(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2018년07월16일
(11) 등록번호 10-1878844
(24) 등록일자 2018년07월10일

(51) 국제특허분류(Int. Cl.)
G06F 17/30 (2006.01)
(52) CPC특허분류
G06F 17/30651 (2013.01)
G06F 17/30958 (2013.01)
(21) 출원번호 10-2016-0129170(분할)
(22) 출원일자 2016년10월06일
심사청구일자 2016년10월06일
(65) 공개번호 10-2016-0143600
(43) 공개일자 2016년12월14일
(62) 원출원 특허 10-2015-0079653
원출원일자 2015년06월05일
심사청구일자 2015년06월05일
(56) 선행기술조사문헌
KR1019990047339 A

(73) 특허권자
포항공과대학교 산학협력단
경상북도 포항시 남구 청암로 77 (지곡동)
서울대학교산학협력단
서울특별시 관악구 관악로 1 (신림동)
(72) 발명자
한욱신
경상북도 포항시 남구 청암로 77 창의IT융합공학과 (지곡동, 포항공과대학교)
이준영
대구광역시 수성구 청호로69길 30, 201동 1002호 (황금동, 가든하이츠2차아파트)
(뒷면에 계속)
(74) 대리인
특허법인이룸리온

전체 청구항 수 : 총 5 항 심사관 : 이복현

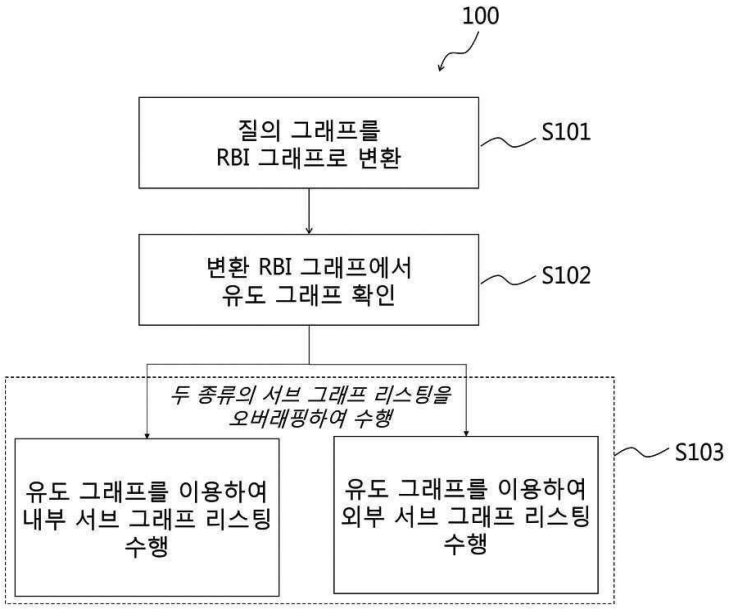
(54) 발명의 명칭 **중첩 오버래핑 기반의 서브 그래프 리스팅 방법**

(57) 요 약

위와 같은 과제를 해결하기 위한 본 발명의 일 측면에 따르면, 디스크에 저장된 데이터 그래프에 대하여 질의 그래프에 대한 서브 그래프 리스팅 방법에 있어서, 상기 디스크로부터 메모리의 내부 구역에 상기 데이터 그래프의 일부를 로드 하는 단계; 내부 메인 스레드가 상기 메모리에 로드 된 정점에 대하여 내부 서브 그래프 리스팅을

(뒷면에 계속)

대 표 도 - 도1



수행하는 단계; 외부 메인 스레드가 메모리의 피벗 구역 및 외부 구역에 데이터 정점의 로드를 추가적으로 요청하고 추가 스레드가 메모리에 로드 된 정점에 대하여 외부 서브 그래프 리스팅을 수행하는 단계;를 포함하되, 상기 내부 메인 스레드와 상기 외부 메인 스레드 및 상기 추가 스레드의 수행이 일정 기간 이상 오버랩 되어 동시에 수행되는 것을 특징으로 하는 서브 그래프 리스팅 방법이 제공된다.

본 발명의 서브 그래프 리스팅 방법에 의하면 데이터 그래프가 큰 경우라도, 디스크 접근 횟수를 최소화하여 데이터 그래프의 질의 그래프에 대한 서브 그래프 리스팅을 효율적으로 수행할 수 있다.

본 발명의 서브 그래프 리스팅 방법에 의하면, 질의 그래프를 R B I 그래프로 변환하고 후보 트리를 사용함으로써, 대규모의 데이터 그래프에 대하여 디스크의 I/O 횟수를 대폭 감소 시킬 수 있다.

본 발명의 서브 그래프 리스팅 방법은 메모리 버퍼를 내부, 피벗 및 외부 서브 구역으로 구별하고, 내부 서브 그래프 리스팅 및 피벗 구역과 외부 구역의 서브 그래프 리스팅을 멀티 스레드로 오버 래핑하여 독립적으로 수행할 수 있어 수행 속도를 크게 개선할 수 있다.

특히, 본 발명의 서브 그래프 리스팅 방법은 메인 스레드에 의한 디스크 I/O 시간 동안 추가 스레드로 외부 서브 그래프 리스팅을 오버래핑으로 동시에 수행할 수 있어 수행 속도를 개선하게 된다.

또한, 본 발명의 서브 그래프 리스팅 방법은 깊이 우선 탐색을 기반으로 한 R B I 매핑 방식을 이용하여 R B I 매핑 중간 결과를 저장하지 않아 저장 공간의 오버헤드 없이 효과적으로 R B I 매핑을 수행할 수 있다.

이와 같이, 본 발명의 서브 그래프 리스팅 방법은 멀티 스레드를 지원하여 스레드 증가에 따라 수행 속도가 비례하여 증가하게 된다.

(72) 발명자

김현지

울산광역시 동구 월봉12길 50, C동 308호 (화정동, 송정타워맨션3차)

이진수

경상북도 경산시 신천길 28 (점촌동)

이 발명을 지원한 국가연구개발사업
과제고유번호 2012M3C4A7033342
부처명 미래창조과학부
연구관리전문기관 한국연구재단
연구사업명 차세대정보컴퓨팅기술개발사업
연구과제명 소셜 및 정보 네트워크 빅데이터 마이닝 소프트웨어 원천 기술 개발
기 여 율 5/100
주관기관 서울대학교
연구기간 2014.07.01 ~ 2015.06.30
이 발명을 지원한 국가연구개발사업
과제고유번호 2015021977
부처명 미래창조과학부
연구관리전문기관 한국연구재단
연구사업명 중견연구자지원사업
연구과제명 신약 발견 및 광고 대행 추천을 위한 고성능 top-K 검색 엔진의 개발
기 여 율 25/100
주관기관 포항공과대학교 산학협력단
연구기간 2015.05.01 ~ 2016.04.30
이 발명을 지원한 국가연구개발사업
과제고유번호 IITP-2015-R0346-15-1007
부처명 미래창조과학부
연구관리전문기관 정보통신기술진흥센터
연구사업명 ICT명품인재양성사업
연구과제명 미래IT융합연구원
기 여 율 25/100
주관기관 포항공과대학교 산학협력단
연구기간 2015.01.01 ~ 2015.12.31
이 발명을 지원한 국가연구개발사업
과제고유번호 40011141
부처명 한국마이크로소프트(유)
연구관리전문기관 한국마이크로소프트(유)
연구사업명 일반산업체 과제
연구과제명 매시브 네트워크에서 효율적인 서브그래프 리스팅
기 여 율 45/100
주관기관 포항공과대학교 산학협력단
연구기간 2014.06.13 ~ 2015.06.12

명 세 서

청구범위

청구항 1

메인 스레드 및 추가 스레드를 사용하는 멀티 스레드 방식을 이용하여, 디스크에 저장된 데이터 그래프 중에서 질의 그래프와 동형인 서브 그래프를 리스팅하는 서브 그래프 리스팅 방법에 있어서,

상기 질의 그래프의 정점들 중에서 상기 디스크에서 메모리로 로딩된 데이터 그래프의 정점들과 직접 매칭해야 하는 복수 개의 제 1 정점을 인출하는 (a) 단계;

상기 인출된 복수 개의 제 1 정점 중 하나를 기준 정점으로 결정하는 (b) 단계;

상기 결정된 기준 정점과, 상기 메모리로 로딩된 데이터 그래프의 정점 및 상기 데이터 그래프의 정점이 속하는 디스크의 페이지를 매칭하는 (c) 단계; 및

상기 복수 개의 제 1 정점 각각이 기준 정점으로 결정될 때까지 상기 (b) 단계 및 상기 (c) 단계를 반복하는 (d) 단계를 포함하는 서브 그래프 리스팅 방법.

청구항 2

제 1 항에 있어서,

상기 (c) 단계는 상기 기준 정점과 매칭되는 상기 데이터 그래프의 정점들로 제 1 후보 정점 집합을 결정하고,

상기 기준 정점과 간선으로 연결된, 다른 제 1 정점과 매칭되는 상기 데이터 그래프의 정점들로 제 2 후보 정점 집합을 결정하는 단계를 포함하는 서브 그래프 리스팅 방법.

청구항 3

제 2 항에 있어서,

상기 제 1 후보 정점 집합 및 상기 제 2 후보 정점 집합 내 정점들과, 상기 질의 그래프의 정점들을 매칭하여 서브 그래프를 탐색하는 (e) 단계를 더 포함하는 서브 그래프 리스팅 방법.

청구항 4

제 1 항에 있어서,

상기 질의 그래프의 정점들을 상기 제 1 정점, 두 개 이상의 상기 제 1 정점의 인접 정보의 교집합을 통해 매칭하는 제2 정점 및 하나의 상기 제 1 정점의 인접 정보를 이용하여 매칭하는 제 3 정점으로 분류하는 (f) 단계를 더 포함하는 서브 그래프 리스팅 방법.

청구항 5

제 1 항에 있어서,

상기 메인 스레드는 비동기적으로 상기 디스크에 접근하여, 상기 디스크로부터 상기 메모리로 상기 데이터 그래프의 정점들을 로딩하고, 상기 추가 스레드는 상기 메인 스레드와 독립적으로, 상기 로딩된 데이터 그래프의 정점들을 이용하여 서브 그래프 리스팅을 수행하는 서브 그래프 리스팅 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 서브 그래프 리스팅 방법에 관한 것이다.

배 경 기 술

[0002] 서브 그래프 리스팅은 질의 그래프가 주어졌을 때 대규모 그래프 데이터로부터 해당 질의 그래프와 동형인 모든 서브 그래프를 나열하는 것이다.

[0003] 서브 그래프 리스팅은 그래프 데이터 마이닝 분야에서 그래프 분석을 위한 연산으로 널리 사용되고 있다.

[0004] 대규모 데이터 그래프 처리에 대한 연구는 데이터 그래프를 여러 대의 머신으로 분해하여 처리하는 분산 처리 방법에 대한 연구, 그리고 단일 머신을 활용하되 데이터 그래프를 대용량 디스크에 저장한 뒤 그 일부만 메모리에 로드 하여 처리하는 방법인 디스크 기반의 처리 방법에 대한 연구가 있다.

[0005] 본 발명에서는 단일 머신을 기반으로 서브 그래프 리스팅을 효율적으로 처리하는 디스크 기반의 방법에 초점을 맞춘다.

[0006] 메모리 기반 서브그래프 리스팅 알고리즘은 데이터 그래프가 메모리에 로드 되어 있다고 가정하기 때문에 데이터 그래프에 접근하는데 드는 비용을 크게 고려하지 않으며, 이를 그대로 디스크 기반 알고리즘으로 확장할 경우 많은 수의 랜덤 디스크 접근이 발생하게 되어 성능이 크게 저하된다.

[0007] 디스크 기반 환경에서 서브 그래프 리스팅을 효율적으로 수행하기 위해, 질의 그래프가 삼각형 형태인 경우에 한하여 효율적인 알고리즘이 제안되었다.

[0008] 이 알고리즘은 삼각형의 두 정점에 대응되는 데이터 정점들의 인접 리스트를 교집합 하여 나머지 한 정점에 대응하는 데이터 정점을 디스크 접근 없이 구하는 방법을 이용함으로써, 디스크에 저장된 데이터 그래프에서 삼각형을 찾는데 드는 디스크 접근의 횟수를 줄이는 방법이다.

[0009] 이 알고리즘은 멀티 스레드 환경에서 CPU 및 I/O 연산을 오버래핑하여 디스크 접근이 완료되는 동안 인접 리스트를 교집합 하는 CPU 연산을 수행함으로써 디스크에 저장된 데이터 그래프로부터 고속으로 삼각형을 찾을 수 있음을 보였다.

[0010] 그러나, 질의 그래프가 삼각형이 아닌 일반 그래프로 주어졌을 경우에는 이 디스크 기반 방법을 그대로 적용하기 어려운 문제가 있다.

[0011] 삼각형 형태의 질의를 처리하기 위해서는 데이터 그래프에서 간선으로 연결된 2개의 데이터 정점들의 인접 리스트 정보들을 얻어와야 하고, 인접 리스트 정보들을 얻어오기 위해서는 디스크로부터 해당 인접 리스트 정보가 저장된 페이지를 읽어 메모리에 로드 해야 한다.

[0012] 삼각형을 찾기 위해 두 개의 인접 리스트 정보를 디스크로부터 읽는 과정은 하나의 디스크 페이지를 메모리에 로드하는 과정, 그 페이지에 속한 정점 v와 간선으로 연결된 정점 v' 을 v 의 인접 리스트로부터 확인하는 과정, v' 가 속한 디스크 페이지 또한 메모리에 로드하고 v'의 인접 리스트를 얻는 과정으로 나뉜다

[0013] 마지막으로 v와 v' 의 인접 리스트를 교집합 함으로써 공통 인접 정점을 구하면 v와 v' 을 포함하는 삼각형을 모두 찾을 수 있다.

[0014] 따라서 디스크에 저장된 데이터 그래프에서 모든 삼각형을 찾기 위해서는 두 종류의 디스크 페이지, 즉 v가 속한 디스크 페이지와 v'가 속한 디스크 페이지를 바꾸어가면서 알고리즘을 시행하는 것이 필요하다.

[0015] 그러나, 일반적인 형태의 그래프가 질의로 주어지는 경우, 삼각형과는 달리 이러한 연결관계가 복잡한 형태로 주어지므로, 질의 정점들을 매핑 하기 위해 디스크에서 필요한 페이지를 얻어오는 과정을 공식화(formalize) 하기가 어려워진다.

[0016] 일반적인 질의 그래프의 질의 정점들을 매핑 하기 위해 디스크로부터 인접 리스트 정보를 읽어오는 과정은 2단계 이상의 중첩 반복(nested iteration)문으로 표현할 수 있다.

[0017] 질의 그래프를 처리하는 가장 단순한 방법은 디스크 페이지를 읽고 질의 그래프의 정점을 매핑 하는 과정을 순차적으로 처리하는 방법이다.

[0018] 그러나 이러한 방법은 질의 그래프의 매핑을 위해 디스크 페이지를 읽는 과정이 마무리 될 때까지 대기가 필요하므로 매우 비효율적이다.

[0019] 본 발명에서는, 디스크 기반 환경에서 삼각형을 효율적으로 찾는 방법을 확장하여 일반적인 형태의 질의 그래프를 처리하기 위해서 멀티 스레드 환경에서 CPU 및 I/O 연산을 오버레핑 하여 다중 중첩 반복문을 포함하는 서브

그래프 리스팅을 효율적으로 처리하는 방안을 고안하였다.

발명의 내용

해결하려는 과제

[0020] 상기와 같은 종래 기술의 문제점을 해결하기 위해, 본 발명은 중첩 오버레핑 개념을 도입하여 멀티 스레드에 의한 성능 향상이 뛰어난 서브 그래프 리스팅 방법을 제공하고자 한다.

과제의 해결 수단

[0021] 위와 같은 과제를 해결하기 위한 본 발명의 일 측면에 따르면, 디스크에 저장된 데이터 그래프에 대하여 질의 그래프에 대한 서브 그래프 리스팅 방법에 있어서, 상기 디스크로부터 메모리의 내부 구역에 상기 데이터 그래프의 일부를 로드 하는 단계; 내부 메인 스레드가 상기 메모리에 로드 된 정점에 대하여 내부 서브 그래프 리스팅을 수행하는 단계; 외부 메인 스레드가 메모리의 피벗 구역 및 외부 구역에 데이터 정점의 로드를 추가적으로 요청하고 추가 스레드가 메모리에 로드 된 정점에 대하여 외부 서브 그래프 리스팅을 수행하는 단계;를 포함하되, 상기 내부 메인 스레드와 상기 외부 메인 스레드 및 상기 추가 스레드의 수행이 일정 기간 이상 오버랩 되어 동시에 수행되는 것을 특징으로 하는 서브 그래프 리스팅 방법이 제공된다.

[0022] 상기 서브 그래프는 상기 내부 구역에 서브 그래프의 제 1 정점과 대응되는 모든 정점이 로드 되어 있는지 여부에 따라 내부 서브 그래프와 외부 서브 그래프로 구별되는 것을 특징으로 할 수 있다.

[0023] 상기 내부 서브 그래프 리스팅과 상기 외부 서브 그래프 리스팅을 오버랩 하여 수행하는 것을 특징으로 할 수 있다.

[0024] 내부 서브 그래프 리스팅은 내부 구역에 로드된 제 1 정점의 인접 리스트만으로 수행되며, 외부 서브 그래프 리스팅은 피벗 구역에 로드된 제 1 정점부터 리스팅을 시작하는 특징을 가진다.

[0025] 상기 외부 서브 그래프 리스팅에서 외부 메인 스레드는 디스크에 저장된 데이터 그래프의 일부에 대한 비동기적 읽기 작업을 요청하고, 상기 추가 스레드는 읽기가 완료된 데이터 그래프의 일부에 대해 독립적으로 외부 서브 그래프 리스팅을 수행하는 것을 특징으로 할 수 있다.

[0026] 상기 외부 서브 그래프 리스팅은 상기 내부 구역에 로드 된 정점 하나 이상, 상기 내부 구역 외부의 정점 하나 이상을 포함하는 것을 특징으로 할 수 있다.

[0027] 상기 외부 서브 그래프 리스팅은 후보 정점 집합 및 후보 페이지 집합을 이용하여 데이터 그래프로부터 어떤 디스크 페이지를 요청할지를 결정하는 것을 특징으로 할 수 있다.

[0028] 상기 외부 서브 그래프 리스팅은 지금까지 로드한 디스크 페이지로 후보 정점 집합 및 후보 페이지 집합을 갱신하고, 갱신한 후보 정점 집합 및 후보 페이지 집합으로 다음에 요청할 디스크 페이지 집합을 결정하는 것을 반복하는 것을 특징으로 할 수 있다.

[0029] 상기 외부 서브 그래프 리스팅은 후보 트리를 활용하여 수행되는 것을 특징으로 할 수 있다.

[0030] 상기 후보 트리는 질의 그래프의 상기 제 1 정점에 대응되는 정점을 트리 정점으로 갖고 있고, 질의 그래프에서 제 1 정점을 잇는 간선 중 일부를 트리 간선으로 갖고 있는 것을 특징으로 할 수 있다.

[0031] 상기 후보 트리의 각각의 트리 노드는 후보 정점 집합 및 후보 페이지 집합을 속성으로 가지고 있는 것을 특징으로 할 수 있다.

[0032] 상기 후보 트리는 하나의 상기 제 1 정점을 기준 정점으로 가지며, 기준 정점은 트리 구조에서 루트 정점으로 표현되는 것을 특징으로 할 수 있다.

[0033] 상기 후보 트리에서 두 정점의 부모와 자식 관계는 두 정점에 대응 하는 질의 정점의 후보 정점 집합에 속하는 후보 정점 간의 연결 관계를 명시하는 것을 특징으로 할 수 있다.

[0034] 상기 서브 그래프 리스팅은 질의 그래프에서 제 1 정점의 개수만큼 다른 후보 트리를 만들고 각각의 후보 트리를 사용한 외부 서브그래프 리스팅을 순차적으로 진행함으로써 모든 외부 서브그래프를 찾는 것을 특징으로 할 수 있다.

[0035] 상기 후보 트리의 기준 정점에 해당하는 후보 정점 집합은 피벗 정점 집합으로 정의되는 것을 특징으로 할 수

있다.

[0036] 상기 후보 트리를 사용한 외부 서브 그래프 리스팅의 과정은 a) 상기 제 1 정점 중에서 기준 정점을 결정하는 단계; b) 상기 기준 정점을 기준으로 상기 제 1 정점들과 대응하는 후보 정점들을 포함하는 디스크 페이지를 로딩하고 매칭하는 단계; c) 상기 모든 제 1 정점을 기준 정점으로 고정하면서 상기 a) 및 b) 단계를 반복하는 단계를 포함하는 것을 특징으로 할 수 있다.

[0037] 상기 단계 b)의 세부 단계는 상기 기준 정점에 매칭할 후보 정점 집합으로 피벗 정점 집합을 결정하는 단계, 피벗 후보 정점의 인접 리스트를 스캔 하여 상기 기준 정점과 간선으로 연결된 다른 제 1 정점에 대하여 매칭될 후보 정점 집합을 결정하는 단계, 그리고 매칭 순서를 따라 나머지 기준 정점에 매칭될 후보 정점 집합을 결정하는 단계를 포함하는 것을 특징으로 할 수 있다.

[0038] 상기 후보 트리에 속하는 모든 제1 정점에 매칭되는 후보 정점들이 결정되고, 필요한 디스크 페이지가 모두 메모리에 로드되면, 후보 정점들이 속한 서브 그래프를 탐색하여 나머지 질의 정점들의 매칭을 완료하는 것을 특징으로 할 수 있다.

[0039] 외부 서브 그래프 리스팅 진행 중 특정 질의 정점의 상기 후보 정점집합에 속한 정점들을 상기 메모리 구역에 한번에 로드 할 수 없는 경우, 상기 메모리 구역의 크기에 맞게 상기 후보 정점 집합을 둘 이상의 부분 집합으로 구분하는 단계; 상기 후보 정점 집합의 어떤 부분 집합이 속하는 디스크 페이지를 메모리에 로드하고 매칭 순서에 따라 다음 외부 서브 그래프 리스팅을 계속 진행하는 단계; 상기 후보 정점 집합의 부분 집합 중 아직 처리하지 않은 정점들이 속한 디스크 페이지를 메모리에 로드하고 위 과정을 다시 반복하는 단계;를 포함하는 것을 특징으로 할 수 있다.

[0040] 여기서 상기 메모리의 구역은 내부 구역, 피벗 구역, 및 외부 구역으로 분류되며, 상기 내부 구역의 정점과 연결된 정점들 중에서 정점 번호가 가장 작은 정점을 상기 피벗 구역에 로드 하는 것을 특징으로 할 수 있다.

발명의 효과

[0041] 본 발명의 서브 그래프 리스팅 방법은 메모리 구역을 내부, 피벗 및 외부 서브 구역으로 구별하고, 내부 서브 그래프 리스팅과 피벗 구역 및 외부 구역의 정점을 사용하여 수행되는 외부 서브 그래프 리스팅을 멀티 스레드로 오버래핑하여 독립적으로 수행할 수 있어 수행 속도를 크게 개선할 수 있다.

[0042] 특히, 본 발명의 서브 그래프 리스팅 방법은 메인 스레드에 의한 디스크 I/O 시간 동안 추가 스레드로 외부 서브 그래프 리스팅을 오버레핑으로 동시에 수행할 수 있어 수행 속도를 개선하게 된다.

[0043] 이와 같이, 본 발명의 서브 그래프 리스팅 방법은 멀티 스레드를 지원하여 스레드 증가에 따라 수행 속도가 비례하여 증가하게 된다.

도면의 간단한 설명

[0044] 도 1은 본 발명의 한 실시예에 따른 서브 그래프 리스팅 방법을 개략적으로 보여준다.

도 2는 본 발명의 한 실시예에 따른 질의 그래프 및 질의 그래프의 변환된 그래프인 RBI 그래프를 나타낸다.

도 3은 본 발명의 한 실시예에 따른 데이터 그래프 상의 정점 방문 순서를 나타낸다.

도 4는 본 발명의 한 실시예에 따른 후보 트리에 대해 나타낸다.

도 5는 본 발명의 한 실시예에 따른 질의 그래프 및 질의 그래프의 부분 순서 집합을 나타낸다.

도 6은 본 발명의 한 실시예에 따른 서브 그래프 리스팅의 스피드 업 결과를 나타낸다.

도 7은 본 발명의 한 실시예에 따른 서브그래프 리스팅 방법을 실제 데이터 셋에 대하여 수행한 시간을 나타낸다.

발명을 실시하기 위한 구체적인 내용

[0045] 본 발명의 서브 그래프 리스팅 방법은 내부 메인 스레드, 외부 메인 스레드, 추가 스레드를 사용하는 멀티 스레드 방식을 이용하여 내부 서브그래프 리스팅과 외부 서브그래프 리스팅을 오버랩하여 수행할 수 있다. 본 발명에서 내부 서브 그래프 리스팅은 메모리의 내부 구역에 로드된 데이터 그래프의 정점에 대하여 서브 그래프 리스팅을 수행하는 것이며, 외부 서브 그래프 리스팅은 메모리의 내부 구역에 로드된 정점을 하나 이상 포함하면

서 피벗 구역 혹은 외부 구역에도 정점을 하나 이상 포함하는 서브 그래프를 리스팅 하는 것을 의미한다.

[0046] 본 발명의 서브 그래프 리스팅 방법은 내부 메인 스레드에서 내부 서브 그래프 리스팅을 완료하면, 내부 메인 스레드가 추가 스레드 역할을 함으로서 서브그래프 리스팅 알고리즘이 CPU의 코어를 항상 최대로 활용할 수 있도록 한다.

[0047] 본 발명의 서브 그래프 리스팅 방법은 외부 메인 스레드가 외부 서브그래프 리스팅의 핵심 작업을 수행하지 않고 필요한 데이터를 디스크로부터 요청함으로써, 장시간이 소요되는 데이터 접근 시간 동안 추가 스레드가 이미 메모리에 로드된 다른 데이터를 사용해서 외부 서브 그래프 리스팅을 수행하므로 결과적으로 CPU 시간과 디스크 접근 시간이 오버랩이 되어 작업 속도가 비약적으로 개선될 수 있게 된다.

[0048] 이하, 첨부한 도면을 참고로 하여 본 발명의 실시예에 대하여 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 상세히 설명한다. 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시예에 한정되지 않는다. 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 동일 또는 유사한 구성요소에 대해서는 동일한 참조부호를 붙였다.

[0049] 먼저, 본 발명에서 서브 그래프 리스팅은 데이터 그래프 $G_D = (V_D, E_D)$와 질의 그래프 $G_Q = (V_Q, E_Q)$의 두 그래프가 주어졌을 때, 데이터 그래프 G_D의 서브 그래프 $G_0 = (V_0, E_0): V_0 \subseteq V_D, E_0 = E_D \cap (V_0 \times V_0)$ 가운데 $G_0 \cong G_Q$(본 발명에서 '동형'이라고 지칭하기도 한다)인 모든 G_0를 찾는 것이다.

[0050] 이때, $G \cong G'$은 다음과 같이 정의된다.

[0051] 두 그래프 $G = (V, E)$와 $G' = (V', E')$에서 V와 V'의 정점이 서로 일대일 대응이 가능하고, $v, u \in V$와 대응되는 정점 $v', u' \in V'$에 대해 $(v', u') \in E'$이면 $(v, u) \in E$ 이고 그 역도 성립하는 경우 $G \cong G'$이다.

[0052] 이하, 도 1 내지 도 6을 참조하여, 본 발명의 서브 그래프 리스팅 방법(100)을 상세히 설명하도록 한다.

[0053] 본 발명의 서브 그래프 리스팅 방법(100)은 도 1에 도시된 바와 같이, 질의 그래프를 RBI 그래프로 변환하는 단계(S101), 변환된 RBI 그래프에서 유도 그래프를 확인하는 단계(S102) 및 데이터 그래프에 대하여 유도 그래프를 이용하여 서브 그래프를 탐색을 수행하는 단계(S103)를 포함한다.

[0054] 먼저, 질의 그래프를 RBI 그래프로 변환한다(단계 S101).

[0055] 본 발명의 질의 그래프 변환 방법에 의하면, 질의 그래프 G_Q의 정점들을 직접 접근 여부에 따라 분류한다.

[0056] 즉, 질의 그래프 G_Q의 정점들을 메모리에서 직접 매핑(Retrieval) 해야 하는 제 1 정점, 제 1 정점들의 인접 정보를 교집합(Intersection)하여 매핑 가능한 제 2 정점, 및 제 1 정점의 인접 정보를 바로 읽어(Browsing) 매핑 가능한 제 3 정점으로 분류한다.

[0057] 본 실시예에서는 제 1 정점을 R (Red; 레드) 정점으로, 제 2 정점을 B (Black; 블랙) 정점, 그리고 제 3 정점을 I (Ivory; 아이보리) 정점으로 정의한다.

[0058] 그에 따라, 질의 그래프 G_Q 내 모든 정점은 R 정점, B 정점, 및 I 정점으로 분류되며, 이와 같이 질의 그래프 G_q의 정점들을 R 정점, B 정점, 및 I 정점으로 분류하여 매칭하는 것을 RBI-Match라 정의한다.

[0059] 이러한 RBI-Match에 따라, 질의 그래프의 모든 정점은 R 정점, B 정점, 및 I 정점으로 표현되고, 이와 같이 R 정점, B 정점, 및 I 정점으로 표현된 질의 그래프를 RBI 그래프로 지칭한다.

[0060] 먼저, 도 2(a)의 질의 그래프 G_Q의 정점들 중에서 R 정점, 즉 V_{red}를 결정한다.

[0061] 이때, R 정점 이외의 정점은 항상 R 정점과 인접하고, R 정점이 아닌 두 정점을 잇는 간선은 없도록 질의 그래프 G_Q의 정점들 중에서 V_{red}를 결정한다.

[0062] 그리고, V_{red}가 결정되면 V_{red}를 제외한 정점에 대해 인접한 R 정점의 개수가 2개 이상인 경우 I 정점, 1개인

경우 B 정점으로 분류한다.

[0063] 그에 따라, 도 2 (b)에 도시된 바와 같이, 질의 그래프 G_Q의 정점들 중 u_1, u_2, u_3는 R 정점, u_4, u_5는 I 정점, u_6는 B 정점이 된다.

[0064] 다음으로, 변환된 RBI 그래프에서 유도 그래프를 생성한다(단계S102).

[0065] 즉, RBI 그래프에서 V_{red} 정점으로 이루어진 유도 그래프(induced graph) $G_{red}=(V_{red}; E_{red})$를 생성한다. 이때, G_{red}는 V_{red} 정점과 V_{red} 정점간의 간선(Edge) E_{red}로 이루어진다.

[0066] 본 발명에서 유도 그래프 G_{red}는 외부 서브 그래프 리스팅에 특히 바람직하게 이용된다. 본 발명에서는 유도 그래프 G_{red}의 간선 정보를 이용하여 내부 및 외부 서브 그래프 탐색을 수행한다(단계 S103).

[0067] 도 3에 도시된 실시예를 참조하여 서브 그래프 탐색을 설명하면, 가장 먼저 질의 그래프의 u_1과 데이터 그래프의 데이터 정점, 예를 들어 v_1을 매핑하고, 다음에는 질의 그래프의 u_1과 인접한 u_2에 매칭을 시도한다.

[0068] 이때, v_1의 이웃한 정점 집합인 $n(v_1)$이 u_2와 매핑 가능한 후보 집합이 된다. 그리고, 질의 그래프의 u_2에 대응하는 데이터 그래프의 데이터 정점, 예를 들어 v_2가 매핑되면, 질의 그래프의 u_2와 인접한 u_3의 매칭을 시도한다. 그리고, 이러한 방법으로 모든 유도 그래프의 정점이 매칭되면, 매핑된 데이터 정점 v를 이용하여 나머지 질의 그래프의 정점 u_4, u_5, u_6에 대한 데이터 정점에 대한 정보를 수득하는 방법으로 서브 그래프를 탐색하는 것이다.

[0069] 한편, 본 발명의 서브 그래프 리스팅 방법은 2-레벨의 오버래핑을 통해 디스크 I/O 시간과 CPU 시간을 오버랩할 수 있다.

[0070] 먼저, 매크로 레벨에서 멀티 스레딩 기법을 이용해 디스크 I/O가 일어나지 않는 내부 서브 그래프 리스팅과 디스크 I/O가 빈번히 일어나는 외부 서브 그래프 리스팅을 오버랩 한다.

[0071] 다음으로 마이크로 레벨에서는, 메인 스레드와 추가 스레드 두 가지 종류의 스레드를 이용하여 외부 서브 그래프 리스팅의 디스크 읽기 작업과 CPU 연산을 오버랩 한다.

[0072] 메인 스레드는 추가로 디스크 읽기 작업이 필요할 때마다 비동기적으로 디스크에 읽기 작업을 요청한다.

[0073] 디스크 읽기 작업이 완료될 때마다 추가 스레드가 콜백 함수의 수행을 시작한다. 콜백 함수는 두 가지 유형이 있는데, 하나는 피벗 후보 집합 혹은 외부 후보 집합을 형성하는 작업을 수행하고 다른 하나는 외부 서브 그래프를 찾는 작업을 수행한다. 두 함수 모두 외부 서브 그래프의 핵심 작업이며, 따라서 외부 서브 그래프 리스팅의 핵심 작업은 추가 스레드에서 수행한다.

[0074] 추가 스레드가 외부 서브 그래프 리스팅의 CPU 작업으로 이루어진 콜백 함수를 수행하는 동안에 비동기적으로 디스크에 접근하여 읽기 작업을 수행할 수 있다.

[0075] 즉, 비동기적으로 디스크에서 데이터를 읽으므로 요청에 대한 응답을 기다리지 않고 다른 작업을 진행할 수 있게 되며, 이에 따라 CPU 작업과 디스크 읽기 작업을 동시에 수행할 수 있다.

[0076] 이러한 메인 스레드에서 수행하는 본 발명의 서브 그래프 탐색에 대한 전체 프로세스는 알고리즘 1과 같다.

[0077] 알고리즘 1 (RBI-Match)

```
Input: query graph G_Q, partial orders PO
⟨G_RBI, G_red⟩ ← RewriteRBIQueryGraph(G_Q);
initialize the bitmap sets in CandidateTree
while ( there are more chunks to process) do
    C_cur ← CaculateNextChunkRange(m_int);
    PinPagesAndFillCandInfo(C_cur, −1);        /* −1 : Current Index on G_red */
    DelegateExtSubgraphListing(G_RBI, G_red, C_cur, CT, PC);  /* ext. listing */
    IntSubgraphListing(G_RBI, G_red, C_cur);                  /* int. listing */
    pool.Sync();
    UnpinPages(C_cur);
end
Function PinPagesAndFillCandInfo
    Input: C_cur, t_last : Current Index on G_red
    foreach ( pid ∈ P(C_cur)) do
        AsyncRead(pid, UpdateCandidates(pid, CT, t_last));
    end
    wait until all invocations of UpdateCandidates executions are finished
end
```

[0078]

[0079] 알고리즘 1에 도시된 바와 같이, 먼저, 초기화 과정에서는 입력 받은 질의 그래프 G_Q를 RBI 그래프로 변환하고, 외부 서브 그래프 리스팅에서 이용하는 자료구조를 초기화한다.

[0080] 그 다음, 내부구역의 크기만큼 데이터 그래프에서 청크를 로드한 뒤, 외부 서브 그래프 리스팅과 내부 서브 그래프 리스팅을 병렬로 수행한 뒤, 두 작업이 모두 끝나면 청크를 메모리에서 해제한다.

[0081] 본 알고리즘은 데이터 그래프에서 처리하지 않은 청크가 더 이상 없을 때까지 반복적으로 위 과정을 수행한다.

[0082] 본 발명의 서브 그래프 리스팅은 중첩 오버래핑 (nested overlapping)을 이용하여 외부 서브 그래프 리스팅을 효율적으로 처리하게 된다.

[0083] 구체적으로, 외부 서브 그래프 리스팅 과정에서 각 질의 그래프 정점의 매핑 후보 집합을 중첩 반복문을 통해 형성할 때, 매 단계마다 디스크 I/O와 CPU 연산을 오버랩 하는 것이다.

[0084] 이하, 도 4을 참조하여 본 발명에 따른 중첩 오버래핑에 의한 외부 서브 리스팅 알고리즘을 보다 상세히 설명하도록 한다.

[0085] 먼저, 질의 그래프에 대하여 외부 서브 그래프 리스팅을 위한 후보 트리(candidate tree: 이하, 간단히 'CT'라 지칭하기도 한다)를 생성한다.

[0086] 후보 트리는 유도 그래프 G_{red}와 매칭되는 후보 정점 집합과 후보 페이지 집합을 저장하는 자료 구조로서, 본 발명의 서브 그래프 리스팅, 특히 외부 서브 그래프 리스팅에서 메모리 구역에서 로드할 정점 후보 집합과 해당 정점 후보 집합이 속하는 페이지를 로드하는 기준으로 이용된다.

[0087] 유도 정점 u_i의 후보 정점 집합이란 u_i에 매핑될 가능성이 있는 데이터 정점의 집합으로 정의된다.

[0088] 유도 정점 u_i의 후보 페이지 집합이란 u_i에 매핑될 가능성이 있는 데이터 정점을 포함하고 있는 디스크 페이지의 집합으로 정의된다.

[0089] 후보 트리의 트리 구조는 유도 그래프 G_{red}의 신장 트리(spanning tree)로 정의된다.

[0090] 하나의 후보 트리는 하나의 기준 정점을 가지고 있으며 이 기준 정점은 후보 트리에서 루트 노드 (root node)로 표현된다.

[0091] 후보 트리는 유도 그래프 G_{red}의 정점들 중에서 기준 정점을 어떤 정점으로 하느냐에 따라 각각 다른 형태로 구성된다.

[0092] 본 서브 그래프 리스팅 방법은 G_{red}의 정점들을 각각 기준 정점으로 하는 총 $|V_{red}|$개의 다른 후보 트리들을 먼저 생성한 다음에 서브 그래프 리스팅을 수행한다.

[0093] 서브그래프 리스팅을 수행하기 이전에 후보 트리들의 후보 정점 집합과 후보 페이지 집합은 공집합으로 초기화되어 있다고 가정한다. 그리고, G_{red}의 정점을 기준 정점으로 고정하여 후보 트리를 결정하더라도, 후보 트리에 포함되지 않았으나 유도 그래프 G_{red}에서는 존재하는 나머지 간선들을 사용하면 이것을 후보 정점 집합의 사이즈를 줄이는데 활용할 수 있기 때문에, 이들을 비트리 간선(non-tree edge)의 형태로 추가적으로 저장한다.

[0094] 여기서 기준 정점이 u_i인 CT를 CT_{ui}로 나타낸다. 본 외부 서브그래프 리스팅 방법에서는 CT_{ui}의 각 정점에 대해서 이 정점과 매칭될 수 있는 후보 정점 집합과 이 정점 집합이 속한 페이지들의 집합을 이용하여 다음에 디스크로부터 읽을 페이지의 집합을 결정한다.

[0095] 본 발명에서 유도 정점 u_i의 매칭 후보 집합은 후보 페이지 집합 P_i과 후보 정점 집합 V_i의 쌍 (P_i, V_i)으로 정의한다. 이하, 도 7을 참조하여, 본 실시예에 따른 후보 트리를 설명한다.

[0096] 도 4 (a)는 질의 그래프 G_Q에서 u_1, u_2, u_3으로 이루어진 유도 그래프를 찾았을 때, u_1를 기준 정점으로 선정해서 생성한 CT를 나타낸다.

[0097] 도 4 (b)의 간선 중 실선으로 표현된 간선은 트리 간선(tree edge)이고 깊이 우선 탐색 신장 트리 모양을 가지며, 점선으로 표현된 간선은 비트리 간선(non-tree edge)이다.

[0098] 도 4에서 CT의 각 정점의 오른쪽에 존재하는 세 행의 박스는 매칭 후보 집합을 표현한 것이다.

[0099] 본 발명의 CT에서 정점 u_i의 매핑 후보 집합은 트리 상에서 u_i의 부모 정점인 $parent(u_i)$에 의해 결정된다.

[0100] 도 4의 예시에 따르면 u_2의 매핑 후보 집합인 (P_2, V_2)에서 후보 정점 집합 V_2는 u_i의 매칭 후보 집합 (P_1, V_1)에 속하는 정점 $v \in V_1$의 인접 리스트 n(v)를 스캔하여 구할 수 있다. 후보 페이지 집합 P_2는 후보 정점 집합 V_2에 속한 데이터 정점들이 각각 속한 디스크 페이지 번호들을 얻어옴으로써 구할 수 있다.

[0101] 다음으로, 본 발명에서 후보 트리를 이용한 외부 서브 그래프 리스팅 과정을 설명한다.

[0102] 이하 알고리즘 2는 외부 서브 그래프 리스팅의 과정을 설명한다.

[0103] 알고리즘 2 (ExtSubgraphListing)

```
Input: RBI query graph G_RBI, Red query graph G_red, current chunk range C_cur,
       candidate tree CT, current depth d_E
1: u_dE ← GetCurrentMappingQueryVertex(d_E);
2: if (d_E = |G_red|) then
3:     foreach (pid in CT[u_dE]) do
4:         AsyncRead(pid, ExtVertexMapping(pid, G_RBI, G_red, CT, u_dE));
5: else
6:     while ( there are more candidates in CT[u_dE] to process) do
7:         CT[u_dE]' ← GetCurrentMappingCandidate(CT[u_dE], m_ext);
8:         foreach (pid in CT[u_dE]') do
9:             AsyncRead(pid, UpdateCandidates(pid, CT, u_dE));
10:        wait until UpdateCandidates(·);
11:        ExtSubgraphListing(G_RBI, G_red, C_cur, CT, d_E + 1);
12:    end
```

[0104]

[0105] 알고리즘 2는 외부 서브 그래프 리스팅이 재귀적으로 수행되는 과정을 나타낸다. 알고리즘 2에서 $CT[u_i]$는 i번째로 매칭할 유도 정점의 매칭 후보 집합이다.

[0106] CT의 각 정점의 매핑 후보 집합이 갱신되는 과정은 페이지 단위의 깊이 우선 탐색을 통하여 이루어지게 된다.

[0107] 알고리즘 2의 과정을 설명하자면 아래와 같다. 먼저 1번째 줄에서 유도 정점 중에서 d_E번째로 매핑할 유도 정점 u_{dE}를 가져온다. 2번째 줄에서 u_{dE}가 메모리 구역에 아직 디스크 페이지를 로드하지 않은 마지막 유도 정점인지를 체크한 뒤, 만약 맞다면 후보 트리가 저장하고 있는 u_{dE}의 후보 페이지 집합으로부터 하나씩 페이지 번호

pid를 들고 온다. 페이지 번호 pid를 사용해서 디스크에 저장된 페이지 pid에 대해 비동기 I/O를 요청하는데, 비동기 I/O가 완료된 순간 콜백 함수 ExtVertexMapping이 실행된다. ExtVertexMapping은 지금까지 메모리에 로드된 디스크 페이지들로부터 서브그래프를 찾는 작업을 수행한다.

[0108] 만약 u_{d_E}가 매핑되지 않은 마지막 유도정점이 아니라면 줄 6에서 줄 12에 걸친 반복문을 수행한다. 줄 6에서 줄 12에 걸친 반복문은 u_{d_E}를 매핑하는데 사용할 디스크 페이지의 집합과 데이터 정점의 집합을 후보 페이지 집합과 후보 정점 집합으로부터 고르는 과정(줄 7), 고른 디스크 페이지 집합을 메모리에 로드하는 과정(줄 8~9), 메모리에 로드된 디스크 페이지로부터 인접 리스트 정보를 읽어 후보 트리를 업데이트 하는 과정(줄 9의 UpdateCandidates 함수), $d_E + 1$ 단계의 외부 서브 그래프 리스팅을 재귀적으로 호출하는 과정(줄 11), 그리고 디스크 페이지의 집합을 메모리로부터 언로드 하는 과정(위 알고리즘에서 생략)으로 이루어진다.

[0109] 줄 8에서 $CT[u_i]$의 후보 페이지 집합에 속한 디스크 페이지 중 일부를 메모리에 로드했을 때, 함수 UpdateCandidates가 호출되어 디스크 페이지 안에 있는 인접 리스트 정보를 사용해 $u_j \in child(u_i)$에 대해서 $CT[u_j]$를 갱신한다. 함수 UpdateCandidates가 어떻게 u_j의 매핑 후보 집합을 갱신하는지는 아래의 알고리즘 3에서 설명한다.

[0110] CT의 모든 정점의 매핑 후보 집합이 정해지면, 위 알고리즘 2에서 설명한 바와 같이, 매핑 후보 집합의 갱신은 깊이 우선 탐색의 방법으로 진행된다.

[0111] 후보 페이지 집합 P_i에 속한 디스크 페이지 개수가 메모리 구역 하나의 크기를 초과하는 경우 P_i의 모든 페이지를 해당 메모리 구역에 동시에 로드할 수 없게 된다.

[0112] 이와 같이 모든 페이지가 해당 메모리 구역에 동시에 로드될 수 없는 경우, 해당 메모리 구역 크기에 로드될 수 있는 $(P_i', V_i') \subset (P_i, V_i)$을 먼저 로드한다.(줄 7)

[0113] 이때, 해당 메모리에 로드된 $CT[u_i]$의 부분 집합을 본 발명에서는 $CT[u_i]_{curr}$로 표현한다. 그 후에 콜백 함수 UpdateCandidates에서 정점 $v \in V_i'$의 인접 리스트 n(v)를 스캔하여 $u_j \in child(u_i)$에 대해서 (P_j, V_j)를 구한다.

[0114] 본 발명에서는 유도 그래프 G_{red}의 모든 정점에서 메모리에 로드될 부분 집합 $(P', V') \subset (P, V)$이 정해지고 해당 페이지들이 메모리에 로드된 다음에 콜백 함수 ExtVertexMapping를 호출하여, CT_{curr}가 저장하고 있는 후보 데이터 집합 안의 데이터 그래프 정점과 유도 그래프 정점을 매핑하여 유도 그래프 정점 만으로 이루어진 서브 그래프를 탐색하고, 최종적으로 G_{RBI} 전체를 매핑하여 서브 그래프를 탐색하게 된다.

[0115] 해당 부분 집합에 대하여, 매핑을 완료하면 알고리즘 2의 줄 6에서 다음 반복문을 실행하면서 줄 7에서 다른 부분 집합 $(P'', V'') \subset (P, V), (P'', V'') \cap (P', V') = \emptyset$ 에 대해 상기의 과정을 반복하면서, 전체 집합을 한 번 스캔하며 해당 디스크 페이지에 대하여 모든 매핑을 수행한다. 이 반복문이 ExtSubgraphListing의 매 단계에서 일어날 수 있기 때문에 본 발명이 제시하는 알고리즘은 중첩 반복문의 구조를 가진다.

[0116] 도 4의 예를 들면, (P_2', V_2')에 의해 구한 (P_3, V_3)의 매핑 후보를 모두 스캔한 다음에, (P_3, V_3)을 완전히 초기화하고 $(P_2'', V_2'') \subset (P, V)$에 대한 (P_3, V_3)을 다시 채우는 과정을 수행한다. 그리고 다시 (P_3, V_3)을 메모리에 차례로 로드하며 질의 그래프에 대하여 서브 그래프 매핑을 수행하는 것이다.

[0117] 본 발명에서 메모리의 내부 구역에 로드된 디스크 페이지가 $pid_s \sim pid_e$ 범위일 때, 외부 서브 그래프 리스팅에서 $1 \sim (pid_s - 1)$의 디스크 페이지는 접근하지 않는다.

[0118] 즉, 본 알고리즘에서 찾는 외부 서브 그래프는 서브 그래프를 구성하는 데이터 정점 중 아이디 id(v)가 최소인 정점 v를 골랐을 때 $v \in MEM_{int}$의 조건을 반드시 만족해야 한다.

[0119] 다음으로, 본 발명의 서브 그래프 리스팅 방법에 사용되는 추가 스레드가 수행하는 콜백 함수를 설명한다. 콜백

함수란 디스크에 요청한 한 개의 읽기 작업이 끝났을 때 자동으로 실행하는 함수이다. 본 발명에서 추가 스레드가 수행하는 콜백 함수는 다음과 같이 두 가지로 구별될 수 있다.

[0120] 첫째는 해당 디스크 페이지 안에 저장된 연결 리스트를 스캔하여 다음 메모리 구역에 로드할 매칭 후보 집합을 형성하는 제 1 콜백 함수이고, 둘째는 V_{red} 정점의 후보 정점 집합과 V_{red}의 정점을 매핑하고, 최종적으로 전체 질의 그래프를 데이터 그래프와 매핑하는 제 2 콜백 함수이다.

[0121] 본 발명에서, 제 1 콜백 함수를 UpdateCandidates라고도 하며, 이하, 알고리즘 3를 참조하여, 제 1 콜백 함수를 설명한다.

[0122] 알고리즘 3 (UpdateCandidates)

```
Input: Page id pid, candidate tree data structure CT, current node t_curr
foreach ((v, N(v)) ∈ pid) do
    foreach (u ∈ N(v)) do
        if (t_curr = -1) then
            PC ← PC ∪ ( (pid of u), u) if u ∉ MEM_int;  /* update pivot candidate PC */
        if (t_curr = -2) then
            NEXTPC ← NEXTPC ∪ ( (pid of u), u);
        else
            foreach (u_i ∈ child(t_curr)) do
                CT[u_i] ← CT[u_i] ∪ ( (pid of u), u);
            end
    end
end
```

[0123]

[0124] 알고리즘 3에서 볼 수 있는 바와 같이, $CT[u_i]$로부터 pid=k인 디스크 페이지가 메모리에 로드되었을 때 제 1 콜백 함수는 u_i의 모든 자식 정점 $u_j \in child(u_i)$에 대해 외부 후보 집합 $CT[u_j]$를 갱신한다.

[0125] 그리고, 디스크 페이지 p에 속한 정점 중 $v \in CT[u_i]$를 만족하는 정점의 인접 리스트 n(v)를 스캔하고 저장한다. 제 1 콜백 함수가 피벗 정점에 대한 매핑 후보 집합을 구하는 경우, 즉, 내부 구역으로부터 호출된 경우, 임시 위치에 매핑 후보 집합을 저장한다.

[0126] 이때 정점 $u \in n(v)$이 $u \notin MEM_{int}$를 만족하는 경우에만 후보 집합이 된다. 그리고, 제 1 콜백 함수가 피벗 후보 집합으로부터 호출된 경우, 즉, 피벗 구역으로부터 호출된 경우에도 임시 위치에 매핑 후보 집합을 저장한다.

[0127] 알고리즘 3에서는 설명의 편의상, 세부적인 가지치기 (pruning) 과정을 생략하였다. 실제로는 피벗 정점(pivot vertex)의 개념을 이용한 가지치기, 정점의 차수(degree)를 이용한 가지치기, CT의 비트리 간선에 의한 가지치기가 수행될 수 있다.

[0128] 교집합 연산을 통해 CT의 비트리 간선 정보를 반영할 수 있다.

[0129] 정점 $v \in V_{red}$를 골랐을 때 후보 트리 CT 의 신장 트리에서의 방문 순서가 v보다 빠른 정점 u중 $(v, u) \in G_{red}$인 정점 u의 집합을 조상 집합 U라 하면, U에 속한 정점 u는 후보 트리 CT에서 v의 부모 정점이거나 혹은 CT에서 u에서 v로의 비트리 간선으로 표현될 수 있다.

[0130] 한편, 정점 v의 매핑 후보 집합 $CT[v]$를 v의 조상 집합 U에 속한 정점 u'의 인접 리스트 n(u')의 교집합으로 나타낼 수 있고, 이를 수식으로 나타내면 $CT[u] = \cap_{u' \in U^{n(M[u'])}}$ 이다.

[0131] 여기서, $U_i \subset U_j$ 이므로 임의의 $u \in U_i$에 대해 $u \in U_j$도 항상 성립하고, CT[u]를 재정의한 식에 의해 $CT[u_j] = CT[u_i] \cap \cap_{v \in (U_j \cap U_{ic})} n(M[u'])$가 된다. 따라서, $CT[u_j] \subset CT[u_i]$가 성립한다.

[0132] 따라서, CT를 갱신할 때 트리 간선을 따라 갱신할 뿐만 아니라 상기와 같은 간단한 교집합 연산을 통해 비트리 간선을 고려할 수 있다. 이러한 최적화를 통해 디스크 페이지 접근 수를 더욱 줄일 수 있다.

[0133] 또한, 본 발명에서 제 2 콜백 함수를 ExtVertexMapping 이라고도 하며, 이하, 알고리즘 4를 참조하여, 제 2 콜백 함수를 설명한다.

[0134] 알고리즘 4 (ExtVertexMapping)

Input: Page id *pid*, RBI query graph G_{RBI}, Red query graph G_{red}, candidate tree data structure CT, last node t_{last}

```
Pin page pid;
foreach (data vertex v in page pid) do
    S_M ← BuildInmemMatchingOrder(CT, t_last);
    M[t_last] ← v;
    RecExtVertexMapping(G_RBI, G_red, S_M, CT, 2);
    M[t_last] ←INVALID;
end
Unpin page pid;
```

[0135]

[0136] 알고리즘 4에서 설명하는 제 2 콜백 함수 ExtVertexMapping은 외부 서브그래프 중 ExternalSubraphListing 알고리즘에서 구한 매핑 후보 집합을 만족하고, 디스크 페이지 pid에 유도 정점 t_{last}가 매핑되는 항목들을 모두 찾는 행동을 수행한다.

[0137] 알고리즘 4에서 볼 수 있는 바와 같이, 페이지 아이디가 pid인 디스크 페이지가 메모리에 로드되었을 때 제 2 콜백 함수는 우선 해당 페이지에 속한 데이터 정점 중 $CT[t_{last}]$의 후보 정점 집합에 속한 정점을 유도 정점 t_{last} 와 매핑한다.

[0138] 본 발명에서는 이와 같이, t_{last}를 가장 먼저 매칭하도록 매칭 순서를 정한 다음 남아 있는 질의 유도 정점들에 대해 재귀적으로 데이터 정점을 매칭한다. 이 때 각각의 질의 유도 정점이 자신이 매핑된 디스크 페이지 안에 있는 데이터 정점에 매핑되도록 한다. 즉, 함수 ExternalSubgraphListing에서 i번째 유도 정점의 매핑 후보 집합 $CT[u_i] = (P_i, V_i)$ 중 $(P_i', V_i') \subset (P_i, V_i)$를 메모리 구역에 로드했다면, 제 2 콜백 함수에서는 u_i가 $v \in V_i$ 에 매핑되는 외부 서브그래프만 찾는다. 이와 같이 데이터 그래프의 정점에 대하여, V_{red}의 정점의 매핑을 완료한 뒤, V_{black}, V_{ivory} 정점을 매핑한다.

[0139] 알고리즘 4의 제 2 콜백 함수는 외부 서브 그래프를 탐색하는 함수이고, 제 2 콜백 함수의 CPU 연산은 디스크 I/O와 바람직하게는 오버랩될 수 있다.

[0140] 본 발명에서는, 이처럼 디스크 I/O를 재귀적으로 요청하고, 콜백 함수의 연산과 디스크 I/O를 연속적으로 중첩하여 오버래핑 할 수 있게 된다.

[0141] 다음으로, 본 발명에서 외부 서브 그래프 리스팅 이전에 후보 트리 CT를 초기화하는 작업과 외부 서브 그래프 리스팅을 시작하는 작업을 알고리즘 5을 참조하여 보다 상세히 설명한다.

[0142] 알고리즘 4에서 설명한 외부 서브 그래프 리스팅은 후보 트리 CT가 주어졌을 때 매칭 후보 집합을 채우면서 디스크 페이지를 메모리에 로드하는 과정, 그리고 메모리에 로드된 페이지로부터 외부 서브그래프를 찾는 과정을 설명하였다.

[0143] 따라서 알고리즘 4에 필요한 후보 트리를 초기화하는 과정이 별도로 필요하며 알고리즘 5에서 설명하는 함수 DelegateExtSubgraphListing이 이를 수행한다.

[0144] 알고리즘 5 (DelegateExtSubgraphListing)

```
Input: RBI query graph G_RBI, Red query graph G_red, current chunk range C_cur,
       candidate tree CT, pivot candidate PC
1: while ( there are more candidates in PC to process) do
2:   PC' ← GetCurrentMappingCandidate(PC, m_pivot);
3:   foreach (pid in PC') do
4:     AsyncRead(pid, UpdateCandidates(pid, CT, −2));
5:   wait until UpdateCandidates(·);
6:   foreach (u_i ∈ G_red) do
7:     initialize the bitmap sets in CandidateTree
       BuildCandidateTreeStructure(CT, G_red, u_i);   /* Set CT structure with
       BFS, which root node is u_i */
8:     d_E ← 1;
9:     CT[u_i] ← PC;
10:    CT[u_i]_curr ← PC';
11:    foreach ( u_j ∈ child(u_i)) do
12:      CT[u_j] ← NEXTPC;
13:    end
14:    ExtSubgraphListing(G_RBI, G_red, C_cur, CT, d_E + 1);
15:  end
16: end
```

[0145]

[0146] 본 발명에서 함수 DelegateExtSubgraphListing 는 RBI 질의 그래프 G_{RBI}, 유도 그래프 G_{red}, 현재 메모리 내부 구역에 로드된 페이지 집합 C_{cur}, 트리 구조 및 매핑 후보 집합이 아직 생성되어 있지 않은 후보 트리 CT, 그리고 피벗 유도 정점을 매핑하는데 사용할 후보 정점 집합 PC 를 인자로 받는다.

[0147] 함수 DelegateExtSubgraphListing은 내부적으로 후보 정점 집합 PC 중 일부를 디스크로부터 읽어 메모리에 로드하는 것을 반복하는 꼴을 가진다(줄 1 ~ 16). 먼저, 후보 정점 집합 PC 중 피벗 메모리 구역에 로드할 정점 집합 PC'을 고른다.(줄 2) 이 때 피벗 메모리 구역의 크기는 $|m_{pivot}|$으로 주어져 있다고 가정한다. 그 다음, PC'에 속한 데이터 정점을 포함하는 페이지들 $pid \in PC'$들에 대해 디스크 읽기를 요청한다. (줄 3 ~ 4) 이 때 함수 UpdateCandidates를 디스크 읽기를 끝낸 다음 수행할 콜백 함수로 지정한다. 그리고 모든 요청한 디스크 읽기 작업이 완료되고 콜백 함수 UpdateCandidates가 수행이 끝날 때까지 기다린다 (줄 5). 그 다음, 유도 그래프 G_{red}에 속한 유도 정점 u_i을 하나 고른 뒤, 후보 트리를 생성하고 외부 서브 그래프 리스팅을 수행한다.(줄 6 ~ 15)

[0148] 알고리즘 5에서 볼 수 있는 바와 같이, 가장 먼저 피벗 후보 집합의 일부를 피벗 구역에 로드하고, 앞서 설명한 매칭 이원화 방식에 따라 고정된 내부 구역과 피벗 구역에 대해 질의 그래프의 정점의 매칭 순서를 바꾸어 가며 질의 그래프에 대한 매칭을 반복적으로 수행하여 서브 그래프를 탐색한다.

[0149] 후보 트리 CT의 구조를 DFS(깊이 우선 탐색) 신장 트리로 구한 뒤 후보 트리 CT의 기준 정점 u_i의 자식 정점 $child(u_i)$중 하나가 반드시 내부 구역에 포함되도록 제한하여, 주어진 후보 트리를 이용한 외부 서브 그래프 리스팅의 탐색이 하나 이상의 유도 정점은 항상 메모리의 외부 구역 안의 데이터 정점에 매핑되고 다른 하나 이상의 유도 정점은 항상 메모리의 내부 구역 안의 데이터 정점에 매핑되도록 강제할 수 있다.

[0150] 그리고, 메인 스레드는 CT를 새로운 매칭 순서에 대해 초기화하고 CT를 재귀적으로 갱신하는 추가 스레드(제 1 콜백 함수를 수행)를 호출한다.

[0151] 다음으로, 알고리즘 6을 참조하여, 본 실시예의 내부 서브 그래프 리스팅에 대하여 R 정점을 매핑할 때 이용하는 RedIntVertexMapping 함수와 내부 서브 그래프 리스팅과 외부 서브 그래프 리스팅에서 B 정점, I 정점을 매핑할 때 이용하는 함수인 NonRedVertexMatching 함수를 설명한다.

[0152] 알고리즘 6 (IntSubgraphListing)

```
   Input: current chunk range C_cur
   /* Step 1. Generate matching order of G_red                    */
1: qo ←GenISLMatchOrder(G_red);
   /* Step 2. Start matching                                      */
2: foreach (v in all data vertices in C_cur) do
3:   M[qo[1]] ← v;
4:   if (deg(v) < deg(qo[1])) then continue;
5:   RecIntSubgraphListing(C_cur, qo, 2);
6:   M[qo[1]] ←INVALID;
7: end
8: Function RecIntSubgraphListing
     Input: current chunk range C_cur, query matching order qo, depth d_INT
9:   if (d = |G_red|) then
10:    NonRedVertexMatching(qo);
11:    return;
12:  end
13:  u_cur ← qo[d_INT];
14:  U_CON ← n(u_cur) ∩ qo[1 : d_INT − 1];
15:  foreach (v in ⋂_{u∈U_CON}(N(M[u]))) do
16:    if (v is visited) then continue;
17:    if (deg(v) < deg(u_cur)) then continue;
18:    if (v.pid ∉ P(C_cur)) then continue;
19:    if (M[u_cur] violates partial order PO) then continue;
20:    M[u_cur] ← v;
21:    RecIntSubgraphListing(C_cur, qo, d_INT + 1);
22:    M[u_cur] ← INVALID;
23:  end
24: end
```

[0153]

[0154] 알고리즘 6에서 볼 수 있는 바와 같이, 우선, GenISLMatchOrder 함수를 통해 질의 그래프 정점의 매칭 순서를 정한다. R 정점의 인접 리스트를 이용해 B 정점, I 정점과 데이터 그래프의 정점을 매핑하므로 R 정점의 매칭 순서가 B 정점, I 정점보다 반드시 앞서야 하며, R 정점에서는 그래프 탐색 순서를 따른다.

[0155] 다음으로, RecIntSubgraphListing 함수에서는 정해진 매칭 순서에 따라 V_{red}의 매핑을 재귀적으로 형성한다. 모든 R 정점의 매핑이 완료되면 NonRedVertexMatching 함수를 통해 V_{red} 와 매핑된 정점의 인접 리스트들을 이용하여 V_{black}, V_{ivory} 를 매핑한다.

[0156] 이하에서는 알고리즘 7을 참조하여, 외부 서브 그래프 리스팅 과정에서 CT가 모두 채워지고 마지막 디스크 접근이 알고리즘 5의 콜백 함수를 호출한 뒤에 콜백 함수로부터 불려지는 RecExtVertexMapping 함수를 설명한다.

[0157] 알고리즘 7 (RecExtVertexMapping)

```
1: Function RecExtVertexMapping
     Input: matching order S_M, candidate tree data structure CT, depth d
2:   t_cur ← S_M[d];
3:   U_CON ← n(t_cur) ∩ S_M[1 : d − 1];
4:   foreach (v in ⋂_{t∈U_CON}(N(M[t]))) do
5:     if (IsJoinable(t_cur, v, CT)) then
6:       M[t_cur] ← v;
7:       if (d = |G_red|) then NonRedVertexMatching(M);
8:       if (d! = |G_red|) then RecExtVertexMapping(S_M, CT, d + 1);
9:       M[t_cur] ← INVALID;
10:    end
11:  end
12: end
13: Function IsJoinable
     Input: Current mapping node t_cur, data vertex v, candidate tree data structure CT
14:  if (v is already matched) then return false;
15:  if (Deg(v) < Deg(t_cur)) then return false;                  /* P1. degree */
16:  if (v ∉ CT[t_cur]) then return false;          /* P2. candidate vertex set */
17:  if ((∃PO(t' < t_cur) s.t. v < M[t']) or (∃PO(t' > t_cur) s.t. v > M[t'])) then
18:    return false;                                          /* P3. partial order */
19:  if (t_cur ≠ t_pivot and v, t_cur makes violation with pivot vertex) then
20:    return false;                                          /* P4. Pivot Vertex */
21:  return true;
22: end
```

[0158]

[0159] 알고리즘 7에서 볼 수 있는 바와 같이, RecExtVertexMapping 함수는 주어진 매칭 순서와 CT의 현재 상태에 기반

하여 외부 서브 그래프의 매핑을 수행한다.

[0160] RecExtVertexMapping의 B 정점, I 정점의 매칭은 내부 서브 그래프 리스팅과 마찬가지고 NonRedVertexMatching 함수를 이용한다.

[0161] IsJoinable 함수는 CT의 정보를 기반으로 R 정점을 매핑할 때 불가능한 해를 피하기 위한 조건문과 세부적인 몇 가지 가지치기를 수행한다.

[0162] 불가능한 해를 피하기 위해서는 현재 매핑 M에 추가적으로 매칭할 데이터 그래프의 정점 v가 이미 매칭 여부, CT에 포함 여부, 부분 순서를 어기는 경우 및 피벗 정점의 개념과 부합 여부에 대해 검사한다.

[0163] 여기서 가지치기는 정점의 차수를 비교한 가지치기가 있다.

[0164] 이하, 알고리즘 8을 참조하여, I 정점 및 B 정점을 매칭하는 함수를 설명한다.

[0165] 알고리즘 8 (NonRedVertexMatching)

```
Input: a red graph matching sequence qseq
C ← ∅;
for (i ← 1 to |V(Q_R)|) do
    /* from topology-data node mapping to query-data node mapping */
    M'[qseq[i]] ← M[t_i];
end
RecNonRedVertexMatching(1, M');
Function RecNonRedVertexMatching
    Input: depth i, mapping M'
    if (i >the number of non-red vertices) then
        /* full mapping */
        report M';
        return;
    u_i ← i-th non-red query vertex;
    if (u_i is a ivory query vertex) then
        /* an ivory query vertex has more than two neighborhoods */
        N ← N(u_i);
        foreach (data vertex v in ∩_{u∈N}(N(M'[u]))) do
            if (v is already matched) then continue;
            if (M'[u_i] = v violates partial order PO) then continue;
            NonRedVertexMatching(i + 1, M');
        end
    else if (u_i is a black query vertex) then
        /* a black query vertex has only one red query vertex as a
           neighborhood */
        n ← the first element in N(u_i);
        foreach (data vertex v in N(M'[n])) do
            if (v is already matched) then continue;
            if (M'[u_i] = v violates partial order PO) then continue;
            NonRedVertexMatching(i + 1, M');
        end
    end
end
```

[0166]

[0167] 서브 그래프 리스팅은 정해진 매칭 순서에 따라 정점을 매칭하는데 I 정점의 경우 2개 이상의 인접 리스트의 교집합 연산을 동반한다.

[0168] B 정점의 경우는 인접 리스트를 한번 스캔함으로써 구할 수 있다.

[0169] 이때, 함수 RecNonRedVertexMatching은 불가능한 해를 피하기 위한 세부적인 조건을 포함할 수 있다.

[0170] 이하, 본 발명의 서브 그래프 리스팅 방법의 정확성을 증명한다.

[0171] 즉, 서브 그래프 리스팅 방법에 사용되는 알고리즘은 서브 그래프 리스트가 중복해서 나타나지 않는 것과, 모든 서브 그래프 리스트를 탐색하였는지를 증명하고자 한다.

[0172] 먼저, 본 발명의 서브 그래프 리스팅 방법의 알고리즘은 동일한 서브 그래프를 중복하여 나열하지 않는 것을 증명한다.

[0173] 내부 구역의 i번째 작업 순서에서 메모리 버퍼 내부 구역에 로드된 데이터 정점 id의 범위를 $[v_{int}^{i}, u_{int}^{j}]$와 같이 나타낸다고 할 때, $i \neq j$ 인 모든 i, j에 대해 $[v_{int}^{i}, w_{int}^{i}] \cap [v_{int}^{j}, w_{int}^{j}] = \emptyset$ 이다. 즉, 각 작업의 내부 구역은 서로 겹치지 않는다.

[0174] 그리고, 내부 서브 그래프는 내부 구역에 속한 정점만을 포함하므로 본 알고리즘이 찾는 내부 서브 그래프 사이

에는 중복 해가 발생하지 않는다.

[0175] 외부 서브 그래프는 내부 구역의 정점 하나와 내부 구역 외부의 정점 하나를 반드시 포함해야 한다. 따라서, 내부 서브 그래프와 외부 서브 그래프는 상호배타적이 된다.

[0176] 예를 들어, 피벗 구역의 k번째 작업에서 메모리 버퍼 피벗 구역에 로드된 데이터 정점 id의 범위를 $[v^k_{pivot}, w^k_{pivot}]$로 표현하면, 외부 서브 그래프는 내부 구역 i번째, 피벗 구역 k 번째 작업일 때, $v_i \in [v^i_{int}, w^i_{int}]$, $v_k \in [v^k_{pivot}, w^i_{pivot}]$ 에 속하는 두 정점 v^i, v^k 와 간선 (v^i, v^k)을 반드시 포함하나 내부 서브 그래프는 이러한 간선이 없어야 한다.

[0177] 각 작업의 피벗 구역도 내부 구역과 마찬가지로 서로 겹치지 않는다.

[0178] 피벗 구역의 데이터 정점 v^k 을 피벗 정점으로 갖는 외부 서브 그래프에서 v^k는 피벗 정점의 성질을 가진다.

[0179] 즉, v^k는 내부 구역의 정점과 간선이 외부 서브 그래프 내에 존재하는 정점 중에서 정점 번호 $id(v^k)$가 가장 작아야 한다는 것을 만족해야 하기 때문에, 간선 (v^i, v^k)을 포함한 외부 서브 그래프는 정확히 한 번의 작업에서만 나타난다.

[0180] 따라서, 본 발명에서 서로 다른 작업에서 찾은 두 외부 서브 그래프는 반드시 다른 서브 그래프가 된다.

[0181] 따라서 본 발명에서 찾은 서브 그래프는 중복해서 발생하지 않는다.

[0182] 다음으로, 본 발명의 서브 그래프 리스팅 방법의 알고리즘은 질의 그래프에 대한 모든 서브 그래프를 빠짐없이 찾는 것을 증명한다.

[0183] 즉, 본 발명에서, 모든 서브 그래프는 내부 서브 그래프 리스팅 혹은 외부 서브 그래프 리스팅 과정을 통해 구할 수 있다.

[0184] 이때, 내부 서브 그래프도 아니고, 외부 서브 그래프도 아니면서, 질의 그래프와 매핑 M이 이루어진 서브 그래프가 존재한다고 가정해 본다.

[0185] 그 경우, 정점 $v_{min} = \min(M[u_i]) \forall u_i \in G_q$ 에서 v_{min} 을 포함하는 내부 구역의 작업이 반드시 존재하게 된다.

[0186] v_{min}을 포함하는 내부 구역의 작업이 i번째 작업이라고 하면, v_{min}과 매핑되는 질의 그래프의 정점 u_{min}과 인접한 정점이 매핑된 데이터 정점 중 i번째 내부 구역에 포함되지 않고, 정점 번호가 가장 작은 데이터 정점 $v_p = \min(M[u]) \forall u \in n(u_{min})$, $M[u] \notin MEM_{int}$가 반드시 존재하게 된다.

[0187] 이때, 데이터 그래프 상에 간선 (v_{min}, v_p)가 존재하며, v_p는 메모리 버퍼 내부 구역 밖의 정점이므로 이를 포함하는 페이지가 피벗 구역에 로드되는 작업이 반드시 존재하게 된다.

[0188] v_p를 포함하는 피벗 구역의 작업이 k번째 작업이라고 하면, i번째 작업의 내부 구역과 k번째 작업의 피벗 구역이 고정된 상태일 때, 서브 그래프 리스팅에서 외부 서브 그래프 리스팅 방법의 방법에 따르면, (v_{min}, v_p)를 질의 그래프의 모든 간선과 매핑하는 과정이 반드시 존재하게 된다.

[0189] 따라서, 간선 (v_{min}, v_p)를 포함하는 서브 그래프는 외부 서브 그래프에 반드시 포함되어야 한다.

[0190] 즉, 본 발명에서는 내부 서브 그래프도 아니고 외부 서브 그래프도 아닌 서브 그래프는 존재할 수 없으며, 모든 서브 그래프의 해는 내부 서브 그래프 리스팅 혹은 외부 서브 그래프 리스팅 과정을 통해 찾을 수 있다.

[0191] 이하, 도 5 내지 도 7을 참조하여, 본 발명의 서브 그래프 리스팅 방법의 효과를 보여주는 실험(이하, 간단히 '본 실험'이라 지칭하기도 한다)에 사용한 실험 환경 및 실험 데이터셋에 대해 설명한다.

[0192] 도 5는 본 발명의 실시예에 따른 서브 그래프 리스팅 방법을 실험한 질의 그래프 및 질의 그래프의 부분 순서 집합을 설명한다.

[0193] 본 실험에서는, 도 5 (a) 에 도시된 바와 같은, QG1에서 QG9까지의 9가지 질의 그래프를 사용하였다.

[0194] 도 5 (b)는 각각의 질의 그래프에 매칭되는 데이터 서브 그래프의 자가 동형을 방지하기 위해 사용된 해당 질의 그래프의 부분 순서를 보여준다.

[0195] 본 실험에서는 다양한 데이터셋을 사용하였으며, 데이터셋들은 크게 실제 데이터셋과 합성 데이터셋의 두 가지 종류의 그래프 데이터셋으로 구분될 수 있다.

[0196] 먼저, 실제 데이터셋은 각종 웹 그래프, 소셜 네트워크 그래프를 포함하고 있는 총 7개의 데이터셋을 포함하고 있다.

[0197] 합성 데이터셋은 R-MAT 모델을 사용하여 정점의 수를 변화시키며 생성한 5개의 데이터셋을 포함하고 있다.

[0198] 표 1은 본 실험에서 이용하는 실제 데이터셋들의 정점과 간선 개수를 표로 나타낸다.

[0199] R-MAT 모델을 사용하면 소셜 네트워크 그래프와 특성이 매우 유사한 데이터 그래프를 생성할 수 있다.

[0200] 표 2는 본 실험에서 사용한 합성 데이터셋의 통계 정보를 표로 나타낸다. 본 실험에서는 최소의 버퍼 크기가 주어진다 하더라도 주어진 질의를 수행할 수 있도록 하기 위하여, 필요한 메모리 버퍼 크기는 사용자로부터 주어진 크기에 스레드 수만큼의 프레임을 추가로 할당하여 실험을 수행하였다.

[0201] 표 1

	Wikitalk	cit-Patents	web-Google	LiveJournal	Twitter	Orkut	UK
$\|V\|$	2.4M	3.8M	0.9M	4.8M	42M	3.1M	106M
$\|E\|$	5.0M	18M	5.1M	69M	1,468M	117M	3,739M

[0202]

[0203] 표 2

	RMAT_v4	RMAT_v8	RMAT_v16	RMAT_v32	RMAT_v64
$\|V\|$	4M	8M	16M	32M	64M

[0204]

[0205] 본 실험에서는 여러 환경 상에서의 성능을 측정하여 본 발명의 서브 그래프 리스팅 방법의 신뢰성을 보이기 위하여, 서로 다른 두 가지 환경에서 실험을 진행하였다.

[0206] 첫 번째 실험 환경(S-ENV)은 24G 바이트 메모리, 1테라 바이트 SSD(Samsung 850Pro), Intel Core i7 3930K CPU 3.20GHz CPU(총 6 코어)를 가지고 있다.

[0207] 두 번째 실험 환경(D-ENV)은 32기가 바이트 메모리, 1테라 바이트 SSD(Samsung 830EVO), Intel Xeon CPU E5-245- 2.10GHz 2개(총 16 코어)를 가진 시스템이다.

[0208] S-ENV는 Windows 7과 D-ENV는 각각 MS windows 7과 MS windows 8 운영 체제를 탑재하고 있다.

[0209] 실험 성능을 측정하기 위한 측정 단위로는 수행 시간을 사용하였으며, 오버래핑 성능 정도를 측정하기 위해서는 스피드 업(speed-up)을 사용한다. 스피드 업은 다중 스레드에서의 수행 시간을 단일 스레드에서의 수행 시간으로 나눈 값으로써 단일 스레드에서의 수행 시간과 비례하여 얼마나 수행 성능이 증가하였는지를 나타내는 지표이다.

[0210] 이하, 도 6을 참조하여, 본 발명의 서브 그래프 리스팅 방법의 스레드 수에 따라 나타난 성능 변화를 설명한다.

[0211] 도 6은 본 발명의 한 실시예에 따른 서브 그래프 리스팅 방법의 스피드 업 결과를 나타낸다.

[0212] 도 6 (a)는 본 발명의 서브 그래프 리스팅 방법의 Web-Google, LiveJournal, cit-Patents 및 Orkut 데이터셋에서의 각 질의 별로 스레드 수가 6일 때의 스피드 업을 나타낸다.

[0213] 즉, 도 6 (a)에 도시된 바와 같이, 본 발명의 한 실시예에 따른 서브 그래프 리스팅 방법은 질의 그래프 QG1의 Web-Google 데이터셋을 제외하고 다른 데이터셋에서 모두 4 이상의 스피드 업을 나타내었다.

[0214] 구체적으로, 본 발명의 서브 그래프 리스팅 방법은 스레드가 6일 때 최소 4.42배에서 최대 5.60배의 스피드 업

을 보여 스레드의 개수의 배수만큼 속도가 향상되었음을 보여준다.

[0215] 이는 본 발명에서 제안한 서브 그래프 리스팅의 방법이 I/O 시간을 효과적으로 오버래핑하고, 여러 스레드가 검색 공간을 독립적으로 동시에 검색함을 보여준다.

[0216] 데이터셋 web-Google은 QG3을 제외하고는 다른 데이터셋과 비교하여 스피드 업이 낮은 것을 확인할 수 있는데, 이는 web-Google의 데이터셋이 매우 작기 때문에, 전체 수행시간이 병렬화 될 수 없는 디스크 I/O 시간에 종속되기 때문이다. 이와 같은 결과는 대규모의 그래프에서 보다 효과적으로 제안한 병렬화 방법을 사용할 수 있음을 의미한다.

[0217] 도 6 (b)는 본 발명의 서브 그래프 리스팅 방법의 LiveJournal 데이터셋에서의 스레드 수에 따른 각 질의 별 스피드 업을 나타낸다.

[0218] 도 6 (b)에 도시된 바와 같이, 모든 질의에 대해서 스레드 수가 증가할수록 스피드가 선형으로 증가한다. 이와 같은 결과는 본 발명에서 제안한 서브 그래프 리스팅 방법의 우수함을 나타내며, 스레드 수에 따른 확장성을 가지고 있다는 것을 나타낸다.

[0219] 이하, 도 7 참조하여, 본 발명의 한 실시예에 따른 종래의 분산 처리 방식을 사용한 PSgL과 비교하여 성능이 향상된 것을 나타낸다.

[0220] 도 7은 본 발명의 한 실시예에 따른 서브그래프 리스팅 방법을 실제 데이터셋에 대하여 수행한 시간을 나타낸다.

[0221] 도 7을 참조하면, 본 발명의 서브 그래프 리스팅 방법은 종래의 PSgL보다 수행 시간 측면에서 우수한 결과를 나타낸다.

[0222] 구체적으로, 본 발명의 서브 그래프 리스팅 방법은 수행 시간면에서 최소 3.6배에서 662배로 성능이 향상되었다.

[0223] 이하, 본 발명에 대해 다시 정리하자면, 본 발명의 서브 그래프 리스팅 방법은 RBI 그래프와 중첩 오버래핑을 이용한 서브 그래프 리스팅의 디스크 기반 해결 방안을 제안하였다.

[0224] 본 발명은 페이지 블록 단위의 깊이 우선 탐색을 이용하는 새로운 서브 그래프 리스팅 방법을 사용하였다.

[0225] 또한, 본 발명의 서브 그래프 리스팅 방법은 CT 구조와 비동기적 디스크 I/O를 이용하여 중첩 루프 구조의 전체 과정에서 CPU 연산과 I/O를 계속적으로 오버랩핑하는 것으로서, 종래의 삼각형 리스팅을 사용한 오버래핑을 확장한 방법이다.

[0226] 또한, 본 발명의 서브 그래프 리스팅 방법은 스레드 증가에 따라 선형의 스피드 업이 나타나는 우수한 병렬성을 보이고 있다.

[0227] 본 발명의 효과를 더욱 구체적으로 설명하자면, 대부분의 질의 그래프와 데이터 그래프의 경우에 있어서 종래의 방법들보다 성능이 크게 향상되었고, 종래의 PSgL에서 51대의 머신을 사용하여 측정한 방법과 비교하여, 최대 44배 성능이 향상되었다.

[0228] PSgL을 디스크 기반으로 수정한 방법과 본 발명의 서브 그래프 리스팅 방법을 비교하면 성능을 최대 662배까지 향상시켰다.

[0229] 이와 같은 결과를 기반으로, 본 발명의 서브 그래프 리스팅 방법은 대규모 데이터 그래프에 대한 서브 그래프 리스팅의 성능을 크게 향상시킬 수 있다.

[0230] 본 발명의 서브 그래프 리스팅 방법에 의하면, 질의 그래프를 R B I 그래프로 변환하고 후보 트리를 사용함으로써, 대규모의 데이터 그래프에 대하여 디스크의 I/O 횟수를 대폭 감소 시킬 수 있다.

[0231] 본 발명의 서브 그래프 리스팅 방법은 메모리 버퍼를 내부, 피벗 및 외부 서브 구역으로 구별하고, 내부 서브 그래프 리스팅 및 피벗 구역과 외부 구역의 서브 그래프 리스팅을 멀티 스레드로 오버 래핑하여 독립적으로 수행할 수 있어 수행 속도를 크게 개선할 수 있다.

[0232] 특히, 본 발명의 서브 그래프 리스팅 방법은 메인 스레드에 의한 디스크 I/O 시간 동안 추가 스레드로 외부 서브 그래프 리스팅을 오버래핑으로 동시에 수행할 수 있어 수행 속도를 개선하게 된다.

[0233] 이와 같이, 본 발명의 서브 그래프 리스팅 방법은 멀티 스레드를 지원하여 스레드 증가에 따라 수행 속도가 비

례하여 증가하게 된다.

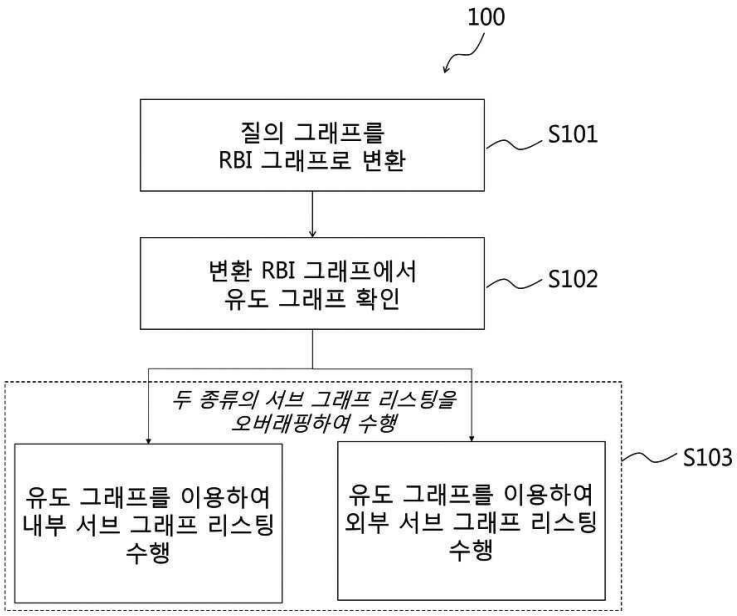
[0234] 이상에서는 본 발명을 실시예에 기초하여 설명하였으나, 본 발명의 사상은 상기 실시예에 제한되지 아니하며, 본 발명의 사상을 이해하는 당업자는 동일한 사상의 범위 내에서, 구성요소의 부가, 변경, 삭제, 추가 등에 의해서 다른 실시 예를 용이하게 제안할 수 있을 것이나, 이 또한 본 발명의 사상범위 내에 든다고 할 것이다.

부호의 설명

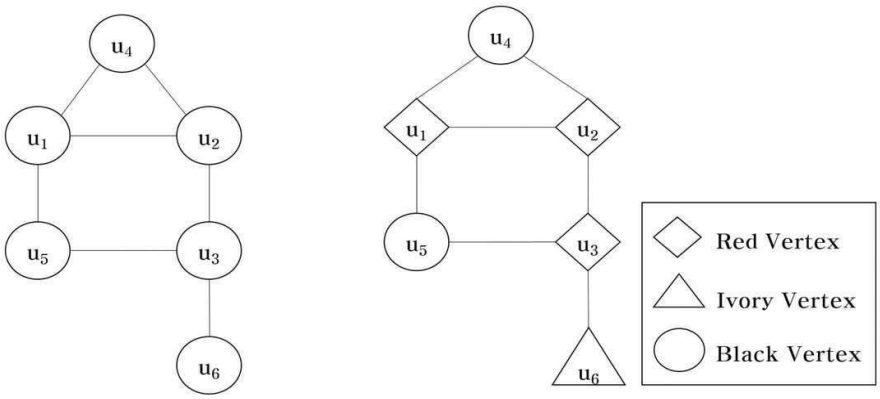
[0235] 100 : 서브 그래프 리스팅 방법

도면

도면1

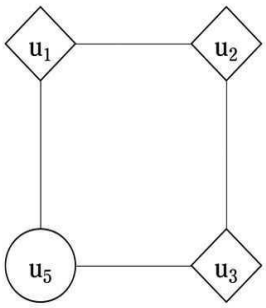


도면2

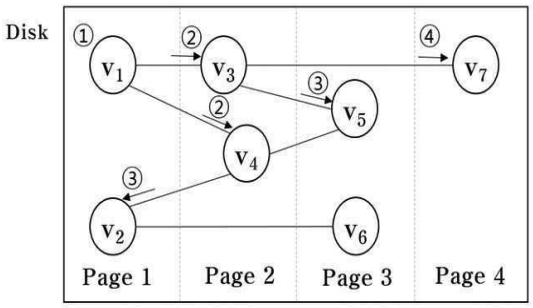


(a) 질의 그래프 G_Q (b) G_Q를 변형한 RBI 그래프 G_{RBI}

도면3

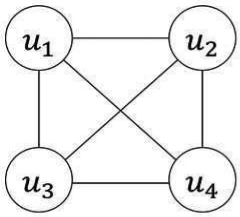


(a)RBI 그래프, V_{red} = $\{u_1, u_2, u_3\}$

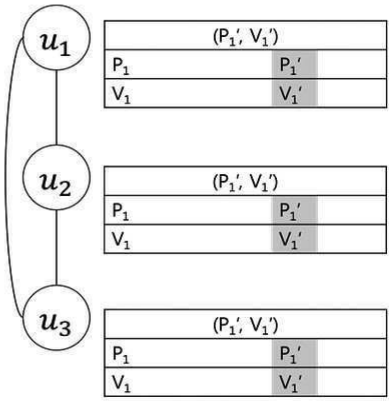


(b) 데이터 그래프 상의 정점 방문 순서의 예

도면4

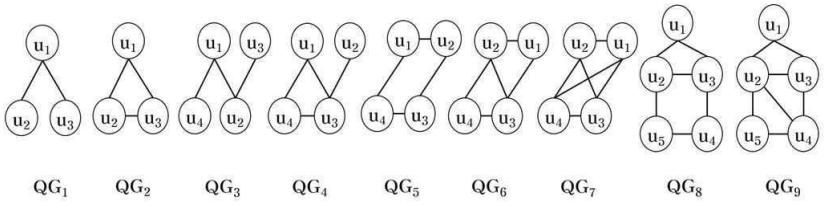


(a) 질의 그래프 G_Q



(b) G_Q의 CT_{ui}

도면5

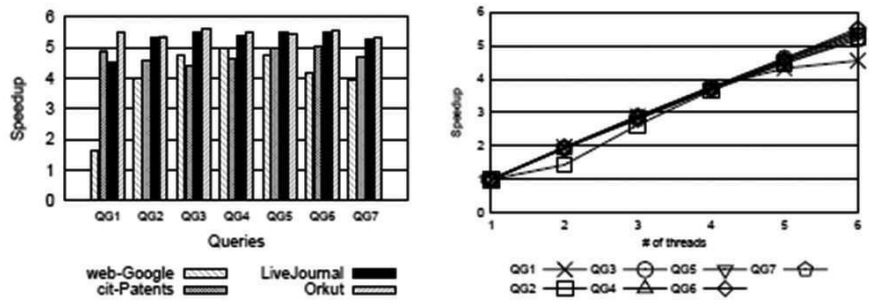


(a) 질의 그래프

QG_1	QG_2	QG_3	QG_4	QG_5	QG_6	QG_7	QG_8	QG_9
$u_2 < u_3$	$u_1 < u_2$ $u_1 < u_3$ $u_2 < u_3$	$u_1 < u_2$	$u_1 < u_2$	$u_1 < u_2$ $u_1 < u_3$ $u_2 < u_3$ $u_2 < u_4$	$u_1 < u_4$ $u_2 < u_3$	$u_1 < u_2$ $u_1 < u_3$ $u_1 < u_4$ $u_2 < u_3$ $u_2 < u_4$ $u_2 < u_4$	$u_2 < u_3$	$u_1 < u_5$

(b) 질의 그래프의 부분 순서 집합

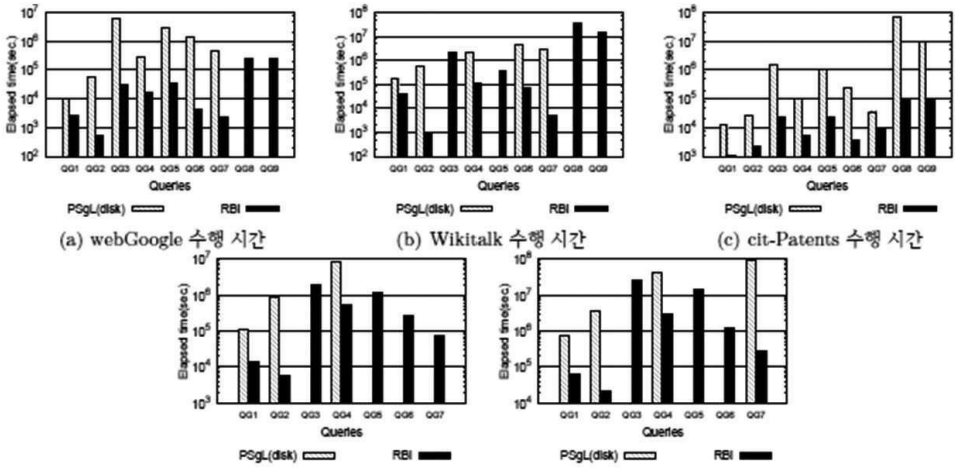
도면6



(a) 각 데이터셋에서의 스피드 업 (# of threads = 6)

(b) LiveJournal 데이터셋에서의 스레드 수에 따른 스피드 업

도면7



(a) webGoogle 수행 시간

(b) Wikitalk 수행 시간

(c) cit-Patents 수행 시간

(d) LiveJournal 수행 시간

(e) Orkut 수행 시간