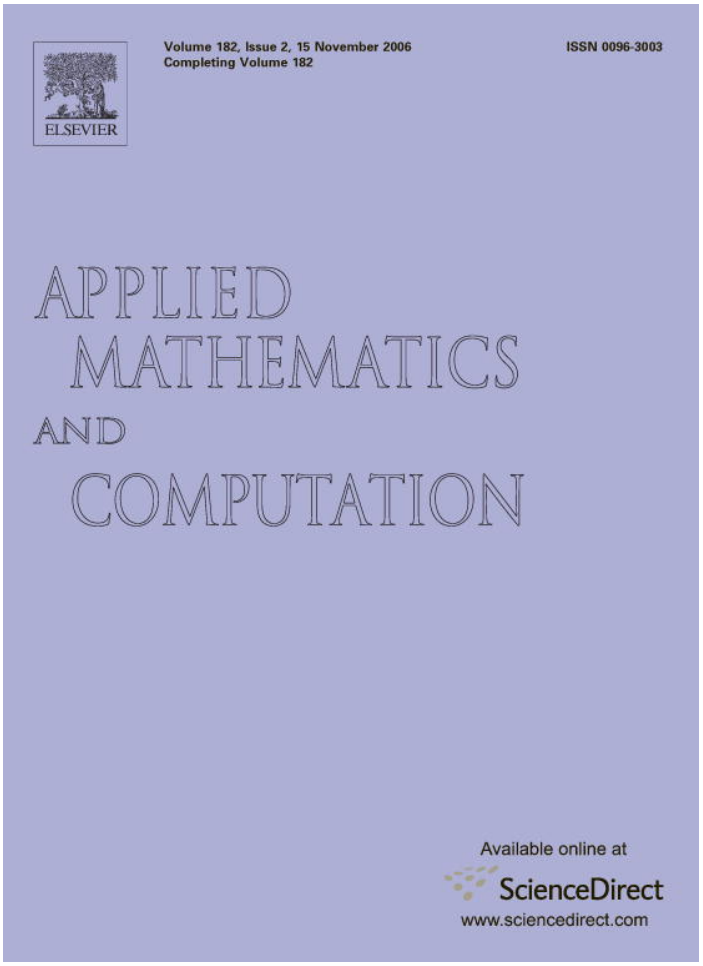

Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation 182 (2006) 977–986
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

DNA ternary addition

Wenxia Li [a,c,*], Dongmei Xiao [b,c], Lin He [c]

[a] Department of Mathematics, East China Normal University, Shanghai 200062, PR China
[b] Department of Mathematics, Shanghai Jiao Tong University, Shanghai 200030, PR China
[c] Bio-X DNA Computer Consortium, Shanghai Jiao Tong University, Shanghai 200030, PR China

Abstract

In this paper, we first show a DNA representation of a ternary number with digit set $\{-1,0,1\}$, which features its address and each bit position. Based on this DNA representation, operations for value assigning and bit position shifting are proposed. The algorithm of DNA computing for adding two ternary integers is presented. The algorithm works in $O(m)$ steps for adding two ternary integers of m bits.

Keywords: DNA representation; Algorithm; DNA computing; Ternary integers

1. Introduction

Biological macromolecules can be used for storing information and biochemical reactions, like nucleic acid hybridization and enzyme reactions, can be used to solve algorithmic problems. Since a vast number of biochemical reactions can take place at many molecules simultaneously, parallel computations involving millions of operations seem possible. Thus computation with DNA molecules, that is, DNA computing, has considerable attention as one of non-silicon based computing. The DNA has two important features, which are Watson-Crick complementarity and massive parallelism. As the first work for DNA computing, Adleman [1] used the features to present an idea of solving the Hamiltonian path problem, an NP-complete problem which usually needs exponential time on a silicon based computer, of size n in $O(n)$ steps using DNA molecules. In recent years methods for solving several well known NP-complete problems have been proposed, and have been performed on small examples in many cases [9,10], etc. Meanwhile, procedures for primitive operations, such as logic or arithmetic operations, are presented for applying DNA computing on a wide range of problems, e.g., refer to [3] and references therein. Some procedures are proposed for an addition of two binary numbers using DNA molecules [3,4]. In this paper, we present an algorithm of DNA computing

* Corresponding author.
E-mail address: wxli@math.ecnu.edu.cn (W. Li).

0096-3003/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2006.04.051

for adding two ternary integers. We take -1, 0 and 1 as the base digits for the ternary expansion of an integer number so that both positive and negative integers can be represented in a unified form.

The rest of this paper is organized as follows. In Section 2, the Adleman-Lipton model is introduced in detail. In Section 3, based on that by Fujiwara et al. [3], a DNA representation of a ternary integer number is shown, which features its address and each bit position and makes operations for value assigning and bit position shifting available. We present an algorithm of DNA computing for adding two ternary integers. Conclusion is arranged in Section 4.

2. The Adleman-Lipton model

DNA is the major information storage molecule in living cells, and billions of years of evolution have tested and refined both this wonderful informational molecule and highly specialized enzymes that can either duplicate the information in DNA molecules or transmit this information to other DNA molecules.

A DNA (deoxyribonucleic acid) is a polymer, which is strung together from monomers called deoxyribonucleotides [11]. Distinct nucleotides are detected only with their bases. Those bases are, respectively, abbreviated as A (adenine), G (guanine), C (cytosine) and T (thymine). Two strands of DNA can form (under appropriate conditions) a double strand, if the respective bases are the Watson-Crick complements of each other—A matches T and C matches G; also 3′ end matches 5′ end, e.g., the singled strands 5′ACCTGGATGTAA3′ and 3′TGGACCTACATT5′ can form a double strand. We also call the strand 3′TGGACCTACATT5′ as the complementary strand of 5′ACCTGGATGTAA3′ and simply denote 3′TGGACCTACATT5′ by $\overline{\text{ACCTGGATGTAA}}$. The length of a single stranded DNA is the number of nucleotides comprising the single strand. Thus, if a single stranded DNA includes 20 nucleotides, it is called a 20 mer. The length of a double stranded DNA (where each nucleotide is base paired) is counted in the number of base pairs. Thus, if we make a double stranded DNA from a single stranded 20 mer, then the length of the double stranded DNA is 20 base pairs, also written as 20 bp.

The Adleman-Lipton model: A (test) tube is a set of molecules of DNA (i.e., a multi-set of finite strings over the alphabet $\{A, C, G, T\}$). Given a tube, one can perform the following operations:

(1) *Merge*: Given two test tubes T_1,T_2, *Merge*(T_1,T_2) stores the union $T_1 \cup T_2$ in T_1;
(2) *Copy*: Given a test tube T_1, *Copy*(T_1,T_2) produces a test tube T_2 with the same contents as T_1;
(3) *Detect*: Given a test tube T, *Detect*(T) outputs "yes" if T contains at least one strand, otherwise, *Detect*(T) outputs "no";
(4) *Separation*: Given a test tube T_1 and a set of strings X, *Separation*(T_1, X, T_2) removes all single strands containing a string in X from T_1, and produce a test tube T_2 with the removed strands;
(5) *Selection*: Given a test tube T_1 and an integer L, *Selection*(T_1, L, T_2) removes all strands, whose length is L, from T_1, and produces a test tube T_2 with the removed strands;
(6) *Cleavage*: Given a test tube T and a string of two symbols $\sigma_0\sigma_1$, *Cleavage*($T, \sigma_0\sigma_1$) cuts each double strand containing $\begin{bmatrix} \sigma_0\sigma_1 \\ \overline{\sigma_0\sigma_1} \end{bmatrix}$ in T into two double strands as follows:

$$\begin{bmatrix} \alpha_0\sigma_0\sigma_1\beta_0 \\ \alpha_1\overline{\sigma_0\sigma_1}\beta_1 \end{bmatrix} \Rightarrow \begin{bmatrix} \alpha_0\sigma_0 \\ \alpha_1\overline{\sigma_0} \end{bmatrix}, \begin{bmatrix} \sigma_1\beta_0 \\ \overline{\sigma_1}\beta_1 \end{bmatrix};$$

(7) *Annealing*: Given a test tube T, *Annealing*(T) produces all feasible double strands in T. (The produced double strands are still stored in T after *Annealing*);
(8) *Denaturation*: Given a test tube T, *Denaturation*(T) dissociates each double strand in T into two single strands;
(9) *Discard*: Given a tube T, *Discard*(T) will discard the tube T;
(10) *Append*: Given a tube T and a short DNA singled strand Z, *Append*(T,Z) will append Z onto the end of every strand in the tube T;
(11) *Read*: Given a tube T, the operation is used to describe a single molecule, which is contained in the tube T. Even if T contains many different molecules each encoding a different set of bases, the operation can give an explicit description of exactly one of them.

Since these eleven manipulations are implemented with a constant number of biological steps for DNA strands [11], we assume that the complexity of each manipulation is O(1) steps.

3. Procedure for adding ternary integers using DNA molecules

For any $x \in \mathbf{R}$, x can be represented as

$$x = (a_k a_{k-1} \cdots a_1 a_0 . a_{-1} \cdots)_3 = \sum_{n=0}^{k} a_n 3^n + \sum_{n=1}^{\infty} a_{-n} 3^{-n}, \quad a_n \in \{-1, 0, 1\}.$$

In particular, for any integer $n \in \mathscr{Z}$ there exists a finite sequence $(a_{m-1} a_{m-2} \cdots a_0)$, $a_k \in \{-1, 0, 1\}$ such that

$$n = (a_{m-1} a_{m-2} \cdots a_1 a_0)_3 = \sum_{k=0}^{m-1} a_k 3^k.$$

In the following, we denote -1 as $\hat{1}$. The sum of two 1-bit integers has the form: $(\alpha)_3 + (\beta)_3 = (cr)_3$ where $\alpha, \beta, c, r \in \{\hat{1}, 0, 1\}$, c and r are the carry number and remaining number, respectively. Thus, $(1)_3 + (0)_3 = (0)_3 + (1)_3 = (01)_3; (\hat{1})_3 + (0)_3 = (0)_3 + (\hat{1})_3 = (0\hat{1})_3; (1)_3 + (\hat{1})_3 = (\hat{1})_3 + (1)_3 = (00)_3; (1)_3 + (1)_3 = (1\hat{1})_3$ and $(\hat{1})_3 + (\hat{1})_3 = (\hat{1}1)_3$.

We consider the sum of two integers x and y where $x = (V_{1,m-1} V_{1,m-2} \cdots V_{1,1} V_{1,0})_3$ is the original summand and $y = (V_{2,m-1} V_{2,m-2} \cdots V_{2,1} \ V_{2,0})_3$ is the original addend. For simplicity of description, let $V_{1,m-1} = V_{2,m-1} = 0$. We first make bitwise addition. Let $(V_{1,k})_3 + (V_{2,k})_3 = (c_k r_k)_3$ for $k = 0, 1, \ldots, m-1$. Then $c_{m-1} = r_{m-1} = 0$. Thus we get two new integers $x_1 = (r_{m-1} \, r_{m-2} \cdots r_1 r_0)_3$ consisting of the remaining numbers, and $y_1 = (c_{m-2} \ c_{m-3} \cdots c_1 c_0 0)_3$ consisting of the carry numbers. Then we have $x + y = x_1 + y_1$ with x_1 being the new summand and y_1 being the new addend. Inductively, for any $k \geqslant 2$ we can get integers x_k, y_k from x_{k-1}, y_{k-1} in the same way as that we get x_1, y_1 from x, y. Then $x_k + y_k = x_{k-1} + y_{k-1} = \cdots = x + y$. Otherwise, suppose that $x_k + y_k \neq x_{k-1} + y_{k-1} = \cdots = x + y$. Then it must be either $x_{k-1} = (1 * \cdots *)_3, y_{k-1} = (1 * \cdots *)_3$ or $x_{k-1} = (\hat{1} * \cdots *)_3, y_{k-1} = (\hat{1} * \cdots *)_3$. For the former case we have $x_{k-1} = (1 * \cdots *)_3 \geqslant (1\hat{1} \cdots \hat{1})_3 > (01 \cdots 1)_3 \geqslant x$ and $y_{k-1} = (1 * \cdots *)_3 \geqslant (1\hat{1} \cdots \hat{1})_3 > (01 \cdots 1)_3 \geqslant y$, leading to $x_{k-1} + y_{k-1} > x + y$. The same argument gives that $x_{k-1} + y_{k-1} < x + y$ for the latter case. Note that the last k bits in the ternary representation of y_k are all 0 by the definition of y_k. Then there exists a $k \leqslant m$ such that $y_k = (0, 0, \ldots, 0)_3$ and so $x + y = x_k$. Let

$$\Sigma = \{A_1, A_2, B_0, B_1, \ldots, B_{m-1}, C_0, C_1, D_0, D_1, E_0, E_1, \hat{1}, 0, 1, \#\},$$

be a set of single strands. Single strands "$\hat{1}$", "0" and "1" are used to denote values of bits. A_1 and A_2 denote addresses of ternary numbers, and $B_0, B_1, \ldots, B_{m-1}$ denote bit positions in a ternary number. # is a special symbol for *Separation*. C_0, C_1, D_0, D_1 and E_0, E_1 are the specified symbols cut by *Cleavage*, that is, *Cleavage*(T, C_0C_1), *Cleavage*(T, D_0D_1) and *Cleavage*(T, E_0E_1) cut all double strands containing

$$\left[\frac{C_0C_1}{\overline{C_0C_1}}\right], \left[\frac{D_0D_1}{\overline{D_0D_1}}\right],$$

and

$$\left[\frac{E_0E_1}{\overline{E_0E_1}}\right],$$

in a test tube T, respectively.

To design procedures for logic and arithmetic operations with DNA molecules, 3 used the alphabet

$$\{A_1, \ldots, A_{n-1}, B_0, B_1, \ldots, B_{m-1}, C_0, C_1, D_0, D_1, 0, 1, \#\},$$

and introduced the single strands

$$D_1 A_i B_j C_0 C_1 V D_0, \quad i = 1, 2, \ldots, n, \quad j = 0, 1, \ldots, m-1, \tag{1}$$

to denote n binary numbers of m bits, where $V =$ "0" if a value of the bit is 0, otherwise $V =$ "1". With this bit representation of a binary number, they proposed some basic procedures for logic and arithmetic operations. One of those operations is called *ValueAssignment*, which is used to assign values to every bits in a test tube. More exactly, $ValueAssignment_V(T_{input}, T_{output})$ is an operation which assigns the same value $V(\in\{0,1\})$ to all memory strands in a test tube T_{input} and store the result in a test tube T_{output}.

Let $x = (V_{1,m-1}V_{1,m-2}\cdots V_{1,1}V_{1,0})_3$ and $y = (V_{2,m-1}V_{2,m-2}\cdots V_{2,1}V_{2,0})_3$ with $V_{1,m-1} = V_{2,m-1} = 0$. In the following, we use the memory strands

$$S_{k,i}(V_{k,i}) = D_1A_kE_0E_1B_iC_0C_1V_{k,i}D_0, \quad k = 1,2,\ i = 0,1,\ldots,m-1, \tag{2}$$

to denote the value of the i-bit of the kth integer, where $V_{k,i} \in \{\hat{1}, 0, 1\}$. Thus, for example, the tube

$$\{D_1A_1E_0E_1B_4C_0C_1 0D_0, D_1A_1E_0E_1B_3C_0C_1\hat{1}D_0, D_1A_1E_0E_1B_2C_0C_1 1D_0,$$
$$D_1A_1E_0E_1B_1C_0C_1 1D_0, D_1A_1E_0E_1B_0C_0C_1 0D_0\},$$

denotes the integer $(0\hat{1}110)_3 = -15$. Note that the DNA bit representation by (2) is made based on that by (1), but inserted a DNA strand E_0E_1 between A_k and B_i for our purpose so that we can make shift operation on the position code B_i. However, the operation *ValueAssignment* is still available for our setting. For readers' convenience, we give a description of the operation *ValueAssignment* as follows.

Let

$$T_{input} = \{D_1A_kE_0E_1B_iC_0C_1V_{k,i}D_0,\ 0 \leqslant i \leqslant m-1\},$$

where $V_{k,i} \in \{\hat{1}, 0, 1\}$. Then the operation $ValueAssignment_V(T_{input}, T_{output})$ will produce both input and output test tubes as

$$T_{input} = T_{output} = \{D_1A_kE_0E_1B_iC_0C_1VD_0,\ 0 \leqslant i \leqslant m-1\},$$

where $V \in \{\hat{1}, 0, 1\}$. Note that all memory strands are set to the same value V. Let $T_{\bar{C}}$ and T_{valueV} be two auxiliary test tubes such that $T_{\bar{C}} = \{\overline{C_0C_1}\}, T_{valueV} = \{C_1VD_0, \overline{C_0C_1}\}$.

Procedure $ValueAssignment_V(T_{input}, T_{output})$

Step 1. Delete values from memory strands.

$Copy(T_{\overline{C}}, T_{\overline{C}}*)$

$Merge\ (T_{input}, T_{\overline{C}}*)$

$$\Rightarrow T_{input} = \{\overline{C_0C_1}, D_1A_kE_0E_1B_iC_0C_1V_{k,i}D_0,\ 0 \leqslant i \leqslant m-1\},$$

$Annealing(T_{input})$

$$\Rightarrow T_{input} = \left\{ \left[\begin{matrix} D_1A_kE_0E_1B_iC_0C_1V_{k,i}D_0 \\ \overline{C_0C_1} \end{matrix} \right] \middle|\ 0 \leqslant i \leqslant m-1 \right\}.$$

$Cleavage(T_{input}, C_0C_1)$

$$\Rightarrow T_{input} = \left\{ \left[\begin{matrix} D_1A_kE_0E_1B_iC_0 \\ \overline{C_0} \end{matrix} \right], \left[\begin{matrix} C_1V_{k,i}D_0 \\ \overline{C_1} \end{matrix} \right] \middle|\ 0 \leqslant i \leqslant m-1 \right\}.$$

$Denaturation(T_{input})$

$$\Rightarrow T_{input} = \{\overline{C_0}, \overline{C_1}, D_1A_kE_0E_1B_iC_0, C_1V_{k,i}D_0|\ 0 \leqslant i \leqslant m-1\}.$$

$Separation(T_{input}, \{C_1, \bar{C}_0, \bar{C}_1\}, T_{tmp})$

$$\Rightarrow T_{input} = \{D_1A_kD_0D_1B_iC_0|\ 0 \leqslant i \leqslant m-1\}.$$

Step 2. Assign values to memory strands.

$Merge(T_{input}, T_{value\ V})$

$$\Rightarrow T_{input} = \{C_1VD_0, \overline{C_0C_1}, D_1A_kE0E_1B_iC_0|\ 0 \leqslant i \leqslant m-1\}.$$

$Annealing(T_{input})$

$$\Rightarrow T_{input} = \left\{ \begin{bmatrix} D_1A_kE_0E_1B_iC_0C_1VD_0 \\ \overline{C_0C_1} \end{bmatrix} \; 0 \leqslant i \leqslant m-1 \right\}.$$

$Denaturation(T_{input})$

$$\Rightarrow T_{input} = \{\overline{C_0C_1}, D_1A_kE_0E_1B_iC_0C_1VD_0 |\; 0 \leqslant i \leqslant m-1\}.$$

$Separation(T_{input}, \{\overline{C_0C_1}\}, T_{\bar{C}})$

$$\Rightarrow T_{input} = \{D_1A_kE_0E_1B_iC_0C_1VD_0 |\; 0 \leqslant i \leqslant m-1\}.$$

$Copy(T_{input}, T_{output})$

$$\Rightarrow T_{output} = \{D_1A_kE_0E_1B_iC_0C_1VD_0 |\; 0 \leqslant i \leqslant m-1\}.$$

The operation *ValueAssignment* will be used in our following procedure. We now use the test tubes T_{input1} and T_{input2} to store the summand x and the addend y, respectively, i.e.,

$$T_{input1} = \{S_{1,i}(V_{1,i}), i = 0, 1, \ldots, m-1\} = \{D_1A_1E_0E_1B_iC_0C_1V_{1,i}D_0,\ i = 0, 1, \ldots, m-1\},$$

and

$$T_{input2} = \{S_{2,i}(V_{2,i}), i = 0, 1, \ldots, m-1\} = \{D_1A_2E_0E_1B_iC_0C_1V_{2,i}D_0,\ i = 0, 1, \ldots, m-1\}.$$

Let $L = \bigcup_{i=0}^{m-1} L_i$ where

$$L_i = \Big\{\overline{0\#D_0S_{1,i}(0)S_{2,i}(0)D_10\#}, \overline{0\#D_0S_{1,i}(0)S_{2,i}(1)D_11\#}, \overline{0\#D_0S_{1,i}(0)S_{2,i}(\hat{1})D_1\hat{1}\#},$$
$$\overline{0\#D_0S_{1,i}(1)S_{2,i}(0)D_11\#}, \overline{1\#D_0S_{1,i}(1)S_{2,i}(1)D_1\hat{1}\#}, \overline{0\#D_0S_{1,i}(1)S_{2,i}(\hat{1})D_10\#}, \overline{0\#D_0S_{1,i}(\hat{1})S_{2,i}(0)D_1\hat{1}\#},$$
$$\overline{0\#D_0S_{1,i}(\hat{1})S_{2,i}(1)D_10\#}, \overline{\hat{1}\#D_0S_{1,i}(\hat{1})S_{2,i}(\hat{1})D_11\#}\Big\}.$$

The tube L_i corresponds to the bitwise addition of the $i+1$th bits $S_{1,i}(\alpha)$ and $S_{2,i}(\beta)$ of the summand and the addend with $\alpha, \beta \in \{\hat{1}, 0, 1\}$. This kind of tubes were introduced by Fujiwara et al. [3] for logic and arithmetic operations with DNA molecules. In fact, each single strand is of form $\overline{c\#D_0S_{1,i}(\alpha)S_{2,i}(\beta)D_1r\#}$ which corresponds to the bitwise addition of $(\alpha)_3 + (\beta)_3 = (cr)_3$. Nine distinct single strands in tube L_i correspond to all distinct evaluations of α and β. We also need the following auxiliary test tubes.

$$F_0 = \Big\{\overline{0\#D_0S_{1,i}(0)}, \overline{0\#D_0S_{1,i}(1)}, \overline{0\#D_0S_{1,i}(\hat{1})}, \overline{S_{2,i}(0)D_10\#}, \overline{S_{2,i}(1)D_10\#}, \overline{S_{2,i}(\hat{1})D_10\#}, i = 0, 1, \ldots, m-1\Big\};$$

$$F_1 = \Big\{\overline{1\#D_0S_{1,i}(1)}, \overline{S_{2,i}(0)D_11\#}, \overline{S_{2,i}(1)D_11\#}, \overline{S_{2,i}(\hat{1})D_11\#}, i = 0, 1, \ldots, m-1\Big\};$$

$$F_{\hat{1}} = \Big\{\overline{\hat{1}\#D_0S_{1,i}(\hat{1})}, \overline{S_{2,i}(0)D_1\hat{1}\#}, \overline{S_{2,i}(1)D_1\hat{1}\#}, \overline{S_{2,i}(\hat{1})D_1\hat{1}\#}, i = 0, 1, \ldots, m-1\Big\};$$

$$Q = \Big\{0\#D_0, 1\#D_0, \hat{1}\#D_0, D_10\#, D_11\#, D_1\hat{1}\#\Big\};$$

$$T_{\bar{E}} = \Big\{\overline{E_0E_1}\Big\}; \quad T_{\bar{C}} = \Big\{\overline{C_0C_1}\Big\}; \quad T_{put0} = \{C_10D_0\};$$

$$T_{carry} = \Big\{D_1A_1E_0E_1B_{i+1}C_0C_1, \overline{D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0}|\; 0 \leqslant i \leqslant m-2\Big\}.$$

We present the procedure for addition of two ternary integers as follows. For readers' convenience we make some descriptions following the symbol $\Rightarrow$.

Step 1. We calculate the carry numbers c_i and the remaining numbers r_i in the bitwise addition, i.e. $(V_{1,i})_3 + (V_{2,i})_3 = (c_i r_i)_3,\ i = 0,1,\ldots,m-1$. In sub-steps 1.13–15, we get T_0,T_1 and $T_{\hat{1}}$ which store information about c_i and r_i, i.e., for $k = 0, 1, \hat{1}$, $c_i = k$ if and only if $k\#D_0S_{1,i}(V_{1,i}) \in T_k$, and $r_i = k$ if and only if $S_{2,i}(V_{2,i})D_1k\# \in T_k$.

1.1. *Merge*(T_{input1},T_{input2})—**the tube T_{input2} now is empty.**

$\Rightarrow T_{input1} = \{D_1A_kE_0E_1B_iC_0C_1V_{k,i}D_0, k = 1,2, i = 0,1,\ldots,m-1\}$.

1.2. *Copy*(L,$L*$)

$$\Rightarrow L* = \Big\{\overline{0\#D_0S_{1,i}(0)S_{2,i}(0)D_10\#}, \overline{0\#D_0S_{1,i}(0)S_{2,i}(1)D_11\#}, \overline{0\#D_0S_{1,i}(0)S_{2,i}(\hat{1})D_1\hat{1}\#},$$
$$\overline{0\#D_0S_{1,i}(1)S_{2,i}(0)D_11\#}, \overline{1\#D_0S_{1,i}(1)S_{2,i}(1)D_1\hat{1}\#}, \overline{0\#D_0S_{1,i}(1)S_{2,i}(\hat{1})D_10\#},$$
$$\overline{0\#D_0S_{1,i}(\hat{1})S_{2,i}(0)D_1\hat{1}\#}, \overline{0\#D_0S_{1,i}(\hat{1})S_{2,i}(1)D_10\#},$$
$$\overline{\hat{1}\#D_0S_{1,i}(\hat{1})S_{2,i}(\hat{1})D_11\#},\ i = 0,1,\ldots,m-1\Big\}.$$

1.3. *Merge*(T_{input1},$L*$)

1.4. *Annealing*(T_{input1})

$$\Rightarrow T_{input1} = \left\{\left[\frac{S_{1,i}(V_{1,i})S_{2,i}(V_{2,i})}{\alpha\#D_0S_{1,i}(V_{1,i})S_{2,i}(V_{2,i})D_1\beta\#}\right]\middle|\ (V_{1,i}+V_{2,i})_3 = (\alpha\beta)_3,\ 0 \leqslant i \leqslant m-1\right\}$$
$$\bigcup \{\text{some other elements of } L*\}.$$

1.5. *Cleavage*(T_{input1}, D_0D_1)

$$\Rightarrow T_{input1} = \left\{\left[\frac{S_{1,i}(V_{1,i})}{\alpha\#D_0S_{1,i}(V_{1,i})}\right], \left[\frac{S_{2,i}(V_{2,i})}{S_{2,i}(V_{2,i})D_1\beta\#}\right]\middle|\ (V_{1,i}+V_{2,i})_3 = (\alpha\beta)_3,\ 0 \leqslant i \leqslant m-1\right\}$$
$$\bigcup \{\text{some other elements of } L*\}.$$

1.6. *Copy*(Q,$Q*$)

1.7. *Merge*(T_{input1}, $Q*$)

1.8. *Annealing*(T_{input1})

$$\Rightarrow T_{input1} = \left\{\left[\frac{\alpha\#D_0S_{1,i}(V_{1,i})}{\alpha\#D_0S_{1,i}(V_{1,i})}\right], \left[\frac{S_{2,i}(V_{2,i})D_1\beta\#}{S_{2,i}(V_{2,i})D_1\beta\#}\right]\middle|\ (V_{1,i}+V_{2,i})_3 = (\alpha\beta)_3,\ 0 \leqslant i \leqslant m-1\right\}$$
$$\bigcup \{\text{some other elements of } L* \text{ and } Q*\}.$$

1.9. *Denaturation*(T_{input1})

1.10. *Separation*(T_{input1},$\{C_0C_1\}$,T_{tmp})

$\Rightarrow T_{tmp} = \{\alpha\#D_0S_{1,i}(V_{1,i}), S_{2,i}(V_{2,i})D_1\beta\#|(V_{1,i}+V_{2,i})_3 = (\alpha\beta)_3,\ 0 \leqslant i \leqslant m-1\}$.

1.11. *Discard*(T_{input1})

1.12. *Merge*(T_{input1},T_{tmp})

$\Rightarrow T_{input1} = \{\alpha\#D_0S_{1,i}(V_{1,i}), S_{2,i}(V_{2,i})D_1\beta\#|(V_{1,i}+V_{2,i})_3 = (\alpha\beta)_3,\ 0 \leqslant i \leqslant m-1\}$.

1.13. *Separation*(T_{input1},$\{0\#D_0$,$D_10\#\}$,T_0)

$\Rightarrow T_0 = \{0\#D_0S_{1,i}(V_{1,i}), S_{2,j}(V_{2,j})D_10\#|$ for some $0 \leqslant i,j \leqslant m-1\}$.

1.14. *Separation*(T_{input1},$\{1\#D_0$,$D_11\#\}$,T_1)

$\Rightarrow T_1 = \{1\#D_0S_{1,i}(V_{1,i}), S_{2,j}(V_{2,j})D_11\#|$ for some $0 \leqslant i,j \leqslant m-1\}$.

1.15. *Separation*(T_{input1}, $\{\hat{1}\#D_0, D_1\hat{1}\#\}$, $T_{\hat{1}}$)—**the tube T_{input1} now is empty.**

$\Rightarrow T_{\hat{1}} = \{\hat{1}\#D_0S_{1,i}(V_{1,i}), S_{2,j}(V_{2,j})D_1\hat{1}\#|$ for some $0 \leqslant i,j \leqslant m-1\}$.

Step 2. As we have seen from step 1, for $k \in \{\hat{1}, 0, 1\}$, both all carry numbers with value k and all remaining numbers with value k are contained in tube T_k. We first assign value "0" to memory strands in T_0, then separate all remaining numbers from the tube. Repeat this procedure for tubes T_1 and $T_{\hat{1}}$ to assign value "*1*" or "$\hat{1}$" to memory strands in T_1 or $T_{\hat{1}}$, and separate all remaining numbers from tubes T_1 and $T_{\hat{1}}$.

2.1. *Copy*(F_0,F_0*)

$$\Rightarrow F_0* = \{\overline{0\#D_0S_{1,i}(0)}, \overline{0\#D_0S_{1,i}(1)}, \overline{0\#D_0S_{1,i}(\hat{1})}, \overline{S_{2,i}(0)D_10\#}, \overline{S_{2,i}(1)D_10\#}, \overline{S_{2,i}(\hat{1})D_10\#}, \quad i = 0, 1, \ldots, m-1\}.$$

2.2. *Merge*(T_0,F_0*)

2.3. *Annealing*(T_0)

$$\Rightarrow T_0 = \left\{ \left[\frac{0\#D_0S_{1,i}(V_{1,i})}{0\#D_0S_{1,i}(V_{1,i})} \right], \left[\frac{S_{2,j}(V_{2,j})D_10\#}{S_{2,j}(V_{2,j})D_10\#} \right] \middle| \text{ for some } 0 \leqslant i, j \leqslant m-1 \right\} \bigcup \{\text{some other elements of } F_0*\}.$$

2.4. *Cleavage*(T_0,D_0D_1)

$$\Rightarrow T_0 = \left\{ \left[\frac{S_{1,i}(V_{1,i})}{S_{1,i}(V_{1,i})} \right], \left[\frac{S_{2,j}(V_{2,j})}{S_{2,j}(V_{2,j})} \right] \middle| \text{ for some } 0 \leqslant i, j \leqslant m-1 \right\} \bigcup \left\{ \left[\frac{0\#D_0}{0\#D_0} \right], \left[\frac{D_10\#}{D_10\#} \right] \right\} \bigcup \{\text{some other elements of } F_0*\}.$$

2.5. *Denaturation*(T_0)

2.6. *Separation*(T_0, $\{\#, \bar{\#}, \overline{C_0C_1}\}$, T_{tmp})

$$\Rightarrow T_0 = \{S_{1,i}(V_{1,i}), S_{2,j}(V_{2,j}) \text{ for some } 0 \leqslant i, j \leqslant m-1\}.$$

2.7. *ValueAssignment*$_0$(T_0,T_{zero})

$$\Rightarrow T_0 = \{S_{1,i}(0), S_{2,j}(0) \text{ for some } 0 \leqslant i, j \leqslant m-1\}.$$

2.8. *Separation*(T_0,{A_2},T_{02})

$$\Rightarrow T_0 = \{S_{1,i}(0) \text{ for some } 0 \leqslant i \leqslant m-1\} T_{02} = \{S_{2,j}(0) \text{ for some } 0 \leqslant j \leqslant m-1\}.$$

2.9. *Separation*(T_0,{$S_{1,m-1}(0)$},T_{tmp})

$$\Rightarrow T_0 = \{S_{1,i}(0) \text{ for some } 0 \leqslant i \leqslant m-2\} := \{S_{1,i}(0) |\ i \in \Gamma_0 \subset \{0, 1, \ldots, m-2\}\}.$$

2.10. Repeat above substeps 2.1–8 for tubes T_1 and $T_{\hat{1}}$, respectively, obtaining tubes T_1, T_{12}, $T_{\hat{1}}$ and $T_{\hat{1}2}$.

Step 3. We give the representation of form (2) for the new summand $(r_{m-1}r_{m-2}\cdots r_1r_0)_3$ and the new addend $(c_{m-2},c_{m-3}\cdots c_00)_3$. It just needs to merge all tubes T_{02}, T_{12} and $T_{\hat{1}2}$ to obtain the representation of form (2) for the new summand $(r_{m-1}r_{m-2}\cdots r_1r_0)_3$. However, in order to obtain the representation of form (2) for the new addend $(c_{m-2},c_{m-3}\cdots c_00)_3$, for each memory strand in $T_0 \cup T_1 \cup T_{\hat{1}}$ one must change its position code B_i,$i = 0,1,\ldots,m-2$ into B_{i+1}. We first perform it for memory strands in T_0. The same treatment can be applied to memory strands in T_1 and $T_{\hat{1}}$.

3.1. *Copy*($T_{\bar{E}}$, $T_{\bar{E}}$*)

3.2. *Merge*(T_0, $T_{\bar{E}}$*)

$$\Rightarrow T_0 = \left\{ \overline{E_0E_1}, D_1A_1E_0E_1B_iC_0C_10D_0 \middle|\ i \in \Gamma_0 \right\}.$$

3.3. *Annealing*(T_0)

$$\Rightarrow T_0 = \left\{ \left[\begin{matrix} D_1A_1E_0E_1B_iC_0C_10D_0 \\ \overline{E_0E_1} \end{matrix} \right] \middle| \; i \in \Gamma_0 \right\}.$$

3.4. *Cleavage*(T_0,E_0E_1)

$$\Rightarrow T_0 = \left\{ \left[\begin{matrix} D_1A_1E_0 \\ \overline{E_0} \end{matrix} \right], \left[\begin{matrix} E_1B_iC_0C_10D_0 \\ \overline{E_1} \end{matrix} \right] \middle| \; i \in \Gamma_0 \right\}.$$

3.5. *Denaturation*(T_0)

3.6. *Separation*($T_1, \{E_0, \overline{E_0}, \overline{E_1}\}, T_{tmp}$) $\Rightarrow T_0 = \{E_1B_iC_0C_10D_0 |\; i \in \Gamma_0\}$

3.7. *Copy*($T_{\bar{C}}, T_{\bar{C}}*$)

3.8. *Merge*($T_0, T_{\bar{C}}*$)

$$\Rightarrow T_0 = \{\overline{C_0C_1}, E_1B_iC_0C_10D_0 | i \in \Gamma_0\}.$$

3.9. *Annealing*(T_0)

$$\Rightarrow T_0 = \left\{ \left[\begin{matrix} E_1B_iC_0C_10D_0 \\ \overline{C_0C_1} \end{matrix} \right] \middle| \; i \in \Gamma_0 \right\}.$$

3.10. *Cleavage*(T_0,C_0C_1)

$$\Rightarrow T_0 = \left\{ \left[\begin{matrix} E_1A_iC_0 \\ \overline{C_0} \end{matrix} \right], \left[\begin{matrix} C_10D_0 \\ \overline{C_1} \end{matrix} \right] \middle| \; i \in \Gamma_0 \right\}.$$

3.11. *Denaturation*(T_0) *Separation*($T_0, \{C_0, \overline{C_0}, \overline{C_1}\}, T_{tmp}$) $\Rightarrow T_0 = \{E_1B_iC_0 |\; i \in \Gamma_0\}$

3.12. *Copy*(T_{carry},T_{carry}*)

$$\Rightarrow T_{carry}* = \{D_1A_1E_0E_1B_{i+1}C_0C_1, \overline{D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0} |\; 0 \leqslant i \leqslant m-2\}.$$

3.13. *Merge*(T_0,T_{carry}*)

$$\Rightarrow T_0 = \{E_1B_iC_0 | i \in \Gamma_0\} \cup \{D_1A_1E_0E_1B_{i+1}C_0C_1, \overline{D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0} |\; 0 \leqslant i \leqslant m-2\}.$$

3.14. *Annealing*(T_0)

$$\Rightarrow T_0 = \left\{ \left[\begin{matrix} D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0 \\ \overline{D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0} \end{matrix} \right] \middle| \; i \in \Gamma_0 \right\} \bigcup \left\{ \left[\begin{matrix} D_1A_1E_0E_1B_{i+1}C_0C_1 \\ \overline{D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0} \end{matrix} \right] \middle| \; i \in \Gamma_0^c \right\}.$$

3.15. *Denaturation*(T_0)

3.16. *Separation*(T_0,$\{C_0C_1E_1\}$,T_{00})

$$\Rightarrow T_{00} = \{D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0 |\; i \in \Gamma_0\}.$$

3.17. *Discard*(T_0)

3.18. *Merge*(T_0,T_{00})

$$\Rightarrow T_0 = \{D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0 |\; i \in \Gamma_0\}.$$

3.19. *Copy*($T_{\bar{C}}, T_{\bar{C}}*$)

3.20. *Merge*($T_0, T_{\bar{C}}*$)

$$\Rightarrow T_0 = \{\overline{C_0C_1}, D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0 |\; i \in \Gamma_0\}.$$

3.21. *Annealing*(T_0)

$$\Rightarrow T_0 = \left\{ \left[\begin{matrix} D_1A_1E_0E_1B_{i+1}C_0C_1E_1B_iC_0 \\ \overline{C_0C_1} \end{matrix} \right] \middle| \; i \in \Gamma_0 \right\}.$$

3.22. *Cleavage*(T_0, C_0C_1)

$$\Rightarrow T_0 = \left\{ \begin{bmatrix} D_1A_1E_0E_1B_{i+1}C_0 \\ \overline{C_0} \end{bmatrix}, \begin{bmatrix} C_1E_1B_iC_0 \\ \overline{C_1} \end{bmatrix} \middle| \; i \in \Gamma_0 \right\}.$$

3.23. *Denaturation*(T_0)
3.24. *Separation*$(T_0, C_1E_1, \overline{C_0}, \overline{C_1})$

$$\Rightarrow T_0 = \{D_1A_1E_0E_1B_{i+1}C_0 |\; i \in \Gamma_0\}.$$

3.25. *Copy*$(T_{\bar{C}}, T_{\bar{C}}*)$
3.26. *Copy*$(T_{put0}, T_{put0}*)$
3.27. *Merge*$(T_0, T_{\bar{C}}*)$
3.28. *Merge*$(T_0, T_{put0}*)$

$$\Rightarrow T_0 = \{\overline{C_0C_1}\}, C_10D_0, D_1A_1E_0E_1B_{i+1}C_0 | i \in \Gamma_0.$$

3.29. *Annealing*(T_0)

$$\Rightarrow T_0 = \left\{ \begin{bmatrix} D_1A_1E_0E_1B_{i+1}C_0C_10D_0 \\ \overline{C_0C_1} \end{bmatrix} \middle| \; i \in \Gamma_0 \right\}.$$

3.30. *Denaturation*(T_0)
3.31. *Separation*$(T_0, \{\overline{C_0C_1}\}, T_{tmp})$

$$\Rightarrow T_0 = \{D_1A_1E_0E_1B_{i+1}C_0C_10D_0 | i \in \Gamma_0\}.$$

3.32. Repeat sub-steps 3.1–31 to change position codes for memory strands in tubes T_1 and $T_{\hat{1}}$.
3.33. *Merge*(T_0, T_1)
3.34. *Merge*$(T_0, T_{\hat{1}})$
3.35. *Merge*$(T_0, \{D_1A_1E_0E_1B_0C_0C_10D_0\})$
3.36. *Merge*(T_{input2}, T_0)—**the new addend number stored in tube** T_{input2}
3.37. *Merge*(T_{02}, T_{12})
3.38. *Merge*$(T_{02}, T_{\hat{1}2})$
3.39. *Merge*(T_{input1}, T_{02})—**the new summand number stored in tube** T_{input1}

Step 4. We check whether or not the added number is zero.

4.1. *Separation*$(T_{input2}, \{C_10D_0\}, T_{tmp2})$

$$\Rightarrow T_{input2} = \{D_1A_1E_0E_1B_iC_0C_1V_{1,i}D_0 | V_{1,i} \neq 0, i = 0, 1, \ldots, m-1\}.$$

4.2. If *Detect*(T_{input2}) is "no", then
4.2.1. *Read*(T_{input1}) to obtain the summation and end the operations, else
4.2.2. *Merge*(T_{input2}, T_{tmp2}) and repeat the above steps 1–4.

As we have shown in the beginning of this section, the renewed added number will be 0 after performing bitwise addition $k(\leqslant m)$ times. The proposed algorithm works in O(m) steps for adding two ternary integers of m bits.

4. Conclusions

In this paper, we consider an algorithm for adding two integers with DNA molecules. The integer is represented as a ternary expansion with base digits -1, 0 and 1 so that both positive and negative integers can be represented in a unified form. Thus our algorithm does not distinguish addition and subtraction. There have been lots of papers related to arithmetic and logic operations with DNA molecules (Ref. [2,3,4,5,6,7,12]). Among these papers several algorithms have been proposed for adding two positive binary integers in O(1) steps. However, all proposed operations there for carry propagation inevitably produce DNA strands with

length depending on the number of bits of the summand and addend. This potentially increase the possibility of occurrence of hairpin loop [8]. Although algorithm proposed in this paper costs O(m) steps for adding two ternary integers of m bits, it only produces DNA strands of length less than a constant so that it effectively avoid occurrence of hairpin loop. Moreover, the DNA representation of a ternary integer features its address and bit position so that operations for value assigning and bit position shifting can be implemented based on this DNA representation. All our results in this paper are based on a theoretical model. However, the proposed procedures can be implemented practically since every DNA manipulation used in this model has been already realized in lab level.

Acknowledgement

This work is supported by Bio-X DNA Computer Consortium No. 03DZ14025. The first author is also supported by National Science Foundation of China #10571058 and Shanghai Priority Academic Discipline.

References

[1] L.M. Adleman, Molecular computation of solution to combinatorial problems, Science 266 (1994) 1021–1024.
[2] P. Frisco, Parallel arithmetic with splicing, Romanian Journal of Information Science and Technology 2 (2002) 113–128.
[3] A. Fujiwara, K. Matsumoto, Wei Chen, Procedures for logic and arithmetic operations with DNA molecules, International Journal of Foundations Computer Science 15 (2004) 461–474.
[4] F. Guarnieri, M. Fliss, C. Bancroft, Making DNA add, Science 273 (1996) 220–223.
[5] V. Gupta, S. Parthasarathy, M.J. Zaki, Arithmetic and logic operations with DNA, in: Proceedings of third DIMACS Workshop on DNA Based Computers, 1997, pp. 212–220.
[6] H. Hug, R. Schuler, DNA-based parallel computation of simple arithmetic, in: Proceedings of the seventh Internatioanl Meeting on DNA Based Computers, 2001, pp. 159–166.
[7] S. Kamio, A. Takehara, A. Fujiwara, Procedures for computing the maximum with DNA strands, in: R.Humid Arabnia, Youngsong Mun (Eds.), Proceedings of the International Conference on DNA Based Computers, 2003.
[8] D. Li, H. Huang, X. Li, X. Li, Hairpin formation in DNA computation presents limits for large NP-complete problems, BioSystem 72 (2003) 203–207.
[9] R.J. Lipton, DNA solution of HARD computational problems, Science 268 (1995) 542–545.
[10] Q. Ouyang, Peter D. Kaplan, S. Liu, A. Libchaber, DNA solution of the maximal clique problem, Science 278 (1997) 446–449.
[11] G. Păun, G. Rozeberg, A. Salomaa, DNA Computing, Springer-Verlag, 1998.
[12] Z.F. Qiu, M. Lu, Arithmetic and logic operations for DNA computers, in: Proceedings of the Second IASTED Internatioanl conference on Parallel and Distributed Computing and Networks, 1998, pp. 481–486.