Report

EGCO 221 : Data Structure and Algorithms

รศ.ดร รังสิพรรณ มฤคทัต

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

นาย ฉัตริน มุกสกุล รหัสนักศึกษา 6213124

นาย วชิรวิชญ์ พีรพิศาลพล รหัสนักศึกษา 6213145

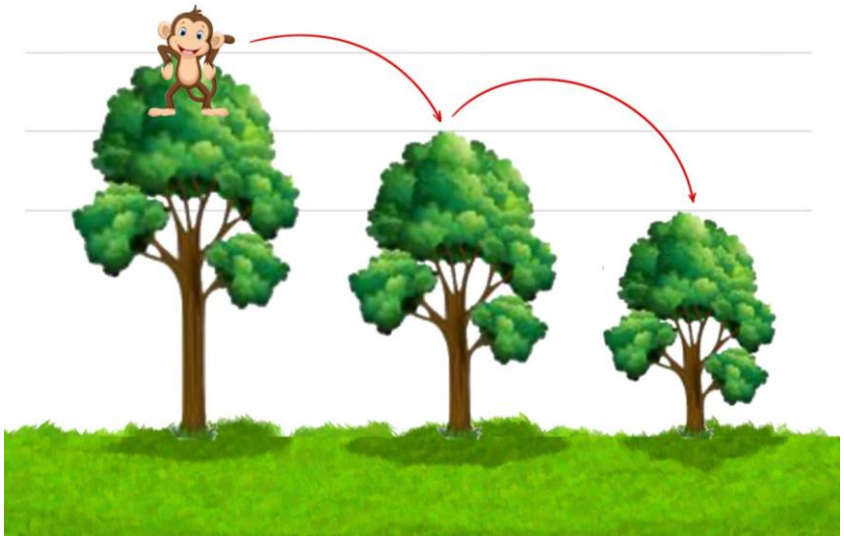
นาย วิญญ์ ลดาวุฒิพันธ์ รหัสนักศึกษา 6213146

Department of Computer Engineering

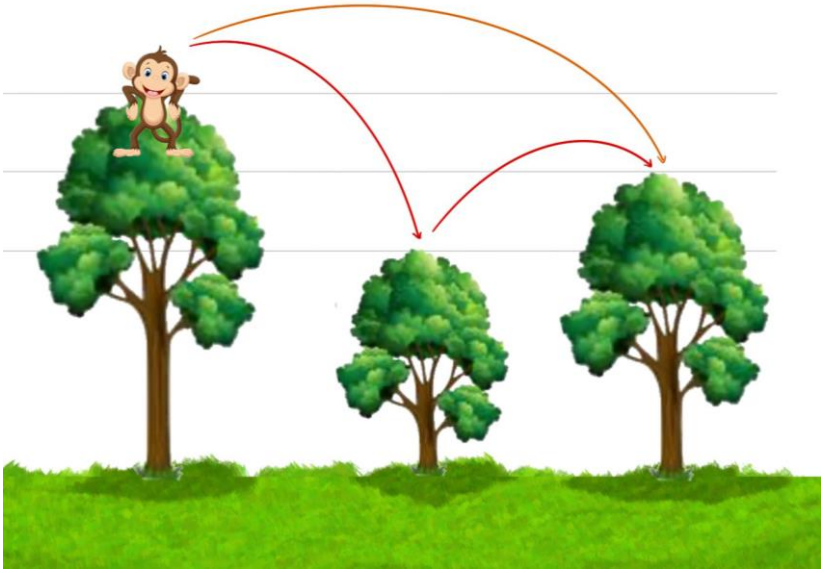
Faculty of Engineering, Mahidol University

เกี่ยวกับตัวโปรแกรม

โปรแกรมนี้มีไว้เพื่อตรวจสอบรูปแบบการจับคู่ระหว่างจุดที่มีค่าเปรียบเทียบตามความสัมพันธ์ที่เปรียบเป็นการโหนต้นไม้ของลิง โดยลิงสามารถโหนต้นไม้ไปยังต้นที่อยู่ติดกันได้ และสามารถโหนข้ามต้นไม้ได้เมื่อต้นที่อยู่ระหว่างจุดที่ลิงอยู่กับจุดเป้าหมายมีความสูงมากกว่าต้นที่อยู่ระหว่างกลาง โดยแสดงผลเป็นจำนวนคู่ของต้นไม้ในการโหนในทุกรูปแบบด้วยการระบุจำนวนของต้นไม้และความสูงของต้นไม้แต่ละต้น



ตัวอย่างการโหนต้นไม้กรณีที่ข้ามไม่ได้



ตัวอย่างการโหนต้นไม้กรณีที่ข้ามได้

คู่มือการใช้โปรแกรม

1. เมื่อเริ่มโปรแกรม จะให้ป้อน input จำนวนเต็มบวกมากกว่า 2 เพื่อบอกจำนวนต้นไม้ตั้งแต่ 3 ต้นขึ้นไป หรือป้อนค่า 0 เพื่อสิ้นสุดการทำงานของโปรแกรม จากนั้นให้ป้อนค่าจำนวนเต็มบวก ซึ่งเป็นความสูงของต้นไม้แต่ละต้น ตามจำนวนต้นไม้

```
Enter number of tree [Enter 0 to exit] :
8
Height of tree (1) =
10
Height of tree (2) =
15
Height of tree (3) =
12
Height of tree (4) =
8
Height of tree (5) =
20
Height of tree (6) =
16
Height of tree (7) =
6
Height of tree (8) =
3
```

2. หลังจากที่ป้อนข้อมูลความสูงของต้นไม้ครบตามจำนวน โปรแกรมจะคำนวณและแสดงรูปแบบใน การจับคู่การโหนต้นไม้แต่ละต้นของลิง โดยจะแสดงผลจากต้นทางซ้ายสุด (ต้นที่ 1) ไปจนถึงต้นทางขวาสุด (ต้นสุดท้าย) พร้อมทั้งแสดงจำนวนคู่ทั้งหมด เมื่อสิ้นสุดการทำงาน จะให้เริ่มป้อนค่าจำนวนต้นไม้ใหม่อีกรอบ

```
--Solution--

Tree( 1),( 10) ft --> Tree( 2),( 15) ft
Tree( 1) pair = 1
-----------------------------------------
Tree( 2),( 15) ft --> Tree( 3),( 12) ft
Tree( 2),( 15) ft --> Tree( 5),( 20) ft
Tree( 2) pair = 2
-----------------------------------------
Tree( 3),( 12) ft --> Tree( 4),(  8) ft
Tree( 3),( 12) ft --> Tree( 5),( 20) ft
Tree( 3) pair = 2
-----------------------------------------
Tree( 4),(  8) ft --> Tree( 5),( 20) ft
Tree( 4) pair = 1
-----------------------------------------
Tree( 5),( 20) ft --> Tree( 6),( 16) ft
Tree( 5) pair = 1
-----------------------------------------
Tree( 6),( 16) ft --> Tree( 7),(  6) ft
Tree( 6) pair = 1
-----------------------------------------
Tree( 7),(  6) ft --> Tree( 8),(  3) ft
Tree( 7) pair = 1
-----------------------------------------
Total Pair = 9

Enter number of tree [Enter 0 to exit]:
0
-----------------------------------------
BUILD SUCCESS
-----------------------------------------
```

Structure

ใช้ ArrayList รับค่าจาก user และใช้เก็บค่า index เส้นทางที่เป็นไปได้ในแต่ละต้นแล้วใช้ Stack ในการเก็บข้อมูลที่รับเข้ามา เนื่องจากเวลาในการเก็บข้อมูลลงในStackนั้น Asymptotic runtime จะเท่ากับ O(1) แต่การดึงข้อมูลต่างๆ Asymptotic runtime จะเท่ากับ O(n) แต่เหตุผลที่ใช้ Stackนั้น เพราะ มีการเพิ่มหรือลบข้อมูลออกอยู่หลายครั้งเวลานำมาทำการเปรียบเทียบข้อมูลกับตัวถัดไป ซึ่งจากที่กล่าวมาข้างต้น Stack จึงเป็น Structure ที่เหมาะสมที่สุด

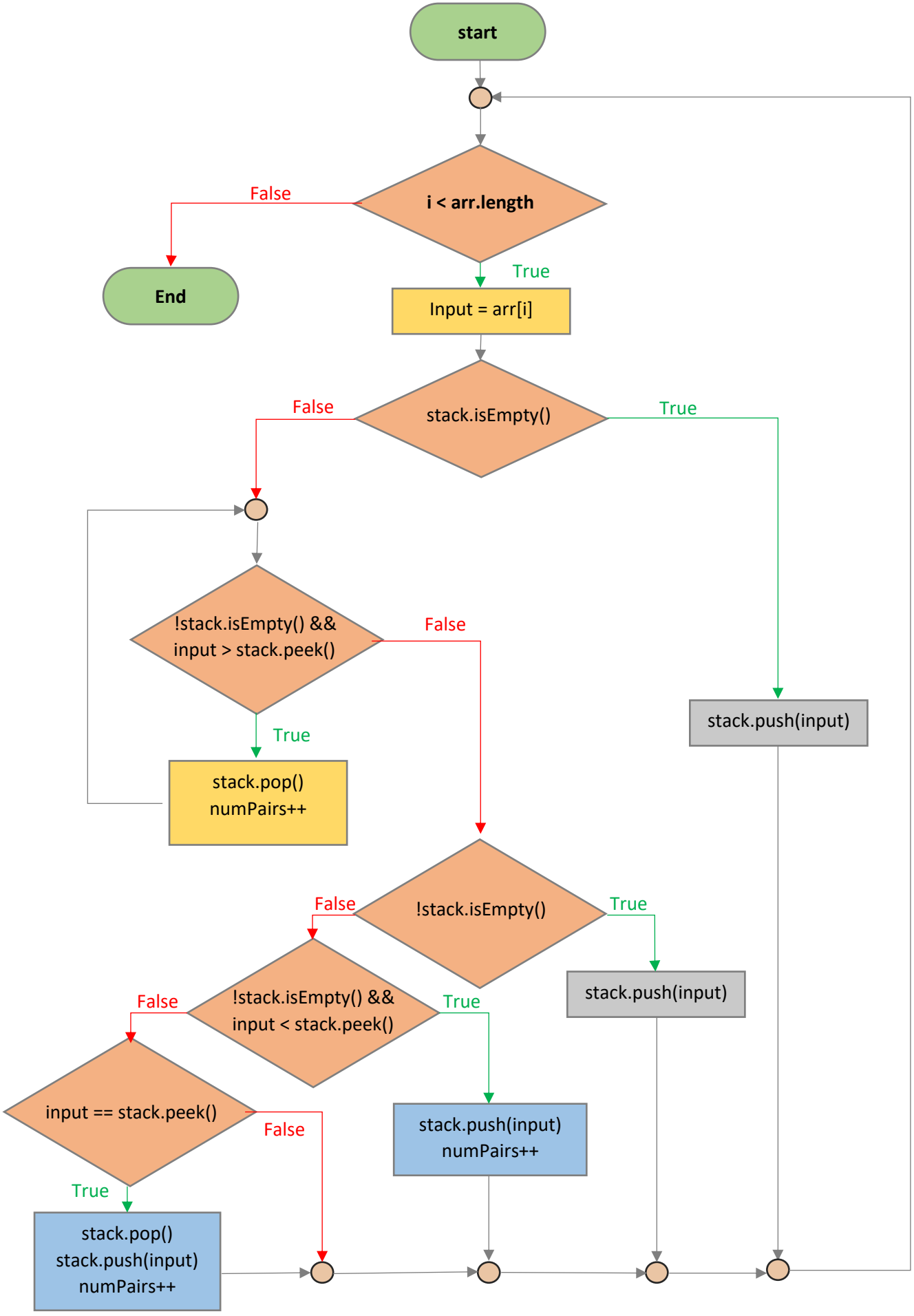
โดยใช้ Stack เก็บข้อมูลที่เป็น Object ซึ่งประกอบด้วย index คือตำแหน่งของต้นไม้ และความสูงของต้นไม้ที่เป็นจำนวนเต็มบวก ซึ่งขนาดของ Stack ในขณะนั้นๆจะมีขนาดเท่ากับจำนวนข้อมูลของต้นไม้ที่มีการเปรียบเทียบกันแล้วมีค่าความสูงของต้นไม้เรียงจากมากไปน้อย เนื่องจากถ้ามีข้อมูลเข้ามาและมีค่าที่มากกว่าตัวที่บนสุดในStack Algorithm นั้นจะทำการดึงข้อมูลตัวบนสุดนั้นออกและทำการนับว่าสามารถจับคู่ได้แค่แบบเดียว

Algorithm

วิธีตรวจสอบรูปแบบการจับคู่ต้นไม้ที่สามารถโหนไปได้ทั้งหมด ใช้วิธีการเปรียบเทียบค่าความสูงของต้นไม้ต้นที่อยู่บนสุดของ stack ด้วยเงื่อนไขต่างๆ แต่ถ้าหากไม่มีค่าใดใน stack จะใส่ค่าต้นไม้ต้นล่าสุดที่นำมาคิด โดยจะเปรียบเสมือนเป็นการเปลี่ยนต้นหลักในการหารูปแบบเป็นต้นใหม่

ลำดับขั้นตอน

1. รับค่าความสูงต้นไม้ใน ArrayList มาเป็น integer ชื่อ input โดยมี index เริ่มต้นที่ 0
2. ตรวจสอบว่ามีค่าใดใน stack หรือไม่ โดยใช้ if (stack.isEmpty()) หากไม่มี จะทำการ push ค่า input ลงไปใน stack แล้วไปคิดต้นไม้ต้นถัดไป (กลับไปที่ขั้นตอนที่ 1) หรือหากมีค่าใน stack จะทำการตรวจสอบค่าในขั้นตอนถัดไป
3. เมื่อตรวจสอบพบว่ามีค่าใน stack จะเปรียบเทียบค่า input กับค่าบนสุดของ stack (ต้นไม้ต้นล่าสุดที่ push ลง stack) หาก input มีความสูงมากกว่า จะทำการ pop stack ออกไปเรื่อยๆพร้อมกับนับจำนวนรูปแบบขึ้นทีละ 1 ทุกครั้งที่ pop stack จนกระทั่งค่า input มีความสูงน้อยกว่าหรือเท่ากับค่าบนสุดของ stack หรือจนไม่เหลือค่าใดใน stack แล้วจึงเปรียบเทียบค่าต่อในขั้นตอนถัดไป
4. ในขั้นตอนนี้จะแบ่งเงื่อนไขเป็น 3 กรณี ดังนี้
    4.1 กรณีที่ไม่เหลือค่าใดใน stack จะนำค่า input นั้น push ลงใน stack
    4.2 กรณีที่ต้นไม้ต้นบนสุดของ stack สูงกว่า input จะทำการ push ต้น input เข้าไปใน stack แล้วนับจำนวนรูปแบบเพิ่มขึ้น 1 ครั้ง
    4.3 กรณีที่ต้นไม้ต้นบนสุดของ stack สูงเท่ากับ input จะทำการ pop ค่าบนสุดของ stack ออกแล้ว push ต้น input ลงไปแทน และนับจำนวนรูปแบบขึ้น 1 ครั้ง
5. ย้อนกลับไปทำตั้งแต่ขั้นตอนที่ 1 จน index ของ ArrayList ครบจำนวนต้นไม้


start
i < arr.length
False
End
True
Input = arr[i]
stack.isEmpty()
False
True
stack.push(input)
!stack.isEmpty() &&
input > stack.peek()
False
True
stack.pop()
numPairs++
!stack.isEmpty()
False
True
stack.push(input)
!stack.isEmpty() &&
input < stack.peek()
False
True
stack.push(input)
numPairs++
input == stack.peek()
False
True
stack.pop()
stack.push(input)
numPairs++

โค้ดส่วนการรับข้อมูล

```
stack = new Stack<Tree>();
numPairs = 0;

boolean inputPhase = true;
int noTree = 0;
ArrayList<Integer> all = new ArrayList();
ArrayList<Integer> path[] = null;

while(inputPhase)
{   try
    {   Scanner in = new Scanner(System.in);
        System.out.println("Enter number of tree : ");
        noTree = in.nextInt();
        if(noTree<3)
            throw new Exception("Number of trees must higher than 2");

        path = new ArrayList[noTree];
        for(int i=0;i<noTree-1;i++)
        {
            path[i] = new ArrayList();
        }
        int i =0;
        int n=0;
        while(i<noTree)
        {   try
            {
                Scanner scan;
                System.out.println("Height of tree ("+(i+1)+") = ");
                scan = new Scanner(System.in);
                n = scan.nextInt();
                if(n<1)
                    throw new Exception("Height of tree must higher than 0");
                all.add(n);
                i++;
            }
            catch(Exception ex)
            {
                System.out.println(ex);
            }
        }
        inputPhase = false;
    }
    catch(Exception ex)
    {
        System.out.println(ex);
    }
}
```

ตรวจสอบว่าใส่จำนวนต้นไม้ตรงตามเงื่อนไขหรือไม่

ถ้าใส่ไม่ตรงตามเงื่อนไขจะบังคับให้ใส่ใหม่

สร้าง ArrayList เพื่อมาเก็บเส้นทางที่ไปได้ของแต่ละต้น

รับค่าเก็บใส่ ArrayList all

ตรวจสอบค่าความสูงของต้นไม้ ถ้าใส่ค่าผิดจะให้ใส่ค่าต้นเดิมใหม่

Object Tree

```
public Tree(int h,int i)
{
    height = h;
    index = i;
}
```

```
int input;
for(int i=0; i<all.size(); i++)
{
    input = all.get(i);
    if(stack.isEmpty())
    {
        stack.push(new Tree(input,i));
    }
    else if(!stack.isEmpty())
    {
        while(!stack.isEmpty() && input > stack.peek().getHeight())
        {
            path[stack.peek().getIndex()].add(i);

            stack.pop();
            numPairs++;
        }
        if(stack.isEmpty())
        {
            stack.push(new Tree(input,i));
        }
        else if(!stack.isEmpty() && input < stack.peek().getHeight())
        {
            path[stack.peek().getIndex()].add(i);

            stack.push(new Tree(input,i));
            numPairs++;
        }
        else if(input == stack.peek().getHeight())
        {
            path[stack.peek().getIndex()].add(i);

            stack.pop();
            stack.push(new Tree(input,i));
            numPairs++;
        }
    }
}
```

- ตรวจสอบว่าภายใน stack มีข้อมูลหรือไม่ ถ้าไม่มีให้เก็บค่าจาก input โดยสร้างเป็น Object Tree
- ตรวจสอบว่าภายใน stack มีข้อมูลหรือไม่ ถ้ามีและตัวบนสุดของ stack น้อยกว่า input แสดงว่า ตัวบนสุดของ stack จะมีเส้นทางเชื่อมกับตัว input เป็นเส้นสุดท้าย จึงสามารถ pop ออกจาก stack ได้ และทำซ้ำจนกว่าจะไม่เข้าเงื่อนไข
- ตรวจสอบว่าภายใน stack มีข้อมูลหรือไม่ ถ้าไม่มีแสดงว่า ต้นก่อนหน้าคิดเส้นทางครบทั้งหมดแล้ว จึงสามารถ push เข้า stack ได้โดยสร้างเป็น Object Tree
- ตรวจสอบว่าภายใน stack มีข้อมูลหรือไม่ ถ้ามีและตัวบนสุดของ stack มากกว่า input แสดงว่าตัวบนสุดของ stack มีเส้นทางเชื่อมกับตัว input และสามารถมีเส้นทางอื่นได้อีก จึงสามารถ push เข้า stack ได้โดยสร้างเป็น Object Tree
- ตรวจสอบว่าถ้าตัวบนสุดของ stack เท่ากับ input แสดงว่าตัวบนสุดของ stack มีเส้นทางเชื่อมกับตัว input และไม่สามารถมีเส้นทางอื่นได้อีก จึงต้อง pop ออกและ push input เข้า stack ได้โดยสร้างเป็น Object Tree

Example (input 5 trees : 19 17 15 20 20)

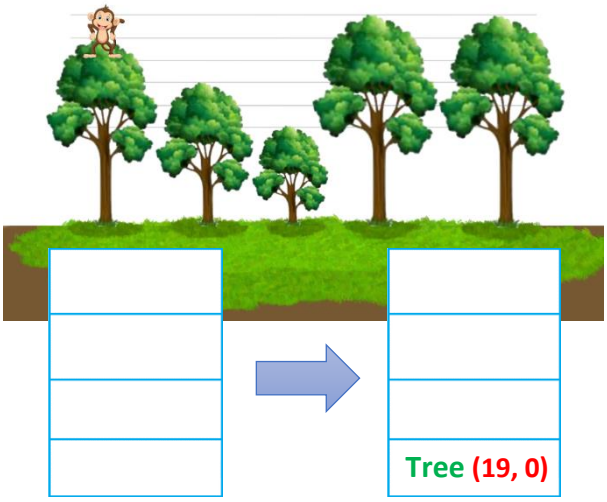
```
public Tree(int h,int i)
{
    height = h;
    index = i;
}

stack = new Stack<Tree>();
numPairs = 0;
```

เริ่มส่วนการคำนวณ

- เริ่มที่ต้นแรก ที่ index 0 สูง 19 ft
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่

ไม่มีข้อมูลใน stack

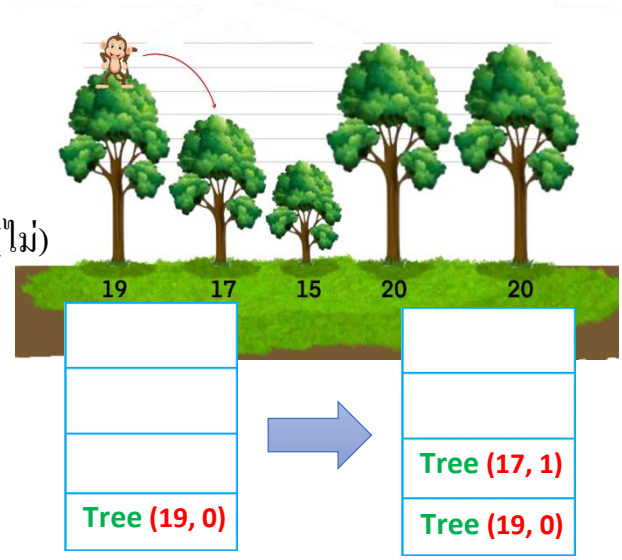
จึงสร้าง Tree(19, 0) แล้ว push ลง stack



- ขยับไปต้น 2 ที่ index 1 สูง 17 ft
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่ (มี)
- ตรวจสอบว่า input มากกว่าตัวบนสุดหรือไม่(ไม่)
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่ (มี)
- ตรวจสอบว่า input น้อยกว่าตัวบนสุดหรือไม่

มีข้อมูลใน stack และ input น้อยกว่า

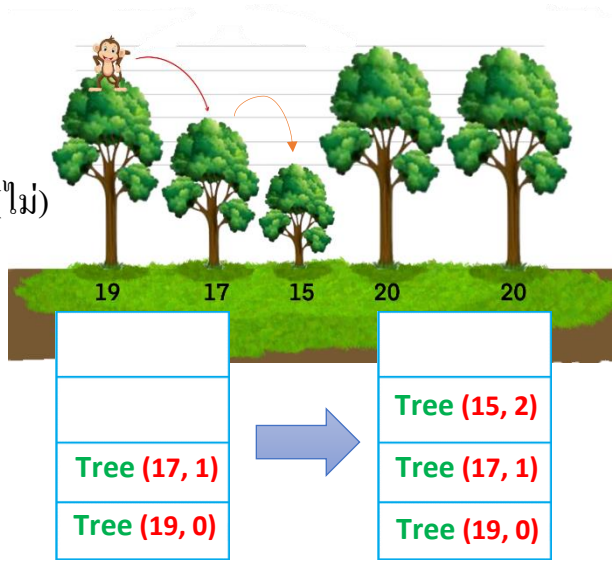
จึงสร้าง Tree(17, 1) แล้ว push ลง stack



- ขยับไปต้น 3 ที่ index 2 สูง 15 ft
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่ (มี)
- ตรวจสอบว่า input มากกว่าตัวบนสุดหรือไม่(ไม่)
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่ (มี)
- ตรวจสอบว่า input น้อยกว่าตัวบนสุดหรือไม่

มีข้อมูลใน stack และ input น้อยกว่า

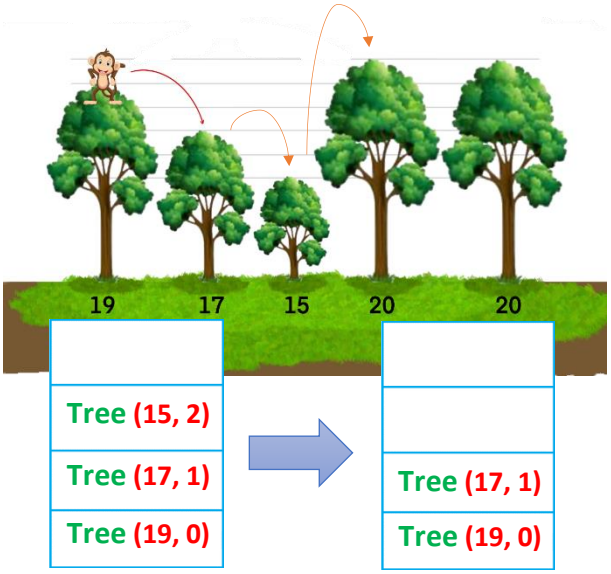
จึงสร้าง Tree(15, 2) แล้ว push ลง stack

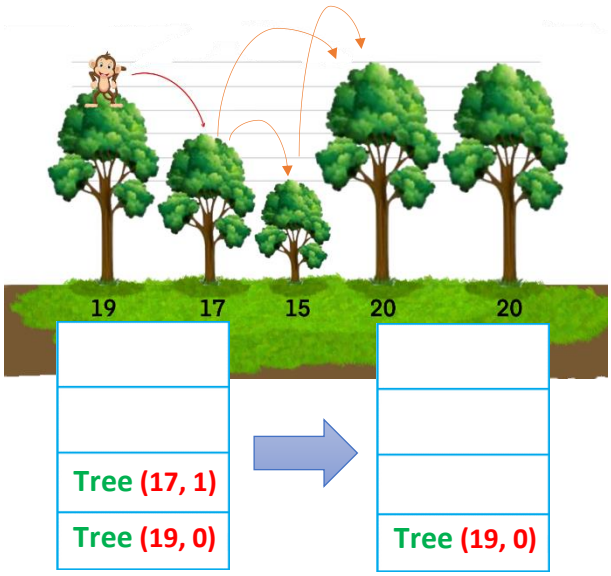


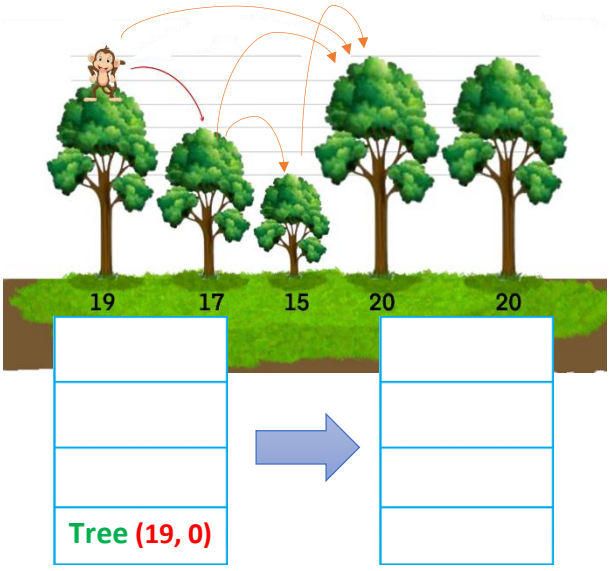
- ขยับไปต้น 4 ที่ index 3 สูง 20 ft
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่ (มี)
- ตรวจสอบว่า input มากกว่าตัวบนสุดหรือไม่

มีข้อมูลใน stack และ input มากกว่า

จึง pop ตัวบนสุดออก แล้วตรวจสอบซ้ำ



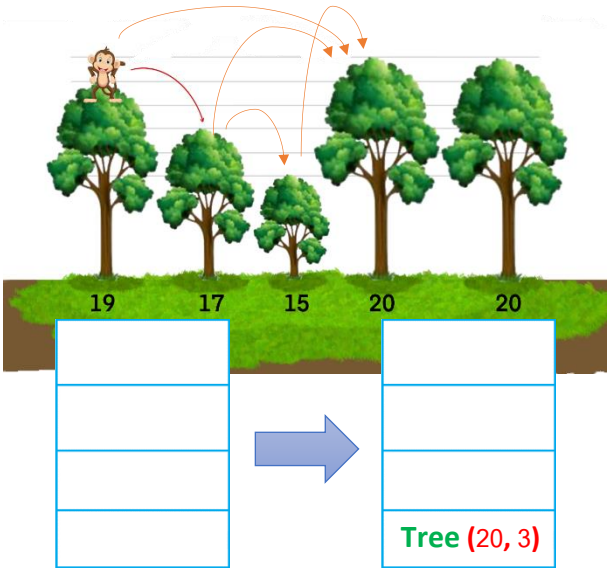




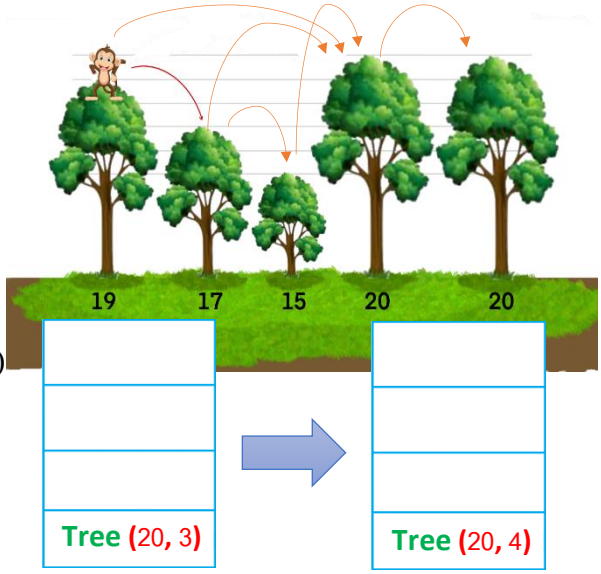
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่

ไม่มีข้อมูลใน stack

จึงสร้าง Tree(20, 3) แล้ว push ลง stack



- ขยับไปต้น 5 ที่ index 4 สูง 20 ft
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่ (มี)
- ตรวจสอบว่า input มากกว่าตัวบนสุดหรือไม่(ไม่)
- ตรวจสอบ stack ว่ามีข้อมูลหรือไม่ (มี)
- ตรวจสอบว่า input น้อยกว่าตัวบนสุดหรือไม่(ไม่)
- ตรวจสอบว่า input เท่ากับตัวบนสุดหรือไม่



Input เท่ากับ ตัวบนสุดของ stack

จึง pop ตัวบนสุดออก แล้ว push Tree(20,4)

```
Enter number of tree [Enter 0 to exit]:
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
--Solution--

Tree( 1),( 19) ft --> Tree( 2),( 17) ft
Tree( 1),( 19) ft --> Tree( 4),( 20) ft
Tree( 1) pair = 2
----------------------------------------
Tree( 2),( 17) ft --> Tree( 3),( 15) ft
Tree( 2),( 17) ft --> Tree( 4),( 20) ft
Tree( 2) pair = 2
----------------------------------------
Tree( 3),( 15) ft --> Tree( 4),( 20) ft
Tree( 3) pair = 1
----------------------------------------
Tree( 4),( 20) ft --> Tree( 5),( 20) ft
Tree( 4) pair = 1
----------------------------------------
Total Pair = 6
```

Output ที่ได้เกิดจากการจัดเรียงมาเรียบร้อยแล้ว

เพื่อให้ง่ายต่อการดูผลลัพธ์ ที่เป็นไปได้ของแต่ละต้น

เหตุผลที่ใช้วิธีนี้แทนที่วิธี BruteForce

เพราะ algorithm ของเรานั้นเป็นการตรวจสอบจากการเปรียบเทียบข้อมูลที่อยู่ในstackในกรณีที่ข้อมูลใน stackนั้นมีค่าน้อยกว่าข้อมูลตัวถัดไปที่รับเข้ามา เราก็จะทำการดึงข้อมูลตัวนั้นออกจาก stack แล้วนับว่าเป็น 1 คู่ที่เป็นไปได้และจะเป็นการตัดว่าตัวนั้นไม่สามารถไปจับคู่กับตัวอื่นได้อีกเพราะว่าเนื่องจากรูปแบบที่เป็นไปได้ของการโหนไปต้นที่สูงกว่ามีได้แค่กรณีเดียว วิธีนี้จะสามารถตัดเคสที่คำนวณซ้ำโดยไม่จำเป็นออกได้ซึ่งจะทำงานได้ดีกว่าการใช้ Brute Force ที่จะทำการตรวจต้นหลักทุกๆต้นแล้วค่อยส่งผลลัพธ์ออกมา ซึ่ง asymptotic runtime ของ BruteForce คือ $O(n^2)$ ซึ่ง asymptotic runtime ของ algorithm นี้จะน้อยกว่า

ข้อจำกัดโปรแกรม

1.ข้อมูลที่กรอกในโปรแกรมสามารถใส่ได้แค่เป็นตัวเลขจำนวนเต็มไม่สามารถใส่เลขที่เป็นทศนิยมได้

2.ถ้าในกรณีที่เกิดการใส่ข้อมูลผิดจะต้องใส่ข้อมูลใหม่หมดตั้งแต่แรก

แหล่งอ้างอิง

- โปรแกรม

  - https://stackoverflow.com/questions/15379466/trouble-with-a-stack-based-algorithm

-รูปภาพ

  - https://www.shutterstock.com/th/image-vector/cute-monkey-cartoon-361764017
  - https://www.shutterstock.com/th/image-vector/beautiful-tree-on-white-background-498661822