Problem Set 5

All parts are due Tuesday, November 24, 2015 at 11:59PM. Please download the .zip archive for this problem set. Remember, your goal is to communicate. Full credit will be given only to a correct solution which is described clearly. **Convoluted and obtuse descriptions might receive low marks, even when they are correct.** Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A

Problem 5-1. [25 points] **Escape with Ben's Life!**

Ben Bitdiddle has found himself trapped in a video game and needs your help to find a way out! To escape, he must navigate from one corner of a $n \times n$ 2-D grid at the coordinate $(1, 1)$, to the opposite corner at the coordinate (n, n) where n can be arbitrarily large. He can only move either up, down, left, or right, to an adjacent integer coordinate.

Ben has some finite *integer* amount of health, M, and must reach his destination without it dropping to or below 0. The bad news is, every single coordinate (x, y) in the grid except the start contains a peril $p(x, y)$ which deals a *positive integer* damage to Ben as he moves into that coordinate.

Fortunately, Ben already knows every single $p(x, y)$, and just needs your guidance to escape. Give an efficient algorithm that provides Ben with a path that he can use to escape (or report that he's trapped for good). But beware! Time is of the essence as you must provide an algorithm that runs in $O(Mn^2)$ time or else Ben will be stuck in the game for good!

Problem 5-2. [25 points] **Escape with Ben's Life 2: Electric Boogaloo!**

Thanks to your valiant algorithm-ing, you helped Ben escape from his virtual prison ... only for Ben to somehow get trapped again. For the most part, the game remains the same as it did in the previous problem, with Ben starting with M health and needing to traverse from the corner $(1, 1)$ of a game grid to another corner (n, n). This time, however, each peril $p(x, y)$ can deduct any *positive real* amount of health.

To help Ben on his way, though, we've also hacked K different health packs into the game that fully restore his health, where $K \leq n$. Thus, now every coordinate in the $n \times n$ game grid contains either a peril that deducts health, or a health pack that fully restores it to M. Ben knows the locations of these health packs.

Again, give an efficient algorithm that provides Ben with a path that he can use to escape (or report that he's trapped for good). This time, your algorithm should run in $O(Kn^2 \log n)$ time.

Part B

Problem 5-3. [50 points] **Whimsical Numbers**

Let $A = \{a_1, a_2, \ldots, a_k\}$ be a set of positive integers. We say that a non-negative integer ℓ is a whimsical number if it can be written as a sum of (zero or more) elements from A (where repetition is allowed).[1] For example, if $A = \{5, 8, 11\}$, then $16 = 5 + 11 = 8 + 8$, and $18 = 5 + 5 + 8$, are whimsical numbers, while 17 is not a whimsical number. 0 is always a whimsical number.

Your goal is to create a program that, given the set A as an input, decides whether each queried number ℓ is a whimsical number. More formally, you must write a class `WhimsicalChecker` which includes the following methods.

- A constructor `__init__(self, A)`, where A represents the set A specified above. Note that A is given as a Python list, containing k distinct positive integers. Let M be an upper bound on the elements in A; that is, $a \leq M$ for every $a \in A$. Your constructor should perform any necessary precomputation within $O(kM \log(kM))$ time.
- A method `is_whimsical(self, l)`, which takes a non-negative integer ℓ as an argument. This method must return `True` if ℓ is a whimsical number, and return `False` otherwise. Your `is_whimsical` method should run in constant time. In particular, note that ℓ can be extremely large.

Your program will be tested with parameters $k \leq 5{,}000$, $M \leq 50{,}000$ and each $\ell \leq 1{,}000{,}000{,}000$. The method `is_whimsical` may be called up to $10{,}000$ times in a single test. Note that in your analysis, you must still treat k, M and ℓ as variables, not constants.

(a) [5 points] **Prove some useful claims**

Prove the following claims. A few sentences should suffice for each claim.

Claim 1. If ℓ is a whimsical number and $a \in A$, then for any $\ell' > \ell$ such that $\ell' \bmod a = \ell \bmod a$, ℓ' is also a whimsical number.

Claim 2. Fix a number $a \in A$. For each $r \in \{0, 1, \ldots, a-1\}$, let x_r be the smallest whimsical number such that $x_r \bmod a = r$. (Note that x_r may not exist for some r.) Consider a non-negative integer ℓ and let $r' = \ell \bmod a$. Then, ℓ is a whimsical number if and only if $x_{r'}$ exists and $\ell \geq x_{r'}$.

(b) [10 points] **Set up a graph problem**

Referring back to Claim 2, if we can find the values x_r's, then we will have a formula for testing whimsicality. To efficiently compute these values, we transform it into a graph problem. We invite you to take a moment to think about how to do that before you continue reading.

[1] ℓ is a whimsical number if and only if it is a location of a whimsical tree from pset 2.

Let $G = (V, E)$ be a weighted directed graph, where $V = \{v_0, v_1, \ldots, v_{a-1}\}$. We will add weighted edges to E, so that any path on this graph represents a summation of numbers in A. The goal is to build a graph in such a way that shortest path from v_0 to v_i corresponds to x_i. Notice that in particular $x_0 = 0$, so v_0 is a good candidate for the source vertex.

For this part, you need to perform the following tasks. First, for $A = \{5, 8, 11\}$ and $a = 5$, compute (by hand) all values x_r's according to Claim 2.[2] Next, create the weighted directed graph corresponding to this set. You may exclude all self-loops – edges starting and finishing at the same node (if any). Then, compute (by hand) the shortest path distances to all nodes in this graph, and put these values inside the corresponding nodes. These values should match with x_r you computed earlier. (Hint: assuming you ignore self-loops, your graph should have 10 edges.)

For this part, it is sufficient to simply provide the graph specified above in your write-up. If you include the description of your algorithm and/or your calculations, these may earn you partial credits in case your graph is incorrect.

(c) [5 points] **Analyze the running time**

Analyze the running time of your algorithm for `__init__` and `is_whimsical`. For `__init__`, be sure to account for the graph construction process, and specify which shortest-path algorithm you use. You may find the overall running time of your algorithm asymptotically lower than what we asked for; we put in this gap that may allow you to implement a slightly simpler program in the next part. You may assume that any single arithmetic operation takes constant time to compute.

(d) [30 points] **Implement**

Implement your algorithm. You are allowed to use Python libraries for maintaining your priority queues. If you notice that your program exceeds the time limit, think about your choice of a.

[2] You may find your solution to the Whimsical Trees problem useful here.