본프로젝트 1 한글모아쓰기 Mealy/Moore Machine

20160389 원종하

1. 프로그램 제작 환경

OS: Windows 10 Home 64bit

언어: Python 3.6.1

Notepad++ v7.5.1로 작성.

2. 실행 방법

폴더 내의 KAuto.py을 실행한다. first_prioirty.py, third_prioirty.py, fp_lambda.txt, tp_lambda.txt, fp_delta.txt, tp_delta.txt가 같은 폴더 내에 존재해야 정상적으로 작동된다.

3. 프로그램 설명

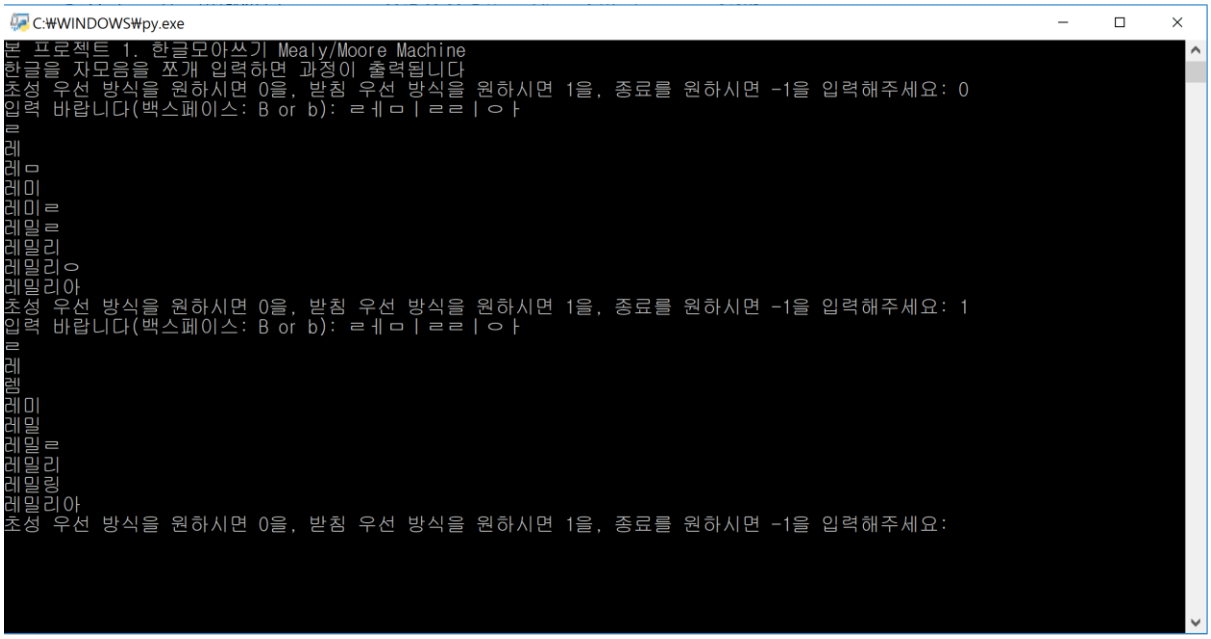
예비프로젝트 1-2에서 제작하였던 Mealy Machine을 변형하여, 자모음이 분리된 한글 문자열을 입력하면 완성된 한글 문자열이 과정과 함께 출력된다.

Ex)

입력: ㄹㅔㅁㅣㄹㄹㅣㅇㅏ	
초성 우선	받침 우선
ㄹ	ㄹ
레	레
렘	레ㅁ
레미	레미
레밀	레미ㄹ
레밀ㄹ	레밀ㄹ
레밀리	레밀리
레밀링	레밀리ㅇ
레밀리아	레밀리아

입력 전 약간의 설명과 함께, 초성 우선 출력과 받침 우선 출력을 선택할 수 있다. 0을 입력하면 초성을, 1을 입력하면 받침을 우선하여 출력한다. -1을 입력하면 종료된다.

아래는 위의 예시를 나타낸 스크린샷이다.



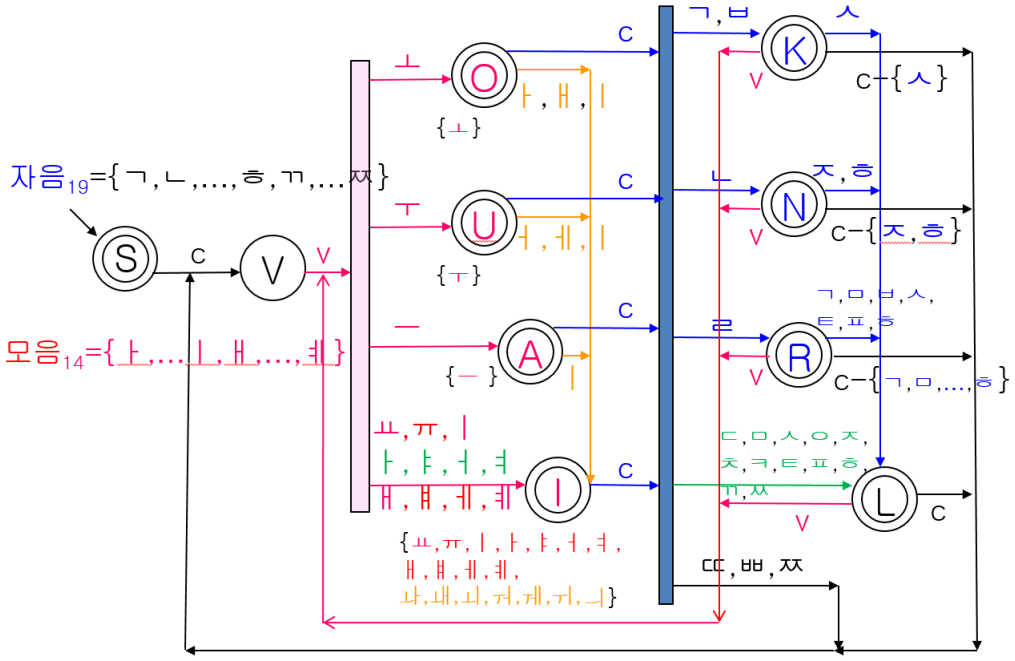
Mealy Machine에서, Q와 Π는 txt 파일에서 입력된 한정된 input만을 사용하므로 필요성을 느끼지 못하여 제거하였다. lambda.txt와 delta.txt는 line마다

$$a\ b_1 + \cdots + b_n\ c$$

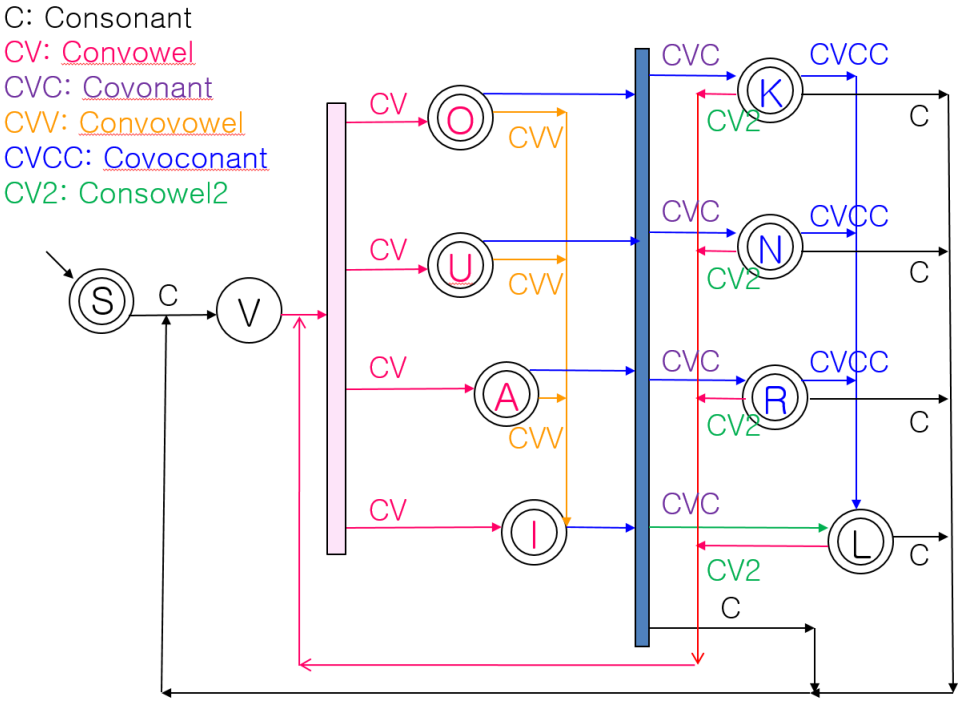
형태로 구성되었고, 이는 $f(a, b_k) = c$를 뜻한다. $(1 \le k \le n$, function f는 δ 혹은 λ). λ function에서의 값 c는 프로그램 내의 method 이름이다.

받침 우선의 오토마타는 다음과 같다.

1) δ function

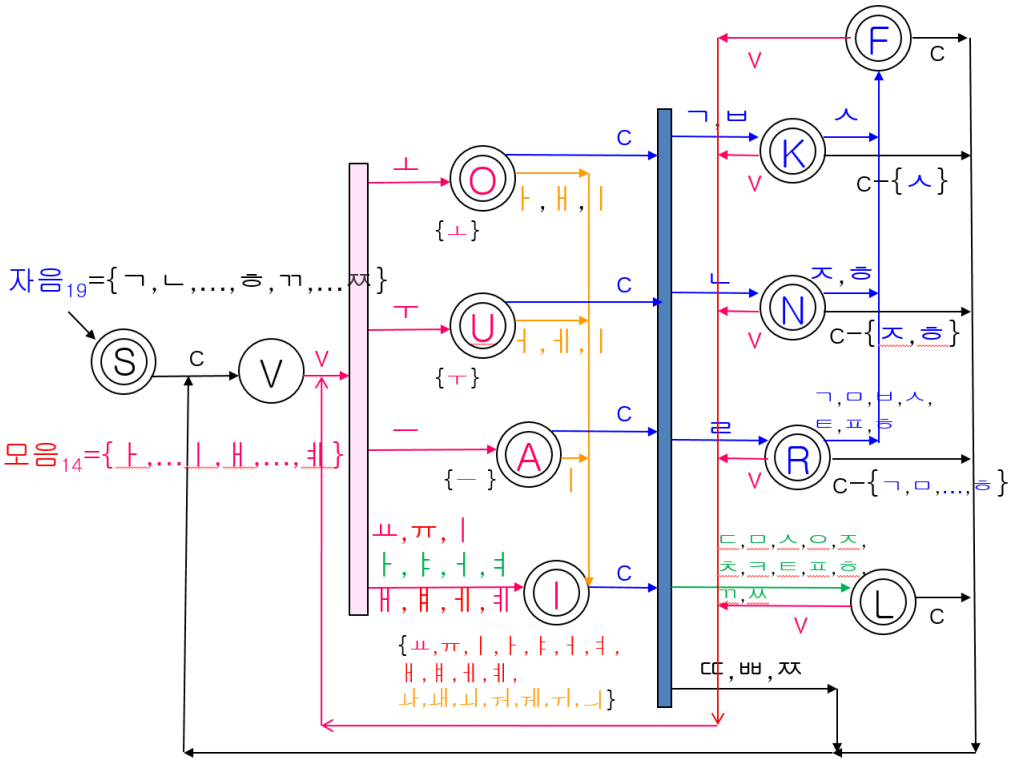


2) λ function

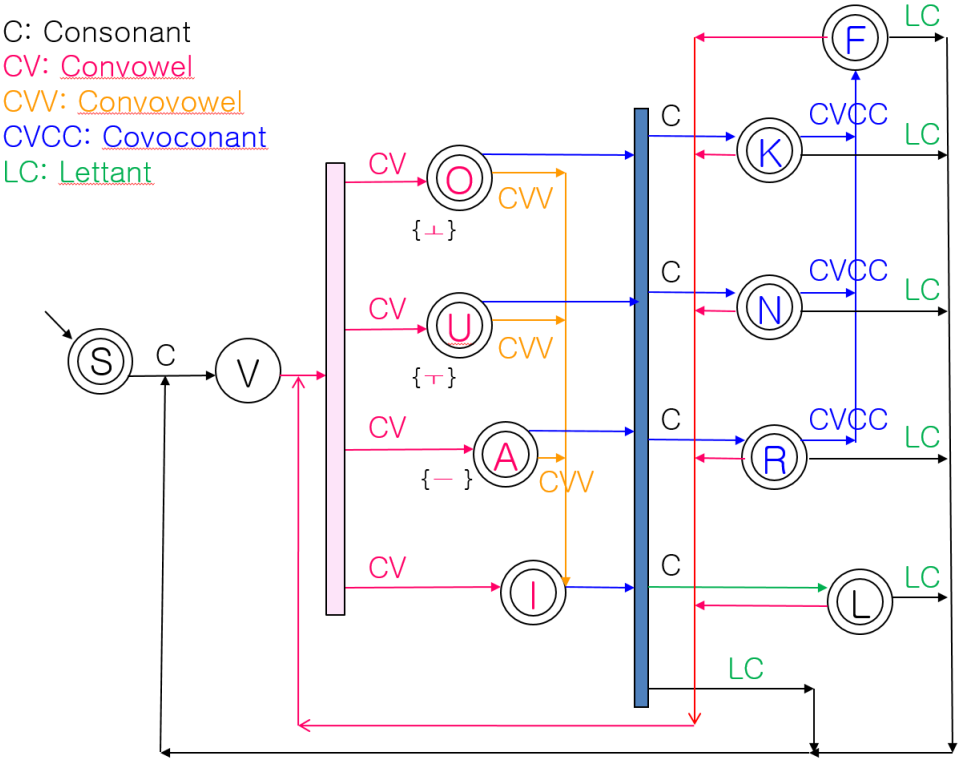


초성 우선의 오토마타는 다음과 같다.

1) δ function



2) λ function



State L과 state F을 구분한 이유는 λ function의 차이 때문이다.

이같이 오토마타의 차이가 존재하기 때문에, 초성 우선과 받침 우선 간의 .txt 파일과 .py 파일을 따로 구분하였다.

4. 코드 설명

1) KAuto.py

SIGMA: 한글 키보드 입력 가능 문자 $\Sigma_{33} + \{\text{SPACE}, \text{B}, \text{b}\}$ list. B와 b는 backspace의 역할을 한다.

init_dfunc(txt), **init_lfunc(txt)**: space와 +를 이용해 적절히 파싱하여 $(\text{a}, \text{b}_\text{k})$를 key로, c를 value로 하는 dictionary를 return한다.

KAutoMealyM(SIGMA, delta_func, lambda_func, qz, L, flag): SIGMA, delta_func, lambda_func는 위에서 언급한 SIGMA와, init_dfunc, init_lfunc을 통해 만들어진 dictionary이며 qz는 초기 state 'S' 이다. L은 case 0|1(초성|받침 우선)에서 first|third_priority.Letter(), flag는 case 값이다. (0 or 1)

KAutoMelayM에 대한 자세한 코드 설명은 주석에서 대신한다.

2) first_priority.py, third_prioirty.py 공통

first, **second**, **third** : 각각 초성, 중성, 종성의 string list이다.

CC: Compound Consonant의 약자로, 겹받침을 이루는 자음 요소 둘의 third list에서의 index tuple을 key로, 겹받침의 third list에서의 index를 value로 하는 dictionary이다.

CV: Compound Vowel의 약자로, 키보드에 존재하는 ㅒ, ㅖ, ㅙ, ㅞ를 제외한 합성 모음(지칭하는 단어가 존재하지 않아 이렇게 칭한다)을 이루는 모음 요소 둘의 second list에서의 index tuple을 key로, 합성 모음의 second list에서의 index를 value로 하는 dictionary이다.

Letter: 아래의 method들과 last_buffer를 포함하는 Class

__init__(self): last_buffer 초기화

value2key(self, element, what): dictionary what에서 value가 element인 key를 return한다. 존재하지 않는다면 None을 return한다.

3) first_priority.py

last_buffer: 아직 완성되지 않은 글자의 요소(자모음)를 저장하는 string list.

Consonant(self, prints, cons): 들어온 자음을 last_buffer에 append한다.

Consowel(self, prints, vowel): 자음 + 모음. last_buffer[:-1]가 하나의 완성된 글자일 때, Lettermake을 이용하여 prints에 추가하고 last_buffer에 앞의 자음만을 남긴다. 들어온 모음을 last_buffer에 append한다.

Convovowel(self, prints, vowel): 자음 + (모음 + 모음). 두 모음이 합성 가능한 모음일 때 원래 존재하던 모음을 pop한 후, CV을 이용하여 합성 모음을 last_buffer에 append한다.

Lettant(self, prints, cons): 글자 + 자음. 이번 자음 입력이 last_buffer가 하나의 완성된 글자가 되는 것을 결정할 때, Lettermake를 이용하여 prints에 추가하고 last_buffer에 입력된 자음만을 남긴다.

Convoconant(self, prints, cons): 자음 + 모음 + 자음 + 자음. 출력 처리는 make와 Lettermake가 담당하기에, 입력된 자음을 last_buffer에 append한다.

위의 method들은 모두 prints을 return한다.

erase(self, x): x가 0일 때는 last_buffer의 마지막 element가 합성 모음일 때 제1모음으로, 단일 모음일 때는 pop한다. x가 그 이외일 때는 last_buffer를 초기화한다.

make(self): last_buffer를 (완성 글자) + (자음) 형태로 만들어 return한다.

Lettermake(self, x = None): x가 None이라면 last_buffer를, x가 일정 범위 내의 integer라면 last_buffer[:x]를 하나의 완성된 글자로 만들어 return한다.

4) third_priority.py

last_buffer: 아직 완성되지 않은 글자의 요소(자모음)의 index를 저장하는 integer list.

Consonant(self, prints, cons): last_buffer가 하나의 완성된 글자일 때, output에 추가하고 last_buffer를 초기화한다. 들어온 초성의 index를 last_buffer에 append한다.

Consowel(self, prints, vowel): 자음 + 모음. 들어온 중성의 index를 last_buffer에 append한다.

Covonant(self, prints, cons): 자음 + 모음 + 자음. 들어온 종성의 index를 last_buffer에 append한다.

Convovowel(self, prints, vowel): 자음 + (모음 + 모음). 두 모음이 합성 가능한 모음일 때 원래 존재하던 모음의 index을 pop한 후, CV을 이용하여 합성 모음의 index를 last_buffer에 append한다.

Covoconant(self, prints, cons): 자음 + 모음 + (자음 + 자음). 두 자음이 합성 가능한 자음일 때

원래 존재하던 자음의 index를 pop한 후, CC를 이용하여 겹자음의 index를 last_buffer에 append 한다.

Consowel2(self, prints, cons): 자음 + 모음 + 자음 + 모음. 종성이 겹받침일 때는 제2자음을 새로운 초성으로, 제1자음을 받침으로 하는 두 개의 글자를 만든 후 앞 글자를 prints에 추가하며, 보통의 받침일 때는 받침을 새로운 초성으로 하는 두 개의 글자를 만든 후 앞 글자를 prints에 추가한다. last_buffer에는 두 번째 글자의 element index를 남긴다.

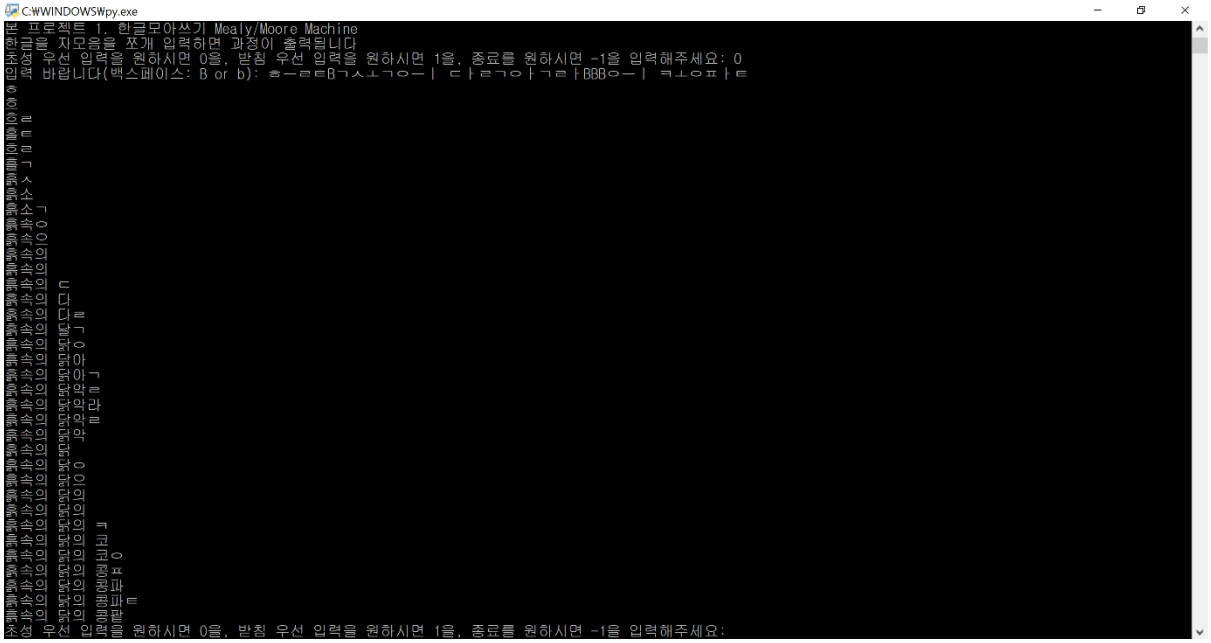
erase(self, x): x가 0일 때는 last_buffer의 마지막 element가 합성 자모음일 때는 단일 자모음으로, 단일 자모음일 때는 pop한다. x가 그 이외일 때는 last_buffer를 초기화한다.

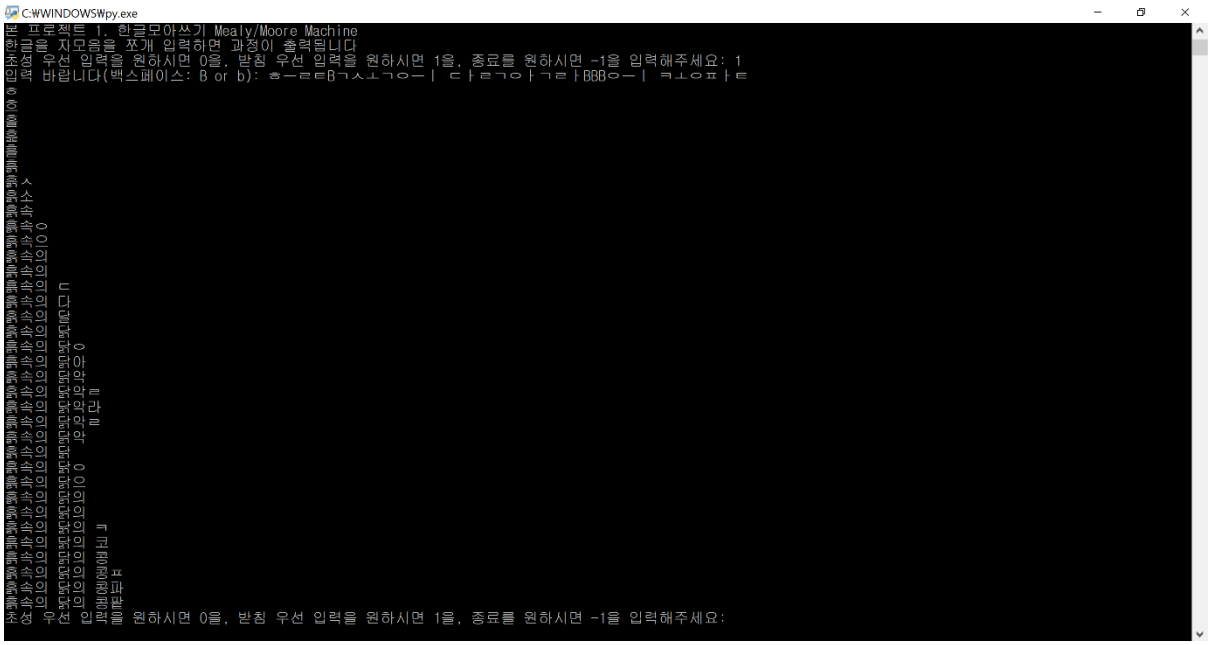
make(self): last_buffer를 하나의 완성된 글자로 만들어 return한다.

5. 보완점

1) ㅏㅏㅏㅏ, ㅇㅇㄴㅇ, ㅛㄲ등의 비문법적인 입력에 대해 유연한 대처가 불가능하다. (현재는 Error Message 출력)
2) 초성 우선 방식에서 흘ㄱ에서 backspace 연산을 할 시 흘이 맞는가? 흐ㄹ이 맞는가? 본 프로그램에서는 흐ㄹ로 출력된다.

6. 입력 예시


C:₩WINDOWS₩py.exe
본 프로젝트 1. 한글모아쓰기 Mealy/Moore Machine
한글을 자모음을 쪼개 입력하면 과정이 출력됩니다
초성 우선 입력을 원하시면 0을, 받침 우선 입력을 원하시면 1을, 종료를 원하시면 -1을 입력해주세요: 0
입력 바랍니다(백스페이스: B or b): ㅎㅡㄹㅌㄱㅅㅗㄱㅇㅣ ㄷㅏㄹㄱㅇㅏㄱㄹㅏBBBㅇㅡㅣ ㅋㅗㅇㅍㅏㅌ
초성 우선 입력을 원하시면 0을, 받침 우선 입력을 원하시면 1을, 종료를 원하시면 -1을 입력해주세요:


C:₩WINDOWS₩py.exe
본 프로젝트 1. 한글모아쓰기 Mealy/Moore Machine
한글을 자모음을 쪼개 입력하면 과정이 출력됩니다
초성 우선 입력을 원하시면 0을, 받침 우선 입력을 원하시면 1을, 종료를 원하시면 -1을 입력해주세요: 1
입력 바랍니다(백스페이스: B or b): ㅎㅡㄹㅌㄱㅅㅗㄱㅇㅣ ㄷㅏㄹㄱㅇㅏㄱㄹㅏBBBㅇㅡㅣ ㅋㅗㅇㅍㅏㅌ
초성 우선 입력을 원하시면 0을, 받침 우선 입력을 원하시면 1을, 종료를 원하시면 -1을 입력해주세요:

7. 초성 우선, 받침 우선 방식에 대한 장단점

각자의 방식은 받침의 유무가 연산의 수를 결정한다. 받침이 없는 글자의 수가 받침이 있는 글자의 수보다 많은 string에서는 초성 우선 방식이 더 좋은 방식이며, 받침이 있는 글자의 수가 받침이 없는 글자의 수보다 많은 string에서는 받침 우선 방식이 더 좋은 방식이다. 극단적으로, "해바라기"와 같이 받침이 없는 string에서 초성 우선 방식은 "해 해바 해바라 해바라기"가 바로 완성되지만 받침 우선 방식에서는 "햅 해발 해바락 해바라기"같이 종성->초성으로 올라오는 연산을 수행하여야 한다. 반대로, "완전한인간"과 같이 받침이 존재하는 string에서 초성 우선 방식은 "와ㄴ 완저ㄴ 완전하ㄴ 완전한이ㄴ 완전한인가ㄴ 완전한인간"과 같이 초성->종성으로 내려가는 연산을 수행하여야 한다. 또한, 초성 우선 방식에서는 마지막 글자가 받침이 있을 시, 그에 대한 처리가 남는다. 위 "완전한인간"의 예시와 같이, 완전한 문장을 이루기 위해서는 "완전한인가ㄴ"을 "완전한인간"으로 바꿔주는 작업이 부가적으로 필요하다.

검색의 관점에서, 기본적으로 초성 우선 방식과 받침 우선 방식 중 하나를 선택하는 것은 불가능하다. 예를 들어, 검색 엔진에 "레밀리아"를 검색할 때 [ㄹㅔㅁ]까지 입력했을 시 검색 엔진의 자동 완성 기능은 '렘'으로 시작하는 단어와, "레ㅁ"인 글자를 모두 생각해주어야 한다. 즉 입력에 대한 출력은 한 가지 방식일지 몰라도, 기능에서는 두 가지 방식을 적용시켜야 한다. 실제로 Google에 [ㄹㅔㅁ]까지 입력하면 출력은 "렘"이지만, 자동완성에서는 [레몬파티, 레몬디톡스, 레몬트리, 레몬, 렘, ...] 순서로 나타난다.