

# Mađarski Metod

Problem dodele posla predstavlja jedan od osnovnih problema kombinatorijalne optimizacije. U svojoj najuopštenijoj formi, problem se može opisati na sledeći način: Instanca konkretnog problema ima određen broj agenata i određen broj zadataka. Bilo kojem agentu može biti dodeljen bilo koji zadatak, što će sa sobom povući određenu cenu u zavisnosti od toga koji zadatak je dodeljen kom agentu. Zahteva se da se odradi što je više zadataka moguće dodeljivanjem najviše jednog agenta svakom zadatku, i obratno, na takav način da je totalna cena ovih dodela optimalna. Ukoliko je broj agenata i zadataka jednak, onda je to problem balansirane dodele. U suprotnom, naziva se problem nebalansirane dodele. Formalna definicija problema dodele je sledeća: Data su dva skupa,  $A$  i  $T$  koji su jednake veličine, zajedno sa težinskom funkcijom  $C: A \times T \rightarrow \mathbb{R}$ . Naći bijekciju  $f: A \rightarrow T$  takvu da funkcija cene

$$\sum_{a \in A} C(a, f(a))$$

bude minimalna.

Naivno rešenje za problem dodele jeste da se proverí svaka dodela kako bi se izračunala njena cena. Ovo može biti veoma neefikasno jer, sa  $n$  agenata i  $n$  poslova, postoji  $n!$  različitih dodela. Iz ovog razloga, razvijeni su mnogi algoritmi za rešavanje algoritma u polinomijalnom vremenu.

Problem dodele predstavlja specijalni slučaj transportnog problema, koji predstavlja specijalni slučaj problema protoka sa minimalnom cenom, koji predstavlja specijalni slučaj linearnog programiranja. Iako je moguće rešiti bilo koji od ovih problema koristeći simplex metod, svaka specijalizacija ostavlja prostora za dodatne optimizacije koje bi određeni algoritmi mogli da iskoriste.

Jedan od prvih algoritama u polinomijalnom vremenu za rešavanje problema balansirane dodele jeste Mađarski algoritam. Ovaj algoritam je razvio i objavio Harold Kan, 1955. godine. Nazvao ga je Mađarski algoritam jer je svoj rad zasnovao na radu dvojice mađarskih matematičara: Dénes Kőnig i Jenő Egerváry.

Madjarski algoritam ,kako je ovde implementiran, se odvija u 7 koraka:

### 1. Transformacija koeficijenata matrice

Nađe se najmanji element po svakom redu i taj element se oduzme od svih elemenata tog reda. To se takođe uradi za svaku kolonu. Funkcije *RowReduction* i *ColReduction* obavljaju ovaj postao ja jednostavan način, iteracijom kroz ulaznu matricu.

### 2. Određivanje nezavisnih nula

Za svaki red se određuje broj nula i kreće se od reda koji ima minimalan broj nula. U tom redu se označava jedna kao nezavisna a ostale nule u tom redu i koloni se označavaju kao zavisne. Kraj koraka je kada se prođe kroz sve redove u matrici. Posao funkcije *FindNextMinZerosRow* jeste da odredi sledeći red za obrađivanje tako što će vratiti indeks sledećeg reda sa najmanjim brojem nula koji nije obrađen. *IndependentMarker* će iskoristiti tu informaciju kako bi u datom redu tražila nulu. Ukoliko nula bude pronađena, sve ostale nule koje se nalaze u datom redu ili koloni biće odbačene i neće moći da postanu nove nezavisne u nule u narednim iteracijama. Za ovo označavanje koristi se matrica koja je istih dimenzija kao ulazna i ima jedinice na mestima nezavisnih nula koje je algoritam odredio, a nule na svim ostalim mestima. Ova matrica predstavlja ulaz koji će naredni koraci koristiti za dalji progres algoritma.

### 3. Označavanje redova bez nezavisnih nula

Za svaki red se određuje broj nezavisnih nula i označavaju se samo oni redovi koji nemaju nezavisnih nula. Funkcija *MarkRowsWithoutIndependentZeros* uzeće matricu nezavisnih nula iz prethodnog koraka i u novi niz boolean vrednosti će staviti vrednost True na one indekse koji odgovaraju indeksima redova koji nemaju nijednu nezavisnu nulu.

### 4. Precrtavanje kolona

Precrtavaju se sve kolone koje imaju nule u označenim redovima.

*CrossoutColumnsWithZerosInMarkedRows* koristi niz boolean vrednosti iz prethodnog koraka kako bi mogla da označi odgovarajuće kolone. Na isti način kao u prethodnom koraku, oformiće se novi niz boolean vrednosti koji će sadržati informaciju o tome koje kolone su precrtane a koje ne.

### 5. Označavanje redova

Označavaju se redovi koji imaju nezavisne nule u precrtanim kolonama. Na isti način kao ranije, niz bool vrednosti koji nosi informaciju o tome koji redovi su označeni a koji ne (iz koraka 3.), sad će biti dopunjen označavanjem redova koji

sadrže bar jednu nezavisnu nulu u kolonama koje su u prethodnom koraku označene kao precrtane.

Koraci 4 i 5 se ponavljaju dok god se pri ponovnom označavanju redova u koraku 5 desi neka promena (ako se označi još jedan red koji pre toga nije bio označen)

## 6. Precrtavanje neoznačenih redova

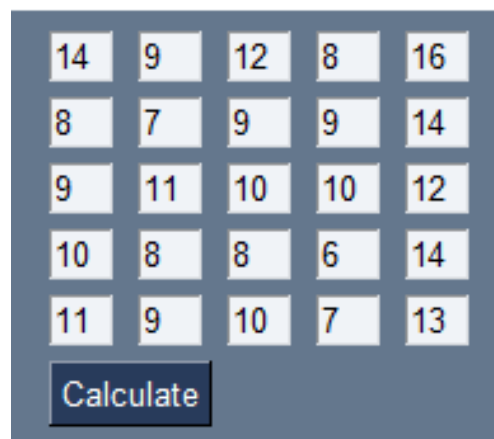
Ovaj korak je veoma jednostavan iz razloga što u ovom trenutku redovi mogu biti ili označeni ili precrtani, što znači da svi redovi koji u prethodnim koracima nisu označeni, predstavljaju zapravo precrtane redove.

## 7. Transformacija koeficijenata matrice

Zadatak funkcije *FindMinNonCrossedOut* je da, među redovima i kolonama koji nisu precrtani pronađe najmanji element. U funkciji *FinalMatrixTransform* se taj element se oduzme od svih elemenata među redovima i kolonama koji nisu precrtani, a dodaje se na elemente koji se nalaze na presecima precrtanih redova i kolona. Ostali elementi se samo prepisuju.

Svi ovi koraci ponavljaju se sve dok u svakom redu i svakoj koloni ne postoji tačno jedna nezavisna nula. Ova provera se vrši u *CheckFinished* funkciji.

Svi prethodno navedeni koraci ujedinjeni su u funkciji *HungarianAlgorithm* koja će za ulaz uzeti matricu koju korisnik unese u napravljeni GUI, a vratiće krajnju matricu u kojoj će za svaki red i svaku kolonu biti označena tačno jedna nezavisna nula. Zatim, program će za korisnika interpretirati značenje ove matrice tako što će ispisati kojem redu je dodeljena koja kolona. Ova interpretacija predstavlja optimalan način da se na osnovu date matrice cena napravi traženi raspored.



14	9	12	8	16
8	7	9	9	14
9	11	10	10	12
10	8	8	6	14
11	9	10	7	13

Calculate

Slika 1 Izgled unetog primera u GUI

```
Radnik 1 treba da radi na poziciji 2  
Radnik 2 treba da radi na poziciji 1  
Radnik 3 treba da radi na poziciji 5  
Radnik 4 treba da radi na poziciji 3  
Radnik 5 treba da radi na poziciji 4  
Optimalna vrednost je:44
```

**Slika 2** Izgled rešenja dobijenog prilikom klika na dugme Calculate

Za testiranje same implementacije algoritma korišćeni su primeri sa predavanja i vežbi, kao i primeri nastali kao rezultat nasumičnog ubacivanja vrednosti u tabelu. Za primer sa predavanja (Slika 1) algoritam izbacuje rešenje u obliku teksta u konzoli (Slika 2).