

# Laporan UTS Riset Operasi

## *Transformation Problem*

Nuzha Musyafira

051116 4000 0014

# Sumpah dan Pernyataan

Demi Allah (Tuhan) Yang Maha Esa, maka dengan ini, saya bersumpah dan menyatakan dengan sebenar-benarnya bahwa saya mengerjakan jawaban soal Ujian Tengah Semester (UTS) ini secara sendiri dan mandiri, tidak melakukan kecurangan dalam bentuk apa pun, tidak menyalin / menjiplak / melakukan plagiat pekerjaan / karya orang lain, serta tidak menerima bantuan pengerjaan dalam bentuk apa pun dari orang lain. Saya bersedia menerima semua konsekuensi dalam bentuk apa pun, apabila saya ternyata terbukti melakukan kecurangan dan/atau penyalinan / penjiplakan / plagiat pekerjaan / karya orang lain.

Surabaya, 30 Oktober 2018

A handwritten signature in black ink, consisting of stylized, overlapping loops and a long horizontal stroke at the bottom.

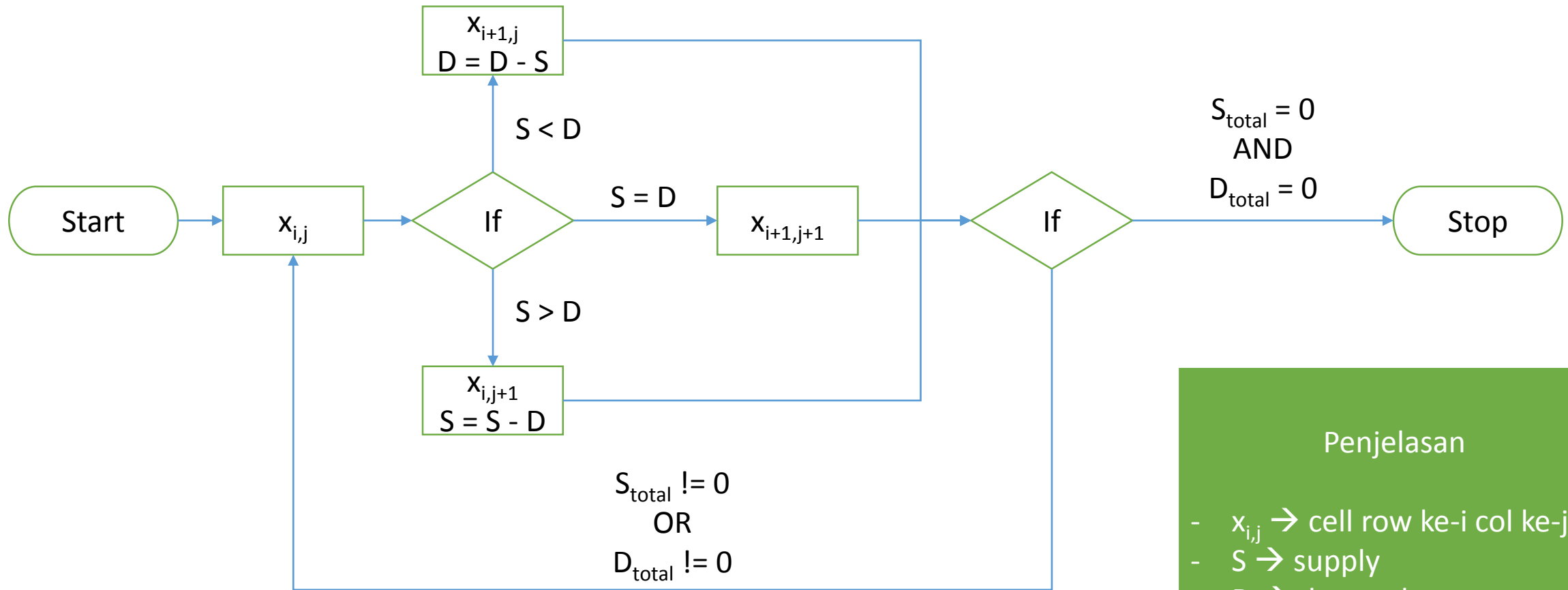
Nuzha Musyafira

051116 4000 0014

# METODE POJOK KIRI ATAS

Pada metode ini, dilakukan assign nilai dimulai dari cell pojok kiri atas ( $x_{11}$ ), dengan mengambil nilai maksimal yang dapat memenuhi nilai corresponding supply dan demandnya. Apabila masih terdapat sisa nilai untuk supply, maka berpindah ke cell sebelah kanannya ( $x_{i,j+1}$ ). Jika supply untuk row yang sama habis, maka berpindah ke cell di bawahnya ( $x_{i+1,j}$ ). Apabila supply dan demand sama-sama habis (supply = demand), maka berpindah ke cell  $x_{i+1,j+1}$  (diagonal). Pola ini diulangi hingga seluruh supply dan demand habis untuk semua cell.

# Diagram Alur



## Penjelasan

- $x_{i,j} \rightarrow$  cell row ke-i col ke-j
- $S \rightarrow$  supply
- $D \rightarrow$  demand

# Pseudocode

```
1 sTot=sum(supply)
2 dTot=sum(demand)
3 s=len(supply)
4 d=len(demand)
5 if sTot<dTot:
6     s+=1
7     supply.append(dTot - sTot)
8     sTot=sum(supply)
9     temp=[]
10    for x in range(len(demand)):
11        temp.append(0)
12    cost.append(temp)
13
14 elif sTot>dTot:
15     d+=1
16     demand.append(sTot - dTot)
17     dTot=sum(demand)
18     for x in range(len(supply)):
19         cost[x].append(0)
```

Jumlah total seluruh supply dan demand

Banyak supply dan demand

Jika total supply < demand (unbalanced), maka program akan otomatis menambah aktor supply baru yaitu dummy, dengan nilai cost 0, dan nilai supply sebanyak selisih total demand dan supply

Jika total supply > demand (unbalanced), maka program akan otomatis menambah aktor demand baru yaitu dummy, dengan nilai cost 0, dan nilai demand sebanyak selisih total supply dan demand

```
1 def pojokKiriAtas(s,d):  
2     table=[]  
3     for x in range(s):  
4         temp=[]  
5         for y in range(d):  
6             temp.append(0)  
7         table.append(temp)  
8     toAllocate(table,0,0)  
9     return table
```

Dibentuk table baru untuk menyimpan nilai pengalokasian ketika proses

Dengan ukuran baris sebanyak jumlah supply dan kolom sesuai jumlah demand, kita alokasikan nilai 0 sebagai nilai mula-mula

Memanggil fungsi pengalokasian dengan  $C_{0,0}$  sebagai inisiasinya (pojok kiri atas)

```

1  def toAllocate(table2,x,y):
2      global sTot,dTot
3      if sTot==0 and dTot==0:
4          return table2
5      if supply[x]<demand[y]:
6          table2[x][y]=supply[x]
7          demand[y]-=supply[x]
8          supply[x]=0
9          sTot=sum(supply)
10         dTot=sum(demand)
11         toAllocate(table2,x+1,y)
12     elif supply[x]>demand[y]:
13         table2[x][y]=demand[y]
14         supply[x]-=demand[y]
15         demand[y]=0
16         sTot=sum(supply)
17         dTot=sum(demand)
18         toAllocate(table2,x,y+1)
19     elif supply[x]==demand[y]:
20         table2[x][y]=supply[x]
21         supply[x]=0
22         demand[y]=0
23         sTot=sum(supply)
24         dTot=sum(demand)
25         toAllocate(table2,x+1,y+1)

```

Fungsi pengalokasian menggunakan recursive yang akan berhenti ketika total supply dan demand mencapai 0 (habis). x dan y mengindikasikan posisi pada table. Fungsi akan dipanggil dengan x dan y = 0 (pojok kiri atas).

Apabila nilai corresponding supply < demand, maka nilai yang dialokasikan pada  $C_{x,y}$  adalah nilai supply nya. Kemudian, kita kurangi nilai demand nya sebanyak nilai supply, lalu set nilai supply menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu recursive dengan x+1 karena bergeser ke bawah.

Apabila nilai corresponding supply > demand, maka nilai yang dialokasikan pada  $C_{x,y}$  adalah nilai demand nya. Kemudian, kita kurangi nilai supply nya sebanyak nilai demand, lalu set nilai demand menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu recursive dengan y+1 karena bergeser ke samping kanan.

Apabila nilai corresponding supply = demand, maka nilai yang dialokasikan pada  $C_{x,y}$  adalah nilai salah satunya. Kemudian, kita set nilai demand dan supply nya menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu recursive dengan x+1 dan y+1 karena bergeser ke diagonal kanan bawah.

# Studi Kasus 1

Gandum dipanen di Midwest (daerah pertanian Amerika bagian Tengah Barat) dan disimpan dalam cerobong butir gandum di tiga kota – Kansas City, Omaha, dan Des Moines. Ketiga cerobong butir gandum ini memasok tiga penggilingan tepung yang berlokasi di Chicago, St. Louis, dan Cincinnati. Butir-butir gandum tersebut dikirim ke penggilingan dengan menggunakan gerbong kereta api, yang tiap gerbongnya memuat satu ton gandum. Setiap bulannya, tiap cerobong butir gandum dapat memasok penggilingan sejumlah ton gandum berikut ini.

<b>Cerobong Butir Gandum</b>	<b>Jumlah yang ditawarkan</b>
1. Kansas City	150
2. Omaha	175
3. Des Moines	275



Jumlah ton gandum yang diminta per bulan dari tiap penggilingan adalah sebagai berikut :

<b>Penggilingan</b>	<b>Jumlah yang diminta</b>
A. Chicago	200
B. St. Louis	100
C. Cincinnati	300

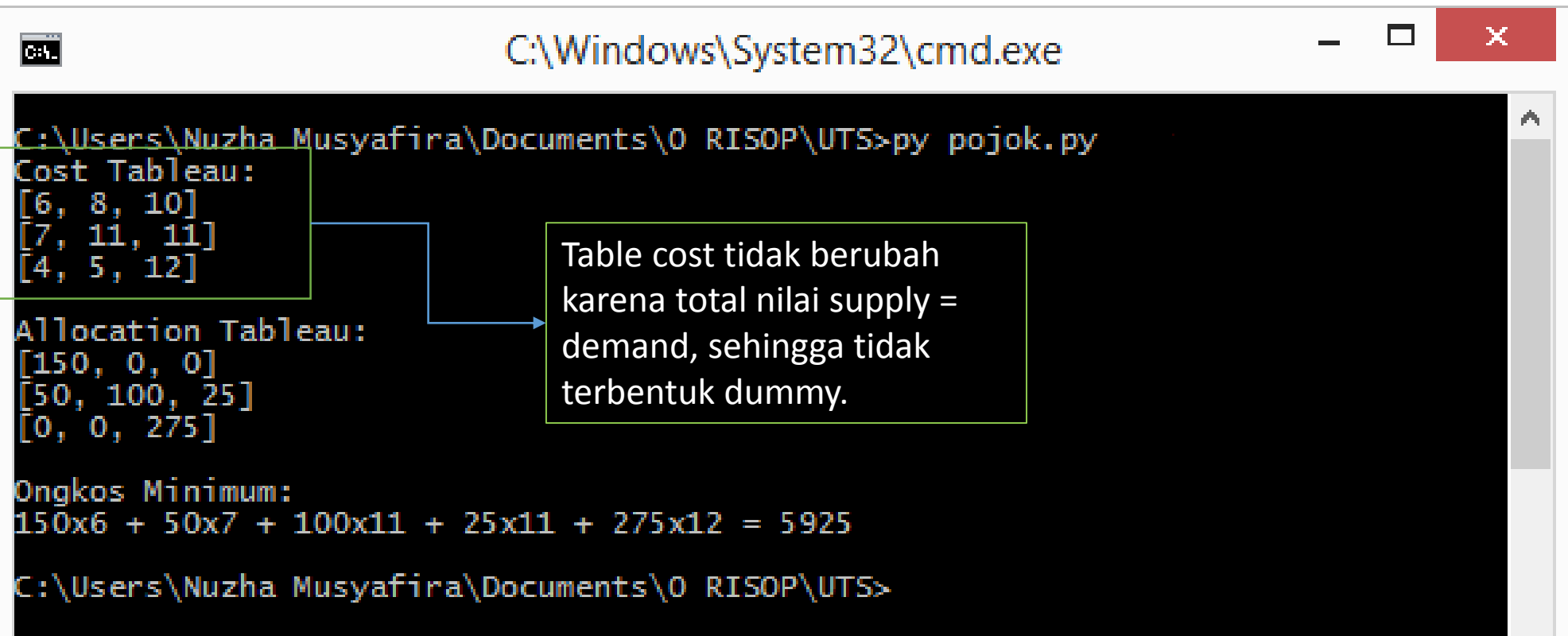
Biaya pengiriman (\$) :

Cerobong Butir Gandum	Penggilingan		
	Chicago (A)	St. Louis (B)	Cincinnati (C)
Kansas City	6	8	10
Omaha	7	11	11
Des Moines	4	5	12

Untuk menentukan berapa banyak ton gandum yang harus dikirim dari tiap cerobong butir gandum ke tiap penggilingan setiap bulannya agar total biaya transportasi minimum

<div>Ke</div> <div>Dari</div>	A	B	C	Pasokan
1	6	8	10	150
2	7	11	11	175
3	4	5	12	275
Permintaan	200	100	300	600

# Hasil Output



```
C:\Windows\System32\cmd.exe

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS>py pojok.py
Cost Tableau:
[6, 8, 10]
[7, 11, 11]
[4, 5, 12]

Allocation Tableau:
[150, 0, 0]
[50, 100, 25]
[0, 0, 275]

Ongkos Minimum:
150x6 + 50x7 + 100x11 + 25x11 + 275x12 = 5925

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS>
```

Table cost tidak berubah karena total nilai supply = demand, sehingga tidak terbentuk dummy.

# TORA

TORA

File EditGrid

TRANSPORTATION MODEL

Problem Title:

No. of Sources

No. of Dest'ns

Editing Grid:  
>>To DELETE, INSERT, COPY, or PASTE a column(row), click heading cell of target column(row), then invoke pull-down EditGrid menu  
>>For INSERT mode, a single(double) click of target row/column will place new row/column after(before) target row/column.

INPUT GRID - TRANSPORTATION

		D1	D2	D3	Supply
S/D Name		Chicago	St. Louis	Cincinnati	
S1	Kansas City	6,00	8,00	10,00	150
S2	Omaha	7,00	11,00	11,00	175
S3	Des Moines	4,00	5,00	12,00	275
Demand		200	100	300	

SOLVE Menu MAIN Menu Exit TORA

22:38

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\yes.txt

TRANSPORTATION MODEL

TORA Optimization System, Windows-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Minggu, October 28, 2018 22:41

TRANSPORTATION TABLEAU - (North-West Corner Method)

Title: **pojokKiriAtas --(minimum cost)**

Steps for generating transportation tableaux:  
1. (Optional step) Initialize ONE of the simplex multiplier (u1, u2, ..., v1, v2 ...) to zero value (default u1 = 0)  
2. Click (in any order) the cells defining the change-of-basis loop (if corret, cell changes color)  
3. Click command button NEXT ITERATION (or ALL ITERATIONS) -- This step may be executed without Step 2

Initialize u or v  
u1=0

Next Iteration All Iterations Write to Printer

Iter 1	ObjVal =	5925,00	D1	D2	D3	Supply
	Name		Chicago	St. Louis	Cincinnati	
			v1=6,00	v2=10,00	v3=10,00	
S1	Kansas City	u1=0,00	6,00 150	8,00	10,00	150
			0,00	2,00	0,00	
S2	Omaha	u2=1,00	7,00 50	11,00 100	11,00 25	175
			0,00	0,00	0,00	
S3	Des Moines	u3=2,00	4,00	5,00	12,00	
					275	
			4,00	7,00	0,00	
	Demand		200	100	300	

View/Modify Input Data

MAIN Menu

Exit TORA

22:41

# Sourcecode

```
File Edit Selection Find View Goto Tools
pojok.py x
1 cost=[
2     [6,8,10],
3     [7,11,11],
4     [4,5,12]
5 ]
6 supply=[150,175,275]
7 demand=[200,100,300]
8
9 def pojokKiriAtas(s,d):
10     table=[]
11     for x in range(s):
12         temp=[]
13         for y in range(d):
14             temp.append(0)
15         table.append(temp)
16     toAllocate(table,0,0)
17     return table
18
19 def toAllocate(table2,x,y):
20     global sTot,dTot
21     if sTot==0 and dTot==0:
22         return table2
23     if supply[x]<demand[y]:
24         table2[x][y]=supply[x]
25         demand[y]-=supply[x]
26         supply[x]=0
27         sTot=sum(supply)
28         dTot=sum(demand)
29         toAllocate(table2,x+1,y)
```

```
File Edit Selection Find View Goto Tools P
pojok.py x
30     elif supply[x]>demand[y]:
31         table2[x][y]=demand[y]
32         supply[x]-=demand[y]
33         demand[y]=0
34         sTot=sum(supply)
35         dTot=sum(demand)
36         toAllocate(table2,x,y+1)
37     elif supply[x]==demand[y]:
38         table2[x][y]=supply[x]
39         supply[x]=0
40         demand[y]=0
41         sTot=sum(supply)
42         dTot=sum(demand)
43         toAllocate(table2,x+1,y+1)
44
45     sTot=sum(supply)
46     dTot=sum(demand)
47     s=len(supply)
48     d=len(demand)
49     if sTot<dTot:
50         s+=1
51         supply.append(dTot - sTot)
52         sTot=sum(supply)
53         temp=[]
54         for x in range(len(demand)):
55             temp.append(0)
56         cost.append(temp)
57
58     elif sTot>dTot:
```

```
File Edit Selection Find View Goto Tools Project Pref
pojok.py x
59     d+=1
60     demand.append(sTot - dTot)
61     dTot=sum(demand)
62     for x in range(len(supply)):
63         cost[x].append(0)
64
65     resultTable=pojokKiriAtas(s,d)
66     print("Cost Tableau:")
67     for row in cost:
68         print(row)
69     print("\nAllocation Tableau:")
70     for row in resultTable:
71         print(row)
72     print("\nOngkos Minimum:")
73     ongkos=0
74     flag=0
75     for x in range(s):
76         for y in range(d):
77             if resultTable[x][y]==0:
78                 continue
79             if flag==1:
80                 print(" + ", end='')
81             print("%d"%(resultTable[x][y]),end='')
82             print("%d"%(cost[x][y]),end='')
83             if x<s-1 or y<d-1:
84                 flag=1
85             ongkos+=resultTable[x][y]*cost[x][y]
86     print(" =",ongkos)
```

# Studi Kasus 2

Tempat peleburan baja yang ada di tiga kota memproduksi sejumlah baja sebagai berikut :

<b>Lokasi</b>	<b>Jumlah yang ditawarkan per minggu (ton)</b>
A. Bethlehem	150
B. Birmingham	210
C. Gary	320



Ketiga tempat peleburan memasok baja ke empat kota dimana pabrik-pabriknya mempunyai permintaan sebagai berikut :

<b>Lokasi</b>	<b>Jumlah yang diminta per minggu (ton)</b>
1. Detroit	130
2. St. Louis	70
3. Chicago	180
4. Nortfolk	240

Biaya pengiriman per-ton baja adalah sebagai berikut

Dari	Ke			
	1	2	3	4
A	14	9	16	18
B	11	8	7	6
C	16	12	10	22

# Hasil Output

```
C:\Windows\System32\cmd.exe

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UT5>py pojok.py
Cost Tableau:
[14, 9, 16, 18, 0]
[11, 8, 7, 6, 0]
[16, 12, 10, 22, 0]

Allocation Tableau:
[130, 20, 0, 0, 0]
[0, 50, 160, 0, 0]
[0, 0, 20, 240, 60]

Ongkos Minimum:
130x14 + 20x9 + 50x8 + 160x7 + 20x10 + 240x22 + 60x0 = 9000

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UT5>
```

Karena total nilai supply tidak sama dengan demand, yaitu supply > demand, maka terbentuk dummy baru untuk mewakili demand yang kurang

# TORA

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\po2.txt

File EditGrid

**TRANSPORTATION MODEL**

Problem Title:

No. of Sources:

No. of Dest'ns:

Editing Grid:  
>>To DELETE, INSERT, COPY, or PASTE a column(row), click heading cell of target column(row), then invoke pull-down EditGrid menu  
>>For INSERT mode, a single(double) click of target row/column will place new row/column after(before) target row/column.

**INPUT GRID - TRANSPORTATION**

		D1	D2	D3	D4	D5	Supply
	S/D Name	Detroit	St. Louis	Chicago	Norfolk	Dummy	
\$1	Bethlehem	14,00	9,00	16,00	18,00	0,00	150
\$2	Birmingham	11,00	8,00	7,00	6,00	0,00	210
\$3	Gary	16,00	12,00	10,00	22,00	0,00	320
Demand		130	70	180	240	60	

SOLVE Menu MAIN Menu Exit TORA

23:07

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\po2.txt

TRANSPORTATION MODEL

TORA Optimization System, Windows-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Minggu, October 28, 2018 23:07

TRANSPORTATION TABLEAU - (North-West Corner Method)

Title: pojokKiriAtas --(minimum cost)

Steps for generating transportation tableaus:  
1. (Optional step) Initialize ONE of the simplex multiplier (u1, u2, ..., v1, v2 ...) to zero value (default u1 = 0)  
2. Click (in any order) the cells defining the change-of-basis loop (if corret, cell changes color)  
3. Click command button NEXT ITERATION (or ALL ITERATIONS) -- This step may be executed without Step 2

Initialize u or v  
u1=0

Next Iteration All Iterations Write to Printer

Iter 1	ObjVal =	9000,00	D1	D2	D3	D4	D5	Supply
	Name		Detroit	St. Louis	Chicago	Norfolk	Dummy	
S1	Bethlehem	u1=0,00	v1=14,00	v2=9,00	v3=8,00	v4=20,00	v5=-2,00	150
			14,00	9,00	16,00	18,00	0,00	
			130	20				
S2	Birmingham	u2=-1,00	0,00	0,00	-8,00	2,00	-2,00	210
			11,00	8,00	7,00	6,00	0,00	
				50	160			
S3	Gary	u3=2,00	2,00	0,00	0,00	13,00	-3,00	320
			16,00	12,00	10,00	22,00	0,00	
					20	240	60	
	Demand		0,00	-1,00	0,00	0,00	0,00	
			130	70	180	240	60	

View/Modify Input Data

MAIN Menu

Exit TORA

23:07

# Sourcecode

```
File Edit Selection Find View Goto Tools
vogel.py x pojok.py
1 cost=[
2     [14,9,16,18],
3     [11,8,7,6],
4     [16,12,10,22]
5 ]
6 supply=[150,210,320]
7 demand=[130,70,180,240]
8
9 def pojokKiriAtas(s,d):
10     table=[]
11     for x in range(s):
12         temp=[]
13         for y in range(d):
14             temp.append(0)
15         table.append(temp)
16     toAllocate(table,0,0)
17     return table
18
19 def toAllocate(table2,x,y):
20     global sTot,dTot
21     if sTot==0 and dTot==0:
22         return table2
23     if supply[x]<demand[y]:
24         table2[x][y]=supply[x]
25         demand[y]-=supply[x]
26         supply[x]=0
27         sTot=sum(supply)
28         dTot=sum(demand)
29         toAllocate(table2,x+1,y)
```

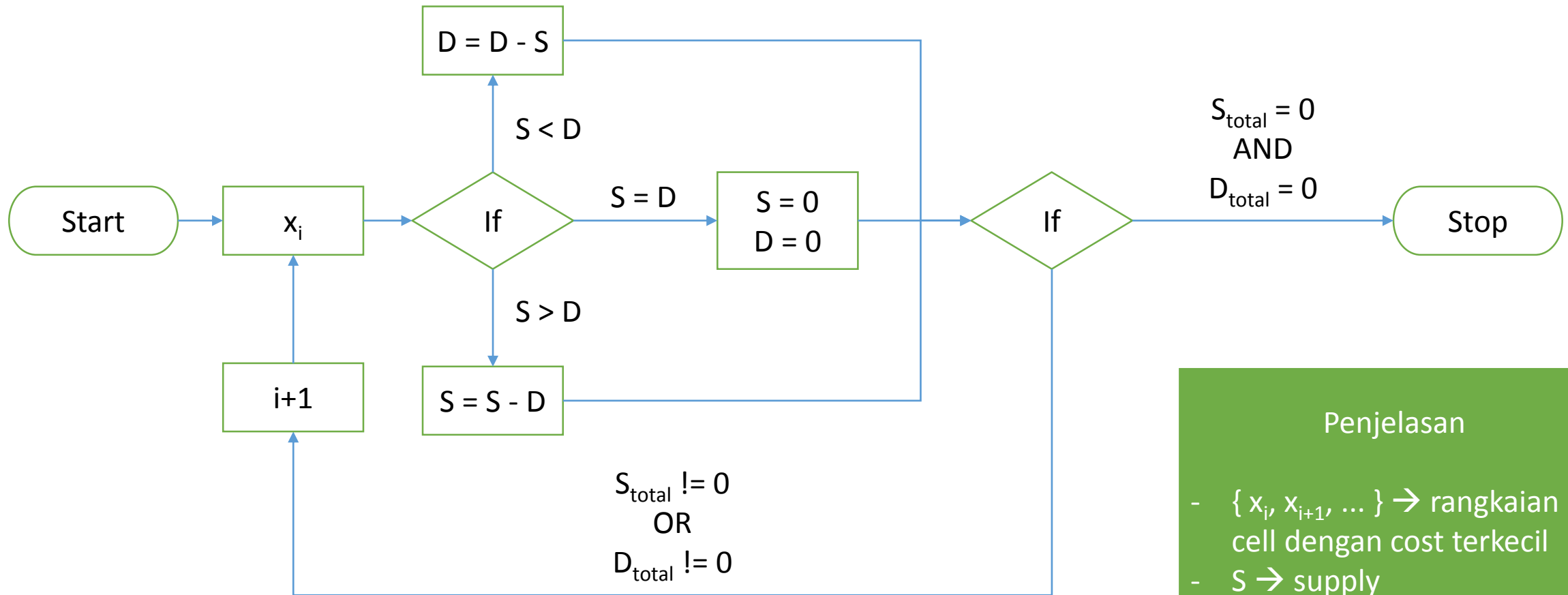
```
File Edit Selection Find View Goto Tools
pojok.py x
30 elif supply[x]>demand[y]:
31     table2[x][y]=demand[y]
32     supply[x]-=demand[y]
33     demand[y]=0
34     sTot=sum(supply)
35     dTot=sum(demand)
36     toAllocate(table2,x,y+1)
37 elif supply[x]==demand[y]:
38     table2[x][y]=supply[x]
39     supply[x]=0
40     demand[y]=0
41     sTot=sum(supply)
42     dTot=sum(demand)
43     toAllocate(table2,x+1,y+1)
44
45 sTot=sum(supply)
46 dTot=sum(demand)
47 s=len(supply)
48 d=len(demand)
49 if sTot<dTot:
50     s+=1
51     supply.append(dTot - sTot)
52     sTot=sum(supply)
53     temp=[]
54     for x in range(len(demand)):
55         temp.append(0)
56     cost.append(temp)
57
58 elif sTot>dTot:
```

```
File Edit Selection Find View Goto Tools Project Pref
pojok.py x
59     d+=1
60     demand.append(sTot - dTot)
61     dTot=sum(demand)
62     for x in range(len(supply)):
63         cost[x].append(0)
64
65 resultTable=pojokKiriAtas(s,d)
66 print("Cost Tableau:")
67 for row in cost:
68     print(row)
69 print("\nAllocation Tableau:")
70 for row in resultTable:
71     print(row)
72 print("\nOngkos Minimum:")
73 ongkos=0
74 flag=0
75 for x in range(s):
76     for y in range(d):
77         if resultTable[x][y]==0:
78             continue
79         if flag==1:
80             print(" + ", end='')
81         print("%d"%(resultTable[x][y]),end='')
82         print("x%d"%(cost[x][y]),end='')
83         if x<s-1 or y<d-1:
84             flag=1
85         ongkos+=resultTable[x][y]*cost[x][y]
86 print("=",ongkos)
```

# METODE ONGKOS TERKECIL

Pada metode ini, dilakukan assign nilai dimulai dari cell yang mempunyai nilai cost terkecil. Apabila terdapat cost yang sama, maka dipilih cell dengan nilai supply atau demand terbesar. Setelah itu, lakukan proses sebelumnya pada cost terkecil berikutnya, kemudian ulangi hingga total nilai supply maupun demand habis.

# Diagram Alur



## Penjelasan

- $\{x_i, x_{i+1}, \dots\} \rightarrow$  rangkaian cell dengan cost terkecil
- $S \rightarrow$  supply
- $D \rightarrow$  demand



# Pseudocode

```
1 sTot=sum(supply)
2 dTot=sum(demand)
3 s=len(supply)
4 d=len(demand)
5 if sTot<dTot:
6     s+=1
7     supply.append(dTot - sTot)
8     sTot=sum(supply)
9     temp=[]
10    for x in range(len(demand)):
11        temp.append(0)
12    cost.append(temp)
13
14 elif sTot>dTot:
15     d+=1
16     demand.append(sTot - dTot)
17     dTot=sum(demand)
18     for x in range(len(supply)):
19         cost[x].append(0)
20
21 terkecil=sortCost()
```

Jumlah total seluruh supply dan demand

Banyak supply dan demand

Jika total supply < demand (unbalanced), maka program akan otomatis menambah aktor supply baru yaitu dummy, dengan nilai cost 0, dan nilai supply sebanyak selisih total demand dan supply

Jika total supply > demand (unbalanced), maka program akan otomatis menambah aktor demand baru yaitu dummy, dengan nilai cost 0, dan nilai demand sebanyak selisih total supply dan demand

Wadah untuk menyimpan cell dengan cost terkecil

```
1  def sortCost():
2      indexCost=[]
3      for x in range(s):
4          for y in range(d):
5              temp=[]
6              temp.insert(0,cost[x][y])
7              temp.insert(1,x)
8              temp.insert(2,y)
9              indexCost.append(temp)
10     indexCost=sorted(indexCost)
11     return indexCost
```

Atribut-atribut tiap sel mulanya disimpan pada temp, antara lain atribut nilai cost, nilai x,y (posisi). Kemudian, object temp akan disimpan pada indexCost.

Mengurutkan object-object yang terdapat dalam indexCost berdasarkan nilai cost terkecil.

```
1 def ongkosMinimum(s,d):  
2     table=[]  
3     for x in range(s):  
4         temp=[]  
5         for y in range(d):  
6             temp.append(0)  
7         table.append(temp)  
8     toAllocate(table,0)  
9     return table
```

Dibentuk table baru untuk menyimpan nilai pengalokasian ketika proses

Dengan ukuran baris sebanyak jumlah supply dan kolom sesuai jumlah demand, kita alokasikan nilai 0 sebagai nilai mula-mula

Memanggil fungsi pengalokasian dengan parameter 0 yang menandakan indeks pertama pada array ongkos terkecil

```

1  def toAllocate(table2,i):
2      global sTot,dTot
3      if sTot==0 and dTot==0:
4          return table2
5      x=terkecil[i][1]
6      y=terkecil[i][2]
7      if supply[x]<demand[y]:
8          table2[x][y]=supply[x]
9          demand[y]-=supply[x]
10         supply[x]=0
11         sTot=sum(supply)
12         dTot=sum(demand)
13         toAllocate(table2,i+1)
14     elif supply[x]>demand[y]:
15         table2[x][y]=demand[y]
16         supply[x]-=demand[y]
17         demand[y]=0
18         sTot=sum(supply)
19         dTot=sum(demand)
20         toAllocate(table2,i+1)
21     elif supply[x]==demand[y]:
22         table2[x][y]=supply[x]
23         supply[x]=0
24         demand[y]=0
25         sTot=sum(supply)
26         dTot=sum(demand)
27         toAllocate(table2,i+1)

```

Fungsi pengalokasian menggunakan recursive yang akan berhenti ketika total supply dan demand mencapai 0 (habis). i mengindikasikan urutan pertama dengan cost terkecil. Fungsi akan dipanggil dengan  $i = 0$  (index awal ongkos minimum).

Mendapatkan nilai posisi x dan y dari tabel array terkecil (berisi data-data hasil dari sortCost)

Apabila nilai corresponding supply < demand, maka nilai yang dialokasikan pada  $C_{x,y}$  adalah nilai supply nya. Kemudian, kita kurangi nilai demand nya sebanyak nilai supply, lalu set nilai supply menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu recursive dengan  $i+1$  untuk cost terkecil selanjutnya.

Apabila nilai corresponding supply > demand, maka nilai yang dialokasikan pada  $C_{x,y}$  adalah nilai demand nya. Kemudian, kita kurangi nilai supply nya sebanyak nilai demand, lalu set nilai demand menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu recursive dengan  $i+1$  untuk cost terkecil selanjutnya.

Apabila nilai corresponding supply = demand, maka nilai yang dialokasikan pada  $C_{x,y}$  adalah nilai salah satunya. Kemudian, kita set nilai demand dan supply nya menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu recursive dengan  $i+1$  untuk cost terkecil selanjutnya.

# Studi Kasus 1

Gandum dipanen di Midwest (daerah pertanian Amerika bagian Tengah Barat) dan disimpan dalam cerobong butir gandum di tiga kota – Kansas City, Omaha, dan Des Moines. Ketiga cerobong butir gandum ini memasok tiga penggilingan tepung yang berlokasi di Chicago, St. Louis, dan Cincinnati. Butir-butir gandum tersebut dikirim ke penggilingan dengan menggunakan gerbong kereta api, yang tiap gerbongnya memuat satu ton gandum. Setiap bulannya, tiap cerobong butir gandum dapat memasok penggilingan sejumlah ton gandum berikut ini.

<b>Cerobong Butir Gandum</b>	<b>Jumlah yang ditawarkan</b>
1. Kansas City	150
2. Omaha	175
3. Des Moines	275

Jumlah ton gandum yang diminta per bulan dari tiap penggilingan adalah sebagai berikut :

<b>Penggilingan</b>	<b>Jumlah yang diminta</b>
A. Chicago	200
B. St. Louis	100
C. Cincinnati	300

Biaya pengiriman (\$) :

Cerobong Butir Gandum	Penggilingan		
	Chicago (A)	St. Louis (B)	Cincinnati (C)
Kansas City	6	8	10
Omaha	7	11	11
Des Moines	4	5	12

Untuk menentukan berapa banyak ton gandum yang harus dikirim dari tiap cerobong butir gandum ke tiap penggilingan setiap bulannya agar total biaya transportasi minimum

<div>Ke</div> <div>Dari</div>	A	B	C	Pasokan
1	6	8	10	150
2	7	11	11	175
3	4	5	12	275
Permintaan	200	100	300	600



# Hasil Output

```
C:\Windows\System32\cmd.exe

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS>py leastcost.py
Cost Tableau:
[6, 8, 10]
[7, 11, 11]
[4, 5, 12]

Allocation Tableau:
[0, 25, 125]
[0, 0, 175]
[200, 75, 0]

Ongkos Minimum:
25x8 + 125x10 + 175x11 + 200x4 + 75x5 = 4550

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS>
```

Table cost tidak berubah  
karena total nilai supply =  
demand, sehingga tidak  
terbentuk dummy.

# TORA

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\yes.txt

File EditGrid

**TRANSPORTATION MODEL**

Problem Title:

No. of Sources

No. of Dest'ns

Editing Grid:  
>>To DELETE, INSERT, COPY, or PASTE a column(row), click heading cell of target column(row), then invoke pull-down EditGrid menu  
>>For INSERT mode, a single(double) click of target row/column will place new row/column after(before) target row/column.

**INPUT GRID - TRANSPORTATION**

		D1	D2	D3	Supply
	S/D Name	Chicago	St. Louis	Cincinnati	
S1	Kansas City	6,00	8,00	10,00	150
S2	Omaha	7,00	11,00	11,00	175
S3	Des Moines	4,00	5,00	12,00	275
Demand		200	100	300	

SOLVE Menu MAIN Menu Exit TORA

22:43

TORA Optimization System, Windows®-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Minggu, Oktober 26, 2018 22:49

**Title: ongkosMinimum --(minimum cost)**

**Steps for generating transportation tableaux:**

1. (Optional step) Initialize ONE of the simplex multiplier ( $u_1, u_2, \dots, v_1, v_2 \dots$ ) to zero value (default  $u_1 = 0$ )
2. Click (in any order) the cells defining the change-of-basis loop (if correct, cell changes color)
3. Click command button NEXT ITERATION (or ALL ITERATIONS) – This step may be executed without Step 2

- Initialize  $u$  or  $v$

u1=0

Next Iteration

All Iterations

Write to Printer

Iter 1	ObjVal =	4550.00	D1	D2	D3	Supply
	Name		Chicago	St. Louis	Cincinnati	
S1	Kansas City	u1=0.00	v1=7.00	v2=8.00	v3=10.00	150
			6.00	8.00	10.00	
			1.00	0.00	0.00	
S2	Omaha	u2=1.00	7.00	11.00	11.00	175
			1.00	-2.00	0.00	
			4.00	5.00	12.00	
S3	Des Moines	u3=-3.00	200	75		275
			0.00	0.00	-5.00	
			200	100	300	

View/Modify Input Data

MAIN Menu

Exit TORA

# Sourcecode

```
File Edit Selection Find View Goto
leastcost.py x
1 cost=[
2     [6,8,10],
3     [7,11,11],
4     [4,5,12]
5 ]
6 supply=[150,175,275]
7 demand=[200,100,300]
8
9 def ongkosMinimum(s,d):
10     table=[]
11     for x in range(s):
12         temp=[]
13         for y in range(d):
14             temp.append(0)
15         table.append(temp)
16     toAllocate(table,0)
17     return table
18
19 def toAllocate(table2,i):
20     global sTot,dTot
21     if sTot==0 and dTot==0:
22         return table2
23     x=terkecil[i][1]
24     y=terkecil[i][2]
25     if supply[x]<demand[y]:
26         table2[x][y]=supply[x]
27         demand[y]-=supply[x]
28         supply[x]=0
29         sTot=sum(supply)
```

```
File Edit Selection Find View Goto Tools
leastcost.py x
30     dTot=sum(demand)
31     toAllocate(table2,i+1)
32     elif supply[x]>demand[y]:
33         table2[x][y]=demand[y]
34         supply[x]-=demand[y]
35         demand[y]=0
36         sTot=sum(supply)
37         dTot=sum(demand)
38         toAllocate(table2,i+1)
39     elif supply[x]==demand[y]:
40         table2[x][y]=supply[x]
41         supply[x]=0
42         demand[y]=0
43         sTot=sum(supply)
44         dTot=sum(demand)
45         toAllocate(table2,i+1)
46
47 def sortCost():
48     indexCost=[]
49     for x in range(s):
50         for y in range(d):
51             temp=[]
52             temp.insert(0,cost[x][y])
53             temp.insert(1,x)
54             temp.insert(2,y)
55             indexCost.append(temp)
56     indexCost=sorted(indexCost)
57     return indexCost
58
```

```
File Edit Selection Find View Goto Tools
leastcost.py x
59 sTot=sum(supply)
60 dTot=sum(demand)
61 s=len(supply)
62 d=len(demand)
63 if sTot<dTot:
64     s+=1
65     supply.append(dTot - sTot)
66     sTot=sum(supply)
67     temp=[]
68     for x in range(len(demand)):
69         temp.append(0)
70     cost.append(temp)
71
72 elif sTot>dTot:
73     d+=1
74     demand.append(sTot - dTot)
75     dTot=sum(demand)
76     for x in range(len(supply)):
77         cost[x].append(0)
78     terkecil=sortCost()
79     resultTable=ongkosMinimum(s,d)
80     print("Cost Tableau:")
81     for row in cost:
82         print(row)
83     print("\nAllocation Tableau:")
84     for row in resultTable:
85         print(row)
86     print("\nOngkos Minimum:")
87     ongkos=0
```

```
File Edit Selection Find View Goto Tools Project
leastcost.py x
88 flag=0
89 for x in range(s):
90     for y in range(d):
91         if resultTable[x][y]==0:
92             continue
93         if flag==1:
94             print(" + ", end='')
95         print("%d"%(resultTable[x][y]),end='')
96         print("%d"%(cost[x][y]),end='')
97         if x<s-1 or y<d-1:
98             flag=1
99         ongkos+=resultTable[x][y]*cost[x][y]
100 print("=",ongkos)
```

# Studi Kasus 2

Tempat peleburan baja yang ada di tiga kota memproduksi sejumlah baja sebagai berikut :

<b>Lokasi</b>	<b>Jumlah yang ditawarkan per minggu (ton)</b>
A. Bethlehem	150
B. Birmingham	210
C. Gary	320

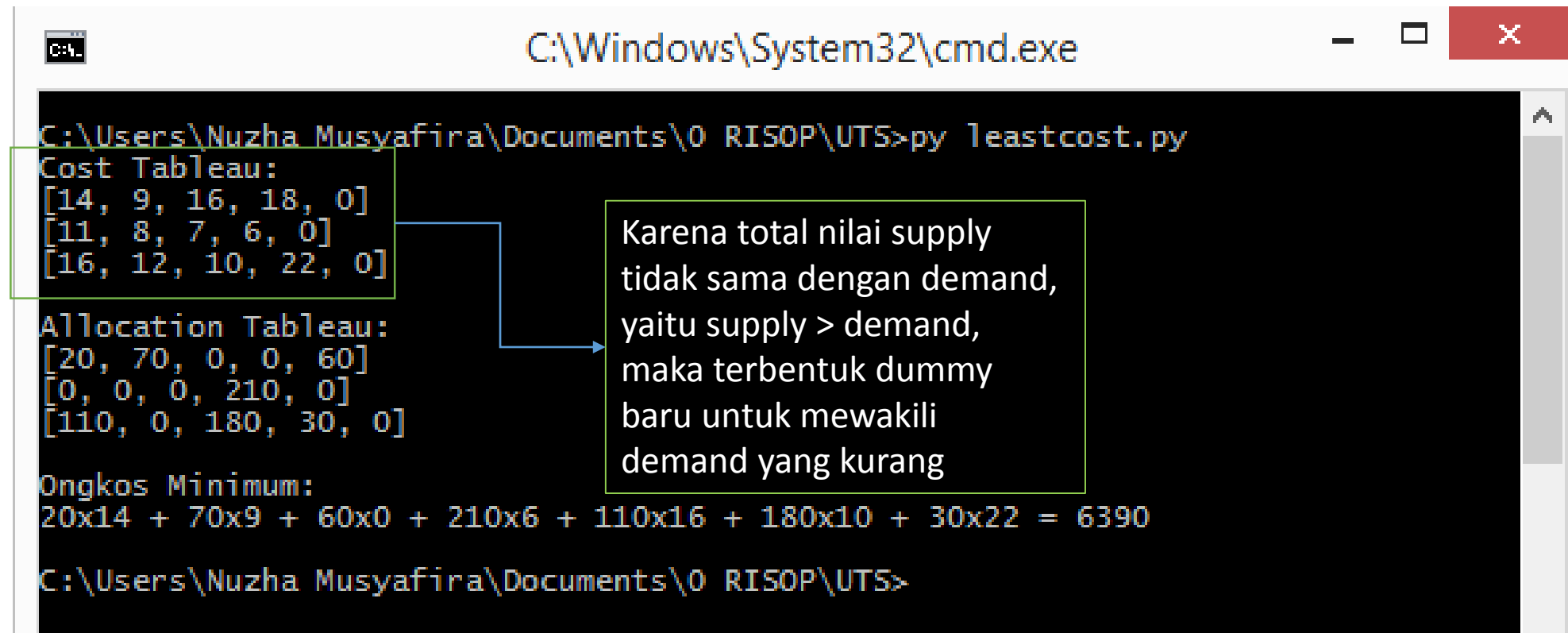
Ketiga tempat peleburan memasok baja ke empat kota dimana pabrik-pabriknya mempunyai permintaan sebagai berikut :

<b>Lokasi</b>	<b>Jumlah yang diminta per minggu (ton)</b>
1. Detroit	130
2. St. Louis	70
3. Chicago	180
4. Nortfolk	240

Biaya pengiriman per-ton baja adalah sebagai berikut

Dari	Ke			
	1	2	3	4
A	14	9	16	18
B	11	8	7	6
C	16	12	10	22

# Hasil Output



```
C:\Windows\System32\cmd.exe

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS>py leastcost.py
Cost Tableau:
[14, 9, 16, 18, 0]
[11, 8, 7, 6, 0]
[16, 12, 10, 22, 0]

Allocation Tableau:
[20, 70, 0, 0, 60]
[0, 0, 0, 210, 0]
[110, 0, 180, 30, 0]

Ongkos Minimum:
20x14 + 70x9 + 60x0 + 210x6 + 110x16 + 180x10 + 30x22 = 6390

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS>
```

Karena total nilai supply tidak sama dengan demand, yaitu supply > demand, maka terbentuk dummy baru untuk mewakili demand yang kurang



# TORA

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\po2.txt

File EditGrid

**TRANSPORTATION MODEL**

Problem Title:

No. of Sources

No. of Dest'ns

Editing Grid:  
>>To DELETE, INSERT, COPY, or PASTE a column(row), click heading cell of target column(row), then invoke pull-down EditGrid menu  
>>For INSERT mode, a single(double) click of target row/column will place new row/column after(before) target row/column.

**INPUT GRID - TRANSPORTATION**

		D1	D2	D3	D4	D5	Supply
	S/D Name	Detroit	St. Louis	Chicago	Norfolk	Dummy	
\$1	Bethlehem	14,00	9,00	16,00	18,00	0,00	150
\$2	Birmingham	11,00	8,00	7,00	6,00	0,00	210
\$3	Gary	16,00	12,00	10,00	22,00	0,00	320
Demand		130	70	180	240	60	

SOLVE Menu MAIN Menu Exit TORA

23:08

TORA Optimization System, Windows®-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Minggu, Oktober 26, 2018 23:09

Title: ongkosMinimum --(minimum cost)

### Steps for generating transportation tableaus:

1. (Optional step) Initialize ONE of the simplex multiplier ( $u_1, u_2, \dots, v_1, v_2, \dots$ ) to zero value (default  $u_1 = 0$ )
2. Click (in any order) the cells defining the change-of-basis loop (if correct, cell changes color)
3. Click command button NEXT ITERATION (or ALL ITERATIONS) – This step may be executed without Step 2

- Initialize  $u$  or  $v$

u1=0

Next Iteration

All Iterations

Write to Printer

Iter 1	ObjVal =	6390.00	D1	D2	D3	D4	D5	Supply
	Name		Detroit	St. Louis	Chicago	Norfolk	Dummy	
S1	Bethlehem	u1=0.00	v1=14.00	v2=9.00	v3=8.00	v4=20.00	v5=0.00	150
			14.00	9.00	16.00	18.00	0.00	
			20	70			60	
S2	Birmingham	u2=-14.00	0.00	0.00	-8.00	2.00	0.00	210
			11.00	8.00	7.00	6.00	0.00	
			-11.00	-13.00	-13.00	0.00	-14.00	
S3	Gary	u3=2.00	16.00	12.00	10.00	22.00	0.00	320
			110		180	30		
			0.00	-1.00	0.00	0.00	2.00	
	Demand		130	70	180	240	60	

View/Modify Input Data

MAIN Menu

Exit TORA

# Sourcecode

```
File Edit Selection Find View Goto
vogel.py x
1 cost=[
2     [14,9,16,18],
3     [11,8,7,6],
4     [16,12,10,22]
5 ]
6 supply=[150,210,320]
7 demand=[130,70,180,240]
8
9 def ongkosMinimum(s,d):
10     table=[]
11     for x in range(s):
12         temp=[]
13         for y in range(d):
14             temp.append(0)
15         table.append(temp)
16     toAllocate(table,0)
17     return table
18
19 def toAllocate(table2,i):
20     global sTot,dTot
21     if sTot==0 and dTot==0:
22         return table2
23     x=terkecil[i][1]
24     y=terkecil[i][2]
25     if supply[x]<demand[y]:
26         table2[x][y]=supply[x]
27         demand[y]-=supply[x]
28         supply[x]=0
29         sTot=sum(supply)
```

```
File Edit Selection Find View Goto Tools
leastcost.py x
30     dTot=sum(demand)
31     toAllocate(table2,i+1)
32     elif supply[x]>demand[y]:
33         table2[x][y]=demand[y]
34         supply[x]-=demand[y]
35         demand[y]=0
36         sTot=sum(supply)
37         dTot=sum(demand)
38         toAllocate(table2,i+1)
39     elif supply[x]==demand[y]:
40         table2[x][y]=supply[x]
41         supply[x]=0
42         demand[y]=0
43         sTot=sum(supply)
44         dTot=sum(demand)
45         toAllocate(table2,i+1)
46
47 def sortCost():
48     indexCost=[]
49     for x in range(s):
50         for y in range(d):
51             temp=[]
52             temp.insert(0,cost[x][y])
53             temp.insert(1,x)
54             temp.insert(2,y)
55             indexCost.append(temp)
56     indexCost=sorted(indexCost)
57     return indexCost
58
```

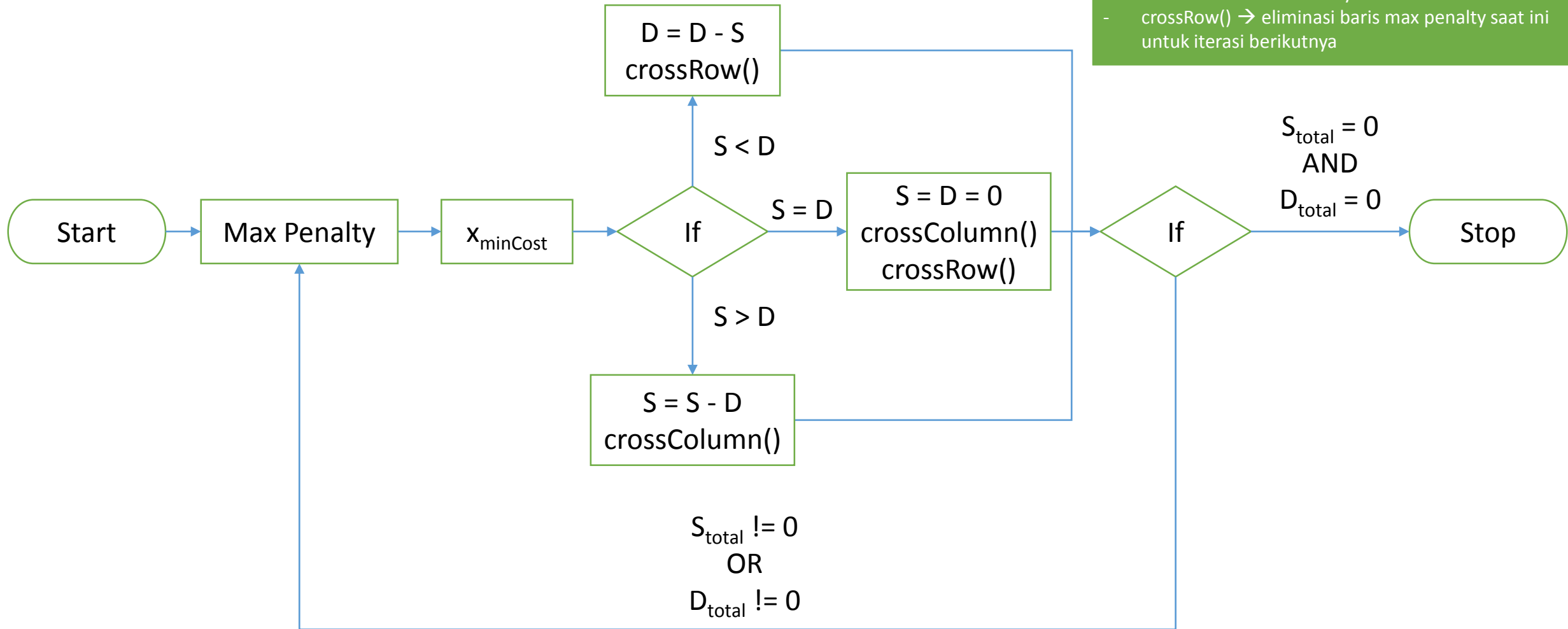
```
File Edit Selection Find View Goto Tools
leastcost.py x
59 sTot=sum(supply)
60 dTot=sum(demand)
61 s=len(supply)
62 d=len(demand)
63 if sTot<dTot:
64     s+=1
65     supply.append(dTot - sTot)
66     sTot=sum(supply)
67     temp=[]
68     for x in range(len(demand)):
69         temp.append(0)
70     cost.append(temp)
71
72 elif sTot>dTot:
73     d+=1
74     demand.append(sTot - dTot)
75     dTot=sum(demand)
76     for x in range(len(supply)):
77         cost[x].append(0)
78     terkecil=sortCost()
79     resultTable=ongkosMinimum(s,d)
80     print("Cost Tableau:")
81     for row in cost:
82         print(row)
83     print("\nAllocation Tableau:")
84     for row in resultTable:
85         print(row)
86     print("\nOngkos Minimum:")
87     ongkos=0
```

```
File Edit Selection Find View Goto Tools Project
leastcost.py x
88 flag=0
89 for x in range(s):
90     for y in range(d):
91         if resultTable[x][y]==0:
92             continue
93         if flag==1:
94             print(" + ", end='')
95         print("%d"%(resultTable[x][y]),end='')
96         print("x%d"%(cost[x][y]),end='')
97         if x<s-1 or y<d-1:
98             flag=1
99         ongkos+=resultTable[x][y]*cost[x][y]
100 print(" =",ongkos)
```

# METODE VOGEL

Metode ini merupakan cara terbaik untuk menemukan solusi basis awal. Tetapi langkah-langkah pengerjaannya cukup rumit. Pertama-tama tentukan penalty row dan penalty col dengan cara mengurangkan 2 cell ongkos terkecil dari tiap-tiap baris dan kolom. Kemudian dari seluruh penalty baris dan penalty kolom, diambil baris atau kolom yang nilai penalty-nya terbesar. Assign nilai sebesar mungkin pada cell dengan nilai cost terkecil. Kemudian lakukan kembali penalty baris dan penalty kolom, tetapi baris atau kolom yang telah dipilih pada penalty sebelumnya tidak diikuti dalam penalty ini. Ulangi proses sebelumnya hingga total supply maupun demand habis.

# Diagram Alur



## Penjelasan

- $x_{\min\text{cost}}$  → cell dengan cost terkecil pada baris/kolom dengan max penalty
- $S$  → supply
- $D$  → demand
- $\text{crossColumn()}$  → eliminasi kolom max penalty saat ini untuk iterasi berikutnya
- $\text{crossRow()}$  → eliminasi baris max penalty saat ini untuk iterasi berikutnya

# Pseudocode

```
1 sTot=sum(supply)
2 dTot=sum(demand)
3 s=len(supply)
4 d=len(demand)
5 if sTot<dTot:
6     s+=1
7     supply.append(dTot - sTot)
8     sTot=sum(supply)
9     temp=[]
10    for x in range(len(demand)):
11        temp.append(0)
12    cost.append(temp)
13
14 elif sTot>dTot:
15     d+=1
16     demand.append(sTot - dTot)
17     dTot=sum(demand)
18     for x in range(len(supply)):
19         cost[x].append(0)
20
21 import copy
22 cost2=copy.deepcopy(cost)
```

Jumlah total seluruh supply dan demand

Banyak supply dan demand

Jika total supply < demand (unbalanced), maka program akan otomatis menambah aktor supply baru yaitu dummy, dengan nilai cost 0, dan nilai supply sebanyak selisih total demand dan supply

Jika total supply > demand (unbalanced), maka program akan otomatis menambah aktor demand baru yaitu dummy, dengan nilai cost 0, dan nilai demand sebanyak selisih total supply dan demand

Karena nilai cost nantinya akan berubah-ubah selama rangkaian iterasi, maka dilakukan backup untuk nilai cost mula-mula pada cost2

```
1 def vogel(s,d):
2     table=[]
3     for x in range(s):
4         temp=[]
5         for y in range(d):
6             temp.append(0)
7         table.append(temp)
8     toAllocate(table,s,d)
9     return table
```

Dibentuk table baru untuk menyimpan nilai pengalokasian ketika proses

Dengan ukuran baris sebanyak jumlah supply dan kolom sesuai jumlah demand, kita alokasikan nilai 0 sebagai nilai mula-mula

Memanggil fungsi pengalokasian dengan parameter s,d yang menandakan banyak supply dan demand mula-mula

```
1  def sortCost(d,s,flag):
2      box=[]
3      for x in range(d):
4          temp=[]
5          for y in range(s):
6              if flag==1:
7                  temp.append(cost[y][x])
8              else:
9                  temp.append(cost[x][y])
10         temp2=sorted(temp)
11         box.append(temp2)
12     return box
```

Nilai cost tiap sel mulanya disimpan pada temp. flag merupakan penanda untuk baris dan kolom. Apabila flag = 1, maka kita sedang mencari nilai cost terkecil dari setiap kolom. Sebaliknya apabila flag = 0, maka dari setiap baris.

Setelah nilai-nilai cost didapatkan berdasarkan baris atau kolom, barulah diurutkan dari yang terkecil kemudian diassign ke temp2. Object pada temp2 lalu diappend oleh box yang menyimpan secara komunal.



```
1 def penaltyVal(Lists):
2     penBox=[]
3     x=0
4     for row in Lists:
5         if(len(row)==1):
6             penalty=row[0]
7         else:
8             penalty=row[1]-row[0]
9         temp=[]
10        temp.insert(0,penalty)
11        temp.insert(1,row[0])
12        temp.insert(2,x)
13        penBox.append(temp)
14        x+=1
15    return penBox
```

Fungsi untuk mendapatkan nilai penalty tiap baris atau kolom. Parameternya merupakan list dari kolom atau baris yang nilai costnya telah diurutkan dari yang terkecil. Untuk setiap baris atau kolom, nilai penalty didapatkan dari pengurangan indeks ke-1 dan 0, yaitu 2 nilai cost terkecil. Apabila hanya tersisa 1 nilai (indeks 0), maka otomatis diambil sebagai nilai penalty.

Setelah nilai penalty didapatkan, data-data seperti nilai penalty, nilai cost terkecil (row[0]), dan baris atau kolom ke berapa (x) disimpan pada temp sebagai object. Tujuan row[0] juga disimpan adalah agar nantinya apabila terdapat nilai penalty terkecil yang sama, maka akan diambil berdasarkan nilai cost terkecil. Indeks x untuk menandakan bahwa list yang sedang dieksekusi merupakan baris atau kolom ke berapa.

```
1 def sortAll(row,col):
2     box=[]
3     for x in row:
4         x+=[0]
5         box.append(x)
6     for x in col:
7         x+=[1]
8         box.append(x)
9     box.sort(key=Lambda z: (-z[0], z[1]))
10    return box
```

Fungsi untuk menyatukan data penalty yang tadinya dipisah berdasarkan baris dan kolom menjadi kesatuan object. Mula-mula, object pada row dan kolom diappend satu persatu pada box dengan tambahan elemen yaitu 0 atau 1. 0 menandakan bahwa object tersebut merupakan baris, sedangkan 1 kolom. Setelah semua menjadi satu kesatuan object, lalu nilai penalty diurutkan.  $-z[0]$  menandakan bahwa diurutkan dari yang terbesar berdasarkan elemen ke-0 (nilai penalty), kemudian apabila ada nilai yang sama, diurutkan berdasarkan  $z[1]$  atau elemen ke-1 (nilai cost, sebelumnya adalah  $row[0]$ ) dari yang terkecil.

```

1  def indexCell(d,s,flag):
2      box=[]
3      for x in range(d):
4          temp2=[]
5          for y in range(s):
6              temp=[]
7              if flag==1:
8                  temp.insert(0,cost[y][x])
9                  temp.insert(1,y)
10                 temp.insert(2,x)
11             else:
12                 temp.insert(0,cost[x][y])
13                 temp.insert(1,x)
14                 temp.insert(2,y)
15             temp2.append(temp)
16         temp2=sorted(temp2)
17         box.append(temp2)
18     return box

```

Merupakan fungsi untuk menyimpan posisi x,y dari setiap cell untuk setiap baris atau kolom. Data yang disimpan pada temp antara lain nilai cost cell, posisi x,y. Apabila flag = 1 maka mengindikasikan sedang menyimpan data untuk kelompok kolom, sedangkan 0 baris. Setelah setiap cell dieksekusi, object disimpan pada temp2 sebagai satu kesatuan per baris atau kolom.

Setelah temp2 menyimpan object cell dari 1 baris, cell-cell itu diurutkan berdasarkan nilai cost terkecil. temp2 kemudian diappend ke box sebagai kesatuan baris-baris.

```

1  def toAllocate(table2,s,d):
2      global sTot,dTot,itors
3      if sTot==0 and dTot==0:
4          return table2
5      sortedRow=sortCost(s,d,0)
6      sortedCol=sortCost(d,s,1)
7      pRow=penaltyVal(sortedRow)
8      pCol=penaltyVal(sortedCol)
9      sortedAll=sortAll(pRow,pCol)
10     indexRow=indexCell(s,d,0)
11     indexCol=indexCell(d,s,1)
12     direc=sortedAll[0][3]
13     if direc==0:
14         idx=sortedAll[0][2]
15         x=indexRow[idx][0][1]
16         y=indexRow[idx][0][2]
17     elif direc==1:
18         idx=sortedAll[0][2]
19         x=indexCol[idx][0][1]
20         y=indexCol[idx][0][2]
21     if supply[x]<demand[y]:
22         table2[x][y]=supply[x]
23         demand[y]-=supply[x]
24         supply[x]=0
25         sTot=sum(supply)
26         dTot=sum(demand)
27         for a in range(d):
28             cost[x][a]=1000000
29         toAllocate(table2,s,d)
30     elif supply[x]>demand[y]:
31         table2[x][y]=demand[y]
32         supply[x]-=demand[y]
33         demand[y]=0
34         sTot=sum(supply)
35         dTot=sum(demand)
36         for z in range(s):
37             cost[z][y]=1000000
38         toAllocate(table2,s,d)
39     elif supply[x]==demand[y]:
40         table2[x][y]=supply[x]
41         supply[x]=0
42         demand[y]=0
43         sTot=sum(supply)
44         dTot=sum(demand)
45         for a in range(d):
46             cost[x][a]=1000000
47         for z in range(s):
48             cost[z][y]=1000000
49         toAllocate(table2,s,d)

```

Fungsi pengalokasian menggunakan recursive yang akan berhenti ketika total supply dan demand mencapai 0 (habis). i mengindikasikan urutan pertama dengan cost terkecil. Fungsi akan dipanggil dengan s dan d mula-mula yaitu sebelum terjadi crossing kolom atau baris.

Beberapa proses yang harus dilakukan sebagai pendukung, antara lain sortedAll untuk mendapat nilai-nilai penalty terbesar, indexRow dan indexCol untuk mendapatkan posisi x,y tiap cell tiap baris/kolom diurutkan dari cost terkecil. sortedAll[0][3] menyimpan nilai 0 (baris) dan 1 (kolom) dari penalty terbesar (indeks 0), untuk menentukan apakah penalty terbesar berasal dari baris atau kolom.

Proses ini untuk menentukan, jika direc 1 berarti kolom, jika 0 berarti baris. Setelah itu, idx akan mendapatkan indeks kolom atau baris ke berapa melalui sortedAll[0][2]. Kemudian, nilai x dan y (posisi) cell didapat dari indexCol atau indexRow ke [0][1] dan [0][2] ([0] karena cost terkecil pasti di indeks 0).

Apabila nilai corresponding supply < demand, maka nilai yang dialokasikan pada Cx,y adalah nilai supply nya. Kemudian, kita kurangi nilai demand nya sebanyak nilai supply, lalu set nilai supply menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu untuk crossing (mencoret) seluruh baris ke-x dari tabel, kita dapat mengassign nilai yang sangat besar agar penalty nya kecil dan diabaikan, contoh: 1000000. Kemudian, recursive kembali.

Apabila nilai corresponding supply > demand, maka nilai yang dialokasikan pada Cx,y adalah nilai demand nya. Kemudian, kita kurangi nilai supply nya sebanyak nilai demand, lalu set nilai demand menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu crossing seluruh kolom ke-y dari tabel dengan mengassign 1000000. Kemudian, recursive kembali.

Apabila nilai corresponding supply = demand, maka nilai yang dialokasikan pada Cx,y adalah nilai salah satunya. Kemudian, kita set nilai demand dan supply nya menjadi 0. Kita kalkulasikan ulang jumlah total supply dan demand, lalu crossing seluruh kolom ke-y dan baris ke-x dari tabel dengan mengassign 1000000. Kemudian, recursive kembali.

# Studi Kasus 1

Gandum dipanen di Midwest (daerah pertanian Amerika bagian Tengah Barat) dan disimpan dalam cerobong butir gandum di tiga kota – Kansas City, Omaha, dan Des Moines. Ketiga cerobong butir gandum ini memasok tiga penggilingan tepung yang berlokasi di Chicago, St. Louis, dan Cincinnati. Butir-butir gandum tersebut dikirim ke penggilingan dengan menggunakan gerbong kereta api, yang tiap gerbongnya memuat satu ton gandum. Setiap bulannya, tiap cerobong butir gandum dapat memasok penggilingan sejumlah ton gandum berikut ini.

## **Cerobong Butir Gandum**

1. Kansas City
2. Omaha
3. Des Moines

## **Jumlah yang ditawarkan**

150  
175  
275

Jumlah ton gandum yang diminta per bulan dari tiap penggilingan adalah sebagai berikut :

<b>Penggilingan</b>	<b>Jumlah yang diminta</b>
A. Chicago	200
B. St. Louis	100
C. Cincinnati	300

Biaya pengiriman (\$) :

Cerobong Butir Gandum	Penggilingan		
	Chicago (A)	St. Louis (B)	Cincinnati (C)
Kansas City	6	8	10
Omaha	7	11	11
Des Moines	4	5	12

Untuk menentukan berapa banyak ton gandum yang harus dikirim dari tiap cerobong butir gandum ke tiap penggilingan setiap bulannya agar total biaya transportasi minimum

Ke Dari	A	B	C	Pasokan
1	6	8	10	150
2	7	11	11	175
3	4	5	12	275
Permintaan	200	100	300	600



# Hasil Output

```
C:\Windows\System32\cmd.exe

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS>py vogel.py
Cost Tableau:
[6, 8, 10]
[7, 11, 11]
[4, 5, 12]

Allocation Tableau:
[0, 0, 150]
[175, 0, 0]
[25, 100, 150]

Ongkos Minimum:
150x10 + 175x7 + 25x4 + 100x5 + 150x12 = 5125

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS>_
```

Table cost tidak berubah karena total nilai supply = demand, sehingga tidak terbentuk dummy.

# TORA

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\yes.txt

File EditGrid

**TRANSPORTATION MODEL**

Problem Title:

No. of Sources

No. of Dest'ns

Editing Grid:  
>>To DELETE, INSERT, COPY, or PASTE a column(row), click heading cell of target column(row), then invoke pull-down EditGrid menu  
>>For INSERT mode, a single(double) click of target row/column will place new row/column after(before) target row/column.

**INPUT GRID - TRANSPORTATION**

	S/D Name	D1	D2	D3	Supply
S1	Kansas City	6,00	8,00	10,00	150
S2	Omaha	7,00	11,00	11,00	175
S3	Des Moines	4,00	5,00	12,00	275
Demand		200	100	300	

SOLVE Menu MAIN Menu Exit TORA

22:51

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\yes.txt

TRANSPORTATION MODEL

TORA Optimization System, Windows-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Minggu, October 28, 2018 22:52

TRANSPORTATION TABLEAU - (Vogel's Method)

Title: vogel -(minimum cost)

Steps for generating transportation tableaus:  
1. (Optional step) Initialize ONE of the simplex multiplier (u1, u2, ..., v1, v2 ...) to zero value (default u1 = 0)  
2. Click (in any order) the cells defining the change-of-basis loop (if corret, cell changes color)  
3. Click command button NEXT ITERATION (or ALL ITERATIONS) -- This step may be executed without Step 2

Initialize u or v  
u1=0

Next Iteration All Iterations Write to Printer

Iter 1	ObjVal =	5125,00	D1	D2	D3	Supply
	Name		Chicago	St. Louis	Cincinnati	
			v1=2,00	v2=3,00	v3=10,00	
			6,00	8,00	10,00	
S1	Kansas City	u1=0,00			150	150
			-4,00	-5,00	0,00	
S2	Omaha	u2=5,00	7,00	11,00	11,00	175
			175			
			0,00	-3,00	4,00	
S3	Des Moines	u3=2,00	4,00	5,00	12,00	275
			25	100	150	
			0,00	0,00	0,00	
	Demand		200	100	300	

View/Modify Input Data

MAIN Menu

Exit TORA

22:52

# Sourcecode

```
File Edit Selection Find View Goto Tools Project
vogel.py x
1 cost=[
2     [6,8,10],
3     [7,11,11],
4     [4,5,12]
5 ]
6 supply=[150,175,275]
7 demand=[200,100,300]
8
9 def vogel(s,d):
10     table=[]
11     for x in range(s):
12         temp=[]
13         for y in range(d):
14             temp.append(0)
15         table.append(temp)
16     toAllocate(table,s,d)
17     return table
18
19 def toAllocate(table2,s,d):
20     global sTot,dTot,itors
21     if sTot==0 and dTot==0:
22         return table2
23     sortedRow=sortCost(s,d,0)
24     sortedCol=sortCost(d,s,1)
25     pRow=penaltyVal(sortedRow)
26     pCol=penaltyVal(sortedCol)
27     sortedAll=sortAll(pRow,pCol)
28     indexRow=indexCell(s,d,0)
29     indexCol=indexCell(d,s,1)
```

```
File Edit Selection Find View Goto
vogel.py x
30 direc=sortedAll[0][3]
31 if direc==0:
32     idx=sortedAll[0][2]
33     x=indexRow[idx][0][1]
34     y=indexRow[idx][0][2]
35 elif direc==1:
36     idx=sortedAll[0][2]
37     x=indexCol[idx][0][1]
38     y=indexCol[idx][0][2]
39 if supply[x]<demand[y]:
40     table2[x][y]=supply[x]
41     demand[y]-=supply[x]
42     supply[x]=0
43     sTot=sum(supply)
44     dTot=sum(demand)
45     for a in range(d):
46         cost[x][a]=1000000
47     toAllocate(table2,s,d)
48 elif supply[x]>demand[y]:
49     table2[x][y]=demand[y]
50     supply[x]-=demand[y]
51     demand[y]=0
52     sTot=sum(supply)
53     dTot=sum(demand)
54     for z in range(s):
55         cost[z][y]=1000000
56     toAllocate(table2,s,d)
57 elif supply[x]==demand[y]:
58     table2[x][y]=supply[x]
```

```
File Edit Selection Find View Goto Tools Project
vogel.py x
59 supply[x]=0
60 demand[y]=0
61 sTot=sum(supply)
62 dTot=sum(demand)
63 for a in range(d):
64     cost[x][a]=1000000
65 for z in range(s):
66     cost[z][y]=1000000
67 toAllocate(table2,s,d)
68
69 def sortCost(d,s,flag):
70     box=[]
71     for x in range(d):
72         temp=[]
73         for y in range(s):
74             if flag==1:
75                 temp.append(cost[y][x])
76             else:
77                 temp.append(cost[x][y])
78         temp2=sorted(temp)
79         box.append(temp2)
80     return box
81
82 def penaltyVal(lists):
83     penBox=[]
84     x=0
85     for row in lists:
86         if(len(row)==1):
87             penalty=row[0]
```

```
File Edit Selection Find View Goto Tools Project Pref
vogel.py x
88     else:
89         penalty=row[1]-row[0]
90         temp=[]
91         temp.insert(0,penalty)
92         temp.insert(1,row[0])
93         temp.insert(2,x)
94         penBox.append(temp)
95         x+=1
96     return penBox
97
98 def indexCell(d,s,flag):
99     box=[]
100     for x in range(d):
101         temp2=[]
102         for y in range(s):
103             temp=[]
104             if flag==1:
105                 temp.insert(0,cost[y][x])
106                 temp.insert(1,y)
107                 temp.insert(2,x)
108             else:
109                 temp.insert(0,cost[x][y])
110                 temp.insert(1,x)
111                 temp.insert(2,y)
112             temp2.append(temp)
113         temp2=sorted(temp2)
114         box.append(temp2)
115     return box
116
117 def sortAll(row,col):
118     box=[]
119     for x in row:
120         x+=[0]
121         box.append(x)
122     for x in col:
123         x+=[1]
124         box.append(x)
125     box.sort(key=Lambda z: (-z[0], z[1]))
126     return box
127
128 sTot=sum(supply)
129 dTot=sum(demand)
130 s=len(supply)
```

```
C:\
File Edit Selection Find View Goto Tools Project Preferen
vogel.py x
131 d=len(demand)
132 if sTot<dTot:
133     s+=1
134     supply.append(dTot - sTot)
135     sTot=sum(supply)
136     temp=[]
137     for x in range(len(demand)):
138         temp.append(0)
139     cost.append(temp)
140
141 elif sTot>dTot:
142     d+=1
143     demand.append(sTot - dTot)
144     dTot=sum(demand)
145     for x in range(len(supply)):
146         cost[x].append(0)
147
148 import copy
149 cost2=copy.deepcopy(cost)
150 resultTable=vogel(s,d)
151 print("Cost Tableau:")
152 for row in cost2:
153     print(row)
154 print("\nAllocation Tableau:")
155 for row in resultTable:
156     print(row)
157 print("\nongkos Minimum:")
158 ongkos=0
159 flag=0
160 for x in range(s):
161     for y in range(d):
162         if resultTable[x][y]==0:
163             continue
164         if flag==1:
165             print(" + ", end='')
166             print("%d"%(resultTable[x][y]),end='')
167             print("%d"%(cost2[x][y]),end='')
168             if x<s-1 or y<d-1:
169                 flag=1
170             ongkos+=resultTable[x][y]*cost2[x][y]
171 print(" =",ongkos)
```

# Studi Kasus 2

Tempat peleburan baja yang ada di tiga kota memproduksi sejumlah baja sebagai berikut :

<b>Lokasi</b>	<b>Jumlah yang ditawarkan per minggu (ton)</b>
A. Bethlehem	150
B. Birmingham	210
C. Gary	320

Ketiga tempat peleburan memasok baja ke empat kota dimana pabrik-pabriknya mempunyai permintaan sebagai berikut :

<b>Lokasi</b>	<b>Jumlah yang diminta per minggu (ton)</b>
1. Detroit	130
2. St. Louis	70
3. Chicago	180
4. Nortfolk	240

Biaya pengiriman per-ton baja adalah sebagai berikut

Dari	Ke			
	1	2	3	4
A	14	9	16	18
B	11	8	7	6
C	16	12	10	22



# Hasil Output

```
C:\Windows\System32\cmd.exe

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UT5>py vogel.py
Cost Tableau:
[14, 9, 16, 18, 0]
[11, 8, 7, 6, 0]
[16, 12, 10, 22, 0]

Allocation Tableau:
[50, 70, 0, 30, 0]
[0, 0, 0, 210, 0]
[80, 0, 180, 0, 60]

Ongkos Minimum:
50x14 + 70x9 + 30x18 + 210x6 + 80x16 + 180x10 + 60x0 = 6210

C:\Users\Nuzha Musyafira\Documents\0 RISOP\UT5>
```

Karena total nilai supply tidak sama dengan demand, yaitu supply > demand, maka terbentuk dummy baru untuk mewakili demand yang kurang

# TORA

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\lc2.txt

File EditGrid

**TRANSPORTATION MODEL**

Problem Title:

No. of Sources

No. of Dest'ns

Editing Grid:  
>>To DELETE, INSERT, COPY, or PASTE a column(row), click heading cell of target column(row), then invoke pull-down EditGrid menu  
>>For INSERT mode, a single(double) click of target row/column will place new row/column after(before) target row/column.

**INPUT GRID - TRANSPORTATION**

		D1	D2	D3	D4	D5	Supply
	S/D Name	Detroit	St. Louis	Chicago	Norfolk	Dummy	
\$1	Bethlehem	14,00	9,00	16,00	18,00	0,00	150
\$2	Birmingham	11,00	8,00	7,00	6,00	0,00	210
\$3	Gary	16,00	12,00	10,00	22,00	0,00	320
Demand		130	70	180	240	60	

SOLVE Menu MAIN Menu Exit TORA

23:11

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\vo2.txt

TRANSPORTATION MODEL

TORA Optimization System, Windows®-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Minggu, October 28, 2018 23:12

TRANSPORTATION TABLEAU - (Vogel's Method)

Title: vogel -(minimum cost)

Steps for generating transportation tableaux:  
1. (Optional step) Initialize ONE of the simplex multiplier (u1, u2, ..., v1, v2 ...) to zero value (default u1 = 0)  
2. Click (in any order) the cells defining the change-of-basis loop (if corret, cell changes color)  
3. Click command button NEXT ITERATION (or ALL ITERATIONS) – This step may be executed without Step 2

Initialize u or v  
u1=0

Next Iteration All Iterations Write to Printer

Iter 1	ObjVal =	6210,00	D1	D2	D3	D4	D5	Supply
	Name		Detroit	St. Louis	Chicago	Nortfolk	Dummy	
			v1=14,00	v2=9,00	v3=8,00	v4=18,00	v5=-2,00	
			14,00	9,00	16,00	18,00	0,00	
S1	Bethlehem	u1=0,00	50	70		30		150
			0,00	0,00	-8,00	0,00	-2,00	
S2	Birmingham	u2=-12,00	11,00	8,00	7,00	6,00	0,00	210
			-9,00	-11,00	-11,00	0,00	-14,00	
S3	Gary	u3=2,00	16,00	12,00	10,00	22,00	0,00	
			80		180		60	320
			0,00	-1,00	0,00	-2,00	0,00	
	Demand		130	70	180	240	60	

View/Modify Input Data MAIN Menu Exit TORA

23:12

# Sourcecode

```
File Edit Selection Find View Goto To
vogel.py x table
1 cost=[
2     [14,9,16,18],
3     [11,8,7,6],
4     [16,12,10,22]
5 ]
6 supply=[150,210,320]
7 demand=[130,70,180,240]
8
9 def vogel(s,d):
10     table=[]
11     for x in range(s):
12         temp=[]
13         for y in range(d):
14             temp.append(0)
15         table.append(temp)
16     toAllocate(table,s,d)
17     return table
18
19 def toAllocate(table2,s,d):
20     global sTot,dTot,itors
21     if sTot==0 and dTot==0:
22         return table2
23     sortedRow=sortCost(s,d,0)
24     sortedCol=sortCost(d,s,1)
25     pRow=penaltyVal(sortedRow)
26     pCol=penaltyVal(sortedCol)
27     sortedAll=sortAll(pRow,pCol)
28     indexRow=indexCell(s,d,0)
29     indexCol=indexCell(d,s,1)
```

```
File Edit Selection Find View Goto
vogel.py x
30 direc=sortedAll[0][3]
31 if direc==0:
32     idx=sortedAll[0][2]
33     x=indexRow[idx][0][1]
34     y=indexRow[idx][0][2]
35 elif direc==1:
36     idx=sortedAll[0][2]
37     x=indexCol[idx][0][1]
38     y=indexCol[idx][0][2]
39 if supply[x]<demand[y]:
40     table2[x][y]=supply[x]
41     demand[y]-=supply[x]
42     supply[x]=0
43     sTot=sum(supply)
44     dTot=sum(demand)
45     for a in range(d):
46         cost[x][a]=1000000
47     toAllocate(table2,s,d)
48 elif supply[x]>demand[y]:
49     table2[x][y]=demand[y]
50     supply[x]-=demand[y]
51     demand[y]=0
52     sTot=sum(supply)
53     dTot=sum(demand)
54     for z in range(s):
55         cost[z][y]=1000000
56     toAllocate(table2,s,d)
57 elif supply[x]==demand[y]:
58     table2[x][y]=supply[x]
```

```
File Edit Selection Find View Goto Tools Project
vogel.py x
59 supply[x]=0
60 demand[y]=0
61 sTot=sum(supply)
62 dTot=sum(demand)
63 for a in range(d):
64     cost[x][a]=1000000
65 for z in range(s):
66     cost[z][y]=1000000
67 toAllocate(table2,s,d)
68
69 def sortCost(d,s,flag):
70     box=[]
71     for x in range(d):
72         temp=[]
73         for y in range(s):
74             if flag==1:
75                 temp.append(cost[y][x])
76             else:
77                 temp.append(cost[x][y])
78         temp2=sorted(temp)
79         box.append(temp2)
80     return box
81
82 def penaltyVal(lists):
83     penBox=[]
84     x=0
85     for row in lists:
86         if(len(row)==1):
87             penalty=row[0]
```

```
File Edit Selection Find View Goto Tools Project Pref
vogel.py x
88     else:
89         penalty=row[1]-row[0]
90         temp=[]
91         temp.insert(0,penalty)
92         temp.insert(1,row[0])
93         temp.insert(2,x)
94         penBox.append(temp)
95         x+=1
96     return penBox
97
98 def indexCell(d,s,flag):
99     box=[]
100     for x in range(d):
101         temp2=[]
102         for y in range(s):
103             temp=[]
104             if flag==1:
105                 temp.insert(0,cost[y][x])
106                 temp.insert(1,y)
107                 temp.insert(2,x)
108             else:
109                 temp.insert(0,cost[x][y])
110                 temp.insert(1,x)
111                 temp.insert(2,y)
112             temp2.append(temp)
113         temp2=sorted(temp2)
114         box.append(temp2)
115     return box
116
117 def sortAll(row,col):
118     box=[]
119     for x in row:
120         x+=[0]
121         box.append(x)
122     for x in col:
123         x+=[1]
124         box.append(x)
125     box.sort(key=Lambda z: (-z[0], z[1]))
126     return box
127
128 sTot=sum(supply)
129 dTot=sum(demand)
130 s=len(supply)
```

```
C:\
File Edit Selection Find View Goto Tools Project Preferen
vogel.py x
131 d=len(demand)
132 if sTot<dTot:
133     s+=1
134     supply.append(dTot - sTot)
135     sTot=sum(supply)
136     temp=[]
137     for x in range(len(demand)):
138         temp.append(0)
139     cost.append(temp)
140
141 elif sTot>dTot:
142     d+=1
143     demand.append(sTot - dTot)
144     dTot=sum(demand)
145     for x in range(len(supply)):
146         cost[x].append(0)
147
148 import copy
149 cost2=copy.deepcopy(cost)
150 resultTable=vogel(s,d)
151 print("Cost Tableau:")
152 for row in cost2:
153     print(row)
154 print("\nAllocation Tableau:")
155 for row in resultTable:
156     print(row)
157 print("\nongkos Minimum:")
158 ongkos=0
159 flag=0
160 for x in range(s):
161     for y in range(d):
162         if resultTable[x][y]==0:
163             continue
164         if flag==1:
165             print(" + ", end='')
166             print("%d"%(resultTable[x][y]),end='')
167             print("%d"%(cost2[x][y]),end='')
168             if x<s-1 or y<d-1:
169                 flag=1
170             ongkos+=resultTable[x][y]*cost2[x][y]
171 print(" =",ongkos)
```

# METODE MULTIPLIER

Metode ini merupakan metode optimalisasi dari metode-metode fisibel sebelumnya. Dalam metode ini, pertama-tama menentukan BV dan NBV. Setelah itu, tentukan  $U_x$  (baris) dan  $V_y$  (kolom). Biasanya,  $U_0$  diinisiasikan dengan 0, kemudian elemen selanjutnya akan mengikuti, disesuaikan dengan BV. Cari penalty untuk setiap NBV. Penalty didapat dari  $= C_{x,y} - U_x - V_y$ . Penalty terkecil akan dijadikan EV. Kemudian LV adalah BV bertanda (-) terkecil. EV akan menjadi BV selanjutnya dan LV menjadi NBV. Iterasi dilakukan hingga penalty tidak ada yang bernilai negatif.

# Studi Kasus 1

Gandum dipanen di Midwest (daerah pertanian Amerika bagian Tengah Barat) dan disimpan dalam cerobong butir gandum di tiga kota – Kansas City, Omaha, dan Des Moines. Ketiga cerobong butir gandum ini memasok tiga penggilingan tepung yang berlokasi di Chicago, St. Louis, dan Cincinnati. Butir-butir gandum tersebut dikirim ke penggilingan dengan menggunakan gerbong kereta api, yang tiap gerbongnya memuat satu ton gandum. Setiap bulannya, tiap cerobong butir gandum dapat memasok penggilingan sejumlah ton gandum berikut ini.

<b>Cerobong Butir Gandum</b>	<b>Jumlah yang ditawarkan</b>
1. Kansas City	150
2. Omaha	175
3. Des Moines	275

Jumlah ton gandum yang diminta per bulan dari tiap penggilingan adalah sebagai berikut :

<b>Penggilingan</b>	<b>Jumlah yang diminta</b>
A. Chicago	200
B. St. Louis	100
C. Cincinnati	300



Biaya pengiriman (\$) :

Cerobong Butir Gandum	Penggilingan		
	Chicago (A)	St. Louis (B)	Cincinnati (C)
Kansas City	6	8	10
Omaha	7	11	11
Des Moines	4	5	12

Untuk menentukan berapa banyak ton gandum yang harus dikirim dari tiap cerobong butir gandum ke tiap penggilingan setiap bulannya agar total biaya transportasi minimum

<div>Ke</div> <div>Dari</div>	A	B	C	Pasokan
1	6	8	10	150
2	7	11	11	175
3	4	5	12	275
Permintaan	200	100	300	600

# TORA

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\yes.txt

File EditGrid

**TRANSPORTATION MODEL**

Problem Title:

No. of Sources

No. of Dest'ns

Editing Grid:  
>>To DELETE, INSERT, COPY, or PASTE a column(row), click heading cell of target column(row), then invoke pull-down EditGrid menu  
>>For INSERT mode, a single(double) click of target row/column will place new row/column after(before) target row/column.

**INPUT GRID - TRANSPORTATION**

	S/D Name	D1	D2	D3	Supply
S1	Kansas City	6,00	8,00	10,00	150
S2	Omaha	7,00	11,00	11,00	175
S3	Des Moines	4,00	5,00	12,00	275
Demand		200	100	300	

SOLVE Menu MAIN Menu Exit TORA

Windows taskbar: 4:17

TORA Optimization System, Windows®-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Selasa, Oktober 30, 2018 4:17

Title: multiplierNorthwest --(minimum cost)

1. (Optional step) Initialize ONE of the simplex multiplier ( $u_1, u_2, \dots, v_1, v_2 \dots$ ) to zero value (default  $u_1 = 0$ )

2. Click (in any order) the cells defining the change-of-basis loop (if correct, cell changes color)
3. Click command button NEXT ITERATION (or ALL ITERATIONS) – This step may be executed without Step 2

– Initialize  $u$  or  $v$

**u1=0** ▼

Next Iteration All Iterations Write to Printer

Iter 1	ObjVal =	5925,00	D1	D2	D3	Supply
	Name		Chicago	St. Louis	Cincinnati	
			v1=6,00	v2=10,00	v3=10,00	
			6,00	8,00	10,00	
S1	Kansas City	u1=0,00	150			150
			0,00	2,00	0,00	
			7,00	11,00	11,00	
S2	Omaha	u2=1,00	50	100	25	175
			0,00	0,00	0,00	
			4,00	5,00	12,00	
S3	Des Moines	u3=2,00			275	275
			4,00	7,00	0,00	
	Demand		200	100	300	
Iter 2	ObjVal =	5225,00	D1	D2	D3	Supply
	Name		Chicago	St. Louis	Cincinnati	
			v1=6,00	v2=3,00	v3=10,00	
			6,00	8,00	10,00	
S1	Kansas City	u1=0,00	150			150
			0,00	-5,00	0,00	
			7,00	11,00	11,00	
S2	Omaha	u2=1,00	50		125	175
			0,00	-7,00	0,00	
			4,00	5,00	12,00	
S3	Des Moines	u3=2,00		100	175	275

View/Modify Input Data

[MAIN Menu](#)

Exit TORA

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\yes.txt

TRANSPORTATION MODEL

TORA Optimization System, Windows-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Setasa, October 30, 2010 4:17

TRANSPORTATION TABLEAU - (North-West Corner Method)  
Title: multiplierNorthwest -(minimum cost)  
Steps for generating transportation tableaux:  
1. (Optional step) Initialize ONE of the simplex multiplier (u1, u2, ..., v1, v2 ...) to zero value (default u1 = 0)  
2. Click (in any order) the cells defining the change-of-basis loop (if corret, cell changes color)  
3. Click command button NEXT ITERATION (or ALL ITERATIONS) -- This step may be executed without Step 2

Initialize u or v  
u1=0

Next Iteration All Iterations Write to Printer

	Demand		200	100	300	
Iter 3	ObjVal =	5025,00	D1	D2	D3	Supply
	Name		Chicago	St. Louis	Cincinnati	
			v1=6,00	v2=7,00	v3=14,00	
S1	Kansas City	u1=0,00	6,00	8,00	10,00	
			150			150
S2	Omaha	u2=-3,00	0,00	-1,00	4,00	
			7,00	11,00	11,00	
S3	Des Moines	u3=-2,00	-4,00	-7,00	0,00	175
			4,00	5,00	12,00	
			50	100	125	275
			0,00	0,00	0,00	
Iter 4	Demand		200	100	300	
	ObjVal =	4525,00	D1	D2	D3	Supply
	Name		Chicago	St. Louis	Cincinnati	
			v1=6,00	v2=7,00	v3=10,00	
			6,00	8,00	10,00	
S1	Kansas City	u1=0,00	25		125	150
			0,00	-1,00	0,00	
S2	Omaha	u2=1,00	7,00	11,00	11,00	
			0,00	-3,00	0,00	175
			4,00	5,00	12,00	

View/Modify Input Data MAIN Menu Exit TORA

4:18

TORA C:\Users\Nuzha Musyafira\Documents\0 RISOP\UTS\yes.txt

TRANSPORTATION MODEL

TORA Optimization System, Windows-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Setasa, October 30, 2010 4:17

TRANSPORTATION TABLEAU - (North-West Corner Method)  
Title: multiplierNorthwest -(minimum cost)  
Steps for generating transportation tableaus:  
1. (Optional step) Initialize ONE of the simplex multiplier (u1, u2, ..., v1, v2 ...) to zero value (default u1 = 0)  
2. Click (in any order) the cells defining the change-of-basis loop (if corret, cell changes color)  
3. Click command button NEXT ITERATION (or ALL ITERATIONS) -- This step may be executed without Step 2

Initialize u or v  
u1=0

Next Iteration All Iterations Write to Printer

S1	Kansas City	u1=0.00	6.00	8.00	10.00	150
			0.00	-1.00	4.00	
S2	Omaha	u2=-3.00	7.00	11.00	11.00	175
			-4.00	-7.00	0.00	
S3	Des Moines	u3=-2.00	4.00	5.00	12.00	275
			0.00	0.00	0.00	
	Demand		200	100	300	
Iter 4	ObjVal =	4525.00	D1	D2	D3	Supply
	Name		Chicago	St. Louis	Cincinnati	
			v1=6.00	v2=7.00	v3=10.00	
S1	Kansas City	u1=0.00	6.00	8.00	10.00	150
			0.00	-1.00	0.00	
S2	Omaha	u2=-1.00	7.00	11.00	11.00	175
			0.00	-3.00	0.00	
S3	Des Moines	u3=-2.00	4.00	5.00	12.00	275
			0.00	0.00	-4.00	
	Demand		200	100	300	

View/Modify Input Data

MAIN Menu

Exit TORA

4:18

TORA Optimization System, Windows®-version 1.00  
Copyright © 2000-2002 Hamdy A. Taha. All Rights Reserved  
Selasa, Oktober 30, 2018 4:19

Title: multiplierNorthwest  
Final Iteration No.: 2  
Objective Value (minimum cost) =4525,00

From	To	Amt Shipped	Obj Coeff	Obj Contrib
S1: Kansas City	D1: Chicago	25	6,00	150,00
S1: Kansas City	D3: Cincinnati	125	10,00	1250,00
S2: Omaha	D3: Cincinnati	175	11,00	1925,00
S3: Des Moines	D1: Chicago	175	4,00	700,00
S3: Des Moines	D2: St. Louis	100	5,00	500,00

Exit TORA

# Kesimpulan Analisis Program dan Hasil Program

Secara rata-rata, program memiliki kesamaan algoritma pada inisiasi awal, pembentukan variabel dummy, dan pengalokasian nilai. Yang membedakan algoritma-algoritma tersebut yaitu passing parameter pada fungsi recursive. Untuk hasil program, dari segi kecepatan algoritma pojok kiri atas adalah yang paling cepat karena proses eksekusi pada fungsi alokasi terbilang simpel dan mudah. Untuk segi presisi dan optimalisasi ongkos, algoritma ongkos minimum dan vogel merupakan yang terbaik karena pengekseskusan yang lebih detail. Selain itu, ketika dilanjutkan dengan metode multiplier, algoritma pojok kiri atas membutuhkan 4 kali iterasi untuk mencapai hasil optimal, sedangkan algoritma ongkos minimum dna vogel hanya membutuhkan 2 kali iterasi.