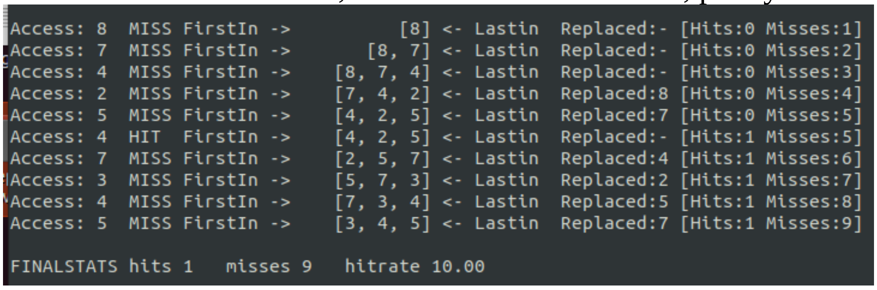
H/W6

2016314364 박수헌

Question 1.

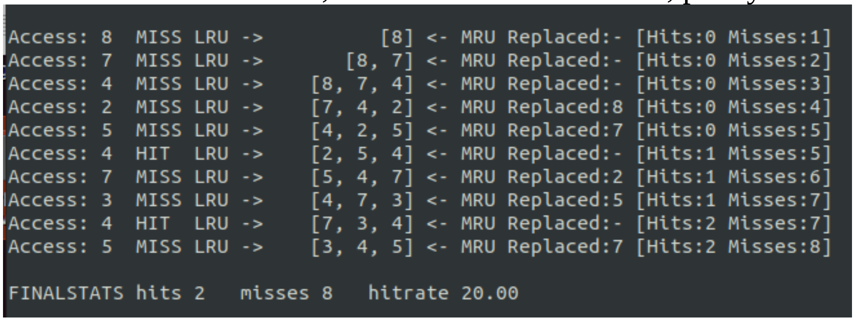
cache size 는 모두 3 으로 설정했다.

-s 0 -n 10 FIFO : random seed number = 0, number of addresses = 10, policy = FIFO



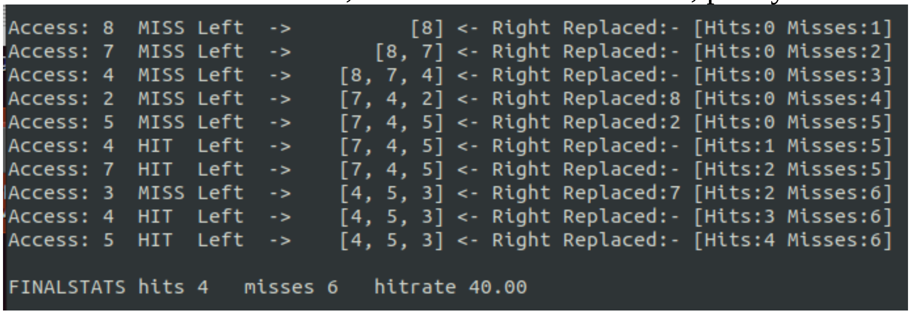
```
Access: 8  MISS FirstIn ->          [8] <- LastIn  Replaced:- [Hits:0 Misses:1]
Access: 7  MISS FirstIn ->       [8, 7] <- LastIn  Replaced:- [Hits:0 Misses:2]
Access: 4  MISS FirstIn ->    [8, 7, 4] <- LastIn  Replaced:- [Hits:0 Misses:3]
Access: 2  MISS FirstIn ->    [7, 4, 2] <- LastIn  Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS FirstIn ->    [4, 2, 5] <- LastIn  Replaced:7 [Hits:0 Misses:5]
Access: 4  HIT  FirstIn ->    [4, 2, 5] <- LastIn  Replaced:- [Hits:1 Misses:5]
Access: 7  MISS FirstIn ->    [2, 5, 7] <- LastIn  Replaced:4 [Hits:1 Misses:6]
Access: 3  MISS FirstIn ->    [5, 7, 3] <- LastIn  Replaced:2 [Hits:1 Misses:7]
Access: 4  MISS FirstIn ->    [7, 3, 4] <- LastIn  Replaced:5 [Hits:1 Misses:8]
Access: 5  MISS FirstIn ->    [3, 4, 5] <- LastIn  Replaced:7 [Hits:1 Misses:9]

FINALSTATS hits 1   misses 9   hitrate 10.00
```

-s 0 -n 10 LRU : random seed number = 0, number of addresses = 10, policy = LRU



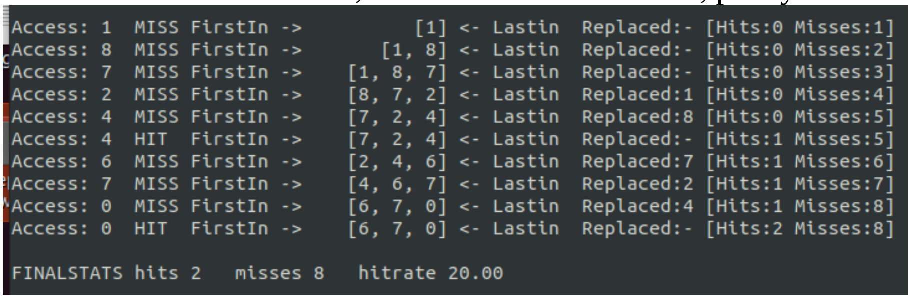
```
Access: 8  MISS LRU ->          [8] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 7  MISS LRU ->       [8, 7] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 4  MISS LRU ->    [8, 7, 4] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 2  MISS LRU ->    [7, 4, 2] <- MRU Replaced:8 [Hits:0 Misses:4]
Access: 5  MISS LRU ->    [4, 2, 5] <- MRU Replaced:7 [Hits:0 Misses:5]
Access: 4  HIT  LRU ->    [2, 5, 4] <- MRU Replaced:- [Hits:1 Misses:5]
Access: 7  MISS LRU ->    [5, 4, 7] <- MRU Replaced:2 [Hits:1 Misses:6]
Access: 3  MISS LRU ->    [4, 7, 3] <- MRU Replaced:5 [Hits:1 Misses:7]
Access: 4  HIT  LRU ->    [7, 3, 4] <- MRU Replaced:- [Hits:2 Misses:7]
Access: 5  MISS LRU ->    [3, 4, 5] <- MRU Replaced:7 [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

-s 0 -n 10 OPT : random seed number = 0, number of addresses = 10, policy = OPT



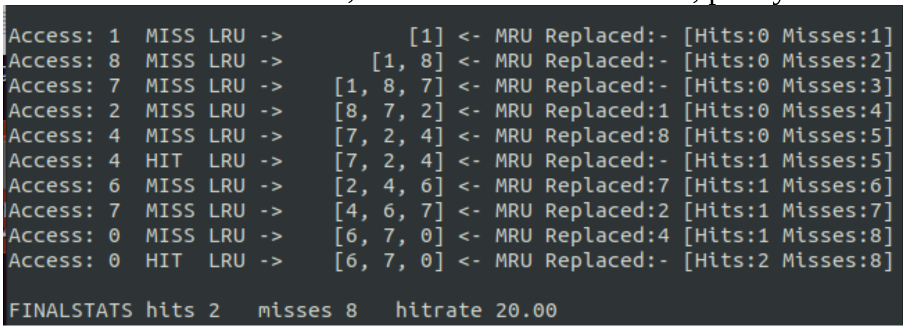
```
Access: 8  MISS Left  ->          [8] <- Right Replaced:- [Hits:0 Misses:1]
Access: 7  MISS Left  ->       [8, 7] <- Right Replaced:- [Hits:0 Misses:2]
Access: 4  MISS Left  ->    [8, 7, 4] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->    [7, 4, 2] <- Right Replaced:8 [Hits:0 Misses:4]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

FINALSTATS hits 4   misses 6   hitrate 40.00
```

-s 1 -n 10 FIFO : random seed number = 0, number of addresses = 10, policy = FIFO



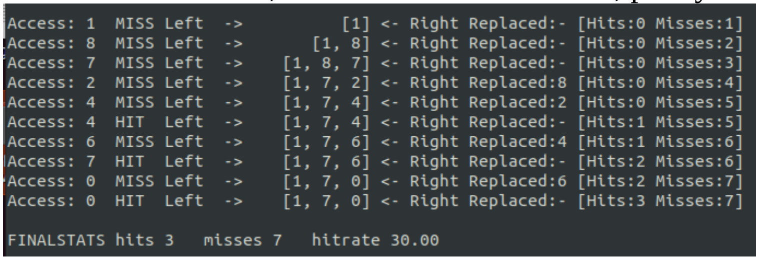
```
Access: 1  MISS FirstIn ->          [1] <- LastIn  Replaced:- [Hits:0 Misses:1]
Access: 8  MISS FirstIn ->       [1, 8] <- LastIn  Replaced:- [Hits:0 Misses:2]
Access: 7  MISS FirstIn ->    [1, 8, 7] <- LastIn  Replaced:- [Hits:0 Misses:3]
Access: 2  MISS FirstIn ->    [8, 7, 2] <- LastIn  Replaced:1 [Hits:0 Misses:4]
Access: 4  MISS FirstIn ->    [7, 2, 4] <- LastIn  Replaced:8 [Hits:0 Misses:5]
Access: 4  HIT  FirstIn ->    [7, 2, 4] <- LastIn  Replaced:- [Hits:1 Misses:5]
Access: 6  MISS FirstIn ->    [2, 4, 6] <- LastIn  Replaced:7 [Hits:1 Misses:6]
Access: 7  MISS FirstIn ->    [4, 6, 7] <- LastIn  Replaced:2 [Hits:1 Misses:7]
Access: 0  MISS FirstIn ->    [6, 7, 0] <- LastIn  Replaced:4 [Hits:1 Misses:8]
Access: 0  HIT  FirstIn ->    [6, 7, 0] <- LastIn  Replaced:- [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

-s 1 -n 10 LRU : random seed number = 0, number of addresses = 10, policy = LRU



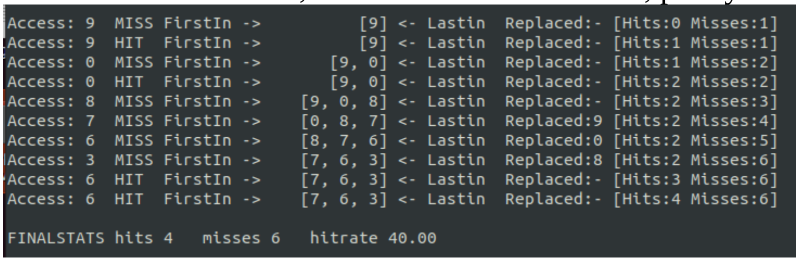
```
Access: 1  MISS LRU ->          [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 8  MISS LRU ->       [1, 8] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 7  MISS LRU ->    [1, 8, 7] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 2  MISS LRU ->    [8, 7, 2] <- MRU Replaced:1 [Hits:0 Misses:4]
Access: 4  MISS LRU ->    [7, 2, 4] <- MRU Replaced:8 [Hits:0 Misses:5]
Access: 4  HIT  LRU ->    [7, 2, 4] <- MRU Replaced:- [Hits:1 Misses:5]
Access: 6  MISS LRU ->    [2, 4, 6] <- MRU Replaced:7 [Hits:1 Misses:6]
Access: 7  MISS LRU ->    [4, 6, 7] <- MRU Replaced:2 [Hits:1 Misses:7]
Access: 0  MISS LRU ->    [6, 7, 0] <- MRU Replaced:4 [Hits:1 Misses:8]
Access: 0  HIT  LRU ->    [6, 7, 0] <- MRU Replaced:- [Hits:2 Misses:8]

FINALSTATS hits 2   misses 8   hitrate 20.00
```

-s 1 -n 10 OPT : random seed number = 0, number of addresses = 10, policy = OPT



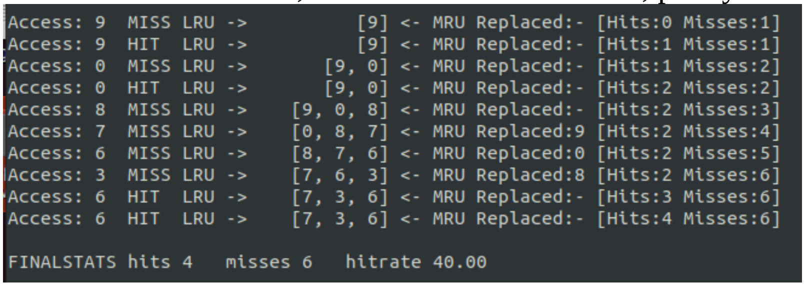
```
Access: 1  MISS Left  ->          [1] <- Right Replaced:- [Hits:0 Misses:1]
Access: 8  MISS Left  ->       [1, 8] <- Right Replaced:- [Hits:0 Misses:2]
Access: 7  MISS Left  ->    [1, 8, 7] <- Right Replaced:- [Hits:0 Misses:3]
Access: 2  MISS Left  ->    [1, 7, 2] <- Right Replaced:8 [Hits:0 Misses:4]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

FINALSTATS hits 3   misses 7   hitrate 30.00
```

-s 2 -n 10 FIFO : random seed number = 0, number of addresses = 10, policy = FIFO



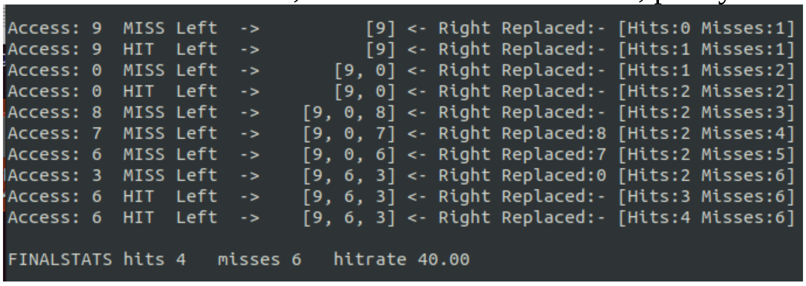
```
Access: 9  MISS FirstIn ->          [9] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 9  HIT  FirstIn ->          [9] <- Lastin  Replaced:- [Hits:1 Misses:1]
Access: 0  MISS FirstIn ->       [9, 0] <- Lastin  Replaced:- [Hits:1 Misses:2]
Access: 0  HIT  FirstIn ->       [9, 0] <- Lastin  Replaced:- [Hits:2 Misses:2]
Access: 8  MISS FirstIn ->    [9, 0, 8] <- Lastin  Replaced:- [Hits:2 Misses:3]
Access: 7  MISS FirstIn ->    [0, 8, 7] <- Lastin  Replaced:9 [Hits:2 Misses:4]
Access: 6  MISS FirstIn ->    [8, 7, 6] <- Lastin  Replaced:0 [Hits:2 Misses:5]
Access: 3  MISS FirstIn ->    [7, 6, 3] <- Lastin  Replaced:8 [Hits:2 Misses:6]
Access: 6  HIT  FirstIn ->    [7, 6, 3] <- Lastin  Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  FirstIn ->    [7, 6, 3] <- Lastin  Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

-s 2 -n 10 LRU : random seed number = 0, number of addresses = 10, policy = LRU



```
Access: 9  MISS LRU ->          [9] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 9  HIT  LRU ->          [9] <- MRU Replaced:- [Hits:1 Misses:1]
Access: 0  MISS LRU ->       [9, 0] <- MRU Replaced:- [Hits:1 Misses:2]
Access: 0  HIT  LRU ->       [9, 0] <- MRU Replaced:- [Hits:2 Misses:2]
Access: 8  MISS LRU ->    [9, 0, 8] <- MRU Replaced:- [Hits:2 Misses:3]
Access: 7  MISS LRU ->    [0, 8, 7] <- MRU Replaced:9 [Hits:2 Misses:4]
Access: 6  MISS LRU ->    [8, 7, 6] <- MRU Replaced:0 [Hits:2 Misses:5]
Access: 3  MISS LRU ->    [7, 6, 3] <- MRU Replaced:8 [Hits:2 Misses:6]
Access: 6  HIT  LRU ->    [7, 3, 6] <- MRU Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  LRU ->    [7, 3, 6] <- MRU Replaced:- [Hits:4 Misses:6]

FINALSTATS hits 4   misses 6   hitrate 40.00
```

-s 2 -n 10 OPT : random seed number = 0, number of addresses = 10, policy = OPT



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

FINALSTATS hits 4   misses 6   hitrate 40.00
```

Question 2.

cache size 가 5 라면 우선 처음 5 번의 access 는 모두 cold-start miss 로 불가피하다.

Worse case address reference stream 이기 위한 조건을 각각 살펴보았다.

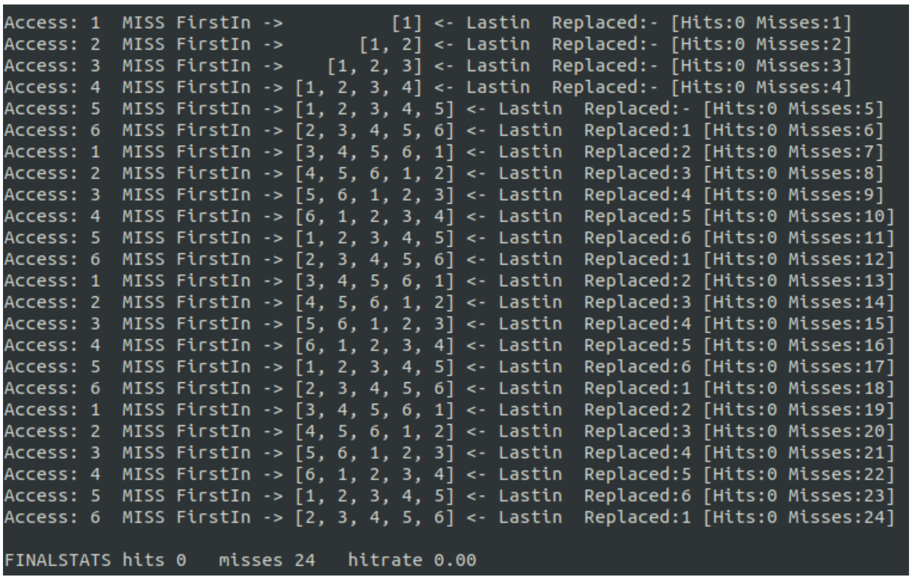
1) FIFO : 가장 먼저 access 된 page 를 evict 한다.

최악의 상황이기 위해선, evict 된 페이지가 바로 다음 instruction 에서 필요하게 되는 경우이다.

1,2,3,4,5,6, 1,2,3,4,5,6, 1,2,3,4,5,6, 1,2,3,4,5,6 같은 반복문을 생각해볼 수 있다.

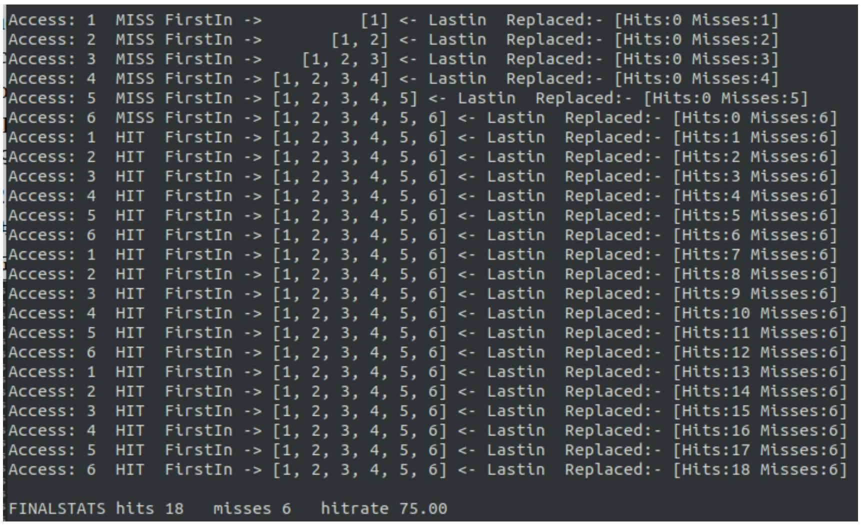
매번 access 되는 페이지는 바로 전에 evict 된 페이지이므로 매우 비효율적이다.

결과는 다음과 같이 모든 access 에서 cache miss 가 발생한다.



```
Access: 1  MISS FirstIn ->          [1] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 2  MISS FirstIn ->       [1, 2] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 3  MISS FirstIn ->    [1, 2, 3] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 4  MISS FirstIn -> [1, 2, 3, 4] <- Lastin  Replaced:- [Hits:0 Misses:4]
Access: 5  MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:0 Misses:5]
Access: 6  MISS FirstIn -> [2, 3, 4, 5, 6] <- Lastin  Replaced:1 [Hits:0 Misses:6]
Access: 1  MISS FirstIn -> [3, 4, 5, 6, 1] <- Lastin  Replaced:2 [Hits:0 Misses:7]
Access: 2  MISS FirstIn -> [4, 5, 6, 1, 2] <- Lastin  Replaced:3 [Hits:0 Misses:8]
Access: 3  MISS FirstIn -> [5, 6, 1, 2, 3] <- Lastin  Replaced:4 [Hits:0 Misses:9]
Access: 4  MISS FirstIn -> [6, 1, 2, 3, 4] <- Lastin  Replaced:5 [Hits:0 Misses:10]
Access: 5  MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin  Replaced:6 [Hits:0 Misses:11]
Access: 6  MISS FirstIn -> [2, 3, 4, 5, 6] <- Lastin  Replaced:1 [Hits:0 Misses:12]
Access: 1  MISS FirstIn -> [3, 4, 5, 6, 1] <- Lastin  Replaced:2 [Hits:0 Misses:13]
Access: 2  MISS FirstIn -> [4, 5, 6, 1, 2] <- Lastin  Replaced:3 [Hits:0 Misses:14]
Access: 3  MISS FirstIn -> [5, 6, 1, 2, 3] <- Lastin  Replaced:4 [Hits:0 Misses:15]
Access: 4  MISS FirstIn -> [6, 1, 2, 3, 4] <- Lastin  Replaced:5 [Hits:0 Misses:16]
Access: 5  MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin  Replaced:6 [Hits:0 Misses:17]
Access: 6  MISS FirstIn -> [2, 3, 4, 5, 6] <- Lastin  Replaced:1 [Hits:0 Misses:18]
Access: 1  MISS FirstIn -> [3, 4, 5, 6, 1] <- Lastin  Replaced:2 [Hits:0 Misses:19]
Access: 2  MISS FirstIn -> [4, 5, 6, 1, 2] <- Lastin  Replaced:3 [Hits:0 Misses:20]
Access: 3  MISS FirstIn -> [5, 6, 1, 2, 3] <- Lastin  Replaced:4 [Hits:0 Misses:21]
Access: 4  MISS FirstIn -> [6, 1, 2, 3, 4] <- Lastin  Replaced:5 [Hits:0 Misses:22]
Access: 5  MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin  Replaced:6 [Hits:0 Misses:23]
Access: 6  MISS FirstIn -> [2, 3, 4, 5, 6] <- Lastin  Replaced:1 [Hits:0 Misses:24]

FINALSTATS hits 0   misses 24   hitrate 0.00
```

하지만 이 경우를 OPT 에 가깝게 개선시키는 방법은 쉽다. 총 6 개의 숫자가 반복되는 반복문이므로, cache size 를 6 으로 늘려주면 한 번의 cache miss 도 일어나지 않는다. 결과는 다음과 같다.

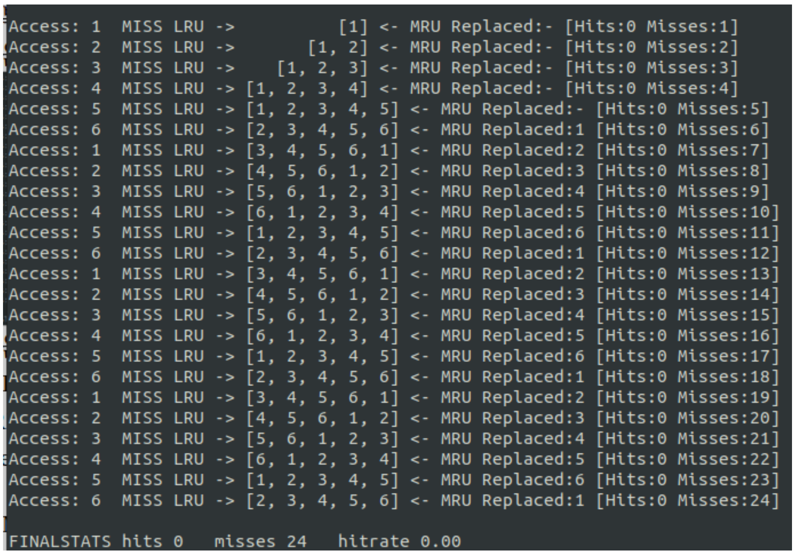


```
Access: 1  MISS FirstIn ->          [1] <- Lastin  Replaced:- [Hits:0 Misses:1]
Access: 2  MISS FirstIn ->       [1, 2] <- Lastin  Replaced:- [Hits:0 Misses:2]
Access: 3  MISS FirstIn ->    [1, 2, 3] <- Lastin  Replaced:- [Hits:0 Misses:3]
Access: 4  MISS FirstIn -> [1, 2, 3, 4] <- Lastin  Replaced:- [Hits:0 Misses:4]
Access: 5  MISS FirstIn -> [1, 2, 3, 4, 5] <- Lastin  Replaced:- [Hits:0 Misses:5]
Access: 6  MISS FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:0 Misses:6]
Access: 1  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:1 Misses:6]
Access: 2  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:2 Misses:6]
Access: 3  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:3 Misses:6]
Access: 4  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:4 Misses:6]
Access: 5  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:5 Misses:6]
Access: 6  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:6 Misses:6]
Access: 1  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:7 Misses:6]
Access: 2  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:8 Misses:6]
Access: 3  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:9 Misses:6]
Access: 4  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:10 Misses:6]
Access: 5  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:11 Misses:6]
Access: 6  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:12 Misses:6]
Access: 1  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:13 Misses:6]
Access: 2  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:14 Misses:6]
Access: 3  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:15 Misses:6]
Access: 4  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:16 Misses:6]
Access: 5  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:17 Misses:6]
Access: 6  HIT  FirstIn -> [1, 2, 3, 4, 5, 6] <- Lastin  Replaced:- [Hits:18 Misses:6]

FINALSTATS hits 18   misses 6   hitrate 75.00
```

처음 6 번의 cold-start miss 를 제외하고는 한 번의 miss 도 발생하지 않았다.

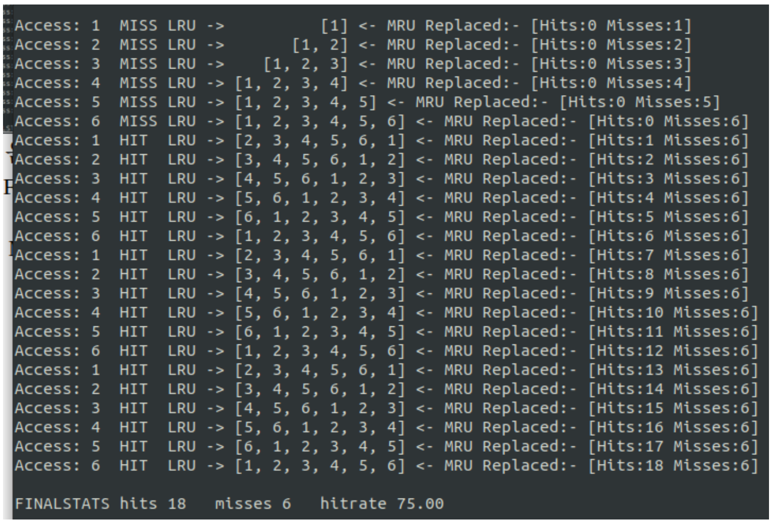
2) LRU : 가장 오래 사용되지 않은 page 를 evict 한다.

LRU 도 FIFO 처럼 반복문에서 최악의 성능을 보여준다. 따라서 FIFO 에 입력한 것과 같은 stream 을 넣어 실행했다.



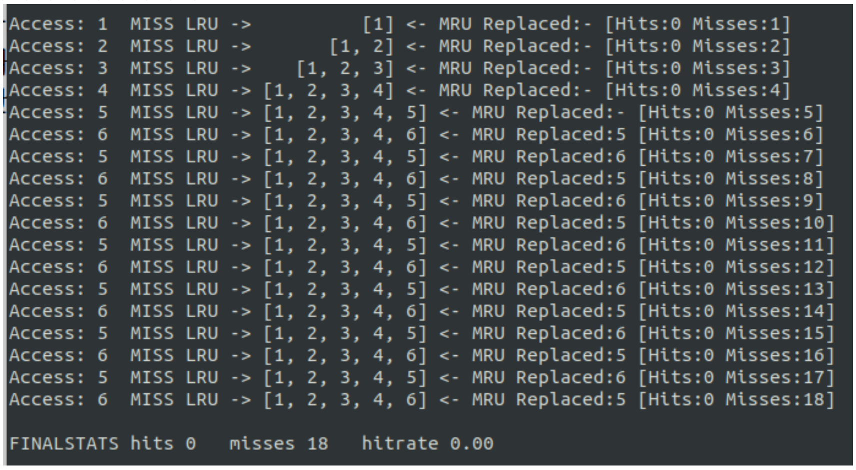
```
Access: 1  MISS LRU ->          [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 2  MISS LRU ->       [1, 2] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 3  MISS LRU ->    [1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 4  MISS LRU -> [1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 6  MISS LRU -> [2, 3, 4, 5, 6] <- MRU Replaced:1 [Hits:0 Misses:6]
Access: 1  MISS LRU -> [3, 4, 5, 6, 1] <- MRU Replaced:2 [Hits:0 Misses:7]
Access: 2  MISS LRU -> [4, 5, 6, 1, 2] <- MRU Replaced:3 [Hits:0 Misses:8]
Access: 3  MISS LRU -> [5, 6, 1, 2, 3] <- MRU Replaced:4 [Hits:0 Misses:9]
Access: 4  MISS LRU -> [6, 1, 2, 3, 4] <- MRU Replaced:5 [Hits:0 Misses:10]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:11]
Access: 6  MISS LRU -> [2, 3, 4, 5, 6] <- MRU Replaced:1 [Hits:0 Misses:12]
Access: 1  MISS LRU -> [3, 4, 5, 6, 1] <- MRU Replaced:2 [Hits:0 Misses:13]
Access: 2  MISS LRU -> [4, 5, 6, 1, 2] <- MRU Replaced:3 [Hits:0 Misses:14]
Access: 3  MISS LRU -> [5, 6, 1, 2, 3] <- MRU Replaced:4 [Hits:0 Misses:15]
Access: 4  MISS LRU -> [6, 1, 2, 3, 4] <- MRU Replaced:5 [Hits:0 Misses:16]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:17]
Access: 6  MISS LRU -> [2, 3, 4, 5, 6] <- MRU Replaced:1 [Hits:0 Misses:18]
Access: 1  MISS LRU -> [3, 4, 5, 6, 1] <- MRU Replaced:2 [Hits:0 Misses:19]
Access: 2  MISS LRU -> [4, 5, 6, 1, 2] <- MRU Replaced:3 [Hits:0 Misses:20]
Access: 3  MISS LRU -> [5, 6, 1, 2, 3] <- MRU Replaced:4 [Hits:0 Misses:21]
Access: 4  MISS LRU -> [6, 1, 2, 3, 4] <- MRU Replaced:5 [Hits:0 Misses:22]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:23]
Access: 6  MISS LRU -> [2, 3, 4, 5, 6] <- MRU Replaced:1 [Hits:0 Misses:24]

FINALSTATS hits 0   misses 24   hitrate 0.00
```

모든 access 에서 miss 가 발생하는 것을 볼 수 있다.
FIFO 처럼 cache size 를 6 으로 증가시키면 급격한 성능 개선을 볼 수 있다.



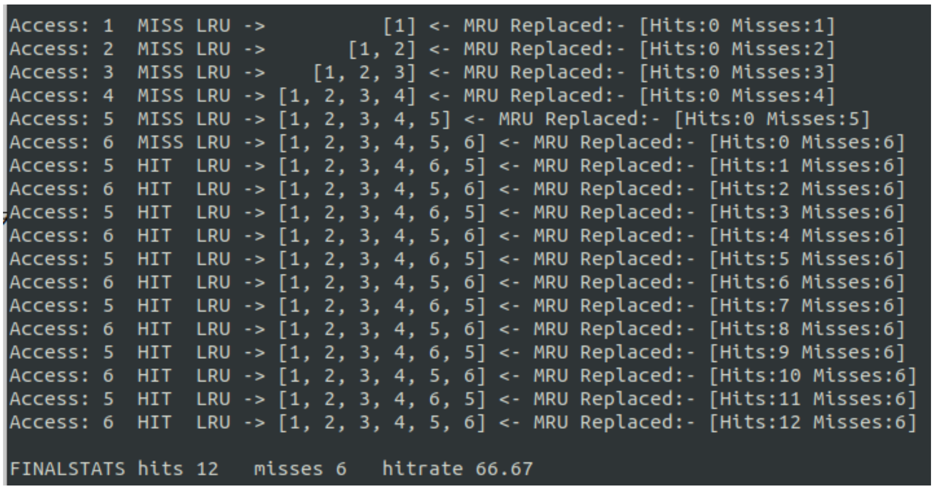
```
Access: 1  MISS LRU ->          [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 2  MISS LRU ->       [1, 2] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 3  MISS LRU ->    [1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 4  MISS LRU -> [1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 6  MISS LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:0 Misses:6]
Access: 1  HIT  LRU -> [2, 3, 4, 5, 6, 1] <- MRU Replaced:- [Hits:1 Misses:6]
Access: 2  HIT  LRU -> [3, 4, 5, 6, 1, 2] <- MRU Replaced:- [Hits:2 Misses:6]
Access: 3  HIT  LRU -> [4, 5, 6, 1, 2, 3] <- MRU Replaced:- [Hits:3 Misses:6]
Access: 4  HIT  LRU -> [5, 6, 1, 2, 3, 4] <- MRU Replaced:- [Hits:4 Misses:6]
Access: 5  HIT  LRU -> [6, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:5 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:6 Misses:6]
Access: 1  HIT  LRU -> [2, 3, 4, 5, 6, 1] <- MRU Replaced:- [Hits:7 Misses:6]
Access: 2  HIT  LRU -> [3, 4, 5, 6, 1, 2] <- MRU Replaced:- [Hits:8 Misses:6]
Access: 3  HIT  LRU -> [4, 5, 6, 1, 2, 3] <- MRU Replaced:- [Hits:9 Misses:6]
Access: 4  HIT  LRU -> [5, 6, 1, 2, 3, 4] <- MRU Replaced:- [Hits:10 Misses:6]
Access: 5  HIT  LRU -> [6, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:11 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:12 Misses:6]
Access: 1  HIT  LRU -> [2, 3, 4, 5, 6, 1] <- MRU Replaced:- [Hits:13 Misses:6]
Access: 2  HIT  LRU -> [3, 4, 5, 6, 1, 2] <- MRU Replaced:- [Hits:14 Misses:6]
Access: 3  HIT  LRU -> [4, 5, 6, 1, 2, 3] <- MRU Replaced:- [Hits:15 Misses:6]
Access: 4  HIT  LRU -> [5, 6, 1, 2, 3, 4] <- MRU Replaced:- [Hits:16 Misses:6]
Access: 5  HIT  LRU -> [6, 1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:17 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:18 Misses:6]

FINALSTATS hits 18   misses 6   hitrate 75.00
```

처음 6 번의 cold-start miss 를 제외하고 모든 access 에서 cache hit 가 발생한다.
3) MRU : 가장 가장 최근에 사용된 page 를 evict 한다.
우선 처음 cache 를 1,2,3,4,5 로 채운다. 이 때 6 이 들어오면 5 가 evict 된다. 가장 안 좋은 성능을 내려면 가장 최근에 evict 된 page 를 다시 access 하는 것이다. 따라서 그 다음엔 5 page 를 access 한다. 이 때, evic 되는 page 는 6 이므로 그 다음 access 는 6, 이런 식으로 반복해볼 수 있다.
1,2,3,4,5,6, 5,6,5,6,5,6,5,6,5,6,5,6 의 stream 으로 실행해 보았다.



```
Access: 1  MISS LRU ->          [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 2  MISS LRU ->       [1, 2] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 3  MISS LRU ->    [1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 4  MISS LRU -> [1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 6  MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:6]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:7]
Access: 6  MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:8]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:9]
Access: 6  MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:10]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:11]
Access: 6  MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:12]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:13]
Access: 6  MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:14]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:15]
Access: 6  MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:16]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:6 [Hits:0 Misses:17]
Access: 6  MISS LRU -> [1, 2, 3, 4, 6] <- MRU Replaced:5 [Hits:0 Misses:18]

FINALSTATS hits 0   misses 18   hitrate 0.00
```

모든 access 에서 miss 가 발생한 것을 볼 수 있다.
이 성능을 개선시키려면 위의 두 경우와 같이 cache size 를 6 으로 증가시키면 된다.



```
Access: 1  MISS LRU ->          [1] <- MRU Replaced:- [Hits:0 Misses:1]
Access: 2  MISS LRU ->       [1, 2] <- MRU Replaced:- [Hits:0 Misses:2]
Access: 3  MISS LRU ->    [1, 2, 3] <- MRU Replaced:- [Hits:0 Misses:3]
Access: 4  MISS LRU -> [1, 2, 3, 4] <- MRU Replaced:- [Hits:0 Misses:4]
Access: 5  MISS LRU -> [1, 2, 3, 4, 5] <- MRU Replaced:- [Hits:0 Misses:5]
Access: 6  MISS LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:0 Misses:6]
Access: 5  HIT  LRU -> [1, 2, 3, 4, 6, 5] <- MRU Replaced:- [Hits:1 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:2 Misses:6]
Access: 5  HIT  LRU -> [1, 2, 3, 4, 6, 5] <- MRU Replaced:- [Hits:3 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:4 Misses:6]
Access: 5  HIT  LRU -> [1, 2, 3, 4, 6, 5] <- MRU Replaced:- [Hits:5 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:6 Misses:6]
Access: 5  HIT  LRU -> [1, 2, 3, 4, 6, 5] <- MRU Replaced:- [Hits:7 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:8 Misses:6]
Access: 5  HIT  LRU -> [1, 2, 3, 4, 6, 5] <- MRU Replaced:- [Hits:9 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:10 Misses:6]
Access: 5  HIT  LRU -> [1, 2, 3, 4, 6, 5] <- MRU Replaced:- [Hits:11 Misses:6]
Access: 6  HIT  LRU -> [1, 2, 3, 4, 5, 6] <- MRU Replaced:- [Hits:12 Misses:6]

FINALSTATS hits 12   misses 6   hitrate 66.67
```

처음 6 번의 cold-strart miss 를 제외하고 모두 cahce hit 가 발생한다.

Question 3.
README.md 를 확인하면 -s argument 를 통해 random number seed 를 지정할 수 있다.
-n 의 숫자를 이용해 해당 random seed number 를 이용해 random 한 수의 trace 를 생성해낼 수 있다. 따라서 본인은 -s 100 -n 30 (random seed number 100 을 사용해 생성한 20 개의 난수)와 cache size 는 5 를 사용했다.
이런 random 한 trace 에 대해서는 locality 가 존재하지 않을 가능성이 크기 때문에 OPT 만 상대적으로 높고 나머지 policy 들은 비슷한 성능을 보일 것으로 예상했다.
사용된 trace 는 다음과 같았다.
[1, 4, 7, 7, 7, 4, 8, 5, 0, 4, 0, 9, 9, 3, 3, 7, 2, 1, 1, 3, 6, 9, 2, 9, 5, 9, 8, 1, 6, 1]

1) OPT 결과
```
FINALSTATS hits 19          misses 11       hitrate 63.33
```
1) FIFO 결과
```
FINALSTATS hits 15          misses 15       hitrate 50.00
```
2) LRU 결과
```
FINALSTATS hits 13          misses 17       hitrate 43.33
```
3) MRU 결과
```
FINALSTATS hits 15          misses 15       hitrate 50.00
```
4) RAND 결과
```
FINALSTATS hits 14          misses 16       hitrate 46.67
```
5) CLOCK 결과
```
FINALSTATS hits 14          misses 16       hitrate 46.67
```

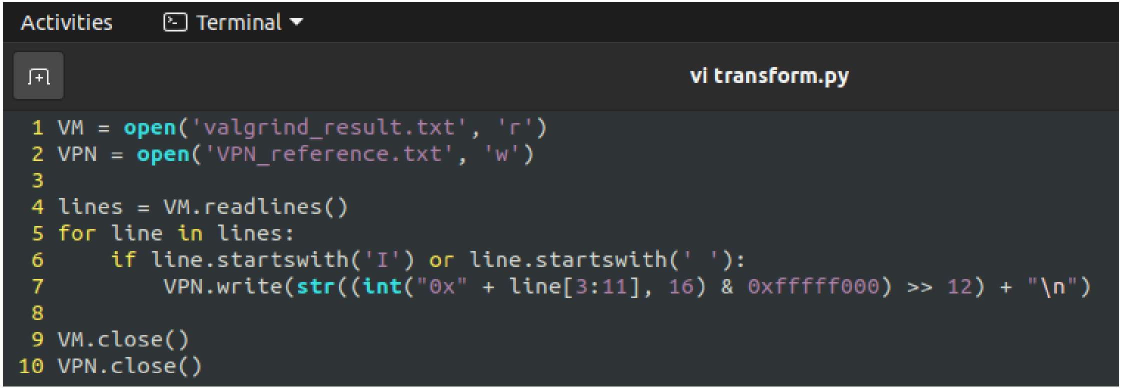
결과는 예상했던 대로 OPT 에서 제일 높은, 나머지에서는 비슷한 hitrate 을 보였다.

Question 4.
locality 가 있는 trace 를 다음과 같이 설정했다.
[1, 2, 3, 3, 3, 3, 4, 2, 3, 3, 5, 2, 2, 6, 3, 3, 2, 2, 2, 3]
20 개의 원소를 갖고 있으며 80-20 workload 를 만족하는 trace 이다.
Cache size 는 3 으로 설정했다.

1) OPT 결과
```
FINALSTATS hits 14   misses 6   hitrate 70.00
```
2) LRU 결과
```
FINALSTATS hits 13   misses 7   hitrate 65.00
```
3) RAND 결과
```
FINALSTATS hits 12   misses 8   hitrate 60.00
```
OPT 가 역시 제일 높은 hitrate 을 보였고, LRU 가 RAND 보다 5% 높은 hitrate 을 보였다.

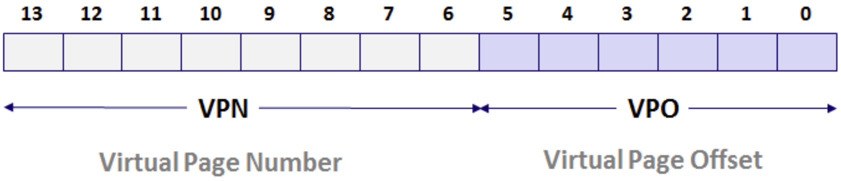
1) CLOCK (use bit = 2)
```
FINALSTATS hits 10   misses 10   hitrate 50.00
```
2) CLOCK (use bit = 3)
```
FINALSTATS hits 13   misses 7   hitrate 65.00
```
3) CLOCK (use bit = 4)
```
FINALSTATS hits 13   misses 7   hitrate 65.00
```
4) CLOCK (use bit = 5)
```
FINALSTATS hits 13   misses 7   hitrate 65.00
```

Question 5.
valgrind --tool=lackey --trace-mem=yes ls &> valgrind_result.txt
의 명령어를 통해 valgrind 의 결과를 valgrind_result.txt 라는 파일로 redirect 해 저장했다.
아래의 코드를 작성해 valgrind_result.txt 파일을 virtual page-number reference 로 변환했다.

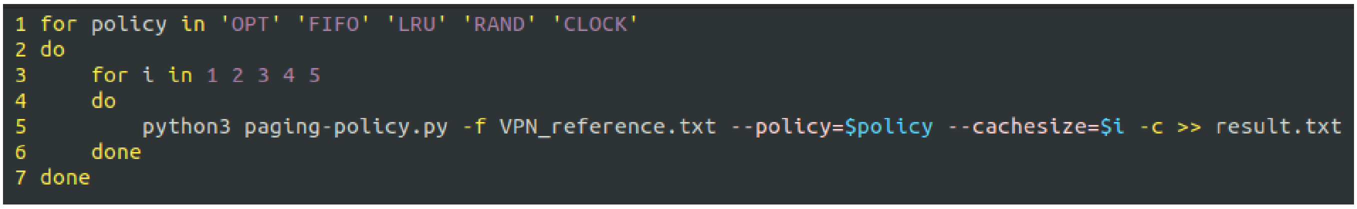


```python
VM = open('valgrind_result.txt', 'r')
VPN = open('VPN_reference.txt', 'w')

lines = VM.readlines()
for line in lines:
    if line.startswith('I') or line.startswith(' '):
        VPN.write(str((int("0x" + line[3:11], 16) & 0xfffff000) >> 12) + "\n")

VM.close()
VPN.close()
```

우선, 각 instruction 들의 값을 16 진수로 바꾸었다. 그리고 이 값을 0xfffff000 과 and 연산 후 오른쪽으로 12 bit 만큼 shift 했다.

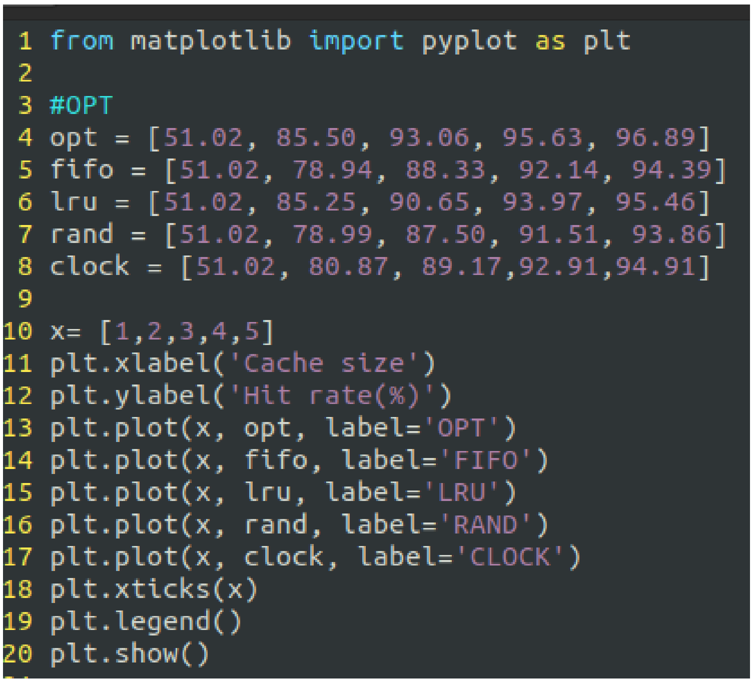


그 이유는, 위 그림처럼 Virtual Page Number 는 VirtualMemory 의 상위 20 개 bit 에 존재하기 때문이다.
그 후, 아래와 같이 shell file 을 만들어 실행했다.



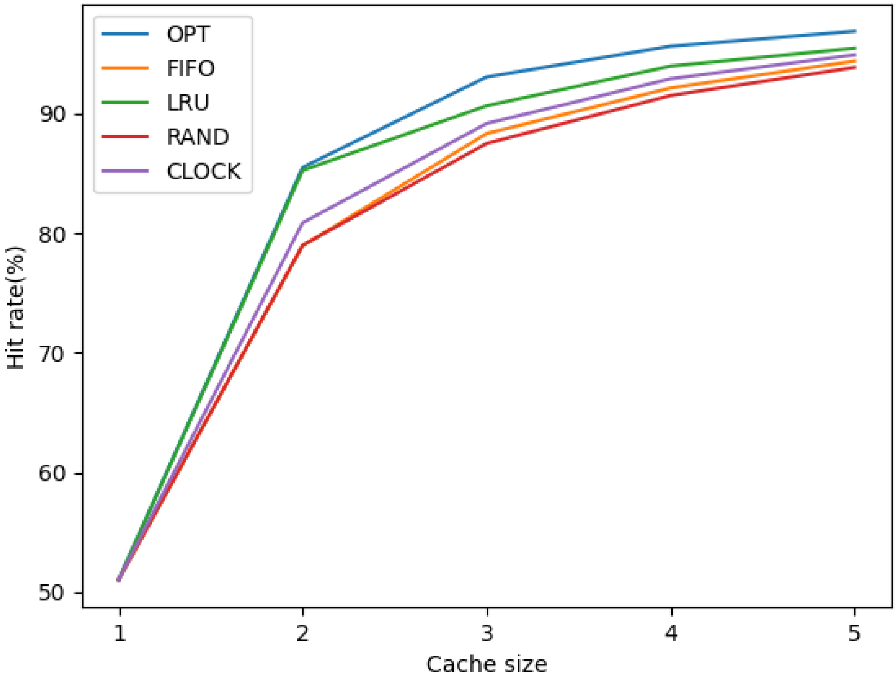
```bash
for policy in 'OPT' 'FIFO' 'LRU' 'RAND' 'CLOCK'
do
    for i in 1 2 3 4 5
    do
        python3 paging-policy.py -f VPN_reference.txt --policy=$policy --cachesize=$i -c >> result.txt
    done
done
```

나온 결과는 result.txt 파일에 저장되었고 이에 대한 plot 을 다음과 같은 코드를 작성해 만들었다.



```python
from matplotlib import pyplot as plt

#OPT
opt = [51.02, 85.50, 93.06, 95.63, 96.89]
fifo = [51.02, 78.94, 88.33, 92.14, 94.39]
lru = [51.02, 85.25, 90.65, 93.97, 95.46]
rand = [51.02, 78.99, 87.50, 91.51, 93.86]
clock = [51.02, 80.87, 89.17,92.91,94.91]

x= [1,2,3,4,5]
plt.xlabel('Cache size')
plt.ylabel('Hit rate(%)')
plt.plot(x, opt, label='OPT')
plt.plot(x, fifo, label='FIFO')
plt.plot(x, lru, label='LRU')
plt.plot(x, rand, label='RAND')
plt.plot(x, clock, label='CLOCK')
plt.xticks(x)
plt.legend()
plt.show()
```

그리고 이에 따른 결과는 아래와 같았다.



대략 cache size 가 4 인 순간부터 hit rate 이 모두 90% 정도를 기록하는 것을 볼 수 있다.