실습과제 2.2

[과제 설명]

실습과제 2.1에서 수행공간을 고려했다면, 2.2에서는 수행시간도 고려한 알고리즘을 이용해 코드를 구현한다.
각 데이터의 행의 빈도수를 row에 저장하고 나서, 전치행렬의 데이터(S_b)에 들어갈 행의 위치 정보(행의 offset)를 미리 계산해서 pos에 넣는다.

행의 빈도수를 구할 때 쓰는 row 행렬을 초기화할 때는 여러 방법이 있다. 먼저, 사용자 정의 함수에서 동적 할당을 받을 때 calloc을 사용해서 선언과 동시에 초기화하는 방법이 있고(malloc으로 선언하면 초기화하지 않는 한, 더미 값이 들어가있다), 교안에 있는 것처럼 반복문을 이용해서 0으로 초기화하는 방법도 있다. 그것 외에도 memset을 이용해서 일괄적으로 초기화하는 방법이 있는데, 인수로 (초기화할 배열, 초기화할 값, 초기화할 범위)가 들어간다. 이 때, 초기화할 범위는 동적할당을 할 때처럼 범위의 크기로 넣어야 한다.

[코드]

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_ELEMENTS 8
//0이 아닌 원소의 개수

typedef struct Element
{
  int row;
  int col;
  int value; //열, 행. 값
}Element;

Element* Transpose_Triple1(Element S_a[]) {
  //static Element S_b[MAX_ELEMENTS];
  //동적 할당 대신 static 변수를 만들 때
  Element* S_b = (Element*)malloc(sizeof(Element) * MAX_ELEMENTS);
  //함수가 끝나고도 유지되도록 동적 할당을 이용

  int num = S_a[0].value; //희소 행렬 중 0이 아닌 값의 개수
  S_b[0].row = S_a[0].col;
  S_b[0].col = S_a[0].row; //행과 열의 크기를 바꿔서 대입
  S_b[0].value = num;
```

```
  if (num > 0) { //0이 아닌 값의 개수가 1개라도 있을 대 실행
    int* row = (int*)malloc(sizeof(int) * S_a[0].col);
    int* pos = (int*)malloc(sizeof(int) * S_a[0].col);
    memset(row, 0, sizeof(int) * S_a[0].col); //0으로 일괄 초기화

    for (int i = 1; i <= num; i++) {
      row[S_a[i].col]++;
    } //1번째부터 num번째까지 반복하면서 각 행의 빈도수를 구한다.

    pos[0] = 1;

    for (int i = 1; i < S_a[0].col; i++) {
      pos[i] = pos[i - 1] + row[i - 1];
    }//변환될 데이터를 기준으로 행의 절대 위치를 미리 구해준다.

    int abs_pos;

    for (int i = 1; i <= num; i++) {
      abs_pos = pos[S_a[i].col];
      S_b[abs_pos].row = S_a[i].col;
      S_b[abs_pos].col = S_a[i].row;
      S_b[abs_pos].value = S_a[i].value;
      pos[S_a[i].col]++;
      //한 행의 데이터가 여러 개일 경우 덮어쓰기 되지 않기 위해 하나를 하고 나면 증가시킨다.
    }
  }

  return S_b;
  //배열을 넘기는 것이기 때문에 값을 넘기는 것이 아니라 시작 주소를 넘기는 것이다.
}

void Print_Sparse_Mat(Element arr[]) {
  int cnt = 1;

  for (int i = 0; i < arr[0].row; i++) {
    for (int j = 0; j < arr[0].col; j++) {
      if (i == arr[cnt].row && j == arr[cnt].col) {
        printf("%d ", arr[cnt].value);
        cnt++;
      }
      else
        printf("0 "); //해당하는 값이 없으면 0을 대신 출력한다.
    }
    puts(""); //행이 바뀔 때마다 줄바꿈
  }
}

int main(void) {
  Element Sparse_A[MAX_ELEMENTS] = { {6,6,7},
    {0,2,6},
    {1,0,5},
    {1,4,7},
    {2,3,3},
    {4,0,8},
    {4,1,9},
    {5,3,2} };
```

```
  Print_Sparse_Mat(Sparse_A); //원본 행렬 출력
  puts("");  //줄바꿈
  Element* Sparse_B = Transpose_Triple1(Sparse_A); //전치행렬로 데이터를 변환
  Print_Sparse_Mat(Sparse_B); //전치행렬 출력
}
```

[과제에 대한 고찰]

실습과제 2.1도 공간 효율성을 고려하니까 평소에 하는 스타일의 코드보다 훨씬 복잡해졌는데, 시간 효율성까지 고려하다보니 실습과제 2.1의 코드보다도 더 복잡해진 것 같다. 파이썬에서 정렬을 배울 때 코드 자체는 버블 정렬이 가장 이해하기 쉽고 짧았지만, 퀵 정렬이 가장 효율적이라고 배웠는데 그런 것과 비슷한 것 같다. 사용할 수 있는 자원이 한정적이면 오히려 코드가 복잡해지는 것을 실감했던 것 같다.

동적 할당을 한 변수를 초기화할 때 처음에 arr = {0, }; 로 해보려고 했지만, 그렇게 하면 오류가 나면서 컴파일이 되지 않았다. 이때 반복문을 사용해서 0으로 초기화할 수도 있었지만 그렇게 하면 오히려 시간효율성이 필요 이상으로 늘어날 수 있어서 다른 방법을 찾았다 (calloc도 있었지만, 익숙한 함수가 아닌데다가 잘 쓰지 않는 것 같았다). memset으로 초기화할 경우, 0번째부터 원하는 부분까지만 초기화할 수 있지만 대신 0이 아닌 다른 수로 초기화할 때도 쓸 수 있다는 장점이 있는 것 같다.