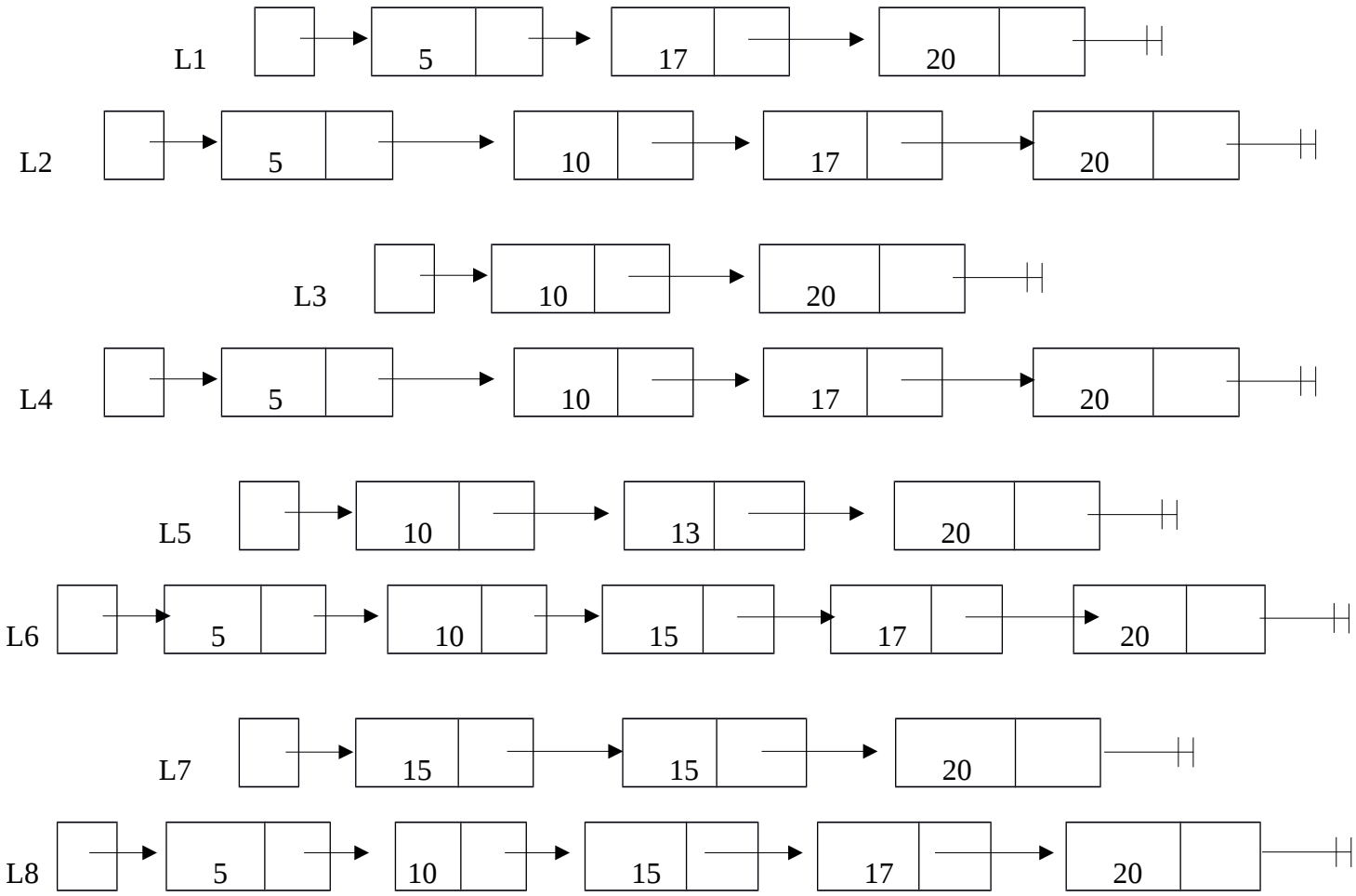
ESTRUCTURAS DE DATOS Y ALGORITMOS Junio 2017

1. Sublista (2,5 puntos)

Dadas 2 listas ordenadas ascendentemente, queremos hacer un subprograma que diga si los elementos de la primera lista están contenidos en la segunda lista, manteniendo el orden.
Por ejemplo, L1 es sublista de L2, y L3 lo es de L4. Sin embargo, L5 no es sublista de L6, ni L7 de L8.



Se deberá calcular el coste del algoritmo resultante, que se valorará (es decir, el coste del algoritmo es importante, cuanto más eficiente, mejor). Cada lista solo puede ser recorrida una sola vez.

```
public class Node<T> {
   T data;
   Node<T> next;
}

public class LinkedList<T> {
   Node<T> first;

   public boolean subLista(DoubleLinkedList<T> subLista)
   // Postcondición: true si “subLista” está contenida y false si no
}
```

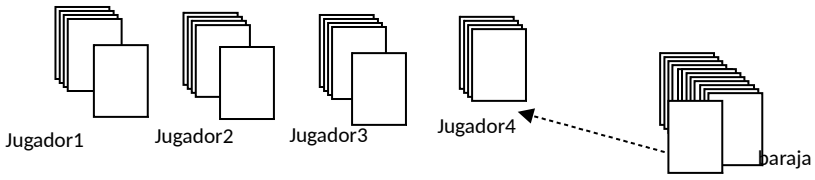
Se pide:

- Implementar el método
- Calcular razonadamente el coste del algoritmo

2. Partida de cartas (2,5 puntos)

Cuatro amigos van a jugar al juego de cartas de los SEISES. Las características del juego son las siguientes:

- La baraja contiene 40 cartas divididas en cuatro palos (oros, copas, espadas y bastos). Cada palo consta de 10 cartas numeradas del 1 al 10.
- Las cartas se reparten en 4 montones, es decir, 10 cartas por jugador.



- El funcionamiento de la partida es el siguiente:
    - El primer jugador juega con la primera carta de su montón,
        - Si puede la coloca en la mesa.
        - Si no puede colocar la carta en la mesa, la deja de nuevo en el fondo de su montón
    - Y pasa el turno al siguiente jugador.

- Un jugador puede colocar su carta en la mesa en 2 casos:
    - La carta es un 6.
    - La carta no es 6, pero es una carta consecutiva a otra que sí se encuentra en la mesa. Por ejemplo, si tengo el 2 de bastos y en la mesa está el 3 de bastos, ó si tengo el 8 de oros y en la mesa está el 7 de oros, etc.

- El juego acaba cuando un jugador se coloca la última carta de su montón, es decir, se queda sin cartas.

Utilizando los TADs Baraja y Bicola, **(no hay que implementar ninguna operación de los TAD's) se pide:**

a) Implementar las estructuras de datos necesarias para representar a los jugadores y la mesa con el estado del juego.

b) *Diseñar y escribir* un subprograma que simule una partida diciendo al final cuál ha sido el jugador ganador.

Baraja:

```
public class Carta {

   String palo;  // oros, copas, espadas, espadas
   int valor;    // valor entre 1 y 10
}

public class Baraja {

   prívate Carta[] cartas;

   public Baraja(); // constructora
   // postcondición: la baraja contiene 40 cartas en orden aleatorio

   public Iterator<Carta> iterador()
   // devuelve un iterador para recorrer las cartas

}
```

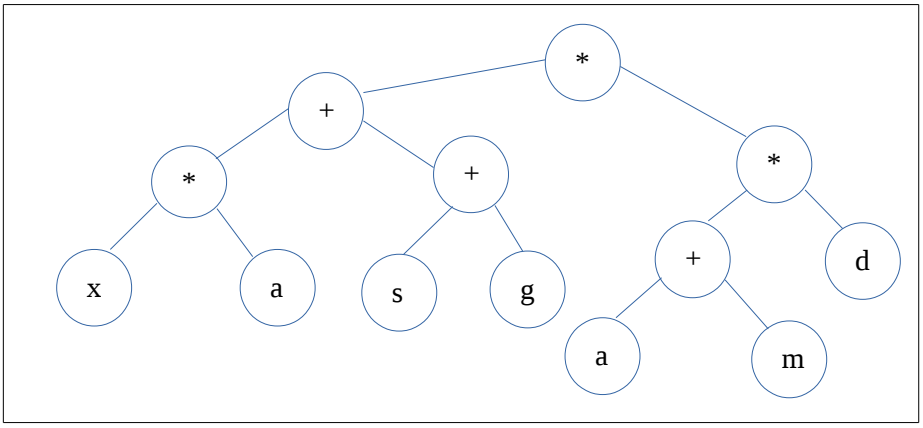
Bicola:

```
public class Bicola<T> {

   public Bicola(); // constructora
   // Inicializa la bicola (vacía)

   public boolean estaVacia();
   // Indicará si la bicola está o no vacía.

   public void insertarIzq(T elemento);
   // añade el elemento E por el extremo izquierdo de la bicola

   public void insertarDer(T elemento);
   // añade el elemento E por el extremo derecho de la bicola

   public void eliminarIzq();
   // borra el elemento del extremo izquierdo de la bicola

   public void eliminarDer();
   // borra el elemento del extremo derecho de la bicola

   public T obtenerIzq();
   // obtiene el elemento.del extremo izquierdo de la bicola

   public T obtenerDer();
   // obtiene el elemento.del extremo derecho de la bicola

}
```

3. Evaluar expresión aritmética (2,5 puntos)

Queremos hacer un subprograma que tome como entrada un árbol que representa una expresión aritmética y una tabla hash que contiene los valores de las variables en un momento dado de la ejecución del programa.

La expresión aritmética puede contener en sus nodos dos tipos de elementos: nombres de variable u operadores (sole se admitirán los operadores de suma y multiplicación).



El árbol anterior representa la expresión: ((x * a) + (s + g)) * ((a + m) * d)

Además, se tiene una tabla hash que contiene las variables con sus valores:

x	4
a	5
s	7
g	1
m	5
d	2

El subprograma deberá devolver el valor numérico resultado de evaluar esa expresión.

En el ejemplo dado, el resultado es 560 = ((4 * 5) + (7 + 1)) * ((5 + 5) * 2)

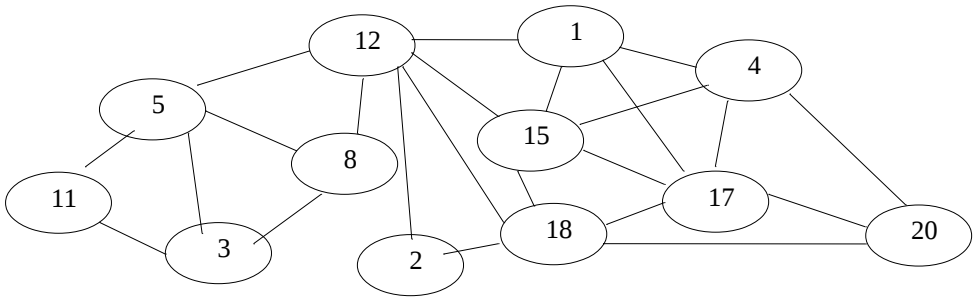
```
public class BinaryTreeNode<T> {
	T element;
	BinaryTreeNode<T> left, right;
}
public class ArbolExpresion {
	BinaryTreeNode<InfoElemExp> root;

	public Integer evaluar(HashMap<String, Integer> tHash)
	// pre: tHash contiene los valores de las variables
	// post: Se ha evaluado la expresión correspondiente al árbol
	//	Los valores de las variables se han tomado de tHash
	//	Si una variable del árbol no se encuentra en la
	//	tabla hash, se asumirá que el valor por defecto es cero
}
public class InfoElemExp {
   String elem;	// *, +, ó nombre de una variable
   boolean operador; // true -> operador, false -> variable
}
```

Se pide:

- Implementar el algoritmo
- Establecer de manera razonada el coste del algoritmo.

4. Buscar camino (2,5 puntos)

Tenemos un grafo:



Queremos desarrollar un algoritmo que, dada una lista de valores, nos diga si existe un camino en el grafo que conecta los elementos de la lista. Es decir, por cada pareja de elementos consecutivos de la lista (x, y), existe un arco en el grafo desde x hasta y.

Por ejemplo:

- el resultado con la lista <12, 1, 4, 17, 18, 15> será true, ya que ese camino sí existe en el grafo, que contiene los arcos (12, 1), (1, 4), (4, 17), (17, 18), y (18, 15)
- el resultado con la lista <12, 1, 20, 17, 18, 15> será false, ya que no se puede ir del nodo 1 al 20 (no hay una conexión directa)

```
public class Grafo
{

   protected int numVertices;   // number of vertices in the graph
   protected int[][] adjMatrix;   // adjacency matrix


   public boolean existeCamino(ArrayList<Integer> lista)
}
```

Se pide:

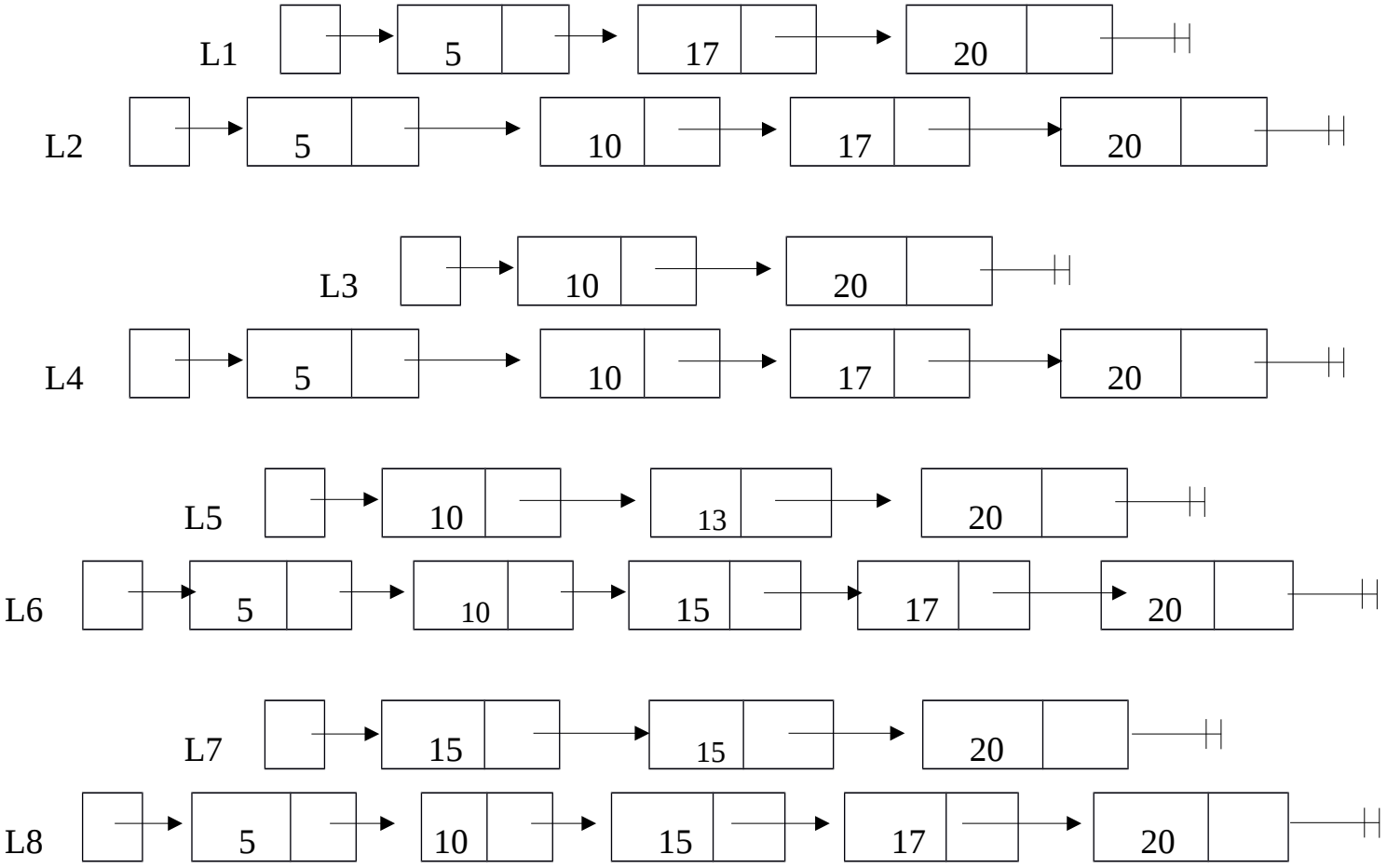
- Implementar el algoritmo
- Establecer de manera razonada el coste del algoritmo.

DATU-EGITURAK ETA ALGORITMOAK EKAINA 2017

1. Aldizkako azpilista (2,5 puntu)

Gorantz ordenatutako 2 lista emanda, azpirograma bat egin L1-eko elementu guztiak L2 listan dauden esateko, ordena mantenduz.

Adibidez, L1 L2-ren aldizkako azpilista da, L3 L4-rena, baina L5 ez da L6-ren aldizkako azpilista, ezta L7 L8-rena.



Lortutako algoritmoaren kostua kalkulatu beharko da. Soluzioan algoritmoaren eraginkortasuna baloratuko da (lista bakoitza behin bakarrik korritu daiteke).

```
public class Node<T> {
   T data;
   Node<T> next;
}

public class LinkedList<T> {
   Node<T> first;

   public boolean azpiLista(DoubleLinkedList<T> zerrenda)
   // Postbaldintza: true “zerrenda” zerrendaren barruan badago eta false bestela
}
```

Hau eskatzen da:

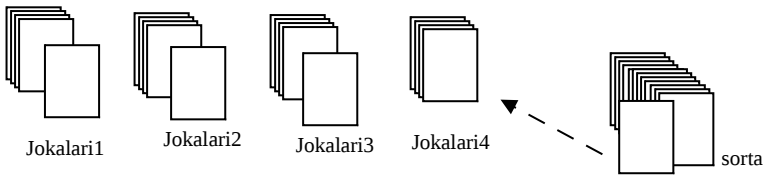
- Algoritmoa inplementatu
- Kostua kalkulatu modu arrazoituan

2. Karta jokaldia (2,5 puntu)

Lau lagunek SEIKOEN jokaldia egin nahi dute. Hauek dira jokoaren ezaugarri nagusiak:

1. Karta sortak 40 karta ditu, lau sailetan banatuta (urreak, kopak, ezpatak eta bastoiak). Sail bakoitzak 10 karta ditu, 1etik 10era.

2. Kartak lau jokalarien artean banatuko dira, hau da, jokalari bakoitzak 10 karta izango ditu.



3. Partida honela antolatuko da:
    - Lehen jokalariak bere lehen karta hartuko du (gainekoa),
        - Ahal badu (ikus behean), mahaian jarriko du.
        - Ezin badu, orduan karta hori bere karta guztien azpian jarriko du
    - Eta hurrengo jokalariari pasako zaio txanda.

4. Jokalari batek ondoko kasuetan jar dezake karta mahaian:
    - Karta seiko bat da.
    - Karta ez da seikoa, baina mahaian badagoen karta baten ondorengoa da. Adibidez, nire karta urrezko bikoa baldin bada, eta mahaian urrezko hirukoa balego, edo bestela nire karta urrezko zortzikoa balitz eta mahaian urrezko zazpikoa balego.

5. Jokoa jokalari batek bere azken karta jartzen duenean amaituko da

Karta_Sorta eta Bilara DMAak erabilita (ez dira inplementatu behar DMA hauen eragiketak) ondokoa eskatzen da:

a) Problema hau ebazteko erabiliko dituzun datu-egiturak definitu (jokalariak eta mahaiaren egoera adierazteko).

b) ***Diseinatu eta idatzi*** azpiprograma bat partida bat simulatu eta irabazlea zein den esango diguna.

KartaSorta DMA:

```
public class Karta {
   String palo;  // urrea, kopa, ezpata, bastoia
   int balioa;    // 1 eta 10 arteko balioa
}
public class KartaSorta {
   prívate Karta[] kartak;
   public KartaSorta (); // eraikitzailea
   // post: 40 karta daude ausaz
   public Iterator<Carta> iteradore()
   // kartak aztertzeko iteradorea bueltatzen du
}
```

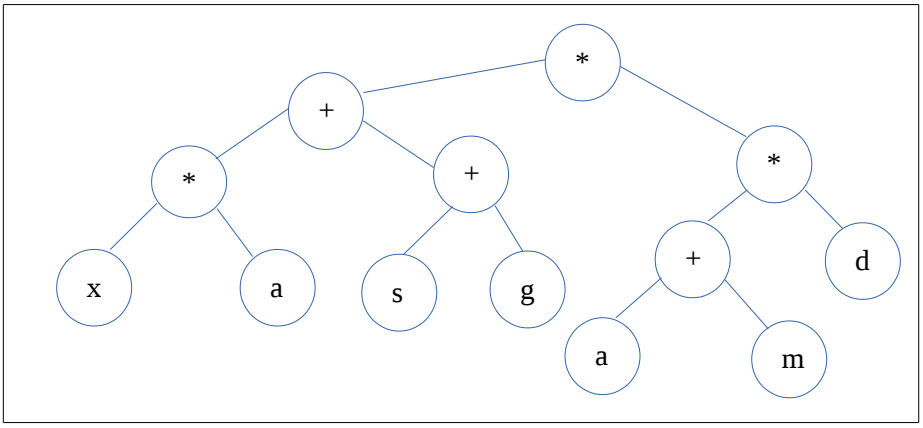
Bilara:

```
public class Bilara<T> {
   public Bilara(); // eraikitzailea
   // bilara hasieratzen du (hutsa)
   public boolean hutsaDa();
   // true B hutsa baldin bada eta false bestela.
   public void txertatuEzk(T elemento);
   // E elementua ezkerretik gehitu dio
   public void txertatuEsk(T elemento);
   // E elementua eskuinetik gehitu dio
   public void ezabatuEzk();
   // ezkerrean dagoen elementua ezabatu du
   public void ezabatuEsk();
   // eskuinean dagoen elementua ezabatu du
   public T lortuEzk();
   // ezkerrean dagoen elementua bueltatzen du
   public T lortuEsk();
   // eskuinean dagoen elementua bueltatzen du
}
```

3. Adierazpenaren ebaluazioa (2,5 puntu)

Azpiprograma bat egin nahi dugu, sarreratzat zuhaitz bat eta hash-taula bat hartuko dituena. Bata espresio aritmetikoa adierazten duen zuhaitza da, eta bestea programa baten exekuzioaren une batean aldagaien balioak gordetzen dituena.

Espresio aritmetikoak bi adabegi-mota ditu: aldagaiak eta eragileak (sinplifikatzearren, batuketa eta biderketarako eragileak bakarrik onartuko dira).



Zuhaitz horrek espresio hau adierazten du: ((x * a) + (s + g)) * ((a + m) * d)

Hash-taulak aldagaien izenak ditu gakotzat, eta bikote bakoitzak ondoko informazioa izango du:

x	4
a	5
s	7
g	1
m	5
d	2

Azpiprogramak espresio horren ebaluazioaren emaitza eman beharko du.

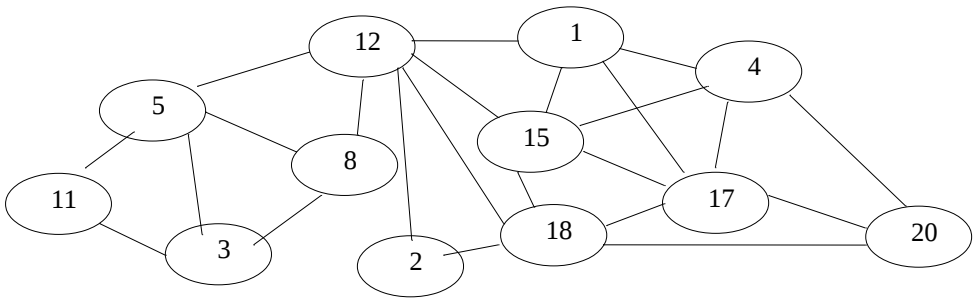
Aurreko adibidean emaitza hau da: 560 = ((4 * 5) + (7 + 1)) * ((5 + 5) * 2)

Hau eskatzen da:

- Algoritmoa inplementatu
- Algoritmoaren kostua kalkulatu, modu arrazoituan.

```
public class BinaryTreeNode<T> {
      T element;
      BinaryTreeNode<T> left, right;
}
public class InfoElemEspresioa {
   String elem;       // *, +, edo aldagai baten izena
   boolean eragigaia; // true -> eragigaia, false -> aldagaia
}
public class Zuhaitza {
      BinaryTreeNode<InfoElemEspresioa> root;

      public Integer ebaluatu(HashMap<String, Integer> tHash)
       // pre: tHash taulak aldagaiak ditu
       // post: Zuhaitzaren adierazpena ebaluatu da.
       //       Aldagaien balioak tHash taulatik hartu dira.
       //       Aldagai bat ez badago hash-taulan, orduan
       //       zero hartuko da bere baliotzat
}
```

4. Bidea bilatu (2,5 puntu)
Ondoko irudiak grafo bat adierazten du:



Balioen zerrenda bat emanda, zerrendako elementuak konektatzen duen bidea dagoen jakin nahi da. Hau da, zerrendan jarraian doazen elementuen (x, y) bikote bakoitzeko, grafoak arku bat du x-etik y-raino.

Adibidez:

- <12, 1, 4, 17, 18, 15> zerrendarekin emaitza true izango da, bide hori grafoan baitago, eta arku hauek daudelako: (12, 1), (1, 4), (4, 17), (17, 18), eta (18, 15)
- <12, 1, 20, 17, 18, 15> zerrendarekin emaitza false izango da, ezin delako 1-etik 20-ra joan (ez dago arku zuzena)

```
public class Grafo
{

   protected int numVertices;   // number of vertices in the graph
   protected int[][] adjMatrix;   // adjacency matrix


   public boolean bideaDago(ArrayList<Integer> lista)
}
```

Hau eskatzen da:
- Algoritmoa inplementatu
- Algoritmoaren kostua kalkulatu, modu arrazoituan.