**Physics Solver: From Cognition to Education**

**Demetrios Kriezis, Athanasios Taprantzis**

Hellenic American Educational Foundation-Athens College

dimitris.kriezis@outlook.com, thatapras@hotmail.gr

**Andreas Karampelas**

Information and Communication Technology Teacher, Hellenic American Educational Foundation

akar@haef.gr

# 

# **Abstract**

Solving problems is one of the core abilities of the human brain. Based on a set of data the brain computes a solution and stores the path followed in the process for future use. The goals of this research project are to: develop a program that follows a similar process to solve Physics problems and to present the solution in such a way so that students can focus more on the ideas, rather than the underlying mathematical calculations. First, the input module of the program receives the statement of a physics problem in a textual, image, or voice form and processes it separating the relevant data into tokens. Then the solver module searches through a set of formulas stored in a database depending on the given data and the requested calculation. It performs an in-depth search, combining different formulas in order to yield the result. If none of the formulas stored in the database are compatible, the system can derive new formulas by transforming the existing ones. Finally, the system outputs the solution as well as the steps followed. By focusing on the ideas in physics rather than the math, this project aims to help the efforts made by educational communities to support every student and teacher in the learning process. Especially those who face major problems like lack of teachers that can provide personalized guidance and support or lack of resources like tools, books etc. Using different input/ output methods this program can also be a valuable learning companion for students with special needs.

# 

# **Keywords**

*Machine Learning, Artificial Intelligence, Graph, Physics*

# 

# **Introduction**

In the last decade there has been a boost of scientific research in Artificial Intelligence resulting to a plethora of applications that followed the specific evolution, tackling problems that were considered hard to solve in terms of reliability and precision. Machine learning is no longer considered a chase of chimeras, giving birth to Narrow Intelligence (also known as Weak AI) empowering voice recognition, computer vision, object detection and manipulation, semantic analysis etc. The impact is huge and it concerns every aspect of our daily lives while also being the main characteristic of the fourth industrial revolution. From medicine to biology and chemistry, to economy and businesses, to market and trends, to social life, education and communication everything is radically changing. Education faces new challenges, massive open online courses (MOOCs), online tutoring, personalized learning, technology enhanced learning (TEL) are some of the new trends. New synthesis and ideas are emerging giving birth to creative, innovative solutions.

There are many applications in the education sector that take advantage of recent achievements in AI. Two known examples are mathway and photomath. These two applications can solve algebraic equations using textual or image input. This work is inspired by software tools like the ones mentioned above since it can be used as a learning companion in the context of physics. To the best of our knowledge there is no such service or tool provided like a physics solver, i.e. a software that produces automatically or semi-automatically solution to physics problems. At face-value the problem we attempt to overcome may seem excessively difficult with little to no reward for any effort made transforming it to a computer related problem. When broken down, though, it can be easily seen that the physics equations relate very much to each other in an interesting way worth taking more than a look at. Looking at how AI and connected applications have made a significant impact on today’s understanding of how and why technology should be used, we saw an opportunity to create a first of a kind learning companion, backed by the tools machine learning is able to provide today. The motivation for this work is to support the efforts made by learning communities to help every student and teacher in the learning process. Especially those who face major problems like lack of teachers who can provide personalized guidance and support, lack of resources like tools, books etc, and students with special needs, by delivering high quality software that will serve as a learning companion. In other words we believe this software can contribute in filling the gap between rich and poor learning communities. Of course there is a counter argument that this could lead to a more passive learner who waits for the solution to come. We are open to critics but being able to have access to a user-friendly, fast, reliable, and personal tutor for physics problems is a inconceivable concept that has great potential in the educational field. Thus, the curiosity that intrigued us to further explore this project as well as the amazing potential of similar works in other educational sectors were the driving forces that motivated us to create this paper in the first place.

This work can produce significant impact on the area of physics education, but it is also open to adaptations in other contexts and to generalization. Until now, it is able to provide help and step-by-step solutions to simple problems. It can be easily used by young students to get help with their homework and get help whenever they feel frustrated when struggling to find a solution. However, the ease of expansion of its current capabilities makes it an exceptional piece of software with great scalability. The more contributors the better the range, efficiency, and quality of the responses it is going to provide. With enough users and time spent on improving it, the physics solution designer can with minimum extra effort expand from a utility being able to solve kinematics-specific problems to almost anything - from harmonic motions to circuits.

This project is also in itself a machine learning application since it attempts to capture the student’s reasoning in solving problems. Using a set of pre-existing formulas, the program solves a new problem and then stores the solution steps. This way, module mimics the students’ capacity to learn a problem-solving method after being faced with a new problem.

The main contributions of our work are the following:

* we present a model - graph for representing knowledge in physics which is generic
* based on the above model we have developed and implemented algorithms and data structures for the exploitation of such a model
* we have combined different input - output methods in order to support differentiation in needs and learning styles by taking advantage of the evolution of Human Computer Interfaces that tends to become more human friendly and physical, leaving raw computational infrastructure

The rest of the paper is organized as follows: In the next section we present the background knowledge and the related work required to be able to follow the rest of the paper. In the section “Overview” we give a brief Overview of all the system. In the section “Description of the system” we give a detailed description of all the modules as well as pseudocode and graph in order to be concise. In the section “Experimental results” we execute different scenarios and test cases which show the functionality of our system. In the section “Discussion” we conclude the paper talking about the cases where our system performs well and others that it performs poorly, as well as future work i.e. work that is needed so that the system delivers better results. In the last section we have included indicative parts of the code and snapshots of the database.

## **Related work & Background knowledge**

The applications that were the most influential to us were mostly algebra-related. One of the best such examples is Photomath with its capability to record real-time camera footage and solve the captured equations with explanatory solutions. With its simple UI yet complex backend processes, Photomath has been a motivational force of us since the beginning. Wolfram Alpha, Mathway, and Socratic were also an inspiration to us. The core module of our system is based on modelling the data of the problem as an unweighted directed acyclic graph where we can perform a depth first search. We have also taken advantage of reliable technologies developed from Google to recognize voice and image. Using Google’s Speech API we are able to recognize voice and translate it to text and, thus, insert a problem’s statement into our software through speech. Google’s Tesseract OCR engine is another technology that we also use to extract text through visual forms of input such as images or live camera feed.

The following definitions (Cormen et al., 2009) are essential for understanding the rest of the paper.

### Definition 1 Directed Graph

*“A directed graph (or digraph) G is a pair (V, E), where V is a finite set and E is a binary relation on V . The set V is called the vertex set of G, and its elements are called vertices (singular: vertex). The set E is called the edge set of G, and its elements are called edges”.*

### Definition 2 Acyclic Graph

*“In a directed graph, a path (V0, V1, V2, …, Vk) forms a cycle if V0 = Vk and the path contains at least one edge …. A graph with no cycles is acyclic”.*

### Definition 3 Unweighted Graph

### An unweighted graph is one whose edges have length equal to one unit. Thus, every edge has the same value. No edge has more significance or “weight” than the others.

### Depth First Search Algorithm

The Depth First Search (DFS) algorithm can find a possible path between two vertices in an unweighted graph. To better understand how this approach works here is the pseudocode simplified:

*DFS (v, end)*

*Mark v as visited*

*for u in set\_v:*

*if u not visited:*

*if u equals end:*

*return u*

*else:*

*return: run DFS (u, end)*

*return none*

Notes:

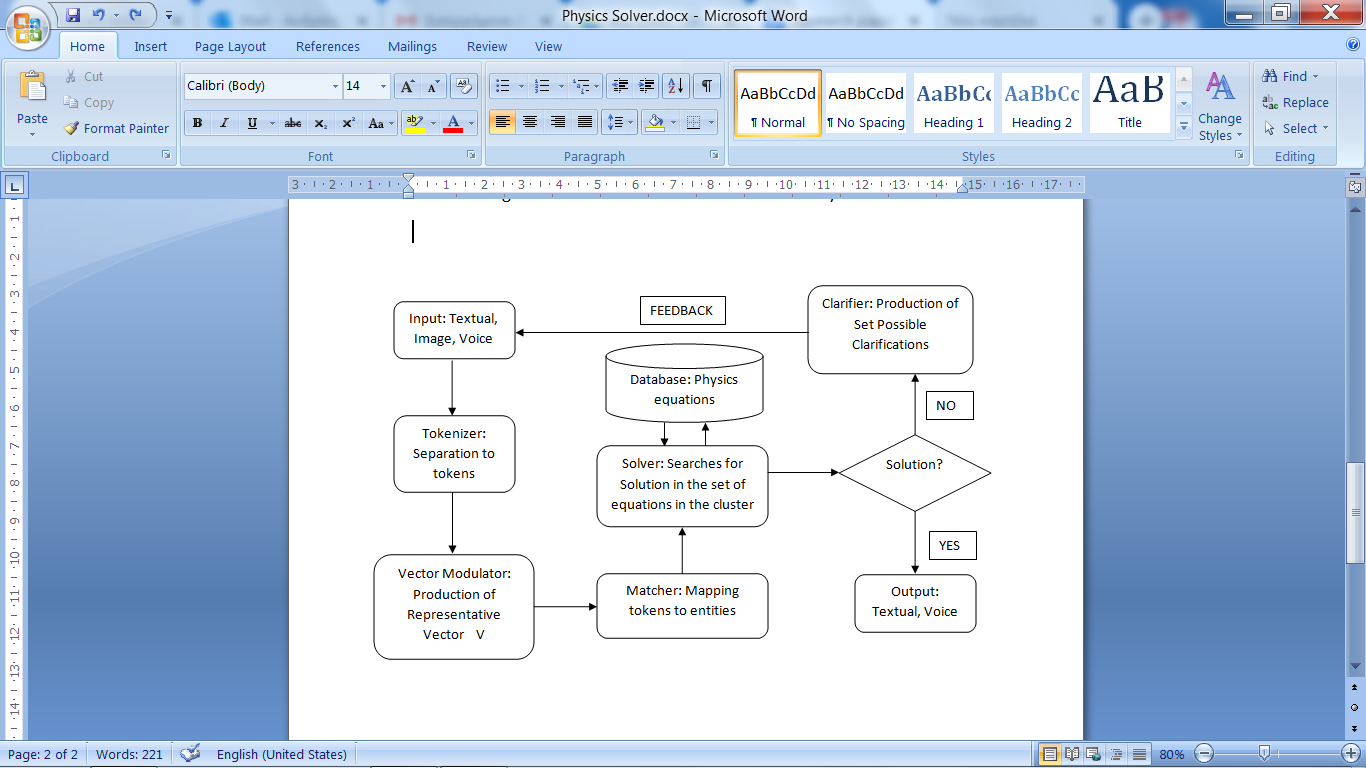
set\_v: The set with all the adjacent vertices of edge v

end: The target vertex we wish to reach

**Description**

## **Overview**

The following schema gives a visual overview of the main modules of the project and their relationships.



Our system receives as an input the formulation of the problem in the three possible formats textual, image or voice. All three result to the textual representation using APIs for voice recognition and optical character recognition. The textual representation is processed by the module called “Tokenizer” which separates the text to a set of tokens. These tokens are processed to produce a vector of properties that represent the data of the problem and can serve as the representative vector of the specific problem. The vector contains variables, their values and keywords. This vector is further processed by the “Matcher” module which contains a mapping between keywords and entities that belong to a specific dictionary developed to handle the variation in expressing the same entities, for example, “velocity” and “speed”. The output of the Matcher is a vector of properties each one corresponding to a specific node of a directed graph representation. The core module of our system the “Solver” receives the output and performs a depth first search to find a solution to the user’s request. It is worth mentioning that the solver performs an uninformed search using user’s request as a starting node without taking into account the given data.

As stated before, we use three distinct ways for the user to interact with the system. The first is the textual interface where the user gives the input data using text, but also the input data can be in the form of speech or image.

## **The Solver**

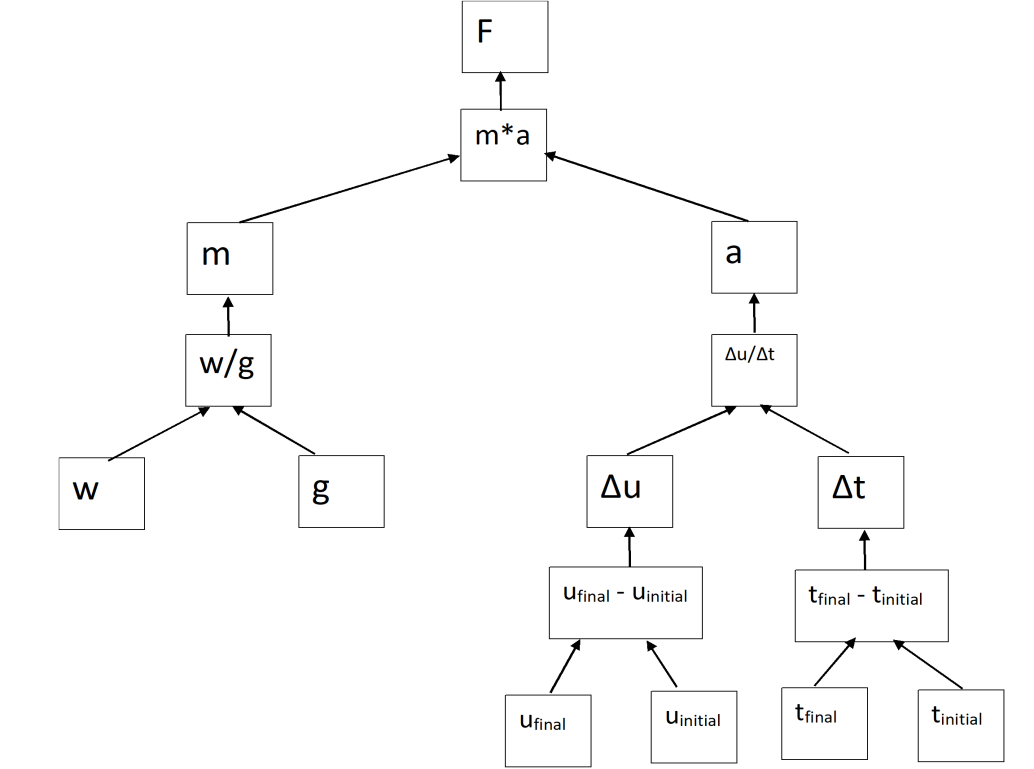
The backbone of the program consists of an SQL Database constructed through the Microsoft SQL Server Management studio, where data relevant to the problem are stored. Namely the database consists of the following tables: problem table where the problem data are stored, keyword table where keywords are stored, variable table where the variables are stored before and after calculation, formula table where the formulas are stored, unit table where the conversion data for the units is stored and a temp table for the equation processing that will be mentioned later.

When the program receives the statement of a problem it stores in the problem table and gives it a problem id. The program then tokenizes the statement and draws keywords, variables, values and unit specifications and their values are stored in separate tables, distinguishing between requested and given data. More specifically, keywords drawn from the statement are stored in the keyword table and the variables, after converting all the units to the S.I. (Systeme Internationale) through the unit table, are stored in the variable table.

Once the statement of the problem is dealt with, the program moves on to solve the problem. It first queries the keyword table and the variable table and finds all the relevant data. Based on the request data the problem selects the relevant formulas. For example, if the problem asks for the value of the force (F), the program will select all the formulas that contain Force as the request. The program then takes the first selected formula and processes it in the following way.

If the values of the variables in the equation are given, the program calculates the result and outputs the solution. However, if one or more of the values are not given the program performs the same process again, this time the requested variable being the one whose value is not given. Following the previous example, if the program selects the formula F=m\*a to calculate the value of the force, ‘m’ being mass and ‘a’ being acceleration, and we are not given the value of ‘a’, the program will make ‘a’ the request and search the table for formulas from which it can calculate its value. If in the new formula a variable is still missing, the program will perform the same procedure until it finds a result. When the program finds a result, it will recursively calculate all the values leading up to the missing one.

We have modelled the data of the entities involved in the context of physics as a Directed Acyclic Graph. The following graph is indicative of an instance in this graph.

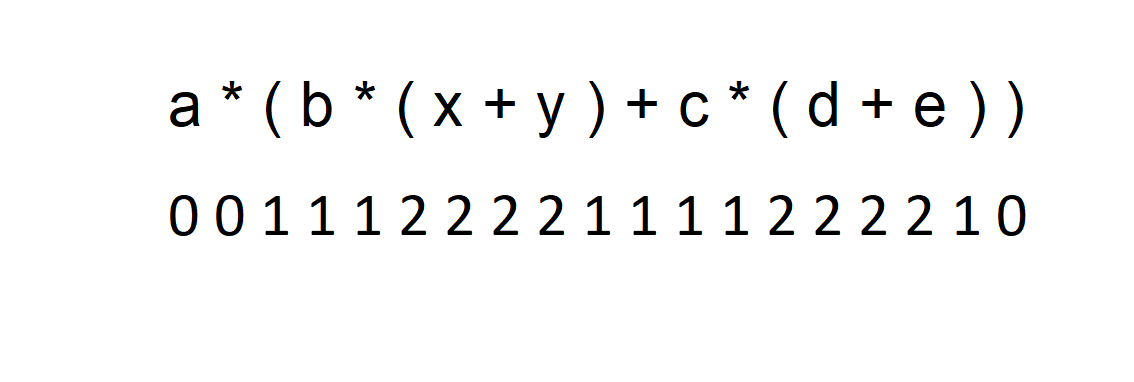


In the case that the given formulas do not suffice to produce the result the program follows the following process. It searches for formulas that have the requested variable as a given and transforms them in order to produce a new formula from which it can calculate the requested result. A simple example is the following: Given the formula W=F\*Δx (W=Work, F=Force, Δx=displacement) and the requested variable F(Force), the program will produce the following formula, F=W/Δx.

The program, however, is not limited to simple operations. It can transform equations that consist of multiplications, additions and parentheses.

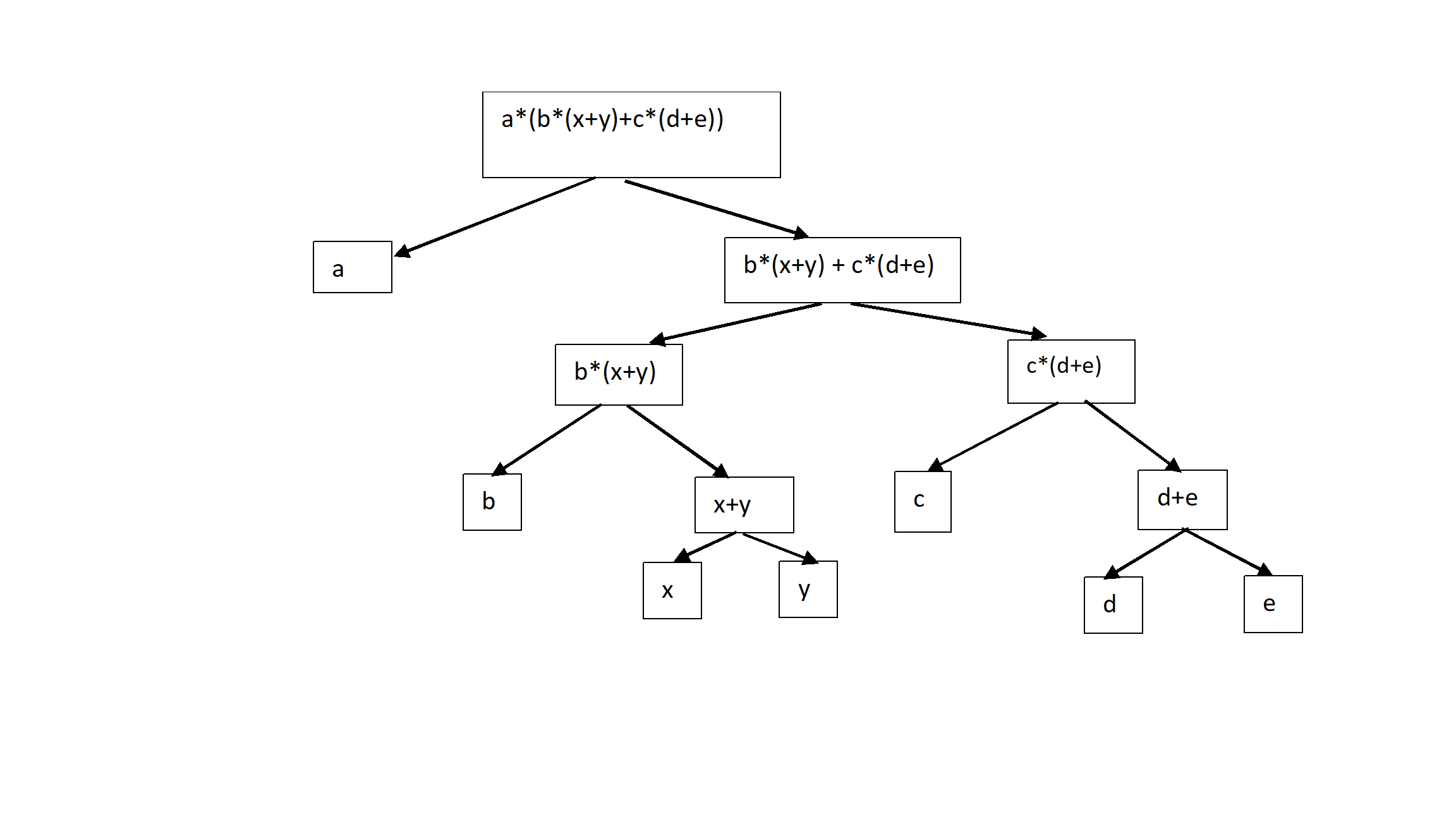
To better explain the concept we will make use of the following complex equation, through which we can demonstrate the procedure more sufficiently. The equation selected is the following: “z=a\*(b\*(x+y)+c\*(d+e))”. In order to solve for a specific variable, the program follows the following steps.

First it calculates the depth of each variable inside a parenthesis by counting the difference between the number of left and right parentheses in each position.



Afterwards, it splits the equations in the following order: First at the operators with the smallest depth, and among the operators of the same depth depending on their priority i.e. in the order “+” or “- “, “\*” or “/”, “^”. As a result, it first splits the equation into two parts at the “\*” symbol at depth “0” generating two different sub-equations: “a” and “b\*(x+y)+c\*(d+e)”, removing parentheses where redundant i.e. when the depth of the first and the last parentheses have a difference of 1.

Once the new equations have been generated, the program re-calculates the depth of each variable in each equation and performs the same procedure each time producing the following binary tree:

Having broken the equation in parts, the program will then combine them in order to generate the new formula. For example, if we want to solve for x, the program will do the following:

It will start from the first level and take the path not containing “x”. It will choose a, and store “z/a”. Then it will move to the second level, again following the opposite path of x and thus store “z/a-c\*(d+e)” etc. Eventually, the program will have generated a new formula with the requested variable being “x” that it will add to the formula table for present and future use.

**The Tokenizer and Matcher module**

The tokenizer module attempts to extract the useful information from the problem statement by applying the sliding window technique. In each scan the matcher tries to match the scanned text to an object inside the dictionary. If it finds a match it stores that value. If it doesn’t it moves on. It is critical for the sliding windowto begin with the largest chunk and end with the smallest; the first chunk contains the whole sentence. With every reduction a word is deleted. After the reduction the chunk shifts to the right until it reaches the end of the text.

Let’s provide an example to better understand this concept:

Text: “The starting velocity”

First Run           Chunk 1: The starting velocity

Second Run      Chunk 1: The starting                Chunk 2: starting velocity

Third Run          Chunk 1: The                             Chunk 2: starting                       Chunk 3: velocity

The matcher module is responsible for matching the output of the tokenizer module to the dictionary. Thus, it performs a comparison between a given token and every element of the dictionary in order to find a match. The comparison is sped up by using hashes. The dictionary contains possible words or phrases that, when present in the text, they may indicate useful information (e.g. starting velocity).

**The memory module**

When the solver module finds a solution, the program stores the solution and the path followed. More specifically, the program will store the keywords and the variables specified in the problem statement along with the solution path. If a similar problem is met in the future, our program will prefer to use the stored path instead of re-executing the depth first search. In a way, this module mimics the students’ capacity to learn a problem-solving method and then use it for a similar problem.

**The clarifier module**

If the solver module does not find a solution it means that it has exhausted all possible paths and has not reached a value to the user’s request. Then the clarifier module recalls from the memory module all the necessary paths that the solver has created. The clarifier asks the user for clarifications by giving as a feedback the missing variables or keywords, starting from the path that had the lowest number of undefined entities.

**Conclusion**

On the whole, through this project we achieved both goals we set out to achieve. First, we created a program that can “learn” from the problems it encounters. When the program is faced with a problem it has already solved, it will recognize repetitive patterns and speed up the computation by recalling the already-computed paths from the memory. Second, the program has become a valuable learning companion for high school students, as it provides the solution and the steps followed in the process of solving a problem. We acknowledge that the program can be further improved to include more advanced concepts of physics that require a higher level of algebraic reasoning and to perform a better text analysis and tokenization in order to deal with more complex written statements. We are actively engaged in enriching the program with everything that can increase its efficiency and accuracy.

**Bibliography**

AI in Education. (2019). Retrieved from <https://www.ted.com/talks/ai_in_education>

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.(2009). *Introduction to Algorithms*. Cambridge Massachusetts: The MIT Press.

Frankish, K., & Ramsey, W. M. (2018). *The Cambridge handbook of artificial intelligence*. Cambridge: Cambridge University Press.

Freeman, A., Adams Becker, S. & Cummins, M. (2017). NMC/CoSN Horizon Report: 2017 K-12 Edition. Austin, Texas: The New Media Consortium. Retrieved January 20, 2020 from https://www.learntechlib.org/p/182003/.

Jurafsky, D., & Martin, J. H. (2014). *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. India: Dorling Kindersley Pvt, Ltd.

Kleinberg, J., & Tardos, E. (2006). *Algorithm design*. Boston: Pearson/Addison-Wesley.

Russell, Stuart J. (Stuart Jonathan). (2010)*. Artificial intelligence : a modern approach.* Upper Saddle River, N.J. :Prentice Hall.

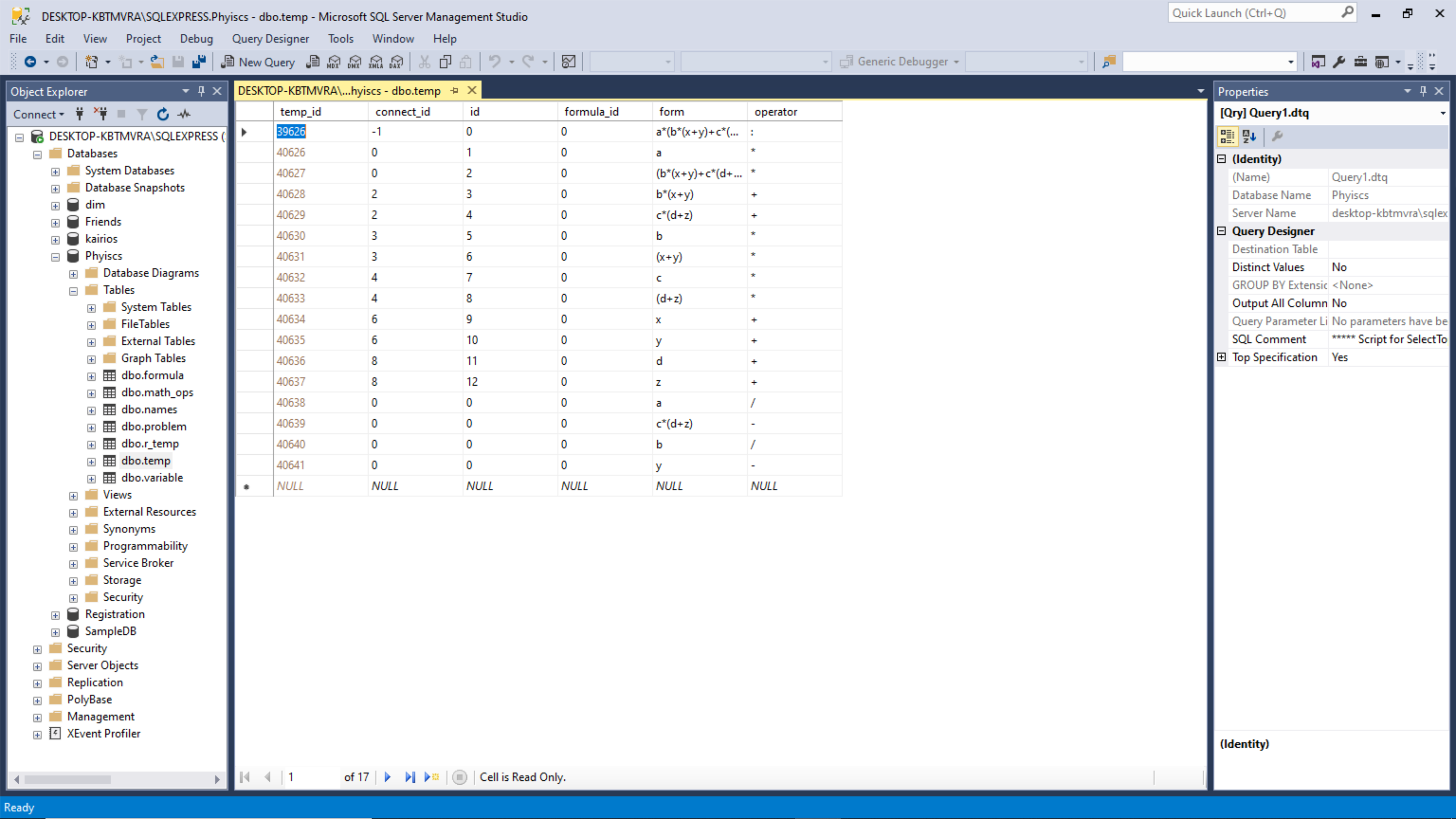
Schuck S., Aubusson P., Burden K., Brindley S. (2018) Current Trends in Technology-Enhanced Learning. In: Uncertainty in Teacher Education Futures. Springer, Singapore.

**Appendix**

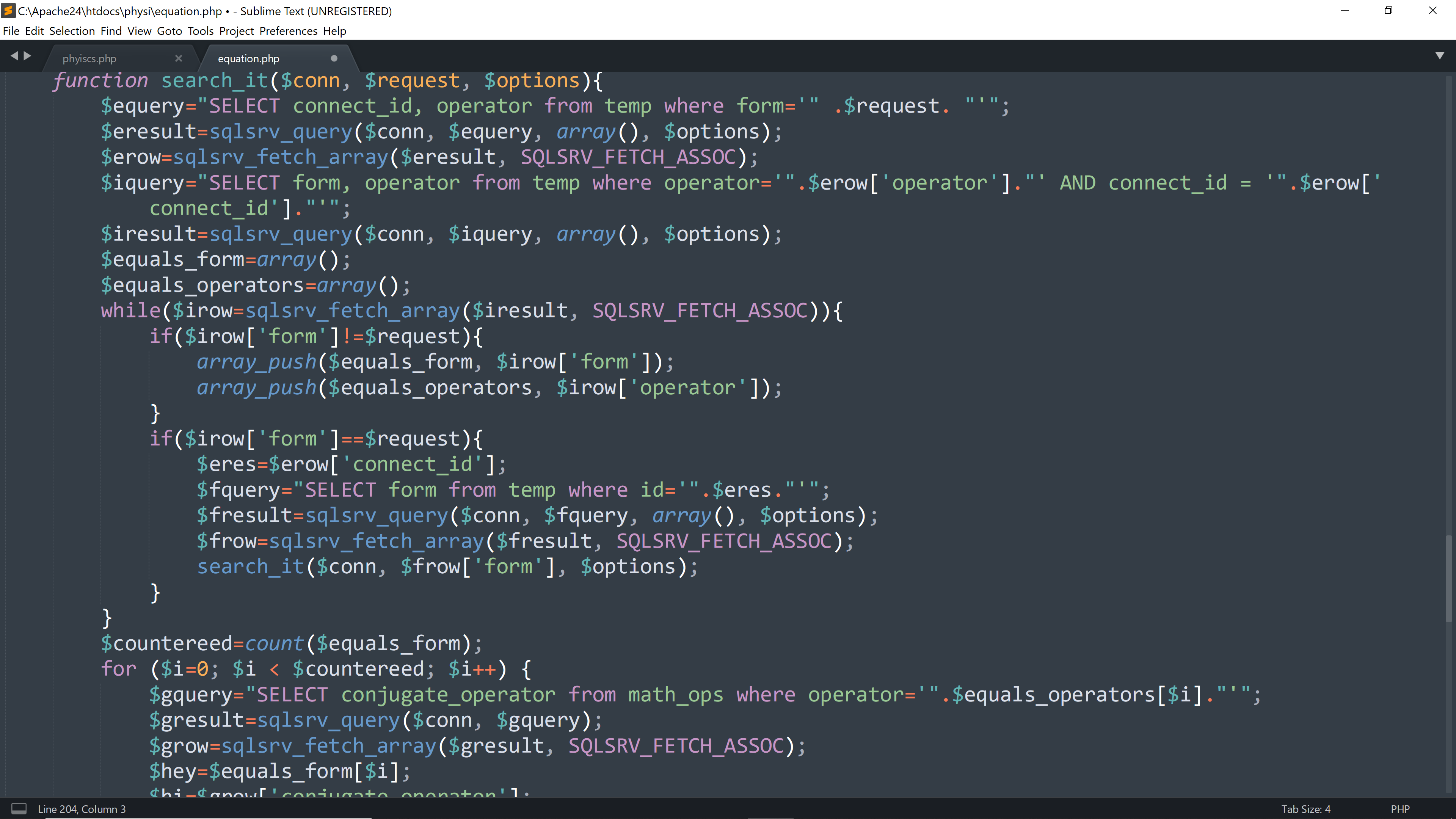
## 

Part of the code of the recursive algorithm

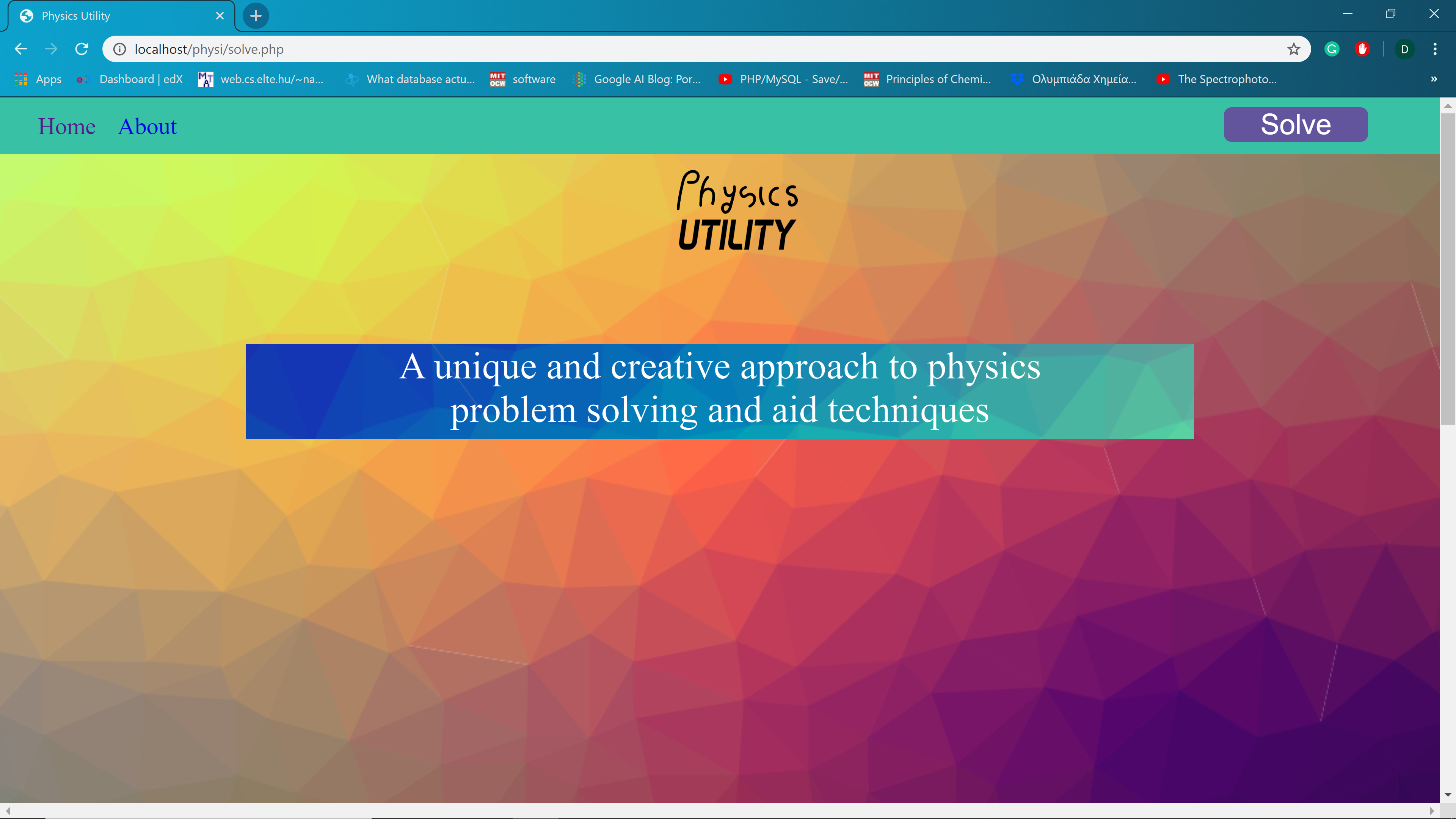
The variable table



The equation processing table



Part of the equation processing code



The user interface