

Data Structure1 Report



과목명		자료구조1
담당교수		홍 민
학과		컴퓨터소프트웨어공학과
학년		2학년
학번		20194059
이름		김태완
제출일		2020.04.14

목차

1. 두 회소 행렬을 연산하는 프로그램

1.1 문제 분석

1.2 소스 코드

1.3 소스 코드 분석

1.4 실행창

1.5 느낀점

2. 느낀점

1. 두 회소 행렬을 연산하는 프로그램

1.1 문제 분석

03과제

- 두개의 행렬 데이터를 **data.txt**파일에서 입력 받아, 두 행렬의 **+**, **-**, ***** 연산을 수행하는 프로그램을 작성하시오. 단, 회소행렬 자료구조와 동적 할당으로 2차원 배열을 생성하여 작성하여야 함. 제출일: 5월 14일(목)

- data.txt 파일 예



이번 문제는 파일에서 일반 행렬을 받아와 회소 행렬 방식으로 바꾼 뒤 회소 행렬을 연산하는 알고리즘을 통해 더하기, 빼기, 곱하기 의 연산을 하는 프로그램이다. 교수님께서 처음 과제를 내 주셨을 때는 회소 행렬 방식을 사용하여 해결하는 프로그램이 아닌 일반 행렬을 이용하는 프로그램 이었지만, 회소 행렬 방식을 이용해야 하기 때문에 프로그램의 난이도가 훨씬 높아진 것 같다. 이 프로그램을 작성하기 위해선 일반 행렬을 회소 행렬로 바꾸는 법, 회소 행렬을 이용해 더하기, 빼기, 곱하기의 연산을 하는 알고리즘을 작성하는 법을 더 알아야 한다.

1.2 소스코드

```
1 1/*****
2 작성자 : 20194059 김태환
3 작성일 : 2020.05.03
4 프로그램명 : 두 희소행렬을 연산하는 프로그램
5 *****/
6
7 #define _CTR_SECURE_NO_WARNINGS
8 #include <stdio.h>
9 #include <stdlib.h>
10
11 int** creat_memory(int rows,int cols); //2차원 메모리를 동적으로 받아주는 함수
12 int scan_data(int **arr, FILE *fp, int rows, int cols); //파일에서 데이터를 읽어오는 함수
13 void get_sparse_matrix(int **arr, int **sparse_matrix,int rows, int cols); //희소행렬을 압축하는 함수
14 void print_arr(int **arr,int rows,int cols); //행렬을 출력하는 함수
15 void print_arr2(int **arr,int rows,int cols); //희소행렬을 일반행렬로 출력해주는 함수
16 int arr_add(int **arr1,int **arr2, int **sum, int rows1,int cols1,int rows2, int cols2, int len1, int len2); //행렬의 덧셈을 실행하는 함수
17 int arr_sub(int **arr1,int **arr2, int **sub, int rows1,int cols1,int rows2, int cols2, int len1, int len2); //행렬의 뺄셈을 실행하는 함수
18 int arr_mul(int **arr1,int **arr2, int **mul, int rows1,int cols1,int rows2, int cols2, int len1, int len2); //행렬의 곱셈을 실행하는 함수
19 void delete_memory(int **arr, int rows); //2차원 메모리를 해제하는 함수
20
21 int main()
22 {
23     FILE *fp;
24     int cols1,rows1,cols2,rows2,len1,len2; //일반 행렬의 크기와 희소행렬의 크기를 저장해주는 변수들
25     int **arr1 = NULL, **arr2 = NULL; //일반 행렬을 저장하고 있는 동적 이중배열
26     int **s_matrix1 = NULL, **s_matrix2 = NULL; //희소행렬을 저장하고있는 동적 이중배열
27     int **sum = NULL, **sub = NULL, **mul = NULL; //연산의 결과를 저장하고 있는 동적 이중배열
28     int sizeof_sum, sizeof_sub, sizeof_mul; //희소행렬의 크기를 갖고 있는 변수
29     int i,j,max; //반복, 최대값을 저장하는 변수
30
31     fp = fopen("data.txt","r"); //파일 오픈
32     if(fp == NULL) //파일 오류시 오류 출력
33     {
34         printf("파일을 열 수 없습니다.");
35         return 0;
36     }
37
38     fscanf(fp,"%d %d",&rows1,&cols1); //행렬의 크기를 읽어온다
39     arr1 = creat_memory(rows1,cols1); //행렬의 크기만큼 메모리를 할당한다
40     len1 = scan_data(arr1,fp,rows1,cols1); //할당한 메모리에 파일에서 읽어온 데이터를 저장하고 0이 아닌 데이터의 갯수를 len1에 저장한다.
41     s_matrix1 = creat_memory(len1,3); //0이 아닌 데이터의 갯수 만큼 메모리를 할당한다.
42     get_sparse_matrix(arr1,s_matrix1,rows1,cols1); //파일에서 읽어온 데이터로 희소행렬을 만든다.
43
44     fscanf(fp,"%d %d",&rows2,&cols2); //행렬의 크기를 읽어온다
45     arr2 = creat_memory(rows2,cols2); //행렬의 크기만큼 메모리를 할당한다
46     len2 = scan_data(arr2,fp,rows2,cols2); //할당한 메모리에 파일에서 읽어온 데이터를 저장하고 0이 아닌 데이터의 갯수를 len2에 저장한다.
47     s_matrix2 = creat_memory(len2,3); //0이 아닌 데이터의 갯수 만큼 메모리를 할당한다.
48     get_sparse_matrix(arr2,s_matrix2,rows2,cols2); //파일에서 읽어온 데이터로 희소행렬을 만든다.
49
50     if(len1>len2) max = len1;
51     else max = len2; //더 큰 길이를 max에 저장
52
53     printf("<<첫번째 행렬(ver.일반행렬)>>\n\n");
54     print_arr(arr1,rows1,cols1); //첫번째 배열 출력
55     printf("\n\n<<첫번째 행렬(ver.희소행렬)>>\n\n");
56     print_arr(s_matrix1,len1,3); //첫번째 배열(희소행렬) 출력
57     printf("\n\n<<두번째 행렬(ver.일반행렬)>>\n\n");
58     print_arr(arr2,rows2,cols2); //두번째 배열 출력
59     printf("\n\n<<두번째 행렬(ver.희소행렬)>>\n\n");
60     print_arr(s_matrix2,len2,3); //두번째 배열(희소행렬) 출력
61
62     sum = creat_memory(max,3); //더 큰 크기를 기준으로 메모리 할당
63     sizeof_sum = arr_add(s_matrix1,s_matrix2,sum,rows1,cols1,rows2,cols2,len1,len2);
64     //행렬의 덧셈 연산을 진행하고, 반환된 행렬의 크기를 저장한다.
65     printf("\n\n<<두 행렬의 합(ver.일반행렬)>>\n\n");
66     print_arr2(sum,rows1,cols1); //합한 행렬을 출력한다.
67     printf("\n\n<<두 행렬의 합(ver.희소행렬)>>\n\n");
68     print_arr(sum,max,3); //합한 행렬을 출력한다.
69
70
71     sub = creat_memory(max,3); //더 큰 크기를 기준으로 메모리 할당
72     sizeof_sub = arr_sub(s_matrix1,s_matrix2,sub,rows1,cols1,rows2,cols2,len1,len2);
73     //행렬의 뺄셈 연산을 진행하고, 반환된 행렬의 크기를 저장한다.
74     printf("\n\n<<두 행렬의 차(ver.일반행렬)>>\n\n");
75     print_arr2(sub,rows1,cols1); //뺀 행렬을 출력한다.
76     printf("\n\n<<두 행렬의 차(ver.희소행렬)>>\n\n");
77     print_arr(sub,max,3); //합한 행렬을 출력한다.
78
79     mul = creat_memory(rows1 + cols2,3); //곱셈 결과 나올 수 있는 최대 크기로 메모리 할당
80     sizeof_mul = arr_mul(s_matrix1,s_matrix2,mul,rows1,cols1,rows2,cols2,len1,len2);
81     //행렬의 곱셈 연산을 진행하고, 반환된 행렬의 크기를 저장한다.
82     printf("\n\n<<두 행렬의 곱(ver.일반행렬)>>\n\n");
83     print_arr2(mul,rows1,cols2); //곱해진 행렬을 출력한다.
84     printf("\n\n<<두 행렬의 곱(ver.희소행렬)>>\n\n");
85     print_arr(mul,rows1+cols2,3); //합한 행렬을 출력한다.
86 }
```

```

87     fclose(fp); //파일 닫기
88     delete_memory(arr1, rows1);
89     delete_memory(arr2, rows2);
90     delete_memory(s_matrix1, len1);
91     delete_memory(s_matrix2, len2);
92     delete_memory(sum, max);
93     delete_memory(sub, max);
94     delete_memory(mul, rows1); //메모리 반납
95     return 0;
96 }
97
98 int** creat_memory(int rows, int cols) //2차원 메모리를 동적으로 할당해주는 함수
99 {
100     int **arr;
101     int i;
102     arr = (int **)calloc(rows, sizeof(int)); //메모리 할당과 동시에 0으로 초기화 해주기 위해 calloc 함수 사용
103     for(i=0; i<rows; i++)
104     {
105         arr[i] = (int *)calloc(cols, sizeof(int));
106     }
107     return arr;
108 }
109
110 int scan_data(int **arr, FILE *fp, int rows, int cols) //파일로부터 데이터를 읽어오고 행렬의 크기를 반환하는 함수
111 {
112     int i, j, count=0;
113     for(i=0; i<rows; i++)
114     {
115         for(j=0; j<cols; j++)
116         {
117             fscanf(fp, "%d", &arr[i][j]);
118             if(arr[i][j]!=0)
119                 count++;
120         }
121     }
122     return count;
123 }
124
125 void get_sparse_matrix(int **arr, int **sparse_matrix, int rows, int cols) //일반행렬을 희소행렬로 바꿔주는 함수
126 {
127     int i, j, count=0;
128     for(i=0; i<rows; i++)
129     {
130         for(j=0; j<cols; j++)
131         {
132             if(arr[i][j]!=0) //0이 아니면
133             {
134                 sparse_matrix[count][0] = i; //count행의 0열에 i
135                 sparse_matrix[count][1] = j; //count행의 1열에 j
136                 sparse_matrix[count][2] = arr[i][j]; //count행의 2열에 값을 대입 해준다
137                 count++;
138             }
139         }
140     }
141 }
142
143 void print_arr(int **arr, int rows, int cols) //행렬을 출력하는 함수
144 {
145     int i, j;
146     for(i=0; i<rows; i++)
147     {
148         for(j=0; j<cols; j++)
149         {
150             printf("%3d ", arr[i][j]);
151         }
152         printf("\n");
153     }
154 }
155
156 void print_arr2(int **arr, int rows, int cols) //희소행렬을 일반함수로 출력하는 함수
157 {
158     int i, j, count=0;
159     for(i=0; i<rows; i++)
160     {
161         for(j=0; j<cols; j++)
162         {
163             if(i==arr[count][0] && j==arr[count][1]) //인덱스 번호와 희소행렬에 저장된 행렬의 값이 같으면 값 출력
164             {
165                 printf("%3d", arr[count][2]);
166                 count++;
167             }
168             else //아니면 0 출력
169             {
170                 printf("%3d", 0);
171             }
172         }
173         printf("\n");
174     }
175 }
176 }
177

```

```

179 int arr_add(int **arr1, int **arr2, int **sum, int rows1, int cols1, int rows2, int cols2, int len1, int len2)
180 {
181     int i, j, idx;
182     int count=0, max;
183     int **buf;
184     if(cols1!=cols2 || rows1 != rows2) //행렬의 덧셈을 할 수 없는 경우 오류 메시지 출력
185     {
186         printf("행렬의 크기가 같지 않아 행렬의 덧셈 연산을 수행 할 수 없습니다.\n");
187         return;
188     }
189
190     if(len1>=len2) max = len1;
191     else max = len2; //두개의 길이 중 더 큰 길이 저장
192
193     buf = creat_memory(max,3); //임시 저장을 위한 메모리 생성
194     for(i=0; i<len1; i++)
195     {
196         idx = arr1[i][0] + cols1 + arr1[i][1];
197         buf[idx][0] += arr1[i][0];
198         buf[idx][1] += arr1[i][1];
199         buf[idx][2] = arr1[i][2];
200     } //arr1을 buf메모리의 각 자리에 대입
201     for(i=0; i<len2; i++)
202     {
203         idx = arr2[i][0] + cols2 + arr2[i][1];
204         buf[idx][2] += arr2[i][2];
205     } //arr1이 더해져있는 buf에 arr2도 마찬가지로 각 자리에 더해준다
206     for(i=0; i<max; i++)
207     {
208         if(buf[i][2]!=0)
209         {
210             sum[count][0] = buf[i][0];
211             sum[count][1] = buf[i][1];
212             sum[count][2] = buf[i][2];
213             count++;
214         }
215     } //buf 전체중 0이 아닌값을 찾아서 sum에 대입
216     delete_memory(buf, max); //메모리 반환
217     return count;
218 }
219
220 int arr_sub(int **arr1, int **arr2, int **sub, int rows1, int cols1, int rows2, int cols2, int len1, int len2)
221 {
222     int i, j, idx;
223     int count=0, max;
224     int **buf;
225     if(cols1!=cols2 || rows1 != rows2) //행렬의 뺄셈을 할 수 없는 경우 오류 메시지 출력
226     {
227         printf("행렬의 크기가 같지 않아 행렬의 뺄셈 연산을 수행 할 수 없습니다.\n");
228         return;
229     }
230
231     if(len1>=len2) max = len1; //두개의 길이 중 더 큰 길이 저장
232     else max = len2;
233
234     buf = creat_memory(max,3); //임시 저장을 위한 메모리 생성
235     for(i=0; i<len1; i++)
236     {
237         idx = arr1[i][0] + cols1 + arr1[i][1];
238         buf[idx][0] += arr1[i][0];
239         buf[idx][1] += arr1[i][1];
240         buf[idx][2] = arr1[i][2];
241     } //arr1을 buf메모리의 각 자리에 대입
242     for(i=0; i<len2; i++)
243     {
244         idx = arr2[i][0] + cols2 + arr2[i][1];
245         buf[idx][2] -= arr2[i][2];
246     } //arr1이 더해져있는 buf에 arr2는 각 자리에 빼준다
247     for(i=0; i<max; i++)
248     {
249         if(buf[i][2]!=0)
250         {
251             sub[count][0] = buf[i][0];
252             sub[count][1] = buf[i][1];
253             sub[count][2] = buf[i][2];
254             count++;
255         }
256     } //buf 전체중 0이 아닌값을 찾아서 sum에 대입
257     delete_memory(buf, max); //메모리 반환
258     return count;
259 }
260

```

```

261 int arr_mul(int **arr1, int **arr2, int **mul, int rows1, int cols1, int rows2, int cols2, int len1, int len2)
262 {
263     int i, j, k;
264     int n, count=0, idx;
265     int **buf1, **buf2;
266     if(rows1!=cols2)
267     {
268         printf("행렬의 크기가 같지 않아 행렬의 곱셈 연산을 수행 할 수 없습니다.\n");
269         return;
270     }
271     buf1 = creat_memory(rows1+cols1, 3);
272     buf2 = creat_memory(rows2+cols2, 3);
273     for(i=0; i<rows1+cols1; i++)
274     {
275         idx = arr1[i][0] * cols1 + arr1[i][1];
276         buf1[idx][0] += arr1[i][0];
277         buf1[idx][1] += arr1[i][1];
278         buf1[idx][2] = arr1[i][2];
279     } //arr1을 buf1메모리의 각 자리에 대입
280     for(i=0; i<rows2+cols2; i++)
281     {
282         idx = arr2[i][0] * cols1 + arr2[i][1];
283         buf2[idx][0] += arr2[i][0];
284         buf2[idx][1] += arr2[i][1];
285         buf2[idx][2] = arr2[i][2];
286     } //arr2를 buf2메모리의 각 자리에 대입
287     for(i=0; i<rows1; i++)
288     {
289         for(j=0; j<cols2; j++)
290         {
291             n=0;
292             for(k=0; k<cols2; k++)
293             {
294                 if(n==0) //처음에만 행, 열값 대입
295                 {
296                     mul[count][0] = i;
297                     mul[count][1] = j;
298                     n++;
299                 }
300                 mul[count][2] += buf1[rows1+i+k][2] * buf2[rows2+k+j][2]; //곱해진 값을 더해 주면서 대입
301             }
302             count++;
303         }
304     }
305     delete_memory(buf1, rows1+cols1);
306     delete_memory(buf2, rows2+cols2);
307     return count;
308 }
309
310 void delete_memory(int **arr, int rows) //동적으로 할당된 메모리를 반납하는 함수
311 {
312     int i;
313     for(i=0; i<rows; i++)
314         free(arr[i]);
315     free(arr);
316 }
317

```

1.3 소스코드 분석

main함수의 이해를 더 편하기 하기 위해 사용된 함수부터 분석 하겠습니다.

```

98 int** creat_memory(int rows, int cols) //2차원 메모리를 동적으로 할당해주는 함수
99 {
100     int **arr;
101     int i;
102     arr = (int **)calloc(rows, sizeof(int)); //메모리 할당과 동시에 0으로 초기화 해주기 위해 calloc 함수 사용
103     for(i=0; i<rows; i++)
104     {
105         arr[i] = (int *)calloc(cols, sizeof(int));
106     }
107     return arr;
108 }
109

```

1, 2차원의 메모리를 동적으로 할당해주는 함수이다. 행과 열의 크기를 받아와 그 크기로 메모리를 할당해준 후, 할당된 메모리를 반환한다.

```

110 int scan_data(int **arr, FILE *fp, int rows, int cols) //파일로부터 데이터를 읽어오고 행렬의 크기를 반환하는 함수
111 {
112     int i, j, count=0;
113     for(i=0; i<rows; i++)
114     {
115         for(j=0; j<cols; j++)
116         {
117             fscanf(fp, "%d", &arr[i][j]);
118             if(arr[i][j]!=0)
119                 count++;
120         }
121     }
122     return count;
123 }
124

```

2, 파일에 있는 행렬의 데이터를 읽어오는 함수이다. 마찬가지로 행과 열의 크기를 받아와 그 크기만큼 행렬의 데이터 값을 읽어 오고, 희소 행렬을 만들어야 하기 때문에 0이 아닌 경우 count의 값을 늘려가면서 0이 아닌 갯수를 반환한다.

```

125 void get_sparse_matrix(int **arr, int **sparse_matrix, int rows, int cols) //일반행렬을 희소행렬로 바꿔주는 함수
126 {
127     int i, j, count=0;
128     for(i=0; i<rows; i++)
129     {
130         for(j=0; j<cols; j++)
131         {
132             if(arr[i][j]!=0) //0이 아니면
133             {
134                 sparse_matrix[count][0] = i; //count행의 0열에 i
135                 sparse_matrix[count][1] = j; //count행의 1열에 j
136                 sparse_matrix[count][2] = arr[i][j]; //count행의 2열에 값을 대입 해준다
137                 count++;
138             }
139         }
140     }
141 }
142

```

3, 일반 행렬을 희소 행렬로 바꾸는 함수로 일반 행렬, 저장될 희소 행렬을 받아와 count의 값을 늘려가며 count행 0열에는 행의 값을, count행 1열에는 열의 값을, count행의 2열에는 행렬의 값을 저장한다.

```

144 void print_arr(int **arr, int rows, int cols) //행렬을 출력하는 함수
145 {
146     int i, j;
147     for(i=0; i<rows; i++)
148     {
149         for(j=0; j<cols; j++)
150         {
151             printf("%3d ", arr[i][j]);
152         }
153         printf("\n");
154     }
155 }
156

```

4, 행렬을 출력하는 함수로, 행, 열의 값과 출력할 행렬을 받아와 출력한다.

```

157 void print_arr2(int **arr, int rows, int cols) //희소행렬을 일반함수로 출력하는 함수
158 {
159     int i, j, count=0;
160     for(i=0; i<rows; i++)
161     {
162         for(j=0; j<cols; j++)
163         {
164             if(i==arr[count][0]&&j==arr[count][1]) //인덱스 번호와 희소행렬에 저장된 행렬의 값이 같으면 값 출력
165             {
166                 printf("%3d", arr[count][2]);
167                 count++;
168             }
169             else //아니면 0 출력
170             {
171                 printf("%3d", 0);
172             }
173         }
174         printf("\n");
175     }
176 }
177

```

5, 희소 행렬을 입력 받아 일반 행렬 의 방식으로 바꾸어 출력하는 함수로써 count값을 늘려가면서 i와 arr[count][0], j와 arr[count][1]의 값이 같으면

arr[count][2]에 저장된 값을, 같지 않으면 0을 출력하는 함수이다.

```
179 int arr_add(int **arr1, int **arr2, int **sum, int rows1, int cols1, int rows2, int cols2, int len1, int len2)
180 {
181     int i, j, idx;
182     int count=0, max;
183     int **buf;
184     if(cols1!=cols2 || rows1 != rows2) //행렬의 덧셈을 할 수 없는 경우 오류 메시지 출력
185     {
186         printf("행렬의 크기가 같지 않아 행렬의 덧셈 연산을 수행 할 수 없습니다.\n");
187         return;
188     }
189
190     if(len1>=len2) max = len1;
191     else max = len2; //두개의 길이 중 더 큰 길이 저장
192
193     buf = creat_memory(max,3); //임시 저장을 위한 메모리 생성
194     for(i=0; i<len1; i++)
195     {
196         idx = arr1[i][0] * cols1 + arr1[i][1];
197         buf[idx][0] += arr1[i][0];
198         buf[idx][1] += arr1[i][1];
199         buf[idx][2] = arr1[i][2];
200     } //arr1을 buf메모리의 각 자리에 대입
201     for(i=0; i<len2; i++)
202     {
203         idx = arr2[i][0] * cols2 + arr2[i][1];
204         buf[idx][2] += arr2[i][2];
205     } //arr1이 더해져있는 buf에 arr2도 마찬가지로 각 자리에 더해준다
206     for(i=0; i<max; i++)
207     {
208         if(buf[i][2]!=0)
209         {
210             sum[count][0] = buf[i][0];
211             sum[count][1] = buf[i][1];
212             sum[count][2] = buf[i][2];
213             count++;
214         }
215     } //buf 전체중 0이 아닌값을 찾아서 sum에 대입
216     delete_memory(buf, max); //메모리 반환
217     return count;
218 }
219
```

6, 행렬의 덧셈을 진행하는 함수로, buf라는 임시 저장 공간을 새로 만든 후 arr1의 행과 열의 값에 해당하는 위치 (arr1[i][0] * cols1 + arr1[i][1]) 에 대입 후 arr1의 데이터가 저장된 buf에 arr2를 마찬가지로 해당하는 위치 (arr2[i][0] * cols2 + arr2[i][1]) 에 더해준다. 그리고 buf[i][2]의 값이 0이 아닌 것들만 골라 sum에 대입 해준다.

```
220 int arr_sub(int **arr1, int **arr2, int **sub, int rows1, int cols1, int rows2, int cols2, int len1, int len2)
221 {
222     int i, j, idx;
223     int count=0, max;
224     int **buf;
225     if(cols1!=cols2 || rows1 != rows2) //행렬의 뺄셈을 할 수 없는 경우 오류 메시지 출력
226     {
227         printf("행렬의 크기가 같지 않아 행렬의 뺄셈 연산을 수행 할 수 없습니다.\n");
228         return;
229     }
230
231     if(len1>=len2) max = len1; //두개의 길이 중 더 큰 길이 저장
232     else max = len2;
233
234     buf = creat_memory(max,3); //임시 저장을 위한 메모리 생성
```

```

235     for(i=0; i<len1; i++)
236     {
237         idx = arr1[i][0] * cols1 + arr1[i][1];
238         buf[idx][0] += arr1[i][0];
239         buf[idx][1] += arr1[i][1];
240         buf[idx][2] = arr1[i][2];
241     } //arr1을 buf메모리의 각 자리에 대입
242     for(i=0; i<len2; i++)
243     {
244         idx = arr2[i][0] * cols2 + arr2[i][1];
245         buf[idx][2] -= arr2[i][2];
246     } //arr1이 더해져있는 buf에 arr2는 각 자리에 빼준다
247     for(i=0; i<max; i++)
248     {
249         if(buf[i][2]!=0)
250         {
251             sub[count][0] = buf[i][0];
252             sub[count][1] = buf[i][1];
253             sub[count][2] = buf[i][2];
254             count++;
255         }
256     } //buf 전체중 0이 아닌값을 찾아서 sum에 대입
257     delete_memory(buf, max); //메모리 반환
258     return count;
259 }
260

```

7, 빼기 연산을 진행하는 함수로 sum과 마찬가지로 진행되지만 arr1이 저장된 buf에 arr2를 빼주는 방식으로 바꾸어 빼기 연산을 진행 해 준다.

```

261 int arr_mul(int **arr1, int **arr2, int **mul, int rows1, int cols1, int rows2, int cols2, int len1, int len2)
262 {
263     int i, j, k;
264     int n, count=0, idx;
265     int **buf1, **buf2;
266     if(rows1!=cols2)
267     {
268         printf("행렬의 크기가 같지 않아 행렬의 곱셈 연산을 수행 할 수 없습니다.\n");
269         return;
270     }
271     buf1 = creat_memory(rows1+cols1, 3);
272     buf2 = creat_memory(rows2+cols2, 3);
273     for(i=0; i<rows1+cols1; i++)
274     {
275         idx = arr1[i][0] * cols1 + arr1[i][1];
276         buf1[idx][0] += arr1[i][0];
277         buf1[idx][1] += arr1[i][1];
278         buf1[idx][2] = arr1[i][2];
279     } //arr1을 buf1메모리의 각 자리에 대입
280     for(i=0; i<rows2+cols2; i++)
281     {
282         idx = arr1[i][0] * cols1 + arr1[i][1];
283         buf2[idx][0] += arr2[i][0];
284         buf2[idx][1] += arr2[i][1];
285         buf2[idx][2] = arr2[i][2];
286     } //arr2를 buf2메모리의 각 자리에 대입
287     for(i=0; i<rows1; i++)
288     {
289         for(j=0; j<cols2; j++)
290         {
291             n=0;
292             for(k=0; k<cols2; k++)
293             {
294                 if(n==0) //처음에만 행, 열값 대입
295                 {
296                     mul[count][0] = i;
297                     mul[count][1] = j;
298                     n++;
299                 }
300                 mul[count][2] += buf1[rows1+i+k][2] * buf2[rows2+k+j][2]; //곱해진 값을 더해주면서 대입
301             }
302             count++;
303         }
304     }
305     delete_memory(buf1, rows1+cols1);
306     delete_memory(buf2, rows2+cols2);
307     return count;
308 }
309

```

8, 행렬의 곱하기 연산을 하는 함수로 buf1에는 arr1의 값을 해당하는 위치에 저장, buf2에는 arr2의 값을 저장하고, 곱셈 연산을 하는 위치를 찾아가 곱하여 더해주는 방식으로 진행한다.

```

310 void delete_memory(int **arr, int rows) //동적으로 할당된 메모리를 반납하는 함수
311 {
312     int i;
313     for(i=0; i<rows; i++)
314         free(arr[i]);
315     free(arr);
316 }
317

```

9, 2차원으로 동적 할당된 메모리를 해제 해주는 함수이다.

여기서부터 main함수 시작

```

1  /******
2  작성자 : 20194059 김태관
3  작성일 : 2020.05.03
4  프로그램명 : 두 희소행렬을 연산하는 프로그램
5  *****/
6
7  #define _CTR_SECURE_NO_WARNINGS
8  #include <stdio.h>
9  #include <stdlib.h>
10

```

10, 작성자, 작성일, 프로그램명이 포함된 주석을 작성하고 프로그램에 필요한 헤더 파일을 추가한다.

```

11 int** creat_memory(int rows,int cols); //2차원 메모리를 동적으로 받아주는 함수
12 int scan_data(int **arr, FILE *fp, int rows, int cols); //파일에서 데이터를 읽어오는 함수
13 void get_sparse_matrix(int **arr, int **sparse_matrix,int rows, int cols); //희소행렬을 압축하는 함수
14 void print_arr(int **arr,int rows,int cols); //행렬을 출력하는 함수
15 void print_arr2(int **arr,int rows,int cols); //희소행렬을 일반행렬로 출력해주는 함수
16 int arr_add(int **arr1,int **arr2, int **sum, int rows1,int cols1,int rows2, int cols2, int len1, int len2); //행렬의 덧셈을 실행하는 함수
17 int arr_sub(int **arr1,int **arr2, int **sub, int rows1,int cols1,int rows2, int cols2, int len1, int len2); //행렬의 뺄셈을 실행하는 함수
18 int arr_mul(int **arr1,int **arr2, int **mul, int rows1,int cols1,int rows2, int cols2, int len1, int len2); //행렬의 곱셈을 실행하는 함수
19 void delete_memory(int **arr, int rows); //2차원 메모리를 해제하는 함수
20

```

11, 아래에 작성한 함수들에 대한 원형을 정의해준다.

```

21 int main()
22 {
23     FILE *fp;
24     int cols1,rows1,cols2,rows2,len1,len2; //일반 행렬의 크기와 희소행렬의 크기를 저장해주는 변수들
25     int **arr1 = NULL, **arr2 = NULL; //일반 행렬을 저장하고 있는 동적 이중배열
26     int **s_matrix1 = NULL, **s_matrix2 = NULL; //희소행렬을 저장하고있는 동적 이중배열
27     int **sum = NULL, **sub = NULL, **mul = NULL; //연산의 결과를 저장하고 있는 동적 이중배열
28     int sizeof_sum, sizeof_sub, sizeof_mul; //희소행렬의 크기를 갖고 있는 변수
29     int i,j,max; //반복, 최대값을 저장하는 변수
30

```

12, 메인 함수를 시작 하고, 프로그램에 필요한 변수들을 생성 해준다.

```

31     fp = fopen("data.txt","r"); //파일 오픈
32     if(fp == NULL) //파일 오류시 오류 출력
33     {
34         printf("파일을 열 수 없습니다.");
35         return 0;
36     }
37

```

13, 파일을 읽기 모드로 열고, 파일을 정상적으로 열 수 없는 경우 경고문을 출력한 후 프로그램을 종료한다.

```

38     fscanf(fp,"%d %d",&rows1,&cols1); //행렬의 크기를 읽어온다
39     arr1 = creat_memory(rows1,cols1); //행렬의 크기만큼 메모리를 할당한다
40     len1 = scan_data(arr1,fp,rows1,cols1); //할당된 메모리에 파일에서 읽어온 데이터를 저장하고 0이 아닌 데이터의 갯수를 len1에 저장한다.
41     s_matrix1 = creat_memory(len1,3); //0이 아닌 데이터의 갯수 만큼 메모리를 할당한다.
42     get_sparse_matrix(arr1,s_matrix1,rows1,cols1); //파일에서 읽어온 데이터로 희소행렬을 만든다.
43
44     fscanf(fp,"%d %d",&rows2,&cols2); //행렬의 크기를 읽어온다
45     arr2 = creat_memory(rows2,cols2); //행렬의 크기만큼 메모리를 할당한다
46     len2 = scan_data(arr2,fp,rows2,cols2); //할당된 메모리에 파일에서 읽어온 데이터를 저장하고 0이 아닌 데이터의 갯수를 len2에 저장한다.
47     s_matrix2 = creat_memory(len2,3); //0이 아닌 데이터의 갯수 만큼 메모리를 할당한다.
48     get_sparse_matrix(arr2,s_matrix2,rows2,cols2); //파일에서 읽어온 데이터로 희소행렬을 만든다.
49

```

14, 먼저 첫번째 행렬의 크기를 읽어오고, 그 크기를 바탕으로 메모리를 생성, 생성한 메모리에 행렬의 값을 대입하고 희소 행렬을 저장할 메모리를 생성 해준 뒤, 일반 행렬을 희소 행렬로 바꿔준다. 두번째 행렬도 똑같이 해준다.

```

50 |         if(len1>=len2) max = len1;
51 |         else max = len2; //더 큰 길이를 max에 저장
52 |

```

15, 두 희소 행렬의 길이 중 더 큰 값을 저장 해준다.

```

53 |     printf("<<첫번째 행렬(ver.일반행렬)>>\n\n");
54 |     print_arr(arr1,rows1,cols1); //첫번째 배열 출력
55 |     printf("\n\n<<첫번째 행렬(ver.희소행렬)>>\n\n");
56 |     print_arr(s_matrix1,len1,3); //첫번째 배열(희소행렬) 출력
57 |     printf("\n\n<<두번째 행렬(ver.일반행렬)>>\n\n");
58 |     print_arr(arr2,rows2,cols2); //두번째 배열 출력
59 |     printf("\n\n<<두번째 행렬(ver.희소행렬)>>\n\n");
60 |     print_arr(s_matrix2,len2,3); //두번째 배열(희소행렬) 출력
61 |

```

16, 첫번째 행렬과 두번째 행렬을 각각 일반 행렬, 희소 행렬 두가지 버전으로 출력 해준다.

```

62 |     sum = creat_memory(max,3); //더 큰 크기를 기준으로 메모리 할당
63 |     sizeof_sum = arr_add(s_matrix1,s_matrix2,sum,rows1,cols1,rows2,cols2,len1,len2);
64 |     //행렬의 덧셈 연산을 진행하고, 반환된 행렬의 크기를 저장한다.
65 |     printf("\n\n<<두 행렬의 합(ver.일반행렬)>>\n\n");
66 |     print_arr2(sum,rows1,cols1); //합한 행렬을 출력한다.
67 |     printf("\n\n<<두 행렬의 합(ver.희소행렬)>>\n\n");
68 |     print_arr(sum,max,3); //합한 행렬을 출력한다.
69 |
70 |

```

17, 덧셈 연산 결과를 저장할 sum의 메모리를 할당 해주고, 덧셈 연산을 진행한 뒤 그 결과를 마찬가지로 일반 행렬, 희소 행렬 두가지 버전으로 출력 해준다.

```

71 |     sub = creat_memory(max,3); //더 큰 크기를 기준으로 메모리 할당
72 |     sizeof_sub = arr_sub(s_matrix1,s_matrix2,sub,rows1,cols1,rows2,cols2,len1,len2);
73 |     //행렬의 뺄셈 연산을 진행하고, 반환된 행렬의 크기를 저장한다.
74 |     printf("\n\n<<두 행렬의 차(ver.일반행렬)>>\n\n");
75 |     print_arr2(sub,rows1,cols1); //뺀 행렬을 출력한다.
76 |     printf("\n\n<<두 행렬의 차(ver.희소행렬)>>\n\n");
77 |     print_arr(sub,max,3); //합한 행렬을 출력한다.
78 |

```

18, 뺄셈 연산 결과를 저장할 sub의 메모리를 할당 해주고, 뺄셈 연산을 진행한 뒤 그 결과를 마찬가지로 일반 행렬, 희소 행렬 두가지 버전으로 출력 해준다.

```

79 |     mul = creat_memory(rows1 + cols2,3); //곱셈 결과 나올 수 있는 최대 크기로 메모리 할당
80 |     sizeof_mul = arr_mul(s_matrix1,s_matrix2,mul,rows1,cols1,rows2,cols2,len1,len2);
81 |     //행렬의 곱셈 연산을 진행하고, 반환된 행렬의 크기를 저장한다.
82 |     printf("\n\n<<두 행렬의 곱(ver.일반행렬)>>\n\n");
83 |     print_arr2(mul,rows1,cols2); //곱해진 행렬을 출력한다.
84 |     printf("\n\n<<두 행렬의 곱(ver.희소행렬)>>\n\n");
85 |     print_arr(mul,rows1+cols2,3); //합한 행렬을 출력한다.
86 |

```

19, 곱셈 연산 결과를 저장할 mul의 메모리를 할당 해주고, 곱셈 연산을 진행한 뒤 그 결과를 마찬가지로 일반 행렬, 희소 행렬 두가지 버전으로 출력 해준다.

```

87 |     fclose(fp); //파일 닫기
88 |     delete_memory(arr1,rows1);
89 |     delete_memory(arr2,rows2);
90 |     delete_memory(s_matrix1,len1);
91 |     delete_memory(s_matrix2,len2);
92 |     delete_memory(sum,max);
93 |     delete_memory(sub,max);
94 |     delete_memory(mul,rows1); //메모리 반납
95 |     return 0;
96 | }
97 |

```

20, 열어 두었던 파일을 닫고, 사용한 모든 메모리를 반납한 뒤 0을 반환함으로써 main함수를 종료시킨다.

1.4 실행창

```
C:\Windows\system32\cmd.exe
<<첫번째 행렬(ver. 일반행렬)>>
1 2 3 4
3 4 1 2
2 3 4 5
1 2 3 4

<<첫번째 행렬(ver. 희소행렬)>>
0 0 1
0 1 2
0 2 3
0 3 4
1 0 3
1 1 4
1 2 1
1 3 2
2 0 2
2 1 3
2 2 4
2 3 5
3 0 1
3 1 2
3 2 3
3 3 4
```

<사진 1. 첫번째 행렬>

```
C:\Windows\system32\cmd.exe
<<두번째 행렬(ver. 일반행렬)>>
5 6 1 3
3 4 5 6
7 8 3 3
1 4 5 6

<<두번째 행렬(ver. 희소행렬)>>
0 0 5
0 1 6
0 2 1
0 3 3
1 0 3
1 1 4
1 2 5
1 3 6
2 0 7
2 1 8
2 2 3
2 3 3
3 0 1
3 1 4
3 2 5
3 3 6
```

<사진 2. 두번째 행렬>

```
선택 C:\Windows\system32\cmd.exe
<<두 행렬의 합(ver. 일반행렬)>>
6 8 4 7
6 8 6 8
9 11 7 8
2 6 8 10

<<두 행렬의 합(ver. 희소행렬)>>
0 0 6
0 1 8
0 2 4
0 3 7
1 0 6
1 1 8
1 2 6
1 3 8
2 0 9
2 1 11
2 2 7
2 3 8
3 0 2
3 1 6
3 2 8
3 3 10
```

<사진 3. 행렬의 합>

```
선택 C:\Windows\system32\cmd.exe
<<두 행렬의 차(ver. 일반행렬)>>
-4 -4 2 1
0 0 -4 -4
-5 -5 1 2
0 -2 -2 -2

<<두 행렬의 차(ver. 희소행렬)>>
0 0 -4
0 1 -4
0 2 2
0 3 1
1 2 -4
1 3 -4
2 0 -5
2 1 -5
2 2 1
2 3 2
3 1 -3
3 2 -5
3 3 -5
0 0 0
0 0 0
0 0 0
```

<사진 4. 행렬의 차>

```
선택 C:\Windows\system32\cmd.exe
<<두 행렬의 곱(ver. 일반행렬)>>
36 54 40 48
36 50 36 48
52 76 54 66
36 54 40 48

<<두 행렬의 곱(ver. 희소행렬)>>
0 0 36
0 1 54
0 2 40
0 3 48
1 0 36
1 1 50
1 2 36
1 3 48
2 0 52
2 1 76
2 2 54
2 3 66
3 0 36
3 1 54
3 2 40
3 3 48
계속하려면 아무 키나 누르십시오 . . .
```

<사진 5. 행렬의 곱>

1.5 느낀점

이 과제가 일반 행렬을 사용하는 것에서 희소 행렬을 사용하는 것으로 바뀌기 전, 이 코드를 일반 행렬을 사용 하여 더하기, 빼기, 곱하기까지 모두 작성 했습니다. 하지만 저번주 수업시간 중 희소 행렬을 이용하여 작성하라고 하신 뒤 뭐 일반 행렬로 다 작성 해 봤으니까 조금만 건들면 어렵지 않게 금방 하겠지 라고 생각 했었습니다. 하지만 희소 행렬 이라는 것을 처음 접해보고, 희소 행렬의 연산이 어떻게 이루어 지는지 사전 지식이 하나도 없어 희소 행렬에 대한 알고리즘을 작성하는데 엄청 많은 시간을 드렸습니다. arr_add()함수 하나를 작성하는데 알고리즘을 5번정도 갈아 엮으면서 희소 행렬의 연산 알고리즘을 작성 했습니다. 물론 arr_mul()함수의 경우 정말 너무 어려워 이것 저것 해보다 보니 얼떨결에 맞아 떨어져 된 느낌이 정말 강해 내 코드를 설명하는데도 나도 잘 모르고 무슨 말을 하고 있는지 잘 몰랐습니다. 다음 번에 한번 더 희소 행렬을 사용 해 알고리즘을 작성해야 하는 문제들이 생기면, 시간의 여유를 갖고 맞아 떨어져서 실행되는게 아닌 내가 다 이해하고 작성할 수 있도록 하고 싶습니다.

3. 느낀점

교수님께서 회소 행렬에 대해 처음 말씀 하셨을 때, 구글, 네이버 등 포털 사이트에서 회소 행렬에 대해 검색 해 보았을 때 모두 회소 행렬에 대한 알고리즘을 사용 하는 이유는 0을 모두 포함해 저장 하면 메모리 낭비가 너무 심해 0에 대한 데이터는 모두 제거하고 메모리를 아끼는 것이 이 알고리즘의 장점이라고 들었습니다.

하지만 저의 코드는 메모리를 절약하기는 커녕 buf라는 임시 저장 위치를 따로 만들어 메모리를 더 많이 사용하는 프로그램이 된 것 같습니다. 다음 번엔 단순히 문제를 해결하는 데서 끝내지 않고 어떻게 하면 메모리를 더 절약하고, 최적화 된 프로그램을 작성 할 수 있을지 까지 생각하여 코드를 작성 하고 싶습니다.