Universidad del Rosario

MACC
Matemáticas Aplicadas y Ciencias de la Computación

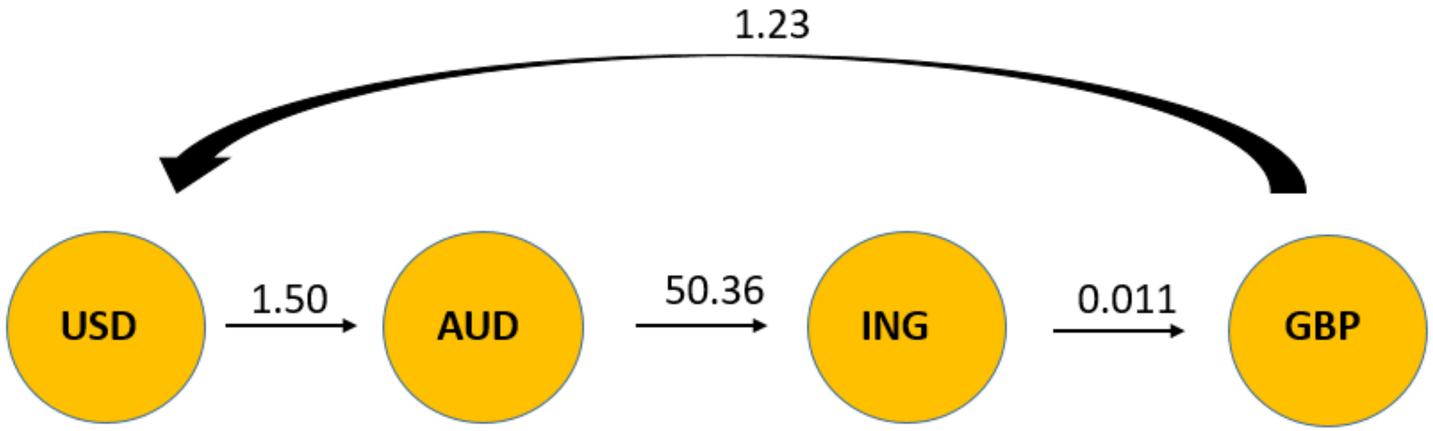
IDENTIFICACIÓN DE ARBITRAJE CON MONEDAS

Juanita Robles y Camila Patarroyo

¿ QUÉ ES EL ARBITRAJE ?

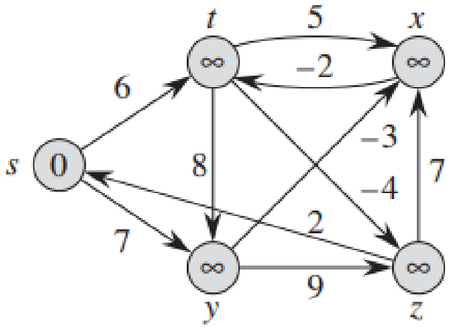
ES LA OPERACION QUE BUSCA OBTENER GANANCIA CON LA DIFERENCIA DE PRECIOS EN DOS MERCADOS DISTINTOS.

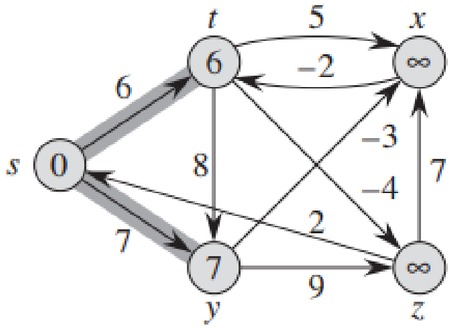
PRESENTACIÓN USANDO GRAFOS DIRIGIDOS

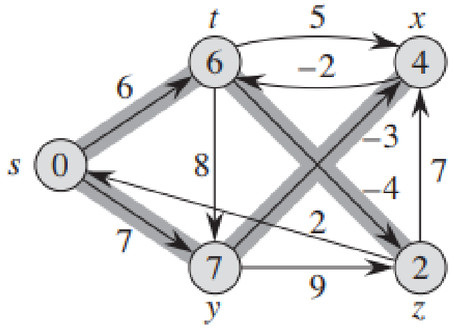


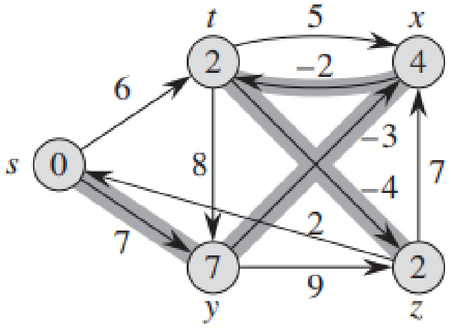
ESTAS OPERACIONES PUEDEN SER REPRESENTADAS USANDO GRAFOS DIRIGIDOS (COLA SERÍA LA MONEDA QUE VENDO Y CABEZA SERÍA LA QUE COMPRO) Y ASIGNANDO PESOS EN LAS ARISTAS QUE REPRESENTAN LOS PRECIOS.

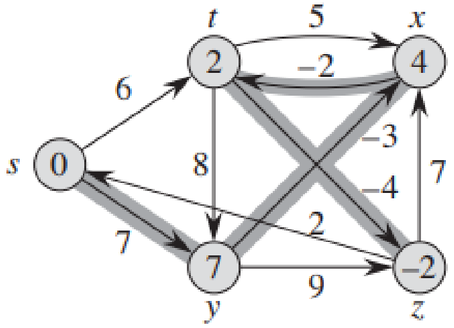
ALGORITMO BELLMAN FORD





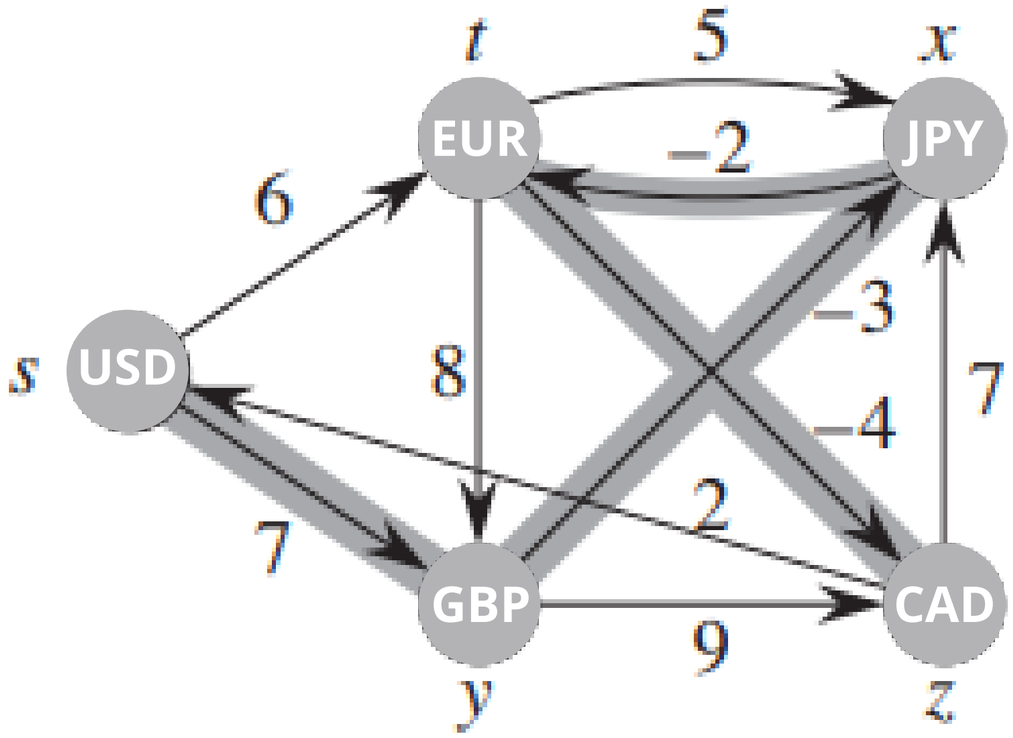







```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3       for each edge (u, v) ∈ G.E
4            RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6       if v.d > u.d + w(u, v)
7            return FALSE
8  return TRUE
```

ALGORITMO BELLMAN FORD EJEMPLO PEQUEÑO




```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

```python
from numpy import genfromtxt
import numpy as np
import math
from numpy import loadtxt

data = loadtxt('ARB.txt')

# Function to print adjacency list representation of a graph
dic = { 0:'USD',1:'EUR',2:'JPY',3:'GBP',4:'CAD',5:'AUD',6:'NZD',7:'CHF',8:'DKK',9:'NOK',10:'SEK'}


class Graph:

    def __init__(self):
        self.V = len(data) # No. of vertices
        self.graph = []

        for u in range(self.V):
            for v in range(self.V):
                if u!=v:
                    self.graph.append([u, v, -math.log(data[u][v])])

    # utility function used to print the solution
    def printArr(self, dist,src):
        print("Vertex Distance from Source ", dic[src] )
        for i in range(self.V):
            print("{0}\t\t{1}".format(dic[i], dist[i]))
        print('\n')
```

```python
def BellmanFord(self):
    # Step 1:
    source = 0
    dist = [float("Inf")] * self.V
    pre = [-1]*self.V
    dist[source] = source
    # Step 2:
    for _ in range(self.V - 1):
        for u, v, w in self.graph:
            if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                dist[v] = dist[u] + w
                pre[v]=u
    # Step 3:
    for u, v, w in self.graph:
        if dist[u] != float("Inf") and dist[u] + w < dist[v]:
            print_cycle = [v,u]
            while pre[u] not in print_cycle:
                print_cycle.append(pre[u])
                u =pre[u]
            print_cycle.append(pre[u])

            if print_cycle[0]==print_cycle[-1]:
                u = 1
                for p in range(len(print_cycle[::-1][:-1])):
                    u = data[print_cycle[::-1][p+1]][print_cycle[::-1][p]] * u
                r =(u/1 -1)*100

                if(r>0 and r<1):
                    print("arbitraje*: ")
                    print(" --> ".join([dic[p] for p in print_cycle[::-1]]))
                    print(r,"%")
            else:
                print_cycle = [print_cycle[-1], *print_cycle]
                u = 1
                for p in range(len(print_cycle[::-1][:-1])):
                    u = data[print_cycle[::-1][p+1]][print_cycle[::-1][p]] * u
                r =(u/1 -1)*100

                if(r>0 and r<1):
                    print("arbitraje: ")
                    print(" --> ".join([dic[p] for p in print_cycle[::-1]]))
                    print(r,"%")

if __name__ == '__main__':

    g = Graph()
    g.BellmanFord()
```

ARB.txt

```
1 1.0532  0.007733  1.247 0.779 0.7012  0.6352  1.0072  0.1416  0.1034  0.100699
0.9493  1 0.73419 1.1839  0.7395  0.6657  0.603 0.9561  0.1344  0.0981  0.0956
129.3 136.18  1 161.236 100.703 90.671  82.129  130.211 18.3013 13.3644 13.0194
0.8017  0.84458 0.6201  1 0.6245  0.56229 0.5093  0.8075  0.1135  0.0829  0.0808
1.2839  1.35221 0.0099265 1.6009  1 0.9003  0.8155  1.2927  0.1817  0.1327  0.1293
1.4257  1.50185 1.1026  1.7782  1.1105  1 0.9058  1.4358  0.2018  0.1474  0.1436
1.5739  1.6579  0.01217 1.9629  1.2259  1.1038  1 1.5855  0.2228  0.1627  0.1585
0.9929  1.0458  0.7678  1.2382  0.7733  0.6963  0.6307  1 14.0528 10.2627 9.9981
7.0644  7.4409  5.4624  8.8094  5.502 4.9536  4.4873  7.1143  1 0.7303  0.71145
9.6707  10.1859 7.4788  12.0587 7.5316  6.78092 6.1418  9.739 1.3686  1 0.9739
9.9264  10.4553 7.6768  12.3776 7.7307  6.9603  6.3042  9.995 1.4047  1.026 1
12
```

EJEMPLO DE TAMAÑO REAL

```
camila@camila:~/Desktop/grafos$ python3 tga.py
arbitraje*:
CHF --> DKK --> USD --> CHF
0.023244735200012023 %
arbitraje*:
DKK --> USD --> DKK
0.031903999999993715 %
arbitraje*:
DKK --> EUR --> DKK
0.0056959999999999479 %
```

CONCLUSIÓN