Teoria współbieżności - zadanie domowe nr 2: Zagadnienia teorii śladów

Wojciech Michaluk

06.12.2024

1 Opis zadania

Mamy jako dane wejściowe:

- Alfabet A, w którym każda litera oznacza akcję,
- Zestaw transakcji na zmiennych,
- Słowo w, które oznacza przykładowe wykonanie sekwencji akcji.

Należy napisać w dowolnym języku program, który:

1. Wyznacza relację zależności D i relację niezależności I,
2. Wyznacza postać normalną Foaty FNF($[w]$) śladu $[w]$,
3. Rysuje graf zależności w postaci minimalnej dla słowa w.

2 Opis programu

Program jest napisany w języku **Python** - w całości, zarówno część przeprowadzająca obliczenia, jak i rysowanie. Do narysowania grafu korzystam z bibliotek Pythona: *networkx* oraz *matplotlib*.

W przesłanej paczce zip znajdują się podkatalogi *examples1* i *examples2* (które zawierają przykładowe pliki wejściowe i zapisane grafy zależności - zostaną one opisane później) oraz kod programu, który jest podzielony na 2 pliki:

- *utils.py* - ten plik zawiera definicje funkcji pomocniczych używanych w głównej części programu,
- *program.py* - główna część programu, przedstawiająca kolejne etapy obliczeń i na końcu rysowanie grafu, wykorzystuje funkcje zdefiniowane w *utils.py*.

Każdy z tych plików jest okraszony komentarzami (przy funkcji / fragmencie kodu), które wskazują, za co jest odpowiedzialny ten kod, natomiast tutaj dodaję dodatkowe objaśnienie. Funkcje w pliku *utils.py*:

- `get_transactions(trans_file)` - ta funkcja odpowiada za pobranie transakcji z pliku, który zawiera transakcje (trans_file),
- `get_alphabet(alpha_file)` - ta funkcja odpowiada za pobranie alfabetu z pliku, który zawiera alfabet (alpha_file),
- `check_correctness(transactions, alphabet, word)` - ta funkcja odpowiada za sprawdzenie spójności oznaczeń akcji pomiędzy transakcjami (transactions), alfabetem (alphabet) a słowem (word),
- `get_dependent_transactions(transactions)` - ta funkcja odpowiada za wyznaczenie relacji zależności na podstawie transakcji (transactions),

- `get_dependencies(dependent, alphabet)` - ta funkcja odpowiada za wyznaczenie dla każdej transakcji (z alphabet) akcji od niej zależnych na podstawie relacji zależności (dependent),
- `agh(dependencies, word, alphabet)` - ta funkcja odpowiada za wyznaczenie postaci normalnej Foaty metodą "górniczo-hutniczą" - potrzebuje do tego informacji o transakcjach zależnych (dependencies), słowa (word) oraz alfabetu akcji (alphabet),
- `get_minerals(mining_shafts)` - ta funkcja odpowiada za odkrycie minerałów z góry szybów kopalnianych (mining_shafts) w powyższej metodzie,
- `get_graph_edges(dependencies, word)` - ta funkcja odpowiada za wyznaczenie postaci minimalnej grafu na podstawie zależności między transakcjami (dependencies) oraz słowa (word).
- `prepare_graph_to_draw(word, graph_edges)` - ta funkcja odpowiada za stworzenie powyższego grafu na podstawie słowa (word), aby ustalić etykiety wierzchołków oraz na podstawie krawędzi (graph_edges) tworzy krawędzie między wierzchołkami. Ustala również pozycje wierzchołków do narysowania.

Kolejne etapy działania programu w pliku *program.py*:

1. Sprawdzenie poprawności uruchomienia programu - czy przekazano odpowiednią liczbę argumentów,
2. Próba otwarcia plików podanych jako argument - zgłoszenie błędu i przerwanie programu w razie niepowodzenia,
3. Przetworzenie danych z plików,
4. Sprawdzenie spójności danych z plików (transakcje, alfabet) i ze słowem,
5. Wyznaczenie relacji zależności i niezależności,
6. Wyznaczenie postaci normalnej Foaty,
7. Wyznaczenie grafu w postaci dot - co prawda nie korzystam z Graphviz tylko z networkx, ale i tak jest to później częściowo przydatne,
8. Stworzenie grafu, korzystając z biblioteki networkx i narysowanie go.

2.1 Szczegóły techniczne

Przyjąłem następujące założenia:

- Program jest uruchamiany z 3 argumentami, kolejno:
  1. plik zawierający transakcje w formacie **dokładnie** takim, jak jest podany w poleceniu zadania - czyli np.
     ```
     (a) x := x+y
     (b) y := y+2z
     (c) x := 3x+z
     (d) z := y-z
     ```
  2. plik zawierający alfabet transakcji - format również identyczny, jak podany w poleceniu, np.
     `A = {a,b,c,d}`
  3. łańcuch znaków (tu już nie plik), który reprezentuje słowo oznaczające przykładowe wykonanie sekwencji akcji, np. `baadcb`.
- Wyniki działania programu są przedstawiono następująco:
  1. Relacja zależności D oraz relacja niezależności I jest wypisywana na terminalu, w formacie **identycznym** jak podano w treści zadania - przykładowo:
     ```
     D = {(a,a),(a,b),(a,c),(b,a),(b,b),(b,d),(c,a),(c,c),(c,d),(d,b),(d,c),(d,d)}
     I = {(a,d),(b,c),(c,b),(d,a)}
     ```

2. Postać normalna Foaty również jest wypisywana na terminalu, także w postaci tej samej, co podana w treści zadania, przykładowo:
   `FNF([w]) = (b)(ad)(a)(bc)`
3. Mimo że nie potrzebuję go wprost (w mojej implementacji rysowania), to wypisuję także na terminalu graf w postaci dot, np.
```
digraph g{
1 -> 2
2 -> 3
1 -> 4
3 -> 5
4 -> 5
3 -> 6
4 -> 6
1[label=b]
2[label=a]
3[label=a]
4[label=d]
5[label=c]
6[label=b]
}
```
4. Narysowany graf (z podpisanymi odpowiednio wierzchołkami i zaznaczonymi krawędziami) pojawia się w "wyskakującym" okienku matplotliba. Z poziomu tego okienka można np. zapisać rysunek. Tutaj go nie umieszczam, zostanie on uwzględniony w następnej sekcji przy pokazaniu wyników uruchomienia programu dla danych testowych.

Przykładowe uruchomienie programu, przy założeniu, że pliki *transactions.txt* oraz *alphabet.txt* znajdują się w tym samym katalogu, co program i zawierają odpowiednio transakcje i alfabet w odpowiednim formacie:

```
py .\program.py transactions.txt alphabet.txt baadcb
```

Żeby móc uruchomić program, trzeba mieć zainstalowaną jedną z nowszych wersji Pythona (np. 3.11) oraz wspomniane na początku rozdziału biblioteki.

3 Wyniki działania dla przykładowych danych

Poniżej przedstawiam pełne wyniki działania programu (wraz z komunikatami porządkującymi) dla przykładowych danych, uwzględnionych w poleceniu. Polecenia uruchomienia programu są spójne ze strukturą plików w przesłanej paczce zip.

3.1 Dane testowe 1

Uruchomienie programu:

```
py .\program.py example1/transactions.txt example1/alphabet.txt baadcb
```

Zawartość plików przekazanych jako argument:

- Plik `example1/transactions.txt`:

```
(a) x := x+y
(b) y := y+2z
(c) x := 3x+z
(d) z := y-z
```

- Plik `example1/alphabet.txt`:

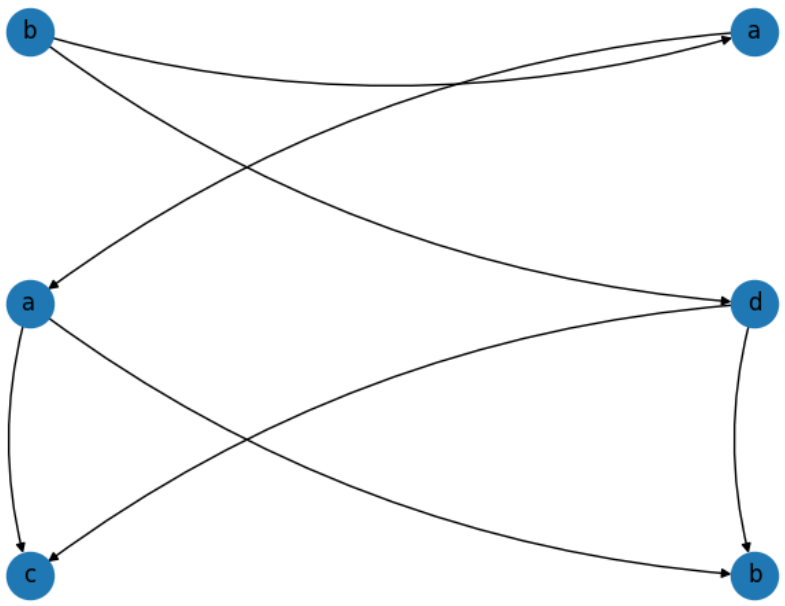
```
A = {a,b,c,d}
```

Wynik działania programu:

- Na terminalu:

```
Relacja zależności:
D = {(a,a),(a,b),(a,c),(b,a),(b,b),(b,d),(c,a),(c,c),(c,d),(d,b),(d,c),(d,d)}

Relacja niezależności:
I = {(a,d),(b,c),(c,b),(d,a)}

Postać normalna Foaty:
FNF([w]) = (b)(ad)(a)(bc)

Graf w postaci dot:
digraph g{
1 -> 2
2 -> 3
1 -> 4
3 -> 5
4 -> 5
3 -> 6
4 -> 6
1[label=b]
2[label=a]
3[label=a]
4[label=d]
5[label=c]
6[label=b]
}
```

- Rysunek grafu w okienku matplotliba:



Rys. 1: Graf zależności w postaci minimalnej dla słowa *baadcb*

3.2 Dane testowe 2

Uruchomienie programu:

```
py .\program.py example2/transactions.txt example2/alphabet.txt acdcfbbe
```

Zawartość plików przekazanych jako argument:

- Plik `example2/transactions.txt`:

```
(a) x := x+1
(b) y := y+2z
(c) x := 3x+z
(d) w := w+v
(e) z := y-z
(f) v := x+v
```

- Plik `example2/alphabet.txt`:

```
A = {a,b,c,d,e,f}
```

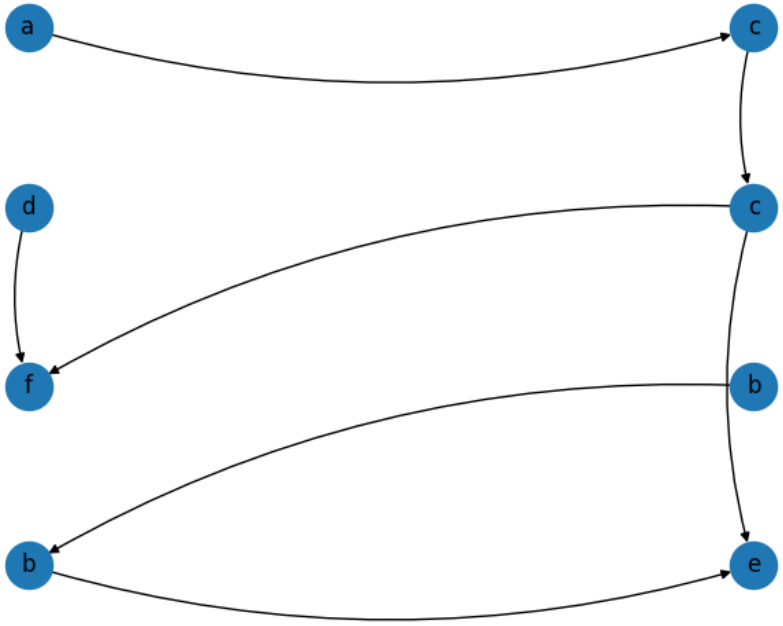
Wynik działania programu:

- Na terminalu*:

```
Relacja zależności:
D = {(a,a),(a,c),(a,f),(b,b),(b,e),(c,a),(c,c),(c,e),(c,f),(d,d),
     (d,f),(e,b),(e,c),(e,e),(f,a),(f,c),(f,d),(f,f)}

Relacja niezależności:
I = {(a,b),(a,d),(a,e),(b,a),(b,c),(b,d),(b,f),(c,b),(c,d),(d,a),
     (d,b),(d,c),(d,e),(e,a),(e,d),(e,f),(f,b),(f,e)}

Postać normalna Foaty:
FNF([w]) = (abd)(bc)(c)(ef)

Graf w postaci dot:
digraph g{
1 -> 2
2 -> 4
3 -> 5
4 -> 5
6 -> 7
4 -> 8
7 -> 8
1[label=a]
2[label=c]
3[label=d]
4[label=c]
5[label=f]
6[label=b]
7[label=b]
8[label=e]
}
```

*Na terminalu normalnie zbiór D oraz zbiór I są wypisane w jednej linii, tutaj podzieliłem je w celu ładniejszego ich pokazania, bo nie mieściły się.

- Rysunek grafu w okienku matplotliba:



Rys. 2: Graf zależności w postaci minimalnej dla słowa *acdcfbbe*