Minh họa danh sách liên kết đôi quản lý cache cơ chế LRU (Least Recently Used Cache), MRU (Most Recently Used Cache), MFU (Most Frequently Used)

LRU (Least Recently Used Cache)
MRU (Most Recently Used Cache)
MFU (Most Frequently Used)

1. **Giới thiệu về LRU Cache**

- **LRU (Least Recently Used)** là một thuật toán thay thế trang dùng trong bộ nhớ cache.
- Khi bộ nhớ cache **đầy**, thuật toán sẽ loại bỏ phần tử **ít được sử dụng gần đây nhất**.
- **Cấu trúc dữ liệu phù hợp: Danh sách liên kết đôi + Hash Map** giúp:
    - **Truy xuất nhanh (O(1))** → dùng **Hash Map**.
    - **Xóa/thêm nhanh (O(1))** → dùng **Danh sách liên kết đôi (DLL)**.

Cấu trúc dữ liệu

Danh sách liên kết đôi (Doubly Linked List - DLL) để **quản lý thứ tự truy cập** (MRU → LRU).
Hash Map (unordered_map) để tra cứu phần tử nhanh (O(1)).

Mỗi **Node** của DLL chứa:

- key, value: Dữ liệu cache.
- prev, next: Con trỏ để quản lý thứ tự truy cập.

Quản lý LRU:

- **Mỗi khi truy cập một phần tử**, ta **đưa nó lên đầu danh sách (Most Recently Used - MRU)**.
- **Nếu cache đầy**, ta **xóa phần tử cuối danh sách (Least Recently Used - LRU)**.

```
typedef struct Node {
    int key;
    char value[MAX_LEN];
    struct Node* prev, * next;
} Node;

typedef struct LRUCache {
    int capacity, size;
    std::unordered_map<int, Node*> cacheMap;
    Node* head, * tail;
} LRUCache;
```

Chúng ta dùng **Hash Map** để tìm và trả về khóa nhanh hơn.

```
// Hàm khởi tạo LRU Cache
LRUCache* initCache(int capacity) {
    LRUCache* cache = (LRUCache*)malloc(sizeof(LRUCache));
    cache->capacity = capacity;
```

```
    cache->size = 0;
    cache->head = cache->tail = NULL;
    return cache;
}
```

Khi tìm kiếm 1 khóa và khóa đó tồn tại, phần tử tương ứng sẽ được di chuyển về đầu.

Hàm **getCache(int key)**

Khi thêm 1 khóa mới, khóa đó cũng sẽ được di chuyển về đầu. Nếu danh sách hiện tại vượt quá kích thước cache ta sẽ loại bỏ phần tử cuối cùng trong cache trước khi thêm phần tử mới.

Hàm **putCache(int key, int value)**

```
// In cache (Debug)
void printCache(LRUCache* cache) {
    Node* cur = cache->head;
    printf("Cache: ");
    while (cur) {
        printf("(%d,%s) ", cur->key, cur->value);
        cur = cur->next;
    }
    printf("\n");
}
```

Hàm in các phần tử và khóa trong cache để test code và debug.

```
CACHE_SIZE  = 3
LRUCache* cache = initCache(CACHE_SIZE);

put(cache, 1, “Value 1”);
put(cache, 2, “Value 2”);
put(cache, 3, “Value 3”);
printCache(cache);

get(cache, 2); // Truy cập 2 -> Đưa 2 lên đầu
printCache(cache);

put(cache, 4, “Value 4”);
printCache(cache);

get(cache, 1);
```

2. Minh họa danh sách liên kết đôi quản lý MRU (Most Recently Used Cache)

MRU và LRU đều là chiến lược thay thế trang/bộ nhớ trong cache nhưng có cách hoạt động ngược nhau:

Tiêu chí	LRU (Least Recently Used)	MRU (Most Recently Used)
Nguyên tắc	Loại bỏ phần tử ít được sử dụng gần đây nhất.	Loại bỏ phần tử vừa được sử dụng gần nhất.
Lựa chọn loại bỏ	Phần tử ít được truy cập nhất trong quá khứ sẽ bị loại.	Phần tử vừa mới truy cập xong sẽ bị loại.
Phù hợp với trường hợp	Dữ liệu cũ ít được sử dụng lại.	Dữ liệu cũ có khả năng được sử dụng lại ngay.
Cấu trúc dữ liệu	Danh sách liên kết đôi + Bảng băm để truy cập nhanh O(1).	Danh sách liên kết đôi + Bảng băm để truy cập nhanh O(1).
Hiệu quả trong	Truy xuất dữ liệu cũ nhưng vẫn có khả năng cần dùng lại.	Khi dữ liệu mới nhất ít khi cần truy xuất lại ngay lập tức.
Ứng dụng thực tế	Bộ nhớ cache CPU, cache trang web, hệ thống quản lý bộ nhớ.	Một số thuật toán cơ sở dữ liệu hoặc buffer trong hệ thống tập tin.

🔹 **MRU Cache**

Cache size là 3

Truy cập A → B → C → Cache: [C, B, A]
Truy cập D → **Loại bỏ C (vừa mới được dùng nhất)** → Cache: [D, B, A]
Truy cập B → Cache: [B, D, A]
Truy cập A → Cache: [A, B, D]
Truy cập E → Loại bỏ A **(vừa mới được dùng nhất)** → Cache: [E, B, D]

Cài đặt minh họa cache với cơ chế MRU tương tự LRU ở trên

3. **Cache MFU (Most Frequently Used)**

MFU (Most Frequently Used) là thuật toán thay thế trang/bộ nhớ ưu tiên loại bỏ phần tử được truy cập nhiều nhất thay vì ít được truy cập như LRU (Least Recently Used) hay MRU (Most Recently Used).

Cơ chế hoạt động

✅ **MFU giả định rằng dữ liệu được truy cập nhiều nhất đã "lỗi thời" và sẽ ít được dùng trong tương lai.**
✅ **Khi cache đầy, MFU sẽ loại bỏ phần tử có số lần truy cập cao nhất**.

📝 **Các bước hoạt động:**

1. Mỗi phần tử trong cache có một bộ đếm **tần suất truy cập**.

2. Khi cần thay thế, MFU sẽ chọn phần tử có **số lần truy cập lớn nhất** và loại bỏ nó.
3. Nếu có nhiều phần tử có cùng số lần truy cập, có thể áp dụng **chiến lược bổ sung** như FIFO (loại phần tử cũ nhất) hoặc Random (chọn ngẫu nhiên). T**rong bài minh là chọn chiến thuật FIFO.**

Cache size là 3

Truy cập A → B → C → Cache: [C, B, A]
Truy cập D → **Loại bỏ A (cùng tần số, FIFO)** → Cache: [D,C,B]
Truy cập B → Cache: [D,C,B] tần số truy cập B là 2
Truy cập A → loại bỏ B → Cache: [A, D, C]
Truy cập E → Loại bỏ C **(cùng tần số, FIFO)** → Cache: [E, A, D]

Cài đặt minh họa cache với cơ chế MFU tương tự LRU ở trên