CS814 AI for Autonomous Systems, 2019

Assignment 1, Part 1 - to be started in Week 2 (week beginning Tuesday 1st October)

The idea of this practical exercise is to work towards building an algorithm which can search for strings of the MIU system presented in Chapter 1 of "Gödel, Escher, Bach".

The MIU system is presented as a puzzle: can you transform the string MI into the string MU? In order to accomplish this, you are given the following rules:

Rule 1: If your string ends in an I, you can add a U on the end: $x\mathrm{I} \rightarrow x\mathrm{IU}$

Rule 2: If the string starts with M, you can double what comes after the M: $\mathrm{M}x \rightarrow \mathrm{M}xx$

Rule 3: If you have III in a string, you can replace it with a U: $x\mathrm{III}y \rightarrow x\mathrm{U}y$

Rule 4: If you have UU in a string, you can delete it altogether: $x\mathrm{UU}y \rightarrow xy$

x or y can be empty strings

It turns out that transforming MI into MU is impossible, but it is possible to derive lots of other strings, such as MIUUI:

MI (Axiom) start state.
→ MII (Rule 2)
→ MIIII (Rule 2)
→ MIIIIU (Rule 1)
→ MIUU (Rule 3)
→ MIUUIUU (Rule 2)
→ MIUUI (Rule 4) end state "Theorem".

another branch of AI "theorem proving"

(There are six steps in this derivation - is this the shortest possible derivation of this string? If you've not seen this puzzle before (or even if you have) try to write down a shorter derivation for MIUUI.)

Since producing these derivations by hand can be tedious, we want to write a search program (in Python) to do the job for us. The overall aim is to start with MI and use the rules provided to do a tree search for the string we want to produce.

Part 1: the next_states(s) function

def next_states(s):

Given a string, s, as input, return a list of all possible strings that can be derived from s in a *single step*.

For example:

```
1. next_states("MI")   → ["MIU","MII"]
2. next_states("MIU")  → ["MIUIU"]
3. next_states("MUI")  → ["MUIU","MUIUI"]
4. next_states("MIIII") → ["MIIIIU","MIIIIIIII","MUI","MIU"]
5. next_states("MUUII") → ["MUUIIU","MUUIIUUII","MII"]
6. next_states("MUUUI") → ["MUUUIU","MUUUIUUUI","MUI"]
```

CS814 AI for Autonomous Systems, 2019

Assignment 1, Part 1 - to be started in Week 2 (week beginning Tuesday 1st October)

The idea of this practical exercise is to work towards building an algorithm which can search for strings of the MIU system presented in Chapter 1 of "Gödel, Escher, Bach".

The MIU system is presented as a puzzle: can you transform the string MI into the string MU? In order to accomplish this, you are given the following rules:

Rule 1: If your string ends in an I, you can add a U on the end: $x\mathrm{I} \rightarrow x\mathrm{IU}$

Rule 2: If the string starts with M, you can double what comes after the M: $\mathrm{M}x \rightarrow \mathrm{M}xx$

Rule 3: If you have III in a string, you can replace it with a U: $x\mathrm{III}y \rightarrow x\mathrm{U}y$

Rule 4: If you have UU in a string, you can delete it altogether: $x\mathrm{UU}y \rightarrow xy$

x & y can be empty strings

It turns out that transforming MI into MU is impossible, but it is possible to derive lots of other strings, such as MIUUI:

MI (Axiom) start state.
→ MII (Rule 2)
→ MIIII (Rule 2)
→ MIIIIU (Rule 1)
→ MIUU (Rule 3)
→ MIUUIUU (Rule 2)
→ MIUUI (Rule 4) end state "Theorem".

another branch of AI "theorem proving"

(There are six steps in this derivation - is this the shortest possible derivation of this string? If you've not seen this puzzle before (or even if you have) try to write down a shorter derivation for MIUUI.)

Since producing these derivations by hand can be tedious, we want to write a search program (in Python) to do the job for us. The overall aim is to start with MI and use the rules provided to do a tree search for the string we want to produce.

Part 1: the next_states(s) function

def next_states(s):

Given a string, s, as input, return a list of all possible strings that can be derived from s in a *single step*.

For example:

```
1. next_states("MI")  → ["MIU","MII"]
2. next_states("MIU")  → ["MIUIU"]
3. next_states("MUI")  → ["MUIU","MUIUI"]
4. next_states("MIIII") → ["MIIIIU","MIIIIIIII","MUI","MIU"]
5. next_states("MUUII") → ["MUUIIU","MUUIIUUII","MII"]
6. next_states("MUUUI") → ["MUUUIU","MUUUIUUUI","MUI"]
```

Note that the resulting list must contain no duplicates: if a string is already in the result from an earlier application of a rule, don't add it again (see example 6).

Write this function and then test it with the examples above (and others).

This part of the assignment is worth 3% of the mark for the class.

To get the credit, you need to get the practical signed off by me. When your program is working correctly, please email your code to `John.Levine@strath.ac.uk`

In the second part of this practical, we will use the `next_states(s)` function to implement various search methods, such as depth-first and breadth-first search.

John Levine
1st October 2019

Number of strings that can be generated is infinite;

"search space is infinite".

✱ see Counter ( ).

---

def search_for(s): — assumption is that string is achievable.

return shortest legal path from MI to s.

eg

MIUIU

["MI", "MIU", "MIUIU"]

use: "breadth first search

expand level | fully at n before

check if it is goal