

HW 5 Report

지하철역 거리 계산

자료구조 및 알고리즘 이해 (19-2, C070-1) / 이정원 교수님

201120696 최제현

문제분석 (요구사항)

- Graph 자료구조 & Dijkstra 알고리즘에 대한 이해
- 첨부된 실행파일과 동일한 동작
 - 출발역과 도착역을 입력 받아 경유하는 역과 구간 거리를 출력
 - 출발역에서 도착역까지의 거리를 출력
 - 출발역으로부터 다른 모든 역까지의 거리를 출력

* 첨부된 실행파일의 동작 스크린샷

```
선택 C:\Windows\system32\cmd.exe
0. 도봉산| 1. 창동| 2. 석계| 3. 동묘앞| 4. 시청| 5. 청량리| 6. 종로3가| 7. 종로4가| 8. 노원| 9. 용산| 10. 신도림| 11. 가산디지털| 12. 신길| 13. 운수| 14. 흥정현| 15. 금정| 16. 춘수| 17. 노량진| 18. 충정로| 19. 출대입구| 20. 대화로| 21. 양산| 22. 영등포구청| 23. 강일| 24. 사당| 25. 교대| 26. 건대입구| 27. 선릉| 28. 장실| 29. 신도림| 30. 정수| 31. 왕십리| 32. 신현| 33. 동대문역사문화공원| 34. 을지로4가| 35. 을지로3가| 36. 대포| 37. 연신내| 38. 광장| 39. 충무로| 40. 약수| 41. 홍대입구| 42. 고속터미널| 43. 양재| 44. 이매| 45. 수서| 46. 가락시장| 47. 오이도| 48. 노원| 49. 삼각지| 50. 풍차| 51. 종작| 52. 이수| 53. 김포공항| 54. 오이도| 55. 청구| 56. 여의도| 57. 공덕| 58. 강동| 59. 군자| 60. 천호| 61. 강릉| 62. 태릉입구| 63. 디지털미디어시티| 64. 성동| 65. 구청| 66. 부평구청| 67. 복정| 68. 강남| 69. 가좌| 70. 계양| 71. 모란| 72. 경자| 73. 원재

출발역을 입력해주세요(ex: 도봉산) : 도봉산
도착역을 입력해주세요(ex: 사당) : 사당
<도봉산->창동> 42
<창동->석계> 46
<석계->회기> 30
<회기->청량리> 14
<청량리->왕십리> 24
<왕십리->옥수> 32
<옥수->고속터미널> 57
<고속터미널->이수> 32
<이수->사당> 11
총 거리 28800m

모든 역까지의 거리(단위 100m)
0. 0| 1. 42| 2. 88| 3. 118| 4. 132|
5. 158| 6. 160| 7. 177| 8. 195| 9. 203|
10. 235| 11. 261| 12. 284| 13. 307| 14. 342|
15. 448| 16. 374| 17. 467| 18. 206| 19. 244|
20. 255| 21. 275| 22. 286| 23. 325| 24. 288|
25. 255| 26. 243| 27. 223| 28. 229| 29. 176|
30. 185| 31. 156| 32. 164| 33. 167| 34. 177|
35. 183| 36. 985| 37. 274| 38. 261| 39. 180|
40. 179| 41. 188| 42. 245| 43. 258| 44. 240|
45. 272| 46. 258| 47. 254| 48. 42| 49. 225|
50. 240| 51. 267| 52. 277| 53. 448| 54. 662|
55. 384| 56. 262| 57. 226| 58. 171| 59. 157|
60. 202| 61. 210| 62. 84| 63. 278| 64. 112|
65. 209| 66. 476| 67. 285| 68. 328| 69. 261|
70. 450| 71. 567| 72. 409| 

계속하려면 아무 키나 누르십시오 . . .
```

사용한 자료구조/알고리즘의 설명

1. Graph 자료구조

특징

- 연결되어 있는 객체간의 관계를 표현하는 자료구조
 - ex. 전기회로, 프로젝트 관리, 지도에서 도시들의 연결, 트리도 그래프의 일종으로 볼 수 있다.
- 그래프는 V와 E로 구성된다.
 - V는 정점(vertices)들의 집합
 - E는 간선(edge)들의 집합
 - 정점과 간선은 모두 관련되는 데이터를 가질 수 있다.
- 그래프의 종류

1. 무방향 그래프(undirected graph)

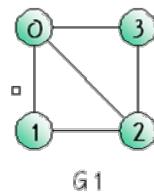
- 간선을 통해서 양방향으로 갈 수 있음을 나타내며 (A, B)와 같이 정점의 쌍으로 표현
- $(A, B) = (B, A)$

2. 방향 그래프(directed graph)

- 방향성이 존재하는 간선으로 도로의 일방통행길과 마찬가지로 한쪽 방향으로만 갈 수 있음을 나타낸다.
- $\langle A, B \rangle \neq \langle B, A \rangle$

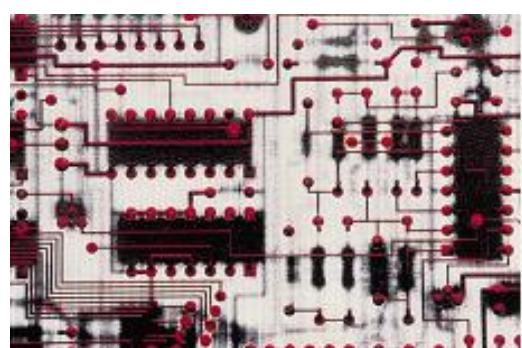
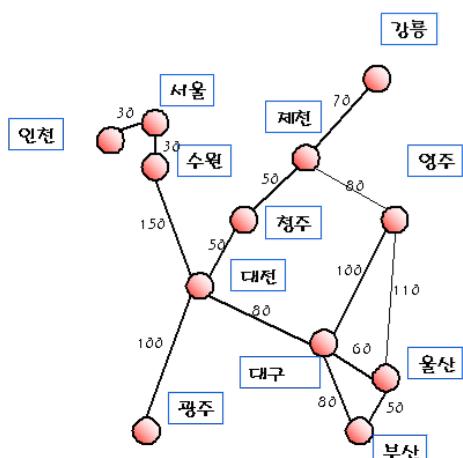
• 그래프 용어

- 인접 정점 (adjacent vertex): 간선에 의해 연결된 정점
- 차수 (degree): 정점에 연결된 다른 정점의 개수
- 경로 (path): 정점의 나열로 표현
 - 단순경로: 0, 1, 2, 3
 - 사이클(cycle): 0, 1, 2, 0
- 경로의 길이: 경로를 구성하는데 사용된 간선의 수
- 완전그래프: 모든 정점이 연결되어 있는 그래프



• 그래프 표현

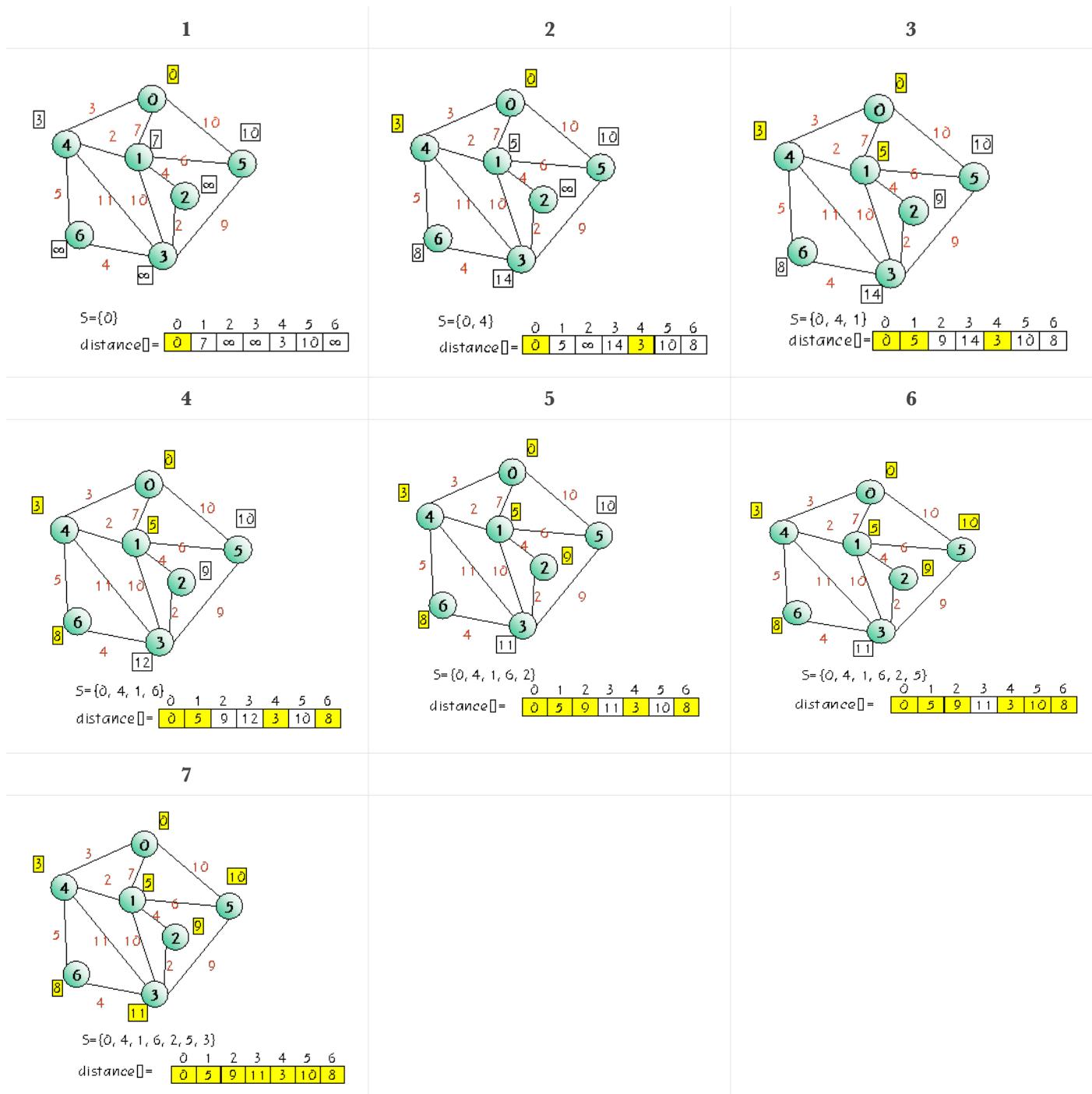
- 인접행렬 (adjacent matrix): 2차원 배열 사용 표현
- 인접리스트 (adjacent list): 연결리스트를 사용 표현
- 그래프를 통한 다양한 알고리즘 존재 (DFS, BFS, MST 등)



2. Dijkstra 알고리즘

특징

- 그래프를 이용한 최단경로 알고리즘 중 하나다.
- 네트워크에서 하나의 시작 정점으로부터 모든 다른 정점까지의 최단 경로를 찾는 알고리즘
- 집합 S: 시작 정점 v로부터의 최단경로가 이미 발견된 정점들의 집합
- distance 배열: 최단 경로를 알려진 정점만을 통하여 각 정점까지 가는 최단경로 길이
- 매 단계에서 가장 distance 값이 적은 정점을 S에 추가한다.
- 매 단계에서 새로운 정점이 S에 추가되면 distance 값을 갱신한다.



출처: 자료구조 및 알고리즘 강의노트 Chap10: Graph

소스코드 분석

Dijkstra 분석

Dijkstra.h #define, extern

```
1 #define TRUE    1
2 #define FALSE   0
3
4 void dijkstra_init(int nodes);
5 int choose(int* distance, int n, int* found);
6 void shortest_path(int start, int n, int **DISTANCE);
7
8 extern int *distance;
9 extern int *path;
10 extern int *found;
11
```

직관적인 boolean 처리를 위해 전처리 매크로로 TRUE는 1의 값을, FALSE는 0의 값을 갖도록 하였다.
dijkstra 알고리즘을 위해 필요한 distance, path, found를 node 포인터 변수를 외부 글로벌 변수를 가져와서 사용하도록 하였다.

dijkstra_init 함수

```
6
7 void dijkstra_init(int nodes)
8 {
9     distance=(int*)malloc(sizeof(int)*nodes);
10    path=(int*)malloc(sizeof(int)*nodes);
11    found=(int*)malloc(sizeof(int)*nodes);
12 }
13
```

dijkstra 알고리즘을 위해 필요한 distance, path, found를 node의 개수 만큼의 크기로 동적할당하여 초기화하였다.

choose 함수

```
15 int choose(int* distance, int n, int* found)
16 {
17     int i, min, minpos;
18     min = INT_MAX;
19     minpos = -1;
20     for(i=0;i<n;i++)
21         if( distance[i]< min && ! found[i] ){
22             min = distance[i];
23             minpos=i;
24         }
25     return minpos;
26 }
```

start 노드로부터 각 node까지의 distance 정보와, n개의 노드, 각 노드별 최소 거리를 찾았는지에 대한 found 정보를 받아서 아직 찾지 않은 min distance position을 찾아 반환하는 역할을 수행한다.

shortest_path 함수

```
28 void shortest_path(int start, int n, int **DISTANCE)
29 {
30     int i, u, w;
31     for(i=0; i<n; i++)
32     {
33         distance[i] = DISTANCE[start][i];
34         path[i] = start;
35         found[i] = FALSE;
36     }
37     found[start] = TRUE;
38     distance[start] = 0;
39     for(i=0; i<n-1; i++){
40         u = choose(distance, n, found);
41         found[u] = TRUE;
42         for(w=0; w<n; w++)
43             if(!found[w])
44                 if( distance[u]+DISTANCE[u][w] < distance[w] ) {
45                     distance[w] = distance[u]+DISTANCE[u][w];
46                     path[w] = u;
47                 }
48     }
49 }
```

#31~38

start 노드기준으로 distance, path, found를 초기화한다.

또한, 현재 start 노드 인덱스 값의 found, distance도 0, TRUE로 할당한다.

#39~48

먼저 start노드는 결정되었으니, n-1회 만큼 choose 함수를 호출하여 min distance position을 찾아내고, found 플래그 값을 TRUE로 바꾼다. 그 후, 노드 n회 만큼 아직 found=FALSE인 노드의 distance를, 현재 찾은 노드 u를 거쳐서 갔을 때 더 distance가 작은지 비교 하며 더 작을 때, 값을 갱신하고, 어떤 노드로 부터 지나온건지 path도 등록해준다.

이 과정을 마치면 start로부터 각 node까지의 최소 distance와, 어떤 path로부터 왔는지에 대한 정보를 알 수 있게 된다.

Filefn 분석

station_init 함수

```
7 char*** station_init(char ***STATION)
8 {
9     FILE *f_station;
10    int i=0,j=0;
11    int n=0,m=0;
12    f_station=fopen("station.txt","r");
13    fscanf(f_station,"%d\t%d",&n,&m);
14
15    STATION=(char***)malloc(sizeof(char**)*n);
16
17    for(i=0;i<n;i++){
18        STATION[i]=(char**)malloc(sizeof(char*)*m);
19        for(j=0;j<m;j++){
20            STATION[i][j]=(char*)malloc(20*sizeof(char));
21        }
22    }
23    fclose(f_station);
24    return STATION;
25 }
```

station.txt 파일의 첫줄의 n, m 총 인덱스 크기를 바탕으로 (첨부된 txt 파일에는 n=1, m=73)

STATION 3차원 포인터에 n Byte, m Byte, 20Byte 만큼 동적할당을 하며 초기화를 한다.

그래프 노드의 지하철역명 정보를 갖기 위해 만든 배열이다.

station_input 함수

```
27 char*** station_input(char ***STATION)
28 {
29     FILE *f_station;
30     int i=0,j=0;
31     int n=0,m=0;
32     f_station=fopen("station.txt","r");
33     fscanf(f_station,"%d\t%d",&n,&m);
34
35     for(i=0;i<n;i++){
36         for(j=0;j<m;j++){
37             fscanf(f_station,"%s",STATION[i][j]);
38         }
39     }
40     fclose(f_station);
41     return STATION;
42 }
```

station.txt 파일의 첫줄의 n, m 총 인덱스 크기를 바탕으로
역 이름들을 읽어와서 STATION 3차원 포인터 각각의 인덱스에 할당한다.

station_print 함수

```
44 char*** station_print(char ***STATION)
45 {
46     FILE *f_station;
47     int i=0,j=0;
48     int n=0,m=0;
49     f_station=fopen("station.txt","r");
50     fscanf(f_station,"%d\t%d",&n,&m);
51     for(i=0;i<n;i++){
52         for(j=0;j<m;j++){
53             printf("%3d.%20s",j,STATION[i][j]);
54             if(j%3==2)printf("\n");
55             else printf("|");
56         }
57     }
58     fclose(f_station);
59     return STATION;
60 }
```

station.txt 파일의 첫줄의 n, m 총 인덱스 크기를 바탕으로
반복하며 STATION 인덱스와 역명을 콘솔에 프린트한다.
각 노드의 지하철역 명을 출력한 것이다.

dist_init 함수

```
62 int** dist_init(int **DISTANCE)
63 {
64     FILE *f_dist;
65     int i=0,j=0;
66     int n=0,m=0;
67
68     f_dist=fopen("dist.txt","r");
69     fscanf(f_dist,"%d\t%d",&n,&m);
70
71     DISTANCE=(int**)malloc(sizeof(int*)*n);
72     for(i=0;i<n;i++){
73         DISTANCE[i]=(int*)malloc(sizeof(int)*m);
74     }
75     fclose(f_dist);
76     return DISTANCE;
77 }
```

dist.txt 파일의 첫줄의 n, m 총 인덱스 크기를 바탕으로 (첨부된 txt 파일에는 n=73, m=73)
DISTANCE 2차원 포인터에 n Byte, m Byte 만큼 동적할당을 하며 초기화를 한다.
그래프 노드의 각 인덱스 노드 간의 간선에 거리 정보를 갖기 위해 만든 배열이다.

dist_input 함수

```
79 int** dist_input(int **DISTANCE)
80 {
81     FILE *f_dist;
82     int i=0,j=0;
83     int n=0,m=0;
84     char temp[5];
85
86     f_dist=fopen("dist.txt","r");
87     fscanf(f_dist,"%d\t%d",&n,&m);
88
89     for(i=0;i<n;i++){
90         for(j=0;j<m;j++){
91             memset(temp,0,5);
92             fscanf(f_dist,"%s",temp);
93             DISTANCE[i][j]=atoi(temp);
94         }
95     }
96     fclose(f_dist);
97     return DISTANCE;
98 }
```

dist.txt 파일의 첫줄의 n, m 총 인덱스 크기를 바탕으로
DISTANCE 2차원 포인터 각각의 인덱스에 각 거리 정보를 정수로 할당한다.

dist_print 함수

```
100 int** dist_print(int **DISTANCE)
101 {
102     FILE *f_dist;
103     int i=0,j=0;
104     int n=0,m=0;
105
106     f_dist=fopen("dist.txt","r");
107     fscanf(f_dist,"%d\t%d",&n,&m);
108     printf("%d\t%d\n",n,m);
109
110    for(i=0;i<n;i++){
111        for(j=0;j<m;j++){
112            printf("%d",DISTANCE[i][j]);
113        }
114    }
115    fclose(f_dist);
116    return DISTANCE;
117 }
```

dist.txt 파일의 첫줄의 n, m 총 인덱스 크기를 바탕으로
반복하며 DISTANCE 거리 정보를 콘솔에 프린트한다.
각 노드간의 간선정보를 포함한 인접행렬을 출력한 것이다.

node_count 함수

```
119 int node_count()
120 {
121     FILE *f_station;
122     int i=0,j=0;
123     int n=0,m=0;
124     f_station=fopen("station.txt","r");
125     fscanf(f_station,"%d\t%d",&n,&m);
126
127     return m;
128 }
```

station.txt 파일의 첫줄의 m 총 인덱스 크기를 반환한다. m은 역 노드의 개수를 의미한다.

Etc 분석

print_path 함수

```
5 void print_path(int start, int end, int* path, char ***STATION, int **DISTANCE)
6 {
7     if( path[end] != start)
8         print_path(start, path[end], path, STATION, DISTANCE);
9     printf("<%s->%s> %d", STATION[0][path[end]], STATION[0][end], DISTANCE[path[end]][end]);
10    printf("\n");
11 }
```

path 1차원 배열 end 인덱스에 어느 인덱스의 node로 부터 온 건지에 대한 정보가 있고 start를 찾을 때까지 print_path 반복 재귀호출을 한다.
프린트 되는 결과는 start -> 중간 노드 … -> end 까지 잘라서 각 한줄씩 프린트한다.

print_distance 함수

```
13 void print_distance(int nodes, int end, int* distance)
14 {
15     int i;
16     printf("총 거리 %d00m\n",distance[end]);
17     printf("모든 역까지의 거리(단위 100m)\n");
18     for(i=0;i<nodes;i++){
19         printf("%3d.%4d|", i, distance[i]);
20         if(i%5==4)printf("\n");
21     }
22     printf("\n");
23 }
```

먼저 distance 배열에 end 노드까지의 거리를 프린트한다. (dijkstra 알고리즘으로 최단 경로 찾아낸 후 프린트)
그 후, distance 배열의 0부터 입력받은 nodes - 1 인덱스까지 인덱스 거리정보를 한줄씩 프린트한다.

input_string 함수

```
25 int input_string(int nodes,char ***STATION)
26 {
27     char string[20];
28     int i=0;
29     scanf("%s",string);
30     for(i=0;i<nodes && strcmp(string,STATION[0][i]);i++);
31     return i;
32 }
```

역 명을 입력받고, STATION 배열에서 어느 index의 노드에 역명이 있는지 찾고, 찾은 index를 반환한다.

Main 분석

```
7 char ***STATION;
8 int **DISTANCE;
9 int *distance;
10 int *path;
11 int *found;
12
13 void main( )
14 {
15     STATION = station_init(STATION);
16     station_input(STATION);
17     station_print(STATION);
18
19     int start, end;
20     int node_cnt = node_count();
21
22     printf("\n출발역을 입력해주세요(ex: 도봉산) : ");
23     start = input_string(node_cnt, STATION);
24
25     printf("도착역을 입력해주세요(ex: 사당) : ");
26     end = input_string(node_cnt, STATION);
27
28     DISTANCE = dist_init(DISTANCE);
29     dist_input(DISTANCE);
30
31     dijkstra_init(node_cnt);
32     shortest_path(start, node_cnt, DISTANCE);
33
34     print_path(start, end, path, STATION, DISTANCE);
35     print_distance(node_cnt, end, distance);
36 }
```

#7~11

그래프 자료구조를 위한 포인터 변수: STATION, DISTANCE / dijkstra 알고리즘을 위한 포인터 변수: distance, path, found

#15~17

그래프 노드의 역 정보를 표시하기 위해, 먼저 STATION을 초기화하고, 정보를 넣고, 프린트한다.

#19~23

입력받은 역의 index를 저장할 start, end 변수를 만들고, node의 count 수가 반복해서 많이 필요하므로 node_count를 재사용하기 위해 node_cnt 변수를 만들어 할당한다. 또한, 출발역 입력, 도착역 입력을 받는다.

#28~29

역 정보를 입력 받은 후, dijkstra 알고리즘 연산 전 그래프 각 노드의 간선을 표현한 인접행렬 DISTANCE를 초기화하고, 정보를 넣는다. (이 부분은 더 먼저 15번 라인에 넣어도 상관없지만, 흐름상 출발역, 도착역을 제대로 할당한 후에 초기화 및 정보할당을 진행하였다.)

#31~32

dijkstra_init을 먼저하여 main의 변수 distance, path, found를 node_cnt 크기로 동적 할당 초기화를 한다. (이 부분도 더 먼저 node_cnt 정보와 함께 15번 라인에 넣어도 상관 없지만, 흐름상 dijkstra 알고리즘 실행 전인 이곳에 두었다.) start 노드 인덱스 기준으로 shortest_path dijkstra 알고리즘 함수를 호출하여, start 노드로부터 각 노드까지 distance가 얼마나 되는지, 어떤 path를 지나왔는지에 대한 정보를 갱신한다.

#34~35

갱신된 distance, path 정보를 바탕으로 end까지의 path, distance 정보를 프린트한다.

⚠ 여기서 #15, #28에서만 반환값을 각 포인터 변수에 다시 할당해주었는데

각 함수들이 포인터 변수 선언이 동일 차수로 되어있기에 value 참조로 인해, STATION, DISTANCE 변수에 제대로 init이 되는게 아니므로 반환 값을 할당해준 것이고, 그 외에는 동일 차수로 변수 선언이 되어있더라도 하위 차수의 동일한 주소 값의 인덱스들을 참조하여 읽기/쓰기를 진행하므로 반환 값을 따로 다시 할당하지 않았다.

출력된 결과화면 캡쳐 및 결과에 대한 분석

도봉산 -> 사당

```
선택 Microsoft Visual Studio 디버그 콘솔
기좌| 70.          계양| 71.          원인재
72.          정자|
출발역을 입력해주세요(ex: 도봉산) : 도봉산
도착역을 입력해주세요(ex: 사당) : 사당
<도봉산->청동> 42
<청동->석계> 46
<석계->회기> 30
<회기->청량리> 14
<청량리->왕십리> 24
<왕십리->목수> 32
<목수->고속터미널> 57
<고속터미널->이수> 32
<이수->사당> 11
총 거리 28800m
모든 역까지의 거리(단위 100m)
0. 0| 1. 42| 2. 88| 3. 118| 4. 132|
5. 158| 6. 160| 7. 177| 8. 195| 9. 203|
10. 235| 11. 261| 12. 284| 13. 307| 14. 342|
15. 448| 16. 374| 17. 467| 18. 206| 19. 244|
20. 255| 21. 275| 22. 286| 23. 325| 24. 288|
25. 255| 26. 243| 27. 223| 28. 229| 29. 176|
30. 185| 31. 156| 32. 164| 33. 167| 34. 177|
35. 183| 36. 385| 37. 274| 38. 261| 39. 180|
40. 179| 41. 188| 42. 245| 43. 258| 44. 240|
45. 272| 46. 258| 47. 254| 48. 42| 49. 225|
50. 240| 51. 267| 52. 277| 53. 448| 54. 662|
55. 384| 56. 262| 57. 226| 58. 171| 59. 157|
60. 202| 61. 210| 62. 84| 63. 278| 64. 112|
65. 209| 66. 476| 67. 285| 68. 328| 69. 261|
70. 450| 71. 567| 72. 409|
```

C:\Users\cjh_windows\source\repos\HW5\Debug\HW5.exe(11248 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.

부평구청 -> 부평

```
선택 Microsoft Visual Studio 디버그 콘솔
45.          수서| 46.          가락시장| 47.          오금
48.          노원| 49.          삼각지| 50.          이촌
51.          동작| 52.          이수| 53.          금정도
54.          오이도| 55.          김포공항| 56.          여의도
57.          곤델| 58.          청구| 59.          군자
60.          천호| 61.          청동| 62.          태릉입구
63.          디지털미디어시티| 64.          상봉| 65.          강남구청
66.          부평구청| 67.          롯강| 68.          모란
69.          가좌| 70.          계양| 71.          원인재
72.          정자|
출발역을 입력해주세요(ex: 도봉산) : 부평구청
도착역을 입력해주세요(ex: 시당) : 부평
<부평구청->부평> 20
총 거리 2000m
모든 역까지의 거리(단위 100m)
0. 466| 1. 424| 2. 382| 3. 352| 4. 338|
5. 312| 6. 306| 7. 289| 8. 271| 9. 273|
10. 243| 11. 217| 12. 194| 13. 169| 14. 161|
15. 295| 16. 102| 17. 20| 18. 260| 19. 232|
20. 221| 21. 201| 22. 190| 23. 180| 24. 268|
25. 294| 26. 306| 27. 326| 28. 373| 29. 356|
30. 357| 31. 328| 32. 311| 33. 302| 34. 292|
35. 286| 36. 362| 37. 308| 38. 303| 39. 293|
40. 307| 41. 314| 42. 278| 43. 321| 44. 341|
45. 390| 46. 402| 47. 418| 48. 432| 49. 261|
50. 262| 51. 253| 52. 271| 53. 428| 54. 215|
55. 154| 56. 204| 57. 240| 58. 311| 59. 372|
60. 406| 61. 414| 62. 390| 63. 255| 64. 378|
65. 315| 66. 0| 67. 422| 68. 465| 69. 272|
70. 88| 71. 120| 72. 479|
```

C:\Users\cjh_windows\source\repos\HW5\Debug\HW5.exe(9528 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.

홍대입구 -> 홍대입구

The screenshot shows a Microsoft Visual Studio interface with a console window titled "Microsoft Visual Studio 디버그 콘솔". The window displays a distance matrix and a shortest path calculation between two locations.

Distance Matrix:

57.	공덕	58.	청구	59.	군자
60.	천호	61.	강동	62.	태릉입구
63.	디지털미디어시티	64.	성동	65.	강남구청
66.	부평구청	67.	狎邪	68.	모란
69.	기장	70.	개양	71.	원인재
72.	정자				

출발역을 입력해주세요(ex: 도봉산) : 홍대입구
도착역을 입력해주세요(ex: 사당) : 홍대입구
<홍대입구->홍대입구> 0
총 거리 000m
모든 역까지의 거리(단위 100m)

0.	244	1.	202	2.	160	3.	130	4.	116
5.	90	6.	84	7.	67	8.	49	9.	60
10.	88	11.	75	12.	62	13.	63	14.	98
15.	232	16.	130	17.	223	18.	38	19.	0
20.	11	21.	31	22.	42	23.	81	24.	125
25.	137	26.	149	27.	169	28.	200	29.	147
30.	135	31.	106	32.	89	33.	80	34.	70
35.	64	36.	141	37.	87	38.	82	39.	71
40.	87	41.	103	42.	121	43.	164	44.	184
45.	233	46.	229	47.	245	48.	210	49.	49
50.	69	51.	96	52.	114	53.	285	54.	418
55.	140	56.	55	57.	28	58.	89	59.	150
60.	195	61.	203	62.	168	63.	34	64.	156
65.	158	66.	232	67.	256	68.	299	69.	51
70.	206	71.	323	72.	322				

C:\Users\cjh_windows\source\repos\HW5\Debug\HW5.exe(3340 프로세스)이(가) 0 코드로 인해 종료되었습니다.
이 창을 닫으려면 아무 키나 누르세요.

- 출발역과 도착역을 입력 받아 경유하는 역과 구간 거리를 출력

- 출발역에서 도착역까지의 거리를 출력

- 출발역으로부터 다른 모든 역까지의 거리를 출력

첨부된 실행파일과 동일한 모든 요구사항에 맞는 동작을 보이는 것을 확인할 수 있다.

또한, dijkstra 알고리즘을 통해 갱신된 distance 배열(=각 역까지의 거리)에 끊어진 값을 의미하는 9999이 하나도 없으므로 과제에 사용된 그래프는 모든 정점이 연결되어있는 완전 그래프임을 알 수 있다.

참고 자료

- 자료구조 및 알고리즘이해, 과제5 PDF, 소스코드
- 자료구조 및 알고리즘이해, 강의노트 10장 Graph