Algorithm #2 Homework

2019-18499 김준혁

실행 결과 및 실행 시간

1. n 크기에 따른 실행 시간 (n:m = 1:10)

(1) size : n = 10, m = 100 (n : m 의 비율을 1 : 10으로 동일시 하기 위해 중복도 포함)

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 10, m = 100, microseconds : 6us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 10, m = 100, microseconds : 6us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 10, m = 100, microseconds : 4us
```

(2) size : n = 100, m = 1000

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 100, m = 1000, microseconds : 52us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 100, m = 1000, microseconds : 23us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 100, m = 1000, microseconds : 21us
```

(3) size : n = 1000, m = 10000

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 1000, m = 10000, microseconds : 1974us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 1000, m = 10000, microseconds : 267us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 1000, m = 10000, microseconds : 162us
```

(4) size : n = 5000, m = 50000

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 5000, m = 50000, microseconds : 40004us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 5000, m = 50000, microseconds : 1879us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 5000, m = 50000, microseconds : 844us
```

2. m 크기(dense)에 따른 실행 시간 (n = 5000)

(1) size : n = 5000, m = 5000

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 5000, m = 5000, microseconds : 77556us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 5000, m = 5000, microseconds : 476us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 5000, m = 5000, microseconds : 437us
```

(2) size : n = 5000, m = 10000

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 5000, m = 10000, microseconds : 70265us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 5000, m = 10000, microseconds : 685us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 5000, m = 10000, microseconds : 618us
```

(3) size : n = 5000, m = 50000

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 5000, m = 50000, microseconds : 40004us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 5000, m = 50000, microseconds : 1879us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 5000, m = 50000, microseconds : 844us
```

(4) size : n = 5000, m = 100000

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 5000, m = 100000, microseconds : 34533us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 5000, m = 100000, microseconds : 7089us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 5000, m = 100000, microseconds : 938us
```

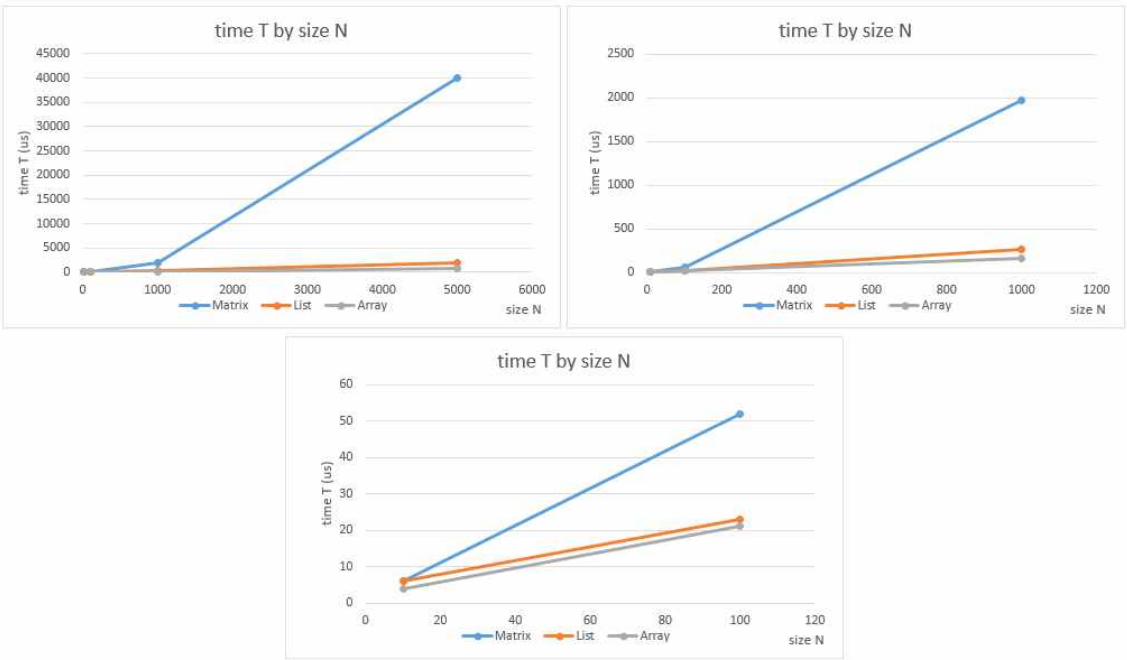
(5) size : n = 5000, m = 300000

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 1 ./1.in ./1.out
Input size : n = 5000, m = 300000, microseconds : 31475us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 2 ./1.in ./1.out
Input size : n = 5000, m = 300000, microseconds : 32586us
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW2_2019-18499_김준혁$ ./main 3 ./1.in ./1.out
Input size : n = 5000, m = 300000, microseconds : 1680us
```

각 알고리즘 시간복잡도 계수 구하기

1. n 크기에 따른 실행 시간 (n:m = 1:10)

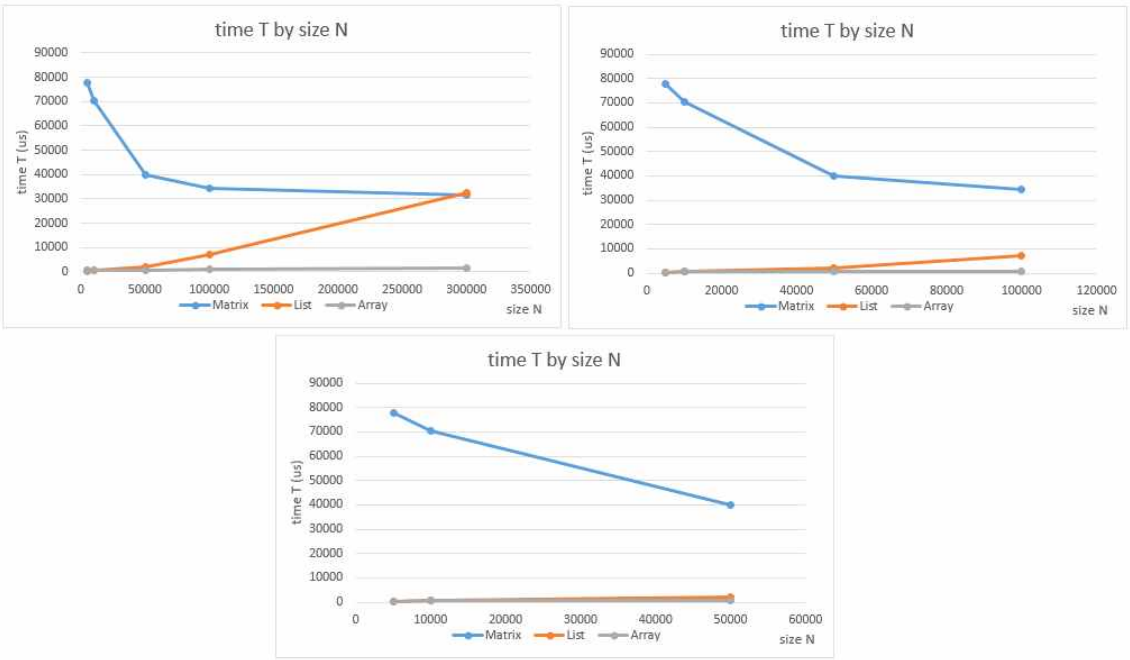
n=10일 때의 case를 통해 O(n^2)의 시간복잡도를 갖는 Matrix를 이용한 Strongly Connected Component 알고리즘의 그것을 $a_i n^2 + p_i m^2 + b_i n + q_i m + c_i$, O(n+m)의 시간복잡도를 갖는 Linked List, 또는 Array를 이용한 알고리즘의 그것을 $a_i n + b_i m + c_i$으로 표현할 때, $c_i = 5(\pm 1)(\mu s)$로 생각할 수 있을 것이다. 다만 이 식의 다른 계수를 직접 구하지는 않고, n과 m의 값이나 비율을 고정하여 n 또는 m의 변화에 따른 각각의 변화 정도를 위주로 관찰하였다. n을 10, 100, 1000, 5000으로 늘려가며 확인한 결과는 아래와 같다.



엑셀의 추세선을 이용하여 이 값을 확인하였을 때, Matrix, List, Array를 이용한 알고리즘은 각각 $0.0015n^2 + 0.4715n - 4.2177$, $0.3803n - 37.136$, $0.1685n + 0.4395$로 나타났다. 이것만 보았을 때 vertex 개수(size, n)에 따라서는, Array > List > Matrix 순으로 높은 시간 효율을 나타냈다.

2. m 크기(dense)에 따른 실행 시간 (n = 5000)

n=5000으로 고정하고 m을 5000, 10000, 50000, 100000, 300000으로 늘려가며 확인한 결과는 아래와 같다.



엑셀의 추세선을 이용하여 이 값을 확인하였을 때, Matrix, List, Array를 이용한 알고리즘은 각각 $0.000002m^2 - 0.6354m + 76600$, $0.1114m - 1817.9$, $0.0038m + 545.87$ 로 나타났다. Matrix의 경우, m이 커질수록 오히려 실행 시간이 감소하였다. edge가 늘어나 오히려 vertex 검색 횟수가 줄어드는 것이 영향을 미치지 않았나라는 추측을 해본다. 아니면 순전히 우연에 의한 결과일 수도 있다는 것 또한 생각해볼 수도 있겠다. 결과적으로 Matrix의 경우 m에 대한 식으로 표현하는 것은 오차가 있을 것이라 판단된다. List의 경우에는 m에 대한 계수가 매우 컸다. 그리하여 m=300000인 경우, Matrix를 이용한 알고리즘보다 실행 시간이 커질 정도였다. Array의 경우, m에 대한 계수가 매우 작아, 다른 두 자료 구조에 비해 현저히 작은 실행 시간을 기록하였다.

그리하여 m이 가질 수 있는 값인 O(n^2)에 근접할수록 Array > Matrix > List 순으로 높은 시간 효율을 가진다고 할 수 있겠다. 즉, 그래프의 edge 밀도가 높을수록 List의 시간 효율이 가장 치명적이었다.

Environment & How to run

```
dragonfish@DESKTOP-K6UG8NU:~/Algorithm/HW1_2019-18499_김준혁$ g++ --version
g++ (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

개인 구동 환경은 vscode에 WSL을 연결한 Ubuntu 18.04를 이용, g++의 버전은 캡처에 나와 있듯, 공지와 같은 컴파일러를 이용하여, 같은 명령어를 통해 컴파일, 실행(파일 위치에

맞게)하였다.