12주차 1차시 그래프의 구조와 구현

【학습목표】

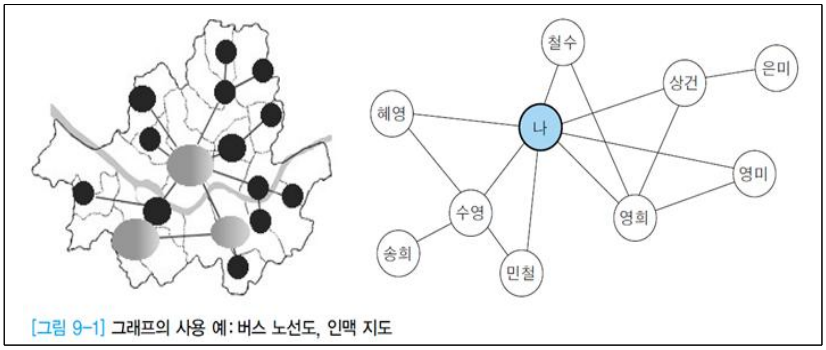
1. 그래프를 구현하기 위해서 표현하는 방법을 설명할 수 있다.
2. 인접행렬을 예를 들어 설명할 수 있다.

학습내용1 : 그래프와 종류

1. 그래프

* 선형 자료구조나 트리 자료구조로 표현하기 어려운 多:多 관계를 가지는 원소들을 표현하기 위한 자료구조

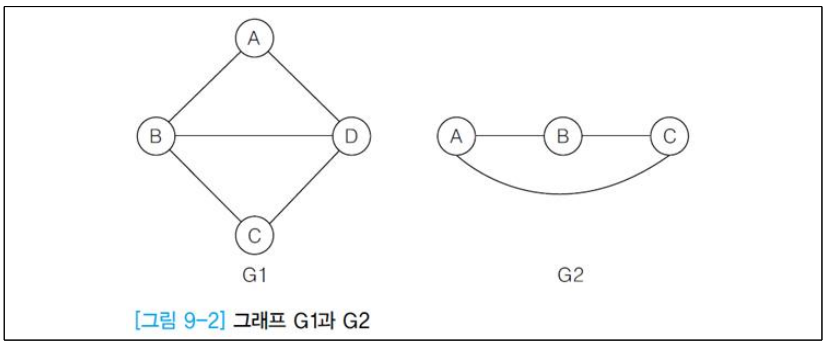
* 그래프 G
- 객체를 나타내는 정점(vertex)과 객체를 연결하는 간선(edge)의 집합
- G=(V, E)
    - V는 그래프에 있는 정점들의 집합
    - E는 정점을 연결하는 간선들의 집합
- 그래프 예제



[그림 9-1] 그래프의 사용 예: 버스 노선도, 인맥 지도

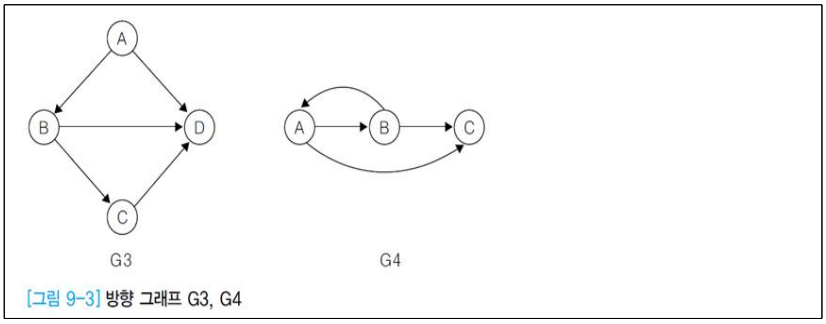
2. 그래프의 종류

* 무방향 그래프(undirected graph)
- 두 정점을 연결하는 간선의 방향이 없는 그래프
- 정점 Vi와 정점 Vj을 연결하는 간선을 (Vi, Vj)로 표현
    - (Vi, Vj)와 (Vj, Vi)는 같은 간선을 의미한다
- V(G1)= {A, B, C, D } E(G1)= {(A,B), (A,D), (B,C), (B,D), (C,D) }
  V(G2)= {A, B, C } E(G2)= {(A,B), (A,C), (B,C) }



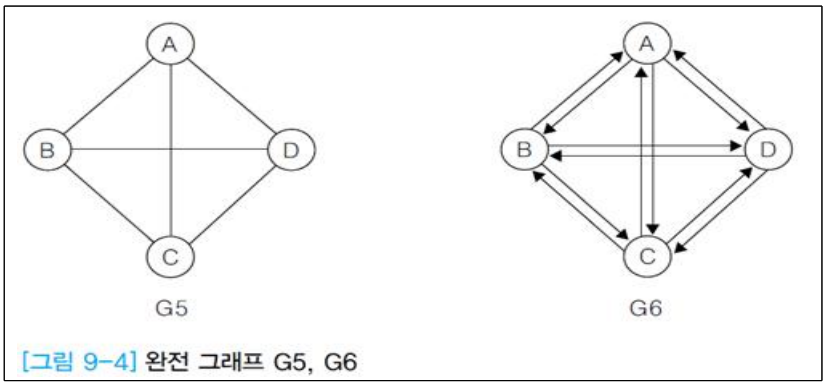
[그림 9-2] 그래프 G1과 G2

* 방향 그래프(directed graph), 다이그래프(digraph)
- 간선이 방향을 가지고 있는 그래프
- 정점 Á¤Ái Vi에서 정점 Vj를 연결하는 간선 즉, Vi→Vj를 <Vi, Vj>로 표현
    - Vi를 꼬리(tail), Vj를 머리(head)라고 한다.
    - <Vi, Vj>와 <Vj, Vi>는 서로 다른 간선을 의미한다.
- V(G3)= {A, B, C, D } E(G3)= {<A,B>, <A,D>, <B,C>, <B,D>, <C,D> }
  V(G4)= {A, B, C } E(G4)= {<A,B>, <A,C>, <B,A>, <B,C> }



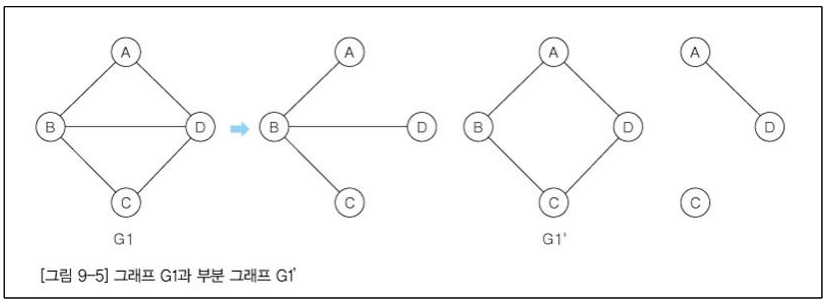
[그림 9-3] 방향 그래프 G3, G4

* 완전 그래프(complete graph)
- 각 정점에서 다른 모든 정점을 연결하여 가능한 최대의 간선 수를 가진 그래프
- 정점이 n개인 무방향 그래프에서 최대의 간선 수 : n(n-1)/2개
- 정점이 n개인 방향 그래프의 최대 간선 수 : n(n-1)개
- 완전 그래프의 예
    - G5는 정점의 개수가 4개인 무방향 그래프이므로 완전 그래프가 되려면 4(4-1)/2=6개의 간선 연결
    - G6은 정점의 개수가 4개인 방향 그래프이므로 완전 그래프가 되려면 4(4-1)=12개의 간선 연결



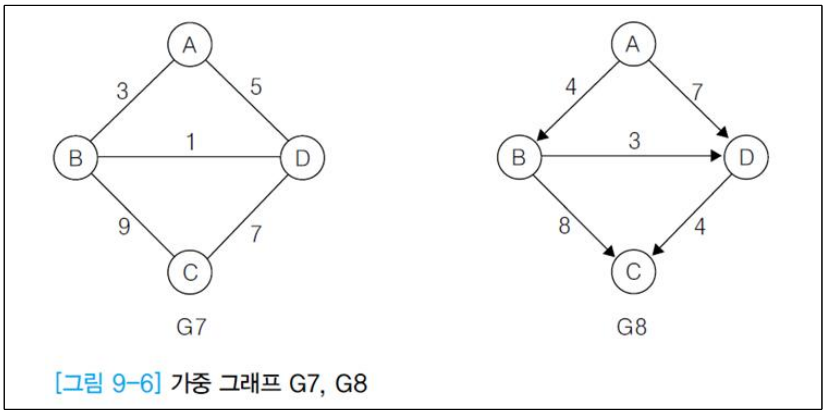
[그림 9-4] 완전 그래프 G5, G6

* 부분 그래프(subgraph)
- 원래의 그래프에서 일부의 정점이나 간선을 제외하여 만든 그래프
- G와 부분 그래프 G'의 관계
    - V(G')⊆V(G), E(G')⊆E(G)
- 그래프 G1에 대한 부분 그래프의 예



[그림 9-5] 그래프 G1과 부분 그래프 G1'

* 가중 그래프(weight graph), 네트워크(network)
    - 정점을 연결하는 간선에 가중치(weight)를 할당한 그래프



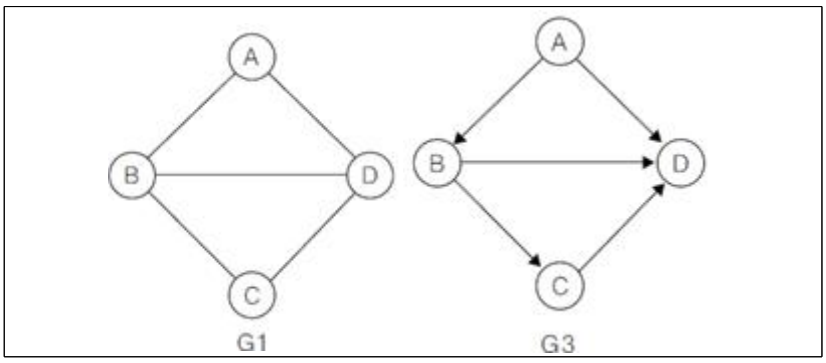
[그림 9-6] 가중 그래프 G7, G8

3. 그래프 관련 용어

* 그래프에서 두 정점 Vi과 Vj를 연결하는 간선 (Vi, Vj)가 있을 때, 두 정점 Vi와 Vj를 인접(adjacent)되어 있다고 하고,

* 간선 (Vi, Vj)는 정점 Vi와 Vj에 부속(incident)되어있다고 한다.
- 그래프G1에서 정점 A와 인접한 정점은 B와 D이고, 정점 A에 부속되어 있는 간선은 (A,B)와 (A,D)이다.

* 차수(degree) - 정점에 부속되어있는 간선의 수
- 무방향 그래프 G1에서 정점 A의 차수는 2, 정점 B의 차수는 3
- 방향 그래프의 정점의 차수 = 진입차수 + 진출차수
    - 방향 그래프의 진입차수(in-degree) : 정점을 머리로 하는 간선의 수
    - 방향 그래프의 진출차수(out-degree) : 정점을 꼬리로 하는 간선의 수
    - 방향 그래프 G3에서 정점 B의 진입차수는 1, 진출차수는 2
    - 정점 B의 전체 차수는 (진입차수 + 진출차수) 이므로 3이 된다



* 경로
- 그래프에서 간선을 따라 갈 수 있는 길을 순서대로 나열한 것 즉, 정점 Vi에서 Vj까지 간선으로 연결된 정점을 순서대로 나열한 리스트
    - 그래프 G1에서 정점 A에서 정점 C까지는 A-B-C 경로와 A-B-D-C 경로, A-D-C 경로, 그리고 A-D-B-C 경로가 있다.

* 경로길이(path length)
- 경로를 구성하는 간선의 수

* 단순경로(simple path)
- 모두 다른 정점으로 구성된 경로
    - 그래프 G1에서 정점 A에서 정점 C까지의 경로 A-B-C는 단순경로이고, 경로 A-B-D-A-B-C는 단순경로가 아니다.

* 사이클(cycle)
- 단순경로 중에서 경로의 시작 정점과 마지막 정점이 같은 경로
    - 그래프 G1에서 단순경로 A-B-C-D-A와 그래프 G4에서 단순경로 A-B-A는 사이클이 된다.

* DAG(Directed acyclic graph)
- 방향 그래프이면서 사이클이 없는 그래프

* 연결 그래프(connected graph)
- 서로 다른 모든 쌍의 정점들 사이에 경로가 있는 그래프 즉, 떨어져있는 정점이 없는 그래프
- 그래프에서 두 정점 Vi에서 Vj까지의 경로가 있으면 정점 Vi와 Vj가 연결(connected)되었다고 한다.

* 트리는 사이클이 없는 연결 그래프이다.

4. 추상 자료형 그래프

```
ADT  Graph
  데이터 : 공백이 아닌 정점의 집합과 간선의 집합
  연산 :
      g∈Graph; u,v∈V;
      createGraph() ::= create an empty Graph;
            // 공백 그래프의 생성 연산

      isEmpty(g) ::= if (g have no vertex) then return true;
                     else return false;
            // 그래프 g가 정점이 없는 공백 그래프인지를 검사하는 연산

      insertVertex(g, v) ::= insert vertex v into g;
            // 그래프 g에 정점 v를 삽입하는 연산

      insertEdge(g, u, v) ::= insert edge (u,v) into g;
            // 그래프 g에 간선 (u,v)를 삽입하는 연산

      deleteVertex(g, v) ::= delete vertex v and all edges incident on v from g;
            // 그래프 g에서 정점 v를 삭제하고 그에 부속된 모든 간선을 삭제하는 연산

      deleteEdge(g, u, v) ::= delete edges (u,v) from  g;
            // 그래프 g에서 간선 (u,v)를 삭제하는 연산

      adjacent(g, v) ::= return set of all vertices adjacent to v;
            // 정점 v에 인접한 모든 정점을 반환하는 연산
End Graph
```

학습내용2 : 인접행렬

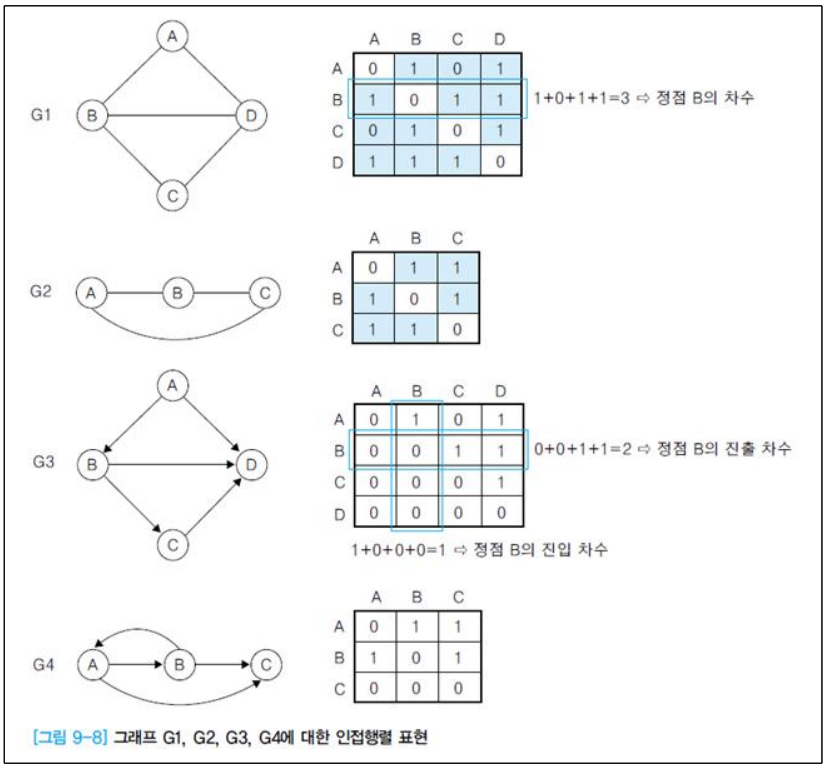
1. 인접행렬의 정의

* 행렬에 대한 2차원 배열을 사용하는 순차 자료구조 방법

* 그래프의 두 정점을 연결한 간선의 유무를 행렬로 저장
- n개의 정점을 가진 그래프 : n x n 정방행렬
- 행렬의 행번호와 열번호 : 그래프의 정점
- 행렬 값 : 두 정점이 인접되어있으면 1, 인접되어있지 않으면 0

* 무방향 그래프의 인접 행렬
- 행 i의 합 = 열 i의 합 = 정점 i의 차수

* 방향 그래프의 인접 행렬
- 행 i의 합 = 정점 i의 진출차수
- 열 i의 합 = 정점 i의 진입차수



[그림 9-8] 그래프 G1, G2, G3, G4에 대한 인접행렬 표현

* 인접 행렬 표현의 단점
- n개의 정점을 가지는 그래프를 항상 n x n개의 메모리 사용
- 정점의 개수에 비해서 간선의 개수가 적은 희소 그래프에 대한 인접 행렬은 희소 행렬이 되므로 메모리의 낭비 발생

2. 인접 행렬 C 프로그램

* 그래프 G1, G2, G3, G4를 인접 행렬로 구현한 프로그램

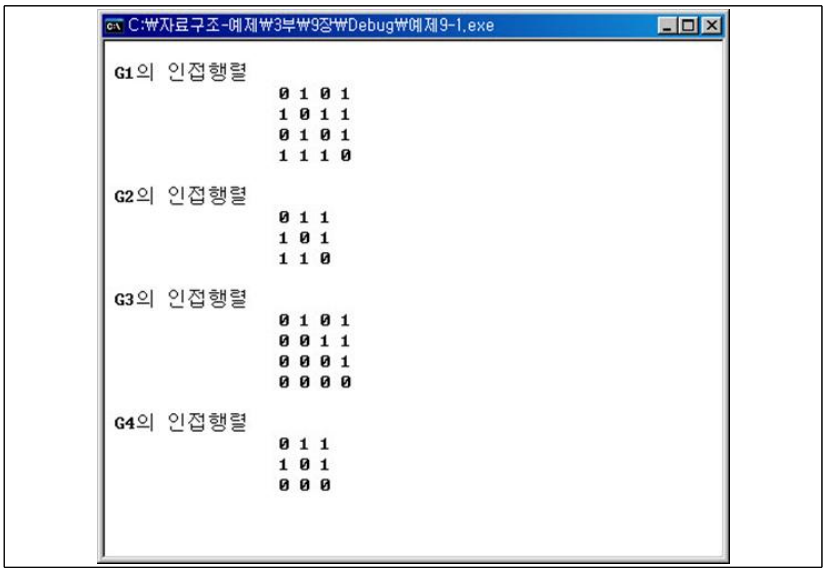
```
001  #include <stdio.h>
002  #include <stdlib.h>
003  #define MAX_VERTEX 30
004  typedef struct graphType{   // 그래프를 인접행렬로 표현하기 위한 구조체 정의
005      int n;                  // 그래프의 정점의 개수
006      int adjMatrix[MAX_VERTEX][MAX_VERTEX];   // 그래프에 대한 30 x 30의 2차원배열
007  } graphType;
008
009  void createGraph(graphType* g)   // 공백 그래프 생성 연산
010  {
011      int i, j;
012      g->n = 0;   // 정점의 개수를 0으로 초기화
013      for(i=0; i<MAX_VERTEX; i++) {
014          for(j=0; j<MAX_VERTEX; j++)
015          g->adjMatrix[i][j]=0;   // 그래프 g에대한 2차원 배열의 값을 0으로 초기화
016      }
017  }
```

```
019  void insertVertex(graphType* g, int v) // 그래프 g에 정점 v를 삽입하는 연산
020  {
021      if(((g->n)+1)>MAX_VERTEX){
022         printf("₩n 그래프 정점의 개수를 초과하였습니다!");
023         return;
024      }
025      g->n++;
026  }
027
028   void insertEdge(graphType* g, int u, int v)
                                  // 그래프 g에 간선(u, v)를 삽입하는 연산
029  {
030      if(u>=g->n || v>=g->n) {
031         printf("₩n 그래프에 없는 정점입니다!");
032         return;
033      }
034      g->adjMatrix[u][v]  = 1; // 삽입한 간선에 대한 2차원 배열 값을 1로 설정
035  }
```

```
036  // 그래프 g의 2차원 배열 값을 순서대로 출력하는 연산
037  void print_adjMatrix(graphType* g)
038  {
039      int i, j;
040      for(i=0; i<(g->n);i++){
041         printf("₩n₩t₩t");
042         for(j=0; j<(g->n);j++)
043            printf("%2d", g->adjMatrix[i][j]);
044       }
045  }
046
047  void main()
048  {
049      int i;
050      graphType *G1, *G2, *G3, *G4;
051      G1 = (graphType *)malloc(sizeof(graphType));
052      G2 = (graphType *)malloc(sizeof(graphType));
053      G3 = (graphType *)malloc(sizeof(graphType));
054      G4 = (graphType *)malloc(sizeof(graphType));
055      createGraph(G1);   createGraph(G2);   createGraph(G3);   createGraph(G4);
056      for(i=0; i<4; i++)     // 그래프 G1
057                  insertVertex(G1, i);
058      insertEdge(G1, 0, 1);
```

```
082      for(i=0; i<4; i++)              // 그래프 G3
083                  insertVertex(G3, i);
084      insertEdge(G3, 0, 1);
085      insertEdge(G3, 0, 3);
086      insertEdge(G3, 1, 2);
087      insertEdge(G3, 1, 3);
088      insertEdge(G3, 2, 3);
089      printf("₩n₩n G3의 인접행렬");
090      print_adjMatrix(G3);
091
092      for(i=0; i<3; i++)              // 그래프 G4
093                  insertVertex(G4, i);
094      insertEdge(G4, 0, 1);
095      insertEdge(G4, 0, 2);
096      insertEdge(G4, 1, 0);
097      insertEdge(G4, 1, 2);
098      printf("₩n₩n G4의 인접행렬");
099      print_adjMatrix(G4);
100
101      getchar();
102  }
```

* 실행결과



학습내용3 : 인접리스트

1. 인접 리스트 정의

* 각 정점에 대한 인접 정점들을 연결하여 만든 단순 연결 리스트

* 각 정점의 차수만큼 노드를 연결
- 리스트 내의 노드들은 인접 정점에 대해서 오름차순으로 연결

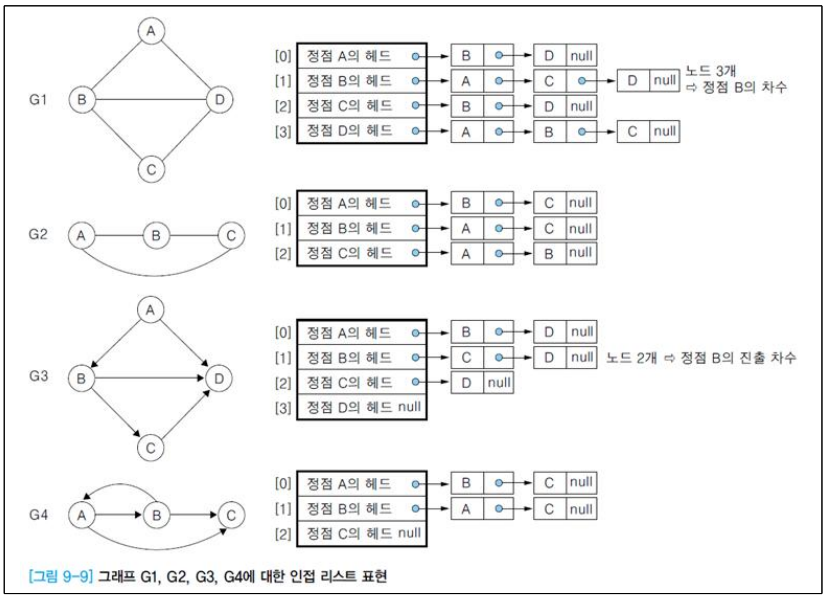
* 인접 리스트의 각 노드
- 정점을 저장하는 필드와 다음 인접 정점을 연결하는 링크 필드로 구성

* 정점의 헤드 노드
- 정점에 대한 리스트의 시작을 표현

* n개의 정점과 e개의 간선을 가진 무방향 그래프의 인접 리스트
- 헤드 노드 배열의 크기 : n
- 연결하는 노드의 수 : 2e
- 각 정점의 헤드에 연결된 노드의 수 : 정점의 차수

* n개의 정점과 e개의 간선을 가진 방향 그래프의 인접 리스트
- 헤드 노드 배열의 크기 : n
- 연결하는 노드의 수 : e
- 각 정점의 헤드에 연결된 노드의 수 : 정점의 진출 차수

* 그래프 G1, G2, G3, G4에 대한 인접 리스트



[그림 9-9] 그래프 G1, G2, G3, G4에 대한 인접 리스트 표현

2. 인접 리스트 C 프로그램

* 그래프 G1, G2, G3, G4를 인접 리스트로 구현한 프로그램
* 그래프의 정점 A, B, C, D 대신에 0,1,2,3의 번호를 사용하여 인식하고, 출력할 때 A, B, C, D 문자로 표시
* 간선의 삽입은 항상 인접 리스트의 첫 번째 노드로 삽입하기

```
001  #include <stdio.h>
002  #include <stdlib.h>
003  #define MAX_VERTEX 30
004  typedef struct graphNode{     // 인접 리스트의 노드 구조를 구조체로 정의
005      int vertex;                  // 정점을 나타내는 데이터 필드
006      struct graphNode* link;   // 다음 인접 정점을 연결하는 링크 필드
007  } graphNode;
008
009  typedef struct graphType{  // 그래프를 인접 리스트로 표현하기 위한 구조체 정의
010      int n;      // 그래프의 정점 개수
011      graphNode* adjList_H[MAX_VERTEX]; // 그래프 정점에 대한 헤드 노드 배열
012  } graphType;
013
014  void createGraph(graphType* g)              // 공백 그래프 생성 연산
015  {
016      int v;
017      g->n = 0;                        // 그래프의 정점의 개수를 0으로 초기화
018      for(v=0; v<MAX_VERTEX; v++)
019      g->adjList_H[v]=NULL;// 그래프의 정점에 대한 헤드노드 배열을 NULL로 초기화
020  }
021
```

```
022  void insertVertex(graphType* g, int v) // 그래프 g에 정점 v를 삽입하는 연산
023  {
024       if(((g->n)+1)>MAX_VERTEX){
025          printf("₩n 그래프 정점의 개수를 초과하였습니다!");
026          return;
027       }
028       g->n++;
029  }
030
031  void insertEdge(graphType* g, int u, int v)  // 그래프g에 간선 (u, v)를 삽입하는 연산
032  {
033       graphNode* node;
034       if(u>=g->n || v>=g->n) {
035          printf("₩n 그래프에 없는 정점입니다!");
036          return;
037       }
038       node = (graphNode *)malloc(sizeof(graphNode));
039       node->vertex = v;
040       node->link = g->adjList_H[u];
041       g->adjList_H[u] = node;
042  }
```

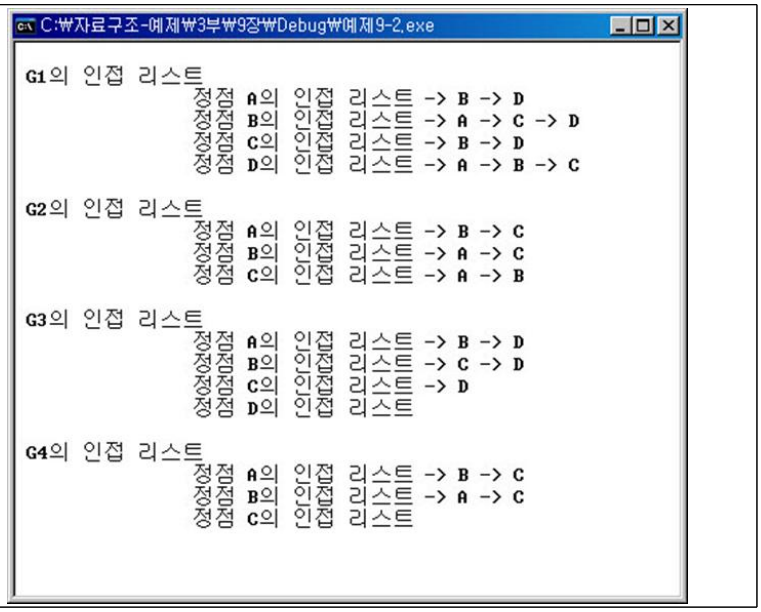
```
043
044  void print_adjList(graphType* g)
                  // 그래프 g의 각 정점에 대한 인접 리스트를 출력하는 연산
045  {
046       int i;
047       graphNode* p;
048       for(i=0; i<g->n; i++){
049          printf("₩n₩t₩t정점 %c의 인접 리스트", i+65);
050          p= g->adjList_H[i];
051          while(p){
052             printf(" -> %c", p->vertex +65); //정점 0~3을 A~D로 출력
053             p = p->link;
054          }
055       }
056  }
057
```

```
058  void main()
059  {
060       int i;
061       graphType *G1, *G2, *G3, *G4;
062       G1 = (graphType *)malloc(sizeof(graphType));
063       G2 = (graphType *)malloc(sizeof(graphType));
064       G3 = (graphType *)malloc(sizeof(graphType));
065       G4 = (graphType *)malloc(sizeof(graphType));
066       createGraph(G1); createGraph(G2); createGraph(G3);createGraph(G4);
067       for(i=0; i<4; i++) // 그래프 G1
068          insertVertex(G1, i);
069       insertEdge(G1, 0, 3);
070       insertEdge(G1, 0, 1);
071       insertEdge(G1, 1, 3);
072       insertEdge(G1, 1, 2);
073       insertEdge(G1, 1, 0);
074       insertEdge(G1, 2, 3);
075       insertEdge(G1, 2, 1);
076       insertEdge(G1, 3, 2);
077       insertEdge(G1, 3, 1);
078       insertEdge(G1, 3, 0);
```

```
079     printf("₩n G1의 인접 리스트");
080     print_adjList(G1);
081
082     for(i=0; i<3; i++) // 그래프 G2
083        insertVertex(G2, i);
084     insertEdge(G2, 0, 2);
085     insertEdge(G2, 0, 1);
086     insertEdge(G2, 1, 2);
087     insertEdge(G2, 1, 0);
088     insertEdge(G2, 2, 1);
089     insertEdge(G2, 2, 0);
090     printf("₩n₩n G2의 인접 리스트");
091     print_adjList(G2);
092
093     for(i=0; i<4; i++) // 그래프 G3
094        insertVertex(G3, i);
095     insertEdge(G3, 0, 3);
096     insertEdge(G3, 0, 1);
097     insertEdge(G3, 1, 3);
098     insertEdge(G3, 1, 2);
099     insertEdge(G3, 2, 3);
100     printf("₩n₩n G3의 인접 리스트");
101     print_adjList(G3);
```

```
102
103     for(i=0; i<3; i++) // 그래프 G4
104        insertVertex(G4, i);
105     insertEdge(G4, 0, 2);
106     insertEdge(G4, 0, 1);
107     insertEdge(G4, 1, 2);
108     insertEdge(G4, 1, 0);
109     printf("₩n₩n G4의 인접 리스트");
110     print_adjList(G4);
111
112     getchar();
113 }
```

* 실행 결과



【학습정리】

1. 연결한 객체를 나타내는 정점과 객체를 연결하는 간선의 집합으로 구성된 그래프 G를 G=(V,E)로 정의한다.
- 그래프는 간선의 방향성 유무에 따라서 방향 그래프와 무방향 그래프가 있고, 연결정도에 따라 연결 그래프와 단절 그래프, 완전 그래프가 있다. 그리고 가중치를 가진 간선으로 이루어진 가중치 그래프가 있다.

2. 그래프를 구현하기 위해서 표현하는 방법은 순차 자료구조를 이용한 2차원 배열의 인접행렬 방법과 연결 자료구조인 연결 리스트를 사용하는 인접 리스트 방법이 있다.
- 그래프의 특성과 필요한 연산에 따라 적당한 표현 방법을 선택한다.

3. 인접행렬은 그래프를 구성하는 정점에 대해서 두 정점을 연결한 간선의 유무를 2차원 배열에 저장하는 방법으로 정점의 수에 대한 정방행렬을 사용한다.
- 행과 열로 표현하는 두 정점이 인접되어 있으면 배열 값을 1, 인접되어 있지 않으면 배열 값을 0으로 표현한다.