

Izomorfizm grafów

Wygenerowano przez Doxygen 1.8.6

So, 4 sty 2014 20:25:22

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 klasy Graph::AdjIter	5
3.1.1	Opis szczegółowy	5
3.1.2	Dokumentacja składowych definicji typu	6
3.1.2.1	Map	6
3.1.3	Dokumentacja konstruktora i destruktora	6
3.1.3.1	AdjIter	6
3.1.4	Dokumentacja funkcji składowych	6
3.1.4.1	operator!=	6
3.1.4.2	operator*	6
3.1.4.3	operator++	6
3.1.4.4	operator++	6
3.1.4.5	operator->	6
3.1.4.6	operator==	6
3.1.5	Dokumentacja atrybutów składowych	6
3.1.5.1	idx_label_map	6
3.1.5.2	vit	6
3.2	Dokumentacja struktury Graph::Edge	6
3.2.1	Opis szczegółowy	7
3.2.2	Dokumentacja konstruktora i destruktora	7
3.2.2.1	Edge	7
3.2.3	Dokumentacja funkcji składowych	7
3.2.3.1	operator<	7
3.2.3.2	operator==	7
3.2.4	Dokumentacja atrybutów składowych	8
3.2.4.1	source	8

3.2.4.2	target	8
3.3	Dokumentacja klasy IsomorphismAlgo::EdgeComparator	8
3.3.1	Opis szczegółowy	8
3.3.2	Dokumentacja konstruktora i destruktora	8
3.3.2.1	EdgeComparator	8
3.3.3	Dokumentacja funkcji składowych	8
3.3.3.1	operator()	8
3.3.4	Dokumentacja atrybutów składowych	9
3.3.4.1	dfs_num	9
3.4	Dokumentacja klasy Graph	9
3.4.1	Opis szczegółowy	12
3.4.2	Dokumentacja składowych definicji typu	12
3.4.2.1	dfs_path	12
3.4.2.2	dfs_visited	12
3.4.2.3	edge_set_t	12
3.4.2.4	iterator	12
3.4.2.5	label_t	12
3.4.2.6	vertex_set_t	12
3.4.3	Dokumentacja konstruktora i destruktora	13
3.4.3.1	Graph	13
3.4.4	Dokumentacja funkcji składowych	13
3.4.4.1	addEdge	13
3.4.4.2	addVertex	13
3.4.4.3	adjBegin	13
3.4.4.4	adjEnd	13
3.4.4.5	begin	14
3.4.4.6	clear	14
3.4.4.7	dumpVertex	14
3.4.4.8	end	14
3.4.4.9	findNextFreeldx	14
3.4.4.10	generateRandom	14
3.4.4.11	getDFSPath	14
3.4.4.12	getEdgeCount	15
3.4.4.13	getEdges	15
3.4.4.14	getEdges	15
3.4.4.15	getEdges	15
3.4.4.16	getIn	16
3.4.4.17	getIn	17
3.4.4.18	getIndex	17
3.4.4.19	getInfo	17

3.4.4.20	getInvariant	17
3.4.4.21	getLabel	18
3.4.4.22	getOut	19
3.4.4.23	getOut	19
3.4.4.24	getSize	19
3.4.4.25	getVertexAt	19
3.4.4.26	getVertexCount	20
3.4.4.27	isConnection	20
3.4.4.28	isConnection	20
3.4.4.29	isEmpty	20
3.4.4.30	isNode	20
3.4.4.31	loadFromFile	20
3.4.4.32	loadVertex	21
3.4.4.33	randomIsomorphic	21
3.4.4.34	saveToFile	21
3.4.4.35	setVertex	21
3.4.4.36	setVertex	21
3.4.5	Dokumentacja przyjaciół i funkcji związanych	22
3.4.5.1	operator<<	22
3.4.5.2	operator>>	22
3.4.6	Dokumentacja atrybutów składowych	22
3.4.6.1	__first_free_idx	22
3.4.6.2	__inv_power	22
3.4.6.3	edge_count	22
3.4.6.4	idx_label_map	22
3.4.6.5	label_idx_map	22
3.4.6.6	labels	22
3.4.6.7	vertex_count	22
3.4.6.8	vertexes	22
3.5	Dokumentacja klasy IsomorphismAlgo	23
3.5.1	Opis szczegółowy	24
3.5.2	Dokumentacja składowych definicji typu	24
3.5.2.1	dfs_idx_t	24
3.5.2.2	dfs_num_t	25
3.5.2.3	dfs_vec_t	25
3.5.2.4	edge_iter	25
3.5.2.5	iso_map	25
3.5.3	Dokumentacja konstruktora i destruktor	25
3.5.3.1	IsomorphismAlgo	25
3.5.4	Dokumentacja funkcji składowych	25

3.5.4.1	countInvBuckets	25
3.5.4.2	getInfo	25
3.5.4.3	getIsoMap	25
3.5.4.4	isIsomorphism	26
3.5.4.5	match	26
3.5.4.6	meetsRequirements	26
3.5.4.7	numberVertexes	27
3.5.4.8	orderEdges	27
3.5.4.9	resetData	27
3.5.4.10	verifyIsomorphism	27
3.5.5	Dokumentacja atrybutów składowych	27
3.5.5.1	dfs_num	27
3.5.5.2	dfs_vec	27
3.5.5.3	edges_count_k	27
3.5.5.4	f_map	27
3.5.5.5	graphX	28
3.5.5.6	graphY	28
3.5.5.7	in_S	28
3.5.5.8	invX_buckets	28
3.5.5.9	invY_buckets	28
3.5.5.10	ordered_edges	28
3.6	Dokumentacja klasy Vertex	28
3.6.1	Opis szczegółowy	29
3.6.2	Dokumentacja składowych definicji typu	29
3.6.2.1	deg_t	29
3.6.2.2	idx_t	29
3.6.2.3	iterator	30
3.6.3	Dokumentacja konstruktora i destruktor	30
3.6.3.1	Vertex	30
3.6.4	Dokumentacja funkcji składowych	30
3.6.4.1	addAdjacent	30
3.6.4.2	addNeighbour	30
3.6.4.3	begin	30
3.6.4.4	end	30
3.6.4.5	getIn	30
3.6.4.6	getIndex	31
3.6.4.7	getInfo	31
3.6.4.8	getOut	31
3.6.4.9	isAdjacent	31
3.6.5	Dokumentacja atrybutów składowych	31

3.6.5.1	adjacent	31
3.6.5.2	degree	31
3.6.5.3	index	32
4	Dokumentacja plików	33
4.1	Dokumentacja pliku graph.cpp	33
4.1.1	Opis szczegółowy	33
4.1.2	Dokumentacja definicji typów	33
4.1.2.1	idx_t	33
4.1.2.2	label_t	33
4.1.3	Dokumentacja funkcji	33
4.1.3.1	operator<<	33
4.1.3.2	operator>>	33
4.2	Dokumentacja pliku graph.hpp	33
4.2.1	Opis szczegółowy	34
4.2.2	Dokumentacja definicji	34
4.2.2.1	COM_SIGN	34
4.2.2.2	DELIMITER_CHAR	35
4.2.2.3	MAX_RANDOM_FAILS	35
4.2.3	Dokumentacja funkcji	35
4.2.3.1	operator<<	35
4.2.3.2	operator>>	35
4.3	Dokumentacja pliku isomorphismAlgo.cpp	35
4.3.1	Opis szczegółowy	35
4.4	Dokumentacja pliku isomorphismAlgo.hpp	35
4.4.1	Opis szczegółowy	35
4.5	Dokumentacja pliku izomorf.cpp	36
4.5.1	Opis szczegółowy	36
4.5.2	Dokumentacja funkcji	36
4.5.2.1	checkIsomorphism	36
4.5.2.2	executeFromFiles	37
4.5.2.3	executeRandom	37
4.5.2.4	executeTests	37
4.5.2.5	helpMsg	37
4.5.2.6	main	37
4.5.2.7	parseInput	37
4.5.2.8	readGraph	38
4.5.2.9	runFileTest	38
4.5.2.10	runRandomTest	38
4.5.2.11	runTestUnit	38

4.6	Dokumentacja pliku <code>utils.cpp</code>	39
4.6.1	Opis szczegółowy	39
4.6.2	Dokumentacja funkcji	39
4.6.2.1	<code>split</code>	39
4.6.2.2	<code>split</code>	40
4.6.2.3	<code>strip_space</code>	40
4.7	Dokumentacja pliku <code>utils.hpp</code>	40
4.7.1	Opis szczegółowy	40
4.7.2	Dokumentacja funkcji	41
4.7.2.1	<code>split</code>	41
4.7.2.2	<code>split</code>	41
4.7.2.3	<code>strip_space</code>	41
4.8	Dokumentacja pliku <code>vertex.cpp</code>	41
4.8.1	Opis szczegółowy	42
4.8.2	Dokumentacja definicji typów	42
4.8.2.1	<code>idx_t</code>	42
4.9	Dokumentacja pliku <code>vertex.hpp</code>	42
4.9.1	Opis szczegółowy	42
Indeks		43

Rozdział 1

Indeks klas

1.1 Lista klas

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

Graph::AdjIter	Klasa pomocnicza, iterator po znacznikach wierzchołków sąsiadujących z wierzchołkiem . . .	5
Graph::Edge	Struktura reprezentuje krawędź skierowaną w grafie	6
IsomorphismAlgo::EdgeComparator	Komparator do sortowania krawędzi w DFS lesie	8
Graph	Klasa implementuje graf w reprezentacji listowej	9
IsomorphismAlgo	Klasa reprezentuje algorytm do weryfikacji izomorfizmu grafów	23
Vertex	Klasa reprezentuje wierzchołek grafie w reprezentacji list sąsiedztwa	28

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

graph.cpp	Implementacja metod klasy Graph	33
graph.hpp	Plik nagłówkowy klasy Graph	33
isomorphismAlgo.cpp	Implementacja metod klasy IsomorphismAlgo	35
isomorphismAlgo.hpp	Plik nagłówkowy klasy IsomorphismAlgo	35
izomorf.cpp	Plik z metodą main	36
utils.cpp	Implementacja metod klasy pomocniczych	39
utils.hpp	Deklaracja funkcji pomocniczych	40
vertex.cpp	Implementacja metod klasy Vertex	41
vertex.hpp	Plik nagłówkowy klasy Vertex	42

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

<code>_idx_label_map</code>	mapa do mapowania
<code>_vit</code>	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 destruktor

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

konstruktor struktury [Edge](#)

Parametry

<code>_source</code>	znacznik wierzchołka źródłowego
<code>_target</code>	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

<code>e</code>	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

<code>e</code>	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 lasie

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 lasie

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

<code>_dfs_num</code>	mapowanie typu <code>IsomorphismAlgo::dfs_num_t</code>
-----------------------	--

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

<i>left</i>	pierwsza krawędź
<i>right</i>	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 włąb (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. Wykorzystywana do porównywania krawędzi np. przy sortowaniu. ścieżka dfs po grafie

Wykorzystywane przy przechodzeniu grafu wgląd

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

zbiór wcześniej odwiedzonych wierzchołków

Wykorzystywane przy przechodzeniu grafu wgląd

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

<i>v</i>	znacznik wierzchołka źródłowego
<i>w</i>	znacznik wierzchołka docelowego

Zwraca

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

3.4.4.2 bool Graph::addVertex (label_t label)

dodaj wierzchołek do grafu

Parametry

<i>label</i>	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

iterador wierzchołków incydentnych

Parametry

<i>label</i>	znacznik wierzchołka źródłowego
--------------	---------------------------------

Zwraca

iterador wskazujący na początek znaczników wierzchołków incydentnych z wierzchołkiem label

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

iterador wierzchołków incydentnych

Parametry

<i>label</i>	znacznik wierzchołka źródłowego
--------------	---------------------------------

Zwraca

iterador 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

<code>v</code>	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::findNextFreeldx () [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

<code>vertex_count</code>	liczba wierzchołków w grafie losowym
<code>density</code>	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 wgłęb (DFS)

Parametry

<i>path</i>	referencja na ścieżkę
<i>start</i>	znacznik wierzchołka startowego
<i>visited</i>	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

<i>edges</i>	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

<i>edges</i>	referencja na zbiór krawędzi
<i>label</i>	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

<i>edges</i>	referencja na zbiór krawędzi
<i>verts</i>	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

<i>label</i>	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

<i>label</i>	znacznik wierzchołka
<i>vset</i>	podzbiór wierzchołków grafu

Zwraca

wejściowość wierzchołka *label* w grafie indukowanym zbiorem *vset*

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

zwróć indeks dla wierzchołka o znaczniku *label*

Parametry

<i>label</i>	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

<i>label</i>	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

<i>idx</i>	indeks wierzchołka
------------	--------------------

Zwraca

znacznik wierzchołka *idx*

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

zwróć wyjściowość wierzchołka

Parametry

<i>label</i>	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

<i>label</i>	znacznik wierzchołka
<i>vset</i>	podzbiór wierzchołków grafu

Zwraca

wyjściowość wierzchołka *label* w grafie indukowanym 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

<i>label</i>	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

<i>v</i>	znacznik wierzchołka źródłowego
<i>w</i>	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

<i>v</i>	znacznik wierzchołka źródłowego
<i>w</i>	znacznik wierzchołka docelowego
<i>vset</i>	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

<i>label</i>	znacznik wierzchołka
--------------	----------------------

Zwraca

czy graf zawiera wierzchołek label

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

wczytaj graf z pliku

Parametry

<i>filename</i>	nazwa pliku
-----------------	-------------

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

wczyta wierzchołek ze stringa

Parametry

<i>str</i>	string z wierzchołkiem
------------	------------------------

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

twórz losowy graf izomorficzny

Parametry

<i>other</i>	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

<i>filename</i>	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

<i>label</i>	znacznik wierzchołka
<i>adj</i>	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

<i>label</i>	znacznik wierzchołka
<i>adj</i>	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

$10^t * \text{wyjściowość} + \text{wejściowość}$

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 licznosc 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

<code>_graphX</code>	graf X
<code>_graphY</code>	graf Y

3.5.4 Dokumentacja funkcji składowych

3.5.4.1 `void IsomorphismAlgo::countInvBuckets () [private]`

wyznacza licznosc 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 $G_X[k] \sim G_Y[S]$.

Procedura przechodzi przez wierzchołki DFS lasu. Porusza się po odpowiednio posortowanych krawędziach grafu [IsomorphismAlgo::graphX](#). Gdy uda się przejść wszystkie krawędzie grafu [IsomorphismAlgo::graphX](#), procedura kończy się sukcesem.

Parametry

<i>iter</i>	iterator po posortowanych krawędziach grafu IsomorphismAlgo::graphX , czyli po IsomorphismAlgo::dfs_num .
<i>dfs_num_k</i>	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 komparator [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 $f: X \rightarrow Y$ 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

<code>_index</code>	indeks dla tworzonego wierzchołka
---------------------	-----------------------------------

3.6.4 Dokumentacja funkcji składowych

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

dodaj wierzchołek sąsiadujący

Parametry

<code>adj</code>	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

<i>gX</i>	graf pierwszy
<i>gY</i>	graf drugi

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

uruchom program na grafach zapisanych w plikach

Parametry

<i>filenameX</i>	plik z grafem 1
<i>filenameY</i>	plik z grafem 2

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

uruchom program na losowych izomorficznych grafach

Parametry

<i>v</i>	ilość wierzchołków w grafach losowych
<i>d</i>	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

<i>argc</i>	ilość parametrów programu
<i>argv</i> []	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

<i>argc</i>	liczba zmiennych
<i>argv[]</i>	tablica argumentów

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

wczytuje z pliku, ew. wypisuje komunikat błędu

Parametry

<i>g</i>	wczytywany graf
<i>filename</i>	nazwa pliku z grafem
<i>graphname</i>	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

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

<i>v</i>	ilosc wierzchołków w grafach losowych
<i>d</i>	gęstość krawędzi w grafach losowych
<i>izom</i>	czy grafy powinny być izomorficzne
<i>nr</i>	numer testu
<i>testname</i>	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

<i>gX</i>	graf pierwszy
<i>gY</i>	graf drugi
<i>meets</i>	czy grafy powinny spełniać warunki
<i>izom</i>	czy grafy powinny być izomorficzne
<i>nr</i>	numer testu
<i>testname</i>	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)`
dzieli 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

<i>s</i>	string wejściowy
<i>delim</i>	znak podziału
<i>elems</i>	wektor z postringami wynikowymi

Zwraca

referencja na wektor elems

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

dzieli string na podstringi

dzielenie stringa na podstawie znaku dzielącego

Parametry

<code>s</code>	string wejściowy
<code>delim</code>	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

<code>s</code>	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)`
dzieli 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

<i>s</i>	string wejściowy
<i>delim</i>	znak podziału
<i>elems</i>	wektor z postringami wynikowymi

Zwraca

referencja na wektor elems

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

dzieli string na podstringi

dzielenie stringa na podstawie znaku dzielącego

Parametry

<i>s</i>	string wejściowy
<i>delim</i>	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

<i>s</i>	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

__first_free_idx
Graph, [22](#)

__inv_power
Graph, [22](#)

addAdjacent
Vertex, [30](#)

addEdge
Graph, [13](#)

addNeighbour
Vertex, [30](#)

addVertex
Graph, [13](#)

adjBegin
Graph, [13](#)

adjEnd
Graph, [13](#)

AdjIter
Graph::AdjIter, [6](#)

adjacent
Vertex, [31](#)

begin
Graph, [13](#)
Vertex, [30](#)

COM_SIGN
graph.hpp, [34](#)

checkIsomorphism
izomorf.cpp, [36](#)

clear
Graph, [14](#)

countInvBuckets
IsomorphismAlgo, [25](#)

DELIMITER_CHAR
graph.hpp, [34](#)

deg_t
Vertex, [29](#)

degree
Vertex, [31](#)

dfs_idx_t
IsomorphismAlgo, [24](#)

dfs_num
IsomorphismAlgo, [27](#)
IsomorphismAlgo::EdgeComparator, [9](#)

dfs_num_t
IsomorphismAlgo, [24](#)

dfs_path
Graph, [12](#)

dfs_vec
IsomorphismAlgo, [27](#)

dfs_vec_t
IsomorphismAlgo, [25](#)

dfs_visited
Graph, [12](#)

dumpVertex
Graph, [14](#)

Edge
Graph::Edge, [7](#)

edge_count
Graph, [22](#)

edge_iter
IsomorphismAlgo, [25](#)

edge_set_t
Graph, [12](#)

EdgeComparator
IsomorphismAlgo::EdgeComparator, [8](#)

edges_count_k
IsomorphismAlgo, [27](#)

end
Graph, [14](#)
Vertex, [30](#)

executeFromFiles
izomorf.cpp, [37](#)

executeRandom
izomorf.cpp, [37](#)

executeTests
izomorf.cpp, [37](#)

f_map
IsomorphismAlgo, [27](#)

findNextFreeIdx
Graph, [14](#)

generateRandom
Graph, [14](#)

getDFSPath
Graph, [14](#)

getEdgeCount
Graph, [15](#)

getEdges
Graph, [15](#)

getIn
Graph, [15](#), [17](#)
Vertex, [30](#)

getIndex
Graph, [17](#)
Vertex, [31](#)

- getInfo
 - Graph, 17
 - IsomorphismAlgo, 25
 - Vertex, 31
- getInvariant
 - Graph, 17
- getIsoMap
 - IsomorphismAlgo, 25
- getLabel
 - Graph, 17
- getOut
 - Graph, 19
 - Vertex, 31
- getSize
 - Graph, 19
- getVertexAt
 - Graph, 19
- getVertexCount
 - Graph, 19
- Graph, 9
 - __first_free_idx, 22
 - __inv_power, 22
 - addEdge, 13
 - addVertex, 13
 - adjBegin, 13
 - adjEnd, 13
 - begin, 13
 - clear, 14
 - dfs_path, 12
 - dfs_visited, 12
 - dumpVertex, 14
 - edge_count, 22
 - edge_set_t, 12
 - end, 14
 - findNextFreeldx, 14
 - generateRandom, 14
 - getDFSPATH, 14
 - getEdgeCount, 15
 - getEdges, 15
 - getIn, 15, 17
 - getIndex, 17
 - getInfo, 17
 - getInvariant, 17
 - getLabel, 17
 - getOut, 19
 - getSize, 19
 - getVertexAt, 19
 - getVertexCount, 19
 - Graph, 13
 - idx_label_map, 22
 - isConnection, 20
 - isEmpty, 20
 - isNode, 20
 - iterator, 12
 - label_idx_map, 22
 - label_t, 12
 - labels, 22
 - loadFromFile, 20
 - loadVertex, 21
 - operator<<, 22
 - operator>>, 22
 - randomIsomorphic, 21
 - saveToFile, 21
 - setVertex, 21
 - vertex_count, 22
 - vertex_set_t, 12
 - vertexes, 22
- graph.cpp, 33
 - idx_t, 33
 - label_t, 33
 - operator<<, 33
 - operator>>, 33
- graph.hpp, 33
 - COM_SIGN, 34
 - DELIMITER_CHAR, 34
 - MAX_RANDOM_FAILS, 35
 - operator<<, 35
 - operator>>, 35
- Graph::AdjIter, 5
 - AdjIter, 6
 - idx_label_map, 6
 - Map, 6
 - operator*, 6
 - operator++, 6
 - operator->, 6
 - operator==, 6
 - vit, 6
- Graph::Edge, 6
 - Edge, 7
 - operator<, 7
 - operator==, 7
 - source, 8
 - target, 8
- graphX
 - IsomorphismAlgo, 27
- graphY
 - IsomorphismAlgo, 28
- helpMsg
 - izomorf.cpp, 37
- idx_label_map
 - Graph, 22
 - Graph::AdjIter, 6
- idx_t
 - graph.cpp, 33
 - Vertex, 29
 - vertex.cpp, 42
- in_S
 - IsomorphismAlgo, 28
- index
 - Vertex, 31
- invX_buckets
 - IsomorphismAlgo, 28
- invY_buckets
 - IsomorphismAlgo, 28
- isAdjacent

- Vertex, 31
- isConnection
 - Graph, 20
- isEmpty
 - Graph, 20
- isIsomorphism
 - IsomorphismAlgo, 25
- isNode
 - Graph, 20
- iso_map
 - IsomorphismAlgo, 25
- IsomorphismAlgo, 23
 - countInvBuckets, 25
 - dfs_idx_t, 24
 - dfs_num, 27
 - dfs_num_t, 24
 - dfs_vec, 27
 - dfs_vec_t, 25
 - edge_iter, 25
 - edges_count_k, 27
 - f_map, 27
 - getInfo, 25
 - getIsoMap, 25
 - graphX, 27
 - graphY, 28
 - in_S, 28
 - invX_buckets, 28
 - invY_buckets, 28
 - isIsomorphism, 25
 - iso_map, 25
 - IsomorphismAlgo, 25
 - IsomorphismAlgo, 25
 - match, 26
 - meetsRequirements, 26
 - numberVertexes, 26
 - orderEdges, 27
 - ordered_edges, 28
 - resetData, 27
 - verifyIsomorphism, 27
- isomorphismAlgo.cpp, 35
- isomorphismAlgo.hpp, 35
- IsomorphismAlgo::EdgeComparator, 8
 - dfs_num, 9
 - EdgeComparator, 8
 - operator(), 8
- iterator
 - Graph, 12
 - Vertex, 29
- izomorf.cpp, 36
 - checkIsomorphism, 36
 - executeFromFiles, 37
 - executeRandom, 37
 - executeTests, 37
 - helpMsg, 37
 - main, 37
 - parseInput, 37
 - readGraph, 38
 - runFileTest, 38
 - runRandomTest, 38
 - runTestUnit, 38
- label_idx_map
 - Graph, 22
- label_t
 - Graph, 12
 - graph.cpp, 33
- labels
 - Graph, 22
- loadFromFile
 - Graph, 20
- loadVertex
 - Graph, 21
- MAX_RANDOM_FAILS
 - graph.hpp, 35
- main
 - izomorf.cpp, 37
- Map
 - Graph::AdjIter, 6
- match
 - IsomorphismAlgo, 26
- meetsRequirements
 - IsomorphismAlgo, 26
- numberVertexes
 - IsomorphismAlgo, 26
- operator<
 - Graph::Edge, 7
- operator<<
 - Graph, 22
 - graph.cpp, 33
 - graph.hpp, 35
- operator>>
 - Graph, 22
 - graph.cpp, 33
 - graph.hpp, 35
- operator*
 - Graph::AdjIter, 6
- operator()
 - IsomorphismAlgo::EdgeComparator, 8
- operator++
 - Graph::AdjIter, 6
- operator->
 - Graph::AdjIter, 6
- operator==
 - Graph::AdjIter, 6
 - Graph::Edge, 7
- orderEdges
 - IsomorphismAlgo, 27
- ordered_edges
 - IsomorphismAlgo, 28
- parseInput
 - izomorf.cpp, 37
- randomIsomorphic
 - Graph, 21

- readGraph
 - izomorf.cpp, [38](#)
- resetData
 - IsomorphismAlgo, [27](#)
- runFileTest
 - izomorf.cpp, [38](#)
- runRandomTest
 - izomorf.cpp, [38](#)
- runTestUnit
 - izomorf.cpp, [38](#)
- saveToFile
 - Graph, [21](#)
- setVertex
 - Graph, [21](#)
- source
 - Graph::Edge, [8](#)
- split
 - utils.cpp, [39](#)
 - utils.hpp, [41](#)
- strip_space
 - utils.cpp, [40](#)
 - utils.hpp, [41](#)
- target
 - Graph::Edge, [8](#)
- utils.cpp, [39](#)
 - split, [39](#)
 - strip_space, [40](#)
- utils.hpp, [40](#)
 - split, [41](#)
 - strip_space, [41](#)
- verifyIsomorphism
 - IsomorphismAlgo, [27](#)
- Vertex, [28](#)
 - addAdjacent, [30](#)
 - addNeighbour, [30](#)
 - adjacent, [31](#)
 - begin, [30](#)
 - deg_t, [29](#)
 - degree, [31](#)
 - end, [30](#)
 - getIn, [30](#)
 - getIndex, [31](#)
 - getInfo, [31](#)
 - getOut, [31](#)
 - idx_t, [29](#)
 - index, [31](#)
 - isAdjacent, [31](#)
 - iterator, [29](#)
 - Vertex, [30](#)
- vertex.cpp, [41](#)
 - idx_t, [42](#)
- vertex.hpp, [42](#)
- vertex_count
 - Graph, [22](#)
- vertex_set_t
 - Graph, [12](#)
- vertexes
 - Graph, [22](#)
- vit
 - Graph::AdjIter, [6](#)