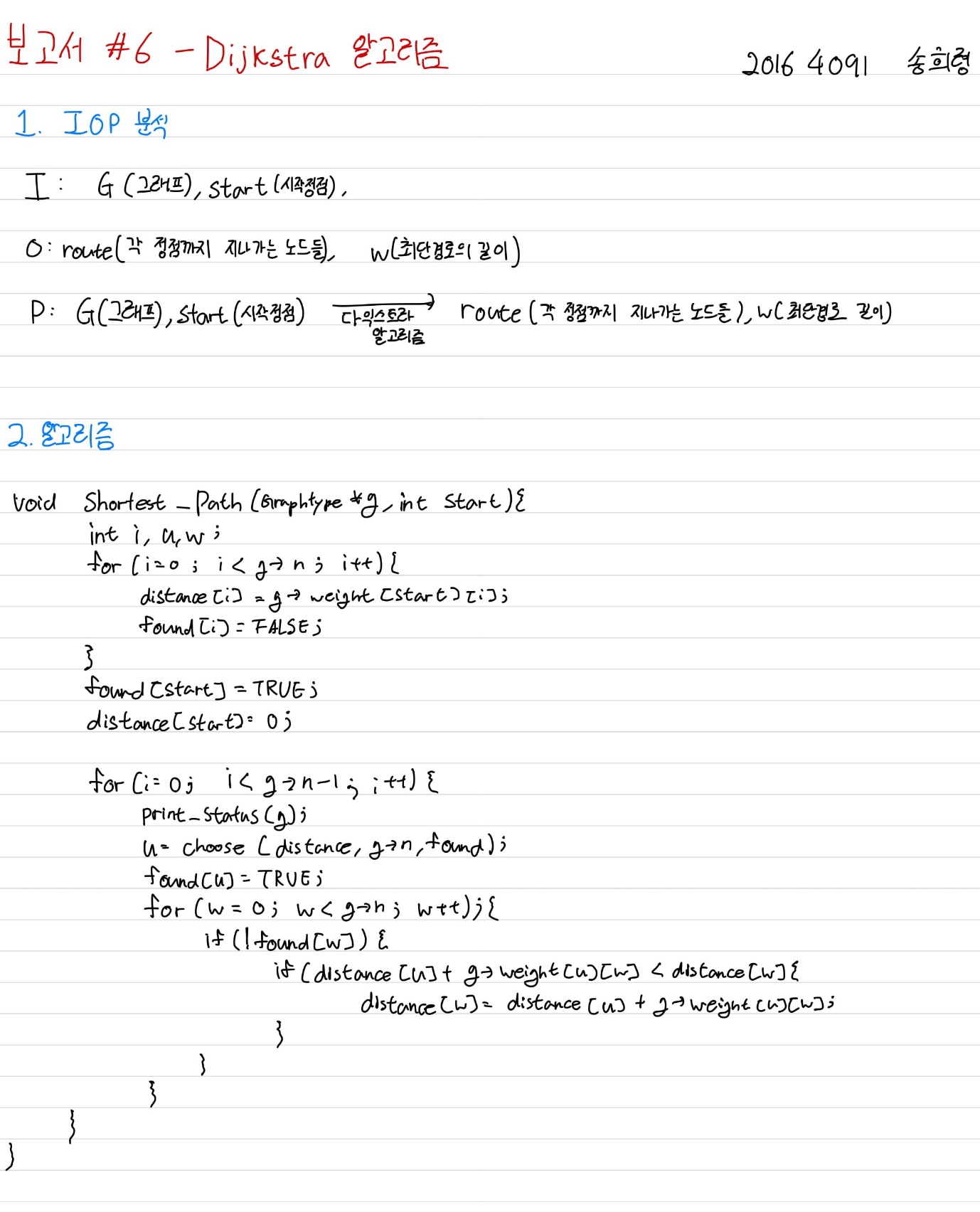
데이터구조 2

보고서 #6

(Dijkstra 알고리즘)

20164091

송희령



3. 코드

#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define MAX\_VERTICES 100

#define INF 1000000 //무한대. (연결이 없는 경우)

typedef struct GraphType {

    int n; //정점의 개수

    int weight[MAX\_VERTICES][MAX\_VERTICES];

} GraphType;

코드 실행에 필요한 헤더와 각 상수값 및 그래프타입 정의입니다

int distance[MAX\_VERTICES];

int found[MAX\_VERTICES];

int trace[MAX\_VERTICES];

int r\_trace[MAX\_VERTICES];

거릴를 저장할 distance, 각 노드별 최소길이를 구했는지 판단을 위한 found, 최소경로를 추척해 나갈 trace, 추적한 경로를 다시 재배치할 r\_trace 배열입니다.

int choose(int distance[], int n, int found[]) {

    int i, min, minpos;

    min = INT\_MAX;

    minpos = -1;

    for (i = 0; i < n; i++) {

        if (distance[i] < min && !found[i]) {

            min = distance[i];

            minpos = i;

        }

    }

    return minpos;

}

계산된 경로들 중 최소경로를 출력해줄 choose함수 입니다.

void shortest\_path(GraphType \*g) {

    int i, u, w, start;

    for (int s = 0; s < g->n; s++) {

        start = s;

        for (i = 0; i < g->n; i++) {

            distance[i] = g->weight[start][i];

            found[i] = FALSE;

            if (g->weight[start][i] != INF)

                trace[i] = start;

        }

        distance[start] = 0;

        for (int tmp = 0; tmp < g->n; tmp++) {

            for (i = 0; i < g->n; i++) {

                u = choose(distance, g->n, found);

                if (tmp == u) {

                    if (start == u)

                        continue;

                    printf("%d 에서 %d 까지 루트 : %d", start, u, start);

                    int n\_route = trace[u];

                    int q = 0;

                    while (1) {

                        if (n\_route == start) {

                            while (q > 0) {

                                printf("-%d", r\_trace[--q]);

                            }

                            break;

                        }

                        r\_trace[q++] = n\_route;

                        n\_route = trace[n\_route];

                    }

                    printf("-%d \t비용 : %d\n", u, distance[u]);

                    for (int k = 0; k < g->n; k++) {

                        distance[k] = g->weight[start][k];

                        found[k] = FALSE;

                    }

                    break;

                }

                found[u] = TRUE;

                for (w = 0; w < g->n; w++) {

                    if (!found[w]) {

                        if (distance[u] + g->weight[u][w] < distance[w]) {

                            distance[w] = distance[u] + g->weight[u][w];

                            trace[w] = u;

                        }

                    }

                }

            }

        }

        printf("\n%d번시작 출력 종료-----------------------------\n\n", s);

    }

}

최소 경로를 출력해줄 shortest\_path함수 입니다.

최소 경로의 값과 그 경로노드들을 출력해 줍니다.

각 경로간 최소값 계산 후, 비교합니다.

int main(void) {

    GraphType g = {7,

                   {{0, 7, INF, INF, 3, 10, INF},

                    {7, 0, 4, 10, 2, 6, INF},

                    {INF, 4, 0, 2, INF, INF, INF},

                    {INF, 10, 2, 0, 11, 9, 4},

                    {3, 2, INF, 11, 0, INF, 5},

                    {10, 6, INF, 9, INF, 0, INF},

                    {INF, INF, INF, 4, 5, INF, 0}}};

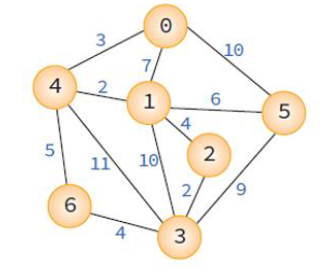
    shortest\_path(&g);

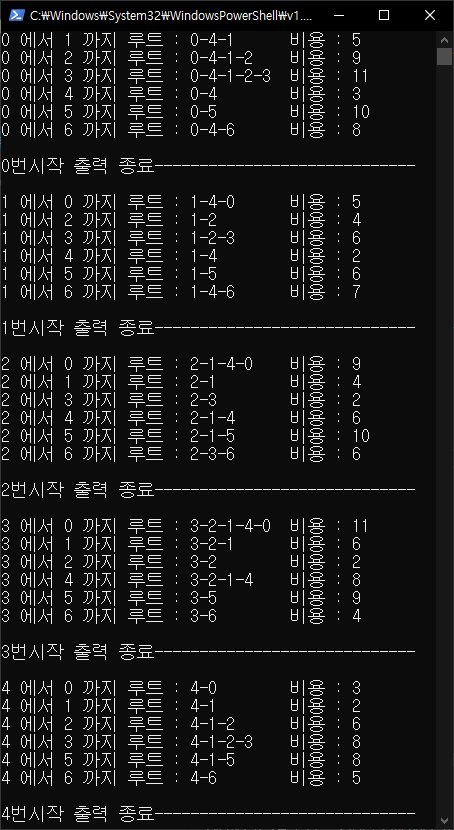
    return 0;

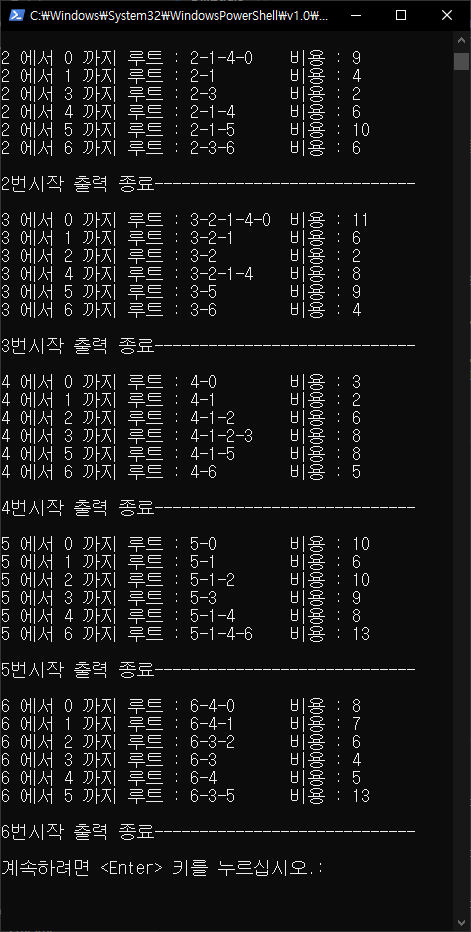
}

실행을 담당할 메인함수 입니다.

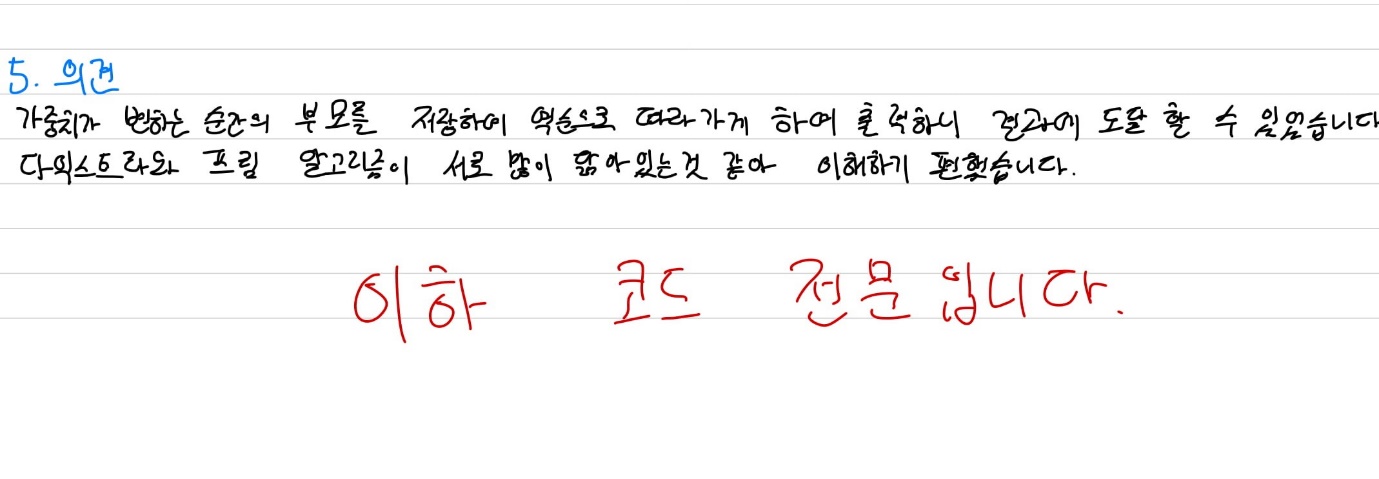
4. 실행 결과







정상 출력되는 것을 확인할 수 있습니다.



#include <limits.h>

#include <stdio.h>

#include <stdlib.h>

#define TRUE 1

#define FALSE 0

#define MAX\_VERTICES 100

#define INF 1000000 //무한대. (연결이 없는 경우)

typedef struct GraphType {

    int n; //정점의 개수

    int weight[MAX\_VERTICES][MAX\_VERTICES];

} GraphType;

int distance[MAX\_VERTICES];

int found[MAX\_VERTICES];

int trace[MAX\_VERTICES];

int r\_trace[MAX\_VERTICES];

int choose(int distance[], int n, int found[]) {

    int i, min, minpos;

    min = INT\_MAX;

    minpos = -1;

    for (i = 0; i < n; i++) {

        if (distance[i] < min && !found[i]) {

            min = distance[i];

            minpos = i;

        }

    }

    return minpos;

}

void shortest\_path(GraphType \*g) {

    int i, u, w, start;

    for (int s = 0; s < g->n; s++) {

        start = s;

        for (i = 0; i < g->n; i++) {

            distance[i] = g->weight[start][i];

            found[i] = FALSE;

            if (g->weight[start][i] != INF)

                trace[i] = start;

        }

        distance[start] = 0;

        for (int tmp = 0; tmp < g->n; tmp++) {

            for (i = 0; i < g->n; i++) {

                u = choose(distance, g->n, found);

                if (tmp == u) {

                    if (start == u)

                        continue;

                    printf("%d 에서 %d 까지 루트 : %d", start, u, start);

                    int n\_route = trace[u];

                    int q = 0;

                    while (1) {

                        if (n\_route == start) {

                            while (q > 0) {

                                printf("-%d", r\_trace[--q]);

                            }

                            break;

                        }

                        r\_trace[q++] = n\_route;

                        n\_route = trace[n\_route];

                    }

                    printf("-%d \t비용 : %d\n", u, distance[u]);

                    for (int k = 0; k < g->n; k++) {

                        distance[k] = g->weight[start][k];

                        found[k] = FALSE;

                    }

                    break;

                }

                found[u] = TRUE;

                for (w = 0; w < g->n; w++) {

                    if (!found[w]) {

                        if (distance[u] + g->weight[u][w] < distance[w]) {

                            distance[w] = distance[u] + g->weight[u][w];

                            trace[w] = u;

                        }

                    }

                }

            }

        }

        printf("\n%d번시작 출력 종료-----------------------------\n\n", s);

    }

}

int main(void) {

    GraphType g = {7,

                   {{0, 7, INF, INF, 3, 10, INF},

                    {7, 0, 4, 10, 2, 6, INF},

                    {INF, 4, 0, 2, INF, INF, INF},

                    {INF, 10, 2, 0, 11, 9, 4},

                    {3, 2, INF, 11, 0, INF, 5},

                    {10, 6, INF, 9, INF, 0, INF},

                    {INF, INF, INF, 4, 5, INF, 0}}};

    shortest\_path(&g);

    return 0;

}