NCKU Programming Contest Training Course
Shortest Path
2013/04/17

Pinchieh Huang (free999)

Pinchieh.huang@gmail.com

http://myweb.ncku.edu.tw/~p76014143/20130417_SP.rar

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Outline

- Shortest Path Algorithm
- Transitive Closure
- Bellman Ford Algorithm
- Shortest Path Fast Algorithm (SPFA)
- Dijkstra Algorithm

Single Source

- Single source shortest path problem
  - Problem: given a weighted directed graph G, find the minimum-weight path from a given source vertex s to another vertex v
  - "Shortest-path" = minimum weight
  - Weight of path is sum of edges
  - E.g., a road map: what is the shortest path from Chapel Hill to Charlottesville?

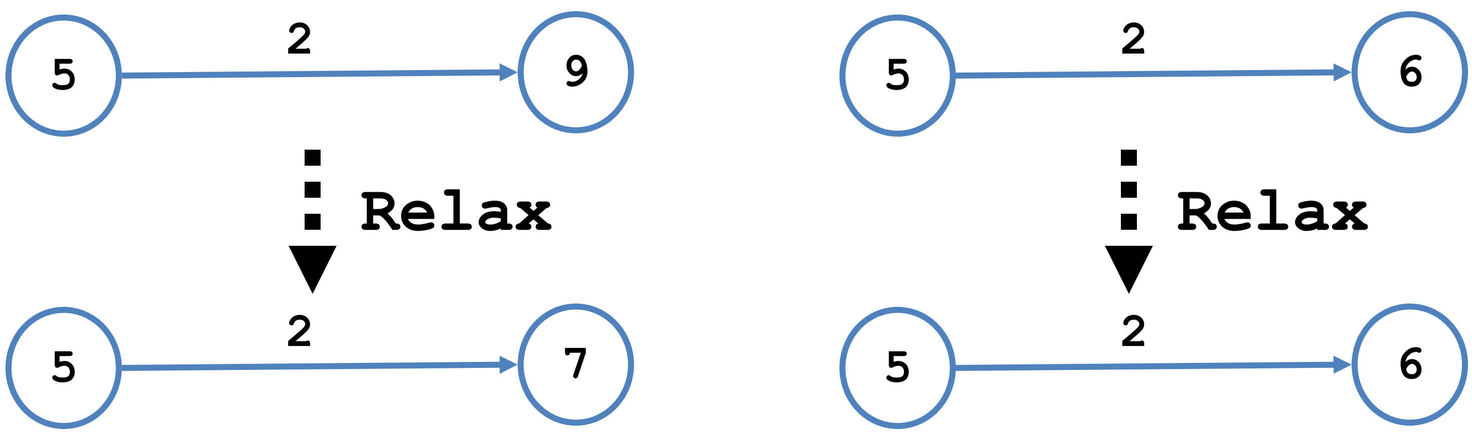
Single Source

- ***Optimal substructure***: the shortest path consists of shortest subpaths
- Let δ(u,v) be the weight of the shortest path from u to v. Shortest paths satisfy the *triangle inequality*
  - *triangle inequality :* $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$
- In graphs with negative weight cycles, some shortest paths will not exist

Single Source

- Key technique: ***relaxation***
  - 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;
}
```



Outline

Shortest Path Algorithm

↓

Transitive Closure

↓

Bellman Ford Algorithm

↓

Shortest Path Fast Algorithm (SPFA)

↓

Dijkstra Algorithm

Transitive Closure

- Transitive Closure
  - A relation *R* on a set *X* is transitive if, for all *x*, *y*, *z* in *X*, whenever *x R y* and *y R z* then *x R z*.
  - Examples of transitive relations include the equality relation on any set, the "less than or equal" relation on any linearly ordered set, and the relation "*x* was born before *y*" on the set of all people.
  - Example:
    - $X > Y$, $Y > Z$ **=> $X > Z$**
    - $X < Y$, $Y < Z$ **=> $X < Z$**

Transitive Closure

- The strategy for deriving a transitive closure matrix will be based on this simple idea
  - Reachability
    - Start with an pair (i j)
    - Check if can from i to j
    - If an dedicated path (i, j) return true
    - Else enumerate a vertex k, check i->k and k->j
  - This will require the use of **3 nested for-loops**, one for the starting vertex of a path, one for the destination vertex of a path, and one to see if a path already exists from start to this point and from this point to destination

Transitive Closure

- Sample Code

```
for(int k=0;k<n;k++)
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            reach[i][j] = reach[i][j] | (reach[i][k] && reach[k][j]);
```

Floyd-Warshall

- Sample Code

```
for(int k=0;k<n;k++)
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            if ( d[i][k] + d[k][j] < d[i][j] )
                d[i][j] = d[i][k] + d[k][j]
```

example

- Anti-Floyd
  - Minimized graph by reducing duplicate path
- UVA-10987

Input:

2
3
100
200 100
3
100
300 100

Output:

Case #1:
2
1 2 100
2 3 100

Case #2:
Need better measurements.

Outline

Shortest Path Algorithm

Transitive Closure

Bellman Ford Algorithm

Shortest Path Fast Algorithm (SPFA)

Dijkstra Algorithm

Bellman Ford

```
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));
```

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

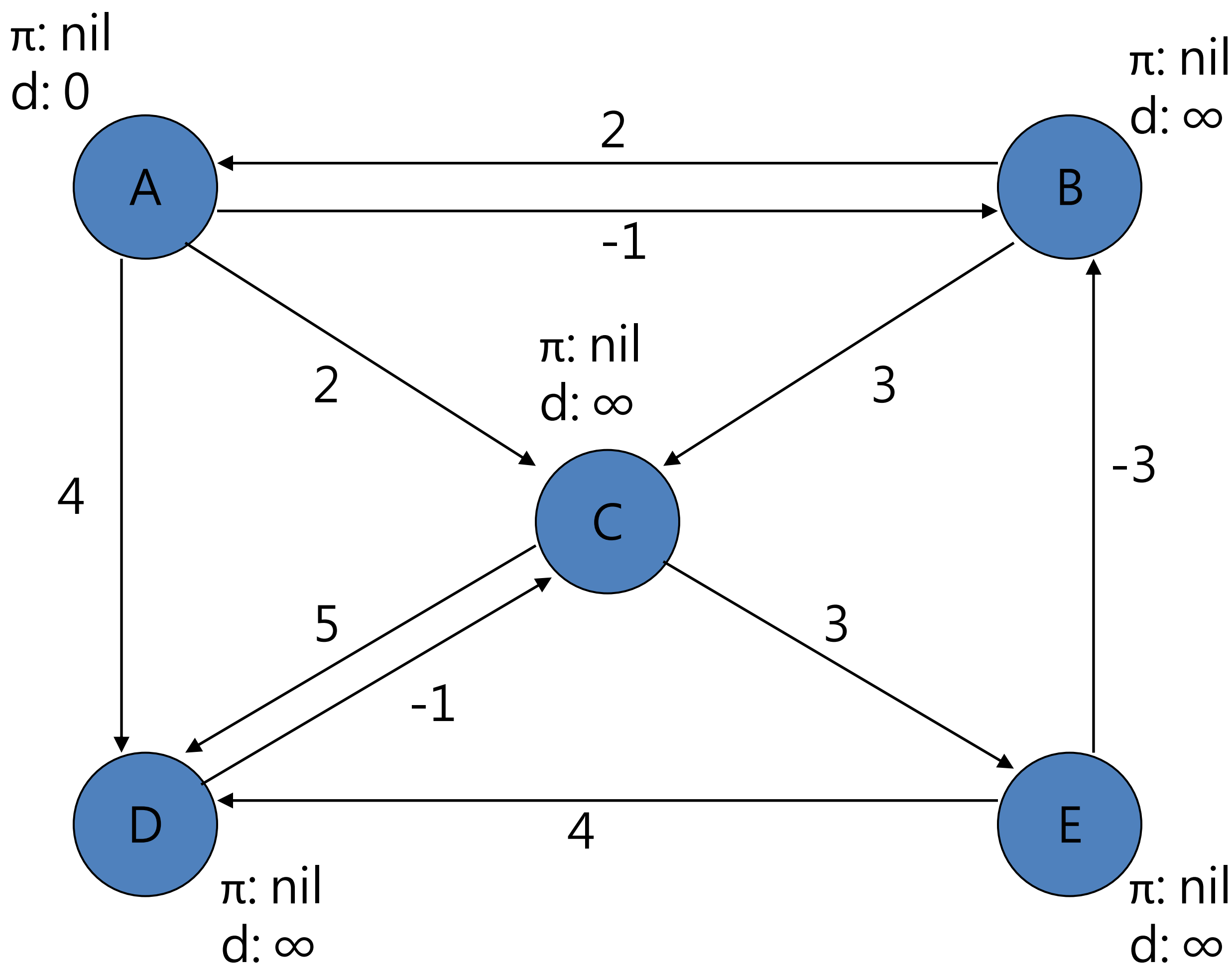
Relaxation: Make |V|-1 passes, relaxing each edge

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

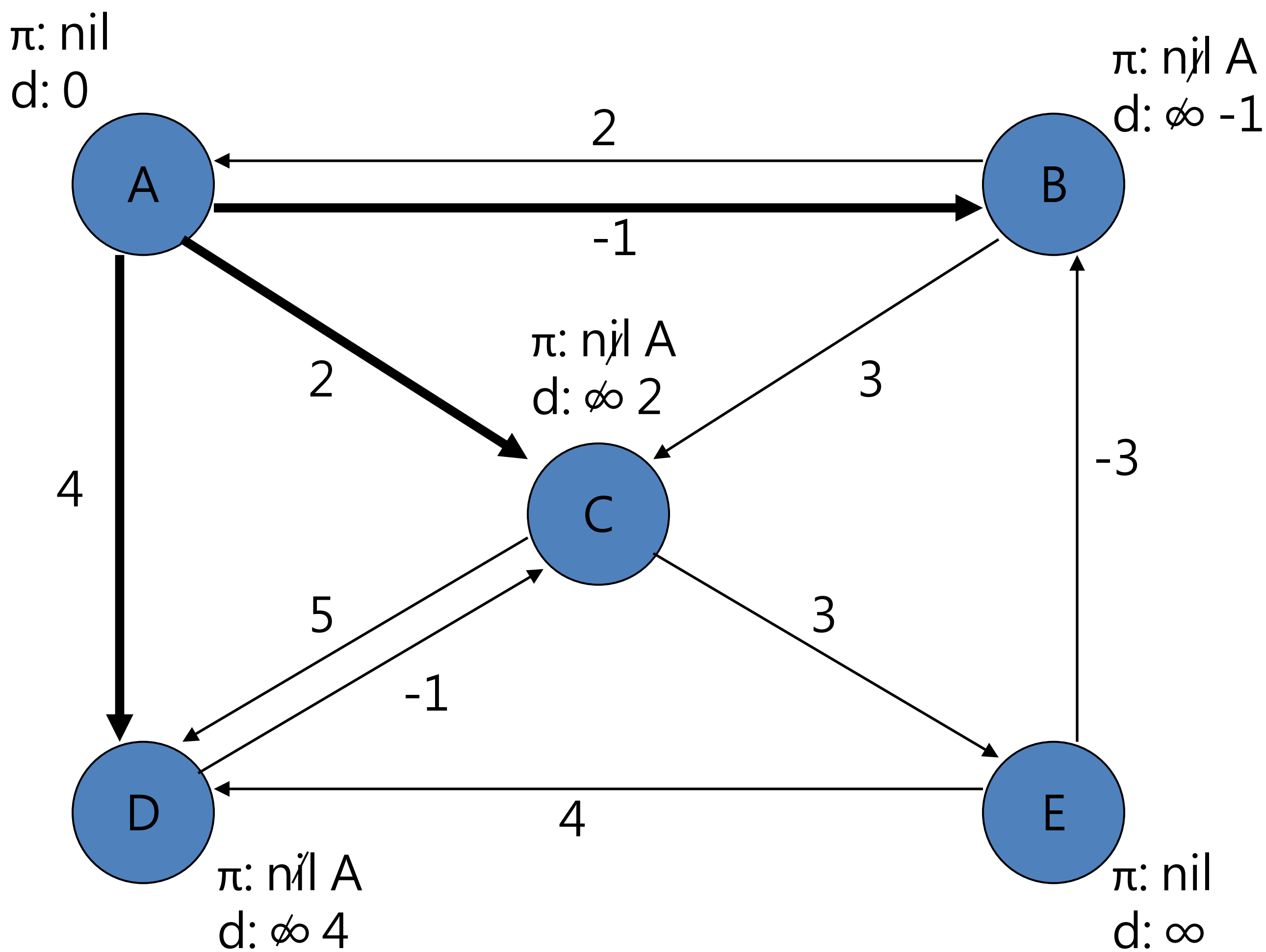
Bellman Ford

- Running time: O(VE)
  - Not so good for large dense graphs
  - But a very practical algorithm in many ways

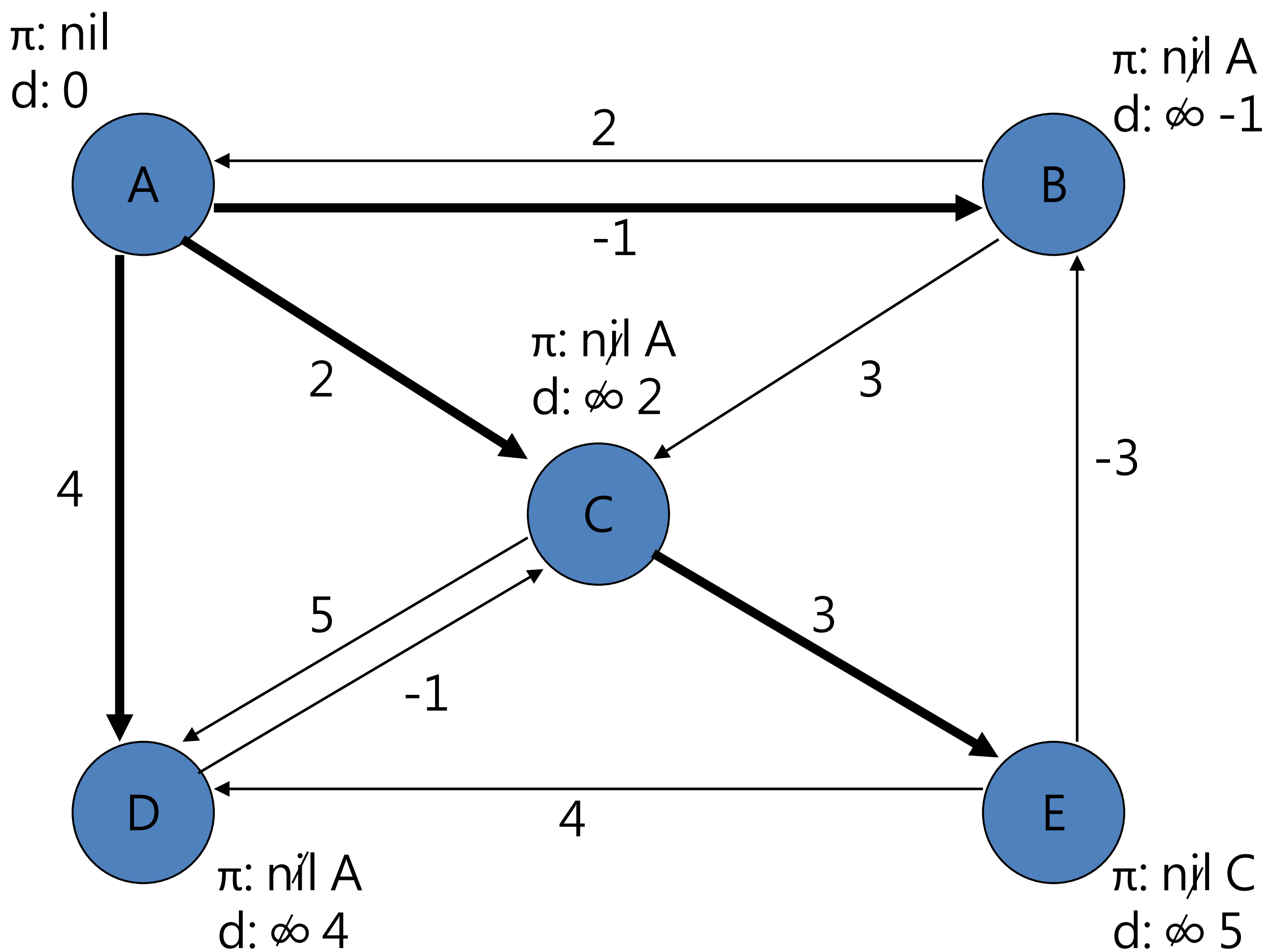
Bellman Ford



Bellman Ford



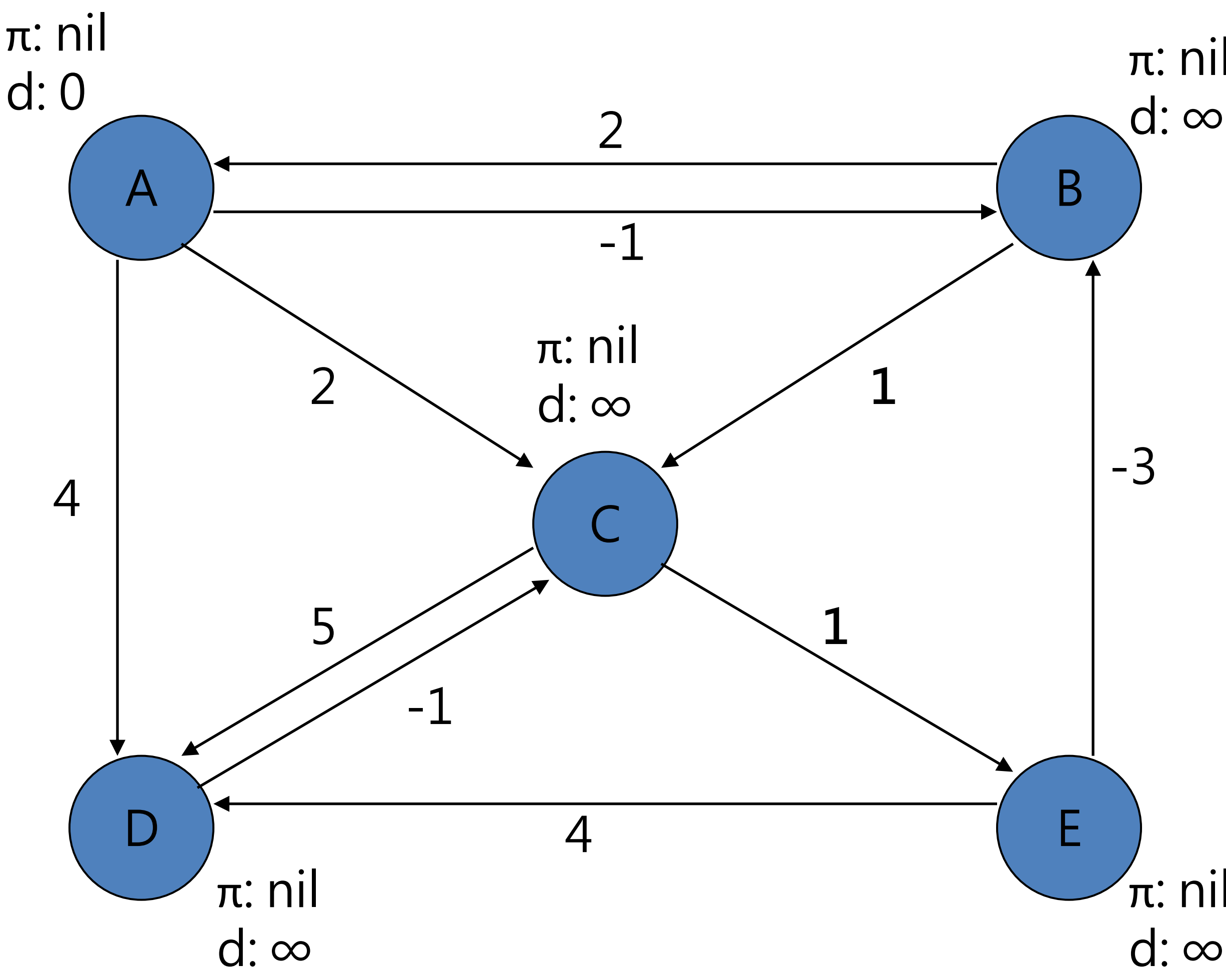
Bellman Ford



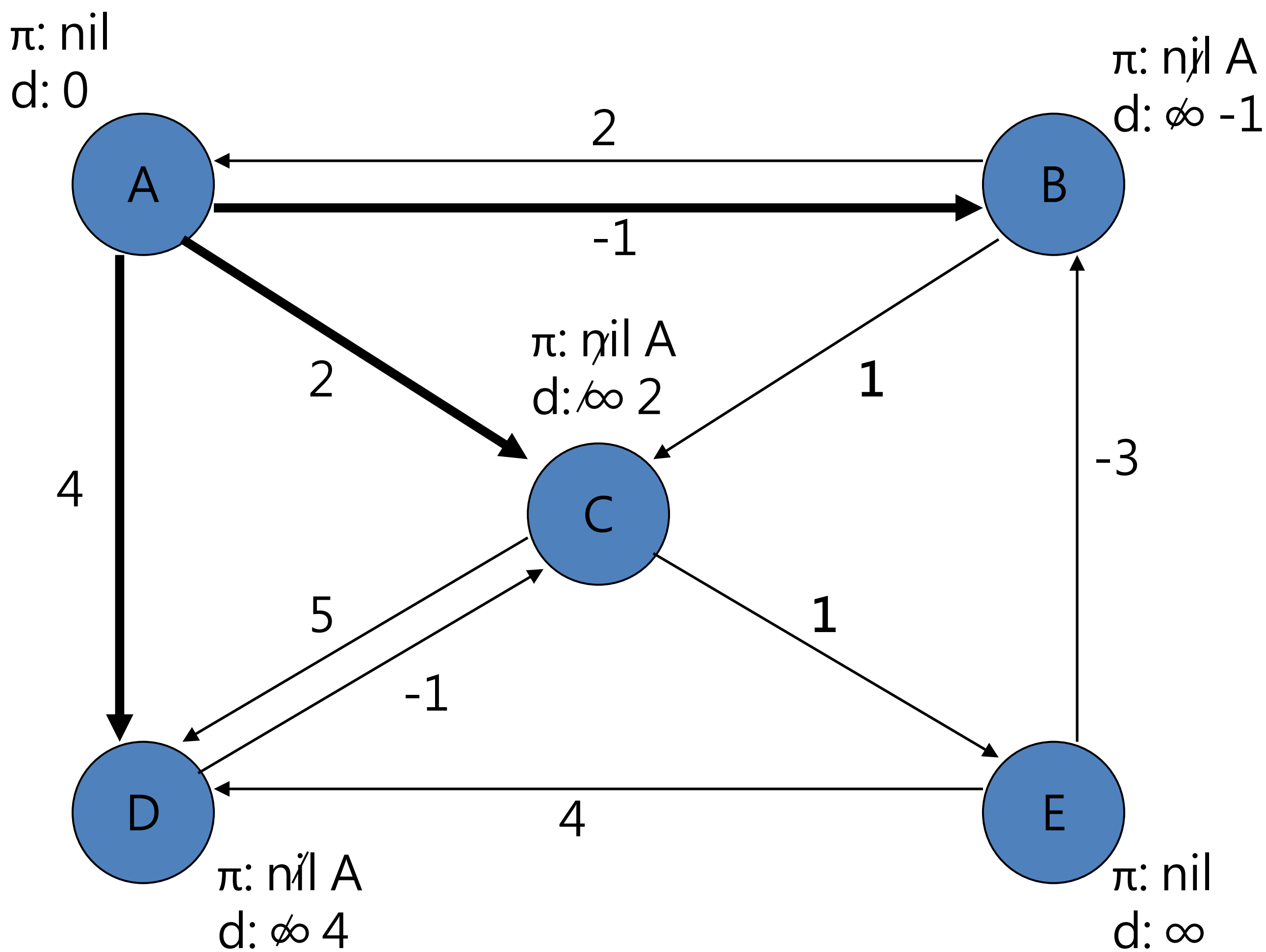
Bellman Ford

- So…consider the negative cycle…

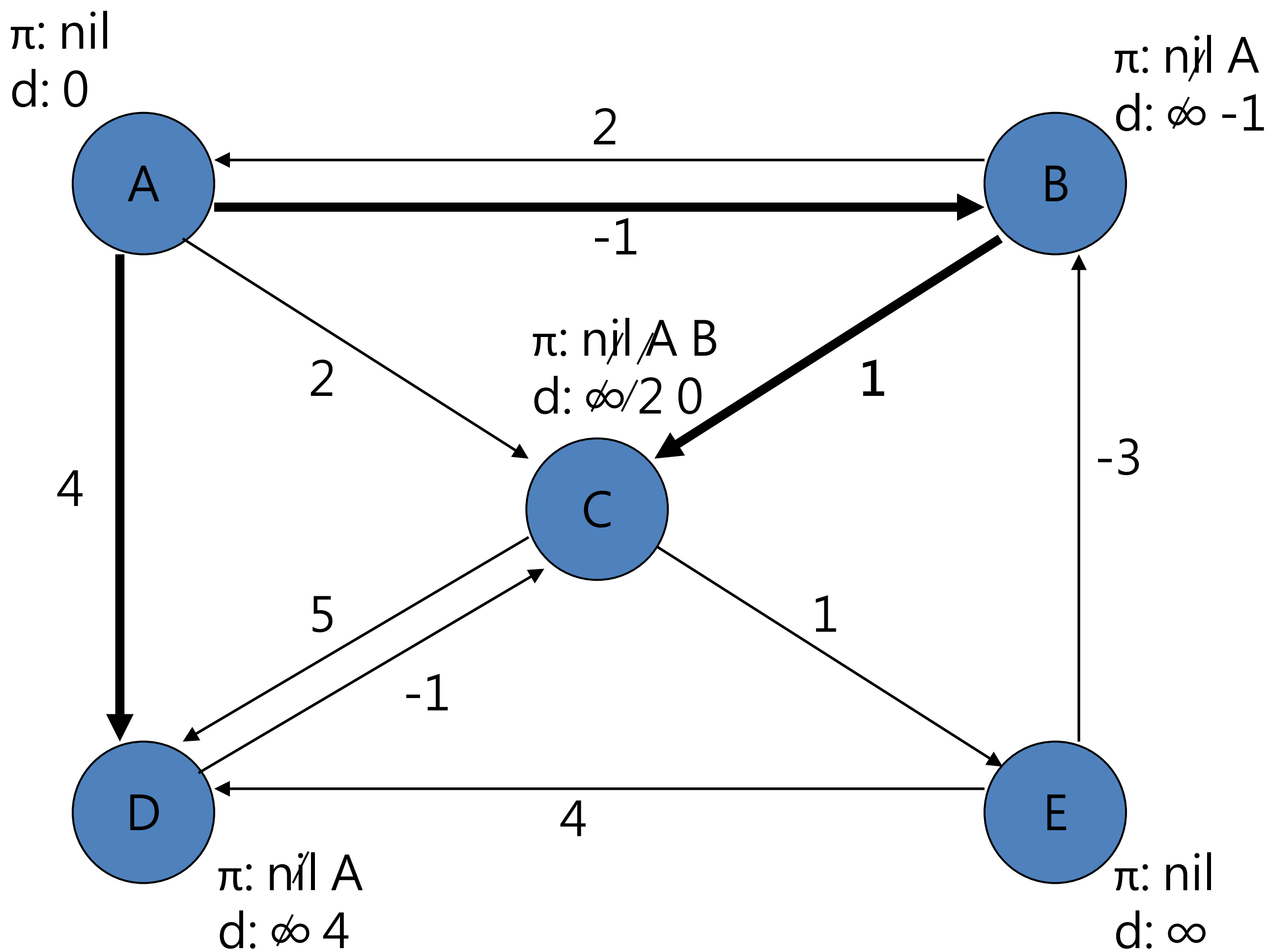
Bellman Ford



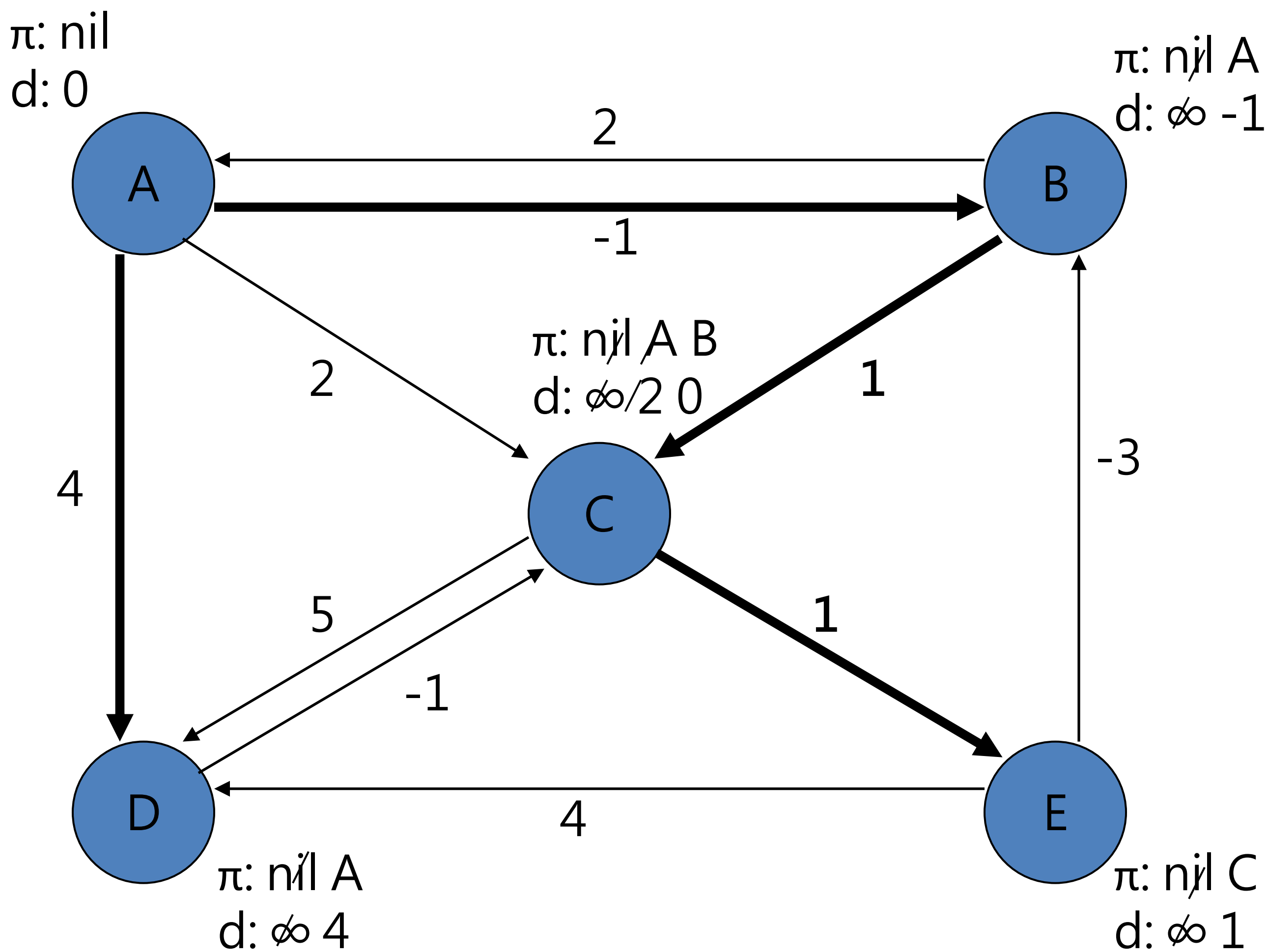
Bellman Ford



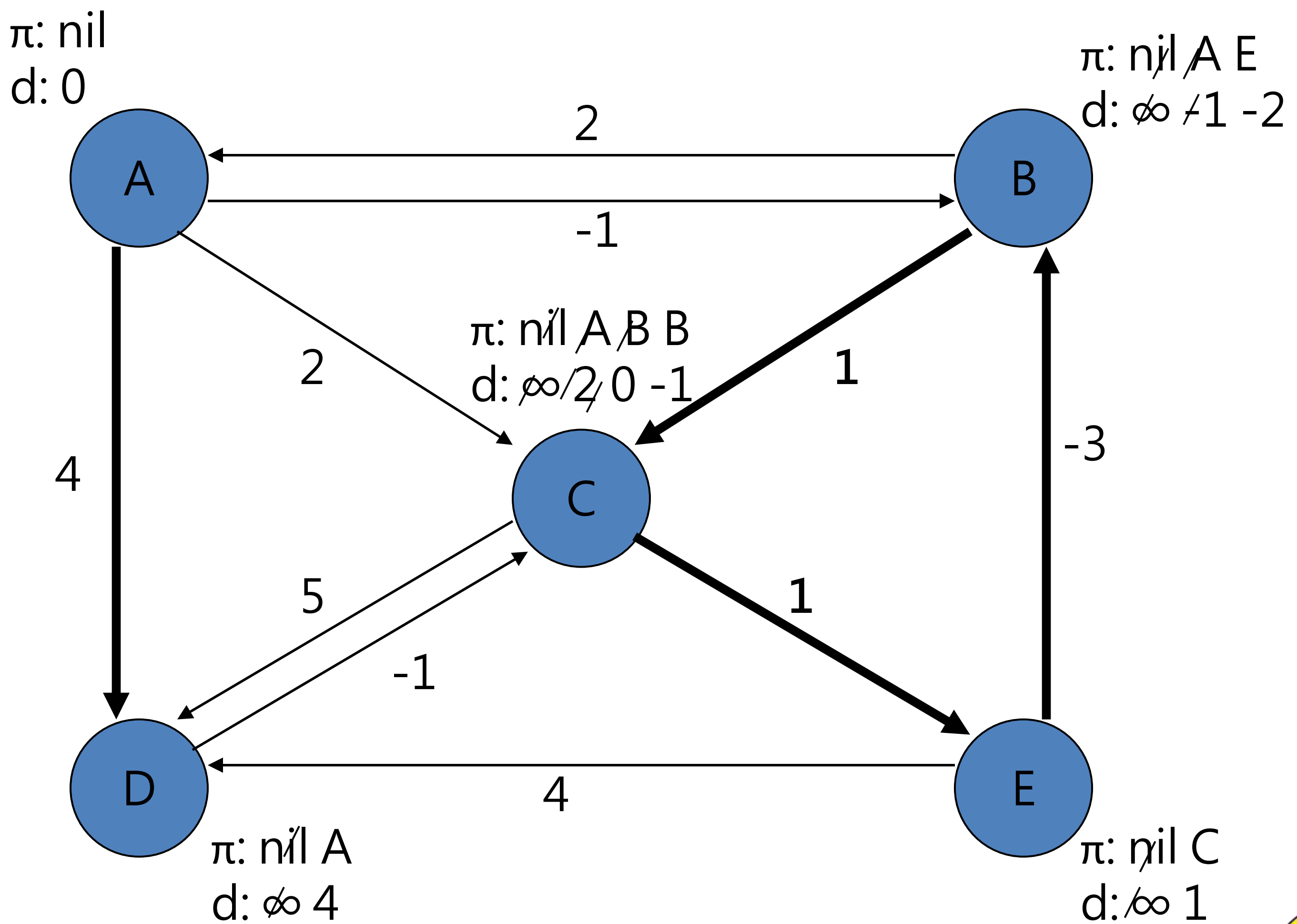
Bellman Ford



Bellman Ford



Bellman Ford



Bellman Ford

```
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: have we converged yet? Ie, ∃ negative cycle?

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

Outline

- Shortest Path Algorithm
- Transitive Closure
- Bellman Ford Algorithm
- Shortest Path Fast Algorithm (SPFA)
- Dijkstra Algorithm

SPFA

- Shortest path fast algorithm
  - A modified bellman ford algorithm
  - A modified bfs search algorithm

```
struct EDGE
{
    int t,w;
}tmp;
vector<EDGE>edge[MAXN];
```

```
memset(count,0,sizeof(count));       ←判斷負環
memset(inqueue,0,sizeof(inqueue));   ←是否在queue裡面
for(int i=0;i<MAXN;++i)              ←起點到各點的距離
    dis[i]=INF;
v.push(start);
inqueue[start]=true;
count[start]=1;
dis[start]=0;
```


made by electron & free999

SPFA

- Shortest path fast algorithm
  - A modified bellman ford algorithm
  - A modified bfs search algorithm

1. 令要 BFS 的點為nowv
2. 對 nowv 相鄰的點做 relax
3. 如果相鄰的點不在 queue 裡面，則丟進 queue，count 記得+1

example

- Uva 10801
- Uva 10841

Outline

Shortest Path Algorithm

Transitive Closure

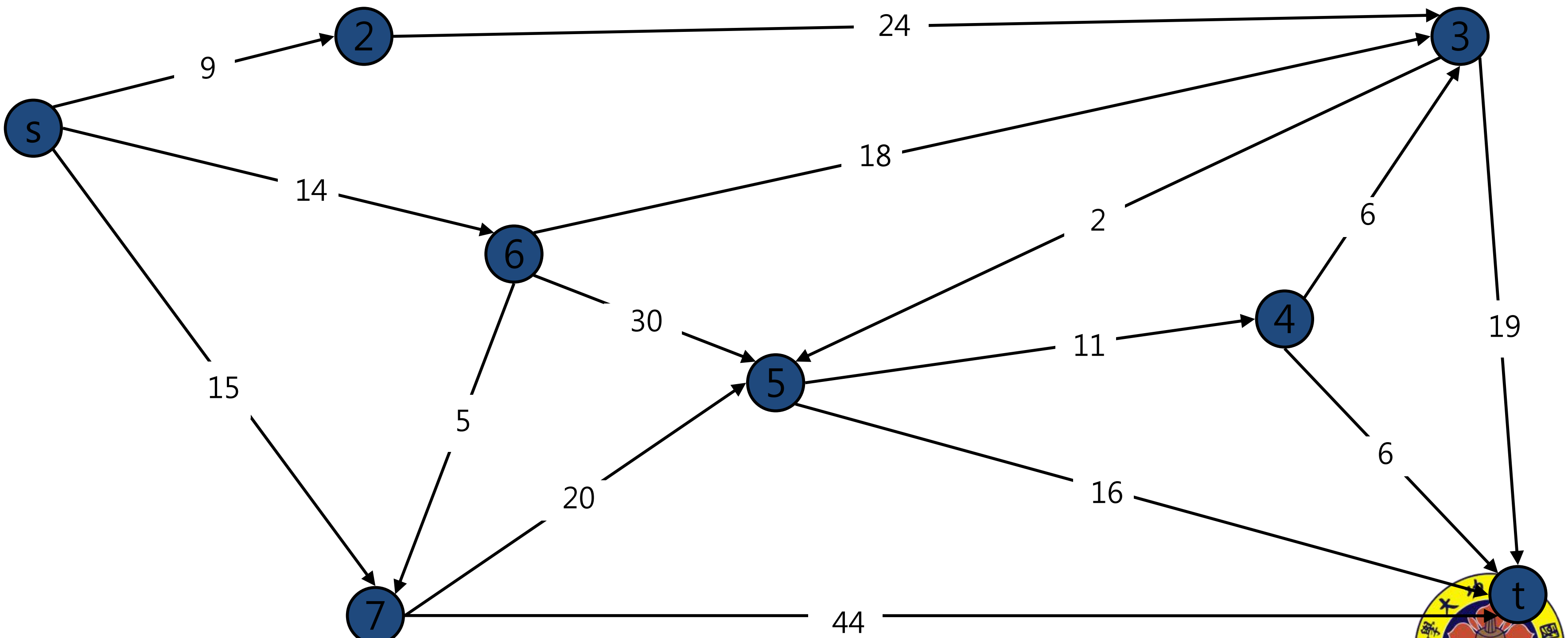
Bellman Ford Algorithm

Shortest Path Fast Algorithm (SPFA)

Dijkstra Algorithm

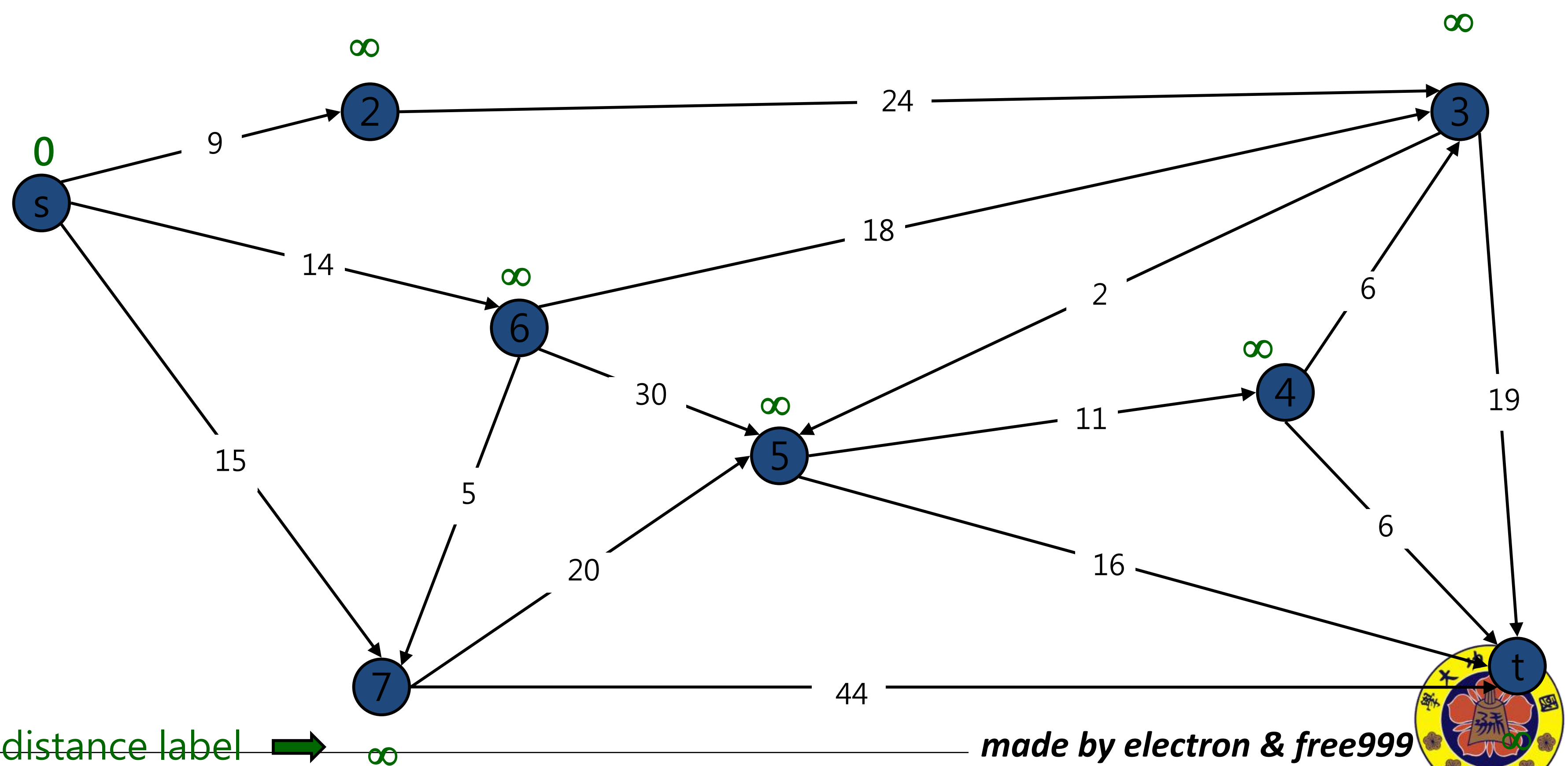
Dijkstra

- Find shortest path from s to t.



S = { }

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

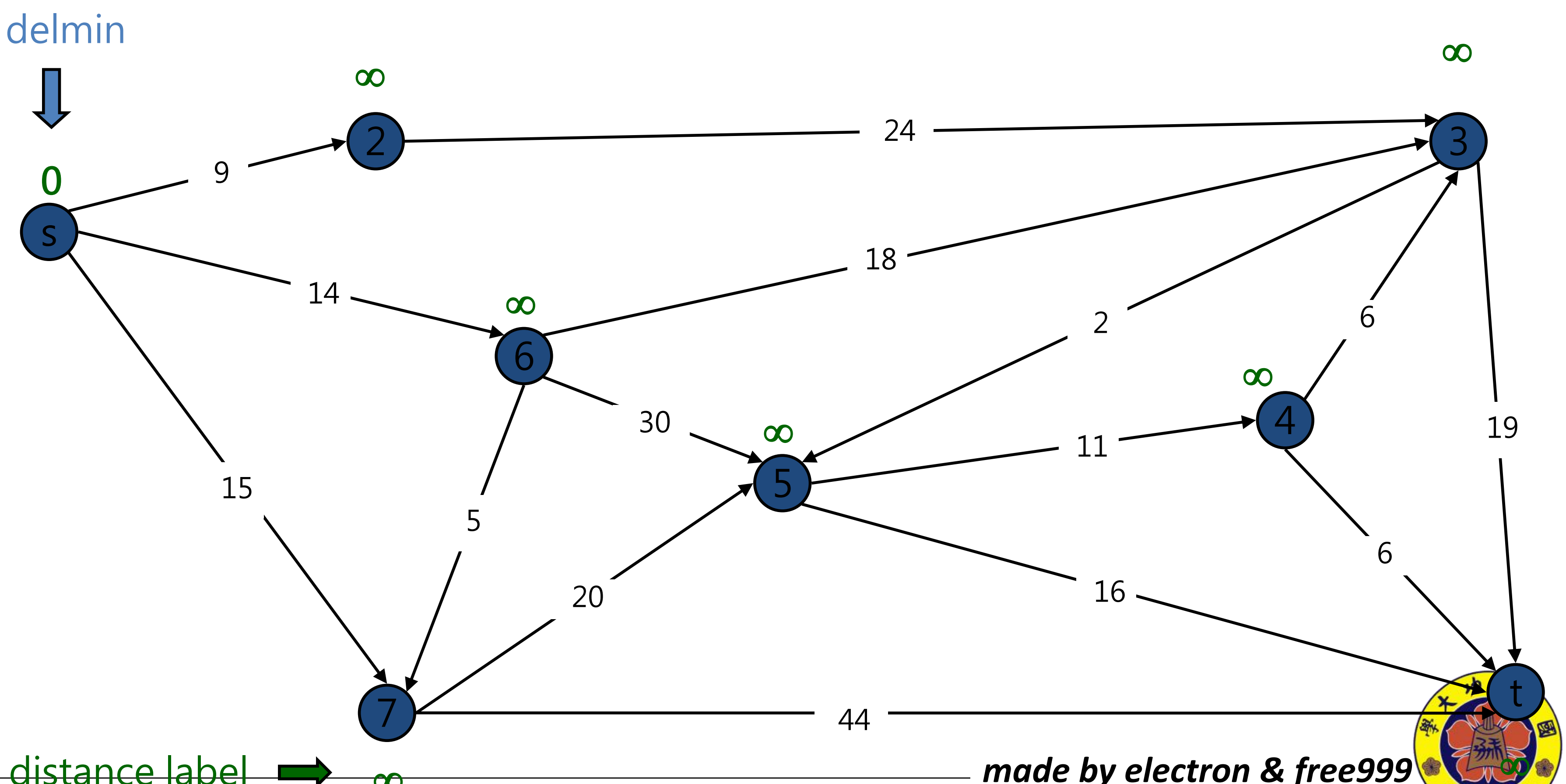


distance label ∞

made by electron & free999

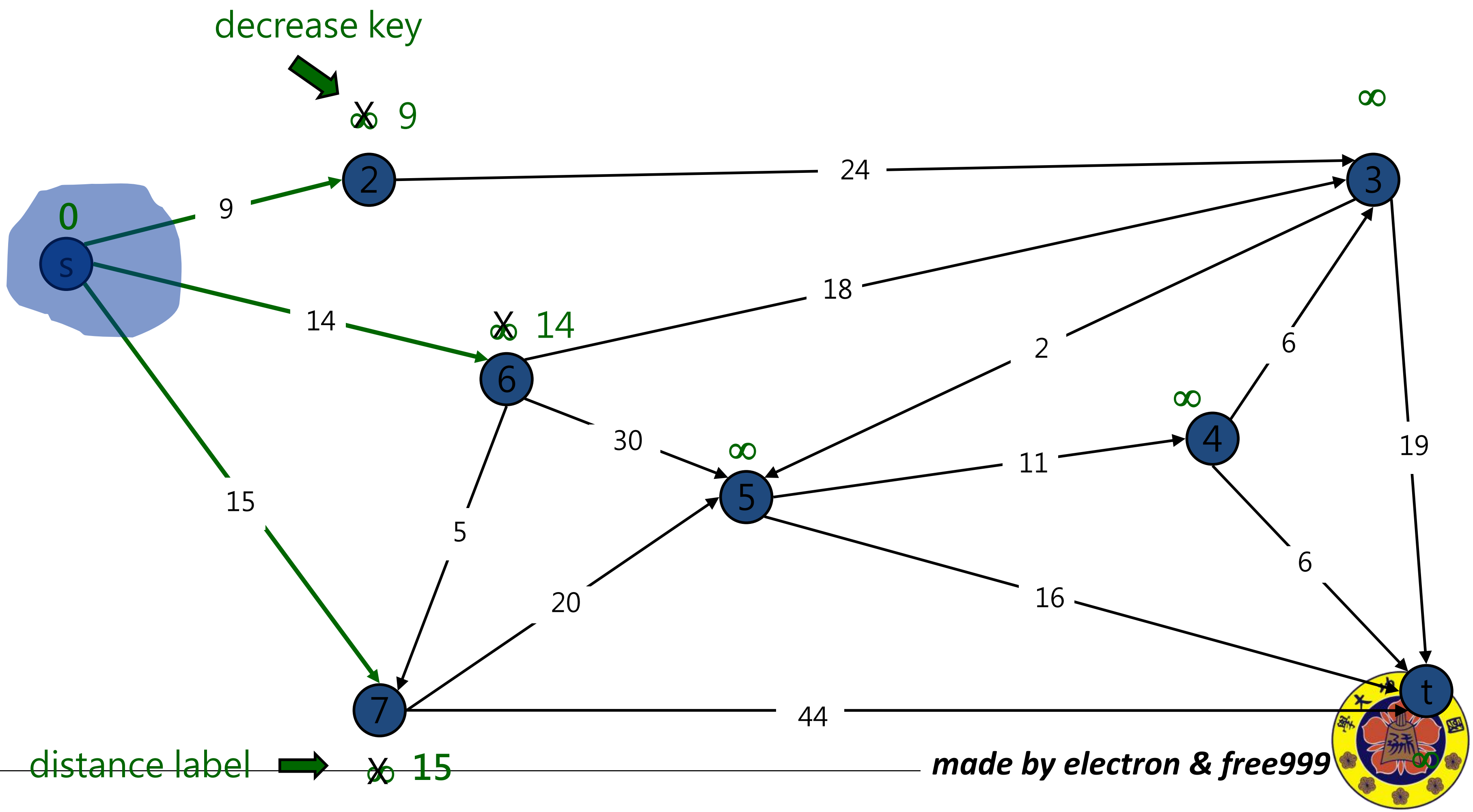
S = { }

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



S = { s }

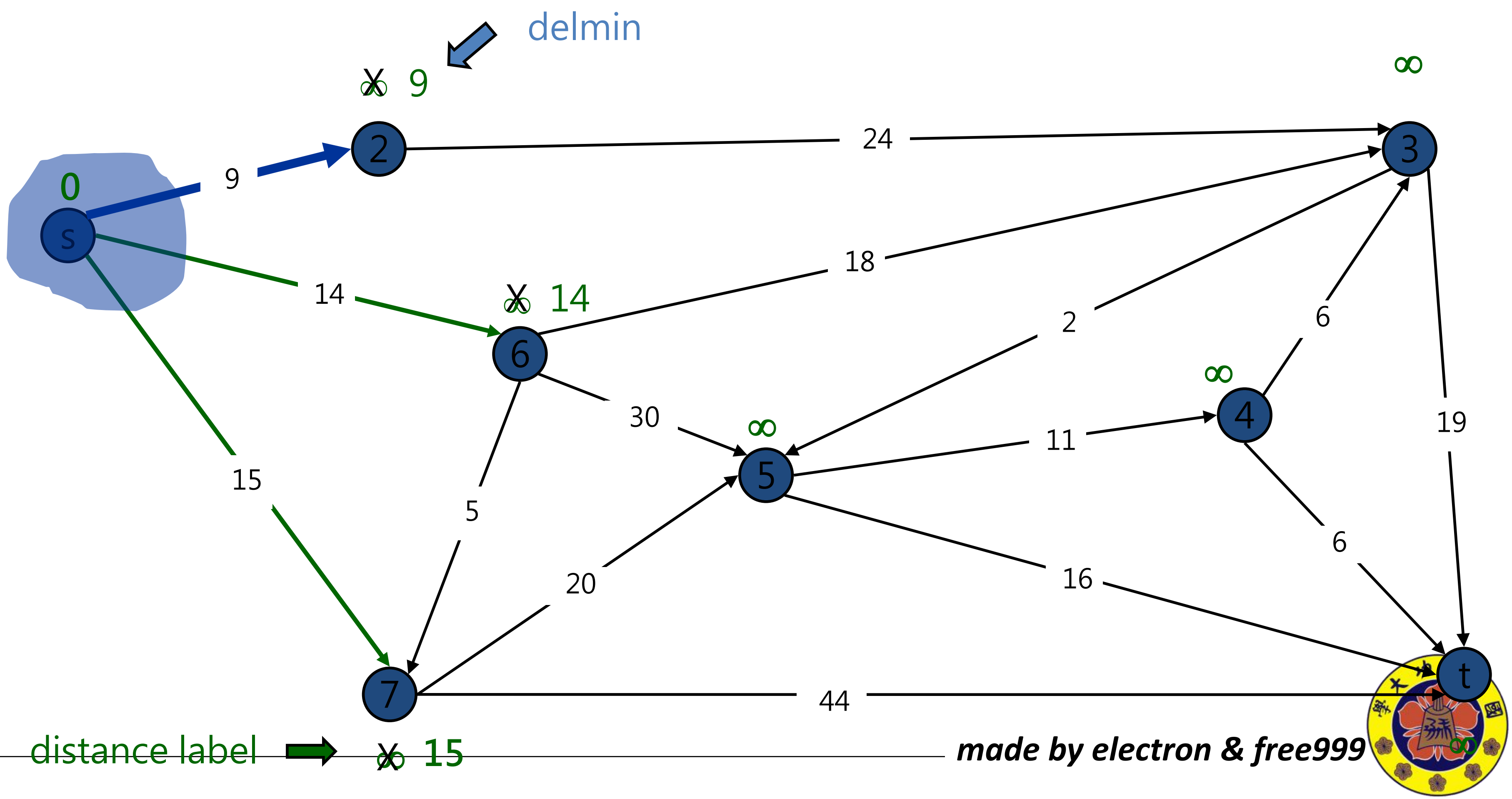
PQ = { 2, 3, 4, 5, 6, 7, t }



made by electron & free999

S = { s }

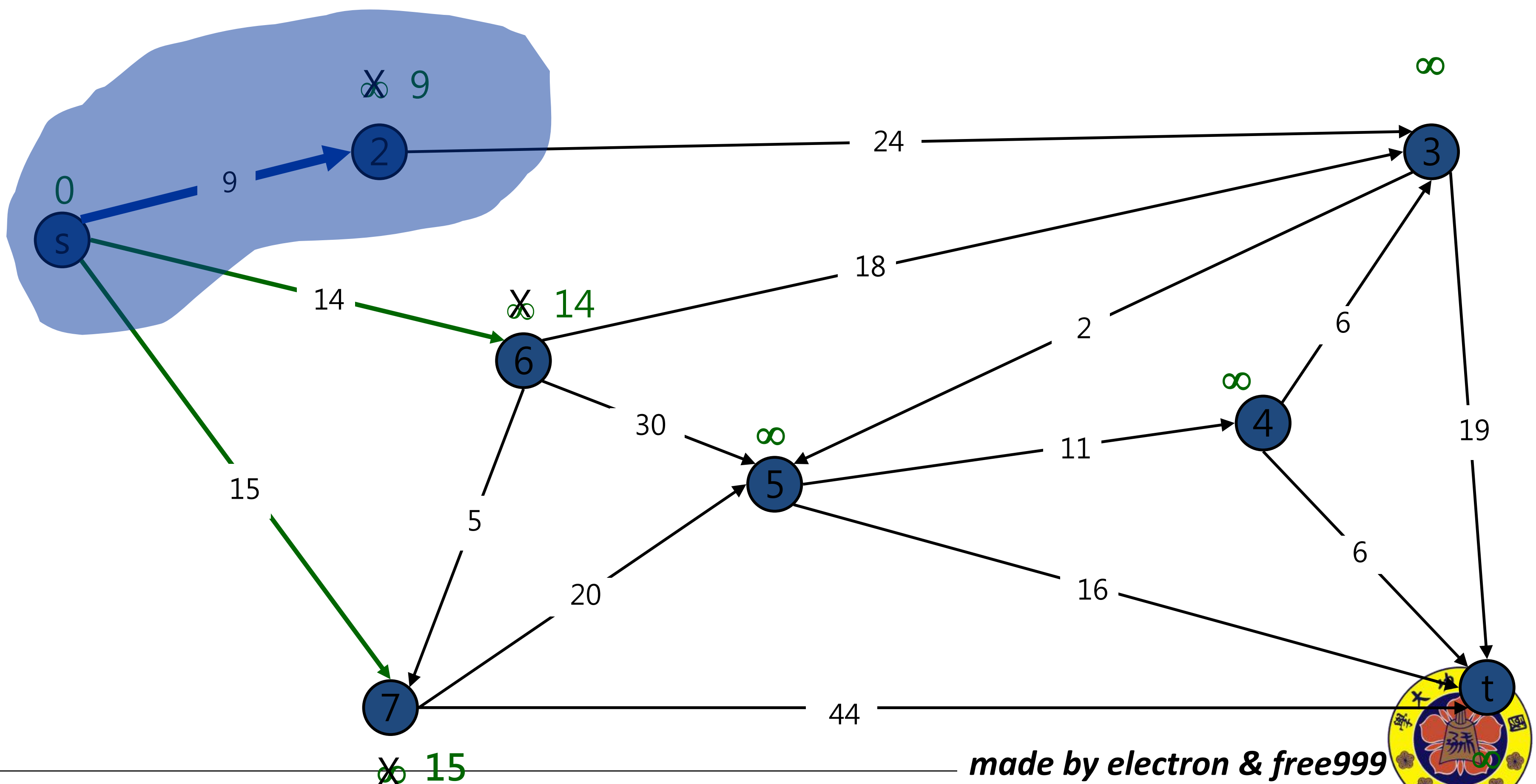
PQ = { 2, 3, 4, 5, 6, 7, t }



made by electron & free999

S = { s, 2 }

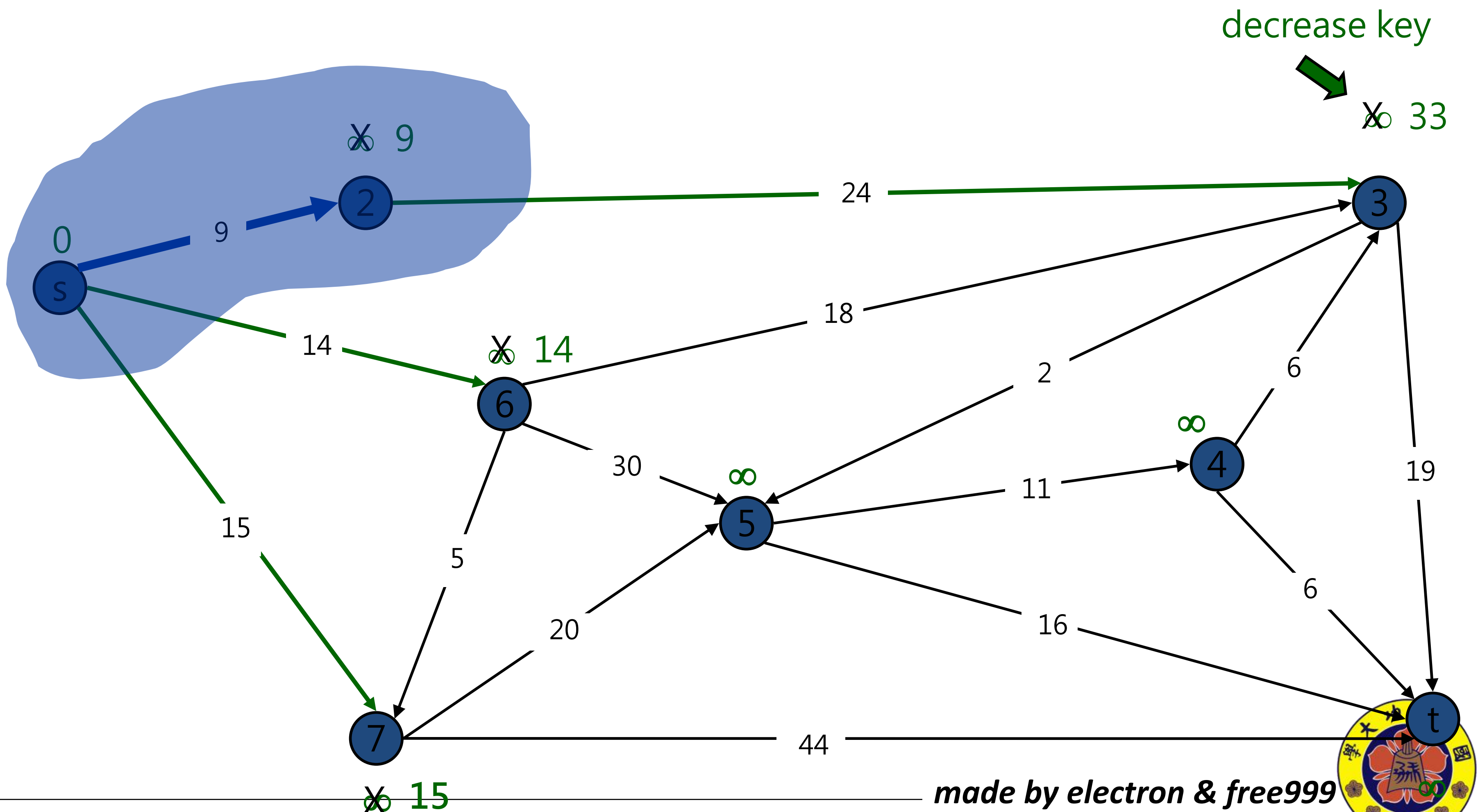
PQ = { 3, 4, 5, 6, 7, t }



made by electron & free999

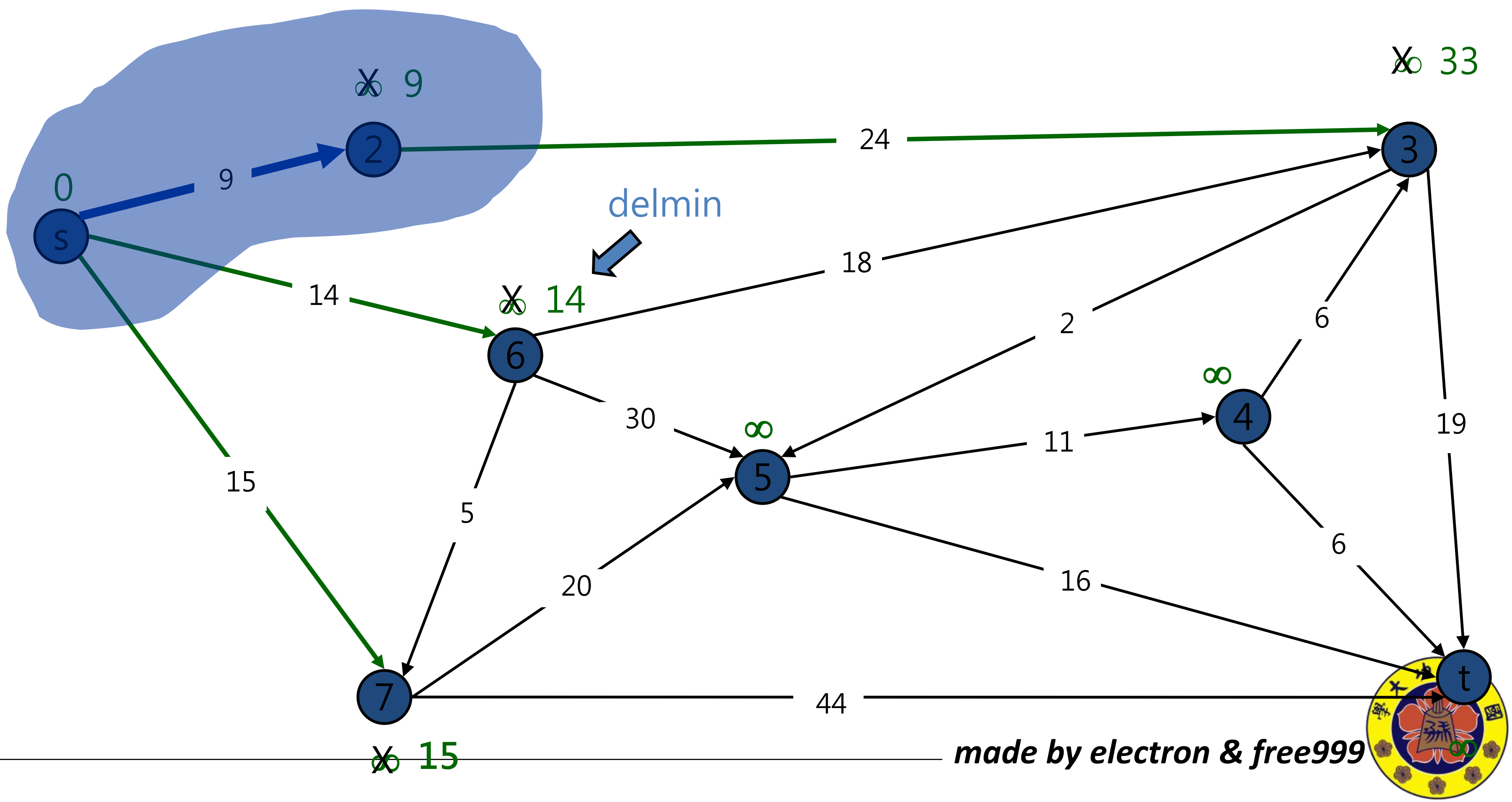
S = { s, 2 }

PQ = { 3, 4, 5, 6, 7, t }

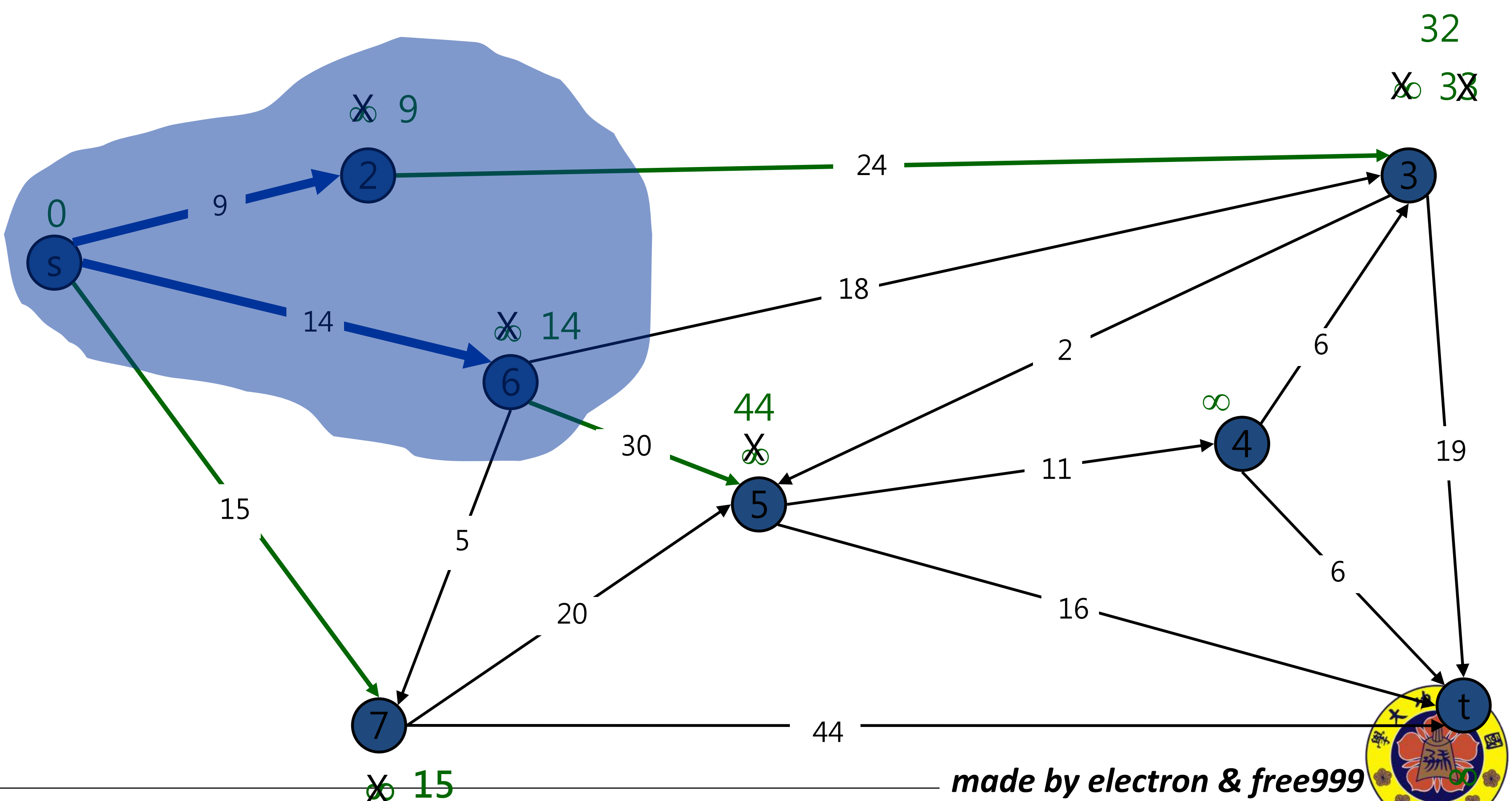


S = { s, 2 }

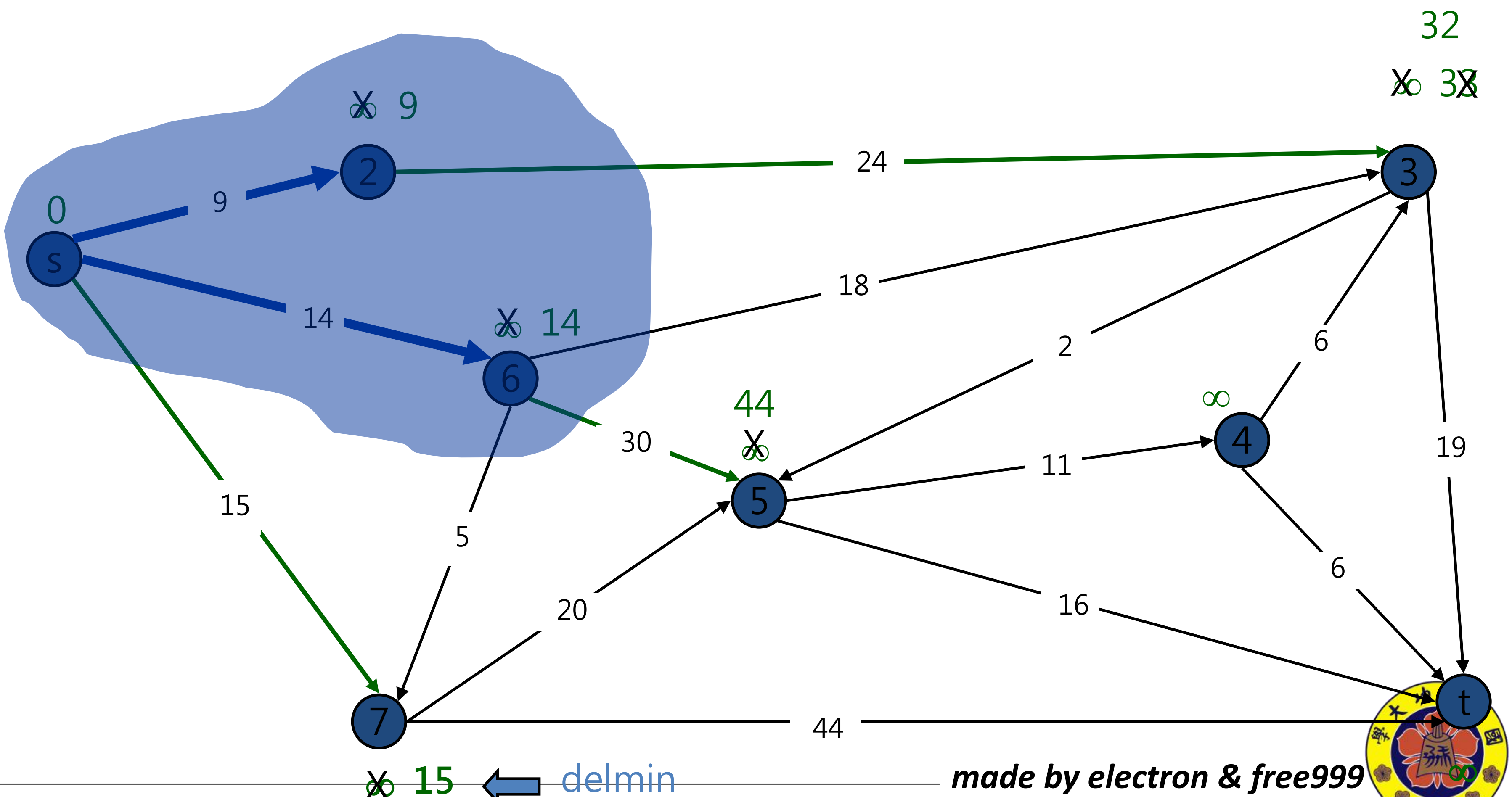
PQ = { 3, 4, 5, 6, 7, t }



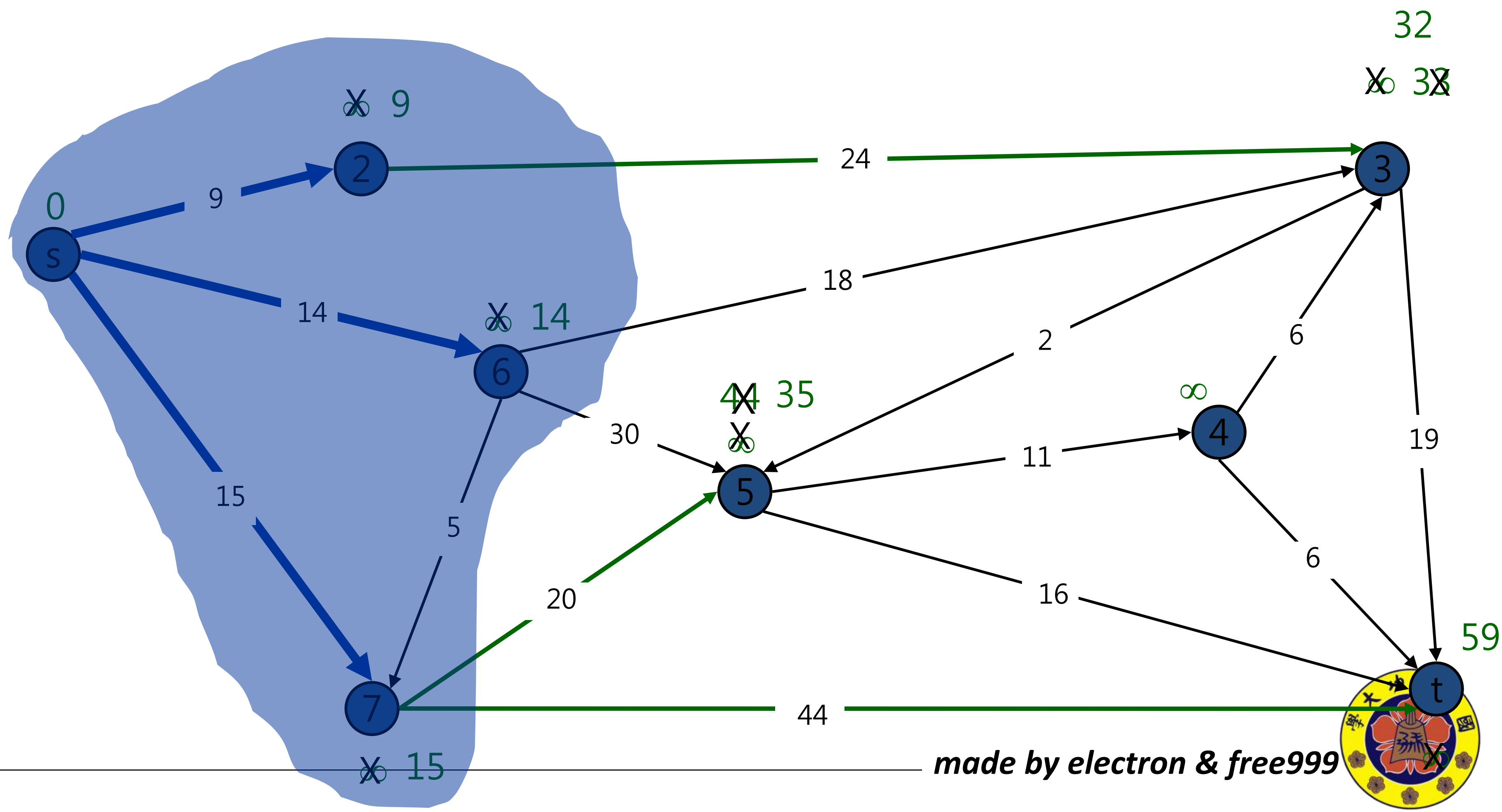
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }



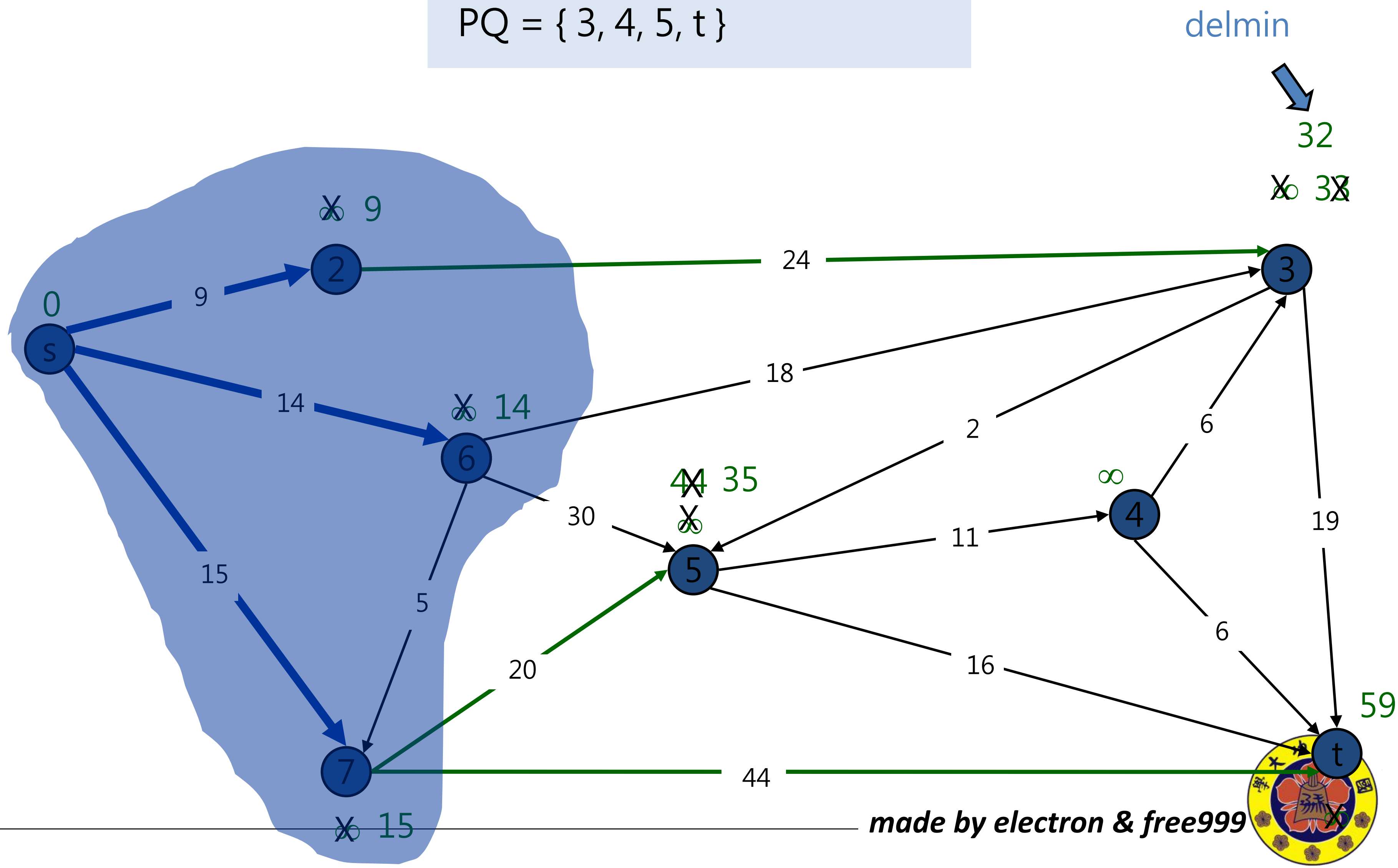
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }



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



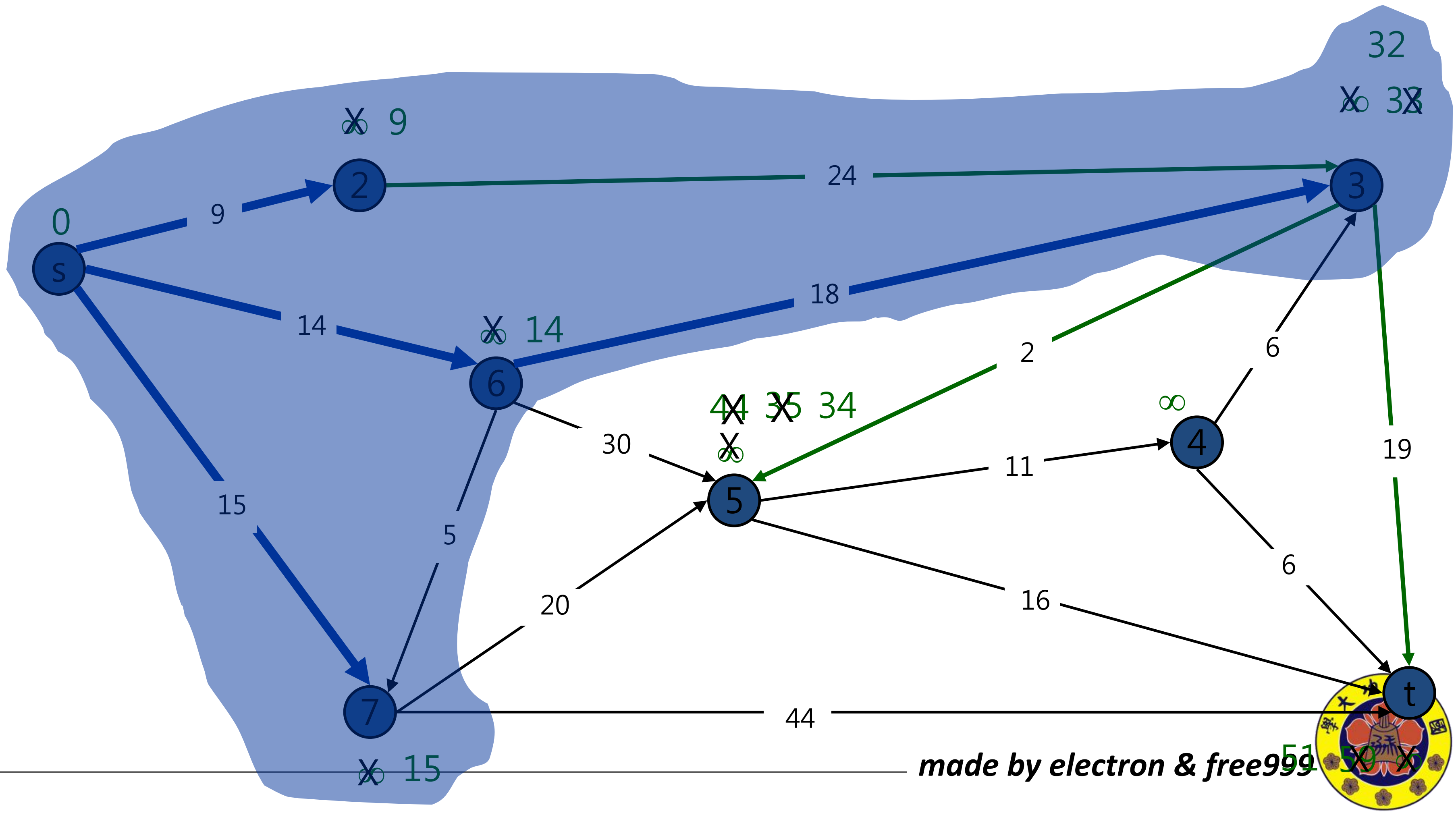
S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }



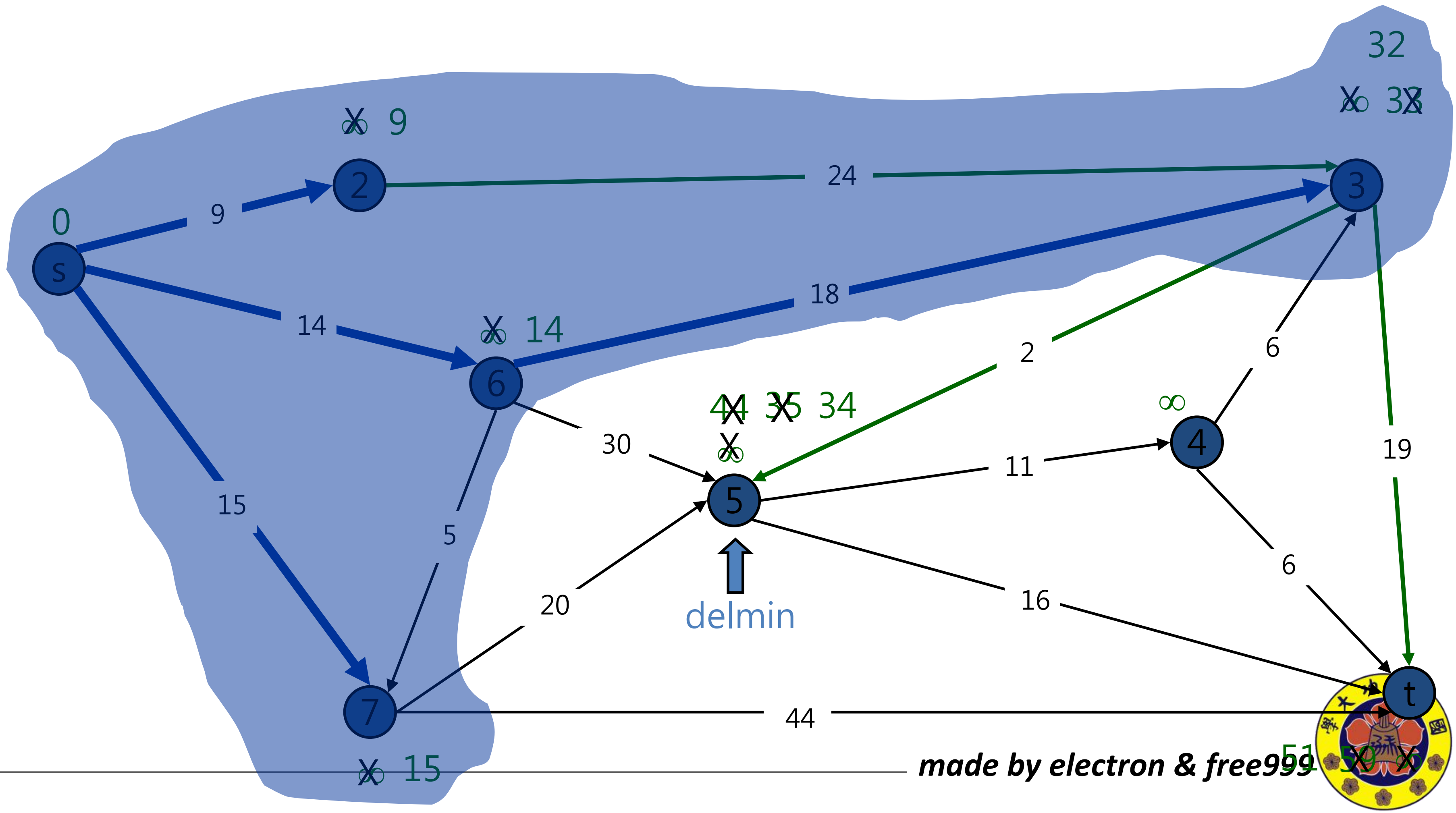
made by electron & free999

S = { s, 2, 3, 6, 7 }

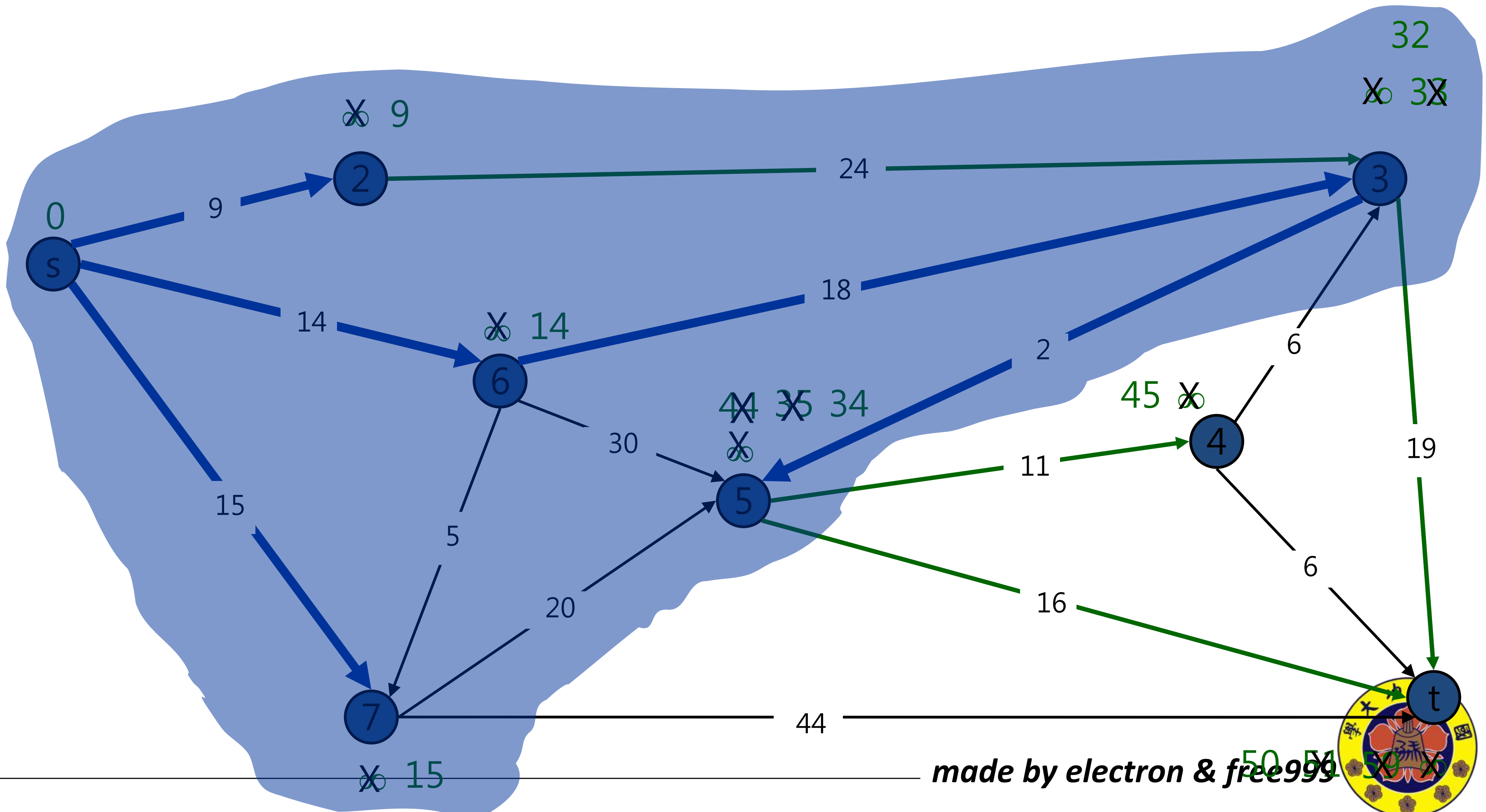
PQ = { 4, 5, t }



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

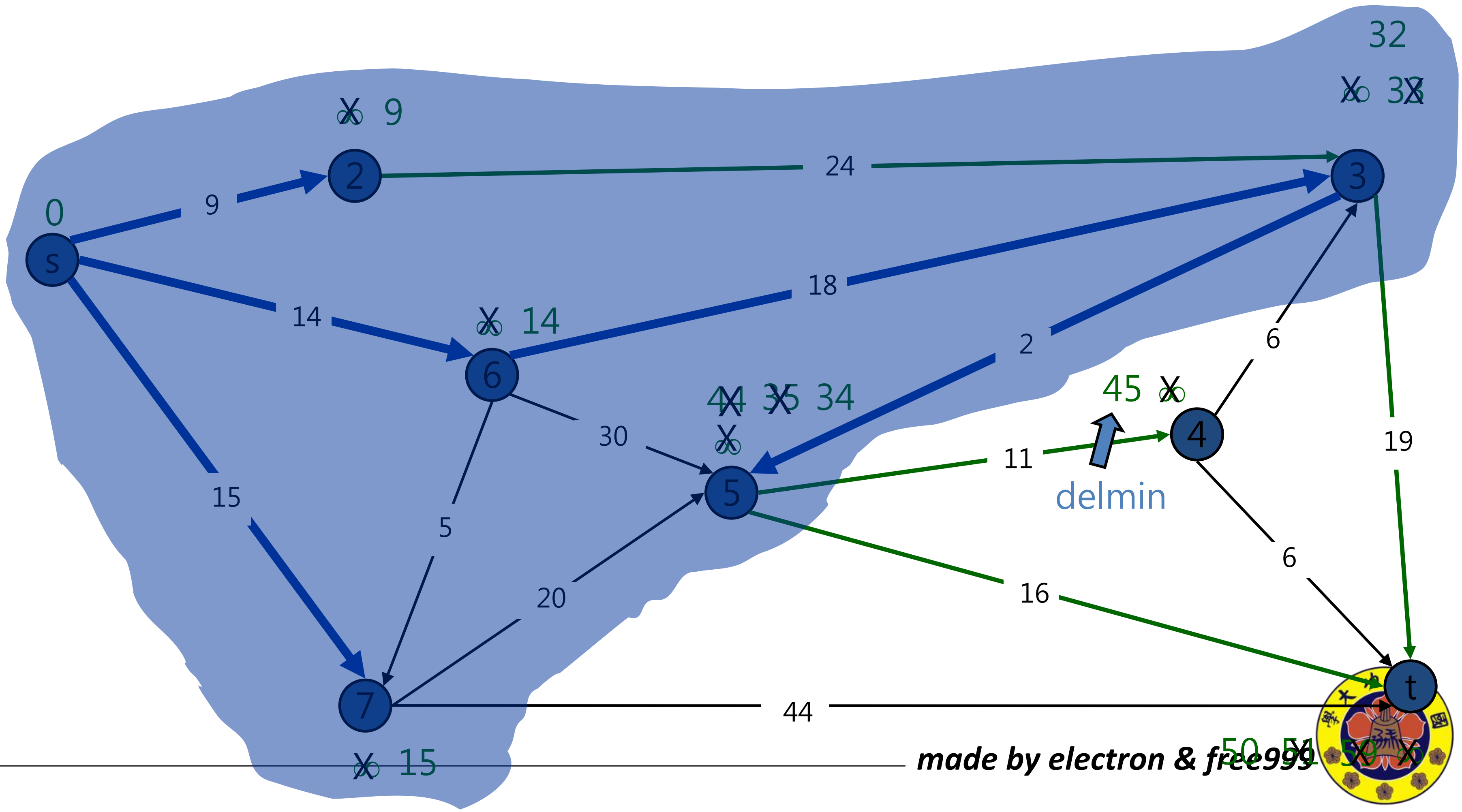


S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }



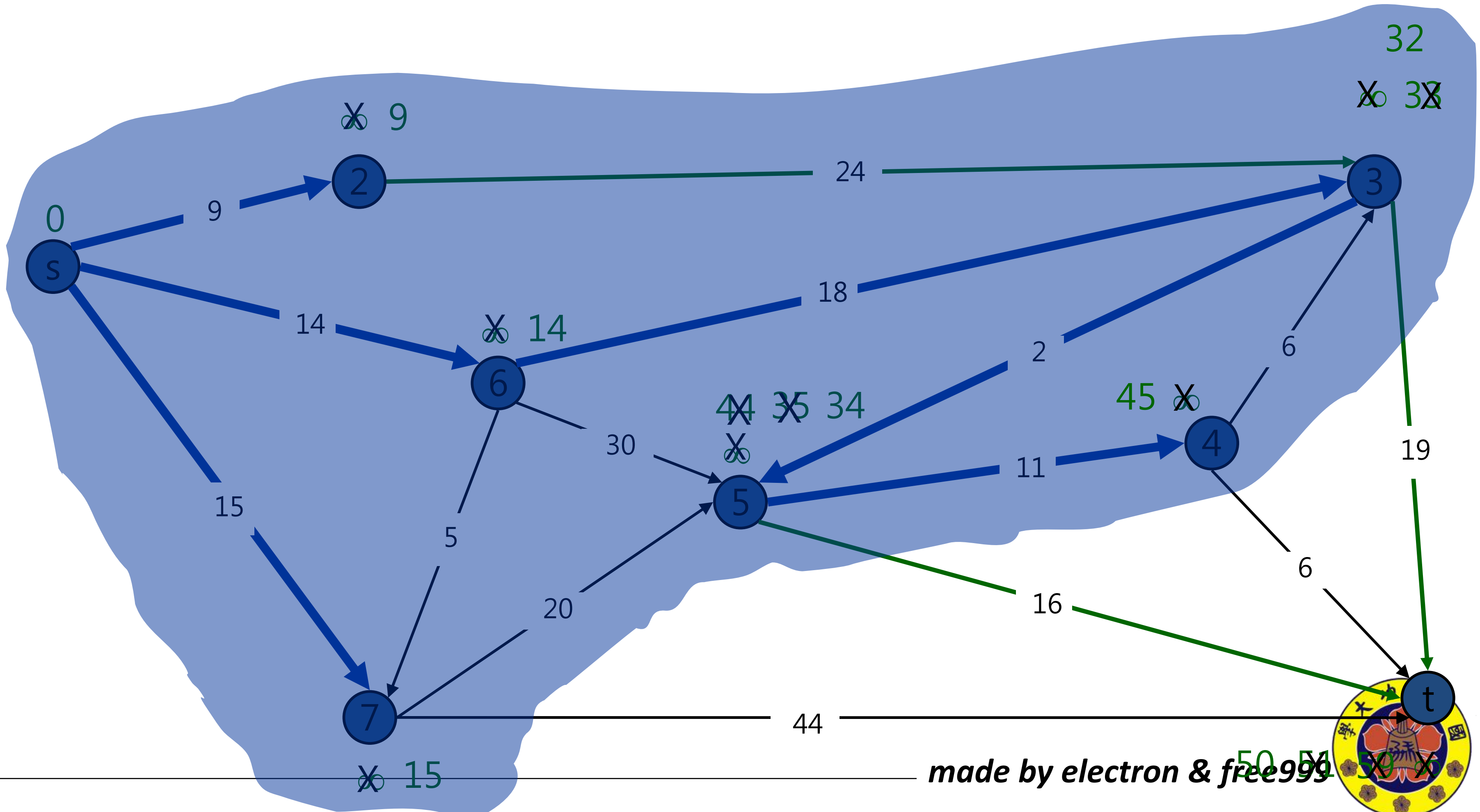
made by electron & free999

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



made by electron & free999

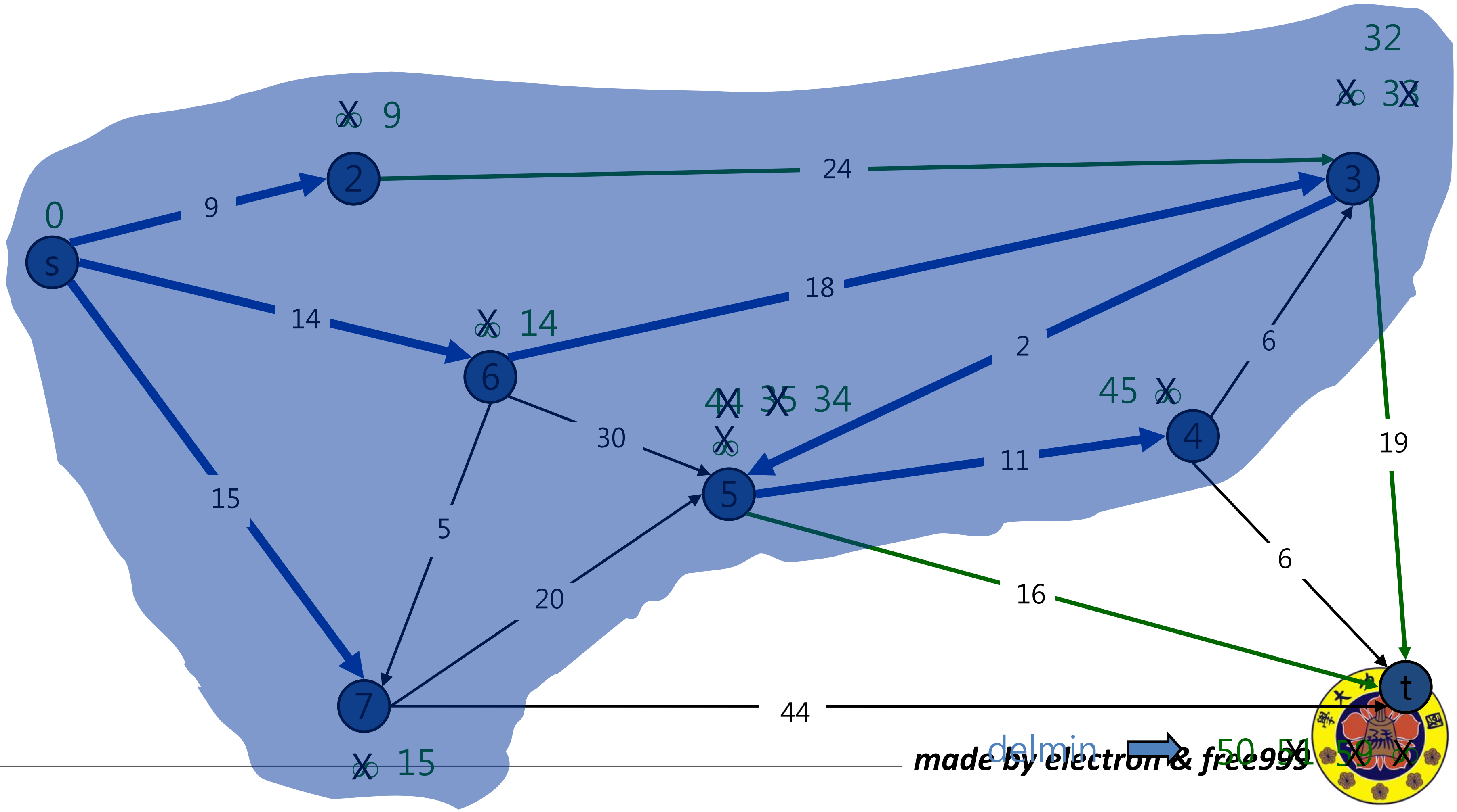
S = { s, 2, 3, 4, 5, 6, 7 }
PQ = { t }



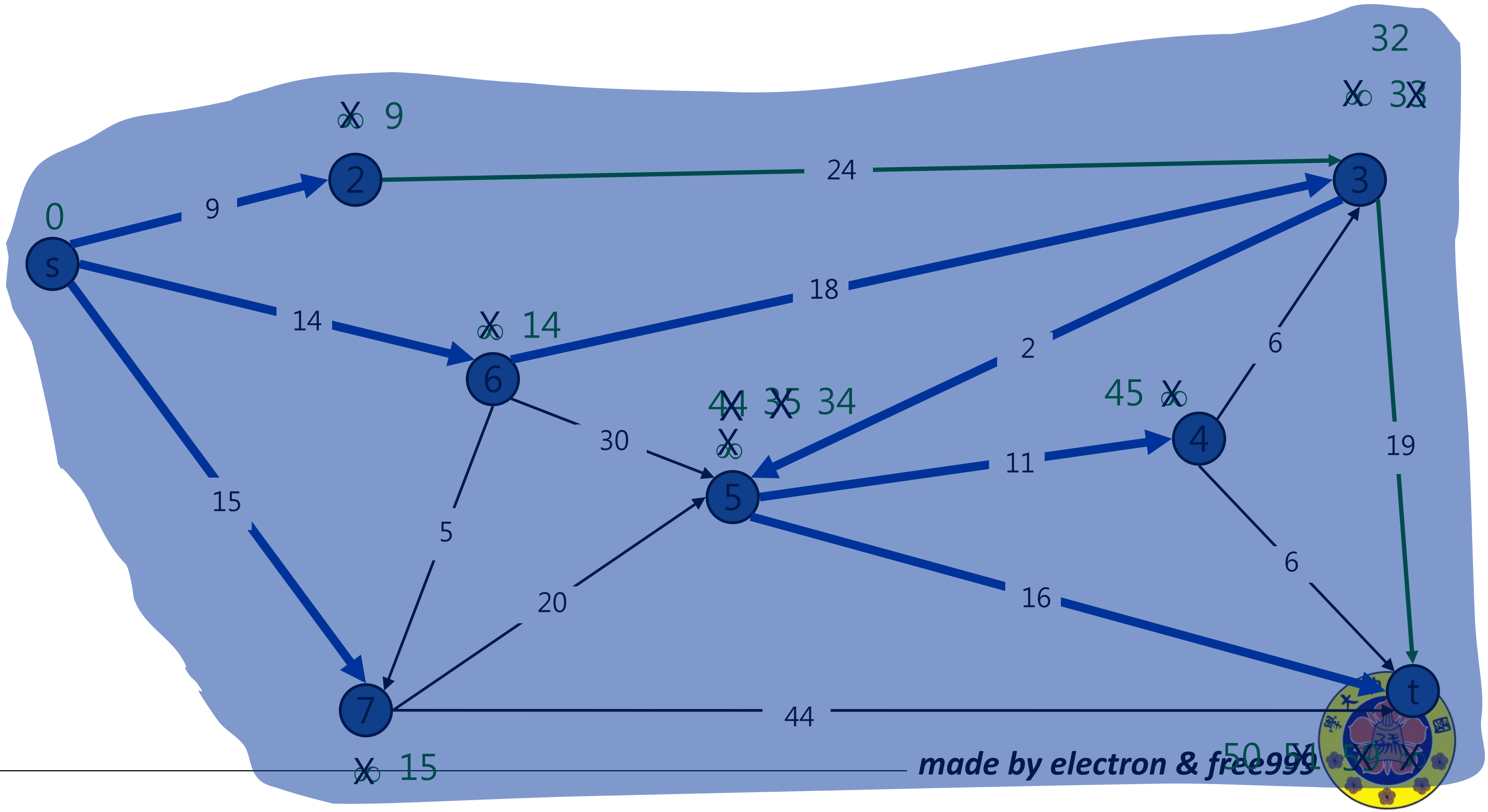
made by electron & free999

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

PQ = { t }

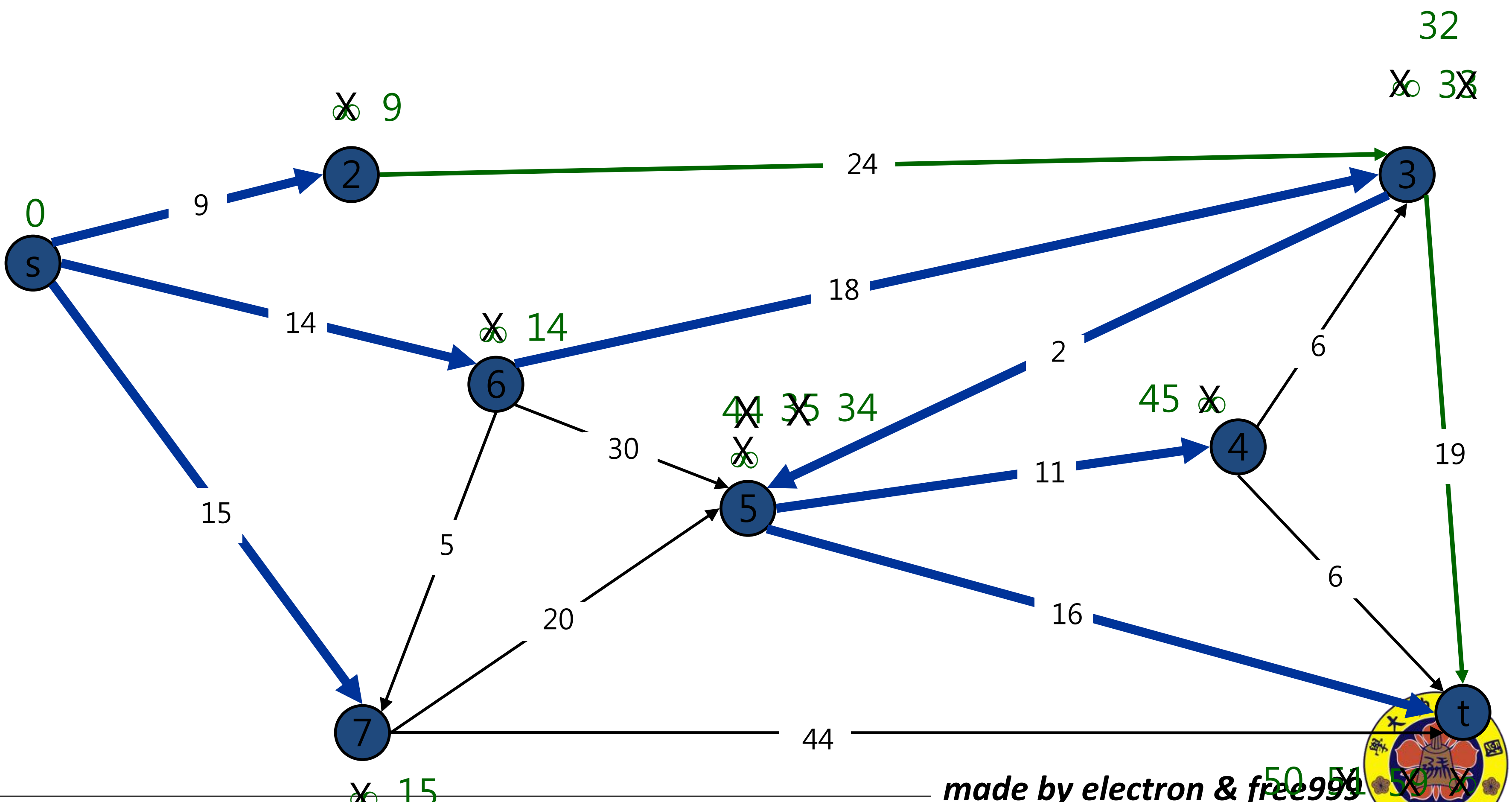


S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



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

PQ = { }



Dijkstra

Queue 改成 Priority_queue

made by electron & free999

example

Uva 10841

PKU 1724

Difference Constraint

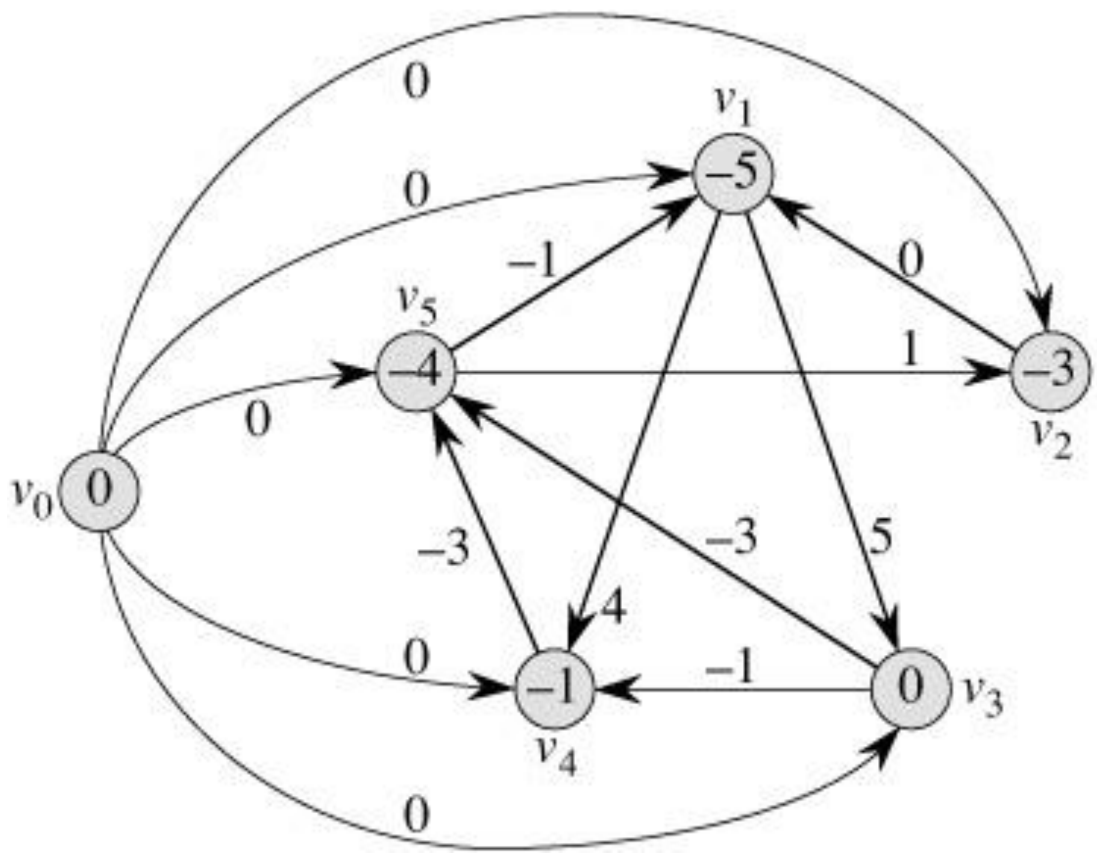
Given :

X1 - X2 <= 0
X1 - X5 <= -1
X2 - X5 <= 1
X3 - X1 <= 5
X4 - X1 <= 4
X4 - X3 <= -1
X5 - X3 <= -3
X5 - X4 <= -3

Find :

A feasible solution of X1, X2, …,X5

Difference Constraint



Difference Constraint

PKU-1201

PKU-2983

K-shortest Path

Homework

Uva

104, 125, 186, 350, 408, 436, 517, 523, 821, 925, 10075, 10171, 10246, 10518, 10591, 10724, 10793, 10803, 11015, 11053, 11156, 11284, 10296, 10987, 11549

- Shortest Path
  - POJ 1860, 3259, 1062, 2253, 1125, 2240, 2949, 1511, 3635, 1376, 3159, 1201, 2983, 1724
  - UVA 563, 753, 820, 10249, 10330, 10380, 10779, 10801, 10841, 10278, 10187, 10039 , 10740, 10986