# Render

สังเกตว่าสำหรับ Subtree ใดๆ ส่ามารถหาได้ว่าจะแปลง Subtree นี้ให้เป็นต้นไม้ไบนารีที่มีความลึกต่ำสุดได้เท่าใดโดยไม่ต้องพิจารณาจุดยอดนอก Subtree นั้นๆ

เราจึงสามารถกำหนด $H[x]$ เป็นความลึกที่ต่ำที่สุดที่เป็นไปได้ของต้นไม้ไบนารีที่มีรากเป็น $x$

สมมิตว่าเราคำนวณ $H[c]$ สำหรับทุกจุดยอดลูก $c$ ของ $x$ ไปแล้ว หาก $x$ ไม่มีลูกหรือมีลูกไม่เกิน 2 จุดยอด ความลึกเป็นเพียง ค่า $H$ ของลูกที่สูงสุดบวก 1 แต่หากมีมากกว่า 2 จุดยอดจะต้องพิจารณาวิธีสร้างต้นไบนารีจากจุดยอดลูกเหล่านี้ที่จะทำให้ความลึกต่ำสุดที่จะเป็นไปได้

### Greedy Algorithm

เราจะพิสูจน์ว่าการในการสร้างต้นไม้ไบนารีที่ตื้นที่สุดที่มีรากเป็น $x$ หาก $x$ มีลูกเกิน 2 ลูก จะสามารถเริ่มโดยการสร้างจุดยอดพิเศษเป็น parent ของสองจุดยอดลูกที่มีค่า $H$ ต่ำสุดเสมอ

นั่นคือหากจุดยอดลูก $a$ และ $b$ เป็นจุดยอดที่ $H[a]$ กับ $H[b]$ มีค่าที่น้อยที่สุดที่เป็นไปได้สองค่า เราจะพิสูจน์ว่ามีต้นไบนารีที่มีรากเป็น $x$ ที่มีความลึก $H[x]$ (กล่าวคือตื้นสุดที่เป็นไปได้) โดยมี $a$ และ $b$ เป็น sibling (มี parent เป็นจุดยอดพิเศษเดียวกัน)

พิจารณาต้นไม้ไบนารี $O$ ใดๆ ที่มีความลึกน้อยที่สุด เราจะพิสูจน์ว่ามีต้นไม้ $O'$ ที่มีความลึกไม่เกิน $O$ โดยมี $a$ และ $b$ เป็น sibling

ให้ $d_O(z)$ เป็นความลึกของ $z$ ใน $O$

ให้ $a'$ เป็นหนึ่งในสองจุดยอด $a$ หรือ $b$ ดังกล่าวที่มี $d_O(a')$ มากสุดและ $b'$ เป็นอีกจุดยอด นั่นคือ $d_O(a') \geq d_O(b')$

สังเกตได้ว่าทุกจุกยอดลูกจะมี sibling ในต้นไบนารี $O$ เพราะในขั้นตอนการเพิ่มจุดยอดพิเศษ เพราะหากจุดยอดพิเศษนั้นไม่มี sibling แสดงว่าก่อนการเพิ่ม parent ของทั้งสองลูกมีเพียงสองลูกอยู่แล้ว ดังนั้นการเพิ่มจุดยอดพิเศษนี้จึงไม่จำเป็นและเพิ่มความลึกของ $O$ ซึ่งขัดกับการที่ $O$ มีความลึกน้อยสุดที่เป็นไปได้

ดังนั้นเราจะสามารถจำแนกเป็น 2 กรณี

1. $a'$ และ $b'$ เป็น sibling กัน
2. $a'$ มี sibling ที่ไม่ใช่ $b'$

หากเข้ากรณีแรกสามารถเลือก $O'=O$ ตามที่ต้องการพิสูจน์

หากเข้ากรณีที่สอง ให้ sibling ของ $a'$ และ $b'$ เป็น $c$ และ $d$ ตามลำดับ เราสามารถเลือก $O'$ เป็นต้น $O$ ที่สลับ $b'$ มาเป็น sibling $a'$ และ $c$ ไปเป็น sibling $d$ ทั้งนี้จะทำให้ความลึกจุดยอด descendent ลึกสุดของ $b'$ ในต้นไบนารีใหม่เป็น $d_O(a') + H[b']$ และของ $d$ เป็น $d_O(b') + H[d]$ จากเดิม $d_O(b') + H[b']$ และ $d_O(a') + H[d]$ ตามลำดับ เนื่องจาก $H[b'] \leq H[d]$ และ $d_O(b') \leq d_O(a')$ (จากนิยามของ $a', b'$ และ $H$) จะได้ว่า $\max(d_O(a') + H[b'], d_O(b') + H[d]) \leq \max(d_O(b') + H[b'], d_O(a') + H[d])$

ข้อนี้ถามว่าหากแปลงต้นไม้ $T$ ให้เป็นต้นไม้ไบนารี $S$ จะทำให้ $S$ มีความลึกน้อยสุดได้เท่าไหร่ โดยในการแปลงจะจะต้องใช้วิธีการสร้างจุดยอดใหม่เป็น parent ของสองจุดยอดใดๆ ที่เคยเป็น sibling กัน

สังเกตว่าสำหรับ Subtree ใดๆ ส่ามารถหาได้ว่าจะแปลง Subtree นี้ให้เป็นต้นไม้ไบนารีที่มีความลึกต่ำสุดได้เท่าใดโดยไม่ต้องพิจารณาจุดยอดนอก Subtree นั้นๆ

เราจึงสามารถกำหนด $H[x]$ เป็นความลึกที่ต่ำที่สุดที่เป็นไปได้ของต้นไม้ไบนารีที่มีรากเป็น $x$

สมมิตว่าเราคำนวณ $H[c]$ สำหรับทุกจุดยอดลูก $c$ ของ $x$ ไปแล้ว หาก $x$ ไม่มีลูกหรือมีลูกไม่เกิน 2 จุดยอด ความลึกเป็นเพียง ค่า $H$ ของลูกที่สูงสุดบวก 1 แต่หากมีมากกว่า 2 จุดยอดจะต้องพิจารณาวิธีสร้างต้นไบนารีจากจุดยอดลูกเหล่านี้ที่จะทำให้ความลึกต่ำสุดที่จะเป็นไปได้

## Greedy Algorithm

เราจะพิสูจน์ว่าการในการสร้างต้นไม้ไบนารีที่ตื้นที่สุดที่มีรากเป็น $x$ หาก $x$ มีลูกเกิน 2 ลูก จะสามารถเริ่มโดยการสร้างจุดยอดพิเศษเป็น parent ของสองจุดยอดลูกที่มีค่า $H$ ต่ำสุดเสมอ

นั่นคือหากจุดยอดลูก $a$ และ $b$ เป็นจุดยอดที่ $H[a]$ กับ $H[b]$ มีค่าที่น้อยที่สุดที่เป็นไปได้สองค่า เราจะพิสูจน์ว่ามีต้นไบนารีที่มีรากเป็น $x$ ที่มีความลึก $H[x]$ (กล่าวคือตื้นสุดที่เป็นไปได้) โดยมี $a$ และ $b$ เป็น sibling (มี parent เป็นจุดยอดพิเศษเดียวกัน)

พิจารณาต้นไม้ไบนารี $O$ ใดๆ ที่มีความลึกน้อยที่สุด เราจะพิสูจน์ว่ามีต้นไม้ $O'$ ที่มีความลึกไม่เกิน $O$ โดยมี $a$ และ $b$ เป็น sibling

ให้ $d_O(z)$ เป็นความลึกของ $z$ ใน $O$

ให้ $a'$ เป็นหนึ่งในสองจุดยอด $a$ หรือ $b$ ดังกล่าวที่มี $d_O(a')$ มากสุดและ $b'$ เป็นอีกจุดยอด นั่นคือ $d_O(a') \geq d_O(b')$

สังเกตได้ว่าทุกจุกยอดลูกจะมี sibling ในต้นไบนารี $O$ เพราะในขั้นตอนการเพิ่มจุดยอดพิเศษ เพราะหากจุดยอดพิเศษนั้นไม่มี sibling แสดงว่าก่อนการเพิ่ม parent ของทั้งสองลูกมีเพียงสองลูกอยู่แล้ว ดังนั้นการเพิ่มจุดยอดพิเศษนี้จึงไม่จำเป็นและเพิ่มความลึกของ $O$ ซึ่งขัดกับการที่ $O$ มีความลึกน้อยสุดที่เป็นไปได้

ดังนั้นเราจะสามารถจำแนกเป็น 2 กรณี

1. $a'$ และ $b'$ เป็น sibling กัน
2. $a'$ มี sibling ที่ไม่ใช่ $b'$

หากเข้ากรณีแรกสามารถเลือก $O'=O$ ตามที่ต้องการพิสูจน์

หากเข้ากรณีที่สอง ให้ sibling ของ $a'$ และ $b'$ เป็น $c$ และ $d$ ตามลำดับ เราสามารถเลือก $O'$ เป็นต้น $O$ ที่สลับ $b'$ มาเป็น sibling $a'$ และ $c$ ไปเป็น sibling $d$ ทั้งนี้จะทำให้ความลึกจุดยอด descendent ลึกสุดของ $b'$ ในต้น

d_O(b') + H[d]) \leq \max(d_O(b') + H[b'], d_O(a') + H[d]) = d_O(a') + H[d]$ กล่าวคือการสลับนี้จะไม่เพิ่มความลึกของ $O'$ เมื่อเทียบกับ $O$ เพราะความลึกที่มากสุดในบรรดาจุดยอดที่ถูกกระทบไม่เพิ่ม

เพราะฉะนั้นจึงสรุปได้ว่าไม่ว่ากรณีใดๆ จะมีต้นไม้ไบนารี $O'$ ที่เลือก $a$ และ $b$ มาเป็น sibling กันและมีความลึกต่ำสุดที่เป็นไปได้

เมื่อเราเลือกสร้างจุดยอดพิเศษ $c$ มาเป็นลูกใหม่ของ $x$ ที่มีลูกเป็น $a$ กับ $b$ เราสามารถใช้เหตุผลแบบเดิมเลือกสองลูกที่มี $H$ ต่ำสุดมาเป็นลูกของจุดยอดพิเศษใหม่อีกรอบเรื่อยๆ จน $x$ เหลือลูกเพียงสองลูกและจะได้ว่า $H[x]$ คือ $\max(H[y],H[z])$ ของลูกที่เหลืออยู่ $y,z$

### Algorithm เต็ม

1. อ่าน input โดยเก็บ Adjacency List ของแต่ละจุดยอดว่ามีจุดยอดไหนเป็นลูกบ้าง
2. คำนวณ $H[x]$ สำหรับทุก $x$ โดยใช้วิธี recursive แบบด้านบน
   - หาก $x$ ไม่มีลูกจะได้ $H[x]=0$ และรีเทิร์น
   - มิเช่นนั้น $x$ มีลูกอย่างน้อย 1 ลูก ให้คำนวณ $H[c]$ สำหรับทุกลูก $c$ ก่อน
   - เอา $H[c]$ ทุกค่ามาไว้ใน minimum priority queue
   - เลือกสองค่าต่ำสุดใน priority queue เพื่อเอาออกและนำมาเป็นลูกของจุดพิเศษใหม่ (หากค่าเดิมคือ $A$ และ $B$ จุดยอดพิเศษใหม่จะมีความลึก $max(A,B)+1$ เพราะเพิ่มความลึกที่มากสุดในนั้นไป 1) และใส่จุดยอดใหม่เข้าไปใน queue โดยทำซ้ำจนใน queue เหลือไม่เกิน 2 ค่า
   - เลือกค่าที่มาก $M$ ที่สุดที่เหลืออยู่ใน queue และตั้ง $H[x]=M+1$

3. คำตอบคือ $H[1]$

การอ่านข้อนำเข้าและเก็บ Adjacency List ใช้เวลา $\mathcal{O}(N)$

การคำนวณ $H$ ใช้เวลา $\mathcal{O}(C_x \log C_x)$ สำหรับทุก $x$ เมื่อ $C_x$ คือจำนวนลูกของ $x$ เพราะเกิดการ push และ pop จาก priority queue $\mathcal{O}(C_x)$ รอบ

ทั้งหมดจึงใช้เวลา $\mathcal{O}(N) + \Sigma_{x=1}^{N} \mathcal{O}(C_x \log C_x) = \mathcal{O}(N \log N)$

#### ตัวอย่างโค้ดสำหรับการคำนวณ $H$ ในขั้นตอนที่ 2

ไบนารีใหม่เป็น $d_O(a') + H[b']$ และของ $d$ เป็น $d_O(b') + H[d]$ จากเดิม $d_O(b') + H[b']$ และ $d_O(a') + H[d]$ ตามลำดับ เนื่องจาก $H[b'] \leq H[d]$ และ $d_O(b') \leq d_O(a')$ (จากนิยามของ $a', b'$ และ $H$) จะได้ว่า $\max(d_O(a') + H[b'], d_O(b') + H[d]) \leq \max(d_O(b') + H[b'], d_O(a') + H[d]) = d_O(a') + H[d]$ กล่าวคือการสลับนี้จะไม่เพิ่มความลึกของ $O'$ เมื่อเทียบกับ $O$ เพราะความลึกที่มากสุดในบรรดาจุดยอดที่ถูกกระทบไม่เพิ่ม

เพราะฉะนั้นจึงสรุปได้ว่าไม่ว่ากรณีใดๆ จะมีต้นไม้ไบนารี $O'$ ที่เลือก $a$ และ $b$ มาเป็น sibling กันและมีความลึกต่ำสุดที่เป็นไปได้

เมื่อเราเลือกสร้างจุดยอดพิเศษ $c$ มาเป็นลูกใหม่ของ $x$ ที่มีลูกเป็น $a$ กับ $b$ เราสามารถใช้เหตุผลแบบเดิมเลือกสองลูกที่มี $H$ ต่ำสุดมาเป็นลูกของจุดยอดพิเศษใหม่อีกรอบเรื่อยๆ จน $x$ เหลือลูกเพียงสองลูกและจะได้ว่า $H[x]$ คือ $\max(H[y], H[z])$ ของลูกที่เหลืออยู่ $y, z$

## Algorithm เต็ม

1. อ่าน input โดยเก็บ Adjacency List ของแต่ละจุดยอดว่ามีจุดยอดไหนเป็นลูกบ้าง

2. คำนวณ $H[x]$ สำหรับทุก $x$ โดยใช้วิธี recursive แบบด้านบน

   - หาก $x$ ไม่มีลูกจะได้ $H[x] = 0$ และรีเทิร์น
   - มิเช่นนั้น $x$ มีลูกอย่างน้อย 1 ลูก ให้คำนวณ $H[c]$ สำหรับทุกลูก $c$ ก่อน
   - เอา $H[c]$ ทุกค่ามาไว้ใน minimum priority queue
   - เลือกสองค่าต่ำสุดใน priority queue เพื่อเอาออกและนำมาเป็นลูกของจุดพิเศษใหม่ (หากค่าเดิมคือ $A$ และ $B$ จุดยอดพิเศษใหม่จะมีความลึก $max(A, B) + 1$ เพราะเพิ่มความลึกที่มากสุดในนั้นไป 1) และใส่จุดยอดใหม่เข้าไปใน queue โดยทำซ้ำจนใน queue เหลือไม่เกิน 2 ค่า
   - เลือกค่าที่มาก $M$ ที่สุดที่เหลืออยู่ใน queue และตั้ง $H[x] = M + 1$

3. คำตอบคือ $H[1]$

การอ่านข้อนำเข้าและเก็บ Adjacency List ใช้เวลา $\mathcal{O}(N)$

การคำนวณ $H$ ใช้เวลา $\mathcal{O}(C_x \log C_x)$ สำหรับทุก $x$ เมื่อ $C_x$ คือจำนวนลูกของ $x$ เพราะเกิดการ push และ pop จาก priority queue $\mathcal{O}(C_x)$ รอบ

ทั้งหมดจึงใช้เวลา $\mathcal{O}(N) + \Sigma_{x=1}^{N} \mathcal{O}(C_x \log C_x) = \mathcal{O}(N \log N)$

**ตัวอย่างโค้ดสำหรับการคำนวณ $H$ ในขั้นตอนที่ 2**

```
int H(int x) {
  if (child[x].size() == 0) {
    return 0;
  }

  std::priority_queue<int, std::vector<int>, std::gre

  for (int i = 0; i < child[x].size(); i++)
    q.push(H(child[x][i]));

  while (q.size() > 2) {
    int A = q.top();
    q.pop();
    int B = q.top();
    q.pop();
    q.push(max(A, B) + 1);
  }

  int H_x = 0;
  while (q.size() > 0) {
    H_x = max(H_x, q.top() + 1);
    q.pop();
  }

  return H_x;
}
```

**PROGRAMMING.IN.TH**

โปรแกรมมิ่งอินทีเอช ศูนย์รวมของโจทย์และเนื้อหาสำหรับ การเขียน
โปรแกรมเพื่อการแข่งขัน และวิทยาการคอมพิวเตอร์

ค้นหาโจทย์