Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Discrete Mathematics 19 (2005) 125–131
Electronic Notes in DISCRETE MATHEMATICS
www.elsevier.com/locate/endm

A Gray code for binary subtraction

Joe Sawada [1]

Computing and Information Science, University of Guelph, Canada

Abstract

We present a 3-close Gray code for the Binary Subtraction Problem that can be implemented to run in constant amortized time.

Keywords: binary subtraction, Gray code, CAT algorithm

1 Binary subtraction problem

Binary Subtraction Problem (BSP): To represent an integer N as the difference of two binary numbers where the total number of non-zero bits is minimized.

A solution to the BSP can be represented using a single string over the alphabet $\{\bar{1},0,1\}$ - a notation introduced by Güntzer and Paul [2]. For example, 100001 - 001010 can be represented by $10\bar{1}10\bar{1}1$. A solution to the BSP for a given integer N is not necessarily unique. For instance, the solutions to BSP(51) are: 0110011, $0110\bar{1}0\bar{1}$, $100\bar{1}\bar{1}0\bar{1}$, $10\bar{1}0011$, $10\bar{1}0\bar{1}0\bar{1}$.

Ganesan and Manku [4] apply the binary subtraction problem to determine the optimal routes in Chord [3], a peer-to-peer network topology. The Chord topology is based on an undirected graph with 2^b nodes arranged on a ring, with edges connecting pairs of nodes that are 2^k positions apart for any $k \geq 0$.

[1] Email: sawada@cis.uoguelph.ca

1571-0653/2005 Published by Elsevier B.V.
doi:10.1016/j.endm.2005.05.018

Nodes in this topology represent computers and the edges represent the overlay network connections. In general, it is desirable to minimize the path length of routed messages between two computers. For this topology, it turns out that a shortest route between an arbitrary pair of nodes at distance N can be determined from either a solution to BSP(N) or BSP($2^b - N$) [4]. Given that some routes may be blocked, it is desirable to know all possible shortest paths.

A *Gray code* is a listing of all solutions to a given problem such that successive solutions in the listing differ by a constant amount. In this paper we describe a 3-close Gray code for listing all solutions to BSP(N) that can be implemented to run in constant amortized time. In addition we identify precisely the values for N that produce the maximal number of solutions given that the binary representation of N is composed of b bits.

2 Generation algorithms

Ganesan and Manku [04] give a non-deterministic algorithm for producing an exhaustive listing of BSP(N), but no analysis of the running time is given. The following is a recursive description of their algorithm where $\mathbf{B}(N)$ denotes a listing of all the solutions to BSP(N). The notation $\mathbf{B}(N) \cdot 1$ denotes the listing $\mathbf{B}(N)$ with an additional 1 appended to each string. The notation $\mathbf{B}(N), \mathbf{B}(M)$ indicates the list of strings $\mathbf{B}(N)$ followed by the list $\mathbf{B}(M)$.

$$\mathbf{B}(N) = \begin{cases} 0 & \text{if } N = 0 \\ \mathbf{B}(\frac{N}{2}) \cdot 0 & \text{if } \mathbf{suffix}(N, 0) \text{ and } N > 0 \\ \mathbf{B}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 0(01)^*01) \\ \mathbf{B}(\frac{N+1}{2}) \cdot \bar{1} & \text{if } \mathbf{suffix}(N, 1(10)^*11) \\ \mathbf{B}(\frac{N+1}{2}) \cdot \bar{1},\ \mathbf{B}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 11(01)^*01) \text{ or} \\ & \text{if } \mathbf{suffix}(N, 00(10)^*11) \end{cases}$$

The function **suffix**(N, *expr*) returns true if a suffix of N, represented in binary, matches the regular expression *expr*. An example computation tree for this algorithm is given in Figure 1 where $N = 51 = 110011$. The nodes in the tree represent the input strings, and the labels on the edges represent the character to be prepended to the output string as specified by $\mathbf{B}(N)$.

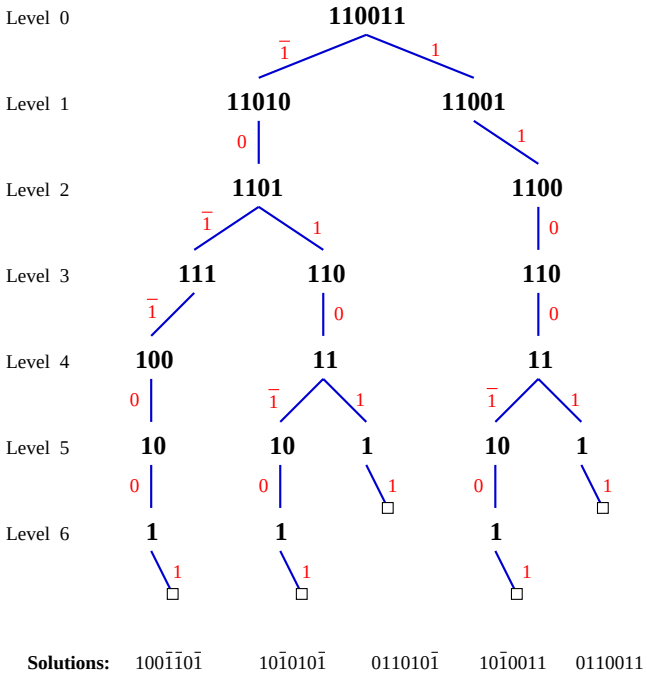

Fig. 1. Computation tree for $\mathbf{B}(51)$

2.1 Efficiency considerations

The running time of an algorithm for listing $\mathbf{B}(N)$ depends on two things: (1) the amount of computation required at each node in the computation tree and (2) the number of nodes in the computation tree. At each node in the computation, the input string must be scanned to determine which regular expression matches a suffix. In the worst case this will take time linear to the length of the input string. Additionally, the input string must be modified before making a recursive call. This will also take linear time in the worst case. This amount of computation is not desirable; however, because there are only $O(b)$ different nodes in any computation tree we can pre-compute the parent-child relationships in linear time to improve the algorithm so that only a constant amount of work is required at each node. Now, given that each node is the result of a constant amount of work, the overall running time of the algorithm will be proportional to the number of nodes in the computation tree. If this number is proportional to the number of solutions generated (the leaves), then the algorithm will run in constant amortized time. However, in general this will not be the case due to the large number of degree one nodes.

2.2 Gray code

One may observe from the computation tree in Figure 1 that in general the listing $\mathbf{B}(N)$ is not a Gray code since successive strings in the listing may differ by up to a linear amount $O(b)$. However by studying the listings for a variety of input values, we discover a 3-close Gray code description for $N > 0$. This new listing is obtained by reversing the order of particular subtrees within the computation tree of $\mathbf{B}(N)$. The result is a listing that will produce the same solutions, but in a different order. The overline in the description of this new listing $\mathbf{L}(N)$ indicates that the listing of strings are reversed.

$$\mathbf{L}(N) = \begin{cases} 0 & \text{if } N = 0 \\ \mathbf{L}(\frac{N}{2}) \cdot 0 & \text{if } \mathbf{suffix}(N, 0) \text{ and } N > 0 \\ \mathbf{L}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 0(01)^*01) \\ \mathbf{L}(\frac{N+1}{2}) \cdot \bar{1} & \text{if } \mathbf{suffix}(N, 1(10)^t11) \\ \overline{\mathbf{L}(\frac{N+1}{2})} \cdot \bar{1},\ \mathbf{L}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 11(01)^t01) \text{ and } t \text{ even} \\ \mathbf{L}(\frac{N+1}{2}) \cdot \bar{1},\ \overline{\mathbf{L}(\frac{N-1}{2})} \cdot 1 & \text{if } \mathbf{suffix}(N, 00(10)^t11) \text{ and } t \text{ even} \\ \mathbf{L}(\frac{N+1}{2}) \cdot \bar{1},\ \mathbf{L}(\frac{N-1}{2}) \cdot 1 & \text{if } \mathbf{suffix}(N, 11(01)^t01) \text{ and } t \text{ odd } \textbf{or} \\ & \text{if } \mathbf{suffix}(N, 00(10)^t11) \text{ and } t \text{ odd} \end{cases}$$

Theorem 2.1 *The listing* $\mathbf{L}(N)$ *of all solutions to BSP(N) where* $N > 0$ *is a 3-close Gray code.*

Proof Outline: Apply induction and show that the interface strings for the last 3 cases differ in exactly the last three positions. □

As an example, Figure 2 displays the computation tree for $\mathbf{L}(51)$. As before the nodes represent the input strings, but now if a subtree is to be reversed, this information is additionally passed down via the edges and represented by R. Naturally, a reversal of a reversed subtree produces the regular ordering (see the 11 node furthest to the right in Figure 2). The following is the final listing generated:

$$\begin{array}{cccccccccc} 0 & 1 & 0 & 0 & \bar{1} & \bar{1} & 0 & \bar{1} & & \\ 0 & 1 & 0 & \bar{1} & 0 & 1 & 0 & \bar{1} & & (2) \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & \bar{1} & & (4) \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & & (0) \\ 0 & 1 & 0 & \bar{1} & 0 & 0 & 1 & 1 & & (4) \end{array}$$

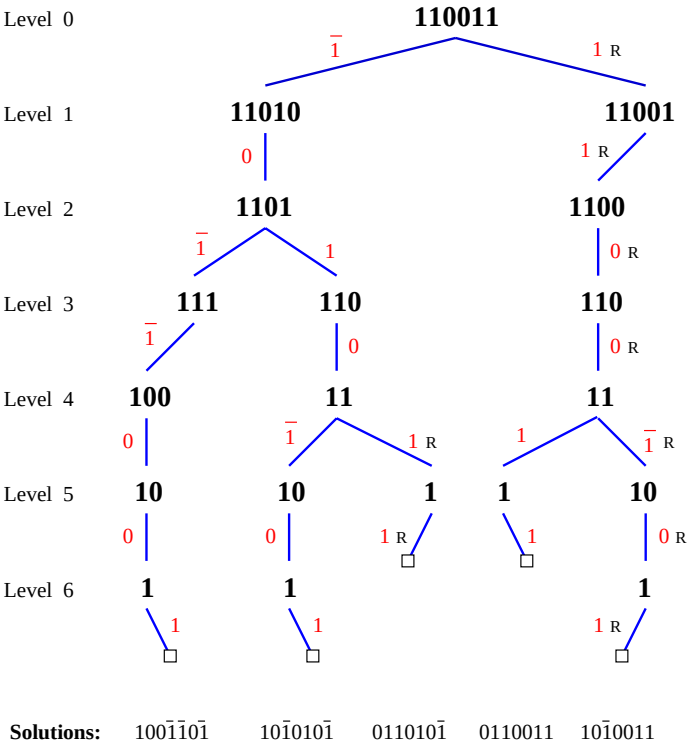

Fig. 2. Computation tree for $\mathbf{L}(51)$

Observe that each successive string differs in exactly 3 *successive* positions by either the transformation $011 \leftrightarrow 10\bar{1}$ or $0\bar{1}\bar{1} \leftrightarrow \bar{1}01$. Also observe that the rightmost of these positions (indicated in brackets) corresponds to the levels of the degree two nodes in the computation tree when visited in-order. Therefore, given the in-order sequence of levels of the degree 2 nodes along with the first output string in the listing, we can generate the Gray code listing $\mathbf{L}(N)$ in constant amortized time.

Theorem 2.2 *The listing $\boldsymbol{L}(N)$ can be generated in constant amortized time with a linear $O(b)$ amount of pre-processing.*

Proof Outline: The parent-child relationships for the compuatation tree of $\mathbf{L}(N)$ can be precomputed in $O(b)$ time. The nearest (left and right) degree 2 descendants of each degree 2 node can also be determined in $O(b)$ time. Thus, the degree 2 nodes can be be traversed in constant amortized time. □.

3 Strings with maximal solutions

If N is represented by the binary string $a_b \cdots a_0$, then the number of solutions to BSP(N) may be 1 or there could potentially be an exponential number of solutions with respect to b. Thus given b, we are interested in finding a tight upper bound on the number of solutions, denoted $Max(b)$, as well as a characterization of the bitstrings $a_b \cdots a_0$ that obtain this upper bound. When $b = 0, 1, 2$, the strings that produce the maximal number of solutions are 1, 11, and 110 respectively. The values $Max(0) = 1$ and $Max(1) = Max(2) = 2$. For $3 \leq b \leq 10$ we apply a generation algorithm to determine which binary representations of N have $Max(b)$ solutions to BSP(N):

Bits: b	**Binary representations of** N		$Max(b)$
3	1011	1101	3
4	10110	11010	3
5	101101	110011	5
6	1011010	1100110	5
7	10110011	11001101	8
8	101100110	110011010	8
9	1011001101	1100110011	13
10	10110011010	11001100110	13

It is not difficult to observe that patterns are occurring both in the strings that produce the maximal solutions and in the number of solutions $Max(b)$.

Theorem 3.1 *$Max(b) = f_{\lceil b/2 \rceil + 2}$, the $\lceil b/2 \rceil + 2^{nd}$ Fibonacci number. Moreover, the two bitstrings that have $Max(b)$ solutions to BSP(N) where $b \geq 3$ are:*

$$\begin{array}{llll} 10(1100)^t11 & \textit{and} & 11(0011)^t01 & \textit{if } b = 4t{+}3, \\ 10(1100)^t110 & \textit{and} & 11(0011)^t010 & \textit{if } b = 4t{+}4, \\ 10(1100)^t1101 & \textit{and} & 11(0011)^t0011 & \textit{if } b = 4t{+}5, \\ 10(1100)^t11010 & \textit{and} & 11(0011)^t00110 & \textit{if } b = 4t{+}6. \end{array}$$

Proof Outline: A generation algorithm will verify the base cases. Then using the recursive definition for $\mathbf{B}(N)$ we examine each possible suffix of N to determine restrictions on the solutions to BSP(N). We then apply induction on the suffixes that yield the most possible solutions. □

4 Interesting tidbit

If we consider the number of bits required to represent each solution to BSP(N) for each value of N starting from 0, we obtain the following sequence:

$$A_N = 0, 1, 1, 2, 1, 2, 2, 2, 1, 2, 2, 3, 2, 3, 2, 2, \ldots$$

For example, $A_3 = 2$ since 2 is the minimum number of bits required to represent 3 as the difference of two binary numbers. This sequence corresponds to sequence A007302 in Sloane's On-Line Encyclopedia of Integer Sequences [5]. Interestingly, these values also corresponds to the cost of grid communications on the Connection Machine [6]. This sequence is also discussed with respect to k-regular sequences in [1].

5 Final remarks

In this paper we have briefly described a 3-close Gray code algorithm to generate all solutions to BSP(N). Details were not specified, but the algorithm can be implemented to run in constant amortized time. An open problem is to determine whether or not the algorithm can be made loop-free. It is also interesting to see if these ideas can be extended to a k-ary alphabet.

References

[1] J.P. Allouche, J.Shallit, The ring of k-regular sequences, II, Theoretical Computer Science, Vol. 307 No. 1 (2003) 3-29.

[2] U. Güntzer and M. Paul, Jump interpolation search trees and symmetric binary numbers, Information Processing Letters, Vol. 26 No.4 (1987), 193-204.

[3] I. Stoica, R. Morris, D. Liben-Lowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan, Chord: A scalable peer-to-peer lookup protocol for internet applications, IEEE/ACM Transactions on Networking, Vol. 11 No. 1 (2003) 17-32.

[4] P. Ganesan and G.S. Manku, Optimal routing in Chord, Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), (2004) 169-178.

[5] N. Sloane, The on-line encyclopedia of integer sequences: ID A007302, www.research.att.com/~njas/sequences/index.html.

[6] A. Weitzman, Transformation of parallel programs guided by micro-analysis, Algorithms Seminar, B.Salvy editor, (1992-1993) 155-159. INRIA, Rapport de Recherche, No. 2130.