Quadtree Implementation Project

**sc17sdc**

**University of Leeds**

**Computer Science**

**COMP1921**

**3/1/2018**

Slavyana Dianova Chervenkondeva

Progress and personal reflection

The aim of this report is to show my progress on the first Programming Project and to demonstrate my understanding of the code I wrote. I am going to write about each of the tasks individually and attach screenshots where necessary. In the end I am also going to reflect on those aspects of the Project that I found most difficult and the things I could have done better.

Table of Contents

[Task 1: Destroy the tree 1](#_Toc507546321)

[Tests 1](#_Toc507546322)

[Code 4](#_Toc507546323)

[Task 2: Growing the Quadtree 5](#_Toc507546324)

[Tests 5](#_Toc507546325)

[Code: 6](#_Toc507546326)

[Task 3: A limit on tree level 7](#_Toc507546327)

[Tests 8](#_Toc507546328)

[Task 4: Generating a data-dependent Quadtree 9](#_Toc507546329)

[Tests 9](#_Toc507546330)

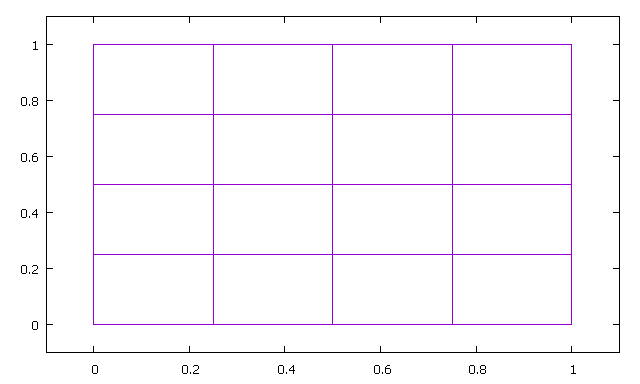
### Task 1: Destroy the tree

It is of prime significance to free all the memory which has been allocated. To ensure that there are no memory leaks the tree has to be destroyed (recursively).

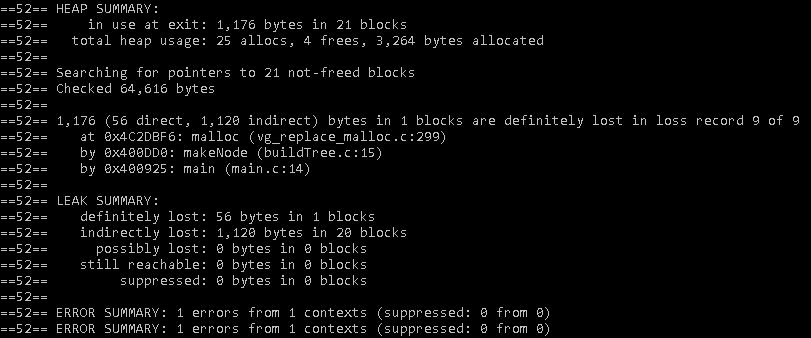
### Tests

Running your algorithm with:

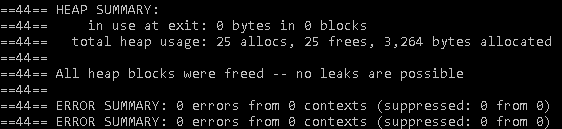
1. **A full tree at Level 2**  
   Expected input:   
   1  
   1  
     
   Expected results: Creation of a full tree of level 2, memory leaks when the memory isn’t freed, no memory leaks after destroying the tree.  
     
    Final results:  
   - A full level 2 tree.



* Valgrind before destroying the tree.



* Valgrind after destroying the tree

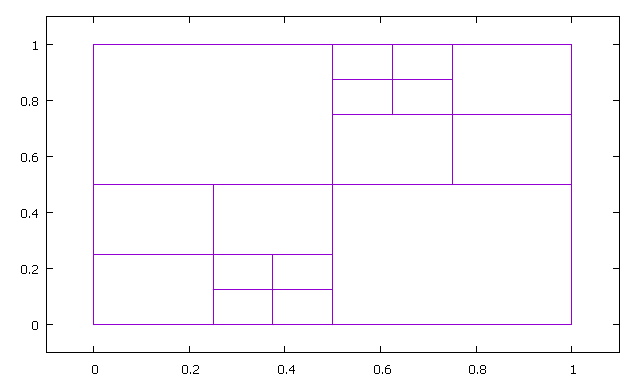


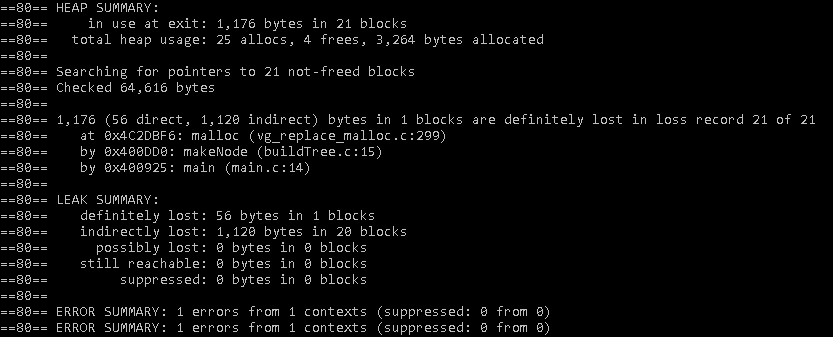
1. **A non-uniform Level 3 tree that you define.**Expected input:

1

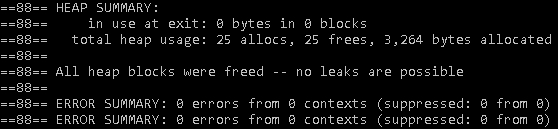
2

Expected results: Creation of a non-uniform tree of Level 3, memory leaks when the memory  
isn’t freed, no memory leaks after destroying the tree.  
  
Final results:   
  
- A level 3 tree



- Valgrind before destroying the tree

* Valgrind after destroying the tree:



### Code

Checks if the node is a leaf node and frees the allocated memory if it is. Otherwise, the recursive function goes through all 4 children of the node until a leaf is reached. After freeing the children, the memory allocated to the parent is also freed.

**void** destroyTree(Node \*node) {  
 **int** i;  
  
 **if**( node->child[0] == NULL )  
 free(node);

**else** {  
 **for** ( i=0; i<4; ++i ) {  
 destroyTree( node->child[i] );  
 }  
 free(node);  
 }  
}

### Task 2: Growing the Quadtree

This function recursively grows the tree uniformly overall by one level.

### Tests

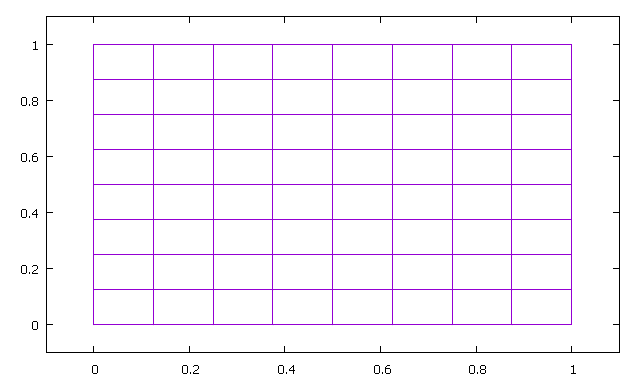
1. Grow a tree from an initial tree structure of a full Level 2 tree.  
   Expected input:

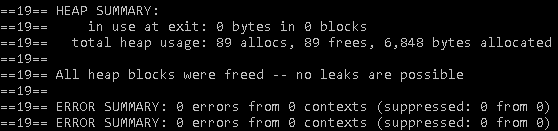
2

1

Expected results: Creation of a full Level 3 tree

Final results:





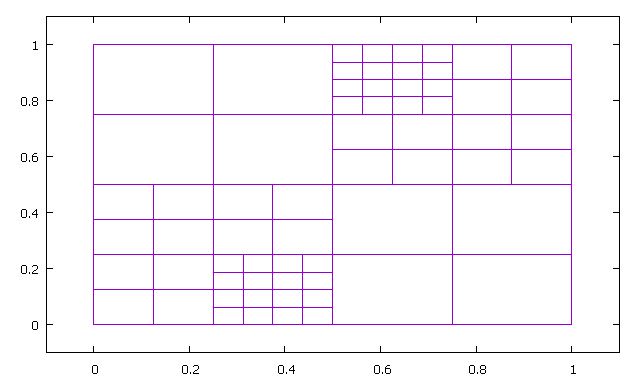
1. Grow a tree from a non-uniform Level 3 tree structure that is not full.   
   Expected input:

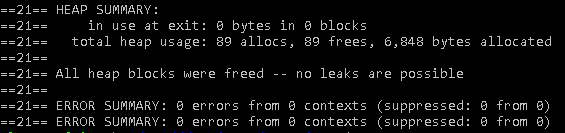
2

2

Expected results: Creating a non-uniform level 4 tree from the level 3 tree in task 1

Final results:





### Code:

This function recursively makes children for all the leaf nodes. If the node is a parent, the function goes through all its children and adds children when a leaf node is reached.

**void** growTree( Node \*node ) {  
  
 **if**( node->child[0] == NULL) {  
 makeChildren(node);  
 }  
  
 **else** {

**int** i;  
 **for** ( i=0; i<4; ++i ) {  
 growTree (node->child[i]);  
 }  
 }

### Task 3: A limit on tree level

*(i) Making some basic assumptions about the size of primitive data types estimate the memory use for a tree Node on paper.*

*Explain your answer and assumptions.*

The Node consists of 1 int, an array of 2 doubles and pointers to 4 children (also nodes). Judging by the size of primitive datatypes, 1 node should be 4 bytes (int) + 2\*8 bytes (doubles) + 4\*8 bytes (pointers) it should be 52 bytes.

*(ii) Produce a table showing how much memory you would expect to use for a full tree with 5, 6, 7, 8, 9 and 10 levels.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| levels | 5 | 6 | 7 | 8 | 9 | 10 |
| size (bytes) | 70,980 | 283,972 | 1,135,940 | 4,543,812 | 18,175,300 | 72,701,252 |

*(iii) Using your quadtree implementation and running the code with valgrind produce data on the actual memory use of your code for a full tree with 5,6,7,8 levels.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| levels | 5 | 6 | 7 | 8 |
| size (bytes) | 76,440 | 305,816 | 1,223,320 | 4,893,336 |

It turns out that 1 Node is actually 56 bytes which is why there is a difference between the expected and the actual size of the quadtrees. The compiler adds 4 bytes of padding because of the data alignment requirements of the system. (https://stackoverflow.com) In other words, the size has to be a multiple of 64 bits or 8 bytes. (52 isn’t, but 56 is).

*(iv) If you would like to limit the overall memory use of the application to 20Mb what maximum level should you choose?*

20 Megabits – level 7 since it’s 9Mb and 8 is over 37Mb

20 Megabytes – level 9 since it’s 18.6MB and 10 is over 72MB

### Tests

*Implement the maximum tree level as a parameter in your code so that the tree cannot grow beyond that level. Your code should continue execution but the tree is prevented from growing beyond the maximum level.*

I added the maximum tree level as a parameter in the growTree() function and the makeChildren() function.

**void** makeChildren( Node \*parent, **int** maxLvl );  
**void** growTree( Node \*node, **int** maxLvl );

For the tests in tasks 1, 2, 3 and 4 it is set to a value that will not influence the end result. However, I have added an additional test in task 3 which allows the user to set a limit on the tree level and grow a full level n tree.

Use your growTree() function with:

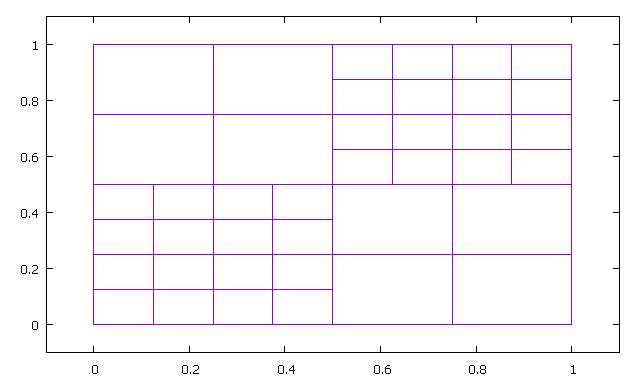
1. maximum level=3  
   Expected input:

3

2

Expected results: Creating children for all level 1 and 2 nodes.

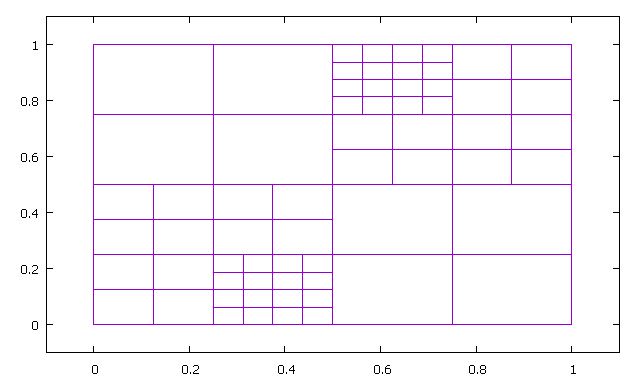
Final results:



1. maximum level=4

Expected results: Creating children for all level 1, 2 and 3 nodes.

Final results:



### Code

Creates children only if the level limit isn’t reached.

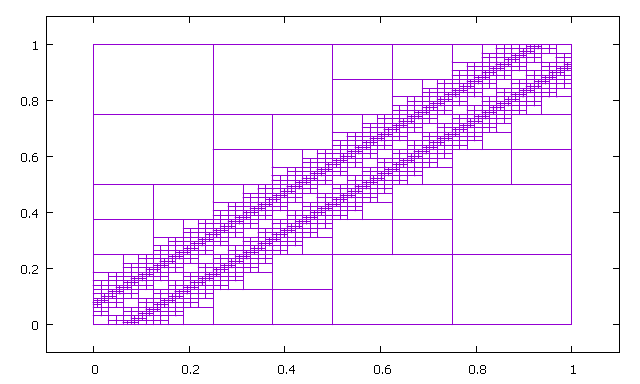
// split a leaf nodes into 4 children  
  
**void** makeChildren( Node \*parent, **int** maxLvl ) {  
 **if**(parent->level >= maxLvl) **return**;  
  
 **double** x = parent->xy[0];  
 **double** y = parent->xy[1];  
  
 **int** level = parent->level;  
  
 **double** hChild = pow(2.0,-(level+1));  
  
 parent->child[0] = makeNode( x,y, level+1 );  
 parent->child[1] = makeNode( x+hChild,y, level+1 );  
 parent->child[2] = makeNode( x+hChild,y+hChild, level+1 );  
 parent->child[3] = makeNode( x,y+hChild, level+1 );  
}  
  
// Grows the tree by one lvl  
**void** growTree( Node \*node, **int** maxLvl ) {  
  
  
 **if**( node->child[0] == NULL) {  
  
 makeChildren(node, maxLvl);  
  
 }  
  
 **else** {  
 **int** i;  
 **for** ( i=0; i<4; ++i ) {  
 growTree (node->child[i], maxLvl);  
 }  
 }  
}

### Task 4: Generating a data-dependent Quadtree

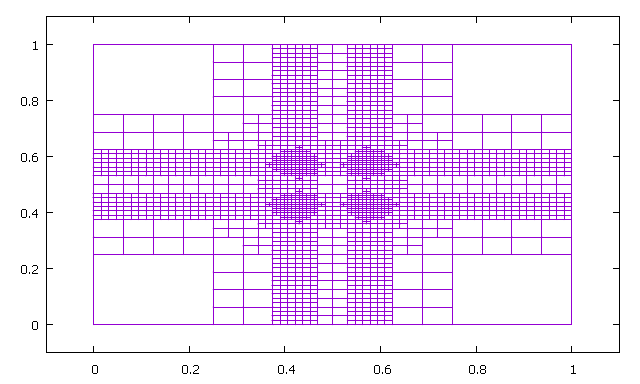
### Tests

Starting with a full quadtree at Level 2 before running your new algorithm.

1. Using choice=0 and tolerance=0.5.



1. Using choice=1 and tolerance=0.2



### Task 5: Reflection

*(i) What went well with this project? Include specific areas of the work, programming, design or testing.*

What I particularly liked about this project was that it was not just a simple console application. The visualization with Gnuplot made it a lot easier to understand what quadtrees are and how they work. It also made the learning a lot more interesting and made testing the program much more enjoyable. I am glad that I managed to grasp the logic behind building and destroying the trees, figuring out if a node is a parent or not and visiting all leaf nodes. Overall, it was a great opportunity to work with pointers and linked lists, which we didn’t get in Semester 1, and I now believe that I have a better understanding of C than before starting the project. Furthermore, I found the use of recursive functions really helpful and I will probably start using them more often in other programs as well. Modular programming is also something I will definitely adapt into my coding style, since it makes errors much easier to find and fix. Moreover, I am now convinced that freeing all allocated memory is of great importance and therefore I will always do my best to prevent memory leaks.

*(ii) What was the hardest part of this work? Why, and what will you do to address this for the future?*

Although the logic behind the algorithm is not too difficult, I did have some trouble with it at first. In Task 1 for example I forgot to free the node after freeing its children and it took me some time to figure out why not all allocated memory was freed. I also had some issues with Task 4, mainly because I did not quite understand the problem. For the future I will try being more focused on the task and rereading the instructions several times.

Another aspect I found problematic was writing the make file and deciding which header files to put where. I finally found a combination that made the program compile and run but I definitely have to do some more research on the topic. I think that we could have used some more examples in the lectures or in the slides to properly understand it. .   
However, the worst part of this project by far was trying to run it on a Windows computer. I do not find it fair that I was put at a disadvantage, because I do not own a Mac or do not run Linux on my computer and while I truly appreciate having all the programs installed on the computers in the labs, going to the labs every day (especially when there are no lectures and during the cold snap that has come over the country) just to do coursework that can be easily done at home is ridiculous. Installing and running valgrind was an absolute nightmare. I am aware that there are Windows alternatives used for exactly the same thing, but since they were not specified in the coursework description, I did not want to take the chance to lose marks.

It is my belief that I am not alone in sharing this opinion. In the future it would be highly appreciated if you could also include Windows alternatives.

### References

(2018 Stack Exchange Inc.) [Last accessed 28.02.2018] https://stackoverflow.com/questions/1841863/size-of-struct-in-c