

윤성우의 열혈 자료구조

: C언어를 이용한 자료구조 학습서



Chapter 14. 그래프

Introduction To Data Structures Using C

Chapter 14. 그래프



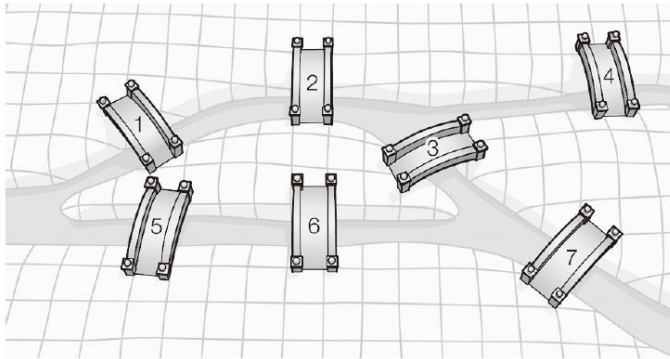
Chapter 14-1:

그래프의 이해와 종류

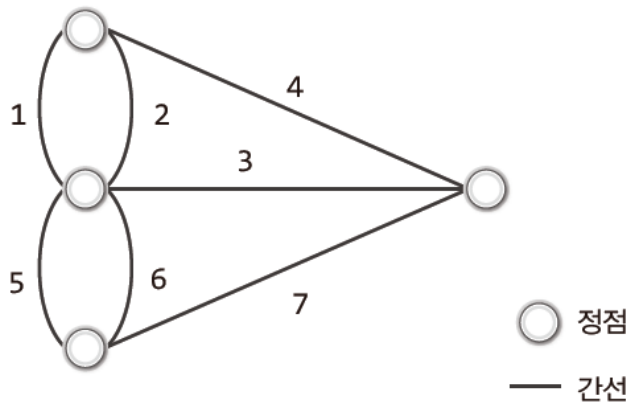


그래프의 역사와 이야깃거리

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



쾨니히스베르크의 다리 문제

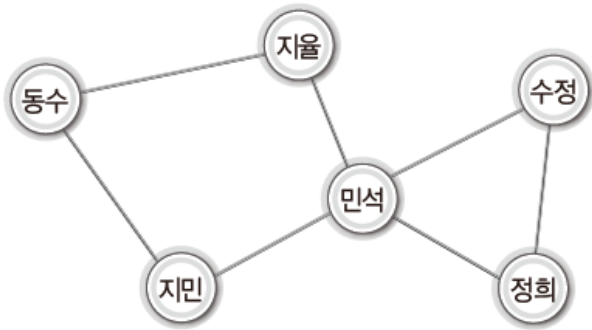


정점 별로 연결된 간선의 수가 모두 짝수 이어야 간선을 한 번씩만 지나서 처음 출발했던 정점으로 돌아올 수 있다.

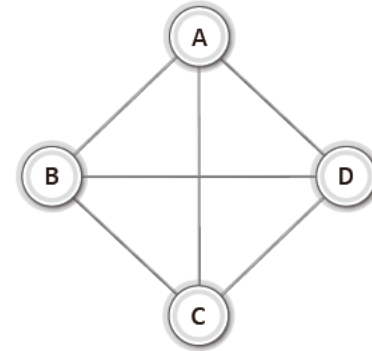
다리 문제의 재 표현

그래프의 이해와 종류

5학년 3반 어린이들의 비상 연락망: 연락의 방향성이 없다.

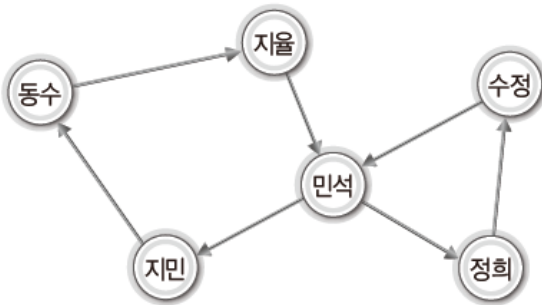


무방향 그래프의 예

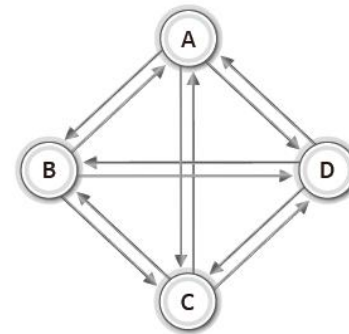


무방향 완전 그래프의 예

5학년 3반 어린이들의 비상 연락망: 방향성이 있다.

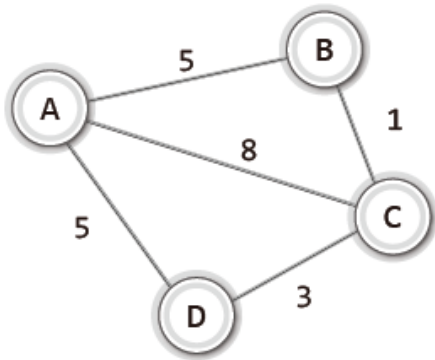


방향 그래프의 예

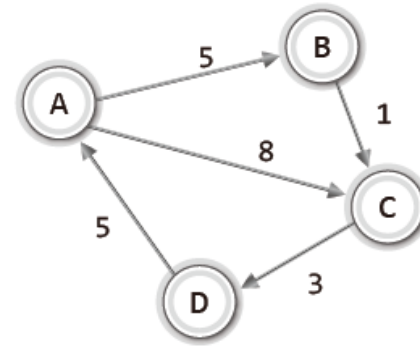


방향 완전 그래프의 예

가중치 그래프와 부분 그래프



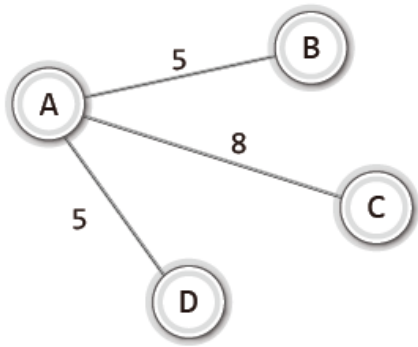
무방향 가중치 그래프의 예



방향 가중치 그래프의 예

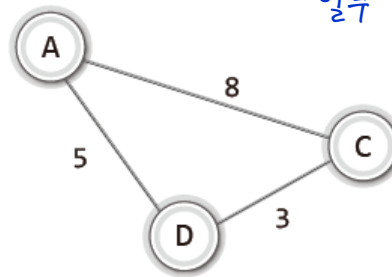


부분 그래프



부분 그래프

일부 정점과 간선으로 구성이 된 그래프



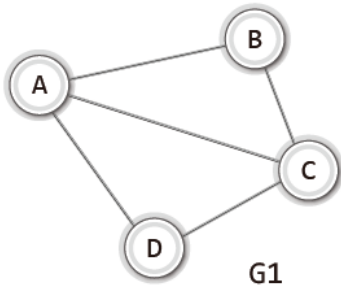
그래프의 집합 표현

• 그래프 G의 정점 집합

$V(G)$ 로 표시함

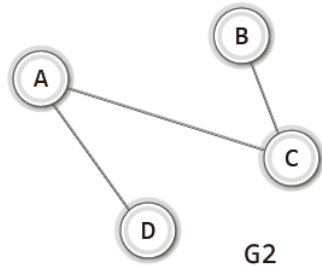
• 그래프 G의 간선 집합

$E(G)$ 로 표시함



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

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

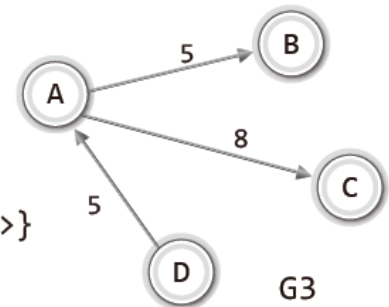


$$V(G2) = \{A, B, C, D\}$$

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

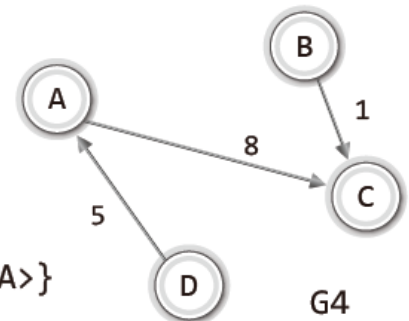
$$V(G3) = \{A, B, C, D\}$$

$$E(G3) = \{\langle A, B \rangle, \langle A, C \rangle, \langle D, A \rangle\}$$



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

$$E(G4) = \{\langle A, C \rangle, \langle B, C \rangle, \langle D, A \rangle\}$$



그래프의 ADT

- `void GraphInit(UALGraph * pg, int nv);`

- 그래프의 초기화를 진행한다.
- 두 번째 인자로 정점의 수를 전달한다.

- `void GraphDestroy(UALGraph * pg);`

- 그래프 초기화 과정에서 할당한 리소스를 반환한다.

- `void AddEdge(UALGraph * pg, int fromV, int toV);`

- 매개변수 `fromV`와 `toV`로 전달된 정점을 연결하는 간선을 그래프에 추가한다.

- `void ShowGraphEdgeInfo(UALGraph * pg);`

- 그래프의 간선정보를 출력한다.

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

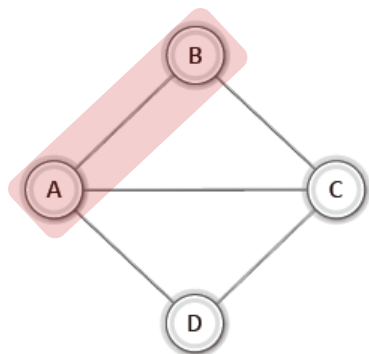
```
enum {SEOUL, INCHEON, DAEGU, BUSAN, KWANGJU};
```

정점의 이름을 선언하는 방법

모든 기능과 가능성을 담아서 ADT를 정의하는 것이 능사는 아니다!

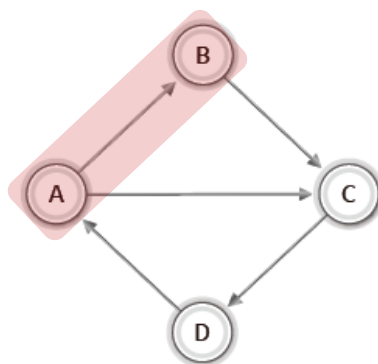
특정 그래프를 대상으로 ADT를 제한하여 정의하는 것이 오히려 현명할 수 있다!

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

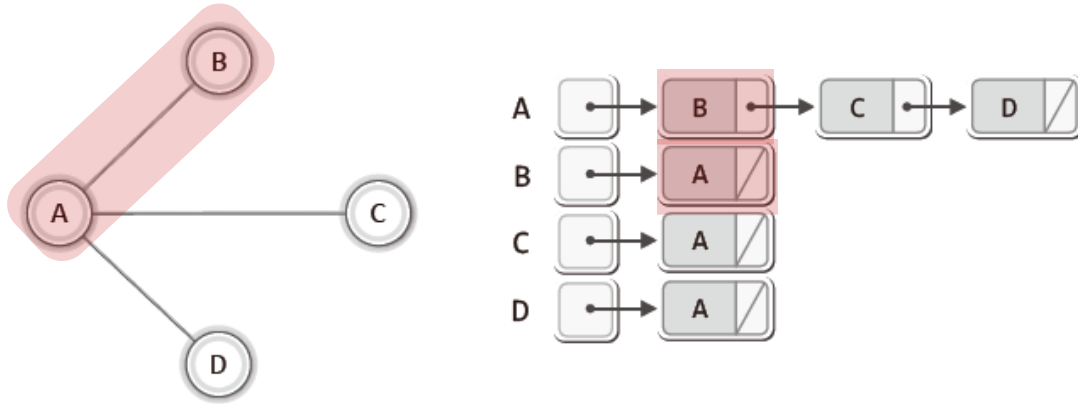
정방 행렬을 이용하는 '인접 행렬 기반 그래프'의 예 1



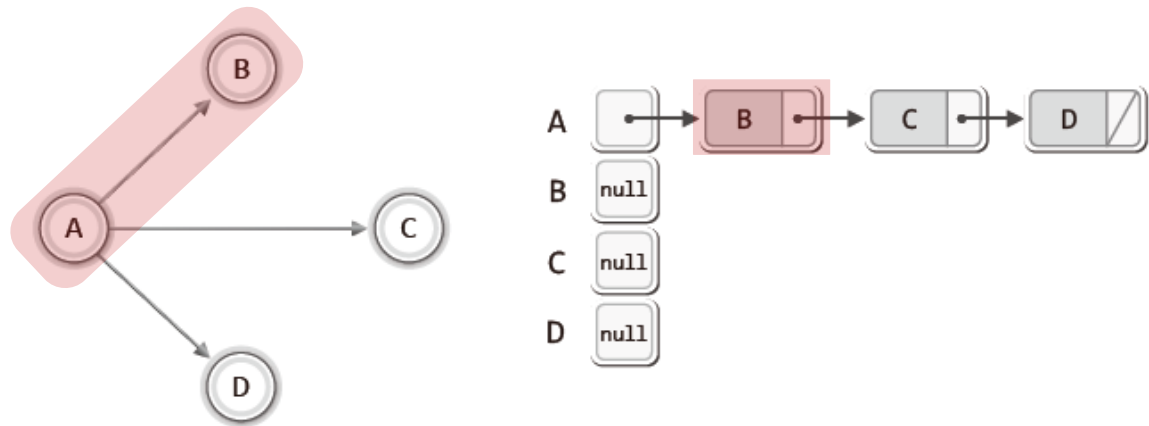
	A	B	C	D
A	0	1	1	0
B	0	0	1	0
C	0	0	0	1
D	1	0	0	0

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

Chapter 14. 그래프



Chapter 14-2:

인접 리스트 기반의 그래프 구현



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

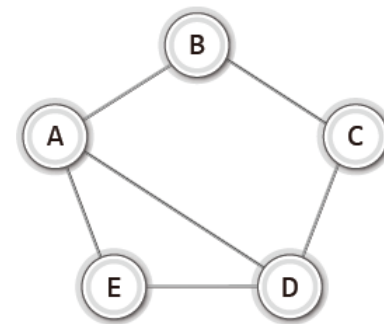
```
int main(void)
{
    ALGraph graph;           // 그래프의 생성
    GraphInit(&graph, 5);    // 그래프의 초기화
                             초기화 과정에서 정점의 수를 결정한다.
    AddEdge(&graph, A, B);   // 정점 A와 B를 연결
    AddEdge(&graph, A, D);   // 정점 A와 D를 연결
    AddEdge(&graph, B, C);   // 정점 B와 C를 연결
    AddEdge(&graph, 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 *LInsert*의 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");
    }
}
```

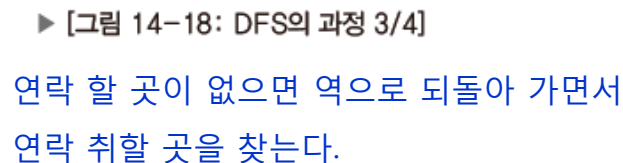
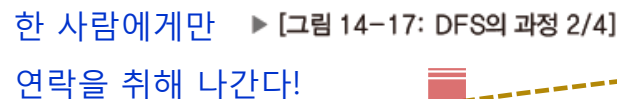
Chapter 14. 그래프



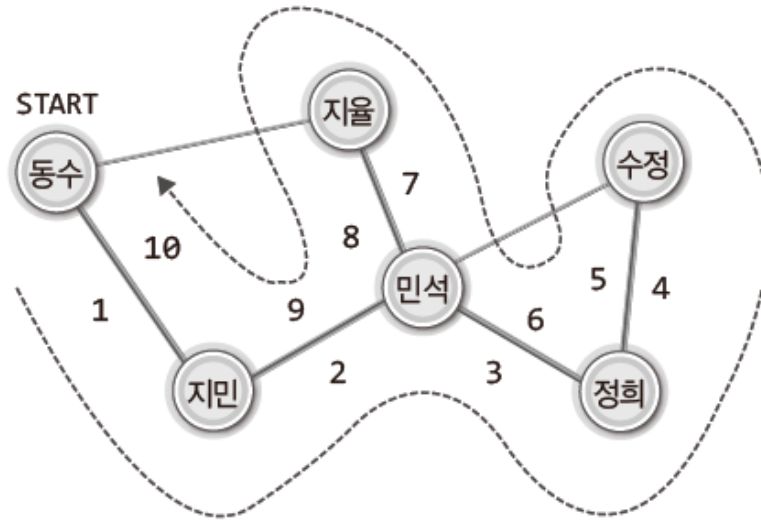
Chapter 14-3:

그래프의 탐색





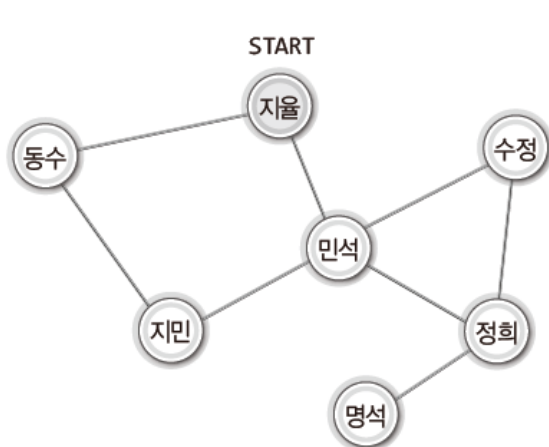
깊이 우선 탐색: 정리



깊이 우선 탐색 과정의 핵심 세 가지

- 한 사람에게만 연락을 한다.
- 연락할 사람이 없으면, 자신에게 연락한 사람에게 이를 알린다.
- 처음 연락을 시작한 사람의 위치에서 연락은 끝이 난다

너비 우선 탐색: Breadth First Search



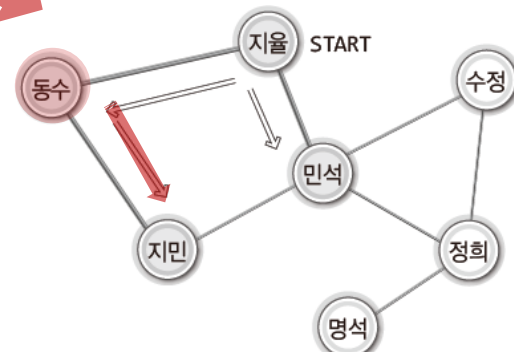
▶ [그림 14-22: BFS의 과정 1/5]



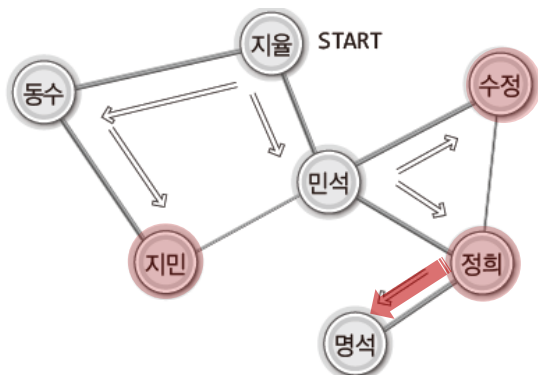
▶ [그림 14-23: BFS의 과정 2/5]



연결된 모든 이에게 연락을!

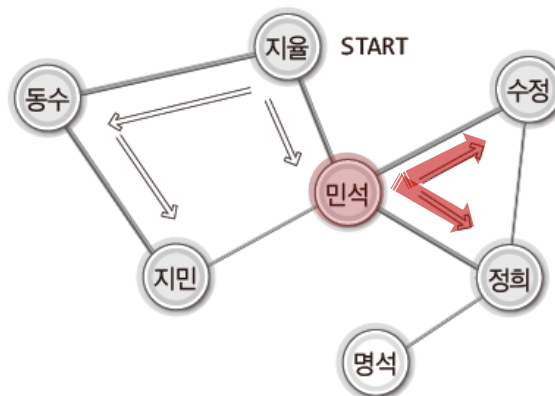


▶ [그림 14-24: BFS의 과정 3/5]



▶ [그림 14-26: BFS의 과정 5/5]

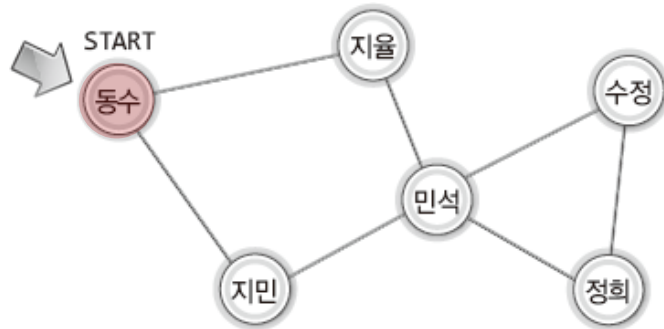
마지막으로 명석 또한 전달의 기회를 갖는다.



▶ [그림 14-25: BFS의 과정 4/5]



깊이 우선 탐색의 구현 모델: 과정 1~



방문 정보의 기록을 목적으로!

동수 지민 민석 정희 수정 지울

경로 정보의 추적을 목적으로!

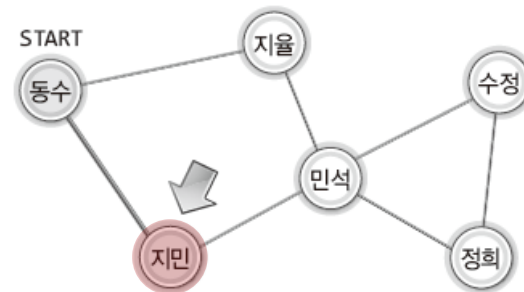


스택!

▶ [그림 14-28: DFS의 구현 1/7]



동수를 떠나 지민에게 연락이 취해질 때
동수의 정보가 스택으로 이동한다!



방문 정보

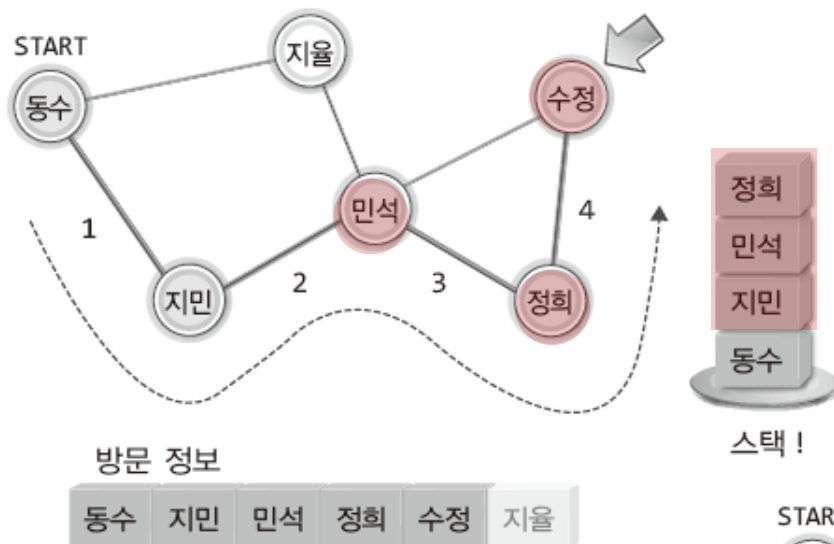
동수 지민 민석 정희 수정 지울



스택!

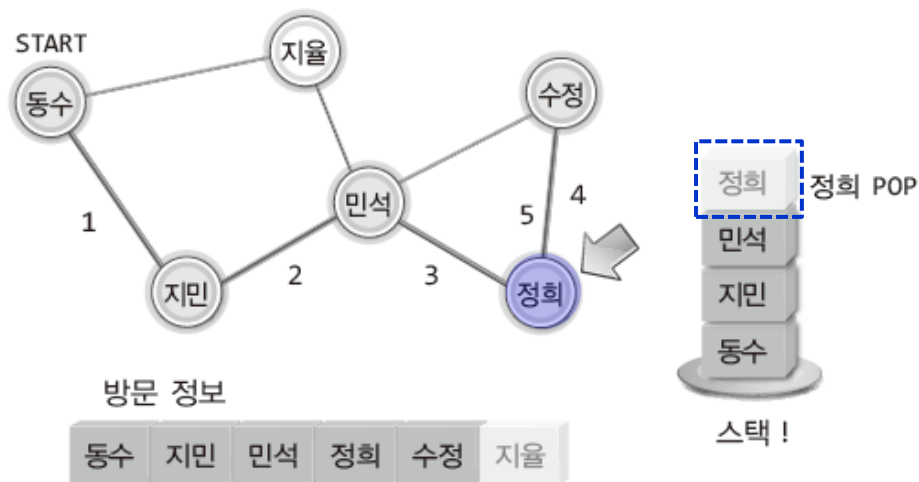
▶ [그림 14-29: DFS의 구현 2/7]

깊이 우선 탐색의 구현 모델: 과정 3~



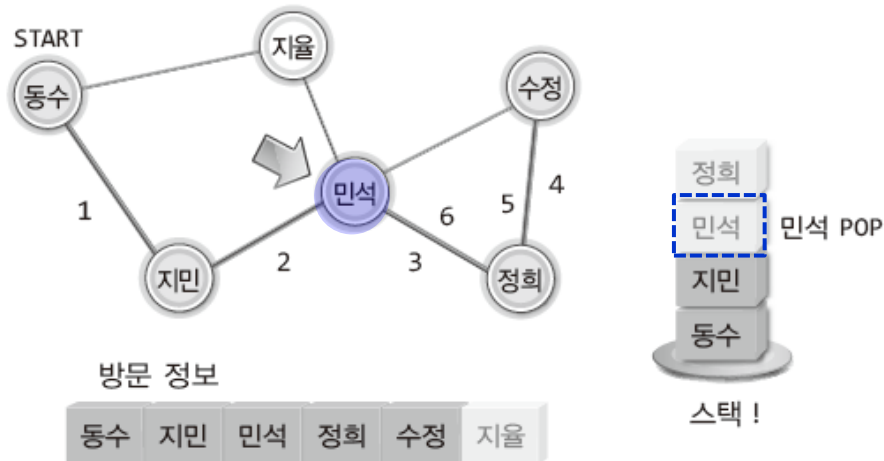
수정은 자신에게 연락한 사람의 정보를
스택에서 얻는다!

▶ [그림 14-30: DFS의 구현 3/7]



▶ [그림 14-31: DFS의 구현 4/7]

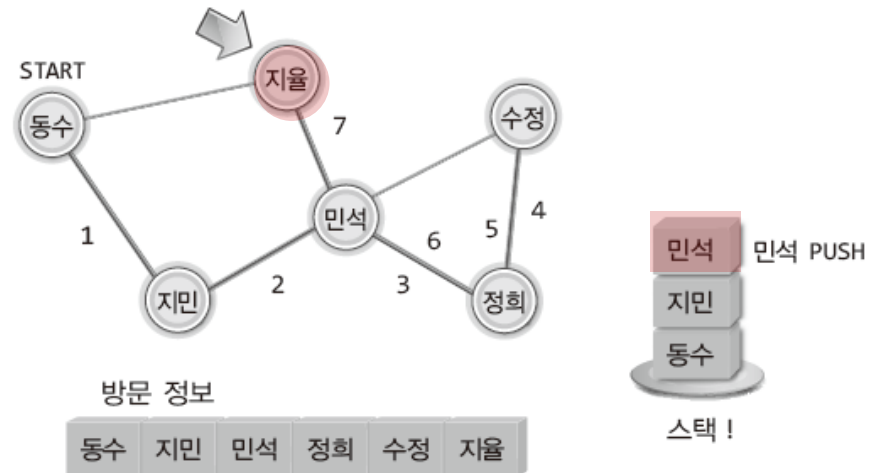
깊이 우선 탐색의 구현 모델: 과정 5~



▶ [그림 14-32: DFS의 구현 5/7]

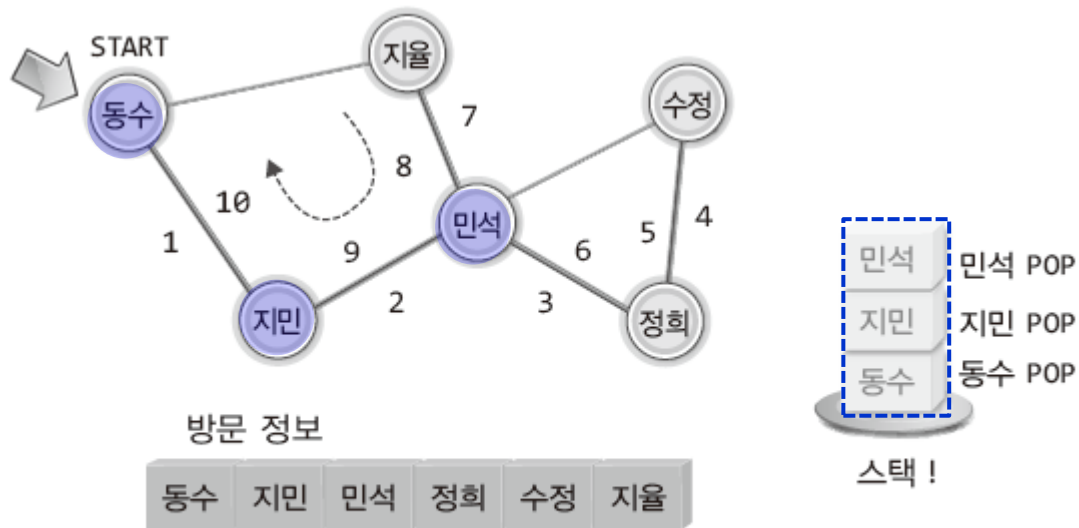


민석은 이전에 방문이 이뤄졌지만, 이와 상관 없이
민석을 떠날때 민석의 정보는 스택에 저장된다.



▶ [그림 14-33: DFS의 구현 6/7]

깊이 우선 탐색의 구현 모델: 과정 7



▶ [그림 14-34: DFS의 구현 7/7]

스택에 저장된 정보를 마지막까지 꺼내어 역으로 그 경로를 추적하다 보면 시작 위치로 이동이 가능하다!

깊이 우선 탐색의 실제 구현: 파일의 구성



깊이 우선 탐색의 실제 구현

```
void DFSShowGraphVertex(ALGraph * pg, int startV);
```

- 그래프의 모든 정점 정보를 출력하는 함수
- DFS를 기반으로 정의가 된 함수

구현 결과를 반영한 파일의 구성

- ALGraphDFS.h, ALGraphDFS.c
- ArrayBaseStack.h, ArrayBaseStack.c
- DLinkedList.h, DLinkedList.c
- DFSMain.c

그래프 관련

스택 관련(Chapter 06에서 구현)

연결 리스트 관련(Chapter 04에서 구현)

깊이 우선 탐색의 실제 구현: ALGraphDFS.h

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

정점의 이름을 결정하는 방법

멤버 *visitInfo* 관련 추가 코드

```
typedef struct _ual
{
    int numV;           // 정점의 수
    int numE;           // 간선의 수
    List * adjList;      // 간선의 정보
    int * visitInfo;     // 탐색과정에서 탐색이 진행된
                        // 정점 정보를 담기 위한 멤버 추가!
} 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);

// 정점의 정보 출력: Depth First Search 기반
void DFSshowGraphVertex(ALGraph * pg, int startV);
```

```
void GraphInit(ALGraph * pg, int nv)
{
    ....
    // 정점의 수를 길이로 하여 배열을 할당
    pg->visitInfo = (int *)malloc(sizeof(int) * pg->numV);

    // 배열의 모든 요소를 0으로 초기화!
    memset(pg->visitInfo, 0, sizeof(int) * pg->numV);
}
```

멤버 *visitInfo* 관련 추가 코드

```
void GraphDestroy(ALGraph * pg)
{
    ....
    // 할당된 배열의 소멸!
    if(pg->visitInfo != NULL)
        free(pg->visitInfo);
}
```


깊이 우선 탐색의 실제 구현: Helper Func

방문한 정점의 정보를 기록 및 출력

```
int VisitVertex(ALGraph * pg, int visitV)
{
    if(pg->visitInfo[visitV] == 0)        // visitV에 처음 방문일 때 '참'인 if문
    {
        pg->visitInfo[visitV] = 1;        // visitV에 방문한 것으로 기록
        printf("%c ", visitV + 65);      // 방문한 정점의 이름을 출력
        return TRUE;                      // 방문 성공!
    }
    return FALSE;                          // 방문 실패!
}
```

이미 방문한 정점이라면 FALSE가 반환된다!

DFShowGraphVertex 함수의 구현에 필요한, DFShowGraphVertex 함수 내에서 호출이 되는 함수로써 방문한 정점의 정보를 그래프의 멤버 visitInfo가 가리키는 배열에 등록하는 기능을 제공한다.



깊이 우선 탐색의 실제 구현: DFShow~ 함수의 정의

```
void DFShowGraphVertex(ALGraph * pg, int startV)
```

```
{
```

.... 초기화 영역

```
while(LFirst(&(pg->adjList[visitV]), &nextV) == TRUE)
```

```
{  
    연결된 정점의 정보를 얻어서!
```

```
    int visitFlag = FALSE;
```

```
    if(VisitVertex(pg, nextV) == TRUE) {
```

```
        .... 방문을 시도했는데 방문에 성공하면
```

```
    } else {
```

```
        .... 방문을 시도했는데 방문한적 있는 곳이라면
```

```
    }
```

```
    if(visitFlag == FALSE) { 연결된 정점과의 방문이 모두 완료되었다면,
```

```
        if(SIsEmpty(&stack) == TRUE)
```

```
            break; 스택이 비면! 종료!
```

```
        else
```

```
            visitV = SPop(&stack);
```

```
        } 되돌아 가기 위한 POP 연산!
```

```
    }
```

.... 마무리 영역

```
}
```

Stack stack;

int visitV = startV;

int nextV;

StackInit(&stack);

VisitVertex(pg, visitV); 시작 정점 방문!

SPush(&stack, visitV);

시작 정점 떠나면서

스택으로!

memset(
 pg->visitInfo, 0, sizeof(int) * pg->numV);

깊이 우선 탐색의 실제 구현: DFShow~ 함수의 정의

```
void DFShowGraphVertex(ALGraph * pg, int startV)
```

```
{
```

.... 초기화 영역

```
while(LFirst(&(amp;pg->adjList[visitV]), &nextV) == TRUE)
```

```
{
```

```
int visitFlag = FALSE;
```

```
if(VisitVertex(pg, nextV) == TRUE) {
```

.... 방문을 시도했는데 방문에 성공하면

```
} else {
```

.... 방문을 시도했는데 방문한적 있는 곳이라면

```
}
```

```
if(visitFlag == FALSE) {
```

```
if(SIsEmpty(&stack) == TRUE)
```

```
break;
```

```
else
```

```
visitV = SPop(&stack);
```

```
}
```

```
}
```

.... 마무리 영역

```
}
```

방문한 정점을 떠나야 하니 해당

정보 스택으로!

```
SPush(&stack, visitV);
```

```
visitV = nextV;
```

```
visitFlag = TRUE;
```

연결된 다른 정점을 찾아서 방문

을 시도하는 일련의 과정!

```
while(LNext(&(amp;pg->adjList[visitV]), &nextV) == TRUE)
```

```
{
```

```
if(VisitVertex(pg, nextV) == TRUE)
```

```
{
```

```
SPush(&stack, visitV);
```

```
visitV = nextV;
```

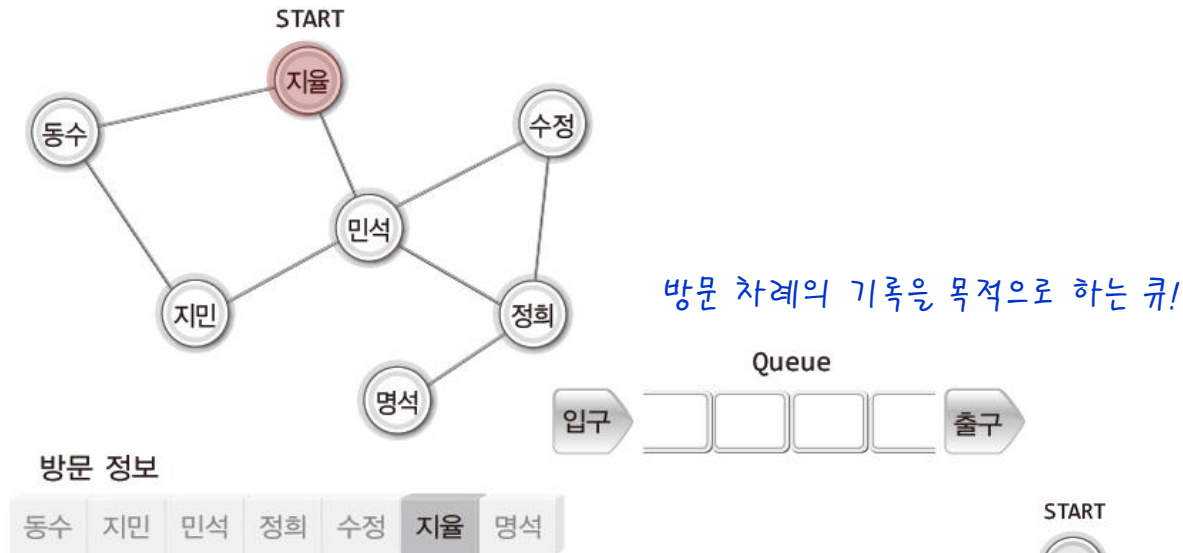
```
visitFlag = TRUE;
```

```
break;
```

```
}
```

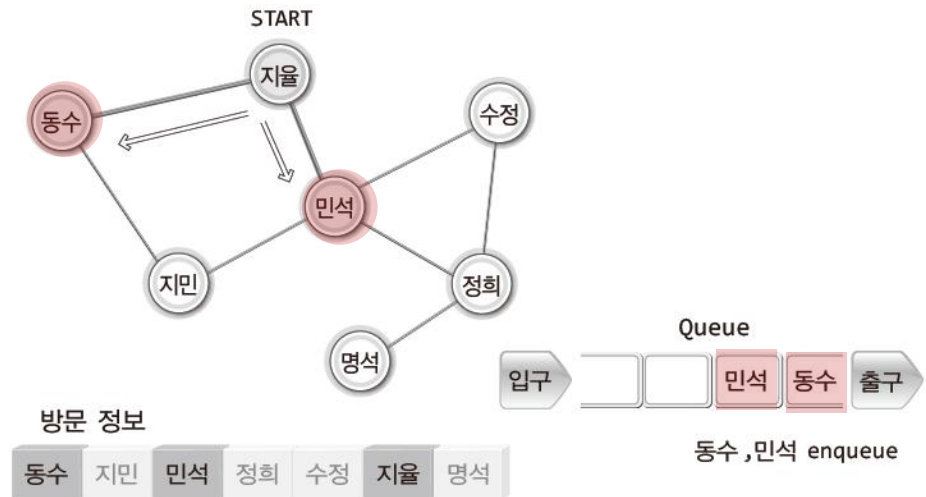
```
}
```

너비 우선 탐색의 구현 모델: 과정 1~



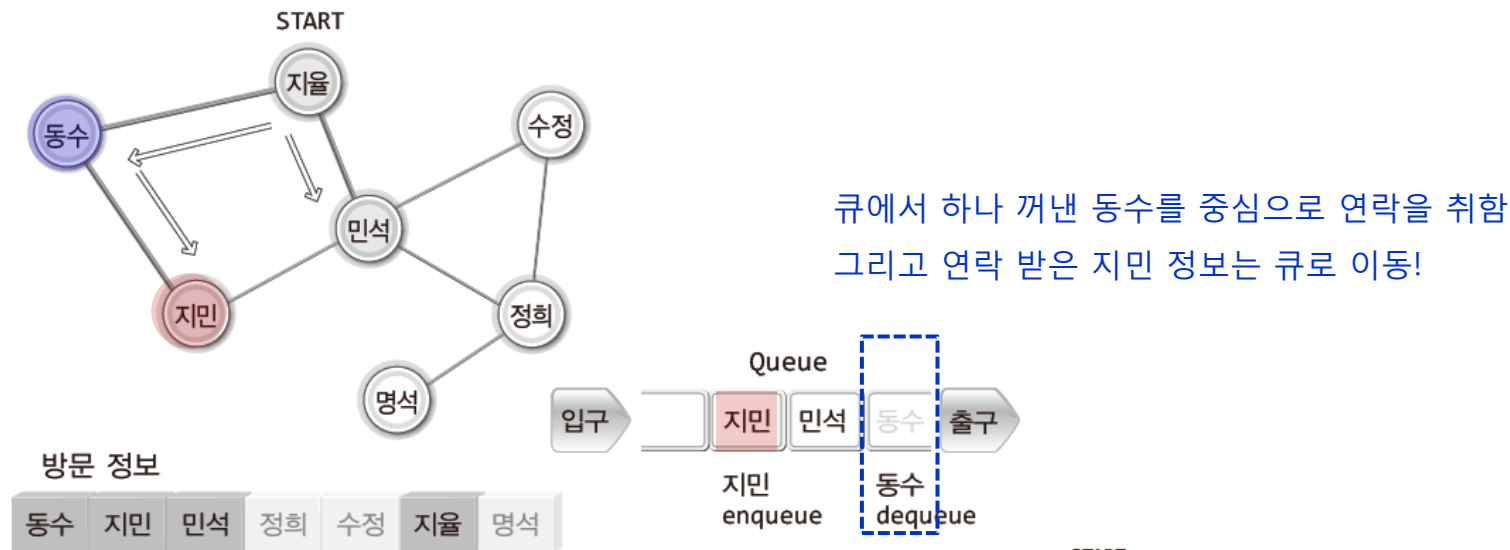
▶ [그림 14-35: BFS의 구현 1/5]

동수와 민석은 연락을 받기만 했을 뿐 연락을 취하지는 않은 대상! 이러한 대상의 정보를 큐에 저장!



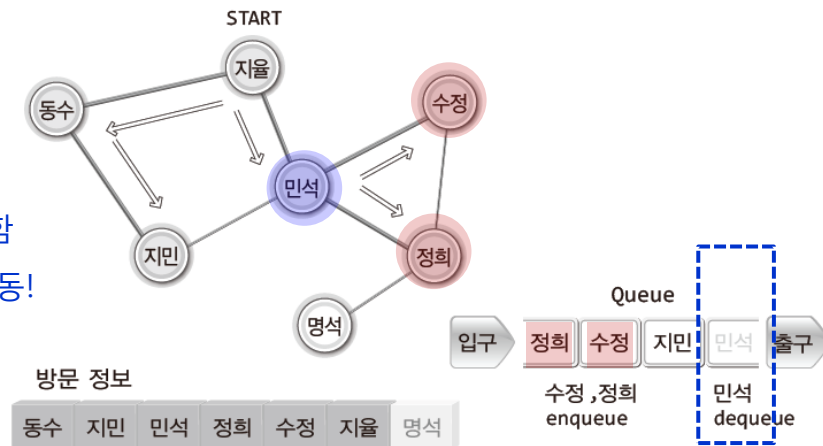
▶ [그림 14-36: BFS의 구현 2/5]

너비 우선 탐색의 구현 모델: 과정 3~



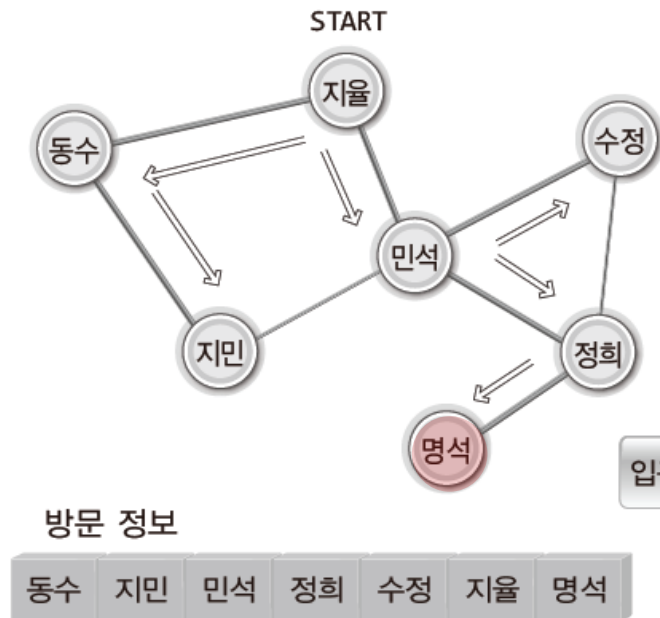
▶ [그림 14-37: BFS의 구현 3/5]

큐에서 하나 꺼낸 민석을 중심으로 연락을 취함
그리고 연락 받은 수정과 정희 정보는 큐로 이동!

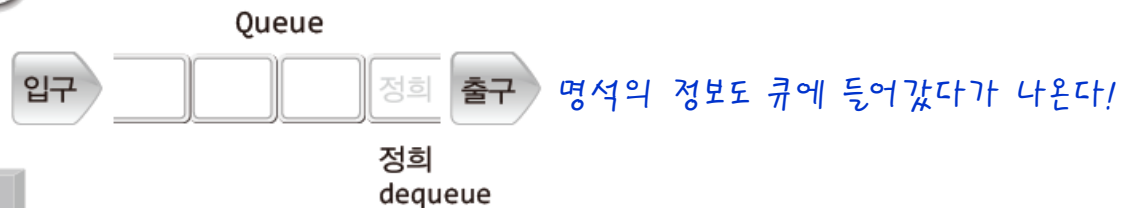


▶ [그림 14-38: BFS의 구현 4/5]

너비 우선 탐색의 구현 모델: 과정 5



큐에서 꺼내어 방문을 진행하고,
방문한 정점의 정보를 다시 큐에 넣고.. 하는 일련의 과정을 반복!
언제까지? 큐가 완전히 빌 때까지!



▶ [그림 14-39: BFS의 구현 5/5]

위의 그림에서는 명석의 정보가 마지막에 큐에 들어간다.
그리고 빠져나오면서 종료하게 된다!

너비 우선 탐색의 실제 구현



너비 우선 탐색의 실제 구현

```
void BFSshowGraphVertex(ALGraph * pg, int startV);
```

- 그래프의 모든 정점 정보를 출력하는 함수
- BFS를 기반으로 정의가 된 함수

구현 결과를 반영한 파일의 구성

- ALGraphBFS.h, ALGraphBFS.c
- CircularQueue.h, CircularQueue.c
- DLinkedList.h, DLinkedList.c
- BFSain.c

그래프 관련

큐 관련(Chapter 07에서 구현)

연결 리스트 관련(Chapter 04에서 구현)

너비 우선 탐색의 실제 구현: ALGraphBFS.h

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

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

```
// 그래프의 초기화
```

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

ALGraphDFS.h와 동일하다!

```
// 그래프의 리소스 해제
```

```
void GraphDestroy(ALGraph * pg);
```

```
// 간선의 추가
```

```
void AddEdge(ALGraph * pg, int fromV, int toV);
```

```
// 그래프의 간선 정보 출력
```

```
void ShowGraphEdgeInfo(ALGraph * pg);
```

```
// BFS 기반 그래프의 정점 정보 출력
```

```
void BFSShowGraphVertex(ALGraph * pg, int startV);
```



너비 우선 탐색의 실제 구현: Helper Func

```
void BFSshowGraphVertex(ALGraph * pg, int startV)
{
    Queue queue;
    int visitV = startV;
    int nextV;

    QueueInit(&queue);
    VisitVertex(pg, visitV);
    시작점 방문!

    while(LFirst(&(pg->adjList[visitV]), &nextV) == TRUE)
    {
        visitV에 연결된 정점 정보 얻음
        if(VisitVertex(pg, nextV) == TRUE)
            Enqueue(&queue, nextV);

        while(LNext(&(pg->adjList[visitV]), &nextV) == TRUE)
        {
            계속해서 visitV에 연결된 정점 정보 얻음
            if(VisitVertex(pg, nextV) == TRUE)
                Enqueue(&queue, nextV);
        }

        if(QIsEmpty(&queue) == TRUE)
            break;
        큐가 비면 탈출 조건이 성립!
        else
            visitV = Dequeue(&queue);
    }

    memset(pg->visitInfo, 0, sizeof(int) * pg->numV);
}
```

코드의 전체적인 느낌이 DFSshowGraphVertex와 유사하다. 그리고 그 함수보다 간결하다!

Chapter 14. 그래프

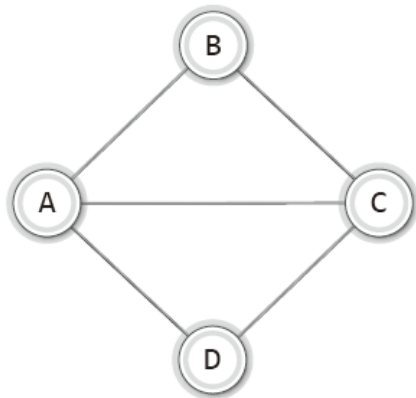


Chapter 14-4:

최소 비용 신장 트리



사이클의 이해



정점 B에서 점점 D에 이르는 단순 경로

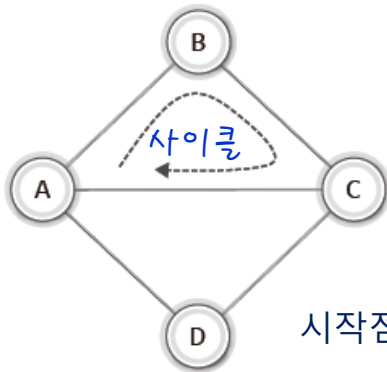
- B-A-D 단순 경로
- B-C-D 단순 경로
- B-A-C-D 조금 돌아가는 단순 경로
- B-C-A-D 조금 돌아가는 단순 경로

단순 경로는 간선을 중복 포함하지 않는다.

단순 경로가 아닌 정점 B에서 점점 D에 이르는 경로

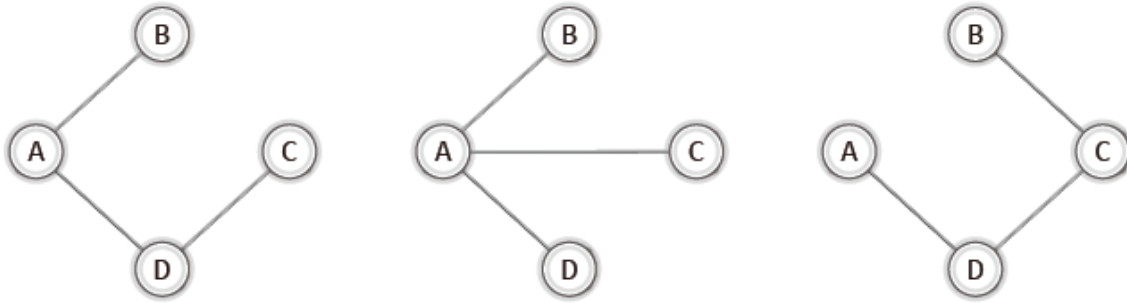
- B-A-C-B-A-D

B와 A를 잇는 간선이 두 번 포함됨!



시작점과 끝점이 같은 단순 경로를 가리켜 '사이클' 이라 한다.

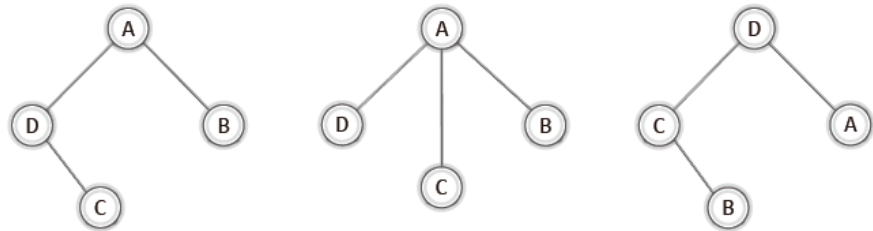
사이클을 형성하지 않는 그래프



어떻게 경로를 구성하더라도 '사이클'을 형성하지 않는 그래프!

이러한 종류의 그래프를 가리켜 '신장 트리'라 한다.

위의 그래프를 회전시킨 결과



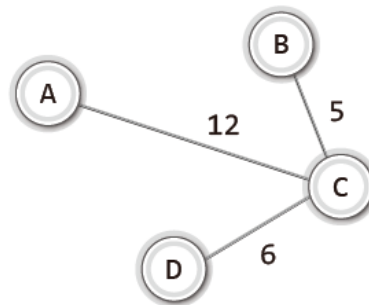
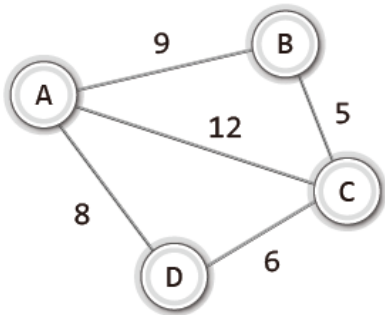
사이클을 형성하지 않는 그래프들은 일종의 트리로 볼 수 있다.

그래서 이들을 가리켜 신장 그래프가 아닌 신장 트리라 한다.

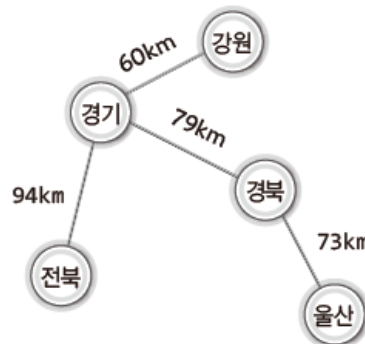
최소 비용 신장 트리의 이해와 적용

- 그래프의 모든 정점이 간선에 의해서 하나로 연결되어 있다.
- 그래프 내에서 사이클을 형성하지 않는다.

신장 트리의 특징



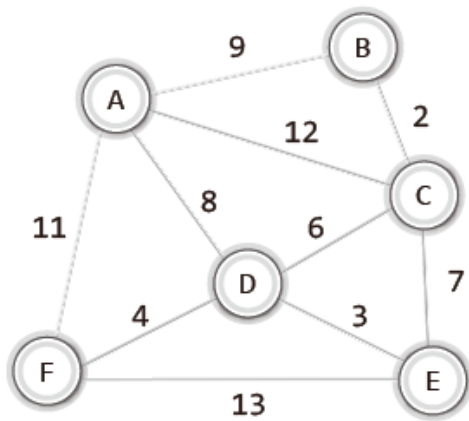
'최소 비용 신장 트리' 구성의 예



'최소 비용 신장 트리' 구성의 예

크루스칼 알고리즘 1: 과정 1~

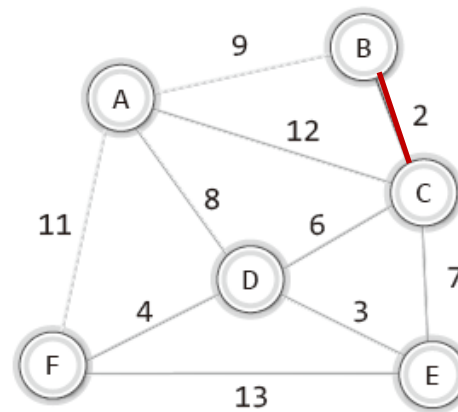
가중치를 기준으로 간선을 정렬한 후에 MST가 될 때까지 간선을 하나씩 선택 또는 삭제해 나가는 방식



▶ [그림 14-49: 크루스칼 알고리즘 1의 1/4]

2, 3, 4, 6, 7, 8, 9, 11, 12, 13

가중치의 오름차순 정렬



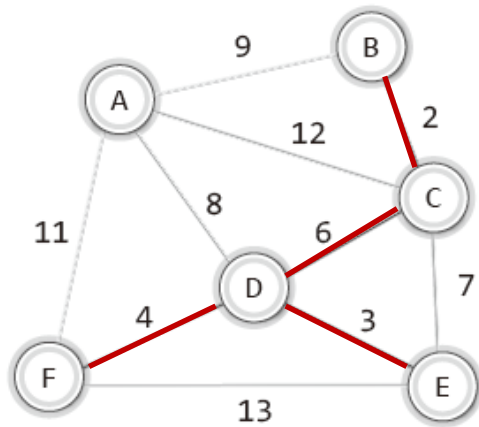
▶ [그림 14-50: 크루스칼 알고리즘 1의 2/4]



2, 3, 4, 6, 7, 8, 9, 11, 12, 13

가중치의 오름차순 정렬

크루스칼 알고리즘 1: 과정 3~



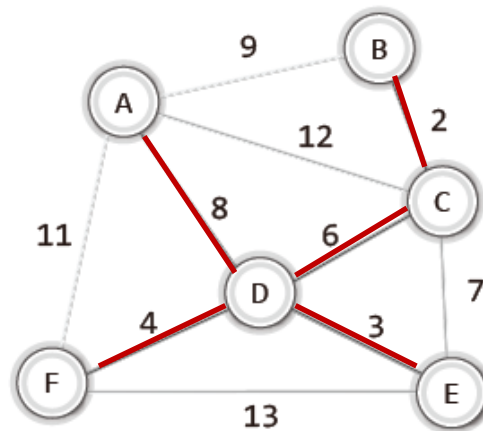
이동
↓ ↓ ↓
2, 3, 4, 6, 7, 8, 9, 11, 12, 13
가중치의 오름차순 정렬

▶ [그림 14-51: 크루스칼 알고리즘 1의 3/4]

최소 비용 신장 트리의 조건인

간선의 수 + 1 = 정점의 수를 만족하니

이것으로 최소 비용 신장 트리 형성 완료!



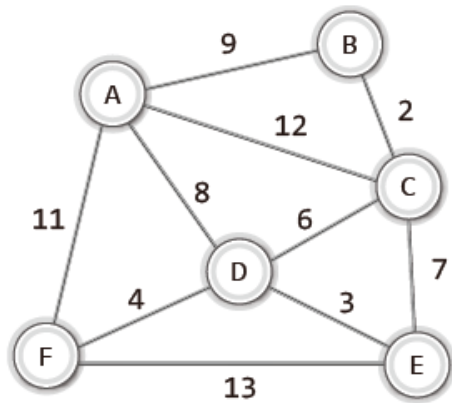
가중치가 7인 간선을 포함시키면
사이클이 형성된다! 따라서 건너 뛴다!

↓
2, 3, 4, 6, 7, 8, 9, 11, 12, 13
가중치의 오름차순 정렬

▶ [그림 14-52: 크루스칼 알고리즘 1의 4/4]

크루스칼 알고리즘 2: 과정 1~

높은 가중치의 간선을 하나씩 빼는 방식의 크루스칼 알고리즘

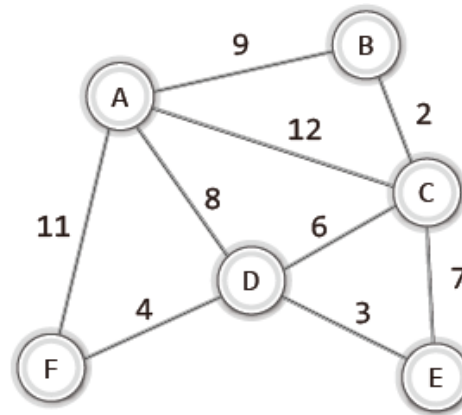


13, 12, 11, 9, 8, 7, 6, 4, 3, 2

가중치의 내림차순 정렬

▶ [그림 14-54: 크루스칼 알고리즘 2의 1/4]

가중치가 13인 간선이 없어도 모든 정점은 연결
이 되므로 이를 삭제한다.



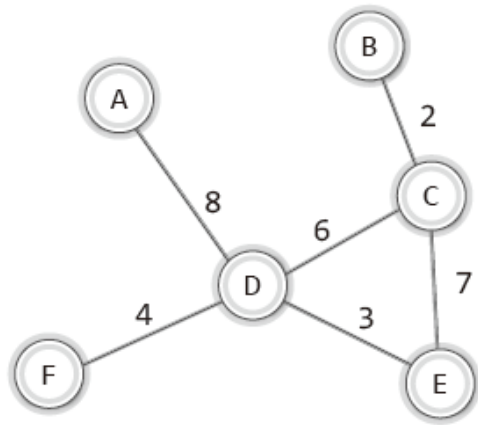
▶ [그림 14-55: 크루스칼 알고리즘 2의 2/4]



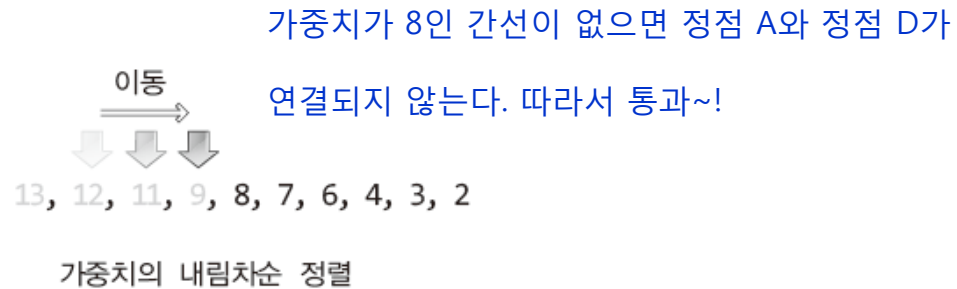
13, 12, 11, 9, 8, 7, 6, 4, 3, 2

가중치의 내림차순 정렬

크루스칼 알고리즘 2: 과정 3~



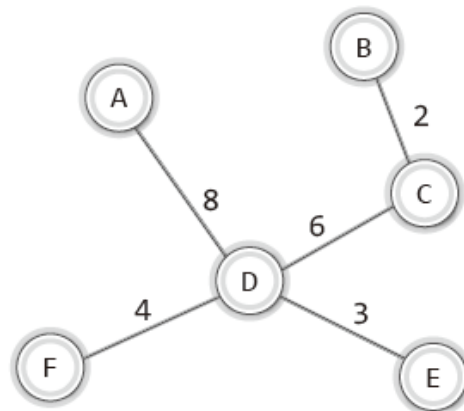
▶ [그림 14-56: 크루스칼 알고리즘 2의 3/4]



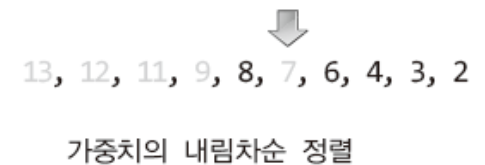
최소 비용 신장 트리의 조건인

간선의 수 + 1 = 정점의 수를 만족하니

이것으로 최소 비용 신장 트리 형성 완료!



▶ [그림 14-57: 크루스칼 알고리즘 2의 4/4]



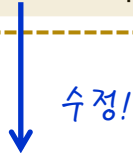
크루스칼 알고리즘의 구현을 위한 계획1

우리가 선택한 구현 방식

가중치를 기준으로 간선을 내림차순으로 정렬한 다음 높은 가중치의 간선부터 시작해서 하나씩 그래프에서 제거하는 방식

구현에 사용할 도구들

- | | |
|--------------------------------------|--------------------|
| · DLinkedList.h, DLinkedList.c | 연결 리스트 |
| · ArrayBaseStack.h, ArrayBaseStack.c | 배열 기반 스택 |
| · ALGraphDFS.h, ALGraphDFS.c | 깊이 우선 탐색을 포함하는 그래프 |



수정!

크루스칼 알고리즘이 담기는 파일들

- | | |
|--------------------------------------|----------------|
| · ALGraphKruskal.h, ALGraphKruskal.c | 가중치 그래프의 구현 결과 |
|--------------------------------------|----------------|

그리고 가중치 그래프의 구현을 위해서는 가중치가 포함된 간선을 표현한 구조체가 정의되어야 한다.
헤더파일 ALEdge.h를 만들어서 해당 구조체를 정의한다.

크루스칼 알고리즘의 구현을 위한 계획2

다음 질문에 답을 하는 함수가 필요하다!

이 간선을 삭제한 후에도 이 간선에 의해 연결된 두 정점을 연결하는 경로가 있는가?

이를 위해 DFS 알고리즘을 활용! DFS의 구현결과인 DFShowGraphVertex 함수를 확장하여 이 질문에 답을 하도록 한다!

이는 크루스칼 알고리즘의 일부이다.

그래프를 구성하는 간선들을 가중치를 기준으로 정렬할 수 있어야 한다.

이를 위해서 앞서 구현한 우선순위 큐를 활용

- PriorityQueue.h, PriorityQueue.c 우선순위 큐
- UsefulHeap.h, UsefulHeap.c 우선순위 큐의 기반이 되는 힙

크루스칼 알고리즘의 구현을 위한 계획3

• DLinkedList.h, DLinkedList.c	연결 리스트
• ArrayBaseStack.h, ArrayBaseStack.c	배열 기반 스택
• ALGraphKruskal.h, ALGraphKruskal.c	크루스칼 알고리즘 기반의 그래프
• PriorityQueue.h, PriorityQueue.c	우선순위 큐
• UsefulHeap.h, UsefulHeap.c	우선순위 큐의 기반이 되는 힙
• ALEdge.h	가중치가 포함된 간선의 표현을 위한 구조체

최종적으로 크루스칼 알고리즘의 구현을 보이기 위한 프로젝트의 헤더파일과 소스파일의 구성

크루스칼 알고리즘의 구현: 헤더파일

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

```
typedef struct _ual
```

```
{
```

```
    int numV;
```

```
    int numE;
```

```
    List * adjList;
```

```
    int * visitInfo;
```

```
    PQueue pqueue;    // 간선의 가중치 정보 저장
```

```
} ALGraph;
```

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

```
void GraphDestroy(ALGraph * pg);
```

```
void AddEdge(ALGraph * pg, int fromV, int toV, int weight);
```

```
void ShowGraphEdgeInfo(ALGraph * pg);
```

```
void DFSshowGraphVertex(ALGraph * pg, int startV);
```

```
void ConKruskalMST(ALGraph * pg);    // 최소 비용 신장 트리의 구성
```

```
void ShowGraphEdgeWeightInfo(ALGraph * pg);    // 가중치 정보 출력
```

```
typedef struct _edge
```

```
{
```

```
    int v1;    // 간선이 연결하는 첫 번째 정점
```

```
    int v2;    // 간선이 연결하는 두 번째 정점
```

```
    int weight; // 간선의 가중치
```

```
} Edge;
```

ALEdge.h

ALGraphKruskal.h

크루스칼 알고리즘을 구현한 함수: 수정된 함수들

```
void GraphInit(ALGraph * pg, int nv)
{
    . . . . 여기까지는 ALGraphDFS.c의 GraphInit 함수와 동일 . . .

    // 우선순위 큐의 초기화
    PQueueInit(&(pg->pqueue), PQWeightComp);    // 추가된 문장
}
```

```
int PQWeightComp(Edge d1, Edge d2)
{
    return d1.weight - d2.weight;
}
```

가중치 기준 내림차순으로
간선 정보 꺼내기 위한 정의!

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

    // 간선의 가중치 정보를 우선순위 큐에 저장
    PEnqueue(&(pg->pqueue), edge);
}
```

크루스칼 알고리즘을 구현한 함수: ConKruskalMST

```
void ConKruskalMST(ALGraph * pg)    // 크루스칼 알고리즘 기반 MST의 구성
{
    Edge recvEdge[20];    // 복원할 간선의 정보 저장
    Edge edge;
    int eidx = 0;
    int i;

    // MST를 형성할 때까지 아래의 while문을 반복
    while(pg->numE+1 > pg->numV)    // MST 간선의 수 + 1 == 정점의 수
    {
        edge = PDequeue(&(pg->pqueue)); 가중치 순으로 간선 정보 획득!
        RemoveEdge(pg, edge.v1, edge.v2); 획득한 정보의 간선 실제 삭제!

        if(!IsConnVertex(pg, edge.v1, edge.v2)) 삭제 후 두 정점 연결 경로 있는지 확인!
        {
            RecoverEdge(pg, edge.v1, edge.v2, edge.weight); 연결 경로 없으면 간선 복원!
            recvEdge[eidx++] = edge;
        }
    }

    // 우선순위 큐에서 삭제된 간선의 정보를 회복
    for(i=0; i<eidx; i++)
        PEnqueue(&(pg->pqueue), recvEdge[i]);
}
```

- RemoveEdge 그래프에서 간선을 삭제한다.
- IsConnVertex 두 정점이 연결되어 있는지 확인한다.
- RecoverEdge 삭제된 간선을 다시 삽입한다.

크루스칼 알고리즘의 완성을 돕는 함수들 1

```
// 간선의 소멸
void RemoveEdge(ALGraph * pg, int fromV, int toV)
{
    RemoveWayEdge(pg, fromV, toV);
    RemoveWayEdge(pg, toV, fromV);
    (pg->numE)--;
}
```

인접 리스트 기반 무방향 그래프인 관계로 하나의 간선을 완전히 소멸하기 위해서는 두 개의 간선 정보를 소멸시켜야 한다.

```
void RecoverEdge(ALGraph * pg, int fromV, int toV, int weight)
{
    LInsert(&(amp;pg->adjList[fromV]), toV);
    LInsert(&(amp;pg->adjList[toV]), fromV);
    (pg->numE)++;
}
```

AddEdge 함수와 달리 간선의 가중치 정보를 별도로 저장하지 않는다. 이렇듯 가중치 정보를 별도로 저장하지 않는 이유는 크루스칼 알고리즘의 구현 내용을 통해 이해할 수 있다.

크루스칼 알고리즘의 완성을 돕는 함수들 2

```
// 한쪽 방향의 간선 소멸
void RemoveWayEdge(ALGraph * pg, int fromV, int toV)
{
    int edge;
    if(LFirst(&(amp;pg->adjList[fromV]), &edge))
    {
        if(edge == toV) {
            LRemove(&(amp;pg->adjList[fromV]));
            return;
        }
        while(LNext(&(amp;pg->adjList[fromV]), &edge))
        {
            if(edge == toV) {
                LRemove(&(amp;pg->adjList[fromV]));
                return;
            }
        }
    }
}
```

이렇듯 RemoveEdge 함수의 완성을 돕는 RemoveWayEdge 함수를 별도로 정의하면 방향 그래프의 구현을 위한 확장이 용이하다!

크루스칼 알고리즘의 완성을 돕는 함수들 3

// 인자로 전달된 두 정점이 연결되어 있다면 TRUE, 그렇지 않다면 FALSE 반환

```
int IsConnVertex(ALGraph * pg, int v1, int v2)
```

```
{
```

```
    Stack stack;
```

```
    int visitV = v1;
```

```
    int nextV;
```

```
    StackInit(&stack);
```

```
    VisitVertex(pg, visitV);
```

```
    SPush(&stack, visitV);
```

```
    while(LFirst(&(amp;pg->adjList[visitV]), &nextV) == TRUE)
```

```
    {
```

```
        int visitFlag = FALSE;
```

```
        // 정점을 돌아다니는 도중에 목표를 찾는다면 TRUE를 반환한다.
```

```
        if(nextV == v2) {
```

```
            // 함수가 반환하기 전에 초기화를 진행한다.
```

```
            memset(pg->visitInfo, 0, sizeof(int) * pg->numV);
```

```
            return TRUE;        // 목표를 찾았으니 TRUE를 반환!
```

```
        }
```

```
    if(VisitVertex(pg, nextV) == TRUE)
```

```
    {
```

```
        SPush(&stack, visitV);
```

```
        visitV = nextV;
```

```
        visitFlag = TRUE;
```

```
    }
```

```
    else
```

```
    {
```

```
        while(LNext(&(amp;pg->adjList[visitV]), &nextV) == TRUE)
```

```
        {
```

```
            // 정점을 돌아다니는 도중에 목표를 찾는다면 TRUE를 반환한다.
```

```
            if(nextV == v2) {
```

```
                // 함수가 반환하기 전에 초기화를 진행한다.
```

```
                memset(pg->visitInfo, 0, sizeof(int) * pg->numV);
```

```
                return TRUE;        // 목표를 찾았으니 TRUE를 반환!
```

```
            }
```

```
        if(VisitVertex(pg, nextV) == TRUE) {
```

```
            SPush(&stack, visitV);
```

```
            visitV = nextV;
```

```
            visitFlag = TRUE;
```

```
            break;
```

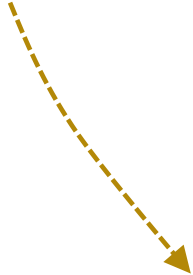
```
        }
```

```
    }
```

```
}
```

DFShowGraphVertex 함수와의 비교를 통해서 어떻게 수정
되었고 또 그 결과 어떻게 두 정점의 연결을 확인하는지 이
해하자!

크루스칼 알고리즘의 완성을 돕는 함수들 3



```
if(visitFlag == FALSE)
{
    if(SIsEmpty(&stack) == TRUE)
        break;
    else
        visitV = SPop(&stack);
}
}

memset(pg->visitInfo, 0, sizeof(int) * pg->numV);
return FALSE;    // 여기까지 왔다는 것은 목표를 찾지 못했다는 것!
}
```

이로써 부분적으로 필요한 모든 설명이 완료 되었으니 전체 코드를 확인하고 교재에서 제공하는 main 함수의 실행 결과도 직접 확인해보자!

수고하셨습니다~



Chapter 14에 대한 강의를 마칩니다!

