Computational Neuroscience, Digital Intelligence Laboratory

IMPLEMENTACJA ASOCJACYJNEJ MASZYNY TURINGA

Waldemar Wietrzykowski

Instytut Inteligencji Stosowanej, 2 listopada 2021

Streszczenie *niniejsza praca pokazuje, że najlepszym językiem do napisania programu asocjacyjnej maszyny Turinga jest język Pyton. Wspomnieć należy, że asocjacyjna maszyna Turinga tym się różni od standardowej maszyny Turinga, że jej działania nie trzeba programować, a jej zadaniem jest samo-nauczanie się języka w zakresie gramatyki regularnej na podstawie ciągów wejściowych. Maszyna ta zgromadzi wiedzę na liście asocjacji.*

Wstęp

Prace teoretyczne nad asocjacyjną maszyną Turinga przedstawione są w pracy „Asocjacje regularne". Prace te należało potwierdzić za pomocą odpowiedniego modelu komputerowego. Aby tego dokonać potrzeba było w pierwszym rzędzie wybrać odpowiedni język programowania najlepiej pasujący do postawionego zadania.

Pierwsze prace programistyczne nad samo-uczeniem się maszyny Turinga przedstawiono w opracowaniu „Implementacja rachunku sekwentowego" (2018 rok). Językiem programowania był C++. Powstawały kolejne wersje programu o nazwach: „mw34" - 558 lini programu źródłowego, „consolid" - 509 linii, „slp" - 388 linii, „rachsekw" - 674 linii, „slp_rach" - 1197 linii, „slp_nl" - 664 linii, „paradygmat" - 745, „elementary" - 831 linii. Kolejna wersja programu zawierała ulepszenia poprzednich wersji. Należy podkreślić, że program „mw34" działał jedynie na ciągach bitowych, a następne wersje na ciągach liczbowych. W rozwoju programów przyświecała jedna myśl, aby znaleźć najprostszy algorytm do konstrukcji takiej maszyny. Dopiero prace nad maszyną asocjacyjną, przedstawione w opracowaniu „Asocjacje regularne", pokazały, że najprostszym modelem samo-uczenia maszynowego w zakresie gramatyki regularnej jest asocjacyjna maszyna Turinga.

Zalety języka Pyton

Do napisania programu asocjacyjnej maszyny Turinga użyto języka Pyton (ver. 3.9.7). Okazał się on najbardziej odpowiednim i doskonale pasującym językiem programowania do napisania tej maszyny.

Po pierwsze, jest on najbardziej intuicyjnym i wygodnym językiem programowania ze wszystkich języków, z którymi miałem do czynienia.

Po drugie program źródłowy podobnej maszyny w C++ zawierał od 500 do 1100 linii programowych (w zależności od wersji programu), natomiast w Pytonie zawiera zaledwie 52 linie.

Po trzecie Pyton posiada filozofię języka i danych zadziwiająco przystającą do metodologii asocjacyjnej maszyny Turinga. Programując asocjacyjną maszynę w Pytonie widzimy od razu ją w całości i nie musimy się najpierw przedzierać przez definiowanie elementarnych składników czy funkcjonalności tej maszyny jak w C++. Pisząc program w Pytonie od razu myślimy o maszynie asocjacyjnej tak, jakby Pyton był natywnym językiem do programowania asocjacyjnej maszyny i dlatego w tym języku najprościej jest ją napisać.

Ponadto nie ma potrzeby deklarowania typów zmiennych ani typów argumentów funkcji. Pyton sam na podstawie kontekstu nazwy zmiennej w programie przydziela jej typ, a także na podstawie kontekstu wywołania argumentów funkcji ustawia ich typ i dla tego typu stosuje realizację funkcji.

Poza tym w Pytonie dozwala się deklarowanie wewnątrz innych funkcji (w C++ jest to niemożliwe), a także można zwracać wiele wartości funkcji w postaci return x,y,c. Po wykonaniu takich funkcji zwracane wartości są dostępne w następujący sposób: a,b,c = funkcja().

Nie potrzeba też pamiętać o podłączaniu bibliotek jak w C++, gdyż wbudowane funkcje są zazwyczaj wystarczające, ani też rozpoczynać programu od funkcji main(). Cały program może składać się nawet z pojedynczej instrukcji matematycznej.

Oprócz tego, zamiast zaznaczania bloku programu znakiem początku i końca np. { }, Pyton stosuje bardzo wygodne wcięcia (Tab). Wielkość wcięcia oznacza przynależność do danego bloku. Większe wcięcie oznacza blok zawarty w danym bloku. Pyton nadzoruje zastosowane wcięcia nadając wszystkim programom w Pyton ten sam format, czytelny i odporny na nieprawidłowe oznaczenie bloków.

Po napisaniu programu możemy od razu go wykonać w okienku interpretera Pytona, a następnie kontynuować konwersję z interpreterem.

Interpreter języka Pyton jest darmowy dla różnych systemów operacyjnych. Jest lekki, nie obciąża systemu i szybko pracuje.

Do okienka Pytona można kopiować całą treść programu, jego wycinki, jak też funkcje czy pojedyncze wyrażenie z danymi oraz kopiować z okienka rezultaty dokonanej interpretacji. Nadaje się szczególnie w pracach naukowych, gdzie szybko można testować różne modele tworzonych teorii.

Można go używać również jako konwersacyjny kalkulator, zapisując działania na liczbach i otrzymując wyniki.

Największą mocą Pytona jest jednak to, że jest on wyspecjalizowany do przetwarzania list i zbiorów. Listy mogą być zagnieżdżane, a każdy element listy może mieć inną strukturę. Listy mogą być przeglądane, a ich elementy dopisywane do niej i usuwane. Łatwy jest też dostęp do każdego elementu listy. Elementami listy mogą też być zbiory. Na zbiorach z kolei dostępne są wszystkie operacje matematyczne, jakie można wykonać na zbiorach takie jak: suma, przekrój, różnica, przynależność do zbioru, itd. Każdy zbiór spełnia aksjomatyczną teorię mnogości, zgodnie z którą każdy element zbioru powtarza się tylko raz. Inna nazwa zbioru to zestaw (set).

Powyższe sprawia, że w Pytonie inna jest też filozofia programowania niż w języku C++. Program w C++ zaczynamy od deklaracji struktur danych, deklaracji funkcji operujących na tych strukturach i przydziałem na nie pamięci. W razie modyfikacji tych struktur musimy dostosowywać funkcje, operujące na tych strukturach. Jest to żmudna praca. W Pytonie struktura danych przedstawiana jest w postaci listy i zbiorów, którą można przygotować nawet na kartce papieru, czy w dowolnym edytorze teksu, a następnie skopiować ją to interpretera Pyton i sprawdzić poprawność tej struktury. Nie potrzeba też myśleć o przydziale pamięci na zaprojektowaną i modyfikowaną strukturę listową. Pyton sam przydziela i zwalnia pamięć komputera.

Pyton szczególnie nadaje się do poszukiwań rozwiązań teoretycznych, technologicznych czy z zakresu interdyscyplinarnej kognitywistyki, a możliwość szybkiego przełożenia ich na program w Pytonie daje gwarancję na prostą i łatwą weryfikację poprawności naszego myślenia.

Wynika z tego, że Pyton jest bardziej uniwersalny i intuicyjnie zbliżony do myślenia człowieka od innych języków (jak C++), w których nadal musimy mieć świadomość o obecności procesora w komputerze.

Asocjacyjna maszyna Turinga w języku Pyton

Wbudowane w języku Pyton operacje na listach i zbiorach przydatne są do napisania asocjacyjnej maszyny Turinga. Pyton obsługuje następujące typy wieloelementowych danych:

```
p = [0,3,6,7]          #  list (lista)

p = (0,3,6,7)          #  tuple (krotka)

p = '{0,3,6,7}'        #  str (łańcuch znakowy)

p = {0,3,6,7}          #  set (zbiór)
```

Korzystając z tych typów danych można zapisać strukturę listy asocjacyjnej maszyny Turinga w języku Pyton. Lista z dwoma asocjacjami wygląda przykładowo następująco:

```
list_asoc = [[{1,2,3,7}, {1,2,4,7}], [{0,4,5,6}, {0,3,5,6}]]
```

Lista ta składa się z dwóch asocjacji:

```
list_asoc[0] = [{1, 2, 3, 7}, {1, 2, 4, 7}]
list_asoc[1] = [{0, 4, 5, 6}, {0, 3, 5, 6}]
```

Każda asocjacja ma swój zbiór poprzedników i zbiór następników.

```
list_asoc[0][0] = {1, 2, 3, 7}      # poprzedniki pierwszej asocjacji
list_asoc[0][1] = {1, 2, 4, 7}      # następniki pierwszej asocjacji

list_asoc[1][0] = {0, 4, 5, 6}      # poprzedniki drugiej asocjacji
list_asoc[1][1] = {0, 3, 5, 6}      # następniki drugiej asocjacji
```

Lista asocjacji może być przeglądana w pętli (asocjacja za asocjacją):

```
for asoc in list_asoc
```

Pętla for może być zakończona instrukcją:

```
else
```

która oznacza, że blok znajdujący się pod else zostaje wykonany, jeżeli w pętli for nie została wywołana żadna instrukcja break, która kończy przed czasem pętlę.

Zmienna asoc jest tym samym egzemplarzem kolejnego elementu list_asoc co oznacza, że wykonując operacje na zmiennej asoc zmiany przenoszą się na list_asoc (jest to odpowiednik działania na wskazaniach zmiennych wskaźnikowych w C++).

Po pierwszej iteracji pętli for zmienna asoc na wartość pierwszej asocjacji:

```
asoc = [{1, 2, 3, 7}, {1, 2, 4, 7}]
```

Poprzednik i następnik tej asocjacji są następujące:

```
asoc[0] = {1, 2, 3, 7}
asoc[1] = {1, 2, 4, 7}
```

Po drugiej interacji pętli for zmienna asoc ma wartość drugiej asocjacji:

```
asoc = [{0, 4, 5, 6}, {0, 3, 5, 6}]
```

Poprzednik i następnik tej asocjacji są następujące:

```
asoc[0] = {0, 4, 5, 6}
asoc[1] = {0, 3, 5, 6}
```

Na liście asocjacji list_asoc można wykonać takie operacje jak:

```
list_asoc.append(asoc)      # dodawanie asocjacji do listy asocjacji:
list_asoc.remove(asoc)      # usuwanie asocjacji z listy asocjacji:
```

Pusta lista asocjacji jest równa:

```
list_asoc = [ ]
```

Na poprzednikach i następnikach asocjacji (zbiorach) można wykonać następujące operacje:

```
x in asoc[0]                # przynależność elementu x do poprzednika asocjacji asoc
asoc[1].add(x)              # dodanie elementu x do następnika asocjacji asoc
asoc1[0] & asoc2[0]         # iloczyn (przekrój) poprzedników asocjacji asoc1 i asoc2
asoc1[1] & asoc2[1]         # iloczyn (przekrój) następników asocjacji asoc1 i asoc2
set()                       # zbiór pusty

(asoc1[0] & asoc2[0] != set()) or (asoc1[1] & asoc2[1] != set())
                            # sprawdzanie, czy dwie asocjacje są łączne

asoc1[0] |= asoc2[0]        # sumowanie poprzedników asocjacji: asoc1 = asoc1 + asoc2
asoc1[1] |= asoc2[1]        # sumowanie następników asocjacji: asoc1 = asoc1 + asoc2
```

Dane wejściowe do maszyny asocjacyjnej mogą być wprowadzone następująco:

```
p = 3,1,4,6,0,6,5,0,3,2,4
```

Pyton automatycznie konwertuje ten zapis na listę typu tuple (krotka):

```
p = (3,1,4,6,0,6,5,0,3,2,4)
```

Dane wejściowe nienumeryczne do maszyny asocjacyjnej mogą być wprowadzone następująco:

```
p = 'p,q,r,s'
```

Następnie korzystamy z metody:

```
p = p.split(',')
```

oznaczającej podział łańcucha znakowego na listę elementów i otrzymujemy:

```
p = ['p', 'q', 'r', 's']
```

Uwaga! Jeżeli wprowadzimy listę zmiennych

```
p = p,q,r,s
```

Pyton konwertuje ją na listę typu: tuple (krotka) złożoną z listy wartości tych zmiennych.

Listy typu tuple i list są przeglądane w ten sam sposób przy pomocy pętli for:

```
for s in p
```

lub w postaci:

```
for i in range(len(p))
```

Funkcja range(x) zachowuje się tak, jakby to był ciąg ze wzrastającymi o 1 liczbami naturalnymi od 0 do x-1.

Aby oczyścić łańcuch s ze zbędnych spacji i apostrofów (') robimy to w następujący sposób:

```
for ch in "' ":
        s = s.replace(ch,'')
```

Listę asocjacji można wyprowadzić w postaci sformatowanego łańcucha znaków:

```
s += f '({str(asoc[0])[1:-1]})({str(asoc[1])[1:-1]})'
```

Dla wyjaśnienia:

```
asoc[0]                 #   {1,2,3,7}
str(asoc[0]) =          #   '{1,2,3,7}'
str(asoc[0])[1:-1]      #   '1,2,3,7'
f '({str(asoc[0])[1:-1]})   #   '(1,2,3,7)'
```

```
f '({str(asoc[0])[1:-1]})({str(asoc[1])[1:-1]})'
                                    #   '(1,2,3,7)(1,2,4,7)'

s += f '({str(asoc[0])[1:-1]})({str(asoc[1])[1:-1]}),'          # stosujemy w pętli for
                                    #   '(1,2,3,7)(1,2,4,7),(0,4,5,6)(0,3,5,6),'
```

Ostatnią asocjację w list_asoc wskazuje indeks: -1:

```
list_asoc[-1]
```

Program asocjacyjnej maszyny Turinga umieszczamy jest w ciele funkcji consolid(list_asoc, p). Funkcja zwraca dwie lokalne wartości:

```
return list_asoc, s
```

Łańcuch s przedstawia sformatowany zapis listy asocjacji list_asoc.

Przy wywoływaniu funkcji consolid() otrzymujemy listę asocjacji a i jej sformatowany zapis s:

```
a, s = consolid(list_asoc, p)
```

Parametr p jest listą elementów wejściowych przeznaczoną do konsolidacji przez maszynę asocjacyjną, a parametr list_asoc jest listą asocjacji uzyskaną z poprzedniej konsolidacji lub jest to lista pusta list_asoc = [ ].

Przykład zastosowania funkcji consolid():

```
list_asoc = [ ]
a, s = consolid(list_asoc, p1)
a, s = consolid(a, p2)
```

Po wywołaniu funkcji consolid() wpisujemy do okienka interpretera Pyton nazwę s oraz zatwierdzamy<Enter>. Otrzymujemy sformatowany zapis listy asocjacji, a wpisując nazwę a i zatwierdzając <Enter> otrzymujemy listę asocjacji do dalszego przetwarzania.

Pełny kod źródłowy asocjacyjnej maszyny Turinga

```
# Maszyna asocjacyjna Turinga
# Waldemar Wietrzykowski 21.10.2021

p = 3,1,4,6,0,6,5,0,3,2,4,3,1,4,6,6,5,0,6,5,0,6,3,2,1,1,7,7,2,1,4

list_asoc = []

def consolid(list_asoc, p):
```

```
    def cons(asoc):
        for a in list_asoc:
            if a != asoc:
                if (a[0] & asoc[0] != set()) or (a[1] & asoc[1] != set()):
                    a[0] |= asoc[0]
                    a[1] |= asoc[1]
                    list_asoc.remove(asoc)
                    break

    if type(p) is str:
        p = p.split(',')
    else:
        if type(p) is tuple:
            pass

    for i in range(len(p)-1):
        if list_asoc == []:
            list_asoc = [ [{p[i]},{p[i+1]}] ]
        else:
            for asoc in list_asoc:
                if p[i] in asoc[0]:
                    asoc[1].add(p[i+1])

                    cons(asoc)

                    break
            else:
                list_asoc.append([{p[i]},{p[i+1]}])
                cons(list_asoc[-1])
    s = ''
    for asoc in list_asoc:
        s += f'({str(asoc[0])[1:-1]})({str(asoc[1])[1:-1]})'

        if asoc != list_asoc[-1]:
            s += ','

    for ch in "' ":
        s = s.replace(ch,'')

    return list_asoc, s


a, s = consolid(list_asoc, p)
```

Przykłady działania funkcji consolid()

Dla sprawdzenia poprawności działania asocjacyjnej maszyny Turinga napisanej w Pytonie przeliczono powtórnie przykłady zamieszczone w poprzednich pracach. Wyniki umieszczono w poniższych tabelach.

Przykłady z: Teoria świadomej maszyny	
Ciąg wejściowy	**Konsolidacja w Pytonie**
3,5,0,6,4,2,1,0,3,6,4,7,1,0,6,2, 1,0,6,1,0,6,7,2,1,5,3,3,6,1,0	(0,1,3,5)(0,3,5,6),(2,4,6,7)(1,2,4,7)
3,5,0,6,4,2,1,0,3,6,4,7,1,0,6,2,1,0,6,1,0	(1,5)(0),(0,3)(3,5,6),(2,4,6,7)(1,2,4,7)
(1,5)(0),(0,3)(3,5,6),(2,4,6,7)(1,2,4,7)+ 6,7,2,1,5,3,3,6,1,0	(0,1,3,5)(0,3,5,6),(2,4,6,7)(1,2,4,7)
(1,5)(0),(0,3)(3,5,6),(2,4,6,7)(1,2,4,7)+ 0,5,6,3	(0,1,2,3,4,5,6,7)(0,1,2,3,4,5,6,7)
3,5,0,6,4,2,1,0,3,6,4,7,1,0,6,2, 1,0,6,1,0,6,7,2,1,5,3,3,6,1,0+ 11,13,8,14,12,10,9,8,11,14,12, 15,9,8,14,10,9,8,14,9,8	(0,1,3,5)(0,3,5,6),(2,4,6,7)(1,2,4,7),(9,13)(8), (8,11)(11,13,14),(10,12,14,15)(9,10,12,15)

Przykłady z: Wystarczalność rachunku sekwentowego	
Ciąg wejściowy	**Konsolidacja w Pytonie**
p,q,r,s,q,s,r,t,u,w	(s,p,r,q)(s,t,r,q),(t)(u),(u)(w)
(s,p,r,q)(s,t,r,q),(t)(u),(u)(w)+ a,z,u,z,a,y	(s,p,r,q)(s,t,r,q),(z,t)(a,u),(a,u)(z,w,y) (p,q,r,s) (q,r,s,t), (z,t) (u,a), a z, u z, a y, u w !!!

!!! - obliczone w źródle

Dowód tożsamości: a z, u z, a y, u w = (a,u)(z,w,y)

Przykłady z: Implementacja rachunku sekwentowego	
Ciąg wejściowy	**Konsolidacja w Pytonie**
'p,q,r,s'	(p)(q),(q)(r),(r)(s)
6,5,0,3,1,2,4,0+ 6,3,1,7,4,0+ 3,2,4,0,3,4,0,0,0+ 3,7,2,4,5,6,6,3,4,0,0	(0,4,5,6)(0,3,5,6),(1,2,3,7)(1,2,4,7)
3,2,4,0,3,4,0,0,0	(2,3)(2,4),(0,4)(0,3)
0,5,3,7,4,3,7,1,7,2,1,1,2,4,0	(0)(5),(4,5)(0,3),(1,2,3,7)(1,2,4,7)
6,0,6,5,5,0,6,6,6,3,4,0,5,0	(0,4,5,6)(0,3,5,6),(3)(4)
0,6,5,0,0,6,5,3,4,3,1,2,1,4,0	(6)(5),(0,4,5)(0,3,6),(1,2,3)(1,2,4)
0,0,3,7,2,4,3,2,2,1,2,7,7,4,0	(0,4)(0,3),(1,2,3,7)(1,2,4,7)
3,4,5,0,5,3,2,4,3,4,0,5,3,4,0+ 3,1,7,1,7,4,6,6,0,0,3,4,0+ 0,6,3,4,3,1,1,7,1,7,7,1,4,0	(1,2,3,7)(1,2,4,7),(0,4,5,6)(0,3,5,6)
5,6,6,0,6,3,4,5,6,0,6,0,0,5,0+ 6,5,3,2,7,4,0,6,0,6,3,4,0,6,0+ 6,3,2,2,2,1,7,2,4,6,5,6,0	(1,2,3,7)(1,2,4,7),(0,4,5,6)(0,3,5,6)
3,1,2,7,1,4,3,2,4,3,4,0,6,0+ 0,3,2,4,3,1,2,2,7,1,7,1,1,4,0+ 5,3,4,6,6,5,6,5,3,2,1,4,0	(1,2,3,7)(1,2,4,7),(0,4,5,6)(0,3,5,6)

6,5,0,5,0,0,5,3,7,2,4,3,4,0+ 6,5,3,7,2,4,0,6,6,5,6,5,0+ 5,3,4,3,7,2,7,2,4,3,2,4,6,5,0+ 5,6,3,4,5,3,4,0,6,0,3,4,0,3,4,0+ 6,0,3,4,0,5,3,1,4,6,6,0,5,0+ 6,6,6,0,3,4,6,3,4,5,0,6,6,5,0+ 0,5,3,7,4,0,6,6,6,3,2,2,7,4,0+ 3,7,2,2,1,4,5,3,1,4,6,0,0,5,0+ 5,3,1,7,1,2,7,4,6,3,7,1,1,4,0+ 0,0,5,3,7,4,0,0,3,7,2,2,7,4,0	(0,4,5,6)(0,3,5,6),(1,2,3,7)(1,2,4,7)

Dalsze przykłady z: Implementacja rachunku sekwentowego	
Ciąg wejściowy	**Konsolidacja w Pytonie**
6,0,0,3,1,2,4,6,6,3,4,6,5,0	(1) (2),(2,3)(1,4),(0,4,5,6)(0,3,5,6)
6,0,0,3,1,2,4,6,6,3,4,6,5,0,6,3,1,7,4	(1)(2,7),(2,3,7)(1,4),(0,4,5,6)(0,3,5,6)
3,7,2,4,5,6,6,3,4,0,0	(7)(2),(2,3)(4,7),(0,4)(0,5),(5,6)(3,6)
3,4,6,6,5,0,6,6,0,0	(3)(4),(0,4,5,6)(0,5,6)
0,3,2,2,1,4,6,6,0,0	(2,3)(1,2),(1)(4),(0,4,6)(0,3,6)
6,5,0,3,1,2,4,0	(6)(5),(4,5)(0),(0)(3),(3)(1),(1)(2),(2)(4)
3,2,4,0,3,4,0,0,0	(2,3)(2,4),(0,4)(0,3)
5,6,0,5,5,3,2,7,4,5,6,0	(6)(0),(0,4,5)(3,5,6),(3)(2),(2)(7),(7)(4)
6,0,3,1,7,7,7,7,7,2,7,4,0	(4,6)(0),(0)(3),(3)(1),(1,2,7)(2,4,7)
6,5,3,1,4,5,6,3,4,0	(4,5,6)(0,3,5,6),(1,3)(1,4)
6,5,0,6,3,2,1,7,2,7,7,4,0	(6)(3,5),(4,5)(0),(0)(6),(1,2,3,7)(1,2,4,7)
6,3,7,1,1,1,2,4,6,0,3,2,4,0,3,4,0	(0,4,6)(0,3,6),(1,2,3,7)(1,2,4,7)
6,3,7,1,2,4,0,3,2,4,3,2,4,0,3,4,0	(0,4,6)(0,3),(7)(1),(1,2,3)(2,4,7)
5,6,0,3,7,2,1,4,5,3,2,4,5,0	(0,5,6)(0,3,6),(3,7)(2,7),(1,2)(1,4),(4)(5)
6,5,5,3,2,7,1,2,7,2,4,0	(5,6)(3,5),(1,3,7)(1,2),(2)(4,7),(4)(0)
6,5,5,3,6,3,5,6,3,6	(3,5,6)(3,5,6)

Przykłady z: Asocjacje regularne	
Ciąg wejściowy	**Konsolidacja w Pytonie**
3,1,4,6,0,6,5,0,3,2,4,3,1,4,6,6,5, 0,6,5,0,6,3,2,1,1,7,7,2,1,4	(1,2,3,7)(1,2,4,7),(0,4,5,6)(0,3,5,6)
7,1,0,2,0,2,1,0,7,2,0,7,1,0,2,2, 1,0,2,1,0,2,7,2,1,1,7,7,2,1,0	(0,1,2,7)(0,1,2,7)
7,5,0,6,0,6,5,0,7,6,0,7,5,0,6, 6,5,0,6,5,0,6,7,6,5,5,7,7,6,5,0	(0,5,6,7)(0,5,6,7)
3,5,0,6,0,6,5,0,3,6,0,3,5,0,6,6, 5,0,6,5,0,6,3,6,5,5,3,3,6,5,0	(0,3,5,6)(0,3,5,6)
1,1,4,4,4,4,1,4,1,4,4,1,1,4,4,4, 1,4,4,1,4,4,1,4,1,1,1,1,4,1,4	(1,4)(1,4)
3,1,4,6,0,6,5,0,3,2,4,3,1,4,6,6, 5,0,6,5,0,6,3,2,1,1,7,7,2,1,4+	(0,1,2,3,4,5,6,7)(0,1,2,3,4,5,6,7)

7,1,0,2,0,2,1,0,7,2,0,7,1,0,2,2, 1,0,2,1,0,2,7,2,1,1,7,7,2,1,0	

Bibliografia

1. Waldemar Wietrzykowski, *Asocjacje regularne*, IIS 2021
2. Waldemar Wietrzykowski, *Subiektywne badanie pamięci*, IIS 2020
3. Waldemar Wietrzykowski, *Konsolidacja elementarna*, IIS 2019
4. Waldemar Wietrzykowski, *Naturalny System Operacyjny*, IIS 2019
5. Waldemar Wietrzykowski, *Dwie maszyny*, IIS 2018
6. Waldemar Wietrzykowski, *Uczenie języka naturalnego*, IIS 2018
7. Waldemar Wietrzykowski, *Implementacja rachunku sekwentowego*, DIL 2018
8. Waldemar Wietrzykowski, *Wystarczalność rachunku sekwentowego. Rachunek tautologiczny*, DIL 2018
9. Waldemar Wietrzykowski, *Teoria wiedzy*, DIL 2018
10. Waldemar Wietrzykowski, *Teoria wiedzy teoretycznej*, DIL 2018
11. Waldemar Wietrzykowski, *Teoria wiedzy praktycznej*, DIL 2018
12. Waldemar Wietrzykowski, *Teoria świadomej maszyny*, DIL 2018
13. Waldemar Wietrzykowski, *Założenia wstępne do projektu świadomej maszyny*, DIL 2017
14. Waldemar Wietrzykowski, *Świadoma maszyna*, DIL 2017
15. Waldemar Wietrzykowski, *Świadoma reakcja maszyn*, DIL 2017
16. Waldemar Wietrzykowski, *Samoucząca się maszyna motoryczna*, DIL 2017
17. Waldemar Wietrzykowski, *Strumień świadomości*, DIL 2017
18. Waldemar Wietrzykowski, S*calanie interpretacji maszyn*, DIL 2017
19. Waldemar Wietrzykowski, *Obraz rzeczywistości z poziomu maszyny*, DIL 2017
20. Waldemar Wietrzykowski, *Świadomość wielu maszyn*, DIL 2017
21. Waldemar Wietrzykowski, *Wiele maszyn*, DIL 2017
22. Waldemar Wietrzykowski, *Znaczenie interpretacji maszyn*, DIL 2017
23. Waldemar Wietrzykowski, *Interpretacje maszyn*, DIL 2016
24. Waldemar Wietrzykowski, *Jednofunkcyjne maszyny*, DIL 2016
25. Waldemar Wietrzykowski, *Samoucząca się maszyna*, DIL 2016
26. Waldemar Wietrzykowski, *Procesor samouczący się*, DIL 2016
27. Waldemar Wietrzykowski, *Processor self-learning*, DIL 2016
