**ՃԱՐՏԱՐԱՊԵՏՈՒԹՅԱՆ ԵՎ ՇԻՆԱՐԱՐՈՒԹՅԱՆ ՀԱՅԱՍՏԱՆԻ ԱԶԳԱՅԻՆ ՀԱՄԱԼՍԱՐԱՆ**

Ինֆորմատիկայի, հաշվողական տեխնիկայի և

կառավարման համակարգերի ամբիոն

Հաշվողական տեխնիկայի և

կառավարման Ֆակուլտետ

**ԿՈՒՐՍԱՅԻՆ ՆԱԽԱԳԾԻ**

**ՀԱՇՎԵԲԱՑԱՏՐԱԳԻՐ**

\_Կ 73\_ խմբի ուսանողուհի

Մանասյան Արուսյակ Դավիթի\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ազգանուն, անուն, հայրանուն` լրիվ)

Համակարգային վերլուծություն և գործողությունների հետազոտում

(կրթական մոդուլի դասիչը և անվանումը)

Կուրսային նախագիծը թույլատրվում է պաշտպանության.

Ղեկավար` \_\_\_ \_Ուզունյան Շ․Գ․\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ազգանուն Ա.,Հ.) (Ստորագրություն)

Ամբիոնի վարիչ` \_\_ Մարկոսյան.Մ.Վ.\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Ազգանուն Ա.,Հ.) (Ստորագրություն)

ԵՐԵՎԱՆ 2020թ.

Բովանդակություն

[**Կուրսային նախագծի հանձնարարությունը** 3](#_Toc41481267)

[**Խնդրի նկարագրությունը և ձևակերպումը** 4](#_Toc41481268)

[**Օպտիմալացման խնդրի ձևակերպումը** 6](#_Toc41481269)

[**Գերիշխող տողերի և սյուների որոշում** 6](#_Toc41481270)

[**Օպտիմալ ստրատեգիաների որոշումը(minmax)** 7](#_Toc41481271)

[**Խնդրի մաթ. մոդելի կառուցումը և դրա օգնությամբ մաթեմատիկական խնդրի ձևավորում** 9](#_Toc41481272)

[**Խնդրի լուծման մեթոդի հիմնավորումը և ձևակերպումը** 11](#_Toc41481273)

[**Սիմպլեքս մեթոդ** 11](#_Toc41481274)

[**ԳԾ երկակի խնդիր** 15](#_Toc41481275)

[**Երկակի սիմպլեքս մեթոդ** 16](#_Toc41481276)

[**Խնդրի լուծման բլոկ-սխեման** 18](#_Toc41481277)

[**Մատրիցի կառուցման բլոկ-սխեման** 18](#_Toc41481278)

[**Գերիշխող տողերի և սյուների որոշման բլոկ-սխեման** 19](#_Toc41481279)

**-ի բլոկ-սխեման՝** 21

[**Խնդրի լուծման ալգորիթմը և դրա իրականացումը (ծրագրային մաս)** 23](#_Toc41481281)

[**Լուծման թվային արժեքները** 26](#_Toc41481282)

[**Առաջին խաղացողի ստրատեգիաների հավանականությունները** 26](#_Toc41481283)

[**Երկրորդ խաղացողի ստրատեգիաների հավանականությունները** 26](#_Toc41481284)

[**Արդյունքների մեկնաբանությունը** 28](#_Toc41481285)

[**Օգտագործված գրականության ցանկ** 30](#_Toc41481286)

[**Հավելված** 30](#_Toc41481287)

# **Կուրսային նախագծի հանձնարարությունը**

Խաղին մասնակցող երկեւ կողմերը միաժամանակ ընտրում են 1,2,……..7 ամբողջ թվերից մեկը:Եթե առաջին խաղացողը ընտրել է x թիվը,իսկ երկրորդը y թիվը, ապա առաջինը շահում է x-y միավոր,եթե x≥y և պարտվում է x+y,եթե x<y:

Որոշել խաղացող կողմերի օպտիմալ ստրատեգիաները և խաղի արժեքը:

1. Կուրսային նախագծի հանձնարարությունը (առաջադրանքը)
2. Խնդրի նկարագրությունը և ձևակերպումը
3. Խնդրի լուծման մեթոդի հիմնավորւմը և ձևակերպումը
4. Խնդրի լուծման բլոկ-սխեման
5. Խնդրի լուծման ալգորիթմը և դրա իրականացումը (ծրագրային մաս)
6. Լուծման թվային արժեքները
7. Արդյունքների մեկնաբանությունը
8. Օգտագործված գրականության ցանկ
9. Հավելված/ներ (պարտադիր չէ)

# **Խնդրի նկարագրությունը և ձևակերպումը**

Խաղերի տեսության խնդիրն է[[1]](#footnote-1) հակառակ կողմերի համար մշակել գործելակերպի մեթոդներ: Խաղին մասնակցող կողմերին անվանում են խաղացողներ,երբ խաղացող կողմերը երկուսն են, այդպիսի խաղն անվանում են զույգի խաղ:Հակառակ դեպքում բազմության խաղ: Խաղացող կողմերի ստրատեգիա ասելով հասկանում ենք խաղացող կողմի կանոններից ընտրված համախումբը:

Օպտիմալ ստրատեգիա ասելով հասկանում ենք խաղացող կողմի ստրատեգիաների համախումբը , որը ապահովում է առավելագույն շահում կամ նվազագույն կորուստ:

Խաղն անվանում են 0-ական գումարով խաղ , եթե կողմերից մեկը շահում է այնքան միավոր,որքան կորցնում է մյուս կողմը: Խաղն անվանում են լրիվ ինֆորմացիայով, եթե խաղացող կողմերին հայտնի են հակառակորդի բոլոր ստրատեգիաները և կատարած բոլոր քայլերը: Խաղի վերլուծությունը բերում է նրան , որ խաղացող կողմերը պետք է ընտրեն իրենց մաքուր ստրատեգիաները ինչ-որ հավանակա- նությամբ:Այդ դեպքում ասում են, որ խաղը ընթանում է խառը ստրատեգիաներով: Այսինքն իրենց մաքուր ստրատեգիաները ընտրում են ինչ-որ հավանականությամբ: խաղի լուծումը մեծ -երի և -երի դեպքում՝ սկզբունքորեն կարելի է կառուցել հավասարումների համակարգը և նրանց լուծումից ստանալ խաղի արդյունքը:Պարզվում է , որ խաղի լուծումը կարելի է բերել գծային ծրագրավորման խնդրի,որի լուծման համարր կարող ենք կիրառել ԳԾ լուծման,որևէ մեթոդ:

Եթե ընտրենք խաղացողների ստրատեգիաները, նշանակելով խաղացողի ստրատեգիաների հավանականությունները ՝ , խաղացողինը՝, ապա լուծել խաղը նշանակում է գտնել օպտիմալ ստրատեգիաների հավանականությունները:

Խաղի լուծումը կատարվում է հետևյալ հերթականությամբ՝

1. *Որոշում ենք մատրիցում կան գերիշխող տողեր կամ սյուներ,եթե կան պահում ենք դրանք, իսկ մնացածը անտեսում : Դրանով պարզեցնում ենք խաղի մատրիցը,*
2. *Որոշում ենք թամբի կետով խաղ է , թե ոչ,եթե թամբի կետով խաղ չէ անցնում ենք խառը ստրատեգիաներով խաղի, եթե թամբի կետով խաղ է ,ապա խնդիրը համարում ենք լուծված,*
3. *Ձևափոխված մատրիցի բոլոր էլեմենտները, եթե մատրիցը ունի բացասական տարրեր դարձնում ենք դրական թվեր՝ ավելացնելով ինչ-որ M մեծություն,*
4. *Կառուցում ենք ԳԾ խնդիրը,*
5. *Լուծում ենք ԳԾ խնդիրը և որոշում -ը և -ը,և քանի որ*

*= (=),* կգտնենք *-*ն,այսինքն խաղի արժեքը, -ից կհանենք այն M մեծությունը կստանանք խաղի իրական արժեքը: Այնուհետև կգտնենք խաղացողների ստրատեգիաների հավանականությունները՝ =,, Ճիշտ նույնն էլ մյուս խաղացողի համար:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [1,1] | [1,2] | ….. | ….. | …… | …… | [1,j] |
| ….. | ….. | ….. | ….. | ….. | ….. | ….. |
| ….. | ….. | ….. | ….. | ….. | ….. | ….. |
| ….. | ….. | ….. | ….. | ….. | ….. | ….. |
| ….. | ….. | ….. | ….. | ….. | ….. | ….. |
| ….. | ….. | ….. | ….. | ….. | ….. | ….. |
| [,1] | [,2] | ….. | ….. | ….. | ….. | [] |

Մեր խնդրի լուծման համար նախ և առաջ անհրաժեշտ է կառուցել խաղի մատրիցը: Մատրիցի կառուցումը կատարվում է հետևյալ քայլերի հաջորդականությամբ: Նախ և առաջ միմյանց հետ համեմատվում են -երը և -ները և ,եթե x≥ ,ապա առաջին խաղացողը շահում է **-** միավոր ,իսկ երկրորդ խաղացողը կորցնում է այդքան միավոր,հակառակ դեպքում ,երբ  **<** առաջին խաղացողը կորցնում է  **+** միավոր, երկրորդ խաղացողը շահում է այդքան միավոր

Մատրիցում տարրերը պետք է դասավորվեն ըստ իրենց ինդեքսների:Քանի որ խաղին մասնակցող երկու կողմերը միաժամանակ ընտրում են 1,2,……..7 ամբողջ թվերից մեկը,հետևաբար ,ապա լրացվում է մատրիցի [1,1] վանդակը եթե 3 լրացվում է մատրիցի [2,3] վանդակը և այսպես շարունակ մինչև մատրիցը ամբողջությամբ լրացվի՝

Վճարման մատրիցը ստանում է հետևյալ տեսքը՝

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Խաղացողներ | B1 | B2 | B3 | B4 | B5 | B6 | B7 |
| A1 | 0 | 3 | 4 | 5 | 6 | 7 | 8 |
| A2 | 1 | 0 | 5 | 6 | 7 | 8 | 9 |
| A3 | 2 | 1 | 0 | 7 | 8 | 9 | 10 |
| A4 | 3 | 2 | 1 | 0 | 9 | 10 | 11 |
| A5 | 4 | 3 | 2 | 1 | 0 | 11 | 12 |
| A6 | 5 | 4 | 3 | 2 | 1 | 0 | 13 |
| A7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Այստեղ կապույտ օղակներում նշված են առաջին խաղացողի շահումները, իսկ դեղին օղակներում նշված են երկրորդ խաղացողի շահումները:Մեկի շահումը կորուստ է հանդիսանում մյուսի համար:

### **Օպտիմալացման խնդրի ձևակերպումը**

Առաջին խաղացողը ընտրում է իր ստրատեգիաները այնպես, որ ստանա առավելագույն շահում , իսկ երկրորդը այնպես է ընտրում իր ստրատեգիաները , որ առաջինի շահումը հասցնի նվազագույնի : Այսպիսով նախ նայում ենք ունե՞նք արդյոք գերիշխող տողեր և սյուներ ,ապա ընտրում ենք խաղի օպտիմալ ստրատեգիաները(minmax):

### **Գերիշխող տողերի և սյուների որոշում**

Գերիշխող**[[2]](#footnote-2)** են անվանում այն ստրատեգիաները ,որոնք ապահովում են խաղի առավելագույն շահումը այլ ստրատեգիաների նկատմամբ կամ արժեքը չեն փոխում:

Եթե խաղի մատրիցում i-րդ տողի բոլոր անդամները մեծ են կամ հավասար j-րդ տողի բոլոր անդամներից ,ապա i տողը գերիշխում է j տողի նկատմամբ :Այդ պատճառով A խաղացողը չի օգտագործում j-րդ ստրատեգիան, քանի որ i–րդ ստրատեգիայի դեպքում անկախ նրանից թե B խաղացողըը ինչպես է խաղում :Նույն ձևով ,եթե i–րդ սյան բոլոր անդամները մեծ են կամ հավասար j–րդ սյան բոլոր անդամներից ,ապա j -րդ սյունն է գերիշխում սյան նկատմամբ:Այդ պատճառով էլ B խաղացողը չի օգտագործում իր i-րդ ստրատեգիան ,քանի որ նրա պարտությունը j–րդ ստրատեգիայի դեպքում ավելի քիչ է քան i-րդ ստրատեգիայի դեպքում , անկախ նրանից թե A խաղացողըը ինչպես է խաղում :

Պահելով գերիշխող տողերը և սյուները հեռացնում ենք մնացածը,այսպես պարզեցնեւմ ենք մատրիցը հետագա հաշվարկներ հեշտացնելու համար:Այնուհետև մատրիցի բոլոր բացասական էլեմենտները անհրաժեշտ է դարձնել դրական գումարելով ինչ-որ M թիվ: Մենք կընտրենք մատրիցի ամենափոքր թիվը: Քանի որ մատրիցում ամենափոքր թիվը մոդուլով ամենամեծ թիվն է: Հետևաբար M թիվը ամենափոքրը ընտրելով,ապա մատրիցի բացասական տարրերից հանելով ստանում եմ ավելի փոքր թվեր,որոնց հետ հեշտ է գործողություններ կատարել:

Մեր տրված խնդրում վճարման մատրիցում բացակայում են գերիշխող տողերը, սյուները և բացասական տարրերը : Հետևաբար մենք անցում ենք կատարել խաղի օպտիմալ ստրատեգիաներ որոշման փուլին:

### **Օպտիմալ ստրատեգիաների որոշումը(minmax)**

Ամեն խաղում կողմերը կարող են ընտրել իրենց ստրատեգիաները**[[3]](#footnote-3)** ելնելով նրանից,որ նվազագույն կորուստներ կրեն խաղի ընթացքում: Դրանից ելնելով առաջին խաղացողը գնահատում է իր –րդ ստրատեգիայի ընտրման դեպքում ստացած նվազագույն շահումը:

Եթե առաջին խաղացողը ընտրում է ստրատեգիան:Նա պետք է գնահատի հակառակորդի ընտրված ստրատեգիաների դեպքում,իր նվազագույն շահումը ,որը նշանակենք ՝ :Հետևաբար դեպքում ստրատեգիայի նվազագույն շահումն է՝

*,*

-ով նշանակենք երկրորդ խաղացողի j-րդ ստրատեգիան

Մեր խաղի մատրիցում ստացվում է հետևյալ պատկերը՝

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Խաղացողներ | B1 | B2 | B3 | B4 | B5 | B6 | B7 |  |
| A1 | 0 | 3 | 4 | 5 | 6 | 7 | 8 | 0 |
| A2 | 1 | 0 | 5 | 6 | 7 | 8 | 9 | 0 |
| A3 | 2 | 1 | 0 | 7 | 8 | 9 | 10 | 0 |
| A4 | 3 | 2 | 1 | 0 | 9 | 10 | 11 | 0 |
| A5 | 4 | 3 | 2 | 1 | 0 | 11 | 12 | 0 |
| A6 | 5 | 4 | 3 | 2 | 1 | 0 | 13 | 0 |
| A7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 0 |
|  | 6 | 5 | 5 | 7 | 9 | 11 | 13 |  |

*=* 0

*=* 5

-ն ցույց են տալիս առաջին և երկրորդ խաղացողների երաշխավորված շահումը:Եթե խաղացող կողմերը ընտրում են համապատասխան ստրատեգիաները ,ապա համոզված են , որ դրանից ավելի շահել կամ կորցնել չեն կարող:Այս մոտեցում խաղերի տեսության մեջ անվանում են մինիմաքսի սկզբունք:

ստրատեգիան անվանում են խաղի ստորին սահման,իսկ -ի ստրատեգիան վերին սահման:Այսպիսով-ն անվանում են խաղի վերին և ստորին արժեքներ , եթե այդ խաղն անվանում են թամբի կետով խաղ:Թամբի կետով խաղում թամբի կետին համապատասխանող ստրատեգիաները անվանում են օպտիմալ ստրատեգիաներ:Թամբի կետով խաղի դեպքում կողմերը պետք է ընտրեն թամբի կետին համապատասխան ստրատեգիաները: Եթե խաղացող կողմերից, որևիցե մեկը հրաժարվի իր թամբի կետին համապատասխանող ստրատեգիայից, չի կարող ավելին շահել:

Խնդրում : Մեր խաղը թամբի կետով չէ,հետևաբար անցում ենք կատարում խառը ստրատեգիաներով խաղի:

# **Խնդրի մաթ. մոդելի կառուցումը և դրա օգնությամբ մաթեմատիկական խնդրի ձևավորում**

Մաթեմատիկակն մոդելի կառուցման**[[4]](#footnote-4)** առաջին փուլում ձևակերպվում է գործողության նպատակը, այսինքն այն պահանջների ձևակերպումը , որը դրվում է հետազոտվող գործողության վրա:

Այսպիսով իմ խնդրում առաջին խաղացողը ընտրում է իր ստրատեգիաները այնպես, որ ստանա առավելագույն շահույթ, իսկ երկրորդը այնպես է ընտրում իր ստրատեգիաները , որ առաջինի շահումը հասցնի նվազագույնի: Այսինքն խնդրի հիմնական նպատակը այնպես անել որ խաղում չպարտվել: Խնդրի նպատակը ձևակերպելուց հետո անցնում ենք նպատակային ֆունկցիայի կառուցմանը,որը գործողության նպատակի մաթեմատիկական նկարագրումն է ֆունկցիայի տեսքով:

Քանի որ առաջին խաղացողը ընտրում է իր ստրատեգիաները այնպես , որ ստանա առավելագույն շահում , ապա նպատակային ֆունկցիան կընդունի հետևյալ տեսքը ՝

=

() =++++++ → min

Քանի որ,եթե () → min,ապա հետևաբար առաջին խաղացողը ստանում է առավելագույն շահում:

Սահմանափակումներ ընդունում են հետևյալ տեսքը՝

+2+3+4+5+6 ≥ 1

3++2+3+4+5 ≥ 1

4+5++2+3+4 ≥ 1

5+6+7++2+3≥ 1

6+7+8+9++2≥ 1

7+8+9+10+11+ ≥ 1

8+9+10+11+12+13 ≥ 1

Որտեղ =, երկրորդ խաղացողի ստրատեգիաների հավանականություններն են,իսկ խաղի արժեքը:

Քանի որ երկրորդ խաղացողը նվազագույնի պետք է հասցներ առաջինի շահումը,նպատակային ֆունկցիան ընդունում է հետևյալ տեսքը՝

I(y) = -> max

Քանի որ,եթե () → max,ապա հետևաբար առաջին խաղացողը ստանում է նվազագույն շահում:

Սահմանափակումներ ընդունում են հետևյալ տեսքը՝

3y2+4y3+5y4+6y5+7y6+8y7 ≤ 1

y1+5y3+6y4+7y5+8y6+9y7 ≤ 1

2y1+y2+7y4+8y5+9y6+10y7 ≤ 1

3y1+2y2+y3+9y5+10y6+11y7 ≤ 1

4y1+3y2+2y3+y4+11y6+12y7 ≤ 1

5y1+4y2+3y3+2y4+y5+13y7 ≤ 1

6y1+5y2+4y3+3y4+2y5+y6 ≤ 1

Որտեղ y=,առաջին խաղացողի ստրատեգիաների հավանականություններն են,իսկ խաղի արժեքը:Այսպիսով մատրիցային խաղի լուծումը բերվում է գծային ծրագրավորման երկակի խնդրի զույգի:Քանի որ երկրորդ խաղացողի սահմանափակումներում արհեստական բազիսի անհրաժեշտություն չկա խնդրի լուծումը կշարունակեմ երկրորդ խաղացողի համար,ապա կստանամ առաջին խաղացողի համար: Եվ օգտվելով սիմպլեքս մեթոդից կկառուցեմ սիմպլեքս աղյուսակը առաջին խաղացողի համար:

# **Խնդրի լուծման մեթոդի հիմնավորումը և ձևակերպումը**

### **Սիմպլեքս մեթոդ**

Խաղը թամբի կետով չէ, հետևաբար անցում ենք կատարել խառը ստրատեգիաներով խաղի և խնդրի լուծման համար օգտվել ենք սիմպլեքս**[[5]](#footnote-5)**, մեթոդից:

Քանի որ խնդրում առկա փոփոխականները շատ են ,ապա հավասարումների համակարգի լուծումը թվային մեթոդներով կարող է հանգել հաշվողական մեծ դժվարությունների այդ պատճառով խնդրի լուծման համար օգտվել եմ սիմպլեքս մեքոդից :

Սիմպլեքս մեթոդի յուրաքանչյուր քայլ բերում է նոր պլանի,որին համապատասխանում է ԳԾ խնդրի նպատակայի ն ֆունկցիայի ավելի փոքր(մեծ) արժեք,քան նախորդ պլանում:Այս գործընթացը շարունակվում է մինչև օպտիմալ պլանի որոշումը:Քանի որ սիմպլեքս մեթոդը կապված է հենային պլանի ուսումնասիրության հետ ,նախ և առաջ պետք է կառուցել հենային պլանը:

Բայց նախքան հենային պլանի կառուցմանը անցնելը տանք բազիս հասկացությունը:Ենթադրենք տրված է միևնույն չափողականություն ունեցող վեկտորների համակարգ

(1)

Անջատենք այդ համակարգից

(2)

ենթահամակարգը: Այստեղ թվեր են 1,2,…,n ինդեքսների բազմությունից:

(2) վեկտորների ենթահամակարգը անվանում են (1) համակրգի բազիս,եթե (2) վեկտորները գծորեն անկախ են,իսկ (1) համակարգի ցանկացած վեկտոր կարելի է ներկայացնել նրանց գծային կոմբինացիայի տեսքով:

Խնդիրը սիմպլեքս մեթոդով լուծելու համար անհրաժեշտ է բերել կանոնական տեսքի: Դրա համար բոլոր անհավասարումերը անհրաժեշտ է դարձնել հավասարումներ: Կանոնական խնդրի անցնելը իրականացնենք նոր փոփոխականների ներմուծման միջոցով:

I(y) = -> max

I(y)=

3y2+4y3+5y4+6y5+7y6+8y7 ≤ 1 🡪 3y2+4y3+5y4+6y5+7y6+8y7+y8 = 1

y1+5y3+6y4+7y5+8y6+9y7 ≤ 1 🡪 y1+5y3+6y4+7y5+8y6+9y7+y9 = 1

2y1+y2+7y4+8y5+9y6+10y7 ≤ 1 🡪 2y1+y2+7y4+8y5+9y6+10y7+y10 = 1

3y1+2y2+y3+9y5+10y6+11y7 ≤ 1 🡪 3y1+2y2+y3+9y5+10y6+11y7+y11 = 1

4y1+3y2+2y3+y4+11y6+12y7 ≤ 1 🡪 4y1+3y2+2y3+y4+11y6+12y7+y12 = 1

5y1+4y2+3y3+2y4+y5+13y7 ≤ 1 🡪 5y1+4y2+3y3+2y4+y5+13y7+y13 = 1

6y1+5y2+4y3+3y4+2y5+y6 ≤ 1 🡪 6y1+5y2+4y3+3y4+2y5+y6+y14 = 1

Սահմանափակումները կանոնական տեսքի բերելուց հետո ,կառուցում ենք առաջին հենային պլանը, քանի որ միավոր վեկտորներ են, ապա

ընտրենք բազիսային ,իսկ ազատ փոփոխականներ և վերջինները հավասարեցնելով զրոյի ,կստանանք՝

( ,)

սկզբնական հենային պլանը, որին համապատասխանում է հետևյալ վերլուծությունը՝

սկզբնական հենային պլանը ըստ օպտիմալության ստուգելու համար կառուցենք առաջին սիմպլեքս աղյուսակը և հաշվենք I()արժեքը գնահատականները: Առաջին սիմպլեքս աղյուսակ՝

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Բազիս |  | B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| y8 | 0 | 1 | 0 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| y9 | 0 | 1 | 1 | 0 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| y10 | 0 | 1 | 2 | 1 | 0 | 7 | 8 | 9 | 10 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| y11 | 0 | 1 | 3 | 2 | 1 | 0 | 9 | 10 | 11 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| y12 | 0 | 1 | 4 | 3 | 2 | 1 | 0 | 11 | 12 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| y13 | 0 | 1 | 5 | 4 | 3 | 2 | 1 | 0 | 13 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| y14 | 0 | 1 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| I() |  |  | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Համաձայն օպտիմալության պայմանի, քանի որ բազիս վեկտորի գնահատականների մեջ կան բացասական գնահատականներ՝ հենային պլանը օպտիմալ չէ : Այն կարելի է լավացնել՝ ներմուծելով բազիս այն վեկտորը ,որին համապատասխանում է :

վեկտորների վերլուծության գործակիցներում բացասական տարրերի առկայությունը հնարավորություն է տալիս որոշել այնպիս մեծություններ, որոնք բազիսից կանջատեն գոնե մեկ վեկտոր: Որպես վճռող տարր վերցնում ենք այն սյունակից , որին պատկանող գործակիցը ամենամեծն է մոդուլով:Այդ սյունակըն է, քանի որ մոդուլով ամենամեծ գործակիցը 13-ն է: Դրանից հետո, որպեսզի իմանանք,որն է վճռող տարրը կատարում ենք հետևյալ քալերը՝

(,,,,,)=

Այսպիսով վճռող տարրը 13-ն է, որը գտնվում է ուղղորդող սյան և ուղղորդող տողի հատման կետում: Հետևաբար վեկտորը մտնում է բազիս, իսկ վեկտորը դուրս է գալիս բազիսից: Եվ այսպես շարունակ մինչև ինդեքսային տողում չլինեն բացասական տարրեր: Վերջին սիմպլեքս աղյուսակ՝

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Բազիս |  | B |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| y2 | 1 | 70/1063 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1133/4252 | -3119/12756 | 35/12756 | 7/4252 | 7/6378 | 5/6378 | 40/1063 |
| y3 | 1 | 140/3189 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 35/3189 | 3259/19134 | -1577/9567 | 7/6378 | 7/9567 | 5/9567 | 80/3189 |
| y4 | 1 | 35/1063 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 35/4252 | 35/12756 | 403/3189 | -132/1063 | 7/12756 | 5/12756 | 20/1063 |
| y5 | 1 | 28/1063 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 7/1063 | 7/3189 | 7/6378 | 107/1063 | -635/6378 | 1/3189 | 16/1063 |
| y6 | 1 | 70/3189 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 35/6378 | 35/19134 | 35/38268 | 7/12756 | 3203/38268 | -3179/38268 | 40/3189 |
| y7 | 1 | 20/1063 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5/1063 | 5/3189 | 5/6378 | 1/2126 | 1/3189 | 457/6378 | -129/2126 |
| y1 | 1 | 57/1063 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | -503/2126 | 280/3189 | 140/3189 | 28/1063 | 56/3189 | 40/3189 | 217/2126 |
| I() |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 70/1063 | 70/3189 | 35/3189 | 7/1063 | 14/3189 | 10/3189 | 160/1063 |

Ինդեքսային տողում չկան բացասական գնահատականներ, հետևաբար այս աղյուսակը որոշում է խնդրի օպտիմալ պլանը:

Օպտիմալ պլանը կարող ենք գրել հետևյալ կերպ՝

1= 57/1063, 2 = 70/1063, 3 = 140/3189, 4 = 35/1063, 5 = 28/1063, 6 = 70/3189, 7 = 20/1063

= 1\*57/1063 + 1\*70/1063 + 1\*140/3189 + 1\*35/1063 + 1\*28/1063 + 1\*70/3189 + 1\*20/1063 = 280/1063

Օգտագործելով խնդրի վերջին իտերացիան կարող ենք գտնել երկակի խնդրի օպտիմալ պլանը: Դրա համար անհրաժեշտ է օգտվել երկակի սիմպլեքս մեթոդից:

### **ԳԾ երկակի խնդիր**

ԳԾ յուրաքանչյուր խնդիր սերտ կապված է մի այլ ԳԾ խնդրի հետ,որոնք միմյանց համարժեք են : ԳԾ խնդրի զույգի համարժեքությունը նշանակում է նրանցից յուրաքանչյուրի օպտիմալ պլանից հետևում է մյուսի օպտիմալ լուծումը:ԳԾ խնդրի զույգը անվանում են փոխադարձ երկակի խնդիրներ:Եթե ԳԾ փոխադարձ երկակի [[6]](#footnote-6)խնդիրների զույգից,որևէ մեկի նպատակային ֆունկցիան սահմանափակ չէ իր թույլատրելի արժեքների տիրույթում,ապա մյուսը չունի թույլատրելի պլան:

ԳԾ սկզբնական և երկակի խնդիրների միջև կապը պարզաբանելու նպատակով դիտարկենք ԳԾ հիմնական խնդիրը՝

,

, ( 1)

……………………………………..

,

1. Խնդրի երկակի խնդիր անվանում են փոփոխականներից հետևյալ ԳԾ խնդիրը`

,

, ( 2)

……………………………………..

,

ԳԾ (1) և (2) խնդիրների միջև կապը հետևյալն է՝

* (2) խնդրում սահմանափակման գործակիցների մատրիցը (1) խնդրի սահմանափակման տրանսպոնացված մատրիցն է՛
* (1) և (2)խնդիրներում սահմանափակումները ունեն հակառակ,իսկ փոփոխականները միևնույն նշաններ,
* (1) խնդրից (2)-ին անցնելիս B և c գործակիցների դերերը փոխվում են,
* ֆունկցիաների էքստրեմումների իմաստը միմյանց հակադարձ են:

Նշված ձևով կառուցված ԳԾ խնդիրները անվանում են փոխադարձ երկակի խնդիրներ:

Փոխադարձ երկակի խնդիրների ուսումնասիրությունը պայմանավորված է նրանով,որ գործնական հետազոտությունների ժամանակ հաճախ ավելի հեշտ է լուծել երկակի քան սկզբնական խնդիրը:

### **Երկակի սիմպլեքս մեթոդ**

Երկակի սիմպլեքս մեթոդը , ինչպես և սիմպլեքս մեթոդը կիրառվում է ԳԾ կանոնական տեսքի խնդիրների օպտիմալ պլանը գտնելու համար:Երկակի սիմպլեքս մեթոդը տարբերվում է սիմպլեքս մեթոդից նրանով,որ սահմանափակումներում ազատ անդամի նշանի վրա ոչ բացասական լինելու պահանջ չի դրվում:

Երկակի սիմպլեքս մեթոդում, ինչպես և սիմպլեքս մեթոդում,յուրաքանչյուր քայլում անցում է կատարվում բազմանիստ որոշման տիրույթի մի անկյունային կետից մյուսին, այնպես որ վերջավոր քայլեր կատարելով որոշվում է օպտիմալ պլանը կամ բացահայտվում է ,որ խնդիրը լուծում չունի:Սակայն ի տարբերություն սիմպլեքս մեթոդի՝ այստեղ յուրաքանչյուր քայլում որոշված լուծումը , ընդհանրապես թույլատրելի լուծում չէ այն իմաստով , որ բավարարելով սահմանափակումներին չեն ապահովում պայմանը:

Երկակի սիմպլեքս մեթոդում[[7]](#footnote-7) առաջին սիմպլեքս աղյուսակը կառուցվում է այնպես,ինչպես սիմպլեքս մեթոդում:Եթե սիմպլեքս աղյուսակի B սյան կոորդինատները ոչ բացասական են և գնահատականները բավարարում են օպտիմալության պայմանին ,սկզբնական խնդիրը լուծված է: Եթե սիմպլեքս աղյուսակի B սյան կոորդինատներից մեկը կամ մի քանիսը բացասական են,ապա հաջորդ սիմպլեքս աղյուսակները կառուցվում են հետևյալ ձևով:

Ընտրվում է B սյան բացասական կոորդինատներից այն ,որը բացարձակ արժեքով ամենամեծ է Եթե այդպիսինները մեկից ավելի են ,ապա ընտրվում է նրացից, որևէ մեկը , ընդունենք այն : ընտրությամբ որոշվում է բազիսից հեռացվող վեկտորը: Բազիս ներմուծվող վեկտորը որոշվում է պայմանից ,որտեղ ըստ այն j-երի , որոնց համար Հաջորդ սիմպլեքս աղյուսակը կառուցվոււմ է այնպես ինչպես սիմպլեքս մեթոդում:Այս գործընթացը շարունակվում է այնքան մինչև B սյան կոորդինատներից հեռացվեն բացասականները: Կառուցված պլանը կլինի սկզբնական, ինչպես նաև երկակի խնդրի օպտիմալ պլան: Եթե ինչ-որ քայլում k  կոորդինատը բացասական է , իսկ k-րդ տողի բոլոր տարրերը դրական են,նշանակում է, որ սկզբնական խնդիրը լուծում չունի:

Երկակի խնդրի օպտիմալ պլանը մեր խնդրում կլինի՝

1 = 70/1063, 2 = 70/3189, 3 = 35/3189, 4 = 7/1063, 5 = 14/3189, 6 = 10/3189, 7 = 160/1063

= 1\*70/1063+1\*70/3189+1\*35/3189+1\*7/1063+1\*14/3189+1\*10/3189+1\*160/1063 = 280/1063

Խաղի արժեքը կլինի :

# **Խնդրի լուծման բլոկ-սխեման**

### **Մատրիցի կառուցման բլոկ-սխեման**

SIZE

X[

X[j]=1,7

X[ **≥** X[j]

Mat[=x[

Mat[=x[

### **Գերիշխող տողերի և սյուների որոշման բլոկ-սխեման**

Mat[row][col]

domcol[col],domrow[row]

Int i=0;i<row;i++

Int k=0;k<row;k++

Int count=0;

Int j=0;j<col;j++

Mat[i][j]>=Mat[k][j]

&& k != i

**ոչ**

**այո**

Հեռացնել k-րդ տողը

count++

count==col

**ոչ**

**այո**

domrow[i]=k

Int j=0;j<row;j++

Int l=0;l<row;l++

Int count=0;

Int i=0;i<col;i++

Mat[i][j]<=Mat[i][l]

&& l!= i

count++

count==row

Հեռացնել l-րդ տողը

domcol[l]=l

### **-ի բլոկ-սխեման՝**

Mat[row][col]

j:=0,col

min[i]=Mat[i][0]

i:=0,row

min[row],max[col]

min[i]

**ոչ**

Mat[i][j]<min[i]

**այո**

min[i]=Mat[[i][j]

i:=0,row

max[j]=Mat[0][j]

max[j]]

j:=0,col

**ոչ**

Mat[i][j]>max[j]

**այո**

min[i]=Mat[[i][j]

maxmin,minmax

j:=0,col

max[j]<maxmin

minmax=min[0]

i:=0,row

min[i]>minmax

minmax=min[0]

maxmin=max[0]

**ոչ ոչ**

**այո այո**

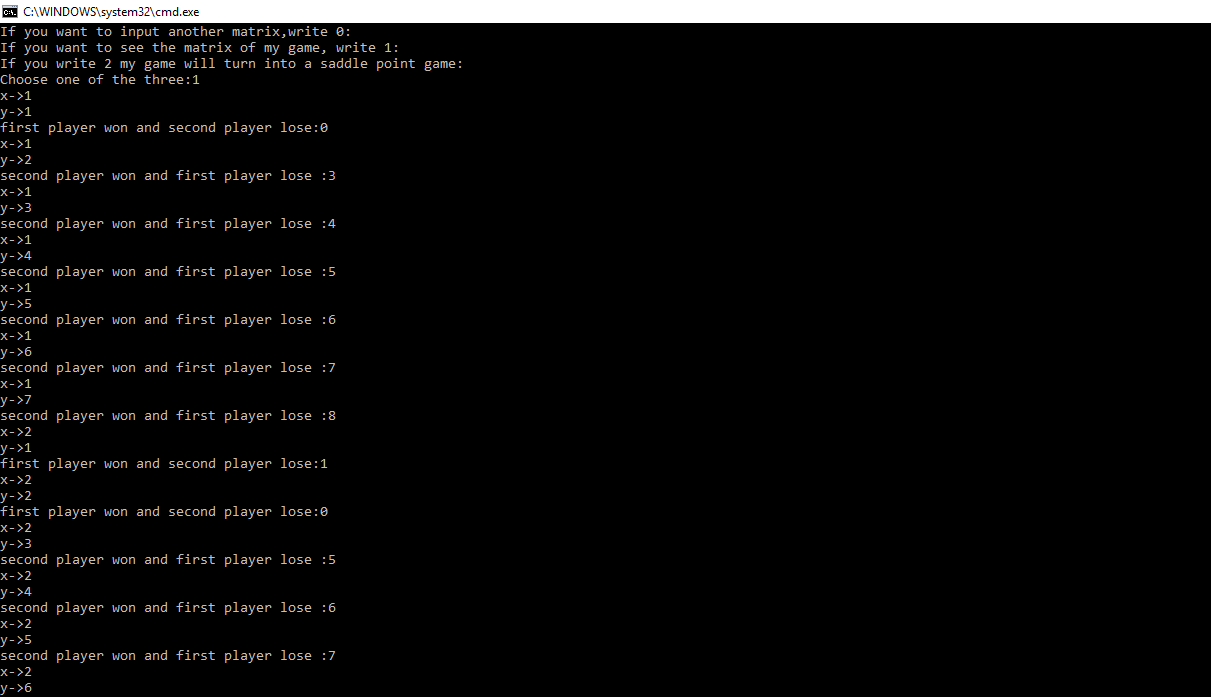
maxmin=max[0]

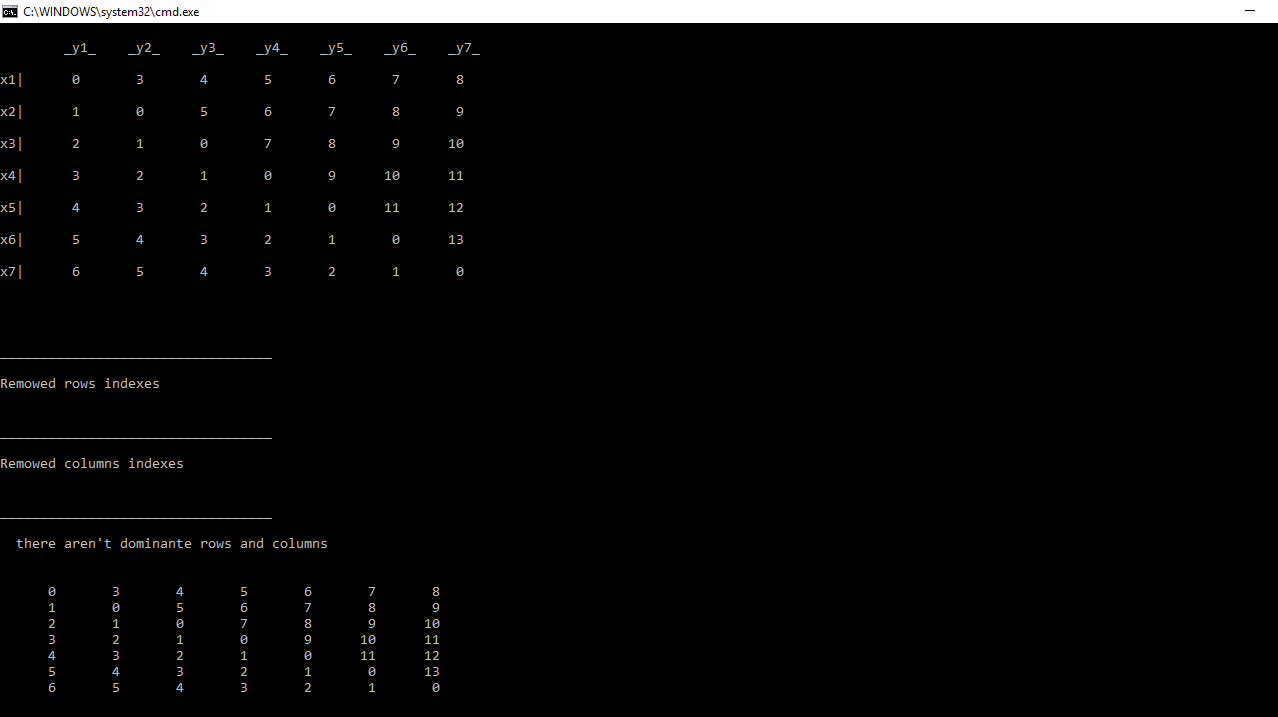
minmax

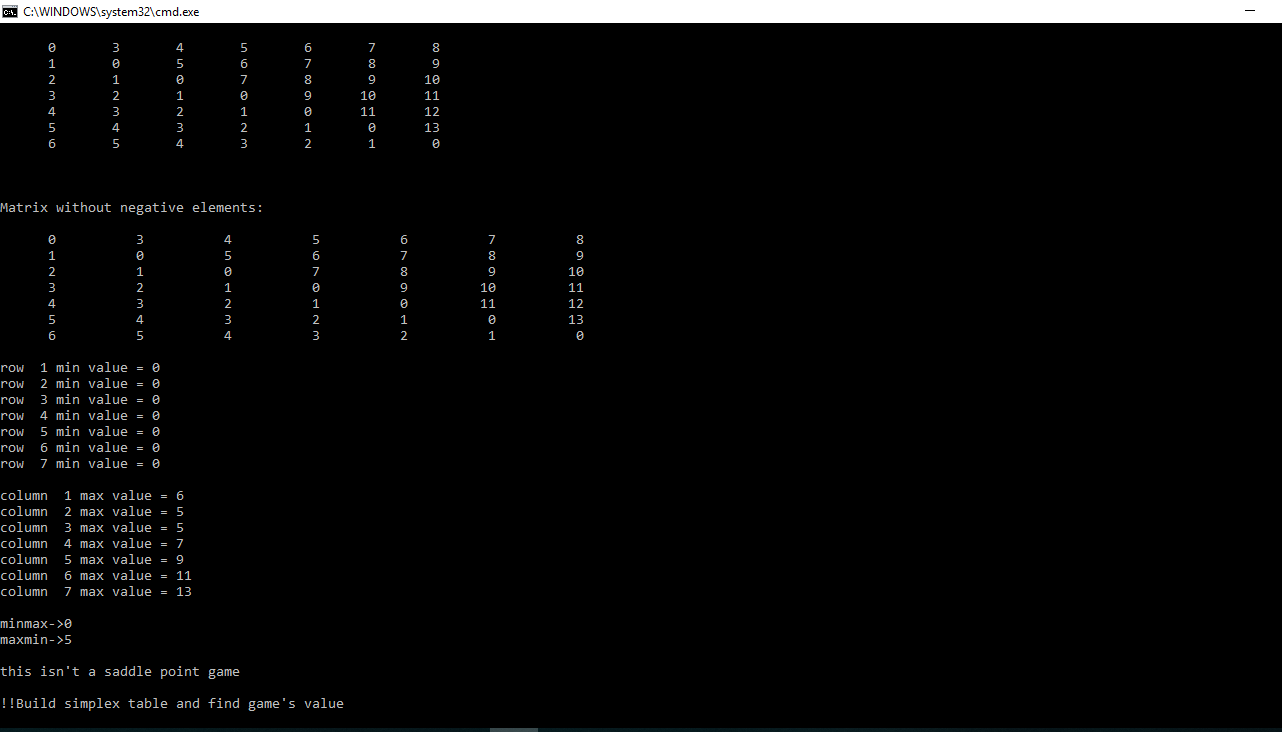
maxmin

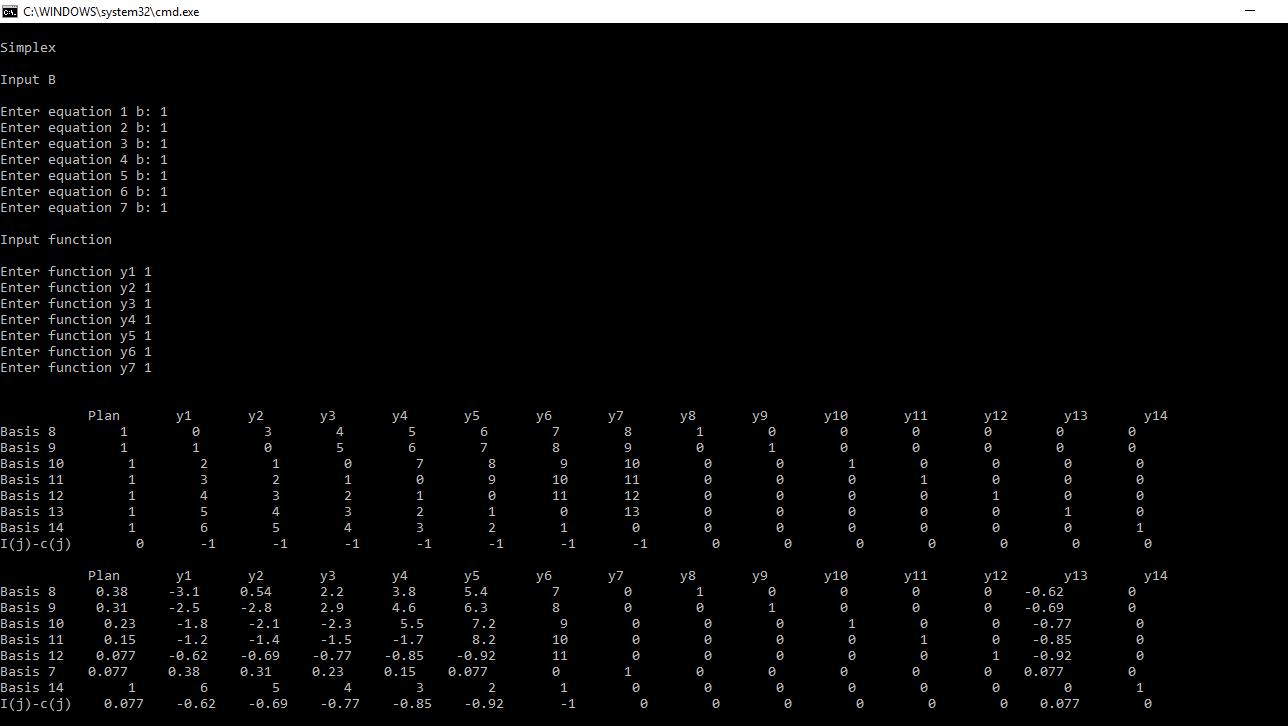
# **Խնդրի լուծման ալգորիթմը և դրա իրականացումը (ծրագրային մաս)**

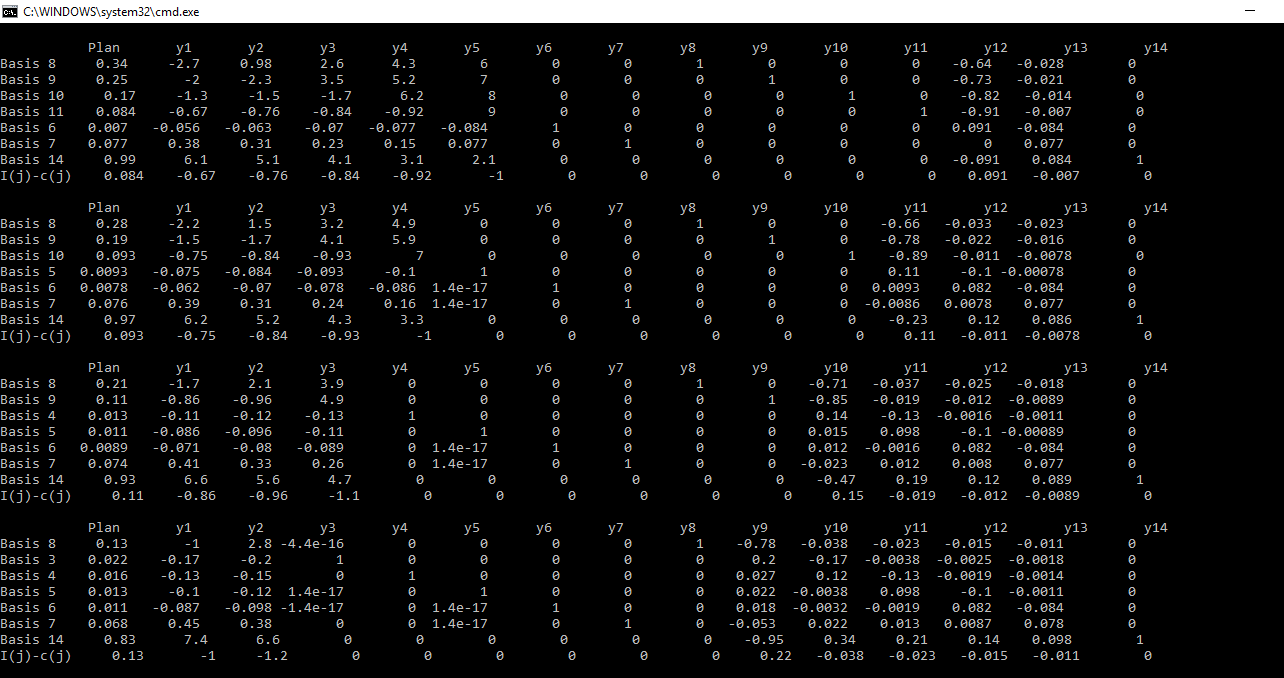
Ծրագրային մասի կոդը դրված է հավելվածում:

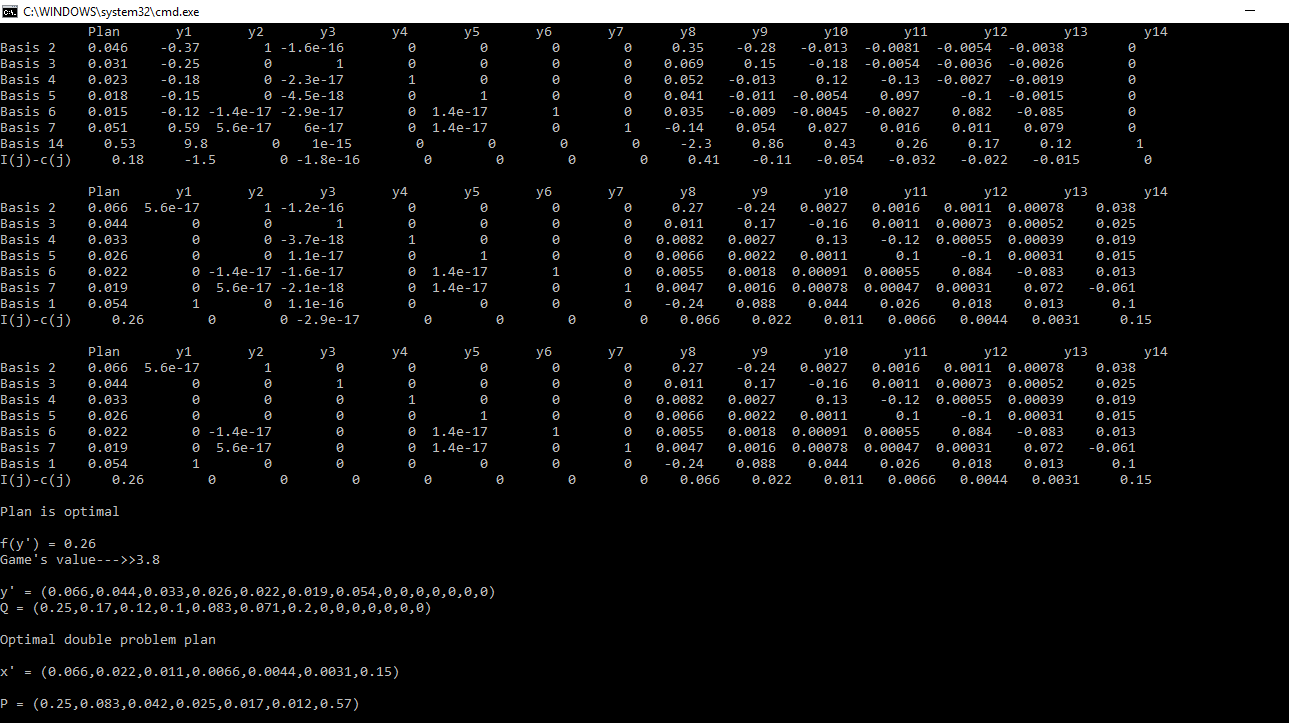












# **Լուծման թվային արժեքները**

### **Առաջին խաղացողի ստրատեգիաների հավանականությունները**

1 = 70/1063, 2 = 70/3189, 3 = 35/3189, 4 = 7/1063, 5 = 14/3189, 6 = 10/3189, 7 = 160/1063

Խաղի արժեքը ` =1063/280 և

1063/280\*70/1063 = 1/4

= 1063/280\*70/3189 = 1/12

= 1063/280\*35/3189 = 1/24

1063/280\*7/1063 = 1/40

= 1063/280\*14/3189 = 1/60

= 1063/280\*10/3189 = 1/84

= 1063/280\*160/1063 = 4/7

Առաջին խաղացողի օպտիմալ խառը ստրատեգիաները կլինեն՝

P = (1/4; 1/12; 1/24; 1/40; 1/60; 1/84; 4/7) :

### **Երկրորդ խաղացողի ստրատեգիաների հավանականությունները**

Խաղի արժեքը` =1063/280

=

1= 57/1063, 2 = 70/1063, 3 = 140/3189, 4 = 35/1063, 5 = 28/1063, 6 = 70/3189, 7 = 20/1063

= 1063/280\*57/1063 = 57/280

= 1063/280\*70/1063 = 1/4

= 1063/280\*140/3189 = 1/6

= 1063/280\*35/1063 = 1/8

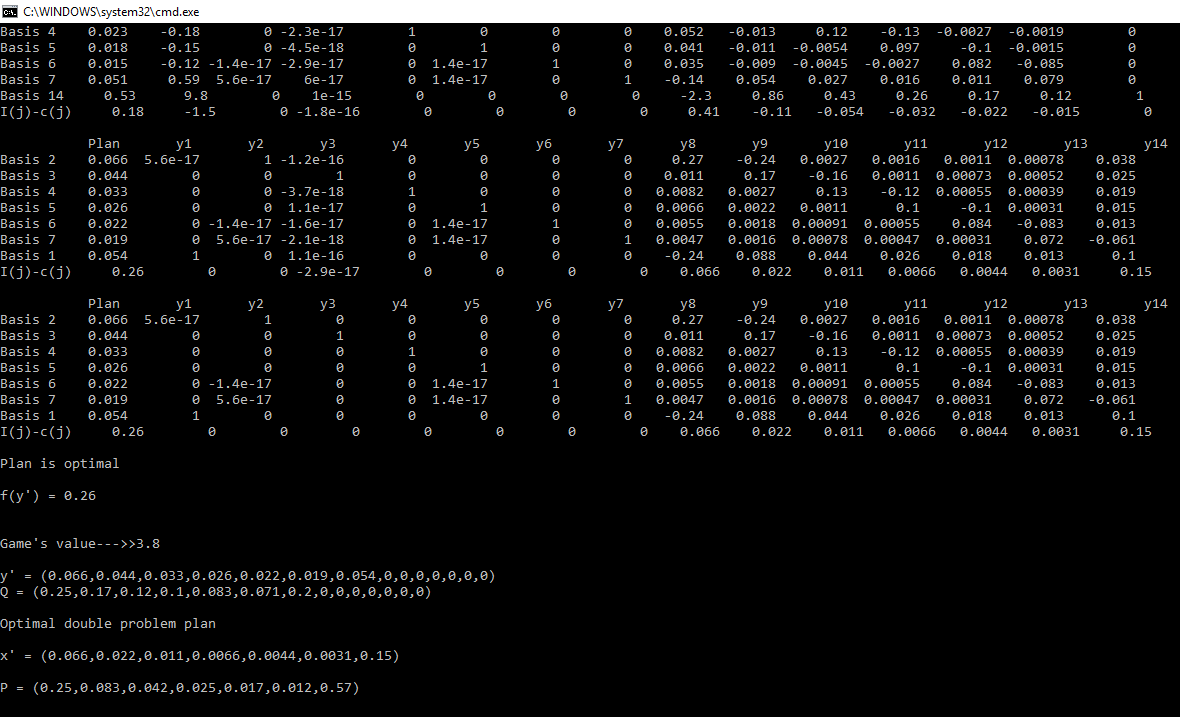
= 1063/280\*28/1063 = 1/10

= 1063/280\*70/3189 = 1/12

= 1063/280\*20/1063 = 1/14

Երկրորդ խաղացողի օպտիմալ խառը ստրատեգիաները կլինեն՝

Q = (57/280; 1/4; 1/6; 1/8; 1/10; 1/12; 1/14) :

Այսպիսով ունենալով խաղի արժեքը,որը այս դեպքում հավասար է՝ =3.796 կարող ենք որոշել,թե խաղացողները ինչ հավանականությամբ կարող են կիրառել իրենց ստրատեգիաները:

# **Արդյունքների մեկնաբանությունը**

Խնդիրը սիմպլեքս մեթոդով լուծելուց հետո օգտագործելով խնդրի վերջին իտերացիան գտել եմ երկակի խնդրի օպտիմալ պլանը՝

1 = 70/1063, 2 = 70/3189, 3 = 35/3189, 4 = 7/1063, 5 = 14/3189, 6 = 10/3189, 7 = 160/1063

= 1\*70/1063+1\*70/3189+1\*35/3189+1\*7/1063+1\*14/3189+1\*10/3189+1\*160/1063 = 280/1063

Խաղի արժեքը կլինի :

Ստանալով խաղի արժեքը` =3.796 , գտել եմ առաջին և երկրորդ խաղացողի ստրատեգիաների հավանականությունները, որոնք առաջին և երկրորդ խաղացողների օպտիմալ խառը ստրատեգիաներն են :

Առաջին խաղացողի օպտիմալ խառը ստրատեգիաներն են՝ P = (1/4; 1/12; 1/24; 1/40; 1/60; 1/84; 4/7):

Երկրորդ խաղացողի օպտիմալ խառը ստրատեգիաներն են ՝ Q = (57/280; 1/4; 1/6; 1/8; 1/10; 1/12; 1/14):

Խաղի արժեքը իրենից ներկայացնում է այն միավորը ,որ մի խաղացողը պետք է վճարի մյուսին, իր պարտության դեպքում:

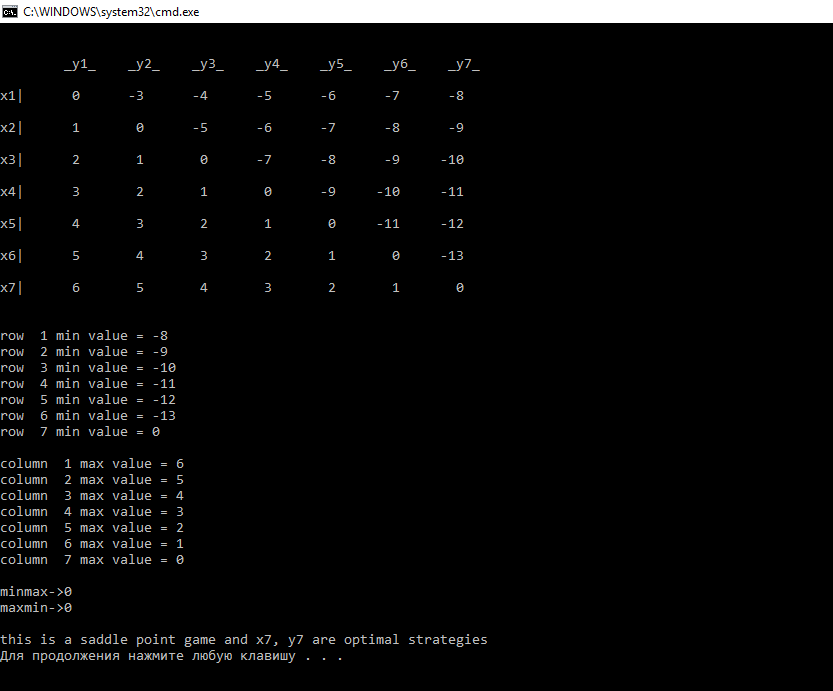
Խաղի մատրիցը սա է՝

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Խաղացողներ | B1 | B2 | B3 | B4 | B5 | B6 | B7 |
| A1 | 0 | 3 | 4 | 5 | 6 | 7 | 8 |
| A2 | 1 | 0 | 5 | 6 | 7 | 8 | 9 |
| A3 | 2 | 1 | 0 | 7 | 8 | 9 | 10 |
| A4 | 3 | 2 | 1 | 0 | 9 | 10 | 11 |
| A5 | 4 | 3 | 2 | 1 | 0 | 11 | 12 |
| A6 | 5 | 4 | 3 | 2 | 1 | 0 | 13 |
| A7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Այսպիսով մենք կարող ենք իմանալ ,որ ստրատեգիան ինչ հավանականությամբ կիրառել:

Առաջին խաղացողը ձգտում է ստանալ առավելագույն շահում, ապա հաճախ պետք է ընտրի իր այն ստրատեգիան, որի հավանականությունը ամենամեծն է՝ = 4/7 , և քանի որ երկրորդ խաղացողը պետք է նվազագույնի հասցնի առաջինի շահումը, ապա նորից հաճախ պետք է ընտրի այն ստրատեգիան, որի հավանականությունը ամենամեծն է՝ = 57/280:

Խնդրի մատրիցում, եթե երկրորդ խաղացողի շահումները ներկայացնենք բացասական, որպես առաջին խաղացողի պարտություն: Այսինքն խաղը ներկայացնենք առաջին խաղացողի դիրքից , ապա կստանամ թամբի կետով խաղ, որտեղ օպտիմալ ստրատեգիաներն են՝ -ը և -ը : Հետևաբար 0-ն այն թիվն է , որը պետք է ընտրի առաջին խաղացողը, որ ստանա առվելագույն շահում և ընտրի երկրորդ խաղացողը, որ իր կորուստը լինի մինիմալ:



# **Օգտագործված գրականության ցանկ**

1. Ա. Ղ. Մարգարյան,Մ.Վ. Մարկոսյան,Ս.Ա. Մանուկյան Համակարգային վերլուծություն և գործողությունների հետազոտում առարկայի դասախոսությունների ձեռնարկ, Երևան, 2013:
2. Воробев Н.Н. Теория игр,М. Наука,1985

# **Հավելված**

#include<iostream>

#include<iomanip>

using namespace std;

#define MAX\_ROWS 11// amenashaty qani tox karox e linel

#define MAX\_COLS 33//amenashaty qani syun karox e linel

int rows; //anhavasarumneri qanaky

int cols; //popoxakanneri qanaky

int bas[MAX\_ROWS]; // // yuraqanchyur havasarman bazisayin popoxakannery

double mat[MAX\_ROWS][MAX\_COLS];/\* matric vory arajin sharqum parunakum e amboxj funkciayi gorcakicnery yev hajordox havasarumneri gorcakicnery \*/

bool InputSize(); /\* funkcia vory talis e havasarumneri gorcakicnery ev gtnum e bazisayin popoxakannery \*/

void InputMatrix();// havasarumneri e popoxakanneri nermucum

void InputFunction(); //// funkciayi gorcakicneri nermucum

void SetIndexString(); // // indexayin toxi arjeqi hashvark

void Show(); // axyusakneri karuucum

bool CheckIsComp(); // optimalutyan stugum

int FindMasterCol(); // uxekcox syan hamary

void ShowResult();//xaxi arjeq optimal lucum

int FindRow(int col); // uxekcox toxi hamari voroshum

void Gauss(int col, int row); // Gaussi metod

int N;//verjnakan matrici toxeri qanaky

int M;//verjnakan matrici sjuneri qanaky

double\*\* mmat;//verjnakan matric

double minmin = 0;//ete matricy bacasakan tarrer uni amenapoqr tarn e

double\*\* NewMatr;//derevs bacasakan tarrerov matricy

double minmax(double min[], int m);

double maxmin(double max[], int n);

double minmaxi(double min[], int m);

double maxmini(double max[], int n);

int main()

{

int wish;

cout << "If you want to input another matrix,write 0:\n";

cout << "If you want to see the matrix of my game, write 1:\n";

cout << "If you write 2 my game will turn into a saddle point game:\n";

cout << "Choose one of the three:";

cin >> wish;

int row = 7, column = 7;

double\*\* matrix = new double\*[row];

int\* x = new int[row];

int\* y = new int[column];

if (wish == 1)

{

int won\_lost = 0;

for (int i = 0; i < row; i++)

{

x[i] = i + 1;

for (int j = 0; j < column; j++)

{

y[j] = j + 1;

if (x[i] >= y[j])

{

won\_lost = x[i] - y[j];

cout << "x->" << x[i] << endl;

cout << "y->" << y[j] << endl;

cout << "first player won and second player lose:" << won\_lost << endl;

}

else if (x[i] < y[j])

{

won\_lost = (x[i] + y[j]);

cout << "x->" << x[i] << endl;

cout << "y->" << y[j] << endl;

cout << "second player won and first player lose :" << won\_lost << endl;

}

}

}

cout << "\n" << endl;

for (int i = 0; i < row; i++)

{

cout << "\t\_y" << i + 1 << "\_";

matrix[i] = new double[column];

}

cout << endl;

cout << endl;

for (int i = 0; i < row; i++)

{

cout << "x" << i + 1 << "|";

for (int j = 0; j < column; j++)

{

if (x[i] >= y[j])

{

matrix[i][j] = x[i] - y[j];

}

else if (x[i] < y[j])

{

matrix[i][j] = (x[i] + y[j]);

}

cout << setw(7) << matrix[i][j] << " ";

}

cout << endl;

cout << endl;

}

}

else if (wish == 0)

{

cout << "row=";

cin >> row;

cout << "column=";

cin >> column;

for (int i = 0; i < row; i++)

{

matrix[i] = new double[column];

}

cout << endl;

cout << endl;

for (int i = 0; i < row; i++)

{

for (int j = 0; j < column; j++)

{

cout << "matrix[" << i + 1 << "]" << "[" << j + 1 << "]=";

cin >> matrix[i][j];

}

}

cout << endl;

for (int i = 0; i < row; i++)

{

for (int j = 0; j < column; j++)

{

cout << matrix[i][j] << " ";

}cout << endl;

}

}

else if (wish == 2)

{

int won\_lost = 0, wish;

for (int i = 0; i < row; i++)

{

x[i] = i + 1;

for (int j = 0; j < column; j++)

{

y[j] = j + 1;

if (x[i] >= y[j])

{

won\_lost = x[i] - y[j];

cout << "x->" << x[i] << endl;

cout << "y->" << y[j] << endl;

cout << "first player won :" << won\_lost << endl;

}

else if (x[i] < y[j])

{

won\_lost = -(x[i] + y[j]);

cout << "x->" << x[i] << endl;

cout << "y->" << y[j] << endl;

cout << "first player lose :" << won\_lost << endl;

}

}

}

cout << "\n" << endl;

for (int i = 0; i < row; i++)

{

cout << "\t\_y" << i + 1 << "\_";

matrix[i] = new double[column];

}

cout << endl;

cout << endl;

for (int i = 0; i < row; i++)

{

cout << "x" << i + 1 << "|";

for (int j = 0; j < column; j++)

{

if (x[i] >= y[j])

{

matrix[i][j] = x[i] - y[j];

}

else if (x[i] < y[j])

{

matrix[i][j] = -(x[i] + y[j]);

}

cout << setw(7) << matrix[i][j] << " ";

}

cout << endl;

cout << endl;

}

}

else

{

cout << "<<<< There is no such option! >>>>";

}

cout << endl;

//////gerishxox toxeri ev syuneri voroshum

double\* domrow = new double[row];

double\* domcol = new double[column];

for (int i = 0; i < row; i++)

{

for (int k = 0; k < row; k++) {

int count = 0;

for (int j = 0; j < column; j++)

{

if (i != k && matrix[i][j] >= matrix[k][j]) {

count++;

if (count == column)

{

domrow[i] = k;

}

}

}

}

}

cout << endl;

int countt = 0;

for (int i = 0; i < row; i++) {

for (int j = i + 1; j < row - 1; j++)

{

if (domrow[i] == domrow[j] && domrow[i] >= 0)

{

countt++;

}

}

}

//gerishxox syuni voroshum

for (int j = 0; j < column; j++)

{

for (int l = 0; l < column; l++) {

int count = 0;

for (int i = 0; i < row; i++)

{

if (j != l && matrix[i][j] <= matrix[i][l]) {

count++;

if (count == row)

{

domcol[l] = l;

}

}

}

}

}

cout << endl;

int counts = 0;

for (int i = 0; i < column; i++) {

for (int j = i + 1; j < column - 1; j++)

{

if (domcol[i] == domcol[j] && domcol[i] >= 0)

{

counts++;

}

}

}

int n1 = row - countt;

int m1 = column - counts;

double\* RepDomRow = new double[n1];

int hashvich = 0;

RepDomRow[0] = domrow[0];

for (int i = 1, l = 1; i < row, l < n1; i++) {

if (domrow[i] < 0) {

RepDomRow[l] = domrow[i];

l++;

}

hashvich = 0;

for (int j = 0; j < l; j++)

{

if (domrow[i] >= 0 && domrow[i] != RepDomRow[j])

{

hashvich++;

if (hashvich == l) {

RepDomRow[l] = domrow[i];

l++;

}

}

}

}

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << endl;

int countrw = 0;

cout << "Remowed rows indexes" << endl;

for (int l = 0; l < n1; l++) {

if (RepDomRow[l] >= 0) {

countrw++;

cout << RepDomRow[l] << ",";

}

}

cout << endl;

cout << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << endl;

double\* RepDomCol = new double[m1];

RepDomCol[0] = domcol[0];

hashvich = 0;

for (int i = 1, k = 1; i < column, k < m1; i++) {

if (domcol[i] < 0) {

RepDomCol[k] = domcol[i];

k++;

}

hashvich = 0;

for (int j = 0; j < k; j++)

{

if (domcol[i] >= 0 && domcol[i] != RepDomCol[j])

{

hashvich++;

if ((hashvich = k) && (j == k - 1)) {

RepDomCol[k] = domcol[i];

k++;

}

}

}

}

int countcl = 0;

cout << "Remowed columns indexes" << endl;

for (int k = 0; k < m1; k++) {

if (RepDomCol[k] >= 0) {

countcl++;

cout << RepDomCol[k] << ",";

}

}

cout << endl;

cout << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << endl;

N = row - countrw;

M = column - countcl;

if (N == row && M == column)

{

cout << " there aren't dominante rows and columns" << endl;

}

else

{

double\* dr = new double[row];

for (int j = 0; j < row; j++)

{

dr[j] = j;

}

cout << "Dominant Rows\n";

cout << endl;

for (int i = 0; i < row; i++)

{

bool buf = false;

for (int j = 0; j < row; j++)

{

if (dr[i] == RepDomRow[j]) {

buf = true; break;

}

}

if (!buf) {

cout << dr[i] << " ";

}

}

cout << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << endl;

double\* dc = new double[column];

for (int i = 0; i < column; i++)

{

dc[i] = i;

}

cout << "Dominant Columns\n";

for (int i = 0; i < column; i++)

{

bool buf = false;

for (int j = 0; j < column; j++)

{

if (dc[i] == RepDomCol[j]) {

buf = true; break;

}

}

if (!buf) {

cout << dc[i] << " ";

}

}

cout << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

double\*\* NewMatrix = new double\*[N];

for (int i = 0; i < N; ++i) {

NewMatrix[i] = new double[column];

}

int hashvich1 = 0;

int hashvich2 = 0;

for (int j = 0, J = 0; j < column, J < column; j++, J++)

{

for (int I = 0, i = 0; I < N, i < row; i++)

{

for (int k = 0; k < n1; k++) {

if (i != RepDomRow[k]) {

hashvich1++;

if (k == (n1 - 1))

{

if (hashvich1 == n1)

{

NewMatrix[I][J] = matrix[i][j];

I++;

k = 0;

hashvich1 = 0;

}

else {

hashvich1 = 0;

}

}

}

}

}

}

cout << endl;

NewMatr = new double\*[N];

for (int i = 0; i < N; ++i) {

NewMatr[i] = new double[M];

}

for (int I = 0, i = 0; I < N, i < N; I++, i++)

{

for (int j = 0, J = 0; j < column, J < M; j++)

{

for (int k = 0; k < m1; k++) {

if (j != RepDomCol[k]) {

hashvich2++;

if (hashvich2 == m1)

{

NewMatr[I][J] = NewMatrix[i][j];

J++;

k = 0;

hashvich2 = 0;

}

if (k == m1 - 1 && hashvich2 != m1) {

hashvich2 = 0;

}

}

}

}

}

cout << endl;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

cout << setw(N) << NewMatr[i][j] << "\t";

}

cout << endl;

}

cout << endl;

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

if (NewMatr[i][j] < minmin)

{

minmin = NewMatr[i][j];

}

}

}

cout << "\n" << endl;

mmat = new double\*[N];

for (int i = 0; i < N; i++)

{

mmat[i] = new double[M];

}

/////bacasakan tarreri heracum

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

if (NewMatr[i][j] < 0 || NewMatr[i][j] >= 0)

{

mmat[i][j] = NewMatr[i][j] - minmin;

}

else

{

mmat[i][j] = NewMatr[i][j];

}

}

}

cout << "Matrix without negative elements:\n\n";

////matrici artacum

for (int i = 0; i < N; i++)

{

for (int j = 0; j < M; j++)

{

cout << setw(N) << mmat[i][j] << " ";

}

cout << endl;

}

cout << endl;

double \*max = new double[M];

double \*min = new double[N];

for (int i = 0; i < N; i++)

{

min[i] = mmat[i][0];

for (int j = 0; j < M; j++)

if (mmat[i][j] < min[i])

min[i] = mmat[i][j];

cout << "row " << i + 1 << " min value = " << min[i] << endl;

}

cout << endl;

for (int j = 0; j < M; j++)

{

max[j] = mmat[0][j];

for (int i = 0; i < N; i++)

if (mmat[i][j] > max[j])

max[j] = mmat[i][j];

cout << "column " << j + 1 << " max value = " << max[j] << endl;

}

cout << endl;

double r1, r2;

int ind1, ind2;

r1 = minmax(min, N);

r2 = maxmin(max, M);

ind1 = minmaxi(min, N);

ind2 = maxmini(max, M);

cout << "minmax->" << r1 << endl;

cout << "maxmin->" << r2 << "\n" << endl;

if (r1 == r2)

{

cout << "this is a saddle point game and x" << ind1 + 1 << ", y" << ind2 + 1 << " are optimal strategies" <<endl;

}

else

{

cout << "this isn't a saddle point game" << "\n" << endl;

cout << "!!Build simplex table and find game's value" << endl;

}

cout << endl;

if (wish == 0 || wish == 1)

{

cout << "Simplex\n\n";

cout << "Input B\n\n";

if (!InputSize())

{

cout << "Invalid sizes!";

cin.get();

return 0;

}

InputMatrix();

InputFunction();

SetIndexString();

Show();

while (true)

{

if (!CheckIsComp())

{

cout << endl << "No optimal plan!";

break;

}

int col = FindMasterCol();

if (!col)

{

cout << endl << endl << "Plan is optimal";

ShowResult();

break;

}

int row = FindRow(col);

Gauss(col, row);

Show();

};

cin.get();

cin.get();

}

delete[] x;delete[] y;

delete[] max;delete[] min;

delete[] domcol;delete[] domrow;

delete[] RepDomCol;delete[] RepDomRow;

for (int i = 0; i < row; i++)

delete[] matrix[i];

delete[] matrix;

for (int I = 0; I < N; I++)

delete[] matrix[I];

delete[] matrix;

for (int i = 0; i < N; i++)

delete[] NewMatr[i];

delete[] NewMatr;

for (int i = 0; i < N; i++)

delete[] mat[i];

delete[] mat;

return 0;

}

double minmax(double min[], int m)

{

double minmax = min[0];

int i;

for (i = 0; i < m; i++)

if (min[i] > minmax)

minmax = min[i];

return minmax;

}

double maxmin(double max[], int n)

{

double maxmin = max[0];

int j;

for (j = 0; j < n; j++)

if (max[j] < maxmin)

maxmin = max[j];

return maxmin;

}

double minmaxi(double min[], int m)

{

double minmax = min[0];

int l = 0;

for (int i = 0; i < m; i++)

{

if (min[i] > minmax)

{

minmax = min[i];

l = i;

}

}

return l;

}

double maxmini(double max[], int n)

{

double maxmin = max[0];

int k = 0;

for (int j = 0; j < n; j++)

{

if (max[j] < maxmin)

{

maxmin = max[j];

k = j;

}

}

return k;

}

bool InputSize()

{

int c = MAX\_ROWS - 2;

rows=N;

if (rows > c)

return false;

c = MAX\_COLS - 2 - c;

cols=M;

return cols <= c;

}

void InputMatrix()

{

for (int s=0,i = 1; i <= rows; i++,s++)

{

for (int p=0,j = 2; j <= cols + 2; j++,p++)

{

if (j == cols + 2)

{

cout << "Enter equation " << i << " b: ";

cin >> mat[i][cols + 2 + rows];

mat[i][cols + i + 1] = 1;

bas[i] = cols + i;

}

else

{

mat[i][j] = mmat[s][p];

//cout << "Enter equation " << i << " y" << j - 1 << ": " << mat[i][j] << endl;;

}

}

}

cols += rows;

for (int i = 1; i <= rows; i++)

mat[i][1] = mat[i][cols + 2];

}

void InputFunction()

{

cout << endl;

cout << "Input function\n\n";

for (int j = 2; j <= cols + 1 - rows; j++)

{

cout << "Enter function y" << j - 1 << " ";

cin >> mat[0][j];

}

}

void SetBasis()

{

for (int i = 1; i <= rows; i++)

{

bas[i] = ++cols;

mat[i][bas[i] + 1] = 1;

}

}

void SetIndexString()

{

mat[rows + 1][1] = 0;

for (int i = 1; i <= rows; i++)

mat[rows + 1][1] = mat[rows + 1][1] + mat[i][1] \* mat[0][bas[i] + 1];

for (int j = 2; j <= cols + 1; j++)

mat[rows + 1][j] = 0;

for (int j = 2; j <= cols + 1; j++)

{

for (int i = 1; i <= rows; i++)

mat[rows + 1][j] = mat[rows + 1][j] + mat[i][j] \* mat[0][bas[i] + 1];

mat[rows + 1][j] = mat[rows + 1][j] - mat[0][j];

}

}

void Show()

{

cout << endl;

for (int i = 0; i <= rows + 1; i++)

{

if (!i)

cout << endl << " ";

else if (i == rows + 1)

cout << endl << "I(j)-c(j) ";

else

cout << endl << "Basis " << bas[i] << " ";

for (int j = 1; j <= cols + 1; j++)

{

if (i == 0)

{

if (j == 1)

{

cout << setw(14) << "Plan";

}

else

{

cout << setw(8) << "y" << j - 1;

}

}

else

{

cout << setw(8) << setprecision(2) << mat[i][j] << " ";

}

}

}

}

bool CheckIsComp()

{

for (int j = 2; j <= cols + 1; j++)

{

if (mat[rows + 1][j] < 0)

{

bool is = false;

for (int i = 1; i <= rows; i++)

is |= mat[i][j] >= 0;

if (!is)

return false;

}

}

return true;

}

int FindMasterCol()

{

int col;

int count = 0;

for (int i = 1; i <= cols; i++)

if (mat[rows + 1][i + 1] >= 0)

count++;

if (count == cols)

return 0;

double min = 0;

for (int i = 1; i <= cols; i++)

if (mat[rows + 1][i + 1] <= min)

{

min = mat[rows + 1][i + 1];

col = i;

}

return col;

}

void ShowResult()

{

cout <<"\n"<< endl;

cout << "f(y') = " << mat[rows + 1][1];

cout << endl;

double v = 1 / mat[rows + 1][1];

double V=v+minmin;

if (!minmin)

{

cout << "Game's value--->>" << v;

cout << endl;

}

else

{

cout << "Game's value without M--->>" << V;

cout<<endl;

}

cout << endl << "y' = (";

for (int i = 1; i <= cols; i++)

{

if (mat[i][1] == mat[rows + 1][1])

mat[i][1] = 0;

cout << mat[i][1];

if (i != cols)

cout << ',';

}cout << ")";

if (!minmin)

{

cout << endl << "Q = (";

for (int i = 1; i <= cols; i++)

{

if (mat[i][1] == mat[rows + 1][1])

mat[i][1] = 0;

cout << mat[i][1] \* v;

if (i != cols)

cout << ',';

}cout << ")";

}

else

{

cout << endl << "Q = (";

for (int i = 1; i <= cols; i++)

{

if (mat[i][1] == mat[rows + 1][1])

mat[i][1] = 0;

cout << mat[i][1] \* V;

if (i != cols)

cout << ',';

}cout << ")";

cout << endl;

}

cout <<"\n"<< endl;

cout << "Optimal double problem plan\n";

int colm = cols - (cols / 2) + 1;

cout << endl << "x' = (";

for (int j = 1; j <= rows; j++)

{

cout << mat[colm][colm + j];

if (j != rows)

cout << ',';

}cout << ")";

cout << endl;

if (!minmin)

{

cout << endl << "P = (";

for (int j = 1; j <= rows; j++)

{

cout << mat[colm][colm + j] \* v;

if (j != rows)

cout << ',';

}cout << ")";

cout << endl;

}

else

{

cout << endl << "P = (";

for (int j = 1; j <= rows; j++)

{

cout << mat[colm][colm + j] \* V;

if (j != rows)

cout << ',';

}cout << ")";

cout << endl;

}

}

int FindRow(int col)

{

bool ok = false;

int row;

double min\_row;

for (int i = 1; i <= rows; i++)

if (mat[i][col + 1] > 0)

{

ok = true;

min\_row = mat[i][1] / mat[i][col + 1];

row = i;

break;

}

if (!ok)

return 0;

for (int i = 1; i <= rows; i++)

{

if (mat[i][col + 1] > 0)

{

double a = mat[i][1] / mat[i][col + 1];

if (a < min\_row)

{

min\_row = a; row = i;

}

}

}

return row;

}

void Gauss(int col, int row)

{

for (int i = row + 1; i <= rows + 1; i++)

{

double min\_one = -1;

double mnog = min\_one \* mat[i][col + 1] / mat[row][col + 1];

for (int j = 1; j <= cols + 1; j++)

mat[i][j] = mat[i][j] + mat[row][j] \* mnog;

}

for (int i = row - 1; i >= 1; i--)

{

double min\_one(-1);

double mnog = min\_one \* mat[i][col + 1] / mat[row][col + 1];

for (int j = 1; j <= cols + 1; j++)

mat[i][j] = mat[i][j] + mat[row][j] \* mnog;

}

double del = mat[row][col + 1];

for (int j = 1; j <= cols + 1; j++)

mat[row][j] = mat[row][j] / del;

bas[row] = col;

}

1. Воробев Н.Н. Теория игр,М. Наука,1985 [↑](#footnote-ref-1)
2. Воробев Н.Н. Теория игр,М. Наука,1985 [↑](#footnote-ref-2)
3. Воробев Н.Н. Теория игр,М. Наука,1985 [↑](#footnote-ref-3)
4. Воробев Н.Н. Теория игр,М. Наука,1985 [↑](#footnote-ref-4)
5. Ա. Ղ. Մարգարյան,Մ.Վ. Մարկոսյան,Ս.Ա. Մանուկյան Համակարգային վերլուծություն և գործողությունների հետազոտում առարկայի դասախոսությունների ձեռնարկ, Երևան, 2013: [↑](#footnote-ref-5)
6. Ա. Ղ. Մարգարյան,Մ.Վ. Մարկոսյան,Ս.Ա. Մանուկյան Համակարգային վերլուծություն և գործողությունների հետազոտում առարկայի դասախոսությունների ձեռնարկ, Երևան, 2013: [↑](#footnote-ref-6)
7. Ա. Ղ. Մարգարյան,Մ.Վ. Մարկոսյան,Ս.Ա. Մանուկյան Համակարգային վերլուծություն և գործողությունների հետազոտում առարկայի դասախոսությունների ձեռնարկ, Երևան, 2013: [↑](#footnote-ref-7)