자료구조 중간고사, 4/24/2019

Open Book, 70분

* 기말고사도 Open Book입니다

* 기말고사에 가지고 들어올 수 있는 것:

  교과서 최대 2권

  강의 노트 pdf 프린트한 것

  - 여러분이 필요한 메모를 해도 됨.
  - 기출 문제 푼 것을 적어놓아도 됨.
  - 단, 강의 노트 pdf 원본은 당연히 프린트해야겠지만 여러분이 메모한 것은 프린트하면 안됩니다. 반드시 손으로 프린트물 위에 쓴 것이라야 합니다. 이것은 타인의 메모를 그대로 복사하는 것을 피하기 위해서입니다.
  - 강의 영상 캡처 불허(여러분이 강의 영상 내용을 pdf 프린트물 위에 적는 것은 괜찮습니다. 스스로 한번 새긴 것이면 괜찮다는 것.)

  * 정리하면, 프린트할 수 있는 것은 깨끗한 강의 노트 pdf 원본뿐입니다. 그 위에 '자필 손글씨로' 적는 것만 허용합니다.

****

* 질문 불허로 표시된 것은 문제 오류가 의심되는 경우를 제외하고는 질문해서는 안된다.
* 그렇지 않기를 바라지만 만일 가정이 추가로 필요한 경우가 있으면 스스로 가정하고 진행하라. 가정이 필요하고 합리적이면 추가로 점수를 줄 수 있다. 불필요한 가정은 감점 요인이 됨.

1. (10점) 아래는 피보나치 수열 값을 계산하는 recursive 알고리즘이다.

```
int fib(int n)
{
        if (n==1 or n==2) then return 1;
        else return (fib(n-1) + fib(n-2);
}
```

   함수가 수행될 때 호출된 후 아직 끝나지 않은 상태의 함수들에 관한 정보는 stack 영역에 저장된다. fib(*n*)이 수행되는 과정에서 stack 영역에는 최대 몇 개까지의 fib() 함수 정보가 저장되는가? (정확한 수를 요구함)

2. (10점) 아래는 Heap Sort 알고리즘이다. Array A[1...10]에 값이 [10,9,8,8,8,5,4,3,2,1]의 순서로 들어있을 때, 함수 percolateDown()은 총 몇 번 호출되는가?

```
heapsort(A[], n) {
        // build array A[1···n] to heap
        for i = n/2 downto 1 {
                percolateDown(A, i, n);
        }
        // delete one by one
        for size = n downto 2 {
                A[1] ↔ A[size]; // 원소 교환
                percolateDown(A, 1, size-1);
        }
```

```
}

percolateDown(A[], i, n) {
        child = 2*i;
        rightChild = 2*i + 1;
        if (child <= n) {
                if ((rightChild <= n) && (A[child] < A[rightChild])) {
                        child = rightChild;  // index of larger child
                }
                if (A[i] < A[child]) {
                        A[i] ↔ A[child]; // 원소 교환
                        percolateDown(A, child, n);
                }
        }
}
```

3. (10점) 아래는 prefix expression을 postfix로 바꾸는 recursive 알고리즘이다. 길이 n인 prefix expression A[1...n]으로 알고리즘 convert()가 수행되면 끝날 때가지 convert()는 총 몇 번 호출되는가? (최초의 호출도 1회로 계산한다) 이유에 대한 간단한 설명도 해야한다.

```
convert(pre)
{  // pre : a valid prefix expression
   // return the equivalent postfix expression
        ch = the 1st character of pre;
        Delete the 1st character from pre;
        if (ch is an identifier) return ch;
        else {  // ch is an operator
                postfix1 = convert(pre);
                postfix2 = convert(pre);
                return postfix1 • postfix2 • ch;  // concatenation
        }
}
```

4. (15점) Maxheap A[1...n]에서 임의의 A[i]($1 \le i \le n$)의 값이 바뀌었다. 값은 커질 수도 있고 작아질 수도 있다. 이 결과 maxheap의 heap property가 깨질 수 있다. 이를 수선하는 알고리즘을 제시하라. 아래 물음표 부분을 채워넣으면 된다. 수업 시간에 배운 알고리즘을 사용하는 경우에는 그냥 호출하면 된다.

```
heapRepair(A[], i, n]
// A[i]: 값이 바뀐 원소
{
                ???
}
```

5. (15점) 아래는 수업 시간에 배운 Hanoi Tower 문제를 위한 알고리즘이다.

```
move(n, A, B, C)
{
        if (n==1) then move the disk from A to B
        else {
                move(n-1, A, C, B);
```

```
                move(1, A, B, C);
                move(n-1, C, B, A);
            }
    }
```

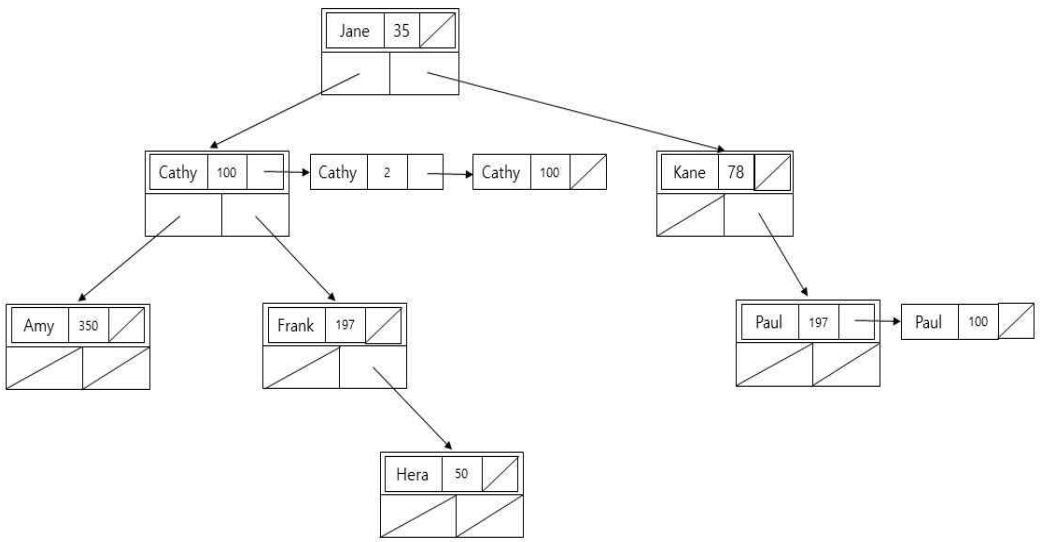
이것을 조금 변형해보자. 이번에는 3개의 pole 대신 4개의 pole A, B, C, D를 사용할 수 있다고 하고, *n*개의 원반을 pole *A*에서 pole *B*로 옮기는 알고리즘을 제시하라. 한 번에 원반을 하나만 옮기는 성질은 그대로 유지한다. **당신이 생각하기에** 원반을 옮기는 횟수를 최소로 하도록 알고리즘을 만들어 보라. 아래 물음표 부분을 채워 넣으면 된다.

```
move(n, A, B, C, D)
{
    ???
}
```

6. (15점) 아래는 Bubble Sort 알고리즘이다. 이를 recursive 알고리즘으로 바꾸어 보아라.

```
bubbleSort(A[], n)
// A[1 ... n]을 정렬한다
{
   for last ← n downto 2
          for i ← 1 to last-1
                if (A[i] > A[i+1]) then A[i] ↔ A[i+1]; // 원소 교환
}
```

7. (25점, 질문 불허) Binary Search Tree(BST)에서 모든 노드의 key 값은 서로 달라야 한다. 그런데 종종 동일한 key 값을 갖는 서로 다른 record들을 색인할 필요가 있을 수 있다. 예를 들면, 동명이인이 있을 수 있는 이름을 key 값으로 BST를 만들어야 할 때가 있다. 간명함을 위해 BST의 key 값을 문자열(string)로 제한하자. 모든 노드의 key 값이 다른 성질을 유지시켜 주되 같은 key 값을 가지는 record는 BST의 한 노드에서 linked list로 매달도록 하려 한다. 아래 그림과 같은 구조로 구현하려 한다. 이름 뒤의 수는 해당 record가 저장된 disk block number이다.



본 문제를 위해 아래와 같이 디자인한다.

```
public class ItemNode {
        private String item;
        private int blockId; // 해당 item의 record가 있는 block#.
        private ItemNode next;
        public ItemNode(String inputItem) {
                item = inputItem; next = null;
        }
        public ItemNode(String inputItem, ItemNode nextNode) {
                item = inputItem; next = nextNode;
        }
        // 아래 method들은 interface 참조용. 그냥 호출하면 된다.
        // public void setItem(String inputItem, int blockNum)
        // public String getItem( )
        // public void setNext(ItemNode nextNode)
        // public ItemNode getNext( )
        …
}

public class TreeNode {
        private ItemNode item;
        private TreeNode leftChild
        private TreeNode rightChild
        public TreeNode(ItemNode newItem) {
                item = newItem; leftChild = rightChild = null;
        }
        public TreeNode(ItemNode newItem, TreeNode left, TreeNode right) {
                item = newItem; leftChild = left; rightChild = right;
        }
        // 아래 method들은 interface 참조용. 그냥 호출하면 된다.
        // public void setItem(ItemNode inputItem)
        // public ItemNode getItem( )
        // public TreeNode getLeft( )
        // public TreeNode getRight( )
        // public void setLeft(TreeNode left)
        // public void setRight(TreeNode right)
        …
}
```

여기서 문자열(string)의 동일 여부나 대소 비교는 문제의 본질에 집중하기 위해 '==', '<', '>'로 처리하기로 한다. 검색은 아래와 같이 한다.

```
ItemNode search(TreeNode root, String x) {
        if (root == null) return null;
        else if (x == root.getItem().getItem()) return root.getItem();
        else if (x < root.getItem().getItem())
                return search(root.getLeft(), x);
        else
                return search(root.getRight(), x);
}
```

삽입을 위한 아래 ① 부분을 완성하라. 새로 삽입되는 record는 항상 기존에 없던 새로운 것이라 가정하라. 가능하면 새 함수를 만들지 말 것.

```
void insert(String newItem, int blockNum) {
```

```
// newItem: 해당 record의 검색을 위한 string
// blockNum: 해당 record가 저장된 disk block#
                root = insertItem(root, newItem, blockNum);
}

TreeNode insertItem(TreeNode tNode, String newItem, int blockNum) {
// 수업 시간에 배운 것처럼 tNode.setLeft(...), tNode.setRight(...)를
// 사용하는 구조로 구현.


        ???        ---------------------------------------- ①


} // end insertItem
```