Threaded Binary Trees

Joseph Chuang-Chieh Lin (林莊傑)

Department of Computer Science & Engineering,
National Taiwan Ocean University

Fall 2024

Outline

Outline

Threaded Binary Trees

Issue

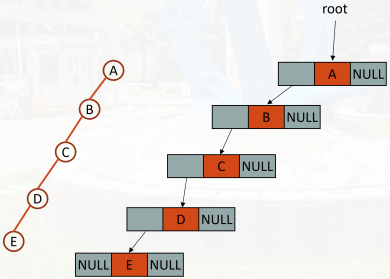
There are more null links than actual points.

Threaded Binary Trees

Issue

There are more null links than actual points.

- Number of nodes: n.
- Number of null non-null links: $n - 1$.
- Number of null links: $n + 1$.

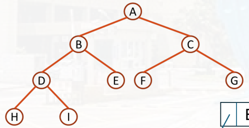


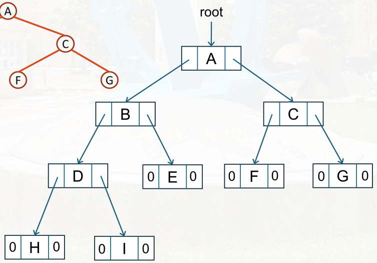
Threaded Binary Trees

Issue

There are more null links than actual points.

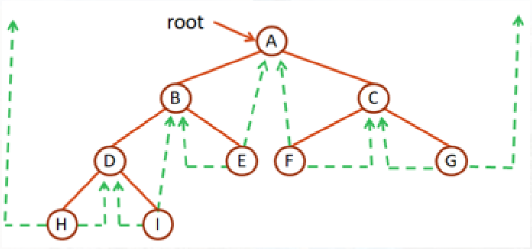
- Number of nodes: n.
- Number of null non-null links: $n-1$.
- Number of null links: $n+1$.





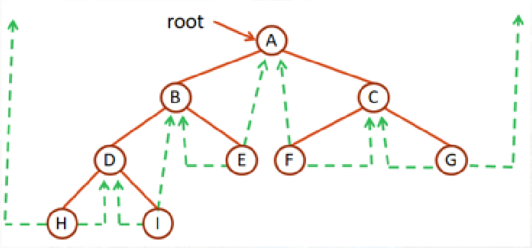
Solution

Replace the NULL links by pointers, threads, pointing to other nodes.



Solution

Replace the NULL links by pointers, threads, pointing to other nodes.



Threading Rules

- if `ptr->leftChild` is NULL, then `ptr->leftChild` = inorder predecessor (中序前行者) of `ptr`.
- if `ptr->rightChild` is NULL, then `ptr->rightChild` = inorder successor (中序後續者) of `ptr`.

To distinguish between normal pointers and threads

- Two additional fields of the node structure: left-thread, right-thread.

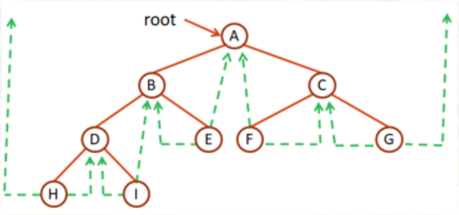
```
typedef struct threadedTree *threadedPointer;

typedef struct threadedTree {
    bool leftThread;
    threadedPointer leftChild;
    char data;
    threadedPointer rightChild;
    bool rightThread;
};
```

leftThread	leftChild	data	rightChild	rightThread

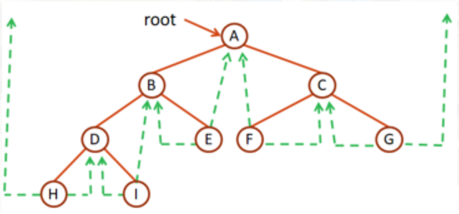
Rules of the Threading Fields

- If `ptr->leftThread == true`, `ptr->leftChild` contains a thread; Otherwise, the node contains a pointer to the left child.
- If `ptr->rightThread == true`, `ptr->righChild` contains a thread; Otherwise, the node contains a pointer to the right child.



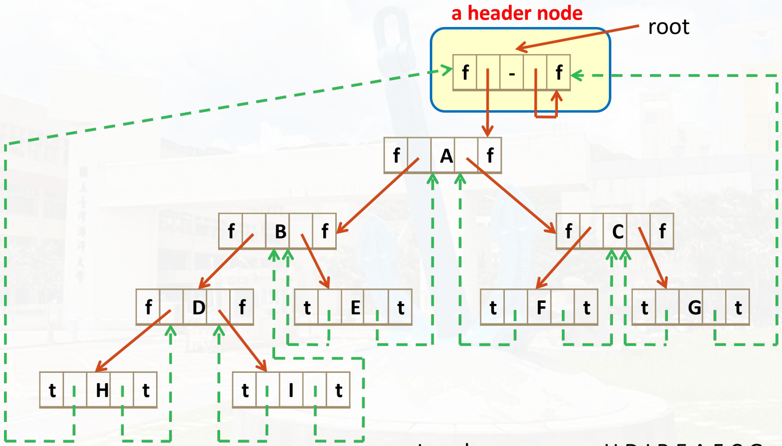
Rules of the Threading Fields

- If `ptr->leftThread == true`, `ptr->leftChild` contains a thread; Otherwise, the node contains a pointer to the left child.
- If `ptr->rightThread == true`, `ptr->righChild` contains a thread; Otherwise, the node contains a pointer to the right child.



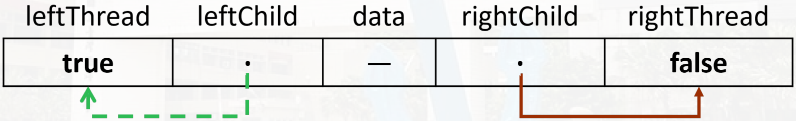
- Two dangling threads at node *H* and *G*.
  $\Rightarrow$ Use a header node to collect them!

- The original tree becomes the left subtree of the head node.



Inorder sequence: H D I B E A F C G

Representing an Empty Binary Tree



Finding the Inorder Successor of Node

```
threadedPointer insucc(threadedPointer tree) {
/* find the inorder sucessor of tree in a threaded
   binary tree */
    threadedPointer temp;
    temp = tree->rightChild;
    if (!tree->rightThread) // rightChild exists!
        while (!temp->leftThread)
            temp = temp->leftChild;
    return temp;
}
```

To perform an inorder traversal, we can simply make repeated calls to insucc!

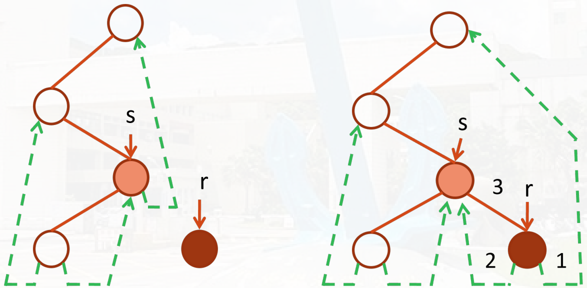
Inorder Traversal of a Threaded Binary Tree

```
void traverseInorder(threadedPointer tree) {
/* traverse the threaded binary tree inorder */
    threadedPointer temp = tree;
    while (1) {
        temp = insucc(temp);
        if (temp == tree)
            break;
        printf("%3c", temp->data);
    }
}
```

- **Note:** `temp == tree` happens when the last node is visited (then the successor becomes the header node).

Inserting *r* as the rightChild of a node *s*

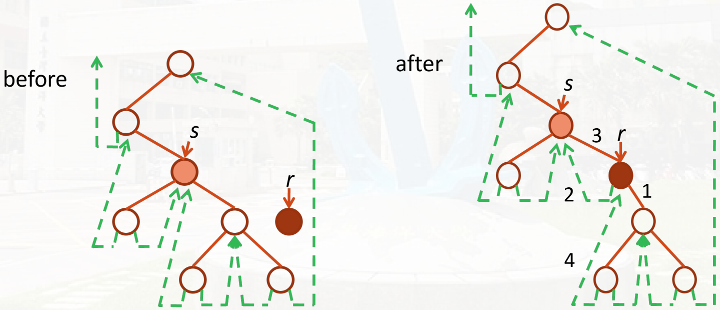
- Case I: `s->rightThread == true` (*s* has an empty subtree)



before after

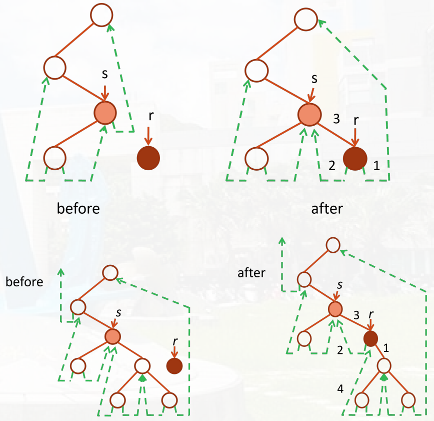
Inserting *r* as the rightChild of a node *s*

- Case II: `s->rightThread == false`
  (the right subtree of *s* is not empty)



The Code for the Insertion

```
void insertRight (threadedPointer s,
                  threadedPointer r) {
/* insert r as the right child of s */
    threadedPointer temp;
    r->rightChild = s->rightChild;
    r->rightThread = s->rightThread; // (*)
    r->leftChild = s;
    r->leftThread = true;
    s->rightChild = r;
    s->rightThread = false;
    if (!r->rightThread){ // step 4 (*)
        temp = insucc(r);
        temp->leftChild = r;
    }
}
```



Discussions