소스코드

lfuSim.py

C: > Users > gift > Desktop > ds > week6 > ds_2024-main > lfu_sim > lfuSim.py > ...

```python
from collections import defaultdict, OrderedDict

class LFUCache:
    def __init__(self, capacity):
        self.capacity = capacity                #최대 크기
        self.node = {}                          #각 키와 빈도수 저장
        self.cache = defaultdict(OrderedDict)   #heap 사용할 경우 원소 indexing을 할 방법을 도저히 모르겠습니다.
                                                #그래서 순서가 보장되고 indexing이 가능한 OrderedDict를 사용했습니다.
        self.current_size = 0                   #현재 저장된 원소의 수(캐시의 크기)

    def cacheCheck(self, key):
        if key not in self.node:                #키가 없으면 cache에 넣어주고(self.Put) 캐쉬사용을 하지 못했다는 의미로 return False
            self.Put(key, None)                 #self.Put 호출
            return False                        #return False

                                                #효율이 아주 좋지 않음 : 호출 -> 생성 -> 삭제 -> 삽입 이라는 비효율적 알고리즘...
        node = self.node[key]                   #기존 key의 빈도수를 가져오기
        del self.cache[node[0]][key]            #기존 key - minFreq 삭제
        node[0] += 1                            #빈도수 + 1 (왜냐하면 캐쉬를 사용했기 때문)
        self.cache[node[0]][key] = None         #OrderedDict에 넣어주기
        return True                             #return True

    def Put(self, key, value):
        if self.current_size >= self.capacity:  #꽉 찼다 -> 빈도수가 가장 적은 값 제거
            self.Remove()                       #self.Remove 호출

        self.node[key] = [1, value]             #빈도 수 : 1, key = value 로 생성
        self.cache[1][key] = None               #OredredDict에 넣어주기
        self.current_size += 1                  #현재 사이즈 + 1

    def Remove(self):
      minFreq, keys = next((_minFreq, _keys) for _minFreq, _keys in self.cache.items() if _keys) #가장 작은 빈도 찾기,
      #next함수를 사용하여 순회
      #for 문을 사용하여도 무방하나 파이썬스럽게 코딩하기 위해서는 위와 같은 방법을 지향(?)한다고 들었음(정확하지 않음)
      #
      #
      minKey, _ = next(iter(keys.items()))      #해당 빈도에서 가장 오래된(?) 키 찾기
      del self.cache[minFreq][minKey]           #OrderedDict에서 제거
      del self.node[minKey]                     #원소(노드) 제거
      self.current_size -= 1                    #원소 및 OrderedDict에서 제거 했으니 캐쉬 크기 감소

def lfu_sim(cache_slots):
    cache = LFUCache(cache_slots)
    cache_hit = 0
    tot_cnt = 0
    data_file = open("C:/Users/gift/Desktop/ds/ds_2024-main/lfu_sim/linkbench.trc")

    for line in data_file.readlines():
        elem = line.split()[0]
        tot_cnt += 1
        if cache.cacheCheck(elem) == True:
            cache_hit += 1

    data_file.close() #닫아주는 센스
    print("cache_slot =", cache_slots, "cache_hit =", cache_hit, "hit ratio =", cache_hit / tot_cnt)

if __name__ == "__main__":
    for cache_slots in range(100, 1100, 100):
        lfu_sim(cache_slots)
```

요구사항 각각에 대한 결과물 캡처

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

cache_slot = 100 cache_hit = 13232 hit ratio = 0.13232
cache_slot = 200 cache_hit = 15627 hit ratio = 0.15627
cache_slot = 300 cache_hit = 16357 hit ratio = 0.16357
cache_slot = 400 cache_hit = 16481 hit ratio = 0.16481
cache_slot = 500 cache_hit = 20818 hit ratio = 0.20818
cache_slot = 600 cache_hit = 20923 hit ratio = 0.20923
cache_slot = 700 cache_hit = 22314 hit ratio = 0.22314
cache_slot = 800 cache_hit = 25764 hit ratio = 0.25764
cache_slot = 900 cache_hit = 25872 hit ratio = 0.25872
cache_slot = 1000 cache_hit = 26121 hit ratio = 0.26121
```

설명

heap으로 indexing을 하는 방법을 모르겠습니다. 죄송합니다

대신에 OrderedDict를 활용하였습니다.

필요한 내용

Git