Algorithm Design and Analysis

Week 9: Minimum Spanning Trees, Prim's algorithm, Kruskal's algorithm

Adisak Supeesun

10 February 2022

Minimum Spanning Trees

Prim’s Algorithm

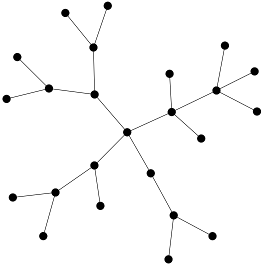
Kruskal’s Algorithm

Tree

Tree คือ connected graph ที่ไม่มี cycle

- connected graph: ทุกคู่ของจุดยอด u, v ในกราฟ มีเส้นทางจาก u ไป v อย่างน้อย 1 เส้นทาง ≥ 1 เส้นทาง
- ไม่มี cycle: ทุกคู่ของจุดยอด u, v ในกราฟ มีเส้นทางจาก u ไป v ไม่เกิน 1 เส้นทาง ≤ 1 เส้นทาง

$\therefore$ ทุกคู่ของจุดยอด u, v ในกราฟ มีเส้นทางจาก u ไป v 1 เส้นทาง



Note

จำนวนเส้นเชื่อมใน tree ที่มี n จุดยอด = n - 1

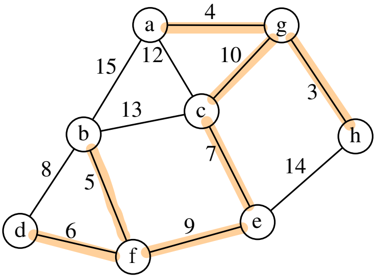
Minimum Spanning Tree

<u>นิยาม</u> spanning tree (SPT) ของกราฟ G คือ connected subgraph ของ G ที่ไม่มี cycle และครอบคลุมทุกจุดยอดของ G

<u>ปัญหา Minimum Spanning Tree (MST)</u>

ให้ undirected connected graph $G = (V, E)$ ที่เส้นเชื่อม $(u, v) \in E$ ใดๆ มีน้ำหนัก $w(u, v) > 0$ ต้องการหา spanning tree ของ G ที่มีน้ำหนักรวมของเส้นเชื่อมน้อยที่สุด

<u>Ex.1</u> จงหา MST ของกราฟต่อไปนี้

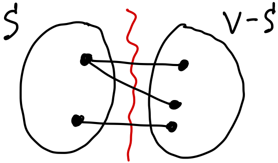


น้ำหนักรวม = 3 + 4 + 5 + 6 + 7 + 9 + 10 = 44

Cut Property (สมมติว่าเส้นเชื่อมทุกเส้นใน G มีน้ำหนักไม่เท่ากันเลย)

<u>นิยาม (cut)</u> สำหรับกราฟ $G = (V, E)$ เราจะเรียก เซต $S \subset V$ ใดๆ ว่า "cut"

และเรียกเส้นเชื่อมที่เชื่อมระหว่างจุดยอดใน S กับจุดยอดใน $V - S$ ว่า "เส้นเชื่อมข้าม cut"



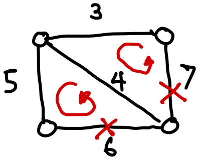
Theorem 1

สำหรับ cut S ใดๆ ในกราฟ $G = (V, E)$, เส้นเชื่อมข้าม cut S ที่มีน้ำหนักน้อยที่สุด ต้องอยู่ใน MST ของ G

 (สมมติว่าเส้นเชื่อมทุกเส้นใน G มีน้ำหนักไม่เท่ากันเลย)

Theorem 2

สำหรับ cycle C ใดๆ ในกราฟ $G = (V, E)$, *เส้นเชื่อมที่มีน้ำหนักมากที่สุดใน C ต้องไม่อยู่ใน MST ของ G*



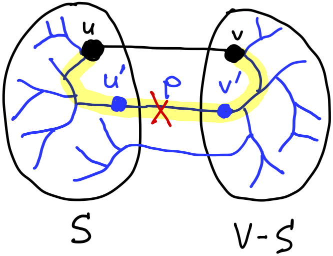
Proof of theorem 1 (by contradiction)

ให้ S เป็น cut ในกราฟ $G = (V, E)$ และ (u, v) เป็นเส้นเชื่อมซึ่งมีน้ำหนักน้อยที่สุดที่ข้าม cut ดังกล่าว

สมมติว่า (u, v) ไม่อยู่ใน MST T^* ของ G

ให้ P เป็น path จาก u ไป v ใน T^*

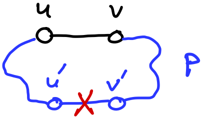
และ ให้ (u', v') เป็นเส้นเชื่อมเส้นแรกใน P ที่ข้าม cut S โดยที่ $u' \in S$, $v' \in V - S$



พิจารณา $T' = T^* \cup \{(u,v)\} - \{(u',v')\}$

จะเห็นว่า 1) T' ไม่มี cycle

เพราะ cycle เดียวที่มีใน $T^* \cup \{(u,v)\}$ คือ

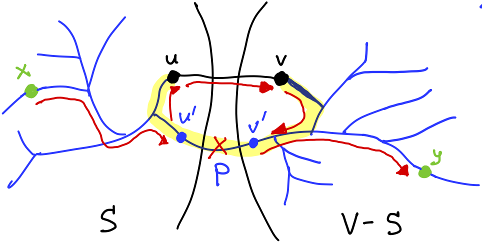


เมื่อลบ (u',v') ออกไป กราฟที่เหลือจึงไม่มี cycle

2) T' connected

เพราะ สำหรับทุกคู่ของจุดยอด x, y ที่ path จาก x ไป y บน T^* ไม่ผ่าน (u',v') path นี้จะยังคงอยู่ใน T'

ส่วนคู่ของจุดยอด x, y ที่ path จาก x ไป y บน T^* ผ่าน (u',v') เราสามารถ reroute เส้นทางจาก x ไป y ได้ดังภาพ



จาก 1) และ 2) จะได้ว่า T' เป็น tree และครอบคลุมทุกจุดยอด

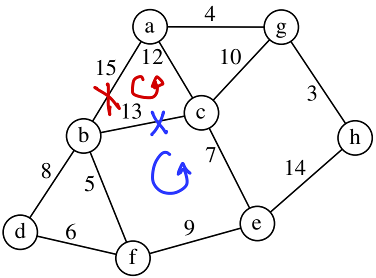
$\therefore$ T' เป็น spanning tree (SPT) ของ G

พิจารณาน้ำหนักรวมของ T'

$$w(T') = w(T^*) + \underbrace{w(u,v) - w(u',v')}_{\text{ติดลบ}}$$
$$< w(T^*)$$

เกิดข้อขัดแย้ง T^* เป็น MST (SPT ที่มีน้ำหนักน้อยที่สุด) ของ G
แต่ดันมี SPT T' ที่น้ำหนักน้อยกว่า T^*

ดังนั้น ข้อสมมติตอนต้นเป็นเท็จ แสดงว่า (u,v) ต้องอยู่ใน MST ∎



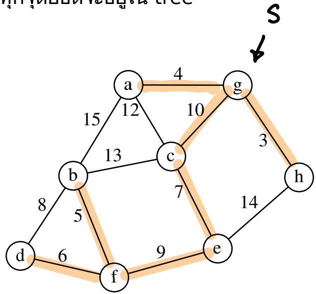
เราสามารถนำ cut property และ cycle property
มาช่วยในการออกแบบอัลกอริทึมสำหรับหา MST ได้

เช่น พิจารณาทุก cycle ในกราฟ ลบเส้นเชื่อมที่หนักที่สุดใน cycle ออกไป

Prim’s Algorithm

ค่อยๆขยาย tree เริ่มจากจุดยอดใดก็ได้

ในแต่ละครั้งที่ขยาย จะทำการเลือกเส้นเชื่อมที่มีน้ำหนักน้อยที่สุดซึ่งเชื่อมระหว่างจุดยอดที่อยู่ใน tree กับจุดยอดที่อยู่นอก tree ทำไปเรื่อยๆจนกว่าทุกจุดยอดจะอยู่ใน tree



$4+10+3+7+9+5+6 = 44$

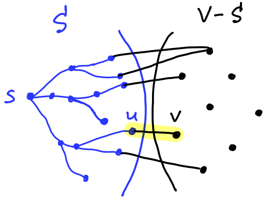
Correctness of Prim's algorithm

Theorem 3

Prim's algorithm ให้คำตอบเป็น MST

<u>Proof</u> พิจารณาการขยาย tree ในรอบใดๆ ของ Prim's algorithm

สมมติว่าอัลกอริทึมเลือกขยายออกไปทางเส้นเชื่อม (u, v)



ด้วยเงื่อนไขในการเลือกเส้นเชื่อมของอัลกอริทึม จะได้ว่า (u,v) ต้องเป็นเส้นเชื่อมที่มีน้ำหนัก
น้อยที่สุดในบรรดาเส้นเชื่อมที่เชื่อมระหว่างจุดยอดใน tree กับจุดยอดนอก tree

ถ้าให้ S' เป็น cut ซึ่งประกอบด้วยจุดยอดทั้งหมดใน tree ก่อนที่อัลกอริทึมจะเลือก (u,v)

จะได้ว่า (u,v) เป็นเส้นเชื่อมข้าม cut S' ที่มีน้ำหนักน้อยที่สุด

ด้วย cut property , (u,v) ต้องอยู่ใน MST

นั่นคือ เส้นเชื่อมใดๆ ที่ Prim's algorithm เลือก ต้องเป็นเส้นเชื่อมใน MST

และเนื่องจาก prim's algorithm ทำงาน $n-1$ ขั้นตอน (สำหรับ input graph ที่มี n จุดยอด) โดยที่ทุกขั้นตอนเลือกเส้นเชื่อมไม่ซ้ำกันเลย เราสามารถสรุปได้ว่า

เส้นเชื่อมทั้งหมดใน MST ซึ่งมี $n-1$ เส้น ต้องถูกเลือกมาเป็นคำตอบของอัลกอริทึม ■

Implementation and Running time of Prim's algorithm

รับ connected undirected graph $G = (V, E)$, น้ำหนักเส้นเชื่อม $w(e)$ ของทุกเส้นเชื่อม $e \in E$

เลือกจุดยอด $s \in V$

$S \leftarrow \{s\}$

$E' \leftarrow \emptyset$

while $S \neq V$

  หาเส้นเชื่อม (u, v) ที่ $u \in S, v \notin S$ ที่มีน้ำหนักน้อยที่สุด

  $E' \leftarrow E' \cup \{(u, v)\}$

  $S \leftarrow S \cup \{v\}$

end while

return (V, E')

$n-1 \leq m \leq n^2$

รับ connected undirected graph $G = (V, E)$, น้ำหนักเส้นเชื่อม $w(e)$ ของทุกเส้นเชื่อม $e \in E$

เลือกจุดยอด $s \in V$

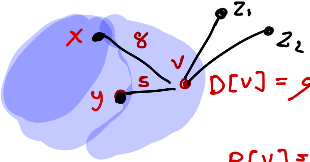
$S \leftarrow \{s\}$

$E' \leftarrow \emptyset$

$D[v] \leftarrow \infty, \quad \forall v \in V$ และ $D[s] \leftarrow 0$

PQ.INSERT(v, $D[v]$), $\forall v \in V$

$p[v] \leftarrow v, \quad \forall v \in V$



$D[v] = \cancel{\infty}\ \cancel{8}\ 5$

$p[v] = \cancel{x}\ y$

while PQ is not empty

    $v \leftarrow PQ$.DELETE-MIN()

    $S \leftarrow \{v\}$

    $E' \leftarrow E' \cup \{(p[v], v)\}$

    for each เส้นเชื่อม $(v, z) \in E$ ที่ติดกับ v และ $z \notin S$

        if $D[z] > w(v, z)$

            $D[z] \leftarrow w(v, z)$

            PQ.UPDATE-KEY(z, $D[z]$)

            $p[z] \leftarrow v$

        end if

    end for

end while

return (V, E')

implement แบบเดียวกับ Dijkstra
ต่างกันแค่ตรงเงื่อนไขส่วนนี้

$\therefore$ เวลาในการทำงานเท่ากับ Dijstra

$O((m+n)\log n) = O(m \log n)$

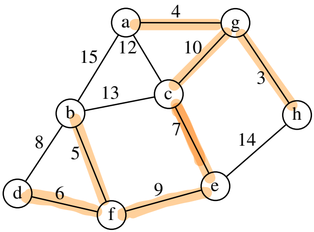
$n \leq m+1$

Kruskal's Algorithm

เริ่มต้นให้ กราฟคำตอบเป็นกราฟว่าง (ที่ไม่มีเส้นเชื่อม)

พิจารณาเส้นเชื่อมทีละเส้น โดยเริ่มจากเส้นที่เบาที่สุดไปจนถึงเส้นที่หนักที่สุด

เติมเส้นเชื่อมลงไปในกราฟคำตอบทีละเส้นตามลำดับ (ถ้าเส้นเชื่อมดังกล่าวไม่ทำให้เกิด cycle ในกราฟคำตอบ)



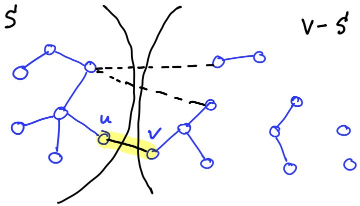
Theorem 4

Kruskal’s algorithm ให้คำตอบเป็น MST

<u>Proof</u> พิจารณาการทำงานของอัลกอริทึมในขณะที่กำลังพิจารณาเส้นเชื่อม (u, v)

<u>กรณี 1</u> อัลกอริทึมเลือกเพิ่ม (u,v) ลงไปในกราฟคำตอบ

ให้ S เป็น cut ที่ประกอบด้วยจุดยอดทั้งหมดใน connected component ของ u



แสดงว่าก่อนเลือก (u,v) ยังไม่มีเส้นเชื่อมข้าม cut S' เส้นใดถูกเติม/ทิ้ง
(u,v) จึงเป็นเส้นเชื่อมข้าม cut S' เส้นแรกที่อัลกอริทึมพิจารณา

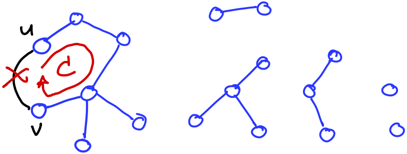
เนื่องจาก Kruskal's algorithm พิจารณาเส้นเชื่อมตามลำดับน้ำหนัก จากน้อยไปมาก เราจะได้ว่า (u,v) เป็นเส้นเชื่อมข้าม cut S' ที่มีน้ำหนักน้อยที่สุด

ด้วย cut property, (u,v) เป็นเส้นเชื่อมใน MST

<u>กรณี</u> 2 อัลกอริทึมเลือกไม่เติม (u,v) ลงในคำตอบ

การที่อัลกอริทึมไม่เลือก (u,v) เป็นเพราะถ้านำ (u,v) ไปรวมกับเส้นเชื่อมที่อัลกอริทึมเลือกไว้ จะทำให้เกิด cycle

สมมติ cycle ดังกล่าวคือ cycle C



เนื่องจากทุกเส้นเชื่อมใน cycle C ถูกเลือกก่อนที่อัลกอริทึมจะพิจารณาเส้นเชื่อม (u,v) แสดงว่า (u,v) เป็นเส้นเชื่อมที่มีน้ำหนักมากที่สุดใน C

ดังนั้น cycle property, (u,v) ไม่อยู่ใน MST

จากทั้ง 2 กรณี เราสามารถสรุปได้ว่า

เส้นเชื่อมใดๆ ที่อยู่ใน MST จะถูกเลือกโดยอัลกอริทึม และ
เส้นเชื่อมใดๆ ที่ไม่อยู่ใน MST จะไม่ถูกเลือกโดยอัลกอริทึม

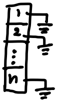
$\therefore$ คำตอบของอัลกอริทึมประกอบด้วยเส้นเชื่อมทั้งหมดใน MST $\blacksquare$

Implementations and Running times of Kruskal's algorithm

$n-1 \leq m \leq n^2$

รับ connected undirected graph $G = (V, E)$, น้ำหนักเส้นเชื่อม $w(e)$ ของทุกเส้นเชื่อม $e \in E$

สร้างกราฟว่าง $(V, E' = \emptyset)$ สำหรับเก็บคำตอบ — $O(n)$ ← สร้าง adj. list ว่างสำหรับเก็บคำตอบ



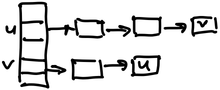
เรียงเส้นเชื่อมใน E ตามลำดับน้ำหนักจากน้อยไปมาก — $O(m \log \underbrace{m}_{\leq n^2}) = O(m \log n)$

m รอบ

for each edge (u, v) ตามลำดับที่เรียงไว้

  if ไม่มี path จาก u ไป v ในกราฟคำตอบ (V, E') — (X)

    $E' \leftarrow E' \cup \{(u, v)\}$ — $O(1)$



  end if

end for

return (V, E')

รวมเวลาทำงาน $\left[O(\underbrace{n}_{\leq m+1}) + O(m \log n)\right] + m\left[X + O(1)\right]$

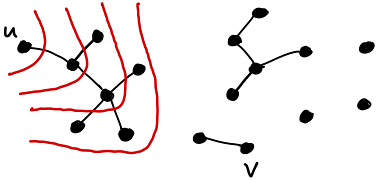
$= O(m \log n) + mX$

การตรวจสอบว่า มี path จาก u ไป v ในกราฟคำตอบหรือไม่

(ตรวจสอบว่า u และ v อยู่ใน connected component เดียวกันหรือไม่)

ใช้ได้โดย BFS/DFS

- ถ้า u และ v อยู่คนละ connected component (CC), เติมเส้นเชื่อม (u,v) ได้ เพราะไม่ทำให้เกิด cycle
- ถ้า u และ v อยู่ใน CC เดียวกัน, เติมเส้นเชื่อม (u,v) ไม่ได้ เพราะจะทำให้เกิด cycle



เวลาในการตรวจสอบโดย BFS/DFS

$O(n'+m') = O(n')$ (CC ของ u เป็น tree $\therefore m' = n'-1$)

n': #จุดยอดใน CC ของ u

m': # เส้นเชื่อมใน CC ของ u

$= O(n)$ (worst case: CC ของ u อาจใหญ่มาก เท่ากับ n)

$\therefore$ เวลาการทำงานรวมของ Kruskal's alg. ที่ใช้ BFS/DFS เป็น

$$O(m\log n) + m \times O(n) = O(mn)$$ ← ช้ากว่า Prim's alg.

Union-Find data structure

- โครงสร้างข้อมูลที่ใช้จัดการกับการ union กันของ disjoint sets
- สามารถค้นหาได้ว่า สมาชิกแต่ละตัวใน universe อยู่ในเซตใด $\mathcal{U} = \{1, 2, \ldots, n\}$
- มี 3 operations:

  - MAKE($\mathcal{U}$) : สร้างเซตตามจำนวนสมาชิกใน universe $\mathcal{U}$ (1 เซต ต่อ 1 สมาชิกใน $\mathcal{U}$)

    $\{1\} \quad \{2\} \quad \{3\} \quad \ldots \quad \{n\}$

  - FIND(x) : หาว่า x อยู่ในเซตใด (ชื่อเซตที่เก็บ x)

    สมาชิกใน $\mathcal{U}$

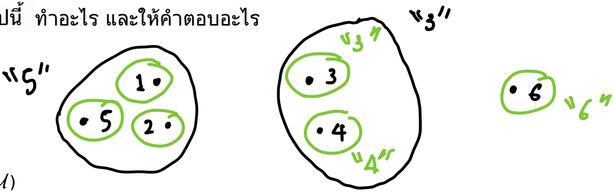
    มักจะเลือกชื่อสมาชิกมาสักตัวภายในเซตมาตั้งเป็นชื่อเซต

  - UNION(A, B) : นำเซต A และ B มา union กัน

    ชื่อเซต

Ex.2 ให้ universe $\mathcal{U} = \{1, 2, 3, 4, 5, 6\}$

คำสั่งต่อไปนี้ ทำอะไร และให้คำตอบอะไร



1. MAKE($\mathcal{U}$)

"1" "2" "3" "4" "5"

2. FIND(1), FIND(2), FIND(3), FIND(4), FIND(5)

"1" "2"

3. UNION(FIND(1), FIND(2))

"3" "4"

4. UNION(FIND(3), FIND(4))

"5" "1"

5. UNION(FIND(5), FIND(2))

"5" "5" "3" "3" "5"

6. FIND(1), FIND(2), FIND(3), FIND(4), FIND(5)

Union-Find using Array

- ใช้อาร์เรย์เก็บชื่อเซตของสมาชิกแต่ละตัวใน universe $\mathcal{U}$

$\mathcal{U} = \{1, 2, 3, \ldots, n\}$

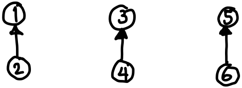
	1	2	3		n
uf	1	2	3	. . .	n

- เวลาในการทำงาน:
  - MAKE($\mathcal{U}$) : กำหนดค่าในช่องที่ i เป็นค่า i $\forall i = 1, 2, \ldots, n$ $O(n)$
  - FIND(x) : ตอบค่าในช่องที่ x ของอาร์เรย์ $O(1)$
  - UNION(A, B) : เปลี่ยนค่าในอาร์เรย์ช่องที่เป็น B ให้เป็น A แทน $O(n)$

Union-Find using Trees

- เก็บแต่ละเซตเป็น rooted tree
- ใช้ root ของ tree เป็นชื่อของเซต

เช่น $\{1,2\}$ $\{3,4\}$ $\{5,6\}$



เซต "1" เซต "3" เซต "5"

Note ในทางปฏิบัติเราสามารถเก็บ rooted tree เหล่านี้ลงในอาร์เรย์ได้ โดยในช่องที่ i ใดๆ ของอาร์เรย์ เก็บชื่อ parent ของ i ใน tree

จากตัวอย่างข้างบนเราสามารถเก็บ rooted tree ทั้ง 3 ต้นในอาร์เรย์ได้ดังนี้

	1	2	3	4	5	6
p	1	1	3	3	5	5

$\mathcal{U} = \{1,2,3,4,5,6\}$

- เวลาในการทำงาน:
  - MAKE($\mathcal{U}$): สร้าง rooted tree n ต้น แต่ละต้นมี 1 จุดยอด $O(n)$

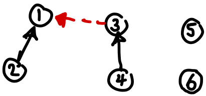
1 2 3 4 5 6 $\longleftrightarrow$ P

	1	2	3	4	5	6
P	1	2	3	4	5	6

  - UNION(A, B): เอา root ของ B ชี้ไปที่ A (เปลี่ยนค่า P[B] เป็น A) $O(1)$

เช่น union("1","3")

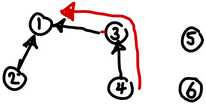
ชื่อเซต



	1	2	3	4	5	6
P	1	1	~~3~~ 1	3	5	5

- FIND(x): วิ่งจาก x ไปหา root ของ tree ต้นที่ x อยู่ $O(n)$
(เพื่อเอาชื่อ root มาตอบ)

เช่น find(4)



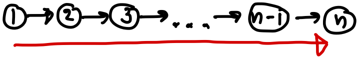
p

1	2	3	4	5	6
1	1	1	3	5	5

เวลาทำงานขึ้นอยู่กับความลึกของ x ใน tree

worst case x อาจอยู่ที่ระดับความลึก n

เช่น union(2,1)
union(3,2)
⋮
union(n,n-1)



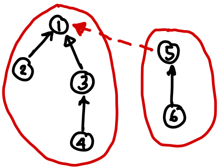
find(1)

- เปรียบเทียบขนาดของเซตก่อนการ union
- นำ root ของ tree ของเซตที่เล็กกว่าชี้ไปหา root ของ tree ของเซตที่ใหญ่กว่า

สามารถทำได้ในเวลา O(1) โดยจะต้องสร้างอาร์เรย์ size มาเก็บขนาดของแต่ละเซต

- ทุกครั้งที่สั่ง union ไปอ่านขนาดของทั้งสองเซตจากอาร์เรย์ size
- เมื่อ union เสร็จ update ค่าในอาร์เรย์ size ของเซตใหม่ โดยแก้ค่าในอาร์เรย์ช่องที่ตรงกับ root ของเซตใหม่ เป็นขนาดของเซตใหม่

เช่น



union("1", "5")

	1	2	3	4	5	6
size	~~4~~ 4+2				2	

	1	2	3	4	5	6
p	1	1	1	3	~~5~~ 1	5

ข้อสังเกต 1 สำหรับ x ใดๆ ใน universe

ความลึกของ x ใน tree = จำนวนครั้งที่เซตของ x ถูกนำไป union
โดยที่เซตของ x เป็นเซตที่เล็กกว่า

ข้อสังเกต 2 สำหรับ x ใดๆ ใน universe

เมื่อ union เซตของ x กับเซตที่ใหญ่กว่า จะทำให้ขนาดของเซตของ x ใหญ่ขึ้น
เป็นอย่างน้อย 2 เท่าของเซตเดิม

สมมติว่า เซตของ x ถูกนำไป union ในฐานะเซตที่เล็กกว่า ทั้งหมด k ครั้ง (นั่นคือ ความลึกของ x ใน tree เป็น k) เราจะได้ว่า

$2^k \leq$ ขนาดของเซตของ $x \leq n$ นั่นคือ

$$
\begin{aligned}
2^k &\leq n \\
\log_2 2^k &\leq \log_2 n \\
k \cancel{\log_2 2}^{\,1} &\leq \log_2 n \\
k &\leq \log_2 n
\end{aligned}
$$

$\therefore$ ความลึกของ x ใน tree ไม่เกิน $\log_2 n$

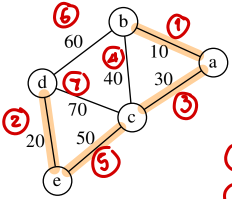
ดังนั้น เวลาในการ find(x) = $O(\log n)$ เมื่อใช้การ union by size

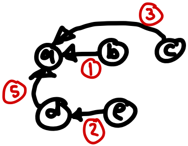
Kruskal's algorithm with Union-Find data structure

การตัดสินใจ เติม/ไม่เติม เส้นเชื่อม (u, v) ลงในกราฟคำตอบ

- เติม (u, v) ถ้า u และ v อยู่คนละ connected component
- ไม่เติม (u, v) ถ้า u และ v อยู่ใน connected component เดียวกัน

เราสามารถนำ union-find มาช่วยในการตรวจสอบ connected component ของจุดยอดในกราฟคำตอบได้





1 FIND(a) ≠ FIND(b)

2 FIND(d) ≠ FIND(e)

3 FIND(a) ≠ FIND(c)

4 FIND(b) = FIND(c)

5 FIND(c) ≠ FIND(e)

6 FIND(b) = FIND(d)

7 FIND(c) = FIND(d)

$n-1 \leq m \leq n^2$

รับ connected undirected graph $G = (V, E)$, น้ำหนักเส้นเชื่อม $w(e)$ ของทุกเส้นเชื่อม $e \in E$ — $O(\cancel{n}^{\leq m+1} + m) = O(m)$

สร้างกราฟว่าง $(V, E' = \emptyset)$ สำหรับเก็บคำตอบ — $O(\cancel{n}^{\leq m+1}) = O(m)$

เรียงเส้นเชื่อมใน E ตามลำดับน้ำหนักจากน้อยไปมาก — $O(m \log \cancel{m}_{\leq n^2}) = O(m \log n)$

UF.MAKE(V) — $O(n)$

m รอบ

for each edge (u, v) ตามลำดับที่เรียงไว้

  $A \leftarrow$ UF.FIND(u) — $O(\log n)$

  $B \leftarrow$ UF.FIND(v) — $O(\log n)$

  if $A \neq B$ — $O(1)$

    $E' \leftarrow E' \cup \{(u, v)\}$ — $O(1)$

    UF.UNION(A, B) — $O(1)$

  end if

end for

รวม $O(m \log n)$

return (V, E')

$$\text{รวมเวลา } \underbrace{[O(m) + O(m) + O(m \log n)]}_{\text{ก่อน for}} + \underbrace{[O(m \log n)]}_{\text{for}}$$

$$= O(m \log n)$$ ← เท่ากับ prim's algorithm