Effective Binary Perspectives in Algorithmic Problem Solving

DAVID GINAT

Tel Aviv University, Tel Aviv, Israel

The effectiveness of viewing, representing, and manipulating data via binary values is illustrated in the domain of algorithmic problem solving. A variety of illuminating binary aspects, regarded as *binary perspectives*, are displayed through four lively algorithmic challenges, some of which are two-player games. The illustrations demonstrate the benefits of invoking binary representation of numbers, binary complement, bit-by-bit processing, parity considerations, and the reduction of an integer task into a 0/1 task. The solutions to the challenges are presented gradually, encapsulating general algorithmic considerations and problem-solving methods, including invariant properties, stepwise refinement, processing by atomic components, auxiliary coloring, backward reasoning, and inductive generalization. Each of the illustrations is tied to a variety of referenced algorithmic schemes and applications, which employ the aspects demonstrated in this article. The variety of binary aspects, algorithmic considerations, and problem-solving methods make these illustrations appealing teaching resources for computer science educators, in particular in the domain of algorithms.

Categories and Subject Descriptors: K.3.2 [**Computers and Education**]: Computer and Information Science Education; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*Assertions*; *invariants*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Algorithmic problem solving, binary representation, education, games, invariant properties

1. INTRODUCTION

A primary aspect of computer science is the utilization of binary, 0/1 values. Bits are the atomic units of computer operation. Binary representation of numbers and symbols is the fundamental means for keeping and manipulating computer data. Binary-based encoding schemes, such as Gray codes, Hamming codes, and Huffman encoding, are valuable computational tools [Manber 1989]. In a broader sense, the notion of binary-choice methods and structures is most relevant in the domain of data-structures and algorithms, as can be seen, for example, in binary search, binary search trees, and bin sorting [Aho et al. 1983; Cormen et al. 1990].

Students learn and exercise binary representations and calculations throughout their studies. Binary elements and binary choices are invoked and utilized in diverse ways for miscellaneous tasks. The even/odd values, or any two-value choices that can be abstracted as 0 and 1. Although diverse, all the invocations share aspects of binary considerations. We

Authors' address: Tel Aviv University, Tel Aviv, Israel. Email: ginat@post.tau.ac.il

refer to aspects such as viewing, representing, and manipulating data via binary values as *binary perspectives*.

Binary perspectives are useful and effective. In particular, they yield very efficient solutions to fundamental algorithmic tasks. In many applications, invocation of a binary perspective is inherent, as its role and contribution are explicitly evident. However, there are applications in which the relevance of a binary perspective is hidden. In developing our students' problem-solving skills, it is beneficial to enhance their awareness of effective and illuminating binary perspectives. Such enhancement will expand the students' scientific point of view and broaden their repertoire of algorithmic problem-solving tools.

The objective of this article is to elaborate illuminating binary perspectives through illustrations that can serve as enriching teaching resources in an "Introduction to Algorithms" course. The illustrations contain novel and lively challenges. The solution to each challenge involves gradual progress, based on effective binary perspectives, using essential problem-solving elements such as stepwise refinement, decomposition, reduction, and auxiliary components. Effective binary perspectives and the problem-solving elements expand the problem solvers' repertoires in inventory, heuristics, and control in seeking efficient and concise solutions [Schoenfeld 1992].

The illustrations make use of mathematical games and tasks with physical objects. They are a primary instructional means for teachers and textbooks and a fundamental learning source for students [Chi and Bassok 1989]. In particular, illustrations that involve games raise student motivation [Bright et al. 1980], and illustrations that connect the natural environment with more formal systems are an asset in enriching student intuition [Nesher 1989].

The challenges in the illustrations can be solved in various ways (which are mentioned), but the focus is on the ways in which use of binary perspectives provides effective and enriched solutions to core problems. Binary perspectives yield binary patterns, which are assertions that illuminate core problem characteristics. Some of the assertions are specified as invariant properties that reflect underlying mathematical insight. Invariants are fundamental in algorithm design and analysis [Dijkstra 1976; Gries 1980].

Each illustration includes a series of observations that gradually develop insight into its algorithmic challenge. At the conclusion of each solution process, the binary perspective elements that yield the solution are linked to a variety of algorithmic schemes and applications, which are based on these elements. These links substantiate the relevance, utilization, and applicability of the binary perspectives, and tie them to diverse schemes in the algorithm domain.

The illustrations were presented to students in the "Introduction to Algorithms" course in order to enhance their intuition and develop their algorithmic problem-solving competence. The students indicated that the illustrations expanded their notion of binary considerations, widened their scientific perspective, and increased their problem-solving confidence.

The illustrations are presented in the following four sections. Section 2 introduces an integer retrieval scheme based on the concept of binary complement; Section 3 shows an effective utilization of bit-by-bit partitioning and processing in solving the less familiar problem of finding a majority in a large data set; Section 4 displays a game-playing strategy based on binary locations; Section 5 presents a game-playing strategy that capitalizes on 0/1 patterns of addition and multiplication; and finally Section 6 summarizes

the various elements displayed in the illustrations and refers to our experience in introducing them in class.

2. BINARY COMPLEMENT

The task in the following illustration is rather introductory. Solutions are inspired by the notion of complement, which is a fundamental term in computer science and mathematics. Its relevance in solving the task is shown at the end of the solution process for the design of various algorithmic schemes.

Integer Calls

An oracle and a computer program call integers in the range 0..*N*, where *N* is odd. The oracle makes the first call. The calls continue in alternating turns. No integer can be called more than once. The oracle keeps track of all the integers called. On its turn, it randomly calls an integer not yet called.

For the computer program, develop a memory-less calling scheme that will respond to each oracle call with an integer not yet called. (The calling process ends when all the integers have been called.)

Example. For *N*=5, the oracle may start by calling 3, the computer may respond with 1, the oracle – 4, the computer – 0, the oracle – 2, and the computer may make the final call, 5.

Note that the initial number of integers is even. Thus, if all the numbers are called, the last call is by the computer program. The challenge for the program is to avoid a repeated call without memorizing previous calls. That is, a response scheme of $O(1)$ space is required. At first glance, this may seem impossible, as the oracle calls may be completely random. Yet there should be some effective processing of information.

Observation 1. A response call to an oracle call should be some function of the oracle's call.

One may initially consider some digit-reversing scheme; that is, the response to an oracle call will be an integer whose digits are the reverse of the oracle's call. For example, for the oracle call 423, the response will be 324. Unfortunately, this scheme does not handle palindrome cases (e.g., 424). Although 'pure reversing' cannot be applied here, it may still lead to a related, effective scheme. Reversing is one kind of 'mirroring.' Perhaps another kind of mirroring will be suitable.

Observation 2. One mirroring scheme that may be effective here is "binary complement," where each bit in the binary representation of a number is replaced by its complementing bit; e.g., the binary complement of 011 is 100.

For the special case where *N* is a power of 2, Observation 2 yields an elegant solution – for every oracle call *i*, the computer program will respond with *i*'s binary complement. However, this scheme may yield an out-of-range response when the original range of integers is not a power of 2; for example, for *N*=13 and the oracle call 1, the binary complement scheme will yield the value 14 (binary 1110). Thus, the utilization of the notion of complement has to be refined.

Observation 3. It is possible to utilize the notion of complement, not in a binary sense but in the distance from the range-ends, i.e., the distance of an integer *i* from the "0-end" is *i*, and its complementing distance, from the *N*-end, is $N - i$.

Since the range size is even, the complement scheme in Observation 3 guarantees the pairing of each range element to a unique complement. Following this scheme, the response to the oracle call 3 in the example of the problem description would be 2, derived from the expression 5-3 (5 is the value of the *N*-end). This is a correct and rather simple solution. But is it possible to have a yet simpler scheme? The next observation sheds additional light.

Observation 4. The notion of binary complement can be applied, on the least significant bit (i.e., the parity bit), but not on all bits of the binary representation, as in Observation 2. The complement of an integer *i* will be the integer whose least significant bit complements *i*'s least significant bit.

This concise and effective binary pattern implies the following calling scheme:

Calling scheme. When the oracle calls an integer *i*, respond as follows: *if i* is even, *then* call $i+1$; *otherwise* (*i* is odd) call $i-1$.

This very simple scheme capitalizes on the notion of bit complement and the invariant property that, before every oracle call, the integers not yet called can be viewed as "pairs of parity buddies." In reflecting on the solution process, one should notice the stepwise refinement through complement schemes. In particular, Observation 2 focused the solution process on binary complement, Observation 3 diverted to range-end complement, and Observation 4 illuminated the sufficiency of single-bit complement. The notion of complement is related to a wide range of computational schemes. It is relevant in assembly language calculations, in logical expression manipulations, in memory allocation ("buddy systems" [Knuth 1973]), and in diverse algorithmic instances that involve complement graphs and sets [Aho et al. 1983; Cormen et al 1990; Manber 1989]. The notion of flipping a single bit at a time is relevant in coding methods, such as Gray codes and routing schemes in cubes and hypercubes [Manber 1989].

3. BIT PARTITIONING

The previous section illustrated the use of the binary representation of integers through the least significant bit. This section illustrates the utilization of the binary representation through the processing of all the bits. It embodies an effective perspective on processing data elements by their "atomic" components. The algorithmic task below is to find the majority in a large set of elements. While this problem has several solutions [Manber 1989], the creative and efficient solution here brings a new point of view, which relates to fundamental sorting schemes. It also illuminates the binary characteristics of majority.

Majority in a Large Sequence of Integers

Given a very long sequence of *N* positive integers, develop an algorithm for determining whether one of the elements in the sequence appears over $N/2$ times. If there is such an element, then output its value, otherwise output 0.

Example. For the sequence of integers: 7 7 9 7 5, the output will be 7; and for the sequence of integers: 7 7 9 7 5 8, the output will be 0.

The solution of keeping a counter for every possible input value is irrelevant, as the range of possible values is too large. The solution of sorting the input elements is also irrelevant, as the input may be very long. A very elegant scheme that does not degrade with large input sequences is based on the concept of disjoint-pairs elimination [Boyer and Moore 1991]. It involves two passes over the input with minimal, $O(1)$ memory and linear computation time. The first pass produces a majority candidate and the second tests the validity of this candidate.

A novel, two-pass method is presented below. It is inspired by the binary representation of integers. Each integer is represented as a sequence of k bits (notice that the bits are numbered from 0 to k) $-\ b_k \ldots\ b_1\ b_0$ (where b_0 is the least significant bit). Viewing the values of bit b_i over the input sequence yields an illuminating observation with respect to majority.

Observation 1. For each bit b_i in the binary representation of the input elements, let z_i be the number of times that b_i is 0 in the input. *If* at least one z_i equals $N/2$, *then* there is no majority element, *else* there is a single candidate for majority, and for every bit b_i of this candidate, b_i is 0 if $z_i > N/2$, and 1 otherwise.

A short example illustrates the above observation. Let the input sequence be 5 2 1 5. The binary representation of the sequence elements is 101 010 001 101. The values of the three z_i 's are $z_0=1$, $z_1=3$, and $z_2=2$ (for example, the value of z_2 is 2 because the value of b_2 is 0 in two of the four sequence elements). Since the value of z_2 is exactly half the number of elements, there cannot be a majority element. Had the 2nd element been 5, all z_is would be different than 2, and there would be a majority value over the sequence for each bit b_i – 1 for b_0, 0 for b_1, and 1 for b_2. These are indeed the bit values of the majority element 5 in the new sequence (5 5 1 5). Yet the z_is all have values different than 2 without having a majority element (e.g., when the 2nd sequence element is 7).

A two-pass scheme can be constructed by following Observation 1: the 1st pass will yield a candidate for majority, based on the z_i values; the 2nd pass will check its validity:

> *1st pass*. Scan the input sequence and calculate its z_i values; *if* some z_i equals $N/2$, *then* output 0 (no majority) and quit, *else* (all the z_is are different from $N/2$) construct the sole candidate element: for every bit b_i of the candidate element; if $z_i > N/2$, then b_i will be 0, otherwise its value will be 1.
>
> *2nd pass*. Scan the input sequence again and count the number of appearances of the candidate; if it is larger than $N/2$, then output the candidate (as the majority element), *else* output 0 (no majority).

One counter is required for each bit in the binary representation of the input elements, and the computation time is linear in N. This scheme encapsulates a stimulating "orthogonal" counting aspect – each input element is decomposed into its bits and the counting over the input sequence is performed separately for each bit. Since the sought-

after value is majority and a bit may have only one of two values, only one counter per bit is required.

The binary perspective inspired decomposition into "atomic" bit components and "orthogonal" counting across the input. Decomposition and "orthogonal" processing of elements by their "atomic" components is useful. It is very effective in the algorithmic schemes of radix sort and bucket sort, which are based on "orthogonal" processing of digits across the input [Aho et al. 1983; Cormen et al. 1990]. It is also most effective in the calculation of *X**N* in *O*(log(*N*)) time (modular exponentiation [Cormen et al. 1990], and in a couple of graph algorithms. Gabow's scaling algorithm for single-source shortest path is based on uncovering bits in the binary representation of graph-edge weights [Cormen et al. 1990], and the 1-1 mapping between Euler circuits and De-Bruijn sequences is derived from path decomposition by bits [Even 1979].

4. BINARY LOCATIONS

The previous two illustrations involve binary perspectives related to binary representation of integers. The following illustration presents a binary perspective related to classification of element addresses in a given structure. Processing by absolute or relative element locations in a given data structure is a basic means in algorithmic problem-solving. The algorithmic challenge in the following illustration involves the development of a game-winning strategy based on hidden binary patterns of element locations.

Coin Collection

Given a line of coins, two players remove the coins and collect them. The coins are quarters, dimes, nickels, and pennies. Each player in turn removes and collects a coin either from the left end or the right end of the (remaining) line, according to his or her choice. The game ends when all the coins are removed (and collected). The winner is the player who has collected the larger total value.

The total number of coins, *N*, is even and the total value of the coins is odd (thus the game cannot end in a draw). The number of coins from each type is random.

Develop a winning strategy for the 1st player (who starts the game).

Example. For *N*=6 and the line of coins, 10 25 1 25 1 5, the 1st player may remove the leftmost coin (10); then the 2nd player removes the leftmost (25); the 1st, the rightmost (5); the 2nd, the rightmost (1); the 1st, the rightmost (25); and the 2nd, the remaining coin, (1). The 1st player wins the game, as she or he has collected a larger total value (40). Had the 1st player started the game by removing the rightmost coin, he would have lost the game.

There is a total of *N* moves in the game, *N*/2 for each player. At first glance, one may offer a greedy approach for removing the larger end at each turn. However, this may not necessarily lead to a win, as can be seen in the example, 5 10 1 1. One may suggest a refinement, which is still greedy: to perform the removal from the end that yields "the better end-delta, " that is, subtract from each end the value of the next-to-the-end coin from the end-coin and perform the removal from the end that yields the larger result (in the latter example, 5-10<1-1; thus, remove the rightmost coin). Unfortunately, this may also fail, for example in 10 25 25 1 5 1. (The 1st player will remove 1; the 2nd, 10; the 1st, 5; the 2nd, 25; the 1st, 25; and the 2nd, 1.) Remember that the 2nd player may follow any strategy he or she likes.

One correct way to approach the game is by unfolding top-down, from the initial line to the game's end, all the possible scenarios, and identifying a path leading to victory. The straightforward implementation is with game trees [Aho et al. 1983]. The time complexity per move is exponential in *N*, making this solution relevant for very short initial lines only.

A more insightful approach is bottom-up, from the game's end to the initial line, capitalizing on the nonexponential number of possible game states. Due to the characteristic that coins are removed from the end of the lines, the total number of game states is quadratic in *N*, as there may be only *N*–1 different sublines of 2 coins (just before the end of the game); *N*–2 different sublines of 3 coins; and 2 different sublines of *N*–1 coins (the *N*–1 leftmost coins and the *N*–1 rightmost coins). Using the dynamic programming algorithm design technique [Cormen et al. 1990], it is possible to build a table that will hold the preferred move for each subline that may be reached during the game. (The construction of the table will be from the shorter, end-game, sublines up to the initial line.) The time and space complexities are quadratic in *N*.

Although the bottom-up solution is a significant improvement of the top-down solution, it only capitalizes on partial characteristics of the game. Further insight can be gained and utilized. By examining the coin removal process carefully, one may see illuminating characteristics, other than the limited number of intermediate game states. These characteristics involve parity.

Observation 1. The first two coins that are removed are collected from different parity locations, one coin is removed from an initially even location and the other from an initially odd location. Moreover, this characteristic holds not only for the 1st and 2nd coins removed, but also for the 3rd and 4th, 5th and 6th, and so on.

Observation 2. The initial line locations can be denoted according to their parity, i.e., initially even locations denoted 0 and initially odd locations denoted 1. Thus, the initial line locations can be viewed as 1 0 1 0 … 1 0. After every even number of turns, one end coin will reside on a 0 location, and the other on a 1 location (and after every odd number of turns, both end coins will reside on equal value locations).

Observation 3. The 1st player always plays after an even number of turns. Following Observation 2, the player has a choice on each turn between a coin on a 0 location and a coin on a 1 location. Hence, throughout the game the player can collect all the 0 location coins or all the 1 location coins.

The above binary perspective yields a very elegant winning strategy. Since the total coin value is odd, it distributes unevenly between the even and odd locations. At the beginning of the game, the 1st player just needs to compare the total coin values of the even locations with the total coin values of the odd locations and play accordingly. The game rules are as follows:

Winning strategy for the 1st player. Calculate the total coin values in the initially even locations and the total coin values in the initially odd locations; *if* the former is larger, *then* remove at each turn the end coin that is in an initially even location (the other end coin must be in an initially odd location); *otherwise* remove at each turn the coin that is in an initially odd location.

In the example of the problem definition, the coin values in the initially even locations are much larger than the total in the initially odd locations (25+25+5>10+1+1). Thus, going for the initially even locations throughout the game guarantees the 1st player victory.

The above scheme is very efficient, as it only requires simple, linear time preprocessing and no extra space. The scheme is based on the invariant properties that are formulated in Observation 1. Observation 2 in particular captures the essence of the game characteristics with binary location classification. This classification embodies a powerful binary perspective: the partition of selected problem elements into two 0/1 classes according to the parity of their addresses.

Partition based on address parity appears in various algorithmic schemes. The array locations of heap elements are partitioned according to parity – all the left sons are in even locations and all the right sons are in odd locations (for the heap node in the *i*th array location, the left son is kept in location 2*i*, and the right son in location 2*i*+1 [Aho et al. 1983]). Any path in a bipartite graph partitions the nodes into two groups – those in the even path locations and those in the odd path locations [Manber 1989]. In computational geometry, the indication of whether a given point is inside or outside a polygon is derived from the parity of the number of lines separating the point location and the surroundings of the polygon [Manber 1989].

The winning strategy for the 1st player encapsulates a loop whose body is composed of a pair of consecutive turns, one by each player. The strategy (loop) development was derived from the invariant properties specified in Observations 2 and 3. This demonstrates the approach advocated by computer scientists, that loop design should be done hand-in-hand with its underlying assertions [Dijkstra 1976; Gries 1981].

5. PARITY ARITHMETIC

The illustration in the previous section involved a binary perspective related to the parity of element locations. The following illustration displays a different parity aspect – parity patterns in arithmetic operations. One important aspect of the illustration is the equivalence between the original task statement, with integer values, and a reduced task statement with 0/1 values. The illustration involves a two-player game with a line of numbers, in which the task is to guarantee that a sequence of arithmetic operations will end in a desired result.

Force an Odd Result

Given the line of the integers 1 2 3 … *N*, where *N* is even, two players apply the arithmetic operations +, –, and x. between pairs of neighboring numbers until only one number remains. Each player, in turn, applies one of the operations, according to his or her choice, on any two neighboring integers. The operation yields a new integer instead of the neighboring pair. The game ends when only one integer remains in the line. The 1st player wins the game if the remaining integer is odd. The 2nd player wins if the remaining integer is even.

Develop a winning strategy for the 1st player.

Example. For *N* =4, the initial line is 1 2 3 4. The 1st player may apply "+" between the last two integers, and the resulting line will be 1 2 7. The 2nd player may apply "x" between the first two integers, and reduce the line to the pair 2 7. The 1st player will apply "+",

yielding the remaining integer 9, and win the game. (Notice that had the 1st player started the game by applying "x" between the last two integers, the player could have lost the game.)

There is a total of *N*–1 moves in the game. Since *N* is even, the 1st player makes the last move. Had the goal of the 1st player been an even integer at the end, her winning strategy would be trivial. Her only important move would be the last one: if the parity of both remaining integers is identical, she would apply "+", otherwise she would apply "x". Obtaining an odd result is much harder. As in the previous illustration, one way to approach the task is top-down, with a game tree that will unfold all game scenarios. Unfortunately, the total number of possible paths is exponential in *N*, making this solution relevant for very short initial lines only. A different approach, which capitalizes on further insight, should be employed. One can make an initial observation when examining the operations "+"and "–" with respect to parity:

Observation 1. There is no difference between applying "+"or "–" with respect to the parity of the result; therefore the original game can be reduced to a game without the operation "–".

The elimination of subtraction is one simplification that evolves from the notion of parity. Another, more significant, simplification derives from reducing the integer values in the game to binary 0/1 values. Originally, any integer could be present in the initial line. However, as only parity is of interest and the relevant operations are now "+"and "x", the following observation yields a simplifying point of view:

Observation 2. Each of the integers in the initial line can be seen as either 0 or 1; an even integer can be seen as 0 and an odd integer as 1. This derives from the equivalence between an even integer and 0 and an odd integer and 1, with respect to parity under multiplication and addition.

Following the above observation, the initial line can now be seen as 1 0 1 0 … 1 0, and the goal of the 1st player is to yield 1 in the last move. At this stage, some common heuristics will be useful.

Heuristic 1. *Backward reasoning*. In order to yield 1 in the last move, the 1st player must "receive" from her opponent a two-element line which includes at least one 1. This implies that the line the 1st player should yield in her next-to-the-last move is either 1 1 1 or 1 0 1.

Heuristic 2. *Examination of simple cases*. Examination of the very simple case of the *4-element* initial line 1 0 1 0 shows that the 1st player can easily yield 1 0 1 in her first move. Examination of the *6-element* initial line 1 0 1 0 1 0 shows that if the 1st player will yield the line 1 0 1 0 1 in her first move, she will be able to yield 1 0 1 in her next move, regardless of the opponent's move (for example, if the opponent will yield 1 0 0 1, then the 1st player's move will be addition of the two middle elements).

Heuristic 3. *Generalization.* Generalizing from the 4-element and 6-element cases, the 1st player will be able to yield in each of her moves an interleaved line of the binary form 1 0 1 0 … 1 0 1. This is verified by realizing that there is a *response* to each possible

"interleaving disruption" by the opponent – a pair of 0s can be transformed to one 0 and a pair of 1s to one 1.

All in all, concluding from the above, a simple invariant underlying the 1st player's winning strategy can be specified:

Game invariant. In each of her moves, the 1st player will yield a line of integers of the interleaved form: *odd even odd even … odd even odd.*

Game rules for the 1st player. Examine the line at the beginning of a move*; if* it is of interleaved form (i.e., either the form *odd even … odd even* or the "opposite" form *even odd … even odd); then* add the *even* end element to its adjacent (*odd)* element; *otherwise* there are (somewhere, not in the line ends) two adjacent elements of the same parity – if they are both even then add them; if they are both odd then multiply them.

In reflecting on the development of the solution, one notices the essential role of the binary 0/1 perspective. Observation 2 introduced a 0/1 pattern that laid the foundation for progress through the three heuristics that followed. The interleaved 0/1 abstraction focused the analysis on the core problem characteristics. The design of the game invariant, i.e., preserving the interleaved 0/1 line throughout the game, then elegantly provided the desired result.

The reduction of general integer values into binary values is a relevant simplification tool. One example of its effectiveness is the "zero-one principle" in sorting networks, which states that "if a sorting network works correctly when each input is drawn from the set {0,1}, then it works correctly on arbitrary input numbers" [Cormen et al. 1990, Ch. 28, p. 639]. Another example is viewing the QuickSort partitioning stage as a "binary ordering" in the sense that all elements smaller than the pivot are put at one end of the sorted array and all those not smaller than the pivot are put at the other end (e.g., Cormen et al. [1990]).

An additional aspect, which evolves from the above game-strategy design, is "parity arithmetic," in the sense of the closure properties of basic operations (such as addition and multiplication) under parity as formulated in Observation 2. Computer chess is an application that involves parity arithmetic, where the sum of the coordinates of every white square is even and the sum of the coordinates of every black square is odd [Frey 1983]. In a broader sense, all applications that involve calculations with modulo-2 numbers involve parity arithmetic. This kind of arithmetic is a very simple instance of modulo-*N* arithmetic [Cormen et al. 1990].

As in the previous game, the principle underlying the winning strategy in the above game is an invariant property. The invariant was reached in a heuristic search through Heuristics 1, 2, and 3. The backward reasoning and inductive generalization heuristics are essential problem-solving elements [Polya 1957; Schoenfeld 1992]. The demonstration of their effective utilization is a didactic contribution to the process of reaching the underlying game invariant.

6. CONCLUSION

This article focused on illustrating the effectiveness of binary perspectives in a variety of aspects through four appealing algorithmic challenges. The binary perspectives follow:

- binary complement in the 1st illustration;
- binary representation of numbers and processing bit-by-bit in the 2nd illustration;
- 0/1 addressing derived from location parity in the 3rd illustration;
- reducing an integer task into a 0/1 task, and parity arithmetic in the 4th illustration.

The solutions to all the challenges were presented gradually. They embedded essential algorithmic problem-solving considerations in conjunction with binary perspectives:

- *Stepwise refinement*. The 1st illustration started with the notion of "reversing," continued with "mirroring," and concluded with "complement."
- *Partitioning into atomic components and processing "orthogonally" across the input*. The 2nd illustration included separate processing across the input of each bit b_i in the binary representation of the input elements.
- *Invariance properties*. Each of the 3rd and 4th illustrations involved the recognition of an invariant property underlying the algorithmic solution. The invariant properties encapsulated interleaving related to parity – odd/even locations in the 3rd illustration and modulo-2 values in the 4th illustration.

The solution process in each illustration involved general problem-solving heuristics.

- *Decomposition*. The 1st and 2nd illustrations involved processing decomposed data elements. The 1st illustration zoomed specifically into the least significant bit of the data elements, and the 2nd illustration engaged all bits, one-by-one.
- *Auxiliary coloring.* The 3rd illustration employed "coloring" of element locations by "0" and "1" in illuminating the underlying invariant properties of the problem at hand.
- *Backward reasoning and inductive generalization*. The 4th illustration displayed gradual advancement, with "arity arithmetic" towards the game invariant through the appropriate heuristics.

The binary perspective aspects in all the illustrations were tied to a variety of referenced algorithmic schemes and applications. These schemes and applications represent a broad spectrum of algorithmic domains, including sorting, graph algorithms, routing, coding, and computational geometry.

In our experience, the illustrations can be presented to a class in two ways: either altogether, where the focus is on algorithmic problem-solving considerations and perspectives; or one-at-a-time, each illustration in a different place in the material. The former is most relevant for elaborating general methods and techniques. The latter is helpful for motivating a new topic or illuminating its presentation with an attractive illustration. We used both ways in the "Introduction to Algorithms" course. The "altogether" presentation of the illustrations was used in instructing college teachers and emphasizing pedagogical aspects. The "one-at-a-time" presentation was used with motivated Olympiad students, who related the lessons from having solved the above challenges to the material being studied. In both cases, the students were enthusiastic, and indicated that the study of this material expanded their values and perspectives and enhanced their scientific point of view.

References

AHO, A. V., HOPCROFT, J. E., AND ULLAM, J. D. 1983. *Data Structures and Algorithms*. Addison-Wesley, Reading, MA.

BOYER, R. S. AND MOORE, J. S. 1991. A fast majority vote algorithm. In *Automated Reasoning: Essays in Honorof Woody Bledsoe*. Kluwer, Dordrecht, 105-117. (Note: the algorithm was invented by the authors in 1980).

BRIGHT, G. W., HARVEY, J. G., AND WHEELER, M. M. 1980. Game constraints, player verbalizations, and mathematical learning. *J. Exper. Edu. 49* (1980), 52-55.

CHI, M. T. AND BASSOK, M. 1989. Learning from examples via self-explanations. In *Knowing, Learning, andInstruction.* L. B. Resnick, Ed. Lawrence Erlbaum, 251-282.

CLANCY, M. J. AND LINN, M. C. 1999. Patterns and pedagogy, In *Proceedings of the 29th SIGCSE TechnicalSymposium on CS Education,* ACM, New York*,* 37-42.

CORMEN, T. H., LEISRSON, E. L., AND RIVEST, R. L. 1990. *Introduction to Algorithms*. McGraw Hill. New York.

DIJKSTRA, E. W. 1976. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, NJ.

EVEN, S. 1979. *Graph Algorithms*. Computer Science Press, Rockville, MD.

FREY, P. W. (ED.) 1983. *Chess Skill In Man and Machine*. 2nd ed., Springer, New York.

GRIES, D. 1981. *The Science of Programming*. Springer, New York.

KNUTH, D. E. 1973. *The Art of Computer Progammin., vol.*1. 2nd ed. Addison-Wesley, Reading, MA.

LINN, M. C. AND CLANCY, M. J. 1992. The case for case studies of programming problems. *Commun. ACM 35* (1992), 121-132.

MANBER, U. 1989. *Introduction to Algorithms – A Creative Approach*. Addison Wesley, Reading, MA.

NESHER, P. 1989. Microworlds in mathematical education: A pedagogical realism. In *Knowing, Learning, andInstruction.* L. B. Resnick (ed.), Lawrence Erlbaum, 187- 215.

POLYA, G. 1957. *How to Solve It.* Princeton University Press, Princeton, NJ.

SCHOENFELD, A. H. 1992. Learning to think mathematically: Poblem-solving, metacognition, and sense making in mathematics. In *Handbook of Research on Mathematics Teaching and Learning*. D. A. Grouws (ed.), Macmillan, New York, 334-370.

Received April 2002; revised July 2002; accepted August 2002