Shortest Path

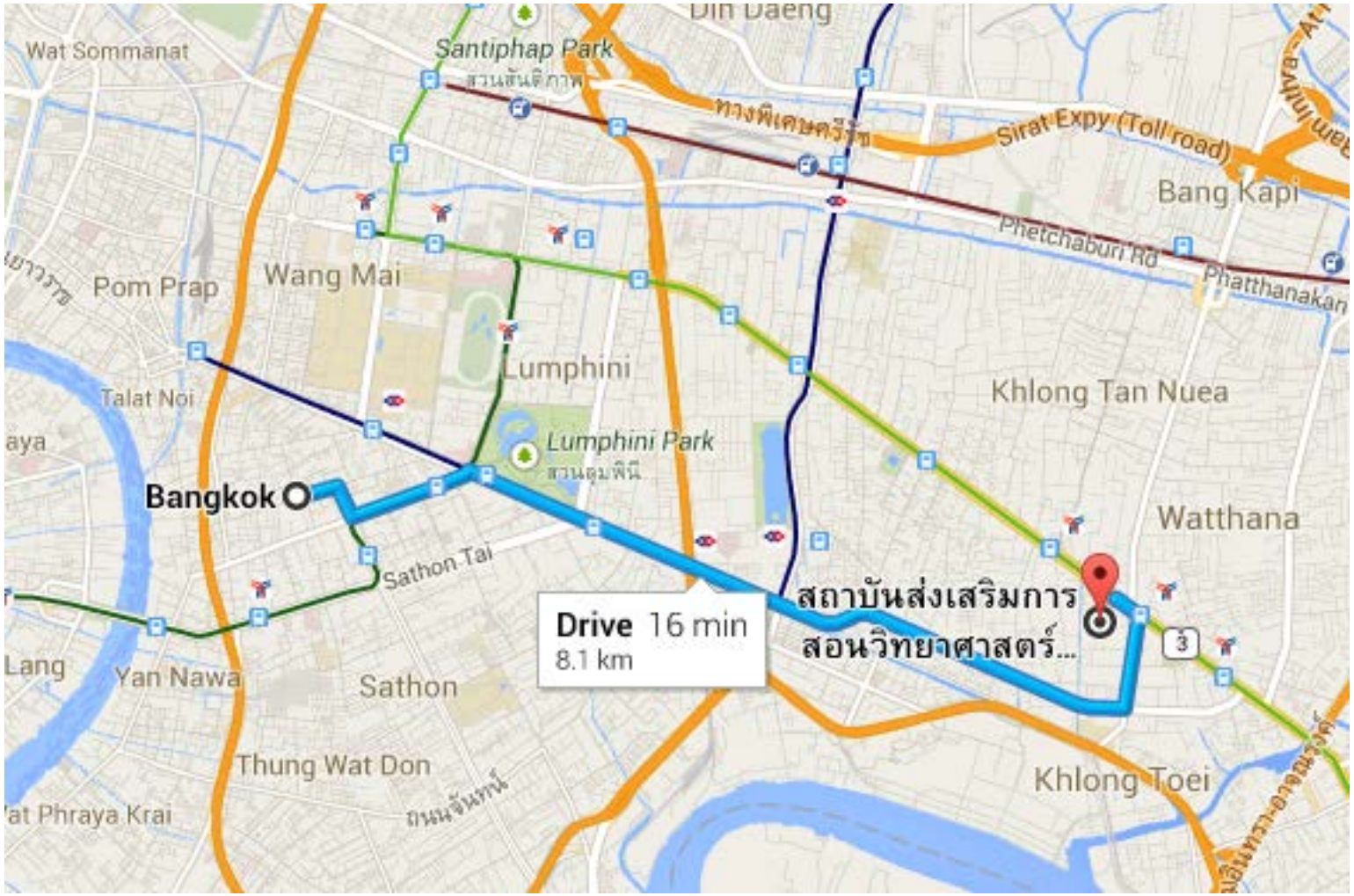
อักฤทธิ์ สังข์เพ็ชร

Slides ปรับเปลี่ยนจาก

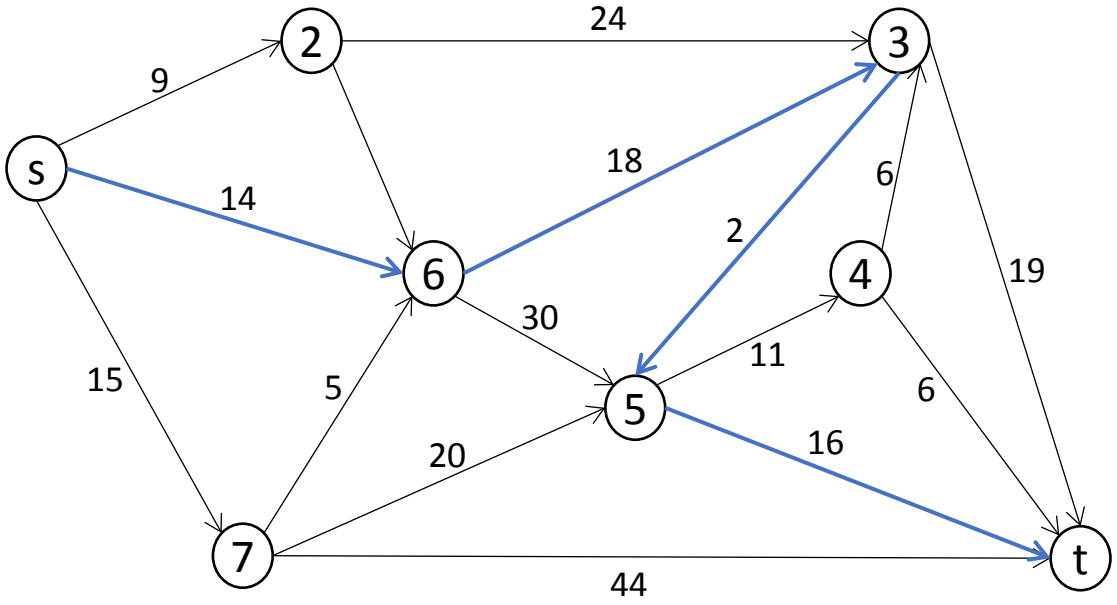
www.cs.princeton.edu/~rs/AlgsDS07/15**ShortestPaths**.pdf

courses.csail.mit.edu/6.006/spring11/lectures/lec15.pdf

Shortest Path



Shortest Path



จาก Weighted directed graph *G = (V,E)*
หาเส้นทาง (path) ที่สั้นที่สุดจาก s ไปยัง t

ระยะทาง = ผลรวมของ weight จากแต่ละ edge ใน path

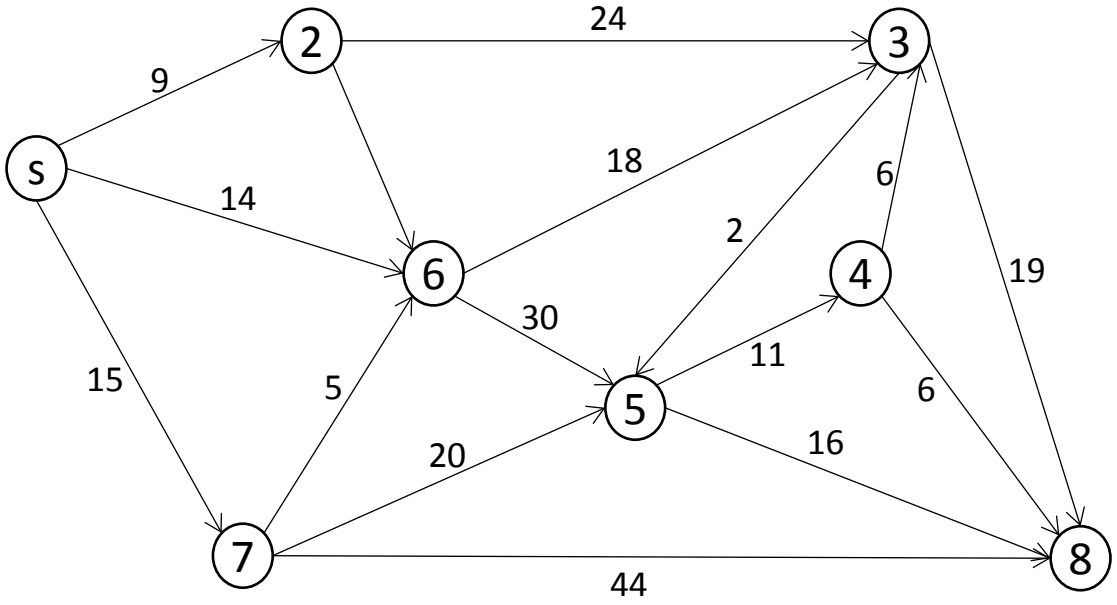
Path: s -> 6 -> 3 -> 5 -> t
Cost: 14+18+2+16 = 50

Weight เป็นตัวเลขอะไรก็ได้
- ไม่จำกัดว่าต้องเป็นระยะทาง

versions

- Source-target
- Single source
- All pairs
- Nonnegative edge weights
- Arbitrary weights
- Euclidean weights

Single-source Shortest Paths

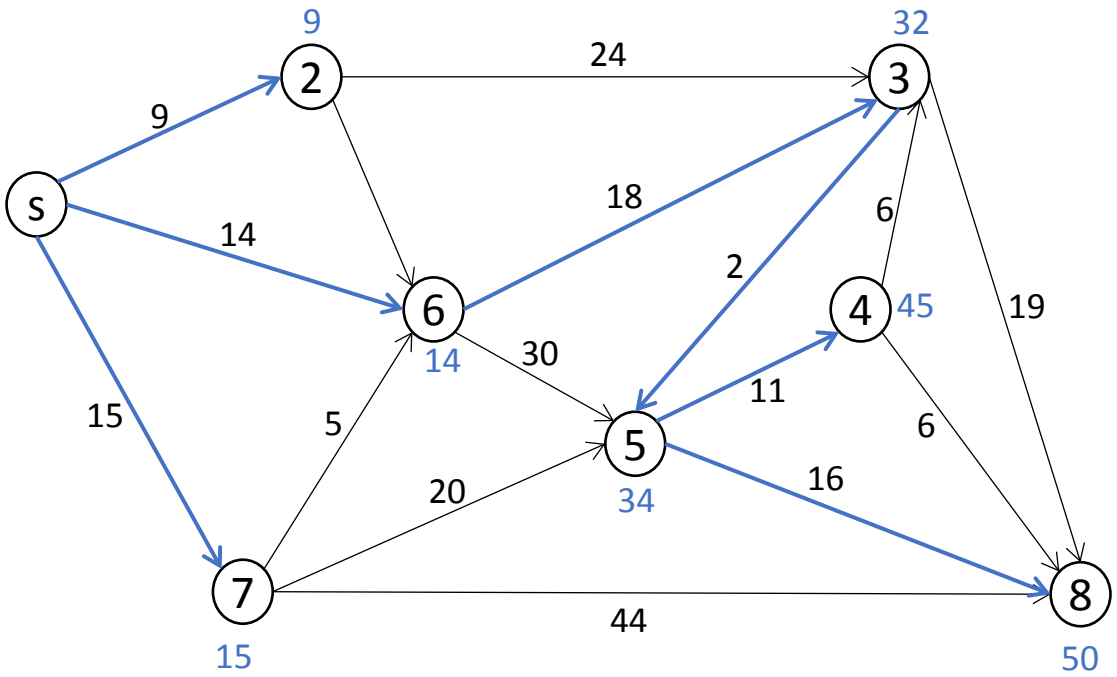


จาก Weighted directed graph *G*
และจุดเริ่มต้น source *s*

ปัญหา: หาระยะทาง (distance) และ เส้นทางที่สั้นที่สุด (shortest path) จาก s ไปยัง vertex ที่เหลือทั้งหมด

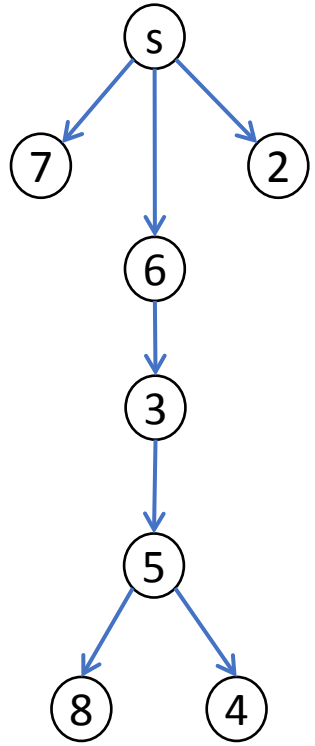
dist[v] = ความยาวของเส้นทางที่สั้นที่สุด (shortest path) จาก s ไป v
pred[v] = vertex สุดท้ายบนเส้นทางที่สั้นที่สุด ก่อนที่จะถึง v

Single-source Shortest Paths



เส้นทาง (path) ที่สั้นที่สุดทั้งหมดสามารถรวมกันเป็น tree ได้

v	dist[v]	pred[v]
S	0	s
2	9	s
3	32	6
4	45	5
5	34	3
6	14	s
7	15	s
8	50	5

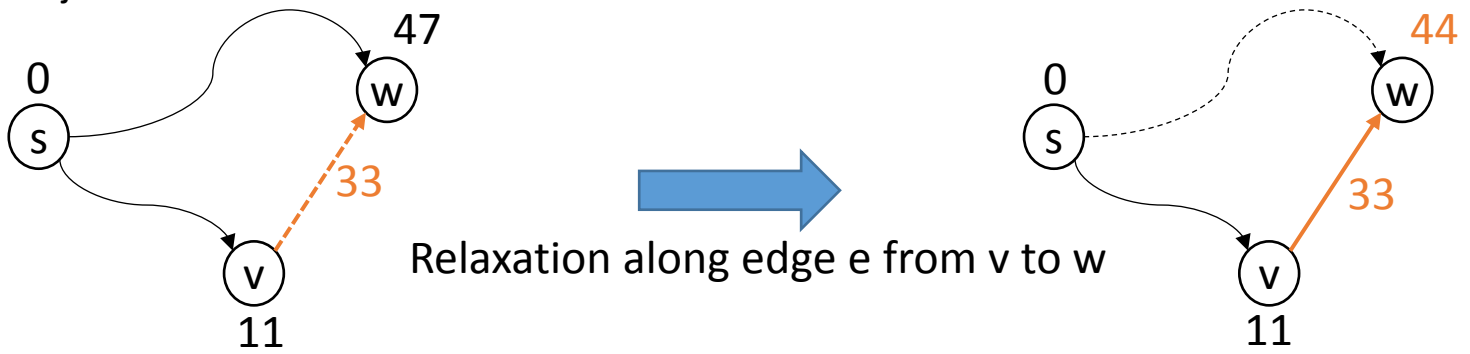


การลดระยะทาง (Edge relaxation)

- สำหรับ vertex v, ให้ dist[v] = ระยะทางของ path อะไรก็ได้จาก s ไป v
- Relaxation บน edge e จาก v ไป w
  - dist[v] = ระยะทางของ path ใด ๆ จาก s ไป v
  - dist[w] = ระยะทางของ path ใด ๆ จาก s ไป w
  - ถ้า v-w ทำให้ path จาก s ไป w สั้นลง (โดยเดินจาก s ผ่าน v แล้วไป w) ให้ update dist[w] และpred[w]

relax(v,w):

```
If (dist[w] > dist[v] + e.weight) {
    dist[w] = dist[v] + e.weight;
    pred[w] = v;
}
```

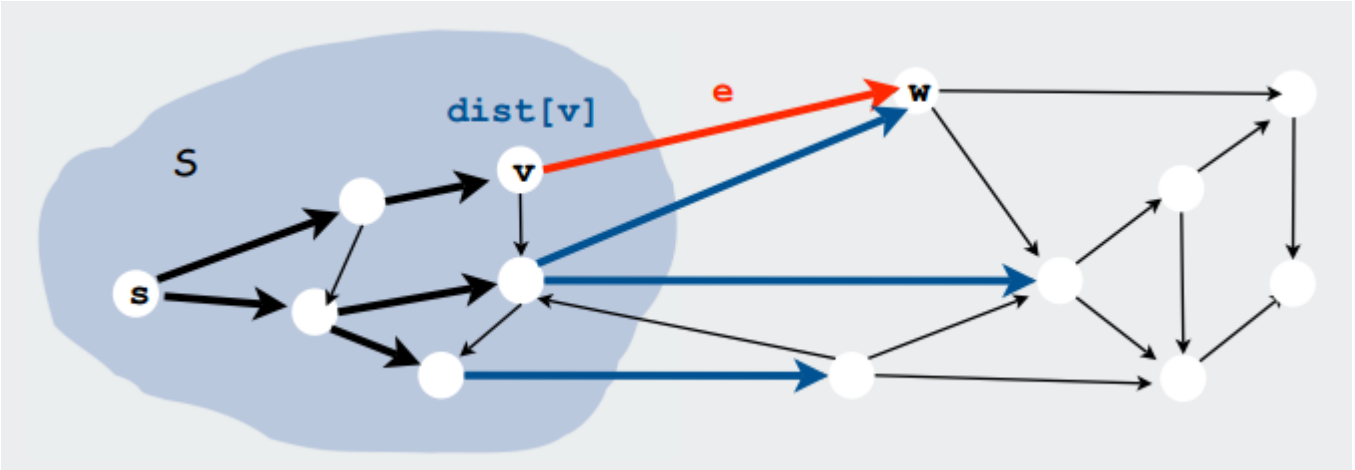


Dijkstra’s algorithm

- *S*: set ของ vertices ที่สามารถหา shortest path จาก s ได้แล้ว
- S’: set ของ vertices ที่ไม่อยู่ใน S
- Invariant: สำหรับ v ในเซต *S*, dist[v] = ระยะทางสั้นที่สุดจาก s ไป v

ให้ S = { s }, dist[s] = 0 และให้ dist[v] = ∞ สำหรับ vertex v ที่เหลือ
วนจนกว่าเซต S จะประกอบด้วย vertex ทั้งหมดที่ connected กับ s

- หา edge e ที่มี v อยู่ใน S และ w ใน S’ ที่ dist[v] + e.weight มีค่าน้อยที่สุด
- *relax(v,w)*
- เพิ่ม w เข้าไปใน S

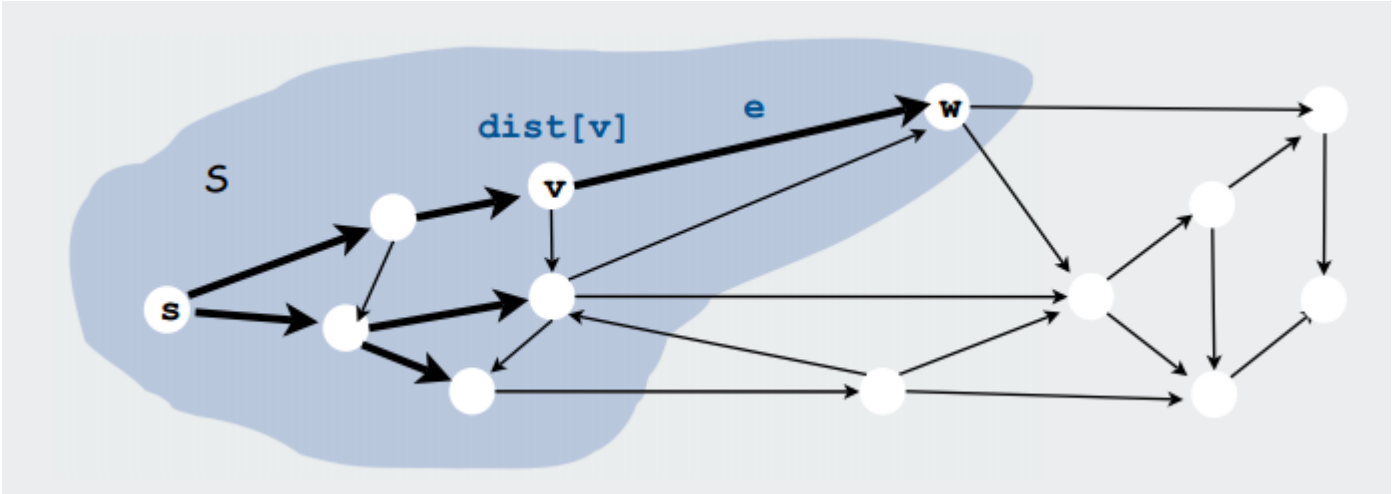


Dijkstra's algorithm

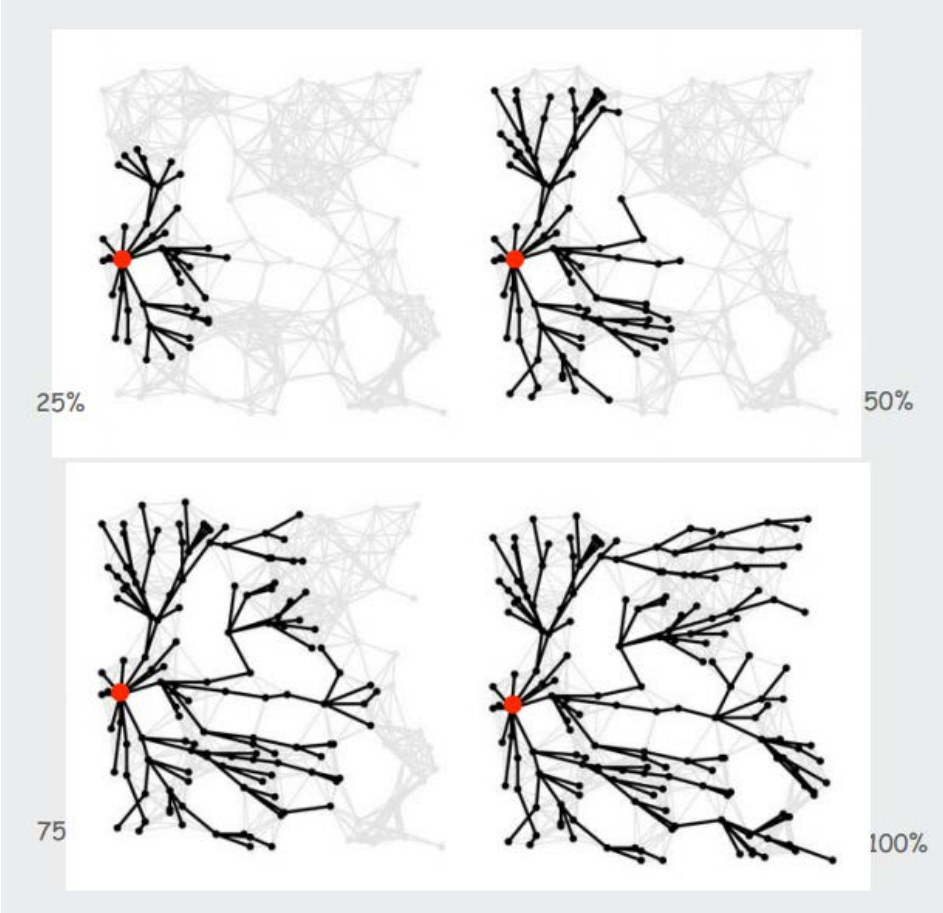
- *S*: set ของ vertex ที่สามารถหา shortest path จาก s ได้แล้ว
- S': set ของ vertex ที่ไม่อยู่ใน S; *S' = V - S*
- Invariant: สำหรับ v ในเซต *S*, dist[v] = ระยะทางสั้นที่สุดจาก s ไป v

ให้ S = { s }, dist[s] = 0 และให้ dist[v] = ∞ สำหรับ vertex v ที่เหลือ
วนจนกว่าเซต S จะประกอบด้วย vertex ทั้งหมดที่ connected กับ s

- หา edge e ที่มี v อยู่ใน S และ w ใน S' ที่ dist[v] + e.weight มีค่าน้อยที่สุด
- *relax(v,w)*
- เพิ่ม w เข้าไปใน S



Shortest Path Tree



Dijkstra’s algorithm: implementation

ให้ S = { s }, dist[s] = 0 และให้ dist[v] = ∞ สำหรับ vertex v ที่เหลือ
วนจนกว่าเซต S จะประกอบด้วย vertex ทั้งหมดที่ connected กับ s

- หา edge e ที่มี v อยู่ใน S และ w ใน S’ ที่ dist[v] + e.weight มีค่าน้อยที่สุด
- relax along edge e
- เพิ่ม w เข้าไปใน S

วิธีที่ 1: ลองทุก edge

….O(VE)

Dijkstra’s algorithm: implementation

ให้ S = { s }, dist[s] = 0 และให้ dist[v] = ∞ สำหรับ vertex v ที่เหลือ
วนจนกว่าเซต S จะประกอบด้วย vertex ทั้งหมดที่ connected กับ s

- หา edge e ที่มี v อยู่ใน S และ w ใน S’ ที่ dist[v] + e.weight มีค่าน้อยที่สุด
- relax along edge e
- เพิ่ม w เข้าไปใน S

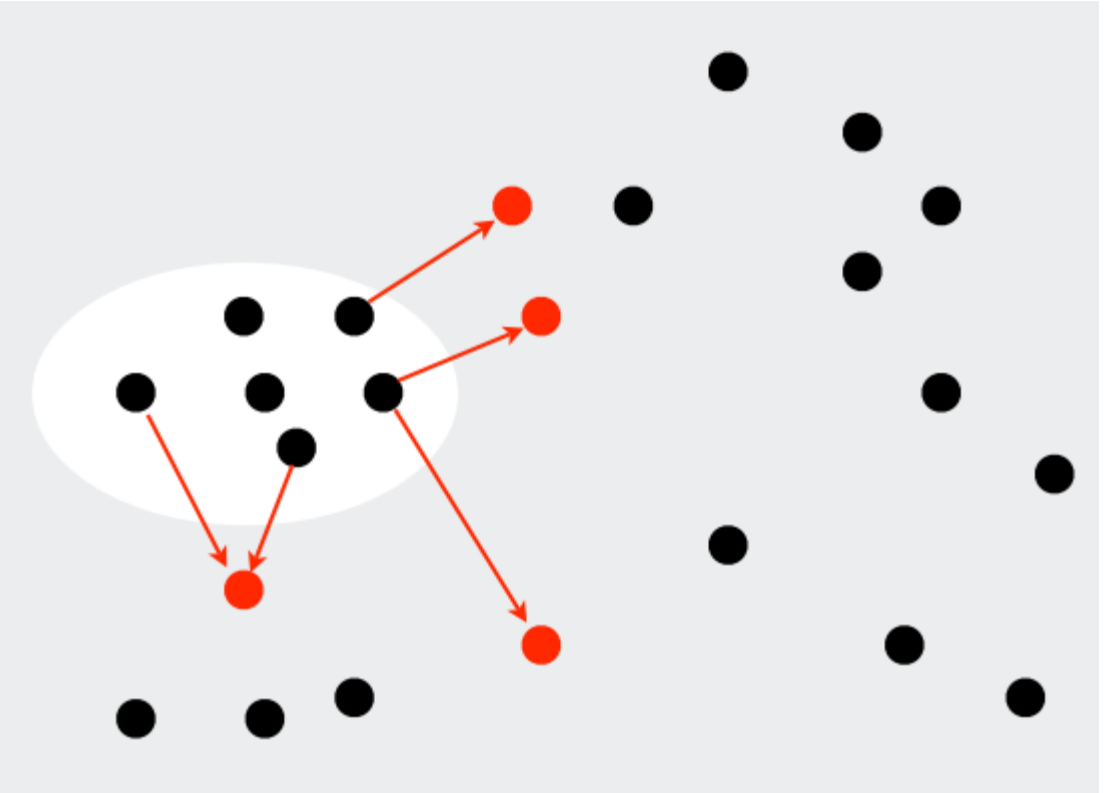
วิธีที่ 2: Dijkstra – ใช้ priority queue เพื่อหา edge ที่จะ relax

Running time ขึ้นกับการใช้ Priority queue : $|V|*T_{delmin} + |E|*T_{deckey}$

Dijkstra’s implementation

เราควรเอาอะไรใส่ใน queue?

- Fringe vertex = vertex ที่มี edge เชื่อมกับ vertex ใน S



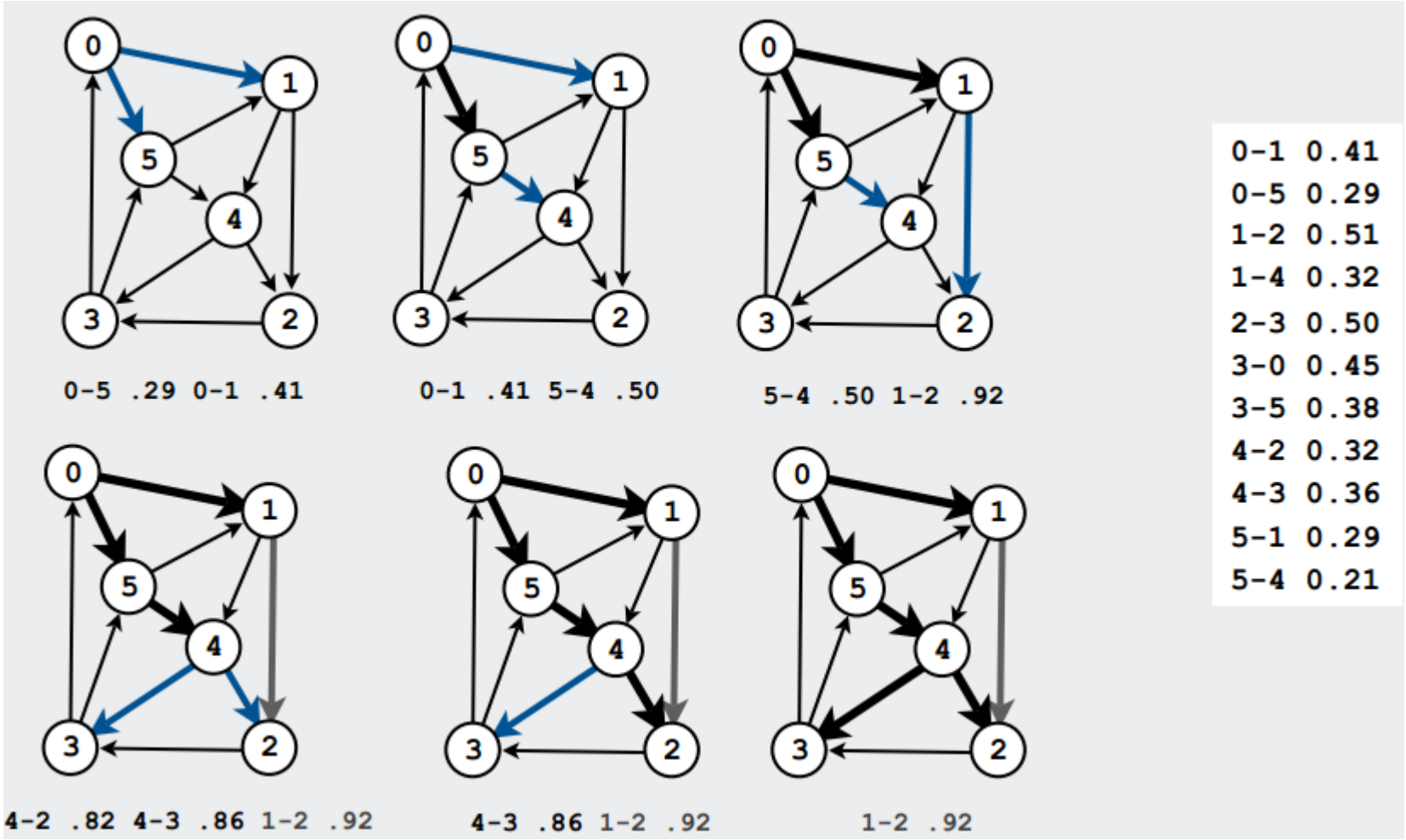
Dijkstra’s algorithm example

Dijkstra’s algorithm [Dijkstra 1957]

เริ่มจาก vertex 0

สร้าง tree T แบบ greedy

ในแต่ละขั้น เพิ่ม path ที่มีระยะทางสั้นที่สุดโดยที่ edge สุดท้ายมี vertex หนึ่งอันอยู่ใน T



Implementation

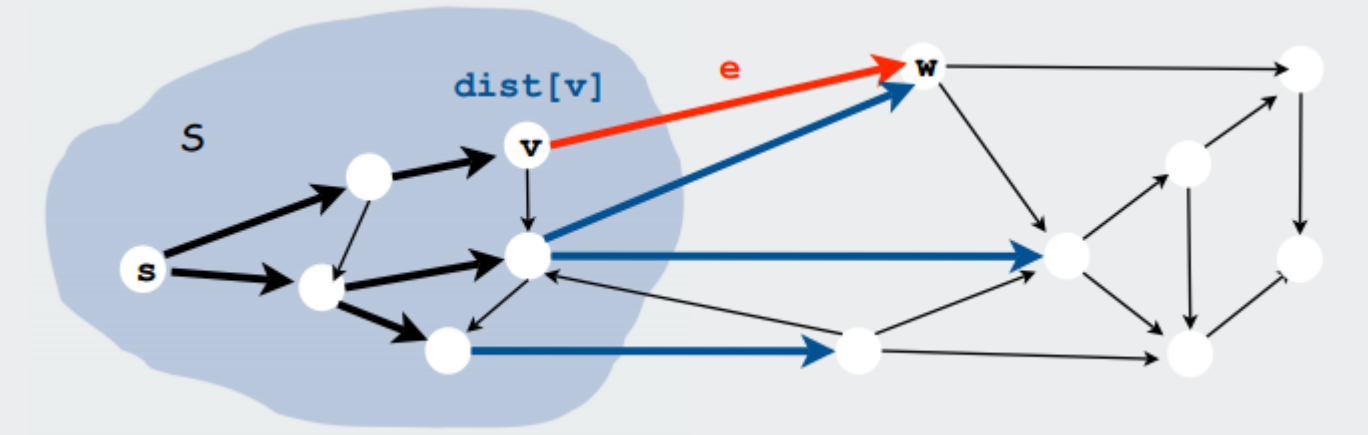
```
bool[] marked = new bool[|V|];
for (v : V) dist[v] = ∞;
MinPQ<double, int> pq = new MinPQ<>();
dist[s] = 0;
pq.put(dist[s], s);


while (!pq.empty()) {
    int v = pq.delmin();
    if (marked[v]) continue;
    marked[v] = true;
    for (Edge e : G.adj(v)) {
        int w = e.to();
        if (dist[w] > dist[v] + e.weight()) {
            dist[w] = dist[v] + e.weight();
            pred[w] = e;
            pq.insert(dist[w], w); // or pq.deckey(dist[w], w)
        }
    }
}
```

Priority first seach

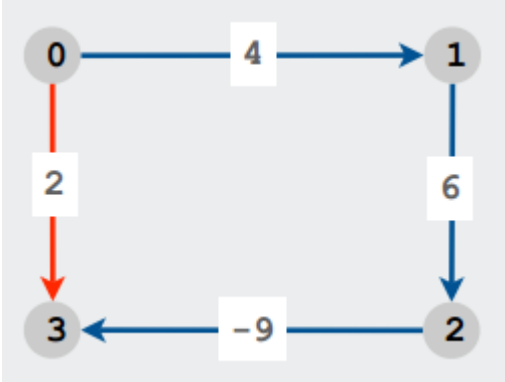
- Graph search ใช้ algorithm แบบเดียวกัน
  - มี Set *S* ของ vertex ที่ถูกสำรวจไปแล้ว
  - ขยาย Set S โดยเติม edge ที่มีปลายหนึ่งข้างอยู่นอก S

DFS	ใช้ edge จาก vertex ล่าสุดที่ถูกค้นพบ (most recent)
BFS	ใช้ edge จาก vertex ที่พบอันแรก (least recent)
Prim	ใช้ edge ที่มี weight น้อยที่สุด
Dijkstra	ใช้ edge ที่มี vertex ใกล้กับ source มากที่สุด



Negative weight (without negative cycle)

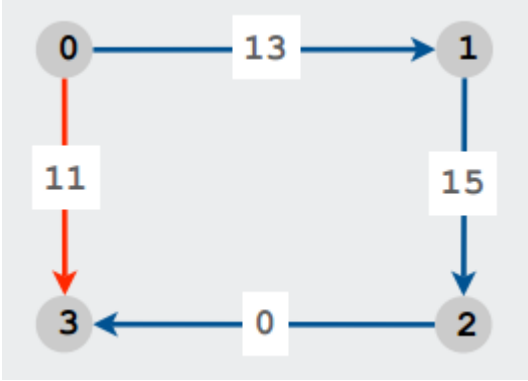
- Dijkstra ไม่สามารถใช้กับ weight ที่ติดลบได้



Dijkstra จะเลือก vertex 3 ถัดจาก 0
ทำให้ shortest path จาก 0 ไป 3 เป็น 0-3 (ระยะทาง 2)

แต่ shortest path ควรเป็น 0-1-2-3 (ระยะทาง 1)

- วิธีเติม weight

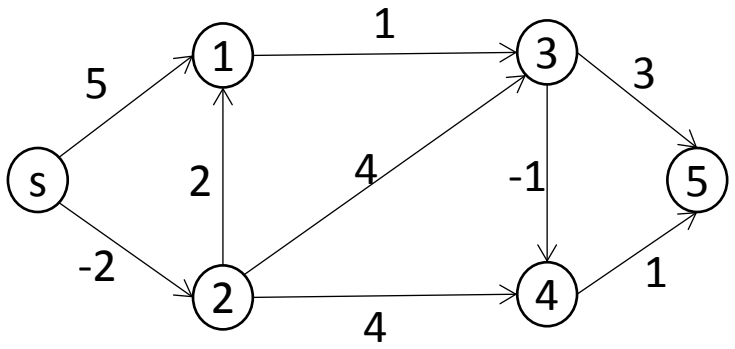


+9 ทุก path... ก็ใช้ไม่ได้เช่นกัน

Bellman-Ford

- relax ทุก edge (ตามลำดับ) เป็นจำนวน |V|-1 รอบ

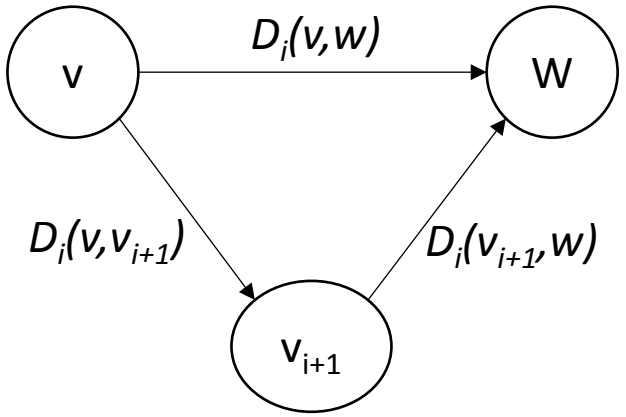
```
dist[s] = 0;
for i from 1 to |V|-1 {
    for e(v,w) in E {
        if (dist[w] > dist[v] + e.weight) {
            dist[w] = dist[v] + e.weight;
            pred[w] = e;
        }
    }
}
//check for negative cycle
for e(v,w) in E {
    if (dist[w] > dist[v] + e.weight)
        report that a negative cycle exists
}
```



Optimization:
ถ้า dist[v] ไม่เปลี่ยนในรอบที่ i,
ไม่จำเป็นต้อง relax edge ที่ออกจาก v ในรอบที่ i+1

All-pair Shortest Paths

- หา **Shortest paths** สำหรับทุกคู่ในรอบเดียว
- Dynamic programming



$V_i = \{v_0, v_1, \ldots, v_i\}$; $0 \le i < |V|$
$D_i(v,w)$ = distance ของ paths จาก v ไป w ที่ผ่าน vertex ภายใน V_i เท่านั้น

$$D_0(v,w) = \begin{cases} weight(v,w) ; (v,w) \in E \\ \infty \text{ ถ้าไม่มี edge (v,w)} \end{cases}$$

$$D_{i+1}(v,w) = min \{ D_i(v,v_{i+1}) + D_i(v_{i+1},w), D_i(v,w) \}$$

Floyd-Warshall Algorithm

```
// w[][] เป็น adjacency matrix ขนาด n x n เมื่อ n = |V|
// w[i][i] = 0
// w[i][j] = infinity ถ้าไม่มี edge (i,j)
// w[i][j] = weight ของ edge (i,j)

Floyd-Warshall:

        for( int k = 0; k < n; k++ )
            for( int i = 0; i < n; i++ )
                for( int j = 0; j < n; j++ )
                    w[i][j] = min( w[i][j], w[i][k] + w[k][j] );
```