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:

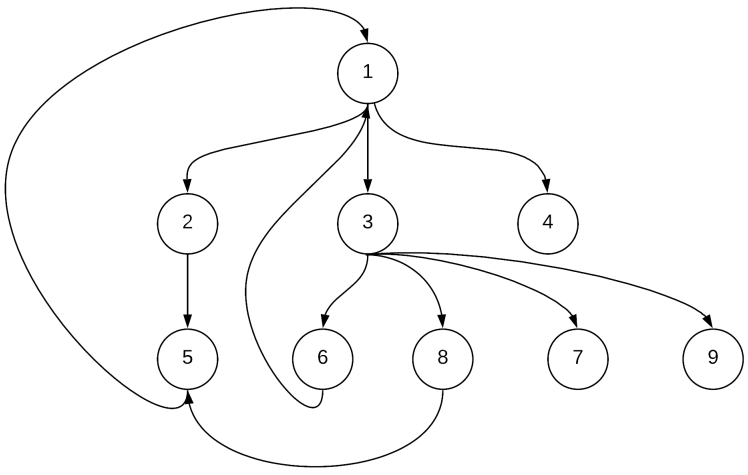
- `-g` plik wejściowy z opisem krawędzi grafu
- `-c` plik wyjściowy z wszystkimi znalezionymi cyklami

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

Wyszukiwanie cyklów w grafie

Wygenerowano przez Doxygen 1.8.13

Spis treści

1 Indeks klas **1**

1.1 Lista klas 1

2 Indeks plików **3**

2.1 Lista plików 3

3 Dokumentacja klas **5**

3.1 Dokumentacja struktury ojciec 5

3.1.1 Opis szczegółowy 5

3.1.2 Dokumentacja atrybutów składowych 5

3.1.2.1 dane 6

3.1.2.2 nastepny 6

3.1.2.3 odwiedzony 6

3.1.2.4 synowie 6

3.2 Dokumentacja struktury wierzcholek 6

3.2.1 Opis szczegółowy 7

3.2.2 Dokumentacja atrybutów składowych 7

3.2.2.1 dane 7

3.2.2.2 nastepny 7

4 Dokumentacja plików **9**

4.1 Dokumentacja pliku funkcje.cpp . . . 9

4.1.1 Dokumentacja funkcji . . . 10

4.1.1.1 DFS() . . . 10

4.1.1.2 dodaj_dane() . . . 10

4.1.1.3 dodaj_do_listy() . . . 10

4.1.1.4 dodaj_ojca() . . . 11

4.1.1.5 dodaj_syna() . . . 11

4.1.1.6 usun() [1/2] . . . 11

4.1.1.7 usun() [2/2] . . . 12

4.1.1.8 wczytaj_graf() . . . 12

4.1.1.9 wyczysc_odwiedzone() . . . 12

4.1.1.10 wypisz_graf() . . . 13

4.1.1.11 wyszukaj_ojca() . . . 13

4.2 Dokumentacja pliku main.cpp . . . 13

Indeks **15**

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

ojciec 5
wierzcholek 6

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

funkcje.cpp 9
funkcje.h . . . . . **??**
main.cpp 13
struktury.h . . . . . **??**

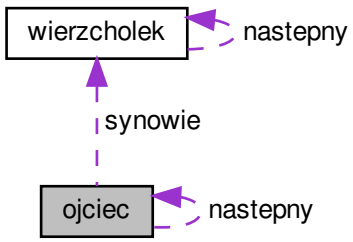
Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury ojciec

`#include <struktury.h>`

Diagram współpracy dla ojciec:



Atrybuty publiczne

- int dane
- bool odwiedzony
- wierzcholek ∗ synowie
- ojciec ∗ nastepny

3.1.1 Opis szczegółowy

Struktura reprezentująca graf. Jest to lista list wierzchołków (synów) wychodzących z danego ojca

3.1.2 Dokumentacja atrybutów składowych

3.1.2.1 dane

```
int ojciec::dane
```

wierzchołek

3.1.2.2 nastepny

```
ojciec* ojciec::nastepny
```

wskaźnik na następnego ojca

3.1.2.3 odwiedzony

```
bool ojciec::odwiedzony
```

czy został odwiedzony

3.1.2.4 synowie

```
wierzcholek* ojciec::synowie
```

wskaźnik na listę synów

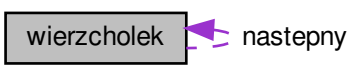
Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

3.2 Dokumentacja struktury wierzcholek

```
#include <struktury.h>
```

Diagram współpracy dla wierzcholek:



Atrybuty publiczne

- int dane
- wierzcholek * nastepny

3.2.1 Opis szczegółowy

Lista trzymająca kolejne wierzchołki wychodzące ze wspólnego ojca.

3.2.2 Dokumentacja atrybutów składowych

3.2.2.1 dane

```
int wierzcholek::dane
```

wierzchołek

3.2.2.2 nastepny

```
wierzcholek* wierzcholek::nastepny
```

wskaźnik do następnego wierzchołka

Dokumentacja dla tej struktury została wygenerowana z pliku:

- struktury.h

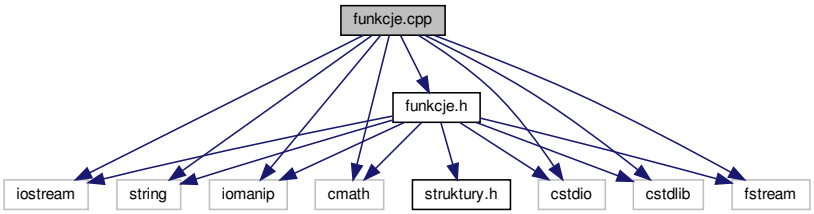
Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku funkcje.cpp

```
#include <iostream>
#include <string>
#include <iomanip>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <fstream>
#include "funkcje.h"
```

Wykres zależności załączania dla funkcje.cpp:



Funkcje

- ojciec ∗ wyszukaj_ojca (ojciec ∗grafHead, int ojciec_dane)
- void dodaj_ojca (ojciec ∗&grafHead, int ojciec_dane)
- void dodaj_syna (ojciec ∗&ojciecSyna, int synDane)
- void dodaj_dane (int ojciecDane, int synDane, ojciec ∗&grafHead)
- void wczytaj_graf (std::string plik, ojciec ∗&start_grafu)
- void wypisz_graf (ojciec ∗grafHead)
- void dodaj_do_listy (ojciec ∗&listaHead, wierzcholek ∗&cykl, int iterator)
- void **wypisz_stos_od_tylu** (wierzcholek ∗stos, std::ofstream ∗plikWy)
- void **dodaj_do_stosu** (wierzcholek ∗&stos, int wartosc)
- void **zdejmij_ze_stosu** (wierzcholek ∗&stos)
- void DFS (ojciec ∗&grafHead, int poczatek, wierzcholek ∗&lista, int pierwszy, std::ofstream ∗plikWy, int &counter)
- void wyczysc_odwiedzone (ojciec ∗&grafHead)
- void usun (ojciec ∗&grafHead)
- void usun (wierzcholek ∗&grafHead)

4.1.1 Dokumentacja funkcji

4.1.1.1 DFS()

```
void DFS (
            ojciec *& grafHead,
            int poczatek,
            wierzcholek *& lista,
            int pierwszy,
            std::ofstream * plikWy,
            int & counter )
```

Funkcja będąca implementacją algorytmu DFS na listach podwieszanych

Parametry

grafHead	graf, którym przechodzimy
poczatek	początkowy wierzchołek do przechodzenia grafu
lista	pomocnicza lista do trzymania stosu wierzchołków
pierwszy	początkowy wierzchołek, do którego ścieżki szukamy
plikWy	plik wyjściowy
counter	licznik cykli

4.1.1.2 dodaj_dane()

```
void dodaj_dane (
            int ojciecDane,
            int synDane,
            ojciec *& grafHead )
```

Funkcja dodająca syna do listy ojca

Parametry

ojciecDane	wartość ojca
synDane	wartość syna
grafHead	adres grafu

4.1.1.3 dodaj_do_listy()

```
void dodaj_do_listy (
            ojciec *& listaHead,
```

```
            wierzcholek *& cykl,
            int iterator )
```

Funkcja dodająca listę z cyklem do listy głównej, zawierającej wszystkie cykle

Parametry

listHead	lista, do której dodajemy cykle

4.1.1.4 dodaj_ojca()

```
void dodaj_ojca (
            ojciec *& grafHead,
            int ojciecDane )
```

Funkcja dodająca ojca do grafu

Parametry

grafHead	graf, w którym dodajemy ojca
ojciecDane	wartość ojca, którą dodajemy do grafu

4.1.1.5 dodaj_syna()

```
void dodaj_syna (
            ojciec *& ojciecSyna,
            int synDane )
```

Funkcja dodająca syna do list adjacencji ojca

Parametry

ojciecSyna	adres ojca, do listy którego dodajemy syna
synDane	wartość syna, którego dodajemy do list ojca

4.1.1.6 usun() [1/2]

```
void usun (
            ojciec *& grafHead )
```

Funkcja usuwająca graf

Parametry

grafHead	graf do usunięcia

4.1.1.7 usun() [2/2]

```
void usun (
            wierzcholek *& grafHead )
```

Funkcja usuwająca listę wierzchołków

Parametry

grafHead	lista do usunięcia

4.1.1.8 wczytaj_graf()

```
void wczytaj_graf (
            std::string plik,
            ojciec *& start_grafu )
```

Funkcja wczytująca graf z podanego pliku do pamięci programu.

Parametry

plik	nazwa pliku, z którego należy wczytać graf
graf	wskaźnik na listę, w której trzymany będzie graf

4.1.1.9 wyczysc_odwiedzone()

```
void wyczysc_odwiedzone (
            ojciec *& grafHead )
```

Funkcja zerująca znacznik `odwiedzony` w grafie

Parametry

grafHead	graf

4.1.1.10 wypisz_graf()

```
void wypisz_graf (
            ojciec * grafHead )
```

Funkcja wypisująca tablicę adjacencji (debugging)

Parametry

grafHead	graf do wypisania

4.1.1.11 wyszukaj_ojca()

```
ojciec* wyszukaj_ojca (
            ojciec * grafHead,
            int ojciecDane )
```

Funkcja wyszukująca adres ojca listy do której należy dodać kolejny wierzchołek.

Parametry

grafHead	graf w którym szukamy ojca
ojciecDane	wartość ojca, którego szukamy

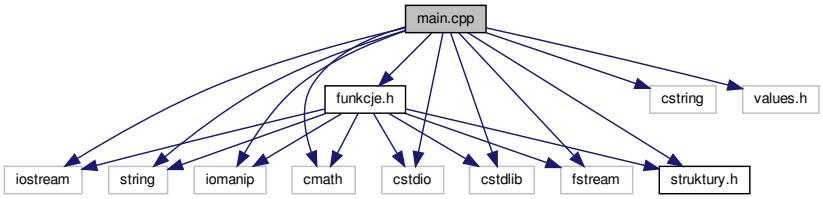
Zwraca

adres ojca, którego szukano

4.2 Dokumentacja pliku main.cpp

```
#include <iostream>
#include <string>
#include <iomanip>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <values.h>
#include <fstream>
#include "funkcje.h"
#include "struktury.h"
```

Wykres zależności załączania dla main.cpp:



Funkcje

- int **main** (int argc, char ∗argv[ ])

Skorowidz

DFS
    funkcje.cpp, 10
dane
    ojciec, 5
    wierzcholek, 7
dodaj_dane
    funkcje.cpp, 10
dodaj_do_listy
    funkcje.cpp, 10
dodaj_ojca
    funkcje.cpp, 11
dodaj_syna
    funkcje.cpp, 11

funkcje.cpp, 9
    DFS, 10
    dodaj_dane, 10
    dodaj_do_listy, 10
    dodaj_ojca, 11
    dodaj_syna, 11
    usun, 11, 12
    wczytaj_graf, 12
    wyczysc_odwiedzone, 12
    wypisz_graf, 12
    wyszukaj_ojca, 13

main.cpp, 13

nastepny
    ojciec, 6
    wierzcholek, 7

odwiedzony
    ojciec, 6
ojciec, 5
    dane, 5
    nastepny, 6
    odwiedzony, 6
    synowie, 6

synowie
    ojciec, 6

usun
    funkcje.cpp, 11, 12

wczytaj_graf
    funkcje.cpp, 12
wierzcholek, 6
    dane, 7
    nastepny, 7
wyczysc_odwiedzone
    funkcje.cpp, 12
wypisz_graf
    funkcje.cpp, 12
wyszukaj_ojca
    funkcje.cpp, 13