Sprawozdanie

Sprawozdanie z ćwiczenia nr 5 z przedmiotu Teoria Współbierzności Autor Seweryn Tasior

Implementacja

Program został napisany w języku Python

Wzyanczenie relacji D i I

Do zbioru relacji D i I na podstawie zbioru akcji

```
def get_realtions_from_transactions(transactions:list[list]):
    D = set()
    I = set()
    for letter, var, depends in transactions:
        for other_letter, other_var, other_depends in transactions:
            if other_letter == letter:
                D.add((letter, letter))
                continue
            if var in other_depends or other_var in depends:
                D.add((letter, other_letter))
            else:
                I.add((letter, other_letter))
    D_list = sorted(D)
    I_list = sorted(I)
    return D_list, I_list
```

Tworzenie grafu zależności

```
def get_graph(word,I,D):
    n = len(word)
    digraph = [[] for _ in range(n+1)]

    for i in range(n):
        for j in range(i + 1, n):
            if (word[i], word[j]) in D:
                digraph[i + 1].append(j + 1)

    def is_edge_replacable(i,j,curr):
        if curr==j:
            return True
        for e in digraph[curr]:
            if curr==i and e==j:
                continue
            if is_edge_replacable(i,j,e):
                return True
        return False

    for i in range(1,len(digraph)):
        to_remove = []
        for j in digraph[i]:
            if is_edge_replacable(i, j, i):
                to_remove.append(j)
        if to_remove:
            digraph[i] = [v for v in digraph[i] if v not in to_remove]
    return digraph
```

Obliczanie postaci FNF na podstawie digraphu

```
def get_FNF(word,digraph)->str:
    n = len(word)
    if len(digraph) < n + 1:
        digraph = digraph + [[] for _ in range(n + 1 - len(digraph))]

    indeg = [0] * (n + 1)
    for u in range(1, min(len(digraph), n + 1)):
        for v in digraph[u]:
            if 1 <= v <= n:
                indeg[v] += 1

    remaining = set(range(1, n + 1))
    factors = []
    while remaining:
        layer = sorted(i for i in remaining if indeg[i] == 0)
        if not layer:
            i = min(remaining)
            layer = [i]

        letters = sorted((word[i - 1] for i in layer))
        factors.append('(' + ''.join(letters) + ')')
```

```
        for u in layer:
            for v in digraph[u] if u < len(digraph) else []:
                if 1 <= v <= n:
                    indeg[v] -= 1
            remaining.remove(u)

    return ''.join(factors)
```

Renderowanie grafu z użyciem Graphviz

```
def render_graph(path,FNF:str,digraph:list[list[int]],word:str):
    '''
    Render the dependency digraph using Graphviz.
    '''
    from graphviz import Digraph
    from graphviz.backend.execute import ExecutableNotFound
    import os

    n = len(word)
    if len(digraph) < n + 1:
        digraph = digraph + [[] for _ in range(n + 1 - len(digraph))]

    dot = Digraph('g')
    dot.attr('node', shape='circle')

    for i in range(1, n + 1):
        dot.node(str(i), label=word[i - 1])

    for u in range(1, n + 1):
        for v in sorted(digraph[u]):
            if 1 <= v <= n:
                dot.edge(str(u), str(v))

    base, ext = os.path.splitext(path)
    dot_path = (base or path) + '.dot'
    with open(dot_path, 'w', encoding='utf-8') as f:
        f.write(dot.source)
    print(f' Saved DOT file to {dot_path}')

    try:
        from graphviz import Source
        src = Source.from_file(dot_path)
        fmt = ext.lstrip('.').lower() or 'png'
        src.format = fmt
        out = src.render(filename=(base or path), cleanup=True)
        print(f' Rendered graph to {out}')
    except ExecutableNotFound:
        print(' Graphviz "dot" not found. Skipping image rendering.')
```

Ostateczny solver

```
def solve(read_path,write_path,graph_path):
    transactions, A, word = read_data_from_txt(read_path)

    D, I = get_realtions_from_transactions(transactions)
    digraph = get_graph(word,I,D)
    FNF = get_FNF(word,digraph)
    render_graph(graph_path,FNF,digraph,word)

    write_data_to_txt(write_path,D,I,FNF,digraph,word)
solve('./test_data/case1.txt','./test_data/case1_out.txt','./test_data/case1_graph.png')
solve('./test_data/case2.txt','./test_data/case2_out.txt','./test_data/case2_graph.png')
```

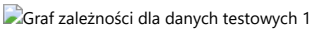
Wyniki

Dane testowe 1

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

```
digraph g {
    node [shape=circle]
    1 [label=a]
    2 [label=c]
    3 [label=d]
    4 [label=c]
    5 [label=f]
    6 [label=b]
    7 [label=b]
    8 [label=e]
    1 -> 2
```

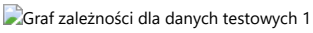
```
    2 -> 4
    3 -> 5
    4 -> 5
    4 -> 8
    6 -> 7
    7 -> 8
}
```


Graf zależności dla danych testowych 1

Dane testowe 2

$ 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)} $ $ I = {(a, f), (e, f), (f, a), (f, e)} $ $ FNF([w]) = (af)(af)(ef)(b)(c)(d) $

```
digraph g {
    node [shape=circle]
    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]
    1 -> 3
    2 -> 5
    3 -> 4
    4 -> 7
    5 -> 6
    6 -> 7
    7 -> 8
    8 -> 9
}
```


Graf zależności dla danych testowych 1