Exploring Optimization Problems in Cayley Trees: Theory and Experiments

A project report submitted to

Indian Institute of Engineering, Science and Technology in partial fulfillment

for the award of the degree of Bachelor of Technology

in Information Technology

By:

Pravin Kumar (2020ITB093)

Panyam Hema (2020ITB103)

Ramavath Nokshith (2020ITB086)

Under the supervision of

Dr. Sukanta Das

Associate Professor

IIEST Shibpur, India

Department of Information Technology

Indian Institute of Engineering, Science and Technology, Shibpur May, 2024

DEPARTMENT OF INFORMATION TECHNOLOGY
INDIAN INSTITUTE OF ENGINEERING, SCIENCE AND TECHNOLOGY SHIBPUR,
HOWRAH - 711103, INDIA

CERTIFICATE

This is to certify that the project report entitled **"Exploring Optimization Problems on Cayley Trees: Theory and Experiments"** submitted by **Pravin Kumar**(2020ITB093), **Panyam Hema** (2020ITB103) and **Ramavath Nokshith**(2020ITB086) to Indian Institute of Engineering, Science and Technology towards partial fulfillment of requirements for the award of degree of Bachelor of Technology in Information Technology is a record of bonafide work carried out by them under my supervision. This dissertation, in my opinion, is worthy of consideration for the purpose for which it is submitted and it fulfills the requirements of the regulations of this Institute. The results incorporated in this dissertation are original and have not been submitted to any University or Institute for the award of any Degree or Diploma.

(Dr. Sukanta Das)
Associate Professor
Dept. of Information Technology
Indian Institute of Engineering Science and Technology,
Shibpur, Howrah, West Bengal, India –711103

(Dr. Prasun Ghosal)
Associate Professor & Head
Dept. of Information Technology
Indian Institute of Engineering Science and Technology,
Shibpur, Howrah, West Bengal, India –711103

Contents

Abstract

This report delves into the study of operations on a Cayley tree of order two, examining its properties and two optimization problems. One of these optimization problems involves finding the maximum and minimum values across the distribution of natural numbers at each node of the Cayley tree. However, with traditional array implementations, it takes linear time complexity (O(n)) to determine the maximum and minimum values in a sequence of natural numbers. To improve its time complexity, we introduced a novel model that incorporates state, memory, and flag registers within each tree node. Utilizing the in-computing memory technique, each node updates sends, and receives the current state from its neighbours. Consequently, the maximum and minimum values are obtained at the root node within O(log n) time complexity by using a parallel computing mechanism. Our proposed model was validated through C++ code experimentation. This report contributes to the advancement of memory optimization techniques and new approaches for solving optimization problems, which can aid in discovering new challenges.

Keywords: Cayley tree, memory optimization, parallel computing, Code experimentation, optimization problems.

Acknowledgements

First, we express my heartfelt gratitude for the support and resources provided by the Indian Institute of Engineering Science and Technology, Shibpur, for my research during the tenure of this research project on optimizing problems in Cayley trees. We are grateful that we had the opportunity to experience an academic journey in IIEST, Shibpur that lasted four years. These four years have been an eye-opening experience for us and we consider ourselves fortunate to have learnt lessons which shall always be useful for us.

We'd like to express my sincere gratitude to my supervisor and mentors Dr Sukanta Das, Associate Professor, Department of Information Technology, Indian Institute of Engineering Science and Technology (IIEST), Shibpur, for his continuous support and help at all stages of preparing this report. we are also thankful for his patience, motivation, enthusiasm, and immense knowledge. His guidance always helped me in writing this thesis. I could not imagine having a better advisor or mentor for my project. During this journey, I learned many things from him, especially how to be disciplined in research and life.

We are also grateful to all the faculty in the Department of Information Technology, IIEST, Shibpur. The knowledge imparted through the courses as part of the B.Tech

programme will be useful in preparing us for future technological challenges. We would also like to thank everyone involved in maintaining and taking care of all the laboratories of the department. Having access to such a wonderful laboratory made life much easier for all of us Undergraduate students.

We'd also like to express my deep respect and appreciation to Mr Subrata Paul, PhD Scholar, Department of Information Technology, Indian Institute of Engineering Science and Technology (IIEST), Shibpur, for their valuable suggestions and advice, which helped me understand the fundamental concepts of this topic, including the ideas behind introducing a new model, scientific methodology, and their immense support throughout this project.

Finally, I'd like to thank my team members, Ramavath Nokshith and Panyam Hema, for their encouragement and support throughout this project.

Pravin Kumar (2020ITB093)

Panyam Hema (2020ITB103)

Ramavath Nokshith (2020ITB086)

Acronyms

CT Cayley Tree

CPU Central Processing Unit

O(log n) Logarithmic Time Complexity

CSP Cayley Structure Properties

Max Maximization Problem

Min Minimization Problem

C++ A programming language

Order number of children

Chapter 1

Introduction

1.1 Introduction to Cayley Tree

A Cayley tree is also known as a Bethe lattice (5). It is a connected, acyclic graph where each node is connected to a fixed number of child nodes, forming a hierarchical structure. In a Cayley tree of order η, each node has exactly η number of children. and only the root node has $\eta + 1$ number of children. This structure is utilized in various fields of mathematics and computer science due to its regular and recursive properties, making it ideal for modelling hierarchical systems and efficient information propagation.

A variant of the Cayley Tree Framework, in which each cell has an attached memory and an additional processing unit, was proposed in (3) and Chapter 6 in (6), which enhances the memory computational capabilities.

Our proposed model, similar to these variants, is constructed based on the specific Cayley Tree of Order $\eta = 2$ (1; 2), Where each node is equipped with a dedicated memory unit and a flag register. The model is specifically designed for the efficient

resolution of optimization problems, including maximum and minimum problems, through in-memory computation, as mentioned in (4). Additionally, it is similar to the research conducted on searching problems over the Cayley tree, as discussed in Chapter 6 of (6).

We consider a specific Cayley tree model or framework of order 2 in which data elements (X) are distributed across the memory of nodes. The main task is to solve the Maximum and minimum problems. In other words, determining the maximum or minimum value within a finite set of natural numbers (X). If the value is present within the set, the model returns output as a maximum or minimum value and this value is present at the root of the Cayley tree.

In subsequent sections, we delve into the details of the Cayley tree architecture and our proposed model. Chapter 2 provides a comprehensive overview of the proposed model, while Chapter 4 covers the implementation of In-Memory computation within this framework or architecture.

1.2 Definition of Cayley Tree

Definition 1: *A Cayley tree is a tree in which each non-leaf vertex has a constant number of branches η (the order of the tree). The Cayley tree κ_η of order $\eta \geq 1$ is an infinite tree from each vertex with exactly $\eta + 1$ edges. The $\kappa_\eta = (V, E, \nu)$, where V is the set of vertices of κ_η, E is the set of edges and ν is the incidence function associating each edge $e \in E$ with its endpoints $v_{neighbors} \in V$.*

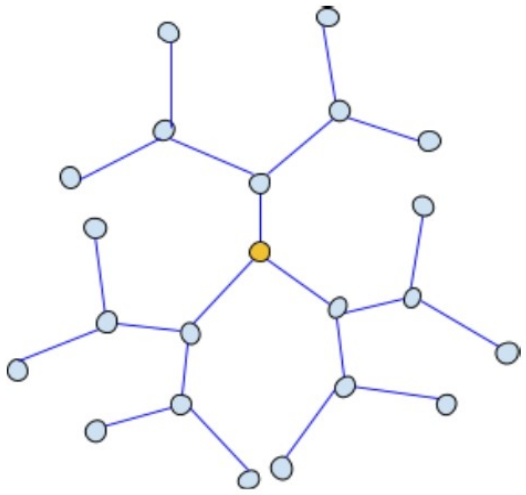
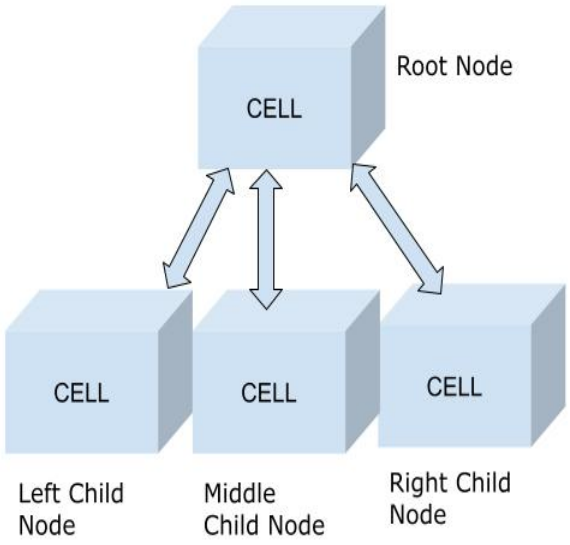


Figure 1.1: Structure of Cayley Tree of height h = 3

In the current work, we focus on a special case of the Cayley tree of order $\eta = 2$, where the root node has three children, and each subsequent level of node has two children, except the leaf node. The Cayley tree $\eta = 2$ is as follows:
the total no of n nodes at a given height h is,

$$n = \begin{cases} (3 \times 2^h) - 2 & \text{if } h \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.1}$$

The proof of this equation is provided in Appendix A.1.



[

Figure 1.2: Structure of Root Cell

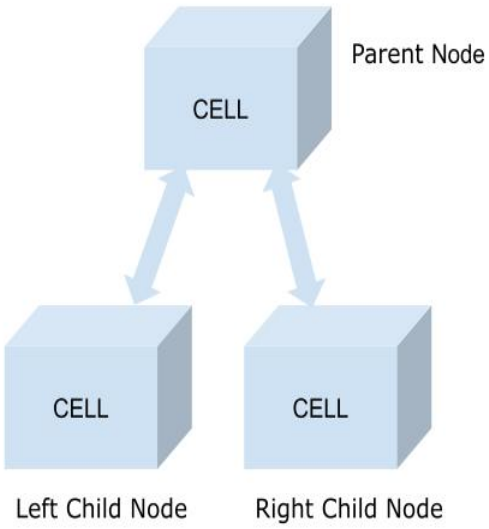


Figure 1.3: Structure of Intermediate Cell

A Cayley tree with a height of h=3 is shown in Figure 1.1, with each vertex representing a cell in the CA. The total number of cells in the tree is n=22, as calculated using Equation 1.1. The root node, yellow in colour, occupies the central position within the tree. Figure 6.1b illustrates the neighbourhood arrangement and shows how each cell in the tree structure is connected to the others.

It's important to note that this model adopts null boundary conditions, which assumes that a leaf node has the default state 0 due to missing neighbours.

Chapter 2

Proposed Computational Model

2.1 Model Framework

The computational model detailed in Chapter 1.2 resembles a network structure, wherein each cell serves as a node in a Cayley tree framework.

In our proposed model, each node in the system comprises three essential components: the state, a flag register, and a memory element. These elements interact through three functions: `send()`, `recv()`, and `g()`. now, We will explore the specifics of each component and its role thoroughly:

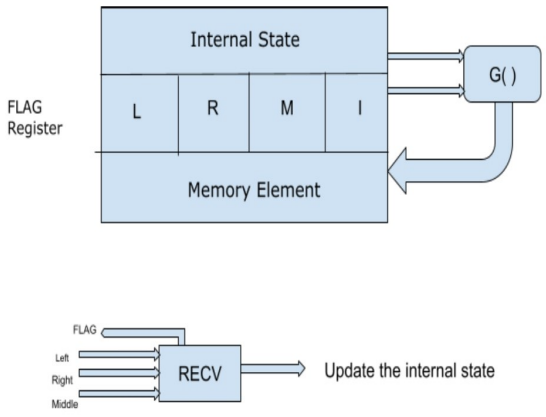


Figure 2.1: Structure of Root Node

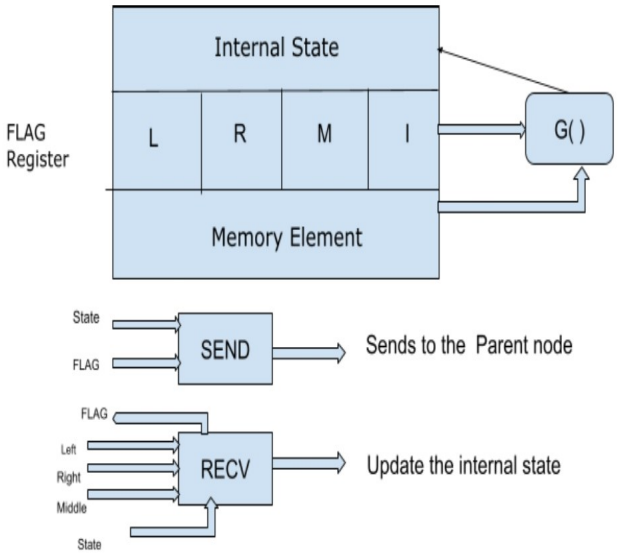


Figure 2.2: Structure of Intermediate Node

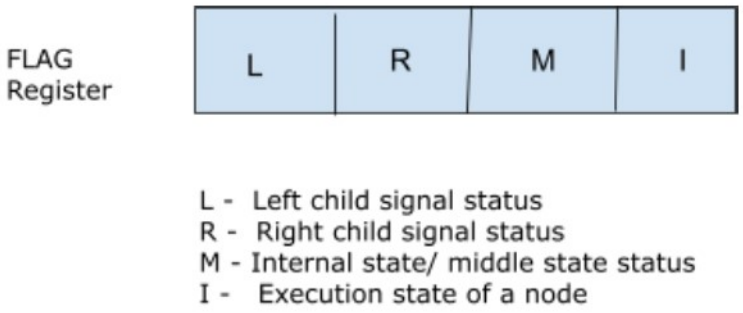


Figure 2.3: Structure of FLAG Register

2.1.1 Node Components

1. **State:** The state represents the current condition or status of the node in the tree. It served as the primary data entity manipulated during computation.

2. **Flag Register:** The flag is a four-bit register associated with each node. Each bit in a register has a distinct function.

    - Flag[0]: These bits represent the incoming signals from the left child. If it is set, the node receives a signal from the left child node. otherwise, it is discarded.
    - Flag[1]: These bits represent the incoming signals from the right child, and if it is set, the node receives the signal from the right child.
    - Flag[2]: These bits represent the incoming signals from the middle child (if the node is the root node) and the current internal state (for other nodes).

It receives a signal if it is set.

- Flag[3]: This bit indicates whether the node is currently executing a computation or not. When it is set, it signifies that the node is active.

3. **Memory Element:** The memory element is used to store data for computational purposes. This model utilizes a 32-bit size of memory element. Here, the left-shift register served as our memory element. A left-shift register reduces the need for counter registers, enabling efficient data handling.

   Initially, the root node memory contains the value 1, and all other node's memory contains the value or information.

   The initial value of 1 in the memory element of the root node acts as a termination signal, stopping the execution once the entire memory of the root node has been processed.

2.1.2 Node Functions

1. **send():** This function is responsible for transmitting the internal state of the node. It takes inputs from the current internal state and flag register and sends them to its parent node.

2. **recv():** The `recv()` function receives input from neighbouring or child nodes, such as the left and right child node states, the flag register, and the middle child states(if the node is root node) or the current internal state (for other nodes). Based on the given input, the node determines its next state and updates its internal state accordingly.

The `g()` function binds the memory location to the node's internal state. The root node takes the current internal state as the input and stores it in the memory element. For other nodes, it uses the memory element as the input and updates the current internal state accordingly.

In the proposed model, each clock pulse initiates a synchronized computational cycle among all nodes. The nodes receive signals from their child nodes through Recv(), update their internal states using the `g()` function and flag register, and then transmit their updated states to the parent node via `send()`. This cycle continues until the entire memory has been processed completely and the root node has achieved a maximum or minimum value.

2.2 Model Simulations

In this study, we explore the application of the proposed model to optimization problems, particularly focusing on scenarios involving minimum and maximum value calculations. The simulation process starts with the initial configuration provided to the system. As the computation progresses, each node iteratively updates its state based on signals received from its child nodes and sends them to its parent node.

Execution starts at the root node and traverses the network hierarchy until it reaches the leaf nodes. Subsequently, the leaf nodes transmit signals upward, passing through intermediate nodes, until reaching the root node.

Upon completion of the computation cycle, the root node has a minimum or maximum value depending on the applied optimization algorithm. This value represents the optimized solution obtained from the initial configuration.

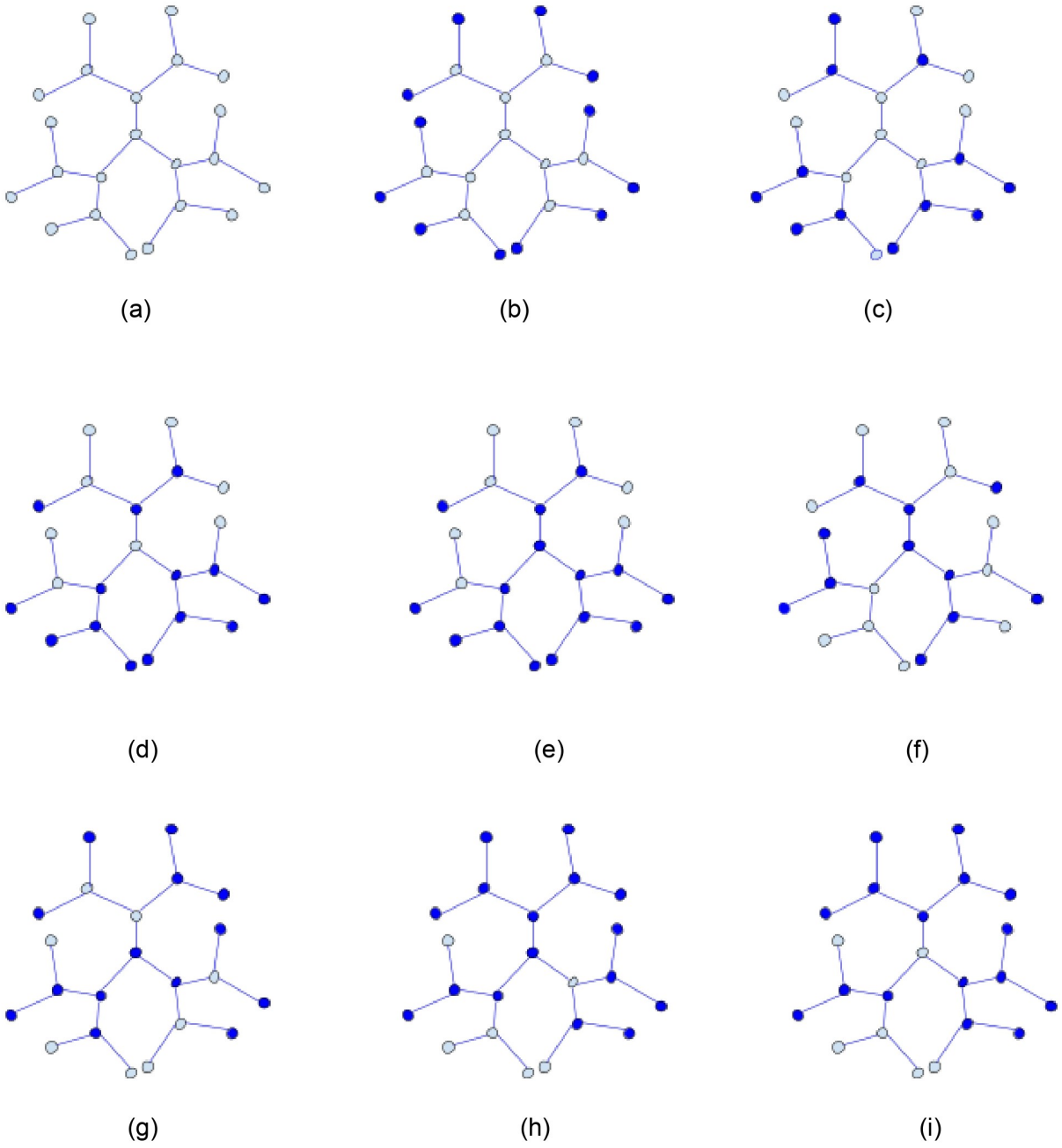


Figure 2.4: initial configuration to final configuration of the Cayley tree, where white node representing node state is 0, blue node representing node state is 1 and at the final configuration, the maximum value is stored at the root node.

Chapter 3

In-Memory Computing for Optimization Problem

The basic objective of in-memory computing is to minimize CPU overload by performing computations with large amounts of data directly in memory. in this section, we present the implementation of in-memory computing using the computational model introduced in Chapter 2, which is specifically designed for solving optimization problems involving finding the maximum and minimum values within sequences of natural numbers. Upon completion of the computation, the final value is stored at the self-node.

3.1 Maximization Problem

In the context of the maximum value problem, we distribute the sequence of natural numbers to each node of the Cayley tree, except for the self-node, which initially has a value of 1.

1. **Computation at the self node:** The self node has three children. the maximum value is determined between the signals received from the three child

nodes using flag registers and comparison operations. The maximum value is stored in the self node's memory and automatically shifted to the left to accommodate the next value (since a left shift register is used as a memory element). When the memory reaches its capacity, the computation halts at the self node, and finally, the self node contains the maximum value.

2. **Computation at the leaf node:** leaf node has no child node, so by default its child node state is treated as 0. Now, we use the comparison function fun(), which takes the child state and memory element as input and returns a new state based on the comparison result, which is stored at the current state of the leaf node. after updating the current state, it sends the state to its parent node.

3. **Computation at the intermediate node:** Intermediate nodes have two children and update their current state based on their children's states and their memory element. The updated state is sent to the parent node via the **send()** function.

The signal is sent to the parent node through the **Send()** function. Signals are received using **Recv()**. The state is updated using function g(). The state and memory values are compared using fun(). Then, the flag register is updated.

Algorithm 1 State Sending Algorithm Send()

```
1: Local variable clock ← 0
2: procedure SND(Self, flag):
3:     clock ← clock + 1
4:     if flag[3] then
5:         return Sell.State
6:     end if
7: end procedure
```

Algorithm 2 State Receiving Algorithm Recv()

```
1: procedure RECV(Self, Left, Right, Middle, Memory):
2:     s ← 0
3:     m ← 0
4:     l ← 0
5:     r ← 0
6:     if Middle ≠ NULL then                                  ▷ for self Node
7:         m ← Middle.state
8:         l ← Left.state
9:         r ← Right.state
10:        if flag[3] then
11:            Self.state ← f(s, m, l, r)
12:            Gself(Self.State)
13:        end if                                             ▷ for Intermediate Node
14:    else if Left ≠ NULL and Right ≠ NULL and Middle = NULL then
15:        Gother(Self, Memory)
16:        l ← left.state
17:        r ← right.state
18:        r ← self.state
19:        if flag[3] then
20:            Self.state ← f(s, m, l, r)
21:        end if
22:    else                                                   ▷ for Leaf Node
23:        if flag[3] then
24:            Gother(Self, Memory)
25:        end if
26:    end if
27: end procedure
```

Algorithm 3 Computation processing function algorithm of self node g()

```
1: procedure GSELF(Self, flag, memory):
2:     if flag[3] then
3:         Memory ← Sell.state
4:         Left shift (Memory)
5:     end if
6: end procedure
```

Algorithm 4 Computation processing function algorithm of other nodes g()

```
1: procedure GOTHER(Self, flag, memory):
2:     if flag[3] and flag[2] then
3:         Self.state ← Memory
4:         Left shift (Memory)
5:     end if
6: end procedure
```

Algorithm 5 Comparison Operation Function Algorithm fun()

```
1: procedure FUN(self, l, m, r)
2:     if l = 0 and r = 0 and s = 0 then
3:         self.state ← 0                                  ▷ Set self state to 0
4:         return
5:     end if
6:     if (l = 0 and r = 0 and self.flag[2]) or (r = 0 and s = 0 and self.flag[0])
   or (s = 0 and l = 0 and self.flag[1]) then
7:         self.state ← 0                                  ▷ Set self state to 0
8:         return
9:     end if
10:    if (l = 0 and self.flag[1] and self.flag[2]) or (r = 0 and self.flag[0] and
   self.flag[2]) or (s = 0 and self.flag[0] and self.flag[1]) then
11:        self.state ← 0                                  ▷ Set self state to 0
12:        return
13:    end if
14:    if l = 0 and not self.flag[0] then
15:        self.flag[0] ← 1                                ▷ Set flag[0] to true
16:    end if
17:    if r = 0 and not self.flag[1] then
18:        self.flag[1] ← 1                                ▷ Set flag[1] to true
19:    end if
20:    if s = 0 and not self.flag[2] then
21:        self.flag[2] ← 1                                ▷ Set flag[2] to true
22:    end if
23:    if self.flag[0] = 0 then
24:        self.state ← l                                  ▷ Set self state to l
25:    end if
26:    if self.flag[1] = 0 then
27:        self.state ← r                                  ▷ Set self state to r
28:    end if
29:    if self.flag[2] = 0 then
30:        self.state ← s                                  ▷ Set self state to s
31:    end if
32: end procedure
```

3.1.1 Memory selection as the left shift register

We use a left-shift register as our memory configuration. With each clock pulse, the memory values shift one position to the left, using only the initial data. Using this method, extra memory consumption is decreased and counter registers are no longer required. However, a significant challenge arises in memory execution, because once it starts it cannot be stopped. To address this, we adopt a strategy in which the last value of the memory is replaced by zero. As the memory value shifts to the left, the last value eventually becomes zero, indicating completion of memory execution and conservation of computational resources. Despite its efficiency, this method poses a risk of losing valuable information, because while all nodes except the self-node discard their stored values, the self-node retains the maximum values which are required for our optimization problem

3.2 Minimization Problem

This is similar to the maximum problem approach discussed in 3.1, in the context of the minimum value problem, we distribute the sequence of natural numbers to each node of the Cayley tree, except for the self-node, which initially has a value of 1.

The self node receives signals from its three child nodes and determines the minimum value using flag registers and comparison operations among the received signals.

This minimum value is then stored in the memory of the self-node, which is automatically shifted to the left to accommodate the next value. Once the memory is full, the computation halts at the self node and finally contains the minimum value.

For leaf nodes with no child nodes, the calculation is based on treating their child state as 1 and comparing it with the memory value. The leaf node updates its current state based on the comparison result using fun() and transmits the updated state to its parent node.

Intermediate nodes update their current state based on the states of their two child nodes and their memory elements. The updated state is sent to the parent node using the `send()` function.

3.2.1 Memory selection as the left shift register

It is similar to the memory selection in the maximization problem. but only the change is to address the issue of the continuously shifting memory value left, we adopt a strategy in which the last value of the memory is replaced by one. As the memory value shifts to the left, the last value eventually becomes one, indicating completion of memory execution and conservation of computational resources.

In this problem, the algorithm for `Send()`, `Recv()`, `g()` for the self node, and `g()` for other nodes remain the same as discussed in Algorithms 1, 2, 3, and 4. The only change is in the comparison operation function `fun()`, which is defined as follows:

Algorithm 6 Comparison Operation Function Algorithm fun()

```
1: procedure FUN(self, l, s, r)
2:     if l = 1 and r = 1 and s = 1 then
3:         self.state ← 1                                  ▷ Set self state to 1
4:         return
5:     end if
6:     if (l = 1 and r = 1 and self.flag[2]) or (r = 1 and s = 1 and self.flag[0])
   or (s = 1 and l = 1 and self.flag[1]) then
7:         self.state ← 1                                  ▷ Set self state to 1
8:         return
9:     end if
10:    if (l = 1 and self.flag[1] and self.flag[2]) or (r = 1 and self.flag[0] and
   self.flag[2]) or (s = 1 and self.flag[0] and self.flag[1]) then
11:        self.state ← 1                                  ▷ Set self state to 1
12:        return
13:    end if
14:    if l = 1 and not self.flag[0] then
15:        self.flag[0] ← 1                                ▷ Set flag[0] to true
16:    end if
17:    if r = 1 and not self.flag[1] then
18:        self.flag[1] ← 1                                ▷ Set flag[1] to true
19:    end if
20:    if s = 1 and not self.flag[2] then
21:        self.flag[2] ← 1                                ▷ Set flag[2] to true
22:    end if
23:    if self.flag[0] = 0 then
24:        self.state ← l                                  ▷ Set self state to l
25:    end if
26:    if self.flag[1] = 0 then
27:        self.state ← r                                  ▷ Set self state to r
28:    end if
29:    if self.flag[2] = 0 then
30:        self.state ← s                                  ▷ Set self state to s
31:    end if
32: end procedure
```

Chapter 4

Model Experimentation

4.1 Experimental Procedures For Max

The model is run with 22 elements (n = 22), X = 14, 2, 13, 11, 9, 11, 2, 13, 10, 2, 3, 8, 6, 5, 9, 2, 1, 7, 8, 2, 12, 7. We consider a Cayley tree of height h = 4 and order $\eta = 2$, where the number of nodes in the tree is 22 (see Eq. 1.1). The elements are distributed over nodes, and finally, the root node contains a maximum value of 14.

The simulation below illustrates the detailed execution, where Fig. (i) to (ix) shows the structure of the tree and describes each node execution. It depicts the initial configuration to the final configuration when the root node contains the maximum value.

4.1.1 Algorithm

You can find the code of the maximization problem in C++ on GitHub at the following link: Code Repository.

Algorithm 7 Computation at the Root Node

```
1: procedure COMPUTE_ROOT
2:     Receive signals from child nodes
3:     Determine maximum value using comparison operations
4:     Store maximum value in memory
5:     Shift memory to the left
6:     Repeat until memory is full
7: end procedure
```

Algorithm 8 Computation at the Intermediate Node

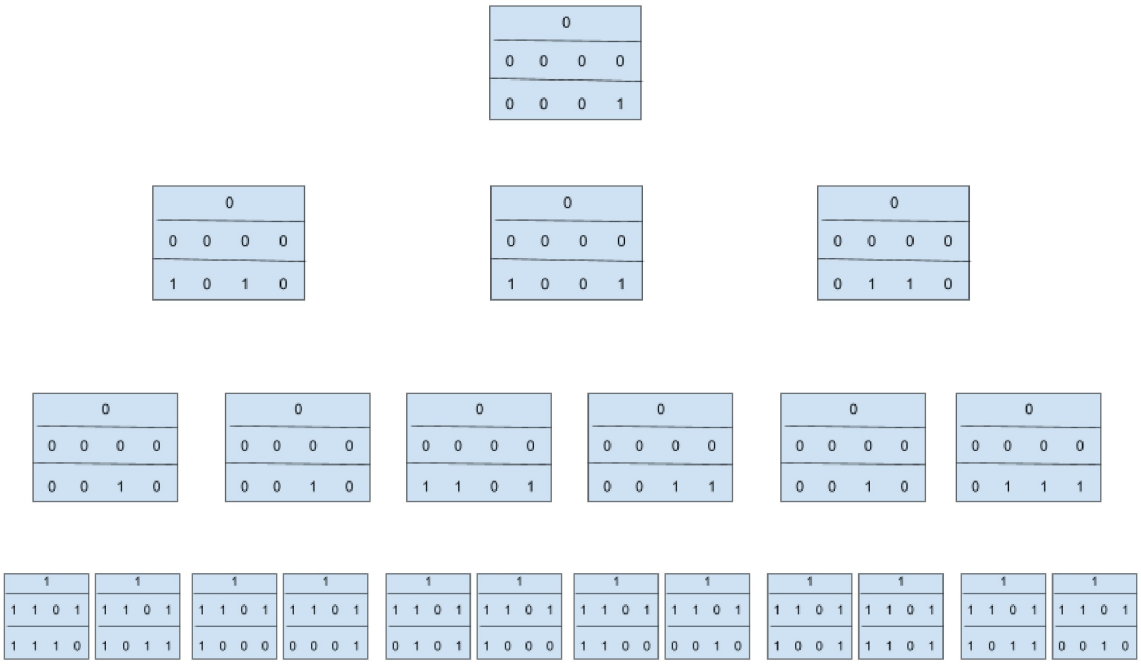
```
1: procedure COMPUTE_INTERMEDIATE
2:     Use child node states and memory to update the current state
3:     Transmit updated state to the parent node
4: end procedure
```

Algorithm 9 Computation at the Leaf Node

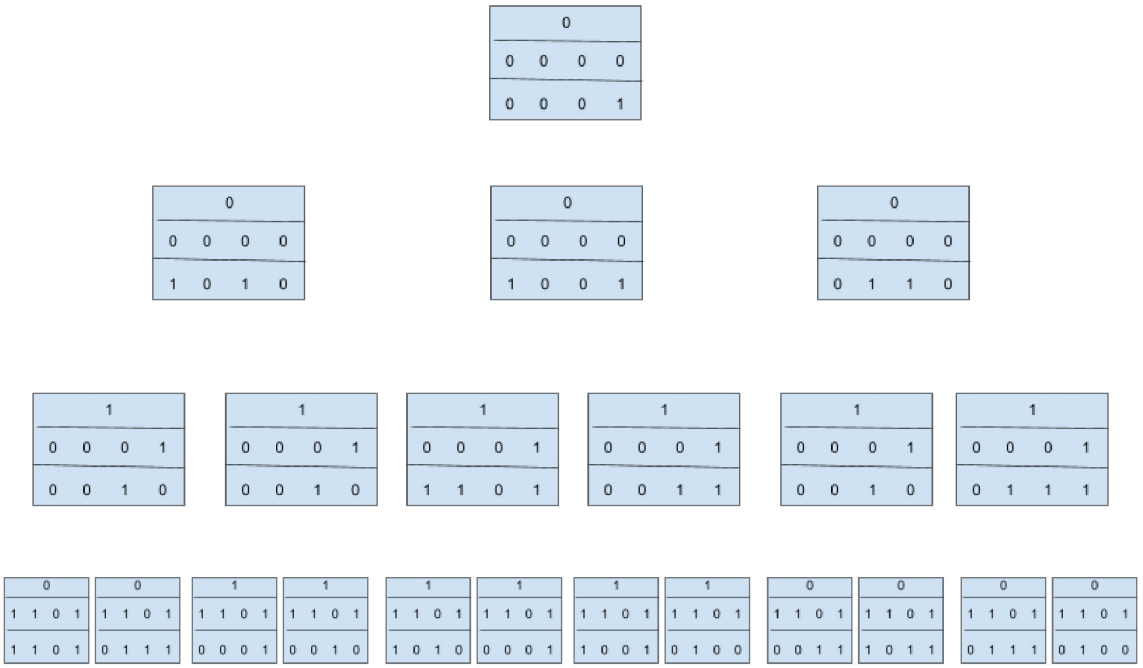
```
1: procedure COMPUTE_LEAF
2:     Treat child state as 0
3:     Compare with the memory value
4:     Update current state based on comparison
5:     Transmit updated state to the parent node
6: end procedure
```

4.1.2 simulation

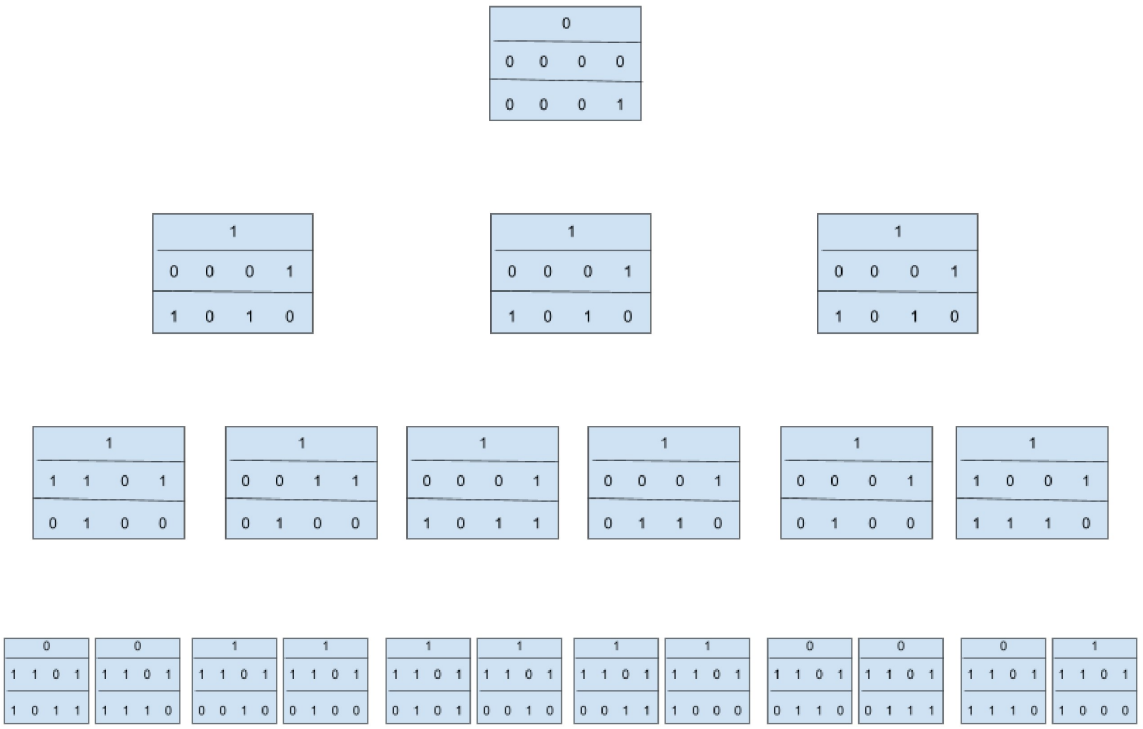
Figure 2.1 shows the structure of a node of the Cayley tree as described in section (2.1).


0
0 0 0 0
0 0 0 1
0
0 0 0 0
1 0 1 0
0
0 0 0 0
1 0 0 1
0
0 0 0 0
0 1 1 0
0
0 0 0 0
0 0 1 0
0
0 0 0 0
0 0 1 0
0
0 0 0 0
1 1 0 1
0
0 0 0 0
0 0 1 1
0
0 0 0 0
0 0 1 0
0
0 0 0 0
0 1 1 1
1
1 1 0 1
1 1 1 0
1
1 1 0 1
1 0 1 1
1
1 1 0 1
1 0 0 0
1
1 1 0 1
0 0 0 1
1
1 1 0 1
0 1 0 1
1
1 1 0 1
1 0 0 0
1
1 1 0 1
1 1 0 0
1
1 1 0 1
0 0 1 0
1
1 1 0 1
1 0 0 1
1
1 1 0 1
1 1 0 1
1
1 1 0 1
1 0 1 1
1
1 1 0 1
0 0 1 0

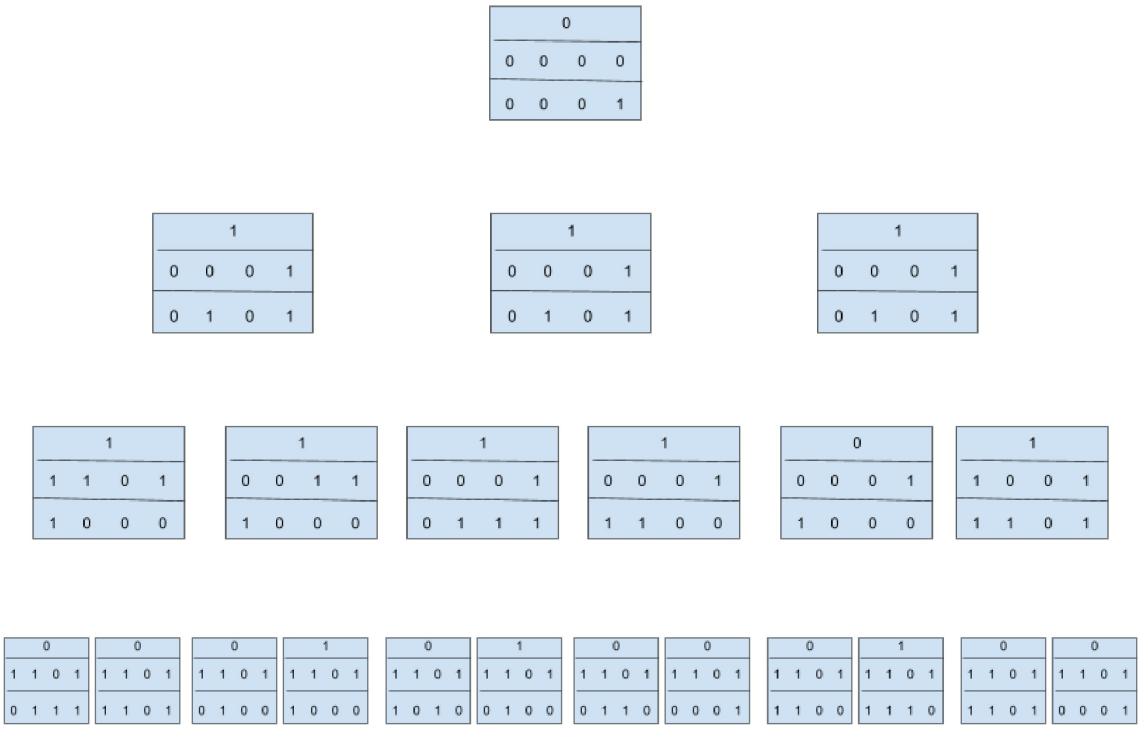
(i)



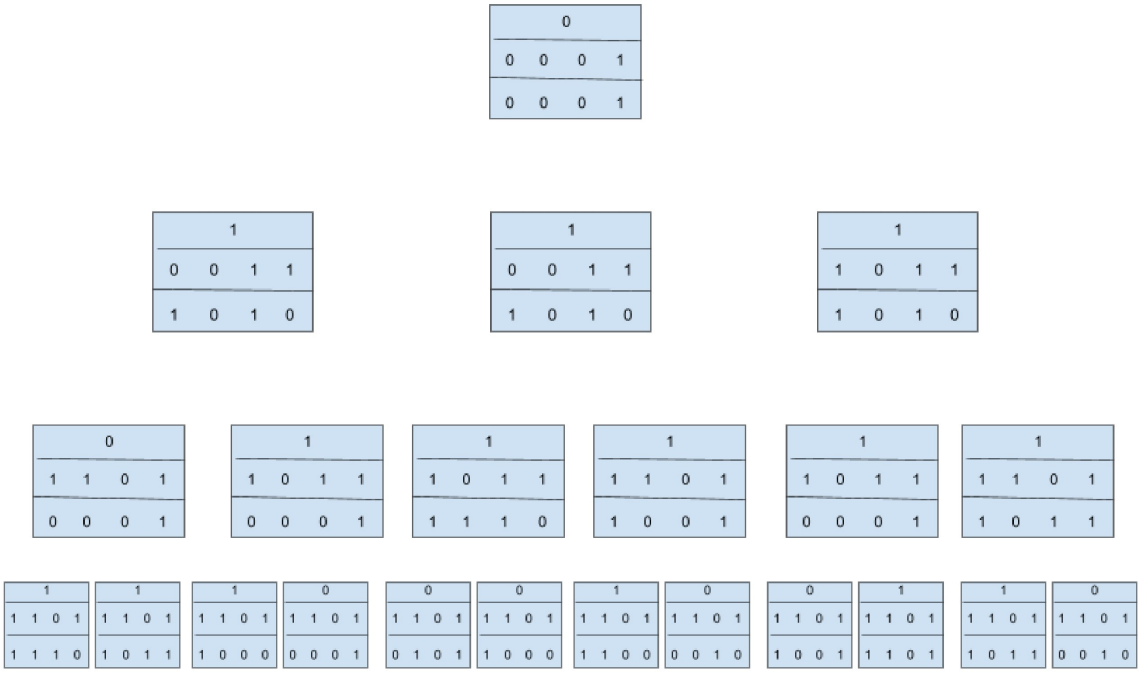
(ii)



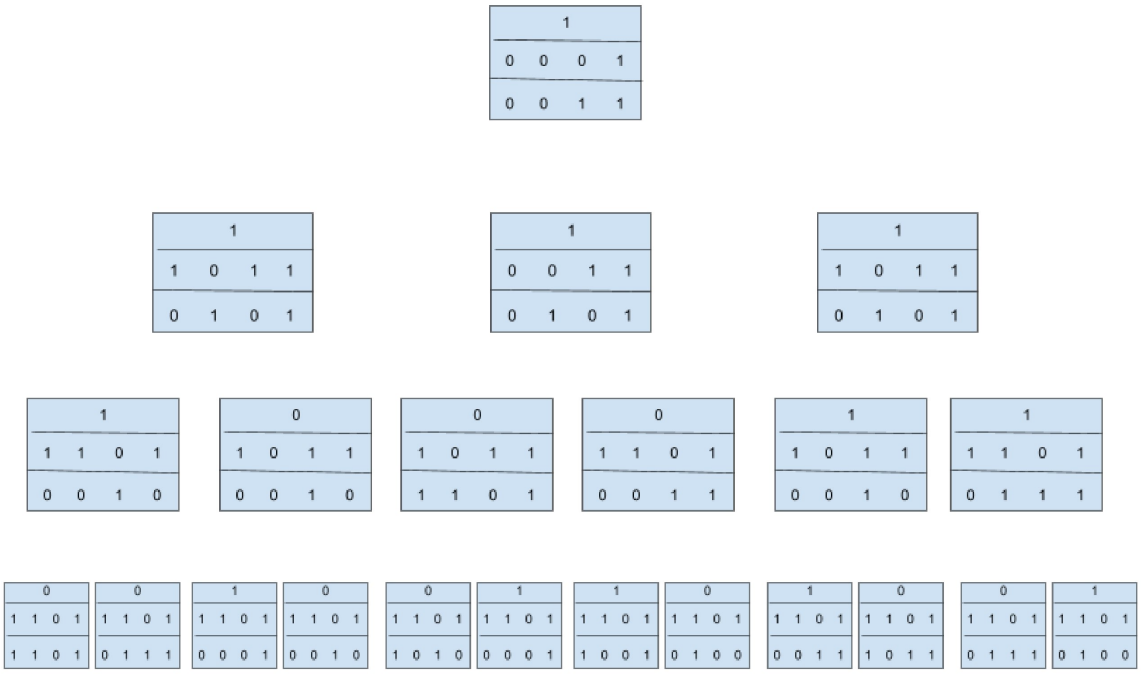
(iii)


0
0 0 0 0
0 0 0 1
1
0 0 0 1
0 1 0 1
1
0 0 0 1
0 1 0 1
1
0 0 0 1
0 1 0 1
1
1 1 0 1
1 0 0 0
1
0 0 1 1
1 0 0 0
1
0 0 0 1
0 1 1 1
1
0 0 0 1
1 1 0 0
0
0 0 0 1
1 0 0 0
1
1 0 0 1
1 1 0 1
0
1 1 0 1
0 1 1 1
0
1 1 0 1
1 1 0 1
0
1 1 0 1
0 1 0 0
1
1 1 0 1
1 0 0 0
0
1 1 0 1
1 0 1 0
1
1 1 0 1
0 1 0 0
0
1 1 0 1
0 1 1 0
0
1 1 0 1
0 0 0 1
0
1 1 0 1
1 1 0 0
1
1 1 0 1
1 1 1 0
0
1 1 0 1
1 1 0 1
0
1 1 0 1
0 0 0 1

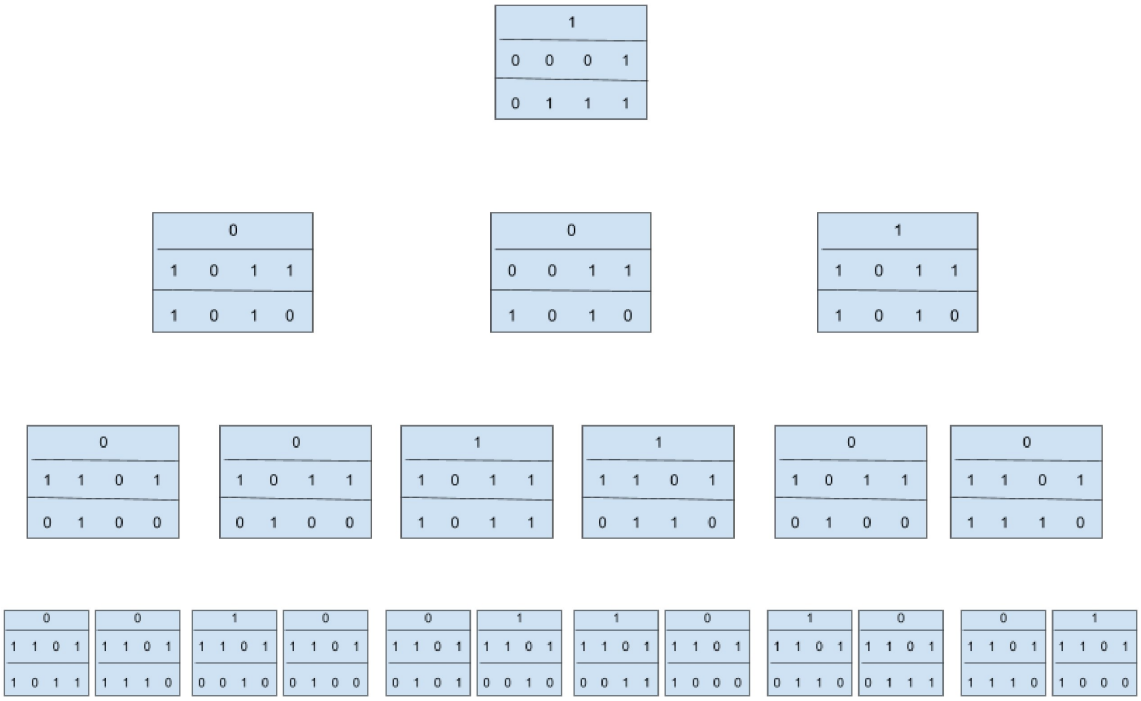
(iv)



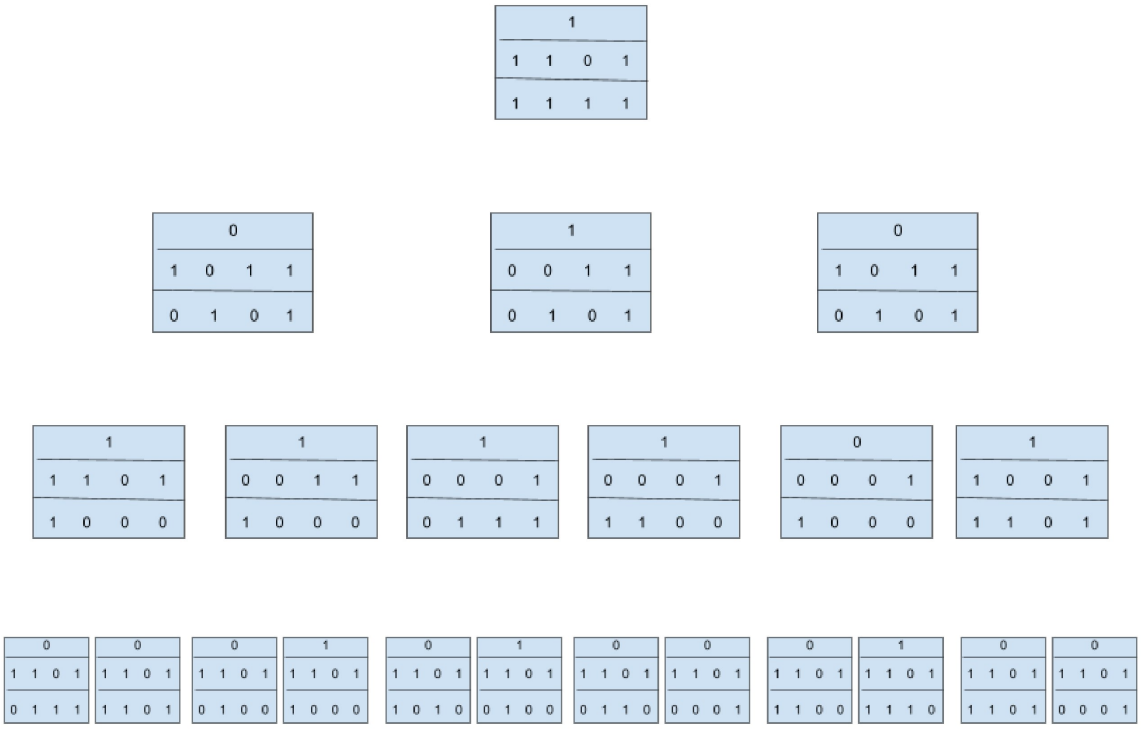
(v)



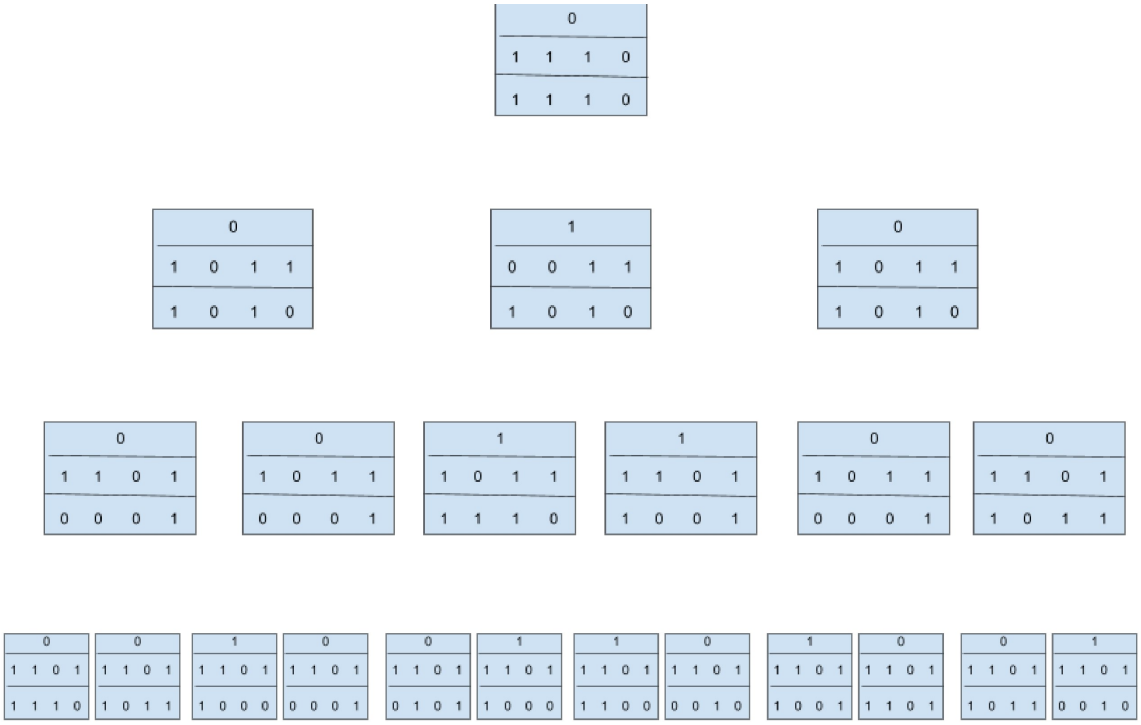
(vi)



(vii)


1
1 1 0 1
1 1 1 1
0
1 0 1 1
0 1 0 1
1
0 0 1 1
0 1 0 1
0
1 0 1 1
0 1 0 1
1
1 1 0 1
1 0 0 0
1
0 0 1 1
1 0 0 0
1
0 0 0 1
0 1 1 1
1
0 0 0 1
1 1 0 0
0
0 0 0 1
1 0 0 0
1
1 0 0 1
1 1 0 1
0 1 1 1
0 1 1 0 1
1 1 0 1
0 1 0 0
0 1 1 0 1
1 0 0 0
0 1 1 0 1
1 0 1 0
1 1 0 1
0 1 0 0
0 1 1 0 1
0 1 1 0
0 1 1 0 1
0 0 0 1
0 1 1 0 1
1 1 0 0
0 1 1 0 1
1 1 1 0
0 1 1 0 1
1 1 0 1
0 1 1 0 1
0 0 0 1

(viii)



(ix)

Note: The arrangement of elements in the array does not affect the flow of the scheme. No matter if the elements in the array are already sorted, reverse sorted, or randomly placed, the scheme works the same for all these cases, and thus the time complexity for all such cases is the same, O(log n).

4.2 Experimental Procedures For Min

Similar to the Max procedures, This model is run with 22 elements (n = 22), X = 14, 2, 13, 11, 9, 11, 2, 13, 10, 2, 3, 8, 6, 5, 9, 2, 1, 7, 8, 2, 12, 7. We consider a Cayley tree of height h = 4 and order $\eta = 2$, where the number of nodes in the tree is 22 (see Eq. 1.1). The elements are distributed over nodes, and finally, the root

node contains a minimum value of 1.

4.2.1 Algorithm

You can find the code of the minimization problem in C++ on GitHub at the following link: Code Repository.

Algorithm 10 Computation at the Root Node for Min Problem

```
1: procedure COMPUTE_ROOT
2:     Receive signals from child nodes
3:     Determine minimum value using comparison operations
4:     Store minimum value in memory
5:     Shift memory to the left
6:     Repeat until memory is full
7: end procedure
```

Algorithm 11 Computation at the Intermediate Node for Min Problem

```
1: procedure COMPUTE_INTERMEDIATE
2:     Use child node states and memory to update the current state
3:     Transmit updated state to the parent node
4: end procedure
```

Algorithm 12 Computation at the Leaf Node for Min Problem

```
1: procedure COMPUTE_LEAF
2:     Treat child state as 1
3:     Compare with the memory value
4:     Update current state based on comparison
5:     Transmit updated state to the parent node
6: end procedure
```

4.2.2 simulation

It is similar to the maximization simulation. the only change is to update the current state with the minimum value which is 0.

Chapter 5

Conclusion and Discussion

In this study, we introduce a new model that aims to reduce CPU workload by incorporating memory and additional processing units within each node of the Cayley tree. This model marks an initial step toward addressing the computational challenges through novel approaches. Specifically, we applied the proposed model to the Cayley tree structure, taking advantage of its unique properties for efficient computation. By distributing a sequence of natural numbers across the Cayley tree nodes, it executed operations successfully to find maximum and minimum values. The inherent properties of the Cayley tree, such as its balanced structure and logarithmic depth, contribute to the complexity of our computations. As a result, the algorithm time complexity is O(log n) compared with traditional array-based approaches.

Our investigations have demonstrated the effectiveness of the model in tackling optimization problems, highlighting its scalability and versatility in handling various computational tasks. Moreover, the use of parallelism concepts within the proposed model enhances computational efficiency, leading to simplified solutions to complex problems. In addition, our study highlights the inherent properties and challenges

associated with Cayley trees. by examining in-depth the properties of Cayley trees and exploring their applications in problem-solving, we have laid the groundwork for future research efforts. As we continue to advance our research, we will further explore our capabilities. our proposed model on the Caleye tree can address various computational problems.

In this model, the main issue is to use a left-shift register for memory. which continuously shifts the value left. because of this, other node values cannot retain the original value. but the root node contains the maximum or minimum value. we can address this issue using a counter register instead of a left shift register.

Note: There is one scenario when we allocate memory with a counter register as a 32-bit size. The counter register acts as an index to access memory elements, enabling efficient memory retrieval and manipulation. However, the drawback lies in the additional memory space required to store the index in the counter register, resulting in reduced memory efficiency.

The focus of our next study is to find the solutions to other computational problems, i.e., the sorting problem.

Bibliography

[1] HASAN AKIN. Cellular automata on cayley tree. *arXiv preprint arXiv:1211.7362*, 2012.

[2] Hasan Akın and Chih-Hung Chang. The entropy and reversibility of cellular automata on cayley tree. *International Journal of Bifurcation and Chaos*, 30(04):2050061, 2020.

[3] Sukanta Das. *Game of Life, Athenian Democracy and Computation*, pages 159–169. Springer International Publishing, Cham, 2022.

[4] Yueting Li, Tianshuo Bai, Xinyi Xu, Yundong Zhang, Bi Wu, Hao Cai, Biao Pan, and Weisheng Zhao. A survey of mram-centric computing: From near memory to in memory. *IEEE Transactions on Emerging Topics in Computing*, 2022.

[5] Massimo Ostilli. Cayley trees and bethe lattices: A concise analysis for mathematicians and physicists. *Physica A: Statistical Mechanics and its Applications*, 391(12):3417–3423, 2012.

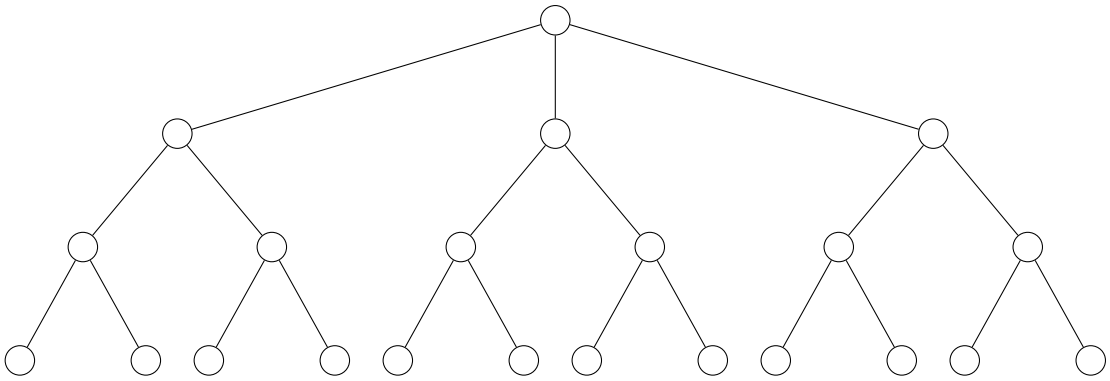
[6] Subrata Paul. Cellular automata: Temporal stochasticity and computability. *arXiv preprint arXiv:2210.13971*, 2022.

Appendix A

Appendix Title

A.1 Proof of Equation 1.1

To prove the formula for the number of nodes in a Cayley tree of order 2 with the height of h:



Let's denote the total number of nodes as n.

At height h, the number of nodes is $3 \times 2^{h-1}$ since:

- At height 0, there is 1 node (the root).
- At height 1, there are $1 + 3$ nodes.
- At height 2, there are $1 + 3 + 3 * 2$ nodes.
- At height h, there are $1 + 3 + 3 * 2 + ... + 3 * 2^{h-1}$ nodes.

Using the sum of a geometric series formula, we get:

$$n = 1+3+3*2+...+3*2^{h-1} = 1+3*(1+2+...+2^{h-1}) = 1+3*\frac{2^h-1}{2-1} = 3*2^h-2$$

Thus, the formula for the number of nodes in a Cayley tree of order 2 with the height h is $3 \times 2^h - 2$.