2021CS50126 | Vaibhav Bajaj

COL100

Assignment - 2

Part 1

Algorithms

Note: The numbers represented below will be used for the entire discussion. All of them are in binary form, unless specified otherwise.

The numbers a and b are represented as follows:

$$a = a_m a_{m-1} \dots a_2 a_1 a_0$$

$$b = b_n b_{n-1} \dots b_2 b_1 b_0$$

Where a_i and b_i belongs to $\{0,1\}$, provided $a_m \neq 0$ if $m \neq 0$ and $b_n \neq 0$ if $n \neq 0$.

s is a single bit number, either 0 or 1.

k is any arbitrary number in base 10, such that $k \in \mathbb{N}$.

The functions required are:

1. The function $join(a, s, k)$ returns the number a having the bit s concatenated to it k times at its end.

$$join(\mathrm{a, s, k}) = \begin{cases} a, & if\ k = 0 \\ a_m a_{m-1} \dots a_2 a_1 a_0 s_{k-1} \dots s_0, & otherwise \end{cases}$$
$$where\ s_i = s\ \forall\ i \in \mathbb{N}$$

2. The function $mult(a, s)$ returns the product of the binary number a and a single bit s.

$$\mathrm{mult(a, s)} = \begin{cases} 0, & if\ s = 0 \\ a, & otherwise \end{cases}$$

3. The function $multiply(a, b)$ returns the product of the two binary numbers a and b.
   The algorithm is as follows:

   $multiply(a, b)$:

   i) If $b = 0$ or $a = 0$, the product is 0.
   ii) Else:

   $$let\ \mathrm{p_i} = mult(\mathrm{a}, \mathrm{b_i}) \quad \forall\ i \in \mathbb{N}, i \leq n$$
   $$let\ q_i = join(p_i, 0, i) \quad \forall\ i \in \mathbb{N}, i \leq n$$

   Therefore, the product is:

   $$prod = \sum_{i=0}^{n} q_i = q_0 + q_1 + \cdots + q_n = c_p c_{p-1} \ldots c_2 c_1 c_0$$

Proving the correctness

1. The function $join()$ is correct by definition.

2. $mult(a, s)$:
   Claim: It returns $a * s$.

   Proof:

   Case I: $\mathrm{s} = 0$
   $\Rightarrow \mathrm{mult}(\mathrm{a}, \mathrm{s}) = 0 = \mathrm{a} * \mathrm{s}$

   Case II: $s = 1$
   $\Rightarrow \mathrm{mult}(\mathrm{a}, \mathrm{s}) = \mathrm{a} = \mathrm{a} * 1 = \mathrm{a} * \mathrm{s}$

   Hence $mult(\mathrm{a}, \mathrm{s}) = a * s\ \ \forall\, a \in \mathbb{N}, s \in \{0,1\}$

   ■

3. $multiply(a, b)$:
   Claim: It returns $a * b\ \ \forall\, a, b \in \mathbb{N}$.

   Proof:
   It will be a proof by strong induction on b.

   <u>Base case</u>:
   For $b = 0, multiply(a, b) = 0 = a * 0 = a * b$

   <u>Inductive hypothesis</u>:
   Let $multiply(a, b) = a * b\ \ \forall\, a, b \in \mathbb{N}, b \leq k$

   <u>Inductive Step</u>:
   Let $b = k + 1(\mathrm{base}10) = b_n b_{n-1} \dots b_2 b_1 b_0 (base2)$.

   Note:
   All the numbers defined ahead would be in base 2 unless specified explicitly.

If $b_0 = 0$:
Then $multiply(a, b)$ returns:

$$\sum_{i=0}^{n} q_i = q_0 + q_1 + \cdots + q_n = c_p c_{p-1} \dots c_2 c_1 c_0$$

Where $q_i = join(p_i, 0, i)\ \ \forall\, i \in \mathbb{N}, i \leq n$ and $\mathrm{p_i = mult(a, b_i)\ \forall\, i \in \mathbb{N}, i \leq n}$

Note that $q_0 = join\left(p_{0,} 0{,}0\right) = p_0 = mult(a, b_0) = 0$

Moreover, for the rest of the q_i, $last(q_i) = 0$. Therefore $c_0 = 0$.

Now, $multiply\left(a, \frac{b}{10}\right) = multiply(a, b_n b_{n-1} \dots b_2 b_1)$

$$= \sum_{i=0}^{n} q'_i = q'_0 + \cdots + q'_n$$

$$= c'_{p'} c'_{p'-1} \dots c'_1 c'_0$$

$$= a * \frac{b}{10}\ (using\ inductive\ hypothesis)$$

$$\text{As } \frac{b}{10} = \frac{k+1}{2} (base\ 10) \leq k$$

Note that as b_i is same, therefore:

$$q'_i = \frac{q_{i+1}}{10} \Rightarrow p' = p - 1 \Rightarrow c'_i = c_{i+1}\ \forall\, i \in \mathbb{N}, i \leq\ p' - 1$$

Therefore, $multiply(a, b) = 10 * c_p c_{p-1} \dots c_1$

$$= 10 * c'_{p\prime} c'_{p\prime-1} \dots c'_1 c'_0$$

$$= 10 * multiply\left(a, \frac{b}{10}\right)$$

$$= 10 * a * \frac{b}{10} = a * b$$

If $b_0 = 1$:

Then $multiply(a, b)$ returns:

$$\sum_{i=0}^{n} q_i = q_0 + q_1 + \cdots + q_n = c_p c_{p-1} \ldots c_2 c_1 c_0$$

Where $q_i = join(p_i, 0, i)\ \ \forall\, i \in \mathbb{N}, i \leq n$ and $\mathrm{p_i} = \mathrm{mult(a, b_i)}\ \forall\, \mathrm{i} \in \mathbb{N}, \mathrm{i} \leq \mathrm{n}$

Note that $q_0 = join\left(p_{0,} 0{,}0\right) = p_0 = mult(a, b_0) = a$

$$multiply(a, \mathrm{b} - 1) = multiply(a, b_n b_{n-1} \ldots b_2 b_1 0)$$

$$= \sum_{i=0}^{n} q'_i = q'_0 + \cdots + q'_n$$

$$= q'_1 + \cdots + q'_n, as\ q_0 = 0$$

$$= c'_{p'} c'_{p'-1} \ldots c'_1 c'_0$$

$$= a * (\mathrm{b} - 1)\ (using\ inductive\ hypothesis)$$

Note that $p' = p$ and $q'_i = \mathrm{q_i}\ \forall\, i \in \mathbb{N}, i \neq 0, i \leq p$
Therefore:

$$multiply(a, b) = \sum_{i=0}^{n} q_i = q_0 + q_1 + \cdots + q_n$$

$$= a + \sum_{i=1}^{n} q_i$$

$$= a + \sum_{i=1}^{n} q'_i$$

$$= a + a * (b - 1)$$

$$= a * b$$

Therefore, by Principle of Mathematical Induction,

$$multiply(a, b) = a * b\ \forall\ a, b \in \mathbb{N}$$

■

Part 2

Addition Algorithm Helpers

1. The function last(a) returns the last digit of the given binary number.

$$\text{last(a)} = a_0$$

2. The function higher(a) returns the binary number left after removing the last digit.

$$\text{higher(a)} = \begin{cases} 0, & \text{if m} = 0 \\ a_m a_{m-1} \dots a_2 a_1, & \text{otherwise} \end{cases}$$

3. The function add1(a_i, b_i, c) returns the sum of the three bits $a_i + b_i + c$.

$$add1(a_i, b_i, c) = \begin{Bmatrix} 00, & \text{if all are 0} \\ 01, & \text{if any one is 1} \\ 10, & \text{if any two are 1} \\ 11, & \text{if all are 1} \end{Bmatrix}$$

4. The function $addc(a, b, c)$ returns the sum $a + b + c$.

$$addc(a,b,c) = \begin{Bmatrix} b & if\ c = a = 0 \\ a & if\ c = b = 0 \\ addc(b, 1{,}0) & if\ a = 0\ and\ c = 1 \\ addc(a, 1{,}0) & if\ b = 0\ and\ c = 1 \\ join\left(s, d_{0,} 1\right) & otherwise \end{Bmatrix}$$

Where:

$$d_1 d_0 = add1(last(a), last(b), c)$$

$$s = addc(higher(a), higher(b), d_1)$$

So, the sum of a and b is $add(a, b) = addc(a, b, 0)$.

Time Complexity

1. The $last()$ function is executed immediately without any delay.

2. The time complexity of $join(), mult()$ and $higher()$ is $O(1)$ as in all three, only single bit comparison is done a constant number of times whenever the function is called.

3. The time complexity of $add1(a_i, b_i, c)$ is $O(1)$ as always 3 comparisons are used, each one for the three input bits.

4. <u>Time complexity for add</u>:

   Case I: $m \geq n$
   For addition of every i^{th} bit, where $i \leq n$, it always takes time proportional to $add1(a_i, b_i, c) + 3$.
   Note that when it checks for the $n + 1^{th}$ bit, then the function stops in the best case, as b becomes 0. In the worst case, it goes till the $m + 1^{th}$ step. Therefore, for the former, the order is $O(n)$. But for the latter, it is $O(m)$. As $m \geq n$, so the time complexity would be $O(m)$.

   Case II: $n > m$
   For addition of every i^{th} bit, where $i \leq n$, it always takes time proportional to $add1(a_i, b_i, c) + 3$.
   But here, the function always runs for $n + 1$ times. Therefore, the time complexity is $O(n)$.

   Therefore, by combining the two cases, we get that the time complexity = $O(max(m, n))$.

5. <u>Time complexity for $multiply()$</u>:

   In the $multiply()$ function, $mult()$ runs n times. Therefore, the time complexity for this step is $O(n * 1) = O(n)$.

Now, for the addition steps, it would call binary addition n times. And for each addition call, the time complexity would be $O(max(n, m))$. Therefore, the time complexity for this step would be $O(n * max(n, m))$.

Therefore, the overall time complexity is:

$$O(n * max(n, m)) + O(n) \ = \ O(n * max(m, n))$$

Therefore, the time complexity of the algorithm is quadratic.

Space Complexity:

1. The space complexity for $last()$ and $higher()$ is $O(q)$, where q is the number of bits of input.

2. The space complexity for $mult() = O(q)$ as the input has $q + 1$ bits.

3. The space complexity for $join() = O\left(max(q, log_2 k)\right) = O(q)$, where q is number of bits of the input number.

4. The space complexity of $add1() = O(1)$.

5. Space complexity for $add(a, b)$:

   Without loss of generality, let us suppose that $m \geq n$.
   The recursion for space complexity is:
   $$s(m, \mathrm{n}) = s(m - 1, n - 1) + m + n + 6$$

   As $s(m, 0) = O(m)$, therefore solving the above recursion relation, we get:
   $$s(m, n) - s(m, 0) = \frac{m(m+1)}{2} + \frac{n(n+1)}{2} + 6n - \frac{(m-n)(m-n+1)}{2}$$

   $$s(m, n) - 2m = mn + 7n$$

   $$s(m, n) = mn + 7n + 2m$$

   Therefore, the space complexity is $O(m * n + 7\mathrm{n} + 2\mathrm{m}) = O(m * n)$.

6. Space complexity of $multiply()$:

   In the $multiply()$ function, the $mult()$ function runs n times, but every run happens only after the previous run has finished. There are n new variables formed, each storing at maximum m digits. Therefore, the order of space complexity is $O(m * n)$.

   For the next $join()$ step, the join function uses at maximum space of $O(m)$.

For the new variables q_i, the space required is:

$$O(nm + n(n+1)/2) = O(nm + n^2) = O\big(n * max(n, m)\big)$$

For the addition, the binary addition would require at max space:

$$O((m+n) * (m+n)) = O\left(\big(max(m, n)\big)^2\right)$$

For the storage of product, the space required is $O(max(m, n))$.

Therefore, the space complexity is:

$$O\big(max(n, m)\big) + O\left(\big(max(m, n)\big)^2\right) + O\big(n * max(n, m)\big) + O(m * n)$$

$$= O\left(\big(max(m, n)\big)^2\right)$$

Therefore, the space complexity of the algorithm is quadratic.

New Iterative Algorithm

For this, a new helper function is needed:

The function $lower()$ removes the first digit of the number and returns the rest.

$$lower(b) = \begin{cases} 0, & \text{if b} = 0 \\ 0, & \text{if n} = 0 \\ b_{n-1}b_{n-2} \dots b_2 b_1 b_0, & \text{otherwise} \end{cases}$$

Now, the iterative algorithm is:

$$itemul(a, b, result) = \begin{cases} result & \text{if } b = 0 \ or \ a = 0 \\ itemul(\text{a}, r, \text{s}) & \text{if } last(b) = 0 \\ itemul(\text{a}, r, \text{s} + a) & otherwise \end{cases}$$

Where:

$$s = join(\text{result}, 0,1)$$
$$r = \text{lower}(b)$$

Therefore $muli(a, b) = itemul(a, b, 0)$.