관계형 데이타베이스에 기반한 그래프 알고리즘의 표현과 구현

(Representation and Implementation of Graph Algorithms based on Relational Database)

박 휴 찬 †

(Hyu-Chan Park)

요 약 그래프는 실세계의 많은 문제를 효과적으로 모델링하여 해를 구할 수 있는 강력한 방법을 제공하기 때문에 그래프의 표현 방법과 알고리즘 개발에 다양한 연구가 진행되어 왔다. 하지만, 대부분의 연구가 메인 메모리에 수용 가능한 크기를 갖는 그래프만을 고려하였기 때문에 큰 문제에 적용하기 위해서는 아직도 많은 어려움이 존재한다. 이를 극복하기 위하여 본 논문에서는 관계형 데이타베이스 이론에 기반하여 그래프를 표현하고 그래프 알고리즘을 정의할 수 있는 방법을 제안한다. 이 방법에서 그래프는 릴레이션으로 표현되며 그래프의 각 정점과 간선은 이 릴레이션의 튜플로서 저장된다. 이렇게 저장된 그래프에 대한 알고리즘은 추출, 선택, 죠인과 같은 관계대수 연산을 이용하여 정의되며 SQL과 같은 데이타베이스 언어를 사용하여 구현될 수 있다. 또한, 본 논문은 그래프의 저장 및 관리뿐만 아니라 다양한 응용프로그램 개발에도 사용될 수 있는 기본적인 그래프 함수들을 라이브러리화 하였다. 이와 같은 데이타베이스에 기반한 방법은 메모리에 수용되지 않는 크기의 그래프를 효과적으로 처리할 수 있는 방법을 제공할 뿐만 아니라 다양한 응용프로그램 개발을 용이하게 할 것이다. 또한, 데이타베이스가 제공하는 기본적인 기능인 다중사용자에 의한 동시공용 등과 같은 많은 장점을 가진다.

키워드 : 그래프, 관계대수, 데이타베이스

Abstract Graphs have provided a powerful methodology to solve a lot of real-world problems, and therefore there have been many proposals on the graph representations and algorithms. But, because most of them considered only memory-based graphs, there are still difficulties to apply them to large-scale problems. To cope with the difficulties, this paper proposes a graph representation and graph algorithms based on the well-developed relational database theory. Graphs are represented in the form of relations which can be visualized as relational tables. Each vertex and edge of a graph is represented as a tuple in the tables. Graph algorithms are also defined in terms of relational algebraic operations such as projection, selection, and join. They can be implemented with the database language such as SQL. We also developed a library of basic graph operations for the management of graphs and the development of graph applications. This database approach provides an efficient methodology to deal with very large-scale graphs, and the graph library supports the development of graph applications. Furthermore, it has many advantages such as the concurrent graph sharing among users by virtue of the capability of database.

Key words : graph, relational algebra, database

1. 서 론

그래프(graph) $G=(V, E)$는 정점(vertex)이라고 불리는 요소의 유한집합 V와 두 정점의 쌍으로 표현되는 간선(edge)이라고 불리는 요소의 유한집합 E로 구성된다. 간선이 방향을 나타내면 방향그래프(directed graph/digraph)라하고, 간선이 가중치를 가지면 가중치그래프(weighted graph)라 한다. 정점 u와 v를 끝점으로 갖는

· 본 연구는 한국과학재단 목적기초연구(R05-2001-000-00851-0) 지원으로 수행되었음.

† 정 회 원 : 한국해양대학교 기계·정보공학부 교수
hcpark@hhu.ac.kr

논문접수 : 2002년 5월 14일
심사완료 : 2002년 8월 28일

간선 e는 정점 u와 v에 부속(incident)된다고 하고, u와 v는 서로 인접(adjacent)인 정점이라고 한다. 정점 v의 차수(degree)는 v에 연결된 간선의 수로서 $deg(v)$라고 표시한다[1,2,3]. 이러한 그래프는 다양한 분야에서 복잡한 문제를 체계적으로 모델링하여 원하는 해를 구할 수 있는 아주 강력한 방법을 제공한다. 이러한 이유로 그래프를 일목요연하게 표현할 수 있는 방법과 효율적인 알고리즘의 개발에 많은 연구가 진행되어 왔다.

그래프를 표현할 수 있는 다양한 방법이 존재하지만, 인접행렬(adjacency matrix), 인접리스트(adjacency list), 인접다중리스트(adjacency multilist)가 실제 응용에서 가장 일반적으로 사용되고 있다[3]. 비록 이런 표현 방법이 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. 그래프의 양이 방대하여 메모리에 모두 상주할 수 없는 경우에는 위의 표현방법에 근거한 그래프 알고리즘을 사용할 수 없다. 이런 상황에서는 그래프의 일부분이 계속적으로 화일에 저장되고 화일로부터 메모리에 읽어들여져야 한다[4,5]. 이는 메모리에 수용되지 않은 크기의 데이타를 정렬하고자 할 때 사용하는 외부정렬 알고리즘에서 채택하는 방식과 유사한 것이다. 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 데이타베이스에 기반하여 그래프를 모델링하고 처리하려는 연구가 있어왔다. 즉, 그래프를 데이타베이스에 저장하기 위하여 기존의 데이타 모델을 확장하였으며, 저장된 그래프를 처리하기 위하여 기존의 질의어를 확장하고 그 질의를 효율적으로 처리하기 위한 질의처리 기법을 개발하는 것이었다[6,7,8,9,10,11]. 하지만, 이러한 연구는 그래프를 효율적으로 처리할 수는 있으나 기존의 데이타 모델이나 질의어를 그대로 사용할 수 없을 뿐만 아니라 표준 데이타베이스를 사용할 수 없기 때문에 호환성이나 이식성이 떨어지게 된다. 또한, 사용자의 입장에서 보면 새로운 데이타 모델이나 질의어를 익혀야 하는 부담을 지게 된다.

본 논문에서는 기존의 데이타 모델이나 질의어의 변경 없이 관계형 데이타베이스에 기반하여 그래프를 표현하고 그래프 알고리즘을 정의하는 방법을 제안한다. 이 방법에서 그래프는 릴레이션(relation)으로 표현되며 그래프의 각 정점과 간선은 이 릴레이션의 튜플(tuple)로서 저장된다. 또한, 각종 그래프 연산과 알고리즘은 추출(projection), 선택(selection), 죠인(join) 등과 같은 관계대수(relational algebra) 연산을 이용하여 정의된다. 이렇게 표현된 그래프와 알고리즘은 관계형 데이타베이스를 사용하여 구현될 수 있다. 이를 위하여 그래프 테이블을 설계하였으며 기본적인 그래프 연산을 라이브러리화하였다. 이러한 라이브러리를 이용하여 중요한 알고리즘을 구현하였으며 그 성능을 실험·분석하였다. 이러한 데이타베이스를 이용한 접근방법은 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.

본 논문의 구성은 다음과 같다. 2장에서 외부-메모리 그래프와 데이타베이스 그래프 등 메모리에 수용되지 않는 크기의 그래프를 처리하기 위한 관련 연구들을 살펴본다. 3장에서는 그래프를 릴레이션으로 표현하고 그래프 연산과 알고리즘을 관계대수를 이용하여 정의하는 방법을 제안한다. 4장에서 그래프를 데이타베이스에 저장하고 처리하는 기본적인 그래프 함수들의 라이브러리 개발에 관하여 기술한다. 또한, 이를 이용한 그래프 알고리즘의 구현 및 실험결과를 제시하고 이를 기존의 연구들과 비교 고찰한다. 5장에서는 결론과 향후 연구과제를 제시한다.

2. 관련 연구

그래프를 효율적으로 저장하고 처리하기 위한 많은 연구가 진행되어 왔다. 메인-메모리에 기반하여 그래프를 표현하기 위한 자료구조와 이를 처리하기 위한 알고리즘 및 라이브러리 개발, 외부-메모리에 기반하여 대용량의 그래프를 저장하고 처리하기 위한 연구, 데이타베이스에 기반하여 그래프를 저장하고 처리하기 위한 연구 등이 활발하게 진행되어 왔다. 이와 관련된 연구를 살펴보고 그 문제점을 고찰하기로 한다.

2.1 메인-메모리 기반 그래프 처리

그래프를 효율적으로 표현하기 위하여 인접행렬(adjacency matrix), 인접리스트(adjacency list), 인접다중리스트(adjacency multilist) 등의 다양한 자료구조가 제안되어 왔다. 또한, 이렇게 표현된 그래프를 처리하기 위하여 탐색(search), 최단경로(shortest path), 신

장트리(spanning tree) 등의 많은 연산과 알고리즘이 제안되어 왔다[1,2,3]. 나아가, 그래프 연산을 라이브러리화한 연구로서 Knuth[12]는 다양한 형태의 그래프를 조작하고 시험할 수 있는 데이타세트와 프로그램 라이브러리를 제안하였다. 즉, 그래프에 기반한 여러 가지 문제를 풀기 위한 예제 데이타 화일, 자료구조 및 입출력 등 커널 루틴, 기본적인 그래프 및 평면 그래프 등의 생성을 위한 그래프 생성 루틴, 최단 경로 및 최소신장 트리 등의 예제 루틴 등으로 구성된 그래프 라이브러리를 제안하였다.

비록 이런 표현 방법이 많은 응용분야에서 성공적으로 적용되어 왔지만 처리할 수 있는 그래프의 크기가 메모리의 크기로 제한된다. 즉, 그래프가 메모리에 모두 상주할 수 없는 경우에는 위의 방법을 적용할 수 없게된다.

2.2 외부-메모리 기반 그래프 처리

메인 메모리의 용량을 초과하는 크기를 갖는 그래프를 처리하기 위해서는 그래프의 일부분이 계속적으로 화일에 저장되고 화일로부터 메모리에 읽어들여져야 한다. 이는 메모리에 수용되지 않은 크기의 데이타를 정렬하고자 할 때 사용하는 외부정렬 알고리즘에서 채택하는 방식과 유사한 것이다. Chiang[4]은 그래프 문제에 적용 가능한 외부-메모리 알고리즘을 설계하는 방법들을 제안하였다. 또한, 이 기법들이 최소 신장트리(minimum spanning tree) 등과 같은 응용에 적용될 수 있음을 보였다. Nodian[5]은 외부-메모리 그래프를 처리함에 있어서 블록 전송을 최소화하기 위한 블록킹 방법에 관한 연구를 하였다.

하지만, 위의 연구들은 외부-메모리로 화일을 가정한 것으로서 그 효율성이 데이타베이스를 활용한 것보다 제한적일 수 있다. 또한, 여러 사용자가 동시에 그래프에 접근하여 수정 작업을 하게되면 일관성이 없는 그래프가 초래될 수도 있다.

2.3 데이타베이스 기반 그래프 처리

데이타베이스에 기반한 연구는 그래프의 저장을 위한 데이타 모델의 확장과 저장된 그래프의 처리를 위한 질의어의 확장 등에 관한 것이 주류를 이루고 있다. Biskup[7]은 기존의 관계형 질의어가 그래프에 관한 정보를 표현하는 표현력이 약하기 때문에 이를 보완하기 위하여 XSQL이라는 확장된 SQL을 제안하였다. 즉, 계통(genealogies), 네트워크(networks), 개념계층(concept hierarchies) 등 그래프 관계에 기반한 다양한 타입의 질의에 대한 분석을 한 후 이를 효율적으로 처리하기 위하여 기존의 SQL을 확장하였다. 특히, SQL과 같은 기존의 데이타베이스 언어는 경로(path)와 같은 회귀(recursion)적인 정보를 검색하는 데는 적합하지 않을 수 있기 때문에 이에 대한 질의가 가능하도록 하였다. 우선, 그래프 릴레이션에 대한 질의가 다음의 3단계로 구성됨을 파악하였다: 튜플 시퀀스(tuple sequence)인 경로들의 집합을 생성한다, 조건에 맞는 경로들을 선택한다, 원하는 출력을 추출한다. 이와 같은 과정은 기존 SQL의 "SELECT ··· FROM ··· WHERE ···"의 기본 구문과 일치하기 때문에 새로운 질의어를 제안하기보다는 SQL을 확장하여 사용하였다. 그 핵심 구문은 "SELECT project_list FROM PATHS path_expression WHERE select_condition"과 같다. 예를 들면, "genealogy[parent, child]"라는 테이블에서 "'Codd'의 모든 조상들을 찾아라."는 질의는 "SELECT START FROM PATHS OVER genealogy WHERE GOAL='Codd'"라는 XSQL 문장으로 표현된다. 이와 같이 확장된 질의어를 처리할 수 있도록 질의처리 과정 내부에 이행적 폐포(transitive closure) 및 관련된 연산을 추가하였으며 이 연산을 수행함에 있어서 페이지 액세스(page access)를 최소화하기 위한 기법들을 제시하였다.

Guting[8] 및 Sheng[9]은 객체지향 데이타베이스를 기반으로 그래프의 모델링과 질의가 가능한 데이타 모델과 질의어를 제안하였다. 그래프를 표현하기 위하여 정점, 간선, 경로 각각에 대응되는 단순 클래스(simple class), 링크 클래스(link class), 경로 클래스(path class)를 제안하였다. 그래프에 대한 질의를 위해서는 부그래프(sub-graph)를 참조할 수 있도록 기존 질의어를 확장하였다. Kiesel[10]은 속성그래프(attributed graph)를 저장하고 처리할 수 있는 전용의 데이타베이스 시스템과 전용의 언어를 제안하였다.

위의 연구들은 그래프를 데이타베이스화 하고 그 연산을 최적화하기 위하여 기존의 데이타베이스 질의어를 확장하든지 새로운 질의어를 제안하는데 주안점을 두었다. 하지만, 이와 같은 접근 방법은 표준 데이타베이스 시스템을 그대로 사용할 수 없기 때문에 호환성이나 이식성이 떨어지게 된다. 또한, 사용자의 입장에서 보면 새로운 데이타 모델이나 질의어를 익혀야 하는 부담을 지게 된다.

2.4 그래프 처리 방법 비교

이상에서 살펴본 바와 같이 그래프를 처리하기 위한 방법인 메인-메모리 기반, 외부-메모리(화일) 기반, 데이타베이스 기반 처리 방법의 장단점을 표 1에 정리하였다.

전통적인 방법인 메인-메모리에 기반한 처리 방법은 효율성이나 성능적인 측면에서 최선일 수는 있지만 처

표 1 그래프 처리 방법 비교

특징 \ 방법	메인 메모리 기반 [1,2,3]	화일 기반 [4,5]	데이타베이스 기반 [6,7,8,9,10,11]
그래프의 크기	메모리 크기로 제한	제한 없음	제한 없음
효율성(성능)	효율적	비효율적	다소 효율적
다중사용자에 의한 동시공용	어려움	어려움	가능
트랜잭션 지원	어려움	어려움	가능

리할 수 있는 그래프의 크기가 메인 메모리의 크기로 제한될 수밖에 없는 문제점을 내포하고 있다. 이러한 한계점을 극복하기 위한 한가지 대안은 화일을 사용하는 것이다. 이 방법에서는 그래프를 알맞은 표현 방법으로 화일에 저장해두고 프로세싱하는 동안에 필요한 부분 그래프를 읽어오고 처리된 결과를 다시 화일에 저장하는 등의 과정을 반복하게 된다. 이러한 화일 기반 방법은 처리 가능한 그래프의 크기는 제한이 없지만 처리하고자 하는 그래프의 크기가 커짐에 따라 효율성과 성능이 저하된다. 이는 일반적으로 화일을 이용한 프로세싱의 단점으로 알려져 있다. 이와 같은 단점을 극복하기 위한 데이타베이스에 기반한 처리 방법은 처리 가능한 그래프의 크기에 제한이 없을 뿐만 아니라 일반적으로 효율성의 측면에서도 화일보다 유리하다. 또한, 데이타베이스가 제공하는 기본적인 기능인 다중사용자에 의한 동시공용, 트랜잭션 처리 등의 장점을 얻을 수 있다.

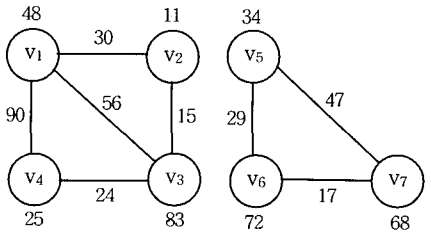
3. 관계대수를 이용한 그래프 알고리즘의 정의

이 장에서는 데이타 모델이나 질의어를 확장한 기존 연구와는 달리 표준의 관계형 모델을 이용하여 그래프를 표현하고 표준의 관계대수를 이용하여 그래프 연산이나 알고리즘을 정의하는 방법을 제안한다.

3.1 릴레이션을 이용한 그래프의 표현

그래프의 핵심적인 구성요소인 정점과 간선은 관계형 데이타 모델의 릴레이션으로 쉽게 표현될 수 있다. 릴레이션은 다음과 같이 정의된다. 도메인 D_1, D_2, $\cdots$, D_n의 카티전곱(cartesian product)은 $D_1 \times D_2 \times \cdots \times D_n$로 표기되며 n-튜플 $<v_1, v_2, \cdots, v_n>$의 집합이다. 여기서 v_1은 D_1의 원소이고 v_2는 D_2의 원소 등등이다. *릴레이션*은 주어진 도메인들의 카티전곱의 부분집합이고 릴레이션의 각 원소는 튜플이라 한다. 릴레이션은 테이블로 표현될 수 있으며 이 테이블에서 각 행은 튜플이고 각 열은 애트리뷰트로서 고유한 이름과 연관된 도메인을 가진다. 애트리뷰트의 집합 $A=\{A_1, A_2, \cdots, A_n\}$를 갖는 릴레이션 R은 $R[A]$ 또는 $R[A_1, A_2, \cdots, A_n]$로 표기한다. t가 $R[A]$의 튜플인 경우, 애트리뷰트 집합 $X \subseteq A$와 관련된 t의 부분은 $t[X]$로 표기한다[13].

이와 같은 릴레이션으로 그래프를 표현할 때, 그래프의 정점들은 *정점릴레이션*(*vertex relation*) $V[vid, v_attr]$으로 모델링되고, 간선들은 *간선릴레이션*(*edge relation*) $E[eid, vid1, vid2, e_attr]$으로 모델링된다. 예를 들면, 그림 1(a)의 그래프는 그림 1(b)의 릴레이션으로 표현되며 테이블의 형태로 가시화될 수 있다.



(a) 그래프 예제

V

vid	*v_attr*
v1	48
v2	11
v3	83
v4	25
v5	34
v6	72
v7	68

E

eid	*vid1*	*vid2*	*e_attr*
e1	v1	v2	30
e2	v2	v3	15
e3	v3	v4	24
e4	v4	v1	90
e5	v1	v3	56
e6	v5	v6	29
e7	v6	v7	17
e8	v7	v5	47

(b) 그래프 릴레이션

그림 1 릴레이션을 이용한 그래프 표현

정점릴레이션 V의 열(column) *vid*는 각 정점의 식별자(identifier)이다. 간선릴레이션 E의 열 *eid*는 각 간선의 식별자이고, *vid1*과 *vid2*는 이 간선에 연결된 정점을 나타낸다. 즉, *vid1*의 값이 v_1이고 *vid2*의 값이 v_2인 경우, 쌍 (v_1, v_2)는 정점 v_1과 v_2를 연결하는 간선을 의미한다. (v_1, v_2)는 무방향그래프(undirected graph)의 간선을 나타내고, $<v_1, v_2>$는 방향그래프(directed graph)의 간선을 나타낸다. 이러한 기본적인 열외에도 정점의 속성 *v_attr*과 간선의 속성 *e_attr* 등의 열도 추가적으로 고려할 수 있다. 이러한 추가적인 열은 주어진 문제를 모델링하는 그래프의 특성에 따라 다를 수 있다.

3.2 관계대수를 이용한 그래프 알고리즘의 정의

그래프를 릴레이션 형태로 표현한 경우 그래프에 대한 연산과 알고리즘은 관계대수 연산을 이용하여 정의되어질 수 있다. 관계대수는 다음과 같다. 릴레이션과

관련된 일련의 연산은 대수형 표기법을 사용하여 정의될 수 있으며 이를 *관계대수*(relational algebra)라 한다. 관계대수의 기본연산으로서는 추출(π), 선택(σ), 합집합(∪), 차집합(−), 카티젼곱(×)이 있다. 이러한 기본연산 외에 추가적인 연산으로서는 교집합(∩), 자연죠인($\bowtie$), 세타죠인($\bowtie_\theta$), 집성($\mathcal{G}$) 연산 등이 있으며 이들은 기본연산을 이용하여 정의할 수 있다[13].

본 논문에서 제시하는 그래프 연산과 알고리즘의 관계대수 표현에서, 정점릴레이션 V에서는 정점식별자 *vid*만을 고려하고 간선릴레이션 E에서는 *vid1*과 *vid2*만을 고려한다. 물론, 추가적인 다른 열을 고려하더라도 알고리즘의 핵심은 유사하게 된다. 다음의 정의 또는 알고리즘에서 특별한 언급이 없는 경우에는 무방향그래프를 가정한다.

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) 정점의 집합 S에 인접한 정점: $\pi_{vid2}(\sigma_{vid1\in S}(E)) \cup \pi_{vid1}(\sigma_{vid2\in S}(E))$

5) 방향그래프(digraph)에서 정점 v에 인접(adjacent to v)한 정점: $\pi_{vid1}(\sigma_{vid2=v}(E))$

6) 방향그래프(digraph)에서 정점 v로부터 인접(adjacent from v)한 정점: $\pi_{vid2}(\sigma_{vid1=v}(E))$

7) 정점 v에 부속(incident)된 간선: $\sigma_{vid1=v \vee vid2=v}(E)$

8) 정점 v의 차수(degree): $\pi_{count(*)}(\sigma_{vid1=v \vee vid2=v}(E))$

새로운 그래프가 결과로 생성되는 그래프 연산이 많지만[2], 그 중에서 중요한 몇 가지 연산을 관계대수를 이용하여 정의하면 다음과 같다.

9) 그래프 $G(V, E)$의 *complement* $\overline{G}(V', E')$는 정점의 집합으로 $V(G)$를 갖고 두 정점이 G에서 인접하지 않은 경우 $\overline{G}$에서 인접하는 그래프로서 다음과 같이 구할 수 있다.

V′ = V

E′ = (V × V) − E

10) 그래프 *G1(V1, E1)*과 *G2(V2, E2)*의 *union* $G(V, E)$ - $G1 \cup G2$로 표기 - 는 다음과 같이 구할 수 있다.

V = V1 ∪ V2

E = E1 ∪ E2

11) 그래프 *G1(V1, E1)*과 *G2(V2, E2)*의 *join* $G(V, E)$ - $G1+G2$로 표기 - 는 $G1 \cup G2$와 모든 *V1*과 *V2*를 서로 연결하는 간선을 포함하는 그래프이다.

V = V1 ∪ V2

E = E1 ∪ E2 ∪ (V1 × V2)

12) 그래프 *G1(V1, E1)*과 *G2(V2, E2)*의 *product* $G(V, E)$ - $G1 \times G2$로 표기 - 는 다음과 같이 정의된다. 정점의 집합 V는 $V1 \times V2$이다. 간선의 집합은 다음과 같이 정의된다. 임의의 두 정점 $u=(u_1, u_2)$와 $v=(v_1, v_2)$에 대해서 [$u_1=v_1$ and u_2 *adj* v_2]이거나 [$u_2=v_2$ and u_1 *adj* v_1]일 때 u와 v는 $G1 \times G2$에서 인접하다.

```
V[vid1, vid2] = V1 × V2
    /* where, a tuple (vid1, vid2) represents a
       vertex */
E[vid11, vid12, vid21, vid22] = V × V
    /* where, a tuple (vid11, vid12, vid21,
       vid22) represents an edge
       connecting the vertex (vid11, vid12) and
       (vid21, vid22) */
E = ((σ vid11=vid21 E) ⋈vid12=vid1∧vid22=vid2 E2)  /*
    [u1=v1 and u2 adj v2] */
    ∪ ((σ vid12=vid22 E) ⋈vid11=vid1∧vid21=vid2 E1)  /*
    [u2=v2 and u1 adj v1] */
```

13) 그래프 *G1(V1, E1)*과 *G2(V2, E2)*의 *composition* $G(V, E)$ - $G1[G2]$로 표기 - 는 다음과 같이 정의된다. 정점의 집합 V는 $V1 \times V2$이다. 간선의 집합은 다음과 같이 정의된다. 임의의 두 정점 $u=(u_1, u_2)$와 $v=(v_1, v_2)$는 [u_1 *adj* v_1]이거나 [$u_1=v_1$ and u_2 *adj* v_2]이면 인접하다.

```
V[vid1, vid2] = V1 × V2
E[vid11, vid12, vid21, vid22] = V × V
E = (E⋈vid11=vid1∧vid21=vid2 E1) /*[u1 adj v1]*/
    ∪ ((σ vid11=vid21 E) ⋈vid12=vid1∧vid22=vid2 E2)  /* [
    u1=v1 and u2 adj v2] */
```

다음은 핵심적인 그래프 알고리즘으로서 관계대수 연산과 C형태의 프로그램 구조로 정의된다.

14) *너비우선탐색*(*breadth first search*) 알고리즘은 다음과 같은 관계대수를 이용하여 정의된다.

```
Algorithm breadth_first_search(v)
{
  S[vid] = {v};  /* starting vertex */
  print S; Visited[vid] = S;
  while (V − Visited != ∅) {  /* exists not
  visited vertices */
```

```
    S = π vid2( σ vid1∈S(E)) ∪ π vid1 ( σ vid2∈S(E))/*
        next vertices */
    S = S − Visited; /* vertices not yet
        visited */
    print S; Visited = Visited ∪ S;
   }
 }
```

15) 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 lower distance
          */
    F = F ∪ {<minvid>}
  }
}
```

16) *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,vid V /* 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 관계대수를 이용하여 정의하였지만, 대부분의 기존 연산과 알고리즘 뿐만 아니라 새로운 알고리즘도 위와 유사한 방법으로 정의될 수 있다.

4. 관계형 데이타베이스에 기반한 구현

관계형 데이타모델에 기반하여 표현된 그래프와 알고리즘은 관계형 데이타베이스 상에서 쉽게 구현될 수 있다. 그래프를 표현하는 릴레이션은 테이블의 형태로 데이타베이스에 저장되고 검색될 수 있다. 관계대수를 이용하여 정의된 그래프 알고리즘은 SQL과 같은 데이타베이스 언어로 구현될 수 있다. 프로그래밍 언어의 특성과 함께 정의된 그래프 알고리즘은 프로그래밍 언어에 SQL문이 함께 사용될 수 있는 내포 SQL(embedded SQL)로 구현될 수 있다.

본 논문에서 제안한 개념을 구현하기 위하여 그래프의 각종 부가적인 정보를 다룰 수 있도록 테이블을 좀 더 상세하게 설계하였으며, 그래프를 처리하기 위한 기본적인 그래프 함수를 라이브러리의 형태로 개발하였다. 이 라이브러리는 다양한 다른 응용프로그램 개발에도 사용될 수 있다.

4.1. 그래프 테이블 상세 설계

그래프를 저장하기 위한 데이타베이스 테이블은 그래프에 대한 메타 정보를 저장하는 그래프정보 테이블(*graph_info*), 정점에 관한 정보를 저장하는 정점 테이블(*vertex*)과 정점속성 테이블(*vertex_att*), 간선에 관한 정보를 저장하는 간선 테이블(*edge*)과 간선속성 테이블(*edge_att*)로 구성된다.

1) 각 그래프의 메타 정보를 저장하는 테이블은 *graph_info*이다. 각각의 그래프를 고유하게 식별하는 그래프식별자(*g_id*)를 기본키로 하고, 그래프의 이름(*g_name*), 그래프에 대한 키워드(*key_word*), 그래프 작성시작날짜(*start_day*)와 최종수정날짜 (*update_day*), 기타정보(*ect*)로 구성된다.

[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) 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 여러개의 속성이 존재할 수 있기 때문이다. *vertex* 테이블은 정점을 고유하게 식별하는 정점식별자인 *v_id*, 정점이 소속된 그래프를 나타내는 *g_id*로 구성된다. 서로 다른 그래프에 동일한 정점식별자가 존재할 수 있으므로 이 정점들을 서로 구별하기 위해 (*g_id, v_id*)가 기본키의 역할을 한다. *vertex_att* 테이블은 속성의 이름을 나타내는 *vatt_name*, 속성의 값을 저장하는 *vatt_value*, 속성이 소속된 정점과 이 정점이 소속된 그래프를 나타내는 *v_id*와 *g_id*로 구성된다. 서로 다른 그래프와 정점에 동일한 속성이 존재할 수 있으므로 이 속성들을 서로 구별하기 위해 (*g_id, v_id, vatt_name*)가 기본키의 역할을 한다.

[vertex]

필드명	설 명
g_id	정점 v_id가 소속된 그래프의 그래프식별자, (기본키)
v_id	정점식별자, (기본키)

[vertex_att]

필드명	설 명
g_id	정점 v_id가 소속된 그래프의 그래프식별자, (기본키)
v_id	속성 vatt_name가 소속된 정점의 정점식별자, (기본키)
vatt_name	속성의 이름, (기본키)
vatt_value	속성의 값

3) 간선에 관한 정보를 저장하기 위한 테이블은 간선의 기본정보를 저장하는 *edge* 테이블과 간선에 부가된 각종 속성들과 그 값을 저장하는 *edge_att* 테이블로 구성된다. 이렇게 두개의 테이블로 분리한 이유는 한 간선에 여러 개의 속성이 존재할 수 있기 때문이다. *edge* 테이블은 간선을 고유하게 식별하는 간선식별자인 *e_id*, 간선이 소속된 그래프를 나타내는 *g_id*, 간선이 연결하는 두개의 정점을 나타내는 *vx_id*와 *vy_id*, 간선의 방향성 유무를 나타내는 *dir*로 구성된다. 서로 다른 그래프에 동일한 간선식별자가 존재할 수 있으므로 이 간선들을 서로 구별하기 위해 (*g_id, e_id*)가 기본키의 역할을 한다. *edge_att* 테이블은 속성의 이름을 나타내는 *eatt_name*, 속성의 값을 저장하는 *eatt_value*, 속성이 소속된 간선과 이 간선이 소속된 그래프를 나타내는 *e_id*와 *g_id*로 구성된다. 서로 다른 그래프와 간선에 동일한 속성이 존재할 수 있으므로 이 속성들을 서로 구별하기 위해 (*g_id, e_id, eatt_name*)가 기본키의 역할을 한다.

[edge]

필드명	설 명
g_id	간선 e_id가 소속된 그래프의 그래프식별자, (기본키)
e_id	간선식별자, (기본키)
vx_id	간선이 연결하는 두 개의 정점식별자
vy_id	
dir	간선의 방향성 유무 · true(y) : x → y, directed edge · false(n) : x — y, undirected edge

[edge_att]

필드명	설 명
g_id	간선 e_id가 소속된 그래프의 그래프식별자, (기본키)
e_id	속성 eatt_name가 소속된 간선의 간선식별자, (기본키)
eatt_name	속성의 이름, (기본키)
eatt_value	속성의 값

각 테이블의 상호 관계는 그림 2에서 보여준다. *vertex, vertex_att, edge, edge_att* 테이블의 *g_id*는 *graph_info* 테이블의 *g_id*를 참조하고, *vertex_att* 테이블의 *v_id*와 *edge* 테이블의 *vx_id, vy_id*는 *vertex* 테이블의 *v_id*를 참조하고, *edge_att* 테이블의 *e_id*는 *edge* 테이블의 *e_id*를 참조함을 알 수 있다.

4.2 그래프 처리를 위한 함수 라이브러리 개발

위의 테이블들은 데이타베이스 생성시에 만들어지게 되며, 각 그래프에 관한 정보를 이 테이블들에 저장하고 처리할 수 있는 기본 함수를 개발하여 라이브러리화 하였다. 라이브러리의 구성은 그래프, 정점, 간선, 정점속성, 간선속성 정보 각각에 대하여 생성, 갱신, 삭제, 검색 연산 등으로 구성된다.

1) 그래프 데이타베이스 연결/해제 함수인 *connect()/disconnect()*는 그래프에 대한 작업을 수행하기 위해 데이타베이스에 연결을 설정하고 작업 완료 후 연결을 해제하는 라이브러리이다.

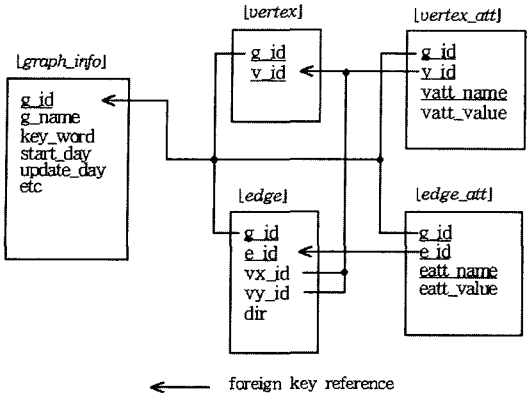


그림 2 그래프 테이블간의 상호 관련성

2) 그래프에 관한 정보를 삽입하기 위한 기능으로서, 그래프 생성 함수는 *create_graph()*이다. 정점에 관한 정보를 삽입하기 위하여 정점 자체를 삽입하는 함수 *insert_vertex()*, 정점에 부가된 속성들을 삽입하는 함수 *insert_vertex_attr()*는 함수 등이 있다. 간선에 관한 정보를 삽입하기 위하여 간선 자체를 삽입하는 함수 *insert_edge()*, 간선에 부가된 속성들을 삽입하는 함수 *insert_edge_attr()*는 함수 등이 있다.

3) 그래프에 관한 정보를 삭제하기 위한 기능으로서, 그래프 삭제 함수 *del_graph()* 는 *graph_info* 테이블에서 해당 그래프를 삭제하고, *vertex* 테이블, *vertex_att* 테이블, *edge* 테이블, *edge_att* 테이블에서 삭제된 그래프에 속하는 모든 튜플들도 삭제한다. 정점 삭제 함수 *del_vertex()*는 *vertex* 테이블에서 해당 정점을 삭제하고, *vertex_att* 테이블, *edge* 테이블에서 삭제된 정점을 참조하는 모든 튜플들도 삭제한다. 간선 삭제 함수 *del_edge()*는 *edge* 테이블에서 해당 간선을 삭제하고, *edge_att* 테이블에서 삭제된 간선을 참조하는 모든 튜플들도 삭제한다.

4) 그래프에 관한 정보를 갱신하기 위한 기능으로서, 그래프 갱신 함수 *update_graph_info()*는 그래프식별자를 기준으로 새로운 그래프식별자, 그래프이름, 키워드, 작업시작일, 최종수정일, 기타정보로 갱신할 수 있다. 그래프식별자를 갱신할 경우, *vertex* 테이블, *edge* 테이블, *vertex_att* 테이블, *edge_att* 테이블에 저장되어 있는 기존의 그래프식별자는 모두 새로운 그래프식별자로 갱신된다. 정점 갱신 함수 *update_vertex_info()*는 새로운 정점식별자, 속성이름, 속성값으로 갱신할 수 있다. 정점식별자를 갱신할 경우 *vertex* 테이블과 *vertex_att* 테이블, *edge* 테이블에서 사용중인 기존의 정점식별자는 모두 새로운 정점식별자로 갱신된다. 간선 갱신 함수 *update_edge_info()*는 새로운 간선식별자, 간선 연결정점, 방향성, 속성이름, 속성값으로 갱신할 수 있다. 간선식별자를 갱신할 경우 *edge* 테이블과 *edge_att* 테이블에 존재하는 기존의 간선식별자는 모두 새로운 간선식별자로 갱신된다.

5) 그래프 정보에 대한 검색 연산은 데이타베이스에 존재하는 테이블에 대하여 죠인 등의 연산을 통하여 이루어진다. 각 검색결과는 임시 테이블에 저장되며 사용자의 요구가 있을 때 한 행씩 읽어져 나오게 된다. 사용자는 검색된 결과에 대하여 다양한 그래프 알고리즘을 적용하여 좀 더 복잡한 작업을 수행할 수 있다. 그래프 검색 함수 *get_graph_info()*는 그래프식별자를 이용하여 *graph_info* 테이블에 저장되어 있는 전체 정보뿐만 아니라 각 속성에 대한 부분정보를 검색할 수 있다. 그리고 해당 그래프에 소속되어 있는 정점과 간선에 대한 정보도 검색할 수 있다.

정점 검색 함수 *get_vertex_ info()*은 그래프식별자와 정점식별자를 이용하여 *vertex* 테이블과 *vertex_att* 테이블에 저장되어 있는 해당 정점에 대한 데이타 검색이 가능하다. 정점의 속성정보를 검색할 경우 그래프식별자, 정점식별자, 속성이름을 이용하여 해당 속성값 검색이 가능하다. 간선 검색 함수 *get_edge_info()*는 그래프식별자와 간선식별자를 이용하여 *edge* 테이블과 *edge_att* 테이블에 저장되어 있는 해당 간선에 대한 데이타 검색이 가능하다. 간선의 속성정보를 검색할 경우 그래프식별자, 간선식별자, 속성이름을 이용하여 해당 속성값 검색이 가능하다. 그밖에 정점과 정점들 사이, 정점과 간선 사이의 관련성을 검색하기 위한 검색 라이브러리가 있다. 즉, 정점들 사이의 인접성 여부를 검색하는 인접성 검색과 정점에 연결된 간선의 개수를 검색하는 차수 검색 등이 있다.

6) 트랜잭션 처리 라이브러리는 일련의 그래프 연산이나 알고리즘 수행을 하나의 트랜잭션으로 처리할 수 있도록 지원한다. 복구 함수인 *rollback()*은 잘못 수행한 작업을 작업수행 이전상태로 복구한다. 저장점 지정 함수 *save_point()*로 저장점(save point)을 지정하면 그 이후 작업에 대해서 잘못된 작업을 수행을 했을 경우 복구위치 상태로 작업을 복구할 수 있다. 완료 함수인 *commit()*는 작업내용을 완료(commit)한다. 데이타베이스 연결을 해제할 때 묵시적으로 완료 작업을 수행하지만 명시적으로 완료할 수도 있다.

이상에서 개발한 그래프 함수 라이브러리와 이를 이용한 그래프 응용 등 전체적인 구성은 그림 3과 같다.

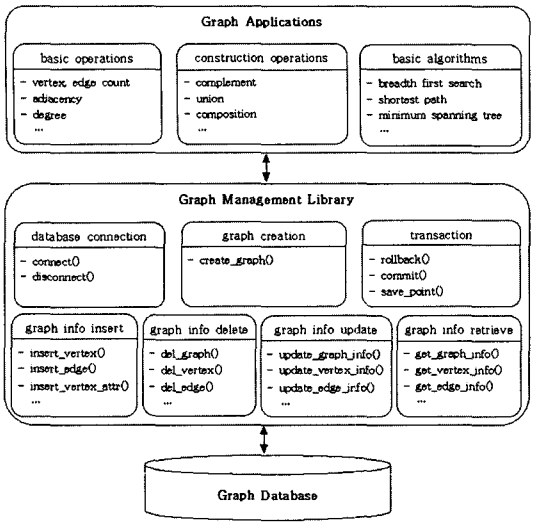


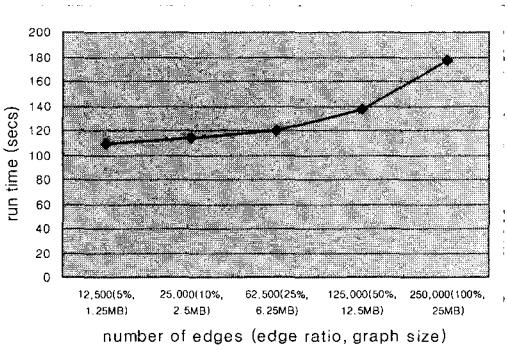
그림 3 그래프 함수 라이브러리 구성도

4.3 그래프 알고리즘 구현 및 실험

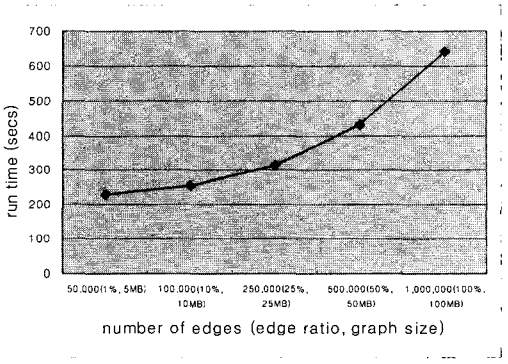
위에서 제안한 그래프 테이블과 그래프 기본 함수을 사용하여 관계대수로 표현된 그래프 알고리즘을 구현하고 그 성능을 실험하였다. 구현 환경으로 SUN Enterprise-450 컴퓨터와 Solaris-2.6 운영체제를 사용하였다. 데이타베이스 관리시스템은 Oracle-8을, 개발언어는 C와 Pro*C를 사용하였다. 실험에서는 대표적인 그래프 알고리즘인 최단경로 알고리즘과 최소비용 신장트리를 고려하였다.

그림 4는 최단 경로 알고리즘에 대한 실험 결과로서, 실험용 그래프는 방향 그래프로서 정점과 간선을 랜덤하게 생성하였다. 정점의 개수가 주어졌을 때, 간선의 개수에 따른 알고리즘 실행시간의 상관관계를 실험하였다. 즉, 정점의 개수가 500개(그림 4(a))와 1,000개(그림 4(b))인 두 가지 경우에 대하여 각각의 간선의 개수를 바꿔가면서 실험하였다. 각 정점으로부터 다른 모든 정점으로의 간선이 존재하는 완전 그래프를 100%의 간선 개수를 가지는 것으로 보고, 간선의 개수가 50%, 25%, 10%, 5%가 되도록 각 그래프를 생성하였다. 방향 그래프에서 완전 그래프는 정점이 n개인 경우 간선의 개수가 $n*(n-1)$이 된다. 각 간선의 크기는 100byte가 되도록 생성하였다.

메모리 기반 알고리즘에서는 정점의 개수가 정해진 경우 실행시간이 간선의 개수에 비례하지만, 그림 4(a)에서 알 수 있듯이 데이타베이스 기반 알고리즘에서는 실행시간이 간선의 개수에 비례하지는 않음을 알 수 있다. 예를 들면, 간선의 개수가 12,500개일 때 실행시간이 109초이었지만 간선의 개수가 그 20배인 250,000개일 때 177초에 불과하다. 이는 데이타베이스의 대용량 처리능력에 기인한다고 볼 수 있다. 이러한 효과는 그림 4(b)에서 알 수 있듯이 그래프의 사이즈가 상대적으로 더 커지면 감쇄되는 것을 알 수 있다. 또한, 간선의 개수가 동일하더라도 정점의 개수 많아지면 실행시간이 많이 걸림을 알 수 있다. 예를 들면, 그림 4(a)에서 간선의 개수가 250,000개인 경우 실행시간이 177초이지만 그림 4(b)에서는 313초이다. 이는 한 정점에 대하여 처리하기 위한 기본적인 오버헤드가 존재함을 알 수 있다.



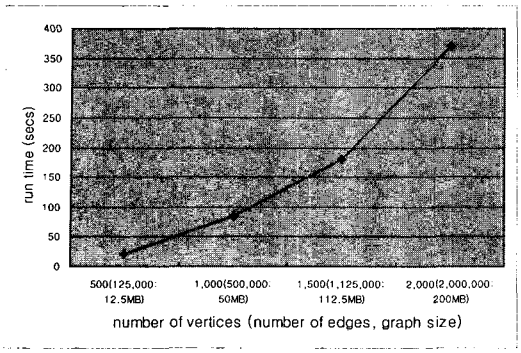
(a) 간선의 개수에 따른 실행시간 (정점: 500개)



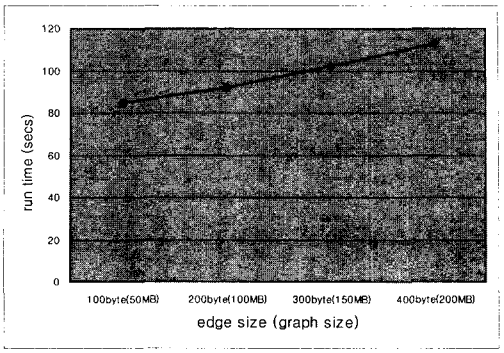
(b) 간선의 개수에 따른 실행시간 (정점: 1,000개)

그림 4 최단경로 알고리즘 실험 결과

그림 5는 최소비용 신장트리 알고리즘에 대한 실험결과로서, 실험용 그래프는 무방향 그래프로서 정점과 간선을 랜덤하게 생성하였다. 이 알고리즘은 그 특성상 정점의 개수가 정해졌을 때 간선의 비율이 어느 정도 이



(a) 정점 및 간선의 개수에 따른 실행시간



(b) 간선의 크기에 따른 실행시간
(정점: 1,000개, 간선: 500,000개)

그림 5 최소비용 신장트리 알고리즘 실험 결과

상이면 간선의 개수에 따른 실행시간은 별로 달라지지 않는다. 따라서, 본 실험에서는 정점의 개수를 500개에서 2,000개까지 바꿔가면서 완전그래프에 대하여 실험하였다 (그림 5(a)). 또한, 정점의 개수가 정해졌을 때 그래프의 크기에 따른 실행시간을 알아보기 위하여, 정점의 개수가 1,000개고 간선의 개수가 500,000개인 완전그래프에 대하여 하나의 간선의 크기를 100byte에서 400byte까지 바꿔가면서 실험하였다 (그림 5(b)). 무방향 그래프에서 완전 그래프는 정점이 n개인 경우 간선의 개수가 $n*(n-1)/2$이 된다.

그림 5(a)의 실험결과에서 알 수 있듯이 실행시간은 간선의 개수에 거의 비례함을 알 수 있다. 이는 메모리 기반과 마찬가지로 최소비용 신장트리 알고리즘이 간선 비용의 분포에 따라 실행시간이 달라지고 간선이 랜덤하게 생성되었기 때문이다. 또한, 그림 5(b)에서 알 수 있듯이 정점과 간선의 개수가 동일할 때, 하나의 간선의 크기(즉, 그래프의 크기)가 증가하더라도 실행시간은 소폭으로만 증가하는 것을 알 수 있다. 따라서, 데이타베이스에 기반한 알고리즘이 정점과 간선에 많은 속성이 부가된 대용량 그래프를 처리하는데 상대적으로 더 효율적임을 알 수 있다.

4.4 기존 연구와의 비교 및 검토

데이타베이스에 기반하여 그래프를 처리하는 기존의 방법과 본 논문에서 제안하는 방법의 특징 및 장단점을 표2에 정리하였다. 본 논문을 포함하여 많은 연구에서 관계형 또는 객체지향형 데이타베이스를 채택하였는데 이는 새로운 데이타베이스 시스템을 개발하는 것보다 이미 검증되고 최적화된 데이타베이스를 사용하는 것이 타당함을 알 수 있다.

표 2 데이타베이스에 기반한 그래프 처리 방법 비교

특징 \ 방법	Biskup[7]	Guting[8]	Kiesel[10]	제안 방법
DBMS	RDB	OODB	Special DB	RDB
질의어	XSQL(SQL extension)	Derive Statement (SQL like)	PROGRES	SQL
호환성, 이식성	나쁨	나쁨	나쁨	우수
라이브러리 지원	없음	없음	일부 지원	지원
효율성	효율적	효율적	효율적	다소 비효율적

기존의 연구들은 데이타베이스 질의어를 확장하든지 새로운 질의어를 제안하는데 주안점을 두었으며 이 질의어를 처리하기 위한 질의처리 방법을 제안하고 있다. 이와 같은 접근 방법은 그래프에 대한 표현력은 향상될 수 있겠지만 표준 데이타베이스 시스템을 그대로 사용할 수 없기 때문에 호환성이나 이식성이 떨어지게 된다. 또한, 사용자의 입장에서 보면 새로운 그래프 표현 방법이나 질의어를 익혀야 하는 부담을 지게 된다. 반면에 본 논문에서 제안한 방법은 데이타 모델이나 언어는 표준 데이타베이스 시스템에서 제공되는 기본 기능을 사용하기 때문에 이러한 문제점은 발생하지 않는다.

기존의 연구에서는 그래프 라이브러리에 대한 고려를 하지 않았지만, 본 연구에서는 그래프를 데이타베이스에 저장하고 처리하기 위한 기본적인 함수들을 라이브러리화 하였다. 이는 본 논문에서 정의한 알고리즘의 구현뿐만 아니라 응용 프로그램 개발에도 매우 유용하게 사용될 것이다.

다만, 기존의 연구에서 그 성능을 평가할 수 있는 데

이타나 알고리즘이 제시되어 있지 않아 정량적인 비교 평가는 할 수 없으나, 성능 측면에서의 효율성을 볼 때 기존의 연구에 비하여 다소 비효율적일 수 있다. 이는 표준 데이타베이스와 질의어를 사용함으로 얻는 장점에 따른 하나의 단점으로 볼 수 있다.

5. 결 론

본 논문은 그래프의 표현과 알고리즘을 정의하기 위한 방법을 제안하였다. 이 방법은 관계형 데이타모델과 관계대수에 기반하고 있다. 즉, 그래프는 관계형 데이타 모델의 릴레이션으로 모델링되며, 그래프의 각 정점과 간선은 이 릴레이션의 튜플로 표현된다. 또한, 그래프 연산과 알고리즘은 관계대수 연산을 이용하여 정의하였다. 이러한 방법은 관계형 데이타베이스에 기반하여 쉽게 구현될 수 있다. 이를 위하여 본 논문에서는 그래프의 저장을 위한 데이타베이스를 설계하였고, 그래프를 처리하기 위한 생성, 갱신, 삭제, 검색 등의 연산을 라이브러리화 함으로써 효과적으로 그래프를 실세계에 응용할 수 있는 방법을 제안하였다. 이를 이용하여 그래프 알고리즘을 구현하고 실험하였다.

이러한 데이타베이스 기반 방법은 메모리에 수용되지 않는 크기를 갖는 그래프를 효과적으로 처리할 수 있을 뿐만 아니라 그래프 라이브러리는 응용프로그램 개발을 용이하게 할 것이다. 또한, 데이타베이스가 제공하는 기본적인 기능인 다중사용자에 의한 그래프의 동시공용 등과 같은 많은 장점을 가진다.

향후 연구과제는 제안한 방법을 실제 응용분야에 적용하여 그 효용성을 검증하고 성능을 분석하는 것이다. 또한, 객체관계 또는 객체지향 데이타 모델에 기반하여 그래프를 모델링하고 알고리즘을 정의한 후 그 효율성을 본 연구와 상호 비교 평가할 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] 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.

[5] 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.

[6] M. V. Mannino and L. D. Shapiro, "Extensions to Query Languages for Graph Traversal Problems," IEEE Trans. on Knowledge and Data Engineering, Vol.2, No.3, pp.353-363, Sept. 1990.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

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

박 휴 찬

1985년 서울대학교 전자공학과(공학사). 1987년 한국과학기술원 전기및전자공학과(공학석사). 1995년 한국과학기술원 전기및전자공학과(공학박사). 1987년 ~ 1990년 금성반도체 연구원. 1997년 ~ 현재 한국해양대학교 기계·정보공학부 조교수. 관심분야는 데이타베이스, 모델링방법론, 그래프, UML.