Analiza algoritmilor: Tema #1
TM accepter in Python

Termen de predare: 9 Noiembrie 2020, ora 23:55
Responsabili: Cătălin Chiru, Matei Popovici

Obiectivele temei:

Obiectivele temei sunt înţelegerea şi aprofundarea felului în care funcţionează o maşină Turing, ce este o configuraţie a unei maşini Turing, acceptarea unui cuvânt, precum şi familiarizarea cu limbajul Python; citirea de la tastatură, folosirea arrayurilor şi a dicţionarelor.

Introducere:

O maşină Turing M reprezintă un tuplu (K, Σ, δ, q_0 ,F) unde:
K este mulţimea stărilor lui M
Σ - alfabetul maşinii
δ : K x Σ → K x Σ x {L,R,H} - funcţia de tranziţie
q_0 - starea iniţială
F - mulţimea de stări finale

Pentru informaţii în legătură cu modul în care codificăm o maşină Turing şi definiţia acceptării: link.
În ceea ce priveşte detalii recapitulative referitoare la noţiuni de Python, accesaţi acest link.

0. Citirea unei MT dintr-un de la tastatură (10p):

Implementaţi o funcţie **`readTM`** care primeşte un string ce conţine codificarea unei Maşini Turing şi întoarce o reprezentare internă a acesteia. Hint: Pentru aceasta din urma puteţi folosi tupluri şi dicţionare din Python.

Pentru codificarea unei maşini Turing, vom folosi următoarele convenţii:

- Stările sunt reprezentate ca valori întregi consecutive, iar starea iniţială este întotdeauna **`0`**.
- Alfabetul este mulţimea caracterelor ASCII alfabetice, de la **`A`** la **`Z`** şi simbolul **`#`**.
- La stânga şi dreapta cuvântului toate pozitiile sunt goale. Pozitiile goale se marchează explicit cu simbolul '#' şi pot apărea ca input/output în cadrul tranziţiilor. Dacă ajungem în situaţii în care la stânga cursorului sau la dreapta rămânem cu un string gol "" vom adăuga un "#"

  Exemple: (cuvânt_stânga, stare, "") => (cuvânt_stânga, stare, "#")

  ("", stare, cuvânt_dreapta) => ("#", stare, cuvânt_dreapta)
- Execuţia maşinii se opreşte în momentul în care nu mai putem tranziţiona din starea curentă

Codificarea unei maşini va avea, pe prima linie, numărul de stări, pe a doua linie, lista stărilor finale separate prin spaţiu, iar pe liniile următoare, cate o tranziţie, una pe fiecare linie:

```
nr_de_stări_ale_maşinii
stare_finală_1 stare_finală_2 stare_finală_p
tranziţie_1
tranziţie_2
…
tranziţie_n
```

O tranziţie este codificata ca un şir de forma:

```
starea_curentă simbol_curent starea_următoare simbol_nou poziţie_cursor
```

Exemplu:

```
2
0 1
0 a 1 b L
```

Maşina de mai sus are două stări, 0 şi 1, iar ambele sunt finale. Singura tranziţie a acesteia este $\delta(0,a) = (1,b,L)$.

1. Un pas (⊢) din execuţia masinii Turing (30p)

Implementaţi funcţia **`step`** care primeşte ca parametru o configuraţie a unei maşini Turing şi întoarce configuraţia care rezultă după execuţia unui pas a maşinii sau valoarea **`False`** dacă maşina se agaţă.

O configuraţie va fi codificata ca un triplet (u,q,v) unde:
u - este cuvântul din stânga cursorului
q - este starea curentă
v - este cuvântul din dreapta cursorului

2. Acceptarea unui cuvânt (30p)

Implementaţi funcţia **`accept`** care primeşte ca parametru o maşină M şi un cuvânt, şi întoarce valoarea booleană **`True`**, dacă cuvântul este acceptat, sau **`False`**, în caz contrar.
! Pentru această cerinţă se garantează faptul că nu vor exista cicluri în maşină. **!**

3. Acceptarea după k paşi a unui cuvânt (30p)

Implementaţi funcţia **`k_accept`** care primeşte ca parametru un întreg **`k`**, o maşină **`M`** şi un cuvant **`w`**, şi întoarce valoarea booleană **`True`**, dacă cuvântul este acceptat în **`k`**, sau **`False`**, în caz contrar.
! PENTRU ACEASTĂ CERINŢĂ **POT** EXISTA CICLURI ÎN MAŞINĂ. !

Modul de testare şi structura unui fişier de test:

Există trei tipuri de fişiere de testare, fiecare dintre acestea testează funcţionalităţile funcţiilor step, accept şi k_accept.
Un fişier de test are următoarea structura:

- Pe prima linie - tipul task-ului.
- Pe linia a doua o listă de inputuri corespunzătoare taskului separate prin spaţiu
- Pe restul liniilor până la sfârşitul fişierului text configuraţia maşinii M pe care se vor testa inputurile

Un fișier de output va arăta astfel:

- O linie pe care se află în ordine output-urile corespunzătoare taskului separate prin spațiu.

<u>Tipul 1:</u>
Input:
step
<config_1> <config_2> … <config_n>
<codificare MT>

Linia a doua conține o listă de configurații care vor reprezenta pe rând inputul funcției **`step`** definită la **`1`**.

Output:
<next_config_1> <next_config_2> … <next_config_n>

Ieșirea constă în lista de configurații întoarse de step pentru fiecare element din input.

<u>Tipul 2:</u>
Input:
accept
cuvânt_1 cuvânt_2 … cuvânt_n
<codificare MT>

Cea de-a doua linie conține o listă de cuvinte care vor reprezenta pe rând inputul funcției **`accept`** definită la **`2`**.

Output:
bool_accept_1 bool_accept_2 … bool_accept_n

Ieșirea constă în lista de booleeni corespunzători răspunsului lui **`accept`** pentru fiecare element din input.

Tipul 3:

Numărul de k paşi corespunzător cuvântului este delimitat de acesta prin virgulă.

Input:

k_accept
cuvânt_1,k_1 cuvânt_2,k_2 … cuvânt_n,k_n
<codificare MT>

Cea de-a doua linie conţine o listă de perechi: (cuvânt, număr maxim de tranziţii permise pentru cuvântul curent) care vor reprezenta pe rând inputul funcţiei **`k_accept`** definită la **`3`**.

Output:

bool_k_accept_1 bool_k_accept_2 … bool_k_accept_n

Ieşirea constă în lista de booleeni corespunzători răspunsului lui **`k_accept`** pentru fiecare element din input.

Exemple:

Step:

Input:

step

(#,0,ABA) (D,2,BBB) (#J,1,BAT) (#AA,2,ZXY) (A,0,BD) (CO,2,DE) (CA,0,A) (K,2,DC)

5

-

0 A 1 B R

1 B 1 B L

1 A 3 A H

2 B 3 A H

0 D 2 A R

2 D 1 F L

0 B 4 B H

Output:

(#B,1,BA) (D,3,ABB) (#,1,JBAT) False (A,4,BD) (C,1,OFE) (CAB,1,#) (#,1,KFC)

Accept:

Input:
accept
ABABAA DABAA #DDAABCC BBB AA AB2 AB#
7
6 5
3 A 5 A H
1 B 4 A L
4 B 2 A R
2 B 3 A L
0 # 6 # H
2 # 6 # H
0 A 1 B R
4 # 6 # H
3 # 3 A R
0 B 4 D R
1 # 6 # H
4 A 3 A R
3 B 6 # H

Output:
False False True True False False False

kAccept:

Input:
k_accept
ABBA,10 ABBBBBA#,7 AB##BBAA,14 AA#A#,5
7
5 6
3 A 5 A H
1 B 4 A L
4 B 2 A R
2 B 3 A L
0 # 6 # H
2 # 6 # H
0 A 1 B R
4 # 6 # H
3 # 3 A R
0 B 4 D R
1 # 6 # H
4 A 3 A R
3 B 6 # H
2 A 1 A R
Output:
True True True False

Checker:

Testarea şi punctarea temei se vor face automat cu ajutorul platformei hackerrank:
https://www.hackerrank.com/tema-1-aa

Chiar dacă nu vi s-a cerut un readme, aş dori să puneţi comentarii sugestive şi să aveţi un coding style cât mai lizibil.

Vă rog să încărcaţi tema atât pe Hackerrank cât şi pe Moodle ca arhivă!

Mult succes! :)