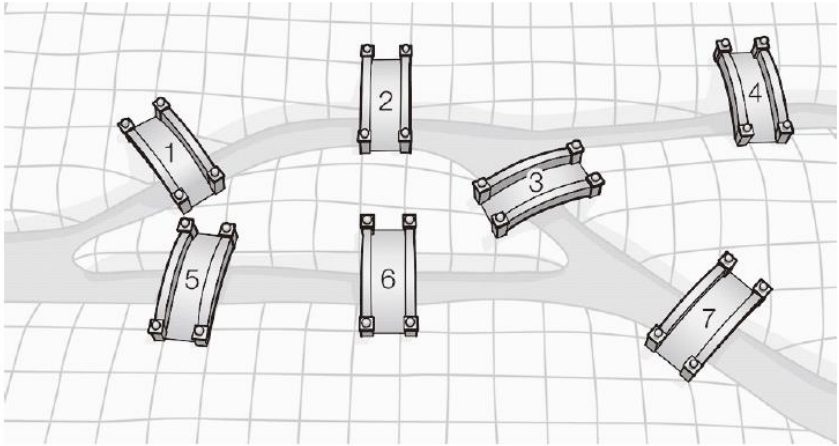
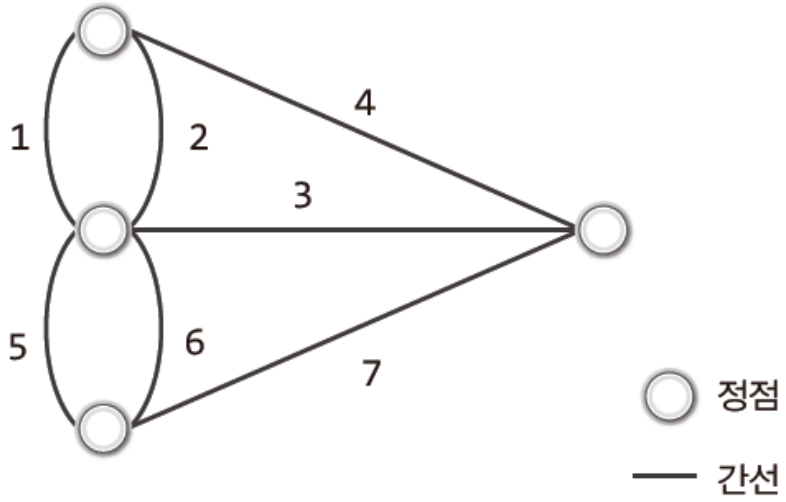
# Chapter 14.
# 그래프 (Graph)

11주차

# 그래프의 역사와 이야깃거리

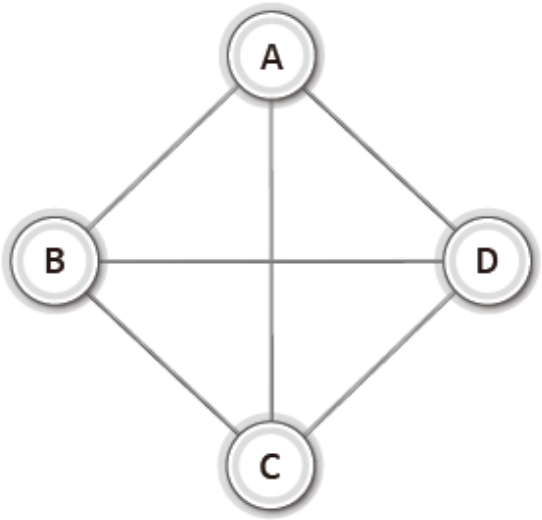
- 그래프 알고리즘의 창시자 : 오일러(Euler)
- 쾨니히스베르크의 다리 문제
  - 모든 다리를 한 번씩만 건너서 처음 출발했던 장소로 돌아올 수 있는가?
  - 도식화하여 해결 시도 : 간선=다리, 정점=강으로 구분되는 땅.
  - [답] 정점 별로 연결된 간선의 수가 모두 짝수 이어야 간선을 한 번씩만 지나서 처음 출발했던 정점으로 돌아올 수 있다. 그러므로 불가능.



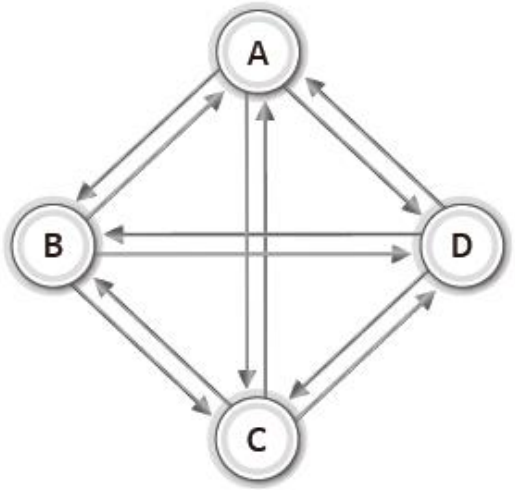



# 그래프의 이해와 종류

- 정점(Vertex) : 연결의 대상이 되는 개체
- 간선(Edge) : 정점 간의 연결 선
- 무방향 그래프 : 간선에 방향이 없는 그래프
- 방향 그래프 : 간선에 방향이 있는 그래프
- 완전 그래프 : 각각의 정점에서 다른 모든 정점으로 연결된 그래프



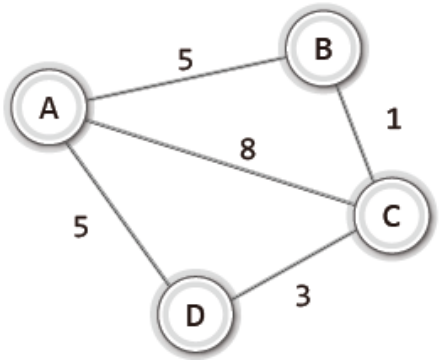

무방향 완전 그래프의 예



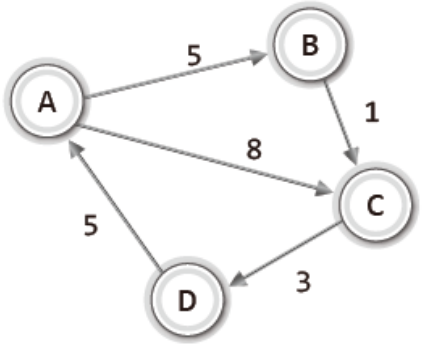

방향 완전 그래프의 예

# 가중치 그래프와 부분 그래프

- 가중치 그래프 : 간선에 가중치가 있는 그래프
  - 가중치 :두 정점 사이의 거리, 이동시간 등의 정보
- 부분 그래프 : 원 그래프의 일부의 정점과 간선으로 이루어진 그래프

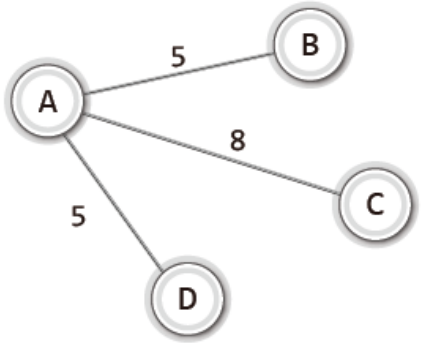



방향 가중치 그래프 ->

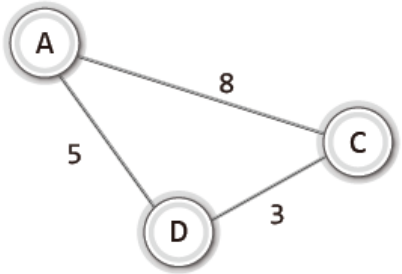



<- 무방향 가중치 그래프

부분 그래프

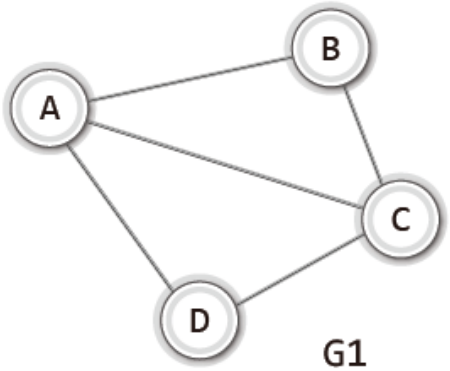



부분 그래프



# 그래프의 집합 표현

- V(G) : 그래프 G의 정점 집합
- E(G) : 그래프 G의 간선 집합
- (A,B) : 무방향 그래프에서 정점 A와 B를 연결하는 간선
- <A,B> : 방향 그래프에서 정점 A에서 B로 연결된 간선



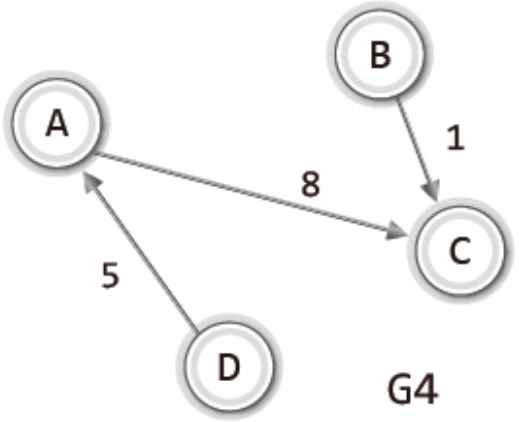

G1

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

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

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

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




G4

# 그래프의 ADT

```
// 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);

// 유틸리티 함수: 간선의 정보 출력
void ShowGraphEdgeInfo(ALGraph * pg);

// 정점들을 enumeration으로 선언.
enum {A, B, C, D, E, F, G, H, I, J};
```

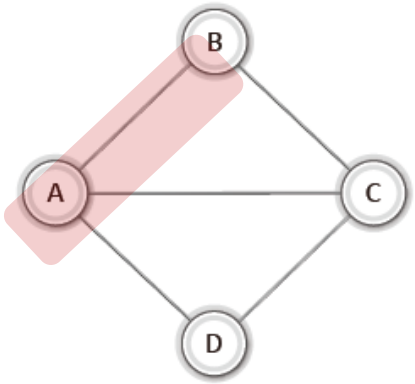
정점의 개수

간선은 추가만 가능.
삭제는 불가.

539-540p

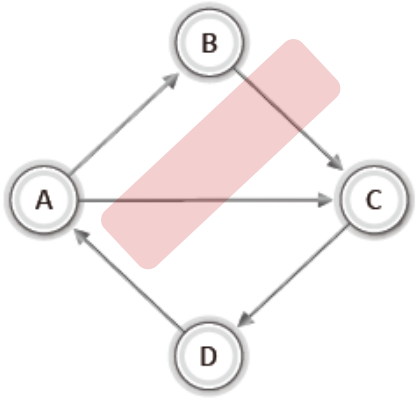
# 그래프를 구현하는 두 가지 방법
## (인접 행렬 기반)

- 정방행렬을 사용
- 각 행은 시작 정점, 각 열은 도착 정점으로 간주하고 간선이 있는 칸에는 1 없는 칸에는 0.




| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 |
| B | 1 | 0 | 1 | 0 |
| C | 1 | 1 | 0 | 1 |
| D | 1 | 0 | 1 | 0 |

무방향 그래프의 인접 행렬 표현



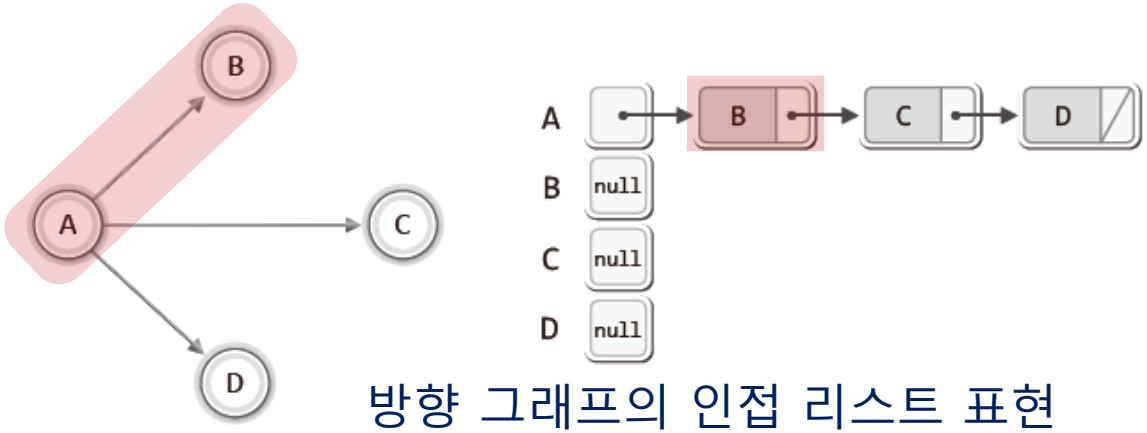

| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 0 |
| C | 0 | 0 | 0 | 1 |
| D | 1 | 0 | 0 | 0 |

방향 그래프의 인접 행렬 표현

# 그래프를 구현하는 두 가지 방법
## (인접 리스트 기반)

- 각 정점은 자신과 연결된 정점들을 담은 리스트를 가짐.
- 위 리스트의 배열이 그래프가 됨.


무방향 그래프의 인접 리스트 표현




방향 그래프의 인접 리스트 표현

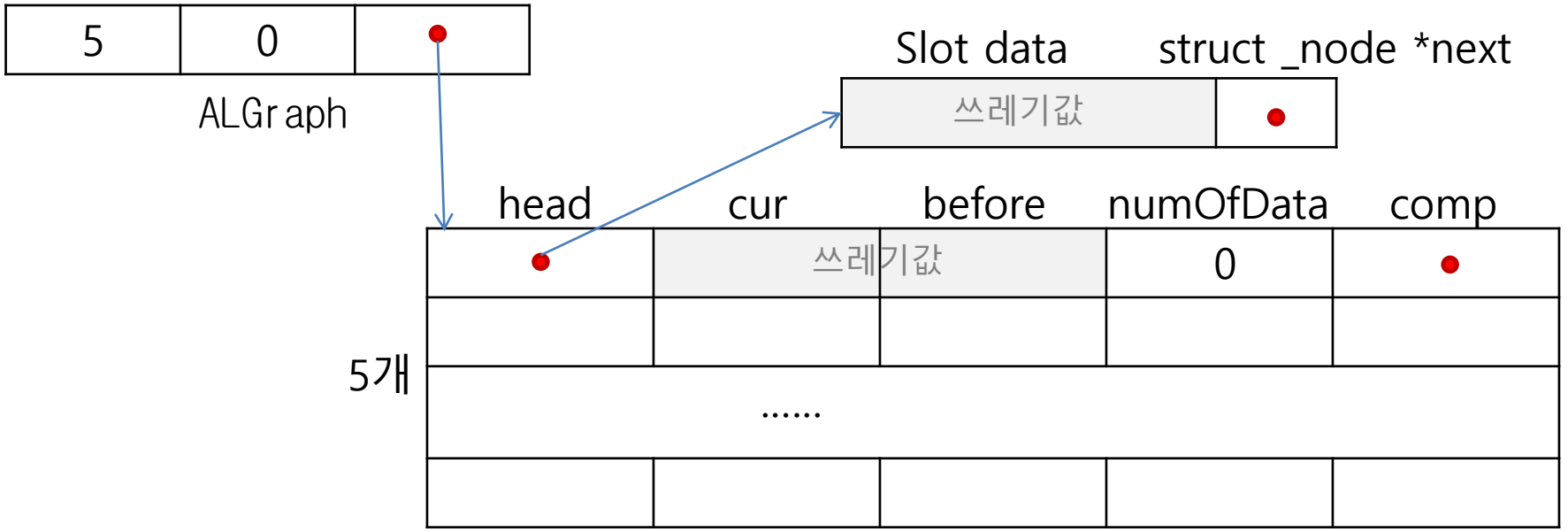
# 인접 리스트 기반의 무방향그래프 구현
## (연결리스트 선언)

```
#include <stdio.h>
#include <stdlib.h>
#define TRUE  1
#define FALSE 0
typedef int LData;
typedef struct _node {
        LData data;
        struct _node* next;
} Node;
typedef struct _linkedList {
        Node* head;
        Node* cur;
        Node* before;
        int numOfData;
        int (*comp)(LData d1, LData d2);
} LinkedList;
typedef LinkedList List;
```

# 인접 리스트 기반의 무방향그래프 구현
## (그래프 선언)

```
// 정점의 이름들을 상수화
enum { A, B, C, D, E, F, G, H, I, J };
typedef struct _ual
{
        int numV;    // 정점의 수
        int numE;    // 간선의 수
        List* adjList;    // 간선의 정보
} ALGraph;
```

0, 1, 2,... 대신
A, B, C,... 사용할 예정.



# 인접 리스트 기반의 무방향그래프 구현
## (연결리스트 구현 1/5)

```
void ListInit(List* plist) {
        plist->head = (Node*)malloc(sizeof(Node));
        plist->head->next = NULL;
        plist->comp = NULL;
        plist->numOfData = 0;
}

void FInsert(List* plist, LData data) {
        Node* newNode = (Node*)malloc(sizeof(Node));
        newNode->data = data;

        newNode->next = plist->head->next;
        plist->head->next = newNode;

        (plist->numOfData)++;
}
```

# 인접 리스트 기반의 무방향그래프 구현
## (연결리스트 구현 2/5)

```
void SInsert(List* plist, LData data)
{
	Node* newNode = (Node*)malloc(sizeof(Node));
	Node* pred = plist->head;
	newNode->data = data;

	while (pred->next != NULL &&
		plist->comp(data, pred->next->data) != 0)
	{
		pred = pred->next;
	}

	newNode->next = pred->next;
	pred->next = newNode;

	(plist->numOfData)++;
}
```

comp 함수에서 0 이외 값을 반환하면 계속 뒤쪽으로 이동.

# 인접 리스트 기반의 무방향그래프 구현
## (연결리스트 구현 3/5)

```
void LInsert(List* plist, LData data) {
	if (plist->comp == NULL)
		FInsert(plist, data);
	else
		SInsert(plist, data);
}

int LFirst(List* plist, LData* pdata) {
	if (plist->head->next == NULL)
		return FALSE;

	plist->before = plist->head;
	plist->cur = plist->head->next;

	*pdata = plist->cur->data;
	return TRUE;
}
```

# 인접 리스트 기반의 무방향그래프 구현
## (연결리스트 구현 4/5)

```
int LNext(List* plist, LData* pdata)
{
	if (plist->cur->next == NULL)
		return FALSE;

	plist->before = plist->cur;
	plist->cur = plist->cur->next;

	*pdata = plist->cur->data;
	return TRUE;
}
```

# 인접 리스트 기반의 무방향그래프 구현
## (연결리스트 구현 5/5)

```
LData LRemove(List* plist) {
	Node* rpos = plist->cur;
	LData rdata = rpos->data;

	plist->before->next = plist->cur->next;
	plist->cur = plist->before;

	free(rpos);
	(plist->numOfData)--;
	return rdata;
}
int LCount(List* plist) {
	return plist->numOfData;
}
void SetSortRule(List* plist, int (*comp)(LData d1, LData d2)) {
	plist->comp = comp;
}
```

# 인접 리스트 기반의 무방향그래프 구현
## (연결리스트 구현 : 함수 추가)

```
void ListAllRemove(List* plist) {
        LData data;
        int OK = LFirst(plist, &data);
        while (OK) {
                data = LRemove(plist);
                OK = LNext(plist, &data);
        }
}
```

현재 노드(cur)가 있다.
즉, Lremove 가능하다!

하나의 연결리스트에 할당된 모든 노드 (첫번째에 있는 dummy node 포함)들을 free.

546-548p

# 인접 리스트 기반의 무방향그래프 구현
## (그래프 구현 1/3)

```
int WhoIsPrecede(int data1, int data2) {
        if (data1 < data2)
                return 0;
        else
                return 1;
}
void GraphInit(ALGraph* pg, int nv) {// 그래프의 초기화
        int i;

        pg->adjList = (List*)malloc(sizeof(List) * nv);
        pg->numV = nv;
        pg->numE = 0;      // 초기의 간선 수는 0개

        for (i = 0; i < nv; i++) {
                ListInit(&(pg->adjList[i]));
                SetSortRule(&(pg->adjList[i]), WhoIsPrecede);
        }
}
```

간선의
정렬 순서 결정

# 인접 리스트 기반의 무방향그래프 구현
## (그래프 구현 2/3)

```
// 그래프 리소스의 해제
void GraphDestroy(ALGraph* pg) {
        if (pg->adjList != NULL) {
                int i = 0;
                for (i = 0;i < pg->numV;i++)
                        ListAllRemove(&pg->adjList[i]);
                free(pg->adjList);
        }
}

// 간선의 추가
void AddEdge(ALGraph* pg, int fromV, int toV)
{
        LInsert(&(pg->adjList[fromV]), toV);
        LInsert(&(pg->adjList[toV]), fromV);
        pg->numE += 1;
}
```

각 연결리스트의 모든 노드를 free 시킨 후 연결리스트의 배열을 free.

무방향그래프이므로 간선노드는 2개 추가하고 간선 개수는1개 증가시킴.

# 인접 리스트 기반의 무방향그래프 구현
## (그래프 구현 3/3)

```
// 유틸리티 함수: 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");
	}
}
```

546-548p

# 인접 리스트 기반의 무방향그래프 구현
## (main 함수 구현)

```
int main(void) {
        ALGraph graph;
        GraphInit(&graph, 5);      // A, B, C, D, E의 정점 생성

        AddEdge(&graph, A, B);
        AddEdge(&graph, A, D);
        AddEdge(&graph, B, C);
        AddEdge(&graph, C, D);
        AddEdge(&graph, D, E);
        AddEdge(&graph, E, A);

        ShowGraphEdgeInfo(&graph);

        GraphDestroy(&graph);
        return 0;
}
```

[실행결과]

```
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
```