웹 컨텐츠에서 강결합요소를 이용한 순환 탐색 알고리즘

이우기, 이정훈
인하대학교 산업공학과, 성결대학교 컴퓨터공학부
wookeylee, BigJames.Lee@Gmail.com

Circuits Detection Algorithms Using Strongly Connected Components in Web Contents

Wookey Lee, James Lee
Industrial Department, Inha University, Computer Department, Sungkyul University

요 약

거대한 웹 컨텐츠 안에는 수많은 링크들로 인한 순환들이 존재하게 된다. 그 순환들은 강하게 뭉쳐있는 실타래 처럼, 강하게 결합한 순환들의 덩어리 형태로 존재하게 된다. 웹 컨텐츠는 흔히 방향그래프로 표현되는데, 즉 웹 컨텐츠에서 나타나는 수많은 링크둘을 방향그래프에서 강결합요소를 이용하면 모든 순환을 효율적으로 발견할 수 있다. 본 논문에서는 강결합요소를 이용하여 거대한 그래프에서 보다 효율적으로 모든 순환을 찾아낼 수 있는 방법을 제시하였다.

1. 서 론

그래프에서 순환(*circuits*)을 찾는 문제는[13, 14]을 비롯한 많은 논문에서 오래전부터 다루어온 주제이다. 그래프(*Graph form*)는 화학, 전기공학, 사회학, 컴퓨터학 등의 문제를 추상적으로 표현하여 복잡한 문제를 풀기에 적합한 것으로 알려져 있다[13]. 본 연구에서는 웹을 대상으로 그래프 접근법(web-as-a-graph)을 사용하여 웹의 효과적인 탐색 및 구조화를 위한 문제해결을 시도하는 것으로, 이러한 연구는 최근에 시도되고 있다. 물론 웹을 대상으로 해도 눈높이(scalability)를 어디에 맞추는가에 따라 접근법과 목표가 다양할 수 있다. 이러한 관점으로 좀 더 구체적으로 분류하면 다음과 같다. 첫째, 상위레벨에서 보는 관점으로서 웹 서버를 하나의 노드로 인식하고 웹 서버사이의 연결구조를 그래프로 인식할 수도 있다. 이러한 접근법은 웹 서버의 부하분석이나 사용자들의 clickstream을 분석하는 연구에서 주로 이루어지고 있다[8]. 둘째, 웹 페이지를 노드로 하이퍼텍스트 링크를 아크로 인식하는 경우로서 이러한 접근법이 그중에서도 가장 많은데, 이는 검색엔진 혹은 웹정보검색 분야에서 주로 연구하고 있다. 그리하여 효율적인 검색시스템의 구현이나 메타 데이터베이스 및 인덱스 관리, 검색로봇 및 에이전트, 웹 사이트의 구성 등에 적용되고 있다[18, 19]. 셋째, 웹 페이지를 하나의 루트노드로 인식하고 그 웹 페이지의 컨텐츠를 개별적으로 나누어 그래프로 혹은 트리구조로 만드는 연구도 모바일 웹 분야에서 이루어지고 있다[15]. 본 연구에서는 그 중에서 두 번째 눈높이를 취하고자 한다. 물론 본 연구의 목표인 웹 페이지들 간의 검색 및 효율적 구조분석을 위하여 순환구조를 찾아내는 문제는 앞서 언급한 세 가지 대상 모두에서 공통적으로 등장할 수 있는 주제로서, 특히 웹과 같이 거대하며 계속 커지는 웹 그래프 상황에서는 그 해법의 중요성도 계속하여 커지고 있다. 예컨대, 검색로봇이 특정 웹 사이트를 방문하였을 때, 해당 웹 사이트에 속한 웹페이지들의 숫자가 매우 커서 검색로봇이 인식 혹은 저장할 수 있는 페이지 인식자의 범위를 넘어가는 경우도 발생 가능하다. 특히, 뒤에서 좀 더 상세히 언급하겠지만, 검색경로를 탐색하면서 방문하고 있는 경로상의 웹 페이지들이 순환하는 경우 웹 로봇이 루프에 빠져 해당 웹 사이트를 제대로 탐색하지 못하는 경우도 빈번하다. 이런 경우 어떤 연구에서는 웹 사이트에 대한 자세한 내용기반의 정향화 혹은 구조화를 포기하고 일정 깊이 이내의 페이지 만을 검색해도 중요 페이지는 탐색(crawl)할 수 있다는 제언도 내놓고 있는 실정이다[9]. 이는 웹 구조화가 쉽지 않다는 반증이 되기도 하며, 그 중에서 가장 어려운 문제의 하나가 웹 그래프 상에서 발생하는 순환(circuit)을 찾고 신속히 해소하는 것이라 하겠다. 순환이란 웹 그래프에서는 어떤 웹 페이지에서 시작해서 그 안에 포함된 하이퍼링크를 따라 다음 웹 페이

지를 방문하고, 차례차례 방문하다가 시작된 웹 페이지로 돌아오는 경우를 일컫는 것이다. 이것이 문제가 되는 이유는 여러 가지가 있는데, 우선 이러한 일련의 웹페이지의 경로가 매우 길어질 경우 방문경로를 모두 저장하고 있다가 시작 웹페이지로 돌아왔을 때 동일한 웹페이지인지를 인식하는 문제가 있고, 이 경로 중에서 일부만 중복되는 웹 페이지 방문 경로를 인식하는 문제가 있으며, 또한 긴 순환의 경우 그 구성요소의 중간에서 시작하여 해당 페이지로 돌아오는 경로를 중복해서 인식하는 문제가 매우 빈번히 발생한다는 점이다.

일반적으로 그래프에서 순환을 찾는 다양한 방법이 있다. [4, 5, 7, 14] 하지만 웹 컨텐츠는 그 자신만의 구조를 가지고 있기 때문에 그 특징을 이용하여 순환을 탐색하면 좀 더 효율적으로 순환을 탐색할 수 있다. 웹 컨텐츠 안에는 수많은 링크가 존재하며, 웹페이지들이 서로 복잡한 구조를 가진다. [그림 2]는 허리케인 정보를 제공하는 홈페이지 이다. 140개의 페이지와 332개의 링크들로 이루어져 있는 다소 작은 웹페이지 이다. 그 안에는 수많은 링크들이 엉켜있는 순환들 즉 강결합요소(Strongly Connected Conponents이하 SCC)를 이루거나 트리구조들의 집합으로 존재하게 된다. 트리구조는 자신들에게 다시 돌아오는 정점이 없는 구조이므로 순환을 찾을 때 무시하고, 순환들이 뭉쳐있는 강결합요소들만을 이용하여 웹 컨텐츠에서 순환을 탐색함으로서 효율적으로 순환을 탐색할 수 있다. [그림 1]은 웹 컨텐츠를 간단한 그래프구조로 표현한 것인데, 정점 1은 해당 웹페이지의 메인 페이지로서 여러 문서들로 링크가 존재한다. 6, 7, 8 정점의 경우 나무구조를 가짐으로써 순환과는 상관없다. 하지만 3, 4, 9, 10의 경우 두 개의 순환이 겹쳐진 SCC구조를 이루고 있기 때문에 그 SCC를 찾아내어 그 안에서 순환을 발견한다면 모든 순환을 효율적으로 탐색할 수 있다. 본 논문에서는 거대한 그래프와 같은 웹페이지들 속에서 SCC를 이용하여 모든 순환을 효율적으로 찾아내는 알고리즘을 제시하였다.

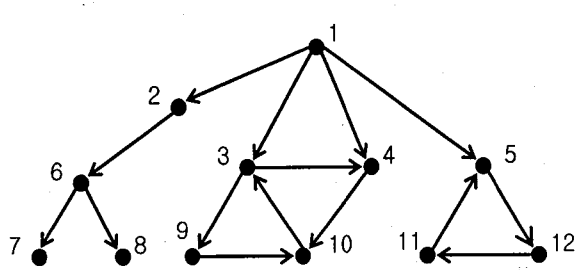

그림 1. 웹 컨텐츠의 그래프 구조.

이하 본 논문의 구성은 다음과 같다. 제2장에서는 문제에 대한 정의 및 표기법(notation)에 대해 기술한다. 제3장에서는 기본적인(naive) 순환탐색 알고리즘들을 정규화 표현(formal representation)하도록 하고, 제4장에서는 본 연구의 초점인 강연결에 기반한 알고리즘을 제시하고, 제5장에서는 실험한 내역을 설명하며, 마지막으로 제6장에서는 결론을 맺기로 하겠다.

2. 정의 및 표기법

그래프를 $G(V, E)$ 라 할 때, V 는 정점(*Vertex*)들의 집합이고, E 는 간선(*Edge*)들의 집합이다. 간선은 $v, w \in V$ 일 때, 정점들의 순서쌍 (v, w) 으로 표현할 수 있으며, v 에서 출발하여 w 로 가는 간선이다. 그래프에서 한 정점만의 집합으로 이루어진 경우 사소하다(*trivial*)라고 표현한다. [그림 1]에서 *tree* 구조의 가지들은 모두 사소한 정점들로 간주할 있다. [4] 본 논문에서 다루는 방향그래프는 주로 사소하지 않은 구조의 방향그래프를 다루기로 한다. *경로(Path)*는 그래프 G 안에 임의의 정점 v, w 가 있을때, 정점 v 에서 w 로 이르는 간선들의 순서쌍이다. $p : v \rightarrow w$ 로 쓸 수 있다. *순환(Circuits)*은 경로에서 다시 출발한 정점으로 돌아오는 경우이다. 예를 들면, v 에서 w 로 가는 경로에서 다시 v 로 돌아오는 경우이다($p : v \rightarrow w \rightarrow v$). 본 논문에서는 방향그래프(웹페이지는 방향성이 있기 때문에)를 다루기 때문에 순환을 "*circuits*" 이라는 용어로 표현한다. [3] *강결합요소(Strongly Connected Components)*는 모든 정점 간에 경로가 존재하는 방향그래프의 *maximal subgraph* 이다.[17] 즉, G 가 방향그래프 일 때, 그래프 G 안에 있는 각각의 v, w 가 서로 경로(*Path*)가 존재할 때($p_1 : v \rightarrow w$, $p_2 : w \rightarrow v$) *SCC* 라 한다. [13]

예컨대, [그림 1]의 경우 3 → 4 → 10 →...→ 9 및 5 → 12 → 11 등이 *SCC* 가 된다. 또한 후자의 *SCC* 처럼 그 자체가 하나의 순환이 될 수도 있지만, 일반적으로는 전자의 *SCC* 처럼 *SCC* 내에 3 → 4 → 10 → 3 및 3 → 9 → 10 → 3 처럼 복수의 순환이 존재할 수 있다.

3. 순환탐색 알고리즘

유방향 그래프에서 순환을 발견하는 효율적인 알고리즘으로 BACKTRACK 알고리즘[14]이 있다. 복잡도 $O(V + E)$를 보이며, 모든 순환을 발견함을 증명하였다. 이 알고리즘에서는 모든 정점에서 순환을 탐색하는 방법

을 사용함으로써 모든 순환을 탐색하는 방법을 사용하였다. 본 장에서는 BACKTRACK 순환탐색 알고리즘 및 다양한 순환 찾는 알고리즘들을 비교해보았다.

3.1 BACKTRACK 순환탐색 알고리즘

BACKTRACK 순환탐색 알고리즘은[14] 모든 순환을 찾기 위해 모든 정점을 기준으로 순환을 탐색하는 방법을 선택하였다. 순환을 찾는데 가장 문제가 되는 것은 중복순환(같은 순환을 다른 경로를 통해 다시 검색하는 것)인데, 모든 정점을 기준으로 검색하면 이를 피할 수 있다. 또한 모든 순환을 다 찾는 것을 정의 하는데 용이하다.

```
(1) Procedure circuit enumeration ;
(2)   begin
(3)     procedure BACKTRACK( integer value v, logical result f);
(4)       begin
(5)         logical g
(6)         f := false;
(7)         place v on point stack;
(8)         mark(v) := true;
(9)         place v on marked stack;
(10)         for w ∈ A(v) do
(11)             if  w < s then delete w from A(v)
(12)             else if w = s then
(13)             begin
(14)              output circuit from s to v given by point stack;
(15)              f := true;
(16)             end
(17)             else if  ¬ mark(w) then
(18)             begin
(19)                BACKTRACK(w, g);
(20)                f := f  V  g
(21)             end
(22)         if f = true then
(23)             begin
(24)               while top of marked stack ≠ v do
(25)                 begin
(26)                 u := top of marked stack;
(27)                 delete u from marked stack;
(28)                 mark(u) := false;
(29)                 end;
(30)               delete v from marked stack;
(31)               mark(v) :=false;
(31)             end;
(32)          delete v from point stack;
(33)       end;
(34)     integer n;
(35)     for i := 1 until v do mark(i) := false;
(36)     for s := 1 until v do
(37)       begin
(38)         b:BACKTRACK(s, flag);
(39)           while marked stack not empty do
(40)             begin
(41)               u:=top of marked stack
(42)               mark(u):=false;
(43)           delete u from marked stack;
(44)       end;
(45)   end;
(46) end;
```

알고리즘 1. BACKTRACK 순환탐색 알고리즘

간단하게 알고리즘을 살펴보면, 34~46번 줄 까지는 모든 정점을 출발점으로 순환을 탐색하기 위해서 함수를 호출하는 부분이다. 36번 줄에서 *v*는 정점개수이다. 즉 정점개수만큼 BACKTRACK함수를 호출하는 것을 알 수 있다. 1~33번 줄 까지가 순환나열함수이다. 3번 줄부터 살펴보면 BACKTRACK함수는 인자로 출발하는 정점과 함수를 호출하고 돌아올 때 순환을 발견하고 돌아오는지를 기억하기 위해 참조변수(*reference parameter*) *f* 를 사용하였다. 6번 줄에 mark[]는 배열을 이용하여 방문한 정점은 *true* 로 표시해서 이미 방문한 정점인지 여부를 검토한다. 7, 9번 줄을 보면 스택을 두 개를 사용하였다. 새로운 정점에 도착할 때 마다 스택에 넣고 다시 돌아오면 빼내면서 현재 스택상황을 저장하는 point stack과 방문하는 대로 모두 스택에 저장하여 현재까지 방문한 정점들을 모두 기억하는 marked stack이 있다. marked stack은 함수가 계속 재귀호출하면서 순환이 발견했을 때를 기억했다가 backtracking시 20번 줄에서 연산을 통해 22~33번 줄 조건에 들어간다. *f* 가 *true* 이면 이미 순환을 발견하고 backtracking하는 것을 의미한다. 이땐 marked stack에서 지금까지 방문한 정점들을 제거한다. 이것이 가지는 중요한 의미는 만약 순환을 발견하지 못하고 돌아오는 경로의 경우 22번 조건에서 빠져나가게 되므로, 나중에 그 정점들을 방문했을 때 다시 그 정점과 연결된 순환들을 탐색하지 않는다. 알고리즘은 핵심은 36번 줄에서 호출한 정점으로 시작하는 순환만을 발견한다는 것이다. 36번 줄에서 1로 함수를 호출하였으면

1로 시작하는 순환만을 순환으로 인정한다는 것이다. 예를 들면, 순환을 탐색하는 도중에 1로 돌아오지 않는 순환들은 모두 무시된다. 그렇기 때문에 결국 모든 순환이 중복 없이 발견한다는 것을 쉽게 알 수 있다. 또한 1로 시작하는 모든 순환을 발견하면 다시 36번 줄을 통해 2번 정점으로 시작하는 순환탐색을 시작하고, 탐색도중에 정점 1을 발견했을 때 그 정점을 삭제한다. 이 알고리즘은 결국 모든 정점에서 출발해서 순환을 찾아내기 때문에 비효율적으로 보일 수 있지만, *marked stack* 이나 참조변수 *f* 를 사용함으로써 각 정점들이 다시 같은 정점에 방문했을 때 비효율적으로 재탐색하는 것을 막음으로서 $O(V+E)$라는 효율적인 탐색속도를 보였다.

3.2 단일출발점 모든 순환탐색 알고리즘

```
(1) PROCEDURE OneVertex(int v)
(2)   BEGIN
(3)       //Initialize visited[] := false,  stack[] := false
(4)       visited[v] := true;
(5)       FOR each vertex w adjacent from v
(6)         BEGIN
(7)           IF visit[w] = false THEN
(8)             BEGIN
(9)               put w on stack;
(10)              OneVertex(w);
(11)            END
(12)          ELSE
(13)            BEGIN
(14)             comment Cutting Repetition Circuit Algo.
(15)             comment Enumerating Circuit Path Algo.
(16)            END
(17)        END
(18)      temp := Pop();
(19)      visited[temp] := false;
(20)  END
```

알고리즘 2. 단일출발점 모든 순환탐색 알고리즘

단일출발점 모든 순환탐색 알고리즘은 이름에서처럼 한 정점에서 출발하여 모든 순환을 다 찾는다. 웹 그래프를 탐색 할 때 한번 방문한 정점을 기록함으로써, 방문이 기록된 정점을 방문했을 때 순환을 발견하는 원리를 이용한 순환탐색 알고리즘이다. 임의의 한 정점에서 출발하여 새로운 정점을 순회하면서 순환을 탐색하는 데, 방문했던 정점들을 방문했던 것을 기록하기위해 방문기록배열(*Visited Record Array*)이 필요하다. 또한, 순환경로를 출력하기 위한 경로저장이 필요하다. 경로저장은 여러 가지 형태로 구현될 수 있는데, 재귀호출 방식을 이용하므로 스택형태의 자료구조를 사용하였다. 순환탐색시 정점을 방문하는 순서대로 스택(*stack*)에 넣고 방문기록을 함으로써 이미 방문한 정점을 방문할 경우 방문기록에 있으므로 순환이 탐색되었음을 알 수 있고, 스택을 이용해서 순환경로를 출력할 수 있다. 방문기록이 되어있지 않는 정점을 탐색했을 경우에는 그 정점을 스택에 넣고 반복한다. 재귀호출을 이용하므로 정점을 스택에 넣은 재귀순환이 끝나면 호출했던 함수로 돌아오면서 다시 스택에서 제거한다. 스택이 모두 비워지면 순환탐색은 끝난다. 그래프를 인접리스트로 표현하면 정점의 개수가 V 개, 간선의 개수가 E 개, 순환수가 C 개 일 때, 모든 그래프를 탐색하는데 $O(|E|)$ 만큼의 시간이 걸리고, 각 정점에서 순환이 발견될 때마다 $O(|V|\cdot|E|)$의 시간이 걸리기 때문에, $O(|E|(|V|\cdot|C| + 1))$의 복잡도를 보인다. 또한 순환을 발견한 후, 이전 방문했던 정점으로 이동하여 그 정점에서 또 다른 인접한 정점이 없는지 확인한다. 만약 없다면 다시 이전 방문했던 정점으로 돌아간다. 이전 정점으로 돌아가면 방문기록배열에서 다시 방문하지 않은 상태로 값을 지워주는데, 이렇게 함으로써 한 정점에서 여러 정점으로 갈 수 있는 간선이 있더라도, 간선을 순서대로 하나씩 방문하고 다시 원래의 정점으로 돌아왔을 때 방문여부배열이 같게 되기 때문에 결국 모든 정점을 탐색하고 순환을 찾을 수 있다. 이 알고리즘은 재귀적으로 탐색할 때 출발점에서 향하는 다른 간선을 통해, 같은 순환을 중복해서 찾아낸다는 단점이 있다. 결국 중복된 순환 때문에 중복순환을 제거해주는 알고리즘이 필요하다. 중복순환의 발견은 다소 비효율적이지만, 중복순환의 개수는 해당 웹 페이지들 간에 접근 빈도가능성과 관련이 있기 때문에 웹 페이지를 구조화 할 때, 사용하기에 적합하다.

3.3 CutNode알고리즘

정점제거 순환탐색 알고리즘은 모든 정점에서 순환을 탐색하고, 순환은 각 출발한 정점으로 시작하는 순환만을 탐색한다. 이 알고리즘은 한번 방문한 정점을 제거함으로써 중복된 순환을 발견하지 않는다. 탐색 시 오직 탐색을 시작하는 정점으로 돌아오는 순환만을 순환으로 발견하기 때문에 시작점을 기억하기 위한 인자를 초기값으로 할당받는다. 그 시작 정점으로 모든 순환을 발견하고 돌아오면 그 정점을 지우고, 다음 정점으로부터 다시 순환을 탐색한다. 모든 순환을 탐색할 수 있지만, 각 정점을 기준으로 순환을 탐색해 나갈 때마다 순환이 존재하는 같은 *subgraph* 로 들어오게 됐을 때, 비교복잡도가

올라가게 된다.

그래프를 인접리스트로 표현하면 정점의 개수가 V 개, 간선의 개수가 E 개, 순환수가 C 개 일 때, 모든 그래프를 탐색하는데 $O(|V|\cdot|E|)$ 만큼의 시간이 걸리고, 각 정점에서 순환이 발견될 때마다 $O(|V|\cdot|E|)$의 시간이 걸리기 때문에, $O(|V|\cdot|E|(|C| + 1))$의 복잡도를 보인다.

```
(1) procedure CutNode_init()
(2)    BEGIN
(3)        FOR i := 1 to v
(4)          BEGIN
(5)            IF(AdjacencyList[i].next ≠ null THEN
(6)                s := i;
(7)          END
(8)
(9)        put s on stack;
(10)       CutNode(s, s);
(11)       Pop();
(12)       AdjacencyList[i].next := null;
(13)       AdjacencyList[i].key := null;
(14)   END
(15) PROCEDURE CutNode(int v, int s)
(16)   BEGIN
(17)       visited[v] := true;
(18)       FOR each vertex w adjacent from v
(19)       BEGIN
(20)             IF w ≠ s THEN
(21)               BEGIN
(22)                   IF visited[w] = true THEN
(23)                         continue;
(24)                   put w on stack;
(25)                   CutNode(w, s);
(26)               END
(27)             ELSE
(28)               BEGIN
(29)                  comment Enumerating Cycle Path
(30)               END
(31)       END
(32)       temp := Pop();
(33)       visited[temp] := false;
(34)   END
```

알고리즘 3. CutNode 순환탐색 알고리즘

4. *SCC* 를 이용한 순환탐색 알고리즘

지금까지 순환탐색 알고리즘을 알아보았다. 각 순환탐색들은 장단점이 있지만, 가장 효율적으로 알려진 BACKTRACK 순환탐색 알고리즘에서도[14] 모든 정점에서 순환을 탐색함으로써 최소 정점 수만큼 순환들을 곱한 비교복잡도($O(|V|\cdot|E|(|C| +1))$)를 요구한다.

본 논문에서 제시할 알고리즘에서는 모든 정점에서 순환을 찾지 않고, *SCC* 를 이용해서 모든 정점으로 순환탐색을 하지 않아도 모든 순환을 찾는 방법을 제시하였다. 알고리즘 분석에 들어가기 앞서, 과연 *SCC* 의 *root* 들만을 이용하여 모든 순환이 다 발견될수있을까? 강결합요소는 2장에서 설명한 바와 같이 동일한 연결요소(Connected Components) 안에 있는 정점들은 모두 연결되었다는 것을 의미한다. 즉 두개 이상의 정점으로 이루어진 강결합요소 안에는 반드시 순환이 존재한다는 것을 알 수 있다. 그것은 어떤 정점으로 탐색하던지 모든 정점에 이를 수 있고, 순환을 발견 할 수 있다는 것을 의미한다.

정리 1. 강결합 요소는 반드시 순환이 존재한다.

증명. 순환을 가지고 있는 임의의 그래프 G는 강결합요소를 갖지 않는다고 가정하자. 그래프 G는 강결합 요소를 가지고 있지 않으므로 G내의 어떠한 임의의 정점을 선택하더라도 다시 그 정점으로 돌아올 수 있는 간선은 존재하지 않는다는 것을 의미한다. 결국 자기 자신으로 돌아올 수 있는 간선이 존재하지 않기 때문에 순환도 존재하지 않는다. 가정과 모순이 되므로 정리1은 참이 된다.

정리 2. 방향그래프를 G 라 할 때, 그래프 내에 있는 *SCC* 만을 발견해서 순환을 찾는다면, 그래프 G 안에 있는 모든 순환을 발견할 수 있다.

증명. 방향그래프 G 안에 두개의 SCC_1, SCC_2 가 있다고 하고, 이 두개의 *SCC*를 포함한 순환도 추가로 존재한다고 가정하자. 그렇다면 두 SCC_1, SCC_2 간에는 서로를 연결할 수 있는 간선이 존재한다는 것이고, 두 *SCC* 는 한 *SCC*가 되어야 하므로 모순이다. 즉 두 *SCC* 를 포함한 순환이 없으므로 각 *SCC* 안에서 순환만 발견한다면 그래프 G 안에 있는 모든 순환을 발견할 수 있다. SCC_1, SCC_2 각자 안에 있는 순환들은 모두 연결되어 있으므로 모든 순환을 찾을 수 있는 것은 자명하다.

[알고리즘 4]는 *root* 를 이용해서 *SCC* 를 찾는 [13] 알고리즘을 응용한 알고리즘으로서, 단순히 *SCC* 만을 찾는 것이 아니라 *SCC* 를 제외한 모든 링크를 삭제시킨다. *SCC* 를 제외한 나머지 링크를 삭제하더라도 정리 1, 2에서 모든 순환을 찾을 수 있음을 보장해준다.

또한 서로 다른 *SCC* 간에 중복으로 탐색하는 것을 막아 준다.

```
(1) INTEGER i;
(2) PROCEDURE FindingSCC(Node v, ref bool f)
(3)   BEGIN
(4)     Node w, bool g := false, f := false;
(5)     LOWLINK(v) := v := i := i + 1;
(6)     put v on stack;
(7)     FOR w in the adjacency list of v DO
(8)     BEGIN
(9)       IF w is not yet numbered THEN
(10)      BEGIN
(11)        FindingSCC(w, ref g);
(12)        LOWLINK(v) := min(LOWLINK(v), LOWLINK(w));
(13)        f := f | g;
(14)        IF f = true THEN
(15)        BEGIN
(16)          delete edge(v, w);
(17)          IF v has adjacent edge THEN
(18)          BEGIN
(19)            f := false;
(20)          END
(21)        END
(22)        ELSE IF w < v THEN
(23)        BEGIN
(24)          IF w is on the stack THEN
(25)           LOWLINK(v) := min(LOWLINK(v), w);
(26)        END
(27)        ELSE IF SCC(w) = true THEN
(28)         delete edge(v, w);
(29)      END
(30)    END
(31)    IF (LOWLINK(v) = v) THEN
(32)      BEGIN comment v is the root of a component;
(33)        WHILE w on top of stack satisfies w ≥ v DO
(34)          BEGIN
(35)            pop w from stack;
(36)            SCC(w) := true;
(37)          END
(38)        f := true;
(39)        IF v has adjacent edge THEN
(40)          root(v) = true;
(41)      END
(42)  END
```

알고리즘 4. 모든 *SCC* 검출 알고리즘

간단히 살펴보면, 7번 줄에서 부터 인접한 새로운 정점을 발견해 나가면서 재귀호출하여 정점들의 번호를 매긴다. 또한 각 정점에는 LOWLINK를 매기고 SCC의 LOWLINK를 가진 정점이 root가 된다. 눈여겨봐야 할 부분은 16, 28번 줄인데, 16번 줄은 SCC를 찾고 돌아올 때 그 SCC을 가리키는 간선이다. 또한 28번줄은 이미 발견한 SCC를 가리키는 간선이다. 이 두 간선은 정리 1,2에 의해 순환을 검색하는데 아무 상관이 없으므로 삭제한다. 이런 간선들을 삭제하기 위해서 프로시져의 인수로 참조변수를 사용하였다. 재귀호출 후 돌아오면서 *SCC* 를 발견한 후 다른 *SCC* 에 포함되는 간선이 아니면 계속 삭제하게 된다. 모든 프로시져를 마치면 *SCC* 들의 집합만 남게 된다. 복잡도를 보면 모든 간선을 한 번 씩 방문하므로 $O(E)$이다.

```
(1) PROCEDURE CircuitDetectionUsingSCC_init()
(2)   BEGIN
(3)   visited[] := false,  possibility[] := false, stack[] := false
(4)       FOR i = 1 to v
(5)       BEGIN
(6)         IF root[i] = true THEN
(7)         BEGIN
(8)           root := AdjacencyList[i];
(9)           CircuitDetectionUsingSCC(root);
(10)        END
(11)      END
(12)  END
(13) PROCEDURE CircuitDetectionUsingSCC(Node v)
(14) BEGIN
(15)     visited[v] := true;
(16)     put v on stack;
(17)       FOR each vertex w adjacent from v
(18)       BEGIN
(19)       IF w ≠ root THEN
(20)         BEGIN
(21)           IF visited[w] = true THEN
(22)           BEGIN
(23)             IF possibility[w] = false THEN
(24)               BEGIN
(25)                 continue;
(26)               END
(27)             comment Enumerating Circuit Path;
(28)             continue;
(29)           END
(30)           CircuitDetectionUsingSCC(w);
(31)         END
(32)       ELSE
(33)           BEGIN
(34)           //Enumerating Circuit Path;
(35)           continue;
(36)           END
(37)       END
(38)     visited[v] := false;
(39)     possibility[pop()] := false;
(40) END
```

알고리즘 5. *SCC* 를 이용한 순환탐색 알고리즘

[알고리즘 5]를 살펴보면, 1번부터 12번까지는 [알고리즘 4]를 이용해 그래프를 각 *SCC* 로 잘라 놓고 각각의 그래프 안에서 *root* 만 저장해놓은 *root*[]를 이용해

SCC 순환탐색 알고리즘을 호출하는 부분이다. 함수가 호출되면 그 내부 *SCC* 의 순환을 탐색을 시작하는데 방문여부를 기록하는 *visited*[]배열, 그래프의 경로를 출력할 스택에 현재 경로를 저장한다. 17번부터 인접리스트 순서대로 검색을 시작하는데, 19번에서 새로 방문한 정점이 *root* 와 같다면 순환을 발견한 것이므로 33~36번이 실행된다. 만약 *root* 와 같지 않다면 이미 방문한 정점인지 여부를 살피고(21번) 아직 방문하지 않은 새로운 정점이면 다시 그 정점으로부터 재귀호출을 시작한다(30번). 마지막으로 root와 같지 않고, 만약 이미 방문한 정점을 또 발견한 것이면 마지막으로 순환이지만 중복 순환이 아닌지 여부를 살펴야한다(23). 최초에 possibility[]변수에 모든 정점을 순환가능성(true)이 있도록 지정한 뒤 해당 정점이 이미 순환에 사용되어 순환가능성이 없어지면, 이미 방문한 정점에 방문 하더라도 다시 순환을 발견하지 않는다. 중요한 것은 언제 가능성이 없는 정점이 되느냐를 결정하는 것이다. 39번에 보면 possibility[]변수가 false가 된다. 그 시기는 그래프를 탐색하다가 정점들을 방문하고 돌아오는 시기인데, 결국 탐색을 마치고 스택에서 제거되면서 그 정점은 순환가능성이 없어진다. 결국 중복순환가능성을 체크함으로써 모든 정점을 기준으로 순환을 탐색하지 않더라도 중복 없이 모든 순환을 찾을 수 있게 된다.

5. 실 험

본 논문에서 제시한 알고리즘이 거대한 웹 컨텐츠 속에서 효율적인지 확인하기 위해서는 가장 복잡한 형태의 웹 구조가 필요하다. 웹 구조가 복잡하다는 것은 웹 페이지 간에 링크 무수히 존재한다는 것이다. 즉 특정 웹 페이지 안에서 모든 페이지들끼리 모두 링크가 있다면 가장 복잡한 웹페이지 구조라고 말할 수 있다. 그래프로 표현하면, 방향그래프 *G* 가 있을 때, *G* 안에 있는 임의의 정점 *V* 에서 *V* 자신을 포함한 *G* 안에 있는 모든 정점에 간선 존재하는 것을 완전그래프(*complete graph*)라 한다. [3] 결국 거대한 웹 컨텐츠란 거의 완전그래프와 가까운 복잡한 구조를 이룰 것이다. 완전그래프에서 정점의 개수가 *n* 개 일 때, 순환의 개수는 $\sum_{k=1}^{n} \frac{n!}{(n-k)!k}$ 이다. 여기서 *k* 는 그 순환요소의 개수인데, 즉 *k* 가 2이면 정점 2개로 이루어진 순환이다. 결국 정점개수만큼까지 순환을 찾으면 모든 순환의 개수를 알 수 있다.

표 1. 완전그래프에서 정점개수 당 순환개수와 BACKTRACK 알고리즘과 복잡도 비교

Number of vertice	3	4	5	6	7	8	9
Number of circuits	8	24	89	415	2372	16072	125673
CduSCC Algorithm	15	64	325	1956	13699	109600	986409
BACKTRACK Algorithm	24	96	445	2490	16604	128576	1131057

위 실험을 통해 그래프에서 *SCC* 를 이용해서 순환을 찾았을 때 모든 순환을 찾을 수 있음을 알 수 있다. 다음은 웹 컨텐츠 구조의 그래프에서 *SCC* 를 이용해서 순환을 찾았을 때 비교복잡도 실험함으로써 웹 구조에서 *SCC* 를 이용한 순환탐색이 효율적임을 보이고자 한다.

또한 본 연구에서는 실제 웹 사이트(*hurricane.lsu.edu*)를 대상으로 본 연구에서 제안하는 알고리즘을 적용하여 실험하였다. 실험 결과는 [그림 2]에서 제시되는 바와 같이 전체 140개의 노드와 332개의 간선들로 구성되어 있는데, 이중에서 그림에서 보이는 바와 같은 *SCC* 들 및 트리구조가 섞여있는 것을 알 수 있다. 이를 기존의 알고리즘으로 분석할 경우 긴 순환들의 경우 중복해서 찾는 문제 등의 이유로 분석 시간이 지체되고 부정확한 결과를 낳았지만, 본 알고리즘의 경우 정확한 수의 회로를 중복없이 짧은 시간 내에 탐색하는 것을 확인할 수 있었다.

다음으로는 알고리즘들 사이에서 좀 더 구체적인 비교를 위하여 고정된 *SCC* 에서 정점수를 늘리는 방법과 *SCC* 자체의 증가를 통해 복잡도를 비교하는 실험을 수행하였다. [그림 3]의 실험은 동일한 *SCC* 를 가진 그래프에서 정점수만 늘려가며 실험했다. BACKTRACK 알고리즘의 경우 순환이 없음에도 불구하고 복잡도가 올라가고 있음을 알 수 있다. [그림 4]는 *SCC* 수를 늘려가며 실험했다. 두 알고리즘 모두 비교횟수가 증가하였으나 *SCC* 부분만 순환탐색하므로 BACKTRACK 알고리즘보다 효율적으로 순환을 찾을 수 있음을 보였다.

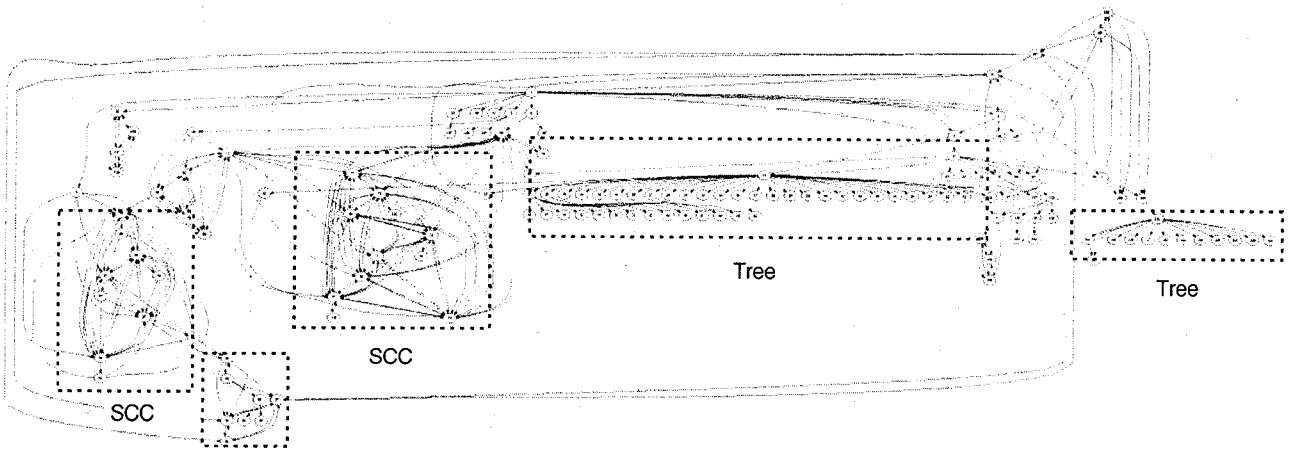

그림 2. 순환(*circuits*)과 트리구조들로 이루어진 웹 페이지(*hurricane.lsu.edu*)

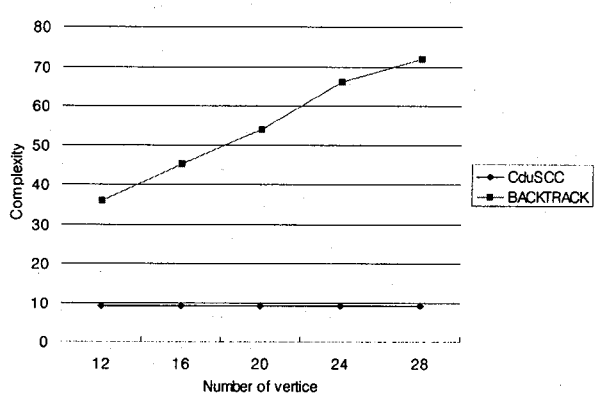

그림. 3 고정된 *SCC* 에서 정점수에 따른 복잡도 비교

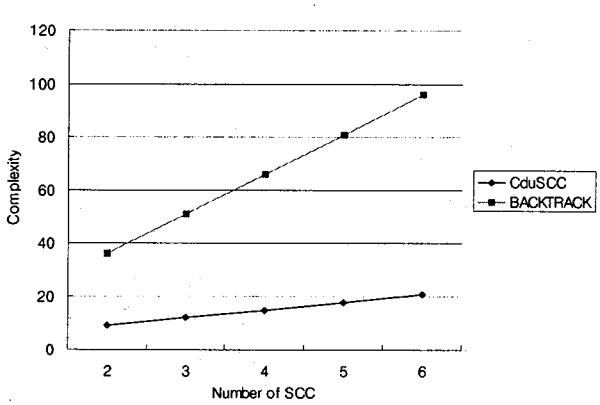

그림. 4 *SCC* 증가에 따른 복잡도 비교

6. 결 론

인터넷의 발전은 폭발적으로 이루어지고 있으며, 앞으로도 더 빠른 속도로 넓어져 갈 것으로 전망되면서, 웹 컨텐츠들을 효과적으로 또한 효율적으로 분석 및 관리하기 위해 구조화하려는 시도가 계속하여 이루어지고 있으며 특히 수리적 관점에서 구조화하는 문제는 쉽지 않으며 그 중요성이 계속 커지고 있다. 본 연구에서는 웹을 대상으로 그래프 접근법(web-as-a-graph)을 사용하여 웹 검색 및 효율적 구조분석을 저해하는 가장 큰 문제의 하나인 웹 페이지들 사이의 순환(curcuits)구조를 찾아내는 문제를 해결하려는 목표를 가지고 있다. 특히 본 논문에서는 *SCC* 를 이용하여 기존의 BACKTRACK 관점의 알고리즘들보다 훨씬 우수한 웹 컨텐츠 안에서 효율적으로 순환을 찾아내는 알고리즘을 제시하였다. 본 알고리즘을 실제 웹 사이트를 대상으로 구현하여 실험하여 정확한 숫자의 순환을 탐색해내고, 성능관점에서 탁월한 점을 입증하였다.

추후 연구주제는 본 연구의 결과를 바탕으로 탐색된 웹 사이트들을 다양한 형태로 구조화하는 시도와 그러한 검색결과를 바탕으로 검색엔진 알고리즘들과 비교하는 정보검색 관점에서의 연구가 후속될 것이다. 또한 시스템으로 구현 관점에서의 다양한 연구주제가 있으며, 결국 상용 검색엔진의 검색결과와 비견되는 새로운 검색엔진을 만드는 노력도 필요하다.

7. 참고문헌

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. Data Structure and Algorithms.

[2] B. Jiang. I/O-and CPU-optimal recognition of strongly connected components. Information Processing Letters, 45(3):111-115, March 1993

[3] C. Nicos. GRAPH THEORY An Algorithmic Approach.

[4] G. Loizou, P. Thanisch. Enumerating the Cycles of a

Digraph: A New Preprocessing Strategy, Information Sciences 27:163-182, 1982
[5] G. Nivasch. Cycle Detection Using a Stack, 2004
[6] J. Garofalakis, P. Kappos, and D. Mourloukos, Web Site Optimization Using Page Popularity, IEEE Internet Computing, 3(4): 22-29, July-Aug 1999
[7] J. Szwarcfiter, P. Lauer. A Search Strategy for the elementary cycles of a directed graph, 1976
[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos, On Power-law Relationships of the Internet Topology. In Proc. SIGCOMM, 251-262, 1999.
[9] M. Najork, and J. Wiener, Breadth-first Crawling Yields High-quality Pages, In Proc. WWW (2001), 114-118
[10] M. Sharir. A strong-connectivity algorithm and its application in data flow analysis. computers and Mathematics with applications, 7:67-72, 1981
[11] N. Esho and S. Eljas. On Finding the Strongly Connected Components in a Directed Graph. Information Processing Letters, 49:9-14, 1993
[12] R. Botafogo, E. Rivlin, B. Shneiderman. Structual Analysis of Hypertexts : Identifying Hierarchies and Useful Metrics, 1992
[13] R. Tarjan. Depth first search and linear graph algorithms. SIAM journal of Computing, 1(2):146-160, June 1972.
[14] R. Tarjan. Enumerating of the elementary circuits of a directed graph. SIAM journal of Computing,2(3):211-216, September 1973
[15] S. Kang, W. Park, and Y. Kim : Dynamically Personalized Web Service System to Mobile Devices. FQAS 2006: 416-426
[16] S. Sahni, E. Horowitz. Fundamental of computer algorithm, 1985
[17] S. Skiena. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Reading, MA: Addison-Wesley, 1990
[18] W. Lee, J. Geller, Semantic Hierarchical Abstraction of web Site Structures for web Searchers, Journal of Research and Practice in Information Technology, 36(1) (2004),71-82
[19] W. Lee, Hierarchical Web Structuring from the Web as a Graph Approach with Repetitive Cycle Proof. APWeb Workshops 2006: 1004-1011

부 록

1. *hurricane.lsu.edu* 의 노드구조

NodeName	FullPath
Node0	http://hurricane.lsu.edu
Node1	http://hurricane.lsu.edu/navbar.htm
Node2	http://hurricane.lsu.edu/home_page.htm
Node3	http://hurricane.lsu.edu/newsbriefs.htm
Node4	http://hurricane.lsu.edu/personnel.htm
Node5	http://hurricane.lsu.edu/research.htm
Node6	http://hurricane.lsu.edu/publications.htm
Node7	http://hurricane.lsu.edu/academic_programs.htm
Node8	http://hurricane.lsu.edu/conferences.htm
Node9	http://hurricane.lsu.edu/meetings.htm
Node10	http://hurricane.lsu.edu/current_storm_info.htm
Node11	http://hurricane.lsu.edu/hurricane_prep.htm
Node12	http://hurricane.lsu.edu/hurricane_links.htm
Node13	http://hurricane.lsu.edu/contact_us.htm
Node14	http://hurricane.lsu.edu/in_the_news.htm
Node15	http://hurricane.lsu.edu/newsarchives.htm
Node16	http://hurricane.lsu.edu/_in_the_news/sciam-0206.1.htm
Node17	http://hurricane.lsu.edu/_in_the_news/july16advocate.htm
Node18	http://hurricane.lsu.edu/_in_the_news/phillyinquirer100804.htm
Node19	http://hurricane.lsu.edu/_in_the_news/apwire0904.htm
Node20	http://hurricane.lsu.edu/_in_the_news/dallas0914.htm
Node21	http://hurricane.lsu.edu/_in_the_news/lsuresearch0704.htm
Node22	http://hurricane.lsu.edu/_in_the_news/lsuhighlights0704.htm
Node23	http://hurricane.lsu.edu/_in_the_news/tmemag0404.htm
Node24	http://hurricane.lsu.edu/_in_the_news/1103tp.htm
Node25	http://hurricane.lsu.edu/_in_the_news/sept03advocate.htm
Node26	http://hurricane.lsu.edu/_in_the_news/lsutoday0703.htm
Node27	http://hurricane.lsu.edu/_in_the_news/jan03_advocate.html
Node28	http://hurricane.lsu.edu/_in_the_news/lsuhighlights0103.htm
Node29	http://hurricane.lsu.edu/_in_the_news/nov02_dailyworld.htm
Node30	http://hurricane.lsu.edu/_in_the_news/oct2_nytimes.htm
Node31	http://hurricane.lsu.edu/_in_the_news/lsuhighlights0902.htm
Node32	http://hurricane.lsu.edu/_in_the_news/june_02_naples.htm
Node33	http://hurricane.lsu.edu/_in_the_news/june02_advocate.html
Node34	http://hurricane.lsu.edu/_in_the_news/april30_ny_times.htm
Node35	http://hurricane.lsu.edu/_in_the_news/april2002_chronicle.htm
Node36	http://hurricane.lsu.edu/_in_the_news/april21_advocate.htm
Node37	http://hurricane.lsu.edu/_in_the_news/lsuhighlights0102.htm
Node38	http://hurricane.lsu.edu/_in_the_news/houston.htm
Node39	http://hurricane.lsu.edu/_in_the_news/oct3_sciencenow.htm
Node40	http://hurricane.lsu.edu/_in_the_news/october_sciam.htm
Node41	http://hurricane.lsu.edu/_in_the_news/sept17_nocb.htm
Node42	http://hurricane.lsu.edu/_in_the_news/sept02_advocate.htm
Node43	http://hurricane.lsu.edu/_in_the_news/july14_advocate.htm
Node44	http://hurricane.lsu.edu/_in_the_news/aug08_disaster.htm
Node45	http://hurricane.lsu.edu/_in_the_news/may20_scinew.htm
Node46	http://hurricane.lsu.edu/sharedcontent/dws/spt/cycling/tour/vit index.html

Node47	http://hurricane.lsu.edu/sports/othersports
Node48	http://hurricane.lsu.edu/sports/colleges/recruiting
Node49	http://hurricane.lsu.edu/sharedcontent/dws/spt/columnists/fluksa/vitindex.html
Node50	http://hurricane.lsu.edu/sharedcontent/dws/spt/columnists/srichardson/vitindex.html
Node51	http://hurricane.lsu.edu/sharedcontent/dws/spt/columnists/mzeske/vitindex.html
Node52	http://hurricane.lsu.edu/sharedcontent/dws/spt/horseracing/vitindex.html
Node53	http://hurricane.lsu.edu/sharedcontent/dws/spt/motorsports/vitindex.html
Node54	http://hurricane.lsu.edu/sharedcontent/dws/spt/olympics/vitindex.html
Node55	http://hurricane.lsu.edu/sharedcontent/dws/fea/texasliving/columnists/bdover/vitindex.html
Node56	http://hurricane.lsu.edu/undergrad_conc.htm
Node57	http://hurricane.lsu.edu/undergrad_minor.htm
Node58	http://hurricane.lsu.edu/graduate_minor.htm
Node59	http://hurricane.lsu.edu/news/abr.shtml
Node60	http://hurricane.lsu.edu/entertainment/coupons
Node61	http://hurricane.lsu.edu/space
Node62	http://hurricane.lsu.edu/_in_the_news/search
Node63	http://hurricane.lsu.edu/_newsbriefs/hefmtg.html
Node64	http://hurricane.lsu.edu/_newsbriefs/dsm0905.html
Node65	http://hurricane.lsu.edu/abbeypaper.htm
Node66	http://hurricane.lsu.edu/story.htm
Node67	http://hurricane.lsu.edu/story1.htm
Node68	http://hurricane.lsu.edu/project_summary.htm
Node69	http://hurricane.lsu.edu/research_reports.htm
Node70	http://hurricane.lsu.edu/newsletters.htm
Node71	http://hurricane.lsu.edu/research1.htm
Node72	http://hurricane.lsu.edu/paper1a.htm
Node73	http://hurricane.lsu.edu/paper2A.htm
Node74	http://hurricane.lsu.edu/paper3a.htm
Node75	http://hurricane.lsu.edu/paper4a.htm
Node76	http://hurricane.lsu.edu/paper5a.htm
Node77	http://hurricane.lsu.edu/paper6a.htm
Node78	http://hurricane.lsu.edu/paper7a.htm
Node79	http://hurricane.lsu.edu/planning.htm
Node80	http://hurricane.lsu.edu/lsu_sponsors.htm
Node81	http://hurricane.lsu.edu/cosponsors.htm
Node82	http://hurricane.lsu.edu/registration_form.htm
Node83	http://hurricane.lsu.edu/paper8a.htm
Node84	http://hurricane.lsu.edu/faculty.htm
Node85	http://hurricane.lsu.edu/phd_fellows.htm
Node86	http://hurricane.lsu.edu/grad_students.htm
Node87	http://hurricane.lsu.edu/undergradhelp.htm
Node88	http://hurricane.lsu.edu/staff.htm
Node89	http://hurricane.lsu.edu/facultyretired.htm
Node90	http://hurricane.lsu.edu/graudatestipends.htm

Node91	http://hurricane.lsu.edu/phd_fellowsformer.htm
Node92	http://hurricane.lsu.edu/graduate_programs.htm
Node93	http://hurricane.lsu.edu/fall2001.htm
Node94	http://hurricane.lsu.edu/ocs.htm
Node95	http://hurricane.lsu.edu/ce4445.htm
Node96	http://hurricane.lsu.edu/graduates.htm
Node97	http://hurricane.lsu.edu/undergraduategrads.htm
Node98	http://hurricane.lsu.edu/staffgrads.htm
Node99	http://hurricane.lsu.edu/robcomments.htm
Node100	http://hurricane.lsu.edu/levcomments.htm
Node101	http://hurricane.lsu.edu/sajocommens.htm
Node102	http://hurricane.lsu.edu/activities.htm
Node103	http://hurricane.lsu.edu/facilities.htm
Node104	http://hurricane.lsu.edu/participatingg_units.htm
Node105	http://hurricane.lsu.edu/_dax/dax1.html
Node106	http://hurricane.lsu.edu/hurricane_eng.htm
Node107	http://hurricane.lsu.edu/dsm.htm
Node108	http://hurricane.lsu.edu/other_hazards.htm
Node109	http://hurricane.lsu.edu/courses.htm
Node110	http://hurricane.lsu.edu/graduatestipends.htm
Node111	http://hurricane.lsu.edu/oldcourses.htm
Node112	http://hurricane.lsu.edu/coursessummer05.htm
Node113	http://hurricane.lsu.edu/coursesspring05.htm
Node114	http://hurricane.lsu.edu/coursesfall04.htm
Node115	http://hurricane.lsu.edu/coursessummer04.htm
Node116	http://hurricane.lsu.edu/coursesspring04.htm
Node117	http://hurricane.lsu.edu/coursesfall03.htm
Node118	http://hurricane.lsu.edu/coursessummer03.htm
Node119	http://hurricane.lsu.edu/coursesspring03.htm
Node120	http://hurricane.lsu.edu/coursesfall02.htm
Node121	http://hurricane.lsu.edu/coursessummer02.htm
Node122	http://hurricane.lsu.edu/coursesspring02.htm
Node123	http://hurricane.lsu.edu/coursesfall01.htm
Node124	http://hurricane.lsu.edu/hurricane_chemical_internet.htm
Node125	http://hurricane.lsu.edu/nat_hurr_conf_.htm
Node126	http://hurricane.lsu.edu/publicpolicysymposium.htm
Node127	http://hurricane.lsu.edu/bevenposter2.pps
Node128	http://hurricane.lsu.edu/workshopsarchive.htm
Node129	http://hurricane.lsu.edu/monday.htm
Node130	http://hurricane.lsu.edu/thursday.htm
Node131	http://hurricane.lsu.edu/sept20_meet.htm
Node132	http://hurricane.lsu.edu/monday_form.htm
Node133	http://hurricane.lsu.edu/thursday_form.html
Node134	http://hurricane.lsu.edu/before.htm
Node135	http://hurricane.lsu.edu/during.htm
Node136	http://hurricane.lsu.edu/after.htm

Node137	http://hurricane.lsu.edu/directions.html
Node138	http://hurricane.lsu.edu/mission_statement.htm
Node139	http://hurricane.lsu.edu/katrinacontact.htm

2. 긴 순환들의 예(숫자는 노드 번호임)

1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 79-> 91-> 86-> 87-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 79-> 91-> 86-> 87-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 79-> 91-> 86-> 87-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 86-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 86-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 86-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 86-> 87-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 86-> 87-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 86-> 87-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 91-> 86-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 91-> 86-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 91-> 86-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 91-> 86-> 87-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 91-> 86-> 87-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 79-> 91-> 86-> 87-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 86-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 86-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 86-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 86-> 87-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 86-> 87-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 86-> 87-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 91-> 86-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 91-> 86-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 91-> 86-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 91-> 86-> 87-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 91-> 86-> 87-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 91-> 86-> 87-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 81-> 84-> 79-> 91-> 86-> 87-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 86-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 86-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 86-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 86-> 87-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 86-> 87-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 86-> 87-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 91-> 86-> 25-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 91-> 86-> 25-> 27-> 26-> 1 .
1-> 2-> 3-> 6-> 55-> 70-> 71-> 73-> 75-> 77-> 74-> 78-> 92-> 82-> 83-> 80-> 85-> 79-> 91-> 86-> 26-> 1 .