COS 423 Lecture 1

Counting in Binary

Amortized and Worst-Case Efficiency

Number	Binary	Total cost	Cost to add 1
0	0	0	1
1	1	1	2
2	10	3	1
3	11	4	3
4	100	7	1
5	101	8	2
6	110	10	1
7	111	11	4
8	1000	15	1
9	1001	16	

Total cost to count to n

Cost to add 1 = number of trailing 1's + 1

= decrease in number of 1's + 2

Worst-case cost to add 1: $\lg(n + 1) + 1$

Total cost to count to n: $n\lg n$?

Amortize: to liquidate a debt by installment payments.

From Medieval Latin: to reduce to the point of death.

In analysis of algorithms: to pay for the total cost of a sequence of operations by charging each operation an equal (or appropriate) amount.

(A little stretch, but there you have it.)

Amortized Efficiency

Coin-operated computer
bit flip costs $1

$2 per addition suffices:
unspent $ = #1's ≥ 0

total cost = 2*n* - #1's
≤ 2*n*

Amortization (banker)

Assign $ to each operation (amortized cost)

Keep track of savings (or borrowings) in state of data structure

If no debt after all operations,

total cost ≤ sum of amortized costs

Amortization (physicist)

Switch perspective: start with savings,

derive cost per operation

Assign “potential” Φ to each state of data structure

Define “amortized cost” of an operation to be actual cost plus net change in potential:

$$a_i = t_i + \Phi_i - \Phi_{i-1}$$

$$\textit{Thus}\quad t_i = a_i + \Phi_{i-1} - \Phi_i$$

total actual cost =

total amortized cost + initial Φ – final Φ

$\leq$ total amortized cost

if initial $\Phi = 0$, final $\Phi \geq 0$

(no net borrowing)

Binary counting: Φ = number of 1's

$\Phi_0 = 0$, $\Phi_n \geq 0$

Amortized cost to add one = 2

$\rightarrow$ total actual cost $\leq 2n$

Frequency of multiple carries

Observation: a cost of $k + 1$ or more occurs at most $n/2^k$ times (out of n)

Proof of observation via a potential function: Fix k. Let $\Phi = n \bmod 2^k$. Each add increases Φ by one, unless cost is $k + 1$ or more. (We call the add *expensive*.) In this case $n \bmod 2^k = 2^k - 1$, so Φ decreases by $2^k - 1$. This can happen at most $n/2^k$ times out of n: $\Phi = n - e2^k \geq 0$, where e = #expensive adds.

Carry-free binary addition and borrow-free subtraction

Redundant Binary Numbers

Allow 2 as a digit as well as 0, 1

Number representations are no longer unique:

210 = 1010 = 1002

How does this help?

22222 + 1 = 111111?

Need to eliminate adjacent 2's

Regularity: At least one 0 between each pair of 2's (adjacent or not)

Regular: 120102, 211

Not regular: 2112021

Fix for addition:

$02 \rightarrow 10$

$12 \rightarrow 20$

To add one:

Fix rightmost 2.

Add 1 to rightmost digit.

$112021 + 1 = 112101 + 1 = 112102$

$21101201 + 1 = 21102001 + 1 = 21102002$

Correct, maintains regularity, and carry-free:

changes at most three digits

Implementation

Stack of positions of 2's, rightmost on top:

2011102111012110 1

16, 10, 4 (top)

Can update stack in O(1) time

What about borrow-free subtraction?

Need to avoid adjacent 0's, which force borrowing

Strict regularity: 0's and 2's alternate, ignoring 1's

Problem: fix for addition can violate strict regularity

Strictly regular addition:
If rightmost non-1 is a 2, fix it.
Add 1 to rightmost digit.

21101201 + 1 = 21101202 (no fix)

Fix for subtraction:

10 → 02

20 → 12

(look familiar?)

To subtract one:

If rightmost non-1 is a 0, fix it.

Subtract 1 from rightmost digit.

112021 – 1 = 112020 (no fix)

2110120 – 1 = 2110112 – 1 = 2110111

Implementation:

Stack of non-1’s, rightmost on top

211011120110121

14, 11, 7, 6, 3, 1 (top)

Addition and subtraction are correct, maintain strict regularity, and are carry-free and borrow-free, respectively.

Proof: A fix does not change the value of the number. A fix preserves regularity:

$$\ldots 21^*021^* \leftrightarrow \ldots 21^*101^*$$

$$\ldots 01^*121^* \leftrightarrow \ldots 01^*201^*$$

(1* = zero or more 1's.) After an add-fix, the rightmost digit is not a 2, so no carry. After a subtract-fix, the rightmost digit is not a 0, so no borrow.

Adding or subtracting 1 changes at most three digits.

Extensions

Addition or subtraction of two numbers in worst-case time proportional to the length of the smaller one

Addition or subtraction of an arbitrary power of 2

(a 1 in position *k*)

Use of any three consecutive digits, e. g.

{–1, 0, 1}, {2,3,4}

Q: If strict regularity works, why bother with regularity?

A1: If no subtractions, add is simpler and extra stack is smaller.

A2: Design space is worth exploring. Options may be incomparable.