Assignments P2.1 - Part 4

General remarks

All assignments are to be programmed in *portable* Fortran 2003. Also all programs have to be "valgrind clean", i.e. valgrind's memcheck tool should report: *All heap blocks were freed -- no leaks are possible* and also: *ERROR SUMMARY: 0 errors from 0 contexts* unless the reported errors are caused by the compiler (e.g. Intel fortran).

21 Binary search tree

Implement a binary search tree (BST) and test and benchmark it in a similar fashion to the linked list and hash table sections (17 and 18). Since tree traversal is most easily done with recursions, use recursive subroutines and functions for that. Implement functionality to add items to the tree, locate items by value (the "key" entry of the pair data type), and to free all allocated storage. Compare lookup performance to array, linked list and hash table.

22 Tree utilities

Now add the following features to your BST implementation: a function to count the number of nodes (i.e. stored items), a subroutine to extract the content of the tree, ordered by value, into an array of suitable size provided by the calling code, a subroutine to print out the "depth" of the tree and number of nodes with only one leaf.

23 Tree rebalancing

Implement the following strategy to rebalance the tree: 1) extract the content of the tree into a sorted array; 2) allocate a new root node; 3) determine the middle of the array and assign the data at this location to the new root node; 4) rebuild the tree by calling a function that takes an array of data elements with the part of the array before the middle and the part of the array after the middle element; this function determines the middle of the passed in array and adds it to the tree and then recursively calls itself on the two remaining parts. 5) free the old tree and assign the root node pointer with the location of the new tree.
Benchmark the tree lookup performance after the rebalancing and compare depth and number of open leafs before and after the rebalancing.