Teoria Współbieżności

Zadanie domowe – CW5

Joanna Klimek

30.11.2021 r.

Program został napisany w języku **Python** z wykorzystaniem biblioteki **graphviz** do tworzenia grafów.

```
from graphviz import Digraph
```

Kod programu dołączony jest w pliku ***cw5_Joanna_Klimek.py***

ZADANIE:

Dane są:

- Alfabet A, w którym każda litera oznacza akcje,
- Zestaw transakcji na zmiennych

Słowo w oznaczające przykładowe wykonanie sekwencji akcji.

Napisz program, który:

1. Wyznacza relacje zależności D

```
# funkcja wyznaczająca relację zależności na podstawie alfabetu oraz zestawu transakcji
def dependent_relations(alphabet, T):
    D = set()
    for a in sorted(alphabet):
        for b in sorted(alphabet):
            if a == b and (a, b) not in D:
                D.add((a, b))
            else:
                a_transaction = T.get(a)
                a_transaction_splited = a_transaction.split("=")
                a_transaction_left, a_transaction_right = a_transaction_splited[0], a_transaction_splited[1]

                b_transaction = T.get(b)
                b_transaction_splited = b_transaction.split("=")
                b_transaction_left, b_transaction_right = b_transaction_splited[0], b_transaction_splited[1]

                if a_transaction_left in b_transaction_right or b_transaction_left in a_transaction_right and (a, b) not in D:
                    D.add((a, b))
    return D
```

Funkcja dla każdej pary sprawdza, czy jedna transakcja nie wykorzystuje zmiennej, którą druga transakcja modyfikuje – jeśli tak, to są zależne. Każda transakcja jest również zależna od samej siebie.

2. Wyznacza relacje niezależności I

```python

# funkcja wyznaczająca relację niezależności na podstawie alfabetu oraz relacji zależności
def independent_relations(alphabet, D):
    I = set()
    for a in sorted(alphabet):
        for b in sorted(alphabet):
            if (a, b) not in D and (a, b) not in I:
                I.add((a, b))
    return I
```

Jeżeli para transakcji jest zależna to nie jest niezależna. Program sprawdza czy dana para znajduje się już w zbiorze relacji zależności.

3. Wyznacza postać normalną Foaty FNF([w]) śladu [w]

```python
# funkcja zwracająca ślad słowa 'w' w postaci stringa
def trace(word):
    result = '<'

    for i in range(len(word) - 1):
        result += word[i]
        result += ', '

    result += word[len(word)-1]
    result += '>'
    return result

# funckja tworząca stosy dla każdej litery alfabetu
# algorytm dodaje na stos literę oraz '*' na stosach liter, które są zależne od badanej litery
def create_stacks(alphabet, D, word):
    stacks = {sign: [] for sign in alphabet}
    index = len(word)

    for sign in word[::-1]:
        for letter in alphabet:
            if letter == sign:
                stacks[sign].append((sign, str(index)))
                index -= 1
            elif (sign, letter) in D:
                stacks[letter].append('*')

    return stacks
```

```
59
60  # funkcja zwracajaca litery ze stosów, wykorzystywana do tworzenia klas foaty
61  def pop_letters_from_stack(stacks, alphabet):
62      poped_letters = []
63      for letter in sorted(alphabet):
64
65          if stacks[letter]:
66              poped_letter = stacks[letter].pop(len(stacks[letter]) - 1)
67
68              if poped_letter != '*':
69                  poped_letters.append(poped_letter)
70
71      poped_letters.sort()
72      return poped_letters
73
74  # funkcja zamieniajaca postać normalna foaty na string
75  def FNF_to_string(letter_mapping):
76      fnf = ''
77      for foata_class in letter_mapping:
78          if len(foata_class) > 0:
79              fnf += '('
80              for sign in foata_class:
81                  fnf += sign[0]
82              fnf += ')'
83      return fnf
84
```

```
85  # funkcja korygujaca wynik algorytmu
86  #   jeżeli którakolwiek z liter nowej potencjalnej klasy fnf (current_class) jest zależna od
87  #   elementów klasy poprzednio dodanej do fnf, to nie możemy połączyć tych klas
88  # jeżeli jednak wszystkie elementy są niezależne to łączymy te klasy
89  def correction(fnf, current_class, D):
90      index = len(fnf) - 1
91
92      for current_letter in current_class:
93          for letter in fnf[index]:
94              if (current_letter[0], letter[0]) in D:
95                  fnf.append(current_class)
96                  return fnf
97
98      for letter in current_class:
99          fnf[index].append(letter)
100     fnf[index].sort()
```

```python
# funkcja zwracająca postać normalną foaty po podstawie algorytmu ze stosami liter
def FNF(alphabet, D, word):
    stacks = create_stacks(alphabet, D, word)
    max_length = max([len(value) for value in stacks.values()])
    fnf = []

    for i in range(max_length):
        letters = pop_letters_from_stack(stacks, alphabet)

        if letters:
            if len(fnf) > 0:
                correction(fnf, letters, D)
            else:
                fnf.append(letters)
    return fnf
```

Do wyznaczenia FNF program wykorzystuje algorytm, w którym tworzone są osobne stosy dla każdej litery w słowie, a następnie kolejno dla każdej litery wstawia ją na stos oraz wstawia znak ‘*’ na stosy liter, które są od niej zależne. Przy odpowiednim ściąganiu elementów ze stosów na danej wysokości otrzymujemy kolejne klasy Foaty. Algorytm jednak nie jest idealny dlatego program wprowadza korektę – jeżeli, żadna z liter nowo utworzonej klasy nie jest zależna od liter w poprzednio utworzonej klasie, to można te klasy połączyć w jedną. W celu wypisania na konsolę dodana jest funkcja konwertująca FNF na stringa.

4. Rysuje graf zależności w postaci minimalnej dla słowa w

```python
# funkcja rysujaca graf na podstawie postaci normalnej foaty, relacji zależności i słowa w
# do grafu dodajemy tylko krawedzie łączące elementy zależne, należące do różnych klas foaty
# oraz nie ma jeszcze między nimi połączenia
def graph(fnf, D, word):
    dot_graph = Digraph()
    multi_edges = []  # bezpośrednie i pośrednie połączenia pomiędzy wierzchołkami
    edges = []  # bezpośrednie połaczenia pomiędzy wierzchołkami

    for i in range(len(fnf) - 2, -1, -1):
        for u in fnf[i]:
            for j in range(i + 1, len(fnf)):
                for v in fnf[j]:
                    if (u[0], v[0]) in D and (u, v) not in multi_edges:

                        edges.append((u, v))
                        dot_graph.edge(u[1], v[1])

                        multi_edges.append((u, v))
                        for e in multi_edges:
                            if e[0] == v:
                                multi_edges.append((u, e[1]))

    for i, letter in enumerate(word):
        dot_graph.node(str(i+1), letter)
    print(dot_graph.source)
```

Funkcja rysuje graf z wykorzystaniem biblioteki **graphviz,** na podstawie FNF relacji zależności i samego słowa.

Kod uruchomieniowy:

```python
# funkcja uruchamiająca analizę danego przykładu
def run(T, alphabet, word):
    D = dependent_relations(alphabet, T)
    I = independent_relations(alphabet, D)
    t = trace(word)
    letter_mapping = FNF(alphabet, D, word)
    fnf_string = FNF_to_string(letter_mapping)

    print("\n=========================================================================================\n")
    print("Tranzakcje: ", T)
    print("Alfabet A =", sorted(alphabet))
    print("Słowo w =", word)
    print("Relacja zależności D =", sorted(D, key=lambda x: x[0]))
    print("Relacja niezależności I =", sorted(I, key=lambda x: x[0]))
    print("Ślad słowa [w] =", t)
    print("Postać normalna Foaty FNF([w]) =", fnf_string)
    print("Graf zależności:")
    graph(letter_mapping, D, word)

```

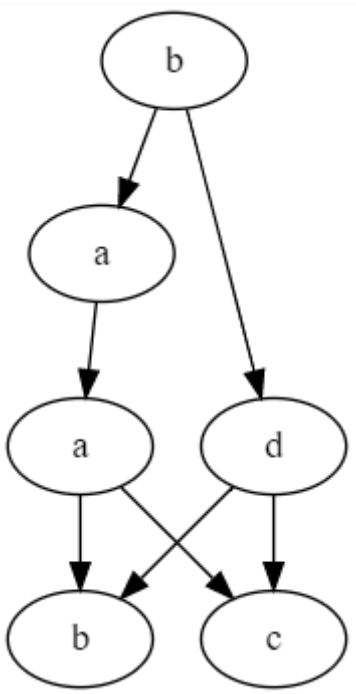
Przykładowe dane:

```
# Przyklad 1
transactions = {
    "a": "x=x+y",
    "b": "y=y+2z",
    "c": "x=3x+z",
    "d": "z=y-z"
}
A = {'a', 'b', 'c', 'd'}
w = 'baadcb'
run(transactions, A, w)


# Przyklad 2
transactions = {
    "a": "x=x+1",
    "b": "y=y+2z",
    "c": "x=3x+z",
    "d": "w=w+v",
    "e": "z=y-z",
    "f": "v=x+v"
}
A = {'a', 'b', 'c', 'd', 'e', 'f'}
w = 'acdcfbbe'
run(transactions, A, w)

```

Wyniki dla przykładu 1:

```
================================================================================

Tranzakcje:  {'a': 'x=x+y', 'b': 'y=y+2z', 'c': 'x=3x+z', 'd': 'z=y-z'}
Alfabet A = ['a', 'b', 'c', 'd']
Słowo w = baadcb
Relacja zależności D = [('a', 'a'), ('a', 'b'), ('a', 'c'), ('b', 'a'), ('b', 'b'), ('b', 'd'), ('c', 'c'), ('c', 'a'), ('c', 'd'), ('d', 'c'), ('d', 'd'), ('d', 'b')]
Relacja niezależności I = [('a', 'd'), ('b', 'c'), ('c', 'b'), ('d', 'a')]
Ślad słowa [w] = <b, a, a, d, c, b>
Postać normalna Foaty FNF([w]) = (b)(ad)(a)(bc)
Graf zależności:
digraph {
    3 -> 6
    3 -> 5
    2 -> 3
    4 -> 6
    4 -> 5
    1 -> 2
    1 -> 4
    1 [label=b]
    2 [label=a]
    3 [label=a]
    4 [label=d]
    5 [label=c]
    6 [label=b]
}
```

Otrzymany graf:



Wyniki dla przykładu 2:

```
Tranzakcje:  {'a': 'x=x+1', 'b': 'y=y+2z', 'c': 'x=3x+z', 'd': 'w=w+v', 'e': 'z=y-z', 'f': 'v=x+v'}
Alfabet A = ['a', 'b', 'c', 'd', 'e', 'f']
Słowo w = acdcfbbe
Relacja zależności D = [('a', 'a'), ('a', 'f'), ('a', 'c'), ('b', 'e'), ('b', 'b'), ('c', 'c'), ('c', 'a'), ('c', 'f'), ('c', 'e'), ('d', 'd'), ('d', 'f'), ('e', 'e'), ('e',
 'b'), ('e', 'c'), ('f', 'd'), ('f', 'c'), ('f', 'a'), ('f', 'f')]
Relacja niezależności I = [('a', 'd'), ('a', 'e'), ('a', 'b'), ('b', 'a'), ('b', 'f'), ('b', 'c'), ('b', 'd'), ('c', 'b'), ('c', 'd'), ('d', 'c'), ('d', 'a'), ('d', 'e'),
 ('d', 'b'), ('e', 'f'), ('e', 'd'), ('e', 'a'), ('f', 'e'), ('f', 'b')]
Ślad słowa [w] = <a, c, d, c, f, b, b, e>
Postać normalna Foaty FNF([w]) = (abd)(bc)(c)(ef)
Graf zależności:
digraph {
    4 -> 8
    4 -> 5
    7 -> 8
    2 -> 4
    1 -> 2
    6 -> 7
    3 -> 5
    1 [label=a]
    2 [label=c]
    3 [label=d]
    4 [label=c]
    5 [label=f]
    6 [label=b]
    7 [label=b]
    8 [label=e]
}
```

Otrzymany graf: