Problem Summaries

Lab 04

Notes about these notes

- These are not to be shared with those outside the class!
  - Especially if students take the class in the future!
- Keep in mind that there may be alternative, and sometimes better ways to approach these problems
- Particularly with the code – there are usually many ways to actually code up a solution

Lab 04 – Warehouse

- Use a map to save the total number of each item
  - Set value if first time entered, increment otherwise
- Output in order
  - First order largest to smallest in terms of value, then order alphabetically
  - Could create your own compare function
  - Or, could create pairs of (-number, name) and sort as usual (i.e. the ordered map would have it in order)
  - Other ways to compare, also.

Lab 04 – Beekeeper

- Read in word and check for pairs. To find pair, go character by character (starting from second character) and check:
  - Is character a vowel, and
  - Is character same as the one before it
  - Note: be sure to include ‘y’ as a vowel, according to the problem
- Important edge cases:
  - Handle words with just 1 character in them
  - Handle the case where the maximum word has 0 pairs
    - Since one word is guaranteed to have more than all others, this should be the only word, i.e. a single word is in the set, like “cat”

Lab 04 – Climbing Stairs

- Analysis problem – computation is simple.
- Go to office first
  - Either it is on the way to the registration desk, so it’s no extra walking
  - Or, it’s above the registration desk, so you’d have to go up there eventually anyway.
- Then go to the registration desk from office
- If you don’t have enough steps yet, then add steps (as a multiple of 2)
  - You can keep going down/up 1 to add 2 steps at a time
- Then travel registration desk down to ground floor.

Lab 04 – Nizovi

- A string processing algorithm – there is no significant trick to it.
  - Read through the string and output as you go, or build a string and output it.
- Need to keep track of details, though (i.e. program carefully!):
  - Maintaining indentation level throughout. Increase for ‘{‘, decrease for ‘}’
  - Keep the ‘,’ after the letters or } when you output
- Note: be sure to have a newline at the end of the last line of output.
  - You would have seen it as an error on one of the sample cases – the first four cases in the test data.

Lab 04 – OvalWatch

- Each leg will swap two elements.
- Begin with array of all numbers, where a[i]=i
- Then, go through legs from top to bottom
  - Can also do bottom to top, and this actually makes last step easier!
- Each leg swaps the values in a[i] and a[i+1]
- At end, for player i, a[i] is the character for player i. But, you need the player per character.
  - Can create a new array b[i] giving player corresponding to character i. Run through a[i] and set b[a[i]] = i. Then print b[i] at the end.
  - Or, if you went bottom to top to begin with, you have the answer already.

Lab 04 – Death and Taxes

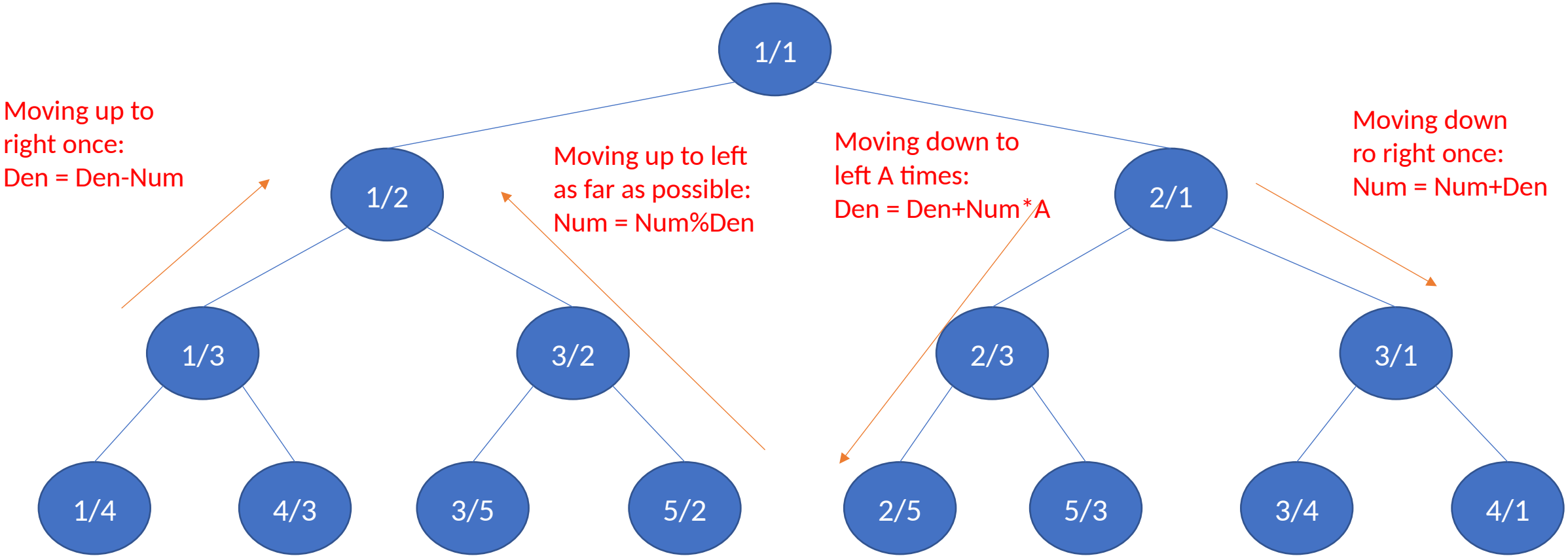
- Not at all a hard problem – very straightforward
- Just requires reading and implementing.
  - Keep track of # of shares and average cost, and update with each operation
  - Be sure to handle leftover shares when merging correctly.
  - At the end, determine IF there is profit, and if so, how much tax is owed, to get the final output
    - This is illustrated in the samples
- Main thing that could catch you: be sure to use enough precision during calculation and output
  - Intermediate precision can be higher than what’s needed at the end

Lab 04 – What's on the Grille?

- Mainly an implementation problem
  - Need to implement a "rotation" on the grid
- Only real "trick" is checking for validity
  - Need to make sure each element is hit once and ONLY once as the grille is rotated around
  - (If you have extra holes, some squares will be hit more than once!)
  - (Need exactly ¼ of the squares to be holes, and need arrangement to work)
- Create the rotations and then see what comes out of the process.

Lab 04 – A Rational Sequence

- Should write out the tree to see how it is structured
- Basically can determine how you would “walk” the tree from one node to the next, based on numerator and denominator
  - **Special case**: if denominator is 1, go to first one in next row
- Think about how you get from current node to next node:
  - Move up to left as much as possible, then over to right once (to sibling), then back down same number of steps
  - Each move up/down the tree just changes the numerator or denominator
- Move up to left A times
  - A = num/den (using integer division)
  - i.e. set num = num%den
- Then up to right once: den = den-num
- Then down to right once: num = num+den
- Then back down A times to left: den = den+num*A



Overall process to get to next node:

- Go up to left as much as possible (A times, where A is Num / Den)
- Go up to right once
- Go down to right once
- Go down to left A times