DS HW7

Deadline(107/12/4)

手寫題

2. Create a binary search tree using the following data entered as a sequential set:

```
14 23 7 10 33 56 80 66 70
```

10. The binary search tree in Figure 7-19 was created by starting with a null tree and entering data from the keyboard. In what sequence were the data entered? If there is more than one possible sequence, identify the alternatives.

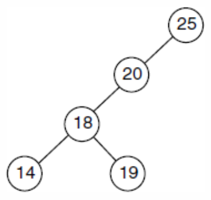

FIGURE 7-19 Figure for Exercise 10

14. Delete the node containing 85 from the binary search tree in Figure 7-22.

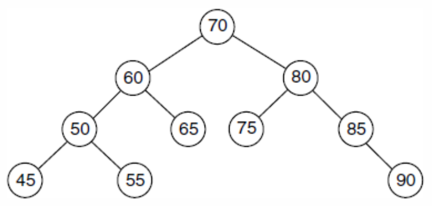

FIGURE 7-22 Figure for Exercises 13 and 14

16. Develop a nonrecursive algorithm for Algorithm 7-3, “Search BST.”

ALGORITHM 7-3 Search BST

```
Algorithm searchBST (root, targetKey)
Search a binary search tree for a given value.
   Pre    root is the root to a binary tree or subtree
          targetKey is the key value requested
   Return the node address if the value is found
          null if the node is not in the tree
1 if (empty tree)
      Not found
   1  return null
2 end if
3 if (targetKey < root)
   1  return searchBST (left subtree, targetKey)
4 else if (targetKey > root)
   1  return searchBST (right subtree, targetKey)
5 else
      Found target key
   1  return root
6 end if
end searchBST
```

程式題

22. Write a program that reads a list of names and telephone numbers from a text file and inserts them into a BST tree. Once the tree has been built, present the user with a menu that allows him or her to search the list for a specified name, insert a new name, delete an existing name, or print the entire phone list. At the end of the job, write the data in the list back to the file. Test your program with at least 10 names.

24. Write a program that processes a threaded binary tree. The program should first build the tree, then use an iterative traversal to process it using the threads.

(以下為課本 Threaded Trees 內容供同學參考)

7.5 Threaded Trees

Binary tree traversal algorithms are written using either recursion or programmer-written stacks. If the tree must be traversed frequently, using stacks rather than recursion may be more efficient. A third alternative is a

threaded tree. In a threaded tree, null pointers are replaced with pointers to their successor nodes.

Using a stack for each call makes the binary tree traversal relatively inefficient, particularly if the tree must be traversed frequently. The reason we use recursion or a stack is that, at each step, we cannot access the next node in the sequence directly and we must use backtracking. The traversal is more efficient if the tree is a threaded tree.

For example, in the inorder traversal of a binary tree, we must traverse the left subtree, the node, and the right subtree. Because an inorder traversal is a depth-first traversal, we follow left pointers to the far-left leaf.

When we find the far-left leaf, we must begin backtracking to process the right subtrees we passed on our way down. This is especially inefficient when the parent node has no right subtree. Consider the tree shown in Figure 7-16(a).

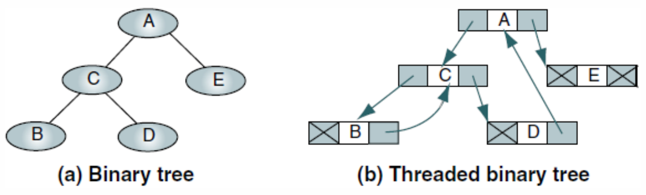

FIGURE 7-16 Threaded Binary Tree

The inorder traversal of this tree is BCDAE. When we traverse the tree in inorder sequence, we follow the left pointers to get the first node, `B`. However, after locating the far-left node, we must go back to `C`, which is why we need recursion or a stack. Note that when we are processing `B`, we do not need recursion or a stack because `B`'s right subtree is empty.

Similarly, when we finish processing node `D` using recursion or stacks, we must return to node `C` before we go to `A`. But again, we do not need to pass through `C`. The next node to be processed is the root, `A`.

From this small example, it should be clear that the nodes whose right subtrees are empty create more work when we use recursion or stacks. This leads to the threaded concept: when the right subtree pointer is empty, we can use the pointer to point to the successor of the node. In other words, we can use the right null pointer as a thread. The threaded tree is shown in Figure 7-16(b).

To build a threaded tree, first build a standard binary search tree. Then traverse the tree, changing the null right pointers to point to their successors.

The traversal for a threaded tree is straightforward. Once you locate the far-left node, you loop, following the thread (the right pointer) to the next node. No recursion or stack is needed. When you find a null thread (right pointer), the traversal is complete.