Graf - Branch & Bound

Wygenerowano przez Doxygen 1.8.6

N, 15 maj 2016 20:39:14

Spis treści

1 Indeks hierarchiczny

1.1 Hierarchia klas

Ta lista dziedziczenia posortowana jest z grubsza, choć nie całkowicie, alfabetycznie:

2 Indeks klas

2.1 Lista klas

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

3 Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

4 Dokumentacja klas

4.1 Dokumentacja klasy AdvancedStopwatch

Klasa implementująca rozbudowany stoper.

`#include <AdvancedStopwatch.hh>`

Diagram dziedziczenia dla AdvancedStopwatch

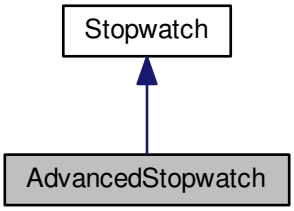
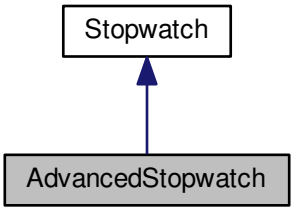


Diagram współpracy dla AdvancedStopwatch:



Metody publiczne

- AdvancedStopwatch ()
- ~AdvancedStopwatch ()
- unsigned & Rozmiar ()
- bool SaveElapsedTime (double rekord)

  Metoda zapisująca wartość pomiaru czasu okrążenia.
- double SeriesAverage ()

  Metoda wyliczająca średni czas okrążenia.
- bool SaveAverageTimeToBuffer (double rekord)

  Metoda zapisująca średni czas okrążenia do bufora plikowego.
- void PrintElapsedTimes ()

  Metoda wypisująca zawartość pamięci stopera.
- void CleanElapsedTimes ()

  Metoda usuwająca zawartość pamięci stopera.
- void CleanFileBuffer ()

  Metoda usuwająca zawartość bufora plikowego stopera.
- bool DumpFileBuffer (string nazwaPliku)

  Metoda zapisująca zawartość bufora plikowego do pliku.
- bool DumpToFile (string nazwaPliku, double rekord)

  Metoda zapisująca pojedynczy rekord bufora plikowego do pliku.

Dodatkowe Dziedziczone Składowe

4.1.1 Opis szczegółowy

Klasa implementująca rozbudowany stoper.

Klasa jest modelelem stopera z funkcją zapisu czasu okrążeń, liczeniem średniego czasu kilku okrążeń, zapisu zmierzonych czasów do pliku.

Definicja w linii 28 pliku AdvancedStopwatch.hh.

4.1.2 Dokumentacja konstruktora i destruktora

4.1.2.1 AdvancedStopwatch::AdvancedStopwatch ( )

Definicja w linii 4 pliku AdvancedStopwatch.cpp.

4.1.2.2 AdvancedStopwatch::∼AdvancedStopwatch ( )

Definicja w linii 15 pliku AdvancedStopwatch.cpp.

4.1.3 Dokumentacja funkcji składowych

4.1.3.1 void AdvancedStopwatch::CleanElapsedTimes ( )

Metoda usuwająca zawartość pamięci stopera.

Definicja w linii 66 pliku AdvancedStopwatch.cpp.

4.1.3.2 void AdvancedStopwatch::CleanFileBuffer ( )

Metoda usuwająca zawartość bufora plikowego stopera.

Definicja w linii 74 pliku AdvancedStopwatch.cpp.

4.1.3.3 bool AdvancedStopwatch::DumpFileBuffer (string *nazwaPliku*)

Metoda zapisująca zawartość bufora plikowego do pliku.

Dokonuje zapisu rekordów w buforze do pliku.

Parametry

>p0.10	>p0.15	p0.678
`in` *nazwaPliku*	- nazwa pliku, do którego mają zostać zapisane czasy

Zwracane wartości

>p0.25	p0.705
true	- jeśli udało się zapisać
false	- jeśli udało się zapisać

Definicja w linii 80 pliku AdvancedStopwatch.cpp.

4.1.3.4 bool AdvancedStopwatch::DumpToFile (string *nazwaPliku,* double *rekord*)

Metoda zapisująca pojedynczy rekord bufora plikowego do pliku.

Dokonuje zapisu wybranego rekordu w buforze do pliku.

Parametry

|>p0.10|>p0.15|p0.678|

`in` *nazwaPliku* - nazwa pliku, do którego ma zostać zapisany czas

`in` *rekord* - wartość pomiaru czasu, która ma być zapisana

Zwracane wartości

|>p0.25|p0.705|

true - jeśli udało się zapisać

false - jeśli udało się zapisać

Definicja w linii 98 pliku AdvancedStopwatch.cpp.

4.1.3.5 void AdvancedStopwatch::PrintElapsedTimes ( )

Metoda wypisująca zawartość pamięci stopera.

Definicja w linii 58 pliku AdvancedStopwatch.cpp.

4.1.3.6 unsigned& AdvancedStopwatch::Rozmiar ( ) [inline]

Definicja w linii 36 pliku AdvancedStopwatch.hh.

4.1.3.7 bool AdvancedStopwatch::SaveAverageTimeToBuffer (double *rekord*)

Metoda zapisująca średni czas okrążenia do bufora plikowego.

Dodaje podany czas do pamięci stopera, z której można dokonać zapisu do pliku.

Parametry

|>p0.10|>p0.15|p0.678|

`in` *rekord* - wartość pomiaru czasu

Zwracane wartości

|>p0.25|p0.705|

true - jeśli udało się zapisać

false - jeśli udało się zapisać

Definicja w linii 49 pliku AdvancedStopwatch.cpp.

4.1.3.8 bool AdvancedStopwatch::SaveElapsedTime (double *rekord*)

Metoda zapisująca wartość pomiaru czasu okrążenia.

Dodaje podany czas do tablicy czasów okrążeń.

Parametry

|>p0.10|>p0.15|p0.678|

`in` *rekord* - wartość pomiaru czasu

Zwracane wartości

|>p0.25|p0.705|

true - jeśli udało się zapisać

false - jeśli udało się zapisać

Definicja w linii 26 pliku AdvancedStopwatch.cpp.

4.1.3.9 double AdvancedStopwatch::SeriesAverage ( )

Metoda wyliczająca średni czas okrążenia.

Definicja w linii 37 pliku AdvancedStopwatch.cpp.

Dokumentacja dla tej klasy została wygenerowana z plików:

- prj/inc/AdvancedStopwatch.hh
- prj/src/AdvancedStopwatch.cpp

4.2 Dokumentacja szablonu klasy BList< Object >

Szablonowa klasa implementująca listę dwukierunkową

`#include <BList.hh>`

Diagram dziedziczenia dla BList< Object >

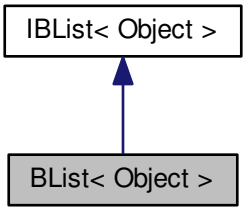
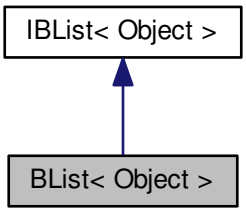


Diagram współpracy dla BList< Object >:



Metody publiczne

- BList ()

  Konstruktor listy dwukierunkowej.

- ~BList ()

  Destruktor listy dwukierunkowej.
- BNode< Object > *& Head ()

  Metoda zwracająca głowę listy.
- Object Front ()
- BNode< Object > *& Tail ()

  Metoda zwracająca ogon listy.
- Object Back ()
- BNode< Object > * Find (Object k)

  Metoda wyszukująca element na liście.
- virtual bool IsEmpty ()

  Metoda sprawdzająca, czy lista jest pusta.
- virtual void AddFront (const Object newItem)
- virtual void AddBack (const Object newItem)
- void AddAfter (BNode< Object > *p, const Object newItem)

  Metoda dodająca element we wskazane miejsce na liście.
- const Object Remove (BNode< Object > *p)

  Metoda usuwająca wskazany element listy.
- virtual const Object RemoveFront ()
- virtual const Object RemoveBack ()
- void Print ()
- void Clear ()

4.2.1 Opis szczegółowy

template<typename Object>class BList< Object >

Szablonowa klasa implementująca listę dwukierunkową

BList jest zbudowana w oparciu o węzły BNode oraz operacje na wskaźnikach.

BList może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 25 pliku BList.hh.

4.2.2 Dokumentacja konstruktora i destruktora

4.2.2.1 template<typename Object > BList< Object >::BList ( )

Konstruktor listy dwukierunkowej.

Inicjuje listę poprzez ustawienie wskaźnika NULL jako początek (head) tej listy.

Definicja w linii 115 pliku BList.hh.

4.2.2.2 template<typename Object > BList< Object >::~BList ( )

Destruktor listy dwukierunkowej.

Usuwa listę poprzez ustawienie wskaźnika NULL jako początek (head) tej listy.

Definicja w linii 119 pliku BList.hh.

4.2.3 Dokumentacja funkcji składowych

4.2.3.1 template<typename Object> void BList< Object >::AddAfter (BNode< Object > * *p*, const Object *newItem*)

Metoda dodająca element we wskazane miejsce na liście.

Alokuje nowy węzeł, dodaje nowy element, dodaje powiązanie tak,

Parametry

>p0.10	>p0.15	p0.678
in *newItem*	- element do dodania
in *p*	- docelowa pozycja elementu

Definicja w linii 187 pliku BList.hh.

4.2.3.2 template<typename Object> void BList< Object >::AddBack (const Object *newItem*) [virtual]

Implementuje IBList< Object >.

Definicja w linii 173 pliku BList.hh.

4.2.3.3 template<typename Object> void BList< Object >::AddFront (const Object *newItem*) [virtual]

Implementuje IBList< Object >.

Definicja w linii 157 pliku BList.hh.

4.2.3.4 template<typename Object > Object BList< Object >::Back ( )

Definicja w linii 150 pliku BList.hh.

4.2.3.5 template<typename Object > void BList< Object >::Clear ( )

Definicja w linii 258 pliku BList.hh.

4.2.3.6 template<typename Object> BNode< Object > * BList< Object >::Find (Object *k*)

Metoda wyszukująca element na liście.

Implementuje algorytm liniowego przeszukiwania listy.

Parametry

>p0.10	>p0.15	p0.678
in *k*	- element do wyszukania

Zwraca

Wskaźnik do znalezionego elementu lub NULL, gdy nie znaleziono.

Definicja w linii 235 pliku BList.hh.

4.2.3.7 template<typename Object > Object BList< Object >::Front ( )

Definicja w linii 143 pliku BList.hh.

4.2.3.8 template<typename Object > BNode< Object > *& BList< Object >::Head ( )

Metoda zwracająca głowę listy.

Zwraca wskaźnik do początku listy lub NULL, jeśli lista jest pusta.

Zwraca

Wskaźnik do głowy listy.

Definicja w linii 131 pliku BList.hh.

4.2.3.9 template<typename Object > bool BList< Object >::IsEmpty ( ) [virtual]

Metoda sprawdzająca, czy lista jest pusta.

Sprawdza, czy head wskazuje na coś innego niż NULL. Implementacja metody wirtualnej z interfejsu IList.

Zwraca

true - jeśli lista jest pusta, false - jeśli nie

Implementuje IBList< Object >.

Definicja w linii 125 pliku BList.hh.

4.2.3.10 template<typename Object > void BList< Object >::Print ( )

Definicja w linii 245 pliku BList.hh.

4.2.3.11 template<typename Object> const Object BList< Object >::Remove (BNode< Object > * p)

Metoda usuwająca wskazany element listy.

Uaktualnia head, aby wskazywał na kolejny element na liście, po czym usuwa stary węzeł.

Parametry

>p0.10	>p0.15	p0.678
in *p* - element do usunięcia	

Definicja w linii 199 pliku BList.hh.

4.2.3.12 template<typename Object > const Object BList< Object >::RemoveBack ( ) [virtual]

Implementuje IBList< Object >.

Definicja w linii 229 pliku BList.hh.

4.2.3.13 template<typename Object > const Object BList< Object >::RemoveFront ( ) [virtual]

Implementuje IBList< Object >.

Definicja w linii 223 pliku BList.hh.

4.2.3.14 template<typename Object > BNode< Object > *& BList< Object >::Tail ( )

Metoda zwracająca ogon listy.

Zwraca wskaźnik do końca listy lub NULL, jeśli lista jest pusta.

Zwraca

Wskaźnik do ogona listy.

Definicja w linii 137 pliku BList.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/BList.hh

4.3 Dokumentacja szablonu klasy BNode< Object >

`#include <BNode.hh>`

Metody publiczne

- BNode ()
- Object & element ()
- unsigned & index ()
- BNode< Object > *& next ()
- BNode< Object > *& prev ()

4.3.1 Opis szczegółowy

template<typename Object>class BNode< Object >

Definicja w linii 8 pliku BNode.hh.

4.3.2 Dokumentacja konstruktora i destruktora

4.3.2.1 template<typename Object> BNode< Object >::BNode ( ) [inline]

Definicja w linii 16 pliku BNode.hh.

4.3.3 Dokumentacja funkcji składowych

4.3.3.1 template<typename Object> Object& BNode< Object >::element ( ) [inline]

Definicja w linii 17 pliku BNode.hh.

4.3.3.2 template<typename Object> unsigned& BNode< Object >::index ( ) [inline]

Definicja w linii 18 pliku BNode.hh.

4.3.3.3 template<typename Object> BNode<Object>* & BNode< Object >::next ( ) [inline]

Definicja w linii 19 pliku BNode.hh.

4.3.3.4 template<typename Object> BNode<Object>* & BNode< Object >::prev ( ) [inline]

Definicja w linii 20 pliku BNode.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/BNode.hh

4.4 Dokumentacja klasy Edge

`#include <Edge.hh>`

Metody publiczne

- Edge ()
- Edge (unsigned start, unsigned end, unsigned w=1)
- unsigned & w ()

- unsigned & beg ()
- unsigned & end ()

4.4.1 Opis szczegółowy

Definicja w linii 7 pliku Edge.hh.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 Edge::Edge ( ) [inline]

Definicja w linii 14 pliku Edge.hh.

4.4.2.2 Edge::Edge (unsigned *start,* unsigned *end,* unsigned *w = 1*) [inline]

Definicja w linii 15 pliku Edge.hh.

4.4.3 Dokumentacja funkcji składowych

4.4.3.1 unsigned& Edge::beg ( ) [inline]

Definicja w linii 17 pliku Edge.hh.

4.4.3.2 unsigned& Edge::end ( ) [inline]

Definicja w linii 18 pliku Edge.hh.

4.4.3.3 unsigned& Edge::w ( ) [inline]

Definicja w linii 16 pliku Edge.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/Edge.hh

4.5 Dokumentacja klasy Graph

`#include <Graph.hh>`

Diagram dziedziczenia dla Graph

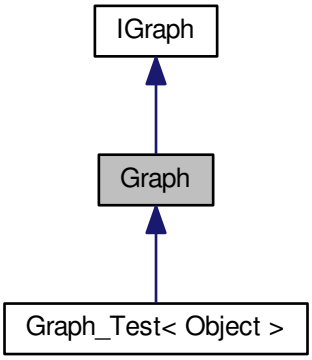
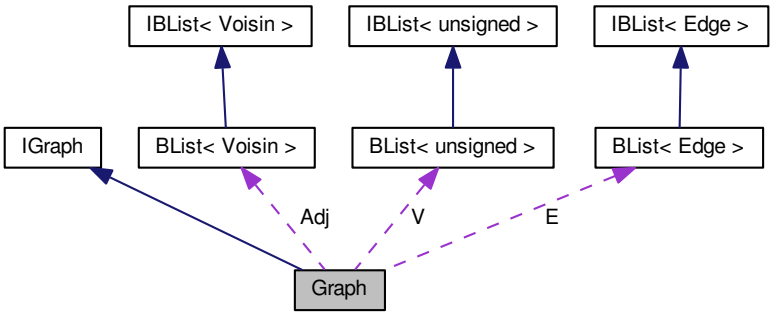


Diagram współpracy dla Graph:



Metody publiczne

- Graph ()

  Konstruktor grafu.
- Graph (int problemSize)

  Parametryczny konstruktor grafu.
- ∼Graph ()

  Destruktor grafu.
- BList< Voisin > * neighbours (unsigned i)

  Podaje listę sąsiedztwa danego wierzchołka.
- virtual bool areAdjacent (unsigned i, unsigned j)

  Sprawdza, czy wierzchołki są sąsiadujące Szuka wierzchołka j na liście sąsiedztwa wierzchołka i.
- virtual void insertVertex (unsigned i)

  Dodaje wierzchołek Dodaje do listy wierzchołków.
- virtual void insertEdge (unsigned i, unsigned j, unsigned w=1)

Dodaje krawędź Dodaje wpisy na listach sąsiedztwa dla obu wierzchołków.

- BList< unsigned > & vertices ()

  Zwraca listę wierzchołków.
- int & maxN ()

  Podaje maksymalną liczbę wierzchołków.
- void Print ()

  Drukuje krawędzie grafu.
- void BFS (unsigned i)

  Przechodzi graf wszerz Wykorzystuje implementację kolejki.
- void DFS (unsigned i)

  Przechodzi graf wgłąb Wykorzystuje implementację stosu.
- void BranchBound (unsigned i, unsigned finish)

  Wyszukuje najkrótszą ścieżkę w grafie metodą Branch&Bound.
- void BranchBoundExtendedList (unsigned i, unsigned finish)

  Wyszukuje najkrótszą ścieżkę w grafie metodą Branch&Bound with Extended List.

Atrybuty chronione

- BList< unsigned > V
- BList< Voisin > * Adj
- BList< Edge > E
- int N

4.5.1 Opis szczegółowy

Definicja w linii 19 pliku Graph.hh.

4.5.2 Dokumentacja konstruktora i destruktora

4.5.2.1 Graph::Graph ( )

Konstruktor grafu.

Inicjuje graf poprzez przypisanie tablicy list sąsiedztwa wskaźnika NULL.

Definicja w linii 137 pliku Graph.hh.

4.5.2.2 Graph::Graph (int *problemSize*)

Parametryczny konstruktor grafu.

Inicjuje graf poprzez utworzenie tablicy list sąsiedztwa dla podanej liczby wierzchołków.

Parametry

>p0.10	>p0.15	p0.678
`in` *problemSize* - liczba wierzchołków		

Definicja w linii 147 pliku Graph.hh.

4.5.2.3 Graph::~Graph ( )

Destruktor grafu.

Zwalnia pamięć zajmowaną przez tablicę list sąsiedztwa. Wywołuje destruktor listy wierzchołków.

Definicja w linii 153 pliku Graph.hh.

4.5.3 Dokumentacja funkcji składowych

4.5.3.1 bool Graph::areAdjacent (unsigned *i,* unsigned *j*) [virtual]

Sprawdza, czy wierzchołki są sąsiadujące Szuka wierzchołka j na liście sąsiedztwa wierzchołka i.

Parametry

>p0.10	>p0.15	p0.678
in *i* - wierzchołek pierwszy
in *j* - wierzchołek druga

Zwracane wartości

>p0.25	p0.705
true - jeśli są sąsiednie
false - jeśli nie są sąsiednie

Implementuje IGraph.

Definicja w linii 170 pliku Graph.hh.

4.5.3.2 void Graph::BFS (unsigned *i*)

Przechodzi graf wszerz Wykorzystuje implementację kolejki.

Parametry

>p0.10	>p0.15	p0.678
in *i* - numer wierzchołka startowego

Definicja w linii 249 pliku Graph.hh.

4.5.3.3 void Graph::BranchBound (unsigned *i,* unsigned *finish*)

Wyszukuje najkrótszą ścieżkę w grafie metodą Branch&Bound.

Parametry

>p0.10	>p0.15	p0.678
in *i* - numer wierzchołka startowego
in *finish* - numer wierzchołka końcowego

Definicja w linii 280 pliku Graph.hh.

4.5.3.4 void Graph::BranchBoundExtendedList (unsigned *i,* unsigned *finish*)

Wyszukuje najkrótszą ścieżkę w grafie metodą Branch&Bound with Extended List.

Parametry

>p0.10	>p0.15	p0.678
in *i* - numer wierzchołka startowego
in *finish* - numer wierzchołka końcowego

Definicja w linii 360 pliku Graph.hh.

4.5.3.5 void Graph::DFS (unsigned *i*)

Przechodzi graf wgłąb Wykorzystuje implementację stosu.

Parametry

>p0.10	>p0.15	p0.678
in *i* - numer wierzchołka startowego

Definicja w linii 214 pliku Graph.hh.

4.5.3.6 void Graph::insertEdge (unsigned *i*, unsigned *j*, unsigned *w = 1*) [virtual]

Dodaje krawędź Dodaje wpisy na listach sąsiedztwa dla obu wierzchołków.

Parametry

>p0.10	>p0.15	p0.678
in *i* - numer wierzchołka pierwszego
in *j* - numer wierzchołka drugiego
in *w* - waga krawędzi

Implementuje IGraph.

Definicja w linii 186 pliku Graph.hh.

4.5.3.7 void Graph::insertVertex (unsigned *i*) [virtual]

Dodaje wierzchołek Dodaje do listy wierzchołków.

Parametry

>p0.10	>p0.15	p0.678
in *i* - numer wierzchołka

Implementuje IGraph.

Definicja w linii 180 pliku Graph.hh.

4.5.3.8 int& Graph::maxN ( ) [inline]

Podaje maksymalną liczbę wierzchołków.

Zwraca

rozmiar tablicy list sąsiedztwa

Definicja w linii 99 pliku Graph.hh.

4.5.3.9 BList< Voisin > * Graph::neighbours (unsigned *i*)

Podaje listę sąsiedztwa danego wierzchołka.

Parametry

>p0.10	>p0.15	p0.678
in *i* - wierzchołek, którego lista sąsiedztwa ma być zwrócona

Zwraca

wskaźnik na lista sąsiedztwa podanego wierzchołka

Definicja w linii 162 pliku Graph.hh.

4.5.3.10 void Graph::Print ( )

Drukuje krawędzie grafu.

Definicja w linii 203 pliku Graph.hh.

4.5.3.11 BList<unsigned>& Graph::vertices ( ) [inline]

Zwraca listę wierzchołków.

Zwraca

lista sąsiedztwa podanego wierzchołka

Definicja w linii 93 pliku Graph.hh.

4.5.4 Dokumentacja atrybutów składowych

4.5.4.1 BList<Voisin>* Graph::Adj [protected]

-> lista wierzchołków

Definicja w linii 23 pliku Graph.hh.

4.5.4.2 BList<Edge> Graph::E [protected]

-> tablica list sąsiedztwa

Definicja w linii 24 pliku Graph.hh.

4.5.4.3 int Graph::N [protected]

-> lista krawędzi

Definicja w linii 25 pliku Graph.hh.

4.5.4.4 BList<unsigned> Graph::V [protected]

Definicja w linii 22 pliku Graph.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/Graph.hh

4.6 Dokumentacja szablonu klasy Graph_Test< Object >

Szablonowa klasa implementująca testowy graf.

`#include <Graph_Test.hh>`

Diagram dziedziczenia dla Graph_Test< Object >

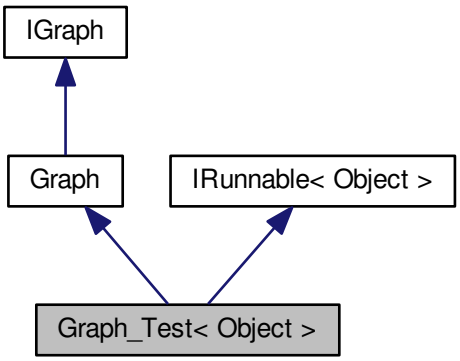
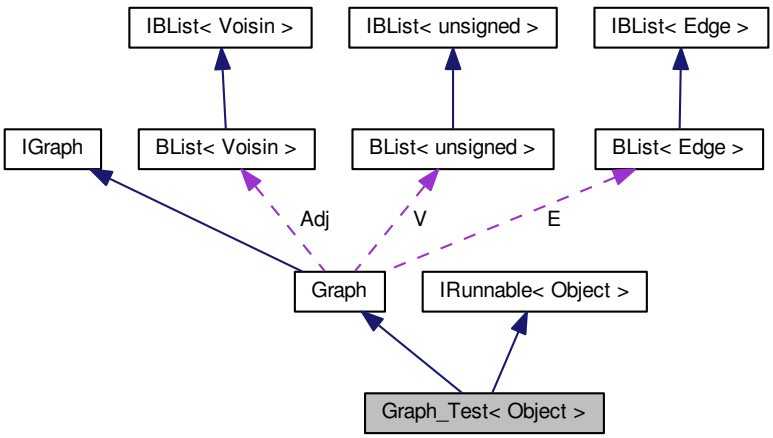


Diagram współpracy dla Graph_Test< Object >:



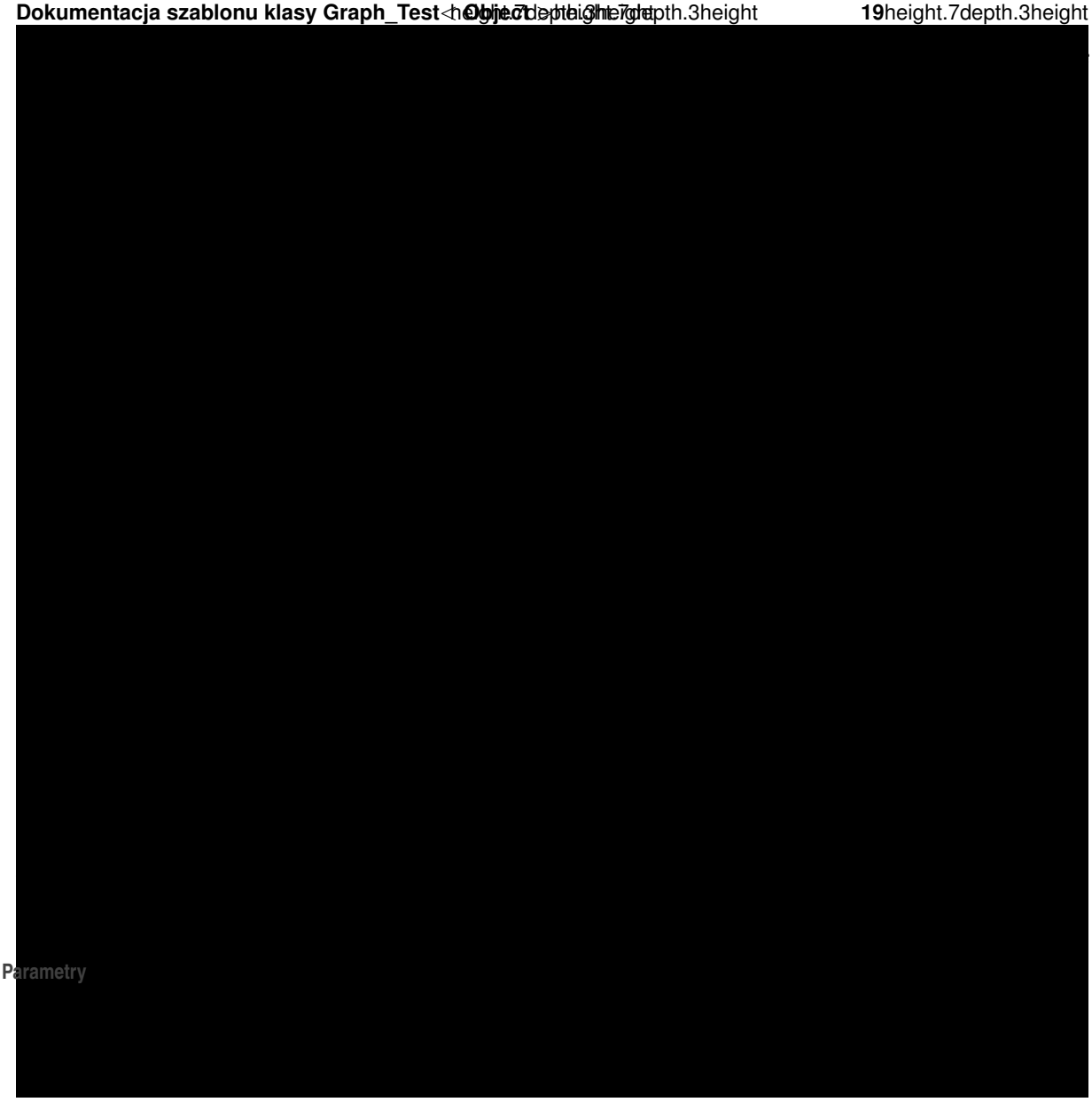
Metody publiczne

- void changeSearchType (char type)

  Zmienia sposób przechodzenia grafu.
- void RandomPathEnd ()

  Losuje końcowy wierzchołek.
- virtual bool Prepare (Object parametr)

  Metoda przygotowująca graf W zależności od podanej liczby, dodaje odpowiednio dużo wierzchołków do grafu i tworzy między nimi spójne powiązanie. W następujący sposób: tworzy tymczasową tablicę z numerami wierzchołków i następnie losowo zamienia komórki tablicy. Następnie dodaje krawędzie zgodnie z wylosowaną kolejnością. Na koniec generuje 2n losowych krawędzi.
- virtual bool Run ()



Parametry

Zwracane wartości

|>p0.25|p0.705|

true - jeśli operacja zakończyła się pomyślnie

false - jeśli wystąpił jakiś błąd

Implementuje IRunnable< Object >.

Definicja w linii 77 pliku Graph_Test.hh.

4.6.2.3 template<typename Object > void Graph_Test< Object >::RandomPathEnd ( )

Losuje końcowy wierzchołek.

Losuje wierzchołek w zakresie od 0 do N, do którego ma zostać znaleziona najkrótsza ścieżka.

Definicja w linii 71 pliku Graph_Test.hh.

4.6.2.4 template<typename Object > bool Graph_Test< Object >::Run ( ) [virtual]

Metoda uruchamiająca przejście grafu W zależności od ustawienia parametru searchType, uruchamia przejście BFS lub DFS.

Zwracane wartości

|>p0.25|p0.705|
---
true - jeśli operacja zakończyła się pomyślnie

---

false - jeśli wystąpił jakiś błąd

Implementuje IRunnable< Object >.

Definicja w linii 125 pliku Graph_Test.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

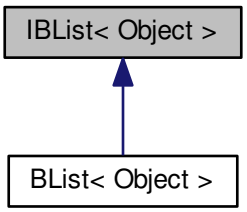
- prj/inc/Graph_Test.hh

4.7 Dokumentacja szablonu klasy IBList< Object >

Interfejs listy dwukierunkowej.

`#include <IBList.hh>`

Diagram dziedziczenia dla IBList< Object >



Metody publiczne

- virtual bool IsEmpty ()=0

  Metoda sprawdzająca, czy lista jest pusta.
- virtual void AddFront (const Object newItem)=0
- virtual void AddBack (const Object newItem)=0
- virtual const Object RemoveFront ()=0
- virtual const Object RemoveBack ()=0

4.7.1 Opis szczegółowy

template<typename Object>class IBList< Object >

Interfejs listy dwukierunkowej.

Definiuje ADT dla listy dwukierunkowej.

Lista może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 22 pliku IBList.hh.

4.7.2 Dokumentacja funkcji składowych

4.7.2.1 template<typename Object> virtual void IBList< Object >::AddBack (const Object *newItem*) [pure virtual]

Implementowany w BList< Object >, BList< Voisin >, BList< unsigned > i BList< Edge >.

4.7.2.2 template<typename Object> virtual void IBList< Object >::AddFront (const Object *newItem*) [pure virtual]

Implementowany w BList< Object >, BList< Voisin >, BList< unsigned > i BList< Edge >.

4.7.2.3 template<typename Object> virtual bool IBList< Object >::IsEmpty ( ) [pure virtual]

Metoda sprawdzająca, czy lista jest pusta.

true - jeśli lista jest pusta truefalse - jeśli nie jest pusta

Implementowany w BList< Object >, BList< Voisin >, BList< unsigned > i BList< Edge >.

4.7.2.4 template<typename Object> virtual const Object IBList< Object >::RemoveBack ( ) [pure virtual]

Implementowany w BList< Object >, BList< Voisin >, BList< unsigned > i BList< Edge >.

4.7.2.5 template<typename Object> virtual const Object IBList< Object >::RemoveFront ( ) [pure virtual]

Implementowany w BList< Object >, BList< Voisin >, BList< unsigned > i BList< Edge >.

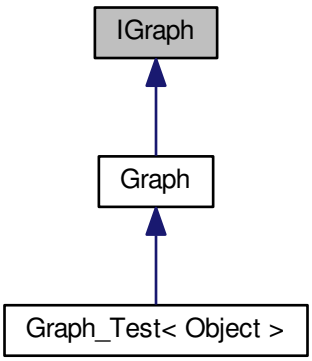
Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/IBList.hh

4.8 Dokumentacja klasy IGraph

`#include <IGraph.hh>`

Diagram dziedziczenia dla IGraph



Metody publiczne

- virtual bool areAdjacent (unsigned i, unsigned j)=0
- virtual void insertVertex (unsigned i)=0
- virtual void insertEdge (unsigned i, unsigned j, unsigned w=1)=0

4.8.1 Opis szczegółowy

Definicja w linii 13 pliku IGraph.hh.

4.8.2 Dokumentacja funkcji składowych

4.8.2.1 virtual bool IGraph::areAdjacent (unsigned *i,* unsigned *j*) [pure virtual]

Implementowany w Graph.

4.8.2.2 virtual void IGraph::insertEdge (unsigned *i,* unsigned *j,* unsigned *w = 1*) [pure virtual]

Implementowany w Graph.

4.8.2.3 virtual void IGraph::insertVertex (unsigned *i*) [pure virtual]

Implementowany w Graph.

Dokumentacja dla tej klasy została wygenerowana z pliku:

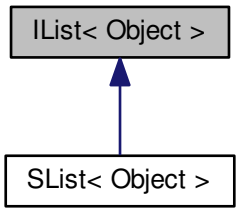
- prj/inc/IGraph.hh

4.9 Dokumentacja szablonu klasy IList< Object >

Interfejs listy jednokierunkowej.

`#include <IList.hh>`

Diagram dziedziczenia dla IList< Object >



Metody publiczne

- virtual bool IsEmpty ()=0

  Metoda sprawdzająca, czy lista jest pusta.
- virtual const Object & Front ()=0

  Metoda zwracająca pierwszy element listy.

- virtual void AddFront (const Object newItem)=0

  Metoda dodająca element na początek listy.
- virtual void RemoveFront ()=0

  Metoda usuwająca element z początku listy.

4.9.1 Opis szczegółowy

template<typename Object>class IList< Object >

Interfejs listy jednokierunkowej.

Definiuje ADT dla listy jednokierunkowej.

Lista może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 22 pliku IList.hh.

4.9.2 Dokumentacja funkcji składowych

4.9.2.1 template<typename Object > virtual void IList< Object >::AddFront (const Object *newItem*) [pure virtual]

Metoda dodająca element na początek listy.

Parametry

>p0.10	>p0.15	p0.678
`in` *newItem* - element do dodania

Implementowany w SList< Object >.

4.9.2.2 template<typename Object > virtual const Object& IList< Object >::Front ( ) [pure virtual]

Metoda zwracająca pierwszy element listy.

Zwraca

pierwszy element listy

Implementowany w SList< Object >.

4.9.2.3 template<typename Object > virtual bool IList< Object >::IsEmpty ( ) [pure virtual]

Metoda sprawdzająca, czy lista jest pusta.

Zwracane wartości

>p0.25	p0.705
true - jeśli lista jest pusta
truefalse - jeśli nie jest pusta

Implementowany w SList< Object >.

4.9.2.4 template<typename Object > virtual void IList< Object >::RemoveFront ( ) [pure virtual]

Metoda usuwająca element z początku listy.

Implementowany w SList< Object >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

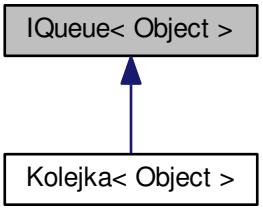
- prj/inc/IList.hh

4.10 Dokumentacja szablonu klasy IQueue< Object >

Klasa modelująca interfejs kolejki.

`#include <IQueue.hh>`

Diagram dziedziczenia dla IQueue< Object >



Metody publiczne

- virtual bool IsEmpty ()=0

  Metoda sprawdzająca, czy lista jest pusta.
- virtual Object Front ()=0

  Metoda zwracająca pierwszy element kolejki.
- virtual void Enqueue (Object item)=0

  Metoda dodająca element do kolejki.
- virtual Object Dequeue ()=0

  Metoda usuwająca element kolejki.

4.10.1 Opis szczegółowy

template<typename Object>class IQueue< Object >

Klasa modelująca interfejs kolejki.

Definiuje ADT dla kolejki.

Kolejka może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 22 pliku IQueue.hh.

4.10.2 Dokumentacja funkcji składowych

4.10.2.1 template<typename Object > virtual Object IQueue< Object >::Dequeue ( ) [pure virtual]

Metoda usuwająca element kolejki.

Usuwa element z początki kolejki.

Zwraca

pierwszy element kolejki

Implementowany w Kolejka< Object >.

4.10.2.2 template<typename Object > virtual void IQueue< Object >::Enqueue (Object *item*) [pure virtual]

Metoda dodająca element do kolejki.

Ustawia element na koniec kolejki.

Parametry

>p0.10	>p0.15	p0.678
`in` *item* - element do dodania

Implementowany w Kolejka< Object >.

4.10.2.3 template<typename Object > virtual Object IQueue< Object >::Front ( ) [pure virtual]

Metoda zwracająca pierwszy element kolejki.

Zwraca

pierwszy element kolejki

Implementowany w Kolejka< Object >.

4.10.2.4 template<typename Object > virtual bool IQueue< Object >::IsEmpty ( ) [pure virtual]

Metoda sprawdzająca, czy lista jest pusta.

true - jeśli lista jest pusta false - jeśli nie jest pusta

Implementowany w Kolejka< Object >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

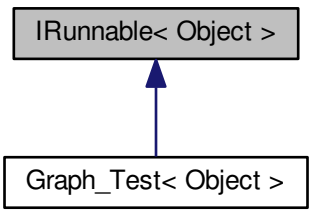
- prj/inc/IQueue.hh

4.11 Dokumentacja szablonu klasy IRunnable< Object >

Klasa szablonowa modelująca interfejs "Biegacza".

`#include <IRunnable.hh>`

Diagram dziedziczenia dla IRunnable< Object >



Metody publiczne

- virtual bool Prepare (Object parametr)=0

  Metoda przygotowująca obiekt do operacji.
- virtual bool Run ()=0

  Metoda uruchamiająca zdefiniowaną operację

4.11.1 Opis szczegółowy

template<typename Object>class IRunnable< Object >

Klasa szablonowa modelująca interfejs "Biegacza".

Klasa jest abstrakcyjnym uogólnieniem obiektu, na którym można wykonać zdefiniowane operacje, którym z kolei można zmierzyć czas wykonywania.

Definicja w linii 22 pliku IRunnable.hh.

4.11.2 Dokumentacja funkcji składowych

4.11.2.1 template<typename Object > virtual bool IRunnable< Object >::Prepare (Object *parametr*) [pure virtual]

Metoda przygotowująca obiekt do operacji.

Parametry

>p0.10	>p0.15	p0.678
in *rozmiar* - liczba elementów do przygotowania;

Zwracane wartości

>p0.25	p0.705
true - jeśli przygotowanie się powiodło
false - jeśli wystąpił jakiś błąd

Implementowany w Graph_Test< Object >.

4.11.2.2 template<typename Object > virtual bool IRunnable< Object >::Run ( ) [pure virtual]

Metoda uruchamiająca zdefiniowaną operację

Parametry

>p0.10	>p0.15	p0.678
in *track* - parametr wykonania operacji

Zwracane wartości

>p0.25	p0.705
true - jeśli operacja zakończyła się pomyślnie
false - jeśli wystąpił jakiś błąd

Implementowany w Graph_Test< Object >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

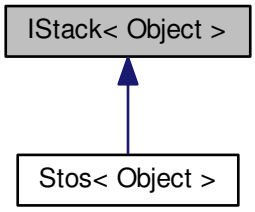
- prj/inc/IRunnable.hh

4.12 Dokumentacja szablonu klasy IStack< Object >

Klasa szablonowa modelująca interfejs stosu.

`#include <IStack.hh>`

Diagram dziedziczenia dla IStack< Object >



Metody publiczne

- virtual bool IsEmpty ()=0

  Metoda sprawdzająca, czy stos jest pusty.
- virtual int Size ()=0

  Metoda obliczająca rozmiar stosu.
- virtual Object Top ()=0

  Metoda zwracająca wierzchołek stosu.
- virtual void Push (Object item)=0

  Metoda dodająca element na stos.
- virtual Object Pop ()=0

  Metoda zrzucająca element ze stosu.

4.12.1 Opis szczegółowy

template<typename Object>class IStack< Object >

Klasa szablonowa modelująca interfejs stosu.

Definiuje ADT dla stosu.

Stos może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 22 pliku IStack.hh.

4.12.2 Dokumentacja funkcji składowych

4.12.2.1 template<typename Object > virtual bool IStack< Object >::IsEmpty ( ) [pure virtual]

Metoda sprawdzająca, czy stos jest pusty.

true - jeśli stos jest pustay false - jeśli stos nie jest pusty

Implementowany w Stos< Object >.

4.12.2.2 template<typename Object > virtual Object IStack< Object >::Pop ( ) [pure virtual]

Metoda zrzucająca element ze stosu.

Usuwa wierzchołek ze stosu i zwraca jego wartość.

Zwraca

element na wierzchu stosu

Implementowany w Stos< Object >.

4.12.2.3 template<typename Object > virtual void IStack< Object >::Push (Object *item*) [pure virtual]

Metoda dodająca element na stos.

Wrzuca element na wierzchołek stosu.

Parametry

>p0.10	>p0.15	p0.678
`in` *item* - element do dodania

Implementowany w Stos< Object >.

4.12.2.4 template<typename Object > virtual int IStack< Object >::Size ( ) [pure virtual]

Metoda obliczająca rozmiar stosu.

Zwraca

liczba elementów na stos

Implementowany w Stos< Object >.

4.12.2.5 template<typename Object > virtual Object IStack< Object >::Top ( ) [pure virtual]

Metoda zwracająca wierzchołek stosu.

Zwraca

element na wierzchu stosu

Implementowany w Stos< Object >.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/IStack.hh

4.13 Dokumentacja szablonu klasy Kolejka< Object >

Klasa szablonowa implementująca kolejkę

`#include <Kolejka.hh>`

Diagram dziedziczenia dla Kolejka< Object >

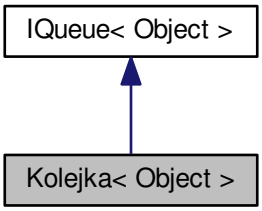
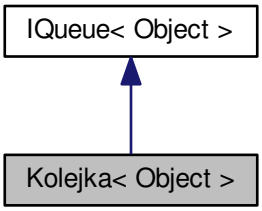


Diagram współpracy dla Kolejka< Object >:



Metody publiczne

- Kolejka ()

  Konstruktor bezargumentowy kolejki.
- ~Kolejka ()

  Destruktor kolejki.
- virtual bool IsEmpty ()

  Metoda sprawdzająca, czy kolejka jest pusta.
- virtual Object Front ()

  Metoda zwracająca pierwszy element kolejki.
- virtual void Enqueue (Object item)

  Metoda dodająca element do kolejki.
- virtual Object Dequeue ()

  Metoda usuwająca element kolejki.

4.13.1 Opis szczegółowy

template<typename Object>class Kolejka< Object >

Klasa szablonowa implementująca kolejkę

Kolejka zbudowana jest w oparciu o dynamiczną tablicę.

Kolejka może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 24 pliku Kolejka.hh.

4.13.2 Dokumentacja konstruktora i destruktora

4.13.2.1 template<typename Object > Kolejka< Object >::Kolejka ( )

Konstruktor bezargumentowy kolejki.

Inicjuje Kolejkę poprzez zaalokowanie tablicy o rozmiarze 1. Ustawia indeksy f (front) i r (rear) na 0.

Definicja w linii 90 pliku Kolejka.hh.

4.13.2.2 template<typename Object > Kolejka< Object >::~Kolejka ( )

Destruktor kolejki.

Zwalnia pamięć zajmowaną przez kolejkę. Ustawia wskaźnik tablicy na NULL.

Definicja w linii 96 pliku Kolejka.hh.

4.13.3 Dokumentacja funkcji składowych

4.13.3.1 template<typename Object > Object Kolejka< Object >::Dequeue ( ) [virtual]

Metoda usuwająca element kolejki.

Sprawdza, czy kolejka jest pusta. Jeśli tak, wyrzuca wyjątek. Jeśli nie, usuwa element z początki kolejki o indeksie f (front), a następnie przesuwa indeks f o jedno miejsce dalej.

Zwraca

pierwszy element kolejki

Implementuje IQueue< Object >.

Definicja w linii 131 pliku Kolejka.hh.

4.13.3.2 template<typename Object > void Kolejka< Object >::Enqueue (Object *item*) [virtual]

Metoda dodająca element do kolejki.

Ustawia element na koniec kolejki

- komórka tablicy o indeksie r (rear). Jeśli nie ma już miejsca w kolejce tablica jest powiększana 2 razy. Następnie element zostaje prawidłowo dodany do kolejki, a indeks r zostaje przesunięty o jedno miejsce dalej.

Parametry

>p0.10	>p0.15	p0.678
`in` *item* - element do dodania

Implementuje IQueue< Object >.

Definicja w linii 118 pliku Kolejka.hh.

4.13.3.3 template<typename Object > Object Kolejka< Object >::Front ( ) [virtual]

Metoda zwracająca pierwszy element kolejki.

Jeśli kolejka jest pusta, wyrzuca wyjątek. Zwraca element tablicy o indeksie f (front).

Zwraca

pierwszy element kolejki

Implementuje IQueue< Object >.

Definicja w linii 109 pliku Kolejka.hh.

4.13.3.4 template<typename Object > bool Kolejka< Object >::IsEmpty ( ) [virtual]

Metoda sprawdzająca, czy kolejka jest pusta.

Kolejka jest pusta, jeśli wartości indeksów f (front) i r (rear) są sobie równe. true - jeśli lista jest pusta false - jeśli nie jest pusta

Implementuje IQueue< Object >.

Definicja w linii 103 pliku Kolejka.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/Kolejka.hh

4.14 Dokumentacja szablonu klasy Node< Object >

Klasa implementująca węzeł listy jednokierunkowej.

`#include <Node.hh>`

Metody publiczne

- string GetKey ()

  Metoda zwracająca obecny klucz węzła.
- Object & GetValue ()

  Metoda zwracająca obecną wartość węzła.
- Node< Object > * GetNext ()

  Metoda zwracająca wskaźnik do następnego węzła.
- void SetKey (string newKey)

  Metoda przypisująca klucz do węzła.
- void SetValue (Object newValue)

  Metoda przypisująca wartość do węzła.
- void SetNext (Node< Object > *newItem)

  Metoda przypisująca następny element do węzła.

4.14.1 Opis szczegółowy

template<typename Object>class Node< Object >

Klasa implementująca węzeł listy jednokierunkowej.

Węzeł pozwalana przechowywać dane w formie: (klucz, wartość). Wartość może być dowolnego typu dzięki zastosowaniu szablonu.

Definicja w linii 21 pliku Node.hh.

|>p0.10|>p0.15|p0.678|

in *newKey* - klucz do ustawienia w węźle

Definicja w linii 52 pliku Node.hh.

4.14.2.5 template<typename Object> void Node< Object >::SetNext (Node< Object > * *newItem*) [inline]

Metoda przypisująca następny element do węzła.

Ustawia podany wskaźnik w polu next węzła.

Parametry

|>p0.10|>p0.15|p0.678|

in *newItem* - wskaźnik do następnego węzła

Definicja w linii 66 pliku Node.hh.

4.14.2.6 template<typename Object> void Node< Object >::SetValue (Object *newValue*) [inline]

Metoda przypisująca wartość do węzła.

Ustawia podany element w polu value węzła.

Parametry

>p0.10	>p0.15	p0.678
`in` *newValue* - wartość do ustawienia w węźle

Definicja w linii 59 pliku Node.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/Node.hh

4.15 Dokumentacja klasy Path

`#include <Path.hh>`

Metody publiczne

- vector< unsigned > & V ()
- int & c ()
- bool Find (unsigned k)

  Szuka wierzchołka j w ścieżce.

4.15.1 Opis szczegółowy

Definicja w linii 9 pliku Path.hh.

4.15.2 Dokumentacja funkcji składowych

4.15.2.1 int& Path::c ( ) [inline]

Definicja w linii 15 pliku Path.hh.

4.15.2.2 bool Path::Find (unsigned *k*)

Szuka wierzchołka j w ścieżce.

Parametry

>p0.10	>p0.15	p0.678
`in` *k* - wierzchołek do wyszukania

Zwracane wartości

>p0.25	p0.705
true - jeśli znalezniono
false - jeśli nie znalezniono

Definicja w linii 26 pliku Path.hh.

4.15.2.3 vector<unsigned>& Path::V ( ) [inline]

-> długość ścieżki

Definicja w linii 14 pliku Path.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/Path.hh

4.16 Dokumentacja szablonu klasy SList< Object >

Szablonowa klasa implementująca listę jednokierunkową

`#include <SList.hh>`

Diagram dziedziczenia dla SList< Object >

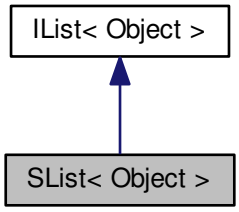
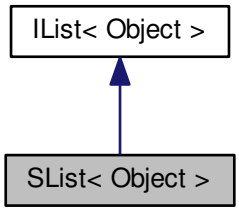


Diagram współpracy dla SList< Object >:



Metody publiczne

- SList ()

  Konstruktor listy jednokierunkowaj.
- ~SList ()

  Destruktor listy jednokierunkowaj.
- virtual bool IsEmpty ()

  Metoda sprawdzająca, czy lista jest pusta.
- virtual const Object & Front ()

  Metoda zwracająca pierwszy element listy.
- virtual void AddFront (const Object newItem)

  Metoda dodająca element na początek listy.
- void AddFront (const string newKey, const Object newItem)

  Metoda dodająca nowy węzeł na początek listy.

- virtual void RemoveFront ()

  Metoda usuwająca element z początku listy.
- Node< Object > * Head ()

  Metoda zwracająca głowę listy.
- Object Remove (string key)

  Metoda usuwająca węzeł o podanym kluczu.
- void Print ()

  Metoda wypisująca zawartość listy na standardowe wyjście.
- Node< Object > * Find (Object k)

  Metoda wyszukująca element na liście.
- Node< Object > * Find (string key)

  Metoda wyszukująca klucz na liście.
- Node< Object > * FindOrAdd (string key)

  Wyszukuje lub tworzy klucz na liście.

4.16.1 Opis szczegółowy

template<typename Object>class SList< Object >

Szablonowa klasa implementująca listę jednokierunkową

SLista jest zbudowana w oparciu o węzły SNode oraz operacje na wskaźnikach.

SLista może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 25 pliku SList.hh.

4.16.2 Dokumentacja konstruktora i destruktora

4.16.2.1 template<typename Object > SList< Object >::SList ( )

Konstruktor listy jednokierunkowaj.

Inicjuje SListę poprzez ustawienie wskaźnika NULL jako początek (head) tej listy.

Definicja w linii 156 pliku SList.hh.

4.16.2.2 template<typename Object > SList< Object >::~SList ( )

Destruktor listy jednokierunkowaj.

Usuwa listę poprzez zwalnianie pamięci kolejnych początkowych węzłów.

Definicja w linii 160 pliku SList.hh.

4.16.3 Dokumentacja funkcji składowych

4.16.3.1 template<typename Object > void SList< Object >::AddFront (const Object *newItem*) [virtual]

Metoda dodająca element na początek listy.

Alokuje nowy węzeł, dodaje nowy element, dodaje powiązanie tak, aby węzeł wskazywał na stary head, uaktualnia head.

Parametry

>p0.10	>p0.15	p0.678
in	*newItem*	- element do dodania

Implementuje IList< Object >.

Definicja w linii 194 pliku SList.hh.

4.16.3.2 template<typename Object > void SList< Object >::AddFront (const string *newKey,* const Object *newItem*)

Metoda dodająca nowy węzeł na początek listy.

Alokuje nowy węzeł, przypisuje podane klucz i wartość, dodaje powiązanie tak, aby węzeł wskazywał na stary head, uaktualnia head.

Parametry

>p0.10	>p0.15	p0.678
in *newKey* - klucz do dodania	
in *newItem* - wartość do dodania	

Definicja w linii 203 pliku SList.hh.

4.16.3.3 template<typename Object > Node< Object > * SList< Object >::Find (Object *k*)

Metoda wyszukująca element na liście.

Implementuje algorytm liniowego przeszukiwania listy.

Parametry

>p0.10	>p0.15	p0.678
in *k* - element do wyszukania	

Zwraca

Wskaźnik do znalezionego elementu lub NULL, gdy nie znaleziono.

Definicja w linii 254 pliku SList.hh.

4.16.3.4 template<typename Object > Node< Object > * SList< Object >::Find (string *key*)

Metoda wyszukująca klucz na liście.

Liniowo przeszukuje listę porównując kolejne klucze z podanym.

Parametry

>p0.10	>p0.15	p0.678
in *key* - klucz do wyszukania	

Zwraca

Wskaźnik do znalezionego elementu lub NULL, gdy nie znaleziono.

Definicja w linii 265 pliku SList.hh.

4.16.3.5 template<typename Object > Node< Object > * SList< Object >::FindOrAdd (string *key*)

Wyszukuje lub tworzy klucz na liście.

Liniowo przeszukuje listę porównując kolejne klucze z podanym. Jeśli klucza nie znaleziono, na początek listy dodawany jest nowy węzeł o podanym kluczu. Jego wartość (domyślnie równą 0) można ustawić poprzez operację przypisania (za pomocą operatora []).

Parametry

|>p0.10|>p0.15|p0.678|
in *key* - klucz do wyszukania lub utworzenia

Zwraca

Wskaźnik do znalezionego lub nowoutworzonego elementu.

Definicja w linii 275 pliku SList.hh.

4.16.3.6 template<typename Object > const Object & SList< Object >::Front ( ) [virtual]

Metoda zwracająca pierwszy element listy.

Sprawdza, czy lista jest pusta i zwraca dane pierwszego węzła listy. Jeśli lista jest pusta, wyrzuca wyjątek.

Zwraca

element pierwszego węzła listy

Implementuje IList< Object >.

Definicja w linii 185 pliku SList.hh.

4.16.3.7 template<typename Object > Node< Object > * SList< Object >::Head ( )

Metoda zwracająca głowę listy.

Zwraca wskaźnik do początku listy lub NULL, jeśli lista jest pusta.

Zwraca

Wskaźnik do głowy listy.

Definicja w linii 176 pliku SList.hh.

4.16.3.8 template<typename Object > bool SList< Object >::IsEmpty ( ) [virtual]

Metoda sprawdzająca, czy lista jest pusta.

Sprawdza, czy head wskazuje na coś innego niż NULL. Implementacja metody wirtualnej z interfejsu IList.

Zwraca

true - jeśli lista jest pusta, false - jeśli nie

Implementuje IList< Object >.

Definicja w linii 167 pliku SList.hh.

4.16.3.9 template<typename Object > void SList< Object >::Print ( )

Metoda wypisująca zawartość listy na standardowe wyjście.

Definicja w linii 247 pliku SList.hh.

4.16.3.10 template<typename Object > Object SList< Object >::Remove (string *key*)

Metoda usuwająca węzeł o podanym kluczu.

Znajduje węzeł o podanym kluczu. Uaktualnia wskaźnik poprzednika na następny element listy. Usuwa go, zapamiętują wartość w nim przechowywaną.

Parametry

|>p0.10|>p0.15|p0.678|
in *key* - klucz elementu do usunięcia

Definicja w linii 225 pliku SList.hh.

4.16.3.11 template<typename Object > void SList< Object >::RemoveFront ( ) [virtual]

Metoda usuwająca element z początku listy.

Uaktualnia head, aby wskazywał na kolejny element na liście, po czym usuwa stary węzeł.

Parametry

|>p0.10|>p0.15|p0.678|
in *newItem* - element do dodania

Implementuje IList< Object >.

Definicja w linii 213 pliku SList.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/SList.hh

4.17 Dokumentacja szablonu klasy SNode< Object >

`#include <SNode.hh>`

Metody publiczne

- SNode ()
- Object GetElement ()
- SNode< Object > * GetNext ()
- void SetElement (Object newItem)
- void SetNext (SNode< Object > *newItem)

4.17.1 Opis szczegółowy

template<typename Object>class SNode< Object >

Definicja w linii 8 pliku SNode.hh.

4.17.2 Dokumentacja konstruktora i destruktora

4.17.2.1 template<typename Object> SNode< Object >::SNode ( ) [inline]

Definicja w linii 14 pliku SNode.hh.

4.17.3 Dokumentacja funkcji składowych

4.17.3.1 template<typename Object> Object SNode< Object >::GetElement ( ) [inline]

Definicja w linii 15 pliku SNode.hh.

4.17.3.2 template<typename Object> SNode<Object>* SNode< Object >::GetNext ( ) [inline]

Definicja w linii 16 pliku SNode.hh.

4.17.3.3 template<typename Object> void SNode< Object >::SetElement (Object *newItem*) [inline]

Definicja w linii 17 pliku SNode.hh.

4.17.3.4 template<typename Object> void SNode< Object >::SetNext (SNode< Object > * *newItem*) [inline]

Definicja w linii 18 pliku SNode.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

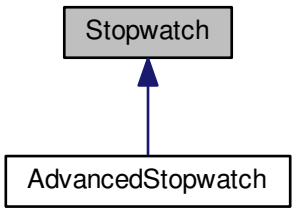
- prj/inc/SNode.hh

4.18 Dokumentacja klasy Stopwatch

Klasa implementująca podstawowy stoper.

`#include <Stopwatch.hh>`

Diagram dziedziczenia dla Stopwatch



Metody publiczne

- virtual void Start ()

  Rozpoczyna pomiar czasu.
- virtual void Stop ()

  Kończy pomiar czasu.
- virtual double GetElapsedTime ()

  Oblicza czas na podstawie pól klasy.

Atrybuty chronione

- timeval start
- timeval stop

4.18.1 Opis szczegółowy

Klasa implementująca podstawowy stoper.

Klasa jest modelelem stopera z funkcjami start, stop i oblicz czas.

Definicja w linii 20 pliku Stopwatch.hh.

4.18.2 Dokumentacja funkcji składowych

4.18.2.1 double Stopwatch::GetElapsedTime ( ) [virtual]

Oblicza czas na podstawie pól klasy.

Odejmuje wartości zapisane w polach stop i start. Daje wynik w mikrosekundach.

Definicja w linii 12 pliku Stopwatch.cpp.

4.18.2.2 void Stopwatch::Start ( ) [virtual]

Rozpoczyna pomiar czasu.

Przypisuje wynik metody gettimeofday() do pola start.

Definicja w linii 4 pliku Stopwatch.cpp.

4.18.2.3 void Stopwatch::Stop ( ) [virtual]

Kończy pomiar czasu.

Przypisuje wynik metody gettimeofday() do pola stop.

Definicja w linii 8 pliku Stopwatch.cpp.

4.18.3 Dokumentacja atrybutów składowych

4.18.3.1 timeval Stopwatch::start [protected]

Definicja w linii 23 pliku Stopwatch.hh.

4.18.3.2 timeval Stopwatch::stop [protected]

struktury timeval start i stop

Definicja w linii 23 pliku Stopwatch.hh.

Dokumentacja dla tej klasy została wygenerowana z plików:

- prj/inc/Stopwatch.hh
- prj/src/Stopwatch.cpp

4.19 Dokumentacja szablonu klasy Stos< Object >

Klasa szablonowa implementująca stos.

`#include <Stos.hh>`

Diagram dziedziczenia dla Stos< Object >

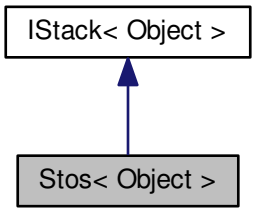
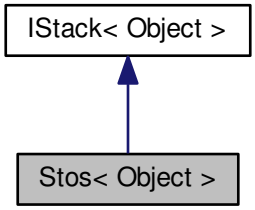


Diagram współpracy dla Stos< Object >:



Metody publiczne

- Stos ()
  Konstruktor bezargumentowy stosu.
- Stos (int ile)
  Konstruktor stosu.
- ~Stos ()
  Destruktor stosu.
- int & Pojemnosc ()
  Metoda sprawdzająca pojemność stosu.
- virtual bool IsEmpty ()
  Metoda sprawdzająca, czy stos jest pusty.
- virtual int Size ()
  Metoda obliczająca rozmiar stosu.
- virtual Object Top ()
  Metoda zwracająca wierzchołek stosu.
- virtual void Push (Object item)
  Metoda dodająca element na stos.
- virtual Object Pop ()
  Metoda zrzucająca element ze stosu.
- void Print ()
- bool Find (Object k)

4.19.1 Opis szczegółowy

template<typename Object>class Stos< Object >

Klasa szablonowa implementująca stos.

Stos zbudowany jest w oparciu o dynamiczną tablicę.

Stos może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 22 pliku Stos.hh.

4.19.2 Dokumentacja konstruktora i destruktora

4.19.2.1 template<typename Object > Stos< Object >::Stos ( )

Konstruktor bezargumentowy stosu.

Inicjuje stos poprzez zaalokowanie tablicy o rozmiarze 1. Ustawia indeks top na -1.

Definicja w linii 140 pliku Stos.hh.

4.19.2.2 template<typename Object > Stos< Object >::Stos (int *ile*)

Konstruktor stosu.

Inicjuje stos poprzez zaalokowanie tablicy o rozmiarze podanym w argumencie konstruktora (o ile jest różny od 0). Ustawia indeks top na -1.

Parametry

>p0.10	>p0.15	p0.678
`in` *ile*	- początkowa pojemność kolejki;	

Definicja w linii 149 pliku Stos.hh.

4.19.2.3 template<typename Object > Stos< Object >::~Stos ( )

Destruktor stosu.

Zwalnia pamięć zajmowaną przez stosu. Ustawia wskaźnik tablicy dynamicznej na NULL.

Definicja w linii 161 pliku Stos.hh.

4.19.3 Dokumentacja funkcji składowych

4.19.3.1 template<typename Object > bool Stos< Object >::Find (Object *k*)

Definicja w linii 120 pliku Stos.hh.

4.19.3.2 template<typename Object > bool Stos< Object >::IsEmpty ( ) `[virtual]`

Metoda sprawdzająca, czy stos jest pusty.

Jeśli indeks top<0, stos jest pusty. W przeciwnym wypadku, stos nie jest pusty. true - jeśli stos jest pusty false - jeśli stos nie jest pusty

Implementuje IStack< Object >.

Definicja w linii 168 pliku Stos.hh.

4.19.3.3 template<typename Object> int& Stos< Object >::Pojemnosc ( ) `[inline]`

Metoda sprawdzająca pojemność stosu.

Zwraca parametr opisujący pojemność tablicy dynamicznej implementującej stos.

Zwraca

pojemność stosu

Definicja w linii 63 pliku Stos.hh.

4.19.3.4 template<typename Object > Object Stos< Object >::Pop ( ) [virtual]

Metoda zrzucająca element ze stosu.

Jeśli stos jest pusty, wyrzuca wyjątek. Zwraca element tablicy o indeksie top. Zmniejsza indeks top o 1.

Zwraca

element na wierzchu stosu

Implementuje IStack< Object >.

Definicja w linii 214 pliku Stos.hh.

4.19.3.5 template<typename Object > void Stos< Object >::Print ( )

Definicja w linii 131 pliku Stos.hh.

4.19.3.6 template<typename Object > void Stos< Object >::Push (Object *item*) [virtual]

Metoda dodająca element na stos.

Wrzuca element na wierzchołek stosu. Indeks top zostaje przesunięty o jedno miejsce dalej. Jeśli nie ma już miejsca na stosie tablica jest powiększana 2 razy. Następnie element zostaje prawidłowo wrzucony na stos, a indeks top zostaje przesunięty o jedno miejsce dalej.

Parametry

>p0.10	>p0.15	p0.678
`in` *item* - element do dodania

Implementuje IStack< Object >.

Definicja w linii 196 pliku Stos.hh.

4.19.3.7 template<typename Object > int Stos< Object >::Size ( ) [virtual]

Metoda obliczająca rozmiar stosu.

Rozmiar obliczany jest przez działanie 'top+1'.

Zwraca

liczba elementów na stosie

Implementuje IStack< Object >.

Definicja w linii 177 pliku Stos.hh.

4.19.3.8 template<typename Object > Object Stos< Object >::Top ( ) [virtual]

Metoda zwracająca wierzchołek stosu.

Jeśli stos jest pusty, wyrzuca wyjątek. Zwraca element tablicy o indeksie top.

Zwraca

element na wierzchu stosu

Implementuje IStack< Object >.

Definicja w linii 185 pliku Stos.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/Stos.hh

4.20 Dokumentacja klasy Voisin

`#include <Voisin.hh>`

Metody publiczne

- Voisin ()
- Voisin (unsigned neighbour, unsigned w=1)
- unsigned & v ()
- unsigned & w ()

4.20.1 Opis szczegółowy

Definicja w linii 7 pliku Voisin.hh.

4.20.2 Dokumentacja konstruktora i destruktora

4.20.2.1 Voisin::Voisin ( ) [inline]

Definicja w linii 11 pliku Voisin.hh.

4.20.2.2 Voisin::Voisin (unsigned *neighbour,* unsigned *w = 1*) [inline]

Definicja w linii 12 pliku Voisin.hh.

4.20.3 Dokumentacja funkcji składowych

4.20.3.1 unsigned& Voisin::v ( ) [inline]

Definicja w linii 13 pliku Voisin.hh.

4.20.3.2 unsigned& Voisin::w ( ) [inline]

Definicja w linii 14 pliku Voisin.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- prj/inc/Voisin.hh

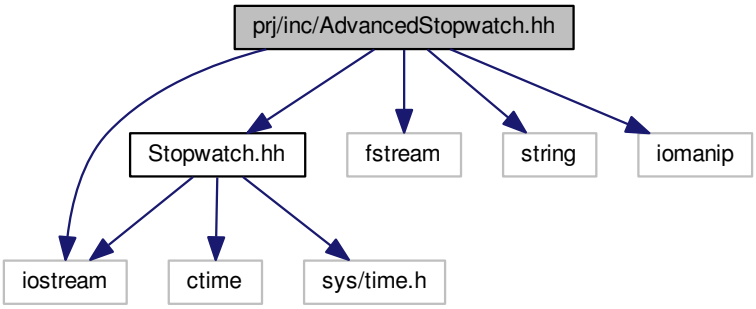
5 Dokumentacja plików

5.1 Dokumentacja pliku prj/inc/AdvancedStopwatch.hh

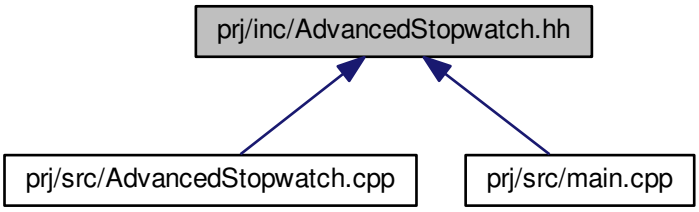
`#include "Stopwatch.hh"`

```
#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>
```

Wykres zależności załączania dla AdvancedStopwatch.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class AdvancedStopwatch

  Klasa implementująca rozbudowany stoper.

Definicje

- #define MAX_LAPS 100
- #define BUFOR 10
- #define DOKLADNOSC 8

5.1.1 Opis szczegółowy

Plik zawiera implementację rozbudowanego stopera.

Definicja w pliku AdvancedStopwatch.hh.

5.1.2 Dokumentacja definicji

5.1.2.1 #define BUFOR 10

Definicja w linii 12 pliku AdvancedStopwatch.hh.

5.1.2.2 #define DOKLADNOSC 8

Definicja w linii 13 pliku AdvancedStopwatch.hh.

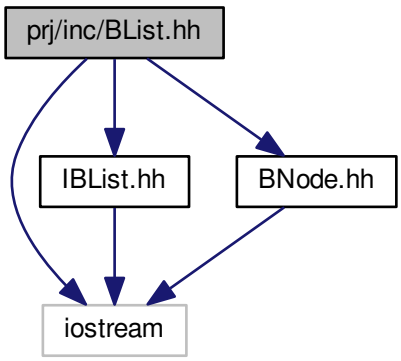
5.1.2.3 #define MAX_LAPS 100

Definicja w linii 11 pliku AdvancedStopwatch.hh.

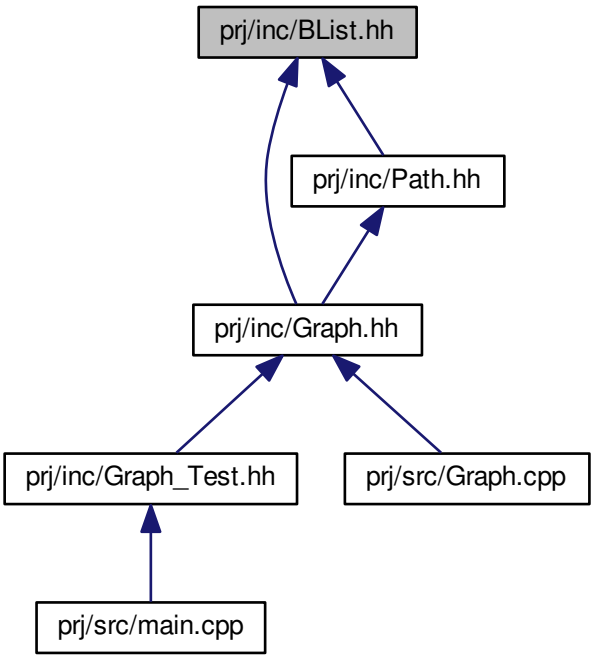
5.2 Dokumentacja pliku prj/inc/BList.hh

```
#include <iostream>
#include "IBList.hh"
#include "BNode.hh"
```

Wykres zależności załączania dla BList.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class BList< Object >

  Szablonowa klasa implementująca listę dwukierunkową

5.2.1 Opis szczegółowy

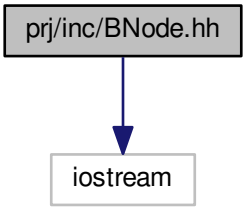
Plik zawiera definicję klasy implementującej listę dwukierunkową.

Definicja w pliku BList.hh.

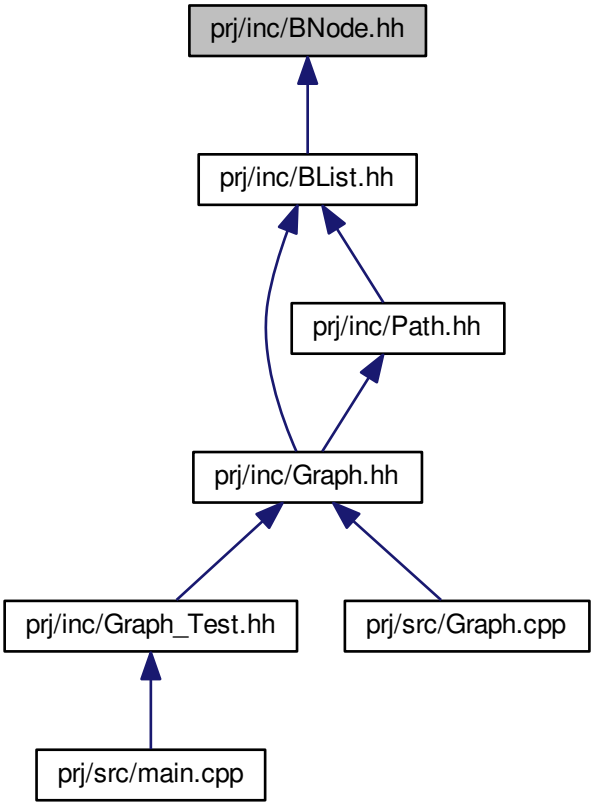
5.3 Dokumentacja pliku prj/inc/BNode.hh

```
#include <iostream>
```

Wykres zależności załączania dla BNode.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



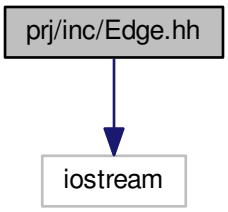
Komponenty

- class BNode< Object >

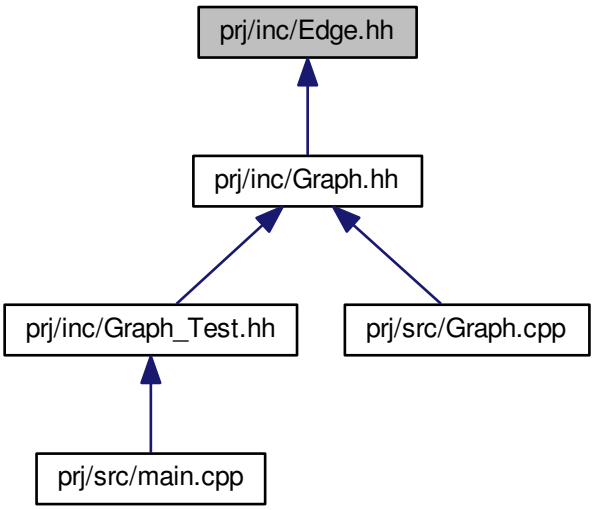
5.4 Dokumentacja pliku prj/inc/Edge.hh

```
#include <iostream>
```

Wykres zależności załączania dla Edge.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Edge

Funkcje

- bool operator== (Edge e1, Edge e2)
- bool operator!= (Edge e1, Edge e2)

5.4.1 Dokumentacja funkcji

5.4.1.1 bool operator!= (Edge *e1,* Edge *e2*)

Definicja w linii 30 pliku Edge.hh.

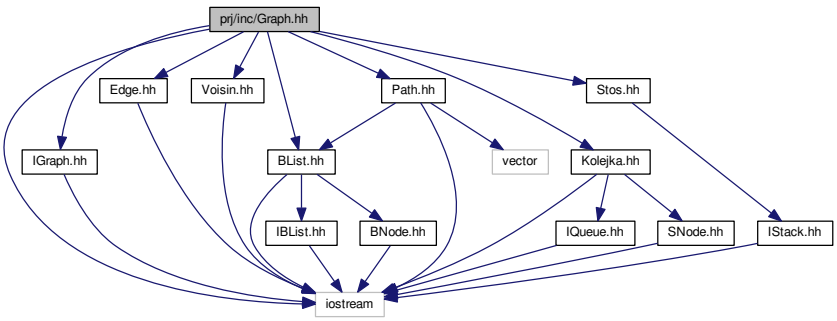
5.4.1.2 bool operator== (Edge *e1,* Edge *e2*)

Definicja w linii 21 pliku Edge.hh.

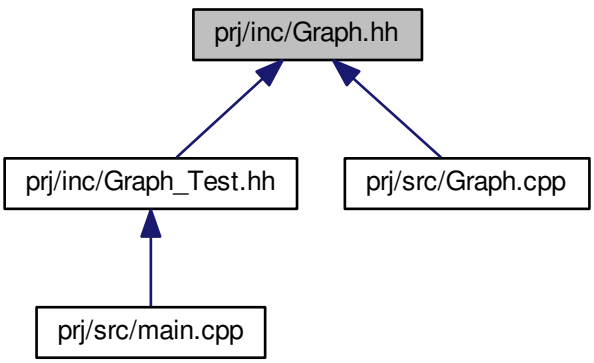
5.5 Dokumentacja pliku prj/inc/Graph.hh

```
#include <iostream>
#include "IGraph.hh"
#include "BList.hh"
#include "Edge.hh"
#include "Voisin.hh"
#include "Path.hh"
#include "Kolejka.hh"
#include "Stos.hh"
```

Wykres zależności załączania dla Graph.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Graph

5.5.1 Opis szczegółowy

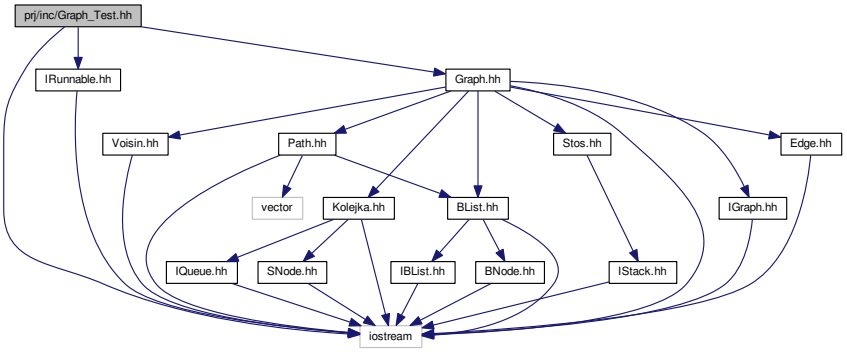
Plik zawiera implementację interfejsu grafu.

Definicja w pliku Graph.hh.

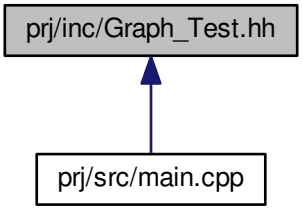
5.6 Dokumentacja pliku prj/inc/Graph_Test.hh

```
#include <iostream>
#include "IRunnable.hh"
#include "Graph.hh"
```

Wykres zależności załączania dla Graph_Test.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Graph_Test< Object >

  Szablonowa klasa implementująca testowy graf.

5.6.1 Opis szczegółowy

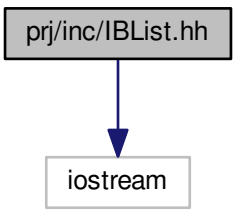
Plik zawiera implementację IRunnable dla grafu (operacje BFS i DFS).

Definicja w pliku Graph_Test.hh.

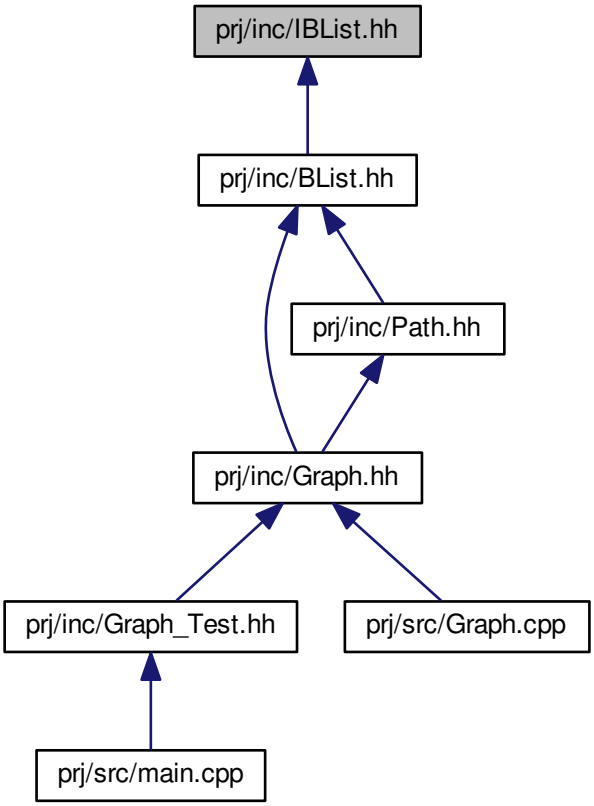
5.7 Dokumentacja pliku prj/inc/IBList.hh

```
#include <iostream>
```

Wykres zależności załączania dla IBList.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class IBList< Object >

  Interfejs listy dwukierunkowej.

5.7.1 Opis szczegółowy

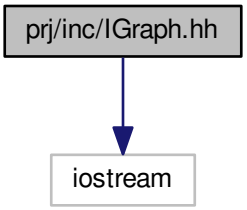
Plik zawiera interfejs listy dwukierunkową

Definicja w pliku IBList.hh.

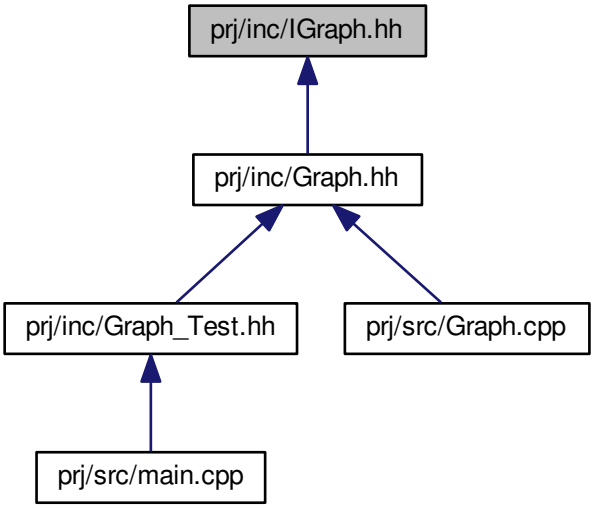
5.8 Dokumentacja pliku prj/inc/IGraph.hh

```
#include <iostream>
```

Wykres zależności załączania dla IGraph.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class IGraph

5.8.1 Opis szczegółowy

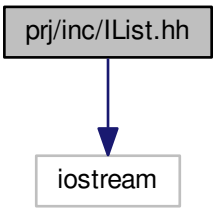
Plik zawiera interfejs grafu.

Definicja w pliku IGraph.hh.

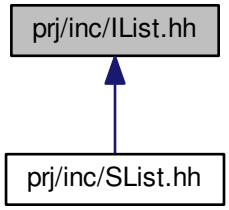
5.9 Dokumentacja pliku prj/inc/IList.hh

```
#include <iostream>
```

Wykres zależności załączania dla IList.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class IList< Object >

  Interfejs listy jednokierunkowej.

5.9.1 Opis szczegółowy

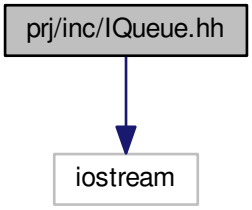
Plik zawiera interfejs listy jednokierunkową

Definicja w pliku IList.hh.

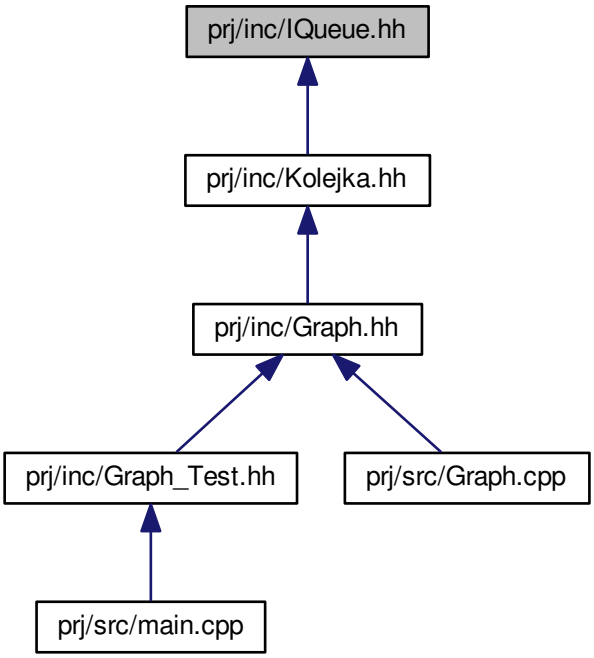
5.10 Dokumentacja pliku prj/inc/IQueue.hh

```
#include <iostream>
```

Wykres zależności załączania dla IQueue.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class IQueue< Object >

  Klasa modelująca interfejs kolejki.

5.10.1 Opis szczegółowy

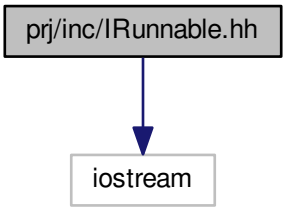
Plik zawiera interfejs kolejki

Definicja w pliku IQueue.hh.

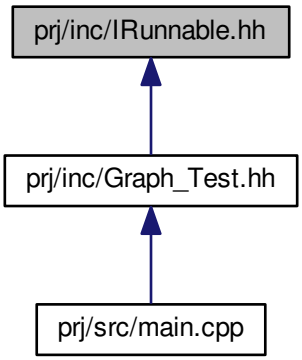
5.11 Dokumentacja pliku prj/inc/IRunnable.hh

```
#include <iostream>
```

Wykres zależności załączania dla IRunnable.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class IRunnable< Object >

  Klasa szablonowa modelująca interfejs "Biegacza".

5.11.1 Opis szczegółowy

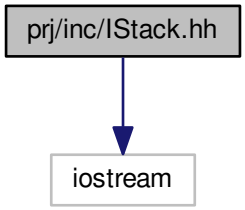
Plik zawiera interfejs obiektu, który można poddawać pomiarom czasu działania.

Definicja w pliku IRunnable.hh.

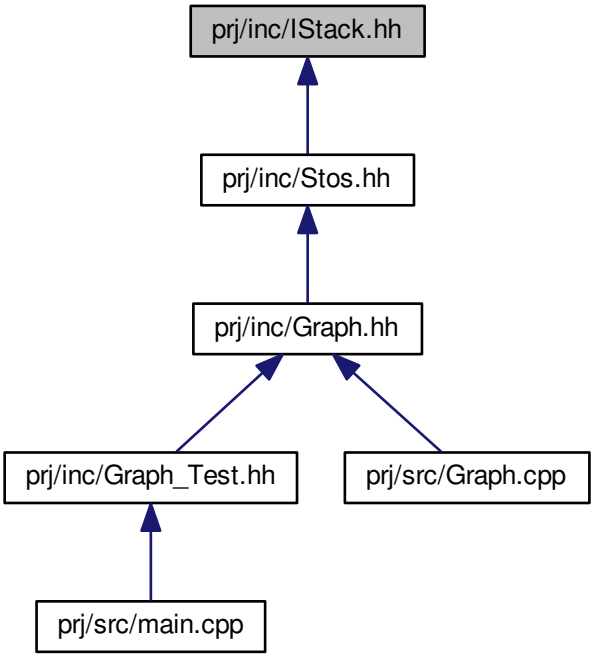
5.12 Dokumentacja pliku prj/inc/IStack.hh

```
#include <iostream>
```

Wykres zależności załączania dla IStack.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class IStack< Object >

  Klasa szablonowa modelująca interfejs stosu.

5.12.1 Opis szczegółowy

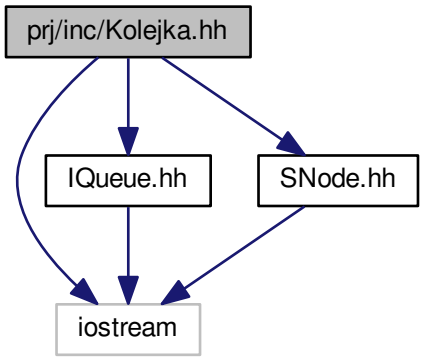
Plik zawiera interfejs stosu

Definicja w pliku IStack.hh.

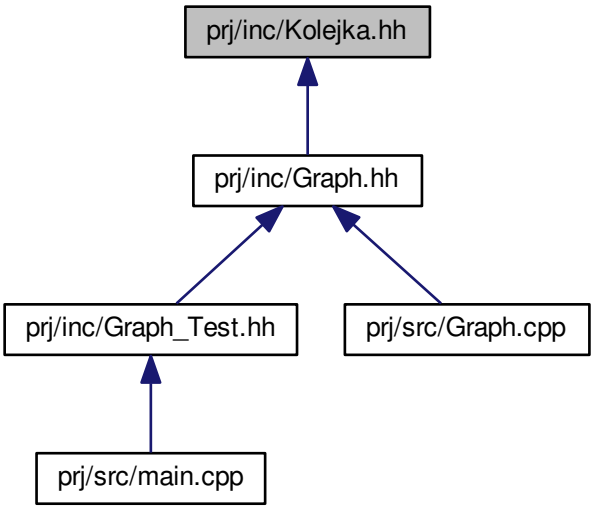
5.13 Dokumentacja pliku prj/inc/Kolejka.hh

```
#include <iostream>
#include "IQueue.hh"
#include "SNode.hh"
```

Wykres zależności załączania dla Kolejka.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Kolejka< Object >

  Klasa szablonowa implementująca kolejkę

5.13.1 Opis szczegółowy

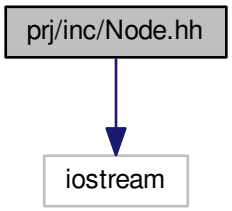
Plik zawiera implementację interfejsu kolejki

Definicja w pliku Kolejka.hh.

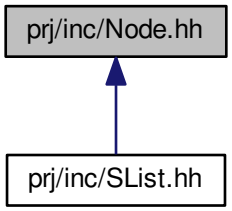
5.14 Dokumentacja pliku prj/inc/Node.hh

```
#include <iostream>
```

Wykres zależności załączania dla Node.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Node< Object >

  Klasa implementująca węzeł listy jednokierunkowej.

5.14.1 Opis szczegółowy

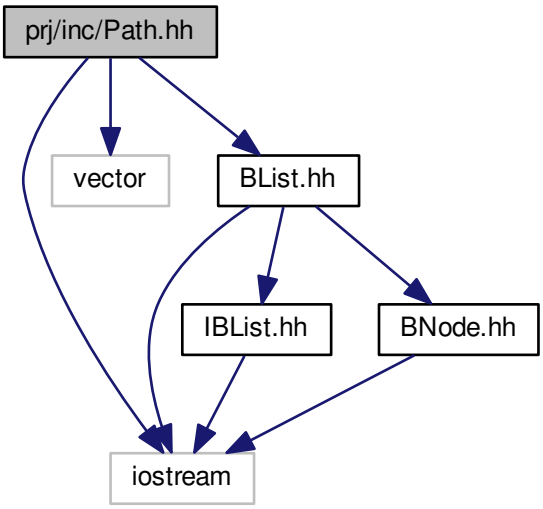
Plik zawiera definicję węzła listy jednokierunkowej.

Definicja w pliku Node.hh.

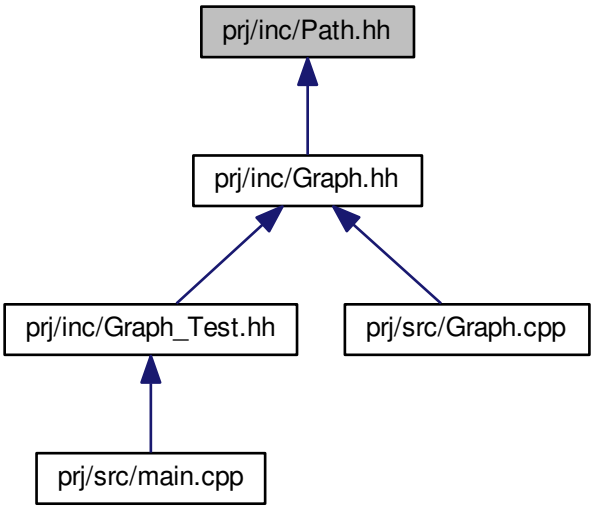
5.15 Dokumentacja pliku prj/inc/Path.hh

```
#include <iostream>
#include <vector>
#include "BList.hh"
```

Wykres zależności załączania dla Path.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Path

Funkcje

- ostream & operator<< (ostream &StrWy, Path s)
- ostream & operator<< (ostream &StrWy, Path *s)

5.15.1 Dokumentacja funkcji

5.15.1.1 ostream& operator<< (ostream & *StrWy,* Path *s*)

Definicja w linii 35 pliku Path.hh.

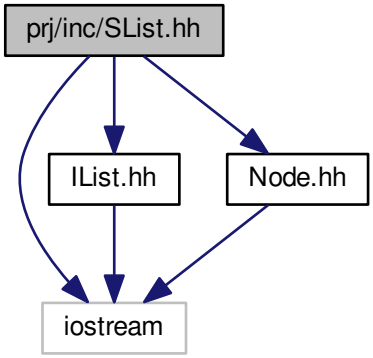
5.15.1.2 ostream& operator<< (ostream & *StrWy,* Path * *s*)

Definicja w linii 43 pliku Path.hh.

5.16 Dokumentacja pliku prj/inc/SList.hh

```
#include <iostream>
#include "IList.hh"
#include "Node.hh"
```

Wykres zależności załączania dla SList.hh:



Komponenty

- class SList< Object >

  Szablonowa klasa implementująca listę jednokierunkową

5.16.1 Opis szczegółowy

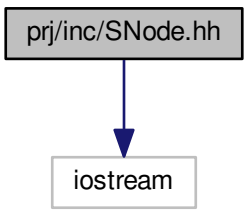
Plik zawiera definicję klasy implementującej listę jednokierunkową.

Definicja w pliku SList.hh.

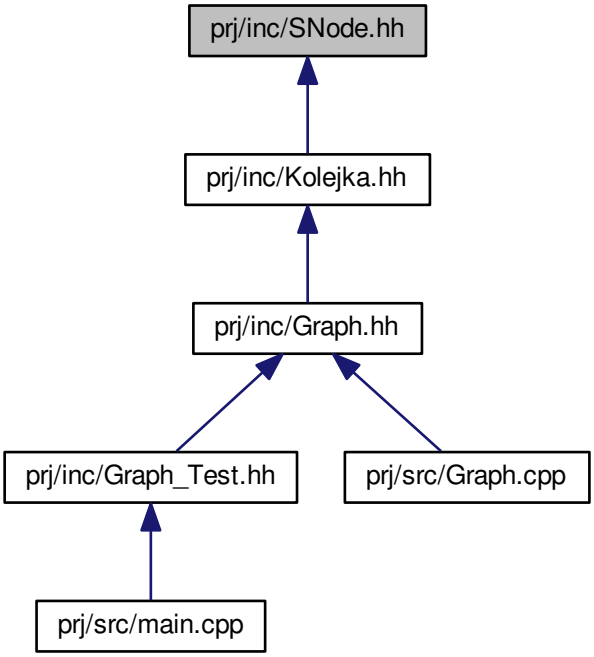
5.17 Dokumentacja pliku prj/inc/SNode.hh

```
#include <iostream>
```

Wykres zależności załączania dla SNode.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



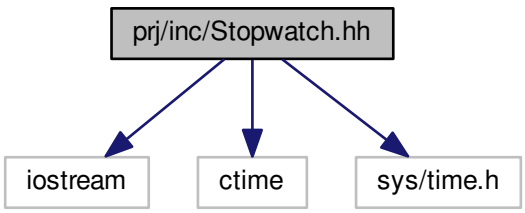
Komponenty

- class SNode< Object >

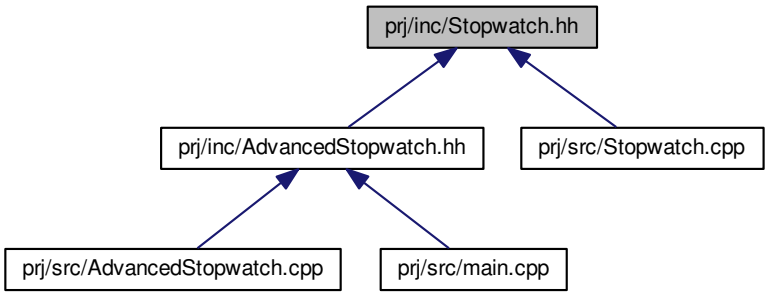
5.18 Dokumentacja pliku prj/inc/Stopwatch.hh

```
#include <iostream>
#include <ctime>
#include <sys/time.h>
```

Wykres zależności załączania dla Stopwatch.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Stopwatch

  Klasa implementująca podstawowy stoper.

5.18.1 Opis szczegółowy

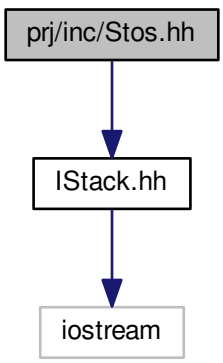
Plik zawiera implementację podstawowego stopera.

Definicja w pliku Stopwatch.hh.

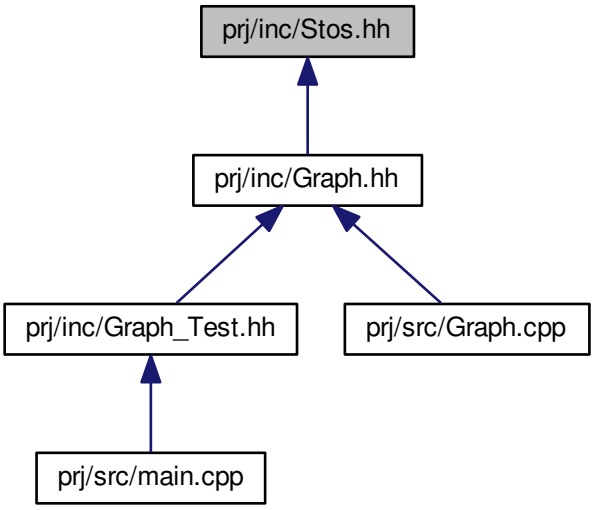
5.19 Dokumentacja pliku prj/inc/Stos.hh

```
#include "IStack.hh"
```

Wykres zależności załączania dla Stos.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Stos< Object >

  Klasa szablonowa implementująca stos.

5.19.1 Opis szczegółowy

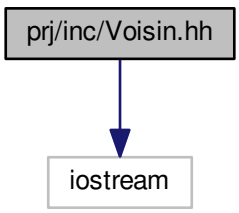
Plik zawiera implementację interfejsu stosu

Definicja w pliku Stos.hh.

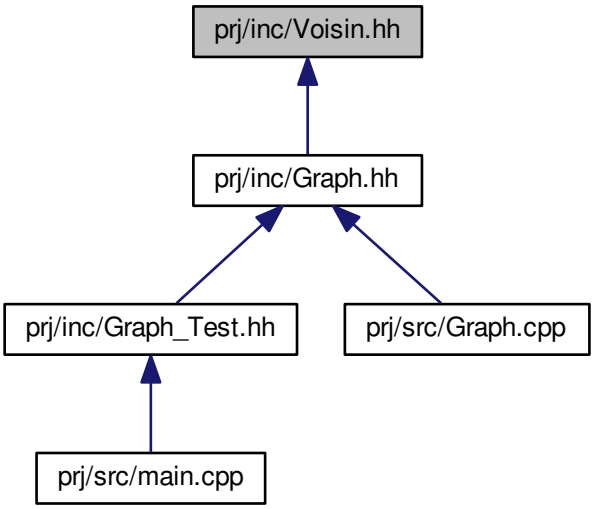
5.20 Dokumentacja pliku prj/inc/Voisin.hh

```
#include <iostream>
```

Wykres zależności załączania dla Voisin.hh:



Ten wykres pokazuje, które pliki bezpośrednio lub pośrednio załączają ten plik:



Komponenty

- class Voisin

Funkcje

- bool operator== (Voisin e1, Voisin e2)
- bool operator!= (Voisin e1, Voisin e2)
- ostream & operator<< (ostream &StrWy, Voisin &s)
- ostream & operator<< (ostream &StrWy, Voisin *s)

5.20.1 Dokumentacja funkcji

5.20.1.1 bool operator!= (Voisin *e1*, Voisin *e2*)

Definicja w linii 25 pliku Voisin.hh.

5.20.1.2 ostream& operator<< (ostream & *StrWy*, Voisin & *s*)

Definicja w linii 36 pliku Voisin.hh.

5.20.1.3 ostream& operator<< (ostream & *StrWy*, Voisin * *s*)

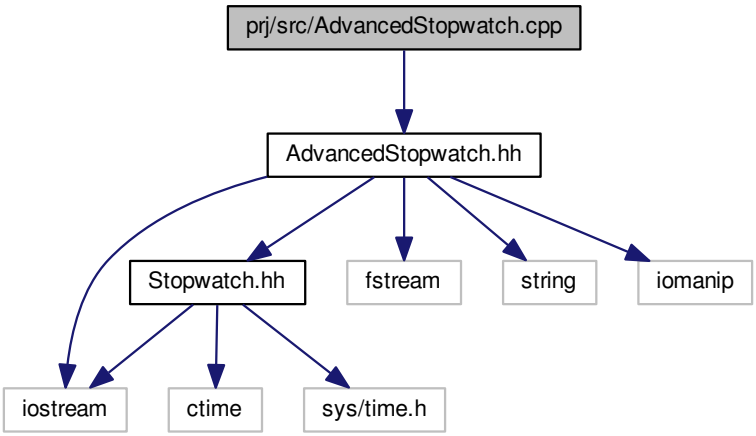
Definicja w linii 42 pliku Voisin.hh.

5.20.1.4 bool operator== (Voisin *e1*, Voisin *e2*)

Definicja w linii 17 pliku Voisin.hh.

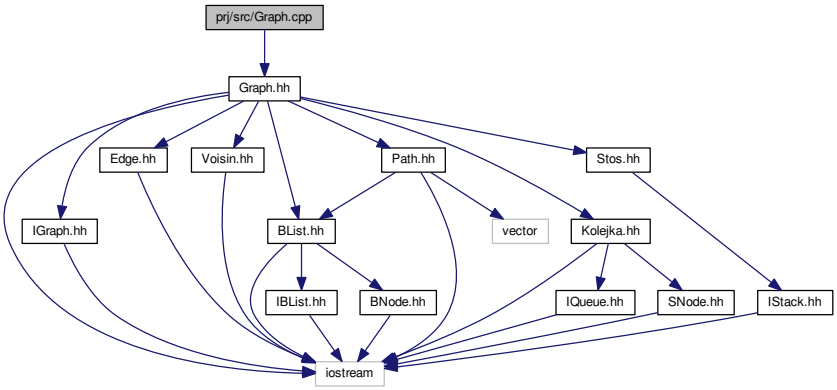
5.21 Dokumentacja pliku prj/src/AdvancedStopwatch.cpp

```
#include "AdvancedStopwatch.hh"
```
Wykres zależności załączania dla AdvancedStopwatch.cpp:



5.22 Dokumentacja pliku prj/src/Graph.cpp

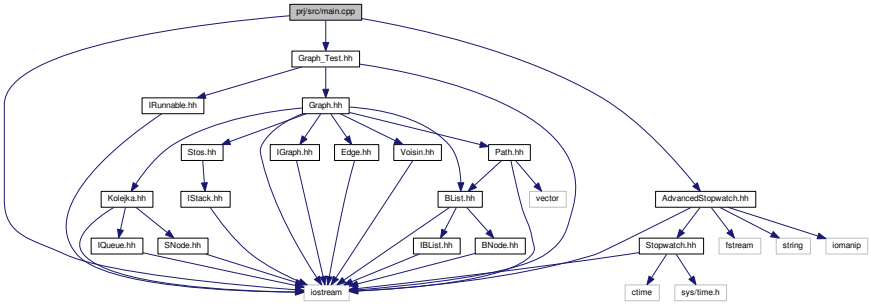
```
#include "Graph.hh"
```
Wykres zależności załączania dla Graph.cpp:



5.23 Dokumentacja pliku prj/src/main.cpp

```
#include <iostream>
#include "Graph_Test.hh"
#include "AdvancedStopwatch.hh"
```

Wykres zależności załączania dla main.cpp:



Funkcje

- int main ()

5.23.1 Dokumentacja funkcji

5.23.1.1 int main ( )

Definicja w linii 7 pliku main.cpp.

5.24 Dokumentacja pliku prj/src/Stopwatch.cpp

```
#include "Stopwatch.hh"
```

Wykres zależności załączania dla Stopwatch.cpp: