그래프

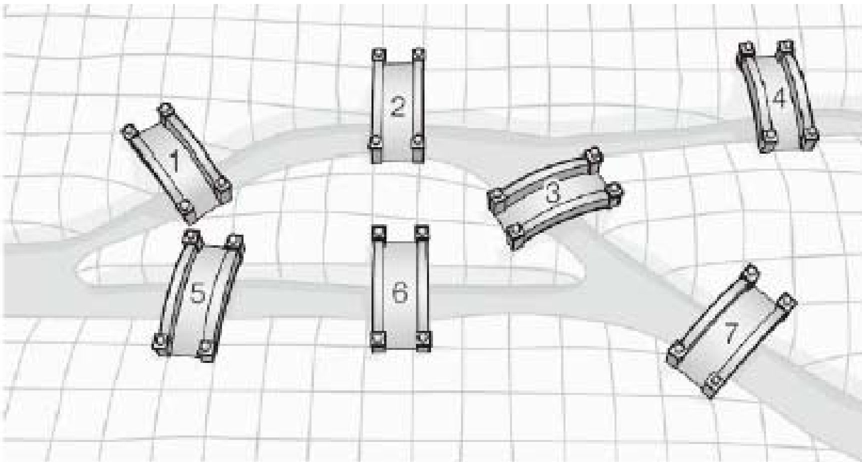
13 주차-강의

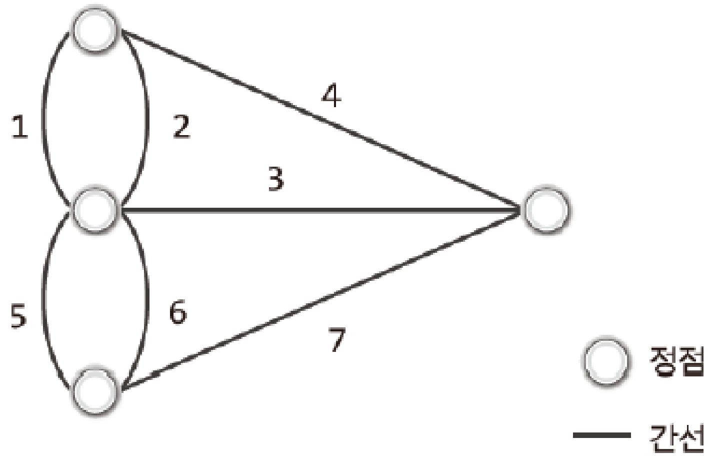
남춘성

SUNG KYUN KWAN UNIVERSITY

그래프의 개념적 이해 - I

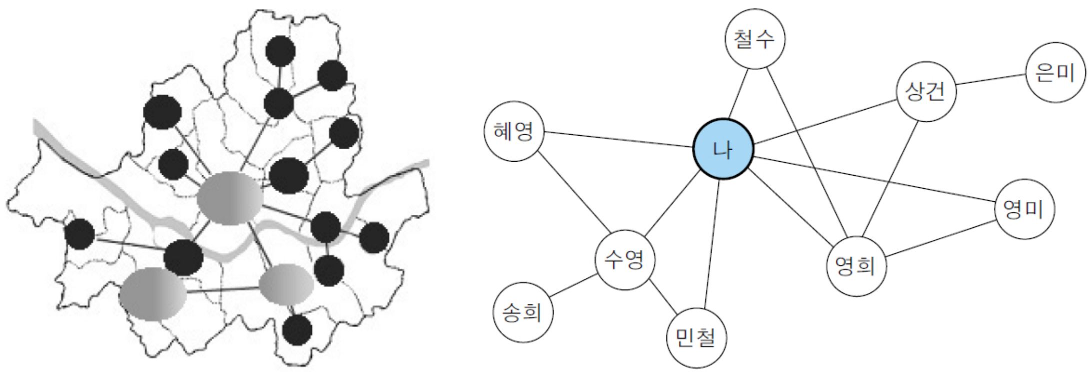
- 오일러(Euler)에 의해 고안
  - '쾨니히스베르크의 다리 문제'를 풀기 위해 그래프 이론을 고안
  - 모든 다리를 한번씩만 건너서 처음 출발했던 장소로 돌아올 수 있을까?
  - "정점 별로 연결된 간선의 수가 모두 짝수이어야 간선을 한 번씩만 지나서 처음 출발했던 정점으로 돌아올 수 있음"





그래프의 개념적 이해 - II

- **그래프(graph)**
  - 선형 자료구조나 트리 자료구조로 표현하기 어려운 n:n의 관계를 가지는 원소들을 표현하기 위한 자료구조
  - 그래프 G
    - 객체를 나타내는 정점(vertex)과 객체를 연결하는 간선(edge)의 집합
    - G = (V, E)
    - V는 그래프에 있는 정점들의 집합
    - E는 정점을 연결하는 간선들의 집합



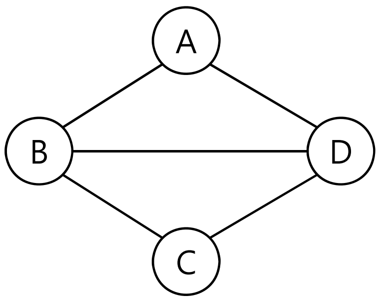
[그림 9-1] 그래프의 사용 예: 버스 노선도, 인맥 지도

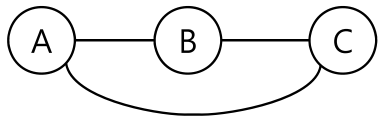
그래프의 개념적 이해 - III

SUNG KYUN KWAN UNIVERSITY

- **그래프(graph)의 종류**
  - 무방향 그래프(undirected graph)
    - 두 정점을 연결하는 간선의 방향이 없는 그래프
  - 방향 그래프(directed graph) , 다이그래프(digraph)
    - 간선이 방향을 가지고 있는 그래프
  - 완전 그래프(complete graph)
    - 각 정점에서 다른 모든 정점을 연결하여 가능한 최대의 간선 수를 가진 그래프
  - 부분 그래프(subgraph)
    - 원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프
  - 가중 그래프(weight graph) , 네트워크(network)
    - 정점을 연결하는 간선에 가중치(weight)를 할당한 그래프

- **그래프(graph)의 종류**
  - 무방향 그래프(undirected graph)
    - 두 정점을 연결하는 간선의 방향이 없는 그래프
    - 정점 Vi와 정점 Vj을 연결하는 간선을 (Vi, Vj)로 표현
      - (Vi, Vj)와 (Vj, Vi)는 같은 간선을 의미
    - V(G1)={A, B, C, D}　　E(G1)={(A,B), (A,D), (B,C), (B,D), (C,D)}
    - V(G2)={A, B, C}　　E(G2)={(A,B), (A,C), (B,C)}

G1


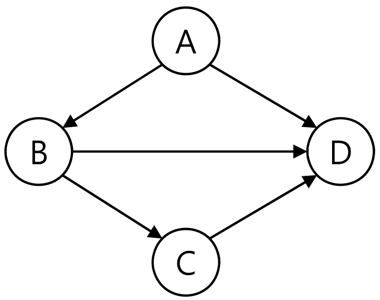
G2


그래프의 종류 : 방향 그래프

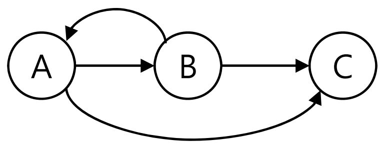
- 방향 그래프(directed graph) , 다이그래프(digraph)
  - 간선이 방향을 가지고 있는 그래프
  - 정점 Vi에서 정점 Vj를 연결하는 간선 즉, Vi→Vj를 <Vi, Vj>로 표현
    - Vi를 꼬리(tail), Vj를 머리(head)라고 한다.
    - <Vi, Vj>와 <Vj, Vi>는 서로 다른 간선을 의미한다.

  - V(G3)={A, B, C, D} E(G3)={<A,B>, <A,D>, <B,C>, <B,D>, <C,D>}
  - V(G4)={A, B, C} E(G4)={<A,B>, <A,C>, <B,A>, <B,C>}

G3

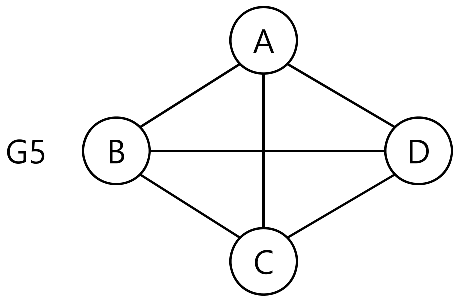


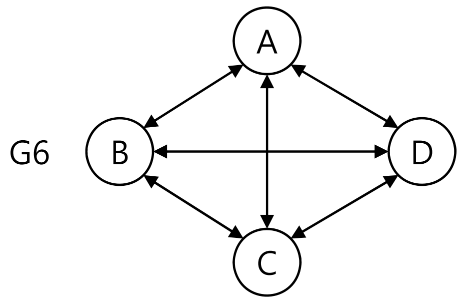
G4



그래프의 종류 : 완전 그래프

- 완전 그래프(complete graph)
  - 각 정점에서 다른 모든 정점을 연결하여 가능한 최대의 간선 수를 가진 그래프
  - 정점이 n개인 무방향 그래프에서 최대의 간선 수 : **n(n-1)/2**개
  - 정점이 n개인 방향 그래프의 최대 간선 수 : **n(n-1)**개
  - 완전 그래프의 예
    - G5는 정점의 개수가 4개인 무방향 그래프이므로 완전 그래프가 되려면 4(4-1)/2=6개의 간선 연결
    - G6은 정점의 개수가 4개인 방향 그래프이므로 완전 그래프가 되려면 4(4-1)=12개의 간선 연결

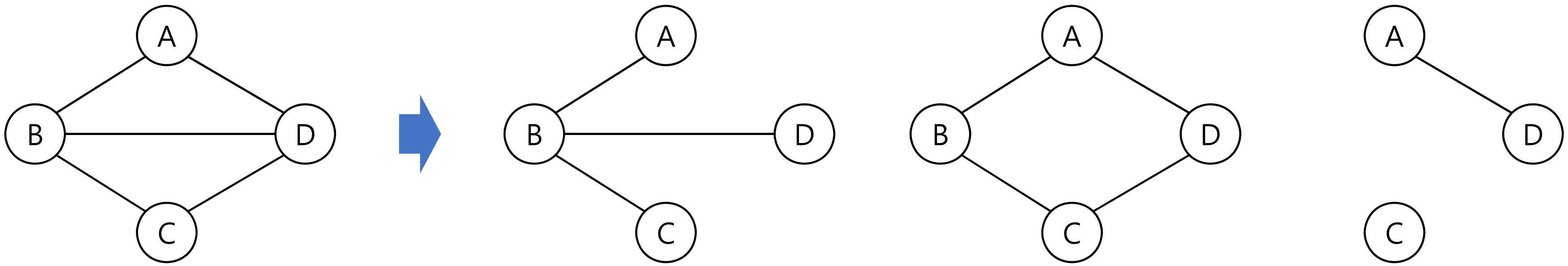




그래프의 종류 : 부분 그래프

- 부분 그래프(subgraph)
  - 원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프
  - 그래프 G와 부분 그래프 G'의 관계
    - V(G')⊆V(G), E(G')⊆E(G)
  - 그래프 G1에 대한 부분 그래프의 예

G1

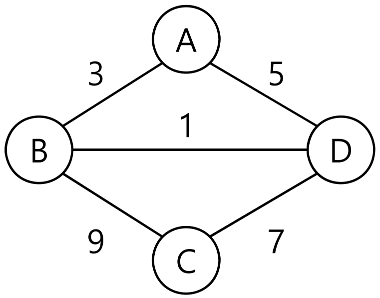


G1'

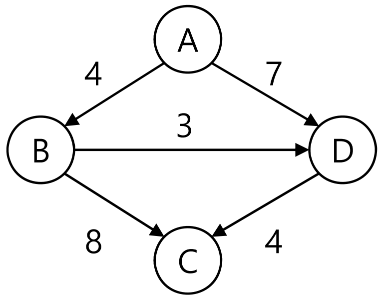
그래프의 종류 : 가중 그래프

- 가중 그래프(weight graph) , 네트워크(network)
  - 정점을 연결하는 간선에 가중치(weight)를 할당한 그래프

G7



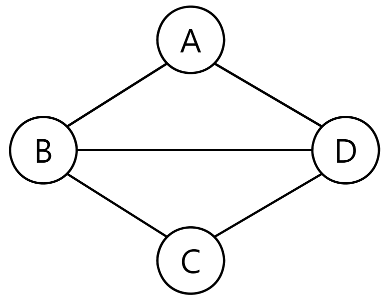
G8



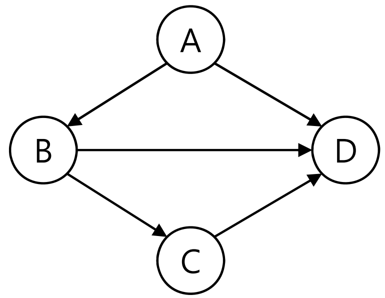
그래프 용어 - I

- 그래프 관련 용어
  - 그래프에서 두 정점 Vi과 Vj를 연결하는 간선 (Vi, Vj)가 있을 때, 두 정점 Vi와 Vj를 인접(Adjacent)되어 있다고 하고, 간선 (Vi, Vj)는 정점 Vi와 Vj에 부속(Incident)되어있다.
    - 그래프G1에서 정점 A와 인접한 정점은 B와 D이고, 정점 A에 부속되어 있는 간선은 (A,B)와 (A,D)
  - 차수(Degree) - 정점에 부속되어있는 간선의 수
    - 무방향 그래프 G1에서 정점 A의 차수는 2, 정점 B의 차수는 3
    - 방향 그래프의 정점의 차수 = 진입차수 + 진출차수
      - 방향 그래프의 **진입차수**(in-degree) : 정점을 머리로 하는 간선의 수
      - 방향 그래프의 **진출차수**(out-degree) : 정점을 꼬리로 하는 간선의 수
      - 방향 그래프 G3에서 정점 B의 진입차수는 1, 진출차수는 2
        정점 B의 전체 차수는 (진입차수 + 진출차수) 이므로 3이 된다.

G1



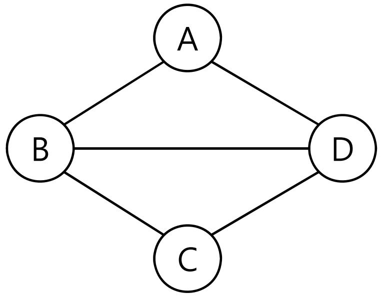
G3



그래프 용어 - II

- **경로(Path)**
  - 그래프에서 간선을 따라 갈 수 있는 길을 순서대로 나열한 것 즉, 정점 Vi에서 Vj까지 간선으로 연결된 정점을 순서대로 나열한 리스트
    - 그래프 G1에서 정점 A에서 정점 C까지는 A-B-C 경로와 A-B-D-C 경로, A-D-C 경로, 그리고 A-D-B-C 경로가 있다.

G1



- **경로길이(Path length)**
  - 경로를 구성하는 간선의 수
- **단순경로(Simple path)**
  - 모두 다른 정점으로 구성된 경로
    - 그래프 G1에서 정점 A에서 정점 C까지의 경로 A-B-C는 단순경로이고, 경로 A-B-D-A-B-C는 단순경로가 아니다.
- **사이클(Cycle)**
  - 단순경로 중에서 경로의 시작 정점과 마지막 정점이 같은 경로
    - 그래프 G1에서 단순경로 A-B-C-D-A와 그래프 G4에서 단순경로 A-B-A는 사이클이 된다.

그래프 용어 - III

- DAG(directed acyclic graph)
  - 방향 그래프이면서 사이클이이 없는 그래프

- 연결 그래프(connected graph)
  - 서로 다른 모든 쌍의 정점들 사이에 경로가 있는 그래프 즉, 떨어져 있는 정점이 없는 그래프
  - 그래프에서 두 정점 Vi에서 Vj까지의 경로가 있으면 정점 Vi와 Vj가 연결(connected) 되었다고 한다.
  - 트리는 사이클이 없는 연결 그래프이다.

그래프의 ADT

- “그래프를 생성 및 초기화 할 때 간선의 방향성 여부를 선택할 수 있고, 가중치의 부여 여부도 선택할 수 있어야 함, 이후 정점과 간선을 삽입하고 삭제할 수 있음”
- Void GraphInit(UALGraph * pg, int nv);
  - 그래프의 초기화, 두 번째 인자로 정점의 수를 전달
- Void GraphDestroy(UALGraph * pg);
  - 그래프 초기화 과정에서 할당한 리소스를 반환
- Void AddEdge(UALGraph * pg, int fromV, int toV);
  - 매개변수 fromV와 toV로 전달된 정점을 연결하는 간선을 그래프에 추가
- Void ShowGraphEdgeInfo(UALGraph * pg);
  - 그래프의 간선정보를 출력

정점의 이름 선언

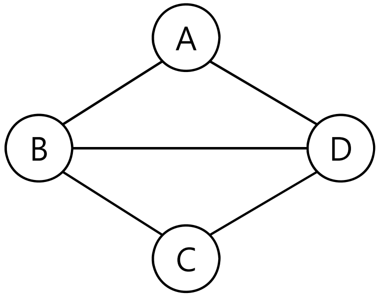
```
enum {A, B, C, D, E, F, G, H, I, J};

enum {SEOUL, INCHEON, DAEGU, BUSAN, KWANGJU};
```

그래프를 표현하는 방법 - 인접행렬

- 인접 행렬(adjacent matrix)
  - 행렬에 대한 2차원 배열을 사용하는 순차 자료구조 방법
  - 그래프의 두 정점을 연결한 간선의 유무를 행렬로 저장
    - n개의 정점을 가진 그래프 : n x n 정방행렬
    - 행렬의 행번호와 열번호 : 그래프의 정점
    - 행렬 값 : 두 정점이 인접되어있으면 1, 인접되어있지 않으면 0
  - 무방향 그래프의 인접 행렬
    - 행 i의 합 = 열 i의 합 = 정점 i의 차수
  - 방향 그래프의 인접 행렬
    - 행 i의 합 = 정점 i의 진출차수
    - 열 i의 합 = 정점 i의 진입차수

인접행렬 예)

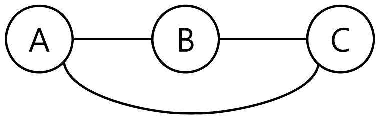
G1



	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	1
D	1	1	1	0

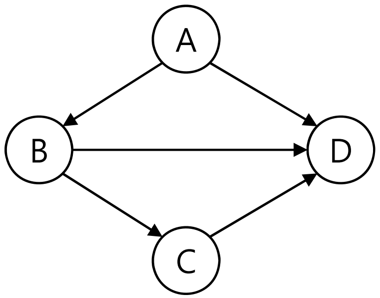
1+0+1+1=3 → 정점 B의 차수

G2



	A	B	C
A	0	1	1
B	1	0	1
C	1	1	0

G3

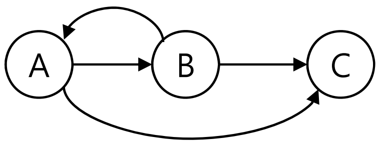


	A	B	C	D
A	0	1	0	1
B	0	0	1	1
C	0	0	0	1
D	0	0	0	0

0+0+1+1=2 → 정점 B의 진출차수

1+0+0+0=2 → 정점 B의 진입차수

G4



	A	B	C
A	0	1	1
B	1	0	1
C	0	0	0

인접 행렬 표현 단점

- n개의 정점을 가지는 그래프를 항상 n x n개의 메모리 사용
- 정점의 개수에 비해서 간선의 개수가 적은 희소 그래프에 대한 인접 행렬은 희소 행렬이 되므로 메모리의 낭비 발생

그래프를 표현하는 방법 - 인접리스트

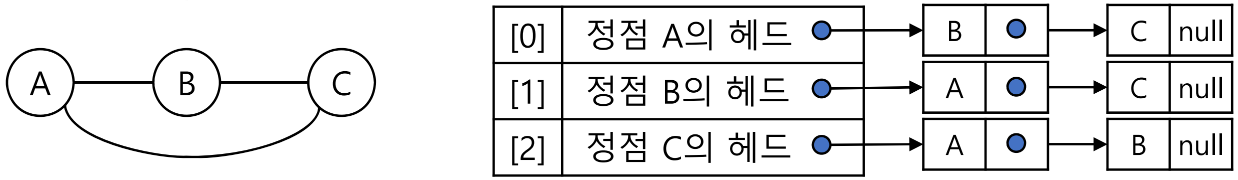
- **인접 리스트(adjacent list)**
  - 각 정점에 대한 인접 정점들을 연결하여 만든 단순 연결 리스트
  - 각 정점의 차수만큼 노드를 연결
    - 리스트 내의 노드들은 인접 정점에 대해서 오름차순으로 연결
  - 인접 리스트의 각 노드
    - 정점을 저장하는 필드와 다음 인접 정점을 연결하는 링크 필드로 구성
  - 정점의 헤드 노드
    - 정점에 대한 리스트의 시작을 표현

인접리스트 예)

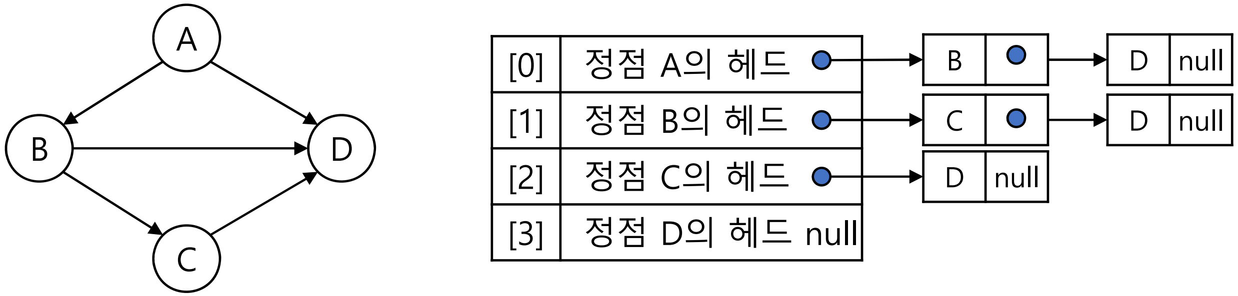
G1

노드 3개 → 정점 B의 차수

G2

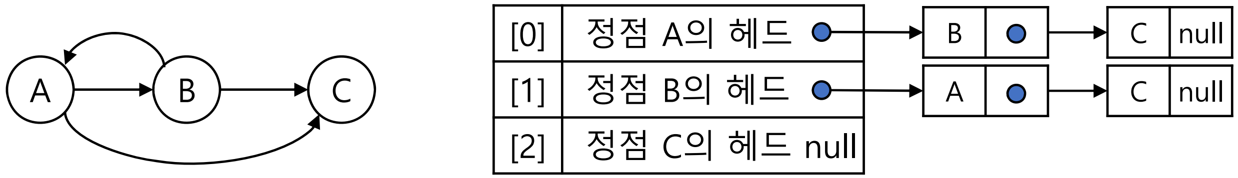


G3



노드 2개 → 정점 B의 진출 차수

G4



그래프 구현 : 헤더파일

```
#include "DLinkedList.h“              // 연결리스트 형식가져옴

enum {A, B, C, D, E, F, G, H, I, J};    //정점의 이름을 선언, 정점 이름 상수화

typedef struct _ual {
        int numV;                  //정점의 수
        int numE;                  //간선의 수
        List * adjList;            //간선의 정보
}ALGraph;

void GraphInit(ALGraph * pg, int nv);
void GraphDestroy(ALGraph * pg);
void AddEdge(ALGraph * pg, int fromV, int toV);
void ShowGraphEdgeInfo(ALGraph * pg);
```

그래프 구현 : main함수

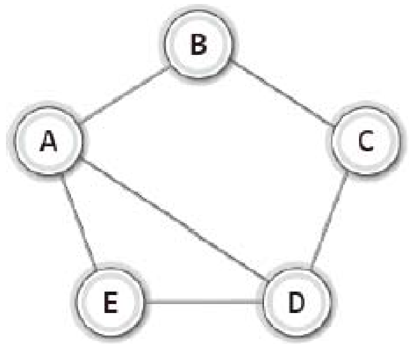
```
#include "DLinkedList.h"            // 연결리스트 형식가져옴

enum {A, B, C, D, E, F, G, H, I, J};    //정점의 이름을 선언, 정점 이름 상수화

typedef struct _ual {
	int numV;               //정점의 수
	int numE;               //간선의 수
	List * adjList;         //간선의 정보
}ALGraph;

void GraphInit(ALGraph * pg, int nv);
void GraphDestroy(ALGraph * pg);
void AddEdge(ALGraph * pg, int fromV, int toV);
void ShowGraphEdgeInfo(ALGraph * pg);
```

그래프 구현 : 헤더파일

```
int main(void) {
        ALGraph graph;                    // 그래프 생성
        GraphInit(&graph, 5);             // 그래프 초기화

        AddEdge(&graph, A, B);            // 정점 A와 B를 연결
        AddEdge(&graph, A, D);
        AddEdge(&graph, B, C);
        AddEdge(&graph, C, D);
        AddEdge(&graph, D, E);
        AddEdge(&graph, E, A);

        ShowGraphEdgeInfo(&graph);        //그래프의 간선정보 출력
        GraphDestroy(&graph);             //그래프의 리소스 소멸

return 0;
}
```

A와 연결된 정점 : B D E
B와 연결된 정점 : A C
C와 연결된 정점 : B D
D와 연결된 정점 : A C E
E와 연결된 정점 : A D

ALGraph.h
ALGraph.c
ALGraphMain.c
DLinkedList.h
DLinkedList.c



그래프 구현 : 초기화와 소멸

```
void GraphInit(ALGraph * pg, int nv) {
	int i;

	//정점의 수에 해당하는 길이의 리스트 배열을 생성
	//간선정보를 저장할 리스트 생성
	pg->adjList = (List*)malloc(sizeof(List)*nv);

	pg->numV = nv;	//정점의 수는 nv에 저장된 값으로 결정
	pg->numE = 0;	//초기의 간선 수는 0개

	// 정점의 수만큼 생성된 리스트들을 초기화한다.
	for (i = 0; i < nv; i++) {
		ListInit(&(pg->adjList[i]));
		SetSortRule(&(pg->adjList[i]), WhoIsPrecede);
	}
}
```

```
void GraphDestroy(ALGraph * pg) {
	if (pg->adjList != NULL)
		free(pg->adjList);
		//동적할당된 연결 리스트 소멸
}
```

그래프 구현 : 간선 추가와 간선정보 출력

```
void AddEdge(ALGraph * pg, int fromV, int toV) {
	//무방향 그래프의 구현을 보여줌
	LInsert(&(pg->adjList[fromV]), toV);
	LInsert(&(pg->adjList[toV]), fromV);

	pg->numE + 1 = 1;
}
```

```
void ShowGraphEdgeInfo(ALGraph * pg) {
int i;
int vx;

	for (i = 0; i < pg->numV; i++) {
	printf("%c와 연결된 정점 : ", i + 65);

		if (LFirst(&(pg->adjList[i]), &vx)) {
			printf("%c ", vx + 65);
			while (LNext(&(pg->adjList[i]), &vx))
				printf("%c ", vx + 65);
			}
			printf("\n");
	}
}
```