자료구조및알고리즘 과제_3

반	1	학과	전자공학과
학번	202021025	이름	안준영

* 다음의 4개의 함수를 구현하고 구현된 코드를 한글파일에도 추가를 하시오.

```
int edge_num(int adj_matrix[MAX_SIZE][MAX_SIZE]);
void complete_graph(int adj_matrix[MAX_SIZE][MAX_SIZE]);
void check_simple_path(int adj_matrix[MAX_SIZE][MAX_SIZE], char path[]);
int get_largest_connected_component(int adj_matrix[MAX_SIZE][MAX_SIZE]);
```

* 함수 설명 및 요구 사항: 과제3은 그래프의 개념을 이해하고 활용하는 과제이며 그래프를 이용하여 4개의 함수를 구현합니다.

* 함수 “edge_num”은 배열로 표현된 무방향그래프를 입력으로 받아서 그래프에 있는 모든 간선의 수를 구해주는 함수입니다.

* 함수 “complete_graph”는 배열로 표현된 무방향그래프를 입력으로 받아서 그래프가 완전그래프(complete graph) 여부를 판별하는 함수입니다. 함수 “complete_graph”를 이용하여 주어진 그래프가 완전그래프이면 “complete graph” 라고 출력하고 완전그래프가 아니면 “Non-complete graph”라고 출력합니다.

* 함수 “check_simple_path”는 배열로 표현된 무방향그래프와 문자형 배열 path에 저장된 문자열로 표현된 경로(예: “0123”)를 입력을 받으며 그래프에서 주어진 경로가 단순경로(simple path)인지를 판별하는 함수입니다. 함수 “check_simple_path”를 이용하여 그래프에서 주어진 경로가 단순경로이면 “Yes”라고 출력을 하고 단순경로가 아니면 “No”라고 출력합니다.

* 함수 “get_largest_connected_component”는 배열로 표현된 무방향그래프를 입력을 받아서 그래프에서 가장 큰 연결요소(connected_component)를 찾아서 가장 큰 연결요소의 정점의 개수를 반환하는 함수입니다.

2. “project3_main_학번_이름.cpp” 파일로 4개 함수를 구현하고 구현한 함수 내용을 한글 파일(과제3_학번_이름.hwp)에도 복사를 합니다. “project3_main_학번_이름.cpp”와 “과제3_학번_이름.hwp”, 2개의 파일을 “과제3_제출”에 제출합니다.

3. 과제3 구현을 위한 구체적인 요구 사항은 다음과 같습니다.
. 함수 구현을 위해서 필요한 전역 및 지역 변수, 전역 및 지역 포인터, 전역 및 지역 배열를 선언하여 사용 가능.
. 구현할 함수 안에서 사용(호출)할 추가적인 함수 구현 가능.
. 4개 구현 함수의 함수 원형 수정 금지.
. main 함수에 있는 배열로 표현된 그래프의 내용(그래프의 모양과 크기) 변경될 수 있다는 가정을 함.
. 학생들이 구현한 코드에 대한 설명을 주석으로 작성.
. 구현한 코드를 한글 파일에도 추가.

1. edge_num 함수 구현

```
{
	int n = 0;
	for (int i = 0; i < 5; i++) //2차원 배열을 다루기 위한 2중 for문
	{
		for (int j = 0; j < 5; j++)
		{
			if (adj_matrix[i][j] == 1)
				n++; // [i][j]요소가 1이면, 즉 간선이 있으면 n에 1을 더함.
		}
	}
	return n / 2; // 무방향 그래프라서, symmetric matrix로 구성되므로 n은 항상 2의 배수이다. 따라서 2로 나누어주면 간선의 개수이다.
}
```

2. complete_graph 함수 구현

```
{
	int n = edge_num(adj_matrix);
	if (n == MAX_SIZE*(MAX_SIZE-1)/2) // 완전 그래프라면, 정점의 개수 n에 대하여 (간선의 수) = n*(n-1)/2를 만족해야한다. n = 5이므로 (간선의 수) = 10이다.
		printf("complete graph\n");
	else
		printf("Non-complete graph\n");
}
```

3. check_simple_path 함수 구현

```
{
	int size = sizeof(path) / sizeof(char); // 배열 path의 길이
	int x[sizeof(path) / sizeof(char)]; // path 배열의 요소를 adj_matrix의 인덱스에 사용하고자 path의 요소를 int형 자료로 저장할 배열 x
	int n = 0; // 경로가 이어지는지를 확인한 결과를 저장할 변수
	int o = 0; // 경로 중 중복 요소가 있는지를 확인한 결과를 저장할 변수

	for (int i = 0; i < size; i++)
	{
		x[i] = path[i] - 48; // x에 path의 값을 각각 대입시키기. path는 char형 변수이므로 ASCII 코드를 고려하여 -48을 해주어야 한다.
	}
	for (int i = 0; i < size-1; i++) // 경로의 정점이 서로 다 이어져 있는지 확인
	{
		if (adj_matrix[x[i]][x[i+1]] == 1)
		{
			n++; // 이어짐
		}
	}

	for (int i = 0; i < size; i++) // 단순 경로라면 경로 내의 중복되는 노드가 없어야 한다. 따라서 아래의 for문으로 배열 내 중복요소가 있는지를 확인한다.
	{
		for (int j = i+1; j < size; j++)
		{
			if (x[i] == x[j])
				o++; // 중복되는 요소가 있다면 o의 값이 증가하여 0이 아니게 된다.
		}
	}
	if (n == size - 1 && o == 0) // 단순 경로인 경우
		printf("Yes\n");
	else
		printf("No\n"); // 입력된 경로의 노드가 그래프 내에서 경로를 구성하지 않거나 중복되는 요소가 있어서 단순 경로가 아닌 경우
}
```

4. get_largest_connected_component 함수 구현

```
{
	int m[MAX_SIZE] = {0}; //각 연결요소의 노드 개수를 저장할 배열 m
	int l = 0; //m의 인덱스를 지정할 변수

	for (int i = 0; i < MAX_SIZE; i++) // adj_matrix는 symmetry matrix이므로, 대칭되는 부분 중 한 부분만 다룬다. 마치 DFS처럼 이어진 부분을 인덱스 순대로 따라가지만 여기서는 단순히 연속되는
                                                // 두 정점이 연결이 되어 있는지만 판단하면 되므로 돌아오는 과정은 필요없다.
	{
		for (int j = i + 1; j < MAX_SIZE; j++)
		{
			if (adj_matrix[i][j]) // 정점 i와 정점j가 연결되어 있다면
			{
				m[l]++; // m에 +1
				j = MAX_SIZE; // j로 제어되는 for문 탈출
			}
			else if (!adj_matrix[i][j] && j == MAX_SIZE - 1) // i정점에 대하여 모든 j 정점이 연결되어 있지 않은 경우 -> 연결요소의 끝에 옴.
			{
				l++; // 다음 연결요소로 넘어감
			}
		}
	}

	int max = m[0];
	for (int i = 0; i < MAX_SIZE; i++)
		if (m[i] > max)
			max = m[i]; // m중 최대값 결정

	return max+1; //+1을 해주어야 정점의 개수 (n개의 간선으로 단순 경로로 타고가면 정점은 n+1개이므로)
}
```

※ 주의사항

1. 과제는 반드시 본인이 작성하여 제출.
2. 모든 제출 과제에 대해서 과제 카피 체크 프로그램으로 확인합니다. 타인의 과제를 카피하면 두 과제 모두 미제출 처리.
3. 제출 기간을 넘기면 감점을 하며 제출 기한에서 1주일이 지난 후에는 과제를 받지 않음.