My Project

Generated by Doxygen 1.8.12

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 BPT< Type > Class Template Reference

Public Member Functions

- BPT ()
- BPT (int)
- void put (Type)

    This method just calls the private method void rec_put(Type, SequenceSet< Type>∗)
- bool remove (Type)

    This method just calls the private method void rec_remove(Type, SequenceSet< Type>∗)
- void print ()

    This method prints all the items in the B+tree.

3.1.1 Constructor & Destructor Documentation

3.1.1.1 BPT() [1/2]

```
template<class Type >
BPT< Type >::BPT ( )
```

Default Constructor code

endcode

3.1.1.2 BPT() [2/2]

```
template<class Type >
BPT< Type >::BPT (
              int order )
```

Copy Constructor code

endcode

3.1.2 Member Function Documentation

3.1.2.1 print()

```
template<class Type >
void BPT< Type >::print ( )
```

This method prints all the items in the B+tree.

Precondition

The tree has items in it

Postcondition

prints all the items

code

endcode

3.1.2.2 put()

```
template<class Type >
void BPT< Type >::put (
             Type item )
```

This method just calls the private method void rec_put(Type, SequenceSet<Type>∗)

Parameters

item	is an int or string

Precondition

Postcondition

the item is placed at the correct location

code

endcode

3.1.2.3 remove()

```
template<class Type >
bool BPT< Type >::remove (
             Type item )
```

This method just calls the private method void rec_remove(Type, SequenceSet<Type>∗)

Parameters

item	is an int or string

Precondition

Postcondition

the item is removed from the location

code

endcode

The documentation for this class was generated from the following file:

- BPT.h

3.2 DualHeap< Type > Class Template Reference

Public Member Functions

- DualHeap ()
- DualHeap (int)
- void put (Type)

  puts item into heap
- vector< vector< Type > > retrieve ()

  returns list of lists

3.2.1 Constructor & Destructor Documentation

3.2.1.1 DualHeap() [1/2]

```
template<class Type >
DualHeap< Type >::DualHeap ( )
```

Default constructor

```
 */
// Default heapsize is 3
HEAPSIZE = 3;
data.reserve(HEAPSIZE);
data.resize(HEAPSIZE);
direction = true;
current_size = 0;
pending_size = 0;
```

3.2.1.2 DualHeap() [2/2]

```
template<class Type >
DualHeap< Type >::DualHeap (
              int heapsize )
```

Copy constructor

```
 */
HEAPSIZE = heapsize;
data.reserve(HEAPSIZE);
data.resize(HEAPSIZE);
direction = true;
current_size = 0;
pending_size = 0;
```

3.2.2 Member Function Documentation

3.2.2.1 put()

```
template<class Type >
void DualHeap< Type >::put (
              Type item )
```

puts item into heap

Parameters

item	is a Type for the item

Precondition

checks whether heap is full with pending items

Postcondition

the item is inserted in the heap

```
 */
if (fullwith_pending()){ // full of pending items => push a run to buffer
    buffer.push_back(run);
    run = vector<Type>(); // empty output run
    direction = not direction;
    assert(current_size == 0);
    current_size = pending_size;
    pending_size = 0;
}
if (!full()){ // exists a spot for new item
    current_heap_push(item);
}else{ // full of active and pending items
    maxmin = current_heap_pop();    // pop from active
    if (maxmin > item){ // item goes to pending
        run.push_back(maxmin);          // popped item goes to run
        pending_heap_push(item);        // push to pending heap
    }else{ // item goes to active
        run.push_back(maxmin);          // popped item goes to run
        current_heap_push(item);        // push to active heap
    }
}
```

3.2.2.2 retrieve()

```
template<class Type >
vector< vector< Type > > DualHeap< Type >::retrieve ( )
```

returns list of lists

Precondition

there should some lists to put in the buffer

Postcondition

the buffer holding the lists is returned

```
 */
finalize();
return buffer;
```

The documentation for this class was generated from the following file:

- DualHeap.h

3.3 SequenceSet< Type > Class Template Reference

Public Member Functions

- SequenceSet ()
- SequenceSet (int)
- void setNext (SequenceSet< Type > ∗)

  sets the pointer that points to the next block
- void setBack (SequenceSet< Type > ∗)

  sets the pointer that points to the back of a block
- void setParent (SequenceSet< Type > ∗)

  sets the pointer that points to the parent of a block
- void writeDataAt (int, Type)

  writes an item at a given position
- void writeChildAt (int, SequenceSet< Type > ∗)

  writes a child at a given position
- void removeChildAt (int)

  removes a child at a given position
- void removeDataAt (int)

  removes an item at a given position
- void setSize (int)

  sets the size of the vector of items
- void setCsize (int)

  sets the size of the vector of children
- void setLeaf (bool)

  sets the leaf
- void setFilename (string)

sets the file name

- bool putData (Type)

  puts the item at the correct index in the vector of items
- bool putChild (SequenceSet< Type > ∗)

  puts the child at the correct index in the vector of children
- SequenceSet< Type > ∗ getNext ()

  gets the pointer to the next block
- SequenceSet< Type > ∗ getBack ()

  gets the pointer to the back of a block
- SequenceSet< Type > ∗ getParent ()

  gets the pointer to the parent of a block
- vector< SequenceSet< Type > ∗>::iterator getChildrenBegin ()

  gets the beginning of the vector of items
- vector< Type >::iterator getDataBegin ()

  gets the beginning of the vector of children
- SequenceSet< Type > ∗ getChildAt (int)

  gets the child at the specified position
- Type getDataAt (int)

  gets the item at the specified position
- int getSize ()

  gets the size of the vector of items
- int getCsize ()
- bool isLeaf ()

  verifies if node is a leaf
- bool isChildrenFull ()

  verifies if vector of children is full
- bool isDataFull ()

  verifies if vector of items is full
- string getFilename ()

  gets the file name

3.3.1 Constructor & Destructor Documentation

3.3.1.1 SequenceSet() [1/2]

```
template<class Type >
SequenceSet< Type >::SequenceSet ( )
```

Default constructor

```
 */
// default ORDER be 3;
SequenceSet(3);
```

3.3.1.2 SequenceSet() [2/2]

```
template<class Type >
SequenceSet< Type >::SequenceSet (
              int MAXSIZE )
```

Copy constructor

```
 */
next = NULL;
back = NULL;
this->MAXSIZE = MAXSIZE;
this->size = 0;
this->csize = 0;
leaf = false;
filename = "";
data.reserve(MAXSIZE + 1); // data capacity = size +1, for separation
data.resize(MAXSIZE + 1);
children.reserve(MAXSIZE + 2); // then children cap = size + 2
children.resize(MAXSIZE + 2);
```

3.3.2 Member Function Documentation

3.3.2.1 getBack()

```
template<class Type >
SequenceSet< Type > * SequenceSet< Type >::getBack ( )
```

gets the pointer to the back of a block

Precondition

the object using the function should be an object of SequenceSet class

Postcondition

the index to the back of a block is returned

```
 */
return back;
```

3.3.2.2 getChildAt()

```
template<class Type >
SequenceSet< Type > * SequenceSet< Type >::getChildAt (
              int pos )
```

gets the child at the specified position

Parameters

pos	is an int for the position at which retrieval is done

Precondition

the parameter pos should be of type int

Postcondition

the child is returned

```
 */
assert(pos < csize);
return children.at(pos);
```

3.3.2.3 getChildrenBegin()

```
template<class Type >
vector< SequenceSet< Type > *>::iterator SequenceSet< Type >::getChildrenBegin ( )
```

gets the beginning of the vector of items

Precondition

the vector using the function should be of type int or string

Postcondition

the index to the beginning of the vector of children is returned

```
 */
return children.begin();
```

3.3.2.4 getCsize()

```
template<class Type >
int SequenceSet< Type >::getCsize ( )
```

```
 */
return csize;
```

3.3.2.5 getDataAt()

```
template<class Type >
Type SequenceSet< Type >::getDataAt (
              int pos )
```

gets the item at the specified position

Parameters

pos	is an int for the position at which retrieval is done

Precondition

the parameter pos should be of type int

Postcondition

the item is returned

```
 */
assert(pos < size);
return data.at(pos);
```

3.3.2.6 getDataBegin()

```
template<class Type >
vector< Type >::iterator SequenceSet< Type >::getDataBegin ( )
```

gets the beginning of the vector of children

Precondition

the vector using the function should be of type int or string

Postcondition

the index to the beginning of the vector of items is returned

```
 */
return data.begin();
```

3.3.2.7 getFilename()

```
template<class Type >
string SequenceSet< Type >::getFilename ( )
```

gets the file name

Precondition

the file name to get should be a string

Postcondition

returns the file name

```
 */
return filename;
```

3.3.2.8 getNext()

```
template<class Type >
SequenceSet< Type > * SequenceSet< Type >::getNext ( )
```

gets the pointer to the next block

Precondition

the object using the function should be an object of SequenceSet class

Postcondition

the index to the next block is returned

```
 */
return next;
```

3.3.2.9 getParent()

```
template<class Type >
SequenceSet< Type > * SequenceSet< Type >::getParent ( )
```

gets the pointer to the parent of a block

Precondition

the object using the function should be an object of SequenceSet class

Postcondition

the index to the parent of a block is returned

```
 */
return parent;
```

3.3.2.10 getSize()

```
template<class Type >
int SequenceSet< Type >::getSize ( )
```

gets the size of the vector of items

gets the size of the vector of children

Precondition

the vector using the function should be of type int or string

Postcondition

the vector's size is returned

```
 */
return size;
```

3.3.2.11 isChildrenFull()

```
template<class Type >
bool SequenceSet< Type >::isChildrenFull ( )
```

verifies if vector of children is full

Precondition

the vector of children should be of type bool

Postcondition

returns whether the vector of children is full

```
 */
// can contain upto max + 1
return csize == MAXSIZE + 1;
```

3.3.2.12 isDataFull()

```
template<class Type >
bool SequenceSet< Type >::isDataFull ( )
```

verifies if vector of items is full

Precondition

the vector of items should be of type bool

Postcondition

returns whether the vector of items is full

```
 */
// can contain upto max
return size == MAXSIZE;
```

3.3.2.13 isLeaf()

```
template<class Type >
bool SequenceSet< Type >::isLeaf ( )
```

verifies if node is a leaf

Precondition

the variable using the function should be of type bool

Postcondition

returns whether the node is a leaf

```
 */
return leaf;
```

3.3.2.14 putChild()

```
template<class Type >
bool SequenceSet< Type >::putChild (
              SequenceSet< Type > * child )
```

puts the child at the correct index in the vector of children

Parameters

child	is an int or string

Precondition

the parameter child should be an int or string
the vector of children should not be full

Postcondition

the child is put at the correct index
the size of the vector of children is increased

```
 */
if (csize == MAXSIZE+2){
    return false;
}else{
    children.at(csize++) = child;

    sortChildren();
    return true;
}
```

3.3.2.15 putData()

```
template<class Type >
bool SequenceSet< Type >::putData (
               Type item )
```

puts the item at the correct index in the vector of items

Parameters

item	is an int or string

Precondition

the parameter item should be an int or string
the vector of items should not be full

Postcondition

the item is put at the correct index
the size of the vector of items is increased

```
 */
if (size == MAXSIZE + 1){
    return false;
}else{
    data.at(size++) = item;

    sortData();

    return true;
}
```

3.3.2.16 removeChildAt()

```
template<class Type >
void SequenceSet< Type >::removeChildAt (
              int pos )
```

removes a child at a given position

Parameters

pos	is an int for the position at which removal is done

Precondition

the parameter pos should be an int

Postcondition

the child is removed at the position specified
the size of the vector of children is decreased

```
 */
typename vector<SequenceSet<Type>* >::iterator it;
it = children.begin();
for (int i=0; i < pos; i++){
    ++it;
}
children.erase(it);
csize--;
```

3.3.2.17 removeDataAt()

```
template<class Type >
void SequenceSet< Type >::removeDataAt (
              int pos )
```

removes an item at a given position

Parameters

pos	is an int for the position at which removal is done

Precondition

the parameter pos should be an int

Postcondition

the item is removed at the position specified
the size of the vector of items is decreased

```
 */
data.at(pos) = data.at(--size);
sortData();
```

3.3.2.18 setBack()

```
template<class Type >
void SequenceSet< Type >::setBack (
             SequenceSet< Type > * back )
```

sets the pointer that points to the back of a block

Parameters

back	is a pointer for the back of a block

Precondition

the parameter should be a pointer of type int or string

Postcondition

the member variable block of class SequenceSet is set with the parameter's value

```
 */
this->back = back;
```

3.3.2.19 setCsize()

```
template<class Type >
void SequenceSet< Type >::setCsize (
             int csize )
```

sets the size of the vector of children

Parameters

size	is an int for the size of the vector of children

Precondition

the parameter size should be an int

Postcondition

the member variable csize of the SequenceSet class is set with the parameter's value

```
 */
this->csize = csize;
```

3.3.2.20 setFilename()

```
template<class Type >
void SequenceSet< Type >::setFilename (
             string filename )
```

sets the file name

Parameters

filename	is of type string for the file name

Precondition

the parameter filename should be a string

Postcondition

the member variable filename of the SequenceSet class is set with the parameter's value

```
 */
this->filename = filename;
```

3.3.2.21 setLeaf()

```
template<class Type >
void SequenceSet< Type >::setLeaf (
              bool leaf )
```

sets the leaf

Parameters

leaf	is of type bool for the leaf

Precondition

the parameter leaf should be a bool

Postcondition

the member variable leaf of the SequenceSet class is set with the parameter's value

```
 */
this->leaf = leaf;
```

3.3.2.22 setNext()

```
template<class Type >
void SequenceSet< Type >::setNext (
              SequenceSet< Type > * next )
```

sets the pointer that points to the next block

Parameters

next	is a pointer for the next block

Precondition

the parameter should be a pointer of type int or string

Postcondition

the member variable next of class SequenceSet is set with the parameter's value

```
 */
this->next = next;
```

3.3.2.23 setParent()

```
template<class Type >
void SequenceSet< Type >::setParent (
             SequenceSet< Type > * parent )
```

sets the pointer that points to the parent of a block

Parameters

parent	is a pointer for the parent of a block

Precondition

the parameter should be a pointer of type int or string

Postcondition

the member variable parent of class SequenceSet is set with the parameter's value

```
 */
this->parent = parent;
```

3.3.2.24 setSize()

```
template<class Type >
void SequenceSet< Type >::setSize (
             int size )
```

sets the size of the vector of items

Parameters

size	is an int for the size of the vector of items

Precondition

the parameter size should be an int

Postcondition

the member variable size of the SequenceSet class is set with the parameter's value

```
 */
this->size = size;
```

3.3.2.25 writeChildAt()

```
template<class Type >
void SequenceSet< Type >::writeChildAt (
             int pos,
             SequenceSet< Type > * child )
```

writes a child at a given position

Parameters

pos	is an int for the position at which writing is done
child	is an int or string for the child to be written

Precondition

the parameter pos should be an int
the parameter child should be either an int or string

Postcondition

the child is written at the position specified
the size of the vector of children is increased

```
 */
children.at(pos) = child;
csize++;
```

3.3.2.26 writeDataAt()

```
template<class Type >
void SequenceSet< Type >::writeDataAt (
             int pos,
             Type item )
```

writes an item at a given position

Parameters

pos	is an int for the position at which writing is done
item	is an int or string for the item to be written

Precondition

the parameter pos should be an int
the parameter item should be either an int or string

Postcondition

the item is written at the position specified
the size of the vector of items is increased

```
 */
assert(size <= MAXSIZE); // can contain upto MAXSIZE+1, but need separation

data.at(pos) = item;
size++;
```

The documentation for this class was generated from the following file:

- SequenceSet.h

3.4 Tournament< Type > Class Template Reference

Public Member Functions

- Tournament ()
- Tournament (vector< vector< Type > >)
- void init (vector< vector< Type > >)

  initialize a vector of items
- void sort ()

  sorts each list
- vector< Type > retrieve ()

  returns list of sorted items
- void logFile (ofstream &)

  generates logfile

3.4.1 Constructor & Destructor Documentation

3.4.1.1 Tournament() [1/2]

```
template<class Type >
Tournament< Type >::Tournament ( )
```

Default constructor

```
 */
    // Use with init()
```

3.4.1.2 Tournament() [2/2]

```
template<class Type >
Tournament< Type >::Tournament (
             vector< vector< Type > > items )
```

Copy constructor

```
 */
    init(items);
```

3.4.2 Member Function Documentation

3.4.2.1 init()

```
template<class Type >
void Tournament< Type >::init (
             vector< vector< Type > > items )
```

initialize a vector of items

Parameters

items	is a vector of vectors to hold items

Precondition

the elements of the vector items should be vectors

Postcondition

a vector of vectors containing items is returned

```
 */
    HEAPSIZE = (int)items.size();
    master_data = items;
    size = 0;
    data.reserve(HEAPSIZE);
    for (int i=0; i < HEAPSIZE; i++){
        its.push_back(master_data[i].begin());
    }
    for (int i=0; i < HEAPSIZE; i++){
        push(i);
    }
```

3.4.2.2 logFile()

```
template<class Type >
void Tournament< Type >::logFile (
             ofstream & outFile2 )
```

generates logfile

Parameters

logfile	is of type ofstream for a pointer

Precondition

the argument should be a pointer

Postcondition

the logfile is returned

```
*/
    outFile2 << "The number of records that can fit in memory are "  << HEAPSIZE << endl;

    int counter = 0;

    for (typename vector<vector<Type> >::iterator outit = master_data.begin(); outit != master_data.end();
      ++outit)
    {
        for (typename vector<Type>::iterator init = (*outit).begin(); init != (*outit).end(); ++
      init)
        {
            counter++; //keeps track of how many elements are in the vector of vectors
            //i.e. the number of records
        }
    }

    outFile2 << "The number of records are " << counter << endl;

    outFile2 << "The number of runs are " << HEAPSIZE << endl;

    int max = (int)master_data[0].size();
    int min = (int)master_data[0].size();

    for(int i = 0; i < HEAPSIZE; i++)
    {

        if(master_data[i].size() > max)
        {
            max = (int)master_data[i].size();
        }

        if(master_data[i+1].size() < min)
        {
            min = (int)master_data[i+1].size();
        }

    }


    outFile2 << "The smallest number of records in all of the runs is " << min << endl;
    outFile2 << "The largest number of records in all of the runs is " << max << endl;
    outFile2 << "The arithmetic mean number of records in all of the runs is " << counter / HEAPSIZE <<
      endl;

    int height = ceil(log2(counter)); //ceil(log2(number of records))
    outFile2 << "The height of the tournament tree for the merge is " << height << endl;
```

3.4.2.3 retrieve()

```
template<class Type >
vector< Type > Tournament< Type >::retrieve ( )
```

returns list of sorted items

Precondition

there should some items in the list

Postcondition

the sorted list is returned

```
*/
    return run;
```

3.4.2.4 sort()

```
template<class Type >
void Tournament< Type >::sort ( )
```

sorts each list

Precondition

the list to be sorted should not be empty

Postcondition

each list is sorted

```
 */
    while (size != 0){
        run.push_back(pop());
    }
```

The documentation for this class was generated from the following file:

- Tournament.h

Chapter 4

File Documentation

4.1 BPT.h File Reference

Function prototypes and implementation for the BPT class.

```
#include <iostream>
#include <vector>
#include <limits>
#include <assert.h>
#include <math.h>
#include "SequenceSet.h"
```

Include dependency graph for BPT.h:

4.2 DualHeap.h File Reference

Function prototypes and implementations for the DualHeap class This contains the header file and implementation for methods of DualHeap class. The following methods are implemented in this file: constructors, mutators, accessors, and helper functions.

```
#include <iostream>
#include <vector>
#include <limits>
#include <assert.h>
```

Include dependency graph for DualHeap.h:

Classes

- class DualHeap< Type >

4.2.1 Detailed Description

Function prototypes and implementations for the DualHeap class This contains the header file and implementation for methods of DualHeap class. The following methods are implemented in this file: constructors, mutators, accessors, and helper functions.

4.3 SequenceSet.h File Reference

Function prototypes and implementation for the SequeneceSet class.

```
#include <iostream>
#include <vector>
#include <limits>
#include <assert.h>
#include <math.h>
#include <algorithm>
```

Include dependency graph for SequenceSet.h: This graph shows which files directly or indirectly include this file:

Classes

- class SequenceSet< Type >

4.3.1 Detailed Description

Function prototypes and implementation for the SequeneceSet class.

This contains the header file for SequeneceSet class. It also contains constructors, mutators, accessors, and helper functions.

4.4 Tournament.h File Reference

Function prototypes and implementations for the Tournament class.

```
#include <iostream>
#include <vector>
#include <limits>
#include <assert.h>
#include <math.h>
```

Include dependency graph for Tournament.h:

Classes

- class Tournament< Type >

4.4.1 Detailed Description

Function prototypes and implementations for the Tournament class.

This contains the header file and implementation for methods of Tournament class. The following methods are implemented in this file: constructors, mutators, accessors, and helper functions.

Author

Team 7

Index