Evan Edelstein

EN.605.621.84.FA25

HW 4

The work in this exercise is mine alone without un-cited help. No AI was used to answer these questions.

1. Pseudocode:

Function print\_in\_out\_degree(G,):

in\_degree = zero initialized array of length G.adj

out\_degree = zero initialized array of length G.adj

For i in G.adj:

d = 0

For j in G.adj[i]:

d++

in\_degree[j]

out\_degree[i] = d

for i in G.adj:

print(“node: ”, i)

print(“in-degree: “, in\_degree[i])

print(“out-degree: out\_degree[i]

This algorithm performs 2 loops (line 4,10) over the adjacency array contributing to a runtime of: . Over the runtime of the algorithm each edge will be probed at least once by the inner loop in line 6, leading to a total cost of O(m). Combining the two terms we see this algorithm has a runtime of Θ (n+m).

1. Since the pipe is straight (Figure 1), we only need to consider the difference in longitudinal value (y) between the wells. In this way we can think of all the wells as being on single vertical line (Figure 2) and reframe the problem as searching for a point on this line that has the minimum distance to all the points. We can use the median of all the wells y axis value (or height on the imaginary line) to determine the horizonal line that minimizes the north/south pipeline length. This is because, by definition, the median is the point that is closest to all the other points.

A diagram of a transmission tower

AI-generated content may be incorrect.

Figure 1: Problem construction from textbook

A tower with a pole

AI-generated content may be incorrect.

Figure 2: Mock projection of 4 wells onto singe x-axis.

* 1. The Select algorithm from the textbook can be used to find the median of an array of the wells y-coordinates in O(n) time. For an array with an odd number of wells (n), we would select for the (n+1)/2 order statistic and for even we select either the (n/2) or (n+1)/2 order statistic. This is accomplished using the median-of-median construction, like quicksort, but without sorting the well coordinates (which wouldn’t be linear).
  2. The worst-case running time of the operations MultiPopA is O(n), since at worst the entire stack A is popped (when k >= n) and each pop operations is O(1). The same applies to MultiPopB, which has the worst-case runtime of O(m). The worst case for Transfer is when the entire Stack A is popped and then pushed onto stack B, which results in n pushes and n pops, each of which are O(1). In this case the runtime would be O(2n) = O(n).
  3. We need to define a potential function that “saves” up for the 2 operations in the Transfer. That is for every item pushed to Stack A we must make sure we can pay off a future pop and push to stack B for that item. Thus, we should define a potential function such that we account for the two operations per element in Trannsfer. The amortized cost for each function after i operations can be defined by:

The following table shows how each operation has an amortized cost of O(1). Note we use x to be equal to the minimum of n and k, and y to be the minimum of k and m.

|  |  |  |
| --- | --- | --- |
| Function | Actual Cost |  |
| PushA | 1 |  |
| PushB | 1 |  |
|  |  |  |
| MultiPopA | min(n,k) = x |  |
| MulitPopB | min(m,k) = y |  |
|  |  |  |
| Transfer | min(n ,k) = x |  |