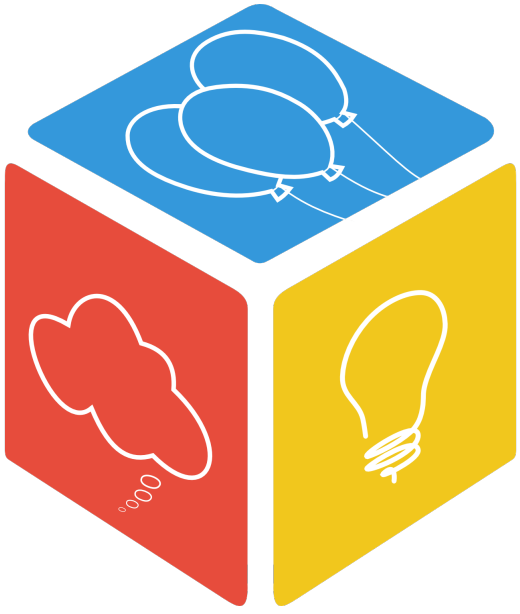
Solucionario Maratón de Programación UFPS 2016



Programación Competitiva
UFPS

Universidad Francisco de Paula Santander

Mayo 21 de 2016

1. Elephants

"M elefantes se balanceaban sobre la tela de una araña".

Teniendo un conjunto de M elefantes, cada uno con un peso w_i, y conociendo el maximo peso que la telaraña soporta, ¿cual es el mayor número de elefantes que se pueden subir a la telaraña sin que se rompa?

1. Elephants

Es un problema de optimización: Añadir el mayor número posible de elefantes sin exceder el peso de la tela. Si los elefantes tuvieran un valor, sería el clásico problema de la mochila.

1. Elephants

Es un problema de optimización: Añadir el mayor número posible de elefantes sin exceder el peso de la tela. Si los elefantes tuvieran un valor, sería el clásico problema de la mochila.

Sin embargo, ningún elefante vale mas que otro. Por lo tanto, la solución es simple: Subir a la tela los elefantes que pesen menos. Para esto, puedo simplemente ordenar los valores, y empezar a sumar los elefantes de menor a menor mientras no superen el limite.

En java puede usarse Collections.sort() o Arrays.sort(). En c/c++ sort().

Ejemplo: 5 elefantes 9, 1, 8, 7, 7. Peso máximo: 22

2. Royale with Cheese

Cada caracter que no ha aparecido previamente en la secuencia toma un valor ascendente numerico empezando en 1. Cuando el caracter ha aparecido previamente, toma ese mismo valor. Numeros intercambiables: 2-5, 6-9.

Ejemplo: "abbchocx" = 15534239

2. Royale with Cheese

Cada caracter que no ha aparecido previamente en la secuencia toma un valor ascendente numerico empezando en 1. Cuando el caracter ha aparecido previamente, toma ese mismo valor. Numeros intercambiables: 2-5, 6-9.

Ejemplo: "abbchocx" = 15534239

La solución es simple: Crear un mapa (TreeMap o HashMap) de <Caracter, Entero> donde guarde por cada letra, el valor que toma. IMPORTANTE: Evitar el doble recorrido.

3. Presidential Election

Existen A candidatos y B estados. En cada estado se realizan elecciones, y nos indican el porcentaje de votos que cada candidato obtuvo en cada ciudad, y la cantidad de votantes en esta ciudad. Debo decir quien gana las elecciones o quienes van a segunda vuelta.

La solución es tan simple como calcular el numero de votos de cada candidato en cada estado (teniendo su porcentaje y el total de votantes es solo una regla de 3), y sumar la cantidad de votos que obtuvo en todos los estados.

3. Presidential Election

Ejemplo:

Input	Output
3 4 3 25.0 25.0 25.0 25.0 4500 26.0 24.0 27.0 23.0 9000 10.0 10.0 10.0 70.0 1000 3 1 51.0 24.0 25.0 1000 2 2 0.7 99.3 2788000 99.3 0.7 2788000	4 3895 3 3655 1 510 1 2788000 2 2788000

IMPORTANTE: Math.round().

4. Sudoku

En este ejercicio se pide verificar que un sudoku esté solucionado correctamente. El tamaño del sudoku puede ser 4x4, 9x9, 16x16 o 25x25. Como se puede ver, los sudokus son diminutamente pequeños, por lo cual no nos preocupamos por la eficiencia.

Se debe verificar linea a linea, que no contenga numeros repetidos. Lo mismo columna a columna. Finalmente, cada uno de los subcuadros. Si todas las condiciones se cumplen, el sudoku es correcto.

4. Sudoku

```
1  static boolean verificarHorizontalVertical() {
2      TreeSet<Integer> v1 = new TreeSet<Integer>();
3      TreeSet<Integer> v2 = new TreeSet<Integer>();
4      int length;
5      for(int i = 0; i < n; i++) {
6        v1.clear();v2.clear();length = 0;
7        for(int j = 0; j < n; j++) {
8          v1.add(sudoku[i][j]);
9          v2.add(sudoku[j][i]);
10       }
11       if(v1.size() != v2.s ize() || v1.size() != n) {
12           return false;
13       }
14     }
15     return true;
16 }
```

4. Sudoku

```
1  static boolean verificarSubCuadricula() {
2      int raiz = (int)Math.sqrt(n);
3      TreeSet<Integer> v1 = new TreeSet<Integer>();
4      for(int i = 0; i < n; i += raiz) {
5          for(int j = 0; j < n; j += raiz) {
6              v1.clear();
7              length = 0;
8              for(k = i; k <  i + raiz; k++) {
9                  for(l = j; l < j + raiz; l++){
10                     v1.insert(sudoku[k][l]);
11                 }
12             }
13             if(v1.size() != n) return false;
14         }
15     }
16     return true;
17 }
```

4. Sudoku

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

5. Alliances in Hogwarts

El escuadron inquisitorial tiene una misión... deben determinar si los demás estudiantes de Hogwarts apoyan o no al ministerio. Para esto deben observar el comportamiento de los estudiantes y determinar las relaciones que existen entre ellos. Las relaciones entre los estudiantes se determinan con base a las siguientes reglas:

5. Alliances in Hogwarts

- Dos estudiantes son aliados si sus lealtades son las mismas.
- Dos estudiantes son enemigos si sus lealtades son diferentes.
- Un estudiante es aliado de si mismo.
- Nadie es enemigo de si mismo.
- Las relaciones de amistad y enemistad son mutuas.
- "Los amigos de mis amigos son mis amigos también".
- "El enemigo de mi enemigo es mi amigo".
- "El enemigo de mi amigo es mi enemigo también".

NOTA: Al comienzo de la investigación se asume que no existe ningún tipo de relación entre los estudiantes.

5. Alliances in Hogwarts

La entrada de este ejercicio esta compuesta por un conjunto de operaciones (c, x, y) donde c es el código de la operacion y x, y indican el id de los estudiantes. Existen 4 tipos de operaciones:

- 1 - El escuadrón marca la relación de x e y como amistad.
- 2 - El escuadrón marca la relación de x e y como enemistad.
- 3 - Umbridge pregunta si los estudiantes x e y son amigos.
- 4 - Umbridge pregunta si los estudiantes x e y son enemigos.

5. Alliances in Hogwarts

¿Cómo solucionarlo?

5. Alliances in Hogwarts

¿Cómo solucionarlo? Si...

5. Alliances in Hogwarts

¿Cómo solucionarlo? Si... Adivinaron ¡Disjoint Set! :3

5. Alliances in Hogwarts

¿Cómo solucionarlo? Si... Adivinaron ¡Disjoint Set! :3

Pero... existe un problema y este es las relaciones de enemistad.
Podemos solucionarlo de dos maneras:

- A. Que cada estudiante maneje una colección (set, array, vector...) de enemigos.
- B. Crear alter/egos para cada esudiante, de manera que se pueda manipular todo a traves del Disjoint Set.

¿Que opción elegí yo?

5. Alliances in Hogwarts

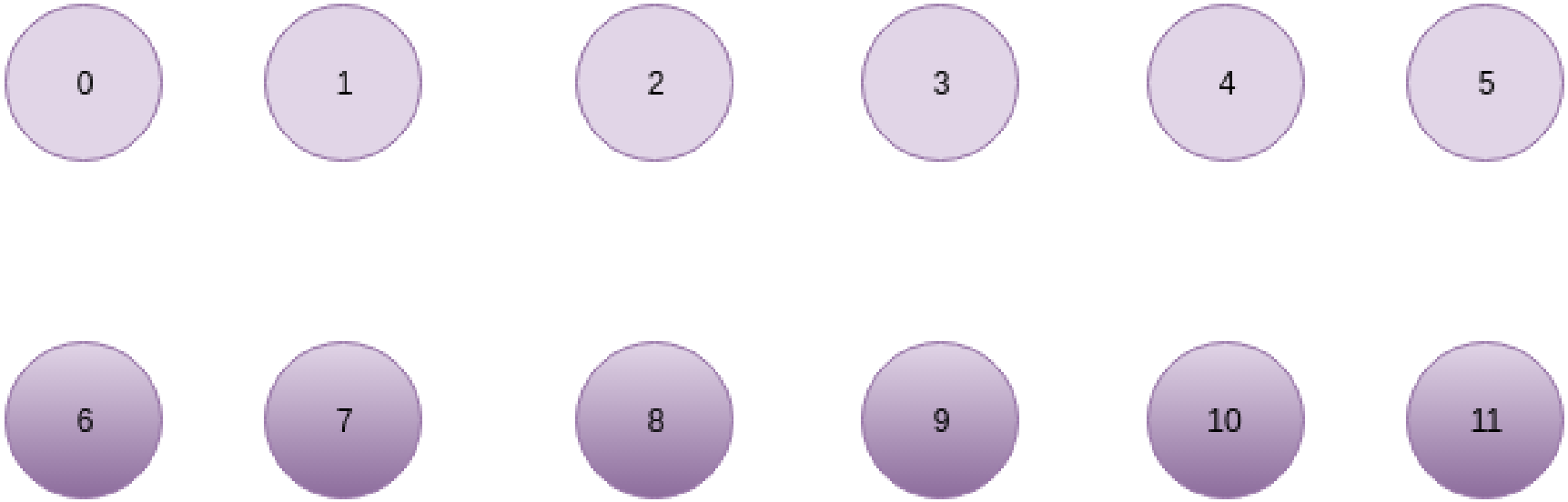
¿Cómo solucionarlo? Si... Adivinaron ¡Disjoint Set! :3

Pero... existe un problema y este es las relaciones de enemistad.
Podemos solucionarlo de dos maneras:

- A. Que cada estudiante maneje una colección (set, array, vector...) de enemigos.
- B. Crear alter/egos para cada esudiante, de manera que se pueda manipular todo a traves del Disjoint Set.

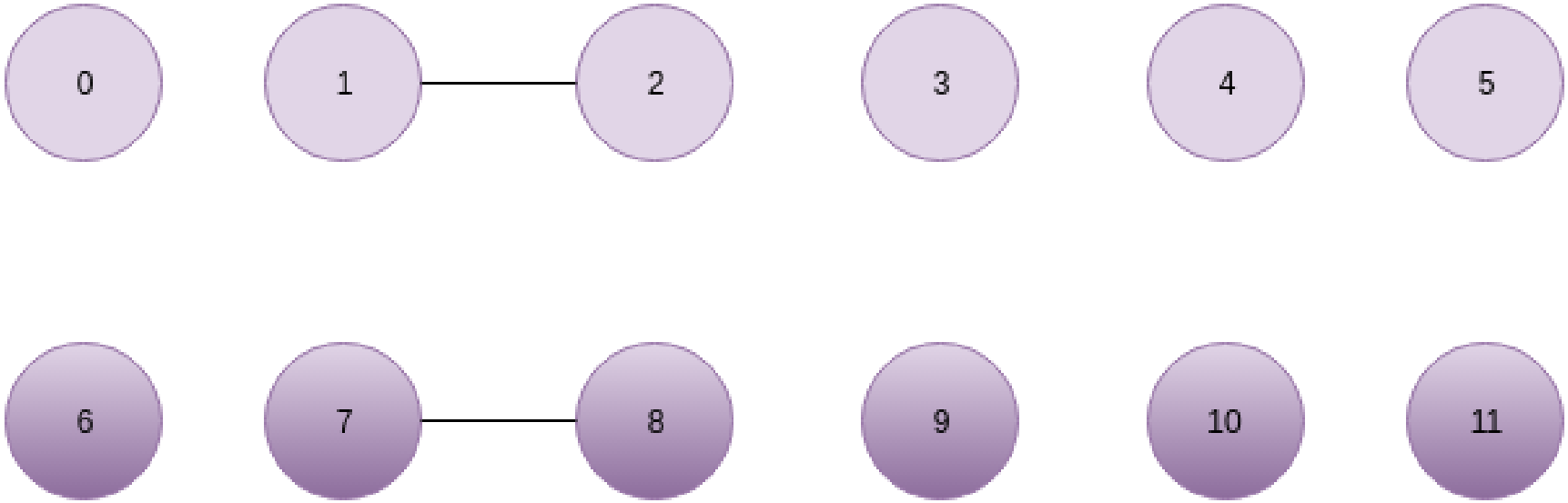
¿Que opción elegí yo? Correcto, la B.

5. Alliances in Hogwarts



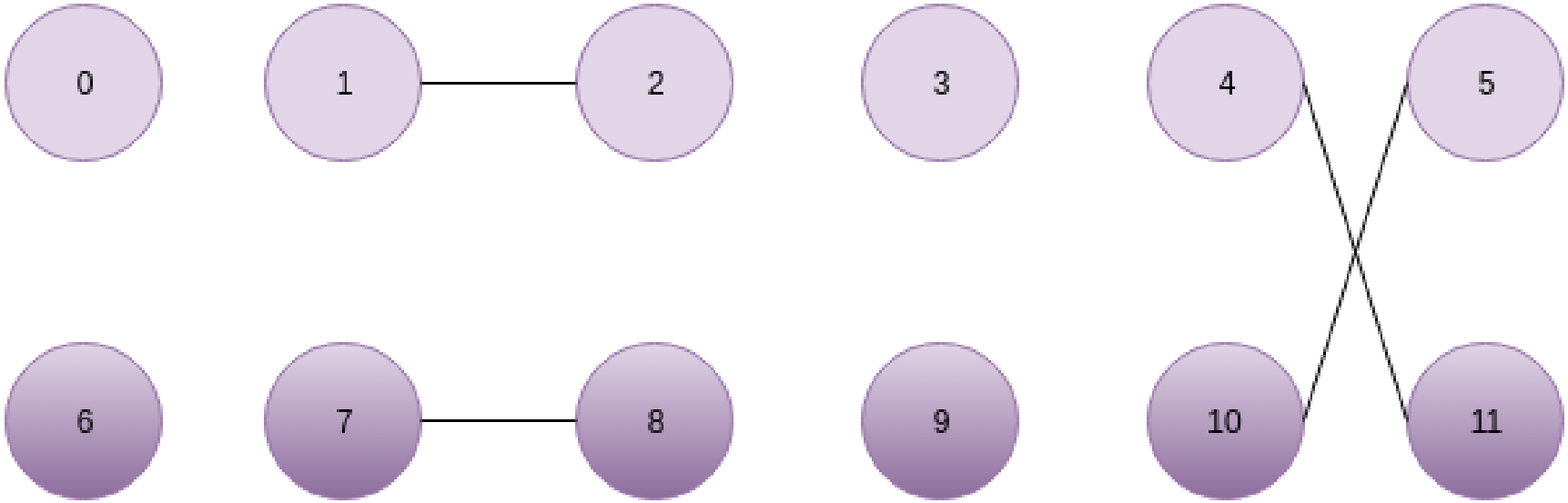
5. Alliances in Hogwarts

¿Amistad? Sencillo...



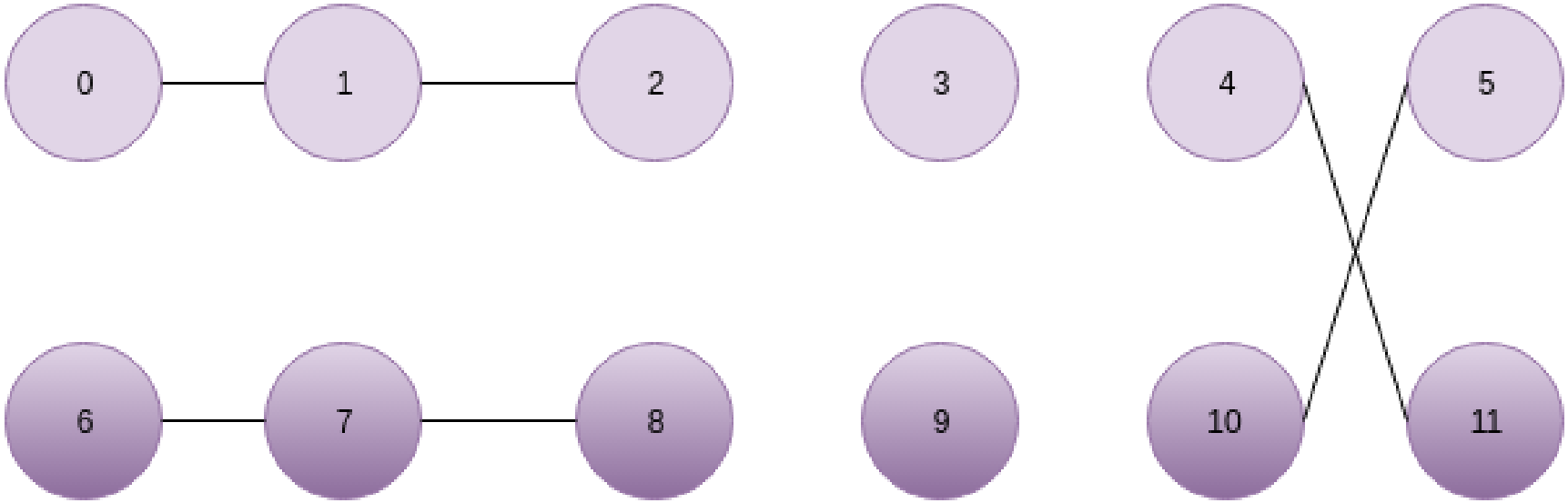
5. Alliances in Hogwarts

¿Enemistad? Interesante...



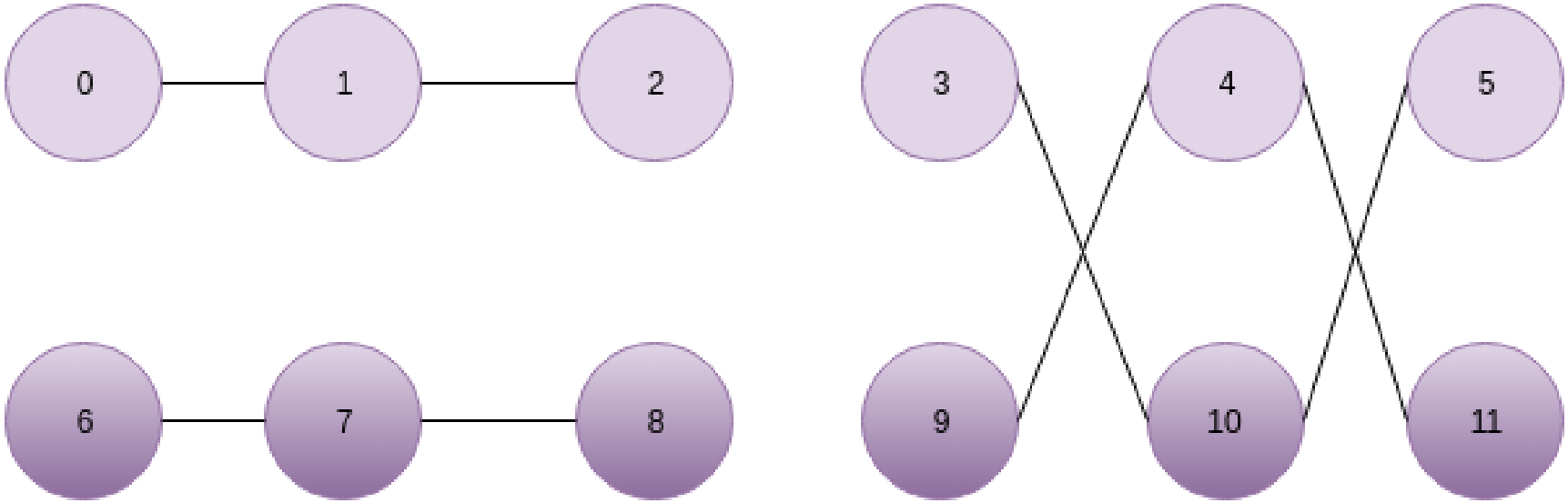
5. Alliances in Hogwarts

"Los amigos de mis amigos son mis amigos también"



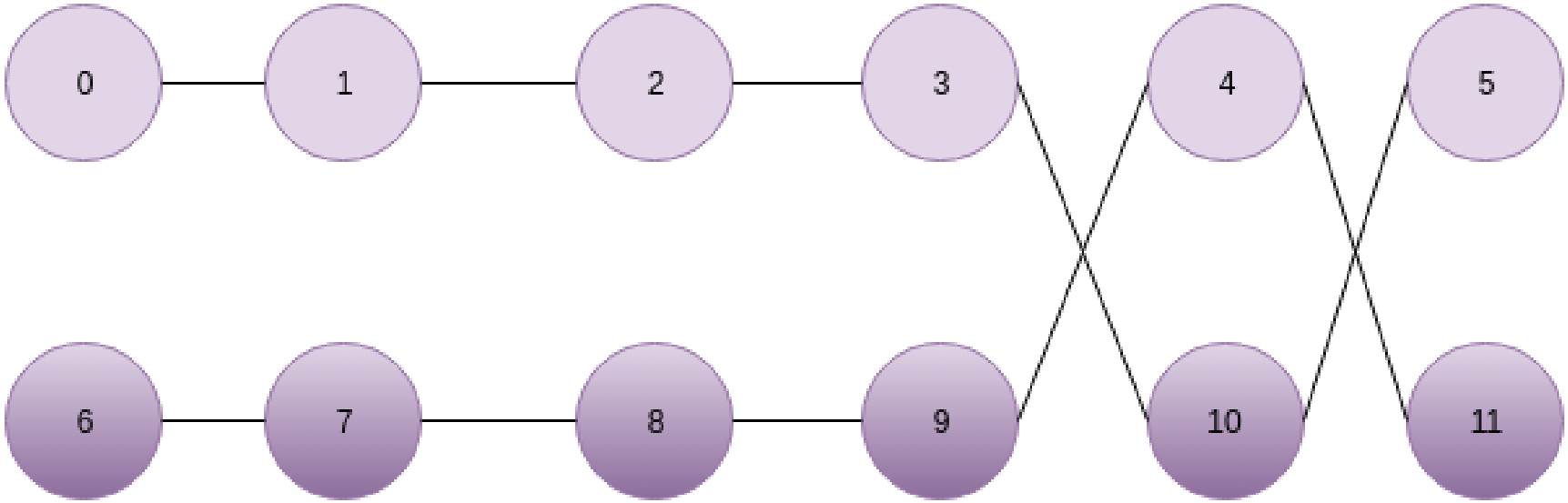
5. Alliances in Hogwarts

"El enemigo de mi enemigo es mi amigo"



5. Alliances in Hogwarts

"El enemigo de mi amigo es mi enemigo también"

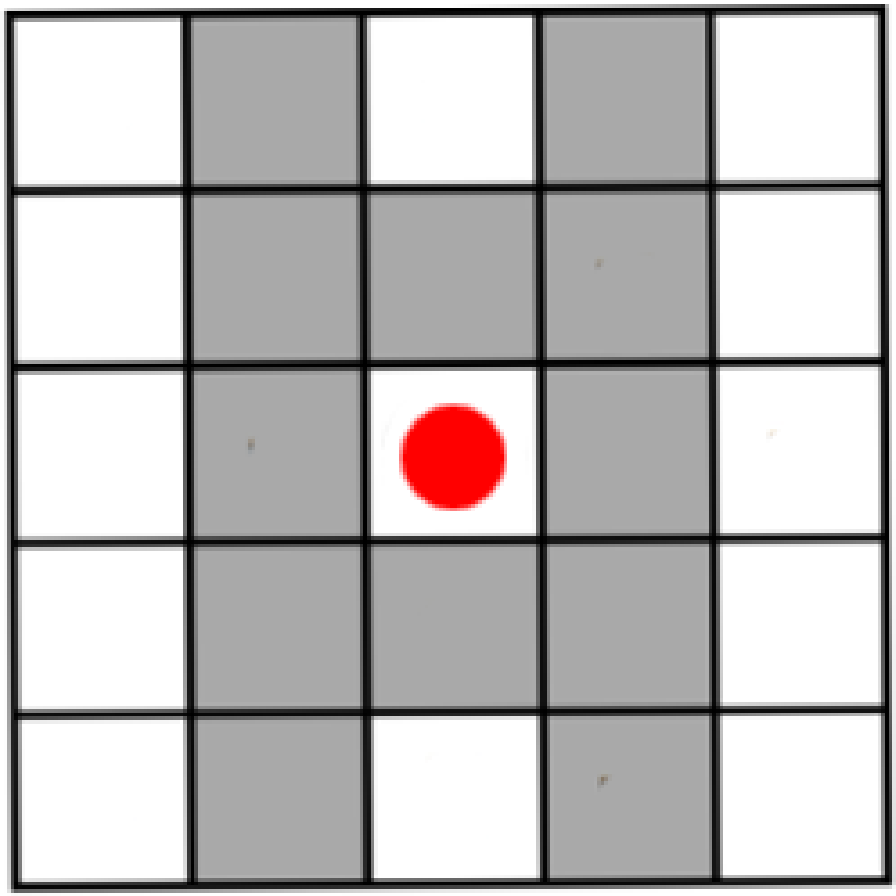


6. Vogons

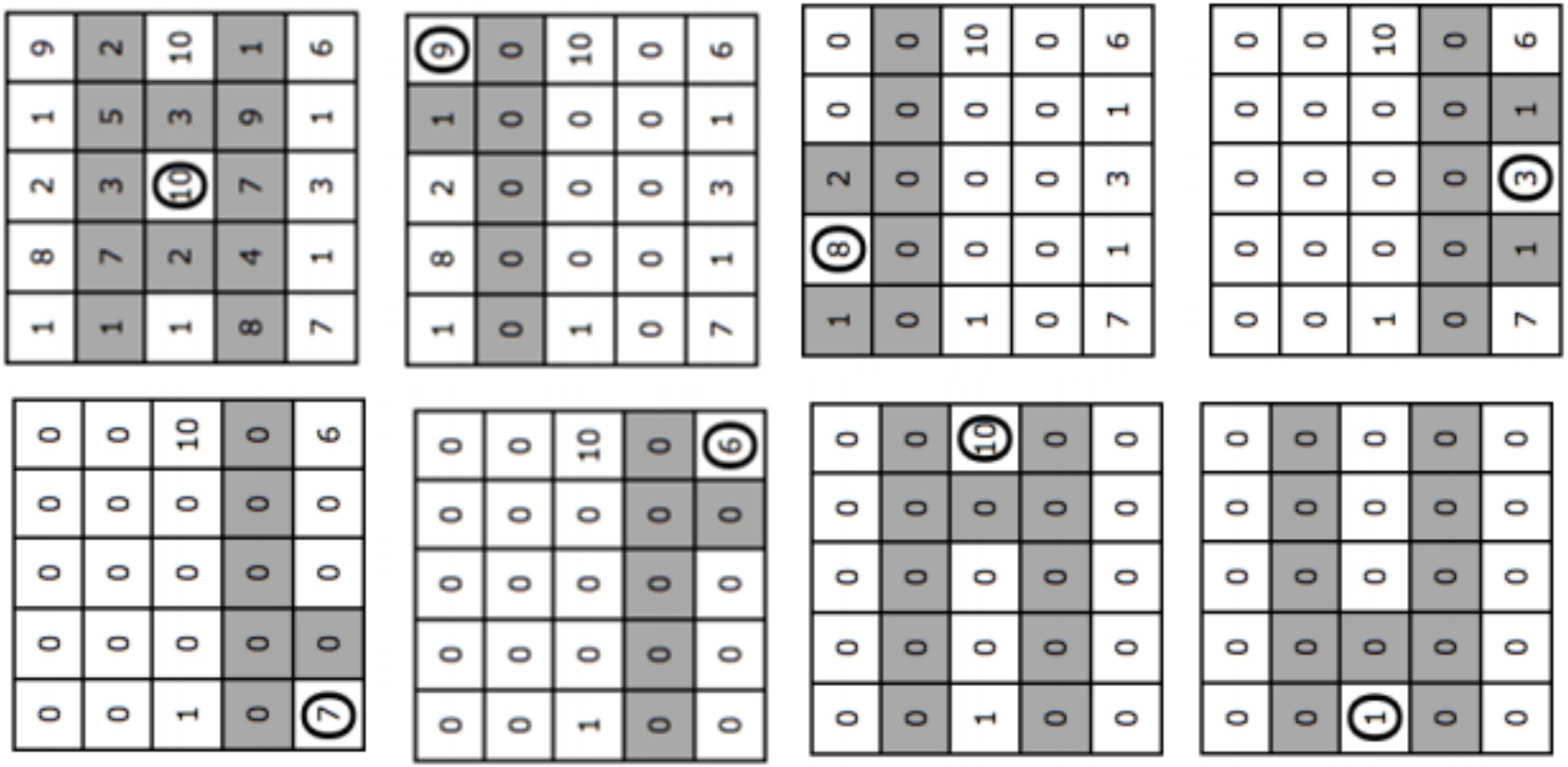
Los vogones destruirán una extensión de tierra con oro distribuido en una cuadricula, donde cada casilla tiene cierta cantidad de oro. El proceso que realizan es el siguiente:

- Selecciona un cuadrado no destruido.
- Extrae el oro que contiene.
- Destruye el cuadrado seleccionado.
- Destruye las casillas superior e inferior.
- Destruye todos los cuadrados de las columnas a la derecha e izquierda del cuadrado seleccionado.

6. Vogons



6. Vogons


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

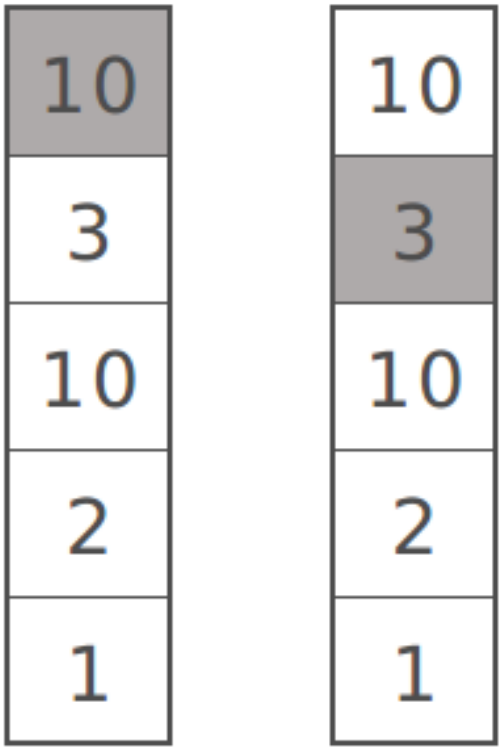
6. Vogons

Al tomar un valor, eliminamos toda la columna a su izquierda, y toda la columna a su derecha. Por lo tanto, nos interesaremos de momento SOLO en resolver el problema para una columna, y no para toda la cuadricula.

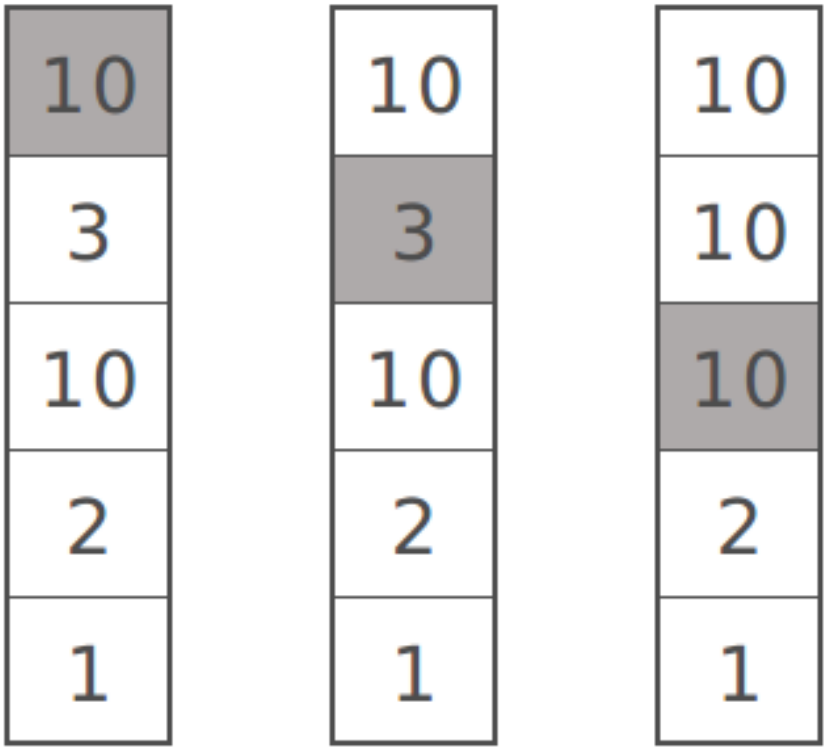
Al tomar un valor, eliminamos su casilla superior e inferior. Así pues, para cada columna debemos buscar cual es el máximo que se puede obtener.

6. Vogons

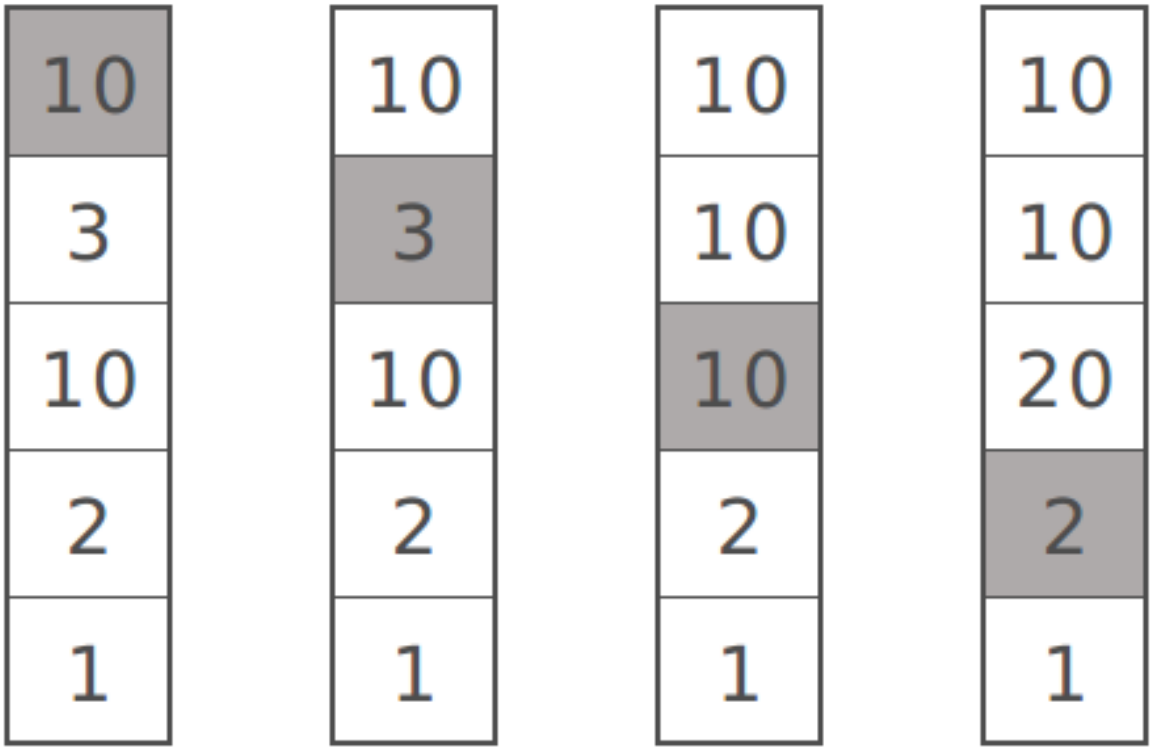
6. Vogons



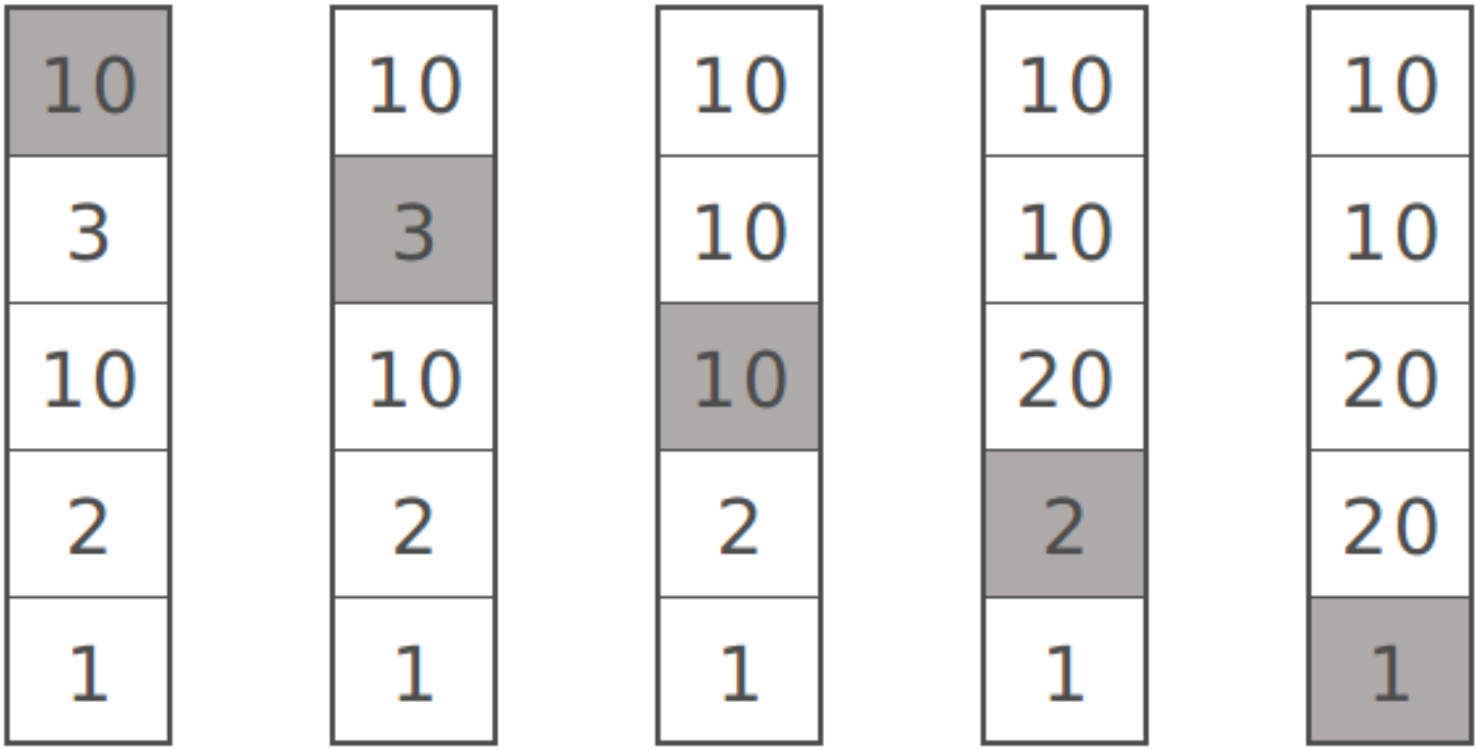
6. Vogons



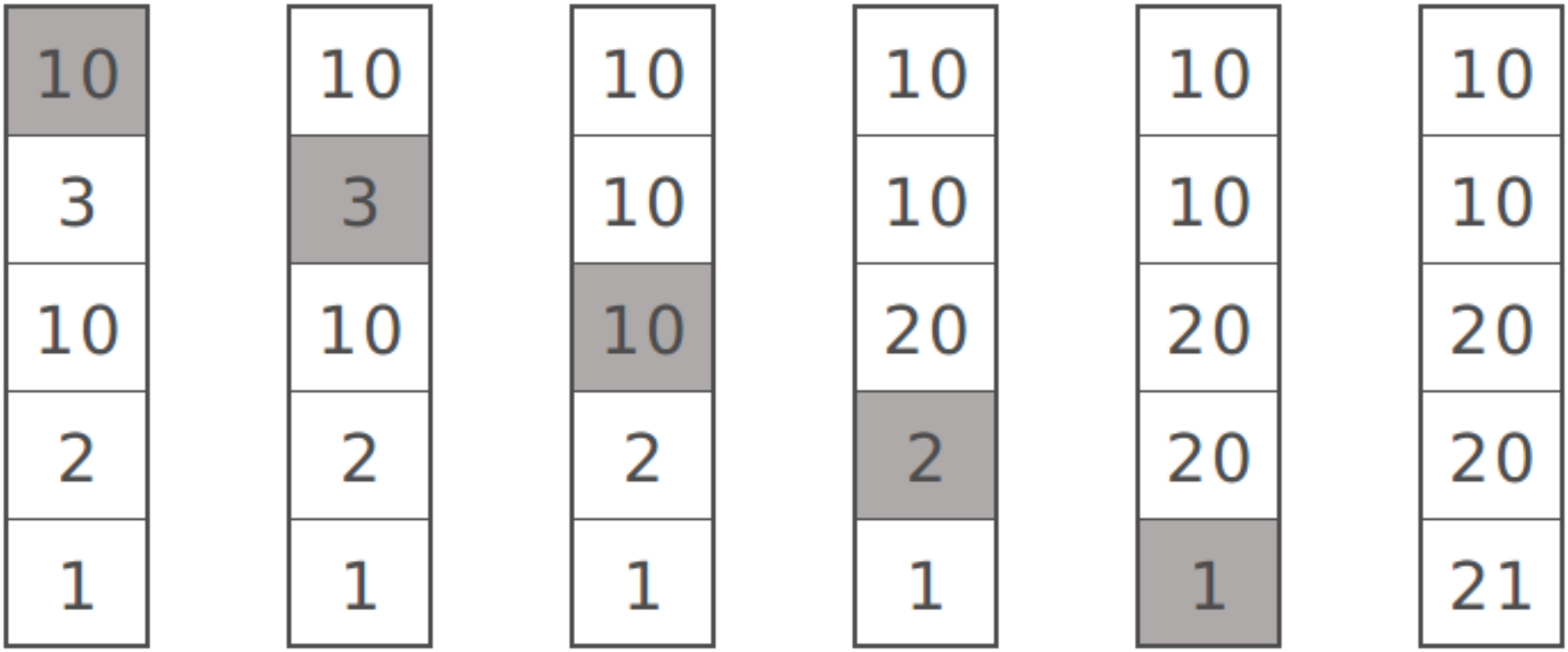
6. Vogons



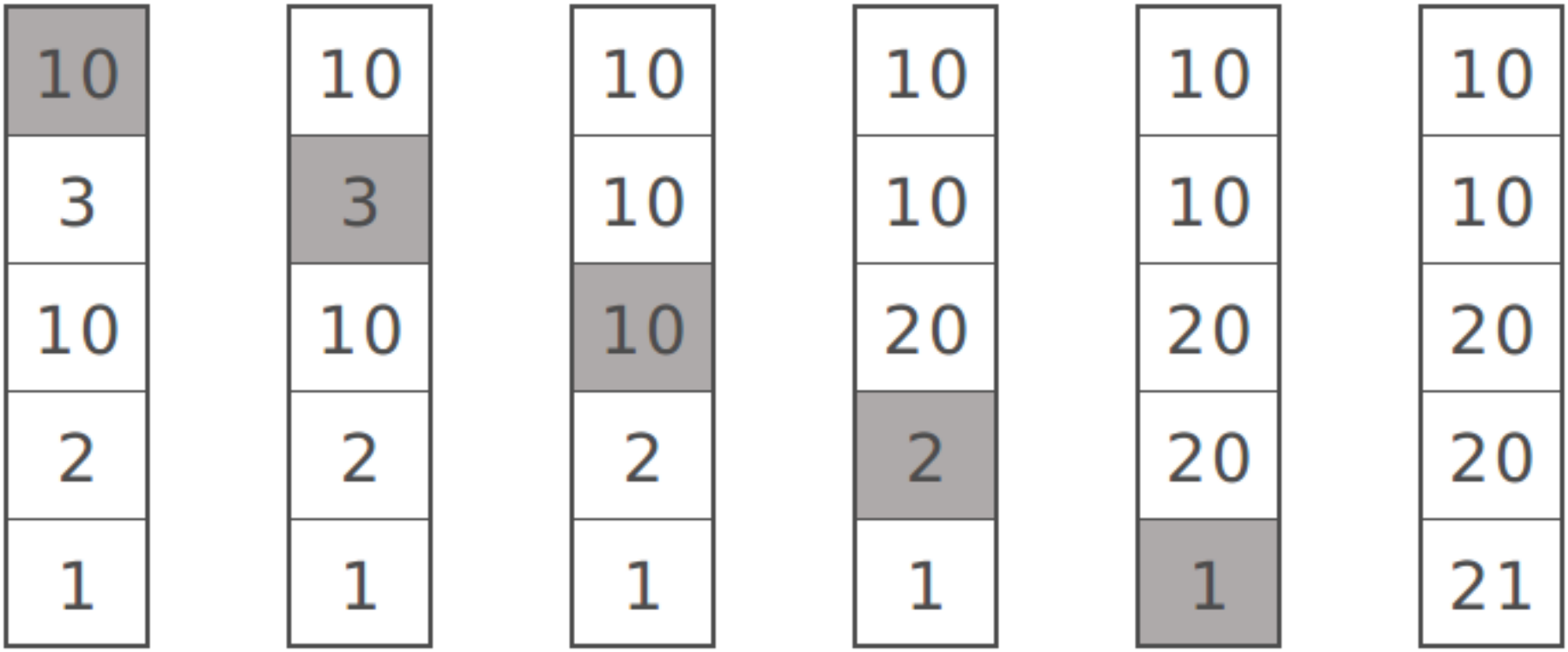
6. Vogons



6. Vogons



6. Vogons



$$D[k] = max(D[k-1], D[k] + D[k-2])$$

6. Vogons

Este proceso se realiza con cada una de las columnas. Al final, tendremos en la última casilla de cada columna el valor total de esa columna. Repetimos ahora el mismo procedimiento pero sobre esa fila.

6. Vogons

17	12	21	17	16

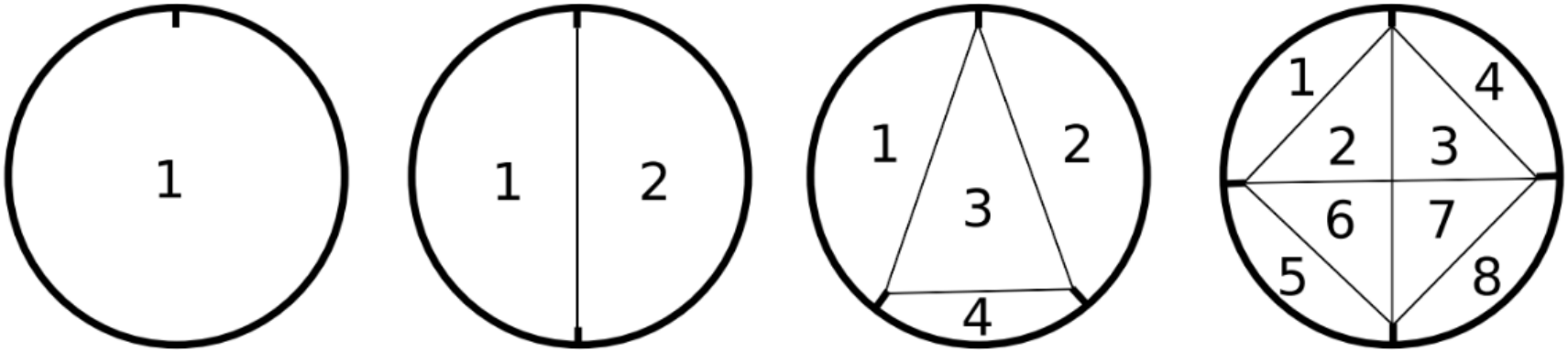
17	17	21	17	16

17	17	38	17	16

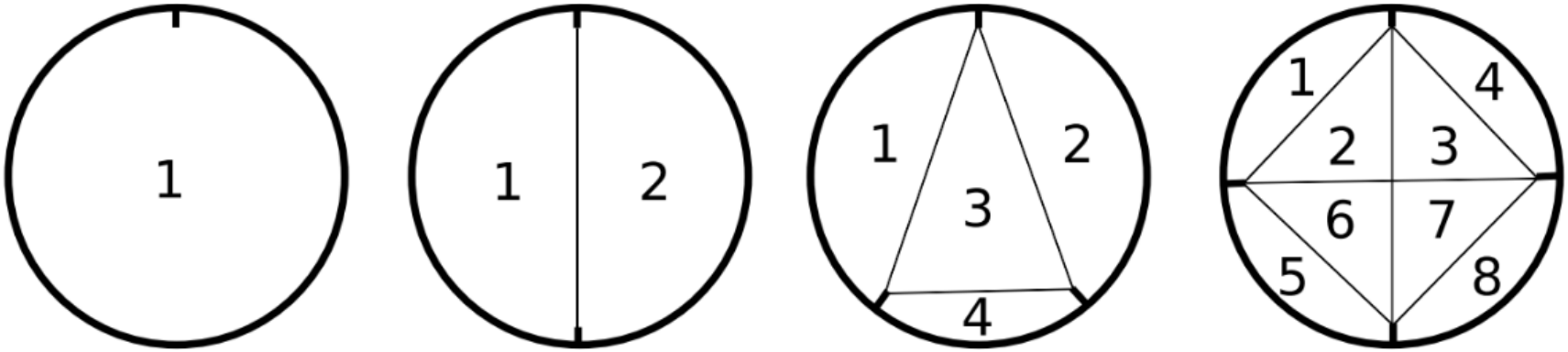
17	17	38	38	16

17	17	38	38	54

7. Juanma and the Drinking Fountains

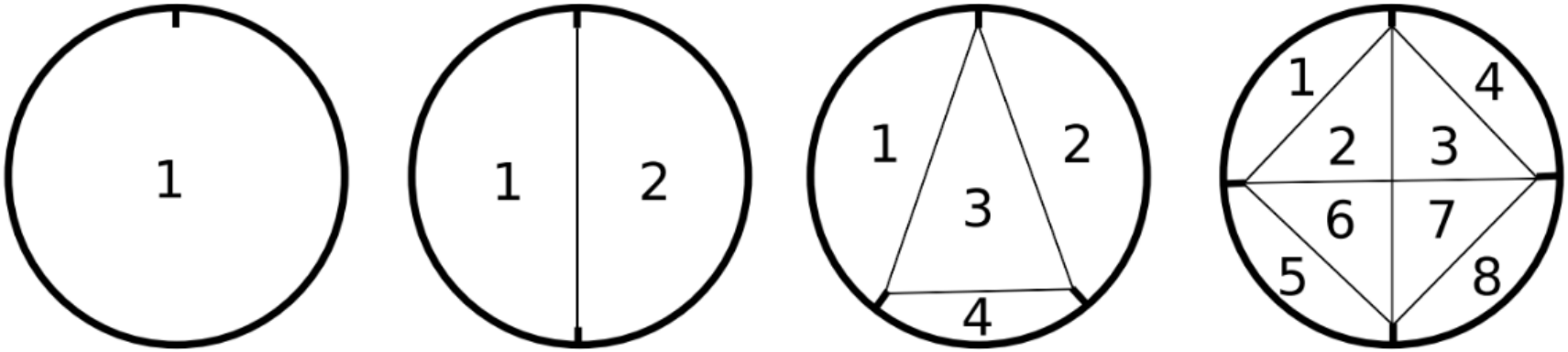


7. Juanma and the Drinking Fountains



Si tomamos n como el número de bebederos, la solución es el número de regiones 4espaciales formadas por n - 1 hiperplanos. ¡Eso es todo!

7. Juanma and the Drinking Fountains



Si tomamos n como el número de bebederos, la solución es el número de regiones 4espaciales formadas por n - 1 hiperplanos. ¡Eso es todo!

¡ES BROMA!

7. Juanma and the Drinking Fountains

Cada intersección se da entre dos lineas, cada una de las cuales se forma con dos bebederos. Por lo tanto, el número de intersecciones internas es igual al número de distintos subconjuntos de 4 puntos alrededor del círculo. Esto es C(n, 4).

7. Juanma and the Drinking Fountains

Cada intersección se da entre dos lineas, cada una de las cuales se forma con dos bebederos. Por lo tanto, el número de intersecciones internas es igual al número de distintos subconjuntos de 4 puntos alrededor del círculo. Esto es C(n, 4).

Si no hay bebederos (o hay solo uno), hay una sola región: la totalidad del círculo.

7. Juanma and the Drinking Fountains

Cada intersección se da entre dos lineas, cada una de las cuales se forma con dos bebederos. Por lo tanto, el número de intersecciones internas es igual al número de distintos subconjuntos de 4 puntos alrededor del círculo. Esto es C(n, 4).

Si no hay bebederos (o hay solo uno), hay una sola región: la totalidad del círculo.

Cada linea añade una nueva región, y dado que una linea se forma con dos bebederos, el número de lineas es igual al número de distintos subconjuntos de 2 puntos alrededor del circulo. Esto es C(n,2).

Entonces tenemos: C(n,4) + C(n,2) + 1

7. Juanma and the Drinking Fountains

$C(n,k) = \frac{n!}{k!(n-k)!}$

7. Juanma and the Drinking Fountains

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

$$C(n,2) = \frac{n(n-1)(n-2)(n-3)...(2)(1)}{2(n-2)(n-3)...(2)(1)}$$

7. Juanma and the Drinking Fountains

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

$$C(n,2) = \frac{n(n-1)(n-2)(n-3)...(2)(1)}{2(n-2)(n-3)...(2)(1)}$$

$$C(n,2) = \frac{n(n-1)}{2}$$

7. Juanma and the Drinking Fountains

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

$$C(n,2) = \frac{n(n-1)(n-2)(n-3)...(2)(1)}{2(n-2)(n-3)...(2)(1)}$$

$$C(n,2) = \frac{n(n-1)}{2}$$

$$C(n,4) = \frac{n(n-1)(n-2)(n-3)(n-4)(n-5)...(2)(1)}{4!(n-4)(n-5)...(2)(1)}$$

7. Juanma and the Drinking Fountains

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

$$C(n,2) = \frac{n(n-1)(n-2)(n-3)...(2)(1)}{2(n-2)(n-3)...(2)(1)}$$

$$C(n,2) = \frac{n(n-1)}{2}$$

$$C(n,4) = \frac{n(n-1)(n-2)(n-3)(n-4)(n-5)...(2)(1)}{4!(n-4)(n-5)...(2)(1)}$$

$$C(n,4) = \frac{n(n-1)(n-2)(n-3)}{24}$$

7. Juanma and the Drinking Fountains

Solución:

$$\frac{n(n-1)(n-2)(n-3)}{24} + \frac{n(n-1)}{2} + 1$$

$$\frac{(n^4-6n^3+23n^2-18n+24)}{24}$$

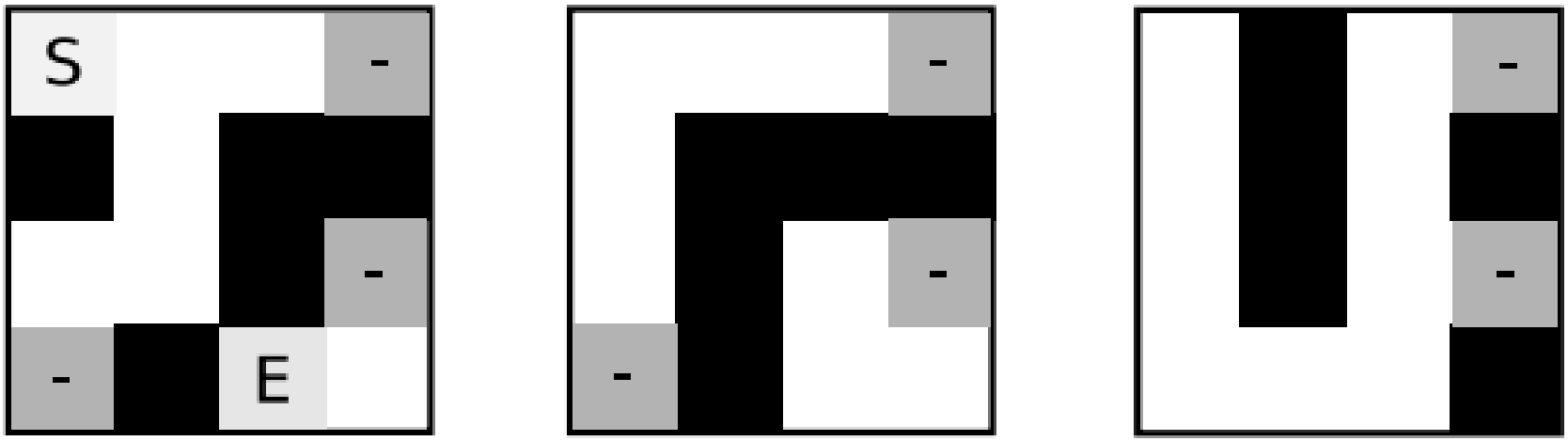
8. Are you ok?

Nos dan el mapa de una región como una lista de puntos que conforman un polígono. Luego indican las personas a buscar en el mapa, para decir si están en la región o no. Es decir, lo que preguntan es, mirar si cada punto (persona) está en el polígono (mapa), y decir la cantidad de personas que cumplieron la condición.

9. Multistory Labyrinth

Encontrar la ruta mas corta desde el inicio hasta el final del laberinto, o imprimir -1 en caso contrario. Se resuelve con un BFS (Busqueda en profundidad) simple.

9. Multistory Labyrinth



10. The price is correct

Un participante corre por un tablero cuadriculado, en el cual encienden luces con diferentes premios. Cada vez que el participante pisa una luz recibe ese premio. El participante solo puede moverse arriba/abajo/derecha/izquierda o mantenerse quieto en cada segundo. Sabiendo en que momento se prende cada luz y en que posición del tablero está dicha luz, debe decirse cual es la mayor cantidad de premios que el concursante puede ganar.

10. The price is correct

Se resuelve por programación dinamica.

$f(x, y, t) = MAX(\text{todos los posibles movimientos}) + valor[x][y][t]$

11. Josephus lottery II

Humbertov toma la lista de estudiantes y comienza a contar desde 1 hasta k (en el sentido de las manecillas del reloj), el estudiante con el número k es removido de la lista. Humbertov comienza a contar nuevamente desde 1 hasta k comenzando en el siguiente estudiante pero... esta vez en el sentido contrario a las manecillas del reloj. Humbertov repetirá este procedimiento hasta que solo quede 1 estudiante.

- La lista es de 100 estudiantes.
- Conocemos la posición inicial m de Pepito.

Debemos encontrar el valor mínimo k ($1 \leq k \leq 1000$) para el cual Pepito ganará la rifa.

11. Josephus lottery II

Este es un proceso de simulación. Debemos simular la situación que nos presenta el problema, pero si realizamos esta simulación por cada caso de prueba probablemente terminemos con un TLE, así que ¿Cómo realizamos una simulación que no termine en un TLE?

11. Josephus lottery II

Este es un proceso de simulación. Debemos simular la situación que nos presenta el problema, pero si realizamos esta simulación por cada caso de prueba probablemente terminemos con un TLE, así que ¿Cómo realizamos una simulación que no termine en un TLE?

SOLUCIÓN: Precalculo.

11. Josephus lottery II

Este es un proceso de simulación. Debemos simular la situación que nos presenta el problema, pero si realizamos esta simulación por cada caso de prueba probablemente terminemos con un TLE, así que ¿Cómo realizamos una simulación que no termine en un TLE?

SOLUCIÓN: Precalculo.

Sabemos de antemano que la lista de estudiantes siempre será de 100, por lo tanto la posición m de pepito estará siempre entre 1 y 100.

11. Josephus lottery II

Este es un proceso de simulación. Debemos simular la situación que nos presenta el problema, pero si realizamos esta simulación por cada caso de prueba probablemente terminemos con un TLE, así que ¿Cómo realizamos una simulación que no termine en un TLE?

SOLUCIÓN: Precalculo.

Sabemos de antemano que la lista de estudiantes siempre será de 100, por lo tanto la posición m de pepito estará siempre entre 1 y 100. Conocemos también que el valor máximo de k será de 1000.

11. Josephus lottery II

```
1  static int solve(int k) {
2      ArrayList<Integer> values = new ArrayList<>();
3      for (int i = 0; i < 100; i++) {
4        values.add(i + 1);
5      }
6
7      /*Simulacion*/
8
9      return values.get(0);
10   }
```

11. Josephus lottery II

```
1     int currentIndex = 0;  boolean clockwise = true;
2
3     while (values.size() > 1) {
4       if (clockwise) {
5         currentIndex += k - 1;
6       } else {
7         currentIndex -= k - 1;
8       }
9       currentIndex += 1000 * values.size();
10      currentIndex %= values.size();
11      values.remove(currentIndex);
12      if (!clockwise) {
13        currentIndex -= 1;
14        currentIndex += 1000 * values.size();
15      }
16      currentIndex %= values.size();
17      clockwise = !clockwise;
18    }
```

11. Josephus lottery II

Teniendo la simulación solo nos queda conocer cual será el menor k para cada posible posición m de Pepito en la lista.

```
1
2       int[] best = new int[101];
3       Arrays.fill(best, Integer.MAX_VALUE);
4       for (int i = 1; i <= 1000; i++) {
5         int v = solve(i);
6         best[v] = Math.min(best[v], i);
7       }
```

11. Josephus lottery II

Teniendo la simulación solo nos queda conocer cual será el menor k para cada posible posición m de Pepito en la lista.

```
1
2       int[] best = new int[101];
3       Arrays.fill(best, Integer.MAX_VALUE);
4       for (int i = 1; i <= 1000; i++) {
5         int v = solve(i);
6         best[v] = Math.min(best[v], i);
7       }
```

Ya con esto resolver cada uno de los casos es tan sencillo como hacer esto:

11. Josephus lottery II

Teniendo la simulación solo nos queda conocer cual será el menor k para cada posible posición m de Pepito en la lista.

```
1
2       int[] best = new int[101];
3       Arrays.fill(best, Integer.MAX_VALUE);
4       for (int i = 1; i <= 1000; i++) {
5         int v = solve(i);
6         best[v] = Math.min(best[v], i);
7       }
```

Ya con esto resolver cada uno de los casos es tan sencillo como hacer esto:

```
1         int v = sc.nextInt();
2         System.out.println(best[v]);
```

12. Funny day in Playland

En playland tenemos atracciones, categorias y tickets... Arya quiere recorrer todas las atracciones de Playland sin repetir ninguna.

- Las atracciones pertenecen a mínimo 1 y máximo 2 categorias.
- Para ingresar a una atracción Arya debe entregar un ticket de una de las categorias a la que pertenece la atracción.
- Cuando Arya se baja de la atracción recibe un ticket de la otra categoria a la que pertenece la atracción (o de la misma si la atracción solo pertenece a una categoria).
- Arya puede elegir en que atracción comienza.

12. Funny day in Playland

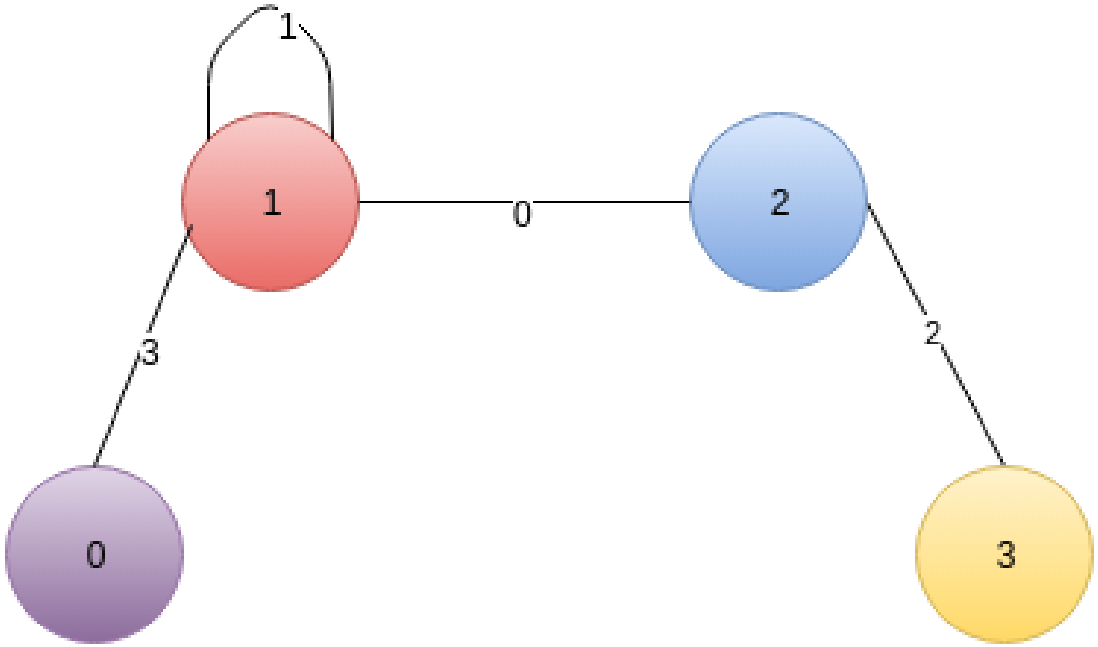
¿Cómo representar Playland?

12. Funny day in Playland

¿Cómo representar Playland? Correcto, a traves de un grafo, donde las categorias son los nodos y las atracciones son los arcos, y cada arco tiene un número que lo identifica.

12. Funny day in Playland

¿Cómo representar Playland? Correcto, a traves de un grafo, donde las categorias son los nodos y las atracciones son los arcos, y cada arco tiene un número que lo identifica.



12. Funny day in Playland

¿Que debemos hacer? Verificar si en Playland es un grafo Euleriano :)

12. Funny day in Playland

¿Que debemos hacer? Verificar si en Playland es un grafo Euleriano :)

TIP: Un grafo euleriano es aquel que cuenta con un camino en el cual se pasa por cada arista una sola vez.

12. Funny day in Playland

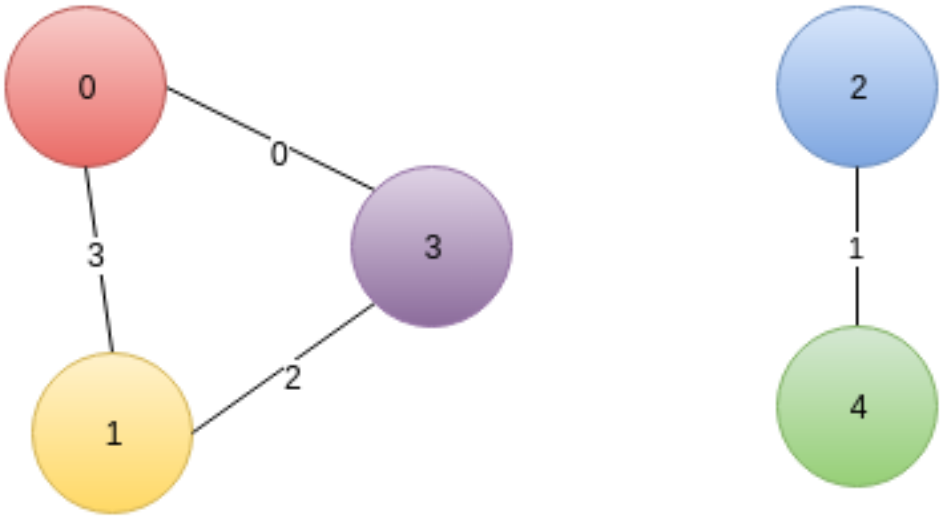
¿Que debemos hacer? Verificar si en Playland es un grafo Euleriano :)

TIP: Un grafo euleriano es aquel que cuenta con un camino en el cual se pasa por cada arista una sola vez.

Veamos un poco de teoria de grafos. Existen 3 condiciones que me permiten determinar si un grafo es Euleriano o no. Estas son...

12. Funny day in Playland

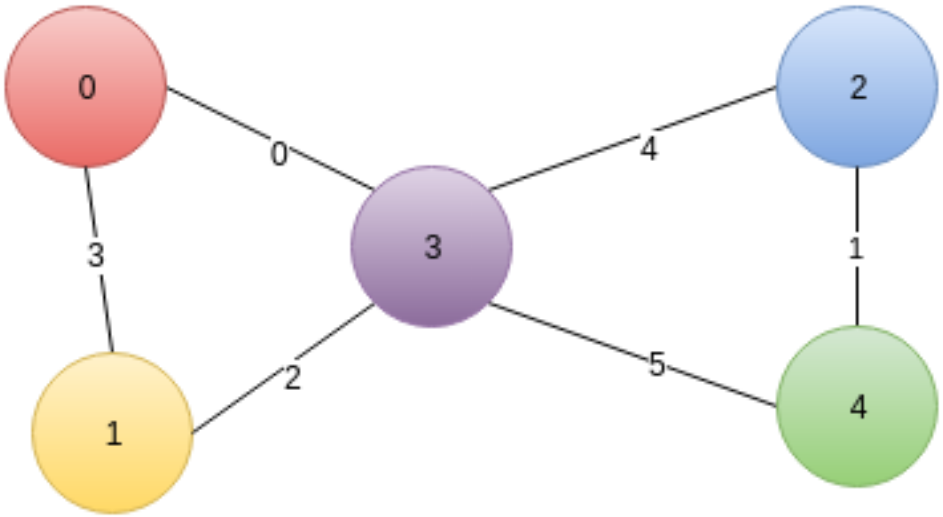
1. El grafo es conexo.



Lo primero que debiamos hacer era validar que el grafo era conexo, esto podiamos hacerlo a traves de un DFS o un BFS. Si el grafo no era conexo podiamos facilmente imprimir un -1.

12. Funny day in Playland

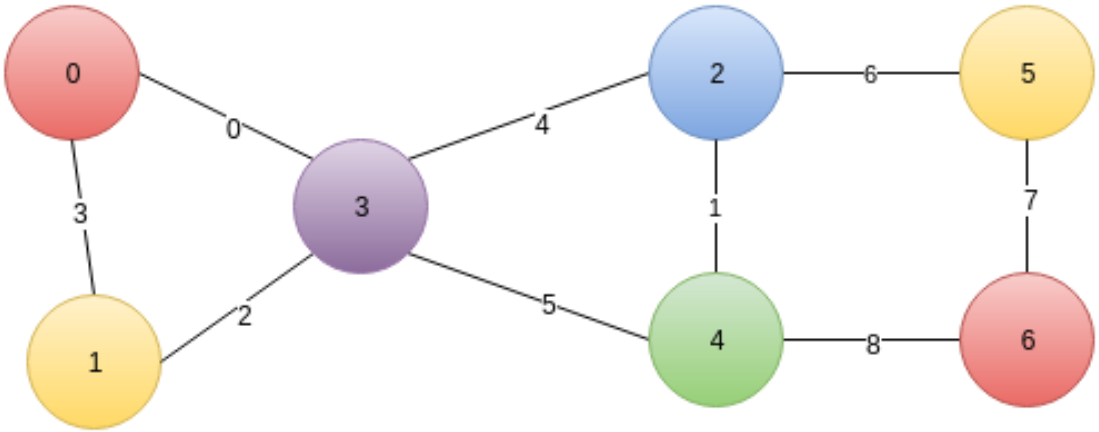
2. El grado de todos los nodos del grafo es un número par.



Si el grafo era conexo y no existen nodos de grado impar Arya podia empezar su recorrido desde cualquier nodo, por lo tanto la respuesta en este caso era siempre 0.

12. Funny day in Playland

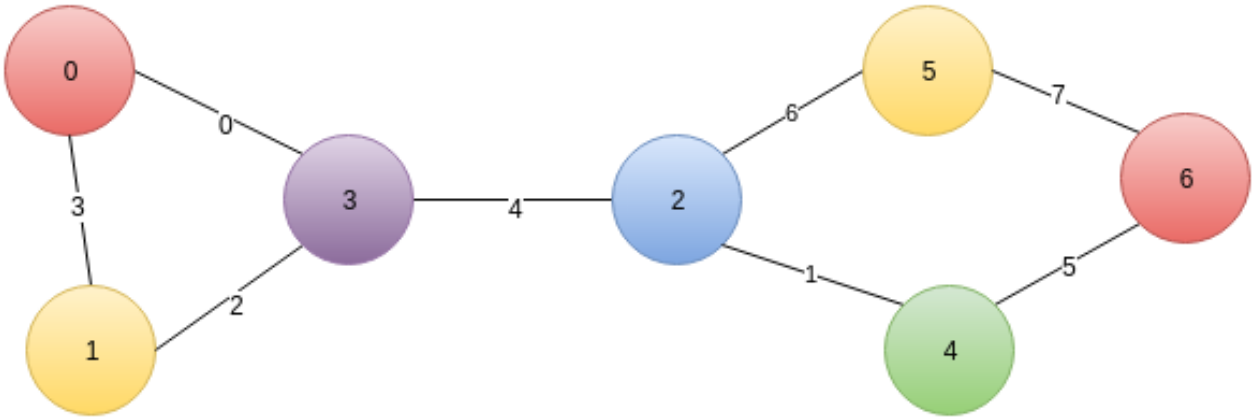
3. Existen 2 y solo 2 nodos de grado impar.



Si la cantidad de nodos impares era diferente de 0 o de 2 Arya no puede realizar el recorrido, por lo tanto la respuesta era -1.

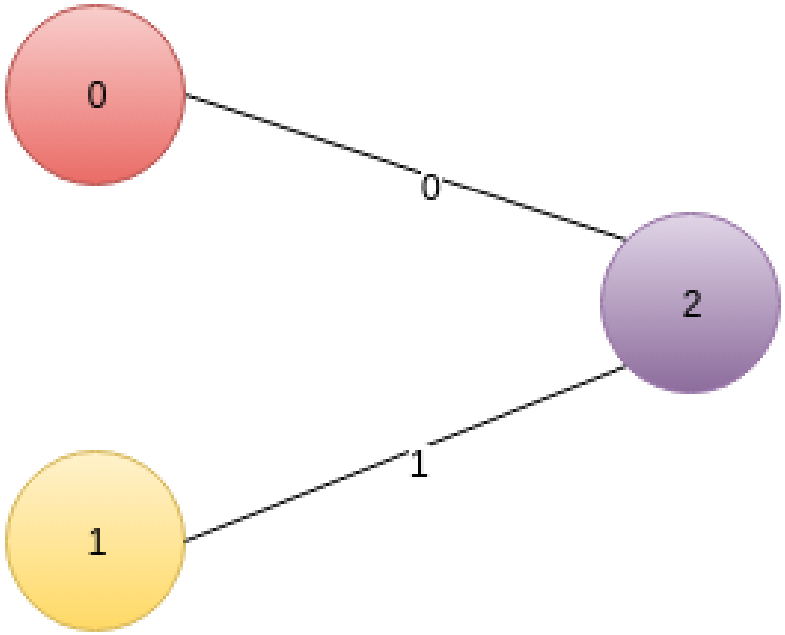
12. Funny day in Playland

Si el grafo es Euleriano y existen 2 nodos de grado impar, se deben revisar cada una de las aristas de los nodos de grado impar para elegir aquella de menor indice, teniendo en cuenta que la arista elegida no debe ser un puente (en lo posible).



12. Funny day in Playland

Pero no todo es tan bonito... siempre hay casos especiales y hay ocasiones en las que si o si debemos elegir una arista puente.



12. Funny day in Playland

¿Cómo determinar los puentes de un grafo?

12. Funny day in Playland

¿Cómo determinar los puentes de un grafo?

Esto podemos hacerlo a través del Algoritmo de Tarjan, que es una leve modificación de la búsqueda en profundidad DFS :)