Course: 조대호 교수님 Algorithm

Theme: Airline reservation system

ID/Name: 2017312605/김요셉

Date: 2022-11-25

1 Introduction

1.1 Purpose of the system

본 시스템의 목적은 크게 알고리즘적 관점과 경제적 관점으로 구분됩니다. 우선 알고리즘 적인 목적은 시간에 따라 달라지는 그래프 상의 경로를 찾는 데에 있습니다. 일반적인 그래프는 시간의 흐름에 불변하는 영속적인 간선이 있다고 가정되므로 경로를 찾는 것이 매우 쉽습니다. 그러나 비행경로와 같이 시간에 따라 달라지는 그래프는 해당 시점에 그 경로가 없을 수 있기 때문에 이러한 문제를 해결하는 것이 알고리즘적 관점에서 본 시스템의 목적이라고 볼 수 있습니다.

다음으로 경제적 관점으로는, 단순히 비행시간을 가중치로 놓을 수 없기 때문에 최소 비용이 무엇인지에 대해 정의하는 것이 필요했습니다. 예를 들어 아무리 비행시간이 좀 더 빠르다 해도 2회로 나눠 타는 것이 더 고가일 수 있고, 반대로 1회로 타는 것이 경유하는 것보다 더 비쌀 수도 있습니다. 따라서 무엇이 경제적인지는 실제 항공사의 가격 산출 방식 등의 시장 조사가 필요하지만, 여기서는 간략화 하여 경유 횟수를 최소화하는 경로를 찾고자 하는 것이 경제적 관점에서 본 시스템의 목적입니다.

1.2 Input and output

1.2.1 Input

실제 구동화면 등은 2에서 확인하실 수 있습니다.

1.2.1.1 System Setting input

시스템 세팅을 위해서는 random화된 그래프 생성을 할 것인지 또는 이미 저장된 지리 데이터 및 노선 데이터를 불러올지 선택할 수 있습니다. 만일 사용자가 랜덤 생성을 요구할 경우 26개 도시의 지리 정보를 랜덤 생성한 후 그래프의 간선을 생성합니다. 그러나 이러한 방식은 몇 초 정도의 대기 시간을 요구한다는 단점이 있습니다. 이유는 특정 경로가 생성될 때, 과제의 조건에 따라 양측으로 연결이 되어야 하는데, 이 경로가 생성되었는지 확인하는 과정에서 최종적으로 후보선택지가 줄어드는 경우가 종종 있기 때문입니다. 만일 사용자가 기존 데이터를 불러온다면 빠르게 시스템을 초기화할 수 있습니다.

1.2.1.2 Test input generation

테스트 인풋은 랜덤 변수를 통해 영미권 국가의 이름에서 추출된 이름을 결합하여 만들어진 사용자 이름과, 경로데이터를 기반으로 형성된 출발지, 목적지, 그리고 출발일을 담고 있습니다. 이 부분은 눈으로 확인할 필요가 있으므로 binary file이 아닌 csv 파일 형태로 다루고 있습니다.

1.2.1.3 Manual input

입력시에는 과제 조건에 제시된 형태와 동일합니다. Name, src, dst, day의 형태로 입력을 받으며, 이는 테스트용 csv 파일과 동일한 형식입니다. 또한 사용자는 도시간의 경로를 검색할 수 있고, 특정 도시의 위치와 두 도시간의 거리를 출력할 수 있습니다. 또한 특정 RID의 삭제를 Red-Black tree를 통해 구현했습니다.

1.2.2 Output

출력 부분은 크게 경로 및 지리 여건에 대한 출력, 예약정보의 엔트리에 대한 출력, 그리고 레드블랙 트리 구조에 대한 출력 등으로 나뉩니다.

1.2.2.1 Path Finder

경로 및 지리 여건의 경우 도시 단위 혹은 두 도시 사이의 경로 및 지리여건을 출력할 수 있으며, 사용자가 예약정보를 입력하면 자동으로 경로 검색 프로시저가 작동됩니다. 직접 메뉴를 통해 특정 도시 사이의 특정 날짜의 경로를 확인할 수 있습니다.

1.2.2.2 Reservation

예약 정보 엔트리의 출력의 경우 특정 RID의 값을 검색하여 출력할 수 있습니다. 정렬된 정보를 보여주기 위해 전체를 inorder로 순회한 값을 최종적으로 출력합니다.

1.3 Red-black tree structure

Red-Black Tree Structure는 기존 hw2와 마찬가지로 90도 반시계방향으로 회전한 형태를 출력합니다. 그 이유는 내부 정보를 가능한 가로로 이어지게 해서 실제 DBMS의 튜플과 유사하게 보이고 가능한 레드블랙 트리의 구조가 정확히 형성되었는지 보이기 위함입니다.

1.4 Important functions

1.4.1 graph.c

1.4.1.1. init_flight_system

```
void init_flight_system(graph_t *g, bool adj[][MAXNODE], bool israng)
{
    if(israng) {
        get_rand_graph(g, adj);
        for (int i = 0; i < MAXNODE; i++)
            get_rand_table(&(g->v[i].t), i, adj);
    }
    else
    {
        load_graph(g, adj);
        for (int i = 0; i < MAXNODE; i++)
            load_table(&(g->v[i].t), i, adj);
    }
}
```

1.4.1.2 find_path

```
int find_path(graph_t *g, city_t src, city_t dst, int day, path_t *p)
{
    if (src == dst)
    {
        printf("src == dst₩n");
        return UNFOUND;
    }
    extern int MAXPATH;
    mhd_t time;
    tunit_t curtime;
    time.d = day;
    time.h = 0;
    time.m = 0;
    curtime = get_tunit(time);

    MAXPATH = 1;
LOOP:
    if (MAXPATH > MAXDEPTH)
        goto ERROR;

    if (is_connected(g, src, dst))
        if (dfs(g, src, dst, curtime, p) == FOUND)
            goto DONE;
    MAXPATH *= 2;
    goto LOOP;
ERROR:
    return UNFOUND;
DONE:
    return FOUND;
}
```

1.4.1.3 structure

1.4.1.5 point_t

```
typedef struct
{
    double x;
    double y;
} point_t;
```

1.4.1.6 gnode_t

```
typedef struct __gnode_
{
    struct __gnode_ *next;
    int dst;  // dst
    double w; // distance in km
} gnode_t;
```

1.4.1.7 graph_t

```
typedef struct
{
    ghead_t v[MAXNODE];
} graph_t;
```

1.4.2 input.c

1.4.2.1 void get_test_inputs(input_t in[MAX_INPUT], int len);

1.4.2.2 void get_test_inputs(input_t in[MAX_INPUT], int len);

1.4.2.3 input_t

```
typedef struct
{
    char src;
    char dst;
    char name[MAX_NAME];
    int day;
    int hour;
    int min;
} input_t;
```

1.4.3 rbtree.c

1.4.3.1 rb_insert_key

```
bool rb_insert_key(rbtree_t *T, key_t k, data_t item)
```

1.4.3.2 rb_delete_key

```
bool rb_delete_key(rbtree_t *T, key_t k)
```

1.4.3.3 rb_print

```
int rb_print(rbtree_t *T, rbnode_t *cur, rbedge_t *previous, bool isLeft)
```

1.4.4 state.c

1.4.4.1 init_state

```
void init_state(sys_state_t *state)
{
    init_flight_system( g: &state->path_for_each_city, state->adj, isrand: true);
    rb_malloc(&state->reservation_information);
    state->rid = 0;
}
```

1.4.4.2 load_state

```c
void load_state(sys_state_t *state)
{
    init_flight_system(g: &state->path_for_each_city, state->adj, isrand: false);
    rb_malloc(&state->reservation_information);
    state->rid = 0;
}
```

1.4.4.3 sys_state_t structure

```c
typedef struct
{
    // like pid, it increases continuously
    size_t rid; // reservation id
    // path graph with time table
    graph_t path_for_each_city;
    // for reservation delete and search
    rbtree_t reservation_information;
    // adjcent matrix for efficiency
    bool adj[MAXNODE][MAXNODE];
    // input buffer for efficiency
    input_t input_buffer[MAX_INPUT];
    // reservation(output) buffer
    path_t reservation_buffer[MAX_RESERVATION];
} sys_state_t;
```

1.4.5 table.c

1.4.5.1 table initializer

```c
void get_rand_table(table_t *t, city_t src, bool adj[][MAXNODE]);
void load_table(table_t *t, city_t src, bool adj[][MAXNODE]);
```

1.4.5.2 data structure

```c
typedef struct
{
    city_t dst;
    tunit_t tunit;
} tnode_t;

typedef struct
{
    city_t dsts[MAXFLIGHT * MAXDAY];
    city_t src;
} table_t;
```

1.5 Random variables

1.5.1 locations of each node

```c
point_t get_rand_point(unsigned seed)
{
    srand((unsigned)time(NULL) + seed);
    point_t ret;
    ret.x = (rand() / (double)RAND_MAX) * MAX_X;
    ret.y = (rand() / (double)RAND_MAX) * MAX_Y;
    return ret;
}
```

1.5.2 time slots tables of each path

```
void get_rand_table(table_t *t, city_t src, bool adj[][MAXNODE])
{
    char fname[512] = {0};
    sprintf(fname, "./pathinfo/table_info_city_%c.dat", src+'a');
    FILE * file_table_dat = fopen(fname, "wb");
    assert(file_table_dat!=NULL);

    int dst;
    int dsts[MAXEDGE] = {0};

    int j = 0;
    int tmp;
    for (int i = 0; i < MAXNODE && j < MAXEDGE; i++)
        if (adj[src][i]) {
            dsts[j] = i;
            j++;
        }

    for (int i = 0; i < MAXFLIGHT * MAXDAY; i++)
    {
        srand((unsigned)time(NULL) + 3*src * i * src * i + src + 2*i*i);
        tmp = rand() % MAXEDGE;
        dst = dsts[tmp];
        t->dsts[i] = dst;
    }
    fwrite(t, sizeof(table_t), 1, file_table_dat);
    fprint_table(t, src);
    fclose(file_table_dat);
}
```

1.5.3 random edges

간선의 경우, 시간이 너무 많이 소요되는 경우 리셋을 용이하게 하기 위해서 가능한 memset 기능을 바로 이용할 수 있는 배열구조로 marker를 두고 랜덤 추출로 두 도시를 서로 연결하였습니다. 이후 전체 marker가 완성된 후에는 각 간선별로 연결리스트를 형성하여 그래프 구조를 만들었습니다. 이 때, 각 노드에는 랜덤 생성된 테이블의 포인터가 포함되어 있으며, 이 값들은 가능한 한 번에 할당 받고 지역성을 활용하기 위해 실제 값들로 구성됩니다. 만일 메모리 할당을 힙 영역에서 할 경우 graph_t 자체를 할당하는 형태로 설계했습니다.

```
void ghinsert(ghead_t *h, int dst, double w)
{
    gnode_t *tmp = h->head.next;
    gnode_t *cur = (gnode_t *)malloc(sizeof(gnode_t));
    h->head.next = cur;
    cur->next = tmp;
    cur->dst = dst;
    cur->w = w;
}
```

```
void get_rand_graph(graph_t *g, bool adj[][MAXNODE])
{
    FILE * file_point_dat  = fopen("./pathinfo/node_locations.dat",  "wb");
    FILE * file_graph_adj  = fopen("./pathinfo/node_adj.dat",  "wb");
    assert(file_point_dat!=NULL);
    assert(file_graph_adj!=NULL);
    point_t p;
    point_t p_save[MAXNODE];
    int dst;
    init_graph(g);
    for (int i = 0; i < MAXNODE; i++)
    {
        p = get_rand_point(i);
        g->v[i].p  = p;
        p_save[i]  = p;
    }
    fwrite(p_save,  sizeof(point_t),  MAXNODE,  file_point_dat);
LOOP:
    srand((unsigned)time(NULL));
    for (int src = 0; src < MAXNODE; src++)
        memset(adj[src],  false,  sizeof(adj[src]));
    for (int src = 0; src < MAXNODE; src++)
    {
        for (int cnt = get_num_of_edges(adj,  src); cnt < 10;)
        {
            int start_time  = time(NULL);
            while (true)
            {
                if(start_time  != time(NULL))  // no hope
                    goto LOOP;
                dst = rand() % MAXNODE;
                if(src  == dst)
                    continue;
                if (!adj[src][dst]  && (get_num_of_edges(adj,  dst)<10))
                    break;
            }
            adj[src][dst]  = true;
            adj[dst][src]  = true;
            cnt++;
        }
    }
    fwrite(adj,  sizeof(adj), 1, file_graph_adj);
    for (int src = 0; src < MAXNODE; src++)
        for (int dst = 0; dst < MAXNODE; dst++)
            if (adj[src][dst]  && src != dst)
                ginsert(g,  src,  dst,  get_dist(g,  src,  dst));

    gprint(g);
    fclose(file_point_dat);
    fclose(file_graph_adj);
}

void print_adj(bool  adj[][MAXNODE])
{
    for (int i = 0; i < MAXNODE; i++)
    {
        for (int j = 0; j < MAXNODE; j++)
            printf("%d  ",  adj[i][j]  ? 1 : 0);
        printf("₩n");
    }
}
```

2 User Interface description

2.1 초기화

2.1.1 랜덤 초기화 실행

2.1.2 초기화 정보 로드(랜덤초기화 X)

2.2 샘플 입력 테스트

2.2.1 테스트 코드 실행

2.2.2 테스트 코드 실행하지 않는 경우

2.3 main menu 창 입력 방법

```
start program
init random system (yes/no)?
 >> no
load system.. wait a moment
test ? (y/n):
 >> n
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 1
input: name, src, dst, day
 >> Yosep Kim, s, w, 1
available path:
 >> s(0:0am, 1)-i(2:25am, 1)-w(4:51am, 1)
input success

1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >>
```

Wrong input 빠져나오는 방법

```
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 1
input: name, src, dst, day
 >> for delete
wrong input

wrong input

wrong input
, a, a, b
wrong input
for delete, a, b, 1
available path:
 >> a(0:0am, 1)-u(4:45am, 1)-b(6:8am, 1)
input success
```

2.3.1 input

```
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 1
input: name, src, dst, day
 >> Good bye, g, b, 10
available path:
 >> g(0:5am, 10)-o(2:30am, 10)-b(4:5am, 10)
input success
```

2.3.2 delete

```
 >> RB-Tree Structure with RID

    .--- R: RID(2): [for delete, 1, a, b, a(0:0am, 1)-u(4:45am, 1)-b(6:8am, 1)]
--- B: RID(1): [Good bye, 10, g, b, g(0:5am, 10)-o(2:30am, 10)-b(4:5am, 10)]
    `--- R: RID(0): [Yosep Kim, 1, s, w, s(0:0am, 1)-i(2:25am, 1)-w(4:51am, 1)]

 >> Sorted Info with RID

RID(0): Yosep Kim, 1, s, w, s(0:0am, 1)-i(2:25am, 1)-w(4:51am, 1)
RID(1): Good bye, 10, g, b, g(0:5am, 10)-o(2:30am, 10)-b(4:5am, 10)
RID(2): for delete, 1, a, b, a(0:0am, 1)-u(4:45am, 1)-b(6:8am, 1)

 >> number of reservations: 3

1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 2
delete rid
 >> 2

1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 3


 >> CURRENT RESERVATIONS

 >> RB-Tree Structure with RID

--- B: RID(1): [Good bye, 10, g, b, g(0:5am, 10)-o(2:30am, 10)-b(4:5am, 10)]
    `--- R: RID(0): [Yosep Kim, 1, s, w, s(0:0am, 1)-i(2:25am, 1)-w(4:51am, 1)]

 >> Sorted Info with RID

RID(0): Yosep Kim, 1, s, w, s(0:0am, 1)-i(2:25am, 1)-w(4:51am, 1)
RID(1): Good bye, 10, g, b, g(0:5am, 10)-o(2:30am, 10)-b(4:5am, 10)

 >> number of reservations: 2
```

2.3.3 print_rbtree

```
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 3


 >> CURRENT RESERVATIONS

 >> RB-Tree Structure with RID

    .--- R: RID(1): [Good bye, 10, g, b, g(0:5am, 10)-o(2:30am, 10)-b(4:5am, 10)]
--- B: RID(0): [Yosep Kim, 1, s, w, s(0:0am, 1)-i(2:25am, 1)-w(4:51am, 1)]

 >> Sorted Info with RID

RID(0): Yosep Kim, 1, s, w, s(0:0am, 1)-i(2:25am, 1)-w(4:51am, 1)
RID(1): Good bye, 10, g, b, g(0:5am, 10)-o(2:30am, 10)-b(4:5am, 10)

 >> number of reservations: 2
```

2.3.4 search path

```
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 4
input: src, dst, day
 >> a, b, 24

a(0:5am, 24)-c(4:40am, 24)-b(6:38am, 24)
```

2.3.5 search location of city

```
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 5
input: city
 >> a
output:
 >> a(x: 626.331, y: 11.8107)
```

2.3.6 search schedule table entry

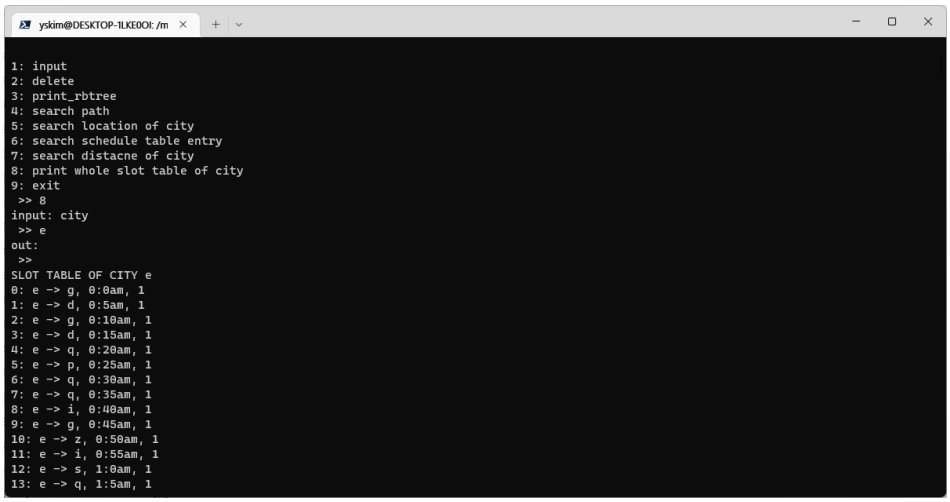
```
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 6
input: src, day, hour, min
 >> c, 24, 4, 40
output: dst, hour:mimute, day
 >> b, 4:40am, 24
```

2.3.7 search distance of city

```
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 7
input: src, dst
 >> c, b
output:
 >> distance: 984.13
 >> between c(x: 626.881, y: 1979.98) and b(x: 626.606, y: 995.849)
 >> time taken : 1:58am, 1
```

(1일 0시0분에 출발하면 1시간 58분이 걸린다는 의미로 표현했습니다)

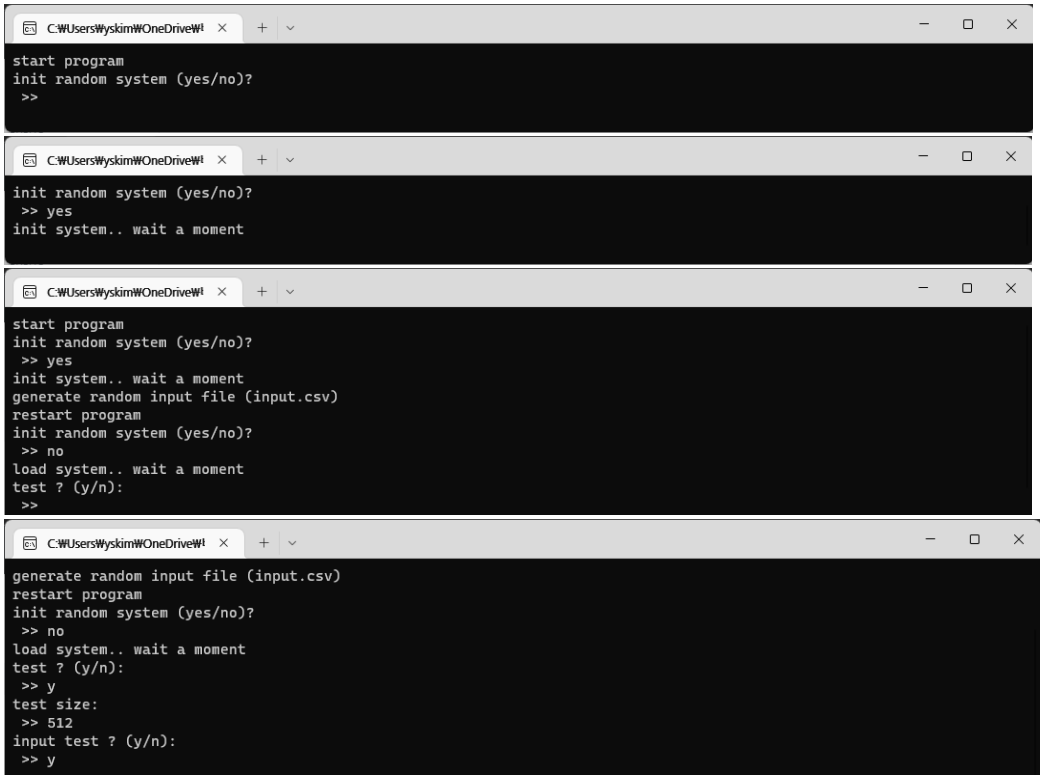
2.3.8 print slot table



2.3.9 exit

```
1: input
2: delete
3: print_rbtree
4: search path
5: search location of city
6: search schedule table entry
7: search distacne of city
8: exit
 >> 8
exit success
```

2.3 Other than the inputs and outputs



3 Conclusion

3.1 system explanations

3.1.1 RID

랜덤 변수 생성을 통해 26개 도시가 각 도시별로 10개의 도시와 연결되어 있도록 생성됩니다. 또한 RID라는 primary key를 도입하여 mysql에서 auto_increment와 유사하게 작동되도록 설계하여 각 reservation이 유일한 id를 갖도록 했습니다. 이름의 경우 자동생성 코드를 통해 적절한 영미권 이름이 생성되도록 했습니다. 또 연결된 도시에 대한 정보를 *.dat 파일 형태로 저장하여 해당 값을 불러온 후 csv 파일 형태로 input이 저장되게 했습니다.

3.1.2 slots

연결된 도시는 5분 간격으로 비행기가 출발하는 것을 가정합니다. 이러한 가정이 합리적인 이유는 실제 김포공항에서 이러한 방식을 적용하고 있고 [1]., 정확히 12회는 아니지만 중복 배정이나 여러 제반 사항을 제외할 경우 26개 5분 간격으로 배정되는 항공 slot이라는 개념이 실제 시장에서 사용되고 법률 용어(국토교통부훈령 제719호)로도 사용되고 있는 점이 확인되기 때문입니다. 공항에 따라서는 분단위로 표시되기도 하지만 DB 사이즈 등의 문제가 있어 5분 단위를 채택하는 경우가 많은 것으로 보입니다. 따라서 본 시스템에서도 이러한 점을 적용하였습니다.

3.2 examples, comments, and modularity

insert, delete의 example을 초반에 자동 생성되도록 설계한 덕분에 가능한 거의 모든 케이스가 자동 생성되므로 다양한 경우의 수와 코너케이스 등을 확인하면서 디버깅 할 수 있었습니다. 소스 코드의 위치는 C Programs₩src에 있으며, 모든 주석은 헤더파일에 있습니다. ide가 헤더파일의 주석을 읽어 오기 때문에 중복된 주석이 있으면 추후에 변경사항을 반영하기 어려웠기 때문에 하나로 통일했습니다. 모듈성을 높일 목적으로 다수의 파일로 나눈 점도 이후에 변경사항을 반영할 때 많은 도움이 되어 디버깅 시간이 크게 절약되었기 때문에 전체 시간에서는 오히려 긍정적인 결과를 얻을 수 있었던 것 같습니다.

3.3 Visualization

프로그램상에서는 그래프의 자동생성 이후 파이썬 스크립트를 자동으로 생성하여 pathinfo/graph_structure_networkx.py에 저장해주는 기능이 있습니다. 이를 통해 다음과 같이 좀 더 시각화된 형태로 그래프가 적절히 형성되었는지 디버깅할 수 있도록 했습니다.