

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2133 - Estructura de datos y algoritmos

Pauta Taller 1

Contexto

En este taller el objetivo fue evaluar el uso del lenguaje C mediante un problema de listas ligadas. Dentro del contexto del taller, existen 2 entidades principales:

- magnemite: con atributos `ID`, `ATK` y `SPEED`. Esta entidad representa un nodo de la lista ligada
- magneZone: con atributo `ID`. Esta entidad representa la lista ligada en si, almacenando una referencia a la cabeza de la lista

Los estudiantes al iniciar el taller contaban en el codigo base que les fue entregado con las siguientes funciones y archivos

magnemite.c

- `magnemite_create(int id, int atk, int speed)`: Crea una instancia de un struct magnemite y lo retorna.
- `magnemite_destroy(Magnemite* target)`: Utiliza free para eliminar una instancia de un magnemite.

magnezone.c

- `magnezone_create(int id)`: Crea una instancia de un magnezone (una lista ligada) vacia y la retorna.
- `magnezone_destroy(MagneZone* target)`: Elimina un magnezone, eliminando primero todos sus nodos.
- `magnezone_length(MagneZone* magnezone)`: Retorna el largo de la lista ligada
- `magnezone_print(MagneZone* magnezone)`: Print human-friendly de la lista ligada
- `magnezone_insert(MagneZone* magnezone, int magnemite_id, int magnemite_atk, int magnemite_speed)`: Crea e inserta un nuevo nodo magnemite al final de la lista.
- `magnezone_remove(MagneZone* magnezone, int magnemite_id)`: Elimina un nodo magnemite de la lista ligada y lo **elimina**. El hecho de que lo elimine será importante para los eventos del taller evaluado.
- `magnezone_count_by_atk(MagneZone* magnezone, int target_atk)`: Retorna la cantidad de nodos magnemite con `atk` igual a `target_atk`
- `magnezone_find_by_atk(MagneZone* magnezone, int target_atk)`: Retorna un arreglo con todos los nodos magnemites con `atk` igual a `target_atk`

main.c y events.c

Para el taller formativo, se tenian que implementar eventos `ENTER`, `REMOVE`, `FIND`, `INFO` todos operaciones basicas en listas ligadas. En el archivo `main.c` se les entrega el flujo inicial del programa (recibir los primeros inputs, crear un arreglo de listas ligadas vacias y el bucle que recibe eventos) mientras que en el archivo `events.c` hay una función para cada uno de los eventos anteriores, que se encarga de recibir el input, usar las funciones auxiliares para resolver el evento y hacer los prints de output.

Evaluación

Teniendo como base lo anterior, los estudiantes tenian que programar 2 funciones (distintas para cada grupo) donde se esperaba que usaran las cosas que se les entregó en el codigo base.

Cada función contará con un puntaje de 3 puntos, tener el puntaje completo el ambas funciones corresponde a 6 puntos que con el punto de base corresponde a un 7.

Además, a cada grupo se le presentó una función bonus, que en caso de tener una idea correcta otorga hasta 0.3 puntos extra, independientes al puntaje de las funciones principales.

Nota importante para los correctores: El objetivo de esta pauta es evaluar que el estudiante demuestre conocimientos de C, más allá de que la implementación realizada este correcta. Lo que significa que buscamos evaluar que la idea de la implementación este presente sin importar que hayan bugs.

Evaluación Grupo 1

Los estudiantes del grupo tenian que programar las funciones `INVERSE` y `SHIFT`

Funcion 1: INVERSE {magnezone_id}

Recibe el id de un magnezone e inverte toda su lista ligada, de forma que el ultimo pasa a ser el primero y el primero el ultimo, y asi sucesivamente para cada nodo.

En base a esto, se consideran dos soluciones con sus puntajes:

Solución 1: Usar listas doblemente ligadas

Usar listas doblemente ligadas te permite que el inverse se reduzca a solo intercambiar los punteros `prev` y `next` de cada nodo. Para implementar esto se consideran los siguientes pasos:

1. Agregar el atributo prev a los magnemites [1 pt]

```
struct magnemite {
    int id;
    int atk;
    int speed;
    struct magnemite* next;
    struct magnemite* prev;    <----- agregar este puntero
};
```

2. Alterar funciones previas [1 pt]

Es necesario cambiar las funciones que requieren el manejo de nodos en la lista ligada para que ahora actualicen bien el puntero prev. Estas son: `magnezone_insert` y `magnezone_remove`

magnezone_insert: Al insertar un nuevo magnemite, su previo es la anterior cola de la lista.

```
new_magnemite->prev = tail;
```

magnezone_remove: Al eliminar un nodo, el previo del siguiente al que estamos eliminando, debe ser el previo del que estamos eliminando.

```
current_magnemite->next->prev = previous_magnemite;
```

3. Bucle principal [1 pt]

Ahora hacemos lo antes mencionado, para cada nodo, intercambiamos puntero prev y next.

```
Magnemite * current = magnezone->head;
while(current != NULL){
    Magnemite * next = current->next;

    //intercambiamos los punteros
    Magnemite * temp = current->prev;
    current->prev = current->next;
    current->next = temp;

    if (next == NULL){
        magnezone->head = current;
    }

    current = next;
}
```

Solución 2. Crear una copia

Tambien se considera valido el crear una nueva lista ligada y copiar cada uno de los nodos de la original pero en orden inverso.

1. Creamos una copia del magnezone [0.5 pts]

```
MagneZone * copy = magnezone_create(magnezone->id);
```

2. Buscamos el ultimo [0.5 pts]

Como la funcion insert que les entregamos, inserta al final de la lista, lo correcto es ir copiando desde el ultimo nodo hacia detras.

```
Magnemite * last = magnezone->head;
while (last->next != NULL){
    last = last->next;
}
```

Nota: tambien era correcto copiar desde la cabeza en adelante pero habia que definir un insert que agregue al inicio de la lista y no al final.

3. Copiar el nodo [1 pt]

Como dijimos antes, la funcion remove elimina al nodo que esta removiendo, asi que lo mejor es crear una nueva copia de ese nodo

```
//insertamos el ultimo creando una copia de este
magnezone_insert(copy, last->id, last->atk, last->speed);
//borramos el nodo anterior
magnemite_destroy(last);
```

Condición de termino [1 pt]

Lo anterior debemos hacerlo en un bucle hasta haber copiado todos los nodos, terminamos cuando hayamos copiado la cabeza de la lista

```
while(true){
    if(last == magnezone->head){

        /*...logica de copia ...*/
        break;
    }

}
```

Nota final

Estas dos soluciones se consideran correctas, pero otra persona pudo haber hecho otra solución, como por ejemplo, intentar dar vuelta a la lista solo recorriendola sin usar puntero prev (es posible). Por lo que en esos casos, evaluar con criterio.

Función 2: SHIFT {magnezone_id} {k}

La función shift consiste en hacer shift right a los nodos de la lista en k posiciones. Noten que hacer esto es solo mover los ultimos k nodos al inicio.

Para esto, consideramos dos soluciones:

1. La primera solución es buscar el ultimo nodo de la lista y moverlo a la cabeza, y repetir esto k veces. En base a eso se considera el siguiente puntaje

- Buscar el ultimo nodo [0.5 pts]
- Sacar el ultimo nodo de la lista [0.5 pts]
- Moverlo al inicio y marcarlo como nueva cabeza [1 pt]
- Repetir k veces [1 pt]

2. La otra solución es hacer un juego con los punteros y mover todos los nodos en un solo movimiento. Para eso hacemos lo siguiente:

1. Calcular el k real [0.5 pts]

k podia ser mayor al largo de la lista, por lo que la cantidad efectiva de nodos a mover se conseguía con

```
int len = magnezone_length(magnezone);
k = k % len;
```

2. Obtener el ultimo nodo de la lista [0.5 pts]

Al mover los ultimos k nodos, el ultimo nodo de la lista ante de mover, será el que termine apuntando a la cabeza actual de la lista.

```
Magnemite * last = magnezone->head;
while (last->next != NULL){
    last = last->next;
}
```

3. Encontrar la nueva cola [1 pt]

La nueva cola despues de mover los ultimos k nodos se encuentra en la posición `len - k - 1` ya que es el nodo numero k + 1 desde el final al inicio.

```
Magnemite * new_tail = magnezone->head;
for (int i = 0; i < len - k - 1; i++){
    new_tail = new_tail->next;
}
```

4. Mover los punteros [1 pt]

Una vez que tenemos la cola actual y la nueva cola despues de mover los k nodos, hacemos el siguiente juego de punteros.

```
tail->next = magnezone->head;
magnezone->head = new_tail->next;
new_tail->next = NULL;
```

Función bonus: TWOSUM {magnezone_id}

La función pide encontrar los dos nodos que maximicen su suma por speed en la lista. En otras palabras, encontrar los dos nodos con mayor velocidad.

La solución más directa es recorrer dos veces la lista y encontrar los 2 más grandes, cuyo puntaje se distribuye de la siguiente forma:

- Encontrar el nodo con mayor `SPEED` [0.1 pt]
- Buscar el segundo nodo con mayor `SPEED` [0.1 pt]
- Verificar que el segundo nodo encontrado sea distinto al nodo encontrado anteriormente [0.1 pts]

Evaluación Grupo 2

Los estudiantes del grupo 2 debian hacer la funciones `SWAP` y `SPEEDIEST`

Función 1: SWAP {magnezone_id}

Este evento pide intercambiar los nodos de la lista ligada pero de a pares. Es decir, intercambiar de posición el 1 con el 2, el 3 con el 4, el 5 con el 6 y asi sucesivamente.

Para esto consideramos 2 soluciones:

1. Intercambiar los valores de los nodos, sin intercambiar su posición realmente. Ante esta solución, se considera el siguiente puntaje

- Almacenar los datos del nodo segundo en el nodo primero. Esto inclute id, atk, speed y puntero next [1 pt]
- Almacenar los datos del nodo primero en el nodo segundo, habiéndolos almacenado previamente en variables temporales. [1 pt]
- Una vez intercambiado un par, mover la referencia de los nodos primero y segundo al siguiente par de nodos [1 pt]

2. La otra solución es hacer un algoritmo que actualice punteros, intercambiando efectivamente los nodos.

1. Definir punteros previo, primero y segundo [1 pt]

El algoritmo trabajan siempre con 3 nodos, previo, primero y segundo. Noten que esto se puede definir siempre salvo al inicio de la lista (ya que no hay un nodo previo a la cabeza). Por lo que ese caso se podia manejar creando un nodo dummy (nodo vacio que solo sirva para el algoritmo y que sea previo a la head) o manejando como un caso especial y partir el algoritmo desde el siguiente par.

2. Actualización de punteros [1 pt]

La secuencia correcta para intercambiar los nodos, dado que ya tenemos las referencias a los nodos previo, primero y segundo es la siguiente:

```
primero->next = segundo->next;
segundo->next = primero;
previo->next = segundo;
```

3. Mover los punteros previo, primero y segundo [1 pt]

Despues de intercambiar un par, movemos los punteros para intercambiar el siguiente par

```
previo = primero;
primero = previo->next;
segundo = primero->next;
```

Función 2. SPEEDIEST {magnezone_id}

Esta función pide encontrar el magnemite más rapido de la lista y moverlo al inicio. La distribución de puntaje es la siguiente:

1. Almacenar estadisticas del maximo [0.5 pts]

Ya que la función remove elimina el nodo, lo mejor es copiar sus estadisticas y crear una copiar de este

```
Magnemite * current = magnezone->head;

int higest_speed = current->speed;
int higest_id = current->id;
int higest_atk = current->atk;
```

2. Buscamos el maximo y actualizamos [0.5 pts]

```
while (current != NULL){

    if (current->speed > higest_speed){
        higest_speed = current->speed;
        higest_id = current->id;
        higest_atk = current->atk;
    }
    current = current->next;
}
```

3. Eliminamos el nodo anterior y creamos una copia [1 pt]

```
magnezone_remove(magnezone, higest_id);

Magnemite* copy_higest = magnemite_create(higest_id, higest_atk, higest_speed);
```

4. Lo colocamos al inicio [1 pt]

```
copy_higest->next = magnezone->head;
magnezone->head = copy_higest;
```

Función Bonus: UNIQUE {magnezone_id}

Esta función busca eliminar las copias de todos los nodos de la lista ligada que tengan la misma speed, conservando unicamente el nodo que aparezca primero en la lista del conjunto de duplicados.

La solución mas simple es hacer un doble bucle que para cada nodo, revise todos los nodos siguiente y elimine los que tengan igual speed. Para esta solución, se considera la siguiente distribución de puntaje:

- Bucle principal que almacena la speed del nodo actual que estamos revisando [0.1 pt]
- Bucle secundario que parte en el siguiente del nodo actual [0.1 pt]
- En caso de encontrar un nodo en el bucle secundario que tenga la misma speed que el actual, usar la funcion remove para eliminarlo [0.1 pt]