Page Replacement

페이지 결함(Page Fault)

- **페이지 결함은 메인 메모리에 없는 페이지에 접근할 때 발생하는 O/S의 트랩**을 의미한다.
- 페이지 결함이 일어났을 때 해결 절차는 아래와 같다.

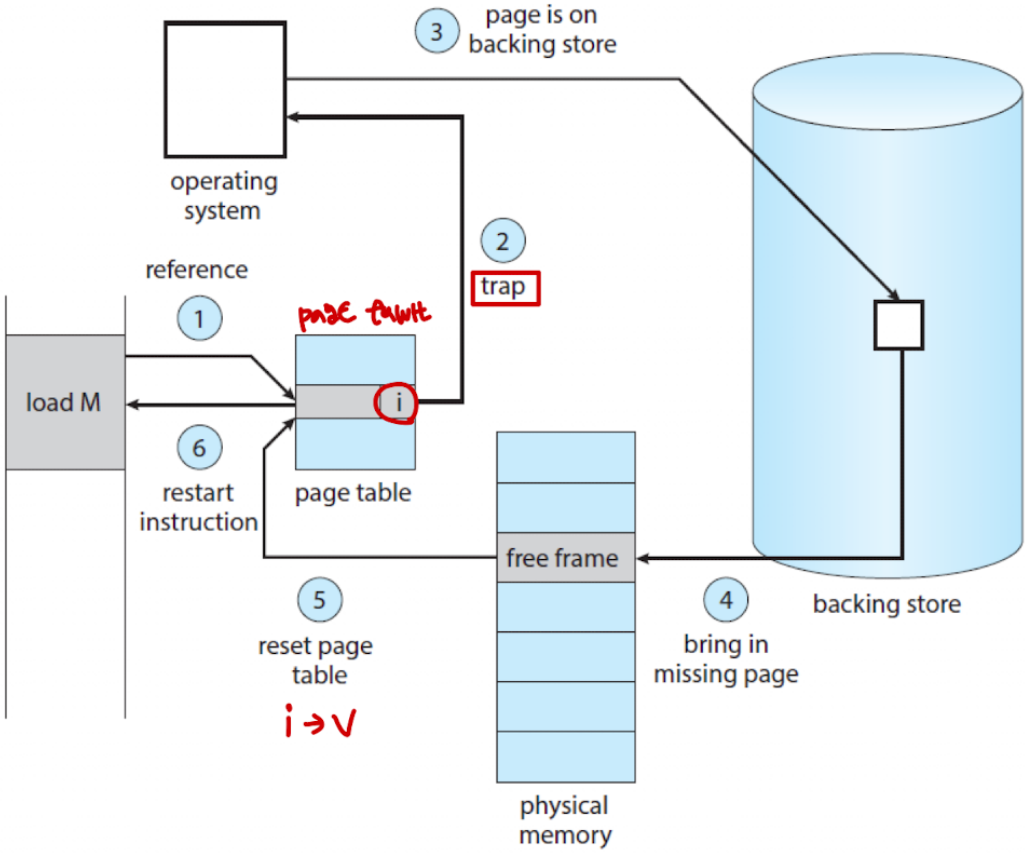


Figure 10.5 *Steps in handling a page fault.*

1. 페이지 테이블을 확인하여 참고 비트가 유효한지, 아닌지 확인한다.
2. 프로세스가 종료됐거나 무효 비트일 경우, 혹은 유효하지만 페이지 결함이 일어나면 페이지 인을 시작한다.
3. 비어있는 프레임이 있는지 탐색한다. `(비어있는 프레임의 목록은 O/S가 관리한다.)`
4. 보조 기억 장치를 스케줄링하여 필요한 페이지를 **새로 할당된 프레임으로 이동**시킨다.

5. 이동 완료시 페이지 테이블을 변경하여 페이지가 메모리에 저장되었음을 표시한다. `(invalid -> valid)`
6. O/S 트랩에 의해 중단된 명령을 재시작한다.

페이지 교체 알고리즘

- **페이지 결함**이 발생하면 **가상 메모리에서 필요한 페이지를 찾아 메인 메모리에 적재**시켜야 하는데, **이 때 메인 메모리의 모든 페이지 프레임이 사용 중이면 어떤 페이지 프레임을 선택해 교체할 것인지 결정**해야 한다.

  ⇒ **이럴 때, 결정에 도움을 주는 기법이 페이지 교체 알고리즘이다.**

페이지 교체 알고리즘과 페이지 결함 처리

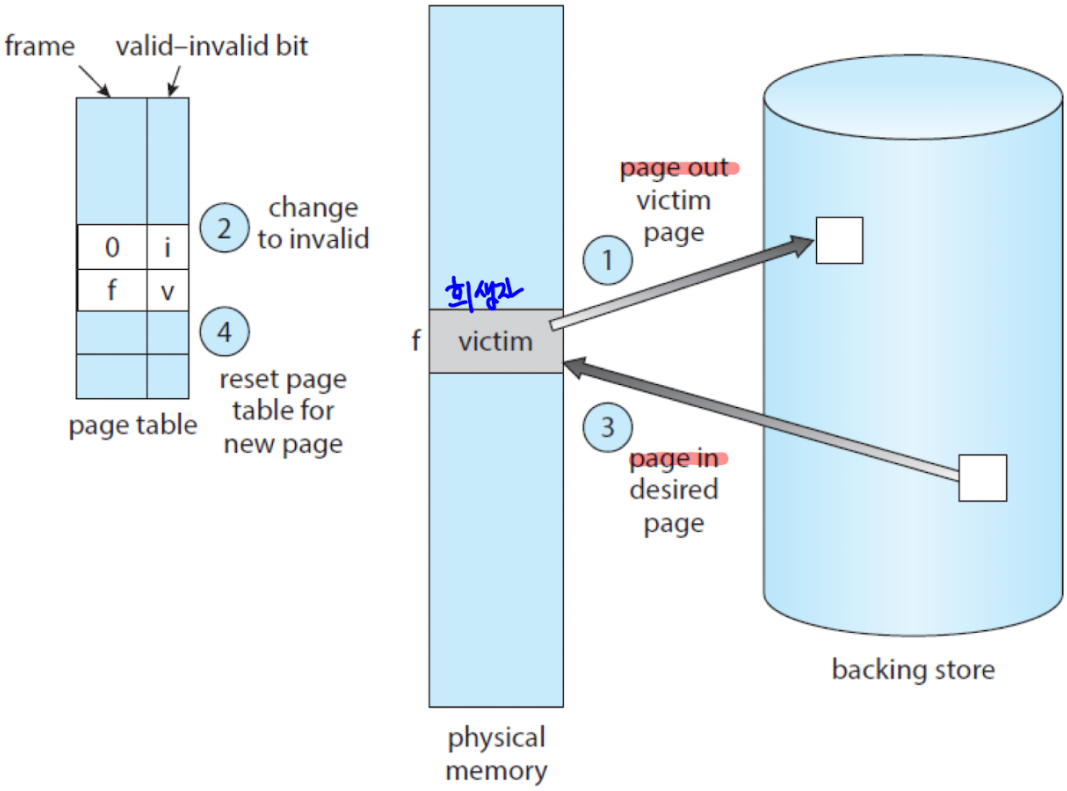


Figure 10.10 *Page replacement.*

- **페이지 교체와 함께 하는 페이지 결함 처리 서비스**는 아래와 같이 진행된다.

1. **보조 기억 장치에서 할당할 페이지의 위치**를 찾는다.
2. 빈 프레임 리스트를 찾는다. →
   2-1. 빈 프레임이 있을 경우 그대로 사용한다. →
   2-2. 빈 프레임이 없으면 페이지 교체 알고리즘을 통해 **희생시킬 프레임을 선정**한다.
3. 희생자로 선정된 **프레임을 보조 기억 장치로 이동시키고, 페이지 테이블의 정보를 변경**한다.
4. 할당할 페이지를 새롭게 희생시키고 남은 빈 프레임에 할당시킨다.
5. 페이지 결함의 나머지 서비스를 진행한다.

페이지 교체 알고리즘의 성능 평가 기준

- 페이지 교체 **알고리즘의 최종 목표는 페이지 결함 발생을 최대한 줄이는 것**이다.
- **프레임의 수가 많아질수록 페이지 결함 역시 줄어든다.**
- 비교를 위해 기준은 아래와 같이 선정한다.
  - 참조 문자열 - `7 - 0 - 1 - 2 - 0 - 3 - 0 - 4 - 2 - 3 - 0 - 3 - 0 - 3 - 2 - 1 - 2 - 0 - 1 - 7 - 0 - 1`
  - 메모리 안에는 3개의 프레임이 들어갈 수 있다.

FIFO 페이지 교체 알고리즘

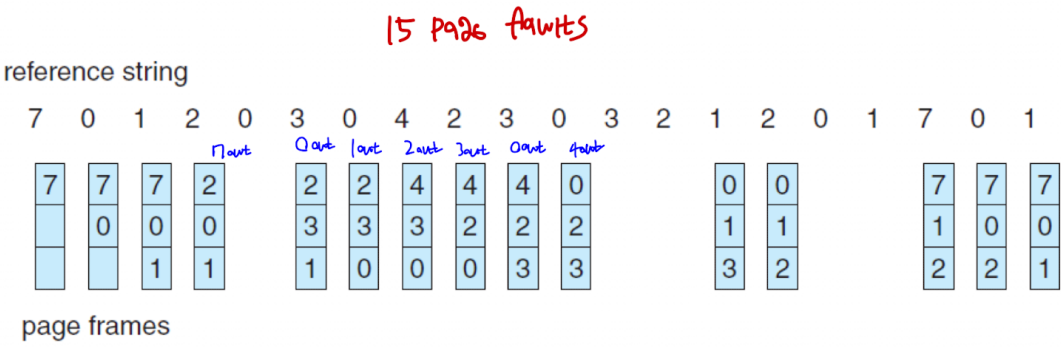
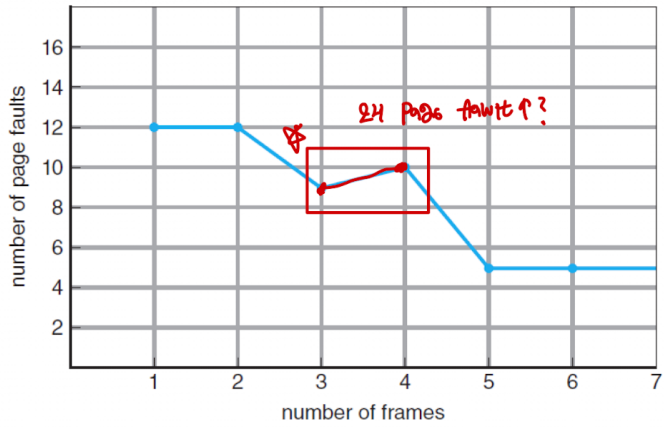


Figure 10.12 *FIFO page-replacement algorithm.*

- **먼저 들어온 프레임을 먼저 희생자로 선정**하는 가장 간단한 알고리즘이다.
- 기준에 맞춰 페이지 교체를 실시하면 **15번의 페이지 결함**이 일어난다.

- **벨라디의 모순**이 발생한다.

벨라디의 모순(Belady's Anomaly)



- **reference string:**
  1 2 3 4 1 2 5 1 2 3 4 5

Figure 10.13 *Page-fault curve of FIFO replacement on a reference string.*

- 원래 프레임의 수가 많아지면 페이지 결함 발생의 수가 적어져야 정상이다.
- 하지만 원칙에 맞지 않게 **프레임의 수가 많아졌음에도 페이지 결함의 발생 수가 늘어나는 현상**을 의미한다.

최적의 페이지 교체 알고리즘 (OPT)

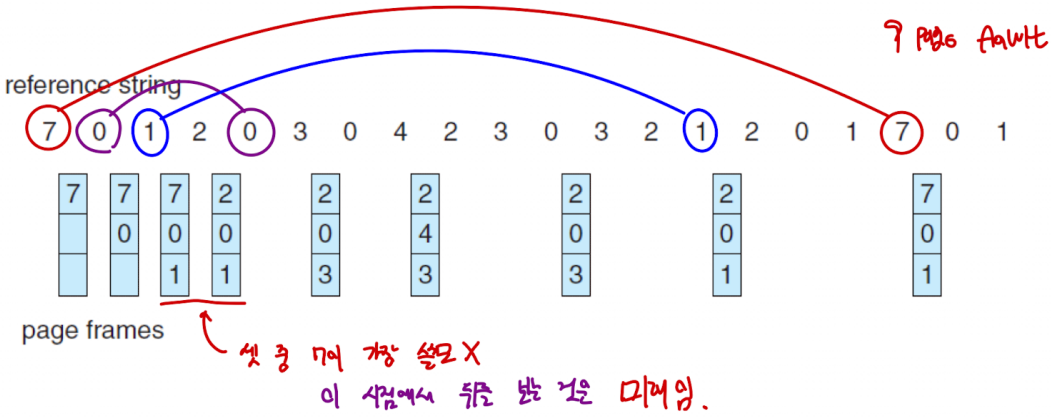


Figure 10.14 *Optimal page-replacement algorithm.*

- **가장 최적의 알고리즘**으로, **앞으로 가장 오랫동안 사용하지 "않을" 프레임을 희생자로 선정**하는 알고리즘이다.

- 페이지 결함이 가장 적게 발생하는 알고리즘이며 벨라디의 모순을 걱정하지 않아도 된다.
- 기준에 맞춰 페이지 교체를 실시하면 9번의 페이지 결함이 일어난다.
- 그러나, **실행되는 시점에서는 미래에 어떤 페이지가 올지 알 수 없기 때문에 실질적으로 적용할 수 없는 알고리즘**이다.

LRU 페이지 교체 알고리즘

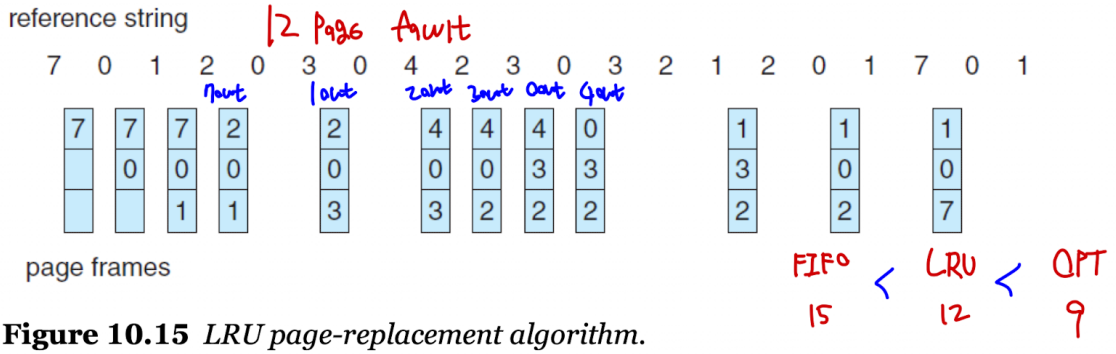


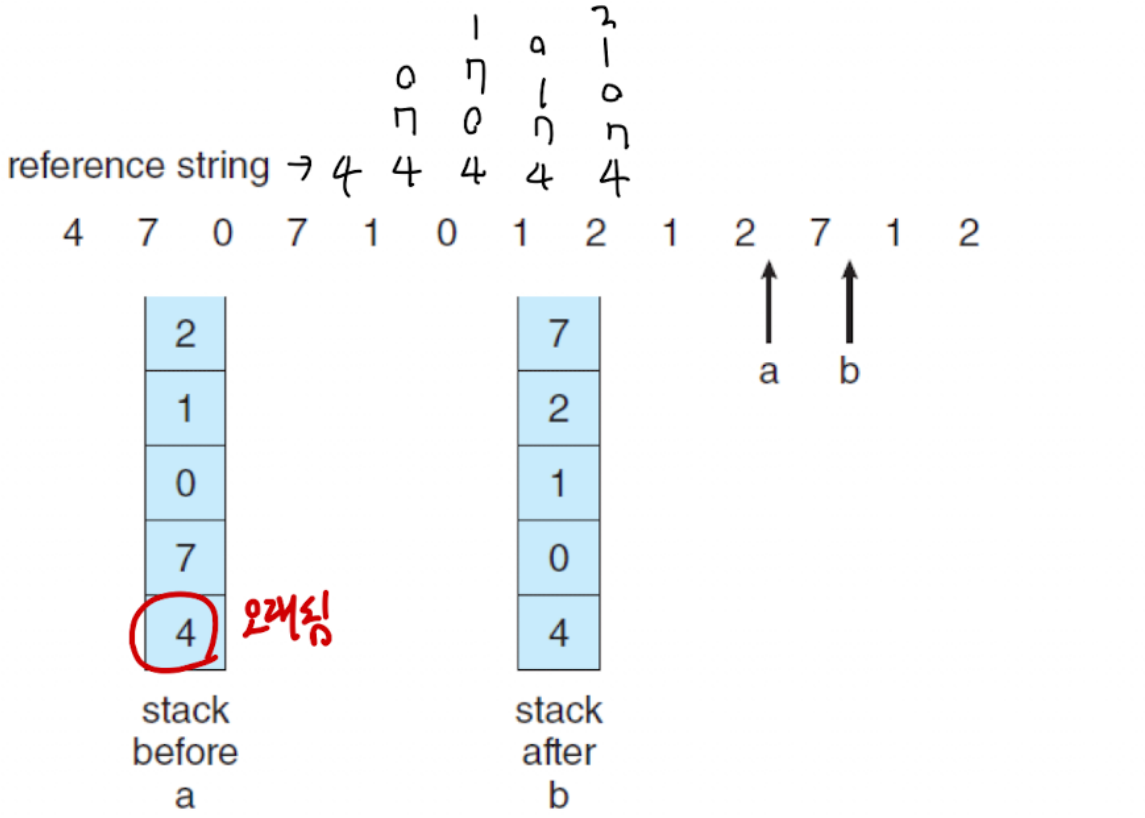
Figure 10.15 *LRU page-replacement algorithm.*

- **LRU : Least Recently Used**
- **가장 오랫동안 사용하지 "않은" 프레임을 희생자로 선정**하는 알고리즘이다.
- 최적 알고리즘 방식과 비슷한 성능을 내기 위해 고안한 알고리즘이다.
- **앞으로 사용할 페이지를 "예측"하여 희생자를 선정하는 방법**으로, **과거에 오랫동안 사용하지 않았다면 미래에도 많이 사용하지 않을 것**이라고 예측하는 것이다.
- 기준에 맞춰 페이지 교체를 실시하면 12번의 페이지 결함이 일어난다.
- **성능이 높아 실제 자주 사용**하고, 벨라디의 모순을 걱정하지 않아도 된다.

LRU 알고리즘 구현을 위한 두 가지 구현 방법

- LRU 알고리즘을 구현하기 위해서는 **프레임이 언제 마지막으로 사용됐는지에 대한 정보를 저장하거나 하드웨어의 지원을 받는** 각각의 방법이 있다.
- **Counter** 를 이용한 구현법
  - **페이지 참조가 발생할 때마다 시간을 체크하는 방법**으로, 페이지 변경이 일어나야 하는 경우 가장 **시간이 오래된 프레임을 희생자로 선정**한다.

- 페이지 변경이 필요할 때 페이지 테이블에서 가장 오래된 것을 찾아야 하고, 페이지 테이블이 변경될 때마다 페이지의 시간도 함께 변경해야 하는 단점이 있다.



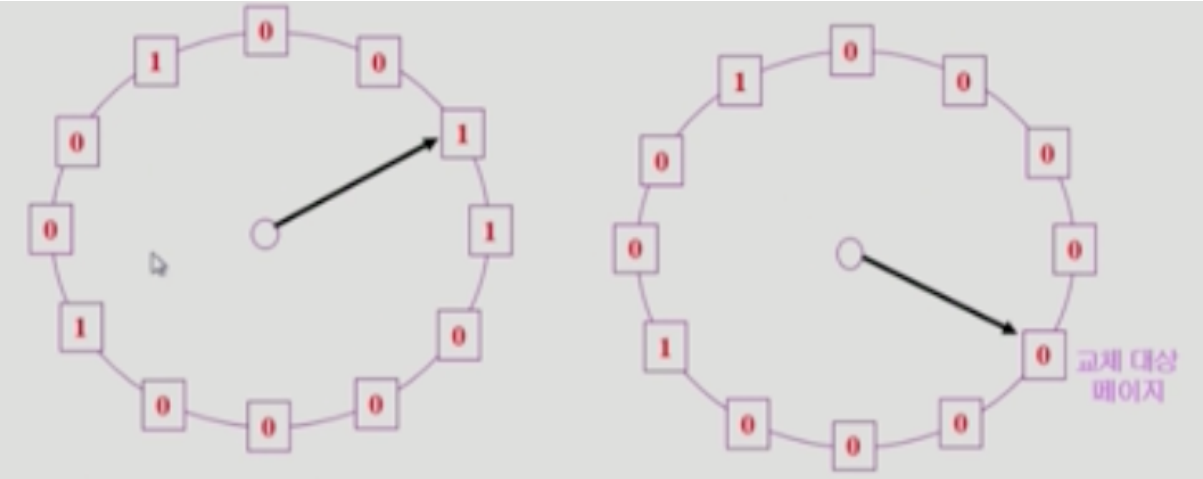
- **Stack** 을 이용한 구현법
  - **페이지 참조가 발생할 때마다 스택에 페이지 넘버를 저장**하는 방법이다.
  - 기존 스택과 차이점은 스택 안에 똑같은 페이지 넘버가 있을 경우 이를 제거하고 다시 맨 위에 페이지 넘버를 등록한다.
  - 결과적으로, **가장 오래된 페이지는 가장 아래에 위치해있기 때문에 희생자로 선정**하면 된다.

LRU-Approximation Page Replacement

- 기본적으로 LRU 알고리즘은 하드웨어의 지원을 받아야만 구현이 가능했으나, **참조 비트(Reference bit)를 사용하면 하드웨어의 지원을 받지 않고 LRU 알고리즘을 구현**할 수 있다.
- **참조 비트**
  - 기본적으로 비트의 값은 0이며, 각 페이지에 할당된다.

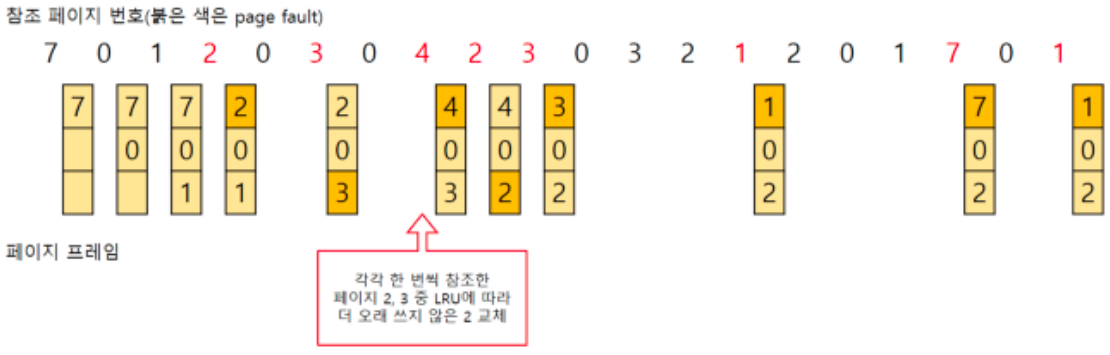
- 페이지가 참조되면 값을 1로 설정한다.
- 비트의 값이 0인 페이지를 희생자로 선정한다.

2차 기회 페이지 알고리즘(Second chance replacement algorithm)



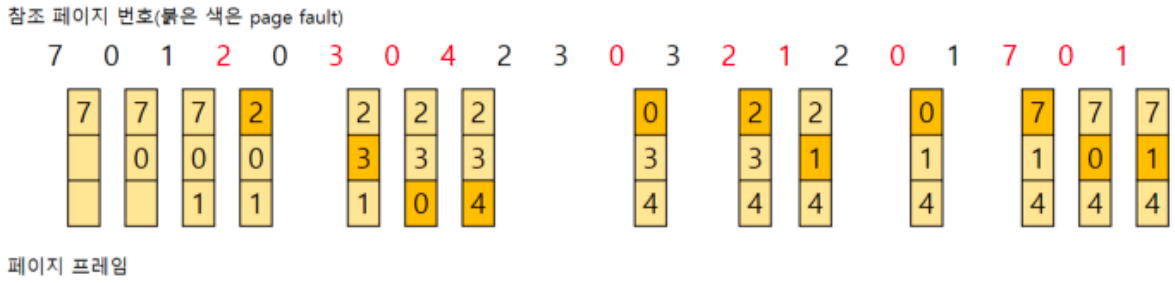
- FIFO 페이지 교체 알고리즘을 보완해서 만든 알고리즘이다.
- **페이지 교체가 필요하면 먼저 들어온 페이지의 참조 비트**부터 검색한다.
  - 만약 값이 0이면, 그대로 교체를 진행한다.
  - 그러나 값이 1이면, 비트의 값을 0으로 설정하고 다음 페이지로 넘어간다.
- **다음번에도 검색했을 때 값이 1이라면, 자주 참조되는 페이지임을 의미하므로, 교체하지 않을 것**이다.
- 마지막으로 모든 비트가 1로 설정된다면 먼저 들어온 것부터 페이지 교체를 진행한다.

LFU 페이지 교체 알고리즘



- **참조횟수가 가장 적은 페이지를 교체하는 알고리즘**이다.
- 교체 대상이 여러 개라면, 가장 오랫동안 사용하지 않은 페이지를 교체한다.

MFU 페이지 교체 알고리즘



- LFU와 반대로, **참조 횟수가 가장 많은 페이지를 교체하는 알고리즘**이다.
