# Transportni problem

## 1. Definicija i tumačenje problema

Transportni problem predstavlja specijalnu vrstu problema linearnog programiranja koji se formalno može definisati na sledeći način.

$$(\min)\ \sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij}$$

pri ograničenjima

$$\sum_{j=1}^{m} x_{ij} \le a_i, i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} \le b_j, j = 1, ..., m$$

$$x_{ij} \ge 0$$

Uobičajeno intuitivno tumačenje problema je sledeće.
Potrebno je preneti robu iz skladišta u prodavnice. Na raspolaganju imamo $m$ prodavnica i $n$ skladišta.

$a_i$ predstavlja ukupnu količinu robe u $i$-tom skladištu (*ponuda).*

$b_j$ predstavlja količinu zahtevane robe u $j$-toj prodavnci (*potražnja).*

$c_{ij}$ predstavlja cenu transporta jedne jedinice robe iz $i$-tog skladišta u $j$-tu prodavnicu.

$x_{ij}$ označava koliko jedinica robe treba prebaciti iz skladišta $i$ u prodavnicu $j$, odnosno to su ciljne promenljive.

Na osnovu ovoga očigledno je i šta sama ograničenja u formalnoj postavci intuitivno predstavljaju.

$$\sum_{j=1}^{m} x_{ij} \le a_i, i = 1, ..., n$$

nam govori da ukupna količina uzete robe iz $i$-tog skladišta ne sme prekoračiti ukupnu količinu robe *(ponudu)* u $i$-tom skladištu.

$$\sum_{i=1}^{n} x_{ij} \le b_j, j = 1, ..., m$$

nam govori da ukupna količina robe dovedene u $j$-tu prodavnicu ne sme prekoračiti količinu zahtevane robe *(potražnju)* u $j$-toj prodavnici.

Ovakav model problema pogodan je za matrično predstavljanje, tako da problem možemo prikazati matrično na sledeći način.

$$
\begin{array}{ccccc}
c_{11} & c_{12} & \dots & c_{1m} & a_1 \\
c_{21} & c_{22} & \dots & c_{2m} & a_2 \\
\dots & \dots. & \dots & & \\
c_{n1} & c_{n2} & \dots & c_{nm} & a_n \\
b_1 & b_2 & \dots & b_m &
\end{array}
$$

## 2. Balansiran i nebalansiran problem

Prirodno se nameće pitanje regulisanja slučaja kada se ukupna ponuda i ukupna potražnja razlikuju. Stoga uvodimo pojmove *balansiranog* i *nebalansiranog* transportnog problema.

Za transportni problem kažemo da je ***balansiran*** ukoliko je ukupna ponuda jednaka ukupnoj potražnji, odnosno ako važi

$$\sum_{i=1}^{n} a_i = \sum_{j=1}^{m} b_j$$

U suprotnom kažemo da je problem ***nebalansiran***.

Primetimo da u slučaju balansiranog transportnog problema možemo nametnuti još jači uslov, tj uslov da sva roba bude raspoređena. Zbog toga nejednakosti u ograničenjima mogu postati jednakosti.

Ukoliko je problem nebalansiran potrebno ga je svesti na balansirani problem uvođenjem *fiktivnih skladišta (redova)* ili *fiktivnih prodavnica (kolona)*. Naime, po samoj definiciji nebalansiranosti problema znamo da važi

$$\sum_{i=1}^{n} a_i \neq \sum_{j=1}^{m} b_j$$

To možemo razdvojiti na dva podslučaja:

***1. Ponuda je veća od potražnje***

$$\sum_{i=1}^{n} a_i > \sum_{j=1}^{m} b_j$$

U ovom slučaju potrebno je uvesti *fiktivnu kolonu* koja će predstavljati još jednu fiktivnu prodavnicu koja će zahtevati onoliko robe koliko ima ,,viška“.

***2. Potražnja je veća od ponude***

$$\sum_{i=1}^{n} a_i < \sum_{j=1}^{m} b_j$$

U ovom slučaju potrebno je uvesti *fiktivni red* koji će predstavljati još jedno fiktivno skladište koje će imati onoliko robe koliko ima ,,manjka“.

U oba slučaja za cenu transporta robe u novonastalom redu/koloni poželjno je koristiti neku vrednost koja je znatno veća od svih ostalih vrednosti cena transporta kako bi algoritam favorizovao ostale (realne) redove i kolone.

U nastavku sledi primer *python* koda koji demonstrira  proceduru balansiranja transportnog problema.

```python
"""
Ulaz:
    C - matrica cena
    a - ponude u skladištima
    b - potražnje u prodavnicama
Izlaz:
    tuple:
        (nova matrica cena,
         nove ponude u skladištima,
         nove potražnje u prodavnicama,
         indeksi fiktivnih redova,
         indeksi fiktivnih kolona)
"""
def balance_problem(C, a, b):
    supply = sum(a)                                     # Ukupna ponuda
    demand = sum(b)                                     # Ukupna potražnja

    if supply == demand:
        # Ukoliko su ponuda i potražnja jednake problem je već balansiran
        return C, a, b, [], []


    diff = abs(supply - demand)                         # Apsolutna vrednost razlike u ponudi i potražnji
    frow = []
    fcol = []
    if supply > demand:                                 # Prvi slučaj, ponuda je veća od potražnje
        new_col = np.repeat(DUMMY_VALUE, C.shape[0])    # Fiktivna kolona
        C = np.hstack((C, new_col.reshape(-1, 1)))
        b = np.append(b, diff)
        fcol = [C.shape[1] - 1]
    else:                                               # Drugi slučaj, potražnja je veća od ponude
        new_row = np.repeat(DUMMY_VALUE, C.shape[1])    # Fiktivni red
        C = np.vstack((C, new_row))
        a = np.append(a, diff)
        frow = [C.shape[0] - 1]

    return C, a, b, frow, fcol
```

## 3. Metod minimalnih cena

Kao u većini problema linearnog programiranja, ideja je početi od nekog dopustivog rešenja i zatim iterativno vršiti popravke tog rešenja tako da teži ka optimalnom. *Metod minimalnih cena* je jedan od metoda za nalaženje početnog dopustivog rešenja transportnog problema. Prvo ćemo videti primer metoda minimalnih cena a zatim i konkretnu *python* implementaciju.

Neka je data naredna matrica problema:

| | *0* | *1* | *2* | *a* |
|---|---|---|---|---|
| *0* | 5 | 7 | 8 | 70 |
| *1* | 4 | 4 | 6 | 30 |
| 2 | 6 | 7 | 7 | 50 |
| *b* | 65 | 42 | 43 | |

Počinjemo tako što tražimo minimalnu cenu u matrici.
U ovom slučaju to je 4 i odgovara ceni transporta iz skladišta 1 u prodavnicu 0

| | *0* | *1* | *2* | *a* |
|---|---|---|---|---|
| *0* | 5 | 7 | 8 | 70 |
| *1* | **4** | 4 | 6 | **30** |
| 2 | 6 | 7 | 7 | 50 |
| *b* | **65** | 42 | 43 | |

Zatim gledamo u ponudu i potražnju za dato skladište i prodavnicu, i uzimamo manju od te dve vrednosti kao broj jedinica robe za naše početno rešenje. U ovom slučaju to će biti 30. Sada za uzetu vrednost umanjimo i ponudu i potražnju. Primetimo da ovim postupkom garantujemo da će u svakom koraku bar jedna od ove dve vrednosti postati 0. Zato je potrebno da odgovarajući red ili kolonu u kom se nalazi nula uklonimo, jer tu robu smatramo iskorišćenom. U ovom slučaju uklanjamo red 1, jer se njegova vrednost anulira. Sada imamo novu matricu problema (levo) i trenutno rešenje (desno):

| | *0* | *1* | *2* | *a* |
|---|---|---|---|---|
| *0* | 5 | 7 | 8 | 70 |
| *1* | **4** | 4 | 6 | **0** |
| 2 | 6 | 7 | 7 | 50 |
| *b* | **35** | 42 | 43 | |

| | *0* | *1* | *2* |
|---|---|---|---|
| *0* | 0 | 0 | 0 |
| *1* | **30** | 0 | 0 |
| 2 | 0 | 0 | 0 |

Nastavljamo proces, naravno izuzimajući izbačene redove i kolone (označene sa *x*). Sada je najmanja vrednost u tabeli 5, i ona odgovara redu 0 i koloni 0. Odgovarajuće vrednosti ponude i potražnje su 70 i 35. Oduzimajući 35 od obe anulira se vrednost u koloni 0, tako da tu kolonu izbacujemo, a 35 dodajemo u trenutno rešenje.

| | *0* | *1* | *2* | *a* |
|---|---|---|---|---|
| *0* | **5** | 7 | 8 | **70** |
| *x* | x | x | x | x |
| 2 | 6 | 7 | 7 | 50 |
| *b* | **35** | 42 | 43 | |

| | *0* | *1* | *2* |
|---|---|---|---|
| *0* | **35** | 0 | 0 |
| *1* | 30 | 0 | 0 |
| 2 | 0 | 0 | 0 |

Nastavljajući na isti način dobijamo:

| | x | *1* | *2* | *a* |
|---|---|---|---|---|
| *0* | x | **7** | 8 | **35** |
| *x* | x | x | x | x |
| 2 | x | 7 | 7 | 50 |
| *b* | x | **42** | 43 | |

| | *0* | *1* | *2* |
|---|---|---|---|
| *0* | 35 | **35** | 0 |
| *1* | 30 | 0 | 0 |
| 2 | 0 | 0 | 0 |

| | x | *1* | *2* | *a* |
|---|---|---|---|---|
| *x* | x | x | x | x |
| *x* | x | x | x | x |
| 2 | x | **7** | 7 | **50** |
| *b* | x | **7** | 43 | |

| | *0* | *1* | *2* |
|---|---|---|---|
| *0* | 35 | 35 | 0 |
| *1* | 30 | 0 | 0 |
| 2 | 0 | **7** | 0 |

| | x | *x* | *2* | *a* |
|---|---|---|---|---|
| *x* | x | x | x | x |
| *x* | x | x | x | x |
| 2 | x | x | **7** | **43** |
| *b* | x | x | **43** | |

| | *0* | *1* | *2* |
|---|---|---|---|
| *0* | 35 | 35 | 0 |
| *1* | 30 | 0 | 0 |
| 2 | 0 | **7** | 43 |

Na taj način dobijamo početno dopustivo rešenje:

$$X_0 = \begin{pmatrix} 35 & 35 & 0 \\ 30 & 0 & 0 \\ 0 & 7 & 43 \end{pmatrix}$$

Nenula pozicije smatramo bazisnim pozicijama, i koristimo oznaku $C_{ij}^B$ da označimo skup bazisnih cena.

Dopustivost rešenja nam garantuje činjenica da u svakom koraku biramo manju od dve vrednosti između ponude i potražnje, što znači da nikada nećemo prekoračiti ni jedno. Ovakvo rešenje, naravno, ne mora biti optimalno.

Takođe primetimo da ovaj proces ima tačno $m+n-1$ iteracija. Razlog za to je činjenica da u svakom koraku izbacujemo ili jedan red ili jednu kolonu, kojih ukupno ima $m+n$.

U nastavku sledi primer *python* koda koji demonstrira proceduru nalaženja početnog dopustivog rešenja metodom minimalnih cena.

```
"""
Ulaz:
    C - matrica cena
    a - ponude u skladištima
    b - potražnje u prodavnicama
Izlaz:
    X - početno dopustivo rešenje
"""
def min_cost_method(C, a, b):
    m, n = C.shape
    X = np.zeros(C.shape)

    # NAPOMENA: Nećemo zaista izbacivati kolone i redove jer na taj način se menjaju i indeksi
    # Zbog toga zapravo pamtimo indekse izbačenih kolona i redova
    removed_rows = np.array([], dtype='int')                        # Niz indeksa izbačenih redova
    removed_columns = np.array([], dtype='int')                     # Niz indeksa izbačenih kolona

    for _ in range(m + n - 1):
        p, q = ut.argmin_exclude(C, removed_rows, removed_columns)    # Tražimo indekse polja sa minimalnom cenom
        the_min = min(a[p], b[q])                                     # Manja od vrednosti ponude i potražnje za dato polje
        X[p][q] = the_min                                             # U rešenje upisujemo tu vrednost

        a[p] = a[p] - the_min                                         # Oduzimamo tu vrednost od ponude
        b[q] = b[q] - the_min                                         # Oduzimamo tu vrednost od potražnje


        if a[p] == 0:
            removed_rows = np.append(removed_rows, p)                  # Brišemo odgovarajući red ili kolonu
        elif b[q] == 0:
            removed_columns = np.append(removed_columns, q)

    return X
```

## 4. Metod potencijala

Nakon nalaženja početnog rešenja metodom minmalnih cena (ili nekim drugim metodom) imamo garanciju samo da je rešenje dopustivo, ali ne i da je optimalno. Optimizacija rešenja se izvodi iterativno, vršenjem odgovarajućih popravki nad početnim rešenjem. Jedan metod kojim se to realizuje je ***metod potencijala.***

Potencijali predstavljaju brojeve oblika $u_i, v_j$ gde $i = 1, ...n$ $j = 1, ...m$, odnosno dodeljuju se vektorima $a$ i $b$. Za početak ih određujemo tako da važi

$$c_{ij}^B - u_i - v_j = 0$$

Ovim smo zapravo definisali linearni sistem od $m+n-1$ jednačina i $m+n$ nepoznatih. Jednu nepoznatu ćemo ipak anulirati a to radimo tako što izaberemo onaj potencijal u čijoj vrsti ili koloni ima najviše bazisnih promenljivih, i njega postavimo na nulu. Na taj način sistem je od tačno $m+n$ nepoznatih i $m+n$ jednačina. Na našem primeru za bazisno rešenje:

$$X_0 = \begin{pmatrix} 35 & 35 & 0 \\ 30 & 0 & 0 \\ 0 & 7 & 43 \end{pmatrix}$$

pozicije bazisnih promenljivih su $B = \{(0,0),(0,1),(1,0),(2,1),(2,2)\}$. Na osnovu toga dobijamo opisani sistem jednačina:

$$\begin{aligned} u_0 + v_0 &= 5 \\ u_0 + v_1 &= 7 \\ u_1 + v_0 &= 4 \\ u_2 + v_1 &= 7 \\ u_2 + v_2 &= 7 \end{aligned}$$

Primetimo da sistem i dalje možemo smestiti u matricu tako što odredimo da prvih $n$ kolona označava $u$-ove a narednih $m$ označava $v$-ove. Zato je sistem u standardnom $Ax = b$ obliku:

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 4 \\ 7 \\ 7 \end{pmatrix}$$

Sada treba pronaći potencijal za anuliranje, na već pomenut način. U ovom primeru to će biti potencijal $u_0$ (primetimo da smo mogli izabrati bilo koji sa dve bazne promenljive u redu/koloni). Modifikovan sistem sada izgleda ovako:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 4 \\ 7 \\ 7 \end{pmatrix}$$

I njegovo rešenje je $(0, -1, 0, 5, 7, 7)$, odnosno:

$$u_0 = 0, u_1 = -1, u_2 = 0, v_0 = 5, v_1 = 7, v_2 = 7$$

Nakon ovoga ja potrebno vršiti popravku počevši od neke nebazisne pozicije. Tu poziciju biramo kao najnegativniju za koju važi:

$$C_{ij}^N - u_i - v_j < 0$$

Ukoliko takvo $i, j$ ne postoji onda se algoritam zaustavlja, i trenutno bazisno rešenje je optimalno:

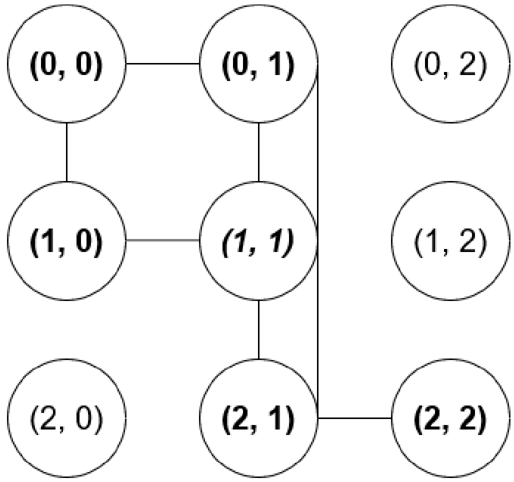
$$C_{ij}^N - u_i - v_j \geq 0 \quad \forall i, j \Rightarrow \text{STOP}$$

U našem slučaju vrednosti su sledeće:

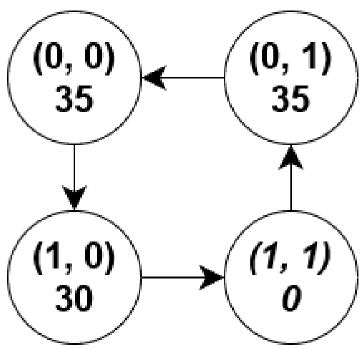
$$\begin{aligned} C_{02} - u_0 - v_2 &= 8 - 0 - 7 = 1 \\ C_{11} - u_1 - v_1 &= 4 + 1 - 7 = -2 \\ C_{12} - u_1 - v_2 &= 6 + 1 - 7 = 0 \\ C_{20} - u_2 - v_0 &= 6 - 0 - 5 = 1 \end{aligned}$$

Vidimo da uslov zaustavljanja nije ispunjen. Jedina negativna, samim tim i najnegativnija, vrednost postiže se za $(r, s) = (1, 1)$.

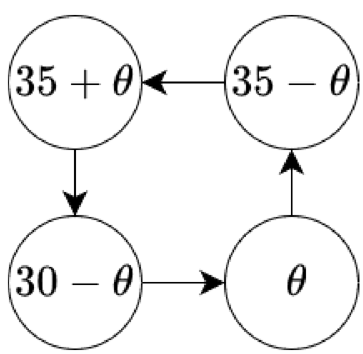
Počevši od ovog polja vršimo prepravke po bazisnim pozicijama. To radimo tako što konstruišemo cikl u grafu koga čine sve trenutno bazisne pozicije i jedna nebazisna pozicija $r, s$. Pritom u datom grafu ne smemo imati dijagonalne grane. Konstruisani graf za date pozicije izgleda ovako:



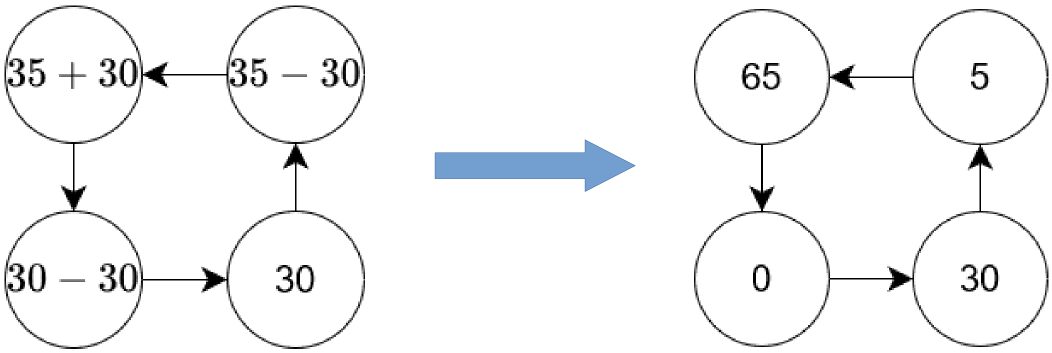

Kao što vidimo, uzimamo u obzir samo bazisne pozicije (podebljanje) i jednu nebazisnu poziciju (podebljanja i iskošena). Cikl tražimo tako da počinje u nebazisnoj poziciji. U ovom slučaju on će uključivati samo čvorove (0, 0), (0, 1), (1, 0), (1, 1), i izgledaće ovako (dopisane su trenutne vrednosti bazisnog rešenja za date promenljive).




Nebazisnoj poziciji dodeljujemo vrednost popravke $\theta$, dok se ostale promenljive u ciklu alternativno popravljaju sa $\pm\theta$.



Ostaje samo odabir početne $\theta$ vrednosti koja se bira kao minimalna trenutna bazisna vrednost u ciklu na negativnim pozicijama (na pozicijama gde su popravke $-\theta$). U ovom slučaju to je vrednost 30, i odgovara poziciji $(1,0)$. Primetimo da nakon primene popravke upravo ta promenljiva postaje anulirana, dok se promenljiva na poziciji $r, s$ menja sa nule na novu vrednost. Na taj način je efektivno odabrana promenljiva napustila bazu, a $x_{rs}$ ušla u bazu.




Na ovaj način dobili smo novo rešenje (popravku bazisnog rešenja):

$$X_1 = \begin{pmatrix} 65 & 5 & 0 \\ 0 & 30 & 0 \\ 0 & 7 & 43 \end{pmatrix}$$

Sada se postupak ponavlja dok se ne ispuni uslov zaustavljanja. Za nove potencijale dobijamo novi sistem jednačina:

$$\begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 4 \\ 7 \\ 7 \end{pmatrix}$$

Na već opisan način anuliramo potencijal $v_1$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ v_0 \\ v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 4 \\ 7 \\ 7 \end{pmatrix}$$

I dobijamo rešenje sistema:

$$u_0 = 3, u_1 = 0, u_2 = 3, v_0 = 2, v_1 = 4, v_2 = 4$$

Proveramo vrednosti izraza $C_{ij}^N - u_i - v_j$:

$$C_{02} - u_0 - v_2 = 8 - 3 - 4 = 1$$
$$C_{10} - u_1 - v_0 = 4 - 0 - 2 = 2$$
$$C_{12} - u_1 - v_2 = 6 - 0 - 4 = 2$$
$$C_{20} - u_2 - v_0 = 6 - 3 - 2 = 1$$

Primećujemo da su ovaj put svi pozitivni, tako da se algoritam ovde zaustavlja. Optimalno rešenje je trenutno bazisno rešenje, tj:

$$X_1 = \begin{pmatrix} 65 & 5 & 0 \\ 0 & 30 & 0 \\ 0 & 7 & 43 \end{pmatrix}$$

## 5. Tumačenje konačnog rešenja

Konačno rešenje nam govori sledeće:

- Potrebno je prebaciti 65 jedinica robe iz skladišta 0 u prodavnicu 0
    - cena je $65 * 5 = 325$
- Potrebno je prebaciti 5 jedinica robe iz skladišta 0 u prodavnicu 1
    - cena je $5 * 7 = 35$
- Potrebno je prebaciti 30 jedinica robe iz skladišta 1 u prodavnicu 1
    - cena je $30 * 4 = 120$
- Potrebno je prebaciti 7 jedinica robe iz skladišta 2 u prodavnicu 1
    - cena je $7 * 7 = 49$
- Potrebno je prebaciti 43 jedinice robe iz skladišta 2 u prodavnicu 2
    - cena je $43 * 7 = 301$

Sabiranjem cena dobijamo optimalnu vrednost funkcije cilja

$$325 + 35 + 120 + 49 + 301 = 830$$

Finalno rešenje problema (optimalna vrednost funkcije cilja i tačka za koju se ona dostiže):

$$\hat{f} = 830 \qquad \hat{X} = \begin{matrix} 65 & 5 & 0 \\ 0 & 30 & 0 \\ 0 & 7 & 0 \end{matrix}$$

## 6. Delovi koda

U nastavku slede isečci *python* koda koji implementira odabrane delove funkcionalnosti metoda potencijala. Kompletan kod može se naći na *github* repozitorijumu:
https://github.com/aleksakojadinovic/DS3/tree/master/04_transport_and_assignment_problem

*Biranje potencijala koji se anulira*

```python
"""
Ulaz:
    caps_mask - Binarna matrica
                caps_mask[i][j] == 1 : promenljiva (i, j) je bazisna
                caps_mask[i][j] == 0 : promenljiva (i, j) nije bazisna
Izlaz:
    tuple: (index, axis)
    index:
        - indeks reda ili kolone onog potencijala koji treba anulirati
    axis:
        - oznaka da li se radi o redu ili koloni
"""
def find_best_potential(caps_mask):
    best_row_index = None
    best_row_count = None
    for i, row in enumerate(caps_mask):
        basic_count = np.count_nonzero(row == 1)                    # broj bazisnih promenljivih u datom redu
        if best_row_index is None or basic_count > best_row_count:
            best_row_index = i
            best_row_count = basic_count

    best_column_index = None
    best_column_count = None
    for j in range(caps_mask.shape[1]):
        col = caps_mask[:, j]
        basic_count = np.count_nonzero(col == 1)                    # broj bazisnih promenljivih u datoj koloni
        if best_column_index is None or basic_count > best_column_count:
            best_column_index = j
            best_column_count = basic_count

    return (best_row_index, 0) if best_row_count > best_column_count else (best_column_index, 1)
```

*Metod potencijala*

```python
"""
Ulaz:
    C                   - matrica cena
    a                   - vektor a (ponuda)
    b                   - vektor b (potražnja)
    basis_solution      - početno rešenje, dobijeno metodom minimalnih cena
    caps                - Binarna matrica
                            caps[i][j] == 1 : promenljiva (i, j) je bazisna
                            caps[i][j] == 0 : promenljiva (i, j) nije bazisna
Izlaz:
    Matrica koja predstavlja rešenje

"""
def potential_method(C, a, b, basis_solution, caps):
    m, n = C.shape
    shape = C.shape
    iteration = 0
    while True:
        basis_indices = list(map(tuple, np.argwhere(caps == 1)))        # lista baznih indeksa
        non_basis_indices = list(map(tuple, np.argwhere(caps == 0)))    # lista nebaznih indeksa

        potentials_systemA = np.zeros((m + n - 1, m + n))               # sistem jednačina za nalaženje potencijala (A)
        potentials_systemB = np.zeros(m + n - 1)                        # sistem jednačina za nalaženje potencijala (b)
        for row, base_coords in enumerate(basis_indices):
            base_i = base_coords[0]
            base_j = base_coords[1]
            potentials_systemA[row][base_i] = 1.0
            potentials_systemA[row][m + base_j] = 1.0
            potentials_systemB[row] = C[base_i][base_j]

        to_set_zero_index, to_set_zero_axis = find_best_potential(caps) # nalazimo potencijal koji se naulira
        to_set_zero_actual_index = to_set_zero_index
        if to_set_zero_axis == 1:
            to_set_zero_index = m + to_set_zero_index

        potentials_systemA = np.delete(potentials_systemA, to_set_zero_actual_index, 1) # uklanjamo celu kolonu
        potential_system_solution = np.linalg.solve(potentials_systemA, potentials_systemB) # rešavamo sistem koristeći numpy biblioteku
        potential_system_solution = np.insert(potential_system_solution, to_set_zero_actual_index, 0)

        r = None
        s = None
        lowest_val = None
        for i, j in non_basis_indices:                              # biramo nebazisnu promenljivu od koje kreće popravka
            ui = potential_system_solution[i]
            vj = potential_system_solution[m + j]
            val = C[i][j] - ui - vj
            if val < 0:
                if lowest_val is None or val < lowest_val:
                    lowest_val = val
                    r = i
                    s = j

        if lowest_val is None:
            return basis_solution                                   # ukoliko ne postoji trenutno rešenje je optimalno

        graph = graphs.get_graph(r, s, caps)                                    # konstruišemo graf i nalazimo cikl
        cycle = graphs.find_cycle(graph, ut.pack_indices(r, s, shape), shape=shape)
        cycle_coordinates = list(map(lambda x: ut.unpack_index(x, shape), cycle))

        corr_theta_i = None                                                     # biramo početnu theta vrednost
        corr_theta_j = None
        corr_theta = None
        for idx, (i, j) in enumerate(cycle_coordinates):
            if idx % 2 == 1:
                if corr_theta is None or basis_solution[i][j] < corr_theta:
                    corr_theta_i = i
                    corr_theta_j = j
                    corr_theta = basis_solution[i][j]

        for idx, (i, j) in enumerate(cycle_coordinates):                        # vršimo popravke
            coeff = 1 if idx % 2 == 0 else -1
            basis_solution[i][j] += coeff * corr_theta

        basis_solution[r][s] = corr_theta

        caps[corr_theta_i][corr_theta_j] = 0                                    # jedna promenljiva izlazi iz baze
        caps[r][s] = 1                                                          # (r, s) ulazi u bazu

        iteration += 1
```

*Konstrukcija grafa*

```python
"""
Ulaz:
    theta_i, theta_j    - pozicija nebazisne promenljive
    caps                - Binarna matrica
                            caps[i][j] == 1 : promenljiva (i, j) je bazisna
                            caps[i][j] == 0 : promenljiva (i, j) nije bazisna
Izlaz:
    Liste povezanosti

"""
def construct_graph(theta_i, theta_j, caps):
    m, n = caps.shape
    shape = caps.shape
    # Liste povezanosti za cvorove
    # NAPOMENA: Cvorovi se `pakuju` tako sto se na zapisuju same koordinate vec m*i + j oblik
    graph_matrix = [[] for _ in range(m*n)]

    # Funkcija koja odredjuje da li cvor treba ukljuciti u graf
    # Cvor ukljucujemo u graf ako predstavlja baznu promenljivu ili ako je to bas cvor theta_i, theta_j
    def should_include_in_graph(ii, jj):
        return caps[ii][jj] == 1 or (ii, jj) == (theta_i, theta_j)

    for i in range(m):
        for j in range(n):
            if not should_include_in_graph(i, j):
                continue

            # Grane izmedju redove
            for other_row in range(m):
                if not should_include_in_graph(other_row, j) or i == other_row:
                    continue
                graph_matrix[ut.pack_indices(i, j, shape)].append(ut.pack_indices(other_row, j, shape))


            # Grane izmedju kolona
            for other_col in range(n):
                if not should_include_in_graph(i, other_col) or j == other_col:
                    continue
                graph_matrix[ut.pack_indices(i, j, shape)].append(ut.pack_indices(i, other_col, shape))

    return graph_matrix
```