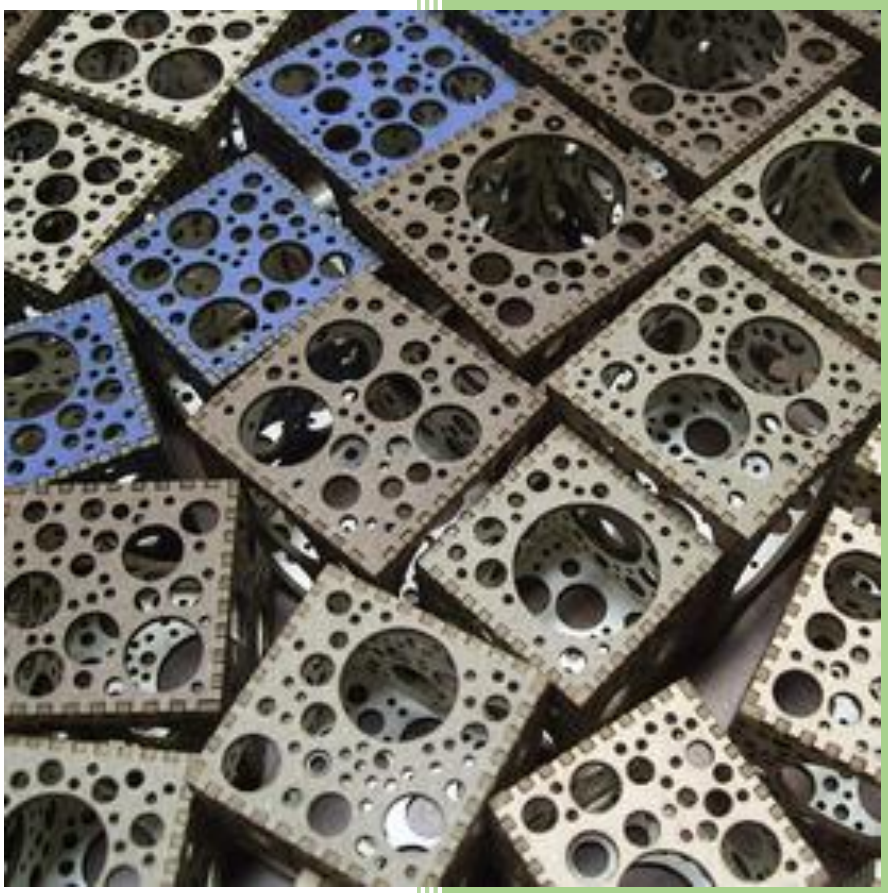
EJERCICIOS TEORIA



David Márquez Mínguez

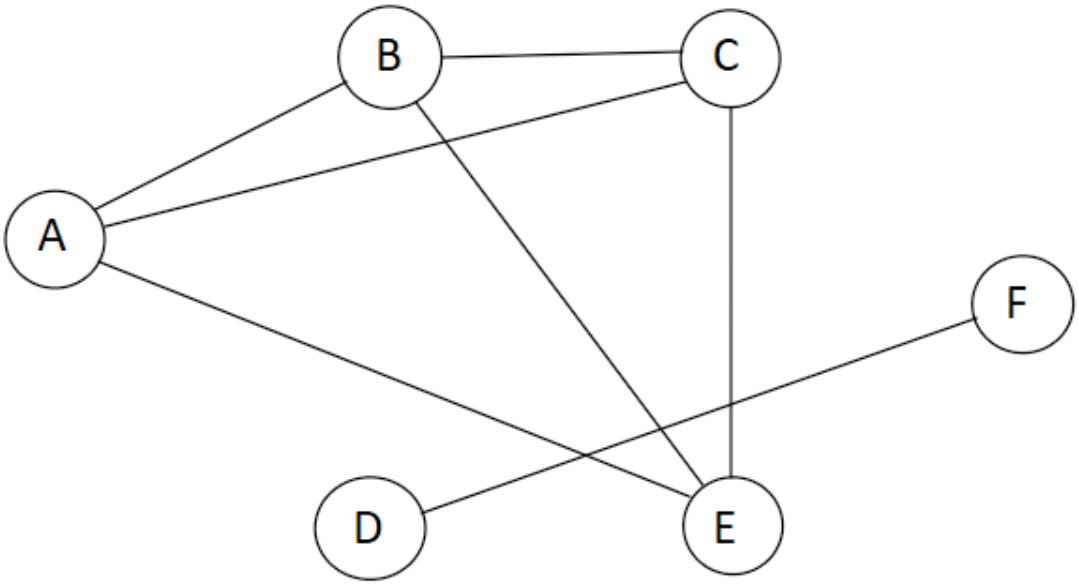
47319570Z

Problema 1

En este ejercicio se pide desarrollar un programa que determine el grado de conexión de un determinado grupo de personas. Se dispone de una tabla como la siguiente en la que podemos observar a 6 usuarios y las conexiones existentes entre ellos:

Antonio	Carlos	Emma	Bea	David	Fernando
Bea Carlos Emma	Antonio Emma	Bea	Emma Carlos Antonio	Fernando	

Como se puede observar disponemos de seis personas y de los contactos de los que disponen. Aunque la conexión entre dos personas no sea reciproca, esta debe estar presente. Por ejemplo David tiene como contacto a Fernando, pero este no le tiene en su lista, aunque la conexión no sea reciproca esta debe estar presente. Por otro lado aunque Emma solo tiene como contacto a Bea, a través de ella esta relacionada con Carlos y con Antonio.
Se pide determinar el grado de conexión entre un grupo de personas a partir de sus contactos, el grado de conexión se determina a partir del numero de grupos formados entre el numero de personas. Por ello si el grado de conexión es cercano a uno, quiere decir que la población de la muestra está muy dispersa, por el contrario cuanto mas se acerque el grado de conexión a cero más conexión habrá entre ellos.
A continuación se muestra un grafo que representa las conexión del grupo de personas anterior, con el objetivo de visualizar de manera más fácil los grupos formados. En el grado los nodos representan a las personas y las aristas la conexión existente. Cada persona se representa por la inicial de su nombre, siendo Emma la letra “E”.



En este grafo se puede observar de manera clara que disponemos de dos grupos, un grupo comprendido por Antonio, Bea, Carlos y Emma, y otro grupo comprendido por Fernando y David. En este ejemplo disponemos de dos grupo para seis personas por lo que el grado de conexión seria:

2 grupos / 6 personas = 0.33

Se pide desarrollar un algoritmo voraz capaz de que a partir de una entrada como la del ejemplo, determine el grado de conexión entre los usuarios.

La solución es muy simple, en primer lugar creare una lista de adyacencia con los datos de conexión que ya se conocen. A continuación creare una lista que determinara los grupos finales donde iré almacenando a los usuarios. Los iré añadiendo de la siguiente manera, por ejemplo, si Antonio no se encuentra dentro de la lista de grupos, le añado a el y a todas sus conexiones. Realizo el mismo proceso para todos los usuarios, para verlo mejor voy a realizar las iteraciones:
En primer lugar la lista de adyacencia será la siguiente:

Antonio	Bea	Carlos	David	Emma	Fernando
[a, b, c, e]	[b, e, c, a]	[c, a, e]	[d, f]	[e, b]	[f]

Como podemos observar es una lista doblemente enlazada donde en cada posición se encuentra la lista de conexiones de cada usuario. Por otro lado tengo la lista de grupos que inicialmente estará vacía.
El primer paso del algoritmo será comprobar si Antonio se encuentra en la lista de grupos, como Antonio no se encuentra, le añado a él y a todas sus conexiones.

[a, b, c, e]

Una vez he añadido a Antonio sigo recorriendo la lista de adyacencia, la siguiente iteración debo analizar a Bea. Bea ya se encuentra en la lista de grupos, por lo que añado a sus conexiones, pero no en una nueva posición de la lista, si no en la posición en la que ya se encuentra.

[a, b, c, e]

Una vez he añadido las conexiones de Bea, sigo iterando y analizo a Carlos. Carlos ya esta en la lista de grupos, por ello añado a todas sus conexiones en la misma posición en la que ya se encuentra(pos-0).

[a, b, c, e]

En la siguiente iteración analizo a David el cual no se encuentra en la lista de grupos, por ello, le añado en una nueva posición junto con todas sus conexiones.

[a, b, c, e]	[d, f]

Esto se repite de forma definida hasta que la lista de adyacencia llega a su fin. La lista de grupos finales es la siguiente:

[a, b, c, e]	[d, f]

Se puede observar que se han creado dos grupos de las seis personas totales por lo que el grado de conexión de este grupo de personas es 2/6 = 0.33.
A continuación se muestra el algoritmo implementado en Java así como la salida de código para el ejemplo mostrado anteriormente:

```java
public static ArrayList crearGrupos(ArrayList<ArrayList> listaAdyacencia) {
    ArrayList<ArrayList> grupos = new ArrayList<>();
    for (int i = 0; i < listaAdyacencia.size(); i++) {
        int pos = estaEnLista(grupos, listaAdyacencia.get(i).get(0).toString());
        if (pos == -1) { //No esta
            grupos.add(listaAdyacencia.get(i));
        } else { //Si esta
            grupos.get(pos).addAll(listaAdyacencia.get(i));
            eliminarRepetidos(grupos.get(pos));
        }
    }

    return grupos;
}

public static int estaEnLista(ArrayList<ArrayList> grupos, String letra) {
    int pos = -1;
    for (int i = 0; i < grupos.size(); i++) {
        for (int j = 0; j < grupos.get(i).size(); j++) {
            if (grupos.get(i).get(j) == letra) {
                pos = i;
                return pos;
            }
        }
    }
    return pos;
}
```

```
run:
[[a, b, c, e], [b, e, c, a], [c, a, e], [d, f], [e, b, a], [f]]
[[a, b, c, e], [d, f]]
El grado de conexion de este grupo de personas es: 0.33333334
BUILD SUCCESSFUL (total time: 7 seconds)
```

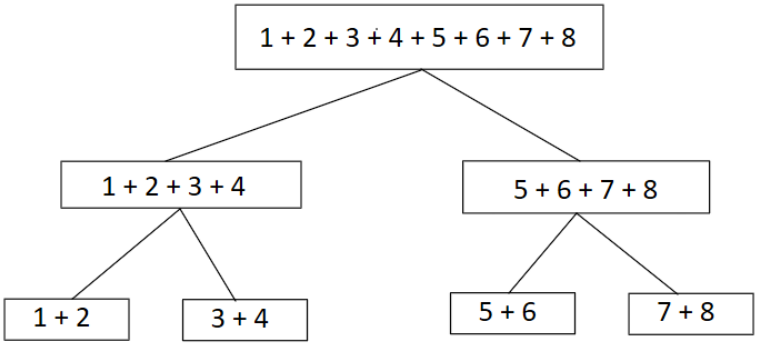
Por otro lado se añaden en el directorio del programa los ficheros de entrada y salida, tanto “ejemplo_voraz.txt” como “resultado.txt” respectivamente. Se añade también un archivo ejecutable del programa que se puede correr en cualquier entorno windows.

Problema 2

En este problema se pide calcular la nota media de un alumno aplicando el algoritmo de divide y vencerás. La nota final consta del cálculo de la media de n asignaturas distintas. En primer lugar resuelvo el problema para cuando n es potencia de dos.

nota 1	nota 2	nota 3	nota 4	nota 5	nota 6	nota 7	nota 8
1	2	3	4	5	6	7	8

El problema inicial se subdividirá el pequeños problemas, tal y como se muestra en el siguiente árbol:



Una vez que el problema ha sido descompuesto en varios casos, cada uno de ellos se resuelve de forma independiente y al final se combinan todos ellos para construir la solución final. El algoritmo empleado es el siguiente:

```
public static float mediaDivYVenc(int inicio, int fin, ArrayList arrayInicial) {
    if (inicio == fin) {
        return (float) arrayInicial.get(inicio);
    } else {
        int mitad = (inicio + fin) / 2;
        float x = mediaDivYVenc(inicio, mitad, arrayInicial);
        float y = mediaDivYVenc(mitad+1, fin, arrayInicial);

        return (y + x) / 2;
    }

}
```

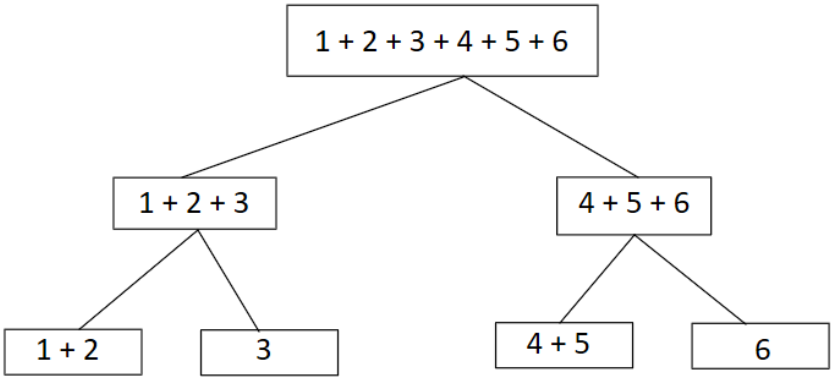
Hasta ahora el algoritmo es sencillo y funciona correctamente, divide el problema principal, en subproblemas cuyo tamaño es potencia de n, y cada subproblema lo soluciona de manera independiente. Por ejemplo para un conjunto de notas {1, 2, 3, 4, 5, 6, 7, 8} la solución debería ser 4,5.

```
El array es potencia de dos
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
4.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

El problema surge cuando el tamaño del array de notas no es potencia de dos, donde una de las ramas se queda descolgada, como en el siguiente ejemplo:

Se puede observar que si el array de notas de un alumno no es potencia de dos, el algoritmo no realizaría bien su trabajo y devolvería valores que no son correctos. Por ejemplo en el siguiente caso, la media seria 3.5 y el algoritmo desarrollado hasta ahora devuelve 3.75.

nota 1	nota 2	nota 3	nota 4	nota 5	nota 6
1	2	3	4	5	6



La solución que se propone es la siguiente, si nos fijamos detenidamente en el algoritmo el problema que existe al dividir el problemas en varios subproblemas, es el tema de las divisiones.

En los nodos donde se suman elementos, estos tras ser sumados se dividen entre dos, lo que hace que los elementos que están descolgados produzcan fallo. ¿Por qué no subdividir el problema en solo sumas, y realizar la división al final?:

Se muestra a continuación el algoritmo implementado. El algoritmo es muy similar al anterior, pero en este caso como se ha mencionado antes, la subdivisiones del problema solo consiste en la suma de elementos, haciendo que la división de los mismos solo ocurra al final.

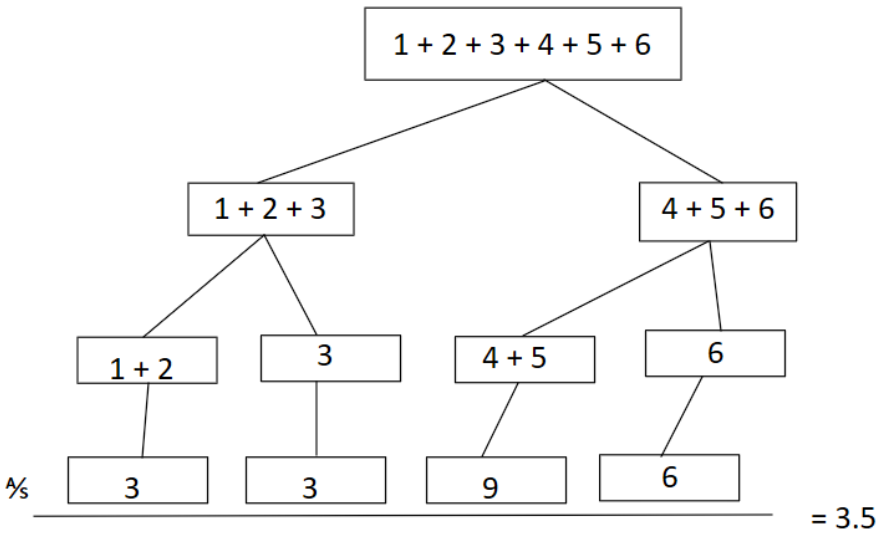
```
public static float mediaDivYVencNoPot2(int inicio, int fin, ArrayList arrayInicial) {
    if (inicio == fin) {
        return (float) arrayInicial.get(inicio);
    } else {
        int mitad = (inicio + fin) / 2;
        float x = mediaDivYVencNoPot2(inicio, mitad, arrayInicial);
        float y = mediaDivYVencNoPot2(mitad + 1, fin, arrayInicial);

        return x + y;
    }

}
```

Se muestra a continuación la salida del programa al ejemplo anterior.

```
run:
El array no es potencia de dos
[1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
3.5
BUILD SUCCESSFUL (total time: 0 seconds)
```



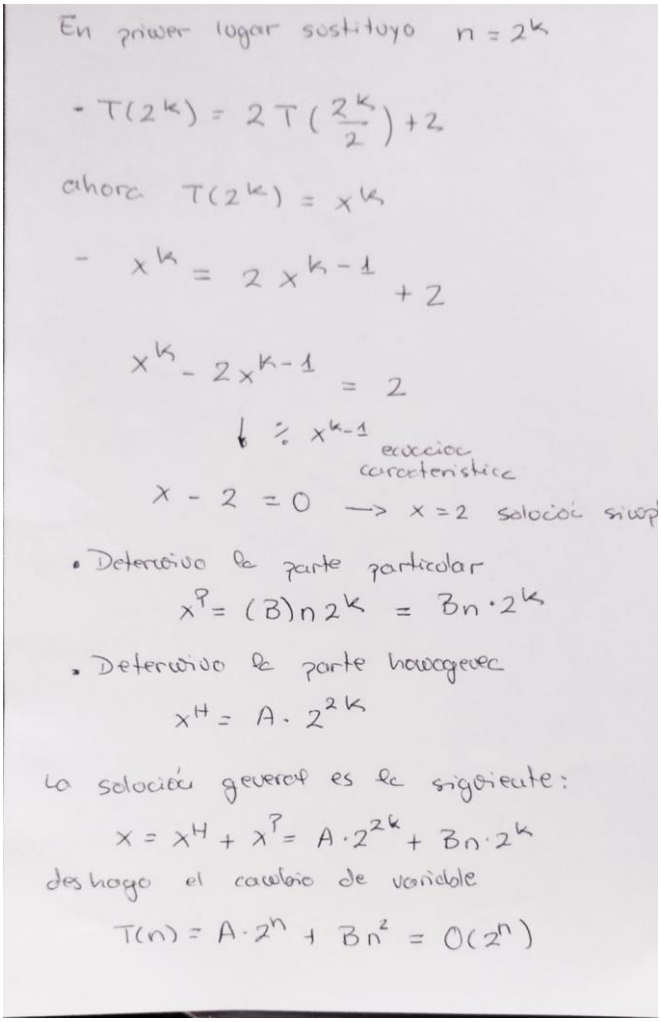
A continuación realizare un análisis de eficiencia del algoritmo creado para realizar la media. En primer lugar nos encontramos una sentencia if, else en la que la parte del if tiene como coste 1 y la parte del else tiene como coste 2T(n/2) + 2. La ecuación seria la siguiente:

$$T(n) = max(1, 2T(n/2) + 2)$$

Ahora realizo las sustituciones para obtener una ecuación recursiva y resolverla para obtener la complejidad del algoritmo. Se muestra en la imagen la resolución del cálculo de la complejidad del mismo. Por otro lado a continuación se muestra el algoritmo capaz de realizar la media aritmética de forma iterativa:

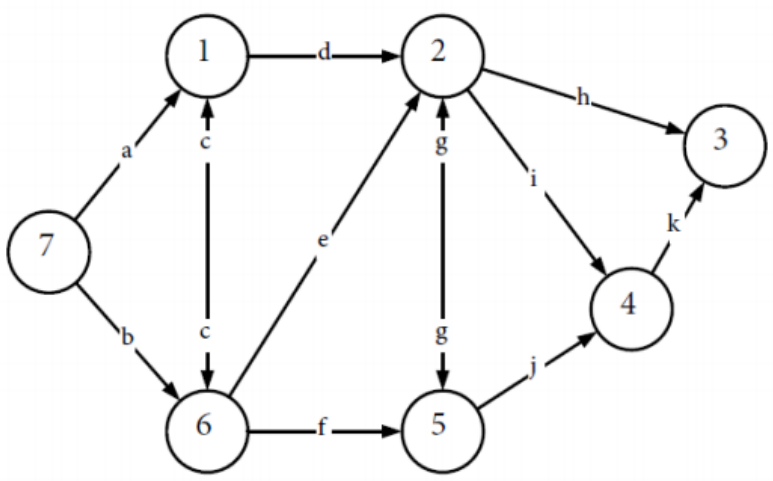
1. Obtener un arreglo de *n* números
2. Desde *i*=1 hasta *n*
3. Hacer *suma* = *suma* + *arreglo*[*i*]
4. Hacer *media* = *suma* / *n*
5. Imprimir *media*

Este algoritmo tiene una complejidad constante y dependiente del número de elementos introducidos, es decir, complejidad O(n). Por tanto si comparamos la version iterativa del cálculo de la media, con la version divide y vencerás del algoritmo es bastante obvio determinar que la version iterativa es mucho más eficiente es términos computacionales. La version iterativa tiene una complejidad constate O(n), mientras que la version recursiva presenta una complejidad exponencial $O(2^n)$.



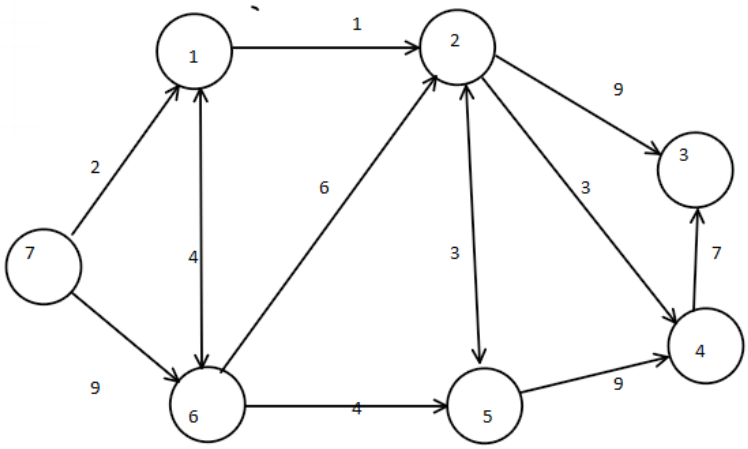
Problema 3

Se dispone el siguiente grafo, y se pide aplicar el algoritmo de Floyd con el objetivo de obtener una matriz de distancias mínimas.



Antes de empezar debo sustituir las letras por valores numéricos aleatorios comprendidos entre 1 y 10. A continuación se muestra la asignacion final para cada valor:

a	b	c	d	e	f	g	h	i	j	k
2	9	4	1	6	4	3	9	3	9	7



Una vez se ha asignado a cada arista con un valor numérico aleatorio muestro la matriz de distancias, donde represento la distancia entre cada uno de los nodos. En el caso en el que ambos nodos no estén conectados directamente, se denotare mediante el símbolo “∞”.

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

K = 1:

A partir de la matriz de distancias, seleccionare la primera fila y columna del mismo e iré sumando cada elemento de la columna, con cada elemento de la fila. Realizare una comparación a la hora de realizar la suma de forma que comparare dicha suma con el elemento intersección de la columna y la fila de los elementos que se estén sumando en se momento. Si la suma es mayor que el elemento intersección, la matriz de distancias no se modifica. Por el contrario si la suma es menor que el elemento intersección la matriz de distancias cambia.

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

Sumo el elemento de posición [2, 1] con el elemento de posición [1, 2] y lo comparo con el elemento intersección:

$$\infty + 1 > 0$$

Puesto que la suma es mayor que el elemento intersección, la matriz de distancias no se modifica. Ahora sigo con el algoritmo y sumo el elemento [2, 1] con el elemento [1, 3], el elemento [2, 1] con el elemento [1, 4], y así sucesivamente…

$$\infty + \infty > 9$$

$$\infty + \infty > 9$$

$$\infty + \infty > 3$$

$$\infty + 4 >= 4$$

$$\infty + \infty >= \infty$$

Como podemos observar, hasta ahora no se ha modificado la matriz de distancias. Seguimos comparando, ahora con los elementos de la tercera fila. De hecho si observamos detenidamente, ningún elemento de las filas 2, 3, 4, 5 para k = 1, va a modificar la matriz puesto que todos tiene como valor ∞.

Por ello comparo directamente al elemento de la fila 6 procediendo de la misma forma que hasta ahora:

$$\underline{4 + 1 < 6}$$

$$4 + \infty >= \infty$$

$$4 + \infty >= \infty$$

$$4 + \infty > 4$$

$$4 + 4 > 0$$

$$4 + \infty >= \infty$$

Como podemos observar en este caso la suma de los valores de [6, 1] y [1, 2] es menor que el elemento intersección, por ello remplazo en la matriz de distancias el elemento intersección por la suma de ambos.

Una vez que se conoce el algoritmo, realizo el mismo proceso para la fila 7. A partir de ahora obviare los pasos que impliquen infinitos puesto que no producirán ningún cambio en la matriz resultante.

$$2 + 1 < \infty$$

$$2 + 4 < 9$$

Ahora que ya he terminado el algoritmo para k = 1, muestro los cambios producidos en la matriz de distancias:

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

Continuo el algoritmo para k = 2:

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

Ahora comparo los elementos de la segunda fila, con los elementos de la segunda columna procediendo de la misma forma que antes. Como he dicho antes obviare los pasos que impliquen infinitos, además mostrare todas las operaciones realizadas en este paso a continuación:

Fila 1:	Fila 5:	Fila 6:	Fila 7:
$1 + 9 < \infty$	$3 + 9 < \infty$	$5 + 9 < \infty$	$3 + 9 < \infty$
$1 + 3 < \infty$	$3 + 3 < 9$	$5 + 3 < \infty$	$3 + 3 < \infty$
$1 + 3 < \infty$	$3 + 3 > 0$	$5 + 3 > 4$	$3 + 3 < \infty$

Una vez he considerado todas las operaciones posibles del paso k = 2 muestro todos los cambios realizados en la matriz:

	1	2	3	4	5	6	7
1	0	1	10	4	4	4	∞
2	∞	0	9	3	3	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	7	0	∞	∞	∞
5	∞	3	6	7	0	∞	∞
6	4	5	14	8	4	0	∞
7	2	3	12	6	6	6	0

Una vez realizado el paso k = 2, continuo el algoritmo para k = 3:

	1	2	3	4	5	6	7
1	0	1	10	4	4	4	∞
2	∞	0	9	3	3	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	7	0	∞	∞	∞
5	∞	3	6	7	0	∞	∞
6	4	5	14	8	4	0	∞
7	2	3	12	6	6	6	0

Ahora comparo los elementos de la tercera fila, con los elementos de la tercera columna procediendo de la misma forma que antes. Si nos fijamos detenidamente, pararnos a realizar este paso no tendría sentido pues la tercera fila está llena de infinitos por lo que la matriz de distancias no cambiaría, por ello paso directamente al paso k = 4.

Paso k = 4:

	1	2	3	4	5	6	7
1	0	1	10	4	4	4	∞
2	∞	0	9	3	3	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	7	0	∞	∞	∞
5	∞	3	6	7	0	∞	∞
6	4	5	14	8	4	0	∞
7	2	3	12	6	6	6	0

Ahora comparo los elementos de la cuarta fila, con los elementos de la cuarta columna procediendo de la misma forma que antes.

Fila 1:	Fila 2:	Fila 5:	Fila 6:	Fila 7:
4 + 7 > 10	*7 + 3 > 9*	*7 + 7 > 6*	*8 + 7 > 14*	*6 + 7 > 12*

En este paso a diferencia del paso anterior, si se podían realizar operaciones, pero ninguna de ellas modificara la matriz de distancias pues la suma de los elementos es menor que el valor del elemento intersección. Por lo tanto la matriz de distancias no sufrirá ningún cambio.

Paso k = 5:

	1	2	3	4	5	6	7
1	0	1	10	4	4	4	∞
2	∞	0	9	3	3	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	7	0	∞	∞	∞
5	∞	3	6	7	0	∞	∞
6	4	5	14	8	4	0	∞
7	2	3	12	6	6	6	0

Ahora comparo los elementos de la quinta fila, con los elementos de la quinta columna procediendo de la misma forma que antes.

Fila 1:	Fila 2:	Fila 6:	Fila 7:
4 + 3 > 1	3 + 3 > 0	4 + 3 > 5	6 + 3 > 3
4 + 6 >= 10	3 + 6 >= 9	4 + 6 < 16	6 + 6 >= 12
4 + 7 > 4	3 + 7 > 3	4 + 7 > 8	6 + 7 > 6

	1	2	3	4	5	6	7
1	0	1	10	4	4	4	∞
2	∞	0	9	3	3	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	7	0	∞	∞	∞
5	∞	3	6	7	0	∞	∞
6	4	5	10	8	4	0	∞
7	2	3	12	6	6	6	0

Paso k = 6

	1	2	3	4	5	6	7
1	0	1	10	4	4	4	∞
2	∞	0	9	3	3	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	7	0	∞	∞	∞
5	∞	3	6	7	0	∞	∞
6	4	5	10	8	4	0	∞
7	2	3	12	6	6	6	0

Fila 1:	Fila 7:
4 + 4 > 0	*6 + 4 > 2*
4 + 5 > 1	*6 + 5 > 3*
4 + 10 > 10	*6 + 10 > 12*
4 + 8 > 4	*6 + 8 > 6*
4 + 4 > 4	*6 + 4 > 6*

En el paso anterior no se ha modificado la matriz de distancias, por ello trabajo con la misma matriz para el paso k = 7:

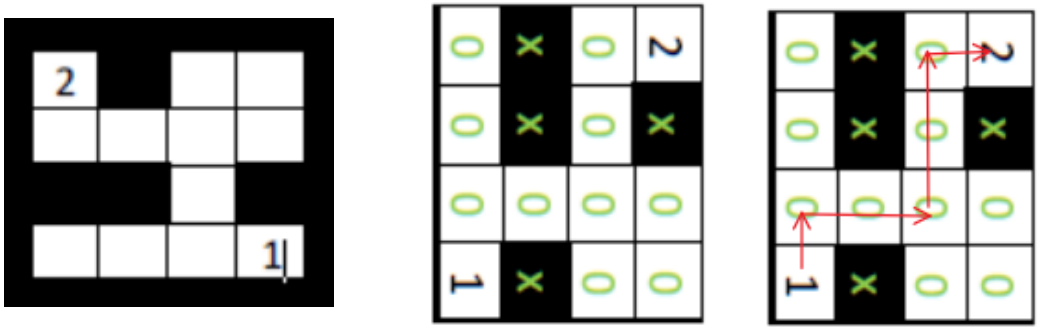
	1	2	3	4	5	6	7
1	0	1	10	4	4	4	∞
2	∞	0	9	3	3	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	7	0	∞	∞	∞
5	∞	3	6	7	0	∞	∞
6	4	5	10	8	4	0	∞
7	2	3	12	6	6	6	0

Si observamos detenidamente al igual que ha ocurrido con pasos anteriores, en este paso no se va a modificar la matriz puesto que la séptima columna está llena de infinitos. Por lo tanto la matriz de distancias mínimas es la siguiente:

	1	2	3	4	5	6	7
1	0	1	10	4	4	4	∞
2	∞	0	9	3	3	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	7	0	∞	∞	∞
5	∞	3	6	7	0	∞	∞
6	4	5	10	8	4	0	∞
7	2	3	12	6	6	6	0

Problema 4

En este problema disponemos de un “mapa” donde partiendo de un punto inicial debemos llegar a un punto final con ciertas restricciones. Los únicos dos movimientos posibles son hacia delante y hacia la derecha. Además se debe llegar al destino en la menor cantidad de tiempo posible, es decir, debemos reducir el numero de movimientos al máximo. Debemos saber que inicialmente estamos orientados hacia la izquierda de forma que en este ejemplo de mapa estaríamos mirando hacia el oeste. A continuación de muestra un ejemplo de mapa, además adicionalmente se muestra cómo debemos indicar al programa el estado del mapa, además he orientado el mapa para que se pueda observar mejor los movimientos disponibles:



Se muestra también una solución valida al problema, donde solo se están realizando movimientos hacia delante y hacia la derecha hasta llegar a la solución final.

El algoritmo implementado consiste el lo siguiente. El objetivo es llegar desde una posición inicial hasta una posición destino empleando solamente dos tipos de movimiento (arriba y derecha). En primer lugar se verifica si la celda actual es la celda destino, es cuyo caso el algoritmo acabara. Si la celda no es la celda destino, se realizan los dos movimientos posibles verificando que las celdas estén vacías. Si ninguno de los dos movimientos es posible, retrocedo y cambio la ruta actual. El algoritmo implementado en java es el siguiente:

```java
/**
 * Se encarga de encontrar la posicion final mediante backtracking
 * @param fila
 * @param columna
 * @return
 */
public static boolean encontrarSolucion(int fila, int columna) {
    if ((fila == 0) && (columna == 3)) {
        solucion.get(fila).add(columna, "x");
        return true;
    }
    if (dentroTablero(fila, columna) && posValida(fila, columna)) {
        solucion.get(fila).set(columna, "x");
        if (encontrarSolucion(fila - 1, columna)) {
            movimientos.add("[" + fila + "," + columna + "]" + " movimiento hacia arriba");
            return true;
        }
        if (encontrarSolucion(fila, columna + 1)) {
            movimientos.add("[" + fila + "," + columna + "]" + " movimiento hacia la derecha");
            return true;
        }
        solucion.get(fila).set(columna, "0");
        return false;
    }
    return false;
}
```

Además en el programa se aportan dos ficheros de texto, un fichero de entrada y un fichero de salida. En el fichero de entrada se extraen los datos del problema, así como el numero de columnas y de filas y el contenido de cada una de ellas. Por otro lado en el fichero de salida se especifican los pasos a realizar para resolver el laberinto. En cuanto a la salida del programa para el ejemplo mostrado anteriormente, es la siguiente:

```
run:
Num filas: 4
Num columnas: 4
-------TABLERO-------
0       x       0       2

0       x       0       x

0       0       0       0

1       x       0       0

------SOLUCION------
0       0       x       x

0       0       x       0

x       x       x       0

x       0       0       0

BUILD SUCCESSFUL (total time: 0 seconds)
```

Problema 5

Disponemos de un cuadro de asignacion como el siguiente, en el que debemos asignar 16 números aleatorios entre 5 y 35.

	1	2	3	4
a	a11	a12	a13	a14
b	a21	a22	a23	a24
c	a31	a32	a33	a34
d	a41	a42	a43	a44

Ahora genero los 16 números aleatorios, se muestra la misma tabla anterior pero sustituyendo los valores de cada celda por los valores numéricos aleatorios comprendidas entre 5 y 35.

	1	2	3	4
a	35	5	14	11
b	18	8	9	6
c	5	15	13	24
d	22	34	28	13

Una vez generados los números aleatorios, procedo a resolver el problema de asignacion. El problema de asignacion consiste en asignar N tareas a N agentes, de manera que cada tarea se realice una única vez y cada agente tenga asociada una única tarea. Para resolver este problema de asignacion, se empleará el algoritmo de ramificación y poda.

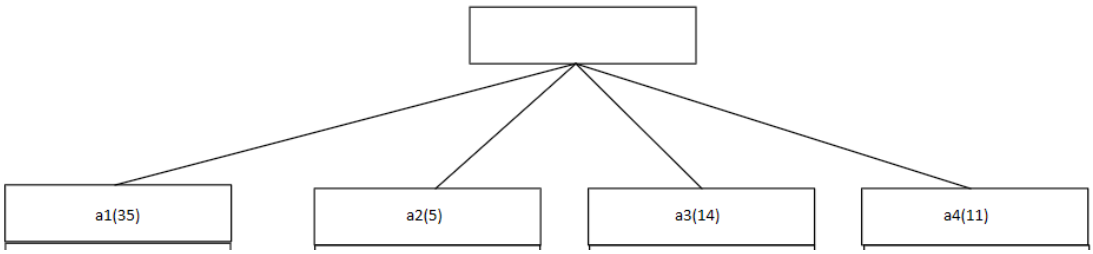
En primer lugar asigno una cota real, asignando a cada agente una tarea específica, por ejemplo una de las soluciones posibles seria coger la diagonal principal cuyo coste sería:

$$a11(35) + a22(8) + a33(13) + a44(13) = 69$$

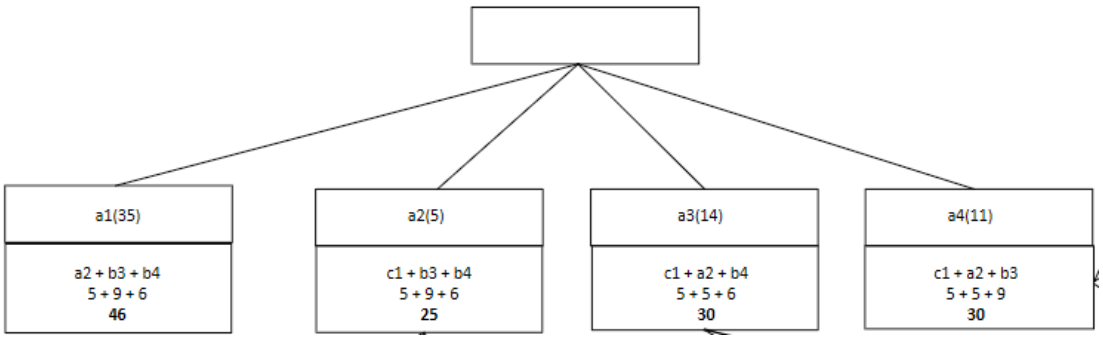
Ahora cualquier candidato parcial que tenga un coste superior a 69 será rechazado o podado ya que en vez de seguir generándolo nos podríamos quedar con la solución completa ya encontrada.

Una vez asignada una cota real, debo asignar una cota estimada, vamos a considerar como cota estimada, por ejemplo, el coste mínimo que se tendría por parte de los agentes. Los mínimos de las filas serian: 5 (a2) + 6 (b4) + 5 (c1) + 13 (d4) = 28, y los mínimos por columnas serian 5 (c1) + 5 (a2) + 9 (b3) + 6 (b4) = 25.

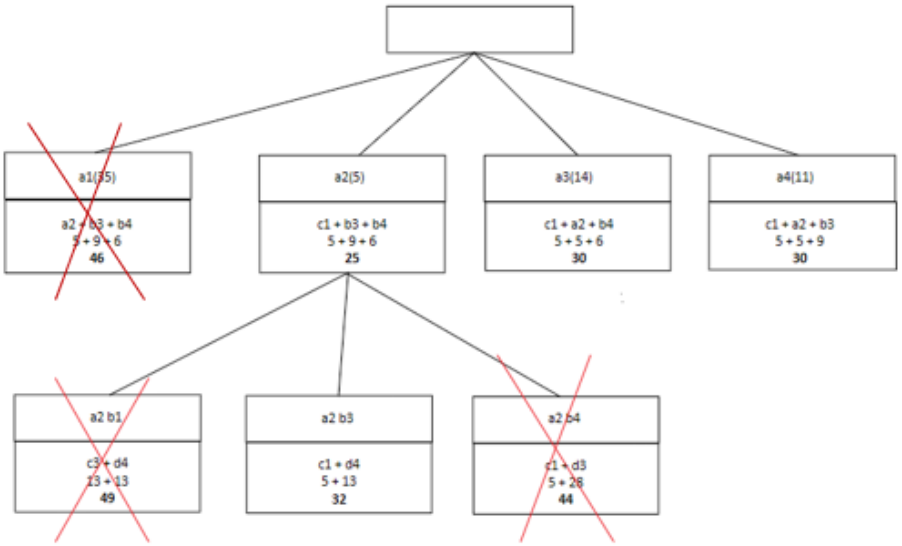
Una vez asignadas cota real y cota estimada, procedo a recorrer el árbol. El cual tendrá los siguientes nodos principales:



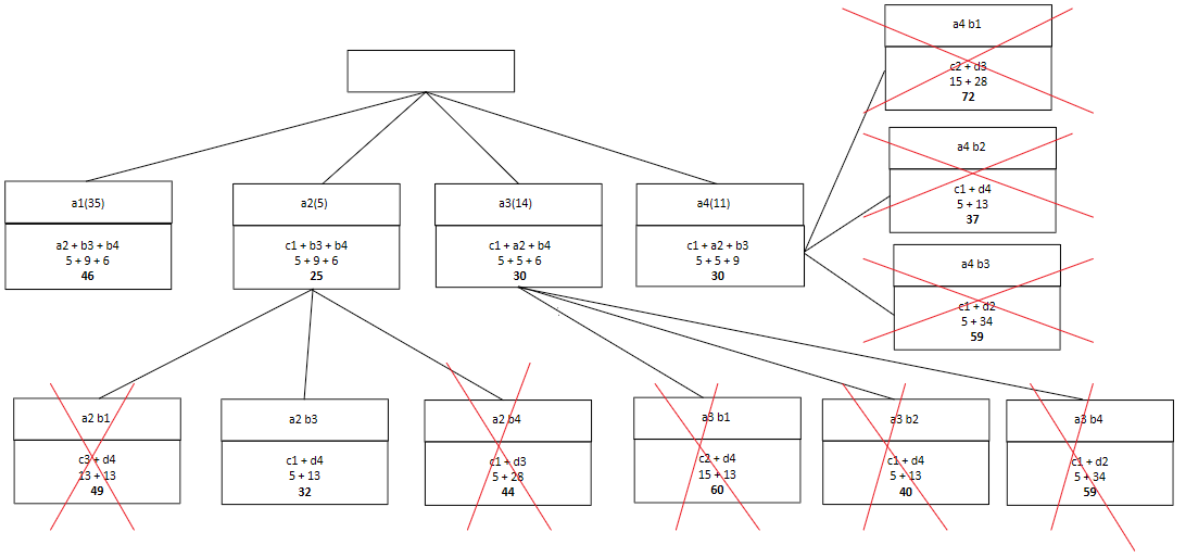
En primer lugar calculo el valor mínimo de cada uno de los nodo principales. En este caso como el valor de ningún nodo es mayor que la cota real, no puedo podar ningún nodo desde el principio.



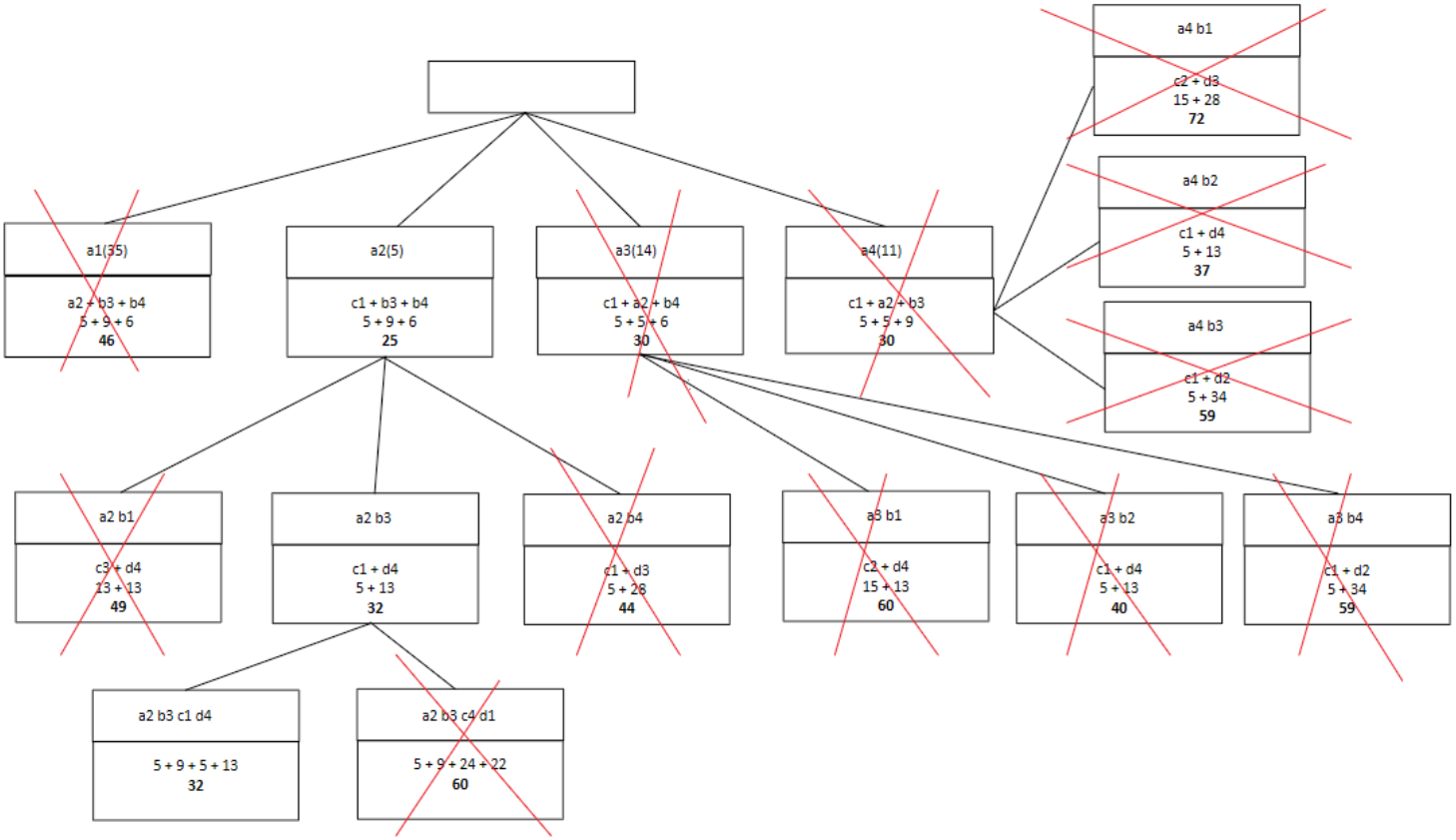
Ahora empiezo a desarrollar el nodo con menor valor, en este caso el nodo a2(5) y empiezo a podar las ramas cuyo valor supere dicha cifra. En este caso las ramas a2 b4 = 44 y a2 b1 = 49 son podadas.



Una vez he desarrolado el primer nivel del nodo a2, repito el mismo prodecimiento para el resto de ramas del arbol, a3 y a4. Como podemos observar la rama a1 ha sido podada pues supera el valor del nodo actual mas pequenno de nuestro arbol.



Como se puede observar, ninguna rama del arbol tiene un valor menor a 25, por ello dichas ramas no seran una solucion al problema. A continuacion sigo desarrollando la ultima rama restante del arbol.



Tras desarrollar a2 b3 se encuentra la que finalmente sera la solucion optima: a2 b3 c1 d4 con un valor de 32.