**Hito 2 - Trabajo Final CC 68**

En este documento se explicarán las estructuras de datos a implementar en el trabajo final y se dará una breve explicación del porqué de su uso y porqué no otras opciones.

1. **Bombas - Colas** : Se implementan las colas por su implementación que sigue el FIFO( First Input First Output ) , debido a que la primera bomba en entrar(se muestra en pantalla) será la primera en salir (cuando explota y se elimina).
   1. ¿ Porque no **arreglos** ? **:** Tienen una importante limitación , no pueden cambiar su tamaño en tiempo de ejecución. Su tamaño está definido en el diseño del programa , se pueden crear arreglo dinámicos pero se debe hacer una reconstrucción total del arreglo que no es eficiente.
   2. ¿Por qué no **pilas**?: Porque el uso de pilas conlleva a que la primera bomba que entra sería la última que sale, por ello no tendría relación con la implementación de bombas con pilas.
   3. ¿ Porque no **listas enlazadas** ? : No es necesario insertar las bombas en una posición determinada además no necesita conexión con su elemento siguiente o anterior , son independientes. Ahí pierde sentido implementar listas.
   4. ¿ Porque no **STL-vector** ? : No es necesario insertar las bombas en una posición determinada , además no cuenta con una función ‘desencolar’ , con la que sí cuenta la cola.

**Demostrando cuantitativamente :**

Las pilas son ideales para cuando la implementación sigue ciertas normas de acceso , en este caso FIFO , mientras que los arreglos son para acceso aleatorio o determinado.

1. **Mapas - Arreglos :** Para los mapas se utilizan matrices o arreglos bidimensionales , que sirven para imprimir el mapa dado un valor determinado.
   1. ¿ Porque no **listas enlazada**? : El mapa no se redimensiona en tiempo de ejecución , por lo tanto el dinamismo de las listas pierde valor.
   2. ¿ Porque no **STL-vector** ? : El mapa no se redimensiona en tiempo de ejecución , por lo tanto el dinamismo de vector pierde valor.

**Demostrando cuantitativamente :**

El mapa no se redimensiona en tiempo de ejecución y se necesita un acceso directo a cada índice de la matriz en tiempo constante algo que no ofrece las listas enlazadas.Es por este motivo que no se utilizan listas enlazadas.



1. **Ranking - Árboles** : Se puede implementar el uso de árboles en un ranking de puntajes en la implementación de una búsqueda para encontrar tu nickname en comparación con los otros dentro de un archivo.
   1. ¿ Porque no **listas enlazada o STL-vector** ? : La ventaja de los árboles es que el costo promedio de búsqueda en un ABB es *O(log(n))* , mientras que en las otras opciones las búsquedas son lineales y menos eficientes.

**Demostrando cuantitativamente:**

**Supuestos:**

* Arbol con *n* nodos.
* Probabilidad de acceso a los elementos uniforme.

Costo de búsqueda exitosa:



donde *In* es el largo de caminos internos.

Costo de búsqueda infructuosa:



donde *En* es el largo de caminos externos.

Recordando que *En=In+2n*, se tiene:



Por lo tanto, la ecuación que relaciona los costos de búsqueda exitosa e infructuosa es: (\*)

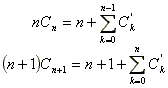


Esto muestra que a medida que se insertan más elementos en el ABB los costos de búsqueda exitosa e infructuosa se van haciendo cada vez más parecidos.

El costo de búsqueda de un elemento en un ABB es igual al costo de búsqueda infructuosa justo antes de insertarlo más 1. Esto quiere decir que si ya habían *k* elementos en el árbol y se inserta uno más, el costo esperado de búsqueda para este último es *1+C'k*. Por lo tanto:



Esta última ecuación implica que:



Restando ambas ecuaciones se obtiene: (\*\*)



De la ecuación (\*) se tiene:



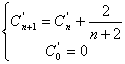
Reemplazando en (\*\*):







Obteniéndose la siguiente ecuación de recurrencia:



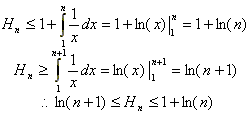
Desenrollando la ecuación: (\*\*\*)



Se define *Hn*, conocido como *números armónicos*, como:



Se puede demostrar que:



Reemplazando en (\*\*\*) y recordando (\*) se obtiene:



Por lo tanto queda demostrado que el tiempo promedio de búsqueda en un árbol es de *O(log(n)).*