TEORIA WSPÓŁBIEŻNOŚCI

Zadanie domowe 2

Aleksandra Smela

Dostarczony program hw2.py na podstawie zadanego alfabetu, zestawu transakcji na zmiennych i słowa w oznaczającego sekwencję akcji wyznacza relację zależności D, relację niezależności I, postać normalną FNF(w) oraz rysuje graf zależności w postaci minimalnej.

1. Dane wejściowe

Na wejściu program przyjmuje jeden argument będący nazwą pliku w formacie txt. Jeżeli argument nie zostanie przekazany to próbuje otworzyć plik o nazwie *example.txt*.

Poprawny plik tekstowy powinien składać się z:

I. alfabet (pierwsza linia)
II. transakcje na zmiennych, każda w nowej linii w postaci:
(*litera alfabetu*) *transakcja*
(kolejne n linii, gdzie n to długość alfabetu)
III. słowo w (ostatnia linia)

Przykład poprawnego pliku:

abcd
(a) x=x+y
(b) y=y+2z
(c) x=3x+z
(d) z=y-z
baadcb

2. Kluczowe funkcje

2.1. find_dependencies

wejście:

- alphabet – alfabet wczytany z pliku
- read – słownik:

litera alfabetu transakcji x*: *zmienne które są czytane w transakcji x

- written – słownik:

litera alfabetu transakcji x*: *zmienne które są zapisywane w transakcji x

wyjście:

- dependencies - relacja zależności D w formie krotki dwuelementowych krotek transakcji zależnych
- independencies - relacja zależności D w formie krotki dwuelementowych krotek transakcji niezależnych

uwaga! W krotkach dependencies i independencies zakładam, że oczywistym jest, że jeżeli (x, y) są (nie)zależne to również (y, x) są niezależne. Zatem jeżeli istnieje wpis (x, y), to nie istnieje wpis (y, x).

działanie:

Transakcje są zależne jeżeli obie zapisują do tej samej zmiennej lub jedna zapisuje, a druga czyta z tej samej zmiennej. Funkcja sprawdza zależność dla każdej pary transakcji.

2.2. get_fnf

wejście:

- word – słowo w
- dep_graph – graf zależności (np. zwrócony przez funkcję build_dependencies_graph [patrz: punkt 3.2])

wyjście:

- fnf – krotka krotek będących kolejnymi grupami w postaci normalnej Foaty.

działanie:

Każda z transakcji w słowie przypisywana jest do grupy za pomocą algorytmu BFS z modyfikacją polegającą na wyeliminowaniu warunku na jednokrotne odwiedzanie wierzchołków. Następnie otrzymane dane są odpowiednio agregowane do ostatecznie zwracanej formy.

2.3. get_min_dep_graph

wejście:

- dep_graph – jak wyżej [patrz: punkt 2.2]

wyjście:

- min_dep_graph –graf zależności w postaci minimalnej.

działanie:

Dla każdej krawędzi (x, y) w grafie sprawdza czy bez niej wciąż istnieje ścieżka z x do y [za pomocą funkcji z punktu 3.3] i jeżeli tak to usuwa tę krawędź z grafu.

2.4. render_min_dep_graph

wejście:

- min_dep_graph – jak wyżej [patrz: punkt 2.3]

wyjście:

- graph –graf zależności w postaci minimalnej jako klasa graphviz.Digraph.

działanie:

Konwertuje graf zależności w postaci minimalnej na obiekt klasy graphviz.Digraph i zapisuje wynik do plików 'min_dep_graph.gv' oraz 'min_dep_graph.gv.pdf'.

3. Funkcje pomocnicze

3.1. parse_file

wyjście:

- alphabet, read, written [patrz: punkt 2.1]
- word [patrz: punkt 2.2]

3.2. build_dependencies_graph

wejście:

- read, written [patrz: punkt 2.1]
- word [patrz: punkt 2.2]

wyjście:

- dep_graph – jak wyżej [patrz: punkt 2.2]

działanie:

Tworzy graf skierowany w postaci tablicy krawędzi, w której pod indeksem i znajduje się zbiór numerów wierzchołków dla i-tego wierzchołka. i-ty wierzchołek reprezentuje i-tą transakcję z sekwencji reprezentowanej przez słowo w.

Sprawdza czy dla każdej pary transakcji ze słowa w zachodzi zależność i jeżeli tak to dodaje krawędź skierowaną od transakcji występującej wcześniej w słowie do drugiej transakcji.

3.3. does_path_exist

wejście:

- dep_graph – jak wyżej [patrz: punkt 2.2]
- start, end

wyjście:

- True/False

działanie:

Stosując BFS sprawdza czy w grafie istnieje ścieżka z wierzchołka start do wierzchołka stop.

4. Wyjście programu

Program na wyjściu wypisuje relację zależności D, relację niezależności I, postać normalną FNF(w) oraz tworzy pliki 'min_dep_graph.gv' oraz 'min_dep_graph.gv.pdf', które są odpowiednio tekstową i graficzną reprezentacją grafu zależności w postaci minimalnej

5. Wyniki działania dla przykładowych danych

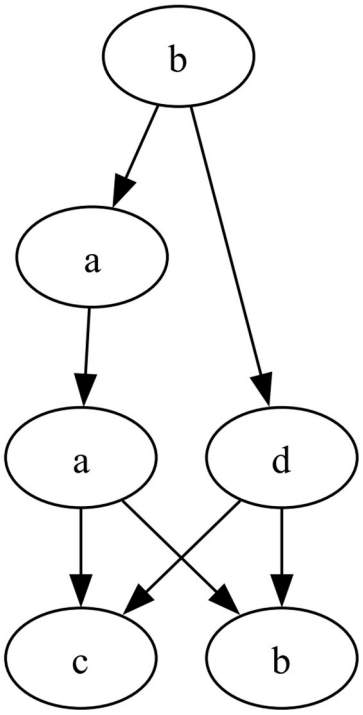
5.1. Dane testowe 1

Input:
abcd
(a) x=x+y
(b) y=y+2z
(c) x=3x+z
(d) z=y-z
baadcb

Output:
D =
{('c', 'c'), ('a', 'a'), ('a', 'b'), ('b', 'd'), ('a', 'c'), ('b', 'b'), ('c', 'd'), ('d', 'd')}

I =
{('b', 'c'), ('a', 'd')}

FNF('baadcb') =
(('b',), ('a', 'd'), ('a',), ('b', 'c'))



5.2. Dane testowe 2

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

Output:
D =
{('b', 'b'), ('a', 'c'), ('d', 'd'), ('e', 'e'), ('f', 'f'), ('c', 'e'), ('c', 'c'), ('d', 'f'), ('b', 'e'), ('c', 'f'), ('a', 'a'), ('a', 'f')}

I =
{('b', 'f'), ('a', 'b'), ('c', 'd'), ('a', 'd'), ('b', 'd'), ('d', 'e'), ('e', 'f'), ('a', 'e'), ('b', 'c')}

FNF('acdcfbbe') =
(('a',), ('c',), ('c',), ('b', 'b', 'd', 'e', 'f'))