Project of COP 5536 Fall 2019

Dinh Nguyen (UFID: 9482-7764)

truc.nguyen@ufl.edu

Usage

Dependencies

- GCC >= 7.4.0

Compile

```
$ make
```

Execute

```
$ ./risingCity </path/to/input/file>
```

- `</path/to/input/file>`: path to the input text file

Overall design

Implementation methods

This project is implemented as a **discrete-event simulator**. There are two kinds of event:

1. Read the next command from the input file
2. Choose the building to work on

Each event is associated with a timestamp. At any point in time, the simulator will choose the event that has the smallest timestamp and update its global timestamp with that of the chosen event. The event loop of the simulator is implemented using the following pseudo-code:

```
global_timestamp = 0;
while (there is some event)
{
    if (the timestamp of event 1 <= the timestamp of event 2)
    {
        // This is the event for reading the next command

        Update the executed_time of the current building

        global_timestamp = timestamp of event 1; // update the global time

        Read and execute the current command

        Update the timestamp of event 1 with the timestamp of the next command in the input file
    }
    else
    {
```

```
        // This is the event for choosing the next building

        Update the executed_time of the current building

        global_timestamp = timestamp of event 2; // update the global time

        if (the current building is finished)
            remove the building
        else
            Reinsert it to the heap

        Choose the next building to work on based on the min heap
        The timestamp of event 2 = global_timestamp + min(totalTime - executedTime, 5)
    }
}
```

As recommended by the TA, when choosing the building to work on, it will perform the `ExtractMin` operation on the min heap (i.e., remove the root of the heap and return that root). Note that it won't do anything to the red-black tree. After the 5-day period, the building will be inserted back to the min heap.

When the program encounters an `Insert` command, it will perform insertion on both the min heap and the red black tree.

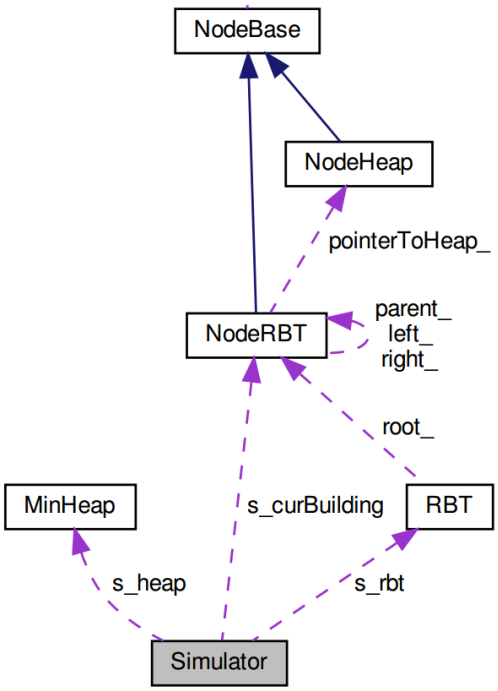
For example:

```
0: Insert(50, 100)
2: Insert(51, 100)
3: Insert(52, 100)
4: Insert(53, 100)
```

At 0, the building 50 is inserted to the data structures, then chosen and extracted from the min heap. At 2, 3, and 4, the building 51, 52, and 53 is inserted to the data structure,respectively. The 50 will be inserted back to the heap at 5, and a new building will be chosen to build.

Architectural overview

The architectural overview of the implementation is illustrated via the following class diagram



There are 6 main classes in this program:

- `Simulator` : this class implements the discrete-event simulator with the event loop. This is a static class and represents the main workflow of the program. The building that is currently being worked on is denoted as `s_curBuilding` and it is of type `NodeRBT`
- `NodeBase` : this is the base class (pure virtual) for representing a node/building. This class stores the building record `(buildingNums, executedTime, totalTime)` and implements some basic and virtual operations.
- `NodeHeap` : this class is derived from `NodeBase` and represents a node in the min heap. It implements some of the necessary methods for the heap structure.
- `NodeRBT` : this class is derived from `NodeBase` and represents a node in the red-black tree. It stores some additional attributes like color, parent, left child, and right child. It also stores a pointer to the corresponding node. Some of the necessary methods for the red-black tree are also implemented.
- `MinHeap` : this class implements the min heap structure using `NodeHeap` as its node. The heap is implemented as array-based.
- `RBT` : this class implements the red-black tree using `NodeRBT` as its node. All the operations are implemented in the same manner as the class lectures.

Detailed design

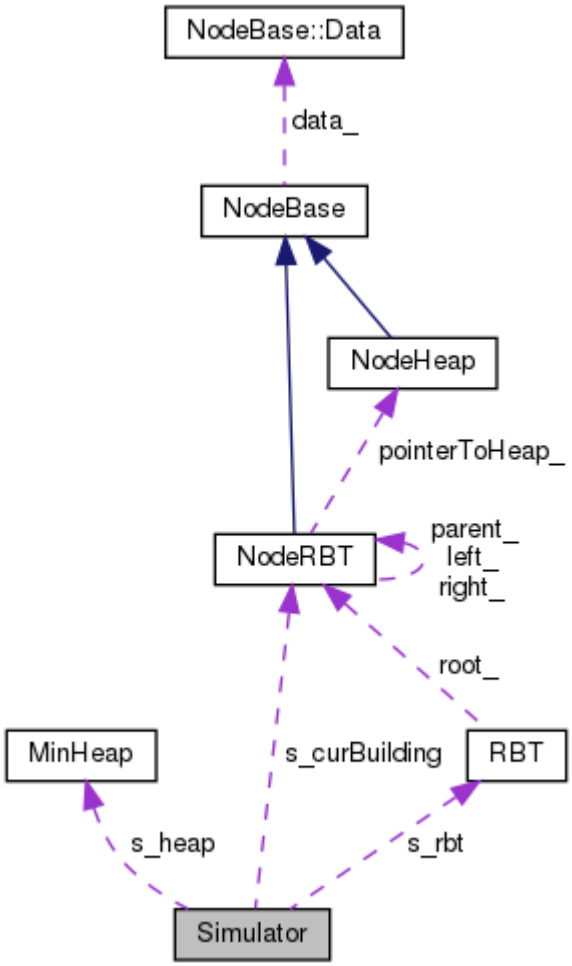
For detailed implementation of each class/function, see the following pages

Simulator Class Reference

A static class for the discrete-event simulator. More...

`#include <Simulator.h>`

Collaboration diagram for Simulator:



[legend]

Static Public Member Functions

static void	**initialize** (const std::string &inputFileName)
static void	**close** ()
static void	**loop** ()

Private Types

enum	**CommandType** { **INSERT**, **PRINT1**, **PRINT2** } Define types of events.

Static Private Member Functions

static void	**parseCommand** (const std::string &cmdStr)
static void	**printBuilding** (uint num1)
static void	**printBuilding** (uint num1, uint num2)
static long	**readCommand** ()
static void	**executePendingCommand** ()
static long	**chooseNextBuilding** ()
static void	**removeCurBuilding** ()

Static Private Attributes

struct {	
uint **arrivalTime**	
	Arrival time of the command.
CommandType **cmdType**	
	Type of command.
std::string **data**	
	Data field.
}	**s_pendingCommand**
	The pending command.
static long	**s_timestamp** = 0
	The global time counter.
static long	**s_cmdTime** = -1
	Next timestamp to read the command.
static long	**s_buildingTime** = -1
	Next timestamp to choose a new building.
static **RBT** *	**s_rbt**
	Pointer to a red-black tree.
static **MinHeap** *	**s_heap**
	Pointer to a min heap.
static **NodeRBT** *	**s_curBuilding** = nullptr
	The current selected building.
static std::ifstream *	**s_inFile** = nullptr
	Stream of the input file.
static std::ofstream *	**s_outFile** = nullptr
	Stream of the output file.

Detailed Description

A static class for the discrete-event simulator.

Member Function Documentation

◆ chooseNextBuilding()

long Simulator::chooseNextBuilding ( ) static private

Choose a new building to work on based on s_heap

Returns

-1 if there is no building to work on

the next timestamp to choose the next building

◆ close()

void Simulator::close ( ) static

Close the **Simulator**

◆ executePendingCommand()

void Simulator::executePendingCommand ( ) static private

Execute the command in s_pendingCommand

◆ initialize()

void Simulator::initialize (const std::string & inputFileName) static

Initialize the **Simulator**

Parameters

inputFileName path to input file

◆ loop()

void Simulator::loop ( ) static

Implement event loop for the simulator

There are two kinds of event in this loop:

1. Read the next command, represented by s_cmdTime
2. Choose the next building, represented by s_buildingTime If there is no event, the corresponding variables are set to -1. At each timestamp, the simulator chooses to work on the event with the smallest timestamp

◆ parseCommand()

void Simulator::parseCommand (const std::string & cmdStr) static private

Parse command string and save to s_pendingCommand

Parameters

cmdStr the command string

◆ printBuilding() [1/2]

void Simulator::printBuilding (uint num1) static private

Print the triplet buildingNum,executed_time,total_time

Parameters

num1 the buildingNum

◆ printBuilding() [2/2]

void Simulator::printBuilding (uint num1,
uint num2
)

static private

Print all triplets bn, executed_time, total_time for which buildingNum1 <= bn <= buildingNum2

Parameters

num1 buildingNum1

num2 buildingNum2

◆ readCommand()

long Simulator::readCommand ( )

static private

Read a new command from the input file

Returns

arrival time of the command

◆ removeCurBuilding()

void Simulator::removeCurBuilding ( )

static private

Remove the current building from s_heap, s_rbt, and set the pointer to null

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

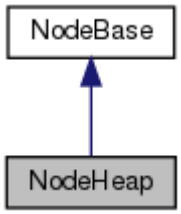
- **Simulator.h**
- Simulator.cpp

NodeHeap Class Reference

Node for min heap. More...

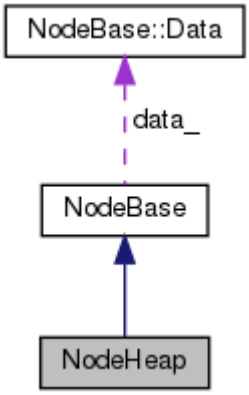
`#include <NodeBase.h>`

Inheritance diagram for NodeHeap:



[legend]

Collaboration diagram for NodeHeap:



[legend]

Public Member Functions

	NodeHeap (uint buildingNums, ulong totalTime)
ulong	**getKey** () const
bool	**operator<** (const **NodeHeap** &rhs) const
bool	**operator>** (const **NodeHeap** &rhs) const

▸ **Public Member Functions inherited from NodeBase**

Private Attributes

int	**heapPos_** Current index of this node in the heap.
friend	**MinHeap**

Additional Inherited Members

▸ **Protected Attributes inherited from NodeBase**

Detailed Description

Node for min heap.

Constructor & Destructor Documentation

◆ NodeHeap()

```
NodeHeap::NodeHeap ( uint  buildingNums,
                     ulong  totalTime
                   )
```

Constructor for **NodeHeap**

Parameters

buildingNums	building number
totalTime	total time

Member Function Documentation

◆ getKey()

```
ulong NodeHeap::getKey ( ) const
```

virtual

Get the key value of the node

Returns

executed_time

Implements **NodeBase**.

◆ operator<()

bool NodeHeap::operator< (const **NodeHeap** & rhs) const

Overload the < operator to compare with another **NodeHeap**

Parameters

rhs the other **NodeHeap**

Returns

true if <, false otherwise

◆ operator>()

bool NodeHeap::operator> (const **NodeHeap** & rhs) const

Overload the > operator to compare with another **NodeHeap**

Parameters

rhs the other **NodeHeap**

Returns

true if >, false otherwise

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

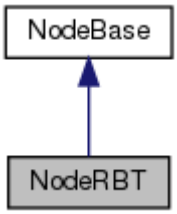
- **NodeBase.h**
- NodeBase.cpp

NodeRBT Class Reference

Node for red-black tree. More...

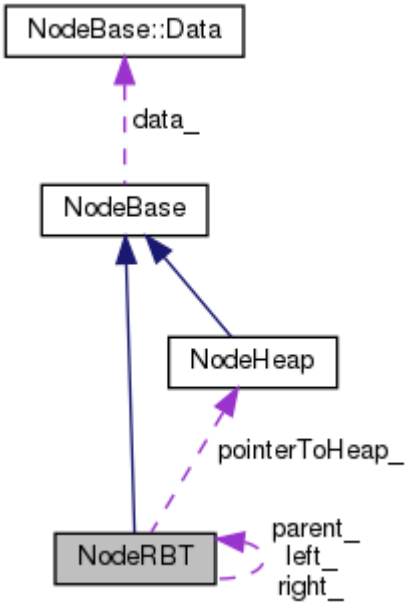
`#include <NodeBase.h>`

Inheritance diagram for NodeRBT:



[legend]

Collaboration diagram for NodeRBT:



[legend]

Public Member Functions

	NodeRBT (uint buildingNums, ulong totalTime, **NodeHeap** *pointerToHeap)
void	**addExecutedTime** (ulong addTime)
void	**swapData** (**NodeBase** *p)
ulong	**getKey** () const
NodeHeap *	**getNodeHeap** () const
bool	**operator<** (const **NodeRBT** &rhs)
bool	**operator>** (const **NodeRBT** &rhs)
bool	**operator<** (const uint &key)
bool	**operator==** (const uint &key)

▸ **Public Member Functions inherited from NodeBase**

Private Member Functions

NodeRBT *	**getUncleNode** ()
uint8_t	**cntRedChild** ()

Private Attributes

uint8_t	**color_** Color of the node.
NodeRBT *	**left_** = nullptr Pointers to related nodes.
NodeRBT *	**right_** = nullptr
NodeRBT *	**parent_** = nullptr
NodeHeap *	**pointerToHeap_** Pointer to the corresponding node in heap.
friend	**RBT**

Additional Inherited Members

▸ **Protected Attributes inherited from NodeBase**

Detailed Description

Node for red-black tree.

Constructor & Destructor Documentation

◆ NodeRBT()

NodeRBT::NodeRBT (uint buildingNums,
ulong totalTime,
NodeHeap * pointerToHeap
)

Constructor for NodeRBT

Parameters

buildingNums	building number
totalTime	total time
pointerToHeap	pointer to the corresponding node in heap

Member Function Documentation

◆ addExecutedTime()

void NodeRBT::addExecutedTime (ulong addTime) virtual

Add time to the executed_time field, update the correspond node in heap

Parameters

addTime	value to add

Reimplemented from NodeBase.

◆ cntRedChild()

uint8_t NodeRBT::cntRedChild ( ) private

Count the number of red children

Returns

the number of red children

◆ getKey()

ulong NodeRBT::getKey ( ) const virtual

Get the key value of the node

Returns

buildingNums

Implements **NodeBase**.

◆ getNodeHeap()

NodeHeap * NodeRBT::getNodeHeap ( ) const

Get the corresponding node in heap

Returns

the **NodeHeap**

◆ getUncleNode()

NodeRBT * NodeRBT::getUncleNode ( ) private

Get uncle node. Make sure it has the a grand parent

Returns

the uncle node

◆ operator<() [1/2]

bool NodeRBT::operator< (const **NodeRBT** & rhs)

Overload the < operator to compare with another **NodeRBT**

Parameters

rhs the other **NodeHeap**

Returns

true if <, false otherwise

◆ operator<() [2/2]

bool NodeRBT::operator< (const uint & key)

Overload the < operator to compare with another key

Parameters

rhs the other key

Returns

true if <, false otherwise

◆ operator==()

bool NodeRBT::operator== (const uint & key)

Overload the == operator to compare with another key

Parameters

rhs the other key

Returns

true if ==, false otherwise

◆ operator>()

bool NodeRBT::operator> (const **NodeRBT** & rhs)

Overload the > operator to compare with another **NodeRBT**

Parameters

rhs the other **NodeHeap**

Returns

true if >, false otherwise

◆ swapData()

void NodeRBT::swapData (**NodeBase** * p) virtual

Swap the data field and the pointerToHeap with another node

Parameters

p another node

Reimplemented from **NodeBase**.

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

- **NodeBase.h**
- NodeBase.cpp

MinHeap Class Reference

A min heap data structure. More...

`#include <MinHeap.h>`

Public Member Functions

	MinHeap ()
void	**insertNode** (**NodeHeap** *p)
NodeHeap *	**extractMin** ()
NodeHeap *	**peekMin** ()
void	**increaseKey** (**NodeHeap** *p)
bool	**isEmpty** ()
void	**remove** (**NodeHeap** *p)

Static Public Member Functions

static void	**unitTest** ()

Private Member Functions

void	**heapify** (uint idx)
void	**swap** (**NodeHeap** *&a, **NodeHeap** *&b)
uint	**getParent** (const uint &idx)
uint	**getLeft** (const uint &idx)
uint	**getRight** (const uint &idx)

Private Attributes

std::vector< **NodeHeap** * >	**heap_** Array to store the node in heap.

Detailed Description

A min heap data structure.

Constructor & Destructor Documentation

◆ MinHeap()

MinHeap::MinHeap ( )

Constructor for **MinHeap**

Member Function Documentation

◆ extractMin()

NodeHeap * MinHeap::extractMin ( )

Extract the min/root of the heap, the root is then removed from the structure. Note that it doesn't delete the object pointed by the root

Returns

root of the heap

◆ getLeft()

uint MinHeap::getLeft (const uint & idx) private

Get the index of the left child

Parameters

idx index of the current node

Returns

index of the left child

◆ getParent()

uint MinHeap::getParent (const uint & idx) private

Get the index of the parent node

Parameters

idx index of the current node

Returns

index of the parent node

◆ getRight()

uint MinHeap::getRight (const uint & idx) private

Get the index of the right child

Parameters

idx index of the current node

Returns

index of the right child

◆ heapify()

void MinHeap::heapify (uint idx) private

Perform heapify on the heap starting from an index

Parameters

idx index

◆ increaseKey()

void MinHeap::increaseKey (**NodeHeap** * p)

Update the structure to increase the key of a node in the heap

Parameters

p the node with the **increased** key

◆ insertNode()

void MinHeap::insertNode (**NodeHeap** * p)

Insert a node of type **NodeHeap** to the heap

Parameters

p node to insert

◆ isEmpty()

bool MinHeap::isEmpty ( )

Check if the heap is empty

Returns

true if empty, false otherwise

◆ peekMin()

NodeHeap * MinHeap::peekMin ( )

Get the min/root of the heap, the structure is unchanged

Returns

root of the heap

◆ remove()

void MinHeap::remove (**NodeHeap** * p)

Remove a node from the heap

Parameters

p node to remove

◆ swap()

void MinHeap::swap (**NodeHeap** *& a,
NodeHeap *& b
) private

Swap two nodes in the heap

Parameters

a first node

b second node

◆ unitTest()

void MinHeap::unitTest ( ) static

Run some unit tests

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

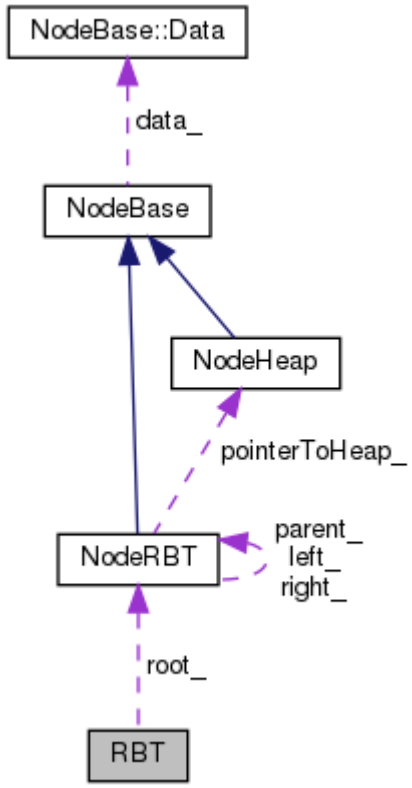
- **MinHeap.h**
- MinHeap.cpp

RBT Class Reference

A red-black tree data structure. More...

`#include <RBT.h>`

Collaboration diagram for RBT:



[legend]

Public Member Functions

	RBT ()
void	**insertNode** (**NodeRBT** *p)
bool	**deleteNode** (uint key)
void	**deleteNode** (**NodeRBT** *y)
NodeRBT *	**searchNode** (uint key)
NodeRBT *	**getRoot** () const

Static Public Member Functions

static void	**printRange** (**NodeRBT** *root, const uint &left, const uint &right, std::ostream &out, bool &comma)
static void	**unitTest** ()

Private Member Functions

void	**insertNodeBST** (**NodeRBT** *p)

NodeRBT *	**getReplaceNodeForDeletion** (NodeRBT *p)
void	**deleteBlackLeaf** (NodeRBT *y)
void	**deleteLeafNode** (NodeRBT *p)
void	**rotateRR** (NodeRBT *y)
void	**rotateLL** (NodeRBT *y)
void	**rotateRL** (NodeRBT *p)
void	**rotateLR** (NodeRBT *p)

Private Attributes

NodeRBT *	**root_** Root of the tree.

Detailed Description

A red-black tree data structure.

Constructor & Destructor Documentation

◆ RBT()

RBT::RBT ( ) inline

Default constructor of **RBT**

Member Function Documentation

◆ deleteBlackLeaf()

void RBT::deleteBlackLeaf (NodeRBT * y) private

Procedure to delete a black leaf

Parameters

y the black leaf

◆ deleteLeafNode()

void RBT::deleteLeafNode (**NodeRBT** * p) private

Delete a leaf node in the tree, regardless of color

Parameters

p the leaf node

◆ deleteNode() [1/2]

bool RBT::deleteNode (uint key)

Delete a node from this tree by its key

Parameters

key the key to delete

Returns

true if the key is found, false otherwise

◆ deleteNode() [2/2]

void RBT::deleteNode (**NodeRBT** * y)

Delete a node from this tree

Parameters

y the **NodeRBT** to delete

◆ getReplaceNodeForDeletion()

NodeRBT * RBT::getReplaceNodeForDeletion (**NodeRBT** * p) private

Find a node from the tree that is used to swap with the *node to delete* using the Binary Search Tree algorithm, then swap the key and return the node that is either leaf or has a single child

Parameters

p node to delete

Returns

the replace node

◆ getRoot()

NodeRBT * RBT::getRoot ( ) const

Getter for the root of the tree

Returns

◆ insertNode()

void RBT::insertNode (**NodeRBT** * p)

Insert a node to this tree

Parameters

p a **NodeRBT** to insert

◆ insertNodeBST()

void RBT::insertNodeBST (**NodeRBT** * p) private

Insert node to the tree using the Binary Search Tree algorithm

Parameters

p node to insert

◆ printRange()

void RBT::printRange (**NodeRBT** * root,
const uint & left,
const uint & right,
std::ostream & out,
bool & comma
) static

Print the list of nodes in which its key lie within a specific range [left, right]

Parameters

root the root of a **RBT**

left the left value

right the right value

out the output stream

comma must be false

◆ rotateLL()

void RBT::rotateLL (**NodeRBT** * y) private

LL rotation

Parameters

y the start node

◆ rotateLR()

void RBT::rotateLR (**NodeRBT** * p) private

LR rotation

Parameters

p the start node

◆ rotateRL()

void RBT::rotateRL (**NodeRBT** * p) private

RL rotation

Parameters

p the start node

◆ rotateRR()

void RBT::rotateRR (**NodeRBT** * y) private

RR rotation

Parameters

y the start node

◆ searchNode()

NodeRBT * RBT::searchNode (uint key)

Search for the node using its key

Parameters

key the key to search

Returns

null if not found, otherwise return the node

◆ unitTest()

void RBT::unitTest ( ) static

Some unit tests for **RBT**

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

- **RBT.h**
- RBT.cpp