Automata #1 Homework

2019-18499 김준혁

실행 결과 (제공 testcase 이용)

```
root@4a2effdaf185:/home/hw1# sh grader.sh 2019-18499 testcase/Q1in.txt testcase/Q2in.txt
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
zip is already the newest version (3.0-12build2).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@4a2effdaf185:/home/hw1# cat testcase/Q1in.txt
0 0 0,1 - -
1 0 - 1,2 -
2 1 2 - -
root@4a2effdaf185:/home/hw1# cat outQ1.txt
0 0 1 2 -
1 0 1 3 -
2 0 2 2 -
3 1 4 3 -
4 1 4 2 -
root@4a2effdaf185:/home/hw1# cat testcase/Q2in.txt
7 100 01 0000011111111 010000 10 0101010101 000000
0 0 0,1 - -
1 0 - 1,2 -
2 1 2 - -
root@4a2effdaf185:/home/hw1# cat outQ2.txt
no
yes
yes
yes
no
no
no
```

프로그램 알고리즘 설명

hw1_q1.py :

기본 NFA→DFA 변환 알고리즘은 lecture에 제시된 것을 따랐다. 이 과정에서는 기본적으로 state_queue를 이용한 BFS를 이용하였다.

NFA로 제시되는 state가 20개 이하이기 때문에, DFA의 state로 변환하는 과정에서 중복 탐색을 피하기 위한 확인 및 개수 파악을 위해, DFA의 state에 포함될 NFA의 state를 각 비트별로 더해주어(3, 5번 state → 2^3 + 2^5의 값을 저장) 확인했다. 또한 state_transform를 이용하여 이들을 순서대로 최종적으로 DFA에서 0부터 순차적인 값으로 변환해주도록 하였다.

ε을 확인하는 과정에서는 먼저 0, 1을 통해 이동하는 state를 먼저 구한 후, 이들을 또다른

path_queue에 저장, bfs를 통해 추가적으로 도달할 수 있는 state를 추가하는 방식으로 DFA의 state를 구성했다.

hw1_q2.py :

23hw1.pdf에 제공된 알고리즘을 따랐으며, 이를 NFA, DFA 모두에 적용이 가능하게 구현하였다. DFA에서는 ε을 통한 state 이동이 없지만, 결국 관련 반복문에서 아무 변화가 없기 때문에 문제가 발생하지 않는다.

문자열을 통해 도달하는 state를 구하는 데에 있어 set을 적극적으로 사용하였다.

Environment & How to run

```
root@4a2effdaf185:/home/hw1# python3 --version
Python 3.10.7
root@4a2effdaf185:/home/hw1# cat ./2019-18499/Automata_HW1_2019-18499_compile.sh
root@4a2effdaf185:/home/hw1# cat ./2019-18499/Automata_HW1_2019-18499_Q1.sh
python3 ./hw1_q1.pyroot@4a2effdaf185:/home/hw1# cat ./2019-18499/Automata_HW1_2019-18499_Q2.sh
python3 ./hw1_q2.pyroot@4a2effdaf185:/home/hw1#
```

제공된 docker container 환경에서 실행하였으며, 사용한 파이썬 버전은 3.10.7, 스크립트 파일의 내용은 바로 위의 figure와 같다.