UVG-MM2015: PROYECTO DE GRAFOS

CARLOS LÓPEZ Y HÉCTOR HURTARTE

Problema 1: Se tienen tres contenedores: de 10, 7 y 4 litros, respectivamente. Los contenedores de 7 y 4 litros están llenos de agua, mientras que el de 10 litros est?a vacío. Tenemos permitido sólo un tipo de operación: verter el agua del contenedor A en el contenedor B, deteniéndonos cuando A esté vacío o cuando B esté lleno.

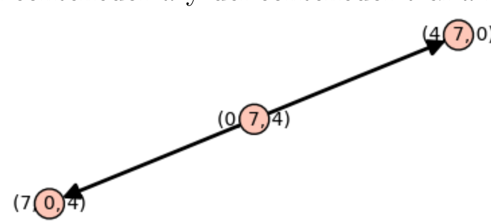
a. Existe alguna secuencia de operaciones que deje exactamente 2 litros en alguno de los contenedores?

Respuesta: Sí

b. Modele este problema como un problema de grafos: proporcione una definición precisa del grafo involucrado y formule la pregunta específica acerca de este grafo que debe ser respondida.

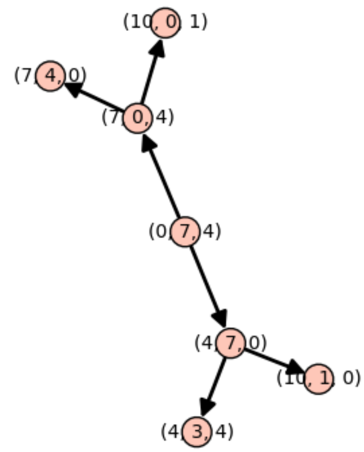
Respuesta: Pensemos en cada vértice v de nuestro grafo G_1, como un "estado" de los contenedores en forma de una tupla (a, b, c), en donde, a representaría la cantidad de agua que tiene el contenedor de 10 litros, b a la cantidad del de 7 litros y c al de 4 litros.

Notemos que en efecto, el primer estado i.e. vértice del sería estaría representado por la tupla $v_1 = (0, 7, 4)$. Llamemos siguientes estados posibles al conjunto de todos los estados a los que se puede llegar a partir de v_1, por inspección vemos que estos son $(7, 0, 4)$ y $(4, 7, 0)$, resultado de vertir el agua del contenedor c al contenedor a y del contenedor b al a respectivamente.



En el digrafo anterior, están representados los tres estados que hemos listado hasta el momento, el inicial $(0, 7, 4)$ que tiene como siguientes estados posibles a $(7,0,4)$ y $(4,7,0)$, como vértices y en donde un enlace representa la transición de un estado a otro.

Podremos entonces, obtener los siguientes estados de $(7,0,4)$ y $(4,7,0)$. Eso es, en un grafo ilustrado..



La pregunta a contestar sería entonces si existe algún vértice i.e. estado que tenga a algun contenedor a, b o c con valor 2.

c. Qué algoritmo debería aplicarse para resolver este problema?

Respuesta: Debería de aplicarse un algoritmo que genere los siguientes estados posibles, dado un estado inicial, este debería obviar a los estados que ya fueron considerados antes. Después, revisar en cada vertice generado, y los generados de este, si existe una tupla cuyo primer elemento sea 2.

Algoritmo para generar los proximos estados posibles

```
def getProximosEstados(lista,estadosAnteriores):
    posibles = []
    posible = copy(lista)

    if (posible[0]<10):    #si cabe mas en 'a'
        # tratar de pasar de b -> a
        posible[0] += posible[1];
```

```
        if (posible[0]>10): #se lleno 'a' antes de vaciar 'c'?
            posible[1] = posible[0] - 10
            posible[0] = 10
        else:             # se vacio b
            posible[1] = 0

        #agregarlo si no existe
        if (estadosAnteriores.count(posible)==0):
            posibles.append(posible)

        # tratar de pasar de c -> a
        posible = copy(lista)
        posible[0] += posible[2];
        if (posible[0]>10): #se lleno 'a' antes de vaciar 'c'?
            posible[2] = posible[0] - 10
            posible[0] = 10
        else:             # se vacio b
            posible[2] = 0

        if (estadosAnteriores.count(posible)==0):
            posibles.append(posible)

    posible = copy(lista)
    #si cabe mas en 'b'
    if (posible[1]<7):
        #print 'cabe en b'
        # tratar de pasar de a -> b
        posible[1] += posible[0];
        if (posible[1]>7): #se lleno 'b' antes de vaciar 'a'?
            posible[0] = posible[1] - 7
            posible[1] = 7
        else:             # se vacio a
            posible[0] = 0

        if (estadosAnteriores.count(posible)==0):
            posibles.append(posible)

        posible = copy(lista)
        # tratar de pasar de c -> b
        posible[1] += posible[2];
        if (posible[1]>7): #se lleno 'b' antes de vaciar 'a'?
            posible[2] = posible[1] - 7
            posible[1] = 7
```

```
        else:             # se vacio a
            posible[2] = 0

        if (estadosAnteriores.count(posible)==0):
            posibles.append(posible)

    posible = copy(lista)
    #si cabe mas en 'c'
    if (posible[2]<4):
        #print 'cabe en c'
        # tratar de pasar de a -> c
        posible[2] += posible[0];
        if (posible[2]>4): #se lleno 'c' antes de vaciar 'a'?
            posible[0] = posible[2] - 4
            posible[2] = 4
        else:             # se vacio a
            posible[0] = 0

        if (estadosAnteriores.count(posible)==0):
            posibles.append(posible)

        posible = copy(lista)
        # tratar de pasar de b -> c
        posible[2] += posible[1];
        if (posible[2]>4): #se lleno 'c' antes de vaciar 'b'?
            posible[1] = posible[2] - 4
            posible[2] = 4
        else:             # se vacio a
            posible[2] = 0

        if (estadosAnteriores.count(posible)==0):
            posibles.append(posible)
    #print 'devolvio posibles: ', posibles
    return posibles
```

Algoritmo para encontrar la respuesta

```
def existeUnoCon2Litros(lista):
    if (type(lista)==list):
        return lista.count(2)
    else:
```

```
        return 0

def encontrar(estados,stackTrace):
    encontro = False

    for estado in estados:
        if (not encontro):
            if (existeUnoCon2Litros(estado)>0):
                global encontro
                encontro = True;
                stackTrace.append(estado)
                print 'encontro, stack trace: ', stackTrace
                return encontro;
                break;

            if (not encontro):
                if (type(estado)==list):
                    # si no existe en el stack trace, mandarlo
                    if (stackTrace.count(estado)==0):
                        stackTrace.append(estado)
                        #print 'mando estado i: ',estado
                        encontrar(getProximosEstados(estado,estados),stackTrace)
                else:
                    #print 'mando estado: ',estados
                    stackTrace.append(estados)
                    encontrar(getProximosEstados(estados,[]),stackTrace)

    return encontro
```

d. Encuentre la respuesta aplicando dicho algoritmo.

Respuesta:

```
encontro, stack trace:  [[0, 7, 4], [7, 0, 4], [10, 0, 1], [3, 7, 1],
[4, 7, 0], [10, 1, 0], [6, 1, 4], [6, 5, 0], [2, 5, 4]]
True
```

Problema 2: Resuelva el problema 107 del Proyecto Euler.

Algoritmo:

```
Grafo = Graph(40)
matriz = ([[100000000,100000000,100000000,427,668,495,377,678,100000000,177,1
for i in range(40):
    for j in range(40):
        Grafo.add_edge(i,j,matriz[i][j])

peso = lambda e: e[2]
peso_minimo = sum([a[2] for a in Grafo.min_spanning_tree(weight_function = pe
peso_maximo = sum([sum([b for b in a if b!=100000000]) for a in matriz])/2
peso_ahorrado = peso_maximo - peso_minimo
peso_ahorrado
```

Respuesta: 259679

Problema 5: Una compañía tiene sucursales en seis ciudades $C1, C2, \ldots, C6$. El costo de transporte de C_i hacia C_j está dado por la entrada (i, j) de la siguiente matriz (donde un costo ∞ significa que no hay ruta directa entre esas ciudades):

$$\begin{pmatrix} 0 & 50 & \infty & 40 & 25 & 10 \\ 50 & 0 & 15 & 20 & \infty & 25 \\ \infty & 15 & 0 & 10 & 20 & \infty \\ 40 & 20 & 10 & 0 & 10 & 25 \\ 25 & \infty & 20 & 10 & 0 & 55 \\ 10 & 25 & \infty & 25 & 55 & 0 \end{pmatrix}$$

La companñía está interesada en preparar una tabla de costos mínimos de transporte entre ciudades. Prepare dicha tabla.

Solución:

Utilizando el algoritmo de Floyd.

```
def FloydWarshal(M):
    if M.is_square():
        n = len(M.columns())
        for k in range (0,n):
            for i in range (0,n):
                for j in range (0,n):
                    fila = copy(M[i])
                    fila[j] = min(M[i][j] , M[i][k] + M[k][j])
                    M[i] = fila
                    #M[i][j] = min (M[i][j] , 40)
```

```
        return M
    else:
        return False


M5 = matrix([[0,50,10000,40,25,10],[50,0,15,20,10000,25],[10000,15,0,10,20,10000],[4

print FloydWarshal(M5)
```

Respuesta:

$$\begin{pmatrix} 0 & 35 & 45 & 35 & 25 & 10 \\ 35 & 0 & 15 & 20 & 30 & 25 \\ 45 & 15 & 0 & 10 & 20 & 35 \\ 35 & 20 & 10 & 0 & 10 & 25 \\ 25 & 30 & 20 & 10 & 0 & 35 \\ 10 & 25 & 35 & 25 & 35 & 0 \end{pmatrix}$$

1