Computer Algorithms

Pleasanton Math Circle

1 Data Structures

Before we delve into algorithms, we will cover the essence of computer science. In this field of science, we use computers to manipulate data structures, or ways of ordering information in a computer. We will cover these data structures to show what is and is not allowed when writing an algorithm for a computer.

The most common method a computer holds data is through a variable. A variable can take on any value up to a certain memory limit.

Arrays are groups of variables in a line. In an array, each variable has an index, or a specific number that indicates the location in which a variable is stored. For example in the array $\{1, 5, 2\}$, the number 1 has index 0, the number 5 has index 1, and the number 2 has index 2.

Most of the operations in computer science involve systematically manipulating data structures. Although there are a lot more data structures that offer greater functionality, only these two are needed this lecture.

2 Algorithms

A **algorithm** is a set of instructions to perform a task.

Exercise 2.1. Jonathan owns a 2017 story building. He also has an infinite supply of identical magical eggs. He wants to determine the highest floor of the building where he can drop an egg so that it doesn't break. Write an algorithm for Jonathan to follow to complete this task. Can you minimize the number of eggs Jonathan needs to test?

Exercise 2.2. Kenny and Steve are playing a dumb version of battleship. Kenny has 40 points on a line which Steve can't see. He places 10 battleships on 10 distinct points. In each of his turns, Steve guesses a point. If it contains a battleship, he wins. Write an algorithm for Steve to follow so that he wins in at most 31 turns.

Exercise 2.3. Given a random finite permutation of integers from $1...n$, find an optimal algorithm to put the list in numerical order, given that you can exchange two elements per move.

Exercise 2.4. Kenny and Katherine Morotto are out on a date at a restaurant. Unfortunately both of them are very hungry (and also very competitive), so they will stop at nothing to eat the greater value of food. There are $2n$ items of food in a line, and each item has a specific value. The problem is that each person can only reach a food item at the end. To settle this disagreement civilly, both Kenny and Katherine decide to take turns eating an item of food from one end. Write an algorithm that will allow Kenny to always eat the same value or more than Katherine, given that he can choose whether he goes first or second.

Exercise 2.5. (Challenge) When Jeffery is coding, he wants to use a function that returns 1 with probability p and 0 with probability $1 - p$. However, he finds that he can only use one library. This library contains one function, B, that returns the number 0 with probability $\frac{1}{2}$ and the number 1 with probability $\frac{1}{2}$. However Jeffery is really lazy, and doesn't want to write this code himself. Help Jeffery out and prove that the algorithm you write terminates (i.e. it doesn't run forever).

Exercise 2.6. (Challenge) Given any maze with a starting point and at least 1 ending point, write an algorithm that gives the minimum number of steps to solve the maze.

3 Big O Notation

Big O notation is used to quantify the time and memory complexity of an algorithm. Since the runtime of an algorithm depends on its input, big O notation shows, as a function of the input, what the runtime is equal to.

For example, consider an algorithm that takes in an integer n and outputs all numbers from 1 to n. When a bigger input is taken, the performance of the algorithm increases linearly. Thus, we say that the time complexity of this algorithm is $O(n)$.

Exercise 3.1. You are given a task to output, for all j such that $1 \leq j \leq n$, the odd numbers from 1 to j. For example, given $n = 5$, you have to output $(1), (1), (1, 3), (1, 3), (1, 3, 5)$. Find the time complexity of the most efficient algorithm that accomplishes this task.

Exercise 3.2. Find the time complexity of looking for your friend's number in a phone book in the most efficient manner possible.

Exercise 3.3. Go back to the algorithms section and write the big O notations for your solutions. Can you think of algorithms that will lower the time complexity?

Exercise 3.4. (Challenge) Find the time complexity of this algorithm that gives the nth Fibonacci number:

Fibonacci (n)
if n = 0: return n
return Fibonacci(n-1)+Fibonacci(n-2)

Can you think of a way to lower the time complexity?

4 Other Algorithms

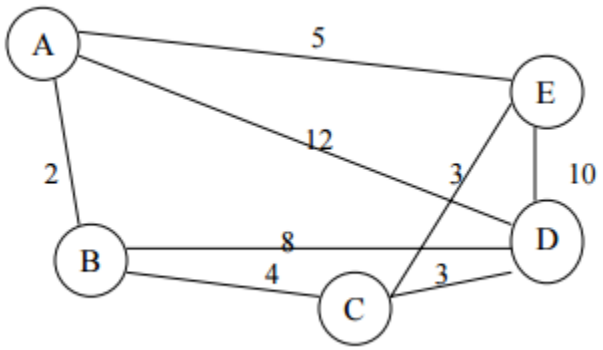
A major branch in computer science is finding algorithms that will efficiently solve problems in the world. Here, we will go over a few other algorithms, two concerning graph theory, to introduce you to this science.

4.1 The Traveling Salesman Problem

The Traveling Salesman Problem is one of the most famous and studied (and pseudo-unsolved) problems in computer science. It is as follows:

Given a list of cities and the distance between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

For instance, Kenny the Jelly is a salesman and wants to sell Greebes. He starts at City A. The numbers on the edges indicate the distances between cities.



Exercise 4.1. Find the shortest path Kenny the Jelly can take to visit each city exactly once that returns to City A.

4.2 Greedy Algorithm

In the above example, you probably first employed the Greedy Algorithm. In the context of the Traveling Salesman Problem, the Greedy Algorithm entails picking the nearest unvisited city to go next. However, note that like in the above example, this *does not* always produce the shortest/correct path.

More generally, the Greedy Algorithm picks the *local* optimal choice at each stage in the hope that this will overall be the best choice. It's often labeled 'short-sighted' because it picks the best immediate choice without regard to the future. Usually, the Greedy Algorithm give an approximate (yet not exact) optimal solution overall, yet there are exceptions.

Exercise 4.2. Say that you, as a cashier, have an unlimited number of quarters, dimes, nickels, and pennies. You want to give someone exactly \$1.47 in change, with the least number of coins possible.
(a). Use the Greedy Algorithm to calculate how many coins you need. Is this the true minimum number of coins needed?
(b). Are quarters, dimes, nickels, and pennies canonical (in other words, for any value of change, will the Greedy Algorithm produce the least number of coins needed?) Why?

Exercise 4.3. You have been upgraded to a cashier on Planet Jangaroo. In this planet, you are instead given infinitely many coins with values 1, 3, 5, 8, and 9 Greebes, and a customer is now requesting 24 Greebes in change.
(a). Use the Greedy Algorithm to calculate how many coins you need.
(b). Is this money system canonical?

4.3 Dijkstra's Algorithm

Dijkstra's algorithm, unlike the Greedy Algorithm, finds the shortest distance between two nodes in a graph in runtime $O(|V|^2)$. It has 6 steps:

1. Create an array of distance values for each node. Initialize the distance value for the initial node as 0 and all the other nodes as infinity.

2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.

3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.

5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Exercise 4.4. Using the graph and cities of Kenny the Jelly, calculate the shortest distance from A to each of the other cities.