CS 32

Long Exam 3

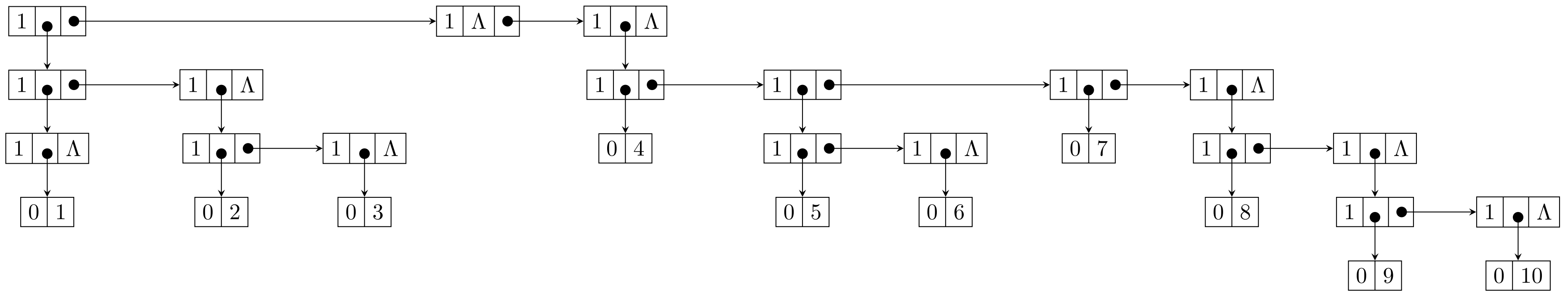
November 6, 2014

General Instructions

- Answer the items completely. Show your solutions/justifications when asked.
- Write as legibly as possible. Illegible or unreadable answers and solutions may not merit any points.
- Refrain from making unnecessary motions and sounds during the exam. Any suspicious behavior will be dealt with accordingly.
- Direct all questions to the proctor.
- If you need to go to the CR, hand your questionnaire, answer sheet, and scratch paper to the proctor before heading out. Only one person at any given time is allowed to go out.
- Once you're done with the exam (one way or the other), place your scratch papers and the questionnaire inside your blue book.

Questions

Consider the following generalized list L = (((1), (2, 3)), ( ), (4, (5, 6), 7, (8, (9, 10))))
1. Draw the schematic representation of a linked implementation of L using the LISP-based node structure with a tag bit.
ANSWER:



2. Create a linear list based on the pure list traversal performed on L. Represent it as a comma-delimited parenthesized list of elements.
ANSWER: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

3. Consider a function named $ACCESS\_LIST\_INDEX(L, i)$ that returns the i^{th} element of a generalized list. Create a pseudocode for $ACCESS\_LIST\_INDEX$ using only the *head* and *tail* functions for accessing generalized list elements. Assume also a function *length* exists that returns the length of a generalized list. You may also use other generalized list functions as needed. Additionally, make sure that the index i is valid, i.e. not below 1 or not greater than the list length, by printing out *"Index i is invalid"* and halting the execution of the program.
ANSWER:

```
function ACCESS_LIST_INDEX(L,i)
  counter ← 1
  list ← L
  if (i < 1) or (i > length(L)) then [output ‘Index i is invalid.’; return null]
  while counter < i do
    list ←tail(list)
    counter ←counter + 1
  endwhile
  return head(list)
end ACCESS_LIST_INDEX
```

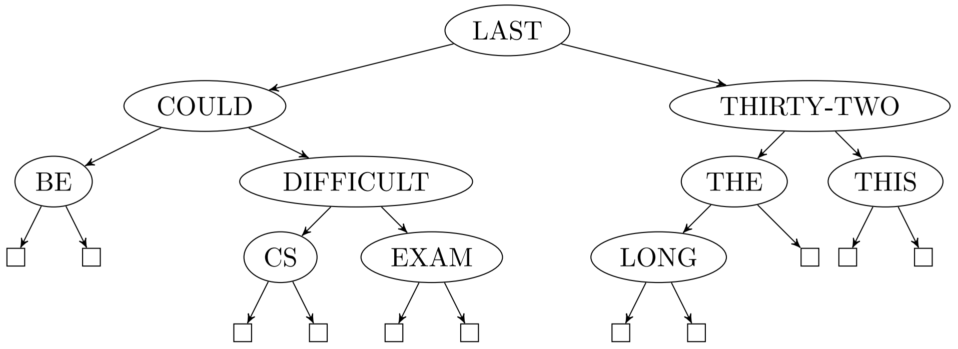
Consider the following keys to be inserted into an AVL tree:
THIS, COULD, BE, THE, LAST, DIFFICULT, CS, THIRTY-TWO, LONG, EXAM

4. Construct the AVL tree from the keys above by inserting them in the order that they appear. Show the status of the tree upon insertion of a key, and then after rotations (if any) to maintain height balance. If rotations are performed, just mention how the rotations and made and on what key.
ANSWER:

Steps in inserting the keys into the AVL tree are as follows:

(a) Insert THIS
(b) Insert COULD
(c) Insert BE
(d) Perform right rotation on THIS
(e) Insert THE
(f) Insert LAST
(g) Perform right rotation on THIS
(h) Insert DIFFICULT
(i) Perform right rotation on THE
(j) Perform left rotation on COULD
(k) Insert CS
(l) Insert THIRTY-TWO
(m) Perform right rotation on THIS
(n) Perform left rotation on THE
(o) Insert LONG
(p) Insert EXAM

Resulting AVL tree:



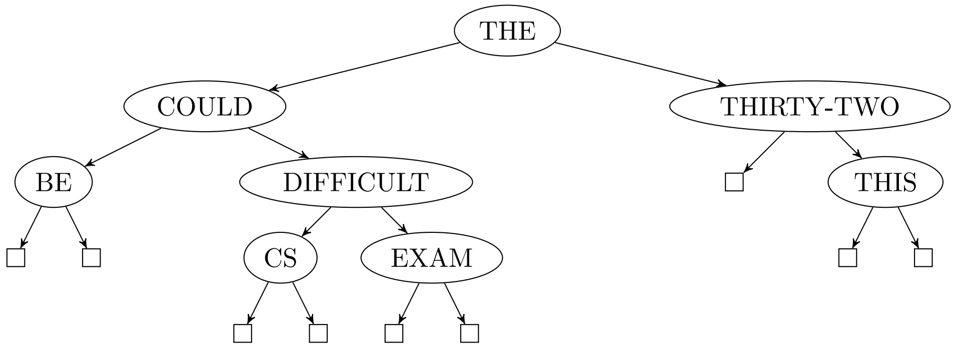
5. Show what happens to the AVL tree upon deletion of the root node twice successively. Show the status of the tree after deletion, after node succession, and whenever possible, after rotation/s.

   ANSWER:

   Steps in deletion are as follows:

   (a) Delete LAST
   (b) Assign LONG as new root
   (c) Delete LONG
   (d) Assign THE as new root

   Resulting AVL tree:



Scoring Mechanics

1. For Items 1 and 2: **0.1 point *deduction*** is given for each erroneous feature.
2. For Item 3:
   - **1 point**: Algorithm is correct and according to specifications.
   - **0.9 point**: Algorithm meets specification, and is correct in that in can return the i^{th} element of the list, but misses the invalid index checking (i.e. the if statement).
   - **0.5 point**: Algorithm is logically correct, but not according to specifications.
   - **0.25 point**: Significant attempt to answer, but algorithm is incorrect.
3. For Item 4: **0.1 point *deduction*** for each mistake in the steps. **An additional 0.1 point *deduction*** is given if the final AVL tree is erroneous.
4. For Item 5: **0.2 point *deduction*** for each mistake in the steps. **An additional 0.1 point *deduction*** is given if the final AVL tree is erroneous.