PARCIAL TEORICO SEGUNDO TERCIO

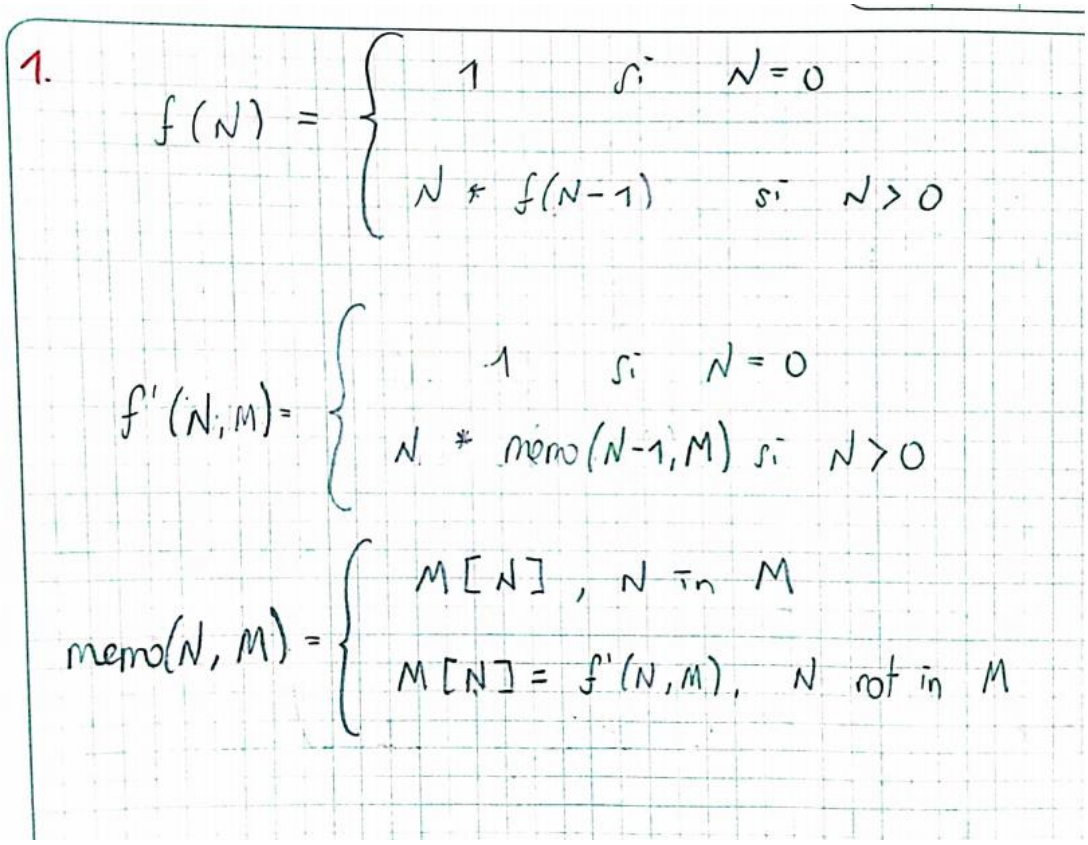
PRESENTADO POR:
JUAN CAMILO BAZURTO ARIAS

PRESENTADO A:
SEBASTIAN CAMILO MARTINEZ REYES

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO
ALGORITMOS Y ESTRUCTURAS DE DATOS
PROGRAMA DE INGENIERÍA DE SISTEMAS
BOGOTÁ D.C.
2021 – 1

1. (10 puntos) Dada la siguiente función recurrente, determine las funciones resultantes utilizando la técnica de memorización.

$$f(N) = \begin{cases} 1 & si \quad N = 0 \\ N * f(N-1) & si \quad N > 0 \end{cases}$$



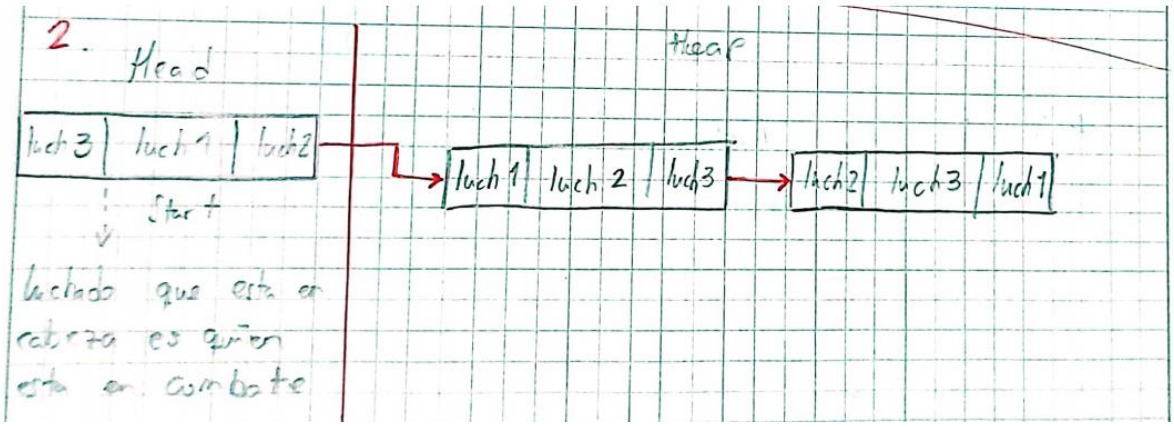
2. (20 puntos) Ha sido contratado por la compañia **Arc System Games** para el desarrollo de un nuevo producto de software, en este caso un videojuego de consumo masivo **Dragon Sphere FighterF**. El género de esté es de Lucha por equipos **3 Vs 3** en donde cada jugador controla un equipo de 3 peleadores, su objetivo es diseñar una estructura de datos que satisfaga lo siguiente:

   - Sólo dos peleadores (uno por equipo) pueden estar en la arena combatiendo.

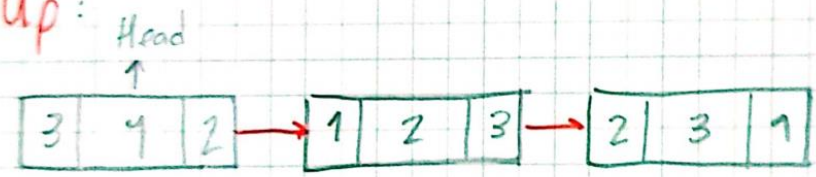
   - Dada la entrada por comando del control de la plataforma de su preferencia, se han definido dos comandos por jugador **ChgUp** y **ChgDown**; En donde **ChgUp** en medio del combate hace una llamada al siguiente jugador del **Rooster** de peleadores del jugador y **ChgDown** llama al anterior.

   - Para un numero indefinido de entradas de **ChgUp** y **ChgDown**, los peleadores llamados a combatir deben estar dentro de los seleccionados por los jugadores.

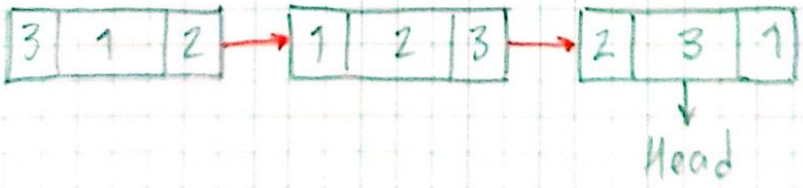
   Sustente su respuesta para el control de un jugador:



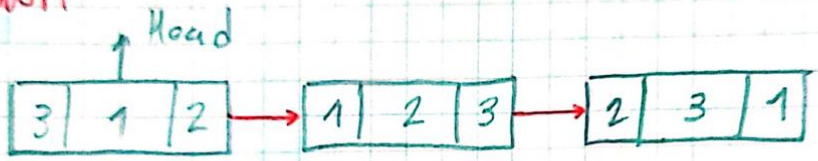
ChgUp:



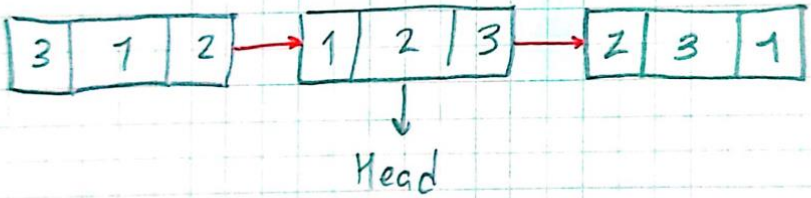
setHead(getPrev())



ChgDown:



setHead(getNext())



3. (20 puntos) Otra de sus tareas cómo desarrollador es la de crear el sistema de multijugador en línea para el título **Dragon Sphere FighterF**, para este modo se ha definido que un jugador podrá crear una sala en línea, en donde jugadores de todo el mundo podrán tener combates con el anfitrión. El sistema es el siguiente:

- El anfitrión abre la sala virtual.

- Los jugadores de todo el mundo ven la sala disponible.

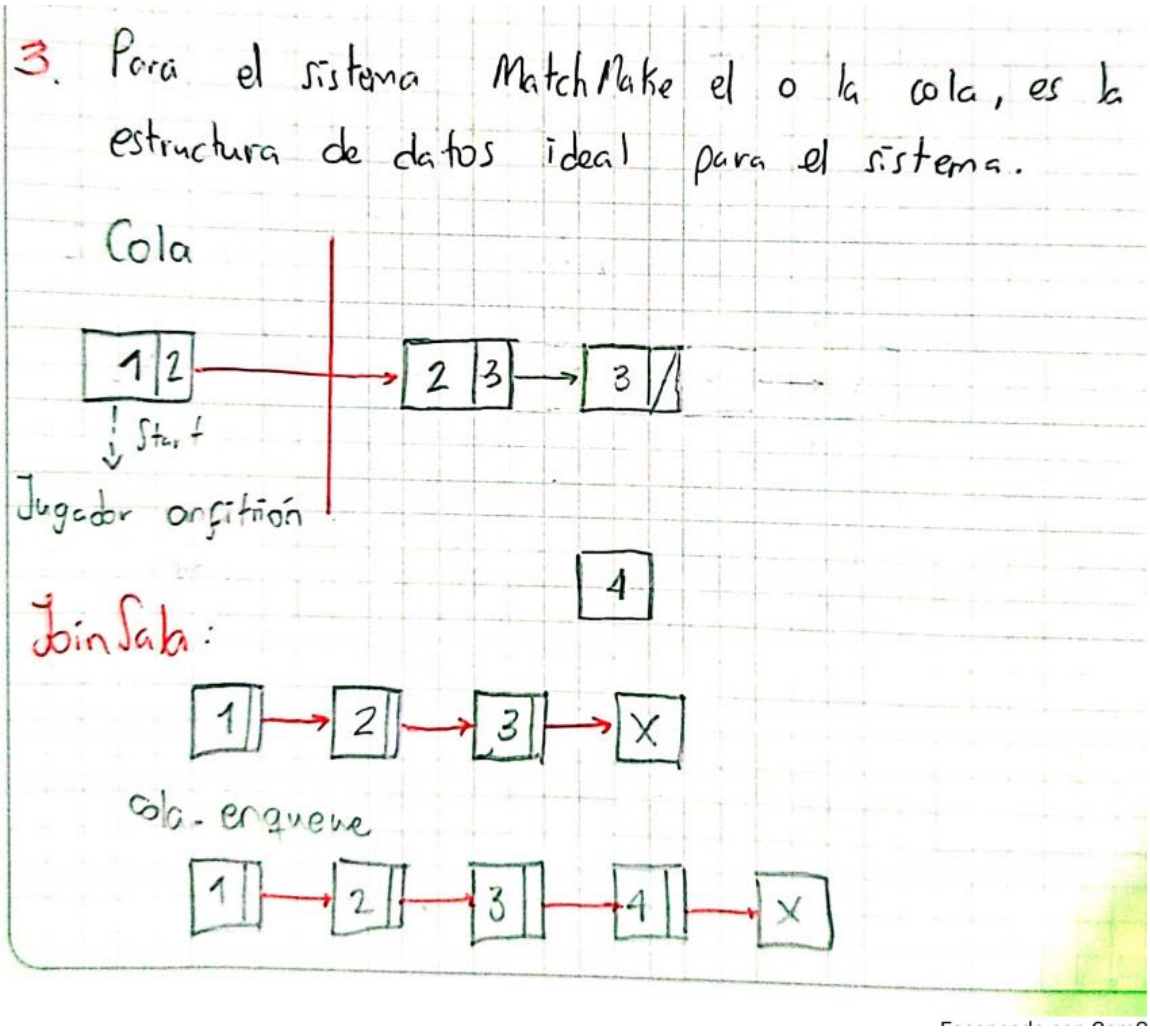
- Los jugadores se unen a la sala.

- El sistema de **MatchMake** coordina los combates en orden de llegada **del primero al último**.

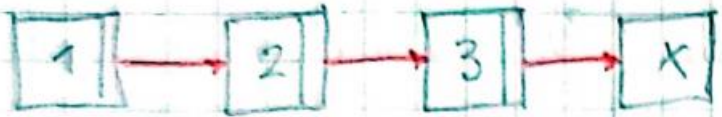
- Si el Anfitrión pierde la partida, el jugador vencedor se vuelve anfitrión de la sala y el actual anfitrión queda ultimo en la lista de jugadores.

¿ Cuál estructura de datos usaría para el sistema de **MatchMake**?

Sustente su respuesta con las simulaciones para los escenarios de **Los jugadores se unen a la sala** y **El Anfitrión pierde la partida**.

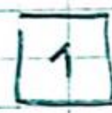


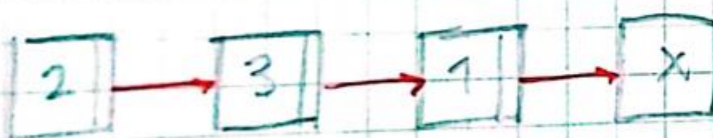
AnsitFail:



cola.dequeue

Cola.enqueue





4. (20 puntos) Debido a su maravilloso trabajo en el equipo de desarrollo se les ha convocado para crear una característica **feature** del sistema que ayudara a que el titulo **Dragon Sphere FighterF** sea mas atractivo a jugadores profesionales, el diseño de esta nueva característica es el siguiente:

   - Cada vez que un combate en una sala en línea ocurre y un jugador gana el combate, el **Roster** de jugadores seleccionado es almacenado en una tripla $(fechacombate, roster, afinidad)$ en donde $fechacombate$ es la fecha exacta del encuentro, $roster$ corresponde a los nombres de los personajes seleccionados en ese combate y $afinidad$ es un valor calculado de acuerdo al tiempo restante en el contador de la batalla y la vida de cada uno de los combatientes (**3vs3**). Para este escenario se garantiza que $afinidad$ es un valor único por cada $roster$, ejemplo de estos registros:

   Jugador: AlienInstinct

   - (23/10/2018 19:27, 'Kakarott, VegitaSSj4, Gon', 25)

   - (22/10/2018 15:07, 'Tronks, Snake, Kiritu', 32)

   - (22/10/2018 13:06, 'Ralf, Carnage, yemcha', 66)

   - (20/10/2018 12:02, 'Kakarott, VegitaSSj4, Gran saiyamen', 55)

   - (22/10/2018 15:07, 'Brook , Beeros, yemcha', 12)

   - Los jugadores competitivos quieren ver estos registros en un árbol en donde **los equipos con las mejores afinidades aparezcan a la derecha y los peores a la izquierda**, de tal manera que visualmente tengan un mecanismo para componer mejor sus **Roster** y realizar operaciones de manera **óptima**.

   ¿ Cuál estructura de datos usaría para el nuevo **feature**?

   Sustente su respuesta con las simulaciones para los escenarios de **registros históricos** y **Un nuevo combate en línea ocurre**.

   ¿ Qué pasa cuando el jugador **gana**?

   ¿ Qué pasa cuando el jugador **pierde**?

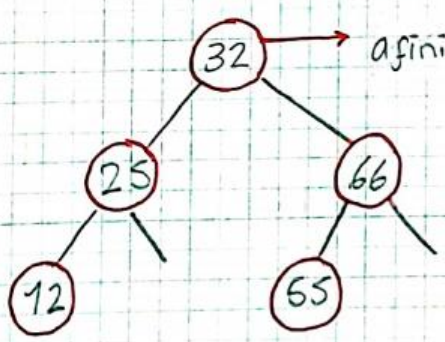
   ¿ Qué pasa cuando la partida **no es válida**?

4. La estructura de datos optima para el nuevo feature es un arbol binario AVL. Esto porque, se tendra de manera ordenada todas las partidas realizadas por el jugador.

Registros Historicos:

Jugador: AlienInstict

Los datos se ordenan segun su afinidad.

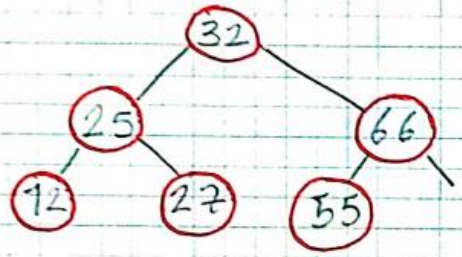


El dato que almacena cada nodo es una tripla tal que (fechacombate, roster, afinidad), por ejemplo en el nodo con afinidad 32 se almacena la tripla: (22/10/2018 15:07, "Tronks, Snake, Kirito", 32)

Nuevo Combate en linea ocurre:

El nuevo combate genera una tripla tal que (06/04/2021 13:45, "Kakarott, Gon, Brook", 27)

El arbol AVL del jugador AlienInstict:



Jugador gana:

Cuando un jugador gana la partida entonces la afinidad de la partida tendra un puntaje alto por lo que se asignara al subarbol derecho.

Jugador pierde:

Cuando un jugador pierde entonces el puntaje de afinidad sera bajo por lo tanto se asignara al subarbol izquierdo.

Partida no valida:

Cuando la partida no es valida significa que su afinidad es nula por lo que no se asignara en el arbol.

5. (10 puntos) [Bono] Para la funciones resultantes usando la técnica de memorización del **punto 1**, ¿Cuál sería la estructura de datos que acompañaría a las implementaciones **Top-down** ?, ¿ Cual sería el estado de la memoría para N=10?

Sustente su respuesta explicando con una simulación el llenado de la memoria

5. la estructura de datos optima para la implementación Top-down será una lista indexada, donde en la posición n de la lista este el valor $f(n)$.

- n = 0 , memolist[1] , memolist[n] = 1
- n = 1 , memolist[1,1], memolist[n] = 1
- n = 2 , memolist[1,1,2], memolist[n] = 2
- n = 3 , memolist[1,1,2,6], memolist[n] = 6
- n = 4 , memolist[1,1,2,6,24], memolist[n] = 24
- n = 5 , memolist[1,1,2,6,24,120]
- n = 6 , memolist[1,1,2,6,24,120,720]
- n = 7 , memolist[1,1,2,6,24,120,720,5040]

memolist[n] = 5040

* n = 8 , memolist[n] = 40320

memolist = [1, 1, 2, 6, 24, 120, 720, 5040, 40320]

* n = 9 , memolist[n] = 362880

memolist = [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]

* n = 10, memolist[n] = 3628800

memolist = [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, 3628800]