운 영 체 제

소프트웨어학부20162820 김영민

HW#2 : CLOCK ALGORITHM

보고서 내용

CLOCK ALGORITHM 구조

paging-policy.py의 클락알고리즘 의 구조 분석.

N값	8	7	4	2	5	4	7	3	4	5
M[0]	8*	8*	🟥 8*	2*	2*	2*	🟥 2	3*	3*	🟥 3
M[1]	🟥	7*	7*	🟥 5*	5*	5*	5	🟥 5	4*	4
M[2]		🟥	4*	4*	🟥 4*	🟥 4*	7*	7*	🟥 7*	5*
H/M	Miss	Miss	Miss	Miss	Miss	Hit	Miss	Miss	Miss	Miss

Clock Point : 클락이 가르키는 시침 **빨간색** 배경으로 설정

Use Bit : Use bit가 1이면 *, 0이면 아무것도 없음으로 설정

paging-policy.py Clock Algorithm을 짤 때 고려해야 할점

Hit 일때는 miss일 때 보다 간단하다. 왜냐하면 클락이 가르키는 바늘이 움직이지 않기 때문이다. 하지만 usebit는 변경 되기 때문에 고려 해주어야 한다.

<Hit일 때>

1) Use bit가 1일 때 : memory index에 있는 n값의 usebit가 1일 때 그냥 다음 N으로 next, clock바늘, usebit값 memory값 변경 없다
2) Use bit가 0일 때 : memory index에 있는 n값이 usebit가 0일 때 는 n값이 가르키는 usebit를 1로 만들고 다음 N으로 Next, 한다. 이때도 next, clock바늘, usebit값 memory값 변경 없다.

<Miss 일 때>

Miss 일때 clock바늘, usebit값, memort[index]값 전부 신경써 주어야한다.

1) use bit가 0일 때 : 클락바늘이 자기를 가르키면 usebit를 0으로 바꾸고 0이면 그값이 victim이되고 1이면 usebit가 0이되고 clock 바늘이 +=1 된다.
2) usebit가 1일 때 : Usebit가 1이라면 전에 사용했다는 것이다 그래서 세컨찬스를 잃고 use bit를 0으로 만들었다, 그리고 클락시침을 +=1 을 해주어서 클락방향을 바꿔준다

수정 개념 설명

# main program

먼저 plicy: clock으로 변경하여 clock알고리즘을 설정해주었고, clockbits를 1로 설정해서 1과 0중 usebit를 가질 수 있게 만들어 주었다. 그리고 compute default 값을 True로 설정해주어 Hit rate의 총 과정과 결과값이 출력이 되게 만들어주었다.

# init memory structure

clock_point = 0 #클락이 가르키는 시침즉 클라바늘을 선언해주었다.

change = 0 #이값은 나중에 메모리 인덱스중에 victim값이 설정되면 그값과 새로운 n값을 교체해주어 야하는데 메모리안에 있는 victim값의 index를 모르면 교체가 힘드니 그값을 저장해주기위해 설정해주었다.

# hack: for now, do random

Page에 들어갈 값을 본격적으로 변경, usebit가 1일때와 usebit가 0일 때 따로 설정해주었다

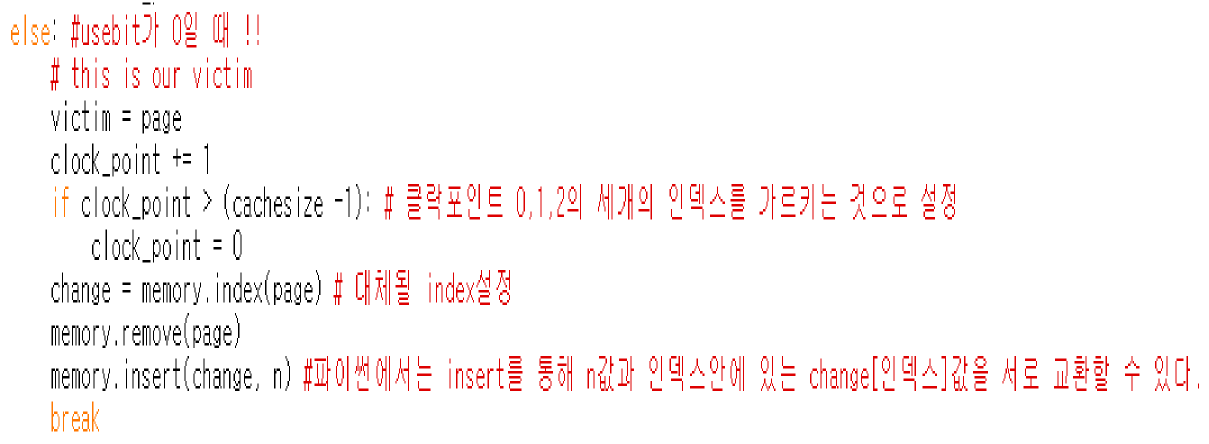
page = memory[clock_point]로 설정해주어 랜덤값이 아닌 page에 클락바늘이 가르키는 값이 page로 들어가게 된다. if ref[page] >= 1:

#usebit가 1일 때 : 앞에서 Miss이고 usebit가 1이면 usebit를 0으로 만들어주고 클락바늘을 다음 인덱스를 가르키게 설정해주었다. 여기서 중요한 것은 이것이다.

if clock_point > (cachesize -1):

clock_point = 0 #캐시사이즈가 3으로 설정되었는데 캐시사이즈이상의 값이 오면 다시 0으로 clock point를 바꿔주었다. 따라서 클락 포인트의 바늘 +1되면 2다음 0으로 바뀐다.

else: #usebit가 0일 때



```
else: #usebit가 0일 때 !!
    # this is our victim
    victim = page
    clock_point += 1
    if clock_point > (cachesize -1): # 클락포인트 0,1,2의 세개의 인덱스를 가르키는 것으로 설정
        clock_point = 0
    change = memory.index(page) # 대체될 index설정
    memory.remove(page)
    memory.insert(change, n) #파이썬에서는 insert를 통해 n값과 인덱스안에 있는 change[인덱스]값을 서로 교환할 수 있다.
    break
```

victim값을 바로 클락 방향이 가르키는 page로 설정하고 그값과 새로들어온 n값을 변경해준다, 잘몰랐는데 파이썬에서는 insert를 통해 변경이 가능하여서 memory인덱스를 찾아 victim(제거대상)과 바로 새로운 n값과 변경해주었다.

# miss, but no replacement needed (cache not full)

현제 설정된 캐시 인덱스 값이 세개 이다. 처음 N값이 들어오게 되면 컨퍼서리 미스가 난다. 이때 고려해줘야할점은 이때도 클락바늘이 증가한다는 것이다.

clock_point += 1

if clock_point > (cachesize -1):

clock_point = 0

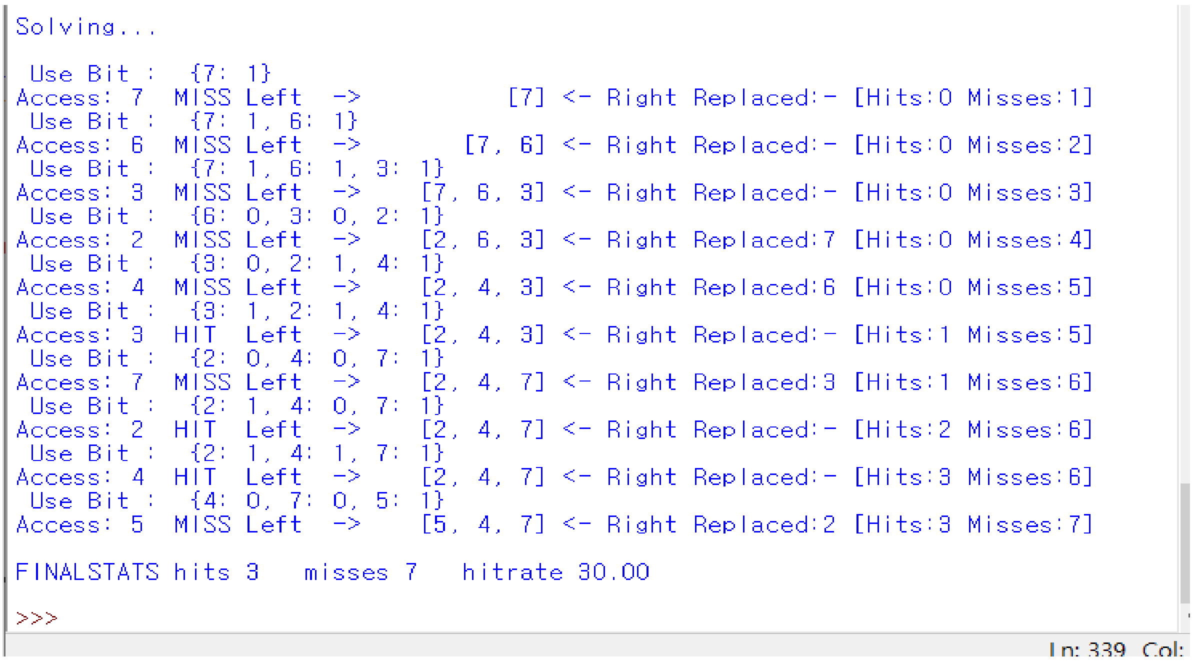
이렇게 추가해주어 컨퍼서리 미스또한 클락바늘이 변경되게 해주었다.

다양한 테스트 케이스 및 설명(Clock)

1)MaxPage : 10 / Cache Size : 3

```
ARG addresses -1
ARG addressfile
ARG numaddrs 10
ARG policy CLOCK
ARG clockbits 1
ARG cachesize 3
ARG maxpage 10
ARG seed 0
ARG notrace False

Solving...

 Use Bit :  {8: 1}
Access: 8  MISS Left  ->          [8] <- Right Replaced:- [Hits:0 Misses:1]
 Use Bit :  {8: 1, 7: 1}
Access: 7  MISS Left  ->       [8, 7] <- Right Replaced:- [Hits:0 Misses:2]
 Use Bit :  {8: 1, 7: 1, 4: 1}
Access: 4  MISS Left  ->    [8, 7, 4] <- Right Replaced:- [Hits:0 Misses:3]
 Use Bit :  {7: 0, 4: 0, 2: 1}
Access: 2  MISS Left  ->    [2, 7, 4] <- Right Replaced:8 [Hits:0 Misses:4]
 Use Bit :  {4: 0, 2: 1, 5: 1}
Access: 5  MISS Left  ->    [2, 5, 4] <- Right Replaced:7 [Hits:0 Misses:5]
 Use Bit :  {4: 1, 2: 1, 5: 1}
Access: 4  HIT  Left  ->    [2, 5, 4] <- Right Replaced:- [Hits:1 Misses:5]
 Use Bit :  {2: 0, 5: 0, 7: 1}
Access: 7  MISS Left  ->    [2, 5, 7] <- Right Replaced:4 [Hits:1 Misses:6]
 Use Bit :  {5: 0, 7: 1, 3: 1}
Access: 3  MISS Left  ->    [3, 5, 7] <- Right Replaced:2 [Hits:1 Misses:7]
 Use Bit :  {7: 1, 3: 1, 4: 1}
Access: 4  MISS Left  ->    [3, 4, 7] <- Right Replaced:5 [Hits:1 Misses:8]
 Use Bit :  {3: 0, 4: 0, 5: 1}
Access: 5  MISS Left  ->    [3, 4, 5] <- Right Replaced:7 [Hits:1 Misses:9]

FINALSTATS hits 1   misses 9   hitrate 10.00
```

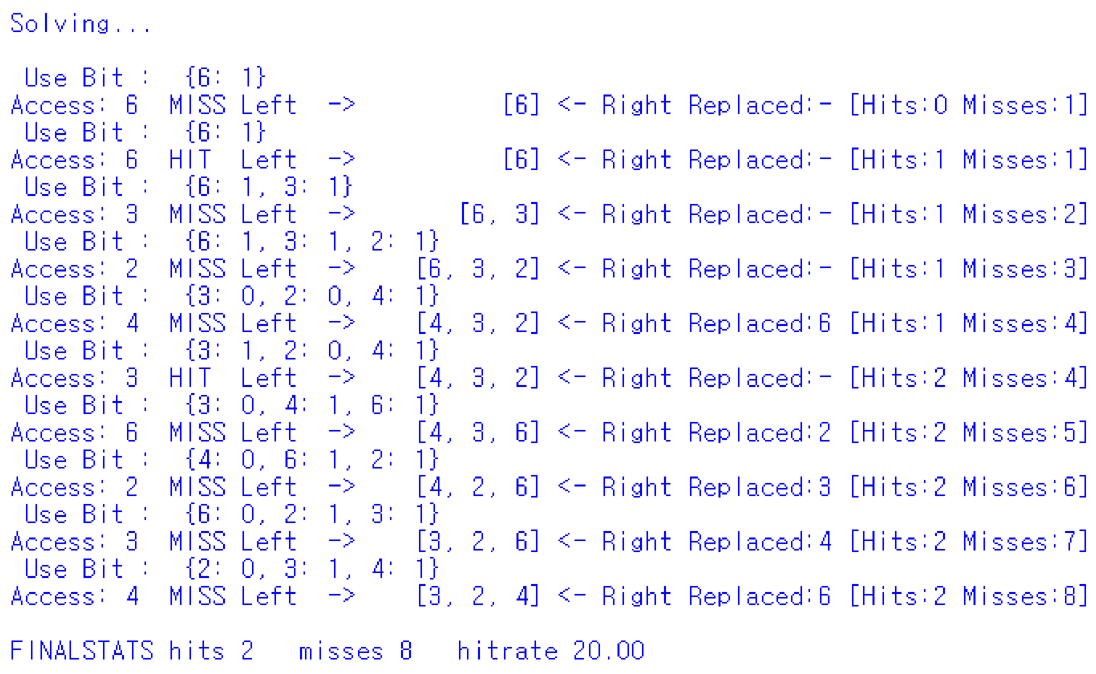
2)MaxPage : 9 / Cache Size : 3



```
Solving...

 Use Bit :  {7: 1}
Access: 7  MISS Left  ->          [7] <- Right Replaced:- [Hits:0 Misses:1]
 Use Bit :  {7: 1, 6: 1}
Access: 6  MISS Left  ->       [7, 6] <- Right Replaced:- [Hits:0 Misses:2]
 Use Bit :  {7: 1, 6: 1, 3: 1}
Access: 3  MISS Left  ->    [7, 6, 3] <- Right Replaced:- [Hits:0 Misses:3]
 Use Bit :  {6: 0, 3: 0, 2: 1}
Access: 2  MISS Left  ->    [2, 6, 3] <- Right Replaced:7 [Hits:0 Misses:4]
 Use Bit :  {3: 0, 2: 1, 4: 1}
Access: 4  MISS Left  ->    [2, 4, 3] <- Right Replaced:6 [Hits:0 Misses:5]
 Use Bit :  {3: 1, 2: 1, 4: 1}
Access: 3  HIT  Left  ->    [2, 4, 3] <- Right Replaced:- [Hits:1 Misses:5]
 Use Bit :  {2: 0, 4: 0, 7: 1}
Access: 7  MISS Left  ->    [2, 4, 7] <- Right Replaced:3 [Hits:1 Misses:6]
 Use Bit :  {2: 1, 4: 0, 7: 1}
Access: 2  HIT  Left  ->    [2, 4, 7] <- Right Replaced:- [Hits:2 Misses:6]
 Use Bit :  {2: 1, 4: 1, 7: 1}
Access: 4  HIT  Left  ->    [2, 4, 7] <- Right Replaced:- [Hits:3 Misses:6]
 Use Bit :  {4: 0, 7: 0, 5: 1}
Access: 5  MISS Left  ->    [5, 4, 7] <- Right Replaced:2 [Hits:3 Misses:7]

FINALSTATS hits 3   misses 7   hitrate 30.00

>>>
```

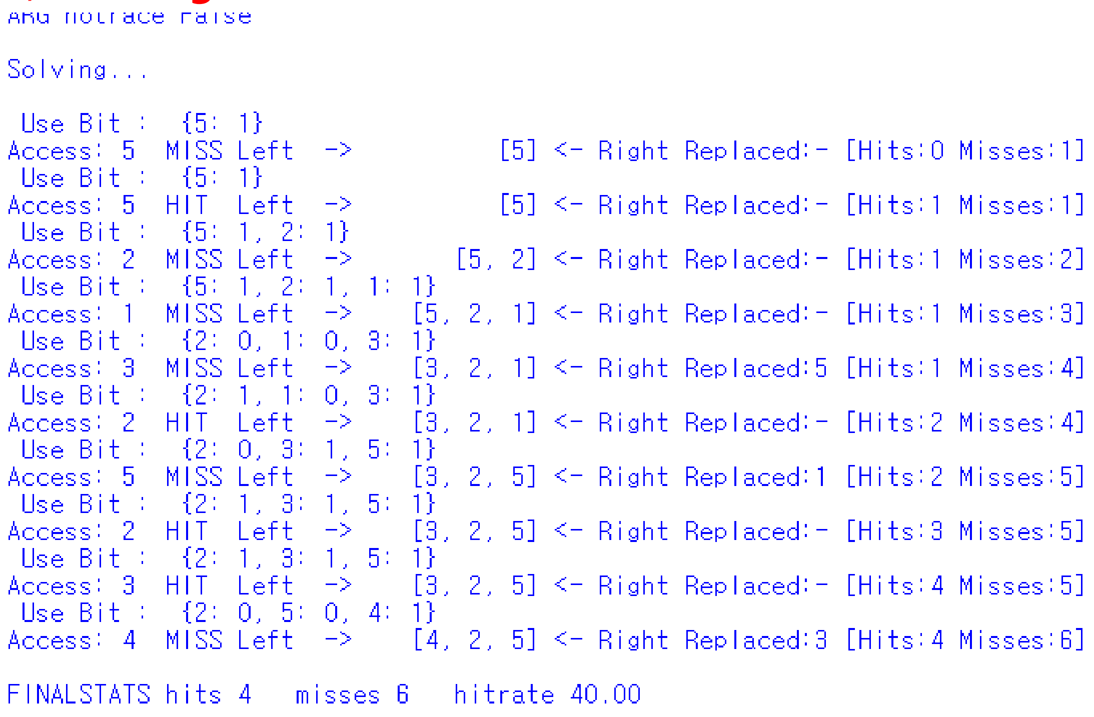
Ln: 339 Col:

3) **MaxPage : 8 / Cache Size : 3**



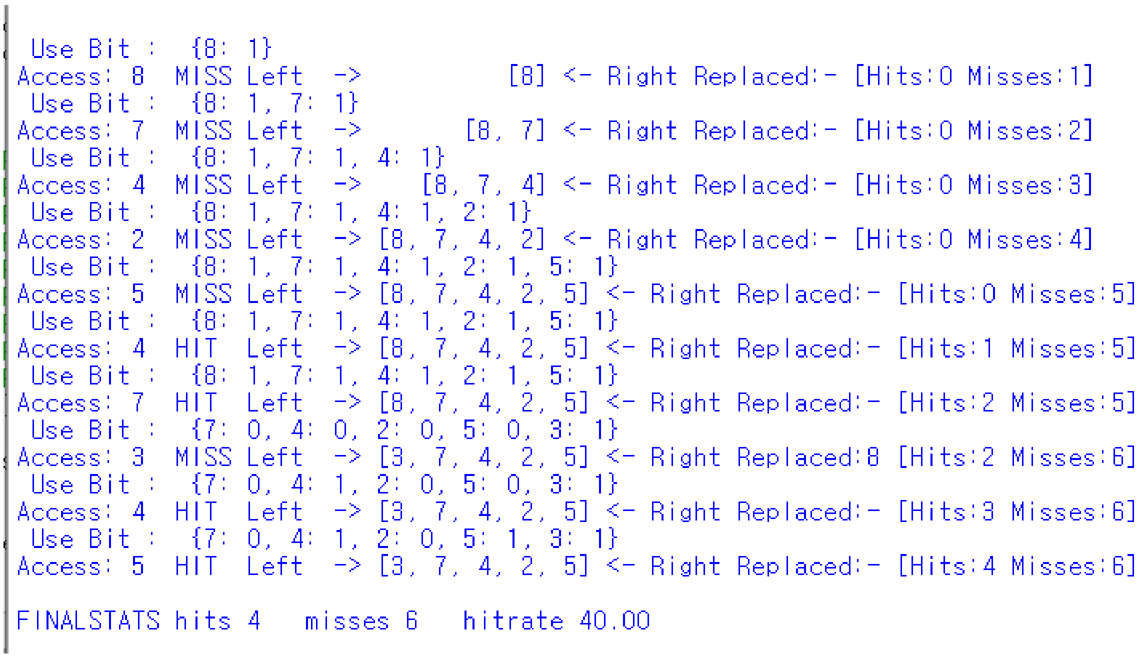
```
Solving...

 Use Bit :  {6: 1}
Access: 6  MISS Left  ->          [6] <- Right Replaced:- [Hits:0 Misses:1]
 Use Bit :  {6: 1}
Access: 6  HIT  Left  ->          [6] <- Right Replaced:- [Hits:1 Misses:1]
 Use Bit :  {6: 1, 3: 1}
Access: 3  MISS Left  ->       [6, 3] <- Right Replaced:- [Hits:1 Misses:2]
 Use Bit :  {6: 1, 3: 1, 2: 1}
Access: 2  MISS Left  ->    [6, 3, 2] <- Right Replaced:- [Hits:1 Misses:3]
 Use Bit :  {3: 0, 2: 0, 4: 1}
Access: 4  MISS Left  ->    [4, 3, 2] <- Right Replaced:6 [Hits:1 Misses:4]
 Use Bit :  {3: 1, 2: 0, 4: 1}
Access: 3  HIT  Left  ->    [4, 3, 2] <- Right Replaced:- [Hits:2 Misses:4]
 Use Bit :  {3: 0, 4: 1, 6: 1}
Access: 6  MISS Left  ->    [4, 3, 6] <- Right Replaced:2 [Hits:2 Misses:5]
 Use Bit :  {4: 0, 6: 1, 2: 1}
Access: 2  MISS Left  ->    [4, 2, 6] <- Right Replaced:3 [Hits:2 Misses:6]
 Use Bit :  {6: 0, 2: 1, 3: 1}
Access: 3  MISS Left  ->    [3, 2, 6] <- Right Replaced:4 [Hits:2 Misses:7]
 Use Bit :  {2: 0, 3: 1, 4: 1}
Access: 4  MISS Left  ->    [3, 2, 4] <- Right Replaced:6 [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

4) **MaxPage : 7 / Cache Size : 3**



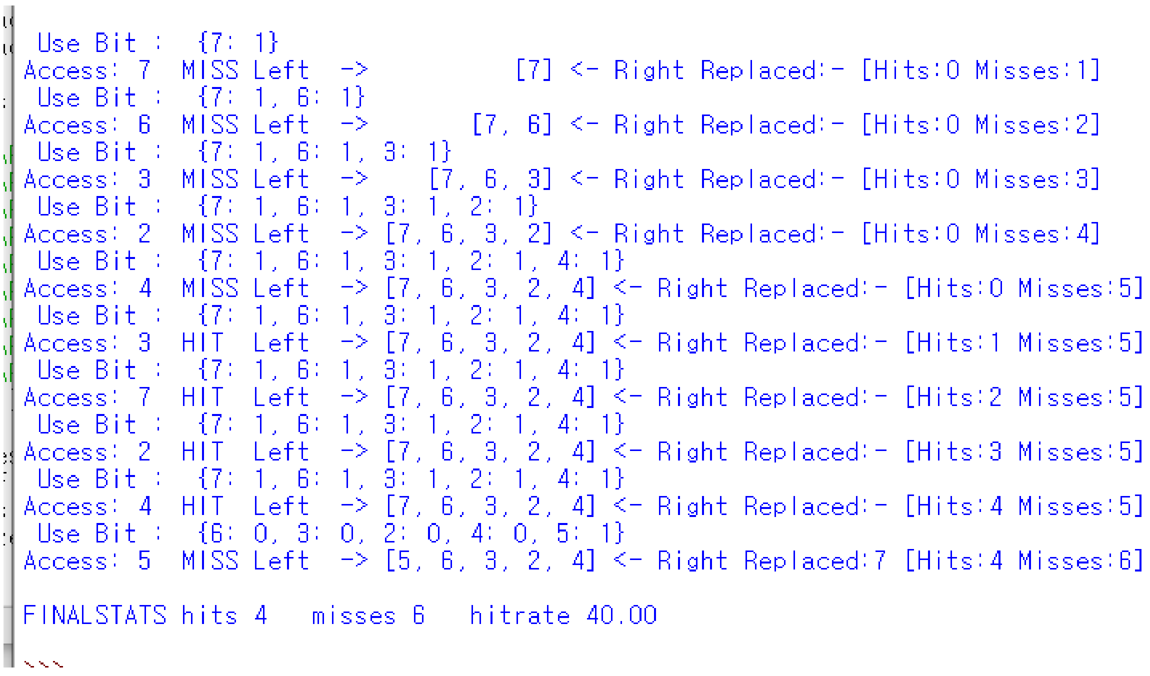
```
ARG notrace False

Solving...

 Use Bit :  {5: 1}
Access: 5  MISS Left  ->          [5] <- Right Replaced:- [Hits:0 Misses:1]
 Use Bit :  {5: 1}
Access: 5  HIT  Left  ->          [5] <- Right Replaced:- [Hits:1 Misses:1]
 Use Bit :  {5: 1, 2: 1}
Access: 2  MISS Left  ->       [5, 2] <- Right Replaced:- [Hits:1 Misses:2]
 Use Bit :  {5: 1, 2: 1, 1: 1}
Access: 1  MISS Left  ->    [5, 2, 1] <- Right Replaced:- [Hits:1 Misses:3]
 Use Bit :  {2: 0, 1: 0, 3: 1}
Access: 3  MISS Left  ->    [3, 2, 1] <- Right Replaced:5 [Hits:1 Misses:4]
 Use Bit :  {2: 1, 1: 0, 3: 1}
Access: 2  HIT  Left  ->    [3, 2, 1] <- Right Replaced:- [Hits:2 Misses:4]
 Use Bit :  {2: 0, 3: 1, 5: 1}
Access: 5  MISS Left  ->    [3, 2, 5] <- Right Replaced:1 [Hits:2 Misses:5]
 Use Bit :  {2: 1, 3: 1, 5: 1}
Access: 2  HIT  Left  ->    [3, 2, 5] <- Right Replaced:- [Hits:3 Misses:5]
 Use Bit :  {2: 1, 3: 1, 5: 1}
Access: 3  HIT  Left  ->    [3, 2, 5] <- Right Replaced:- [Hits:4 Misses:5]
 Use Bit :  {2: 0, 5: 0, 4: 1}
Access: 4  MISS Left  ->    [4, 2, 5] <- Right Replaced:3 [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

1) MaxPage : 10 / Cache Size : 5



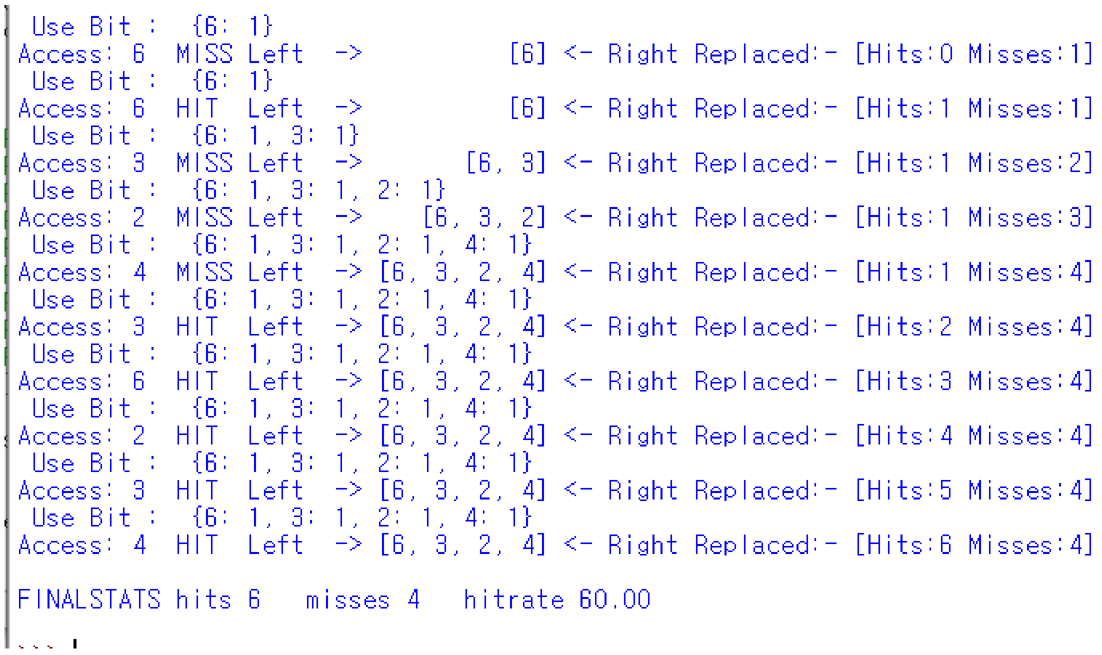
```
 Use Bit :  {8: 1}
Access: 8  MISS Left  ->          [8] <- Right Replaced:- [Hits:0 Misses:1]
 Use Bit :  {8: 1, 7: 1}
Access: 7  MISS Left  ->       [8, 7] <- Right Replaced:- [Hits:0 Misses:2]
 Use Bit :  {8: 1, 7: 1, 4: 1}
Access: 4  MISS Left  ->    [8, 7, 4] <- Right Replaced:- [Hits:0 Misses:3]
 Use Bit :  {8: 1, 7: 1, 4: 1, 2: 1}
Access: 2  MISS Left  -> [8, 7, 4, 2] <- Right Replaced:- [Hits:0 Misses:4]
 Use Bit :  {8: 1, 7: 1, 4: 1, 2: 1, 5: 1}
Access: 5  MISS Left  -> [8, 7, 4, 2, 5] <- Right Replaced:- [Hits:0 Misses:5]
 Use Bit :  {8: 1, 7: 1, 4: 1, 2: 1, 5: 1}
Access: 4  HIT  Left  -> [8, 7, 4, 2, 5] <- Right Replaced:- [Hits:1 Misses:5]
 Use Bit :  {8: 1, 7: 1, 4: 1, 2: 1, 5: 1}
Access: 7  HIT  Left  -> [8, 7, 4, 2, 5] <- Right Replaced:- [Hits:2 Misses:5]
 Use Bit :  {7: 0, 4: 0, 2: 0, 5: 0, 3: 1}
Access: 3  MISS Left  -> [3, 7, 4, 2, 5] <- Right Replaced:8 [Hits:2 Misses:6]
 Use Bit :  {7: 0, 4: 1, 2: 0, 5: 0, 3: 1}
Access: 4  HIT  Left  -> [3, 7, 4, 2, 5] <- Right Replaced:- [Hits:3 Misses:6]
 Use Bit :  {7: 0, 4: 1, 2: 0, 5: 1, 3: 1}
Access: 5  HIT  Left  -> [3, 7, 4, 2, 5] <- Right Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

2) MaxPage : 9 / Cache Size : 5



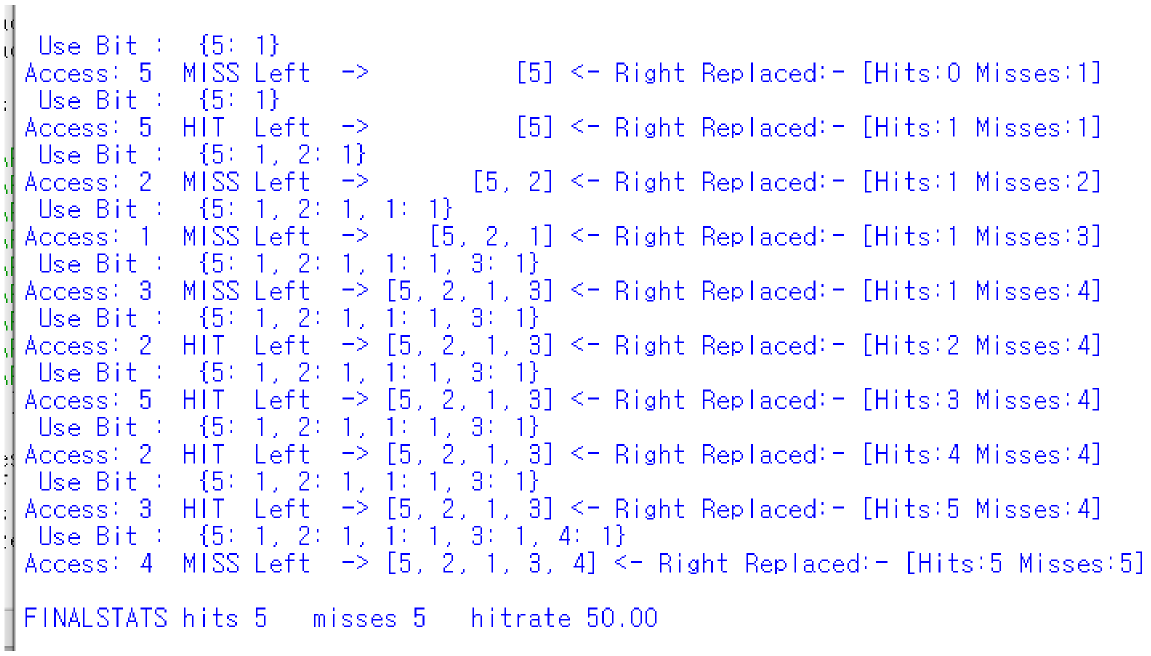
```
 Use Bit :  {7: 1}
Access: 7  MISS Left  ->          [7] <- Right Replaced:- [Hits:0 Misses:1]
 Use Bit :  {7: 1, 6: 1}
Access: 6  MISS Left  ->       [7, 6] <- Right Replaced:- [Hits:0 Misses:2]
 Use Bit :  {7: 1, 6: 1, 3: 1}
Access: 3  MISS Left  ->    [7, 6, 3] <- Right Replaced:- [Hits:0 Misses:3]
 Use Bit :  {7: 1, 6: 1, 3: 1, 2: 1}
Access: 2  MISS Left  -> [7, 6, 3, 2] <- Right Replaced:- [Hits:0 Misses:4]
 Use Bit :  {7: 1, 6: 1, 3: 1, 2: 1, 4: 1}
Access: 4  MISS Left  -> [7, 6, 3, 2, 4] <- Right Replaced:- [Hits:0 Misses:5]
 Use Bit :  {7: 1, 6: 1, 3: 1, 2: 1, 4: 1}
Access: 3  HIT  Left  -> [7, 6, 3, 2, 4] <- Right Replaced:- [Hits:1 Misses:5]
 Use Bit :  {7: 1, 6: 1, 3: 1, 2: 1, 4: 1}
Access: 7  HIT  Left  -> [7, 6, 3, 2, 4] <- Right Replaced:- [Hits:2 Misses:5]
 Use Bit :  {7: 1, 6: 1, 3: 1, 2: 1, 4: 1}
Access: 2  HIT  Left  -> [7, 6, 3, 2, 4] <- Right Replaced:- [Hits:3 Misses:5]
 Use Bit :  {7: 1, 6: 1, 3: 1, 2: 1, 4: 1}
Access: 4  HIT  Left  -> [7, 6, 3, 2, 4] <- Right Replaced:- [Hits:4 Misses:5]
 Use Bit :  {6: 0, 3: 0, 2: 0, 4: 0, 5: 1}
Access: 5  MISS Left  -> [5, 6, 3, 2, 4] <- Right Replaced:7 [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00

```
```
```

3) MaxPage : 8 / Cache Size : 5



```
 Use Bit :  {6: 1}
Access: 6  MISS Left  ->           [6] <- Right Replaced:- [Hits:0 Misses:1]
 Use Bit :  {6: 1}
Access: 6  HIT  Left  ->           [6] <- Right Replaced:- [Hits:1 Misses:1]
 Use Bit :  {6: 1, 3: 1}
Access: 3  MISS Left  ->        [6, 3] <- Right Replaced:- [Hits:1 Misses:2]
 Use Bit :  {6: 1, 3: 1, 2: 1}
Access: 2  MISS Left  ->     [6, 3, 2] <- Right Replaced:- [Hits:1 Misses:3]
 Use Bit :  {6: 1, 3: 1, 2: 1, 4: 1}
Access: 4  MISS Left  -> [6, 3, 2, 4] <- Right Replaced:- [Hits:1 Misses:4]
 Use Bit :  {6: 1, 3: 1, 2: 1, 4: 1}
Access: 3  HIT  Left  -> [6, 3, 2, 4] <- Right Replaced:- [Hits:2 Misses:4]
 Use Bit :  {6: 1, 3: 1, 2: 1, 4: 1}
Access: 6  HIT  Left  -> [6, 3, 2, 4] <- Right Replaced:- [Hits:3 Misses:4]
 Use Bit :  {6: 1, 3: 1, 2: 1, 4: 1}
Access: 2  HIT  Left  -> [6, 3, 2, 4] <- Right Replaced:- [Hits:4 Misses:4]
 Use Bit :  {6: 1, 3: 1, 2: 1, 4: 1}
Access: 3  HIT  Left  -> [6, 3, 2, 4] <- Right Replaced:- [Hits:5 Misses:4]
 Use Bit :  {6: 1, 3: 1, 2: 1, 4: 1}
Access: 4  HIT  Left  -> [6, 3, 2, 4] <- Right Replaced:- [Hits:6 Misses:4]

FINALSTATS hits 6   misses 4   hitrate 60.00
```

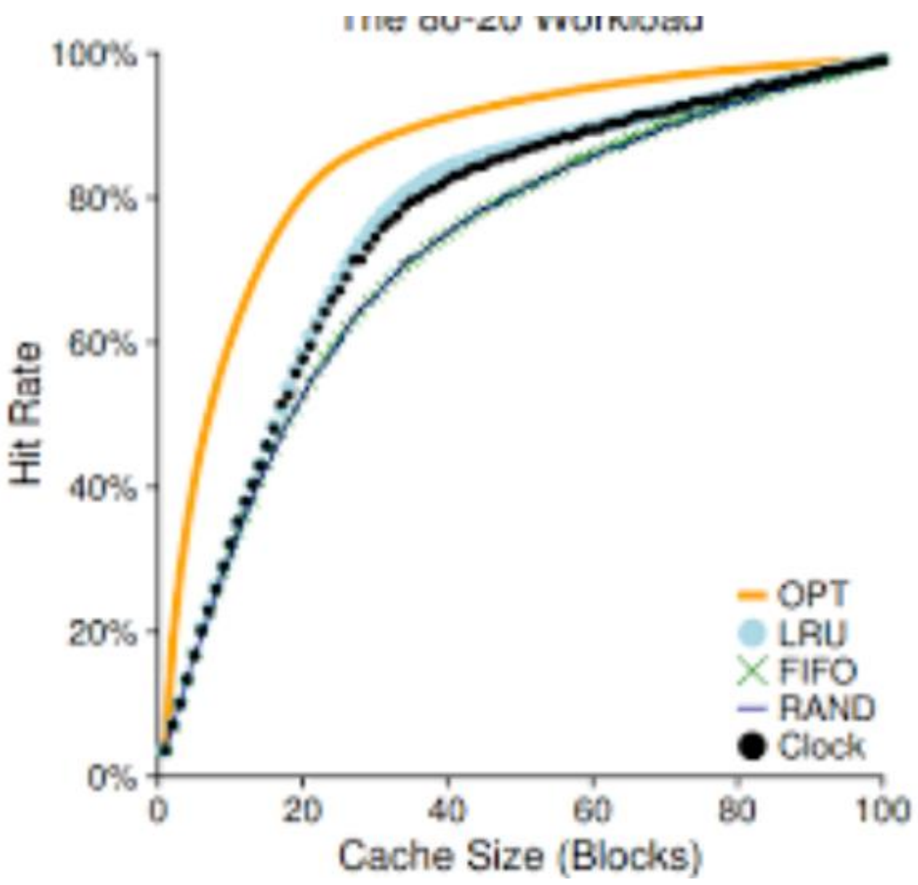
4) MaxPage : 7 / Cache Size : 3



```
 Use Bit :  {5: 1}
Access: 5  MISS Left  ->           [5] <- Right Replaced:- [Hits:0 Misses:1]
 Use Bit :  {5: 1}
Access: 5  HIT  Left  ->           [5] <- Right Replaced:- [Hits:1 Misses:1]
 Use Bit :  {5: 1, 2: 1}
Access: 2  MISS Left  ->        [5, 2] <- Right Replaced:- [Hits:1 Misses:2]
 Use Bit :  {5: 1, 2: 1, 1: 1}
Access: 1  MISS Left  ->     [5, 2, 1] <- Right Replaced:- [Hits:1 Misses:3]
 Use Bit :  {5: 1, 2: 1, 1: 1, 3: 1}
Access: 3  MISS Left  -> [5, 2, 1, 3] <- Right Replaced:- [Hits:1 Misses:4]
 Use Bit :  {5: 1, 2: 1, 1: 1, 3: 1}
Access: 2  HIT  Left  -> [5, 2, 1, 3] <- Right Replaced:- [Hits:2 Misses:4]
 Use Bit :  {5: 1, 2: 1, 1: 1, 3: 1}
Access: 5  HIT  Left  -> [5, 2, 1, 3] <- Right Replaced:- [Hits:3 Misses:4]
 Use Bit :  {5: 1, 2: 1, 1: 1, 3: 1}
Access: 2  HIT  Left  -> [5, 2, 1, 3] <- Right Replaced:- [Hits:4 Misses:4]
 Use Bit :  {5: 1, 2: 1, 1: 1, 3: 1}
Access: 3  HIT  Left  -> [5, 2, 1, 3] <- Right Replaced:- [Hits:5 Misses:4]
 Use Bit :  {5: 1, 2: 1, 1: 1, 3: 1, 4: 1}
Access: 4  MISS Left  -> [5, 2, 1, 3, 4] <- Right Replaced:- [Hits:5 Misses:5]

FINALSTATS hits 5   misses 5   hitrate 50.00
```

설명 : n = int(maxpage * random.random())으로 설정돼있기 때문에 n 값을 변경해주려면 maxpage 의 크기를 다양하게 해주었다, 그리고 cachesize 는 기존 3 으로 설정된 것을 5 까지 늘려서 테스트케이스를 진행해보았다. 그결과 cache 사이즈가 클수록 hitrate 가 증가하는 것을 보였고 maxpage 를 늘려주었지만 랜덤값이 곱해져있기 때문에 hitrate 가 정확하게 측정되지는않지만 maxpage 가 작을수록 hitrate 가 좋아지는 것을 볼 수 있었다.

즉 maxpage 와 cachesize 가 낮을수록 clock 알고리즘은 LRU 에 근사한다는 것을 알 수 있다.



이런식으로 구현가능한 알고리즘중 LRU 에 가장 근사한 알고리즘은 CLOCK 알고리즘이라고 생각 가능하다.

소스 코드 - 충분한 주석

```
#! /usr/bin/env python

# -*- coding: cp949 -*-



import sys

from optparse import OptionParser

import random

import math



def convert(size):

    length = len(size)

    lastchar = size[length-1]

    if (lastchar == 'k') or (lastchar == 'K'):

        m = 1024

        nsize = int(size[0:length-1]) * m

    elif (lastchar == 'm') or (lastchar == 'M'):

        m = 1024*1024

        nsize = int(size[0:length-1]) * m

    elif (lastchar == 'g') or (lastchar == 'G'):

        m = 1024*1024*1024
        nsize = int(size[0:length-1]) * m
    else:
        nsize = int(size)
    return nsize

def hfunc(index):
    if index == -1:
        return 'MISS'
    else:
        return 'HIT '

def vfunc(victim):
    if victim == -1:
        return '-'
    else:
        return str(victim)

#
# main program
#
```

```
parser = OptionParser()
parser.add_option('-a', '--addresses', default='-1',    help='a set of comma-separated pages to access; -1 means randomly generate',  action='store', type='string', dest='addresses')
parser.add_option('-f', '--addressfile', default='',    help='a file with a bunch of addresses in it', action='store', type='string', dest='addressfile')
parser.add_option('-n', '--numaddrs', default='10',    help='if -a (--addresses) is -1, this is the number of addrs to generate', action='store', type='string', dest='numaddrs')
parser.add_option('-p', '--policy', default='CLOCK', help='replacement policy: FIFO, LRU, OPT, UNOPT, RAND, CLOCK', action='store', type='string', dest='policy')
parser.add_option('-b', '--clockbits', default=1,    help='for CLOCK policy, how many clock bits to use', action='store', type='int', dest='clockbits')
parser.add_option('-C', '--cachesize', default='3',    help='size of the page cache, in pages', action='store', type='string', dest='cachesize')
parser.add_option('-m', '--maxpage', default='10',    help='if randomly generating page accesses, this is the max page number',    action='store', type='string', dest='maxpage')
parser.add_option('-s', '--seed', default='0', help='random number seed', action='store', type='string', dest='seed')

parser.add_option('-N', '--notrace', default=False, help='do not print(out a detailed trace', action='store_true', dest='notrace')

parser.add_option('-c', '--compute', default=True, help='compute answers for me', action='store_true', dest='solve')


(options, args) = parser.parse_args()


print('ARG addresses', options.addresses)

print('ARG addressfile', options.addressfile)

print('ARG numaddrs', options.numaddrs)

print('ARG policy', options.policy)

print('ARG clockbits', options.clockbits)

print('ARG cachesize', options.cachesize)

print('ARG maxpage', options.maxpage)

print('ARG seed', options.seed)

print('ARG notrace', options.notrace)
print('')


addresses     = str(options.addresses)

addressFile = str(options.addressfile)

numaddrs      = int(options.numaddrs)

cachesize     = int(options.cachesize)

seed              = int(options.seed)

maxpage        = int(options.maxpage)

policy          = str(options.policy)

notrace        = options.notrace

clockbits     = int(options.clockbits)


random.seed(seed)


addrList = []

if addressFile != '':

    fd = open(addressFile)

    for line in fd:

        addrList.append(int(line))

    fd.close()
```

```
else:
    if addresses == '-1':
        # need to generate addresses
        for i in range(0,numaddrs):
            n = int(maxpage * random.random())
            addrList.append(n)
    else:
        addrList = addresses.split(',')

if options.solve == False:
    print('Assuming a replacement policy of %s, and a cache of
size %d pages,' % (policy, cachesize))
    print('figure out whether each of the following page
references hit or miss')
    print('in the page cache.₩n')

    for n in addrList:
        print('Access: %d  Hit/Miss?  State of Memory?' %
int(n))
    print('')
```

```
else:

    if notrace == False:

        print('Solving...₩n')


    # init memory structure

    count = 0

    memory = []

    hits = 0

    miss = 0

    clock_point = 0 # 클락이 가르키는 방향(클락의 시침)

    change = 0 # 대체될 인덱스 값설정




    if policy == 'FIFO':

        leftStr = 'FirstIn'

        riteStr = 'Lastin '

    elif policy == 'LRU':

        leftStr = 'LRU'

        riteStr = 'MRU'

    elif policy == 'MRU':
        leftStr = 'LRU'

        riteStr = 'MRU'

    elif policy == 'OPT' or policy == 'RAND' or policy ==
'UNOPT' or policy == 'CLOCK':

        leftStr = 'Left '

        riteStr = 'Right'

    else:

        print('Policy %s is not yet implemented' % policy)

        exit(1)


    # track reference bits for clock

    ref    = {}


    cdebug = False


    # need to generate addresses

    addrIndex = 0

    for nStr in addrList:

        # first, lookup

        n = int(nStr)
```

```
try: # Hit 일 때 설정
    idx = memory.index(n)
    hits = hits + 1
    if policy == 'LRU' or policy == 'MRU':
        update = memory.remove(n)
        memory.append(n) # puts it on MRU side
except: # Miss 일 때 설정
    idx = -1
    miss = miss + 1

victim = -1
if idx == -1:
    # miss, replace?
    # print('BUG count, cachesize:', count, cachesize
    if count == cachesize:
        # must replace
        if policy == 'FIFO' or policy == 'LRU':
            victim = memory.pop(0)
        elif policy == 'MRU':
            victim = memory.pop(count-1)
```

```
                elif policy == 'RAND':
                    victim = memory.pop(int(random.random() * count))
                elif policy == 'CLOCK':
                    if cdebug:
                        print('REFERENCE TO PAGE', n)
                        print('MEMORY ', memory)
                        print('REF (b)', ref)

                    # hack: for now, do random
                    # victim = memory.pop(int(random.random() * count))
                    victim = -1
                    while victim == -1: #여기서 page 의 usebit 를 확인하고 0 이면 그값이 victim 이되고 1 이면 usebit 가 0 이되고 clock time 이 +=1 된다.
                        while True:
                            page = memory[clock_point]
                            if ref[page] >= 1: #usebit 가 1 일 때 !!
                                ref[page] -= 1
                                clock_point += 1

                                #ref[page]가 >=1 이라는 것은
        usebit 가 1 로 설정돼있어 전에 사용했다는 것이다 그래서
        세컨찬스를 잃고

                                #ref[page] -= 1 설정해줘 use
        bit 를 0 으로 만들었다, 그리고 클락시침을 +=1 을 해주어서
        클락방향을 바꿔준다

                                if clock_point > (cachesize -1):
        # 캐쉬크기가 3 으로 설정돼있다. 하지만 클락포인트는 0,1,2 만
        필요하기 때문에 시침이 3 이되면 0 으로 바꿈

                                    clock_point = 0

                            else: #usebit 가 0 일 때 !!

                                # this is our victim

                                victim = page

                                clock_point += 1

                                if clock_point > (cachesize -1):
        # 클락포인트 0,1,2 의 세개의 인덱스를 가르키는 것으로 설정

                                    clock_point = 0

                                change = memory.index(page)
        # 대체될 index 설정

                                memory.remove(page)
```

```
                                memory.insert(change,        n)
#파이썬에서는 insert 를 통해 n 값과 인덱스안에 있는
change[인덱스]값을 서로 교환할 수 있다.
                                break

                        # remove old page's ref count
                        if page in memory:
                            assert('BROKEN')
                        del ref[victim]
                        if cdebug:
                            print('VICTIM', page)
                            print('LEN', len(memory))
                            print('MEM', memory)
                            print('REF (a)', ref)

                    elif policy == 'OPT':
                        maxReplace   = -1
                        replaceIdx   = -1
                        replacePage = -1
                        # print('OPT: access %d, memory %s' % (n,
memory)
```

```
                        # print('OPT: replace from FUTURE (%s)' %
addrList[addrIndex+1:]
                        for pageIndex in range(0,count):
                            page = memory[pageIndex]
                            # now, have page 'page' at index
'pageIndex' in memory
                            whenReferenced = len(addrList)
                            # whenReferenced tells us when, in the
future, this was referenced
                            for futureIdx in
range(addrIndex+1,len(addrList)):
                                futurePage =
int(addrList[futureIdx])
                                if page == futurePage:
                                    whenReferenced = futureIdx
                                    break
                            # print('OPT: page %d is referenced
at %d' % (page, whenReferenced)
                            if whenReferenced >= maxReplace:
                                # print('OPT: ??? updating
maxReplace (%d %d %d)' % (replaceIdx, replacePage,
maxReplace)
```

```
                            replaceIdx   = pageIndex
                            replacePage = page
                            maxReplace   = whenReferenced
                            # print('OPT: --> updating maxReplace (%d %d %d)' % (replaceIdx, replacePage, maxReplace)
                    victim = memory.pop(replaceIdx)
                    # print('OPT: replacing page %d (idx:%d) because I saw it in future at %d' % (victim, replaceIdx, whenReferenced)
                elif policy == 'UNOPT':
                    minReplace   = len(addrList) + 1
                    replaceIdx   = -1
                    replacePage = -1
                    for pageIndex in range(0,count):
                        page = memory[pageIndex]
                        # now, have page 'page' at index 'pageIndex' in memory
                        whenReferenced = len(addrList)
                        # whenReferenced tells us when, in the future, this was referenced
                        for futureIdx in
range(addrIndex+1,len(addrList)):
                            futurePage =
int(addrList[futureIdx])
                            if page == futurePage:
                                whenReferenced = futureIdx
                                break
                        if whenReferenced < minReplace:
                            replaceIdx  = pageIndex
                            replacePage = page
                            minReplace  = whenReferenced
                    victim = memory.pop(replaceIdx)
            else: #cache size 가 3 인데 그보다 작은 값들이
있을때 그냥 클락 포인트 +=1 해준다.
                # miss, but no replacement needed (cache not
full)
                victim = -1
                count = count + 1


                clock_point += 1
                if clock_point > (cachesize -1):
```

```
            clock_point = 0

        # now add to memory

        memory.append(n)

        if cdebug:

            print('LEN (a)', len(memory))

        if victim != -1:

            assert(victim not in memory)


# after miss processing, update reference bit

if n not in ref:

    ref[n] = 1

else:

    ref[n] += 1

    if ref[n] > clockbits:

        ref[n] = clockbits


if cdebug:

    print('REF (a)', ref)


if notrace == False:
```

```
                print(" Use Bit : ", ref)
                print('Access: %d  %s %s -> %12s <- %s Replaced:%s [Hits:%d Misses:%d]' % (n, hfunc(idx), leftStr, memory, riteStr, vfunc(victim), hits, miss))
            addrIndex = addrIndex + 1


    print('')
    print('FINALSTATS hits %d   misses %d   hitrate %.2f' % (hits, miss, (100.0*float(hits))/(float(hits)+float(miss))))
    print('')
```