CMSC 351: Integer Addition

Justin Wyss-Gallifent

April 4, 2021

1 Introduction

Suppose we have two n-digit numbers and wish to add them. What is the worst-case time complexity of this operation?

2 Schoolbook Addition

The first and most obvious way to add two numbers is the way we learn in school. We add digit-by-digit and carry if necessary:

$$\begin{array}{cccccc} & {}^{1} & {}^{1} & & {}^{1} & \\ & 3 & 6 & 7 & 2 & 8 \\ & 6 & 5 & 9 & 1 & 6 \\ \hline 1 & 0 & 2 & 6 & 4 & 4 \end{array}$$

3 Pseudocode

If we store each number digit-by-digit in arrays `A` and `B` then the pseudocode for adding them and putting the result in `c` is as follows. To make things a little simpler we are storing the 1's digit in `A[0]`, the 10's digit in `A[1]` and so on, so we print the lists backwards.

```
\\ PRE: A and B are lists of length n containing
\\      the digits of two numbers.
\\ PRE: C is an empty list of length n+1.
C = list of 0s of length n+1
carry = 0
for i in range(0,n):
    C[i] = A[i] + B[i] + carry
    if C[i] > 9
        carry = the 10s digit of C[i]
        C[i] = the 1s digit of C[i]
    else
        carry = 0
    end
end
C[n] = carry
\\ POST: C contains the digit-by-digit result of adding A and B.
```

4 Pseudocode Time Complexity

What is the time complexity of this algorithm? Well it does constant-time operations before the loop, n constant-time operations for the loop, and constant-time operations after the loop, so worst-case, best-case, and average-case are all $\Theta(n)$.

5 Improvments

Could we do any better?
For numbers $a_n...a_1$ and $b_n...b_1$ we wish to find $c_{n+1}c_n...c_1$ (we go to c_{n+1} because there may be an additional digit). To find c_1 we absolutely have to calculate $a_1 + b_1$ since there is no other way to find that digit since we certainly can't figure it out from the remaining a_i and b_i.

Likewise to calculate c_2 we'll potentially need a carry digit from $a_1 + b_1$ but again we absolutely have to calculate $a_2 + b_2$. This pattern continues and in general we have no choice but to do the individual digit additions. Thus there are n required operations for a time complexity of $\Theta(n)$.

6 Thoughts, Problems, Ideas

1. Assume `A` and `B` are binary strings of length n and rewrite the pseducode, removing addition and comparison and instead using logical operators `and` (only once) and `xor` (only once).

2. The addition pseudocde can be rewritten to eliminate `carry` and instead store the carry pre-emptively in `C`. Do so.

3. Two's Complement: For a given binary number `B` the Two's Complement of the number is obtained by negating all the bits and adding 1. For example the two's complement of `B=01101` is `not(B)+1=10010+1=10011`. For a number `B` with N bits if we add `B` and its two's complement we always get 2^N, for example `B+not(B)+1=01101+10011=100000`. Consequently for `A>=B` we have `A+not(B)+1=A+(2^N)-B=2^N+(A-B)` and so we can calculate `A-B` by instead calculating `A+not(B)+1` and ignoring the resulting leftmost digit. For example:

$$\begin{aligned}\texttt{1011101-0110111} &= \texttt{1011101+not(0110111)+1}\\ &= \texttt{1011101+1001000+1}\\ &= \cancel{\texttt{1}}\texttt{0100110}\end{aligned}$$

   Write the pseudocode for this. Just for extra fun and excitement:

   - Do not use a carry bit.
   - Do not use any conditionals.
   - Use only one loop.

   You can just assume the additional resulting bit will be ignored.

7 Python Test and Output

Code:

```
import random
A = []
B = []
for i in range(0,7):
    A.append(random.randint(0,9))
    B.append(random.randint(0,9))
n = len(A)

print('    ' + str(A[::-1]))
print('    ' + str(B[::-1]))

C = [0] * (n+1)
carry = 0
for i in range(0,n):
    C[i] = A[i] + B[i] + carry
    if carry == 0:
        print(str(A[i])+'+'+str(B[i])+'='+str(C[i]))
    else:
        print(str(A[i])+'+'+str(B[i])+'+'+str(carry)+'='+str
            (C[i]))
    if C[i] > 9:
        carry = C[i] // 10
        C[i] = C[i] % 10
        print('Carry the '+str(carry))
    else:
        carry = 0
C[n] = carry

print(C[::-1])
```

Output:

```
   [7, 2, 8, 9, 9, 6, 2]
   [2, 6, 8, 3, 4, 3, 0]
2+0=2
6+3=9
9+4=13
Carry the 1
9+3+1=13
Carry the 1
8+8+1=17
Carry the 1
2+6+1=9
7+2=9
[0, 9, 9, 7, 3, 3, 9, 2]
```