Using Algorithmic Thinking to Design Algorithms: The Case of Critical Path Analysis

Algorithmic thinking is emerging as an important competence in mathematics education, yet research appears to be lagging this shift in curricular focus. The aim of this generative study is to examine how students use the cognitive skills of algorithmic thinking to design algorithms. Task-based interviews were conducted with four pairs of Year 12 students (n = 8) to analyze how they used decomposition and abstraction to specify the projects, designed algorithms to solve scheduling problems by first devising fundamental operations and then using algorithmic concepts to account for complex and special cases of the problems, and tested and debugged their algorithms. A deductive-inductive analytical process was used to classify students' responses according to the four cognitive skills to develop sets of subskills to describe how the students engaged these cognitive skills.

Keywords: algorithmic thinking; algorithm; critical path analysis; discrete mathematics; graph theory

Introduction

The integration of algorithmic thinking into mathematics curriculums at all levels is becoming increasingly important (Stephens, 2018) as students prepare for a society that is heavily reliant on digital technologies and complex systems (Wing, 2006). The shift towards integrating algorithmic thinking is in line with the recognition that *computational thinking*—which some argue encompasses algorithmic thinking among other key practices—is a fundamental competence that students should develop in order to meet the current and future demands of the STEM workforce (Weintrop et al., 2016; Wing, 2006). Efforts to integrate algorithmic thinking in school curriculums is well underway in many countries, including Australia, France, Japan, and the United Kingdom (Stephens, 2018). In Australia, recent revisions to the F–10 mathematics curriculum now explicitly include algorithmic thinking content across the domains of number (Years 3–5), algebra (Year 6), and space (Years 8–9) (Australian Curriculum, Assessment and Reporting Authority [ACARA], 2022).

Mathematics education researchers have long proposed that mathematics curriculums should include graph theory topics because they offer students many opportunities to engage in algorithmic thinking (Hart, 1998; Rosenstein, 2018). This proposition is reflected in the revised Australian mathematics curriculum, which requires Year 10 students to use graph theory to model real-world networks using vertex-edge graphs, and then design, test, and refine algorithms to solve problems related to these networks (ACARA, 2022). In Queensland (the state where this study was conducted), Year 12 students studying *General Mathematics*[1] (Queensland Curriculum & Assessment Authority [QCAA], 2019) are

[1]General Mathematics is a course for senior students whose intended future study or work does not require a knowledge of calculus and is based on the ACARA Senior Secondary curriculum. In addition to graph theory topics, it includes bivariate and time series analysis, growth and decay in sequences, Earth geometry and time zones, and loans, investments and annuities.

introduced to a range of graph theory topics, such as c*ritical path analysis*, and learn how to use standard algorithms to solve problems in an unplugged environment. However, the integration of algorithmic thinking into mathematics curriculums in this way is progressing at a time when there is a lack of research into ways that mathematics teachers might develop students' algorithmic thinking.

The literature about algorithmic thinking in relation to graph theory topics largely consists of proposed instructional tasks, or approaches, that invite students to design algorithms to solve common graph problems (Hart, 1998; Hart et al., 2008; Maurer & Ralston, 1991); however, empirical research into students' responses to such *algorithmatising tasks* is only just emerging (Moala, 2021; Moala et al., 2019; Tupouniua, 2020b). For example, Moala (2021) observed a mechanism that he called "accounting for features of a solution" (p. 264) to describe how a group of novice students developed the rules for a Hamiltonian path algorithm based on the numerical attributes of their solution. Similarly, Moala et al. (2019) introduced two practices—*localized considerations* and *patching*—to describe how novice students approached the revision and validation of the algorithms they designed to solve a series of friendship network optimization problems based on their initial solutions. They used the term localized considerations to refer to instances in which students adjusted the rules of an algorithm to fix an error that occurred at a particular vertex in a counterexample for which the initial algorithm was incorrect. Patching refers to the practice of retaining the overall structure of an algorithm but adding or removing rules in response to counterexamples. Accounting for features of a solution, localized considerations, and patching offer much needed insights into how students design algorithms in response to graph algorithmatizing tasks, but it remains unclear whether these mechanisms and practices apply beyond the tasks posed to the students in these studies or if they are used by students with more experience in graph theory.

The aim of the present study was to build on the emerging literature about the mechanisms and practices that students might use to design graph algorithms by examining how student-invented algorithms emerged across a sequence of graph algorithmatizing tasks of increasing complexity. To achieve this aim, I conducted an analysis of the algorithmic thinking of four pairs of Year 12 students as they developed viable algorithms to solve common scheduling problems in the context of critical path analysis (described further below). Critical path analysis was chosen as a graph theory topic in which to study students' algorithmic thinking because it involves multiple graph algorithms as well as opportunities for students to use algorithmic concepts including loops, branching statements, and variables. I found that the students devised a method for solving the scheduling problems in relation to the first problem, which I termed a *fundamental operation*, and then adjusted their fundamental operations using loops, branching statements, and variables in response to complex and special cases of the problems.

Review of Related Literature

One potential obstacle to this type of research is that the construct of algorithmic thinking is ambiguous because the term has been interpreted and applied inconsistently throughout the mathematics education literature (Lockwood et al., 2022). This ambiguity is further exacerbated by the use of the term throughout computer science education and computational thinking education literature, where the connection between computational and algorithmic thinking remains unresolved. Further research is needed to better understand the contemporary meaning of algorithmic thinking to promote consistency across empirical studies and support curriculum reforms. In this section, I first address definitions of algorithmic thinking and then examine the cognitive skills that comprise algorithmic thinking. Finally, I examine the literature related to graph theory in mathematics education.

Defining Algorithmic Thinking

Mathematics education researchers have occasionally used the term algorithmic thinking in reference to the construction of algorithms, although the term has not been defined consistently (Abramovich, 2015; Knuth, 1985; Petosa, 1985; Schwank, 1993). Mingus and Grassl (1998) proposed the following definition in the 1998 NCTM yearbook about the teaching and learning of algorithms in school mathematics:

> Algorithmic thinking is a method of thinking and guiding thought processes that uses step-by-step procedures, requires inputs and produces outputs, requires decisions about the quality and appropriateness of information coming in and information going out, and monitors the thought processes as a means of controlling and directing the thinking process. In essence, algorithmic thinking is simultaneously a method of thinking and a means for thinking about one's thinking. (p. 34)

More recently, Lockwood et al. (2016) interviewed five mathematicians about how they use computation and used their findings to formulate a working definition of algorithmic thinking: "A logical, organized way of thinking used to break down a complicated goal into a series of (ordered) steps using available tools" (p. 1591). Both definitions specify that algorithmic thinking is a way of thinking that results in a set of steps although do not describe this proposed way of thinking.

In contrast, Stephens (2018) proposes that algorithmic thinking is a form of mathematical reasoning comprised of a subset of computational thinking skills including, "decomposition (breaking a complex problem down into component sub-problems and sub-tasks), pattern recognition, generalization and abstraction" (p. 1). Stephens and Kadijevich (2020) also define algorithmic thinking as a form of mathematical reasoning that "is required whenever one has to comprehend, test, improve, or design an algorithm" (p. 2) and suggest that algorithmic thinking involves decomposition, abstraction, and algorithmization (or algorithm design). They exclude pattern recognition from their definition because they assume that it is an instance of abstraction and generalization. They also exclude generalization from their definition but do not explain why. The notion that algorithmic thinking is defined by a set of cognitive skills is consistent with definitions in computer science education literature. From this perspective, cognitive skills include analyzing and understanding problems, specifying a problem precisely so that it can be solved, formulating basic actions that will solve the problem, constructing an algorithm comprised of those basic actions, considering normal/special and simple/complex cases of the problem, and evaluating and improving the efficiency of the algorithm by considering alternative methods (Doleck et al., 2017; Futschek, 2006; Kanaki et al., 2020).

There are clearly diverse definitions of algorithmic thinking throughout education literature although the notion that a set of cognitive skills—drawn from models of computational thinking—is required to understand, construct, or evaluate algorithms is emerging as a common theme. There appears to be agreement, however, that algorithmic thinking refers to the creation of an algorithm independent of the coding language involved in programming a machine to process it, and this is the perspective adopted in this study. In the next section, I review each of the cognitive skills before describing how these abilities relate to the algorithmic thinking process.

Cognitive Skills

Decomposition

In mathematics education, heuristics such as *decomposing and recombining* (Pólya, 1945) and *establishing subgoals* (Schoenfeld, 1985) have long been taught to students to solve problems across domains of mathematics. These heuristics involve solving simpler problems that partially fulfil the conditions of the problem, and then recombining those solutions to solve the initial problem. Indeed, Hart et al. (2008) proposed a festival planning project for teaching students critical path analysis and envisage that students will first break down the project into many individual activities. Stephens and Kadijevich (2020) define decomposition in relation to algorithmic thinking in a similar way, “breaking a problem down into subproblems” (p. 117). Blannin and Symons (2019) provide an example of decomposition in a middle-years algorithmic thinking project about the busyness of playground areas in the school. The students broke down the playground into five locations, and then focused their analysis on each location and playground activity one at a time before combining their findings to determine which noisy activities disrupted the quiet areas.

From a computer science education perspective, decomposition appears to align with the ability to analyze a problem by splitting the main problem up into smaller problems (Futschek & Moschitz, 2010), such as identify the subtasks required to create and code a game in Scratch (Kwon & Cheon, 2019). However, from a computational thinking perspective, Shute et al. (2017) propose a broader definition:“[d]issect a complex problem/system into manageable parts. The divided parts are not random pieces, but functional elements that collectively comprise the whole system/problem.” (p. 153), which reflects Wing’s (2006) perspective that computational thinking is about both solving problems and designing systems. Examples of decomposition from a computational thinking perspective are the ability to decompose the modeling process used to build models of an ecological system in Year 6 science (Basu et al., 2017) and decomposing the research design process in undergraduate psychology into smaller issues such as the choice of research design, demographics of participants, and type of stimuli used in the experiment (Anderson, 2016). There appears to be consistency across the definitions and examples of decomposition through the education literature; however, there appears to be limited empirical research into how students use decomposition in algorithmic thinking contexts.

Abstraction

Stephens and Kadijevich (2020) define abstraction in relation to algorithmic thinking as “making general statements summarizing particular examples regarding underlying concepts, procedures, relationships, and models” (p. 118). In the computational thinking literature, abstraction refers to the ability to express a complex problem or system in terms of its essential elements. This ability involves distinguishing between relevant and irrelevant information or data (Wing, 2008), recognizing patterns and generalizing from particular instances (Wing, 2010), and building models that show how the problem or systems works by using the essential elements (Shute et al., 2017) using an appropriate representation to make the problem tractable (Wing, 2006). For example, Lee et al. (2011) describe how a group of middle school students used abstraction to represent the spread of disease throughout their school by identifying relevant factors, such as the layout of the school, the number of students, and the virulence of the disease, and then represented these factors using a virtual 3D model that showed the layout of the school and students. By using these abstractions, the students were able to simplify the complexity of the real-world scenario and focus on the key factors that were important in determining if a disease would spread throughout their school. This example highlights how the purpose of abstraction is to simplify a complex problem so that it can be solved (Grover & Pea, 2013; Standl, 2017), which aligns with the second skill in Futschek's (2006) model of algorithmic thinking from a computer science perspective: precisely specifying the problem. Despite the consistency in definitions throughout the literature, there are very few examples of how students use abstraction in algorithmic thinking.

Algorithmization

Algorithmization refers to the ability to construct a sequence of ordered steps for solving a problem (Maurer, 1992; Shute et al., 2017), or algorithm. To be effective, the rules specifying the algorithm must be precise and satisfy certain criteria (Maurer, 1992, 1998). First, the algorithm must be *determinate*, meaning that the first action is uniquely determined for each allowed input, and the next action is uniquely determined for each action in the sequence. Second, an algorithm must be *finite*, that is, it must stop after a finite set of actions for any allowed input. Finally, an algorithm must be *conclusive*, which means that when it terminates, it must either output a solution to the problem or indicate that it cannot solve the problem.

Algorithmization is the ability to write a series of steps that transform inputs into outputs, as outlined in the definition of algorithmic thinking by Mingus and Grassl (1998) above. These steps are comprised of basic actions devised to solve the given problem (Futschek, 2006), which may include algorithmic concepts such as branching (if/then/else conditions), iteration (loops or repeating steps), or the use of a variable to store intermediate results (Knuth, 1985; Modeste, 2016; Peel et al., 2019). For example, Peel et al. (2019) analyzed the handwritten algorithms constructed by secondary biology students to describe the natural selection process and found that the students used algorithmic concepts including branching, iteration, and variables. However, their study did not consider the process by which the students arrived at their sequence of steps as written, a gap that is common throughout literature about algorithmic thinking.

Debugging

Debugging is the ability to evaluate an algorithm, and find and fix errors in an algorithm to ensure that it solves the problem as intended (Shute et al., 2017). From a computer science education perspective, algorithmic thinking involves evaluating algorithms for correctness, particularly in relation to normal/special or simple/complex cases of the problem (Futschek, 2006; Kanaki et al., 2020) or efficiency (Futschek, 2006; Futschek & Moschitz, 2010). Evaluating algorithms may involve considering alternative actions or approaches, particularly when an algorithm is incorrect (Ginat, 2008). These skills are similar to those suggested by mathematics education researchers in relation to the design of algorithms. In the mathematics education literature, Maurer (1992) uses the term *algorithm verification* to describe the process of confirming or proving that an algorithm solves the problem, which can be achieved through mathematical induction. He also uses the term *algorithm analysis* to describe the process of evaluating the efficiency or complexity of an algorithm, which might also form part of algorithm evaluation. Again, there appears to be some consistency in the literature about the construct of debugging, but there are very few studies that examine how students use this ability.

Algorithmic Thinking Process

Researchers propose that the process of constructing an algorithm is similar to a problem-solving process, and that the cognitive skills outlined above are used at each stage of that process (Futschek & Moschitz, 2010; Mingus & Grassl, 1998; Ritter & Standl, 2023). In mathematics education, Mingus and Grassl (1998) compared the algorithmic thinking process to Pólya's (1945) four-stage problem-solving model (understand the problem; devise a plan, possibly an algorithm; carrying out the plan; looking back) and in computer science, Futschek and Moschitz (2010) proposed an iterative process involving five stages (analyze problem; find idea; formulate algorithm; play algorithm; reflect algorithm). From a computational thinking perspective, Ritter and Standl (2023) recently proposed a three-stage process that makes the links to the cognitive skills clear: "1. Describe, abstract and decompose the problem, 2. Design the algorithm, 3. Test the solution" (p. 4). I used these three stages to structure the comparison of the cognitive skills from each of the three perspectives discussed above, summarized in Table 1, to understand how they are used to construct an algorithm.

Table 1

Perspectives on Cognitive Skills Involved in Constructing an Algorithm

Algorithmic Thinking Process (Ritter & Standl, 2023)	**Mathematics Education**	**Computer Science Education**	**Computational Thinking**
Describe, abstract and decompose problem	• break down a problem into subproblems (Stephens & Kadijevich, 2020) • abstraction (Stephens & Kadijevich, 2020)	• analyze a problem by splitting it up into smaller problems • specify problem precisely	• decomposition: break down a problem/system into smaller parts • abstraction: determine the essential components of a system
Design algorithm	• break down a goal into a sequence of steps using available tools (Lockwood et al., 2016) • algorithmization (Stephens & Kadijevich, 2020)	• define basic actions that will solve the problem • develop a sequence of steps comprised of basic actions	• algorithms/algorithm design: create a sequence of steps to solve the problem
Test the solution	• confirm or prove that the algorithm solves the problem (Maurer, 1992) • compare algorithm with others that solve the same problem (Hart, 1998) • test and refine the algorithm using problems of the same type (Moala et al., 2019)	• improve or consider alternative actions (Doleck et al., 2017; Ginat, 2008) • optimize efficiency • consider normal and special cases; simple and complex (Kanaki et al., 2020)	• debugging: detect and fix errors • iteration: repeat algorithm design process

I drew three conclusions from this comparison of the cognitive skills from each perspective. First, decomposition and abstraction are complementary skills involved in understanding a problem, simplifying it, and specifying it precisely so that it can be solved using an algorithm. Second, algorithmization is the ability to design basic actions, which may include algorithmic concepts such as branching, iteration, and variables, and put these actions into a sequence that transforms inputs into the desired outputs. Finally, debugging not only refers to the ability to detect and fix errors in an algorithm to ensure that it solved the problem accurately, but also ensuring that the algorithm solves all problems of a particular type, improving the algorithm by considering alternative actions or entirely new approaches, or optimizing the efficiency of the algorithm. In the next section, I use this algorithmic thinking process to structure a review of the graph theory literature to illustrate its close relationship with algorithmic thinking.

Graph Theory in Mathematics Education

Researchers who advocate for the inclusion of graph theory in mathematics education describe a problem-solving process similar to the one described in Table 1 (Hart, 1998; Hart et al., 2008). Students should begin by understanding a real-world problem and then construct a vertex-edge graph to describe the relationships between the elements in the problem. Researchers provide examples to describe how they envisage students would represent graph problems with vertex edge graphs, including finding Eulerian circuits (Ferrarello & Mammana, 2018; Hart, 1998), minimal spanning trees (Hart, 1998), the travelling salesperson problem (Mingus & Grassl, 1998), optimizing traffic lights (Hart, 2008), and compatibility of radio station locations (Hart & Martin, 2018). Several researchers argue that constructing a vertex-edge graph is closely related to mathematical modelling (Greefrath et al., 2022; Hart, 2008; Medová et al., 2019). For example, Greefrath et al. (2022) showed how Year 9 mathematics students with no prior experience in graph theory were able to draw on their mathematical modeling competencies to successfully construct a vertex-edge graph to represent a road network and solve an Eulerian circuit problem using the graph. The authors argue that their findings highlight the reciprocity between developing students' knowledge of vertex-edge graph concepts and mathematical modeling competences. From a computational thinking perspective, Wetzel et al. (2020) conducted a similar case study in which they showed three novice 17-year-old students a road network and asked them to find the fastest route between two cities. The students were asked to simplify the map to solve the problem, but the researchers found that the students struggled to disregard irrelevant information from the map, with one student attempting to draw a vertex-edge graph in direct proportion to the source map. However, it appears that students in both these studies used their abstract representations of the road maps to solve the problems by *graph traversal*, which is the process of systematically examining and updating each vertex in a graph.

The second stage of solving graph theory problems is to use the vertex-edge graph to develop an algorithm to solve the problem (Hart, 1998). Again, researchers describe anecdotal evidence that students can create rudimentary algorithms to solve various graph problems including finding shortest paths (Gibson, 2012), optimal assignments (Hart & Martin, 2018), and Eulerian cycles (Ferrarello & Mammana, 2018), although there is little analysis of the steps that the students devised. In contrast, Moala (2021) analyzed the instructions in the algorithms written in the form a letter by two groups of students to find the optimal seating arrangement from a given graph. The analysis showed that the students first identified the optimal arrangement and then developed rules based on the contents of their solution. The students in the aforementioned study by Wetzel et al. (2020) found the fastest route between two towns using a brute-force approach, but realized that this was an inefficient approach and began looking for a more efficient, general algorithm. Studies that analyze students' approaches to writing basic actions or overall approaches to structuring an algorithm, such as a brute-force approach, provide long-overdue insights into algorithmization, and more research of this kind is needed.

Finally, Hart (1998) suggests that the final stage in solving graph theory problems is for students to check that their algorithm will always work, and compare their algorithms with other students' or standard algorithms. Researchers have begun to study how students revise their algorithms and try to adapt them to solve similar problems (Moala, 2021; Moala et al., 2019; Tupouniua, 2020a, 2020b). For instance, Moala et al. (2019) asked a group of pre-degree students to design an algorithm that could solve a friendship optimization problem represented by a vertex-edge graph, and then asked them to ensure that their algorithm could apply to two additional friendship graphs. The researchers found that the students retained certain parts of their algorithm that they thought were appropriate for some of the networks (local considerations) and added or removed instructions (patching) when the algorithm did not yield the correct solution. The students were initially able to solve the problem but were ultimately unsuccessful in constructing a correct algorithm. Nevertheless, the study provides insights into how novice students might approach debugging in the context of graph theory.

The existing literature about how mathematics students solve graph theory problems is largely speculative, involving mostly novice students in order to show how this topic would be easily accessible and engaging for students if it were included in school curriculums (Greefrath et al., 2022; Hart & Martin, 2018). Researchers also argue that graph theory provides opportunities for students to develop general mathematical processes, such as mathematical modelling and problem solving (Goldin, 2010; Greefrath et al., 2022), and, as in the case of the present study, algorithmic thinking. The aim of the present study, therefore, is to build on this foundational research by examining how more experienced Year 12 mathematics students, in a context where graph theory is included in the curriculum, engage algorithmic thinking as they are introduced to critical path analysis. Critical path analysis was used to examine students' algorithmic thinking in involves the design of multiple algorithms that include opportunities for using loops, branching, and variables. The following research question guided this study:

How do students use the cognitive skills of algorithmic thinking to develop algorithms?

Conceptual Framework

In this section, I outline a working conceptual framework for algorithmic thinking that I developed throughout this study. I began constructing this framework by assuming that algorithmic thinking is defined by a subset of cognitive skills as defined in the literature. There are conflicting perspectives about which cognitive skills constitute algorithmic thinking, so I turned to the algorithmic thinking process outlined in Table 1 and determined that algorithmic thinking involves decomposition, abstraction, algorithmization, and debugging. Finally, I developed elaborations for each of the cognitive skills based on my review of mathematics, computer science, and computational thinking education literature. The resultant conceptual framework for algorithmic thinking, contained in Table 2, is a working framework because I acknowledge that discussions about definitions are ongoing and unlikely to be resolved in a single paper such as this.

Table 2

Framework for Algorithmic Thinking

Cognitive Skills	**Elaboration**
Decomposition	Break down a problem or system into subproblems or smaller parts (Shute et al., 2017; Stephens & Kadijevich, 2020)
Abstraction	Determine the essential components of a problem or system, which involves: • collecting relevant information/data and disregarding irrelevant information/data (Shute et al., 2017; Wetzel et al., 2020) • building representations using the essential components that show how the problem or system works using an appropriate representation (Hart, 1998; Shute et al., 2017; Wing, 2006).

Algorithmization	Design a set of ordered steps to produce a solution or achieve a goal (Lockwood et al., 2016; Shute et al., 2017). Steps include algorithmic concepts such as: • inputs (Mingus & Grassl, 1998) • basic actions (Futschek, 2006; Moala, 2021) • outputs (Mingus & Grassl, 1998) • branching (if/then/else) (Peel et al., 2019) • iteration/loops (Peel et al., 2019) • variables/intermediate results (Mingus & Grassl, 1998; Peel et al., 2019).
Debugging	Test that the algorithm solves the problem or other problems of the same type, which involves: • detecting and fixing errors (Moala et al., 2019; Shute et al., 2017) • considering alternative approaches (Ginat, 2008; Moala et al., 2019) • improve efficiency of algorithm (Futschek & Moschitz, 2010; Hart, 1998).

As will become apparent in the sections that following, I found it necessary to develop a secondary construct to explain the ways that students used the cognitive skills, which I called *subskills*. By way of example, the accounting for features of the solution mechanism (Moala, 2021) outlined above would be a subskill of the basic actions elaboration of the algorithmization cognitive skill because it explains one way that students might develop instructions for an algorithm. Similarly, the practice of patching (Moala et al., 2019) would be a subskill of the detecting and fixing errors elaboration of the debugging cognitive skill because it explains one way that students might debug their algorithm. I used the construct of a subskill to operationalize the working framework to answer the research question.

Critical Path Analysis

Before describing the methodology, I will first provide a brief analysis of critical path analysis, which was the topic used in this study used to examine students' algorithmic thinking. Critical path analysis (or critical path method) is a widely-used project modeling tool developed during the 1950s to analyze and manage the scheduling of large engineering and military projects comprised of many interdependent activities (Kelley & Walker, 1959). The goal of critical path analysis is to build a model of a project and develop a schedule to ensure that a project is completed on time. Schedules typically include the following information:

(1) *Earliest Start Time (EST)*: the earliest time the activity must start without delaying the entire project.

(2) *Latest Finishing Time (LFT)*: the latest possible time the activity can finish without delaying the entire project.

(3) *Float time*: the duration that an activity can be delayed without delaying the completion of the project.

(4) *Critical path*: The critical path of a project is the path(s) from start to finish in the graph comprised of the sequence of activities that cannot be delayed without delaying the entire project. The *minimum time* required to complete the project is equal to the sum of the durations of the activities that comprise the critical path.

The purpose of critical path analysis for project managers is to identify the activities that cannot be delayed without delaying the entire project as well as activities with positive float time so that resources can be temporarily diverted from these activities to activities without float time, when necessary, to ensure the project is completed on time. I will use the preparation of the Waffle Breakfast, pictured in Figure 1, as an example project to explain the purpose of each step in critical path analysis in terms of the algorithmic thinking skills.

Figure 1

Waffle Breakfast Project and Precedence Table*

Activity	Description	Immediate Predecessor(s)	Duration (seconds)
A	Heat stove	–	5
B	Cook sausage	A	30
C	Cook waffle	A	20
D	Wash raspberries	C	5
E	Plate raspberries	D	3
F	Load cream	C	8
G	Plate waffle and sausages	B, C	10
H	Add ketchup and cream	F, G	20

*Project is shown as a picture of a meal made with playdough, which is discussed in the methodology section.

Decomposition

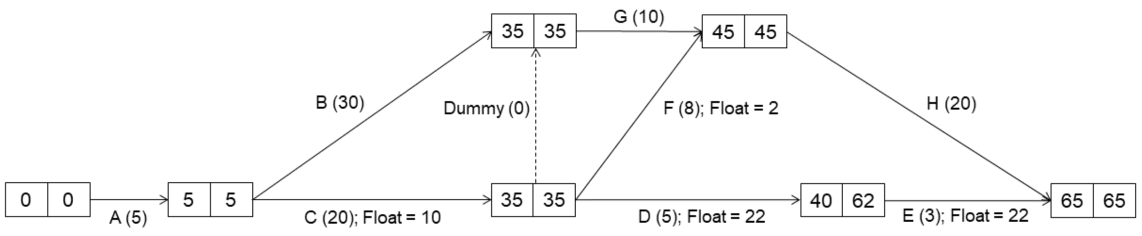
The first step in critical path analysis is to decompose the project into a list of all the activities required to complete the project, the duration of each of these activities, and the dependent relationships between the activities. This information can be summarized in a *precedence table*, as shown in Figure 1, which also shows that each activity has been assigned a letter. The *immediate predecessor(s)* column shows the activity or activities that must occur directly before an activity can start. For example, the plate waffle and sausages activity (G) can only commence after both cook sausage (B) and cook waffle (C) activities are completed.

Abstraction

The second step in critical path analysis is to construct a visual representation of the project using a vertex-edge graph, also referred to as a PERT (Program Evaluation and Review Technique) chart (Hart, 2008). Figure 2 shows such a graph for the Waffle Breakfast project. The first and last vertices represent the start and completion of the project respectively, and the vertices in between represent the end of an activity and the beginning of the next activity. The edges represent the duration of the activity in between the start and end of that activity. The purpose of constructing a vertex-edge graph is to make the precedence relationships between the activities more apparent than in the precedence table, particularly for activities that occur concurrently but have different durations.

Figure 2

Waffle Breakfast Vertex-Edge Graph



The vertices in Figure 2 are drawn as a box with two cells, which display the schedule for the project in terms of the EST and LFT for the activities. The EST for an activity is written in the left-hand cell and the LFT is written in the right-hand cell of the subsequent vertex. The durations of each activity are written on the edge that represents that activity, with an arrow facing towards the end of the project representing the passage of time. For example, the cook sausage activity (B) can only start after the heat stove activity (A) is finished, hence the EST for B is 5 seconds, which is written in the left-hand cell of the vertex at the start of Activity B. The duration of B is 30 seconds, so 30 seconds is added to the EST resulting in a LFT of 35 seconds, which is written in the right-hand cell of the vertex at the end of Activity B.

The cook waffle activity (C) also starts at 5 seconds and the duration of this activity is 20 seconds. However, as noted above, the plate waffle and sausage activity (G) can only start once both activities B and C are finished. Therefore, Activity C must also have a LFT of 35 seconds otherwise the schedule would indicate that G can start before activity B is finished. The earliest B can start is 5 seconds and the latest it can finish is 35 seconds, but its duration is 20 seconds, therefore it has a float time of 10 seconds. In other words, the cook waffle activity can be delayed up to 10 seconds without delaying the start of activity G. The vertex-edge graph makes these precedence relationships easier to follow than the precedence table (Kelley & Walker, 1959).

Activity C is an immediate predecessor for activities D and F in addition to activity G, however, activity B is not an immediate predecessor for these activities. To represent this arrangement, a dummy activity with a duration of 0 is drawn as a dotted edge, as shown in Figure 2, to separate activity B from activities D and F.

Algorithms

The third step in critical path analysis is to analyze the graph to plan a schedule for the project using three graph algorithms, which use the graph as inputs. *Forward scanning* is used to determine the EST for each activity and the minimum time required to complete the project. Forward scanning begins by assigning a 0 to the EST at the first vertex. The duration of the subsequent activity is added to the EST, which becomes the EST for the next activity. For example, the EST for A is 0, the EST for B is $0 + 5 = 5$, the EST for G is $5 + 30 = 35$. This fundamental operation is repeated from the start of the graph until the end of the graph, and therefore requires the user to traverse the graph activity-by-activity because the EST of one activity becomes an input for the subsequent activity. An exception is needed for parallel activities with different durations, such as activities B and C outlined above, and the highest EST is used in these instances. The minimum time required to complete the entire project is the value of the EST at the final vertex (65).

The second algorithm is *backwards scanning* and is used to determine the LFT for each activity. Backwards scanning begins by assigning the minimum time required to complete the project to the LFT at the last vertex because this is the latest finishing time for the entire project. The duration of each preceding activity is then subtracted from the LFT, which becomes the LFT for the preceding activity. For example, 65 is written in the LFT at the final vertex and then 20 and 3 are subtracted from 65 to determine the LFT for activities H and E respectively. This fundamental operation is repeated from the end of the graph back to the start, activity-by-activity, and each LFT becomes an input for the next iteration. The smallest difference between LFT and duration is used for vertices that are preceded by more than one activity, otherwise the preceding activity will be scheduled too late, and the project will be delayed. For example, the LFT for both B and C is 35; however, the LFT for A must be $35 - 30 = 5$ rather than $35 - 20 = 15$ because if A is scheduled any later than 5, it will delay the start of G and hence the whole project.

As outlined above, this gives rise to float time for activity C, which can be calculated as $LFT - EST - duration = 35 - 20 - 5 = 10$. An algorithm for float time is to repeat this calculation for each activity in the project.

Methodology

I used task-based interviews (Goldin, 2000) to investigate how students' use the algorithmic thinking cognitive skills in response to critical path analysis tasks, which is an appropriate methodology in generative studies such as this because my objective was to generate new observation categories for algorithmic thinking (Clements, 2000). I conducted the interviews with pairs of students so that they could discuss their thinking, which enabled me to make observations and draw inferences about their cognitive skills from both verbal and nonverbal behaviors (Goldin, 2000).

Participants

Four pairs of Grade 12 students (n = 8; mean age = 17.5) from a large, high-socioeconomic secondary school in Australia participated in the study. The students were studying General Mathematics (QCAA, 2019) and were in their final semester of school. Table 3 shows the names (pseudonyms), age, and grade at the end of Semester 1 for the students in each pair. The students and their parents gave ethical assent and consent to participate respectively.

Table 3

Demographic Information about Participants

Pair	Name (pseudonym)	Age at time of study	Grade at end of Semester 1
A	Archer	17	A
	Aubrey	18	B
B	Brooks	17	B
	Bradley	17	B
C	Carter	18	B
	Chandler	18	A
D	Dakota	18	A
	Dominique	17	B

I recruited these students because at the time of the study they had sufficient experience in the fundamentals of graph theory necessary to complete the critical path analysis tasks and thus enable me to focus on their algorithmic thinking. First, the students had recently completed an introductory unit on graphs and networks, which involved learning how to draw vertex-edge graphs and construct adjacency matrices. Second, they had experience in tasks that required them to develop algorithms for solving shortest path and minimum spanning tree problems, and then compare their algorithms with standard algorithms. Hence, the students were familiar with the sequential structure of graph algorithms written in plain English and the process of algorithmic thinking. Note that the students had not covered critical path analysis at the time of the study and this study formed their introduction to the topic.

Tasks

I designed a sequence of six algorithmatizing tasks that progressed from simple to complex, and included normal and special cases, the latter defined as a project with a dummy activity. Table 4 contains the sequence of tasks, complexity of each project, and the purpose of each task in terms of designing algorithms for EST, LFT, and float times. Appendices A–D contain the tasks sheets that I gave to the students during the interviews. I structured the sequence in this way because often students can solve graph problems with only a few vertices and edges easily, and then generalize their solution method for larger graphs (Goldin, 2010). This design feature enabled me to examine how students used algorithmic concepts to adjust the fundamental operations designed for simple/normal cases for complex/special cases of the same class of problem.

Table 4

Sequence of Tasks

Tasks	Complexity			Purpose
	Number of Paths	**Number of Activities**	**Normal/ Special**	
1. Hamburger Patty	1	3	normal	• decompose and abstract problem • make sense of EST and LFT
2. Hamburger Patty with Melted Cheese	2	5	normal	• account for parallel activities
3. Hamburger*	3	7	normal	• test and refine fundamental operations and
4. Parfait*	3	8	normal	• decompose and abstract more complex problem • apply fundamental operations for calculating EST, LFT, and float time
5. Waffle Breakfast	4	8	special	• account for dummy activity • design general algorithm for EST and LFT
6. Decontextualized	3	8	normal	• design general algorithm float time • debug general algorithms

*Number of paths or activities are indicative and may vary for Hamburger and Parfait projects depending on students' formulation.

The tasks for sessions 1 (tasks 1–3) and 2 (task 4) were designed so that students would be compelled to formulate the problem before solving the scheduling problems. Sessions 3 (task 5) and 4 (task 6) were intended to elicit students' algorithmization and debugging skills.

In designing the tasks, I had to account for the syllabus requirements, which required student to be able to "construct a network diagram to represent the durations and interdependencies of activities that must be completed during the project, e.g. preparing a meal" (QCAA, 2019, p. 40). Hence, all the tasks required students to construct a graph. I introduced the specific critical-path notation described above in the first session and the use of the precedence table in the third and fourth sessions. I designed the first three tasks in meal preparation contexts, as suggested by the syllabus. Aside from being a readily accessible context for these students—as opposed to contexts such as planning a festival or building a house (Hart et al., 2008)—the meal context enabled me to use playdough and toy equipment as proxies for real equipment that could be operated in real time. Students were able to physically quantify the durations of the activities and hence emulate the temporal differences between activities that give rise to the scheduling problems, which critical path analysis is intended to solve. The fourth task was presented without a context because I wanted students to begin thinking about critical path analysis as a general method that could be used beyond the food preparation context.

Data Collection

The interviews occurred over four 60- to 90-minute sessions that took place each week for four weeks out of school hours. The sessions coincided with the COVID-19 pandemic, which shaped the task environment. Government regulations required us to sit at least 1.5 metres away from each other and we achieved this by arranging the desks into a U-shape. The two students sat at the side and bottom of the U, and I sat on the opposite side.

At the start of interviews 1 and 2, I posed the projects to the students by providing a picture of the meal, the toy equipment, and/or precedence table (the first page of the task sheets in the appendices). The students formulated models of the projects, which included the activities, durations, and vertex-edge graphs, after which I posed the scheduling problems (the subsequent pages of the task sheets in the appendices). At the start of interview 3, I provided the preformulated model of the Waffle Breakfast in the form of a precedence table, which the students used to construct a graph, solve the scheduling problems, and then draft a general algorithm for finding EST and LFT. At the start of interview 4, I asked students to design algorithms for finding float times and the critical path. I then provided the students with a precedence table and incomplete graph, which the students used to test and refine the draft algorithms. At times, I prompted students to explain their reasoning about their nonverbal actions and made field notes about their responses as I attempted to follow their thinking. I also made suggestions to help overcome impasses when they arose (Goldin, 2000), and these prompts are written in italics on the task sheets in the appendices (but did not appear on the students' version). Students also had access to additional blank A4 paper. The students wrote their responses individually either on paper (which I copied after the sessions), on the computer at the front of the room, which they operated from a wireless keyboard and mouse or on a wall-mounted whiteboard. I recorded the interviews via Zoom with one camera positioned so that it captured the students and the wall-mounted whiteboard. Zoom also captured the virtual whiteboard that some students used to record their working, which I downloaded using the *Save Whiteboard* function. I transcribed the audio files generated from the Zoom recordings and paired these transcriptions with the associated student responses.

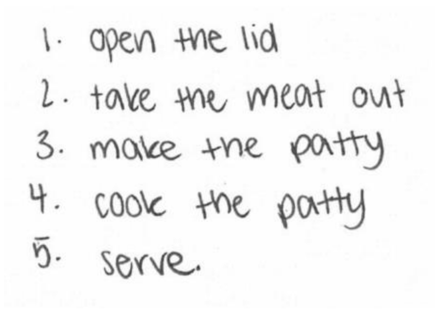
Analysis

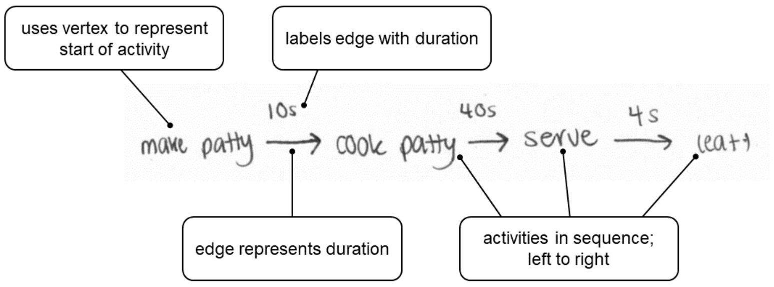
The data set included transcriptions of the audio recordings, video recordings, copies of student work, photographs of the wall-mounted whiteboard, and screenshots of the virtual whiteboard. I prepared the data for analysis by constructing a display of the transcriptions and associated student responses. I analyzed the data using a two-phase deductive-inductive approach. During the deductive phase, I read through the data set and coded transcriptions and student work according to the four cognitive skills. For example, I coded the first group of lines in Figure 3 as "decomposition" and the students' work as "abstraction". I began the subsequent inductive phase by grouping all instances relating to each of the four cognitive skills together. Each of these skills required slightly different methods of analysis, which I will discuss in turn.

Figure 3

Excerpts from Group A's First Interview

Transcript	Student Response	Memo

`00:04:42 Aubrey`
`Is it, like, the steps?`
 	
`00:04:45 Archer`
`Is it like take the Play-Doh [meat] out?`
 	
`00:04:49 Aubrey`
`Yeah, open the lid.`
 	
`00:04:52 Archer`
`Yup.`
 	
`00:04:55 Aubrey`
`Take the playdough out.`
 	
`00:04:56 Archer`
`And then put it in this thing [picks up toy mold]. Make a patty! [Makes patty using mold.]`
	 1. open the lid
	2. take the meat out
	3. make the patty
	4. cook the patty
	5. serve.
		Decomposition
		• decomposes project into discrete activities
		• uses toy equipment to emulate tasks
		• produces sequential list of activities

 uses vertex to represent start of activity labels edge with duration make patty 10s cook patty 40s serve 4s (eat) edge represents duration activities in sequence; left to right		**Abstraction**

Decomposition and Debugging. To analyze students' decomposition and debugging skills, I re-read each line of the dual display while watching the video footage and wrote my interpretations using memos (Corbin & Strauss, 2015). For example, the third column in Figure 3 shows my memo for the first decomposition episode of the Hamburger project, which reflects my interpretation of the students' physical, verbal, and written behaviors. I then grouped similar memos together to form what I called subskills (described above). The memo in Figure 3 was grouped with similar memos to form a subskill "analyze source material", which represented the students' ability to determine the activities by either looking at the picture or emulating the activities with the toy equipment.

Abstraction. To analyze abstraction skills, I read through the relevant episodes and wrote memos of my interpretation of the students' verbal reasoning about abstraction choices and how they traversed the graphs. I then turned to their written responses and analyzed the vertices, edges, and labels as shown in Figure 3. I grouped similar memos together to produce codes for abstraction subskills.

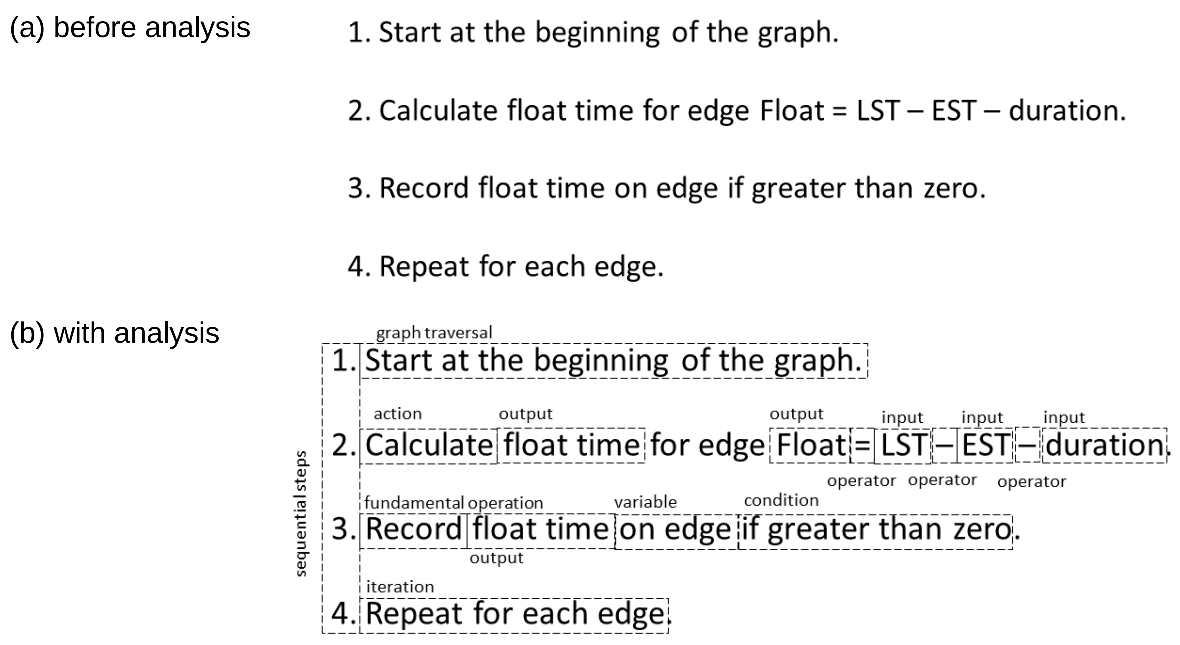
Algorithmization. I began my analysis of algorithmization by reading through the relevant episodes and coding the students' verbal and written behavior according to the elements of algorithms from Table 2. In the third and fourth sessions, the students wrote their algorithms in plain-English text, as illustrated by Group C's algorithm for float time in

Figure 4a. I analyzed the text by coding each word, phase, or clause according to the elements of algorithms. For example, I coded the word “calculate” in step 2 in Carter’s algorithm (

Figure 4b) as *action*; “float time” and “float” as *output*; “LFT”, “EST”, and “duration” as *input*; and the mathematical operators “=” and “−” as *operator*. After coding the transcripts and algorithms according to the elements, I re-read the transcripts and wrote memos to describe my interpretations of the students’ reasoning about these algorithmic elements and produced subskills by grouping similar memos together.

Figure 4

Analysis of Group C's Algorithm for Float Time


(a) before analysis

1. Start at the beginning of the graph.

2. Calculate float time for edge Float = LST – EST – duration.

3. Record float time on edge if greater than zero.

4. Repeat for each edge.

(b) with analysis

At each stage of the deductive and inductive analysis, an experienced mathematics educator reviewed my interpretations, and we resolved any conflicting interpretations through discussion. Finally, I used comparative analysis (Corbin & Strauss, 2015) until no new codes emerged.

Findings

Describe, Abstract, and Decompose Problem

The first phase of the algorithmic thinking process is to specify the problem so that it can be solved. The findings show that the students first decomposed the Hamburger and Parfait projects into discrete activities by analyzing the source material. This involved making observations from the picture to identify the components of the meal and handling the toy equipment to determine how the components are produced, as illustrated by the following excerpt from Group A's interview:

1	Archer:	Let's look at the picture and work this out…One. Make ice-cream. With this machine [points to toy ice-cream machine].
2	Aubrey:	Make macaroon, cut peach…with these? [picks up toy molds]

3	Archer:	…cut raspberries…
4	Aubrey:	You don’t need to cut raspberries! You just put them on top.
5	Archer:	Yeah, you do… [points to picture] …if you look, they are cut in halves…anyway we need a step.
6	Aubrey:	Okay.
7	Archer:	Five. Serve ice-cream. What then?
8	Aubrey:	Add fruit?
9	Archer:	[Points to picture] Place macaroon on plate.
10		How many workers are we going to have in this parfait shop?
11	Aubrey:	Two.
12	Archer:	Yeah, cause there’s two of us.
13	Aubrey:	One makes the ice-cream, one makes the…
14	Archer:	…macaroon!
15	Both:	[Laughter.]
16	Aubrey:	We have to split the macaroon one up.
17	Archer:	Oh, you’re joking!
18	Aubrey:	We need steps for that one.
19	Archer:	Oh, you’re joking! Okay let’s do it.

This excerpt shows how the students decomposed the project into discrete activities by analyzing the picture of the parfait to determine the components (ice-cream, macaron, peach, and raspberry pieces) and linking these components with the relevant toy equipment (lines 1–9). They also formulated a draft list of sequential activities based on their ongoing analysis of the parfait, as illustrated in (Lines 10–19 of Group A’s excerpt also illustrate how the students also recomposed activities into two separate paths. Archer and Aubrey decided that there were two “workers” involved in preparing the meal (lines 10–12) and that each worker would complete a separate component. They then decomposed the “make macron [sic]” activity further into activities H–K (see **Error! Not a valid bookmark self-reference.**a) and recomposed the relevant activities into paths for each worker: the “Strawberry Ice-cream” path ($A - B - C - H$) and the “Blueberry Macaroon” path ($D - E - F - G - H$).

Table 5 contains the subskills for abstraction that I identified across the sequence of tasks according to the two elaborations established in the existing literature (see Table 2), and I will address each of these elaborations in turn.

Figure 5a).

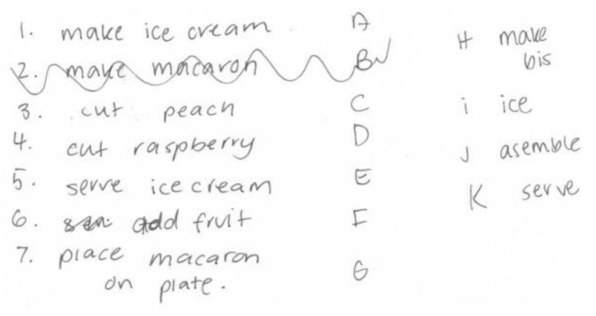
Lines 10–19 of Group A's excerpt also illustrate how the students also recomposed activities into two separate paths. Archer and Aubrey decided that there were two "workers" involved in preparing the meal (lines 10–12) and that each worker would complete a separate component. They then decomposed the "make macron [sic]" activity further into activities H–K (see **Error! Not a valid bookmark self-reference.**a) and recomposed the relevant activities into paths for each worker: the "Strawberry Ice-cream" path ($A - B - C - H$) and the "Blueberry Macaroon" path ($D - E - F - G - H$).

Table 5 contains the subskills for abstraction that I identified across the sequence of tasks according to the two elaborations established in the existing literature (see Table 2), and I will address each of these elaborations in turn.

Figure 5

Group A Parfait Project

(a)



1. make ice cream A
2. make macaron B
3. cut peach C
4. cut raspberry D
5. serve ice cream E
6. ~~[illegible]~~ add fruit F
7. place macaron on plate. G

H make bis
i ice
J asemble
K serve

(b)

Strawberry Ice-cream		Blueberry Macaroon	
A	pour ice cream (67)	D	make biscuit (18)
B	cut peaches (25)	E	bake biscuit (30)
C	cut raspberry (30)	F	make icing (14)
H	serve	G	assemble macaroon (18)
		H	serve

Table 5

Abstraction Subskills

Subskills	**Tasks**
Collecting relevant information/data and disregarding irrelevant information/data	
(i) identify and/or quantify variable	*Hamburger Patty* • identify time as relevant variable *Hamburger, Parfait* • quantify time using stop-watch function on cell phone
(ii) establish implicit criteria for relevance of information	*Hamburger, Parfait* • information effects time • information effects number of paths
Building representations using the essential components that shows how the problem or system works using an appropriate representation	
(iii) select an appropriate type of representation	*Hamburger Patty* • matrix or vertex-edge graph
(iv) determine how to represent essential components	*Hamburger, Parfait, Waffle Breakfast* • edges represent passing of time for an activity • vertices represent start and finish for an activity • draw edges from start of project (on left of page) to finish (on right of page) • arrows reflect order of activities • use precedent relationships to determine position of edges and parallel activities

The first subskill is the ability to *identify and/or quantify the relevant variable*. All groups identified time as the primary variable involved in planning the Hamburger Patty project. They quantified the duration of each activity by using the stopwatch function on their cell phones to time how long it took to complete the activities with the toy equipment.

The second subskill was the ability to *establish implicit criteria for relevance of information*. Each group appeared to use two implicit criteria to decide whether information about the project was relevant or not: time and paths. The groups appeared to use information that affected the time taken for activities or the number of paths required, and disregard information it if did not. For example, Group C regarded the quantity of ingredients (playdough) required to prepare parts of the meals as irrelevant because it would not affect the time required to complete the activity. Conversely, the number of workers assumed to be preparing the meal, as discussed in lines 10–12 by Group A above, was information regarded as relevant because it affected the number of paths.

The third subskill is the ability to *select an appropriate type of representation*. The groups all selected a vertex-edge graph, as opposed to a matrix, to represent their Hamburger Patty project, and appeared to use their knowledge about the properties of graphs to justify their selection. For example, Group B considered both types of representations in the following discussion:

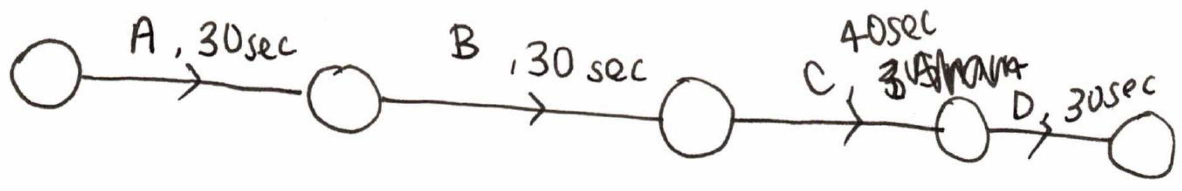
1	Brooks:	Okay, should we use a matrix or graph?
2	Bradley:	Not matrix.
3	Researcher:	What makes you say that Bradley?
4	Bradley:	Because we need to show, like, the steps.
5	Brooks:	Yeah, in a matrix.
6	Bradley:	No, like you can't show make patty comes before cook patty in a matrix.
7		That would only show that they are connected.
8		If you drew a graph, you can literally show the order.
9	Brooks:	Okay, jokes. You're right.

In line 5, Brooks asserted that a matrix was an appropriate representation, but Bradley argued that a graph was appropriate because it would show the unidirectional links between the activities (line 6, 8). She also argued that a matrix contains only the number of links between each activity (line 7), which suggests that knowledge about the properties of representations underpins the ability to select an appropriate representation.

Determine how to represent the essential components was found to be a distinct subskill for this elaboration of abstraction. I will use two examples of students' difficulties with these subskills to illustrate this finding. In the Hamburger Patty project, three of the four groups initially represented the activities as vertices and the edges as durations, as illustrated by Group A's graph in Figure 3. In contrast, Group B represented the activities and their durations as edges, and the start and finish of these activities as vertices (Figure 6). Note that I introduced the notational system described above to the students after they had decided to use a graph to represent the project Hamburger with Patty project.

Figure 6

Group B's Vertex-Edge Graphs for Hamburger Patty



Algorithm Design

Having specified the problem, the next stage in the algorithmic thinking process was to design algorithms for solving the scheduling problems. Table 6 contains a summary of the algorithmization subskills used by the students to design viable algorithms. This table shows that the fundamental operations for EFT and LFT devised for the Hamburger Patty project, and the fundamental operation for float time devised for the Hamburger Patty with Cheese project, were sustained throughout the remainder of the tasks, but adjusted using algorithmic concepts.

Figure 7 contains Group D’s final EST algorithm and Group C’s final LFT algorithm. I will refer to Table 6, Figure 4 (above), and

Figure 7 throughout this section to exemplify the findings.

Table 6

*Algorithmization Subskills**

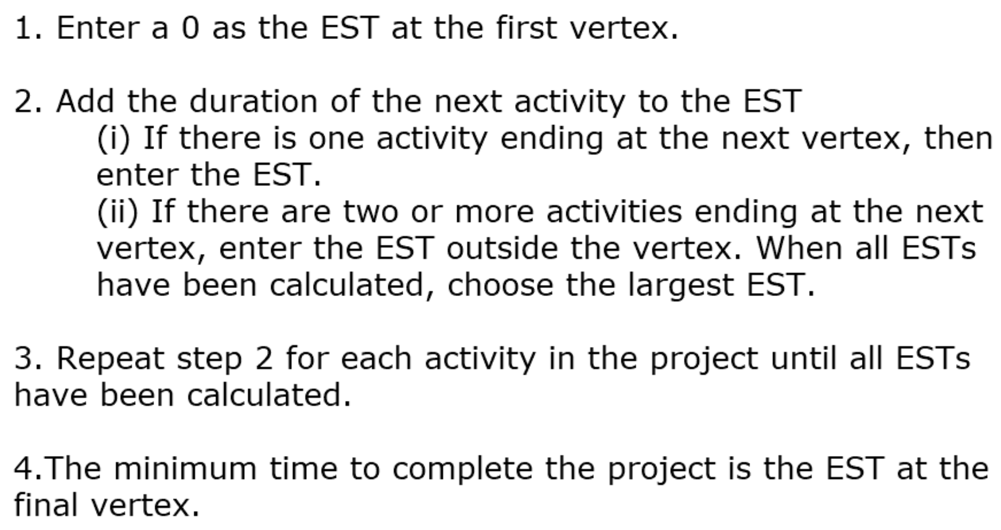
Subskills	**Earliest Start Time**	**Latest Start Time**	**Float Time**
(i) determine inputs	durations and previous EST *Hamburger Patty*	durations and subsequent LFT *Hamburger Patty*	EST, LFT, durations *Hamburger Patty with Cheese*
(ii) devise fundamental operation to solve problem	add duration to previous EST *Hamburger Patty*	subtract duration from subsequent LFT *Hamburger Patty*	$LFT - EST - duration$ *Hamburger Patty with Cheese*
(iii) determine output	EST; minimum time *Hamburger Patty*	LFT *Hamburger Patty*	float time *Hamburger Patty with Cheese*
(iv) use loop to repeat fundamental operation	repeat action for each activity until end of graph *Hamburger Patty*	repeat action for each activity until start of graph *Hamburger Patty*	repeat action for each activity in the graph *Hamburger Patty with Cheese*
(v) use branching to account for constraints of problem	if two or more edges finish at a vertex then select the highest EST *Hamburger Patty with Cheese*	if two or more edges finish at a vertex then select the lowest LFT *Hamburger Patty with Cheese*	–
(vi) use variable to store intermediate results	if two or more edges finish at a vertex, enter the ESTs outside the vertex *Waffle Breakfast Project*	If two or more edges finish at a vertex, enter the LFTs outside the vertex *Waffle Breakfast Project*	–

*Projects appearing in italics indicate when the concept emerged in relation to each algorithm.

Figure 7

Group D’s EST Algorithm and Group C’s LFT Algorithm

(a) Group D’s EST Algorithm



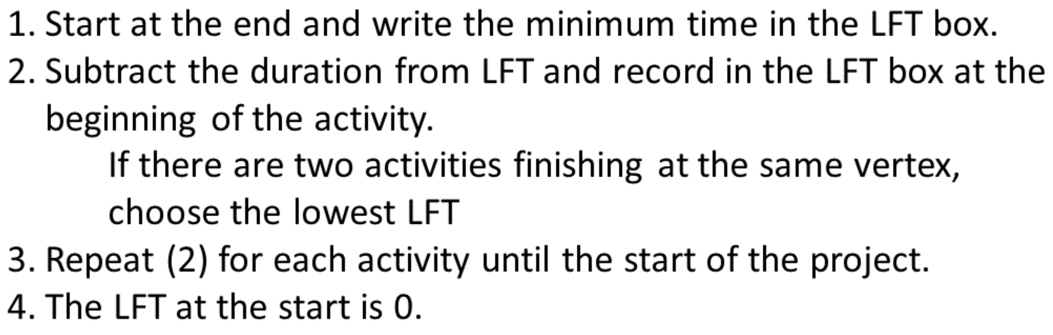
1. Enter a 0 as the EST at the first vertex.

2. Add the duration of the next activity to the EST
   (i) If there is one activity ending at the next vertex, then enter the EST.
   (ii) If there are two or more activities ending at the next vertex, enter the EST outside the vertex. When all ESTs have been calculated, choose the largest EST.

3. Repeat step 2 for each activity in the project until all ESTs have been calculated.

4.The minimum time to complete the project is the EST at the final vertex.

(b) Group C’s LFT Algorithm



1. Start at the end and write the minimum time in the LFT box.
2. Subtract the duration from LFT and record in the LFT box at the beginning of the activity.
   If there are two activities finishing at the same vertex, choose the lowest LFT
3. Repeat (2) for each activity until the start of the project.
4. The LFT at the start is 0.

Fundamental Operations

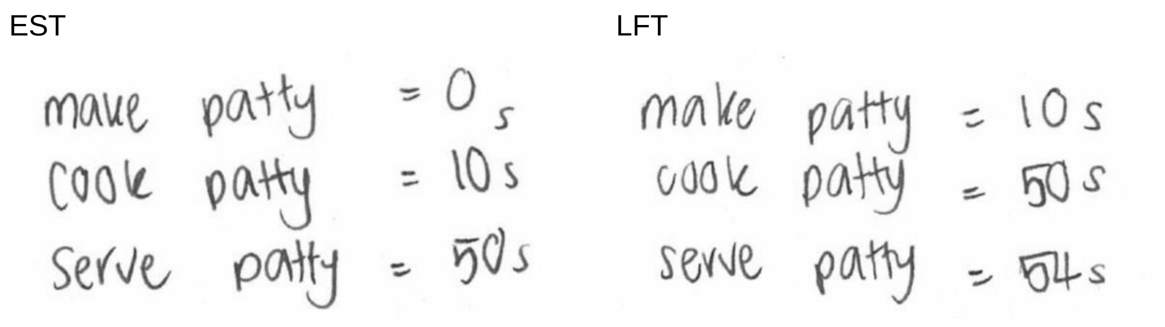
All groups devised fundamental operations for determining the EST and LFT to solve the Hamburger Patty problem. For example, Group A first devised the fundamental operation for determining EST by making sense of the sequential relationships between the activities, as illustrated by Archer’s written response in Figure 8 and the following explanation:

1	Researcher:	How did you get the earliest starting time?
2	Archer:	If you start at zero, then like, it takes 10 seconds to make the patty. And you need to finish making the patty before you grill it. So, you’re starting grilling at 10 seconds.
3	Researcher:	What about “serve patty”?
4	Archer:	Same thing. Like, you can’t serve the patty until you’ve grilled it. So, the earliest start time is 50 seconds.
5	Researcher:	How did you get 50 seconds?
6	Archer:	Well, its 10 seconds plus 40 seconds for grilling.

In line 2, Archer explained that the EST for the cook patty activity was 10 seconds because this activity could occur only after the patty has been made, which took 10 seconds. In line 4, Archer again used the sequential relationships between the activities to explain that the EST for the serve patty activity was 50 seconds because the durations of the preceding two activities added to 50.

Figure 8

Group A Output for EST and LFT Actions for Hamburger Patty Project



EST

mave patty = 0 s
cook patty = 10 s
Serve patty = 50 s

LFT

make patty = 10 s
cook patty = 50 s
serve patty = 54 s

Archer and Aubrey similarly devised the fundamental operation for the LFT by analyzing the precedent relationships between the activities of the project:

1	Archer:	We have to work backwards now.
2	Researcher:	What do you mean by that?
3	Archer:	Working back and then you have to…so you start…see how it finishes at 54 seconds? So, if you take off your four seconds, the cooking finishes at 50 seconds.
4		Then if you take off the 40 seconds for cooking…
5	Aubrey:	But that's how long it takes…
6	Archer:	Yeah, so that's when it finishes…after 40 seconds…
7	Aubrey:	Oh right, right. Because it takes 10 seconds to make the patty.
8	Archer:	Yeah, so it finishes at 50 seconds. And you take 40 seconds off that and make patty finishes at 10 seconds.
9		You just keep taking each one off! Is that it?
10	Researcher:	What do you mean?
11	Archer:	Will that always be the case?
12	Aubrey:	Yeah, like will you always be able to take the time off to get it?
13	Researcher:	What do you think?
14	Archer:	It is. I know it. Admit it! [Laughs.]
15	Researcher:	Okay, I admit it.

In lines 1 and 3, Archer suggested that the LFT could be determined by traversing the graph backwards from the minimum time and subtracting the duration of the serving time (4).

This fundamental operation yielded a LFT of 50 seconds, which Archer and Aubrey justified as the sum of the two remaining activities (lines 4–8).

The groups used the fundamental operations they devised for EST and LFT to solve the subsequent tasks and included them in their final written algorithms. For example, Group D used the fundamental operation for EST in the second step of their algorithm, “Add the duration of the next activity to the EST”. Similarly, Group C used the fundamental operation for LFT in the second step of their algorithm, “Subtract the duration from LFT and record in the LFT box at the beginning of the activity”. In each of these steps, the students used the duration and preceding EST or LFT as inputs that were transformed by the fundamental operations into outputs.

The student first devised a fundamental operation to calculate float time in response to the Hamburger Patty with Cheese project, as will be shown in the Branching section below. As for the fundamental operations for EST and LFT, the students sustained the use of this fundamental operation throughout the sequence of tasks and incorporated it into their final algorithms, as illustrated by Group’s C’s algorithm in

Figure 4.

Loop

The excepts from Group A used above to illustrate how the students justified their fundamental operations for EST and LFT also illustrate how the students recognized a pattern in how their fundamental operations could be repeated for subsequent activities in the project. For EST, Archer stated in line 4, “same thing” and then repeated the fundamental operation for the serve patty. Similarly for LFT, in line 9 Archer suggested that the fundamental operation of subtracting the duration from the previous LFT could be repeated to find the LFT for each activity in the project. The students also proposed that the pattern of repeatedly applying the LFT fundamental operation was a general step that would apply to other projects (10–12).

The other groups also recognized these patterns and expressed them in their final algorithms as a loop. For example, Step 3 of both Group D’s EST algorithm and Group C’s LFT algorithm (

Figure 7) contain an instruction to repeat Step 2 for each activity in the project. In each case, the loop also contains a condition for terminating the loop, that is, stop repeating Step 2 until all ESTs have been calculated (Group D) or when the start of the project is reached (Group C). This latter termination condition assumes that the user of the algorithm is traversing the graph from right to left and will apply Step 2 to parallel activities.

Branching

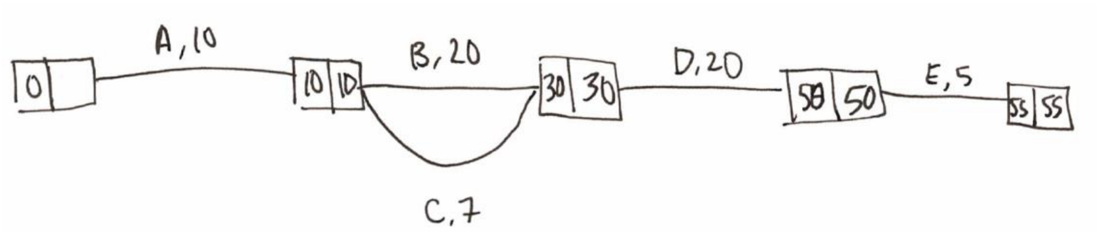
The Hamburger with Melted Cheese task elicited students' reasoning about parallel activities, which ultimately led to the use of branching in the students' final algorithms. Group B's graph for this task, reproduced in Figure 9, shows how they formulated the project such that the cook patty (B) and cut cheese (C) activities occurred at the same time, and were both immediate predecessors of the melt cheese on patty (D) activity. This formulation led the students to reason about the EST for activity D, as follows:

1	Brooks:	First one is zero.
2	Bradley:	First one is zero. Yeah.
3	Brooks:	And then the next one starts at 10.
4	Bradley:	When does cut cheese start?
5	Brooks:	It takes 7 seconds into the 20 seconds. It's how long it will take.
6		But it will start at the same time.
7	Researcher:	Start at the same time as what?
8	Brooks:	Starts the same time as cook patty.
9	Bradley:	Oh, so it's the same [start] time.
10	Brooks:	Yeah, they start at the same time.
11	Bradley:	When does melt cheese start?
12	Brooks:	Isn't it 30? Or not? Because you don't need to add the 7 to the 20.
13	Bradley:	It's 30.
14	Researcher:	Why do you say that?
15	Bradley:	Because 10 plus 20.
16	Brooks:	Oh, so it's still 30.

In line 6–10, Brooks concluded that their cut cheese activity would start at the same time as their cook patty activity and, as a result, the duration of the cut cheese activity was not included in their calculation of the EST for the melt cheese activity (line 12) because it was shorter (7) than the cook patty activity (20).

Figure 9

Group B's Hamburger Patty with Melted Cheese Graph



Similarly, this task also required students to reason about the LFT for parallel activities. For example, Group B compared the two possible LFTs as they traversed the graph from right to left:

1	Bradley:	30 minus 20 is 10.
2	Brooks:	Yeah, I know. What about the 7 seconds? Like 30 minus 7…[uses calculator]…is 23.
3	Bradley:	You don't need that one. Its bigger.
4	Researcher:	What do you mean by bigger Bradley?
5	Bradley:	Like if you finish making the patty at 23 seconds, it won't have enough time to cook.
6	Brooks:	Yeah, so you gotta choose the 10, the smaller one.

In lines 1 and 2, the students computed the two possible LFTs for their activity A. In lines 3–5, Bradley claimed that the 23 seconds LFT was not needed because it would delay activity D. In line 6, Brooks appears to devise a branching condition that the lower LFT should be selected when two (or more) activities begin at a vertex.

The reasoning about parallel activities illustrated by these two excerpts continued throughout the subsequent tasks, which all had parallel activities. The students then transformed this reasoning into branching statements in their final algorithms, as illustrated by the second steps in

Figure 7. Group D’s EST algorithm contains if/then conditions that instruct the user how to apply the fundamental operation to instances of either one activity, or two or more activities. Group C, on the other hand, included an if/then condition for only two activities finishing at a vertex as an exception to the initial instruction about how to apply the basic LFT action. This branching statement only accounts for two activities, and is thus an instance in which the algorithm would not work for all cases.

Variables

Three groups included a variable to store intermediate results for ESTs and LFTs at vertices that represented the beginning or end of multiple activities. Figure 10 is a screenshot of Group D’s response to the decontextualized project and shows how the students wrote the intermediate values of the ESTs for activities G (8 and 19) and H (24 and 25) outside the vertex. These intermediate values reflect Step (ii) of their EST algorithm (

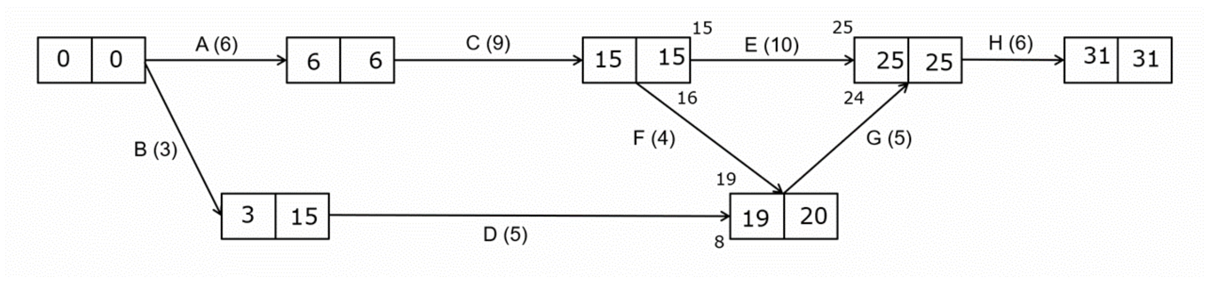
Figure 7), which forms part of the branching statement. Dakota explained this use of intermediate variables as follows:

1	Dakota:	You don't know what's going to be the EST for G. It could be D, it could be F. [Points to graph.] It depends on how you're doing it.
2	Researcher:	What do you mean by that?
3	Dakota:	Like, if you go along the top first [points to graph], you have to watch out for F because D also comes before G. So, it's like a way to keep track of what you're doing.

In line 1, Dakota recognized that the EST for G is dependent on the completion of activities D and F, but will not become apparent until both D and F are considered. She explained further in line 3 that writing the possible ESTs for G is needed because of how the graph is being traversed.

Figure 10

Group D's Decontextualized Project



Inputs

The inputs for each of the three algorithms were the durations, the precedence relationships between the activities, and the EST or LFT of the preceding or succeeding activity. Each group determined the durations and precedence relationships themselves in formulating the hamburger and parfait tasks but were provided with these inputs for the waffle breakfast and decontextualized projects.

The inputs used for the fundamental operation were the durations, ESTs and LFTs, but the selection of these inputs in the students' algorithms depended on the precedence relationships between the activities contained in the graph, which in turn depended on traversing the graph. The students' algorithms reflected this graph traversal implicitly. For example, Group D's EST algorithm (

Figure 7) instructs the user to start at the first vertex and then execute the fundamental operation on the “next activity”. The loop (step 3) then instructs the user to always choose the next activity in the graph “until all ESTs have been calculated”, which is somewhat ambiguous because it does not explicitly account for parallel activities. Similarly, Group C’s LFT algorithm (

Figure 7) is similarly ambiguous because they instruct the user to “start at the end” (step 1) and repeat the fundamental operation “until the start of project”.

Figure 11

Group C’s LFT Algorithm

(a) original algorithm

1. Start at the end and write the minimum time in the LFT box.
2. Subtract the duration from LFT.
3. Repeat (2) for each activity.
4. Choose the lowest LFT if there are two.
5. The LFT at the start is 0.

(b) revised algorithm

1. Start at the end and write the minimum time in the LFT box.
2. Subtract the duration from LFT and record in the LFT box at the beginning of the activity.
   If there are two activities finishing at the same vertex, choose the lowest LFT
3. Repeat (2) for each activity until the start of the project.
4. The LFT at the start is 0.

Test the Solution

Table 7 contains the two debugging subskills that the students used throughout the tasks. The first, *use alternative method to confirm solution*, was used to confirm the minimum time for the Hamburger and Parfait projects. Two groups used a brute-force approach by using decomposition subskill (ii) and abstraction subskill (v) to decompose the hamburger project into all possible paths and then compare the total times for each path, as illustrated in Figure 12.

Table 7

Decomposition Subskills

Subskills	**Tasks**
Considering alternative approaches	
(i) use alternative approach to confirm solution	*Hamburger, Parfait*

	• use brute-force or greedy approaches as alternative methods
Detecting and fixing errors	
(ii) execute algorithm and detect errors using algorithmic concepts	*Waffle Breakfast, Decontextualized* • ambiguous branching statement • non-terminating loop • step in wrong order

Figure 12

Group B Paths for Hamburger Project

make patty (10) + cook patty (20) + melt cheese (20) + assemble (5) + sauce (5) = 60

make patty (10) + cut cheese (7) + melt cheese (20) + assemble (5) + sauce (5) = 47

slice tomato (7) + melt cheese (20) + assemble (5) + sauce (5) = 37

The other two groups used a greedy approach, that is, they started at the beginning of the project and chose the edge with the longest duration at each vertex (abstraction subskill (v)). Dominique (Group D) explained her algorithm as follows:

1	Dominique:	Well, this is kind of like shortest path. But instead of using the shortest path, you use the longest one. You’re finding the longest path.
2	Researcher:	Can you show me what you mean?
3	Dominique:	So, you take the earliest starting time for cook patty and cut cheese, and then you look at the next two steps and choose the largest one. Because otherwise it will be too early to add the cheese.

In line 1, Dominique compared her algorithm with the greedy approach for finding the shortest path in a graph that the students learned previously, and then provided an example of her method for examining the nearest neighbor in line 3. Dominique then justified the selection of the longest duration in terms of the relationships between the ESTs of the two successive activities (line 3).

The ability to *execute algorithm and detect errors using algorithmic concepts* was the second subskill for debugging and, as shown in Table 7, there were three types of errors and revisions. An *ambiguous branching statement* refers to a branching statement that did not contain a condition. For example, Group C found that their branching statement in step 4 of their original algorithm (Figure 11a) would not be executed at step 2. To correct this error, they incorporated step 4 into step 2, as shown in Figure 11b. A *non-terminating loop* error refers to an iteration step that did not include a condition for when a user should stop repeating the instruction. Group C corrected this error by adding the condition, "until the start of the project" (Figure 11b). Finally, a *step in wrong order* error refers to a situation in which the students realized that their algorithm would not function as intended if the steps were carried out in the order given. For example, Group C found that step 4 in their draft algorithm (Figure 11a) was to be carried out after they moved onto the next activity. To correct this error, they incorporated the condition into step 2.

Discussion

The aim of the present study was to build on previous literature by examining how students use the cognitive skills of algorithmic thinking to develop algorithms. The context used to investigate the research question was critical path analysis because it provides many opportunities to observe students' algorithmic thinking. The most significant contribution to the literature about algorithmic thinking arising from this study relates to how the students designed viable algorithms in ways that differ substantially from prior research. In this section, I begin by discussing the findings related to algorithm design before addressing how the students used decomposition and abstraction to formulate the projects.

Algorithm Design

Previous research into the approaches used by novice students to design graph algorithms suggests that they tend to base their algorithms on their initial numerical solutions to the problems they are asked to solve (Moala, 2021; Moala et al., 2019). The findings of the present study build on this research by showing how the ability to devise a fundamental operation that is based on an initial solution, but expressed in more general terms, was a central element of the students' viable algorithms. The students used the fundamental operations that they devised to solve the initial problem to solve all the subsequent problems, and then included the fundamental operations in their final algorithms. These fundamental operations were comprised of basic mathematical operators (addition, subtraction) and were based on their reasoning about the goals of the problem. Standard algorithms for solving other graph algorithms also contain fundamental operations comprised of basic mathematical operators such as using addition to calculate the shortest distance between two locations (Dijkstra, 1959) or using an inequality $w_1 < w_2$ to select the next edge when finding a minimum spanning tree (Prim, 1957). Indeed, the use of basic mathematical operators to express fundamental operations can be observed in common algorithms beyond graph theory such as an inequality sign for the fundamental operation $n_1 > n_2$ in common algorithms for sorting sequences of numbers or division in the fundamental operation $m \div n$ in algorithms for finding the greatest common divisor of two integers, m and n (Thomas, 2020). Hence, the finding about the need to devise a fundamental operation that solves the problem is likely to have general applicability beyond the critical path analysis context in the present study and hence constitutes a significant contribution to the existing literature about algorithmic thinking.

The second major finding about students' algorithm design from the present study is their use of three algorithmic concepts. First, the students' reasoning about parallel activities led to the emergence of a condition for selecting an output for a particular activity based on their reasoning about the scheduling problems. These conditions were expressed in their final algorithms as branching statements. There is extremely limited research into students' use of branching statements, let alone the reasoning that underpins the construction of those statements. The findings of the present study, therefore, extend the existing literature by demonstrating that students' reasoning about multiple outputs resulting from applying the fundamental operation requires them to interpret those outputs in terms of the constraints of the problem. Second, the students' use of loops emerged when they recognized that the fundamental operation could be repeated for each activity in their project, that is, for each edge in the graph. Researchers suggest that pattern recognition is an algorithmic thinking cognitive skill, although there appears to be limited discussion in the literature about how this skill is used (Peel et al., 2019; Stephens & Kadijevich, 2020). Peel et al. (2019), however, suggest that the iteration of a sequence of steps in an algorithm is a pattern, the recognition of which is manifest in a loop. The students' use of loops in this study is an instantiation of this conjecture because the students first recognized the pattern of repeatedly applying the fundamental operation, and then constructed an if/then instruction based on their recognition of that pattern. This finding extends the existing literature by showing one way in which pattern recognition is used in algorithmic thinking, but further research into other ways that this skill can be used is needed.

Finally, the students in this study used a variable to keep track of intermediate results returned by the repeated application of the fundamental operation, specifically at vertices that represented the start or finish of multiple edges. The conditions contained in the branching statements mentioned above depended on these values either explicitly or implicitly. Researchers have long recognized the need for students to comprehend that a variable in an algorithm is used to assign an intermediate value that may change with subsequent iterations of an instruction, which is different from how a variable is used in algebra (Knuth, 1985; Modeste, 2016). The students in the present study had learned how to use a variable in this way when they learned other graph algorithms but recognized that they could use a variable in this new context when they needed to keep track of the multiple outputs.

The findings about students' use of fundamental operations, branching statements, loops, and variables are most likely a consequence of two key features of the tasks sequence. First, the initial tasks posed in the present study were comprised of only a small number of vertices and edges, and hence the students easily developed the fundamental operations and branching conditions that could be generalized to larger graphs (Goldin, 2010). Second, the complex and special cases of critical path analysis were introduced incrementally, which enabled the students to recognize the need for a condition when confronted with two or more outputs at a vertex in a situation deliberately manipulated to do achieve this aim. These two features of the task design differ substantially from existing studies (Moala, 2021; Moala et al., 2019), which require students to work with complex graphs initially. From a pedagogical perspective, such an incremental design may support novice students in developing these algorithmic concepts.

Describe, Abstract, and Decompose Problem

The students used a range of decomposition and abstraction subskills to formulate the projects so that the scheduling problems could be solved, and algorithms designed. The ability to analyze the source materials to break the projects down into discrete activities, as envisaged by Hart et al. (2008), identify and quantify time as the principle variable, and use implicit criteria to determine relevant information from the source material (cf Wetzel et al., 2020) led to the successful formulation of the projects. Choosing an appropriate representation, determining how to represent the activities using vertices and edges, and constructing representations were subskills necessary to represent the projects using vertex-edge graphs accurately. The use of decomposition and abstraction in these ways could just as easily be interpreted as competence in graph theory (Hart, 2008) or mathematical modeling (Greefrath et al., 2022; Medová et al., 2019), which reinforces the notion that algorithmic thinking is a collection of cognitive skills drawn from mathematics and computer science. However, the ability to select a vertex-edge graph over a matrix to representation should not be overlooked. The selection in the context of this study was not surprising, given that the tasks were part of a unit on networks; however, three groups considered using a matrix but ultimately chose a graph because they appeared to comprehend the underlying structure of the problem. This subskill is likely to have utility beyond the context of critical path analysis, and perhaps graph theory, which further research should address.

Test the Solution

Research into the debugging skills of students is also limited and the findings for this study suggest that students' ability to use algorithmic concepts was instrumental in the debugging of their algorithms. The students' competence in traversing a graph enabled them to recognize the need to formulate a branching statement and/or add a branching statement in response to detecting an error during the debugging session. Similarly, their competence in traversing a graph from start to finish enabled them to identify the opportunity to use iteration and/or recognize and fix a non-terminating loop during the debugging session. The students' approaches to debugging are consistent with the patching practice identified by Moala et al. (2019) in that the students added, removed, or otherwise adjusted steps but retained the structure of the algorithm. However, the students' recognition of errors in their use of algorithmic concepts is a subskill that enhanced the accuracy of their algorithms.

Limitations

I acknowledge that my claims about algorithmic thinking and the associated subskills are limited in at least four significant ways. First, the subskills are limited to critical path analysis problems posed within a food preparation context, although the students did appear to generalize their algorithms to solve the same scheduling problems for a decontextualized project. Second, the reproducibility of the responses to the projects used in this study by students in other cohorts will be limited because the fake meals and toy equipment were familiar to the students. However, similar, simple projects can be devised by adapting the design of these tasks to projects familiar to students in other contexts. Third, the study was conducted during the COVID-19 pandemic restrictions, which necessitated various adjustments to ensure the safety of the participants. Although I took reasonable steps to minimize the effect of these adjustments on the students' ability to think or produce written and verbal responses, the potential impact of the pandemic on the students should be considered when interpreting the task environment. Finally, the subskills documented in this study emerged out of an instructional sequence designed to support my operationalization of algorithmic thinking. Although I adapted an approach suggested by previous researchers (Hart et al., 2008), my approach to designing the tasks differs substantially from the approaches taken in more closely-related research (Moala, 2021; Moala et al., 2019). These differences in task design limit the comparability of this study, and therefore further research should focus on the difference in task design if this type of research is to support teachers as they begin to embed algorithmic thinking into mathematics education.

References

Abramovich, S. (2015). Mathematical problem posing as a link between algorithmic thinking and conceptual knowledge. *Teaching of Mathematics*, *18*(2), 45–60.

Anderson, N. D. (2016). A call for computational thinking in undergraduate psychology. *Psychology Learning & Teaching*, *15*(3), 226–234. https://doi.org/10.1177/1475725716659252

Australian Curriculum Assessment and Reporting Authority. (2022). *Australian Curriculum: Mathematics v.9*. ACARA. https://v9.australiancurriculum.edu.au/

Basu, S., Biswas, G., & Kinnebrew, J. S. (2017). Learner modeling for adaptive scaffolding in a computational thinking-based science learning environment. *User Modeling and User-Adapted Interaction*, *27*(1), 5–53. https://doi.org/10.1007/s11257-017-9187-0

Blannin, J., & Symons, D. (2019). Algorithmic thinking in primary schools. In A. Tatnall (Ed.), *Encyclopedia of education and information technologies* (pp. 1–8). Springer. https://doi.org/10.1007/978-3-319-60013-0_128-1

Clements, J. (2000). Analysis of clinical interviews: Foundations and model viability. In A. E. Kelly & R. A. Lesh (Eds.), *Handbook of research design in mathematics and science education* (pp. 547–589). Lawrence Erlbaum.

Corbin, J. M., & Strauss, A. L. (2015). *Basics of qualitative research: Techniques and procedures for developing grounded theory* (4th ed.). SAGE Publications.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*(1), 269–271. https://doi.org/10.1007/BF01386390

Doleck, T., Bazelais, P., Lemay, D. J., Saxena, A., & Basnet, R. B. (2017). Algorithmic thinking, cooperativity, creativity, critical thinking, and problem solving: exploring the relationship between computational thinking skills and academic performance. *Journal of Computers in Education*, *4*(4), 355–369. https://doi.org/10.1007/s40692-017-0090-9

Ferrarello, D., & Mammana, M. F. (2018). Graph theory in primary, middle, and high school. In E. W. Hart & J. Sandefur (Eds.), *Teaching and learning discrete mathematics worldwide: Curriculum and research* (pp. 183–200). Springer. https://doi.org/10.1007/978-3-319-70308-4

Futschek, G. (2006, November 7–11). *Algorithmic thinking: The key for understanding computer science* [Paper presentation]. 2nd International Conference on Informatics in Secondary Schools, Vilnius, Lithuania.

Futschek, G., & Moschitz, J. (2010, August 16–20). *Developing algorithmic thinking by inventing and playing algorithms* [Paper presentation]. Constructionism 2010 The 12th EuroLogo Conference, Paris, France.

Gibson, J. P. (2012, July 3–5). *Teaching graph algorithms to children of all ages* [Paper presentation]. 17th ACM Annual Conference on Innovation and Technology in Computer Science Education, Haifa, Israel.

Ginat, D. (2008, March 12–15). *Learning from wrong and creative algorithm design* [Paper presentation]. 39th ACM Technical Symposium on Computer Science Education, Portland, OR, United States.

Goldin, G. A. (2000). A scientific perspective on structured, task-based interviews in mathematics education research. In A. E. Kelly & R. A. Lesh (Eds.), *Handbook of research design in mathematics and science education* (pp. 517–545). Lawrence Erlbaum.

Goldin, G. A. (2010). Problem solving heuristics, affect, and discrete mathematics: A representational discussion. In B. Sriraman & L. D. English (Eds.), *Theories of mathematics education: Seeking new frontiers* (pp. 241–250). Springer. https://doi.org/10.1007/978-3-642-00742-2

Greefrath, G., Siller, H.-S., Vorhölter, K., & Kaiser, G. (2022). Mathematical modelling and discrete mathematics: opportunities for modern mathematics teaching. *ZDM*, *54*(4), 865–879. https://doi.org/10.1007/s11858-022-01339-5

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, *42*(1), 38–43.

Hart, E. W. (1998). Algorithmic problem solving in discrete mathematics. In L. Morrow (Ed.), *Teaching and learning algorithms in school mathematics* (pp. 251–267). National Council of Teachers of Mathematics.

Hart, E. W. (2008). Vertex-edge graphs: An essential topic in high school geometry. *The Mathematics Teacher*, *102*(3), 178–185.

Hart, E. W., DeBellis, V. A., Kenney, M. J., & Rosenstein, J. G. (2008). *Navigating through discrete mathematics in grades 6–12*. National Council of Teachers of Mathematics.

Hart, E. W., & Martin, W. G. (2018). Discrete mathematics is essential mathematics in a 21st Century school curriculum. In E. W. Hart & J. Sandefur (Eds.), *Teaching and learning discrete mathematics worldwide: Curriculum and research* (pp. 3–19). Springer. https://doi.org/10.1007/978-3-319-70308-4

Kanaki, K., Kalogiannakis, M., & Stamovlasis, D. (2020). Assessing algorithmic thinking skills in early childhood education: Evaluation in physical and natural science courses. In M. Kalogiannakis & S. Papadakis (Eds.), *Handbook of research on tools* *for teaching computational thinking in P-12 education* (pp. 103–138). IGI Global. https://doi.org/10.4018/978-1-7998-4576-8.ch005

Kelley, J. E., & Walker, M. R. (1959, December 1–3). *Critical-path planning and scheduling* [Paper presentation]. Eastern Joint Computer Conference, Boston, MA, United States. https://doi.org/10.1145/1460299.1460318

Knuth, D. E. (1985). Algorithmic thinking and mathematical thinking. *The American Mathematical Monthly, 92*(3), 170–181. https://doi.org/10.1080/00029890.1985.11971572

Kwon, K., & Cheon, J. (2019). Exploring problem decomposition and program development through block-based programs. *International Journal of Computer Science Education in Schools, 3*(1), 1–14.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., & Werner, L. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37.

Lockwood, E., Dejarnette, A., Thomas, M., & Mørken, K. (2022, February 2–5). *Exemplifying algorithmic thinking in mathematics education* [Paper presentation]. Twelfth Congress of the European Society for Research in Mathematics Education, Bozen-Bolzano, Italy.

Lockwood, E., DeJarnette, A. F., Asay, A., & Thomas, M. (2016, November 3–6). *Algorithmic thinking: An initial characterization of computational thinking in mathematics* [Paper presentation]. 38th Annual Meeting of the North American Chapter of the International Group for the Psychology of Mathematics Education, Tucson, AZ, United States.

Maurer, S. B. (1992). What are algorithms? What is algorithmics. In B. Cornu & A. Ralston (Eds.), *The influence of computers and informatics on mathematics and its teaching* (pp. 39–50). UNESCO.

Maurer, S. B. (1998). What is an algorithm? What is an answer? In L. J. Morrow & M. J. Kenney (Eds.), *The teaching and learning of algorithms in school mathematics: 1998 yearbook* (pp. 21–31). National Council of Teachers of Mathematics.

Maurer, S. B., & Ralston, A. (1991). Algorithms: You cannot do discrete mathematics without them. In M. J. Kenney (Ed.), *Discrete mathematics across the curriculum, K–12* (pp. 195–206). National Council of Teachers of Mathematics.

Medová, J., Páleníková, K., Rybanský, L., & Naštická, Z. (2019). Undergraduate students' solutions of modeling problems in algorithmic graph theory. *Mathematics, 7*(7), 1–16.

Mingus, T. T. Y., & Grassl, R. M. (1998). Algorithmic and recursive thinking: Current beliefs and their implications for the future. In L. J. Morrow & M. J. Kenney (Eds.), *The teaching and learning of algorithms in school mathematics: 1998 yearbook* (Vol. 1998, pp. 32–43). National Council of Teachers of Mathematics.

Moala, J. G. (2021). Creating algorithms by accounting for features of the solution: the case of pursuing maximum happiness. *Mathematics Education Research Journal*, *33*(2), 263–284. https://doi.org/10.1007/s13394-019-00288-9

Moala, J. G., Yoon, C., & Kontorovich, I. (2019). Localized considerations and patching: Accounting for persistent attributes of an algorithm on a contextualized graph theory task. *The Journal of Mathematical Behavior*, *55*, 100704. https://doi.org/10.1016/j.jmathb.2019.04.003

Modeste, S. (2016). Impact of informatics on mathematics and its teaching. In F. Gadducci & M. Tavosanis (Eds.), *History and philosophy of computing* (pp. 243–255). Springer.

Peel, A., Sadler, T. D., & Friedrichsen, P. (2019). Learning natural selection through computational thinking: Unplugged design of algorithmic explanations. *Journal of Research in Science Teaching*, *56*(7), 983–1007. https://doi.org/10.1002/tea.21545

Petosa, R. L. (1985). The benefits of computer programming in developing algorithmic thinking. *The Mathematics Teacher*, *78*(2), 128–130.

Pólya, G. (1945). *How to solve it: A new aspect of mathematical method*. Princeton University Press.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal*, *36*(6), 1389–1401.

Queensland Curriculum and Assessment Authority. (2019). *General Mathematics 2019 v.1.2*. Queensland Curriculum and Assessment Authority. https://www.qcaa.qld.edu.au/downloads/senior-qce/syllabuses/snr_maths_general_19_syll.pdf

Ritter, F., & Standl, B. (2023). Promoting student competencies in informatics education by combining semantic waves and algorithmic thinking. *Informatics in Education*, *22*(1), 141–160. https://doi.org/10.15388/infedu.2023.07

Rosenstein, J. G. (2018). The absence of discrete mathematics in primary and secondary education in the United States...and why that is counterproductive. In E. W. Hart & J. Sandefur (Eds.), *Teaching and learning discrete mathematics worldwide: Curriculum and research* (pp. 21–40). Springer. https://doi.org/10.1007/978-3-319-70308-4

Schoenfeld, A. H. (1985). *Mathematical problem solving*. Academic Press.

Schwank, I. (1993). On the analysis of cognitive structures in algorithmic thinking. *Journal of Mathematical Behavior*, *12*(2), 209–231.

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142–158.

Standl, B. (2017). Solving everyday challenges in a computational way of thinking. In V. Dagienė & A. Hellas (Eds.), *Informatics in schools: Focus on learning programming* (pp. 180–191). Springer.

Stephens, M. (2018). Embedding algorithmic thinking more clearly in the mathematics curriculum. In Y. Shimizu & R. Vithal (Eds.), *ICME 24 School mathematics curriculum reforms: challenges, changes and opportunities* (pp. 483–490). International Commission on Mathematical Instruction.

Stephens, M., & Kadijevich, D. M. (2020). Computational/algorithmic thinking. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (pp. 117–123). Springer. https://doi.org/10.1007/978-3-030-15789-0_100044

Thomas, M. O. J. (2020). Algorithms. In S. Lerman (Ed.), *Encyclopedia of mathematics education* (2nd ed., pp. 48–50). Springer. https://doi.org/10.1007/978-3-030-15789-0_8

Tupouniua, J. G. (2020a). Explicating how students revise their algorithms in response to counterexamples: building on small nuanced gains. *International Journal of Mathematical Education in Science and Technology*, *53*(7), 1711–1732. https://doi.org/10.1080/0020739X.2020.1837402

Tupouniua, J. G. (2020b). Finding correct solution(s) $\nRightarrow$ creating correct algorithm(s): Shedding more light on how and why. *The Mathematician Educator*, *1*(2), 102–121.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining computational thinking for mathematics and science classrooms. *Journal of Science Education and Technology*, *25*(1), 127–147. https://doi.org/http://dx.doi.org/10.1007/s10956-015-9581-5

Wetzel, S., Milicic, G., & Ludwig, M. (2020, July 6–7). *Gifted students' use of computational thinking skills approaching a graph problem: A case study* [Paper presentation]. 12th International Conference on Education and New Learning Technologies, Palma de Mallorca, Spain.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical transactions of the Royal Society of London. Series A: Mathematical, physical, and engineering sciences*, *366*(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Wing, J. M. (2010). *Computational thinking: What and why?* Computer Science Department, Carnegie Mellon University.

Appendix A

Hamburger Patty Project

Create a model that represents the preparation of this hamburger patty:

1. What is the minimum time required to prepare the meal?
2. Determine the earliest starting time for each activity.
3. Determine the latest finishing time for each activity.
4. Determine if any of the tasks can be delayed or extended without increasing the minimum project completion time.

Hamburger Patty with Melted Cheese Project

Create a model that represents the preparation of this hamburger patty with cheese:

[*Prompt: Introduce notational system.*]

1. What is the minimum time required to prepare this meal?
2. Determine the earliest starting time for each activity.
3. Determine the latest finishing time for each activity.
4. Determine if any of the tasks can be delayed or extended without increasing the minimum project completion time.

Hamburger Project

Create a model that represents the preparation of this hamburger:

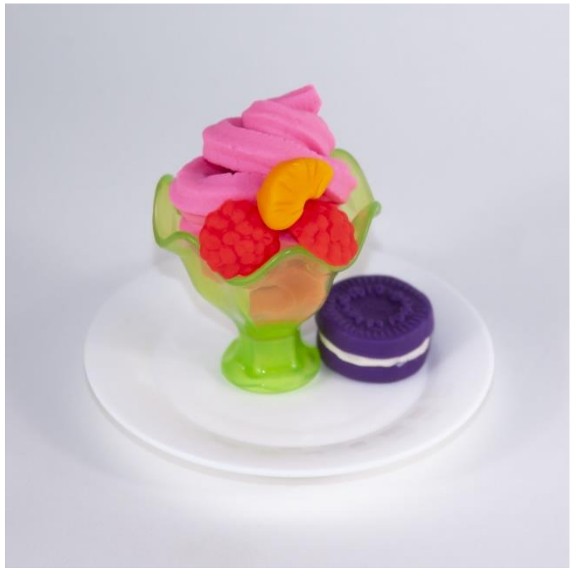
1. What is the minimum time required to prepare this meal?
2. Determine the earliest starting time for each activity.
3. Determine the latest finishing time for each activity.
4. Determine if any of the tasks can be delayed or extended without increasing the minimum project completion time.

[*Prompt: Introduce definition of critical path.*]

Appendix B

Parfait Project

Construct a model that represents the preparation of this strawberry parfait with blueberry macaroon.



Determine the following:

(a) earliest starting time for each activity

(b) latest finishing time for each activity

(c) any float times

(d) critical path and minimum time required to prepare the dessert.

Appendix C

Waffle Breakfast Project

1. The table below contains the immediate predecessors and durations for the activities required to prepare the waffle with raspberries and cream, and sausage with ketchup. Use the information contained in this table, the picture of the waffle breakfast, and the Play-Doh equipment to formulate a vertex-edge graph for this project. [*Prompt: Can you decompose the project into separate paths?*]

Activity	Description	Immediate Predecessors	Duration (seconds)
A	Heat stove	–	5
B	Cook sausage	A	30
C	Cook waffle	A	20
D	Wash raspberries	C	5
E	Plate raspberries	D	3
F	Load cream	C	8
G	Plate waffle and sausages	B, C	10
H	Add ketchup and cream	F, G	20

2. Determine the following:

(a) EST, LFT, float time for each activity

(b) critical path and minimum time required to prepare the meal.

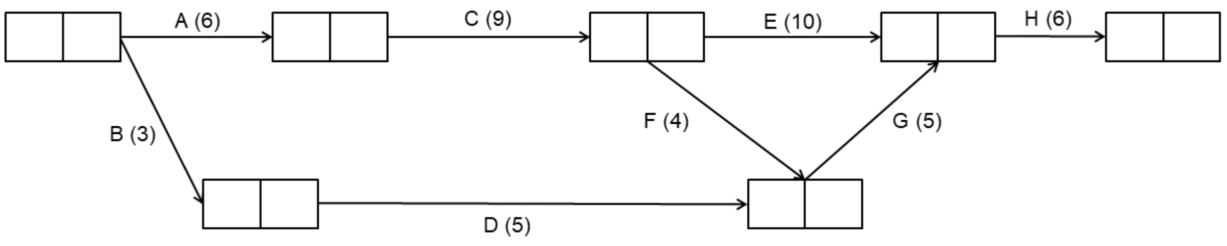
3. Draft a general algorithm for finding EST and LFT.

Appendix D

Decontextualized Project

Design algorithms for finding float times of all activities in a project and the critical path for a project.

Use your draft algorithms to complete a critical path analysis of this project. Explain any revisions you make to your algorithms.



Activity	Immediate Predecessors	Duration
A	–	6
B	–	3
C	A	9
D	B	5
E	C	10
F	C	4
G	D, F	5
H	E, G	6

[*Prompt: Did that step really explain the step the way you performed it?*]