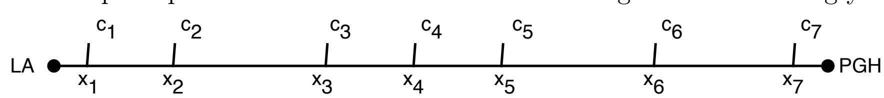
02-713 Homework #6: Dynamic Programming
Due: Mar. 7 by 11:59pm (note the unusual time)

You may discuss these problems with your current classmates, but you must write up your solutions independently, without using common notes or worksheets. You must indicate at the top of your homework who you worked with. Your write up should be clear and concise. You are trying to convince a skeptical reader that your answers are correct. Your homework should be submitted via Autolab (`https://autolab.cs.cmu.edu/02713-s14/`) as a typeset PDF. A LaTeX tutorial and template are available on the class website if you choose to use that system to typeset. For problems asking for an algorithm: describe the algorithm, an argument why it is correct, and an estimation of how fast it will run. Use O notation for running times.

1. You're driving from Los Angeles, CA to Pittsburgh, PA. There are gas stations along the way at distance $x_1, x_2, \ldots, x_n$ from Los Angeles. Because of different wait times and pump speeds, filling up at gas station x_i takes c_i minutes (the gas costs the same everywhere, so we ignore its cost). Your car can hold enough gas to go 100 miles, and you start with a full tank of gas. If you decide to stop at a gas station, you have to fill your entire tank up. Give a dynamic programming algorithm that finds where you should stop to spend the minimum amount of time at gas stations during your trip.



   Hint: you know you'll have to stop at a gas station within 100 miles of Pittsburgh, for example.

2. Let's change the problem above slightly: suppose if you stop, you don't need to fill up the entire tank. Instead, if you put in m miles worth of gas, it will take you $c_i + mg_i$ minutes at station x_i. For simplicity, assume that you start out with an *empty* tank but you start at a station x_1 in LA, and that x_n is your destination in Pittsburgh. Give a dynamic programming algorithm to solve this problem.

3. In some languages, such as Chinese, words are sometimes not separated by spaces. For another example, in German, numbers are sometimes written together "Dreihundertfünfzigfünftousand" and compound words are often created: "Glückszahl" means "lucky (Glück) number (zahl)." We would like to decompose such strings into the component words that were used to form them. Assume you have a function **word**(s) that takes a string s and returns a score indicating how likely it is that s is a indivisible word. For example, in German, "zahl" would receive a high score, but "kszahl" would not.

   Give a dynamic programming algorithm to break a given string $a = a_1 a_2 \ldots, a_n$ into words $w_1, \ldots, w_k$ to maximize $\sum_i \textbf{word}(w_i)$. (Note that you are **not** given k as input.)