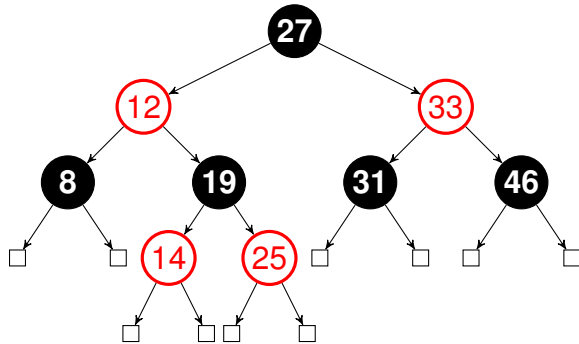


## 1 Balanced Search Trees

---

(a) Convert the red-black tree into a 2-4 tree.



(b) Insert 13 into the 2-4 tree.

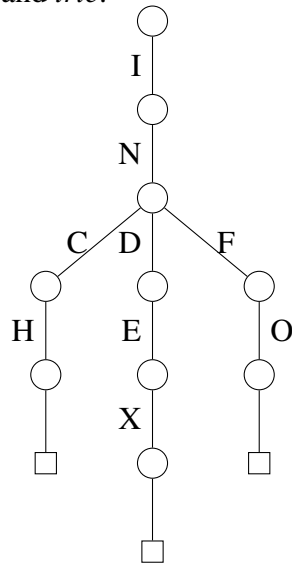
(c) Convert the resulting 2-4 tree into a valid red-black tree.

(d) Given a (2, 4) tree containing  $N$  keys, how would you obtain the keys in sorted order in worst case  $O(N)$  time? We don't need actual code—pseudo code or an unambiguous description will do (Final Fall '13).

(e) If a (2,4) tree has depth  $h$  (that is, the (empty) leaves are at distance  $h$  from the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree? (Final Spring '06)

## 2 Tries

First, list the words encoded by the trie. Then draw the trie after inserting the words *indent*, *inches*, and *trie*.



## 3 Runtime Analysis

(a) Give the best and worst case runtimes for method *A* in  $\Theta(\cdot)$  in terms of *N*.

```
public boolean A(int[] arr, int x) {
    //Assume arr is sorted; N is arr.length
    return A(arr, x, 0, arr.length-1);
}

public boolean A(int[] arr, int x, int low, int high) {
    if (low > high) return false;
    int mid = (low + high) / 2;
    if (arr[mid] == x) return true;
    return A(arr, x, low, mid-1) || A(arr, x, mid+1, high);
}
```

(b) Give the best and worst case runtimes for method *B* in  $\Theta(\cdot)$  in terms of *N*.

```
public int B(int[] arr) {
    //N is arr.length
    int count = arr.length - 1;
    while(count > 50) {
        count = count - arr.length / 50;
    }
    return count;
}
```