CS 341 - Programming Project 1

Awnon K. Bhowmik

September 27, 2019

The Battery Powered Car Problem

The Setting

- There is one battery powered car
- There is a pile of n batteries
- The maximum mileage of the car, is k km per battery
- The car can carry a maximum of one spare battery
- The car starts from a point S and goes back and forth to displace the batteries, and itself a certain distance of x km from S

The Question

Find the maximum displacement from the starting point S for a given pile of n batteries.

Analysis

Given n batteries, if the car has a mileage of k km on one battery, then on n batteries, it will travel a maximum of nk km. Moving in this manner means, with an odd number of batteries, the total displacement will be k km, whereas with an even number of batteries, the displacement will essentially be 0.

This is why we are forced to take another way out. Suppose we move x km with the pile of batteries. The car uses one, so the other $(n-1)$ batteries are moved a distance of x km one at a time. The round trip is $2x$ km for the first $n-2$ batteries.

The effective displacement from S is given by the following algebraic equation

$$2x(n-2) + x = k$$

where x is the one way trip in km. Solving for x gives...

$$x = \frac{k}{2n-3} \tag{1}$$

Equation (1) can be turned into one with a **recursive nature**, by simple algebraic manipulations. Suppose that we rewrite equation (1) as a sequence

$$x_n = \frac{k}{2n-3}$$

The next term can then be calculated as....

$$
\begin{aligned}
x_{n+1} &= \frac{k}{2(n+1)-3} \\
\frac{1}{x_{n+1}} &= \frac{2n-3+2}{k} \\
\frac{1}{x_{n+1}} &= \frac{2n-3}{k} + \frac{2}{k} \\
\frac{1}{x_{n+1}} &= \frac{1}{x_n} + \frac{2}{k} \\
x_{n+1} &= \frac{1}{\frac{1}{x_n} + \frac{2}{k}} \qquad \textbf{This is what we'll use} \\
x_{n+1} &= \frac{k x_n}{2x_n + k} \qquad \begin{cases} \textbf{Using this requires two recursive calls} \\ \textbf{which slows down execution time by a lot} \end{cases}
\end{aligned}
$$

The $C++$ implementation

Algorithm

- We pre-define the mileage k of the car running on a single battery. This is typically a 100 miles for an average battery powered car, with the exception of Tesla Model S that averages about 250 miles on a single charge.
- We read in a battery pile size n.
- While $n > 0$ we do the following

$$
x_{n+1} = \begin{cases} 1 & n = 1, 2 \\ \dfrac{k x_n}{2x_n + k} & \text{otherwise} \end{cases}
$$

- We make the program display the battery remaining and the distance traveled. This step is optional, we can only print out given the initial pile size n and the distance traveled x.

Algorithm 1 Pseudocode for algorithm

procedure MyProcedure
    $k \leftarrow$ distance traveled in single charge
    $n \leftarrow$ battery pile size
    $x \leftarrow 0$ Initially distance traveled in one direction is zero
    if $n = 1$ or $n = 2$ **then return** k
    else return $\dfrac{k x_n}{2x_n + k}$
loop:
    $x \leftarrow x + \dfrac{k x_n}{2x_n + k}$
    print n, x
    goto *loop*
    close

The C++ Code

```cpp
#include <iostream>
#include <iomanip>
constexpr auto k = 100;

using namespace std;

double dist(int n)
{
    if (n == 1 || n == 2)
        return k;
    else
        return (1.0 / (1.0 / dist(n - 1) + 2.0 / k));
}

int main()
{
    int n;
    double x = 0;
    cout << "Enter the number of batteries:";
    cin >> n;

    cout << "Batteries"
         << "\tDistance" << endl;
    while (n > 0)
    {
        x += dist(n);
        cout << setw(5) << n << setw(15) << x << endl;
        n--;
    }

    return 0;
}
```

Data Analysis

The following is a sample output data from the program.

```
Enter the number of batteries:20
Batteries	Distance
   20         2.7027
   19        5.55985
   18        8.59015
   17         11.816
   16        15.2642
   15        18.9679
   14        22.9679
   13        27.3158
   12        32.0777
   11        37.3408
   10        43.2232
    9        49.8898
    8        57.5822
    7        66.6731
    6        77.7842
    5        92.0699
    4         112.07
    3        145.403
    2        245.403
    1        345.403
```

Table 1

According to the data from Table 1, it seems like with more and more batteries, the car can initially move only a little, but it gradually covers more distance with the remaining batteries, and drastically when the batteries are about to run out. This is exactly what we were supposed to get, the situation model turned out as such.
This also tells us that

$$x \propto \frac{1}{n}$$

where n represents the number of batteries remaining in the pile. The input of 20 was chosen at random. Initially there were some thoughts of splitting up into even and odd inputs, but the way the code is written took care of it after all. Basically, the car initially travels shorter distance since it has to compensate going back and forth to move the pile of batteries. As a result, it *loses effective distance traveled forth* by quite a lot. This can be found by just comparing the distance we have from our output with the theoretical distance possible. This is

$$\textbf{difference} = 20 \cdot 100 - 345.403 = 1654.597$$

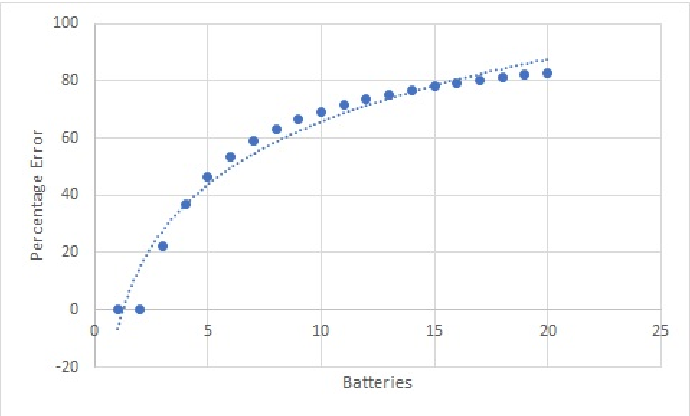
and

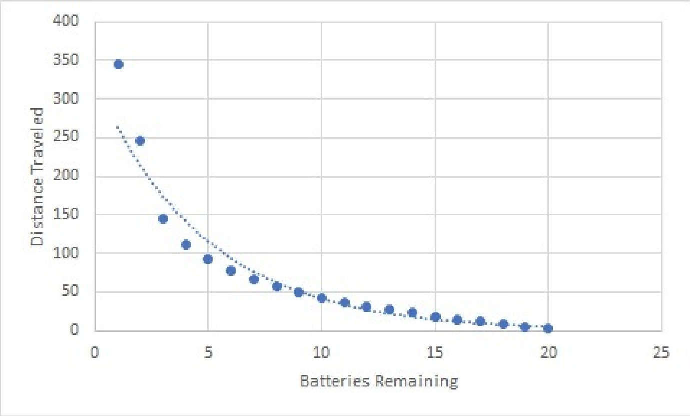
$$\textbf{percentage difference} = \frac{1654.597}{2000} \times 100 = 82.7\%$$

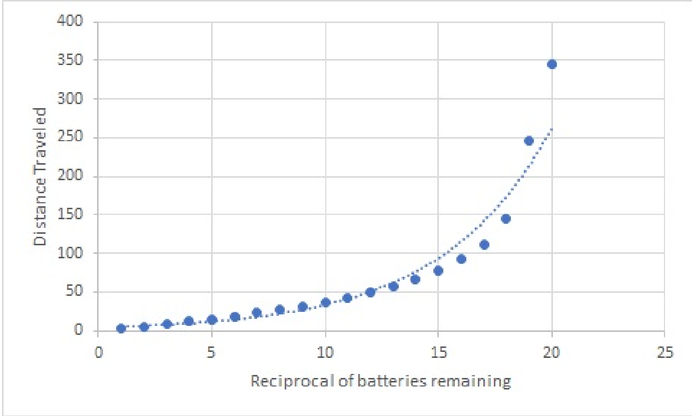
Needless to say, this error increases (logarithmically) as the starting number of batteries in the pile increases. Let's look at a table...

Batteries	Theoretical Distance	Actual Distance	Percentage Difference
1	100	100	0
2	200	200	0
3	300	233.333	22.22233333
4	400	253.333	36.66675
5	500	267.619	46.4762
6	600	278.73	53.545
7	700	287.821	58.88271429
8	800	295.513	63.060875
9	900	302.18	66.42444444
10	1000	308.062	69.1938
11	1100	313.326	71.51581818
12	1200	318.087	73.49275
13	1300	322.435	75.19730769
14	1400	326.435	76.68321429
15	1500	330.139	77.99073333
16	1600	333.587	79.1508125
17	1700	336.813	80.18747059
18	1800	339.843	81.11983333
19	1900	342.701	81.96310526
20	2000	345.403	82.72985

Here's a visual







Our original C++ code can be tweaked so that it gives us the maximum distance traveled for a number of batteries in the starting pile. This time we will use a custom defined input, and not user input to keep it consistent with our previous experimental results. The code is as follows....

A variation of the C++ code

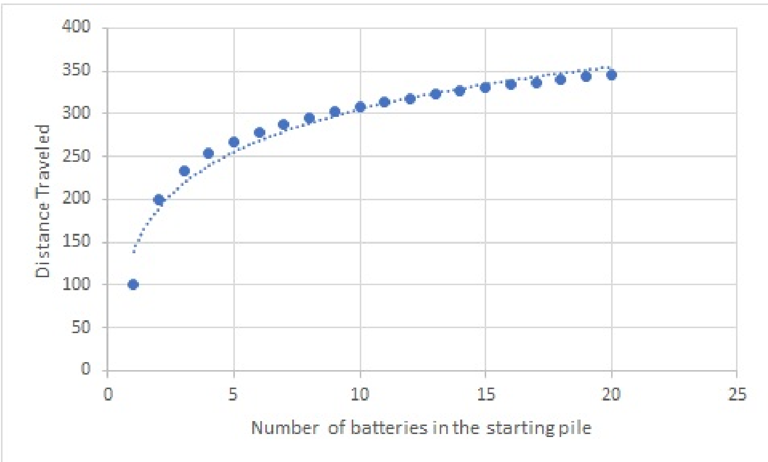
```
#include <iostream>
#include <iomanip>
constexpr auto k = 100;

using namespace std;

double dist(int n)
{
   if (n == 1 || n == 2)
      return k;
   else
      return (1.0 / (1.0 / dist(n - 1) + 2.0 / k));
}

int main()
{
   int n;
   double x = 0;

   cout << "Batteries"
      << "   Distance" << endl;

   /* This gives the maximum distance that can be
   traveled by the car with n number of batteries
   in the initial pile */
   for (int m = 1; m <= 20; m++) {
      n = m;
      while (n > 0)
      {
         x += dist(n);
         n--;
      }
      cout << setw(5) << m << setw(15) << x << endl;
      x = 0;
   }

   return 0;
}
```

Here is a snapshot of the output

Batteries	Distance
1	100
2	200
3	233.333
4	253.333
5	267.619
6	278.73
7	287.821
8	295.513
9	302.18
10	308.062
11	313.326
12	318.087
13	322.435
14	326.435
15	330.139
16	333.587
17	336.813
18	339.843
19	342.701
20	345.403

Table 2

A graph plotted from the results in Table 2 is given below



It seems like a logarithmic trend line is close to the best fit for this graph. This is what was expected just by looking at the output data.