Combinatorial Analysis and Recursive Algorithm for Solving the Countdown Numbers Game

Raymond Jonathan Dwi Putra Julianto - 13524059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: annayuliati69@gmail.com , 13524059@std.stei.itb.ac.id

Abstract— The Countdown Numbers Game is a mathematical puzzle that challenges players to combine a set of given integers using basic arithmetic operations (+, −, ×, ÷, ( )) to reach a specific target number. This game popularized by the British television show "Countdown," that presents a rich of combination and step to get the target number. This paper analyzes the mathematical structure of the game using combinatorics to model number combinations and implements a recursive algorithm to explore all valid expression trees that could lead to a solution. The result highlights the effectiveness of combining discrete mathematics concepts with computational search methods in solving arithmetic-based puzzles.

Keywords—countdown numbers game, combinatorics, recursive algorithm, arithmetic puzzle,

I. INTRODUCTION

Number games are among the most intriguing and intellectually stimulating forms of entertainment in the world. These games not only provide fun and enjoyment for many people, but they also serve as effective exercises to keep the brain active and sharp. The Countdown Numbers Game is a mathematical puzzle game in which the objective is to use a set of given numbers with basic arithmetic operations to reach a target number. One possible solution could involve combining the numbers using addition, subtraction, multiplication, or division to get as close as possible or exactly reach the target. This game has been popularized by the British television show *Countdown*, which has been airing since 1982 and remains on the air to this day. In the show, contestants are challenged to solve number puzzles under time pressure, typically within 30 seconds, adding an element of urgency and excitement to the game.

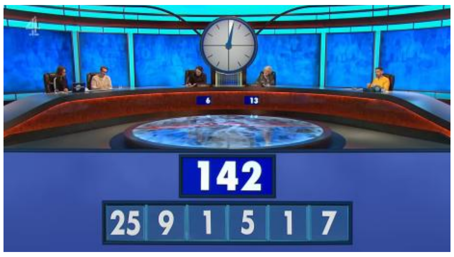


Fig. 1 Countdown Show and Example of the Countdown Numbers Game (source: https://youtu.be/b2VAPADS4N0?si=41v9vd2Nx2Qgoblr)

This game is simple. There will be six numbers randomly selected from the following integer list :{1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 25, 50, 75, 100}.The numbers 1 to 10 are referred to as small numbers, and the numbers 25, 50, 75, and 100 are referred to as large numbers. Players may choose up to a maximum of four large numbers for the game, meaning that the remaining numbers will be taken from the small numbers to complete a total of six.

Once the six numbers are chosen, a random three-digit target number (usually between 100 and 999) is generated. The goal of the player is to combine the six selected numbers using basic arithmetic operations to reach the exact target number or get as close as possible. The numbers only be used once, and not all the numbers must be used. In Countdown, participants with an exact solution will get 10 points. If no one succeeds in doing so, the contestant with the closest answer gets 7 points if their result is within 5 of the target number, or 5 points if it is within 10.

An example of the Countdown Numbers Game from figure1. the numbers 25, 9, 1, 5, 1, and 7 with a target number of 142. One solution that can be possible is: (25 × 5) + (9 × 1) + (7 + 1) = 142.

II. FUNDAMENTAL THEOREM

A. *Combinatorics*

Combinatorics is one of the main branches of mathematics that focuses on counting, arrangement, and selection of elements from a set based on specific rules, without the need to manually enumerate all possible outcomes. There are two basic counting principles in combinatorics, the product rule and the sum rule.

1) Product Rule

   When a task can be broken down into two tasks, with n ways to do the first task and m ways to do the second, then there is n × m possible ways for the whole task to happen.

2) Sum rule

   When a task can be done in n ways or in m ways, and the set of n ways does not overlap with the set of m ways (i.e., they are mutually exclusive), then there are n + m possible ways for the whole task to happen.

These two basic counting principles are often not sufficient to solve more complex problems. So, there are some advanced counting principles techniques that are required.

1) The Subtraction Rule (Inclusion–Exclusion for Two Sets)
This rule is an extends the idea of the sum rule. When a task can be done in **n** ways or in **m** ways and some of the set **n** ways does overlap with some of the set **m** ways, which means there are some ways on that two set and counting twice, we can use find possible ways with their union of the two set with the formula would be:

$$|n \cup m| = |n|+|m|-|n \cap m|$$

2) Permutation
Permutation is a total arrangement of objects where the order of the arrangement matters. In other words, different sequences of the same elements are considered distinct outcomes. Permutations are used to count the number of possible ways to arrange elements when the position or order is important. Permutation can be written P(*n,r*), where n is the number of available elements, and r is the number of arranged elements. The formula for permutations is:

$$P(n,r) = \frac{n!}{(n-r)!}$$

3) Combination
Combination is a total arrangement of objects where the order of the arrangement does not matter. Unlike permutations, combinations focus solely on which elements are chosen, not how they are arranged. As a result, different orders of the same elements are counted as a single outcome. Combination can be written as C(n,r**)** where n is the total number of elements, and r is the number of elements to be chosen. The formula for combinations is:

$$C(n,r) = \frac{n!}{r!\,(n-r)!}$$

4) Combinations with Repetition (star and bars)
Combinations with repetition involve selecting items from a set where elements can be chosen more than once, and the order of selection does not matter. This is different from standard combinations where each element can be used only once. The formula for combination with repetition is :

$$C(n+k-1,k) = C(n+k-1,n-1)$$

Which n is the number of distinct item types and k is the number of items to be selected.
We can visualize these combinations using stars to represent the number of items selected, and bars to represent the separation between different item types. This method is known as the stars and bars technique.
Example, Suppose we want to place 9 identical objects into 5 distinct boxes. Each arrangement of stars and bars such as:

$$*** \,|*|\, *** \,|\,|\, **$$
$$** \,|*|\, *** \,|\, * \,|\, **$$
$$*** \,|*|\, * \,|\, * \,|\, ***$$

represents a different way to distribute the 9 objects. Since there are 9 stars and 4 bars (for 5 boxes), the total number of such combinations is:

$$C(5+9-1,9-1) = C(13,9) = 715\, ways$$

B. Countdown Numbers Game

Countdown numbers game is the mathematical puzzle from Britain's oldest game show with the gameplay is arranged in 6 numbers with basic operation to get the numbers target. We can use this game just to play for fun or even for competition.

In countdown numbers game, there are 25 numbers from this integer list :{1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 25, 50, 75, 100} which the number 1 to 10 is called small numbers and the number 25, 50, 75, and 100 is called large numbers. The target, often called the "CECIL", will display the target numbers between 100 to 999 from the 6 numbers that have been chosen.

The rules of play of countdown numbers game is,

1) One of the players requests the number of large numbers to be used in the game with the maximum of four. The moderator then randomly selects that amount of large and the remaining slots are filled with randomly chosen small numbers.
2) Players are allowed to use basic arithmetic operators, such as addition (+), subtraction (-), multiplication (+), division (÷), and parentheses (( )). These operators can be used more than once as needed. However, when using the division operator, the result must be a whole number.
3) Each number can be used only once, and not all six numbers are required to be used in forming the solution.
4) The game will be played within a 1-minute time limit, and the player with the exact number target will earn 10 points. If no one succeeds in doing so, the contestant with the closest answer gets 7 points if their result is within 5 of the target number, or 5 points if it is within 10.

The target numbers are randomly generated by "CECIL" using a set of six numbers. 1226 sets can solve any problem, ranging from 100 to 999. We can know it by the number of large numbers.

TABLE I. PERFECT SOLUTION SETS BY NUMBER OF LARGE NUMBERS

#Big	Count
0	5
1	614
2	603
3	4
4	0

[a] #Big refers to how many large numbers use in the sets

From table I, sets with one large number and two large numbers dominate the perfect solution, and sets with zero, three, and four large numbers have least the perfect solution. There is no perfect solution for set with four large numbers. This happens because including too many large numbers limits the flexibility of operations using small values, making it harder to form all target numbers between 100 and 999.

In a set of numbers, not all numbers contribute equally to the probability of finding a solution. Some numbers, due to their mathematical properties, are more flexible and strategic for use in arithmetic operations than others.

TABLE II. AMOUNT REACHABLE BY SINGLE SMALL NUMBER

Game Set Contains	Amount Reachable
1	775
2	804
3	819
4	813
5	821
6	824
7	842
8	836
9	842
10	835

[b] Game Set Contains refers to the specific small number that is guaranteed to be included in a game set, while Amount Reachable refers to the total amount of target numbers that can be solved.

Table II shows the number of reachable targets when a game set is guaranteed to contain a specific small number. For example, if our set of numbers includes 3, we can expect to reach 819 different target numbers. Single number 1 is the least useful for reaching targets, single number 7 and 9 are the most effective small numbers for approaching target numbers.

TABLE III. AMOUNT REACHABLE BY DOUBLE SMALL NUMBER

Game Set Contains	Amount Reachable
1	655
2	738
3	775
4	773
5	780
6	798
7	828
8	818
9	831
10	808

[c] Game Set Contains refers to the specific small number that is guaranteed to be included in a game set, while Amount Reachable refers to the total amount of target numbers that can be solved.

Table III explains the number of reachable targets when the set contains a pair of identical small numbers. If our set of numbers has a pair of 3, we can expect to reach 775 target numbers. Double numbers 1 are the least effective for reaching targets, otherwise double numbers 7 and 9 are the most effective pairs for approaching target numbers.

Based on these two tables, the numbers 7 and 9 are highly effective for reaching many targets. This is likely due to their properties, such as being relatively large prime numbers within the set of small numbers, which provides greater flexibility in calculations.

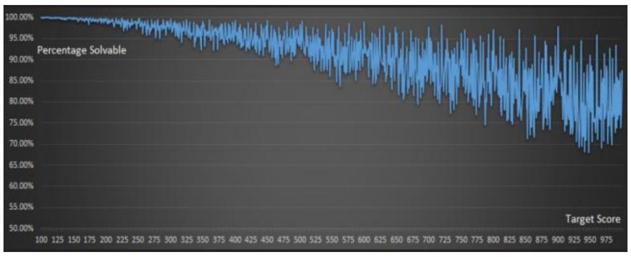


Fig. 2 Percentage of Game Sets Solving a Given Target (source: https://datagenetics.com/blog/august32014/index.html*)*

The target numbers have percentage chance to be solved with random set numbers. By recording figure 2, we can see that the target numbers get larger, the percentage of be able solve gets smaller. This is normally because the big target numbers need to be multiplication rather than the small target numbers that can be reached by addition or multiplication small numbers. Under 316 number targets, the percentage chance to be solved is above 95%. It is the opportunity to get to know the set of numbers that are chance solvable. The most difficult to reach the target number is 947 with the percentage chance is 68.09%, meanwhile the most complicated reach target number is 961.

III. IMPLEMENTATION

A. *Combination Sets of the Six Numbers Choose*

There are 6 numbers required to play the Countdown Numbers Game, which means there are many possible combinations of number selections. The six numbers can be filled with a random mix of large numbers and duplicated small numbers, since small numbers from 1 to 10 appear twice each.

For example, if a player chooses 2 large numbers for the start game, then 4 small numbers left to complete the 6-numbers set. Since small numbers can appear more than once, several variations are possible in how these small numbers are selected. Let L_i represent the chosen large number and S_i represent the chosen small number. There are multiple ways the small number portion can be formed. For instance:

1) *Combination 1 : L_1, L_2, S_1,S_1,S_2,S_2*
The small number set contains two duplicated values. The total number of such combinations is: $C(4,2) \times C(10,2) = 270$.

2) *Combination 2 : L_1, L_2, S_1,S_1,S_2,S_3*
The small number set contains one duplicated and two distinct values. The total number of such combinations is: $C(4,2) \times 10 \times C(9,2) = 2160$.

3) *Combination 3 : L_1, L_2, S_1,S_2,S_3,S_4*
The small number set contains four completely distinct values. The total number of such combinations is: $C(4,2) \times C(10,4) = 1260$.

From this illustration, we can get the amount of combination for 2 of large numbers is $270 + 2160 + 1260 = 3690$.

We can count the combination of each large number by using:

$$C(4,n) \times ValidSmallNumbers(6-n)$$

which n is the number of large numbers chosen (ranging from 0 to 4), and ValidSmallNumbers (6 − n) represent the number of valid small-number combinations needed to complete the 6-number set. The small number combinations are constrained such that each small number from 1 to 10 can appear at most twice.

The function ValidSmallNumbers uses bounded multiset counting to determine the number of valid combinations that can complete the 6-number set. This method ensures that no small number is used more than twice, as limited by the Countdown game rules. The total number of valid small-number combinations for each possible count of large numbers is summarized in table below:

TABLE IV. VALIDSMALLNUMBERS(6-N)

Large number (n)	k = 6 - n	ValidSmallNumbers(k)
0	6	2850
1	5	1452
2	4	615
3	3	210
4	2	55

[d] ValidSmallNumbers(k) refers to the number of valid small-number combinations of size k.

From Table IV, we can find the total number of valid combinations for each possible count of large numbers. When :

- 0 Large Numbers (n = 0)
  $C(4,0) \times ValidSmallNumbers(6) = 1 \times 2850 = 2850$
- 1 Large Numbers (n = 1)
  $C(4,1) \times ValidSmallNumbers(5) = 4 \times 1420 = 5808$
- 2 Large Numbers (n = 2)
  $C(4,2) \times ValidSmallNumbers(4) = 6 \times 615 = 3690$
- 3 Large Numbers (n = 3)
  $C(4,3) \times ValidSmallNumbers(3) = 4 \times 210 = 840$
- 4 Large Numbers (n = 4)
  $C(4,4) \times ValidSmallNumbers(2) = 1 \times 55 = 55$

Adding all of these together, we obtain the total number of valid 6-number combinations that can be formed in the Countdown Numbers Game: 2850 + 5808 + 3690 + 840 + 55 = 13243. If the target value is selected randomly, there are exactly 900 possible target values (ranging from 100 to 999 inclusive). Given that there are 13,243 unique 6-number sets, the total number of possible game instances is: 900 × 13243 = 11918700 games.

B. *Reachable Numbers from a Given Game Set*

A set of six numbers selected according to specific rules must be combined using basic arithmetic operations in order to reach a target number between 100 and 999. Assume a, b, c, d, e, f represents the six chosen integers. These elements may be paired and operated on sequentially, generating intermediate results that can bring the total closer to the target or assist in subsequent operations.

For instance, choosing two values from the set, such as a and d, and applying an operation (e.g., addition) results in a new value g = a + d. This newly derived value g is then introduced into the pool of usable operands and may serve one of two strategic purposes:

- Be directly combined with another number to approach the target, or
- Serve as a steppingstone toward a more complex multi-step construction.

This process continues recursively, where each new result becomes a candidate for further combination, adhering to the constraint that each original number can be used at most once.

We can estimate the total number of valid arithmetic expression sequences by stepwise combining two numbers at a time and applying one of the four basic operations, with only integer results allowed. The total number of valid operation trees formed from a six-number set is:

- First operation: $C(6,2) \times 4 = 60$
- Second operation: $60 \times C(5,2) \times 4 = 2400$
- Third operation: $2400 \times C(4,2) \times 4 = 57600$
- Forth operation: $57600 \times C(3,2) \times 4 = 691200$
- Last operation: $691200 \times C(2,2) \times 4 = 2764800$

Thus, the total number of valid arithmetic expression sequences formed from a six-number set is:

60 + 2400 + 57600 + 691200 + 2764800 = 3516060

This value represents the total number of valid expression trees for a single six-number set. It does not indicate how many target numbers between 100 and 999 can be reached. Rather, it defines the maximum number of distinct operation paths evaluated during brute-force solution attempts in the Countdown Numbers Game.

C. *Algorithm to find the solution Countdown Numbers Game*

To find the solution for the Countdown Numbers Game, the program iterates through all possible combinations to find a valid solution. The code is structured into three specific steps to accomplish this.

- Generating a Permutation Tree
  A permutation tree is created from the given set of numbers to efficiently generate all unique orders of those numbers. The procedure responsible for this step is `generate_permutation_tree`
- Finding Solutions from the Permutation Tree
  The program iterates through each branch of the permutation tree to find a possible solution that matches the target number. The algorithm uses Reverse Polish Notation (RPN) with a stack and a recursive backtracking approach to try all operator combinations. The function used for this process is `recursive_place_operators`
- Formatting the Solution Result
  A raw solution, once found, is converted into an expression tree using the `Operation` class. This

allows the solution to be optimized and formatted into a human-readable string.

Here is the implementation of the code using python language

```
def solve(arg_input, arg_target):
    """
The main entry point function for the solver.
It initializes the process, calls the
permutation tree generation, starts the
solution search,and prints the final unique
solutions.
    """

class NumberNode:
    """
Defines the data structure for a node within
the permutation tree. Each node represents a
single number in a sequence, and it holds
references to subsequent numbers (children) in
the permutation.
    """

def generate_permutation_tree(input):
    """
Builds the permutation tree data structure from
the provided list of numbers. Its purpose is to
efficiently generate all unique permutations
(orders) of the numbers.
    """

def
recursive_generate_permutation_tree(numbers,
indices, node):
    """
A recursive helper function that does the
actual work of building the branches of the
permutation tree, ensuring no duplicate
permutations are generated.
    """

def place_operators(root):
    """
Kicks off the second major stage: the solution
search. It initializes the evaluation stack and
the RPN record, then calls the recursive
solver.
    """

def recursive_place_operators(node,
num_consumed):
    """
The core recursive solver that uses a
backtracking algorithm. At each step, it tries
either adding a new number to the stack or
applying an operator(+, -, *, /) to the numbers
already on the stack.
    """

class Operation:
    """
    Defines the data structure for a node
within an expression tree. This class is used
to convert a raw RPN solution into a
mathematical expression that can be formatted
and optimized. It holds the value, operator,
children, and flags for optimization (e.g.,
negative, reciprocal).
    """

def sort_children(children):
    """
A helper function to sort the children of an
expression node. This is crucial for ensuring a
consistent, canonical output (e.g., "100 + 50"
instead of "50 + 100").
    """

def compare_operations_less_than(op1, op2):
    """
A helper comparison function that defines the
standard order between expressions, used by the
sorting function.
    """

def contruct_solution_string(ops_record):
    """
The main function for the third stage. It takes
a raw solution record (in RPN format) and
orchestrates the process of converting it into
a final, human-readable string.
    """

def construct_operation_tree(ops_record):
    """
Builds the initial expression tree data
structure from an RPN record.
    """

def recursive_optimize_operation_tree(node):
    """
Simplifies the expression tree by "flattening"
sequential operations (like a+b+c) and
normalizing operators to eliminate duplicates
and unnecessary parentheses.
    """

def
recursive_print_binary_operation_tree(node):
    """
an alternative formatting method that prints
the expression exactly as it was calculated,
including all original parentheses, without
optimization.
    """

def main():
    """
Handles all user interaction, such as asking
for the numbers and the target, and then calls
the main 'solve' function to start the process.
    """
```

Fig. 3 Function and Class used on the Code (source: writer's archive)

IV. Analysis

A. *Decision Making*

Brute-forcing each possible combination of the number set is the simplest method available to all players; however, it requires a significant amount of time and a fair degree of luck to find a valid solution. There are some strategic, but not exactly the target number, in every single situation.

1) *Pitch and Put Technique*

The Pitch and Put technique involve a large number with another number to get close to the target number, called pitch, followed by adjusting the number into the target number by another set number left, called put. This technique is very common use by the player of countdown construct efficient solutions from the given numbers. Here are some examples of the game.

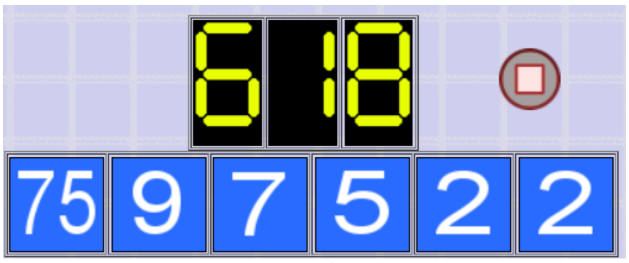


Fig. 4 Game Example for Pitch and Put
(source: https://happysoft.org.uk/countdown/numgame.php)

In this game example, we use one large number for the numbers of set. The set numbers are 75, 9, 7, 5, 2, and 2 with **75** being the largest. The number 75 we can multiply by 9 to get number 675, which is pretty close we are about 60 away. We can subtract 675 by 9 multiply 7 to get 612, but because 9 is can only use by one time, we can make correspond to another equation that same value with 612 it is $(75 - 7) \times 9 = 612$. After performing the pitch operation, we apply the put step by adding a number to reach the final target by addition 6. The number of 6 can be find by $5 + (2 \div 2)$ or $2 \times (5 - 2)$. The solution of this game example is $((75 - 7) \times 9) + 5 + (2 \div 2)$ or $((75 - 7) \times 9) + (2 \times (5 - 2))$.

2) *Prime Factorizations*

This technique attempts to solve the puzzle by expressing the target number as a product of its prime factors and then reconstructing these factors using the available set numbers. It can be difficult because you need to factor the target number which can the factor of target number is so big or maybe the target number itself is the prime number. Here are some examples of prime factorizations:

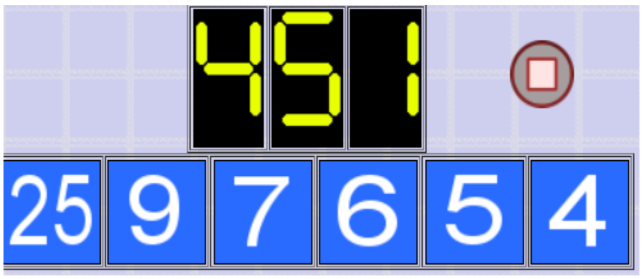


Fig. 5 Game Example for Prime factorization
(source: https://happysoft.org.uk/countdown/numgame.php)

In this game example, we use one large number for the numbers of set. The set numbers are 25, 9, 7, 6, 5, and 4. The target number of this game is 451. By factoring the target, we find that $451 = 11 \times 41$. This factor might take time if player is not easy for factorized. This factor can be got by addition 5 and 6 to get 11 and addition 25, 9, and 7 to get 41. So, the solution of this game example is $(5 + 6) \times (25 + 9 + 7)$.

This example of a target number may seem conveniently suited for factorization. But what happens when the next target number is 452?

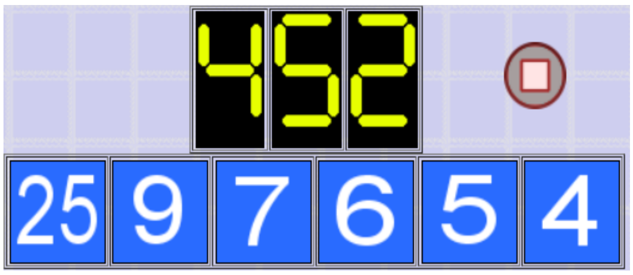


Fig. 6 Game Example for Prime factorization
(source: https://happysoft.org.uk/countdown/numgame.php)

In this case, we use the same set of numbers as in the previous example: 25, 9, 7, 6, 5, 4. However, the new target number 452 is not as easily factorizable into values that can be readily constructed from the given set. While 452 can be factorized as 4 × 113 or 2 × 2 × 113, the number 113 is a prime that is too large to be built from the available digits in the set.

Instead, we can look for nearby numbers that possess simpler and more accessible prime factorizations. One such number is 448, which factors into 64 × 7. Factor 7 is already present in the set, and 64 can be composed using the expression 25 + 9 + (6 × 5). By computing (25 + 9 + (6 × 5)) × 7, we arrive at 448 and then add the remaining number 4 to reach the target value. So, the solution of this case is $(25 + 9 + (6 \times 5)) \times 7 + 4$.

3) *Using 7 and 9 for Reachable*

We know before that single and double numbers of 7 and 9 are the most effective for approaching target numbers. We can use this knowledge to form a practical strategy during the game. Here are some examples of using 7 and 9 for reachable:

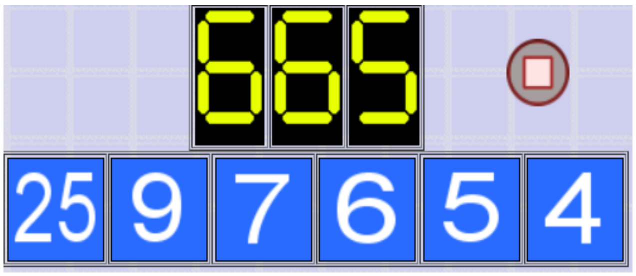


Fig. 7 Game Example for Using 7 and 9 for reachable
(source: https://happysoft.org.uk/countdown/numgame.php)

In this game example, we use one large number for the numbers of set. The set numbers are 25, 9, 7, 6, 5, and 4. The target number of this game is 665. The target number can be factorized by 7, which results in the factor 95. The factor of 95 can be got by $(25 \times 4) - 5$. So, the solution of this game example is $7 \times ((25 \times 4) - 5)$.
However, this method can be problematic because there is no guarantee which small numbers will be included in the set. The situation becomes worse if the number of large numbers chosen is not one or two, due to the limited number of reachable targets in those configurations.

B. *Code Analysis*

The implementation of solving the Countdown Numbers Game will be done by code in python language. The algorithm implemented is a recursive algorithm with backtracking. This approach is used to systematically explore all possible mathematical expressions that can be formed from the given numbers.

The process begins with the `solve` function, which accepts 6 input numbers and a target number. The first crucial step is to call `generate_permutation_tree` to generate all unique sequences (permutations) of these numbers. This ensures that each sequence of numbers is processed only once, even if there are duplicate numbers.

```
# Initialization and Permutation Tree Generation
def solve(arg_input, arg_target):
    # ... initialize target ...
    root = generate_permutation_tree(arg_input)
    # ... begin solution search ...

def generate_permutation_tree(input):
    # ... create all unique permutations ...
```

Fig. 8 Initialization and Permutation Tree Function Call Code (source: writer's archive)

After the permutation tree is created, the core algorithm is executed by the `place_operators` function, which then calls `recursive_place_operators`. This recursive function operates using the *Reverse Polish Notation* (RPN) method with a *stack*. At each step, this function attempts one of two possibilities: either taking a new number from the permutation to place on the stack, or applying one of the four arithmetic operators (+, -, *, /) to the top two numbers on the stack. This process continues until a solution is found or all possibilities have been exhausted.

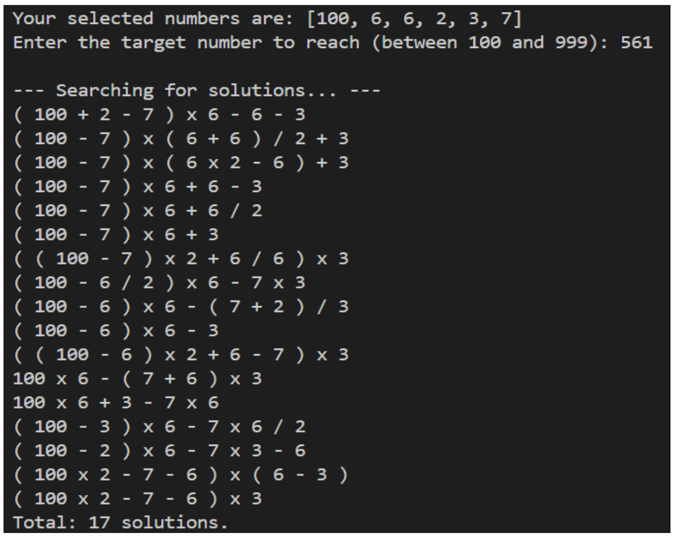
```
def recursive_place_operators(node, num_consumed):

# ... (stopping condition if solution is found)...

# ... (recursive step to add a new number) ...

# ... (recursive step to try all operators) ...
```

Fig. 9 Recursive Place Operator Code to Find the Solution (source: writer's archive)

For a single set consisting of 6 distinct numbers, the algorithm must explore all possible valid "operation trees". The total number of these distinct operation paths that the algorithm might evaluate for a single permutation of numbers is 3516060.

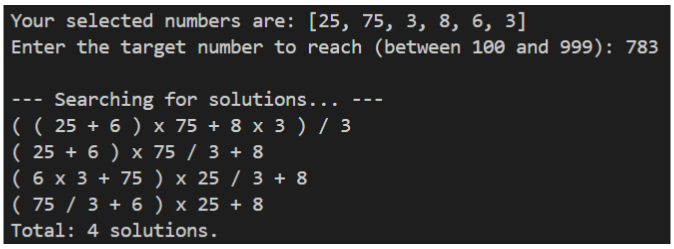
When a solution is found in its raw RPN format (e.g., [100, 25, '+', 5, '*']), functions `contruct_solution_string` and `recursive_optimize_operation_tree` are called to convert it into a canonical, simplified, and human-readable string (e.g., (100 + 25) * 5). This optimization process is important to ensure that mathematically identical solutions are displayed in a uniform format.

Here some test cases using the code implementation



```
Your selected numbers are: [100, 6, 6, 2, 3, 7]
Enter the target number to reach (between 100 and 999): 561

--- Searching for solutions... ---
( 100 + 2 - 7 ) x 6 - 6 - 3
( 100 - 7 ) x ( 6 + 6 ) / 2 + 3
( 100 - 7 ) x ( 6 x 2 - 6 ) + 3
( 100 - 7 ) x 6 + 6 - 3
( 100 - 7 ) x 6 + 6 / 2
( 100 - 7 ) x 6 + 3
( ( 100 - 7 ) x 2 + 6 / 6 ) x 3
( 100 - 6 / 2 ) x 6 - 7 x 3
( 100 - 6 ) x 6 - ( 7 + 2 ) / 3
( 100 - 6 ) x 6 - 3
( ( 100 - 6 ) x 2 + 6 - 7 ) x 3
100 x 6 - ( 7 + 6 ) x 3
100 x 6 + 3 - 7 x 6
( 100 - 3 ) x 6 - 7 x 6 / 2
( 100 - 2 ) x 6 - 7 x 3 - 6
( 100 x 2 - 7 - 6 ) x ( 6 - 3 )
( 100 x 2 - 7 - 6 ) x 3
Total: 17 solutions.
```

Fig. 10 Countdown Number Game with number set 100, 6, 6, 2, 3, 7 and target number 561 (source: writer's archive)



```
Your selected numbers are: [25, 75, 3, 8, 6, 3]
Enter the target number to reach (between 100 and 999): 783

--- Searching for solutions... ---
( ( 25 + 6 ) x 75 + 8 x 3 ) / 3
( 25 + 6 ) x 75 / 3 + 8
( 6 x 3 + 75 ) x 25 / 3 + 8
( 75 / 3 + 6 ) x 25 + 8
Total: 4 solutions.
```

Fig. 11 Countdown Number Game with number set 25, 75, 3, 8, 6, 3 and target number 783 (source: writer's archive)

V. CONCLUSION

The Countdown Numbers Game is a mathematical puzzle that can be used for entertaiment or to improve our logical thinking skills. The game offers immense variety through its different combinations of six numbers and the target number. With 11,918,700 possible game instances, a significant portion 3,516,060 are valid and solvable. To find a solution, players can employ several strategies:

- Pitch and Put Technique**:** A method where one number is multiplied by another to get close to the target, and the result is then adjusted using the remaining numbers.
- Prime Factorization**:** This strategy involves using the prime factors of the target number (or a number close to it) to guide the calculation.
- Using 7 and 9 for Reachable**:** Statistically, the numbers 7 and 9 are the most effective small numbers for reaching a target, making them strategic starting points.

From a computational standpoint, the solution can be found efficiently using an algorithm that combines a permutation tree (to handle number order), Reverse Polish Notation (RPN) with a stack (to manage calculations), and a recursive backtracking approach to explore all possibilities.

Appendix

Source code for implementing the solution Countdown Numbers Game program : https://github.com/Raymond13524059/Countdown-Number-Game-Solver---13524059---Raymond-Jonathan-Dwi-Putra-Julianto.git

Acknowledgment

The author would like to express sincere gratitude to Dr. Ir. Rinaldi, M.T. and Mr. Arrival Dwi Sentosa, S.Kom., M.T., lecturers of the IF1220 Discrete Mathematics course, for their guidance, encouragement, and support throughout the semester. Thanks to their support, the author was able to gain valuable insights and learn many new things throughout the course. The author is also deeply thankful to family and friends who have been motivating and supporting.

References

[1] R. Munir, "Kombinatorika Bagian 1," Departemen Informatika, Institut Teknologi Bandung, 2024. Accessed: 13 June 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/18-Kombinatorika-Bagian1-2024.pdf

[2] R. Munir, "Kombinatorika Bagian 2," Departemen Informatika, Institut Teknologi Bandung, 2024. Accessed: 13 June 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025/19-Kombinatorika-Bagian2-2024.pdf

[3] "Countdown numbers game," dCode. Accessed: 09 June 2025. [Online]. Available: https://www.dcode.fr/countdown-numbers-game#q2

[4] exitexit, "countdown-numbers-solver," GitHub. Accessed: 09 June 2025. [Online]. Available: https://github.com/exitexit/countdown-numbers-solver/tree/main

[5] N. C. C. Browne, "Countdown math game," DataGenetics, August 3, 2014. Accessed: 17 June 2025. [Online]. Available: https://datagenetics.com/blog/august32014/index.html

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Juni 2025

[illegible]

Raymond Jonathan Dwi Putra Julianto - 13524059