POSCAT Seminar 10 : Graph 2

yougatup @ POSCAT

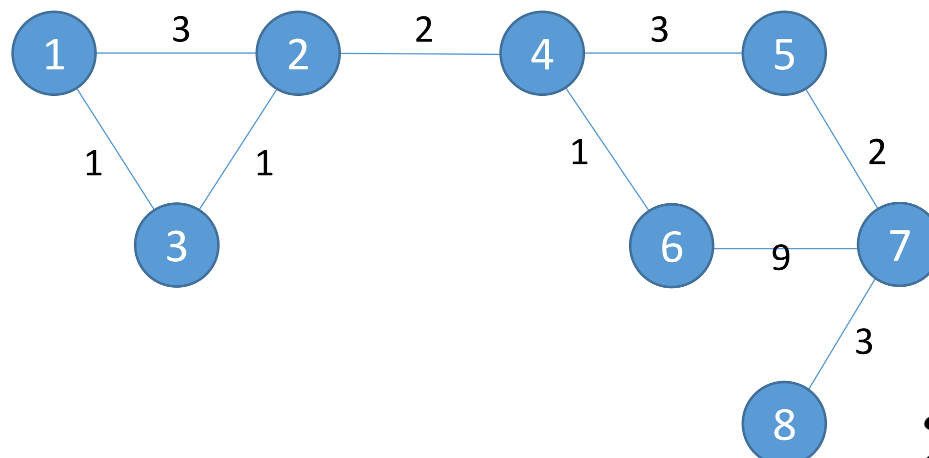
Topic

- Topic today
  - Shortest Path
    - Dijkstra Algorithm
    - Floyd Algorithm
    - Bellman-Ford Algorithm
  - Disjoint Set
    - Union & Find
    - Path Compression

Shortest Path

- Problem

  Given a graph, find a shortest path from start vertex to end vertex

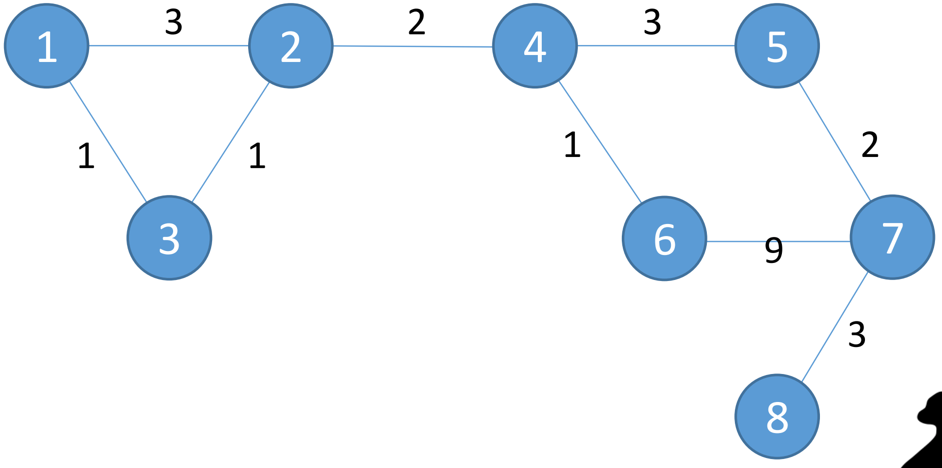


Shortest Path

- Problem

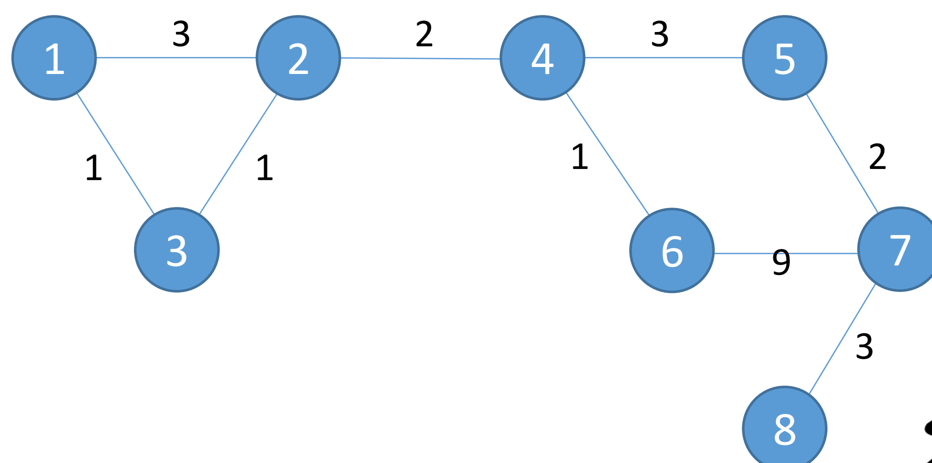
Given a graph, find a shortest path from start vertex to end vertex

1. Greedy Approach
2. Iterative Approach
3. Dynamic Programming Approach



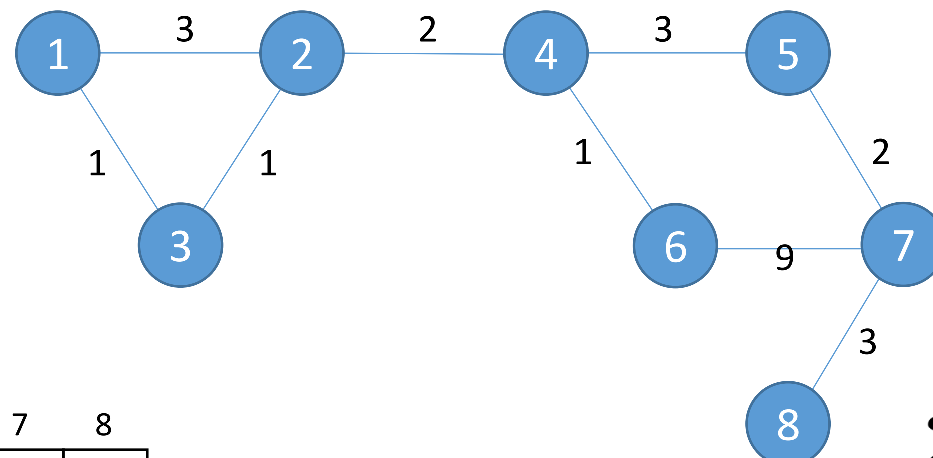
Dijkstra Algorithm

- Approach
  - Greedy approach
  - Let $S(i) = the\ length\ of\ shortest\ path\ from\ the\ start\ vertex\ to\ vertex\ i$



Dijkstra Algorithm

- Approach
  - Greedy approach
  - Let $S(i) = \textit{the length of shortest path from the start vertex to vertex } i$



	1	2	3	4	5	6	7	8
S	0	∞	∞	∞	∞	∞	∞	∞

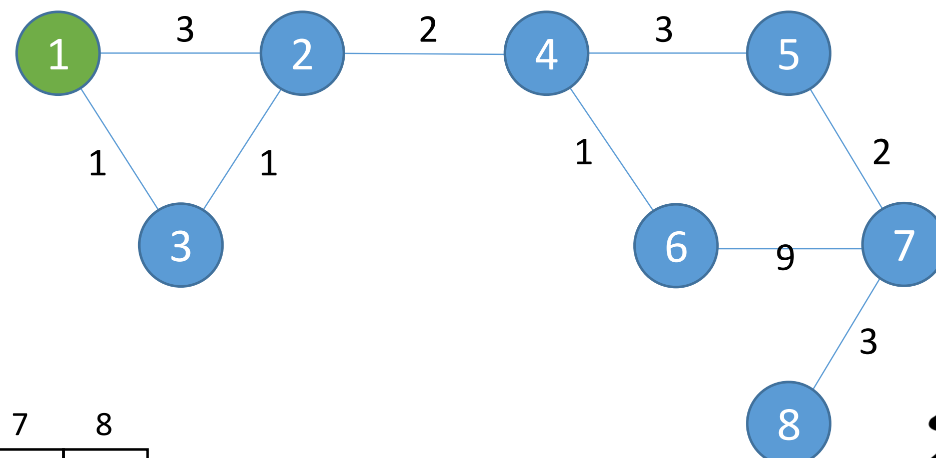
We don't know the shortest path length of these

Poscat

Dijkstra Algorithm

- Approach

  Currently, we only know that the shortest path length from the start vertex to start vertex → 0

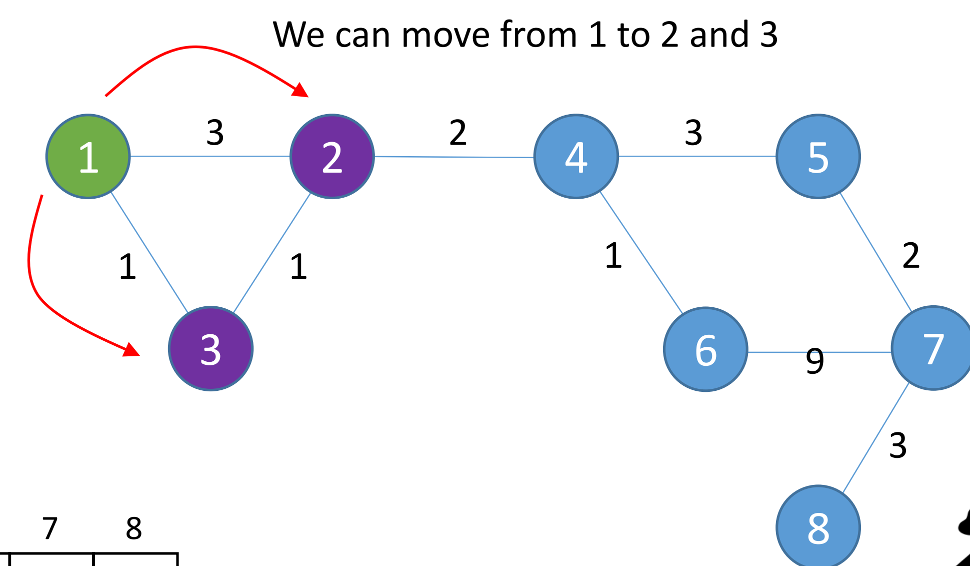


	1	2	3	4	5	6	7	8
S	**0**	∞	∞	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Approach

  Also, we can fill another cell of S by using this information !

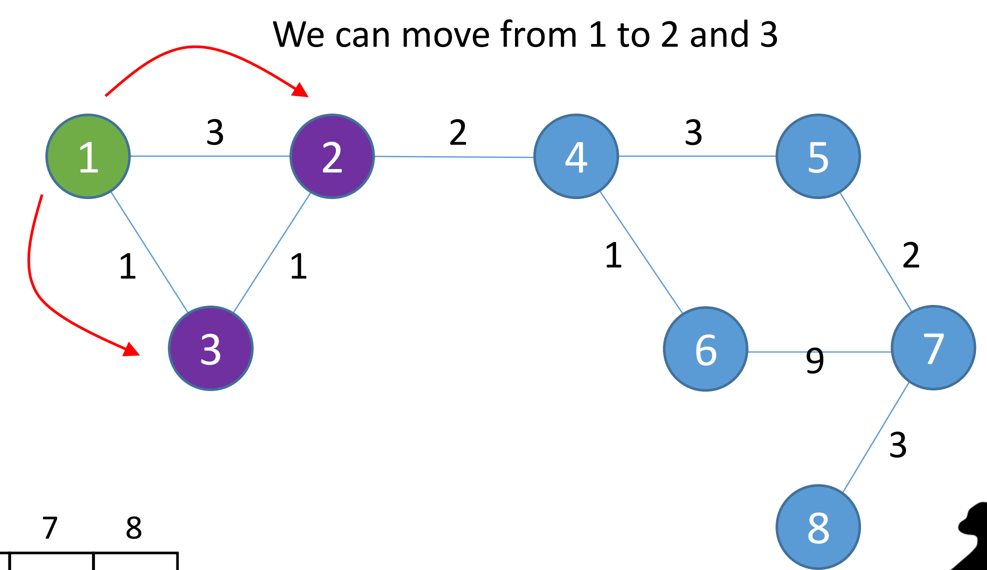


	1	2	3	4	5	6	7	8
S	0	∞	∞	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Approach

  Also, we can fill another cell of S by using this information !



	1	2	3	4	5	6	7	8
S	0	**3**	**1**	∞	∞	∞	∞	∞

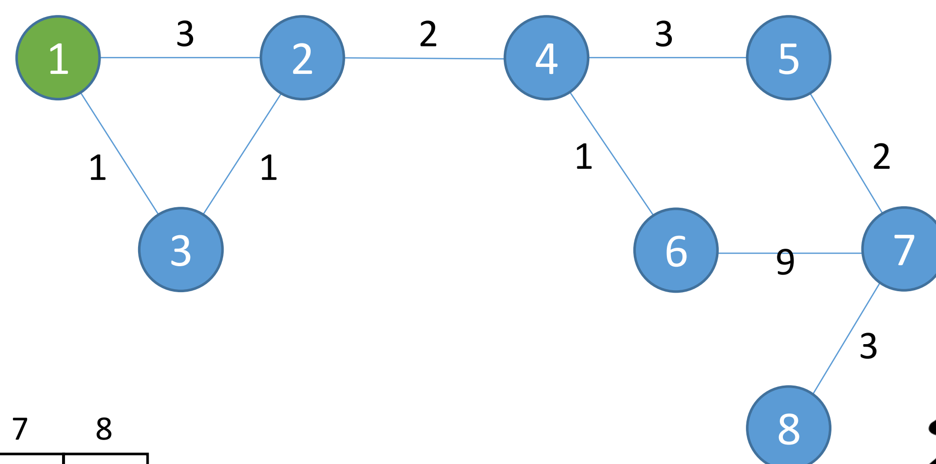
Poscat

Dijkstra Algorithm

- Approach

  Also, we can fill another cell of S by using this information !

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

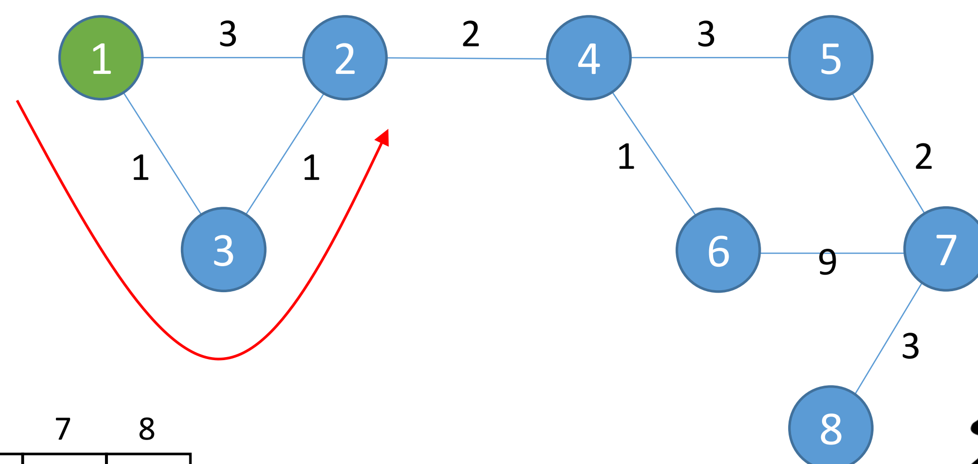
By the way, is it correct value by definition ?

Dijkstra Algorithm

- Approach

  Also, we can fill another cell of S by using this information !

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

By the way, is it correct value by definition ? **NO !**

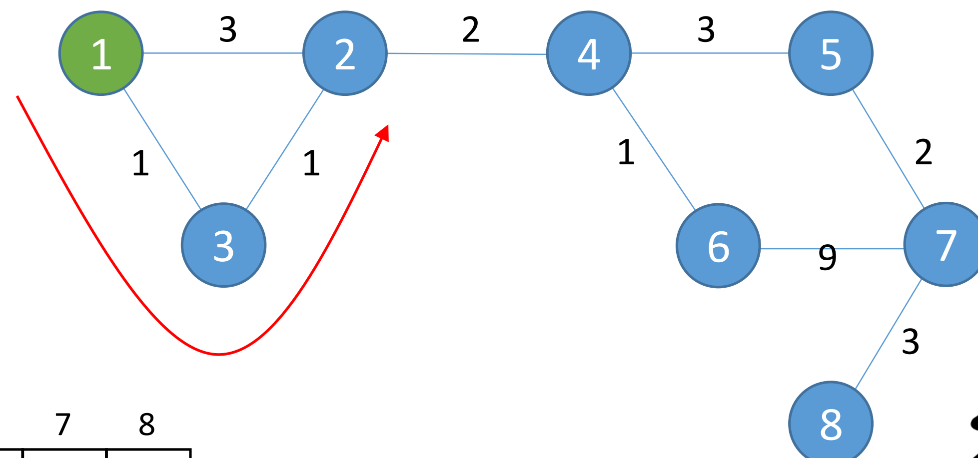
Poscat

Dijkstra Algorithm

- Approach

  Also, we can fill another cell of S by using this information !

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

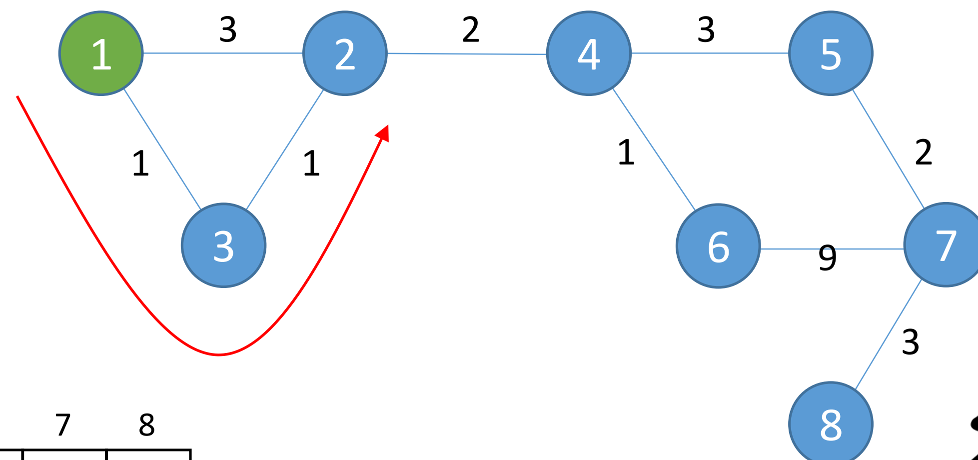
Then, which one is the correct value ?

Dijkstra Algorithm

- Approach

Also, we can fill another cell of S by using this information !

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

Then, which one is the correct value ?

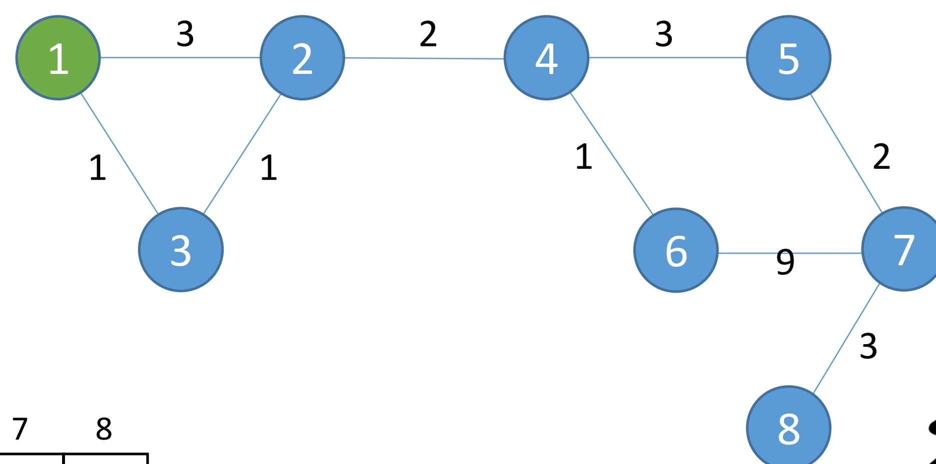
Poscat

Dijkstra Algorithm

- Approach

  Also, we can fill another cell of S by using this information !

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

Okay, keep going

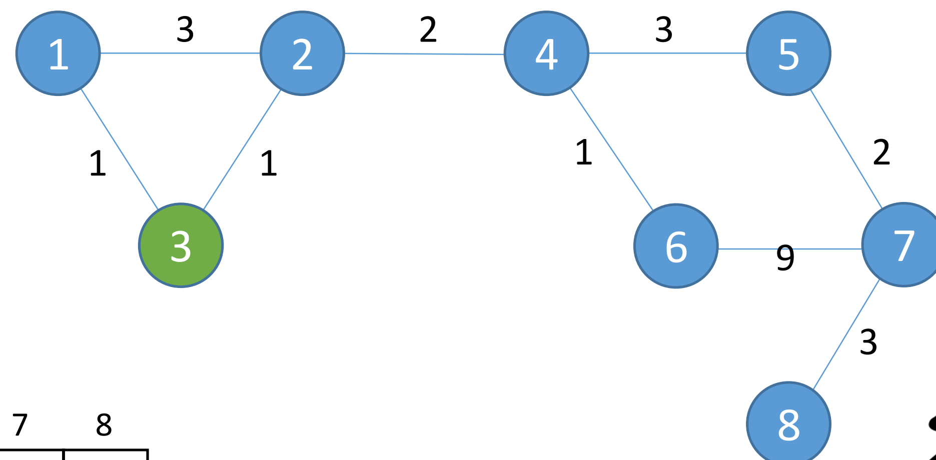
Poscat

Dijkstra Algorithm

- Approach

  Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



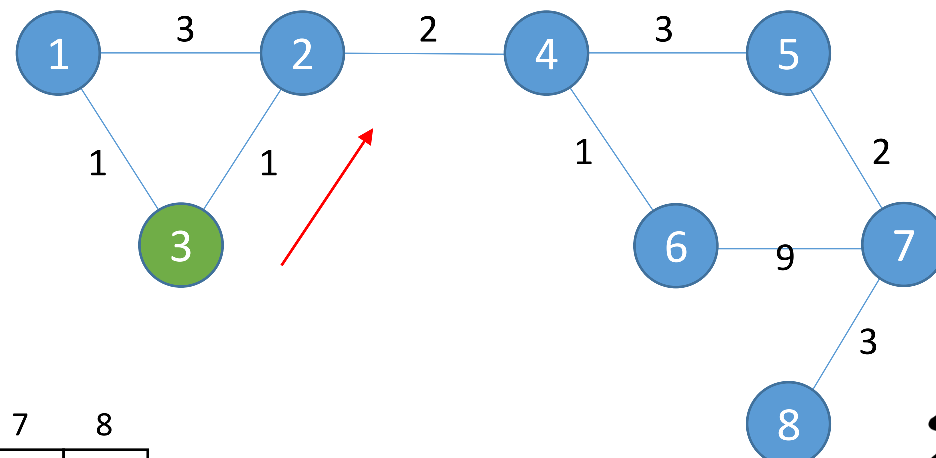
	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Approach

  Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

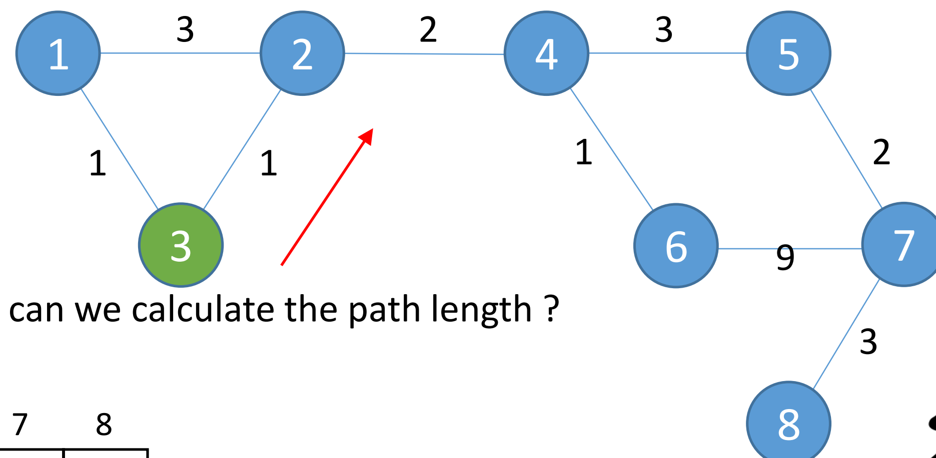
Poscat

Dijkstra Algorithm

- Approach

  Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



How can we calculate the path length ?

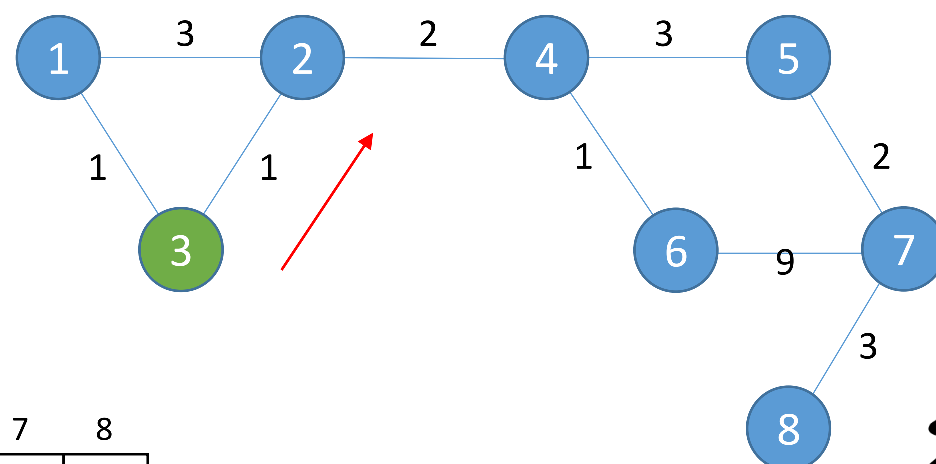
	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Approach

Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



First, we move from 1 to 3 via a shortest path
Second, we move from 3 to 2 via a edge

	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

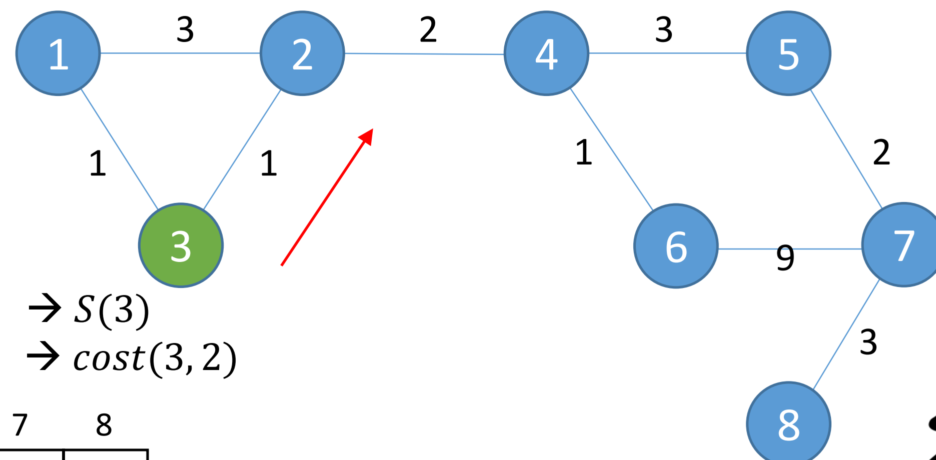
Poscat

Dijkstra Algorithm

- Approach

  Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



First, we move from 1 to 3 via a shortest path → $S(3)$
Second, we move from 3 to 2 via a edge → $cost(3,2)$

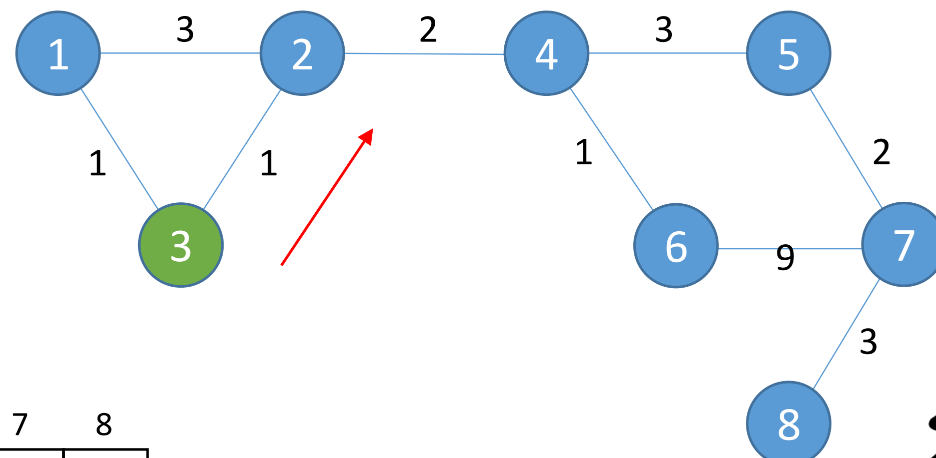
	1	2	3	4	5	6	7	8
S	0	3	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Approach

  Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	**2**	1	∞	∞	∞	∞	∞

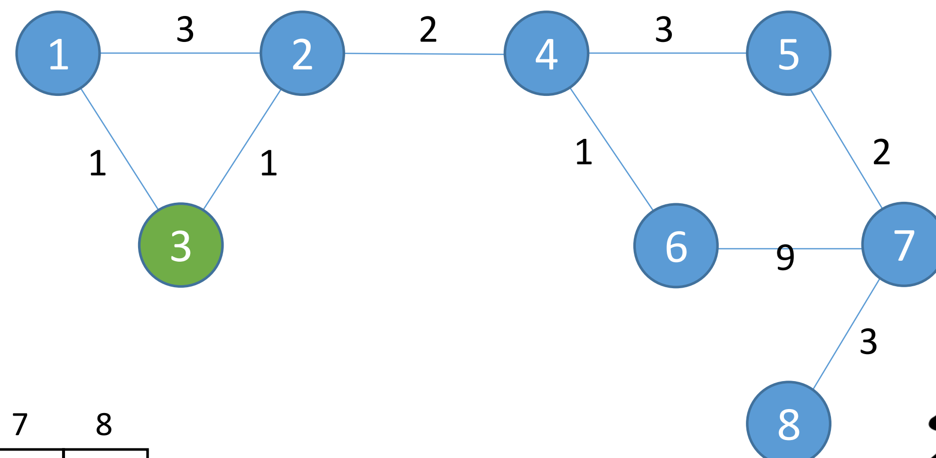
We can update this value !

Dijkstra Algorithm

- Approach

  Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Then, which is the correct shortest path length ? **WHY ?**

Poscat

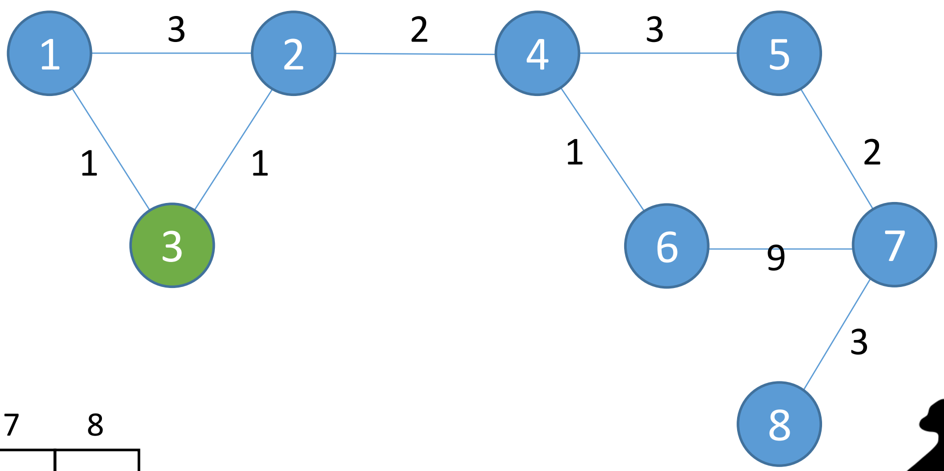
Dijkstra Algorithm

- Approach

Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3

We already use the information of $S(1)$ and $S(3)$



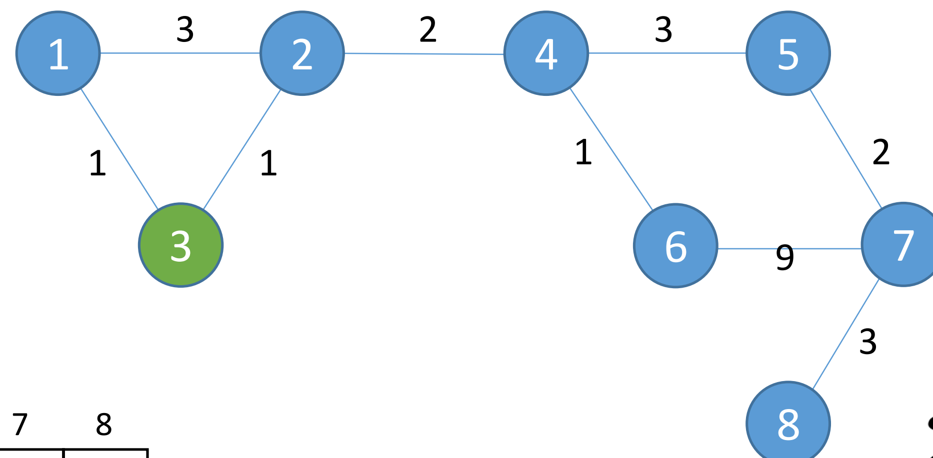
	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Approach

Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



We already use the information of $S(1)$ and $S(3)$
It means that we already consider paths which contains the vertex 1 or 3

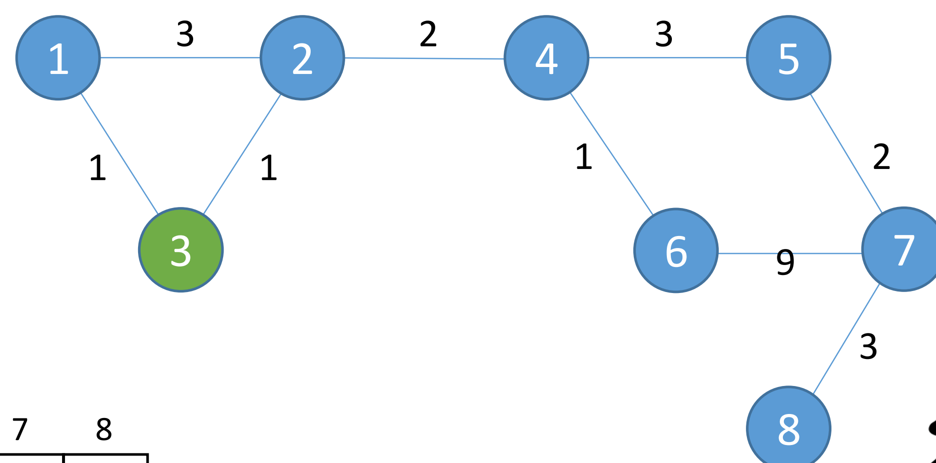
	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Approach

Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



We already use the information of $S(1)$ and $S(3)$
It means that we already consider paths which contains the vertex 1 or 3
The remaining case is to consider paths which contains the another vertices (white boxes)

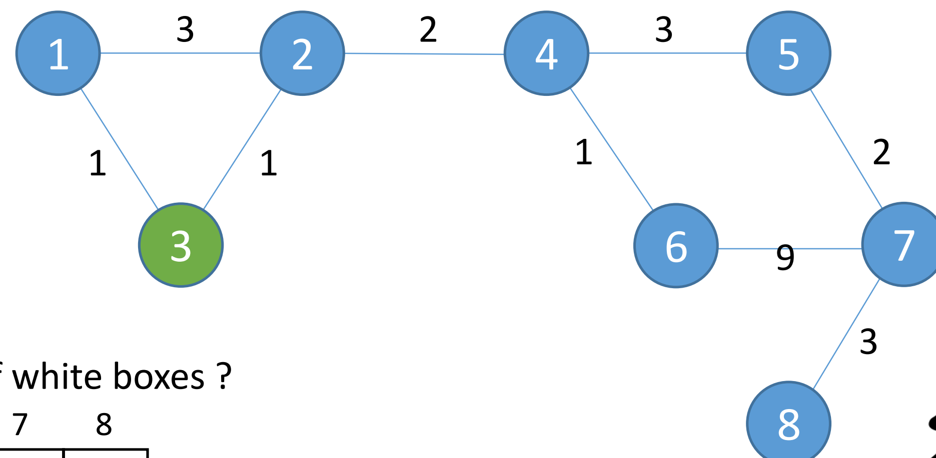
	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Approach

Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



We already use the information of $S(1)$ and $S(3)$
It means that we already consider paths which contains the vertex 1 or 3
The remaining case is to consider paths which contains the another vertices (white boxes)
Can we update the value of $S(2)$ with the value of white boxes ?

	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

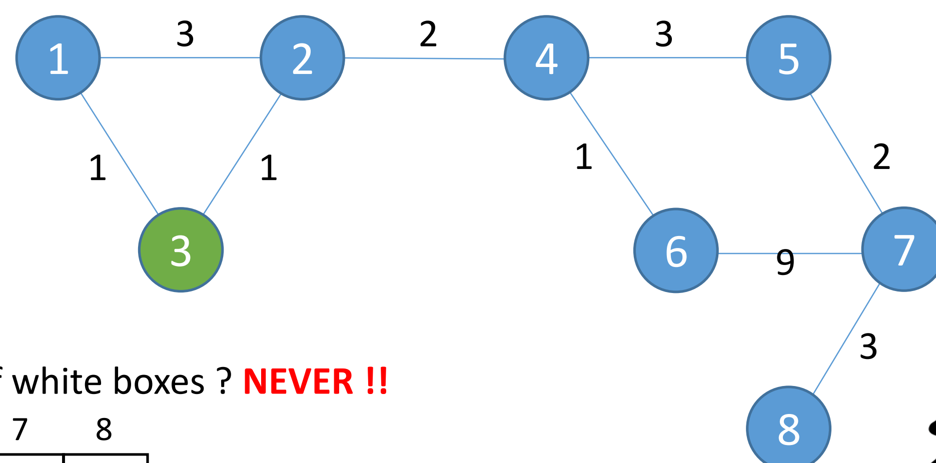
Poscat

Dijkstra Algorithm

- ## Approach

Because $S(3)$ is the correct value, also we can fill another cell

We can move from 1 to 2 and 3



We already use the information of $S(1)$ and $S(3)$
It means that we already consider paths which contains the vertex 1 or 3
The remaining case is to consider paths which contains the another vertices (white boxes)
Can we update the value of $S(2)$ with the value of white boxes ? **NEVER !!**

	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Poscat

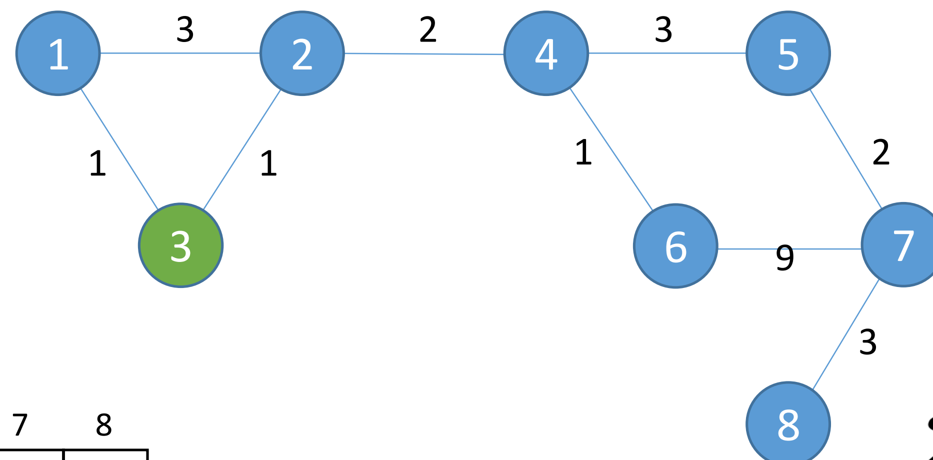
Dijkstra Algorithm

- Approach

  Blue boxes contain the correct value of shortest path length

  We NEVER update the minimum value of white boxes !

We can move from 1 to 2 and 3



	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

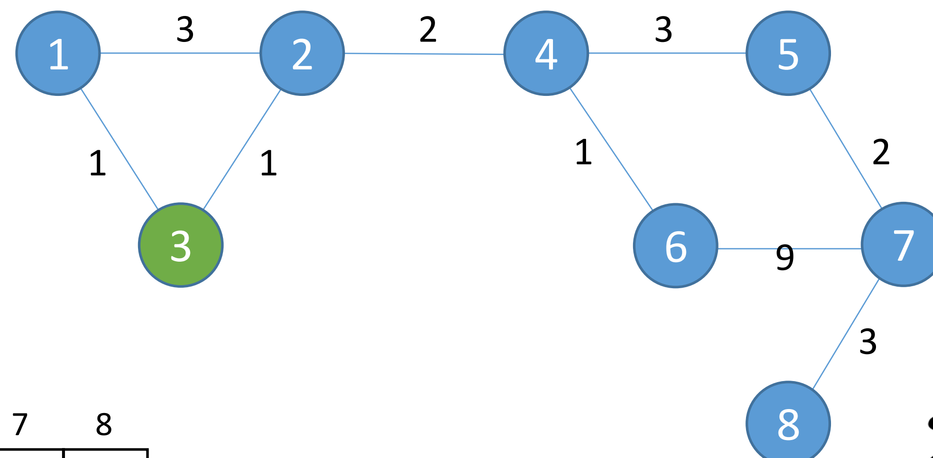
- Approach

  Blue boxes contain the correct value of shortest path length

  We NEVER update the minimum value of white boxes !

  Therefore, the minimum value of white boxes is the correct value

We can move from 1 to 2 and 3

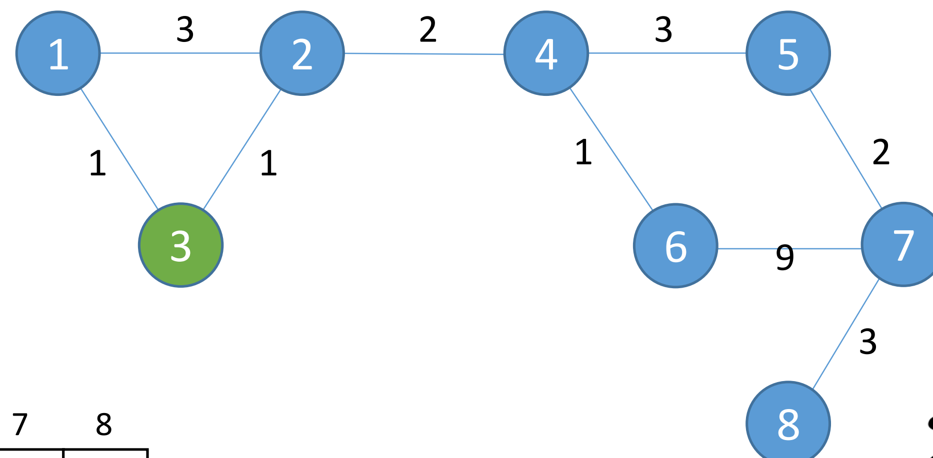


	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3

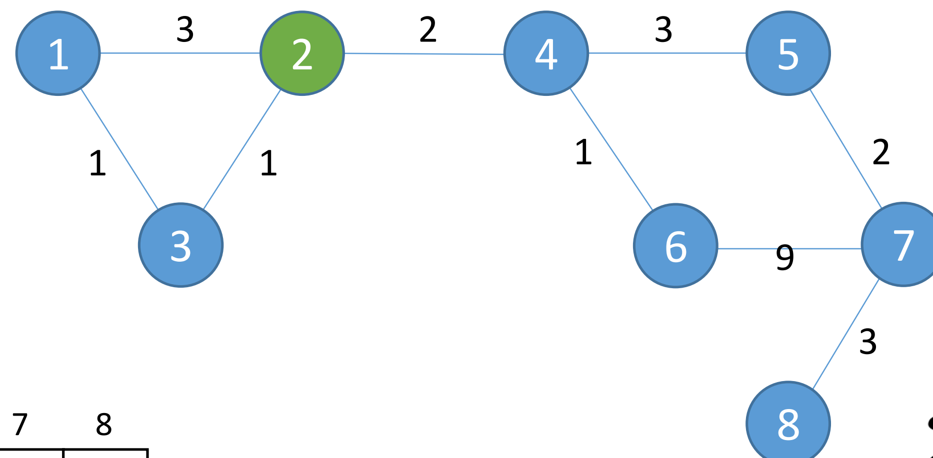


	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3



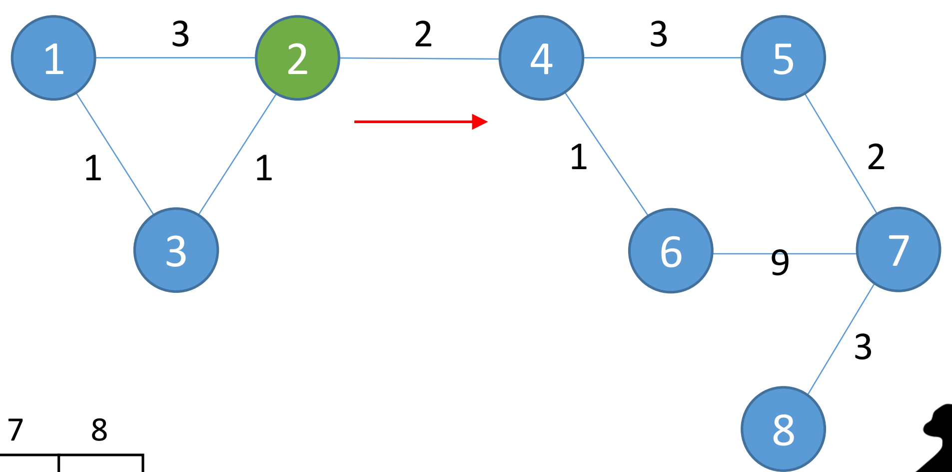
	1	2	3	4	5	6	7	8
S	0	2	1	∞	∞	∞	∞	∞

Poscat

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3



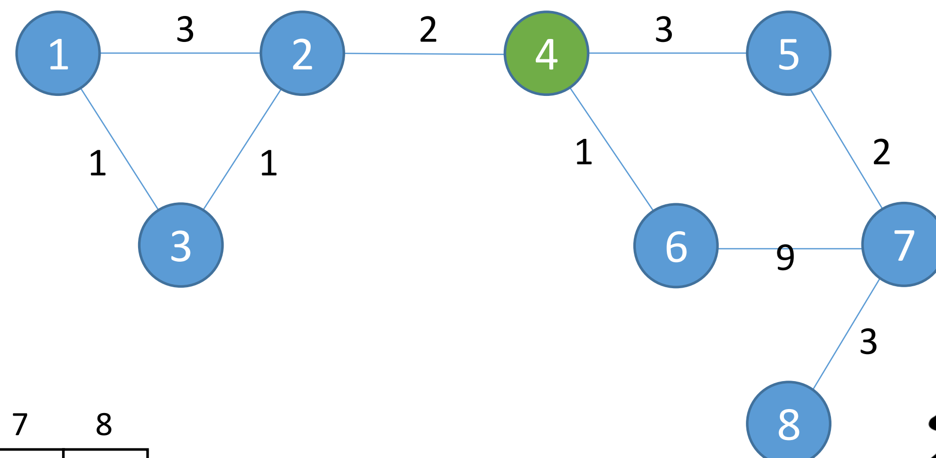
	1	2	3	4	5	6	7	8
S	0	2	1	**4**	∞	∞	∞	∞

Poscat

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3



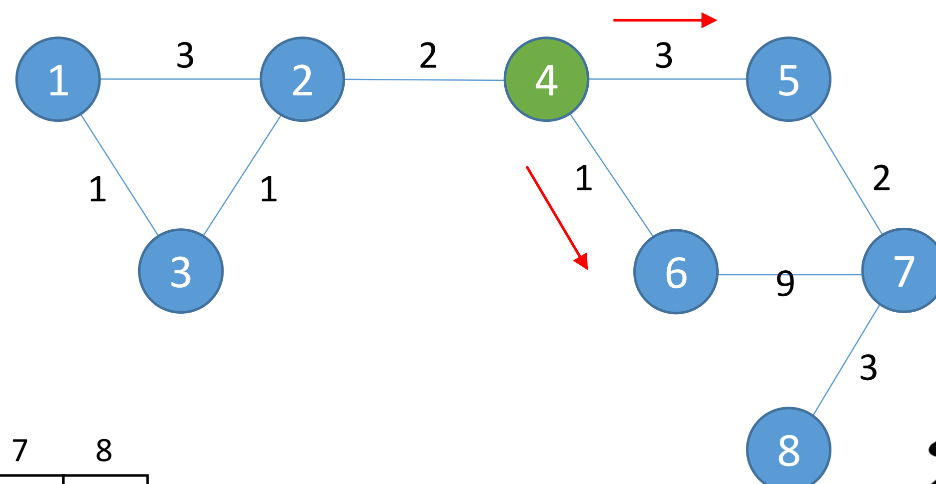
	1	2	3	4	5	6	7	8
S	0	2	1	4	∞	∞	∞	∞

Poscat

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3

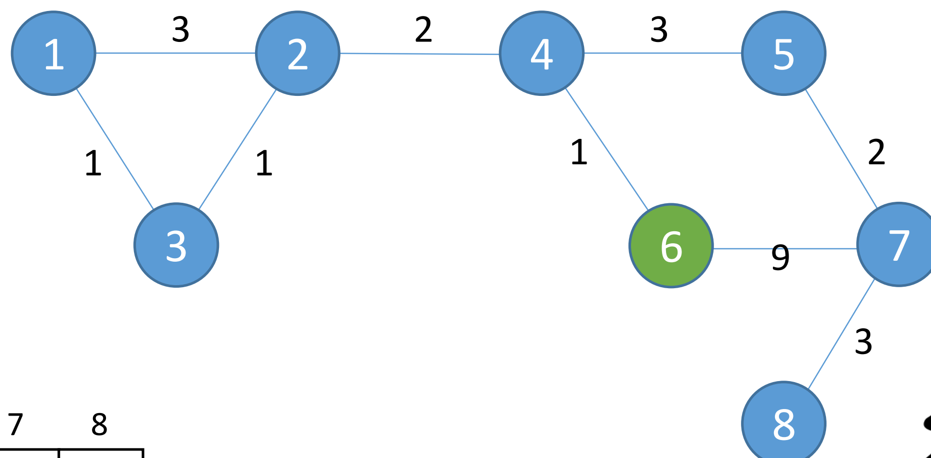


	1	2	3	4	5	6	7	8
S	0	2	1	4	**7**	**5**	∞	∞

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3



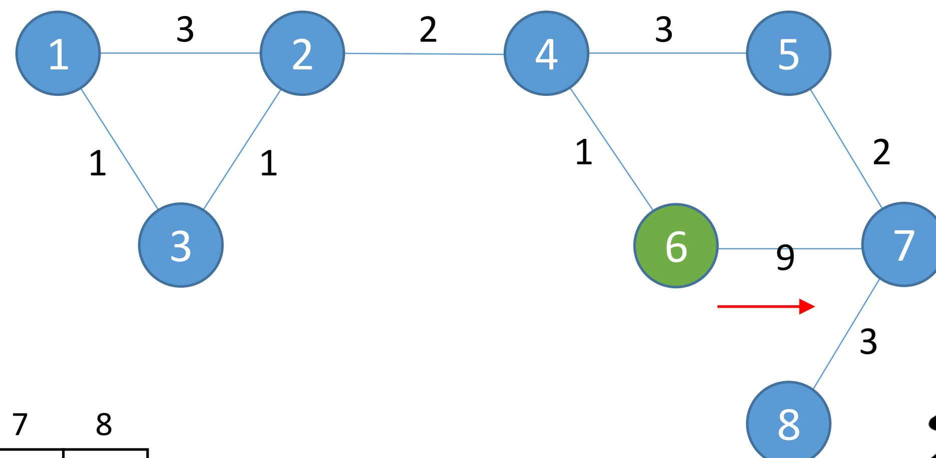
	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	∞	∞

Poscat

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3

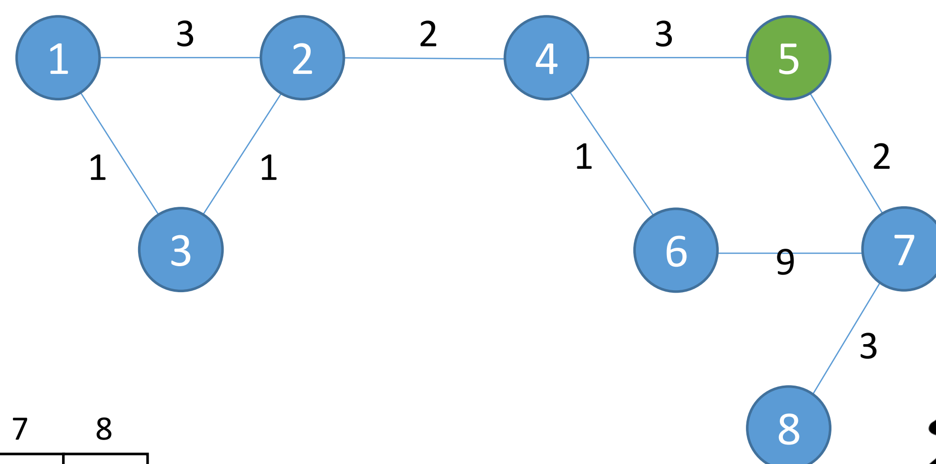


	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	**14**	∞

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3



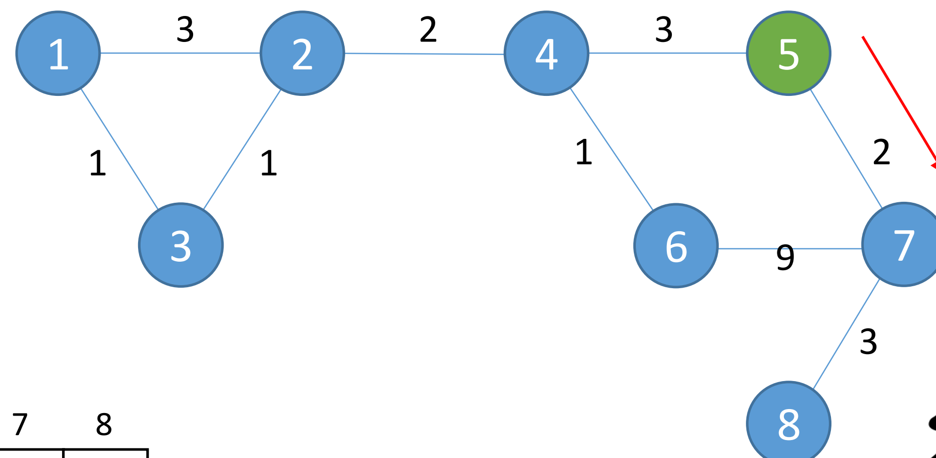
	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	14	∞

Poscat

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3

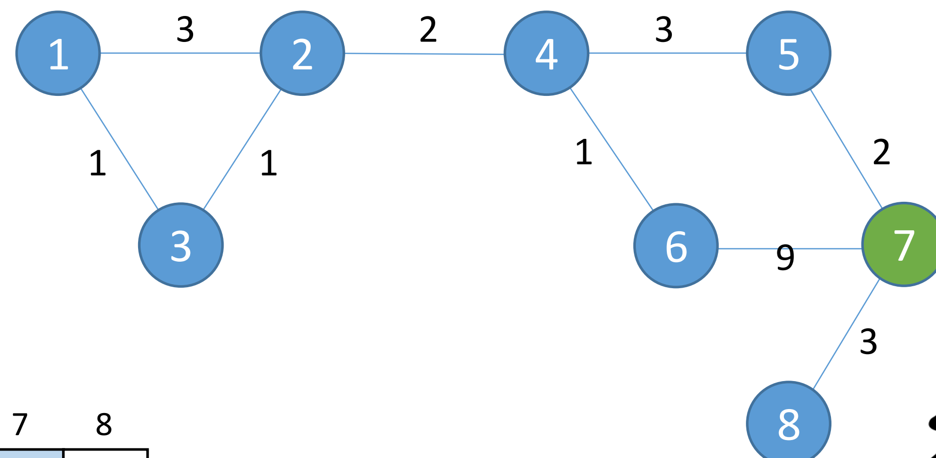


	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	**9**	∞

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3

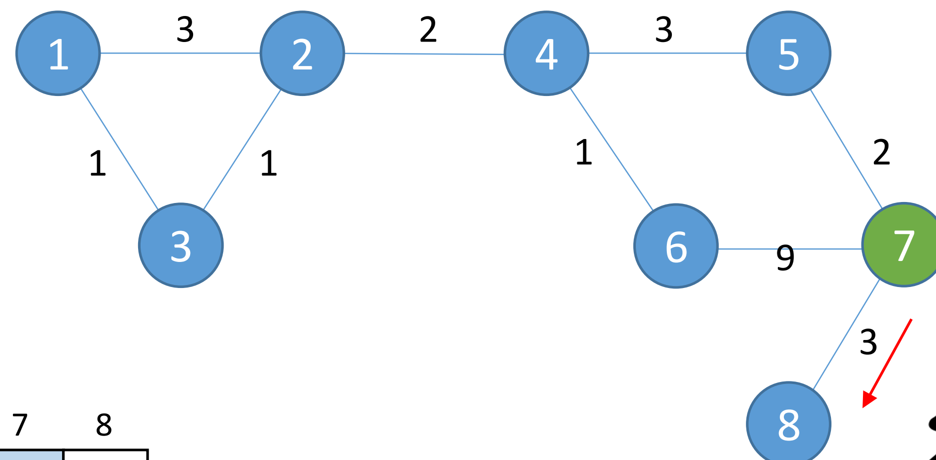


	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	∞

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3

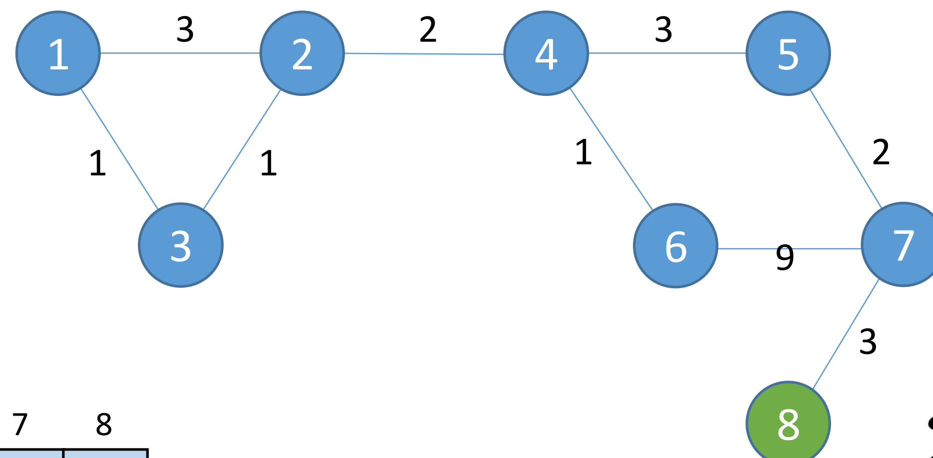


	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	**12**

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3

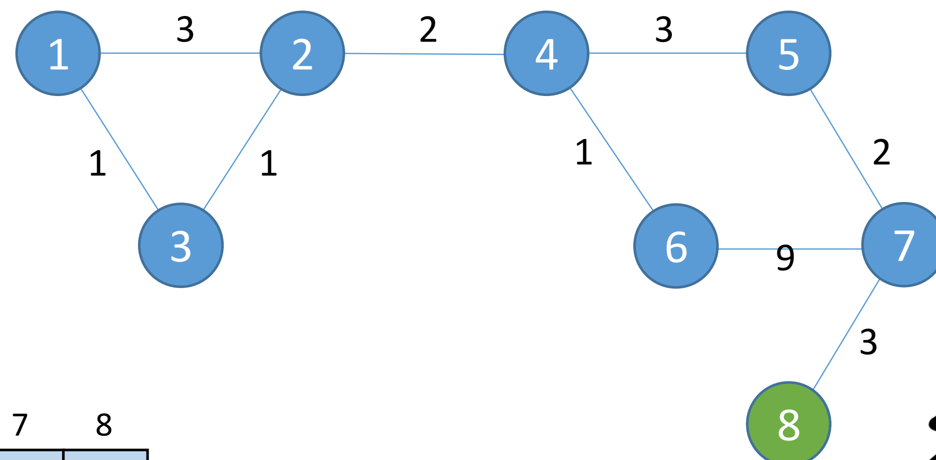


	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3



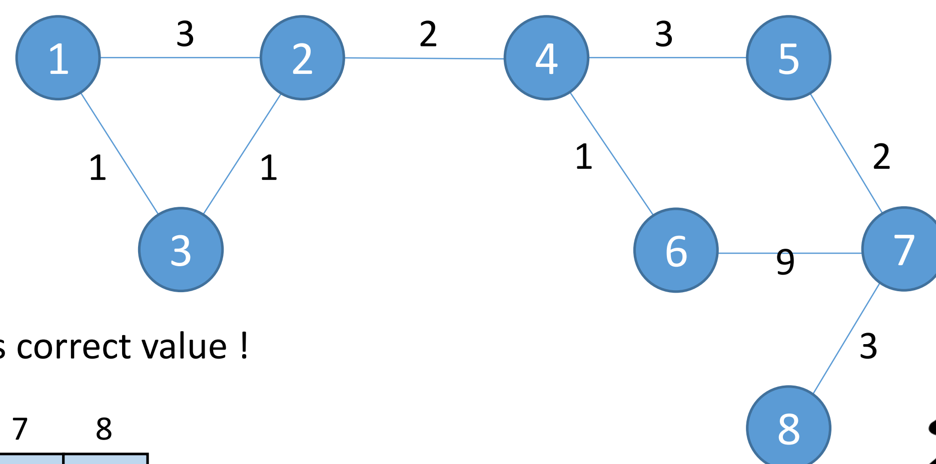
No one is updated ☺

	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Dijkstra Algorithm

- Overall algorithm
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3



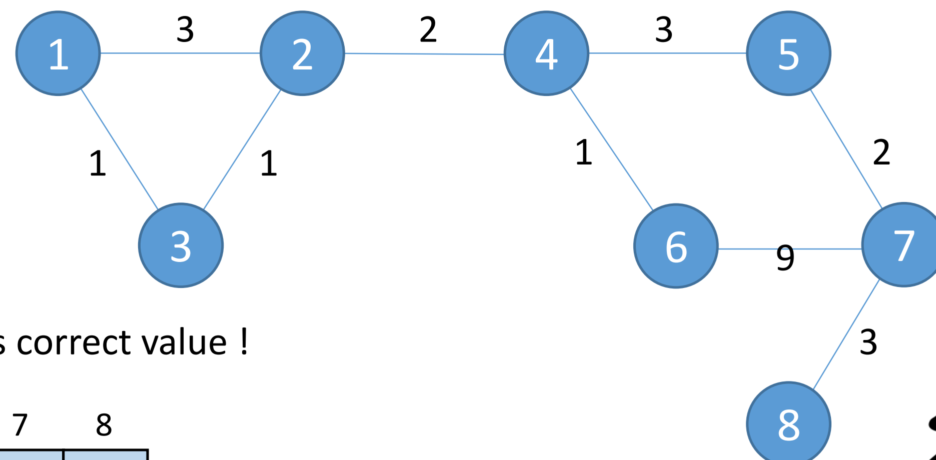
Done! Because all the values in S is correct value !

	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Dijkstra Algorithm

- Analysis
  1. Choose the white box which contains minimum value
  2. Update another cell
  3. Repeat !

We can move from 1 to 2 and 3



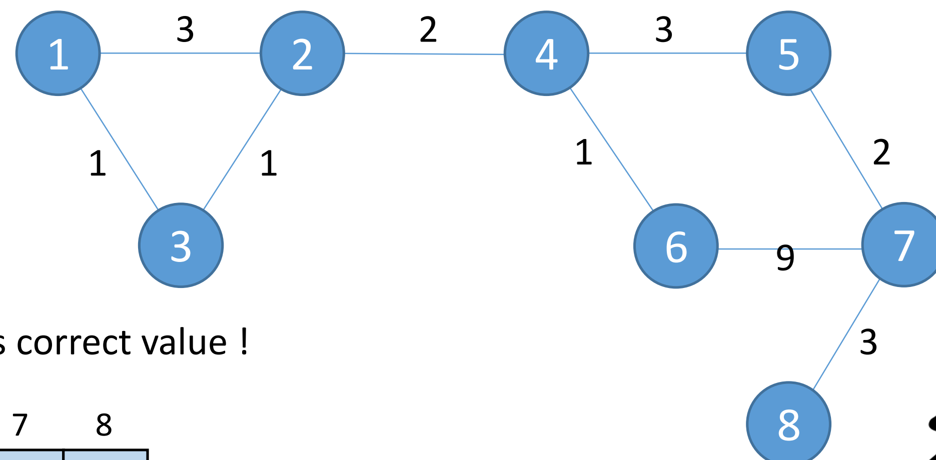
Done! Because all the values in S is correct value !

	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Dijkstra Algorithm

- Analysis
  1. Choose the white box which contains minimum value → O(V)
  2. Update another cell → O(V)
  3. Repeat !

We can move from 1 to 2 and 3



Done! Because all the values in S is correct value !

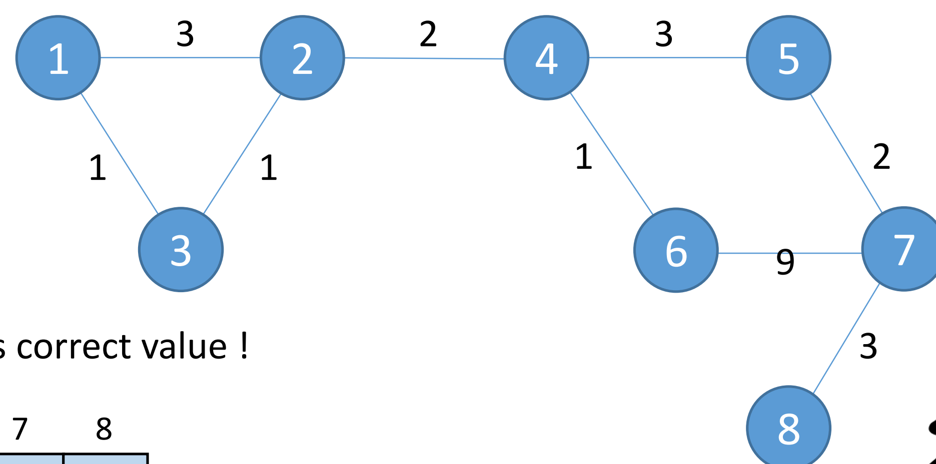
	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Dijkstra Algorithm

- Analysis
  1. Choose the white box which contains minimum value → O(V)
  2. Update another cell → O(V)
  3. Repeat !

We can move from 1 to 2 and 3

O(V^2)



Done! Because all the values in S is correct value !

	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Poscat

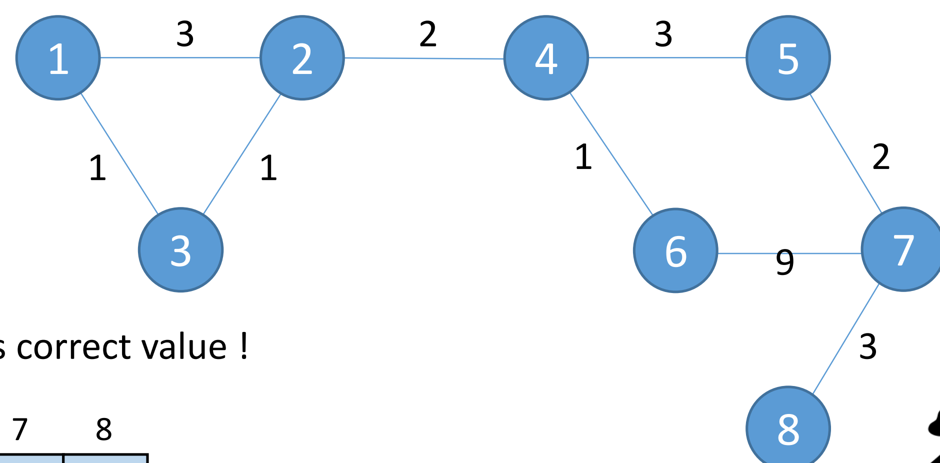
Dijkstra Algorithm

- Analysis
  1. Choose the white box which contains minimum value → O(V)
  2. Update another cell → O(V)
  3. Repeat !

We can move from 1 to 2 and 3

O(V^2)

We can improve this time complexity as **O($E \log V$)** by using priority queue



Done! Because all the values in S is correct value !

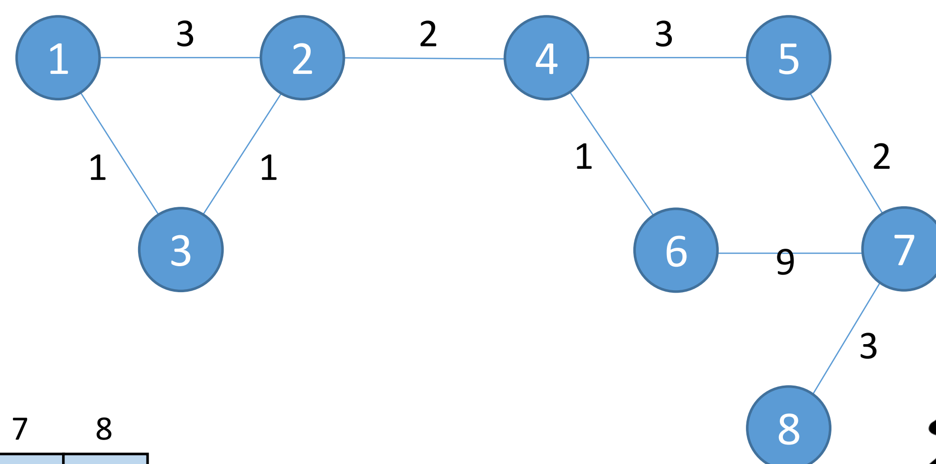
	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Dijkstra Algorithm

- Pitfall

  However, does our logic clear ?

  In other words, is it really true that white cell with minimum value never be updated ?



	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

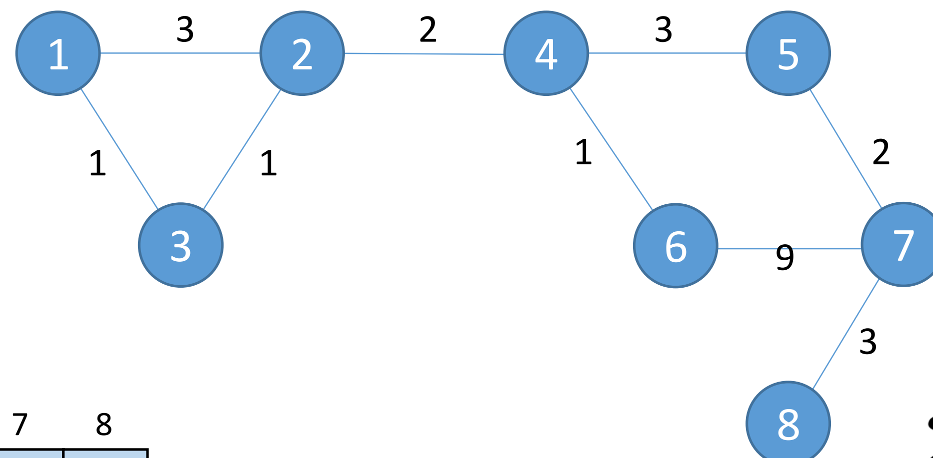
Dijkstra Algorithm

- Pitfall

However, does our logic clear ?

In other words, is it really true that white cell with minimum value never be updated ?

What if we have **negative cost** ?



	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Poscat

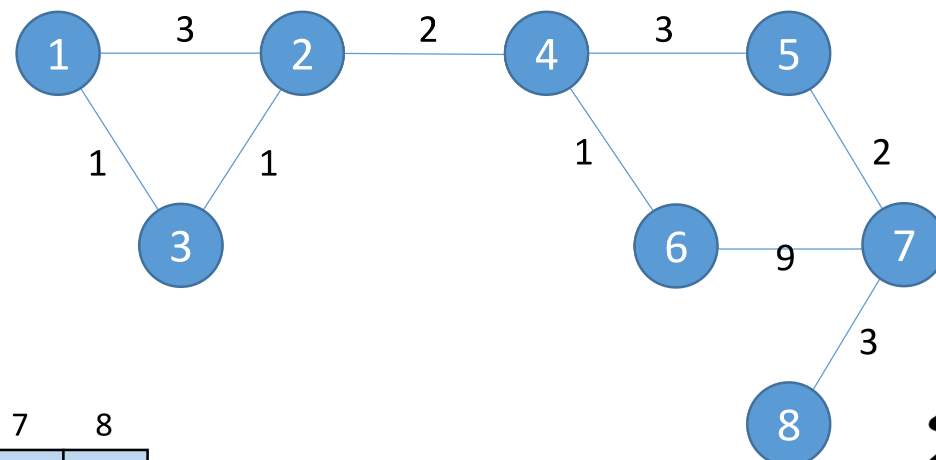
Dijkstra Algorithm

- Pitfall

However, does our logic clear ?

In other words, is it really true that white cell with minimum value never be updated ?

What if we have **negative cost** ? → **Fail !**



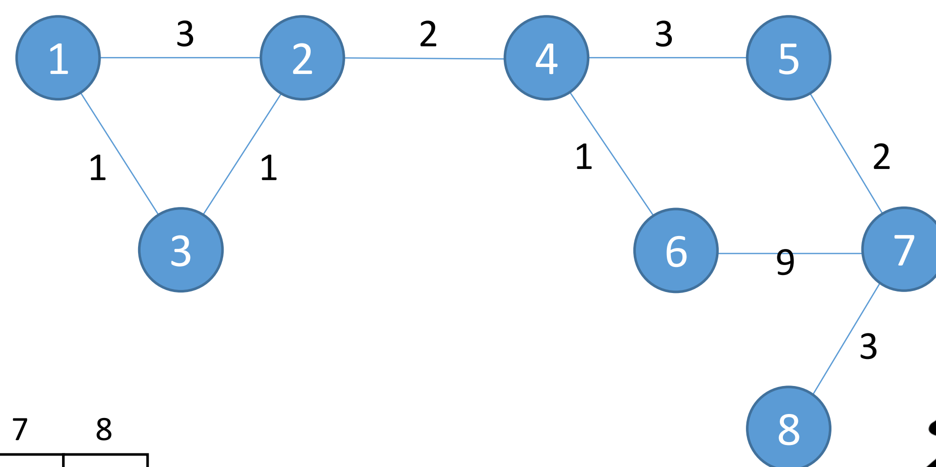
	1	2	3	4	5	6	7	8
S	0	2	1	4	7	5	9	12

Bellman Ford Algorithm

- Approach

  Iterative Approach

  Let me use $S(i)$ one more time

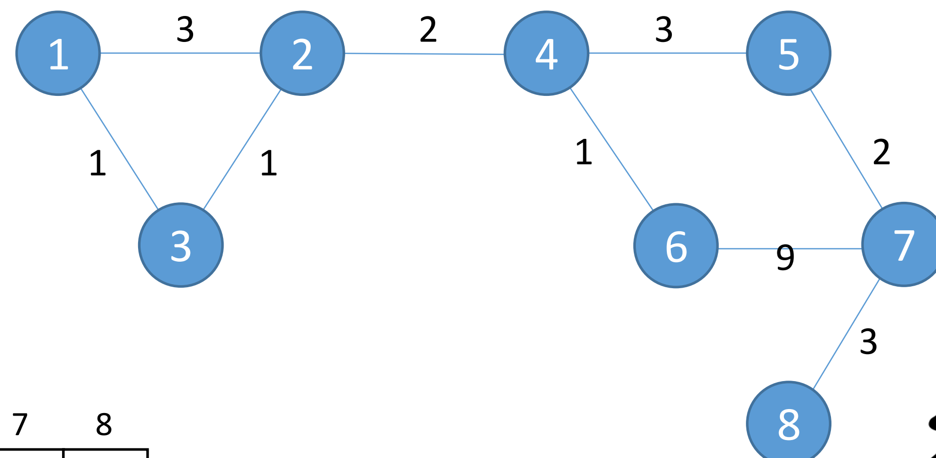


	1	2	3	4	5	6	7	8
S	0	∞	∞	∞	∞	∞	∞	∞

Bellman Ford Algorithm

- Approach

  Just update one time by using S



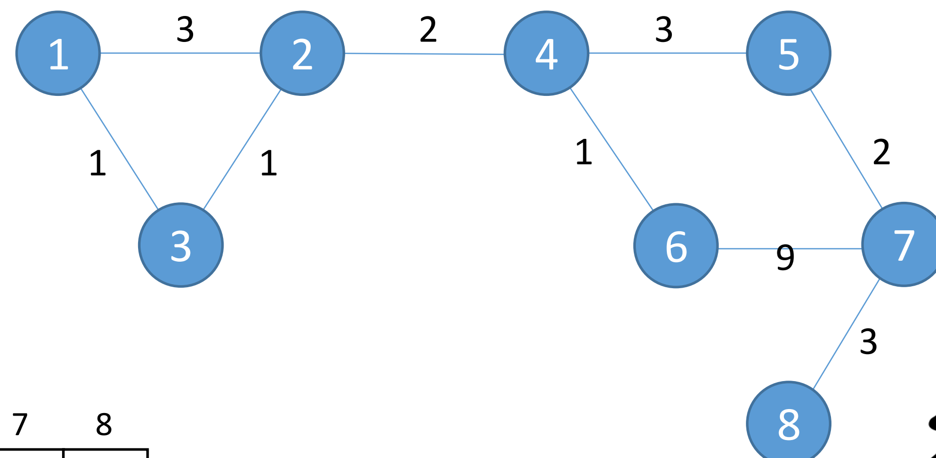
S

1	2	3	4	5	6	7	8
0	∞	∞	∞	∞	∞	∞	∞

Bellman Ford Algorithm

- Approach

  Just update one time by using S



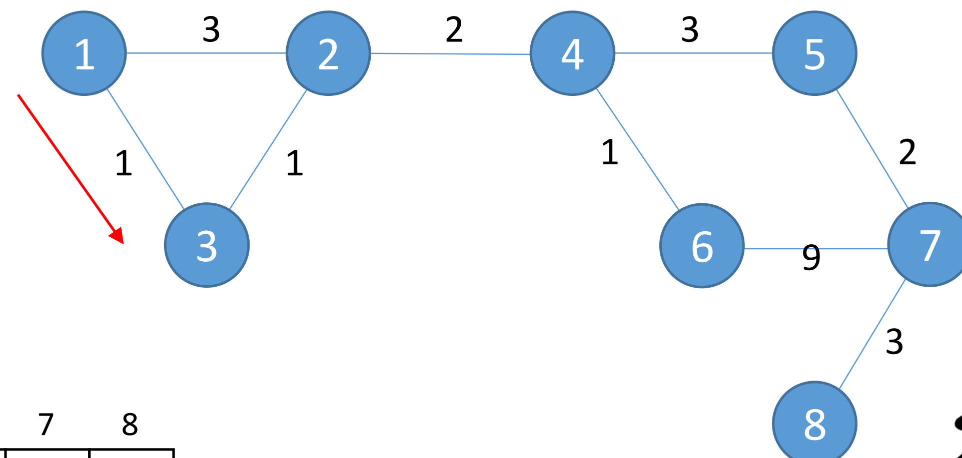
	1	2	3	4	5	6	7	8
S	0	**3**	**1**	∞	∞	∞	∞	∞

Bellman Ford Algorithm

- Approach

  Just update one time by using S

We can guarantee that we successfully calculate the **length** of shortest path **which consists of 1 edge.**
The shortest path consists of 1 edge for vertex 3. Therefore, we calculate the shortest path length of vertex 3 !



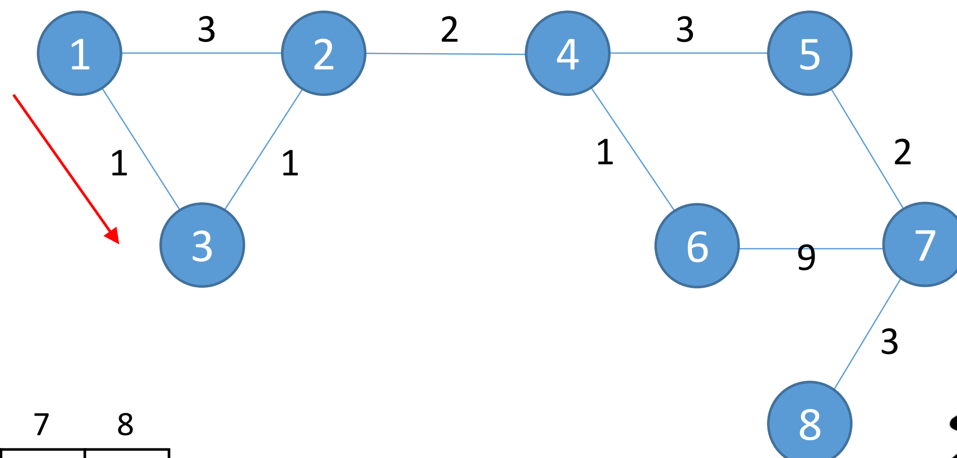
	1	2	3	4	5	6	7	8
S	0	**3**	**1**	∞	∞	∞	∞	∞

Bellman Ford Algorithm

- Approach

Just update one time by using S

However, we don't know which one is the shortest path.
In other words, we don't know whether the value of $S(3)$ is correct or not

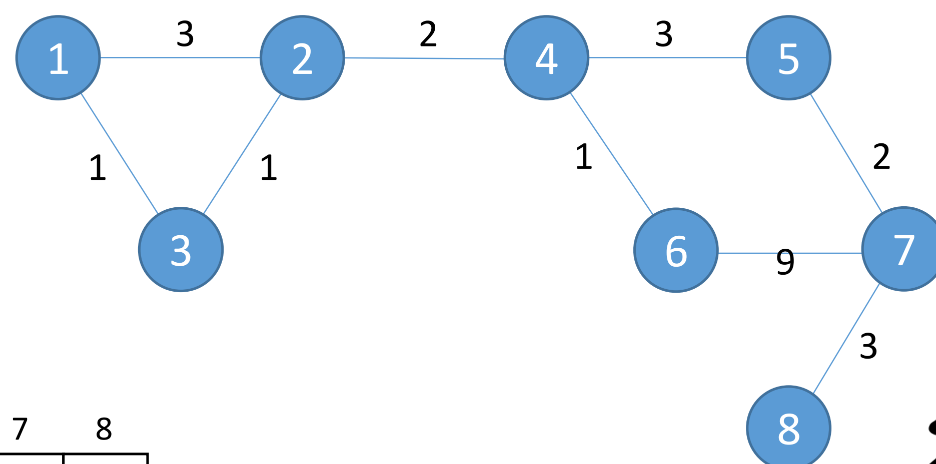


	1	2	3	4	5	6	7	8
S	0	**3**	**1**	∞	∞	∞	∞	∞

Bellman Ford Algorithm

- Approach

  Update one more time !



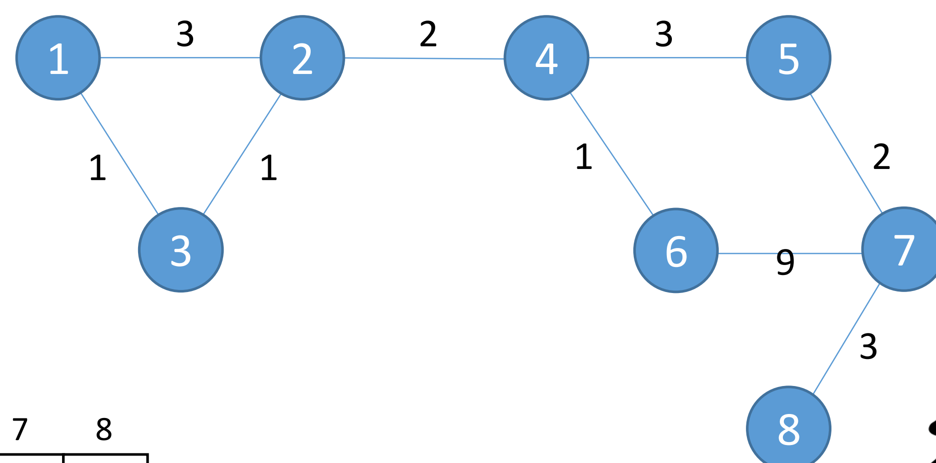
	1	2	3	4	5	6	7	8
S	0	**3**	**1**	∞	∞	∞	∞	∞

Bellman Ford Algorithm

- Approach

  Update one more time !

We can calculate the length of shortest path with 2 edges correctly !
Also, we don't know whether $S(2)$ is correct value or not

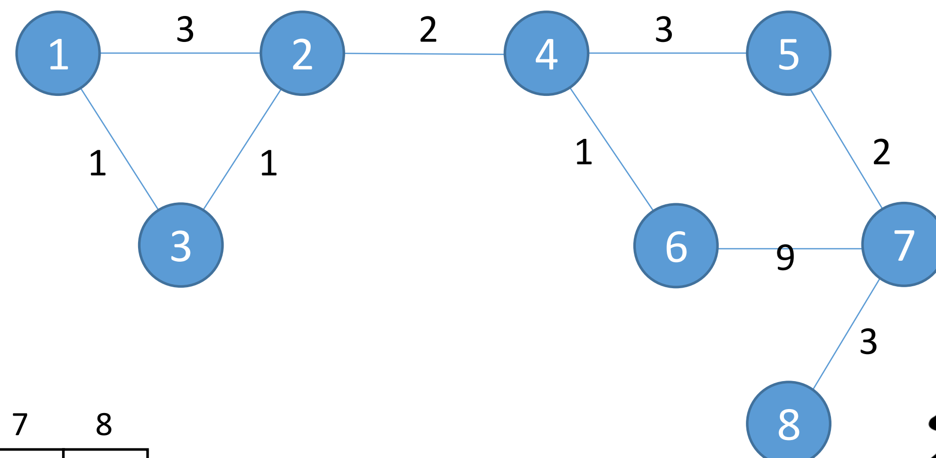


	1	2	3	4	5	6	7	8
S	0	**2**	1	**5**	∞	∞	∞	∞

Bellman Ford Algorithm

- Approach

If we update n times, then we can guarantee that
We successfully calculate the length of shortest path with n edges !



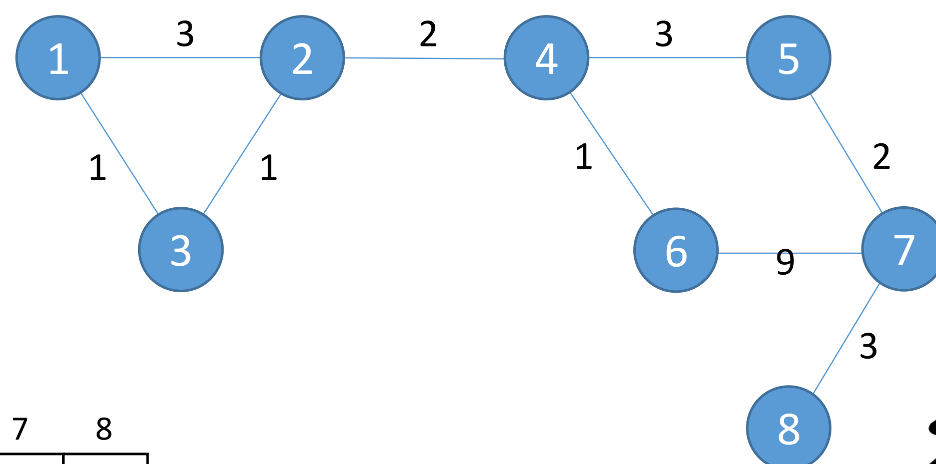
	1	2	3	4	5	6	7	8
S	0	**2**	1	**5**	∞	∞	∞	∞

Bellman Ford Algorithm

- Approach

  However, shortest path have to consist of at most (V-1) edges

  Therefore, (V-1) iteration is enough to get the correct value



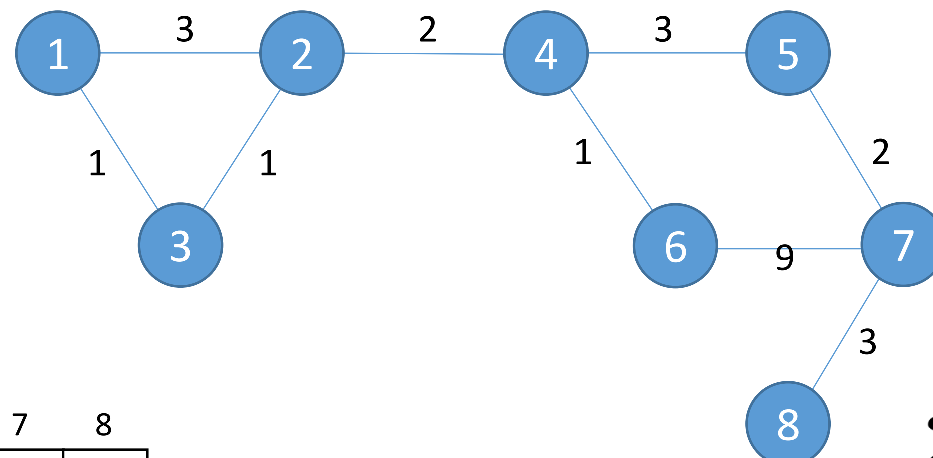
	1	2	3	4	5	6	7	8
S	0	**2**	1	**5**	∞	∞	∞	∞

Bellman Ford Algorithm

- Analysis

  Each iteration needs $O(E)$

  We need $O(V)$ iterations



	1	2	3	4	5	6	7	8
S	0	**2**	1	**5**	∞	∞	∞	∞

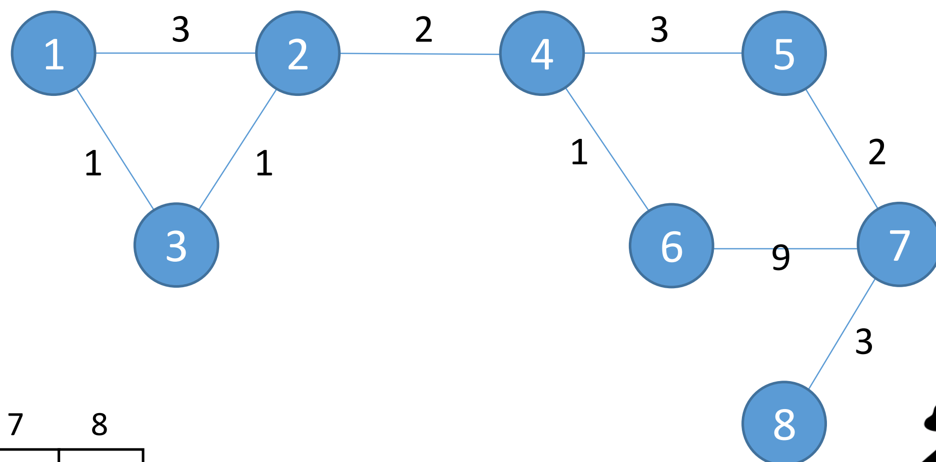
Bellman Ford Algorithm

- Analysis

  Each iteration needs $O(E)$

  We need O(V) iterations

O(VE)



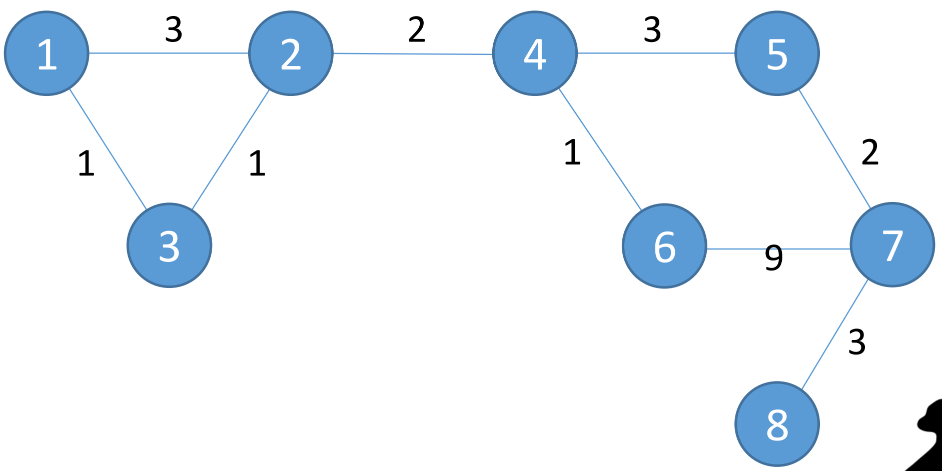
	1	2	3	4	5	6	7	8
S	0	**2**	1	**5**	∞	∞	∞	∞

Floyd Algorithm

- Approach

  Dynamic Programming Approach

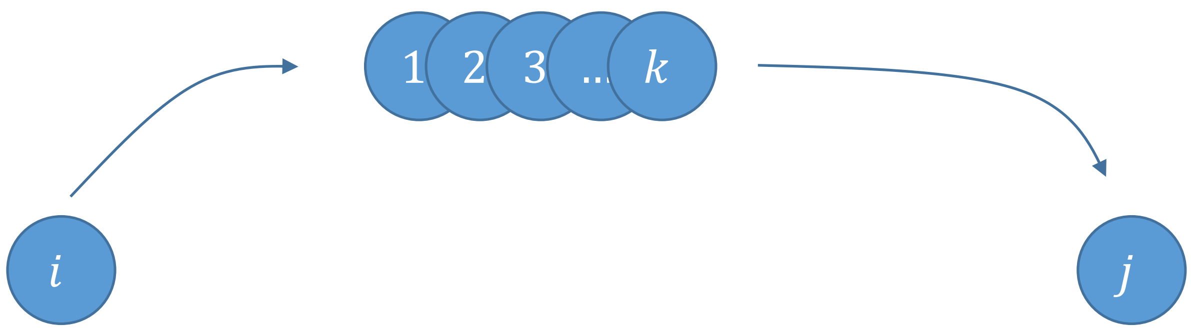
  Remember 3 steps



Floyd Algorithm

- Approach

  Let T(k, i, j) = the shortest path length from i to j
  when we use vertices from 1 to k only

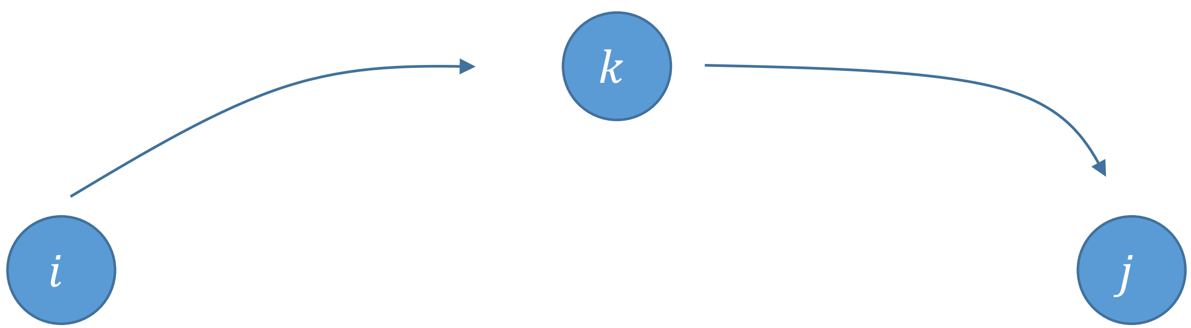


Floyd Algorithm

- Approach

Let T(k, i, j) = the shortest path length from i to j
when we use vertices from 1 to k only

Case 1. we use Vertex k

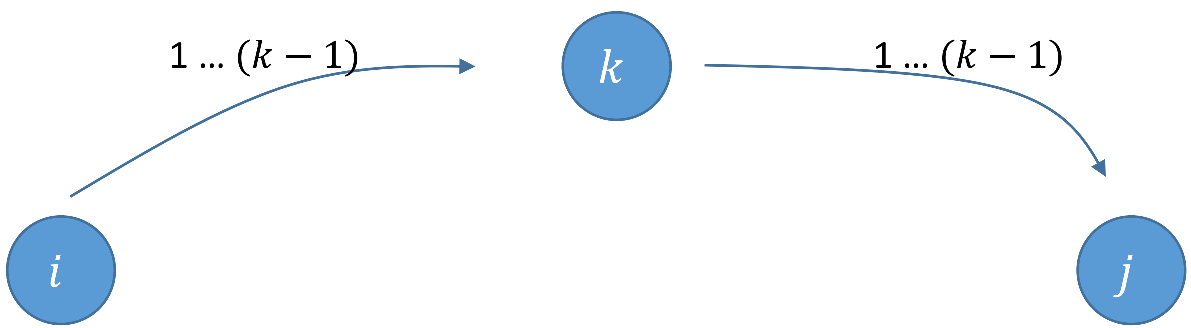


Floyd Algorithm

- Approach

Let $\mathrm{T}(k, i, j)$ = the shortest path length from i to j
when we use vertices from 1 to k only

Case 1. we use Vertex k



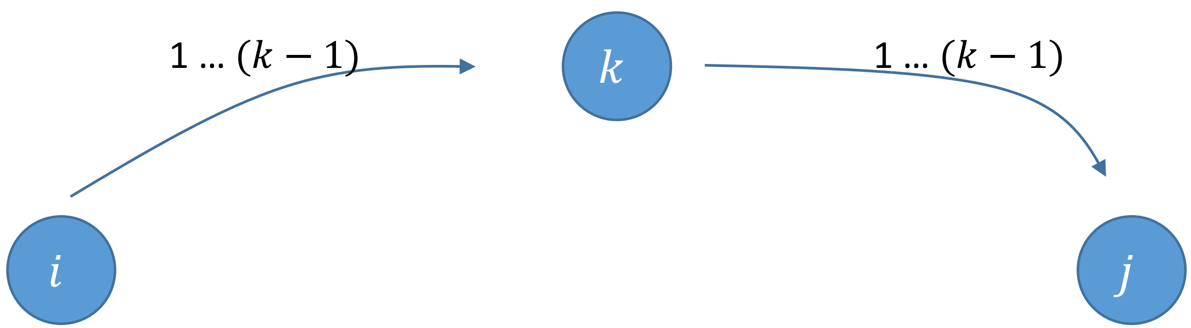
Then we can use vertex 1 ~ $(k-1)$ to move from i to k
Also, we can use same vertices to move from k to j

Floyd Algorithm

- Approach

Let T(k, i, j) = the shortest path length from i to j
when we use vertices from 1 to k only

Case 1. we use Vertex k



$\therefore$ T(k, i, j) = T$(k-1, i, k)$ + T$(k-1, k, j)$

Floyd Algorithm

- Approach

  Let T(k, i, j) = the shortest path length from i to j
  when we use vertices from 1 to k only

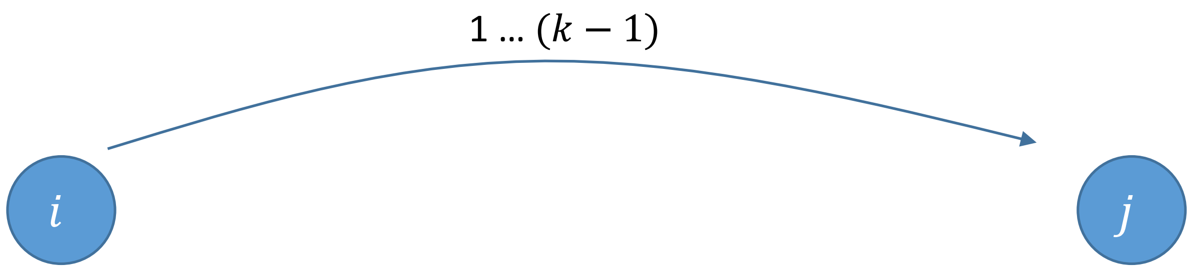
  Case 2. we don't use vertex k

Floyd Algorithm

- Approach

  Let $\mathrm{T}(k, i, j)$ = the shortest path length from i to j when we use vertices from 1 to k only

  Case 2. we don't use vertex k



$$\mathrm{T}(k, i, j) = \mathrm{T}(k-1, i, j)$$

Floyd Algorithm

- Approach

  Let $\mathrm{T}(k, i, j)$ = the shortest path length from i to j
  when we use vertices from 1 to k only

  $$\therefore \mathrm{T}(\mathrm{k}, \mathrm{i}, \mathrm{j}) = \min\,(\,\mathrm{T}(\mathrm{k}-1, \mathrm{i}, \mathrm{j}), \mathrm{T}(\mathrm{k}-1, \mathrm{i}, \mathrm{k}) + \mathrm{T}(\mathrm{k}-1, \mathrm{k}, \mathrm{j}))$$

Floyd Algorithm

- Analysis

Let $\mathrm{T}(k, i, j)$ = the shortest path length from i to j when we use vertices from 1 to k only

$$\therefore \mathrm{T(k, i, j) = min\ (\ T(k-1, i, j), T(k-1, i, k) + T(k-1, k, j))}$$

Poscat

Floyd Algorithm

- Analysis

Let T(k, i, j) = the shortest path length from i to j
when we use vertices from 1 to k only

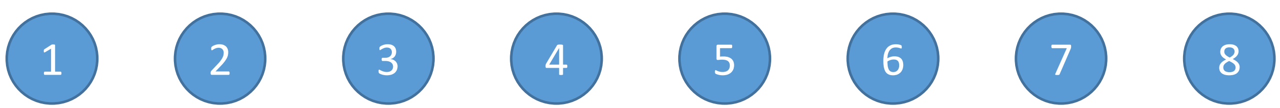
$$\therefore \mathrm{T(k,i,j) = min\ (\ T(k-1,i,j), T(k-1,i,k) + T(k-1,k,j))}$$

O(n^3)

Disjoint Set

- Problem

Initially, there are n groups. You can merge two group into one.
Determine whether two elements is in the same group or not

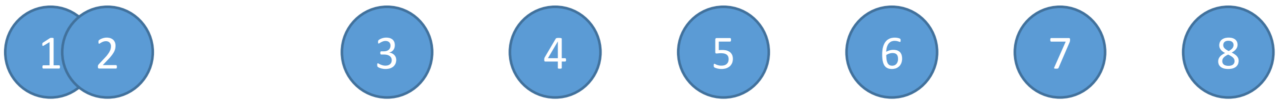


Disjoint Set

- Problem

  Initially, there are n groups. You can merge two group into one.

  Determine whether two elements is in the same group or not



Merge !

Disjoint Set

- Problem

  Initially, there are n groups. You can merge two group into one.

  Determine whether two elements is in the same group or not

Merge !

Merge !

Disjoint Set

- Problem

  Initially, there are n groups. You can merge two group into one.
  Determine whether two elements is in the same group or not

Disjoint Set

- Problem

  Initially, there are n groups. You can merge two group into one.

  Determine whether two elements is in the same group or not

Disjoint Set

- Problem

Initially, there are n groups. You can merge two group into one.

Determine whether two elements is in the same group or not

Determine whether vertex 1 and 3 is in the same group

Disjoint Set

- Problem

  Initially, there are n groups. You can merge two group into one.
  Determine whether two elements is in the same group or not

Determine whether vertex 1 and 3 is in the same group **NO**

Disjoint Set

- Approach

How can we represent a group ?

Disjoint Set

- Approach

How can we represent a group ? Using array is bad

Disjoint Set

- Approach

  How can we represent a group ? Using array is bad

  We use tree structure !

Disjoint Set

- Approach

How can we represent a group ? Using array is bad

We use tree structure !

Disjoint Set

- Approach

How can we determine that two vertices are in the same group ?

Disjoint Set

- Approach

How can we determine that two vertices are in the same group ?

by comparing their root node !

Disjoint Set

- Approach

How can we determine that two vertices are in the same group ?

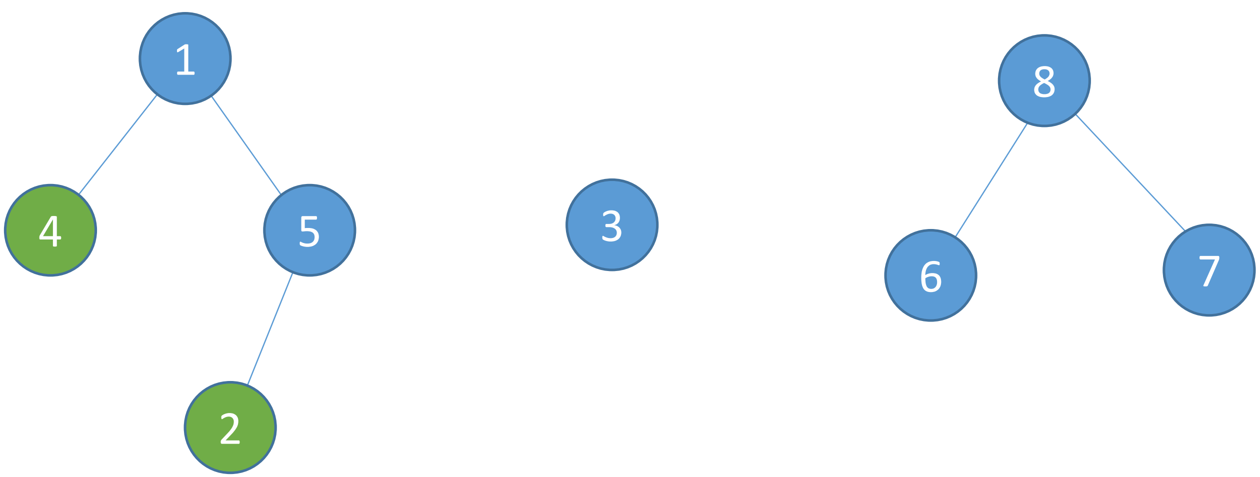
by comparing their root node !

Disjoint Set

- Approach

How can we determine that two vertices are in the same group ?

by comparing their root node !



Same !

Disjoint Set

- Approach

  How can we determine that two vertices are in the same group ?

  by comparing their root node !

Disjoint Set

- Approach

  How can we determine that two vertices are in the same group ?

  by comparing their root node !

No !

Disjoint Set

- Approach

  Finding their root is quite simple. Just follow their parent node.

Disjoint Set

- Approach

Finding their root is quite simple. Just follow their parent node.

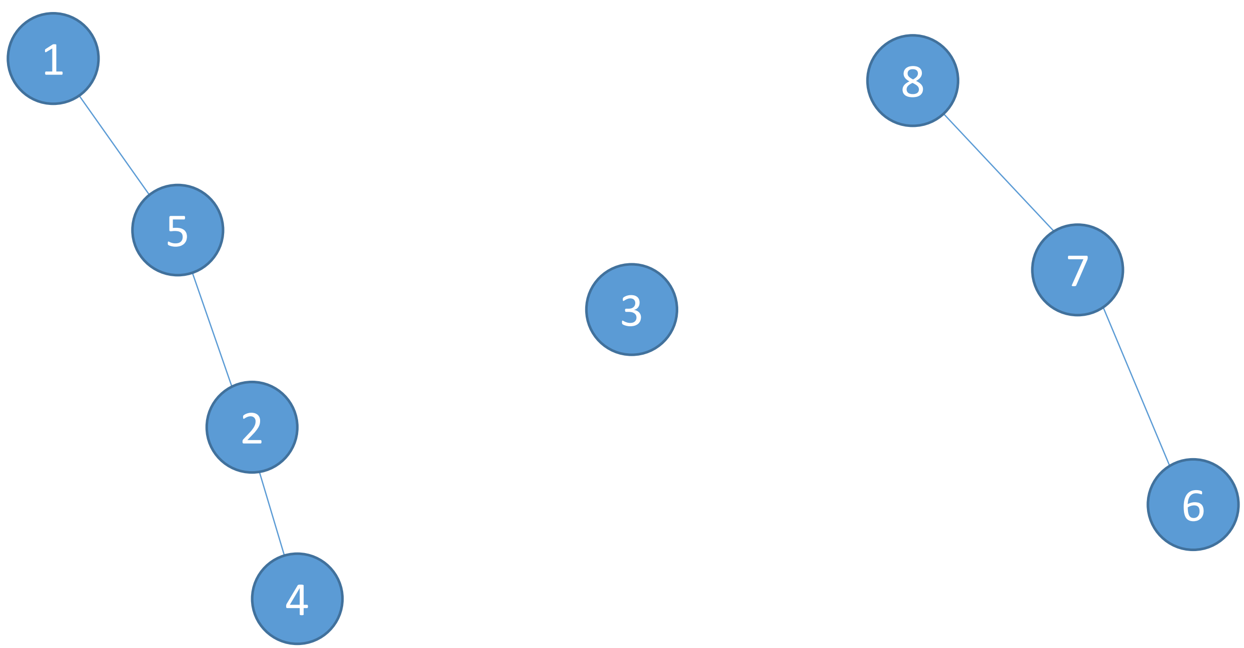
Is it efficient ?

Disjoint Set

- Approach

  Finding their root is quite simple. Just follow their parent node.

  Is it efficient ?

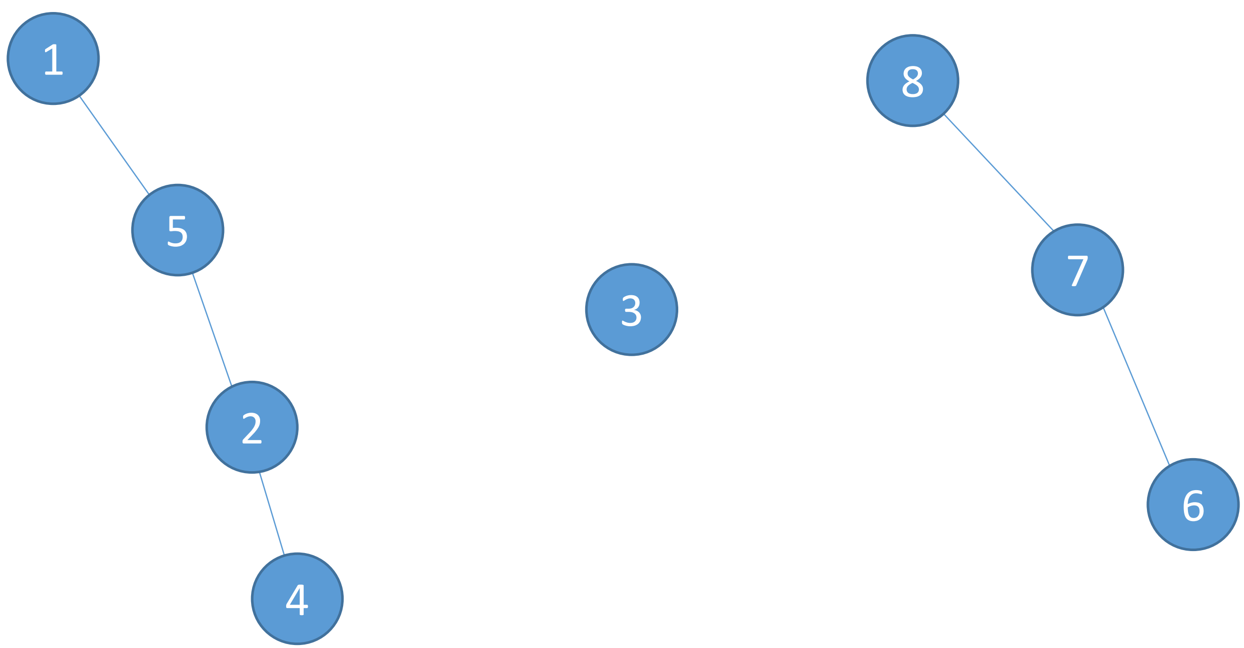


Disjoint Set

- Approach

Finding their root is quite simple. Just follow their parent node.

Is it efficient ? If the shape of tree is bad, it takes O(n)

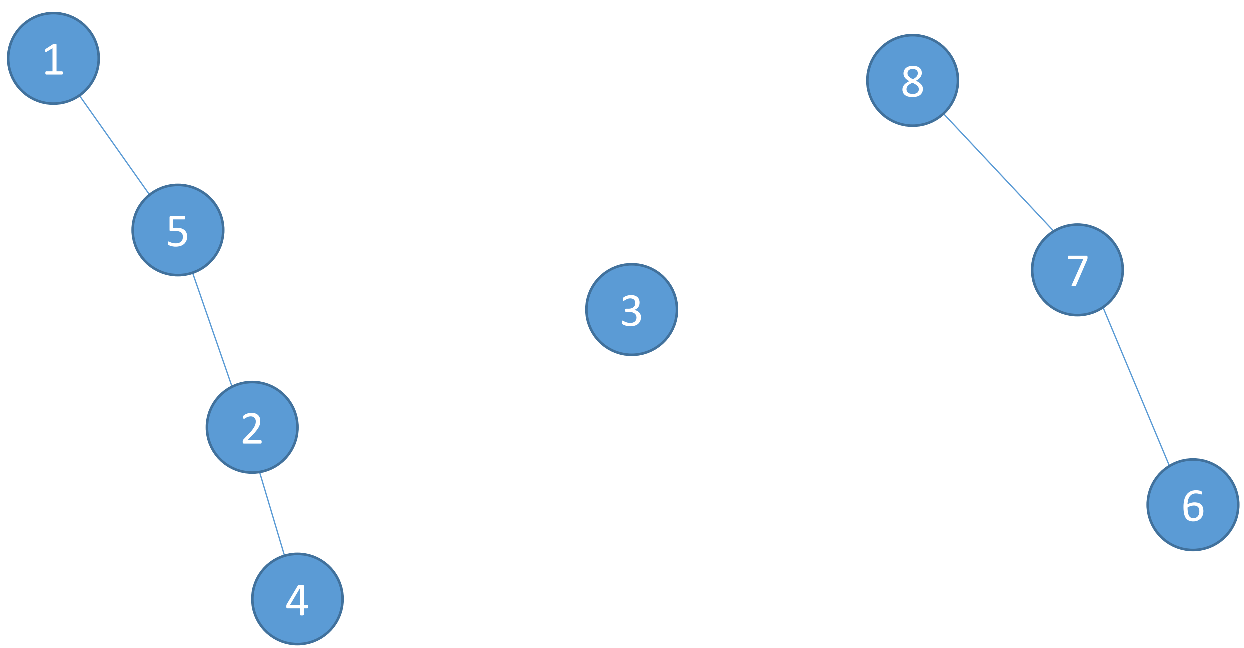


Path Compression

- Approach

However, we can “Compress” tree when we find the root node !

The idea is that the only requirement is to know **root node**

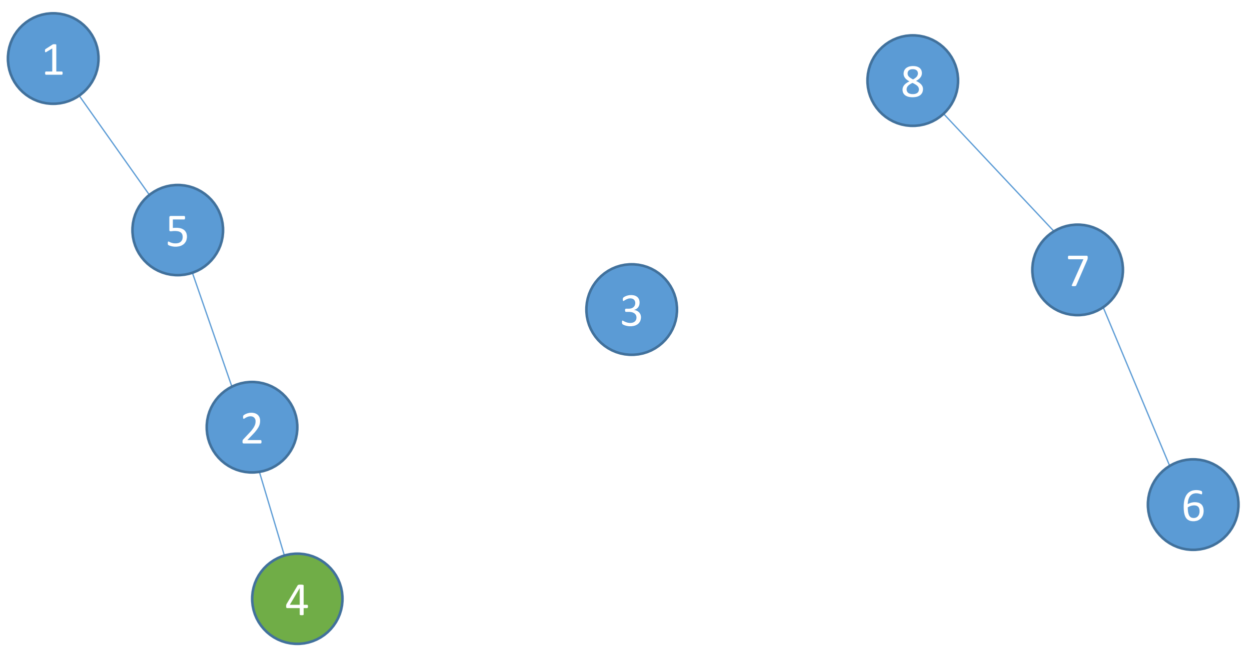


Path Compression

- Approach

However, we can “Compress” tree when we find the root node !

The idea is that the only requirement is to know **root node**

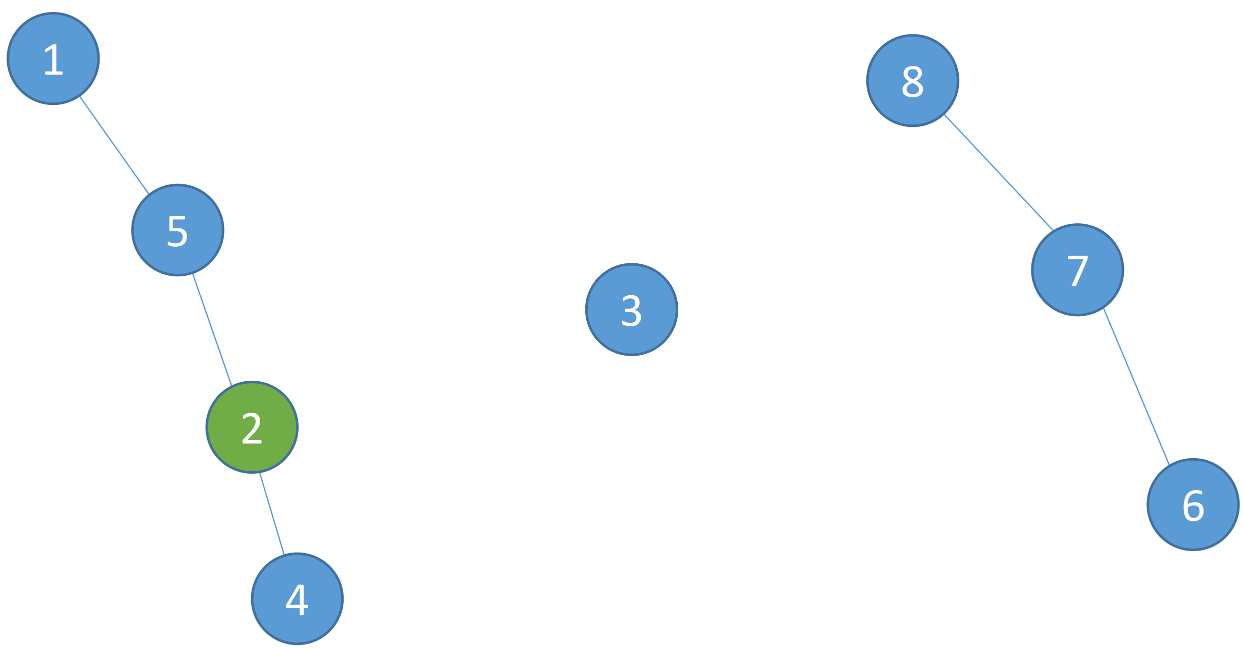


Path Compression

- Approach

  However, we can "Compress" tree when we find the root node !

  The idea is that the only requirement is to know **root node**

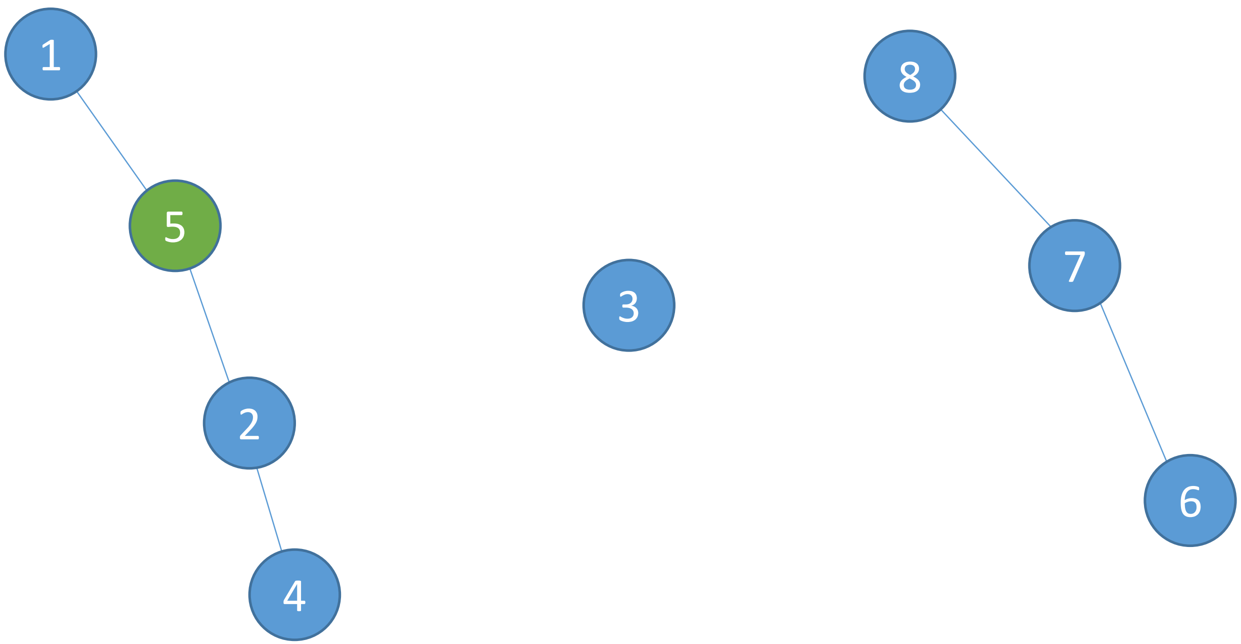


Path Compression

- Approach

  However, we can "Compress" tree when we find the root node !

  The idea is that the only requirement is to know **root node**

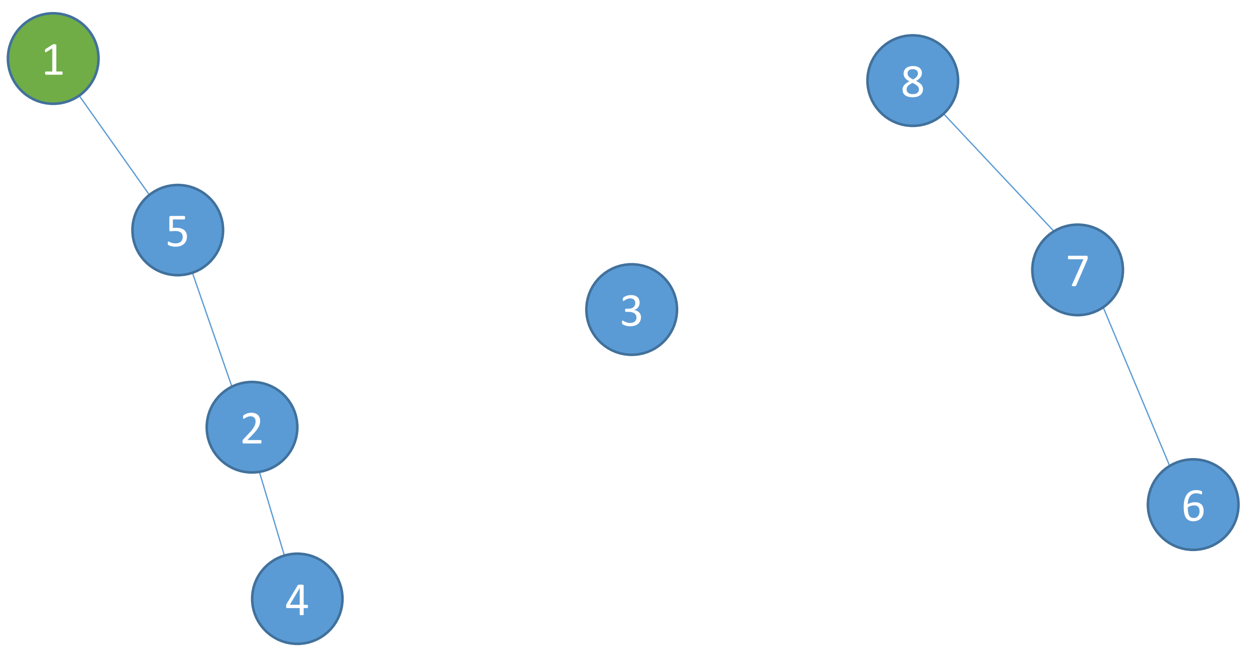


Path Compression

- Approach

However, we can “Compress” tree when we find the root node !

The idea is that the only requirement is to know **root node**

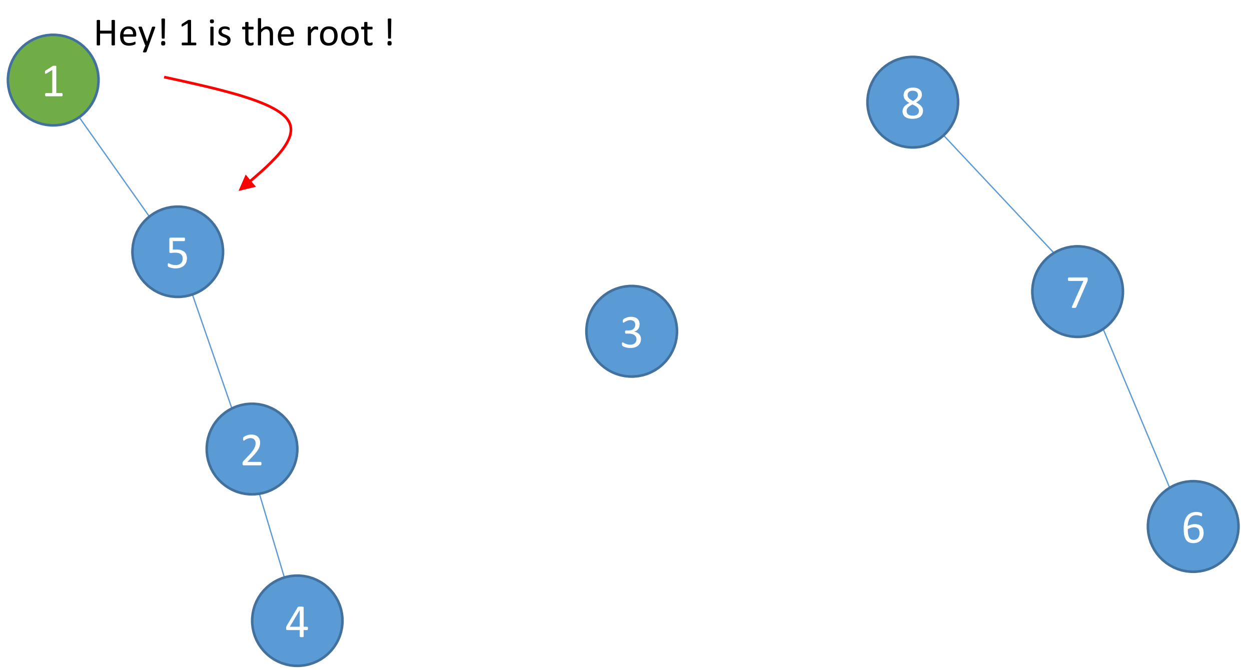


Path Compression

- Approach

However, we can "Compress" tree when we find the root node !

The idea is that the only requirement is to know **root node**

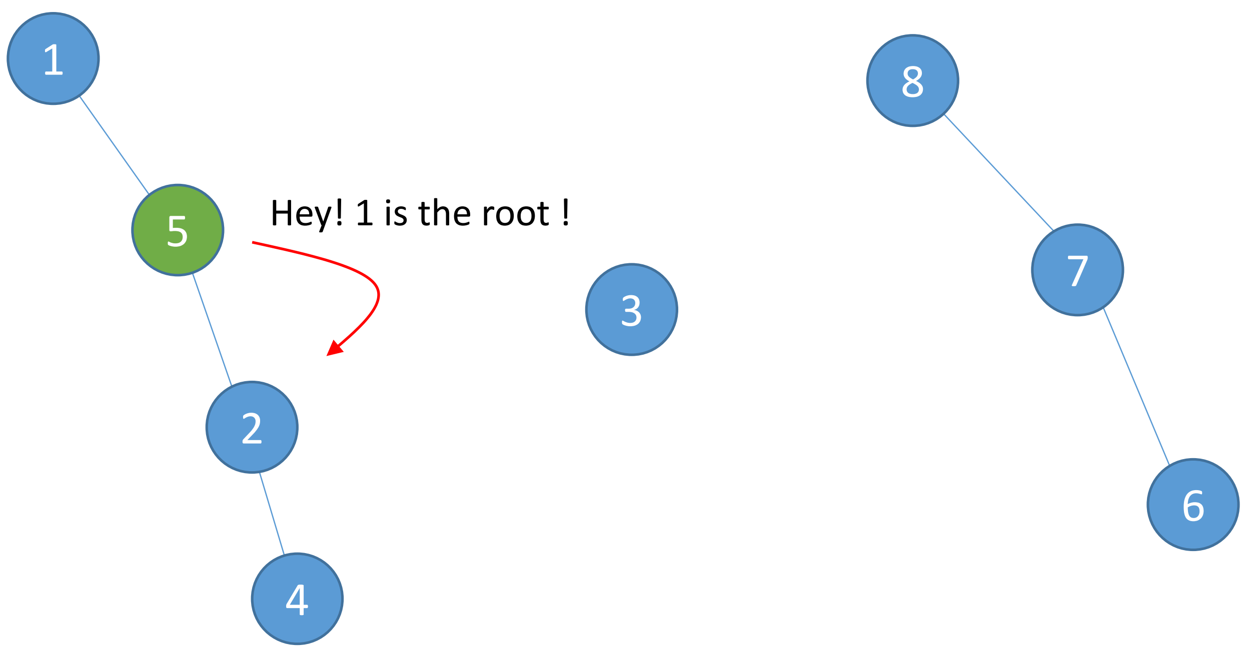


Path Compression

- Approach

However, we can "Compress" tree when we find the root node !

The idea is that the only requirement is to know **root node**

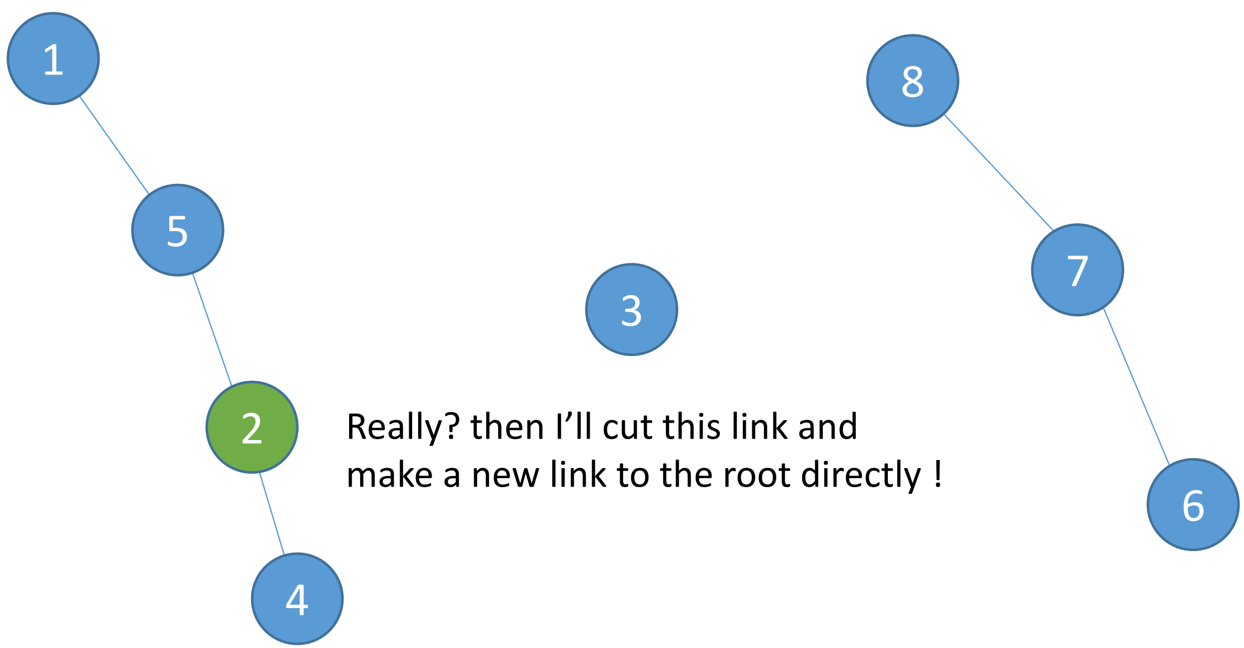


Path Compression

- Approach

  However, we can "Compress" tree when we find the root node !

  The idea is that the only requirement is to know **root node**

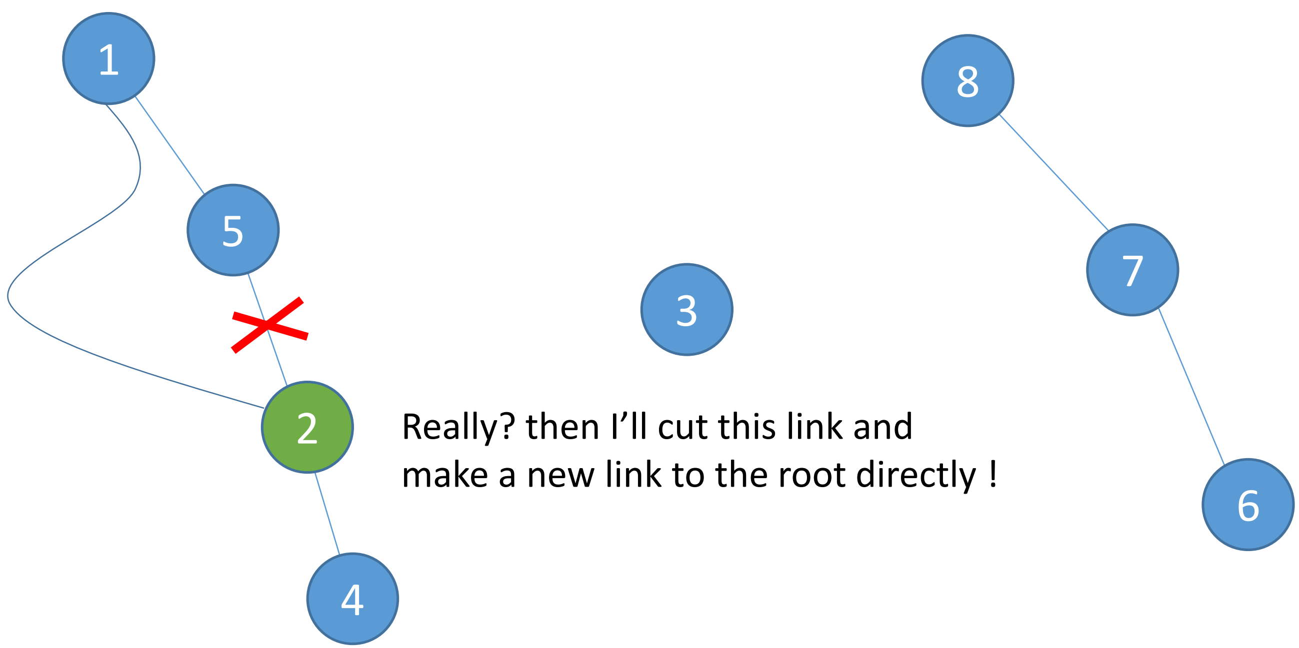


Path Compression

- Approach

  However, we can “Compress” tree when we find the root node !

  The idea is that the only requirement is to know **root node**



Poscat

Path Compression

- Approach

  However, we can "Compress" tree when we find the root node !

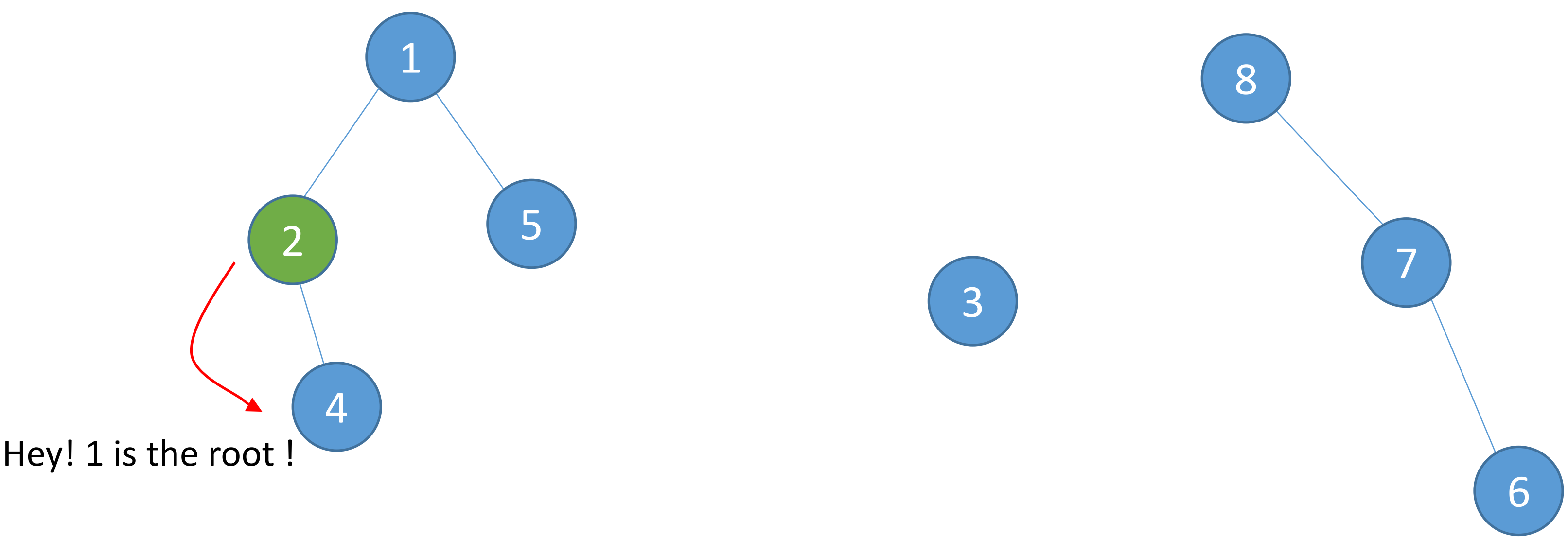
  The idea is that the only requirement is to know **root node**

Path Compression

- Approach

However, we can “Compress” tree when we find the root node !

The idea is that the only requirement is to know **root node**

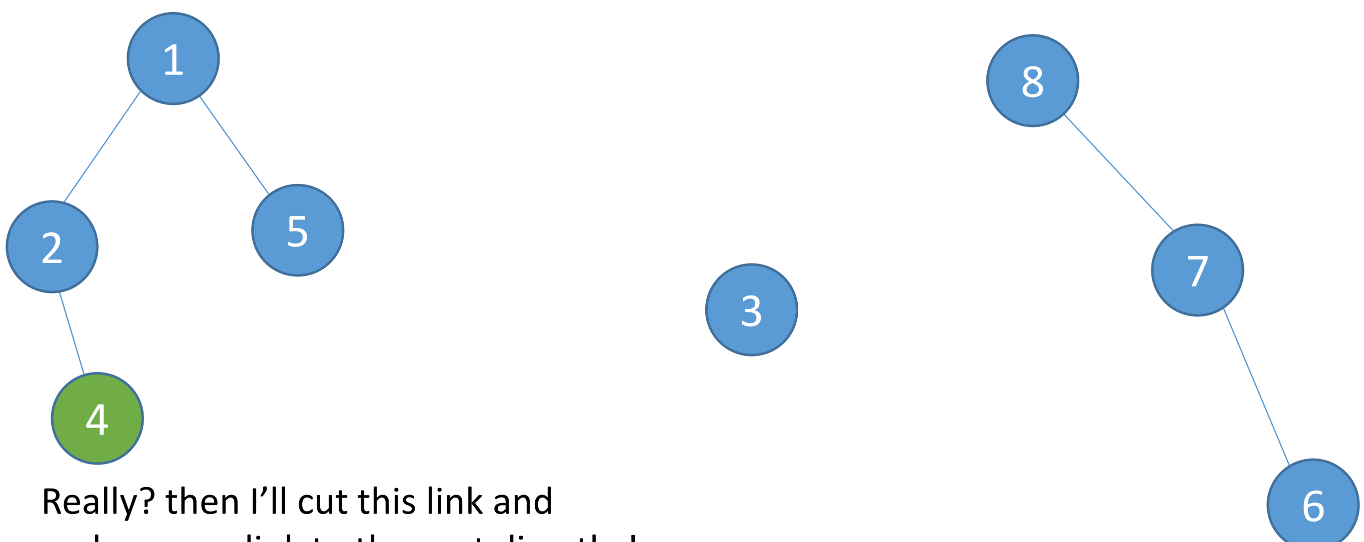


Path Compression

- Approach

  However, we can “Compress” tree when we find the root node !

  The idea is that the only requirement is to know **root node**



Really? then I’ll cut this link and
make a new link to the root directly !

Poscat

Path Compression

- Approach

However, we can "Compress" tree when we find the root node !

The idea is that the only requirement is to know **root node**

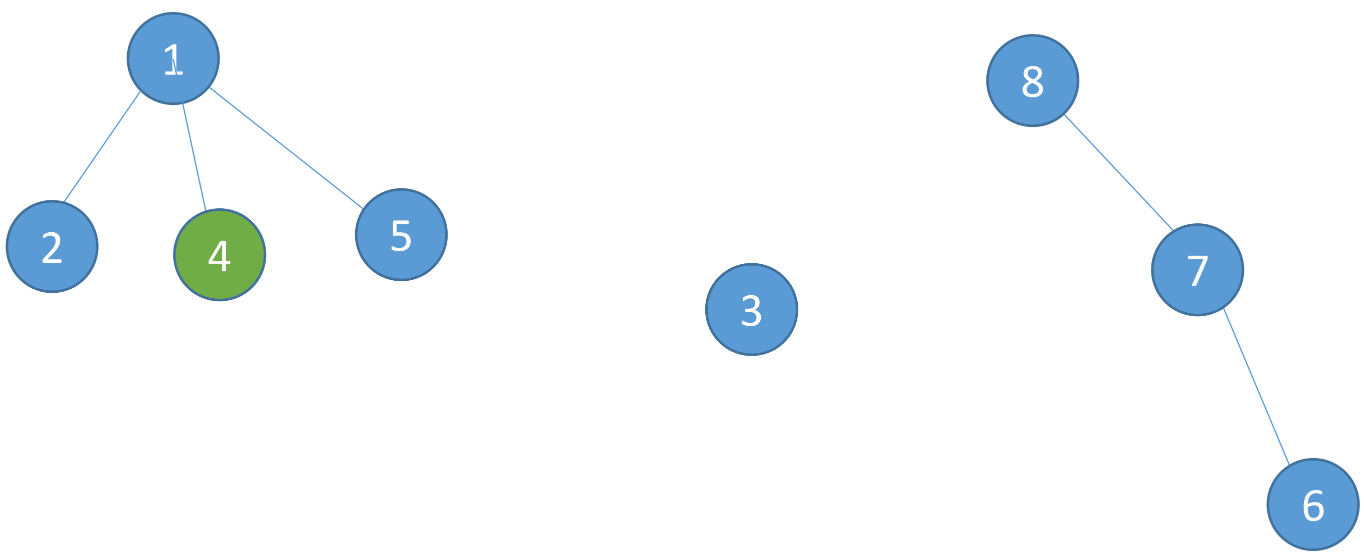
Really? then I'll cut this link and
make a new link to the root directly !

Path Compression

- Approach

  However, we can “Compress” tree when we find the root node !

  The idea is that the only requirement is to know **root node**

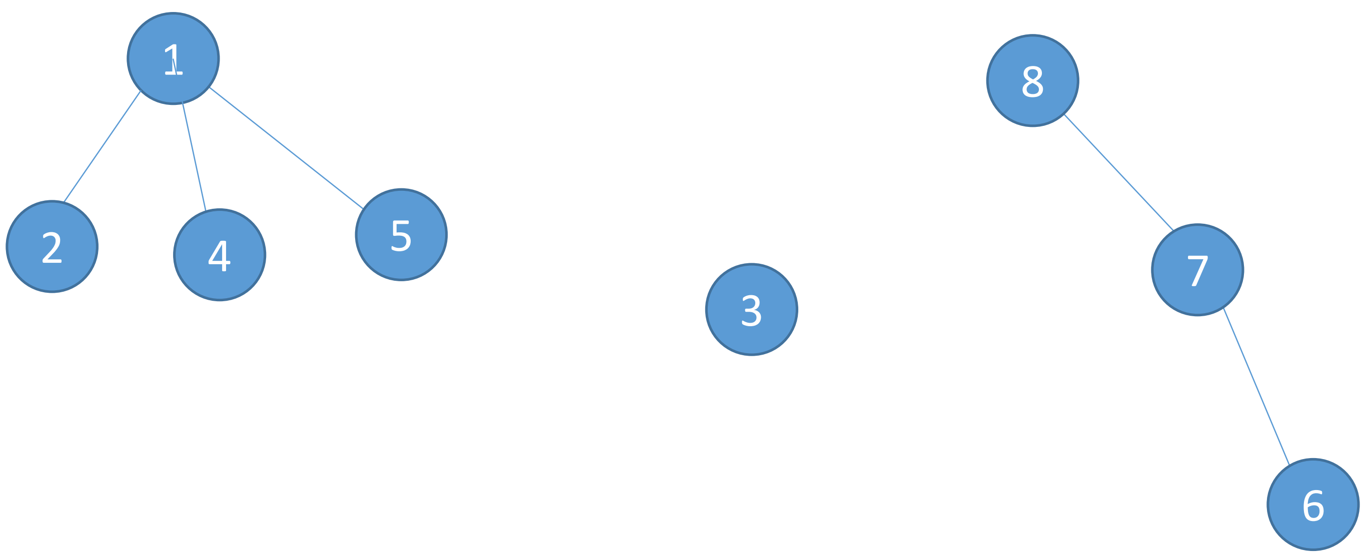


Path Compression

- Approach

However, we can "Compress" tree when we find the root node !

The idea is that the only requirement is to know **root node**

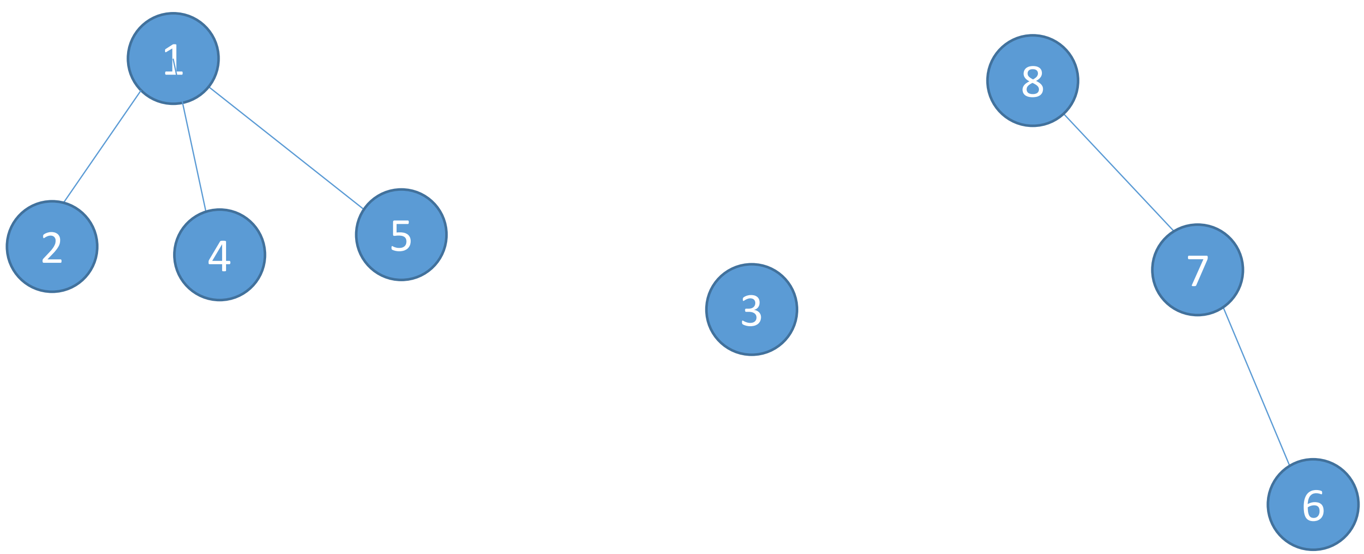


Path Compression

- Approach

We perform two things simultaneously

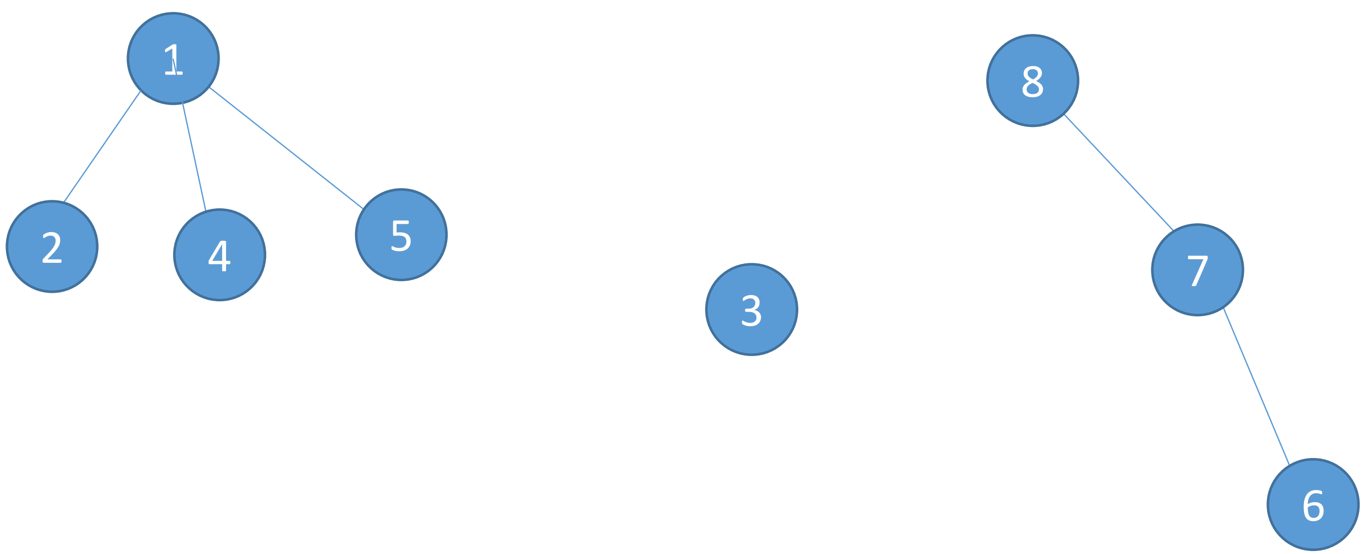
1. Find a root
2. Compress the tree !



Path Compression

- Approach

Then, How long does it take ?



Path Compression

- Approach

  Then, How long does it take ?

  Amazingly, it takes just constant time. i.e. O(1)

  Analysis is very complex. We don't discuss it.