Winning Indonesian Traditional Game 'Congklak' in One Turn

BFS Approach

Nicholas Reymond Sihite - 13522144
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522144@std.stei.itb.ac.id

***Abstract*—*Congklak* is a popular Indonesian traditional two-players game which goal is to secure as many seeds as possible in each player's *gudang* until there are no more seeds left. *Congklak* is a static game, meaning there is a large-but-limited number of possible combinations that can be played in the game. Choosing the correct combination can even lead a player to victory in just one move. This paper aims to find the shortest path possible to achieve *Congklak* victory in one turn and to find out whether it is possible to collect all 98 seeds in just one turn using the BFS algorithm. The results are it is possible to collect 54 seeds using the shortest path: 1-2-7-7-1-3-1-4-7 and the current known maximum number of seeds that can be collected in one turn is 92 seeds using path: 1-7-4-1-6-7-4-3-1-1-4-1-4-7-2-1-5-2-3-6-1-5-5-7-1.**

***Keywords*—*Congklak*; Victory, One Turn; BFS;**

I. Introduction

One of the most well-known Indonesian traditional two-players games is called *Congklak* (also known as *congkak* or *dhakon*). This traditional game originated from the Middle East and has been dated back to 7000-5000 BC. Over the centuries, it has traveled across many countries, one of which is Indonesia where it evolved into the traditional game *Congklak* that is known today. [5].

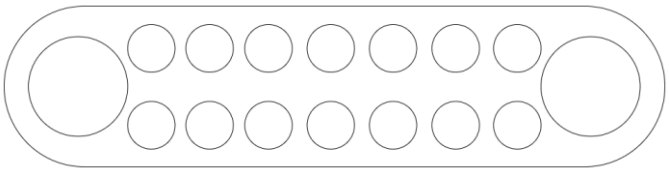

Fig. 1 Congklak Board
(source: writer's archive)

The *Congklak* board consists of 16 holes: 14 small holes and 2 large holes (known as *gudang*). Each player get 7 small holes, 1 *gudang*, and 49 seeds, which are initially divided equally into the seven holes. *Congklak* is played by selecting a hole one of the seven small holes, picking up the seeds from that hole, then distribute it one-by-one in a clockwise manner. The goal of the game is to collect as many seeds as possible in *gudang* by the end of game, which is when there are no more seeds left in the small holes.

Efficient strategy is required to win this game. Players need to contemplate which small hole will result in the most number of seeds in *gudang* by the end of the turn. The question is, can a player truly win this game in just one turn? In the writer's previous attempt at answering this question (Constructing a Decision Tree to Find the Best Starting Hole in Indonesian Traditional Game 'Congklak': Maximizing the Number of Seed in Gudang by The End of The Turn), it has been found out that a player can get 73 seeds in just one turn [6].

This paper answers that question using a different approach, which is by utilizing an algorithm called Breadth First Search (BFS) Algorithm, not only to find the minimum amount of steps to win the game, but also to find out the maximum number of seeds that can be achieved in just one player turn.

II. Theoretical Basis

A. Queue Data Structure

Queue is a type of data structure which stores its elements in a buffer that can be modified with the First-In-First-Out (FIFO) rule. There are four important terms often used when working with a queue data structure. The first two are *head* and *tail*. Head is the term used to refer to the start of the queue, i.e. the element at the first index. Tail is the term used to refer to the end of the queue, i.e. the element at the last index. The last two are *enqueue* and *dequeue*. Enqueue is the procedure of adding an element to the end of the queue as the new tail. Dequeue is the procedure of removing an element from the start of the queue (the head). When working with queues, it is 'illegal' to access elements other than the one at the head. The following is an example of a queue data structure.

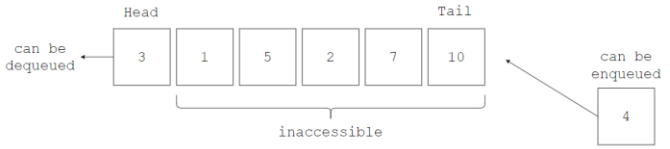

Fig. 2 Queue Data Structure
(source: writer's archive)

Queue data structure is widely used in the informatics world. Some examples include page replacement algorithms and disk scheduling algorithms in operating systems. Real life problems, like who should go first in a line, can also be solved by using a type of queue called priority queue.

B. Breadth First Search Algorithm

Breadth First Search Algorithm is a type of solution-finding algorithm that converts the problem context into a tree, then traverses the tree breadth-first [1], [2]. In other words, this algorithm prioritizes the minimum depth first and if the solution is not found, it checks the next depth. Here is an example of how the BFS algorithm traverses a tree.

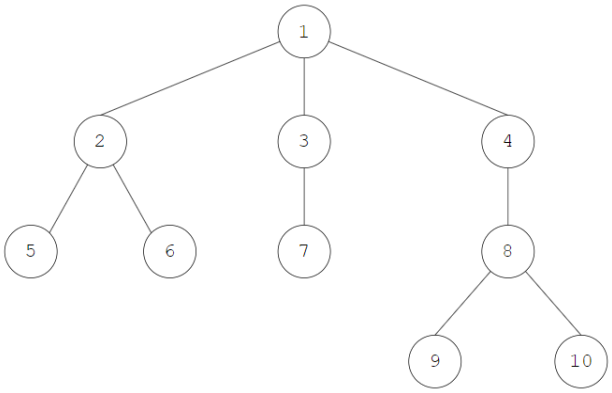

Fig. 3 BFS Algorithm Illustration
(source: writer's archive)

As illustrated, it started at depth 0 (node 1), then depth 1 (node 2, 3, and 4), and so on until it finds (one of) the problem's solution. The outline of how this algorithm works is: visit a node; visit all nodes that is adjacent to said node; visit all nodes that have not been visited and is adjacent to the previous nodes; and so on [1].

Since computers do not have eyes and are not able to 'see' the tree, this algorithm is implemented using a queue. In the form of queue, every node is put in the queue in a specific rule. To search the next node, dequeue a node from the queue. To add a node for future visitation, enqueue the node to the queue. The problem described in Fig. 3 can be represented in a queue like the following.

Current Node	Queue
1	2, 3, 4
2	3, 4, 5, 6
3	4, 5, 6, 7
4	5, 6, 7, 8
5	6, 7, 8, 9, 10
and so on	

Fig. 4 Queue Representation of Fig. 3
(source: writer's archive)

The time complexity of BFS algorithm is known to be $O(b^d)$ and the space complexity is also $O(b^d)$, where b is the maximum possible branch of a single node and d is the depth of the tree. This shows that the BFS algorithm is not the most efficient in time as well as space. However, at the cost of high memory and time usage, this algorithm always finds the optimal path to the answer.

C. Rules of Congklak

After being around for thousands of years, the rules of *Congklak* had varied, even in Indonesia. The outline of the game is still the same, which is pick a hole then distribute the seeds until all small holes are empty. The rule varies when it comes to dealing with four special cases, which are running out of seeds in *gudang*, running out of seeds in a non-empty hole, running out of seeds in an empty small hole in self turf, and running out of seeds in an empty small hole in opponent's turf. To put aside the differences, the following are the rules used in the making of this paper [3].

1. The board is divided into two areas.
2. Each player gets seven small holes in the area closest to them and one *gudang* on the left side of the board in each player's perspective.
3. Seven seeds are put in each small holes
4. Players determine which one should go first (method used is left up to the player).
5. For each turn, the current player may take all seeds from one of the player's non-empty small holes.
6. All seeds taken are to be distributed one by one in a clockwise manner for every hole (small and large) with the exception of the opponent's *gudang* (must be skipped).
7. When faced with the four special cases, do the following:
    a. If the player ran out of seeds in self *gudang*, the player may choose another non-empty small hole to restart the distribution.
    b. If the player ran out of seeds in a non-empty small hole, the player may take all the seeds in that hole and continue the distribution.

c. If the player ran out of seeds in an empty small hole in self turf, the player may take the only seed in that hole (put by the player) and the seeds in the exact opposite of that hole (opponent's turf) then put them all in the player's *gudang* (end of turn).

d. If the player ran out of seeds in an empty small hole in opponent's turf, the player's turn will end.

8. After a player finishes his/her turn, the other player may play, and so on until no more seeds are left in the small holes.

9. The player with the most number of seeds in his/her *gudang* by the end of the game, wins the whole game.

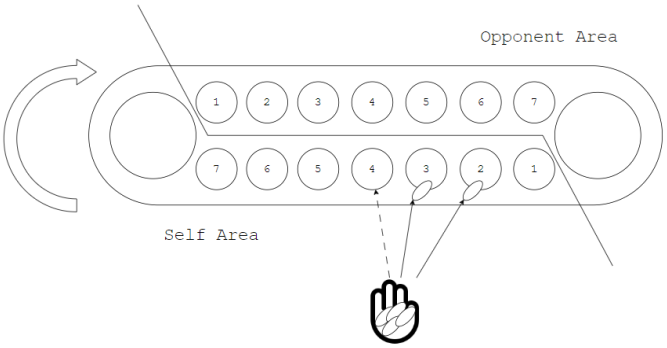

Fig 5. Congklak Rules
(source: writer's archive)

III. Analysis

A. *Defining Congklak Terms and Conditions of Winning*

According to the established rules written on chapter II part C, each player would have 7 seeds × 7 small holes = 49 seeds. Since there are two players in one game, there would be a total of 98 seeds constantly throughout the entire game. Also, the rules stated that the "The player with the most number of seeds in his/her *gudang* by the end of the game, wins the whole game". Theoretically, a player would only need half of the total seeds plus one, which is 50, gathered in the player's *gudang* to win. Therefore, should a player obtain 50 or more seeds at one point of the game, the player had already secured the victory seat.

B. *Mapping Congklak into BFS Elements*

BFS search algorithm can be represented as a tree; therefore, the first thing to do is determine what would the nodes represent. In this paper, every node represents the state of the current *Congklak* board (how many seeds are in every small hole) and it is displayed as how many seeds are currently in *gudang*. Therefore, the root of the tree represents the initial condition (every small hole contains 7 seeds and every *gudang* has 0 seed) and every branch would represent the small hole chosen to get to that state.

Every chosen small hole would represent the path taken. Based on condition 7a in the game rules, "If the player ran out of seeds in self-*gudang*, the player may choose another non-empty small hole to restart the distribution". That means when this condition is met, choosing another small hole would increase the depth of the tree. Since BFS prioritize breadth over depth, the node that the path leads to should be searched later and therefore be put in the queue. Consider the following illustration (Fig. 6).

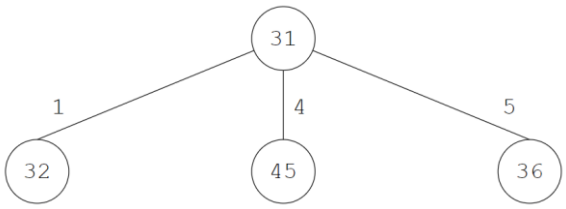

Fig. 6 Congklak BFS Tree Illustration 1
(source: writer's archive)

In the figure, initially there are 31 seeds in the player's *gudang*. The branch represents the small hole chosen, that means if the player chooses small hole 1, there would be 32 seeds in *gudang*, if the player chooses 4, there would be 45 seeds in *gudang*, and so on.

When a player chooses a small hole that leads to condition 7a, the node would branch into several other nodes based on the next small hole chosen. Consider the following illustration (Fig. 7).

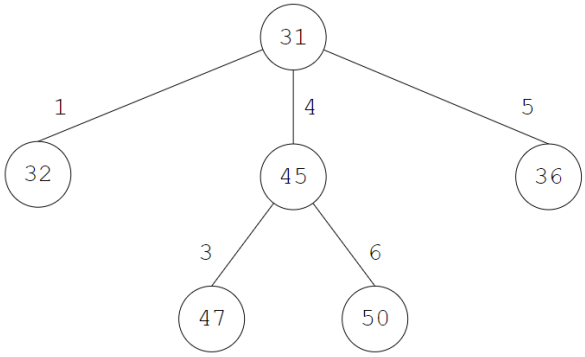

Fig. 7 Congklak BFS Tree Illustration 2
(source: writer's archive)

In the figure, if the player chooses small hole 4, the node results in having 45 seeds in *gudang* and two new branches. That means that path leads to condition 7a, meaning the player can choose another small hole. In this case, there are only two non-empty small holes available for choosing, which are small hole 3 and 6 that each leads to having 47 and 50 seeds in *gudang*. Notice that choosing small hole 6 results in 50 seeds in *gudang*, this means that this path secures victory for the player. The reconstructed path (another part of BFS) would be [path that leads to 31] – 4 – 6.

C. *Congklak BFS Algorithm*

Having established the rules and elements of *Congklak*, the following is the complete procedure of the BFS approach in winning *Congklak* in one turn.

1. Put the initial board state inside the queue.

2. If the queue is not empty, dequeue the first element of the queue. If the queue is empty, the solution is not found.
3. For every non-empty small hole available, calculate the number of seeds in *gudang* after distributing the seeds.
4. If the number of seeds exceeds 49, stop the search and go to step 6. If the number of seeds is 49 or below and condition 7a is fulfilled (the distribution ended in the player's *gudang*), enqueue the board state.
5. Repeat from step 2 until the stop condition in step 4 is fulfilled or the queue is empty.
6. Reconstruct the path taken to reach the winning state using any method.

Since the number of possible nodes can increase exponentially in base 7, the time complexity of this algorithm in big-o notation is less than $O(7^n)$, where n is the depth of the tree, because not all nodes can be expanded. In pseudocode, the algorithm would look something like this.

```
procedure BFSCongklak()
declaration
q : queue of board state
maxSeeds : integer
found : boolean
current : board state
algorithm
enqueue(q, initial state)
while (queue not empty) and (solution not found) do
   current = dequeue(q)
   for each non-empty small hole
      next = current.mainkanCongklak(small hole chosen)
      if (next does not lead to condition 7a) then
         if (seeds in next gudang > 49) then
            maxSeeds = seeds in next gudang
            found = true
         end if
      else
         enqueue(q, next)
      end if
   end for
end while

if (found) then
   // reconstruct path
   print(path and seeds in gudang)
else
   print("not found")
end if
```

Fig. 8 Congklak BFS Pseudocode
(source: writer's archive)

D. Implementation in Java

In order to test the correctness and effectiveness of the algorithm, a Java program has been developed. The program represents the state of the board as a class called 'Congklak' and utilizes Java ArrayList API to represent the queue. The program starts by initializing the board. At the end, the program will show the number of seeds in *gudang*, the path to get to that state, and the execution time of the algorithm. The following is a part of the source code that implements the BFS algorithm.

```
import java.util.ArrayList;
import java.util.List;


public class Main {
    public static void main(String[] args) {
        List<Congklak> queueCongklak = new
ArrayList<Congklak>();
         Congklak congklak = new Congklak();
         queueCongklak.add(congklak);
         int maxBiji = -1;
         Congklak maxCongklak = null;
         long startTime = System.currentTimeMillis();
         boolean found = false;

         while ((!queueCongklak.isEmpty()) && (!found)){
             Congklak currentCongklak =
queueCongklak.remove(0);
             for (int i = 1; i <= 7; i++){
                 if (!currentCongklak.isHoleEmpty(i)){
                     Congklak nextCongklak = new
Congklak(currentCongklak);
                     nextCongklak.playCongklak(i);
                     if (nextCongklak.isTurnOver()){
                         if
(nextCongklak.getSeedsInGudang() > 49){
                             maxBiji =
nextCongklak.getSeedsInGudang();
                             maxCongklak = nextCongklak;
                             found = true;
                             break;
                         }
                     } else {
                         queueCongklak.add(nextCongklak);
                     }
                 }
```

```
            }
        }


        long endTime = System.currentTimeMillis();
        System.err.println();
        System.out.println("Execution Time: " + (endTime - startTime) + " ms");
        System.out.println("Seeds in Gudang: " + maxBiji);
        maxCongklak.printRoute();
        maxCongklak.printBoard();
    }
}
```

Fig. 9 Congklak BFS in Java
(source: writer's archive)

IV. Results and Conclusion

The following figure is the result of running the Java program.

```
Execution Time: 103 ms
Seeds in Gudang: 54
Route: 1 2 7 7 1 3 1 4 7


    own Gudang                                  opponent's Gudang
[----------------+-------------------------+----------------]
|                |   |   |   |   |   |   |   |                |
|                | 8 | 2 | 1 | 7 | 0 | 4 | 0 |                |
|                |   |   |   |   |   |   |   |                |
|       54        |---|---|---|---|---|---|---|       0        |
|                |   |   |   |   |   |   |   |                |
|                | 2 | 3 | 4 | 5 | 0 | 8 | 0 |                |
|                |   |   |   |   |   |   |   |                |
[----------------+-------------------------+----------------]
```

Fig. 10 Result of BFS Search to Win Congklak
(source: writer's archive)

It has been found that it is very possible to win a *Congklak* game in just one turn. The first solution that the program found is 54 seeds in *gudang* reached by the following path: 1-2-7-7-1-3-1-4-7. Since BFS algorithm traverses the tree while prioritizing breadth, this path can also be considered as the shortest path to winning *Congklak* game in just one turn.

By slightly modifying the algorithm, an astonishing 92 seeds can be collected in *gudang* in just one turn using the following path: 1-7-4-1-6-7-4-3-1-1-4-1-4-7-2-1-5-2-3-6-1-5-5-7-1.

```
Execution Time: 1248598 ms
Seeds in Gudang: 92
Route: 1 7 4 1 6 7 4 3 1 1 4 1 4 7 2 1 5 2 3 6 1 5 5 7 1


    own Gudang                                  opponent's Gudang
[----------------+-------------------------+----------------]
|                |   |   |   |   |   |   |   |                |
|                | 0 | 1 | 0 | 3 | 0 | 0 | 0 |                |
|                |   |   |   |   |   |   |   |                |
|       92        |---|---|---|---|---|---|---|       0        |
|                |   |   |   |   |   |   |   |                |
|                | 0 | 0 | 1 | 0 | 1 | 0 | 0 |                |
|                |   |   |   |   |   |   |   |                |
[----------------+-------------------------+----------------]
```

Fig. 11 Most Amount of Seed Found
(source: writer's archive)

Besides the results, an attempt to use the algorithm to collect all 98 seeds in the game was made. However, it was discovered that the time complexity had increased exponentially. As a result, the attempt failed and the maximum number of seeds in *gudang* that can be achieved from the first turn is still unknown. In order to finish the search, it is suggested that: a better computer is used; multithreading is implemented; or a better algorithm is applied.

Acknowledgment

The writer firstly would like to give praise and thank Almighty God for His guidance in helping the writer finish this paper. Secondly, the writer would like to thank himself for continuing the topic that he had chosen for an unfinished paper in a previous course, IF2120 Discrete Mathematics and discovering new knowledges that were previously unknown. Thirdly, the writer would like to thank all lecturers of IF2211 Algorithm Strategies, especially Ir. Rila Mandala, M.Eng., Ph.D. and Monterico Adrian, S.T., M.T. as lecturers of K03 Jatinangor for teaching the writer the diverse algorithm strategies that exists in the informatics world. Lastly, the writer would like to thank all other parties indirectly involved in encouraging the writer to finish this paper.

References

[1] Munir, R. (2024). *Breadth/Depth First Search (BFS/DFS) (Bagian 1)*. Retrieved June 11, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/BFS-DFS-2021-Bag1-2024.pdf

[2] Munir, R. (2024). *Breadth/Depth First Search (BFS/DFS) (Bagian 1)*. Retrieved June 11, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/TugasMakalah2024.pdf

[3] Kids, S. (2022, August). *Bermain Congklak | Permainan Tradisional Anak Indonesia | Video Belajar Anak | Video Edukasi*. [Video]. Retrieved June 11, 2024, from https://www.youtube.com/watch?v=JJAxKqjJHcQ

[4] Indonesia. Ministry of Education, Culture, Research, and Technology. (2012). *Congklak*. Retrieved June 11, 2024, from https://warisanbudaya.kemdikbud.go.id/?newdetail&detailCatat=2196

[5] Pratama, S. (2023, January 5). *Masih Ingat Cara Main Congklak? Di Balik Permainan Ini Ternyata Ada Maknanya, Lho!*. Retrieved June 11, 2024, from https://www.kompas.tv/video/365326/masih-ingat-cara-main-congklak-di-balik-permainan-ini-ternyata-ada-maknanya-

lho#:~:text=Congklak%20merupakan%20permainan%20yang%20berasal,menyebar%20ke%20negara%2Dnegara%20Asia.

[6] Sihite, N. S. (2023, December 11). *Constructing A Decision Tree to Find the Best Starting Hole in Indonesian Traditional Game 'Congklak': Maximizing the Number of Seed in Gudang by The End of The Turn*. Retrieved June 11, 2024, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/Makalah2023/Makalah-Matdis-2023%20(144).pdf

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.

Jatinangor, 12 Juni 2024

Nicholas Reymond Sihite – 13522144