Introduction to Computer and Lab

Homework #9

Due date: Jun 12, 2016

학번: 201404051

이름: 정 용 석

1. 행렬의 합을 출력

1.1 Solution

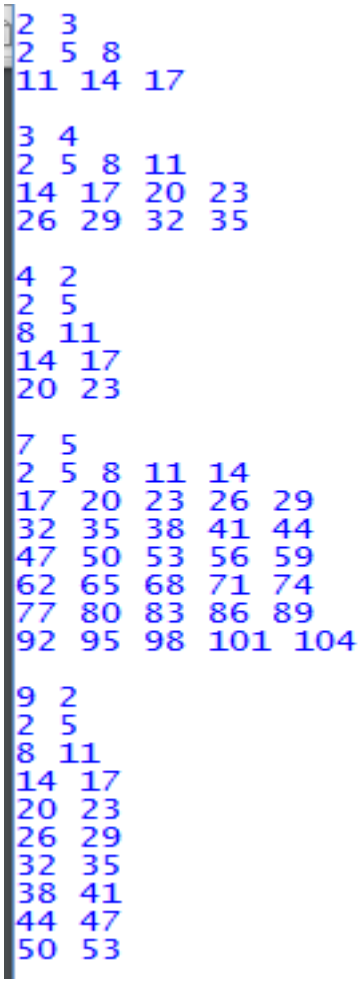
일단 모든 문제의 행렬의 가로 세로의 최대 크기는 10 이기 때문에 MAX_SIZE 를 10 으로 두고, 최대 크기를 가지는 2 차원 행렬 2 개를 만들었다. 그리고 첫 번째 행렬에는 1 부터 m*n 까지 값을 순차적으로 가지도록 대입하고, 두 번째 행렬은 2 씩 증가하는 홀 수의 값들을 순차적으로 대입하였다. 마지막으로 출력과 동시에 첫 번째와 두 번째의 행렬 각각의 행과 열에 원소들을 더함과 동시에 출력했다.

1.2. Source code

```
void arraySum() // 행렬의 합 출력
{
	int m, n; //m: 행, n: 열
	int setA[MAX_SIZE][MAX_SIZE]; //행렬 A
	int setB[MAX_SIZE][MAX_SIZE]; //행렬 B
	int value = 1;
	int i, j;

	scanf("%d %d", &m, &n);

	//행렬 A에는 1~m*n을 차례대로 대입
	for (i = 0; i < m; i++)
		for (j = 0; j < n; j++)
			setA[i][j] = value++;

	value = 1;
	//행렬 B에는 1부터 +2 증가하는 값을 차례대로 대입
	for (i = 0; i < m; i++)
		for (j = 0; j < n; j++) {
			setB[i][j] = value;
			value += 2;
		}
	//행렬A+행렬B 출력
	for (i = 0; i < m; i++) {
		for (j = 0; j < n; j++)
```

```
                printf("%d ", setA[i][j] + setB[i][j]);
            printf("₩n");
        }
}
```

1.3. Result (snapshot)



2. 행렬의 곱을 출력

2.1. Solution

행렬의 곱을 일단 보면 곱해지는 행렬의 행의 각각의 원소와 곱하려는 열의 각각의 원소를 곱하여 모두 더한 값이 결과 값의 원소가 된다. 따라서 결과는 항상 행렬의 행 M X M 의 모습을 가지게 된다. 따라서 이러한 연산을 위해서는 3 중 반복 문을 사용하면 된다. 행렬 A 의 첫 번째 행의 원소들을 행렬 B 의 열의 원소들과 곱하여 합한 값을 출력하는 방식으로 코드를 보면 변수 k 를 추가하여 이를 가능케 하였다.

2.2. Source code

```
void arrayMul()  // 행렬의 곱을 출력
{
	int m, n; //m: 행, n: 열
	int set1[MAX_SIZE][MAX_SIZE]; //행렬 A
	int set2[MAX_SIZE][MAX_SIZE]; //행렬 B
	int i, j, k;
	int value = 1;

	scanf("%d %d", &m, &n);

	// m X n 행렬
	for (i = 0; i < m; i++)
		for (j = 0; j < n; j++)
			set1[i][j] = value++;

	value = 1;

	// n X m 행렬
	for (i = 0; i < n; i++) {
		for (j = 0; j < m; j++)
			set2[i][j] = value++;
	}

	value = 0;
	k = 0;

	// 행렬 A X 행렬 B 출력
	for (i = 0; i < m; i++) {
		for (j = 0; j < m; j++) {
			while (k != n) {
				//행렬 A의 각각의 행의 원소 X 행렬 B의 각각의 열의 원소
				value += set1[i][k] * set2[k][j];
				k++;
			}
			//출력 및 변수 초기화
			printf("%d ", value);
			value = 0;
			k = 0;
		}
		printf("₩n");
	}
}
```

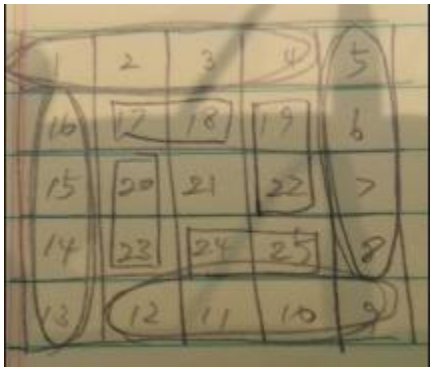
2.3. Result (snapshot)

```
2 3
22 28
49 64

3 4
70 80 90
158 184 210
246 288 330

4 5
175 190 205 220
400 440 480 520
625 690 755 820
850 940 1030 1120

6 7
700 728 756 784 812 840
1631 1708 1785 1862 1939 2016
2562 2688 2814 2940 3066 3192
3493 3668 3843 4018 4193 4368
4424 4648 4872 5096 5320 5544
5355 5628 5901 6174 6447 6720

5 4
110 120 130 140 150
246 272 298 324 350
382 424 466 508 550
518 576 634 692 750
654 728 802 876 950

Press any key to continue . . .
```

3. 나선형 모양 행렬

3.1. Solution

정말 다양한 방법이 있고, 실제로 다양한 방법으로 이 문제를 풀 수 있다. 여러가지를 시도하던 중 그나마 가장 간단하고 정확한 방법으로 구현하였다. 일단 정사각형 행렬이라는 특성 상 내가 발견한 패턴은 이것이다.



옆의 그림과 같이 일단 원소의 출력은 4 번을 반복하게 되는데, 한 바퀴를 돌고 나면 출력 해야하는 원소의 개수 또한 - 2 가 된다. 이는 어떠한 크기의 배열에서도 동일하다. 따라서 반복 횟수는 일단 사이즈 m – 1 이 된다. 그리고 출력이 진행되는 동안은 행 혹은 열의 원소의 값은 고정이다. 따라서 begin, end 의 변수와 x, y 의 변수를 따로 만들어 begin 과 end 는 고정된 행 또는 열의 원소 값을 가지게 하고, x, y 같은 경우는 행과 열의 원소 이동을 담당한다. 또한 한 바퀴를 돌고 나면 begin 과 end 는 각각 1 식 증가, 감소하게 되고 말했듯이 반복 횟수 또한 -2 가 된다. 또 하나

생각해야 될 것이 m 의 홀수 짝수 여분이다. 홀수일 때는 반복이 0 을 마지막으로 끝나게 되는데 이 때는 출력하고 반복 문을 탈출 해야하고, 짝수일 때는 마지막 원소, 즉 가운데 홀로 있는 원소가 없기 때문에 그냥 나오면 된다.

3.2. Source code

```
void spiralArray()  // 나선형 모양 행렬
{
	int m; //m: 열과 행
	int i, j;
	int set[MAX_SIZE][MAX_SIZE];
	int value = 1;
	int x, y;	//x: 열 이동, y: 행 이동
	int begin;	//시작점
	int end;	//끝점
	int repeat;	//반복 횟수

	scanf("%d", &m);

	//행렬 생성
	for (i = 0; i < m; i++)
		for (j = 0; j < m; j++)
			set[i][j] = value++;
	//변수들 초기화
	i = 0;
	x = 0;
	y = 0;
	begin = 0;
	end = m - 1;
	//행의 길이 보다 1 작게 반복
	repeat = m - 1;

	while (1) {
		//반복 횟수 0 혹은 -1 이면
		//홀수 일시, 마지막 원소 출력 후 종료
		//짝수 일시, 그냥 종료
		if (repeat <= 0) {
			if (m % 2 == 0) break;
			else {
				printf("%d ", set[y][x]);
				break;
			}
		}

		for (i = 0; i < repeat; i++) //행렬의 좌상->우상
			printf("%d ", set[begin][x++]);
		for (i = 0; i < repeat; i++) //행렬의 우상->우하
			printf("%d ", set[y++][end]);
		for (i = 0; i < repeat; i++) //행렬의 우하->좌하
			printf("%d ", set[end][x--]);
		for (i = 0; i < repeat; i++) //행렬의 좌하->좌상
```

```
                printf("%d ", set[y--][begin]);
            //변수들 증가 및 감소, 초기화
            y++;
            x++;
            begin++;
            end--;
            repeat -= 2;
        }
        printf("₩n");
}
```

3.3. Result (snapshot)

```
2
1 2 4 3

3
1 2 3 6 9 8 7 4 5

4
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

5
1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 6 7 8 9 14 19 18 17 12 13

6
1 2 3 4 5 6 12 18 24 30 36 35 34 33 32 31 25 19 13 7 8 9 10 11 17 23 29 28 27 26 20 14 15 16 22 21

Press any key to continue . . .
```

4. 가능한 Path 의 개수

4.1. Solution

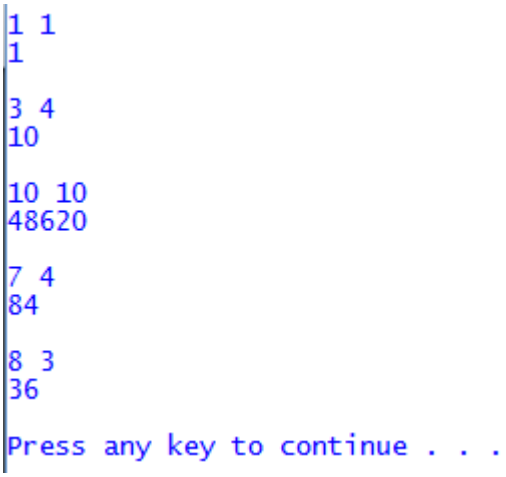
단순하게 행렬의 특성을 이용하여 풀 수도 있지만, 보자마자 든 생각이 재귀의 활용이었고, 그 이외의 방법은 도무지 단순하게 풀 수 있을 것 같지 않았다. 따라서 재귀 함수를 만들어서 호출하는 방식으로 구현하였다. 0 부터 시작하지 않고, 행과 열 m, n 이 주어졌을 때, 이 지점에서 출발하여 둘 중에 하나라도 배열의 한계 선을 닿게 되면 가야할 길은 1 개이다. 이를 이용하여 재귀 함수를 구현하였고 성공적이었다.

4.2. Source code

```
void totalPath()  // 가능한 Path의 개수
{
        int set[MAX_SIZE][MAX_SIZE];
        int m, n; //행: m, 열: n

        scanf("%d %d", &m, &n);
        printf("%d₩n", RecursivePath(m, n));
}
int RecursivePath(int m, int n) // Path개수를 찾기 위한 재귀함수
{
        if (m == 1 || n == 1) //행이나 열의 끝에 도달하면 리턴 1
                return 1;
        return RecursivePath(m - 1, n) + RecursivePath(m, n - 1);
```

}

4.3. Result (snapshot)



5. Longest Path 의 값 출력

5.1. Solution

실습 내용에 나와있는 힌트를 바탕으로 하면 쉽다. 일단 주어진 m 과 n 으로 2 개의 행렬 판을 만들고, 첫 번째 행렬에는 임의 값을 채워 넣는다. 그리고 나서 일단 두 번째 행렬의 가장 첫 번째 원소 [0, 0]에는 첫 번째 행렬의 값이 무조건적으로 들어간다. 그 이후, 점화 식을 코드로 구성하면 되는데, 개인적으로는 일단 첫 번째 원소 이후에 참조해야 할 원소들이 1 개 밖에 없는 행과 열을 일단 처리하고, 배열 중간에 있으면서도 위와 왼쪽의 원소들을 참조해야 할 원소들만 조건 문을 이용하여 더하고 출력하였다. 실제 점화 식을 이용하여 구한 배열은 소스 코드 안에 주석 처리해 놓았다.

5.2. Source code

```
void LongestPath()  // Longest path의 값 출력
{
	srand(1); //랜덤 시드값 1
	int set[MAX_SIZE][MAX_SIZE]; //랜덤 행렬
	int d[MAX_SIZE][MAX_SIZE];     //점화식 행렬
	int m, n; //행: m, 열: n
	int i, j;
	int add;

	scanf("%d %d", &m, &n);

	//랜덤 행렬 생성 및 출력
	for (i = 0; i < m; i++)
		for (j = 0; j < n; j++)
			set[i][j] = rand() % 29 + 1;
	/*
	for (i = 0; i < m; i++) {
```

```
			for (j = 0; j < n; j++)
				printf("%d ", set[i][j]);
			printf("₩n");
		}
		*/
		//점화식 행렬 생성
		for (i = 0; i < m; i++) {
			for (j = 0; j < n; j++) {
				if (i == 0 && j == 0) //첫 번째 [0, 0] 원소
					d[i][j] = set[i][j];
				else if (i == 0 && j != 0) {//열 번호가 0일때
					d[i][j] = d[i][j - 1] + set[i][j];
				}
				else if (i != 0 && j == 0) {//행 번호가 0일때
					d[i][j] = d[i - 1][j] + set[i][j];
				}
				else {//위와 왼쪽의 원소를 비교 큰 대상과 합
					if (d[i - 1][j] > d[i][j - 1])
						add = d[i - 1][j];
					else
						add = d[i][j - 1];
					d[i][j] = add + set[i][j];
				}
			}
		}
		/*
		printf("₩n₩n");
		//점화식 출력
		for (i = 0; i < m; i ++) {
			for (j= 0; j < n; j++)
				printf("%d ", d[i][j]);
			printf("₩n");
		}
		*/
		printf("%d₩n", d[m-1][n-1]);
}
```

5.3. Result (snapshot)

```
5 5
156

8 4
197

2 3
57

1 2
37

10 10
404

Press any key to continue . . .
```