Teoria Współbieżności

Zadanie domowe 1

Wiktor Satora 411502

30.11.2023

Uruchomienie programu:

Program został napisany w Pythonie.
Dane wejściowe w poprawnym formacie należy umieścić w pliku input.txt w tym samym katalogu co wykonywany program. Wyjście programu zostanie wypisane w konsoli oraz do pliku output.txt.
Komenda uruchamiająca program: *python3 fnf.py*

Opis programu

Najpierw program parsuje plik wejściowy input.txt korzystając z poniższych funkcji:

```python
def get_alphabet_chars(line_strip):
    alphabet_chars = []
    for char in line_strip[3:]:
        if char.isalpha():
            alphabet_chars.append(char)
    return alphabet_chars

def parse_file(file_path):
    file_array=[]
    alphabet_array=[]
    with open(file_path, 'r') as file:
        lines = file.readlines()
    for line in lines:
        line_strip = line.strip()
        if(len(line_strip)) == 0:
            continue
        first_char = line_strip[0]
        if first_char == '(':
            alphabet_chars=get_alphabet_chars(line_strip)
            file_array.append({'name':line_strip[1],'left':alphabet_chars[0], 'right':alphabet_chars[1:]})
        elif first_char == 'A':
            alphabet_array=get_alphabet_chars(line_strip)
        elif first_char == 'w':
            word=line_strip[4:]
    return file_array, alphabet_array, word
```

Następnie odbywa się wypełnienie zbiorów D oraz I w oparciu o funkcje:

```python
def fillDandI(data):
    D=[]
    I=[]
    for x in range(len(data)):
        for y in range(x,len(data)):
            if data[x]['left'] in data[y]['right'] or data[y]['left'] in data[x]['right'] or data[x]['left']==data[y]['left']:
                if ((data[x]['name'],data[y]['name']) not in D): D.append((data[x]['name'],data[y]['name']))
                if ((data[y]['name'],data[x]['name']) not in D): D.append((data[y]['name'],data[x]['name']))
            else:
                if ((data[x]['name'],data[y]['name']) not in I): I.append((data[x]['name'],data[y]['name']))
                if ((data[y]['name'],data[x]['name']) not in I): I.append((data[y]['name'],data[x]['name']))
    return D,I
```

Dalej jest tworzony graf korzystając ze zbioru D:

```
graph = [[] for _ in range(len(word))]
for x in range(len(word)):
    for y in range(x+1, len(word)):
        if (word[x], word[y]) in D:
            graph[x].append(y)
```

Optymalizacja grafu (funkcje oraz ich wywołanie):
https://en.wikipedia.org/wiki/Transitive_reduction
Algorytm działający w oparciu o algorytm BFS.

Przykładowo, graf:
[[1, 2, 3, 5], [2, 4, 5], [4, 5], [4, 5], [], []]

Zostanie zredukowany do:
[[1, 3], [2], [4, 5], [4, 5], [], []]

Kod:

```
def decreaseIndegree(e):
    global indeg
    g = indeg[e]
    indeg[e] -= 1
    indegs[g].remove(e)
    indegs[g-1].append(e)

def remove(r, elems):
    global grafo, indeg
    for e in elems:
        if e in graph[r]:
            graph[r].remove(e)
            decreaseIndegree(e)

def removeRoot(r):
    global graph, indeg
    children = graph[r]
    for child in children:
        decreaseIndegree(child)

def bfs(r):
    global graph
    INF = 10000
    dist = n * [INF]
    dist[r] = 0
    queue = [r]
    while queue:
        node = queue.pop()
        remove(r, [i for i in graph[node] if dist[i]==1])
        for child in graph[node]:
            if dist[child] == INF:
                queue.append(child)
            dist[child] = min(dist[child], dist[node]+1)
```

```
n = len(graph)
indeg = n*[0]
for vecs in graph:
    for i in vecs:
        indeg[i] += 1

indegs = [[] for i in range(n)]
for i in range(n):
    indegs[indeg[i]].append(i)

while indegs[0]:
    root = indegs[0].pop()
    bfs(root)
    removeRoot(root)
```

Następnie algorytm oblicza ilość wejść do poszczególnych wierzchołków, te które nie mają wejść, umieszcza w kolejce q1 i uruchamia pętlę wykonywaną, dopóki kolejka q1 i q2 nie będą puste. Wewnątrz pętli są dwie kolejne dla poszczególnych kolejek, wyciągają one wierzchołek z kolejki i usuwają krawędzie biegnące z nich do innych. Ponownie wierzchołki które mają zerowe wejście są umieszczane w przeciwnej kolejce. Jednocześnie do tablicy dot_array dopisywane są linijki, które będą odpowiedzialne za wygenerowanie grafu (wizualnie):

```
indeg = n*[0]
for vecs in graph:
    for i in vecs:
        indeg[i] += 1

q1 = deque()
q2 = deque()
for i in range(len(indeg)):
    if indeg[i]==0:
        q1.append(i)
fnf=""
dot_array=[]
dot_array.append("digraph G {")
while len(q1) != 0 or len(q2) != 0:
    fnf+="("
    while len(q1) != 0:
        u=q1.popleft()
        for v_id in range(len(graph)):
            if u in graph[v_id]:
                dot_array.append(str(v_id)+" -> "+str(u)+";")
        fnf+=word[u]
        for v in graph[u]:
            indeg[v]-=1
            if indeg[v]==0:
                q2.append(v)
    fnf+=")("
    while len(q2) != 0:
        u=q2.popleft()
        for v_id in range(len(graph)):
            if u in graph[v_id]:
                dot_array.append(str(v_id)+" -> "+str(u)+";")
        fnf+=word[u]
        for v in graph[u]:
            indeg[v]-=1
            if indeg[v]==0:
                q1.append(v)
    fnf+=")"
```

Uzupełnienie tablicy dot_array o nazwy poszczególnych wierzchołków:

```
for x in range(len(word)):
    dot_array.append(str(x)+"[label="+word[x]+"]")
dot_array.append("}")
print()
dot_array="\n".join(dot_array)
print(dot_array)
```

Zapis danych do pliku wyjściowego output.txt. Plik zostanie zapisany w tym samym katalogu gdzie nastąpiło uruchomienie programu:

```
f = open("output.txt", "w")
f.write("D = "+str(D)+"\n\n")
f.write("I = "+str(I)+"\n\n")
f.write("FNF["+word+"] = "+fnf+"\n\n")
f.write(dot_array)
f.close()
```

Aby zwizualizować graf, wygenerowany kod w formacie DOT można skopiować pod ten adres:
https://dreampuf.github.io/GraphvizOnline

Przykład 1:

Dane wejściowe:

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

A = {a, b, c, d}
w = baadcb

Odpowiedź programu:

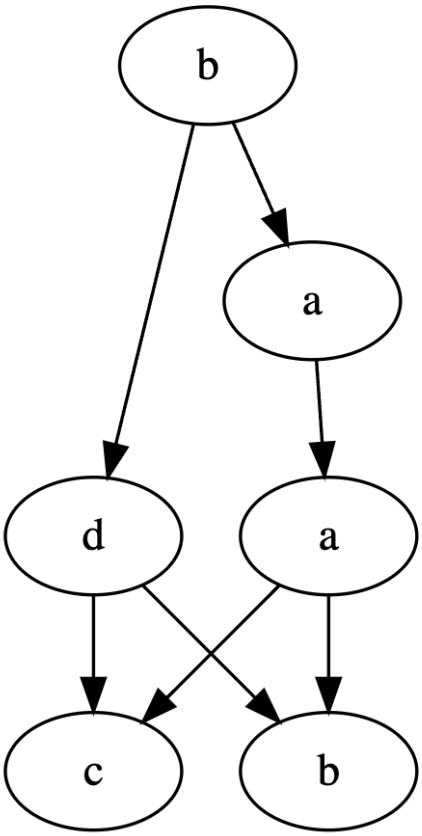
D = [('a', 'a'), ('a', 'b'), ('b', 'a'), ('a', 'c'), ('c', 'a'), ('b', 'b'), ('b', 'd'), ('d', 'b'), ('c', 'c'), ('c', 'd'), ('d', 'c'), ('d', 'd')]

I = [('a', 'd'), ('d', 'a'), ('b', 'c'), ('c', 'b')]

FNF[baadcb] = (b)(ad)(a)(cb)

digraph G {
0 -> 1;
0 -> 3;
1 -> 2;
2 -> 4;
3 -> 4;
2 -> 5;
3 -> 5;
0[label=b]
1[label=a]
2[label=a]
3[label=d]
4[label=c]
5[label=b]
}

Graf:



Przykład 2

Dane wejściowe:

(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

Odpowiedź programu:

D = [('a', 'a'), ('a', 'c'), ('c', 'a'), ('a', 'f'), ('f', 'a'), ('b', 'b'), ('b', 'e'), ('e', 'b'), ('c', 'c'), ('c', 'e'), ('e', 'c'), ('c', 'f'), ('f', 'c'), ('d', 'd'), ('d', 'f'), ('f', 'd'), ('e', 'e'), ('f', 'f')]

I = [('a', 'b'), ('b', 'a'), ('a', 'd'), ('d', 'a'), ('a', 'e'), ('e', 'a'), ('b', 'c'), ('c', 'b'), ('b', 'd'), ('d', 'b'), ('b', 'f'), ('f', 'b'), ('c', 'd'), ('d', 'c'), ('d', 'e'), ('e', 'd'), ('e', 'f'), ('f', 'e')]

FNF[acdcfbbe] = (adb)(cb)(c)(fe)

```
digraph G {
0 -> 1;
5 -> 6;
1 -> 3;
2 -> 4;
3 -> 4;
3 -> 7;
6 -> 7;
0[label=a]
1[label=c]
2[label=d]
3[label=c]
4[label=f]
5[label=b]
6[label=b]
7[label=e]
}
```

Graf: