David Sun

405006245

Project 4 Report

Overall Issues

-All parts of the project were completed.

-Banned STL components were not used.

-There were no bugs to my knowledge.

RadixTree Logic

I had commented in the code that I would specify the logic behind the six cases that were dealt with within the RadixTree insertion method. Following was the logic:

When a string “key” enters a node (p) from a “prev” node, following an edge, and gets rid of overlapping chars with the string held by the current node(m\_curr), the “key” is either:

1. Empty, which became empty when
   1. Following an edge. In which case, there could be
      1. m\_curr present within the node,
         1. Then, make new node (m\_curr = empty;m\_val = value; m\_leaf = true; m\_next[m\_curr[0]], alter “prev” to point to the new node) Also change p->m\_curr = p->m\_curr.substr(1); [Case 2]
      2. Or no m\_curr
         1. Then, the p->m\_leaf = true, p->m\_val = value [Case 1]
   2. Sub-stringing the overlap with m\_curr
      1. Overlap could be Complete, Overlap == m\_curr
         1. Then, the p->m\_leaf = true, p->m\_val = value [Case 1]
      2. Partially, Overlap < m\_curr
         1. Then, make new node(m\_curr = overlap; m\_leaf = true;m\_next[m\_curr[overlap]], “prev” [Case 3]
2. Or Non-Empty, which could be due to
   1. Empty m\_curr, and
      1. There could be No path for key[0] in the current Node
         1. Then, Simply add on the new node for the key and value[Case 4]
      2. There is path for key[0]
         1. Then, Follow Path to next Node
   2. Nonempty m\_curr, and there could be
      1. Partial Overlap with m\_curr
         1. Then, make temp\_1 and temp\_2 Nodes. temp\_2 will store the inputted “key” without the overlapping and the edge chars(m\_leaf = true; m\_value=value; hooked up to temp\_1). temp\_1 will store the overlapping key, (m\_leaf = false, m\_next : key[0] = temp\_2 and p->m\_curr[overlap]). “p” node’s m\_curr will change to m\_curr.substr(overlap+1).[Case 6]
      2. Complete Overlap with m\_curr with key longer than m\_curr
         1. There could be NO path for key[0]
            1. Then, Simply add on the new node for the key and value[Case 4]
         2. There could be path for key[0]
            1. Then, Follow Path to next Node
      3. No overlap
         1. Then, make temp\_1 and temp\_2 Nodes. temp\_2 will store the inputted “key” without edge char(m\_leaf = true; m\_value=value; hooked up to temp\_1). temp\_1 will store NO key, (m\_leaf = false, m\_next : key[0] = temp\_2 and p->m\_curr[0]). “p” node’s m\_curr will change to m\_curr.substr(1).[Case 5]

Tests(In order of implementation)

1. RadixTree

*Insertion:* Following the insertion logic that was mentioned in the previous section, multiple cases were tested using int as “value” of the radixtree, and via a print() function that I made to see all strings, edges, and values that the tree had (

**template** <**typename** ValueType>

**void** RadixTree<ValueType>::print()

{

std::cout<<"TOTAL LEAF:"<<m\_numLeaf<<std::endl;// Display the count of Leaf nodes

printing(m\_head,0);

}

**template** <**typename** ValueType>

**void** RadixTree<ValueType>::printing(Node\* a, size\_t space)

{

std::string spc(space, ' ');

**if** (a->m\_leaf)

{

std::cout <<spc << a->m\_curr <<a->m\_val<< std::endl;

**if** (emptyChild(a))

**return**;

}

**else**

{

std::cout <<spc << a->m\_curr<< std::endl;

}

std::string spac(space+2, ' ');

**for**(**int** i = 0; i < 128; i++)

{

**if**(a->m\_next[i] != **nullptr**)

{

std::cout<<spac<< (**char**)i<<std::endl;

printing(a->m\_next[i],space + 4);

}

}

}

). Few tests cases were : “”, “cash”,”zash”, “zaash”,”zaaash”,”zzaaash”,”zash”, “zzzash”, “Zash”, “ZaAsh”, “zaderfwea”, “zad”, “ash”, “z”, “a”, “s”, “a”,”aa”,”aaaaaaa”,”aaaa”. When the structure was observed using the print() function, all the edge, string, and values were displayed correctly. Furthermore, all the emails from the provided members.txt were read and added into the radix tree. By tweaking [Case 1] of insertion, any overlap was made to be detected. When all the emails were inputted, there were no overlap, and correctly implemented Radix structure confirmed by print() function, and the number of emails were counted to be 100,000.

*Search:* By using previously stated methods of test for the Insertion method of Radix tree, I tweaked few values of specific nodes, and verified the results via the print() function. I searched for values that were inserted, and ones that were not inserted.

1. PersonProfile

*string GetName()* and *string GetEmail()* and *int* *GetNumAttValPairs()*: Simply tested by printing the values returned.

*void AddAttValPair() & bool GetAttVal():*Since Radixtree was rigorously tested, and was confirmed of its functions, test for these functions were not too rigorous. I Added multiple AttValPairs to a person profile, then printed all of them using the GetAttVal() function. Case of duplicate AttValPair addition was also assessed. Also, the structure of the Radixtree within the PersonProfile was assessed using the print() function previously mentioned in the Radixtree, which was tweaked a bit accordingly.

PersonProfile pp("Person", "email@email.com");

AttValPair av1("job", "chef");

AttValPair av2("job", "ch32ef");

AttValPair av3("job", "ch3ef");

AttValPair av4("job", "ch12ef");

AttValPair av5("job", "chef");

AttValPair av6("job", "c2hef");

AttValPair av7("ffick", "ch41ef");

AttValPair av8("job", "ch324ef");

AttValPair av9("job", "ch0011ef");

pp.AddAttValPair(av1);

pp.AddAttValPair(av2);

pp.AddAttValPair(av3);

pp.AddAttValPair(av4);

pp.AddAttValPair(av5);

pp.AddAttValPair(av6);

pp.AddAttValPair(av7);

pp.AddAttValPair(av8);

pp.AddAttValPair(av9);

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

AttValPair av;

pp.GetAttVal(k, av);

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

}

pp.print();//

1. AttributeTranslator

*Load():* Load function was verified by loading the provided “translator.txt”. By using the print() function of the radixtree, which was tweaked for this specific value, and confirming the loaded number of pairs and the actual structure of the radixtree, the validity of the load function was verified.

*FindCompatibleAttValPairs():* After files were loaded in with the Load() function, I called FindCompatibleAttValPairs() function to specific AttValPairs in the loaded file, and compared the outputs with the actual text file usin cmd+F and searching the source/compatible AttValPairs. After confirming the implementation of the function, few compatible AttValPairs were tweaked within the provided file to check for no duplicates. Outputs confirmed that there were no duplicates.

1. MemberDatabase

*LoadDatabase():* Load function was verified by loading the provided “members.txt”. By using the print() function of the radixtree, which was tweaked for this specific value, and confirming the loaded number of pairs and the actual structure of the radixtree (within both of the radixtree used as private data member within the class), the validity of the load function was verified.

*FindMatchingMembers() & GetMemberByEmail() :* Both of these functions were heavily reliant on the search function of RadixTree(which was already rigorously tested multiple time). After load() function was executed, both of these functions were tested by simply plugging in specific AttValPairs and emails from the members.txt and checking if the outputs were accurately in accordance with the actual data in the members.txt.

1. MatchMaker

*IdentifyRankedMatches():* This function was tested by simply testing the entirety of the program and its outputs against the outputs from sample executable. Hundreds of emails were tests, specified from the “members.txt” and threshold values were constantly changed. All of the outputs were identical to the outputs from the sample executable. One thing to note is that the sample executable seemed to have simply used char comparison for ordering(in case emails had same count) because uppercase letters came before the lowercase letters regardless of the alphabetical order. But because the Spec said to sort the emails in “alphabetical” orders (in case emails had same count), I did not directly compare the ascii code of the emails, but converted the emails to lower and then compared the chars in order to sort them “alphabetically”.