Tutorial 4

Nov 2019

Q1 – Using stack to see if #0's > 1

```
def whichIsMore( input ):
"""
[input] is a string, where each character is '1' or '0'.
Return:
= 1 if there are more 1's than 0's
= 0 if there are more 0's than 1's
= -1 if there are equal number of 1's and 0's

Restriction: You are not allowed to count the numbers of 1's and
0's.
"""
```

- Cannot count, can only use a stack (to be declared in the function)
  - I.e, cannot do something like
    sum([int(i) for i in input]) > len(input)/2
    (or use a for loop to iterate and count the number of 1's)

- Idea – somehow let the 0's and 1's cancel out each other.
- Initialize an empty stack s
  - Stack has three useful operations top (let us see the first item), pop (deletes the top item and returns it) and push items to top of stack
- For each character c in input:
  - If s is empty OR if s.peek() == int(c):
    - Push c to the top of s
  - Else (if s.top() != c):
    - Delete the top item of s (pop)
- 3 possibilities at the end
  - If s is empty, it means equal numbers of 0 and 1
  - If s.top() == 1, what does this mean?
    - We can use this to deduce that one number has occurred more than the other
  - Else, …?

Some explanations

- For each character c in input:
  - If s is empty OR if s.top() == int(c):
    - Push c to the top of s
  - Else (if s.top() != c):
    - Delete the top item of s (pop)
- First, note that the number on the top of s (s.top()) contains the number which is the most frequent so far.
- When s is empty, 2 possibilities. Either we just started, or we encountered an equal number of 0's and 1's so far and they cancel each other out.
  - Either way, this results in the current number being the number which has been encountered the most so far

Q1 – Extra challenge

- After we have executed the function, can we use the resultant stack to answer this?

Q2 – Infix/postfix

- Define
  - Operands – the numbers
  - Operators - + - * / etc…
- To evaluate a postfix expression
  - Evaluate from left to right
  - Keep track of operands encountered until we encounter an operator
  - Once we encounter an operator, then use it on the operands most recently encountered
- Eg, 2 3 4 + *. From left to right
  - Encounter operands 2, 3, 4 before encountering operator +
  - Use + on the 2 most recent operands, so 3 + 4 = 7.
  - Next operand is *, so it becomes 2 * 7 = 14

Q3 – Queue using Stack

- Represent each patient as a 3-tuple of (Name, Arrival Time, Consultation Time).
- Patients sorted based on arrival time.
- Eg (John, 10, 5) means John arrives at time 10, and when he sees the doctor, he will take 5 mins.
  - Note that arrival time != time he meets the doctor
  - Eg if we have [(Mary, 8, 7), (John, 10, 5)], then
    - since Mary is the first patient, she sees the doctor at time 8 and leaves at time 15
    - When John arrives at time 10, he can only see the doctor at time 15 and leave at time 20

Some considerations

- Need a data structure to represent the queue. (Queue)
- Need to keep track of patients who have been seen (and have yet been seen)
- Keep track of time taken to see all patients
- To compute average queue length,
  - Determine the number of persons queuing at every time interval, and average this.
  - Eg if the queue lasts for 5 secs, and the number of people at the queue at every sec is 1, 3, 2, 1, 0, then the average queue length is (1+3+2+1+0)/5
- Similarly, average waiting time is the sum of all waiting times of a patient/number of patients

Pseudocode sketch
- Variables needed

```
def simulateQueue( patients ):
        """
        [patients] is a list of patients stored in order of arrival time.

        This function simulate a clinic with 1 doctor and print out events happening, e.g. patients
arriving, queueing up, consulating, leaving etc.

        This function also returns a 2-tuple (avg. queue length, avg. waiting time) at the end.
        """
        #for accessing the patient tuple, can use namedtuple for advanced coder
        NAME = 0
        ATIME = 1 #arrival time
        CTIME = 2 #consult time

        waitQ = QueueLinkedList()
        time = 1
        pInConsultation = None
        pIdx = 0
        done = False
        avgQLen = avgWTime = 0
```

- while not done,
  - Before we do anything, check if we are currently seeing a patient.
    - If there is a current patient being consulted, check if he's done and remove him if possible.
  - Then, check to see if there are any patients to add to our queue. (if we have reached the next patient arrival time)
  - If Queue not empty
    - No patient in consultation
      - Get the first patient, let him be consulted.
  - If we have iterated through all the patients, set done = True and terminate
  - Else, increment time by 1

Computing Time

- Average Queue Length
  - At every time step, get the size of the queue (after inserting and removing patients) and add it to variable avgQLen
  - When we are done, the Average Queue Length is avgQLen/time
- Average Waiting Time
  - The waiting time for a given patient is given by (time he is served – arrival time)
  - When we serve a patient, compute his waiting time and add it to avgWtTime
  - When we are done, get average wait time by avgWtTime/len(patients)

Question 4

- 1, 2, 3, 4, 5, 10, 9, 8, 7, 6 – ok
  - When user enters 1,2,3,4,5, step 2 prints it out immediately
  - 6,7,8,9,10 are stored in a stack in step 2 also.
  - In step 3, the stack can output all its contents. Since its LIFO, will get 10,9,8,7,6
- 6, 5, 7, 4, 8, 3, 9, 2, 10, 1 – ok
  - Initially, 1, 2, 3, 4, 5 stored in stack
  - When input 6, immediately output it
  - Output 5 from stack (stack now contains 4,3,2,1)
  - Output 7 immediately
  - Output 4 from stack (stack now – 3,2,1)
  - Output 8 immediately
  - Output 3 from stack (stack – 2, 1)
  - Output 9 immediately
  - Output 2 from stack
  - Output 10 immediately, then output 1 from stack

- 8, 6, 7, 5, 4, 3, 9, 10, 2, 1 – No
  - Suppose its possible.
  - Then only way for the first output to be 8 is that 1-7 is stored in the stack.
    - Cannot store 8 in the stack as when we are processing 9, we either have to output 9 immediately or save it to the stack.
  - Then the next output must be 7 (output from stack) or 9 (the next input).
  - But this isn't the case, hence proven by contradiction this is false.

Q5

- 6, 7, 8, 1, 2, 3, 9, 10, 4, 5 – Possible (try to trace yourself)
- 2, 1, 4, 3, 6, 5, 8, 7, 10, 9 – Possible
- 5, 1, 7, 10, 2, 9, 3, 4, 6, 8 – No. Instead of tracing let's reason this.
  - A queue is FIFO.
  - Hence, output of the queue must be in ascending order.