Izomorfizm grafów
GIS
Sprawozdanie 1

Monika Woźniak Paweł Szynkiewicz

10 stycznia 2014

Prowadzący projekt: mgr inż. Łukasz Błaszczyk

1 Problem izomorfizmu grafów

Grafy $G_X = (V_X, E_X)$ i $G_Y = (V_Y, E_Y)$ są izomorficzne wtedy i tylko wtedy, gdy istnieje co najmniej jedno przekształcenie wzajemnie jednoznaczne; zbiorów wierzchołków $f\colon V_X \to V_Y$ oraz zbiorów krawędzi $g\colon E_X \to E_Y$ zachowujące relację incydencji krawędzi (albo przylegania wierzchołków), tzn. takie że $(v, w) \in E_X$ wtedy i tylko wtedy, gdy $(f(v), f(w)) \in E_Y$.

Grafy izomorficzne mają identyczną strukturę, różnią się tylko etykietami wierzchołków. Rozpatrywane będą tylko grafu skierowane. W ogólności, każdy graf nieskierowany da się przekształcić w graf skierowany, zamieniając każdą krawędź w grafie nieskierowanym na dwie krawędzie o przeciwnych zwrotach. Grafy nieskierowane są izomorficzne, gdy otrzymane z nich grafy skierowane są izomorficzne.

Warunki konieczne, jakie muszą spełniać grafy izomorficzne to:

- Jednakowa liczba wierzchołków i krawędzi.
- Ten sam rozkład stopni wierzchołków – izomorficzne mogą być tylko wierzchołki o tych samych stopniach.
- Wierzchołki sąsiadujące z wierzchołkami izomorficznymi muszą mieć identyczny rozkład stopni.
- Grafy muszą mieć taką samą liczbę cykli o identycznej długości [2, str. 49].

W ogólnym przypadku, bez dodatkowych założeń, stwierdzenie izomorfizmu grafów należy do klasy problemów NP trudnych. W praktyce, dla większości grafów możliwe jest ograniczenia przeszukiwanego zbioru wszystkich przyporządkowań. Ograniczenia te wynikają bezpośrednio z warunków koniecznych jakie muszą spełniać grafy izomorficzne.

2 Metoda powrotów

Problem rozstrzygania izomorfizmu grafów skierowanych rozwiązywany będzie za pomocą algorytmu powrotów (ang. *backtracking*). Jest to ogólna strategia konstruowania optymalnych rozwiązań ponadwielomianowych, gdy nie znane jest rozwiązanie wielomianowe, lub gdy takie rozwiązanie nie istnieje.

2.1 Opis algorytmu

Rozpatrywana jest przestrzeń stanów, gdzie stan to pewne przyporządkowanie węzłów z podzbioru wierzchołków grafu G_X, węzłom z podzbioru wierzchołków grafu G_Y. Stan jest sytuacją stanowiącą rozwiązanie problemu albo mogącą prowadzić do rozwiązania poprzez przechodzenie z jednego stanu w drugi.

Aby znaleźć rozwiązanie, konieczne jest przeszukanie przestrzeni stanów, przechodząc z jednego stanu w drugi, aż do uzyskania stanu określającego rozwiązanie problemu. Ponieważ dla każdego stanu może istnieć wiele dopuszczalnych ruchów, czyli wiele stanów, do których można dojść, możliwy jest wybór złego posunięcia. Jeżeli wykonano zły ruch i obecny stan nie może prowadzić do rozwiązania (nie osiągnięto poprawnego rozwiązania i nie ma więcej dopuszczalnych posunięć), konieczne jest cofnięcie się do ostatniego stanu, w którym istnieje możliwość wykonania poprawnego przejścia. Metoda powrotów wymaga zapamiętania wszystkich wykonanych ruchów, czy też wszystkich odwiedzonych stanów.

Przestrzeń wszystkich, możliwych wzajemnych przyporządkowań wierzchołków dwóch n-wierzchołkowych grafów skierowanych składa się z $n!$ elementów. Ze względu na zależność $n!$ naiwny algorytm powrotów ma dużą złożoność i jest niezadowalający. Do rozwiązania problemu zastosowana będzie metoda powrotów z ograniczeniami. Gałęzie drzewa wyszukiwania, nie prowadzące do poprawnego przyporządkowania, będą zawczasu odcinane.

Definicja 1.

(i) ***Wejściowość*** *wierzchołka grafu skierowanego odpowiada liczbie krawędzi* ***wchodzących*** *do tego wierzchołka.*

(ii) ***Wyjściowość*** *wierzchołka grafu skierowanego odpowiada liczbie krawędzi* ***wychodzących*** *z tego wierzchołka.*

Obserwacja 1. *Wierzchołek z grafu G_X może być przyporządkowany wierzchołkowi z grafu G_Y tylko, gdy ich wejściowość i wyjściowość są sobie równe.*

2.2 Założenia algorytmu

Dane są dwa spójne grafy skierowane:

$$G_X = (V_X, E_X)$$
$$G_Y = (V_Y, E_Y)$$

należy rozstrzygnąć, czy grafy są izomorficzne, $G_X \cong G_Y$. Zakładamy, że oba grafy mają taką samą liczbę wierzchołków i taką samą liczbę krawędzi ($|V_X| = |V_Y|$ i $|E_X| = |E_Y|$). Dodatkowo multizbiory zawierający wejściowości, odpowiednio wyjściowości wierzchołków grafu

G_X, odpowiada multizbiorom wejściowości i wyjściowości wierzchołków grafu G_Y (obs. 1) W przypadku nie spełnienia założeń, grafy G_X i G_Y nie są izomorficzne. Jeden z grafów, (załóżmy, że G_X) przyjmuje się za graf odniesienia. Niech $G_X(k)$ oznacza podgraf grafu G_X indukowany przez zbiór wierzchołków $\{1, 2 \ldots k\}$, $0 \leq k \leq n$ [1, str. 362].

2.3 Kroki algorytmu

Do rozstrzygnięcia, czy $G_X \cong G_Y$ użyta zostanie metoda powrotów, gdzie stanem jest przyporządkowanie wierzchołków grafu $G_X(k)$ i pewnego podgrafu grafu G_Y. $G_X(0)$ jest izomorficzny z pustym pografem G_Y. Po k krokach znaleziono podgraf G_Y złożony z wierzchołków $S \subseteq V_Y$ izomorficzny z $G_X(k)$. Następnie próbuje się rozszerzyć izomorfizm na graf $G_X(k+1)$, wybierając wierzchołek $v \in V_Y - S$, który może być przyporządkowany wierzchołkowi $k+1 \in V_X$. Jeżeli uda się znaleźć taki wierzchołek, przyporządkowane zostaje zapamiętane, izomorfizm próbuje się rozszerzyć na $G_X(k+2)$. W przypadku, gdy nie istnieje taki wierzchołek v, powraca się do przyporządkowania z $G_X(k-1)$ i próbuje się wybrać inny wierzchołek który można przyporządkować węzłowi $k \in V_X$. Proces kontynuuje się aż do momentu, gdy $G_X(n) = G_X$, co jest jednoznaczne stwierdzeniu, że grafy G_X i G_Y są izomorficzne lub, gdy proces powróci do $G_X(0)$ co oznacza, że grafy nie są izomorficzne, $G_X \ncong G_Y$.

Algorithm 1: Sprawdzanie izomorfizmu grafów skierowanych

Data: G_X, G_Y, flag k
Result: czy $G_X \cong G_Y$, przy przyporządkowaniu $i \leftrightarrow f_i$

flag $\leftarrow$ *false*
k $\leftarrow 0$
`Isomorph`$(\emptyset)$
if *flag* **then**
  $G_X \cong G_Y$
  przyporządkowanie: $i \leftrightarrow f_i$
else
  $G_X \ncong G_Y$
end

Procedure `Isomorph`(S)
  k $\leftarrow$ k $+ 1$
  if $S = V_Y$ **then**
    flag $\leftarrow$ *true*
  end
  for $v \in V_Y - S$ ***and not*** *flag* **do**
    if `Match`(v) **then**
      $f_k \leftarrow v$
      `Isomorph`$(S \cup \{v\})$
    end
  end
  k $\leftarrow$ k $- 1$

Procedure `Match()`
  if *wierzch.* $v \in V_Y - S$ *można przyporządkować wierzch.* $k \in V_X$ **then**
    return *true*
  else
    return *false*
  end

2.4 Modyfikacja metody powrotów

Modyfikacje metody powrotów mają na celu ograniczanie drzewa wyszukiwania. Gałęzie drzewa, które nie prowadzą do poprawnego przyporządkowania świadczącego o izomorfizmie rozpatrywanych grafów, nie powinny być odwiedzane.

2.4.1 Numerowanie wierzchołków

W powyższym algorytmie, wierzchołki grafu G_X mają etykiety z zakresu $1 \ldots n$. Dobre ponumerowanie wierzchołków pozwala na lepsze ograniczenie przestrzeni przeszukiwania. Metoda numerowania wierzchołków oparta jest na poniższych heurystykach:

Pierwszy – najbardziej ograniczony

Na początku działania algorytmu rozpatrywane powinny być wierzchołki o najmniejszej możliwej liczbie przyporządkowań. W ten sposób drzewo przeszukiwania nie rozrasta się na wstępie. Częste cofanie się do stanu $G_X(1)$ byłoby niekorzystne.

Pierwszy – sąsiadujący

Aktualnie przyporządkowywany wierzchołek powinien sąsiadować z jak największą liczbą wierzchołków przyporządkowanych w poprzednim kroku. Ogranicza to liczbę możliwych przyporządkowań, ponieważ nakładane są dodatkowe wymagania na rozpatrywany wierzchołek.

DFS od najbardziej ograniczonego

Wykorzystywane jest połączenie obu heurystyk. Warunkiem przyporządkowania pierwszego wierzchołka jego stopień. Dlatego numerowanie zaczynamy od wierzchołka $x \in V_X$, takiego że najmniej wierzchołków $v \in V_X$ spełnia warunek $d^+(x) = d^+(v)$ i $d^-(x) = d^-(v)$. Kolejne wierzchołki numerowane są zgodnie z metodą przeszukiwania zstępującego, czyli *DFS*.

2.4.2 Stopień wierzchołka

Wierzchołki można przyporządkować wtedy i tylko wtedy, gdy ich wejściowości i wyjściowości są takie same. Dlatego przy rozpatrywaniu kolejnego wierzchołka grafu G_X, pod uwagę brane będą jedynie takie wierzchołki grafu G_Y, które spełniają powyższy warunek i nie mają jeszcze przyporządkowania. Zasada ta powinna znacznie ograniczyć drzewo przeszukiwania w przypadku większości grafów.

2.4.3 Warunki początkowe

Przed zastosowaniem metody powrotów, sprawdzane będą warunki początkowe, czyli niektóre warunki konieczne jakie muszą spełniać grafy izomorficzne.

1. Jednakowa liczba wierzchołków
2. Jednakowa liczba krawędzi
3. Taki sam rozkład stopni wierzchołków (wejściowość i wyjściowość)

2.5 Poprawność algorytmu

Algorytm opiera się na metodzie powrotów. Metoda ta polega na przechodzeniu drzewa przeszukiwania. Każdy węzeł drzewa, reprezentuje dokładnie jedno unikalne przyporządkowanie wierzchołków grafu G_X wierzchołkom grafu G_Y. Istnieje dokładnie $n!$ takich przyporządkowań, gdzie n to liczba wierzchołków w grafach. Metoda powrotów zakończy się po znalezieniu przyporządkowania będącego izomorfizmem. Zakończy się również, po przejściu całego drzewa przeszukiwania. Ponieważ drzewo ma określony rozmiar, metoda zawsze się zakończy.

W przypadku, gdy grafy są izomorficzne, metoda zakończy swoje działanie na pewnym wierzchołku drzewa przeszukiwania. Wierzchołek ten reprezentuje przyporządkowanie będącym izomorfizmem grafów. Ponieważ przeglądane są wszystkie węzły drzewa, do momentu znalezienia izomorfizmu, metoda na pewno znajdzie przyporządkowanie izomorficzne. Kolejność rozpatrywanych wierzchołków nie ma znaczenia dla poprawnego działania algorytmu. Wszystkie możliwe drzewa przeszukiwania są równoważne (są ze sobą izomorficzne).

Usprawnienia metody powrotów nie wpływają na poprawność działania algorytmu. W szczególności jeżeli dane przyporządkowanie nie spełnia izomorfizmu, to rozszerzenie tego przyporządkowania, także nie spełnia izomorfizmu. Uznanie, że dany wierzchołek $x \in G_X$ nie może być przyporządkowany wierzchołkowi $v \in G_Y$ (bo nie spełnia izomorfizmu) oznacza, że odrzucamy wszystkie przyporządkowania w których x jest przyporządkowane v. Jest to jednoznaczne odcięciu gałęzi drzewa przeszukiwania, w której na pewno nie ma rozwiązania problemu izomorfizmu.

2.6 Złożoność algorytmu

Złożoność metody powrotów to $O(n!)$ gdzie n to liczba wierzchołków w grafie. W przypadku pesymistycznym, pierwszy wierzchołek będzie przyporządkowany n razy, drugi $n-1$, itd... co razem stanowi $n!$ sprawdzonych przyporządkowań. Należy zaznaczyć, że wprowadzone usprawnienia metody powrotów, oparte są na heurystykach. Stosuje się je, mając na celu jak największą redukcję drzewa przeszukiwania. Nie oznacza to jednak, że metody te sprawdzą się w przypadku wszystkich grafów. Próby obcinania gałęzi drzewa mogą zakończyć się niepowodzeniem. W takim przypadku konieczne będzie sprawdzenie wszystkich lub prawie wszystkich $n!$ możliwych przyporządkowań.

W praktyce usprawniona metoda powrotów, powinna działać sprawnie dla rozsądnych wymiarów zadania, zwłaszcza dla grafów rzadkich i grafów o różnorodnym rozkładzie stopni wierzchołków.

3 Założenia implementacyjne

Program zostanie napisany w języku C++. Możliwe będą dwa scenariusze użycia programu:

- Podanie ścieżek do dwóch plików tekstowych z grafami
- Podanie ilości wierzchołków i gęstości do generowania grafu losowego

W pierwszym przypadku, program rozwiązuje problem izomorfizmu zadany przez użytkownika. W drugim, program sam generuje losowe grafy izomorficzne i rozwiązuje problem ich izomorfizmu.

Grafy w plikach zapisane są w reprezentacji list sąsiedztwa. Każda lista zajmuje dokładnie jedną linię. Aby otrzymać parę losowych grafów izomorficznych w pierwszym kroku generowany jest graf losowy. W drugim, graf jest kopiowany otrzymana kopia poddawana jest losowemu przekształceniu izomorficznemu.

Na wyjściu program zwraca komunikat informujący, czy zadane grafy są izomorficzne, oraz odpowiadające temu izomorfizmowi przyporządkowanie wierzchołków.

4 Testy

W celu zweryfikowania poprawności i wydajności powstałego programu, przeprowadzone zostaną dwie grupy testów.

4.1 Testy poprawnościowe

Testy będą służyły weryfikacji wyników zwracanych przez program. Na wejście programu zadawane będą pliki zawierające reprezentację grafów w postaci list sąsiedztwa. Pliki zostaną przygotowane ręcznie. Celem testów będzie sprawdzenie:

- Czy program wykryje parę grafów nieizomorficznych
- Czy program wykryje parę grafów izomorficznych
- W wypadku grafów izomorficznych, czy program zwróci poprawne przyporządkowanie wierzchołków

Przygotowane testy sprawdzą, także poprawność działania następujących etapów programu:

- Weryfikacji warunków początkowych (koniecznych) izomorfizmu
- Metody powrotów

4.2 Testy wydajnościowe

Testy posłużą ocenie jakości działania programu. Złożoność problemu izomorfizmu grafów szybko rośnie wraz z wymiarem zadania. Przeprowadzone testy pomogą przy ocenie programu, jak i samej metody powrotów przy rozpatrywaniu izomorfizmu dla grafów większych rozmiarów.

Grafy do testów, o danej ilości wierzchołków i gęstości, będą generowane losowo. Liczony będzie czas działania programu, w zależności od ilości wierzchołków i krawędzi. Wyniki będą uśredniane.

5 Program Izomorf

Program **Izomorf** służy do weryfikacji izomorfizmu dwóch grafów spójnych, skierowanych. Program jest napisany w języku *C++* w standardzie *C++11*. Został skompilowany w środowisku *Linux*, kernel 3.13.6-1.

5.1 Kod

Logikę programu stanowią trzy klasy: `Vertex`, `Graph`, `IsomorphismAlgo`. Funkcja `main` znajduje się w pliku *izomorf.cpp*, wykorzystuje funkcje pomocnicze z tego pliku do parsowania parametrów wykonania programu.

5.1.1 Klasa Vertex

Klasa reprezentuje wierzchołek grafu i jego listę sąsiedztwa. Obiekt klasy pozwala na:

- Dodawanie wierzchołka sąsiedniego.
- Iterowanie po liście sąsiedztwa.
- Uzyskanie *wejściowości*/*wyjściowości* wierzchołka.

5.1.2 Klasa Graph

Klasa implementuje graf skierowany w reprezentacji list sąsiedztwa. Do przechowywania informacji o listach wykorzystywane są obiekty klasy `Vertex`. Klasa `Graph` to interfejs, który umożliwia tworzenie przechowywanie grafów. Obiekty klasy `Graph` umożliwiają.

- Dodawania wierzchołków i krawędzi do grafu.
- Iterowanie po wierzchołkach i krawędziach grafu.
- Wczytywanie i zapisywanie grafu do strumieni danych.
- Uzyskanie *wejściowości*/*wyjściowości* wierzchołka grafu.

Klasa `Graph` udostępnia metodę do generowania losowych, spójnych grafów skierowanych. Istnieje także możliwość tworzenia losowych grafów, które będą izomorficzne do zadanego grafu.

5.1.3 Klasa IsomorphismAlgo

Algorytm weryfikacji zaimplementowano za pomocą klasy `IsomorphismAlgo`. Pozwala to na przechowywanie w polach klasy struktur danych wykorzystywanych przez algorytm powrotów. Dzięki temu kod jest czytelniejszy, ponieważ funkcje nie pobierają dużej ilości parametrów. Jednocześnie unika się z korzystania ze zmiennych globalnych.

Klasa `IsomorphismAlgo` udostępnia dwie metody publiczne:

`meetsRequirements`	funkcja sprawdza czy dwa grafy spełniają warunki wstępne izomorfizmu grafów.
`isIsomorphism`	funkcja sprawdza czy dwa grafy są izomorficzne.

Można wyróżnić dwa etapy działania klasy `IsomorphismAlgo`

1. inicjalizacja struktur danych.

2. wywołanie metody powrotów.

Metoda powrotów wykonywana jest w metodzie `match`. Funkcja wywoływana jest rekurencyjnie.

5.2 Instrukcja obsługi

Program **Izomorf** należy wywołać z linii komend z odpowiednimi parametrami wykonania.

<brak> wywołanie programu bez parametrów powoduje wyświetlenie pomocy użytkownika.

f FN1 FN2 po opcji `f` należy podać ścieżki do dwóch plików (`FN1`, `FN2`) z grafami w rozpoznawanym formacie. Program wczytuje oba grafy i weryfikuje ich izomorfizm.

r V D program generuje dwa losowe grafy izomorficzne, o liczbie wierzchołków `V` i gęstości krawędzi `D`. Następnie izomorfizm wylosowanych grafów jest weryfikowany. ($0 \leq V \leq 1000$, $D \in (0, 1]$).

t opcja `t` powoduje uruchomienie testów programu i wypisanie ich wyniku na konsolę.

Wiadomość pomocnicza, zwracana jest na konsolę, gdy program zostanie wywołany bez argumentów, lub gdy podane parametry są błędne. Podanie złych parametrów generuje dodatkowo stosowny komunikat o błędzie.

Format wiadomości pomocniczej:

```
Program do weryfikowania izomorfizmu grafów
IZOMORF [OPCJA] [VAL1] [VAL2]
 OPCJE:
-------------------------------------------------------------------------------
    f <plik z grafem 1> <plik z grafem 2>
          wczytaj grafy z plików i przetestuj ich izomorfizm
-------------------------------------------------------------------------------
    r <V = liczba wierzchołków> <D = gęstość>
          wygeneruj graf dwa izomorficzne grafy losowe o danej ilości
          wierzchołków i gęstości i przetestuj ich izomorfizm
          0 <= V <= 1000, D in (0, 1]
-------------------------------------------------------------------------------
    t
          przeprowadź serię testów
```

6 Wyniki testów

Testy zostały przeprowadzone na maszynie z systemem operacyjnym *Linux* 64 bit o z procesorem *Intel Core Duo* 2GHz i pamięcią *RAM* 2GiB. Rozmiar stosu jaki przydzielany funkcji rozszerzono do 20 MB.

6.1 Wyniki testów poprawnościowych

Testy poprawnościowe przeprowadzono na ręcznie przygotowanych parach grafów, zapisanych w plikach w folderze *tests*. Testy poprawnościowe podzielono na trzy klasy, dla każdej klasy wykonano po kilka testów:

1. Para grafów nie spełnia warunków wstępnych izomorfizmu.
   - Para grafów ma różną ilość wierzchołków.
   - Para grafów ma różną ilość krawędzi.
   - Para grafów ma różną liczność wierzchołków o takich samych stopniach.
2. Para grafów spełnia warunki początkowe, ale nie jest izomorficzna.
3. Para grafów jest izomorficzna.

Każda klasa testów sprawdzała inne zachowanie programu.

Ad. 1 Program wykrywał grafy nieizomorficzne, już na etapie sprawdzania warunków wstępnych.

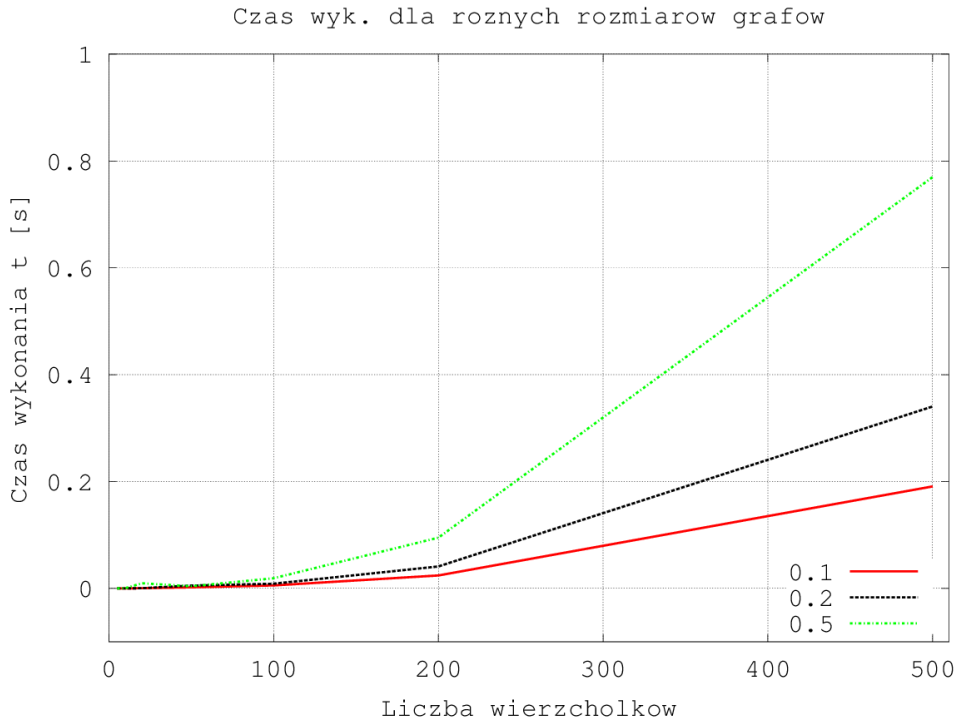
Ad. 2 Grafy przechodziły testy wstępne, ale algorytm wykrywał, że nie są izomorficzne.

Ad. 3 Program weryfikował grafy jako izomorficzne.

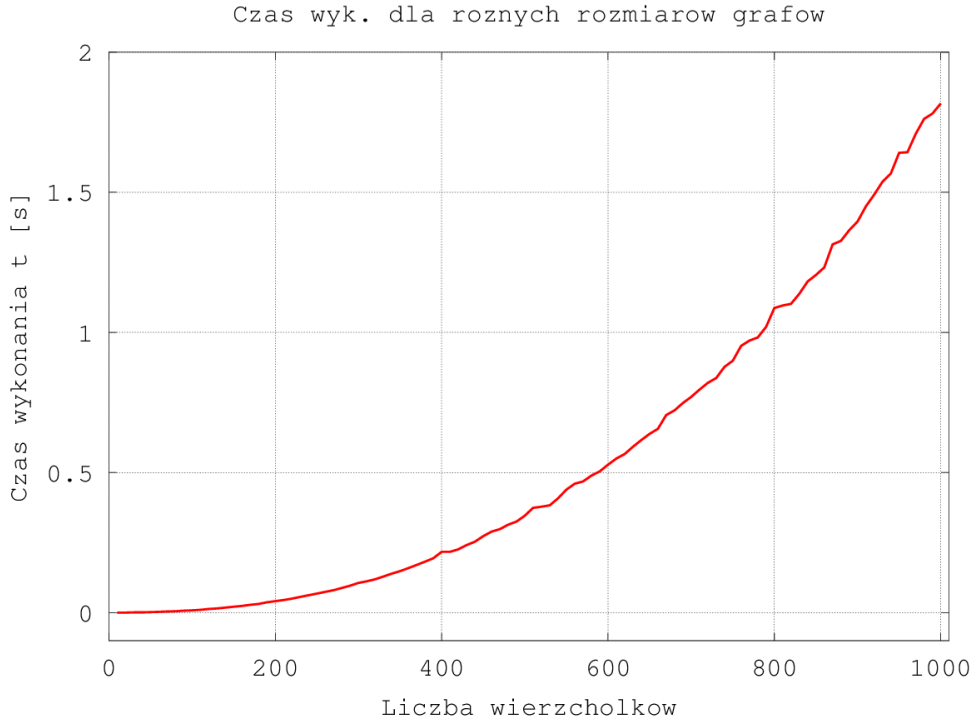
Program przeszedł pomyślnie wszystkie testy poprawnościowe.

6.2 Wyniki testów wydajnościowych

Testy wydajnościowe polegały na weryfikowaniu izomorfizmu dwóch grafów losowych, które z założenia były izomorficzne. Sprawdzano szybkość działania programu dla grafów o różnej ilości wierzchołków i krawędzi. Czas wykonania rośnie wykładniczo wraz ze wzrostem liczby wierzchołków. Dla grafów o większej ilości krawędzi program działa i takiej samej ilości wierzchołków, program działa dłużej. Poniższy wykres przedstawia uśrednione wyniki, przy odrzucaniu skrajnych przypadków.



Rysunek 1: Czas wykonania dla różnych rozmiarów grafów, gęstości krawędzi 0.1, 0.2 i 0.5



Rysunek 2: Czas wykonania dla różnych rozmiarów grafów

7 Wnioski

Wykorzystanie metody powrotów, z dodatkowym obcinaniem gałęzi drzewa przeszukiwań, dawało zadowalające wyniki. Weryfikowano izomorfizm grafów o rozmiarach do 2000 wierzchołków. Średni czas działania metody dla losowych grafów mających 2000 wierzchołków to 5.5 s. Chociaż problem izomorfizmu grafów jest problemem *NP-trudnym* metoda powrotów, wzbogacona o pewne heurystyki działa sprawnie dla większości wylosowanych grafów. Trzeba jednak pamiętać, że pesymistyczna złożoność tej metody to dalej $n!$. Podczas testów na grafach losowych zaobserwowano przypadki, w których program działał wyraźnie dłużej. W skrajnych przypadkach trzeba było przerwać działanie programu. Należy więc podkreślić, że metoda nie sprawdza się dla wszystkich grafów. Ponieważ przestrzeń poszukiwań dla złożoności $n!$ rośnie szybko wraz z wymiarem zadania, przypadki pesymistyczne mogą okazać się nierozwiązywalne już dla grafów o liczbie wierzchołków przekraczającej 20.

Słabą stroną metody może okazać się nie tylko jej złożoność czasowa. Podczas implementacji problemu zauważono, także problemy związane ze złożonością pamięciową metody. Ponieważ funkcja weryfikująca izomorfizm wywołuje się rekurencyjnie, dla większych wymiarów zadania dochodziło do przepełnienia stosu. Rozwiązaniem tymczasowym było zwiększenie wielkości stosu w parametrach systemu (polecenie `ulimits -s` w systemie *Linux*). Być może lepszym rozwiązaniem, byłoby usunięcie rekurencji i symulowanie stosu na stercie.

Literatura

[1] N. Deo E.M. Reingold, J. Nievergelt. *Algorytmy kombinatoryczne*. Wydawnictwo Naukowe PWN, Warszawa, 1985.

[2] K. Pieńkosz J. Wojciechowski. *Grafy i sieci*. Wydawnictwo Naukowe PWN, Warszawa, 2013.

Izomorfizm grafów

Wygenerowano przez Doxygen 1.8.6

So, 4 sty 2014 20:25:22

Spis treści

Rozdział 1

Indeks klas

1.1 Lista klas

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

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja klasy Graph::AdjIter

klasa pomocnicza, iterator po znacznikach wierzchołków sąsiadujących z wierzchołkiem

`#include <graph.hpp>`

Metody publiczne

- AdjIter (const Map &_idx_label_map, const Vertex::iterator &_vit)

  konstruktor potrzebuje mapy i wierzchołka
- AdjIter & operator++ ()
- const AdjIter operator++ (int)
- const label_t & operator∗ ()
- const label_t ∗ operator-> ()
- bool operator== (const AdjIter &rhs)
- bool operator!= (const AdjIter &rhs)

Typy prywatne

- typedef std::map
  < Vertex::idx_t, label_t > Map

  mapowanie za pomocą mapy index -> znacznik

Atrybuty prywatne

- const Map & idx_label_map

  mapa indeks -> znacznik
- Vertex::iterator vit

  iterator po indeksach wierzchołków sąsiadujących

3.1.1 Opis szczegółowy

klasa pomocnicza, iterator po znacznikach wierzchołków sąsiadujących z wierzchołkiem

Ponieważ wewnątrz grafu wierzchołki są indeksowane w inny sposób niż z zewnątrz, potrzebny jest iterator mapujący indeks wierzchołka na jego znacznik

3.1.2 Dokumentacja składowych definicji typu

3.1.2.1 typedef std::map<Vertex::idx_t, label_t> Graph::AdjIter::Map [private]

mapowanie za pomocą mapy index -> znacznik

3.1.3 Dokumentacja konstruktora i destruktora

3.1.3.1 Graph::AdjIter::AdjIter (const Map & *_idx_label_map,* const Vertex::iterator & *_vit*) [inline]

konstruktor potrzebuje mapy i wierzchołka

Parametry

_idx_label_map	mapa do mapowania
_vit	wierzchołek

3.1.4 Dokumentacja funkcji składowych

3.1.4.1 bool Graph::AdjIter::operator!= (const AdjIter & *rhs*) [inline]

3.1.4.2 const label_t& Graph::AdjIter::operator* ( ) [inline]

3.1.4.3 AdjIter& Graph::AdjIter::operator++ ( ) [inline]

3.1.4.4 const AdjIter Graph::AdjIter::operator++ (int) [inline]

3.1.4.5 const label_t* Graph::AdjIter::operator-> ( ) [inline]

3.1.4.6 bool Graph::AdjIter::operator== (const AdjIter & *rhs*) [inline]

3.1.5 Dokumentacja atrybutów składowych

3.1.5.1 const Map& Graph::AdjIter::idx_label_map [private]

mapa indeks -> znacznik

3.1.5.2 Vertex::iterator Graph::AdjIter::vit [private]

iterator po indeksach wierzchołków sąsiadujących

Dokumentacja dla tej klasy została wygenerowana z pliku:

- graph.hpp

3.2 Dokumentacja struktury Graph::Edge

struktura reprezentuje krawędź skierowaną w grafie

`#include <graph.hpp>`

Metody publiczne

- Edge (Graph::label_t _source, Graph::label_t _target)

konstruktor struktury Edge

- bool operator== (const Edge &e) const

  porównywanie krawędzi

- bool operator< (const Edge &e) const

  leksykograficzny porządek na krawędziach

Atrybuty publiczne

- Graph::label_t source
- Graph::label_t target

3.2.1 Opis szczegółowy

struktura reprezentuje krawędź skierowaną w grafie

3.2.2 Dokumentacja konstruktora i destruktora

3.2.2.1 Graph::Edge::Edge (Graph::label_t *_source,* Graph::label_t *_target*) [inline]

konstruktor struktury Edge

Parametry

_source	znacznik wierzchołka źródłowego
_target	znacznik wierzchołka docelowego

3.2.3 Dokumentacja funkcji składowych

3.2.3.1 bool Graph::Edge::operator< (const Edge & *e*) const [inline]

leksykograficzny porządek na krawędziach

Parametry

e	druga krawędź

Zwraca

czy krawędź jest mniejsza od krawędzi e

3.2.3.2 bool Graph::Edge::operator== (const Edge & *e*) const [inline]

porównywanie krawędzi

krawędzie są równe jeżeli źródło i cel są takie same

Parametry

e	druga krawędź

Zwraca

czy są to te same krawędzie

3.2.4 Dokumentacja atrybutów składowych

3.2.4.1 Graph::label_t Graph::Edge::source

3.2.4.2 Graph::label_t Graph::Edge::target

Dokumentacja dla tej struktury została wygenerowana z pliku:

- graph.hpp

3.3 Dokumentacja klasy IsomorphismAlgo::EdgeComparator

komparator do sortowania krawędzi w DFS lesie

Metody publiczne

- EdgeComparator (const dfs_num_t &_dfs_num)

  konstruktor pobiera referencje na mapowanie typu IsomorphismAlgo::dfs_num_t
- bool operator() (const Graph::Edge &left, const Graph::Edge &right)

  sortowanie leksykograficzne krawędzi

Atrybuty prywatne

- const dfs_num_t & dfs_num

  mapa IsomorphismAlgo::dfs_num_t

3.3.1 Opis szczegółowy

komparator do sortowania krawędzi w DFS lesie

Krawędzie z grafu IsomorphismAlgo::graphX są sortowane leksykograficznie względem indeksów wierzchołków IsomorphismAlgo::dfs_idx_t DFS lasu.

3.3.2 Dokumentacja konstruktora i destruktora

3.3.2.1 IsomorphismAlgo::EdgeComparator::EdgeComparator (const dfs_num_t & *_dfs_num*) [inline]

konstruktor pobiera referencje na mapowanie typu IsomorphismAlgo::dfs_num_t

Parametry

_dfs_num	mapowanie typu IsomorphismAlgo::dfs_num_t

3.3.3 Dokumentacja funkcji składowych

3.3.3.1 bool IsomorphismAlgo::EdgeComparator::operator() (const Graph::Edge & *left*, const Graph::Edge & *right*)

sortowanie leksykograficzne krawędzi

Sortowanie krawędzi DFS lasu wymagane w algorytmie powrotów.

Parametry

left	pierwsza krawędź
right	druga krawędź

Zwraca

czy pierwsza krawędź jest mniejsza od drugiej

3.3.4 Dokumentacja atrybutów składowych

3.3.4.1 const dfs_num_t& IsomorphismAlgo::EdgeComparator::dfs_num [private]

mapa IsomorphismAlgo::dfs_num_t

Dokumentacja dla tej klasy została wygenerowana z plików:

- isomorphismAlgo.hpp
- isomorphismAlgo.cpp

3.4 Dokumentacja klasy Graph

klasa implementuje graf w reprezentacji listowej

`#include <graph.hpp>`

Komponenty

- class AdjIter

  klasa pomocnicza, iterator po znacznikach wierzchołków sąsiadujących z wierzchołkiem
- struct Edge

  struktura reprezentuje krawędź skierowaną w grafie

Typy publiczne

- typedef unsigned int label_t

  znacznik wierzchołka w grafie
- typedef std::set< label_t > vertex_set_t

  zbiór znaczników wierzchołków
- typedef vertex_set_t::const_iterator iterator

  iterator po zbiorze znaczników
- typedef std::vector< label_t > dfs_path

  struktura pomocnicza reprezentująca krawędź skierowaną w grafie
- typedef std::set< label_t > dfs_visited

  zbiór wcześniej odwiedzonych wierzchołków
- typedef std::set< Edge > edge_set_t

  zbiór krawędzi

Metody publiczne

- Graph ()

  stwórz pusty graf
- unsigned int getVertexCount () const

  zwróć liczbę wierzchołków w grafie
- unsigned int getEdgeCount () const

  zwróć liczbę krawędzi w grafie
- unsigned int getSize () const

  zwróć liczbę wierzchołków w grafie
- unsigned int getInvariant (label_t label) const

  zwróć 'stopień' (invariant) wierzchołka
- unsigned int getIn (label_t label) const

  zwróć wejściowość wierzchołka
- unsigned int getIn (label_t label, const vertex_set_t &vset) const

  zwróć wejściowość wierzchołka
- unsigned int getOut (label_t label) const

  zwróć wyjściowość wierzchołka
- unsigned int getOut (label_t label, const vertex_set_t &vset) const

  zwróć wyjściowość wierzchołka
- unsigned int getEdges (edge_set_t &edges) const

  zapełnij zbiór krawędziami grafu
- unsigned int getEdges (edge_set_t &edges, label_t label) const
- unsigned int getEdges (edge_set_t &edges, const vertex_set_t &verts) const
- void getDFSPath (dfs_path &path, label_t start, const dfs_visited &visited=dfs_visited()) const

  zapisz ścieżkę po przejściu grafu wgłąb (DFS)
- std::string getInfo () const

  informacje na temat grafu
- bool addVertex (label_t label)

  dodaj wierzchołek do grafu
- void setVertex (label_t label, std::initializer_list< label_t > adj)

  ustaw wierzchołek
- void setVertex (label_t label, std::vector< label_t > adj)

  ustaw wierzchołek
- bool addEdge (label_t v, label_t w)

  dodaje krawędź między dwoma wierzchołkami
- void clear ()

  usuwa wszystkie krawędzie i wierzchołki z grafu
- bool isEmpty () const

  sprawdź czy graf nie zawiera wierzchołków
- bool isNode (label_t label) const

  czy wierzchołek o danym znaczniku należy do grafu
- bool isConnection (label_t v, label_t w) const

  czy w grafie występuje połączenie między wierzchołkami
- bool isConnection (label_t v, label_t w, const vertex_set_t &vset) const

  czy w grafie indukowanym występuje połączenie między wierzchołkami
- iterator begin () const

  iterator wierzchołków grafu
- iterator end () const

  iterator wierzchołków grafu
- AdjIter adjBegin (label_t label) const

iterator wierzchołków incydentnych

- AdjIter adjEnd (label_t label) const

  iterator wierzchołków incydentnych
- std::string dumpVertex (const Vertex &v) const

  zapisz wierzchołek do postaci stringa
- void loadVertex (std::string &str)

  wczyta wierzchołek ze stringa
- void saveToFile (const std::string &filename)

  zapisz graf do pliku
- void loadFromFile (const std::string &filename)

  wczytaj graf z pliku
- Graph & randomIsomorphic (const Graph &other)

  twórz losowy graf izomorficzny

Statyczne metody publiczne

- static Graph generateRandom (unsigned int vertex_count, double density)

  wygeneruj losowy, skierowany graf spójny

Metody prywatne

- label_t getLabel (Vertex::idx_t idx) const

  zwróć znacznik dla wierzchołka o indeksie idx
- Vertex::idx_t getIndex (label_t label) const

  zwróć indeks dla wierzchołka o znaczniku label
- const Vertex & getVertexAt (label_t label) const

  zwraca referencję na wierzchołek o podanym znaczniku
- void findNextFreeIdx ()

  znajdź kolejny wolny indeks dla następnego wierzchołka

Atrybuty prywatne

- unsigned int vertex_count

  liczba wierzchołków w grafie
- unsigned int edge_count

  liczba krawędzi w grafie
- unsigned int __inv_power

  potęga t używana przy liczeniu 'stopnia' wierzchołka
- Vertex::idx_t __first_free_idx

  pierwszy wolny indeks dla kolejnego wierzchołka grafu
- std::map< Vertex::idx_t, Vertex > vertexes

  mapa indeks -> wierzchołek
- std::set< label_t > labels

  wszystkie znaczniki w grafie
- std::map< Vertex::idx_t, label_t > idx_label_map

  mapa indeks -> znacznik
- std::map< label_t, Vertex::idx_t > label_idx_map

  mapa znacznik -> indeks

Przyjaciele

- std::ostream & operator<< (std::ostream &strm, const Graph &g)
- std::istream & operator>> (std::istream &strm, Graph &g)

3.4.1 Opis szczegółowy

klasa implementuje graf w reprezentacji listowej

Klasa Graph to implementacja grafu w reprezentacji list sąsiedztwa. Pozwala na:

- ręczne generowanie grafu
- wczytywanie i zapisywanie grafu do pliku
- generowanie losowego grafu

Wierzchołki grafu identyfikowane są typem Graph::label_t, ale wewnątrz grafu są indeksowane typem Vertex::idx_t. Pozwala to na łatwe wygenerowanie grafów izomorficznych. Wystarczy zmienić mapowanie znaczników na indeksy.

3.4.2 Dokumentacja składowych definicji typu

3.4.2.1 typedef std::vector<label_t> Graph::dfs_path

struktura pomocnicza reprezentująca krawędź skierowaną w grafie

Prosta struktura do reprezentacji krawędzi skierowanych. Wykorzystywania do porównywania krawędzi np. przy sortowaniu. ścieżka dfs po grafie

Wykorzystywane przy przechodzeniu grafu wgłąb

3.4.2.2 typedef std::set<label_t> Graph::dfs_visited

zbiór wcześniej odwiedzonych wierzchołków

Wykorzystywane przy przechodzeniu grafu wgłąb

3.4.2.3 typedef std::set<Edge> Graph::edge_set_t

zbiór krawędzi

3.4.2.4 typedef vertex_set_t::const_iterator Graph::iterator

iterator po zbiorze znaczników

3.4.2.5 typedef unsigned int Graph::label_t

znacznik wierzchołka w grafie

Identyfikuje wierzchołki w grafie. Zewnętrzne oznakowanie wierzchołków.

3.4.2.6 typedef std::set<label_t> Graph::vertex_set_t

zbiór znaczników wierzchołków

3.4.3 Dokumentacja konstruktora i destruktora

3.4.3.1 Graph::Graph ( )

stwórz pusty graf

3.4.4 Dokumentacja funkcji składowych

3.4.4.1 bool Graph::addEdge (label_t *v*, label_t *w*)

dodaje krawędź między dwoma wierzchołkami

Parametry

v	znacznik wierzchołka źródłowego
w	znacznik wierzchołka docelowego

Zwraca

czy dodano krawędź (nie dodano gdy już ustnieje lub nie w grafie nie ma któregoś z wierzchołków)

3.4.4.2 bool Graph::addVertex (label_t *label*)

dodaj wierzchołek do grafu

Parametry

label	znacznik dodawanego wierzchołka

Zwraca

czy rzeczywiście dodano wierzchołek (jeżeli znacznik jest już w użyciu to nie dodano)

3.4.4.3 Graph::AdjIter Graph::adjBegin (label_t *label*) const

iterator wierzchołków incydentnych

Parametry

label	znacznik wierzchołka źródłowego

Zwraca

iterator wskazujący na początek znaczników wierzchołków incydentnych z wierzchołkiem label

3.4.4.4 Graph::AdjIter Graph::adjEnd (label_t *label*) const

iterator wierzchołków incydentnych

Parametry

label	znacznik wierzchołka źródłowego

Zwraca

iterator wskazujący na koniec znaczników wierzchołków incydentnych z wierzchołkiem label

3.4.4.5 iterator Graph::begin ( ) const [inline]

iterator wierzchołków grafu

Zwraca

iterator wskazujący na początek znaczników wierzchołków grafu

3.4.4.6 void Graph::clear ( )

usuwa wszystkie krawędzie i wierzchołki z grafu

3.4.4.7 std::string Graph::dumpVertex (const Vertex & *v*) const

zapisz wierzchołek do postaci stringa

Parametry

v	referencja na wierzchołek

Zwraca

string z zapisanym wierzchołkiem

3.4.4.8 iterator Graph::end ( ) const [inline]

iterator wierzchołków grafu

Zwraca

iterator wskazujący na koniec znaczników wierzchołków grafu

3.4.4.9 void Graph::findNextFreeIdx ( ) [private]

znajdź kolejny wolny indeks dla następnego wierzchołka

3.4.4.10 Graph Graph::generateRandom (unsigned int *vertex_count,* double *density*) [static]

wygeneruj losowy, skierowany graf spójny

Parametry

vertex_count	liczba wierzchołków w grafie losowym
density	gęstość grafu

Zwraca

losowy, skierowany graf spójny o vertex_count wierzchołkach i gęstości density krawędzi

3.4.4.11 void Graph::getDFSPath (dfs_path & *path,* label_t *start,* const dfs_visited & *visited* = dfs_visited()) const

zapisz ścieżkę po przejściu grafu wgłąb (DFS)

Parametry

path	referencja na ścieżkę
start	znacznik wierzchołka startowego
visited	zbiór wierzchołków które należy uznać za wcześniej odwiedzone

3.4.4.12 unsigned int Graph::getEdgeCount ( ) const

zwróć liczbę krawędzi w grafie

Zwraca

liczba krawędzi w grafie

3.4.4.13 unsigned int Graph::getEdges (edge_set_t & *edges*) const

zapełnij zbiór krawędziami grafu

Parametry

edges	referencja na zbiór krawędzi

Zwraca

liczba umieszczonych krawędzi

3.4.4.14 unsigned int Graph::getEdges (edge_set_t & *edges*, label_t *label*) const

zapełnij zbiór krawędziami grafu

krawędzie grafu incydentne z wierzchołkiem label

Parametry

edges	referencja na zbiór krawędzi
label	znacznik wierzchołka z incydentnego

Zwraca

liczba umieszczonych krawędzi

3.4.4.15 unsigned int Graph::getEdges (edge_set_t & *edges*, const vertex_set_t & *verts*) const

zapełnij zbiór krawędziami grafu

krawędzie grafu indukowanego wierzchołkami z verts

Parametry

edges	referencja na zbiór krawędzi
verts	referencja na zbiór znaczników wierzchołków

Zwraca

liczba umieszczonych krawędzi

3.4.4.16 unsigned int Graph::getIn (label_t *label*) const

zwróć wejściowość wierzchołka

Parametry

label	znacznik wierzchołka

Zwraca

wejściowość wierzchołka label

3.4.4.17 unsigned int Graph::getIn (label_t *label,* const vertex_set_t & *vset*) const

zwróć wejściowość wierzchołka

wejściowość wierzchołka w podgrafie indukowanym zbiorem wierzchołków

Parametry

label	znacznik wierzchołka
vset	podzbiór wierzchołków grafu

Zwraca

wejściowość wierzchołka label w grafie indkowanym zbiorem vset

3.4.4.18 idx_t Graph::getIndex (label_t *label*) const [private]

zwróć indeks dla wierzchołka o znaczniku label

Parametry

label	znacznik wierzchołka

Zwraca

indeks wierzchołka label

3.4.4.19 std::string Graph::getInfo ( ) const

informacje na temat grafu

Metoda pomocnicza.

Zwraca

string z informacjami o grafie

3.4.4.20 unsigned int Graph::getInvariant (label_t *label*) const

zwróć 'stopień' (invariant) wierzchołka

Parametry

label	znacznik wierzchołka

Zwraca

'stopień' wierzchołka label

3.4.4.21 label_t Graph::getLabel (Vertex::idx_t *idx*) const [private]

zwróć znacznik dla wierzchołka o indeksie idx

Parametry

idx	indeks wierzchołka

Zwraca

znacznik wierzchołka idx

3.4.4.22 unsigned int Graph::getOut (label_t *label*) const

zwróć wyjściowość wierzchołka

Parametry

label	znacznik wierzchołka

Zwraca

wyjściowość wierzchołka label

3.4.4.23 unsigned int Graph::getOut (label_t *label,* const vertex_set_t & *vset*) const

zwróć wyjściowość wierzchołka

wyjściowość wierzchołka w podgrafie indukowanym zbiorem wierzchołków

Parametry

label	znacznik wierzchołka
vset	podzbiór wierzchołków grafu

Zwraca

wyjściowość wierzchołka label w grafie indkowanym zbiorem vset

3.4.4.24 unsigned int Graph::getSize ( ) const

zwróć liczbę wierzchołków w grafie

Zwraca

liczba wierzchołków w grafie

3.4.4.25 const Vertex & Graph::getVertexAt (label_t *label*) const [private]

zwraca referencję na wierzchołek o podanym znaczniku

Parametry

label	znacznik wierzchołka

Zwraca

referencja na wierzchołek o znaczniku label

3.4.4.26 unsigned int Graph::getVertexCount ( ) const

zwróć liczbę wierzchołków w grafie

Zwraca

liczba wierzchołków w grafie

3.4.4.27 bool Graph::isConnection (label_t *v*, label_t *w*) const

czy w grafie występuje połączenie między wierzchołkami

Parametry

v	znacznik wierzchołka źródłowego
w	znacznik wierzchołka docelowego

Zwraca

czy wstępuje połączenie

3.4.4.28 bool Graph::isConnection (label_t *v*, label_t *w*, const vertex_set_t & *vset*) const

czy w grafie indukowanym występuje połączenie między wierzchołkami

Parametry

v	znacznik wierzchołka źródłowego
w	znacznik wierzchołka docelowego
vset	zbiór wierzchołków rozpinających

Zwraca

czy wstępuje połączenie

3.4.4.29 bool Graph::isEmpty ( ) const

sprawdź czy graf nie zawiera wierzchołków

Zwraca

czy graf jest pusty

3.4.4.30 bool Graph::isNode (label_t *label*) const

czy wierzchołek o danym znaczniku należy do grafu

Parametry

label	znacznik wierzchołka

Zwraca

czy graf zawiera wierzchołek label

3.4.4.31 void Graph::loadFromFile (const std::string & *filename*)

wczytaj graf z pliku

Parametry

filename	nazwa pliku

3.4.4.32 void Graph::loadVertex (std::string & *str*)

wczyta wierzchołek ze stringa

Parametry

str	string z wierzchołkiem

3.4.4.33 Graph & Graph::randomIsomorphic (const Graph & *other*)

twórz losowy graf izomorficzny

Parametry

other	graf który będzie izomorficzny z nowym grafem

Zwraca

referencja na graf izomorficzny do grafu other

3.4.4.34 void Graph::saveToFile (const std::string & *filename*)

zapisz graf do pliku

Parametry

filename	nazwa pliku

3.4.4.35 void Graph::setVertex (label_t *label,* std::initializer_list< label_t > *adj*)

ustaw wierzchołek

ustawia listę sąsiedztwa wierzchołka. Jeżeli w grafie nie istnieją jeszcze rozpatrywane wierzchołki to są one dodawane.

Parametry

label	znacznik wierzchołka
adj	lista znaczników wierzchołków (lista sąsiedztwa)

3.4.4.36 void Graph::setVertex (label_t *label,* std::vector< label_t > *adj*)

ustaw wierzchołek

ustawia listę sąsiedztwa wierzchołka. Jeżeli w grafie nie istnieją jeszcze rozpatrywane wierzchołki to są one dodawane.

Parametry

label	znacznik wierzchołka
adj	lista znaczników wierzchołków (lista sąsiedztwa)

3.4.5 Dokumentacja przyjaciół i funkcji związanych

3.4.5.1 std::ostream& operator<< (std::ostream & *strm,* const Graph & *g*) [friend]

3.4.5.2 std::istream& operator>> (std::istream & *strm,* Graph & *g*) [friend]

3.4.6 Dokumentacja atrybutów składowych

3.4.6.1 Vertex::idx_t Graph::__first_free_idx [private]

pierwszy wolny indeks dla kolejnego wierzchołka grafu

3.4.6.2 unsigned int Graph::__inv_power [private]

potęga t używana przy liczeniu 'stopnia' wierzchołka

10^t * wyjściowość + wejściowość

3.4.6.3 unsigned int Graph::edge_count [private]

liczba krawędzi w grafie

3.4.6.4 std::map<Vertex::idx_t, label_t> Graph::idx_label_map [private]

mapa indeks -> znacznik

3.4.6.5 std::map<label_t, Vertex::idx_t> Graph::label_idx_map [private]

mapa znacznik -> indeks

3.4.6.6 std::set<label_t> Graph::labels [private]

wszystkie znaczniki w grafie

3.4.6.7 unsigned int Graph::vertex_count [private]

liczba wierzchołków w grafie

3.4.6.8 std::map<Vertex::idx_t, Vertex> Graph::vertexes [private]

mapa indeks -> wierzchołek

Dokumentacja dla tej klasy została wygenerowana z plików:

- graph.hpp
- graph.cpp

3.5 Dokumentacja klasy IsomorphismAlgo

klasa reprezentuje algorytm do weryfikacji izomorfizmu grafów

`#include <isomorphismAlgo.hpp>`

Komponenty

- class EdgeComparator

  komparator do sortowania krawędzi w DFS lesie

Typy publiczne

- typedef std::map
  < Graph::label_t,
  Graph::label_t > iso_map

  mapa do reprezentacji izomorfizmu dwóch grafów

Metody publiczne

- IsomorphismAlgo (const Graph &_graphX, const Graph &_graphY)

  konstruktor biorący referencje na dwa grafy
- bool isIsomorphism ()

  funkcja weryfikująca izomorfizm grafów podanych w konstruktorze
- bool meetsRequirements ()

  sprawdź warunki podstawowe izomorfizmu grafów podanych w konstruktorze.
- const iso_map & getIsoMap () const

  zwraca referencję na przekształcenie izomorficzne, otrzymane przy weryfikacji izomorfizmu.
- std::string getInfo () const

  informacje pomocnicze na temat obiektu klasy IsomorphismAlgo

Statyczne metody publiczne

- static bool verifyIsomorphism (const Graph &_graphX, const Graph &_graphY, const iso_map &f)

  Weryfikuje przekształcenie izomorficzne dwóch grafów.

Typy prywatne

- typedef int dfs_idx_t

  indeks wierzchołka DFS lasu
- typedef std::vector
  < Graph::Edge >
  ::const_iterator edge_iter

  iterator po posortowanych krawędziach
- typedef std::map
  < Graph::label_t, dfs_idx_t > dfs_num_t

  mapowanie Graph::label_t -> IsomorphismAlgo::dfs_idx_t
- typedef std::vector
  < Graph::label_t > dfs_vec_t

  mapowanie IsomorphismAlgo::dfs_idx_t -> Graph::label_t

Metody prywatne

- bool match (edge_iter iter, int dfs_num_k)

  główna procedura sprawdzająca izomorfizm grafów
- void resetData ()

  czyści struktury danych
- void countInvBuckets ()

  wyznacza liczność wierzchołków o tych samych 'stopniach' (invariant) w obu grafach.
- void numberVertexes ()

  numeruje wierzchołki grafu IsomorphismAlgo::graphX zgodnie z czasem przechodzenia DFS
- void orderEdges ()

  pobiera i sortuje krawędzie grafu IsomorphismAlgo::graphX

Atrybuty prywatne

- const Graph & graphX

  Graf X na podstawie którego budowany będzie DFS las.
- const Graph & graphY

  Graf Y w którym szukane będą wierzchołki izomorficzne.
- iso_map f_map

  przekształcenie izomorficzne wierzchołków grafu IsomorphismAlgo::graphX na IsomorphismAlgo::graphY
- dfs_num_t dfs_num

  mapowanie Graph::label_t -> IsomorphismAlgo::dfs_idx_t dla wierzchołków grafu IsomorphismAlgo::graphX
- dfs_vec_t dfs_vec

  mapowanie IsomorphismAlgo::dfs_idx_t -> Graph::label_t dla wierzchołków grafu IsomorphismAlgo::graphX
- std::set< Graph::label_t > in_S

  wierzchołki grafu IsomorphismAlgo::graphY już dopasowane
- std::vector< Graph::Edge > ordered_edges

  posortowane krawędzie DFS lasu, odpowiadające krawędziom grafu IsomorphismAlgo::graphX
- std::map< unsigned int, unsigned int > invX_buckets

  struktura pomocnicza, do wyznaczenia liczby wierzchołków o takim samym 'stopniu' (invariant) w grafie IsomorphismAlgo::graphX
- std::map< unsigned int, unsigned int > invY_buckets

  struktura pomocnicza, do wyznaczenia liczby wierzchołków o takim samym 'stopniu' (invariant) w grafie IsomorphismAlgo::graphY
- int edges_count_k

  licznik wykorzystywany w metodzie IsomorphismAlgo::match

3.5.1 Opis szczegółowy

klasa reprezentuje algorytm do weryfikacji izomorfizmu grafów

Klasa zawiera struktury danych i funkcje pomocnicze realizujące rozszerzoną wersję algorytmu powrotów.

Udostępnia także funkcję weryfikującą izomorfizm dwóch grafów.

3.5.2 Dokumentacja składowych definicji typu

3.5.2.1 typedef int IsomorphismAlgo::dfs_idx_t [private]

indeks wierzchołka DFS lasu

3.5.2.2 typedef std::map<Graph::label_t, dfs_idx_t> IsomorphismAlgo::dfs_num_t [private]

mapowanie Graph::label_t -> IsomorphismAlgo::dfs_idx_t

3.5.2.3 typedef std::vector<Graph::label_t> IsomorphismAlgo::dfs_vec_t [private]

mapowanie IsomorphismAlgo::dfs_idx_t -> Graph::label_t

3.5.2.4 typedef std::vector<Graph::Edge>::const_iterator IsomorphismAlgo::edge_iter [private]

iterator po posortowanych krawędziach

3.5.2.5 typedef std::map<Graph::label_t, Graph::label_t> IsomorphismAlgo::iso_map

mapa do reprezentacji izomorfizmu dwóch grafów

mapa znacznik (grafu X) -> znacznik (grafu Y)

3.5.3 Dokumentacja konstruktora i destruktora

3.5.3.1 IsomorphismAlgo::IsomorphismAlgo (const Graph & *_graphX,* const Graph & *_graphY*)

konstruktor biorący referencje na dwa grafy

Parametry

_graphX	graf X
_graphY	graf Y

3.5.4 Dokumentacja funkcji składowych

3.5.4.1 void IsomorphismAlgo::countInvBuckets ( ) [private]

wyznacza liczność wierzchołków o tych samych 'stopniach' (invariant) w obu grafach.

Generuje struktury danych IsomorphismAlgo::invX_buckets i IsomorphismAlgo::invY_buckets.

3.5.4.2 std::string IsomorphismAlgo::getInfo ( ) const

informacje pomocnicze na temat obiektu klasy IsomorphismAlgo

Zwraca

string z informacjami pomocniczymi

3.5.4.3 const iso_map& IsomorphismAlgo::getIsoMap ( ) const [inline]

zwraca referencję na przekształcenie izomorficzne, otrzymane przy weryfikacji izomorfizmu.

Zwraca

referencja na mapę znaczników grafu X na graf Y

3.5.4.4 bool IsomorphismAlgo::isIsomorphism ( )

funkcja weryfikująca izomorfizm grafów podanych w konstruktorze

- W pierwszym kroku sprawdzane jest czy grafy spełniają wymagania podstawowe IsomorphismAlgo::meetsRequirements.
- Jeżeli grafy spełniają te wymagania, przygotowywane są struktury danych.
- Następnie wykonywany jest właściwy algorytm (metoda powrotów).
- Jeżeli grafy są izomorficzne, po wykonaniu się funkcji obiekt klasy zawiera przekształcenie izomorficzne z grafu X na graf Y.

Zwraca

czy grafy IsomorphismAlgo::graphX i IsomorphismAlgo::graphY są izomorficzne.

3.5.4.5 bool IsomorphismAlgo::match (edge_iter *iter,* int *dfs_num_k*) [private]

główna procedura sprawdzająca izomorfizm grafów

Metoda rekurencyjna, która próbuje przyporządkować k-temu wierzchołkowi DFS lasu, wierzchołek z grafu G_Y, tak żeby G_X[k] ∼= G_Y[S].

Procedura przechodzi przez wierzchołki DFS lesu. Porusza się po odpowiednio posortowanych krawędziach grafu IsomorphismAlgo::graphX. Gdy uda się przejść wszystkie krawędzie grafu IsomorphismAlgo::graphX, procedura kończy się sukcesem.

Parametry

iter	iterator po posortowanych krawędziach grafu IsomorphismAlgo::graphX, czyli po IsomorphismAlgo::dfs_num.
dfs_num_k	indeks rozpatrywanego wierzchołka DFS lasu.

Zwraca

czy krawędź wskazywana przez `iter` może zostać przyporządkowana jakieś niewykorzystanej krawędzi grafu IsomorphismAlgo::graphY.

3.5.4.6 bool IsomorphismAlgo::meetsRequirements ( )

sprawdź warunki podstawowe izomorfizmu grafów podanych w konstruktorze.

Sprawdzane warunki to:

- Taka sama liczba wierzchołków.
- Taka sama liczba krawędzi.
- Taka sama liczba wierzchołków o jednakowych 'stopniach' (invariant)

Zwraca

czy grafy IsomorphismAlgo::graphX i IsomorphismAlgo::graphY mają szansę być izomorficzne.

3.5.4.7 void IsomorphismAlgo::numberVertexes ( ) [private]

numeruje wierzchołki grafu IsomorphismAlgo::graphX zgodnie z czasem przechodzenia DFS

Generuje struktury danych IsomorphismAlgo::dfs_num i IsomorphismAlgo::dfs_vec.

3.5.4.8 void IsomorphismAlgo::orderEdges ( ) [private]

pobiera i sortuje krawędzie grafu IsomorphismAlgo::graphX

Do sortowania wykorzystywany jest komaprator IsomorphismAlgo::EdgeComparator

Generuje strukturę danych IsomorphismAlgo::ordered_edges

3.5.4.9 void IsomorphismAlgo::resetData ( ) [private]

czyści struktury danych

3.5.4.10 bool IsomorphismAlgo::verifyIsomorphism (const Graph & *_graphX*, const Graph & *_graphY*, const iso_map & *f*) [static]

Weryfikuje przekształcenie izomorficzne dwóch grafów.

Weryfikuje, czy przekształcenie f: X -> Y jest izomorfizmem grafów.

Parametry

_graphX	referencja na graf X
_graphY	referencja na graf Y
f	referencja na przekształcenie

Zwraca

czy przekształcenie jest izomorfizmem

3.5.5 Dokumentacja atrybutów składowych

3.5.5.1 dfs_num_t IsomorphismAlgo::dfs_num [private]

mapowanie Graph::label_t -> IsomorphismAlgo::dfs_idx_t dla wierzchołków grafu IsomorphismAlgo::graphX

3.5.5.2 dfs_vec_t IsomorphismAlgo::dfs_vec [private]

mapowanie IsomorphismAlgo::dfs_idx_t -> Graph::label_t dla wierzchołków grafu IsomorphismAlgo::graphX

3.5.5.3 int IsomorphismAlgo::edges_count_k [private]

licznik wykorzystywany w metodzie IsomorphismAlgo::match

Licznik wskazuje ile jest krawędzi incydentnych z rozpatrywanym wierzchołkiem k grafu G_X[k].

3.5.5.4 iso_map IsomorphismAlgo::f_map [private]

przekształcenie izomorficzne wierzchołków grafu IsomorphismAlgo::graphX na IsomorphismAlgo::graphY

3.5.5.5 const Graph& IsomorphismAlgo::graphX [private]

Graf X na podstawie którego budowany będzie DFS las.

3.5.5.6 const Graph& IsomorphismAlgo::graphY [private]

Graf Y w którym szukane będą wierzchołki izomorficzne.

3.5.5.7 std::set<Graph::label_t> IsomorphismAlgo::in_S [private]

wierzchołki grafu IsomorphismAlgo::graphY już dopasowane

3.5.5.8 std::map<unsigned int, unsigned int> IsomorphismAlgo::invX_buckets [private]

struktura pomocnicza, do wyznaczenia liczby wierzchołków o takim samym 'stopniu' (invariant) w grafie IsomorphismAlgo::graphX

3.5.5.9 std::map<unsigned int, unsigned int> IsomorphismAlgo::invY_buckets [private]

struktura pomocnicza, do wyznaczenia liczby wierzchołków o takim samym 'stopniu' (invariant) w grafie IsomorphismAlgo::graphY

3.5.5.10 std::vector<Graph::Edge> IsomorphismAlgo::ordered_edges [private]

posortowane krawędzie DFS lasu, odpowiadające krawędziom grafu IsomorphismAlgo::graphX

Dokumentacja dla tej klasy została wygenerowana z plików:

- isomorphismAlgo.hpp
- isomorphismAlgo.cpp

3.6 Dokumentacja klasy Vertex

klasa reprezentuje wierzchołek grafie w reprezentacji list sąsiedztwa

`#include <vertex.hpp>`

Typy publiczne

- typedef unsigned int idx_t

  typ po którym indeksowane są wierzchołki w grafie
- typedef std::pair< unsigned int, unsigned int > deg_t

  typ reprezentujący parę wejściowość - wyjściowość
- typedef std::set< idx_t > ::const_iterator iterator

  iterator wierzchołka

Metody publiczne

- Vertex (idx_t _index)

  konstruktor wierzchołka o podanym indeksie
- bool addAdjacent (Vertex &adj)

  dodaj wierzchołek sąsiadujący
- bool isAdjacent (idx_t v) const

  sprawdź czy wierzchołek o danym indeksie jest sąsiadem
- idx_t getIndex () const

  zwraca indeks danego wierzchołka
- unsigned int getOut () const

  zwraca wyjściowość wierzchołka
- unsigned int getIn () const

  zwraca wejściowość wierzchołka
- iterator begin () const

  iterator po indeksach wierzchołków sąsiadujących
- iterator end () const

  iterator po indeksach wierzchołków sąsiadujących
- std::string getInfo () const

  informacje na temat wierzchołka

Metody prywatne

- void addNeighbour ()

  procedura wywoływana gdy wierzchołek jest dodawany do listy sąsiedztwa innego wierzchołka

Atrybuty prywatne

- idx_t index

  indeks wierzchołka
- deg_t degree

  wejściowość i wyjściowość wierzchołka
- std::set< idx_t > adjacent

  lista sąsiedztwa wierzchołka

3.6.1 Opis szczegółowy

klasa reprezentuje wierzchołek grafie w reprezentacji list sąsiedztwa

Klasa odpowiada liście sąsiedztwa jednego wierzchołka.

3.6.2 Dokumentacja składowych definicji typu

3.6.2.1 typedef std::pair<unsigned int, unsigned int> Vertex::deg_t

typ reprezentujący parę wejściowość - wyjściowość

3.6.2.2 typedef unsigned int Vertex::idx_t

typ po którym indeksowane są wierzchołki w grafie

3.6.2.3 typedef std::set<idx_t>::const_iterator Vertex::iterator

iterator wierzchołka

Iterator po indeksach wierzchołków sąsiadujących z danym wierzchołkiem

3.6.3 Dokumentacja konstruktora i destruktora

3.6.3.1 Vertex::Vertex (idx_t *_index*)

konstruktor wierzchołka o podanym indeksie

Parametry

_index	indeks dla tworzonego wierzchołka

3.6.4 Dokumentacja funkcji składowych

3.6.4.1 bool Vertex::addAdjacent (Vertex & *adj*)

dodaj wierzchołek sąsiadujący

Parametry

adj	referencja na dodawany wierzchołek

Zwraca

czy dodano wierzchołek (nie dodano jeżeli wierzchołek jest już sąsiadem)

3.6.4.2 void Vertex::addNeighbour ( ) [private]

procedura wywoływana gdy wierzchołek jest dodawany do listy sąsiedztwa innego wierzchołka

3.6.4.3 Vertex::iterator Vertex::begin ( ) const

iterator po indeksach wierzchołków sąsiadujących

Zwraca

iterator na początek listy wierzchołków

3.6.4.4 Vertex::iterator Vertex::end ( ) const

iterator po indeksach wierzchołków sąsiadujących

Zwraca

iterator na koniec listy wierzchołków

3.6.4.5 unsigned int Vertex::getIn ( ) const

zwraca wejściowość wierzchołka

Zwraca

wejściowość

3.6.4.6 idx_t Vertex::getIndex ( ) const

zwraca indeks danego wierzchołka

Zwraca

indeks

3.6.4.7 std::string Vertex::getInfo ( ) const

informacje na temat wierzchołka

procedura pomocnicza

Zwraca

string z info. wierzchołka

3.6.4.8 unsigned int Vertex::getOut ( ) const

zwraca wyjściowość wierzchołka

Zwraca

wyjściowość

3.6.4.9 bool Vertex::isAdjacent (idx_t *v*) const

sprawdź czy wierzchołek o danym indeksie jest sąsiadem

Parametry

v	indeks potencjalnego sąsiada

Zwraca

czy v to indeks wierzchołka sąsiadującego z danym wierzchołkiem

3.6.5 Dokumentacja atrybutów składowych

3.6.5.1 std::set<idx_t> Vertex::adjacent [private]

lista sąsiedztwa wierzchołka

3.6.5.2 deg_t Vertex::degree [private]

wejściowość i wyjściowość wierzchołka

3.6.5.3 idx_t Vertex::index [private]

indeks wierzchołka

Dokumentacja dla tej klasy została wygenerowana z plików:

- vertex.hpp
- vertex.cpp

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku graph.cpp

implementacja metod klasy Graph

```
#include "graph.hpp"
```

Definicje typów

- typedef Vertex::idx_t idx_t
- typedef Graph::label_t label_t

Funkcje

- std::ostream & operator<< (std::ostream &strm, const Graph &g)
- std::istream & operator>> (std::istream &strm, Graph &g)

4.1.1 Opis szczegółowy

implementacja metod klasy Graph Coś didatkowegoh

4.1.2 Dokumentacja definicji typów

4.1.2.1 typedef Vertex::idx_t idx_t

4.1.2.2 typedef Graph::label_t label_t

4.1.3 Dokumentacja funkcji

4.1.3.1 std::ostream& operator<< (std::ostream & *strm,* const Graph & *g*)

4.1.3.2 std::istream& operator>> (std::istream & *strm,* Graph & *g*)

4.2 Dokumentacja pliku graph.hpp

plik nagłówkowy klasy Graph

```
#include <iostream>
#include <sstream>
#include <string>
#include <fstream>
#include <map>
#include <vector>
#include <stack>
#include <initializer_list>
#include <stdexcept>
#include <chrono>
#include <random>
#include <cmath>
#include "vertex.hpp"
#include "utils.hpp"
```

Komponenty

- class Graph

  klasa implementuje graf w reprezentacji listowej
- struct Graph::Edge

  struktura reprezentuje krawędź skierowaną w grafie
- class Graph::AdjIter

  klasa pomocnicza, iterator po znacznikach wierzchołków sąsiadujących z wierzchołkiem

Definicje

- #define COM_SIGN "#"

  znak komentarza w plikach z rep. grafu
- #define DELIMITER_CHAR ','

  znak oddzielający w plikach z rep. grafu
- #define MAX_RANDOM_FAILS (1000∗10)

  maksymalna liczba losowań przy generowanie grafu losowego

Funkcje

- std::ostream & operator<< (std::ostream &strm, const Graph &g)
- std::istream & operator>> (std::istream &strm, Graph &g)

4.2.1 Opis szczegółowy

plik nagłówkowy klasy Graph Detailed description starts here.

Deklaracja klasy Graph i klas wewnętrznych klasy Graph

4.2.2 Dokumentacja definicji

4.2.2.1 #define COM_SIGN "#"

znak komentarza w plikach z rep. grafu

4.2.2.2 #define DELIMITER_CHAR ','

znak oddzielający w plikach z rep. grafu

4.2.2.3 #define MAX_RANDOM_FAILS (1000*10)

maksymalna liczba losowań przy generowanie grafu losowego

Liczba możliwych nieudanych prób przy losowaniu nowej krawędzi do grafu losowego.

4.2.3 Dokumentacja funkcji

4.2.3.1 std::ostream& operator<< (std::ostream & *strm,* const Graph & *g*)

4.2.3.2 std::istream& operator>> (std::istream & *strm,* Graph & *g*)

4.3 Dokumentacja pliku isomorphismAlgo.cpp

implementacja metod klasy IsomorphismAlgo

```
#include "isomorphismAlgo.hpp"
```

4.3.1 Opis szczegółowy

implementacja metod klasy IsomorphismAlgo Detailed description starts here.

4.4 Dokumentacja pliku isomorphismAlgo.hpp

plik nagłówkowy klasy IsomorphismAlgo

```
#include <map>
#include <set>
#include <sstream>
#include <iostream>
#include "graph.hpp"
```

Komponenty

- class IsomorphismAlgo

  klasa reprezentuje algorytm do weryfikacji izomorfizmu grafów
- class IsomorphismAlgo::EdgeComparator

  komparator do sortowania krawędzi w DFS lesie

4.4.1 Opis szczegółowy

plik nagłówkowy klasy IsomorphismAlgo Detailed description starts here.

Deklaracja klasy IsomorphismAlgo i klas wewnętrznych klasy IsomorphismAlgo

4.5 Dokumentacja pliku izomorf.cpp

plik z metodą main

```
#include <iostream>
#include <iomanip>
#include <string>
#include <sstream>
#include <iterator>
#include <chrono>
#include <exception>
#include <assert.h>
#include <vector>
#include "graph.hpp"
#include "isomorphismAlgo.hpp"
```

Funkcje

- std::string helpMsg ()

  zwraca wiadomość pomocniczą programu
- void readGraph (Graph &g, std::string filename, std::string graphname)

  wczytuje z pliku, ew. wypisuje komunikat błędu
- void checkIsomorphism (const Graph &gX, const Graph &gY)

  wypisuje na konsolę proces weryfikacji izomorfizmu dwóch grafów
- bool runTestUnit (const Graph &gX, const Graph &gY, bool meets, bool izom, unsigned int nr, std::string testname)

  uruchom unittest
- bool runFileTest (std::string filenameX, std::string filenameY, bool meets, bool izom, unsigned int nr, std::string testname)

  uruchom unittest na grafach zapisanych w plikach
- bool runRandomTest (unsigned int v, double d, bool izom, unsigned int nr, std::string testname)

  uruchom unittest na grafach losowych
- void executeTests ()

  uruchom testy
- void executeFromFiles (std::string filenameX, std::string filenameY)

  uruchom program na grafach zapisanych w plikach
- void executeRandom (unsigned int v, double d)

  uruchom program na losowych izomorficznych grafach
- void parseInput (int argc, const char ∗argv[])

  interpretuj argumenty wywołania programu
- int main (int argc, const char ∗argv[])

  funkcja main

4.5.1 Opis szczegółowy

plik z metodą main Detailed description starts here.

4.5.2 Dokumentacja funkcji

4.5.2.1 void checkIsomorphism (const Graph & *gX*, const Graph & *gY*)

wypisuje na konsolę proces weryfikacji izomorfizmu dwóch grafów

Parametry

gX	graf pierwszy
gY	graf drugi

4.5.2.2 void executeFromFiles (std::string *filenameX*, std::string *filenameY*)

uruchom program na grafach zapisanych w plikach

Parametry

filenameX	plik z grafem 1
filenameY	plik z grafem 2

4.5.2.3 void executeRandom (unsigned int *v*, double *d*)

uruchom program na losowych izomorficznych grafach

Parametry

v	ilosc wierzchołków w grafach losowych
d	gęstość krawędzi w grafach losowych

4.5.2.4 void executeTests ( )

uruchom testy

4.5.2.5 std::string helpMsg ( )

zwraca wiadomość pomocniczą programu

Zwraca

string z wiadomością pomocniczą.

4.5.2.6 int main (int *argc*, const char * *argv[]*)

funkcja main

Parametry

argc	ilość parametrów programu
argv[]	parametry programu

Zwraca

zwraca 0 w przypadku poprawnego wykonania, w razie błędnego wartość różną od 0.

4.5.2.7 void parseInput (int *argc*, const char * *argv[]*)

interpretuj argumenty wywołania programu

Parametry

argc	liczba zmiennych
argv[]	tablica argumentów

4.5.2.8 void readGraph (Graph & *g,* std::string *filename,* std::string *graphname*)

wczytuje z pliku, ew. wypisuje komunikat błędu

Parametry

g	wczytywany graf
filename	nazwa pliku z grafem
graphname	nazwa grafu dla komunikatu o błędzie

4.5.2.9 bool runFileTest (std::string *filenameX,* std::string *filenameY,* bool *meets,* bool *izom,* unsigned int *nr,* std::string *testname*)

uruchom unittest na grafach zapisanych w plikach

Parametry

filenameX	plik z grafem 1
filenameY	plik z grafem 2
meets	czy grafy powinny spełniać warunki izomorfizmu
izom	czy grafy powinny być izomorficzne
nr	numer testu
testname	nazwa testu

Zwraca

czy test wykonał się poprawnie

4.5.2.10 bool runRandomTest (unsigned int *v,* double *d,* bool *izom,* unsigned int *nr,* std::string *testname*)

uruchom unittest na grafach losowych

Generuje dwa losowe grafy izomorficzne i weryfikuje ich izomorfizm

Parametry

v	ilosc wierzchołków w grafach losowych
d	gęstość krawędzi w grafach losowych
izom	czy grafy powinny być izomorficzne
nr	numer testu
testname	nazwa testu

Zwraca

czy test wykonał się poprawnie

4.5.2.11 bool runTestUnit (const Graph & *gX,* const Graph & *gY,* bool *meets,* bool *izom,* unsigned int *nr,* std::string *testname*)

uruchom unittest

Unittest sprawdza czy grafy spełniają warunki izomorfizmu i czy są izomorficzny

Parametry

gX	graf pierwszy
gY	graf drugi
meets	czy grafy powinny spełniać warunki
izom	czy grafy powinny być izomorficzne
nr	numer testu
testname	nazwa testu

Zwraca

czy test wykonał się poprawnie

4.6 Dokumentacja pliku utils.cpp

implementacja metod klasy pomocniczych

```
#include "utils.hpp"
```

Funkcje

- std::vector< std::string > & split (const std::string &s, char delim, std::vector< std::string > &elems)

  dzieli string na podstringi
- std::vector< std::string > split (const std::string &s, char delim)

  dźieli string na podstringi
- std::string & strip_space (std::string &s)

  usuwa niewidzialne znaki ze stringa

4.6.1 Opis szczegółowy

implementacja metod klasy pomocniczych

Detailed description starts here.

4.6.2 Dokumentacja funkcji

4.6.2.1 std::vector<std::string>& split (const std::string & *s*, char *delim*, std::vector< std::string > & *elems*)

dzieli string na podstringi

dzielenie stringa na podstawie znaku dzielącego

Parametry

s	string wejściowy
delim	znak podziału
elems	wektor z postringami wynikowymi

Zwraca

referencja na wektor elems

4.6.2.2 std::vector<std::string> split (const std::string & *s*, char *delim*)

dźieli string na podstringi

dzielenie stringa na podstawie znaku dzielącego

Parametry

s	string wejściowy
delim	znak podziału

Zwraca

wektor podstringów

4.6.2.3 std::string& strip_space (std::string & *s*)

usuwa niewidzialne znaki ze stringa

usuwanie znaku spacji, tabulacji, nowej linii ze stringa

Parametry

s	string wejściowy

Zwraca

string s bez białych znaków

4.7 Dokumentacja pliku utils.hpp

deklaracja funkcji pomocniczych

```
#include <vector>
#include <string>
#include <algorithm>
#include <functional>
#include <iostream>
#include <sstream>
```

Funkcje

- std::vector< std::string > & split (const std::string &s, char delim, std::vector< std::string > &elems)
  dzieli string na podstringi
- std::vector< std::string > split (const std::string &s, char delim)
  dźieli string na podstringi
- std::string & strip_space (std::string &s)
  usuwa niewidzialne znaki ze stringa

4.7.1 Opis szczegółowy

deklaracja funkcji pomocniczych Detailed description starts here.

Funkcje pomocnicze, służące do obsługi stringów

4.7.2 Dokumentacja funkcji

4.7.2.1 std::vector<std::string>& split (const std::string & *s*, char *delim*, std::vector< std::string > & *elems*)

dzieli string na podstringi

dzielenie stringa na podstawie znaku dzielącego

Parametry

s	string wejściowy
delim	znak podziału
elems	wektor z postringami wynikowymi

Zwraca

referencja na wektor elems

4.7.2.2 std::vector<std::string> split (const std::string & *s*, char *delim*)

dźieli string na podstringi

dzielenie stringa na podstawie znaku dzielącego

Parametry

s	string wejściowy
delim	znak podziału

Zwraca

wektor podstringów

4.7.2.3 std::string& strip_space (std::string & *s*)

usuwa niewidzialne znaki ze stringa

usuwanie znaku spacji, tabulacji, nowej linii ze stringa

Parametry

s	string wejściowy

Zwraca

string s bez białych znaków

4.8 Dokumentacja pliku vertex.cpp

implementacja metod klasy Vertex

```
#include "vertex.hpp"
```

Definicje typów

- typedef Vertex::idx_t idx_t

4.8.1 Opis szczegółowy

implementacja metod klasy Vertex Detailed description starts here.

4.8.2 Dokumentacja definicji typów

4.8.2.1 typedef Vertex::idx_t idx_t

4.9 Dokumentacja pliku vertex.hpp

plik nagłówkowy klasy Vertex

```
#include <set>
#include <string>
#include <iostream>
#include <sstream>
```

Komponenty

- class Vertex

  klasa reprezentuje wierzchołek grafie w reprezentacji list sąsiedztwa

4.9.1 Opis szczegółowy

plik nagłówkowy klasy Vertex Detailed description starts here.

Deklaracja klasy Vertex

Wierzchołek w grafie w reprezentacji list sąsiedztwa

Skorowidz