The size of integers that a computer can take

Weihu Hong, Department of Mathematics, Clayton State University, 2000 Clayton State Blvd, Morrow, GA 30260, Email: weihuhong@clayton.edu
Mingshen Wu, Dept. of Mathematics, Statistics, and Computer Science, University of Wisconsin-Stout, Menomonie, WI 54751

Abstract. The size of integers that a computer can take depends on compilers as well as algorithms. We will discuss the sizes of integers for any given compiler with different algorithms.

Keywords: Size of integers.

§1. Introduction

How large is an integer that a computer can take? Certainly it depends on what software that you are going to use. For instance, the largest unsigned long integer in C++ is $2^{32} - 1$ (see [1]), while it is $2^{64} - 1$ in C# (see [2]). If a variable in C++ is declared to be of unsigned long integer type, it has the maximum value of 4,294,967,295. It will result in an integer overflow when a program in C++ is to compute an integer value larger than that maximum value unless your algorithm takes a special care. Algorithms discussed in [3] will be able to deal with very large integers. In this note, we will discuss the sizes of integers that a computer can deal within a given compiler with different algorithms.

§2. Estimates based on Algorithms

Let $C$ be a given compiler, $M$ the maximum value of the integer that the compiler can take, $S$ the size of integers that a computer can deal with. As mentioned in [3] that although $M$ is relatively small, it is possible to write algorithms that can deal with very large integers. Since the $M$ depends on the compiler $C$, the $S$ is a function of the compiler $C$ and the algorithm $\alpha$, that is, $S = S(C, \alpha)$. We will give estimates of the $S$ for the algorithms given in [3].

Proposition 2.1. Given a compiler $C$. If an algorithm $\alpha$ is the addition in digit by digit (see [3]), then

$$S(C, \alpha) = 5 \times 10^M - 1 = 499\ldots9_{M-1}.$$

Proof: Let $x$ and $y$ be positive integers with $n$ digits. Declare them as type of string. Since the maximum length of strings is $M$, the largest value of $x$ and $y$ should be $499\ldots9_{M-1}$ such that the sum of them will be a string of length $M$ if we perform the addition digit by digit. The proof is complete.

Example 2.1. Let the complier $C$ be the C++ and the algorithm $\alpha$ is the addition in digit by digit. The $M$ is $2^{32} - 1 = 4,294,967,295$. Therefore, the largest integer that a computer can add via the algorithm $\alpha$ with the complier C++ is $S(C +, \alpha) = 5 \times 10^{4,294,967,295} - 1 = 499\ldots9_{4,294,967,294}$. Surely it is a very large number.
**Proposition 2.2.** Given a compiler $C$. If an algorithm $\alpha$ is an addition in block by block (see [3]), then

$$S(C, \alpha) = A_{M-1} \times 10^{(M-1)k} + A_{M-2} \times 10^{(M-2)k} + A_{M-3} \times 10^{(M-3)k} + \cdots + A_1 \times 10^k + A_0$$

where $A_j \leq \frac{M}{2}$, $j = 0, 1, 2, \cdots, M - 1; k := \text{the length of } M$.

**Proof:** Let $x$ and $y$ be positive integers of the form

$$x = A_{M-1} \times 10^{(M-1)k} + A_{M-2} \times 10^{(M-2)k} + A_{M-3} \times 10^{(M-3)k} + \cdots + A_1 \times 10^k + A_0$$

$$y = B_{M-1} \times 10^{(M-1)k} + B_{M-2} \times 10^{(M-2)k} + B_{M-3} \times 10^{(M-3)k} + \cdots + B_1 \times 10^k + B_0$$

where $A_j, B_j \leq \frac{M}{2}$, $j = 0, 1, 2, \cdots, M - 1; k := \text{the length of } M$.

Since the total length of each number exceeds $M$, we decompose them into blocks of regular integers. We can add each pair of the correspondent integers in regular addition and then concatenate the sums of all pairs of the correspondent integers in the original order. This completes our proof.

**Example 2.2.** Let the compiler $C$ be the C++ and the algorithm $\alpha$ is the addition in block by block. The $M$ is $2^{32} - 1 = 4,294,967,295$. Therefore, the largest integers that a computer can add via the algorithm $\alpha$ is

$$S(C + , \alpha) = 2147483647 \times 10^{42949672940} + 2147483647 \times 10^{42949672930} + \cdots + 2147483647 \times 10^{10} + 2147483647$$

It is obvious that the addition in block by block can deal with larger number than the addition in digit by digit.

**Proposition 2.3.** Given a compiler $C$. If an algorithm $\alpha$ is the subtraction in digit by digit (see [3]), then the value of $S$ can be as large as

$$S(C, \alpha) = 10^M - 1 = \underbrace{99\cdots9}_M$$

**Proof:** Let $x$ and $y$ be positive integers with $n$ digits. Declare them as type of string. Since the maximum length of strings is $M$, the largest value of $x$ and $y$ should be $\underbrace{99\cdots9}_M$. Therefore, the difference of the two numbers will be a string of length at most $M$ if we perform the subtraction digit by digit. The proof is complete.

**Example 2.3.** Let the compiler $C$ be the C++ and the algorithm $\alpha$ is the subtraction in digit by digit. The $M$ is $2^{32} - 1 = 4,294,967,295$. Therefore, the largest integer that a computer can subtract via the algorithm $\alpha$ is

$$S(C + , \alpha) = 10^{4,294,967,295} - 1 = \underbrace{99\cdots9}_{4,294,967,295}$$

**Proposition 2.4.** Given a compiler $C$. If an algorithm $\alpha$ is the subtraction in block by block (see [3]), then

$$S(C, \alpha) = A_{M-1} \times 10^{(M-1)k} + A_{M-2} \times 10^{(M-2)k} + A_{M-3} \times 10^{(M-3)k} + \cdots + A_1 \times 10^k + A_0$$

where $A_j \leq M$, $j = 0, 1, 2, \cdots, M - 1; k := \text{the length of } M$. 


Proof: Let \( x \) and \( y \) be positive integers of the form

\[
x = A_{M-1} \times 10^{(M-1)k} + A_{M-2} \times 10^{(M-2)k} + A_{M-3} \times 10^{(M-3)k} + \cdots + A_1 \times 10^k + A_0
\]

\[
y = B_{M-1} \times 10^{(M-1)k} + B_{M-2} \times 10^{(M-2)k} + B_{M-3} \times 10^{(M-3)k} + \cdots + B_1 \times 10^k + B_0
\]

where \( A_j, B_j \leq M, j = 0, 1, 2, \ldots, M - 1; k := \text{the length of } M. \)

Since the total length of each number exceeds \( M \), we decompose them into blocks of regular integers. We can subtract each pair of the correspondent integers in regular subtraction and then concatenate the differences of all pairs of the correspondent integers in the original order. This completes our proof.

Example 2.4. Let the compiler \( C \) be the C++ and the algorithm \( \alpha \) is the subtraction in block by block. The \( M \) is \( 2^{32} - 1 = 4,294,967,295 \). Therefore, the largest integers that a computer can subtract via the algorithm \( \alpha \) is

\[
S(C + , \alpha) = 4,294,967,295 \times 10^{42949672940} + 4,294,967,295 \times 10^{42949672930}
\]

\[
+ \cdots + 4,294,967,295 \times 10^0 + 4,294,967,295
\]

Proposition 2.5. Given a compiler \( C \). If an algorithm \( \alpha \) is the multiplication in digit by digit (see [3]), then the value of \( S \) can be determined by the two multipliers \( x \) and \( y \) such that

\[
\log x + \log y \leq M - 1.
\]

Proof: Let \( x \) and \( y \) be positive integers. Declare them as type of string. Since the maximum length of strings is \( M \), the largest value of \( x \cdot y \) should be \( 99 \cdots 9 \). Therefore, \( \log x + \log y \leq M - 1 \). The proof is complete.

Example 2.5. Let the compiler \( C \) be the C++ and the algorithm \( \alpha \) is the multiplication in digit by digit. The \( M \) is \( 2^{32} - 1 = 4,294,967,295 \). Therefore, the largest integers that a computer can multiply via the algorithm \( \alpha \) is determined by the two multipliers \( x \) and \( y \) such that

\[
\log x + \log y \leq 2^{32} - 2 = 4,294,967,293.
\]

Proposition 2.6. Given a compiler \( C \). If an algorithm \( \alpha \) is the multiplication in block by block (see [3]), then the value of \( S \) can be as large as

\[
S(C, \alpha) = A_p \times 10^{pk} + A_{p-1} \times 10^{(p-1)k} + A_{p-2} \times 10^{(p-2)k} + \cdots + A_1 \times 10^k + A_0
\]

where \( A_i \leq \sqrt{M \over 2}, (0 \leq i \leq p); k := \text{the length of } M \over 2 ; p \leq (M-1)/2 \).

Proof: Let \( x \) and \( y \) be positive integers of the form

\[
x = A_p \times 10^{pk} + A_{p-1} \times 10^{(p-1)k} + A_{p-2} \times 10^{(p-2)k} + \cdots + A_1 \times 10^k + A_0
\]

\[
y = B_q \times 10^{qk} + B_{q-1} \times 10^{(q-1)k} + B_{q-2} \times 10^{(q-2)k} + \cdots + B_1 \times 10^k + B_0
\]

where \( A_i \cdot B_j \leq M \over 2, (0 \leq i \leq p, 0 \leq j \leq q); k := \text{the length of } M \over 2 ; p + q \leq M - 1.\)
We can perform multiplications of $A_i$ and $B_j$ in regular multiplication and write the product

$$xy = \sum_{i=0}^{p+q} \left( \sum_{j=1}^i A_j B_j \right)10^i$$

where the summation can be done in block by block. This completes the proof.

**Proposition 2.7.** Given a compiler $C$. If an algorithm $\alpha$ is the division in digit by digit (see [3]), then the value of $S$ can be as large as

$$S(C, \alpha) = 10^M - 1 = \underbrace{9\ldots9}_M.$$  

*Proof:* Let $x$ and $y$ be positive integers with $n$ digits. Declare them as type of string. Since the maximum length of strings is $M$, the largest value of $x$ and $y$ should be $\underbrace{9\ldots9}_M$. The proof is complete.

§3. Conclusion

In order to avoid any error of integer overflow, it is very important to know the size of integers that a computer can take for a given compiler $C$ and an algorithm $\alpha$. By adopting the function notation $S(C, \alpha)$ for the size of integers that a computer can take for the compiler $C$ and the algorithm $\alpha$, we can better understand the fact that it depends on both the compiler $C$ and the algorithm $\alpha$. To increase the size of integers, we can improve either the algorithm $\alpha$ or the compiler $C$. As the computer technology is improving, the size of the integers that a computer can take will be increasing.

**References**

