จอมทัพมองโกลผู้บ้าคลั่ง

- มีจอมทัพมองโกลผู้บ้าคลั่งรายหนึ่ง ต้องการเป็นใหญ่โดยการเคลื่อนทัพไปตีเมืองอื่น หากแต่ในสมัยก่อนนั้น การจะไปตีเมืองใดนั้น กองทัพต้องตีเมืองระหว่างทางไปด้วย เพื่อสะสมกำลังคนและสะสมเสบียงอาหาร ดังนั้นเพื่อไม่ให้กองทัพเหนื่อยเกินไป ท่านจอมทัพ จึงต้องการหาเส้นทางที่สั้นที่สุดในการเคลื่อนทัพไปตีเมืองที่ต้องการ
- กำหนดแผนผังของประเทศต่างๆ และเมืองที่ท่านจอมทัพต้องการไปตี จงออกแบบโปรแกรม เพื่อช่วยบอกว่าท่านจอมทัพต้องตีเมืองใดบ้าง เพื่อให้ไปถึงเมืองที่ท่านต้องการ โดยตีเมืองระหว่าง ทางให้น้อยที่สุด

Shortest Path Algorithm

- Single Source Shortest Path (SSSP)
  - บนกราฟที่เส้นเชื่อมไม่มีน้ำหนัก (Unweighted graph)
  - บนกราฟที่เส้นเชื่อมมีน้ำหนักเป็นบวกหรือศูนย์ (Non-negative edge)
  - บนกราฟที่เส้นเชื่อมติดลบได้ แต่ไม่มีลูปที่ติดลบ
  - บนกราฟที่มีลูปที่ติดลบ
- Single Pair Shortest Path
  - ประยุกต์จาก SSSP แล้วตัดเอาเฉพาะคู่ที่ต้องการ
  - A* Heuristic Search
- All-Pair Shortest Path
  - Floyd Warshall's Dynamic Programming Algorithm

SSSP บนกราฟที่เส้นเชื่อมไม่มีน้ำหนัก

- หากเส้นเชื่อมไม่มีน้ำหนัก เราจะถือว่าจำนวนเส้นเชื่อมที่ต้องเดินทางผ่านคือระยะทาง
  → ดังนั้นเส้นทางที่สั้นที่สุดก็คือเส้นทางที่ผ่านเส้นเชื่อมน้อยที่สุดนั่นเอง
- เราใช้ Breadth-First Search มาแก้ปัญหานี้ได้ทันที
- เพราะ BFS จะสำรวจโหนดที่ใกล้ที่สุดจนหมดก่อนที่จะย้ายไปสำรวจโหนดที่ไกลออกไป
- เมื่อสิ้นสุด BFS จากโหนด Source เราจะรู้ระยะทางที่สั้นที่สุดจาก Source ดังกล่าวไปยังโหนดทุกโหนดที่สามารถไปถึงได้จาก Source

SSSP บนกราฟที่ไม่มีเส้นเชื่อมค่าน้ำหนักติดลบ

เราใช้ BFS ไม่ได้ แต่เราจะใช้อัลกอริทึมที่ชื่อว่า Dijkstra's Algorithm

แนวคิด

- เนื่องจากค่าน้ำหนักในเส้นเชื่อมไม่มีค่าติดลบ แสดงว่ายิ่งเดินทางผ่านโหนดมากเท่าไหร่ ค่าน้ำหนักก็จะยิ่งเพิ่มมากขึ้นเท่านั้น
- ดังนั้นจากโหนด Source ถ้ามองออกไปรอบ ๆ ที่เพื่อนบ้าน หากเลือกเส้นเชื่อมที่มีค่าน้อยที่สุด เราเลือกเส้นเชื่อมนั้นได้เลยนั่นจะเป็นทางที่สั้นที่สุดจาก Source แน่ ๆ
  - ➢ อย่างที่บอกไว้ว่าเส้นเชื่อมไม่มีค่าติดลบ ดังนั้นใครที่เป็นผู้ชนะแล้วจะไม่มีทางถูกแย่งชิงตำแหน่งไปได้แน่นอน
- ผนวกโหนดที่ไปถึงแล้วเข้ามาเป็น Super Node พร้อมกับเส้นเชื่อมของมัน จากนั้นทำไปเรื่อย ๆ จนไม่พบโหนดใหม่ในกราฟอีก

Dijkstra's Algorithm

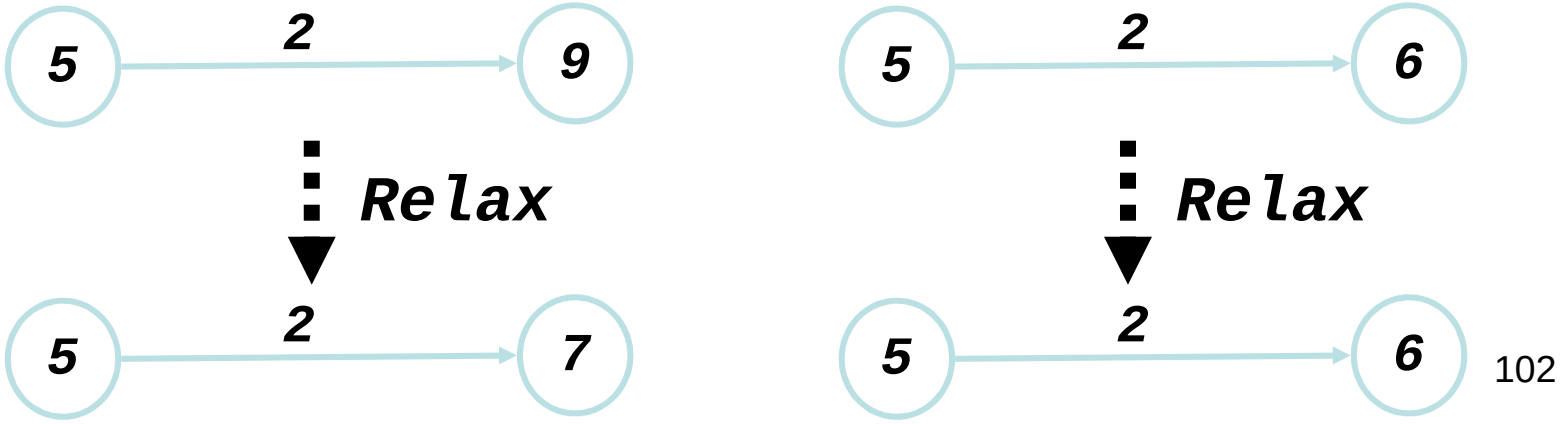
```
Dijkstra(G)
    for each v ∈ V
        d[v] = ∞;
    d[s] = 0; S = ∅; Q = V;
    while (Q ≠ ∅)
        u = ExtractMin(Q);
        S = S U {u};
        for each v ∈ u->Adj[]
            if (d[v] > d[u]+w(u,v))
                d[v] = d[u]+w(u,v);
```

Relaxation Step

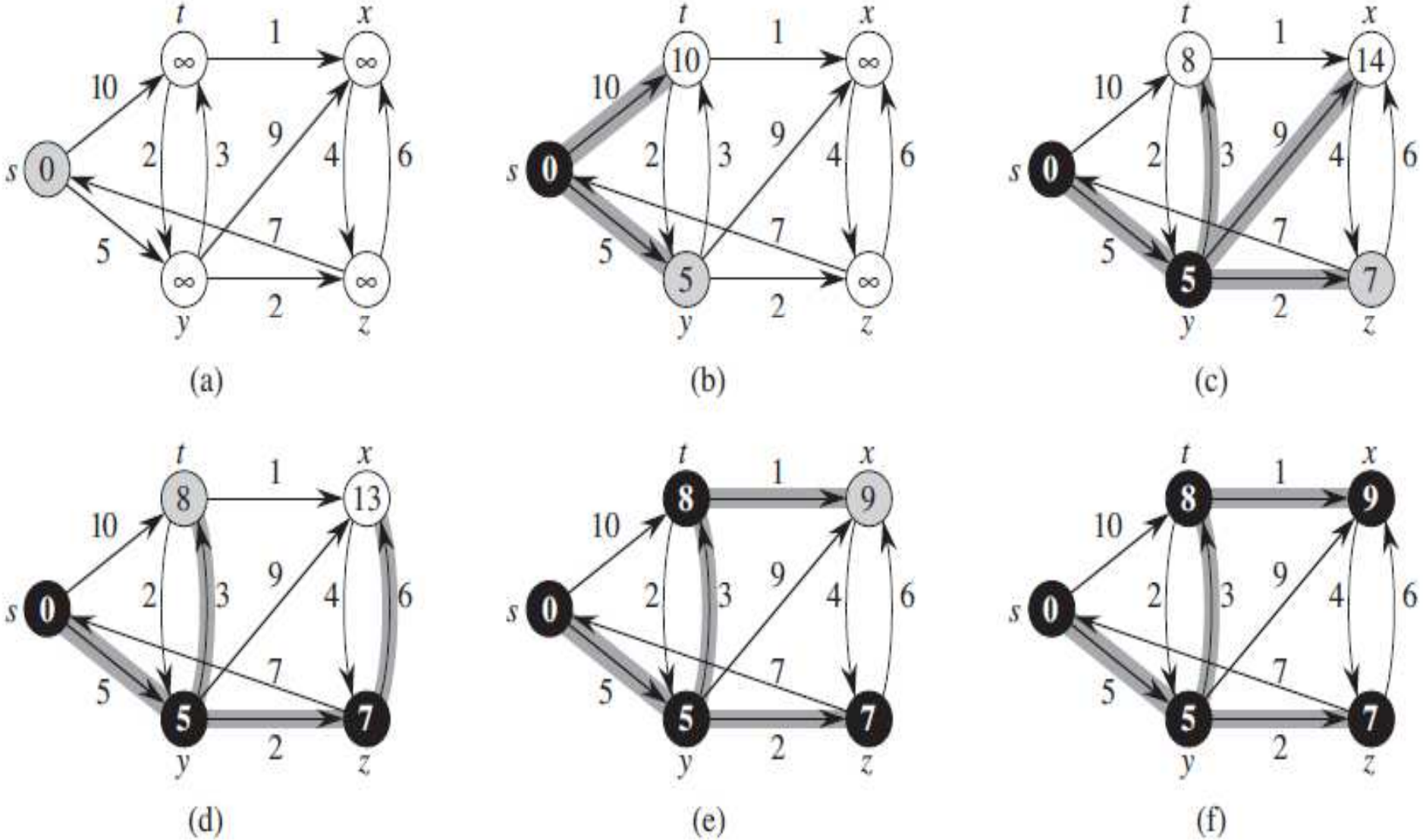
Relaxation

- A key technique in shortest path algorithms is *relaxation*
  - Idea: for all *v*, maintain upper bound d[*v*] on δ(s,v)

```
Relax(u,v,w) {
    if (d[v] > d[u]+w) then d[v]=d[u]+w;
}
```



Dijkstra's: Example



Bellman-Ford Algorithm

- เป็นอัลกอริทึมที่สามารถใช้กับกราฟที่มีค่าน้ำหนักติดลบได้
  - ➢แต่ถ้ากราฟไม่มีค่าน้ำหนักติดลบ ใช้ Dijkstra's จะเร็วกว่า
- ใช้หลักการ relaxation คือในตอนแรกค่าที่ได้จะเป็นค่าประมาณ แต่ค่าจะได้รับการปรับเรื่อย ๆ จนในที่สุดจะได้ค่าที่ถูกต้องและดีที่สุดสำหรับการคำนวณเส้นทางที่สั้นที่สุด
  - การ relaxation จะทำโดยพิจารณาการเปลี่ยนแปลงค่าน้ำหนักรวมที่เกิดจากเส้นเชื่อมทุกเส้น → ใช้เวลา $O(|E|)$
  - แต่การทำ relaxation จะต้องทำหลายรอบ รวมทั้งหมดเป็นตามจำนวนโหนด นั่นก็คือ $|V|$ → รวมการใช้เวลาเป็น $O(|V|\,|E|)$

Bellman-Ford Algorithm

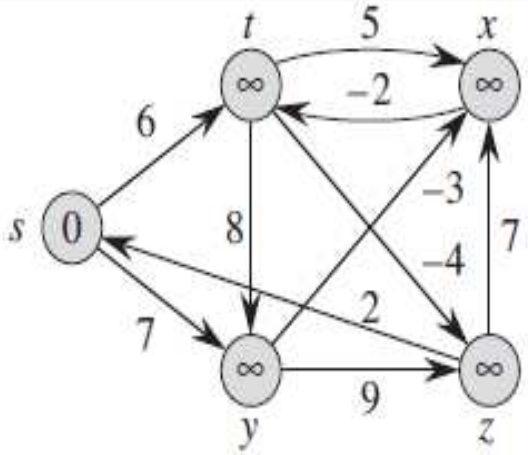
```
BellmanFord()
   for each v ∈ V
      d[v] = ∞;
   d[s] = 0;
   for i=1 to |V|-1
      for each edge (u,v) ∈ E
         Relax(u,v, w(u,v));
   for each edge (u,v) ∈ E
      if (d[v] > d[u] + w(u,v))
           return “no solution”;
```

Initialize d[], which will converge to shortest-path value δ

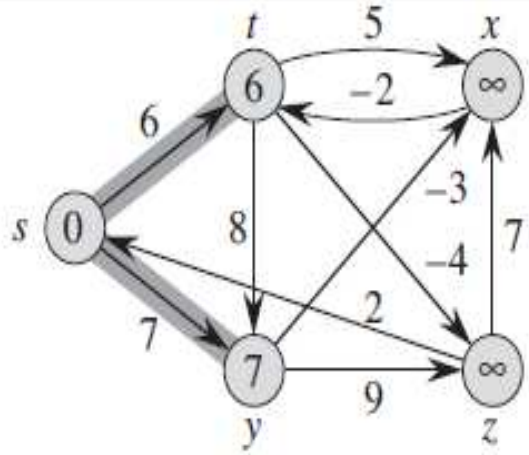
Relaxation: Make |V|-1 passes, relaxing each edge

Test for solution Under what condition do we get a solution?

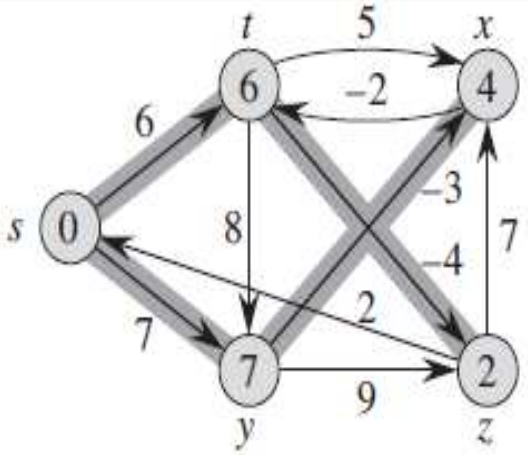
```
Relax(u,v,w): if (d[v] > d[u]+w) then d[v]=d[u]+w
```


t
5
x
∞
−2
∞
6
−3
s
0
8
7
−4
2
∞
∞
9
y
z

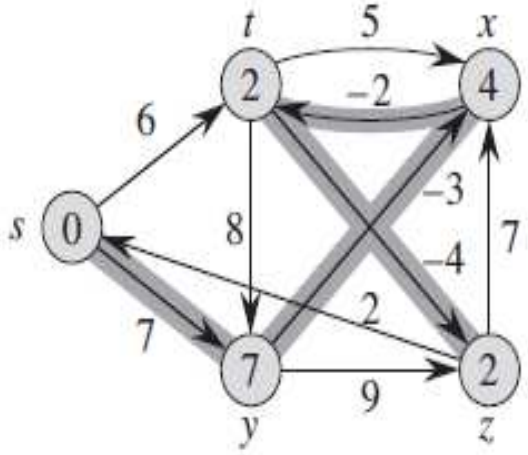
(a)


t
5
x
6
−2
∞
6
−3
s
0
8
7
−4
2
7
∞
9
y
z

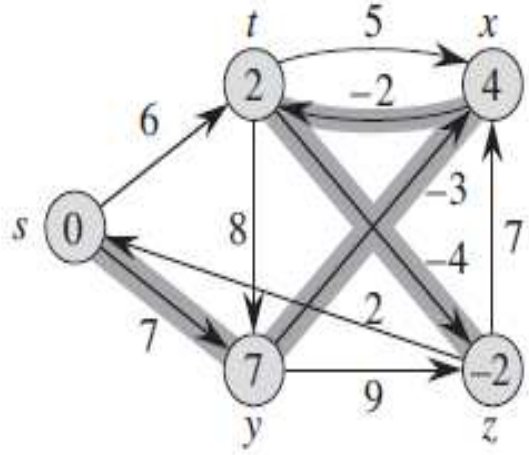
(b)


t
5
x
6
−2
4
6
−3
s
0
8
7
−4
2
7
2
9
y
z

(c)


t
5
x
2
−2
4
6
−3
s
0
8
7
−4
2
7
2
9
y
z

(d)


t
5
x
2
−2
4
6
−3
s
0
8
7
−4
2
7
−2
9
y
z

(e)

Problem

- **341 - Non-Stop Travel**
- **558 - Wormholes**