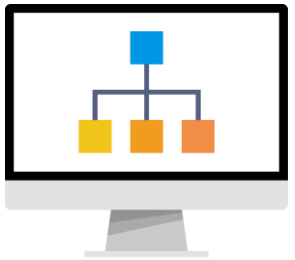


Data Structure & Algorithm
자료구조 및 알고리즘

23. 그래프 (Graph, Part 1)

남은 학기 일정

- 5월 29일(금) – 수업 (오늘)
- 6월 1일(월) – 수업
- 6월 3일(수) – 과제 2 마감
- 6월 5일(금) – 수업
- 6월 8일(월) – 과제 2 풀이
- 6월 12일(금) – 보강 수업
- 6월 15일(월) - **기말고사**
- 6월 18일(목) – 과제 3 마감

과제 3

- 자료구조 및 알고리즘 Cheat sheet 만들기
  - 앞으로의 남은 아미공 생활을 위해

- Cheat sheet: 어떤 주제에 대한 내용을 다음에 참고할 수 있도록 간단하게 요약해 둔 것.

- 자유 양식/자유 주제/자유 분량
  - 예쁘게 만들지 않아도 됨
  - 본인만 알아볼 수 있어도 됨

- 제출 기한: 6월 19일 (금) 23:59

과제 3

- 수업 시간에 배운 내용 또는 추가로 알아본 내용을 자유롭게 포함 (모든 주제를 포함할 필요 없음)

- 시간 및 공간 복잡도, 재귀호출, 배열, (단방향, 양방향, 원형) 연결 리스트, 스택, 큐, 트리, 힙, 해쉬, 그래프, 각 자료구조에서의 초기화/삽입/삭제/탐색, 이진 탐색, 보간 탐색, (선택, 버블, 삽입, 힙, 병합, 퀵, 기수) 정렬, 트리 순회, 그래프 알고리즘 등, 최악의 경우 시간복잡도, 평균 경우 시간복잡도, 구조체, 포인터, 구현 시 주의사항 등…

기말고사

- 6월 15일 월요일 오후 1시에 비대면으로 진행

- 1시에 포털에 기말고사 문제를 공지하고 2시까지 포털의 과제 제출 란에 제출
  - 한글 및 워드 파일로 공지할 예정
  - 따라서, 해당 시간에 한글 또는 MS 워드가 설치된 컴퓨터 환경에 있어야 합니다.
  - 5분 전 제출 확인 권장
  - 혹시 일정/환경상 문제가 있는 수강생은 메일로 미리 연락주세요.

- 과제 3 cheat sheet 사용 가능/인터넷 검색 가능/수강생간 상의는 **불가** (적발 시 학칙에 따름)

21강 보고서 돌아보기

- 보간 탐색에서 최악의 경우 시간복잡도는 얼마일까? 또, 어떤 형태의 입력데이터가 주어졌을 때 최악의 경우가 될까?

- 보간 탐색은 탐색 범위의 가장 왼쪽 값과 오른쪽 값을 가지고 탐색할 값의 인덱스를 추정한다.
  - 만약 추정이 계속 빗나간다면?

- 아래 배열에서 7을 찾아보자. O(N)

index	0	1	2	3	4	5	6	7
key	1	2	3	4	5	6	7	10,000

21강 보고서 돌아보기

- 이분 탐색은 항상 배열의 중간에 있는 값과 비교한다.
- 보간 탐색은 양 끝 값을 활용하여 조금 더 좋을 것 같은 위치를 찾는다.
  - 그러나, 이런 선행 지식을 기반으로 한 개선이 항상 좋은 것은 아니다.

	평균 시간복잡도	최악의 경우 시간 복잡도
이분 탐색	O(log N)	O(log N)
보간 탐색	O(log log N)	O(N)

22강 보고서 돌아보기

- 이진 탐색 트리에서 데이터가 어떤 순서로 삽입이 되는 경우 최악의 시간복잡도를 지닐까? 또, 이 때 삽입/삭제/탐색의 시간복잡도는 얼마일까?

- 데이터가 “정렬 된 순서”로 삽입이 될 때 최악의 시간복잡도를 지니고 이 때에는 O(N)이다.

- 4 2 1 3 6 5 7
- 1 2 3 4 5 6 7
- Self-balancing Binary Search Tree의 필요성!

이중 포인터 이해하기

- 변수의 값을 다른 함수 내에서 바꾸기 위해서는 변수에 대한 포인터를 사용한다.

```c
#include <stdio.h>

void set3(int b) {
    b = 3;
}

int main() {
    int a = 1;
    set3(a);

    printf("%d\n", a);

    return 0;
}
```

```c
#include <stdio.h>

void set3(int *b) {
    *b = 3;
}

int main() {
    int a = 1;
    set3(&a);

    printf("%d\n", a);

    return 0;
}
```

이중 포인터 이해하기

- **포인터** 변수의 값을 함수 내에서 바꾸기 위해서는 **포인터** 변수에 대한 포인터를 사용한다 (= 이중 포인터).

```c
#include <stdio.h>

int c = 3;

void set_global(int** b) {
    *b = &c;
}

int main() {
    int a = 1;
    int* p = &a;

    printf("%d\n", *p);
    set_global(&p);
    printf("%d\n", *p);

    return 0;
}
```

이중 포인터 이해하기

```c
#include <stdio.h>

void set_global(int **b) {
    // b의 주소는? 모른다. 컴파일러가 정해 줌
    printf("&b = %p\n", &b);

    // b가 가리키는 주소는? = b = (p의 주소) = &p
    printf("b = %p\n", b);

    // *b가 가리키는 주소는? = (p가 가리키는 주소) = p = &a
    printf("*b = %p\n", *b);

    // **b = *p = a = 1
    printf("**b = %d\n", **b);

    // b에 무엇을 대입하면? 외부 변수는 변하지 않음
    // *b에 무엇을 대입하면? 외부 p의 값이 바뀜
    // **b에 무엇을 대입하면? 외부 a의 값이 바뀜
}
```

```c
int main() {
    int a = 1;
    int *p = &a;

    printf("a = %d\n", a);
    printf("&a = %p\n", &a);
    printf("p = %p\n", p);
    printf("&p = %p\n", &p);

    set_global(&p);

    return 0;
}
```

메모리 주소 값

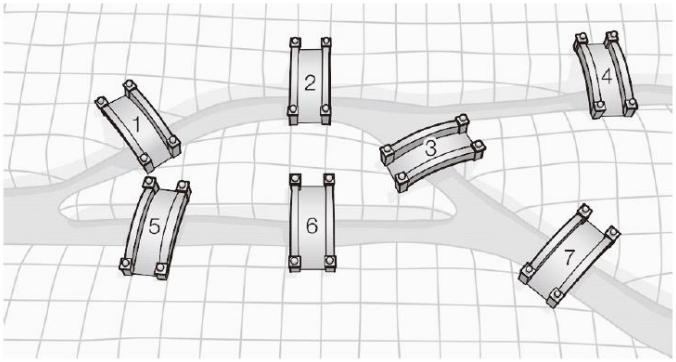
a			p			b		
104	108	112	116	120	124	128	132	136
1			104			116		
*p **b			*b &a			&p		

32비트 머신으로 가정

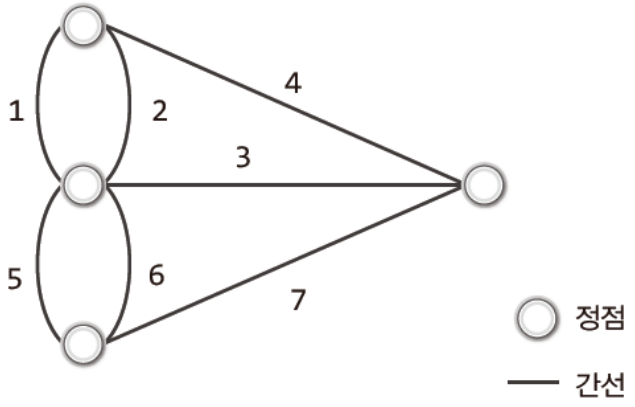
그래프의 이해와 종류

그래프의 역사와 이야깃거리

모든 다리를 한 번씩만 건너서 처음 출발했던 장소로 돌아올 수 있는가?



쾨니히스베르크의 다리 문제

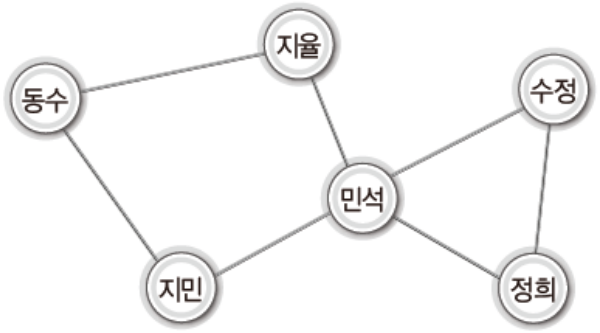


다리 문제의 재 표현

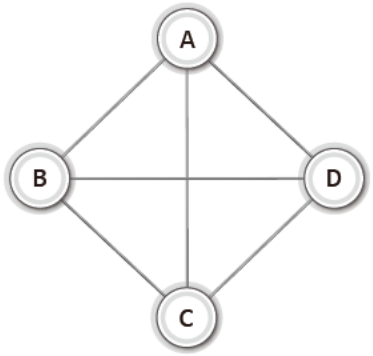
정점 별로 연결된 간선의 수가 모두 짝수 이어야 간선을 한 번씩만 지나서 처음 출발했던 정점으로 돌아올 수 있다.

그래프의 이해와 종류

5학년 3반 어린이들의 비상 연락망: 연락의 방향성이 없다.

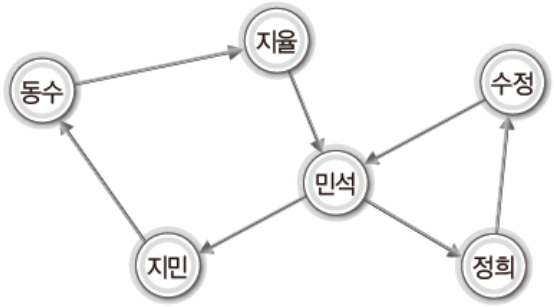


무방향 그래프의 예

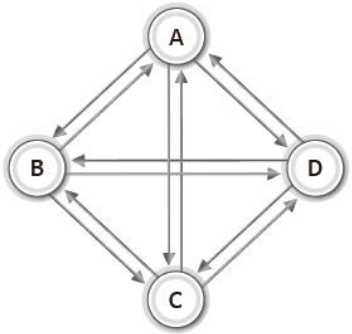


무방향 완전 그래프의 예

5학년 3반 어린이들의 비상 연락망: 방향성이 있다.

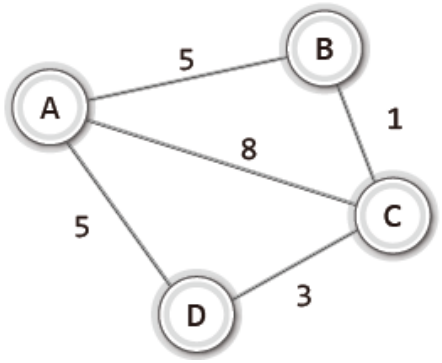


방향 그래프의 예

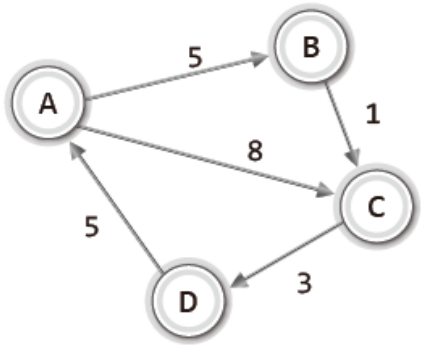


방향 완전 그래프의 예

가중치 그래프와 부분 그래프

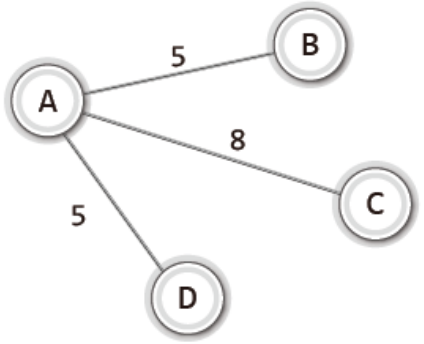


무방향 가중치 그래프의 예

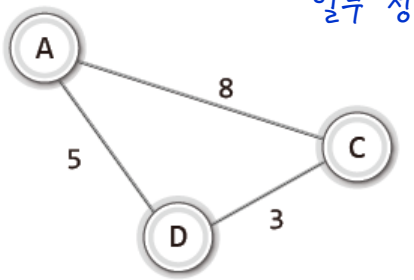


방향 가중치 그래프의 예

부분 그래프



부분 그래프

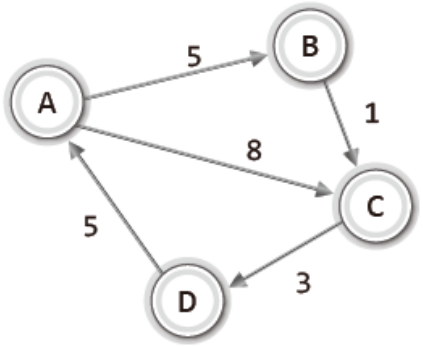


일부 정점과 간선으로 구성이 된 그래프

그래프의 다른 용어

- 차수 (degree): 하나의 정점에 연결된 간선의 개수
  - 진입 차수(in-degree): 정점으로 들어오는 간선의 개수
  - 진출 차수(out-degree): 나가는 간선의 개수

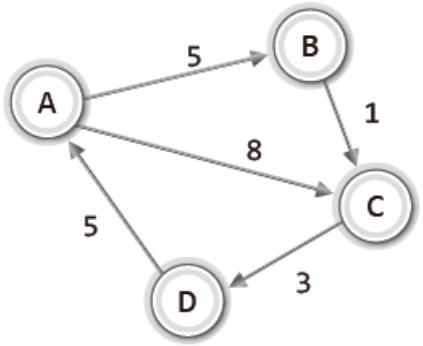
- 사이클(cycle): 한 노드에서 시작해서 같은 노드에서 끝나는 경로
  - Acyclic 그래프: 사이클이 없는 그래프



그래프의 다른 용어

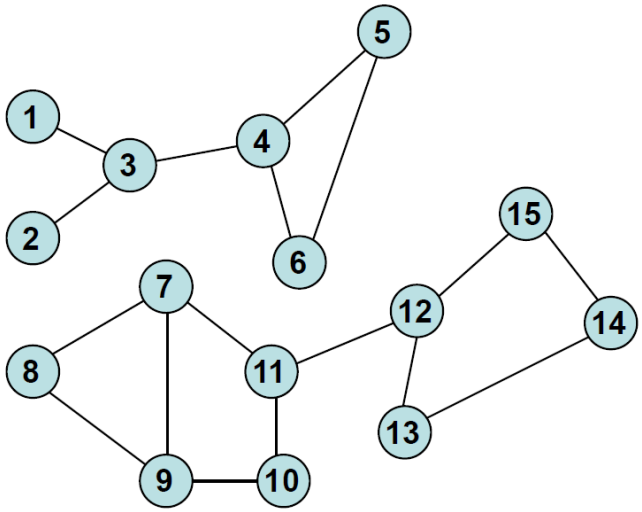
- 차수 (degree): 하나의 정점에 연결된 간선의 개수
  - 진입 차수(in-degree): 정점으로 들어오는 간선의 개수
  - 진출 차수(out-degree): 나가는 간선의 개수

- 사이클(cycle): 한 노드에서 시작해서 같은 노드에서 끝나는 경로
  - Acyclic 그래프: 사이클이 없는 그래프

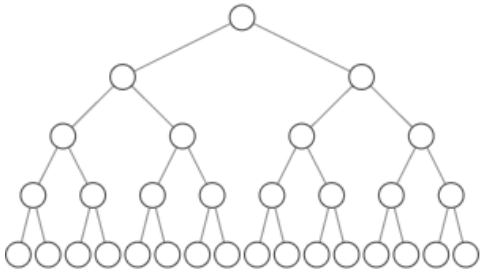


그래프의 다른 용어

- 연결된 그래프(connected): 그래프의 모든 정점에서 다른 모든 정점으로 가는 경로가 존재하는 그래프

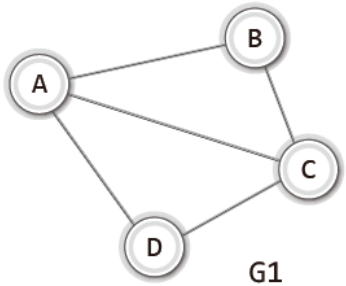


- 트리: connected acyclic undirected graph



그래프의 집합 표현

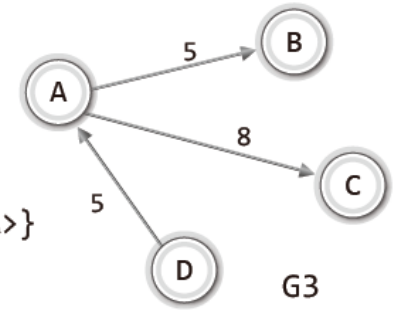
- 그래프 G의 정점 집합 V(G)로 표시함
- 그래프 G의 간선 집합 E(G)로 표시함



G1

V(G1) = {A, B, C, D}

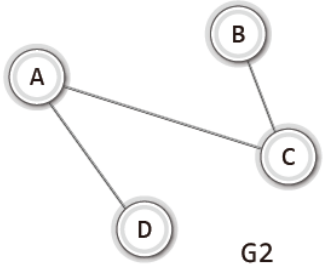
E(G1) = {(A, B), (A, C), (A, D), (B, C), (C, D)}



G3

V(G3) = {A, B, C, D}

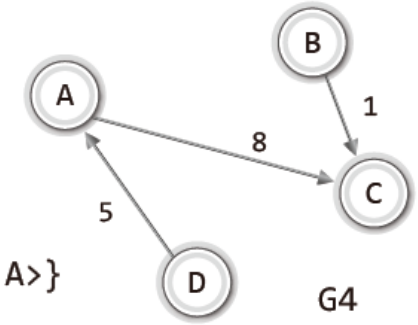
E(G3) = {<A, B>, <A, C>, <D, A>}



G2

V(G2) = {A, B, C, D}

E(G2) = {(A, C), (A, D), (B, C)}

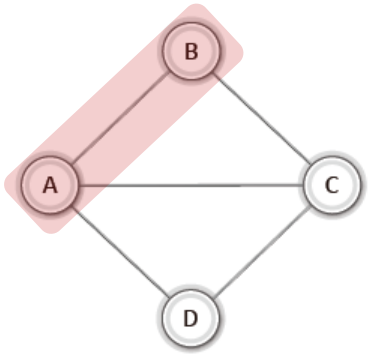


G4

V(G4) = {A, B, C, D}

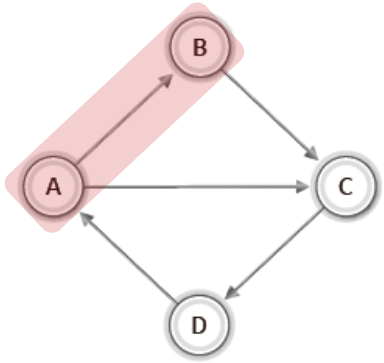
E(G4) = {<A, C>, <B, C>, <D, A>}

그래프를 구현하는 두 가지 방법: 인접 행렬 기반



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

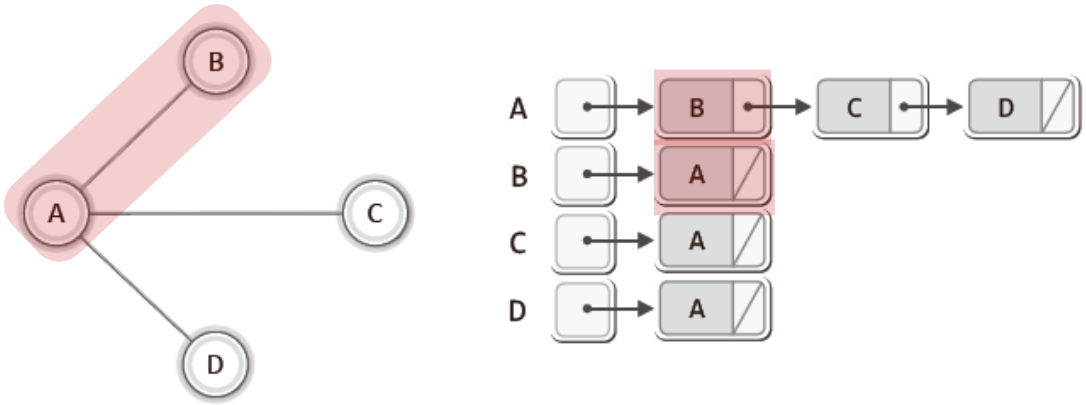
정방 행렬을 이용하는 '인접 행렬 기반 그래프'의 예 1



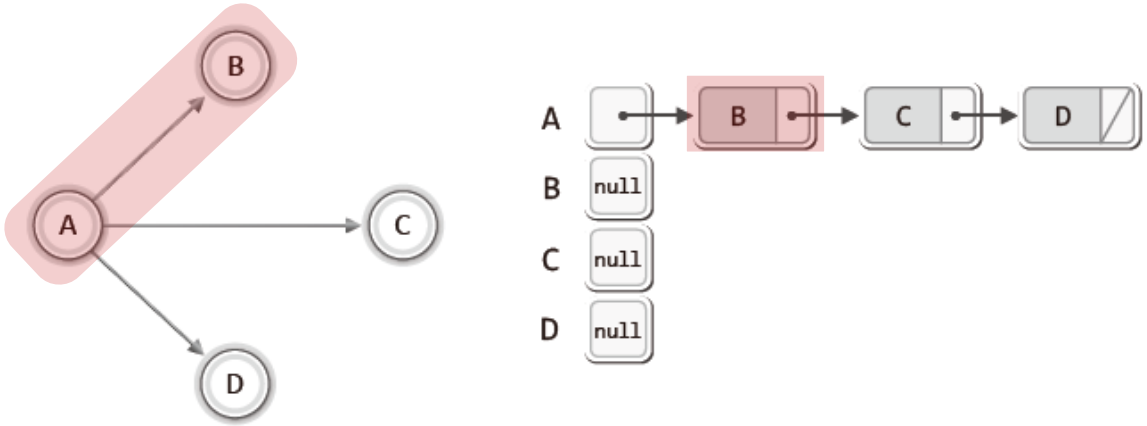
	A	B	C	D
A	0	1	1	0
B	0	0	1	0
C	0	0	0	1
D	1	0	0	0

정방 행렬을 이용하는 '인접 행렬 기반 그래프' 의 예 2

그래프를 구현하는 두 가지 방법: 인접 리스트 기반



연결 리스트를 이용하는 '인접 리스트 기반 그래프'의 예 1



연결 리스트를 이용하는 '인접 리스트 기반 그래프'의 예 2

인접 리스트 기반의 그래프 구현

그래프의 ADT

- `void GraphInit(UALGraph * pg, int nv);`
  - 그래프의 초기화를 진행한다.
  - 두 번째 인자로 정점의 수를 전달한다.

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

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

정점의 이름을 선언하는 방법

- `void GraphDestroy(UALGraph * pg);`
  - 그래프 초기화 과정에서 할당한 리소스를 반환한다.

- `void AddEdge(UALGraph * pg, int fromV, int toV);`
  - 매개변수 fromV와 toV로 전달된 정점을 연결하는 간선을 그래프에 추가한다.

- `void ShowGraphEdgeInfo(UALGraph * pg);`
  - 그래프의 간선정보를 출력한다.

모든 기능과 가능성을 담아서 ADT를 정의하는 것이 능사는 아니다!

특정 그래프를 대상으로 ADT를 제한하여 정의하는 것이 오히려 현명할 수 있다!

그래프의 헤더파일 정의

```
// 연결 리스트를 가져다 쓴다!
#include "DLinkedList.h"
```

앞서 구현한 연결 리스트를 그대로 활용하여 구현하기 위한 선언!

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

정점의 이름을 선언하는 방법!

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

```
// 그래프의 초기화
void GraphInit(ALGraph * pg, int nv);

// 그래프의 리소스 해제
void GraphDestroy(ALGraph * pg);

// 간선의 추가
void AddEdge(ALGraph * pg, int fromV, int toV);

// 간선의 정보 출력
void ShowGraphEdgeInfo(ALGraph * pg);
```

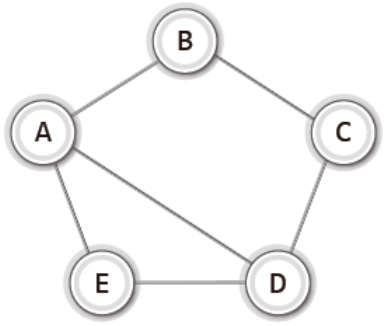
선언된 함수의 이해를 돕기 위한 main 함수

```
int main(void)
{
    ALGraph graph;                  // 그래프의 생성
    GraphInit(&graph, 5);           // 그래프의 초기화
                    초기화 과정에서 정점의 수를 결정한다.
    AddEdge(&graph, A, B);              // 정점 A와 B를 연결
    AddEdge(&graph, A, D);              // 정점 A와 D를 연결
    AddEdge(&graph, B, C);              // 정점 B와 C를 연결
    AddEdge(&graph, C, D);              // 정점 C와 D를 연결
    AddEdge(&graph, D, E);              // 정점 D와 E를 연결
    AddEdge(&graph, E, A);              // 정점 E와 A를 연결

    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
```

실행결과

ALGraph.h
ALGraph.c
ALGraphMain.c
DLinkedList.h
DLinkedList.c **파일구성**



main 함수를 통해서 생성한 그래프

그래프의 구현: 초기화와 소멸

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

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

    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);
    }
}
```

```
int WhoIsPrecede(int data1, int data2)
{
    if(data1 < data2)
        return 0;
    else
        return 1;
}
```

그래프와 연관성 없다! 다만 연결 리스트가 요구하므로 적당한 함수를 등록하였다.

```
void GraphDestroy(ALGraph * pg)      // 그래프 리소스의 해제
{
    if(pg->adjList != NULL)
        free(pg->adjList);       // 동적으로 할당된 연결 리스트의 소멸
}
```

그래프의 구현: 간선의 추가와 간선 정보 출력

```
// 간선의 추가
void AddEdge(ALGraph * pg, int fromV, int toV)
{
    LInsert(&(pg->adjList[fromV]), toV);
    LInsert(&(pg->adjList[toV]), fromV);

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

무방향 그래프의 구현을 보여준다.

방향 그래프의 구현이라면 LInsert의 함수 호출이 1회로 끝이 난다.

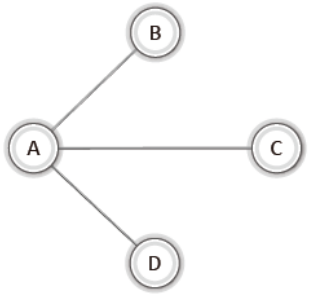
```
// 간선의 정보 출력
void ShowGraphEdgeInfo(ALGraph * pg)
{
    int i;
    int vx;

    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");
    }
}
```

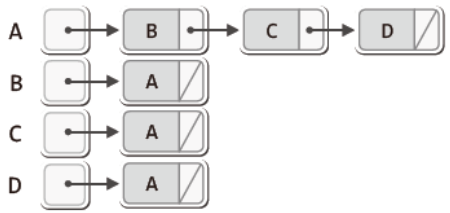
요약

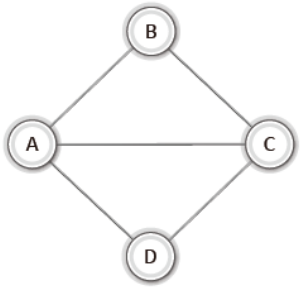
- 그래프(graph)는 정점(vertex)와 간선(edge)로 이루어진 자료 구조

- 트리도 그래프, 단 사이클이 없는 그래프 (acyclic graph)

- 그래프를 나타낼 때: 인접 리스트와 인접 행렬







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

출석 인정을 위한 보고서 제출

- 다음 질문에 대한 답을 PDF로 포털에 제출

- 그래프의 정점 개수를 V, 간선 개수를 E라고 하자. 다음을 빅-오 표기법으로 나타내어라.

- 인접 행렬을 사용하여 그래프를 나타낼 때 공간복잡도
- 인접 행렬을 사용했을 때, 두 노드 간 간선 존재 유무를 알아 낼 때 시간복잡도
- 인접 리스트를 사용하여 그래프를 나타낼 때 공간복잡도
- 인접 리스트를 사용했을 때, 두 노드 간 간선 존재 유무를 알아 낼 때 시간복잡도