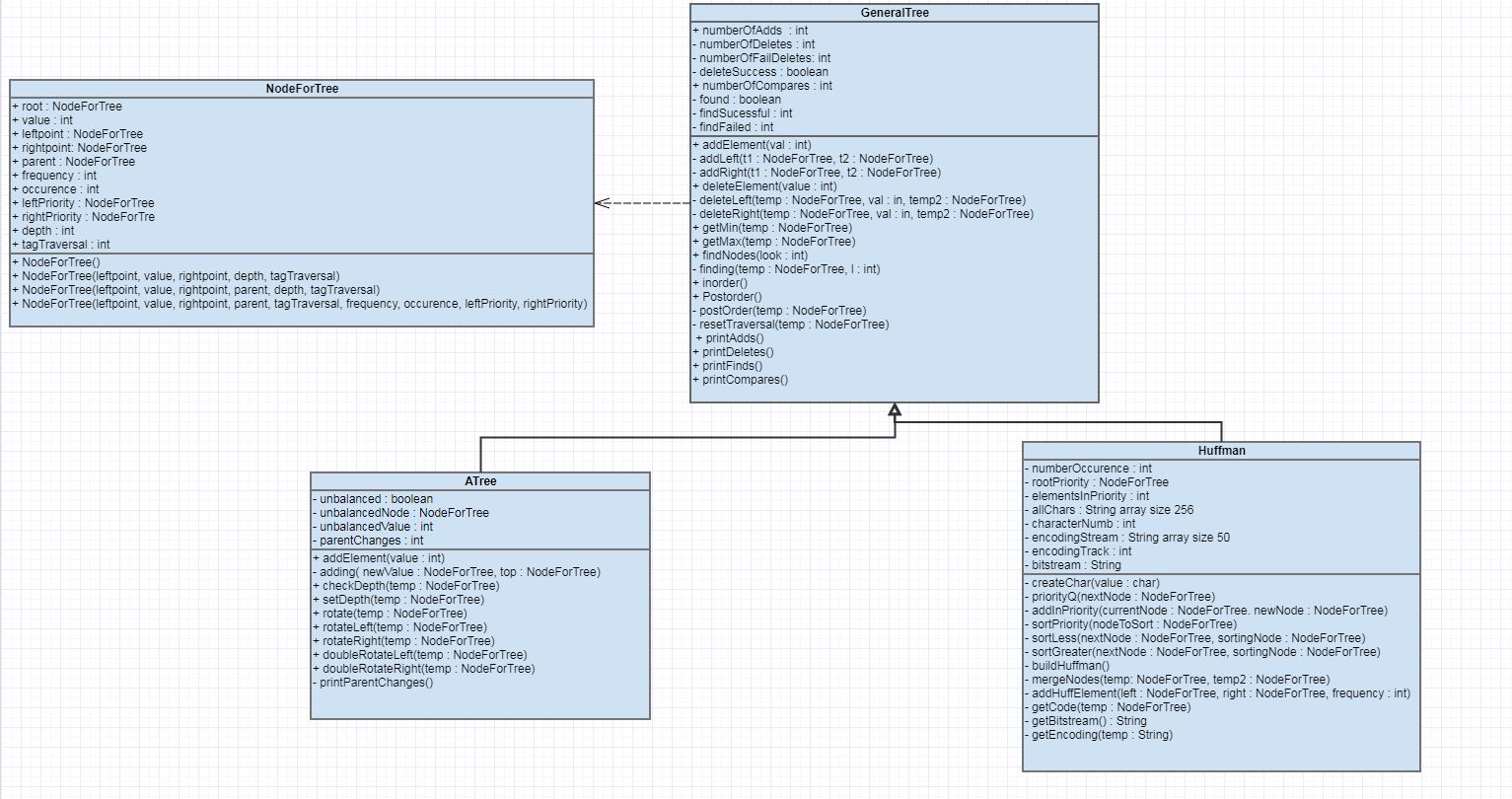
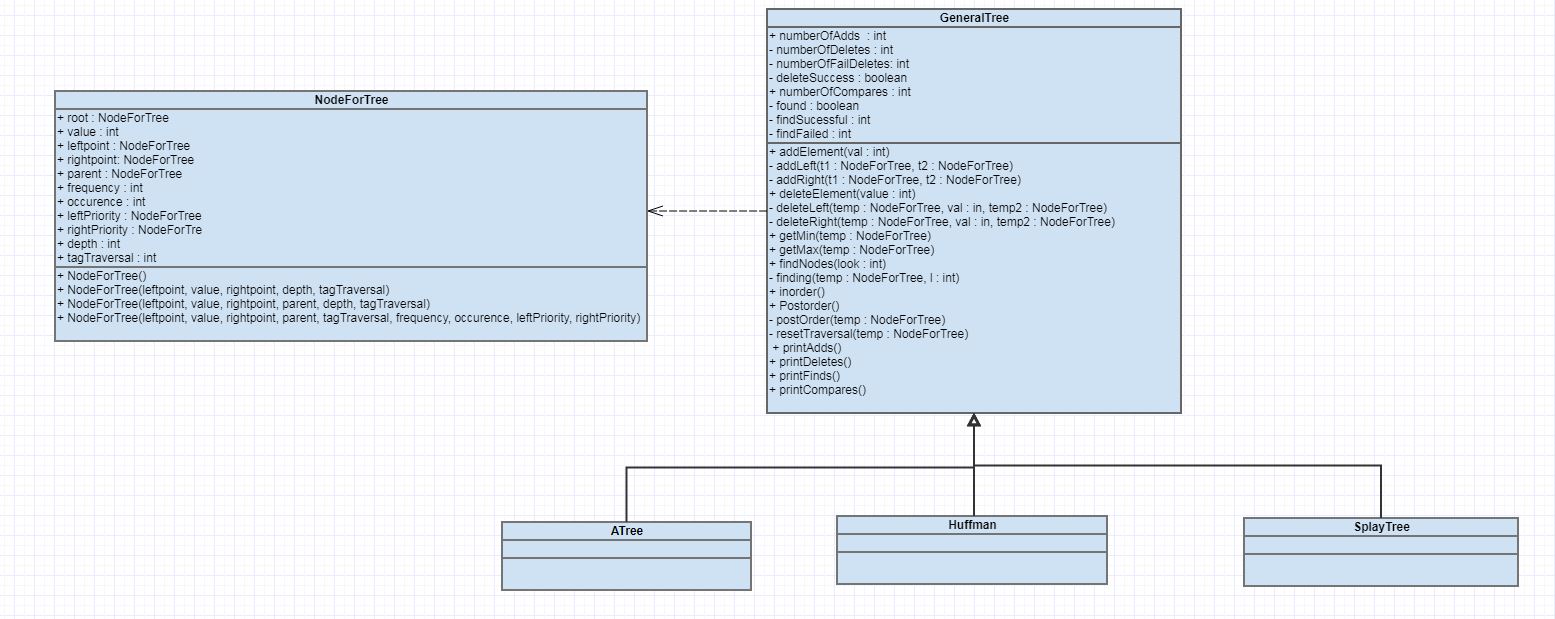
Nicholas Gattuso 40007087

COEN 352 Assignment 3

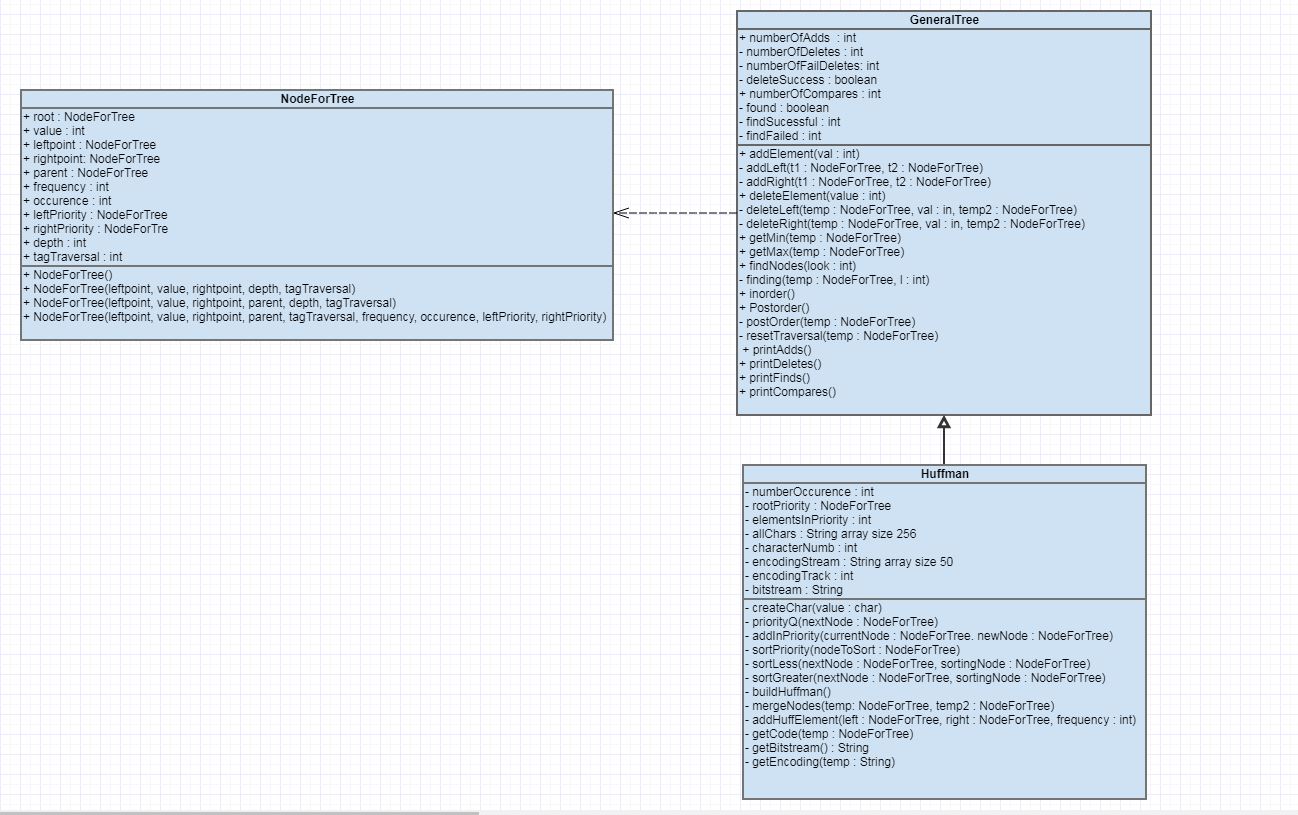
Section 1:

The following is the general UML Class Diagram for my assignment:

1. The following is the UML Class Diagram for my General Tree structure including all the possible subclasses that it can support.

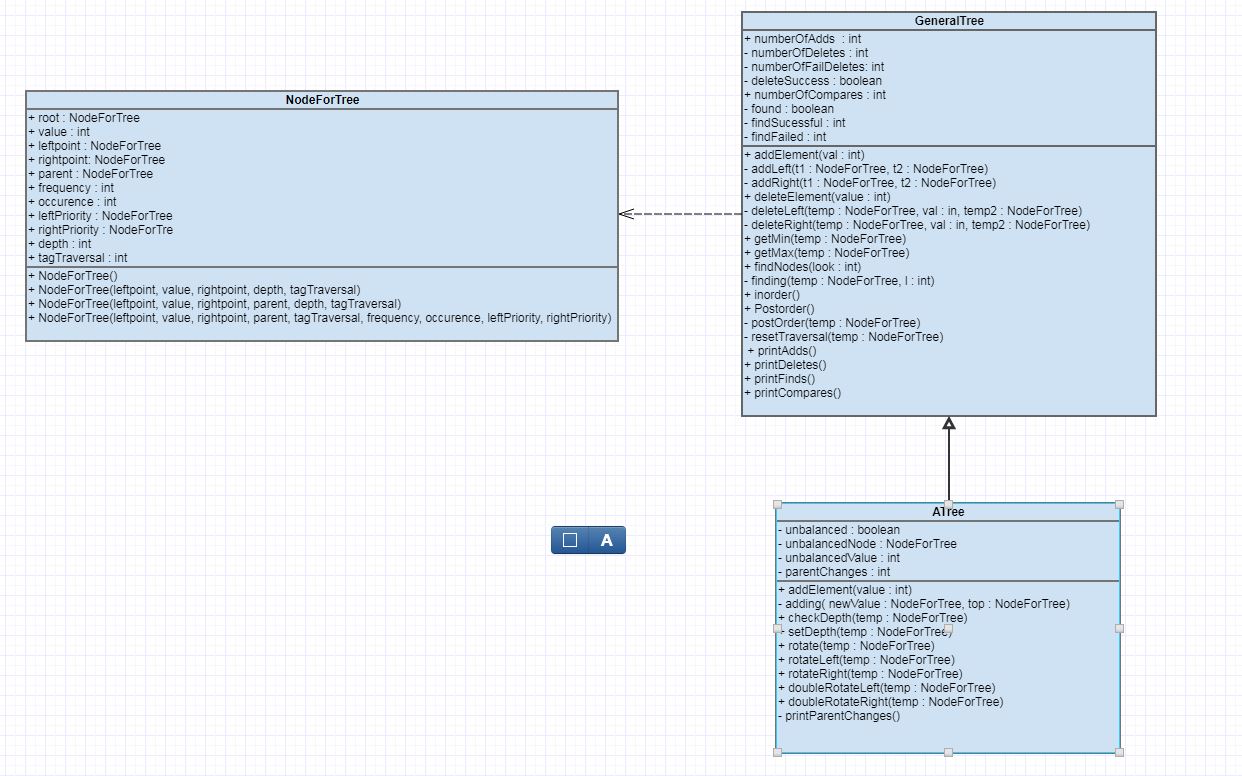
My General Tree depends on the my node class “NodeForTree.” In my node class, I have seven variables and 4 constructors that are used throughout the assignment to create nodes for the trees. I programmed my nodes to have the ability to store the element to its left, to its right, its value, its parent, the depth, if its been traversed, its frequency, its occurrence and its left and right node for a priority queue. Additionally, they can be created using 3 different types of constructors: one for the General tree, one for the AVL tree and, one for the Huffman tree. I made my nodes to contain all this information because for a general tree, the nodes should have a left element, a right element and a value. In addition, I knew that my General Tree will be a superclass and should support other trees; for that reason, I added frequency, occurrence, depth, the parent and, the left and right nodes for the priority queue.

My General tree has been programmed to do the basic operations of any tree structure, thus allowing it to support other tree types. I programmed it to add nodes in the tree, delete a specific node, find a node value and, traverse the tree either in post order or in inorder. Additionally, I have my General tree keeping track of the number of adds, deletes, compares and, finds. With these functions, my General tree will be able to support the AVL Tree, the Huffman Tree and, the Splay Tree, as seen in the Class Diagram. Please note, these subclasses are empty in this question because the focus is the General Tree and what it can support.

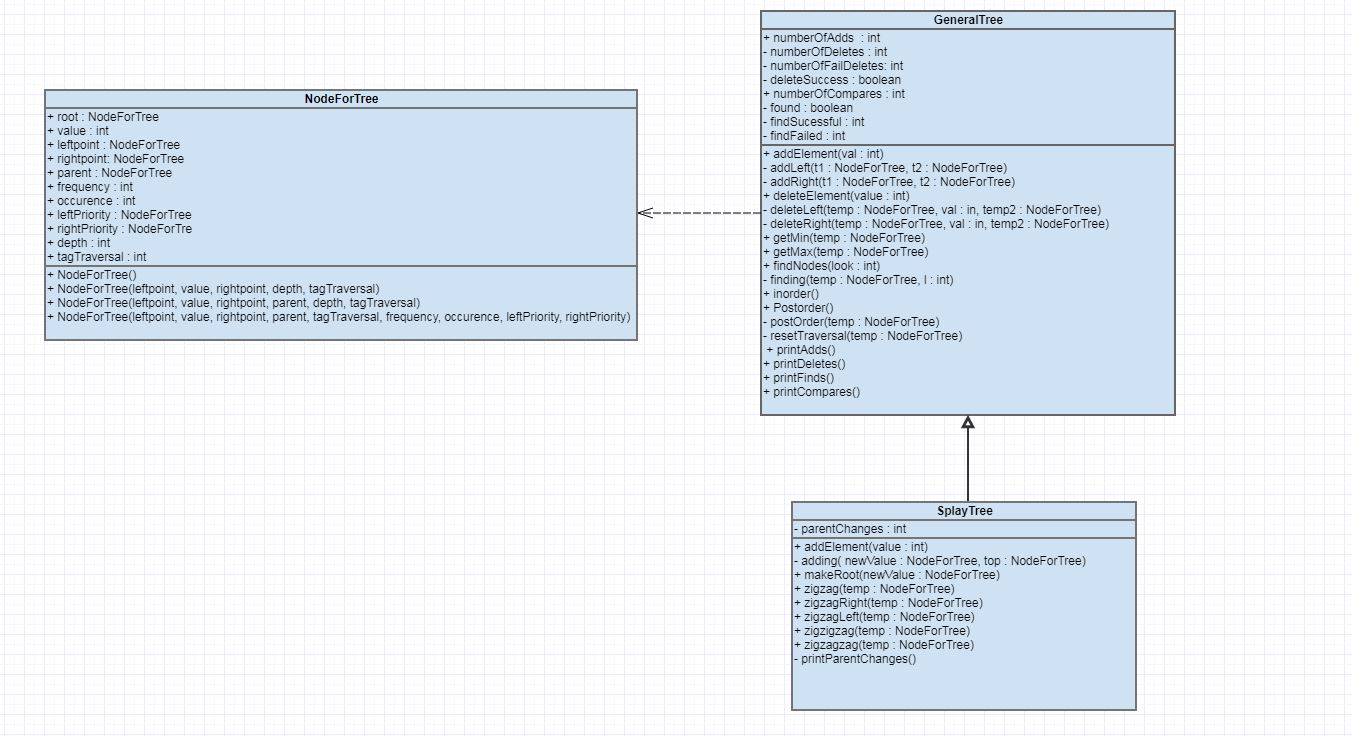
d) The following is the UML Class Diagram that shows how my Huffman Coding tree will extend from my General tree.

Due to how the Huffman Coding tree should performance (based on the assignment question 2), there isn’t much that the General Tree can pass down to my Huffman Tree. That is because all we want to do is build a tree then encode it. My Huffman Coding tree is capable of finding, deleting and, traversing using the functions from the General tree but the assignment doesn’t ask us to do that. In my Huffman class, I added the functions to create a priority queue, merge two nodes, build the Huffman tree from the merged nodes and, encode the Huffman tree. Note, I could’ve added the merge node function in my General tree because merging nodes are used in a General tree for Parent Pointer and for Path Compression. However, I excluded those two implementations in my General tree because I found that they weren’t necessary in this assignment; I would’ve programmed implementations that wouldn’t be used. It’s easier and more ideal for me to make a merge node function that is directly applied with the Huffman tree. By directly applied, I mean that I purposely build my Huffman tree through my merging node function.

f) The following is the UML Class Diagram that shows how my AVL tree will extend from my General tree.



The AVL tree is very similar to the General tree, however it rebalances itself. My AVL tree uses the functions from my General tree and overrides the adding function. I overridden the add function because in my AVL add, I wanted to include the ability for every added node to keep track of its parent, to set and check the depths after an add and, to perform a rotation if the newly added node created an unbalanced node in the tree. I created functions in my ATree class to set the depth starting at the new node and working its way to the root, to check the depth from new node to the root and, to do a rotation at an unbalanced node (single left, single right, double left or, double right). In addition, I added the ability to keep track of every parent change that can occur due to a rotation.

h) The following is the UML Class Diagram that shows how a Splay tree will extend from my General tree.

A Splay tree performs similarly to a General tree except that it makes every recent node the root. Therefore, my thought of the Splay tree implementation is very similar to the AVL implementation. An overridden add function will be created to include the parent pointer. The makeRoot function will make the recent node the root through the various zig zag operations. Please note, I forgot to add a function in the class for SplayTree. A function called performing with parameters task of type char and value of type int will be implemented. This function will read the task then perform either the add, delete or, find. After the call to the certain task, the function will end with calling the makeRoot function.

Section 2 and 3:

These 2 sections are the programs. The files have been included in the same folder as this document.

Section 4:

1. The string that I need to encode based on my student ID is “the lake is a long way from here.”

Running my Huffman program, I get the following output:

Please enter an input to find its encoding: the lake is a long way from here.

The code for the lake is a long way from here. is: 10111000111000100100111010111011100011011111100001110010010011011010111110000100101111010000001000010101011001010001000111010101111011001011

If I converted the string using ASCII conversion I would get:

011101000110100001100101001000000110110001100001011010110110010100100000011010010111001100100000011000010010000001101100011011110110111001100111001000000111011101100001011110010010000001100110011100100110111101101101001000000110100001100101011100100110010100101110

Using the Huffman Coding tree, the encoding of the phrase is almost 2x shorter than using the ASCII conversion. This makes sense because in general, the characters would have 8 bits but Huffman will reduce this based on frequency. The fact that the code from my Huffman is shorter proves that I have some characters in the phrase that occurred more frequently in the Jabberwock text. With a quick glance, we can observe that o, a, i, e and, space occur frequently in the Jabberwock text. These characters should then be higher up in the tree and should have a shorter code. In my phrase, there’s 7 spaces, 4 e’s, 3 a’s, 2 o’s and, 1 i; this results to 17 characters of 33 characters, ie: 51% of the more frequent characters appear in my phrase, thus my encoding should be almost 50% shorter than using ASCII. Therefore, I believe that my encoded string matched the source text frequencies.

1. Using the sample source text file and a step-to-traversal of 100, the output of my ATree class is: (please note, I deleted most traversal prints because they added 10 pages to the report. I kept the first 2 and the last 2)

Traversing Postorder...

4 72 22 1 88 118 111 211 235 225 204 157 126 261 84 82 296 282 304 378 385 400 456 470 349 500 529 598 581 571 641 639 634 646 691 707 709 743 718 674 672 871 830 899 794 915 947 900 563 280

Traversing Postorder...

17 18 4 62 72 22 1 105 88 114 118 111 196 194 162 206 211 256 235 225 204 157 126 261 84 82 296 282 325 330 305 345 304 378 396 385 426 445 400 456 488 495 470 349 502 508 505 500 533 529 582 587 633 619 598 581 571 641 639 634 664 646 691 684 707 709 735 780 772 792 743 718 674 672 838 870 861 848 894 871 830 899 794 906 915 949 957 947 900 563 280

• • •

Traversing Postorder...

0 2 11 16 5 12 17 3 7 8 10 13 15 21 14 9 20 27 41 29 43 23 25 30 28 32 34 35 36 39 40 42 38 33 46 31 48 47 50 52 54 59 56 60 51 49 24 57 58 65 66 64 67 68 70 71 69 53 73 79 80 77 76 75 74 81 72 4 1 85 86 87 90 89 92 94 97 99 95 100 103 102 101 93 109 107 106 91 88 115 113 117 112 120 119 121 123 124 122 125 116 111 128 127 131 132 129 135 137 139 140 138 143 142 145 144 141 149 148 147 151 150 146 133 153 155 154 156 152 159 161 160 158 164 163 167 165 168 173 170 177 176 175 179 182 181 185 189 187 184 180 178 174 191 193 192 190 169 188 195 198 199 202 203 201 196 183 194 205 208 210 206 213 216 215 214 219 220 222 218 224 229 228 231 232 230 226 221 217 236 234 237 238 233 242 246 245 247 244 241 239 254 248 250 252 251 257 255 258 260 259 253 249 212 211 204 157 126 263 264 262 266 267 270 274 276 277 275 271 268 265 261 84 82 284 286 290 287 292 291 297 285 293 283 289 294 301 303 298 306 300 309 310 307 313 315 312 318 314 311 319 320 321 316 323 324 328 322 329 326 338 325 299 331 335 337 336 332 340 341 339 344 347 342 330 282 348 355 351 354 357 367 359 368 352 356 362 361 364 365 363 360 372 369 366 373 375 374 377 376 358 382 383 380 378 389 391 390 394 392 387 395 397 386 385 399 398 401 404 402 405 409 408 412 413 410 406 416 418 420 415 424 419 417 423 444 425 427 429 430 428 433 432 435 437 438 436 442 441 446 443 440 439 448 447 449 431 453 451 454 452 450 426 396 460 457 461 463 465 459 456 467 464 466 472 475 478 476 474 480 482 486 484 481 469 491 493 492 489 483 497 499 498 496 495 462 346 501 503 504 502 506 507 514 509 516 513 519 522 526 523 527 518 510 508 505 500 530 532 531 536 534 537 539 541 543 540 547 545 542 538 548 550 551 552 554 556 558 560 561 557 559 555 553 549 533 529 564 567 565 572 570 577 576 573 580 578 583 585 584 586 582 590 588 592 594 593 596 589 587 602 599 601 604 607 608 603 611 613 617 605 612 610 618 616 615 625 628 626 623 633 606 595 579 569 624 629 632 627 622 636 638 635 640 643 637 619 645 642 647 648 652 654 653 651 650 657 659 658 656 662 661 660 663 666 667 668 671 670 669 664 641 675 673 681 683 682 680 685 686 688 692 690 689 696 695 698 699 701 700 697 693 704 706 705 702 687 677 707 710 711 713 712 716 717 715 714 709 720 726 721 723 727 724 730 734 722 719 732 737 738 731 736 746 749 751 754 750 757 755 762 766 769 765 763 759 779 786 791 787 785 780 772 792 735 718 676 672 748 745 752 756 753 744 760 758 764 768 767 771 776 775 777 781 784 782 778 770 790 789 793 788 761 796 806 808 799 829 812 836 831 840 839 841 847 838 859 856 865 870 861 848 877 881 878 875 874 886 887 894 871 830 899 743 795 797 816 800 810 803 814 805 807 821 813 802 825 815 801 824 833 832 835 834 827 843 844 850 852 849 846 837 855 851 860 862 858 866 864 863 868 872 873 869 883 882 885 879 889 891 892 890 880 896 902 895 867 898 897 822 905 911 913 907 916 914 918 917 910 919 922 921 920 909 926 935 927 924 931 928 930 938 934 932 940 929 942 945 944 946 943 941 906 948 950 952 954 953 951 955 956 958 960 962 961 963 968 969 966 971 972 975 974 973 970 976 965 978 980 979 982 981 977 986 987 988 985 990 989 984 992 993 995 994 996 991 983 959 999 998 949 947 794 563 280

Traversing Postorder...

0 2 11 16 5 12 17 3 6 7 8 10 13 15 21 14 9 20 27 41 29 43 23 25 30 28 32 34 35 37 36 39 42 38 33 46 31 48 47 50 52 54 59 56 60 51 49 24 55 57 65 64 67 68 70 71 69 53 73 79 80 77 76 75 74 81 72 4 1 85 87 90 92 94 98 97 99 95 100 103 102 101 93 109 107 106 89 88 115 113 117 112 120 119 121 123 124 122 125 116 111 128 127 130 131 132 129 135 137 139 140 138 143 142 145 144 141 149 148 147 151 150 146 133 153 155 154 156 152 159 161 160 158 164 163 167 165 168 171 173 170 177 176 175 179 182 181 185 189 187 184 180 178 174 191 193 192 190 169 188 195 198 199 202 203 201 196 183 194 205 207 209 208 210 206 213 216 215 214 219 220 222 218 224 227 229 228 231 232 230 226 221 217 236 234 237 238 233 246 245 247 242 241 239 254 244 250 252 251 257 255 258 260 259 253 249 212 211 204 157 126 263 264 262 266 267 270 274 276 277 275 271 268 265 261 86 82 281 284 286 290 287 292 291 297 285 293 283 289 295 294 301 303 298 306 300 309 307 313 315 312 318 314 311 319 320 321 316 323 324 328 322 329 326 338 325 299 331 334 335 337 336 332 340 341 339 343 344 347 342 330 282 348 353 355 351 354 357 367 359 368 352 356 362 361 364 365 363 360 370 372 369 366 373 375 374 377 376 358 379 382 383 380 378 389 391 390 392 387 395 397 386 385 399 398 401 404 402 405 409 408 411 412 414 413 410 406 416 418 420 415 424 419 417 423 444 425 427 429 430 428 433 432 435 437 438 436 442 441 446 443 440 439 448 447 449 431 453 454 451 450 426 396 460 457 461 463 465 459 456 467 464 466 472 475 478 476 474 480 482 486 484 481 469 491 493 492 489 483 497 499 498 496 495 462 346 501 503 504 502 506 507 514 509 511 512 516 513 519 520 522 526 523 527 518 510 508 505 500 530 532 531 536 534 537 539 543 540 547 545 542 538 548 550 551 554 556 558 560 561 557 559 555 553 549 533 529 564 566 567 565 572 570 577 576 573 580 578 583 585 584 586 582 590 588 592 594 593 596 589 587 600 602 599 601 604 607 608 603 611 613 617 605 612 610 618 616 615 625 628 626 623 633 606 595 579 569 624 629 632 627 622 636 638 635 640 643 637 619 645 642 648 652 654 653 651 650 657 659 658 656 662 661 660 663 666 667 668 671 669 664 641 675 673 681 683 682 680 685 686 688 692 690 689 696 695 698 699 701 700 697 693 704 706 705 702 687 677 707 710 711 713 712 716 717 715 714 709 720 726 721 723 727 724 730 734 722 719 732 737 738 731 741 740 736 746 749 751 754 750 757 755 762 766 769 765 763 759 779 786 791 787 785 780 772 792 735 718 676 670 748 745 752 756 753 744 760 758 764 768 767 773 771 776 775 777 781 784 782 778 770 790 789 793 788 761 796 806 808 799 829 812 836 831 840 839 841 847 838 859 856 865 870 861 848 877 881 878 875 874 886 887 894 871 830 899 743 795 797 816 800 810 803 814 805 809 807 821 813 802 817 825 815 801 824 828 833 832 835 834 827 843 844 850 852 849 846 837 855 851 860 862 858 866 864 863 868 872 873 869 883 882 885 879 884 889 891 893 892 890 880 896 902 895 867 901 898 897 822 905 911 913 907 916 914 918 917 910 919 923 922 921 920 909 926 935 927 924 931 928 930 937 938 934 932 940 929 942 945 944 946 943 941 906 948 950 952 954 953 951 955 956 958 960 962 961 964 963 968 969 966 971 972 975 974 973 970 976 965 978 980 979 982 981 977 986 987 988 985 990 989 984 992 993 995 994 996 991 983 959 999 998 949 947 794 563 280

There were 1000 adds completed

There were 27948 compares performed

There were 147 element(s) deleted and 237 unsuccessful deletes possibly due to non-existing element

The number of successful finds is 451 the number of failed finds is 810

There were 0 parent changes performed

By observing the last traversal, we can see how the output is almost in increasing order. This is due to constantly balancing the tree. We can also observe the number of adds, compares, deletes, finds and, parent changes performed. The number of parent changes in this example is being shown as 0. I believe this is an error because in my code, I increment the variable for counting the parent changes only when there’s a rotation. The fact that the output says that there’s 0 changes would then imply that I never enter my rotation function that would then imply that the elements in the text were added pretty equally so an unbalanced never occurred. This may be possible but I would assume that the text given by the teacher for us to test would at least cause an unbalance at some point. With that being said, I couldn’t narrow down any other possible cases for why I’m getting 0 parent changes in this case. I do know that my rotation functions work when there’s an unbalanced because before testing with the text file, I purposely added nodes that would create an unbalance. It was this sort of testing that allowed me to check and correct my rotation function before implementing testing with the text file.

I believe that AVL was a right choice for the given data. If we were to run this data on a General tree, there would be more nodes on the right subtree since the first add element was 280. That would mean any find operations on the right subtree is would have taken longer and is more likely to reach the worst case. With the AVL tree, our tree kept balanced which would increase the performance time compared to the General tree case.

Lastly, I woudn’t say that there’s much that would improve my future designs. The best way to see how I can improve is through the results of this assignment. From there, I can see where I went wrong and work on making that better. However, I can say that I made more use of recursion as I progressed in the assignment. This can be seen through the General tree where I didn’t use recursion as much, to the ATree and Huffman where I used it more frequently. I have learnt quite a bit about creating nodes, trees, priority queue, double linked list and, reading text files that I might be able to reprogram them in a simpler and more efficient method if I was to come across them again.