Universidad de Magallanes

Programa caché LRU.
(Estructura de datos)

Integrantes: Dario Duarte,
Christopher Gutierrez,
Franco Cartes,
Alejandro Carrasco

Fecha: 24/10/2025

Asignatura: Estructura de datos

Punta Arenas 2025

Índice:

1. Introducción:

1.1 Algoritmo LRU:

El algoritmo LRU (Least recently used), es una estrategia de gestión de caché, la cual consta que cuando la caché está llena, elimina el dato al que no se ha accedido en el tiempo más largo.

1.2 Cache:

La caché es una memoria intermedia entre la RAM y CPU la cual guarda los datos o instrucciones más utilizados, de esta forma la CPU no buscará las instrucciones en la RAM (lo cual se traduce a lentitud).

En la presente tarea se desarrollará un programa, escrito en lenguaje C, el cual imitara el funcionamiento del algoritmo LRU.

2. Objetivos

2.1 Objetivo General:

- Crear un programa el cual pueda simular el comportamiento del algoritmo LRU , y al mismo tiempo comprender la funcionalidad del mismo

2.2 Objetivos específicos:

- Implementar las listas enlazadas para la creación del programa
- No ocupar más de 50 líneas en funciones
- Evitar el uso desmedido de variables globales
- Implementar todas las condiciones necesarias para la captura de fallas en la ejecución del programa

3. Funcionamiento del programa

3.1 Estructuras

La primera estructura creada lleva por nombre **Node_** , la cual representa un nodo de una lista doblemente enlazada, donde cada elemento contiene un dato, y 2 punteros:
***data:** dato ingresado
***prev**: apuntando al nodo anterior
***next:** apuntando al nodo siguiente.

La segunda estructura creada con nombre **Cache_ ,** la cual modela el funcionamiento de un caché tipo LRU. Contiene elementos tales como:
****data**: Almacena un arreglo de datos
size: Número actual de elementos almacenados
capacity: Capacidad máxima de elementos que se pueden almacenar
***head:** puntero al nodo más recientemente utilizado
***tail:** puntero al nodo menos utilizado

3.2 Main.c

Proceso principal del programa lru, el cual capta la interacción usuario/programa, el cual consta principalmente de un switch con 7 casos.

`case help`: Al usuario se le entrega una serie de comandos para ejecutar correctamente el programa. En caso de no ingresar ningún comando válido se le notifica al usuario como acceder a help.

`Case create`: Con el comando `./lru create <tamaño caché>` se crea una carpeta que contiene la información y los datos del caché en archivos de texto.
cachedata.txt: guarda la capacidad máxima y el tamaño actual del caché.
data.txt: guarda la data ingresada por medio del comando add.

A la estructura `cache` se le asigna el valor que entrega la función `lru _create`. Esta función, pasa como parámetro la capacidad ingresada por el usuario; capacidad, que mediante la función `CharToNum` pasa de ser un string a un dato numérico.

Case add: Ingresa un nuevo dato a la caché, al cual previamente se le asigna memoria y los datos correspondientes a `cache` mediante las funciones `lru_load_cache` y `load_data`. Luego, se agrega el dato mediante la función `lru_add`, la cual se encargará de asignarle al dato reciente la mayor prioridad. Además, actualiza los archivos texto de acuerdo a la situación.

Case search: Se le asigna memoria y los datos correspondientes a `cache` mediante las funciones `lru_load_cache` y `load_data`. Luego, se ingresa a la función `lru_search` que busca el dato ingresado dentro de `cache`.

Case all: Se le asigna memoria y los datos correspondientes a `cache` mediante las funciones `lru_load_cache` y `load_data`. Ingresa a la función `lru_all`, que imprime los datos dentro de `cache` según su orden de prioridad.

Case get: Se le asigna memoria y los datos correspondientes a `cache` mediante las funciones `lru_load_cache` y `load_data`. Entra a la función `lru_get`, en donde cambia la prioridad del dato ingresado dentro del caché.

Case exit: Entra en la función `lru_exit`.

3.3 commands.h

Sección del programa donde se almacenan funciones que involucran los siguientes comandos de ejecución del programa create, add, all, search, get y exit.

`lru_create`: Función que se encarga de crear el cache, reservando memoria dinámica a la estructura `Cache_`, mediante una función **`create_folder()`** crea una carpeta y crea un archivo llamado data.txt.

Posibles errores:
1-) Asignar un número de capacidad menor a 5 para el cache
2-) Falta de memoria para el caché o los datos del caché (malloc)
3-) Error al crear la carpeta cache (función **create folder** devuelve un valor distinto de 0)

`lru_load_cache`: Función que se encarga de crear un nuevo caché, reservando memoria para la estructura `Cache_`, se reserva la memoria con **`malloc`,** valores de la estructura cache (“data”, “size” y “capacity”) se inicializan en valores nulos y se devuelve el puntero al `Cache_`

Posibles errores:

1-) Falta de memoria para crear el cache.

2-) Error al leer el archivo `cachedata` en la función **`load_cachedata`**

`lru_add`: Una de las funciones esenciales del programa, ya que se encarga de agregar un nuevo dato a la caché utilizando lista doblemente enlazada para asignarle la prioridad correspondiente a los datos. Con `search_data` se verifica que no se esté ingresando dos veces la misma data al caché.

En caso de que el tamaño del cache supere la capacidad ingresada por el usuario, se podrá apreciar como el dato con menor prioridad se elimina para crear espacio al nuevo dato

Posibles errores:

1-) Falta de memoria para el nodo (malloc)

2-) No haber escrito create al iniciar el programa

`lru_all`: Con ayuda de una estructura `Node_` se recorre la lista enlazada imprimiendo la data contenida en el orden de prioridad separándose con guiones (-).

`lru_search`: Gracias a la función `search_data`, verifica la existencia del dato ingresado.

`lru_get`: Usando nuevamente `search_data`, se verifica primero que el dato ingresado efectivamente exista. Si existe, con la función `swap` se cambia el orden dentro de `cache`. Finalmente, con la función `rewrite_data` se actualiza el archivo data.txt para que esté de acuerdo al orden de `cache`.

`lru_exit`: Verifica que la carpeta “cache” exista y borra los archivos texto junto a carpeta del caché, funcionando para sistemas Windows y Linux.

3.4 functions.h

Contiene todas las funciones que se usan dentro de `commands.h`.

`create_folder`: Crea el directorio cache haciendo uso de la función `system("mkdir cache")`

`load_cachedata`: Lee el archivo cachedata.txt e ingresa la capacidad y el tamaño actual del caché a `cache`.

`update_cache`: Actualiza según los valores actuales de `cache` el archivo cachedata.txt.

`search_data`: Recorre con ayuda de una estructura `Node_` hasta que encuentra dicho valor buscado o hasta que la lista se acabe.

`rewrite_data`: Reescribe el archivo data.txt ingresando los datos previamente ordenados en un arreglo, con el fin de que al volver a cargar el caché esten ordenados según su prioridad

`load_data`: Ingresa los valores guardados en data.txt.

`update_data`: Ingresa el último dato ingresado con add, al archivo data.txt.

`swap`: Hace el movimiento de los nodos cuando se esté realizando un get, dejando al dato ingresado en la cabecera.

`DirExists`: Verifica la existencia de carpetas/directorios.

4. Procedimiento

Para iniciar, lo primero que se discutió fue cómo iba funcionar el programa, que funcionara como una consola interactiva a través de un while o que funcionara a través de ingreso argumentos y archivos texto. Se optó por la última alternativa. Inicialmente se trabajó con solo un archivo texto, pero debido a que guardaba la capacidad, el tamaño del caché y además los datos, se decidió trabajar con dos archivos textos(cachedata.txt y data.txt).

Se hicieron tres headers, que contuviera las funciones principales, como los comandos (commands.h); otra en la que estén las funciones que usan los comandos (function.h) y la última que tuviera las estructuras usadas (structures.h). Se trabajó inicialmente con un main con el nombre de “main.c” en el que se utilizaban estos headers, pero se le cambió el nombre a “lru.c”, con el fin de que al ejecutar el programa en la terminal fuera similar al ejemplo mostrado en la tarea.

Se tuvo precaución al momento de reescribir data.txt, ya que, de la forma en la que funciona el programa invertía el orden de los datos, provocando incoherencia.

5. conclusión

Se tiene un programa funcional, en el sentido de que cumple con lo que solicitado, el usuario puede ingresar una capacidad máxima de datos que quiera guardar, se elimina el menos utilizado, se puede cambiar prioridades de datos, etc... Por lo tanto se puede decir que el **objetivo general** se ha cumplido.

En cuanto al código, se cuenta con uno ordenado, con secciones de comandos, funciones, main, lo cual puede facilitar la comprensión de quien lo pueda ver.

Objetivos específicos cumplidos:

- Se ocuparon listas enlazadas (doblemente enlazadas) para determinar el nivel de prioridad de los datos guardados en caché.
- Hay un buen manejo de errores dentro del programa.
- Hubo un uso controlado de variables globales

Objetivo específico parcialmente cumplido:

- Si bien la gran mayoría de funciones cumplen con el requisito de no ocupar más de 50 líneas de código, y que no se ocupen más de 4 líneas por nivel de indentación . Hay algunos casos como **lru_add** que supera esa cantidad, y **rewrite_dat**a que casi lo hace.

6. Ejemplo:

```
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru create 4
La capacidad minima es 5
Creando cache de capacidad 5
Cache creado con exito
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru add A
Dato 'A' agregado al cache. Tamano actual: 1
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru add B
Dato 'B' agregado al cache. Tamano actual: 2
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru add C
Dato 'C' agregado al cache. Tamano actual: 3
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru add D
Dato 'D' agregado al cache. Tamano actual: 4
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru add E
Dato 'E' agregado al cache. Tamano actual: 5
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru all
E - D - C - B - A
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru add F
Cache lleno. Reescribiendo data.txt...
Eliminando dato menos reciente: 'A'
Cache lleno. Dato 'F' agregado al cache, el dato menos reciente fue eliminado.
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru get B
Dato 'B' movido a la posicion mas reciente del cache.
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru all
B - F - E - D - C
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru exit
Carpeta cache eliminada con exito
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru> ./lru all
Error al cargar el cache. Asegurese de crear uno primero.
PS C:\Users\ACER\Documents\EeD\TRABAJO_2_EdD_lru>
```