Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y sistemas
Estructuras de Datos
Ingenieros:

- Ing. Luis Espino
- Ing. Edgar Ornelis
- Ing. Álvaro Hernández

Auxiliares:

- Alex Lopez
- Walter Mach
- Wilfred Perez

Proyecto 1 Fase 3

USAC Games, Batalla naval

Índice

Objetivos

Objetivo general

- Aplicar los conocimientos del curso Estructuras de Datos en el desarrollo de una aplicación que permita manipular la información de forma óptima.

Objetivos específicos

- Demostrar los conocimientos adquiridos sobre estructuras de datos no lineales: tablas hash y grafos, poniéndolos en práctica en el desarrollo del juego batalla naval.
- Utilizar el lenguaje C++ para implementar estructuras de datos no lineales.
- Utilizar el lenguaje de programación Python para el desarrollo de interfaces gráficas.
- Utilizar la herramienta Graphviz para graficar estructuras de datos no lineales.
- Definir e implementar algoritmos de búsqueda, recorrido y eliminación.

Descripción General del Proyecto

La empresa Usac Games desea implementar un videojuego que permitan desarrollar la agilidad mental de los usuarios, por lo cual ha planeado desarrollar una aplicación con el juego Batalla naval y le solicita a usted como estudiante de estructura de datos poder implementar algoritmos, funciones y estructuras que permitan que el juego tenga un rendimiento óptimo y fluido.

Debido al alcance y complejidad de las funcionalidades del videojuego se ha decidido dividirlo en 3 fases. A continuación se describen de forma general las funcionalidades a cubrir en cada una de las fases.

Fase 1 (20 pts - C++)

- **Listas**: Registro de usuarios
    - **Encriptación**: Aplicar seguridad a la información de los usuarios
- **Lista de listas**: Tienda de Skins del juego (los puntos por partida serán la moneda)
- **Pila**: Retroceder jugadas (Push y pop de movimientos)
- **Cola**: Tutorial del juego (push y pop de información con movimientos)

Fase 2 (30 pts - C++ y Python)

- **Matrices**: Tablero del juego y eventos para realizar movimientos
- **Árbol B**: Indexación de usuarios por orden alfabético
- **Árbol AVL**: Compras del usuario ordenadas por precio.

Fase 3 (40 pts - C++ y Python)

- **Tabla Hash:** Bitácora de eventos del sistema
    - Registro de usuarios, skins, compras de los usuarios y jugadas realizadas
- **Grafos**: Lista adyacente y grafo de movimientos por partida del usuario
- **BlockChain**: Transacciones al realizar compras

Consideraciones

1. Las funcionalidades están descritas de forma general, las mismas pueden sufrir algunas modificaciones en el enunciado final de cada fase.

2. La funcionalidad correcta del videojuego será cubierto por las 3 fases, por lo que de no realizar una de las fases, supondrá un mayor esfuerzo por parte del estudiante para completar la fase siguiente, el desarrollo de las estructuras se realizará en C++ y la funcionalidad visual se realizará en Python.

Fase 3

En esta fase del proyecto, Usac Games desea agregar nuevas funcionalidades al juego, se debe agregar una bitácora de transacciones, registro de movimientos por cada partida y hacer uso de la tecnología blockchain para guardar el registro de las transacciones realizadas en la aplicación, relacionadas con la compra de skins. Además debe implementarse la funcionalidad de ejecución de partida entre dos jugadores reales.

Modalidad de juego: Jugador vs Jugador

Se deben mostrar los dos tableros en juego (principal y secundario). En el mapa principal se mostrarán los barcos ubicados inicialmente, así como los disparos recibidos por el adversario. En el mapa secundario se desplegarán los disparos fallidos y acertados hacia el oponente. En ambos casos (mapa principal y secundario) deben de mostrar el número correspondiente a cada fila y columna. Queda a discreción del estudiante la simbología a utilizar para un disparo y el límite del tablero.

Lógica del disparo

Al inicio del juego se debe de escoger de forma aleatoria el turno de cada jugador, se debe de pedir las coordenadas (x, y) de cada disparo, y repintar el mapa por jugador. hasta que sea el fin del juego. Si en caso el jugador es la computadora cuando este realice el disparo, las coordenadas deben de ser aleatorias dentro del rango del tablero, para la máquina no se mostrará sus tableros(principal y secundario).

- Cada disparo solo puede abarcar una casilla o una coordenada.
- Un barco quedará destruido cuando acierten los disparos en cada una de las posiciones de él dentro del tablero.
- Si el disparo acierta deberá notificar al jugador en curso
- El juego terminará cuando alguno de los dos jugadores se quede sin barcos.
- Si las coordenadas no existen dentro del tablero en juego se debe de notificar al jugador en curso.

Scores

La aplicación deberá poder llevar el registro de puntajes de todos los jugadores registrados en el sistema.

Durante la ejecución de una partida

1. Mostrar el estado actual del tablero
    a. Se debe mostrar el tablero principal y secundario indicando el jugador en curso.
2. Solicitar el movimiento del jugador en turno
    a. En caso de que el movimiento sea inválido se deberá notificar al usuario y se volverá a pedir que ingrese de nuevo su movimiento.
    b. Si el disparo acierta a un barco deberá notificarlo
3. Repetir los pasos anteriores hasta que no existan barcos enemigos o barcos propios.

Al finalizar la partida

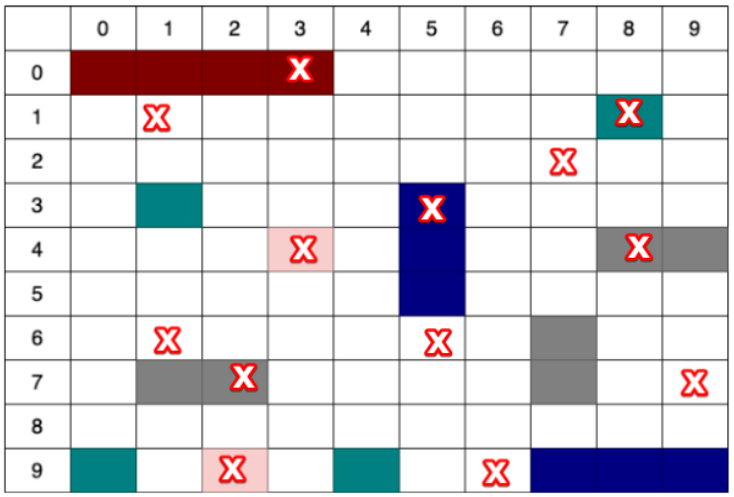
1. Mostrar los resultados de ambos jugadores, es decir el número de barcos destruidos por jugador y que barcos fueron.
2. Aumentar el registro de partidas ganadas del jugador que ganó la partida
    a. Los jugadores se identificarán por su nombre de usuario.
3. Preguntar si se desea comenzar una partida con la configuración ya establecida o si se desea regresar al menú principal.

Resumen gráfico del jugador ganador

Al finalizar la partida se utilizarán los movimientos que el usuario ganador haya realizado para generar una lista de adyacencia, dicha lista posteriormente se utilizará para generar el grafo de movimientos realizados.

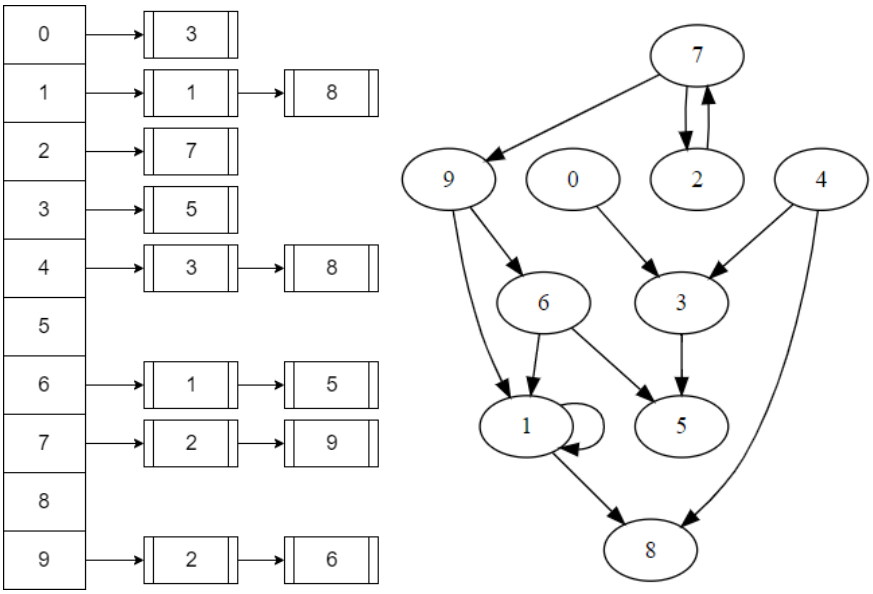
Por ejemplo:

- En el tablero se muestran “x” indicando los movimientos que realizó el jugador ganador:



Nota: Este tablero no debe ser mostrado nuevamente

- De dicho tablero se obtiene la siguiente lista de adyacencia y su respectivo grafo.



En la pantalla de visualización de resultados debe existir las opciones correspondientes para visualizar cualquiera de las dos imágenes, para estas opciones es posible utilizar ventanas modales o emergentes. La estructura lista de adyacencia solo existirá al momento

de finalizar la partida y mostrar los resultados, cuando se inicie una nueva partida o se salga de la pantalla de visualización de resultados esta estructura se eliminará.

Registro de transacciones de compras

Debido a que es necesario tener persistencia en los datos que se almacenan en la aplicacion, para que estos sean confiables, no solo para los usuarios sino también para futuras revisiones de la empresa, por lo tanto, todo el registro de transacciones de compra debe almacenarse usando blockchain, ya que esta tecnologia provee un registro inviolable debido a las caracteristicas de implementacion de la estructura de datos que la conforma.

Árbol de Merkle

El árbol de Merkle es una estructura de datos que sirve para realizar el guardado de transacciones que se realizan dentro de las aplicaciones, para lo cual se necesita seguir ciertos pasos.

Este árbol permite registrar todas las operaciones que son realizadas dentro de la plataforma, con esto se refiere a que debe de crearse un árbol para almacenar las operaciones de compras.

Generar una operación en el arbol de Merkle

Para poder insertar en el árbol, se debe realizar la compra de una skin.

Cuando el proceso sea generado de manera exitosa se debe generar un nodo con todos los datos indicados en la sección **“Operación de compra”** para generar un hash que sea único.

Ejemplo

id = funcionHash(Data)
Donde:

- **id** es con el que se debe guardar en el árbol de merkle
- **funcionHash:** método utilizado para encriptar la información, la función que se utilizara para la generación del hash será sha256.
- **Data:** Corresponde a toda la información que generará el hash, debe incluir todos los datos de la operación realizada.

Ejemplo de nodos hojas

id = funcionHash(“3425123542135 , 13/11/2022 , ...”)

id = 39b923082a194166d8d92989116bd

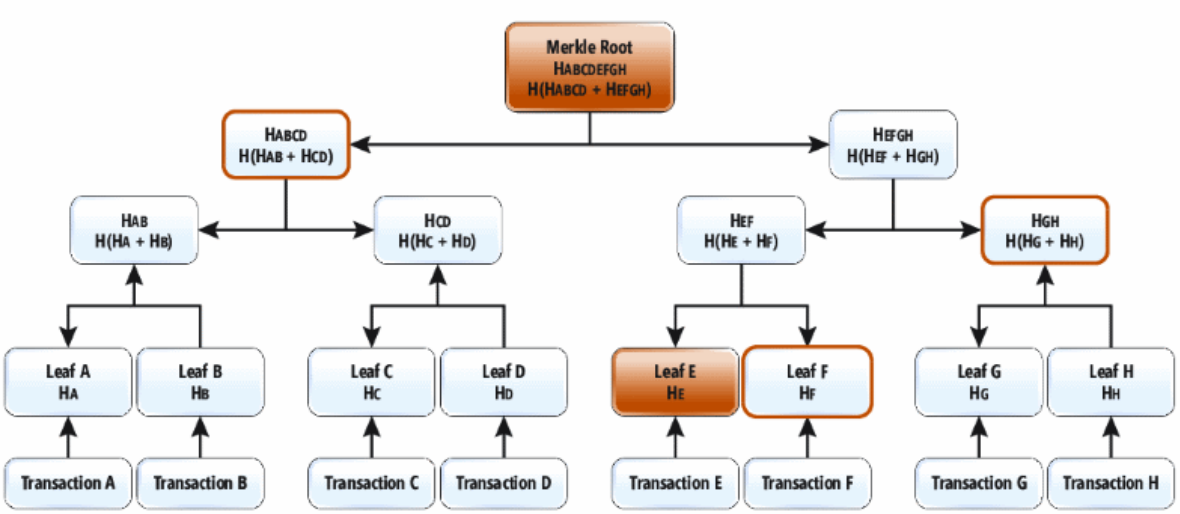
Se crea un nodo interno cuando una transacción está esperando al número mínimo para poder crear un hash (2 valores hash como mínimo para poder generar un nuevo hash).

Ejemplo de nodos internos (se toman los hash de los nodos hoja para generar el hash del nodo interno)

id=funcionHash("39b923082a194166d8d92989116bd, aa17d5fdf24761b54ad640691146656095b")

id = 113b685b12dbfa76c04bec7759a24ba66dd

Ejemplo de salida de un árbol Merkle



Notas:

- Las cadenas de caracteres utilizadas en la ejemplificación del proceso de construcción del árbol no son cadenas válidas correspondientes a algún método de cifrado existente, por lo que el árbol mostrado es solo una representación de la estructura de datos.
- El árbol de merkle se caracteriza por ser un árbol que siempre se encuentra completamente lleno, así que cuando el número de transacciones no complete el árbol, este debe de llenarse con valores representativos como nulos, por ejemplo **-1**.

Para la comprobación de las transacciones válidas, se debe mostrar de forma gráfica el árbol para poder visualizar que todos los hash correspondan, para ello **se deben leer los bloques de blockchain** (descritos en la siguiente sección) y luego validar los hashes de merkle y los hashes de la cadena de bloques, si por algún caso se modifica alguno de los procesos o asignaciones debe mostrarse de diferente color donde se rompe la relación.

Blockchain

La aplicación va a funcionar utilizando los conceptos de blockchain. Esta estructura almacena las operaciones de compra dentro de la aplicación.

Almacenamiento de información en árboles Merkle:

Inicialmente se estableció que se debe guardar la información de todas las compras realizadas dentro de la plataforma en los árboles merkle para validar la consistencia de la información, por lo cual esto se mantendrá en memoria mientras transcurre el tiempo de la creación del nuevo bloque, en el momento que se cumple el tiempo el bloque se escribirá, los arboles de merkle deben reiniciarse, esto para poder guardar la nueva informacion que se incluirá en el nuevo bloque.

Si no existieron operaciones realizadas desde la creación del último bloque, no se debe agregar ningún bloque a la cadena.

Estructuras a utilizar en el Blockchain

- **Blockchain:** Esta es una estructura de bloques, los cuales se comportan como una lista enlazada simple, en donde cada uno de los nodos (bloques) se ingresan por delante.

Prueba de Trabajo

Es el proceso por el cual se encuentra un hash que cumpla con la condición de

tener un prefijo de n ceros, donde n corresponde a un número entero, este puede ser cambiado por el administrador y por defecto tendrá el valor de 4. Para ello se debe de iterar un entero denominado **NONCE** hasta encontrar un hash válido para el bloque.

Operaciones de Blockchain

Nuevo Bloque: Pasado el tiempo de configuración en la aplicación se genera un nuevo bloque que almacena las sumarización del árbol de merkle anteriormente descrito. El tiempo en que un nuevo bloque se genera está determinado por un valor m, el cual es un número entero representando la cantidad de minutos, este puede modificarse y por defecto tendrá el valor de 3 (minutos).

Bloque

El bloque se define de la siguiente manera:

- **INDEX:** representa el número de bloque el bloque génesis tendrá valor de index 0 , los bloques posteriores deberán tener valores 1, 2, 3, 4 ... etc.
- **TIMESTAMP (fecha y hora de creación):** Es la fecha y hora exacta en la que se creó el bloque**.** Debe de tener el siguiente formato: (DD-MM-YY::HH:MM:SS).
- **DATA:** Contendrá cada una de compras realizadas por los clientes los que serán almacenados en este apartado.
- **NONCE:** Será el número entero que se debe iterar de uno en uno hasta encontrar un hash que cumpla con la prueba de trabado, es decir que contenga como prefijo un numero de 0s configurado por el adminsitrador.
- **PREVIOUSHASH:** Es el hash del bloque previo, este es necesario para validar que la cadena de bloques no esté corrupta. En caso del bloque génesis, el hash anterior debe de ser 0000.
- **ROOTMERKLE**: En este bloque se almacena el nodo padre del árbol de Merkle.Este árbol de Merkle se forma con los datos del campo DATA, que son las operaciones de entrega
- **HASH (bloque actual):** El hash que protege que la data no se ha comprometido, el hash deberá generarse aplicando la función SHA256 a las

propiedades: INDEX, TIMESTAMP, PREVIOUSHASH, ROOTMERKLE y NONCE **todas estas propiedades como cadenas concatenadas sin espacios en blanco ni saltos de línea**. Es decir SHA256(INDEX+TIMESTAMP+PREVIOUSHASH+ROOTMERKLE+NONCE). Para considerar el hash como válido este debe de tener un prefijo de cuatro ceros. Es decir que un hash valido sería el siguiente: 000082b12041cb5a7bac8ec90f86b654af6b1ac8bfc5ed08092e217235df0229

Definición de bloques

Cada uno de los bloques tendrá la siguiente estructura

```
{
  "INDEX": 0,
  "TIMESTAMP": "05-06-22::10:34:45",
  "NONCE": 2345,
  "DATA":{

  },
  "PREVIOUSHASH":"0000axwsde...",
  "ROOTMERKLE": "39b923082a194166d8d92989116bd",
  "HASH":"000082b12041cb5a7bac8ec90f86b654af6b1ac8bfc5ed08092e217235df0229"
}
```

En la sección DATA se colocarán los datos de las operaciones de compra que se ejecutaron en el nodo que creó el bloque.

Operación Compra

Carrito de compras

El carrito de compras servirá para almacenar las skins que el usuario quiera comprar, pero aun no se pagará.

Función de añadir al carrito:

Cada vez que se vaya añadiendo un producto al carrito debe de aumentarse el contador de cuantos productos hay en el carrito y el precio total de las skin añadidas.

Cuando se le de clic al carrito de de compras debe hacer los siguiente:

- Debe de mostrar una pantalla con una lista de las skin añadidas
- Debe tener la opción de poder eliminar una skin del carrito
    - Cuando se elimine deberá disminuir el contador en 1 y el precio total.
- Debe tener un botón que tendrá la funcionalidad de cancelar la compra
    - Cuando se cancele, se debe vaciar el carrito.
- Debe tener un botón que tendrá la funcionalidad de pagar lo que está en carrito, deberá mostrar la pantalla de pago.

TODOS LOS PRODUCTOS DEL CARRITO DEBEN DE ALMACENARSE EN LA TABLA HASH.

Tabla hash

Los productos se almacenarán en una tabla hash utilizando como llave el id del usuario concatenado con el nombre del producto a comprar.

Características de la tabla hash:

- El tamaño inicial de la tabla hash será 13 (M=13)
- La función de dispersión a utilizar es: $h(llv) = llv \bmod M$
- Donde lleva el id del artículo.
- La tabla es única por usuario.
- El porcentaje máximo de ocupación será el 80%
- Al aumentar el tamaño de la tabla el porcentaje de ocupación quedará 80% max para la tabla vieja y 20% min para la nueva tabla
- La política para la resolución de colisiones será la doble dispersión por medio de la función: $s(llv, i) = (llv \bmod 3 + 1) * i$ donde i es el número de colisiones que han ocurrido al insertar un nuevo valor.

Ejemplo del uso de la función hash y la resolución de colisiones

inicialmente al utilizar el módulo siempre da un valor dentro del rango de índices de la tabla

h(3)=3 % 13 = 3

Al ocurrir alguna colisión se aplica la segunda función

En la aplicación de la segunda función se retorna el número de espacios que se deben sumar al cálculo del primer índice, si este resultado excede se considera al vector de la tabla como circular y se continúa con la suma de posiciones a partir del inicio de la tabla.

por ejemplo para un valor con clave 16 el proceso sería el siguiente

h(16)=(16 % 3 +1)* 1 = 2 el nuevo índice es 3+2 = 5

para la segunda colisión, tomando como ejemplo un valor con clave 29

h(29)=(29 % 3 +1)* 2 = 6 el nuevo índice es 3+6 = 11

Visualización del carrito (Graphviz)

Para este árbol se debe mostrar una representación en graphviz. A la ventana del carrito, se debe de agregar un nuevo botón que tenga la funcionalidad de ver la imagen generada por graphviz.

Funcionamiento:

- Al presionar el botón “ver en graphviz” se debe de generar automáticamente la imagen y debe de ser abierta automáticamente por el programa para poder visualizarla.

La imagen generada debe mostrar el índice de la tabla, el id de la skin y el nombre de la skin.

índice	**id**	**nombre**
0	22	skin 1
5	5	legendaria 2
9	31	textura madera
10	43	skin a

Pago de skins

Wallet

Una wallet es un monedero digital en el cuál almacenamos nuestras criptos. Esto facilita el poder disponer de ellas en cualquier plataforma dedicada a la tecnología Blockchain, en el caso del juego almacenará la cantidad de tokens ganados en el total de partidas jugadas.

Si el usuario no ha realizado ninguna compra el sistema debe de ser capaz de mostrar la opción de crear una wallet usando web3 en python (Se brindará un ejemplo en el laboratorio),

El sistema únicamente aceptará pagos a través de los tokens ganados.

proceso de compra:

a. Al estar logueado, los datos del usuario se cargarán automáticamente.
b. Si el usuario ya tiene una cuenta, la aplicación debe mostrar la dirección de la cuenta en la pantalla de pago.
c. Al confirmar la compra se deberá solicitar la clave privada de la cuenta del usuario a través de un campo de texto.
d. Al momento de finalizar la compra se debe limpiar el carrito de compras y se debe agregar la transacción al árbol merkle, para posteriormente formar parte de algún bloque en la cadena.

La operación representará una compra, la definicion sera la siguiente:

```
"DATA": {
  "FROM": "0x4281ecf07378ee595c564a59048801330f3084ee",
  "SKINS": [
    {
      "SKIN": 123,
      "VALUE": 456
    },
    {
      "SKIN": 21,
      "VALUE": 789
    },
  ]
}
```

Guardar bloque

Luego de terminar el proceso de crear un nuevo bloque, se procede a almacenar el archivo en la carpeta de bloques.

Al momento de iniciar el programa se deben leer los bloques que ya existan.

Nombre de la carpeta de bloques: Bloques

Ruta de la carpeta de bloques (relativa a la carpeta principal del proyecto): \blockchain\

Nombres de los bloques: INDEX_TIMESTAMP.json donde INDEX Y TIMESTAMP son atributos definidos en el bloque. (dado que en los nombres de archivos no son permitidos algunos caracteres como “dos puntos”, debe sustituir esos caracteres por guión)

Arranque de la aplicación:

Al iniciar la aplicación se procede a leer cada uno de los bloques generados, luego se procede a cargar las estructuras de datos.

Cierre de la aplicación:

La aplicación debe tener la capacidad de que antes de cerrarse, pueda guardar la última instancia de procesos o transacciones.

Notas:

1. Solo el usuario administrador debe tener acceso a la interfaz de configuración de blockchain.

Administración

Configuración de bloques:

El tiempo por defecto de la creación de bloques será de 3 minutos.

- El administrador puede cambiar el lapso de tiempo entre la creación de cada bloque, el tiempo siempre va a estar en minutos. Este tiempo se inicia luego de iniciar la aplicación y luego de que se carguen los bloques anteriores.
- También podrá crear un bloque inmediatamente sin necesidad de esperar el tiempo establecido.
- El administrador también puede cambiar el número de ceros iniciales en el hash del bloque, prueba de trabajo.

Observaciones

- Lenguaje de Programación:
    - C++ para el desarrollo de las estructuras
    - Python para la interfaz gráfica
- Sistema Operativo: Elección del estudiante.
- El IDE a utilizar queda a discreción del estudiante.
- Librería para graficar las estructuras: Graphviz
- Los archivos de entrada serán documentos en formato JSON (.json)
- El estudiante debe tener un repositorio privado en github con el nombre [EDD_2S]BatallaNaval_#carnet, se crearán 3 carpetas fase1, fase 2 y fase 3.
- Agregar a su tutor como colaborador al repositorio del proyecto.
    - Auxiliar 1: wltomv
    - Auxiliar 2: aexlopez@gmail.com
    - Auxiliar 3: willop
- Se entregará en UEDI un .zip con los entregables solicitados.
- Las copias tendrán nota de 0 puntos y serán reportadas al catedrático y a la escuela de sistemas.

Entregables

- Manual Técnico
- Compilado de C++
- Código fuente en un .zip
- Link a repositorio con el código fuente.

Restricciones

- Las estructuras deben de ser desarrolladas por los estudiantes sin el uso de ninguna librería o estructura predefinida en el lenguaje a utilizar.
- No se permite la modificación de código durante la calificación, únicamente se calificará sobre el ejecutable entregado en UEDI.

Fecha de Entrega

- 27 de octubre a las 23:59 horas.
- **No se aceptarán entregas tarde**