# გიორგი მარჯანიძე

სამაგისტრო პროგრამა: ინფორმაციული სისტემები


ინტელექტუალური მხარდამჭერი სისტემა:
სატრანსპორტო მარშრუტიზაციის ამოცანის
ევრისტიკული ალგორითმები

# შესავალი

**სატრანსპორტო მარშრუტიზაციის პრობლემები ექსტრემალურ და განუზღვრელ გარემოში**

- თანამედროვე მსოფლიოში უფრო და უფრო პრობლემური ხდება სატრანსპორტო საშუალებების (სს) მარშრუტებზე ოპტიმალური გადაადგილების მენეჯმენტი ექსტრემალური და გართულებული პროცესების მიმდინარეობის პირობებში. ესენია: 1. კატასტროფების, მიწისძვრების, მასობრივი განადგურების იარაღის გამოყენების შედეგად და სხვ. დაზიანებულ გეოგრაფიულ ზონებში არსებული სამხედრო, და სხვა ტიპის ობიექტების ოპტიმალური და უსაფრთხო მომარაგების მენეჯმენტი; 2. ექსტრემალურ და რთულ სიტუაციებში სწრაფი რეაგირებისა და მოსახლეობისათვის უსაფრთხო დახმარების დაგეგმვა;3. ექსტრემალურ სიტუაციაში სამხედრო მოქმედებისას სატრანსპორტო საშუალებებით ტვირთების გადაზიდვის მარშრუტების სტრატეგიული მენეჯმენტი; 4. მჭიდროდ დასახლებულ გეოგრაფიულ ზონებში (ქალაქები და სხვ.) გზებზე გართულებულ სიტუაციებში ტრანსპორტით გადატვირთული გზები, სამოქალაქო მიტინგები და გაფიცვები, გზებზე მეტეოროლოგიური და სხვა მიზეზებით გამოწვეული ცუდი ხილვადობა, მოყინული გზები და სხვ.) სატრანსპორტო საშუალებების მარშრუტებზე ოპტიმალური გადაადგილების მენეჯმენტი და სხვა.

  ამ პრობლემატიკით დაინტერესებული სახელმწიფო თუ კერძო ორგანიზაციები ცდილობენ შექმნან მაღალი სანდოობის ინტელექტუალური ინფორმაციული ტექნოლოგიები, რომლებიც გაითვალისწინებენ ექსტრემალური სიტუაციებში წარმოქმნილ განუზღვრელობებს და მხარდაჭერას გაუკეთებენ სს–ების გადაადგილების ოპტიმალური მარშრუტების დაგეგმვას.

  ასეთი ტიპის პრობლემატიკაზე მუშაობისას დეტერმინისტული თუ სტოქასტური მოდელების ბაზაზე აგებული სიმულაციური მხარდამჭერი ტექნოლოგიები ხშირად ვერ გვაძლევენ სანდო და დამაკმაყოფილებელ შედეგებს საკვლევი ობიექტის სირთულის, წინააღმდეგობრივი, ბუნდოვანი და არასაკმარისი ინფორმაციის ან ობიექტური ინფორმაციის სიმცირის გამო, რაც პირველ რიგში გამოწვეულია მიმდინარე რთული სიტუაციებით. პრობლემატიკის სირთულის ზრდასთან ერთად ჩვენი შესაძლებლობა გავაკეთოთ სანდო დასკვნები საკვლევი ობიექტების მომავალ ქცევაზე, გარკვეულ ზღვრამდე ეშვება, რომლის მიღმაც ინფორმაციის ისეთი მახასიათებლები, როგორიცაა სიზუსტე და განსაზღვრელობა, ურთიერთგამომრიცხავი ხდება. მნიშვნელოვან როლს იძენს პრობემატიკის გადაწყვეტის სისტემური კვლევა და ანალიზი. აუცილებელი ხდება შეფასებებში და ანალიზში ჩავრთოთ ექსპერტთა ჯგუფები და მათი ცოდნა, რომელთა სუბიექტური მონაცემები მოდელის კონსტრუქციებში წარმოშობს ახალ, სუბიექტურ განუზღვრელობას. მოდელირების კლასიკურ მიმართულებათა პარალელურად მნიშვნელოვანი ხდება სუბიექტური, ფაზი–განუზღვრელობის დაშვება. ასეთ შემთხვევაში აუცილებელია ექსპერტული ცოდნის ინჟინერიის ფაზი-მეთოდებისა და ფაზი-ლოგიკის გამოყენება, რაც შესაბამისი მაღალი ღირებულების ავტომატიზირებული სისტემებისა და ინტელექტუალური ხელშემწყობი ტექნოლოგიების კონსტრუირებას უზრუნველყოფს.

## ფაზი-სატრანსპორტო მარშრუტიზაციის პრობლემები

- საქმე ეხება სატრანსპორტო მარშრუტიზაციის პრობლემატიკას (Vehicle Routing Problem (VRP)). მარშრუტიზაციის ამოცანები სკალარული მიზნის ფუნქციის შემთხვევაშიც ე.წ. NP- რთულ ამოცანებს განეკუთვნებიან და მათი ამოხსნის ზუსტი ალგორითმები რეალური განზომილებების შემთხვევაში არ არსებობს.VRP-ის ერთ-ერთი ყველაზე გავრცელებული მიმართულებაა ფაზი–სატრანსპორტო მარშრუტიზაციის პრობლემატიკა (Fuzzy Vehicle Routing Problem (FVRP)). FVRP -ამოცანებმა რთულ და განუზღვრელ სიტუაციებში უნდა უზრუნველყოს ოპტიმალური მარშრუტების გენერაცია.

  მკვლევართა ჯგუფმა, რომელშიც შესრულდა სამაგისტრო ნაშრომი, ერთ-ერთი პროექტის ფარგლებში შეიმუშავა ახალი FVRP-მიდგომა. ცოტა რამ ამ პროექტის ამოცანებზე და მიზნებზე: ეს მიდგომა გაითვალისწინებს ზემოთ წარმოდგენილი პრობლემის გადაწყვეტას. აიგება აგრეგირების ახალი ინსტრუმენტი. ეს ინსტრუმენტი უზრუნველყოფს გზებზე გდაადგილების მარშრუტების სანდოობის კრიტერიუმების აგებას. რაც FVRP-მიდგომებსა და კვლევებში საერთოდ ახალ მიმართულებას მოგვცემს.ამ მიდგომის საფუძველზე შეიქმნება პროგრამული პროდუქტი, რომელიც უზრუნველყოფს ექსტრემალური და რთული მოვლენების შედეგად გზებზე გართულებული გადაადგილების გამო სს–თვის ოპტიმალური და სანდო მარშრუტების დაგეგმვას. პროგრამული უზრუნველყოფის ფუნქცია იქნება სატრანსპორტო საშუალებათა მართვის სახელმწიფო სამსახურებსა, ტვირთების გადაზიდვების კომპანიებსა, სადისტრიბუციო ქსელებსა თუ სხვა კონპანიებს შეუქმნას მხარდაჭერა საჭიროების შემთხვევაში სწრაფი რეაგირებისა და მნიშვნელოვან გეორაფიულ პუნქტებში ტვირთების გადაზიდვის ოპტიმალური მარშრუტების დაგეგმვაში. სისტემაზე მუშაობის პროცესში სისტემის მომხმარებლებს შესაძლებლობა ექნებათ ინფორმაციის მიღების მიზნით ჩართონ დარგის ცნობილი ექსპერტები, რათა მათი ცოდნა გამოყენებული იყოს კონკრეტულ სიტუაციებში პუნქტებს შორის გადაადგილების შესაძლებლოს ხარისხების შეფასებისა და სს–ებისთვის სანდო მარშრუტების აგების მიზნით. პროექტში წარმოდგენილი პრობლემისთვის აიგება ახალი ტიპის შესაძლებლობითი კრიტერიუმი - მარშრუტებზე გადაადგილების სანდოობის მაქსიმიზაცია. მარშრუტებზე გადაადგილების სიგრძის მინიმიზაციის კრიტერიუმთან ერთად შეიქმნება ორკრიტერიალური ამოცანის რეალიზების ორ ფაზიანი სქემა. ეს მიდგომა წარმოშობს ახალ მიმარულებას და პერსპექტივებს FVPP-პრობლემატიკაში. ყოველ მიდგომაში, რომელიც სწავლობს FVRP-ამოცანებს, შეიძლება ჩაიდოს ჩვენი ახალი მეთოდოლოგია და იქ აიგოს ახალი კრიტერიუმები და შეზღუდვები. რაც მათ შემატებს მეტ სანდოობას ექსტრემალურ და განუზღვრელ გარემოში.

## კომივოიაჟერის ამოცანა

- კომივოიაჟერის ამოცანა დისკრეტული ოპტიმიზაციის ერთ-ერთ ყველაზე საინტერესო ამოცანას წარმოადგენს, როგორც თეორიული კვლევის ასევე პრაქტიკული გამოყენების თვალსაზრისით.

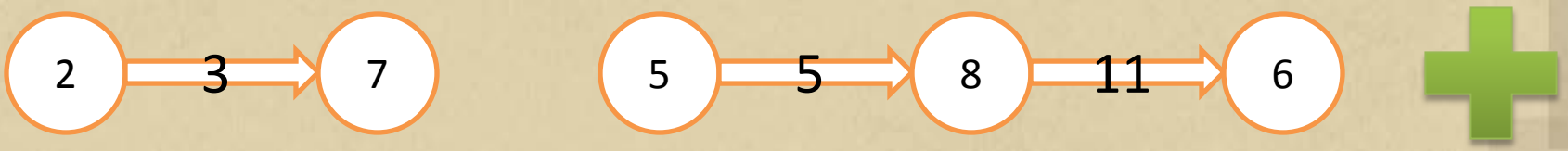
  კომივოიაჟერმა (მოხეტიალე ვაჭარმა) უნდა შემოიაროს ქალაქების ფიქსირებული ერთობლიობა. მოძრაობა უნდა დაიწყოს იმ ქალაქიდან სადაც იმყოფება და უნდა დაამთავროს იმავე ქალაქში. ამასთან ყველა ქალაქში უნდა შევიდეს ერთხელ. ეს ისე უნდა გააკეთოს, რომ მთელი მოგზაურობის ჯამური დანახარჯები იყოს მინიმალური. დანახარჯებში იგულისხმება ან თანხა, ან დრო, ან ჯამური გავლილი მანძილი.

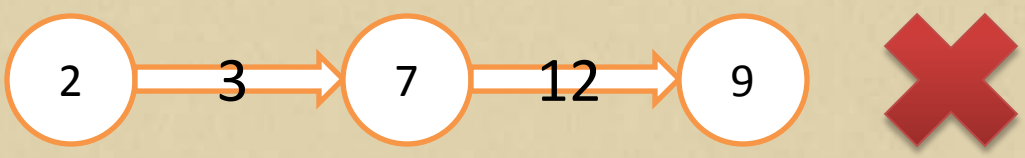
  კომივოიაჟერის ამოცანა (TSP - Travelling Salesman Problem)  კომბინატორული ოპტიმიზაციის ტიპიური ამოცანაა, რომლის ჩამოყალიბება (სიტყვიერი ფორმულირება) მარტივი ამოცანის შთაბეჭდილებას ტოვებს, თუმცა ზუსტი ამონახსნის პოვნა სერიოზულ ძალისხმევას ითხოვს და დიდი განზომილებისთვის გადაუჭრელ პრობლემად რჩება. ბიზნეს-გამოყენებებში გვხვდება  კომივოიაჟერის ამოცანის სხვა ვარიანტი - ხელსაწყოთა ოპტიმალური გადაწყობის ამოცანის სახით.

  ზოგად შემთხვევაში კომივოიაჟერის ამოცანა შეიძლება ჩამოყალიბდეს როგორც გრაფში უმოკლესი სიგრძის ჰამილტონის ციკლის მოძებნის ამოცანა.

## უმოკლესი რკალის არჩევის ალგორითმი

- ალგორითმი მუშაობას იწყებს მინიმალური რკალის არჩევით (თუ ასეთი რამდენიმეა, ვირჩევთ ნებისმიერს), შემდეგ ეტაპზე ირჩევს ასევე მინიმალურ რკალს, ოღონდ არა აუცილებლად ბმულობის პირობით. ანუ ალგორითმის მუშაობის პროცესში ხდება ე.წ. „ფრაგმენტების“ აგება, ყოველი ახალი რკალის დამატებისას ვაკონტროლებთ -ზე ნაკლები განზომილების ციკლების წარმოშობის საკითხს, თან ვახდენთ „ფრაგმენტების“ კორექტირებას. გარკვეული ბიჯების შემდეგ „ფრაგმენტები“ გადაბმას დაიწყებენ და შემდეგ ვღებულობთ ჰამილტონის ციკლს.



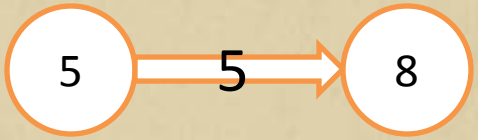



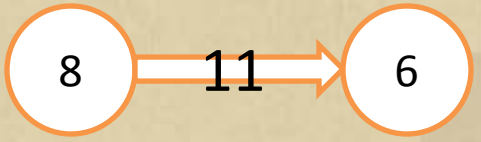
1)


2)

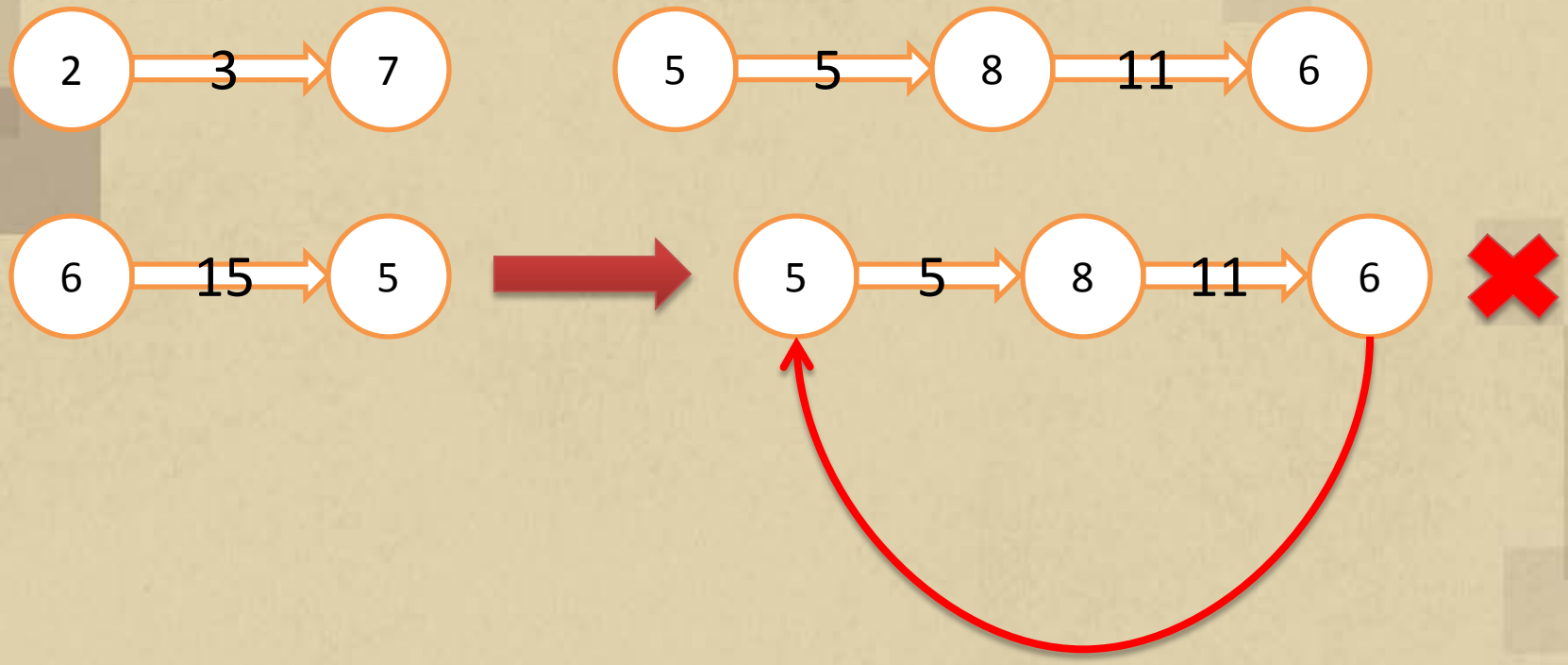


3)


4)



## 2-opt ალგორითმი

- ოპტიმიზაციაში, 2-opt არის მარტივი ლოკალური ძიების ალგორითმი პირველად შემოთავაზებული Croes-ის მიერ 1958 წელს კომივოიაჟერის პრობლემის გადასაწყვეტად. მთავარი იდეა მდგომარეობს იმაში რომ აიღოს მარშრუტი რომელიც კვეთს თავის თავს და შეცვალოს მიმდევრობა ისეთნაირად რომ ეს გადაკვეთა აღარ მოხდეს.


სრული 2-opt ლოკალური ძიება შეადარებს შეცვლის მექანიზმის ყველა შესაძლო დასაშვებ კომბინაციას. ეს მეთოდი შეიძლება გამოყენებულ იქნას კომივოიაჟერის ამოცანისთვის და ასევე ბევრი მსგავსი ამოცანისთვის. ამ ამოცანებში იგულისხმება ტრანსპორტის მარშრუტიზაციის პრობლემა (VRP), ასევე მოცულობით ტმპ (Capacitated VRP). ამისათვის ალგორითმის უმნიშვნელოდ ცვლილება იქნება საჭირო.

*ეს არის მექანიზმი რომლითაც 2-opt ჩანაცვლება (swap) ცვლის მოცემულ მარშრუტს*

2optSwap(route, i, k) {

1. წაიყვანე route[0] route[i-1]-თან და დაამატე ისინი new_route-ს
2. წაიყვანე route[i] route[k]-თან, დაალაგე ისინი საპირისპირო მიმართულებით და დაამატე new_route-ს
3. წაიყვანე route[k+1] ბოლოსთან და დაამატე ისინი new_route-ს

return new_route;

}

*მაგალითი*

მარშრუტი: A ==> B ==> C ==> D ==> E ==> F ==> G ==> H ==> A
მაგალითისთვის i = 3 და k = 6
new_route:

1. (A ==> B ==> C)
2. A ==> B ==> C ==> (G ==> F ==> E ==> D)
3. A ==> B ==> C ==> G ==> F ==> E ==> D (==> H ==> A)

*ეს არის სრული 2-opt ჩანაცვლება (swap) რომელიც იყენებს ზემოთ აღწერილ მექანიზმს*

```
გაიმეორე მანამ სანამ არ გაუმჯობესდება {
        start_again:
        best_distance = calculateTotalDistance(existing_route)
        for (i = 0; i < გადასანაცვლებელი კვანძების რაოდენობა - 1; i++) {
                for (k = i + 1; k < გადასანაცვლებელი კვანძების რაოდენობა; k++) {
                        new_route = 2optSwap(existing_route, i, k)
                        new_distance = calculateTotalDistance(new_route)
                        if (new_distance < best_distance) {
                                existing_route = new_route
                                goto start_again
                        }
                }
        }
}
```

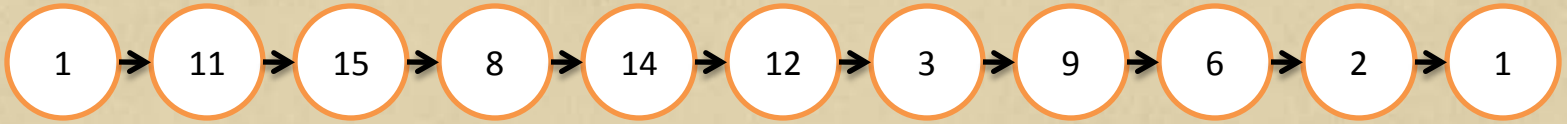
შენიშვნა: თუ მარშრუტი იწყება/სრულდება კონკრეტულ კვანძთან ან დეპოსთან, მაშინ ის კონკრეტული კვანძი ძიებიდან უნდა წაიშალოს როგორც გადასანაცვლებელი კვანძი, რადგან წყობის გადანაცვლების შედეგად მივიღებთ არასწორ გზას.

მაგალითისათვის, როდესაც A არის დეპო: A ==> B ==> C ==> D ==> A

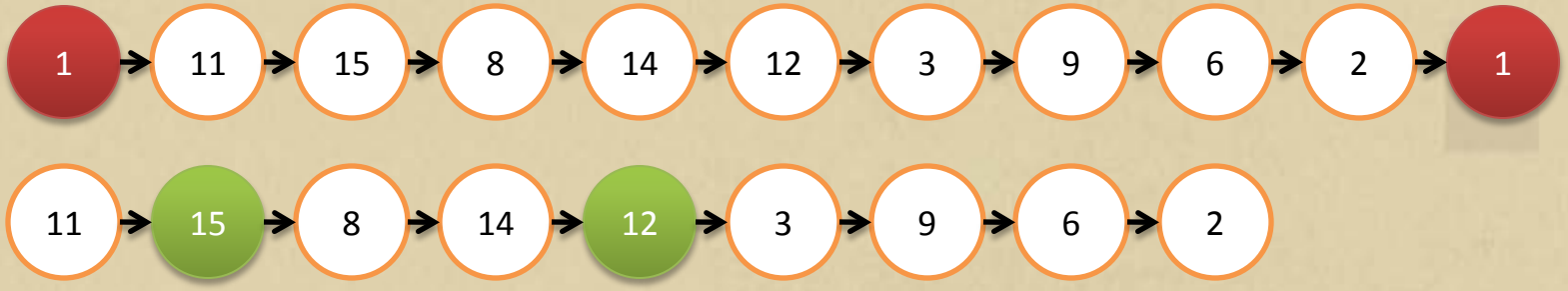
კვანძი[0]-სა და კვანძი[2]-ს გადანაცვლება მოგვცემს C ==> B ==> A ==> D ==> A რომელიც არასწორია (A-დან (დეპოდან) არ გადის).

---

საწყისი მარშრუტი:

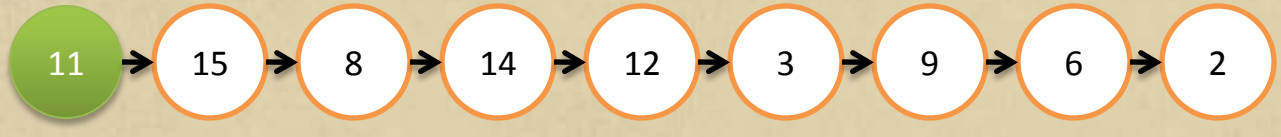



i=1, k=4



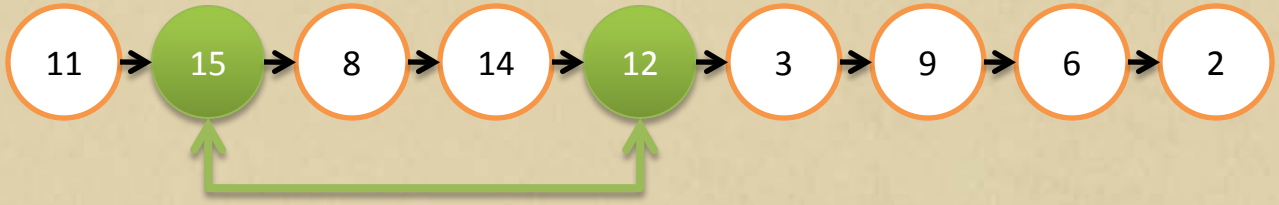
## ახალი მარშრუტი:

1) ახალ მარშრუტს ემატება 0-დან i-დე კვანძები საწყისი მარშრუტიდან, ამ შემთხვევაში “11”-ი.

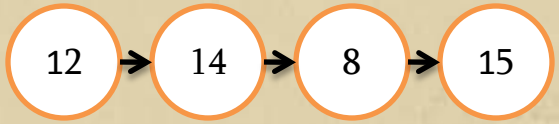



ანუ ახალი მარშრუტი გამოიყურება ასე:


2) შემდეგ ახალ მარშრუტს ემატება i-დან k+1-დე კვანძები (შეტრიალებული) საწყისი მარშრუტიდან.

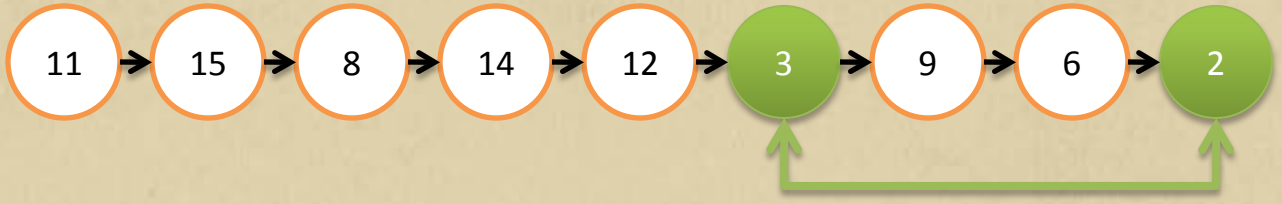



ანუ ახალ მარშრუტს დაემატება შემდეგი კვანძები:

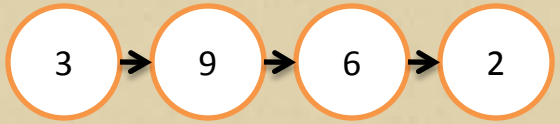


და ახალ მარშრუტს ექნება შემდეგი სახე:


3) ბოლოს ახალ მარშრუტს ემატება k+i-დან დაწყებული ბოლო კვანძის ჩათვლით ყველა კვანძი საწყისი მარშრუტიდან.




ანუ ახალ მარშრუტს დაემატება შემდეგი კვანძები:




და ახალ მარშრუტს ექნება შემდეგი სახე:

## დაფარვის ამოცანის რეალიზაცია მონტე-კარლოს მეთოდით

მაგალითისათვის განვიხილოთ დაფარვის ამოცანა, როდესაც ერთ-ერთ ორგანიზაციას ესაჭიროება განსხვავებული ენის სპეციალისტები-თარჯიმნები. კონკურსის საფუძველზე შეიქმნა შესაძლო კანდიდატთა სია, რომელთა შორის არიან ისეთი პერსონები, რომლებმაც კომპანიისთვის სასურველი ენებიდან მინიმუმ ერთი ენა მაინც იციან. კომპანიისთვის დაისვა შემდეგი ამოცანა: კანდიდატებიდან გაკეთდეს ისეთი შერჩევა, რომ 1. ყოველი ენისთვის შერჩეული სპეციალისტებიდან მოიძებნებოდეს სულ მცირე ერთი სპეციალისტი, რომელმაც იცის ეს ენა; 2. შერჩეული კანდიდატების მიერ სახელფასო მოთხოვნილი თანხების ჯამი (შერჩეული სპეციალისტების ჯამური ხელფასი) იყოს მინიმალური.

ვთქვათ კომპანიას ესაჭიროება 6 ენის სპეციალისტი და აქვს შესაბამისი კანდიდატების სია.

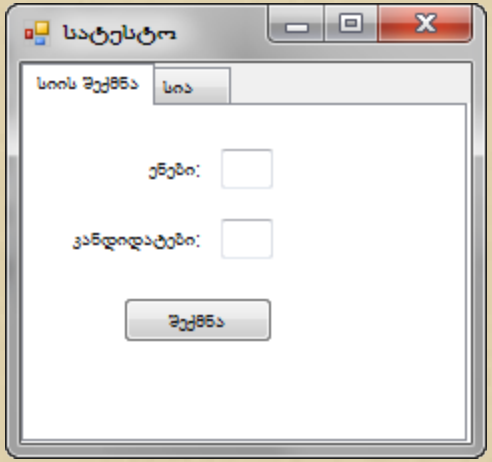
ენების სია:

| # | ენა |
|---|---|
| 1 | ქართული |
| 2 | ინგლისური |
| 3 | გერმანული |
| 4 | ფრანგული |
| 5 | იტალიური |
| 6 | რუსული |

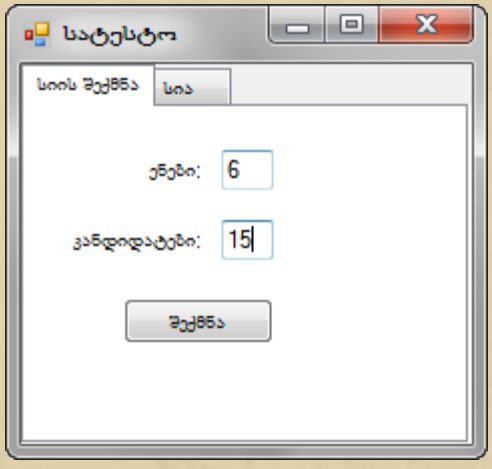
კანდიდატების სია:

| კანდიდატი | ენები რომელიც კანდიდატმა იცის | ანაზღაურება |
|---|---|---|
| კანდიდატი 1 | 1 | 100 ლარი |
| კანდიდატი 2 | 4 | 350 ლარი |
| კანდიდატი 3 | 1, 2 | 1000 ლარი |
| კანდიდატი 4 | 1, 2 | 700 ლარი |
| კანდიდატი 5 | 1 | 500 ლარი |
| კანდიდატი 6 | 3, 4 | 460 ლარი |
| კანდიდატი 7 | 5 | 320 ლარი |
| კანდიდატი 8 | 1 | 150 ლარი |
| კანდიდატი 9 | 5, 6 | 1200 ლარი |
| კანდიდატი 10 | 4, 5 | 800 ლარი |
| კანდიდატი 11 | 2, 3 | 900 ლარი |
| კანდიდატი 12 | 3 | 700 ლარი |
| კანდიდატი 13 | 5 | 600 ლარი |
| კანდიდატი 14 | 2 | 420 ლარი |
| კანდიდატი 15 | 4 | 320 ლარი |

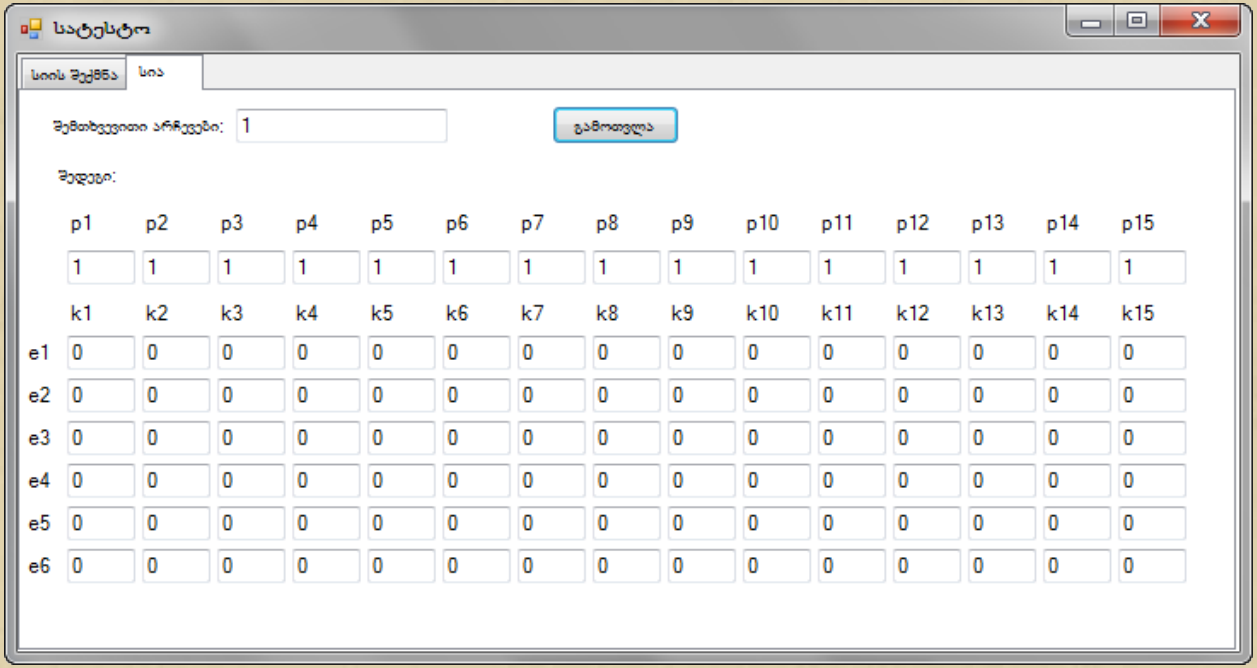
საწყისი მდგომარეობით პროგრამა ასე გამოიყურება:




ველში „ენები“ ვუთითებთ ენების რაოდენობას, ამ შემთხვევაში 6-ს, ხოლო ველში „კანდიდატები“ - შესაბამისად კანდიდატების რაოდენობას, ამ შემთხვევაში 15-ს:

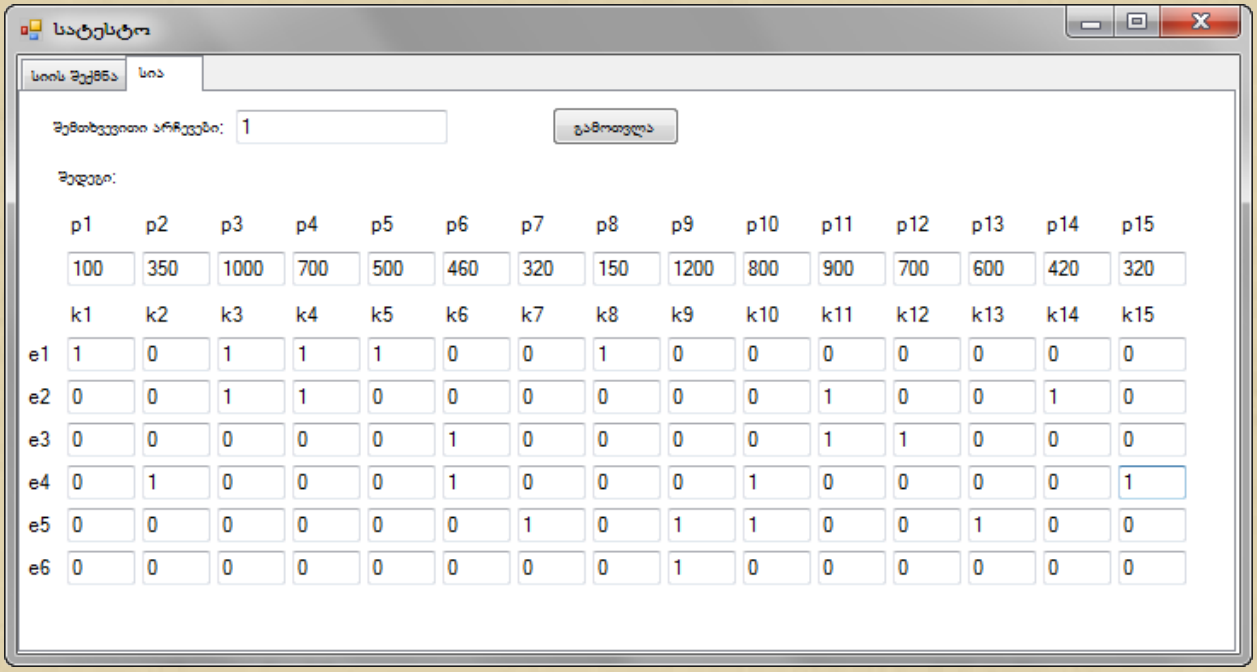


და ვაჭერთ ღილაკს, რის შედეგად იქმნება შესაბამისი ცხილი:




p1, p2, …, p15 უჯრებში ვწერთ k1, k2, …, k15 კანდიდატების ანაზღაურებას. k1, k2, …, k15 სვეტები შეესაბამება კანდიდატებს, ხოლო e1, e2, …, e6 სტრიქონები კი ენებს. სვეტისა და სტრიქონის გადაკვეთაზე მდებარე უჯრაში უნდა ჩავწეროთ 1 ან 0 იმის მიხედვით იცი თუ არა კონკრეტულმა კანდიდატმა კონკრეტული ენა. მაგ.: k1 სვეტისა და e1 სტრიქონის გადაკვეთაზე უნდა ჩაიწეროს 1 რადგან პირველმა კანდიდატმა იცის პირველი ენა (ქართული). ხოლო k1 სვეტის დანარჩენ სტრიქონებთან გადაკვეთაში უნდა ჩაიწეროს 0, რადგან პირველმა კანდიდატმა მხოლოდ პირველი ენა იცის.

ჩვენი სიების მიხედვით ცხრილი მიიღებს შემდეგ სახეს:




| | p1 | p2 | p3 | p4 | p5 | p6 | p7 | p8 | p9 | p10 | p11 | p12 | p13 | p14 | p15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 100 | 350 | 1000 | 700 | 500 | 460 | 320 | 150 | 1200 | 800 | 900 | 700 | 600 | 420 | 320 |
| | k1 | k2 | k3 | k4 | k5 | k6 | k7 | k8 | k9 | k10 | k11 | k12 | k13 | k14 | k15 |
| e1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| e2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| e3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| e4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| e5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| e6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

„შემთხვევითი არჩევების“ რაოდენობაში იწერება ის რაოდენობა თუ რამდენჯერ უნდა აირჩიოს პროგრამამ შემთხვევითად კანდიდატები მოცემული კანდიდატების სიიდან. ყოველ შემთხვევით დაგენერირებაზე ხდება შემდეგი რამ: პროგრამა აგენერირებს იმდენ შემთხვევით რიცხვს, 0-ს ან 1-ს, რამდენი კანდიდატიცაა და წერს ამ შემთხვევით რიცხვებს მასივში. მაგალითად: 0 1 0 0 1 1 0 0 0 1 1 0 0 0 1 . გენერირება ხდება შემდეგი კოდით:

```
var ran = new Random();

for (int r = 0; r < jM; r++) /* jM ცვლადი ტოლია კანდიდატების რაოდენობისა*/
{
ranArr[0, r] = ran.Next(2);
}
```

შემდეგ დგინდება არჩეული კანდიდატებით კმაყოფილდება თუ არა პირობა, რომ ყველა ენისათვის უნდა მოიძებნებოდეს თითო სპეციალისტი მაინც:

```
for (int r = 0; r < iN; r++) /* iN ენების რაოდენობაა */
{
int temp = 0;
for (int m = 0; m < jM; m++)
      {
if (ranArr[0,m] > 0) { temp = temp + valueArr[r, m]; } /* კონკრეტული ენისათვის მოწმდება იცის თუ არა ის რომელიმე სპეციალისტმა და შესაბამისად temp ცვლადი 0-ზე მეტი ხდება */
}

if (temp == 0) { daf[r] = 0; } /* თუ temp ცვლადი 0-ის ტოლია ე.ი. ამ ენისთვის სპეციალისტი არ მოიძებნა და daf მასივში შესაბამისი ენის ადგილზე იწერება 0 */
else { daf[r] = 1; } /* თუ  temp დადებითია ე.ი. შესაბამისი ენისთვის მოიძებნა სპეციალისტი და daf მასივში შესაბამისი ენის ადგილზე იწერება 1  */
}
```

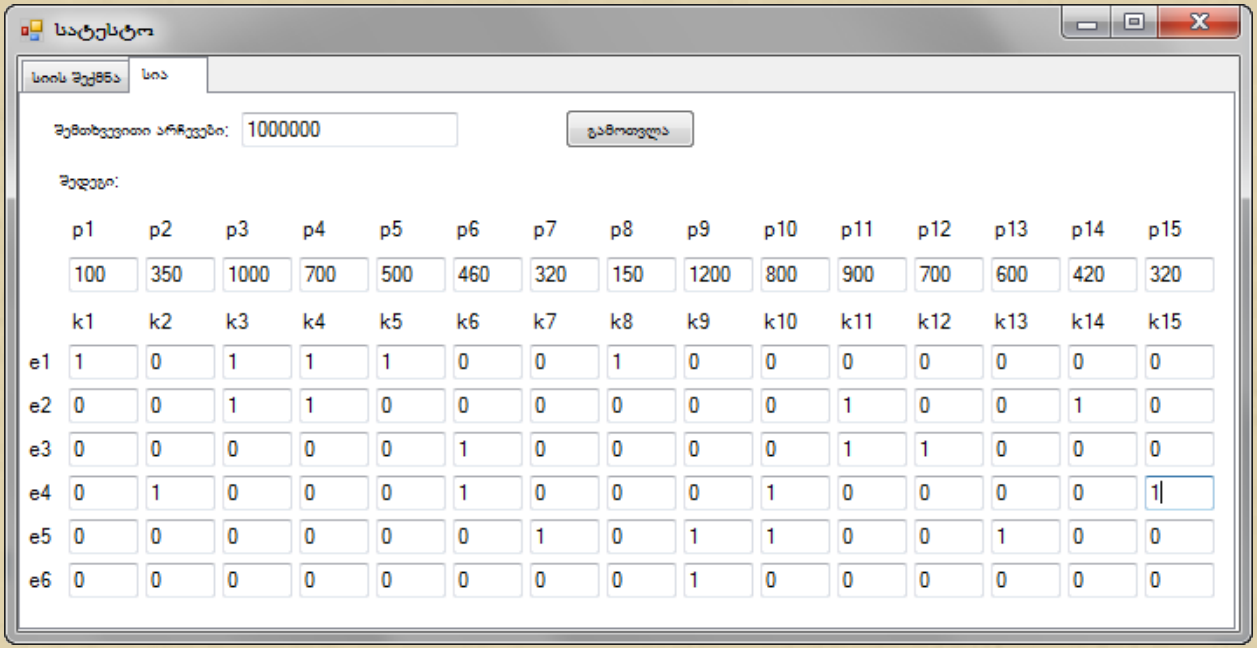
ამ კოდის შესრულების შემდეგ სრულდება შემდეგი კოდი:

```
if (daf.Sum() == iN) /* თუ daf მასივის ელემენტების ჯამი უდრის ენების საერთო რაოდენობას (iN) ე.ი.
ყველა ენისთვის მოიძებნა შესაბამისი ერთი სპეციალისტი მაინც და კონკრეტული შემთხვევითი არჩევის
მიხედვით დანახარჯი გამოითვლა. წინააღმდეგ შემთხვევაში პროგრამის შესრულება ამ ნაწილში წყდება,
გენერირდება ახალი შემთხვევითი რიცხვები/კანდიდატების სია, და გადადის ისევ ზემოთ აღწერილი
კოდის შესრულებაზე */
{
float temp = 0;

                if (first == 0) /* სრულდება მაშინ  თუ ეს პირველი შემთხვევაა როდესაც შემთხვევითმა სიამ
დააკმაყოფილა პირობა */
      {
                for (int m = 0; m < jM; m++)
            {
                if (ranArr[0, m] > 0) { temp += prices[m]; } /* შემთხვევითი სიის მიხედვით დანახარჯების
დათვლა */
                ranArr[1, m] = ranArr[0, m]; /* შემთხვევითი სიის ელემენტის დამახსოვრება, როგორც ყველაზე
ნაკლები დანახარჯის მქონესი */
            }

            minPrice = temp; /* მინიმალურ ფასს ენიჭება პირველივე ვარგისი შემთხვევითი სიის მიხედვით
დათვლილი ხარჯი */
            first++;
      }

      else /* სრულდება მაშინ  როდესაც ეს უკვე აღარაა პირველი შემთხვევა, როდესაც შემთხვევითმა სიამ
დააკმაყოფილა პირობა  */
      {
                for (int m = 0; m < jM; m++)
            {
                if (ranArr[0, m] > 0) { temp += prices[m]; } /* შემთხვევითი სიის მიხედვით დანახარჯების
დათვლა */
            }
```

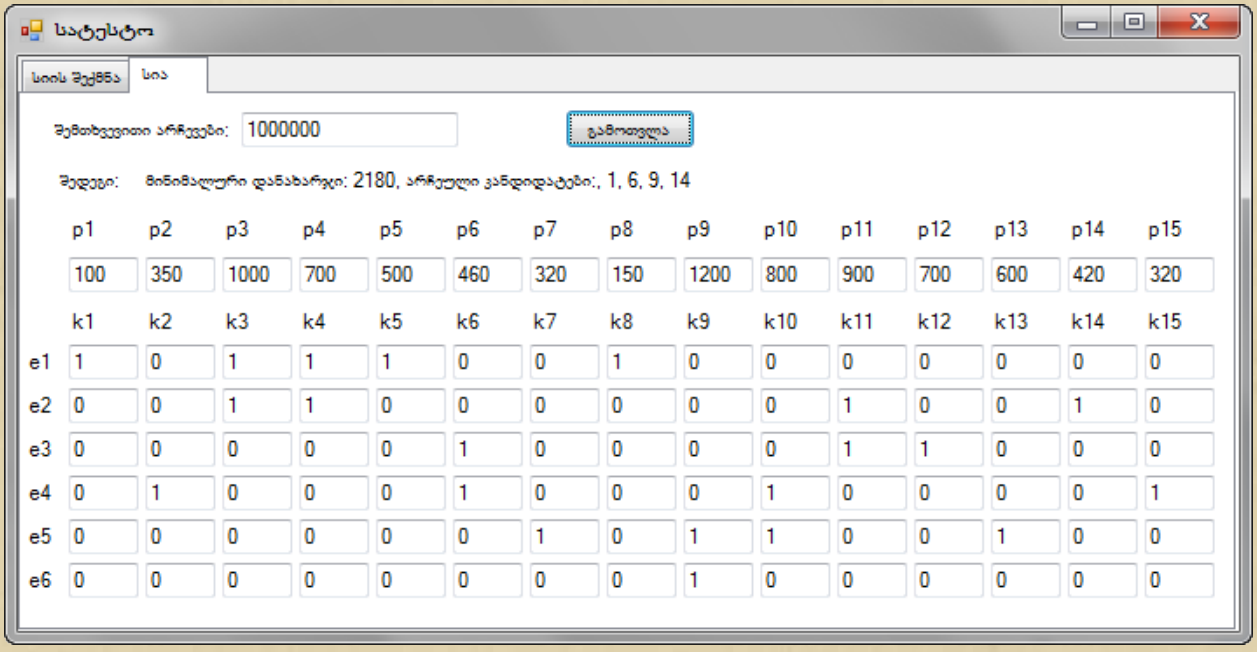
```
            if (temp < minPrice) /* სრულდება თუ ახალი შემთხვევითი სიის მიხედვით დათვლილი ხარჯი ნაკლებია წინა ხარჯზე */
            {
                minPrice = temp; /* მინიმალურ ფასს ენიჭება ახალი/უკეთესი სიის მიხედვით დათვლილი ხარჯი */
                for (int m = 0; m < jM; m++)
                {
                    ranArr[1, m] = ranArr[0, m]; /* შემთხვევითი სიის ელემენტის დამახსოვრება, როგორც ყველაზე ნაკლები დანახარჯის მქონესი */
                }
            }
        }

    }
```

ზემოთ მოყვანილი კოდის ფრაგმენტი სრულდება იმდენჯერ, რამდენი „შემთხვევითი არჩევების“ რიცხვსაც მივუთითებთ. შესაბამისად რაც უფრო დიდი იქნება ეს რიცხვი მით უფრო ზუსტი (უკეთესი) იქნება შედეგი რადგან ეს ალგორითმი მიეკუთვნება ევრისტიულ ალგორითმებს. მაგალითად თუ „შემთხვევითი არჩევების“ რაოდენობაში მივუთითებთ 1-ს შეიძლება პირველივე შედეგი აღმოჩნდეს 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 რაც ნიშნავს, რომ კანდიდატების სიიდან მოხდა მე-2-ე, მე-5-ე და მე-6-ე კანდიდატების არჩევა, ეს არჩევა კი საერთოდ არ აკმაყოფილებს იმ პირობას რომ თითო საგანი თითო კადნდიდატმა მაინც უნდა იცოდეს.

ახლა კი პროგრამაში მივუთითო ბოლო პარამეტრი, „შემთხვევითი არჩევების“ რაოდენობა



ამ შემთხვევაში 1000000 და ღილაკზე „გამოთვლა“ დაჭერით გავუშვათ პროგრამას შესრულებაზე. პროგრამის შესრულების შემდეგ მივიღებთ ასეთ შედეგს:




| | p1 | p2 | p3 | p4 | p5 | p6 | p7 | p8 | p9 | p10 | p11 | p12 | p13 | p14 | p15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 100 | 350 | 1000 | 700 | 500 | 460 | 320 | 150 | 1200 | 800 | 900 | 700 | 600 | 420 | 320 |
| | k1 | k2 | k3 | k4 | k5 | k6 | k7 | k8 | k9 | k10 | k11 | k12 | k13 | k14 | k15 |
| e1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| e2 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| e3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| e4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| e5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| e6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

ანუ პროგრამამ შეარჩია შემდეგი კანდიდატები: კანდიდატი 1, კანდიდატი 6, კანდიდატი 9 და კანდიდატი 14. ამ კანდიდატების ხელფასების ჯამი გამოვიდა 100+460+1200+420=2180 რაც ამ კონკრეტული შემთხვევისათვის არის საუკეთესო ვარიანტი. და რა თქმა უნდა ასევე შესრულდა ის პირობა, რომ თითოეული საგნისთვის მინიმუმ ერთი სპეციალისტი მაინც მოიძებნება:

| # | ენა | კანდიდატი |
|---|---|---|
| 1 | ქართული | კანდიდატი 1 |
| 2 | ინგლისური | კანდიდატი 14 |
| 3 | გერმანული | კანდიდატი 6 |
| 4 | ფრანგული | კანდიდატი 6 |
| 5 | იტალიური | კანდიდატი 9 |
| 6 | რუსული | კანდიდატი 9 |