

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)
- $N \leq 100$, $M \leq 20$, $W \leq 100$, $K \leq 100000$
- 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 실제로는 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 which 배열에서 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];
        zeroOrOne[w] = 1;
    }
    else {
        count++;
        while (zeroOrOne[out] == 1) { // reference bit이 0인 page를 찾을 때까지
            zeroOrOne[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;
        zeroOrOne[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

```
# of pages: 100
# of page frame: 10
window size: 10
length of reference string: 1000
reference string: 70 20 62 68 76 86 0 57 85 20 87 11 89 46 71 6 1 66 64 0 29 98 3 98 69 1 36 27 54 40 8 64 92 26 31 72 83 58 98 52 86 58 41
92 68 3 44 51 5 62 4 71 61 18 23 10 63 45 36 16 66 73 25 83 44 29 7 86 98 52 21 72 4 9 86 9 7 87 0 70 61 20 52 13 59 97 33 24 95 80 20 31 2
83 1 61 94 28 6 25 12 34 20 33 16 72 88 14 28 20 46 57 45 85 96 70 95 7 97 24 47 64 11 84 0 22 56 49 26 30 92 83 67 39 52 58 29 72 1 90 25 8
6 32 16 42 43 41 79 68 94 9 0 44 90 56 14 79 97 48 87 30 59 46 66 92 10 26 20 23 27 7 70 73 15 12 92 52 40 60 5 56 26 11 16 92 5 38 69 22 46
25 9 74 65 0 95 61 39 98 88 90 60 28 45 38 14 22 60 71 53 5 98 67 15 66 84 15 87 64 72 88 22 61 23 67 65 83 41 48 33 82 23 24 20 49 3 10 66
52 12 80 79 91 29 90 90 18 24 49 59 20 64 71 75 21 64 82 95 35 48 28 84 62 71 15 95 68 29 0 27 11 4 16 63 14 0 98 39 11 25 21 84 28 25 82 3
4 73 16 1 30 35 16 11 33 55 52 33 33 37 79 56 47 33 88 30 89 44 2 33 35 42 22 30 23 23 26 10 85 15 19 17 71 46 48 54 31 18 6 73 74 37 77 18
78 70 64 78 86 34 6 36 84 50 45 47 29 54 58 6 30 76 66 35 68 29 73 90 2 6 97 94 19 9 66 23 38 46 11 12 83 45 34 7 46 94 1 2 39 81 30 26 45 3
7 39 31 22 94 38 22 89 15 72 96 45 59 62 73 30 45 64 52 26 0 86 57 21 49 65 59 9 24 79 79 49 39 78 51 62 5 20 85 54 89 46 47 86 32 70 33 76
93 35 41 10 19 33 38 7 78 87 77 44 64 24 97 20 54 16 68 97 21 60 44 37 75 95 82 64 60 12 83 75 75 1 61 21 59 94 62 44 70 31 78 54 27 14 27 6
8 76 48 3 27 19 84 90 8 62 91 56 54 58 2 15 5 60 22 73 58 86 57 50 20 29 51 18 39 39 51 26 68 0 82 24 59 4 3 23 0 39 87 30 32 94 91 80 70 12
58 56 82 39 34 67 25 15 20 29 27 11 58 6 11 47 38 65 14 98 24 67 5 10 20 49 42 1 44 79 48 69 64 90 31 39 26 55 55 1 72 3 47 71 16 89 17 21
33 92 11 60 10 14 72 94 75 67 87 76 2 65 40 12 27 3 62 51 75 74 68 52 76 19 14 20 48 51 93 55 24 68 51 47 21 3 35 43 96 4 56 13 60 50 57 37
4 41 40 32 94 87 39 87 90 1 4 47 68 47 39 46 15 8 4 92 7 55 50 44 1 84 42 17 90 30 92 81 82 94 24 45 11 16 34 63 24 37 80 42 51 12 66 25 62
77 69 12 74 91 20 40 69 45 54 26 18 39 50 7 77 94 69 98 24 55 12 43 35 22 34 31 47 62 41 92 38 14 29 44 68 45 85 56 79 46 33 60 18 60 65 7 7
2 41 48 87 45 26 76 62 38 80 7 16 83 79 48 46 27 55 62 8 88 49 94 60 77 62 57 51 9 0 76 65 31 29 35 15 57 16 86 28 8 77 57 12 49 24 59 54 64
8 48 72 1 40 31 83 22 92 4 38 39 90 57 7 91 35 62 73 20 44 25 5 3 26 11 81 66 68 28 49 54 27 72 17 51 43 51 69 20 79 17 56 82 92 81 74 29 7
7 6 5 95 53 8 14 58 34 63 18 30 96 64 33 42 25 45 18 95 74 44 72 72 13 26 67 24 44 98 59 53 63 58 50 22 52 1 59 94 53 41 92 28 80 68 54 35 5
6 33 19 28 63 47 53 13 35 15 82 45 84 16 0 63 19 39 12 46 12 51 32 76 9 43 92 3 86 32 22 50 61 38 68 78 98 13 38 70 15 29 69 50 38 1 17 84 5
1 72 53 60 59 56 64 51 79 69 26 86 83 92 67 87 57 52 2 54 86 31 9 20 60 96 67 82 72 78 7 50 61 27 48 18 37 17 57 16 23 87 26 33 58 84 34 69
68 19 47 59 78 83 35 46 91 23 1 68 93 91 13 3 52 9 39 32 60 75 93 73 19 90 60 27 12 4 1 35 32 98 70 68 55 4 5 55
MIN: 657
FIFO: 905
LRU: 907
LFU: 906
CLOCK: 906
WS: 898
```

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

```
# of pages: 100
# of page frame: 10
window size: 50
length of reference string: 100000
MIN: 64560
FIFO: 89798
LRU: 89814
LFU: 89818
CLOCK: 89799
WS: 59557
```

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

```
# of pages: 100
# of page frame: 20
window size: 100
length of refernce string: 100000

MIN: 47963
FIFO: 79819
LRU: 79691
LFU: 79678
CLOCK: 79742
WS: 35883
```

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.