데이터베이스에 기반한 그래프 라이브러리 및 그래프 알고리즘 개발

박휴찬* · 추인경**

Development of Database Supported Graph Library and Graph Algorithms

Hyu-Chan Park* · In-Kyung Chu**

이 논문은 2002년도 한국과학재단 목적기초연구(R05-2001-000-00851-0) 지원으로 수행되었음

요 약

본 논문은 관계형 데이터베이스 기반하여 그래프를 저장하고 그래프 알고리즘을 정의할 수 있는 방법을 제안한다. 이 방법에서 그래프는 릴레이션으로 표현되며, 그래프의 각 정점과 간선은 이 릴레이션의 튜플로서 데이터베이스에 저장된다. 이를 위해 그래프의 저장 및 관리뿐만 아니라 다양한 응용프로그램 개발에도 사용될 수 있는 기본적인 그래프 함수들을 라이브러리로 개발하였다. 또한, 그래프에 대한 알고리즘을 추출, 선택, 죠인과 같은 관계대수 연산을 이용하여 정의하였으며, SQL과 같은 데이터베이스 언어를 사용하여 구현하였다. 이와 같은 데이터베이스에 기반한 방법은 메모리에 수용되지 않는 크기의 그래프를 효과적으로 처리할 수 있을 뿐만 아니라 다양한 응용프로그램 개발을 용이하게 할 것이다.

ABSTRACT

This paper proposes a method for storing graphs and defining graph algorithms based on the well-developed relational database. In this method, graphs are represented in the form of relations. Each vertex and edge of a graph is represented as tuples of the table and saved in a database. We developed a library of graph operations for the storage and management of graphs and the development of graph applications. Furthermore, we defined graph algorithms in terms of relational algebraic operations such as projection, selection, and join. They can be implemented with the database language such as SQL. This database approach provides an efficient methodology to deal with very large-scale graphs and to support the development of graph applications.

키워드

그래프(Graph), 데이터베이스(Database), 그래프 라이브러리(Graph Library)

*한국해양대학교 기계 · 정보공학부 **하우리

접수일자 : 2002. 5. 2

I. 서 론

그래프를 표현할 수 있는 다양한 방법이 존재하지만, 인접행렬(adjacency matrix), 인접리스트(adjacency list), 인접다중리스트(adjacency multilist) 등이 일반적으로 많이 사용되고 있다[1-3]. 비록 이런 표현 방법과 그래프 알고리즘이 많은 응용분야에서 성공적으로 적용되어 왔지만, 메모리 기반 특성(in-memory property) 때문에 다소간의 제한점을 가지고 있다.

우선, 그래프 객체의 생명이 영구적이 아닌 임시적이기 때문에 프로그램이 실행되는 동안에만 그래프의 표현이 유효하고 프로그램이 끝나면 사라지게 된다. 다음으로, 처리할 수 있는 그래프의 크기가 메모리의 크기로 제한된다. 이런 상황에서는 그래프의 일부분이 계속적으로 화일에 저장되고 화일로부터 메모리에 읽어들여져야 한다. 또한, 어떤 응용에서는 많은 사용자가 동시에 그래프에 접근할 필요가 있다. 이런 환경에서 그래프에 대한 변경 작업이 동시에 이루지면 일관성이 없는 그래프가 초래될 수도 있다.

이러한 한계점을 극복하기 위하여, 본 논문에서는 관계형 데이터베이스(relational database)에 기반하여 그래프를 저장하고 관리하기 위한 방법을 제안한다. 이 방법에서 그래프는 릴레이션(relation)으로 표현되며 그래프의 각 정점과 간선은 이 릴레이션의 튜플(tuple)로서 데이터베이스에 저장된다. 이를 위하여 그래프 테이블을 설계하였으며 기본적인 그래프 연산을 라이브러리화 하였다.

또한, 본 논문에서는 관계형 데이터베이스에 기반하여 그래프 알고리즘을 정의하고 개발하는 방법을 제안한다. 이 방법에서 각종 알고리즘은 추출(projection), 선택(selection), 죠인(join) 등과 같은 관계대수(relational algebra) 연산을 이용하여 정의된다[4, 5]. 대표적인 그래프 알고리즘인 최단경로(shortest path)과 신장트리(spanning tree) 알고리즘을 정의하였다. 이렇게 정의된 그래프 알고리즘은 관계형 데이터베이스를 사용하여 쉽게 구현될 수 있다.

이러한 데이터베이스를 이용한 방법은 기존의 방법에 비하여 많은 장점을 가지고 있다. 메모리에 수용되지 않는 크기를 갖는 그래프를 효과적으로 처리할 수 있을 뿐만 아니라 다수의 사용자들이 저장된 그래프를 공유할 수도 있다. 또한 라이브러리에서 제공하는 기본 함수를 이용하면 각종 응용 프로그램의 개발이 용이해질 수 있다.

II. 관련 연구

그래프를 표현하고 처리하기 위한 방법들이 제안되어 왔다. 우선, 그래프 연산을 라이브러리화한 연구로서 Knuth[6]는 다양한 형태의 그래프를 조작하고 시험할 수 있는 데이터세트와 프로그램 라이브러리를 제안하였다. 하지만 이 라이브러리는 그래프가 메인 메모리에 모두 수용되는 경우를 가정한 문제점이 있다.

메인 메모리의 용량을 초과하는 크기를 갖는 그래프를 처리하기 위해 Chiang[7]과 Nodian[8]은 화일에 기반한 외부-메모리 그래프 알고리즘과 그래프를 처리함에 있어서 디스크 입출력을 최소화하기 위한 방법에 관한 연구를 하였다. 하지만 위의 연구들은 그 효용성이 데이터베이스를 활용한 것보다 제한적일 수 있다.

데이터베이스 기술을 그래프에 활용한 연구로서 Biskup[9]은 기존의 관계형 질의어가 그래프에 관한 정보를 표현하는 표현력이 약하기 때문에 이를 보완한 XSQL이라는 확장된 SQL을 제시하였다. Guting[10]은 객체지향 데이터베이스를 기반으로 그래프 모델과 질의어를 제안하였다. 이러한 연구와 그 외의 관련 연구[11-13]에서는 그래프를 데이터베이스화고 질의하기 위하여 새로운 데이터 모델을 제안하든지 데이터베이스 질의어를 확장하는데 주안점을 두었다. 하지만 이와 같은 접근 방법은 기존의 데이터베이스 시스템을 그대로 사용할 수 없기 때문에 호환성이 떨어지게 된다. 또한 사용자의 입장에서 보면 새로운 데이터베이스나 질의어를 익혀야 하는 부담을 지게 된다.

III. 데이터베이스에 기반한 그래프 라이브러리 개발

본 연구에서는 그래프 체계적으로 표현할 수 있도록 데이터베이스 테이블을 상세하게 설계하였다. 또

한, 그래프를 데이터베이스에 저장하고 관리하기 위한 기본적인 연산을 라이브러리로 개발하였다.

3.1 그래프 테이블 상세 설계

그래프를 저장하기 위한 데이터베이스 테이블은 그래프정보(graph_info) 테이블, 정점(vertex) 테이블과 정점속성(vertex_att) 테이블, 간선(edge) 테이블과 간선속성(edge_att) 테이블로 구성된다.

1) 그래프정보 테이블: 각 그래프의 메타 정보를 저장하는 테이블은 graph_info이다. 각각의 그래프를 고유하게 식별하는 그래프식별자(g_id)를 기본키(primary key)이다.

[*graph_info*]

필드명	설 명
g_id	그래프식별자, (기본키)
g_name	그래프이름
key_word	키워드
start_day	그래프 작성시작일
update_day	그래프 최종수정일
ect	기타정보

2) 정점정보 테이블: 정점에 관한 정보를 저장하는 테이블은 정점의 기본정보를 저장하는 vertex 테이블과 정점에 부가된 각종 속성들과 그 값을 저장하는 vertex_att 테이블로 구성된다. 이렇게 두개의 테이블로 분리한 이유는 한 정점에 여러 개의 속성이 존재할 수 있기 때문이다. vertex 테이블의 구조는 정점을 고유하게 식별하는 정점식별자인 v_id, 정점이 소속된 그래프를 나타내는 g_id로 구성된다. vertex_att 테이블의 구조는 속성의 이름을 나타내는 vatt_name, 속성의 값을 저장하는 vatt_value, 속성이 소속된 정점과 이 정점이 소속된 그래프를 나타내는 v_id와 g_id로 구성된다.

[*vertex*]

필드명	설 명
g_id	정점이 소속된 그래프식별자, (기본키)
v_id	정점식별자, (기본키)

[*vertex_att*]

필드명	설 명
g_id	정점이 소속된 그래프식별자, (기본키)
v_id	속성이 소속된 정점식별자, (기본키)
vatt_name	속성의 이름, (기본키)
vatt_value	속성의 값

3) 간선정보 테이블: 간선에 관한 정보를 저장하기 위한 테이블은 간선의 기본정보를 저장하는 edge 테이블과 간선에 부가된 각종 속성들과 그 값을 저장하는 edge_att 테이블로 구성된다. edge 테이블의 구조는 간선을 고유하게 식별하는 간선식별자인 e_id, 간선이 소속된 그래프를 나타내는 g_id, 간선이 연결하는 두개의 정점을 나타내는 vx_id와 vy_id, 간선의 방향성 유무를 나타내는 dir로 구성된다. edge_att 테이블의 구조는 속성의 이름을 나타내는 eatt_name, 속성의 값을 저장하는 eatt_value, 속성이 소속된 간선과 이 간선이 소속된 그래프를 나타내는 e_id와 g_id로 구성된다.

[*edge*]

필드명	설 명
g_id	간선이 소속된 그래프식별자, (기본키)
e_id	간선식별자, (기본키)
vx_id	간선이 연결하는 두 개의 정점식별자
vy_id	
dir	간선의 방향성 유무

[*edge_att*]

필드명	설 명
g_id	간선이 소속된 그래프식별자, (기본키)
e_id	속성이 소속된 간선식별자, (기본키)
eatt_name	속성의 이름, (기본키)
eatt_value	속성의 값

이상에서 설계한 테이블로 그래프가 표현될 수 있음을 예제를 통하여 설명한다. 그림 1의 그래프는 도시들과 그들 사이의 도로들을 모델링한 것이다. 각각의 도시와 도로는 여러 가지 속성을 가지고 있다. 그림 1의 그래프는 그림 2와 같이 위에서 설계한 테이블로 표현될 수 있다.

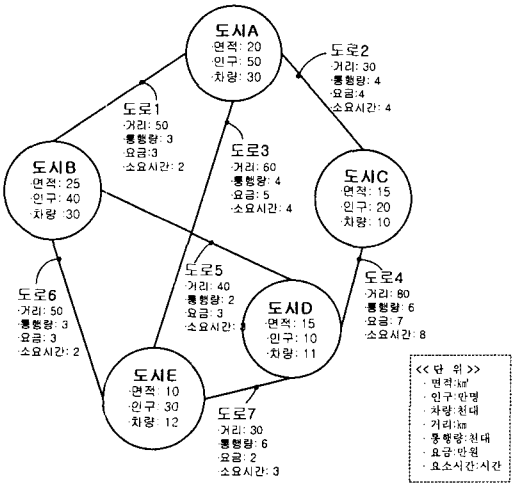

그림 1. 도시와 도로의 관련성을 나타내는 예제그래프
Fig. 1 Graph for relationships among cities and roads

[*vertex*]

g_id	*v_id*
G1	도시A
G1	도시B
G1	도시C
G1	도시D
G1	도시E

[*vertex_att*]

g_id	*v_id*	*vatt_name*	*vatt_value*
G1	도시A	면적	20
G1	도시A	인구	50
G1	도시A	차량	30
G1	도시B	면적	25
…			

[*edge*]

g_id	*e_id*	*vx_id*	*vy_id*	*dir*
G1	도로1	도시A	도시B	n
G1	도로2	도시A	도시C	n
G1	도로3	도시A	도시E	n
G1	도로4	도시C	도시D	n
G1	도로5	도시B	도시D	n
G1	도로6	도시B	도시E	n
G1	도로7	도시D	도시E	n

[*edge_att*]

g_id	*e_id*	*eatt_name*	*eatt_value*
G1	도로1	거리	50
G1	도로1	통행량	3
G1	도로1	요금	3
G1	도로1	소요시간	2
G1	도로2	거리	30
G1	도로2	통행량	4
…			

그림 2. 예제 그래프의 테이블 표현
Fig. 2 Tables representation of the example graph

3.2 그래프 라이브러리 개발

테이블들로 표현된 그래프는 데이터베이스에 저장되고 관리될 수 있다. 이를 위하여 기본적인 함수들을 라이브러리로 개발하였다. 데이터베이스는 Oracle8을, 개발 언어는 C와 Pro*C를 사용하였다.

1) 그래프 데이터베이스 연결/해제: 그래프에 대한 작업을 수행하기 위해 데이터베이스에 연결을 설정하고 작업 완료 후 연결을 해제하는 라이브러리이다.
- connect (id, passwd)
- disconnect ()

2) 그래프 생성: 새로운 그래프를 생성할 경우에 graph_info 테이블에 새로운 그래프에 대한 그래프식별자와 그래프이름 및 기타정보 등에 대한 값을 입력한다.
- create_graph (gid, gname, key, start, update, etc)

3) 그래프 정보 삽입: 정점 삽입은 그래프식별자와 정점식별자를 이용해서 이루어지며 vertex 테이블에 저장된다. 정점에 대해 속성 데이터를 삽입하기 위해서는 그래프식별자, 정점식별자, 속성이름, 속성값을 이용하여 이루어지며 vertex_att 테이블에 저장된다. 간선 삽입은 그래프식별자, 간선식별자, 두 개의 연결정점, 방향성을 이용해서 이루어지며 edge 테이블에 저장된다. 간선에 대해 속성 데이터를 삽입하기 위해서는 그래프식별자, 간선식별자, 속성이름, 속성값을 이용하여 이루어지며 edge_att 테이블에 저장된다.
- insert_vertex (gid, vid)
- insert_vertex_attr (gid, vid, vatt_name, vatt_value)
- insert_edge (gid, eid, vertex_xid, vertex_yid, direction)
- insert_edge_attr (gid, eid, edge_attname, edge_attvalue) 등

4) 그래프 정보 삭제: 그래프 삭제는 그래프식별자를 이용하여 graph_info 테이블에 저장된 그래프 데이터를 삭제할 수 있다. 이 삭제 결과는 vertex 테이블, edge 테이블, vertex_att 테이블, edge_att 테이블에도 적용되며 해당 그래프식별자에 소속된 모든 데이터가 삭제된다. 정점 삭제는 그래프식별자와 정점식별자를 이용하여 삭제 가능하다. 정점속성 삭제는 그래프식별자와 정점식별자 그리고 해당 속성이름을 이용하여 vertex_att 테이블에 저장된 정점 속성 데

이터를 삭제할 수 있다. 간선 삭제는 그래프식별자와 간선식별자를 이용하여 삭제 가능하다. 간선속성 삭제는 그래프식별자와 간선식별자 그리고 해당 속성 이름을 이용하여 edge_att 테이블에 저장된 간선 속성 데이터를 삭제할 수 있다.

- del_graph (gid)
- del_vertex (gid, vid)
- del_edge (gid, eid) 등

5) 그래프 정보 갱신: 그래프 갱신은 그래프식별자를 기준으로 새로운 그래프식별자, 그래프이름, 키워드, 작업시작일, 최종수정일, 기타정보로 갱신할 수 있다. 정점 갱신은 그래프식별자, 정점식별자를 기준으로 두고 새로운 정점식별자, 속성이름, 속성값으로 갱신할 수 있다. 간선 갱신은 그래프식별자, 간선식별자를 기준으로 새로운 간선식별자, 간선 연결정점, 방향성, 속성이름, 속성값으로 갱신할 수 있다.

일괄적으로 갱신하는 함수:

- update_graph_info (old_gid, new_gid, new_gname, new_key_word, new_start, new_update, new_etc)
- update_vertex_info (gid, old_vid, new_vid, old_vatt_name, new_vatt_name, new_vatt_value)
- update_edge_info (gid, old_eid, new_eid, new_vx_id, new_vy_id, new_direction, old_edge_attname, new_edge_attname, new_edge_attvalue)

필드별로 갱신하는 함수:

- update_graph_id (old_gid, new_gid)
- update_vertex_id (gid, old_vid, new_vid)
- update_edge_id (gid, old_eid, new_eid) 등

6) 그래프 정보 검색: 검색 연산은 데이터베이스에 존재하는 테이블에 대하여 죠인 등의 연산을 통하여 이루어진다. 사용자는 검색된 결과에 대하여 다양한 그래프 알고리즘을 적용하여 좀 더 복잡한 작업을 수행할 수 있다. 그래프 검색은 그래프식별자를 이용하여 graph_info 테이블에 저장되어 있는 전체 정보뿐만 아니라 각 속성에 대한 부분정보를 검색할 수 있다. 정점 검색은 그래프식별자와 정점식별자를 이용하여 vertex 테이블과 vertex_att 테이블에 저장되어 있는 해당 정점에 대한 데이터 검색이 가능하다. 정점의 속성정보를 검색할 경우 그래프식별자, 정점식별자, 속성이름을 이용하여 해당 속성값 검색이 가능하다. 간선 검색은 그래프식별자와 간선식별자를 이용하여 edge 테이블과 edge_att 테이블에 저장되어 있는 해당 간선에 대한 데이터 검색이 가능하다. 간선의 속성정보를 검색할 경우 그래프식별자, 간선식별자, 속성이름을 이용하여 해당 속성값 검색이 가능하다.

일괄적으로 검색하는 함수:

- get_graph_info (gid)
- get_vertex_info (gid, vid)
- get_edge_info (gid, eid) 등

필드별로 검색하는 함수:

- get_graph_name (gid)
- get_vertex_attribute (gid, vid, vatt_name)
- get_edge_attribute (gid, eid, eatt_name) 등

7) 트랜잭션 처리: 트랜잭션 처리 라이브러리는 일련의 그래프 연산이나 알고리즘 수행을 하나의 트랜잭션으로 처리할 수 있도록 지원한다. 복구 함수는 잘못 수행한 작업을 작업수행 이전상태로 복구한다. 복구위치(save point)를 지정하면 그 이후 작업에 대해서 잘못된 작업을 수행을 했을 경우 복구위치 상태로 작업을 복구할 수 있다. 완료 함수는 작업내용을 완료(commit)한다.

- rollback ()
- commit ()
- save_point ()

그림 3은 위의 라이브러리를 이용하여 그래프를 생성하고, 정점과 간선을 차례로 삽입한 후, 검색하는 예제 프로그램이다. 이 프로그램에서 알 수 있는 바와 같이 비록 그래프를 데이터베이스에 저장하고 관리하는 그래프 응용 프로그램이지만 마치 일반적인 프로그램과 유사하게 개발할 수 있음을 보여준다. 나아가 그래프를 표현하기 위한 자료구조 등이 따로 없어도 되기 때문에 프로그램의 구조가 아주 간단하게 된다.

```
#include "graph.h" /* 그래프 라이브러리 헤더파일 선언 */

main()
{
    /* 그래프 데이터 베이스에 연결 설정 */
  connect("myid", "mypasswd");

   /* 그래프 생성*/
  create_graph("G1","수송계획","도시,도로,수송","2000/10/11",
               "2000/11/22", "도시와 인접 도로간의 관계");

  while(key!=ESC) {  /* vertex 삽입 루프 */
    scanf("%s %s", graph_id, vertex_id);
    insert_vertex(graph_id, vertex_id);
  }

  while(key!=ESC) {  /* vertex 속성 삽입 루프 */
    scanf("%s %s %s %s", graph_id, vertex_id, vatt_name,
          vatt_value);
    insert_vertex_attribute(graph_id, vertex_id, vatt_name,
          vatt_value);
  }

  while(key!=ESC) {  /* edge 삽입 루프 */
    scanf("%s %s %s %s %s", graph_id, edge_id, vx_id,
          vy_id, dir);
    insert_edge(graph_id, edge_id, vx_id, vy_id, dir);
  }

  while(key!=ESC) {  /* edge 속성 삽입 루프 */
    scanf("%s %s %s %s",graph_id, edge_id, eatt_name,
          eatt_value);
    insert_vertex_attribute(graph_id, edge_id, eatt_name,
          eatt_value);
  }

   /* 그래프 ID 'G1'에 소속된 모든 vertex ID를 검색*/
  result=get_all_vertex("G1");
  while(row=graph_fetch_row(result)) {
    printf("vertex list: %s", row);
  }
}
```

그림 3. 라이브러리를 이용한 응용 프로그램 예제
Fig. 3 An application program using the graph library

IV. 데이터베이스에 기반한 그래프 알고리즘 개발

본 장에서는 먼저 관계대수를 이용하여 그래프 알고리즘이 정의될 수 있음을 보이고, 다음으로 이렇게 정의된 알고리즘이 데이터베이스 상에서 쉽게 구현될 수 있음을 보이고자 한다.

4.1 관계대수를 이용한 그래프 알고리즘 정의

그래프를 릴레이션(테이블) 형태로 표현한 경우, 그래프에 대한 연산과 알고리즘은 관계대수 연산을 이용하여 정의되어질 수 있다. 알고리즘의 정의를 간단하게 하기 위하여 정점은 V[vid]라는 릴레이션으로 표현하고, 간선은 E[vid1, vid2, cost]라는 릴레이션으로 표현한다. 물론, 추가적인 다른 속성을 고려하더라도 알고리즘의 핵심은 유사하게 된다.

우선, 그래프로부터 정보를 추출하는 몇 가지의 기본적인 연산은 다음과 같이 간단한 관계대수식으로 정의될 수 있다.

1) 정점의 수: $\pi_{count(*)}V$
2) 간선의 수: $\pi_{count(*)}E$
3) 정점 v에 인접(adjacent)한 정점:
   $\pi_{vid2}(\sigma_{vid1=v}(E)) \cup \pi_{vid1}(\sigma_{vid2=v}(E))$
4) 정점 v에 부속(incident)된 간선:
   $\sigma_{vid1=v \vee vid2=v}(E)$
5) 정점 v의 차수(degree):
   $\pi_{count(*)}(\sigma_{vid1=v \vee vid2=v}(E))$

다음은 핵심적인 그래프 알고리즘으로서 관계대수 연산과 C형태의 프로그램 구조로 정의된다.

6) 방향그래프에서 하나의 출발점에서 모든 목표점까지의 최단경로(shortest path) 는 다음과 같이 정의된다. 여기서는 Dijkstra 알고리즘[3]을 이용하였다.

```
Algorithm shortest_path(v)
{ /* D[vid, cost] contains the shortest distance to every
     vertex */
  n = π count(*)(V)  /* number of vertices */
  D[vid, cost] = {<v, 0>}  /* starting vertex */
  F[vid] = {}  /* found vertices */
  for ( i = 0; i < n - 1; i++) {
    <mincost> = π min(cost)(D − (D ⋈ F))
                          /* smallest not yet found */
    <minvid> = π vid( σ cost=mincost(D − (D ⋈ F))
    Dnew[vid, cost] = π vid2,ecost+mincost( σ vid1 minvid E)
                          /* new distances */
    D = vid Gmin(cost)(D ∪ Dnew)
                          /* select smaller distance */
    F = F ∪ {<minvid>}
  }
}
```

7) 최소비용 신장트리(minimum cost spanning tree)는 다음과 같이 정의된다. 여기서는 Kruskal 알고리즘[3]이 사용되었다.

```
Algorithm minimum_cost_spanning_tree()
{ /* Emst[vid1, vid2, ecost] contains the edges of the
```

```
    minimum spanning tree */
  n = π count(*)(V)  /* number of vertices */
  Emst[vid1, vid2, ecost] = {}
  Group[gid, vid] = π vid,vidV  /* each vertex is in
      different group, group is used for cycle-test */
  while ((π count(*)Emst < n-1) && π count(*)E > 0) {
    <v1, v2, mincost> = (π min(ecost)E) ⋈ E
                              /* least cost edge */
    E = E - {<v1, v2, mincost>}
                    /* delete it from the edge table */
    <g1> = π gid(σ vid=v1Group)  /* group of v1 */
    <g2> = π gid(σ vid v2Group)  /* group of v2 */
    if (<g1> != <g2>) { /* if the least cost edge
                          does not create a cycle */
      Emst = Emst ∪ {<v1, v2, mincost>}
        /* change the group g2 with the group g1 */
      Group = π gid←<g1>,vid(σ gid <g2>Group)
            ∪ (σ gid! <g2>Group)
    }
  }
}
```

이상에서 비록 핵심적인 그래프 연산과 알고리즘만 관계대수로 정의하였지만 대부분의 기존 연산과 알고리즘 뿐만 아니라 새로운 알고리즘도 위와 유사한 방법으로 정의될 수 있다.

4.2 그래프 알고리즘 구현

관계대수를 이용하여 정의된 알고리즘은 관계형 데이터베이스 상에서 쉽게 구현될 수 있다. 즉, 관계대수로 정의된 그래프 알고리즘은 SQL과 같은 데이터베이스 언어와 프로그래밍 언어를 함께 사용할 수 있는 내포 SQL(embedded SQL)을 이용하여 구현될 수 있다. 데이터베이스는 Oracle8을, 개발 언어는 C와 Pro*C를 사용하였다. 그림 4는 최단경로 알고리즘을 구현한 프로그램이다.

```
void shortest_path(v)
{
  int i, n;
  int mincost, minvid;

   /* connect to the graph database */
  EXEC SQL CONNECT :username IDENTIFIED BY
     :password;

   /* calculate the number of vertices */
  EXEC SQL select count(*) into :n from V;

   /* create the distance table D and initialize */
  EXEC SQL create table D (vid int, cost int);
  EXEC SQL insert into D values(:v, 0);

   /* create the found table F */
  EXEC SQL create table F (vid int);

  for (i = 0; i < n - 1; i++) {
   /* find smallest distance vertex among not yet found */
    EXEC SQL create table D1 (vid, cost)
             as (select vid, cost from D) minus
                (select vid, cost from D, F
                 where D.vid = F.vid);
    EXEC SQL select min(cost) into :mincost
                from D1;
    EXEC SQL select vid into :minvid from D1
                where cost = :mincost;

     /* recalculate new distances */
    EXEC SQL create table Dnew (vid int, cost int)
             as (select vid2, cost + :mincost
                 from E where vid1 = :minvid;

     /* select smaller distance */
    EXEC SQL create table D2 (vid, cost)
             as (select * from D) union
                (select * from Dnew);
    EXEC SQL create table D (vid, cost)
             as (select vid, min(cost) from D2
                 group by vid);

     /* add the vertex into the found table F */
    EXEC SQL insert into F values(:minvid);
  }
}
```

그림 4. 최단경로 알고리즘의 구현
Fig. 4. Implementation of shortest path algorithm

V. 결 론

본 연구에서는 그래프의 저장과 관리를 위하여 데이터베이스를 설계하고 라이브러리를 개발하였으며, 그래프 알고리즘을 데이터베이스에 기반하여 정의하고 구현하는 방법을 제안하였다.

즉, 그래프를 관계형 데이터모델의 릴레이션으로 모델링하여 정점과 간선을 이 릴레이션의 튜플로 데이터베이스에 저장하였다. 나아가 그래프에 대한 생성, 갱신, 삭제, 검색 등의 연산을 라이브러리화 함으로써 효과적으로 그래프를 실세계에 응용할 수 있도록 하였다. 또한, 그래프 연산과 알고리즘을 관계대수 연산을 이용하여 정의하였고 SQL과 같은 데이터베이스 언어를 사용하여 구현하였다.

이러한 데이터베이스에 기반한 방법은 메모리에 수용되지 않는 크기를 갖는 그래프를 효과적으로 처

리할 수 있을 뿐만 아니라 응용 프로그램 개발을 용이하게 할 것이다. 또한, 데이터베이스가 제공하는 기본적인 기능인 다중사용자에 의한 그래프의 동시 사용 등과 같은 장점도 가질 수 있다. 향후 연구과제는 객체지향 또는 객체관계 데이터 모델에 기반하여 그래프를 모델링하고 알고리즘을 개발하는 것이다.

참 고 문 헌

[1] R. Gould, Graph Theory, The Benjamin/Cummings Publiching, Menlo Park, CA. 1988.

[2] J. A. Mchugh, Algorithmic Graph Theory, Prentice-Hall, 1990.

[3] E. Horowitz, S. Sahni, and S. Anderson-Freed, Fundamentals of Data Structures in C, Computer Science Press, New York, 1993.

[4] A. Silberschatz, H. F. Korth, and S. Sudarshan, Database System Concepts, 3rd ed., McGraw-Hill, New York, 1997.

[5] C. J. Date, A Guide to The SQL Standard, Addison-Wesley, Reading, MA, 1989.

[6] D. E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, Addison-Wesley, 1993.

[7] Y. J. Chiang, M. T. Goodrich, E. F. Grove, and R. Tamassia, "External Memory Graph Algorithms," Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms, 1995.

[8] M. H. Nodian, M. T. Goodrich, and J. S. Vitter, "Blocking for External Graph Searching," Algorithmica, Vol.16, No.2, pp.181-214, Aug. 1996.

[9] J. Biskup, U. Rasch, and H. Stiefeling, "An Extension of SQL for querying Graph Relations," Comput. Lang. Vol.15, No.2, pp.65-82, 1990.

[10] R. H. Guting, "GraphDB: Modeling and Querying Graphs in Databases," Proc. of the 20th Conference on VLDB, pp.297-308, 1994.

[11] N. Kiesel, A. Schuerr, and B. Westfechtel, "GRAS, A Graph-Oriented (Software) Engineering Database System," Information Systems, Vol.20, No.1, pp.21-52, 1995.

[12] A. O. Mendelzon and P. T. Wood, "Finding Regular Simple Paths in Graph Databases," SIAM J. Comput. Vol.24, No.6, pp.1235-1258, Dec. 1995.

[13] L. Sheng, Z. M. Ozsoyoglu, and G. Ozsoyoglu, "A Graph Query Language and Its Query Processing," Proc. of 15th Conference on Data Engineering, pp.572-581, 1999.

저 자 소 개

박휴찬(Hyu-Chan Park)

1985년 서울대학교 전자공학과(공학사)

1987년 한국과학기술원 전기 및 전자공학과(공학석사)

1995년 한국과학기술원 전기 및전자공학과(공학박사)

1987년～1990년 금성반도체

1997년～현재 한국해양대학교 조교수

※관심분야 : 데이터베이스, 모델링방법론, CAD

추인경(In-Kyung Chu)

1998년 한국해양대학교 자동화·정보공학부(공학사)

2001년 한국해양대학교 컴퓨터공학과(공학석사)

2001년～현재 하우리

※관심분야 : 데이터베이스, 컴퓨터보안