Teoria Współbieżności

Zadanie domowe 1

Przemysław Popowski

08.11.2024 r.

I. Wykorzystane środowisko

Kod programu został napisany w języku Python 3.10

Program Graphviz dostępny pod linkiem:
- https://graphviz.org/download/

II. Instrukcja

0. Kod wystarczy uruchomić dowolnym interpreterem języka *Python 3.10*.
1. W terminalu wyskoczy komunikat: „*Enter the name of an input file”*. Należy podać nazwę pliku z danymi do zadania. Plik musi znajdować się razem z programem „main.py”. Dla ułatwienia, podane są 4 pliki testowe – *case0.txt, case1.txt, case2.txt oraz case3.txt*.
2. Następnie program wyznaczy relację zależności D, niezależności I, postać normalną foaty FNF śladu w, a także graf zależności w postaci minimalnej dla słowa w.
3. Graf wypisany jest w terminalu w postaci tekstu oraz zapisany jest w pliku *„[nazwa_pliku_txt].dot*”. Aby uzyskać graficzną reprezentację, potrzebny jest zainstalowany program „Graphviz” oraz musi dodany być do zmiennych środowiskowych PATH. Następnie wpisujemy następującą linijkę w PowerShell bądź CMD: {dot -Tpng [nazwa_pliku_txt].dot -o [nazwa_pliku_txt].png}. Dzięki temu w folderze z programem, pojawi się zdjęcie .png będące graficzną reprezentacją naszego grafu.

III. Opis programu z komentarzami

a. Zmienne wykorzystywane do poprawnego działania programu

```
# List representing the English alphabet, with indices corresponding to each letter:
# 'a' = 0, 'b' = 1, ..., 'z' = 25
alphabet = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
            'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']



# Matrix (26x26) initialized with zeros to represent dependencies between letters.
# word_matrix[i][j] = 1 is a dependency from letter i to letter j.
word_matrix = [[0 for _ in range(26)] for _ in range(26)]

# List to map a character's dependency position in the alphabet.
# word_list[i] stores the index of the character that letter alphabet[i] depends on, or 0 if
none.
word_list = [0 for _ in range(26)]

# Variable to store a sequence of actions (word). This will represent the order in a dependency
graph.
word = ''

# Counter to track the number of unique actions (unique letters).
counter = 0

# List to store pairs of dependent letters.
# Each item in `dependency_list` is a tuple (letter1, letter2) indicating dependency.
dependency_list = []

# List to store pairs of independent letters.
# Each item in `independency_list` is a tuple (letter1, letter2) indicating independence.
independency_list = []
```

b. Funkcja importująca dane z pliku i przygotowująca je pod poprawne działanie programu

```
# Function to import data from a .txt file and make it useful for other functions.
def import_data(file_name):
    # Declare global variables to use in this function
    global word, counter

    # Open the file for reading
    file = open(file_name, „r”)

    # Process each line in the file
    for line in file:

        # Check if the line defines an action. (for example: x := x + y)
        # Define a dependency or independency pair
        if line[0] == '(':
            # Calculate index positions for characters in the dependency pair
            active_int_char = ord(line[1]) - 97  # Convert first char to an index (assuming 'a'
is index 0)
            int_char = ord(line[4]) - 97  # Convert second char to an index

            # Store the dependency relationship in `word_list`
            word_list[active_int_char] = int_char

            #  umer  sany characters listed after the dependency pair
            for char in line[8:]:
                # If the character is in the alphabet, mark it as a dependency in word_matrix
                if char in alphabet:
                    inside_int_char = ord(char) - 97
                    word_matrix[active_int_char][inside_int_char] = 1

        elif line[0] == 'A':
            # Determine the  umer of instructions and set it to counter
            counter = ord(line[len(line) - 3]) - 96

        # Extract the word with sequence of actions and save it to variable word
        elif line[0] == 'w':
            word = line[4:]

    # Close the file after reading
    file.close()
```

c. Funkcja wyznaczająca relacje zależności D oraz niezależności I

```
# Function to define dependencies and independencies based on `word_matrix`
def dependencies():
    global counter  # Use the global counter for loop bounds

    # Loop through each pair of characters in the alphabet
    for i in range(counter):
        for j in range(counter):
            # Add self-dependencies (for example: (a, a))
            if i == j:
                dependency_list.append((alphabet[i], alphabet[j]))

            # Check if a dependency exists between different characters
            elif word_matrix[i][word_list[j]] == 1 or word_matrix[j][word_list[i]] == 1:
                dependency_list.append((alphabet[i], alphabet[j]))

            # If no dependency is found, they are independent, so add to independency_list
            else:
                independency_list.append((alphabet[i], alphabet[j]))
```

d. Funkcja obliczająca postać normalną Foaty – FNF

```
# Calculate the Foata Normal Form (FNF) of the given word based on dependencies
def calculate_fnf(word, dependencies):
    # Step 1: Initialize the dependency tracker
    dependency_tracker = [[] for _ in range(26)]

    # Step 2: Populate the dependency tracker based on dependencies in the input word
    for i in range(len(word) - 1, -1, -1):
        event = word[i]
        event_index = ord(event) - 97

        # Mark dependencies in `dependency_tracker`
        for dependent, dependency in dependencies:
            if dependent == event and dependency != event:
                dependency_tracker[ord(dependency) - 97].append('-')

        # Append the event itself to track its position
        dependency_tracker[event_index].append(event)

    # Step 3: Construct the Foata Normal Form (FNF)
    fnf_result = []
    while not all(not sublist for sublist in dependency_tracker): # Check if all lists
within the dependency_tracker are empty
        current_str = ""

        # Gather the next set of independent events
        for i in range(len(dependency_tracker)):
            if dependency_tracker[i] and dependency_tracker[i][-1] != '-':
                current_str += dependency_tracker[i].pop()

        # Remove dependencies of events found in current_str
        for event in current_str:
            for dependent, dependency in dependencies:
                if dependent == event and dependency != event and
dependency_tracker[ord(dependency) - 97]:
                    dependency_tracker[ord(dependency) - 97].pop()

        # Append the concurrent events to the final FNF result
        fnf_result.append(current_str)

    return fnf_result
```

e. Funkcja tworząca graf oraz reprezentująca go w formacie .dot, czytelnym dla użytkownika i rozróżnialnym dla aplikacji Graphviz

```
# Function to build a directed graph in DOT format for visualizing dependencies in a
word sequence
def build_graph(word, dependencies):
    edges = []  # List to store the directed edges in the graph
    nodes = []  # List to store nodes with their labels
    min_indices = []  # Track the minimum indices of dependencies for current nodes

    # Step 1: Generate initial nodes and edges based on dependencies
    for i in range(len(word) - 1, -1, -1):
        min_indices.append(i)

        # Check dependencies with nodes processed so far
        for x in min_indices:
            if x != i and (word[i], word[x]) in dependencies:
                # Create a directed edge if a dependency exists
                edges.append((i + 1, x + 1))

        # Record the node with its character label
        nodes.append((i + 1, word[i]))

    # Step 2: Simplify edges by removing transitive edges
    for edge_a in edges[:]:
        for edge_b in edges:
            if edge_a != edge_b:
                # Remove transitive edges
                if edge_a[1] == edge_b[0] and (edge_a[0], edge_b[1]) in edges:
                    edges.remove((edge_a[0], edge_b[1]))
                if edge_a[0] == edge_b[1] and (edge_b[0], edge_a[1]) in edges:
                    edges.remove((edge_b[0], edge_a[1]))

    # Step 3: Format the graph in DOT format
    graph = ''
    # Add edges in sorted order by the source node
    edges.sort(key=lambda edge: edge[0])
    for src, dest in edges:
        graph += "  " + str(src) + " -> " + str(dest) + "\n"

    # Add nodes with labels
    nodes.sort(key=lambda node: node[0])
    for index, label in nodes:
        graph += "  " + str(index) + " [label=" + str(label) + "]\n"

    # Return ready graph
    return "digraph g {\n" + graph + "}"
```

f. Funkcja odpowiedzialna za uruchomienie każdej z metod, wczytaniu pliku o podanej przez użytkownika nazwie oraz wypisująca czytelnie wyniki użytkownikowi

```
def run():
    print("Enter the name of an input file.")
    print("for example: case0.txt, case1.txt, case2.txt, case3.txt.")
    file_name = input()
    import_data(file_name)
    dependencies()
    fnf = calculate_fnf(word, dependency_list)
    graph = build_graph(word, dependency_list)
    print("==============")
    print("RESULTS:")
    print("==============")
    print("Dependency relation D:")
    print("D = {" + ",".join("(" + a + "," + b + ")" for a, b in dependency_list) + "}")
    print("==============")
    print("Independency relation I:")
    print("I = {" + ",".join("(" + a + "," + b + ")" for a, b in independency_list) + "}")
    print("==============")
    print("Foata Normal Form (FNF):")
    print("FNF([w]) = " + "".join("(" + block + ")" for block in fnf))
    print("==============")
    print("Graph:")
    print(graph)
    print("==============")
    save_file_name = str(file_name[:-4]) + ".dot"
    file = open(save_file_name, "w")
    file.write(graph)
    file.close()
    print("Graph has been saved to a file: " + save_file_name)
    print("==============")

run()
```

IV. Wyniki działania dla przykładowych danych

a. Dane testowe 1 – case0.txt

(a) x := x + y
(b) y := y + 2z
(c) x := 3x + z
(d) z := y - z
A = {a, b, c, d}
w = baadcb

Wyniki:

==========

RESULTS:

==========

Dependency relation D:

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)}

==========

Independency relation I:

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

==========

Foata Normal Form (FNF):

FNF([w]) = (b)(ad)(a)(bc)

==========

Graph:

```
digraph g {
 1 -> 4
 1 -> 2
 2 -> 3
 3 -> 6
 3 -> 5
 4 -> 6
 4 -> 5
 1 [label=b]
 2 [label=a]
 3 [label=a]
 4 [label=d]
 5 [label=c]
```

6 [label=b]

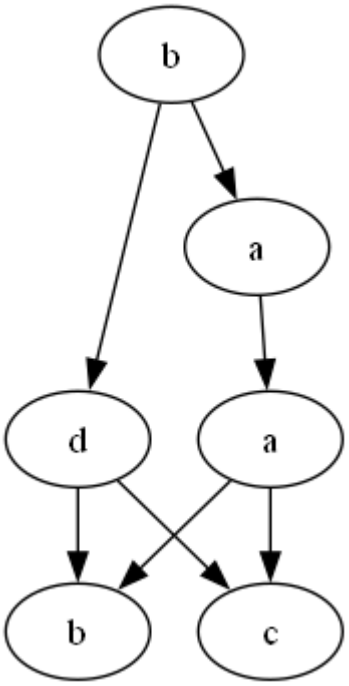
}

==========

Graph has been saved to a file: case0.dot

==========

Po uruchomieniu linijki: *dot -Tpng case0.dot -o case0.png* i sprawdzeniu pliku *case0.png* :



b. Dane testowe 2 – case1.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
A = {a, b, c, d, e, f}
w = acdcfbbe

==========

RESULTS:

==========

Dependency relation D:

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)}

==========

Independency relation I:

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)}

==========

Foata Normal Form (FNF):

FNF([w]) = (abd)(bc)(c)(ef)

==========

Graph:

digraph g {

 1 -> 2

 2 -> 4

 3 -> 5

 4 -> 8

 4 -> 5

 6 -> 7

 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]

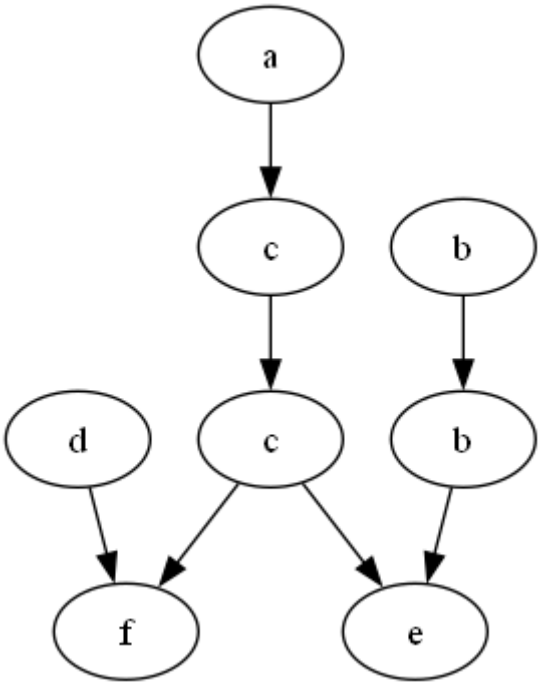
}

==========

Graph has been saved to a file: case1.dot

==========

Po uruchomieniu linijki: *dot -Tpng case1.dot -o case1.png* i sprawdzeniu pliku *case1.png* :



c. *Dane testowe 3 – case2.txt*

(a) x := x + y

(b) y := z - v

*(c) z := v * x*

(d) v := x + 2y

(e) x := 3y + 2x

(f) v := v - 2z

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

w = afaeffbcd

==========

RESULTS:

==========

Dependency relation D:

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

==========

Independency relation I:

I = {(a,f),(e,f),(f,a),(f,e)}

==========

Foata Normal Form (FNF):

FNF([w]) = (af)(af)(ef)(b)(c)(d)

==========

Graph:

digraph g {

1 -> 3

2 -> 5

3 -> 4

4 -> 7

5 -> 6

6 -> 7

7 -> 8

8 -> 9

1 [label=a]

2 [label=f]

3 [label=a]

4 [label=e]

5 [label=f]

6 [label=f]

7 [label=b]

8 [label=c]

9 [label=d]

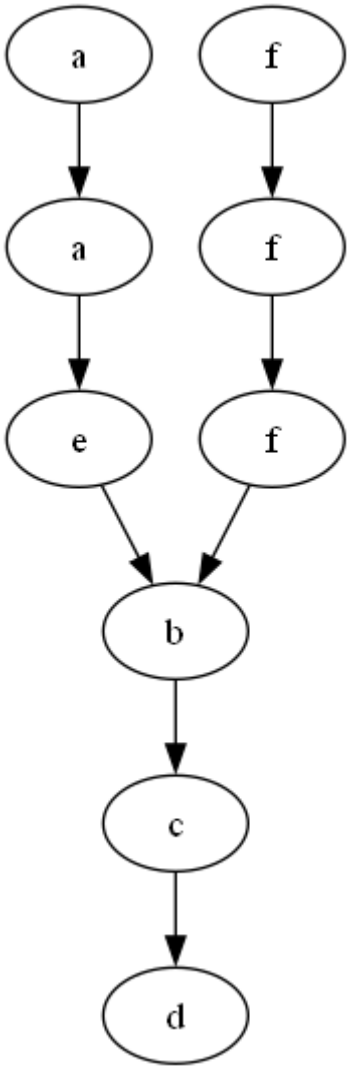
}

==========

Graph has been saved to a file: case2.dot

==========

Po uruchomieniu linijki: *dot -Tpng case2.dot -o case2.png* i sprawdzeniu pliku *case2.png* :



d. Dane testowe 4– case3.txt

(a) x := y + z

(b) y := x + w + y

(c) x := x + y + v

(d) w := v + z

(e) v := x + v + w

(f) z := y + z + v

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

w = acdcfbbe

==========

RESULTS:

==========

Dependency relation D:

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

==========

Independency relation I:

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

==========

Foata Normal Form (FNF):

FNF([w]) = (ad)(cf)(c)(be)(b)

==========

Graph:

digraph g {

1 -> 5

1 -> 2

2 -> 4

3 -> 5

4 -> 8

4 -> 6

5 -> 8

5 -> 6

6 -> 7

1 [label=a]

2 [label=c]

3 [label=d]

4 [label=c]

5 [label=f]

6 [label=b]

7 [label=b]

8 [label=e]

}

==========

Graph has been saved to a file: case3.dot

==========

Po uruchomieniu linijki: *dot -Tpng case3.dot -o case3.png* i sprawdzeniu pliku *case3.png* :