Homework 3

Due Friday, Jun 4 at 11:59 PM

Fun Sea Creature Fact: Certain species of male penguin ”propose” to their lady with a pebble during mating season.

Installing and Handing In

Accept the GitHub assignment **here**. Homework is handed in through Gradescope. For written portion, submit a PDF and match the questions accordingly. For the coding portion, submit the .py files.

1 Written Problems

Problem 3.1

Induction

1. Consider the statement $P(n) : (1 + a)^n \geq 1 + a(n - 1)$, given some $a \geq 0$. Use induction to prove $P(n)$ for all $n \geq 1$ (where n and a are both integers). While it may be a helpful reference, do not use the results of the proof in the **induction handout** (on the Docs page) in this proof–the problems are similar but not the same.

Problem 3.2

Greedy vs. Dynamic Programs

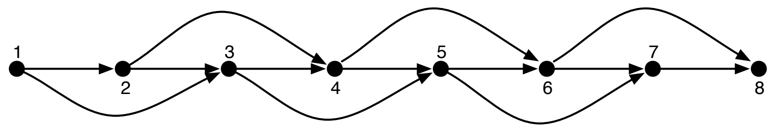

Figure 1: Possible trains to take between location 1 and 8.

Suppose you have to get from one location to another via a train line, and there are several stops along the way. At each stop you can either take a train that will take you directly to the next stop, or you can take an express train that skips the next stop and goes one stop further. This is modelled in Figure 1 above, where each arrow represents a unique train.

Let i, j be two stops that are either next to each other or separated by one stop. $time(i, j)$ then returns the time it takes to ride the direct train between those stops. Figure 2 shows a representation of this problem, similar to Figure 1, but with possible train times written in. Your task is to figure out the fastest

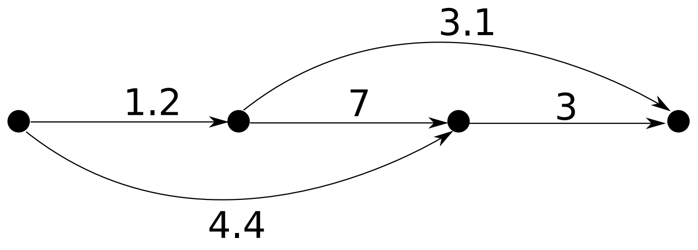

Figure 2: Possible trains with their travel times.

possible time to get from your starting location to the final location via the train line. Some questions that may be helpful in thinking about this problem are: Where will your algorithm start? What decision must you make at each train stop? What could the sub problems consist of?

Rules:

1. You can assume it takes no time to transfer between the regular and express trains.
2. You may not assume that taking the express train is faster than taking the two equivalent regular trains (the express train is old and can sometimes be slow ☺)
3. You can only move towards your destination and never backwards.
4. **Your solution should return the overall fastest time, not the fastest path itself.**

Think about your approach and then answer the following two problems. You do **NOT** have to write pseudocode for either problem, a description of your logic will suffice (but you may if it helps you explain)!

1. Describe a greedy algorithm for solving the problem. A greedy algorithm is an algorithm that makes the best choice locally (at each step), with the hope of finding a global optimum solution. Does your algorithm find the quickest route in Figure 2? Will it always find the fastest route? Prove that your greedy algorithm is the optimal algorithm, or draw a diagram (similar to Figure 2) that shows that the greedy algorithm would not produce the fastest time.
2. Describe a dynamic programming approach (like the one we used for solving seam-carving) that finds the overall fastest time to get from a starting train station to a destination train station. Remember, an algorithm that uses dynamic programming breaks the larger problem into smaller subproblems, and then it uses the solutions to those subproblems to build up to solving the overall problem. See the lecture slides for more information.

2 Python Problems

Problem 3.3

Minimum Steps

Overview

You are given an array of integers where each integer represents the maximum number of steps that can be taken forward from that position. Each time you move from one index in the array to another, this is called a jump. Design an algorithm using **dynamic programming** that returns the minimum number of jumps it takes, starting at the first element of the array, to reach the last element of the array (it is not necessary to step past the last element).

Input/Output

You can assume your function will be tested on anything that fits within the below definition for Input, but will **not** be tested on inputs that don't fit within the below input definition.

Input: An array consisting of positive (including 0) integers. This array can also be `None` or empty.

Output:

- If the input is not `None` or empty, return an integer that represents the minimum number of **jumps** (not the indices or the path) needed to get from the start of the array, to the end of the array.
  - If the input array is of length 1, return 0 since you begin already at the last index
  - If the end cannot be reached you should return `None`
- If the input is an empty array or None raise an `InvalidInputException`.

Example:

Let's say the array is $[3, 1, 2, 0, 8]$. The first node has the value 3, this means you have three options:

1. Use a step of length 1, and jump to element [1]
2. Use a step of length 2 and jump to element [2]
3. Use a step of length 3 and jump to element [0]

Even though the jump in the third option gets you the farthest, you can only take 0 steps after the jump, so this path is not viable. Instead, the path with the minimum number of jumps would be to take 2 steps from the start and then 2 steps to the end. Therefore, your minimum number of jumps, and the value you would return, is 2.

Problem 3.4

The Jewelry Set Conundrum

Overview

You start off with 180 dollars, and you want to make more money by buying and selling a set of jewelry. You know how much the value of the jewelry set will fluctuate for n days, and you have written it down in an array of n numbers $r[0] \dots r[n-1]$, where $r[i]$ represents the change in value for day i.

You can buy this jewelry set once, and sell it once. However, we encourage you to not think about this problem in terms of which days would be best to buy and sell, and to instead think about choosing (based on the array of values) which **sequence of days** would be best to collect the profit the jewelry set generates. For every day you have the set, you gain the value outlined in the array. Implement a **linear time** algorithm for finding the maximum amount of money you could end up with after buying and selling this jewelry set.

Input/Output

You can assume your function will be tested on anything that fits within the below definition for Input, but will **not** be tested on inputs that don't fit within the below input definition.

Input: An array consisting of integers. This array can also be `None` or empty.

Output:

- If the input is not `None` or empty, return an integer that represents the maximum amount of money (not the indices or days you have the jewelry set) you can gain from your jewelry set investment.
- If the input is an empty array you should return 180.
- If the input array is None you should raise an `InvalidInputException`

Example

Let's say the input array is $[-1, 2, 7, -8, 13, -2]$. If you only have the set on the first day $i = 0$ and the second day $i = 1$ you would have a total profit of $180 + (-1) + 2 = 181$. In this case, what would maximize your profits is having the jewelry from day two ($i = 1$) to day five ($i = 4$), which would leave you with \$194 ($180 + 2 + 7 - 8 + 13$).

Details

- Remember that one option is to not hold the jewelry set at all, leaving you at $180.
- Another option is for you to hold it for one day (your worth would be $180 + the dollar change for that day).
- Your algorithm must run in linear time.

3 Optional Readings

Problem 3.5

The STAs have compiled optional readings topically related to content in the course.

In a limited capacity like the shortest path algorithm, we can be mathematically sure that we have the correct answer. When algorithmic thinking is applied to the real world, because of necessarily imperfect data and modeling, results are necessarily imperfect. The first reading concerns the physical impact of Waze in Los Angeles. The second and third readings concern our immense collective trust in data:

1. **Waze Hijacked L.A. in the Name of Convenience. Can Anyone Put the Genie Back in the Bottle?**. - Los Angeles Magazine

2. **'Homo sapiens is an obsolete algorithm': Yuval Noah Harari on how data could eat the world**. - Wired

3. **Yuval Noah Harari on big data, Google and the end of free will** . - Financial Times