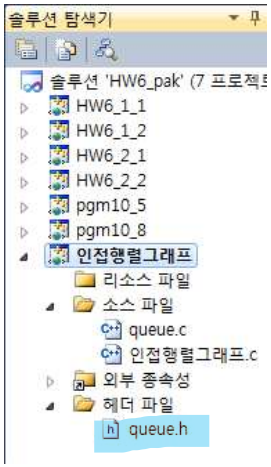
10장 그래프 #1(인접행렬을 이용해 구현한)

//인접행렬그래프.c 소스파일

```
#include <stdio.h>
#include "queue.h"   // <------

#define MAX_VERTICES 50
#define TRUE 1;
#define FALSE 0;

int visited[MAX_VERTICES]; // 전역변수는 0 으로 자동 초기화
typedef struct GraphType {
        int n;
        int adj_mat[MAX_VERTICES][MAX_VERTICES];
} GraphType;

void graph_init(GraphType *g) // 그래프 초기화
{




}

void insert_edge(GraphType *g, int start, int end) // 간선 삽입 연산
{
        if( start >= g->n || end >= g->n ){
                fprintf(stderr,"그래프: 정점 번호 오류");
                return;
        }



}

void insert_vertex(GraphType *g, int v) // 정점 삽입 연산
{
        if (((g->n) + 1) > MAX_VERTICES) {
                fprintf(stderr, "그래프 정점의 개수 초과");
                return;
        }
        g->n++;
}

/* 깊이 우선 탐색
depth_first_search(v)
        v 를 방문되었다고 표시;
        for all u ∈ (v 에 인접한 정점) do
            if (u 가 아직 방문되지 않았으면)
                then depth_first_search(u)
*/
void dfs_mat(GraphType *g, int v) // 깊이 우선 탐색
{
```



```
}

/* 넓이 우선 탐색
breadth_first_search(v)
      v 를 방문되었다고 표시;
      큐 Q 에 정점 v 를 삽입;
      while (not is_empty(Q)) do
            큐 Q 에서 정점 w 를 삭제;
            for all u ∈ (w 에 인접한 정점) do
                  if (u 가 아직 방문되지 않았으면) then      u 를 방문되었다고 표시;
                                                            u 를 큐 Q 에 삽입;

*/
void bfs_mat(GraphType *g, int v) // 넓이 우선 탐색
{




}

 void visited_init()
{
        int i;
        for (i = 0; i < MAX_VERTICES; i++)
                visited[i] = 0;
}
```




```
int main(void)
{
        GraphType g;
        graph_init(&g);

        insert_vertex(&g, 0);
        insert_vertex(&g, 1);
        insert_vertex(&g, 2);
        insert_vertex(&g, 3);

        insert_edge(&g, 0, 1);
        insert_edge(&g, 1, 2);
        insert_edge(&g, 2, 3);
        insert_edge(&g, 3, 0);
        insert_edge(&g, 0, 2);

        printf("₩n 깊이 우선 탐색₩n");
        dfs_mat(&g, 0); printf("₩n");

        visited_init();
        dfs_mat(&g, 1); printf("₩n");

        visited_init();
        dfs_mat(&g, 2); printf("₩n");

        visited_init();
        dfs_mat(&g, 3); printf("₩n");
        printf("₩n 넓이 우선 탐색₩n");

        visited_init();
        bfs_mat(&g, 0); printf("₩n");

        visited_init();
        bfs_mat(&g, 1); printf("₩n");

        visited_init();
        bfs_mat(&g, 2); printf("₩n");

        visited_init();
        bfs_mat(&g, 3); printf("₩n");
}
```

10장 그래프 #2(인접리스트을 이용해 구현한)

//인접리스트그래프.c 소스파일

```
#include <stdio.h>
#include <stdlib.h>2
#include "queue.h"    // <-----

#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 50 // 디버깅할 때는 작은 수로 설정해서 하는 것이 좋습니다!!

int visited[MAX_VERTICES];

typedef struct GraphNode
{
    int vertex;
    struct GraphNode *link;
} GraphNode;

typedef struct GraphType {
     int n;    // 정점의 개수
     GraphNode *adj_list[MAX_VERTICES];
} GraphType;

// 그래프 초기화
void graph_init(GraphType *g)
{




}

// 정점 삽입 연산
void insert_vertex(GraphType *g, int v)
{
     if( ((g->n)+1) > MAX_VERTICES ){
          fprintf(stderr, "그래프: 정점의 개수 초과");
          return;
     }
     g->n++;
}

// 간선 삽입 연산, v를 u의 인접 리스트에 삽입한다.
void insert_edge(GraphType *g, int u, int v)
{
     GraphNode *node;
     if( u >= g->n || v >= g->n ){
          fprintf(stderr, "그래프: 정점 번호 오류");
          return;
     }

     // u에 v를 매단다




     // v에 u를 매단다




}
```

```
/* 깊이 우선 탐색 depth_first_search(v)
        v를 방문되었다고 표시;
        for all u ∈ (v에 인접한 정점) do
        if (u가 아직 방문되지 않았으면)
        then depth_first_search(u) */
void dfs_list(GraphType *g, int v) // 깊이 우선 탐색(인접 리스트)
{
    GraphNode *w;
    visited[v] = TRUE;          // 정점 v의 방문 표시
    printf("%d ", v);          // 방문한 정점 출력




}

/* 넓이 우선 탐색 breadth_first_search(v)
     v를 방문되었다고 표시;
     큐 Q에 정점 v를 삽입;
     while (not is_empty(Q)) do
          큐 Q에서 정점 w를 삭제;
          for all u ∈ (w에 인접한 정점) do
          if (u가 아직 방문되지 않았으면) then     u를 방문되었다고 표시;
                                                   u를 큐 Q에 삽입;        */
void bfs_list(GraphType *g, int v) // 너비 우선 탐색(인접 리스트)
{
     GraphNode *w;
     QueueType q;
     init(&q);                    // 큐 초기 화
     visited[v] = TRUE;          // 정점 v 방문 표시
     printf("%d ", v);
     enqueue(&q, v);             // 시작정점을 큐에 저장
     while(!is_empty(&q)){
          v = dequeue(&q);          // 큐에 저장된 정점 선택









     }
}

void visited_init()
{
     int i;
     for (i = 0; i < MAX_VERTICES; i++)
          visited[i] = 0;
}
```

```
int main(void)
{
     GraphType g;
     graph_init(&g);

     insert_vertex(&g, 0);
     insert_vertex(&g, 1);
     insert_vertex(&g, 2);
     insert_vertex(&g, 3);

     insert_edge(&g, 0, 1);
     insert_edge(&g, 1, 2);
     insert_edge(&g, 2, 3);
     insert_edge(&g, 3, 0);
     insert_edge(&g, 0, 2);
```

```
     printf("₩n 깊이 우선 탐색₩n");
     dfs_list(&g, 0);    printf("₩n");

     visited_init();
     dfs_list(&g, 1);    printf("₩n");

     visited_init();
     dfs_list(&g, 2);    printf("₩n");

     visited_init();
     dfs_list(&g, 3);    printf("₩n");
```

```
     printf("₩n 넓이 우선 탐색₩n");

     visited_init();
     bfs_list(&g, 0);    printf("₩n");

     visited_init();
     bfs_list(&g, 1);    printf("₩n");

     visited_init();
     bfs_list(&g, 2);    printf("₩n");

     visited_init();
     bfs_list(&g, 3);    printf("₩n");
}
```

10장 그래프 #3(Kruskal의 MST 알고리즘)

```
#include <stdio.h>
#define MAX_VERTICES 100

int parent[MAX_VERTICES];       // 부모 노드
int num[MAX_VERTICES];          // 각 집합의 크기

// 초기화
void set_init(int n) //
{
        생략;
}

// vertex 가 속하는 집합을 반환한다.
// 대표 정점을 반환한다)
int set_find(int vertex)
{
        생략;
}

// 두개의 원소가 속한 집합을 합친다.
// 대표정점 1 과 대표정점 2 를 파라미터로 받아 하나의 집합으로 만든다.
void set_union(int s1, int s2)
{
        생략;
}

#define MAX_ELEMENT 100
typedef struct {
        int key;  // 간선의 가중치   // weight
        int u;              // 정점 1
        int v;              // 정점 2
} element;
typedef struct {
        element heap[MAX_ELEMENT];
        int heap_size;
} HeapType;

// 초기화 함수
init(HeapType *h)
{
        생략;
}

// 삽입 함수
void insert_min_heap(HeapType *h, element item)
{
        생략;
}

// 삭제 함수
element delete_min_heap(HeapType *h)
{
        생략;
}
```

```
//u, v, weight 를 element 로 히프에 삽입
void insert_heap_edge(HeapType *h, int u, int v, int weight)
{
        // 힙에
        element e;
        e.u = u;
        e.v = v;
        e.key = weight;
        insert_min_heap(h, e);

}
// 인접 행렬이나 인접 리스트에서 간선들을 읽어서 최소 히프에 삽입
// 현재는 예제 그래프(교재 [그림 10-25])의 간선들을 삽입한다.
void insert_all_edges(HeapType *h)
{
        insert_heap_edge(h,0,1,29);
        insert_heap_edge(h,1,2,16);
        insert_heap_edge(h,2,3,12);
        insert_heap_edge(h,3,4,22);
        insert_heap_edge(h,4,5,27);
        insert_heap_edge(h,5,0,10);
        insert_heap_edge(h,6,1,15);
        insert_heap_edge(h,6,3,18);
        insert_heap_edge(h,6,4,25);
}

// kruskal 의 최소 비용 신장 트리 프로그램
void kruskal(int n)
{
        int edge_accepted=0;    // 현재까지 선택된 간선의 수
        HeapType h;             //최소 히프
        int uset, vset;         // 정점 u 와 정점 v 의 집합 번호
        element e;              // 히프 요소

        init(&h)                // 히프 초기화
        insert_all_edges(&h);   // 히프에 간선들을 삽입
        set_init(n);            // 집합 초기화

        while( edge_accepted < (n-1) )    // 간선의 수 < (n-1)
        {
                e = delete_min_heap(&h);  // 최소 히프에서 삭제 : 가장 key(→weight)가 작은 간선

                }
}

int main(void)
{
        kruskal(7);
}
```

10장 그래프 #4(Prim의 MST 알고리즘)

2020 수정

```c
#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 100
#define INF 1000L

typedef struct GraphType {
    int n; // 정점의 개수
    int weight[MAX_VERTICES][MAX_VERTICES];
} GraphType;

int selected[MAX_VERTICES];
int distance[MAX_VERTICES];

// 최소 dist[v] 값을 갖는 정점을 반환
int get_min_vertex(int n)
{
    int v, i;
    for (i = 0; i <n; i++)
        if (!selected[i]) {
            v = i;
            break;
        }
    for (i = 0; i < n; i++)
        if (!selected[i] && (distance[i] < distance[v])) v = i;
    return (v);
}
//
void prim(GraphType* g, int s)
{
    int i, u, v;

    for (u = 0; u<g->n; u++)
        distance[u] = INF;
    distance[s] = 0;
    for (i = 0; i<g->n; i++) {









    }
}

int main(void)
{
    GraphType g = { 7,
    {{ 0, 29, INF, INF, INF, 10, INF },
    { 29,  0, 16, INF, INF, INF, 15 },
    { INF, 16, 0, 12, INF, INF, INF },
    { INF, INF, 12, 0, 22, INF, 18 },
    { INF, INF, INF, 22, 0, 27, 25 },
    { 10, INF, INF, INF, 27, 0, INF },
    { INF, 15, INF, 18, 25, INF, 0 } }
    };
    prim(&g, 0);
    return 0;
}
```


10장 그래프 #5 (Dijkstra shortest path 알고리즘)

2020 수정

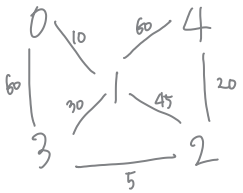
```c
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define TRUE 1
#define FALSE 0
#define MAX_VERTICES    100
#define INF 1000000 /* 무한대 (연결이 없는 경우) */

typedef struct GraphType {
    int n;  // 정점의 개수
    int weight[MAX_VERTICES][MAX_VERTICES];
} GraphType;

int distance[MAX_VERTICES];/* 시작정점으로부터의 최단경로 거리 */
int found[MAX_VERTICES];        /* 방문한 정점 표시 */

int choose(int distance[], int n, int found[])
{
    // 생략
}
void print_status(GraphType* g)
{
    // 생략
}
void shortest_path(GraphType* g, int start)
{
    int i, u, w;
    for (i = 0; i<g->n; i++) /* 초기화 */
    {
        distance[i] = g->weight[start][i];
        found[i] = FALSE;
    }
    found[start] = TRUE;    /* 시작 정점 방문 표시 */
    distance[start] = 0;
    for (i = 0; i<g->n-1; i++) {
        print_status(g);
        u = choose(distance, g->n, found);
        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];
    }
}
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, 0);
}
```

10장 그래프 #6(Floyd의 shortest path 알고리즘)

2020 수정

```c
#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 A[MAX_VERTICES][MAX_VERTICES];

void printA(GraphType *g)
{
    int i, j;
    printf("============================₩n");
    for (i = 0; i<g->n; i++) {
        for (j = 0; j<g->n; j++) {
            if (A[i][j] == INF)
                printf("  * ");
            else printf("%3d ", A[i][j]);
        }
        printf("₩n");
    }
    printf("============================₩n");
}
void floyd(GraphType* g)
{

    int i, j, k;
    for (i = 0; i<g->n; i++)
        for (j = 0; j<g->n; j++)
            A[i][j] = g->weight[i][j];
   printf("초기 상태₩n");
    printA(g);

    for (k = 0; k<g->n; k++) {
        for (i = 0; i<g->n; i++)
            for (j = 0; j<g->n; j++)
                if (A[i][k] + A[k][j] < A[i][j])
                    A[i][j] = A[i][k] + A[k][j];
        printf("%d를 경유해서 다시 계산₩n", k);
      printA(g);
    }
}
int main(void)
{
    GraphType g = { 7,
    {{0, 10, INF, 50, INF},
    {10, 0, 45, 30, 60},
    {INF, 45, 0, 5, 20},
    {50, 30, 5, 0, INF},
    {INF, 60, 20, INF, 0}}
    };
    floyd(&g);
    return 0;
}
```

```
초기 상태
      [0] [1] [2] [3] [4]
--------------------------
[0]|   0  10 INF  60 INF
[1]|  10   0  45  30  60
[2]| INF  45   0   5  20
[3]|  60  30   5   0 INF
[4]| INF  60  20 INF   0
==========================
0를 경유해서 다시 계산
      [0] [1] [2] [3] [4]
--------------------------
[0]|   0  10 INF  60 INF
[1]|  10   0  45  30  60
[2]| INF  45   0   5  20
[3]|  60  30   5   0 INF
[4]| INF  60  20 INF   0
==========================
1를 경유해서 다시 계산
      [0] [1] [2] [3] [4]
--------------------------
[0]|   0  10  55  40  70
[1]|  10   0  45  30  60
[2]|  55  45   0   5  20
[3]|  40  30   5   0  90
[4]|  70  60  20  90   0
==========================
2를 경유해서 다시 계산
      [0] [1] [2] [3] [4]
--------------------------
[0]|   0  10  55  40  70
[1]|  10   0  45  30  60
[2]|  55  45   0   5  20
[3]|  40  30   5   0  25
[4]|  70  60  20  25   0
==========================
3를 경유해서 다시 계산
      [0] [1] [2] [3] [4]
--------------------------
[0]|   0  10  45  40  65
[1]|  10   0  35  30  55
[2]|  45  35   0   5  20
[3]|  40  30   5   0  25
[4]|  65  55  20  25   0
==========================
4를 경유해서 다시 계산
      [0] [1] [2] [3] [4]
--------------------------
[0]|   0  10  45  40  65
[1]|  10   0  35  30  55
[2]|  45  35   0   5  20
[3]|  40  30   5   0  25
[4]|  65  55  20  25   0
```



10장 그래프 #7(위상 정렬)

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
#define MAX_VERTICES 50

//그래프의 연결리스트 표현--------------------------------
typedef struct GraphNode
{
    int vertex;
    struct GraphNode *link;
} GraphNode;

typedef struct GraphType {
    int n; // 정점의 개수
    GraphNode *adj_list[MAX_VERTICES];
} GraphType;

// 그래프 초기화
void graph_init(GraphType *g) {... 생략...}

// 정점 삽입 연산
void insert_vertex(GraphType *g, int v) {... 생략...}

// 간선 삽입 연산, v를 u의 인접 리스트에 삽입한다.
void insert_edge(GraphType *g, int u, int v) {... 생략...}

//스택----------------------------------------------------
#define MAX_STACK_SIZE 100
typedef int element;
typedef struct {
    element stack[MAX_STACK_SIZE];
    int top;
} StackType;

// 스택 초기화 함수
void init(StackType *s) {... 생략...}

// 공백 상태 검출 함수
int is_empty(StackType *s) {... 생략...}

// 포화 상태 검출 함수
int is_full(StackType *s) {... 생략...}

// 삽입함수
void push(StackType *s, element item) {... 생략...}

// 삭제함수
element pop(StackType *s) {... 생략...}

// 피크함수
element peek(StackType *s) {... 생략...}
```

```
// 위상정렬을 수행한다.
int topo_sort(GraphType *g)
{
    int i;
    StackType s;
    GraphNode *node;

    // 모든 정점의 진입 차수를 계산
    int *in_degree = (int *)malloc(g->n* sizeof(int));
    for (i = 0; i < g->n; i++)          // 초기화
        in_degree[i] = 0;
    [redacted]

    }

    // 진입 차수가 0인 정점을 스택에 삽입
    init(&s);



    }
    // 위상 순서를 생성
    while (!is_empty(&s)) {
        int w;
        w = pop(&s);
        printf("%d", w);      //정점 출력









    }
    free(in_degree);
}
//
main()
{
    GraphType g;

    graph_init(&g);
    insert_vertex(&g, 0);
    insert_vertex(&g, 1);
    insert_vertex(&g, 2);
    insert_vertex(&g, 3);
    insert_vertex(&g, 4);
    insert_vertex(&g, 5);

    //정점 0의 인접 리스트 생성
    insert_edge(&g, 0, 2);
    insert_edge(&g, 0, 3);
    //정점 1의 인접 리스트 생성
    insert_edge(&g, 1, 3);
    insert_edge(&g, 1, 4);
    //정점 2의 인접 리스트 생성
    insert_edge(&g, 2, 3);
    insert_edge(&g, 2, 5);
    //정점 3의 인접 리스트 생성
    insert_edge(&g, 3, 5);
    //정점 4의 인접 리스트 생성
    insert_edge(&g, 4, 5);
    //위상 정렬
    topo_sort(&g);
}
```

HW 6 : 그래프

■ HW6_0

10장 그래프의 Quiz/연습문제 중 일부

■ HW6_1(그래프 추상 데이터 타입의 구현)

□ HW6_1_1(인접 행렬 이용)

(1) 인접 행렬로 표현된 무방향 그래프에서 간선을 삽입 또는 삭제하는 함수를 구현한다.
간선 삽입을 위한 함수는 교재 10.3 절의 코드를 그대로 사용한다.

(교재에서의 간선 삽입 함수에서와 마찬가지로) 삭제 함수에서도 입력으로 주어지는 간선이 실제로 존재하는 유효한 간선인지를 먼저 확인한 후 삭제를 수행하도록 한다. (GraphType 의 정의는 10.3 절의 코드를 참조할 것)

```
insert_edge(GraphType *g, int u, int v) // 교재에 있는 코드 그대로
delete_edge(GraphType *g, int u, int v)
```

(2)
다음과 같은 무방향 그래프에 대한 정보가 다음과 같은 형식으로 텍스트 파일에 저장된다고 가정하자.

```
5       (정점들의 개수를 나타냄; 정점의 ID는 0부터 1씩 증가하는 정수라고 가정함)
0 1     (간선들을 한 줄에 하나씩 나열함; 이 때 간선들은 특정 순서로 정렬되어 있지 않다고 가정함)
0 2
0 4
1 2
2 3
2 4
3 4
(EOF)
```

위와 같은 파일을 읽어서 인접 행렬로 표현된 그래프에 저장하거나,
반대로 인접 행렬로 표현된 그래프를 파일에 저장하는(형식은 위의 입력파일과 같게) 다음 함수를 각각 구현하시오.

```
read_graph(GraphType *g, char *filename)
write_graph(GraphType *g, char *filename)
```

(3) 위 함수들을 테스트하기 위한 테스트 프로그램을 작성하고 위 그래프에 대해 실행한 결과를 출력하시오. 즉, read_graph() 함수를 이용하여 그래프 구조를 생성하고 insert_edge()와 delete_edge() 함수를 이용하여 하나 이상의 간선들을 삽입 및 삭제해보라. 입력파일에서 읽은 후, 그리고 삽입과 삭제를 할 때마다 write_graph() 함수를 이용하여 그래프를 화면(stdout)에 출력하여보라
또한 write_graph()함수를 이용하여 그래프를 출력 파일에도 출력하여보라

출력시 간선내의 정점의 순서나 간선의 순서는 바뀔 수 있음에 유의하자.

□ HW6_1_2(인접 리스트 이용)

그래프를 인접 리스트로 표현한다고 가정하고, 위 HW6_1_1 의 함수들과 테스트 프로그램을 작성하시오

참고 1: 교재 p375 의 insert_edge 는 방향 그래프에 대한 간선 삽입을 보여준다. 수업시간에 다루었듯이 무방향그래프의 함수로 수정한다.
참고 2: delete_edge 함수를 위해 4 장에서 배운 remove_node 함수를 사용할 수 있다.

■ HW6_2(그래프 탐색, 신장 트리)

+ 간선

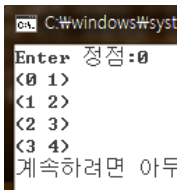
□ HW6_2_1a(깊이 우선 탐색)(인접행렬 이용)

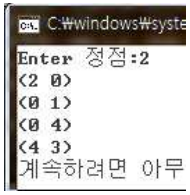
1

인접 행렬로 표현된 그래프에 대해 **깊이 우선 탐색**을 수행하면서 **신장 트리를 구하는 프로그램**을 다음과 같이 작성하시오.

- 그래프 정보는 HW6_1_1 에서 정의된 것과 같은 형식과 내용의 텍스트 파일로 제공된다고 가정하고, HW6_1_1 에서 작성한 read_graph() 함수를 이용하여 인접 행렬로 읽어들인다.
- 특정 정점이 주어질 때 이 정점에서부터 **깊이 우선 탐색을** 시작한다.
- 탐색 과정에서 사용되는 간선들을 차례대로 화면에 출력한다. (이 간선들이 깊이 우선 신장 트리를 구성하는 간선들이 됨)
- 10.5 절과 10.6 절의 알고리즘 및 프로그램 코드를 참고할 것

입력되는 정점을 0, 1, 2, 3, 4 로 바꾸어서 간선을 출력해보라.(자신의 예측과 비교해보라) 실행예들의 일부는 다음과 같다.

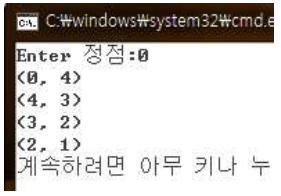


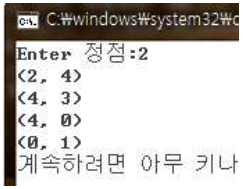


□ HW6_2_1b(깊이 우선 탐색)(인접 리스트 이용)

2

인접 리스트로 표현된 그래프에 대해 **깊이 우선 탐색**을 수행하면서 신장 트리를 구하는 프로그램을 HW6_2_1a 와 유사하게 작성하시오. 모든 정점에 대해서 실행시켜보고 자신의 예측과 결과를 비교한다.

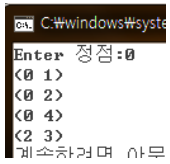


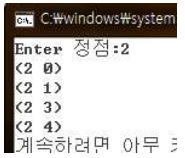


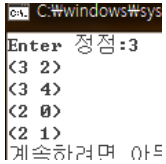
□ HW6_2_2a(너비 우선 탐색)(인접행렬 이용)

3

인접 행렬로 표현된 그래프에 대해 **너비** 우선 탐색을 수행하면서 신장 트리를 구하는 프로그램을 HW6_2_1a와 유사하게 작성하시오. 모든 정점에 대해서 실행시켜보고 자신의 예측과 결과를 비교한다.



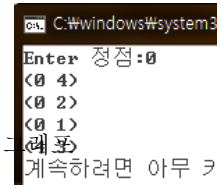


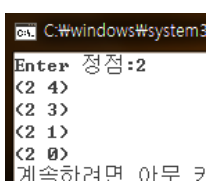


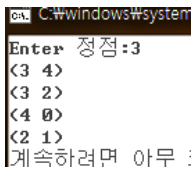
4

□ HW6_2_2b(너비 우선 탐색)(인접리스트 이용)

인접 리스트로 표현된 그래프에 대해 **너비** 우선 탐색을 수행하면서 신장 트리를 구하는 프로그램을 HW6_2_1a와 유사하게 작성하시오. 모든 정점에 대해서 실행시켜보고 자신의 예측과 결과를 비교한다.







■ HW6_3(최소 비용 신장 트리)

□ HW6_3_1 Kruskal

(1) Kruskal 의 최소 비용 신장 트리를 구현한 프로그램 11.8 을 다음과 같이 수정해 보시오.

- 그래프를 다음과 같이 인접 행렬로 표현한다.

```
#define MAX_VERTICES 100
typedef struct GraphType {
    int n;                                          // 정점의 개수
    int adj_mat[MAX_VERTICES][MAX_VERTICES]; // 간선의 가중치 저장
} GraphType;
.
.
.

main()
{
    GraphType g, t;          // 입력 그래프, 결과 트리

    graph_init(&g);
    read_graph(&g, "input.txt");

    graph_init(&t);
    t.n = g.n;

    kruskal(&g, &t);
    printf("파일로 출력:₩n");
    write_graph(&t, "output.txt");
    write_graph(&t, NULL);    // to stdout
}
```

- 위와 같은 인접 행렬을 사용하여 입력 그래프와 출력 그래프(최소 비용 신장 트리)를 나타내려 한다. (인접 행렬의 각 원소는 두 정점 사이의 간선의 가중치를 저장하되, 간선이 존재하지 않을 경우 미리 정의된 매우 큰 정수(INF)를 저장한다. 이를 위해 프로그램 10.9 의 weight 배열을 참고할 것)

  이를 위해 아래의 함수를 구현한다.

- **graph_init()**: n 은 0 으로, adj_mat[][]는 아래와 같이 미리 정의 된 매우 큰 정수(INF)로 초기화한다.

  ```
  #define INF 1000L
  ```

- **read_graph()**: 그래프 정보는 HW6_1_1 에서 정의된 것과 같은 형식의 텍스트 파일로 제공된다고 가정한다. 단, 각 간선에 대해 그것의 가중치도 함께 저장한다 (예: 간선 (1, 2)의 가중치가 30 인 경우 1 2 30 을 저장함). 그래프 정보를 인접 행렬로 읽어들이기 위해 HW6_1_1 에서 작성한 read_graph() 함수를 수정하여 사용한다.

input.txt

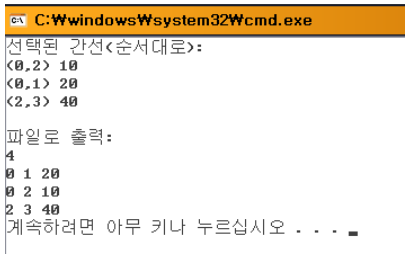
```
4       (정점들의 개수를 나타냄; 정점의 ID 는 0 부터 1 씩 증가하는 정수라고 가정함)
0 1 20  (간선과 가중치를 한 줄에 나열함; 이 때 간선들은 특정 순서로 정렬되어 있지않다고 가정함)
1 2 30
2 3 40
3 0 50
0 2 10
(EOF)
```


- **(challenge)** Union-Find 연산을 구현하기 위해 프로그램 11.7 을 활용하되, 배열 num[]은 제외하고 parent[]만 사용하게 수정하라(hint: num[]의 정보를 parent[]에 통합하여 저장하는 방법을 생각해볼 것)

- **kruskal()**: 교재의 kruskal 을 조금 수정한다. 입력 그래프 g 와 출력 그래프 t 를 매개변수로 한다. Kruscal 알고리즘을 구현하여 g 로부터 최소 비용 신장 트리인 t 를 만들어낸다. 최소 비용 신장 트리를 구하는 과정에서 선택되는 간선(과 가중치)를 출력하도록 하라.

- **write_graph()**: 최소 비용 신장 트리가 생성된 후 이를 파일(stdout 와 출력파일)로 출력해 보려한다. (HW6_1_1 에서 작성한 write_graph() 함수를 수정하여 구현한다.)

위의 예로 제공된 텍스트 파일의 경우 실행결과는 다음과 같다.



Output.txt

```
4
0 1 20
0 2 10
2 3 40
```

(2) 작성한 프로그램을 p.437 Quiz 01 의 그래프에 대해 실행해 보고 결과를 출력하시오.

8/10(자료구조 11주차이후: 그래프

□ HW6_3_2 Prim

교재의 프로그램 11.9 을 조금 수정(selected[]와 dist[]의 값을 출력하는 문장을 삽입)하여 HW6_3_2 를 작성하였다. adj_mat 는 수업시간에 다룬 예제를 이용하여 초기화하였다. 실행결과는 아래와 같다.

- selected 와 dist 의 변화를 이해해본다.

<주어진 프로그램 실행>

```
C:₩Windows₩system32₩cmd.exe

0 선택
selected[] =       1     0     0     0     0     0
dist[] =           0    10  1000    20    70  1000

1 선택
selected[] =       1     1     0     0     0     0
dist[] =           0    10    50    20    60  1000

3 선택
selected[] =       1     1     0     1     0     0
dist[] =           0    10    50    20    60    80

2 선택
selected[] =       1     1     1     1     0     0
dist[] =           0    10    50    20    40    80

4 선택
selected[] =       1     1     1     1     1     0
dist[] =           0    10    50    20    40    80

5 선택
selected[] =       1     1     1     1     1     1
dist[] =           0    10    50    20    40    80
계속하려면 아무 키나 누르십시오 . . .
```

주어진 프로그램은 시작정점부터 시작해서 간선을 선택하면서 정점을 출력한다.
아래의 실행결과와 같이 **<u>간선을 출력하도록</u>** 프로그램을 수정하라

힌트

- 정점을 선택했을 때 관련된 간선정보(다른쪽 정점?)를 저장한다.(배열 사용?)
- 아래의 실행예에서 간선 (0, 0) 0에 대한 출력은 생략해도 좋겠으나 프로그램의 이해를 위해 포함시켰다.

```
C:₩Windows₩system32₩cmd.exe

<0 0> 0
selected[] =       1     0     0     0     0     0
dist[] =           0    10  1000    20    70  1000

<0 1> 10
selected[] =       1     1     0     0     0     0
dist[] =           0    10    50    20    60  1000

<0 3> 20
selected[] =       1     1     0     1     0     0
dist[] =           0    10    50    20    60    80

<1 2> 50
selected[] =       1     1     1     1     0     0
dist[] =           0    10    50    20    40    80

<2 4> 40
selected[] =       1     1     1     1     1     0
dist[] =           0    10    50    20    40    80

<3 5> 80
selected[] =       1     1     1     1     1     1
dist[] =           0    10    50    20    40    80
계속하려면 아무 키나 누르십시오 . . .
```


■ HW6_4(최단 경로)

□ HW6_4_1(Dijkstra Algorithm 구현)

(1) Dijkstra 의 최단 경로 알고리즘을 구현한 프로그램 11.10 을 다음과 같이 수정해 보시오.

- 그래프는 HW6_3 과 같은 인접 행렬로 표현한다.
- 그래프 정보가 아래와 같이 텍스트 파일로 저장된다고 가정하고 이것을 인접 행렬로 읽어들인다. (아래의 정점의 개수와 간선(가중치)를 이용하여 그래프 그림을 그려보라)

```
5
0 1 10
0 3 60
1 3 30
1 2 45
1 4 50
2 3 5
2 4 20
```

- 프로그램 11.10 을 이용, 그리고 HW6_3 에서의 read_graph 함수를 이용하여 아래처럼 시작 정점부터 각 정점에 대한 **<u>최단 경로의 길이</u>**를 구하도록 프로그램을 작성하라. 시작 정점을 0 과 4 로 한 실행결과를 보여준다.

```
C:₩windows₩system32₩cm
0 -> 1<10>
0 -> 3<40>
0 -> 2<45>
0 -> 4<60>
계속하려면 아무 키나
```

```
C:₩windows₩system32₩cmd.exe
4 -> 2<20>
4 -> 3<25>
4 -> 1<50>
4 -> 0<60>
계속하려면 아무 키나 누르
```

Step 1

Step 2

- 다시 이를 다음과 같이 수정하라. 각 정점에 대한 **<u>최단 경로를 최단 경로의 길이와 함께</u>** 순서대로 출력한다. (Dijkstra 의 알고리즘에서 어떤 정점에 대한 최단 경로가 발견될 때 그 경로 상에 있는 그 정점의 바로 앞 정점을 알 수 있다. 이것을 별도의 배열 previous[]에 저장하고 이를 활용한다.)

- 시작 정점을 0 과 4 로 한 실행결과는 다음과 같다

```
C:₩windows₩system32₩cmd.exe
0 -> 1  <10>
0 -> 1 -> 3  <40>
0 -> 1 -> 3 -> 2  <45>
0 -> 1 -> 4  <60>
계속하려면 아무 키나 누르십시
```

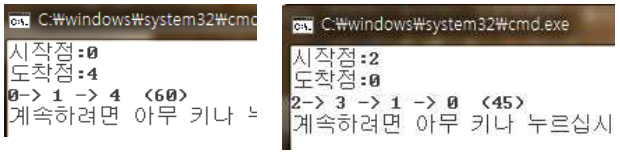
```
C:₩windows₩system32₩cmd.exe
4 -> 2  <20>
4 -> 2 -> 3  <25>
4 -> 1  <50>
4 -> 1 -> 0  <60>
계속하려면 아무 키나 누르십
```

□ (challenge)HW6_4_2(성능 개선)

distance 배열 대신 우선순위 큐를 사용하도록 하면 매우 큰 그래프에 대해 효율적으로 알고리즘을 구현할 수 있다. 프로그램 10.8과 유사하게 Min Heap을 사용하여 각 정점의 distance 정보를 저장하고 알고리즘의 각 단계에서 Min Heap으로부터 최소 distance 값을 갖는 정점을 찾도록 Dijkstra의 알고리즘을 구현하시오. distance 배열과 found 배열은 사용할 필요가 없으므로 삭제한다. (프로그램 10.8, 프로그램 8.5, 프로그램 8.1을 참고할 것)

□ HW6_4_3(지하철역 간 최단거리 구하기)

이제 Dijkstra의 알고리즘을 이용하여 하나의 시작점부터 하나의 도착점까지의 경로와 최단거리를 출력하려한다. 위의 HW6_4_1의 프로그램을 수정하여 다음과 같은 실행결과를 내는 프로그램을 구현하라.



이제 위의 프로그램을 이용하여 지하철역 간 최단거리를 구하는 프로그램을 작성하자. (문제를 간단히 하기 위해서 호선의 구분도 없고 그냥 지하철역을 자유롭게 갈아탈 수 있다고 가정하자)

출발지와 도착지를 읽어 들여서 최단거리를 계산하고 경유 역과 최단거리를 보여준다.