

C언어 프로젝트

〈Smart UOS Map〉



수학과 송 통일
박 도형

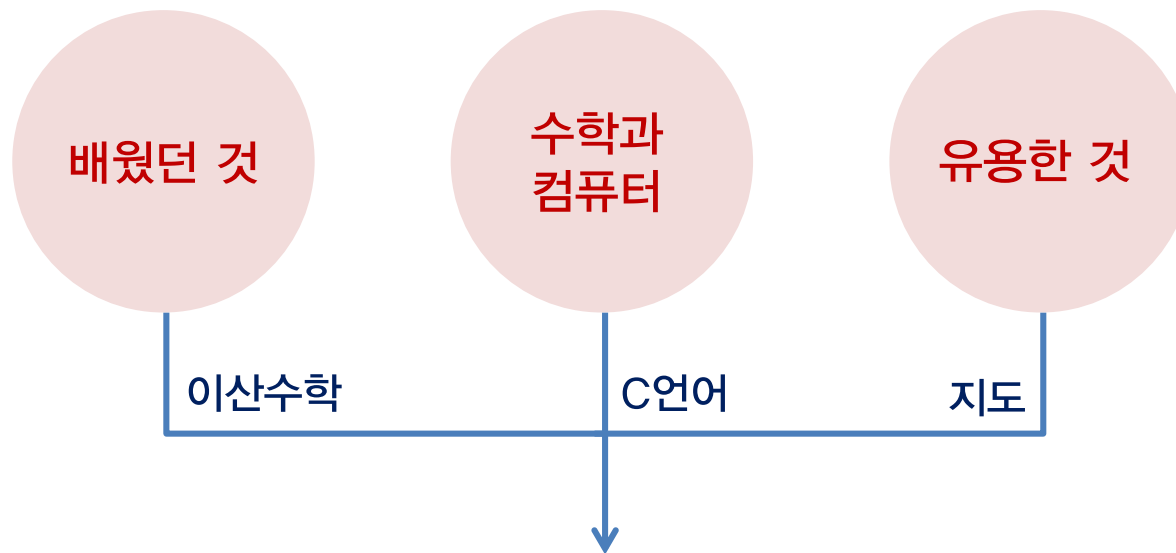
I N D E X

Step 1 주제 선정 이유

Step 2 준비과정

Step 3 구현과정 및
소스 설명

Step 4 실행



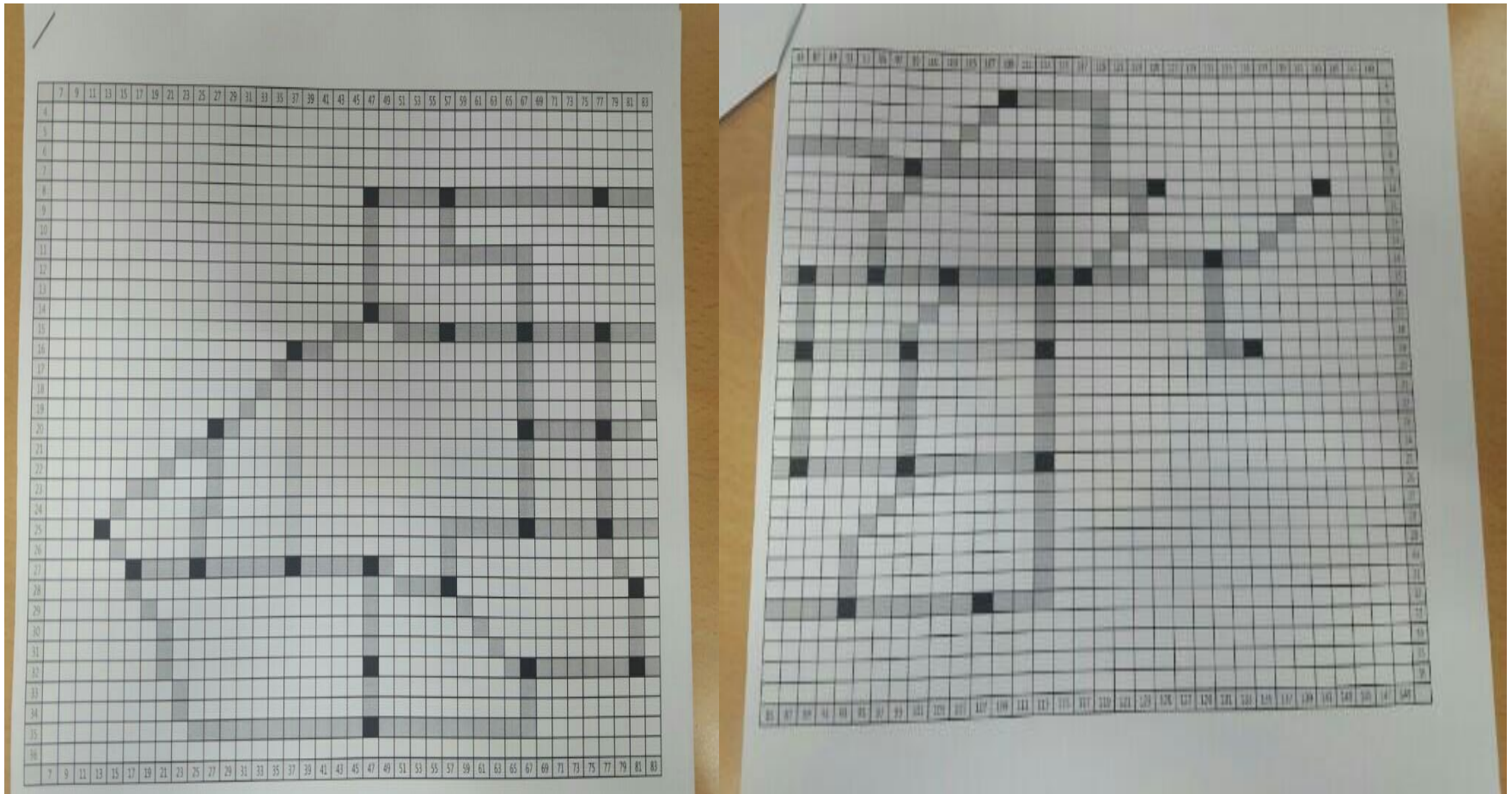
Smart UOS Map

- ✓ 시립대의 각 위치에서의 다른 위치까지의 최단루트/시간 검색
- ✓ 현재 위치에서 가장 가까운 ATM 찾기

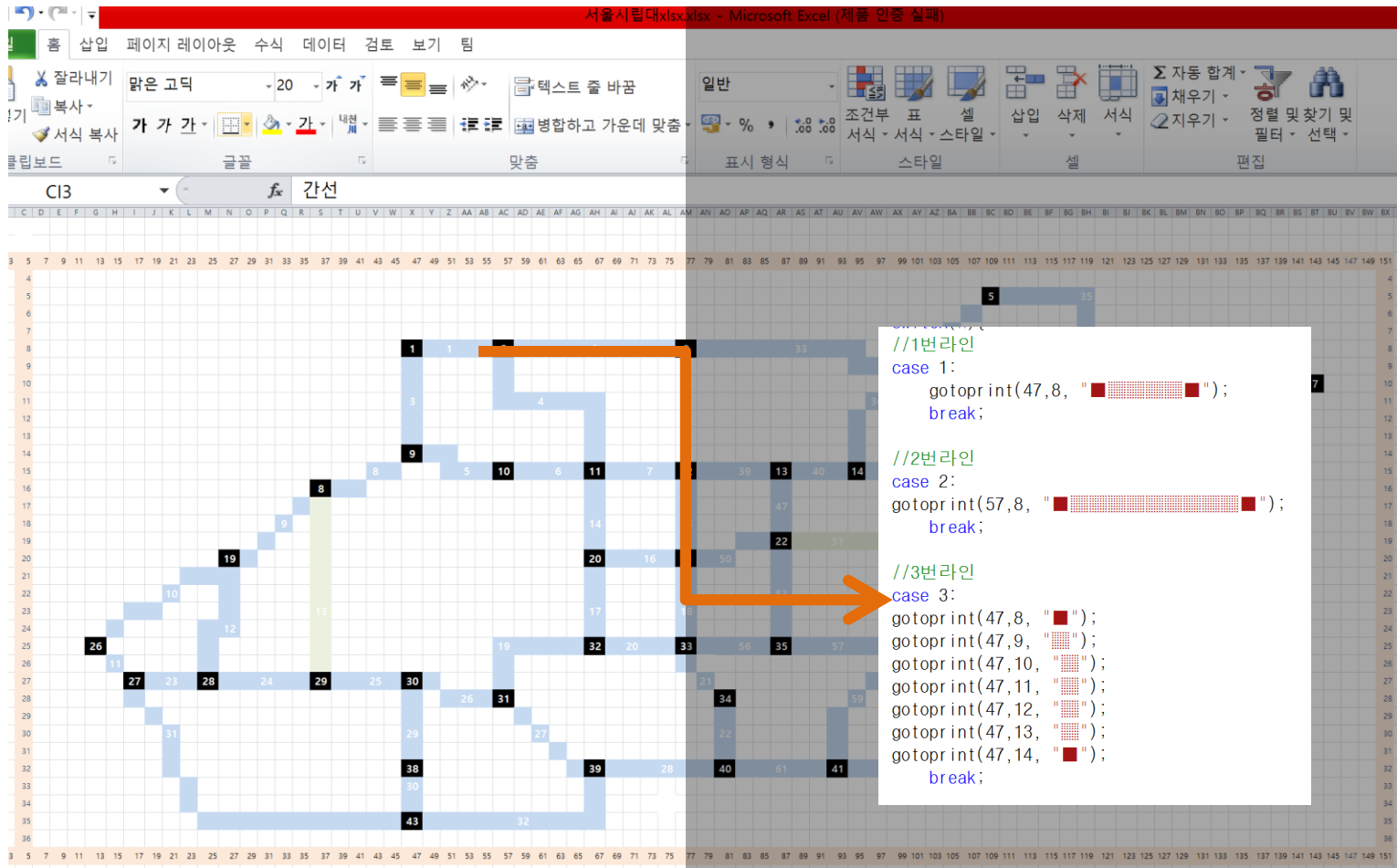
1. 지도를 통한 노드와 간선 만들기



2. 콘솔에 입력할 좌표를 알기 위한 작업



2. 콘솔에 입력할 좌표를 알기 위한 작업



The image shows a Microsoft Excel spreadsheet with a crossword puzzle grid. The grid is composed of blue cells for letters and black cells for empty space. Numbers are placed in the starting cells of the words. An orange arrow points from a cell in the grid to a code editor overlay on the right side of the image.

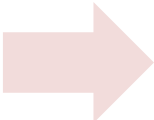
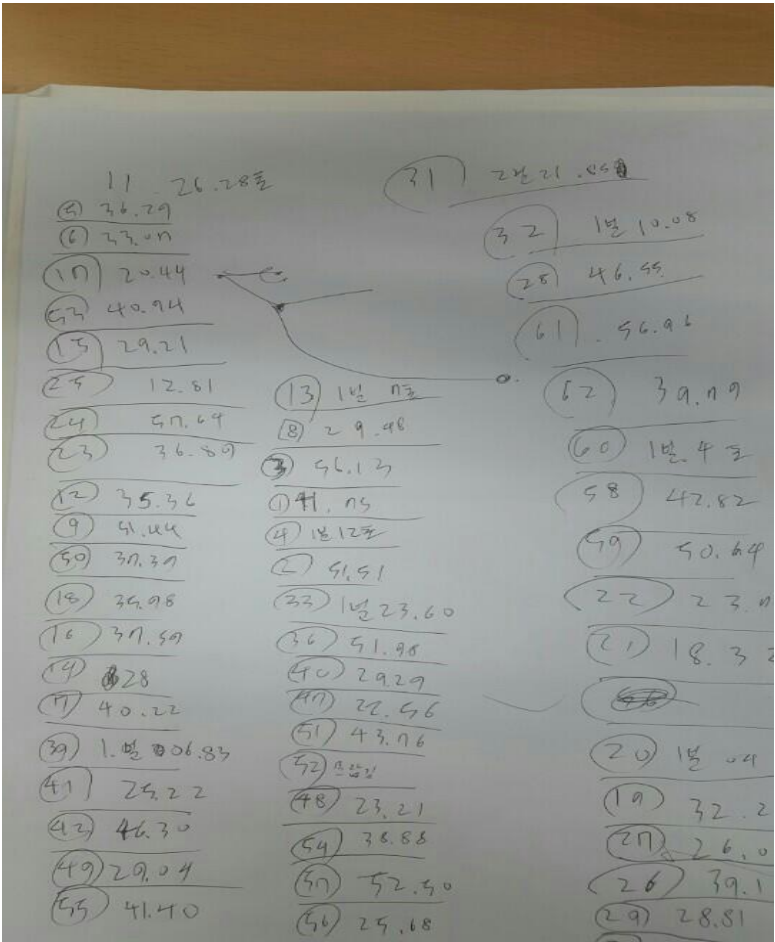
The code editor overlay contains the following code:

```
//1번라인
case 1:
    goto print(47,8, " ■■■■■■■■■■ ");
    break;

//2번라인
case 2:
    goto print(57,8, " ■■■■■■■■■■ ");
    break;

//3번라인
case 3:
    goto print(47,8, " ■ ");
    goto print(47,9, " ■ ");
    goto print(47,10, " ■ ");
    goto print(47,11, " ■ ");
    goto print(47,12, " ■ ");
    goto print(47,13, " ■ ");
    goto print(47,14, " ■ ");
    break;
```

3. 각각의 노드 사이의 간선 시간 재기



| 간선 | 시간 | 간선 | 시간 |
|----|-----|----|------|
| 1 | 41 | 40 | 29 |
| 2 | 52 | 41 | 25 |
| 3 | 56 | 42 | 46 |
| 4 | 72 | 43 | 19.6 |
| 5 | 36 | 44 | 38 |
| 6 | 33 | 45 | 67 |
| 7 | 40 | 46 | 50 |
| 8 | 29 | 47 | 22 |
| 9 | 52 | 48 | 23 |
| 10 | 94 | 49 | 29 |
| 11 | 26 | 50 | 37 |
| 12 | 35 | 51 | 44 |
| 13 | 67 | 52 | 42 |
| 14 | 28 | 53 | 41 |
| 15 | 29 | 54 | 39 |
| 16 | 37 | 55 | 41 |
| 17 | 20 | 56 | 26 |
| 18 | 36 | 57 | 52 |
| 19 | 32 | 58 | 43 |
| 20 | 64 | 59 | 50 |
| 21 | 18 | 60 | 64 |
| 22 | 24 | 61 | 57 |
| 23 | 37 | 62 | 40 |
| 24 | 57 | | |
| 25 | 13 | | |
| 26 | 39 | | |
| 27 | 26 | | |
| 28 | 46 | | |
| 29 | 29 | | |
| 30 | 21 | | |
| 31 | 141 | | |
| 32 | 70 | | |
| 33 | 83 | | |
| 34 | 40 | | |
| 35 | 88 | | |
| 36 | 52 | | |
| 37 | 74 | | |
| 38 | 37 | | |
| 39 | 67 | | |

준비완료 !

각 노드 간 최단 거리를
어떻게 구현할 것인가?

=> 이산수학, 다익스트라 알고리즘

다 익스트라 C로 구현

```
void mapSetting(int matrix[N][N]){
```

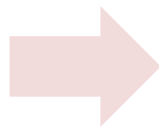
```
    matrix[0][1]=1;  
    matrix[0][8]=3;
```

```
    matrix[1][0]=1;  
    matrix[1][10]=4;  
    matrix[1][2]=2;
```

```
    matrix[2][1]=2;  
    matrix[2][3]=33;
```

```
    matrix[3][2]=33;  
    matrix[3][4]=34;  
    matrix[3][15]=37;  
    matrix[3][13]=36;
```

```
    matrix[4][3]=34;  
    matrix[4][5]=35;
```



시립대 지도에서 미리 정해놓은 각각의 노드와 간선을 셋팅 해주는 함수

한마디로 2차원 행렬로 그래프 각 노드간의 연결여부와 그에 해당되는 간선을 입력해줌

다 익스트라 C로 구현

다 익스트라를 이용해 가장 빠른 길을 출력해주는 함수 !!

```
void getpatharray(){
    int time[62]={41,52,56,72,36,33,40,29,52,94,26,35,67,28,29,37,20,36,32,64,18,24,37,57,13,39,26,46,
                29,21,141,70,83,40,88,52,74,37,67,29,25,46,19,38,67,50,22,23,29,37,44,42,41,39,41,
                26,52,43,50,64,57,40}; // 각 노선에 대응되는 시간(초)
    int matrix[N][N]={0};
    int timematrix[N][N]={0};
    int patharray[20]={0}; // 차례대로 노드를 저장할 배열
    int linearray[20]={0}; // 차례대로 노선을 저장할 배열

    char temp[16];          // 그냥 시작 노드를 입력받기 위한 키보드 입력용 버퍼
    int start, cur, finish; // 각각 시작 노드 번호, 현재 처리중인 노드 번호를 저장할 변수
    int i,j;                // 반복문(for)을 돌리기 위한 변수
    int min;                // 현재 처리중인 최소값을 저장하기 위한 변수
    int cost[N];            // (시작점에서) 각 노드로의 비용을 저장하기 위한 배열변수
    int solution[N];        // 확정이 된 (계산이 끝난) 노드인지 표시하는 기능
    int index[N]={0};       // 경로를 저장해줄 변수
    int size=0;
```

다 익스트라 C로 구현

```
mapSetting(matrix);  
mapSetting(timematrix);  
  
for(i=0;i<N;i++){  
    printf("Wn");  
    for(j=0;j<N;j++){  
        if(timematrix[i][j]==M||timematrix[i][j]==0){  
            printf("%3d ",timematrix[i][j]);  
        }  
        else {  
            printf("%3d ",time[matrix[i][j]-1]);  
            timematrix[i][j]=time[matrix[i][j]-1];  
        }  
    }  
}  
} //timematrix의 각각의 노선에 대응되는 경과 시간을 time matrix에 대입
```

다 익스트라 C로 구현

[illegible]

다익스트라 알고리즘 핵심 부분

```
for(i=0; i<N; i++)           // 결과가 저장 될 공간을 초기화 하기 위해 N만큼 루프를 돈다
{
    cost[i] = M;               // 비용(사용자가 지정한 노드 -> i번 노드로의 비용을 의미)
    solution[i] = 0;           // 확정된 노드인지 여부 (1이라면 계산이 끝나서 더이상 후보 노드가 아님을 의미)
}
cost[start] = 0;              // 시작점(start번 노드)에서 시작점으로는 당연히 비용이 0

for(i=0; i<N; i++)           // 총 노드의 개수만큼 루프를 돈다.
{
    min = M;                  // 우선 최소값을 Maximum으로 놓고 시작
    for(j=0 ; j<N ; j++)      // 서브 루프문, 사용되지 않은 노드중 비용이 최소인 노드를 찾기 위한 것임.
    {
        if(solution[j]==0 && cost[j]<min) //남아있는 노드(solution[i]가 0인)의 비용 중 현재 최소값보다 작으면
        {
            cur = j;           // 노드 번호 기록.
            min = cost[j];      // 그 비용을 최소값으로 기록
        }
    }
    solution[cur] = 1;          // 기록한 노드는 사용된 노드로 마킹.

    if(min == M)                // min값이 M(무한대값)이면 위의 for문에서 최소값을 구하지 못한 것, 즉 연결이 안되어있다는 의미
    {
        printf("그래프가 연결되어있지 않음.\n");
    }

    for(j=0; j<N ; j++)         // 두번째 서브 루프문, 최소비용을 확정하기 위한 루프문이다
    {
        if(solution[j] ==1)      // 새로 계산해 낼 노드가 이미 확정된 노드이라면 계산할 필요없으므로
            continue;           // 패스.
        if(cost[cur]+timematrix[cur][j] <cost[j]) // 현재 노드별로 저장된 비용(시작점으로부터의)과 위에서 얻어낸 최소비용을 가지는 노드과의 비용합
        {
            cost[j] = cost[cur] + timematrix[cur][j]; // 이 이전에 저장된 비용보다 작으면 갱신해 준다.
            index[j]=cur;
        }
    }
    index[start]--1;
}
```

가장 빠른 길 찾기

다익스트라 알고리즘 구현 후에 gets 함수를 통해 받은 start값과 finish값을 입력하여 역추적 과정을 통해 경로를 찾아 낸 후 각각의 경로를 노란색으로 콘솔창에 출력해주는 방식

다익스트라 알고리즘 구현

가장 가까운 ATM 기기 찾기

다익스트라 알고리즘 구현 후에 gets 함수를 통해 받은 start값을 입력받고 start 값과 ATM이 존재하는 8개의 노드와 걸리는 시간 중 최솟값을 구하는 방법을 이용하여 해당 노드를 찾은 뒤 그 노드와의 경로를 출력

전체 함수 목록

```
extern int getpatharray();  
extern int getatmpath();
```

```
void mapprint();  
void mainmenu();  
void gotoxy(int, int);  
void gotoprint(int, int, char*);  
void pathprint(int n);  
void mapprintnum();  
void timecount();  
void atm();
```

→ 각각의 노드에 ■ 이 아닌 숫자로 표시

1) Mapprint함수

```
for(i=4; i<=75; i++)
    goto print(2*i-1, 39, "—"); //맨아래 맵 틀
goto print(151, 39, "┘");
```

```

gotopr int(5, 41, "출발지");
gotopr int(5, 42, "도착지");
gotopr int(5, 43, "소요 시간");
gotopr int(5, 44, "출발지");
gotopr int(5, 45, "도착지");
gotopr int(5, 46, "소요 시간");

```

[illegible][illegible]

```
//3번라인
gotopr int(47,8, "■");
gotopr int(47,9, "■");
gotopr int(47,10, "■");
gotopr int(47,11, "■");
gotopr int(47,12, "■");
gotopr int(47,13, "■");
gotopr int(47,14, "■");
```


2) Pathprint 함수 => 간선의 번호를 받으면 해당 간선을 노란색으로 출력

```
void pathprint(int n){//경로를 출력하는 함수
    SetConsoleTextAttribute( GetStdHandle( STDOUT_HANDLE ), 6); //노란색

    switch(n){
        //1번라인
        case 1:
            goto print(47,8, " ■■■■■■■■■■ ");
            break;

        //2번라인
        case 2:
            goto print(57,8, " ■■■■■■■■■■■■■■■■■■■■■■ ");
            break;

        //3번라인
        case 3:
            goto print(47,8, " ■ ");
            goto print(47,9, " ■ ");
            goto print(47,10, " ■ ");
```

3) Mainmenu 함수

```

while(1){

gotoprint(65, 10, "_____");
gotoprint(63, 11, " | ※ 서울 시립대 Map※ | ");
gotoprint(65, 12, "_____");

gotoprint(65, 25, " [                ] ");
gotoprint(65, 26, " |   빠른길 찾기   | ");
gotoprint(65, 27, " [                ] ");

gotoprint(65, 28, " [                ] ");
gotoprint(65, 29, " |   가까운 ATM 찾기   | ");
gotoprint(65, 30, " [                ] ");

gotoprint(63, 32, "이동 : ↑↓ 선택 : Enter");

gotoprint(65, 40, " ");

```

```

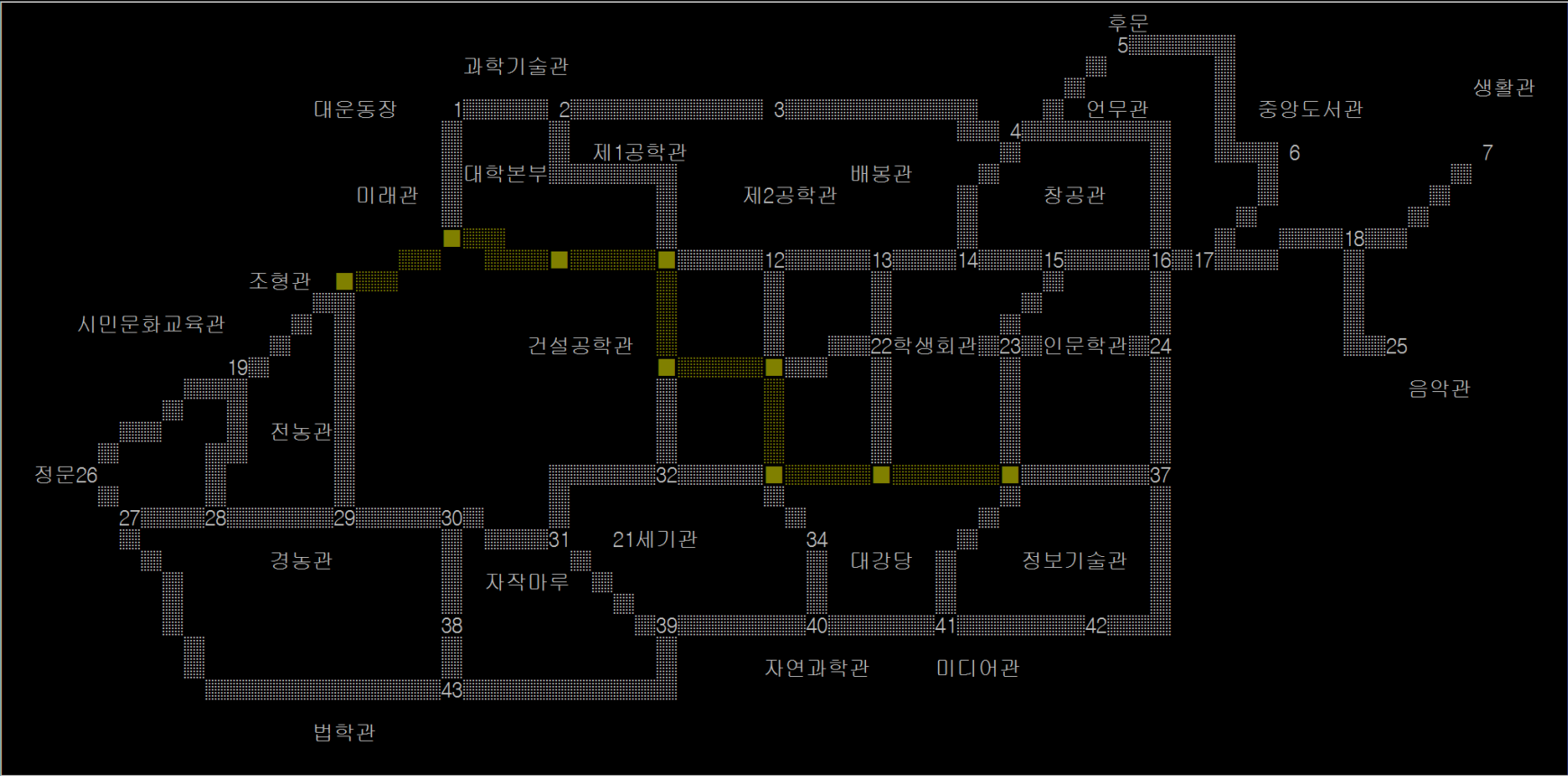
if (GetKeyState(VK_RETURN) < 0)
    continue;

Select = _getch();

switch (Select) {
case 80: // 아래
    num = 2;
    gotoprint(63, 26, " ");
    gotoprint(63, 29, "☞");
    break;
case 72: // 위
    num = 1;
    gotoprint(63, 26, "☞");
    gotoprint(63, 29, " ");
    break;
case 13: // 엔터
    system("cls");
    if (num == 1) {
        truth = 0;
        system("cls");
        break;
    } else if (num == 2) {
        atm();
    }
}

```

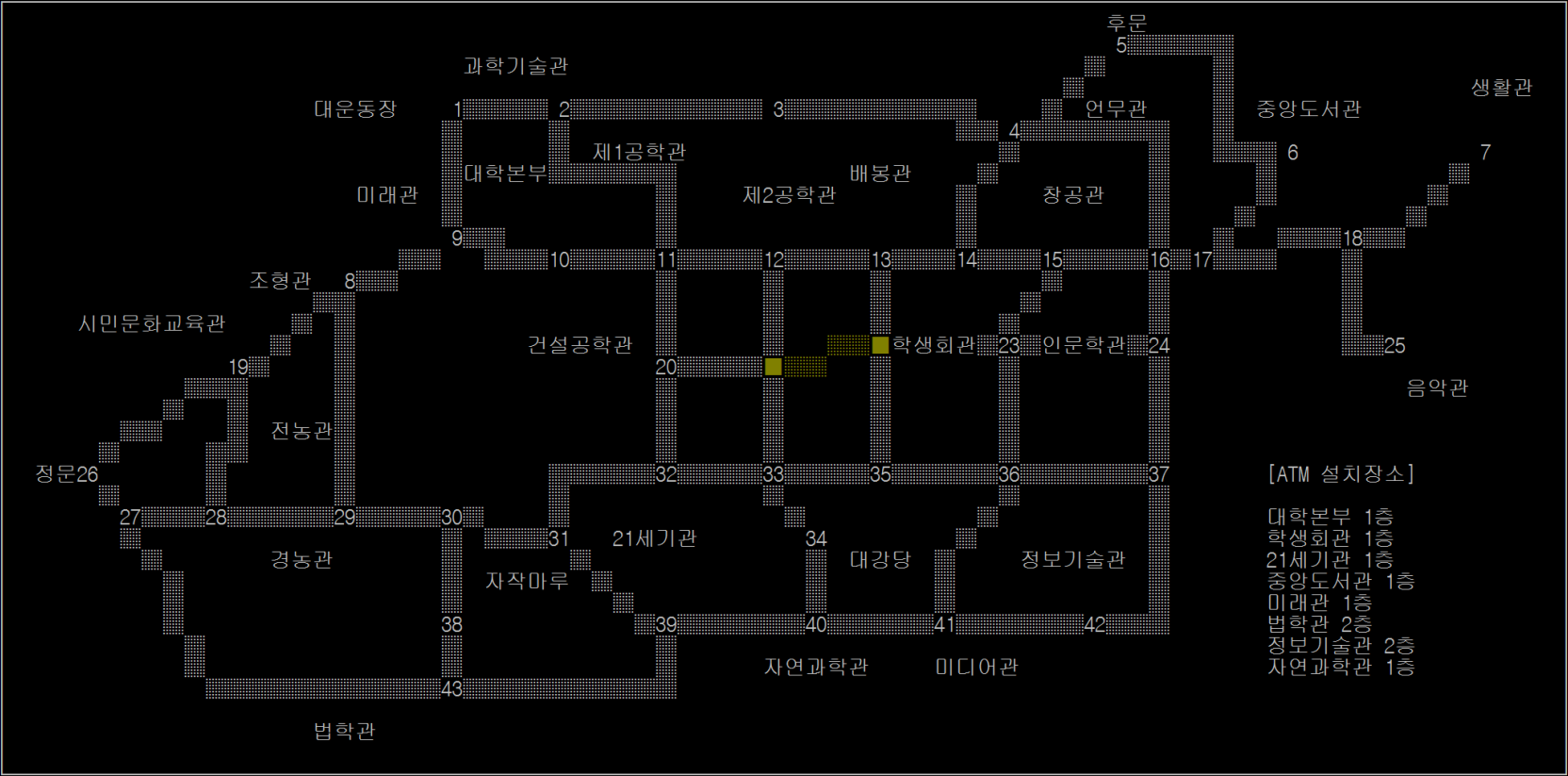
Step 4 **실행 및 Q&A**



| 출발지 | 도착지 | 소요 시간 |
|-----|-----|----------|
| 8 | 36 | 약 4분 37초 |



Step 4 실행 및 Q&A



| 현재위치 | 가까운 ATM 위치 | 소요 시간 |
|------|------------|----------|
| 21 | 학생회관 | 약 0분 37초 |

