Teoria Współbieżności
Zadanie domowe nr 1 - FNF

Wiktor Dybalski

07.11.2024

1 Cel zadania

Celem zadania jest analiza zależności między akcjami w ramach transakcji, wyznaczenie relacji zależności i niezależności, uzyskanie postaci normalnej Foaty (FNF) oraz stworzenie minimalnego grafu zależności dla słowa w. Wyniki te pozwalają lepiej zrozumieć równoważności operacji wykonywanych równocześnie.

2 Opis programu

Program napisano w języku Java przy użyciu narzędzia Maven. Poniżej przedstawiono kluczowe kroki implementacji:

1. **Przetworzenie danych wejściowych** — Program wczytuje dane z pliku, tworzy mapę powiązań między wierzchołkami grafu a unikalnymi literami identyfikującymi operacje, a następnie filtruje dane do istotnych informacji.
2. **Wyznaczenie relacji zależności D i niezależności I** — Program analizuje relacje między operacjami na zmiennych, określając, które operacje są zależne i niezależne.
3. **Stworzenie oraz minimalizacja grafu zależności** — Program generuje graf zależności przy użyciu struktury typu `HashMap<Integer, List<Integer»`, a następnie minimalizuje go, usuwając zbędne krawędzie.
4. **Obliczenie postaci normalnej Foaty (FNF)** — Wykorzystując minimalny graf zależności, program wyznacza postać normalną Foaty.
5. **Wizualizacja grafu za pomocą biblioteki jgraphx** — Program wizualizuje graf w przyjaznej formie dla użytkownika.

3 Wyniki działania programu

3.1 Dane wejściowe

Dla danych:

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

gdzie $A = \{a, b, c, d\}$ oraz $w =$ baadcb

3.2 Relacja zależności D

Relacja zależności D dla słowa w:

$$D = \{(a,a), (a,b), (a,c), (b,a), (b,b), (b,d), (c,a), (c,c), (c,d), (d,b), (d,c), (d,d)\}$$

3.3 Relacja niezależności I

Relacja niezależności I dla słowa w:

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

3.4 Graf zależności

Minimalny graf zależności stworzony w postaci `HashMap<Integer, List<Integer>>` dla słowa w:

```
Minimized Graph:
0 -> 1 3
1 -> 2
2 -> 4 5
3 -> 4 5
4 -> []
5 -> []
```

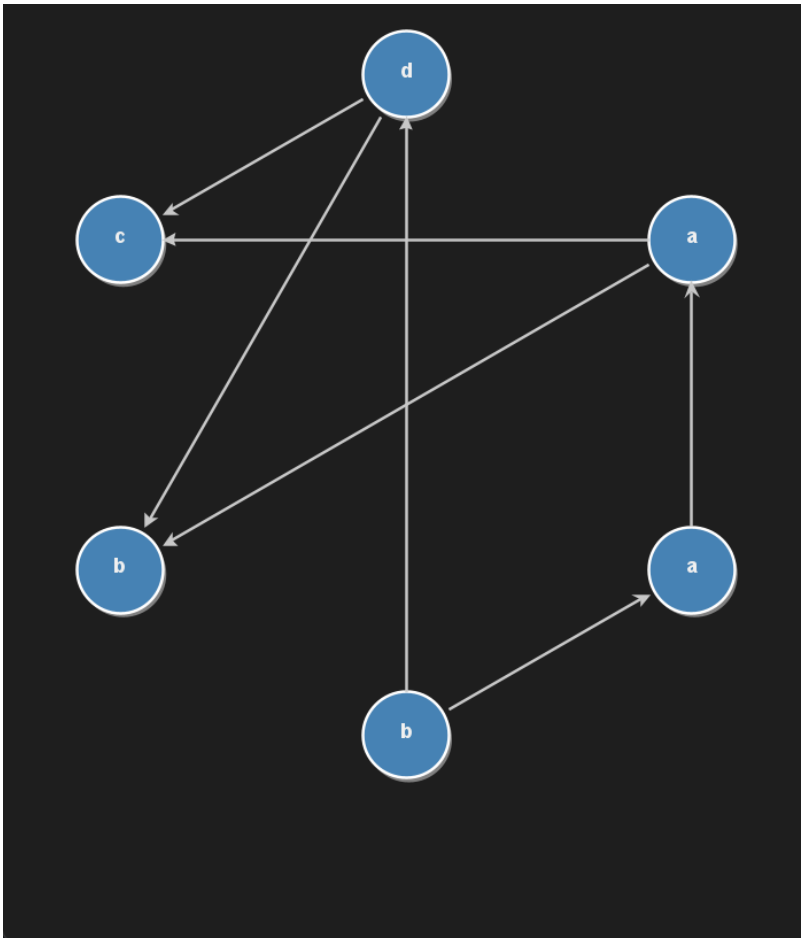
3.5 Postać normalna Foaty (FNF)

Postać normalna Foaty (FNF) dla słowa w:

$$\text{FNF}([w]) = (b)(ad)(a)(cb)$$

3.6 Wizualizacja Grafu

Wizualizacja grafu przy pomocy biblioteki jgraphx:



Rysunek 1: Graf zależności dla danych wejściowych

3.7 Logi z wykonania programu

```
Input file:
(a) x := x + y
(b) y := y + 2z
(c) x := 3x + z
(d) z := y - z
A = {a, b, c, d}
w = baadcb
----------------------
Formatted equations: {a=[x, x, y], b=[y, y, z], c=[x, x, z], d=[z, y, z]}
Dependence and Independence Lists:
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)]
---------------------------
Filled Graph:
0 -> 1 2 3 5
1 -> 2 4 5
2 -> 4 5
3 -> 4 5
4 -> []
5 -> []
---------------------------
Minimized Graph:
0 -> 1 3
1 -> 2
2 -> 4 5
3 -> 4 5
4 -> []
5 -> []
---------------------------
Foata Normal Form (FNF): [b][ad][a][cb]

Process finished with exit code 0
```

Rysunek 2: Logi z działania programu

4 Opis najważniejszych klas i funkcji programu

Program składa się z kilku kluczowych klas, każda pełniąca określoną rolę w zarządzaniu danymi wejściowymi oraz prezentacją graficzną:

4.1 InputManager

Klasa `InputManager` zarządza danymi wejściowymi — odczytuje je, organizuje i formatuje do analizy.

- `readInput()`: Odczytuje dane z pliku.
- `createIntCharMaps()`: Tworzy mapy indeksów znaków i listy ich wystąpień, co ułatwia generowanie grafu.
- `formatInput()`: Przetwarza dane, formatując je do dalszej analizy.

4.2 DependencyBuilder

Klasa `DependencyBuilder` buduje listę zależności i niezależności, sprawdzając, czy zmienne danej operacji występują w innych operacjach.

- `isDependent()`: Sprawdza zależności między dwiema operacjami.
- `createDependencyLists`: Generuje pełną listę zależności, porównując każdą operację z każdą.

4.3 GraphManager

Klasa `GraphManager` tworzy i minimalizuje graf zależności.

- `hasPath()` i `hasPathDFS`: Wykorzystując algorytm DFS, sprawdzają istnienie ścieżki między wierzchołkami.
- `fillGraph()`: Na podstawie tablicy zależności tworzy graf, dodając krawędzie jedynie od mniejszych do większych indeksów śladu.

- `minimizeGraph()`: Usuwa zbędne krawędzie, upewniając się, że nadal istnieje połączenie między danymi wierzchołkami.

4.4 FNFCreator

Klasa `FNFCreator` odpowiada za budowę FNF na podstawie grafu zależności.

- `createFNF()`: Przechodzi po grafie, tworząc tablicę rodziców (Lista InDegree). Zapisuje wierzchołki bez rodziców w danej iteracji, po czym zmniejsza wartości dzieci rodziców i kontynuuje kolejne iteracje.

4.5 GraphPresenter

`GraphPresenter` zajmuje się wizualizacją grafu zależności.

- `presentGraph()`: Renderuje graf na podstawie ustalonych wcześniej zależności.

5 Instrukcja korzystania z programu

5.1 Plik wejściowy

Plik `Input` służy do wpisania danych przez użytkownika. Plik znajduje się w folderze `resources` i powinien być sformatowany według poniższego przykładu:

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

Rysunek 3: Przykładowy plik wejściowy

5.2 Uruchomienie programu

Program można uruchomić w środowisku Maven, np. w Intelij IDEA lub innym IDE wspierającym Javę. Można również skorzystać z terminala:

```
mvn clean compile exec:java
```

Uwaga: Program wymaga Javy w wersji 21, aby działać poprawnie. Należy upewnić się, że używa się odpowiedniej wersji, ponieważ inne wersje mogą powodować problemy z kompatybilnością. Ważne jest także aby upewnić się, że znajduje się w odpowiednim folderze projektu w którym znajduje się plik pom.xml

6 Wnioski

Analiza wyników prowadzi do następujących wniosków:

- Relacja zależności D wskazuje operacje, które wpływają na zmienne w sposób wykluczający ich równoczesne wykonanie.

- Relacja niezależności I identyfikuje operacje możliwe do wykonywania równolegle.
- Postać normalna Foaty (FNF) daje optymalną reprezentację równoczesnych operacji w sekwencji działań.