CSE 6140 Assignment 1
due Sept. 15, 2016 at 11:55pm EDT on T-Square

Please upload 1) a PDF with solutions of Problems 1, 2 and 3; 2) a PDF of your report for Problem 4; 3) a single zip file of your code, README, results for Problem 4.

1 Simple Complexity

1. For each pair of functions f and g, write whether f is in $\mathbb{O}(g)$, $\Omega(g)$, or $\Theta(g)$.

   (a) $f = (n + 1000)^4$, $g = n^4 - 3n^3$

   (b) $f = \log_{1000} n$, $g = \log_2 n$

   (c) $f = n^{1000}$, $g = n^2$

   (d) $f = 2^n$, $g = n!$

   (e) $f = n^n$, $g = n!$

   (f) $f = \log n!$, $g = n \log n$ (hint: if uncertain, Stirling's a certain)

2. Determine the Big-O time complexity for the algorithm below (**show your work**). Also, very briefly explain (in one or two sentences) what the algorithm outputs (note: the % symbol is the modulo operator):

```
   Data: n
 1 i = 1;
 2 while i ≤ n do
 3 |   j = 0;
 4 |   k = i;
 5 |   while k % 3 == 0 do
 6 |   |   k = k/3 ;
 7 |   |   j++ ;
 8 |   end
 9 |   print i,j;
10 |   i++;
11 end
```

2 Greedy - Why is the pool always so busy anyway?

Georgia Tech is trying to raise money for the Technology Square Research Building and has decided to host the "Tech Swim Run Bike" (TSRB) Triathalon to start off the fundraising campaign!

Usually in a triathalon all athletes will perform the three events in order (swimming, running, then biking) asynchronously, but unfortunately due to some last minute planning, the race committee was only able to secure the use of one lane of the olympic pool in the campus recreation center. This means that there will be a bottleneck during the first portion (the swimming leg) of the race where only one person can swim at a time. The race committee needs to decide on an ordering of athletes where the first athlete in the order will swim first, then as soon as the first athlete completes the swimming portion the next athlete will start swimming, etc. The race committee, not wanting to wait around for an extremely long time for everyone to finish, wants to come up with a schedule that will minimize the time it takes for everyone to finish the race. Luckily, they have prior knowledge about how fast the athletes will complete each portion of the race, and they have you, an algorithm-ista, to help!

Specifically, for each athlete, i, they have an estimate of how long the athlete will take to complete the swimming portion, s_i, running portion, r_i and biking portion, b_i. A schedule of athletes can be represented as a list of athletes (e.g. $[athlete_7, athlete_4, ...]$) that indicate in which order the athletes will start the race. Using this information, they want you to find an ordering for the athletes to start the race that will minimize the time taken for everyone to finish the race, assuming all athletes perform at their estimated time. More precisely, give an efficient algorithm that produces a schedule of athletes whose completion time is as small as possible and prove that it gives the optimal solution using an exchange argument.

Keep in mind that once an athlete finishes swimming, they can proceed with the running and biking portions of the race, even if other athletes are already running and biking. Also note that all athletes have to swim first (i.e. some athletes won't start off running or biking). For example: if we have a race with 3 athletes scheduled to go in the order $[2, 1, 3]$, and the finishing time for athlete i is represented as a_i, then the finishing times would be as follows: (with a total finishing time of $\max(a_1, a_2, a_3)$)

$$a_2 = (s_2) + r_2 + b_2$$
$$a_1 = (s_2 + s_1) + r_1 + b_1$$
$$a_3 = (s_2 + s_1 + s_3) + r_3 + b_3$$

3 Divide and Conquer

You're given a set of n high-tech tags. All tags look exactly the same, but embedded in each is a chip with an ID t_i. If two tags i and j have the same ID $t_i = t_j$, then they light up when tapped against each other (and turn back off

once they're separated).

Your task is simple: Given n of these tags, is there a collection of more than $n/2$ of them with the same ID? Note: there's no way of figuring out the tag's actual ID. Your only option is to tap two tags against each other and see whether they light up. Each tap costs one "operation", and you can only compare two tags at a time.

Divise a Divide and Conquer algorithm that requires operations on the order of $O(n \log n)$ to execute. In particular, write down the pseudocode, briefly explain why it works, and formally show that it's complexity is $O(n \log n)$.

4 Programming Assignment

You are to implement *either* Prim's or Kruskal's algorithm for finding a Minimum Spanning Tree (MST) of an undirected graph, and evaluate its running time performance on a set of graph instances. The 12 input graphs are RMAT graphs [1], which are synthetic graphs with power-law degree distributions and small-world characteristics. These graphs might have multiple arcs between the same vertices, with each of them having an associated weight (i.e. the graphs are multigraphs), your code should take this into account.

4.1 Static Computation

The first part of the assignment entails coding either Prim's or Kruskal's algorithm to find the cost of an MST given a graph file. You may use the programming language of your choice (either C++, Java, or Python). We provide a wrapper function in all three languages to help you get started with the assignment. You may call your own functions inside the wrapper. We also have implemented a timer in the wrapper that records the running time of your algorithms. To implement these algorithms, you may make use of data structure implementations in the programming language of your choice; e.g. in python, the heapq library may be used for implementing priority queues and set operations may be used for implementing the union-find data structure. In Java, java.util.PriorityQueue may be used for implementing priority queues and java.util.Set may be used for set operations.

The 'graph file' format is as follows:

> Line 1: N E (N = number of vertices, E = number of edges)
>
> Every subsequent line contains three integers: u v weight (u = source of edge, v = destination of edge, weight = weight of edge between u and v)

The MST calculation is to be implemented in the `computeMST` function, as indicated in the wrapper code.

4.2 Dynamic Recomputation

The next part of the assignment requires you to update the cost of the MST given new edges to be added to the graph. You are provided with a 'changes file' and the format is as follows:

Line 1: N (N = number of changes/edges to be added)

Every subsequent line contains three integers: u v weight (u = source of new edge to be added, v = destination of edge, weight = weight of edge between u and v)

You are to implement the function `recomputeMST` as indicated in the wrapper code that computes the new MST given the new edge to be added into the graph. You are responsible for maintaining the old MST before recomputing.

Note: it is very easy to complete this part of the assignment by simply adding the new edge and calling your old `computeMST` function from part 1 (Subsection 4.1). The objective is to minimize computation and efficiently recompute the cost of the MST.

4.3 Execution

The wrapper code is set up to require three command line arguments: <executable> <graph_file.gr> <change_file.extra> <output_file>. The <graph_file> is the one described in Subsection 4.1 and the <change_file> is the one described in Subsection 4.2.

Note: You do not have to use the wrapper code provided; however, if you write your own code from scratch, please indicate so in your README (how to compile and run your code)

4.4 Experiments

You are required to run your code for all 12 input graphs provided. The wrapper functions we provide in C++, Java, and Python describe the following procedure. For each graph:

- parse the edges (`parseEdges`): **to be implemented**
- compute the MST using either Prim's or Kruskal's algorithm (`computeMST`): **to be implemented**
- write the cost of the initial MST and time taken to compute it to the output file: **provided in wrapper code**
- For each line in the <change_file>,
  - Parse the new edge to be added: **provided in wrapper code**
  - Call the function `recomputeMST`: **to be implemented**
  - Write to the output file the cost of the new MST and the time taken to compute it: **provided in wrapper code**

Name the output files as such: <graph_file>_output.txt and place them in the folder *results*. You should have one output file for each of the 12 input graphs provided.

4.5 Report

Write a brief report wherein you:

a) List which data structures you have used for your choice of algorithm (Prim's/Kruskal's). Explain the reasoning behind your choice and how that has influenced the running time of your algorithms, and the theoretical complexity (i.e., specify the big-Oh for your implementation of each of the two algorithms - computeMST and recomputeMST).

b) Plots: We require you to make two plots. Plot the running time as the number of edges in the graph increases (across the 12 graphs you were given) for both the static and dynamic calculations, i.e. one plot showing on the x-axis the number of edges the graph has and the y-axis the time for the static MST calculation, and another plot where the y-xis the time needed to insert 100 edges (as given the changes files) using your recomputeMST code. Discuss the results you observe. How does the empirical scaling you observe match your big-Oh analysis? How does the behavior vary with the dynamic recomputation?

4.6 Deliverables

- code for initial, static MST implementation (this is called computeMST in the provided wrapper code)
- code for dynamic MST recomputation (this is called recomputeMST in the provided wrapper code)
- README, explaining how to run your code
- Report (Subsection 4.5)
- output files (12) within the *results* folder - one for each graph

References

[1] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-MAT: A Recursive Model for Graph Mining. In *Proc. 4th SIAM Intl. Conf. on Data Mining,* Florida, USA, April 2004.