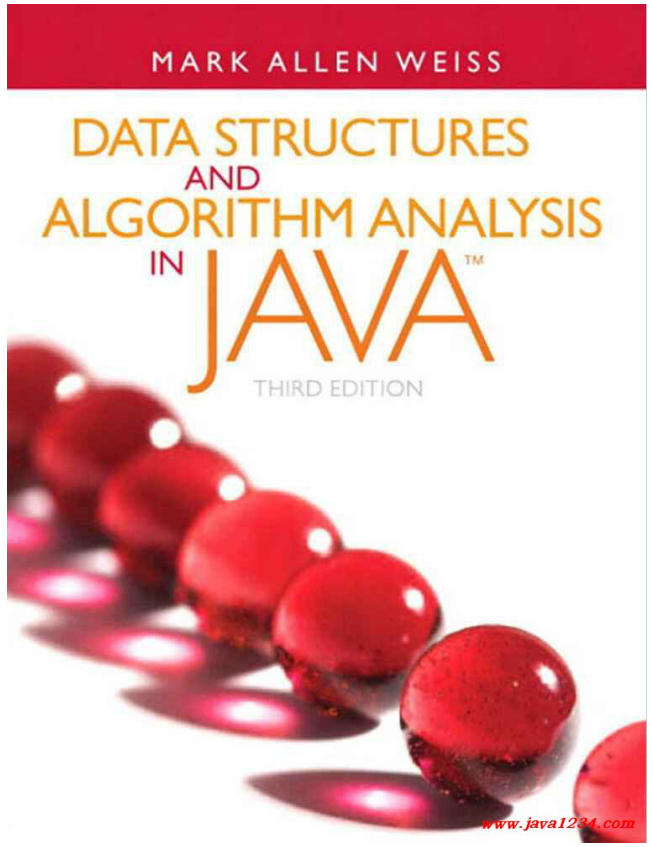
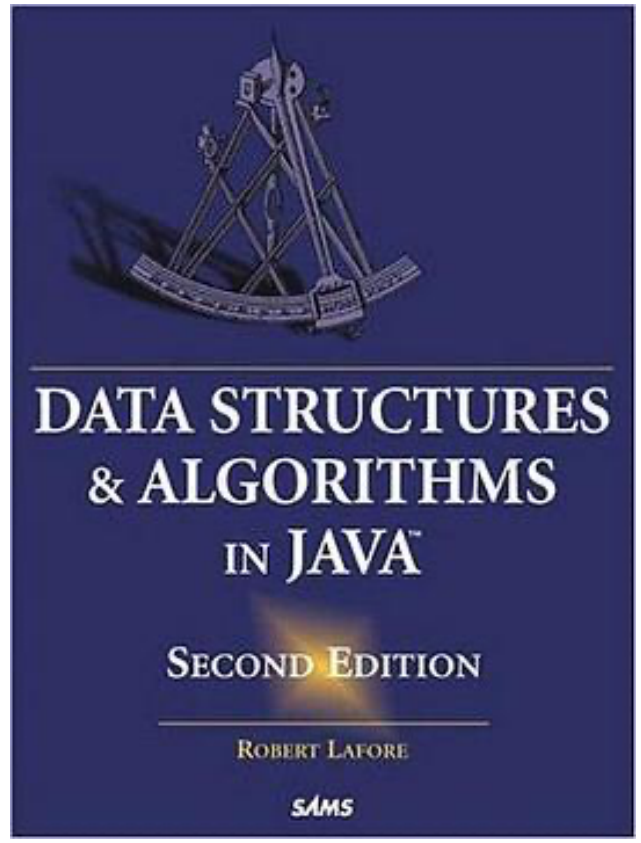
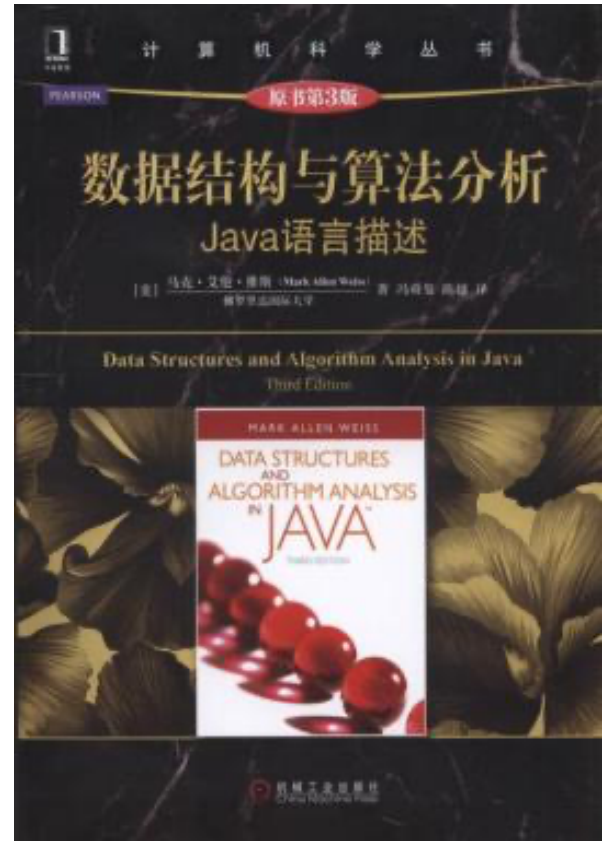
Topic 19 –
Shortest Path Algorithms







- **Types of Shortest Path Problems**
- **Relaxation**
- **Bellman-Ford Algorithm**
- **Dijkstra's Algorithm**
- **Floyd-Warshall Algorithm**

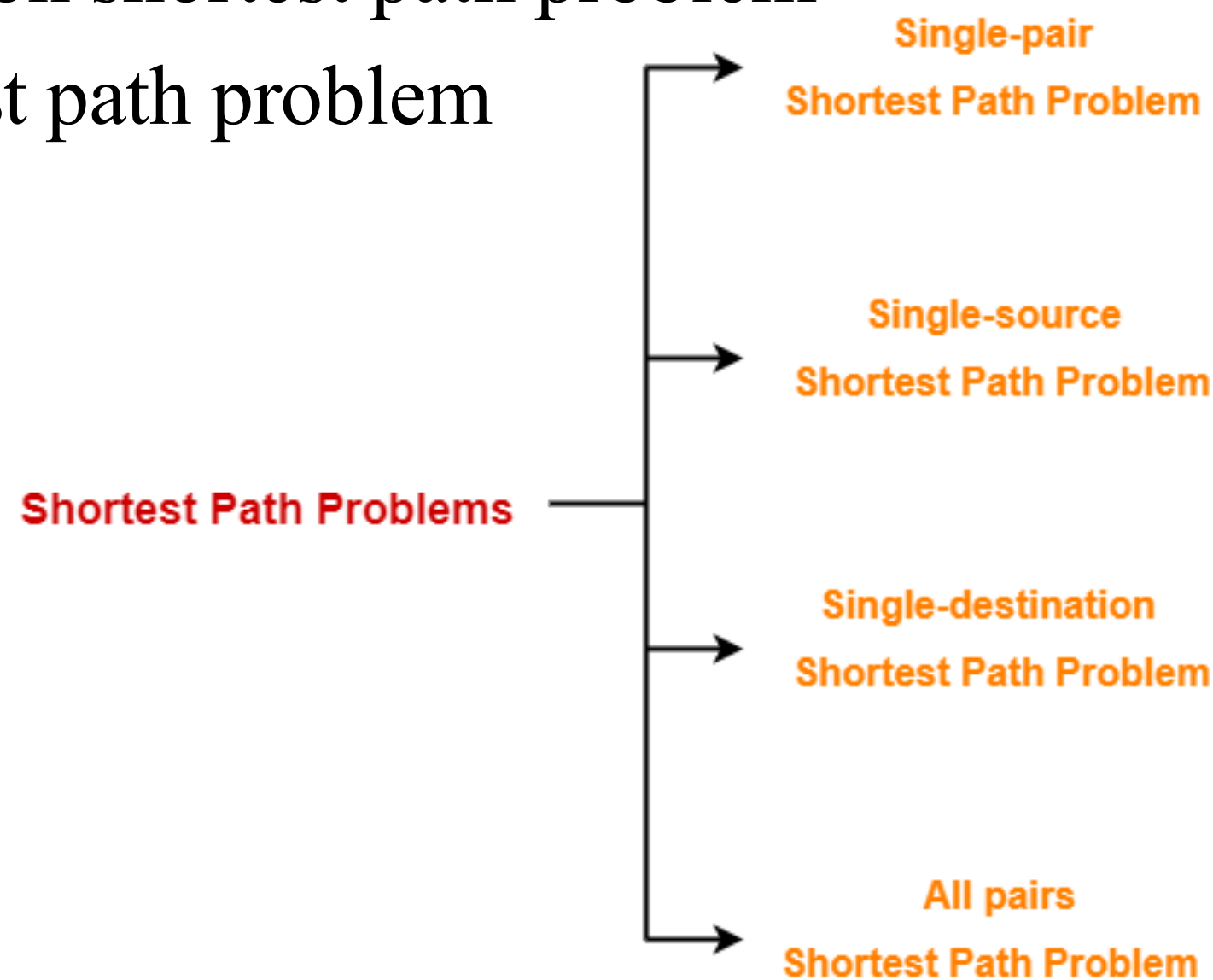
Shortest Path Problem

- Shortest path problem is a problem of finding the shortest path(s) between vertices of a given graph.
- Shortest path between two vertices is a path that has the least cost as compared to all other existing paths.

- **<u>Applications</u>**
  - Google Maps
  - Road Networks
  - Logistics Research

Types of Shortest Path Problem

- Single-pair shortest path problem
- Single-source shortest path problem
- Single-destination shortest path problem
- All pairs shortest path problem



Shortest Path Problem

- **Single-Pair Shortest Path Problem**
  - It is a shortest path problem where the shortest path between **a given pair of vertices** is computed.
  - A* Search Algorithm is a famous algorithm used for solving single-pair shortest path problem.

Shortest Path Problem

- **Single-Source Shortest Path Problem**
  - It is a shortest path problem where the shortest path from **a given source vertex to all other remaining vertices** is computed.
  - Dijkstra's Algorithm and Bellman Ford Algorithm are the famous algorithms used for solving single-source shortest path problem.

Shortest Path Problem

- **Single-Destination Shortest Path Problem**
  - It is a shortest path problem where the shortest path from **all the vertices to a single destination vertex** is computed.
  - By reversing the direction of each edge in the graph, this problem reduces to single-source shortest path problem.
  - Dijkstra's Algorithm is a famous algorithm adapted for solving single-destination shortest path problem.

Shortest Path Problem

- **All Pairs Shortest Path Problem**
  - It is a shortest path problem where the shortest path between **every pair of vertices** is computed.
  - Floyd-Warshall Algorithm and Johnson's Algorithm are the famous algorithms used for solving All pairs shortest path problem.

- **Types of Shortest Path Problems**
- **Relaxation**
- **Bellman-Ford Algorithm**
- **Dijkstra's Algorithm**
- **Floyd-Warshall Algorithm**

Optimal Substructure Property

- Consider a shortest path P between vertices u and v in G. Let z be a vertex on P.
- The portion of P between u and z is a shortest path between u and z.
- The portion of P between z and v is a shortest path between z and v.

Cycles and Shortest Paths

- All edges have **positive weights**:
  - Shortest paths cannot contain cycles.
- All edges have **non-negative weights**.
  - 0-cycles can occur in shortest paths but can be removed without changing the length.
- Some edges have negative weigths:
  - If negative weight cycles are in the graph, shortest path problem is not well-defined.
  - Otherwise, we are back in one of the above two cases.
- Conclusion: Shortest paths in graphs with n vertices have at most n-1 edges.

Triangle Inequality for Shortest Paths

原共

好好第一段和第二段的和

- Claim:
  - $\delta(s,v) \leq \delta(s,u) + w(u,v)$ for all $(u,v) \in E$
  - *Note.* $\delta(s,v)$ is the total weight of the shortest path between s and v

$s \to u$ 最短 + $u \to v$ 最短至少是 $s \to v$ 最短

- Proof:
  - The shortest path from s to u followed by the single edge path from u to v is a path from s to v.
  - The length of this path must be at least equal to the length of the shortest path from s to v.

Shortest Path Algorithms - Intuition

- A path from s to every vertex v will be maintained.
  - $d[v]$ = weight of a path **from s to v**.
  - $p[v]$ = predecessor of v on a path from s to v.
- Initially
  - $d[s] = 0$,
  - $d[v] = \infty$ for all $v \neq s$,
  - $p[v]$ = null for all $v \in G$
- Suppose that an edge (u,v) exists such that $d[v] > d[u] + w(u,v)$
- Relaxation:
  - Update $p[v]$ and reduce $d[v]$.
  - Repeat.

Initialization

- INITIALIZE-SINGLE-SOURCE(s)
  - $d[s] = 0$;
  - for each vertex v in $V - \{s\}$ do
    - $d[v] = \infty$;
  - for each vertex v in V do
    - $p[v]$ = null;

Relaxation

- RELAXATION(u,v)
  - if $d[v] > d[u] + w(u,v)$ then
    - $d[v] = d[u] + w(u,v)$;
    - $p[v] = u$; 前一个点
  - else do nothing

- After relaxation:
  - $d[v] \leq d[u] + w(u,v)$

Algorithm

- INITIALIZE-SINGLE-SOURCE(s)
- RELAX as long as possible.

Upper Bound Property

- CLAIM:
  - $d[v] \geq \delta(s,v)$ for all v in V at all times.
  - Once $d[v] = \delta(s,v)$, $d[v]$ never changes.

一开始，d[v]是很大的，一旦相等，不再变化

- PROOF:
  - By induction on the number of relaxations.
  - BASIS: Clearly true before the first relaxation.
  - INDUCTIVE STEP: Suppose that we are about to relax edge (u,v). $d[u]$ will not change after the relaxation. By inductive hypothesis:
    - $d[v] = d[u] + w(u,v) \geq \delta(s,u) + w(u,v) \geq \delta(s,v)$
- NO-PATH PROPERTY:
  - If $\delta(s,v) = \infty$ (no path from s to v) then $d[v] = \infty$.
  - Follows directly from the upper bound property.

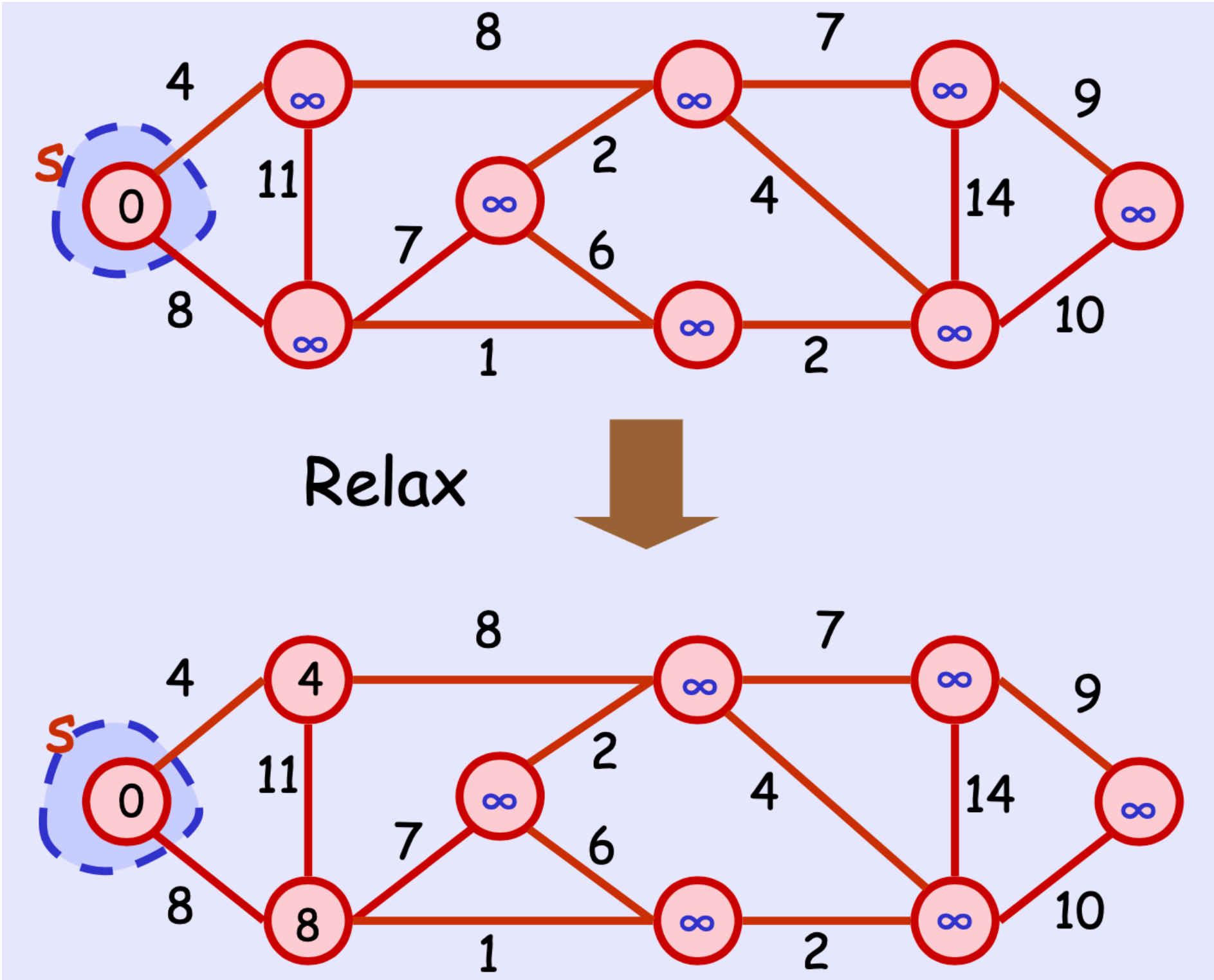
Convergence Property

- Let (u,v) be an edge in G.
  - Assume that the shortest path from s to v goes through (u,v).
  - Assume that at some stage of the execution of the shortest path algorithm we have $d[u] = \delta(s,u)$, and we are about to relax edge (u,v).
  - After the relaxation:

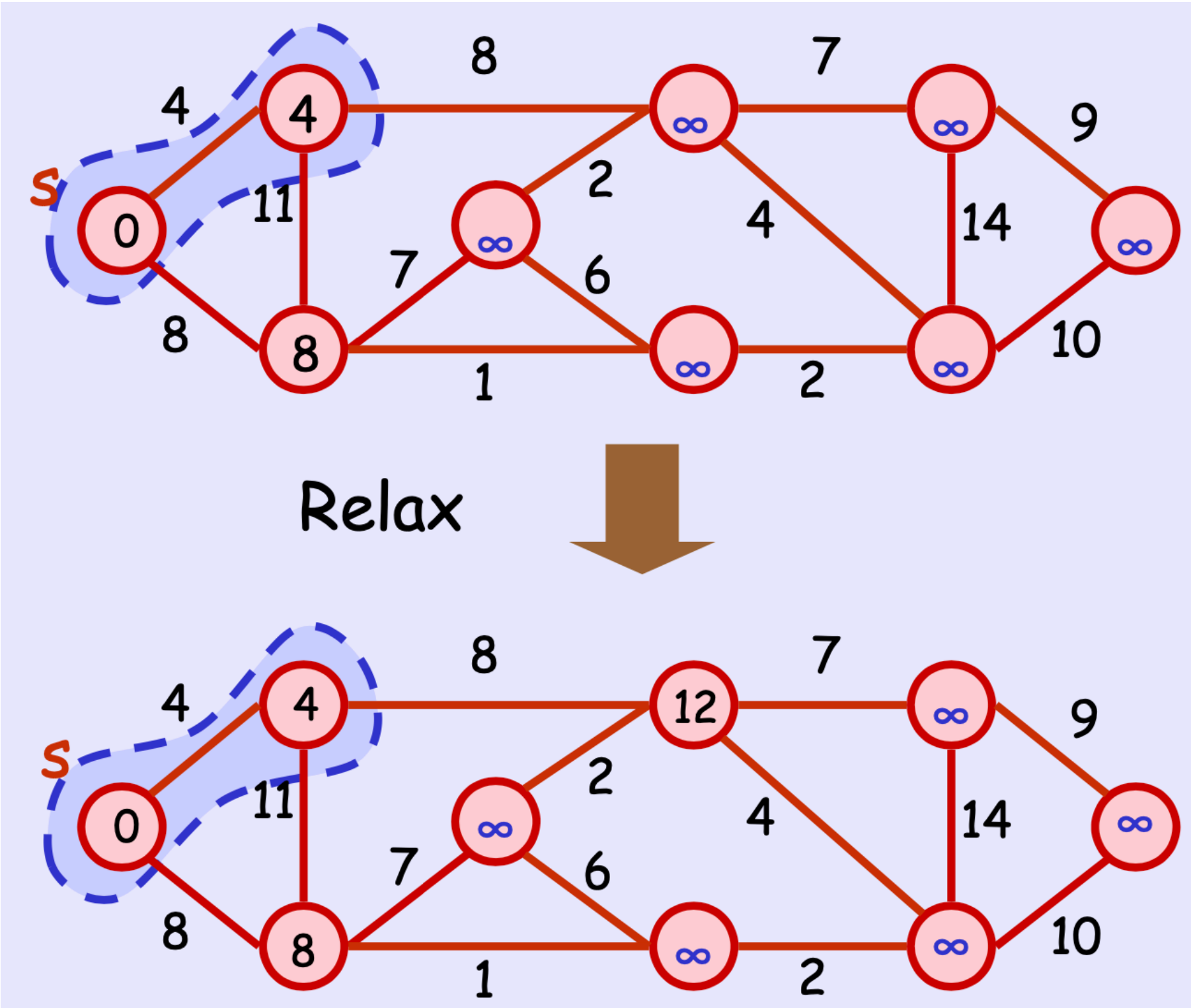
$$d[v] \leq d[u] + w(u,v) = \delta(s,u) + w(u,v) = \delta(\mathrm{s,v})$$

  - Upper bound property: $d[v] \geq \delta(s,v)$.

Example

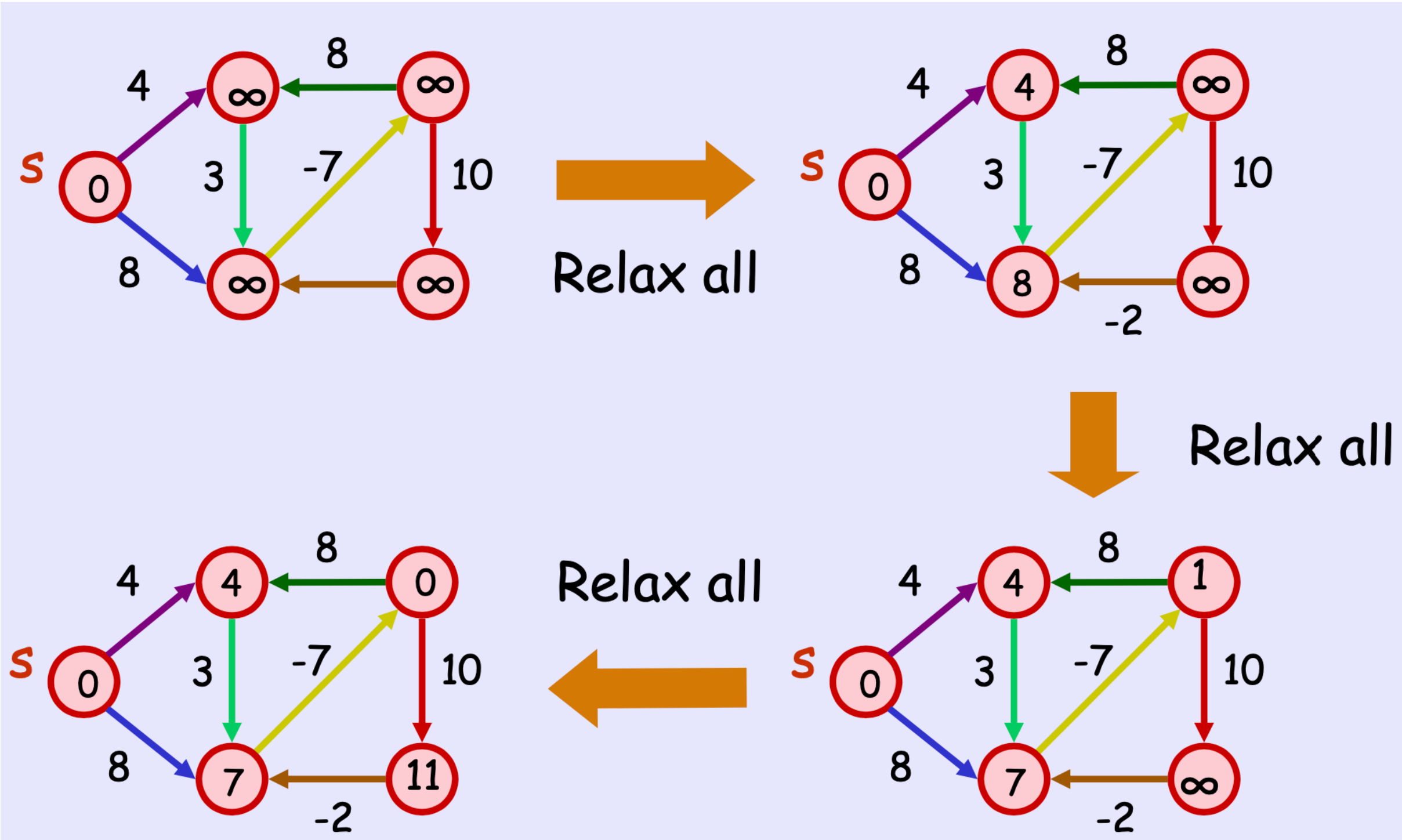


Example

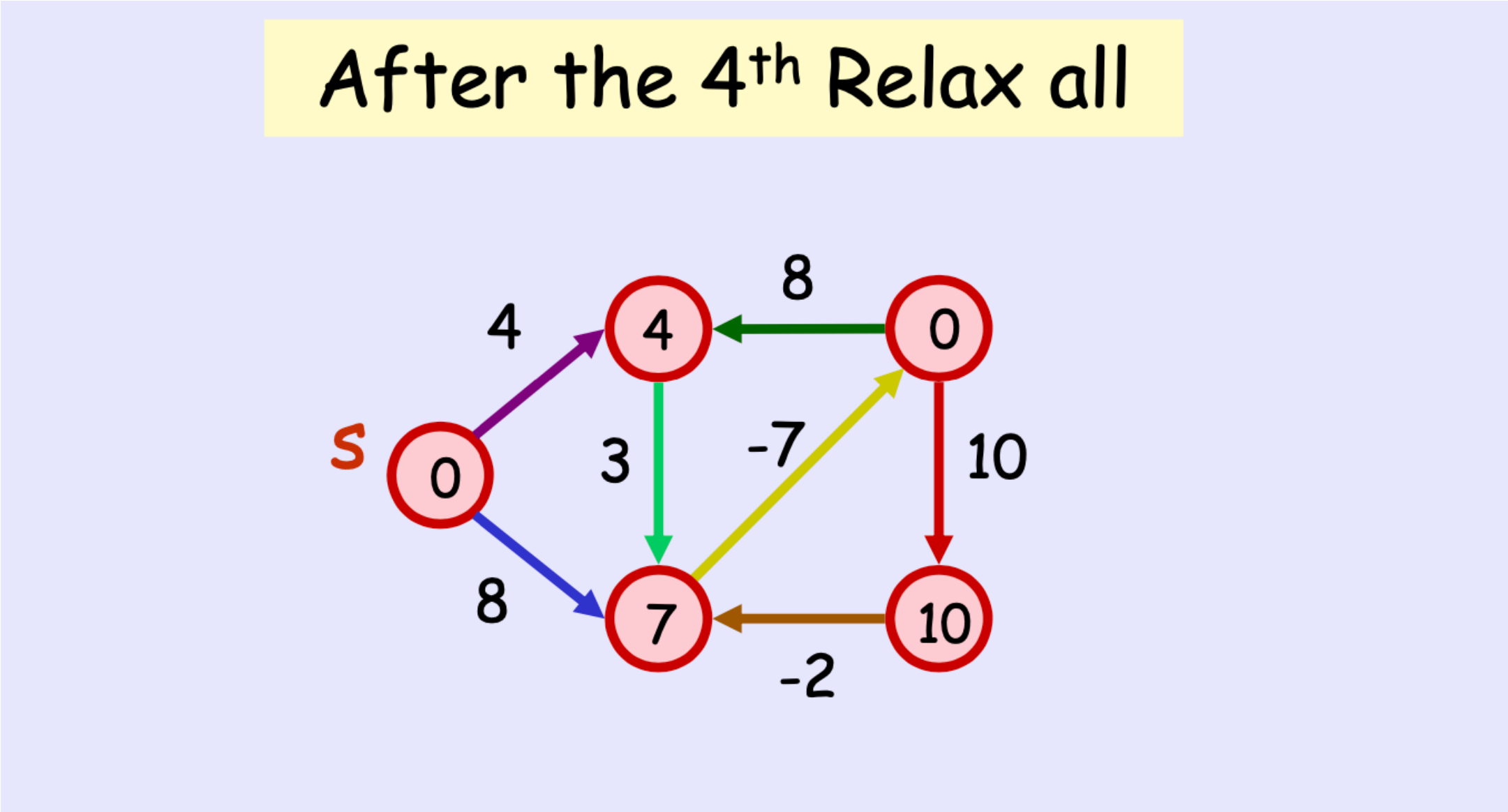

S
0
4
4
8
8
∞
∞
∞
∞
∞
∞
4
8
11
7
8
2
6
1
7
4
2
14
9
10
Relax
S
0
4
4
8
8
12
∞
∞
∞
∞
∞
4
8
11
7
8
2
6
1
7
4
2
14
9
10

- **Types of Shortest Path Problems**
- **Relaxation**
- **Bellman-Ford Algorithm**
- **Dijkstra's Algorithm**
- **Floyd-Warshall Algorithm**

Example 1

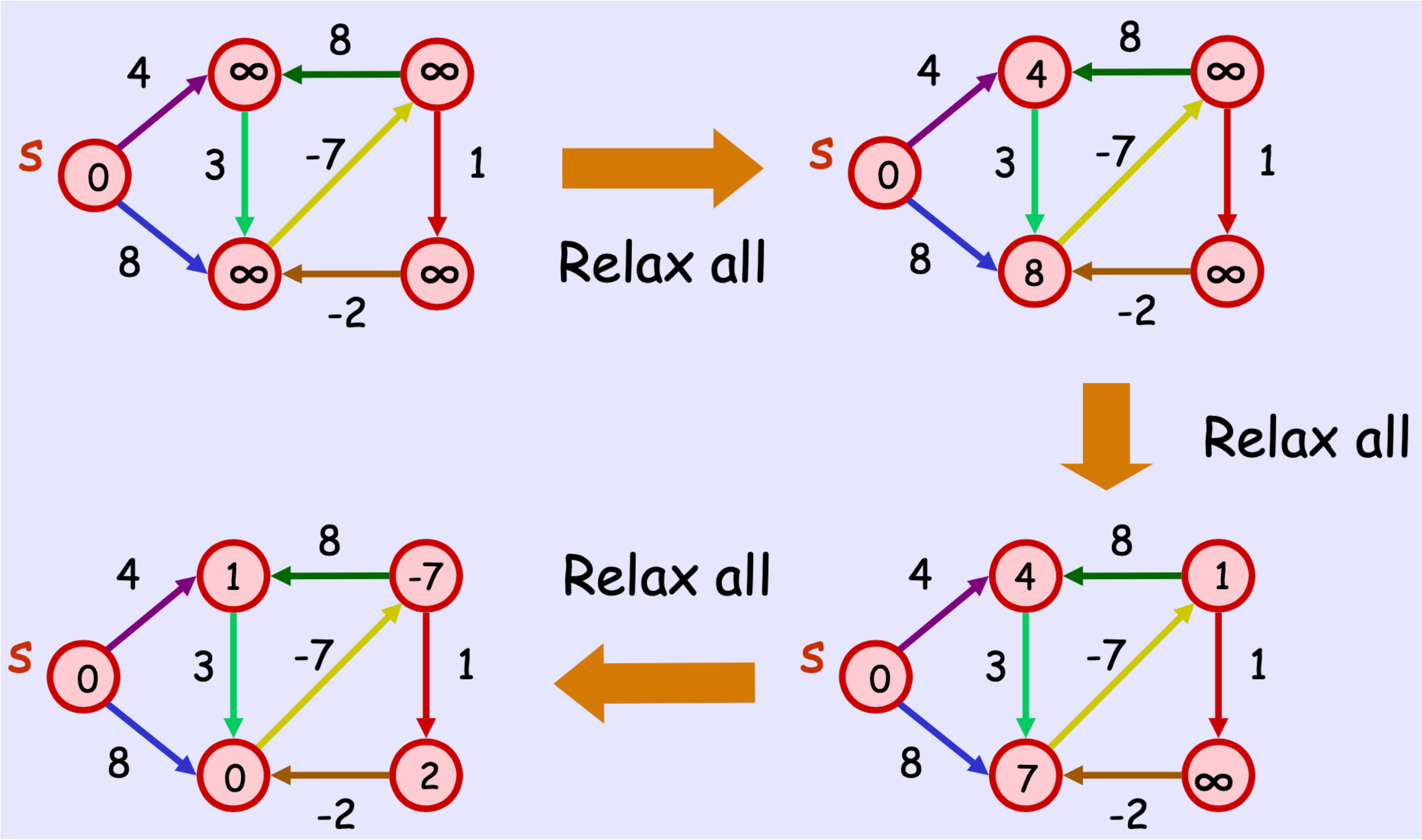

s
0
4
8
∞
∞
∞
∞
3
-7
10
Relax all
s
0
4
8
4
∞
8
∞
3
-7
10
-2
Relax all
s
0
4
8
4
1
8
7
∞
3
-7
10
-2
Relax all
s
0
4
8
4
0
8
7
11
3
-7
10
-2

Example 1

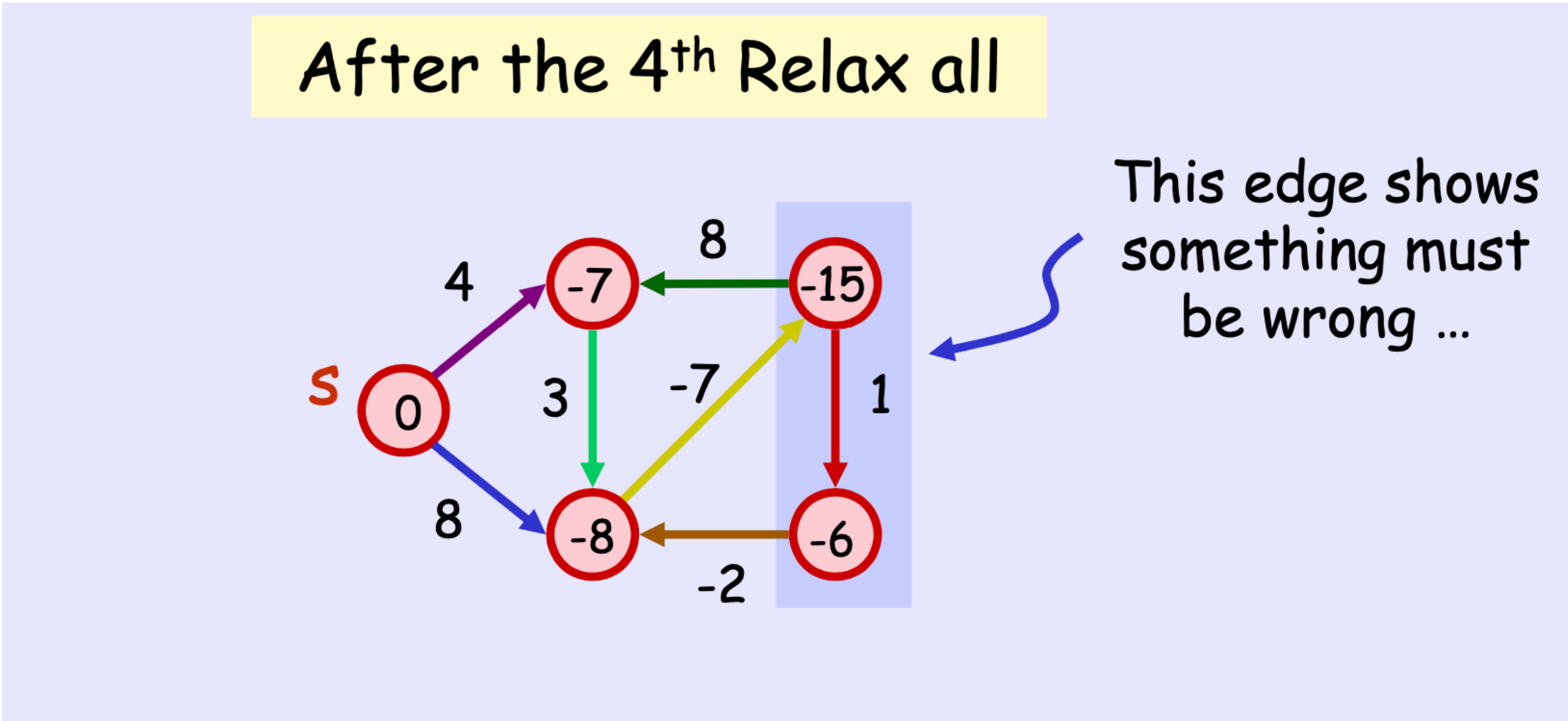


After checking, we found that there is nothing wrong ➔ distances are correct

Example 2

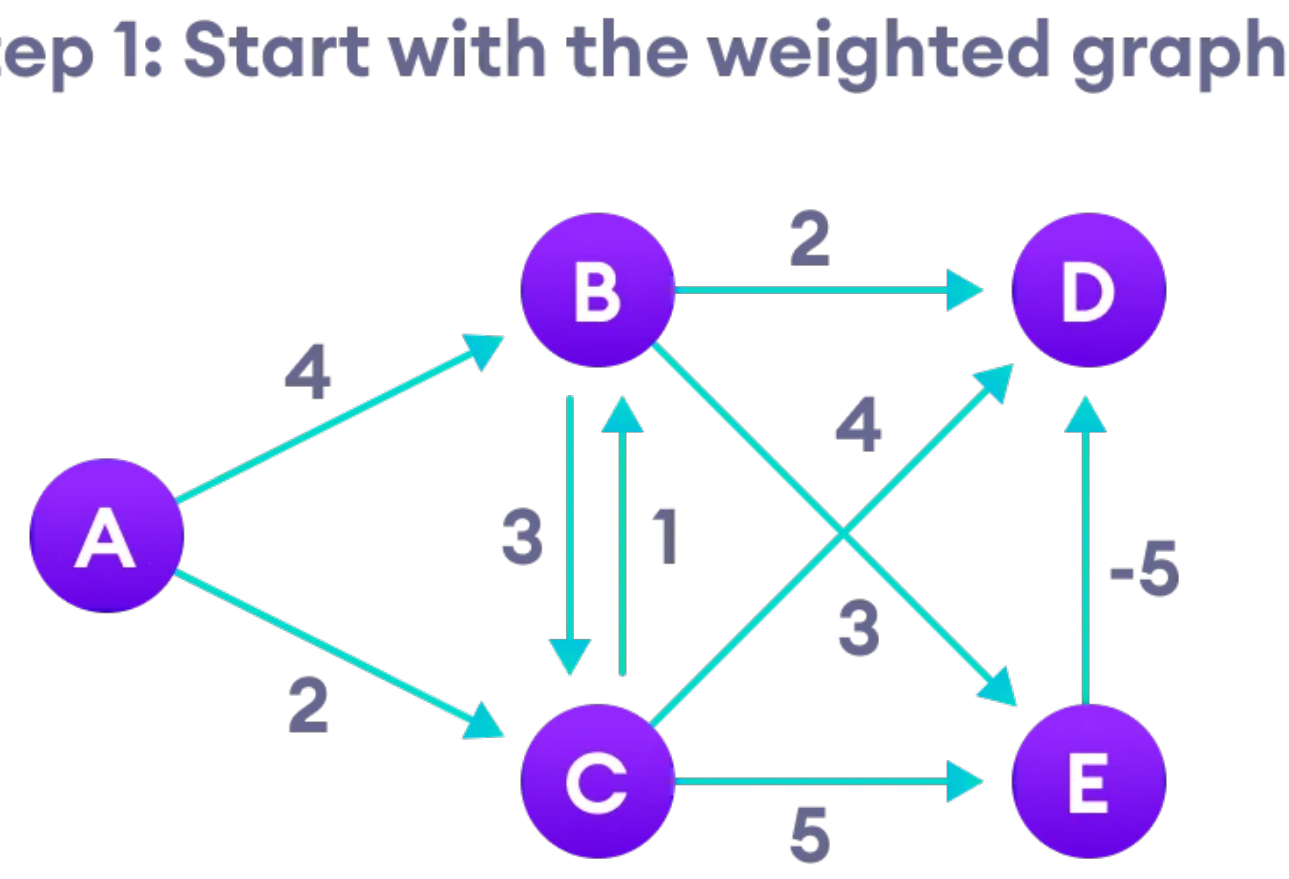

s
0
4
8
∞
∞
∞
∞
3
-7
1
-2
Relax all
s
0
4
8
4
∞
8
∞
3
-7
1
-2
Relax all
s
0
4
8
4
1
8
7
∞
3
-7
1
-2
Relax all
s
0
4
8
1
-7
8
0
2
3
-7
1
-2

Example 2

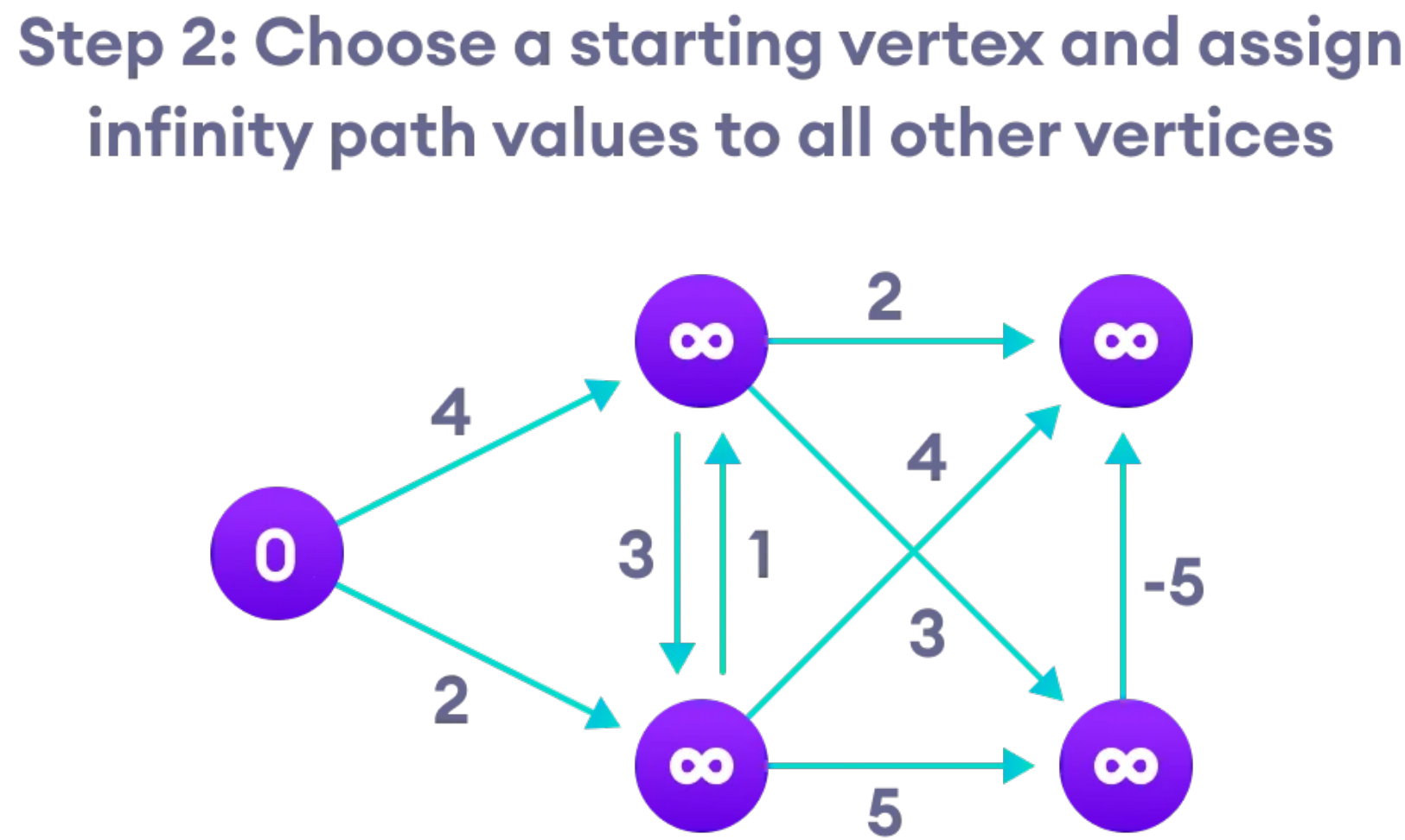


After checking, we found that something must be wrong ➔ distances are incorrect

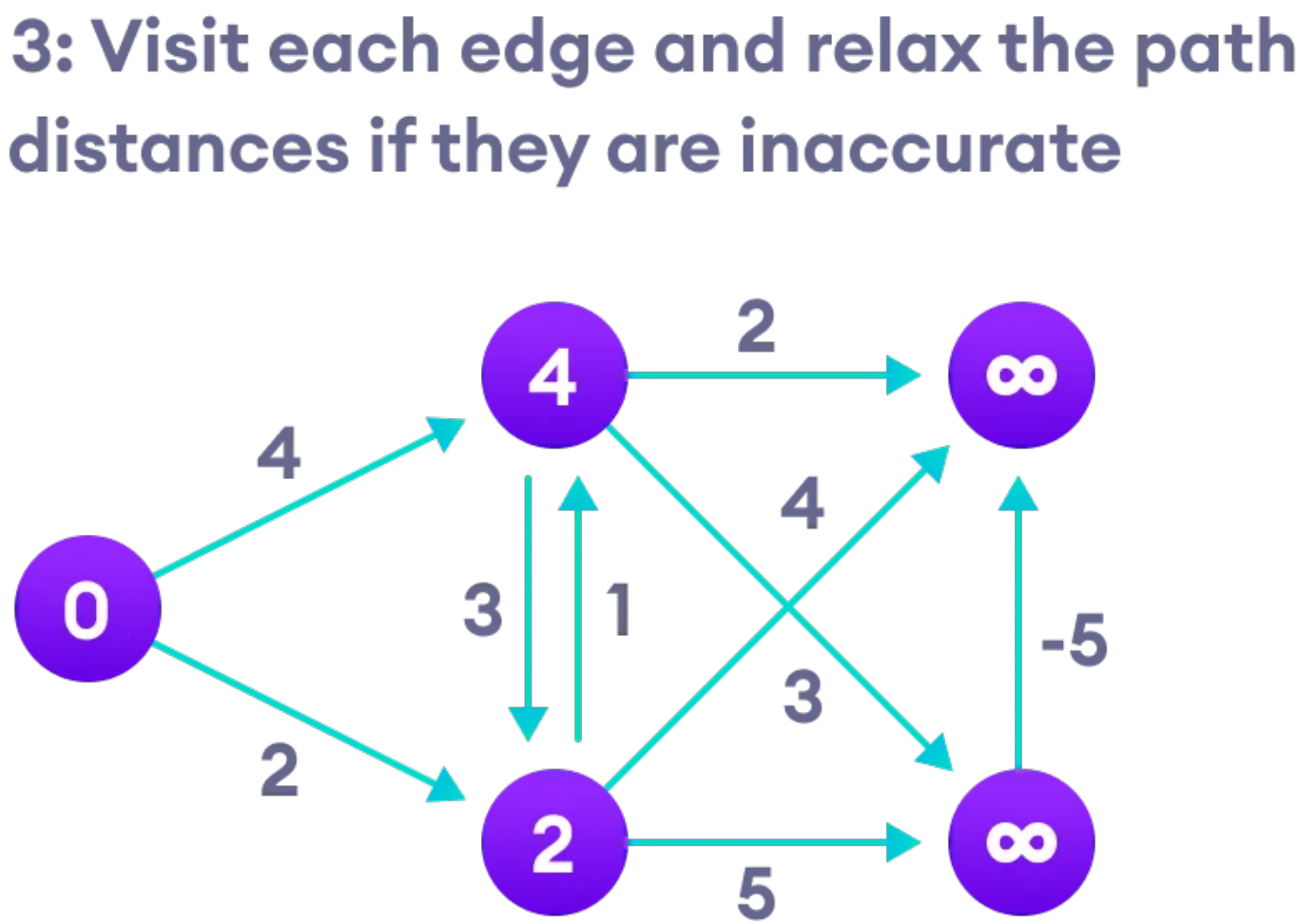
Example 3



Example 3

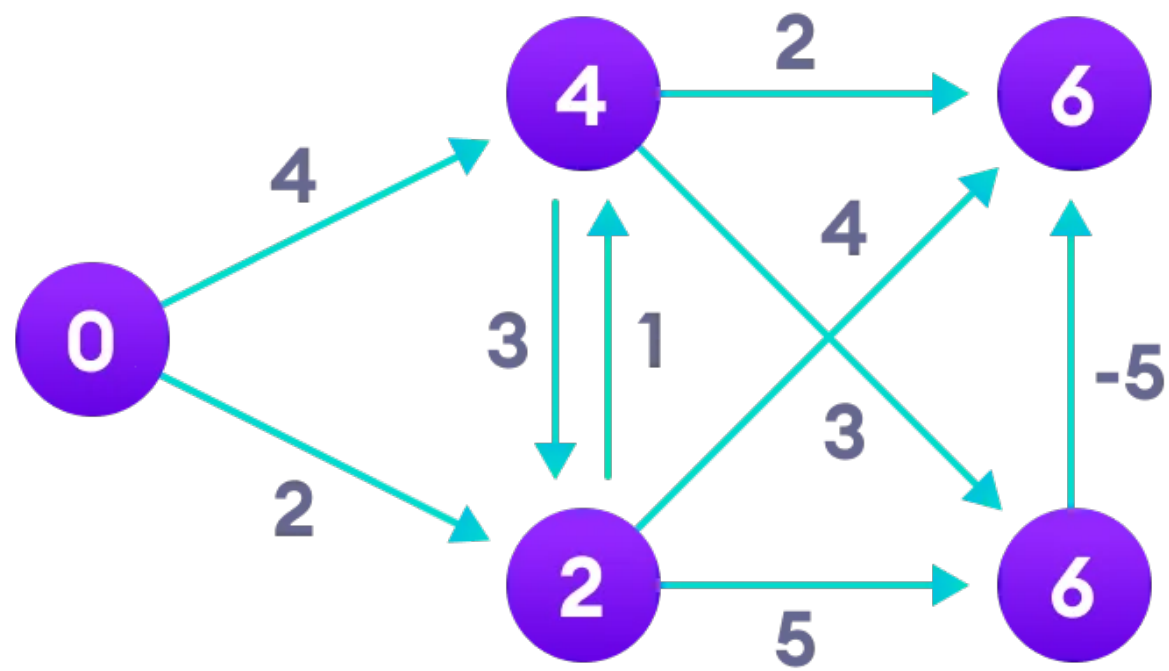

Step 2: Choose a starting vertex and assign infinity path values to all other vertices
0
∞
∞
∞
∞
4
2
2
3
1
4
3
-5
5

Example 3

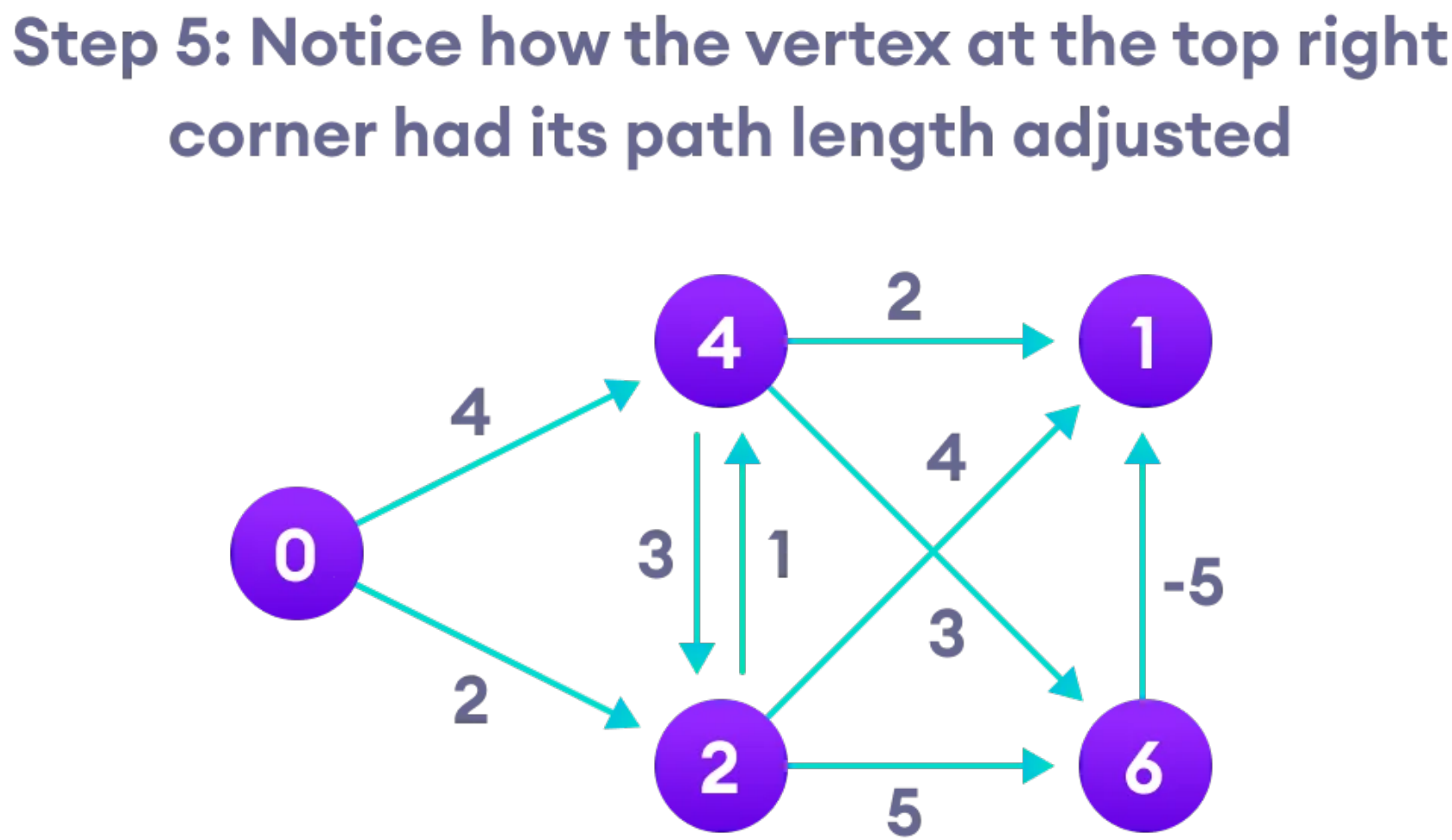

Step 3: Visit each edge and relax the path distances if they are inaccurate
0
4
2
∞
2
∞
4
2
3
1
4
3
-5
5

Example 3

Step 4: We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times



Example 3


Step 5: Notice how the vertex at the top right corner had its path length adjusted
0
4
1
2
6
4
2
4
3
1
3
-5
2
5

Example 3

Step 6: After all the vertices have their path lengths, we check if a negative cycle is present

	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	6
0	3	2	1	6
0	3	2	1	6

Bellman-Ford Algorithm

- INITIALIZE-SINGLE-SOURCE(G,s)
  - for (i = 1; $i \leq n$-1; i++) do
    - for each edge $(u,v) \in G$ do relax (u,v);
    - for each edge $(u,v) \in G$ do
      - if $d[v] > d[u] + w(u,v)$ then return FALSE;
    - return TRUE;

- Worst case time complexity: $O(nm)$
  - $n = |V|$
  - $m = |E|$

Correctness of BF Algorithm

- Let $P = \{ v_0, v_1, ..., v_k \}$ be a shortest path from $s=v_0$ to v_k.
- CLAIM: If G has no negative cycles then $d[v] = \delta(s,v)$ for all v after n-1 iteration.
- PROOF:
- Any path in G without cycles has at most n-1 edges.
- Therefore $k <= n$-1.
- Before the first round of relaxations, $d[v_0] = \delta(s,v_0)$.
- When (v_0,v_1) is relaxed in the first round, $d[v_1] = \delta(s,v_1)$.
- Before the second round of relaxations, $d[v_1] = \delta(s,v_1)$.
- When (v_1,v_2) is relaxed in the second round, $d[v_2] = \delta(s,v_2)$
- Before the i-th round of relaxations, $d[v_{i-1}]=\delta(s,v_{i-1})$.
- When $(v_{i-}1,v_i)$ is relaxed in the i-th round, $d[v_i] = \delta(s,v_i)$.

Correctness of BF Algorithm

- If G has no negative cycles then after n-1 iteration: $d[v] = \delta(s,v)$ for every $v \in G$.
- Let (u,v) be arbitrary edge in G.
  - $d[v] = \delta(s,v) \leq \delta(s,u) + w(u,v) = d[u] + w(u,v)$
- Bellman-Ford returns TRUE.

Correctness of BF Algorithm

- Suppose that G contains a negative cycle reachable from s.
- Assume for the purpose of contradiction that BF returns TRUE.
- Let $C = \{ v_0, v_1, \ldots, v_k \}$, $v_0 = v_k$, be a negative cycle.
- Since BF is assumed to return TRUE, we must have
  - $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_{i-1})$, i = 1, 2, …, k
- Summing these inequalities around the cycle, we get
  - $\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$
- This leads to a contradiction since we get
  - $0 \leq \sum_{i=1}^{k} w(v_{i-1}, v_i)$

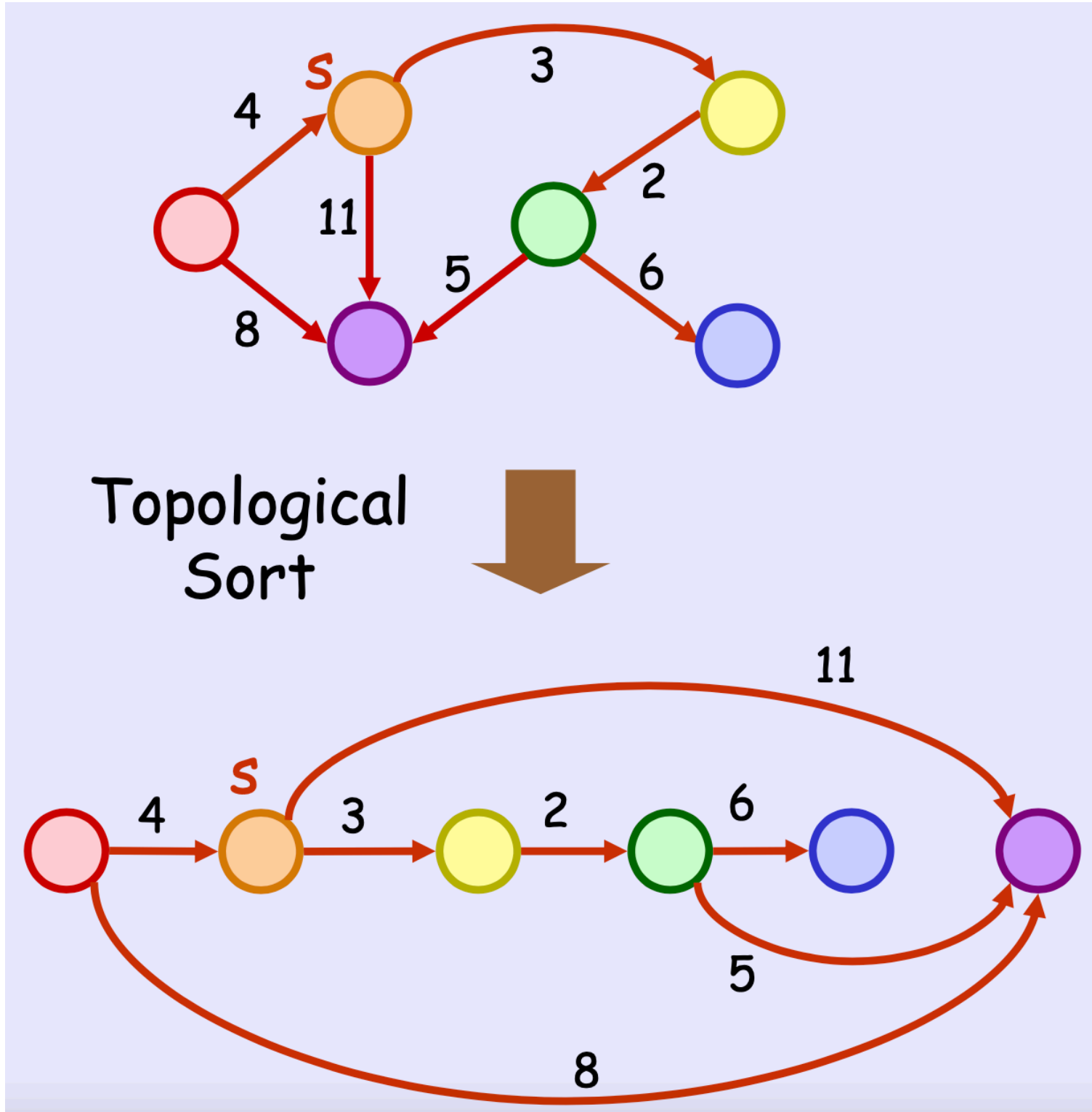
Shortest Paths in DAGs

- Topologically sort vertices of the DAG.
- Process vertices in topological order.
- For each vertex, relax its outgoing edges.

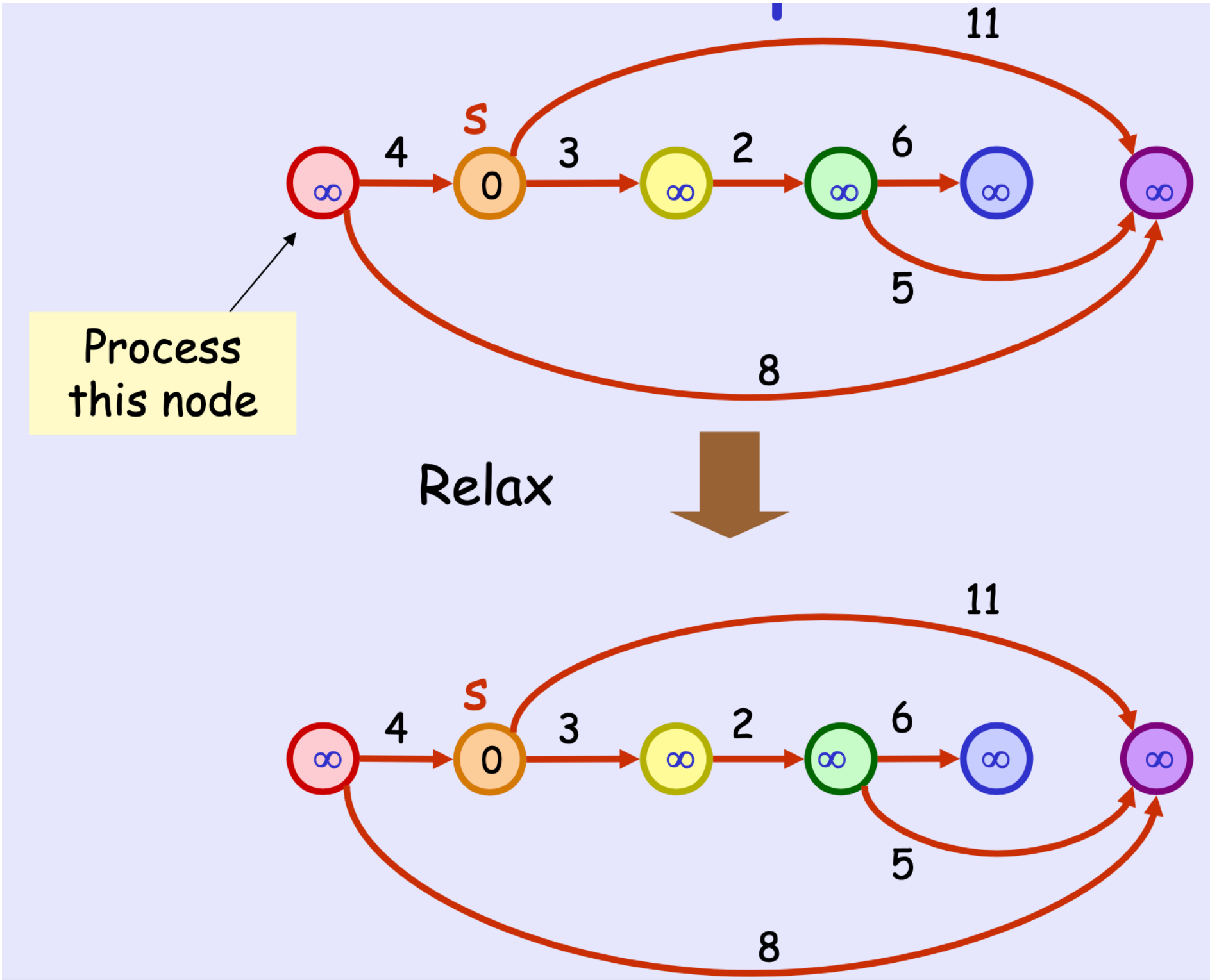
We have a faster algorithm for DAG :

```
DAG-Shortest-Path(G, s)
    Topological Sort G ;
    For each v, set d(v) = ∞; Set d(s) = 0 ;
    for (k = 1 to |V|) {
        v = kth vertex in topological order ;
    Relax all outgoing edges of v ;
    }
return d ;
```

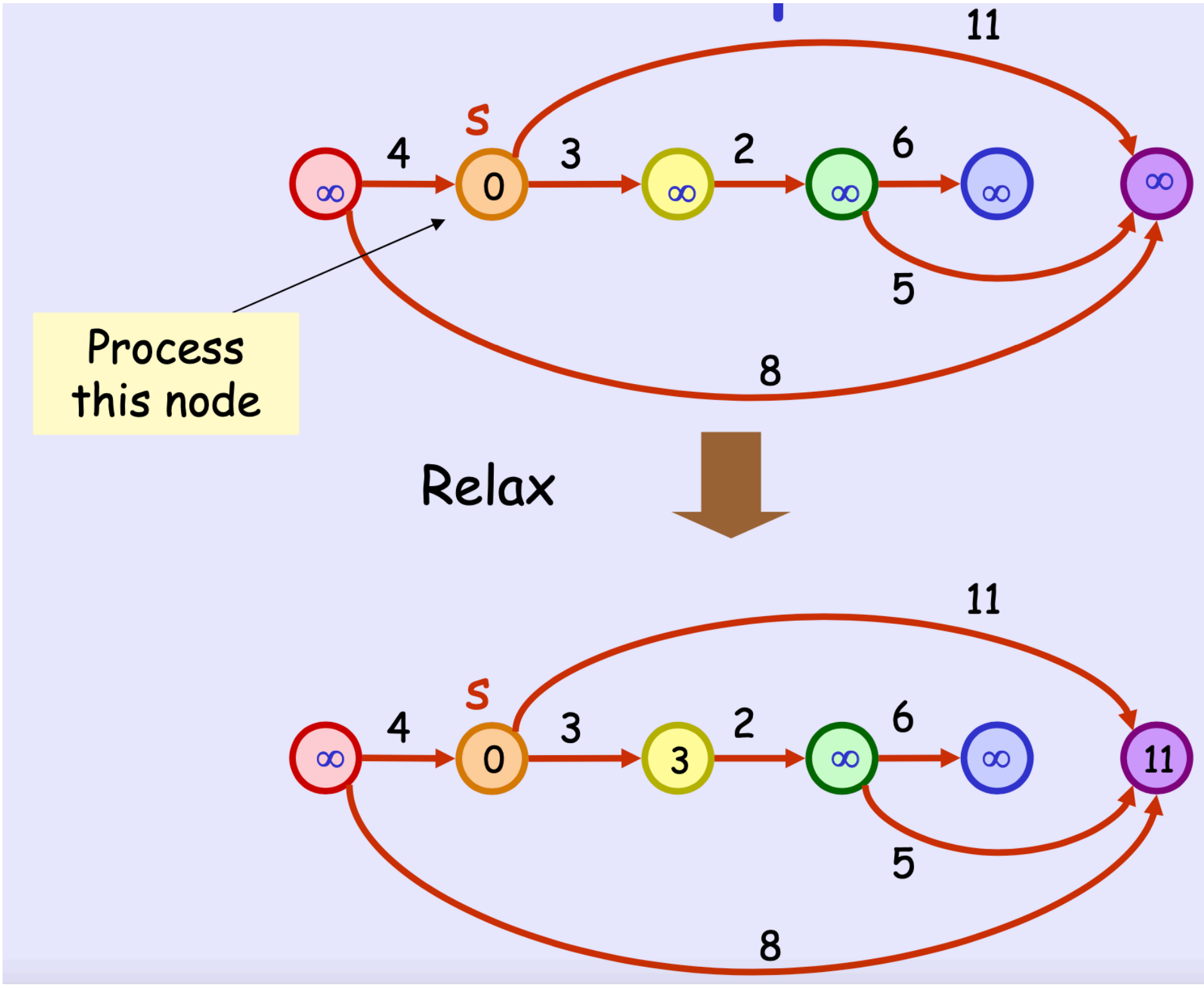
Example



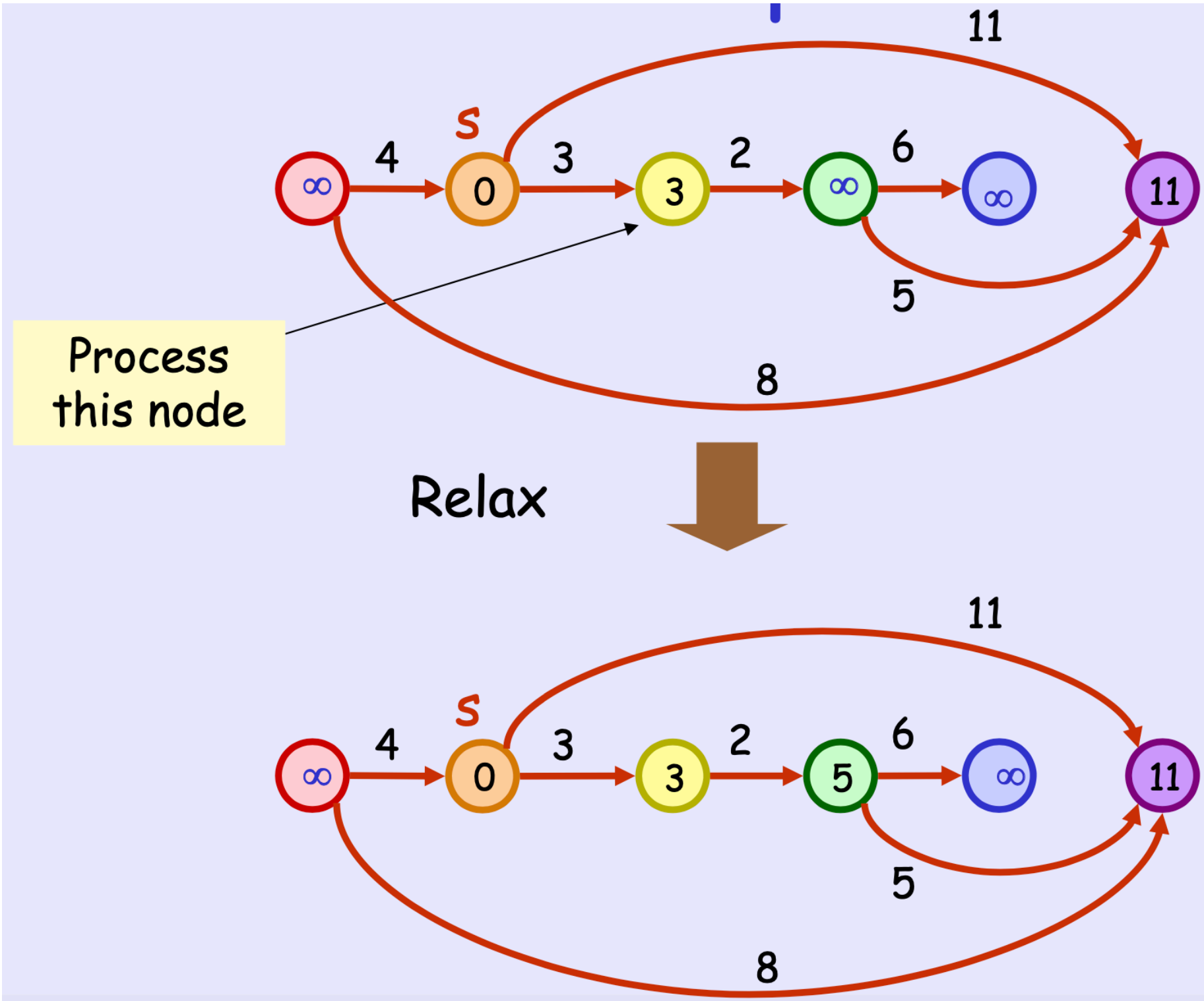
Example



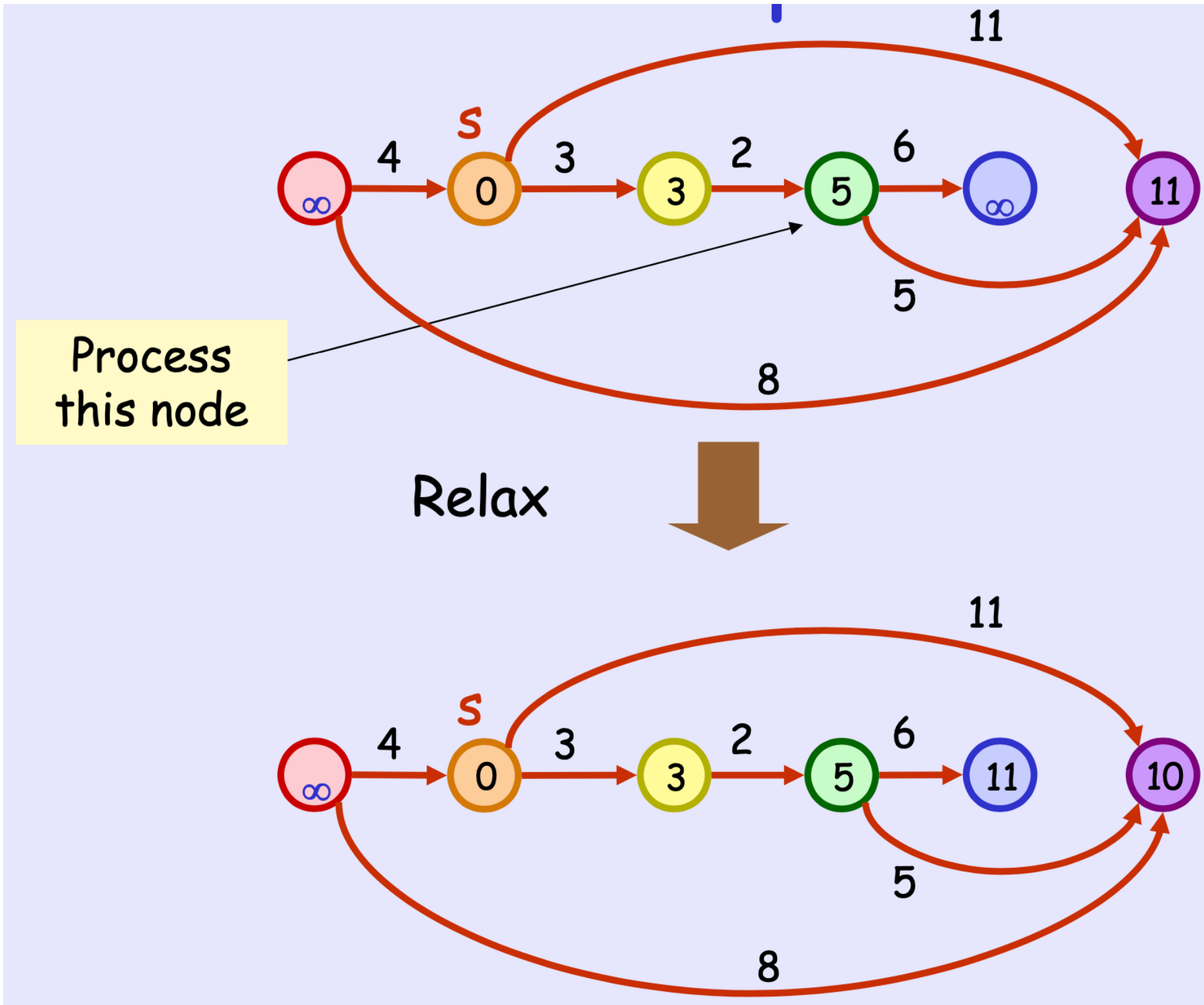
Example



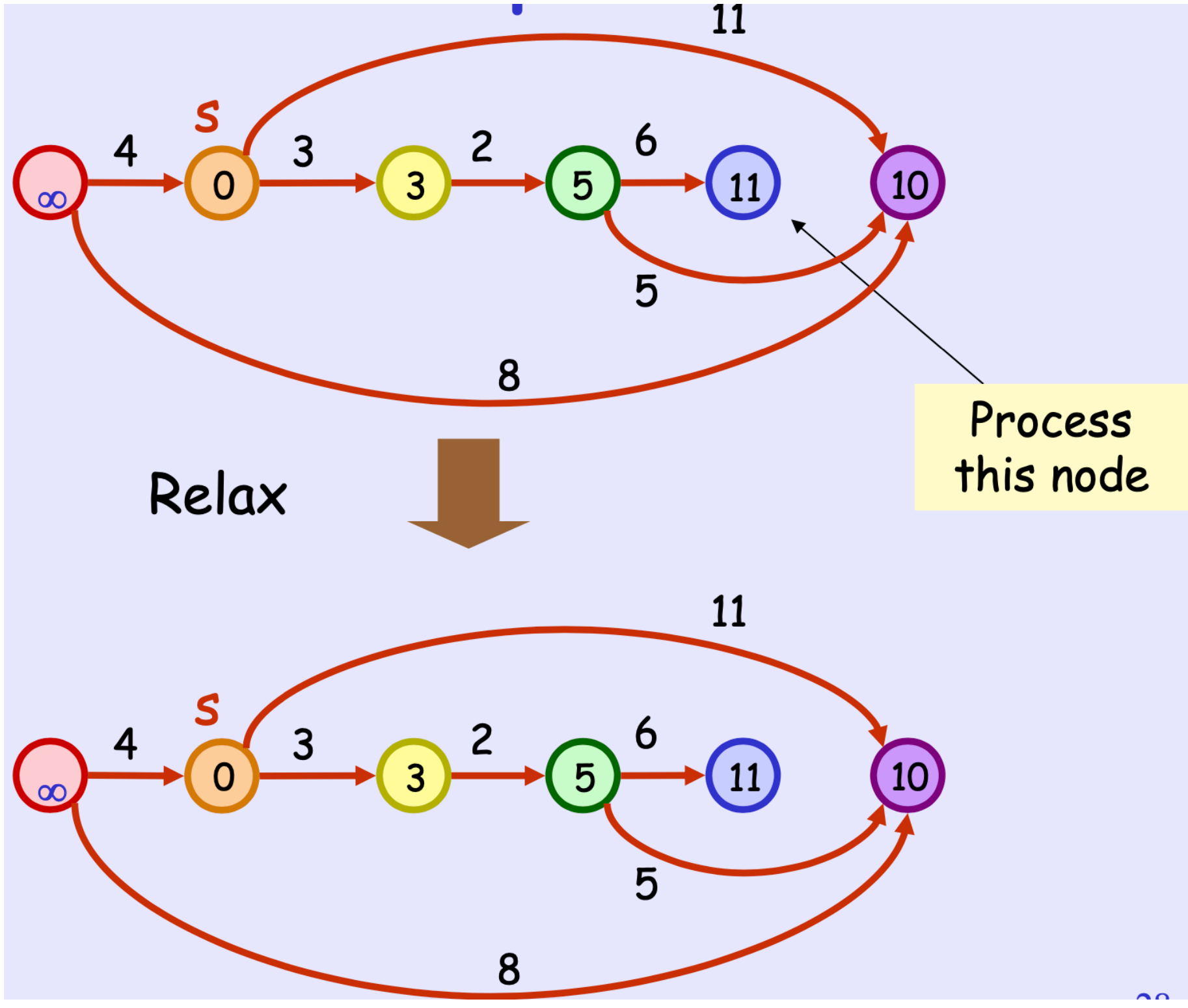
Example



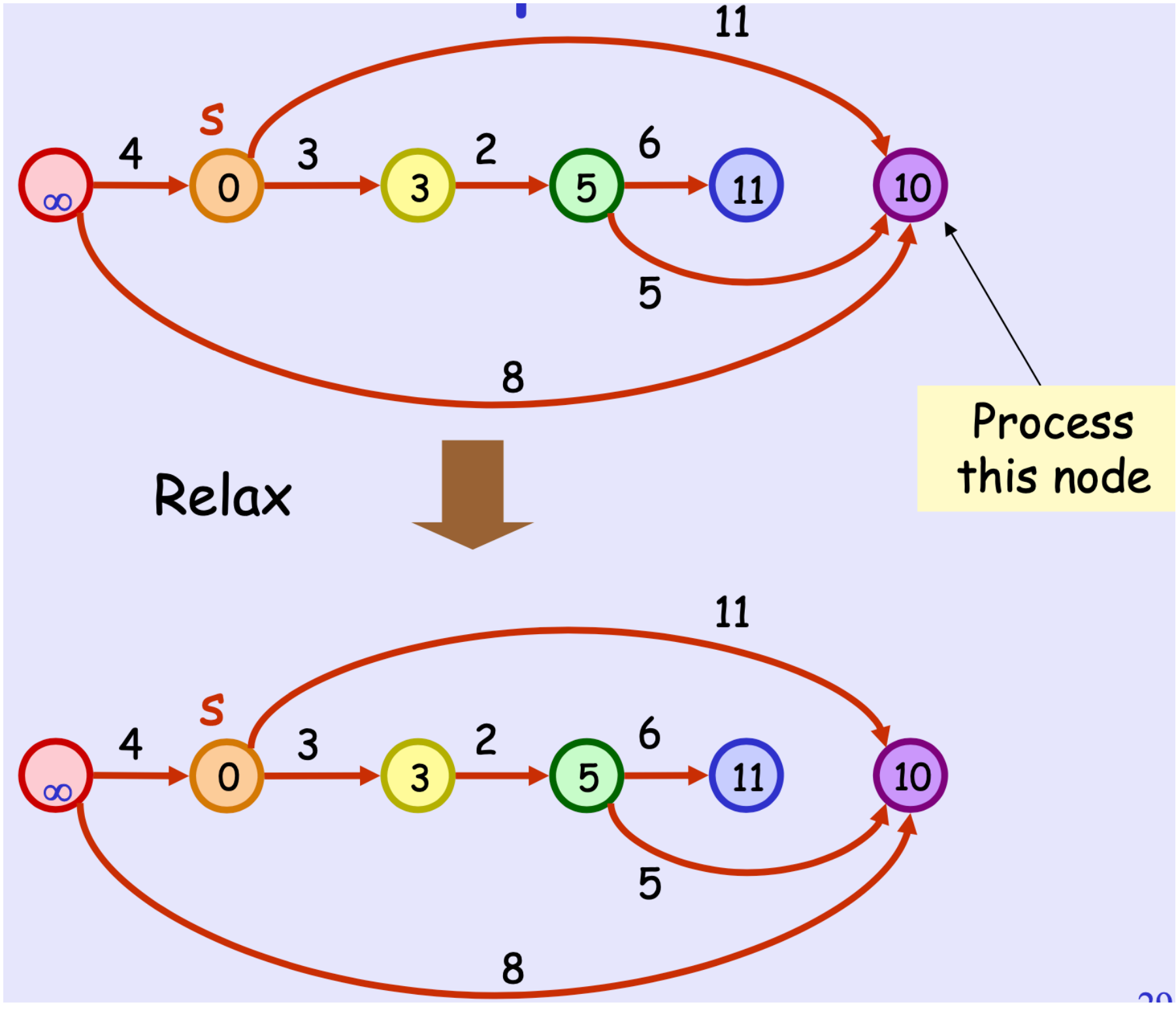
Example


11
s
4
3
2
6
∞
0
3
5
∞
11
5
8
Process this node
Relax
11
s
4
3
2
6
∞
0
3
5
11
10
5
8

Example


11
s
4
3
2
6
∞
0
3
5
11
10
5
8
Process
this node
Relax
11
s
4
3
2
6
∞
0
3
5
11
10
5
8

Example


11
s
4
3
2
6
∞
0
3
5
11
10
5
8
Process
this node
Relax
11
s
4
3
2
6
∞
0
3
5
11
10
5
8

Correctness of the Shortest Paths Algorithm for DAGs

- If v is not reachable from s than the algorithm returns $d[v] = \infty$ (no edge into v is ever relaxed).
- If v is reachable from s, consider the shortest path $\{v_0, v_1, \ldots, v_k\}$ with $v_0 = s$ and $v_k = v$.
- The edges of this path: $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ are relaxed in this order since the vertices are processed in topological order.
- As a consequence, shortest paths in a DAG are computed after one relaxation pass of all its edges.
- Worst-case time complexity of the algorithm is $O(m+n)$.

- **Types of Shortest Path Problems**
- **Relaxation**
- **Bellman-Ford Algorithm**
- **Dijkstra's Algorithm**
- **Floyd-Warshall Algorithm**

Dijkstra's Algorithm

- Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.
- It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

How Dijkstra's Algorithm works

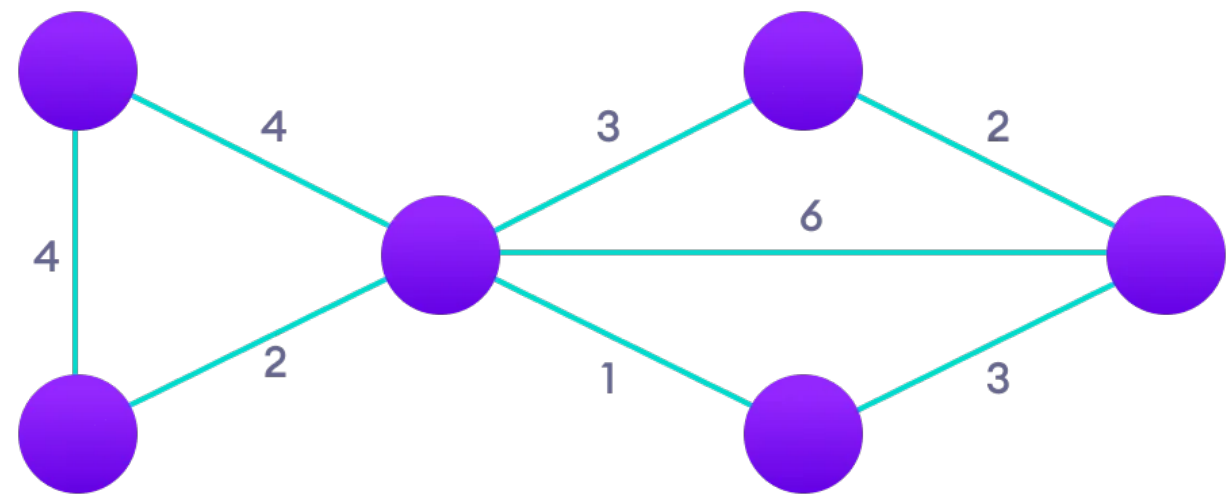
- Dijkstra's Algorithm works on the basis that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.

How Dijkstra's Algorithm works

- Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbors to find the shortest subpath to those neighbors.
- The algorithm uses a greedy approach 贪心 in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

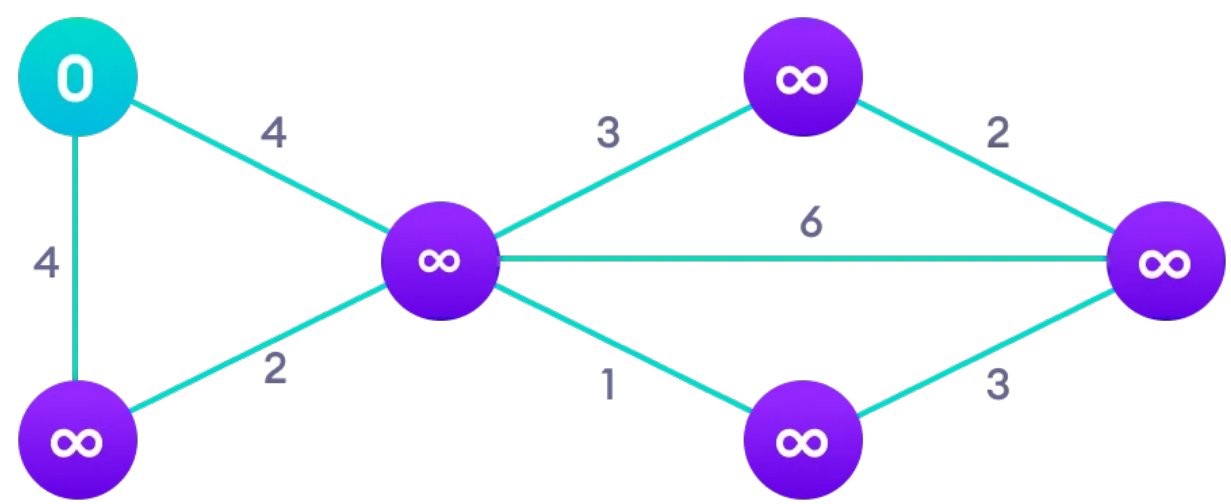
Example of Dijkstra's algorithm

- Start with a weighted graph



Step: 1

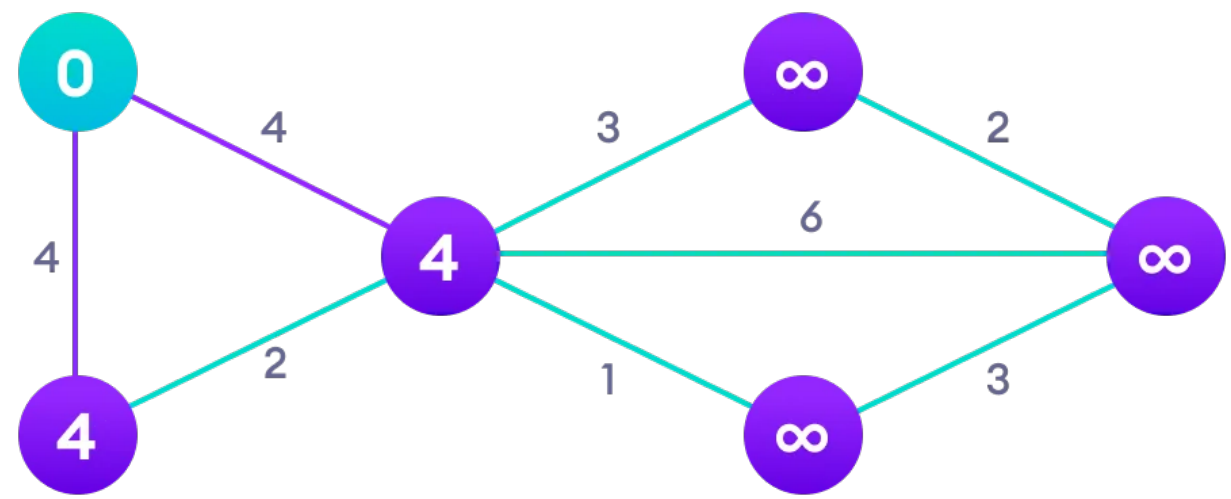
- Choose a starting vertex and assign infinity path values to all other devices



Step: 2

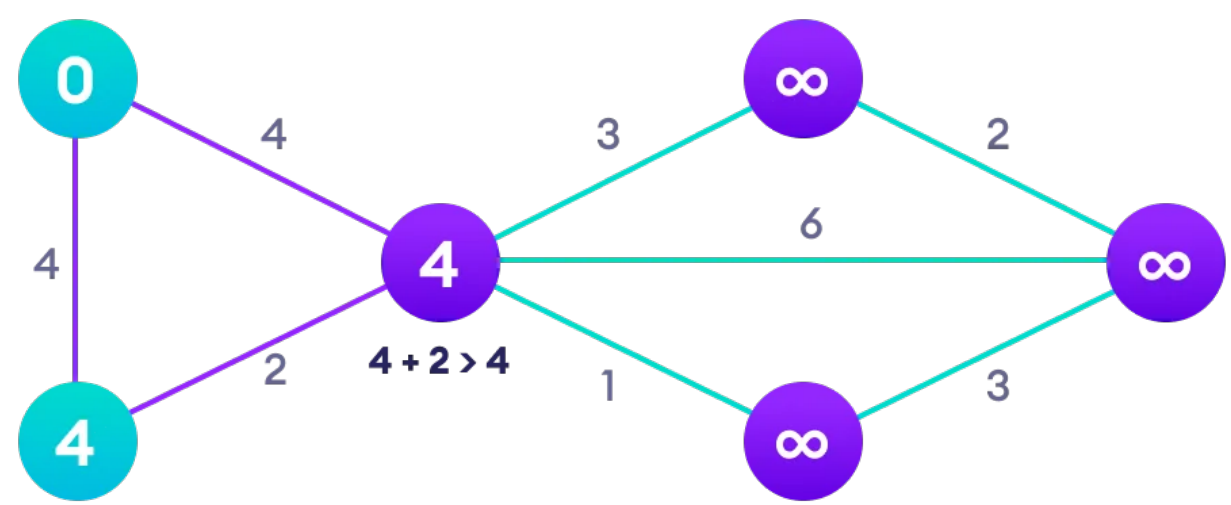
Example of Dijkstra's algorithm

- Go to each vertex and update its path length



Step: 3

- If the path length of the adjacent vertex is lesser than new path length, don't update it

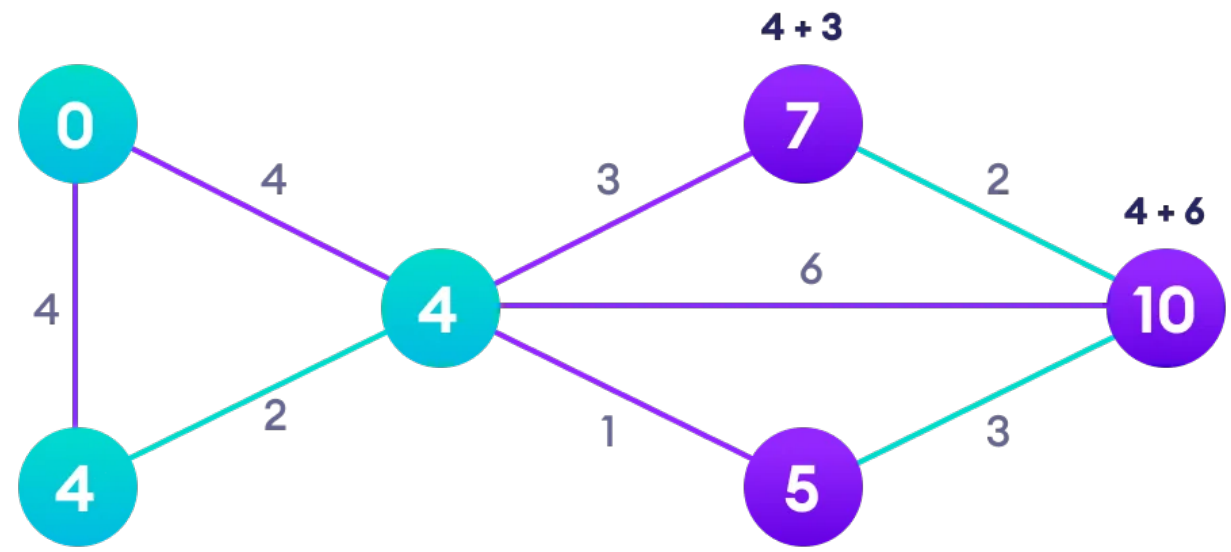


Step: 4

Example of Dijkstra's algorithm

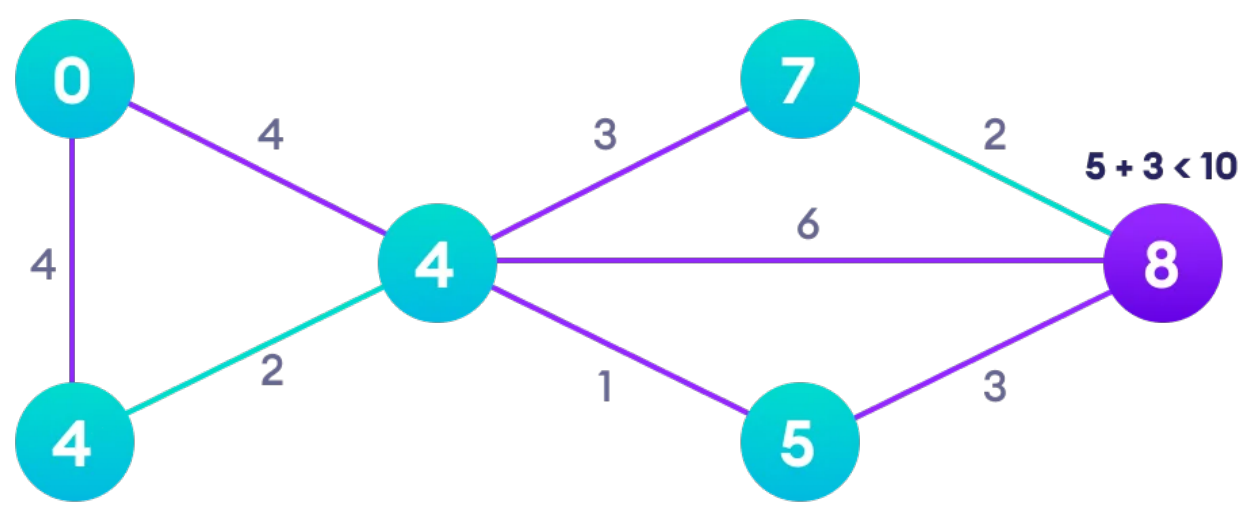
- Avoid updating path lengths of already visited vertices

已经走过的点，就不再走了，



Step: 5

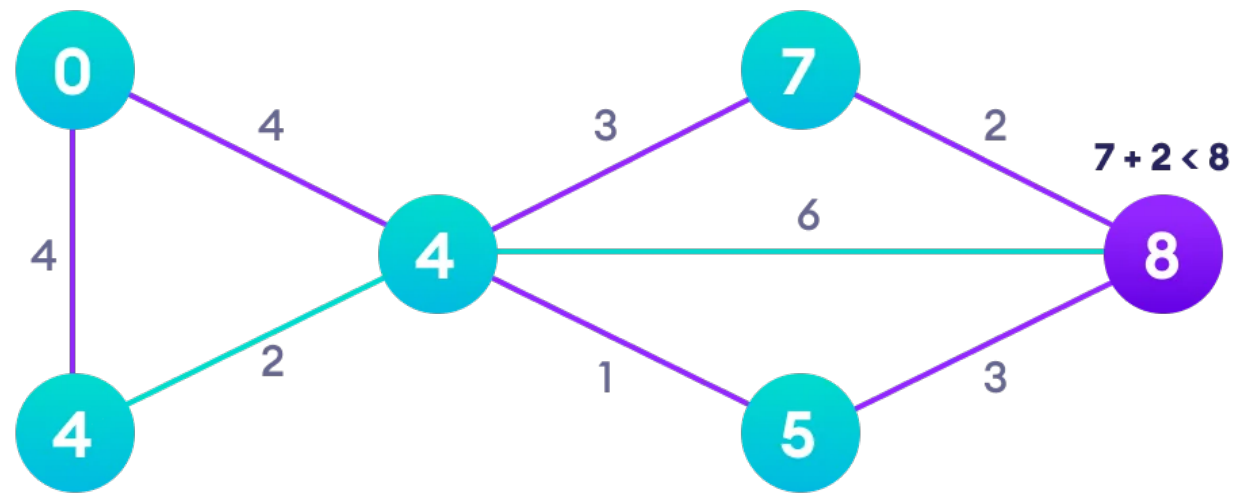
- After each iteration, we pick the unvisited vertex with the least path length. So we choose 5 before 7



Step: 6

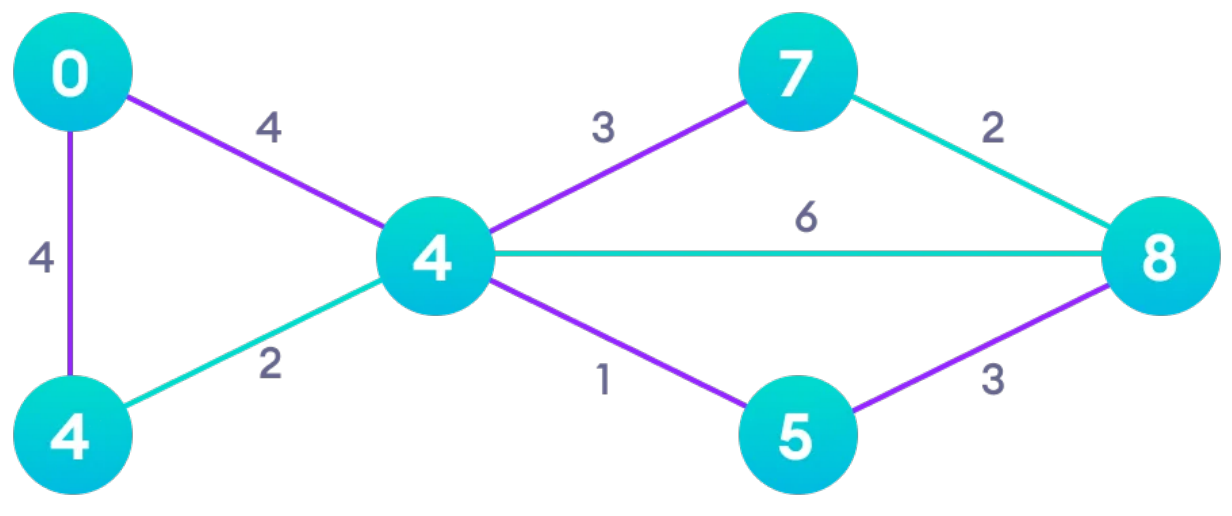
Example of Dijkstra's algorithm

- Notice how the rightmost vertex has its path length updated twice



Step: 7

- Repeat until all the vertices have been visited

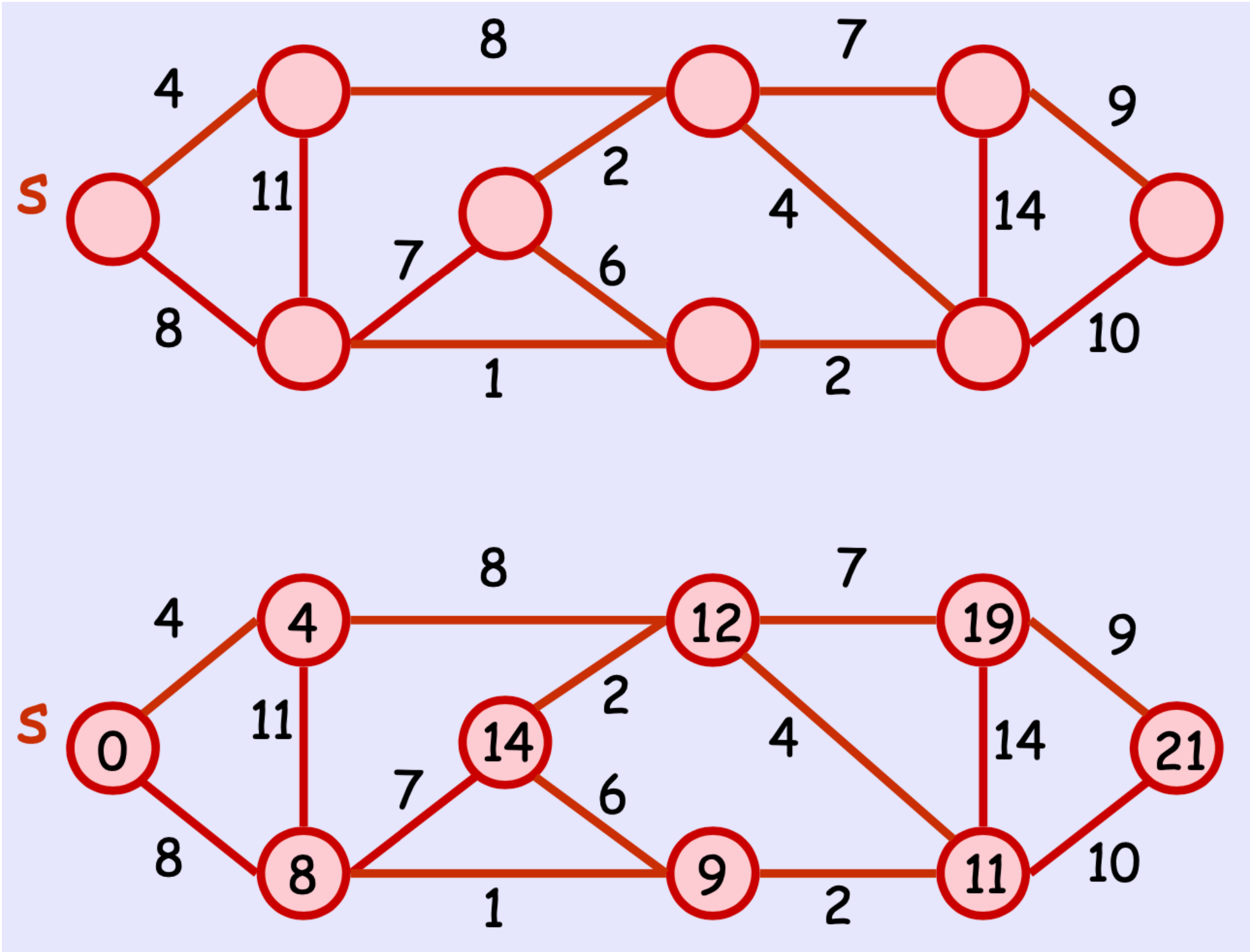


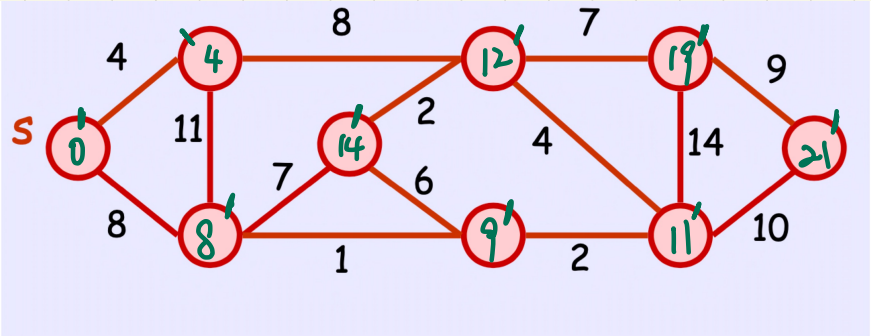
Step: 8

Dijkstra's Algorithm

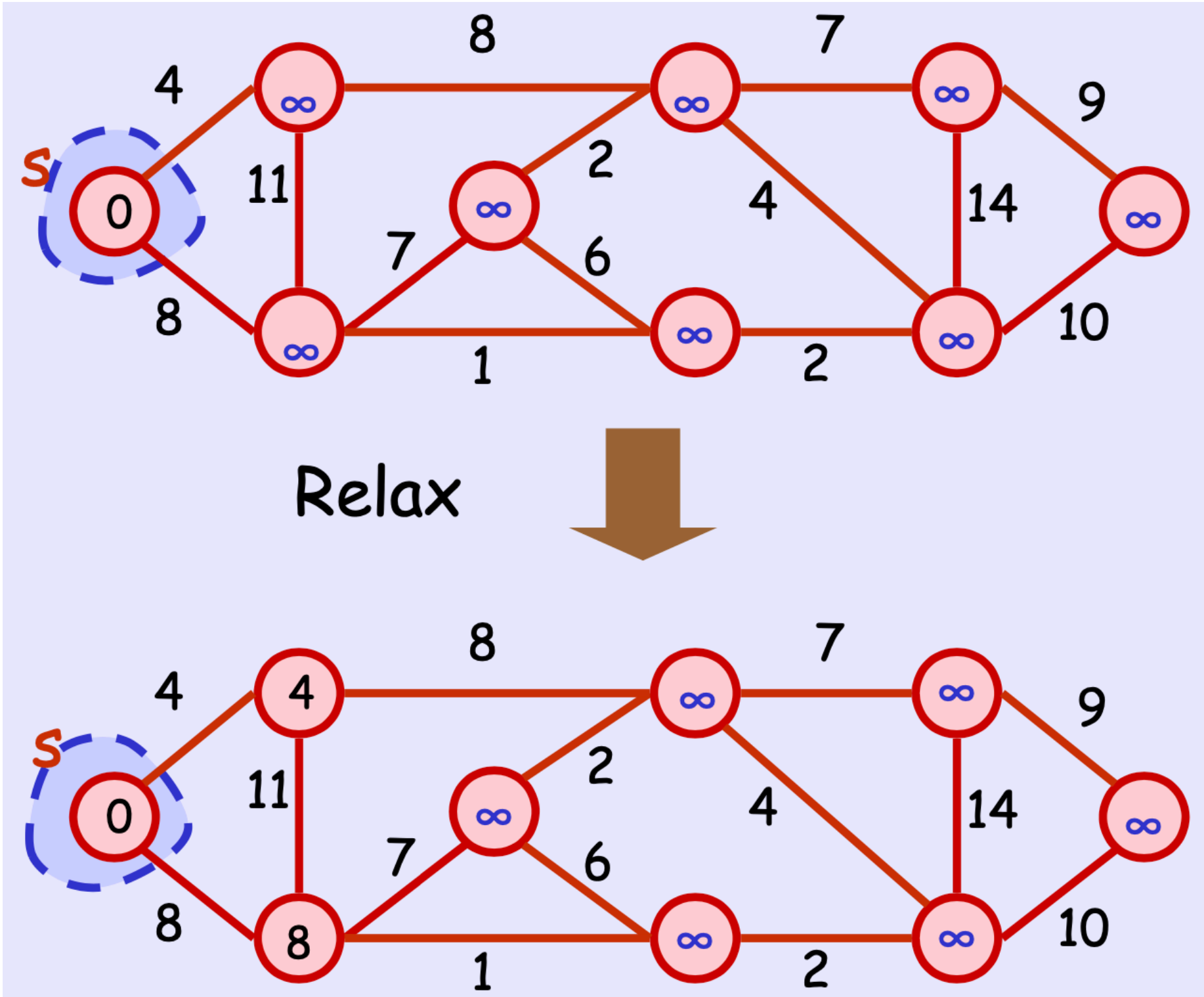
- Non-negative edge weights.
- Let S be a set of vertices whose shortest paths from s have been determined. Initially $S = \emptyset$, $d[s] = 0$, $d[u] = \infty$ for all $u \neq s$.
- At each iteration, select a vertex u with the smallest $d[u]$ and not in S.
  - Relax along all edges going out from u.
  - Add u to S.
- Very similar to breadth-first search algorithm in graphs.

Example

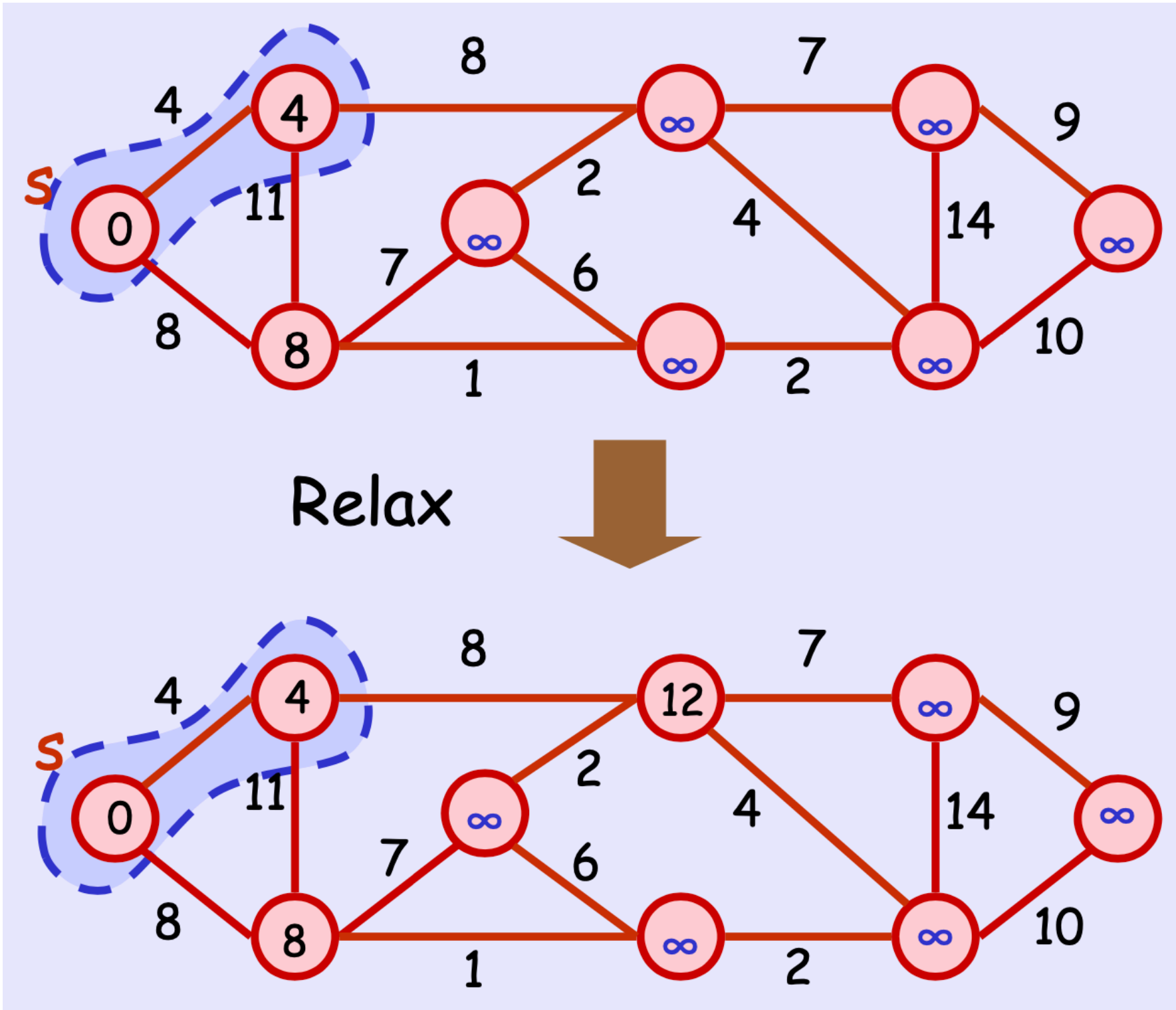



s
0
4
8
14
12
19
21
9
11
4
8
8
7
2
6
1
7
4
2
9
14
10

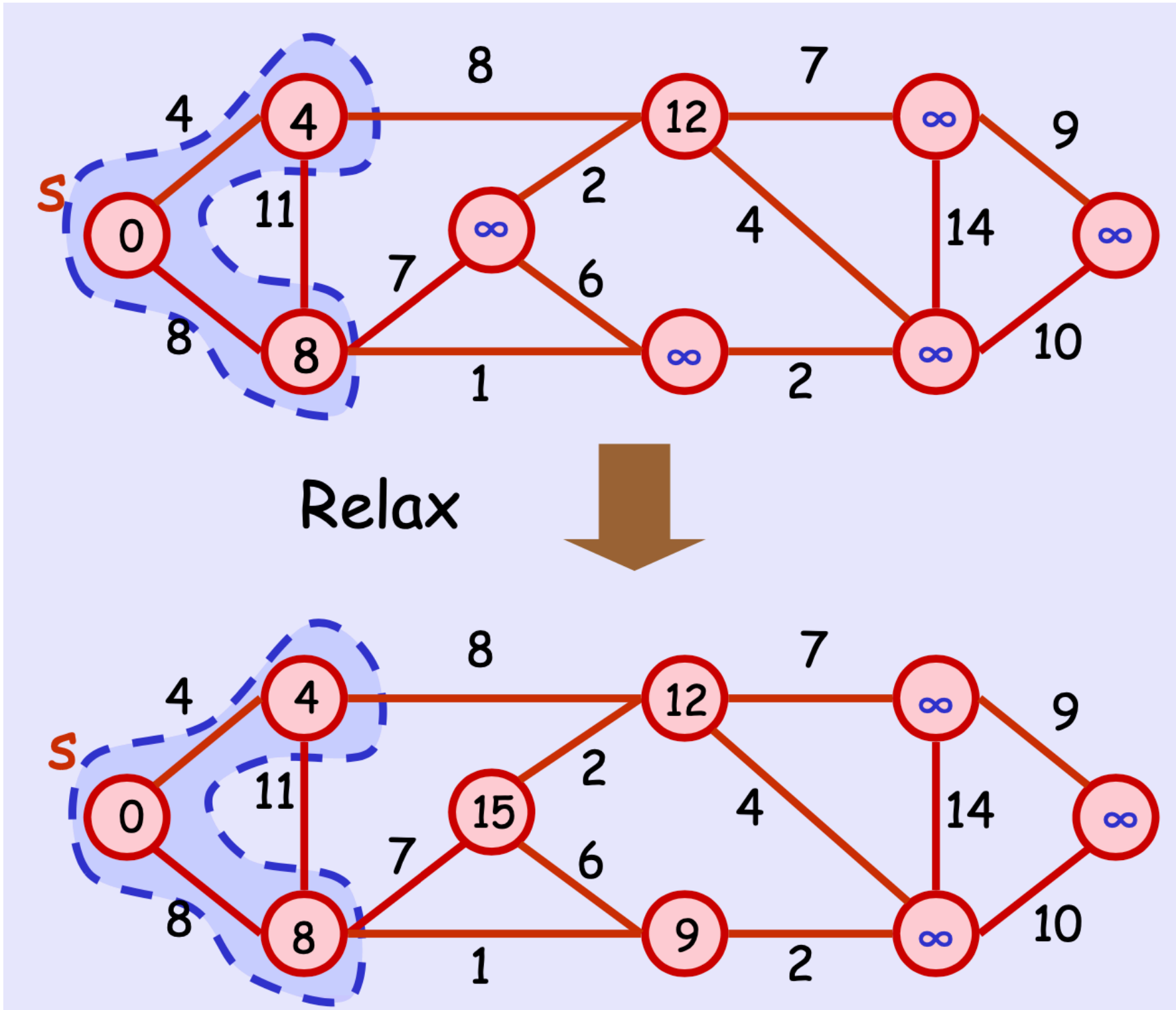
Example



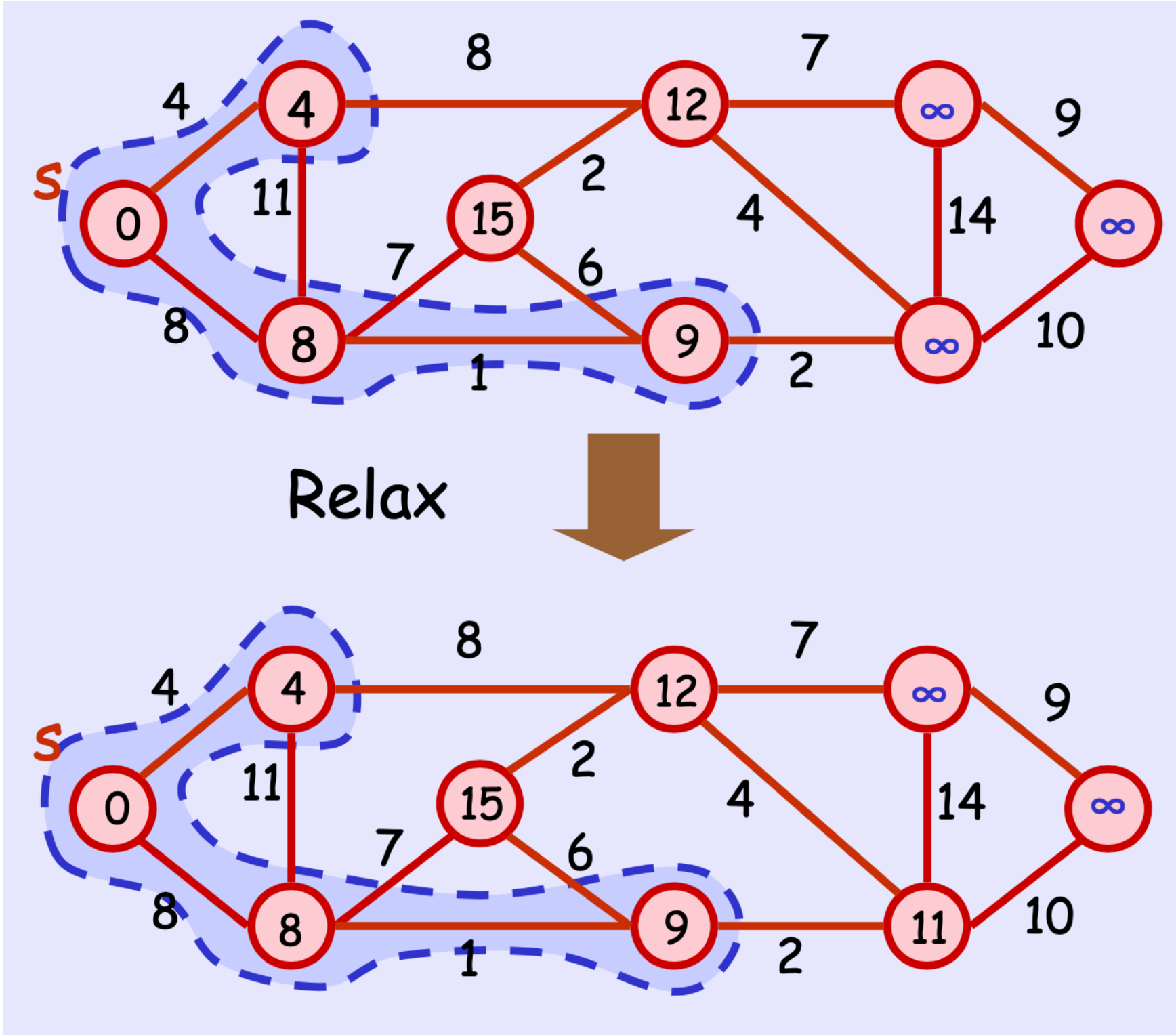
Example



Example

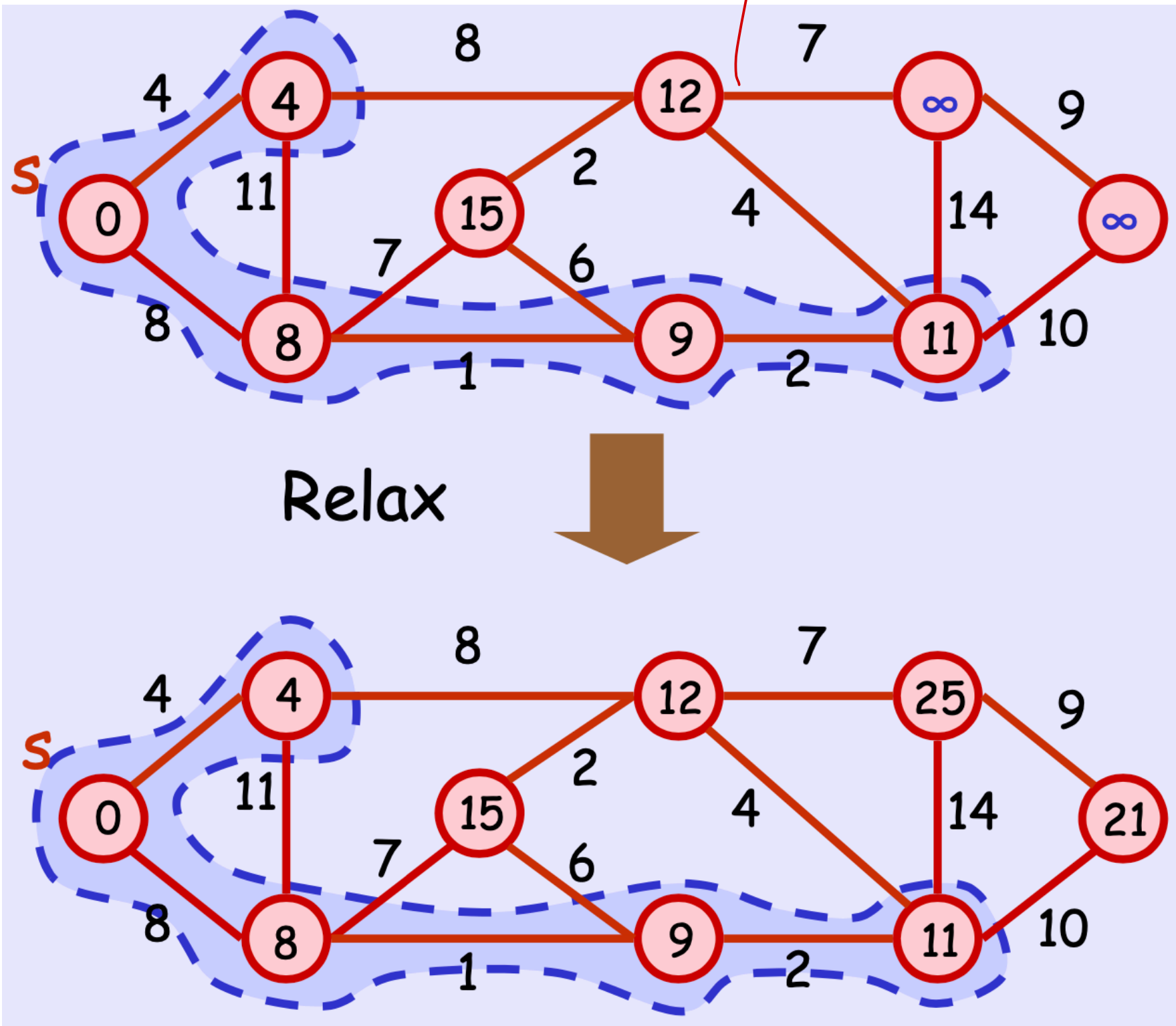

S
0
4
8
11
4
8
7
1
∞
2
6
12
∞
4
7
∞
2
∞
14
9
10
∞
Relax
S
0
4
8
11
4
8
7
1
15
2
6
12
9
4
7
∞
2
∞
14
9
10
∞

Example

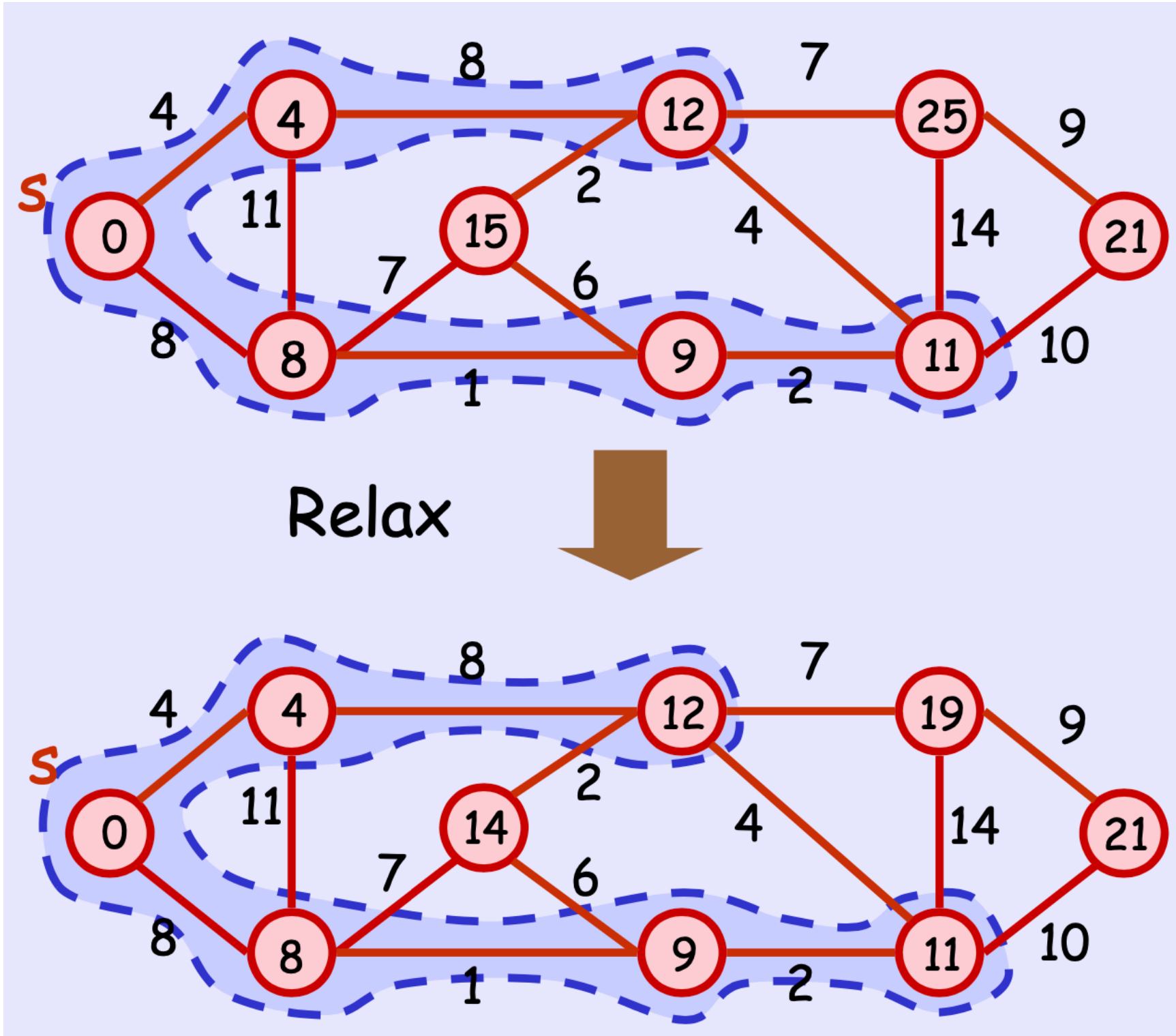

S
0
4
8
15
12
∞
9
∞
∞
4
8
11
7
8
1
2
6
9
2
4
7
14
10
Relax
S
0
4
8
15
12
∞
9
∞
11
4
8
11
7
8
1
2
6
9
2
4
7
14
10

Example

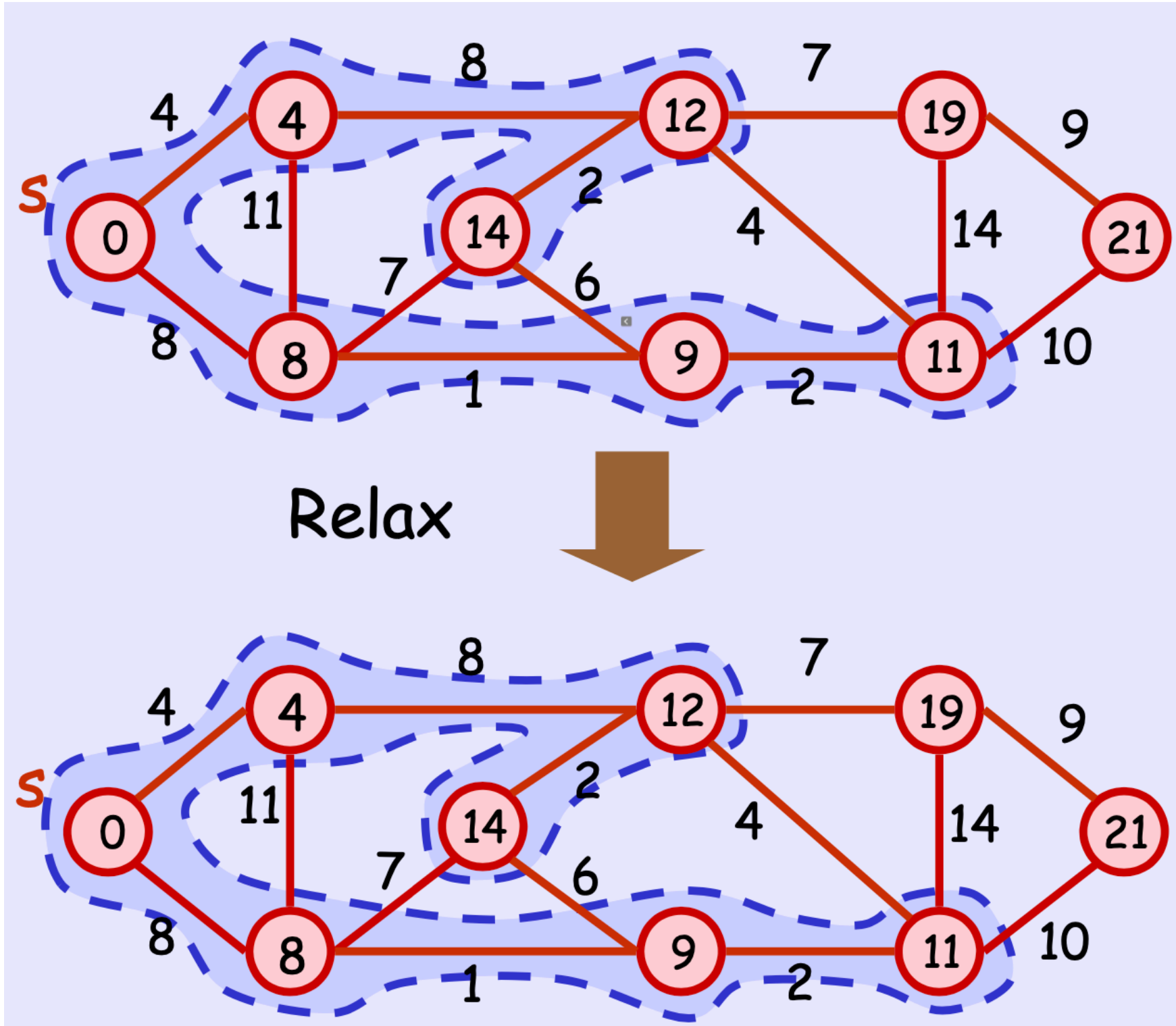
个为啥选8.



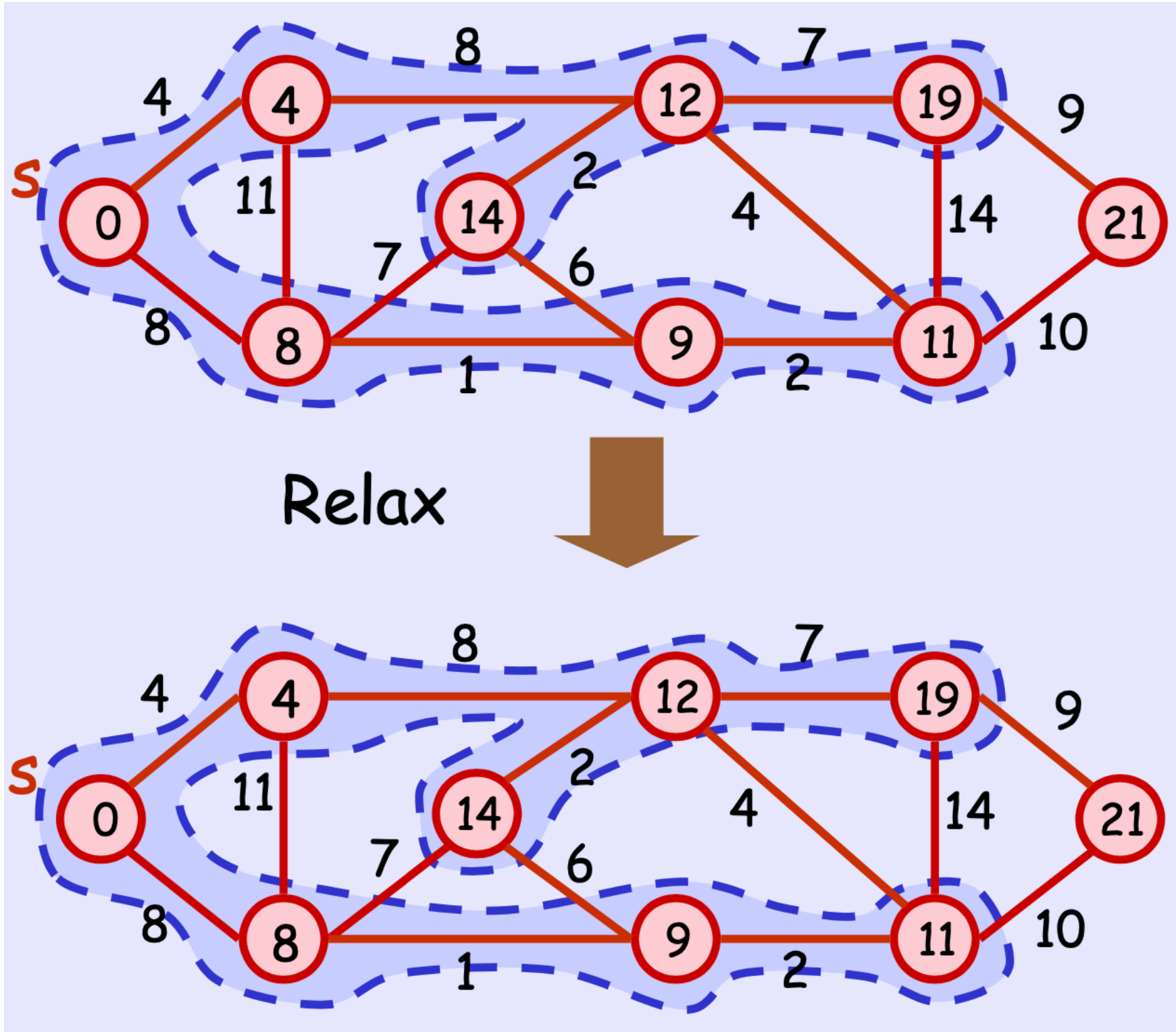
Example


s
0
4
8
15
12
25
21
11
9
4
8
11
7
2
6
1
8
7
4
14
10
2
Relax
s
0
4
8
14
12
19
21
11
9
4
8
11
7
2
6
1
8
7
4
14
10
2

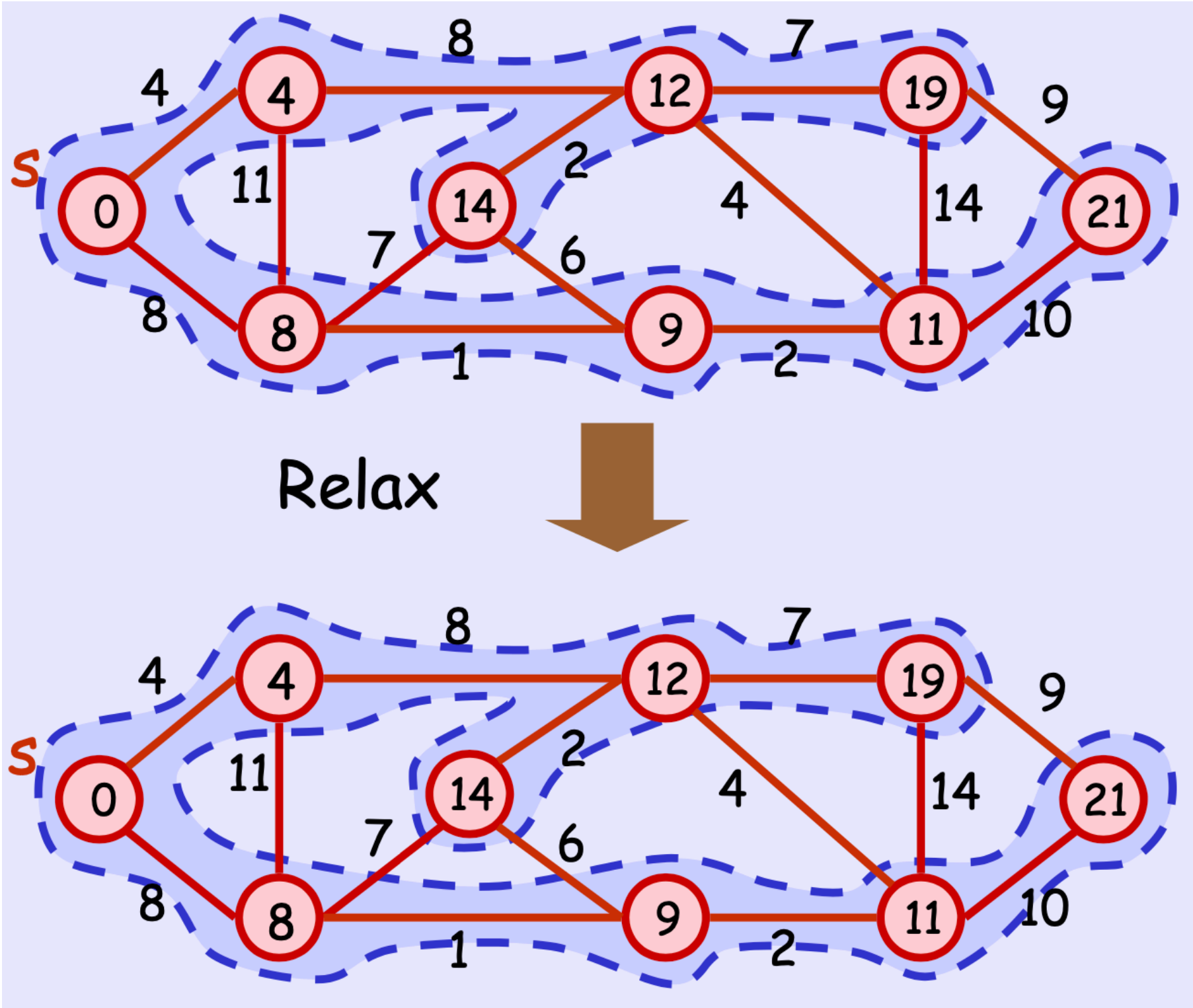
Example



Example


s
0
4
8
14
12
19
21
9
11
4
8
11
7
8
1
2
6
2
4
7
9
14
10
Relax
s
0
4
8
14
12
19
21
9
11
4
8
11
7
8
1
2
6
2
4
7
9
14
10

Example


s
0
4
8
12
19
21
14
9
11
4
8
7
9
11
2
4
14
7
6
8
1
2
10
Relax
s
0
4
8
12
19
21
14
9
11
4
8
7
9
11
2
4
14
7
6
8
1
2
10

```
Dijkstra(G, s)
    For each vertex v,
        Mark v as unvisited, and set d(v) = ∞ ;
    Set d(s) = 0 ;
    while (there is unvisited vertex) {
        v = unvisited vertex with smallest d ;
        Visit v, and Relax all its outgoing edges;
    }
    return d ;
```

- Theorem:
  - The kth vertex closest to the source s is selected at the kth step inside the while loop of Dijkstra's algorithm
  - Also, by the time a vertex v is selected, $d(v)$ will store the length of the shortest path from s to v

- How to prove ? (By induction)

Correctness of Dijkstra's Algorithm

- We need to show that when a vertex u is added to S, then $d[u] = \delta(s,u)$.
- For the purpose of contradiction, let u be the first vertex added to S for which $d[u] > \delta(s,u)$. Note that $u \neq s$.
- Let P be the shortest path from s to this u.
- Let x be the last vertex on P that is in S. Note that it is possible that $x = s$.
- Let (x,y) be the edge on P. Note that it is possible that $y = u$.
- When x was added to S, the edge (x,y) was relaxed,
- Therefore, $d[y] = \delta(s,x) + w(x,y) = \delta(s,y) \leq \delta(s,u) \leq d[u]$
- Neither u nor y are in S. Since u was added to S, we have $d[u] \leq d[y]$. It follows that $\delta(s,u) = d[u]$, a contradiction.

Performance

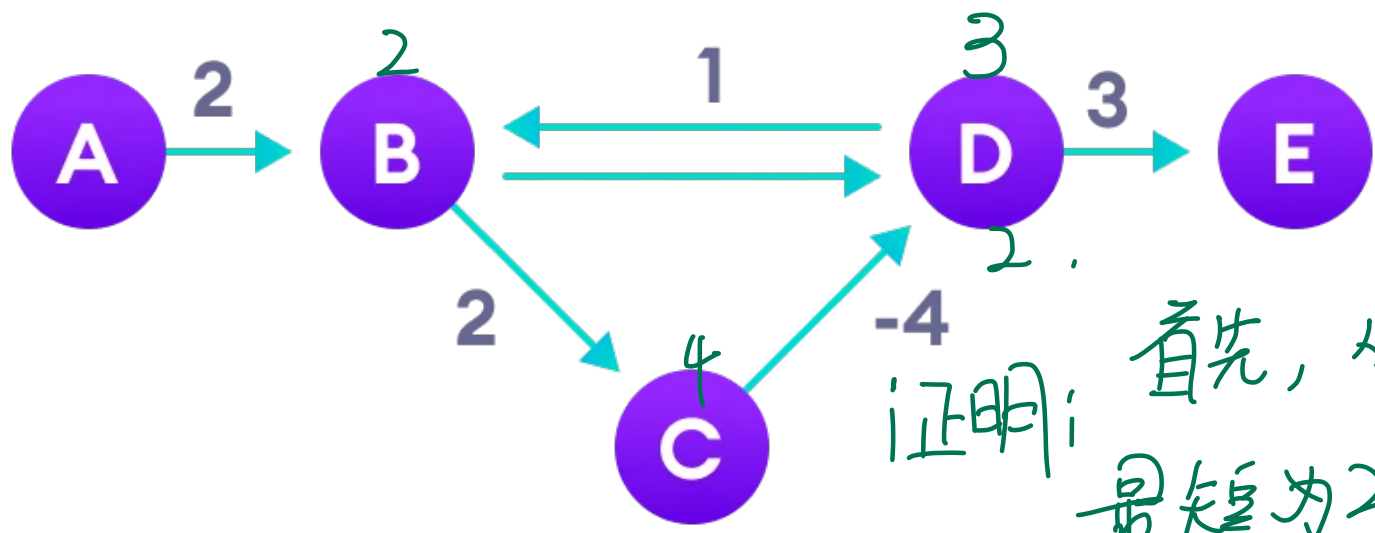
- Dijkstra's algorithm is similar to Prim's
- By simply stored (v) in the vth array.
  - Relax (Decrease-Key):O(1)
  - Pick vertex (Extract-Min):O(V)
- Running Time:
  - The cost of |V| operation Extract-Min is $O(|V|^2)$
  - At most O(|E|) Decrease-Key
  - Total Time: $O(|E| + |V|^2) = O(|V|^2)$

Performance

- By using binary Heap,
  - Relax ←→ Decrease-Key: O(log |V|)
  - Pick vertex ←→ Extract-Min: O (log |V|)
- RunningTime:
  - The cost of |V| operation Extract-Min is O(|V| log |V|)
  - At most O(|E|) Decrease-Key
  - Total Time: O((|E| + |V|) log |V|) = O(|E| log |V|)

Why do we need to be careful with negative weights?

- Shortest path algorithms like Dijkstra's Algorithm that aren't able to detect such a cycle can give an incorrect result because they can go through a negative weight cycle and reduce the path length.



证明：首先，从A→B，路径最短为2，而后由于ABD<ABC，因而走ABD，

D的最短路径为3，但 ABCD 居然 < ABD，

因而推翻了之前的结论。

- **Types of Shortest Path Problems**
- **Relaxation**
- **Bellman-Ford Algorithm**
- **Dijkstra's Algorithm**
- **Floyd-Warshall Algorithm**

All-Pairs Shortest Path Problem

- Find shortest paths between all pairs of vertices.
- Focus on finding minimum weights of paths. Shortest paths itself can be reconstructed if an appropriate predecessor matrix is used.
- Recall optimal substructure property of shortest paths.

通过矩阵，完成和
DK算法一样的功效，

Floyd-Warshall Algorithm

不适带负圈

- Floyd-Warshall Algorithm is an algorithm for finding the shortest path between all the pairs of vertices in a weighted graph.
- This algorithm works for both the directed and undirected weighted graphs.
- But, it does not work for the graphs with negative cycles (where the sum of the edges in a cycle is negative).

Transitive Closure of a Graph

- Alternatively, Floyd-Warshall computes the so-called "transitive closure" of a graph (Warshall's variant).
- The transitive closure extends a graph by edges between all *indirectly* connected pairs of nodes.
- For example, if the graph has two edges – one from A to B and one from B to C – then the transitive closure extends the graph by the edge from A to C (since a path from A to C via B exists).

↓ 核心思想：传递闭包；

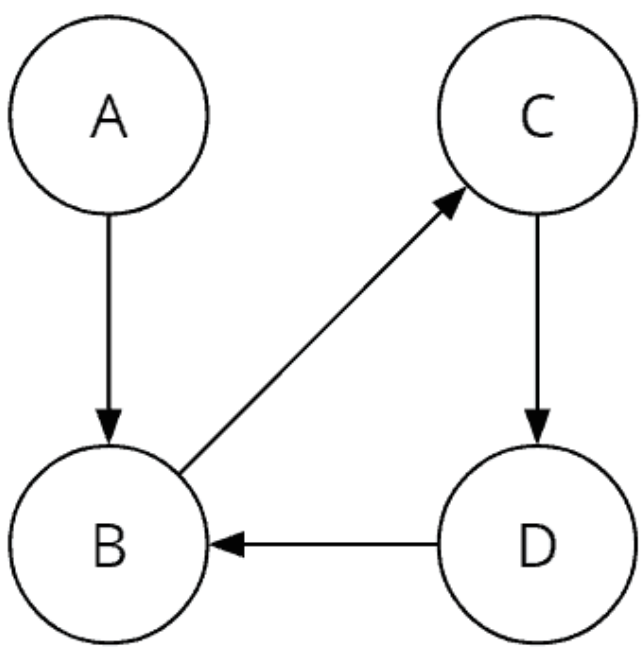
A 可到 B.

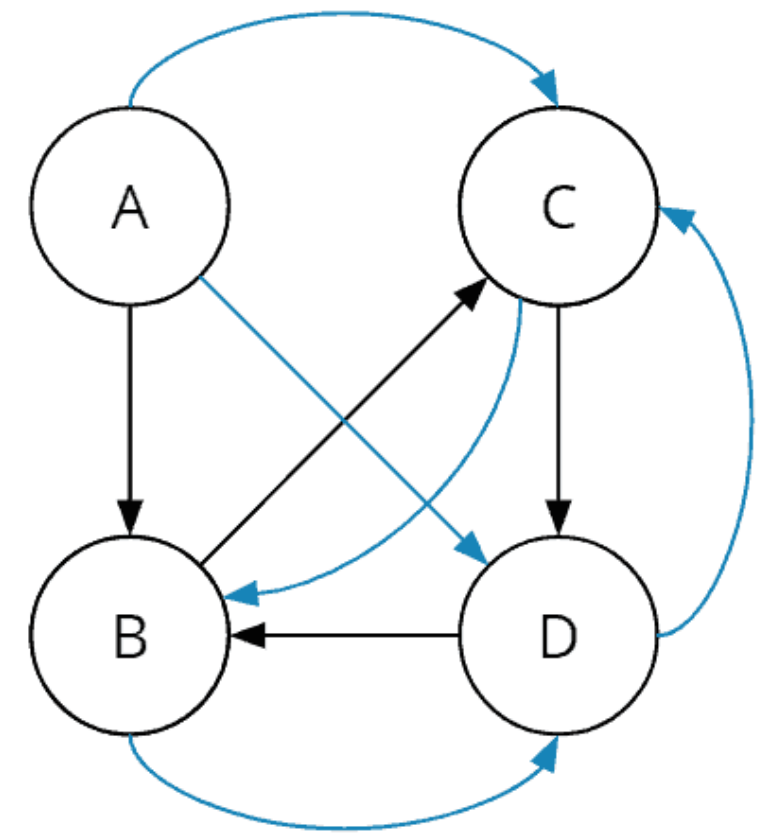
B 可到 C.

A → C

Transitive Closure of a Graph

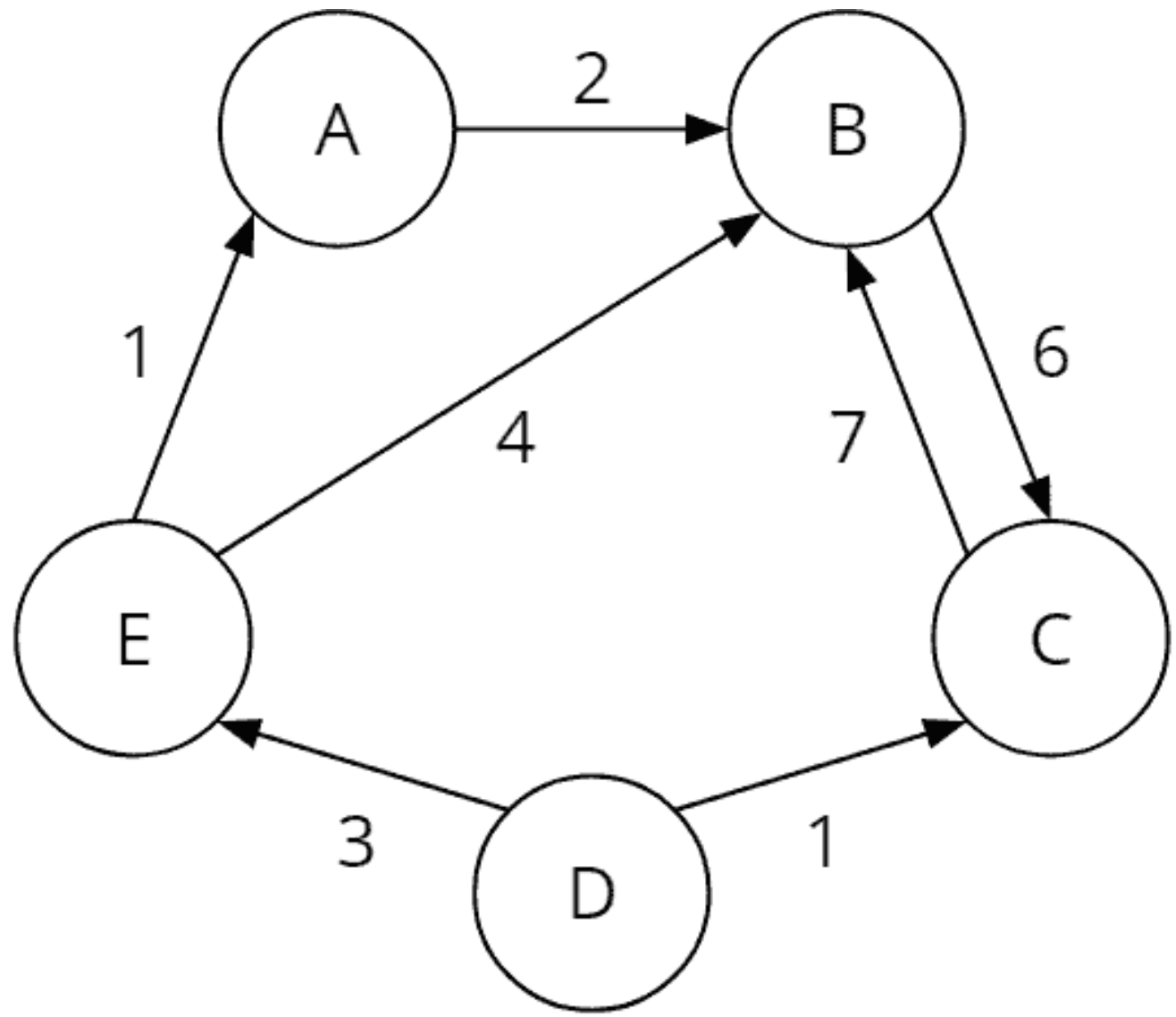
- The following graphic shows a somewhat more complex example with four nodes - the initial graph on the left and its transitive closure on the right.
- The blue arrows represent the added, indirect connections:





Floyd-Warshall Algorithm – Example

- The following example graph contains five nodes, labeled A, B, C, D, E, and various directed and weighted edges:



Floyd-Warshall Algorithm – Example

- Preparation – Node Pair Matrix
  - In preparation, we create an $n \times n$ matrix (n is the number of nodes) in which we enter – for each pair of nodes (i, j) – the weight of the edge from i to j if it exists. Otherwise, we enter *infinity* (∞).
  - On the diagonal (the distance of a node to itself), we enter 0.

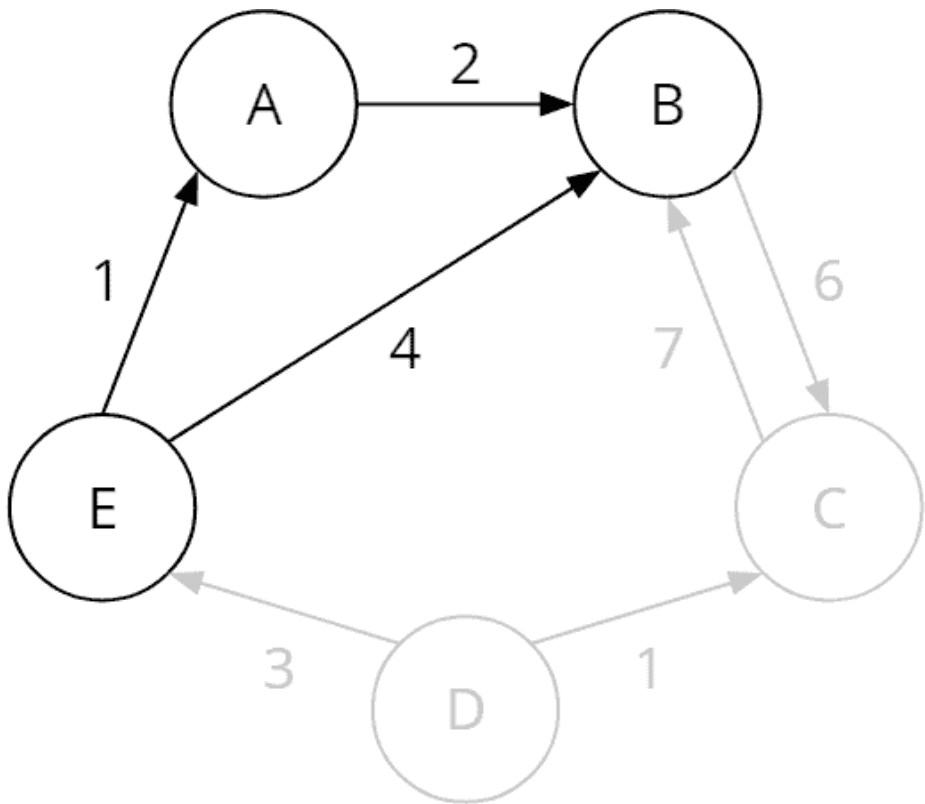
	A	**B**	**C**	**D**	**E**
A	0	2	∞	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	∞	∞	1	0	3
E	1	4	∞	∞	0

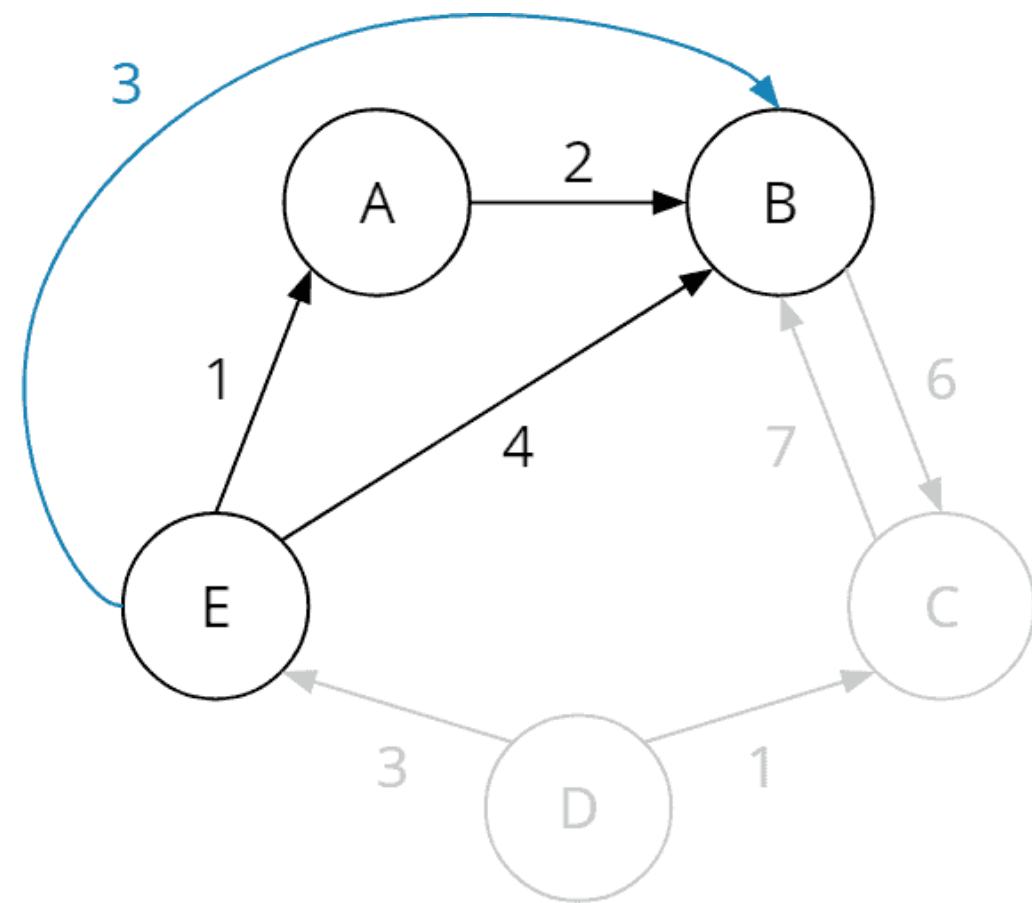
From the table, we can read, for instance:
The cost from A to B is 2 (row A, column B).

行A 列B: A→B,
行B 列A: B→A.

Floyd-Warshall Algorithm – Step by Step

- Iteration 1 – Indirect Paths **via** Intermediate Node A

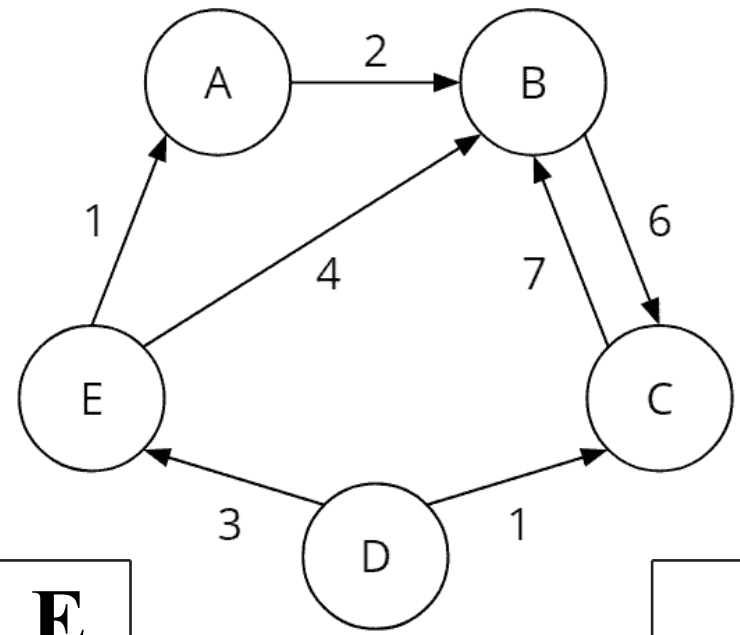




Iteration 1:
Comparing paths E→B and E→A→B
Path E→B→A is shorter than E→B

Floyd-Warshall Algorithm – Step by Step

- Iteration 1 – Indirect Paths **via** Intermediate Node **A**



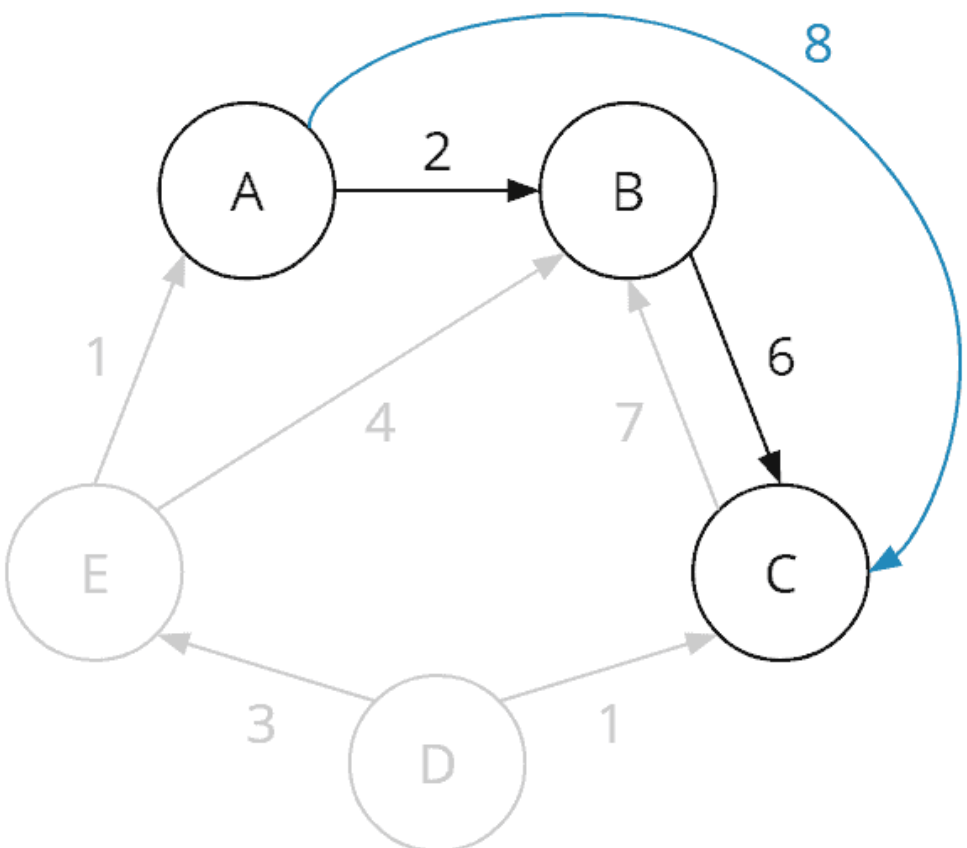
	A	B	C	D	E
A	***0***	***2***	∞	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	∞	∞	1	0	3
E	***1***	4	∞	∞	0

Iteration 1 **via** Node A

	A	B	C	D	E
A	0	2	∞	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	∞	∞	1	0	3
E	1	**3**	∞	∞	0

Floyd-Warshall Algorithm – Step by Step

- Iteration 2 – Indirect Paths **via** Intermediate Node **B**

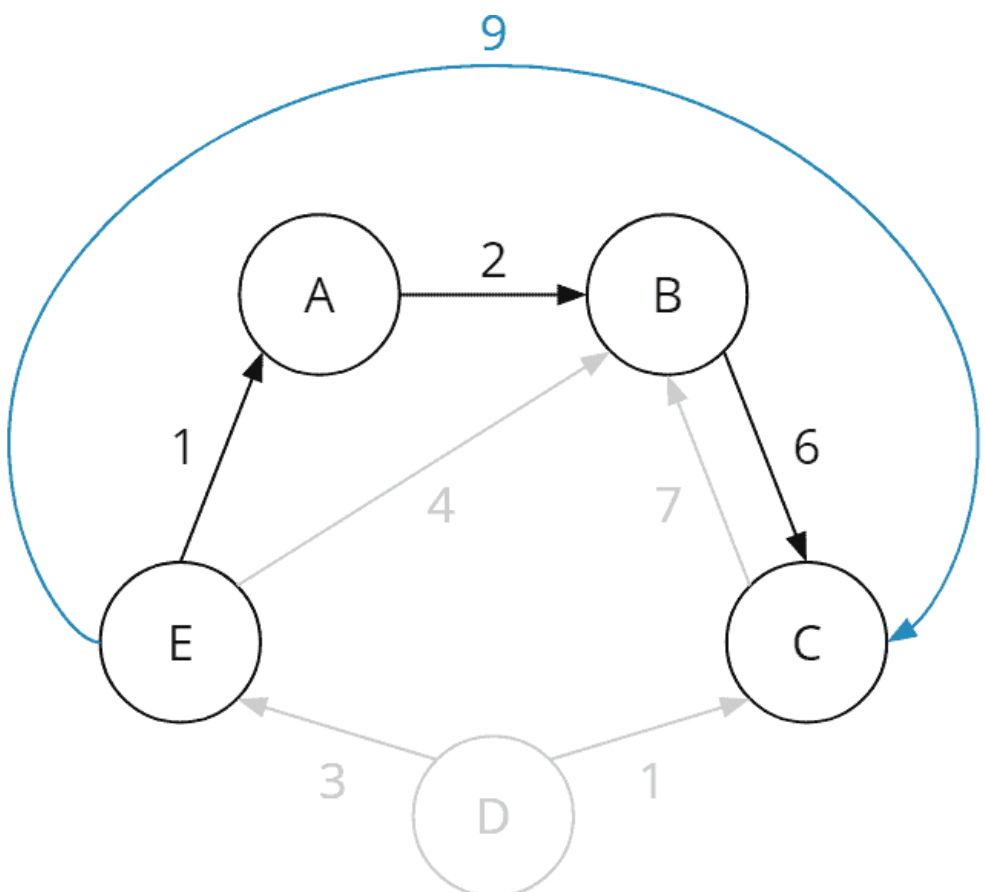


	A	B	C	D	E
A	0	**2**	∞	∞	∞
B	∞	**0**	**6**	∞	∞
C	∞	**7**	0	∞	∞
D	∞	∞	1	0	3
E	1	**3**	∞	∞	0

	A	B	C	D	E
A	0	2	**8**	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	∞	∞	1	0	3
E	1	3	∞	∞	0

Floyd-Warshall Algorithm – Step by Step

- Iteration 2 – Indirect Paths **via** Intermediate Node **B**

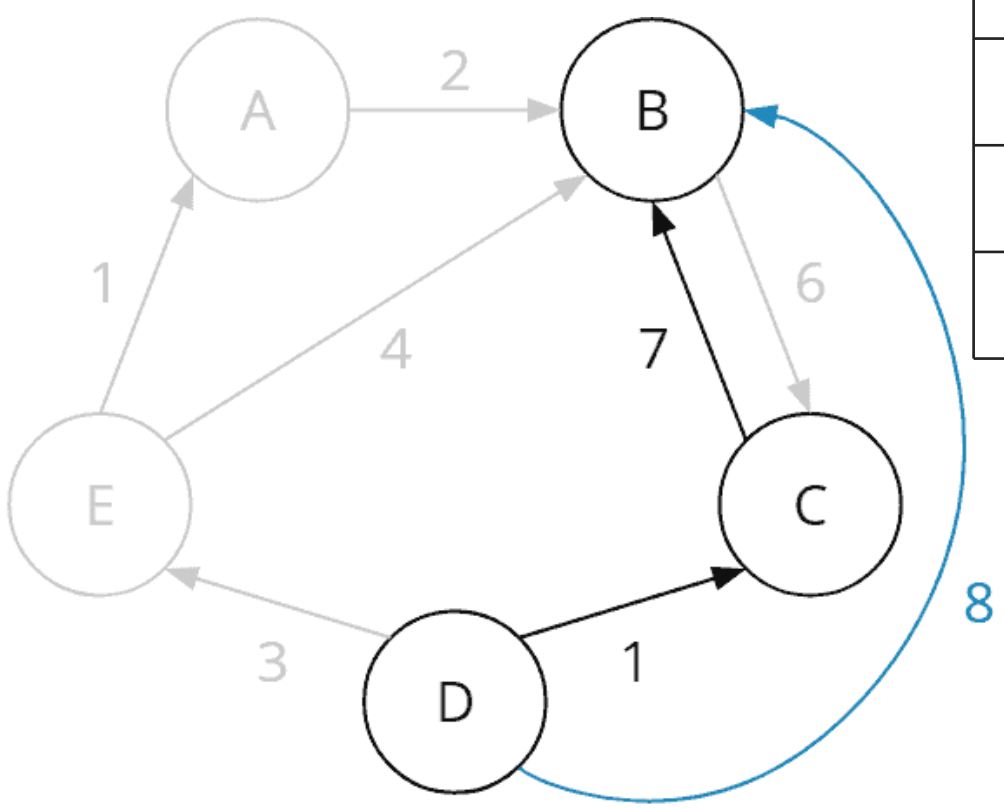


	A	B	C	D	E
A	0	**2**	**8**	∞	∞
B	∞	**0**	**6**	∞	∞
C	∞	**7**	0	∞	∞
D	∞	∞	1	0	3
E	1	**3**	∞	∞	0

	A	B	C	D	E
A	0	2	**8**	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	∞	∞	1	0	3
E	1	3	**9**	∞	0

Floyd-Warshall Algorithm – Step by Step

- Iteration 3 – Indirect Paths **via** Intermediate Node **C**



	A	B	C	D	E
A	0	2	**8**	∞	∞
B	∞	0	**6**	∞	∞
C	∞	**7**	**0**	∞	∞
D	∞	∞	**1**	0	3
E	1	3	**9**	∞	0

	A	B	C	D	E
A	0	2	**8**	∞	∞
B	∞	0	**6**	∞	∞
C	∞	**7**	**0**	∞	∞
D	∞	**8**	**1**	0	3
E	1	3	**9**	∞	0

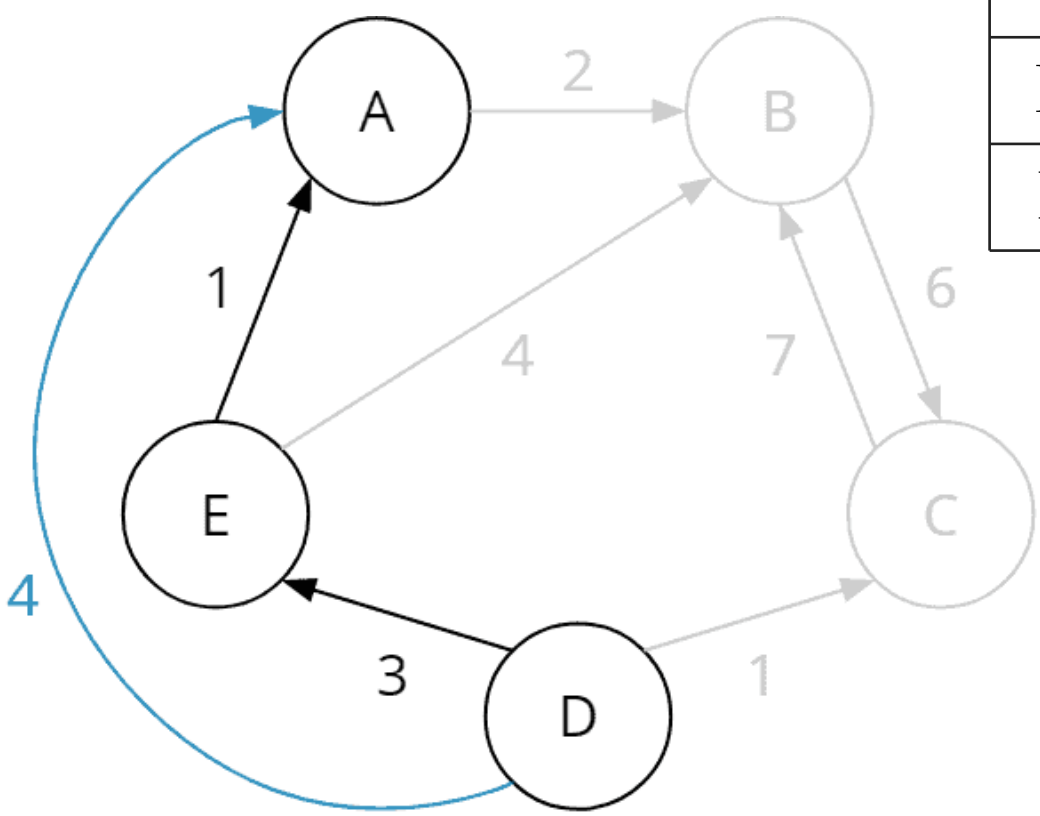
Floyd-Warshall Algorithm – Step by Step

- Iteration 4 – Indirect Paths **via** Intermediate Node **D**
  - We can abbreviate iteration 4:
    - There is no path from any node to intermediate node D. Thus, we will not find a route via D for any node pair.

	A	**B**	**C**	**D**	**E**
A	0	2	8	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	∞	**8**	**1**	**0**	**3**
E	1	3	9	∞	0

Floyd-Warshall Algorithm – Step by Step

- Iteration 5 – Indirect Paths **via** Intermediate Node **E**

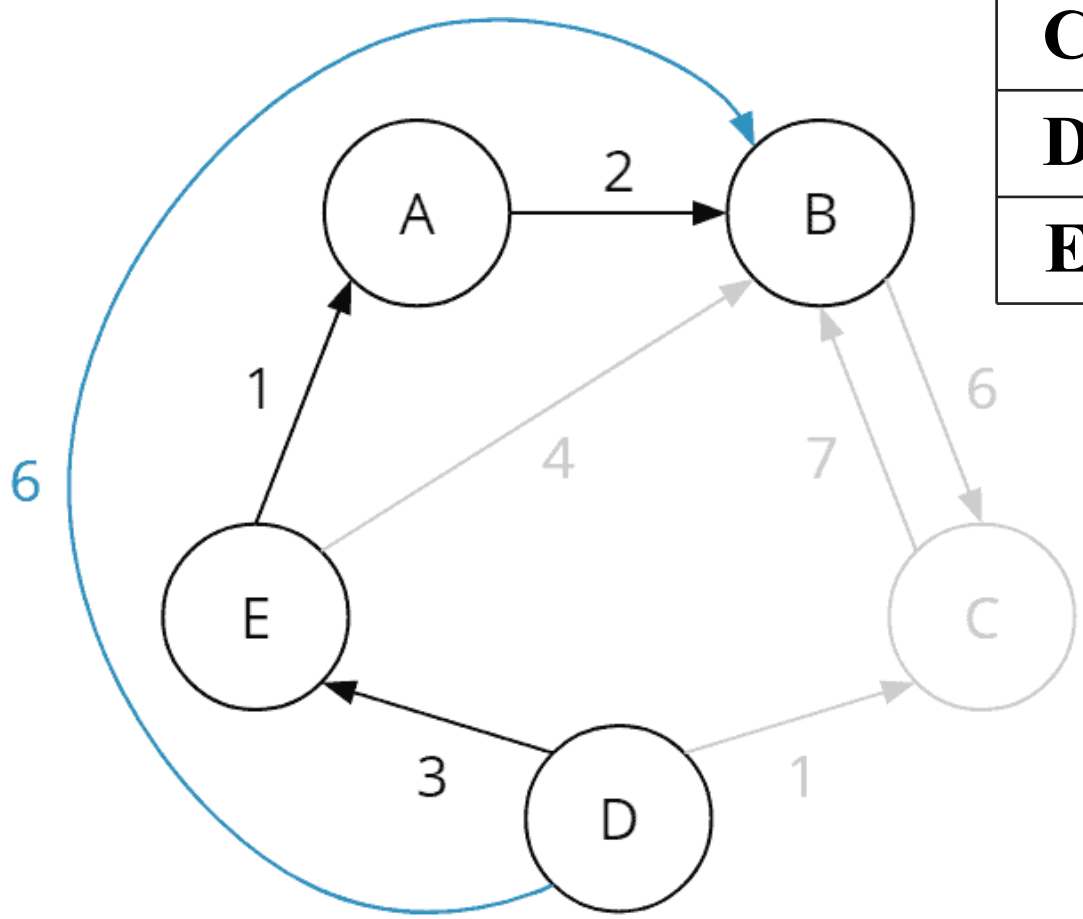


	A	B	C	D	E
A	0	2	8	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	∞	8	1	0	**3**
E	**1**	**3**	**9**	∞	**0**

	A	B	C	D	E
A	0	2	8	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	**4**	8	1	0	**3**
E	**1**	**3**	**9**	∞	**0**

Floyd-Warshall Algorithm – Step by Step

- Iteration 5 – Indirect Paths **via** Intermediate Node **E**

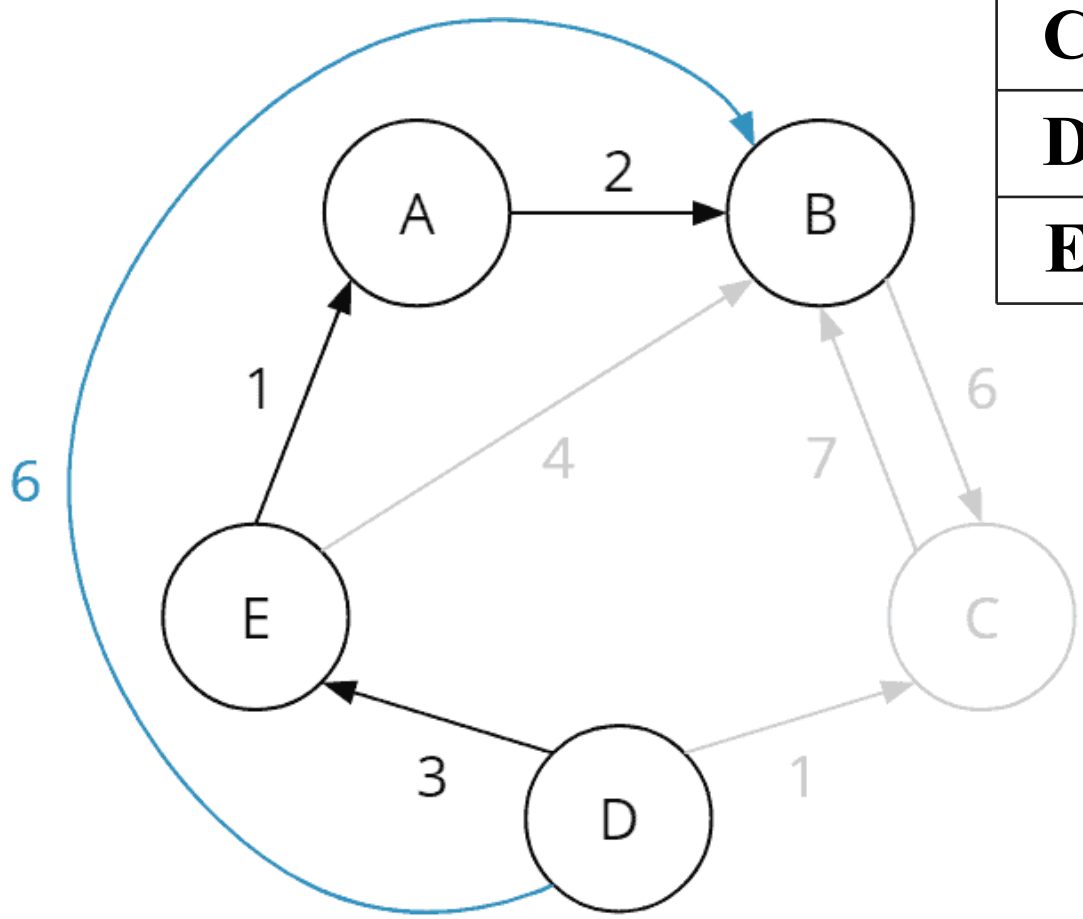


	A	B	C	D	E
A	0	2	8	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	**4**	8	1	0	**3**
E	**1**	**3**	**9**	∞	**0**

	A	B	C	D	E
A	0	2	8	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	**4**	**6**	1	0	**3**
E	**1**	**3**	**9**	∞	**0**

Floyd-Warshall Algorithm – Step by Step

- Iteration 5 – Indirect Paths **via** Intermediate Node **E**



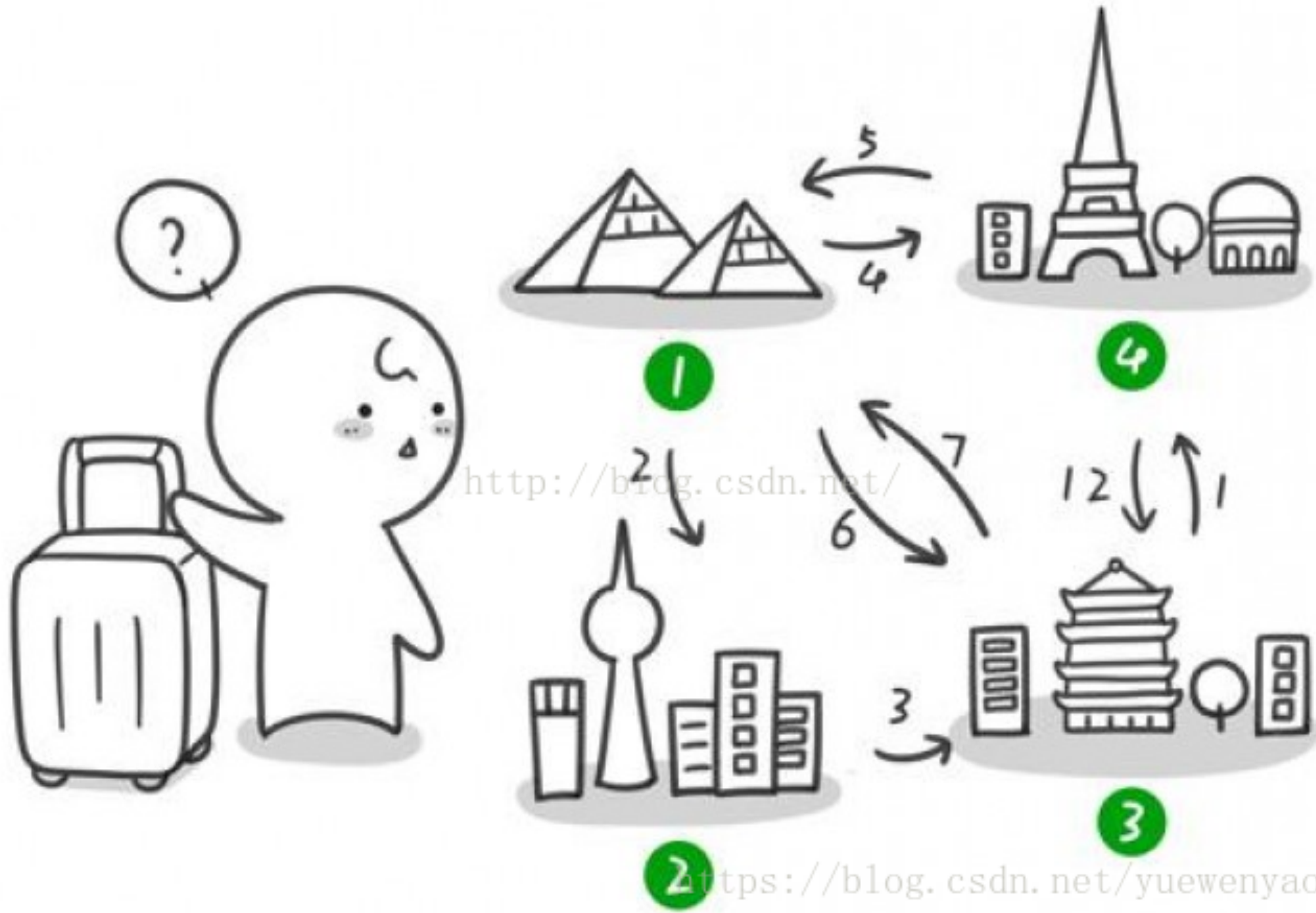
	A	B	C	D	E
A	0	2	8	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	**4**	**6**	1	0	**3**
E	**1**	**3**	**9**	∞	**0**

	A	B	C	D	E
A	0	2	8	∞	∞
B	∞	0	6	∞	∞
C	∞	7	0	∞	∞
D	**4**	**6**	1	0	**3**
E	**1**	**3**	**9**	∞	**0**

Floyd-Warshall Algorithm – Step by Step

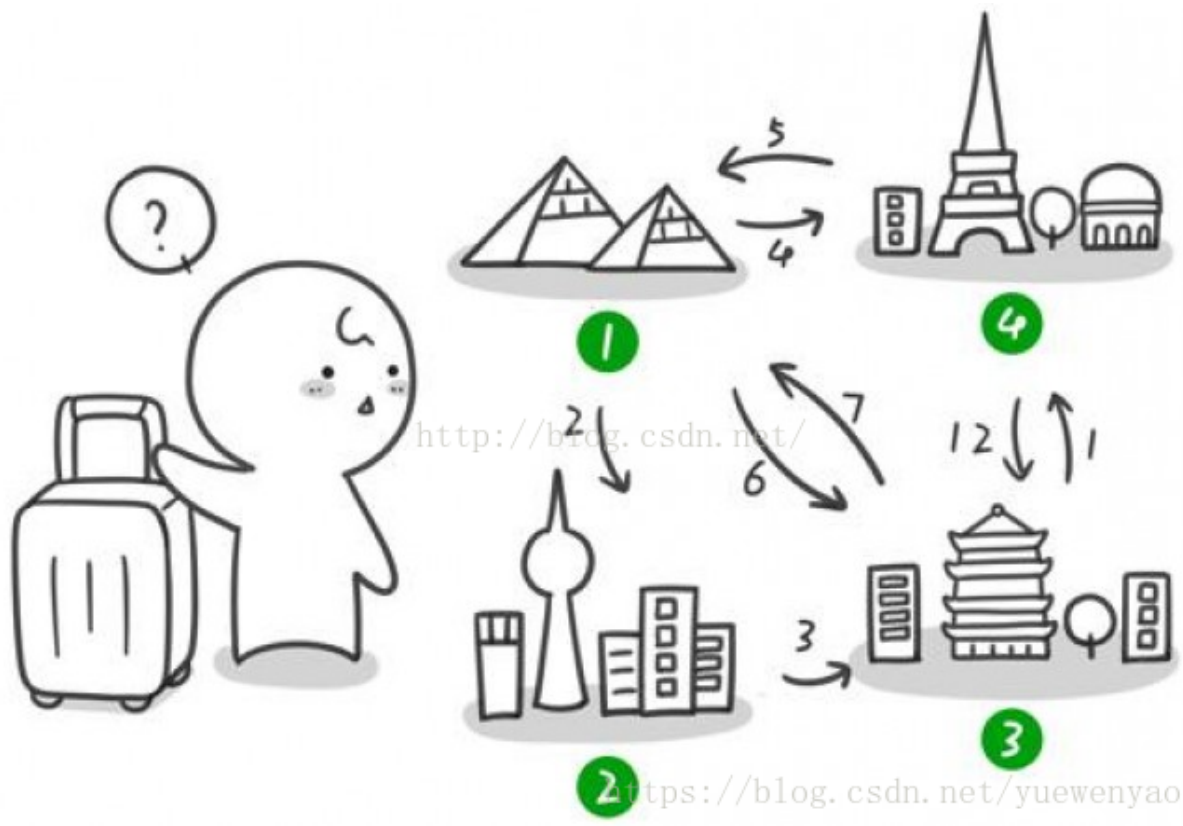
- We have reached the end of the fifth iteration and now know the lowest cost for all node pairs if we also allow indirect paths via node E (and thus also via A, B, C, D) – that is, via any other nodes.
- The goal of the algorithm is thus achieved.

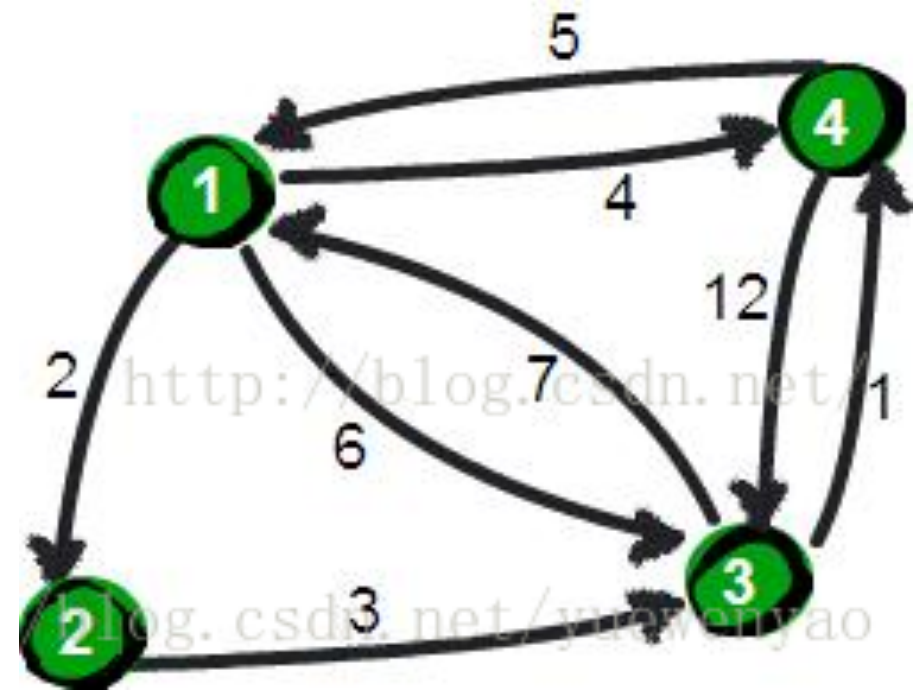
Example



http://blog.csdn.net/
https://blog.csdn.net/yuewenyao

Example



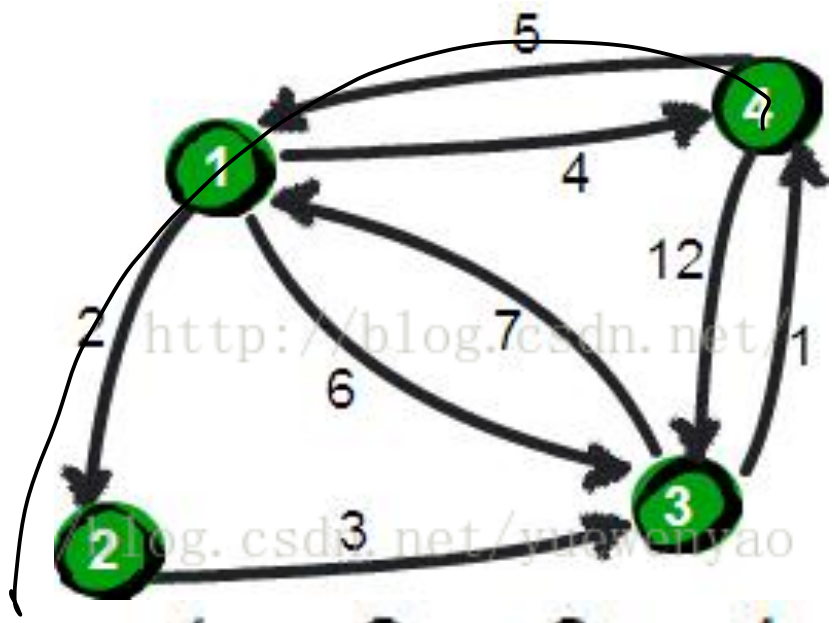


	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	∞	0	1
4	5	∞	12	0

http://blog.csdn.net/
https://blog.csdn.net/yuewenyao
http://blog.csdn.net/
blog.csdn.net/yuewenyao
.csdn.net/yuewenyao

Example

- Iteration 1 – Indirect Paths **via** Intermediate Node **1**



	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	∞	0	1
4	5	∞	12	0

	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	9	0	1
4	5	7	11	0

http://blog.csdn.net/yuewenyao

Example

- Iteration 1 – Indirect Paths **via** Intermediate Node **1**

```
for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
        if ( e[i][j] > e[i][1]+e[1][j] )
            e[i][j] = e[i][1]+e[1][j];
```

	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	∞	0	1
4	5	∞	12	0

.csdn.net/yuewenyao

	1	2	3	4
1	0	2	6	4
2	∞	0	3	∞
3	7	9	0	1
4	5	7	11	0

.csdn.net/yuewenyao

Example

- Iteration 3 – Indirect Paths **via** Intermediate Node **3**

	1	2	3	4
1	0	2	5	4
2	10	0	3	4
3	7	9	0	1
4	5	7	10	0

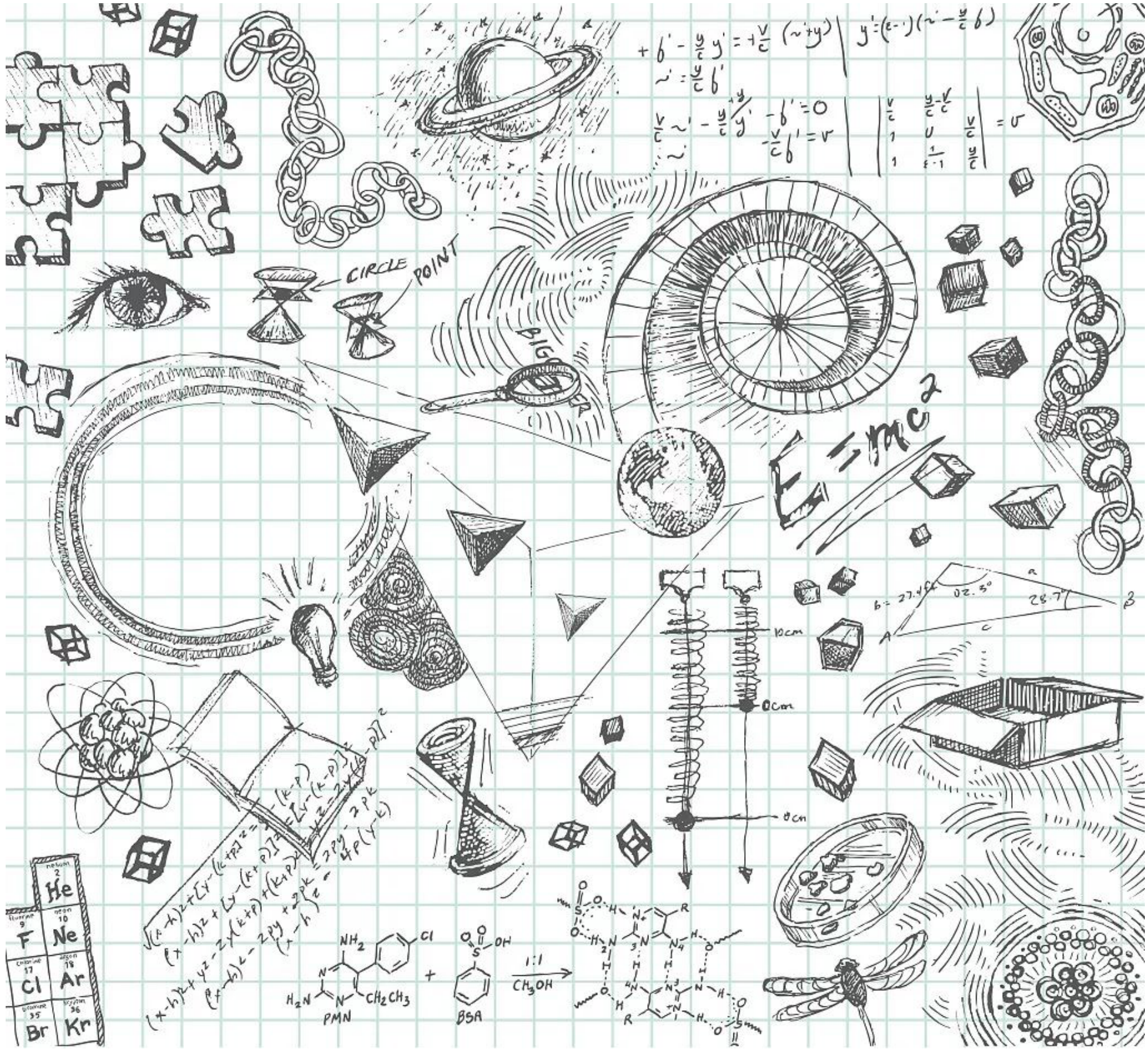
- Iteration 4 – Indirect Paths **via** Intermediate Node **4**

	1	2	3	4
1	0	2	5	4
2	9	0	3	4
3	6	8	0	1
4	5	7	10	0

csdn.net/yuewenyao

Floyd-Warshall Algorithm

```
n = no. of vertices
A = matrix of dimension n*n
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            A^k[i, j] = min (A^(k-1)[i, j], A^(k-1)[i, k] + A^(k-1)[k, j])
return A
```


CIRCLE
POINT
BIG
E=mc²
10cm
0cm
8cm
He
F
Ne
Cl
Ar
Br
Kr
PMN
BSA
1:1
CH₃OH