UT6_PD1

Property	COMPLETED
Date	@June 5, 2022
BLOCKED	

Primera medicion:

📌 Medicion: MedicionBuscarTrie - Consumo de memoria=10.450.571 Bytes , tiempo de ejecución = 7 milisecs

📌 Medicion: MedicionBuscarTrieH - Consumo de memoria=9.206.636 Bytes , tiempo de ejecución = 13 milisecs

Segunda medicion:

📌 Medicion: MedicionBuscarTrie - Consumo de memoria=10450571 Bytes , tiempo de ejecución = 9 milisecs

📌 Medicion: MedicionBuscarTrieH - Consumo de memoria=9206636 Bytes , tiempo de ejecución = 22 milisecs

Tercera medicion:

📌 Medicion: MedicionBuscarTrie - Consumo de memoria=10450571 Bytes , tiempo de ejecución = 9 milisecs

📌 Medicion: MedicionBuscarTrieH - Consumo de memoria=9206636 Bytes , tiempo de ejecución = 15 milisecs

Conclusión:

Como se puede ver a traves de las tres mediciones realizadas, el trie con una implementacion por array es un poco mas rapida, pero mucho mas costosa en cuanto a memoria que la del hash, para hacerse una idea, con 26 letras en el abecedario el consumo es superior por 1 millon de bytes. Lo cual si nuestro abecedario tuviese mas caracteres podria suponer un problema.

Por ejemplo podemos ver que, suponiendo que nuestro abecedario consiste en todos los caracteres pertenecientes al idioma japones (un total de 2210 caracters), los resultados son los siguientes:

```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

Por los cual nos muestra que nuestro modelo para el trie con array es sumamente ineficiente, e inaplicable si extendemos nuestro abecedario a cantidades .

Otro ejemplo es porbando con 1000 caracteres, el cual da los siguientes resultados:

📌 Medicion: MedicionBuscarTrie - Consumo de memoria=252801251 Bytes , tiempo de ejecución = 13 milisecs

📌 Medicion: MedicionBuscarTrieH - Consumo de memoria=9206636 Bytes , tiempo de ejecución = 16 milisecs

Como podemos observar, el tamaño del abecedario es totalmente independiente al rendimiento del programa utilizando un HashMap, en cambio el Trie con un array se vuelve sumamente ineficiente.