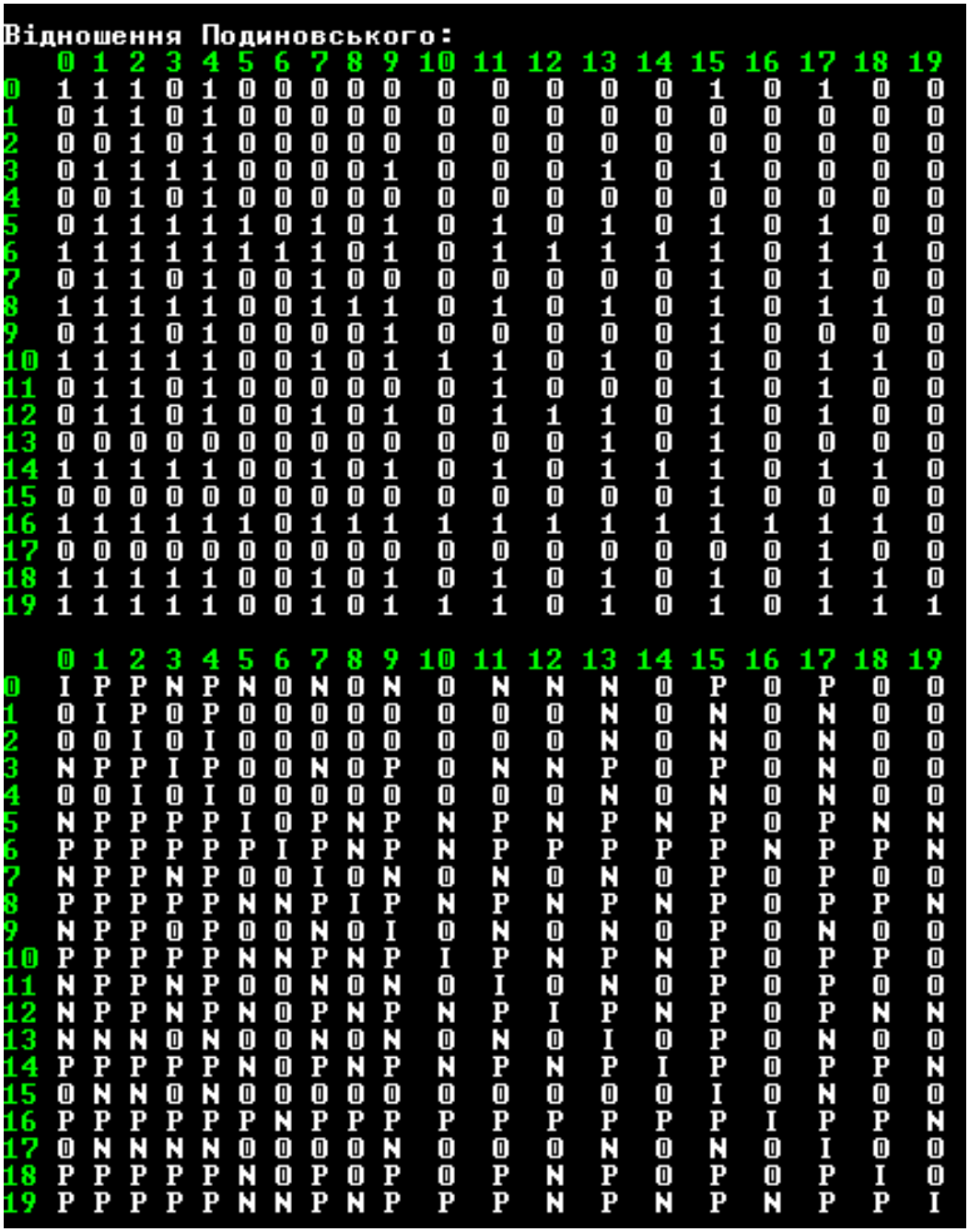
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	N	P	P	N	P	N	N	N	N	N	0	N	N	N	0	N	0	N	N	N
1	0	N	P	0	P	0	0	N	0	0	0	N	0	N	0	N	0	N	N	0
2	0	0	N	0	N	0	0	0	0	0	0	N	0	N	0	N	0	N	0	0
3	N	P	P	N	P	N	0	N	N	N	0	N	0	N	N	N	0	P	N	N
4	0	0	N	0	N	0	0	0	0	0	0	N	0	N	0	N	0	N	0	0
5	N	P	P	N	P	N	0	P	P	P	N	N	N	N	N	N	N	P	N	N
6	N	P	P	P	P	P	N	P	P	P	N	N	N	N	N	N	N	P	N	N
7	N	N	P	N	P	0	0	N	0	N	N	N	N	N	N	N	N	N	N	N
8	N	P	P	N	P	0	0	P	N	P	N	N	N	N	N	N	N	P	N	N
9	N	P	P	N	P	0	0	N	0	N	0	N	0	N	0	N	0	N	N	N
10	P	P	P	P	P	N	N	N	N	P	N	P	N	N	N	N	0	P	N	N
11	N	N	N	N	N	N	N	N	N	N	0	N	0	N	N	N	0	N	N	N
12	N	P	P	P	P	N	N	N	N	P	N	P	N	P	N	P	0	P	N	N
13	N	N	N	N	N	N	N	N	N	N	N	N	0	N	0	N	0	N	N	N
14	P	P	P	N	P	N	N	N	N	P	N	N	N	P	N	P	N	P	N	N
15	N	N	N	N	N	N	N	N	N	N	N	N	0	N	0	N	0	N	0	0
16	P	P	P	P	P	N	N	N	N	P	P	P	P	P	N	P	N	P	N	N
17	N	N	N	0	N	0	0	N	0	N	0	N	0	N	0	N	0	N	0	0
18	N	N	P	N	P	N	N	N	N	N	N	N	N	N	N	P	N	P	N	0
19	N	P	P	N	P	N	N	N	N	N	N	N	N	N	N	P	N	P	P	N

<u>Оптимізація за домінуванням:</u>

I = ∅, тому шукаємо X^*_P.
X^*_P = ∅, оскільки нема рядку зі всіма одиницями, окрім клітинки, що характеризує пару альтернативи з самою собою.

<u>Оптимізація за блокуванням:</u>

I = ∅, тому шукаємо X^0_P.
X^0_P = {6, 14, 16, 19}, оскільки у відповідних стовпцях всі нулі.



Відношення Подиновського:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	1	1	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0
4	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0
6	1	1	1	1	1	1	1	1	0	1	0	1	1	1	1	1	0	1	1	0
7	0	1	1	0	1	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0
8	1	1	1	1	1	0	0	1	1	1	0	1	0	1	0	1	0	1	1	0
9	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
10	1	1	1	1	1	0	0	1	0	1	1	1	0	1	0	1	0	1	1	0
11	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0
12	0	1	1	0	1	0	0	1	0	1	0	1	1	1	0	1	0	1	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
14	1	1	1	1	1	0	0	1	0	1	0	1	0	1	1	1	0	1	1	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
16	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
18	1	1	1	1	1	0	0	1	0	1	0	1	0	1	0	1	0	1	1	0
19	1	1	1	1	1	0	0	1	0	1	1	1	0	1	0	1	0	1	1	1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	I	P	P	N	P	N	0	N	0	N	0	N	N	N	0	P	0	P	0	0
1	0	I	P	0	P	0	0	0	0	0	0	0	0	N	0	N	0	N	0	0
2	0	0	I	0	I	0	0	0	0	0	0	0	0	N	0	N	0	N	0	0
3	N	P	P	I	P	0	0	N	0	P	0	N	N	P	0	P	0	N	0	0
4	0	0	I	0	I	0	0	0	0	0	0	0	0	N	0	N	0	N	0	0
5	N	P	P	P	P	I	0	P	N	P	N	P	N	P	N	P	0	P	N	N
6	P	P	P	P	P	P	I	P	N	P	N	P	P	P	P	P	N	P	P	N
7	N	P	P	N	P	0	0	I	0	N	0	N	0	N	0	P	0	P	0	0
8	P	P	P	P	P	N	N	P	I	P	N	P	N	P	N	P	0	P	P	N
9	N	P	P	0	P	0	0	N	0	I	0	N	0	N	0	P	0	N	0	0
10	P	P	P	P	P	N	N	P	N	P	I	P	N	P	N	P	0	P	P	0
11	N	P	P	N	P	0	0	N	0	N	0	I	0	N	0	P	0	P	0	0
12	N	P	P	N	P	N	0	P	N	P	N	P	I	P	N	P	0	P	N	N
13	N	N	N	0	N	0	0	N	0	N	0	N	0	I	0	P	0	N	0	0
14	P	P	P	P	P	N	0	P	N	P	N	P	N	P	I	P	0	P	P	N
15	0	N	N	0	N	0	0	0	0	0	0	0	0	0	0	I	0	N	0	0
16	P	P	P	P	P	P	N	P	P	P	P	P	P	P	P	P	I	P	P	N
17	0	N	N	N	N	0	0	0	0	N	0	0	0	N	0	N	0	I	0	0
18	P	P	P	P	P	N	0	P	0	P	0	P	N	P	0	P	0	P	I	0
19	P	P	P	P	P	N	N	P	N	P	P	P	N	P	N	P	N	P	P	I

<u>Оптимізація за домінуванням:</u>

$I \neq \emptyset$, тому шукаємо X^{*}_{R} та X^{**}_{R}.

$X^{*}_{R} = \emptyset$, оскільки немає рядка зі всіма одиницями.

$X^{**}_{R} = \emptyset$, оскільки немає рядка зі всіма одиницями, $X^{*}_{R} = \emptyset$.

<u>Оптимізація за блокуванням:</u>

$I \neq \emptyset$, тому шукаємо X^{0}_{R} та X^{00}_{R}.

$X^{0}_{R} = \{ 6, 16, 19 \}$, оскільки у відповідних стовпцях присутні тільки нулі та I.

$X^{00}_{R} = \{ 6, 16, 19 \}$, оскільки у відповідних стовпцях всі нулі, окрім клітинок, що характеризують пару альтернативи з самою собою.

4. Лістинг програми.

Посилання на github-репозиторій з кодом:

https://github.com/KasprukNastia/decisions/tree/master/Lab3

Клас **CriteriaRelation**

```
/// <summary>
  /// Клас, що описує значення критеріїв для альтернатив
  /// </summary>
  public class CriteriaRelation
  {
    /// <summary>
    /// Значення критеріїв для альтернатив
    /// </summary>
    public int[][] Evaluations { get; }

    /// <summary>
    /// Упорядкована за спаданням важливості множина критеріїв
    /// </summary>
    public IReadOnlyCollection<int> CriteriasImportance { get; }

    /// <summary>
    /// Класи впорядковані за зростанням важливості
    /// </summary>
    public IReadOnlyCollection<IReadOnlyCollection<int>> CriteriasImportancesClasses { get; }

    /// <summary>
    /// К-сть критеріїв
    /// </summary>
    public int CriteriasCount { get; }

    /// <summary>
    /// К-сть альтернатив
    /// </summary>
    public int AlternativesCount { get; }

    /// <summary>
    /// Матриця дельта векторів
    /// </summary>
    private List<int>[][] _deltaVectors;
    /// <summary>
    /// Матриця дельта векторів
    /// </summary>
    public List<int>[][] DeltaVectors
    {
      get
      {
        if (_deltaVectors != null)
          return _deltaVectors;

        _deltaVectors = new List<int>[AlternativesCount][];
        for (int i = 0; i < AlternativesCount; i++)
          _deltaVectors[i] = new List<int>[AlternativesCount];

        for (int i = 0; i < AlternativesCount; i++)
        {
          _deltaVectors[i][i] = Enumerable.Repeat(0, AlternativesCount).ToList();

          for (int j = i + 1; j < AlternativesCount; j++)
          {
            _deltaVectors[i][j] = Evaluations[i].Select((elem, index) => elem - Evaluations[j][index]).ToList();
            _deltaVectors[j][i] = _deltaVectors[i][j].Select(elem => elem * -1).ToList();
          }
```

```
      }

      return _deltaVectors;
    }
  }

  /// <summary>
  /// Матриця сигма векторів
  /// </summary>
  private List<int>[][] _sigmaVectors;
  /// <summary>
  /// Матриця сигма векторів
  /// </summary>
  public List<int>[][] SigmaVectors
  {
    get
    {
      if(_sigmaVectors != null)
        return _sigmaVectors;

      _sigmaVectors = new List<int>[AlternativesCount][];
      for (int i = 0; i < AlternativesCount; i++)
        _sigmaVectors[i] = new List<int>[AlternativesCount];

      for (int i = 0; i < AlternativesCount; i++)
      {
        _sigmaVectors[i][i] = Enumerable.Repeat(0, CriteriasCount).ToList();

        for (int j = i + 1; j < AlternativesCount; j++)
        {
          _sigmaVectors[i][j] = DeltaVectors[i][j].Select(elem => elem > 0 ? 1 : elem == 0 ? 0 : -1).ToList();
          _sigmaVectors[j][i] = _sigmaVectors[i][j].Select(elem => elem * -1).ToList();
        }
      }

      return _sigmaVectors;
    }
  }

  /// <summary>
  /// Значення критеріїв для альтернатив, упорядковані за спаданням важливості критеріїв
  /// </summary>
  private CriteriaRelation _sortedCriteriaRelation;
  /// <summary>
  /// Значення критеріїв для альтернатив, упорядковані за спаданням важливості критеріїв
  /// </summary>
  public CriteriaRelation SortedCriteriaRelation
  {
    get
    {
      if (_sortedCriteriaRelation != null)
        return _sortedCriteriaRelation;

      int[][] sortedEvaluations = new int[AlternativesCount][];

      for (int i = 0; i < AlternativesCount; i++)
        sortedEvaluations[i] = new int[CriteriasCount];

      int counter = 0;
      foreach (int criteria in CriteriasImportance)
      {
        for (int j = 0; j < AlternativesCount; j++)
          sortedEvaluations[j][counter] = Evaluations[j][criteria];
        counter++;
      }
```

```
      _sortedCriteriaRelation = new CriteriaRelation(sortedEvaluations);
      return _sortedCriteriaRelation;
    }
  }

  /// <summary>
  /// Відношення Парето
  /// </summary>
  private Relation _paretoRelation;
  /// <summary>
  /// Відношення Парето
  /// </summary>
  public Relation ParetoRelation
  {
    get
    {
      if (_paretoRelation != null)
        return _paretoRelation;

      int[][] paretoRelation = new int[AlternativesCount][];

      for (int i = 0; i < AlternativesCount; i++)
        paretoRelation[i] = new int[AlternativesCount];

      for(int i = 0; i < AlternativesCount; i++)
      {
        for(int j = 0; j < AlternativesCount; j++)
        {
          // альтернатива i переважає j, якщо сигма вектор пари (i,j) не містить значень -1
          if (SigmaVectors[i][j].Any(elem => elem == -1))
            paretoRelation[i][j] = 0;
          else
            paretoRelation[i][j] = 1;
        }
      }

      _paretoRelation = new Relation(paretoRelation);
      return _paretoRelation;
    }
  }

  /// <summary>
  /// Мажоритарне відношення
  /// </summary>
  private Relation _majorityRelation;
  /// <summary>
  /// Мажоритарне відношення
  /// </summary>
  public Relation MajorityRelation
  {
    get
    {
      if (_majorityRelation != null)
        return _majorityRelation;

      int[][] majorityRelation = new int[AlternativesCount][];

      for (int i = 0; i < AlternativesCount; i++)
        majorityRelation[i] = new int[AlternativesCount];

      for (int i = 0; i < AlternativesCount; i++)
      {
        for (int j = 0; j < AlternativesCount; j++)
        {
          // альтернатива i переважає j, якщо сума елементів вектору сигма більша нуля
          if (SigmaVectors[i][j].Sum() > 0)
```

```
                majorityRelation[i][j] = 1;
            else
                majorityRelation[i][j] = 0;
        }
      }

      _majorityRelation = new Relation(majorityRelation);
      return _majorityRelation;
    }
  }

  /// <summary>
  /// Лексикографічне відношення
  /// </summary>
  private Relation _lexicographicRelation;
  /// <summary>
  /// Лексикографічне відношення
  /// </summary>
  public Relation LexicographicRelation
  {
    get
    {
      if (_lexicographicRelation != null)
        return _lexicographicRelation;

      int[][] lexicographicRelation = new int[AlternativesCount][];

      for (int i = 0; i < AlternativesCount; i++)
        lexicographicRelation[i] = new int[AlternativesCount];

      for (int i = 0; i < AlternativesCount; i++)
      {
        for (int j = 0; j < AlternativesCount; j++)
        {
          foreach(int elem in SortedCriteriaRelation.SigmaVectors[i][j])
          {
            // альтернатива і переважає j, якщо сигма вектор має на своєму початку
            // будь-яку кількість нулів, а потім одиницю
            if (elem == 0)
              continue;
            if(elem == 1)
              lexicographicRelation[i][j] = 1;
            else
              lexicographicRelation[i][j] = 0;
            break;
          }
        }
      }

      _lexicographicRelation = new Relation(lexicographicRelation);
      return _lexicographicRelation;
    }
  }

  /// <summary>
  /// Відношення Березовського
  /// </summary>
  private Relation _BerezovskyRelation;
  /// <summary>
  /// Відношення Березовського
  /// </summary>
  public Relation BerezovskyRelation
  {
    get
    {
      if (_BerezovskyRelation != null || CriteriasImportancesClasses.Count == 0)
```

```
            return _BerezovskyRelation;

        List<CriteriaRelation> criteriaRelationsByClasses =
            new List<CriteriaRelation>(CriteriasImportancesClasses.Count);

        int[][] sortedEvaluations;
        int counter;
        // Формування CriteriaRelation для кожного з класів CriteriasImportancesClasses
        foreach (IReadOnlyCollection<int> criteriaClass in CriteriasImportancesClasses)
        {
            sortedEvaluations = new int[AlternativesCount][];
            for (int i = 0; i < AlternativesCount; i++)
                sortedEvaluations[i] = new int[criteriaClass.Count];

            counter = 0;
            foreach (int criteria in criteriaClass)
            {
                for (int j = 0; j < AlternativesCount; j++)
                    sortedEvaluations[j][counter] = Evaluations[j][criteria];
                counter++;
            }
            criteriaRelationsByClasses.Add(new CriteriaRelation(sortedEvaluations));
        }

        Relation currentBerezovskyRelation = criteriaRelationsByClasses.First().ParetoRelation;
        List<char> possibleCharacteristics = new List<char> { 'P', 'N', 'I' };
        Relation currentClassParetoRelation;
        int[][] nextBerezovskyRelation;
        // Ітераційний процес для формування відношення Березовського
        for (int criteriaClass = 1; criteriaClass < criteriaRelationsByClasses.Count; criteriaClass++)
        {
            currentClassParetoRelation = criteriaRelationsByClasses[criteriaClass].ParetoRelation;
            nextBerezovskyRelation = new int[AlternativesCount][];
            for (int i = 0; i < AlternativesCount; i++)
                nextBerezovskyRelation[i] = new int[AlternativesCount];

            for (int i = 0; i < AlternativesCount; i++)
            {
                for (int j = 0; j < AlternativesCount; j++)
                {
                    if ((currentClassParetoRelation.Characteristic[i][j].Equals('P') &&
                        possibleCharacteristics.Any(c => c.Equals(currentBerezovskyRelation.Characteristic[i][j]))) ||
                        (currentClassParetoRelation.Characteristic[i][j].Equals('I') &&
                        currentBerezovskyRelation.Characteristic[i][j].Equals('P')))
                    {
                        nextBerezovskyRelation[i][j] = 1;
                        nextBerezovskyRelation[j][i] = 0;
                    }
                    else if (currentClassParetoRelation.Characteristic[i][j].Equals('I') &&
                        currentBerezovskyRelation.Characteristic[i][j].Equals('I') &&
                        criteriaClass != criteriaRelationsByClasses.Count - 1)
                        nextBerezovskyRelation[i][j] = nextBerezovskyRelation[j][i] = 1;
                    else
                        nextBerezovskyRelation[i][j] = 0;
                }
            }

            currentBerezovskyRelation = new Relation(nextBerezovskyRelation);
        }

        _BerezovskyRelation = currentBerezovskyRelation;
        return _BerezovskyRelation;
    }
}

/// <summary>
```

```
    /// Відношення Подиновського
    /// </summary>
    private Relation _PodinovskyRelation;
    /// <summary>
    /// Відношення Подиновського
    /// </summary>
    public Relation PodinovskyRelation
    {
      get
      {
        if (_PodinovskyRelation != null)
          return _PodinovskyRelation;

        // Сортування значень критеріїв для кожної з альтернатив
        int[][] sortedEvaluations = new int[AlternativesCount][];
        for (int i = 0; i < AlternativesCount; i++)
          sortedEvaluations[i] = Evaluations[i].OrderByDescending(e => e).ToArray();

        var podinovskyCriteriaRelation = new CriteriaRelation(sortedEvaluations);

        // Отримання відношення Парето для відсортованих критеріїв
        _PodinovskyRelation = podinovskyCriteriaRelation.ParetoRelation;
        return _PodinovskyRelation;
      }
    }

    public CriteriaRelation(int[][] evaluations,
      HashSet<int> criteriasImportance = null,
      List<HashSet<int>> criteriasImportancesClasses = null)
    {
      Evaluations = evaluations ?? throw new ArgumentNullException(nameof(evaluations));

      AlternativesCount = evaluations.Length;

      if (AlternativesCount > 0)
        CriteriasCount = evaluations[0].Length;

      if (criteriasImportance == null)
        criteriasImportance = Enumerable.Range(0, CriteriasCount).ToHashSet();
      else
      {
        CriteriasImportance = criteriasImportance;
        if (criteriasImportance.Count != CriteriasCount)
          throw new ArgumentException($"The number of criterias does not match");
        if (CriteriasImportance.Any(elem => elem < 0 || elem >= CriteriasImportance.Count))
          throw new ArgumentException($"{nameof(criteriasImportance)} contains not existing criteria");
      }

      CriteriasImportancesClasses = criteriasImportancesClasses;

      for (int i = 1; i < AlternativesCount; i++)
      {
        if (evaluations[i].Length != CriteriasCount)
          throw new ArgumentException($"Evaluation must be provided only for {CriteriasCount} criterias");
      }
    }
  }
```

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)
    {
      CriteriaRelation criteriaRelation = ReadCriteriaRelation();
      Console.WriteLine("Сигма вектори:");
      PrintCriteriaRelationVectors(criteriaRelation, () => criteriaRelation.SigmaVectors);
      Console.WriteLine("Відношення Парето:");
      PrintRelation(criteriaRelation.ParetoRelation, () => criteriaRelation.ParetoRelation.Connections);
      PrintRelation(criteriaRelation.ParetoRelation, () => criteriaRelation.ParetoRelation.Characteristic);
      Console.WriteLine("Мажоритарне відношення:");
      PrintRelation(criteriaRelation.MajorityRelation, () => criteriaRelation.MajorityRelation.Connections);
      PrintRelation(criteriaRelation.MajorityRelation, () => criteriaRelation.MajorityRelation.Characteristic);
      Console.WriteLine("Лексикографічне відношення:");
      PrintRelation(criteriaRelation.LexicographicRelation, () => criteriaRelation.LexicographicRelation.Connections);
      PrintRelation(criteriaRelation.LexicographicRelation, () => criteriaRelation.LexicographicRelation.Characteristic);
      Console.WriteLine("Відношення Березовського:");
      PrintRelation(criteriaRelation.BerezovskyRelation, () => criteriaRelation.BerezovskyRelation.Connections);
      PrintRelation(criteriaRelation.BerezovskyRelation, () => criteriaRelation.BerezovskyRelation.Characteristic);
      Console.WriteLine("Відношення Подиновського:");
      PrintRelation(criteriaRelation.PodinovskyRelation, () => criteriaRelation.PodinovskyRelation.Connections);
      PrintRelation(criteriaRelation.PodinovskyRelation, () => criteriaRelation.PodinovskyRelation.Characteristic);

      WriteResults(criteriaRelation);
    }

    public static CriteriaRelation ReadCriteriaRelation()
    {
      string directoryPath = Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.FullName;
      string fileName = $"{directoryPath}\\relations_var11.txt";
      string[] allFileLines = File.ReadAllLines(fileName);

      int[][] relation = new int[20][];
      for (int i = 0; i < 20; i++)
      {
        relation[i] = allFileLines[i].Split(' ')
          .Where(s => !string.IsNullOrWhiteSpace(s))
          .Select(s => int.Parse(s))
          .ToArray();
      }

      HashSet<int> criteriasImportance =
        new HashSet<int> { 1, 8, 4, 10, 12, 3, 11, 5, 2, 7, 6, 9 }.Select(c => c - 1).ToHashSet();
      List<HashSet<int>> criteriasImportancesClasses =
        new List<HashSet<int>>
        {
          new HashSet<int> { 6, 8, 12 }.Select(c => c - 1).ToHashSet(),
          new HashSet<int> { 4, 5, 10, 11}.Select(c => c - 1).ToHashSet(),
          new HashSet<int> { 1, 2, 3, 7, 9}.Select(c => c - 1).ToHashSet()
        };

      return new CriteriaRelation(relation, criteriasImportance, criteriasImportancesClasses);
    }

    public static void WriteResults(CriteriaRelation criteriaRelation)
    {
      string directoryPath = Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.FullName;
      string fileName = $"{directoryPath}\\Var11-КаспрукАнастасія.txt";

      File.AppendAllLines(fileName,
        new List<string>
        {
          "1",
```

```
            criteriaRelation.ParetoRelation.ToString(),
            "2",
            criteriaRelation.MajorityRelation.ToString(),
            "3",
            criteriaRelation.LexicographicRelation.ToString(),
            "4",
            criteriaRelation.BerezovskyRelation.ToString(),
            "5",
            criteriaRelation.PodinovskyRelation.ToString(),
        });
    }

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

        Console.ForegroundColor = ConsoleColor.Green;
        Console.WriteLine($"   {string.Join(' ', Enumerable.Range(0, relation.Dimension))}");
        for (int i = 0; i < relation.Dimension; i++)
        {
            Console.Write($"{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 PrintCriteriaRelationVectors(CriteriaRelation criteriaRelation, Func<List<int>[][]> printingSelector)
    {
        List<int>[][] toPrint = printingSelector();

        for (int i = 0; i < criteriaRelation.AlternativesCount; i++)
        {
            for(int j = 0; j < criteriaRelation.AlternativesCount; j++)
            {
                Console.WriteLine($"[{i}][{j}]: {string.Join(' ', toPrint[i][j])}");
            }
        }
        Console.WriteLine();
    }
}
```

5. Опис класів. Перелік розроблених функцій на методів.

Клас	Властивість	Опис	Тип значення, що повертає
CriteriaRelation	Evaluations	Значення критеріїв для альтернатив	int[][]
	CriteriasImportance	Упорядкована за спаданням важливості множина критеріїв	IReadOnlyCollection<int>
	CriteriasImportancesClasses	Класи критеріїв впорядковані за зростанням важливості	IReadOnlyCollection<IReadOnlyCollection<int>>
	CriteriasCount	К-сть критеріїв	int
	AlternativesCount	К-сть альтернатив	int
	DeltaVectors	Матриця дельта векторів	List<int>[][]
	SigmaVectors	Матриця сигма векторів	List<int>[][]
	SortedCriteriaRelation	Значення критеріїв для альтернатив, упорядковані за спаданням важливості критеріїв	CriteriaRelation
	ParetoRelation	Відношення Парето	Relation
	MajorityRelation	Мажоритарне відношення	Relation
	LexicographicRelation	Лексикографічне відношення	Relation
	BerezovskyRelation	Відношення Березовського	Relation
	PodinovskyRelation	Відношення Подиновського	Relation

Клас	Ф-ція/Метод	Параметри	Опис	Значення, що повертає
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

6. Висновки.

В будь-яких системах з обмеженими ресурсами виникають задачі їх раціонального розподілу, що, враховуючи велику кількість обмежень та критеріїв вибору, зробити складно. Методи оптимізації застосовуються як у повсякденному житті (розрахунок бюджету, оптимізація витрат), так і при функціонуванні держави, підприємств, об'єктів інфраструктури (вибір оптимального портфелю інвестицій, розрахунок бюджету країни, мінімізація часу виконання проекту, витрати на рекламу тощо).

У даній лабораторній роботі ми мали можливість навчитися розв'язувати задачі багатокритеріальної опитимізації при глобальній порявнюваності критеріїв.

Враховуючи інформацію про порівнюваність критеріїв, шукати оптимальні альтернативи можна по-різному:

- якщо інформація про порівнюваність критерії несуттєва, шукати оптимальні альтернативи можна побудувавши на множині альтернатив відношення Парето;
- якщо критерії рівноважливі, треба будувати мажоритарне відношення;
- якщо на множині критеріїв задано відношення строгого порядку, варто будувати лексикографічне відношення;
- якщо на множині критеріїв задано відношення квазіпорядку, треба будувати відношення Березовського;
- для випадку рівноважливих критеріїв можна побудувати на множині альтернатив відношення Подиновського.

Після побудови відновідного відношення на множині альтернатив, далі шукати на ньому оптимальні альтернативи наййзручніше використовуючи принципи домінування або блокування.

У результаті ми навчилися шукати оптимальні альтернативи для відношень переваги при глобальній порівнюваності критеріїв та змогли знайти рішення задачі у відповідності з варіантом.