21-6-2020

Ejercicios Evaluación

Algoritmia Y Complejidad

Diego Ortiz Martínez – 53756321P
PROFESORA DEL LABORATORIO: LORENA

Problema 1 (2 puntos).

En la gestión de la red social PiensaQue (www.piensaque.com, donde puedes escuchar qué piensan tus contactos) quieren determinar el grado de conexión entre sus usuarios. El grado de conexión es la cantidad de grupos de usuarios conectados entre sí dividido por el total de usuarios. Si este valor es 1 es que todos que los usuarios están aislados y no contactos comunes y cuánto más pequeño es este valor mayor es el grado de conexión de los usuarios.

Entre los miles de usuarios de PiensaQue se escoge una población aleatoria de cien identificadores para obtener su grado de conexión. Cada uno de esos cien identificadores representa a un usuario, del que se además se tienen sus contactos almacenados en una lista. Por ejemplo, se podría tener el siguiente caso de seis usuarios (en cursiva) y sus contactos (separados por comas):

Antonio: Bea, Carlos, Emma	*Carlos:* Antonio, Emma	*Emma:* Bea
Bea: Emma, Carlos, Antonio	*David:* Fernando	*Fernando:* NINGUNO

Aunque Emma solo tiene por contacto a Bea, a través de ella está relacionada con Carlos y con Antonio. Por otra parte, Fernando y David también tienen una conexión entre ellos (David tiene a Fernando entre sus contactos, aunque el recíproco no se cumpla). Sin embargo, los dos grupos no tienen nexos en común.

En este ejemplo el grado de conexión sería 2 grupos entre 6 usuarios, aproximadamente el valor 0'33.

Diseñar un algoritmo **voraz** que, teniendo como datos los usuarios elegidos (cada uno de ellos con su identificador y la lista de sus contactos), obtenga el grado de conexión entre ellos.

En este primer ejercicio, he visto una oportunidad de resolver este problema como si de un problema de conjuntos se tratara. Es por ello, que una resolución válida del problema sería representar el anterior conjunto de datos en un diccionario, de tal manera que cada llave fuera cada uno de los usuarios presentados, y sus contactos fueran los valores de cada llave. Hasta ahí todo sencillo, ahora queda tratar el problema de conjuntos. Si lo que hacemos es añadir a una lista común una lista que contenga tanto la llave como sus valores, es decir, para el caso del ejemplo quedaría una lista: ['Antonio', 'Bea', 'Carlos', 'Emma'], que se insertaría como el resto de las listas resultantes en una lista común. Por lo tanto, en este paso contaríamos con una estructura como la siguiente:

[['Antonio', 'Bea', 'Carlos', 'Emma'], ['Carlos', 'Antonio', 'Emma'], ['Emma', 'Bea'], ['Bea', 'Emma', 'Carlos', 'Antonio'], ['David', 'Fernando'], ['Fernando']]

Para conseguir que este algoritmo fuera lo más voraz posible, la lista anterior ha sido ordenada por longitud, de tal manera que se vayan tratando los distintos grupos de menor a mayor longitud, cumpliendo con una característica que presentan los Greedy Algorithms.

A partir de aquí, podemos tratar cada sublista como un conjunto. Seguiremos el siguiente procedimiento:

- Mientras siga habiendo subconjuntos, se tomará el primero de ellos y se comparará con cada grupo del resto de ellos.
- Estas listas se convertirán en conjuntos para un recibir un tratamiento más adecuado al problema.
- Si tienen en común al menos un elemento (es decir si la longitud de su intersección es distinta de 0), entonces se conseguirá el set1 actualizado con los valores que no tenga el set1 del set2.
- Por lo contrario, si la intersección proporciona el conjunto vacío, se añadirá a una lista restoGrupos2 que será usada una vez se acabe la iteración de restoGrupos anterior.

De esta manera, se irán actualizando los conjuntos, resultando por tanto en una lista como la siguiente:

[{'Emma', 'Antonio', 'Carlos', 'Bea'}, {'Fernando', 'David'}]

Que es el resultado que deseamos obtener, es decir, los grupos de personas que quedan interconectadas al tenerse como contactos en la aplicación. Por lo tanto, lo único que nos queda hacer para obtener el grado de conexión en la aplicación es dividir el número de grupos de personas que indirectamente están conectadas entre ellas que hemos obtenido (en este caso 2), entre el número total de usuarios de la aplicación (6 personas). De esta manera obtenemos que el grado de conexión en este ejemplo es de 0,33 periódico.

Además, se ha realizado el código correspondiente a la lectura de un fichero con los datos, que mantendrá el siguiente formato:

Usuario1: Contacto1, Contacto2, Contacto3…

Usuario2: Contacto4, Contacto2…

De esta manera se creará el diccionario de usuarios ya comentado anteriormente.

Por último, queda hablar del fichero de salida generado. Este tendrá el nombre de resultado_voraz.txt y tendrá el siguiente formato:

- Número de grupos
- Los grupos de usuarios son:
    - Grupo 1: {grupo de conexión 1}
    - Grupo 2: {grupo de conexión 2}
    - …
- Grado de conexión

Aquí se muestra el código del programa:

```
def ejer1(listaUsuarios):
    grupos = []
    for key in listaUsuarios:
        unidadGrupo = [key]
        for i in range(len(listaUsuarios[key])):
            unidadGrupo.append(listaUsuarios[key][i])
        grupos.append(unidadGrupo)
    grupos_sorted = list(sorted(grupos, key=len))
    resFinal = []
    while len(grupos_sorted) > 0:
        primerElem = grupos_sorted[0]
        restoGrupos = grupos_sorted[1:]
        primerElem = set(primerElem)
        numElemsPrimero = -1
        while len(primerElem) > numElemsPrimero:
            numElemsPrimero = len(primerElem)
            restoGrupos2 = []
            for grupo in restoGrupos:
                if len(primerElem.intersection(set(grupo))) != 0:
                    primerElem.update(grupo)
                else:
                    restoGrupos2.append(grupo)
            restoGrupos = restoGrupos2
        resFinal.append(primerElem)
        grupos_sorted = restoGrupos
    file.write("Numero de grupos: " + str(len(resFinal)) + "\n" + "Los grupos de u
suarios son: \n\n")
    for i in range(len(resFinal)):
        file.write("- Grupo " + str(i + 1) + ": " + str(resFinal[i]) + "\n")
    file.write("\n")
    gradoConexion = len(resFinal) / len(listaUsuarios)
    return gradoConexion


archivo = open("ejemplo_voraz.txt").read()
usuarios = {}
lineasArchivo = archivo.split('\n')
for linea in lineasArchivo:
    if linea != "":
        separacion = linea.split(":")
        key = separacion[0]
        value = []
        for elem in separacion[1].split(","):
            if (elem) != '':
                value.append(elem)
        usuarios[key] = value

f = open("resultado_voraz.txt", "w")
f.write("")
f.close()
file = open("resultado_voraz.txt", "a")
w = ejer1(usuarios)
file.write("Y, por tanto, este es el grado de conexión: " + str(w))
```

Problema 2 (1 punto).

Calcular la nota media de un alumno en un curso formado por n asignaturas usando la técnica **divide y vencerás**, siendo n potencia de 2. Modificar el algoritmo para permitir valores de n que no sean potencia de 2. Calcular la eficiencia y determinar si mejora la que se obtiene respecto a una simple versión iterativa.

En este ejercicio se nos pide calcular la nota media de un alumno mediante la técnica de divide y vencerás. Es cierto que este problema es sencillo siempre y cuando puedas subdividir el problema en 2, es decir, teniendo en este caso un número n de asignaturas que sea potencia de dos. Por el contrario, la cosa se complica cuando tenemos un n que no sea potencia de dos. Y es que la única forma de obtener una media correcta, es decir, una media en la que el denominador sea equivalente para todos los valores y no se dé mayor peso a una parte de la media en vez de a otra; es hacer la suma mediante el algoritmo de divide y vencerás, postergando la división de la media hasta el final del proceso.

Es por ello por lo que se presenta la siguiente solución, con una función específica para cada variante:

En la primera función nos encontramos con que separamos la lista en dos partes y, hasta que quedan únicamente dos elementos. En ese momento extraeremos la media y retornaremos el valor resultante. Se irán retornando los valores hacia fuera, donde se sumarán y se dividirá entre 2 (sacando su media correspondiente), hasta llegar a la primera llamada recursiva, donde se retornará finalmente el valor de la media correcto.

En la segunda función hemos decidido sacar a la división del proceso recursivo. De esta manera, podíamos realizar correctamente un uso de la técnica de divide y vencerás puesto que lo único que se tiene que hacer es “sumar por partes”. De la misma manera que el anterior, al final se retornaría una suma de todos los elementos, que, finalmente, dividiríamos entre el número de elementos, obteniendo la media correspondiente.

Cada función se ejecutará evaluando el número n de asignaturas del alumno (a través de una función *esPotenciaDos* creada al comienzo del código), llamando a la función correspondiente en el momento adecuado.

Se leerá para ello un archivo de entrada que contendrá cada una de las notas de cada asignatura evaluada, dispuestas una por cada línea del fichero. Es decir:

8

7

5

…

Para la salida, se ha seguido el siguiente formato:

- Notas de las asignaturas: [Lista con todas las notas de las asignaturas cursadas]
- El array es/no es potencia de dos
- Valores medios de cada una de las iteraciones(*)
- Resultado final de la media del alumno

(*)Los valores medios en el caso de la primera función en la que n es potencia de dos será el valor de la media obtenido y que se retorna a cada llamada. En el caso de la segunda función, sin embargo, retornará el valor medio de la suma obtenida, pues es lo que especifica el enunciado (aunque este valor no sirva a la hora de conseguir una media ponderada).

Además, se nos pide extraer la eficiencia. Para extraer la eficiencia de una técnica divide y vencerás deberemos extraer estos valores:

- n: tamaño del problema original.
- k: número de subcasos.
- b: constante. El tamaño de los subcasos es n/b
- p: existe un p entero tal que $g(n) \in O(n^p)$, concluyendo que:

$$T(n) = \begin{cases} O(n^p) & k < b^p \\ O(n^p \log n) & k = b^p \\ O(n^{\log_b k}) & k > b^p \end{cases} \quad (*)_1$$

Una vez tengamos estos valores, podemos usar lo siguiente para calcular la eficiencia:

t(n) = k*t(n/b) + g(n) donde t(n) es el tiempo total requerido por el algoritmo divide y vencerás, siempre que n sea lo suficientemente grande.

Algoritmo DyV: n es potencia de dos

k = 2 puesto que son dos las llamadas al método recursivo por cada llamada recursiva.

b = 2 puesto que dividimos el problema en dos a cada llamada

Además tenemos que p = 0 porque $O(n^p) = O(1)$

Si nos fijamos en la figura $(*)_1$ anterior, tendremos que k va a ser mayor que b^p ($2>2^0$), dando como resultado una eficiencia de $O(n^{\log_2(2)})$. Dado que el $\log_2(2) = 1$, tenemos que la eficiencia es de O(n).

Algoritmo DyV: n no es potencia de dos

En este caso nos encontramos con que la eficiencia sería la misma por realizar los mismos cálculos en cuanto a llamadas recursivas y a la división de la llamada como tal.

Algoritmo iterativo

Para recrear una versión iterativa que obtenga los mismos resultados, deberíamos tener una función con un bucle que recorriera cada una de las asignaturas de las que va a sacar una media. Es por ello por lo que obtendríamos una eficiencia de O(n)

Conclusión

Como ya hemos visto, las tres eficiencias son iguales para los trés métodos de resolución utilizados.

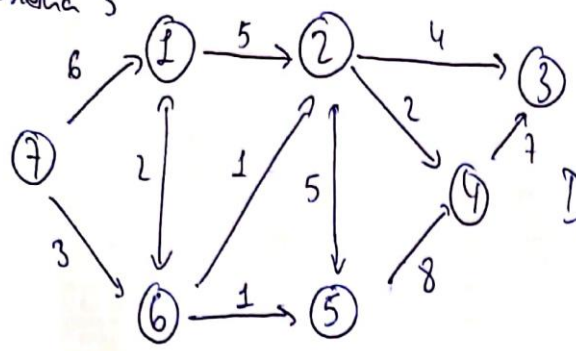
Este es el código resultante:

```
1.  def esPotenciaDos(num):
2.      if num == 0:
3.          return False
4.      while num != 1:
5.          if num % 2 != 0:
6.              return False
7.          num = num // 2
8.      return True
9.
10.
11. def divideYVenceras(listaNotas):
12.     if len(listaNotas) / 2 == 1:
13.         media = (listaNotas[0] + listaNotas[1]) / 2
14.         f.write("Valor medio de " + str(listaNotas[0]) + " " + str(listaNotas[
    1]) + " = " + str(media) + "\n")
15.         return media
16.     else:
17.         n = len(listaNotas)
18.         lista1 = listaNotas[0:int(n / 2)]
19.         lista2 = listaNotas[int(n / 2):n]
20.         media = (divideYVenceras(lista1) + divideYVenceras(lista2)) / 2
21.         f.write("Valor medio de " + str(lista1) + " " + str(lista2) + " = " +
    str(media) + "\n")
22.         return media
23.
24.
25. def divideYVenceras2(listaNotas):
26.     if len(listaNotas) == 1:
27.         f.write("Valor medio de " + str(listaNotas[0]) + " = " + str(listaNota
    s[0]) + "\n")
28.         return listaNotas[0]
29.     elif len(listaNotas) == 2:
30.         suma = listaNotas[0] + listaNotas[1]
31.         f.write("Valor medio de " + str(listaNotas[0]) + " " + str(listaNotas[
    1]) + " = " + str((listaNotas[0] + listaNotas[1])/2) + "\n")
32.         return suma
33.     else:
34.         n = len(listaNotas)
35.         lista1 = listaNotas[0:int(n / 2)]
36.         lista2 = listaNotas[int(n / 2):n]
37.         sumaTotal = divideYVenceras2(lista1) + divideYVenceras2(lista2)
38.         f.write("Valor medio de " + str(lista1) + " " + str(lista2) + " = " +
    str(sumaTotal/n) + "\n")
39.         return sumaTotal
40.
```

```
41. 
42. 
43. archivo = open("ejemplo_dyv.txt").read()
44. notasAsignaturas = []
45. lineasArchivo = archivo.split('\n')
46. for linea in lineasArchivo:
47.     if linea != "":
48.         notasAsignaturas.append(int(linea))
49. f = open("resultado_dyv.txt", "w")
50. f.write("Notas de las asignaturas: " + str(notasAsignaturas) + "\n")
51. f.close()
52. if esPotenciaDos(len(notasAsignaturas)):
53.     f = f = open("resultado_dyv.txt", "a")
54.     f.write("El tamaño del array es potencia de dos\n")
55.     w = divideYVenceras(notasAsignaturas)
56.     f.write("--> Resultado = " + str(w))
57. else:
58.     f = open("resultado_dyv.txt", "a")
59.     f.write("El tamaño del array no es potencia de dos\n")
60.     l = len(notasAsignaturas)
61.     w = divideYVenceras2(notasAsignaturas)
62.     f.write(str(w) + "/" + str(l) + "--> Resultado = " + str(w/l))
```

Problema 3

Diego Ortiz Martínez
53756321P



$D_0 = L =$

0	5	∞	∞	∞	2	∞
∞	0	4	2	5	∞	∞
∞	∞	0	∞	∞	∞	∞
∞	∞	7	0	∞	∞	∞
∞	5	∞	8	0	∞	∞
2	1	∞	∞	1	0	∞
6	∞	∞	∞	∞	3	0

$D_1 =$

0	5	∞	∞	∞	2	∞
∞	0	4	2	5	∞	∞
∞	∞	0	∞	∞	∞	∞
∞	∞	7	0	∞	∞	∞
∞	5	∞	8	0	∞	∞
2	1	∞	∞	1	0	∞
6	11	∞	∞	∞	3	0

$D_2 =$

0	5	9	7	10	2	∞
∞	0	4	2	5	∞	∞
∞	∞	0	∞	∞	∞	∞
∞	∞	7	0	∞	∞	∞
∞	5	9	7	0	∞	∞
2	1	5	3	1	0	∞
6	11	15	13	16	3	0

$D_3 =$

0	5	9	7	10	2	∞
∞	0	4	2	5	∞	∞
∞	∞	0	∞	∞	∞	∞
∞	∞	7	0	∞	∞	∞
∞	5	9	7	0	∞	∞
2	1	5	3	1	0	∞
6	11	15	13	16	3	0

$D_4 =$

0	5	9	7	10	2	∞
∞	0	4	2	5	∞	∞
∞	∞	0	∞	∞	∞	∞
∞	∞	7	0	∞	∞	∞
∞	5	9	7	0	∞	∞
2	1	5	3	1	0	∞
6	11	15	13	16	3	0

$D_5 =$

0	5	9	7	10	2	∞
∞	0	4	2	5	∞	∞
∞	∞	0	∞	∞	∞	∞
∞	∞	7	0	∞	∞	∞
∞	5	9	7	0	∞	∞
2	1	5	3	1	0	∞
6	11	15	13	16	3	0

$D_6 =$

0	3	7	5	3	2	∞
∞	0	4	2	5	∞	∞
∞	∞	0	∞	∞	∞	∞
∞	∞	7	0	∞	∞	∞
∞	5	9	7	0	∞	∞
2	1	5	3	1	0	∞
5	4	8	6	4	3	0

$D_7 =$

0	3	7	5	3	2	∞
∞	0	4	2	5	∞	∞
∞	∞	0	∞	∞	∞	∞
∞	∞	7	0	∞	∞	∞
∞	5	9	7	0	∞	∞
2	1	5	3	1	0	∞
5	4	8	6	4	3	0

Siendo D_7 la matriz de distancias mínimas del grafo.

Problema 4 (2 puntos).

Bruce Springsteen va a terminar su gira mundial de este año dando el último concierto en Abecelandia, ciudad famosa por sus bellas plazas y que puede que conozcas, por lo que un camión cargado de instrumentos, aparatos de sonido, bengalas, soportes y tuercas ha entrado en la ciudad y se dirige al estadio para montar el escenario y los fuegos artificiales. Repentinamente, una de las ruedas delanteras explota y deja parcialmente inutilizado el sistema de dirección del camión, impidiéndole realizar giros hacia la izquierda. Debido a su tamaño y peso, el camión no puede ir marcha atrás (así que no puede retroceder) ni cambiar de sentido en la misma calle (es decir, no puede dar un giro de 180° sin cambiar de calle) así que, de momento, el camión solo puede circular en línea recta y realizar giros de 90° a la derecha. Afortunadamente el conductor dispone de un plano de Abecelandia como el siguiente, donde el punto 1 es la posición actual del camión, el punto 2 es el estadio al que debe ir, y cada posición está marcada como circulable (si está vacía) o no circulable (si está en negro). Puede suponerse que inicialmente el camión está orientado hacia la izquierda, y que el mapa tiene tamaño FxC. Por tanto, en cada posición del mapa el camión puede continuar avanzando (suponiendo que la siguiente casilla esté libre) o girar 90° a la derecha (una vez más, suponiendo que esa posición sea circulable). Como no se sabe si el resto de ruedas aguantarán hasta llegar al estadio, se quiere reducir al máximo la cantidad de giros para no sobrecargar demasiado los neumáticos que aún funcionan. Implementar un algoritmo con metodología **Backtracking**, eficiente en la medida de lo posible, que teniendo como datos los tamaños en filas F y columnas C del mapa, el mapa MFxC y las coordenadas de los puntos 1 y 2, encuentre el camino que lleva de 1 a 2 con la menor cantidad de giros hacia la derecha.

En este ejercicio se nos pide, sobre un mapa dado, llegar desde un punto 1 a un punto 2, pudiendo únicamente atravesar aquellas casillas que están marcadas inicialmente con un 0. Existirán casillas que no se pueden recorrer, que quedarán marcadas con una X. Además, se deberá tener en cuenta que la posibilidad de movimientos se reduce a seguir de frente o girar a la derecha. Esto resulta en que hemos de tener en cuenta a cada paso de la resolución la orientación actual y la orientación a la que se va a cambiar.

En este ejercicio nos encontramos con que hemos de usar la técnica de backtracking, de tal forma que iremos generando un árbol cuyas ramas abrirán nuevas y únicas formas de alcanzar (o no) una solución. De esta manera, llamaremos al método de backtracking con los siguientes parámetros:

- Una matriz correspondiente al mapa que se está explorando.
- El ancho y el largo de dicha matriz.
- Los movimientos disponibles dada una orientación e indicando el resultado de efectuarse.
- La posición de inicio con sus coordenadas X e Y correspondientes y la orientación inicial
- Número de giros realizados
- El número mínimo de giros realizados (para obtener la solución óptima)
- La posición final, es decir, el destino.

De esta manera, cada vez que llamemos al método se seguirá el siguiente esquema:

- Si la casilla actual es el destino:
    - Se añadirán las coordenadas de la casilla final, además de la orientación con la que se llega hasta ella.
    - Se añadirá a una lista de soluciones una lista que contiene el número de giros realizados hasta encontrar la solución y todas las casillas recorridas (cada una con sus coordenadas X e Y, y su orientación).
- Si no lo es entonces:
    - Se comprobará que no se haya pasado por la misma casilla más de tres veces. Este número es una cota variable que se ha añadido como método de poda frente a bucles infinitos que puedan producirse a la hora de recorrer el mapa.
    - Posteriormente, si no se ha recorrido más de 3 veces, el programa podrá continuar, evaluando la posibilidad de visitar la casilla primero siguiendo de frente, y posteriormente la casilla que queda al girar a la derecha (dada la dirección actual), añadiendo un giro en este último caso. Si la posición es válida, entonces se marcará la casilla visitada con un asterisco, indicando que ha sido recorrida. Se actualizará además el número de veces por el que se ha pasado por dicha casilla, y se añadirá dicha posición al camino actual.
    - En el caso de que se retorne falso en alguna llamada, el árbol volverá para atrás, efectuando cambios en el mapa (cambiando las posiciones visitadas por no visitadas), sacando del camino esa posición que no ha resultado en una solución finalmente, y restando uno al valor de las veces que se ha recorrido dicha casilla.

De esta manera, se irá creando un árbol donde se irán recorriendo las ramas añadiendo todas las soluciones posibles proporcionada la cota anteriormente comentada. Además, se ha tenido que crear un método *convertirCaminoTupla* para solucionar la sobreescritura de los elementos dada una lista (puesto que las listas en Python son mutables y las tuplas son inmutables).

Sin embargo, se ha usado la mutabilidad de las listas a mi favor, dado que se ha llegado a necesitar contadores globales como pueden ser el mínimo de giros o incluso la posición final de destino (puesto que dependiendo del ejemplo se podría llegar a ella desde distintas orientaciones).

Además, el enunciado requería obtener los datos de un fichero txt de entrada. Como bien indica, se ha seguido el siguiente formato:

- Número de filas (F).
- Número de columnas (C).
- Descripción de la primera fila (C caracteres separados por tabulaciones).
- Descripción de la segunda fila (C caracteres separados por tabulaciones).
- …
- Descripción de la última fila (C caracteres separados por
- tabulaciones).

Por último, el enunciado requería también crear un archivo de salida que contuviera los datos precisados. Este es el formato del fichero de salida:

- Solución (númeroSolución).
- Mapa del recorrido realizado por la solución.
- Casillas recorridas desde la inicial a la final.
- En el caso de tener el mínimo de giros, se añadirá que es una solución óptima.

Esto por cada una de las soluciones encontradas establecida la cota.

Este es el código resultante:

```
1.  def camionBacktracking(mapa, numFilas, numColumnas, movimientos, actX, actY, o
    rientacion, giros, minGiros, posFinal):
2.      if (mapa[actX][actY] == '2'):
3.          posFinal.append(actX)
4.          posFinal.append(actY)
5.          posFinal.append(orientacion)
6.          if minGiros[0] == -1:
7.              minGiros[0] = giros
8.          else:
9.              if giros < minGiros[0]:
10.                 minGiros[0] = giros
11.         datos = []
12.         datos.append(giros)
13.         caminosTupla = convertirCaminoTupla(caminos)
14.         datos.append(caminosTupla)
15.         sols.append(datos)
16.         return True
17.     else:
18.         for value in list(conteoCasillas.values()):
19.             if value > 3:
20.                 return False
21.         for i in range(2):
22.             testOrientacion = movimientos[orientacion][i][0]
23.             testX = actX + movimientos[orientacion][i][1]
24.             testY = actY + movimientos[orientacion][i][2]
25.             if i == 1:
26.                 giros = giros + 1
27.             if (0 <= testX < numColumnas and 0 <= testY < numFilas and (
28.                     mapa[testX][testY] == '0' or mapa[testX][testY] == '*' or
    mapa[testX][testY] == '2')):
29.                 if mapa[testX][testY] != "2":
30.                     mapa[testX][testY] = '*'
31.                     posicion = [testX, testY, testOrientacion]
32.                     caminos.append(posicion)
33.                     casillaKey = str(testX) + str(testY)
34.                     if casillaKey not in conteoCasillas:
35.                         conteoCasillas[casillaKey] = 1
36.                     else:
37.                         conteoCasillas[casillaKey] = conteoCasillas.get(casill
    aKey) + 1
38.                 if not camionBacktracking(mapa, numFilas, numColumnas, movimie
    ntos, testX, testY, testOrientacion,
39.                                           giros, minGiros, posFinal):
40.                     casillaKey = str(testX) + str(testY)
41.                     mapa[testX][testY] = '0'
42.                     if len(caminos) != 0:
43.                         caminos.pop()
44.                     if casillaKey in conteoCasillas:
```

```
                        conteoCasillas[casillaKey] = conteoCasillas.get(casillaKey) - 1
    return False


def convertirCaminoTupla(caminos):
    tupla = []
    for i in range(len(caminos)):
        lineaTupla = []
        for j in range(3):
            lineaTupla.append(caminos[i][j])
        tupla.append(lineaTupla)
    tuplafinal = tuple(tupla)
    return tuplafinal


archivo = open("ejemplo_backtracking.txt").read()
lista = archivo.split('\n')
lado1 = int(lista[1][0])
lado2 = int(lista[0][0])
matriz = []
for i in range(2, lado2 + 2):
    if lista != "":
        linea = lista[i].split()
        lineaMatriz = []
        for elem in linea:
            lineaMatriz.append(elem)
        matriz.append(lineaMatriz)
for i in range(len(matriz)):
    for j in range(len(matriz[i])):
        if matriz[i][j] == '1':
            posInicioX = i
            posInicioY = j
movimientosXY = {
    'u': [['u', -1, 0], ['r', 0, 1]],
    'r': [['r', 0, 1], ['d', 1, 0]],
    'd': [['d', 1, 0], ['l', 0, -1]],
    'l': [['l', 0, -1], ['u', -1, 0]]
}
```

```
posFinal = []
minGiros = [-1]
sols = []
caminos = []
conteoCasillas = {}
dirIni = 'l'
f = open("resultado_backtracking.txt", "w")
f.write("")
f.close()
res = camionBacktracking(matriz, lado1, lado2, movimientosXY, posInicioX, posInicioY, dirIni, 0, minGiros, posFinal)
f = f = open("resultado_backtracking.txt", "a")
for i in range(len(sols)):
    f.write("Solución " + str(i + 1) + "\n")
    mapaTupla = tuple(matriz)
    for j in range(lado2):
        for k in range(lado1):
            for elem in sols[i][1]:
                        if j == elem[0] and k == elem[1]:
                            mapaTupla[j][k] = '*'
          for l in range(lado2):
              for m in range(lado1):
                  f.write(str(mapaTupla[l][m]) + " ")
              f.write("\n")
          f.write("[" + str(posInicioX) + ", " + str(posInicioY) + ", " + dirIni + "], ")
          for w in range(len(sols[i][1]) - 1):
              f.write("[" + str(sols[i][1][w][0]) + ", " + str(sols[i][1][w][1]) + ", " + sols[i][1][w + 1][2] + "], ")
          f.write("[" + str(sols[i][1][len(sols[i][1]) - 1][0]) + ", " + str(sols[i][1][len(sols[i][1]) - 1][1]) + ", " + posFinal[2 + (3 * i)] + "], [" + str(posFinal[0 + (3 * i)]) + ", " + str(posFinal[1 + (3 * i)]) + ", FINISHED]")
          if sols[i][0] == minGiros[0]:
              f.write("\nEsta solución es óptima dado que tiene " + str(minGiros[0]) + " giros")
          f.write("\n\n")
```

Problema 5

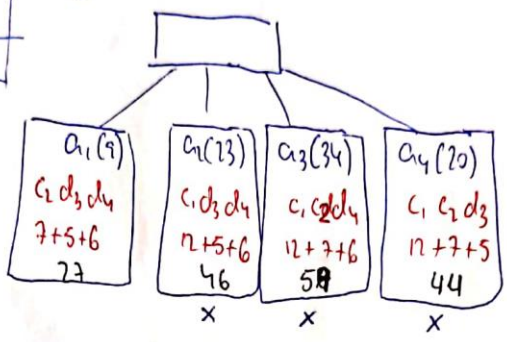
Diop Olfit Martínez
53756321P

	1	2	3	4
a	9	23	34	20
b	21	22	11	7
c	12	7	6	32
d	15	34	5	6

Cota real - es escogemos, por ejemplo:

$a_1 + b_1 + c_3 + d_4 = 43$

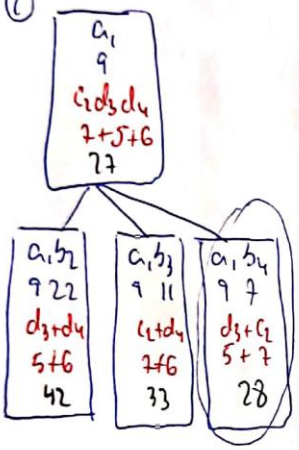
①



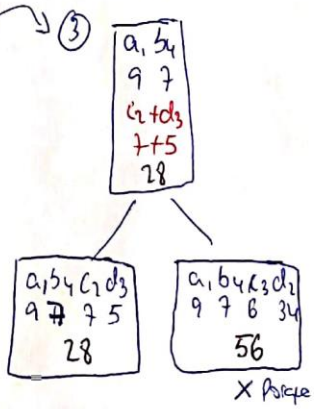
Estas 3 ramas de la derecha quedan cerradas dado que tenemos un ejemplo (cota real), que es inferior al coste (mínimo) de cada rama:

$43 < 46$
$43 < 59$
$43 < 44$

②

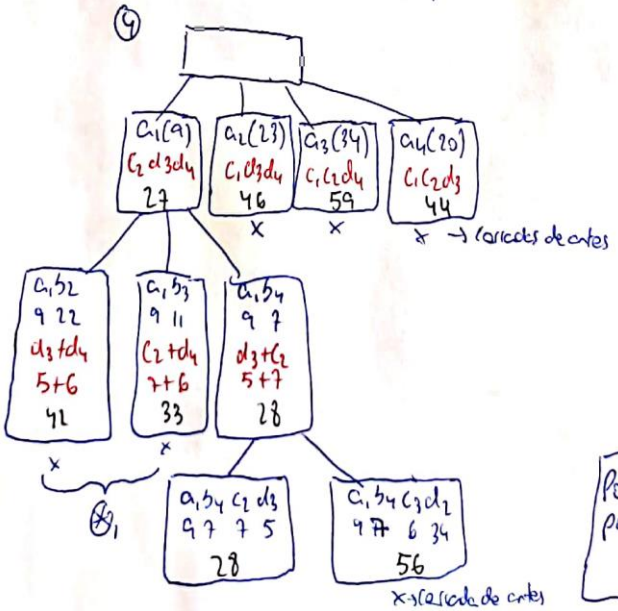


③



Como ahora tenemos un ejemplo que da 28, deberemos revisar las ramas anteriores, o, podarlas en caso de ser necesario.

④



⊗1 Estas ramas no hace falta explorarlas los puestos que tenemos que $a_1 b_4 c_2 d_3 = 28$ y $28 < 42$ y $28 < 33$

Por lo tanto la solución correcta para la asignación es

$a_1 b_4 c_2 d_3 = 28$