CLOCK 알고리즘 시뮬레이션

Hyokyung Bahn

EWHA WOMANS UNIVERSITY

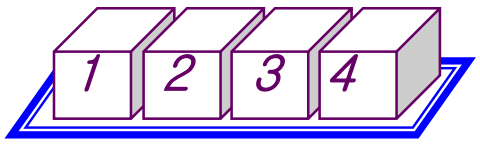
CLOCK 알고리즘 시뮬레이션

- Memory size와 reference string이 주어질 때 CLOCK 알고리즘이 발생시키는 page fault 횟수와 디스크 쓰기 연산 횟수, 디스크 헤드의 이동 거리를 구하시오.

Reference string

1(r), 2(r), 3(r), 4(r), 1(w), 2(w), 5(r), 1(r), 6(r),…

Memory size = 4



Page fault count

352회

Disk write count

174회

Disk head moving distance

97432

프로그램 구현 환경 및 방법

- 구현 환경
  - ✓ C 언어로 구현 (C++ 컴파일러 사용가능, C++ 문법 사용 가능)
    (다만, class 등 객체지향 기법으로 코딩하지는 말 것)
- 구현 방법
  - ✓ 실행 형식 **/* 반드시 형식 엄수 – 형식이 다르면 0점 처리 */**
    - <실행파일명> <메모리크기>
    - (예) 1871000.exe 2000
  - ✓ Input file 형식
    - Reference string이 저장된 파일로 파일 이름은 a.txt
    - 각 라인은 "페이지 번호"와 "연산종류"가 정수로 표시
    - "페이지 번호"는 0 이상 200,000 미만의 값
    - "연산종류"는 read인 경우 0, write인 경우 1
  - ✓ Output 형식
    - 화면에 page fault 횟수, 디스크 쓰기 연산 횟수, 디스크 헤드의 이동 거리를 각각 정수값으로 표시

```
:
12000 1
12376 0
12112 0
12112 1
:
```

시뮬레이션 시나리오

페이지 번호	메모리 위치	valid
0		i
1	3	v
2		i
3	1	v
4		i
5	2	v
:		i
450	0	v
:		i
199,999		i

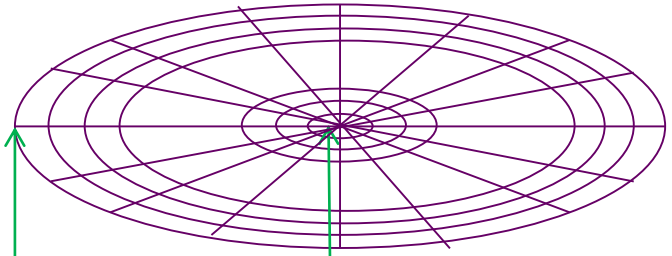
```
struct PAGE
{    int memory_location;
     int valid_bit;
};
struct PAGE page[200000];
```

메모리 위치	페이지 번호	Reference bit	Dirty bit
0	450		
1	3		
2	5		
3	1		

```
struct MEMORY
{    int page_num;
     int reference_bit;
     int dirty_bit
};
struct MEMORY memory[4];
/* 메모리 크기는 가변적 */
/* 다음 슬라이드 참조 */
```

디스크

- 20만개의 페이지가 디스크에 순차적으로 저장돼 있음
- 트랙 당 1000개씩의 페이지가 저장됨
- 디스크 상의 위치(트랙 번호)는 "페이지번호/1000"으로 구함
- 최초의 디스크 헤드 위치는 0번 트랙에서 출발한다고 가정함



디스크 헤드 이동거리 199

트랙 번호 199번

페이지 번호 199,000~199,999까지 저장

트랙 번호 0번

페이지 번호 0~999까지 저장

메모리 크기를 실행 시 입력받아 처리하는 방법

```
struct MEMORY       /* 메모리 자료구조  */
{    int page_num;       /* 이 메모리 위치에 보관하고 있는 page의 번호 */
     int reference_bit;   /* 메모리에 들어온 후 참조(=읽기,쓰기)가 발생할 때마다 1로 셋팅 */
     int dirty_bit;          /* 쓰기 참조가 일어날 때마다 1로 셋팅 (적어도 1번이상 쓰기 발생 시 값이 1임)
                                 => 쫓겨날 때 디스크에 써줘야 하는 페이지임을 표시*/
};


int main(int argc, char *argv[])
{
        int memory_size = atoi(argv[1]);     /* 실행 시에 입력받은 argument를 숫자로 형변환 */
        FILE * input_file = fopen("a.txt", "r");
        struct MEMORY *memory = (struct MEMORY *) malloc(memory_size * sizeof(struct MEMORY));
        /* 이제 배열처럼 사용 가능 memory[0], memory[1], … */
```

클락 시계바늘과 디스크 헤드의 이동

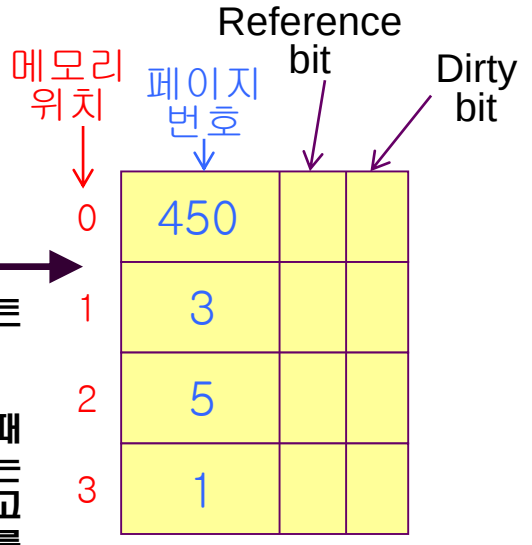
클락 시계 바늘

1. 클락 시계 바늘은 페이지폴트가 났을 때에만 움직인다.

2. 클락 시계 바늘이 지나갈 때 reference bit이 1인 페이지는 0으로 바꾼 후 그냥 지나가고 reference bit이 0인 페이지를 만나면 쫓아낸다.

3. 쫓아낼 때 dirty bit이 1인 페이지는 디스크의 해당 페이지 위치에 써준 후 쫓아낸다.
(디스크 쓰기 연산 발생)

4. 새로 들어온 페이지의 reference bit을 1로 하고, 다음 번 페이지 폴트 시 그 위치부터 검색한다. (=새로 들어온 페이지의 reference bit을 0으로 하고, 다음 번 페이지 폴트 시 그 위치 바로 다음부터 검색한다.)

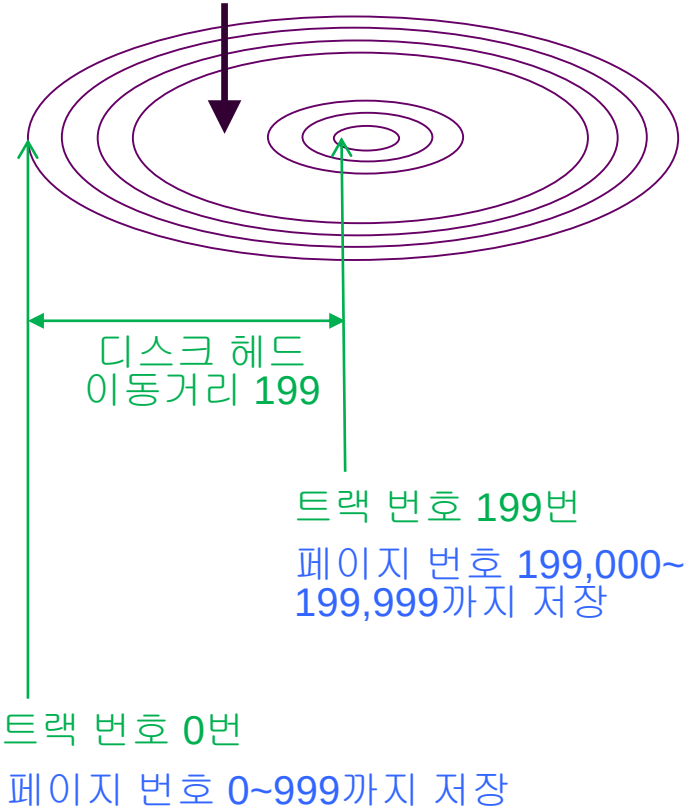


두 가지는 같은 이야기임

디스크 헤드 이동

1. 페이지폴트가 난 페이지의 트랙 위치 (페이지번호/1000)로 디스크 헤드가 이동한다.

2. 단, 메모리에서 쫓겨나는 페이지의 dirty bit이 셋팅된 경우 쫓겨나는 페이지의 트랙 위치에 먼저 가서 페이지를 저장한 후 페이지폴트를 낸 페이지의 트랙위치로 이동한다.



주의사항

- 결과 출력시 결과값 숫자만 3줄로 순서대로 출력요망
- 주석문은 평가 대상이 아니며, 본인 필요에 따라 작성 가능함 (한글로 된 주석문 사용하지 말 것)
- 쫓아낼 페이지 검색을 위해 메모리 내의 모든 페이지를 매번 스캔하는 방식을 사용하지 말것
- 소스코드(.c)만 과제함에 제출할 것
  - ✓ 파일이름은 “본인학번.c”로 할 것

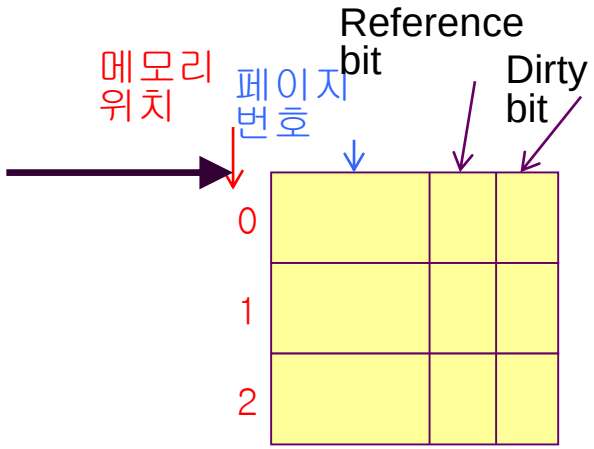
예 제

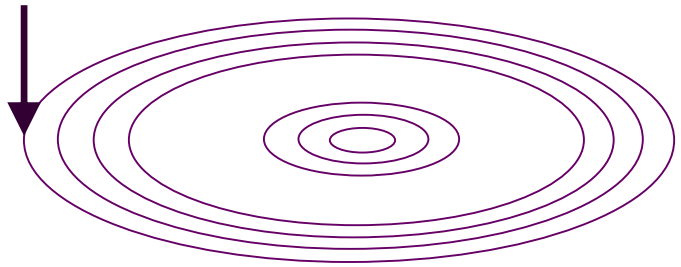
Hyokyung Bahn

EWHA WOMANS UNIVERSITY

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

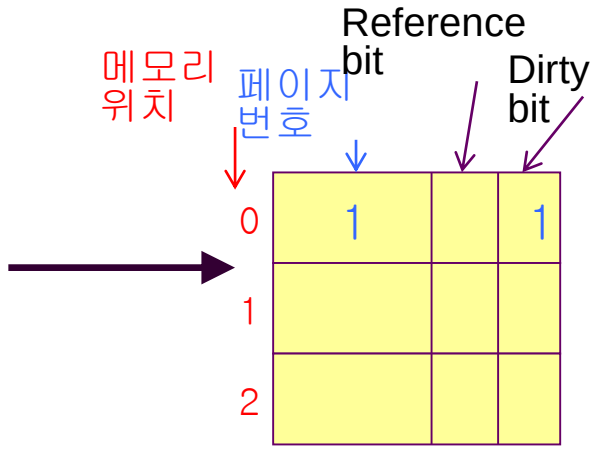
Page fault: 0

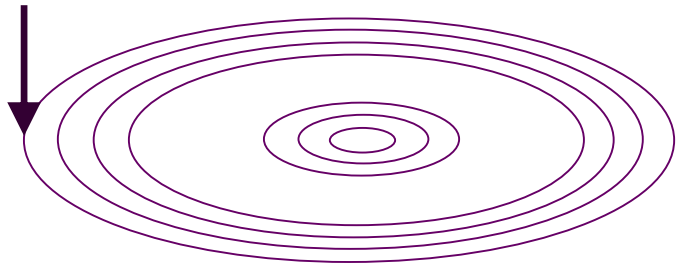
Disk write: 0

Head movement: 0

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

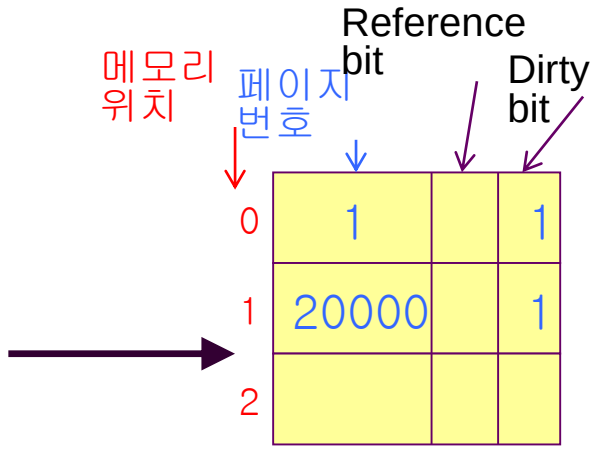
Page fault: 1

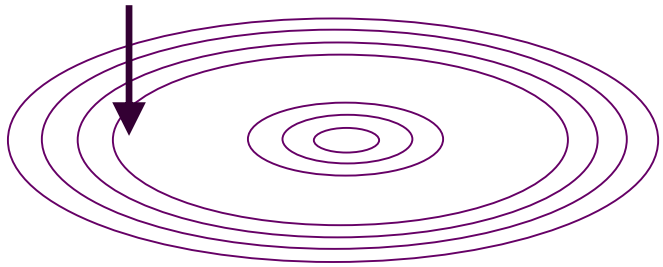
Disk write: 0

Head movement: 0

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 20

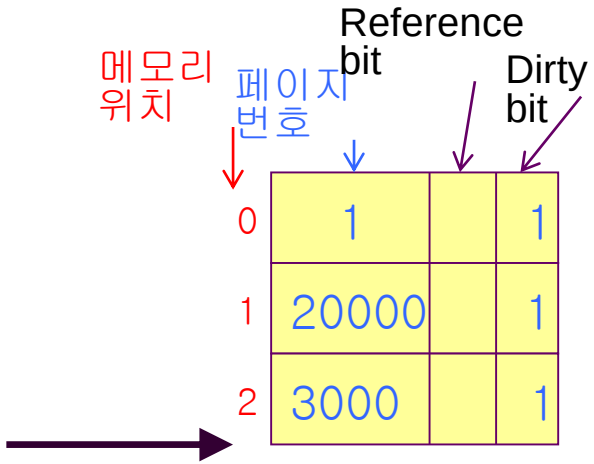
Page fault: 2

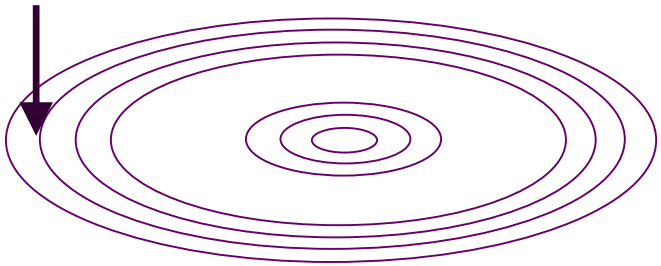
Disk write: 0

Head movement: 20

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 3

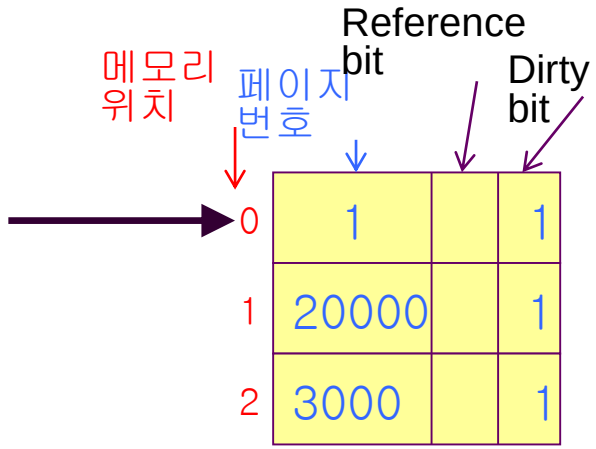
Page fault: 3

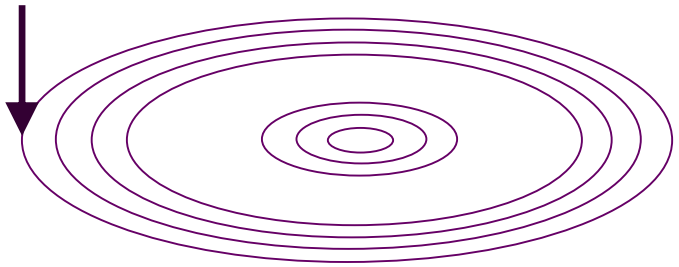
Disk write: 0

Head movement: 37

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

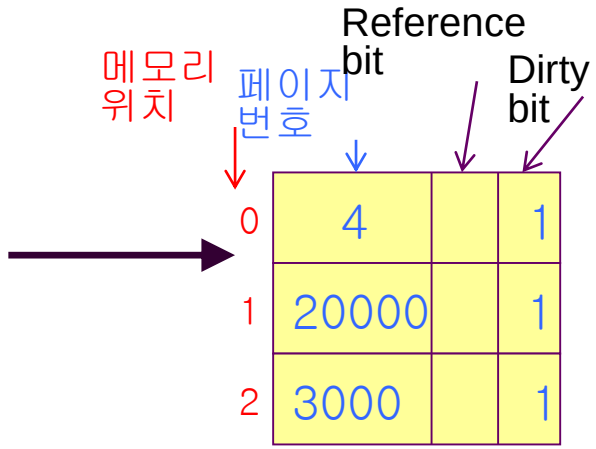
Page fault: 4

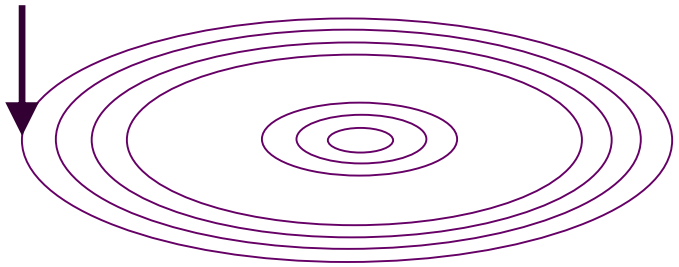
Disk write: 1

Head movement: 40

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

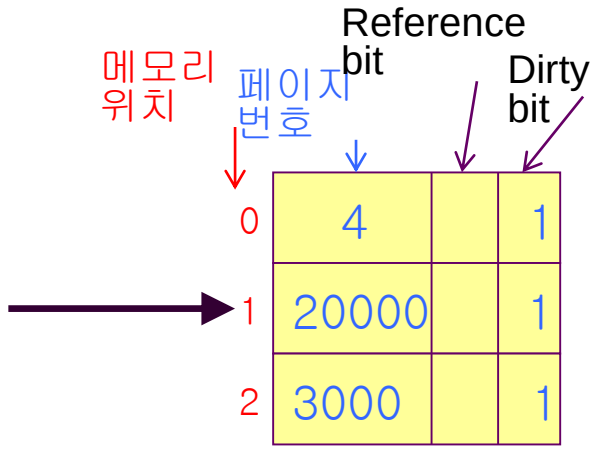
Page fault: 4

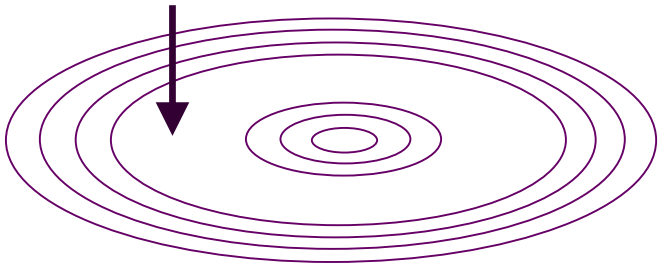
Disk write: 1

Head movement: 40

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 20

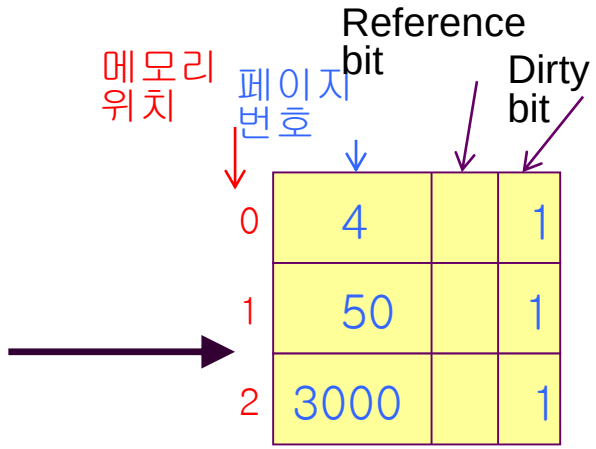
Page fault: 5

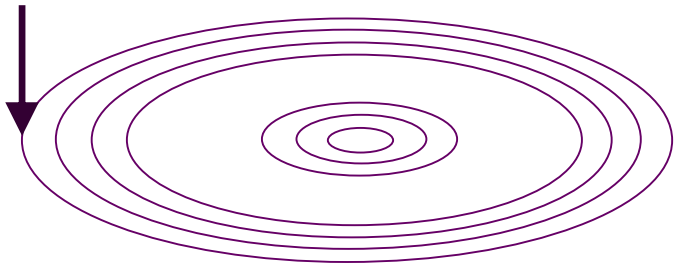
Disk write: 2

Head movement: 60

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

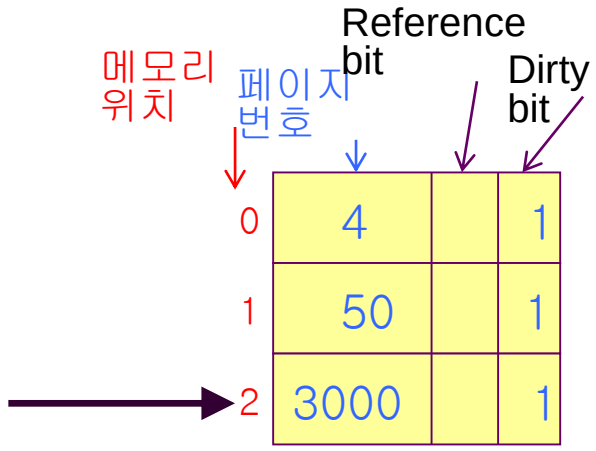
Page fault: 5

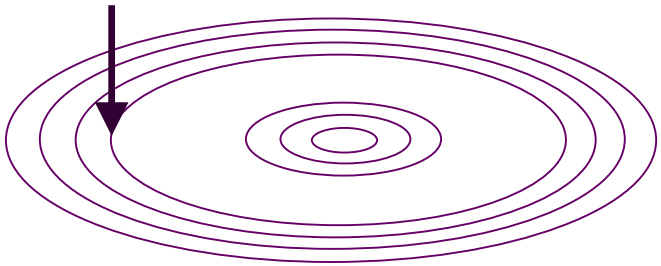
Disk write: 2

Head movement: 80

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 3

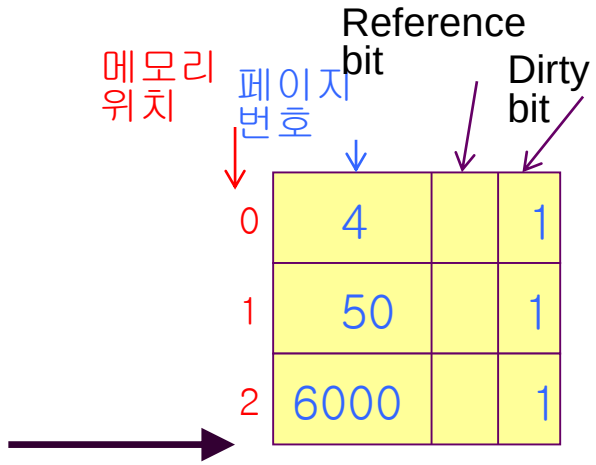
Page fault: 6

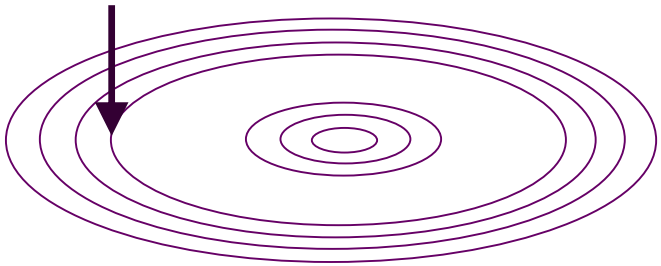
Disk write: 3

Head movement: 83

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
→ 7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 6

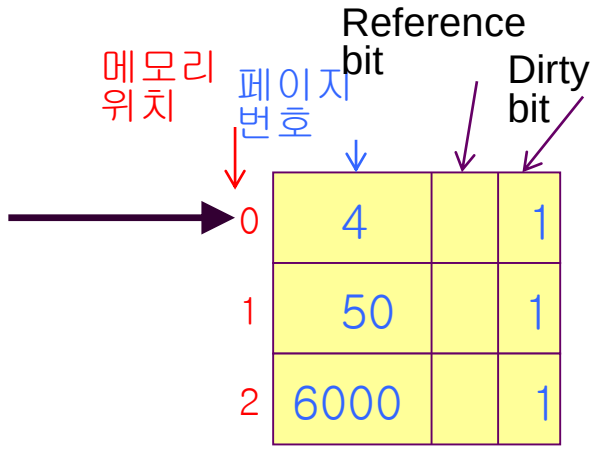
Page fault: 6

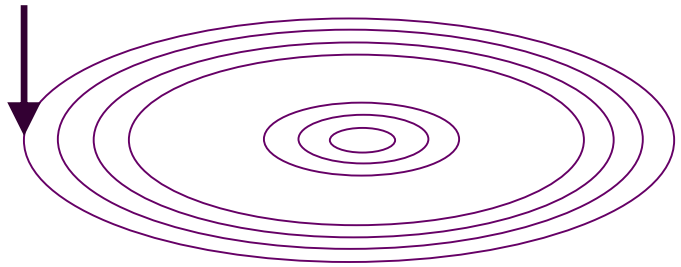
Disk write: 3

Head movement: 86

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

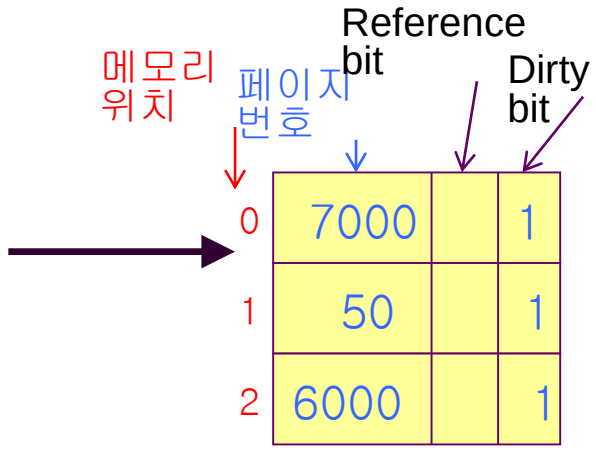
Page fault: 7

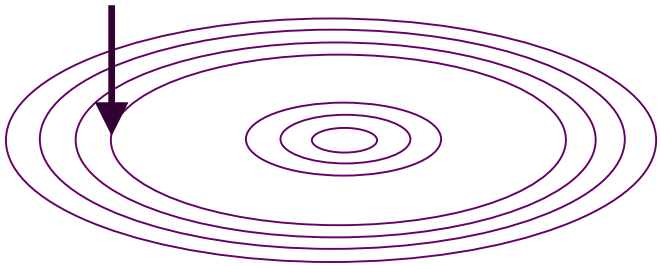
Disk write: 4

Head movement: 92

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
→ 1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 7

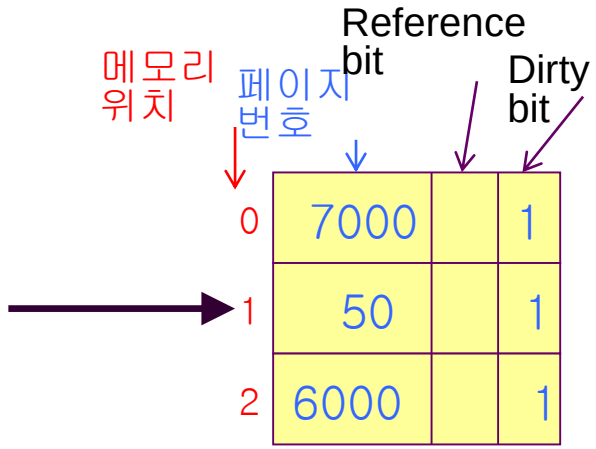
Page fault: 7

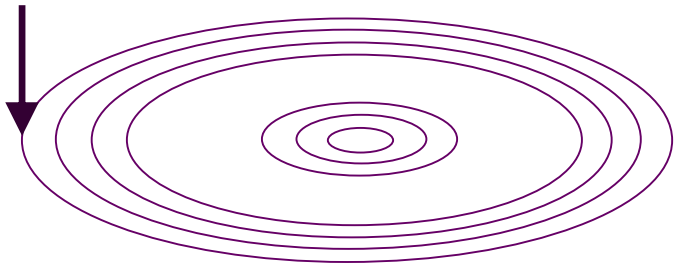
Disk write: 4

Head movement: 99

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

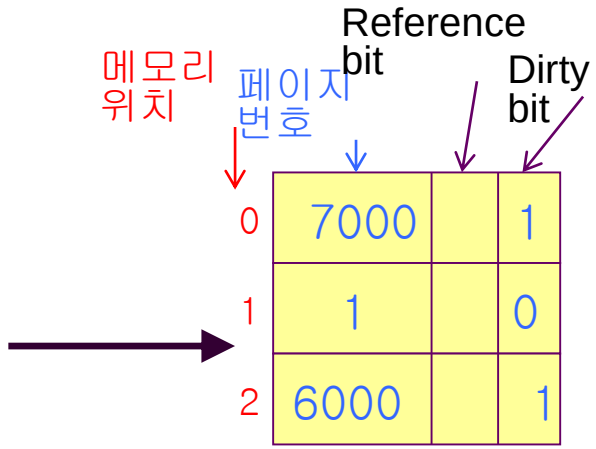
Page fault: 8

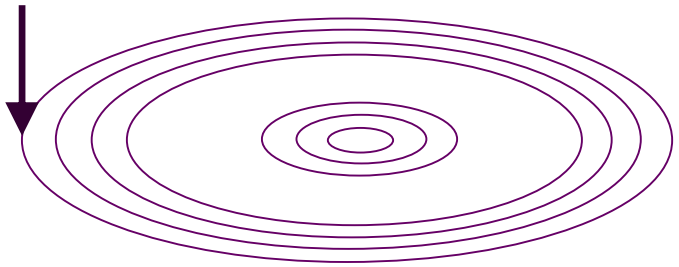
Disk write: 5

Head movement: 106

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

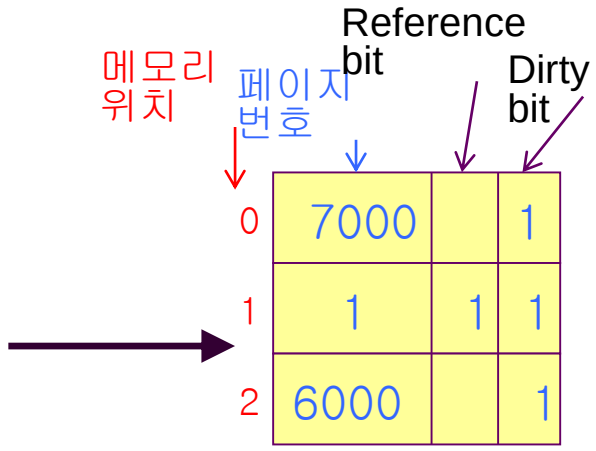
Page fault: 8

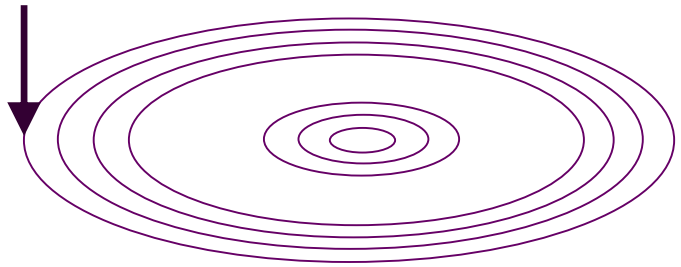
Disk write: 5

Head movement: 106

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
→ 3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

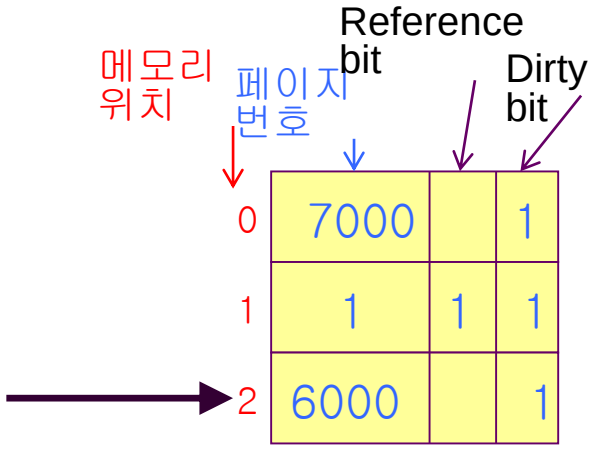
Page fault: 8

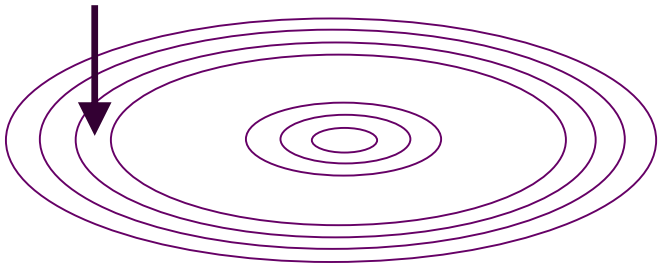
Disk write: 5

Head movement: 106

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
→ 3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 6

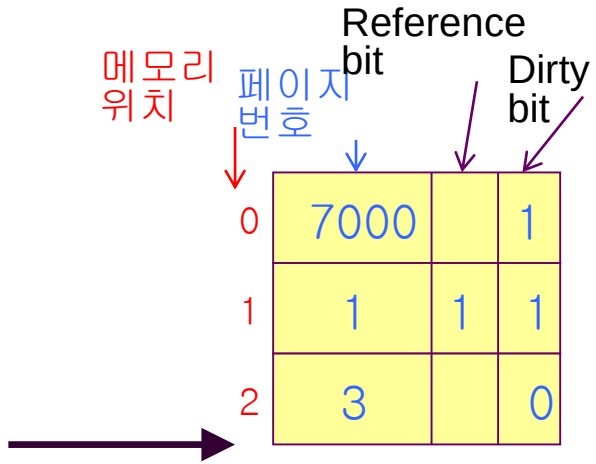
Page fault: 9

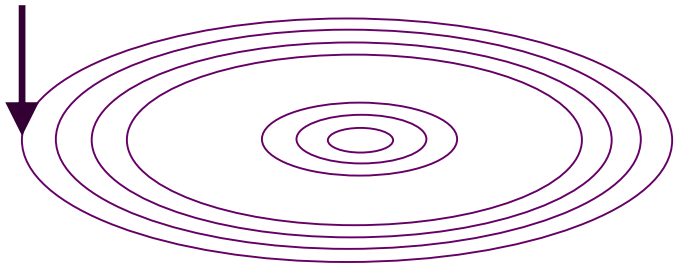
Disk write: 6

Head movement: 112

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
→ 3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

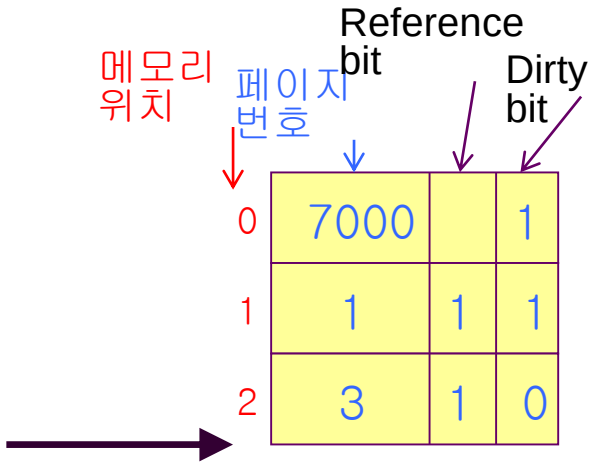
Page fault: 9

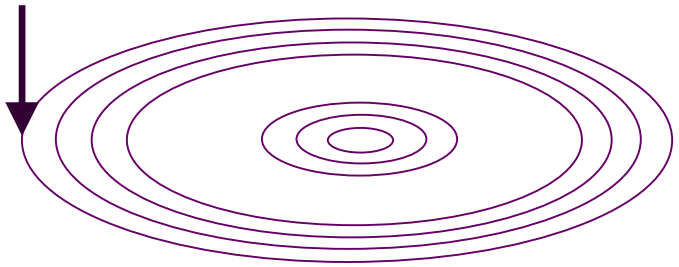
Disk write: 6

Head movement: 118

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

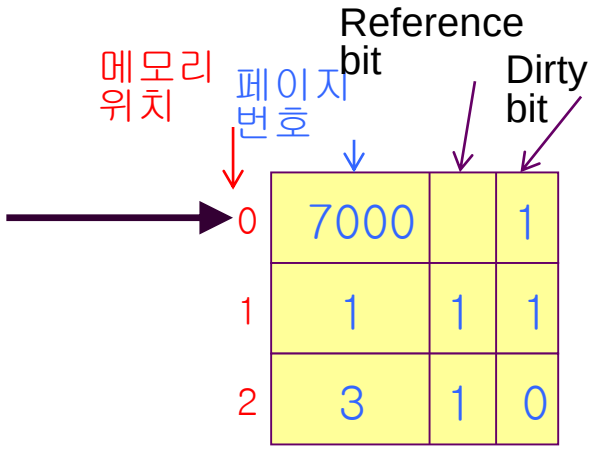
Page fault: 9

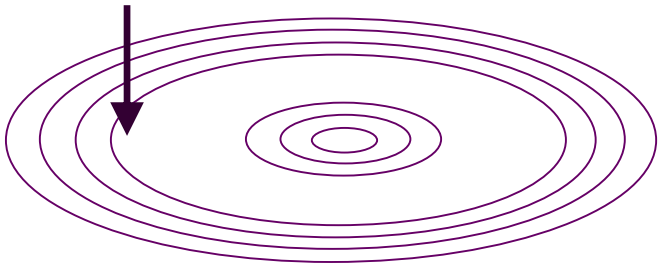
Disk write: 6

Head movement: 118

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
→ 20000	0
20000	1
4	0
1	1





헤드 위치: 7

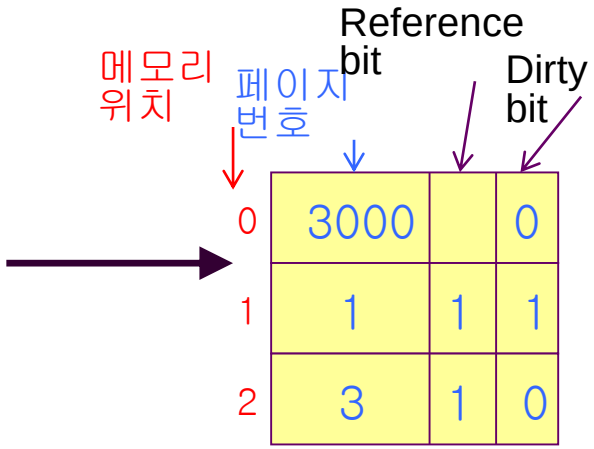
Page fault: 10

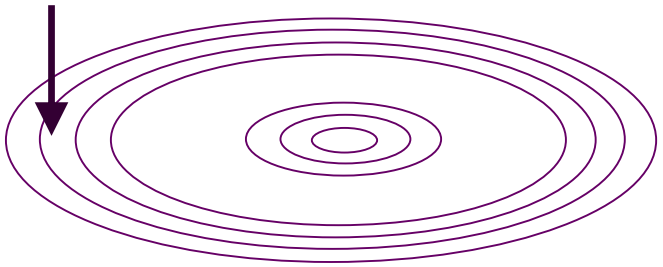
Disk write: 7

Head movement: 125

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 3

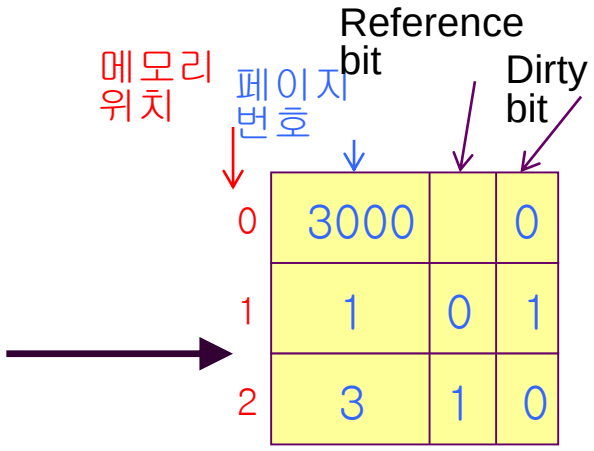
Page fault: 10

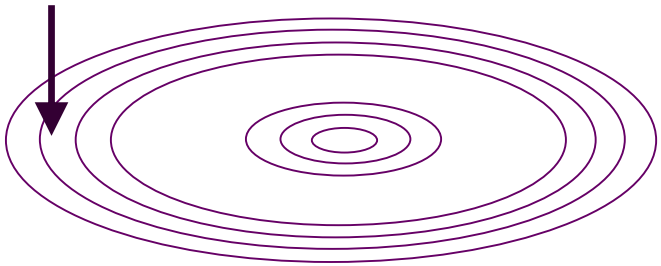
Disk write: 7

Head movement: 129

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
→ 20000	1
4	0
1	1





헤드 위치: 3

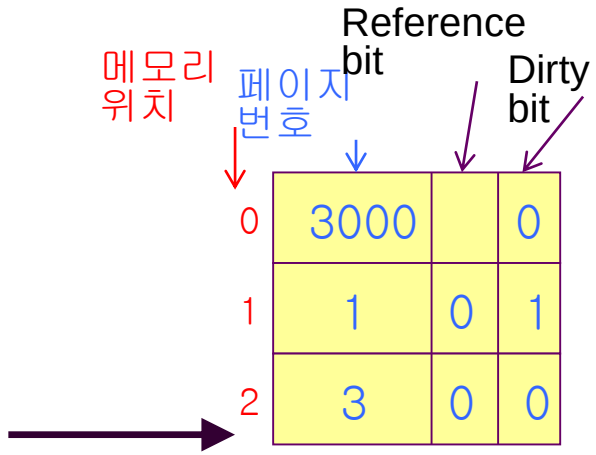
Page fault: 11

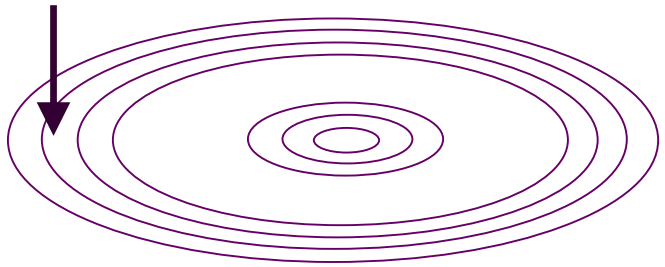
Disk write: 7

Head movement: 129

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
→ 20000	1
4	0
1	1





헤드 위치: 3

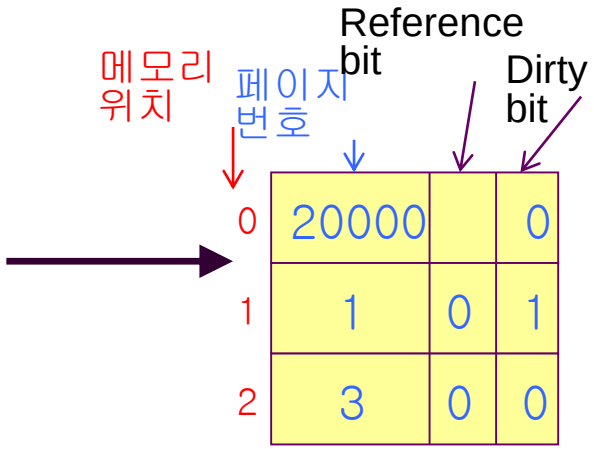
Page fault: 11

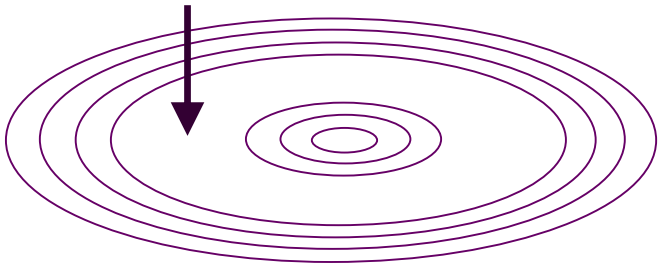
Disk write: 7

Head movement: 129

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 20

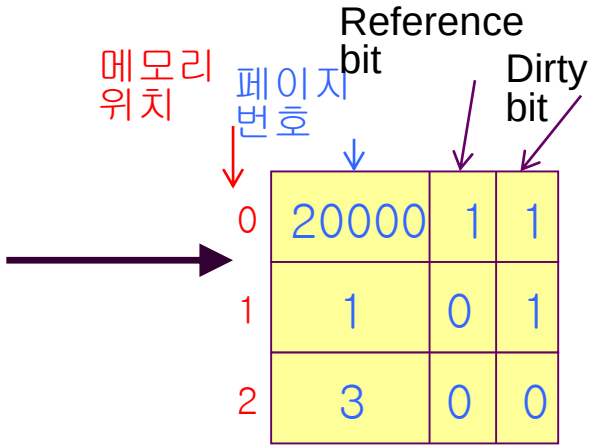
Page fault: 11

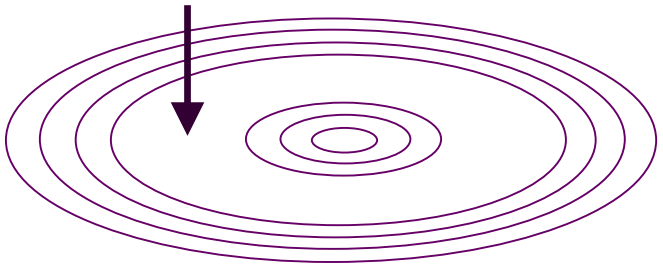
Disk write: 7

Head movement: 146

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 20

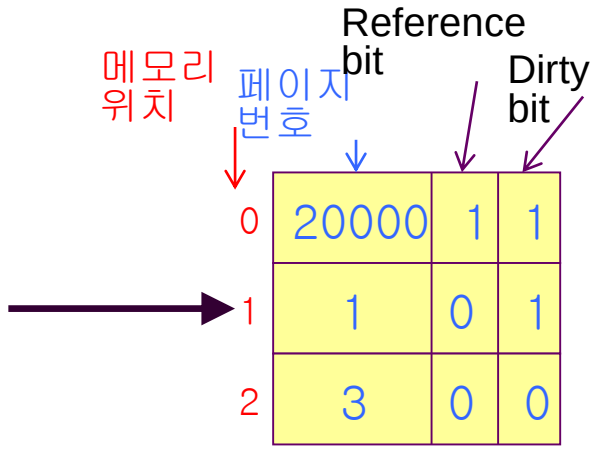
Page fault: 11

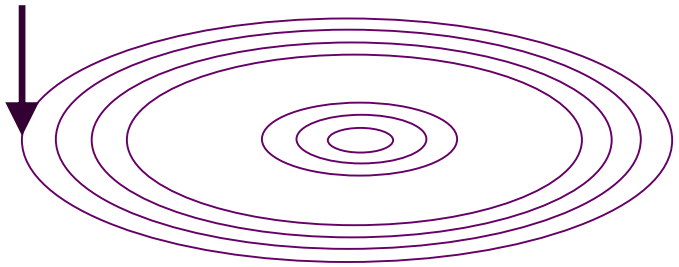
Disk write: 7

Head movement: 146

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
→ 1	1





헤드 위치: 0

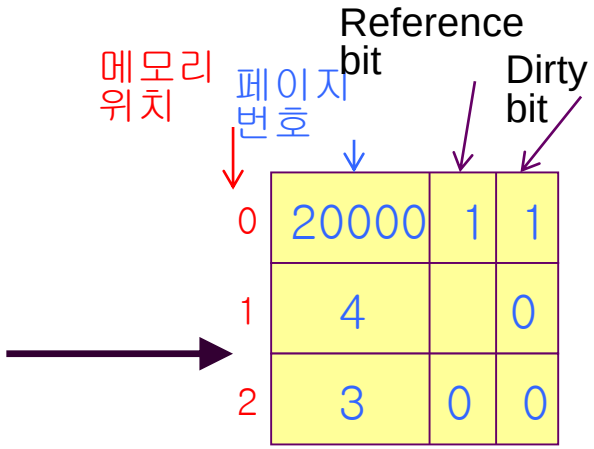
Page fault: 12

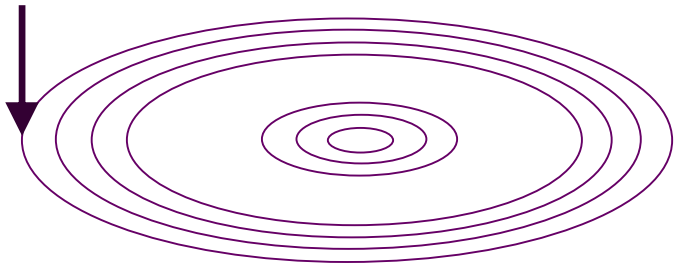
Disk write: 8

Head movement: 166

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
→ 1	1





헤드 위치: 0

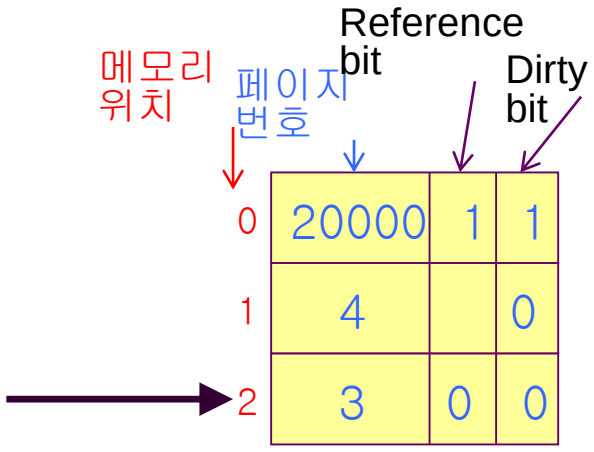
Page fault: 12

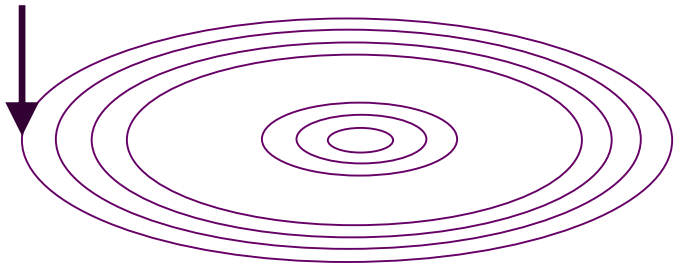
Disk write: 8

Head movement: 166

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

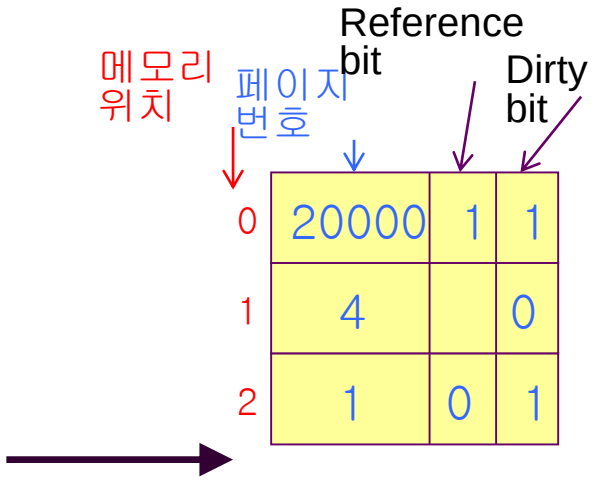
Page fault: 13

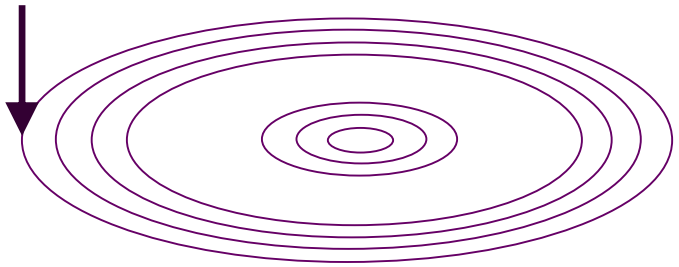
Disk write: 8

Head movement: 166

예제

1	1
20000	1
3000	1
4	1
50	1
6000	1
7000	1
1	0
1	1
3	0
3	0
3000	0
20000	0
20000	1
4	0
1	1





헤드 위치: 0

Page fault: 13

Disk write: 8

Head movement: 166

기본 코드

Hyokyung Bahn

EWHA WOMANS UNIVERSITY

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```

```
for(i = 0; i < memory_size; i++)
 {       memory[i].reference_bit = 0;
         memory[i].dirty_bit = 0;
 }
for(i = 0; i < 200000; i++) page[i].valid_bit = 0;
while(!feof(input_file))
{       fscanf(input_file, "%d %d", &pg, &op);
        if(page[pg].valid_bit == 1)
        {    reference_bit 1로 세팅; write인 경우 dirty_bit 1로 셋팅; }
        else
        {    페이지폴트 횟수 증가;
             빈공간이 없는 경우
             {  클락 시계바늘 이동시키며 reference_bit 1은 0으로, 0인 페이지 탐색;
                쫓겨나는 페이지의 valid_bit 0으로 리셋;
                쫓겨나는 페이지의 dirty_bit이 1이면 디스크 기록을 위해
                디스크 헤드 이동 거리 및 쓰기 횟수 증가;
             }
             요청된 페이지를 디스크에서 읽어 오면서 디스크 헤드 이동 거리 증가;
             클락 바늘이 가리키는 메모리 위치에 새로 들어온 페이지 번호 입력;
             새로 들어온 페이지에 대한 요청이 쓰기 요청인 경우 dirty_bit 세팅;
             새로 들어온 페이지에 대한 reference_bit 0으로 셋팅;
             새로 들어온 페이지의 memory_location에 프레임번호 지정;
             새로 들어온 페이지의 valid_bit 1로 세팅;
        }
}
```