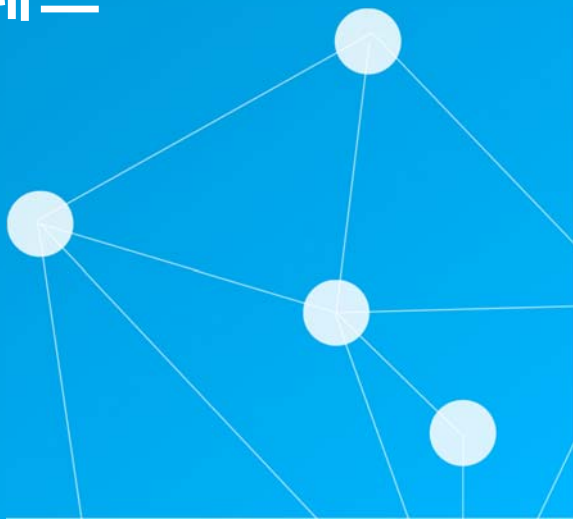
DATA
STRUCTURES
USING C

- 그래프의 개념을 이해한다.
- 그래프를 표현하는 2가지 방법을 이해한다.
- 그래프의 탐색 방법을 이해한다.
- 그래프 탐색을 이용한 문제해결 능력을 배양한다.
- 다양한 문제에 그래프를 활용할 수 있는 능력을 기른다.

CHAPTER 11

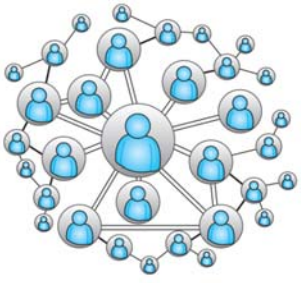
# 그래프





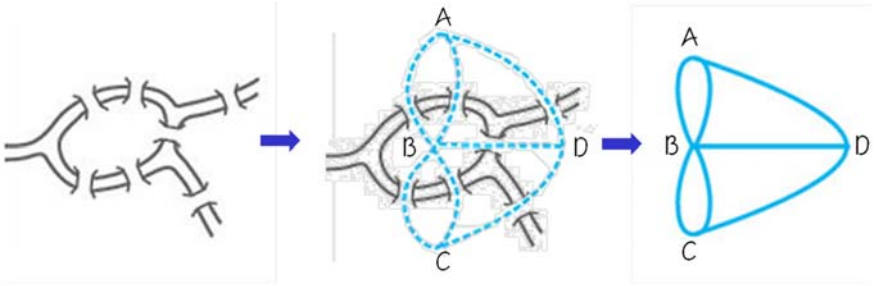
## 11.1 그래프(graph)란?

- 연결되어 있는 객체 간의 관계를 표현하는 자료구조
- 가장 일반적인 자료구조 형태
  - 그래프의 예
    - 트리(tree)
    - 지도, 지하철 노선도
    - 전기회로의 연결 상태
    - OS의 프로세스와 자원 관계
    - 인맥지도



# 그래프 역사

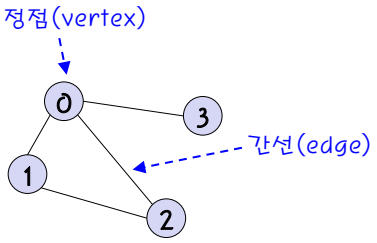
- 오일러 문제 (1800년대)
  - 다리를 한번만 건너서 처음 출발했던 장소로 돌아오는 문제
    - 위치: 정점(node), 다리: 간선(edge)
  - 오일러 정리
    - 모든 정점에 연결된 간선의 수가 짝수이면 오일러 경로 존재함
    - 따라서 그래프 (b)에는 오일러 경로가 존재하지 않음



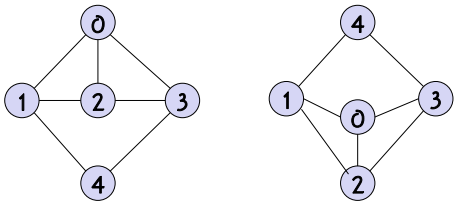



# 그래프 정의

- 그래프 G는 (V, E)로 표시
  - 정점(vertices) 또는 노드(node)
    - 여러 가지 특성을 가질 수 있는 객체 의미
    - V(G) : 그래프 G의 정점들의 집합
  - 간선(edge) 또는 링크(link)
    - 정점들 간의 관계 의미
    - E(G) : 그래프 G의 간선들의 집합

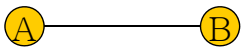



  - 동일한 그래프?

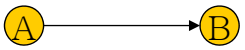


# 그래프의 종류

- 간선의 종류에 따라
  - 무방향 그래프(undirected graph)
    - 간선을 통해서 양방향으로 갈수 있음
    - (A, B)로 표현
    - (A, B) = (B, A)

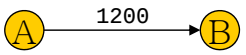


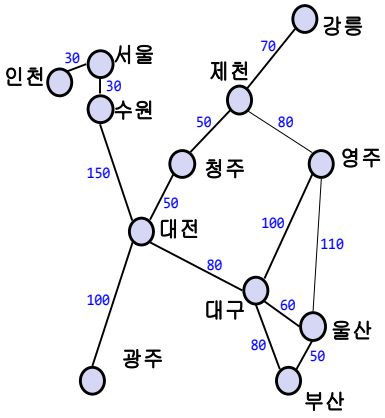

  - 방향 그래프(directed graph)
    - 간선을 통해서 한쪽 방향으로만 갈 수 있음
    - 일방통행 길
    - <A, B> 로 표현
    - <A, B> ≠ <B, A>






- 가중치 그래프(weighted graph)
  - 네트워크(network)라고도 함
  - 간선에 비용(cost)이나 가중치(weight)가 할당된 그래프
  - 가중치 그래프 예
    - 정점 : 각 도시를 의미
    - 간선 : 도시를 연결하는 도로 의미
    - 가중치 : 도로의 길이

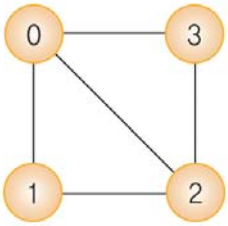


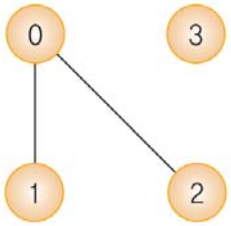

- 부분 그래프
  - 정점 집합 V(G)와 간선 집합 E(G)의 부분 집합으로 이루어진 그래프

# 그래프 표현의 예

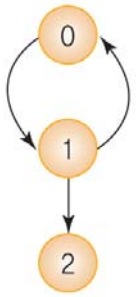
- 그래프 표현의 예
  - V(G1)= {0, 1, 2, 3}, E(G1)= {(0, 1), (0, 2), (0, 3), (1, 2), (2, 3)}
  - V(G2)= {0, 1, 2, 3}, E(G2)= {(0, 1), (0, 2))}
  - V(G3)= {0, 1, 2}, E(G3)= {<0, 1>, <1, 0>, <1, 2>}



G1



G2

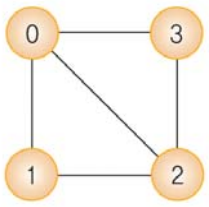


G3

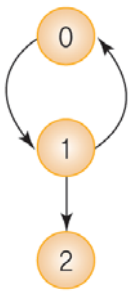


# 그래프의 용어

- 인접 정점(adjacent vertex)
  - 하나의 정점에서 간선에 의해 직접 연결된 정점
  - G1에서 정점 1의 인접 정점
    - 정점 0, 정점 2
- 차수(degree)
  - 하나의 정점에 연결된 다른 정점의 수
  - 무방향 그래프
    - G1에서 정점 0의 차수: 3
    - 차수의 합은 간선 수의 2배
  - 방향 그래프
    - 진입차수, 진출차수
    - 모든 진입(진출) 차수의 합은 간선의 수



G1

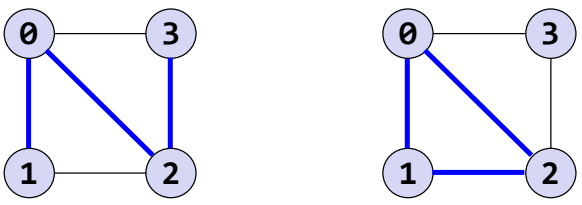


G3

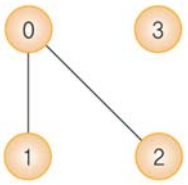
- 그래프의 경로(path)
  - 무방향 그래프의 정점 s로부터 정점 e까지의 경로
    - 정점의 나열 s, v1, v2, ..., vk, e
    - 반드시 간선 (s, v1), (v1, v2), ... , (vk, e) 존재해야 함
  - 방향 그래프의 정점 s로부터 정점 e까지의 경로
    - 정점의 나열 s, v1, v2, ..., vk, e
    - 반드시 간선 <s, v1>, <v1, v2>, ... ,<vk, e> 존재해야 함
- 경로의 길이(length)
  - 경로를 구성하는데 사용된 간선의 수



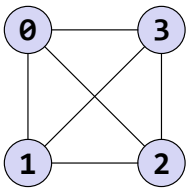
- 단순 경로(simple path)
  - 경로 중에서 반복되는 간선이 없는 경로
    - 1, 0, 2, 3은 단순경로
    - 1, 0, 2, 0은 단순경로 아님
- 사이클(cycle)
  - 시작 정점과 종료 정점이 동일한 단순 경로
    - 0, 1, 2, 0은 사이클



- 연결 그래프(connected graph)
  - 모든 정점쌍에 대한 경로 존재
  - G2는 비연결 그래프임
- 트리(tree)
  - 그래프의 특수한 형태로서 사이클을 가지지 않는 연결 그래프
- 완전 그래프(complete graph)
  - 모든 정점 간에 간선이 존재하는 그래프
  - n개의 정점을 가진 무방향 완전그래프의 간선의 수
    - n×(n-1)/2
  - n=4, 간선의 수 = (4×3)/2 = 6

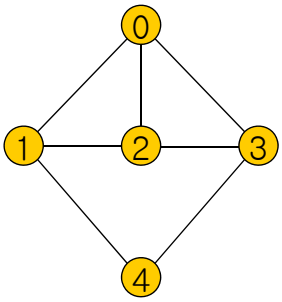


G2






- 아래 그래프를 집합으로 표현하고, 각 노드의 차수를 보여라.



# 그래프 ADT

*데이터*

정점의 집합과 간선의 집합

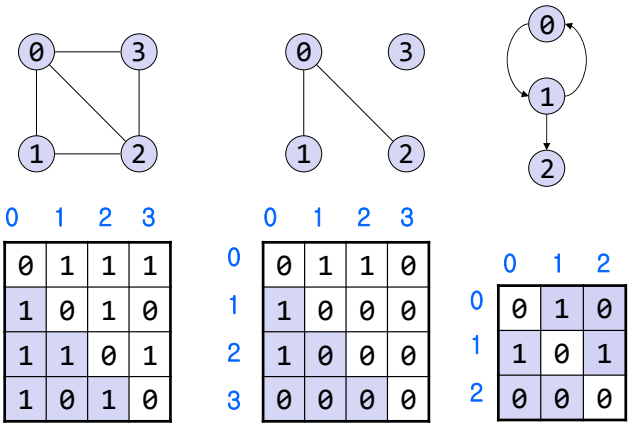
*연산*

- **init()**: 그래프를 초기화한다.
- **is_empty()**: 그래프가 공백 상태인지 확인한다.
- **insert_vertex(v)**: 그래프에 정점 v를 삽입한다.
- **insert_edge(u,v)**: 그래프에 간선 (u,v)를 삽입한다.
- **delete_vertex(v)**: 그래프의 정점 v를 삭제한다.
- **delete_edge(u,v)**: 그래프 g의 간선 (u,v)를 삭제한다.
- **adjacent(v)**: 정점 v에 인접한 모든 정점의 집합을 반환한다.



# 11.2 그래프 표현 방법 1 : 인접 행렬

- nxn 의 인접행렬 M을 이용
  - 간선 (i, j)가 있으면
    - M[i][j] = 1, 또는 true
  - 그렇지 않으면
    - M[i][j] = 0, 또는 false
  - 대각선 성분은 모두 0
  - 무방향 그래프
    - 인접 행렬이 대칭



- 인접 행렬 adj_mat[][]에서 어떤 정점 v의 진입 차수를 알고 싶으면 어떻게 하면 되는가?



# 인접 행렬을 이용한 그래프 표현

- 그래프 데이터와 기본 연산

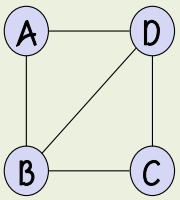
```
typedef char VtxData;              // 그래프 정점에 저장할 데이터의 자료형
int adj[MAX_VTXS][MAX_VTXS];       // 인접 행렬
int vsize;                         // 전체 정점의 개수
VtxData vdata[MAX_VTXS];           // 정점에 저장할 데이터 배열
```

```
void init_graph() {
  int i, j;
  vsize=0;
  for(i=0 ; i<MAX_VTXS; i++)
  for(j=0 ; j<MAX_VTXS; j++)
    adj[i][j] = 0;
}
void insert_vertex( char name ) {
  if (is_full_graph())
    error("Error: 정점 개수 초과\n");
  else
    vdata[vsize++] = name;
}
```

```
void insert_edge(int u, int v, int val)
{
  adj[u][v] = val;
}
void insert_edge2(int u, int v, int val)
{
  adj[u][v] = adj[v][u] = val;
}
```

# 전체 프로그램

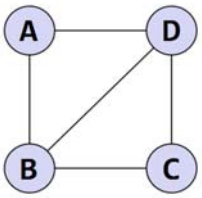
```
void main()
{
        int i;

        init_graph( );
        for( i=0 ; i<4 ; i++ )
                insert_vertex( 'A'+i );
        insert_edge2(0,1, 1);
        insert_edge2(0,3, 1);
        insert_edge2(1,2, 1);
        insert_edge2(1,3, 1);
        insert_edge2(2,3, 1);
        print_graph("그래프(인접행렬)\n");
}
```




```
C:₩WINDOWS₩system32₩cmd.exe
그래프(인접행렬)
4
A     0   1   0   1
B     1   0   1   1
C     0   1   0   1
D     1   1   1   0
계속하려면 아무 키나 누르십시오 . . .
```



# 그래프 파일 입출력



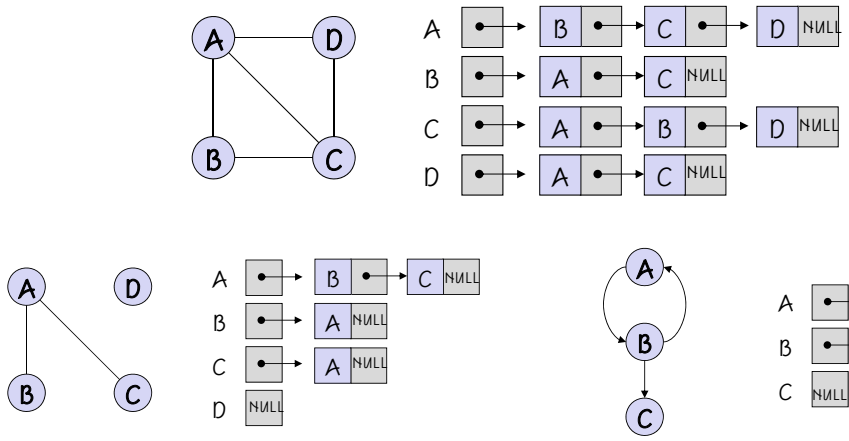


```
void print_graph(FILE *fp, char* msg)
{
  int i, j;
  fprintf(fp, "%s", msg);
  fprintf(fp, "%d\n", vsize);
  for( i=0 ; i<vsize ; i++ ) { ... }
}
void store_graph (char *filename)
{
  FILE *fp = fopen(filename, "w");
  if( fp != NULL ) {
    print_graph( fp, "" );
    fclose(fp);
  }
}
```

```
void load_graph ( char *filename)
{
  int i, j, val, n;
  char   str[80];
  FILE *fp = fopen(filename, "r");
  if( fp != NULL ) {
    init_graph();
    fscanf(fp, "%d", &n);

    for(i=0 ; i<n ; i++ ) {
        ...
    }
    fclose(fp);
  }
}
```

# 11.3 그래프 표현 방법 2 : 인접 리스트

- adjacency list
  - 각 정점이 연결 리스트를 가짐
  - 인접한 정점들을 연결리스트로 표현









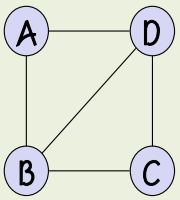
# 인접 리스트를 이용한 그래프 표현

- 그래프 데이터와 기본 연산

```
typedef struct GraphNode {
        int id;                     // 정점의 id
        struct GraphNode* link;     // 다음 노드의 포인터
} GNode;
typedef char VtxData;               // 그래프 정점에 저장할 데이터의 자료형


int     vsize;                      // 정점의 개수
VtxData vdata[MAX_VTXS];            // 정점에 저장할 데이터 배열
GNode*  adj[MAX_VTXS];              // 각 정점의 인접 리스트


int is_empty_graph() { return (vsize == 0); }
int is_full_graph() { return (vsize >= MAX_VTXS); }
void init_graph( ) {…}
void reset_graph() {…}
void insert_vertex( char name ) {…}
void insert_edge( int u, int v ) {…}
```

# 전체 프로그램

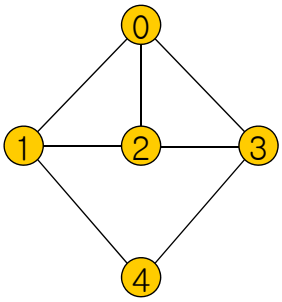
```
void main()
{
        load_graph( "graph.txt");
        print_graph("그래프(인접리스트)\n");
}
```




```
선택 C:₩WINDOWS₩system32₩cmd.exe
그래프(인접리스트)
4
A       D   B
B       D   C   A
C       D   B
D       C   B   A
계속하려면 아무 키나 누르십시오 . . .
```



- 아래 그래프를 인접 행렬과 인접 리스트로 각각 표현하라.



- 인접 행렬이 아래와 같다면 여기에 대응되는 인접 리스트를 그려라.

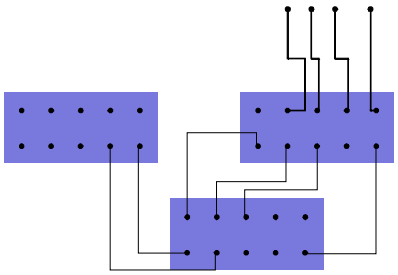
| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |

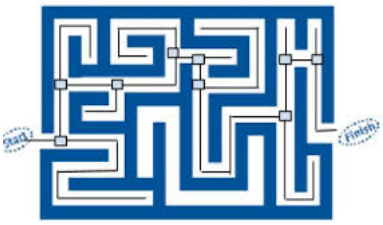


- 정점의 개수를 n, 간선의 개수가 e인 무방향 그래프를 인접 리스트로 표현하였을 경우, 인접 리스트상의 총 노드의 개수는?

# 11.4 그래프 탐색

- 그래프의 가장 기본적인 연산
  - 시작 정점부터 차례대로 모든 정점들을 한 번씩 방문
  - 많은 문제들이 단순히 탐색만으로 해결됨
    - 도로망 예: 특정 도시에서 다른 도시로 갈 수 있는지 여부
    - 전자회로 예: 특정 단자와 다른 단자의 연결 여부



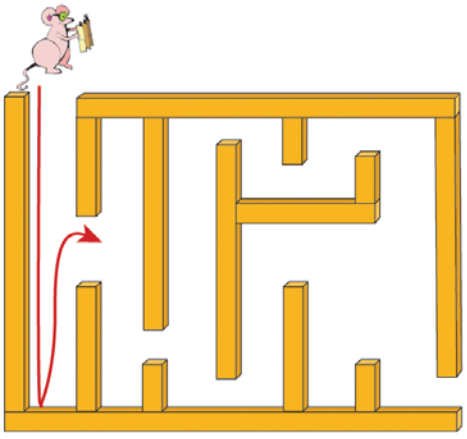


  - 깊이 우선 탐색 (DFS)
  - 너비 우선 탐색 (BFS)



# 깊이 우선 탐색(DFS)

- **깊이 우선 탐색** (DFS: depth-first search)
  - 한 방향으로 갈 수 있을 때까지 가다가 더 이상 갈 수 없게 되면 가장 가까운 갈림길로 돌아와서 이 곳으로부터 다른 방향으로 다시 탐색 진행
  - 되돌아가기 위해서는 스택 필요(순환함수 호출로 묵시적인 스택 이용 가능)



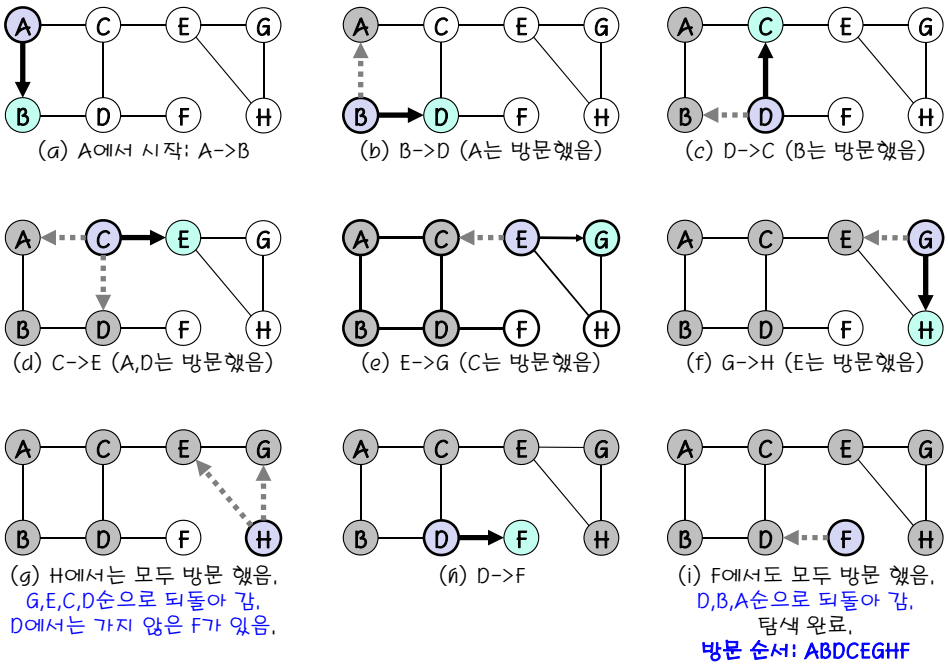
# 깊이 우선 탐색 (DFS)

- DFS: depth-first search
  - 한 방향으로 갈 수 있을 때까지 가다가 더 이상 갈 수 없게 되면 가장 가까운 갈림길로 돌아와서 이 곳으로부터 다른 방향으로 다시 탐색 진행
  - 되돌아가기 위해서는 스택 필요
    - 순환함수 호출로 묵시적인 스택 이용

```
depthFirstSearch(v)

     v를 방문되었다고 표시;
     for all u ∈ (v에 인접한 정점) do
        if (u가 아직 방문되지 않았으면)
          then depthFirstSearch(u)
```



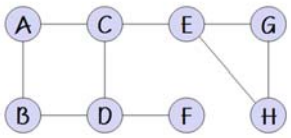
# DFS 알고리즘

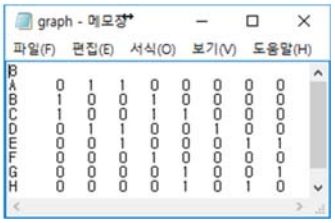


# DFS 구현(인접 행렬)

```
int visited[MAX_VTXS];


void reset_visited()
{
  int i;
  for( i=0 ; i<vsize ; i++ )
    visited[i] = 0;
}
```

```
void DFS( int v)
{
  int w;
  visited[v] = 1;
  printf("%c ", vdata[v]);
  for( w=0 ; w<vsize ; w++)
    if( adj[v][w]!=0 && visited[w]==0)
        DFS( w );
}
```





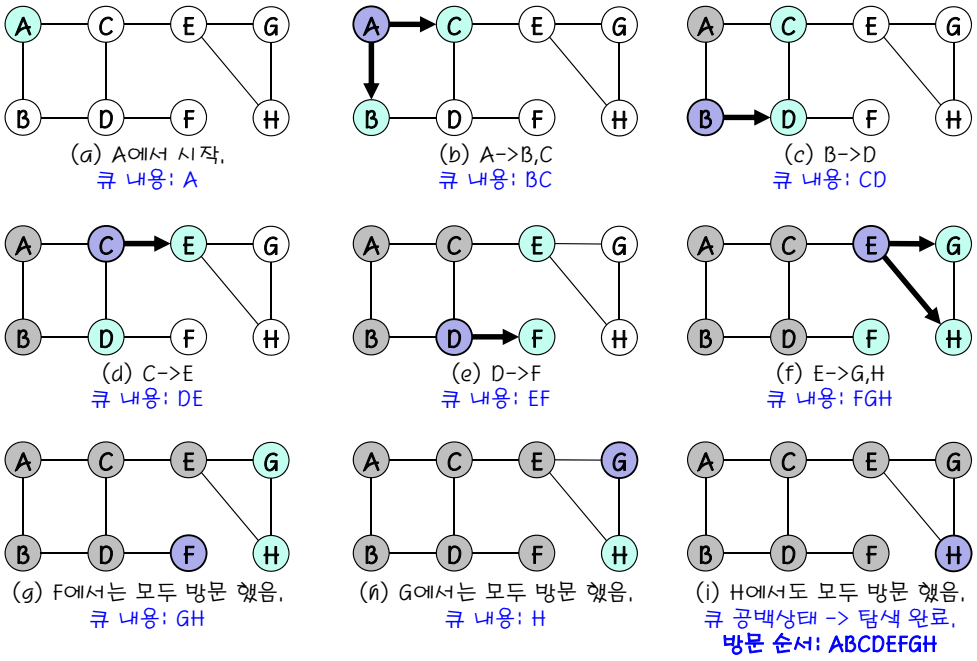


# 너비 우선 탐색 (BFS)

- BFS: breadth-first search
  - 시작 정점으로부터 가까운 정점을 먼저 방문하고 멀리 떨어져 있는 정점을 나중에 방문하는 순회 방법
  - 큐를 사용하여 구현됨

```
breadthFirstSearch(v)

    v를 방문되었다고 표시;
    큐 Q에 정점 v를 삽입;
    while (not is_empty(Q)) do
       Q에서 정점 w를 삭제;
       for all u ∈ (w에 인접한 정점) do
           if (u가 아직 방문되지 않았으면)
               then     u를 큐에 삽입;
                        u를 방문되었다고 표시;
```

# BFS 알고리즘



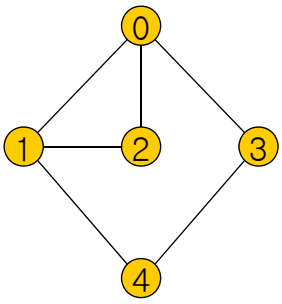



# BFS 구현 (인접 리스트)

```
void BFS( int v)
{
  GNode *w;
  init_queue( );
  visited[v] = 1;
  printf("%c ", vdata[v]);
  enqueue( v );
  while(!is_empty()){
    v = dequeue();

    for( w=adj[v] ; w!=NULL ; w=w->link ) {
        if(!visited[w->id]){
          visited[w->id] = 1;
          printf("%c ", vdata[w->id]);
          enqueue(w->id);
        }
    }
  }
}
```

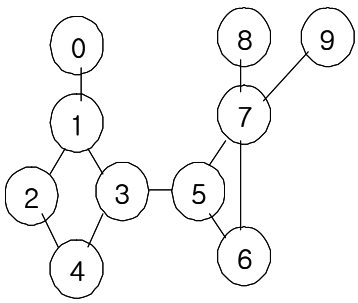
```
C:₩WINDOWS₩system32₩cmd.exe
BFS ==> A C B E D H G F
계속하려면 아무 키나 누르십시오 . . .
```

- 아래 그래프에서 0번 정점을 시작 정점으로 깊이 우선 탐색할 때 방문되는 정점을 순서대로 나열하라.





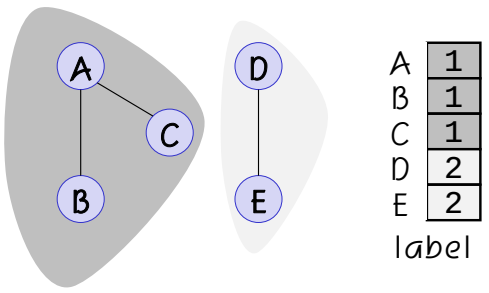

- 다음 그래프에 대하여 답하라. 인접 행렬로 표현되었다고 가정한다.




  - 정점 3에서 출발하여 깊이 우선 탐색했을 경우의 방문 순서
  - 정점 6에서 출발하여 깊이 우선 탐색했을 경우의 방문 순서
  - 정점 3에서 출발하여 너비 우선 탐색했을 경우의 방문 순서
  - 정점 6에서 출발하여 너비 우선 탐색했을 경우의 방문 순서

# 11.5 연결 성분 검사

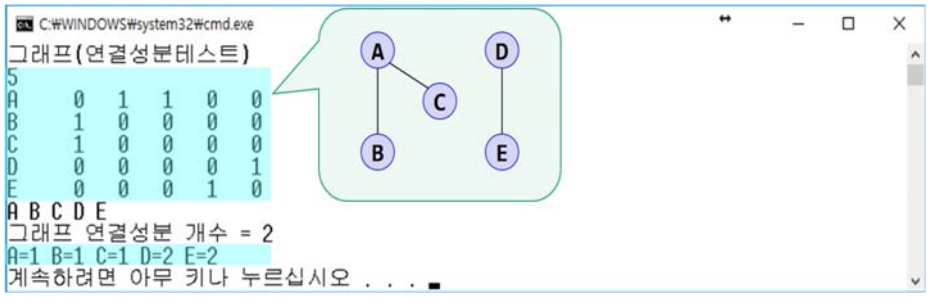
- 최대로 연결된 부분 그래프들을 구함
  - DFS 또는 BFS 반복 이용
    - label[i]에는 i번째 정점의 색이 저장됨





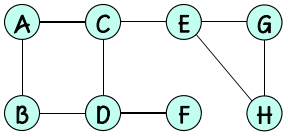

```
int      visited[MAX_VTXS];
int      label[MAX_VTXS];
void labelDFS( int v, int color)
{
  int w;
  visited[v] = 1;
  label[v] = color;

  printf("%c ", vdata[v]);
  for( w=0 ; w<vsize ; w++)
    if( adj[v][w]!=0 && visited[w]==0)
         labelDFS( w, color );
}
```

```
void findConnectedComponent()
{
  int i, count = 0;

  for( i=0 ; i<vsize ; i++ )
    visited[i] = 0;
  for(i=0; i<vsize ; i++)
    if( visited[i]==0)
         labelDFS(i, ++count);
  printf("\n연결성분 개수 = %d\n", count);
  for( i=0 ; i<vsize ; i++ )
    printf("%c=%d ", vdata[i], label[i]);
}
```

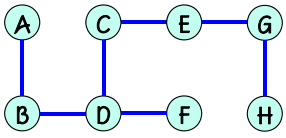


# 11.6 신장 트리(spanning tree)

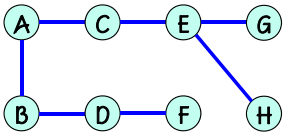
- Spanning Tree
  - 모든 정점들이 연결되어야 하고 사이클을 포함하면 안됨
  - 신장 트리는 n-1개의 간선을 가짐
  - 최소의 링크를 사용하는 네트워크 구축 시 사용
    - 통신망, 도로망, 유통망 등
  - 다양한 신장 트리 가능



(a) 연결 그래프



(b) 신장 트리의 예
(DFS에서의 간선)



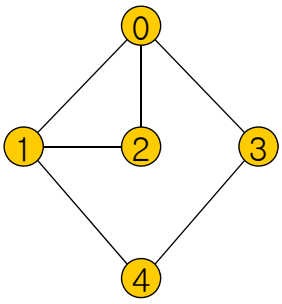
(c) 신장 트리의 예
(BFS에서의 간선)



# 신장 트리(spanning tree)

```
spanningTreeByDFS(v)

      v를 방문되었다고 표시;
      for all u ∈ (v에 인접한 정점) do
          if (u가 아직 방문되지 않았으면)
             then   (v,u)를 신장 트리 간선이라고 표시;
                    spanningTreeByDFS(u);
```

- 아래 그래프의 신장 트리를 3개 이상 보여라.

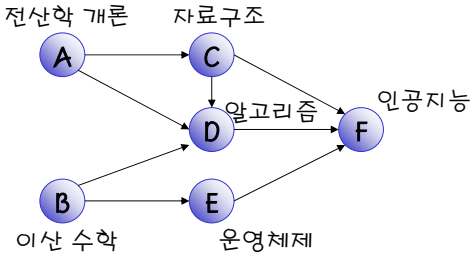





# 11.7 위상 정렬

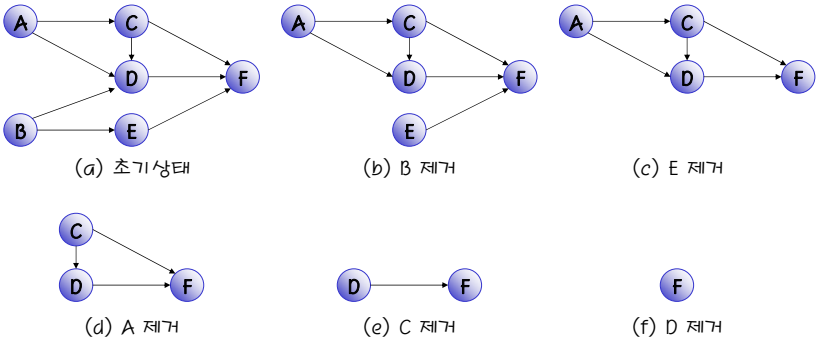
- Topological sort
  - 방향 그래프에 대해 정점들의 선행 순서를 위배하지 않으면서 모든 정점을 나열하는 것
    - 여러 방법이 가능

| *과목번호* | *과목명* | *선수과목* |
|---|---|---|
| *A* | *컴퓨터개론* | *없음* |
| *B* | *이산수학* | *없음* |
| *C* | *자료구조* | *A* |
| *D* | *알고리즘* | *A, B, C* |
| *E* | *운영체제* | *B* |
| *F* | *인공지능* | *C, D, E* |




<A,B,C,D,E,F>
<B,A,C,D,E,F>

# 위상 정렬 알고리즘




```
topoSort()

for i←0 to n-1 do
    if 모든 정점이 선행 정점을 가지면
        then 사이클이 존재하고 위상 정렬 불가;
   선행 정점을 가지지 않는 정점 v 선택;
   v를 출력;
   v와 v에서 나온 모든 간선들을 그래프에서 삭제;
```



- 아래의 그래프에 위상 정렬을 적용하는 과정을 보이고, 이에 해당하는 2개 이상의 위상 순서를 보여라.