ค่ายอบรมโอลิมปิกวิชาการ 2

อัลกอริทึม (Algorithm)

รัชดาพร คณาวงษ์

24 มีนาคม 2566

ศูนย์มหาวิทยาลัยศิลปากร

หัวข้อ

- Greedy Algorithms
- Dijkstra's Algorithm
- Minimum Spanning Tree
- Prim's Algorithm using std::heap&std::map
- Kruskal's Algorithm using std::set

ปัญหาและการหาคำตอบ

- จุดมุ่งหมายของการแก้ปัญหาเพื่อให้ได้คำตอบที่ดีที่สุดและคุ้มค่าที่สุดในขณะนั้น
- หรือ เพิ่มประสิทธิภาพการทำงาน
- หรือ จัดสรรทรัพยากรให้มีประสิทธิภาพมากที่สุด

หมายเหตุ ทรัพยากร อาจจะเป็นเวลา กำลังคน หรือ พื้นที่

ขั้นตอนวิธีประเภทละโมภ (Greedy algorithm)

- Greedy Algorithm เป็นวิธีหาคำตอบโดยเลือกทางออกที่ดีที่สุดที่พบในขณะนั้นเพื่อให้ได้คาตอบที่ดีที่สุด

แนวคิดของขั้นตอนวิธีประเภทละโมภคือ

- เลือกทางที่สามารถเลือกได้
- ตัดสินใจเลือกทางออกที่ดีที่สุด
- ดำเนินการเลือกทางลำดับถัดไปอีกเรื่อยๆ ด้วยเงื่อนไขที่ดีที่สุด ณ ลำดับนั้นๆ

ปัญหาการแลกเหรียญ (Coin Changing)

ในระบบเงินมีเงินเหรียญ 3 ประเภทคือ เหรียญ 10 บาท เหรียญ 5 บาท และเหรียญ 1 บาท

ถ้าต้องการแลกเงิน 89 บาท จะได้เหรียญ 10 บาท 8 เหรียญ, 5 บาท 1 เหรียญ และเหรียญ 1 บาท 4 เหรียญ

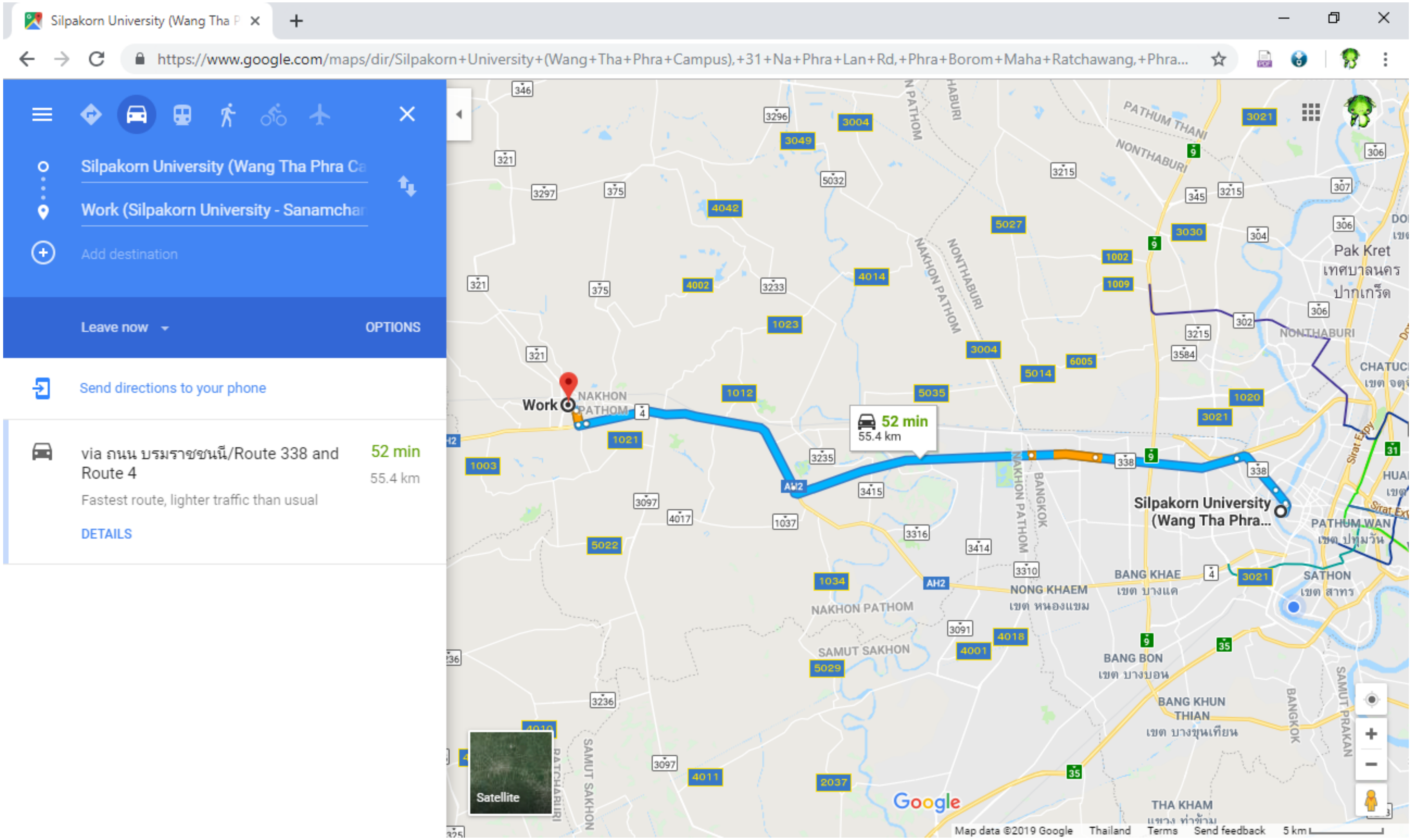
หลักการคือเราจะเลือกเหรียญที่มีค่ามากที่สุด แต่ไม่เกิน 89 บาท ออกมาก่อน จะได้เหรียญ 10 บาท 8 เหรียญ ทำให้เหลือเงิน 9 บาท เราก็เลือกเหรียญที่มากที่สุดคือเหรียญ 5 จำนวน 1 เหรียญและที่เหลืออีก 4 บาท แลกเป็นเหรียญ 1 บาททั้งหมด

ตัวอย่างวิธีคิดแบบละโมภ (Greedy)

วิธีการเหล่านี้สามารถแก้ปัญหาจากโดเมนของปัญหาที่เป็นกราฟ

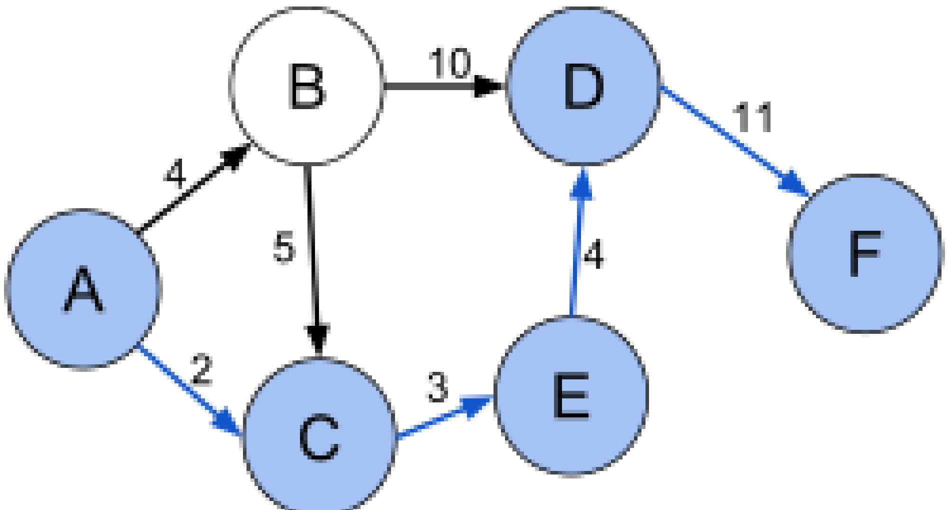
- Dijkstra's Algorithm
- Minimum Spanning Tree
- Prim's Algorithm
- Kruskal's Algorithm

Shortest Paths in a Graph



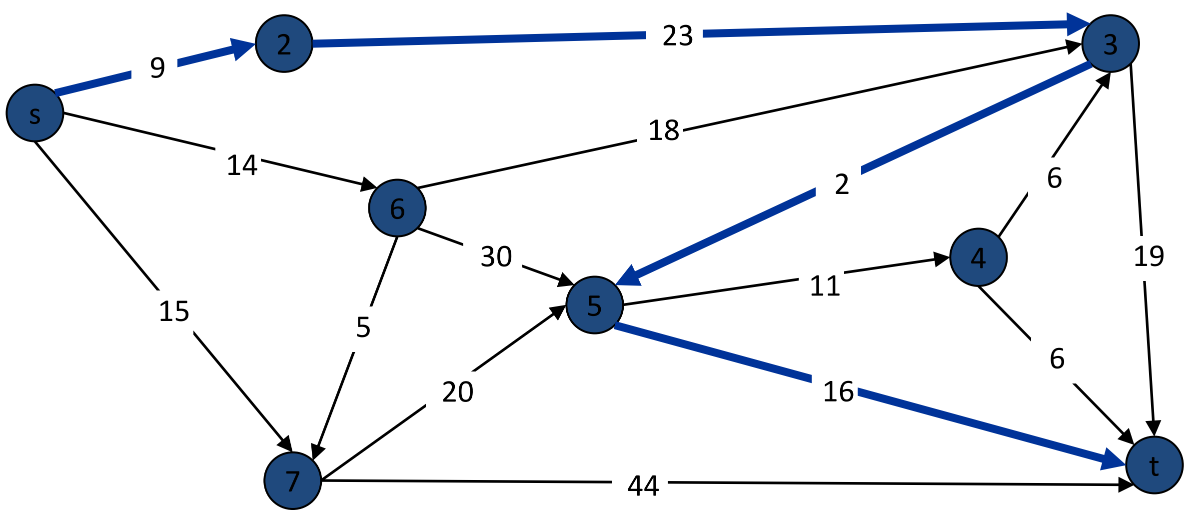
ปัญหาวิถีสั้นสุด (Shortest Path Problem)

- เป็นปัญหาที่ต้องการหาเส้นทางที่สั้นสุดระหว่างจุดยอด 2 จุดภายในกราฟ กล่าวคือ ผลรวมของน้ำหนักของเส้นเชื่อมในเส้นทางรวมกันแล้วน้อยที่สุดในบรรดาเส้นทางที่เป็นไปได้ทั้งหมด



ปัญหาวิถีสั้นสุด (Shortest Path Problem)

ปัญหาถูกแสดงด้วยกราฟแบบมีทิศทาง (Directed Graph) และกำหนดจุดเริ่มต้น s, จุดสิ้นสุด t เส้นเชื่อมมีค่าน้ำหนักกำหนด (ค่าน้ำหนักอาจเป็นข้อมูลค่าใช้จ่าย เวลา หรือระยะทางก็ได้) สิ่งที่ต้องการหาคือเส้นทางที่สั้นที่สุดจากจุด s เดินทางไปจุด t



Cost of path s-2-3-5-t

= 9 + 23 + 2 + 16

= 48.

Dijkstra’s Algorithm

1. แบ่งเป็นกรณีพื้นฐาน (base case) ได้หนึ่งหรือมากกว่าที่สามารถหาค่าได้ตรงไปตรงมา
2. ปัญหาสามารถดำเนินการด้วยกรณีในข้อหนึ่งหรือการเรียกตัวเอง

Dijkstra's Algorithm

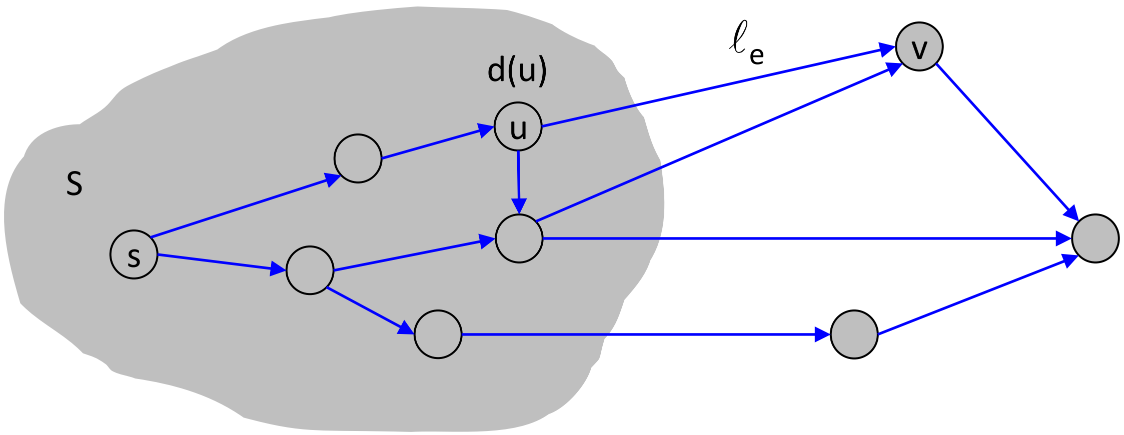
Dijkstra's algorithm.

- Maintain a set of explored nodes S for which we have determined the shortest path distance d(u) from s to u.
- Initialize S = { s }, d(s) = 0.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$$

shortest path to some u in explored part, followed by a single edge (u, v)

add v to S, and set d(v) = π(v).



Dijkstra's Algorithm

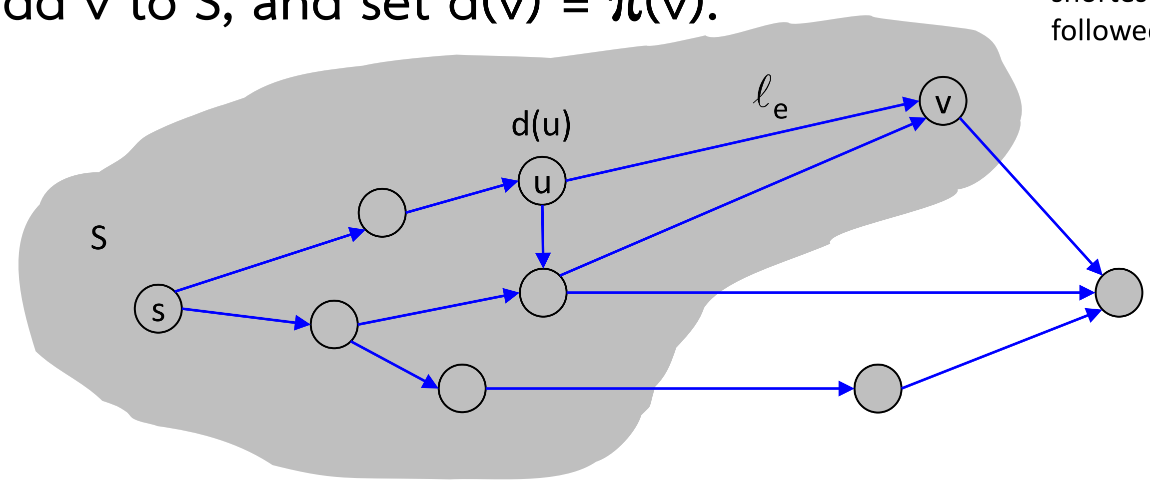
Dijkstra's algorithm.

- Maintain a set of explored nodes S for which we have determined the shortest path distance d(u) from s to u.
- Initialize S = { s }, d(s) = 0.
- Repeatedly choose unexplored node v which minimizes

$$\pi(v) = \min_{e = (u,v)\,:\, u \in S} d(u) + \ell_e,$$

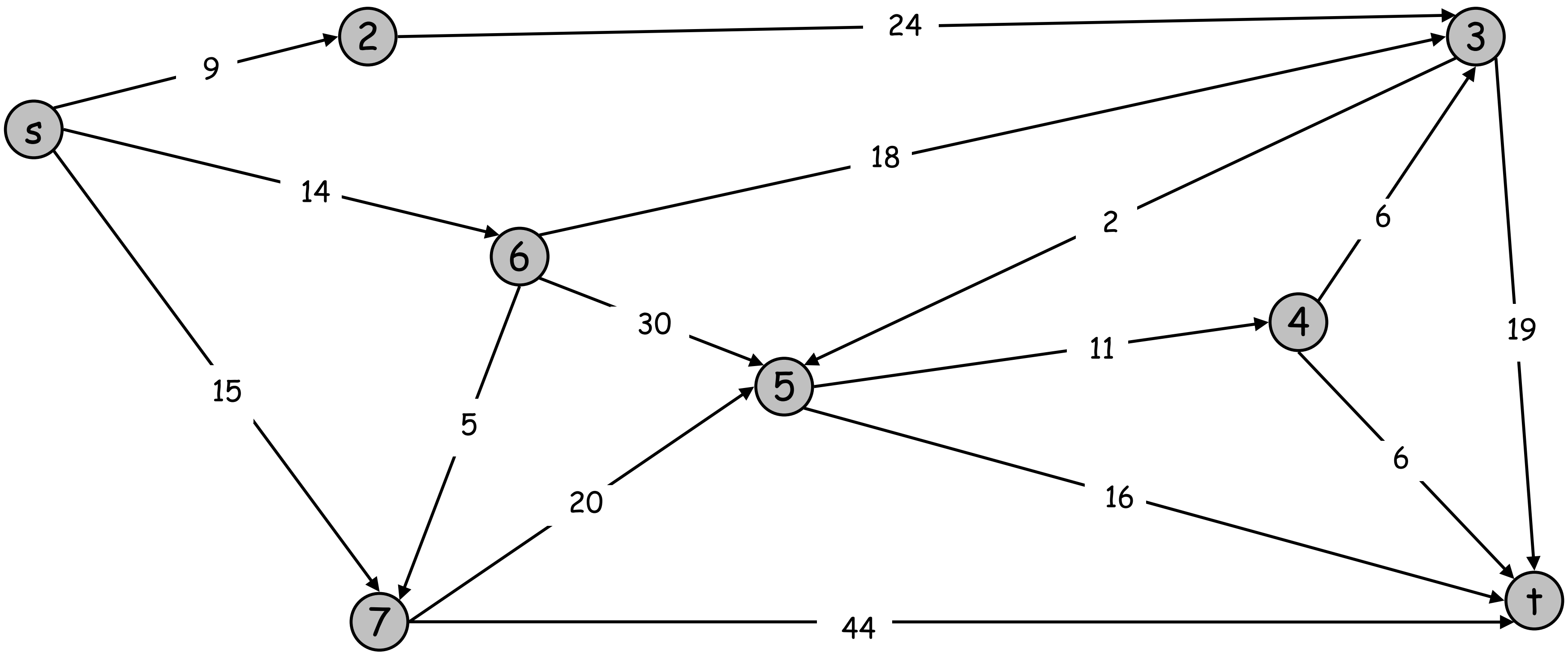
shortest path to some u in explored part, followed by a single edge (u, v)

add v to S, and set d(v) = π(v).



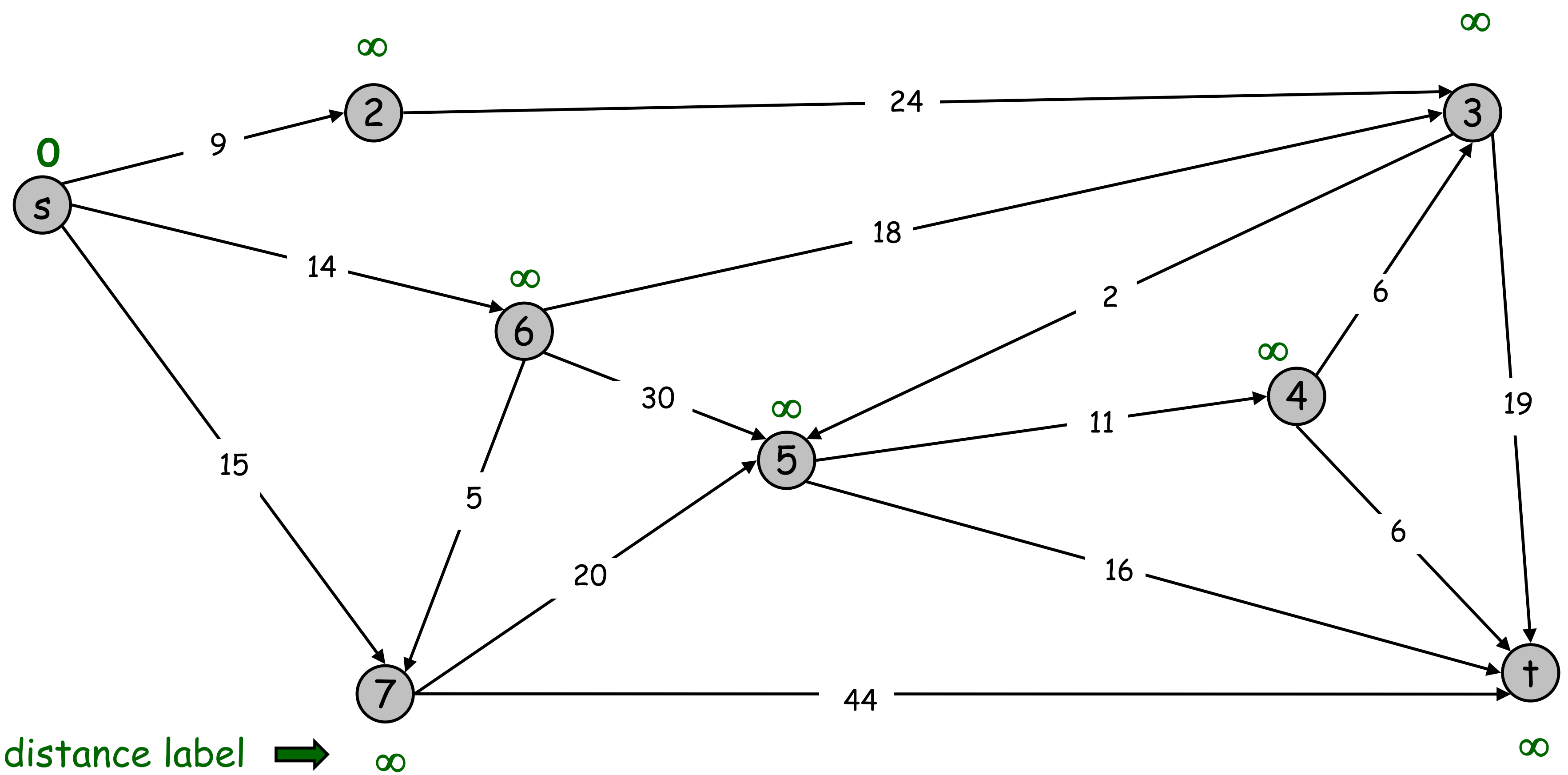
Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.



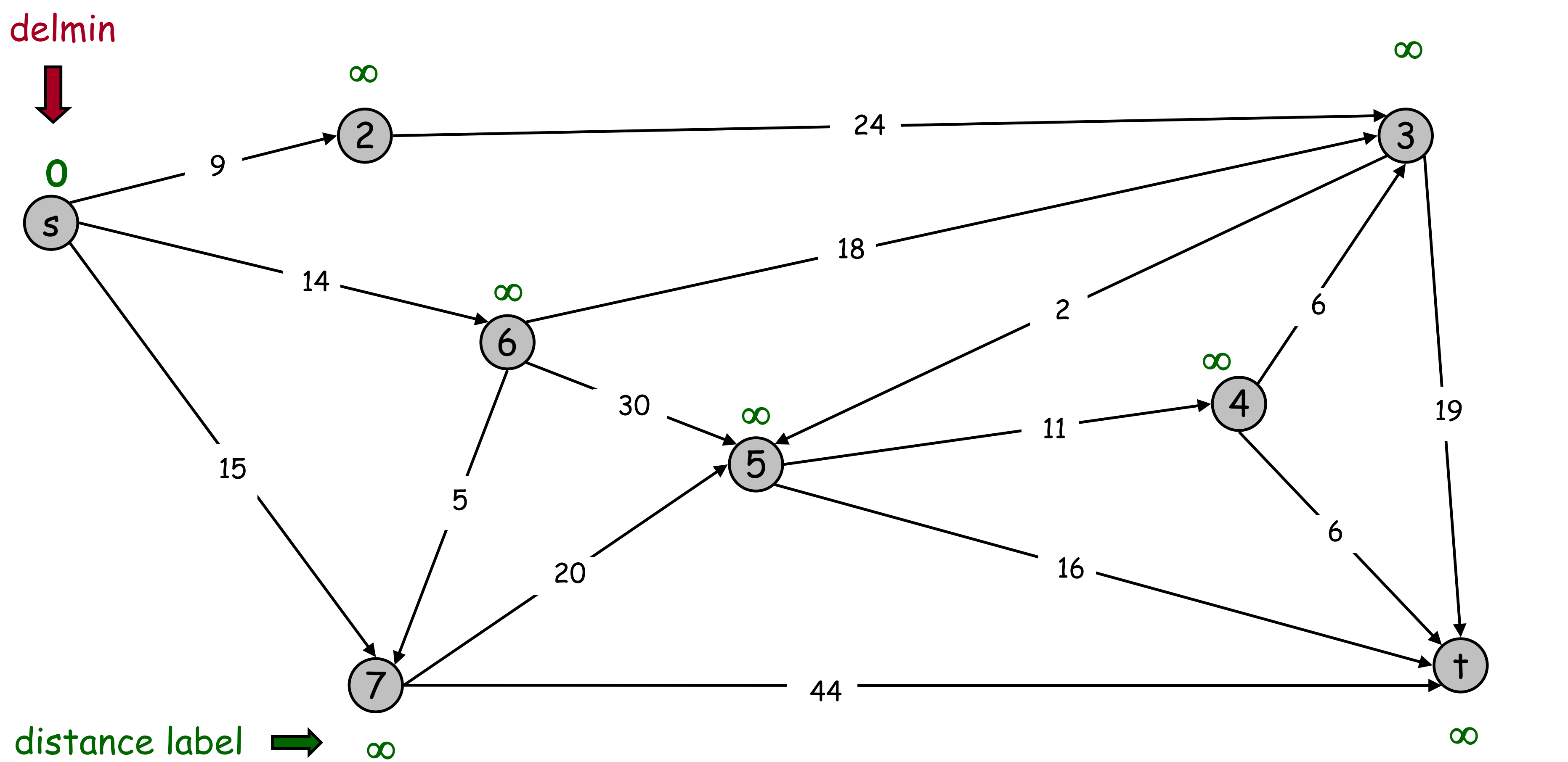
Dijkstra's Shortest Path Algorithm

S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }



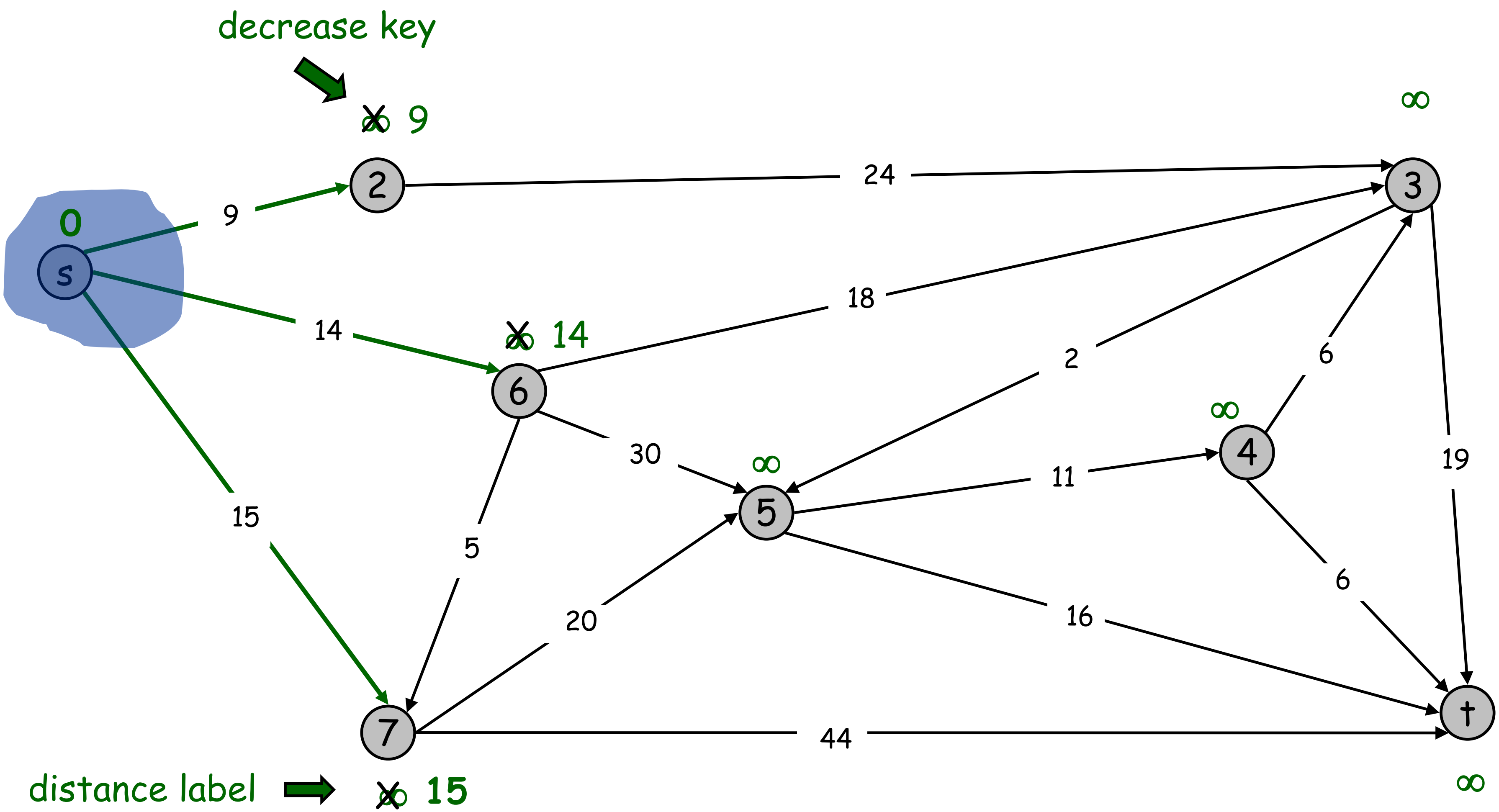
Dijkstra's Shortest Path Algorithm

S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }



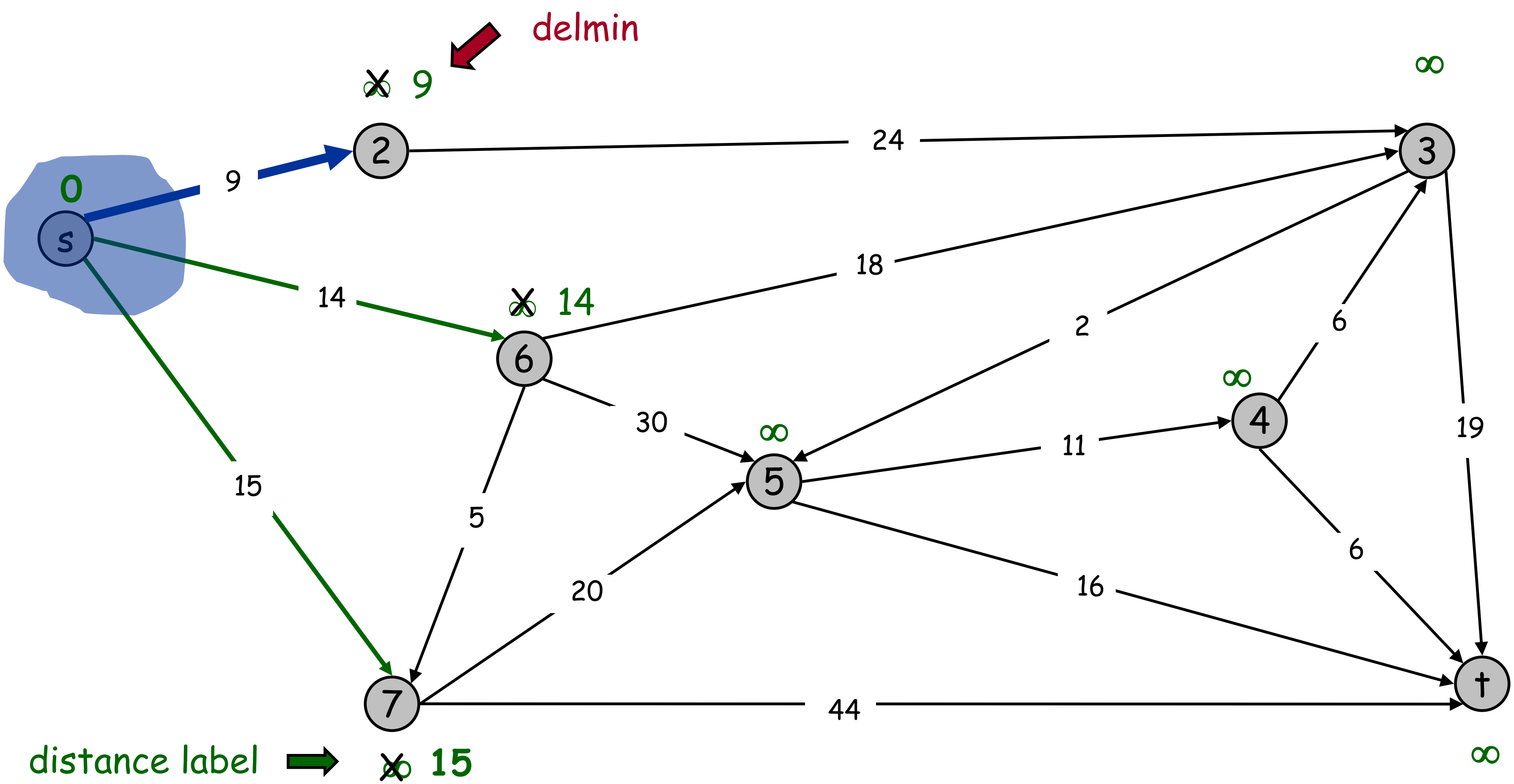
Dijkstra's Shortest Path Algorithm

S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }



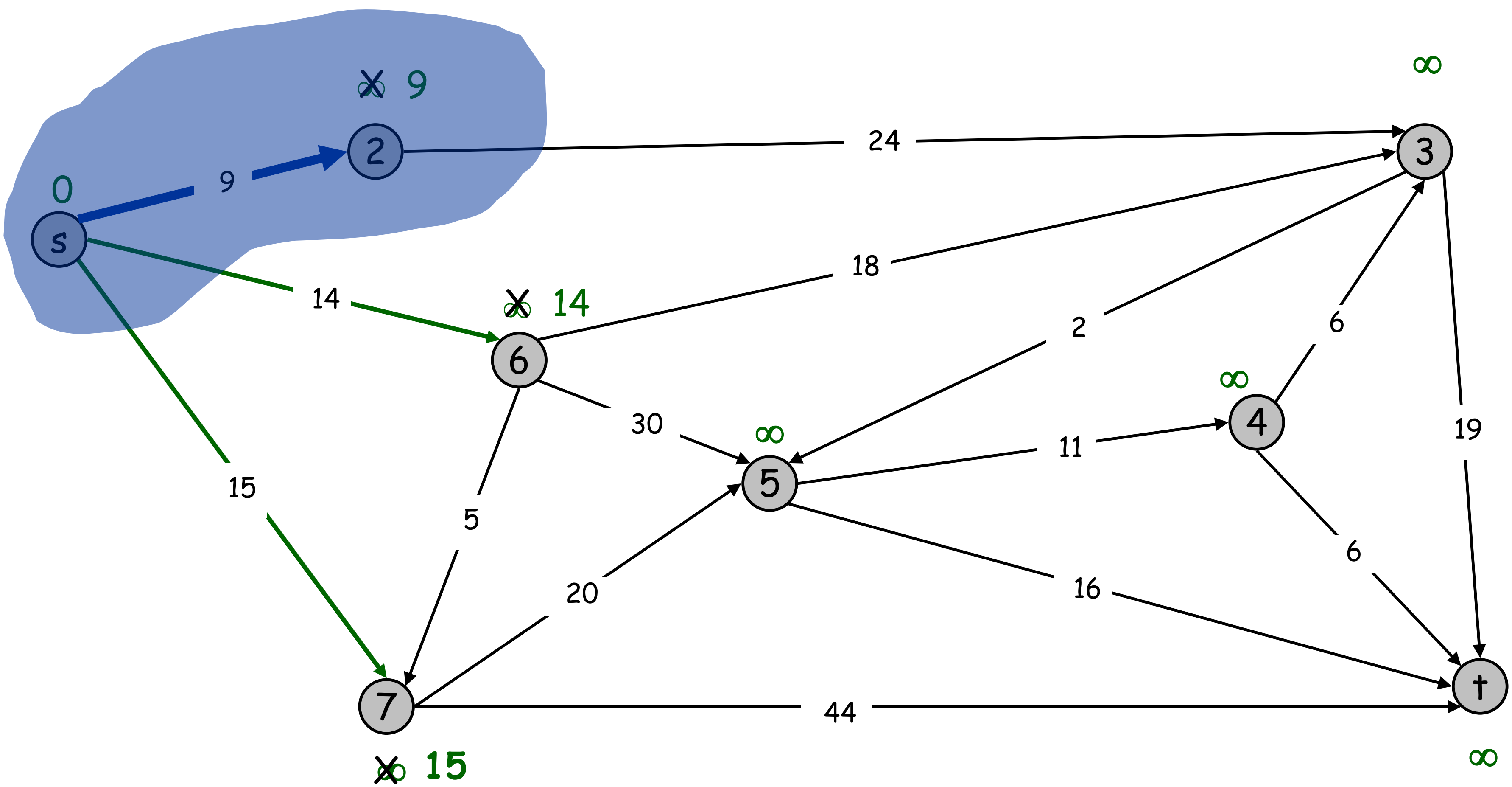
Dijkstra's Shortest Path Algorithm

S = { s }
PQ = { 2, 3, 4, 5, 6, 7, t }



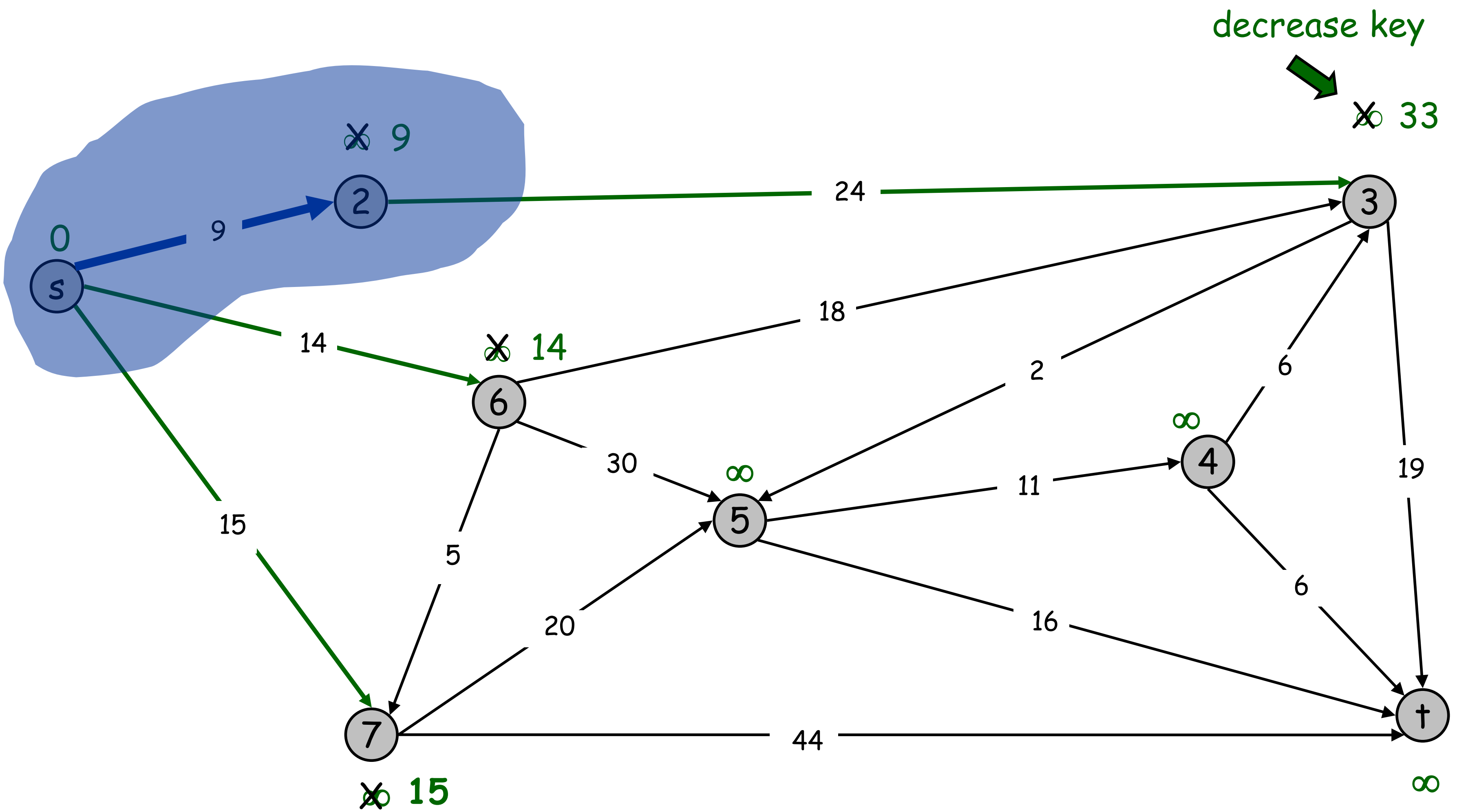
Dijkstra's Shortest Path Algorithm

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }



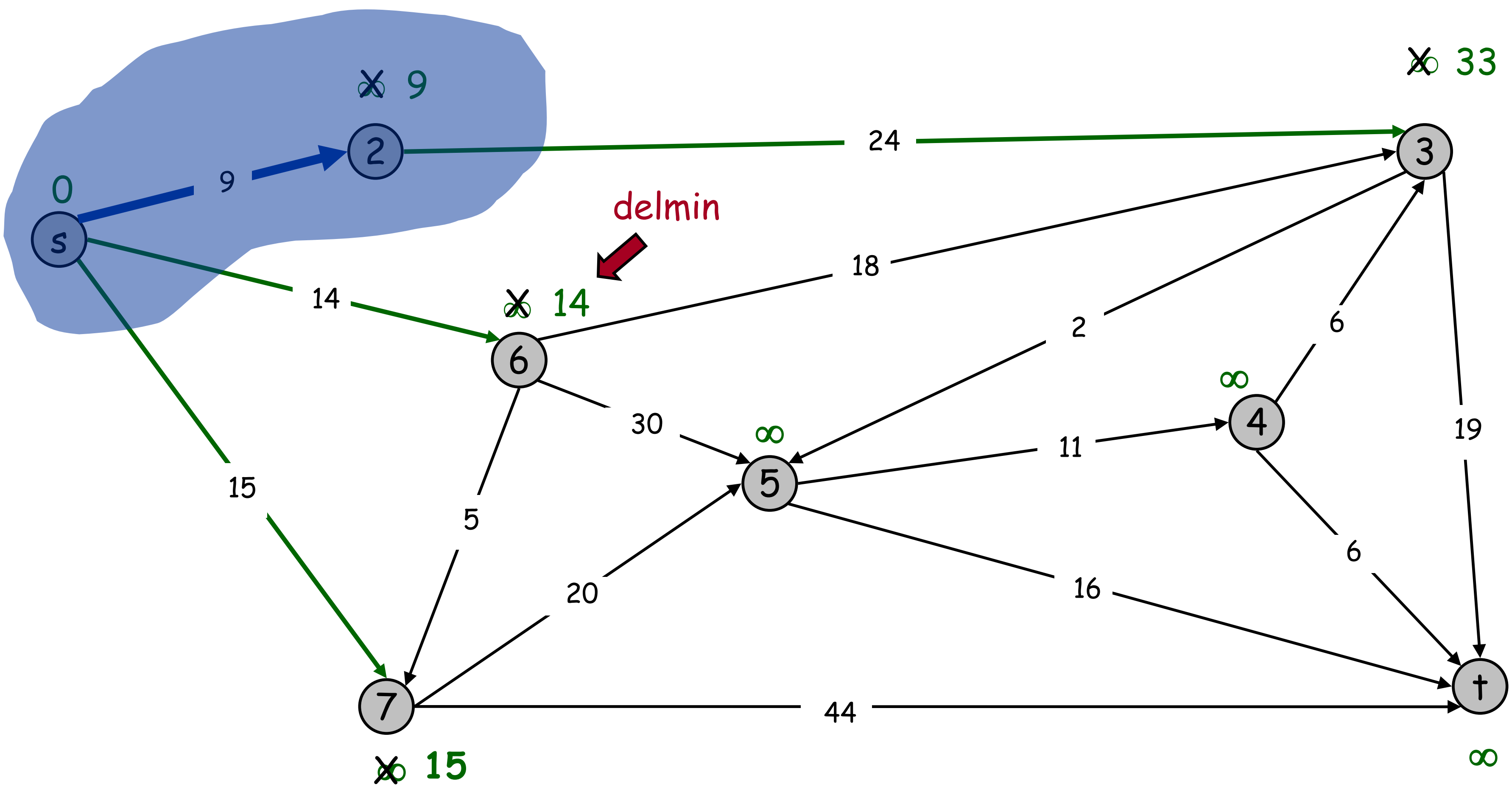
Dijkstra's Shortest Path Algorithm

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }



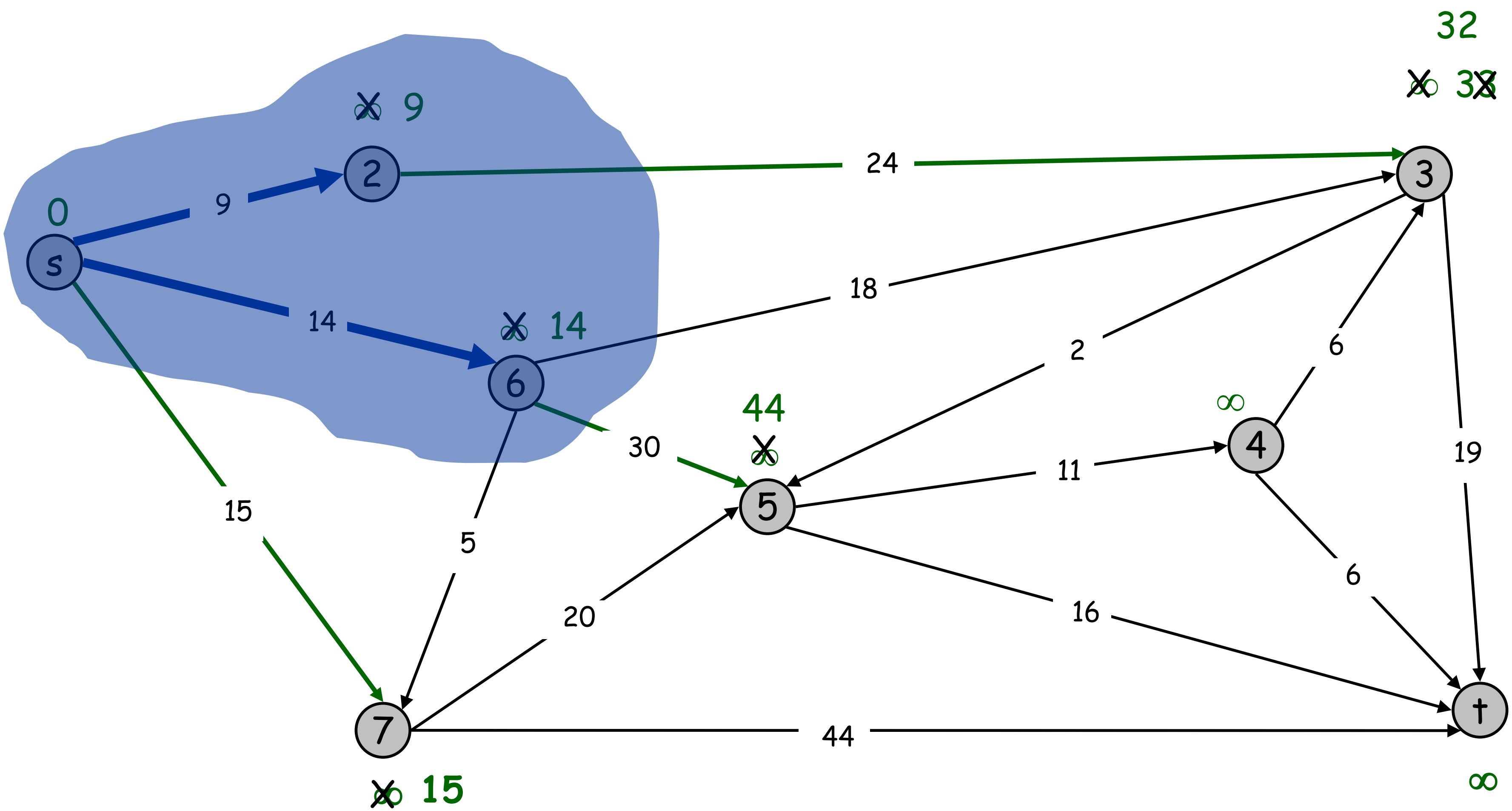
Dijkstra's Shortest Path Algorithm

S = { s, 2 }
PQ = { 3, 4, 5, 6, 7, t }



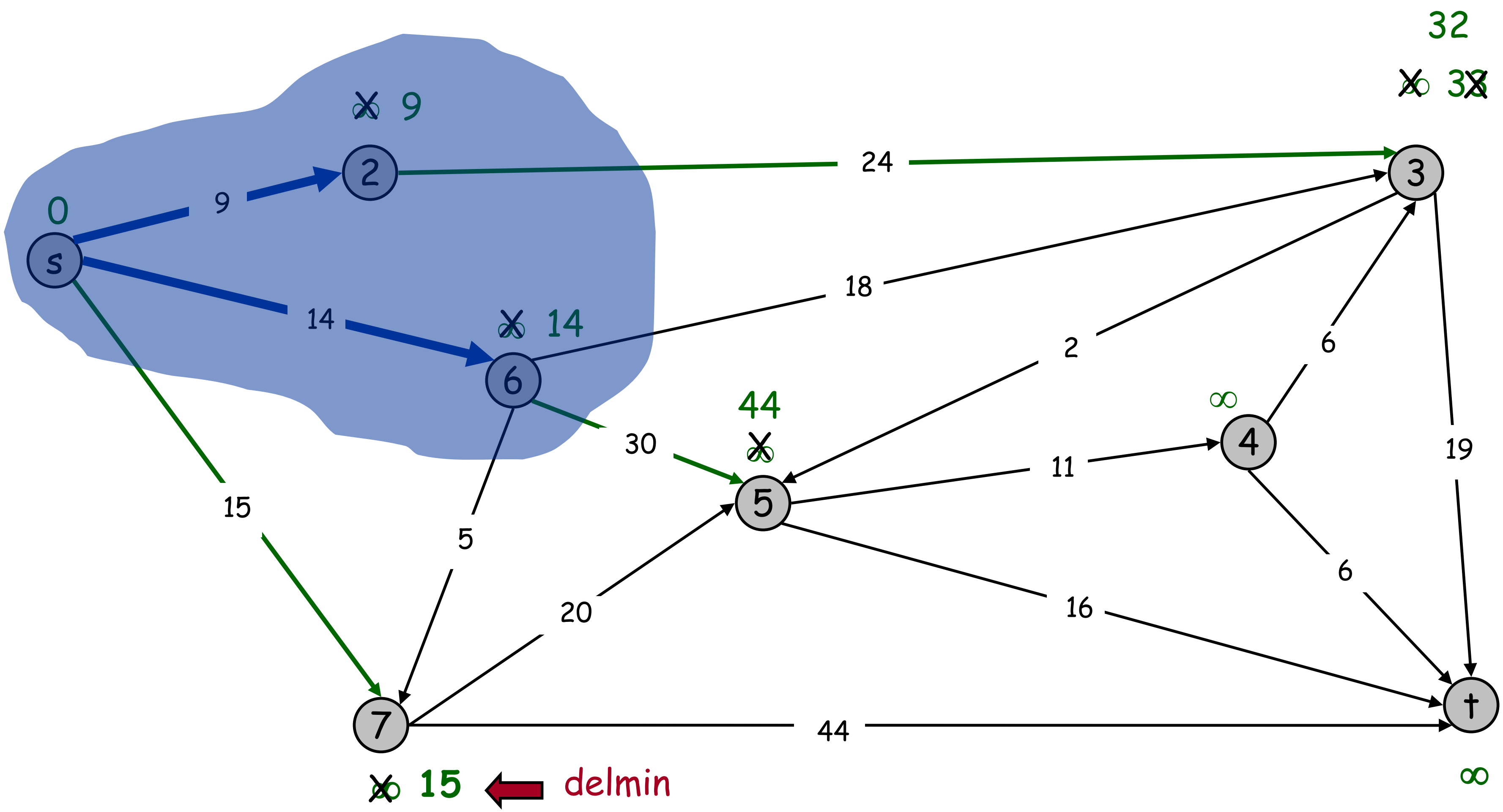
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }



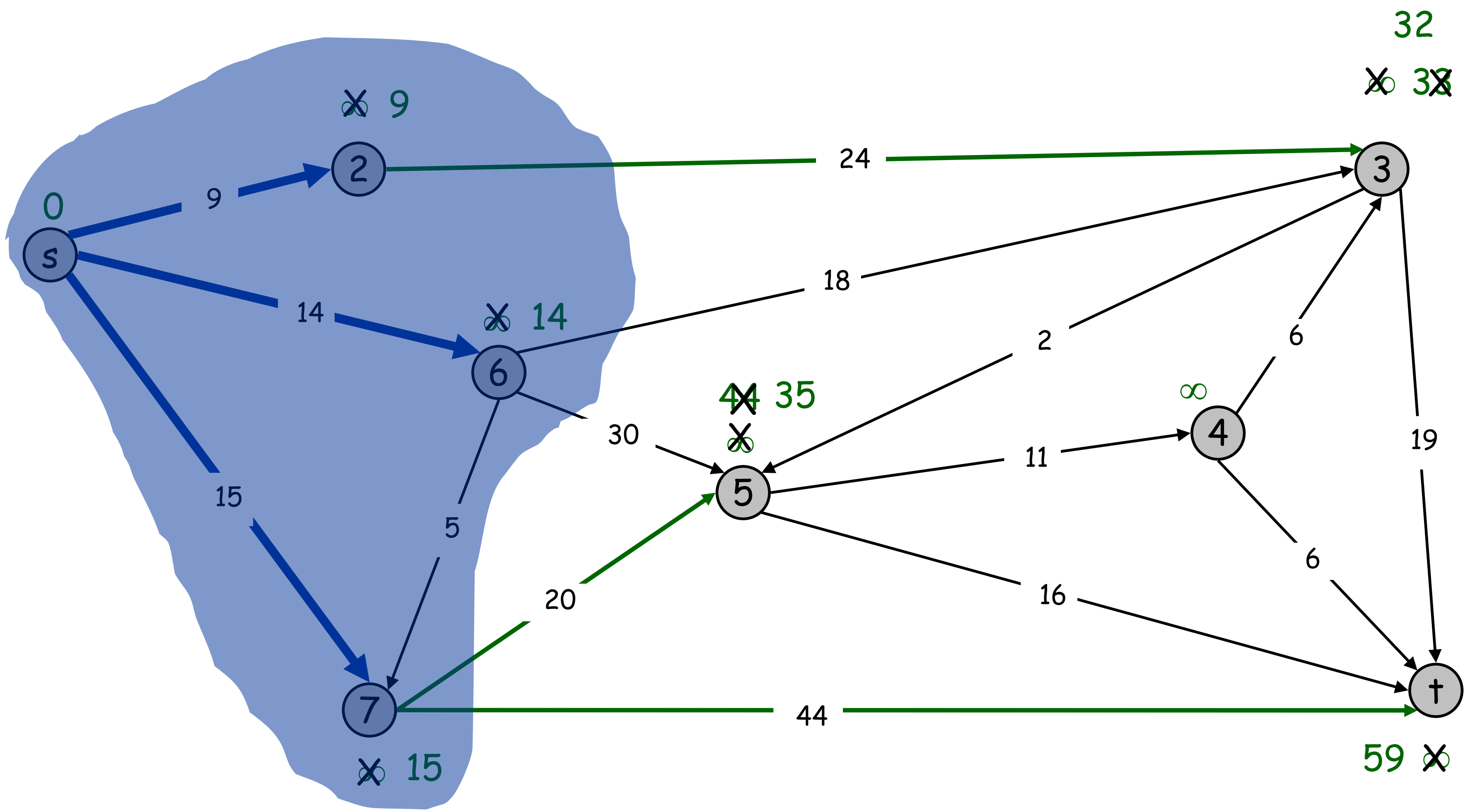
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }



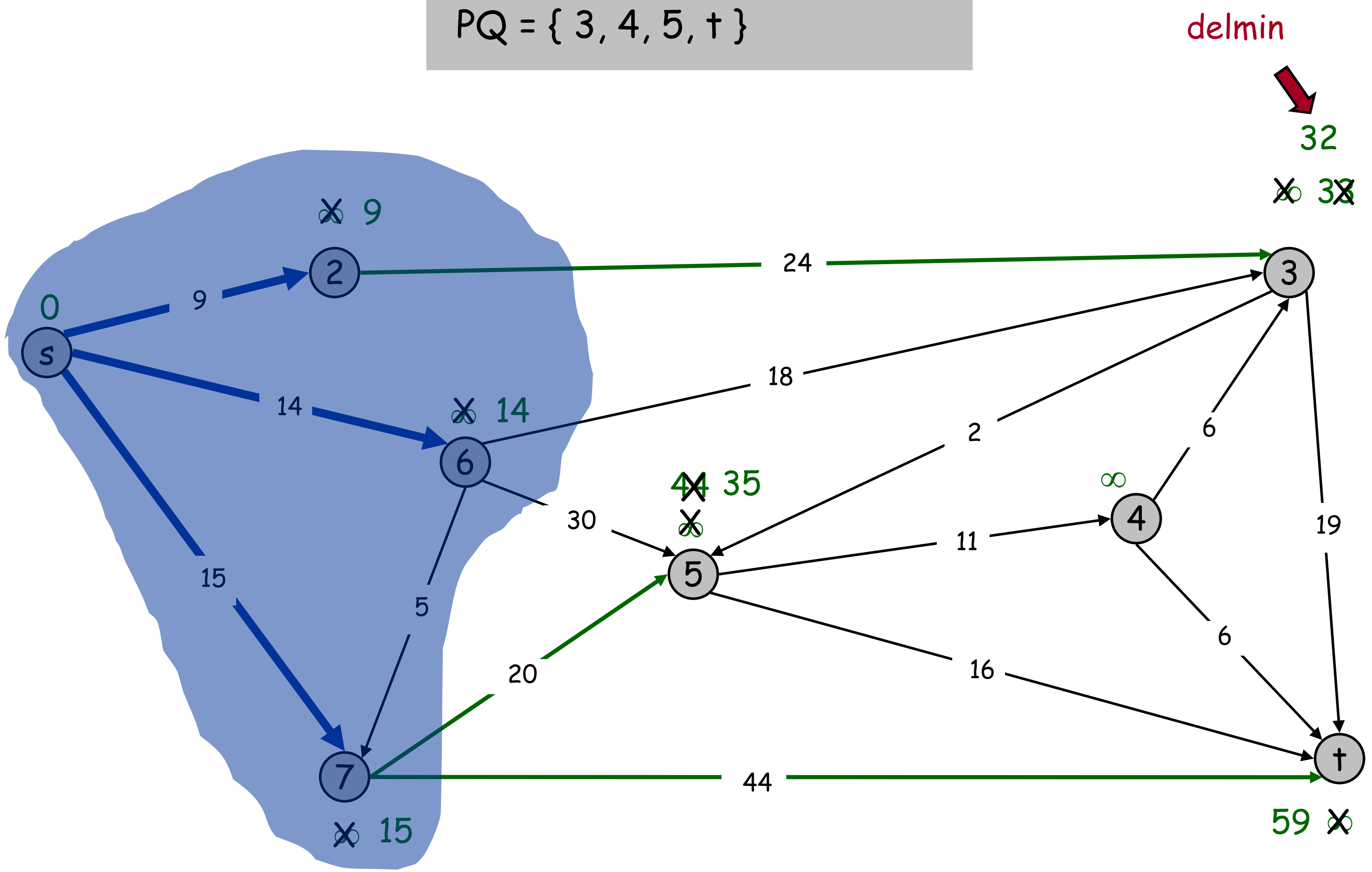
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }



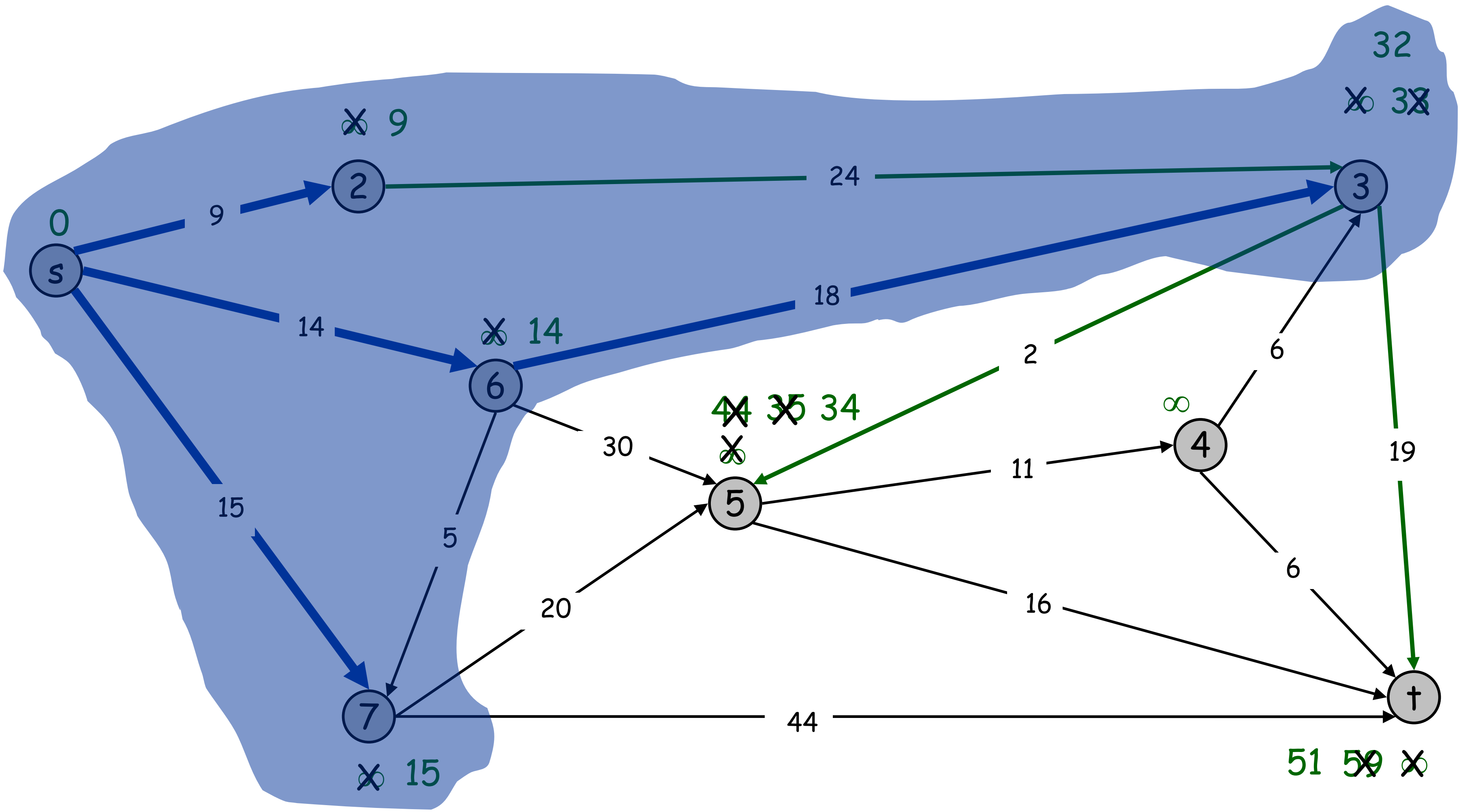
Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }



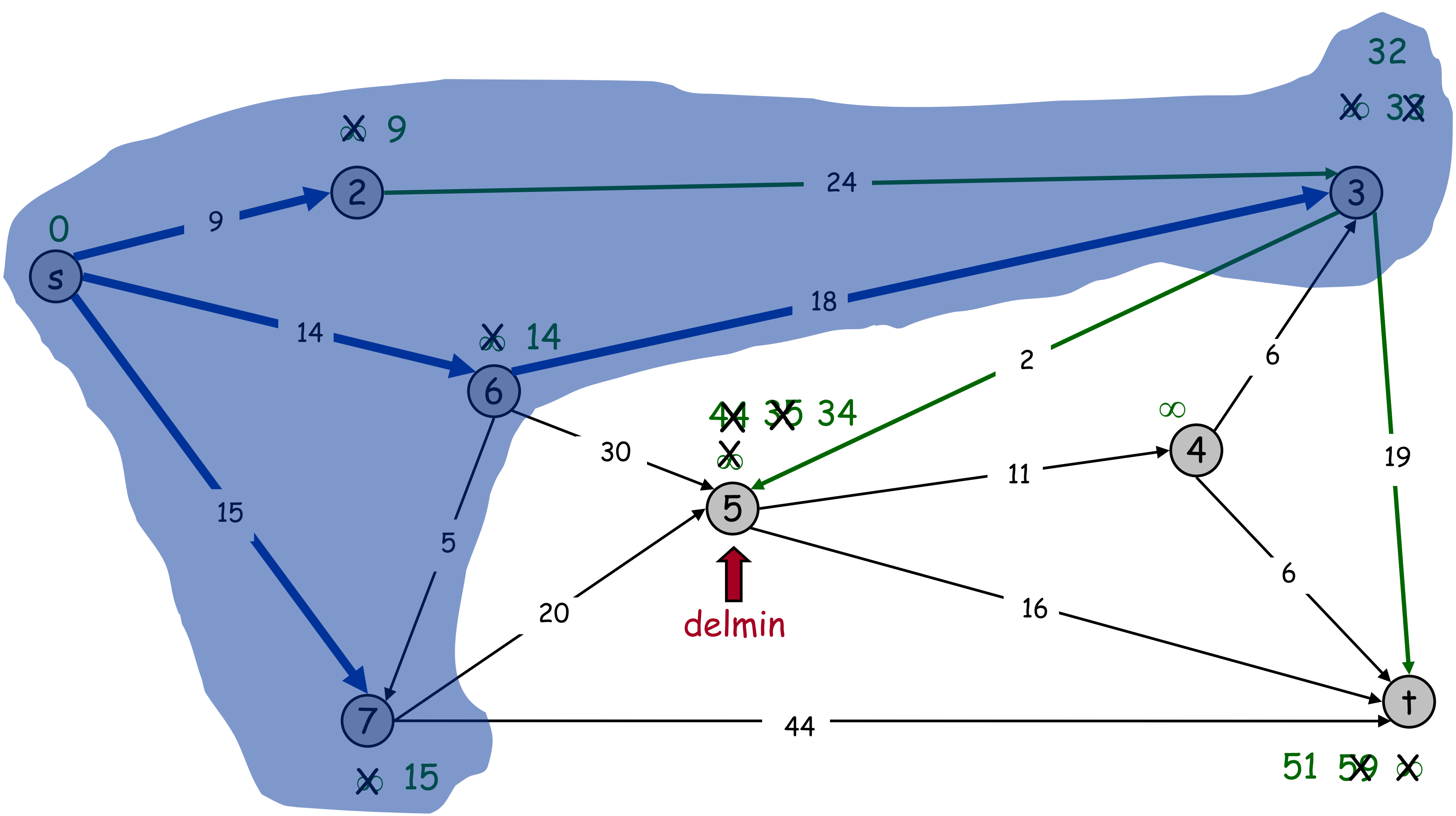
Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 6, 7 }
PQ = { 4, 5, t }



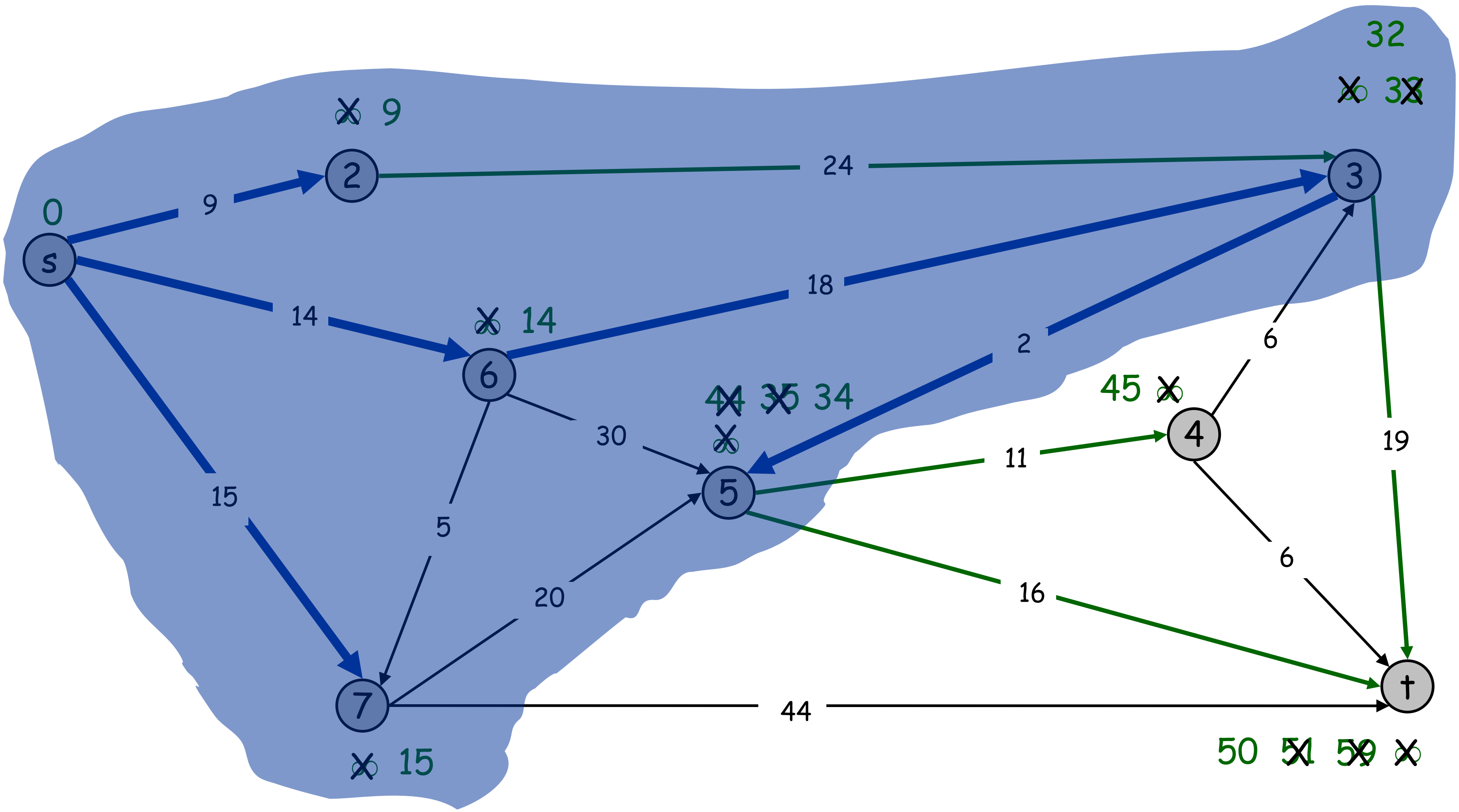
Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 6, 7 }
PQ = { 4, 5, t }



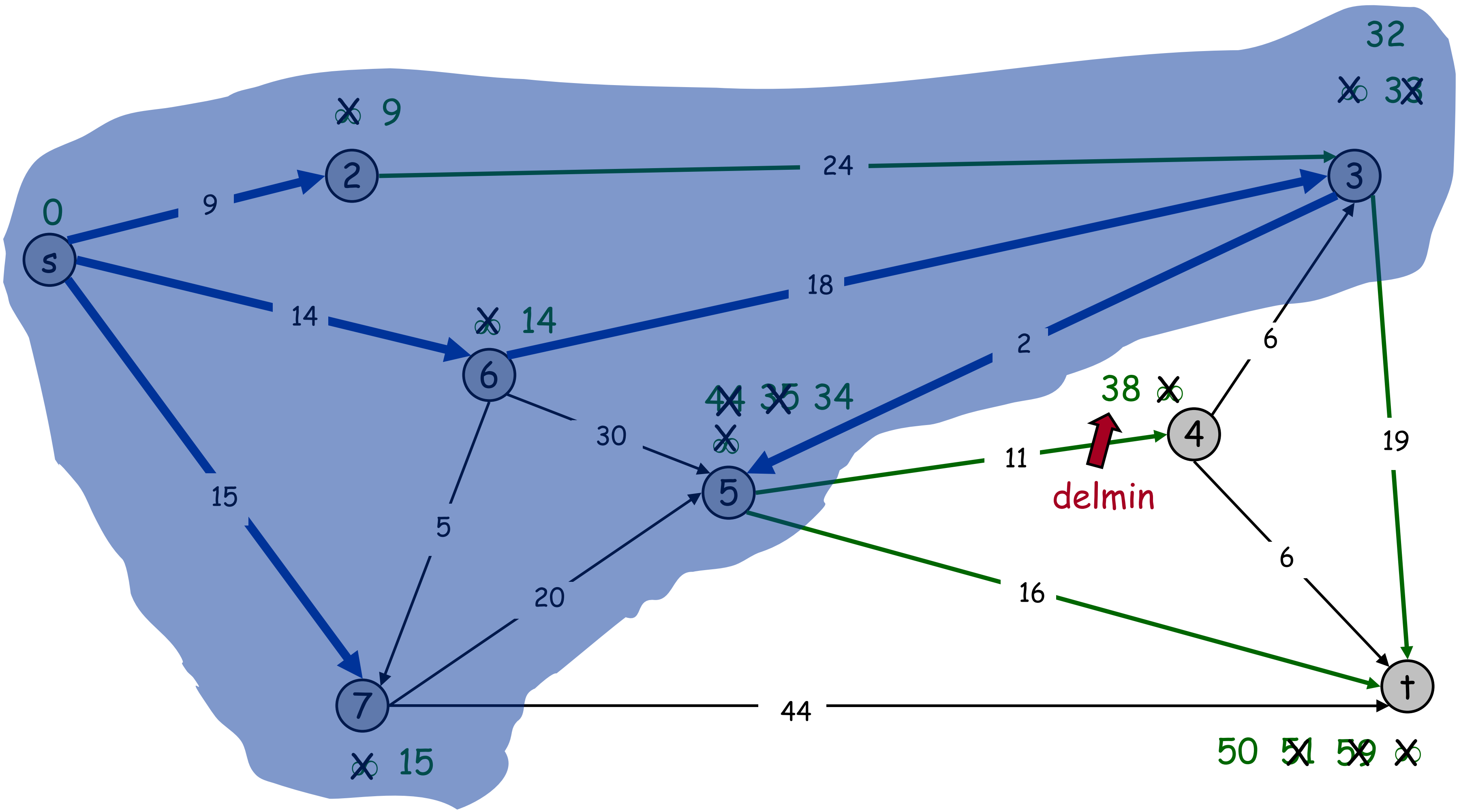
Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }



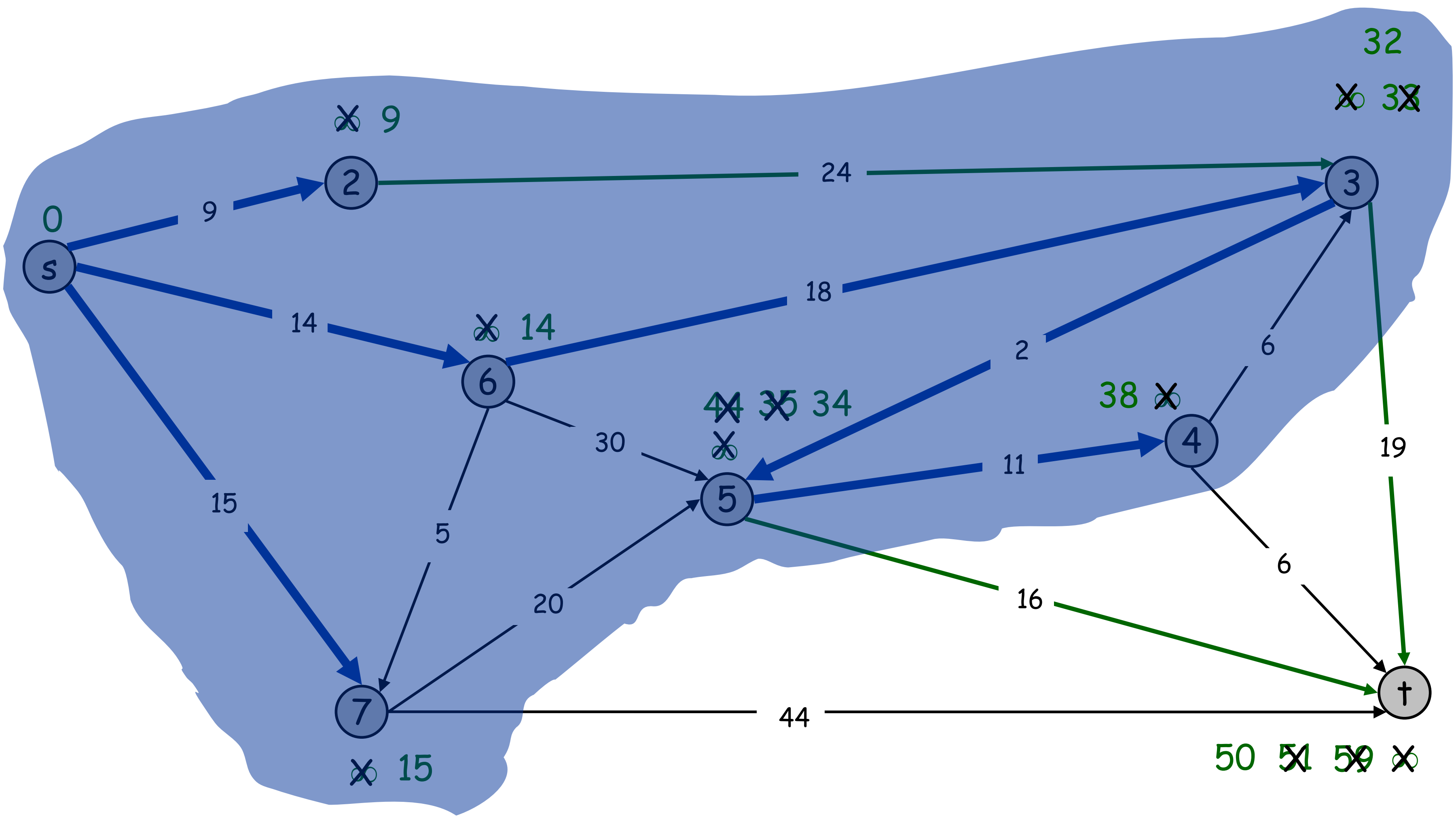
Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }



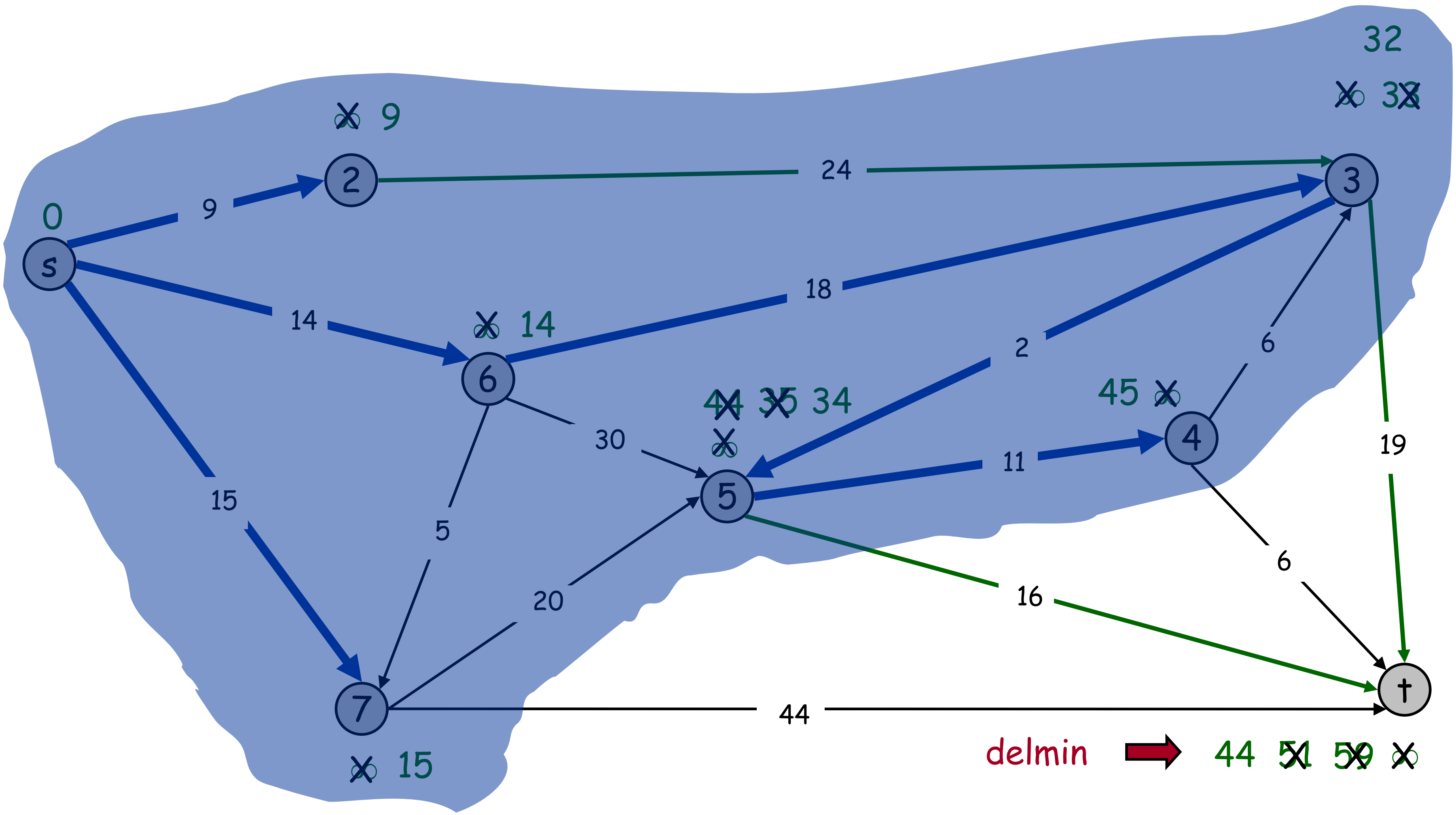
Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7 }
PQ = { t }



Dijkstra's Shortest Path Algorithm

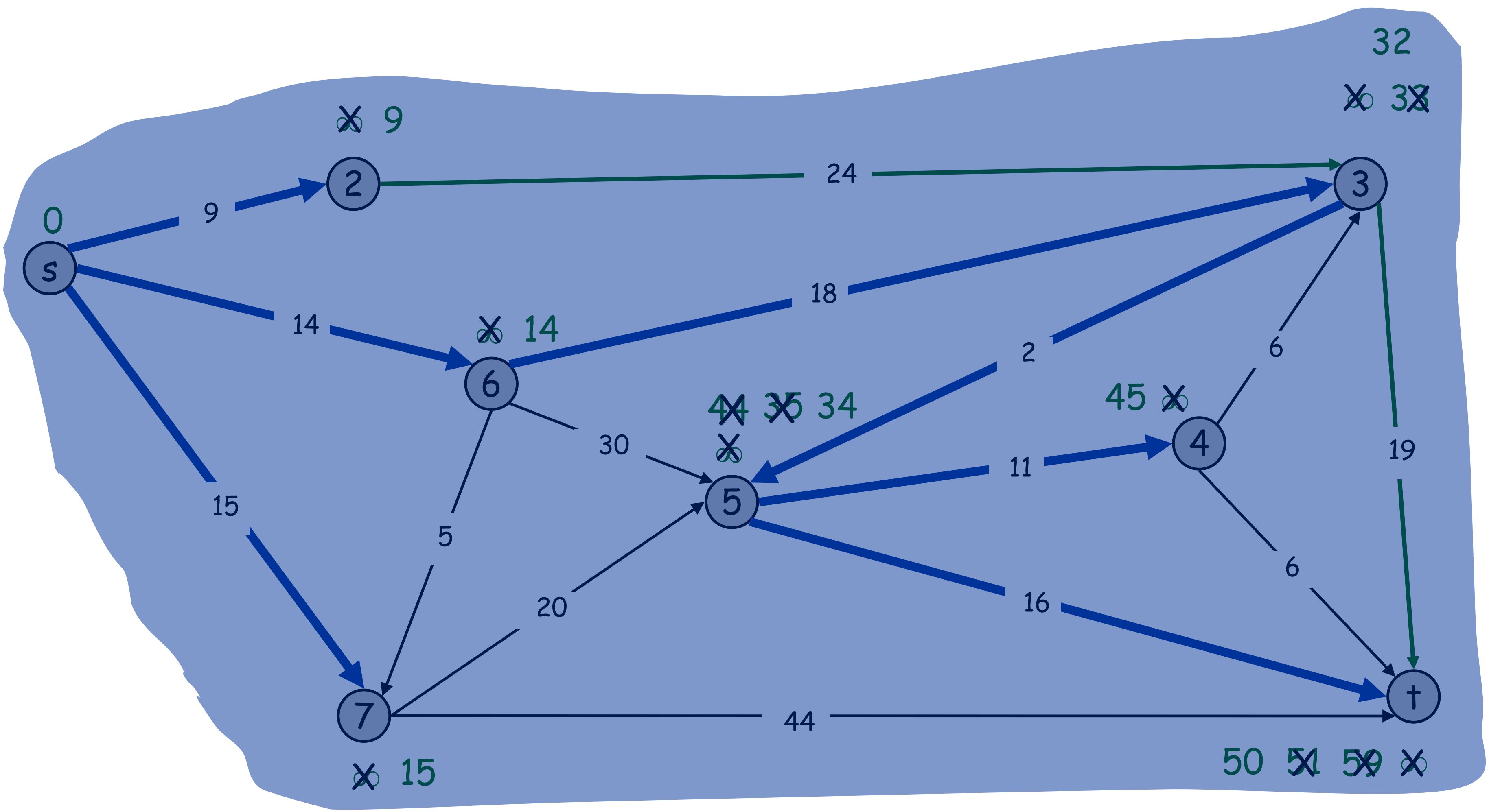
S = { s, 2, 3, 4, 5, 6, 7 }
PQ = { t }



Dijkstra's Shortest Path Algorithm

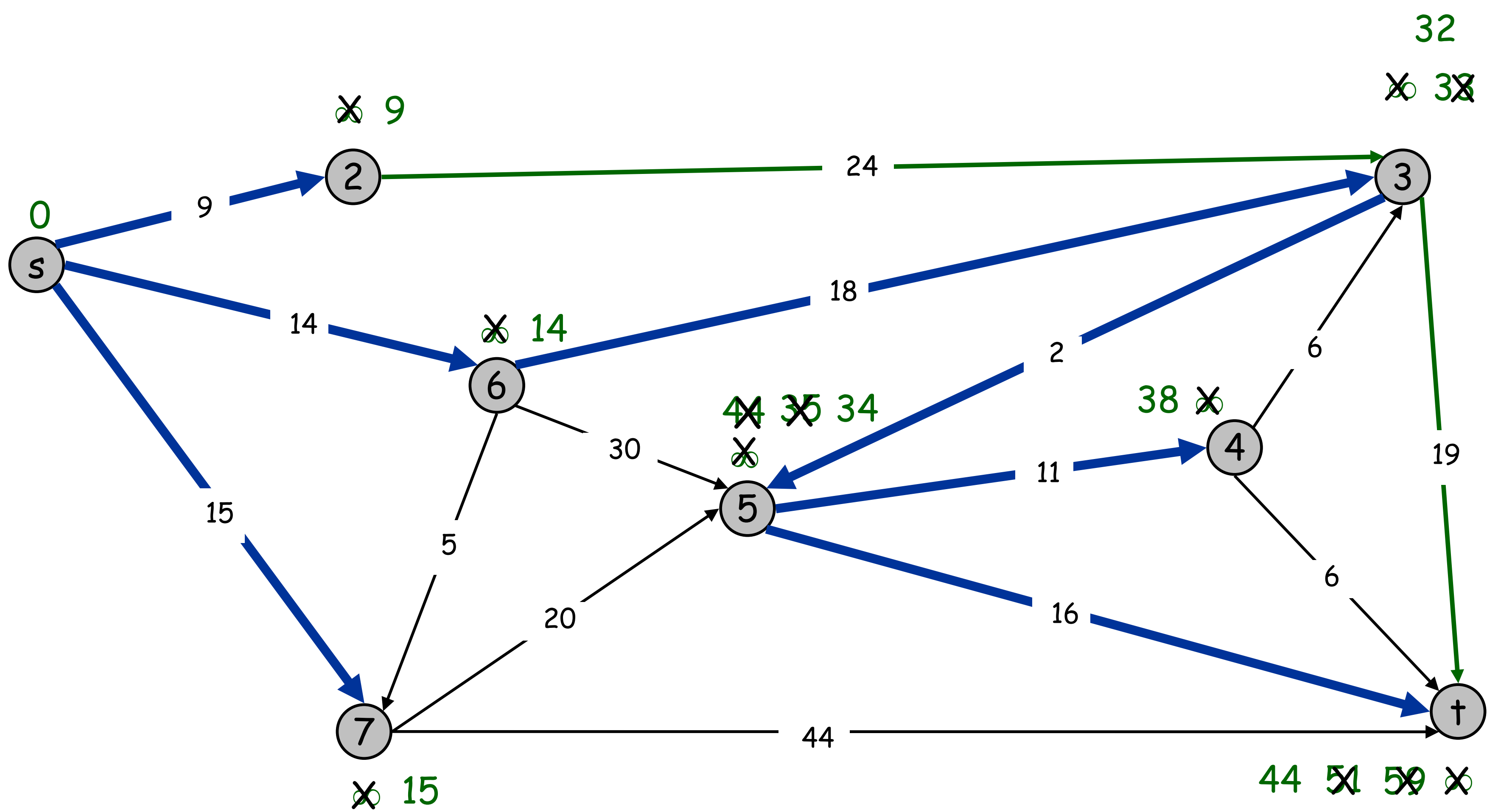
S = { s, 2, 3, 4, 5, 6, 7, t }

PQ = { }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



ทวนอัลกอริทึมอีกรอบ

Algorithm:

1. Initially **Dset** = **src,** dist[s]=0, dist[v]= ∞

2. While all the elements in the graph are not added to **'Dset'**

A. Let ‘u’ be any vertex not in ‘Dset’ and has minimum label dist[u]

B. Add ‘u’ to Dset

C. Update the label of the elements adjacent to u

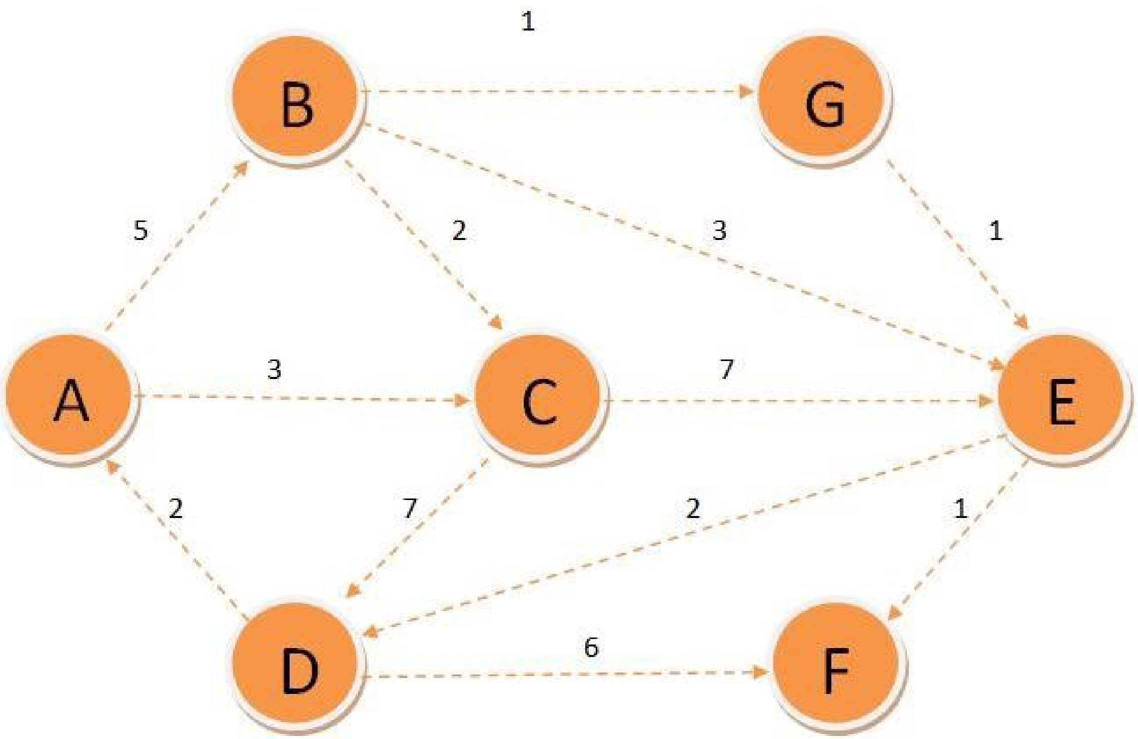
    For each vertex ‘v’ adjacent to u

        If ‘v’ is not in ‘Dset’ then

            If dist[u]+weight(u,v)<dist[v]

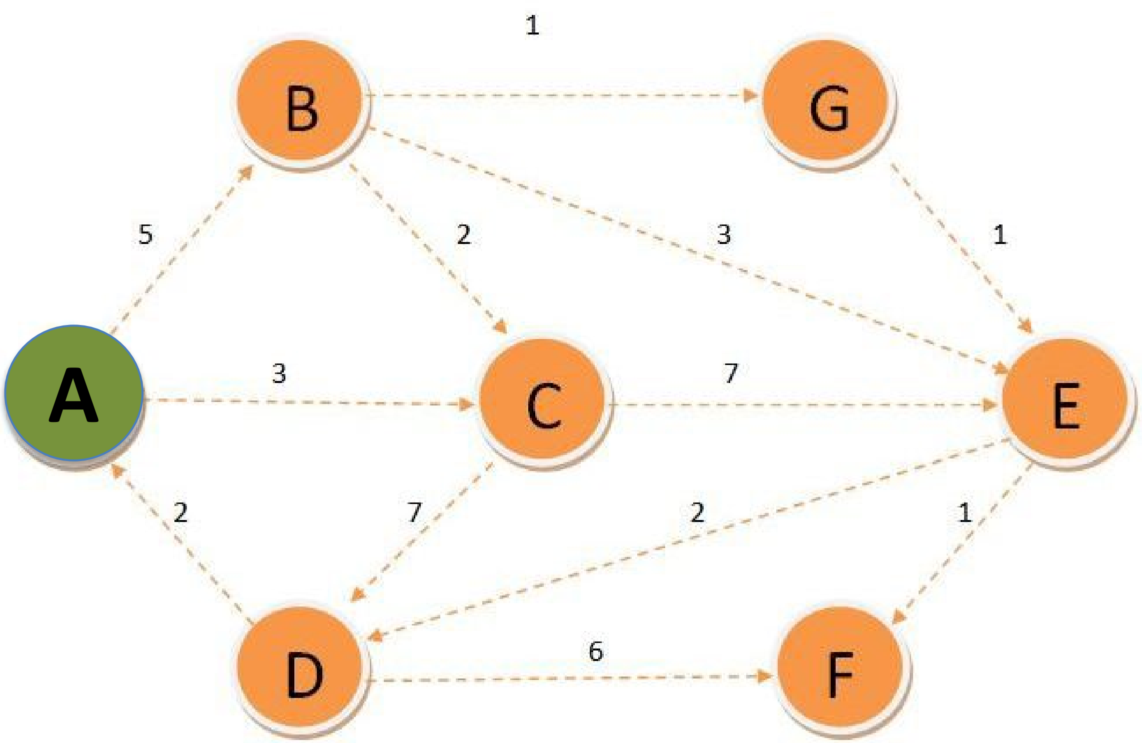
                then Dist[v]=dist[u]+weight(u,v)

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



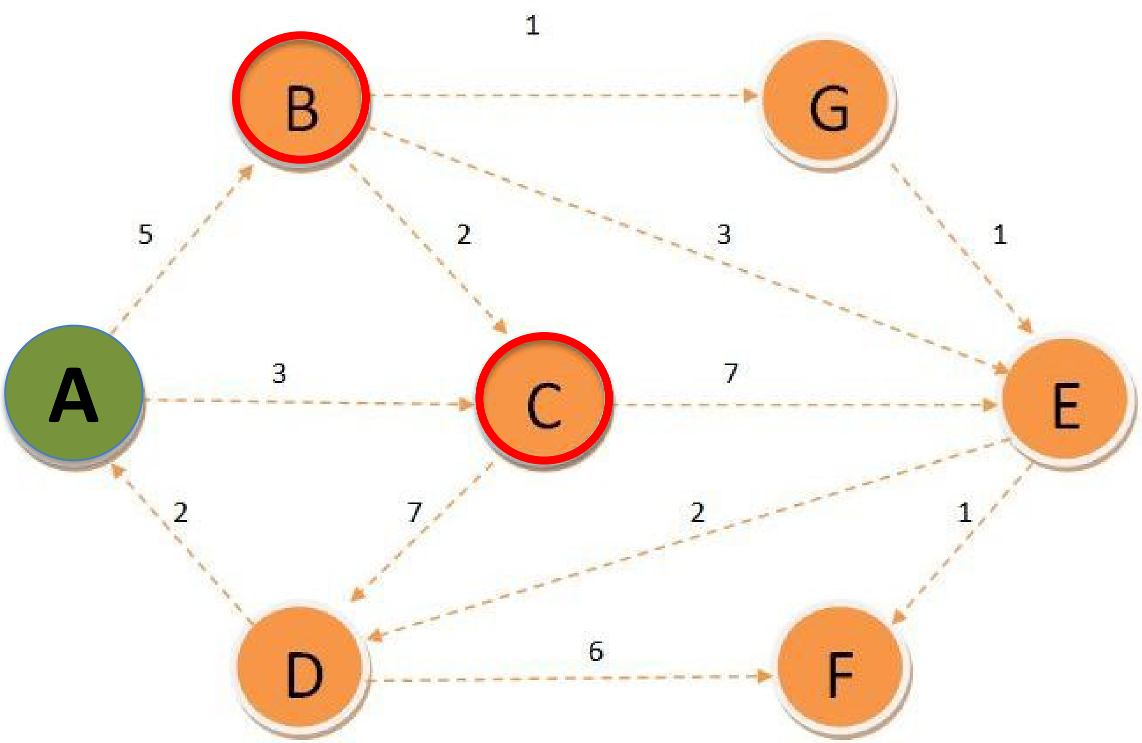
	Dset	Dist	Index
A	T	0	-
B	F	∝	-
C	F	∝	-
D	F	∝	-
E	F	∝	-
F	F	∝	-
G	F	∝	-

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



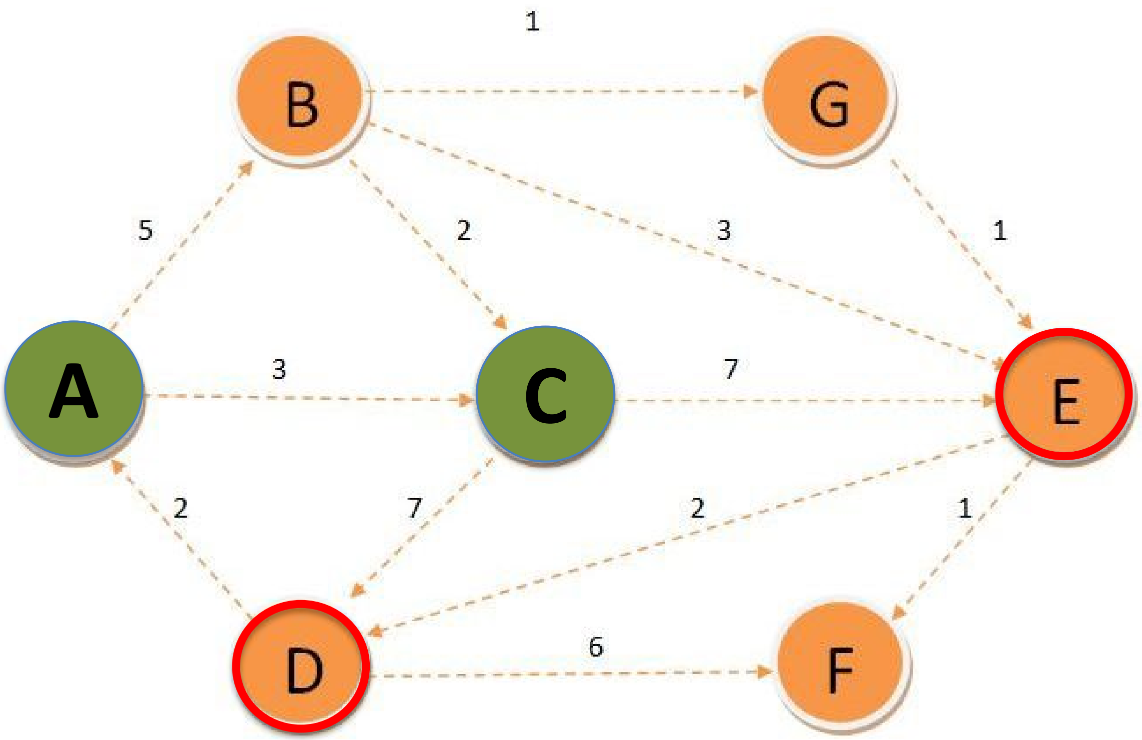
	Dset	Dist	Index
A	T	0	-
B	F	∝	-
C	F	∝	-
D	F	∝	-
E	F	∝	-
F	F	∝	-
G	F	∝	-

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



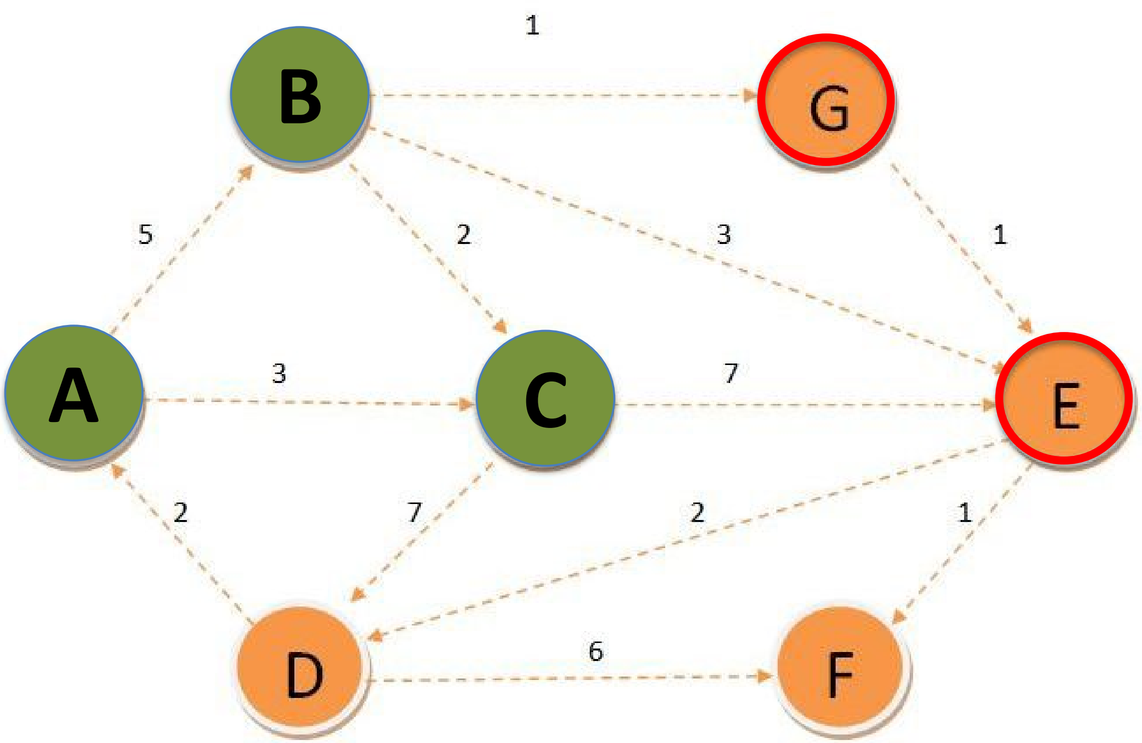
	Dset	Dist	Index
A	T	0	-
B	F	5	A
C	T	3	A
D	F	∝	-
E	F	∝	-
F	F	∝	-
G	F	∝	-

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



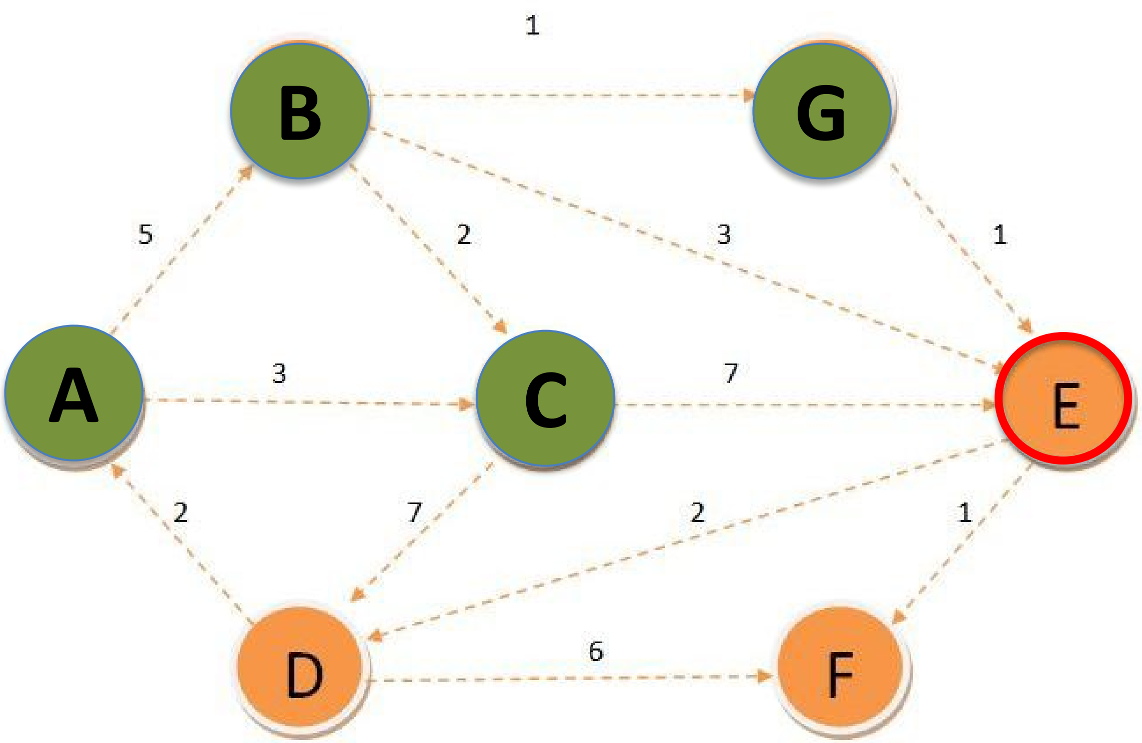
	Dset	Dist	Index
A	T	0	-
B	T	5	A
C	T	3	A
D	F	10	C
E	F	10	C
F	F	∝	-
G	F	∝	-

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



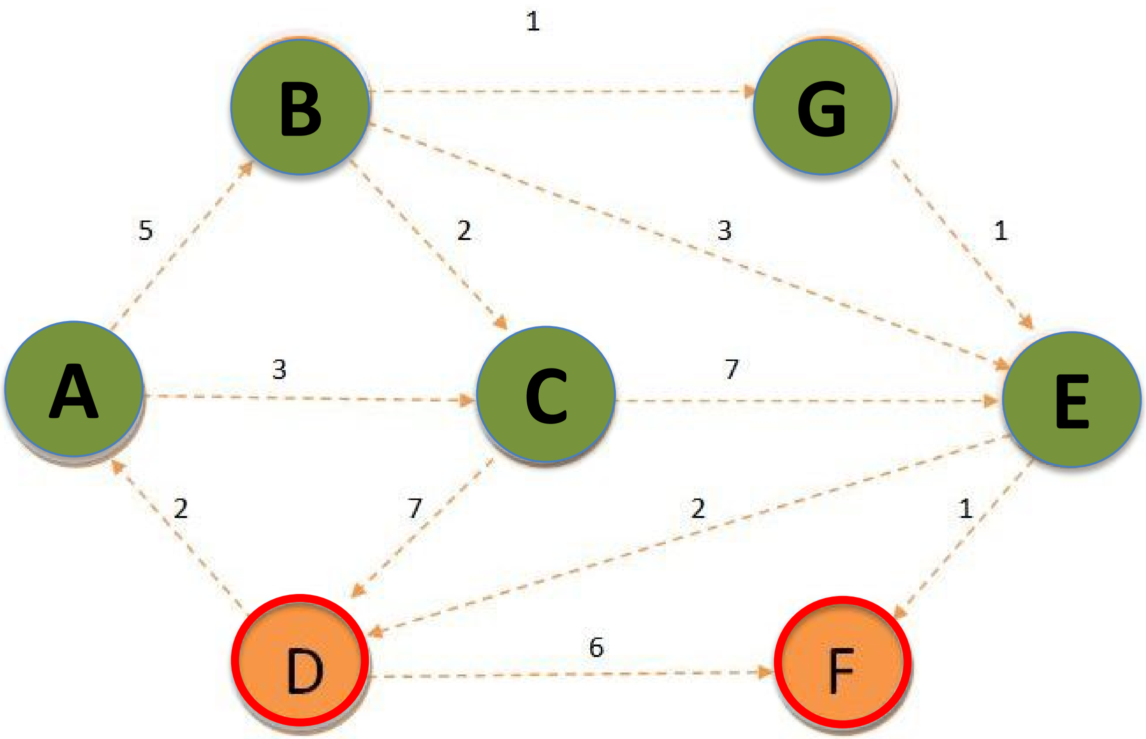
	Dset	Dist	Index
A	T	0	-
B	T	5	A
C	T	3	A
D	F	10	C
E	F	8	~~C~~ B
F	F	∝	-
G	T	6	B

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



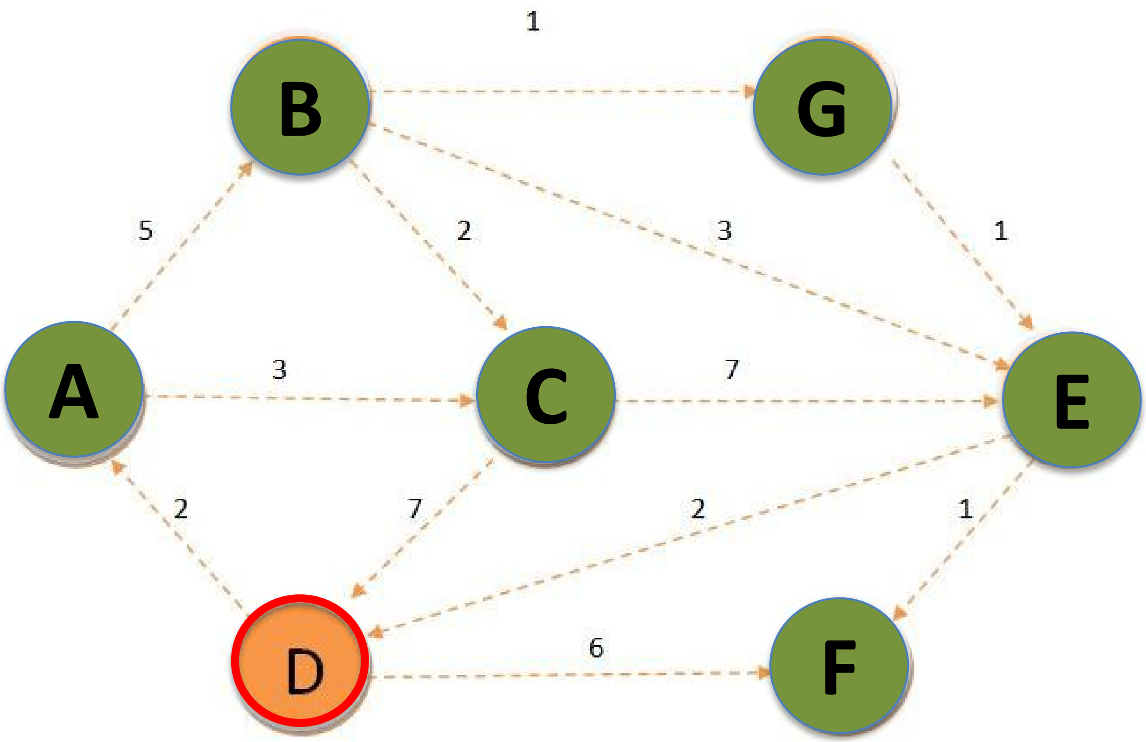
	Dset	Dist	Index
A	T	0	-
B	T	5	A
C	T	3	A
D	F	10	C
E	T	7	G
F	F	∝	-
G	T	6	B

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



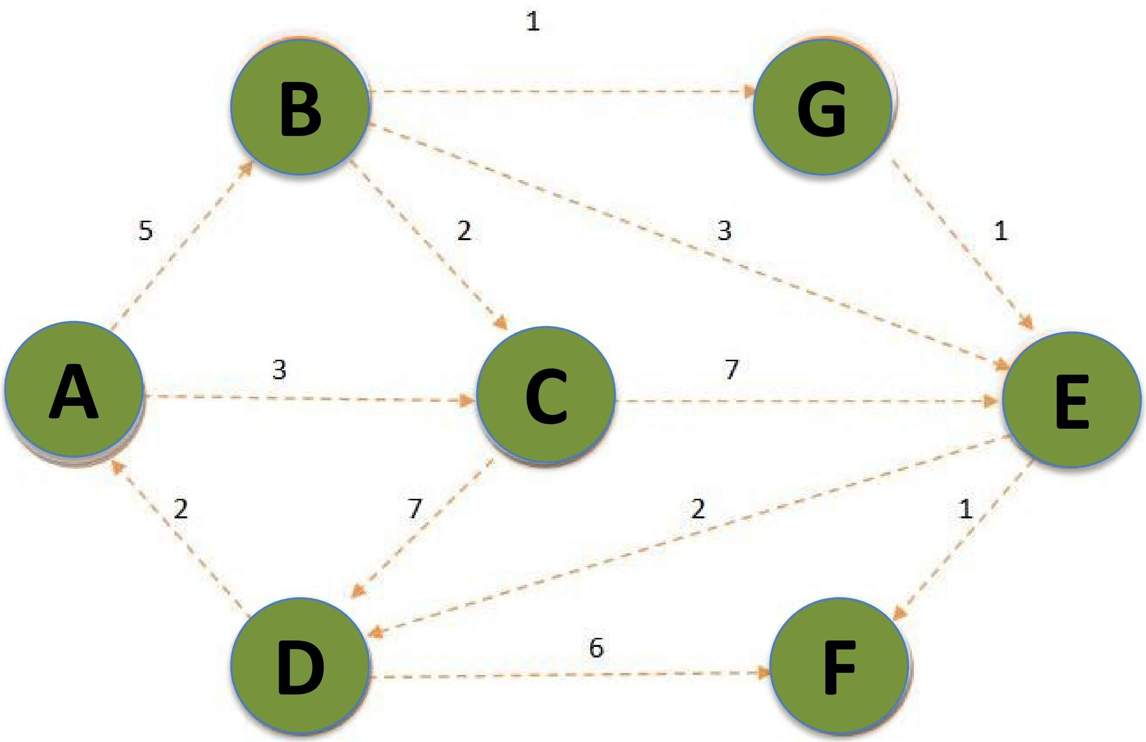
	Dset	Dist	Index
A	T	0	-
B	T	5	A
C	T	3	A
D	F	9	E
E	T	7	G
F	T	8	E
G	T	6	B

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



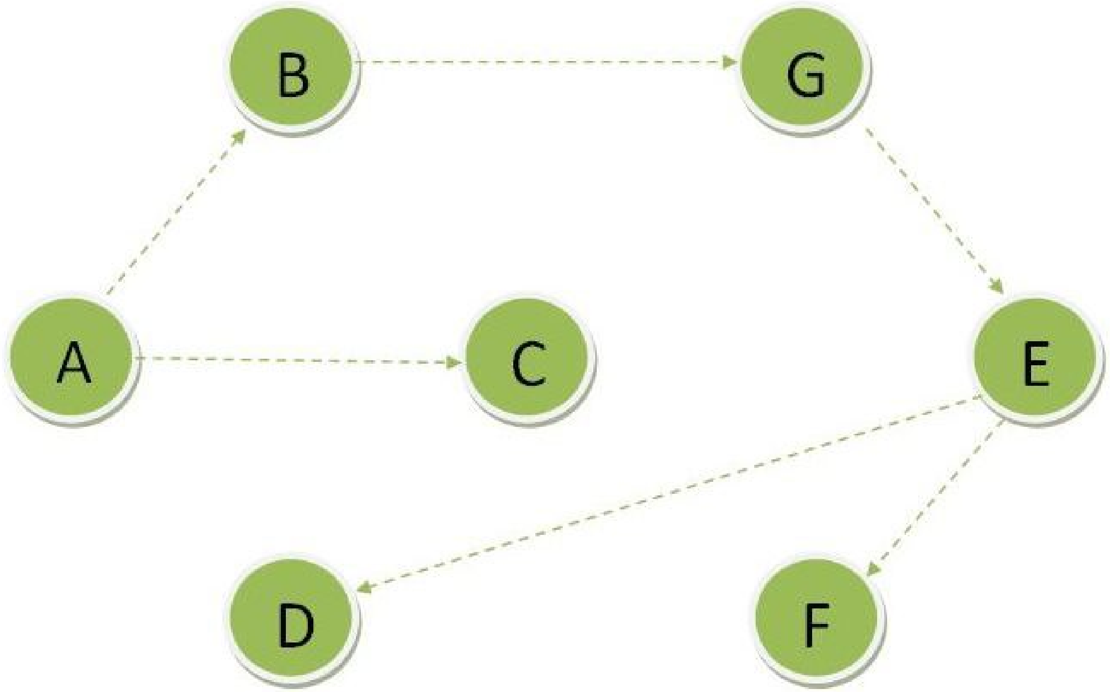
	Dset	Dist	Index
A	T	0	-
B	T	5	A
C	T	3	A
D	T	9	E
E	T	7	G
F	T	8	E
G	T	6	B

ให้หาระยะทางของจุดต่างๆ เมื่อเริ่มจาก A



	Dset	Dist	Index
A	T	0	-
B	T	5	A
C	T	3	A
D	T	9	E
E	T	7	G
F	T	8	E
G	T	6	B

ผลลัพธ์



```
#include<iostream>
#include<climits>     /*Used for INT_MAX*/
using namespace std;
#define vertex 7      /*the total no of vertices in the graph*/
int minimumDist(int dist[], bool Dset[]);
void dijkstra(int graph[vertex][vertex], int src);
int main(){
  int graph[vertex][vertex]={{0,5,3,0,0,0,0},{0,0,2,0,3,0,1},{0,0,0,7,7,0,0},{2,0,0,0,0,6,0},
{0,0,0,2,0,1,0},{0,0,0,0,0,0,0}, {0,0,0,0,1,0,0}};
  dijkstra(graph,0);
  cout<<"Vertex\t\tDistance from source"<<endl;
  for(int i=0;i<vertex;i++) { char c=65+i;  cout<<c<<"\t\t"<<dist[i]<<endl; }
  return 0;
}
```

```
int minimumDist(int dist[], bool Dset[]) {
/*initialize min with the maximum possible value as infinity does not exist */
    int min=INT_MAX,index;
    for(int v=0;v<vertex;v++)     {
        if(Dset[v]==false && dist[v]<=min)      {
            min=dist[v];
            index=v;
        }
    }
    return index;
}
```

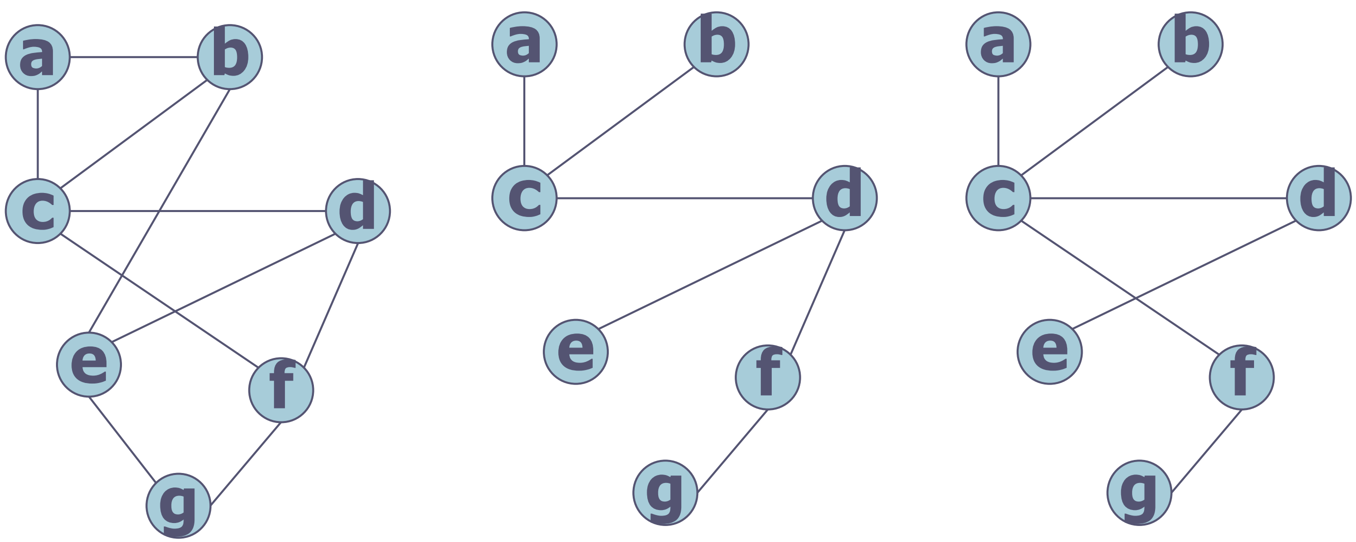
```
void dijkstra(int graph[vertex][vertex],int src) {
    int dist[vertex];
    bool Dset[vertex];
    dist[src]=0;
    for(int i=1;i<vertex;i++)  {  dist[i]=INT_MAX;  Dset[i]=false;  }
    for(int c=0;c<vertex;c++)  {
      int u=minimumDist(dist,Dset);
      Dset[u]=true; /*If the vertex with minimum distance found include it to Dset*/
      for(int v=0;v<vertex;v++)    {
        if(!Dset[v] && graph[u][v] && dist[u]!=INT_MAX &&dist[u]+graph[u][v]<dist[v])
        dist[v]=dist[u]+graph[u][v];
      }
    }
}
```

Spanning Tree

- Spanning Tree คือต้นไม้ที่ประกอบด้วยโหนดทุกโหนดของกราฟ โดยแต่ละคู่ของโหนดจะต้องมีเส้นเชื่อมเพียงเส้นเดียว นั่นคือไม่มี loop หรือ cycle

Spanning Tree

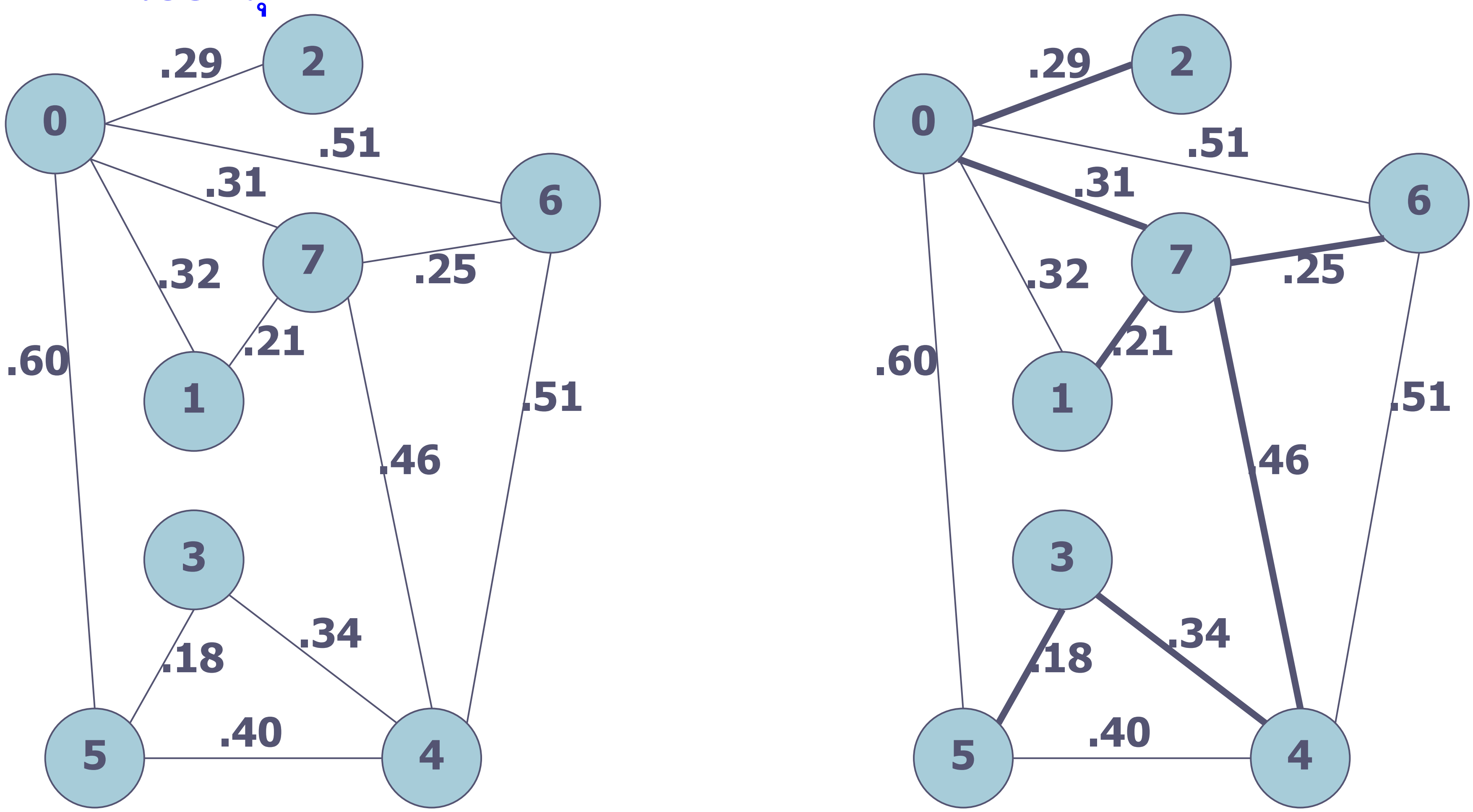
- สมมติสถานการณ์ให้กราฟแสดงเส้นทางการบินระหว่าง 7 เมือง แต่ด้วยเหตุผลทางธุรกิจทำให้ต้องปิดเส้นทางการบินไปให้มากที่สุดแต่ยังคงสามารถเชื่อมต่อถึงกันได้หมด



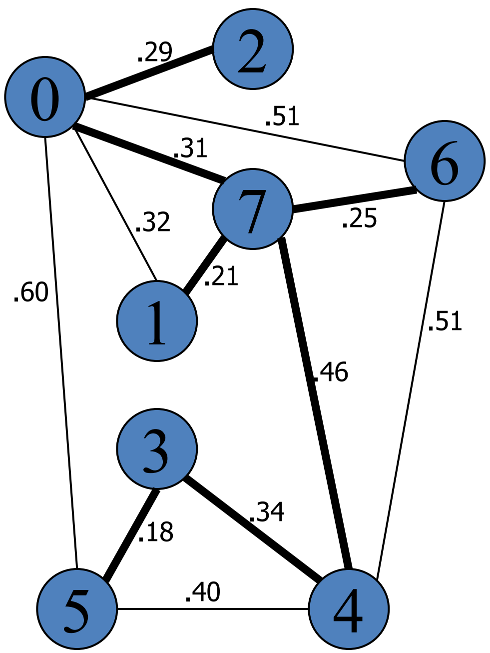
1. ห้ามมี 2 เส้น
2. ห้ามตัดเส้นออก
3. ห้ามมี cycle

Minimum Spanning Tree

MST หมายถึงเวทจ์กราฟและเป็นสแปนนิ่งทรี ที่มีค่าน้ำหนักรวมกันแล้วมีค่าน้อยที่สุด



Minimum Spanning Tree



0	0	1	2	3	4	5	6	7
0	*	.32	.29	*	*	.60	.51	.31
1	.32	*	*	*	*	*	*	.21
2	.29	*	*	*	*	*	*	*
3	*	*	*	*	.34	.18	*	*
4	*	*	*	.34	*	.40	.51	.46
5	.60	*	*	.18	.40	*	*	*
6	.51	*	*	*	.51	*	*	.25
7	.31	.21	*	*	.46	*	.25	*

Minimum Spanning Tree

วิธีการในการหา Minimum Spanning Tree ที่นิยมใช้มี 2 วิธี

1. Kruskal's Algorithm
2. Prim's Algorithm

1. Kruskal’s Algorithm

Kruskal’s Algorithm ค้นพบโดย Joseph Kruskal ในปี 1956 โดยมีหลักการดังต่อไปนี้

(1) เลือก Edge ใน Graph ที่มี Weight ต่ำสุด เป็น edge เริ่มต้น

(2) เพิ่ม Edge ที่มี Weight ต่ำสุดที่เหลืออยู่ ที่จะไม่ทำให้เกิด Simple Circuit กับ Edge ที่เลือกไว้แล้ว หยุดเมื่อได้ n-1 Edge

Kruskal’s Algorithm

เรียงลำดับ weight จากน้อยไปมาก

3-5 = .18 ✓

1-7 = .21 ✓

6-7 = .25 ✓

0-2 = .29 ✓

0-7 = .31 ✓

0-1 = .32 เกิดวงจร

4.3 = .34 ✓

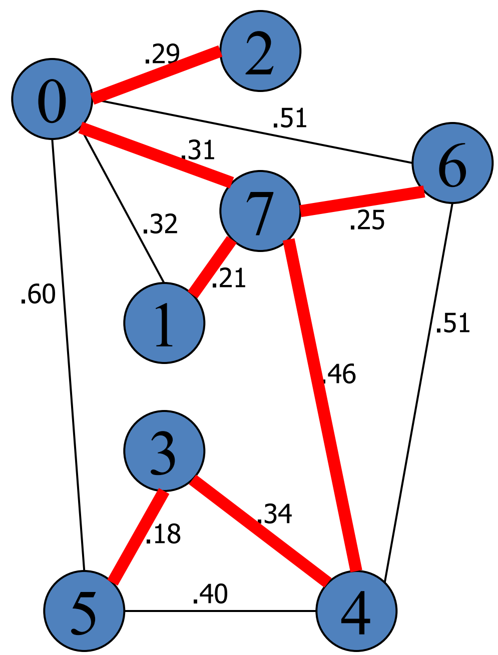
4-5 = .40 เกิดวงจร

4-7 = .46 ✓

0-6 = .51

4-6 = .51

0-5 = .60



2. Prim's algorithm

หลักการของ Prim's Algorithm คล้าย Dijkstra's Algorithm แต่ใช้การเริ่มต้นจากจุดเดียว แล้วทำให้โหนดเข้ามาใน set ทุกโหนด เริ่มต้นจากเวอร์เท็กซ์ที่กำหนดแล้วหาเวอร์เท็กซ์ข้างเคียง เรียงตามค่าน้ำหนักของเอดจ์ มีขั้นตอนดังนี้

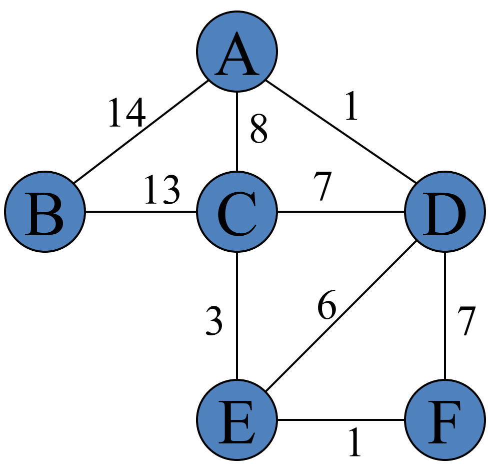
(1) เลือก 1 จุด

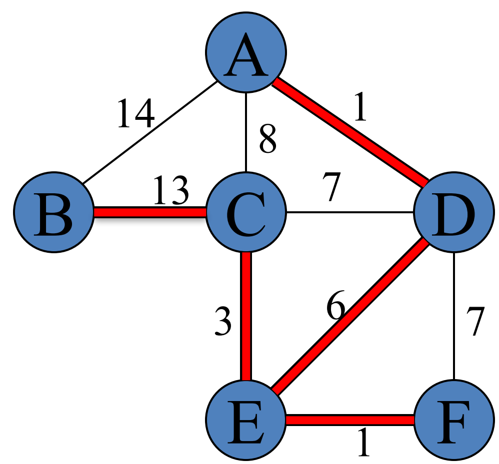
(2) เลือก edge สั้นสุดที่ต่อกับที่ได้เลือกไว้

(3) รวม edge นี้ถ้าไม่เกิดวงจร

Prim's algorithm

- เลือก 1 จุด
- เลือก edge สั้นสุดที่ต่อกับที่ได้เลือกไว้
- รวม edge นี้ถ้าไม่เกิดวงจร



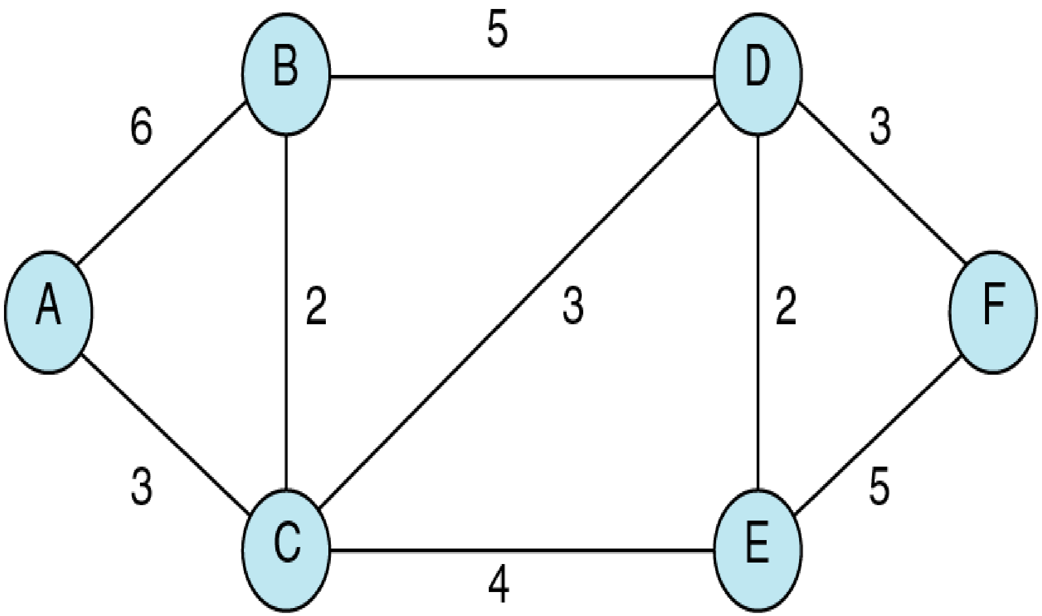


Prim

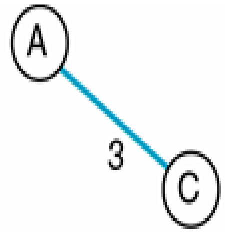
```
Prim ( G = (V,E) ) {
  for each vertex v in V {
    d[v] = ∞; p[v] = null;
    inMST[v] = false;
  }
  select an arbitrary vertex v; d[v] = 0;
  H = a min heap of all vertices ordered by d[]
  while (H != ∅){
    u = H.removeMin();
    inMST[u] = true;
    for each v ∈ adj(u){
      if(!inMST[v] && w(u,v)<d[v])) {
        d[v] = w[u,v]; H.decreaseKey(v);
        p[v] = u
      }
    }
  }
  return p;
};
```

ตัวอย่าง

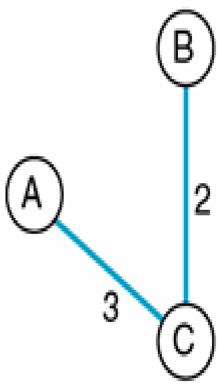
จงหา Minimum Spanning Tree



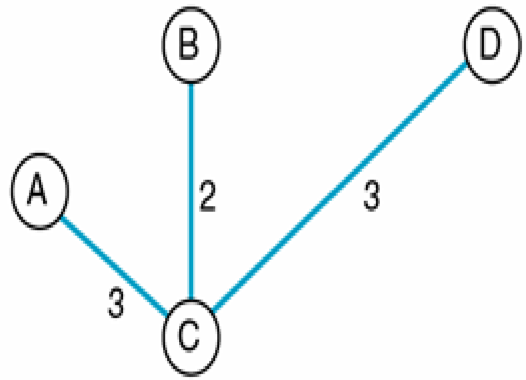
(a) Insert first vertex



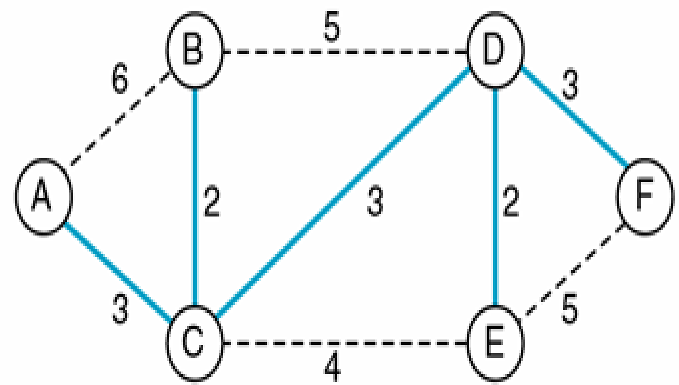
(b) Insert edge AC



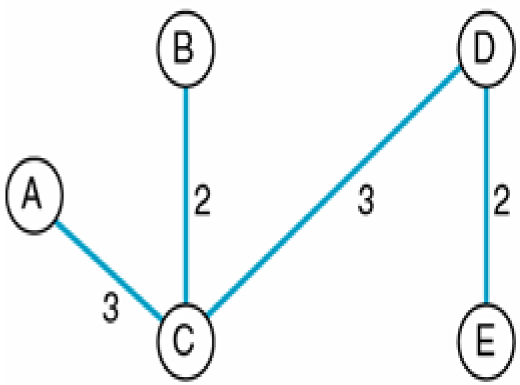
(c) Insert edge BC



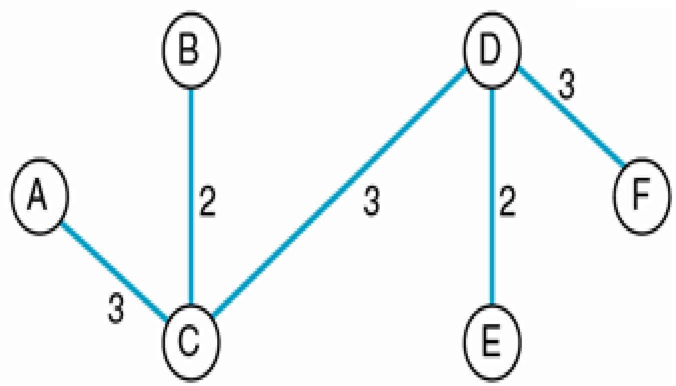
(d) Insert edge CD



(g) The final tree in the graph



(e) Insert edge DE

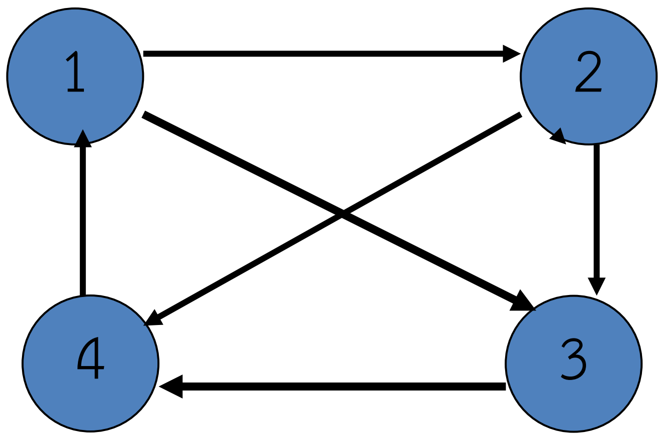


(f) Insert edge DF

Total weight = 13

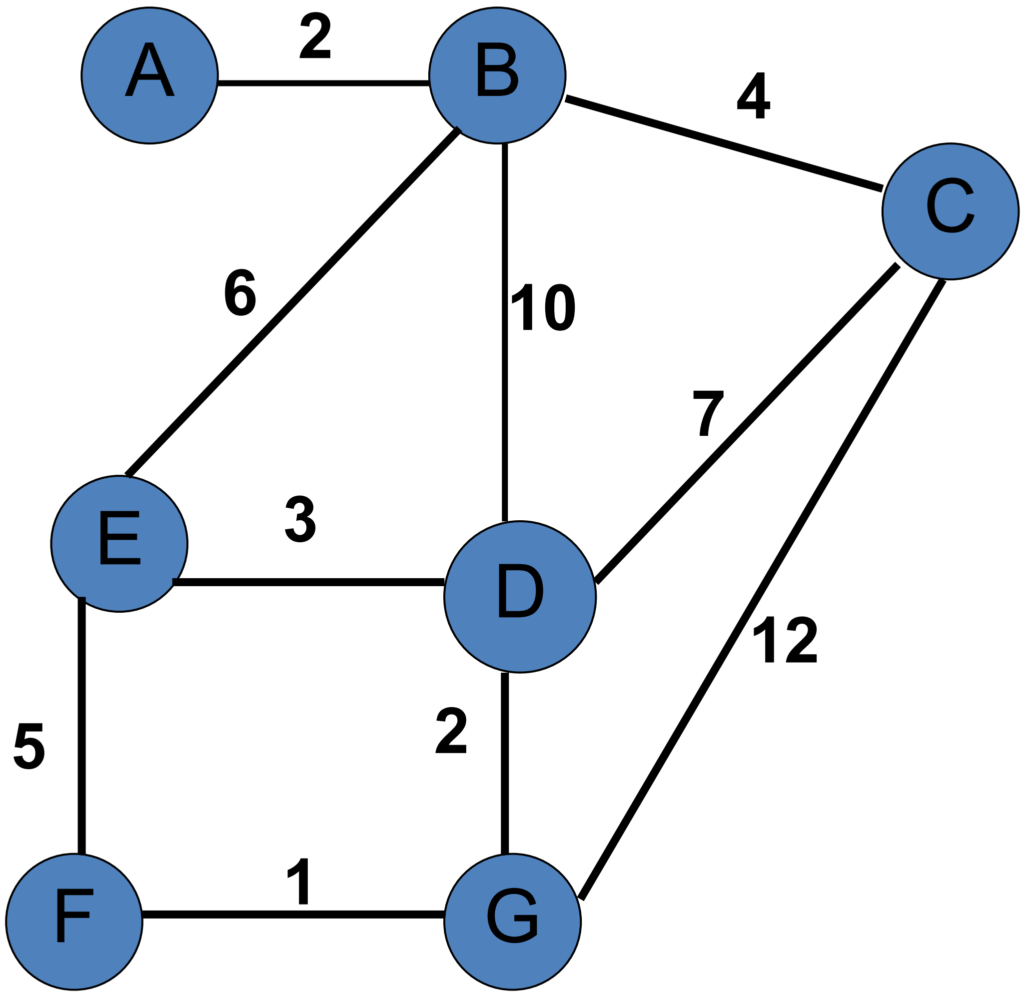
แบบฝึกหัด ลองทำดู

1. จากภาพต่อไปนี้ จงการแทนกราฟด้วยอะเรย์สองมิติ



	1	2	3	4
1	0	1	1	0
2	0	0	1	1
3	0	0	0	1
4	1	0	0	0

2. จงหาระยะทางโดยใช้วิธีการของ Dijkstra
โดยเริ่มที่จุด B ถึงจุด G



3. จงหา MST โดยใช้วิธีการของ Kruskal’s และ Prim เริ่มที่จุด B