자료구조 보고서

Homework#11

학과 : 소프트웨어학과
학번 : 2016039040
이름 : 윤용진

1. Graph 기능 구현에 관한 것이다.

(a) 다음과 같은 메뉴기반의 프로그램이 실행되도록 graph-search.c를 구현하라.

```
---------------------------------------------------------------
                        Graph Searches
---------------------------------------------------------------
Initialize Graph = z
Insert Vertex = v I             nsert Edge = e
Depth First Search = d          Breath First Search = b
Print Graph = p                 Quit = q
---------------------------------------------------------------
```

(b) Graph에 대한 자료구조는 인접리스트 (Adjacent List)로 구현한다.

(c) 최대 Vertex의 수는 10개로 하고, Vertex 번호는 0부터 9까지 부여된다.

(d) DFS, BFS를 통해 탐색을 할 때 여러 Edge가 있을 경우 Vertex의 번호가 작은 Vertex를 먼저 탐색한다.

(e) 구현한 함수들은 적절한 오류처리 기능이 있어야 한다.

(f) 코드의 기능을 이해할 수 있도록 충분한 주석을 작성한다.

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

/* 최대 정점의 수는 10개로 설정 */
#define MAX_VERTEX_NUM 10
#define MAX_QUEUE_NUM 10

/* 참거짓 정의 */
#define TRUE 1
#define FALSE 0

/* int형의 큐 생성 */
int Queue[MAX_QUEUE_NUM];
int front = -1;
int rear = -1;

/* 정점 구조체 */
typedef struct Vertex {
        int VertexNum;
        struct List* list;
        int visitflag;
} Vertex;

/* 간선 표현을 위한 리스트 구조체 */
typedef struct List {
```

```
        int adjNode;
        struct List* list;
        int visitflag;
} List;
/* 두 구조체는 같은 구조를 가지지만,
*  헤드노드와 리스트노드를 구분하기 위해 둘로 나누었다.*/

void InitializeGraph(Vertex**);
void freeGraph(Vertex**);
void InsertVertex(Vertex**);
void InsertEdge(int tail, int head,Vertex**);
void printGraph(Vertex **);
void makeList(Vertex* v, int i);
void DepthFirstSearch(Vertex**, int);
void BreathFirstSearch(Vertex**, int);
int deQueue();
void enQueue(int aNode);
void InitializeVisitFlag(Vertex** head);

int main(void)
{
        char command;
        int a, b;
        /* 헤드노드(정점)의 배열 선언 */
        Vertex* head[MAX_VERTEX_NUM] = {NULL};

        printf("[----- [Yoon Yong Jin] [2016039040] -----]");

        /* 무방향성의 인접그래프로 설정 */

        do {
                printf("\n\n");

printf("----------------------------------------------------------------\n");
                printf("                                        Graph  Searches
  \n");

printf("----------------------------------------------------------------\n");
                printf(" Initialize Graph          = z
                                \n");
                printf(" Insert Vertex             = v      Insert Edge              =
e \n");
                printf(" Depth First Search        = d      Breath First Search
```

```
= b \n");
                printf(" Print Graph          = p      Quit                    =
q \n");

printf("-----------------------------------------------------------------\n");

                printf("Command = ");
                scanf(" %c", &command);

                switch (command) {
                case 'z': case 'Z':
                        /* 그래프 초기화 */
                        InitializeGraph(&head);
                        break;
                case 'q': case 'Q':
                        /* 그래프 메모리 할당 해제 */
                        freeGraph(&head);
                        break;
                case 'v': case 'V':
                        /* 정점 추가 */
                        InsertVertex(&head);
                        break;
                case 'e': case 'E':
                        /* 그래프 초기화 전처리 */
                        if (*head == NULL)
                        {
                                printf("Initialize First\n");
                                break;
                        }
                        /* 현재 그래프를 출력해 어떤 정점들을 연결할 수 있는지 확인 */
                        printf("Current Graph\n");
                        printGraph(&head);
                        /* 간선으로 연결할 두 정점을 입력받는다 */
                        printf("Enter the number of the two vertices to connect.\n");
                        scanf("%d %d", &a, &b);
                        InsertEdge(a, b, &head);
                        break;
                case 'd': case 'D':
                        /* 그래프 초기화 전처리 */
                        if (*head == NULL)
                        {
                                printf("Initialize First\n");
                                break;
```

```
                        }
                        /* 깊이 우선 탐색을 시작할 정점을 선택한다. */
                        printf("Enter the number of the vertex to start.\n");
                        scanf("%d", &a);
                        /* 정점 a에서부터 시작하는 깊이우선 탐색 */
                        DepthFirstSearch(&head,a);
                        /* 각 노드들의 방문여부를 초기화해, 다음 탐색이 가능하도록 한
다. */
                        InitializeVisitFlag(head);
                        break;
                case 'b': case 'B':
                        /* 그래프 초기화 전처리 */
                        if (*head == NULL)
                        {
                                printf("Initialize First\n");
                                break;
                        }
                        /* 너비 우선 탐색을 시작할 정점을 선택한다. */
                        printf("Enter the number of the vertex to start.\n");
                        scanf("%d", &a);
                        /* 정점 a에서부터 시작하는 너비우선 탐색 */
                        BreathFirstSearch(&head,a);
                        /* 각 노드들의 방문여부를 초기화해, 다음 탐색이 가능하도록 한
다. */
                        InitializeVisitFlag(head);
                        break;
                case 'p': case 'P':
                        /* 그래프 출력 */
                        printGraph(&head);
                        break;
                default:
                        printf("\n       >>>>>   Concentration!!   <<<<<      \n");
                        break;
                }

        } while (command != 'q' && command != 'Q');

        return 1;
}

/* 그래프 초기화함수 */
void InitializeGraph(Vertex** head)
{
```

```
	/* 기존 그래프가 있는 경우 메모리 해제 후 초기화 */
	if (*head != NULL)
		freeGraph(head);

	/* Vertex[] 초기화 */
	for (int i = 0; i < MAX_VERTEX_NUM; i++)
	{
		head[i] = (Vertex*)malloc(sizeof(Vertex));
		/* 해당 정점의 이름(번호), 정점을 생성하기 전에는 -1로 초기화 */
		head[i]->VertexNum = -1;
		/* 정점에 간선으로 연결된 리스트 노드 */
		head[i]->list = NULL;
		/* 탐색 시 해당 노드 방문 여부 */
		head[i]->visitflag = FALSE;
	}
}

/* 메모리 해제 */
void freeGraph(Vertex** head)
{
	for (int i = 0; i < MAX_VERTEX_NUM; i++)
	{
		free(head[i]);
	}
}

/* 정점 삽입 */
void InsertVertex(Vertex** head)
{
	/* 그래프 초기화 전처리 */
	if (*head == NULL)
	{
		printf("Initialize First\n");
		return;
	}

	/* 기존에 초기화한, Vertex[] 탐색 */
	for (int i = 0; i < MAX_VERTEX_NUM; i++)
	{
		/* 가장 처음으로 만나는 VertexNum이 -1인 노드는 해당 노드가 아직 추
가되지 않아, 이번에 추가할 노드임을 알 수 있다. */
		if (head[i]->VertexNum == -1)
		{
```

```
                        /* 노드 이름 부여 */
                        head[i]->VertexNum = i;
                        /* 간선으로 연결된 리스트 노드 없음 */
                        head[i]->list = NULL;
                        /* 아직 방문하지 않음으로 초기화 */
                        head[i]->visitflag = FALSE;
                        break; /* 후 종료 -> 한번에 하나의 노드만 추가 */
                }
        }
}

/* 간선 추가, 연결할 두 노드의 이름(번호)를 입력받음 */
void InsertEdge(int tail,int head,Vertex** h)
{
        /* 그래프 초기화 전처리 */
        if (*h == NULL)
        {
                printf("Initialize First\n");
                return;
        }

        /* 무방향 그래프 이므로 하나의 엣지당 두개의 리스트 노드가 필요하다. */
        makeList(h[tail], head);
        makeList(h[head], tail);
}

/* 리스트 노드 생성 */
/* DFS, BFS를 통해 탐색을 할 때 여러 Edge가 있을 경우,
*  정점의 번호가 작은노드먼저 탐색하도록 하기 위해
*  리스트 노드를 오름차순으로 정렬해야한다. */
void makeList(Vertex* v, int i)
{
        /* Vertex[i]의 리스트 노드들을 탐색하기 위한 p를 선언 */
        List* p;
        List* prev = (List*)malloc(sizeof(List));
        /* 새롭게 추가할 리스트 노드 생성 */
        List* temp = (List*)malloc(sizeof(List));
        temp->adjNode = i;
        temp->list = NULL;
        temp->visitflag = 0;

        /* 리스트 노드 탐색 */
        p = v;
```

```
        /* 첫 리스트노드가 비어있는경우 == 아직 연결된 노드가 없는 경우 */
        if (p == NULL);
        /* 기존에 연결된 리스트 노드가 있는 경우 */
        else
        {
                do
                {
                        /* 자기 간선 및 동일 간선의 중복 제외 */
                        if (p->adjNode == i) return;
                        /* 오름차순 삽입 */
                        if (p!=v && p->adjNode > i)/* i보다 큰 번호의 리스트 노드 발
견시 앞에 리스트노드 삽입 */
                        {
                                temp->list = p;
                                prev->list = temp;
                                return;
                        }
                        /* 다음 리스트 노드 탐색 */
                        prev = p;
                        p = p->list;
                } while (p != NULL);
        }
        prev->list = temp;/* 리스트노드 삽입 */
}

/* 그래프 출력 */
void printGraph(Vertex **head)
{
        /* 그래프 초기화 전처리 */
        if (*head == NULL)
        {
                printf("Initialize First\n");
                return;
        }

        /* Vertex[]를 탐색 */
        Vertex* p;
        /* p=Vertex[i]의 리스트 노드들을 탐색 */
        List* p2;
        int i = 0;

        p = head[0];
        /* 최대 10개의 정점을 탐색한다. */
```

```
        while (i < MAX_VERTEX_NUM && p->VertexNum != -1)
        {
                /* p에 해당하는 헤드노드 출력 */
                printf("헤드노드 [%d]", i);
                /* 헤드노드 하위의 리스트 노드들을 탐색*/
                p2 = p->list;
                while (p2 != NULL)
                {
                        /* 리스트 노드 출력 */
                        printf(" -> ");
                        printf("[%d]",p2->adjNode);
                        p2 = p2->list;
                }
                printf("\n");
                i++;
                p = head[i];
        }
}

/* 깊이우선 탐색 */
void DepthFirstSearch(Vertex** head, int i)
{
        /* 그래프 초기화 전처리 */
        if (*head == NULL)
        {
                printf("Initialize First\n");
                return;
        }
        /* 10이상의 번호로 탐색 시도 처리 */
        if(i>=MAX_VERTEX_NUM)
        {
                printf("Out of range\n");
                return;
        }
        /* 생성되지 않은 정점 탐색 시도 */
        if(head[i]->VertexNum == -1)
        {
                printf("There is no [%d] Vertex\n",i);
                return;
        }

        List* p;
        int min;
```

```
        /* 현재 정점노드 방문여부 참으로 수정 */
        head[i]->visitflag = TRUE;
        /* 탐색한 현재 노드 출력 */
        printf("%5d", i);
        /* 아직 방문하지 않은 간선으로 연결된 노드 탐색 */
        for (p = head[i]; p; p = p->list)
        {
                /* 방문하지 않은 노드 발견시 */
                if (head[p->adjNode]->visitflag==FALSE)
                        /* 재귀적으로 탐색 */
                        DepthFirstSearch(head, p->adjNode);
        }
        /* 정점에서 간선으로 연결된 노드를 탐색 하고 탐색한 노드를 다시 정점으로 삼아
탐색하기 때문에 깊이 우선 탐색이 이루어진다. */
}

/* 너비우선 탐색 */
void BreathFirstSearch(Vertex** head, int i)
{
        /* 그래프 초기화 전처리 */
        if (*head == NULL)
        {
                printf("Initialize First\n");
                return;
        }
        /* 10이상의 번호로 탐색 시도 처리 */
        if (i >= MAX_VERTEX_NUM)
        {
                printf("Out of range\n");
                return;
        }
        /* 생성되지 않은 정점 탐색 시도 */
        if(head[i]->VertexNum == -1)
        {
                printf("There is no [%d] Vertex\n",i);
                return;
        }

        List* p;
        front = rear = -1;
        /* 현재 탐색 노드 출력 */
        printf("%5d", i);
        /* 현재 정점노드 방문여부 참으로 수정 */
```

```
        head[i]->visitflag = TRUE;
        /* 현재 노드 큐에 삽입 */
        enQueue(i);

        /* 무한반복 */
        while (1) {
                /* 큐로부터 탐색할 정점 노드의 번호 추출*/
                i = deQueue(); /* dequeue */
                if (i==-1) break;/* 비어있는 노드가 추출되면 반복 종료 */
                for (p = head[i]; p; p = p->list)/* 해당 정점 노드의 리스트 노드들을 탐
색 */
                        /* 아직 방문하지 않은 노드 발견 시 */
                        if (head[p->adjNode]->visitflag == FALSE)
                        {
                                printf("%5d", p->adjNode);/* 출력 */
                                enQueue(p->adjNode);/* 큐에 삽입 */
                                head[p->adjNode]->visitflag = TRUE;/* 방문 했던 노
드임을 기억한다. */
                        }
        }
        /* 탐색 우선 순위를 큐에 저장한뒤 큐를 기준으로 탐색하기 때문에 너비 우선 탐색
이 이루어진다. */
}

/* 큐 deQueue 이진 탐색 트리2의 큐를 재활용 */
int deQueue()
{
        /* 큐가 공백이면 종료 */
        if (front == rear) return -1;/* front와 rear의 값이 같다면 큐에 어떤 노드도 없
다.
                                                (int형 큐를 선언하
였기 때문에 NULL은 0으로 0번 정점과 겹쳐 탐색에 방해가 되므로 -1로 대체) */
        front++;/* 큐의 시작을 가르키는(front 방향으로 노드가 큐를 탈출(삭제)함) front값
을 1만큼 증가시킨다. */
        int temp = Queue[front];/* 삭제할 노드 저장 */
        Queue[front] = -1;/* 노드 삭제(int형 큐를 선언하였기 때문에 NULL은 0으로 0번
정점과 겹쳐 탐색에 방해가 되므로 -1로 대체) */
        return temp;/* 노드 반환 */
}

/* 큐 enQueue */
void enQueue(int aNode)
{
```

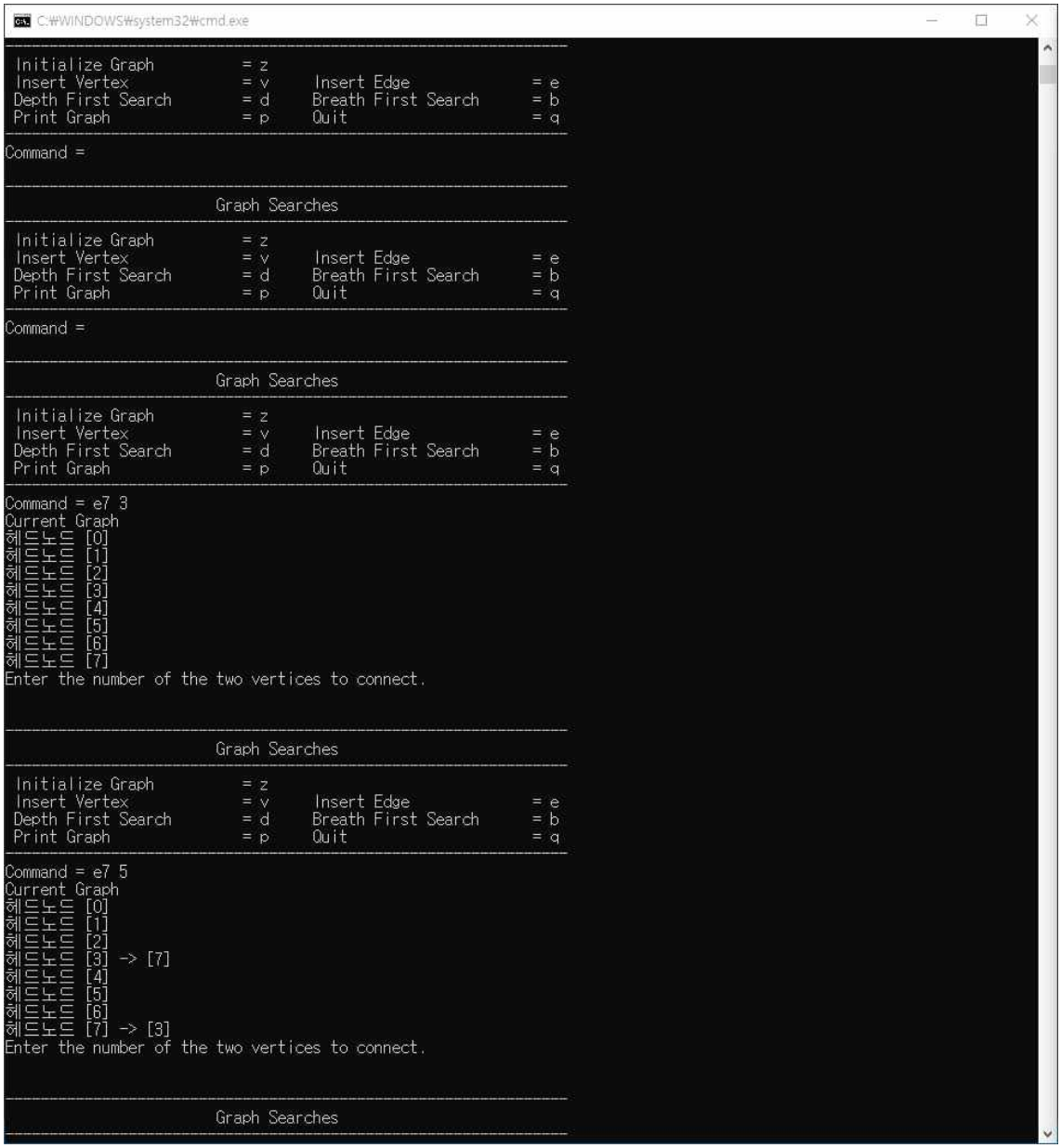
```
        /* 큐가 가득 찼다면 */
        if (rear == MAX_QUEUE_NUM - 1)
        {
                /* 선형 큐 앞으로 끌어당기기 */
                int size = rear - front;/* 큐에 남아있는 노드의 수를 구한다. */
                for (int i=0;i<size;i++)/* 남은 노드 수 만큼 반복 */
                {
                        Queue[i] = Queue[front+(i+1)];/* 노드들을 큐의 앞으로 이동시킨
다*/
                        Queue[front + (i + 1)] = NULL;
                }
                front = -1;/* front 초기화 */
                rear = size-1; /* rear값은 size-1로 초기화 하여 front와 rear간의 거리
유지 */
                return;
        }
        rear++;/* 큐의 끝을 가르키는 rear값을 1만큼 증가시킨다.*/
        Queue[rear] = aNode;/* 새로운 노드 삽입 */
}

/* VisitFlag 초기화, 한번 탐색한 이후 초기화 하지 않으면,
*  모든 노드를 이미 방문한 것으로 기억하기 때문에 다음 탐색이 불가능하다. */
void InitializeVisitFlag(Vertex** head)
{
        /* 그래프 초기화 전처리 */
        if (*head == NULL) return;

        List* p;
        for (int i = 0; i < MAX_VERTEX_NUM && head[i]->VertexNum!= -1; i++)
        {   /* Vertex[]의 노드 중 insertVertex()함수로 생성된 정점들을 탐색 */
                for (p = head[i]; p; p = p->list) /* 정점과, 정점에 연결된 리스트 노드들
을 탐색하여 방문여부 초기화 */
                {
                        p->visitflag = FALSE; /* 방문여부 초기화 */
                }
        }
}
```

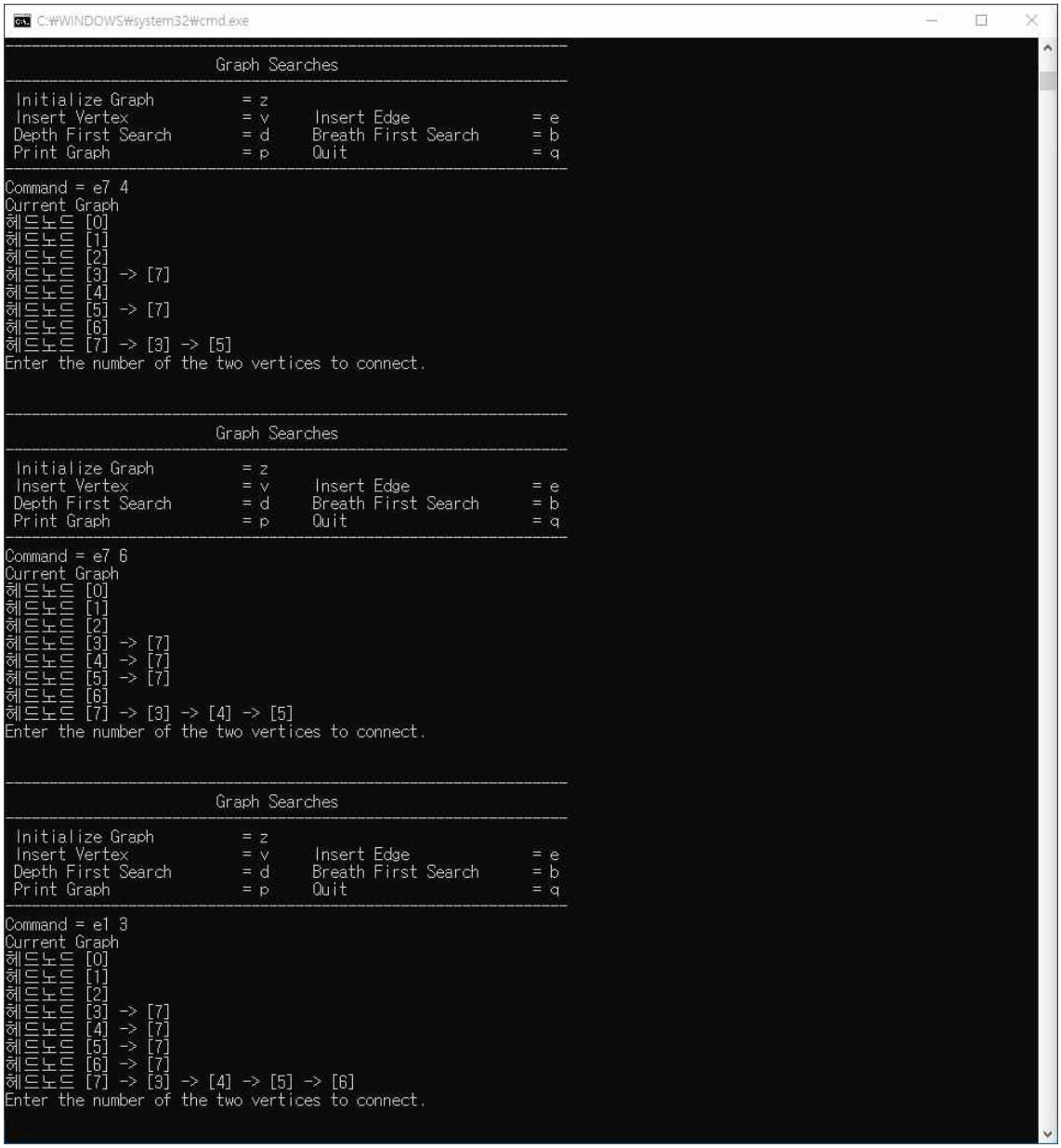
4. GitHub에 hw11 Repository를 생성하고 graph-search.c를 업로드 한다.

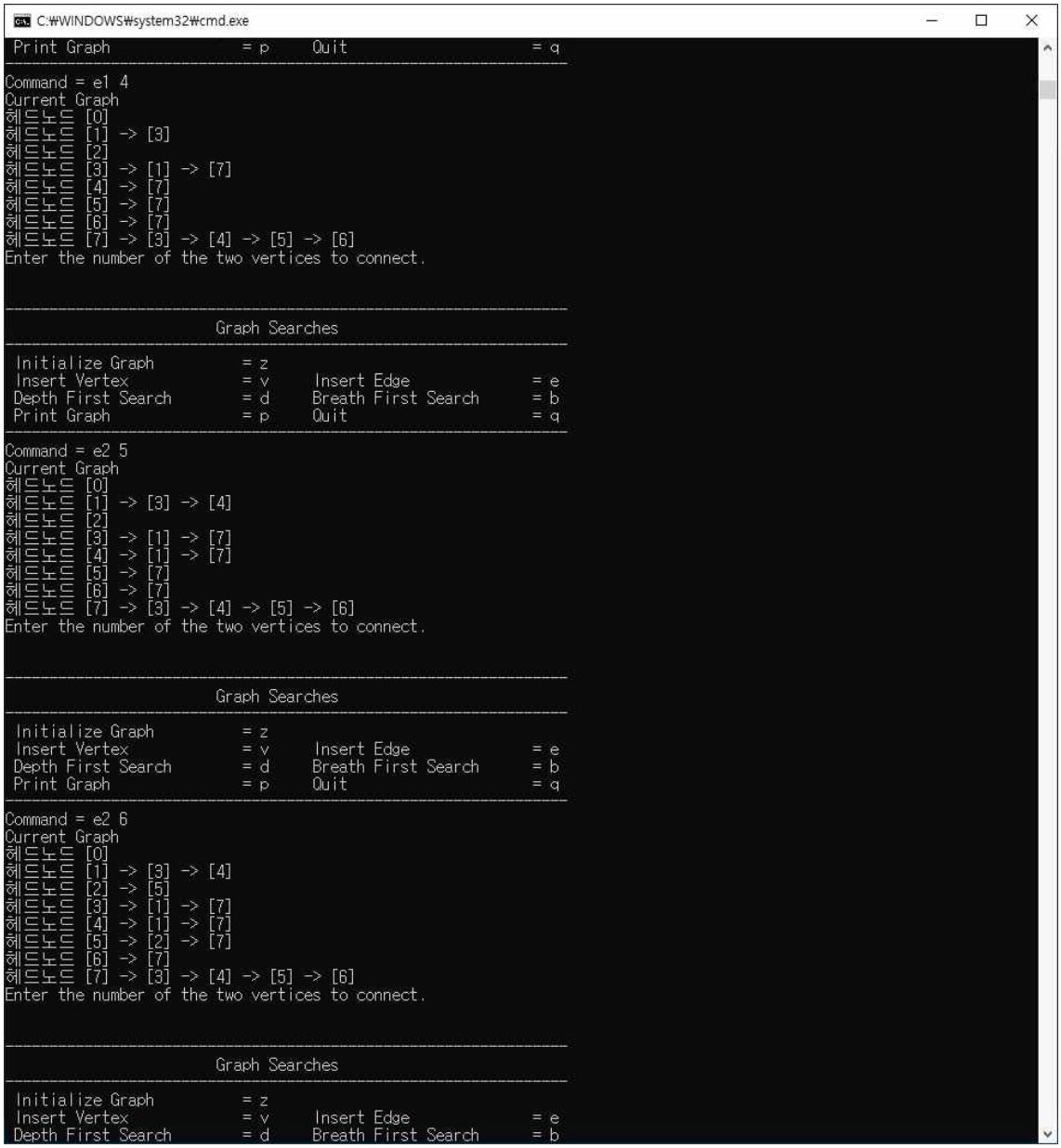
https://github.com/uniz21/DataStructure-HW-11

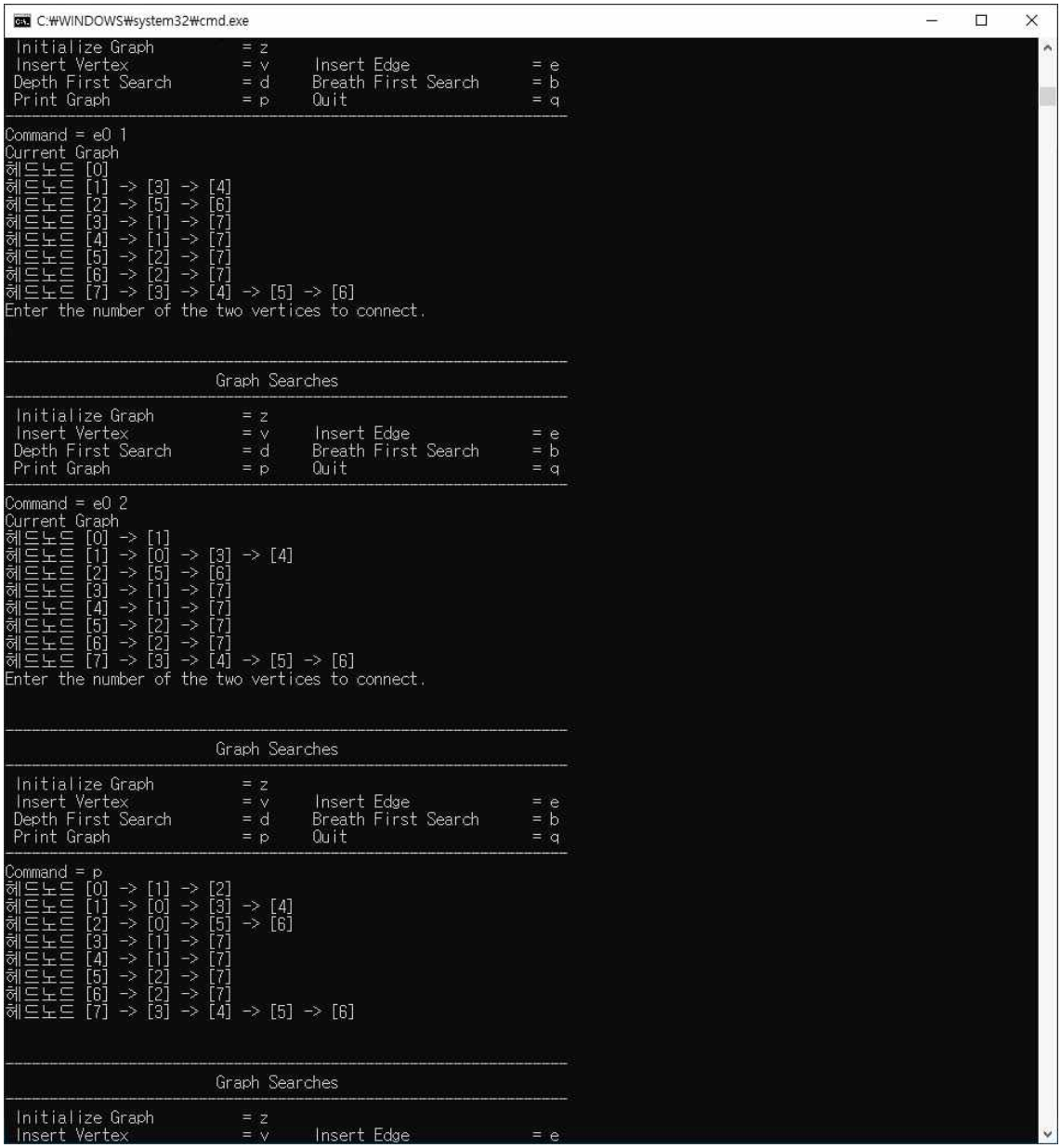
실행결과

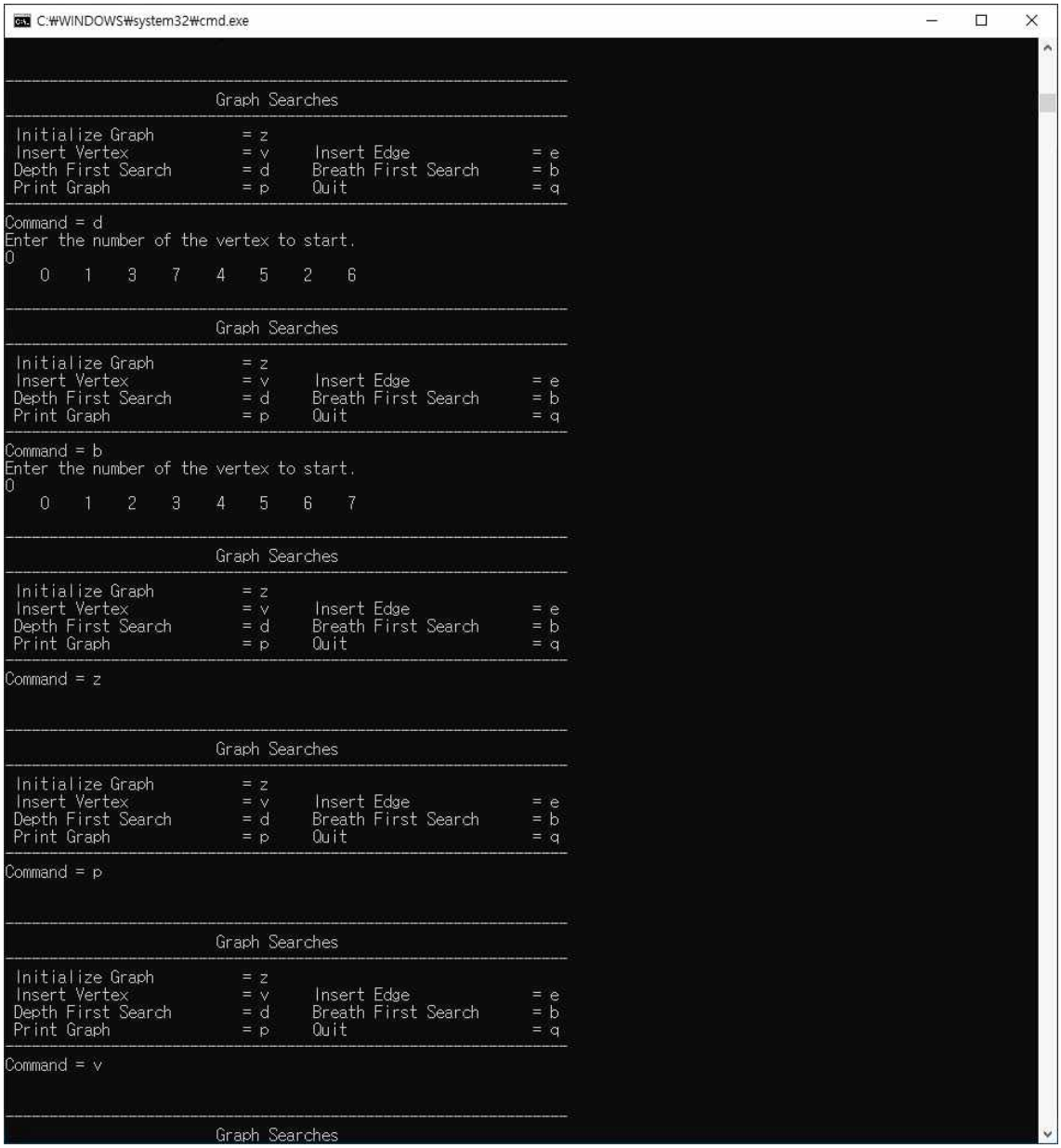
```
C:\WINDOWS\system32\cmd.exe
[------ [Yoon Yong Jin] [2016039040] ------]

-----------------------------------------------------------------------
                          Graph Searches
-----------------------------------------------------------------------
 Initialize Graph             = z
 Insert Vertex                = v      Insert Edge                = e
 Depth First Search           = d      Breath First Search        = b
 Print Graph                  = p      Quit                       = q
-----------------------------------------------------------------------
Command = zvvvvvvvvv

-----------------------------------------------------------------------
                          Graph Searches
-----------------------------------------------------------------------
 Initialize Graph             = z
 Insert Vertex                = v      Insert Edge                = e
 Depth First Search           = d      Breath First Search        = b
 Print Graph                  = p      Quit                       = q
-----------------------------------------------------------------------
Command =
-----------------------------------------------------------------------
                          Graph Searches
-----------------------------------------------------------------------
 Initialize Graph             = z
 Insert Vertex                = v      Insert Edge                = e
 Depth First Search           = d      Breath First Search        = b
 Print Graph                  = p      Quit                       = q
-----------------------------------------------------------------------
Command =
-----------------------------------------------------------------------
                          Graph Searches
-----------------------------------------------------------------------
 Initialize Graph             = z
 Insert Vertex                = v      Insert Edge                = e
 Depth First Search           = d      Breath First Search        = b
 Print Graph                  = p      Quit                       = q
-----------------------------------------------------------------------
Command =
-----------------------------------------------------------------------
                          Graph Searches
-----------------------------------------------------------------------
 Initialize Graph             = z
 Insert Vertex                = v      Insert Edge                = e
 Depth First Search           = d      Breath First Search        = b
 Print Graph                  = p      Quit                       = q
-----------------------------------------------------------------------
Command =
-----------------------------------------------------------------------
                          Graph Searches
-----------------------------------------------------------------------
 Initialize Graph             = z
 Insert Vertex                = v      Insert Edge                = e
 Depth First Search           = d      Breath First Search        = b
 Print Graph                  = p      Quit                       = q
-----------------------------------------------------------------------
Command =
```

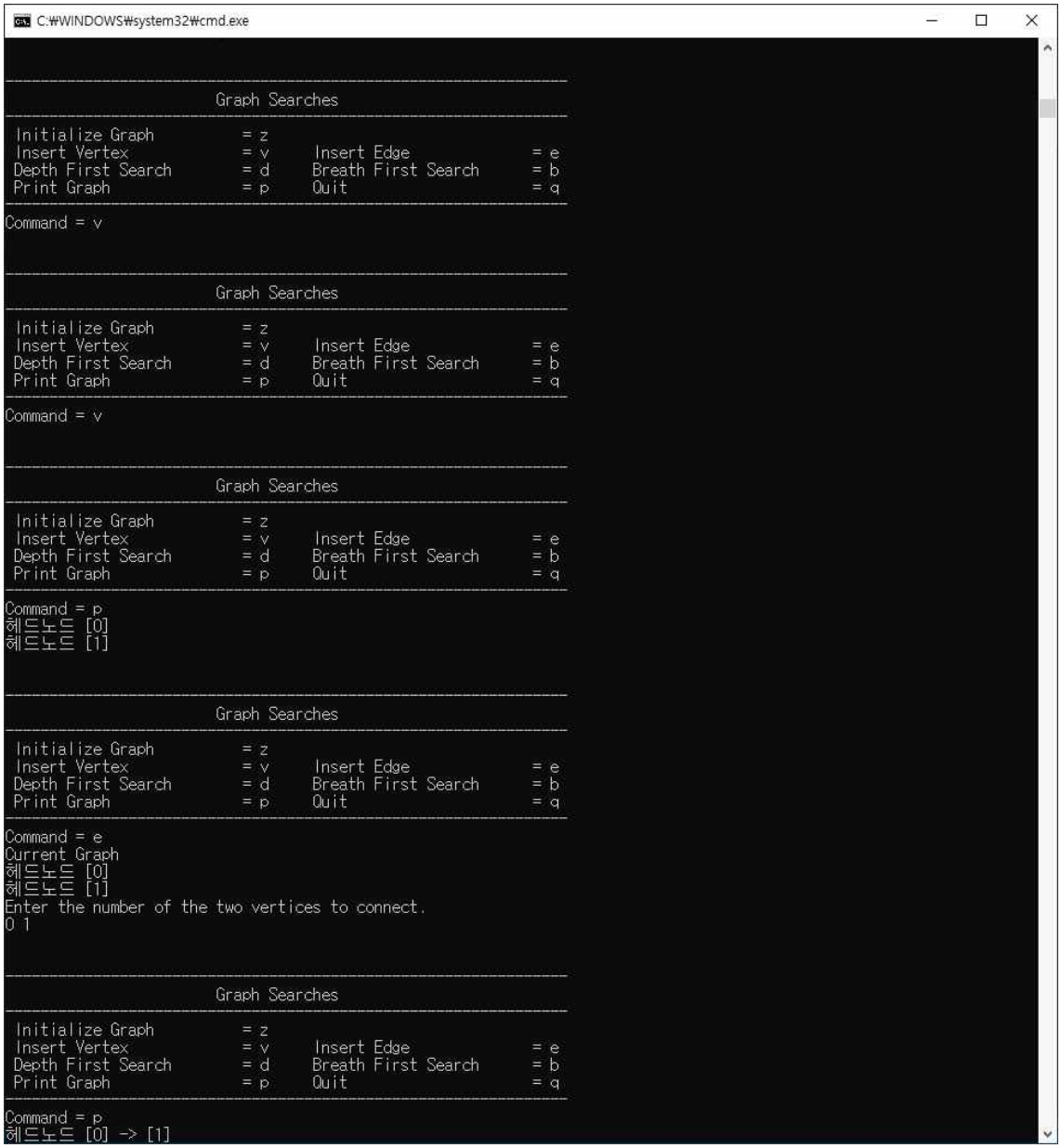

C:₩WINDOWS₩system32₩cmd.exe
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command =
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command =
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command = e7 3
Current Graph
헤드노드 [0]
헤드노드 [1]
헤드노드 [2]
헤드노드 [3]
헤드노드 [4]
헤드노드 [5]
헤드노드 [6]
헤드노드 [7]
Enter the number of the two vertices to connect.
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command = e7 5
Current Graph
헤드노드 [0]
헤드노드 [1]
헤드노드 [2]
헤드노드 [3] -> [7]
헤드노드 [4]
헤드노드 [5]
헤드노드 [6]
헤드노드 [7] -> [3]
Enter the number of the two vertices to connect.
Graph Searches


C:₩WINDOWS₩system32₩cmd.exe
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = e7 4
Current Graph
헤드노드 [0]
헤드노드 [1]
헤드노드 [2]
헤드노드 [3] -> [7]
헤드노드 [4]
헤드노드 [5] -> [7]
헤드노드 [6]
헤드노드 [7] -> [3] -> [5]
Enter the number of the two vertices to connect.
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = e7 6
Current Graph
헤드노드 [0]
헤드노드 [1]
헤드노드 [2]
헤드노드 [3] -> [7]
헤드노드 [4] -> [7]
헤드노드 [5] -> [7]
헤드노드 [6]
헤드노드 [7] -> [3] -> [4] -> [5]
Enter the number of the two vertices to connect.
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = e1 3
Current Graph
헤드노드 [0]
헤드노드 [1]
헤드노드 [2]
헤드노드 [3] -> [7]
헤드노드 [4] -> [7]
헤드노드 [5] -> [7]
헤드노드 [6] -> [7]
헤드노드 [7] -> [3] -> [4] -> [5] -> [6]
Enter the number of the two vertices to connect.


C:₩WINDOWS₩system32₩cmd.exe
Print Graph = p Quit = q
Command = e1 4
Current Graph
헤드노드 [0]
헤드노드 [1] -> [3]
헤드노드 [2]
헤드노드 [3] -> [1] -> [7]
헤드노드 [4] -> [7]
헤드노드 [5] -> [7]
헤드노드 [6] -> [7]
헤드노드 [7] -> [3] -> [4] -> [5] -> [6]
Enter the number of the two vertices to connect.
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command = e2 5
Current Graph
헤드노드 [0]
헤드노드 [1] -> [3] -> [4]
헤드노드 [2]
헤드노드 [3] -> [1] -> [7]
헤드노드 [4] -> [1] -> [7]
헤드노드 [5] -> [7]
헤드노드 [6] -> [7]
헤드노드 [7] -> [3] -> [4] -> [5] -> [6]
Enter the number of the two vertices to connect.
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command = e2 6
Current Graph
헤드노드 [0]
헤드노드 [1] -> [3] -> [4]
헤드노드 [2] -> [5]
헤드노드 [3] -> [1] -> [7]
헤드노드 [4] -> [1] -> [7]
헤드노드 [5] -> [2] -> [7]
헤드노드 [6] -> [7]
헤드노드 [7] -> [3] -> [4] -> [5] -> [6]
Enter the number of the two vertices to connect.
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b


C:₩WINDOWS₩system32₩cmd.exe
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command = e0 1
Current Graph
헤드노드 [0]
헤드노드 [1] -> [3] -> [4]
헤드노드 [2] -> [5] -> [6]
헤드노드 [3] -> [1] -> [7]
헤드노드 [4] -> [1] -> [7]
헤드노드 [5] -> [2] -> [7]
헤드노드 [6] -> [2] -> [7]
헤드노드 [7] -> [3] -> [4] -> [5] -> [6]
Enter the number of the two vertices to connect.
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command = e0 2
Current Graph
헤드노드 [0] -> [1]
헤드노드 [1] -> [0] -> [3] -> [4]
헤드노드 [2] -> [5] -> [6]
헤드노드 [3] -> [1] -> [7]
헤드노드 [4] -> [1] -> [7]
헤드노드 [5] -> [2] -> [7]
헤드노드 [6] -> [2] -> [7]
헤드노드 [7] -> [3] -> [4] -> [5] -> [6]
Enter the number of the two vertices to connect.
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e
Depth First Search = d Breath First Search = b
Print Graph = p Quit = q
Command = p
헤드노드 [0] -> [1] -> [2]
헤드노드 [1] -> [0] -> [3] -> [4]
헤드노드 [2] -> [0] -> [5] -> [6]
헤드노드 [3] -> [1] -> [7]
헤드노드 [4] -> [1] -> [7]
헤드노드 [5] -> [2] -> [7]
헤드노드 [6] -> [2] -> [7]
헤드노드 [7] -> [3] -> [4] -> [5] -> [6]
Graph Searches
Initialize Graph = z
Insert Vertex = v Insert Edge = e


C:¥WINDOWS¥system32¥cmd.exe
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = d
Enter the number of the vertex to start.
0
0 1 3 7 4 5 2 6
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = b
Enter the number of the vertex to start.
0
0 1 2 3 4 5 6 7
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = z
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = p
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = v
Graph Searches


C:₩WINDOWS₩system32₩cmd.exe
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = v
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = v
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = p
헤드노드 [0]
헤드노드 [1]
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = e
Current Graph
헤드노드 [0]
헤드노드 [1]
Enter the number of the two vertices to connect.
0 1
Graph Searches
Initialize Graph = z
Insert Vertex = v
Insert Edge = e
Depth First Search = d
Breath First Search = b
Print Graph = p
Quit = q
Command = p
헤드노드 [0] -> [1]