EECS268:Lab7

Navigation

Home

Information

Syllabus
Schedule
Lecture Archive

Classwork

Labs
Submitting Work

Contents

Due time

This lab is due one week from the start of your lab.

Lab conduct

- Do not use any unauthorized aid, such sites like rentacoder or chegg to obtain answers
- Do not use code provided by another student
- Do not reuse code (by you or anyone) from prior semesters
- If you need help, seek it from:
  - Your lab TA.
  - ACM Tutoring
  - Me, Dr. Gibbons, my email is jwgibbo@ku.edu / jwgibbo@gmail.com
- If equipment you don't own (e.g. cycle servers) needs attention or you're having account issues put in a ticket! (https://tsc.ku.edu/request-support-engineering-tsc)

Overview

In this lab we will time various methods from the data structures we've made so far, save those times, and plot them to see if they match the theoretical expectations.

Timing a block of code

To handle the timing we'll be wanting to track the time our process spent in the CPU (as oppose to clock-on-the-wall time).

Luckily, python's time module can easily help with this.

Here's an example of timing a loop that has one-hundred million iterations.

```
def nanosec_to_sec(ns):
    BILLION = 1000000000
    return ns/BILLION

print("Beginning the timing code...")
num_iterations = 100000000 #one hundred million
start_time = time.process_time_ns()

#loop that does nothing but repeat, hence the _ variable and the keyword pass
#which just means do nothing
for _ in range(num_iterations):
    pass

end_time = time.process_time_ns()

print("Total time in ns: ", end_time-start_time)
print("Total time in sec: ", nanosec_to_sec(end_time-start_time))
```

Notes on time.process_time_ns():

- It returns an int that represents the current number of nanosecond the process has spent in the CPU
- If you want seconds, you'll need to convert it, like I did in the example

Requirements

You will time several methods from Stacks, Queues, and Lists using several values for n (size of the data structure)

Data sets

You will fill each data structure with an increasing number of elements and record a time for each. For each method, start with a data size of 1000 then increase by 1000, recording another time, and repeat until you've reached 100,000 elements

For example. Let's say I want to time Pop for a Stack. I would do the following:

1. Fill a stack with 1000 elements
2. Record time to perform a single pop with that size stack
3. Fill a stack with 2000 elements
4. Record a time to perform a single pop with that size stack
5. Repeat these steps, increasing the size by 1000 each time until I've recorded a time for a stack with 100,000 elements in it

Operations to time

Operation
Popping a single item from a stack
Popping all items from a stack
Queue's enqueue
Linked List get_entry at specifically index 0
Linked List get_entry at specifically the last index
Printing all elements in a LinkedList using get_entry

Plotting

In addition to your code, you will submit graphs (e.g. excel graphs) of the data you've collected. One graph for each operation.

Rubric

- [10pts] No crashing or unhandled exceptions
- [10pts] Code design (no giant mains!)
- [5pts] Documentation
- [10pts] Pop single item data
- [10pts] Pop all items data
- [10pts] Enqueue data
- [10pts] get_entry at index 0 data
- [15pts] get_entry at last index data
- [20pts] printing entire list using get_entry data

Submission instructions

Send a tarball/zip with all .py files and plots (e.g. excel file)