Introduction to Programming

RMHS Computer Team

October 11 2016

1 Pseudocoding

Pseudocoding writes out the steps of an algorithm in human readable language. This is typically written as a step of math and input/output statements. The computer can primarily do several things

1. Get Input/Produce Output
2. Store Variables
3. Do Math
4. Check if a condition is true/false
5. Loop

2 Big O Notation

We can display the amount of steps taken, as a function of input size, using Big O Notation. For an input size n, we would write that the worst case runtime of the algorithm is $O(f(n))$, where $f(n)$ is some function of n. For example, if a function takes $4n^2 + 2n + 3$ steps, then we would write that as $O(n^2)$, as only the biggest term is written down. A computer can typically process 10^9 steps per second. Some common big O functions are listed below, from smallest, to biggest.

1. $O(1)$
2. $O(logn)$
3. $O(\sqrt{n})$
4. $O(n)$
5. $O(nlogn)$
6. $O(n^2)$

7. $O(n^3)$
8. $O(2^n)$
9. $O(n!)$

3 Example Problems

3.1 Cube Volume

Problem Statement: Given a positive integer, which represents the side length of a cube, find the cube's volume
We know that the volume of a cube is given by the formula $V = s^3$.
We can write this in pseudocode as follows.

Algorithm 1 Find the Volume of a cube

```
1: Side ← Input the side length of the cube
2: Volume ←(Side)^3
3: Print Side
```

The number of steps taken not dependent upon the input size (there is always, only 1 step), so the runtime is $O(1)$.

3.2 List Maximum (UMD 2016)

Problem Statement: Your given a list of numbers (there are less than 10^4 numbers). You want to determine a start and stop value, such that the cards between start and stop (inclusive) has the maximum sum.

We could brute force all possible start values, then try all end values, and calculate the sum. We use a for loop, which allows for us to loop through a list of numbers. We also use an array, which is just another word for a list, to store the values of all the cards.

Algorithm 2 Find the maximal sum

```
1: NumberList ← Input the list of cards, of length n
2: Initialize the bestSum ← 0
3: Initialize the bestStart and bestEnd as 0
4: for start between 1 ··· n do
5:     CurrentSum = 0 (because the sum of zero numbers is zero)
6:     for end between start ··· n do
7:         CurrentSum ← CurrentSum + NumberListatEnd
8:         if CurrentSum > bestSum then
9:             bestSum ← currentSum
10:            set bestStart and bestEnd to start and end
11: Print the value of bestStart, bestEnd and bestSum
```

So some things to note. We initialized the best sum value, and then changed it, if we found a better sum. We also went from every start value, and tried every possible end value that was greater (or equal).

The algorithm tries every start value, so that's n different start values. For each start value, there are, on average, $n/2$ values ahead of it. Therefore the runtime is $O(\frac{n^2}{2}) = O(n^2)$.

3.3 Date Problem (USACO 2014)

Problem Statement: If today's date is October 11 2016, then given a number, n ($n < 10^5$), print the new date n days into the future

An easy solution would be to determine, first if it is within this year, if not, then determine what year its in. Then, after the year is determined, determine the month, then the day.

Algorithm 3 Find the date

```
1: Days ← Input the number of days into the future
2: DaysPerMonth ← [31,28,31,30,31,30,31,31,30,31,30,31
3: DaysPerMonthLeap ← DaysPerMonth, except 29 instead of 28
4: CurrentMonth = 10, CurrentYear = 2016, CurrentDay = 11
5: while Days>0 do
6:     Add one to currentDay
7:     Check if the currentDay>30/31 (depending on the month)
8:         If so, change the currentMonth and currentDay
9:     Check if the currentMonth>13
10:        If so, change the currentMonth and currentYear
11:    Subtract one from Days
12: Print out currentDay, currentMonth, currentYear
```

The solution runs in time $O(n)$, as we simply loop through each day. We accordingly check if either its a new month, or a new year, and as such, change the date. This solution should be sufficient, as $n < 10^5 < 10^9$.

3.4 Necklace Problem (USACO Traingate)

Problem Statement: Given a necklace of red, blue and white (wildcard) beads, you break the necklace at a certain point, collect necklaces that are the same until you reach a different color, for both sides, find the location where the maximum break point is

Algorithm 4 Necklace breaking

1: *Necklace* ← Input the necklace, as a string of r, b, w, of length n
2: bestBreak ← 0
3: **for** Each each breaking point along the necklace, i **do**
4: Initialize the number of similar beads to the right and left, to 1
5: Count the number of beads to the right that are the same, set it to numberRight
6: Do a similar thing for left, set it to numberLeft
7: **if** NumberRight = n **then**
8: All the beads in the necklace are of the same color, bestBreak ← n
9: Else
10: bestBreak = $max(bestBreak, numberLeft + numberRight)$
11: Print bestBreak

3.5 Challenge Problems

Problem Statement: You're given n dials, each with 10 settings (0, 1, ⋯ 9). They're arranged in a circle. The dials work such that moving one dial clockwise, moves the one to the left counterclockwise, and vice versa. Given an initial setting of dials, determine how many moves it would take to move all of them back to 0. If it's not possible, print -1.
http://www.cs.umd.edu/Outreach/hsContest16/questions/problems.pdf (Num 7)

Problem Statement: At a party where all the guests drink milk, one of the m milks has gone bad. Everyone who drinks the bad milk will become sick either during the party, or after. Your given who drank which milk, at what time, and also who got sick, at what time. Your job is to find out how many doses of medicine need to be brought back for those who got poisoned
http://www.usaco.org/index.php?page=viewproblem2cpid=569