George Luan

14 March 2022

I finished the whole project without any problems. As I understood correctly, I did not use a banned STL container. I did not have known bugs in my program as far as I know.

**List of Test Cases**

Radix Tree

Ensure that the Radix Tree works correctly with different ValueTypes.

RadixTree<**int**> r1;

r1.insert("Catch", 3);

r1.insert("Catie", 4);

r1.insert("Catchy", 5);

r1.insert("Hello", 6);

r1.insert("He", 7);

r1.insert("By", 8);

r1.insert("Byte", 9);

r1.insert("Byte", 1);

* Insert “Catch” first, “Catie” Next and then “Catchy” and ensure that they are all found in the same Radix tree with the correct corresponding values.

assert(r1.search("Catch") != **nullptr** && \*r1.search("Catch") == 3);

assert(r1.search("Catie") != **nullptr** && \*r1.search("Catie") == 4);

assert(r1.search("Catchy") != **nullptr** && \*r1.search("Catchy") == 5);

* Insert a world and then insert a world that is its prefix and ensure that the insertion works correctly.

assert(r1.search("Hello") != **nullptr** && \*r1.search("Hello") == 6);

assert(r1.search("He") != **nullptr** && \*r1.search("He") == 7);

* Insert a prefix and then insert a world that contains it and ensure that the insertion works correctly.
* assert(r1.search("By") != **nullptr** && \*r1.search("By") == 8);
* Insert duplicate will just update the value.

assert(r1.search("Byte") != **nullptr** && \*r1.search("Byte") == 1);

* Test the case sensitivity

assert(r1.search("byte") == **nullptr**);

RadixTree<string> r2; *// Same Logistics but with a different ValueType*

r2.insert("Catch", "Bob");

r2.insert("Catie", "Bob");

r2.insert("Catchy", "Larry");

r2.insert("By", "Junior");

r2.insert("Byte", "Cindy");

* assert(r2.search("Catch") != **nullptr** && \*r2.search("Catch") == "Bob");
* assert(r2.search("Catie") != **nullptr** && \*r2.search("Catie") == "Bob");
* assert(r2.search("Catchy") != **nullptr** && \*r2.search("Catchy") == "Larry");
* assert(r2.search("By") != **nullptr** && \*r2.search("By") == "Junior");
* assert(r2.search("Byte") != **nullptr** && \*r2.search("Byte") == "Cindy");
* assert(r2.search("byte") == **nullptr**);

AttributeTranslator

Consider the following .txt file:

“favorite\_food,del taco,favorite\_food,del taco\n

favorite\_food,del taco,favorite\_food,mexican\n

favorite\_food,del taco,occupation,del taco employee\n

favorite\_food,taco bell,favorite\_food,del taco\n”

I loaded it into the translator and then searched the attribute value pair “favorite\_food,del taco”. I made sure that the result vector contained “favorite\_food,del taco”, “favorite\_food,Mexican”, and “occupation,del taco employee”. I also made sure that the result vector did not contain “favorite\_food,taco bell”. I then searched the attribute value pair “favorite\_food,taco bell” and made sure the result vector only contained “favorite\_food,del taco”. I finally searched “favorite\_food,” and made sure that the result vector is empty.

PersonProfile

PersonProfile pp("Carey Nachenberg", "climberkip@gmail.com");

AttValPair at;

at.attribute = "Career";

at.value = "Prof";

* assert(!pp.GetAttVal(0, at)); // ensures that the PersonProfile is empty when first created
* assert(at.attribute == "Career" && at.value == "Prof");

AttValPair at1;

at1.attribute = "hobby";

at1.value = "climbing";

pp.AddAttValPair(at1);

AttValPair at2;

at2.attribute = "hobby";

at2.value = "teaching";

pp.AddAttValPair(at2);

* **for** (**int** k = 0; k != pp.GetNumAttValPairs(); k++) {

AttValPair av;

pp.GetAttVal(k, av);

cout << av.attribute << " -> " << av.value << endl;

} // Ensures that all inserted AttriValPairs are found in the profile but nothing more.

pp.AddAttValPair(at2);

* assert(pp.GetNumAttValPairs() == 2); // Inserting duplicate AttriValPairs does nothing

Member Database

Consider the following member file:

“Abdullah Fowler\n

AbFow2483@charter.net\n

13\n

job,architect\n

hobby,canyoneering\n

hobby,slacklining\n

hobby,painting\n

hobby,painting\n

trait,sly\n

trait,incurious\n

trait,excitable\n

trait,unself-critical\n

trait,vacuous\n

trait,discouraging\n

trait,disloyal\n

trait,aloof\n

\n

Mordechai Hayes\n

MHa7@sky.com\n

6\n

job,accountant\n

hobby,origami\n

trait,narrow\n

trait,quirky\n

trait,pedantic\n

trait,uncritical\n

\n

Mordechai Hayes\n

MHa7@gmail.com\n

2\n

job,professor\n

trait,quirky\n

\n

Alice Green\n

ag@sky.com\n

6\n

job,accountant\n

hobby,origami\n

trait,narrow\n

trait,quirky\n

trait,pedantic\n

trait,uncritical\n

\n

Bob Smith\n

sBob@sky.com\n

0\n”.

MemberDatabase md;

md.LoadDatabase("/somePathName/members.txt");

* assert(md.GetMemberByEmail("MHa7@gmail.com") != **nullptr** && md.GetMemberByEmail("MHa7@gmail.com")->GetNumAttValPairs() == 2);
* assert(md.GetMemberByEmail("MHa7@sky.com") != **nullptr** && md.GetMemberByEmail("MHa7@sky.com")->GetNumAttValPairs() == 6);

// Ensures that both members are found in the database, despite they have the same name

AttValPair at;

at.attribute = "job";

at.value = "accountant";

std::vector abc = md.FindMatchingMembers(at);

* assert(abc.size() == 2);

at.attribute = "hobby";

at.value = "painting";

abc = md.FindMatchingMembers(at);

* assert(abc.size() == 1); // Ensures that there is no duplicate in the result

MatchMaker

I used the same member file as above and the following translator file:

“job,architect,job,professor\n

trait,quirky,trait,quirky\n

trait,quirky,hobby,painting\n

hobby,painting,trait,quirky\n”.

I first called for matches for AbFow2483@charter.net and set threshold to 2. I made sure the result only contains MHa7@gmail.com with 2 matches. I next called for matches for AbFow2483@charter.net and set threshold to 0. I made sure the result only contains MHa7@gmail.com with 2 matches, MHa7@sky.com with 1 match, and ag@sky.com with 1 match. I then called for matches for MHa7@gmail.com and made sure the result only contains AbFow2483@charter.net with 1 match, MHa7@sky.com with 1 match ag@sky.com with 1 match. I finally called for match for sBob@sky.com and made sure not matches are found.

Radix Tree Design Concerns

charSequence = “”

value = nullptr

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | C | null | … | Null |

charSequence = “Ca”

value = nullptr

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | r | null | t | …. |

charSequence = “r”

value = someValue

|  |  |  |  |
| --- | --- | --- | --- |
| null | …. | null | null |

charSequence = “t”

value = someValue

|  |  |  |  |
| --- | --- | --- | --- |
| c | … | null | …. |

charSequence = “ch”

value = someValue

|  |  |  |  |
| --- | --- | --- | --- |
| null | …. | null | null |

This is a dummy node that helps with insertion.

Root

Insertion

Case 1: The tree is empty.

charSequence = “”

value = nullptr

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | C | null | … | null |

Root

This is a dummy node that helps with insertion.

charSequence = “Catch”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | …. |

Case 2: Insert an item that doesn’t have any common prefix.

charSequence = “”

value = nullptr

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | B | C | … | null |

Root

This is a dummy node that helps with insertion.

charSequence = “Bob”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | …. |

charSequence = “Catch”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | …. |

Case 3: Insert a key that is a prefix of another existing key.

charSequence = “”

value = nullptr

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | B | C | … | null |

Root

This is a dummy node that helps with insertion.

charSequence = “Cat”

value = someVal

|  |  |  |  |
| --- | --- | --- | --- |
| null | … | null | … |

charSequence = “Catch”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | … |

charSequence = “”

value = nullptr

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| null | … | C | … | null |

Root

This is a dummy node that helps with insertion.

charSequence = “Cat”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | c | null | … |

charSequence = “ch”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | … |

Case 4: Insert a key that has a common prefix with another existing key.

charSequence = “”

value = nullptr

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | B | C | … | null |

Root

This is a dummy node that helps with insertion.

charSequence = “Catie”

value = someVal

|  |  |  |  |
| --- | --- | --- | --- |
| null | … | null | … |

charSequence = “Catch”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | … |

Root

charSequence = “”

value = nullptr

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| null | … | C | … | null |

This is a dummy node that helps with insertion.

charSequence = “Cat”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | c | … | i | null | … |

charSequence = “ie”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | c | … | i | null | … |

charSequence = “ch”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | … |

Case 5: Insert a key that is a superfix of another existing key.

charSequence = “”

value = nullptr

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | B | C | … | null |

Root

This is a dummy node that helps with insertion.

charSequence = “Catch”

value = someVal

|  |  |  |  |
| --- | --- | --- | --- |
| null | … | null | … |

charSequence = “Cat”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | … |

charSequence = “”

value = nullptr

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| null | … | C | … | null |

Root

This is a dummy node that helps with insertion.

charSequence = “Cat”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | c | null | … |

charSequence = “ch”

value = someVal

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| null | … | null | null | null | … |