

Graf

Wygenerowano przez Doxygen 1.8.6

Pt, 6 maj 2016 21:34:04

Spis treści

1 Indeks hierarchiczny

1.1 Hierarchia klas

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

BNode< Object >	??
BNode< unsigned >	??
IGraph	??
Graph	??
Graph_Test< Object >	??
IList< Object >	??
BList< Object >	??
IList< unsigned >	??
BList< unsigned >	??
IQueue< Object >	??
Kolejka< Object >	??
IRunnable< Object >	??
Graph_Test< Object >	??
IStack< Object >	??
Stos< Object >	??
SNode< Object >	??
Stopwatch	??
AdvancedStopwatch	??

2 Indeks klas

2.1 Lista klas

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

AdvancedStopwatch	
Klasa implementująca rozbudowany stoper	??
BList< Object >	
Szablona klasa implementująca listę dwukierunkową	??
BNode< Object >	??
Graph	??

Graph_Test< Object >	
Szablonowa klasa implementująca testowy graf	??
IGraph	??
IList< Object >	
Interfejs listy dwukierunkowej	??
IQueue< Object >	
Klasa modelująca interfejs kolejki	??
IRunnable< Object >	
Klasa szablonowa modelująca interfejs "Biegacza"	??
IStack< Object >	
Klasa szablonowa modelująca interfejs stosu	??
Kolejka< Object >	
Klasa szablonowa implementująca kolejkę	??
SNode< Object >	??
Stopwatch	
Klasa implementująca podstawowy stoper	??
Stos< Object >	
Klasa szablonowa implementująca stos	??

3 Indeks plików

3.1 Lista plików

Tutaj znajduje się lista wszystkich plików z ich krótkimi opisami:

prj/inc/AdvancedStopwatch.hh	??
prj/inc/BList.hh	??
prj/inc/BNode.hh	??
prj/inc/Graph.hh	??
prj/inc/Graph_Test.hh	??
prj/inc/IGraph.hh	??
prj/inc/IList.hh	??
prj/inc/IQueue.hh	??
prj/inc/IRunnable.hh	??
prj/inc/IStack.hh	??
prj/inc/Kolejka.hh	??
prj/inc/SNode.hh	??
prj/inc/Stopwatch.hh	??
prj/inc/Stos.hh	??

prj/src/AdvancedStopwatch.cpp	??
prj/src/Graph.cpp	??
prj/src/main.cpp	??
prj/src/Stopwatch.cpp	??

4 Dokumentacja klas

4.1 Dokumentacja klasy AdvancedStopwatch

Klasa implementująca rozbudowany stoper.

```
#include <AdvancedStopwatch.hh>
```

Diagram dziedziczenia dla AdvancedStopwatch

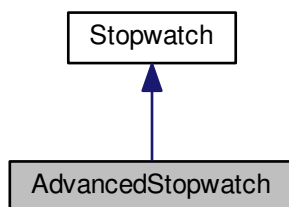
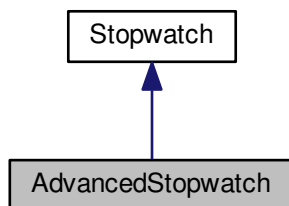


Diagram współpracy dla AdvancedStopwatch:



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 modelem 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 destruktoru

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 >

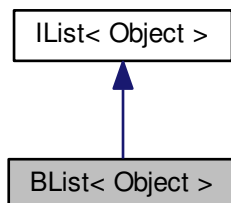
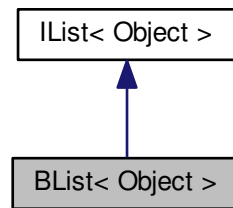


Diagram współpracy dla BList< Object >:



Metody publiczne

- **BList** ()
Konstruktor listy dwukierunkowej.
- **~BList** ()
Destruktor listy dwukierunkowej.
- **BNode**< Object > * & **Head** ()
Metoda zwracająca głowę listy.
- **BNode**< Object > * & **Tail** ()
Metoda zwracająca ogon listy.
- **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** ()

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 112 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 116 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 170 pliku BList.hh.

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

Implementuje `IList< Object >`.

Definicja w linii 156 pliku BList.hh.

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

Implementuje `IList< Object >`.

Definicja w linii 140 pliku BList.hh.

4.2.3.4 `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 znalezionej elementu lub NULL, gdy nie znaleziono.

Definicja w linii 217 pliku BList.hh.

4.2.3.5 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 128 pliku BList.hh.

4.2.3.6 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 [IList< Object >](#).

Definicja w linii 122 pliku BList.hh.

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

Definicja w linii 227 pliku BList.hh.

4.2.3.8 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 182 pliku BList.hh.

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

Implementuje [IList< Object >](#).

Definicja w linii 211 pliku BList.hh.

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

Implementuje [IList< Object >](#).

Definicja w linii 205 pliku BList.hh.

4.2.3.11 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 134 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 destruktor

```
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 Graph

```
#include <Graph.hh>
```

Diagram dziedziczenia dla Graph

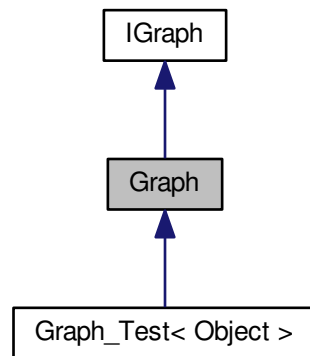
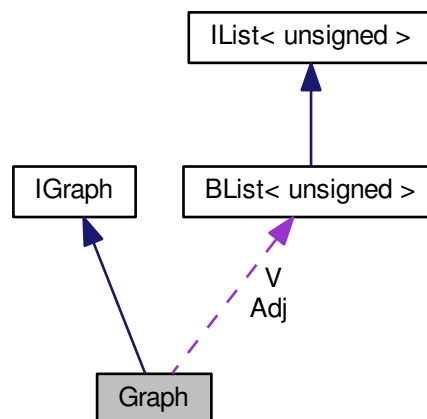


Diagram współpracy dla Graph:



Metody publiczne

- `Graph ()`
Konstruktor grafu.
- `Graph (int problemSize)`
Konstruktor grafu.
- `~Graph ()`
Destruktor grafu.

- **BList**< unsigned > * **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 wszerek Wykorzystuje implementację kolejki.
- void **DFS** (unsigned i)
Przechodzi graf wgłąb Wykorzystuje implementację stosu.

Atrybuty chronione

- **BList**< unsigned > **V**
- **BList**< unsigned > * **Adj**
- int **N**

4.4.1 Opis szczegółowy

Definicja w linii 15 pliku Graph.hh.

4.4.2 Dokumentacja konstruktora i destruktora

4.4.2.1 Graph::Graph ()

Konstruktor grafu.

Inicjuje graf poprzez utworzenie tablicy list sąsiedztwa. Domyślnie 10 wierzchołków.

Definicja w linii 118 pliku Graph.hh.

4.4.2.2 Graph::Graph (int *problemSize*)

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 124 pliku Graph.hh.

4.4.2.3 Graph::~~Graph ()

Destruktor grafu.

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

Definicja w linii 130 pliku Graph.hh.

4.4.3 Dokumentacja funkcji składowych

4.4.3.1 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 147 pliku Graph.hh.

4.4.3.2 void Graph::BFS (unsigned i)

Przechodzi graf wszerek Wykorzystuje implementację kolejki.

Parametry

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

in i - numer wierzchołka startowego

Definicja w linii 185 pliku Graph.hh.

4.4.3.3 void Graph::DFS (unsigned i)

Przechodzi graf wglab Wykorzystuje implementację stosu.

Parametry

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

in i - numer wierzchołka startowego

Definicja w linii 219 pliku Graph.hh.

4.4.3.4 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 162 pliku Graph.hh.

4.4.3.5 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 156 pliku Graph.hh.

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

Podaje maksymalną liczbę wierzchołków.

Zwraca

rozmiar tablicy list sąsiedztwa

Definicja w linii 96 pliku Graph.hh.

4.4.3.7 BList< unsigned > * 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 139 pliku Graph.hh.

4.4.3.8 void Graph::Print ()

Drukuje krawędzie grafu.

Definicja w linii 175 pliku Graph.hh.

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

Zwraca listę wierzchołków.

Zwraca

lista sąsiedztwa podanego wierzchołka

Definicja w linii 90 pliku Graph.hh.

4.4.4 Dokumentacja atrybutów składowych

4.4.4.1 BList<unsigned>* Graph::Adj [protected]

-> lista wierzchołków

Definicja w linii 19 pliku Graph.hh.

4.4.4.2 int Graph::N [protected]

-> tablica list sąsiedztwa

Definicja w linii 20 pliku Graph.hh.

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

Definicja w linii 18 pliku Graph.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- [prj/inc/Graph.hh](#)

4.5 Dokumentacja szablonu klasy Graph_Test< Object >

Szablonowa klasa implementująca testowy graf.

```
#include <Graph_Test.hh>
```

Diagram dziedziczenia dla Graph_Test< Object >

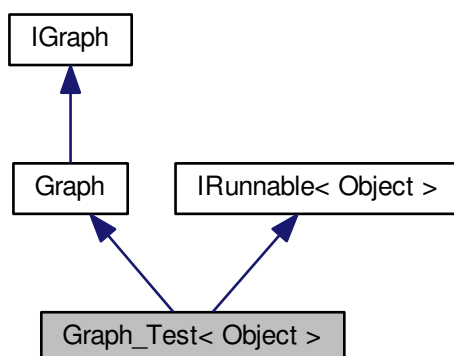
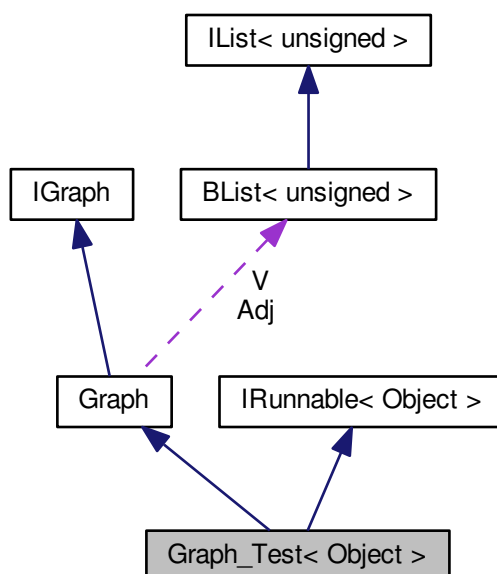


Diagram współpracy dla Graph_Test< Object >:



Metody publiczne

- void **changeSearchType** (char type)
Zmienia sposób przechodzenia grafu.
- 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** ()
Metoda uruchamiająca przejście grafu W zależności od ustawienia parametru searchType, uruchamia przejście BFS lub DFS.

Dodatkowe Dziedziczone Składowe

4.5.1 Opis szczegółowy

```
template<typename Object>class Graph_Test< Object >
```

Szablonowa klasa implementująca testowy graf.

Definicja w linii 17 pliku Graph_Test.hh.

4.5.2 Dokumentacja funkcji składowych

```
4.5.2.1 template<typename Object > void Graph_Test< Object >::changeSearchType ( char type )
```

Zmienia sposób przechodzenia grafu.

Ustawia podany typ znakowy jako wyznacznik przejścia grafu metodą BFS lub DFS.

Definicja w linii 56 pliku Graph_Test.hh.

4.5.2.2 `template<typename Object > bool Graph_Test< Object >::Prepare (Object parametr) [virtual]`

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.

Parametry

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

in *parametr* - liczba wierzchołków

Zwracane wartości

|>p0.25|p0.705|

true - jeśli operacja zakończyła się pomyślnie

false - jeśli wystąpił jakiś błąd

Implementuje [IRunnable< Object >](#).

Definicja w linii 62 pliku Graph_Test.hh.

4.5.2.3 `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 101 pliku Graph_Test.hh.

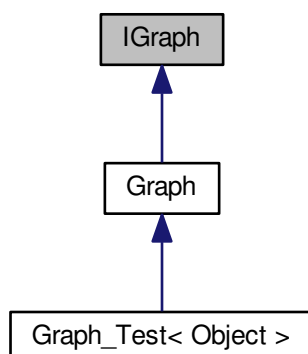
Dokumentacja dla tej klasy została wygenerowana z pliku:

- [prj/inc/Graph_Test.hh](#)

4.6 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

4.6.1 Opis szczegółowy

Definicja w linii 13 pliku IGraph.hh.

4.6.2 Dokumentacja funkcji składowych

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

Implementowany w [Graph](#).

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

Implementowany w [Graph](#).

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

Implementowany w [Graph](#).

Dokumentacja dla tej klasy została wygenerowana z pliku:

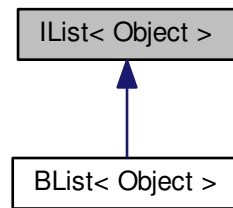
- [prj/inc/IGraph.hh](#)

4.7 Dokumentacja szablonu klasy IList< Object >

Interfejs listy dwukierunkowej.

```
#include <IList.hh>
```

Diagram dziedziczenia dla IList< 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 IList< Object >

Interfejs listy dwukierunkowej.

Definiuje ADT dla listy dwukierunkowej.

Lista może przechowywać dowolny typ danych dzięki zastosowaniu szablonu.

Definicja w linii 22 pliku IList.hh.

4.7.2 Dokumentacja funkcji składowych

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

Implementowany w [BList< Object >](#) i [BList< unsigned >](#).

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

Implementowany w [BList< Object >](#) i [BList< unsigned >](#).

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

Metoda sprawdzająca, czy lista jest pusta.

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

Implementowany w [BList< Object >](#) i [BList< unsigned >](#).

4.7.2.4 `template<typename Object> virtual const Object& IList< Object >::RemoveBack () [pure virtual]`

Implementowany w `BList< Object >` i `BList< unsigned >`.

4.7.2.5 `template<typename Object> virtual const Object& IList< Object >::RemoveFront () [pure virtual]`

Implementowany w `BList< Object >` i `BList< unsigned >`.

Dokumentacja dla tej klasy została wygenerowana z pliku:

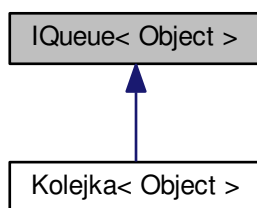
- `prj/inc/IList.hh`

4.8 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.

4.8.1 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`.

4.8.2 Dokumentacja funkcji składowych

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

Metoda usuwająca element kolejki.

Usuwa element z początku kolejki.

Zwraca

pierwszy element kolejki

Implementowany w [Kolejka< Object >](#).

4.8.2.2 `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 >](#).

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

Metoda zwracająca pierwszy element kolejki.

Zwraca

pierwszy element kolejki

Implementowany w [Kolejka< Object >](#).

4.8.2.4 `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:

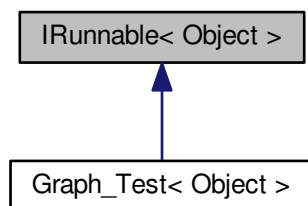
- [prj/inc/IQueue.hh](#)

4.9 Dokumentacja szablonu klasy IRunnable< Object >

Klasa szablona 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

4.9.1 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`.

4.9.2 Dokumentacja funkcji składowych

4.9.2.1 `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 >`.

4.9.2.2 `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:

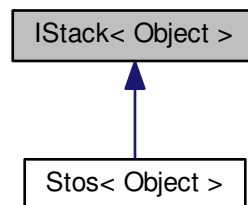
- [prj/inc/IRunnable.hh](#)

4.10 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.

4.10.1 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.

4.10.2 Dokumentacja funkcji składowych

4.10.2.1 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 >](#).

4.10.2.2 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 >](#).

4.10.2.3 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 >](#).

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

Metoda obliczająca rozmiar stosu.

Zwraca

liczba elementów na stos

Implementowany w [Stos< Object >](#).

4.10.2.5 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](#)

4.11 Dokumentacja szablonu klasy Kolejka< Object >

Klasa szablonowa implementująca kolejkę

```
#include <Kolejka.hh>
```

Diagram dziedziczenia dla Kolejka< Object >

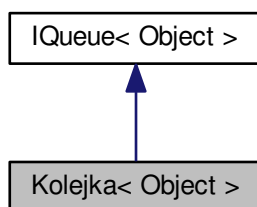
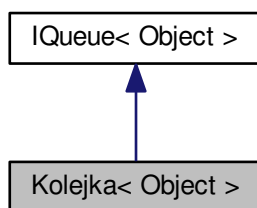
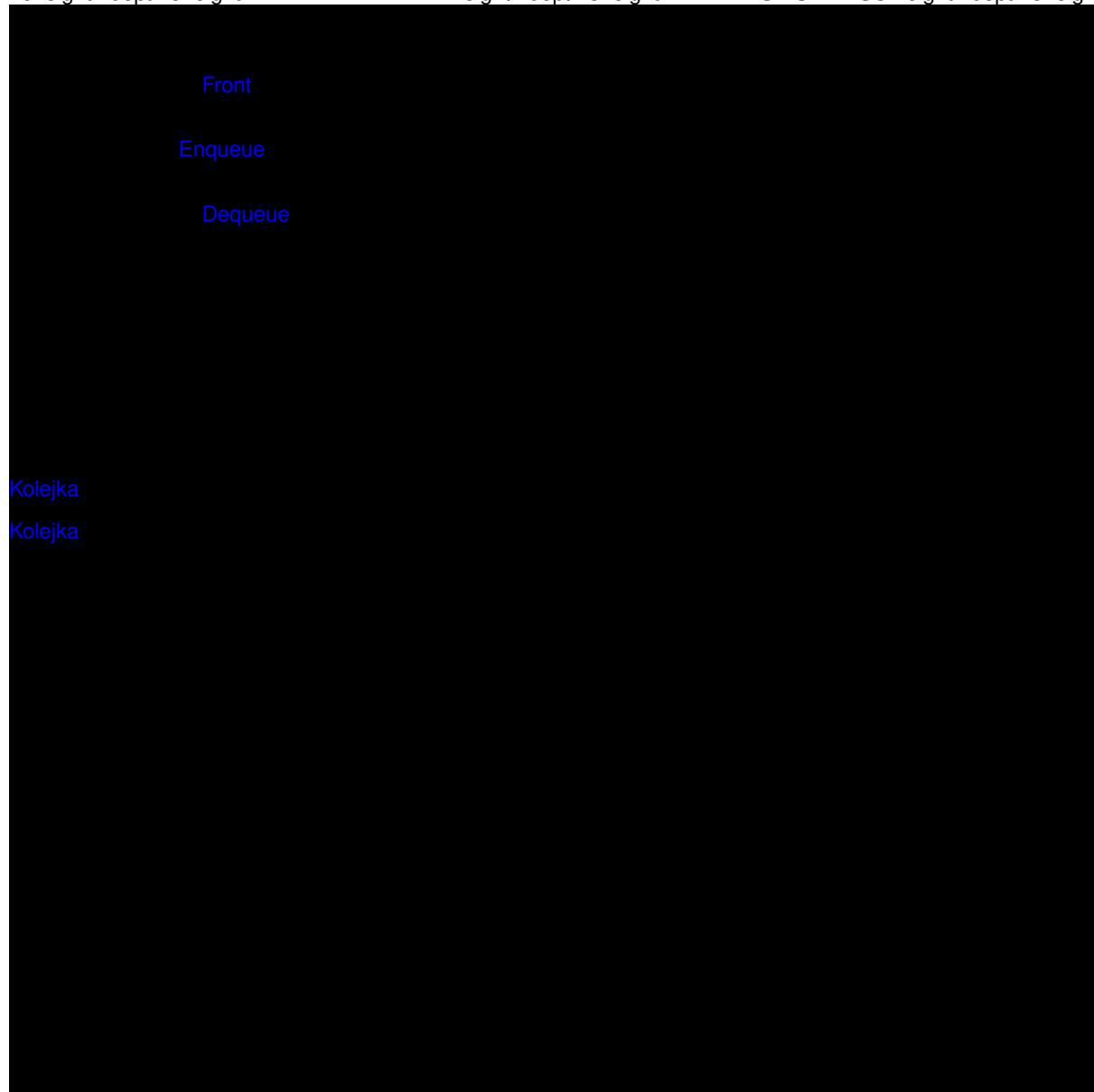


Diagram współpracy dla Kolejka< Object >:



Metody publiczne

- [Kolejka \(\)](#)
Konstruktor bezargumentowy kolejki.
- [~Kolejka \(\)](#)
Destruktor kolejki.
- virtual bool [IsEmpty \(\)](#)
Metoda sprawdzająca, czy kolejka jest pusta.



4.11.3 Dokumentacja funkcji składowych

4.11.3.1 `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 początku kolejki o indeksie `f` (front), a następnie przesuwając indeks `f` o jedno miejsce dalej.

Zwraca

pierwszy element kolejki

Implementuje `IQueue< Object >`.

Definicja w linii 131 pliku `Kolejka.hh`.

4.11.3.2 `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.

4.11.3.3 `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.

4.11.3.4 `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](#)

4.12 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](#))

4.12.1 Opis szczegółowy

```
template<typename Object>class SNode< Object >
```

Definicja w linii 8 pliku SNode.hh.

4.12.2 Dokumentacja konstruktora i destruktor

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

Definicja w linii 14 pliku SNode.hh.

4.12.3 Dokumentacja funkcji składowych

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

Definicja w linii 15 pliku SNode.hh.

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

Definicja w linii 16 pliku SNode.hh.

4.12.3.3 `template<typename Object> void SNode< Object >::SetElement (Object newItem) [inline]`

Definicja w linii 17 pliku SNode.hh.

4.12.3.4 `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:

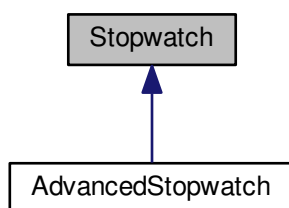
- [prj/inc/SNode.hh](#)

4.13 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](#)

4.13.1 Opis szczegółowy

Klasa implementująca podstawowy stoper.

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

Definicja w linii 20 pliku Stopwatch.hh.

4.13.2 Dokumentacja funkcji składowych

4.13.2.1 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.

4.13.2.2 void Stopwatch::Start () [virtual]

Rozpoczyna pomiar czasu.

Przypisuje wynik metody gettimeofday() do pola start.

Definicja w linii 4 pliku Stopwatch.cpp.

4.13.2.3 void Stopwatch::Stop () [virtual]

Kończy pomiar czasu.

Przypisuje wynik metody gettimeofday() do pola stop.

Definicja w linii 8 pliku Stopwatch.cpp.

4.13.3 Dokumentacja atrybutów składowych

4.13.3.1 timeval Stopwatch::start [protected]

Definicja w linii 23 pliku Stopwatch.hh.

4.13.3.2 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](#)

4.14 Dokumentacja szablonu klasy Stos< Object >

Klasa szablonowa implementująca stos.

```
#include <Stos.hh>
```

Diagram dziedziczenia dla Stos< Object >

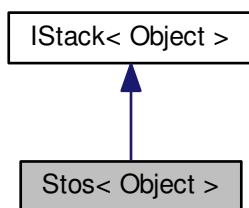
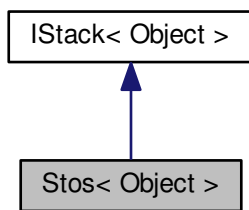


Diagram współpracy dla Stos< Object >:



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.

4.14.1 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.

4.14.2 Dokumentacja konstruktora i destruktor

4.14.2.1 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 119 pliku Stos.hh.

4.14.2.2 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 128 pliku Stos.hh.

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

Destruktor stosu.

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

Definicja w linii 140 pliku Stos.hh.

4.14.3 Dokumentacja funkcji składowych

4.14.3.1 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 147 pliku Stos.hh.

4.14.3.2 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.

4.14.3.3 `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 193 pliku Stos.hh.

4.14.3.4 `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 175 pliku Stos.hh.

4.14.3.5 `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 156 pliku Stos.hh.

4.14.3.6 `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 164 pliku Stos.hh.

Dokumentacja dla tej klasy została wygenerowana z pliku:

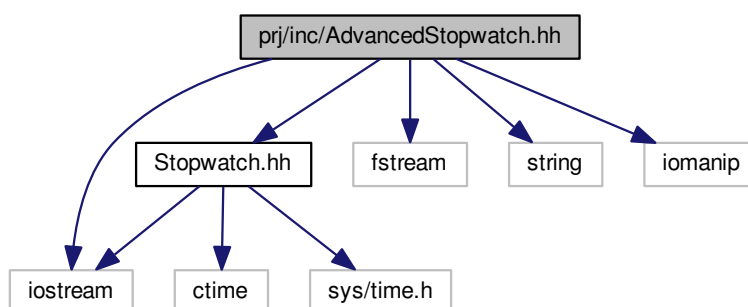
- [prj/inc/Stos.hh](#)

5 Dokumentacja plików

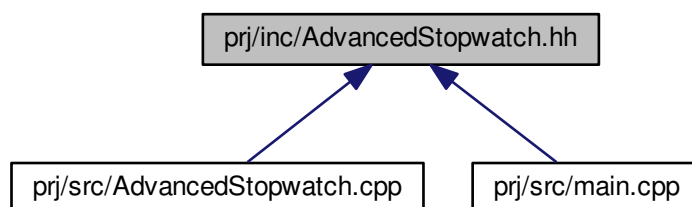
5.1 Dokumentacja pliku prj/inc/AdvancedStopwatch.hh

```
#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.

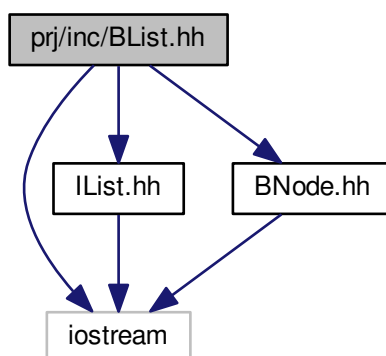
5.1.2.3 #define MAX_LAPS 100

Definicja w linii 11 pliku AdvancedStopwatch.hh.

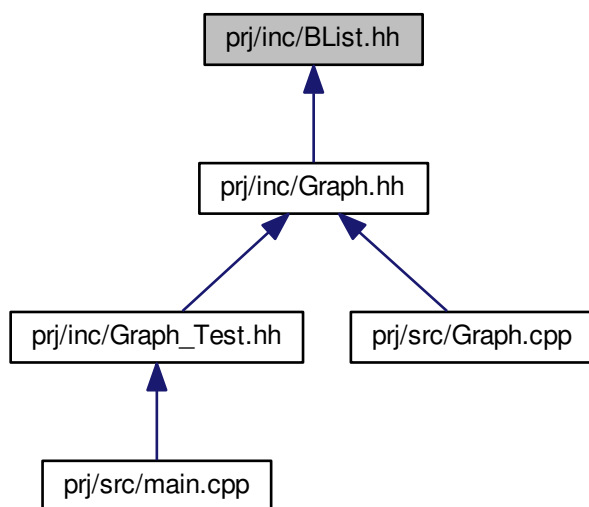
5.2 Dokumentacja pliku prj/inc/BList.hh

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

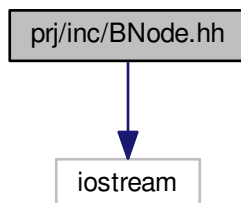
Plik zawiera definicję klasy implementującej listę dwukierunkową.

Definicja w pliku [BList.hh](#).

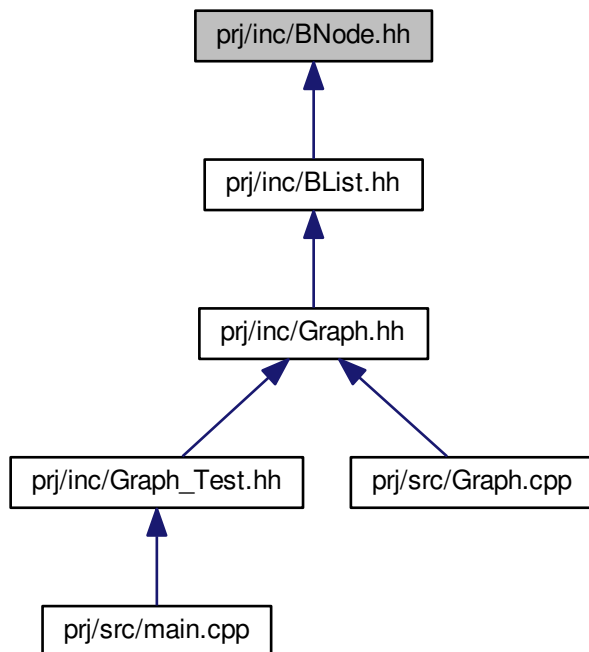
5.3 Dokumentacja pliku prj/inc/BNode.hh

```
#include <iostream>
```

Wykres zależności załączania dla BNode.hh:



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



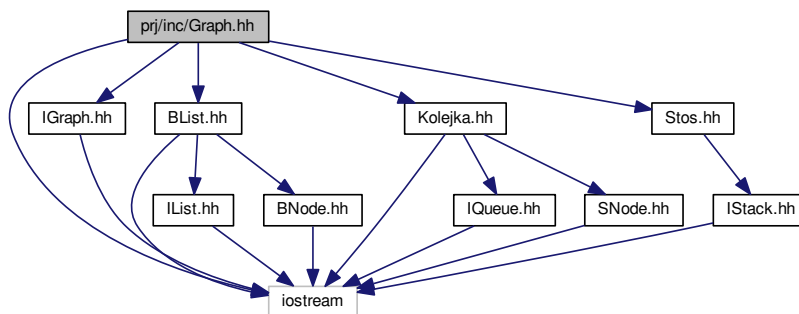
Komponenty

- class `BNode` < `Object` >

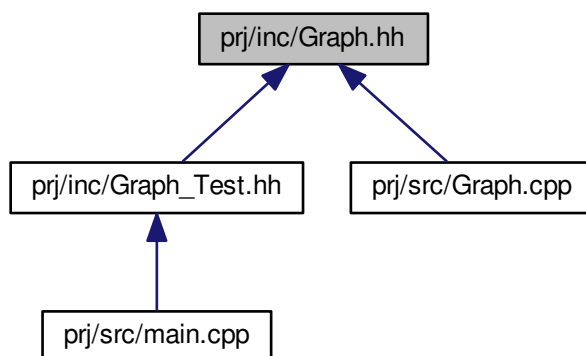
5.4 Dokumentacja pliku prj/inc/Graph.hh

```
#include <iostream>
#include "IGraph.hh"
#include "BList.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](#)

5.4.1 Opis szczegółowy

Plik zawiera implementację interfejsu grafu.

Definicja w pliku [Graph.hh](#).

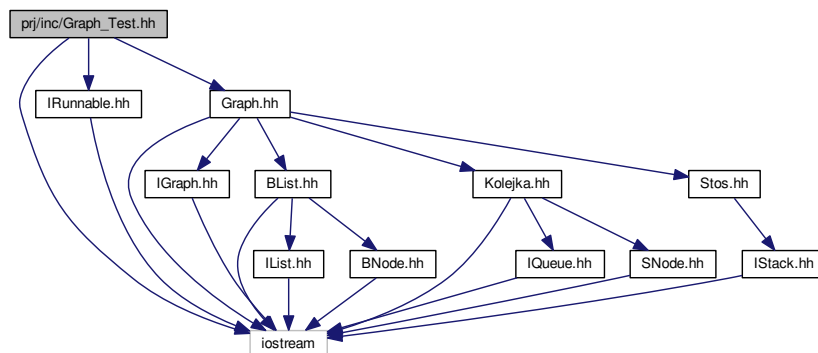
5.5 Dokumentacja pliku prj/inc/Graph_Test.hh

```

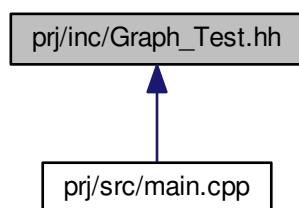
#include <iostream>
#include "IRunnable.hh"
#include "Graph.hh"

```

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.

5.5.1 Opis szczegółowy

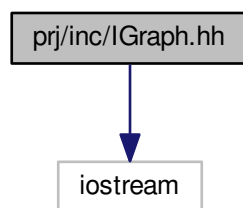
Plik zawiera implementację [IRunnable](#) dla grafu (operacje BFS i DFS).

Definicja w pliku [Graph_Test.hh](#).

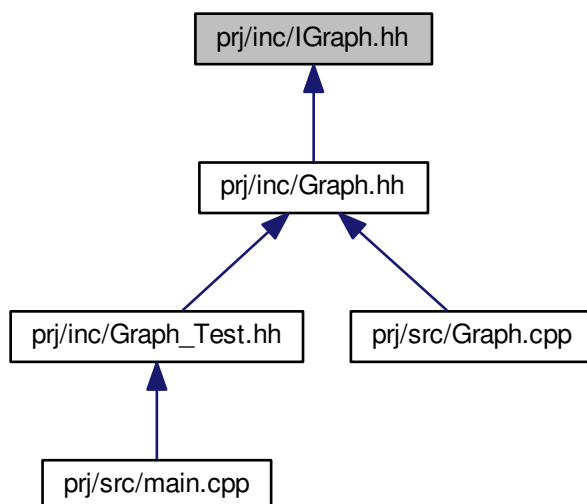
5.6 Dokumentacja pliku prj/inc/IGraph.hh

```
#include <iostream>
```

Wykres zależności załączania dla IGraph.hh:



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



Komponenty

- class `IGraph`

5.6.1 Opis szczegółowy

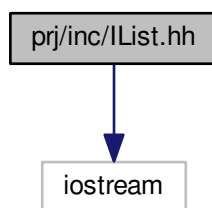
Plik zawiera interfejs grafu.

Definicja w pliku `IGraph.hh`.

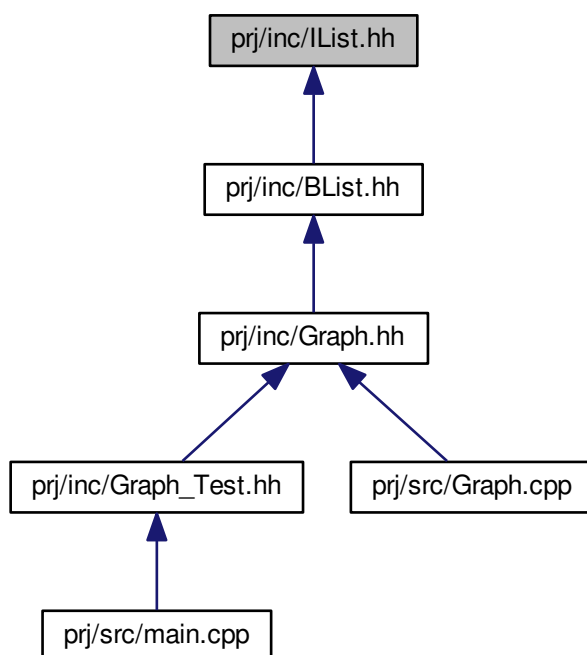
5.7 Dokumentacja pliku prj/inc/IList.hh

```
#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 dwukierunkowej.

5.7.1 Opis szczegółowy

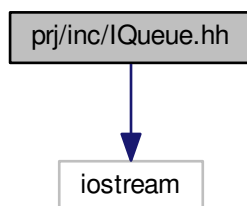
Plik zawiera interfejs listy dwukierunkową

Definicja w pliku [IList.hh](#).

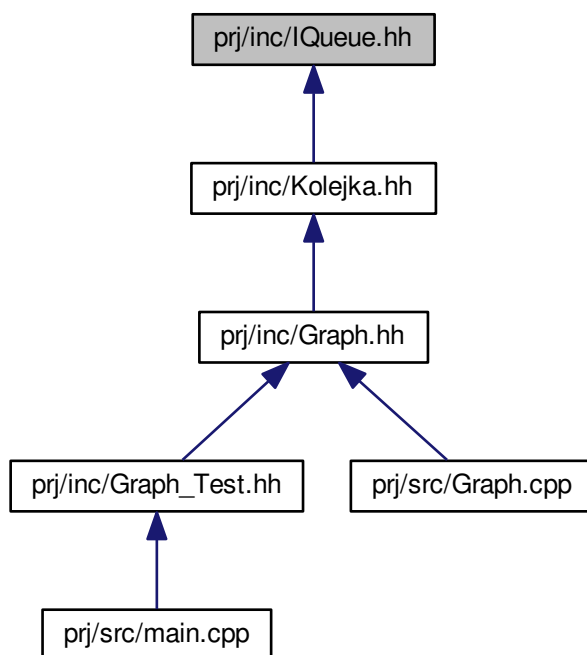
5.8 Dokumentacja pliku prj/inc/IQueue.hh

```
#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.

5.8.1 Opis szczegółowy

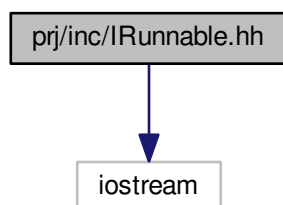
Plik zawiera interfejs kolejki

Definicja w pliku [IQueue.hh](#).

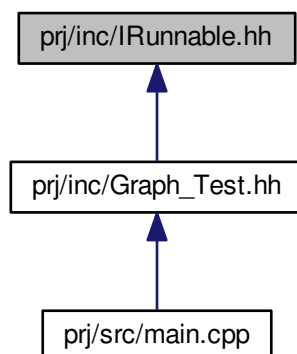
5.9 Dokumentacja pliku prj/inc/IRunnable.hh

```
#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".

5.9.1 Opis szczegółowy

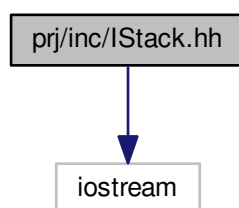
Plik zawiera interfejs obiektu, który można poddawać pomiarom czasu działania.

Definicja w pliku [IRunnable.hh](#).

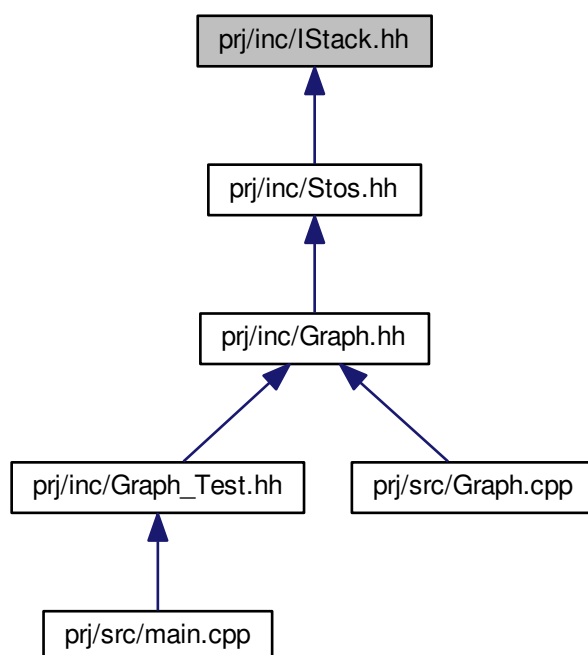
5.10 Dokumentacja pliku prj/inc/IStack.hh

```
#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.

5.10.1 Opis szczegółowy

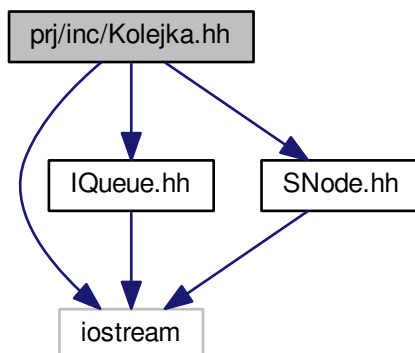
Plik zawiera interfejs stosu

Definicja w pliku `IStack.hh`.

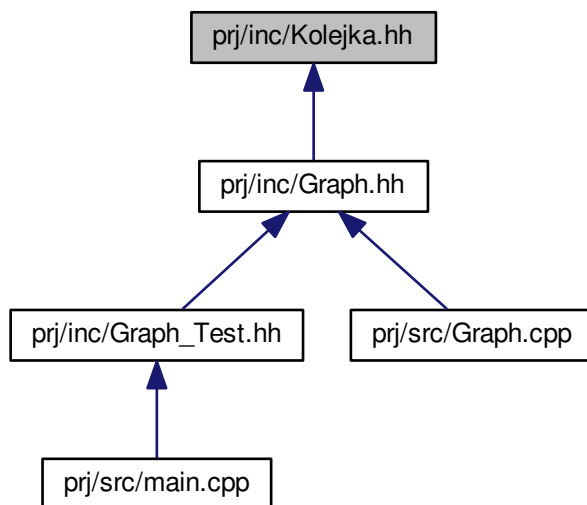
5.11 Dokumentacja pliku `prj/inc/Kolejka.hh`

```
#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

5.11.1 Opis szczegółowy

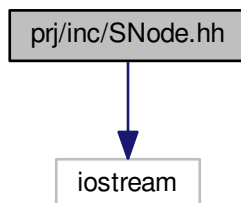
Plik zawiera implementację interfejsu kolejki

Definicja w pliku `Kolejka.hh`.

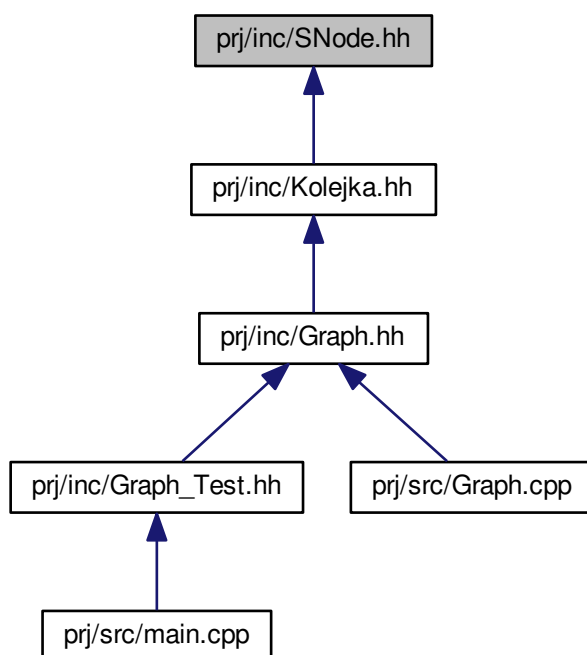
5.12 Dokumentacja pliku prj/inc/SNode.hh

```
#include <iostream>
```

Wykres zależności załączania dla `SNode.hh`:



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



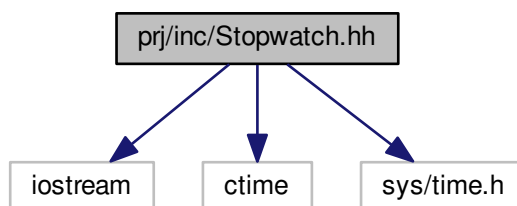
Komponenty

- class `SNode< Object >`

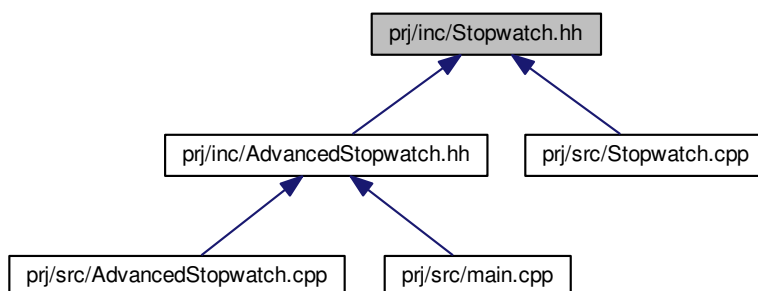
5.13 Dokumentacja pliku `prj/inc/Stopwatch.hh`

```
#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.

5.13.1 Opis szczegółowy

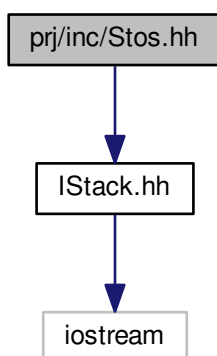
Plik zawiera implementację podstawowego stopera.

Definicja w pliku [Stopwatch.hh](#).

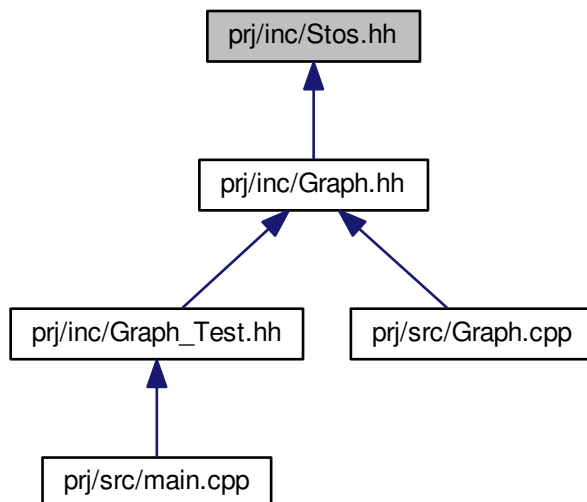
5.14 Dokumentacja pliku prj/inc/Stos.hh

```
#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.

5.14.1 Opis szczegółowy

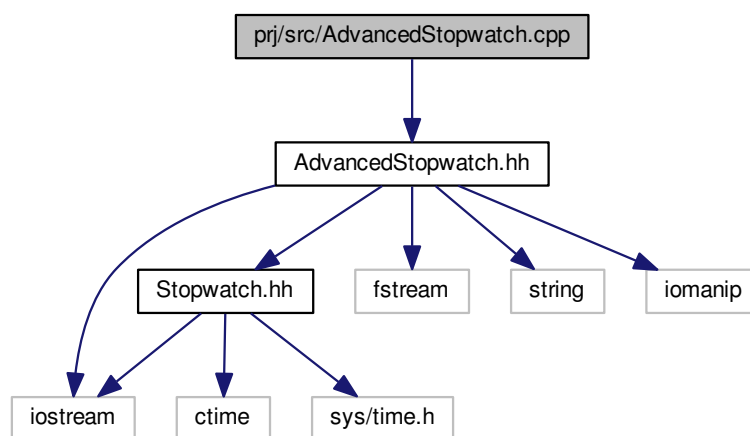
Plik zawiera implementację interfejsu stosu

Definicja w pliku [Stos.hh](#).

5.15 Dokumentacja pliku prj/src/AdvancedStopwatch.cpp

```
#include "AdvancedStopwatch.hh"
```

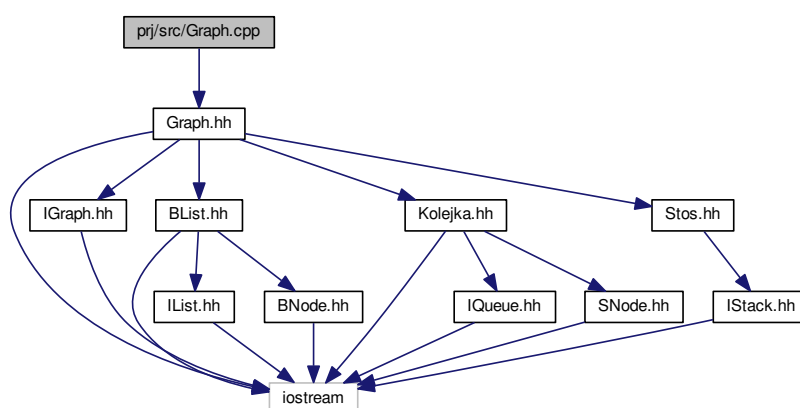
Wykres zależności załączania dla AdvancedStopwatch.cpp:



5.16 Dokumentacja pliku prj/src/Graph.cpp

```
#include "Graph.hh"
```

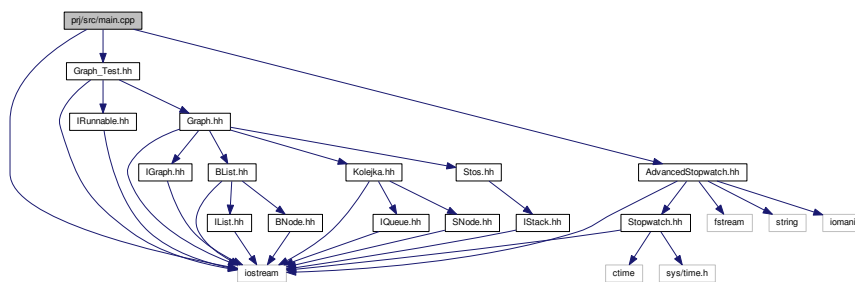
Wykres zależności załączania dla Graph.cpp:



5.17 Dokumentacja pliku prj/src/main.cpp

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

Wykres zależności załączania dla main.cpp:



Funkcje

- int [main](#) ()

5.17.1 Dokumentacja funkcji

5.17.1.1 int main ()

Definicja w linii 7 pliku main.cpp.

5.18 Dokumentacja pliku prj/src/Stopwatch.cpp

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

Wykres zależności załączania dla Stopwatch.cpp:

