Solving Water Pouring Problem with Branch & Bound Algoritm

Jonathan Christhoper Jahja 13519144
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : 13519144@std.stei.itb.ac.id

***Abstract*—Water pouring problem is a common problem that has existed for a long time. The problem uses a simple mathematical and logical concept. The most common variation of this puzzle and the variation that this paper will observed consists of 3 jugs of water that have different capacity. The jugs are irregularly shaped and unmarked, so it isn't possible to accurately measure any quantity of mater in the jug. This paper will show how to solves the water pouring problem using branch and bound algorithm.**

***Keywords*—Algorithm, cost, goal, branch, water, jug.**

I. Introduction

The water jug problem or the water pouring puzzles is a common problem that dates back into 1960s[1]. It is a famous puzzle that usually used for teaching kids problem-solving skills. The puzzles use basic mathematical and logical concepts. There are many variations that was used, but the most common ones are the three jugs variation and the two jugs variation.

The three jugs variation has the following rules, there are three jugs that have each capacity of 3, 5, and 8 litres. The jugs have irregular shapes and are unmarked, so the amount of water inside of the jugs couldn't be measured. The only way to determine the amount of water is when the jug is full or water left inside the jug from pouring into another jug. Goal of this puzzle is to divide the water so the 8 and 5 litres jug has 4 litres of water and the 3 litres jug is empty. The puzzle starts with 8 litres jug full of water and then we can pour the jugs back and forth into each other to get the goal state. There are no extra water or loss of water when pouring. This paper will use the three jugs variation because of the familiarity it holds.

Meanwhile the two jugs variation has similar rules, the main difference is in this variation there are taps and sink. So we have the option to fill the water into full or emptied the jug. Usually there are only 5 litres jug and 3 litres jug, and both starts empty. The goal is to have 4 litres of water inside the 5 litres jug and the 3 litres jug empty. Amount of water provided is infinity so you can fill and empty the jugs multiple times. This puzzles are also famous when it is featured in the movie *Die Hard: With a Vengeance* in 1995. In the movie, they use the two jugs variation.

Branch and Bound algorithm is an algorithm design paradigm which is generally used for solving combinatorial optimization problems. These problems are typically exponential in terms of time complexity and may require exploring all possible permutations in worst case. The branch and Bound algorithm technique solves these problems relatively quick[2]. Branch and Bound algorithm can solve problems like 1/0 knapsack problem, sliding puzzle, and route finding even though it's not the most efficient. This algorithm measure cost for each steps and expands on the step that has minimum cost. The formula to measure the cost is different for every problem.

II. Basic Theory

Branch and Bound algorithm is a combination of the Breadth First Search algorithm and least cost search[4]. For instance, the Breadth First Search is expanding the nodes based in FIFO order, so the first node that is founded are being reviewed first. Meanwhile in Branch and Bound, the expansion is based on the order of the cost of each node.

Besides BFS, B&B algorithm is also similar to backtracking algorithm. Both algorithm find solution by creating a state tree and both algorithm eliminate nodes that aren't leading to the solution or goal. The difference with backtracking is that backtracking doesn't use optimization meanwhile B&B measure each possible solution with a cost. The B&B and backtracking expansion are also different like with BFS, because backtracking uses BFS when expanding the nodes.

The cost is basically to measure the closeness of a state to the goal state. It usually calculated by heuristically. The formula to measure the cost is usually different for each problem. So the optimization and efficiency is based on the formula that was used. B&B algorithm has function that measure a bound for each node, this function is called bound function. It will eliminate or ignored every node that doesn't lead to optimal solution. In the water pouring problem, the cost will be used as the lower bound because in this problem we are trying to find the minimal steps to get to the goal. Therefore the algorithm will eliminate or ignore every node that has cost bigger than the lower bound.

For instance, in the following is steps for B&B algorithm implementation for tile sliding puzzle.

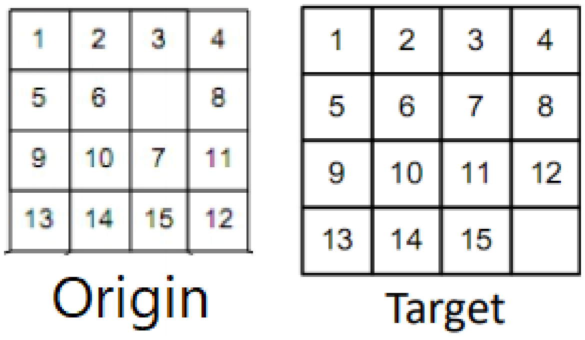


Fig. 2.1 Example of Tile Sliding Puzzle

Each node from the state tree will represent every possible movement from the empty tile. Formula for the node will be measure is,

$$c(i) = f(i) + g(i) \tag{1}$$

c(i) : the cost for the node

f(i) : the distance between current empty tile position and the original position

g(i) : the sum of tile that are not

Algorithm starts by creating an empty queue. With the formula we can expand the state root as shown in the following,

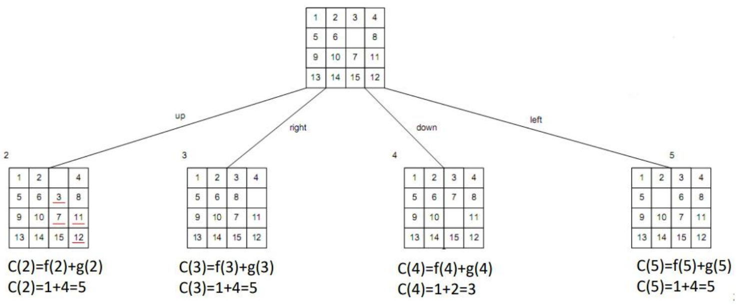


Fig. 2.2 Expansion of state tree from the origin state

As shown in Fig. 2.1, the root has 4 child node we can insert to queue with the order the lowest cost is in the head of the queue. We can observe that the lowest cost of all possible moves is the 4^{th} node with the cost of 3, so the 4^{th} node is the head of the queue. By following the rules of B&B algorithm, we continue by reviewing the head of the queue. If the head isn't the goal, we expand it again and insert the new found node by order of the cost it has. Algorithm will loop this sequence until the head is the goal or until the queue is empty meaning there are no solution. In the following are the results of the tree expansion,

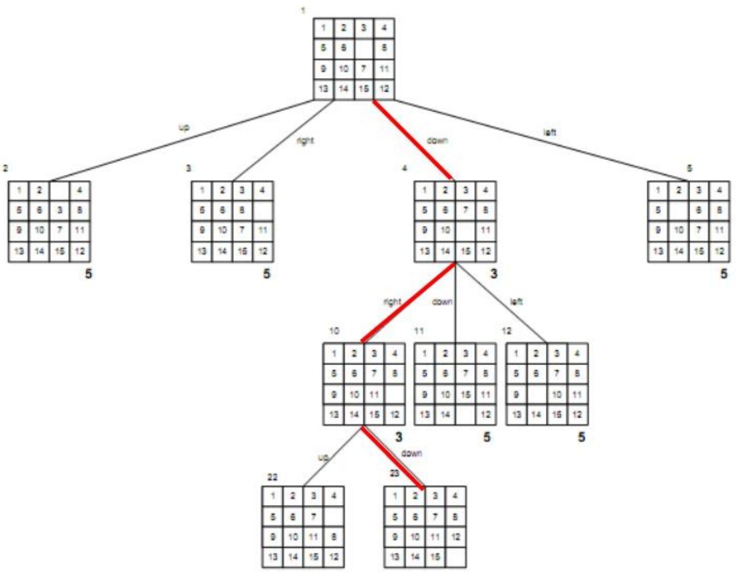


Fig. 2.3 The complete expansion until finding the target state of the puzzle

After reaching the target, we can measure its cost with

$$c(23) = f(23) + g(23) = 3 + 0 = 3 \tag{2}$$

Because the cost of the target state is 3, we can set this as the lower bound and eliminate other states or nodes that have a cost bigger than 3. In this case all of the states left have the cost of 5, so we can eliminate all of other possibilities of expansion leaving one solution only.

III. Algorithm Implementation

To implement Branch & Bound Algorithm, first we need to determine the cost formula to measure each states or nodes. In water pouring problem, we can use the goal state as the base to count the cost. So for each state we can count the difference between the volume of water it holds currently and the goal. For instance, the jugs in original state each consist 8 litres, 0 litres, and 0 litres of water. The target is 4 litres, 4 litres, and 0 litres of water in each jug. By measuring the difference between two states we get

$$|8-4| + |0-4| + |0-0| = 8 \tag{3}$$

We got the main formula to count the cost, but it isn't enough because the algorithm can expand the same node that it already observed multiple times if that state is close enough to the goal. To prevent the expanding the same node, we can add one more variable which is the amount of pouring that has been done. With this extra cost, the state that is the same with the previous state will have more cost because the amount of step to get to the latest state is bigger and we need the shortest step to the solution. In simpler words the lower you go in the tree, the bigger its cost. Following is the formula to measure the cost,

$$c(i) = f(i) + g(i) \tag{4}$$

c(i) : the cost for the state

f(i) : the amount of pouring to get to this state

g(i) : the difference of quantity of water between this state and the goal

By using this formula, we can finally construct the state tree starts from the root which is the original state from each jug. In this case, we will also avoid expanding to a state that has been founded before, because the possibilities of a state always going to be merely the same. For convenience, nodes will represent the state of each jug by the current amount of water it holds in the following order, 8 litres jug, 5 litres jug, and 3 litres jug. The paper will also name the 8 litres jug, 5 litres jug, and 3 litres jug in the following A, B, and C.

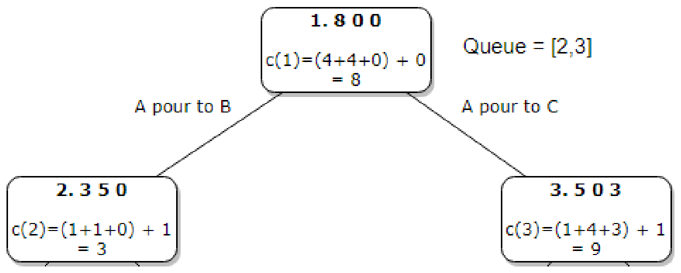


Fig 3.1 Expansion of the root state

In Fig 3.1, the expansion of the root only resulted in two possible moves, because B and C are empty. Then insert the 2nd and 3rd state into the queue with 2nd state on the head of the queue because 2nd state has a cost of 3 which is lower than the cost of the 3rd state. Then we continue by reviewing the head of the queue which is the 2nd state.

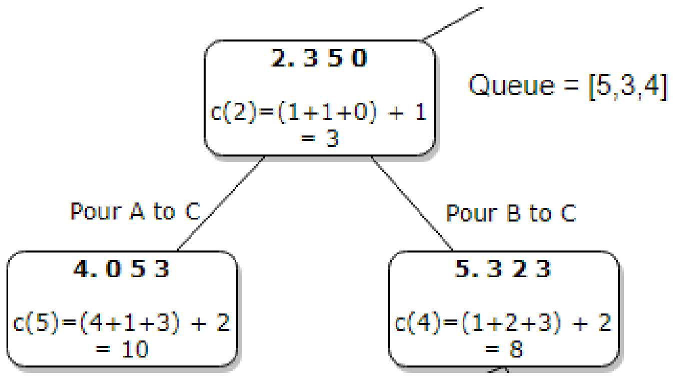


Fig 3.2 Expansion of the 2nd state

From Fig 3.2, the expansion resulted to the 4th and 5th state each with a cost of 10 and 8. The 8 0 0 states weren't included as possible moves because that is the previous state. Then insert the new found nodes into the queue, that makes 5th state to be the new head. After repeating these steps for until the goal is found we will get the following tree,

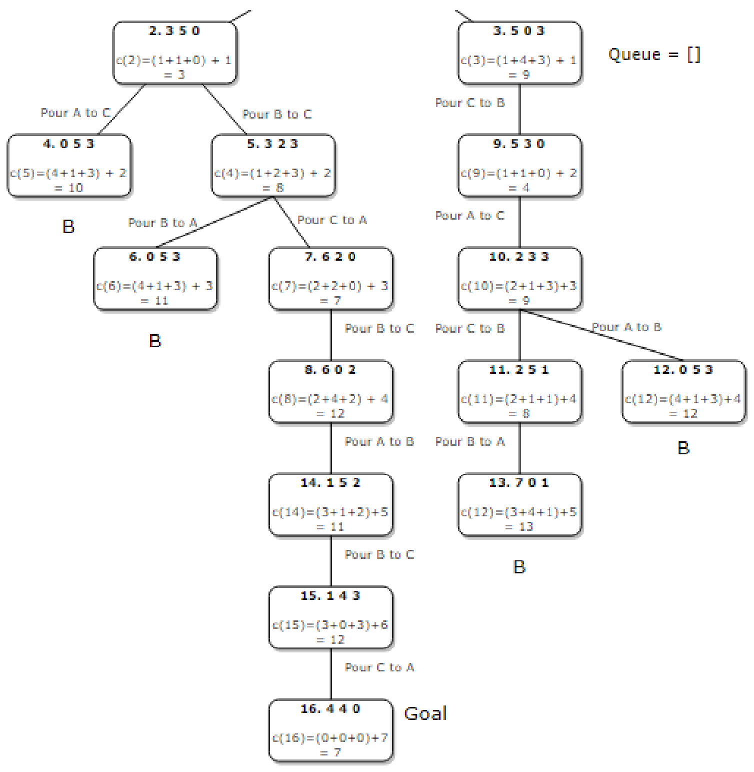


Fig 3.3 The full state tree

By observing the Fig 3.3, the number of the nodes represent the order they are being founded. The result that was found is in the 16th state with the cost of 7, with the path starts from 1-2-5-7-8-14-15-16. The 4th, 6th, 12th is being terminated because the only possible from the expansion is 5 0 3 or 3 5 0 which is already found before. Meanwhile the 13th state is terminated because its cost is higher that the goal and every state that has the higher cost will be terminated or ignored.

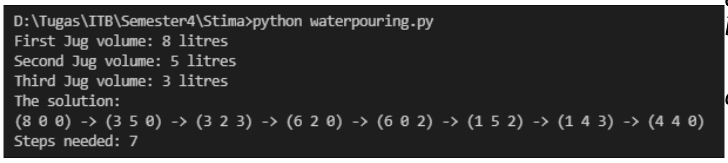
From expanding the 5th state, you will get the new queue with the new head of 7th state. Then expanding the 7th state, we will only get the 8th state. It is because the only other possible state is 8 0 0 which is the root. The new head of the queue is the 3rd state since the cost of 7th state is 12. 3rd state expands to 9th state with the cost of 4 so it becomes the new head of the queue. Same as before, 9th state expands to 10th state then the 10th state expands to the 11th and 12th state. The 11th state has the lower cost so it expands to 13th state with the cost of 13, makes the 8th state to become the new head.

From here the expansion becomes straightforward because the rest of the state cost lower than 13. 8th expands to 14th then to 15th and finally expands to 16th state which is the goal. The 16th cost has a cost of 7 which will become the lower bound, so every other node that has cost higher than 7 will be eliminated. The only state left is 13th state that has cost higher than 7 so it is eliminated. That left us with only one path that makes it the most optimal path to get to the goal.

This algorithm can be used not only for this occasion, you can have variate the root and the goal. But not all variance can have a solution. This problem or puzzle only have solution if and only if the desired volume is a multiple of the greatest common divisor of all the integer volume capacities of jugs. This principal is based on Bézout's identity[4].

IV. Code Implementation

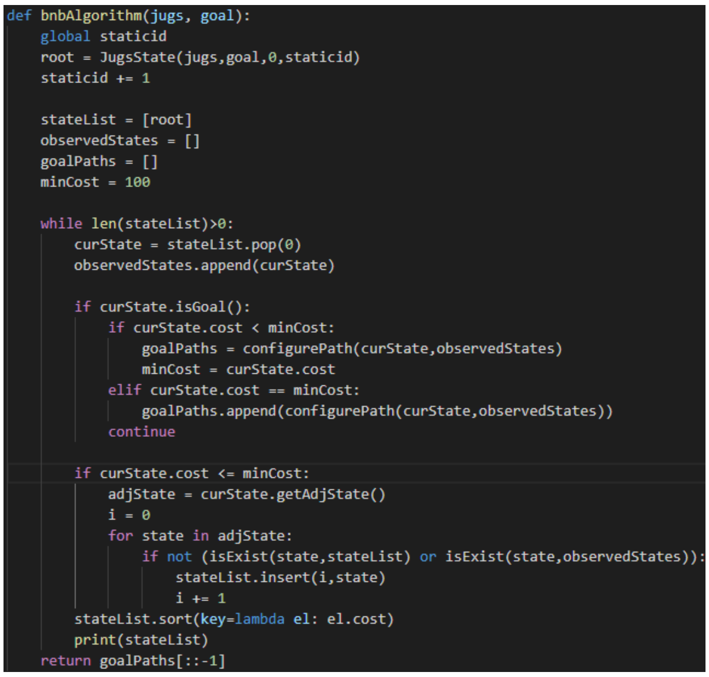
This paper will show the implementation of this algorithm using the Python programing language.



```
D:\Tugas\ITB\Semester4\Stima>python waterpouring.py
First Jug volume: 8 litres
Second Jug volume: 5 litres
Third Jug volume: 3 litres
The solution:
(8 0 0) -> (3 5 0) -> (3 2 3) -> (6 2 0) -> (6 0 2) -> (1 5 2) -> (1 4 3) -> (4 4 0)
Steps needed: 7
```

Fig 4.1 Output of the Branch & Bound algorithm using Python

In Fig 4.1, we can observe that the program also has the same exact solution of the manual search. In the following is the Branch & Bound algorithm code in Python,



```python
def bnbAlgorithm(jugs, goal):
    global staticid
    root = JugsState(jugs,goal,0,staticid)
    staticid += 1

    stateList = [root]
    observedStates = []
    goalPaths = []
    minCost = 100

    while len(stateList)>0:
        curState = stateList.pop(0)
        observedStates.append(curState)

        if curState.isGoal():
            if curState.cost < minCost:
                goalPaths = configurePath(curState,observedStates)
                minCost = curState.cost
            elif curState.cost == minCost:
                goalPaths.append(configurePath(curState,observedStates))
            continue

        if curState.cost <= minCost:
            adjState = curState.getAdjState()
            i = 0
            for state in adjState:
                if not (isExist(state,stateList) or isExist(state,observedStates)):
                    stateList.insert(i,state)
                    i += 1
        stateList.sort(key=lambda el: el.cost)
        print(stateList)
    return goalPaths[::-1]
```

Fig 4.2 Code implementation of Branch & Bound algorithm

In the code we need to initialize the queue containing the root state, empty list of states that has been observed, and the list of all paths that leads to goal (notice that this is a list because there can be multiple solution for a problem). The writer uses object oriented programming for defining a state as an object. State object has attributes of list of jugs, the cost, the id state, and the parent id of the state. The jug is also defined as an object that have attributes of max capacity and current amount of water it holds.

In summary, the algorithm just implements the basic theory mentioned in past section. The main loop of while works in the following steps,

a) Get the head of the queue to be reviewed.

b) Insert the state into the list of observed states.

a)

c) Check if current state is the goal, if it is the goal we need to check if the cost if it is below the lower bound. This to handle if there are multiple steps that leads to goal, we just get the lowest one.

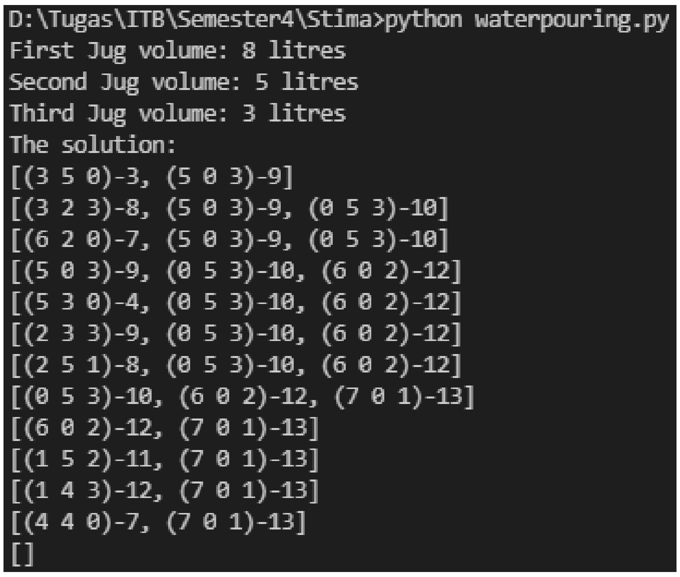
d) If it isn't the goal, compare the cost to the lower bound. If its higher don't expand the state, else continue expanding the node by finding possible movement that hasn't been founded before.

e) Insert the new found states in order of the cost to the queue.

b)

f) Repeat a-e steps until the queue is empty.

We can observe the alteration of the queue by printing it each iteration like in the following figure,



```
D:\Tugas\ITB\Semester4\Stima>python waterpouring.py
First Jug volume: 8 litres
Second Jug volume: 5 litres
Third Jug volume: 3 litres
The solution:
[(3 5 0)-3, (5 0 3)-9]
[(3 2 3)-8, (5 0 3)-9, (0 5 3)-10]
[(6 2 0)-7, (5 0 3)-9, (0 5 3)-10]
[(5 0 3)-9, (0 5 3)-10, (6 0 2)-12]
[(5 3 0)-4, (0 5 3)-10, (6 0 2)-12]
[(2 3 3)-9, (0 5 3)-10, (6 0 2)-12]
[(2 5 1)-8, (0 5 3)-10, (6 0 2)-12]
[(0 5 3)-10, (6 0 2)-12, (7 0 1)-13]
[(6 0 2)-12, (7 0 1)-13]
[(1 5 2)-11, (7 0 1)-13]
[(1 4 3)-12, (7 0 1)-13]
[(4 4 0)-7, (7 0 1)-13]
[]
```

Fig. 4.3 The queue in the algorithm

In Fig. 4.3, we can observe the order of the states being reviewed until the goal is found. Each state is represented by the "jugs-cost", that's why the queue is ordered by the cost.

V. Variance of the Problem

To demonstrate the algorithm, this paper will use another variance of the jugs.

A. The first variance

In this first case we will use a 12 litres jug, 5 litres jug, and 2 litres jug. The goal is to get a state of 6 5 1 in each respective jug. The following is the state tree resulted by using the algorithm,

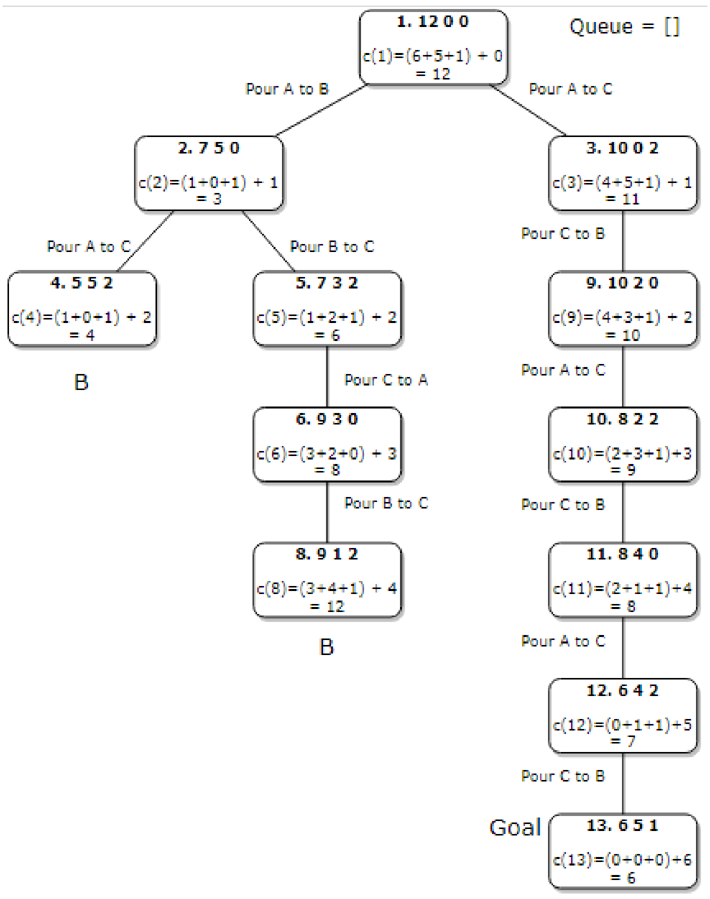
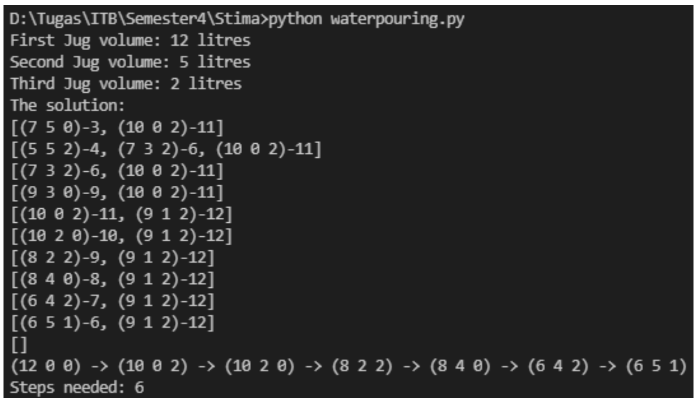


Fig 5.1 State tree of the 12 litres, 5 litres, and 2 litres jug.

Because there are only 3 jugs, the state tree from Fig 5.1 and Fig 3.3 will look similar. Same as before after we found the goal in the 13th state, we can eliminate the expansion from 8th state because the cost of 8th state is higher than the lower bound which is 6.

We will find the same result if we use the program like in the previous section. The following is the output and the queue iterations of the program,



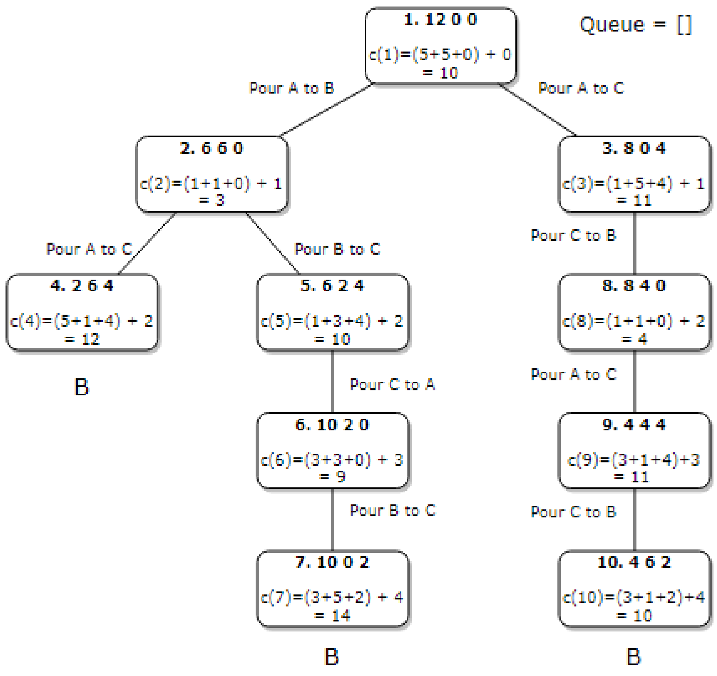
```
D:\Tugas\ITB\Semester4\Stima>python waterpouring.py
First Jug volume: 12 litres
Second Jug volume: 5 litres
Third Jug volume: 2 litres
The solution:
[(7 5 0)-3, (10 0 2)-11]
[(5 5 2)-4, (7 3 2)-6, (10 0 2)-11]
[(7 3 2)-6, (10 0 2)-11]
[(9 3 0)-9, (10 0 2)-11]
[(10 0 2)-11, (9 1 2)-12]
[(10 2 0)-10, (9 1 2)-12]
[(8 2 2)-9, (9 1 2)-12]
[(8 4 0)-8, (9 1 2)-12]
[(6 4 2)-7, (9 1 2)-12]
[(6 5 1)-6, (9 1 2)-12]
[]
(12 0 0) -> (10 0 2) -> (10 2 0) -> (8 2 2) -> (8 4 0) -> (6 4 2) -> (6 5 1)
Steps needed: 6
```

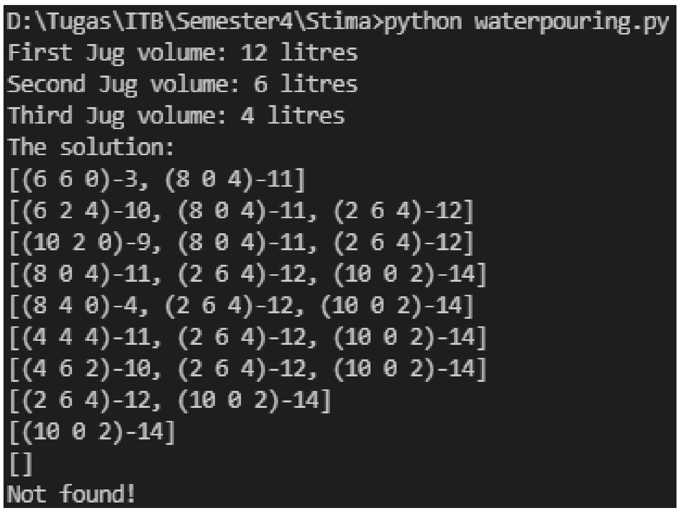
Fig 5.2 Output of the program with the 12 litres, 5 litres, and 3 litres jug

As observed in Fig 5.2, the order of queue matches the order of the reviewed state in the state tree. This proofs that the algorithm in the program works exactly like the manual algorithm.

B. *The no solution*

In this case, we will observe when there is no solution of the puzzle given. The capacity of three jugs are 12 litres, 6 litres, and 4 litres. The goal to achieve is 7 5 0 in each respective jug. State tree resulted from the expansion of root is the following and the result from the program,





```
D:\Tugas\ITB\Semester4\Stima>python waterpouring.py
First Jug volume: 12 litres
Second Jug volume: 6 litres
Third Jug volume: 4 litres
The solution:
[(6 6 0)-3, (8 0 4)-11]
[(6 2 4)-10, (8 0 4)-11, (2 6 4)-12]
[(10 2 0)-9, (8 0 4)-11, (2 6 4)-12]
[(8 0 4)-11, (2 6 4)-12, (10 0 2)-14]
[(8 4 0)-4, (2 6 4)-12, (10 0 2)-14]
[(4 4 4)-11, (2 6 4)-12, (10 0 2)-14]
[(4 6 2)-10, (2 6 4)-12, (10 0 2)-14]
[(2 6 4)-12, (10 0 2)-14]
[(10 0 2)-14]
[]
Not found!
```

Fig 5.3 State tree and result of the program with the 12 litres, 6 litres, and 2 litres jug

As we can observe from Fig 5.3, the state tree has eliminated all the end state because there are no new possible movement that hasn't been found in previous states. We can proof there are no possible solution by finding the greatest common division of 12, 6, and 2 which is 2. So based on the terms of having a solution is the target of this problem must be a multiple of 2 in which the target of 7, 5, and 1 is not.

VI. Conclusion

The three water jugs puzzle or the water pouring problem is a simple mathematical problem. There are many other variations of this puzzle like the two jugs variation with the sink and taps.

Because the solution of this problem can be in a form of optimization of combinatorial possibilities, we can find the optimal solution with many algorithms. Algorithm like brute force, graph search like DFS and BFS, greedy, and in this case Branch and Bound algorithm. Branch and Bound algorithm works like backtracking but each node has a cost that can be compared to the bound we assign. Order of the expansion also follows the cost by state with the lowest cost are expanded first. For this problem we use lower bound, so we can eliminate any state that has cost higher than the lower bound. Another bound is that we can avoid expanding to a state that is already founded before. This is because the same state will always have the same exact expansion even in different level of the tree.

Cost of each state can be measured by counting the difference between the current jugs water quantity and the goal in addition with the amount of pouring that was needed to reach the corresponding state. After the goal is found, we can assign the cost of the goal state as the lower bound to eliminate other node. This algorithm can also be used to other variation of water capacity of the jugs. For a variation to have a solution, the desired target must be a multiple of the greatest common divisor between all the capacity of the jugs.

This paper shows that the most common problem can be solve by algorithm. Therefore algorithm aren't just for software development, but if used correctly it can solve many other problems.

References

[1] https://mathworld.wolfram.com/ThreeJugProblem.html. Accessed on 10 May 2021.

[2] https://www.geeksforgeeks.org/branch-and-bound-algorithm/. Accessed on 10 May 2021.

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf. Accessed on 10 May 2021

[4] https://archive.org/details/bub_gb_RDEVAAAAQAAJ. Accessed on 11 May 2021

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.

Bekasi, 11 Mei 2021

[illegible]

Jonathan Christhoper Jahja 13519144