Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

# Podstawy Programowania Komputerów

## Cykl

| | |
|---|---|
| autor | Wojciech Janota |
| prowadzący | mgr inż. Wojciech Łabaj |
| rok akademicki | 2019/2020 |
| kierunek | informatyka |
| rodzaj studiów | SSI |
| semestr | 1 |
| termin laboratorium | czwartek, 14:30 – 16:00 |
| sekcja | 32 |
| termin oddania sprawozdania | 2020-01-23 |

# 1 Treść zadania

Należało napisać program znajdujący wszystkie cykle w grafie skierowanym. Opis grafu znajduje się w pliku wejściowym, gdzie każda krawędź opisana jest w następujący sposób: `1->2` i oddzielona przecinkiem. Wszystkie znalezione cykle zostaną wypisane do pliku wyjściowego, każdy cykl w osobnej linii. Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

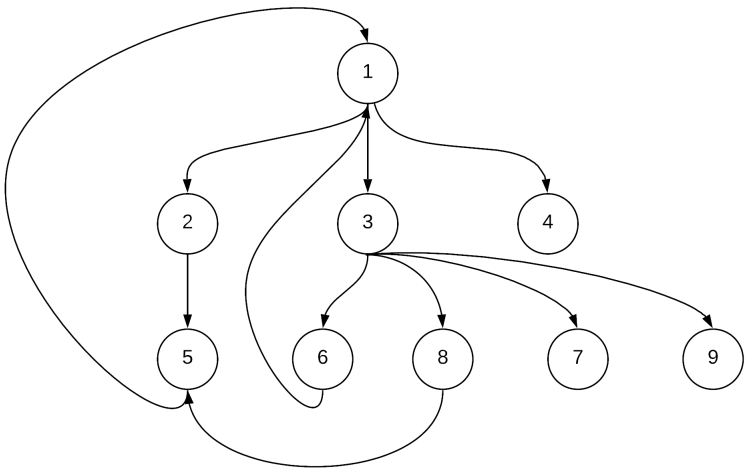
- `-g` plik wejściowy z opisem krawędzi grafu
- `-c` plik wyjściowy z wszystkimi znalezionymi cyklami

# 2 Analiza zadania

Zagadnienie przedstawia problem znajdowania cyklu w grafie skierowanym. Trudność sprawiało także prawidłowe zapisanie reprezentacji grafu w pamięci programu korzystając z list podwieszanych.

## 2.1 Struktury danych

W programie wykorzystano listy podwieszane do przechowywania reprezentacji grafu skierowanego 1 w postaci list sąsiedztwa. Lista podwieszana to lista w której każdy element listy przechowuje oprócz danych wskaźnik na kolejną listę, tworząc coś w rodzaju dynamicznej tablicy dwuwymiarowej. Taka struktura umożliwia łatwe przechowywanie list sąsiednich wierzchołków dla każdego wierzchołka, co z kolei umożliwia przechowywanie grafu w pamięci programu jako list sąsiedztwa. Uzyto także prostego stosu, do przechowywania listy wierzchołków w cyklu. Jest to prosta struktura FILO, w której odkłada się wierzchołki i ma się dostęp tylko do pierwszego elementu stosu.




Rysunek 1: Przykład grafu utworzonego dla danych:
`1->2,2->5,5->1,1->3,6->1,3->7,3->9,1->4, 3->6, 3->8, 8->5`

## 2.2 Algorytmy

Program wykorzystuje listy podwieszane do przechowywania list sąsiedztwa. Następnie wykorzystywany jest algorytm DFS do znalezienia ścieżek, a dokładniej wariant tego algorytmu, który wyszukuje wszystkie ścieżki z wierzchołka A do wierzchołka B. Podając ten sam wierzchołek jako startowy i końcowy jesteśmy w stanie otrzymać wszystkie cykle prowadzące do danego wierzchołka. Algorytm DFS, czyli Depth First Search, działa w następujący sposób [1]: najpierw oznacza wierzchołek początkowy jako odwiedzony, odkłada go na stos, a następnie dla każdego sąsiada wierzchołka z listy sąsiedztwa, który nie jest odwiedzony, wywołuje rekurencyjnie tę funkcję z danym elementem jako wierzchołkiem początkowym. W przeciwnym wypadku następuje sprawdzenie, czy dany wierzchołek jest wierzchołkem końcowym. Jeżeli tak to następuje wypisanie stosu. Po przetworzeniu wszystkich wierzchołków z listy sąsiedztwa następuje zdjęcie ze stosu elementu na jego szczycie.

Złożoność obliczeniowa [1] tego algorytmu wynosi $O(\|V\| + \|E\|)$, gdzie $V$ – liczba wierzchołków, $E$ – liczba krawędzi.

# 3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii komend. Należy przekazać flagi z odpowiednimi argumentami (plik wejściowy i wyjściowy), np dla systemów UNIX:

```
./program -g wejscie -c wyjscie
./program -c cykle -g graf
```

Pliki wejściowe mogą, lecz nie muszą posiadać rozszerzenia. Powinny to być pliki tekstowe, o następującej składni:

```
<wierzcholek_startowy>-><wierzcholek_koncowy>,...
```

Uruchomienie programu z parametrem `-h`

```
program -h
```

powoduje wyświetlenie krótkiej pomocy. Uruchomienie programu z nieprawidłowymi parametrami nie powoduje żadnego działania.

Podanie nieprawidłowej nazwy pliku lub brak jej podania powoduje wyświetlenie następującego błędu:

```
Błąd: nie znaleziono pliku!
```

Brak podania nazwy pliku wyjściowego powoduje wyświetlenie nastepującego błędu:

```
Brak pliku wyjściowego...!
```

# 4 Specyfikacja wewnętrzna

Program został wykonany w oparciu o paradygmat strukturalny. Rozdzielono także interfejs (komunikacja z użytkownikiem) od logiki aplikacji (wyznaczanie cykli).

## 4.1 Ogólna struktura programu

W funkcji głównej najpierw inicjowane są potrzebne zmienne i struktury. Wpierw deklarowane są zmienne typu `string` o nazwach: `NPlikuWej` oraz `NPlikuWyj`. Służą one do przechowywania nazw plików, na których ma operować program. Następnie inicjowana jest struktura `ojciec` o nazwie `graf`. Słuzy ona do przechowywania reprezentacji grafu, pobranego z pliku.

Później pobrane zostają argumenty programu. Po sprawdzeniu poprawności argumentów wywołana zostaje funkcja *wczytaj_graf*. Sprawdza ona czy plik istnieje, jeżeli nie, zwróci ona stosowny komunikat oraz zakończy program. W przeciwnym przypadku korzystając z pomocniczych funkcji: *wyszukaj_ojca*, *dodaj_ojca*, *dodaj_syna*, *dodaj_dane* dane zostaną wczytane do pamięci programu.

Po wczytaniu danych tworzony jest stos typu `wierzcholek` o nazwie `test`. Służy on do przechowywania stosu pomiędzy kolejnymi wywołaniami rekurencyjnymi funkcji $DFS$. Następnie dla każdego wierzchołka jest wywoływana funkcja $DFS$, która analizuje algorytmem DFS podany graf oraz wypisuje znalezione cykle do pliku oraz na wyjście `STDERR`. Zamyka także plik wyjściowy po zakończeniu wypisywania do niego.

Ostatnie zostają wywołane przeładowane funkcje *usun* usuwające stos oraz graf z pamięci oraz zamknięty zostaje plik wyjściowy. Następnie program kończy swoją pracę.

## 4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

# 5 Testowanie

Program został sprawdzony dla róznych grafów skierowanych zawierających, lub nie zawierających cykli. Pliki niezgodne ze specyfikacją są nieobsługiwane. Plik pusty powoduje utworzenie pliku wyjściowego z informacją o braku cykli. Dla grafów acyklicznych także zwrócona zostanie informacja o braku cykli. Aplikacja została przetesowana także dla danych skrajnych, niestety nie byłem w stanie napisać generatora losowego grafów, który generowałby interesujące przypadki, ponieważ rzadko w takich grafach zdarzały się cykle.

Program nie zawiera wycieków pamięci, co zostało sprawdzone narzędziem `valgrind` dostępnym z poziomu domyślnego menadżera pakietów dla systemu Ubuntu 19.10.

# 6 Uzyskane wyniki

Program zwraca opis cykli zaczynających się z każdego wierzchołka. W szczególności program zwróci ten sam cykl dla wszystkich wierzchołków cyklu, lecz zaczynający się w innym punkcie. Przykład: dla danych `2->3,3->2` wypisane zostaną cykle: `2->3->2` oraz `3->2->3`. Postanowiłem zaimplementować wypisywanie cykli w ten sposób aby podkreślić, że cykl może zaczynać i kończyć się w dowolnym wierzchołku cyklu.

# 7 Wnioski

Problem wyszukiwania cykli w grafie sprawił mi małe kłopoty koncepcyjne dot. zastosowania algorytmu DFS do wyszukiwania ścieżki w grafie. Najbardziej wymagającą częścią projektu było dla mnie jednak zaimplementowanie wczytywania danych i obsługa wyjątków. Niestety nadal nie udało mi się zaimplementować sprawdzania poprawności wczytywanych danych, ponieważ nie potrafię wymyślić zgrabnego rozwiązania tego problemu, dlatego też mój program zakłada poprawność danych wejściowych.

## 7.1 Opinie

Wykłady były prowadzone bardzo ciekawie, poruszane zagadnienia były interesujące i życiowe. 'Live coding' także jest ciekawym doświadczeniem podczas wykładu, które bardzo doceniłem, ponieważ o wiele łatwiej było mi przyswoić materiał. Przypadła mi do gustu także metoda prowadzenia ćwiczeń na bazie "tasków", gdzie na początku zajęć od razu wiadomo, co trzeba osiągnąć.

# Literatura

[1] Błażej Osiński Marcin Andrychowicz, Tomasz Kulczyński. Przegląd podstawowych algorytmów. 2010.

# Dodatek
# Szczegółowy opis typów i funkcji