Written Assignment #2

Vancouver Summer Program 2019 – Algorithms – UBC

- You should work with a partner.
- You must typeset your solutions.
- Submit your work using Gradescope by **5:00 p.m. on July 26**.
- **Notation.** $\mathbb{N} = \{1, 2, \ldots\} \subset \{0, 1, 2, \ldots\} = \mathbb{Z}_+$, and $\mathbb{R}_+ = [0, \infty)$.

---

1. **Path Finding in an Hourglass Map.** In an hourglass map (example below) a path is marked. A path always starts at the first row and ends at the last row. Each cell in the path (except the first) should be directly below to the left or right of the cell in the path in the previous row. The value of a path is the sum of the values in each cell in the path.

   A path is described with an integer representing the starting point in the first row (the leftmost cell being 1) followed by a direction string containing the letters L and R, telling whether to go to the left or right. For instance, the path below is described as 3 RRRLLRRRLR.

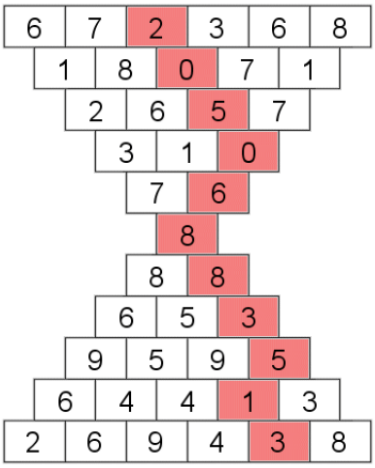

Figure 1: Example of a path in an hourglass

   Describe an algorithm to find the path of maximum value through the hourglass. We shall use n to denote the number of elements in the first and last row. An algorithm with worst-case running time in $\Theta(n^2)$ exists and your algorithm should have similar running time.

   Derive the runtime complexity of the algorithm and prove its correctness.

   Solution. Suppose that we started our path at row i column j (assuming the rows are numbered starting from the top and that the columns are numbered starting from the left). We will number the top-most row as row $-n+1$ where n is the number of elements in that row. The row numbers increase until row $n-1$. Row i has $|i|+1$ elements. We will use $H(i,j)$ to represent the value of cell (i,j). Let $v(i,j)$ denote the maximum value that can be obtained if a path starts at cell (i,j).

The dynamic programming equation can be stated as follows:

$$v(i,j) = \begin{cases} \max\{v(i+1,j-1)+H(i,j), v(i+1,j)+H(i,j)\} & i < n, 1 \le j \le |i|+1 \\ 0 & i \ge n \\ 0 & j < 1 \\ 0 & j > |i|+1 \end{cases}$$

What we seek is $\max_{1 \le j \le n}\{v(-n+1,j)\}$.

Using memoization, this problem can be solved in $O(n^2)$ time.

2. Consider n non-preemptable jobs $J_1, \ldots, J_n$ that are all available (and ready to run) at time 0. Job J_i requires worst-case execution time $e_i \in \mathbb{N}$ and worst case (peak) memory requirement $M_i \in \mathbb{N}$ (in MB). Once assigned to processors, jobs cannot migrate to other processors. Moreover, the order of the jobs is fixed by the job dispatching system and so cannot be rearranged: In any allocation of processors to jobs, the jobs must appear in the same order as they are given in the input. Assume that the jobs are ordered according to their indices such that J_1 is the first job to be assigned, and if $i < j$ and J_i is assigned processor k, then job J_j must either execute on processor k and start after J_i, or execute on the $(k+1)$st processor. The processors onto which the jobs are to partitioned are identical and each has maximum available time $L \in \mathbb{N}$; that is, the sum of execution times of the jobs assigned to any processor cannot exceed L. You may assume that $e_i \le L$ for every $i \in \{1, \ldots, n\}$.

   Given an allocation of processors to the input jobs, the peak memory demanded by processor π_j under the given allocation is defined as $\max\{M_i : J_i \text{ is assigned to processor } \pi_j\}$. Our goal is to assign the jobs to as many processors as needed, respecting the given processor available time and the no rearrangement constraint, so that the overall peak memory usage is minimized; that is, the sum of the peak memory used on all processors is kept at a minimum. Develop an algorithm to solve this problem optimally and prove its correctness. Derive the asymptotic running time of your algorithm in terms of n, e_i, M_i, and L (or any subset thereof).

   Solution. We will do this with dynamic programming. Think about the way in which the jobs are placed on the processor. We can iteratively place the jobs on the processor where, at each step, we can make a decision to either place the job on the current processor, or to open a new processor. Sometimes, of course, we will have no choice: The processor may not be able to fit the job, and we would then be forced to open a new processor. We will solve this problem first by going backwards. We will loop from n down to 1 and determine the cheapest way (in terms of memory) of placing jobs i through n if we start a new processor with job i. We store the best costs in an array cost, where $\mathsf{cost}[i]$ is the best possible memory cost of placing jobs i through n. The recursion is the following:

$$\mathsf{cost}[i] = \min_{1 \le k \le m} \left\{ \mathsf{cost}[i+k] + \max\{M_i, \ldots, M_{i+k-1}\} \right\},$$

   where m is the maximum integer such that $\sum_{j=i}^{m} e_j \leqslant L$. The base case is $\mathsf{cost}[n] = M_n$. This recurrence can be converted to an iterative algorithm that runs in time $O(n^2)$, which is *strongly* polynomial.