Recency 의 주기 정보 범위는 Frequency 주기 정보 범위의 2 배로 한다.
그 이유는
일정한 주기마다 Frequency 정보를 리셋하는 것처럼
Recency 정보도 리셋할 때에 이전 주기의 정보를 보존하는 것이 더 정확도가 높을 것이다.
Frequency 정보와 Recency 정보를 바탕으로 대략적인 Stack Distance 정보를 얻어낼 것인데
이를 위해서 Stack Distance 정보를 저장하는 위치를 두 개 설정해놓는 것이 나을 것이다.

Frequency 는 기본적으로 Counting Bloom Filter 자료구조를 사용한다.
Recency 는 . . . boolean 정보를 저장하는 비트의 2D-Array 이다.

예를 들어서 Frequency 주기로 0 ~ 15 를 저장할 수 있다면 (NF = 16)
Recency 주기는 0 ~ 31 로 저장한다. (NR = 32, [NR === NF*2])

Recency 는 주기를 여러개의 슬라이스로 나누고 각 슬라이스마다 번호를 부여하고 해당 슬라이스 내에 액세스가 발생할 때마다 몇번 슬라이스였는지를 기록한다.
슬라이스 번호 중에 0~15 에는 과거의 정보가 저장되어 있다. 16~31 에는 최근의 정보가 저장되어 있다.

Recency 정보를 리셋할 때에는

```
if (X – NR/2) < 0
      X = 0
else
      X = X – NR/2
```

와 같이 한다.

단, 새롭게 액세스가 발생할 때마다 이전 슬라이스 번호와 현재 슬라이스 번호의 차이를 바탕으로 Stack Distance(ST 라고 줄여서 부르자) 정보를 업데이트 한다.
예를 들어서
이번에 블록 X 에 대한 액세스가 발생했는데 X 의 마지막 Recency 에 3 이 기록되어 있었고 이번에 5 를 기록할 차례라면
ST = ST + (5 – 3). . . 아니면 ST = ST + (5-3)^2
로 업데이트한다.
이번에 블록 X 에 대한 액세스가 발생했는데 X 의 마지막 Recency 에 8 이 기록되어 있었고 이번에 8 을 기록할 차례라면
ST = ST – K (K 는 일정한 상수)
로 없데이트한다.

그리고 두 개의 ST 변수를 번갈아가면서 사용한다.

이와 같은 방법으로 정보를 처리하면
가장 최근에 액세스한 블록에의 요청이 증가할 경우 Stack Distance 는 감소할 것이다.
오래 전에 액세스한 블록에의 요청이 감소할 경우 Stack Distance 는 증가할 것이다.
ST 정보에 따라서 dynamic workload 에 효율적으로 대응할 수 있게 된다.

아니면 또 다른 방법,
해당 블록에 대한 [Frequency Value * Recency Value] 결과로 가중치를 주는 방법이 있을 수도 있다.

예를 들어서
블록 A 에 대해서 Frequency 가 2 이고 Recency 가 3 이면 → A 의 weight 는 6 이 된다.
블록 B 에 대해서 Freqency 가 2 이고 Recency 가 9 이면 → B 의 weight 는 18 이 된다.
블록 C 에 대해서 Frequency 가 6 이고 Recency 가 3 이면 → C 의 weight 는 18 이 된다.
블록 D 에 대해서 Frequency 가 6 이고 Recency 가 9 이면 → D 의 weight 는 54 가 된다.

weight 가 높을 수록 해당 블록의 가치는 높아진다.