# DOCUMENTO DEL MÉTODO DE LA INGENIERÍA

**FASE 1**

**Identificación del problema:**

Un grupo de universitarios tienen la tarea de crear un juego de búsqueda del tesoro. El jugador visualiza un tablero de 50x50 casillas, para iniciar selecciona la casilla inicial, una vez iniciado, el jugador debe lanzar un dado de seis caras, dependiendo del valor que resulte del lanzamiento (de 1 a 6) el jugador se podrá mover entre las casillas. Para ayudar al jugador en su búsqueda, el programa es capaz de calcular el camino más corto que lleva al tesoro. El jugador tiene la habilidad de “cavar” en una de las casillas adyacentes, en la cual se pueden descubrir monedas, vidas, trampas o el tesoro, al empezar, el jugador tiene 5 vidas, si el jugador encuentra una vida se le otorga una extra, pero si encuentra una trampa, pierde una vida. El juego concluye una vez el jugador encuentra el tesoro o pierde todas sus vidas.

Una vez el juego finaliza, el jugador es capaz de ver el tiempo total que le tomó el juego desde el momento en que el jugador seleccionó la casilla inicial hasta la conclusión del juego, cada moneda que haya recolectado le resta 15 segundos al tiempo final, reduciendo su tiempo hasta llegar a 0.

Además de lo mencionado anteriormente, el usuario tendrá la opción de pedir ayuda, esta ayuda le mostrará al usuario el camino más corto al tesoro.

**Fase 2.**

**Búsqueda de posibles soluciones:**

Tras realizar nuestro análisis concluimos que es necesario tener una estructura en la que podamos almacenar por cada casilla varios datos, ya que no solo tenemos que contar con el hecho de que haya ítems dentro de las casillas, además hay que contar con el hecho de que el usuario puede ponerse en esta casilla, por lo tanto la estructura que elijamos debe de ser capaz de almacenar varios datos a la vez mediante un orden y que se pueda acceder a estos datos de manera sencilla.

Tenemos 3 candidatos para la estructura que hará de tablero:

-ArrayList

-LinkedList

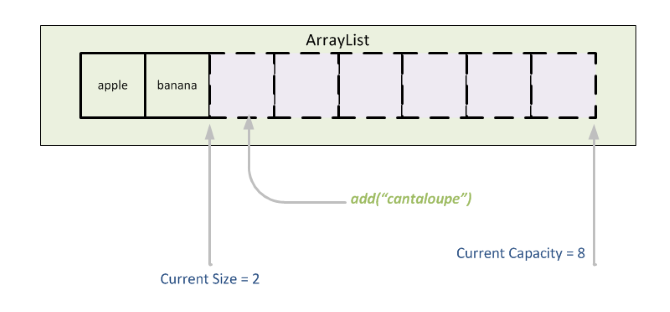
-Graph

ArrayList

The ArrayList class is a resizable [array](https://www.w3schools.com/java/java_arrays.asp), which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one).

[W3schools - ArrayList](https://www.w3schools.com/java/java_arraylist.asp)



*Ilustración de cómo actúa un ArrayList*

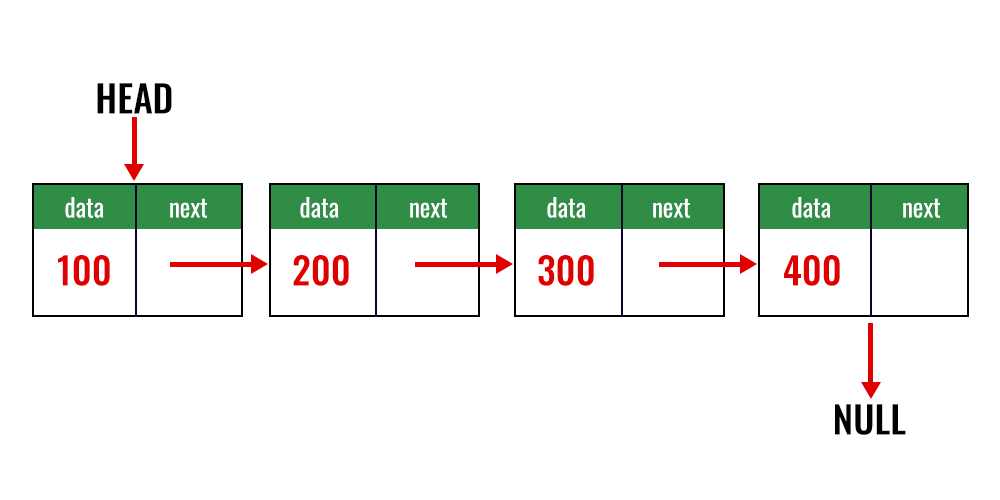
Podríamos usar ArrayList gracias a la capacidad que tiene de almacenar objetos de manera ordenada, característica que estamos buscando para nuestra estructura. ArrayList almacena datos dentro de un Array, pero en vez de ser estático se le puede cambiar el tamaño al ArrayList para que almacene n elementos.

Además de lo mencionado anteriormente, los métodos de ArrayList tienen un tiempo de recorrida máximo de BigO(n), por lo que es una estructura relativamente rápida.

LinkedList

The LinkedList stores its items in "containers." The list has a link to the first container and each container has a link to the next container in the list. To add an element to the list, the element is placed into a new container and that container is linked to one of the other containers in the list.

[W3schools - LinkedList](https://www.w3schools.com/java/java_linkedlist.asp)



*Ilustración de cómo se ve una LinkedList simple*

La LinkedList es una manera más sofisticada de hacer el tablero comparada al ArrayList, ya que la forma en la que se construye esta estructura nos permite tener una idea más gráfica y ordenada de los datos que tenemos, además, nos podemos permitir modificar la propia estructura para que funcione de la manera que queramos, además sabemos que LinkedList se utiliza para realizar tableros tales como Snakes & Ladders, por lo tanto, se puede transferir bien a un nuevo tablero.

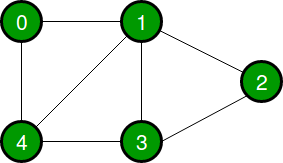
Lo mejor de todo es que esta estructura tiene un tiempo máximo de BigO(n), siendo el mismo tiempo de ejecución que el ArrayList.

Graph

A graph is a [data structure](https://www.javatpoint.com/data-structure-tutorial) that stores connected data. In other words, a graph G (or g) is defined as a set of vertices (V) and edges (E) that connects vertices. The examples of graph are a social media network, computer network, Google Maps, etc.

Each graph consists of edges and vertices (also called nodes). Each vertex and edge have a relation. Where vertex represents the data and edge represents the relation between them. Vertex is denoted by a circle with a label on them. Edges are denoted by a line that connects nodes (vertices).

[javaTpoint - graph](https://www.javatpoint.com/java-graph)

****

***Ilustración de un grafo dirigido***

El grafo es una estructura altamente modificable que aprovecha el concepto de LinkedList de conectar nodos que contienen información con otros nodos y lo aplica con vértices y aristas, convirtiendo en una versión aún más sofisticada que las anteriores estructuras.

El tiempo de ejecución varía, pero por regla general termina siendo BigO(V+E), siendo V el número de vértices y E el número de aristas.

**Fase 3.**

**Generación de ideas para una solución creativa:**

**Método para generación de ideas: Brainstorming.**

**¿Qué estructuras podríamos usar para resolver el problema?**

Establecimos mediante investigación que algunas posibles estructuras son ArrayList, LinkedList y Grafos, no hay necesidad de hacer brainstorm.

**Implementación de recorrido más corto:**

-Al inicio del juego definir un contador de acciones que tiene que hacer el usuario para llegar al tesoro.

-Usar un algoritmo ya existente (Dependiendo de cuál estructura terminamos eligiendo).

-Implementar un guía que le muestre el paso óptimo, pero no los pasos completos.

-Al pedir ayuda redirigirlo a una página en la que pueda ver una guía de cómo encontrar el camino más corto.

**Mecánica de cavar:**

-Colocar un Array en cada casilla que almacene un objeto del tipo de ítem.

-Asignarle unos atributos booleanos a cada casilla para definir si hay un ítem dentro o no.

-Poner un objeto para el ítem dentro de la casilla como un atributo.

-Usar enumeración para definir los ítems que pueden llegar a estar dentro de la casilla, usando booleanos para reflejar los contenidos.

**Fase 4.**

**Descarte de ideas no viables:**

**Estructuras para el tablero:**

**ArrayList**

Llegamos a la conclusión de que el ArrayList no se puede utilizar para crear el tablero debido a la baja capacidad de modificación de la estructura, además, a la hora de trabajar con ArrayList el manejo de datos queda muy incómodo, por estas razones el ArrayList queda completamente descartado.

**Implementación de camino más corto:**

**-Implementar un guía que le muestre el paso óptimo, pero no los pasos completos.**

Esta posible solución es relativamente sencilla, sin embargo, llegamos a la conclusión de que queremos que el usuario tenga acceso completo a la solución mediante la ayuda, esta posible solución tiene más una funcionalidad de pista que de un camino más corto, por lo tanto, descartamos esta posible solución.

**-Al pedir ayuda redirigirlo a una página en la que pueda ver una guía de cómo encontrar el camino más corto.**

Termina siendo muy engorroso, además de tener que crear una página se debe de subir un video en el que se explique la funcionalidad entera del juego, lo cual el usuario es muy probable que no desee, ya que muchos usuarios les gusta averiguar las cosas por ellos mismos, por lo dicho anteriormente, esta posible solución queda descartada.

**Mecánica de cavar:**

**-Poner un objeto para el ítem dentro de la casilla como un atributo.**

Esta solución implica la creación de potencialmente cientos de objetos lo cual generaría mucho tiempo de ejecución extra, además, los ítems no llegan a tener atributos únicos, por lo tanto, no es necesario crear un objeto solo para un ítem específico, por lo dicho anteriormente descartamos esta posible solución.

**-Colocar un Array en cada casilla que almacene un objeto del tipo de ítem.**

Para esta solución tenemos un problema similar para el anterior respecto al hecho de que hay que crear un array por cada casilla, si estamos trabajando con un tablero de 50x50 eso implica cientos de Arrays, eso sin contar los que hacen otras funcionalidades, lo peor de todo es que muchos de esos Array nunca se usarían, ya que no todas las casillas tienen un ítem dentro, por lo que sería un atributo nulo, lo cual no es lo ideal, por las razones dadas anteriormente, esta posible solución es descartada.

**Fase 5.**

**Selección de soluciones:**

Tras realizar el proceso debido, tenemos unas cuantas posibles soluciones, la manera en la que definimos cuales se harán es mediante un sistema de criterios calificados del 1 al 10, el que consigue la mayor cantidad de puntos en su debida categoría es la solución que se hará, lo criterios son:

**Viabilidad: ¿**Qué tan plausible es que lo podamos realizar?

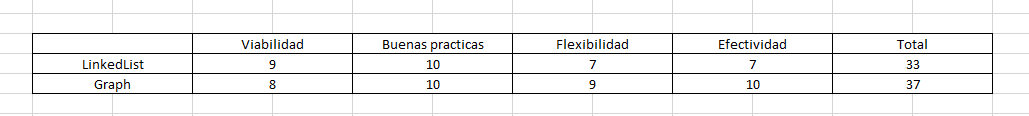
**Buenas prácticas:** ¿Lo que implica la solución cae en las buenas prácticas?

**Flexibilidad:** ¿Deja espacio para nuevas mecánicas e implementaciones?

**Efectividad:** Relación entre fácil acceso y qué tan efectivo es al manipular datos (Exclusivo para las estructuras)

Llegamos a un consenso para cada uno de los puntajes, los cuales quedaron así:

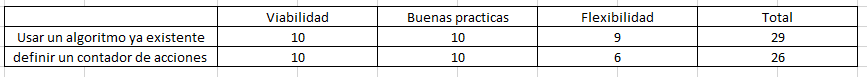
**Estructura para el tablero:**

****

LinkedList: 33 puntos

Graph: 37 puntos

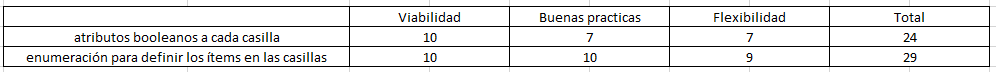
**Camino más corto:**



Usar un algoritmo ya existente: 29 puntos

Definir un contador de acciones: 26 puntos

**Mecánica de cavar:**

****

atributos booleanos a cada casilla: 24 puntos

enumeración para definir los ítems en las casillas: 29 puntos

En conclusión, usaremos un grafo para realizar el tablero, usaremos un algoritmo existente de grafos para realizar el recorrido de camino mínimo y usaremos enumeración para cuando hagamos la mecánica de excavar.