0. 설정 및 환경

- 사용 언어: C
- 사용 프로그램: Visual Studio 2019
- 개발 환경: Windows 10
- TEST한 환경: Windows 10, 인의예지 서버

1. 목표

- Demand paging system을 위한 page replacement 기법 구현 및 검증
- page replacement 기법 (MIN, FIFO, LRU, LFU, CLOCK, WS)

2. 규칙

- page reference string을 입력으로 받아 6가지 page replacement 기법을 수행했을 경우의 memory residence set 변화 과정 및 page fault 발생 과정을 추적 및 출력한다.
- process가 갖는 page 수, page frame 할당량 및 window size, page reference string 길이를 입력으로 받는다.
- 초기 할당된 page frame은 모두 비어 있는 것으로 가정한다.
- 입력 포맷은 N M W K s (N: process가 갖는 page 개수, M: 할당 page frame 개수, W: window size, K: page reference string 길이, s: page reference string)
- N<=100, M<=20, W<=100, K<=100000
- 출력 포맷은 자유롭게 설정한다.

3. 구현

- MIN 기법

: page fault 횟수를 최소화하는 기법으로 가장 나중에 참조될 page를 victim으로 한다.

Tie breaking rule으로는 greatest page number을 우선적으로 victim으로 한다.

이 MIN 기법은 실제는 사용이 불가능한 기법으로 다른 기법들의 성능을 측정하는 기준이 된다.

```
reset_table(); // table 초기화
int tf[100] = { 0, }; // index= == page number, page가 memory에 있으면 1, 없으면 0
int whereisNext[100]; // 다음으로 참조될 타이밍
int where[100]; // index == page number, page가 존재하는 page frame number
for (i = 0;i < 100;i++) {
    whereisNext[i] = -1;
    where[i] = -1;
} // 초기화

int in = 0; // 초기 page frame 채우는 용도
int count = 0; // page fault 횟수
```

```
for (i = 0;i < stringLength;i++) {
    int temp_in = arr[i]; // 대상 page number
    // whereisNext 계산
    ////////////////////////////////////////
    int j = 1;
    while (i + j < stringLength && arr[i + j] != temp_in) {
        j++;
    }
    int next;
    if (i + j == stringLength) next = 2147483647;
    else next = i + j;
    whereisNext[temp_in] = next;
    ////////////////////////////////////////
```

MIN 기법은 page frame에 존재하는 page중에서 가장 나중에 참조될 page를 victim으로 하기 때문에 언제 다시 참조될 지를 알아야 한다. 이 과정에서 실제의 경우 구현 불가능한 이유가 된다.

```
if (tf[temp_in] == 1) continue; // 이미 memory에 있다면 continue
else { // page fault 발생 부분
    count++;
    if (in < pageframeN) { // 초기 page frame 채우기
        table[in] = temp_in;
        where[temp_in] = in;
        tf[temp_in] = 1;
        in++;
        printf("Page Fault at %d / IN: %d, OUT: -1₩n", i, temp_in);
    }
```

tf배열을 통해 대상으로 하는 page가 page frame내에 존재하는지 확인한다.

```
else { // victim 발생
    int out = 0;
    int target = -1;
    for (j = 0;j < 20;j++) { // page frame내의 page중 가장 나중에 참조될 page를 victim으로 함
        int temp = table[j];
        if (temp == -1) continue;
        if (target <= whereisNext[temp]) {
            target = whereisNext[temp];
            out = where[temp];
        }
    }

    // 위에서 고른 victim을 table에서 제거하고 대상으로 한 page로 교체
    int temp_out = table[out];
    if (temp_out != -1) {
        tf[temp_out] = 0;
        where[temp_out] = -1;
        whereisNext[temp_out] = -1;
    }

    table[out] = temp_in;
    tf[temp_in] = 1;
    where[temp_in] = out;
    printf("Page Fault at %d / IN: %d, OUT: %d₩n", i, temp_in, temp_out);
}
```

page frame이 가득 차 victim이 발생하는 경우로 page frame내의 page들의 whereisNext 배열을 통해 가장 나중에 참조 될 page를 찾아서 교체해준다.

- FIFO 기법

: page frame에 있는 page중 가장 먼저 들어온 page를 victim으로 한다.

```
int FIFO(){
    reset_table(); // table 초기화
    int tf[100] = { 0, }; // index= == page number, page가 memory에 있으면 1, 없으면 0
    int out = 0; // victim이 될 index
    int count = 0;
    for (i = 0;i < stringLength;i++) {
        int temp_in = arr[i];
        if (tf[temp_in] == 1) continue;
        else {
            count++;
            int temp_out = table[out];
            if (temp_out != -1) tf[temp_out] = 0;
            table[out] = temp_in;
            tf[temp_in] = 1;
            printf("Page Fault at %d / IN: %d, OUT: %d₩n", i, temp_in, temp_out);
            out = (out + 1) % pageframeN; // page fault가 발생하면 out이 1씩 늘어남
        }
    }
    printf("FIFO > page fault count: %d₩n₩n", count);
    return count;
}
```

Out 변수를 page fault가 발생했을 때, victim이 될 index 번호로 한다.

tf배열을 통해 대상으로 하는 page가 page frame내에 존재하는지 확인하고 out 변수를 통해 page를 교체해준다.

Out 변수는 0부터 차례대로 1씩 증가하며 page frame의 마지막 다음은 0이 된다.

- LRU 기법

: page frame에 있는 page중 가장 오래전에 참조된 page를 victim으로 한다.

```
for (i = 0;i < stringLength;i++) {
    int target = arr[i];
    int index = where[target];
    if (index != -1) { // 이미 table에 page 존재
        int t; // recently used 처리
        for (j = 0;j < in;j++) {
            if (which[j] == index) {
                t = j;
                break;
            }
        }
        swapping(which, t, in, index); // 맨 뒤로 보냄
    }
```

```
void swapping(int* arr, int start, int end, int last) {
    int k;
    for (k = start; k < end - 1;k++) {
        swap(&arr[k], &arr[k + 1]);
    }
    arr[end - 1] = last;
}
```

Which 배열을 두어 page 번호를 집어넣고 가장 나중에 참조된 순서로 정렬되도록 한다. Index 0 : 가장 오래된 page로 victim의 대상이 된다.

이미 page frame내에 존재하는 page를 참조하려는 경우, 해당 page를 swapping을 통해 which배열에서 제일 뒤로 보낸다.

```
else {
    count++;
    if (in < pageframeN) {
        which[in] = in;
        table[in] = target;
        where[target] = in;
        in++;
        printf("Page Fault at %d / IN: %d, OUT: -1₩n", i, target);
    }
    else {
        int out = which[0]; // which[0]이 참조된지 가장 오래된 page number
        swapping(which, 0, pageframeN, out);
        printf("Page Fault at %d / IN: %d, OUT: %d₩n", i, target, table[out]);
        where[table[out]] = -1;
        where[target] = out;
        table[out] = target;
    }
}
```

Page frame이 가득 찬 상태에서 page 교체가 이루어진 경우, which[0]를 victim 번호로 하고, which 배열에서 없앤다.

- LFU 기법

: page frame에 있는 page중 참조 횟수가 가장 적은 page를 victim으로 한다.

```
int find_MIN(int *cnt, int* howlong) {
    int k;
    int r;
    int t = 2147483647;
    for (k = 0;k < pageframeN;k++) { // 가장 적게 참조된 page 찾기
        int N = table[k];
        if (t > cnt[N]) {
            t = cnt[N];
            r = k;
        }
        else if (t == cnt[N]) { // tie-breaking rule
            if (howlong[table[r]] < howlong[table[k]]) { // lru
                r = k;
            }
        }
    }

    return r;
}
```

Page frame내의 page중에서 참조된 횟수가 가장 적은 페이지를 찾아서 반환한다. 그 페이지가 두개 이상일 경우에는 tie breaking rule으로 howlong 배열을 통해 page frame에 들어온지 더 오래된 page를 찾아 반환한다.

```
reset_table();
int cnt_page[100] = { 0, };
int where[100];
int howlong[100] = { 0, }; // tie-breaking 때 사용할 용도, LRU
for (i = 0;i < 100;i++) where[i] = -1;

int in = 0;
int out = 0;
int count = 0;
for (i = 0;i < stringLength;i++) {
    // page frame에 들어있는 page들이 얼마나 오래됐는지를 나타냄
    for (j = 0;j < in;j++) howlong[table[j]]++;
    int target = arr[i];
    int index = where[target];
    if (index != -1) { // 이미 table에 있는 경우
        cnt_page[target]++; // 참조된 횟수 증가
        howlong[target] = 0; // recently referenced 처리
    }
```

howlong 배열은 index가 page number에 해당하고 해당 page가 page frame에 들어온지 얼마나 오래되었는지를 나타낸다.

이미 page frame내에 있는 page를 참조한 경우 이 값을 0으로 다시 초기화한다.

cnt_page 배열은 index가 page number로 해당 page가 참조된 횟수를 나타낸다.

```
else {
    count++;
    if (in < pageframeN) {
        cnt_page[target]++;
        table[in] = target;
        where[target] = in;
        in++;
        printf("Page Fault at %d / IN: %d, OUT: -1₩n", i, target);
    }
    else {
        int out = find_MIN(cnt_page, howlong); // 가장 적게 참조된 page를 victim으로 함
        printf("Page Fault at %d / IN: %d, OUT: %d₩n", i, target, table[out]);
        howlong[table[out]] = 0;
        //cnt_page[table[out]] = 0;
        //cnt_page[target] = 1;
        cnt_page[target]++;
        where[table[out]] = -1;
        where[target] = out;
        table[out] = target;
    }
}
```

find_min함수를 통해 victim이 될 page를 고르고 page를 교체한다.

이 때, 제거 된 page의 참조 횟수는 0으로 초기화 되지 않는다.

- CLOCK 기법

: reference bit을 두어 page frame에 있는 page를 차례대로 돌면서 reference bit이 0인 page를 victim으로 한다.

```
for (i = 0;i < stringLength;i++) {
    int temp_in = arr[i];
    if (tf[temp_in] == 1) {
        int w = where[temp_in];
        zero0Rone[w] = 1;
    }
    else {
        count++;
        while (zero0Rone[out] == 1) { // reference bit이 0인 page를 찾을 때까지
            zero0Rone[out] = 0; // reference bit이 1이면 0으로 바꿈
            out = (out + 1) % pageframeN;
        }
        int temp_out = table[out];
        if (temp_out != -1) {
            tf[temp_out] = 0;
            where[temp_out] = -1;
        }

        table[out] = temp_in;
        zero0Rone[out] = 1;
        tf[temp_in] = 1;
        where[temp_in] = out;
        printf("Page Fault at %d / IN: %d, OUT: %d₩n", i, temp_in, temp_out);
        out = (out + 1) % pageframeN;
    }
}
```

clock 기법은 fifo 기법과 비슷하게 구현하되, reference bit이 추가되었다.

Out은 차례대로 1씩 늘어나고 page frame의 마지막이 되면 0으로 돌아온다.

Page가 page frame에 들어오면 reference bit은 1로 하고 out을 1 늘린다. page frame이 가득 차 page 교체가 발생하면 out을 1씩 늘리면서 reference bit이 0인 page를 찾는다.

이 때, out이 증가하면서 확인 한 reference bit가 1인 경우, 이 값을 0으로 바꾸면서 넘어간다.

- WS 기법

: 위 기법들과는 다르게 page frame의 개수가 제한되지 않고 window size내의 page는 working set으로 page frame에 들어가고 window size를 벗어난 page는 자동으로 page frame에서 지워진다.

```
for (i = 0;i < stringLength;i++) {
    int temp_in = arr[i];

    if (tf[temp_in] == 0) { // page frame에 없으면
        count++;
        printf("Page Fault at %d / IN: %d₩n", i, temp_in);
    }
    tf[temp_in]++;

    if (i - windowSize > 0) { // window size를 벗어난 page
        int t = i - windowSize - 1;
        tf[arr[t]]--;
        if (tf[arr[t]] == 0) printf("At %d, OUT %d from page frame₩n", i, arr[t]);
    }

}
```

Ws 기법에서 tf 배열은 위와 다르게 1보다 큰 값을 가질 수 있다. Window size내의 page들의 개수를 나타낸다. Window size를 벗어나면 1씩 감소시키는데 이 때 0이라면 page frame에서 out된 경우이고, 1이상의 값을 가진다면 아직 page frame내에 존재한다는 의미이다.

4. 테스트

(-1 의 의미는 아무것도 없음을 나타낸다.)

(page fault가 발생한 경우에만 출력이 되도록 하였다.)

(printFLAG를 두어 요약결과만 볼지, 상세 과정까지 볼지 결정할 수 있도록 하였다.)

1) 과제 교안에 있는

6 3 3 15

0 1 2 3 2 3 4 5 4 1 3 4 3 4 5 으로 테스트 해 보았다.

1. MIN 기법

TIME	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
STRING	0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
MEMORY STATE	0	0	0	3	3	3	3	5	5	5	5	5	5	5	5
		1	1	1	1	1	1	1	1	1	3	3	3	3	3
			2	2	2	2	4	4	4	4	4	4	4	4	4
PAGE FAULT	F	F	F	F			F	F			F				

```
Page Fault at 0, IN: 0, OUT: -1 / Memory residence set: 0 -1 -1
Page Fault at 1, IN: 1, OUT: -1 / Memory residence set: 0 1 -1
Page Fault at 2, IN: 2, OUT: -1 / Memory residence set: 0 1 2
Page Fault at 3, IN: 3, OUT: 0 / Memory residence set: 3 1 2
Page Fault at 6, IN: 4, OUT: 2 / Memory residence set: 3 1 4
Page Fault at 7, IN: 5, OUT: 3 / Memory residence set: 5 1 4
Page Fault at 10, IN: 3, OUT: 1 / Memory residence set: 5 3 4
MIN > page fault count: 7
```

위 표는 reference string에 대한 시뮬레이션 결과이고, 아래 사진은 작성한 코드를 실행했을 때 나오는 결과이다. 동일한 결과임을 확인할 수 있다.

2. FIFO 기법

TIME	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
STRING	0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
MEMORY STATE	0	0	0	3	3	3	3	3	3	1	1	1	1	1	5
		1	1	1	1	1	4	4	4	4	3	3	3	3	3
			2	2	2	2	2	5	5	5	5	4	4	4	4
PAGE FAULT	F	F	F	F			F	F		F	F	F			F

```
Page Fault at 0, IN: 0, OUT: -1 / Memory residence set: 0 -1 -1
Page Fault at 1, IN: 1, OUT: -1 / Memory residence set: 0 1 -1
Page Fault at 2, IN: 2, OUT: -1 / Memory residence set: 0 1 2
Page Fault at 3, IN: 3, OUT: 0 / Memory residence set: 3 1 2
Page Fault at 6, IN: 4, OUT: 1 / Memory residence set: 3 4 2
Page Fault at 7, IN: 5, OUT: 2 / Memory residence set: 3 4 5
Page Fault at 9, IN: 1, OUT: 3 / Memory residence set: 1 4 5
Page Fault at 10, IN: 3, OUT: 4 / Memory residence set: 1 3 5
Page Fault at 11, IN: 4, OUT: 5 / Memory residence set: 1 3 4
Page Fault at 14, IN: 5, OUT: 1 / Memory residence set: 5 3 4
FIFO > page fault count: 10
```

위 표는 reference string에 대한 시뮬레이션 결과이고, 아래 사진은 작성한 코드를 실행했을 때 나오는 결과이다. 동일한 결과임을 확인할 수 있다.

3. LRU 기법

TIME	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
STRING	0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
MEMORY STATE	0	0	0	3	3	3	3	3	3	1	1	1	1	1	5
		1	1	1	1	1	4	4	4	4	4	4	4	4	4
			2	2	2	2	2	5	5	5	3	3	3	3	3
PAGE FAULT	F	F	F	F			F	F		F	F				F

```
Page Fault at 0, IN: 0, OUT: -1 / Memory residence set: 0 -1 -1
Page Fault at 1, IN: 1, OUT: -1 / Memory residence set: 0 1 -1
Page Fault at 2, IN: 2, OUT: -1 / Memory residence set: 0 1 2
Page Fault at 3, IN: 3, OUT: 0 / Memory residence set: 3 1 2
Page Fault at 6, IN: 4, OUT: 1 / Memory residence set: 3 4 2
Page Fault at 7, IN: 5, OUT: 2 / Memory residence set: 3 4 5
Page Fault at 9, IN: 1, OUT: 3 / Memory residence set: 1 4 5
Page Fault at 10, IN: 3, OUT: 5 / Memory residence set: 1 4 3
Page Fault at 14, IN: 5, OUT: 1 / Memory residence set: 5 4 3
LRU > page fault count: 9
```

위 표는 reference string에 대한 시뮬레이션 결과이고, 아래 사진은 작성한 코드를 실행했을 때 나오는 결과이다. 동일한 결과임을 확인할 수 있다.

4. LFU 기법

TIME	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
STRING	0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
MEMORY STATE	0	0	0	3	3	3	3	3	3	3	3	3	3	3	3
		1	1	1	1	1	4	5	4	4	4	4	4	4	4
			2	2	2	2	2	2	2	1	1	1	1	1	5
PAGE FAULT	F	F	F	F			F	F	F	F					F

```
Page Fault at 0, IN: 0, OUT: -1 / Memory residence set: 0 -1 -1
Page Fault at 1, IN: 1, OUT: -1 / Memory residence set: 0 1 -1
Page Fault at 2, IN: 2, OUT: -1 / Memory residence set: 0 1 2
Page Fault at 3, IN: 3, OUT: 0 / Memory residence set: 3 1 2
Page Fault at 6, IN: 4, OUT: 1 / Memory residence set: 3 4 2
Page Fault at 7, IN: 5, OUT: 4 / Memory residence set: 3 5 2
Page Fault at 8, IN: 4, OUT: 5 / Memory residence set: 3 4 2
Page Fault at 9, IN: 1, OUT: 2 / Memory residence set: 3 4 1
Page Fault at 14, IN: 5, OUT: 1 / Memory residence set: 3 4 5
LFU > page fault count: 9
```

위 표는 reference string에 대한 시뮬레이션 결과이고, 아래 사진은 작성한 코드를 실행했을 때 나오는 결과이다. 동일한 결과임을 확인할 수 있다.

5. CLOCK 기법

TIME	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
STRING	0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
MEMORY STATE	0/1	0/1	0/1	3/1	3/1	3/1	3/1	3/0	3/0	1/1	1/0	1/0	1/0	1/0	5/1
		1/1	1/1	1/0	1/0	1/0	4/1	4/0	4/1	4/1	3/1	3/1	3/1	3/1	3/1
			2/1	2/0	2/1	2/1	2/1	5/1	5/1	5/1	5/0	4/1	4/1	4/1	4/1
PAGE FAULT	F	F	F	F			F	F		F	F	F			F

```
Page Fault at 0, IN: 0, OUT: -1 / Memory residence set: 0 -1 -1
Page Fault at 1, IN: 1, OUT: -1 / Memory residence set: 0 1 -1
Page Fault at 2, IN: 2, OUT: -1 / Memory residence set: 0 1 2
Page Fault at 3, IN: 3, OUT: 0 / Memory residence set: 3 1 2
Page Fault at 6, IN: 4, OUT: 1 / Memory residence set: 3 4 2
Page Fault at 7, IN: 5, OUT: 2 / Memory residence set: 3 4 5
Page Fault at 9, IN: 1, OUT: 3 / Memory residence set: 1 4 5
Page Fault at 10, IN: 3, OUT: 4 / Memory residence set: 1 3 5
Page Fault at 11, IN: 4, OUT: 5 / Memory residence set: 1 3 4
Page Fault at 14, IN: 5, OUT: 1 / Memory residence set: 5 3 4
CLOCK > page fault count: 10
```

위 표는 reference string에 대한 시뮬레이션 결과이고, 아래 사진은 작성한 코드를 실행했을 때 나오는 결과이다. 동일한 결과임을 확인할 수 있다.

노란 부분은 clock이 해당 time에서 가리키고 있는 위치이고, n/m에서 n은 page 번호, m은 reference bit이다.

6. WS 기법

TIME	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
STRING	0	1	2	3	2	3	4	5	4	1	3	4	3	4	5
MEMORY STATE	0	0	0	0			4	4	4	4	4	4	4	4	4
		1	1	1	1			5	5	5	5				5
			2	2	2	2	2	2		1	1	1	1		
				3	3	3	3	3	3		3	3	3	3	3
PAGE FAULT	F	F	F	F			F	F		F	F				F

```
Page Fault at 0, IN: 0 / Memory residence set: 0
Page Fault at 1, IN: 1 / Memory residence set: 0 1
Page Fault at 2, IN: 2 / Memory residence set: 0 1 2
Page Fault at 3, IN: 3 / Memory residence set: 0 1 2 3
At 4, OUT 0 from page frame / Memory residence set: 1 2 3
At 5, OUT 1 from page frame / Memory residence set: 2 3
Page Fault at 6, IN: 4 / Memory residence set: 2 3 4
Page Fault at 7, IN: 5 / Memory residence set: 2 3 4 5
At 8, OUT 2 from page frame / Memory residence set: 3 4 5
Page Fault at 9, IN: 1 / At 9, OUT 3 from page frame / Memory residence set: 1 4 5
Page Fault at 10, IN: 3 / Memory residence set: 1 3 4 5
At 11, OUT 5 from page frame / Memory residence set: 1 3 4
At 13, OUT 1 from page frame / Memory residence set: 3 4
Page Fault at 14, IN: 5 / Memory residence set: 3 4 5
WS > page fault count: 9
```

위 표는 reference string에 대한 시뮬레이션 결과이고, 아래 사진은 작성한 코드를 실행했을 때 나오는 결과이다. 동일한 결과임을 확인할 수 있다. Memory residence set의 경우 순서와 상관없이 오름차순으로 출력하게 하였다.

```
# of pages: 6
# of page frame: 3
window size: 3
length of refernce string: 15
reference string: 0 1 2 3 2 3 4 5 4 1 3 4 3 4 5

MIN: 7
FIFO: 10
LRU: 9
LFU: 9
CLOCK: 10
WS: 9
```

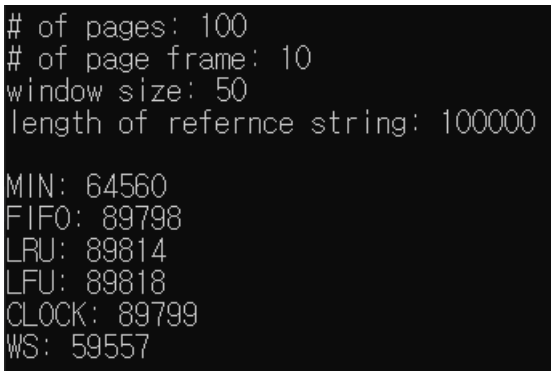
6가지 기법들의 결과가 출력되고 나서 왼쪽과 같은 요약된 결과를 확인할 수 있다.

2) 100 10 10 1000인 test case

Reference string의 경우에는 0부터 page 수까지 랜덤으로 reference string length만큼 생성한 test case이다. 1)의 경우처럼 table을 그리면서 하나씩 확인할 수는 없지만 위 사진처럼 오류 없이 정상 작동함을 알 수 있다.

3) 최대 입력값을 넣었을 때

1. 100 10 50 100000

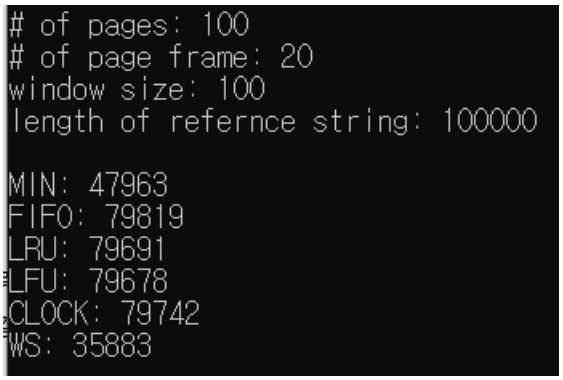


Reference string의 경우에는 0부터 page 수까지 랜덤으로 reference string length만큼 생성한 test case이다.

Reference string의 경우 너무 길어 생략하였다.

오류 없이 정상 작동함을 알 수 있다.

2. 100 20 100 100000



1.의 test case와 같은 reference string을 사용하되 page frame의 크기를 10에서 20으로 늘려보았다.

Page frame의 크기와 관계 있는 MIN, FIFO, LRU, LFU, CLOCK의 기법들의 page fault가 감소함을 알 수 있다.

즉, page frame의 크기가 커질수록 page fault 횟수는 감소한다.

Window size가 커질수록 많은 page를 page frame 내에 위치시킬 수 있기 때문에 page fault의 횟수가 감소하는데 test case에서 Window size도 50에서 100으로 늘림에 따라 WS 기법의 page fault 횟수가 감소함을 볼 수 있다.