dining philosophers
UNIVERSITY OF PENNSYLVANIA | COMPUTER SCIENCE CLUB

Philadelphia Classic

The Dining Philosophers
Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
`http://dp.seas.upenn.edu`

19 February 2011

1. Communication

Disaster has struck! The dead are rising and there isn't much time to stock up on food, medicine, and weapons! Perhaps the greatest tragedy, however, is not the ungodly abominations walking the streets in search for blood, but the distrust among the living. You and your crew have a pretty good thing going, but you're in need of a means of communicating basic supply information without informing other humans who would attempt to rob you. As luck would have it, you've found just the way to deal with it: roman numerals! Since roman numerals are antiquated, no one will know how to read them. Unfortunately, you don't know how to read them either. So, your first task is to build a function which reads in either a roman numeral or a natural number, and outputs the corresponding natural number or roman numeral. Be sure to use correct roman numeral form (in other words, IIII does not count as 4, etc).

For those who truly don't know roman numerals:
I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000
Recall: IV = 4, IX = 9. This rule occurs for 4, 9, 40, 90, etc.
You may assume that the numbers you read in will be less than 4000

Example

Input: 2951
Output: MMCMLI

Input: CXXXIV
Output: 134

2. Driving In Circles

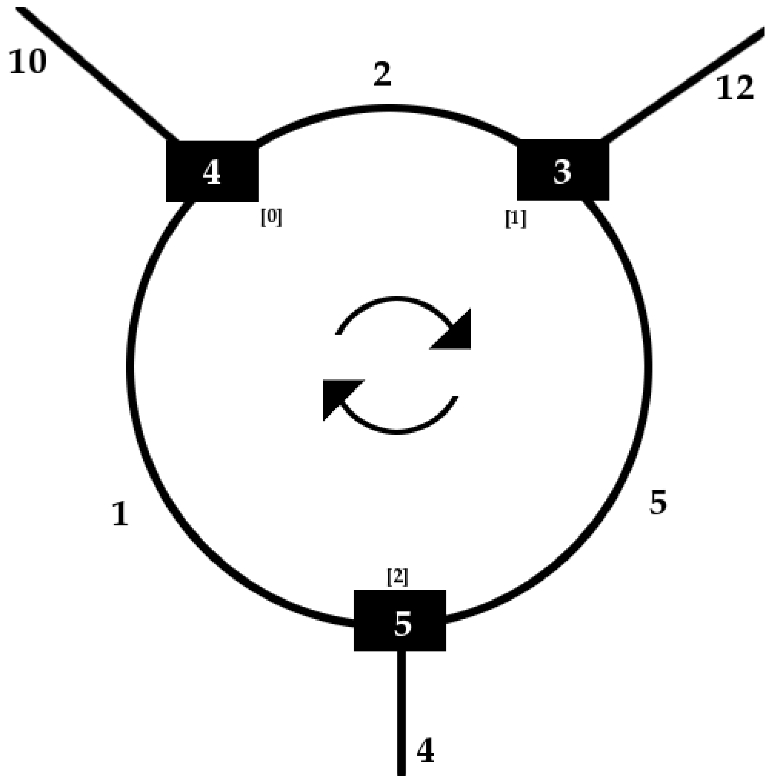
Now that you have your method of communicating crucial numerical information down, it's time to look for a way out of zombie infested territory. You've found a decent map of the city, and have to plan a route out. You know of a set of gas stations, all of which contain different amounts of fuel, and you know that you can find a car just about anywhere. Unfortunately, most of the cars that are still around are only here because they don't have any fuel in them. You determine that by passing through all the gas stations to refuel in a particular order, you may be able to find enough fuel to get out of the city. Time to get cracking!

You are given three arrays of integers, and a double rate of consumption. The first encodes fuel supply at a given gas station, the next encodes distances between gas stations, and the final encodes fuel needed to escape the city if you begin your trip at a particular station, where fuel needed to escape is the amount of fuel that you must have before departing the station to leave the city. Determine whether it is possible to amass enough fuel to escape the city. You may start at any gas station you like, but must travel in a circle from that station (linearly left to right over the array). You need to go all the way around the circle. You begin with no fuel. Output the station to begin at, or -1 if no such station exists. If there are multiple possible answers, any is acceptable.

Example

Input: [4,3,5], [2,5,1], [10,12,4], 1

Output: 2



3. The Game Of Death

Well, escaping the city doesn't look like it's going to work out. This is, of course, thoroughly problematic. You and your crew need to begin planning for the worst, and that means understanding how this zombie outbreak is being spread. You decide to model the spread of this horrid disease along the lines of Conway's Game of Life. Given a particular set of rules and a number of days, you must determine how the outbreak will affect the population.

The game will be played out on an n by m grid, each cell can be either a zombie (z), human (h) or empty (e). Two cells are considered adjacent if they touch in any way (including diagonals). A cell has at least 3 (corner) and at most 8 (middle of grid) adjacent cells: Rules will be input in the following format:
Number of adjacent zombies needed to infect a human
Number of adjacent humans needed to kill a zombie
Number of adjacent empty spaces that result in zombie death (starvation from lack of humans)
Number of turns to run

Round modifications should occur as batch modification, as seen in the example below.

Example

Input:

$$\begin{array}{ccc} z & z & e \\ h & e & e \\ h & z & e \end{array}, 2, 1, 4, 1$$

Output:

$$\begin{array}{ccc} e & e & e \\ z & e & e \\ h & e & e \end{array}$$

4. Blades Don't Need Reloading

Your model of zombie activities has served you well, and you're doing a great job avoiding the undead, but that isn't enough to win this war. You're going to need to hit them hard if you plan to end this nightmare. Fortunately, there's a small group of doctors and biomedical engineers hiding at a nearby hospital, working on cures to the disease. This cure would interest you more if you were turning into a zombie, but you decide to head to the hospital anyway, as that same group of doctors and biomedical engineers discovered a powerful weapon to destroy zombies during their quest to save lives. The journey is long and arduous, so you'll have to pack carefully.

Every weapon has a weight and value in combat. Your goal is to optimize your survivability. Given two arrays of integers, representing weight of a weapon and value in combat, and a distance you must cover in order to reach the hospital, determine the optimal set of weapons to carry. Value in combat is measured as the amount of time that weapon can keep you safe, and weight determines how fast you can move

Survival time = f(firepower), Speed = g(weight)

Note, f and g are specified at run time. Each is a polynomial of the form: $ax^2 + bx + c$, where a, b, and c are given as input. The final input is of the following form: String[], int[], int[], double[], double[], int distance, where the first two arrays specify the input to f and g, and last two arrays specify the a,b, and c values for f and g respectively. The total distance you can cover for any given weapon is the survival time multiplied by your speed. Output a list of strings, representing the maximal set of weapons.

Example

Input: [Knife,Mini Gun,Missile Launcher],[5,100,100],[1,100,500],[0,1,0],[0,0.1,0], 6000

Output: [Mini Gun, Missile Launcher]

5. Don’t Hide In A Hospital

Congratulations, you’ve made it to the hospital! And not a moment too soon, as your weapons supplies are running low and the zombies are looking mighty hungry. You enter the hospital and find the doctors. They offer you their incredible zombie slaying agent for safe passage out of the facility. It is at this point you quickly realize, hanging around facilities that are frequented by sick people is a good way to get waylayed by zombies. By the time you’ve realized this classic blunder it’s too late, and the zombies are upon you. No time to fight. Just run!

Given a 2D array of integers, representing the floor of the room, and an array of (x,y) pairs representing feasible exits, and an (x,y) pair representing the starting position, find the fastest way out of this death trap. The integer values in the 2D array is the cost incurred by traveling through a particular part of the room. The value at each room represents the time it takes to cross that area (sometimes you have to crawl to sneak by the zombies).

Example

Input:

```
 7  5  1  4
 8  6  9  9
 2  5  1  3 , [(4,1),(0,0)], (2,2)
 4  1  3  4
 0  3  6  1
```

Output: [(2,2),(3,2),(3,1),(4,1)]

6. Turrets and Mines

You narrowly evaded the zombies at the hospital (some of the biomedical engineers didn't make it. Thank goodness we're computer scientists). Much to your dismay, the doctors have been holding out on you. The chemical agent they created is going to require several more days before it can be weaponized. As a result, you're stuck defending them until they can hand off the weapon to you. You return to your base camp, around which you've laid remote mines. You also have a set of automated turrets you'd like to set up to gun down any incoming zombies. Naturally, it would be foolish to set a turret too close to a mine, so be careful when building your defenses.

You are given an NxN array with e's and m's. The e's represent empty spaces in your defenses, and the m's represent mines. Place turrets, represented as t's, in the array such that every column and row of the provided defenses contains exactly one turret or exactly one mine.

Example

Input:

$$\begin{array}{cccc} e & e & e & e \\ e & m & e & e \\ e & e & e & e \\ e & e & m & e \end{array}$$

Output:

$$\begin{array}{cccc} t & e & e & e \\ e & m & e & e \\ e & e & e & t \\ e & e & m & e \end{array}$$

7. Chemical Warfare

The doctors have pulled through, and you now possess incredible zombie-killing grenades! Since your defenses are holding off the hoard, it's the perfect time to go on the offensive. You and your crew leave base camp to nd and destroy as many zombies as you can. Unfortunately scurvy has set in, so you can only throw one grenade on each excursion, and you want to use them as efficiently as possible.

Given an NxM array of integers, representing a concentration of zombies, and a burst radius of your grenade, produce an optimal rectangle of zombies to attack with your newfound power. Note, a burst radius of 2 means you should consider a 2x2 burst.

Example

Input:

$$\begin{matrix} 1 & 6 & 3 & 0 \\ 4 & 2 & 1 & 9 \\ 1 & 15 & 7 & 8 \\ 1 & 5 & 2 & 2 \end{matrix}, 2$$

Output:

$$\begin{matrix} 15 & 7 \\ 5 & 2 \end{matrix}$$

8. Zombie Genetics

The good news about this chemical weapon is that the zombies are being killed in waves. The bad news is that some of the zombies are becoming smarter instead of dying (you may recall, this was originally intended as a cure). Intelligent zombies, while highly improbable, are clearly a great danger to you. You return to the doctors in the hopes that they can help you determine friend from foe.

Given a sample input string of human DNA, an array of DNA strings, and a threshold for error, determine which DNA strings belong to human beings. Apply the Hamming Distance Metric (number of character swaps until matching) to compare the strings, and reject any with a number of flips higher than the input threshold percentange.

You may assume all the DNA strings you are given (the human DNA, and the test strings) all have the same length.

Example

Input: ACGGTGGTACGG, [ACGGCGGTGCGA,ACGGAGGTACAG,ACGGTCCAGATT], .5

Output: [ACGGCGGTGCGA,ACGGAGGTACAG]