Taller de Heurísticas

Integrantes

- JHONATAN STIVENS BARRERA ORDOÑEZ **1088274600**
- SERGIO ALEXANDER FLOREZ CÓDIGO **1088297456**

Ejercicio 1. Problema de la Mochila

- Plantee usted mismo un problema tipo Mochila, que contenga al menos quince (15) variables, con los pesos y valores que usted desee.

Artículo (qi)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Peso (wi)	42	23	21	15	7	28	30	10	18	33	10	10	50	30	25
Valor (vi)	100	60	70	15	15	50	30	60	23	90	10	80	10	10	77

$$\begin{aligned} \text{maximizar} \quad & \sum_{i=1}^{n} v_i x_i \\ \text{tal que} \quad & \sum_{i=1}^{n} w_i x_i \leq W \\ \text{y} \quad & 1 \leq q_i \leq \infty. \end{aligned}$$

RESTRICCIÓN PRINCIPAL

W = 300

a. Mediante seis (6) heurísticas definidas por usted, de solución a dicho problema. Muestre los cuadros de solución indicando el valor de la función obtenido y cuáles son las variables.

In [4]:
```
W = 300
qCount = 15
```

In [5]:
```
from pprint import pprint

weights = [ 42, 23, 21, 15, 7, 28, 30, 10, 18, 33, 10, 10, 50, 30, 25
 ]
values  = [ 100, 60, 70, 15, 15, 50, 30, 60, 23, 90, 10, 80, 10, 10,
77 ]
# Crear lista de elementos para poder ordenar y operar más fácil con
 los elementos
items = list()

wMax = 0
vMax = 0
for i in range(0, qCount):
    wMax += weights[i]
    vMax += values[i]
    items.append((i, weights[i], values[i], weights[i]/ values[i],))


def getI(item):
    return item[0]

def getW(item):
    return item[1]

def getV(item):
    return item[2]

def getD(item):
    return item[3]

print("wMax: ", wMax)
print("vMax: ", vMax)
print()
print("(index, weigth, value, value/weigth(density))")
print()
pprint(items)
```

```
wMax:  352
vMax:  700

(index, weigth, value, value/weigth(density))

[(0, 42, 100, 0.42),
 (1, 23, 60, 0.38333333333333336),
 (2, 21, 70, 0.3),
 (3, 15, 15, 1.0),
 (4, 7, 15, 0.4666666666666667),
 (5, 28, 50, 0.56),
 (6, 30, 30, 1.0),
 (7, 10, 60, 0.16666666666666666),
 (8, 18, 23, 0.782608695652174),
 (9, 33, 90, 0.36666666666666664),
 (10, 10, 10, 1.0),
 (11, 10, 80, 0.125),
 (12, 50, 10, 5.0),
 (13, 30, 10, 3.0),
 (14, 25, 77, 0.3246753246753247)]
```

Heurística 1: Seleccionar elementos con base a su orden original.

In [72]:
```
wAcum = 0
vAcum = 0

for i in range(0, qCount):
    if wAcum + getW(items[i]) <= W:
        wAcum += getW(items[i])
        vAcum += getV(items[i])

print("wAcum: ", wAcum)
print("vAcum: ", vAcum)
```

```
wAcum:  297
vAcum:  613
```

Heurística 2: Seleccionar los más valiosos primero.

In [73]:
```
# Ordenar elementos por valor más alto primero
itemsSorted = sorted(items, key=getV, reverse=True)

wAcum = 0
vAcum = 0

for i in range(0, qCount):
    if wAcum + getW(itemsSorted[i]) <= W:
        wAcum += getW(itemsSorted[i])
        vAcum += getV(itemsSorted[i])

print("wAcum: ", wAcum)
print("vAcum: ", vAcum)
```

```
wAcum:  272
vAcum:  680
```

Heurística 3: Seleccionar los más livianos primero.

In [74]:
```
# Ordenar elementos por el peso más liviano primero
itemsSorted = sorted(items, key=getW)

wAcum = 0
vAcum = 0

for i in range(0, qCount):
    if wAcum + getW(itemsSorted[i]) <= W:
        wAcum += getW(itemsSorted[i])
        vAcum += getV(itemsSorted[i])

print("wAcum: ", wAcum)
print("vAcum: ", vAcum)
```

```
wAcum:  260
vAcum:  590
```

Heurística 4: Seleccionar los que tengan menor relación **valor/peso (densidad)**.

In [75]:
```
# Ordenar elementos la menor relación de densidad.
itemsSorted = sorted(items, key=getD)

wAcum = 0
vAcum = 0

for i in range(0, qCount):
    if wAcum + getW(itemsSorted[i]) <= W:
        wAcum += getW(itemsSorted[i])
        vAcum += getV(itemsSorted[i])

print("wAcum: ", wAcum)
print("vAcum: ", vAcum)
```

```
wAcum:  272
vAcum:  680
```

Heurística 5: Seleccionar los que tengan mayor relación **valor/peso (densidad)**.

In [84]:
```
# Ordenar elementos la mayor relación de densidad.
itemsSorted = sorted(items, key=getD, reverse=True)

wAcum = 0
vAcum = 0

for i in range(0, qCount):
    if wAcum + getW(itemsSorted[i]) <= W:
        wAcum += getW(itemsSorted[i])
        vAcum += getV(itemsSorted[i])

print("wAcum: ", wAcum)
print("vAcum: ", vAcum)
```

```
wAcum:  296
vAcum:  473
```

Heurística 6: Seleccionar los elementos de forma aleatoria

In [212]:
```
# Ramdonizo los elementos y voy seleccionando
import random
itemsRandom = list(items)
random.shuffle(itemsRandom)

pprint(itemsRandom)

wAcum = 0
vAcum = 0

for i in range(0, qCount):
    if wAcum + getW(itemsRandom[i]) <= W:
        print("i: ", getI(itemsRandom[i])+1)
        wAcum += getW(itemsRandom[i])
        vAcum += getV(itemsRandom[i])

print("wAcum: ", wAcum)
print("vAcum: ", vAcum)
```

```
[(2, 21, 70, 0.3),
 (8, 18, 23, 0.782608695652174),
 (5, 28, 50, 0.56),
 (1, 23, 60, 0.38333333333333336),
 (9, 33, 90, 0.36666666666666664),
 (4, 7, 15, 0.4666666666666667),
 (6, 30, 30, 1.0),
 (12, 50, 10, 5.0),
 (10, 10, 10, 1.0),
 (13, 30, 10, 3.0),
 (11, 10, 80, 0.125),
 (7, 10, 60, 0.16666666666666666),
 (3, 15, 15, 1.0),
 (0, 42, 100, 0.42),
 (14, 25, 77, 0.3246753246753247)]
i:  3
i:  9
i:  6
i:  2
i:  10
i:  5
i:  7
i:  13
i:  11
i:  14
i:  12
i:  8
i:  4
wAcum:  285
vAcum:  523
```

Solución

- **Heurística 1 (H1):** Seleccionar elementos con base a su orden original.
- **Heurística 2 (H2):** Seleccionar los más valiosos primero.
- **Heurística 3 (H3):** Seleccionar los más livianos primero.
- **Heurística 4 (H4):** Seleccionar los que tengan menor relación valor/peso (densidad).
- **Heurística 5 (H5):** Seleccionar los que tengan mayor relación **valor/peso (densidad)**.
- **Heurística 6 (H6):** Seleccionar los elementos de forma aleatoria

	H1	**H2**	**H3**	**H4**	**H5**	**H6**	**CHU-BEASLY**
WeigthAcum	297	272	260	272	296	285	272
ValueAcum	613	680	590	680	473	523	680

b. Muestre dos pasos usando el algoritmo de vecindarios usando alguna heruristica para la escogencia de la siguiente solución.

Para el algoritmo de vecindad escogeré como mi vector inicial el siguiente:

Vector inicial:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	0	1	1	1	1	1	1	0

Vecindarios:

Vecino 1

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	0	0	0	0	1	1	1	1	1	1	0

Vecino 2

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	1	1	1	1	1	0

Vecino 3

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	0	0	0	0	1	1	1	1	1	1	0

Vecino 4

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	0	0	0	0	1	1	1	1	1	1	0

Vecino 5

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	1	0	0	0	1	1	1	1	1	1	0

Vecino 6

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	1	0	0	1	1	1	1	1	1	0

Vecino 7

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	1	0	1	1	1	1	1	1	0

Vecino 8

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	1	1	1	1	1	1	1	0

Vecino 9

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	0	0	1	1	1	1	1	0

Vecino 10

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	0	1	0	1	1	1	1	0

Vecino 11

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	0	1	1	0	1	1	1	0

Vecino 12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	0	1	1	1	0	1	1	0

Vecino 13

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	0	1	1	1	1	0	1	0

Vecino 14

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	0	1	1	1	1	1	0	0

Vecino 15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	0	0	1	1	1	1	1	1	1

Heurística para escoger el siguiente

Usaré la heurística de **best improvement**, donde elijo el mejor de los vecinos generados.

In [220]:
```
solution = [ 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0 ]
```

In [219]:
```
def FO(solution):
    vAcum = 0
    for i in range(0, qCount):
        vAcum += getV(items[i])*solution[i]
    return vAcum

def calcW(solution):
    wAcum = 0
    for i in range(0, qCount):
        wAcum += getW(items[i])*solution[i]
    return wAcum


initialBestFO = FO(solution)
initialBestW = calcW(solution)

print("Original FO: ", initialBestFO)
print("Original W: ", initialBestW)
print()

bestNeigborIndex = 0
bestNeigborFO = initialBestFO
bestNeigborW = initialBestW
bestNeigborSol = list(solution)


iteration = 1
bestIterationFO = initialBestFO
bestIterationW = initialBestW
bestIterationSol = list(solution)
while (True):
    if (bestIterationFO < bestNeigborFO):
        bestIterationFO = bestNeigborFO
        bestIterationSol = list(bestNeigborSol)
        bestIterationW = bestNeigborW

    for i in range(0, qCount):
        neighbor = list(bestIterationSol)
        neighbor[i] = 1 - neighbor[i]
        pprint(neighbor)
        neighborFO = FO(neighbor)
        neighborW  = calcW(neighbor)
        print("Neighbor FO(%d): %d -- W: %d" % (i+1 , neighborFO, nei
ghborW))
        if (neighborFO > bestIterationFO and neighborW <= W):
            bestNeigborIndex = i
            bestNeigborFO = neighborFO
            bestNeigborSol = list(neighbor)
            bestNeigborW = neighborW

    print("---------------")
    print("BEST SOLUTION ITERATION (%d)" % iteration)
    print("bestNeigborIndex= %d" % bestNeigborIndex)
    print("bestNeigborW= %d" % bestNeigborW)
    print("bestNeigborFO= %d" % bestNeigborFO)
    print("bestNeigborSol= ", bestNeigborSol)
    print()
```

```
    iteration = iteration + 1

    if (bestIterationFO == bestNeigborFO):
        break
```

```
Original FO:  368
Original W:  210

[1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(1): 468 -- W: 252
[0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(2): 308 -- W: 187
[0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(3): 298 -- W: 189
[0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(4): 353 -- W: 195
[0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(5): 383 -- W: 217
[0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(6): 418 -- W: 238
[0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(7): 398 -- W: 240
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(8): 428 -- W: 220
[0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0]
Neighbor FO(9): 345 -- W: 192
[0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0]
Neighbor FO(10): 278 -- W: 177
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0]
Neighbor FO(11): 358 -- W: 200
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0]
Neighbor FO(12): 288 -- W: 200
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0]
Neighbor FO(13): 358 -- W: 160
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0]
Neighbor FO(14): 358 -- W: 180
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(15): 445 -- W: 235
---------------
BEST SOLUTION ITERATION (1)
bestNeigborIndex= 14
bestNeigborW= 235
bestNeigborFO= 445
bestNeigborSol=  [0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(1): 545 -- W: 277
[0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(2): 385 -- W: 212
[0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(3): 375 -- W: 214
[0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(4): 430 -- W: 220
[0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(5): 460 -- W: 242
[0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(6): 495 -- W: 263
[0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(7): 475 -- W: 265
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(8): 505 -- W: 245
[0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
```

```
Neighbor FO(9): 422 -- W: 217
[0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1]
Neighbor FO(10): 355 -- W: 202
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1]
Neighbor FO(11): 435 -- W: 225
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1]
Neighbor FO(12): 365 -- W: 225
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1]
Neighbor FO(13): 435 -- W: 185
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1]
Neighbor FO(14): 435 -- W: 205
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(15): 368 -- W: 210
---------------
BEST SOLUTION ITERATION (2)
bestNeigborIndex= 7
bestNeigborW= 245
bestNeigborFO= 505
bestNeigborSol=  [0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(1): 605 -- W: 287
[0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(2): 445 -- W: 222
[0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(3): 435 -- W: 224
[0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(4): 490 -- W: 230
[0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(5): 520 -- W: 252
[0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(6): 555 -- W: 273
[0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(7): 535 -- W: 275
[0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(8): 445 -- W: 235
[0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1]
Neighbor FO(9): 482 -- W: 227
[0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1]
Neighbor FO(10): 415 -- W: 212
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1]
Neighbor FO(11): 495 -- W: 235
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1]
Neighbor FO(12): 425 -- W: 235
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1]
Neighbor FO(13): 495 -- W: 195
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1]
Neighbor FO(14): 495 -- W: 215
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(15): 428 -- W: 220
---------------
BEST SOLUTION ITERATION (3)
bestNeigborIndex= 6
bestNeigborW= 275
bestNeigborFO= 535
bestNeigborSol=  [0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
[1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(1): 635 -- W: 317
[0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(2): 475 -- W: 252
[0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(3): 465 -- W: 254
[0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(4): 520 -- W: 260
[0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(5): 550 -- W: 282
[0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(6): 585 -- W: 303
[0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(7): 505 -- W: 245
[0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(8): 475 -- W: 265
[0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1]
Neighbor FO(9): 512 -- W: 257
[0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1]
Neighbor FO(10): 445 -- W: 242
[0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1]
Neighbor FO(11): 525 -- W: 265
[0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1]
Neighbor FO(12): 455 -- W: 265
[0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1]
Neighbor FO(13): 525 -- W: 225
[0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1]
Neighbor FO(14): 525 -- W: 245
[0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(15): 458 -- W: 250
---------------
BEST SOLUTION ITERATION (4)
bestNeigborIndex= 4
bestNeigborW= 282
bestNeigborFO= 550
bestNeigborSol=  [0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]

[1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(1): 650 -- W: 324
[0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(2): 490 -- W: 259
[0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(3): 480 -- W: 261
[0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(4): 535 -- W: 267
[0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(5): 535 -- W: 275
[0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(6): 600 -- W: 310
[0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(7): 520 -- W: 252
[0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1]
Neighbor FO(8): 490 -- W: 272
[0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1]
Neighbor FO(9): 527 -- W: 264
[0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1]
Neighbor FO(10): 460 -- W: 249
```

```
[0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1]
Neighbor FO(11): 540 -- W: 272
[0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1]
Neighbor FO(12): 470 -- W: 272
[0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1]
Neighbor FO(13): 540 -- W: 232
[0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1]
Neighbor FO(14): 540 -- W: 252
[0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0]
Neighbor FO(15): 473 -- W: 257
---------------
BEST SOLUTION ITERATION (5)
bestNeigborIndex= 4
bestNeigborW= 282
bestNeigborFO= 550
bestNeigborSol=  [0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Para la solucción elegida, lastimosamente usando la heurística de vencindades no fue posible encontrar una mejor solución que las de las otras heurísticas planteadas, pero si mejoramos bastante la solución inicial.

Ejercicio 2. Problema del Agente viajero para 6 ciudades

Se tiene la siguiente matriz de distancias entre ciudades. Se desea solucionar el problema del agente viajero.

Tabla 1: Matriz de Distancias

Ciudades	1	2	3	4	5	6
1	0	9.5	7	10	3.5	7
2	9.5	0	4.4	3.5	9	7.5
3	6	4	0	5.5	4.5	4.5
4	9	3.5	3.5	0	7.5	5.5
5	5.5	11	4.5	7.5	0	4.1
6	7.6	7.5	3.5	7	4.1	0

a. Defina el Modelo matemático

El TSP puede ser formulado por la programación lineal en enteros. Sea **Xij** igual 1, si existe el camino de ir de la i a la ciudad j, y 0 en otro caso, para el conjunto de ciudades 0,..., n. Sean **Ui** para i = 1,..., n variables artificiales y sea **Cij** la distancia desde la ciudad i a la ciudad j. Entonces el modelo de programación lineal en enteros puede ser escrito como:

$$\min \sum_{i=0}^{n} \sum_{j \neq i, j=0}^{n} c_{ij} x_{ij}$$

$$0 \leq x_{ij} \leq 1 \qquad i, j = 0, \ldots n$$

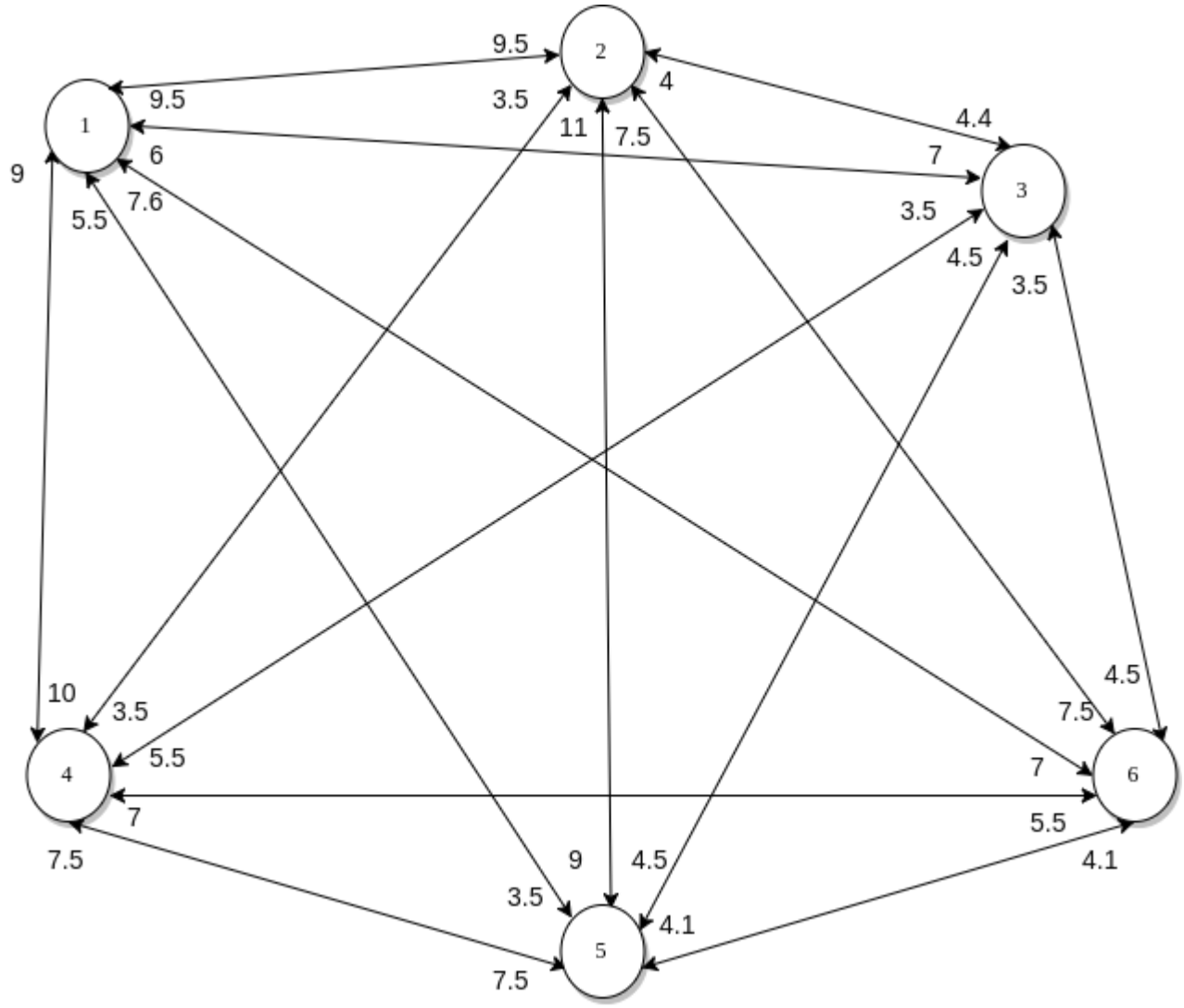
$$x_{ij} \text{ integer} \qquad i, j = 0, \ldots n$$

$$\sum_{i=0, i \neq j}^{n} x_{ij} = 1 \qquad j = 0, \ldots, n$$

$$\sum_{j=0, j \neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 1 \leq i \neq j \leq n.$$

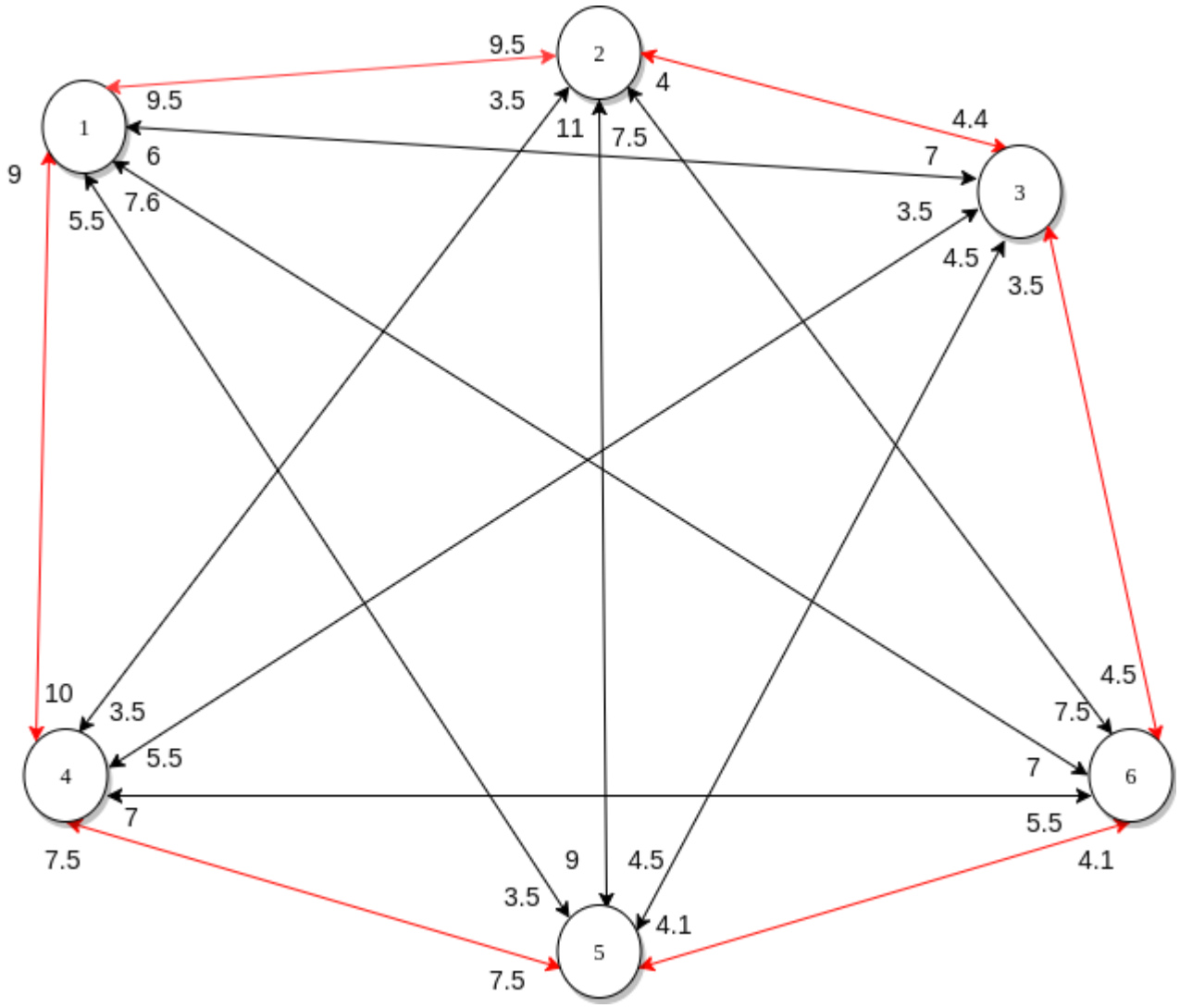
In [29]:
```
distances = [
    [0.0, 9.5 , 7.0, 10.0, 3.5, 7.0],
    [9.5, 0.0 , 4.4, 3.5 , 9.0, 7.5],
    [6.0, 4.0 , 0.0, 5.5 , 4.5, 4.5],
    [9.0, 3.5 , 3.5, 0.0 , 7.5, 5.5],
    [5.5, 11.0, 4.5, 7.5 , 0.0, 4.1],
    [7.6, 7.5 , 3.5, 7.0 , 4.1, 0.0]
]

V = [ 1, 2, 3, 4, 5 , 6 ]
E = [
    [(1, 2), (1, 3), (1, 4), (1, 5), (1, 6)],
    [(2, 1), (2, 3), (2, 4), (2, 5), (2, 6)],
    [(3, 1), (3, 2), (3, 4), (3, 5), (3, 6)],
    [(4, 1), (4, 2), (4, 3), (4, 5), (4, 6)],
    [(5, 1), (5, 2), (5, 3), (5, 4), (5, 6)],
    [(6, 1), (6, 2), (6, 3), (6, 4), (6, 6)],
]
```



b. Sugiera un vector que represente una alternativa de solución para este problema.

Vector solución usando Heurística simple seleccionando las ciudades con base a su orden.

1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 1



In [41]:
```
# 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 1
solution = [
    (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 1),
]

def calcGraphFO(solution):
    distFO = 0
    for edge in solution:
        distFO += distances[edge[0]-1][edge[1]-1]

    return distFO

print("SolutionFO: %f" % calcGraphFO(solution))
```

```
SolutionFO: 38.600000
```

---

c. Usando al menos dos (2) heurísticas, encuentre una solución y compárelas.

Heurística 1: Seleccionar la ciudad más cercana primero.

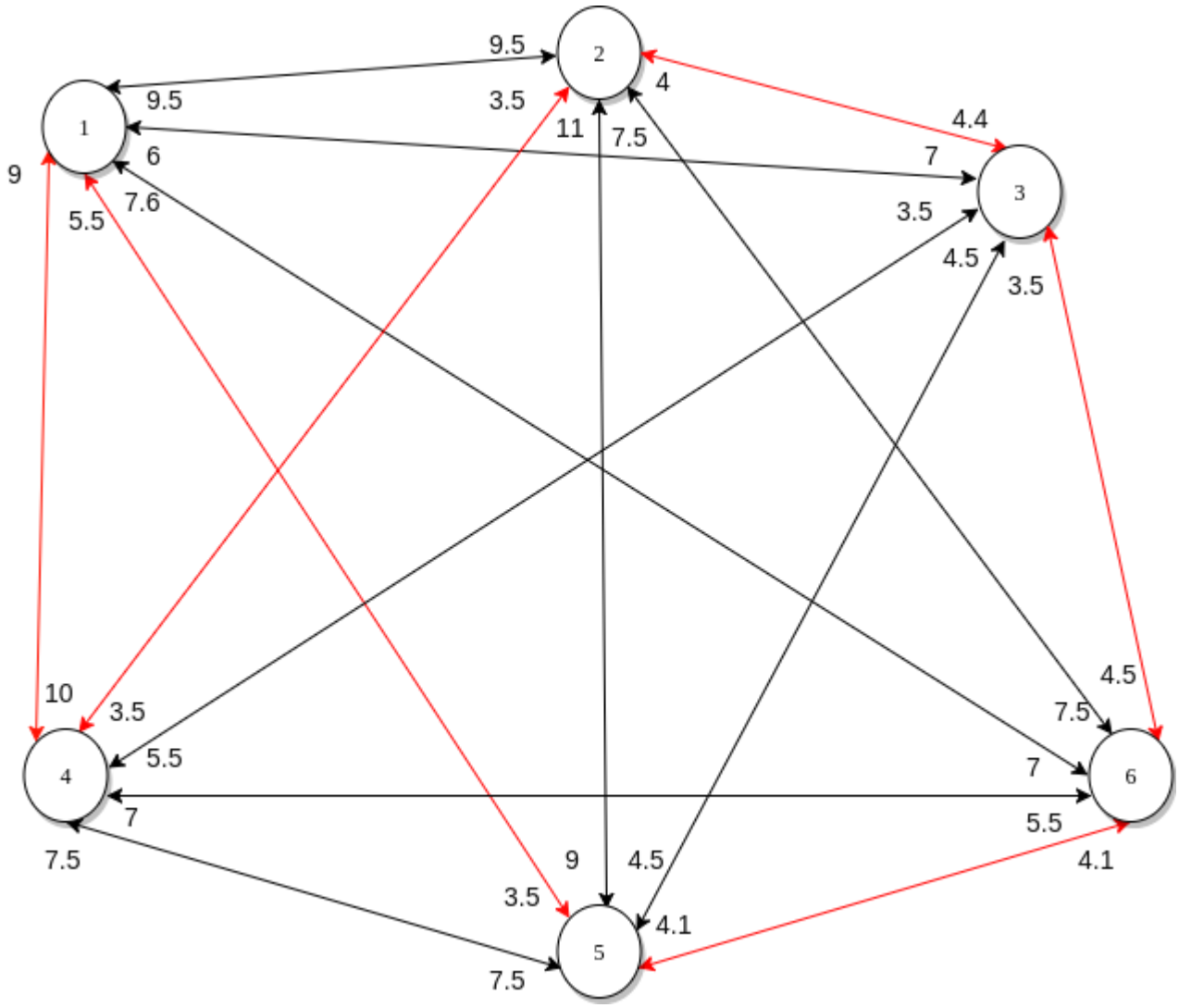
In [67]:
```
# 1 -> 5 -> 6 -> 3 -> 2 -> 4 -> 1
solution = list()

originV   = 1
nextV     = 1
totalDist = 0

markedV = [ 1, 0, 0, 0, 0, 0 ]

def solSolved(markedV):
    for i in markedV:
        if not i:
            return False
    return True

def findMinEdge(selectedV):
    minEdge = ()
    minDist = 100000000000
    for edge in E[selectedV-1]:
        if not markedV[edge[1]-1]:
            dist = distances[edge[0]-1][edge[1]-1]
            if dist < minDist:
                minEdge = edge
                minDist = dist

    global totalDist, nextV
    totalDist = totalDist + minDist
    solution.append(minEdge)
    nextV = minEdge[1]
    markedV[nextV-1] = 1


while not solSolved(markedV):
    findMinEdge(nextV)

markedV[originV-1] = 0
findMinEdge(nextV)


pprint(solution)
pprint(totalDist)
```

```
[(1, 5), (5, 6), (6, 3), (3, 2), (2, 4), (4, 1)]
27.6
```

SE OBTIENE COMO RESULTADO DE LA HEURÍSTICA UN

TotalDistance = 27.6

1 -> 5 -> 6 -> 3 -> 2 -> 4 -> 1



Heurística 2: Seleccionar las ciudades de forma aleatoria

In [122]:
```
import random

# 1 -> 5 -> 6 -> 3 -> 2 -> 4 -> 1
solution = list()

originV   = 1
nextV     = 1
totalDist = 0

markedV = [ 1, 0, 0, 0, 0, 0 ]

def solSolved(markedV):
    for i in markedV:
        if not i:
            return False
    return True

def findMinEdge(selectedV):
    minEdge = ()
    minDist = 100000000000

    edge = random.choice(E[selectedV-1])
    while markedV[edge[1]-1]:
        edge = random.choice(E[selectedV-1])

    dist = distances[edge[0]-1][edge[1]-1]
    if dist < minDist:
        minEdge = edge
        minDist = dist

    global totalDist, nextV
    totalDist = totalDist + minDist
    solution.append(minEdge)
    nextV = minEdge[1]
    markedV[nextV-1] = 1


while not solSolved(markedV):
    findMinEdge(nextV)

markedV[originV-1] = 0
findMinEdge(nextV)


pprint(solution)
pprint(totalDist)
```

```
[(1, 4), (4, 3), (3, 6), (6, 2), (2, 5), (5, 1)]
40.0
```

SE OBTIENE COMO RESULTADO DE LA HEURÍSTICA:

TotalDistance = 40

1 -> 4 -> 3 -> 6 -> 2 -> 5 -> 1