**그래프 내 릴레이션 찾기 201611229 장정아**

**1. 2차원 배열을 활용하는 방법**

**1.1 코드**

#include <iostream>

#include <time.h>

#include <ctime>

#include <stack>

using namespace std;

int main(){

stack<int> s;

clock\_t start, finish; double duration;

start = clock();

srand((unsigned)time(NULL));

int size = 0;

cout << "그래프의 크기를 입력하세요 : ";

cin >> size;

int \*\*arr = new int\*[size];

for (int i = 0; i < size; i++){

arr[i] = new int[size];

}

for (int i = 0; i < size; i++){

for (int j = i; j < size; j++){

arr[i][j] = rand() % 2;

arr[j][i] = arr[i][j];

}arr[i][i] = 0;

} // 그래프 완성

for (int i = 0; i < size; i++){

for (int j = 0; j < size; j++){

cout << arr[i][j];

} cout << endl;

}

int search = 0; // search할 정점

int searchcheck = -1; //search를 더 해야할지 여부 체크

int temp = 1; //visited에 저장할 변수

int\* mark = new int[size];

for (int i = 0; i < size; i++){

mark[i] = 0;

}

while (true){

while (true){

mark[search] = temp;

int depthcheck = -1; // 더 깊이 들어갈지 말지 결정하는 변수, 깊이체크 아님x

for (int i = 0; i < size; i++){

if (arr[search][i] == 1 && mark[i] == 0){

depthcheck = 0;

s.push(i);

}

}

if (depthcheck == -1){

s.pop();

if (s.empty())

break;

else

search = s.top();

} else

search = s.top();

}

cout << endl << "그래프 내 릴레이션 : " << endl;

for (int i = 0; i < size; i++){

if (mark[i] == temp)

cout << i << " ";

} cout << endl;

temp++;

for (int i = 0; i < size; i++){

if (mark[i] == 0){

search = i;

searchcheck = 1;

}

} if (searchcheck == -1)

break;

}

finish = clock();

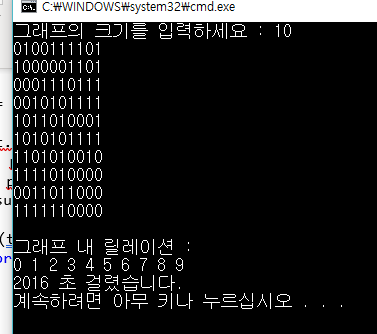
duration = (double)(finish - start);

cout << duration << " 초 걸렸습니다." << endl;

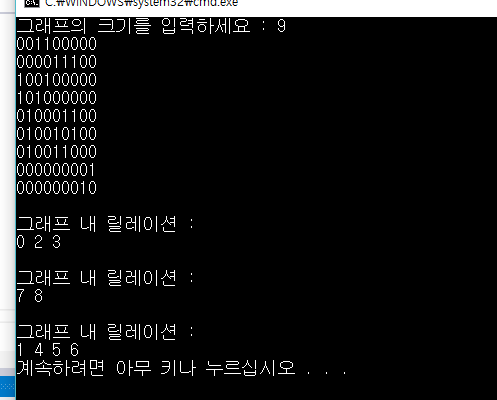
}

**1.2 실행 결과**

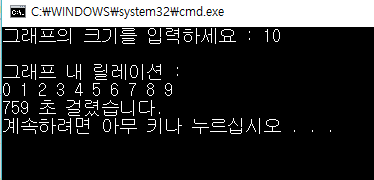
* 임의로 size 정보를 주었을 때

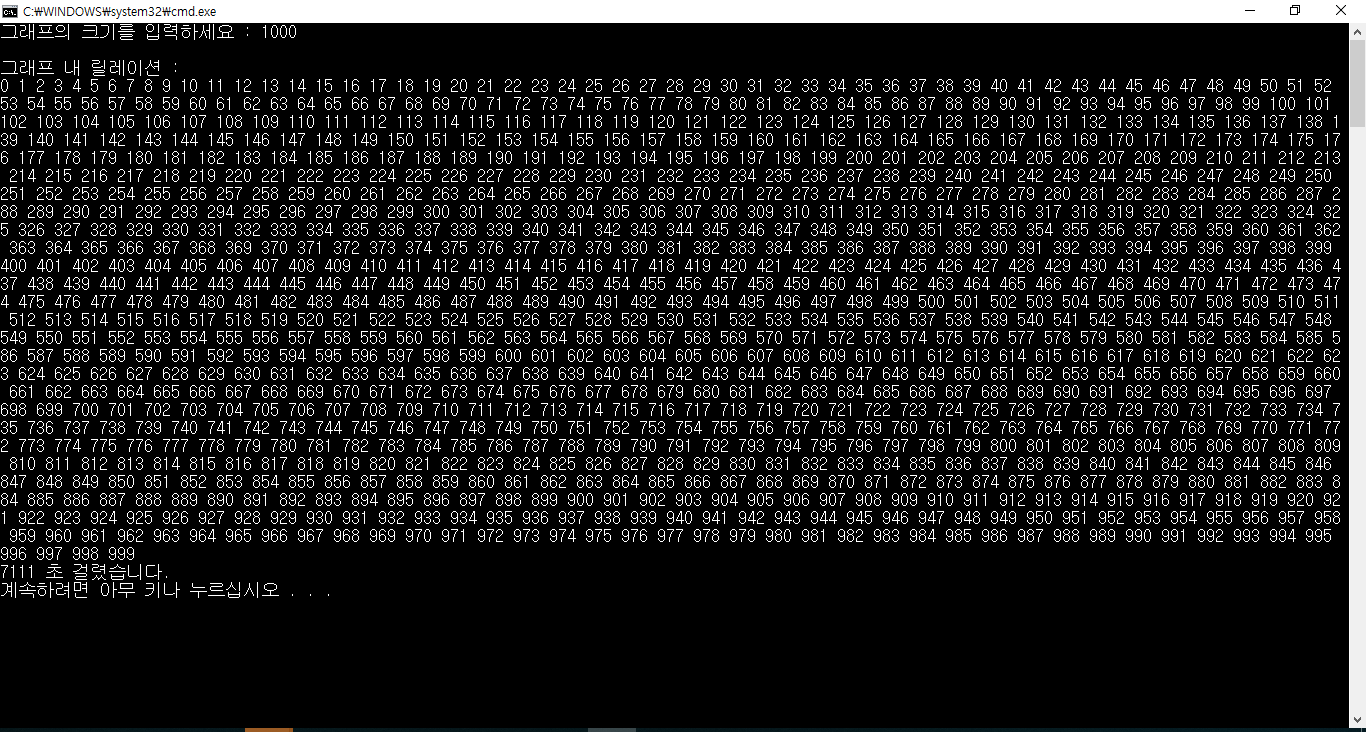


* 성능 분석을 위해 릴레이션이 여러 개 존재하는 그래프를 넣었을 때



**1.3 수행시간 분석 ( size = 1000 )**





2차원 배열로 그래프 내의 릴레이션을 찾았을 때, 사이즈가 10개인 그래프였을 경우 759초가 소요되지만 사이즈가 1000개가 되자 총 7111초의 시간이 걸린다.

**2. LinkedList를 활용하는 방법**

**2.1 코드**

#include <iostream>

#include <ctime>

#include <time.h>

#include <stack>

#include <list>

using namespace std;

stack<int> s;

class Graph{

public:

int size;

list<int> \*linked;

int\* mark;

public:

Graph(int size);

void Graphprint();

void DFS(int x, int temp);

void result();

};

Graph::Graph(int size){

srand((unsigned)time(NULL));

this->size = size;

mark = new int[size];

for (int i = 0; i < size; i++)

mark[i] = 0;

linked = new list<int>[size];

for (int i = 0; i < size; i++){

for (int j = i + 1; j < size; j++){

if (rand() % 2 == 1){

linked[i].push\_back(j);

linked[j].push\_back(i);

}

} // linked[i] 종료

}

}

void Graph::Graphprint(){

for (int i = 0; i < size; i++){

for (list<int>::iterator j = linked[i].begin(); j != linked[i].end(); j++){

cout << \*j << " ";

} cout << endl;

}

}

void Graph::DFS(int x, int temp){

int depthcheck = -1;

mark[x] = temp;

for (list<int>::iterator j = linked[x].begin(); j != linked[x].end(); j++){

if (mark[\*j] == 0){

depthcheck = 0;

DFS(\*j, temp);

}

} if (depthcheck == -1)

return;

}

void Graph::result(){

int temp = 1;

int searchcheck = -1;

int search = 0;

while (true){

searchcheck = -1;

DFS(search, temp);

cout << endl << "그래프 내 릴레이션 : " << endl;

for (int i = 0; i < size; i++){

if (mark[i] == temp)

cout << i << " ";

} temp++;

for (int i = 0; i < size; i++){

if (mark[i] == 0){

search = i;

searchcheck = 1;

}

} if (searchcheck == -1)

break;

}

return;

}

int main(){

clock\_t start, finish; double duration;

start = clock();

srand((unsigned)time(NULL));

int size = 0;

cout << "그래프의 크기를 입력하세요 : ";

cin >> size;

Graph arr(size);

arr.Graphprint();

arr.result();

finish = clock();

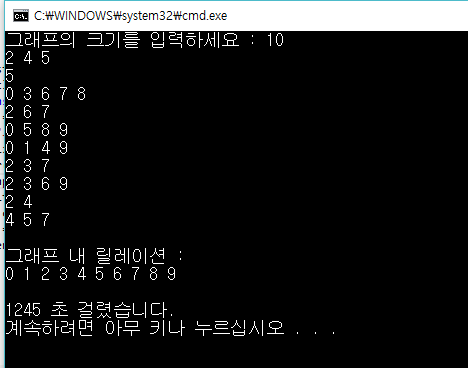
duration = (double)(finish - start);

cout << endl << endl << duration << " 초 걸렸습니다." << endl;

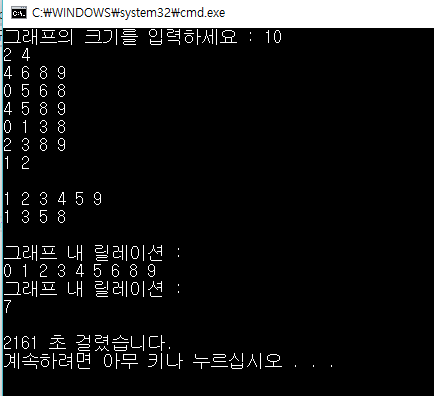
}

**2.2 실행 결과**

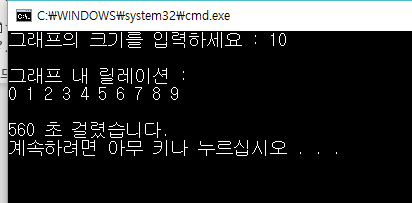
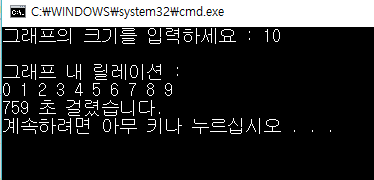
* 임의로 size 정보를 주었을 때



* 릴레이션이 여러 개 존재하는 그래프에서의 결과

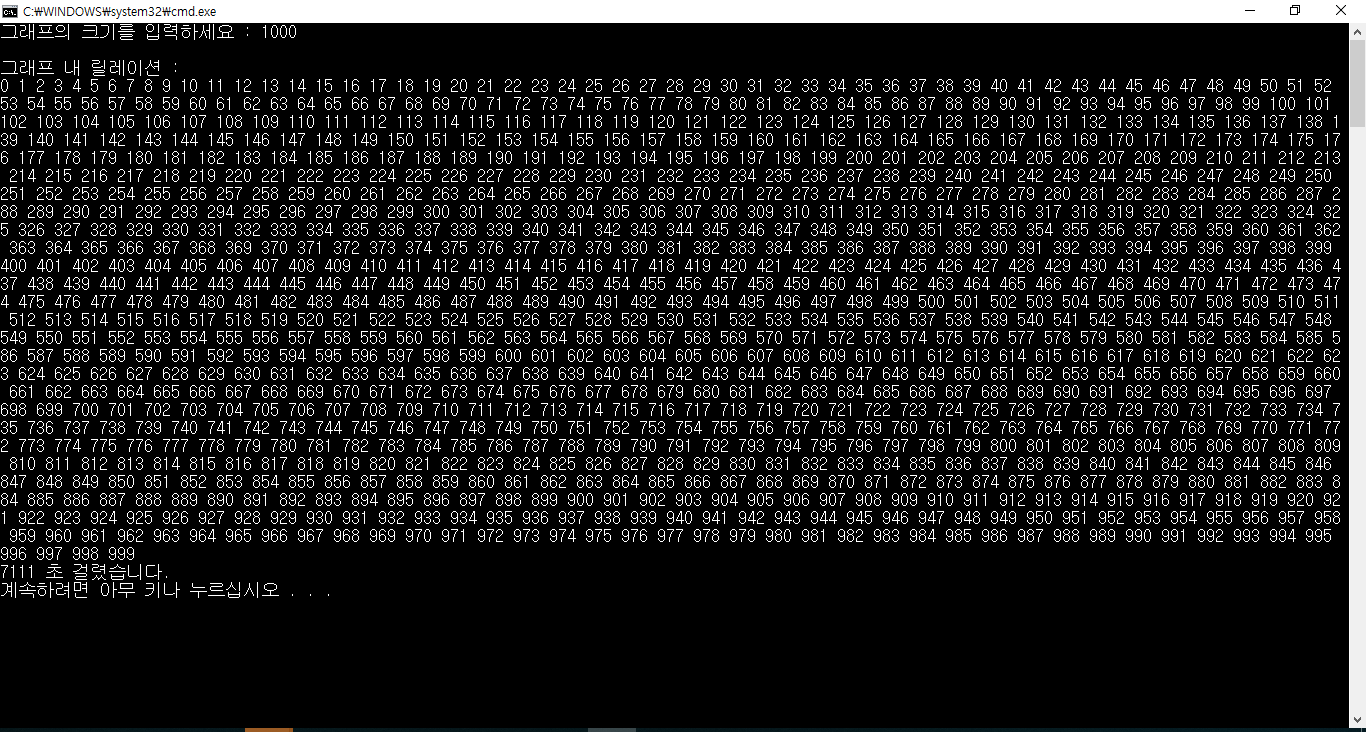


**2.3 수행시간 분석 ( size = 1000 )**

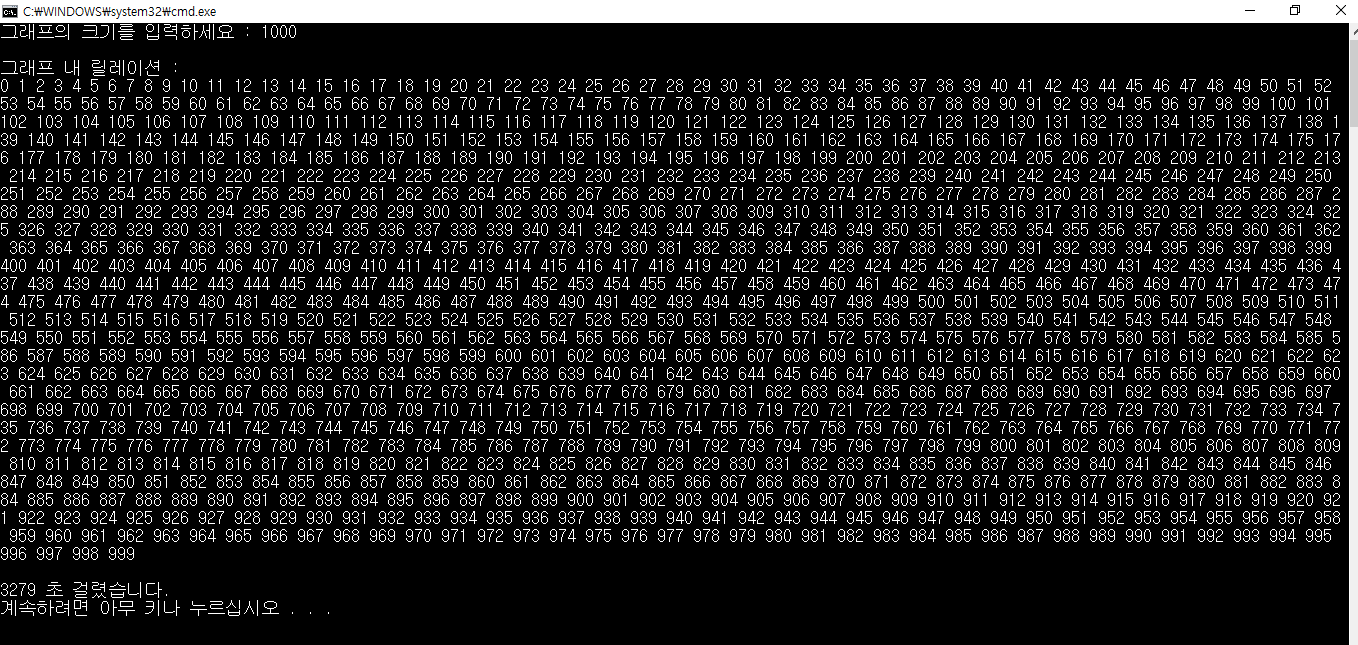


(좌) 2차원 배열로 구현, (우) LinkedList로 구현. 둘 다 그래프 출력을 수행하였다.

수행 시간에서 미세한 차이가 나긴 하나, 그래프를 랜덤으로 생성하기 때문에 같은 조건에서의 비교할 수가 없었기 때문에 어떤 그래프가 만들어지느냐에 따라서 그 차이는 거의 없다는 결론을 내릴 수 있었다. 하지만 동일하게 1개의 릴레이션만 존재하는 그래프를 가졌을 경우 LinkedList가 미세하게 빠른 것을 볼 수 있었다. ( 하지만 차이가 없다고 해도 무방하다. )



(상) 2차원 배열로 구현



(하) LinkedList로 구현

2차원 배열로 구현했을 경우 7111초, LinkedList로 구현했을 경우 3279초가 걸렸다. 둘의 결과만 놓고 비교했을 때, 2차원 배열로 구현했을 때 무려 LinkedList의 2배가 넘는 시간이 걸렸다고 볼 수 있다.