

운영체제 과제 #1

: 연립 방정식을 구하는 프로그램

과 목	운영체제 _02
전 공	컴퓨터공학과
학 번	20190850
이 름	이수진

목 차

I. 문제 기술

1. 문제 정의 및 해결

II. 세부 구현사항

1. 코드

2. 연립 방정식을 구하는 알고리즘

III. 체크 포인트

1. 질문

2. 성능 평가

3. 결과 분석

IV. 결론

V. 참고문헌

I. 문제 기술

1. 문제 정의 및 해결

문제는 아래 [그림 1]과 같은 연립 방정식을 간단한 forward / back substitution 방법을 이용하여 해를 구하는 프로그램의 작성이다. 이 문제는 사용자로부터 입력받은 파일을 open, read 명령어로 읽어 들이고, Gaussian, Back substitution 알고리즘을 이용하여 해결한 x 벡터를 open, write 명령어로 생성함으로 해결하였다.

$$\begin{aligned} a_{0,0}x_0 + a_{0,1}x_1 + \cdots + a_{0,n-1}x_{n-1} &= b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 + \cdots + a_{1,n-1}x_{n-1} &= b_1 \\ &\vdots \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + \cdots + a_{n-1,n-1}x_{n-1} &= b_{n-1} \end{aligned}$$

[그림 1] 연립 방정식

II. 세부 구현사항

1. 코드

(1) 명령어 read 사용

이진 파일인 dat 파일을 읽어 들이는 데 가장 난해를 겪었다. 처음 n의 값이 1000 이 나와야 하는데 엉뚱한 숫자가 나오는 등의 문제가 발생했다. 이는 int, float 형식인 data를 고려하지 않고, read에서 파일로부터 읽을 내용의 크기를 제대로 설정하지 않았기 때문에 발생하는 문제였다. 발생한 문제는 read로 읽을 때 먼저 자료형 선언 후, 메모리 주소와 파일로부터 읽을 내용의 크기를 올바르게 설정하여 다음과 같이 해결하였다.

```
int n1, n2;

Read(Afd, &n1, sizeof(n1));
Read(Bfd, &n2, sizeof(n2));

...

float Amat[n1][n1];
float bmat[n2];
float x[n2];

for (int i=0; i<n1; i++)
{
    memset(Amat[i], 0x00, sizeof(float)*n1);
}
```

```
memset(bmat, 0x00, sizeof(float)*n2);
memset(x, 0x00, sizeof(float)*n2);

for (int i=0;i<n1;i++)
{
    Read(Afd, Amat[i], sizeof(float)*n1);
}

Read(Bfd, bmat, sizeof(float)*n2);
```

2. 연립 방정식을 구하는 알고리즘

주어진 pseudo-code를 바탕으로 작성하였다. 테스트하는 과정에서 문제가 있음을 발견하여 변수를 조금 수정하였다.(ex. Gaussian - int j 선언 시 1 -> j+1)

(1) Gaussian

$n \times n$ 행렬인 A를 행 사다리꼴 행렬로 만드는 알고리즘이다.

```
void Gaussian(int n, float A[][n], float b[])
{
    for (int l=0;l<=n-2;l++)
    {
        for (int i=l+1;i<=n-1;i++)
        {
            for (int j=l+1;j<=n-1;j++)
            {
                A[i][j] = A[i][j] - (A[i][l] / A[l][l]) * A[l][j];
            }
            b[i] = b[i] - (A[i][l] / A[l][l]) * b[l];
        }
    }
}
```

(2) Back substitution

각 수식마다 미지수가 1개씩 남도록 다른 미지수를 제거하는 과정이다.

```

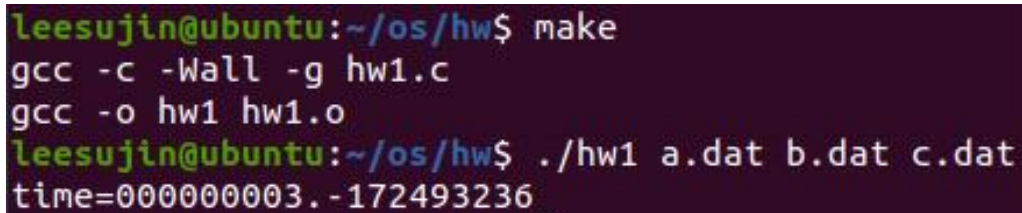
void BackSubstitution(int n, float A[][n], float b[], float x[])
{
    for (int i=n-1;i>=0;i--)
    {
        x[i] = b[i] / A[i][i];
        for (int j=0;j<=i-1;j++)
        {
            b[j] = b[j] - x[i] * A[j][i];
            A[j][i] = 0;
        }
    }
}

```

III. 체크 포인트

1. 수행 결과

입력 시 인자는 ./hw1 [행렬 A file] [벡터 b file] [벡터 x file]이며, 수행 시 수행 시간과 함께 c.dat 파일이 생성된다.



```

leesujin@ubuntu:~/os/hw$ make
gcc -c -Wall -g hw1.c
gcc -o hw1 hw1.o
leesujin@ubuntu:~/os/hw$ ./hw1 a.dat b.dat c.dat
time=000000003.-172493236

```

[그림 2] 수행 결과 스냅샷

2. 질문

(1) 행렬 파일 a.dat에서 행렬 원소 $A[i][j]$ 의 위치로 이동한다면, lseek()의 인자는 어떻게 입력해야 하는가?

`lseek(fd, sizeof(int)+sizeof(float)*(n*i+j), SEEK_SET);`

단, 여기서 fd는 파일 기술자이고 n은 $n \times n$ 행렬의 n을 의미한다.

a.dat 파일은 이진 integer 1개와 이진 float data n^2 인 행렬이 들어있다. 따라서 offset 인자는 sizeof(int), 그리고 행렬의 위치와 sizeof(float)을 곱한 값을 더해야 한다. 이 답안의 옳음을 다음과 같이 임의의 숫자로 증명하였다.

```

float x = 0;

lseek(Afd, sizeof(int)+sizeof(float)*(n1*9+9), SEEK_SET);
Read(Afd, &x, sizeof(x));
printf("x : %f \n", x);
printf("A[%d][%d] : %f\n", 9, 9, Amat[9][9]);

```

[그림 3] lseek 실행 코드

```
leesujin@ubuntu:~/os/hw$ ./test a.dat b.dat c.dat
x : 2.000000
A[9][9] : 2.000000
```

[그림 4] 그림 3의 실행 결과

여기서 x는 임의의 float형 수로, lseek 후 read를 통해 해당 위치에 있는 행렬값을 알기 위해 추가하였다. 이 외에도 10개의 임의의 수를 넣어본 결과, 모두 일치하였다.

3. 성능 평가

(1) 수행 시간

절차에 따른 수행 시간을 구하고 표로 나타내었다. 연립 방정식의 해를 구하는 시간이 가장 오래 걸렸음을 알 수 있다.

```
leesujin@ubuntu:~/os/hw$ ./hw1 a.dat b.dat c.dat
read time=000000000.010366601
solve time=000000003.-107602407
write time=000000000.000269292
```

[그림 5] 수행 시간 스냅샷

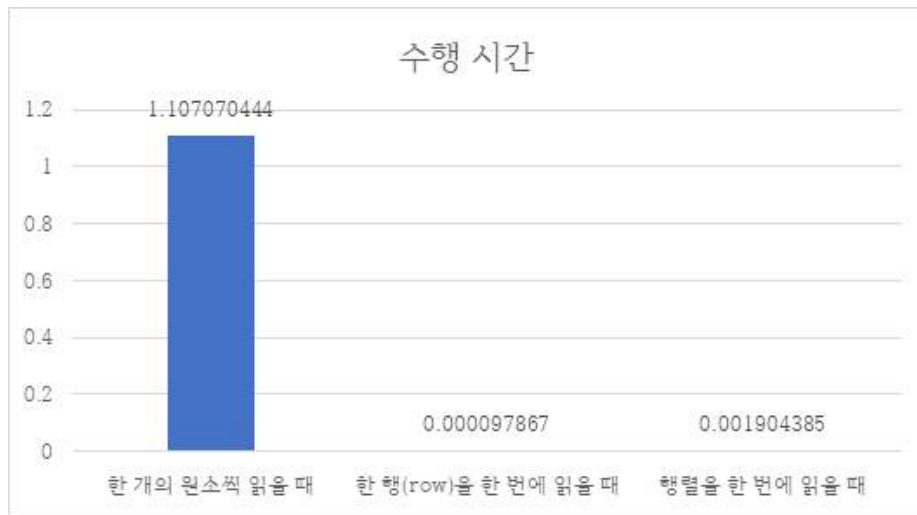
[표 1] 수행 시간 정리

항목	시간(sec.µsec)
두 데이터의 파일 읽기	000000000.005966657
연립 방정식의 해 구하기	000000003.371651335
결과 벡터를 파일에 저장	000000000.000277472

3. 결과 분석

(1) read()

행렬 데이터 파일을 읽을 때 다음과 같은 3개의 방식을 적용하고 I/O 시간을 구하여 그래프로 나타내었다. 한 개의 원소씩 읽을 때 시간이 가장 오래 걸렸다. 그리고 미미한 차이지만, 예상외로 행렬 전체보다 한 행(row)을 한 번에 읽을 때 시간이 덜 걸렸다.



IV. 결론

1. 전체 요약

행렬 및 벡터가 쓰인 이진 파일을 `open()`, `read()` 명령어로 읽고 연립 방정식을 해결, 그리고 결과를 이진 파일로 생성하는 프로그램을 작성하였다. 이때 시간은 연립 방정식의 해를 구하는 데 가장 오래 걸렸으며, `write()` 명령어로 A 행렬 파일을 읽을 때 한 개의 원소씩 읽으면 가장 오래 걸렸다.

2. 느낀 점

본 과제를 통해 파일을 읽고 쓸 때 쓰는 명령어인 `open()`, `read()`, `write()`를 다시 한번 복습할 수 있었다. 그리고 파일을 읽을 땐 바이트 수를 고려해야 한다는 것, 그리고 리눅스 환경에서 수행 시간을 구하는 법까지 배울 수 있었다. 앞으로의 과제 또한 연립 방정식과 관련된 것으로 알고 있는데 그때는 지금보다 능숙하고 깔끔한 코드를 짜서 해결했으면 좋겠다.

IV. 참고문헌

[1] "<https://reakwon.tistory.com/39>", 2021-03-20

[2] 같은 19학번 최연서 학생의 도움을 받아 Gaussian 함수에서 j 선언 시 l+1로 수정하였다.