Politechnika Śląska
Wydział Informatyki, Elektroniki i Informatyki

Podstawy Programowania Komputerów

Decyzja

Autor	Paweł Syska
Prowadzący	dr inż. Adam Gudyś
Rok akademicki	2019/2020
Kierunek	Informatyka
Rodzaj studiów	SSI
Semestr	1
Termin laboratorium	poniedziałek 10:15 – 11:45
Sekcja	12
Termin oddania sprawozdania	2020-01-23

1. Treść zadania

Napisać program dokonujący analizy zbioru danych z wykorzystaniem drzewa decyzyjnego. Program wykorzystuje dwa pliki wejściowe: jeden zawierający zbiór danych (przykładów opisanych pewnymi atrybutami), drugi opisujący strukturę drzewa. Działanie programu polega na przyporządkowaniu każdemu przykładowi ze zbioru pewnej etykiety na podstawie drzewa, a następnie zapisaniu przyporządkowań w pliku wyjściowym. Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

```
-i    plik wejściowy ze zbiorem danych do analizy
-t    plik wejściowy ze strukturą drzewa
-o    plik wyjściowy
```

2. Analiza zadania

Zagadnienie przedstawia problem przyporządkowania etykiet elementom ze zbioru danych, na podstawie kryteriów zawartych w drzewie decyzyjnym.

2.1 Struktury danych

W programie wykorzystano drzewo decyzyjne, które jest drzewem binarnym zbudowanym z węzłów posiadających od 0 do 2 potomków. Każdy węzeł reprezentuje wybór, który jest podejmowany przy analizie elementu zbioru danych. Jeśli warunek węzła dla danego elementu jest spełniony, to podejmowana jest ścieżka po prawej stronie węzła rodzicielskiego lub po lewej gdy warunek nie jest spełniony. Zamiast połączenia z potomkiem, możliwe jest zakończenie ścieżki wyboru i tym samym przypisanie elementowi ze zbioru danych etykiety ustalonej zamiast połączenia. Wykorzystanie tej struktury danych wynika z warunków zadania.

Elementy zbioru danych (rekordy) to zbiór liczb rzeczywistych przyporządkowanych pewnym nazwom (format przypominający tabelę z nazwanymi kolumnami). Do przechowywania poszczególnych rekordów użyto drzewa binarnego, które wykorzystuje nazwy atrybutów jako klucze. Ta struktura umożliwia szybki dostęp do wartości, korzystając z nazw atrybutów a to przyda się przy podejmowaniu decyzji.

Ilość atrybutów każdego z rekordów nie jest ściśle określona, więc zostały one umieszczone w liście jednokierunkowej. Taki wybór umożliwia zachowanie właściwej ich kolejności, co jest kluczowe przy wczytywaniu rekordów.

2.2 Algorytmy

W programie wykorzystano podstawowe algorytmy związane z listą jednokierunkową, drzewem binarnym oraz drzewem decyzyjnym: dodawanie elementów i przeszukiwanie. Utworzenie drzewa decyzyjnego odbywa się przez wstawianie kolejnych węzłów zgodnie z indeksami wejścia. Ze względu na to, że kolejność węzłów w pliku wejściowym drzewa nie jest ściśle określona, drzewo to musi być przeszukiwane celem znalezienia miejsca, w które należy wstawić następny węzeł (tj. znalezienia jego nowego rodzica). Drzewo decyzyjne nie jest posortowane, więc do szybkiej lokalizacji odpowiednich węzłów użyto drzew przeszukiwania binarnego. Takie rozwiązanie jest szybsze niż rekurencyjne przeszukiwanie całego drzewa (czas $O(\log n)$ dla drzewa i $O(n)$ dla przeszukiwania rekurencyjnego).

3. Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń. Należy przekazać do programu nazwy plików: wejściowego zbioru danych, wejściowego ze strukturą drzewa oraz wyjściowego, po odpowiednich przełącznikach:

```
-i    plik wejściowy ze zbiorem danych do analizy
-t    plik wejściowy ze strukturą drzewa
-o    plik wyjściowy
```

Przykład użycia:

```
program -i zbior_danych.txt -t drzewo.txt -o wyjście.txt
program -t drzewo.txt -o wyjście.txt -i zbior_danych.txt
```

Pliki są plikami tekstowymi ale mogą mieć dowolne rozszerzenie (lub go nie mieć). Przełączniki mogą być podane w dowolnej kolejności. Uruchomienie programu z parametrem **-h**

```
program -h
```

powoduje wyświetlenie krótkiej pomocy. Uruchomienie programu z nieprawidłowymi parametrami powoduje wyświetlenie komunikatu o błędzie, którego treść zależy od rodzaju błędu.

4. Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. Kod zawiera klasy oraz szablony użyte w celu zapewnienia przejrzystości i skalowalności kodu oraz poprawnego zarządzania pamięcią.

4.1 Ogólna struktura programu

Główna funkcja programu interpretuje argumenty oraz wywołuje funkcję `run`, realizującą logikę programu, z odpowiednim ustawieniem. Zajmuje się również obsługą błędów na najwyższym poziomie.

Funkcja `run`, zależnie od przekazanego trybu uruchomienia:

a) wyświetla komunikat pomocy (parametr `-h`)
b) wywołuje trzy kolejne etapy programu:
    1. wczytanie drzewa decyzyjnego (`readDecisionTree`)
    2. przeczytanie wejściowego zbioru danych i przypisanie odpowiednich etykiet (`parseInput`)
    3. zapisanie wyniku programu do pliku wyjściowego

Kod zawiera trzy główne klasy (i szablony klas) związane z drzewami: `BasicTree`, `BinarySearchTree` oraz `DecisionTree`.

`BasicTree` jest klasą bazową dla dwóch pozostałych i zapewnia poprawne zwolnienie pamięci.

4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

5. Testowanie

Program został przetestowany na różnych zbiorach danych. Pliki, które nie zawierają poprawnych danych powodują zgłoszenie błędu. Pusty plik spowoduje wygenerowanie pustego pliku wynikowego. Poprawnie przygotowane dane wejściowe powodują zwrócenie poprawnego wyniku. Drzewa były ręcznie przygotowywane a zbiór danych wejściowych generowany, w liczbach rekordów sięgających 1000000. Największym problemem okazała się wydajność funkcji z biblioteki standardowej do operowania na łańcuchach znaków. Przy 1000000 rekordów program na komputerze testowym potrzebował ok. 20 sekund na zakończenie działania (procesor AMD Ryzen 7 2700X).

Program został sprawdzony pod kątem wycieków pamięci.

6. Wnioski

Drzewo decyzyjne w formie, na której pracuje to zadanie potrzebuje stosunkowo mało pamięci operacyjnej do pracy. Mechanizm przydzielania etykiet nadaje się do zrównoleglenia, co znacznie polepszyłoby wydajność programu. Formatowane strumienie wejścia i wyjścia z biblioteki standardowej są powolne, w miarę możliwości powinno się używać jak najprymitywniejszych metod do analizy danych z tekstu. Wielki przyrost wydajności przyniosło sekwencyjne wczytywanie dużych obszarów pliku naraz.

Dodatek
Szczegółowy opis typów i funkcji

Decyzja (PPK Projekt)

Wygenerowano przez Doxygen 1.8.17

Rozdział 1

Indeks hierarchiczny

1.1 Hierarchia klas

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

Rozdział 2

Indeks klas

2.1 Lista klas

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

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury DecisionTreeNode::Anchor

`#include <DecisionTree.h>`

Metody publiczne

- **Anchor** ()=default
- **Anchor** (std::string label_)
- **Anchor** (std::uint32_t nodeIndex_)

Atrybuty publiczne

- std::string **label**
  etykieta
- std::uint32_t **nodeIndex** = 0
  identyfikator węzła
- bool **isLabel** = false
  czy połączenie jest etykietą?

3.1.1 Opis szczegółowy

Połączenie w drzewie decyzyjnym (albo id węzła albo etykieta)

3.1.2 Dokumentacja konstruktora i destruktora

3.1.2.1 Anchor() [1/3]

```
DecisionTreeNode::Anchor::Anchor ( )  [default]
```

Inicjalizuje instancje klasy **Anchor** (str. 5).

3.1.2.2 Anchor() [2/3]

```
DecisionTreeNode::Anchor::Anchor (
            std::string label_ )
```

Inicjalizuje instancje klasy **Anchor** (str. 5) etykietą.

Parametry

label↩ _	etykieta

3.1.2.3 Anchor() [3/3]

```
DecisionTreeNode::Anchor::Anchor (
            std::uint32_t nodeIndex_ )
```

Inicjalizuje instancje klasy **Anchor** (str. 5) identyfikatorem węzła.

Parametry

node↩ *Index_*	identyfikator węzła

Dokumentacja dla tej struktury została wygenerowana z plików:

- src/DecisionTree.h
- src/DecisionTree.cpp

3.2 Dokumentacja szablonu klasy BasicTree< T >

```
#include <BasicTree.h>
```

Typy publiczne

- using **Node** = T

Metody publiczne

- **BasicTree** ()=default
- **BasicTree** (**BasicTree** const &rhs_)=delete
- **BasicTree** (**BasicTree** &&rhs_)
- ∼**BasicTree** ()
- T ∗ **release** ()
- **BasicTree** & **operator=** (**BasicTree** const &rhs_)=delete
- **BasicTree** & **operator=** (**BasicTree** &&rhs_)

Atrybuty publiczne

- T ∗ **root** = nullptr

  korzeń drzewa

3.2.1 Opis szczegółowy

template<typename T>
class BasicTree< T >

Szablon klasy bazowej dla drzew binarnych. Szablon klasy bazowej dla drzew binarnych.

Parametry Szablonu

T	typ węzła

3.2.2 Dokumentacja konstruktora i destruktora

3.2.2.1 BasicTree() [1/3]

```
template<typename T >
BasicTree< T >::  BasicTree ( )  [default]
```

Inicjalizuje instancje klasy **BasicTree** (str. 6).

3.2.2.2 BasicTree() [2/3]

```
template<typename T >
BasicTree< T >::  BasicTree (
              BasicTree< T > const & rhs_ )  [delete]
```

Usunięty konstruktor kopiujący klasy **BasicTree** (str. 6).

Parametry

*rhs_* ↩	drugi obiekt.

3.2.2.3 BasicTree() [3/3]

```
template<typename T >
BasicTree< T >::  BasicTree (
              BasicTree< T > && rhs_ )  [inline]
```

Inicjalizuje instancje klasy **BasicTree** (str. 6) przenosząc wartość z rhs_.

Parametry

*rhs_*	inny obiekt, z którego drzewo zostanie przeniesione

3.2.2.4 ~BasicTree()

```
template<typename T >
BasicTree< T >::~ BasicTree ( )  [inline]
```

Niszczy instancje klasy **BasicTree** (str. 6).

3.2.3 Dokumentacja funkcji składowych

3.2.3.1 operator=() [1/2]

```
template<typename T >
BasicTree&  BasicTree< T >::operator= (
              BasicTree< T > && rhs_ )  [inline]
```

Przenosi wartość z rhs_ do siebie.

Parametry

*rhs_*	inny obiekt, z którego drzewo zostanie przeniesione

Zwraca

referencja na siebie

3.2.3.2 operator=() [2/2]

```
template<typename T >
BasicTree&  BasicTree< T >::operator= (
              BasicTree< T > const & rhs_ )  [delete]
```

Usunięty kopiujący operator przypisania klasy **BasicTree** (str. 6).

Parametry

rhs↩ _	inny obiekt

Zwraca

referencja na siebie

3.2.3.3 release()

```
template<typename T >
T*  BasicTree< T >::release ( )  [inline]
```

Porzuca wskaźnik na korzeń drzewa, zwracając go

Zwraca

wskaźnik na korzeń drzewa lub nullptr.

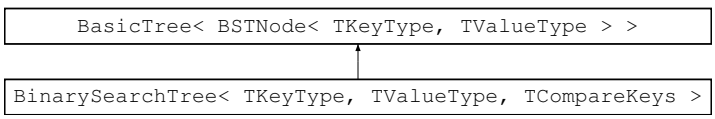
Dokumentacja dla tej klasy została wygenerowana z pliku:

- src/BasicTree.h

3.3 Dokumentacja szablonu klasy BinarySearchTree< TKeyType, TValueType, TCompareKeys >

```
#include <BinarySearchTree.h>
```

Diagram dziedziczenia dla BinarySearchTree< TKeyType, TValueType, TCompareKeys >



Typy publiczne

- using **KeyType** = TKeyType
- using **ValueType** = TValueType
- using **CompareKeys** = TCompareKeys
- using **Node** = **BSTNode**< TKeyType, TValueType >

Metody publiczne

- ValueType & **set** (KeyType key_, ValueType value_, CompareKeys comp_=CompareKeys())
- ValueType & **get** (KeyType const &key_, CompareKeys comp_=CompareKeys()) const
- ValueType * **tryGet** (KeyType const &key_, CompareKeys comp_=CompareKeys()) const
- template<typename TWhatToDo >
  void **forEach** (TWhatToDo whatToDo_)
- template<typename TWhatToDo >
  void **forEach** (TWhatToDo whatToDo_) const

Metody prywatne

- template<typename TWhatToDo >
  void **forEach** (**Node** &node_, TWhatToDo whatToDo_) const

Dodatkowe Dziedziczone Składowe

3.3.1 Opis szczegółowy

template<typename TKeyType, typename TValueType, typename TCompareKeys = std::less<TKeyType const&>>
class BinarySearchTree< TKeyType, TValueType, TCompareKeys >

Drzewo poszukiwania binarnego

3.3.2 Dokumentacja funkcji składowych

3.3.2.1 forEach() [1/3]

```
template<typename TKeyType , typename TValueType , typename TCompareKeys = std::less<TKeyType
const&>>
template<typename TWhatToDo >
void  BinarySearchTree< TKeyType, TValueType, TCompareKeys >::forEach (
             Node & node_,
            TWhatToDo whatToDo_ ) const  [inline], [private]
```

Wykonuje whatToDo_ dla każdego podwęzła elementu node_ w drzewie.

Parametry Szablonu

TWhatToDo	typ funkcji whatToDo_ o sygnaturze typ(KeyType const&, ValueType const&)

Parametry

*node_*	element drzewa
*whatTo↩ Do_*	funkcja operująca na kluczu i wartości węzła

3.3.2.2 forEach() [2/3]

```
template<typename TKeyType , typename TValueType , typename TCompareKeys = std::less<TKeyType
const&>>
template<typename TWhatToDo >
void  BinarySearchTree< TKeyType, TValueType, TCompareKeys >::forEach (
            TWhatToDo whatToDo_ )  [inline]
```

Wykonuje whatToDo_ dla każdego elementu drzewa.

Parametry Szablonu

TWhatToDo	typ funkcji whatToDo_ o sygnaturze typ(KeyType const&, ValueType const&)

Parametry

*whatTo↩ Do_*	funkcja operująca na kluczu i wartości węzła

3.3.2.3 forEach() [3/3]

```
template<typename TKeyType , typename TValueType , typename TCompareKeys = std::less<TKeyType
const&>>
template<typename TWhatToDo >
void  BinarySearchTree< TKeyType, TValueType, TCompareKeys >::forEach (
            TWhatToDo whatToDo_ ) const  [inline]
```

Wykonuje niemodyfikującą funkcję whatToDo_ dla każdego elementu drzewa.

Parametry Szablonu

TWhatToDo	typ funkcji whatToDo_ o sygnaturze typ(KeyType const&, ValueType const&)

Parametry

*whatTo↩ Do_*	funkcja operująca na kluczu i wartości węzła

3.3.2.4 get()

```
template<typename TKeyType , typename TValueType , typename TCompareKeys = std::less<TKeyType
const&>>
```

```
ValueType&  BinarySearchTree< TKeyType, TValueType, TCompareKeys >::get (
             KeyType const & key_,
             CompareKeys comp_ = CompareKeys() ) const  [inline]
```

Zwraca referencję na wartość węzła o danym kluczu. Rzuca wyjątek, jeśli węzła o danym kluczu nie znaleziono.

Parametry

*key_*	klucz
*comp_*	funkcja porównująca dwa klucze (lhs < rhs)

Zwraca

wartość

3.3.2.5 set()

```
template<typename TKeyType , typename TValueType , typename TCompareKeys = std::less<TKeyType
const&>>
ValueType&  BinarySearchTree< TKeyType, TValueType, TCompareKeys >::set (
             KeyType key_,
             ValueType value_,
             CompareKeys comp_ = CompareKeys() )  [inline]
```

Wstawia wartość do drzewa pod danym kluczem lub ustawia wartość już istniejącemu węzłowi.

Parametry

*key_*	klucz
*value_*	wartość
*comp_*	funkcja porównująca dwa klucze (lhs < rhs)

Zwraca

referencja na wstawioną wartość

3.3.2.6 tryGet()

```
template<typename TKeyType , typename TValueType , typename TCompareKeys = std::less<TKeyType
const&>>
ValueType*  BinarySearchTree< TKeyType, TValueType, TCompareKeys >::tryGet (
```

```
KeyType const & key_,
CompareKeys comp_ = CompareKeys() ) const  [inline]
```

Zwraca wskaźnik na wartość węzła o danym kluczu. Zwraca nullptr, jeśli węzła o danym kluczu nie znaleziono.

Parametry

*key_*	klucz
*comp_*	funkcja porównująca dwa klucze (lhs $<$ rhs)

Zwraca

wartość

Dokumentacja dla tej klasy została wygenerowana z pliku:

- src/BinarySearchTree.h

3.4 Dokumentacja szablonu klasy BSTNode< TKeyType, TValueType >

`#include <BinarySearchTree.h>`

Metody publiczne

- **BSTNode** ()=default
- **BSTNode** (TKeyType key_, TValueType value_)
- **BSTNode** (**BSTNode** const &&rhs_)=delete
- **BSTNode** (**BSTNode** &&rhs_)
- ~**BSTNode** ()

Atrybuty publiczne

- TKeyType **key**

  klucz węzła
- TValueType **value**

  wartość węzła
- **BSTNode** $*$ **left** = nullptr

  wskaźnik na korzeń lewego poddrzewa
- **BSTNode** $*$ **right** = nullptr

  wskaźnik na korzeń prawego poddrzewa

3.4.1 Opis szczegółowy

template<typename TKeyType, typename TValueType>
class BSTNode< TKeyType, TValueType >

Węzeł drzewa poszukiwania binarnego

3.4.2 Dokumentacja konstruktora i destruktora

3.4.2.1 BSTNode() [1/4]

```
template<typename TKeyType , typename TValueType >
BSTNode< TKeyType, TValueType >::  BSTNode ( )  [default]
```

Inicjalizuje instancje klasy **BSTNode** (str. 14).

3.4.2.2 BSTNode() [2/4]

```
template<typename TKeyType , typename TValueType >
BSTNode< TKeyType, TValueType >::  BSTNode (
            TKeyType key_,
            TValueType value_ )  [inline]
```

Inicjalizuje instancje klasy **BSTNode** (str. 14) za pomocą klucza i wartości.

Parametry

key↩ _	klucz
value↩ _	wartość

3.4.2.3 BSTNode() [3/4]

```
template<typename TKeyType , typename TValueType >
BSTNode< TKeyType, TValueType >::  BSTNode (
             BSTNode< TKeyType, TValueType > const && rhs_ )  [delete]
```

Usunięty konstruktor kopiujący klasy **BSTNode** (str. 14).

Parametry

rhs↩ _	drugi obiekt

3.4.2.4 BSTNode() [4/4]

```
template<typename TKeyType , typename TValueType >
BSTNode< TKeyType, TValueType >::  BSTNode (
             BSTNode< TKeyType, TValueType > && rhs_ )  [inline]
```

Inicjalizuje instancje klasy **BSTNode** (str. 14) przenosząc wartość z rhs_.

Parametry

rhs↩ _	drugi obiekt

3.4.2.5 ∼BSTNode()

```
template<typename TKeyType , typename TValueType >
BSTNode< TKeyType, TValueType >::~ BSTNode ( )  [inline]
```

Niszczy instancje klasy **BSTNode** (str. 14). Niszczy również lewe i prawe poddrzewo.

Dokumentacja dla tej klasy została wygenerowana z pliku:

- src/BinarySearchTree.h

3.5 Dokumentacja struktury DecisionTreeNode::Condition

```
#include <DecisionTree.h>
```

Typy publiczne

- enum **Op** { **LowerThan**, **GreaterThan** }

Atrybuty publiczne

- std::string **attributeName**
  nazwa porównywanego atrybutu
- **Op op**
  operator porównania
- double **value**
  wartość z którą atrybut jest porównywany

3.5.1 Opis szczegółowy

Warunek drzewa decyzyjnego

3.5.2 Dokumentacja składowych wyliczanych

3.5.2.1 Op

```
enum DecisionTreeNode::Condition::Op
```

Rodzaj operatora

Wartości wyliczeń

LowerThan	operator "<"
GreaterThan	operator ">"

Dokumentacja dla tej struktury została wygenerowana z pliku:

- src/DecisionTree.h

3.6 Dokumentacja klasy DecisionTreeNode

`#include <DecisionTree.h>`

Komponenty

- struct **Anchor**
- struct **Condition**

Metody publiczne

- **DecisionTreeNode** ()=default
- **DecisionTreeNode** (**DecisionTreeNode** &&rhs_)
- ~**DecisionTreeNode** ()

Atrybuty publiczne

- std::uint32_t **index** = 0

  indeks węzła
- **Condition cond**

  warunek
- **Anchor failedAnchor**

  połączenie dla niespełnienia warunku
- **Anchor succeededAnchor**

  połączenie dla spełnienia warunku
- **DecisionTreeNode** * **failed** = nullptr

  wskaźnik na węzeł niespełnionego warunku (ważny tylko gdy failedAnchor.isLabel == false)
- **DecisionTreeNode** * **succeeded** = nullptr

  wskaźnik na węzeł spełnionego warunku (ważny tylko gdy succeededAnchor.isLabel == false)

3.6.1 Opis szczegółowy

Węzeł drzewa decyzyjnego

3.6.2 Dokumentacja konstruktora i destruktora

3.6.2.1 DecisionTreeNode() [1/2]

```
DecisionTreeNode::DecisionTreeNode ( )  [default]
```

Inicjalizuje instancje klasy **DecisionTreeNode** (str. 17).

3.6.2.2 DecisionTreeNode() [2/2]

```
DecisionTreeNode::DecisionTreeNode (
            DecisionTreeNode && rhs_ )
```

Inicjalizuje instancje klasy **DecisionTreeNode** (str. 17) przenosząc wartość z rhs_.

Parametry

*rhs_*	inny obiekt, z którego wartość zostanie przeniesiona

3.6.2.3 ~DecisionTreeNode()

```
DecisionTreeNode::~DecisionTreeNode ( )
```

Niszczy instancje klasy **DecisionTreeNode** (str. 17).

Dokumentacja dla tej klasy została wygenerowana z plików:

- src/DecisionTree.h
- src/DecisionTree.cpp

3.7 Dokumentacja struktury ExecSetup

```
#include <ArgumentParsing.h>
```

Atrybuty publiczne

- std::string **inputFile**
  Ścieżka do pliku wejściowego rekordów.
- std::string **treeFile**
  Ścieżka do pliku wejściowego drzewa.
- std::string **outputFile**
  Ścieżka do pliku wyjściowego.
- bool **help** = false
  Czy wyświetlić pomoc?

3.7.1 Opis szczegółowy

Parsowanie argumentów Informacje o trybie uruchomienia.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- src/ArgumentParsing.h

3.8 Dokumentacja szablonu klasy ForwardList< T >

`#include <ForwardList.h>`

Komponenty

- class **Node**

Metody publiczne

- ~**ForwardList** ()
- void **clear** ()
- template<typename TAcceptFunc >
  Node * **findIf** (TAcceptFunc accept_) const
- **Node** & **push** (T value_)

Atrybuty publiczne

- **Node** * **head** = nullptr
  głowa listy (pierwszy element)
- **Node** * **tail** = nullptr
  ogon listy (ostatni element)

3.8.1 Opis szczegółowy

template<typename T>
class ForwardList< T >

Szablon listy jednokierunkowej. Szablon listy jednokierunkowej.

Parametry Szablonu

T	typ danych przechowywanych w liście

3.8.2 Dokumentacja konstruktora i destruktora

3.8.2.1 ~ForwardList()

```
template<typename T >
ForwardList< T >::~ ForwardList ( )  [inline]
```

Niszczy instancje szablonu klasy **ForwardList** (str. 19).

3.8.3 Dokumentacja funkcji składowych

3.8.3.1 clear()

```
template<typename T >
void  ForwardList< T >::clear ( )  [inline]
```

Czyści listę, zwalniając pamięć.

3.8.3.2 findIf()

```
template<typename T >
template<typename TAcceptFunc >
Node*  ForwardList< T >::findIf (
             TAcceptFunc accept_ ) const  [inline]
```

Szuka wartości, korzystając z funkcji accept

Parametry Szablonu

TAcceptFunc	typ pozwalający wykonać funkcję o sygnaturze bool(T const&)

Parametry

*accept_*	wartość typu TAcceptFunc zwracająca true dla wartości węzła, który ma być zwrócony

Zwraca

wskaźnik na znaleziony węzeł lub nullptr jeśli nie znaleziono

3.8.3.3 push()

```
template<typename T >
Node& ForwardList< T >::push (
            T value_ ) [inline]
```

Dodaje wartość na koniec.

Parametry

*value_*	wartość do dodania

Zwraca

referencja na węzeł zawierający wartość (ogon listy)

Dokumentacja dla tej klasy została wygenerowana z pliku:

- src/ForwardList.h

3.9 Dokumentacja struktury LessAttributeName

```
#include <AttributeTree.h>
```

Metody publiczne

- bool **operator()** (std::string const ∗lhs_, std::string const ∗rhs_)

3.9.1 Opis szczegółowy

Drzewo atrybutów dla rekordów. Funktor porównujący dwa stringi za pomocą wskaźników

3.9.2 Dokumentacja funkcji składowych

3.9.2.1 operator()()

```
bool LessAttributeName::operator() (
            std::string const * lhs_,
            std::string const * rhs_ ) [inline]
```

Operator porównujący dwa stringi za pomocą wskaźników

Parametry

lhs↩ _	lewy łańcuch
rhs↩ _	prawy łańcuch

Zwraca

true jeśli lewy łańcuch jest mniejszy od prawego.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- src/AttributeTree.h

3.10 Dokumentacja klasy ForwardList< T >::Node

`#include <ForwardList.h>`

Metody publiczne

- **Node** (T v_)

Atrybuty publiczne

- T **value**
  przechowywana wartość
- **Node** * **next** = nullptr
  wskaźnik na następny węzeł

3.10.1 Opis szczegółowy

template<typename T>
class ForwardList< T >::Node

Węzeł listy.

3.10.2 Dokumentacja konstruktora i destruktora

3.10.2.1 Node()

```
template<typename T >
ForwardList< T >::Node::Node (
            T v_ )  [inline]
```

Inicjalizuje instancje klasy **Node** (str. 22).

Parametry

$v\hookleftarrow$ $\_\hookleftarrow$	wartość do przechowania

Dokumentacja dla tej klasy została wygenerowana z pliku:

- src/ForwardList.h

Indeks