

10장 그래프

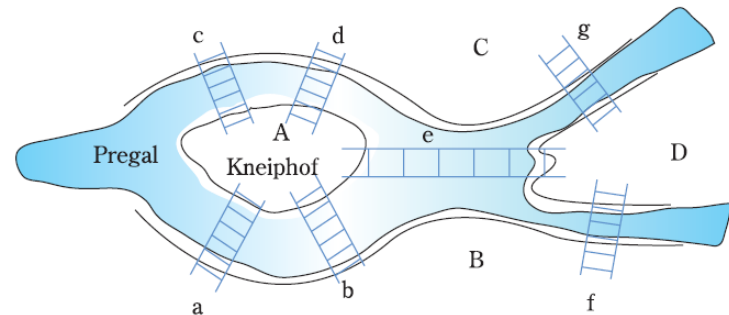


그래프 (graph)

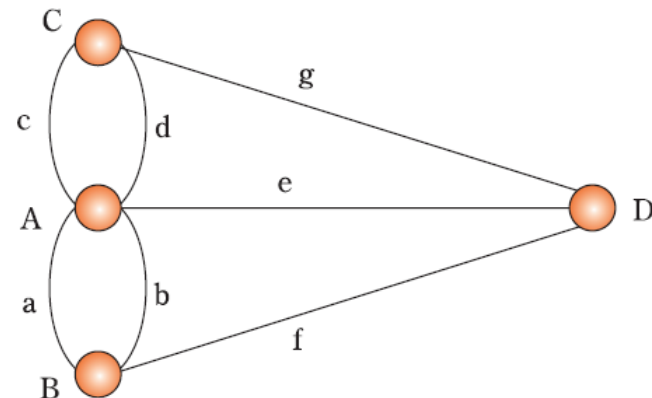
- 연결되어 있는 객체 간의 관계를 표현하는 자료구조
 - ▣ (예) 우리가 배운 트리(tree)도 그래프의 특수한 경우임
 - ▣ (예) 전기회로의 소자 간 연결 상태
 - ▣ (예) 지도에서 도시들의 연결 상태



- 1800년대 오일러에 의하여 창안
- 오일러 문제
 - ▣ 모든 다리를 한번만 건너서 처음 출발했던 장소로 돌아오는 문제
- A,B,C,D 지역의 연결 관계 표현
 - ▣ 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



(a) 모든 다리를 한 번만 건너 돌아오는 경로 문제



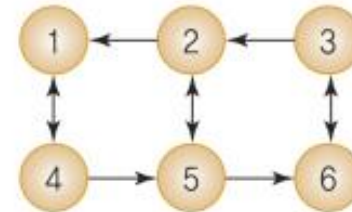
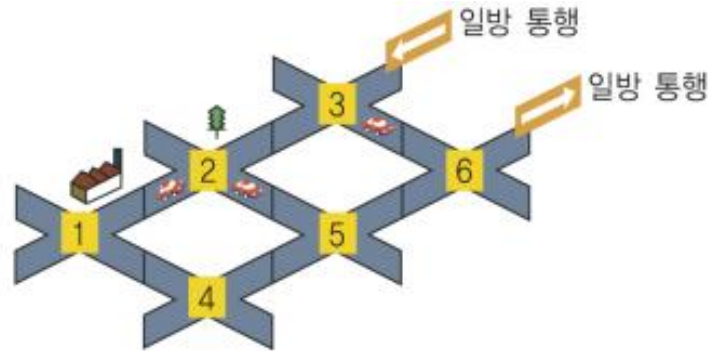
(b) 문제 (a)의 그래프 표현



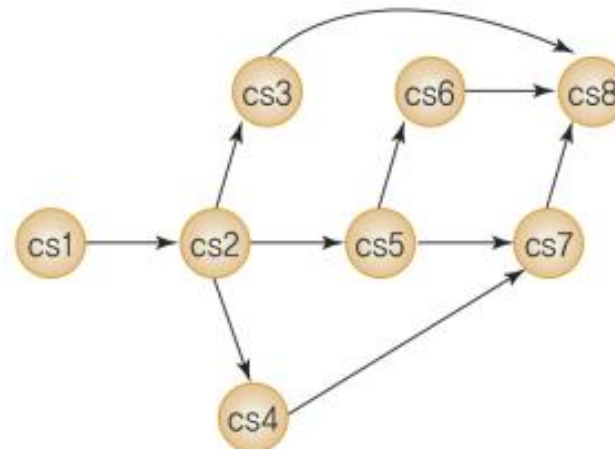


그래프로 표현하는 것들

□ 도로망



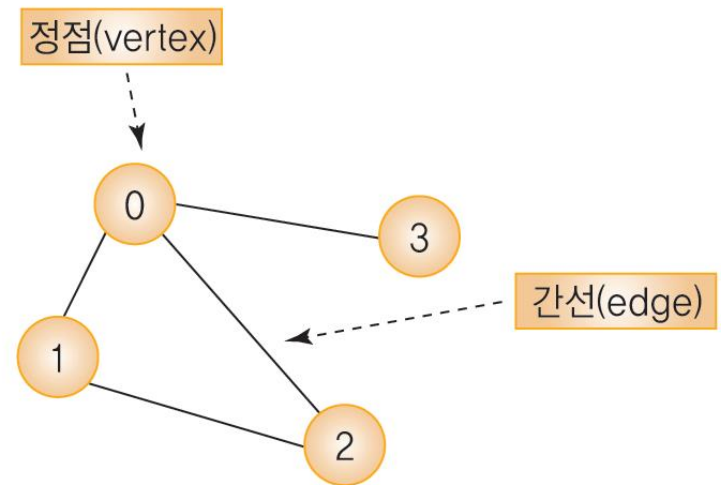
□ 선수과목 관계





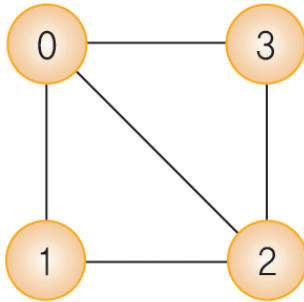
그래프 정의

- 그래프 G 는 (V, E) 로 표시
- 정점(vertices)
 - ▣ 여러 가지 특성을 가질 수 있는 객체 의미
 - ▣ $V(G)$: 그래프 G 의 정점들의 집합
 - ▣ 노드(node)라고도 불림
- 간선(edge)
 - ▣ 정점들 간의 관계 의미
 - ▣ $E(G)$: 그래프 G 의 간선들의 집합
 - ▣ 링크(link)라고도 불림

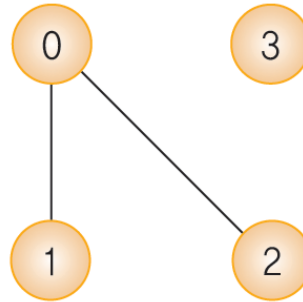




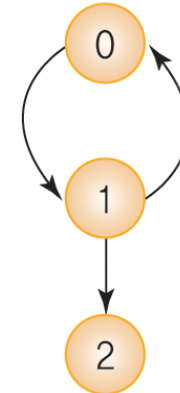
그래프 표현의 예



G1



G2



G3

$V(G1) = \{0, 1, 2, 3\}, \quad E(G1) = \{(0, 1), (0, 2), (0, 3), (1, 2), (2, 3)\}$

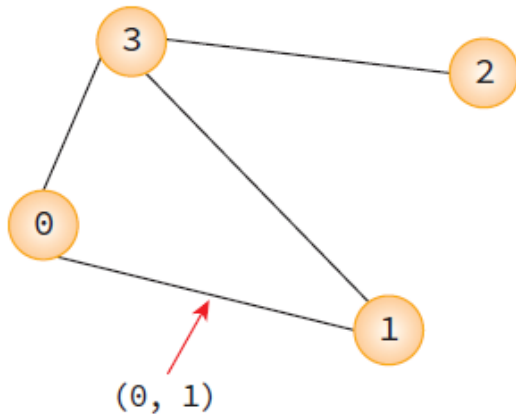
$V(G2) = \{0, 1, 2, 3\}, \quad E(G3) = \{(0, 1), (0, 2)\}$

$V(G2) = \{0, 1, 2\}, \quad E(G2) = \{<0, 1>, <1, 0>, <1, 2>\}$

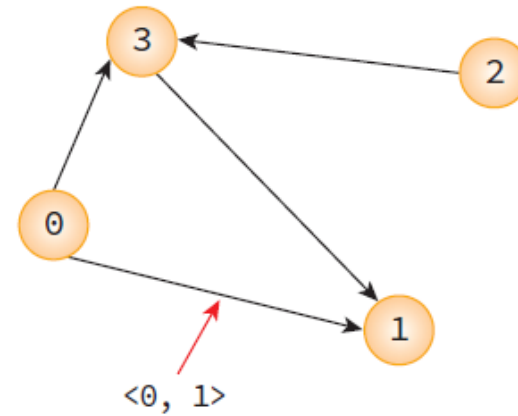




그래프의 종류



$V(G_1) = \{0, 1, 2, 3\},$
 $E(G_1) = \{(0, 1), (0, 3), (1, 3), (2, 3)\}$



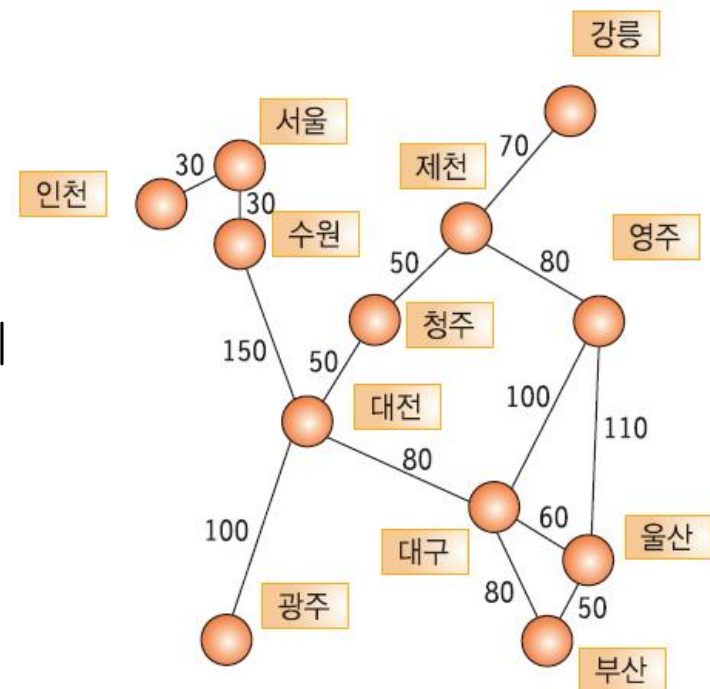
$V(G_2) = \{0, 1, 2, 3\},$
 $E(G_2) = \{\langle 0, 1 \rangle, \langle 0, 3 \rangle, \langle 3, 1 \rangle, \langle 2, 3 \rangle\}$

[그림 10-9] 무방향 그래프와 방향 그래프



- 가중치 그래프(weighted graph)는 네트워크(network)라고도 함
- 간선에 비용(cost)이나 가중치(weight)가 할당된 그래프

- 네트워크 예
 - ▣ 정점 : 각 도시를 의미
 - ▣ 간선 : 도시를 연결하는 도로의
 - ▣ 가중치 : 도로의 길이

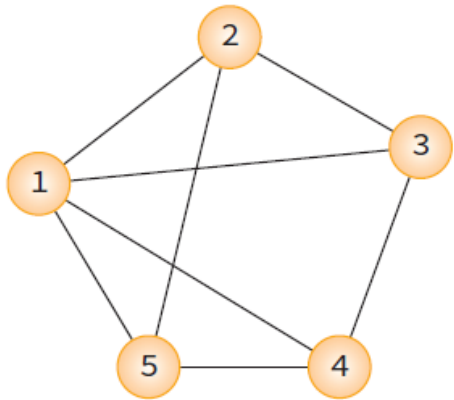




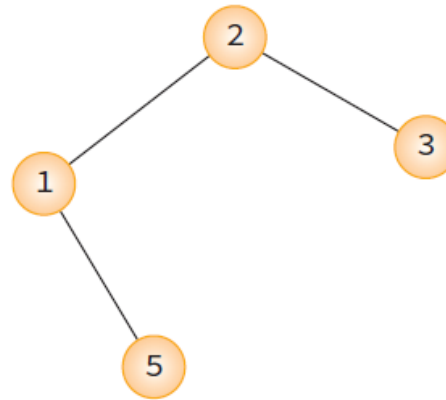
부분 그래프(subgraph)

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

$$V(S) \subseteq V(G)$$
$$E(S) \subseteq E(G)$$



(a) 그래프

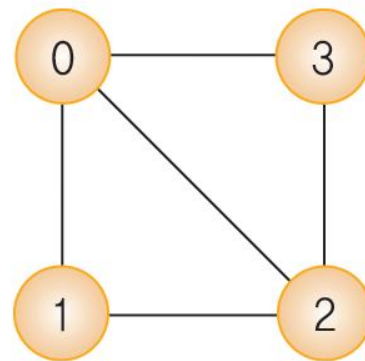


(b) 부분 그래프





- 인접 정점(adjacent vertex)
 - ▣ 하나의 정점에서 간선에 의해 직접 연결된 정점
 - ▣ G1에서 정점 0의 인접 정점: 정점 1, 정점 2, 정점 3
- 무방향 그래프의 차수(degree)
 - ▣ 하나의 정점에 연결된 다른 정점의 수
 - ▣ G1에서 정점 0의 차수: 3



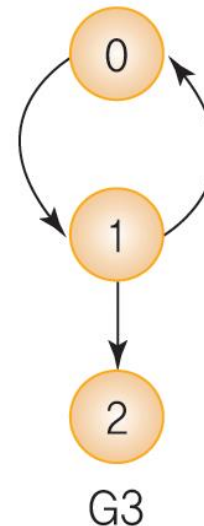
G1





□ 방향 그래프의 차수(degree)

- 진입 차수(in-degree) : 외부에서 오는 간선의 수
- 진출 차수(out-degree) : 외부로 향하는 간선의 수
- G3에서 정점 1의 차수: 내차수 1, 외차수 2
- 방향 그래프의 모든 진입(진출) 차수의 합은 간선의 수
 - G3의 진입 차수의 합: 3
 - G3의 진출 차수의 합: 3
 - G3의 간선 합: 3





그래프의 경로(path)

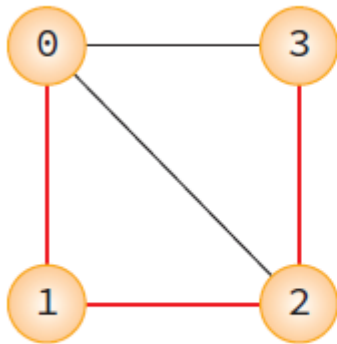
- 무방향 그래프의 정점 s 로부터 정점 e 까지의 경로
 - ▣ 정점의 나열 $s, v_1, v_2, \dots, v_k, e$
 - ▣ 나열된 정점들 간에 반드시 간선 $(s, v_1), (v_1, v_2), \dots, (v_k, e)$ 존재
- 단순 경로(simple path)
 - ▣ 경로 중에서 반복되는 간선이 없는 경로
- 사이클(cycle)
 - ▣ 단순 경로의 시작 정점과 종료 정점이 동일한 경로



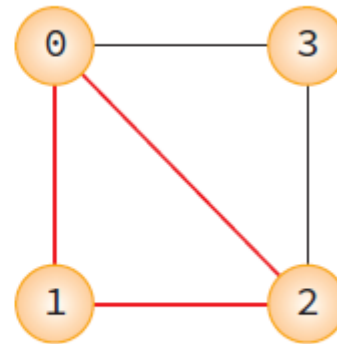


그래프의 경로(path)

- G1의 0, 1, 2, 3은 경로지만 0, 1, 3, 2는 경로 아님
- G1의 1, 0, 2, 3은 단순경로이지만 1, 0, 2, 0은 단순경로 아님
- G1의 0, 1, 2, 0과 G3의 0, 1, 0은 사이클



단순경로: 0, 1, 2, 3



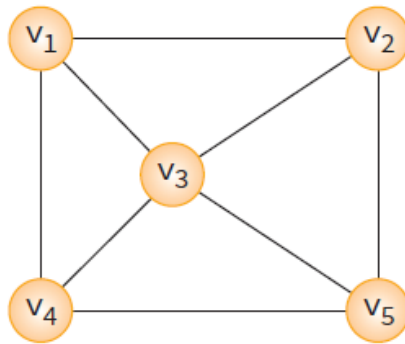
사이클: 0, 1, 2, 0



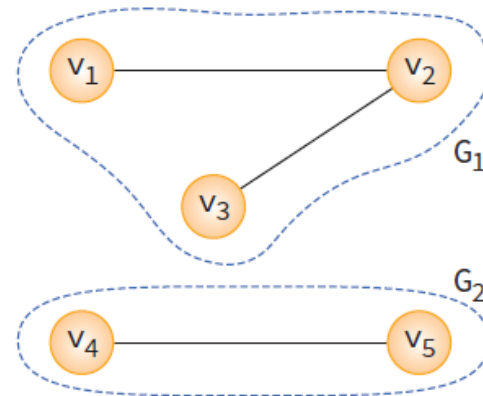


그래프의 연결정도

- 연결 그래프(connected graph)
 - ▣ 무방향 그래프 G 에 있는 모든 정점쌍에 대하여 항상 경로 존재
 - ▣ G_2 는 비연결 그래프임



(a)



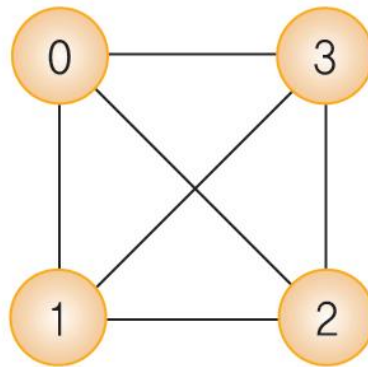
(b)





그래프의 연결정도

- 완전 그래프(complete graph)
 - ▣ 모든 정점이 연결되어 있는 그래프
 - ▣ n개의 정점을 가진 무방향 완전그래프의 간선의 수: $n \times (n-1) / 2$
 - ▣ $n=4$, 간선의 수 = $(4 \times 3) / 2 = 6$



·객체: 정점의 집합과 간선의 집합

·연산:

- `create_graph()` ::= 그래프를 생성한다.
- `init(g)` ::= 그래프 `g`를 초기화한다.
- `insert_vertex(g,v)` ::= 그래프 `g`에 정점 `v`를 삽입한다.
- `insert_edge(g,u,v)` ::= 그래프 `g`에 간선 `(u,v)`를 삽입한다.
- `delete_vertex(g,v)` ::= 그래프 `g`의 정점 `v`를 삭제한다.
- `delete_edge(g,u,v)` ::= 그래프 `g`의 간선 `(u,v)`를 삭제한다.
- `is_empty(g)` ::= 그래프 `g`가 공백 상태인지 확인한다.
- `adjacent(v)` ::= 정점 `v`에 인접한 정점들의 리스트를 반환한다.
- `destroy_graph(g)` ::= 그래프 `g`를 제거한다.





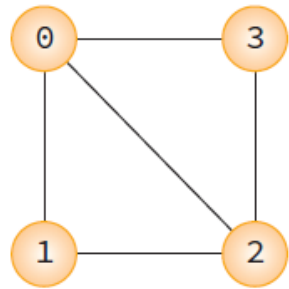
그래프 표현 방법

- 인접행렬 (adjacent matrix) 방법
- 인접 리스트(adjacent list) 방법



□ 인접행렬 (adjacent matrix) 방법

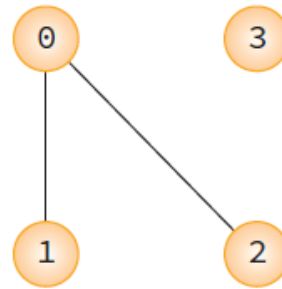
if(간선 (i, j)가 그래프에 존재) $M[i][j] = 1$,
그렇지 않으면 $M[i][j] = 0$.



정점 0의 차수
= 0+1+1+1=3

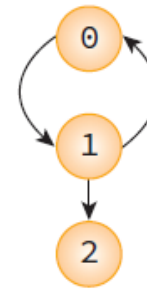
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	1
3	1	0	1	0

(a)



	0	1	2	3
0	0	1	1	0
1	1	0	0	0
2	1	0	0	0
3	0	0	0	0

(b)

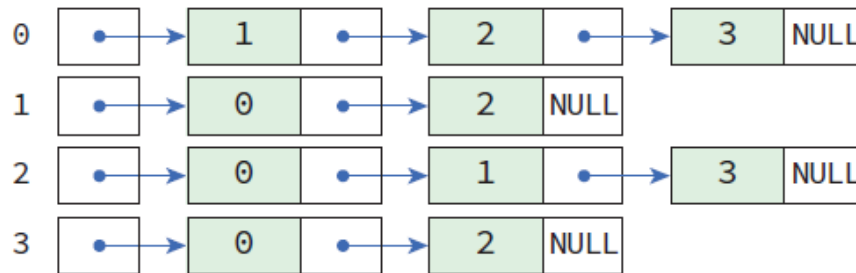
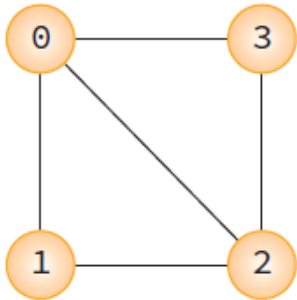


	0	1	2
0	0	1	0
1	1	0	1
2	0	0	0

(c)



- 인접리스트 (adjacency list) 방법
 - ▣ 각 정점에 인접한 정점들을 연결리스트로 표현





이제 행렬 구현

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 50
typedef struct GraphType {
    int n;        // 정점의 개수
    int adj_mat[MAX_VERTICES][MAX_VERTICES];
} GraphType;

// 그래프 초기화
void init(GraphType* g)
{
    int r, c;
    g->n = 0;
    for (r = 0; r < MAX_VERTICES; r++)
        for (c = 0; c < MAX_VERTICES; c++)
            g->adj_mat[r][c] = 0;
}
```





이제 행려 구현

// 정점 삽입 연산

```
void insert_vertex(GraphType* g, int v)
{
    if (((g->n) + 1) > MAX_VERTICES) {
        fprintf(stderr, "그래프: 정점의 개수 초과");
        return;
    }
    g->n++;
}
```

// 간선 삽입 연산

```
void insert_edge(GraphType* g, int start, int end)
{
    if (start >= g->n || end >= g->n) {
        fprintf(stderr, "그래프: 정점 번호 오류");
        return;
    }
    g->adj_mat[start][end] = 1;
    g->adj_mat[end][start] = 1;
}
```



이제 행렬 구현

그래프 행렬 구현

```
void main()
{
    GraphType *g;
    g = (GraphType *)malloc(sizeof(GraphType));
    init(g);
    for(int i=0;i<4;i++)

        insert_vertex(g, i);
        insert_edge(g, 0, 1);
        insert_edge(g, 0, 2);
        insert_edge(g, 0, 3);
        insert_edge(g, 1, 2);
        insert_edge(g, 2, 3);
        print_adj_mat(g);

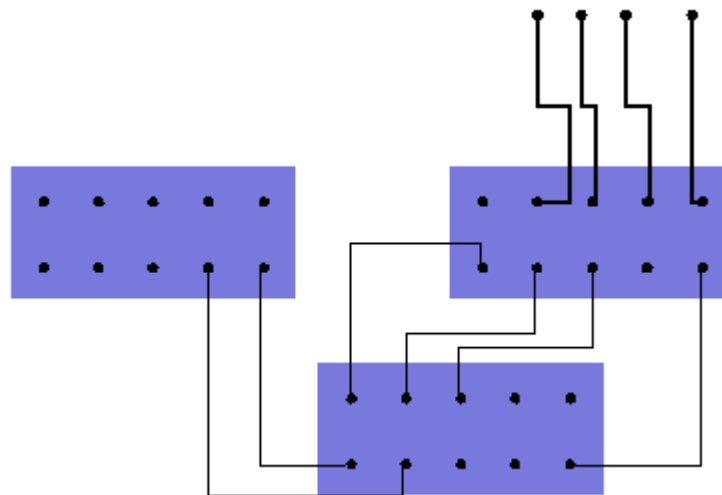
    free(g);
}
```



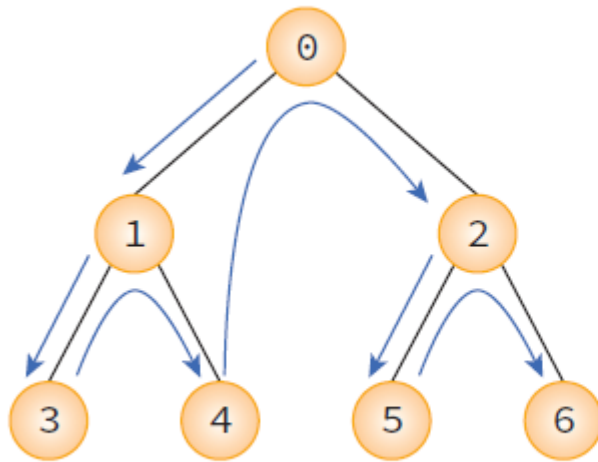
- 하나의 정점으로부터 시작하여 차례대로 모든 정점들을 한번씩 방문
- 많은 문제들이 단순히 그래프의 노드를 탐색하는 것으로 해결

(예) 도로망에서 특정 도시에서 다른 도시로 갈 수 있는지 여부

(예) 전자회로에서 특정 단자와 다른 단자가 서로 연결되어 있는지 여부

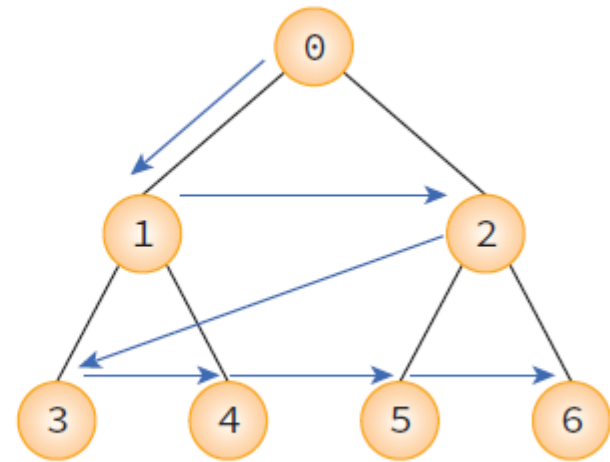


- 깊이 우선 탐색(DFS: depth first search)
- 너비 우선 탐색(BFS: breath first search)



깊이 우선 탐색

방문 기록을 스택에 저장
재귀로 구현



너비 우선 탐색

- * 트리의 Level Order 순회와 비슷
- * 그래프에는 루트 노드가 없으므로 아무 노드나 잡고 시작해도 됨
- * 방문 기록을 큐에 저장

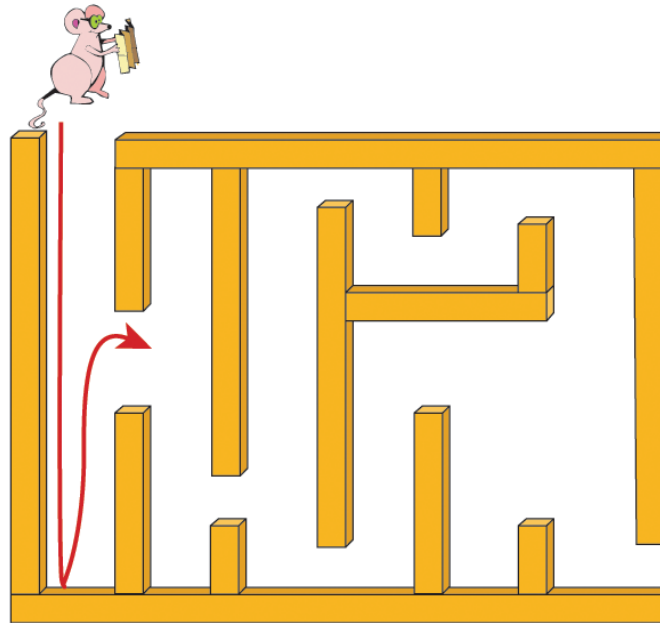




깊이 우선 탐색 (DFS)

□ 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)





DFS 알고리즘

depth_first_search(v):

 v를 방문되었다고 표시;

 for all $u \in$ (v에 인접한 정점) do

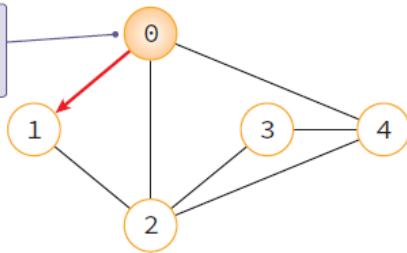
 if (u가 아직 방문되지 않았으면)

 then depth_first_search(u)



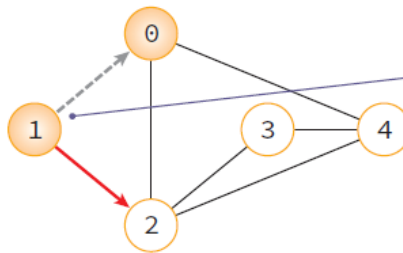
DFS 알고리즘

정점 0을 시작 정점으로 깊이 우선 탐색을 시작한다.



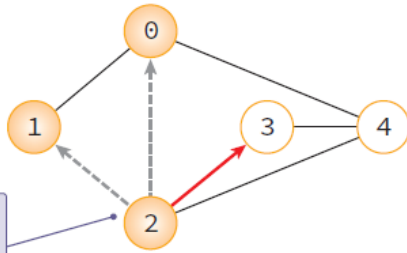
(a)

정점 1을 시작 정점으로 깊이 우선 탐색을 시작한다.



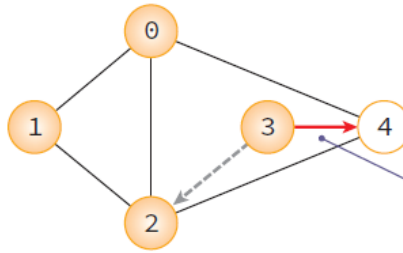
(b)

정점 2을 시작 정점으로 깊이 우선 탐색을 시작한다.



(c)

정점 3을 시작 정점으로 깊이 우선 탐색을 시작한다.

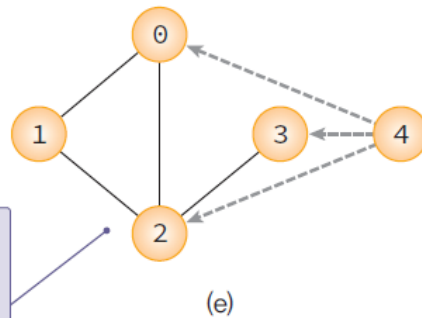


(d)

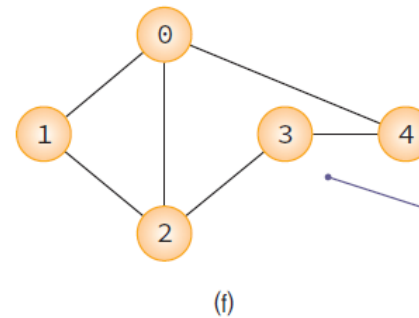




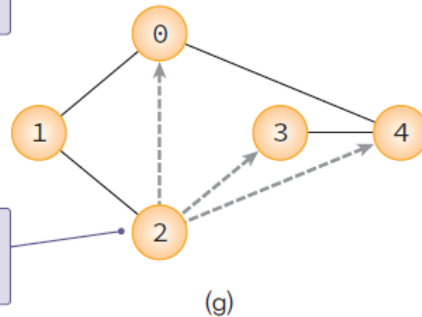
DFS 알고리즘



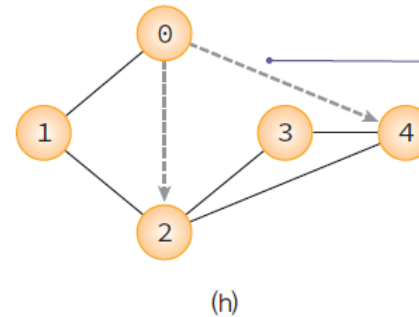
정점 4를 시작 정점으로 깊이 우선 탐색을 시작한다. 전부 이미 방문한 정점들이라 노드 3으로 되돌아간다 (backtracking).



정점 3에서도 더 이상 방문할 정점이 없어서 정점 순환 호출을 되돌아간다.



정점 2과 정점 1에서도 순환 호출을 되돌아간다.



정점 0에서 더 이상 방문할 정점이 없어서 탐색이 종료된다.





DFS 프로그램

// 인접 행렬로 표현된 그래프에 대한 깊이 우선 탐색

```
void dfs_mat(GraphType* g, int v)
```

```
{
```

```
    int w;
```

```
    visited[v] = TRUE;           // 정점 v의 방문 표시
```

```
    printf("정점 %d -> ", v);    // 방문한 정점 출력
```

```
    for (w = 0; w < g->n; w++)    // 인접 정점 탐색
```

```
        if (g->adj_mat[v][w] && !visited[w])
```

```
            dfs_mat(g, w);       // 정점 w에서 DFS 새로 시작
```

```
}
```





DFS 프로그램

```
int main(void)
{
    GraphType *g;
    g = (GraphType *)malloc(sizeof(GraphType));
    init(g);
    for (int i = 0; i<4; i++)
        insert_vertex(g, i);
    insert_edge(g, 0, 1);
    insert_edge(g, 0, 2);
    insert_edge(g, 0, 3);
    insert_edge(g, 1, 2);
    insert_edge(g, 2, 3);

    printf("깊이 우선 탐색\n");
    dfs_mat(g, 0);
    printf("\n");
    free(g);
    return 0;
}
```





깊이 우선 탐색

정점 0 -> 정점 1 -> 정점 2 -> 정점 3 ->





DFS 프로그램(인접 리스트 버전)

```
int visited[MAX_VERTICES];

// 인접 리스트로 표현된 그래프에 대한 깊이 우선 탐색
void dfs_list(GraphType* g, int v)
{
    GraphNode* w;
    visited[v] = TRUE;           // 정점 v의 방문 표시
    printf("정점 %d -> ", v);   // 방문한 정점 출력
    for (w = g->adj_list[v]; w; w = w->link) // 인접 정점 탐색
        if (!visited[w->vertex])
            dfs_list(g, w->vertex); // 정점 w에서 DFS 새로 시작
}
```





DFS 프로그램(명시적 스택 사용)

DFS-iterative(G, v):

스택 S를 생성한다.

S.push(v)

while (not is_empty(S)) do

 v = S.pop()

 if (v가 방문되지 않았으면)

 v를 방문되었다고 표시

 for all u \in (v에 인접한 정점) do

 if (u가 아직 방문되지 않았으면)

 S.push(u)





기이우선탐색의 분석

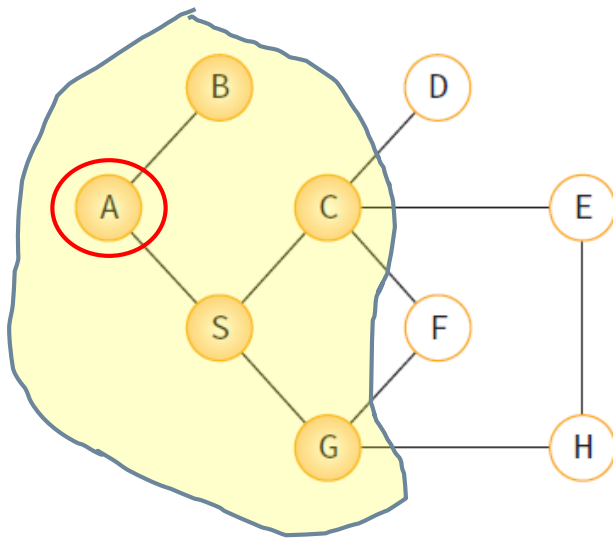
- 인접 행렬 $\rightarrow O(n^2)$
- 인접 리스트 $\rightarrow O(n+e)$





너비우선 탐색(BFS)

- 너비 우선 탐색(BFS: breadth-first search)
 - ▣ 시작 정점으로부터 가까운 정점을 먼저 방문하고 멀리 떨어져 있는 정점을 나중에 방문하는 순회 방법
 - ▣ 큐를 사용하여 구현됨



BFS는 시작 정점 A에서 가까운 정점들을 먼저 방문하는 기법이다.





BFS 알고리즘

```
breadth_first_search(v):
```

```
    v를 방문되었다고 표시;
```

```
    큐 Q에 정점 v를 삽입;
```

```
    while (Q가 공백이 아니면) do
```

```
        Q에서 정점 w를 삭제;
```

```
        for all  $u \in$  (w에 인접한 정점) do
```

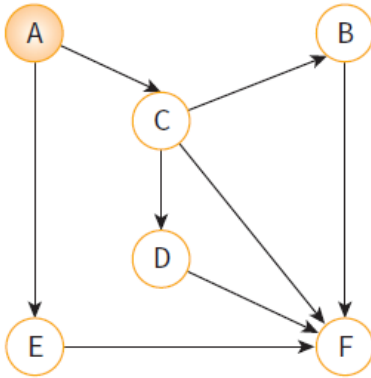
```
            if (u가 아직 방문되지 않았으면)
```

```
                then    u를 큐에 삽입;  
                        u를 방문되었다고 표시;
```

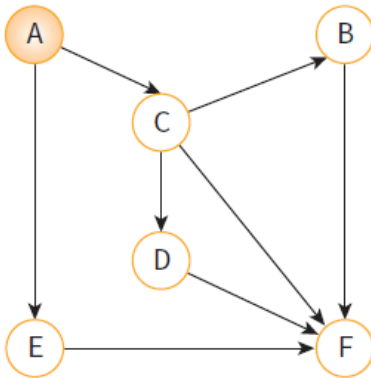




너비우선 탐색(BFS)



처음에는 시작 정점인 A를 큐에 추가한다.

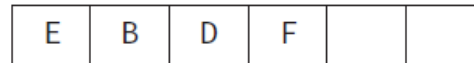
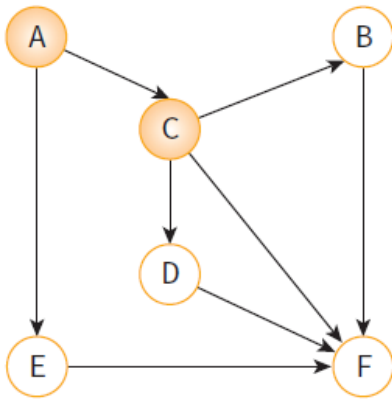


큐에서 A를 꺼내서 방문하고 정점 A의 인접 정점들을 큐에 추가한다.

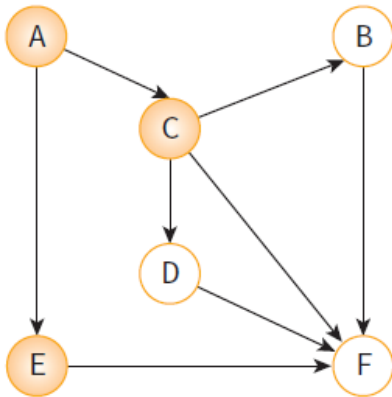




너비우선 탐색(BFS)



큐에서 C를 꺼내서 방문하고 C의 인접 정점들을 큐에 추가한다.



큐에서 E를 꺼내서 방문하고 인접 정점인 F는 이미 큐에 들어 있으므로 추가하지 않는다.





BFS 프로그램(인접행렬)

```
void bfs_mat(GraphType* g, int v)
{
    int w;
    QueueType q;

    queue_init(&q);    // 큐 초기화
    visited[v] = TRUE;    // 정점 v 방문 표시
    printf("%d 방문 -> ", v);
    enqueue(&q, v);    // 시작 정점을 큐에 저장
    while (!is_empty(&q)) {
        v = dequeue(&q);    // 큐에 정점 추출
        for (w = 0; w < g->n; w++)    // 인접 정점 탐색
            if (g->adj_mat[v][w] && !visited[w]) {
                visited[w] = TRUE;    // 방문 표시
                printf("%d 방문 -> ", w);
                enqueue(&q, w);    // 방문한 정점을 큐에 저장
            }
    }
}
```





BFS 프로그램(이제 해려)

```
int main(void)
{
    GraphType *g;
    g = (GraphType *)malloc(sizeof(GraphType));
    graph_init(g);
    for (int i = 0; i<6; i++)
        insert_vertex(g, i);
    insert_edge(g, 0, 2);
    insert_edge(g, 2, 1);
    insert_edge(g, 2, 3);
    insert_edge(g, 0, 4);
    insert_edge(g, 4, 5);
    insert_edge(g, 1, 5);

    printf("너비 우선 탐색\n");
    bfs_mat(g, 0);
    printf("\n");
    free(g);
    return 0;
}
```





너비 우선 탐색

0 방문 -> 2 방문 -> 4 방문 -> 1 방문 -> 3 방문 -> 5 방문 ->





BFS 프로그램(인접리스트)

```
void bfs_list(GraphType* g, int v)
{
    GraphNode* w;
    QueueType q;

    init(&q); // 큐 초기화
    visited[v] = TRUE; // 정점 v 방문 표시
    printf("%d 방문 -> ", v);
    enqueue(&q, v); // 시작정점을 큐에 저장
    while (!is_empty(&q)) {
        v = dequeue(&q); // 큐에 저장된 정점 선택
        for (w = g->adj_list[v]; w; w = w->link) //인접 정점 탐색
            if (!visited[w->vertex]) { // 미방문 정점 탐색
                visited[w->vertex] = TRUE; // 방문 표시
                printf("%d 방문 -> ", w->vertex);
                enqueue(&q, w->vertex); //정점을 큐에
                삽입
            }
    }
}
```





너비우선탐색의 분석

- 인접 행렬 $\rightarrow O(n^2)$
- 인접 리스트 $\rightarrow O(n+e)$

