2005 한국경영과학회/대한산업공학회 춘계공동학술대회
2005년 5월 13일 ~ 14일, 충북대학교

유방향그래프에서의 순환 검출 알고리즘
A Cycle Detection Algorithm in Directed Graphs

이우기, 이정훈, 박상언, 김능회
[wook|bbo1234|skyloverd|nunghoy]@sungkyul.edu
성결대학교 컴퓨터공학과, 경기도 안양시 안양8동 산 147-2, 430-740

요 약

순환탐색 알고리즘 및 스택기반 알고리즘 등은 유방향그래프에서 순환과 순환경로를 발견하는 특정 정점으로부터 출발하여 연결된 그래프에서 순환을 탐색하는 것이다. 기존 연구의 단점은 모든 순환을 다 찾아내지지 못하는 경우라든지, 동일한 순환을 중복해서 찾아내는 문제가 있었다. 본 연구에서 제시하는 정점제거 순환탐색 알고리즘은 특정 정점의 순환을 발견한 뒤 그 정점을 삭제하므로 중복된 순환을 발견하지 않고 모든 순환을 찾을 수 있다. 또한 순환을 발견했을 때, 순환경로를 출력하는데 있어서 스택의 인덱스를 이용해, 저장경로를 탐색하지 않고 출력하는 방법을 제안하였다. 실험에서는 임의의 정점과 간선을 생성하여 그래프로 만들고, 각 알고리즘에 따른 모든 정점을 찾을 수 있는지, 그래프 상황에 따라 어떠한 장단점이 있는지, 간선이 많아질수록 인덱스 순환탐색 알고리즘보다 탐색시간이 얼마나 차이를 보이는지를 확인하였다. 웹 구조처럼 일정한 크기의 웹페이지와 많은 수의 링크가 존재하는 그래프에서 정점제거 순환탐색 알고리즘이 순환을 찾는데 적합하다는 것을 입증했다.

Keywords

순환(Cycle), 순환경로(Cycle Path), 중복순환(Repetition Cycle)

1. 서론

정보의 양이 기하급수적으로 증가함에 따라 사용자가 원하는 정확한 자료를 검색할 수 있게 하기 위해, 웹페이지를 구조화하여 자료를 저장, 관리하게 되었다. 웹페이지를 구조화하여 저장하기위해서는 적합한 자료구조 형태로 바꿔주어야 한다. 그러나 웹페이지들에는 수많은 링크로 인해 특정 웹페이지에서 출발하여 다시 돌아오는 링크가 많이 존재하게 된다. 이런 경우를 '순환'이라고 하는데, 이런 순환은 웹페이지를 구조화 하는데 지장을 초래한다. [1]

웹페이지의 구조를 분석하기 위해, 그 구조를 그래프로 표현하였다. 웹페이지는 실제 그래프의 형태와 매우 흡사하고, 그래프로 표현 시 탐색과 표현이 용이하다. 따라서 본 논문은 웹페이지를 구조화하는데 문제가 되는 순환을 발견하기 위하여, 웹페이지를 그래프의 형태로 표현하고, 그 그래프를 탐색하여 모든 순환을 찾아내고 순환의 경로를 찾는데 목적이 있다.

2. 웹페이지의 그래프 표현

웹페이지의 구조를 분석하기 위해서 그 구조를 방향그래프로 표현 할 수 있는데, 그것을 우리는 여기서 웹 그래프라고 정의한다. 웹 그래프에서 웹페이지는 정점이 되고, 웹페이지끼리 연결된 링크는 간선으로 표현한다. 웹페이지 구조를 그래프로 만들기 위해 인접리스트를 이용한다. 이때 웹페이지주소는 복잡하고 길기 때문에 일련의 인덱스를 부여하고, 그 인덱스를 이용해 인접리스트를 생성한다.

2.1. 용어정리

웹상에서 구조화 할 임의의 웹페이지가 주어졌을 때,

- 웹페이지의 집합을 P 로 정의한다.
- 웹페이지 P 에서 출발하는 처음 페이지를 $P_0 \in P$ 라고 하고, P_0 으로부터 출발하여 연결된 모든 페이지들의 집합을 수열로 $\{P_i\}$ ($i \in Z, i>=0$) 정의한다. 여기서 i는 웹페이지인덱스 라고 정의한다.
- $|P| = n$ ($n \in Z, n \neq 0$)라면, $P = \{P_0, ,...P_{n-1}\}$이다.

웹페이지의 집합 P를 웹 그래프로 표현 할 때,

- 웹 그래프 $G_p = (I_p, E_p)$로 정의한다.
- I_p는 G_p의 정점, 즉 웹페이지인덱스 i의 집합이다. I_p는 G_p의 모든 정점들의 집합이므로 $I(G_p) = \{0, ..., n-1\}$로 정의한다.
- E_p는 G_p의 간선, 즉 웹페이지에서 다른 웹페이지

를 연결한 링크이다. 임의의 두 웹페이지인덱스 i, j (i, $j \in I_p$)가 있을 때, i에서 출발하여 j로 가는 링크를 $<i, j> \in E_p$ 로 표현한다.

• E_p는 G_p의 모든 링크들의 집합이므로 $\forall <i, j> \in E_p$ 일 때, $E(G_p) = \{<i, j>\}$로 정의한다.

[그림 1]에 G_A 웹 그래프를 예를 들면, $I(G_A) = \{0, 1, 2, 3\}$; $E(G_A) = \{<0, 1>, <0, 2>, <1, 2>, <2, 3>, <3, 1>\}$처럼 나타낼 수 있다.

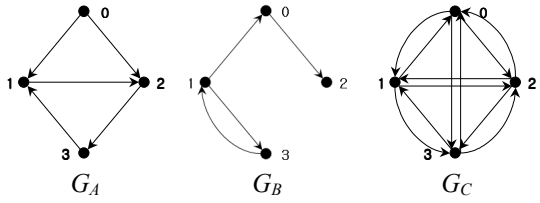


그림 1. 웹 그래프의 예

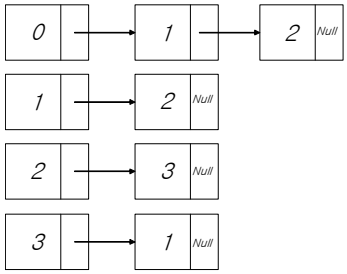


그림 2. [그림 1]의 웹 그래프 G_A를 인접리스트로 표현한 예

[그림 1]은 웹 그래프 예이다. 각 정점 위에는 각 웹페이지에 해당하는 인덱스 값을 쓰고, 각 페이지에서 다른 페이지로 가리키는 링크를 간선으로 표시하였다. [그림 2]는 [그림 1]의 웹 그래프 G_A를 인접리스트로 표현한 것이다. 그래프는 여러 가지 방법으로 표현될 수 있으나, 후에 제시 할 정점제거 알고리즘에서 정점들을 제거하기 용이하기 위해서 인접리스트를 사용하였다. 인접리스트에서 각 정점은 노드로 정의 되는데, 노드는 값과 다음 노드를 가리키는 포인터로 구성되어있다. 여기서 값은 웹페이지의 인덱스 값이 되고, 포인터는 그래프의 다음 정점을 가리키는 간선이 된다. 인접리스트의 첫 번째 노드들은 헤드노드로서 모든 웹페이지인덱스들의 리스트가 되고, 헤드노드의 포인터가 가리키고 있는 노드들은 그 정점에서 연결된 정점들 이다. 예를 들어, [그림 2]에서 보면 인덱스 값을 0을 갖고 있는 정점에서 연결된 정점들은 1, 2이다. 헤드노드에 연결된 노드들의 순서는 무관하다.

3. 그래프의 탐색과 순환

위에서 웹페이지를 그래프로 표현하였다. 그 웹 그래프에서 순환을 찾기 위해서 특정 정점으로부터 탐색을 시작하여, 최소한 모든 정점들을 한번은 방문해야만 한다. 깊이 우선 탐색(Depth First Search)과 너비 우선 탐색(Breadth First Search)은 모두 그래프의 모든 정점을 탐색하는 방법이다. BFS는 네트워크 구조와 같은 무방향 그래프에서 빠른 속도로 순환을 찾을 수 있는 장점이 있다. 하지만, 웹페이지의 구조를 분석하기 위하여 BFS를 이용 하는 데는 문제가 있다. 순환이란 특정 정점에서 출발하여 특정 정점으로 다시 돌아오는 것인데 BFS 자체가 출발정점을 계속해서 바꿔서 그래프를 탐색해 나가기 때문에 새로운 정점을 방문 할 때마다 지나온 정점들과 일일이 비교해야만 순환이 발생했는지 알 수 있다. 반면 DFS는 특정한 정점을 출발하여 모든 경로를 끝까지 탐색하므로 순환을 찾는데 적합 할뿐만 아니라 모든 순환을 찾을 수 있음을 보장 한다. [2] 따라서 본 논문에서는 DFS에 기반 한 여러 가지 순환탐색 방식에 대해 살펴보고자 한다.

3.1. 순환(Cycle), 순환경로(Cycle Path)

웹 그래프 $G_p = (I_p, E_p)$ 탐색 시, 순환(Cycle)이란, 아래와 같은 조건을 만족하는 웹페이지인덱스 $(i_1, \ldots, i_k)$ for $k \le n$ 들의 수열이다.

웹 그래프에서 순환을 C 라 하고, 웹 그래프 G_p에 있는 모든 순환의 집합을 $C(G_p)$ 했을 때, i_x, $i_y \in I_p$ ($1 \le x, y \le k, k \ge 2$), $E_p' = \{<i_1, i_x>, <i_x, i_y>, <i_y, i_k>\}$ ($E_p' \subseteq E_p$,) 일 때, E_p' 에서 $i_1 = i_k$ 이면, $C_1 = i_1, i_x, i_y, i_k$ ($C_1 \subseteq I(G_p)$, $C_1 \subseteq C(G_p)$) 이다. 또 $i_1 \rightarrow i_x \rightarrow i_y \rightarrow i_k$ 는 순환경로(Cycle Path)이다.

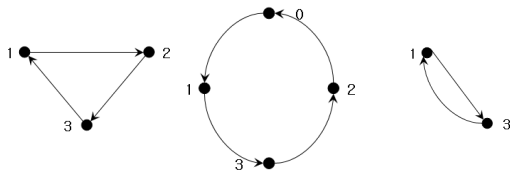


그림 3. [그림 1]에서 순환발생의 예

[그림 3]에서 순환경로는 왼쪽에서부터 각각 1→2→3→1, 0→1→2→3→0, 1→3→1 이다.

3.2. 중복순환(Repetition Cycle)

웹 그래프 G_p = (I_p, E_p) 탐색 시, 중복순환(Repetition Cycle)이란, 아래와 같은 조건을 만족하는 이미 발견한 같은 순환을 중복하여 발견하는 것이다.

순환 C 의 수열의 개수를 $|C| = k$ 라고 정의할 때, $C_1 = i_1, i_2..., i_{k-1}, i_k$ ($C_1 \subseteq C(G_p)$, $|C_1| = k$) 이고, $C_2 = i_2, i_3..., i_k, i_1$ ($C_2 \subseteq C(G_p)$) 일 때, $|C_2|$ = k 이면, C' 는 C와 같은 순환이다.

[그림 4]와 같이 1→2→3→1의 순환과 2→3→1→2의 순환은 총 순환의 수열에 수가 서로 같고, 각 정점끼리 연결된 순서도 같기 때문에, 그래프 탐색 시 두 순환을 모두 발견한다면 두 순환은 중복순환이다.

4. 순환경로탐색 알고리즘

4.1. 방문기록 순환탐색 알고리즘

방문기록 순환탐색 알고리즘은 웹 그래프를 탐색 할 때 깊이우선탐색 방법으로 그래프를 순회하면서 방문 했던 정점을 방문하게 되었을 때 방문했음을 기록하고, 이미 방문했던 정점을 방문했을 때 순환을 발견하는 원리를 이용한 순환탐색 알고리즘이다. 임의의 최초 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, 스택과 비슷한 구조로 자료가 삽입되고 삭제되므로 스택형태의 자료구조를 사용하였다. 순환탐색 시 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. 깊이우선탐색을 기반으로 탐색하기 때문에 그래프를 인접리스트로 표현하면 정점의 개수가 V 개 이고, 간선의 개수가 E 개 일 때, $O(|V| + |E|)$의 복잡도를 보인다.

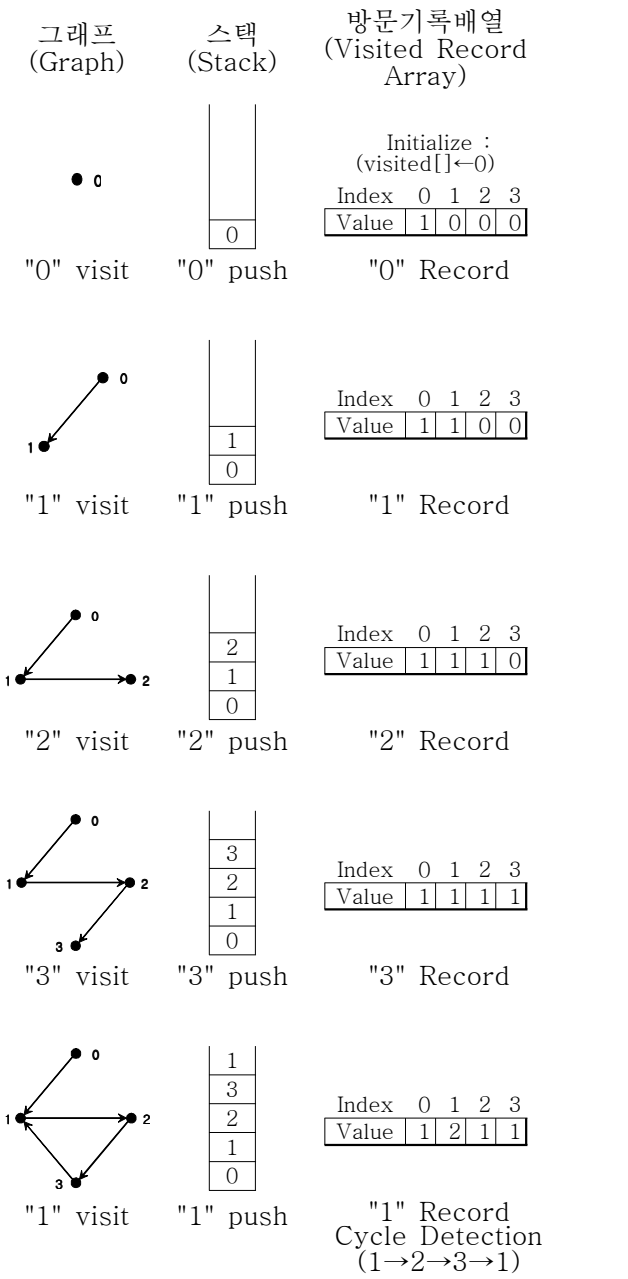


그림 5. [그림 1]의 웹 그래프 G_A 에서 방문기록 순환탐색 알고리즘을 이용한 순환을 탐색한 경우의 예

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 방문여부

배열이 같게 되기 때문에 결국 모든 정점을 탐색하고 순환을 찾을 수 있는 것이다.

```
void Visited_Record_DFS(int v)
        // visited[] ← ∅
        // stack[] ←∅
        visited[v] ← 1; //정점v 방문
        for(each vertex w adjacent from v)
        {
            if(visit[w] is ∅)
            {
                push(w);
                Visited_Record_DFS(w);
            }
            else
            {
                /* Cycle Detection */
                /* Output Cycle Path */
            }
        }
        temp←Pop();
        visited[temp]←∅;
```

알고리즘 1. 방문기록 순환탐색 알고리즘

이 알고리즘은 깊이우선탐색을 하기 때문에 모든 순환을 발견하지만, 같은 순환을 중복해서 찾아낸다는 단점이 있다. [그림 6]은 정점 0에서 2로 출발하여 방문기록 순환탐색 시 [그림 5]와 중복된 순환을 발견하는 예이다.

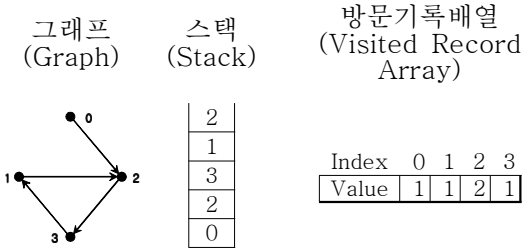


그림 6. 방문기록 순환탐색 알고리즘에서 중복된 순환을 발견하는 예

[그림 5]에서 발견한 순환 1→2→4→1과 [그림 6]에서 발견한 순환 2→4→1→2은 중복된 순환이다.

4.1.1 순환경로 탐색 알고리즘

순환탐색을 하다가 순환을 발견했을 때, 순환경로를 출력해야한다. 지금까지 스택에 저장한 경로들은 순환이 일어난 경로뿐 아니라 정점이 방문하는 모든 경로를 저장하고 있기 때문에, 순환이 발생한 정점부터 다시 같은 정점으로 돌아오기 까지 순환경로를 어떻게 출력하는가가 문제이다. 결국 스택에 저장한 방문한 경로를 스택 위에서부터 아래로 하나씩 비교하면서 순환이 발생한 정점과 같은 값을 가지고 있는 스택의 위치를 알아낸 뒤 그 위치부터 스택의 끝까지 출력해 줘야만 한다. 그렇게 순환경로를 계산한 경우는 정점수가 V개 있을 때, 최악의 경우 $O(V)$의 복잡도를 가지게 된다. 또한 매 순환발생시 이러한 계산과정을 거쳐야 하기 때문에 실제 방문기록 순환탐색 알고리즘에 복잡도는 정점의 수만큼 곱해지게 되서 $O(|V|^2 + |E|\cdot|V|)$가 된다.

이렇게 복잡한 계산과정을 거치지 않게 하기 위하여, 탐색을 하면서 방문기록배열(Visited Record Array)에 방문여부를 기록 할 때, 정점이 저장된 스택에 인덱스를 저장함으로써 다시 방문을 하는지 확인 할 수 있을 뿐 아니라(0이 아닌 경우는 이미 방문한 정점이다) 같은 정점에 방문했을 때, 해당하는 인덱스의 값으로부터 스택의 끝까지 출력하면 어떠한 경우에도 모든 노드에 대해 $O(1)$의 복잡도로 순환경로를 출력할 수 있다.

위에 [그림 6]의 예로 나타내면,

	0	1	2	3	4
stack[]	0	2	3	1	2

그림 7. [그림 6]에서 스택구조

	index	0	1	2	3
visited[]	value (stack index)	0	3	1	2

그림 8. [그림 6]에서 값을 스택의 인덱스로 저장한 예(방문기록배열에 값에 스택의 인덱스를 저장한다.)

[그림 6]에서 정점 “1”에서 정점 “2”로 방문하면서 정점 “2”를 방문기록하려고 하면, 이미 방문한 것으로 기록되어있다. 이미 방문한 정점이므로 방문기록배열의 인덱스 “2”에 값을 보면 스택의 인덱스가 표기되어 있으므로 스택에 저장해둔 경로를 저장된 인덱스로부터 스택의 끝까지 출력하면 스택의 경로를 한 번에 출력할 수 있게 된다. 순환경로는 2→4→1→2 가 출력하게 된다.

4.1.2 모든 정점 탐색 알고리즘

방문기록 순환탐색 알고리즘은 모든 순환을 탐색할 수는 있지만 최초 정점이 가리키고 있지 않는 정점은 탐색에서 제외되기 때문에 모든 순환을 탐색할 수 없다. 따라서 방문하지 않은 정점이 있다면

다시 그 정점을 시작점으로 순환탐색을 해야만 모든 정점에 대한 순환을 찾을 수 있다.

본 알고리즘은 탐색을 마친 후 방문하지 않은 정점이 있는지 확인하고, 만약 존재한다면 그 정점으로부터 다시 방문기록 순환탐색해서 모든 순환을 찾아야 하기 때문에 최악의 경우 $O(|V|^2 + |E|\cdot|V|)$의 복잡도를 보인다. 또한, 이 방법은 최초에 방문기록 순환탐색에서 발견한 순환까지 중복으로 발견하기 때문에 더 많은 중복을 발생하는 단점이 있다.

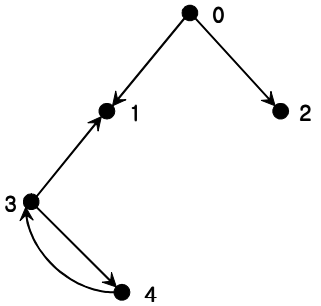


그림 9. 정점 “0”에서 출발하여 모든 순환을 발견하지 못하는 예

4.2. 정점제거 순환탐색 알고리즘

방문기록 순환탐색 알고리즘과 기본조건은 같다. 하지만, 정점제거 순환탐색 알고리즘은 모든 정점에서 순환을 탐색하고, 순환은 각 출발한 정점으로 시작하는 순환만을 탐색한다. 이 알고리즘은 한 번 방문한 정점을 제거함으로써 중복된 순환을 발견하지 않는다. 탐색 시 탐색을 시작하는 정점과 탐색을 오직 시작점으로부터 나오는 순환만을 탐색하기 위한 인자를 초기값으로 받는다. 그 시작 정점으로 모든 순환을 발견하고 돌아오면 그 정점을 지우고, 다음 정점으로부터 다시 순환을 탐색한다.

```
void CutVertex_DFS(int v, int s)
        // visited[] ← ∅
        // stack[] ←∅
        visited[v] ← 1;
        for(each vertex w adjacent from v)
        {
                if(u ≠ s)
                {
                        if(visited[w] == 1)
                            continue;
                        push(w);
                        CutVertex_DFS(w, s);
                }
                else
                {
                        /* Cycle Detection */
                        /* Output Cycle Path */
                }
        }
        temp←Pop();
        visited[temp]←∅;
```

알고리즘 2. 정점제거 순환탐색 알고리즘

[그림 1]에서 웹 그래프 G_A를 정점 “0”에서부터 출발해서 순환을 탐색하면, [그림 5]와 [그림 6]과 같이 중간에 순환이 발생하기는 하지만, 정점 “0”을 포함하는 순환이 발생하지 않으므로 순환을 탐색할 수 없다. 정점 “0”으로 들어오는 간선이 없기 때문이다. 이렇게 정점 “0”으로 부터의 순환을 모두 마치면 인접리스트의 헤드노드에서 삭제한다.

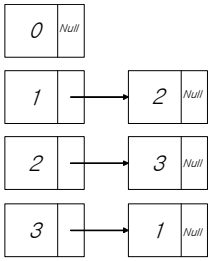


그림 10. [그림 2]에 인접리스트에서 헤드노드 “0”에 연결된 노드를 삭제하는 예

인접리스트에서 헤드노드 “0”에 연결된 노드들을 삭제한 뒤, 다시 정점 “1”로부터 탐색을 시작한다. [그림 11]에서 발견한 것과 같은 순환이 발견될 것이다. 또 다시 인접리스트에서 헤드노드 “1”에 연결된 노드들을 삭제한다. 이런 방식으로 정점 “3”까지 반복해서 모든 정점으로부터 시작하는 순환탐색을 마치면 순환탐색은 종료된다.

그래프 (Graph) 스택 (Stack) 방문기록배열 (Visited Record Array)

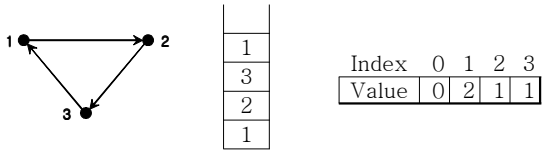


그림 11. 정점 "0"을 삭제한 뒤 정점 "1"에서 출발하여 순환을 발견한 예

이런 방법으로 순환을 탐색하면, 모든 순환을 찾을 수 있을 뿐 아니라, 탐색을 마친 정점은 삭제하므로 중복된 순환은 발견하지 않는다. 또한 순환경로를 찾는데 있어서도 출발정점으로 부터의 순환만을 찾기 때문에 순환발생시 스택에 저장되어있는 순서대로 출력하면 순환경로를 얻을 수 있다. 모든 정점에서 순환을 탐색하므로 최악의 경우 $O(|V|^2 + |E| \cdot |V|)$의 복잡도를 보인다.

[알고리즘 1]과 [알고리즘 2]가 다른 점은 [알고리즘 1]은 단지 방분기록배열만 확인해서(0이 아닌 경우) 순환을 발견하는 반면 [알고리즘 2]는 시작정점으로 부터의 순환만을 탐색하게 되므로 시작정점 순환탐색을 위한 인자가 하나가 더 필요하다는 점과, 시작정점과 같은지 비교하는 과정이 필요하다는 점이 다르다.

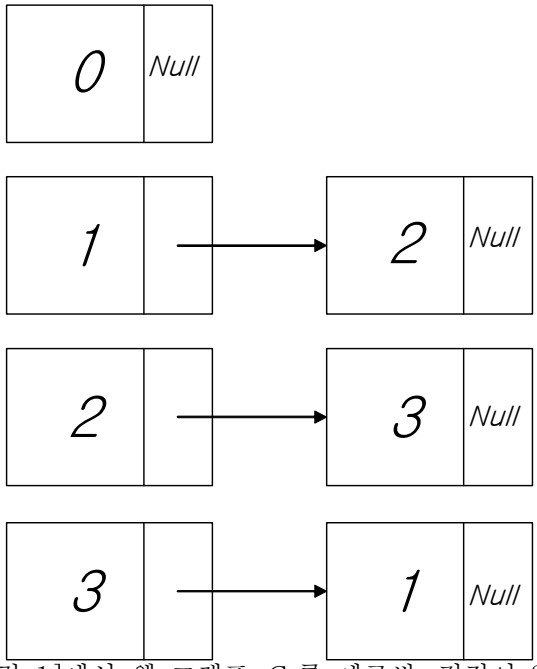
5. 실험

실험은 모든 순환을 찾을 수 있는지, 또 중복순환이 발견되는지 여부와, 알고리즘에 따른 성능 비교 실험으로 구분하였다.

5.1 완전그래프를 이용한 순환 탐색 실험

실험환경은 프로그램은 Microsoft Visual Studio. Net, 코딩은 C#으로 작성하였다. 위 알고리즘을 이용해서 모든 순환을 탐색하는지 여부를 알아내기 위해, 완전그래프를 이용해 모든 순환을 찾을 수 있는지, 또 중복순환을 발견하는지 실험하였다. 여기서 완전그래프란 [그림 1]에서 G_C와 같이, 각 정점에서 최대 차수만큼 간선이 존재하는 그래프를 완전그래프라 한다.

완전그래프를 정점의 개수를 늘려가면서 실험하여, 나오는 순환개수가 실제 순환개수와 일치하는지 실험한다. 각 정점수에 따른 순환개수를 구하고 순환개수를 모두 더하면 총 순환개수를 구할 수 있다.



[그림 1]에서 웹 그래프 G_C를 예로써, 정점이 2개일 때 순환 개수는 0 → 1 → 0등 6개 이며, 정점이 3개일 때는 1 → 2 → 3 → 1 등 순환 개수는 8개 이다. 또한 정점이 4개일 때 순환 개수는 0 → 3 → 2 → 1 → 0 등 6개 이다.

실험한 결과 방문기록 순환탐색 알고리즘은 33개, 정점제거 순환탐색 알고리즘은 20개의 순환을 발견했다. 정점제거 순환탐색 알고리즘은 정확히 모든 순환을 찾아냈고, 순환탐색 알고리즘은 중복된 순환이 13개가 포함되어 있었다.

5.2 알고리즘 비교 실험

실험은 정점이 30개, 간선수 100개인 임의의 그래프를 생성한 후, 간선수를 100개에서부터 10개씩 증가시켜가며 실험하였다.

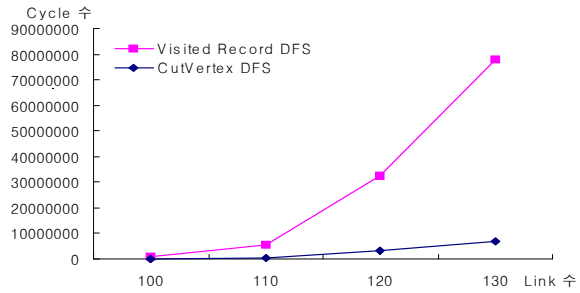


그림 12. 알고리즘에 따른 순환탐색 수

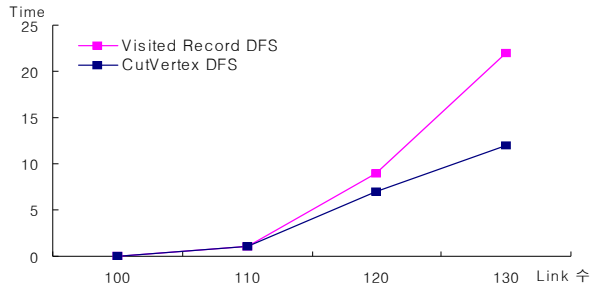


그림 13. 알고리즘에 따른 순환탐색시간

[그림 12]는 정점에 따른 순환개수 비율로서 간선이 많아질수록 방문기록 순환탐색알고리즘(VisitedRecord DFS)이 급격하게 많은 순환개수를 발견하는 것을 알 수 있다. 그림 13]는 정점에 따른 순환탐색시간 비율로서 방문기록 순환탐색이 정점제거 순환탐색 알고리즘(Cutvertex DFS)보다 급격하게 탐색시간이 증가하는 것을 알 수 있다. 정해진 정점에서 간선이 증가함에 따라 기하급수적으로 순환이 많이 발생하게 되는, 방문기록 순환탐색 알고리즘은 중복된 순환까지 발견하게 되므로 간선이 증가함에 따라 순환개수가 크게 증가하는 것을 볼 수가 있다. 또한 정점제거 순환탐색 알고리즘은 한 정점에서 아무리 많은 간선이 생성되더라도 한번 탐색이 끝나면 삭제하기 때문에 순환개수가 완만하게 증가하는 것을 볼 수 있다. 또 [그림 13]에서 보면 어느 정도 간선에서는 DFS에서 중복된 순환을 찾는 시간과 DFS에서 모든 정점을 찾고 제거하는 시간이 비슷하게 비례하다가 간선이 많아지고 중복된 순환이 많이 발생함에 따라 탐색시간도 많이 차이가 나게 되는 것을 알 수 있다.

6. 결론

웹페이지의 구조화가 중요하게 된 시점에서, 웹페이지를 구조화하기 위해 방향그래프를 이용해 순환을 탐색하고 그 경로를 발견하는 알고리즘을 제시하였다. 위 실험과 같이 방문기록 순환탐색 알고리즘은 특정 웹페이지를 기준으로 링크들이 뻗어 나가는 구조의 웹페이지를 구조화 할 때는 유용하다. 하지만, 중복된 순환이 많이 발견됨으로 중복을 제거하려면, 발견된 순환경로를 일일이 비교해서 같은 경로를 제거해야만 하기 때문에 많은 연산량을 요구한다. 최악의 경우 복잡도를 비교했을 때, 방문기록 순환탐색 알고리즘과 정점제거 순환탐색 알고리즘은 각각, $O(|V| + |E|)$과 $O(|V|^2 + |E|\cdot|V|)$로 방문기록 순환탐색 알고리즘이 정점제거 알고리즘 보다 좋지만, 실제로 방문기록 순환탐색 알고리즘은 중복된 순환을 제거하기 위해서, 순환들이 중복된 순환인지 확인해야 한다. 즉 순환의 집합들의 개수 $|C(G_p)|^2$만큼 비교 연산이 필요하다. 그렇기 때문에 정점제거 순환탐색 알고리즘은 간선이 많아질수록 한 정점에 연결된 간선이 많아지기 때문에 정점이 삭제되면서 탐색이 좋아지는 반면, 방문기록 순환탐색 알고리즘은 간선이 많아지면 많아질수록 중복된 순환을 많이 발견하게 되어 오히려 탐색시간이 큰 폭으로 증가하게 된다.

정점제거 순환탐색 알고리즘은 각 정점에서 출발하여 탐색을 마친 후 그 정점을 그래프에서 제거하기 때문에 중복된 순환은 발견되지 않는다. 또한 제거된 정점으로는 방문하지 않기 때문에, 한 정점만 삭제하여도 많은 간선을 삭제할 수 있다. 따라서 정점제거 순환탐색 알고리즘은 실제 웹 페이지구조처럼 페이지는 적고 링크가 많은 구조에서 웹페이지를 구조화하는데 적합하다.

7. 참고문헌

[1] A. Itai, R. Lipton, C. Papadimitriou, M. Rodeh, "Covering Graphs by Simple Circuits," SIAM Journal on Computing, 10(4): (1981)746-750

[2] A. Itai, C. Papadimitriou, J. Szwarcfiter, "Hamilton Paths in Grid Graphs," SIAM Journal on Computing. 11(4): (1982)676-686

[3] G. Nivasch, "Cycle detection using a stack," Information Processing Letters, 90(3) (2004) 135-140

[4] S. Sahni, E. Horowitz, Fundamental of computer algorithm, 1985

[5] J. Szwarcfiter, G. Navarro, R. Baeza-Yates, J. Oliveira, W. Cunto, N. Ziviani, “Optimal binary search trees with costs depending on the access paths,” Theoretical Computer Science, Vol 290(3): (2003)1799-1814

[6] J. Szwarcfiter, G. Chaty, "Enumerating the Kernels of a Directed Graph with no Odd Circuits," Information Processing Letters, 51(3): (1994)149-153