International Research Journal of Advanced Engineering and Science
ISSN (Online): 2455-9024

Big-Integer

Riya Tyagi[1], Anmol Agarwal[2]
[1, 2]Dept. of Computer Science and Engineering, Graphic Era University, Dehradun, Uttarakhand, India-248002

Abstract— *In the world of the stupendous amount of data, it is necessary to perform operation or calculation on such data rapidly and accurately. Data in itself is [1] not small, but also includes the characteristics of huge variety and high velocity which causes difficulty in operational tools and algorithms because they are limited to perform operation on a certain amount of data and such data usually comes in numerical form, which consists of an infinite number of digits [10]. Daily data are generated in enormous amount (approx. billion TB) from different sources like the stock market, big-data, data-warehouses, big IT companies, Internet, banking, business market and many more which need to perform operations on the data quickly and accurately for the better growth and smooth functioning in their areas. Due to continuously increasing data, it had become difficult to calculate mathematical operation on data in the given time. So, here we had come up with a solution, as an algorithm for performing mathematical operation [2] on such data like addition, subtraction and multiplication which is implementing in the middle-level language C++ with the help of data structure "link-list" and "dynamic memory allocation" of the data. This type of data is called Big-integer which having 'N' number of digits. Our algorithm performs the operation on this data till system or device memory becomes full and gives the result within 0-1 second with almost 100% accuracy. This paper major objective behind this is, to introduce an algorithm in C++ for obtaining the rapid result which must be understandable and can be implemented easily in every field.*

Keywords— *Link-list, data structure, dynamic memory, and data type.*

I. Introduction

In this research paper, we are presenting a very optimize and understandable algorithm for handling a large amount of data [11] which is very big in terms of numbers. This methodology is already there but we decide to use C++ so that everyone could understand easily and able to identify the methodology and data processing behind the algorithm [3] which is helpful to make required changes for everyone in the future. In this algorithm, we use a linear data structure called "Link-List" which stores the data node by node in the memory and each node contains the data and the address of his next node. The last node of the link-list contains the NULL as its address field and the first node of the link-list is called the head of the link-list. There are three kinds of the link-list present in programming. The first one is a singly link-list in which we can move only one direction, the second one is doubly link-list in which we can move both the direction forward and backward and the third one is circular link-list in which we can move both the direction as well as we can move circularly. The main reason behind the usage of link-list data structure is to store the data dynamically not statically or continuous so that we can create the nodes as much as memory present in the CPU [4] and it decreases the wastage of memory which is a major aspect of dynamic programming. The one disadvantage of the link-list is that we can't go directly on a specific node, we need to go one by one. In this algorithm, we used a singly link-list to store the numbers. The dynamic memory allocation is done by malloc, realloc, calloc and free. We used "malloc" to allocate the specified number of bytes or block of memory [9] in the heap manually in the runtime, takes a value as argument which is the size of the memory block. This method returns a void pointer of that block of memory of not-defined data type through which we access the allocated memory and when we don't need that block of memory we use free to deallocate the memory which decreases memory wastage. These dynamic programming methods come under the "cstdlib" header file in C++. We use type-casting to use dynamic memory allocation methods on different datatypes.

II. Construction and Working

In our algorithm for the creation of dynamic memory we use malloc with the data structure "link-list".

```
struct node{
        int info;
        struct node *next;
};
```

The structure named node contains two fields. The first field is info that stores the digit of the number and the second field is next which store the address of the next memory block. For allocating the dynamic memory we use-

```
p=(node *)malloc(sizeof(node));
```

This p pointer store the address of the memory block which is created by malloc. To store the information and for performing operations in the memory block, we create three methods as insert, len and reverse which used for inserting the digit of the number, calculate the total length of the number and reverse the number respectively. Insert function has two arguments as head and d. "Head" is a node type variable that points to the first node of the link-list through-out the program and "d" is an integer type variable that stores the digits of the number. This function returns node type value. Len function has one argument as head and returns integer type value. The reverse function has also one argument as head and return node type value.

1- node *insert(node *head,int d)
2- int len(node *head)
3- node *reverse(node *head)

In our algorithm, we use string data type for storing the number and typecast them into an integer with the help of ASCII values for the mathematical operations like addition, subtraction and multiplication. We use switch for performing them separately and for this we use three methods like add, sub, and mul which perform addition, subtraction, and multiplication of two numbers respectively.

1- node *add(node *head1,node *head2)
2- node *sub(node *head1,node *head2)
3- node *mul(node *head1,node *head2)

III. Addition

node *add(node *head1,node *head2)

This method performs an addition operation of two numbers. We use two link lists for storing the two numbers. It has two arguments as head1 and head2. Head1 points to the first node of the first number and head2 points to the first node of the second number. Both are of the node type. We access both the link list at the same time and use two more node type pointer i and j. "I" for the first number and j for the second number. Both are a point at the beginning of the number or the link-list and we access the link-list from start to end of the link-list. For handle the carry operation we use integer type variable "c" whose value can change according to the calculation. If the numbers are negative or either one number is positive or the second number is negative, our algorithm also gives the accurate result of it. For this methodology, our algorithm also checks negative and positive value whether the entered number is positive or negative and performs operation accordingly.

```
while(i!=NULL&&j!=NULL)
r=i->info+j->info+c;
```

If the value of r is greater than and equal to 10, we'll get the carry by subtraction 10 from the value of r otherwise we set the carry value as 0.

```
if(r>=10)
{
r=r-10;
c=1;
}
```

If the length of both the numbers is different then we access the larger number link-list and repeat the same procedure as above but this time we only add i or j values into carry and rest of it, we copy to output link-list.

```
if(j==NULL)
{
while(i!=NULL)
{
r=i->info+c;
if(r>=10)
{
r=r-10;
c=1;
}
else
c=0;
head3=insert(head3,r);
i=i->next;
}
}
else
{
while(j!=NULL)
{
r=j->info+c;
if(r>=10)
{
r=r-10;
c=1;
}
else
c=0;
head3=insert(head3,r);
j=j->next;
}
}
```

IV. Subtraction

node *sub(node *head1,node *head2)

This method perform subtraction operation of two numbers. We use two link-list for storing the two numbers. It has two arguments as head1 and head2. Head1 is points to the first node of the first number and head2 points to the first node of the second number. Both are of node type. We access both the link list at the same time and use two more node type pointer i and j. "i" for the first number and j for the second number. Both are point at the beginning of the number or the link-list and we access the link-list from start to end. After this we check the node value for the borrow operation. If first node is having bigger value than second node then it perform normal subtraction but if it is less than we add 10 [7] to the first node and subtract 1 from the next node of the respective link list. It can give the correct answer with-in 1 sec.

```
if(i->info<j->info)
{
k=i;
if(i->next==NULL)
{
r=j->info-i->info;
head3=insert(head3,r);
}
else
{
i->info=i->info+10;
while(k->next->info==0)
{
k=k->next;
k->info=9;
}
k->next->info=k->next->info-1;
r=i->info-j->info;
head3=insert(head3,r);
}
}
else
{
r=i->info-j->info;
head3=insert(head3,r);
}
```

If there is no value in the second link-list then first link-list value normally added to the output link-list.

```
if(j==NULL&&i!=NULL)
{
while(i!=NULL)
{
head3=insert(head3,i->info);
i=i->next;
}
}
```

V. Multiplication

node *mul(node *head1,node *head2)

This method performs the multiplication operation of two numbers. We use two link-list for storing the two numbers. It has two arguments as head1 and head2. Head1 points to the first node of the first number and head2 points to the first node of the second number. Both are of the node type. We access both the link list at the same time and use two more node type pointer i and j. "i" for the first number and j for the second number. Both are the point at the beginning of the number or the link-list and we access the link-list from start to end and for the carry handler, we use integer variable ‘c' which is initialized with zero [6]. If the multiplication of two digits is greater than 10 then we find the carry from that number by dividing it by 10 [8] and remainder. After it, we proceed with the normal calculation.

```
while(j!=NULL)
{
i=head1;
c=0;
head4=NULL;
head5=NULL;
if(j==head2)
{


while(i!=NULL)
{
r=(j->info*i->info)+c;
if(r>=10)
{
c=r/10;
r=r%10;
}
else
c=0;
head3=insert(head3,r);
i=i->next;
}
```

Same for all digits we calculate the multiplication of each digit from another number and store the result in the third link list as multiplication.

```
else
{
for(k=0;k<z;k++)
head4=insert(head4,0);
while(i!=NULL)
{
r=(j->info*i->info)+c;
if(r>=10)
{
c=r/10;
r=r%10;
else
c=0;
head4=insert(head4,r);
i=i->next;
}
if(c!=0)
head4=insert(head4,c);
z++;
}
if(head4!=NULL)
{
head3=reverse(head3);
head4=reverse(head4);
head5=add(head3,head4);
head3=head5;
}
j=j->next;
}
return(head3);
}
```

VI. Characteristics

- Our algorithm is very user-friendly and easy to understand.
- It gives the correct answer in 1 second.
- This algorithm can be used in many research fields, stock markets, share markets, etc.
- It is very efficient because it can be run on any no. of digits.
- It takes less memory to run the program which makes it more reliable.
- It is 100% accurate.
- It is time-saving and runs in real-time.

VII. Conclusion

This is necessary to perform operation data rapidly and accurate and our algorithm becomes more reliable with these characteristics. It can operate TB of data easily [5] which become more helpful in business, stock market, trading, Internet, data-warehouses or banking, etc. So, our algorithm is very user-friendly, accurate and time-saving to calculate big mathematical problems.

References

[1] S. A. Cook, On the minimum computation time of functions [Ph.D. thesis], Department of Mathematics, Harvard University, May 1966.
[2] Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *USSR Academy of Sciences*, vol. 145, pp. 293–294, 1962
[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, Introduction to Algorithms, MIT Press, 2000.

[4] Schonhage and V. Strassen, “Schnelle Multiplikation großerZahlen,” *Computing in Science & Engineering*, vol. 7, pp. 139–144, 1971.

[5] P. L. Montgomery, “Modular multiplication without trial divi-sion,” *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, 1985.

[6] T.-J. Chang, C.-L. Wu, D.-C. Lou, and C.-Y. Chen, “A low-complexity LUT-based squaring algorithm,” *Computers &Mathematics with Applications*, vol. 57, no. 9, pp. 1494–1501, 2009

[7] N. S. Szabo and R. I. Tanaka, Residue Arithmetic and Its Applications to Computer Technology, McGraw-Hill, 1967.

[8] D. Zuras, “On squaring and multiplying large integers,” in *Proceedings of the IEEE 11th Symposium on Computer Arithmetic*, pp.260–271, July1993

[9] M. Sadiq and J. Ahmed, “Complexity analysis of multiplication of long integers,” *Asian Journal of Information Technology*, vol. 5, no. 2, 2006

[10] D. Zuras, “More on squaring and multiplying large integers,” *IEEE Transactions on Computers*, vol. 43, no. 8, pp. 899–908, 1994.

[11] D. M. Gordon, “A survey of fast exponentiation methods,” *Journal of Algorithms*, vol. 27, no. 1, pp. 129–146, 1998.