Algorithms

HW#3 Report

2013-11706
기계항공공학부 강인재

[Strongly Connected Component]

strongly connected component란 주어진 그래프에서 서로에게 도달할 수 있는 경로가 존재하는 정점들의 집합을 의미한다. 즉, strongly connected component 내의 임의의 두 정점 사이에는 서로에게 도달할 수 있는 경로가 항상 존재한다. 이번 과제에서는 그래프가 주어졌을 때 strongly connected component들을 구하고, 그래프를 표현하는 여러 자료구조에 따른 소요 시간을 알아보고자 하였다.

strongly connected component를 구하는 알고리즘은 다음과 같다.

```
SCC(G)
{
    그래프 G에 대해 DFS(G)를 수행해 각 정점의 완료 시간을 계산한다;
    G의 모든 간선이 반대로 구성된 새로운 그래프 G^R를 만든다;
    앞서 계산한 완료 시간의 역순으로 DFS(G^R)을 수행한다;
    DFS(G^R)을 수행하는 과정에서 분리된 부분 그래프들을 리턴한다;
}
```

[구현 내용]

입력받은 그래프의 형식으로부터 정점의 수를 받아 2차원 배열을 동적 할당하고, 배열의 원소들에 간선의 정보를 대입하였다. 즉, 인접 행렬을 구성하였다. 그리고 이 인접 행렬로부터 전치된 인접 행렬도 구성하였다.

- adjMat()

앞서 구성한 인접 행렬 및 전치 행렬을 가지고 SCC 알고리즘을 구현하고, 시간을 측정하였다.

- adjList()

먼저 int형 변수 elem과 struct node*형 next를 갖는 Node 구조체를 정의하였다. 그리고 Node*형 변수를 원소로 갖는 헤더 배열을 만들어, 각 원소가 각 정점이 갖는 첫 번째 노드를 가리키도록 하였다. 인접 리스트는 앞서 구성한 인접 행렬 및 전치 행렬을 가지고 구성하였다.

- adjArr()

인접 행렬을 구성할 때 각 정점이 갖는 간선의 개수의 합을 계산하기 위한 별도의 변수를 정의하여 adjArr() 함수 내에서 배열을 동적 할당하였다. 헤더를 구성하기 위해 int형 변수 elem과 int*형 변수 end를 갖는 HeadElem 구조체를 정의하였다. 그리고 HeadElem형 변수를 원소로 갖는 헤더 배열을 구성하여 각 원소의 elem은 첫 번째 정점부터 해당 정점까지의 총 간선의 개수를, end는 인접 배열에서의 각 정점의 첫 번째 값이 위치한 주소값이 되도록 하였다.

각 함수 내의 strongly connected component를 계산하는 부분에 대한 소요 시간을 clock 함수를 이용하여 계산하였다. 또한 크기 $(|V|+1)^2$의 2차원 배열을 만들어 SCC 알고리즘의 결과를 저장하였다. 각 정점의 마지막 간선 다음 원소에는 -1을 저장하고, 모든 정점이 포함되었을 경우 그 다음 1차원 배열의 첫 번째 원소에도 -1을 저장함으로써 정렬 알고리즘을 위한 범위를 확정하였다. 그리고 각 정점의 간선에 대해 원소를 정렬하고, 각 정점의 첫 번째 간선을 비교하여 배열을 정렬하여 lexicographic order가 되도록 하였다. 정렬은 병합정렬을 사용하였다.

[결과 분석]

1. 밀도

정점의 개수를 1000으로 고정하고, 간선의 개수를 변화시키면서 각 자료구조에 대한 소요 시간을 측정하였다. 간선과 연결된 두 정점은 랜덤하게 선택되도록 테스트 입력을 구성하였다. 간선의 개수는 100개부터 500000개까지 변화시켰다. 단위는 ms(밀리초)이다.

$\|E\|$	100	500	1000	5000	10000	50000	100000	500000
행렬	6	7	7	7	7	9	9	20
리스트	9	11	14	43	76	427	821	2686
배열	15	17	19	39	82	305	542	1960

이 결과로부터 각 자료구조에 따른 차이가 확실히 나타난다. 인접 행렬의 경우 간선의 개수에 따른 소요 시간의 차이가 그리 크지 않은 반면, 인접 리스트의 경우 간선의 개수가 많아짐에 따라 시간이 가장 오래 걸렸다. 인접 행렬의 경우 간선 존재 여부를 확인하는 시간이 상수인 데 비해, 인접 리스트는 최악의 경우 (특정 간선의) 탐색 시간이 정점의 개수에 비례한다. 인접 배열의 경우 인접 행렬과 인접 리스트의 중간 정도의 시간이 소요되었다. 인접 배열의 소요 시간은 위 결과보다 더욱 개선할 수 있을 것이다. 본 알고리즘에서는 각 정점에 대한 나

가는 간선이 정렬되어 있지 않은 경우도 가정하여, 인접 배열을 선형적으로 탐색하도록 구현하였다. 하지만 입력이 정렬되어 있다고 가정한다면, (또는 입력을 정렬하는 과정이 추가된다면) 이진 검색을 통해 탐색 시간을 줄일 수 있다. 또는 해시 테이블을 이용하는 방법도 가능하다.

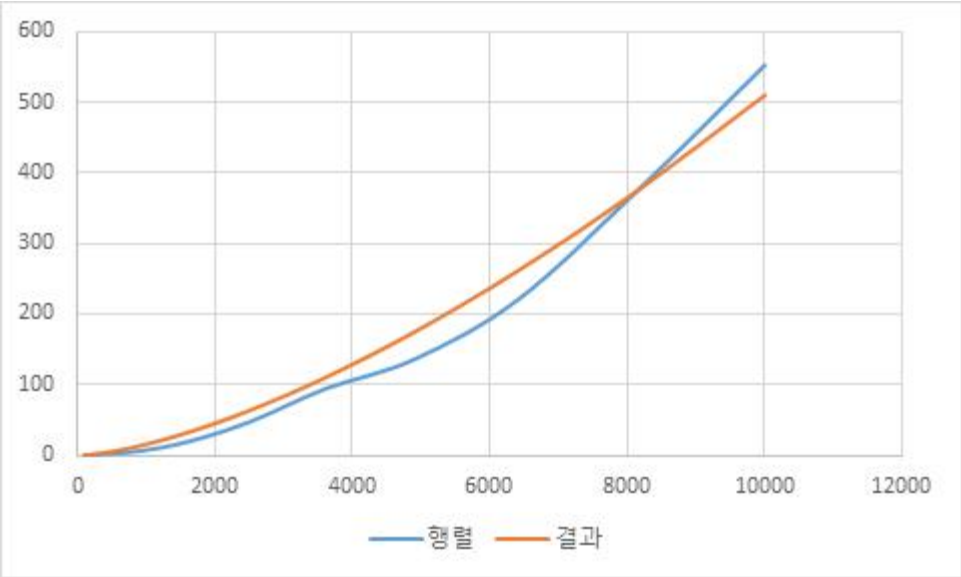
본 알고리즘에서는 자료구조를 구성하는 시간은 고려하지 않았으며, 구성된 자료구조를 가지고 strongly connected component를 찾는 시간만을 계산하였다. 위의 결과만 놓고 보면 인접 행렬을 사용하는 것이 좋은 것처럼 보인다. 그러나 인접 행렬을 구성하는 시간을 고려한다면, sparse한 그래프의 경우 인접 리스트나 인접 배열을 사용하는 것이 적절하다. 왜냐하면 행렬을 구성하는 데에만 $\Theta(|V|^2)$의 시간이 소모되며, 모든 간선의 정보를 담게 되므로 공간도 상당 부분 차지하기 때문이다.

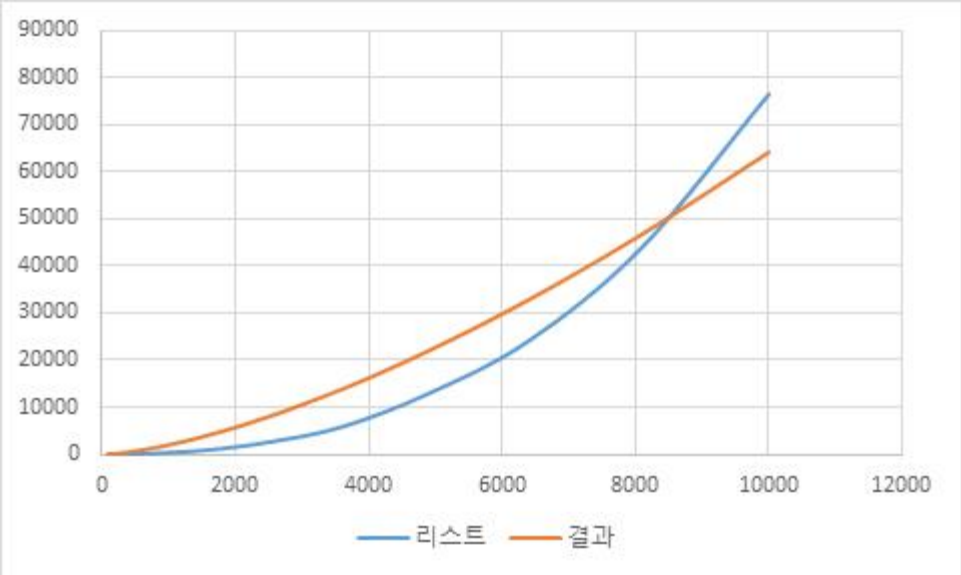
2. 정점의 개수

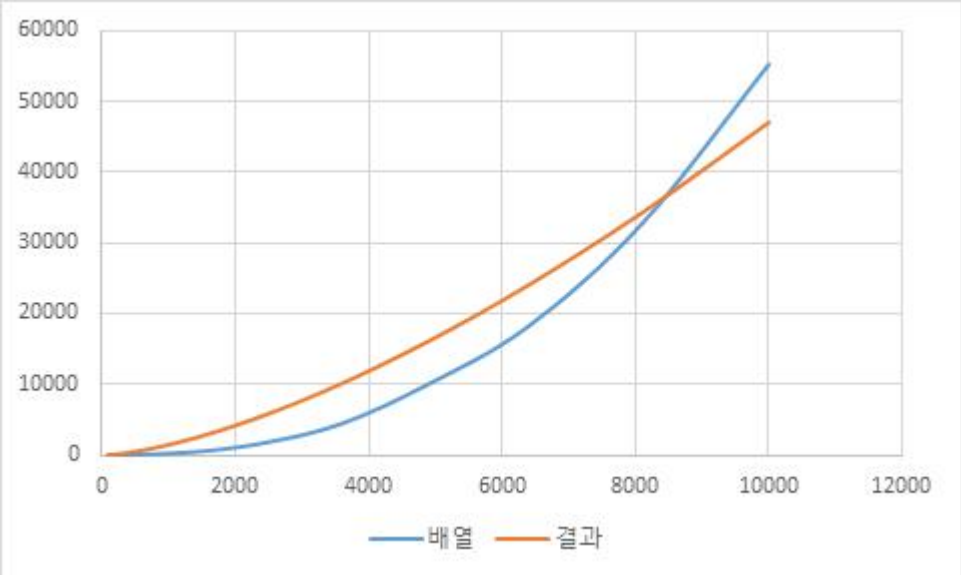
$|E|=|V|$인 경우 그래프가 '겨우' 연결될 정도이므로 극히 sparse하며, $|E|=|V|^2$의 경우 모든 간선이 존재하므로 극히 dense하다. 따라서 1과 2의 평균을 지수로 취하여 $|E|=|V|^{1.5}$의 관계로 $|V|$의 값을 최소 100부터 최대 10000까지 변화시키며 각 자료구조에 대한 소요 시간을 측정하였다. 단위는 ms(밀리초)이다.

$\|V\|$	100	400	900	1600	2500	3600	4900	6400	8100	10000
행렬	0	2	6	19	47	94	136	220	371	553
리스트	2	32	262	898	2540	5779	12862	24026	43922	76382
배열	1	28	171	619	1837	4446	10043	18238	32770	55300

다음은 $y=ax^{1.5}$의 관계식으로 추세선을 그려 본 결과이다. (왜냐하면 이론으로부터 $\Theta(|V|+|E|)=\Theta(|V|+|V|^{1.5})=\Theta(|V|^{1.5})$이다.) 추세선을 참값으로 상정하고 결과 값들의 분산을 최소로 하는 a를 구하였다.







세 자료구조 모두 큰 차이를 보이지는 않았으나, 파란색 선(실제 소요시간)은 결과(추세선)와 달리 2차 회귀곡선에 좀 더 가까웠다. 따라서 알고리즘을 구현하는 과정에서 정점의 개수의 제곱에 비례하는 오버헤드가 있었음을 알 수 있으며, 이를 개선한다면 더욱 빠른 SCC 알고리즘을 얻을 수 있을 것이다.

[트러블 슈팅]

1. 자료구조의 차이로 인해 DFS와 aDFS 함수의 경우 세 가지 자료구조에 대해 구현해야 했다. 또한 원래의 그래프와 전치 그래프 간에 필요한 알고리즘이 달라서, 전치 그래프에 대

한 DFS와 aDFS까지 총 12개의 함수를 구현하였다. 이 과정에서 함수 및 변수의 이름을 헷갈려, 전치된 그래프를 가지고 해야 할 절차를 원래 그래프를 가지고 하는 등의 실수가 있었다. 결국 그래프를 일일이 그려 가며 과정을 따라가 디버깅을 할 수 있었다.

2. 결과값을 저장하는 2차원 배열의 마지막 (1차원) 배열에 -1을 저장하는 것을 제대로 구현하지 않아 이를 검사하는 과정에서 무한루프가 발생하였다. 디버깅할 때 인덱스의 값이 $|V|$ 값을 벗어난 것을 보고 문제를 해결하였다.

[리눅스 환경에서 컴파일 방법]

```
SCC (실행 파일)
$ gcc -o SCC main.c adjMat.c adjList.c adjArr.c sort.c header.h
```

[리눅스 환경에서 실행 방법]

```
SCC 실행
$ ./SCC

입력 파일이 있을 경우
$ cat input | ./SCC
(input은 주어진 입력의 이름임.)

시간만 확인하고 싶은 경우
$ cat input | ./SCC | grep 'time'
```

[실행의 예]

SCC 프로그램을 실행하면 다음과 같이 strongly connected components와 계산 시간을 인접 행렬, 인접 리스트, 인접 배열의 순서로 콘솔에 출력한다.
(이 경우 콘솔에 직접 그래프를 입력한 것이며, 입력 파일이 있을 경우 실행 방법을 참고하면 된다.)

```
abcinje@martini:~/al3$ ./SCC
10
3 2 7 10
2 3 6
1 4
1 5
1 3
2 3 7
2 2 8
1 9
1 6
1 8
1
2 6 7 8 9
3 4 5
10
*** Adjacency Matrix Result ***
Program running time : 0ms

1
2 6 7 8 9
3 4 5
10
*** Adjacency List Result ***
Program running time : 0ms

1
2 6 7 8 9
3 4 5
10
*** Adjacency Array Result ***
Program running time : 0ms
```