Algorithm Design and Analysis

Week8: Dijkstra's algorithm, Minimum Spanning Trees (Intro.)

Adisak Supeesun

3 February 2022

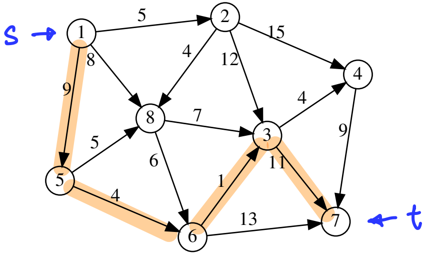
Review

ปัญหา shortest path (STP)

ให้ directed graph $G = (V, E)$ โดยเส้นเชื่อม $e \in E$ ใดๆ มีความยาว $\ell(e) \geq 0$,

จุดยอดต้นทาง $s \in V$ และ จุดยอดปลายทาง $t \in V$

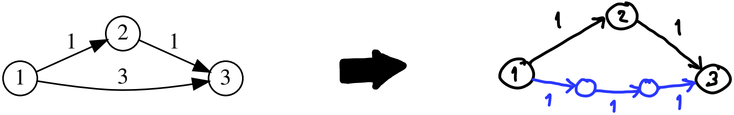
ต้องการหา path ที่สั้นที่สุดจาก s ไปยัง t



Review (Cont.)

$O(\#\text{nodes} + \#\text{edges})$

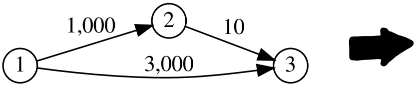
- ถ้าความยาวเส้นเชื่อมทั้งหมดเป็นจำนวนเต็ม เราสามารถประยุกต์ใช้ bfs ได้

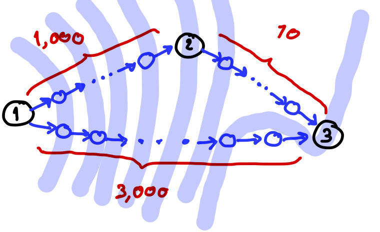


- ถ้ามีเส้นเชื่อมที่มีความยาวมากๆ การใช้ bfs อาจจะช้ากว่า bruteforce

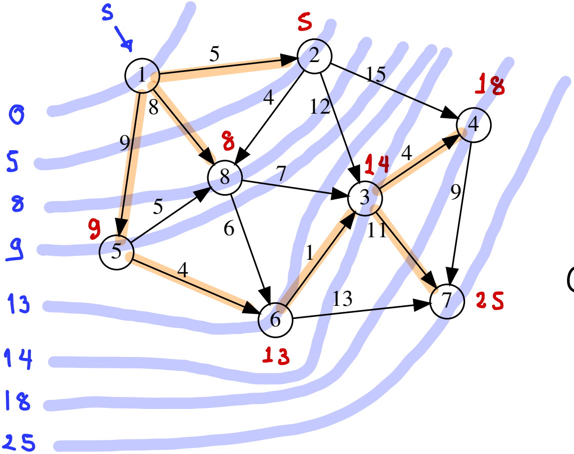
1000

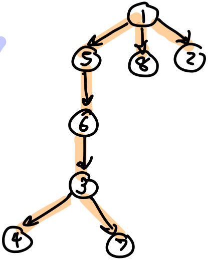
1010





Designing Algorithm

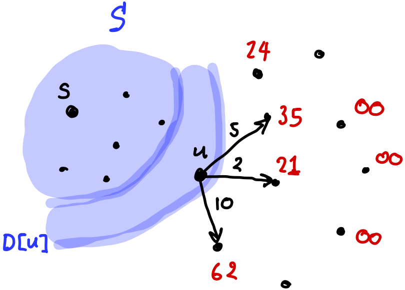




" stp tree "

Dijkstra's Algorithm

```
รับ input: กราฟ $G = (V, E)$ ,ความยาวเส้นเชื่อม $\ell(e)$ ของทุกเส้นเชื่อม $e \in E$, จุดยอดต้นทาง $s \in V$
$S \leftarrow \{s\}$
$D[v] \leftarrow \infty, \ \forall v \in V$  และ  $D[s] \leftarrow 0$
$p[v] \leftarrow v \quad \forall v \in V$
while $V \neq S$
    เลือกจุดยอด $u \notin S$  ที่มีค่า $D[u]$ น้อยที่สุด
    $S \leftarrow S \cup \{u\}$
    for each เส้นเชื่อม $(u, v) \in E$ ที่ชี้ออกจาก $u$
        if $D[v] > D[u] + \ell((u, v))$
            $D[v] \leftarrow D[u] + \ell((u, v))$
            $p[v] \leftarrow u$
        end if
    end for
end while
```

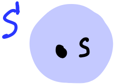


Theorem 1

สำหรับจุดยอด u ∈ *S ใดๆ,* $D[u]$ *เป็นความยาวของ shortest path จากจุดยอด s ไปยัง u*

proof (by induction บนขนาดของเซต S')

Base case เมื่อ $|S'| = 1$



เซต S' มีแค่จุดยอด s เพียงจุดยอดเดียว

และจากอัลกอริทึม กำหนดค่าเริ่มต้น $D[s] = 0$

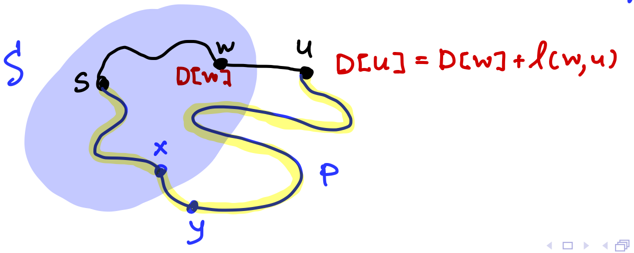
ซึ่งเป็นค่าความยาวของ stp จาก s ไป s จริง ✓

Inductive step

I.H. { สมมติว่าในตอนที่เซต S' มีขนาด k-1, ทุกจุดยอด v ∈ S' มีค่า D[V] เท่ากับความยาวของ stp จากจุดยอด s ไปยัง

(จะแสดงว่า เมื่อ S' มีขนาด k, ทุกจุดยอด u ∈ S' ต้องมีค่า D[V] เท่ากับความยาวของ stp จาก s ไป v)

สมมติว่าจุดยอด u เป็นจุดยอดลำดับที่ k ที่ถูกใส่เข้าไปใน S'



$D[u] = D[w] + \ell(w,u)$

พิจารณา path P ใดๆ จาก s ไป u

ให้ x เป็นจุดยอดแรกใน P ที่ไม่อยู่ในเซต S'

และ y เป็นจุดยอดใน P ที่อยู่ก่อน x

แบ่ง P เป็น 3 ช่วง

- จาก s ไปถึง x
- เส้นเชื่อม (x, y)
- จาก y ไปถึง u

จะได้ว่า $\ell(P) = \ell(P_{sx}) + \ell((x,y)) + \underbrace{\ell(P_{yu})}_{\geqslant 0}$

$$\geqslant \ell(P_{sx}) + \ell((x,y))$$

$$\geqslant \underbrace{\text{ความยาว stp จาก s ไป x}}_{D[x] \text{ (จาก I.H.)}} + \ell((x,y))$$

$$= D[x] + \ell((x,y))$$

$$\geq D[y]$$

เนื่องจาก u คือจุดยอดที่ถูกเลือกไปใส่ใน S ไม่ใช่ y
(อัลกอริทึมจะเลือกจุดยอดที่ค่า D น้อยสุด)

$$\geq D[u]$$

$\therefore$ D[u] มีค่าไม่เกินความยาวของทุก path จาก s ไป u

และเนื่องจาก D[u] เป็นความยาวของ path หนึ่งจาก s ไป u

(ถ้าดูตามภาพด้านบนคือ stp จาก s ไป w ตามด้วยเส้นเชื่อม (w,u))

ดังนั้น เราจึงสามารถสรุปได้ว่า

D[u] เป็นความยาวของ stp จาก s ไป u ■

Implementations and Running Times

```
รับ input: กราฟ $G = (V, E)$ ,ความยาวเส้นเชื่อม $\ell(e)$ ของทุกเส้นเชื่อม $e \in E$, จุดยอดต้นทาง $s \in V$ — O(n+m)
$S \leftarrow \{s\}$ — O(n) (สร้างอาร์เรย์ S)
$D[v] \leftarrow \infty, \ \forall v \in V$  และ  $D[s] \leftarrow 0$ — O(n)
$P[v] \leftarrow v, \ \forall v \in V$ — O(n)
while $V \neq S$ — O(1) (ใช้ตัวแปร count นับรอบของ while)
    เลือกจุดยอด $u \notin S$ ที่มีค่า $D[u]$ น้อยที่สุด — O(n) (หาค่าน้อยสุดในอาร์เรย์ D)
    $S \leftarrow S \cup \{u\}$ — O(1) ( S[u] ← 1 )
    for each เส้นเชื่อม $(u, v) \in E$ ที่ชี้ออกจาก $u$
        if $D[v] > D[u] + \ell((u, v))$ — O(1)
            $D[v] \leftarrow D[u] + \ell((u, v))$ — O(1)
            $p[v] \leftarrow u$ — O(1)
        end if
    end for
end while
```

n รอบ

outdegree(u) รอบ

s

S [0|0|0|1|0|0|0]

0: อยู่ใน S , 1: อยู่นอก S

ทุก node $u \in V$ ส่วนนี้จะถูกทำครั้งเดียว

∴ รวมเวลาส่วนนี้ของทุก node จะได้

$$\sum_{u \in V} \text{outdegree}(u) \times O(1)$$

$$= O(1) \times \sum_{u \in V} \text{outdegree}(u) = O(m)$$

Running time เมื่อหาจุดยอด u จากการเปรียบเทียบค่าในอาร์เรย์ D โดยตรง

$$\left[\text{เวลาการทำงานนอกลูป while}\right] + \left[\text{เวลาการทำงานลูป while}\right]$$

$$= \left[\underbrace{O(n+m)+O(n)}_{O(n+m)}\right] + \underbrace{\left[\underbrace{(\text{เวลานอก for})}_{n\times O(n)} + \underbrace{(\text{เวลาทำ for})}_{O(m)}\right]}_{\text{รวมเป็น } O(n^2+m)}$$

$$= O(n^2+m)$$

Note • ถ้ากราฟมีเส้นเชื่อมเยอะ $m \approx n^2$ จะได้ว่า $O(n^2+\overset{\approx n^2}{m}) = \overset{\text{ดีเมื่อ m เยอะ}}{O(n^2)}$

• แต่ถ้ากราฟมีเส้นเชื่อมน้อย เช่น $m \approx n$ จะได้ว่า $O(n^2+\underset{n}{m}) = \underset{\text{เยอะไปเมื่อ m น้อย}}{O(n^2)}$

Priority Queue

- คิวที่สมาชิกตัวแรกมี prioriry สูงที่สุด แต่ละข้อมูลที่เก็บใน priority queue จะเก็บเป็น

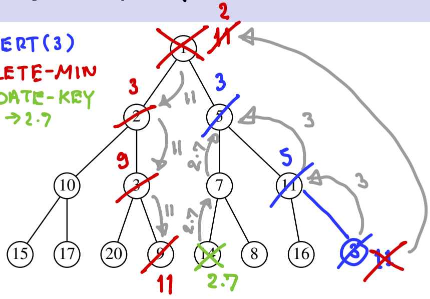
$$(\underbrace{\text{value}}_{\text{ข้อมูล}}, \underbrace{\text{key}}_{\text{priority}})$$

key ต่ำ : priority สูง
key สูง : priority ต่ำ

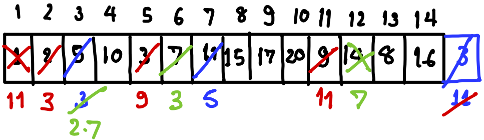
- Operations:
  - INSERT(*value*, *key*) : เพิ่มข้อมูลลงคิว
  - DELETE-MIN() : นำข้อมูลออกจากคิว (ข้อมูลที่ key ต่ำสุดออกมาก่อน)
  - UPDATE-KEY(*value*, *new_key*) : เปลี่ยนค่า key ของข้อมูล

Binary Heap (พิจารณา min heap)

- INSERT(3)
- DELETE-MIN
- UPDATE-KEY 14 → 2.7



เราสามารถเก็บ binary heap โดยใช้ array ได้



ถ้าเริ่ม node u ใดๆ เราสามารถคำนวณหาตำแหน่งของ แม่ และ ลูกทั้ง 2 ของ u ในอาร์เรย์ได้ใน O(1)

- almost complete binary tree
- พ่อแม่มี key ไม่เกิน key ของลูก
- เวลาในการทำงาน
  - INSERT : O(log n)
  - DELETE-MIN : O(log n)
  - UPDATE-KEY : O(log n)

เมื่อ n คือจำนวนข้อมูลใน heap

Dijkstra’s algorithm using priority queue

```
รับ input: กราฟ $G = (V, E)$ ,ความยาวเส้นเชื่อม $\ell(e)$ ของทุกเส้นเชื่อม $e \in E$, จุดยอดต้นทาง $s \in V$   — $O(n+m)$
$D[v] \leftarrow \infty$,  $\forall v \in V$  และ  $D[s] \leftarrow 0$   — $O(n)$
$PQ$.INSERT($v$, $D[v]$),  $\forall v \in V$   — $O(n \log n)$
$P[v] \leftarrow v$,  $\forall v \in V$   — $O(n)$
while $PQ$ is not empty   — $O(1)$
    $u \leftarrow PQ$.DELETE-MIN()   — $O(\log n)$
    for each เส้นเชื่อม $(u, v) \in E$ ที่ชี้ออกจาก $u$
        if $D[v] > D[u] + \ell((u, v))$   — $O(1)$
            $D[v] \leftarrow D[u] + \ell((u, v))$   — $O(1)$
            $PQ$.UPDATE-KEY($v$, $D[v]$)   — $O(\log n)$
            $p[v] \leftarrow u$   — $O(1)$
        end if
    end for
end while
```

n รอบ

outdegree(u) รอบ

ทุก node $u \in V$ ส่วนนี้จะถูกทำครั้งเดียว

∴ รวมเวลาส่วนนี้ของทุก node จะได้

$$\sum_{u \in V} \text{outdegree}(u) \times O(\log n)$$

$$= O(\log n) \times \sum_{u \in V} \text{outdegree}(u)$$

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

Running time เมื่อหาจุดยอด *u* จาก prioriry queue ที่ implement ด้วย binary heap

[เวลาการทำงานนอกลูป while] + [เวลาการทำงานในลูป while]

$$= \left[\underbrace{O(n+m) + O(n\log n)}_{O(m+n\log n)}\right] + \left[\underbrace{\underbrace{(\text{เวลานอก for})}_{n\times O(\log n)} + \underbrace{(\text{เวลาใน for})}_{O(m\log n)}}_{\text{รวมเป็น } O(n\log n + m\log n)}\right]$$

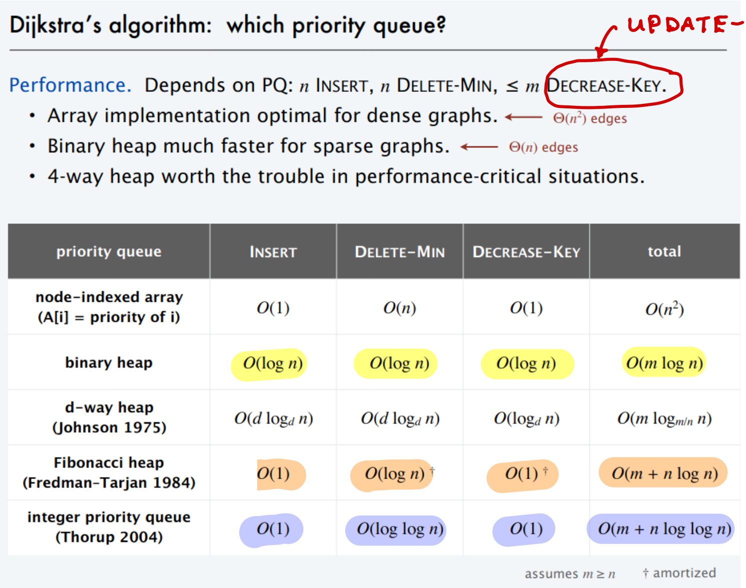
$$= O((n+m)\log n)$$

Note ถ้า input เป็น connected graph จะได้ว่า $n-1 \le m \le n^2$

$$O(\overset{\le 2m}{(n+m)}\log n) = O(m\log n)$$

และถ้าเส้นเชื่อมน้อย เช่น $m \approx n$, $O(m\log n) \approx \boxed{O(n\log n)}$

ดีกว่าวิธีที่ไม่ใช้ priority queue ที่เป็น $O(n^2)$



Dijkstra's algorithm: which priority queue?

UPDATE-KEY

Performance. Depends on PQ: n INSERT, n DELETE-MIN, $\leq m$ DECREASE-KEY.

- Array implementation optimal for dense graphs. ← $\Theta(n^2)$ edges
- Binary heap much faster for sparse graphs. ← $\Theta(n)$ edges
- 4-way heap worth the trouble in performance-critical situations.

priority queue	INSERT	DELETE-MIN	DECREASE-KEY	total
node-indexed array (A[i] = priority of i)	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
d-way heap (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
Fibonacci heap (Fredman-Tarjan 1984)	$O(1)$	$O(\log n)^\dagger$	$O(1)^\dagger$	$O(m + n \log n)$
integer priority queue (Thorup 2004)	$O(1)$	$O(\log \log n)$	$O(1)$	$O(m + n \log \log n)$

assumes $m \geq n$ † amortized [1]

---

[1] source: https://www.cs.princeton.edu/ wayne/kleinberg-tardos/