연결리스트로 LRU 시뮬레이터 구현하기

20211741 이지연

1. LRU 시뮬레이터

lruSim.py

LRU_20211741 > lruSim.py > do_sim

```python
from circularDoublyLinkedList import CircularDoublyLinkedList

def do_sim(cache_slots):

    cache_hit = 0
    tot_cnt = 0

    cache = CircularDoublyLinkedList() #양방향 연결리스트 이용

    data_file = open("LRU_20211741\linkbench_short.trc") #절대주소를 이용해 파일을 불러옴

    for line in data_file.readlines():
        lpn = line.split()[0]

        if cache.count(lpn) == 0: #<miss> - 새로 넣으려는 값이 없는 기존에 경우
            if (cache.size() == cache_slots): #cash slots이 가득찬 경우
                cache.pop(0) #가장 예전에 호출된 값을 삭제

        else: #<hit> - 값이 있는 경우
            cache.remove(lpn) #기존에 있는 값을 삭제함
            cache_hit += 1

        cache.append(lpn) #받은 값을 넣어줌

        tot_cnt += 1



    print("cache_slot = ", cache_slots, "cache_hit = ", cache_hit, "hit ratio = ", cache_hit / tot_cnt)

if __name__ == "__main__":

    for cache_slots in range(100, 1000, 100):
        do_sim(cache_slots)
```

2. 실행결과 출력

```
C:\Users\USER\Desktop\Me\SSU\SSU2-1\Data Structure>C:/Users/USER/AppData/Local/Microsoft/WindowsApps/python3.9.exe "c:/Users/USER/Desktop/Me/SSU/SSU2-1/Data Structure/LRU_20211741/lruSim.py"
cache_slot =  100 cache_hit =  943 hit ratio =  0.0943
cache_slot =  200 cache_hit =  1031 hit ratio =  0.1031
cache_slot =  300 cache_hit =  1101 hit ratio =  0.1101
cache_slot =  400 cache_hit =  1154 hit ratio =  0.1154
cache_slot =  500 cache_hit =  1236 hit ratio =  0.1236
cache_slot =  600 cache_hit =  1387 hit ratio =  0.1387
cache_slot =  700 cache_hit =  1501 hit ratio =  0.1501
cache_slot =  800 cache_hit =  1554 hit ratio =  0.1554
cache_slot =  900 cache_hit =  1813 hit ratio =  0.1813
```

3. 시간복잡도 분석

전체적으로 볼 때, 파일을 호출하기 위해 **for** line **in** data_file.readlines()를 사용함 -> **n**

Miss, hit을 판별하기 위해 count를 사용함 -> **n**

Doubly linked list에서 remove를 사용함, 하지만 count 이후에 실행되는 과정임 -> **n**

n*(n+n) = $2n^2$

따라서, 전체적으로는 $O(n^2)$, 새로 작성한 코드를 기준으로 시간 복잡도는 $O(n)$이다.

4. 시간복잡도를 O(1)으로 개선할 수 있는 방안

새로 짠 코드를 기준으로, 파일을 읽어오는 것을 제외하고 시간복잡도를 O(1)로 개선하는 방안

딕셔너리를 이용한다.

cache = {} 로 딕셔너리를 생성하고, count 대신 if lpn in cache: 를, remove 대신 del 함수를 이용한다.

이렇게 될 경우 if in 함수는 **hash**를 이용하기 때문에 시간 복잡도가 O(1)이다.

del 함수도 시간복잡도가 O(1)이다. 따라서 딕셔너리를 사용하면 시간복잡도가 O(1)이다.