# Automata #3 Homework

2019-18499 김준혁

**실행 결과 (제공 testcase 및 개인 테스트케이스 이용, 터미널에서 바로 입출력)**

```
3 01#
2 0 S
2 1 S
1 # R
1 1 R
1 0 R
3 # S
2 0 S
2 1 S
2 # S
4
10010110101001001000 1001
01101001010110110 1110110 3
10101010111101001000 1000
01010101000010110 1110111 3
0000000000000000
1111111111111111 3
11111111111111
00000000000000 3
```

23hw3.pdf 제공 테스트케이스 결과

```
4 01#
4 0 S
4 1 S
1 # R
1 1 R
1 1 R
2 # L
4 0 S
3 # L
4 # S
4 0 S
3 1 L
5 # S
2
11101111
1111111 5
10111
1111 5
```

hw3 guideline.pdf 제공 테스트케이스 결과

```
14 01#
15 0 S
15 1 S
1 # R
2 0 L
1 1 R
15 # S
2 0 L
3 0 R
4 # R
3 0 R
2 0 L
11 # L
5 1 R
15 1 S
15 # S
6 0 R
15 1 S
15 # S
7 0 R
12 1 L
15 # S
7 0 R
8 1 L
8 # L
9 1 L
15 1 S
15 # S
10 0 L
15 1 S
15 # S
11 0 L
13 1 R
15 # S
11 0 L
4 1 R
4 # R
12 # L
12 # L
14 # R
13 # R
13 # R
14 # L
11
11111111110111111
1111111 14
11111101111111111111111111
111111 14
1111111111101111111111
11111111111 14
101
1 14
1101
1 14
1011
1 14
11011
11 14
11111111101
1 14
10111111111
1 14
110111111111111
11 14
1111111111111011
11 14
```

문제2 min(a,b) 테스트케이스 결과

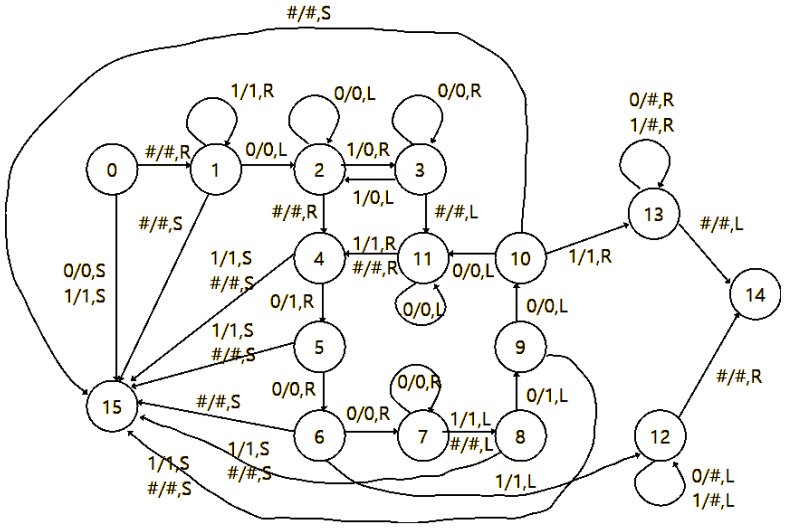
## 프로그램 알고리즘 설명

**hw3.py** :

```
#
#   turing machine structure
#   turing_machine[state number][alphabet][0] : next state
#   turing_machine[state number][alphabet][1] : change alphabet into
#   turing_machine[state number][alphabet][2] : move L/R
#
```

다음과 같은 구조로 입력받은 turing_machine을 구성하고, run_turing_machine()을 통해 실행한다. 처음 tape는 입력받은 문자열로 시작하는데, 만약, head가 범위를 넘어갈 경우, 넘어간 부분에 '#'를 추가하는 방식으로 돌아간다. transition은 단순한 turing machine의 동작 과정에 따라 진행되도록 구성하였다. 함수의 끝에서는 tape의 양쪽 끝에 존재하는 '#'을 strip()을 통해 제거하여 결과 문자열을 반환하도록 하였다.

## 문제 2 Turing machine 설명 (Figure, 내용 정리 스프레드 시트 포함)




| state | alphabet | | | State 설명 |
|---|---|---|---|---|
| N=14 | 0 | 1 | # | |
| 0 | 15 0 S | 15 1 S | 1 # R | start state |
| 1 | 2 0 L | 1 1 R | 15 # S | move to center 0 |
| 2 | 2 0 L | 3 0 R | 4 # R | Move left until reach left num and erase 1 in left num |
| 3 | 3 0 R | 2 0 L | 11 # L | Move right until reach right num and erase 1 in right num |
| 4 | 5 1 R | 15 1 S | 15 # S | Restore 1 in left num |
| 5 | 6 0 R | 15 1 S | 15 # S | helper state for state 6 |
| 6 | 7 0 R | 12 1 L | 15 # S | state for check only one '0' while move right |
| 7 | 7 0 R | 8 1 L | 8 # L | move until reach right num |
| 8 | 9 1 L | 15 1 S | 15 # S | Restore 1 in right num |
| 9 | 10 0 L | 15 1 S | 15 # S | helper state for state 10 |
| 10 | 11 0 L | 13 1 R | 15 # S | state for check only one '0' while move left |
| 11 | 11 0 L | 4 1 R | 4 # R | move until reach left num |
| 12 | 12 # L | 12 # L | 14 # R | erase left num |
| 13 | 13 # R | 13 # R | 14 # L | erase right num |
| | | | | |
| 14 | | | | dead state (success) |
| 15 | | | | dead state (error) |

| state | Alphabet – transition 설명 | | | | |
|---|---|---|---|---|---|
| N=13 | 0 | 1 | # | | |
| 0 | | | move out from # | | |
| 1 | If 0, move left for check 1 in left num | move right until 0 | | | |
| 2 | move until reach left num | Erase 1 in left num | left num is min | | |
| 3 | move until reach right num | Erase 1 in right num | right num is min | | |
| 4 | Restore 1 in left num | | | | |
| 5 | move right | | | | |
| 6 | there are more than one '0' | right num is min, start to erase left num | | | |
| 7 | move right until reach right num | reached right num | reached right num(end) | | |
| 8 | Restore 1 in right num | | | | |
| 9 | move left | | | | |
| 10 | there are more than one '0' | left num is min, start to erase right num | | | |
| 11 | move left until reach left num | reached left num | reached left num(end) | | |
| 12 | erase left num | erase left num | erase finished | | |
| 13 | erase right num | erase right num | erase finished | | |
| | | | | | |
| 14 | | | | | |
| 15 | | | | | |

각 state, transition별 상세한 설명을 일일이 적는 것은 힘들기 때문에 스프레드 시트 형태에 정리하였다. 전반적인 설명으로는, 먼저 가운데에 유일하게 존재하는 0을 찾은 후, 왼쪽부터 좌우 왔다갔다 이동하며 1을 하나씩 지워나간다. 이때, min(a,b)에 대해 a<=b일 때 왼쪽에서, 반대는 오른쪽에서 먼저 #에 도달한다. 어느 쪽에서 먼저 도달하든지, 왼쪽에서부터 다시 0이 1개 남을 때까지 좌우 양끝을 번갈아가며 다시 1로 채운다. 이때 a<=b일 경우 왼쪽으로 이동하던 상황에서, 반대는 오른쪽 이동 중에 0이 1개인 것을 발견한다. 이를 통해 본래의 입력으로 복구되며, 어느 쪽이 최솟값인지 파악이 된다. 이에 따라 최솟값이 아닌 부분을 # 으로 채우면 최솟값을 구할 수 있다.

## 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 환경에서도 올바른 결과가 나오는 것을 확인하였으며, 보고서에 첨부된 파일은 입출력을 한 번에 터미널에서 볼 수 있도록 개인 환경에서 실행했으며, python 3.11.4를 이용한 vscode 내에서 실행했다.