Project Report

Data Structure and Algorithms

เสนอ

รศ.ดร.รังสิพรรณ มฤคทัต

(Assoc.Prof. Rangsipan Marukatat, Ph.D)

จัดทำโดย

นาย	ณัฐวีพ์	เกิงฝาก	รหัสนักศึกษา	6213125
นาย	วสวัตติ์	เพ็งประโคน	รหัสนักศึกษา	6213132
นางสาว	ณิชารีย์	เฉลิมสุขศรี	รหัสนักศึกษา	6213198
นาย	ภูวิช	โรจนกนกโชค	รหัสนักศึกษา	6213209

Department of computer Engineering

Faculty of Engineering, Mahidol University

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา Data Structure and Algorithms (EGCO 221) โดยคณะผู้จัดทำได้จัดทำขึ้นเพื่ออธิบายการทำงานของโปรแกรม Swinging Money ซึ่งเป็นโปรแกรมที่ใช้ในการตรวจสอบหาเส้นทางที่เป็นไปได้ทั้งหมดของการกระโดดข้ามต้นไม้ของลิง โดยใช้ Data Structure และ Algorithms มาประยุกต์ใช้ในการแก้ปัญหานี้ โดยในรายงานประกอบไปด้วยคู่มือการใช้งานโปรแกรม การอธิบายการทำงานของ Code และ Algorithms รวมไปถึงข้อจำกัดต่าง ๆ ของโปรแกรม

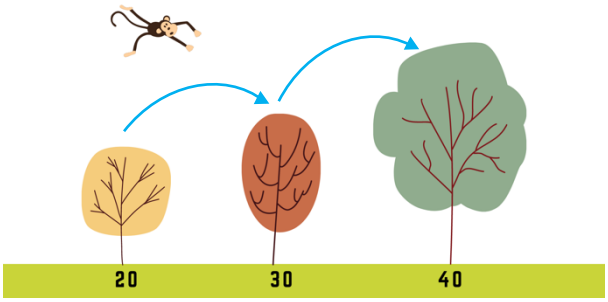
คณะผู้จัดทำขอขอบพระคุณ รศ.ดร.รังสิพรรณ มฤคทัต ผู้ให้ความรู้ และแนวทางในการศึกษา สุดท้ายนี้ทางคณะผู้จัดทำหวังว่ารายงานฉบับนี้ จะให้ความรู้และประโยชน์ไม่มากก็น้อยแก่ผู้อ่านทุกท่าน หากมีข้อผิดพลาดประการใด ผู้จัดทำจะขอน้อมรับไว้ และขออภัยมา ณที่นี้ด้วย

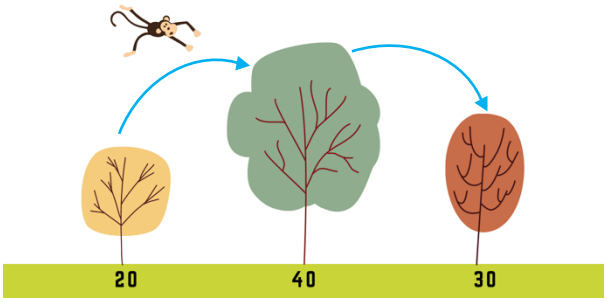
คณะผู้จัดทำ

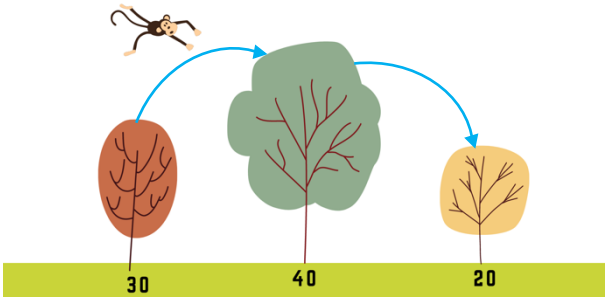
สารบัญ

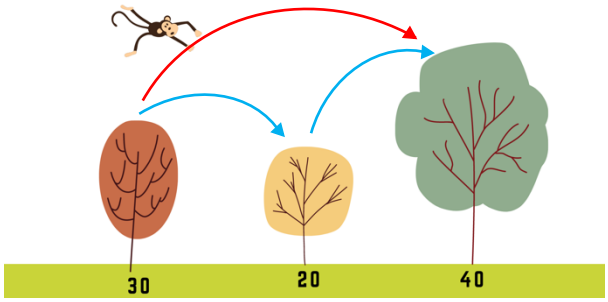
เกี่ยวกับโปรแกรมนี้

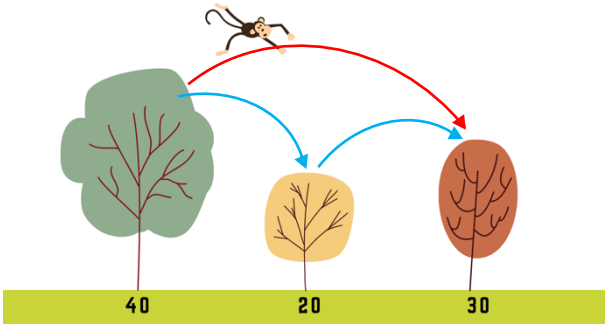
โปรแกรมนี้มีใช้ตรวจสอบหาเส้นทางที่เป็นไปได้ทั้งหมดของการกระโดดข้ามต้นไม้ของลิง โดยมีเงื่อนไขว่าลิงสามารถกระโดดข้ามต้นไม้ไปยังต้นข้างๆที่ห่างออกไป 1 ต้นได้เสมอ และสามารถกระโดดข้ามไปยังต้นที่อยู่ห่างออกไปมากกว่า 1 ต้นได้ก็ต่อเมื่อ ต้นที่อยู่ระหว่างสองต้นนั้นมีความสูงที่เตี้ยกว่าทั้งสองต้น ดังตัวอย่างต่อไปนี้

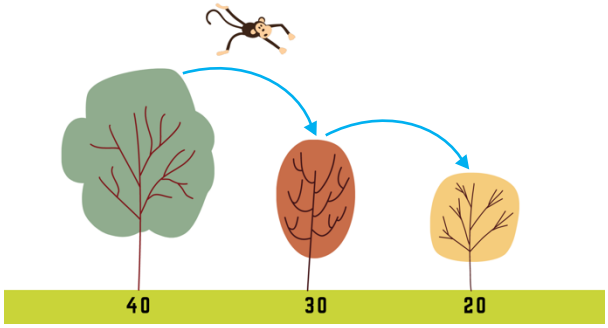






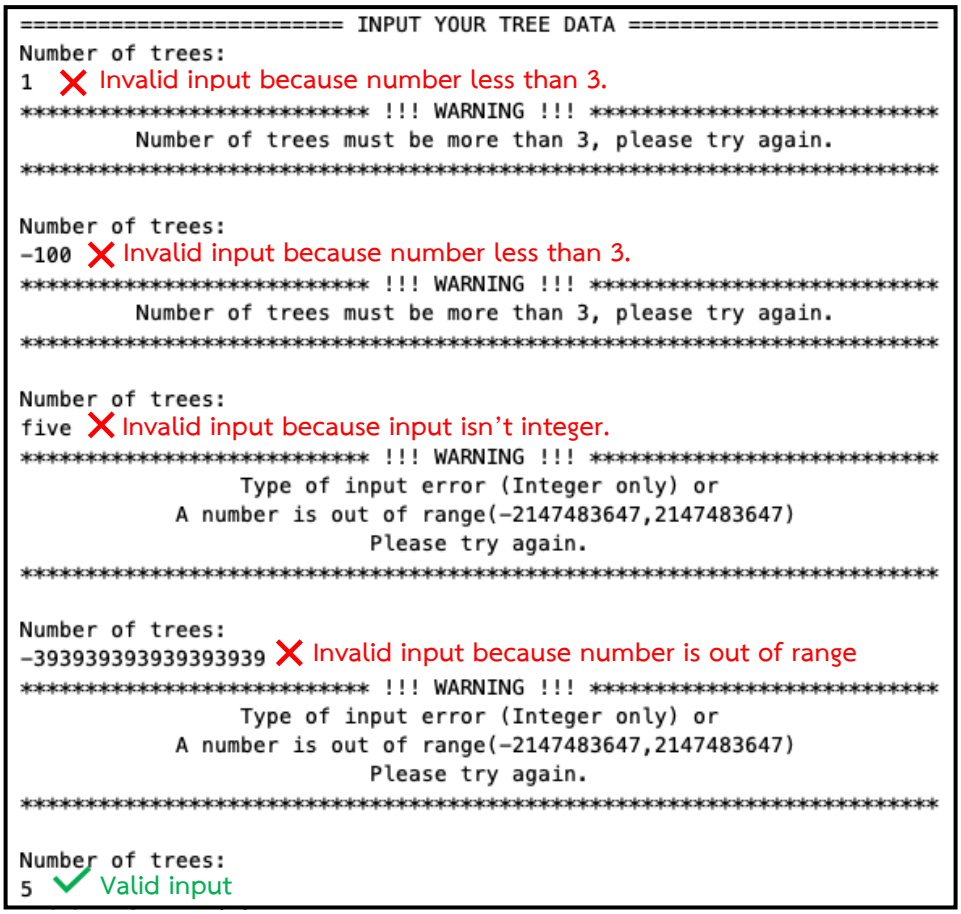






คู่มือการใช้งานโปรแกรม (User Manual)

1. เมื่อโปรแกรมเริ่มทำงาน จะให้ User ป้อนค่าจำนวนต้นไม้ที่ต้องการประมวลผล โดยโปรแกรมกำหนดไว้ว่าจำนวนต้นไม้ที่รับค่าเข้ามาจะต้องมีค่าอย่างน้อย 3 ต้น และจำนวนต้นไม้ที่ไม้ที่ป้อนเข้ามาจะต้องเป็นตัวเลขที่อยู่ในช่วง (-2147483647,2147483647)



2. จากนั้นให้ User ทำการป้อนความสูงของต้นไม้แต่ละต้น โดยโปรแกรมกำหนดไว้ว่าความสูงของต้นไม้ที่รับค่าเข้ามาจะต้องมีค่ามากกว่า 0 และ จำนวนต้นไม้ที่ไม้ที่ป้อนเข้ามาจะต้องเป็นตัวเลขที่อยู่ในช่วง (-2147483647,2147483647)

```
Number of trees:
5
Height of tree (1):
21  ✓ Valid input
Height of tree (2):
-10  ✗ Invalid input because number less than 3.
**************************** !!! WARNING !!! ****************************
         Height of tree must be more than 0, please try again.
*************************************************************************

Height of tree (2):
two  ✗ Invalid input because input isn't integer.
**************************** !!! WARNING !!! ****************************
                Type of input error (Integer only) or
           A number is out of range(-2147483647,2147483647)
                          Please try again.
*************************************************************************

Height of tree (2):
393939393939393939393939  ✗ Invalid input because number is out of range.
**************************** !!! WARNING !!! ****************************
                Type of input error (Integer only) or
           A number is out of range(-2147483647,2147483647)
                          Please try again.
*************************************************************************

Height of tree (2):
19  ✓ Valid input
Height of tree (3):
15  ✓ Valid input
Height of tree (4):
17  ✓ Valid input
Height of tree (5):
20  ✓ Valid input
```

3. หลังจากที่ User กรอกจำนวนต้นไม้ และความสูงของต้นไม้แต่ละต้นเรียบร้อยแล้ว โปรแกรมจะทำการคำนวณเส้นทางทั้งหมดที่ลิงสามารถกระโดดได้ออกมา

```
============================== ALL SOLUTION ==============================
   path 1 :  Tree (1), Height = 21 ft >>>>>> Tree (2), Height = 19 ft
   path 2 :  Tree (2), Height = 19 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 17 ft
   path 4 :  Tree (2), Height = 19 ft >>>>>> Tree (4), Height = 17 ft
   path 5 :  Tree (4), Height = 17 ft >>>>>> Tree (5), Height = 20 ft
   path 6 :  Tree (1), Height = 21 ft >>>>>> Tree (5), Height = 20 ft
   path 7 :  Tree (2), Height = 19 ft >>>>>> Tree (5), Height = 20 ft
==========================================================================
Total tree-pairs : 7 paths
==========================================================================
```

เส้นทางทั้งหมดที่ลิงสามารถกระโดดข้ามต้นไม้มีทั้งหมด 7 เส้นทาง

เส้นทางที่ 1 : จากต้นไม้ต้นที่ 1 สูง 21 ft ลิงสามารถกระโดดไปยังต้นไม้ต้นที่ 2 ความสูง 19 ft ได้
เส้นทางที่ 2 : จากต้นไม้ต้นที่ 2 สูง 19 ft ลิงสามารถกระโดดไปยังต้นไม้ต้นที่ 3 ความสูง 15 ft ได้
เส้นทางที่ 3 : จากต้นไม้ต้นที่ 3 สูง 15 ft ลิงสามารถกระโดดไปยังต้นไม้ต้นที่ 4 ความสูง 17 ft ได้
เส้นทางที่ 4 : จากต้นไม้ต้นที่ 2 สูง 19 ft ลิงสามารถกระโดดไปยังต้นไม้ต้นที่ 4 ความสูง 17 ft ได้
เส้นทางที่ 5 : จากต้นไม้ต้นที่ 4 สูง 17 ft ลิงสามารถกระโดดไปยังต้นไม้ต้นที่ 5 ความสูง 20 ft ได้
เส้นทางที่ 6 : จากต้นไม้ต้นที่ 1 สูง 21 ft ลิงสามารถกระโดดไปยังต้นไม้ต้นที่ 5 ความสูง 20 ft ได้
เส้นทางที่ 7 : จากต้นไม้ต้นที่ 2 สูง 19 ft ลิงสามารถกระโดดไปยังต้นไม้ต้นที่ 5 ความสูง 20 ft ได้

โครงสร้างข้อมูล(Data Structures)

1. Package java.util.*

```
import java.util.*;
```

เป็นการเรียกใช้งานไลบรารี่ของภาษาที่อยู่ใน Package java.util และใน Collection ซึ่งโดยส่วนมากนั้นสืบทอดมาจาก Abstract class และมีการ Implement บาง Interface มารวมกันไว้ โดย Collection นั้นเป็นการเก็บข้อมูลแบบไดนามิกส์ นั่นหมายความว่าเราสามารถเพิ่มข้อมูลเข้าไปใน Collection ได้ไม่จำกัด โดยที่ไม่ต้องกำหนดขนาดสูงสุดในการเก็บข้อมูลล่วงหน้า Collection มีเมธอดที่ทำงานด้วยอัลกอริทึมที่มีประสิทธิภาพเป็นจำนวนมากให้สามารถใช้งานได้ทันที โดยที่ไม่ต้องเขียนขึ้นอีก Collection ในภาษา Java เบื้องต้น เช่น Arraylist, Stack และ Deque **ซึ่งอัลกอริทึมที่เราเลือกใช้ในโปรแกรมคือ ArrayDeque ซึ่งมีพื้นฐานมาจาก Array และ Deque**

2. โครงสร้างข้อมูลใน Class Forest

```
public class Forest {

    ArrayDeque<Tree> treeQ1;
    ArrayDeque<Tree> treeQ2 = new ArrayDeque<Tree>();
    ArrayDeque<Tree> treeQ3 = new ArrayDeque<Tree>();

    Forest(ArrayDeque<Tree> t) {
        treeQ1 = t;
        Swing();
    }

    void Swing()
    {...61 lines }
}
```

Class Forest เป็น Class ที่ทำหน้าที่ในการจัดเก็บต้นไม้ทั้งหมดที่จะทำการประมวลผล โดย **ใช้วิธีการจัดเก็บข้อมูลแบบ ArrayDeque** ที่ Implement Deque Interface มา**ทำให้สามารถเข้าถึงข้อมูลได้ทั้งด้านบนและด้านล่างของข้อมูล** และมีการเก็บข้อมูลแบบ First in , First out (FIFO) หรือ Last in, First out (LIFO) โดยในโปรแกรมเราได้นำ 4 method ที่สำคัญมาใช้ในการทำงาน ดังนี้

1. pollFirst () : เป็นฟังก์ชันที่ใช้คืนค่า และลบข้อมูลใน ArrayDeque จากทางด้านล่าง(หัว)สุดของข้อมูล
2. pollLast () : เป็นฟังก์ชันที่ใช้คืนค่า และลบข้อมูลใน ArrayDeque จากทางด้านบน(ท้าย)สุดของข้อมูล
3. peekFirst () : เป็นฟังก์ชันที่ใช้คืนค่าข้อมูลใน ArrayDeque จากทางด้านล่าง(หัว)สุดของข้อมูล
4. addFirst () : เป็นฟังก์ชันที่ใช้เพิ่มข้อมูลลงไปใน ArrayDeque จากทางด้านล่าง(หัว)สุดของข้อมูล

3. โครงสร้างข้อมูลใน Class Tree

```
class Tree {

    private String ID;
    private int height;
    private int max = 0;

    public Tree(String name, int h) {
        ID = name;
        height = h;
    }

    public String getID() {...3 lines }

    public void setID(String name) {...3 lines }

    public int getHeight() {...3 lines }

    public void setHeight(int h) {...3 lines }

    public int getMax() {...3 lines }

    public void setMax(int m) {...3 lines }

    public void print(Tree t2,int count){...3 lines }
}
```

Class Tree เป็นคลาสที่เปรียบเสมือนกับเป็นต้นไม้ต้นหนึ่ง โดยจะมีข้อมูลเฉพาะของต้นไม้แต่ละต้นดังนี้ 1. ID หรือ หมายเลขประจำต้นไม้ 2. height = ความสูงของต้นไม้ 3. max = ค่าสูงสุดที่ไว้สำหรับตรวจสอบว่า ต้นไม้สามารถกระโดดข้ามต้นอื่น ๆ ได้หรือไม่ และภายใน Class ยังประกอบไปด้วยฟังก์ชันสำหรับคืนค่าและกำหนดค่าตัวแปรต่าง ๆ ใน Class อีกด้วย เช่น ฟังก์ชัน getMax () คือ ฟังก์ชันที่มีไว้ให้ return ค่าสูงสุดของต้นไม้ต้นนั้น เป็นต้น

4. โครงสร้างข้อมูลใน Class Monkey (Main Class)

```
public class Monkey {

    public static void main(String[] args) {...57 lines }
}
```

Class Monkey เป็น Class ที่เปรียบเสมือนกับเป็น Main Class ของโปรแกรม โดยจะทำหน้าที่ในการรับค่าและตรวจสอบ Input

อัลกอริทึมที่ใช้ในการแก้ปัญหา

การแก้ปัญหานี้เราเลือกใช้ ArrayDeque ในการเก็บข้อมูล เนื่องจากสามารถเข้าถึงข้อมูลทั้งด้านบนและด้านล่างของข้อมูล โดยเราจะใช้ทั้งหมด 3 ArrayDeque ในการแก้ปัญหา ดังนี้

- T1 คือ ArrayDeque ที่เก็บต้นไม้ทั้งหมด
- T2 คือ ArrayDeque ที่เก็บต้นไม้ที่มีโอกาสกระโดดข้ามไปยังต้นไม้ต้นอื่นได้อีก
- T3 คือ ArrayDeque ที่เก็บต้นไม้ที่มีโอกาสกระโดดข้ามต้นไม้ต้นอื่นได้อีกเหมือนกับ T2 แต่จะสลับการทำงานกับ T2

โดยมีขั้นตอนการประมวลผลดังนี้

1. เริ่มจากแสดงผลลัพธ์การกระโดดจากต้นแรกใน T1 ไปยังต้นถัดไป เนื่องจากสามารถกระโดดไปยังต้นที่อยู่ติดกันได้อยู่แล้ว

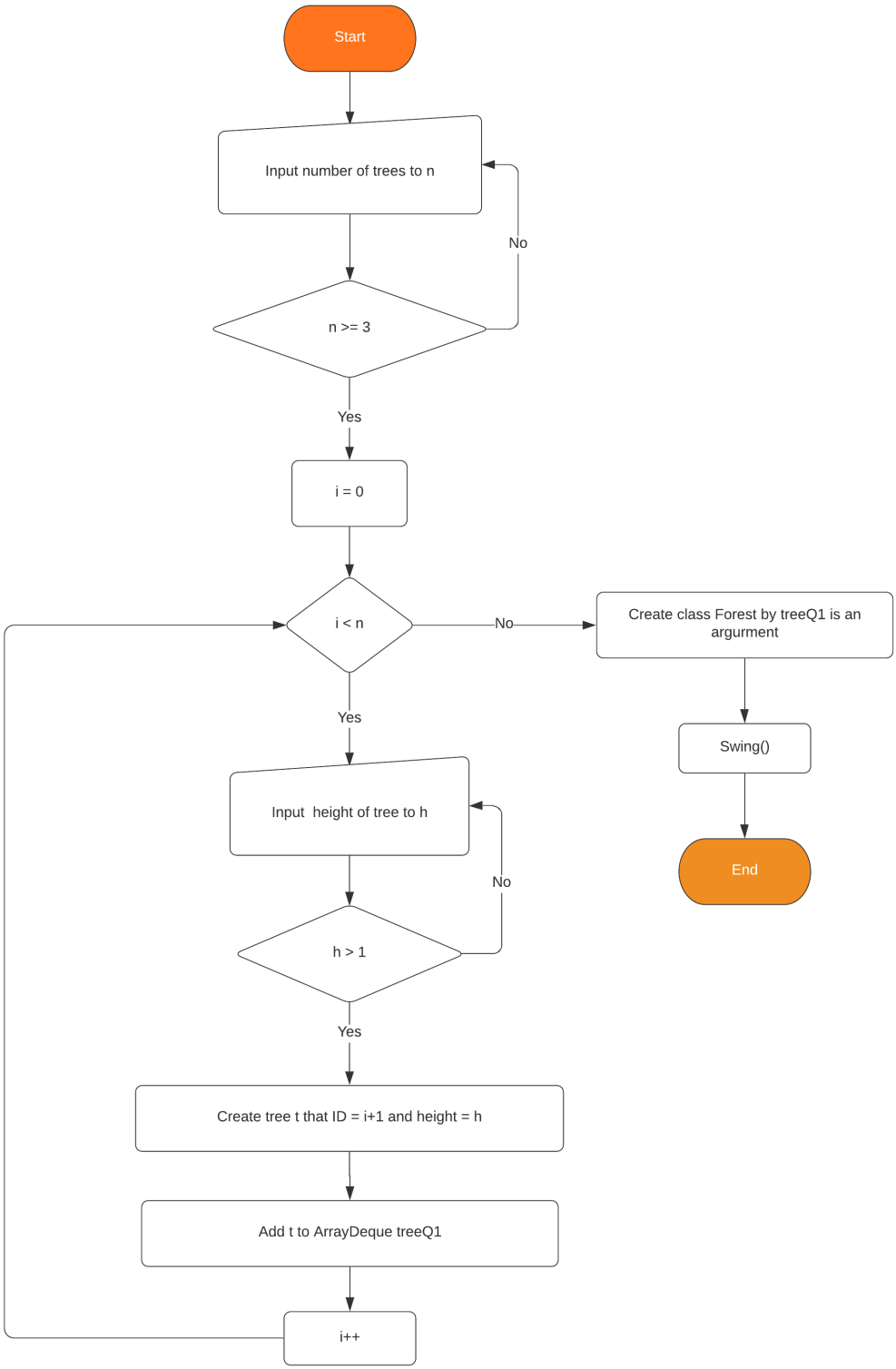
2. ตรวจสอบว่าต้นที่อยู่ปัจจุบันนั้นสูงกว่าต้นถัดไปหรือไม่ ถ้าสูงกว่าแสดงว่าจากต้นที่อยู่ปัจจุบันสามารถกระโดดข้ามไปยังต้นอื่นได้อีก ดังนั้นให้นำต้นที่อยู่ปัจจุบันย้ายไปใส่ใน T2 และ set ค่า max ของต้นนั้นให้เท่ากับความสูงของต้นถัดไป

3. แต่ถ้าต้นที่อยู่ปัจจุบันเตี้ยกว่าต้นถัดไป ให้ไปตรวจสอบต้นไม้ใน T2 จนกว่า T2 จะไม่มีข้อมูล โดยเป็นการตรวจสอบว่าต้นก่อนหน้านี้สามารถกระโดดไปยังต้นถัดไปได้หรือไม่ โดยการตรวจสอบว่าค่า max มีค่ามากกว่าหรือน้อยกว่าความสูงของต้นถัดไป **ถ้ามากกว่าแสดงว่าไม่สามารถกระโดดไปได้ (Invalid tree pairs)** แต่ถ้าน้อยกว่าแสดงว่าสามารถกระโดดไปได้และให้ set ค่า max ของต้นก่อนหน้าให้เท่ากับความสูงของต้นถัดไปแทนค่าเดิม จากนั้นให้ตรวจสอบว่าต้นก่อนหน้าสูงกว่าต้นถัดไปหรือไม่ ถ้าสูงกว่าแสดงว่ามีโอกาสที่จะกระโดดข้ามไปต้นอื่นได้อีก จึงให้นำกลับไปใส่ใน T3 **แต่ถ้าเตี้ยกว่าแสดงว่าไม่สามารถกระโดดข้ามไปต้นอื่นได้อีกแล้ว จึงไม่ต้องนำมาคิดอีก** (ในการประมวลผลรอบถัดไปจะไปตรวจสอบใน T3 แทนและถ้ามีต้นที่สามารถกระโดดข้ามได้อีกจะนำกลับมาใส่ T2 สลับกันไปเรื่อย ๆ ในแต่ละครั้ง)

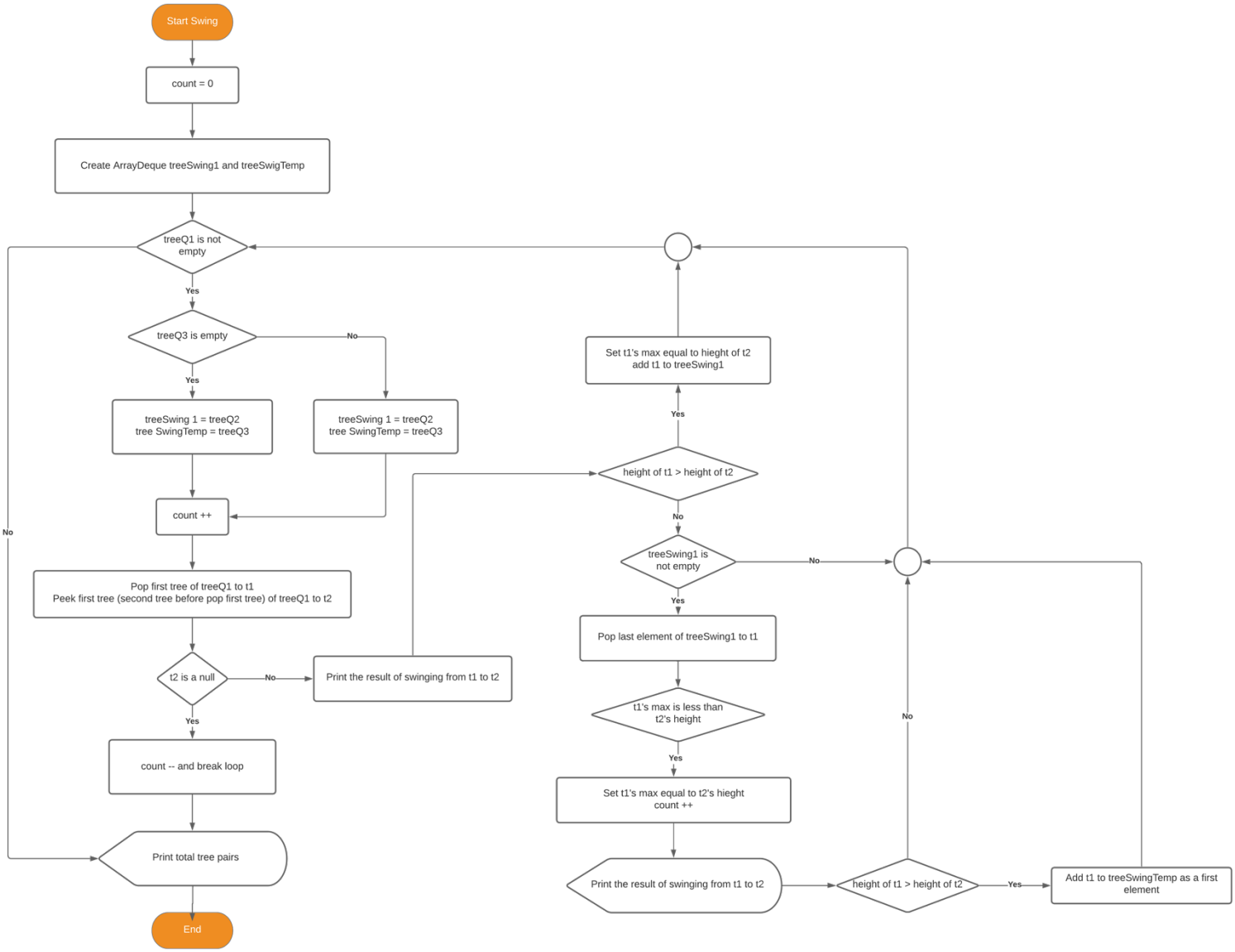
4. ทำซ้ำเรื่อย ๆ จนกว่า T1 จะไม่มีข้อมูลอยู่ โดยทำถึงต้นสุดท้ายแล้วแสดงว่าไม่มีต้นถัดไป ทำให้ไม่สามารถกระโดดต่อได้อีกจึงให้ออกจาก while-loop

แผนผังการทำงานของโปรแกรม (Flow Chart)

1. แผนผังการทำงานของฟังก์ชัน Main



2. แผนผังการทำงานของฟังก์ชัน Swing



อธิบายการทำงานของโปรแกรม

1. ส่วน Data Structure

1.1 Class Forest : เป็น Class ที่ทำหน้าที่จัดเก็บต้นไม้ทั้งหมดที่จะทำการประมวลผล

```
public class Forest {

    ArrayDeque<Tree> treeQ1;
    ArrayDeque<Tree> treeQ2 = new ArrayDeque<Tree>();
    ArrayDeque<Tree> treeQ3 = new ArrayDeque<Tree>();

    Forest(ArrayDeque<Tree> t) {
        treeQ1 = t;
        Swing();
    }

    void Swing() {
```

ไว้สำหรับเก็บข้อมูลต้นไม้

Constructor จะรับ Argument เป็น ArrayDeque ของต้นไม้ทั้งหมดมาไว้ที่ treeQ1 และทำการประมวลผล

Function Swing() : Function สำหรับประมวลผลการกระโดด

1.2 Class Tree : ทำหน้าที่เป็น Class ของต้นไม้แต่ละต้น

```
class Tree {

    private String ID;
    private int height;
    private int max = 0;

    public Tree(String name, int h) {
        ID = name;
        height = h;
    }

    public String getID() {
        return ID;
    }

    public void setID(String name) {
        ID = name;
    }

    public int getHeight() {
        return height;
    }
```

ID = หมายเลขประจำต้นไม้

height = ความสูงของต้นไม้

max = ค่าสูงสุดที่ไว้สำหรับตรวจสอบว่าต้นไม้สามารถกระโดดข้ามต้นอื่น ๆ ได้หรือไม่

Constructor กำหนดค่า ID และ heigh ของต้นไม้เมื่อทำการประกาศตัวแปร Tree

Function สำหรับคืนค่าและกำหนดค่าตัวแปรต่าง ๆ ใน class Tree

```java
public int getMax() {
    return max;
}

public void setMax(int m) {
    max = m;
}
```

Function สำหรับคืนค่าและกำหนดค่าตัวแปรต่าง ๆ ใน class Tree

```java
public void print(Tree t2,int count){
    System.out.println("   path " + count +" :"
            + "  Tree (" + this.getID() + "), Height = "
            + this.getHeight() + " ft >>>>>> Tree "
            + "(" + t2.getID() + "), Height = "
            + t2.getHeight() + " ft");
}
}
```

Function สำหรับแสดงผลลัพธ์การกระโดดไปยังต้นไม้แต่ละต้น

1.3 Class Monkey (Main Class) : ทำหน้าที่เป็น Main class ของโปรแกรม โดยจะทำการรับและตรวจสอบ Input

รับค่าจำนวนต้นไม้ที่จะทำการป้อนเข้าไปในโปรแกรม และตรวจสอบว่า Input ที่รับค่าเข้ามาถูกต้องตามเงื่อนไขหรือไม่

```java
public class Monkey {

    public static void main(String[] args) {
        int n = 0;
        Scanner scanIn = new Scanner(System.in);
        boolean checkInput = false;
        System.out.println("======================== INPUT YOUR TREE DATA ========================");
        while (!checkInput) {
            try {
                System.out.println("Number of trees: ");
                n = Integer.parseInt(scanIn.nextLine());
                if (n < 3) {
                    System.out.println("************************** !!! WARNING !!! **************************");
                    System.out.println("         Number of trees must be more than 3, please try again.        ");
                    System.out.println("*********************************************************************\n");
                } else {
                    checkInput = true;
                }
            } catch (Exception NumberFormatException) {
                System.out.println("************************** !!! WARNING !!! **************************");
                System.out.println("                Type of input error (Integer only) or");
                System.out.println("           A number is out of range(-2147483647,2147483647)");
                System.out.println("                              Please try again. ");
                System.out.println("*********************************************************************\n");
                scanIn.reset();
            }
        }
```

โดยโปรแกรมกำหนดไว้ว่าจำนวนต้นไม้ที่รับค่าเข้ามาจะต้องมีค่าอย่างน้อย 3 ต้น และได้ทำการดัก Exception ของกรณีที่ผู้ใช้กรอกข้อมูลจำนวนต้นไม้เป็นข้อมูลชนิดที่ไม่ใช่ integer หรือ กรณีที่กรอกจำนวนต้นไม้ที่ไม้เป็นตัวเลขที่ไม่อยู่ในช่วง (-2147483647,2147483647) เนื่องจากเกินขอบเขตที่ตัวแปรชนิด integer สามารถรับค่าได้

รับค่าจำนวนความสูงต้นไม้แต่ละต้น ตรวจสอบว่า Input ถูกต้องหรือไม่ และสร้างตัวแปร Tree ตามจำนวนที่กรอกเข้าไปและนำไปเก็บไว้ใน ArrayDeque treeQ1 เพื่อรอส่งให้ class Forest

```
ArrayDeque<Tree> treeQ1 = new ArrayDeque<Tree>();
int h = 0;
for (int i = 0; i < n; i++) {
    checkInput = false;
    while (!checkInput) {
        try {
            System.out.println("Height of tree (" + (i + 1) + "): ");
            h = Integer.parseInt(scanIn.nextLine());
            if (h < 1) {
                System.out.println("************************** !!! WARNING !!! **************************");
                System.out.println("         Height of tree must be more than 0, please try again. ");
                System.out.println("*********************************************************************\n");
            }else{
                checkInput = true;
            }
        } catch (Exception NumberFormatException) {
            System.out.println("************************** !!! WARNING !!! **************************");
            System.out.println("                Type of input error (Integer only) or");
            System.out.println("            A number is out of range(-2147483647,2147483647)");
            System.out.println("                          Please try again. ");
            System.out.println("*********************************************************************\n");
            scanIn.reset();
        }
    }
    Tree t = new Tree(Integer.toString(i + 1), h);
    treeQ1.add(t);
}
new Forest(treeQ1);
}
}
```

ส่ง ArrayDeque treeQ1 ไปสร้างตัวแปร Forest เพื่อทำการประมวลผลในการแก้ปัญหา

โดยโปรแกรมกำหนดไว้ว่าความสูงของต้นไม้ที่รับค่าเข้ามาจะต้องมีค่ามากกว่า 0 และได้ทำการดัก Exception ของกรณีที่ผู้ใช้กรอกข้อมูลความสูงของต้นไม้เป็นข้อมูลชนิดที่ไม่ใช่ integer หรือ กรณีที่กรอกความสูงของต้นไม้ที่ไม้เป็นตัวเลขที่ไม่อยู่ในช่วง (-2147483647,2147483647) เนื่องจากเกินขอบเขตที่ตัวแปรชนิด integer สามารถรับค่าได้

2. ส่วนฟังก์ชันในการประมวลผลแก้ปัญหา

2.1 ฟังก์ชัน Swing : เป็นฟังก์ชันที่ใช้ในการประมวลผลการกระโดด

```
void Swing()
{
    Tree t1;
    Tree t2;
    int count = 0;
    System.out.println("============================ ALL SOLUTION ============================");

    ArrayDeque<Tree> treeSwing1 = new ArrayDeque<Tree>();
    ArrayDeque<Tree> treeSwingTemp = new ArrayDeque<Tree>();

    while (!treeQ1.isEmpty())
    {
        if (treeQ3.isEmpty())
        {
            treeSwing1 = treeQ2;
            treeSwingTemp = treeQ3;
        }
        else
        {
            treeSwing1 = treeQ3;
            treeSwingTemp = treeQ2;
        }
        count++;
        t1 = treeQ1.pollFirst();
        t2 = treeQ1.peekFirst();

        if (t2 == null)
        {
            count--;
            break;
        }
        t1.print(t2, count);

        if (t1.getHeight() > t2.getHeight())
        {
            t1.setMax(t2.getHeight());
            treeSwing1.add(t1);
        }
        else if (t1.getHeight() < t2.getHeight())
        {
            while (!treeSwing1.isEmpty())
            {
                t1 = treeSwing1.pollLast();
                if (t1.getMax() < t2.getHeight())
                {
                    //System.out.println("Check");
                    t1.setMax(t2.getHeight());
                    count++;
                    t1.print(t2, count);
                }

                if (t1.getHeight() > t2.getHeight())
                {
                    treeSwingTemp.addFirst(t1);
                }
            }
        }
    }
    System.out.println("=====================================================================");
    System.out.println("Total tree-pairs : " + count + " paths");
    System.out.println("=====================================================================");
}
}
```

ประกาศ ArrayDeque treeSwing1 และ treeSwingTemp เพื่อใช้ในการชี้ treeQ2 และ treeQ3 สำหรับการนำไปใช้สลับการทำงาน

วนลูปทำจนกว่าจะประมวลผลครบทุกต้นไม้

ใช้สำหรับสลับการทำงานระหว่าง treeQ2 และ treeQ3 โดยเมื่อ treeQ3 ว่างให้ไปทำงานใน treeQ2 ถ้าtreeQ3 ยังมีข้อมูลอยู่ก็ให้ทำใน treeQ3

ทำการ pop ข้อมูลแรกใน treeQ1 ออกมาและ peek ตัวล่างสุดใน treeQ1 (ตัวถัดจากข้อมูลแรก) เพื่อแสดงผลลัพธ์การกระโดดเนื่องจากอยู่ติดกันจึงกระโดดไปได้แน่นอน

ถ้าหากต้นปัจจุบันเป็นต้นสุดท้ายแล้วให้สิ้นสุดลูป

ถ้าหากต้นแรกสูงกว่าต้นถัดไปแสดงว่ามีโอกาสกระโดดข้ามไปต้นอื่น ๆ ได้อีก ให้ set ค่า max เท่ากับความสูงของต้นถัดไป และนำต้นปัจจุบันไปใส่ไว้ใน treeSwing1

ถ้าหากต้นถัดไปสูงกว่าให้ทำการตรวจสอบเงื่อนไขว่าต้นก่อนหน้าต้นใดบ้างที่สามารถกระโดดไปยังต้นถัดไปได้

ตรวจสอบต้นก่อนหน้าทั้งหมดที่มีโอกาสกระโดดไปต้นถัดไป โดยถ้าหากค่า max มีค่าน้อยกว่าความสูงของต้นถัดไปแสดงว่าสามารถกระโดดไปได้ ให้ทำการแสดงผลลัพธ์การกระโดด

ถ้าหากต้นก่อนหน้านั้นสูงกว่าต้นถัดไปแสดงว่ามีโอกาสกระโดดไปข้ามไปต้นอื่นได้อีก ให้ทำการใส่ไปใน ArrayDeque ที่ว่างอยู่

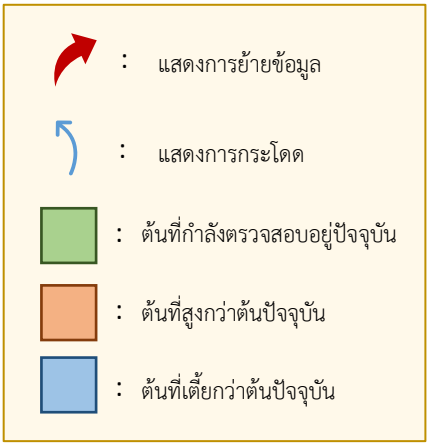
เมื่อทำการประมวลผลเสร็จสิ้นแล้วทำการแสดงจำนวนในการกระโดดทั้งหมด เป็นอันเสร็จสิ้น

ตัวอย่างแสดงการประมวล

ตัวอย่างที่ 1 : Dataset : 19 ,17, 15, 20, 20

```
========================= INPUT YOUR TREE DATA ========================
Number of trees:
5
Height of tree (1):
19
Height of tree (2):
17
Height of tree (3):
15
Height of tree (4):
20
Height of tree (5):
20
```

สัญลักษณ์ที่ใช้ในการอธิบาย



1. นำข้อมูลความสูงของต้นไม้แต่ละต้นมาเก็บไว้ใน T1 โดยเริ่มจากการนำข้อมูลของต้นที่ถูกรับค่าเข้ามาก่อนมาใส่ไว้ใน T1 ก่อน นั่นคือ จะใส่ 19, 17, 15, 20และ 20 เข้าไปใน T1 ตามลำดับ

2. หลังจากรับข้อมูลเข้ามาเก็บไว้ที่ T1 แล้ว จะเริ่มประมวลผลจากต้นแรก เริ่มโดยแสดงผลลัพธ์การกระโดดของต้น 19 (ต้นแรกของ T1) ไปยังต้น 17 (ต้นที่อยู่ถัดไป) หลังจากนั้นตรวจสอบว่า 19 สูงกว่า 17 หรือไม่ ซึ่งต้น 19 นั้นสูงกว่า ดังนั้น นำต้น 19 ย้ายไปใส่ไว้ใน T2 และ set ค่า max ของต้น 19 เท่ากับ 17 (T_{19max} = 17)

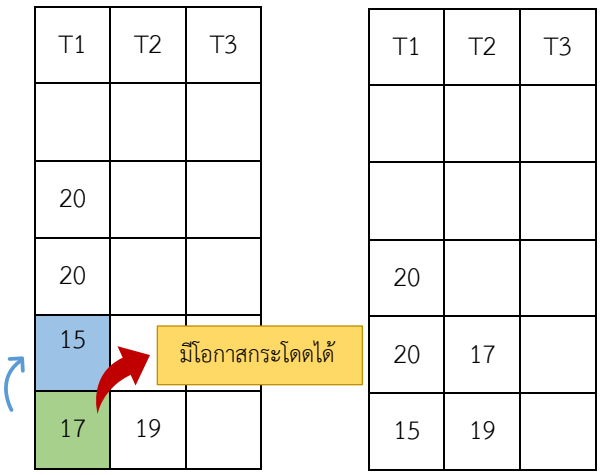
T1	T2	T3
20		
20		
15		
17		
19		

มีโอกาสกระโดดได้

T1	T2	T3
20		
20		
15		
17	19	

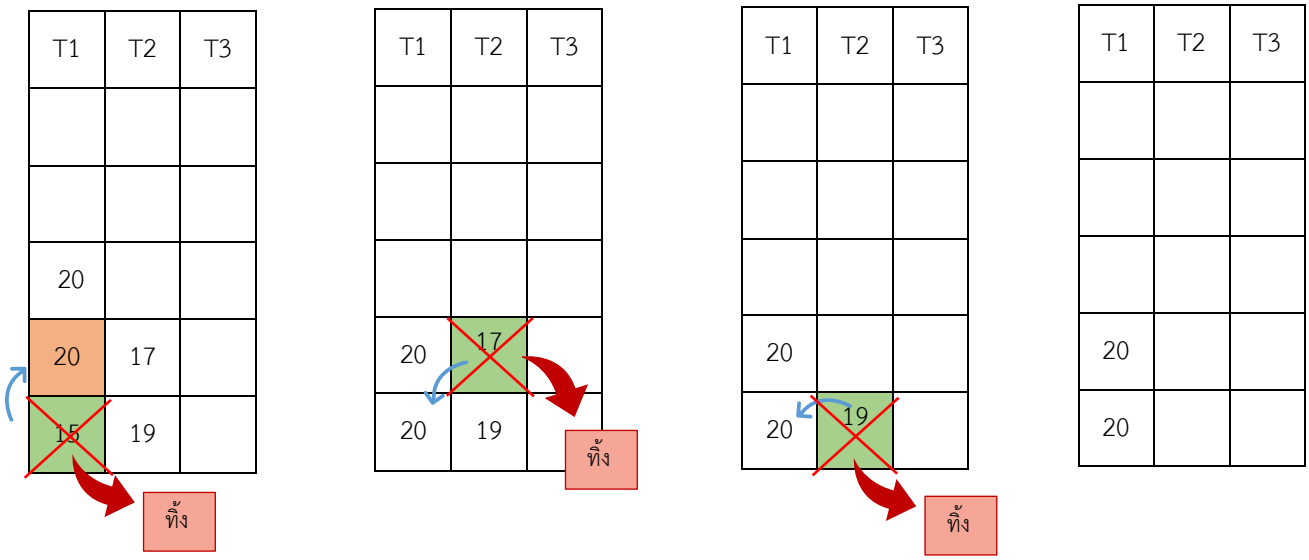
```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 19 ft >>>>>> Tree (2), Height = 17 ft
   path 2 :  Tree (2), Height = 17 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 20 ft
   path 4 :  Tree (2), Height = 17 ft >>>>>> Tree (4), Height = 20 ft
   path 5 :  Tree (1), Height = 19 ft >>>>>> Tree (4), Height = 20 ft
   path 6 :  Tree (4), Height = 20 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 6 paths
======================================================================
```

3. แสดงผลลัพธ์การกระโดดของต้น 17 ไปยังต้น 15 จากนั้นตรวจสอบว่า 17 สูงกว่า 15 หรือไม่ ซึ่งต้น 17 นั้นสูงกว่า ดังนั้น นำต้น 17 ย้ายไปใส่ไว้ใน T2 และ set ค่า max ของต้น 17 เท่ากับ 15 (T_{17max} = 15)



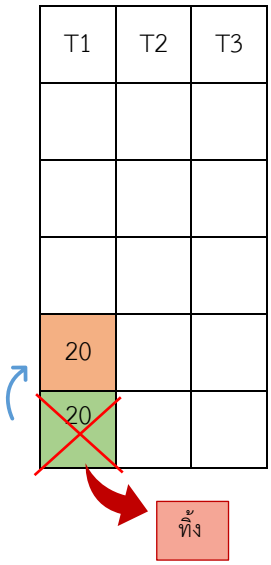

```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 19 ft >>>>>> Tree (2), Height = 17 ft
   path 2 :  Tree (2), Height = 17 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 20 ft
   path 4 :  Tree (2), Height = 17 ft >>>>>> Tree (4), Height = 20 ft
   path 5 :  Tree (1), Height = 19 ft >>>>>> Tree (4), Height = 20 ft
   path 6 :  Tree (4), Height = 20 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 6 paths
======================================================================
```

4. แสดงผลลัพธ์การกระโดดของต้น 15 ไปยัง 20 จากนั้นตรวจสอบว่าต้น 15 สูงกว่าต้น 20 หรือไม่ ซึ่งต้น 15 นั้นเตี้ยกว่าต้น 20 ดังนั้นต้น 15 ไม่สามารถกระโดดข้ามไปต้นอื่นได้อีก จึงนำต้น 15 ออก จากนั้นเข้าไปทำงานใน T2 โดยการตรวจสอบว่าต้น 17 ว่ามีค่า max น้อยกว่า 20 หรือไม่ ซึ่งค่า max ของต้น 17 (T_{17max} = 15) นั้นน้อยกว่า 20 จึงแสดงผลลัพธ์การกระโดดจากต้น 17 ไปยังต้น 20 ซึ่งต้น 17 นั้นเตี้ยกว่าต้น 20 ดังนั้นต้น 17 ไม่สามารถกระโดดข้ามไปต้นอื่นได้อีก จึงนำต้น 17 ออก จากนั้นตรวจสอบว่าต้น 19 (ต้นแรกของ T2) มีค่า max น้อยกว่า 20 หรือไม่ ซึ่งค่า max ของต้น 19 (T_{19max} = 17) นั้นมีค่าน้อยกว่า 20 จึงแสดงผลลัพธ์การกระโดดจากต้น 19 ไปยังต้น 20 จากนั้นตรวจสอบว่าต้น 19 สูงกว่าต้น 20 หรือไม่ ซึ่งต้น 19 นั้นเตี้ยกว่าต้น 20 ดังนั้นต้น 19 ไม่สามารถกระโดดข้ามไปต้นอื่นได้อีก จึงนำต้น 19 ออก



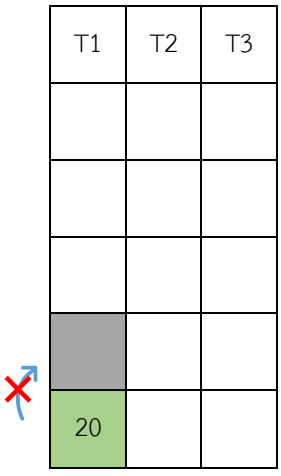
```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 19 ft >>>>>> Tree (2), Height = 17 ft
   path 2 :  Tree (2), Height = 17 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 20 ft
   path 4 :  Tree (2), Height = 17 ft >>>>>> Tree (4), Height = 20 ft
   path 5 :  Tree (1), Height = 19 ft >>>>>> Tree (4), Height = 20 ft
   path 6 :  Tree (4), Height = 20 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 6 paths
======================================================================
```

5. แสดงผลลัพธ์การกระโดดของต้น 20 (ปัจจุบัน) ไปยัง 20 (ที่ถัดไปจากต้นปัจจุบัน) จากนั้นตรวจสอบว่าต้น 20 (ปัจจุบัน) สูงกว่าต้น 20 (ที่ถัดไปจากต้นปัจจุบัน) หรือไม่ ซึ่งต้น 20 (ปัจจุบัน) ไม่ได้สูงกว่าต้น 20 (ที่ถัดไปจากต้นปัจจุบัน) ดังนั้นต้น 20 (ปัจจุบัน) ไม่สามารถกระโดดข้ามไปต้นอื่นได้อีก จึงนำต้น 20 (ปัจจุบัน) ออก




```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 19 ft >>>>>> Tree (2), Height = 17 ft
   path 2 :  Tree (2), Height = 17 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 20 ft
   path 4 :  Tree (2), Height = 17 ft >>>>>> Tree (4), Height = 20 ft
   path 5 :  Tree (1), Height = 19 ft >>>>>> Tree (4), Height = 20 ft
   path 6 :  Tree (4), Height = 20 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 6 paths
======================================================================
```

6. ต้นถัดไปจากต้น 20 นั้นไม่มีแล้ว จึงไม่สามารถกระโดดได้อีก ดังนั้นจบการทำงานและแสดงจำนวนครั้งที่สามารถกระโดดได้ทั้งหมด

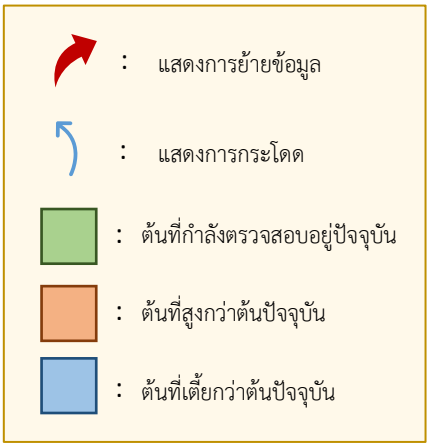



```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 19 ft >>>>>> Tree (2), Height = 17 ft
   path 2 :  Tree (2), Height = 17 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 20 ft
   path 4 :  Tree (2), Height = 17 ft >>>>>> Tree (4), Height = 20 ft
   path 5 :  Tree (1), Height = 19 ft >>>>>> Tree (4), Height = 20 ft
   path 6 :  Tree (4), Height = 20 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 6 paths
======================================================================
```

ตัวอย่างที่ 2 : Dataset = 21, 19 ,15, 17, 20

```
======================== INPUT YOUR TREE DATA ========================
Number of trees:
5
Height of tree (1):
21
Height of tree (2):
19
Height of tree (3):
15
Height of tree (4):
17
Height of tree (5):
20
```

สัญลักษณ์ที่ใช้ในการอธิบาย



1. นำข้อมูลความสูงของต้นไม้แต่ละต้นมาเก็บไว้ใน T1 โดยเริ่มจากการนำข้อมูลของต้นที่ถูกรับค่าเข้ามาก่อนมาใส่ไว้ใน T1 ก่อน นั่นคือ จะใส่ 21, 19, 15, 17 และ 20 เข้าไปใน T1 ตามลำดับ

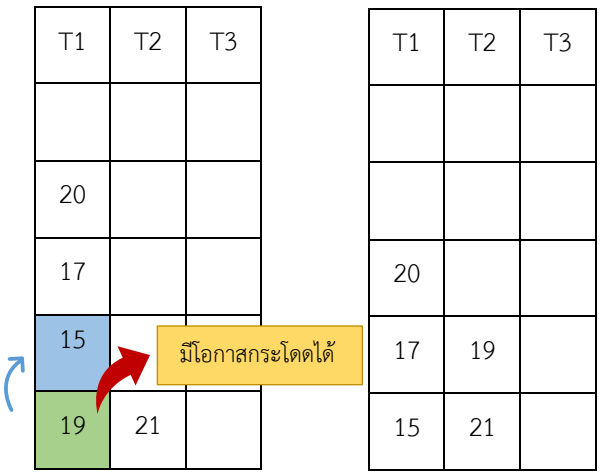
2. หลังจากรับข้อมูลเข้ามาเก็บไว้ที่ T1 แล้ว จะเริ่มประมวลผลจากต้นแรก เริ่มโดยแสดงผลลัพธ์การกระโดดของต้น 21 (ต้นแรกของ T1) ไปยังต้น 19 (ต้นที่อยู่ถัดไป) หลังจากนั้นตรวจสอบว่า 21 สูงกว่า 19 หรือไม่ ซึ่งต้น 21 นั้นสูงกว่า ดังนั้น นำต้น 21 ย้ายไปใส่ไว้ใน T2 และ set ค่า max ของต้น 21 เท่ากับ 19 (T_{21max} = 19)

T1	T2	T3
20		
17		
15		
19		
21		

T1	T2	T3
20		
17		
15		
19	21	

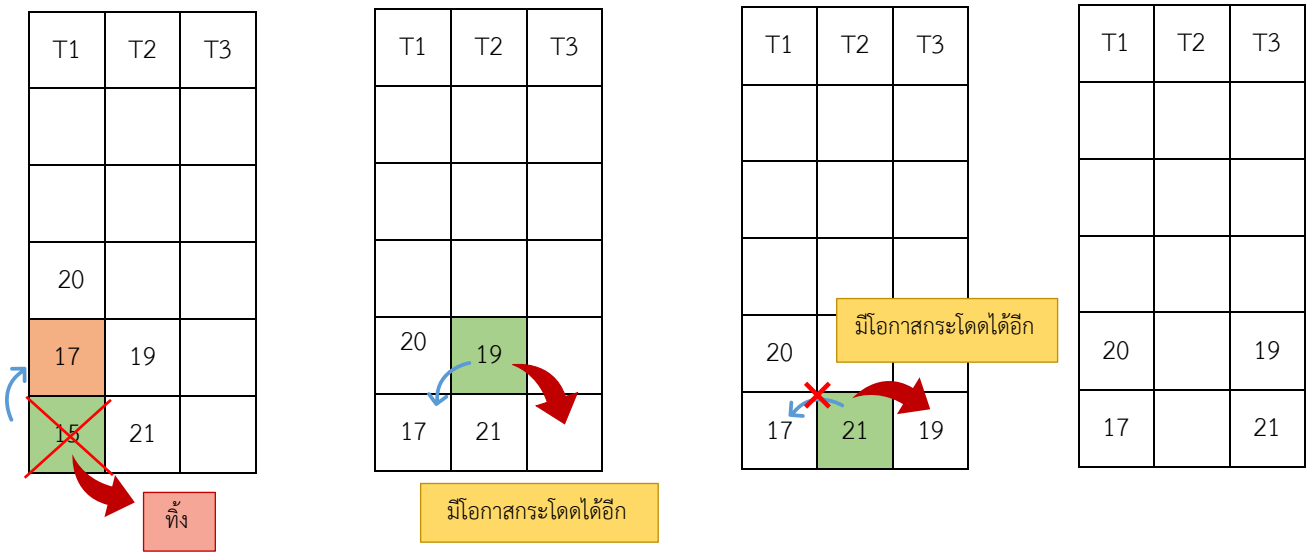
```
========================== ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 21 ft >>>>>> Tree (2), Height = 19 ft
   path 2 :  Tree (2), Height = 19 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 17 ft
   path 4 :  Tree (2), Height = 19 ft >>>>>> Tree (4), Height = 17 ft
   path 5 :  Tree (4), Height = 17 ft >>>>>> Tree (5), Height = 20 ft
   path 6 :  Tree (1), Height = 21 ft >>>>>> Tree (5), Height = 20 ft
   path 7 :  Tree (2), Height = 19 ft >>>>>> Tree (5), Height = 20 ft
=====================================================================
Total tree-pairs : 7 paths
=====================================================================
```

3. แสดงผลลัพธ์การกระโดดของต้น 19 ไปยังต้น 15 จากนั้นตรวจสอบว่า 19 สูงกว่า 15 หรือไม่ ซึ่งต้น 19 นั้นสูงกว่า ดังนั้น นำต้น 19 ย้ายไปใส่ไว้ใน T2 และ set ค่า max ของต้น 19 เท่ากับ 15 (T_{19max} = 15)



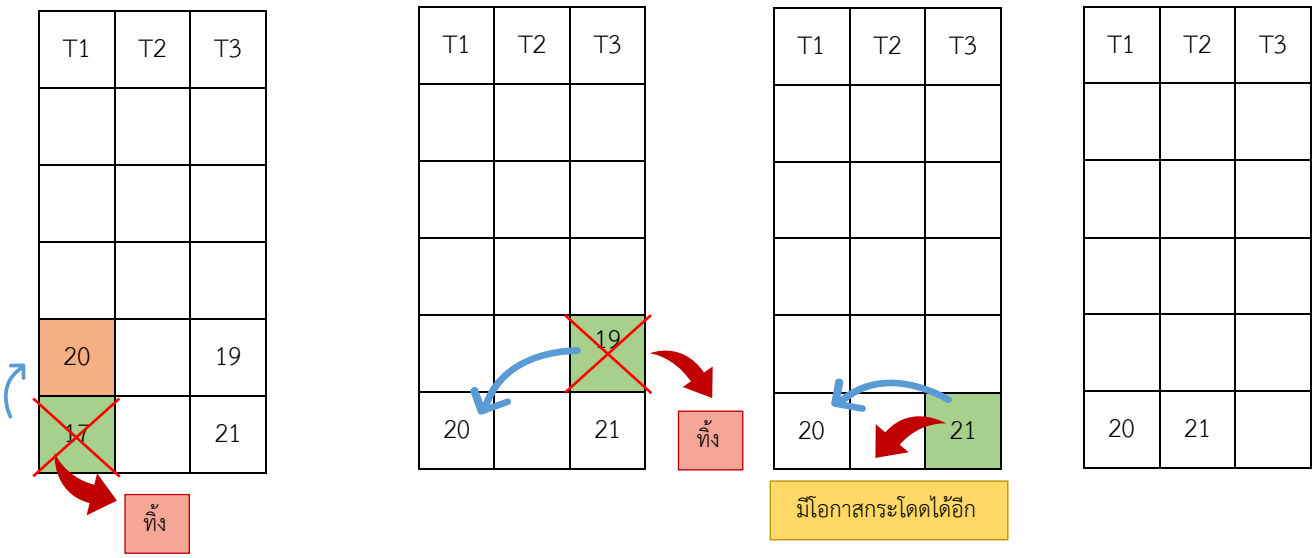

```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 21 ft >>>>>> Tree (2), Height = 19 ft
   path 2 :  Tree (2), Height = 19 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 17 ft
   path 4 :  Tree (2), Height = 19 ft >>>>>> Tree (4), Height = 17 ft
   path 5 :  Tree (4), Height = 17 ft >>>>>> Tree (5), Height = 20 ft
   path 6 :  Tree (1), Height = 21 ft >>>>>> Tree (5), Height = 20 ft
   path 7 :  Tree (2), Height = 19 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 7 paths
-======================================================================
```

4. แสดงผลลัพธ์การกระโดดของต้น 15 ไปยัง 17 จากนั้นตรวจสอบว่าต้น 15 สูงกว่าต้น 17 หรือไม่ ซึ่งต้น 15 นั้นเตี้ยกว่าต้น 17 ดังนั้นต้น 15 ไม่สามารถกระโดดข้ามไปต้นอื่นได้อีก จึงนำต้น 15 ออก จากนั้นเข้าไปทำงานใน T2 โดยการตรวจสอบว่า ต้น 19 ว่ามีค่า max น้อยกว่า 17 หรือไม่ ซึ่งค่า max ของต้น 19 (T_{19max} = 15) นั้นน้อยกว่า 17 จึงแสดงผลลัพธ์การกระโดดจากต้น 19 ไปยังต้น 17 และ set ค่า max ของต้น 19 เท่ากับ 17 (T_{19max} = 17) นอกจากนี้ต้น 19 ก็สูงกว่าต้น 17 จึงมีโอกาสกระโดดข้ามได้อีก จึงนำต้น 19 ไปใส่ไว้ในต้น T3 จากนั้นตรวจสอบต้น 21 (ต้นแรกของ T2) มีค่า max น้อยกว่า 17 หรือไม่ ซึ่งค่า max ของต้น 21 (T_{21max} = 19) นั้นมากกว่า 17 ดังนั้นไม่สารมารถกระโดดจากต้น 21 ไปยังต้น 17 ได้ แต่ต้น 21 ก็ยังสูงกว่าต้น 17 ดังนั้นยังมีโอกาสที่จะกระโดดข้ามต่อได้ จึงนำต้น 21 ไปใส่ไว้ใน T3



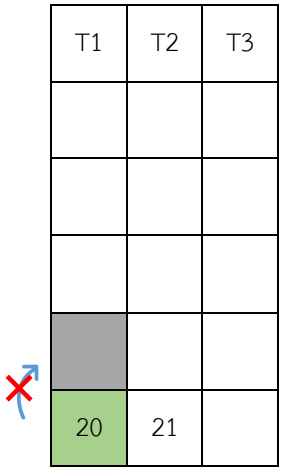
```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 21 ft >>>>>> Tree (2), Height = 19 ft
   path 2 :  Tree (2), Height = 19 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 17 ft
   path 4 :  Tree (2), Height = 19 ft >>>>>> Tree (4), Height = 17 ft
   path 5 :  Tree (4), Height = 17 ft >>>>>> Tree (5), Height = 20 ft
   path 6 :  Tree (1), Height = 21 ft >>>>>> Tree (5), Height = 20 ft
   path 7 :  Tree (2), Height = 19 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 7 paths
-======================================================================
```

5. แสดงผลลัพธ์การกระโดดของต้น 17 ไปยังต้น 20 จากนั้นตรวจสอบว่าต้น 17 สูงกว่าต้น 20 หรือไม่ ซึ่งต้น 20 นั้นเตี้ยกว่าต้น 20 ดังนั้นต้น 17 ไม่สามารถกระโดดข้ามต้นอื่นได้อีก จึงนำต้น 17 ออก จากนั้นเข้าไปทำงานใน T3 โดยการตรวจสอบว่า ต้น 19 (ต้นสุดท้ายของ T3) ว่ามีค่า max น้อยกว่า 20 หรือไม่ ซึ่งค่า max ของต้น 19 (T_{19max} = 17) นั้นน้อยกว่า 20 จึงแสดงผลลัพธ์การกระโดดจากต้น 19 ไปยังต้น 20 และ set ค่า max ของต้น 19 เท่ากับ 20 (T_{19max} = 20) แต่ต้น 19 เตี้ยกว่าต้น 20 ไม่สามารถสกระโดดข้ามได้อีก จึงนำต้น 19 ออก จากนั้นตรวจสอบต้น 21 (ต้นแรกของ T3) ว่ามีค่า max น้อยกว่า 20 หรือไม่ ซึ่งค่า max ของต้น 21 (T_{21max} = 19) นั้นน้อยกว่า 20 จึงแสดงผลลัพธ์การกระโดดจากต้น 21 ไปยังต้น 20 และ set ค่า max ของต้น 21 เท่ากับ 20 (T_{21max} = 20) นอกจากนี้ต้น 21 ก็สูงกว่าต้น 20 ดังนั้นมีโอกาสกระโดดได้อีก จึงนำต้น 21 กลับไปใส่ใน T2




```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 21 ft >>>>>> Tree (2), Height = 19 ft
   path 2 :  Tree (2), Height = 19 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 17 ft
   path 4 :  Tree (2), Height = 19 ft >>>>>> Tree (4), Height = 17 ft
   path 5 :  Tree (4), Height = 17 ft >>>>>> Tree (5), Height = 20 ft
   path 6 :  Tree (1), Height = 21 ft >>>>>> Tree (5), Height = 20 ft
   path 7 :  Tree (2), Height = 19 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 7 paths
-======================================================================
```

6. ต้นถัดไปจากต้น 20 นั้นไม่มีแล้ว จึงไม่สามารถกระโดดได้อีก ดังนั้นจบการทำงานและแสดงจำนวนครั้งที่สามารถกระโดดได้ทั้งหมด




```
============================ ALL SOLUTION ============================
   path 1 :  Tree (1), Height = 21 ft >>>>>> Tree (2), Height = 19 ft
   path 2 :  Tree (2), Height = 19 ft >>>>>> Tree (3), Height = 15 ft
   path 3 :  Tree (3), Height = 15 ft >>>>>> Tree (4), Height = 17 ft
   path 4 :  Tree (2), Height = 19 ft >>>>>> Tree (4), Height = 17 ft
   path 5 :  Tree (4), Height = 17 ft >>>>>> Tree (5), Height = 20 ft
   path 6 :  Tree (1), Height = 21 ft >>>>>> Tree (5), Height = 20 ft
   path 7 :  Tree (2), Height = 19 ft >>>>>> Tree (5), Height = 20 ft
======================================================================
Total tree-pairs : 7 paths
======================================================================
```

Runtime ของโปรแกรม

```
void Swing() {
    Tree t1;
    Tree t2;
    int count = 0;
    System.out.println("============================ ALL SOLUTION ============================");

    ArrayDeque<Tree> treeSwing1 = new ArrayDeque<Tree>();
    ArrayDeque<Tree> treeSwingTemp = new ArrayDeque<Tree>();
    while (!treeQ1.isEmpty()) {
        //o1++;
        if(treeQ3.isEmpty()){
            treeSwing1 = treeQ2;
            treeSwingTemp = treeQ3;
        }
        else {
            treeSwing1 = treeQ3;
            treeSwingTemp = treeQ2;
        }
        count++;
        t1 = treeQ1.pollFirst();
        t2 = treeQ1.peekFirst();
        if (t2 == null) {
            count--;
            break;
        }
        t1.print(t2,count);
        if (t1.getHeight() > t2.getHeight()) {
            t1.setMax(t2.getHeight());
            treeSwing1.add(t1);
        } else if (t1.getHeight() < t2.getHeight()){
            while (!treeSwing1.isEmpty()) {
                t1 = treeSwing1.pollLast();
                if (t1.getMax() < t2.getHeight()) {
                    //System.out.println("Check");
                    t1.setMax(t2.getHeight());
                    count++;
                    t1.print(t2,count);
                }
                if (t1.getHeight() > t2.getHeight()) {
                    treeSwingTemp.addFirst(t1);
                }
            }
        }
    }
    System.out.println("=======================================================================");
    System.out.println("Total tree-pairs : " + count +" paths");
    System.out.println("=======================================================================");
}
```

Asymptotic runtime ของลูปชั้นนอก จะมีค่าเท่ากับผลรวมของ Asymptotic runtime ของลูปชั้นในทุก loop

Asymptotic runtime ของลูปชั้นในจะมีค่าไม่เท่ากันในแต่ละรอบ ซึ่งในกรณีที่แย่ที่สุด Asymptotic runtime จะมีค่า ประมาณ O(n)

Asymptotic runtime ของลูปชั้นใน หรือ ลูป while (!treeSwing1.isEmpty) จะมีค่าไม่เท่ากันในแต่ละรอบ เนื่องจากเรานำ ArrayDeque มาใช้ในการเก็บข้อมูล ทำให้ไม่จำเป็นต้องไล่ตรวจสอบข้อมูลทุกตัว ซึ่งในกรณีที่แย่ที่สุดจะมี Asymptotic runtime ประมาณ O(n) ส่งผลให้ในกรณีที่แย่ที่สุดลูปชั้นนอก หรือ ลูป while (!treeQ1.isEmpty()) ซึ่งมี Asymptotic runtime เท่ากับผลรวมของลูปชั้นในทุก ๆ ครั้งจะมี Asymptotic runtime ประมาณ O(n) ดังนั้น Asymptotic runtime ของ Algorithm นี้ จะมีค่าเท่ากับ O(n)

Algorithm นี้ถึงดีกว่าการ Brute-fore อย่างไร?

Brute-fore เป็นการทำงานแบบลองผิดลองถูก จึงทำให้บาง process เกิดการทำงานซ้ำในกรณีนี้เปรียบได้กับการใช้ For loop 2 ชั้น เนื่องจากทำการเทียบเงื่อนไขในทุกกรณีที่เป็นไปได้ ซึ่งจะมีค่า Asymptotic runtime เท่ากับ $O(n^2)$ แต่ Algorithm ของเราได้นำ ArrayDeque เข้ามาใช้เก็บค่าก่อนหน้าทำให้ไม่จำเป็นต้องไล่ตรวจสอบข้อมูลทุกตัว และสามารถกระโดดข้ามการทำงานที่ซ้ำซ้อนได้ โดยจากข้อมูลในหน้าก่อนหน้าสรุปได้ว่า **Asymptotic runtime ของ Algorithm นี้ จะมีค่าเท่ากับ O(n) ซึ่งมี Asymptotic runtime ดีกว่า Brute-fore**

ข้อจำกัดของโปรแกรม

1. จำนวนต้นไม้ที่ทำให้ Runtime นานกว่า 5 นาที

เงื่อนไข สุ่มความสูงของต้นไม้แต่ละต้นให้อยู่ในช่วง 10 ถึง 30 และจำนวนต้นไม้อยู่ในรูป 10^n

* <u>หมายเหตุ</u> : Runtime เริ่มนับจากในส่วนของการประมวลผลเท่านั้น ไม่รวมตอนที่ป้อนข้อมูลเข้าไป

1.1 10 ต้น

```
Total tree-pairs : 15 paths
====================================
Total runtime = 0.0322489 seconds
```

1.2 100 ต้น

```
Total tree-pairs : 174 paths
====================================
Total runtime = 0.0408376 seconds
```

1.3 1000 ต้น

```
Total tree-pairs : 1778 paths
====================================
Total runtime = 0.0711711 seconds
```

1.4 10000 ต้น

```
Total tree-pairs : 18001 paths
====================================
Total runtime = 0.5625438 seconds
```

1.5 100000 ต้น

```
Total tree-pairs : 180121 paths
====================================
Total runtime = 5.2294462 seconds
```

1.6 1000000 ต้น

```
Total tree-pairs : 1800647 paths
====================================
Total runtime = 58.2308588 seconds
```

1.7 10000000 ต้น

```
Total tree-pairs : 18009826 paths
=====================================
Total runtime = 598.5170473 seconds
```

พบว่าเมื่อจำนวนต้นไม้เท่ากับ 10000000 ต้น Runtime จะอยู่ที่ประมาณ 10 นาที ซึ่งมากกว่า 5 นาทีไปมาก จึงทดลองด้วยการลดลงมาครึ่งนึง คือ จำนวนต้นไม้เท่ากับ 5000000 ต้น ได้ผลลัพธ์เป็น

1.8 5000000 ต้น

```
Total tree-pairs : 9004793 paths
=======================================
Total runtime = 303.5343376 seconds
```

พบว่าเมื่อจำนวนต้นไม้เท่ากับ 5000000 ต้น Runtime จะอยู่ที่ประมาณ 5 นาที ดังนั้นให้จำนวนต้นไม้เท่ากับ 5000000 ต้น เป็นลิมิตของโปรแกรมนี้

2. ช่วงความสูงของต้นไม้

ตรวจสอบว่าช่วงความสูงของต้นไม้มีผลต่อ Runtime หรือไม่ โดยจากผลของข้อที่ Runtime เพิ่มอย่างมีนัยสำคัญเมื่อจำนวนต้นไม้มากกว่า 10000 ต้น จึงเริ่มทดลองในจำนวนต้นไม้ 10000 ต้น 100000 ต้น และ 1000000 ต้น

2.1 ช่วงความสูง 1-1000

10000 ต้น →

```
Total tree-pairs : 19502 paths
==================================
Total runtime = 0.728495 seconds
```

100000 ต้น →

```
Total tree-pairs : 195317 paths
===================================
Total runtime = 5.6469043 seconds
```

1000000 ต้น →

```
Total tree-pairs : 1954639 paths
====================================
Total runtime = 59.3685607 seconds
```

2.2 ช่วงความสูง 1-10000

10000 ต้น →

```
Total tree-pairs : 19637 paths
==================================
Total runtime = 0.6634494 seconds
```

100000 ต้น →

```
Total tree-pairs : 195824 paths
==================================
Total runtime = 5.7142589 seconds
```

1000000 ต้น →

```
Total tree-pairs : 1959757 paths
==================================
Total runtime = 58.9640247 seconds
```

เมื่อทดลองในช่วง 1-1000 และ 1-10000 พบว่า runtime เปลี่ยนแปลงน้อยมาก จึงตั้งสมมติฐานว่าช่วงความสูงของต้นไม้ไม่ส่งผลต่อ Runtime ลองพิสูจน์สมมติฐานด้วยการเพิ่มช่วงขนาดใหญ่เข้าไปคือช่วงความสูง 1-10000000

2.3 ช่วงความสูง 1-10000000

10000 ต้น →

```
Total tree-pairs : 19635 paths
==================================
Total runtime = 0.6267718 seconds
```

100000 ต้น →

```
Total tree-pairs : 196142 paths
==================================
Total runtime = 5.838031 seconds
```

1000000 ต้น →

```
Total tree-pairs : 1961623 paths
==================================
Total runtime = 60.0687074 seconds
```

พบว่า Runtime ก็ยังแทบไม่เปลี่ยนแปลงอยู่ดี ดังนั้นจึงสรุปได้ว่าช่วงความสูงของต้นไม้ไม่ส่งผลต่อ Runtime ของโปรแกรม

แหล่งอ้างอิง

1. Mohammed H และ Vitalii Fedorenko. Measure execution time for a Java method [duplicate]. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 10 มีนาคม 2564). เข้าถึงได้จาก : https://stackoverflow.com/
2. MarcusCode. Collections. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 17 มีนาคม 2564). เข้าถึงได้จาก : http://marcuscode.com/lang/java/collections
3. pixabay. Wild Monkey Animal. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 20 มีนาคม 2564). เข้าถึงได้จาก : https://www.canva.com/media/
4. sketchify. Tree. [ออนไลน์]. (วันที่ค้นหาข้อมูล : 20 มีนาคม 2564). เข้าถึงได้จาก : https://www.canva.com/media/