Homework2 Technical Report

1829008 김민영

Homework 2-1 : 희소 행렬 전치

1. 목적 : Sparse Matrix의 경우 모든 요소를 배열에 저장하는 것은 메모리 공간이 많이 낭비된다.

따라서 오직 0이 아닌 요소만을 저장하는 방식으로 문제를 해결할 것이다.

이렇게 저장한 행렬을 바탕으로 transpose를 하고 예제를 출력할 것이다.

2. 용어 설명

Sparse Matrix(희소행렬) : 0을 많이 포함하는 행렬

전치 행렬 : 행과 열을 교환하여 얻은 행렬이다. 즉 대각선을 축으로 반으로 접은 행렬이다. mxn의 행렬의 전치행렬은 nxm 행렬이 나온다. 전치행렬은 m^T로 표현한다.

3. 코드 설명

element 구조체 – (행, 열, 값)을 저장한다.

SparseMatrix 구조체 – element들과 행렬의 총 행(rows), 총 열(cols), terms를 저장한다.

sparse_matrix_T 함수

행렬 a를 인자로 받아, 행렬의 전치 수행하고 행렬 b를 반환하는 함수이다.

a를 전치한 행렬이 b이므로, b의 rows와 cols는 a의 rows, cols와 반대이다. terms는 a와 같다. terms가 0이면 전치를 수행할 필요가 없으므로 terms가 0보다 클 때, 행렬의 전치를 for문 두개 돌리면서 수행한다. b의 인덱스는 0부터 시작해서 하나씩 늘리는 방식으로 저장한다. b는 row, col 순으로 저장이되므로, 첫번째 for문은 rows에 대해 for문을 돌린다.(i) 두번째 for문은 terms의 개수에 대해 for문을 돌린다.(j) 이때 만약 a의 j번째 term의 col이 b의 row와 같게되면 b의 인덱스에 값들을 저장한다. Row는 a의 col, col은 a의 row과 같다. 또한 값을 같으므로 a의 값을 저장해준뒤, 다음 비교를 위해 b의 인덱스를 1 증가시킨다. 이런방식으로 모든 반복문이 수행되면, 모든 요소들이 전치되어 저장된다.

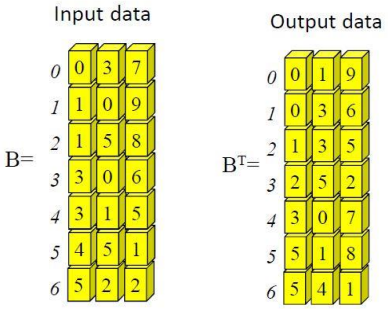
main 함수

1) 전치를 수행할 SparseMatrix B를 예시로 넣어준다.

2) sparse_matrix_T(B) 함수를 통해 전치행렬을 구해준다.

3) 이를 확인하기위해 예제를 출력해준다.

4. 실행 결과




```
***B***
(0,3,7)
(1,0,9)
(1,5,8)
(3,0,6)
(3,1,5)
(4,5,1)
(5,2,2)
==check==
000700
900008
000000
650000
000001
002000
=========

***B_T***
(0,1,9)
(0,3,6)
(1,3,5)
(2,5,2)
(3,0,7)
(5,1,8)
(5,4,1)
===check===
090600
000500
000002
700000
000000
080010
=========계속하려면 아무 키나 누르십시오 . . .
```

5.분석

Sparse Matrix의 경우 0이 굉장히 많기 때문에 모든 요소를 배열에 저장하는 것은 메모리 공간이 많이 낭비된다. 따라서 원래 기존의 방식(강의피피티 (1)번방식) 보다는 이렇게 오직 0이 아닌 요소만을 저장하는 방식으로 문제를 해결하는 것이 메모리 공간 관리에 효율적이다.

Codes

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_TERMS 101

typedef struct {
        int row;
        int col;
        int value;
} element;

typedef struct SparseMatrix {
        element data[MAX_TERMS];
        int rows;
        int cols;
        int terms;
}SparseMatrix;

SparseMatrix sparse_matrix_T(SparseMatrix a) {
        SparseMatrix b;
        int bind;
        b.rows = a.cols;
```

```c
int main(void) {
	//Add B as an input.

	SparseMatrix B ={{{0,3,7},{1,0,9},{1,5,8},{3,0,6},{3,1,5},{4,5,1},{5,2,2}}, 6,6,7};

	//Perform the transpose operation
	SparseMatrix B_T;
	B_T = sparse_matrix_T(B);

	//Print out B and B^T in a dense matrix from to check
	//whether the operation works correctly.

	printf("***B*** \n");

	for (int i = 0; i < B.terms; i++) {
		printf("(%d,%d,%d) \n", B.data[i].row, B.data[i].col, B.data[i].value);
	}
	printf("==check==\n");
	int t = 0;
	for (int i = 0; i < B.rows; i++) {
		for (int j = 0; j < B.cols; j++) {
			if (B.data[t].row == i && B.data[t].col == j) {
				printf("%d", B.data[t].value);
				t++;
			}
			else {
				printf("0");
			}
		}
		printf("\n");
	}
	printf("=========\n");
```

Homework 2-2 : 3차원 배열 메모리 동적할당 덧셈

1. 목적

메모리를 효율적으로 관리하기 위해 static memory allocation 보다는 dynamic memory allocation 방식을 활용할 것이다. 이 방식으로 3차원 행렬을 double 타입으로 A,B를 만든다.이후 A와 B를 더하는 연산을 수행하고, A와 B를 deallocate 한다.

2. 용어 설명

static memory allocation : 프로그램이 시작되기 전에 메모리 사이즈가 고정되어있고 실행하는 동안 바꿀 수 없다. ex) int buffer[100];

dynamic memory allocation(메모리동적할당) : 프로그램을 실행하는 동안 메모리를 할당한다. 사용하고 싶은 만큼 메모리를 할당할 수 있다.

3. 코드 설명

1) Mem_alloc_3D() 함수

메모리 할당 : 제일 먼저, malloc함수를 이용해 너비만큼 동적 할당을 하고, 높이, 두께 순으로 동적할당을 한다.

정의 : 각각에 값을 넣어준다. 그 값은 i*H*M + j*M+k로 해서 하나씩 증가하는 값으로 저장한다.

출력: 값을 확인하기 위해 출력을 한다.

이후 arr을 반환한다.

2) Addtion_3D 함수 :

For문 3개를 돌리면서 각 너비,높이,두께마다 A와 b를 더한 c를 출력해준다.

3) Main 함수

mem_alloc_3D() 함수를 이용해서 A, B를 정의한다.

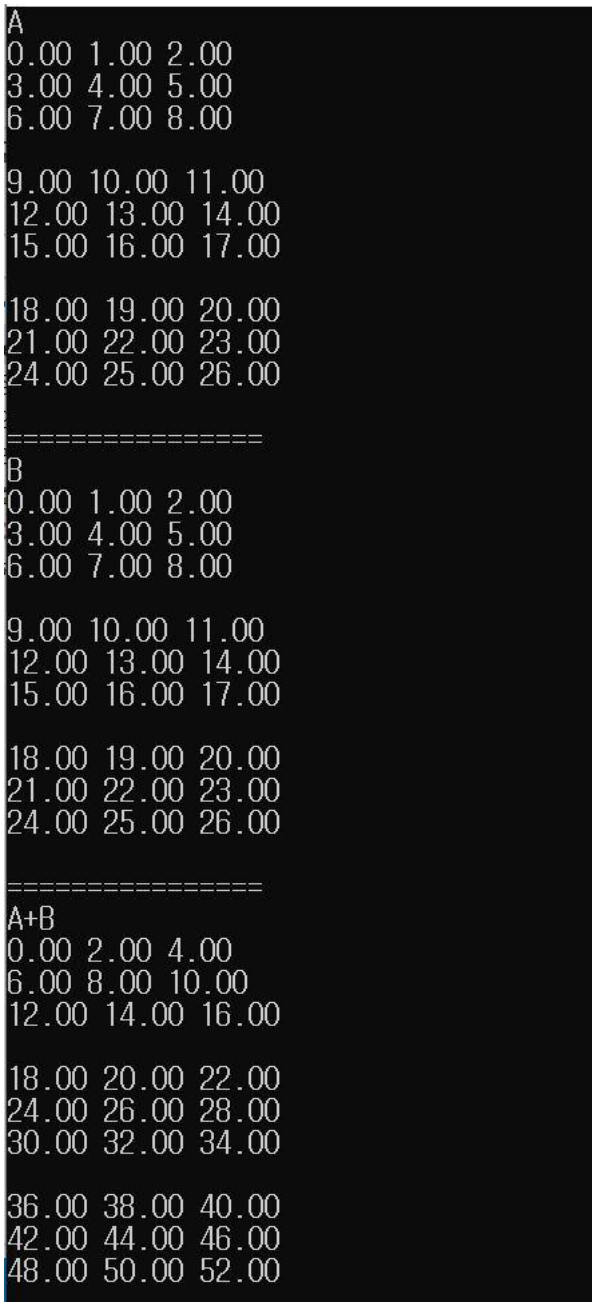
Addition_3D(A,B) 함수를 이용해서 A와 B를 더한다.

A와 B를 Deallocate한다. Allocate 할때와는 반대로 두께-> 높이-> 너비순으로 Deallocate한다. Deallocate 할때 쓰는 함수는 free 이다.

4. 실행결과

위에서 언급했듯이, A,B에는 0~26까지 1씩증가시키면서 값을 저장했다.

이 둘을 addition_3D함수를 통해 덧셈을 진행하면 0-52까지 잘 출력이 된 것을 확인할 수 있다.



5. 분석

Static memory allocation에서보다 우리가 한 dynamic allocation 이 더 메모리 공간에 효율적이었다.

Codes

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

#define W 3
#define H 3
#define M 3

double *** mem_alloc_3D_double() {
        //Memory Allocation
        double ***arr = (double ***)malloc(sizeof(double**)*W);
        for (int i = 0; i < W; i++) {
                arr[i] = (double **)malloc(sizeof(double*)*H);
                for (int j = 0; j < H; j++) {
                        arr[i][j] = (double*)malloc(sizeof(double)*M);
                }
        }
        //Define
        for (int i = 0; i < W; i++) {
                for (int j = 0; j < H; j++) {
                        for (int k = 0; k < M; k++) {
                                *(*(*(arr + i) + j) + k) = i * H*M + j * M + k;
                        }
                }
        }
        //Print
        for (int i = 0; i < W; i++) {
                for (int j = 0; j < H; j++) {
                        for (int k = 0; k < M; k++) {
```

```c
int main(void) {
	//Define two matrices A and B using 'mem_alloc_3D_double';
	printf("A\n");
	double ***A = mem_alloc_3D_double();
	printf("================\n");
	printf("B\n");
	double ***B = mem_alloc_3D_double();
	printf("================\n");
	//Perform addition of two matrices using 'addtion_3D'
	printf("A+B\n");
	addition_3D(A, B);
	printf("================\n");
	//Deallocate A and B
	for (int i = 0; i < W; i++) {
		for (int j = 0; j < H; j++) {
```