PS1Q2

Anthony Tam

January 20, 2017

Contents

1 Question 1

1.1 Part a) Consider the case where the player loses the most

The worst case return value is: $-4.99 This occurs when the first value tested is equal to 263. The probability of this occuring is:

$$\frac{1}{len(A)+1}$$

We could also find the probability with a limit.

$$\lim_{n\to\infty} \frac{1}{n}$$

This limit shows that as the size of the list increases, the pribability of this occuring approaches 0%.

1.2 Part b) Consider the case where the player wins the most

The best case ruturn value cannot be determined for any input size; it depends on the length of the list. It can be determined by the following formula.

$$len(A) - 500$$

The probability of the best case for any specific sized list can be determined by:

$$\frac{len(A)}{len(A)+1}$$

For find the overall probability for any length list, we can use the following limit:

$$\lim_{n} \to \frac{n}{n+1}$$

This shows that as the list increases in size, the probability of not having 263s a value appraoches 1.

1.3 Part c) Consider the average case, what is the expected value of the winnings of a player

Let x represent the size of the list. Let E(x) represent the expected number of wins with a list of size x
The sequence of wins will then look like the following

$$E(x) = \frac{i}{i+1} + \frac{i}{i+1} \cdot \frac{i+1}{i+2} + \frac{i}{i+1} \cdot \frac{i+1}{i+2} \cdot \frac{i+2}{i+3} ...$$

This simplifies to

$$E(x) = \frac{i}{i+1} + \frac{i}{i+2} + \frac{i}{i+3}...$$

In this case, the denominator is always the total number of possibilites for the last element of the list, so we can replace the denominator with x+1 (Number of possible outcoves for the length of the list)

$$E(x) = \frac{i}{x+1} + \frac{i}{x+1} + \frac{i}{x+1}...$$

Turning this into a series, we get

$$E(x) = \sum_{i=263}^{x} \frac{i}{x+1}$$

Since $x + 1$ can be seen as a constant in the sequence, it can be removed from the series block

$$E(x) = \frac{1}{x+1} \cdot \sum_{i=263}^{x} i$$

Now we have a simple series which can be reduced to the following form

$$E(x) = \frac{1}{x+1} \cdot \frac{x^2 + x - 68906}{2}$$

$$E(x) = \frac{x^2 + x - 68906}{2(x+1)}$$

1.4 Part d) Suppose that you are the owner of the casino and that you want to determine a length of the input list A so that the expected winnings of a player is between 1.01 and 0.99 dollars

For this to occur, the player would need to win between 399 and 401 times. We can plug this into the formula for expected value.

$$399 = \frac{x^2 + x - 68906}{2(x+1)}$$

Computing this statement will result in x≈877
The same also must be done for 401 wins

$$401 = \frac{x^2 + x - 68906}{2(x+1)}$$

Which results in a solution of x≈880
This means the length of the list should be between 887 to 880 to have an outcome of $-1.01 and $-0.99

2 Question 2

2.1 Part a) Given the index i of an element in the array, what are the indices of the left child, the middle child, the right child, and the parent of the element?

The left element can be found by:

$$(3 \cdot index) + 1$$

The middle element can be found by:

$$(3 \cdot index) + 2$$

The right element can be found by:

$$(3 \cdot index) + 3$$

The parent element can be found by:

$$floor(\frac{index - 1}{3})$$

2.2 Part b) How do EXTRACT-MAX and INSERT work for a ternary max-heap?

Extract Max:
Very similar to a binary heap, the only difference being when checking for the largest child object, we need to check all 3 children instead of 2.

Insert:
There are no differences, simply keep checking the parent element and bubble up.

2.3 Part c) Write the pseudo-code of a recursive implementation of IS-TERNARY-MAX-HEAP, explain its correctness, and its asymptotic upper-bound

2.3.1 Pseudo-Code

```
def IS-TERNARY-MAX-HEAP(A, i = 0):
```

```
'''
    Where right_child, middle_child, and left_child
    are the formulas from Part a)
'''
if left_child > len(A): # No left child, single node
   return True
elif right_child <= len(A):
   if A[i] > A[right_child] and A[i] > A[middle_child] and A[i] > A[left_child]:
      return IS-TERNARY-MAX-HEAP(A, right_child)
             and IS-TERNARY-MAX-HEAP(A, middle_child)
             and IS-TERNARY-MAX-HEAP(A, left_child)
      return False
elif middle_child <= len(A):
   if A[i] > A[middle_child] and A[i] > A[left_child]:
      return IS-TERNARY-MAX-HEAP(A, middle_child)
             and IS-TERNARY-MAX-HEAP(A, left_child)
      return False
elif left_child <= len(A) and A[i] > A[left_child]:
   return IS-TERNARY-MAX-HEAP(A, left_child)
return False
```

2.3.2 Correctness

- If there is no child to the node, a single node is always a valid heap, so return true.
- If we have a right child, check if all 3 children are less then the parent value. If so, return the result of the smaller heaps where the child is the parent. Otherwise return false.
- If we have no right child, do the same as above for the middle and left child.
- If we have no middle child, so the same as above for only the left child.

2.3.3 Asymptotic Upper-Bound:

- The function will check every node in the tree to ensure it is in a valid position. $\therefore$ O(n)

2.4 Part d) Write the pseudo-code of a iterative implementation of IS-TERNARY-MAX-HEAP, explain its correctness, and its asymptotic upper-bound

2.4.1 Pseudo-Code

```
def IS-TERNARY-MAX-HEAP(A):
   '''
       Where right_child, middle_child, and left_child
       are the formulas from Part a)
   '''
   foreach (i in A):
       if right_child <= len(A):
          if not (i > A[right_child] and i > A[middle_child] and i > A[left_child]):
             return False
       elif middle_child <= len(A):
          if not (i > A[middle_child] and i > A[left_child]):
             return False
       elif left_child <= len(A) and i <= A[left_child]:
          return False
```

2.4.2 Correctness

- If we have a right child, check if all 3 children are less then the parent value. If they are not, return false.
- If we have no right child, do the same as above for the middle and left child.
- If we have no middle child, so the same as above for only the left child.

2.4.3 Asymptotic Upper-Bound

- The function uses a foreach to cycle every element in the list. $\therefore O(n)$