

## 6장 연습문제

### 6.1절

1. 그림 6.16의 다중그래프에 오일러 행로가 존재하는가? 존재한다면, 오일러 행로를 하나 보여라.

>> 각 정점의 차수가 짝수인 경우이므로 오일러 행로 존재

- 0-2-3-2-3-0-1-0
- 0-3-2-3-2-0-1-0
- 2-0-1-0-3-2-3-2
- 이 외에도 많음

2. 그림 6.17의 다이그래프에 대해 다음 물음에 답하라.

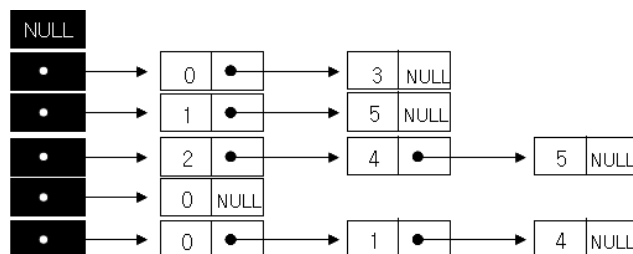
(a) 각 정점의 진입 차수와 진출 차수

	in-degree	out-degree
0	3	0
1	2	2
2	1	2
3	1	3
4	2	1
5	2	3

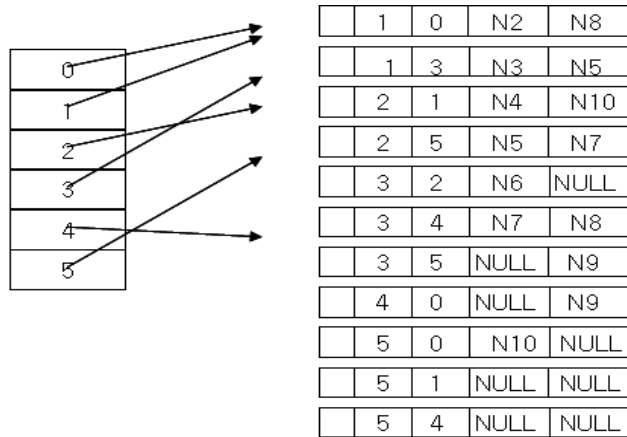
(b) 인접 행렬

0	0	0	0	0	0
1	0	0	1	0	0
0	1	0	0	0	1
0	0	1	0	1	1
1	0	0	0	0	0
1	1	0	0	1	0

(c) 인접 리스트 표현



(d) 인접 다중 리스트 표현



--> 리스트 :

정점 0 : N1->N8->N9

정점 1 : N1->N2->N3->N10

정점 2 : N3->N4->N5

정점 3 : N2->N5->N6->N7

정점 4 : N6->N8->N11

정점 5 : N4->N7->N9->N10->N11

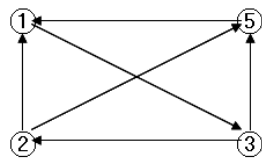
>> 방향 그래프이므로 HEAD와 TAIL을 알고 자신의 위치를 안다면 in-degree와 out-degree를 알수 있다.

(e) 강력 연결 요소

- ⑦

- ④

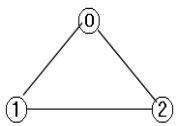
-



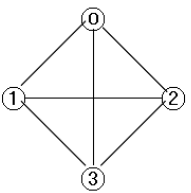
4. 한 개, 두 개, 세 개, 네 개, 다섯 개의 정점으로 된 무방향 완전 그래프를 그려라. n개의 정점을 갖는 완전 그래프의 간선의 수가  $n(n-1)/2$ 임을 증명하라.

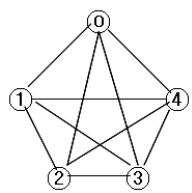
- ①

-



- ① — ①

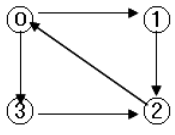




>> 정점의 수를 n이라 할때 self loop를 제외한 각 정점마다  $v_0=n-1, v_1=n-2, v_3=n-3 \dots \dots v_{n-1}=0$ 이다.

즉 모든 간선의 수는  $(n-1)+(n-2)+(n-3)+\dots\dots+0=\sum_{i=0}^{n-1} v_i=n(n-1)/2$ 이다.

5. 아래의 방향 그래프가 강력 연결되었는가? 모든 단순경로를 나열하라.



- 서로 다른 두 정점의 모든 쌍에 대해 방향 경로가 존재하면 그 그래프는 강력연결되었다고 함
- 위의 그래프는 모든 쌍에 대해 경로가 존재하므로 강력 연결되었음
- 단순경로는 한 경로상에 있는 모든 정점들이 서로 다르고, 처음과 마지막 정점은 같을수 있다.

- <0 1>, <0 1 2>, <0 1 2 0>, <0 3>, <0 3 2>, <0 3 2 0>
- <1 2>, <1 2 0>, <1 2 0 1>, <1 2 0 3>
- <2 0>, <2 0 1>, <2 0 1 2>, <2 0 3>, <2 0 3 2>
- <3 2>, <3 2 0>, <3 2 0 3>, <3 2 0 1>

6. 문제 5의 그래프를 인접 행렬, 인접 리스트, 인접 다중리스트로 표현해 보라.

(a) adjacency matrix

0	1	0	1
0	0	1	0
1	0	0	0
0	0	1	0

(b) adjacency lists

0

•

→

1

•

→

3

NULL

1

•

→

2

NULL

2

•

→

0

NULL

3

•

→

2

NULL

(c) adjacency multilists

0

→

N1

0

1

N2

N3

1

→

N2

0

3

N4

N5

2

→

N3

1

2

NULL

N4

3

→

N4

2

0

N5

NULL

N5

3

3

NULL

NULL

리스트 : 정점 0 : N1->N2->N4

정점 1 : N1->N3

정점 2 : N3->N4->N5

정점 3 : N2->N5

10~11. 다음 사항을 위한 C함수를 작성하라.

- (a) 무방향 그래프를 위한 정점의 수와 간선들을 하나씩 읽어들인다.
- (b) 그래프를 위한 연결 인접 리스트를 만든다. (두 번 입력되는 간선은 없다고 가정)
- (c) 생성된 인접리스트를 이용하여 역 인접 리스트를 생성하라.
- (d) 인접 리스트와 역 인접 리스트를 인쇄하는 함수를 작성하라.

```
/*          make_lists.c :                               */
/*          undirected graph에서 정점의 수와 간선들을 하나씩 읽어서      */
/*          (inverse) adjacency list를 만들고 이를 인쇄하는 프로그램   */
/*                                                                    */

#include <stdio.h>
#include <stdlib.h>          /*for malloc(), exit()*/

#define MAX_VERTICES 50      /*maximum size of vertex*/
#define IS_FULL(ptr) (!ptr)  /*determine available memory*/

/*node struct prototype*/
typedef struct node *node_pointer;
struct node {
    int vertex;
    node_pointer link;
}node;
/*구조체 리스트 배열*/
node_pointer graph[MAX_VERTICES];
node_pointer inverse_graph[MAX_VERTICES];
int vertices;          /*정점의 수*/

void read_graph(node_pointer *headnode); /*input from user*/
int insert_graph(node_pointer *headnode, int vertex1, int vertex2); /*make list*/
void inverse_adjacency_lists(int vertices); /*create inverse adjacency lists*/
void print_graph(node_pointer *graph); /*print lists*/

void free_memory(node_pointer *ptr); /*memory해제 함수*/

int main() {
    int i;

    read_graph(graph);
    printf("WnWn----- Adjacency lists -----Wn");
    print_graph(graph);

    inverse_adjacency_lists(vertices);
    printf("WnWnWn----- Inverse Adjacency lists -----Wn");
    print_graph(inverse_graph);

    /*메모리 해제*/
    for(i = 0; i<vertices; i++) {
        free_memory(&graph[i]);
        free_memory(&inverse_graph[i]);
    }
    return 0;
}
/*user로부터 정점의 수와 간선을 하나씩 입력받아*/
/*insert_graph의 인자로 값을 넘겨줌*/
```

```

void read_graph(node_pointer *headnode) {
    int i;
    int repetition;                                /*중복된 간선 체크*/
    int vertex, vertex1, vertex2;                  /*간선들을 읽어들이는 변수*/
    int maxedge;                                    /*최대 읽어들이 수 있는 간선의 수*/
    int count = 0;                                  /*입력받은 간선수를 계산*/

    printf("Input the number of vertex(50보다 작게 입력하세요) : ");
    scanf("%d", &vertices);

    maxedge = vertices *(vertices - 1)/2;
    printf("입력 종료 => -1, Vertex start = 0Wn");

    while(count++ < maxedge) { /*최대 간선수가 되거나 사용자 입력 종료시까지*/
        printf("Insert edge(vertex1 vertex2) : ");
        scanf("%d", &vertex1);
        if(vertex1 == -1) /*user 입력 종료 체크*/
            break;
        scanf("%d", &vertex2);

        vertex = vertex1;
        i = 0;
        repetition = 0;

        /*undirected graph이므로 양쪽에 모두 추가*/
        /*directed graph로 사용시 while반복문을 사용 안하면 됨*/
        /*repetition이 있다면 입력 종료*/
        while((i++ < 2) && !repetition) {
            if((repetition = insert_graph(&graph[vertex], vertex, vertex2))) {
                count -= 1;
            }
            vertex = vertex2;
            vertex2 = vertex1;
        }
    }
}

/*리스트에 간선이 있는 정점을 추가 하는 함수*/
int insert_graph(node_pointer *headnode, int vertex1, int vertex2) {

    int repetition = 0;
    node_pointer element, trail, lists;

    if(!(*headnode)) { /*노드가 null이면 바로 head에 추가*/
        lists = (node_pointer)malloc(sizeof(node));
        lists->vertex = vertex2;
        lists->link = NULL;
        *headnode = lists;
    }

    else {
        for(trail = *headnode; trail; trail = trail->link) { //edge가 존재 하는지 확인
            element = trail;
            /*중복 간선 존재*/
            if(trail->vertex == vertex2) {
                printf("The edge is already exist!Wn");
            }
        }
    }
}

```

```

        repetition = 1;
        break;
    }
}
if(!repetition) {    /*중복되는 간선이 없으면 추가*/
    lists = (node_pointer) malloc (sizeof(node));
    if(IS_FULL(lists)) {
        fprintf(stderr, "The memory is full\n");
        exit(1);
    }
    lists->vertex = vertex2;
    lists->link = NULL;
    element->link = lists;
}
/*간선의 중복 여부 반환*/
else
    return repetition;
}
return repetition;
}

void inverse_adjacency_lists(int vertices) {

    int i;
    node_pointer element;

    /*인접리스트의 각 정점의 리스트 값들을 읽어서 역인접리스트에 하나씩 추가하면서 생성*/
    for(i = 0; i<vertices; i++) {
        for(element = graph[i]; element; element = element->link) {
            insert_graph(&inverse_graph[element->vertex], element->vertex, i);
        }
    }
}

void print_graph(node_pointer *graph) {

    int i;
    node_pointer ptr;

    for(i = 0; i<vertices; i++) {
        ptr = graph[i];
        printf("Head[%d] : ", i);

        for(;ptr;ptr = ptr->link) {
            if(!(ptr->vertex == -1)) {
                printf("%4d", ptr->vertex);
            }
        }
    }
}

```

```

        printf("Wn");
    }
}

/*메모리 해제 함수*/
void free_memory(node_pointer *ptr){
    /*ptr에 의해 참조되는 리스트를 하나씩 제거*/
    node_pointer trail;
    while(*ptr) {
        trail = *ptr;
        *ptr = (*ptr)->link;
        free(trail);
    }
}

```

## 6.2절

1~2. dfs 와 bfs 함수가 그래프의 인접 행렬 표현을 사용하도록 재작성하라.

```

/*      dfs&bfs.c :                                  */
/*      dfs & bfs 가 adjacency matirx를 사용하도록 작성 */
/*                                                    */

#include <stdio.h>
#include <string.h>          /*for memset()*/
#include <stdlib.h>          /*for exit()*/

#define TRUE 1

#define MAX_VERTICES 8      /*maximum size of vertex*/

int adj_matrix[][MAX_VERTICES] = {
    0, 1, 1, 0, 0, 0, 0, 0,
    1, 0, 0, 1, 1, 0, 0, 0,
    0, 0, 0, 0, 0, 1, 1, 0,
    0, 1, 0, 0, 0, 0, 0, 1,
    0, 1, 0, 0, 0, 0, 0, 1,
    0, 0, 1, 0, 0, 0, 0, 1,
    0, 0, 1, 0, 0, 0, 0, 1,
    0, 0, 0, 1, 1, 1, 1, 0,
};

/*queue array*/
int queue[MAX_VERTICES];
/*grobal queue variable, initialize queue*/
int front = -1;
int rear = -1;
/*방문 여부를 확인하는 전역 배열*/
short int dfs_visited[MAX_VERTICES];
short int bfs_visited[MAX_VERTICES];

void dfs(int v);
void bfs(int v);

```



```

/*queue fuction prototype*/
void addq(int *rear, int vertex);
int deleteq(int *front, int rear);

int main() {
    /*모두 0으로 초기화*/
    memset(&dfs_visited, 0, sizeof(dfs_visited));
    memset(&bfs_visited, 0, sizeof(bfs_visited));

    printf("----- Depth first search(DFS) algorithm -----Wn");
    dfs(0);

    printf("WnWn----- Breath first search(BFS) algorithm -----Wn");
    bfs(0);
    printf("WnWnWn");

    return 0;
}

void dfs(int v) {
    /*dfs of a graph beginning with vertex v*/
    int i;

    dfs_visited[v] = TRUE;
    printf("%5d", v);

    for(i = 0; i<MAX_VERTICES; i++) {
        /*아직 방문하지 않았으면서 경로가 존재*/
        if(!dfs_visited[i] && adj_matrix[v][i])
            dfs(i);
    }
}

void bfs(int v) {
    /*bfs of a graph beginning with vertex v*/
    int i;

    printf("%5d", v);
    bfs_visited[v] = TRUE;

    addq(&rear, v);

    while(1) {
        /*remove queue*/
        v = deleteq(&front, rear);
        for(i = 0; i<MAX_VERTICES; i++)
            /*아직 방문하지 않았으면서 경로 존재*/
            if(!bfs_visited[i] && adj_matrix[v][i]) {
                printf("%5d", i);
                addq(&rear, i);
                bfs_visited[i] = TRUE;
            }
        if(front == rear) /*큐가 공백상태이면 end*/
            break;
    }
}

```

```

}

void addq(int *rear, int vertex) {
    /*queue에 vertex 삽입*/
    if(*rear == MAX_VERTICES - 1) {
        fprintf(stderr, "Queue is full\n");
        exit(1);
    }
    queue[++*rear] = vertex;
}

int deleteq(int *front, int rear) {
    /*queue의 앞에서 원소 삭제*/
    if(*front == rear) {
        fprintf(stderr, "Queue is empty");
        exit(1);
    }
    return queue[++*front];
}

```

3. G를 연결 무방향 그래프라 하자. G의 어떤 간선도 두 개 이상의 이중결합 요소에 포함될 수 없음을 보여라. G의 한 정점이 두 개 이상의 이중결합 요소에 포함될 수 있는가?

- 동일한 그래프에 속하는 두 개의 이중결합 요소는 많아야 한 개의 정점만을 공통으로 가질 수 밖에 없다. 즉 하나의 단절점으로 인해 두 개의 이중결합 요소로 구분되어지기 때문이다.  
이는 하나의 정점만을 공통으로 가지기 때문에 한 그래프 내에서 하나의 간선이 둘 이상의 이중결합 요소에 포함 될 수 없음을 나타낸다.
- 단절점들은 두 개 이상의 이중결합 요소에 포함 될 수 있다.

5.bicon 함수를 완전하게 구현하는 데 필요한 스택연산을 작성하라. 스택에 대하여 동적으로 연결된 표현 사용하라.

```
/*for biconnected, stack의 정의문*/
typedef struct stack *stack_pointer;
typedef struct stack {
    int vertex1;
    int vertex2;
    stack_pointer link;
}stack;
stack_pointer top;

/*함수 선언문*/
void deletes(stack_pointer *top, int *v1, int *v2);
void adds(stack_pointer *top, int v1, int v2);

void adds(stack_pointer *top, int v1, int v2) {
    stack_pointer temp = (stack_pointer)malloc(sizeof(stack));
    if(IS_FULL(temp)) {
        fprintf(stderr, "The memory is full\n");
        exit(1);
    }
    temp->vertex1 = v1;
    temp->vertex2 = v2;
    temp->link = *top;
    *top = temp; /*새로 추가된 node를 top*/
}

void deletes(stack_pointer *top, int *v1, int *v2) {
    stack_pointer temp = *top;
    if(IS_EMPTY(temp)) {
        fprintf(stderr, "The memory is empty\n");
        exit(1);
    }
    *v1 = temp->vertex1;
    *v2 = temp->vertex2;
    *top = temp->link;      /*top의 node를 제거하고 새로운 top 설정*/
    free(temp);
}
```

7. 이분할 그래프(bipartite graph)  $G=(V, E)$ 는 정점들을 두 개의 서로 다른 집합  $V_1$ 과  $V_2=V-V_1$ 로 분할할 수 있는 무방향 그래프로 다음과 같은 특성을 갖는다.

- $V_1$ 에 속해 있는 어떤 두 정점도  $G$ 에서 인접하지 않음
- $V_2$ 에 속해 있는 어떤 두 정점도  $G$ 에서 인접하지 않음

그래프의 이분할 여부를 결정하는 함수를 작성하라. 작성된 함수는 이분할인 그래프에 대해 위에 기술한 두 특성을 만족하는 서로 다른 정점들의 집합  $V_1$ 과  $V_2$ 를 구해야 한다.

```

/*          bipartite.c          :
/*          graph의 이분할 여부 결정,          */
/*          다른 정점들의 집합 결정          */

#include <stdio.h>
#include <stdlib.h>          /*for exit()*/
#include <string.h>          /*for memset()*/

#define FALSE 0
#define TRUE 1
#define MAX_VERTICES 8      /*maximum size of vertex*/

/*struct prototype*/
typedef struct node *node_pointer;
struct node{
    int vertex;
    node_pointer link;
};
node_pointer graph[MAX_VERTICES];

int matrix[][MAX_VERTICES] = {
    0, 1, 1, 0, 0, 0, 0, 0,
    1, 0, 0, 1, 0, 0, 0, 0,
    1, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 1, 0, 0,
    0, 0, 0, 0, 1, 0, 1, 0,
    0, 0, 0, 0, 0, 1, 0, 1,
    0, 0, 0, 0, 0, 0, 1, 0,
};

/*0으로 초기화, 집합 A는 1, 집합 B는 2*/
int partition[MAX_VERTICES];
/*function prototype*/
void make_lists(node_pointer *headnode);
void bipartite(int next, int prev);
/*모든 정점을 다 방문했는지 확인 하기 위한 전역변수*/
int first;

int main() {
    int i;

    make_lists(graph);

    /*initialize array*/
    memset(&partition, 0, sizeof(partition));

```

```

first = 0;          /*start vertex*/

bipartite(first, -1);

/*print set*/
printf("Set A[] = {");
for(i = 0; i<MAX_VERTICES; i++) {
    if(partition[i] == 1)
        printf("%2d", i);
}
printf(" }Wn");
printf("Set B[] = {");
for(i = 0; i<MAX_VERTICES; i++) {
    if(partition[i] == 2)
        printf("%2d", i);
}
printf(" }Wn");
}

/*V1, V2 각각에 속해 있는 어떤 두 정점도 G에서 인접하지 않음*/
void bipartite(int u, int prev) {

    node_pointer ptr;
    int w;          /*next vertex*/

    /*한 정점에서 시작해서 연결된 정점들을 recursion으로 계속 따라감*/
    for(ptr = graph[u]; ptr; ptr = ptr->link) {

        if(prev == -1)    //처음 방문하는 곳은 A집합으로 간주
            partition[u] = 1;

        w = ptr->vertex;
        if(w == prev)    //바로전에 방문한 정점은 제외
            continue;

        if(partition[w]) {                                /*이미 방문한 정점인 경우에*/
            if(partition[u] == partition[w]) {
                /*그 정점과의 edge가 있고 같은 집합이면 cycle*/
                fprintf(stderr, "--> This graph isn't bipartite graph!Wn");
                exit(1);
            }
        }
        else {
            /*prev vertex와는 다른 집합 생성*/
            partition[w] = partition[u] > 1 ? 1 : 2;
            bipartite(w, u);
        }
    }

    if(u == first){
        /*모든 정점을 방문했는지 확인*/
        for(w = 0; w<MAX_VERTICES; w++)
            if(!partition[w])    /*아직 방문하지 않은 정점이 있다면*/
                bipartite(w, -1); /*그 정점을 시작점으로 해서 다시 recursion*/
    }
}

```

```

/*make list from adjacency matrix*/
void make_lists(node_pointer *headnode) {
    node_pointer element, temp;
    node_pointer *ptr;
    int i, j;

    for(i = 0; i<MAX_VERTICES; i++)
        for(j = 0; j<MAX_VERTICES; j++) {
            if(matrix[i][j]) {
                element = (node_pointer) malloc(sizeof(struct node));
                element->vertex = j;
                element->link = NULL;

                ptr = &headnode[i];
                /*node가 없으면 자신이 head*/
                if(!(*ptr))
                    *ptr = element;
                else {
                    /*node가 있다면 마지막 노드를 찾아서 리스트 생성*/
                    for(temp = *ptr; temp->link; temp = temp->link);
                    temp->link = element;
                }
            }
        } /*end of if*/

    } /*end of for*/
}

```

10. 깊이 우선 탐색과 너비 우선 탐색을 네 개의 정점을 갖는 완전 그래프에 적용하라. 정점들을 방문된 순서대로 나열하라.

- 시작 정점을 0이라 할때 두 알고리즘 모두 0->1->2->3의 순서대로 정점을 방문한다.

11. dfs가 connected에서 사용되었을 때, 새로 방문된 모든 정점들의 리스트를 출력하도록 하라.

```

Void dfs(int v) {
    node_pointer w;
    visited[w] = TRUE;
    printf("%5d", v);
    for (w=graph[v]; w; w=w->link)
        if(!visited[w->vertex]) {
            printf("%2d", w->vertex);
            dfs(w->vertex);
        }
}

```

14. 연결 그래프 G의 간선들 중에서 그 삭제가 그래프를 연결되지 않도록 하는 간선(u, v)를 브리지(bridge)라 한다. 그래프에서 브리지를 찾아내는 함수를 작성하라. 단, 이 함수의 시간 복잡도는  $O(n+e)$ 가 되도록 해야 한다.

```
/*determine bridge*/
void bridge(int u, int v){
    node_pointer ptr;
    int w;

    dfn[u] = low[u] = num++;
    for(ptr = graph[u]; ptr; ptr = ptr->link) {
        w = ptr->vertex;
        if(dfn[w] < 0) {
            bridge(w, u);
            low[u] = low[u] < low[w] ? low[u] : low[w];
            if(low[w] > dfn[u])
                printf("<%d, %d> ", u, w);
        }
        else if(w != v)
            low[u] = low[u] < dfn[w] ? low[u] : dfn[w];
    }
}
```

>>동작 함수는 6.7 추가연습문제 5번 프로그램 참고

## 6.3절

2. Prim 알고리즘을 최소 비용 신장 트리를 발견하는 C 함수로 작성하라. 그래프의 정점의 수를  $n$ 이라 할 때, 이 함수의 복잡도는  $O(n^2)$ 이어야 한다.

```
/*      prim.c :
/*      find a minimum cost spanning tree */
/*
*/

#include <stdio.h>
#include <string.h>      /*for memcpy()*/

#define TRUE 1
#define FALSE 0
#define VERTICES 6      /*maximum size of vertex*/

/*struct prototype*/
typedef struct{
    int from; /*vetex 1*/
    int to;    /*vetex 2*/
    int cost; /*cost of edge*/
}edge_weight;

int cost[VERTICES][VERTICES] = {
    9999, 5, 4, 9999, 9999, 9999,
    5, 9999, 2, 7, 9999, 9999,
    4, 2, 9999, 6, 11, 9999,
    9999, 7, 6, 9999, 3, 8,
    9999, 9999, 11, 3, 9999, 8,
    9999, 9999, 9999, 8, 8, 9999,
};

/*최소비용 스패닝 트리*/
edge_weight minspantree[VERTICES-1];
/*prim function prototype*/
void prim(int alledge[][VERTICES], int start);

int main() {
    int i, totalcost = 0;
    /*start with vertex 0 and no edges*/
    prim(cost, 0);

    printf("Vertex 1WtVertex 2WtCostWn");
    for(i = 0; i<VERTICES-1; i++) {
        totalcost +=minspantree[i].cost;
        printf("%5dWtWt%4dWtWt%3dWn", minspantree[i].from, minspantree[i].to,
            minspantree[i].cost);
    }
    printf("Wnprim algorithm's minimum cost = %dWn", totalcost);
    return 0;
}

void prim(int alledge[][VERTICES], int start)
{
    int i;
```



```

int count = 0;                                /*T의 간선의 수*/
edge_weight mincost;

int nearest[VERTICES];                        /*각 정점들의 최소 비용 정점 집합*/
int distance[VERTICES];                       /*두 정점사이의 최소 비용*/

for(i = 0; i < VERTICES; i++) {              /*정점 start로 가는 모든 비용을 담는다*/
    nearest[i] = start;
    distance[i] = alledge[i][0];
}
distance[start] = FALSE;

while(count < VERTICES - 1)
{
    /*다른 cost보다 상당히 큰값으로 설정*/
    mincost.cost = 9999;
    for(i = 0; i < VERTICES; i++) {
        /*트리에 있는 정점외에 cost가 제일 작은 새로운 이웃정점 선택*/
        if(distance[i] && mincost.cost > distance[i]) {
            mincost.cost = distance[i];
            mincost.from = i;
            mincost.to = nearest[i];
        }
    }
    distance[mincost.from] = FALSE;            /*선택된 정점은 false*/

    for(i = 0; i < VERTICES; i++) {
        /*다른 이웃 정점들에서 새로 선택된 정점으로 가는 비용이 */
        /*기존 비용보다 싸면 새로 선택*/
        if(distance[i] > alledge[i][mincost.from]) {
            distance[i] = alledge[i][mincost.from];
            nearest[i] = mincost.from;
        }
    }

    if(mincost.cost == 9999)
        break;
    memcpy(&minspantree[count++], &mincost, sizeof(edge_weight));
}

if(count < VERTICES - 1)
    printf("No Spanning treeWn");
}

```

## 5. Sollin 알고리즘을 이용해 최소 비용 신장 트리를 발견하는 C 함수를 작성하라.

```

/*          sollin.c :
/*          find a minimum cost spanning tree */
/*          */

#include <stdio.h>
#include <string.h>          /*for memcpy(), memset()*/

#define TRUE 1
#define FALSE 0
#define VERTICES 6 /*maximum size of vertex*/

/*struct prototype*/
typedef struct{
    int from;
    int to;
    int cost;
}edge_weight;

int cost[VERTICES][VERTICES] = {
    9999,  5,    4,    9999,  9999,  9999,
    5,    9999,  2,    7,    9999,  9999,
    4,    2,    9999,  6,    11,   9999,
    9999,  7,    6,    9999,  3,    8,
    9999,  9999,  11,   3,    9999,  8,
    9999,  9999,  9999,  8,    8,    9999,
};

/*function prototype*/
void sollin(int edges[][VERTICES], int n);

void union2(int *parent, int i, int j);
int find(int *parent, int i);

edge_weight Minspantree[VERTICES - 1]; /*최소 비용 신장 트리*/
int main() {
    int i, totalcost = 0; /*spanning tree's cost*/
    sollin(cost, VERTICES);

    printf("Vertex 1WtVertex 2WtCostWn");
    for(i = 0; i<VERTICES-1; i++) {
        totalcost +=Minspantree[i].cost;
        printf("%5dWtWt%4dWtWt%3dWn", Minspantree[i].from, Minspantree[i].to,
            Minspantree[i].cost);
    }
    printf("Wnsollin algorithm's minimum cost = %dWn", totalcost);
    return 0;
}

/*각 단계에서 T에 포함될 간선을 여러개 선택*/
/*첫단계에서 n개의 모든 그래프 정점들을 포함하면서 포리스트 생성*/
/*단계를 거치면서 중복된 간선을 제거하면서 포리스트 생성*/
void sollin(int edges[][VERTICES], int n)
{
    edge_weight edgelist[VERTICES]; /*선정된 간선의 집합*/
    /*mincost : 최소 비용, select : 선택된 struct*/

```

```

edge_weight mincost, select;

int edgecount = 0;                                /*선택된 간선의 수*/
int i, j, k, l = 0;
int root, fromroot, toroot;                       /*각 forest의 root*/

int group[VERTICES];                              /*각 노드 그룹*/
int parent[VERTICES];                             /*루트 노드*/
/*initialize array*/
memset(group, -1, sizeof(group));
memset(parent, -1, sizeof(parent));

while((edgecount < n - 1)){
    /*각 그룹에 대해서 최소비용 선택*/
    for(i = 0; i<VERTICES; i++) {
        if(group[i] < 0) {
            root = i;    /*루트를 찾음*/
            mincost.cost = 9999;

            for(j = 0; j<VERTICES; j++) {
                /*루트의 그룹멤버들 중에서 최소비용을 찾음*/
                if((group[j] == root) || (j == root)) {
                    for(k = 0; k < VERTICES; k++)
                        if((mincost.cost > edges[j][k]) && edges[j][k]) {
                            mincost.from = j;
                            mincost.to = k;
                            mincost.cost = edges[j][k];
                        }
                }/*end of if*/
            }/*end of for*/
            //중복 간선을 제거해서 list에 추가
            edges[mincost.from][mincost.to] = FALSE;
            if(edges[mincost.to][mincost.from])
                memcpy(&edgelist[l++], &mincost, sizeof(edge_weight));

        }/*end of if*/
    }/*end of for*/

    while(l > 0) {                                /*list가 공백이 될 때까지*/
        memcpy(&select, &edgelist[--l], sizeof(edge_weight));
        /*두 vertex의 루트가 다른지 확인*/
        if((fromroot = find(parent, select.from)) != (toroot = find(parent, select.to))) {
            memcpy(&group, &parent, sizeof(parent));
            union2(parent, fromroot, toroot);      /*두 집합을 합집합*/
            memcpy(&Minspantree[edgecount++], &select, sizeof(edge_weight));
            edges[select.to][select.from] = FALSE;
        }
    }/*end of while*/
}/*end of while*/
}

void union2(int *parent, int i, int j)
{
    /*가중법칙을 이용하여 루트가 i와 j(i != j)인 집합을 합집합*/
    /*parent[i]=-count[i]이며 parent[j]=-count[j]*/

```

```

    int temp = parent[i] + parent[j];
    if (parent[i] > parent[j])
    {
        parent[i] = j; /*j를 새 루트로 만듦*/
        parent[j] = temp;
    }
    else
    {
        parent[j] = i; /*i를 새 루트로 만듦*/
        parent[i] = temp;
    }
}

int find(int *parent, int i)
{
    /*원소 i를 포함하는 루트를 찾음*/
    /*붕괴 법칙을 사용하여 i로부터 루트로 가는 모든 노드를 붕괴시킴*/
    int root, trail, lead;
    for(root = i; parent[root] >= 0; root = parent[root]);

    for(trail = i; trail != root; trail = lead)
    {
        lead = parent[trail];
        parent[trail] = root;
    }
    return root;
}

```

6. Kruskal 알고리즘을 이용해 최소 비용 신장 트리를 구하는 C 함수를 작성하라.  
5장의 union & find 함수, sort 함수나 최소 heap 함수를 이용하라.

```
/*          kruskal.c :                               */
/*          find a minimum cost spanning tree         */
/*          최소 heap 함수 사용                       */
/*                                                     */

#include <stdio.h>
#include <string.h>          /*for memset(), memcpy()*/

#define VERTICES 6          /*maximum size of vertex*/
#define NUMEDGE 9           /*maximum size of edge*/

/*struct prototype*/
typedef struct{
    int from;
    int to;
    int cost;
}edge_weight;
/*heap sort array*/
edge_weight edgelist[NUMEDGE + 1];

int cost[VERTICES][VERTICES] = {
    9999,  5,    4,    9999,  9999,  9999,
    5,    9999,  2,    7,    9999,  9999,
    4,    2,    9999,  6,    11,   9999,
    9999,  7,    6,    9999,  3,    8,
    9999,  9999, 11,    3,    9999,  8,
    9999,  9999, 9999,  8,    8,    9999,
};

/*function prototype*/
/*heapsort function*/
void heapsort_MinHeap(edge_weight list[], int n);
void adjust_MinHeap(edge_weight list[], int root, int n);
edge_weight delete_MinHeap(edge_weight *heap, int *n);
/*kruskal function*/
void kruskal(edge_weight E[]);

void union2(int *parent, int i, int j);
int find(int *parent, int i);
/*minimum cost spanning tree*/
edge_weight minspantree[VERTICES - 1];

int main() {

    int i, j, k = 1;
    int totalcost = 0;

    /*존재하는 간선에 대해 리스트에 추가*/
    for(i = 0; i<VERTICES - 1; i++)
    {
        for(j = i+1; j<VERTICES; j++)
        {
            if(cost[i][j] != 9999) {
                edgelist[k].cost = cost[i][j];
```

```

        edgelist[k].from = i;
        edgelist[k].to = j;
        k++;
    }
}
/*heap sorting*/
heapsort_MinHeap(edgelist, NUMEDGE-1);
/*call kruskal*/
kruskal(edgelist);

printf("Vertex 1WtVertex 2WtCostWn");
for(i = 0; i<VERTICES-1; i++) {
    totalcost +=minspantree[i].cost;
    printf("%5dWtWt%4dWtWt%3dWn",      minspantree[i].from,      minspantree[i].to,
        minspantree[i].cost);
}
printf("Wnkruskal algorithm's minimum cost : %dWn", totalcost);
return 0;
}
void kruskal(edge_weight Edgelist[])
{
    int fromroot, toroot;      /*root of each vertex*/
    int e = NUMEDGE;          /*number of edge*/
    int v = 0;                /*number of vertices*/
    int T[VERTICES];
    int l = 0;

    edge_weight select;
    memset(T, -1, sizeof(T));

    /*T가 n-1개의 간선 포함 && E가 비어있지 않음*/
    while(e > 0 && v < VERTICES - 1)
    {
        /*E에서 최저비용간선 선택, 삭제*/
        select = delete_MinHeap(Edgelist, &e);
        /*각 정점의 루트가 다르면 두 정점을 합집합*/
        if((fromroot = find(T, select.from)) != (toroot = find(T, select.to))) {
            union2(T, fromroot, toroot);      /*각 원소의 루트값을 넘겨줌*/

            memcpy(&minspantree[l++], &select, sizeof(edge_weight));
            ++v;
        }
    }

    if(v < VERTICES-1)
        printf("No spanning treeWn");
}

void union2(int *parent, int i, int j)
{
    int temp = parent[i] + parent[j];
    if (parent[i] > parent[j])
    {
        parent[i] = j;
    }
}

```

```

        parent[j] = temp;
    }
    else
    {
        parent[j] = i;
        parent[i] = temp;
    }
}

int find(int *parent, int i)
{
    int root, trail, lead;
    for(root = i; parent[root] >=0; root = parent[root]);

    for(trail = i; trail != root; trail = lead)
    {
        lead = parent[trail];
        parent[trail] = root;
    }
    return root;
}

/*heap sorting*/
void heapsort_MinHeap(edge_weight list[], int n)
{
    int i;
    edge_weight temp;

    for(i = n/2; i > 0; i--)
        adjust_MinHeap(list, i, n);
    for(i = n-1; i > 0; i--) {
        temp = list[1];
        list[1] = list[i+1];
        list[i+1] = temp;
        adjust_MinHeap(list, 1, i);
    }
}

void adjust_MinHeap(edge_weight list[], int root, int n)
{
    int rootkey, child;
    edge_weight temp;
    temp = list[root];
    rootkey = list[root].cost;
    child = 2*root;

    while ( child <= n){
        if ((child < n) && (list[child].cost < list[child+1].cost))
            child++;

        if (rootkey > list[child].cost)
            break;
    }

```

```

        else {
            list[child/2] = list[child];
            child *=2;
        }
    }
    list[child/2] = temp;
}

edge_weight delete_MinHeap(edge_weight *heap, int *n)
{
    int parent, child;
    edge_weight item, temp;

    item = heap[1];
    temp = heap[(*n)--];
    parent = 1;
    child = 2;

    while(child <= *n){

        if((child < *n) && (heap[child].cost > heap[child+1].cost))
            child++;
        if(temp.cost <= heap[child].cost)
            break;
        heap[parent] = heap[child];
        parent = child;
        child *= 2;
    }
    heap[parent] = temp;
    return item;
}

```



## 6.4절

1. 그래프를 입력하는 C 함수를 작성하라. 그래프는 `cost[i][j]`인접 행렬로 표현하고, 존재하지 않는 간선의 값은 `shortestpath`와 `allcosts`에서 모두 이용할 수 있도록 초기화하라.

```
/*          insert graph.c :                               */
/*          그래프를 입력하는 C함수                         */
/*          adjacency matrix로 표현하고 존재하지 않는 간선은 초기화*/

#include <stdio.h>
#include <stdlib.h>      /*for malloc(), free()*/

#define FALSE 0
#define TRUE 1

int inputcost(void);
/*이차원 배열 포인터*/
int **cost;

int main(void)
{
    int i, j, n;
    int *distance; /*distance[] pointer*/
    short int *found; /*found[] pointer*/

    printf(" ***** Insert the graph *****\n");
    n = inputcost();      /*입력된 그래프의 정점의 수*/
    /*user로부터 vertex수를 입력받아 동적으로 배열 할당*/
    distance = (int*)malloc(sizeof(int)*n);
    found = (short int *)malloc(sizeof(short int) * n);

    printf("\n + + + Adjacency Matrix + + + \n");
    for(i = 0; i<n; i++) {
        for(j = 0; j<n; j++)
            printf("%7d", cost[i][j]);
        printf("\n");
    }
    /*열에 해당하는 메모리 공간 해제*/
    for(i = 0; i < n; i++)
        free(cost[i]);
    /*행에 해당하는 메모리 공간 해제*/
    free(cost);
    free(distance);
    free(found);
    return 0;
}

/*user로부터 vertex수와 간선들을 입력받음*/
int inputcost(void)
{
    /*vertices : 정점의 수, vertex1, vertex2 : 각 정점, weight : 비용*/
    int vertices, vertex1, vertex2, weight;
    int i, j, count = 0; /*count : array 크기를 넘는지 확인*/

    printf(" Input the number of Vertex : ");
    scanf("%d", &vertices);
```

```

/*cost인접행렬의 행의 수만큼 동적 할당*/
cost = (int**)malloc(sizeof(int) * vertices);
/*cost인접행렬의 열의 수만큼 동적 할당*/
for(i = 0; i<vertices; i++)
    cost[i] = (int*)malloc(sizeof(int) * vertices);

/*initialize cost[][]*/
for(i = 0; i<vertices; i++)
    for(j = 0; j<vertices; j++)
        cost[i][j] = -1;

printf(" Input the weight each vertex(exit = -1) : Usage(vertex1, vertex2, cost)Wn");
/*minus integer가 입력될때까지 간선을 입력받음*/
while(count < (vertices * vertices)) {
    printf("%d : ", count++);
    scanf("%d", &vertex1);
    if(vertex1 == -1)
        break;
    scanf("%d%d", &vertex2, &weight);

    cost[vertex1][vertex2] = weight;
}

/*존재하지 않는 간선의 값 초기화*/
for(i = 0; i<vertices; i++)
    for(j = 0; j<vertices; j++) {
        if(cost[i][j] == -1) {
            if(i == j)
                cost[i][j] = 0;
            else
                cost[i][j] = 1000;
        }
    }
return vertices;
}

```

2. `shortestpath`가 각 최단 경로의 거리는 물론 경로도 출력할 수 있도록 재작성하라.

```
/*      shortestpath.c :
/*      최단 경로의 거리와 경로를 출력하는 함수 */
/*
*/

#include <stdio.h>
#include <limits.h>      /*for INT_MAX*/
#include <stdlib.h>      /*for malloc(), free()*/

#define FALSE  0
#define TRUE   1

#define MAX_VERTICES 6    /*maximum size of vertex*/

/*struct prototype*/
typedef struct node *node_pointer; /*struct pointer*/
struct node {
    int vertex;
    node_pointer link;
}node;
/*각 경로를 저장하는 구조체 포인터 배열*/
node_pointer path[MAX_VERTICES];

int cost[][MAX_VERTICES] = {
    0,    50,    10,    1000,    45,    1000,
    1000,  0,    15,    1000,    10,    1000,
    20,    1000,  0,    15,    1000,    1000,
    1000,  20,    1000,  0,    35,    1000,
    1000,  1000,  1000,  30,    0,    1000,
    1000,  1000,  1000,  3,    1000,  0,
};

int distance[MAX_VERTICES];
short int found[MAX_VERTICES];

int n = MAX_VERTICES;

int choose(int distance[], int n, short found[]);
void shortestpath(int v, int cost[][MAX_VERTICES], int distance[],
                  int n, short int found[]);
void free_memory(node_pointer *ptr);

void main() {
    int i,j;
    node_pointer temp;
    printf("++++ Shortest Path +++++ WnWn");
    printf("WtpathWtWtWtcostWn");
    /*모든 정점에 대해 경로와 최소비용 선택*/
    for(i = 0; i<n; i++) {
        shortestpath(i, cost, distance, n, found);
        for(j = 0; j<n; j++) {
            printf("%2d->%2d : ", i, j);
            for(temp = path[j]; temp; temp = temp->link)
                printf("V%d ", temp->vertex);
            if(!path[j])
                printf("\n");
        }
    }
}
```

```

        printf("No path!WnWn");
    else
        printf("V%dWnWtWtWtWt%dWn",j, distance[j]);
    }
    printf("Wn");
    for(j = 0; j<n; j++)
        free_memory(&path[j]);
}

/*메모리 해제 함수*/
void free_memory(node_pointer *ptr){
    node_pointer trail;
    while(*ptr) {
        trail = *ptr;
        *ptr = (*ptr)->link;
        free(trail);
    }
}

int choose(int distance[], int n, short int found[])
{
    /*아직 조사안된 정점 중에서 distance값이 최소인 것을 찾음*/
    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 shortestpath(int v, int cost[][MAX_VERTICES], int distance[],
                 int n, short int found[])
{
    /*distance[i] : 정점 v에서 i로의 최단 경로, 최단 경로가 발견되면 found[i] = 1*/
    int i, u, w;
    node_pointer ptr, temp, next;

    for(i = 0; i < n; i++) {
        found[i] = FALSE;
        distance[i] = cost[v][i];
    }
    found[v] = TRUE;
    distance[v] = 0;

    for(i = 0; i < n-2; i++) {
        u = choose(distance, n ,found);
        found[u] = TRUE;

        if(!path[u] && !(distance[u] == 1000)) { /*발견되지 않은 정점*/

```

```

        temp = (node_pointer)malloc(sizeof(node));
        temp->vertex = v;
        temp->link = NULL;
        path[u] = temp;
    }

    for(w = 0; w < n; w++)
        if(!found[w]) {
            if(distance[u] + cost[u][w] < distance[w]) {
                distance[w] = distance[u] + cost[u][w];
                /*기존의 최단 경로에서 다시 최단경로로 업데이트 되었을때 */
                if(path[w]){
                    free_memory(&path[w]);          /*기존의 경로 삭제*/
                }

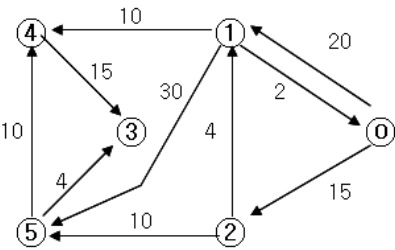
                /*새로 발견된 경로들을 정점에 입력*/
                for(ptr = path[u]; ptr; ptr = ptr->link) {
                    temp = (node_pointer)malloc(sizeof(node));
                    temp->vertex = ptr->vertex;
                    temp->link = NULL;
                    if(!path[w])
                        path[w] = next = temp;
                    else {
                        next->link = temp;
                        next = next->link;
                    }
                }
                /*마지막 정점 입력*/
                temp = (node_pointer)malloc(sizeof(struct node));
                temp->vertex = u;
                temp->link = NULL;
                next->link = temp;

            } /*end of if*/
        } /*end of if*/

    } /*end of for*/
} /*end of function*/

```

4. shortestpath 함수를 이용하여 아래의 다이그래프에서 정점 0에서 다른 모든 정점으로의 최단 경로 길이를 구하라. 단, 길이의 크기 순서로 경로들을 생성하라.



	0	1	2	3	4	5
정점 0	0	19	15	29	29	25

5. 다음 두 가정하에 shortestpath를 재작성하라.

- (a) G는 인접 리스트에 의해 표현된다. 각 리스트의 노드는 vertex, cost, link 필드를 갖는다. cost필드는 goekd 간선의 길이를 나타내고 n은 그래프 G의 정점수이다.
- (b) S(최단 경로가 이미 발견된 정점의 집합) 대신  $T=V(G)-S$ 를 사용하라. T를 연결리스트로 표현하라.

```
/*      shortestpath_list.c : list version      */
/*      G는 리스트에 의해 표현                  */
/*      T=V(G)-S 사용                          */

#include <stdio.h>
#include <limits.h>      /*for INT_MAX*/
#include <stdlib.h>      /*for malloc(), free()*/

#define FALSE  0
#define TRUE   1

#define MAX_VERTICES 6    /*maximum size of vertex*/
/*node 구조체*/
typedef struct node *node_pointer; /*struct pointer*/
struct node {
    int vertex;
    int cost;
    node_pointer link;
};
node_pointer graph[MAX_VERTICES]; /*struct array*/

/*최단 경로 발견 여부 확인을 위한 구조체*/
typedef struct found *found_pointer;
struct found {
    int vertex;
    found_pointer link;
};

int cost[][MAX_VERTICES] = {
    0,      50,      10,      1000,      45,      1000,
    1000,    0,      15,      1000,      10,      1000,
    20,      1000,    0,      15,      1000,      1000,
    1000,    20,      1000,    0,      35,      1000,
    1000,    1000,    1000,    30,      0,      1000,
    1000,    1000,    1000,    3,      1000,    0,
};

int distance[MAX_VERTICES];
found_pointer p_fnd = NULL; /*조사되지 않은 정점을 가지는 구조체 포인터*/

int n = MAX_VERTICES;

void make_lists(node_pointer *headnode);

int choose(int distance[], int n, found_pointer *p_fnd);
void shortestpath(int v, node_pointer *head_node, int distance[],
```

```

        int n, found_pointer *p_fnd);

void free_memory(node_pointer *ptr);

void main() {

    int i, j;

    make_lists(graph);

    for(i = 0; i<n; i++) {
        shortestpath(i, graph, distance, n, &p_fnd);

        for(j = 0; j<n; j++)
            printf("%10d", distance[j]);
        printf("Wn");
    }
}

void make_lists(node_pointer *headnode) {
    node_pointer element, temp;
    node_pointer *ptr;          /*head node*/
    int i, j;

    /*adjacency matrix를 list로 변환*/
    for(i = 0; i<MAX_VERTICES; i++)
        for(j = 0; j<MAX_VERTICES; j++) {
            element = (node_pointer) malloc(sizeof(struct node));
            element->vertex = j;
            element->cost = cost[i][j];
            element->link = NULL;

            ptr = &headnode[i];
            /*head node가 null이면 head node에 추가*/
            if(!(*ptr))
                *ptr = element;
            /*아니면 마지막 node를 찾아 list추가*/
            else {
                for(temp = *ptr; temp->link; temp = temp->link);
                temp->link = element;
            }
        }
}

/*memory 해제 함수*/
void free_memory(node_pointer *ptr){
    node_pointer trail;
    while(*ptr) {
        trail = *ptr;
        *ptr = (*ptr)->link;
        free(trail);
    }
}

```



```

int choose(int distance[], int n, found_pointer *p_fnd)
{
    int i, min, minpos, exist;
    found_pointer ptr;

    min = INT_MAX;
    minpos = -1;
    exist = FALSE;

    for(i = 0; i < n; i++) {
        /*리스트를 따라가면서 최소비용 선택*/
        for(ptr = *p_fnd; ptr; ptr = ptr->link){
            if(distance[ptr->vertex] < min) {
                min = distance[ptr->vertex];
                minpos = ptr->vertex;
            }
        }
    }
    return minpos;
}

void shortestpath(int v, node_pointer *head_node, int distance[],
                  int n, found_pointer *p_fnd)
{
    int i, u, w;
    node_pointer ptr;
    found_pointer lead, trail;

    /*p_fnd리스트에 v를 제외한 모든 정점 저장*/
    for(i = 0; i < n; i++) {
        if(i == v)
            continue;

        lead = (found_pointer)malloc(sizeof(struct found));
        lead->vertex = i;
        lead->link = NULL;
        /*head node가 null이면 head node*/
        if(!(*p_fnd))
            *p_fnd = trail = lead;
        /*아니면 trail다음에 추가*/
        else{
            trail->link = lead;
            trail = lead;
        }
    }

    /*리스트 정점의 cost를 distance배열에 넣음*/
    for(ptr = head_node[v], i = 0; ptr; ptr = ptr->link, i++)
        distance[i] = ptr->cost;
}

```

```

distance[v] = 0;

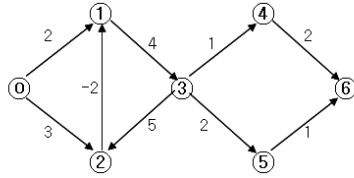
for(i = 0; i < n-2; i++) {
    u = choose(distance, n ,p_fnd);
    /*T = V(G) - S*/
    /*찾은 vertex가 head일때 head를 변경하고 head삭제*/
    if((*p_fnd)->vertex == u) {
        lead = *p_fnd;
        *p_fnd = lead->link;
        free(lead);
    }
    /*head가 아니라면 found를 탐색해서 선택된 정점 삭제*/
    else {
        for(trail = *p_fnd; trail; trail = trail->link) {
            if(trail->vertex == u) {
                lead->link = trail->link;
                free(trail);
                break;
            }
            lead = trail;
        }
    }
    /*아직 조사안된 정점들중에서 최소비용 선택*/
    for(lead = *p_fnd; lead; lead = lead->link) {
        w = lead->vertex;

        for(ptr = head_node[u]; ptr; ptr = ptr->link)
            if(ptr->vertex == w)
                break;

        if(distance[u] + ptr->cost < distance[w])
            distance[w] = distance[u] + ptr->cost;
    }
}
free(*p_fnd);
*p_fnd = NULL;
}

```

6. 아래의 다이 그래프에 shortestpath가 제대로 실행되지 않는 이유를 설명하라. 정점  $v_0$ 와  $v_6$ 사이의 최단 경로는 무엇인가?



- 음의 가중치가 부여되어 있다. 정점 0에서 1로 가는 cost는 2이다.  
 하지만 실제로 음의 가중치가 부여된 경로(0->2->1)로 가면 cost가 1이 된다.  
 즉 shortestpath에서 계산된 값보다 더 작은 값이 나온다.  
 이러한 음의 간선이 부여된 그래프는 negativeWeightpath 알고리즘을 이용하면 된다.
- 알고리즘 상 :  $v_0 \rightarrow v_1 \rightarrow v_3 \rightarrow v_4(v_5) \rightarrow v_6$  : cost=9  
 하지만 음의 가중치가 부여된 간선을 이용하면  
 $v_0 \rightarrow v_2 \rightarrow v_1 \rightarrow v_3 \rightarrow v_4(v_5) \rightarrow v_6$  : cost = 8의 비용이 든다.

8. allcosts 함수가 경로의 길이는 물론 경로 그 자체도 인쇄할 수 있도록 수정하라.

```
/*      allcosts.c :
/*      경로의 길이와 경로를 출력하는 프로그램 */
/*
*/

#include <stdio.h>
#include <limits.h>          /*for INT_MAX*/
#include <stdlib.h>          /*for malloc(), free()*/

#define FALSE  0
#define  TRUE  1

#define MAX_VERTICES 6      /*maximum size of vertex*/

/*node 구조체*/
/*각 정점의 최단 경로를 저장하는 구조체*/
typedef struct node *node_pointer;
struct node {
    int vertex;
    node_pointer link;
};
/*경로를 저장하기 위한 구조체 포인터 배열*/
node_pointer path[MAX_VERTICES][MAX_VERTICES];

int cost[][MAX_VERTICES] = {
    0,      50,      10,      1000,      45,      1000,
    1000,    0,      15,      1000,      10,      1000,
    20,      1000,    0,      15,      1000,    1000,
    1000,    20,      1000,    0,      35,      1000,
    1000,    1000,    1000,    30,      0,      1000,
    1000,    1000,    1000,    3,      1000,    0,
};

void allocosts(int cost[][MAX_VERTICES], int distance[][MAX_VERTICES], int n);

void free_memory(node_pointer *ptr);

int distance[MAX_VERTICES][MAX_VERTICES];
short int found[MAX_VERTICES];

int n = MAX_VERTICES;

void main() {
    int i, j;
    node_pointer temp;

    printf( "Wt+ + + allocosts + + + Wn");
    allocosts(cost, distance, n);

    printf("WtpathWtWtWtcostWn");
    for(i = 0; i<n; i++) {
```

[illegible]

```

        for(ptr = path[k][j]; ptr; ptr = ptr->link) {
            temp = (node_pointer)malloc(sizeof(struct node));
            temp->vertex = ptr->vertex;
            temp->link = NULL;
            next->link = temp;
            next = next->link;
        }

        distance[i][j] = distance[i][k] + distance[k][j];
    }
}

void free_memory(node_pointer *ptr){
    node_pointer trail;
    while(*ptr) {
        trail = *ptr;
        *ptr = (*ptr)->link;
        free(trail);
    }
}

```

10. 문제 4의 다이 그래프에 대한 행렬  $A^*$  와  $A^+$ 를 구하라.

- transitive closure	- reflexive transitive closure
$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$

- 11, 12. (a)  $n$ 개의 정점을 갖는 무방향 그래프의 반사 이행적 폐쇄를  $O(n^2)$ 시간에 구하는 C 함수를 작성하라. 그래프의 인접 행렬에서 시작하여 연결 요소들을 구하라.  
(b) 이행적 폐쇄의 경우에 대해서도 함수를 작성하라

```
/*      (reflexive)transitive closure.c :                               */
/*      undirected graph의 (반사)이행적 폐쇄를 구하는 프로그램      */
/*      인접행렬에서 시작하여 연결 요소들을 구한다.                */

#include <stdio.h>

#define MAX_VERTICES  8    /*maximum size of vertex*/

//그림 6.5그래프
int adj_matrix[][MAX_VERTICES] = {
    0, 1, 1, 0, 0, 0, 0, 0,
    1, 0, 0, 1, 0, 0, 0, 0,
    1, 0, 0, 1, 0, 0, 0, 0,
    0, 1, 1, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 1, 0, 0,
    0, 0, 0, 0, 1, 0, 1, 0,
    0, 0, 0, 0, 0, 1, 0, 1,
    0, 0, 0, 0, 0, 0, 1, 0,
};

short int visited[MAX_VERTICES];    /*방문 정점확인 배열*/
int transitive[MAX_VERTICES][MAX_VERTICES];

/*initialize function*/
void init();
/*connected component*/
void dfs(int v, int c);
void connected(void);

void re_trans_closure();
void trans_closure();
int n = MAX_VERTICES;

void main() {
    int i, j;
    init();

    trans_closure();
    printf("== Transitive closure ==\n");
    for(i = 0; i<n; i++) {
        for(j = 0; j<n; j++)
            printf("%3d", transitive[i][j]);
        printf("\n");
    }
    init();
    re_trans_closure();
    printf("\n\n== Reflexive Transitive closure ==\n");
    for(i = 0; i<n; i++) {
        for(j = 0; j<n; j++)
            printf("%3d", transitive[i][j]);
        printf("\n");
    }
}
```

```

    }
}
/*i에서 j로의 길이 > 0인 경로가 있으면 A+ [i][j]=1*/
void trans_closure() {
    int i, j;

    connected();

    for(i = 0; i<n; i++) {
        for(j = 0; j<n; j++) {
            /*같은 연결요소들은 서로 경로가 존재, selfloop는 제외*/
            if((visited[j] == visited[i]) && (i != j))
                transitive[i][j] = 1;
        }
    }
}

/*i에서 j로의 길이 >= 0인 경로가 있으면 A*[i][j]=1*/
void re_trans_closure() {
    int i, j;

    connected();

    for(i = 0; i<n; i++) {
        for(j = 0; j<n; j++) {
            /*같은 연결요소들은 서로 경로가 존재*/
            if(visited[j] == visited[i])
                transitive[i][j] = 1;
        }
    }
}

void dfs(int v, int c) {
    int i;

    /*같은 연결요소들은 c를 통해 서로 같은 집합 생성*/
    visited[v] = c;

    for(i = 0; i<MAX_VERTICES; i++) {
        if(!visited[i] && adj_matrix[v][i])
            dfs(i, c);
    }
}

void connected(void) {
    /*그래프의 연결 요소 결정*/
    int i;
    int connect = 1; /*연결 요소들을 구분짓기 위한 변수*/

    for (i = 0; i<n; i++)
        if(!visited[i]) {
            dfs(i, connect);
            connect++; /*서로다른 연결 요소 구분*/
        }
}

```



```
void init() {  
    int i, j;  
    /*초기화*/  
    for(i = 0; i<n; i++) {  
        visited[i] = 0;  
        for(j = 0; j<n; j++)  
            transitive[i][j] = 0;  
    }  
}
```

## 6.5절

2. (a) 그림 6.47의 AOE 네트워크에 대해 각 작업의 early 값과 late 값을 구하라.

단, 전진-후진방법을 사용하라.

(b) 프로젝트를 종료할 수 있는 가장 이른 시간은 언제인가?

(c) 어떤 작업들이 임계 작업인가?

(d) 그 자신의 완료 시간을 단축하면 프로젝트의 완료 시간이 단축되는 단일 작업이 존재하는가?

(a-1) earliest의 계산												(a-2) latest의 계산											
Earliest	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	스택	latest	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	스택
초기	0	0	0	0	0	0	0	0	0	0	0	초기	23	23	23	23	23	23	23	23	23	23	9
v0 출력	0	5	6	0	0	0	0	0	0	0	2 1	v9 출력	23	23	23	23	23	23	19	23	21	23	8 6
V2 출력	0	5	6	12	9	0	0	0	0	0	1	V8 출력	23	23	23	23	23	16	19	19	21	23	7 5 6
V1 출력	0	5	6	12	9	0	0	0	0	0	3	V7 출력	23	23	23	23	15	16	19	19	21	23	5 6
V3 출력	0	5	6	12	15	16	16	0	0	0	6 4	V5 출력	23	23	23	12	15	16	19	19	21	23	4 6
V6 출력	0	5	6	12	15	16	16	0	0	20	4	V4 출력	23	23	12	12	15	16	19	19	21	23	6
V4 출력	0	5	6	12	15	16	16	19	0	20	7 5	V6 출력	23	23	12	12	15	16	19	19	21	23	3
V7 출력	0	5	6	12	15	16	16	19	21	20	5	V3 출력	23	9	6	12	15	16	19	19	21	23	2 1
V5 출력	0	5	6	12	15	16	16	19	21	20	8	V2 출력	0	9	6	12	15	16	19	19	21	23	1
V8 출력	0	5	6	12	15	16	16	19	21	23	9	V1 출력	0	9	6	12	15	16	19	19	21	23	0
V9 출력												V0 출력											

(a-3) early, late 값과 임계성 여부				
작업	Early	Late	Late-Early	임계성 여부
a0	0	4	4	N
a1	0	0	0	Y
a2	5	9	4	N
a3	6	6	0	Y
a4	6	12	6	N
a5	12	12	0	Y
a6	12	12	0	Y
a7	12	15	3	N
a8	15	15	0	Y
a9	15	15	0	Y
a10	16	16	0	Y
a11	16	19	3	N
a12	19	19	0	Y
a13	21	21	0	Y

(b) 프로젝트를 종료 할수 있는 가장 이른 시간 : 23

(c) 임계 작업 : a1, a3, a5, a8, a9, a10, a12, a13

(d) 단일 작업 : a1, a3, a13

3.[프로그래밍 프로젝트] AOE프로그램을 입력하는 C 프로그램을 작성하라. 이 프로그램은 early(i)와 late(i), 각 작업의 임계성 여부를 계산해 출력해야 한다. 프로젝트가 실현 가능성이 없으면 이를 표시해야 한다. 프로젝트가 가능한 경우는 적절한 양식에 따라 임계 작업들을 인쇄해야 한다.

```

/*      aoenetwork.c :                                                    */
/*      프로그램을 입력하여 early(i)와 late(i), 각 작업의 임계성 여부 출력    */
/*      실현 가능성이 없는 프로젝트에 대해 이를 표시                        */
/*                                                                           */

#include <stdio.h>
#include <stdlib.h>                /*for malloc(), free(), exit()*/

#define IS_FULL(ptr) (!(ptr))
#define MAX_VERT 50               /*maximum size of vertex*/

/*struct prototype*/
typedef struct node *node_pointer;
typedef struct node {
    int vertex;
    int dur;
    node_pointer link;
} node;

/*topsort를 위한 struct prototype*/
typedef struct {
    int count;
    node_pointer link;
} hdnodes;

hdnodes graph[MAX_VERT]; /* Adjacent List */
hdnodes r_graph[MAX_VERT]; /* Reverse Adjacent List */
int eest[MAX_VERT]; /* earliest Array */
int lest[MAX_VERT]; /* latest Array */
int stack[MAX_VERT]; /* stack */
int top = -1;

void topsort(hdnodes [], int n, int flag); /* Top Sort */
int graph_insert(hdnodes [], int s_vertex, int e_vertex, int dur); /* Adjacent List create */
int graph_print(hdnodes [], int s_vertex, int flag); /* Graph print */
void rev_topsort(int n);
void crit_print(hdnodes [], int n); /* Critical Activity print */
void push(int value); /*stack operation*/
int pop();

void free_memory(int max_vertex, hdnodes *graph);
/*===== Main 함수 =====*/
int main()
{
    int start_vertex, end_vertex, duration;
    int max_vertex;

    printf("=====Wn");
    printf("==          AOE network          ==Wn");
    printf("=====Wn");

```

```

printf("First Start Vertex is Zero(0) Wn");
printf("입력 종료 : -1 WnWn");

while(1){

    printf("Insert Start Vertex, End_vertex, Duration : ");
    scanf("%d", &start_vertex);
    if(start_vertex == -1 )
        break;

    scanf("%d %d", &end_vertex, &duration);

    if(start_vertex >= end_vertex)
        printf("WnWrong Value! Inset again.Wn");

    else {

        if(max_vertex < end_vertex)
            max_vertex = end_vertex;

        graph_insert(graph, start_vertex , end_vertex, duration);

    }
} /* end while */

graph_print(graph , max_vertex, 0);      /* Graph 출력 */
topsort(graph, max_vertex+ 1, 0);        /* 인접 리스트 위상 정렬 */

rev_topsort(max_vertex);                 /* 역인접 리스트 생성 */

graph_print(r_graph , max_vertex,1);
topsort(r_graph, max_vertex+ 1, 1);      /* 역인접 리스트 위상 정렬 */

crit_print(graph, max_vertex);           /* Critical Activity 출력 */

free_memory(max_vertex, graph);
free_memory(max_vertex, r_graph);
return 0;
}
/*memory 해제 함수*/
void free_memory(int max_vertex, hdnodes *graph) {
    int i;
    node_pointer temp;

    for(i = 0; i<max_vertex; i++) {
        while(graph[i].link) {
            temp = graph[i].link;
            graph[i].link=graph[i].link->link;
            free(temp);
        }
    }
}
/*===== Graph Insert =====*/
int graph_insert(hdnodes graph[], int s_vertex, int e_vertex, int dur)
{

```

```

node_pointer element, adjlist;
element = (node_pointer)malloc(sizeof(node));

if(IS_FULL(element)) {
    fprintf(stderr, "The memory is Full ! \n");
    exit(1);
}

graph[e_vertex].count = graph[e_vertex].count + 1; /* Vertex Count */

if( graph[s_vertex].link != NULL){

    for(element=graph[s_vertex].link; element->link != NULL; element = element->link)

        if( element->vertex == e_vertex ){ /* 같은 정점이 있을 경우 error */
            /* 인접 리스트를 마지막으로 옮기 */
            printf("%d Same Vertex Exist!\n", element->vertex);
            exit(1);
        }

        adjlist = (node_pointer)malloc(sizeof(node)); /* Attach new vertex, duration */
        adjlist->vertex = e_vertex;
        adjlist->dur = dur;
        adjlist->link = NULL;
        element->link = adjlist;

    } else { /* 리스트가 첫번째일 경우 Insert */
        adjlist = (node_pointer)malloc(sizeof(node));
        adjlist->vertex = e_vertex;
        adjlist->dur = dur;
        adjlist->link = NULL;
        graph[s_vertex].link = adjlist;
    }
    return 0;
}

/*===== Reverse Topological Sorting =====*/
void rev_topsort(int n)
{
    int i;
    node_pointer ptr;
    ptr = (node_pointer)malloc(sizeof(node));

    for(i=0; i < n; i++){

        for(ptr=graph[i].link; ptr; ptr=ptr->link) {

            graph_insert(r_graph, ptr->vertex, i, ptr->dur);

        }

    }
}

/*===== Topological Sorting =====*/
void topsort(hdnodes graph[], int n, int flag)
{

```

```

int i,j,k,l;
node_pointer ptr;

if(flag == 0){ /* Array 초기화 */
    printf("Wn===== Earliest Calculation =====WnWn");
    printf("Earliest ");
    for(i=0; i<n; i++){
        eest[i] = 0;
        printf(" [%d]", i );
    }
} else {
    printf("Wn===== Latest Calculation =====WnWn");
    printf("Latest ");
    for(i=0; i<n;i++){
        lest[i] = eest[n-1];
        printf(" [%d]", i );
    }
}
printf(" [stack] Wn 초기 ");

for(i=0; i<n; i++) /*선행자를 갖지 않는 정점들의 스택생성*/
    if(!graph[i].count)
        push(i);

for(i=0;i<n;i++){
    if(top == -1){
        fprintf(stderr, "WnNetwork ha a cycle. Sort terminated. Wn");
        exit(1);
    } else {
        /* ===== print earliest , latest , stack print =====*/
        if(flag==0){
            for(l=0; l < n ; l++)
                printf(" %2d ", eest[l]);
        }else{
            for(l=0; l < n ; l++)
                printf(" %2d ", lest[l]);
        }
        for(l=top; l >= 0 ; l--)
            printf(" [%d]", stack[l]);
        printf("Wn");

        j=pop(); /*정점을 스택에서 꺼냄*/
        printf("V%d출력 : ", j);

        for(ptr=graph[j].link; ptr; ptr=ptr->link) {
            /* j의 후속자 정점의 수를 감소시킴 */
            k = ptr->vertex;
            graph[k].count--;

            if(!graph[k].count) /*add vertex k to the stack */
                push(k);

            if(flag==0){ /* earliest Calculation */
                if(eest[k] < eest[j]+ptr->dur)

```

```

                eest[k]= eest[j]+ ptr->dur;
            }else{          /* latest Calculation */
                if(lest[k] > lest[j]-ptr->dur)
                    lest[k]= lest[j]-ptr->dur;
            }

        } /* end of for statment. */

    } /* end of else */

} /*end of for statment*/

}/* end of function */

/*===== STACK PUSH =====*/
void push(int value) {

    if(top >= MAX_VERT){
        printf("STACK FULL!");
        exit(1);
    }
    stack[++top] = value;
}

/*===== STACK POP =====*/
int pop(){

    if((top)==-1){
        printf("STACK EMPTY!");
        exit(1);
    }
    return stack[top--];
}

/*===== print Vertex =====*/
int graph_print(hdnodes graph[],int max_vertex, int flag)
{

    int i;
    node_pointer ptr;
    ptr = (node_pointer)malloc(sizeof(node));

    if(flag==0)
        printf("===== Adjacent List =====Wn");
    else
        printf("WnWn===== Reverse Adjacent List =====Wn");

    printf("Vertex:Count   vertex:durationWn");

    for(i=0; i <= max_vertex; i++) /* 인접 리스트에 연결된 노드 모드 출력 */
    {
        printf(" %2d : %2d --> ", i, graph[i].count);
        for(ptr=graph[i].link; ptr; ptr=ptr->link) {
            printf(" %2d :", ptr->vertex);

```

```

        if(ptr->link != NULL)
            printf(" %2d    --> ", ptr->dur);
        else
            printf(" %2d", ptr->dur);
    }
    printf("\n");
}
return 0;
}

/*===== Critical activity =====*/
void crit_print( hdnodes graph[], int n)
{
    /* early() , late()를 구하고 critical activity 조사 */
    int i, early, late, cnt=0;
    node_pointer ptr;
    ptr = (node_pointer)malloc(sizeof(node));

    printf("\n\n===== Critical Activity ===== \n\n");
    printf("작업   Early   Late   Late-Early   임계성여부 \n");

    for(i=0; i < n; i++) {
        for(ptr=graph[i].link; ptr; ptr=ptr->link) {
            early=eest[i]; /* early() 계산 */
            late=lest[ptr->vertex]-ptr->dur; /* late() 계산 */
            printf("a[%2d]    %2d    %2d    %2d    %s \n",cnt , early, late,
                late-early, (late-early==0) ? "YES" : "NO" );
            cnt++;
        }
    }
}

```



## 6.7절

5. [프로그래밍 프로젝트] 그래프를 조작하는 C 프로그램을 작성하라. 이 프로그램은 임의의 그래프를 입력하여 그래프를 인쇄하고 연결 요소와 단절점과 브리지를 결정하여야 한다. 또한 신장트리들도 인쇄하여야 한다.

```
#include <stdio.h>
#include <stdlib.h>          /*for exit(), malloc(), free()*/
#include <string.h>          /*for memset()*/
#include <conio.h>           /*for getch()*/

#define TRUE 1
#define MAX_VERTICES 50 /*maximum size of vertex*/

#define IS_FULL(ptr) (!(ptr)) /*stack full*/
#define IS_EMPTY(ptr) (!(ptr)) /*stack empty*/
#define CLEAR(arr) memset(&arr, 0, sizeof(arr)) /*initialize array*/

/*for adjacency lists*/
typedef struct node *node_pointer;
typedef struct node {
    int vertex;
    node_pointer link;
}node;

/*for bfs, queue의 정의문*/
typedef struct queue *queue_pointer;
typedef struct queue {
    int vertex;
    queue_pointer link;
}queue;

/*for biconnected, stack의 정의문*/
typedef struct stack *stack_pointer;
typedef struct stack {
    int vertex1;
    int vertex2;
    stack_pointer link;
}stack;
stack_pointer top;

/*display menu*/
void prompt();

/*depth first search(DFS)*/
void dfs(int v);

/*breath frist search(BFS)*/
void bfs(int v);
void addq(queue_pointer *, queue_pointer *, int);
int deleteq(queue_pointer *);

/*connected component*/
void connected(int n);
```

```

/*determine dfn & low*/
void dfnlow(int u, int v);

/*insert graph and make adjacency lists*/
void read_graph(node_pointer *headnode);
int insert_graph(node_pointer *headnode, int vertex1, int vertex2);
void print_graph(node_pointer *graph);

/*biconnected graph*/
void biconnected(int u, int v);
void deletes(stack_pointer *top, int *v1, int *v2);
void adds(stack_pointer *top, int v1, int v2);

/*bridge*/
void bridge(int u, int v);

/*함수 array선언*/
node_pointer graph[MAX_VERTICES];          /*인접리스트 배열*/

short int visited[MAX_VERTICES];           /*방문 정점확인 배열*/

short int dfn[MAX_VERTICES];
short int low[MAX_VERTICES];

short int articulation_point[MAX_VERTICES]; /*단절점 표시 배열*/

/*전역 변수 선언*/
int child, root;                          /*루트의 단절점 여부확인*/
int vertices;                             /*전체 정점의 수를 위한 전역변수*/
int num;                                  /*dfn and low를 증가시키기 위한 전역변수*/

int main() {

    char menu;
    int i;

    /*user입력 선택*/
    while(1) {
        CLEAR(visited);

        prompt();
        printf("input menu : ");
        /*user로부터 메뉴를 입력받음*/
        menu = getch();
        printf("%c\n", menu);

        switch(menu) {
            case 'i' : case 'I' :
                read_graph(graph);
                break;
            case 'p' : case 'P' :
                print_graph(graph);
                break;
            case 'd' : case 'D' :

```

```

        printf("WnWn === Depth first search(DFS) ===WnWn");
        dfs(0);
        printf("Wn");
        break;
    case 'r' : case 'R' :
        printf("WnWn=== Breath first search(BFS) ===WnWn");
        bfs(0);
        printf("Wn");
        break;
    case 'c' : case 'C':
        printf("WnWn ===== connected component =====WnWn");
        connected(vertices);
        break;
    case 'a' : case 'A' :
        /*초기화*/
        child = num = 0;
        memset(&dfn, -1, sizeof(dfn));
        memset(&low, -1, sizeof(low));
        CLEAR(articulation_point);

        printf("root를 입력하세요 : ");
        scanf("%d", &root);

        dfnlow(root, -1);
        printf(" ===== Articulation Point =====Wn");

        for(i = 0; i<vertices; i++)
            if(articulation_point[i])
                printf("%2d", i);
        printf("Wn");
        break;
    case 'b' : case 'B':
        num = 0;
        memset(&dfn, -1, sizeof(dfn));
        memset(&low, -1, sizeof(low));
        bridge(3, -1);
        break;
    case 't' : case 'T' :
        num = 0;
        top = NULL;
        memset(&dfn, -1, sizeof(dfn));
        memset(&low, -1, sizeof(low));

        biconnected(3, -1);
        break;
    case 'x' : case 'X' :
        printf("Program terminated...Wn");
        exit(1);
    }
}

/*menu*/
void prompt() {
    printf("Wn===== Menu =====Wn");
    printf("= Insert Graph          : i =Wn");
}

```

```

printf("= Print  Graph          : p =Wn");
printf("= Depth first search(DFS) : d =Wn");
printf("= Breath first search(BFS) : r =Wn");
printf("= Connected Component      : c =Wn");
printf("= Articulation point       : a =Wn");
printf("= Bridge                     : b =Wn");
printf("= Biconnected component      : t =Wn");
printf("= Exit                       : x =Wn");
printf("=====Wn");
}

/*사용자로부터 정점과 간선을 입력받아 그래프를 생성*/
void read_graph(node_pointer *headnode) {

    int i;
    int repetition;                /*중복된 간선 체크*/
    int vertex, vertex1, vertex2;  /*간선들을 읽어들이는 변수*/
    int maxedge;                   /*최대 읽어들이 수 있는 간선의 수*/
    int count = 0;

    printf("Input the number of vertex(50보다 작게 입력하세요) : ");
    scanf("%d", &vertices);

    maxedge = vertices *(vertices - 1)/2;
    printf("입력 종료 => -1, Vertex start = 0Wn");

    while(count++ < maxedge) { /*최대 간선수가 되거나 사용자 입력 종료시까지*/
        printf("Insert edge(vertex1 vertex2) : ");
        scanf("%d", &vertex1);
        if(vertex1 == -1)
            break;
        scanf("%d", &vertex2);

        vertex = vertex1;

        i = 0;
        repetition = 0;

        /*undirected graph이므로 양쪽에 모두 추가*/
        /*directed graph로 사용시 while반복문을 사용 안하면 됨*/
        while((i++ < 2) && !repetition) {
            if((repetition = insert_graph(&graph[vertex], vertex, vertex2))) {
                count -= 1;
            }
            vertex = vertex2;
            vertex2 = vertex1;
        }
    }
}

/*리스트에 간선이 있는 정점을 추가 하는 함수*/
int insert_graph(node_pointer *headnode, int vertex1, int vertex2) {

```

```

int repetition = 0;
node_pointer element, trail, lists;

if(!(*headnode)) {          /*노드가 null이면*/
    lists = (node_pointer)malloc(sizeof(node));
    lists->vertex = vertex2;
    lists->link = NULL;
    *headnode = lists;
}

else {
    for(trail = *headnode; trail; trail = trail->link) { /*edge가 존재 하는지 확인*/
        element = trail;
        if(trail->vertex == vertex2) {
            printf("The edge is already exist!\n");
            repetition = 1;
            break;
        }
    }

    if(!repetition) { /*중복되는 간선이 없으면 추가*/
        lists = (node_pointer) malloc (sizeof(node));
        if(IS_FULL(lists)) {
            fprintf(stderr, "The memory is full\n");
            exit(1);
        }
        lists->vertex = vertex2;
        lists->link = NULL;
        element->link = lists;
    }
    else
        return repetition;
}
return repetition;
}

/*determine biconnected component*/
void biconnected(int u, int v){
    /*dfn, low를 계산하고 G의 이중결합 요소별로 간선들을 출력*/
    /*v는 결과 신장트리에서 u의 부모*/
    /*dfn[]=-1로 초기화, num=0으로 초기화, 스택은 비어있음*/
    node_pointer ptr;
    int w, x, y;

    dfn[u] = low[u] = num++;
    for(ptr = graph[u]; ptr; ptr = ptr->link) {
        w = ptr->vertex;
        if(v != w && dfn[w] < dfn[u])
            adds(&top, u, w); /*add edge to stack*/
        if(dfn[w] < 0) { /*w has not been visited*/
            biconnected(w, u);
            low[u] = low[u] < low[w] ? low[u] : low[w];
            if(low[w] >= dfn[u]) {
                printf("New biconnected component : ");
                do { /*delete edge from stack*/

```

```

        deletes(&top, &x, &y);
        printf("<%d, %d> ", x, y);
    }
    while(!((x==u) && (y==w)));
    printf("Wn");
}
}
else if(w != v)
    low[u] = low[u] < dfn[w] ? low[u] : dfn[w];
}
}

/*stack operation for bicon method*/
void adds(stack_pointer *top, int v1, int v2) {
    stack_pointer temp = (stack_pointer)malloc(sizeof(stack));
    if(IS_FULL(temp)) {
        fprintf(stderr, "The memory is fullWn");
        exit(1);
    }
    temp->vertex1 = v1;
    temp->vertex2 = v2;
    temp->link = *top;
    *top = temp; /*새로 추가된 node를 top*/
}

void deletes(stack_pointer *top, int *v1, int *v2) {
    stack_pointer temp = *top;
    if(IS_EMPTY(temp)) {
        fprintf(stderr, "The memory is emptyWn");
        exit(1);
    }
    *v1 = temp->vertex1;
    *v2 = temp->vertex2;
    *top = temp->link;      /*top의 node를 제거하고 새로운 top 설정*/
    free(temp);
}

/*determine bridge*/
void bridge(int u, int v){
    node_pointer ptr;
    int w;

    dfn[u] = low[u] = num++;
    for(ptr = graph[u]; ptr; ptr = ptr->link) {
        w = ptr->vertex;
        if(dfn[w] < 0) {
            bridge(w, u);
            low[u] = low[u] < low[w] ? low[u] : low[w];
            if(low[w] > dfn[u])
                printf("<%d, %d> ", u, w);
        }
        else if(w != v)
            low[u] = low[u] < dfn[w] ? low[u] : dfn[w];
    }
}
}

```

```

/*연결 그래프의 간선들을 이중결합 요소로 분할*/
void dfnlow(int u, int v) {
    node_pointer ptr;
    int w;
    dfn[u] = low[u] = num++;

    for(ptr = graph[u]; ptr; ptr = ptr->link) {
        w = ptr->vertex;
        if(dfn[w] < 0) { /*w는 방문이 안된 정점*/
            if(v == -1) /*루트가 둘이상의 자식을 가지는지 검사*/
                child++;
            dfnlow(w, u);
            /*자식의 low중 최소값*/
            low[u] = low[u] < low[w] ? low[u] : low[w];
            /*root이면서 child가 둘이상이거나 root가 아니면
            low(w) >= dfn(u)를 만족하면 단절점*/
            if (((u == root) && (child > 1)) || ((u != root) && (dfn[u] <= low[w])))
                articulation_point[u] = TRUE;
        }
        else if(w != v)
            low[u] = low[u] < dfn[w] ? low[u] : dfn[w];
    }
}

void connected(int n) {
    /*그래프의 연결 요소 결정*/
    int i;
    for (i = 0; i < n; i++)
        if(!visited[i]) {
            dfs(i);
            printf("Wn");
        }
}

/*깊이 우선 탐색*/
void dfs(int v) {
    node_pointer w;
    visited[v] = TRUE;
    printf("%3d", v);

    for(w = graph[v]; w; w = w->link)
        if(!visited[w->vertex])
            dfs(w->vertex);
}

/*너비 우선 탐색*/
void bfs(int v) {
    node_pointer w;
    queue_pointer front, rear;
    front = rear = NULL; /*initialize queue*/
    printf("%3d", v);
    visited[v] = TRUE;
    addq(&front, &rear, v);

    while(front) {
        v = deleteq(&front);
        for(w = graph[v]; w; w = w->link)

```

```

        if(!visited[w->vertex]) {
            printf("%3d", w->vertex);
            addq(&front, &rear, w->vertex);
            visited[w->vertex] = TRUE;
        }
    }
}

void addq(queue_pointer *front, queue_pointer *rear, int item) {
    /*큐의 rear에 원소를 삽입*/
    queue_pointer temp = (queue_pointer) malloc(sizeof(queue));
    if(IS_FULL(temp)) {
        fprintf(stderr, "The memory is full\n");
        exit(1);
    }
    temp->vertex = item;
    temp->link = NULL;
    if(*front)
        (*rear)->link = temp;
    else
        *front = temp;
    *rear = temp;
}

int deleteq(queue_pointer *front) {
    queue_pointer temp = *front;
    int item;
    if(IS_EMPTY(*front)) {
        fprintf(stderr, "The queue is empty\n");
        exit(1);
    }
    item = temp->vertex;
    *front = temp->link;
    free(temp);
    return item;
}

/*그래프를 출력하는 함수*/
void print_graph(node_pointer *graph) {
    int i;
    node_pointer ptr;

    for(i = 0; i<vertices; i++) {
        ptr = graph[i];
        printf("Head[%d] : ", i);

        for(;ptr;ptr = ptr->link) {
            if(!(ptr->vertex == -1)) {
                printf("%4d", ptr->vertex);
            }
        }
        printf("\n");
    }
}

```