Informatics in Education, 2009, Vol. 8, No. 2, 217–226
© 2009 *Institute of Mathematics and Informatics, Vilnius*

The Overlooked Don't-Care Notion in Algorithmic Problem Solving

David GINAT
CS Group, Science Education Department, Tel-Aviv University
69978 Tel-Aviv, Israel
e-mail: ginat@post.tau.ac.il

Received: September 2009

Abstract. The notion of "don't care", that encapsulates the unimportance of which of several scenarios will occur, is a fundamental notion in computer science. It is the core of non-determinism; it is essential in various computational models; it is central in distributed and concurrent algorithms; and it also is relevant in sequential, deterministic algorithms. It is a valuable tool in algorithmic problem solving. Yet, in the teaching of (deterministic) algorithms it is not made explicit, and left unexplored. Its implicit exposition yields limited student invocations and limited student comprehension upon its utilization. These phenomena are also due to its rather unintuitive "black-box" characteristic. In this paper, we illuminate and elaborate on this notion with six algorithmic illustrations, and describe our experience with novice difficulties with respect to this notion.

Keywords: algorithmic problem solving, non-determinism.

1. Introduction

Algorithmic schemes involve concrete actions, which lead an execution from a well-defined starting point to a desired goal. Two usual themes in these schemes are: an apparent starting point and the utilization of all the information specified in the algorithmic task. In addition, it often is the case that the order of the task's input is relevant, and the order of an algorithm's sequence of operations is important.

However, there are algorithmic schemes in which the starting point may be selected randomly, with no explicit, well-defined specification (e.g., maximal matching in a bipartite graph (Manber, 1989). In addition, sometimes only part of the task's information is relevant for the computation, and some of the information is unnecessary (e.g., branch-and-bound computations (Cormen *et al.*, 1991)). In some cases, the ordering of the information, or the ordering of the computation operations does not matter or may not be a-priori assumed (e.g., distributed algorithms (Lynch, 1996)). In many such cases, the design perspective of the algorithmic scheme may not be intuitive, or straightforward.

The characteristic common to all the above cases is a "don't-care" notion, encapsulated in the algorithmic scheme. The "don't care" notion is a fundamental notion in computer science. It is the core of non-determinism. It appears in computational models, in asynchronous systems, in some programming languages, in distributed and concurrent algorithms, and also in sequential, deterministic algorithms.

The idea of ”not caring” for supposedly relevant occurrences is unintuitive. The natural tendency of algorithm designers, in particular at the novice level, is to be as concrete as possible, at all the computation stages, and utilize all the tasks’ information, as stated. This evolves, probably from people’s natural tendency to a concrete, deterministic viewpoint, which does not encapsulate any non-determinism or ”black box” phenomena (Armoni and Gal-Ezer, 2007; Armoni *et al.*, 2008).

Although it is unnatural and unintuitive, computer science students should be aware of the ”don’t care” notion, comprehend it, and employ it, in diverse computer science domains, including that of algorithmic problem solving. Unfortunately, in our experience, this is not quite the case.

The objective of this paper is to illuminate and elaborate on the ”don’t care” notion in algorithmic problem solving, and describe our primary experience with student difficulties with respect to this notion. We observed two kinds of difficulties – overlooking, or even unawareness of this notion, and puzzling reactions upon seeing its employment in algorithmic solutions.

In the next section we display various cases of the ”don’t care” notion in algorithmic problem solving, and reveal our initial observations of student behaviors in each case. We then discuss our observations and advocate the need of further elaboration of this notion in algorithmics.

2. ”Don’t’ Care” Occurrences

In this section, we name, exemplify, and discuss ”don’t care” occurrences. In Subsection 2.1 we display the case of not caring for an algorithm’s starting point; in Subsection 2.2 we present the case of not caring for parts of an algorithmic task’s information; and in Subsection 2.3 we display the cases of not caring for the order of algorithmic operations and not caring for the input order. We introduce two examples for each case, and describe our primary experience with students in these examples.

2.1. *Random Starting Point*

Each computation starts from some starting point. In many computations, the starting point is obvious. For example, in the summation of N values, the computation starts with the initialization of a *Sum* variable. However, in some algorithmic tasks, there may be many possibilities for the starting point, and one may intuitively feel that it is very relevant to recognize the suitable one, select it and advance accordingly. Yet, sometimes, there is no particular suitable one, as we demonstrate in the following two examples.

Separating between foes. Given a list of N diplomats, each having 0, 1, 2, or 3 foes (among the other diplomats), develop an algorithm which splits the diplomats into two groups, A and B, such that each diplomat does not have more than one foe in her own group. We consider a rivalry between diplomats to be symmetric; thus if p is a foe of q, then q is also a foe of p.

At first glance one may wonder if the goal may always be obtained. Indeed, attainability is not that obvious. One way of getting convinced of attainability is the design of an algorithm that will always yield the desired goal.

Novices who approach this task seek an algorithm that will start from a well defined, possibly advantageous starting point, and steadily progress to the desired goal. The more common algorithm they design is based on the following intuitive scheme. They put a selected diplomat with maximal number of foes in group A, and all her foes in group B. Then, they put another such diplomat in group A, and all her foes in group B, and so on. Some offer a supposedly-improved version in which one foe is left in group A and the others are put in group B.

Unfortunately, one may find concrete examples that fail both of these ideas, as diplomats that were previously put in group B may have too many foes in their group once the newly-processed diplomat and her foes are added to the groups. Once they realize the difficulty, some offer various "patches" of transferring newly added foes from one group to another. This yields back-and-forth transfers, of which the students are unable to argue successful termination. Quite a few wonder whether the goal may always be attained.

At this point we try to direct them to the suitable solution. This solution does involve repeated transfers of diplomats between the groups; but unlike the original novices' inductive-construction idea, it is based on inductive-refinement, of an initial division of the diplomats into the groups A and B. The initial diplomats' division is random. In each step, a diplomat with more than one foe in her group is sought after. If no such diplomat is found, then the goal has been attained. Otherwise, this diplomat is transferred to the other group. Surprisingly, this process always yields attainability!

Students are rather surprised. The starting point is random, and diplomats are repeatedly transferred between the groups; but "magically" this process always yields successful termination. How can this be? This seems intriguing to many.

The key point is that one can define a simple metric, with which to argue progression of the algorithm. The metric is: *the num of rivalries in group $A+$ the num of rivalries in group B*. Each step of the above algorithm reduces the value of this metric (since the transfer of a diplomat reduces the num of rivalries in her group by 2 or 3, and increases the num of rivalries in the other group by at most 1). Since at the beginning the value of the metric does not exceed $3N$, and this value repeatedly decreases, the goal will always be attained.

Some students understand and appreciate the underlying idea; others find it hard to be convinced by this very unintuitive process. The important lesson yielded from the above description is that both the notion of a random starting point and the notion of "metric argumentation" seem rather unnatural and unintuitive to students, and should be illuminated and elaborated.

On the verge of an empty tank. N gas stations are located along a cyclic road. In each station, there is a limited amount of fuel. The total amount of fuel in the N stations is sufficient for completing exactly one round of the cyclic road. In some stations the fuel amount may be more than needed for reaching the next station, and in others – less than needed for reaching the next station. A driver should start driving her car from one station and complete one round (while filling fuel in every visited station), without ”getting stuck” with no fuel. The goal is to efficiently compute a suitable starting point, given the amounts of fuel in the N stations, and the fuel needed for driving between every two neighboring stations.

As in the previous task, here too, one may wonder if the goal may always be attained. That is, is there always a station from which one can start and complete a round? Obviously, a station in which the fuel is insufficient for reaching the next station is not a candidate for a starting point. On the other hand, perhaps there is more than one starting point (e.g., in the case that the fuel is split evenly among the stations, any station is a valid starting point).

Novices offer various solutions to this task. Some offer the straightforward solution of trying every station as a starting point, until finding a valid one. This solution is correct but inefficient ($O(N^2)$ time and $O(N)$ space), as it requires repeated passes over the input. Others suggest the solution of starting from the station with the maximal amount of fuel. This solution requires only a single pass over the input, but it may yield incorrect output. Some propose an inductive idea in which they group together adjacent stations where one can drive without ”getting stuck” from the first to the last station in the group. This solution encapsulates valuable insight, but it does not lead (in our experience) those who propose it to an efficient implementation.

Seeking further insight, while combing the ideas considered above, we may obtain an elegant and efficient solution. We may start from any station, and check the value of the fuel remaining in the car’s fuel-tank at each station, prior to filling fuel in that station. In this process, we may allow (artificially for our story) negative values of the fuel remaining in the tank. Thus, at the beginning, the value of the remaining fuel will be 0. After advancing one hop (and reaching the next station), the value of the remaining fuel will be the amount of the fuel filled in the starting station minus the fuel needed to drive that hop. After advancing two hops, the value of the remaining fuel will be the sum of the amounts of fuel filled in the first two stations minus the amount of fuel needed to drive these two consecutive hops; and so on.

A valid station, from which the cyclic drive may be performed successfully, is the station for which the value of the remaining fuel, in the above process, is minimal (there is at least one such station, but perhaps more than one).

Again, students are rather surprised. The starting point is random, and we just seek a minimum value, in an ”artificial” cycle, which allows negative remaining-fuel values. Again, ”magically” this process always yields the desired output. And it is elegant and efficient ($O(N)$ time and $O(1)$ space).

The key point is that no matter which station will be chosen as the starting point, the minimum value of the remaining-fuel will always be in the same location(s). And, if the drive will start from such a location, the value of the remaining fuel will never be negative (since otherwise, the location were it will become negative should have been the minimum).

Once more, some students understand and appreciate the underlying idea, while others are not fully convinced by this unintuitive solution. Once more, the notion of a random starting point seems unnatural to students, this time in a "supposedly indirect" computation that directly yields the desired goal.

2.2. *Unnecessary Task Information*

In most algorithmic tasks one is expected to utilize all the information specified in the task. However, sometimes, proper insight reveals that the task may be solved by using only part (or even a small part) of the task information, and one may actually not care about the rest of the information. We demonstrate it with the following two algorithmic tasks.

> **Colliding Balls.** N small balls are put on a track, each in a different location. At time t each ball is rolled towards one of the ends of the track. The leftmost ball is rolled right, the rightmost ball is rolled left, and the rolling direction of each of the other balls is arbitrary. Each ball is rolled in the same velocity v. In time, balls collide. Upon a collision between two balls, each ball switches its rolling direction and rolls in velocity v to the opposite direction. A ball that reaches one of the track ends leaves the track. Given v, the balls constant velocity, the locations of the track's beginning and end, and the N locations and rolling directions of the balls, compute the time in which all the balls will leave the track.

As in the previous tasks, here too it may not be obvious that the process will end (all balls out of the track). Quite a few novices feel at loss upon approaching the task, as the only solution that they can come-up with is a computation that simulates the physical occurrences described in the task's specification. Such a computation is not trivial to implement, and one still remains unclear about the question of whether this process always terminates.

Proper insight is required. The key point here is that we do not need to explicitly simulate the rolling scenario of each ball. Rather, since the balls are identical and roll all the time at the same velocity, we may artificially view each ball as continuously rolling in its original direction. That is, when balls A and B collide we may now view ball A as becoming ball B, and vice versa. Adopting this viewpoint, the task becomes trivial, and we may see that the output depends <u>only</u> on the locations of the leftmost and the rightmost balls. The one that is further from the end towards which it rolls yields the ending time of the rolling process.

Some students rather quickly comprehend the above unintuitive, intriguing viewpoint, whereas others remain puzzled. They check and recheck that this viewpoint is indeed valid, as it yields the use of very small part of the information of the task specification. The notion of not using information that initially seems relevant, is not natural, and requires illumination and elaboration.

> **Shrinking-piles game.** Given N piles of sizes $2^0, 2^1, 2^2 \ldots 2^{N-2}, 2^{N-1}$, two players decrease the pile sizes according to the following rule: each player, on her turn, decreases by 1 any K of the piles ($N > K > 3$) that she wishes. (On different moves a player may "touch" different piles.) Eventually, there will be less than K piles. The

player who makes the last move, which yields less than K piles, wins. Devise a strategy for winning the game.

In our experience, many students like this kind of tasks, as it involves a game, which they may play with each other. However, only some of them obtain the desired insight and properly devise the winning strategy.

Students play the game with each other, for small values of N and K. Some are more systematic in their search for a solution, and some are less systematic. They notice that the smallest pile is rather special, as it is of an odd size, whereas all the others are of an even size. They all notice that each pile is twice the size of its previous pile. Some students observe that, actually, each pile is not only twice its previous pile, but also greater than the sum of all its previous piles. This is one of the properties of binary numbers. Many try some heuristic ideas based on the latter observations, but do not yield sound and convincing winning strategies.

Yet, further insight of the last observation yields the elegant answer. If we play a game in which $K = 2$, and pile p will be "touched" in each move together with one of the piles smaller than p, all the smaller piles will disappear (shrink to 0), while p will still remain in the game. This is due to the characteristic that p is greater than the sum of all the piles that are smaller than p.

Generalizing the above to our original game, the $K - 1$ largest piles will never be empty in the end of the game. This assertion holds even if each of these $K - 1$ piles will be "touched" in each move of the game. So, we may not care for the sizes of these $K - 1$ largest piles during the game, as they will always "be there".

A winning strategy follows. We may focus only on the $N - K + 1$ first (small) piles. The game will end when they will all be 0. The first of these piles is of an odd size (size 1), and the rest of them is of an even size. We will play the first turn, and: zero the first pile, not "touch" the others, and remove 1 from each of the largest $K - 1$ piles. This will yield a situation in which each of the $N - K + 1$ smaller piles in the game will be of an even size (the smallest pile will be of size 0), after our turn. At this stage, each one of them will shrink to size 0 after being "touched" an even number of times.

From this point on, we will "touch" in each of our following turns exactly the piles that our opponent "touched" just before our turn. This will guarantee the invariant property that after each of our moves, all the $N - K + 1$ smaller piles in the game will be of an even size. Since the game end depends only on these piles, we will be the player who plays the last turn that yields 0 in all these piles. (Note that different piles among these piles may be zeroed in different stages of the games (e.g., the first pile is zeroed already in the first turn). Eventually they will all be zeroed.)

The above game solution is based on the idea that we do not care for some of the game information – the sizes and the parities of the largest $K - 1$ piles during the game. We know that all these sizes remain positive throughout the game. Some novices take time to be convinced, as the above strategy involves not only the "don't care" notion, but also the notion of invariance (Dijkstra, 1976; Ginat, 2001; Gries, 1981). Both notions are essential, and most relevant in the teaching of algorithmic problem solving.

2.3. *Unimportant Order*

All algorithmic tasks involve sequences of operations. Many algorithmic tasks involve sequences of data. In most algorithmic tasks the order of the sequences of operations is significant. In many data-sequence tasks the order of the given data is important. Yet, this is not always the case. Sometimes, although it may seem relevant, the order of the algorithmic operations is unimportant, and so is the order of the information. We display this with the following two tasks.

Chocolate bar. Given a chocolate bar of size $N \times M$, one may break the bar into pieces in horizontal and vertical lines. Devise an algorithm that outputs the minimal number of breaking operations that yields $N \times M$ pieces of size 1×1. (Note: a rectangular piece may only be broken into two rectangular pieces; and pieces may not be grouped and broken together.)

Novices examine different paths of breaking the bar. Some just try arbitrary ways. Others attempt small bars (e.g., $1 \times M$) first, and larger ones later. Many turn to the idea of binary search and try to apply it here. Unfortunately, this does not help much, since it yields a rather cumbersome computation when N and M are not powers of 2. As a result, many devise a computation that unfolds all the possible breaking sequences and selects the best one.

The key point is to carefully examine a single breaking operation. In such examination one may notice that every breaking operation increases that total number of pieces by exactly one, no matter how it is performed. The initial number of pieces is 1, and the final number is $N \times M$; thus, any sequence of operations must be of $N \times M - 1$ breaking operations. The order of the operations does not at all matter.

Some students notice the above characteristic, particularly after some gradual attempts with various bars. Others notice that the order of the operations does not matter, but are unable to provide the convincing justification. Some are not completely convinced of the unintuitive, "don't care" answer even after seeing the above justification. Again, further elaboration of the "don't care" notion is needed.

Wire connections. N red dots and N blue dots are spread on a line, each in a separate location. The N red dots are indexed $1..N$, and so are the N blue dots. Each *red-i* dot is to the left of its corresponding *blue-i* dot. One would like to connect with a wire each *red-i* dot to its corresponding *blue-i* dot; and would like to know the total length of the required wire. Given the list of the locations of the N red dots and the N blue dots, can you obtain the desired goal in $O(N)$ time and $O(1)$ space, without any assumption on the input order?

Obviously, the computation can be easily performed in $O(N)$ time and $O(1)$ space if the locations of corresponding dots are adjacent to one another in the input. But what if we make no such assumption on the input order?

Novices offer two solutions – a straightforward, very inefficient solution, and an additional, more efficient solution, which is, unfortunately not efficient enough. The straightforward solution is based on the idea of storing the input in memory, and computing separately for each dot-pair i the wire length required to connect *red-i* with its corresponding *blue-i*. This computation requires $O(N^2)$ time and $O(N)$ space.

The better offered solution scans the input only once, using an array with a separate entry for each dot-pair i, for keeping the distance between *red-i* and its corresponding *blue-i*. The initial value of each such entry is 0. When the location of *blue-i* is read from the input, it is added to the current entry value; and when the location of *red-i* is read from the input, it is subtracted from the current entry value (recall that the desired distance is obtained from the *blue-i* location minus the *red-i* location). At the end of the input scan, each array entry will keep the distance between each dot pair. This computation requires only $O(N)$ time, but still – $O(N)$ space. Some novices conjecture that attaining the required $O(N)$-time and $O(1)$-space measures must only be done with some assumption on the input order.

However, one can meet these measures without any assumption, by taking the latter offered solution one step further. We are only interested in the total sum of the distances between pairs. When we look at the computation of this total sum, we notice that each *red-i* location is added to this sum with a negative sign, and each *blue-i* location is added to this sum with a positive sign. The intuitive way is to view these two values as one entity, and have them adjacent in the summation process. But, the end result will still be the same if they are not adjacent! The same summation will be obtained $\underline{\text{regardless of}}$ the place of these values in the input. The computation can be a simple summation in which: whenever the location of a red dot is read, it will be added to the total sum with a minus sign, and whenever the location of a blue dot is read, it will be added to the total sum with a plus sign. This very simple scheme requires only $O(N)$ time and $O(1)$ space.

All in all, the above example displays the case of starting with an intuitive feeling that the order of the input data does indeed matter, and realizing that it actually is not. Some students reach the above illumination by themselves, but many remain with the notion that each dot-pair distance has to be separately computed. In order to ”depart” from this impasse, one needs to recognize that what may seem to be an atomic entity (dot pair, here) need not be atomic, and may be broken into sub-entities that can be processed separately.

3. Conclusion

The latter section displayed various occurrences of the ”don’t care” notion, and revealed novice difficulties in employing, and sometimes comprehending this notion. The displayed occurrences involved cases of random starting points of computations, unused task information, and unimportant data ordering.

The ”don’t care” notion seems unintuitive, as it encapsulates some non-deterministic characteristics. Problem solvers feel more comfortable when all the elements in a solved problem are concrete and explicit, and feel less comfortable when solution elements enclose some fuzziness, of ”black box” characteristics. The latter seems to occur due the need of a higher level of abstraction.

Nevertheless, the notion of ”don't care” is relevant in algorithmic tasks. Although unintuitive, computer science students should be aware of it, and invoke it when suitable. In a sense, we may view this notion as one more tool that they may consider upon solving algorithmic tasks. This tool should be added to their problem-solving toolbox, which includes means like backward-reasoning, task-decomposition heuristics (including recursion), inductive progression, auxiliary variables, and the like. In attempting algorithmic solutions, one may consider these tools, and select those that seem most appropriate for the task at hand.

In the examples demonstrated in the previous section, the ”don't care” notion appeared with additional algorithmic problem solving notions, such as the notion of using a metric (for showing progress), the notion of invariance, and the notion of atomicity in viewing task entities. The solution of algorithmic tasks may often require the invocation and the utilization of a couple, or more, of such notions.

In this study novices overlooked the relevant ”don't care” notion, possibly due to lack of familiarity with this notion and its applications, and due its unintuitive nature. Any problem solver tries the intuitive way first. However, experts consider additional ways, whereas novices tend to keep following their initial direction (Schoenfeld, 1992). In previous studies, we have shown various occurrences of this phenomenon with the particular cases of greedy traps (Ginat, 2003a) and design-by-keyword tendencies (Ginat, 2003b). This paper sheds light on the additional case of overlooking the notion of ”don't care”, and possibly even demonstrating impasses in its comprehension.

”Don't care” applications are apparent and relevant in algorithmic problem solving. Unfortunately textbooks do not make this explicit enough, and do not dedicate some discussion to this notion. In this sense, there also is an ”overlooking phenomenon” on the part of the tutors, in the sense that they overlook the importance of making this notion explicit. This is particularly unfortunate in light of the findings regarding the difficulties that students demonstrate with non-determinism (Armoni and Gal-Ezer, 2007; Armoni *et al.*, 2008).

We believe that the ”don't care” notion should be made explicit and discussed with students in the places that it appears in the curriculum. Tutors should be aware of the importance of its illumination and elaboration, and be aware of potential student difficulties of the kinds demonstrated here. The illumination and elaboration may start already at the CS2 level, and surely at the Introduction-to-Algorithms level, with classical tasks like Bipartite matching and Hamiltonian cycles in dense graphs (Manber, 1989), as well as with less familiar tasks like those presented here.

References

Armoni, M., Gal-Ezer, J. (2007). Non-determinism: An abstract concept in computer science studies. *Computer Science Education*, 17, 243–262.

Armoni, M., Lewenstein, N., Ben-Ari, M. (2008). Teaching students to think nondeterministically. In: *SIGCSE Conference*. ACM Press, 4–8.

Cormen T.H., Leiserson, C.E., Rivest, R.L. (1991). *Introduction to Algorithms*. MIT Press, Massachusetts.

Dijkstra, E.W. (1976). *A Discipline of Programming*. Prentice-Hall.

Ginat, D. (2001). Loop invariants, exploration of regularities, and mathematical games. *Int. J. of Mathematical Education in Science and Technology*, 32, 635–651.

Ginat, D. (2003a). The greedy trap and learning from mistakes. In: *SIGCSE Conference*. ACM Press, 11–15.

Ginat, D. (2003b). The novice programmer's syndrome of design-by-keyword. In: *ITiCSE Conference*. ACM Press, 154–157.

Gries, D. (1981). *The Science of Programming*. Springer-Verlag.

Lynch, A.N. (1996). *Distributed Algorithms*. Morgan Kaufman.

Manber, U. (1989). *Introduction to Algorithms: A Creative Approach*. Addison-Wesley.

Schoenfeld, A. (1992). learning to think mathematically: problem solving, metacognition, and sense making in mathematics. In: Grouws, D.A. (Ed.), *Handbook of Research on Mathematics Teaching and Learning*. Macmillan, 334–370.

D. Ginat obtained his PhD in computer science from the University of Maryland, USA, while also collaborating with Robert Tarjan from Princeton University. His dissertation is in the domain of distributed algorithms and amortized analysis of algorithms. His main interest for more than a decade is computer science and mathematics education. He is currently the head of the Computer Science Group, in the Science Education Department of Tel-Aviv University.

Dėmesio nekreipimo koncepcija sprendžiant algoritminius uždavinius

David GINAT

Dėmesio nekreipimo sąvoka (anglų k. don't care) yra viena iš pagrindinių kompiuterių mokslo sąvokų. Ji nusako dėmesio nekreipimą į aplinkybes, kuriomis vyksta kai kurie scenarijai. Ši sąvoka – neapibrėžtumo pagrindas. Ji svarbi įvairiuose skaičiavimo modeliuose, ypač – paskirstytose ir lygiagrečiuose skaičiavimuose. Ji taip pat svarbi nuosekliuose bei deterministiniuose algoritmuose. Dėmesio nekreipimo koncepcija – vertinga priemonė algoritminiams uždaviniams spręsti. Tačiau mokant deterministinių algoritmų ši sąvoka nėra aiški ir dažnai lieka neištirta. Jos panaudojimo visiškas išaiškinimas ribotų mokinių išradingumą ir supratimą . Šis reiškinys atsiranda ir dėl gana neintuityvios "juodosios dėžės principo" savybių. Straipsnyje pateikiami ir aprašomi šeši šios algoritminės sąvokos vaizdingi pavyzdžiai ir iš jos išplaukiantys sunkumai.