그래프 구현하기 (인접리스트 구조)

■ 들어가기에 앞서

- 본 자료는 **인접리스트 구조**를 사용해 그래프를 구축하는 코드이다.
  - 인접리스트 구조의 추상적인 그림은 교재나 강의자료 참조
- **정점과 리스트는 배열**로 구현하였다.
- **부착리스트는 헤더 노드가 있는 단순 연결리스트**로 구현하였다.
  - 부착리스트의 노드들은 반대쪽 정점 번호의 오름차순으로 유지
  - 헤더 노드 덕분에 코드가 단순해짐 (삽입하는 부분)
- 본 코드에서 구축하는 그래프는 **실습 10주차의 문제에 제시된 그래프**이다.
  - 삭제나 간선의 가중치 변경은 고려하지 않았다.

■ 코드

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

// ////////////////////////////////////////////
// 그래프의 인접리스트 구조에 필요한 자료형
// ////////////////////////////////////////////

typedef struct node {
   int edgeId;
   struct node *next;
} Node;              // 부착리스트의 노드

typedef struct vertex {
   int vname;
   Node *incid_list;
} Vertex;         // 정점 배열의 원소

typedef struct edge {
   int ename;
   int vtx1, vtx2;      // 간선의 양 끝점
   int weight;
} Edge;           // 간선 배열의 원소

typedef struct graph {
   Vertex *vertices;    // 정점 배열(동적 할당)
   int vsize;           // 정점 배열의 크기
   Edge *edges;         // 간선 배열(동적 할당)
   int esize;           // 산선 배열의 크기
} Graph;  // 그래프의 구성 요소
```

```
// ////////////////////////////////////////////
// 기본 요소 함수
// ////////////////////////////////////////////

// 부착 리스트의 node를 할당하고 주어진 인자로 초기화
Node *get_node(int eid, Node *next) {
   Node *node = (Node *)malloc(sizeof(Node));
   node->edgeId = eid;
   node->next = next;

   return node;
}

// 부착 리스트의 노드(출발정점은 vs)가 나타내는 간선의 반대쪽 정점 찾기
int opposite(Graph *G, Node *node, int vs) {
   Edge e = G->edges[node->edgeId];
   return (e.vtx1 == vs) ? e.vtx2 : e.vtx1;
}

// ////////////////////////////////////////////
// 간선 추가와 관련된 함수들
// ////////////////////////////////////////////

// 간선배열의 eid 원소로 간선 (v1,v2) 설정
void set_edges_arr(Graph *G, int eid, int v1, int v2, int w) {
   G->edges[eid].ename = eid;
   G->edges[eid].vtx1 = v1;
   G->edges[eid].vtx2 = v2;
   G->edges[eid].weight = w;
}

// 노드 vs의 부착리스트에 간선 (vs, vd)에 대한 노드 추가
void insert_incid_node(Graph *G, int eid, int vs, int vd) {
   Node *node = G->vertices[vs].incid_list;
   Node *new_node;

   // 반대쪽 정점 번호의 오름차순으로 부착 리스트 유지하기 위해
   // 삽입할 위치의 앞 노드 찾기 (헤더 노드 있는 연결 리스트라서 코드가 단순함)
   while (node->next && opposite(G, node->next, vs) < vd)  // node->next가 대상임에 주목
      node = node->next;                    // while문 종료후 node는 삽입할 위치의 앞 노드

   new_node = get_node(eid, node->next);    // 삽입할 노드의 정보 설정
   node->next = new_node;                // node --> new_node 로 링크 설정
}

// 그래프에 간선 (v1, v2) 추가
void add_edge(Graph *G, int eid, int v1, int v2, int w) {
   set_edges_arr(G, eid, v1, v2, w);     // 간선 배열의 eid 원소에 간선 정보 저장

   insert_incid_node(G, eid, v1, v2); // 정점 v1의 부착리스트에 노드 추가
   if (v1 != v2)        // loop 가 아니면
      insert_incid_node(G, eid, v2, v1); // 반대쪽 정점(v2)의 부착리스트에 노드 추가
}
```

```
// ////////////////////////////////////////////
// 그래프 구축을 위한 상위 레벨 함수
// ////////////////////////////////////////////

// 그래프 정점 정보 설정
void set_vertices(Graph *G, int vsize) {
   G->vertices = (Vertex *)malloc(sizeof(Vertex)*vsize);     // 정점 배열 할당
   G->vsize = vsize;

   for (int i = 0; i < vsize; ++i) {
      G->vertices[i].vname = i;          // 정점이름 설정
      G->vertices[i].incid_list = get_node(-1, NULL);  // 헤더 노드 달기
   }
}

// 그래프 간선 정보 설정
void set_edges(Graph *G, int esize) {
   G->edges = (Edge *)malloc(sizeof(Edge)*esize);    // 간선 배열 할당
   G->esize = esize;

   for (int i = 0; i < esize; ++i) {   // 간선 배열 초기화
      G->edges[i].ename = -1;    // 미사용 원소는 -1로 초기화
      G->edges[i].vtx1 = -1;
      G->edges[i].vtx2 = -1;
   }

   add_edge(G, 0, 1, 2, 1);      // 실습 10주차 문제의 그래프 구축
   add_edge(G, 1, 6, 1, 2);      // 프로그램이 잘 동작하는 지 확인하기 위해
   add_edge(G, 2, 1, 4, 1);      // 간선 추가 순서를 일부러 섞었음
   add_edge(G, 3, 2, 3, 1);
   add_edge(G, 4, 3, 5, 4);
   add_edge(G, 5, 1, 3, 1);
   add_edge(G, 6, 5, 5, 4);
   add_edge(G, 7, 5, 6, 3);
}

//그래프 구축
void build_graph(Graph *G) {
   int vsize, esize;
   vsize = 7;        // 실습 10주차 문제의 그래프 구축 (정점 0은 사용 안함)
   esize = 21;       // 실습 10주차 문제의 그래프 구축 (최대 21개의 간선 가능)

   set_vertices(G, vsize);// 정점 정보 설정
   set_edges(G, esize);// 간선 정보 설정
}
```

```c
// ////////////////////////////////////////////
// 그래프의 정보 출력과 메모리 해제 관련 함수
// ////////////////////////////////////////////

// 특정 정점의 정보(인접 정점 번호와 간선의 가중치) 출력
void print_adj_vertices(Graph *G, int vid) {
   Node *node = G->vertices[vid].incid_list->next;  // 헤드의 다음 노드

   printf("%d:", vid);
   while (node) {
      printf("[%d,", opposite(G, node, vid));     // 반대쪽 정점 번호 출력
      printf(" %d] ", G->edges[node->edgeId].weight);  // 가중치 출력
      node = node->next;
   }
   printf("\n");
}

// 그래프의 정보 출력
void print_graph(Graph *G) {
   int i;

   for (i = 0; i < G->vsize; ++i)
      print_adj_vertices(G, i);
}

// 부착리스트의 노드들 해제
void free_incid_list(Node *node) {
   Node *p;

   while (node) {
      p = node->next;      // 다음 노드 주소 임시 저장
      free(node);       // node 해제
      node = p;         // 다음 노드 주소를 node 변수에
   }

}

// 그래프에 할당된 메모리 해제
void free_graph(Graph *G)
{
   int i;

   for (i = 0; i < G->vsize; ++i)      // 부착리스트 해제
      free_incid_list(G->vertices[i].incid_list);

   free(G->vertices);      // 정점 배열 해제
   free(G->edges);         // 간선 배열 해제
}
```

```
// ////////////////////////////////////////////
// main() 함수
// ////////////////////////////////////////////

int main()
{
   Graph G;       // 그래프 변수 선언 (정적할당)

   build_graph(&G);     // 그래프 구축
   print_graph(&G);     // 그래프 정보 출력
   free_graph(&G);         // 그래프의 메모리 해제

   return 0;
}
```