Written Assignment #4

Vancouver Summer Program 2019 – Algorithms – UBC

- You should work with a partner.
- You must typeset your solutions.
- Submit your work using Gradescope by **10:00 p.m. on Monday, August 6**.
- **Notation.** $\mathbb{N} = \{1, 2, \dots\} \subset \{0, 1, 2, \dots\} = \mathbb{Z}_+$, and $\mathbb{R}_+ = [0, \infty)$.

---

1. You went on a world trip in which you visited n countries, staying one day in each country. You were on a very low budget, however, so you decided to work in every country you visit to cover your daily expenses. Suppose that in country $i \in [n]$ you earned e_i dollars, but spent s_i dollars. You might have made more than what you spent on that day, but it is possible that you might have spent more than what you made (in which case you pulled some cash from your initial budget). Now, after the trip had concluded, you are curious about the contiguous sequence of countries along the trip in which you maximized your total net profit (your profit per day is the amount you made minus the amount you spent). Describe a $\Theta(n \log n)$-time algorithm that solves this problem using the divide-and-conquer approach.

   Solution. This problem can be abstracted as follows. We may assume that we are given as input an array $A[1 \dots n]$, where $A[i] = e_i - s_i$. Then the problem boils down to determining the maximum sum that can be achieved by considering any contiguous subarray of A. We can use $A[i \dots j]$, $j \geq i$, to represent the contiguous subarray that starts with $A[i]$ and ends with $A[j]$ and the subarray sum for $A[i \dots j]$ is $\sum_{k=i}^{j} A[k]$.

   As a base case, if there were only two or fewer days then the problem can be solved easily. If the array is of length more than 2 then we can split the array into two pieces (of approximately equal length) and recursively solve the subproblems. This yields three cases:

   (a) The solution is in the first portion of the array;

   (b) The solution is in the second portion of the array;

   (c) The solution spans the two pieces of the array.

   The interesting case is the third case, when we need to compute the maximum subarray sum that spans the two pieces of the array. Computing the maximum subarray sum that includes the two pieces can be done in $\Theta(n)$ time. A procedure for this is outlined below (although the code description is not required to receive credit):

```
MAXCROSSINGSUM(A[1...n], l, m, h)
⟪Include elements on left of mid⟫
  sum ← 0
  left_sum ← −∞
  for i ← m down to l
      sum ← sum + A[i]
      if sum > left_sum
          left_sum ← sum

⟪Include elements on right of mid⟫
  sum ← 0
  right_sum ← −∞
  for i ← m + 1 to h
      sum ← sum + A[i]
      if sum > right_sum
          right_sum ← sum
  return left_sum + right_sum
```

The running time for the algorithm is described by $T(n) \leq 2T(n/2) + \Theta(n)$, which yields $T(n) \in O(n \log n)$.

Proof of correctness: A simple inductive proof is all that is needed. (Although I am skipping the proof, it is required for full credit.)

Note: There is a linear-time dynamic-programming algorithm for this problem.

2. What does **Algorithm 1** below do? The initial call to the algorithm is $F(A)$, where A is an $n \times n$ 0-1 matrix. Analyze the running time of this algorithm and express your answer in $O(\cdot)$. You may assume that matrix multiplication can be done in $O(n^{2.807})$ using Strassen's algorithm. You might want to see what the algorithm outputs on small examples.

   Solution. This algorithm receives as input the adjacency matrix of an undirected graph on n nodes, and computes the lengths of unweighted shortest paths between every pair of nodes (i.e., it is an all-pairs shortest path algorithm.) That is, after the algorithm terminates, the output matrix D is such that $d_{i,j}$ is the number of edges in any shortest path between vertices i and j. In general, this algorithm runs in time $O(M(n) \log n)$, where $M(n)$ is the time required to multiply two $n \times n$ matrices, and with Strassen's matrix multiplication algorithm, the running time is $O(n^{2.807} \log n)$. This algorithm is due to Raimund Seidel, and the following is the original paper that contains all the analysis: R. Seidel. On the All-Pairs-Shortest-Paths Problem in Unweighted Undirected Graphs, J. Comput. Syst. Sci., 51 (3) (1995), pp. 400-403.

3. Suppose two nodes in a distributed system have clocks that gradually drift apart by 1 second every 100 seconds. Consider a resynchronization mechanism that is invoked periodically to eliminate the skew. If the resynchronization is performed every 200 milliseconds, can we be certain that the skew between the two clocks is no more than 10 milliseconds? Skew is the difference in time between the two clocks.

4. Consider the events depicted in the following timeline:

   (a) Identify pairs of events that are logically concurrent. (In other words, in the absence of a global clock and tight time synchronization, which events cannot be distinguished as having occurred at different times?)

Algorithm 1 $F(A)$

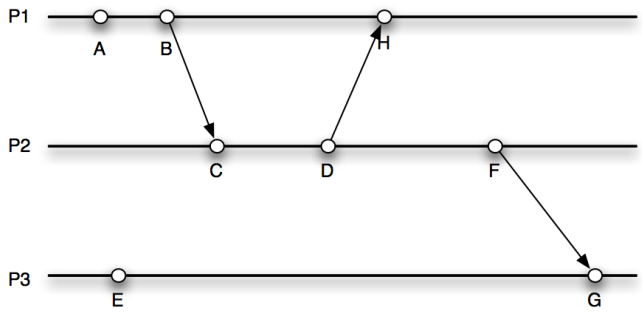
$Z \leftarrow A \cdot A$

Let B be an $n \times n$, 0-1 matrix, where

$$b_{i,j} = \begin{cases} 1 & \text{if } i \neq j \text{ and } (a_{i,j} = 1 \text{ or } z_{i,j} > 0) \\ 0 & \text{otherwise} \end{cases}$$

if $\forall i, j \; i \neq j, b_{i,j} = 1$ **then**
  return $D \leftarrow 2B - A$
end if
$T \leftarrow F(B)$
$X \leftarrow T \cdot A$
return $n \times n$ matrix D, where

$$d_{i,j} = \begin{cases} 2t_{i,j} & \text{if } x_{i,j} \geq t_{i,j} \cdot \sum_{k=1}^{n} a_{j,k} \\ 2t_{i,j} - 1 & \text{otherwise.} \end{cases}$$



(b) Specify Lamport logical timestamps and vector clock timestamps for each event.