UCSC Silicon Valley Extension

Advanced C Programming

Shortest Paths Algorithms

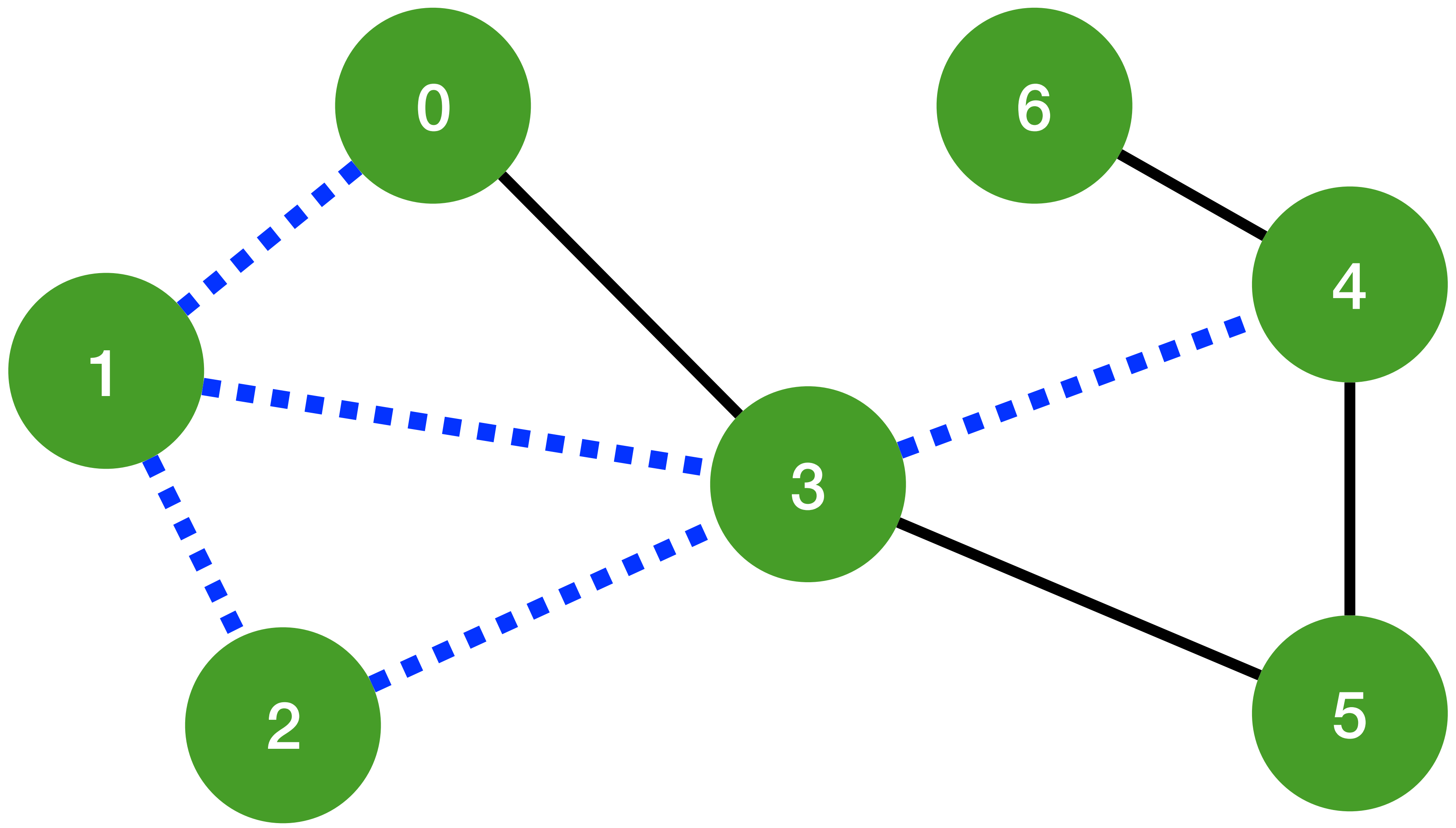
Instructor: Radhika Grover

Overview

- Shortest-paths properties
- Dijkstra’s algorithm
- A* algorithm
- Grid graphs

Walks

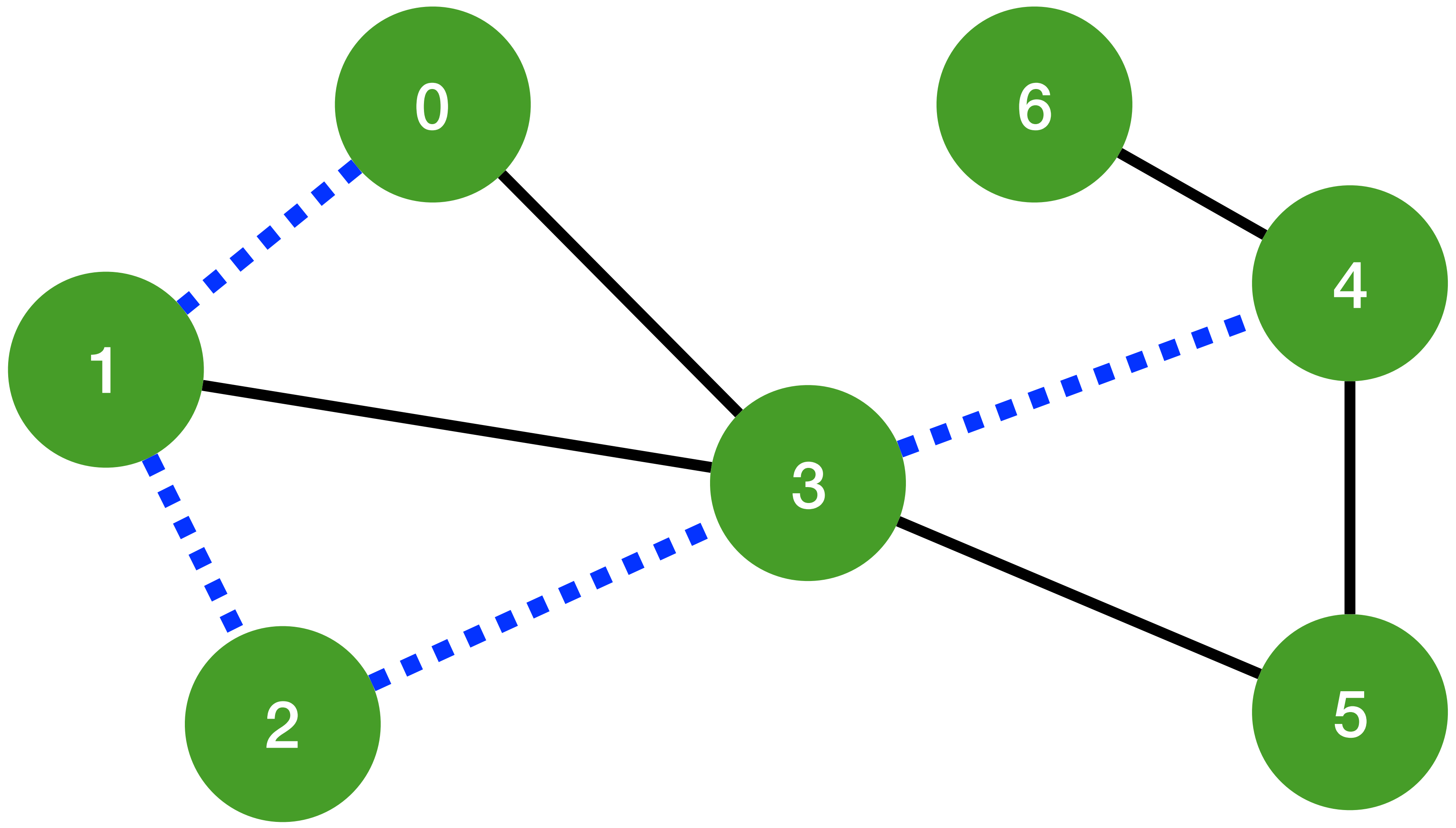
Any route from vertex to vertex



0 → 1 → 2 → 3 → 1 → 2 → 3 → 4

Paths

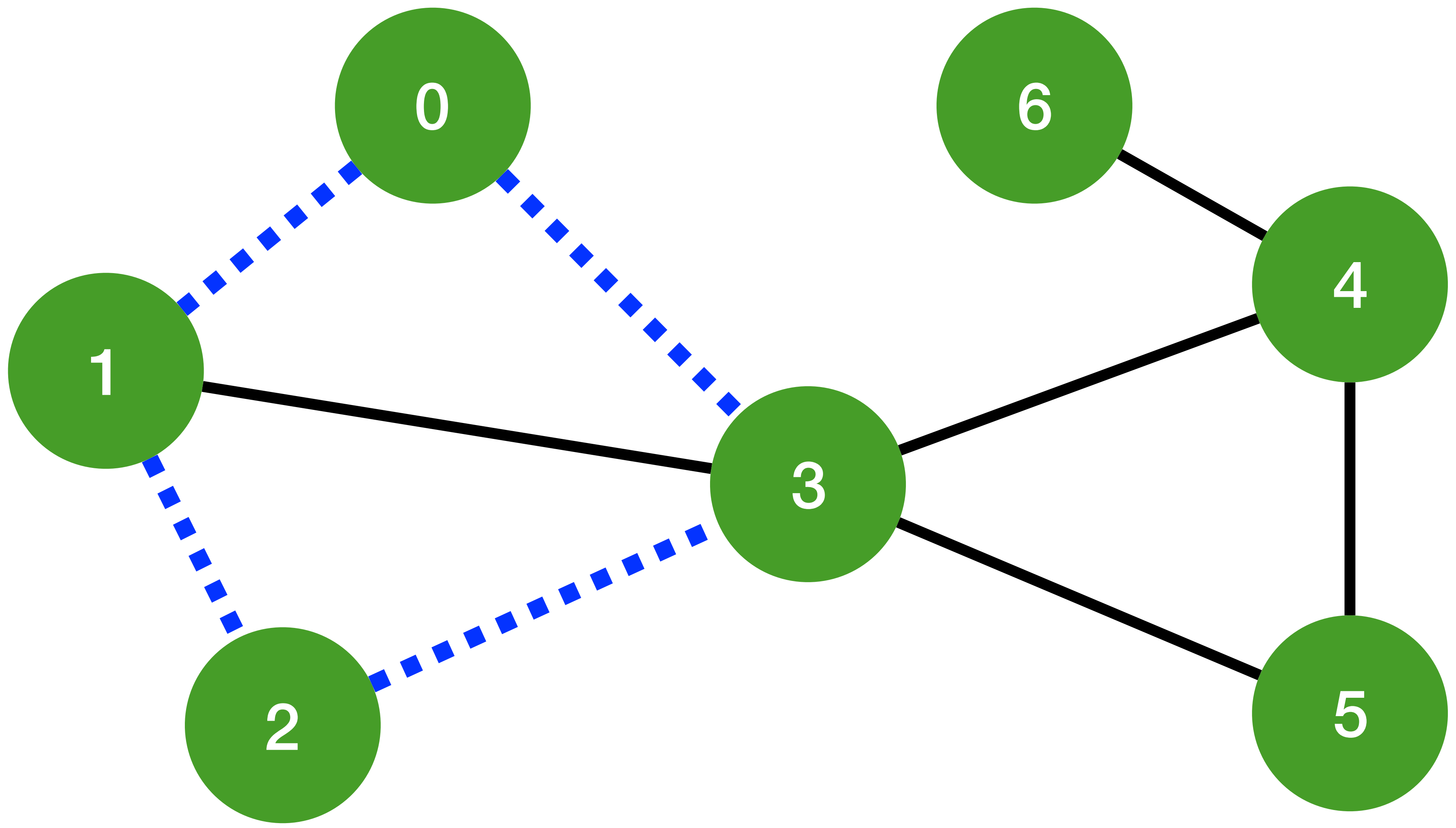
A walk that does not include any vertex twice (but starting vertex may be same as ending vertex)



0 → 1 → 2 → 3 → 4

Cycle

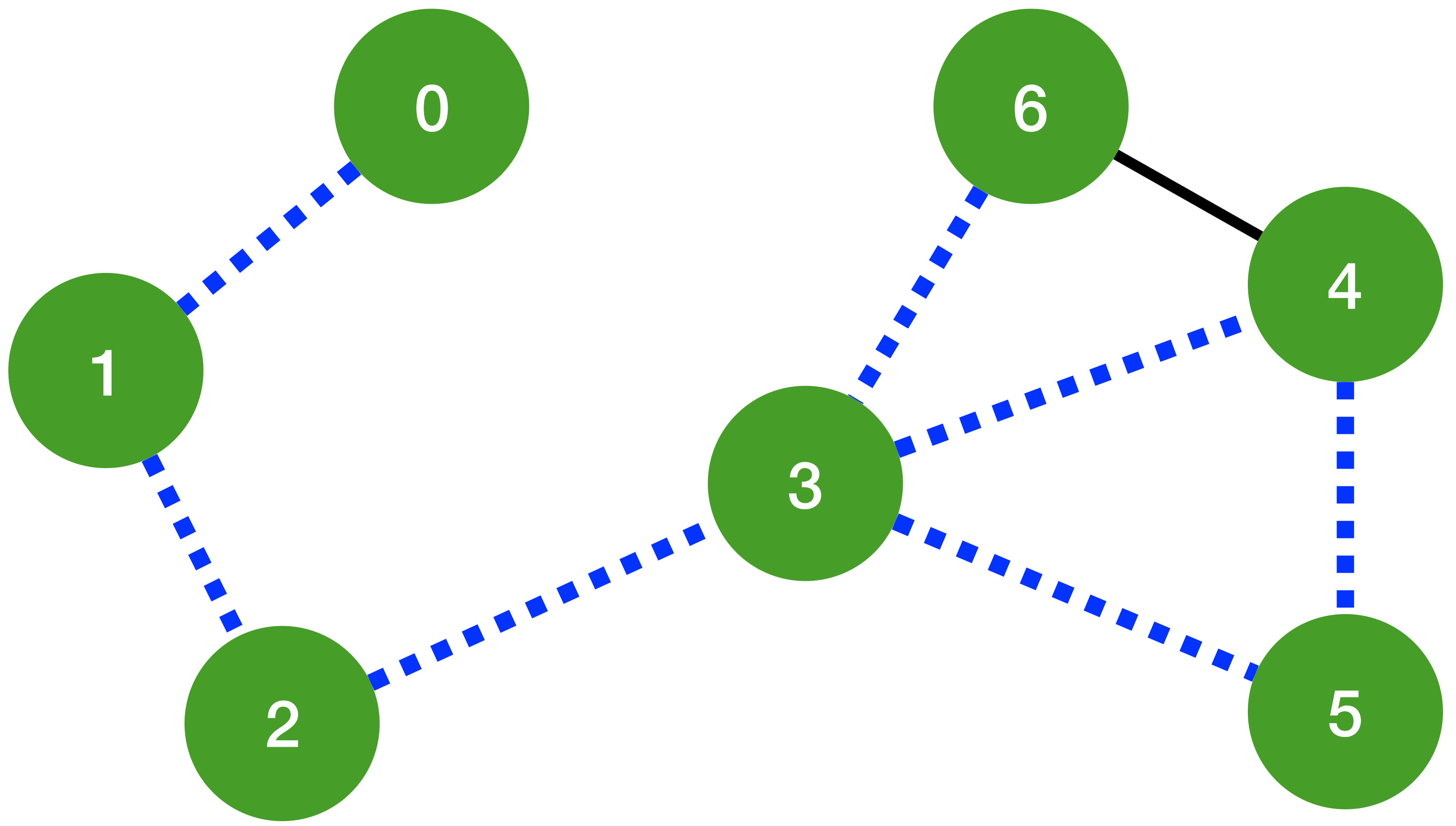
A path that begins and ends on the same vertex



0 → 1 → 2 → 3 → 0

Trail

A walk that does not pass over same edge twice

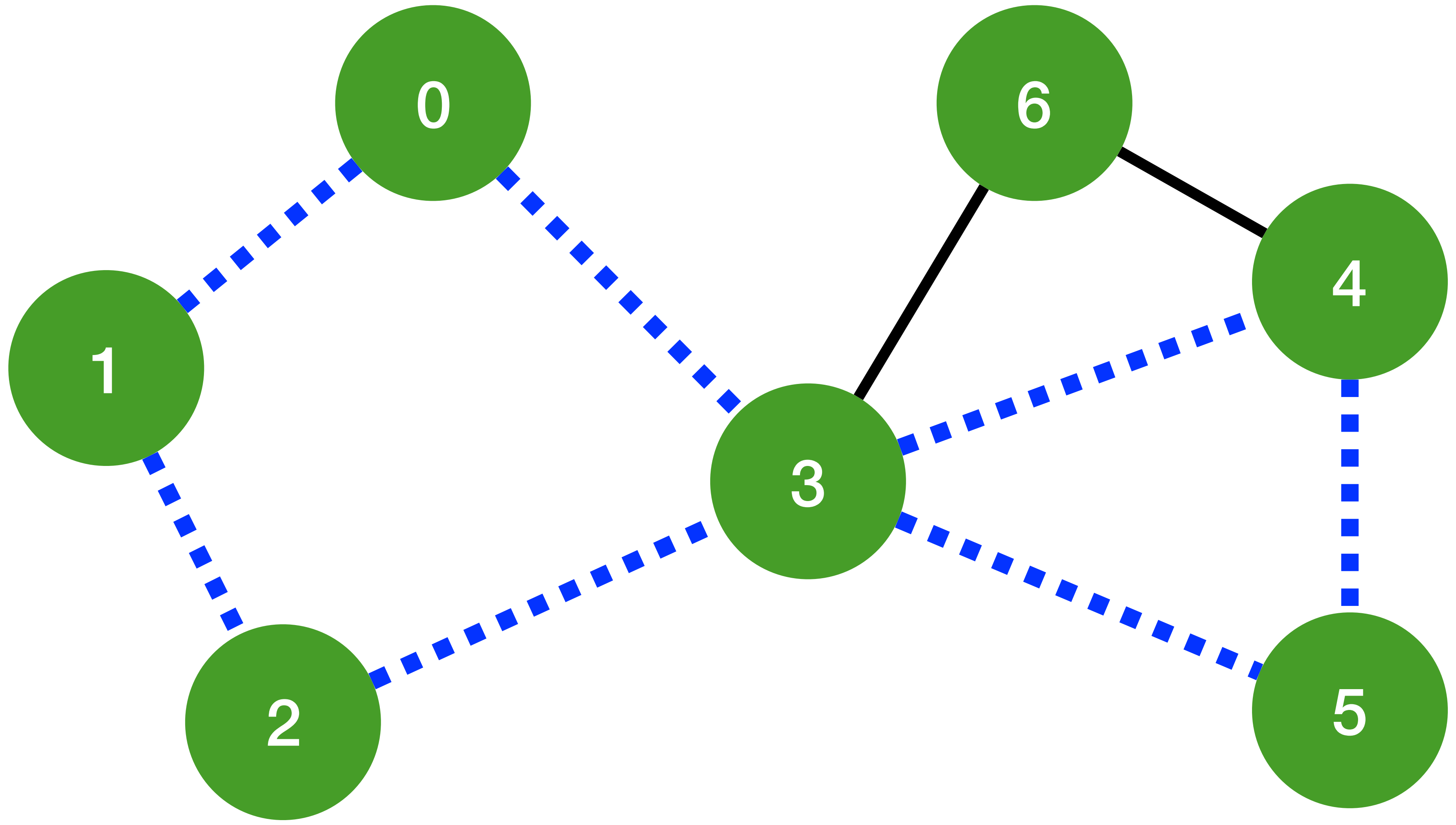


$0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 6$

Vertex 3 is visited twice

Circuit

A trail that begins and ends on the same vertex

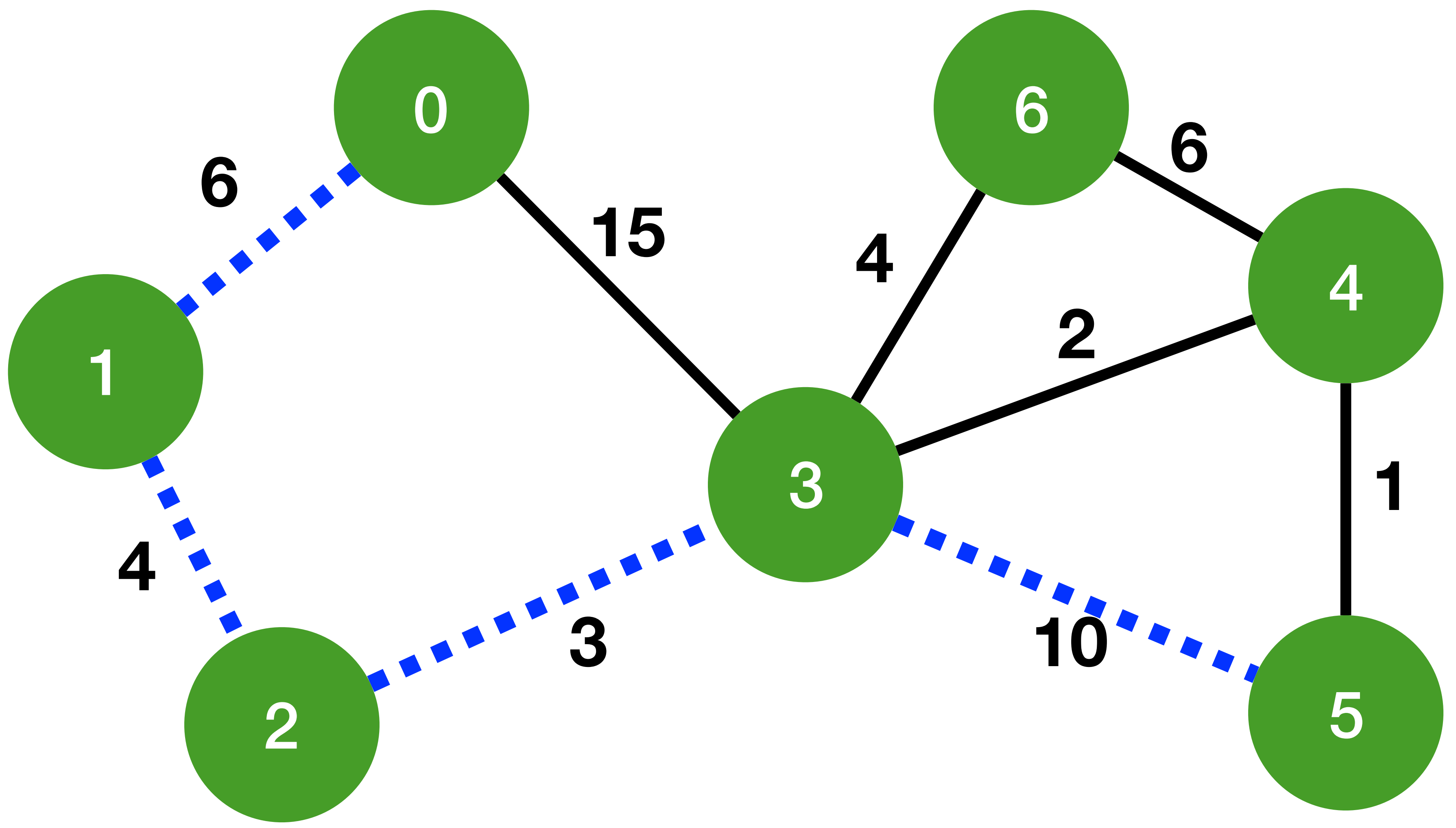


0 → 1 → 2 → 3 → 4 → 5 → 3 → 0

Vertex 3 is visited twice

Length of a Path

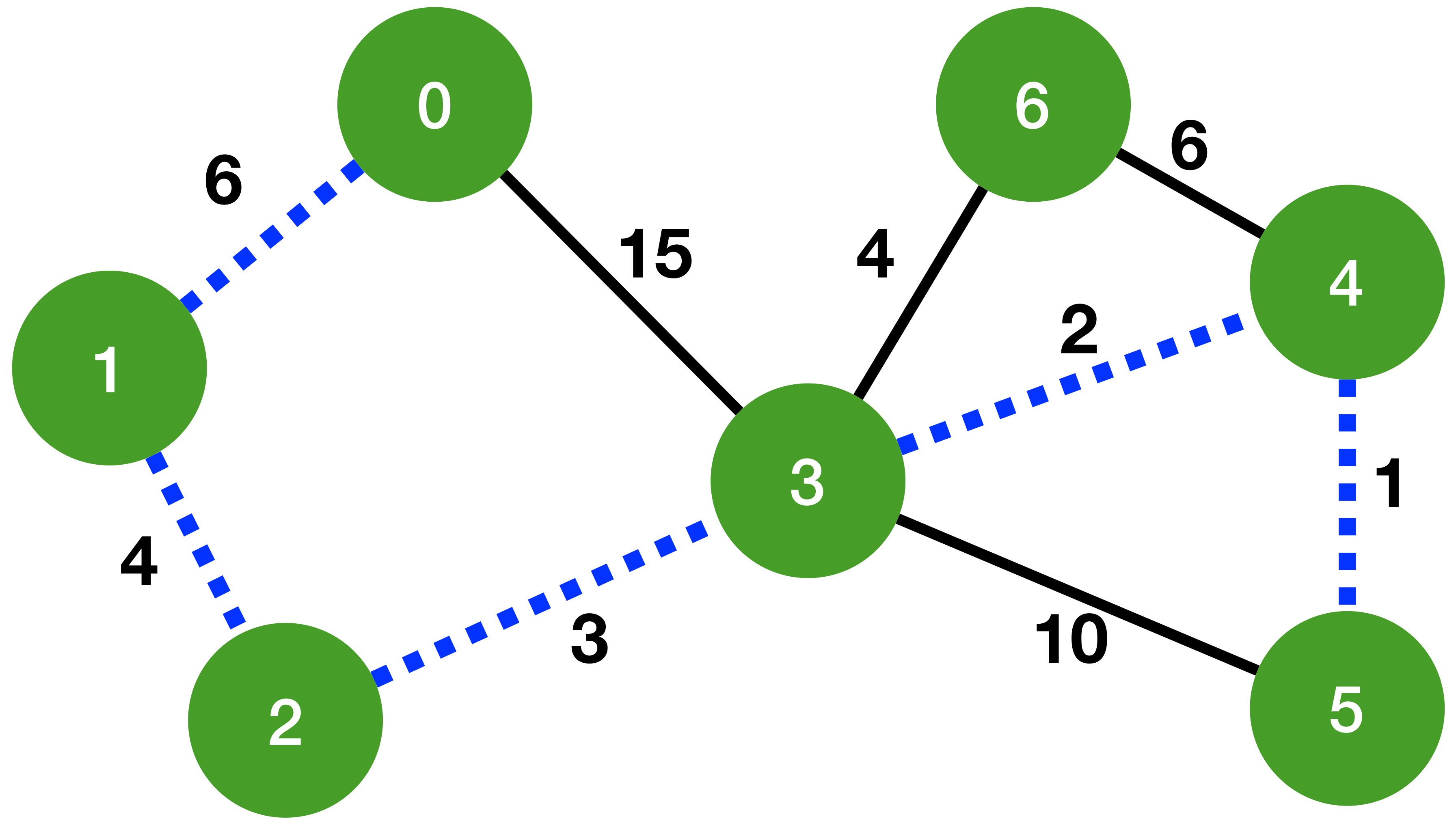
Sum of the weight of edges in the path



Path from 0 to 5
Length = 6 + 4 + 3 + 10 = 23

Shortest - Paths

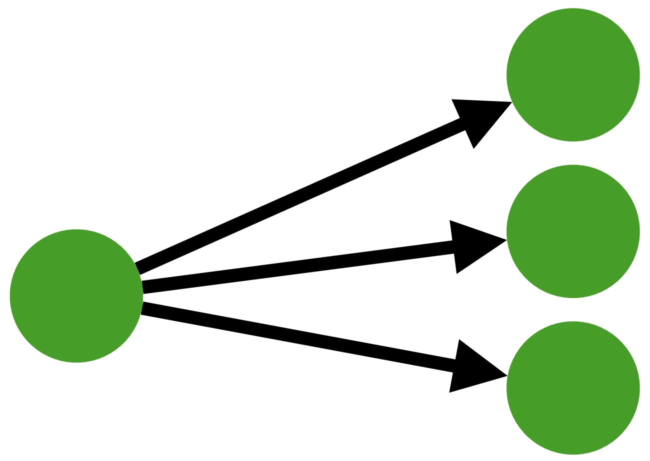
- Path with shortest length
- May not be unique
- Cannot contain cycles



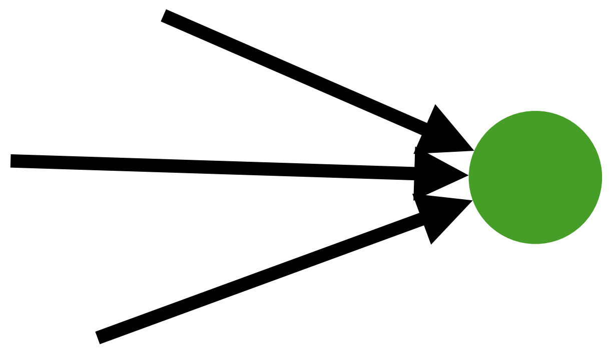
Shortest-Path from 0 to 5
Length = 6 + 4 + 3 + 2 + 1 = 16

Shortest - Paths Problem

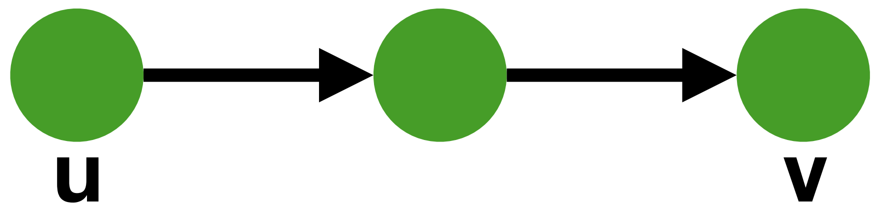
Find shortest-paths



Single - Source



Single - Destination



Single - Pair

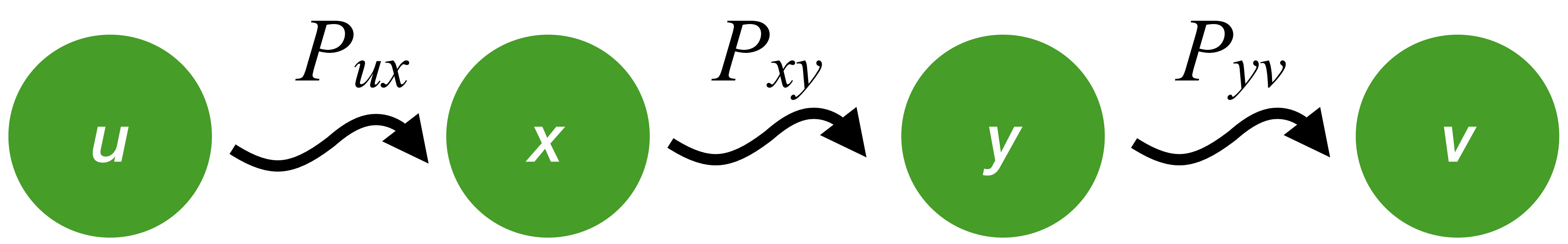
All Pairs - from all source vertices to all destination vertices

Notations

$w(u, v)$	weight of edge (u, v)
$w(P_{xy})$	*sum of weights on path P from x to y*
$\delta(u, v)$	shortest distance from u to v
$u \rightsquigarrow v$	shortest path from u to v

Shortest-Paths Property: Optimal subpath

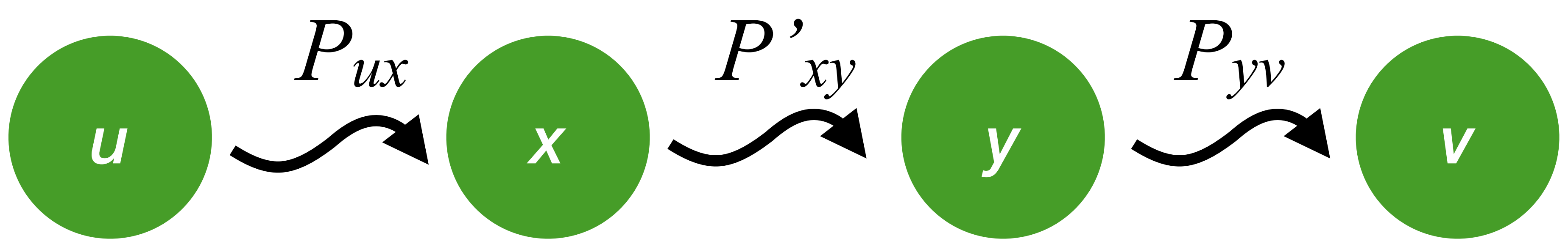
Lemma: Any subpath of a shortest path is a shortest path



Given : P is the shortest path from u to v

$$\delta(u, v) = w(P) = w(P_{ux}) + w(P_{xy}) + w(P_{yv})$$

Proof by contradiction : Assume that P'_{xy} is a shortest path between x and y



Then $w(P') = w(P_{ux}) + w(P'_{xy}) + w(P_{yv}) < w(P)$

=> P cannot be a shortest path and our assumption is incorrect

Shortest-Paths Property: Triangle inequality

If (u, v) is an edge, then $\delta(s, v) \leq \delta(s, u) + w(u, v)$. If p is a path from u to v, then $\delta(s, v) \leq \delta(s, u) + w(p)$.

Proof :

$\delta(s, v)$ = weight of shortest path $s \rightsquigarrow v$ $\leq$ weight of any path from s to v.

Weight of path $p = \delta(s, u) + w(p)$

Shortest-Paths Property: Upper - Bound

$d\,[v]$ (shortest path estimate) upper bound on the weight of a shortest path from source s to v

$d\,[v] \geq \delta\,(s, v)$ for all v

After $d\,[v] = \delta\,(s, v)$, it will never change

Shortest-Paths Property: No - Path

If $\delta(s, v) = \infty$, then $d[v] = \infty$ always

Idea : $d[v]$ cannot be less than $\delta(s, v)$

Proof: $d[v] \geq \delta(s, v) = \infty \Rightarrow d[v] = \infty$

Shortest-Paths Property: Convergence

If $s \rightsquigarrow u \to v$ is a shortest-path, and if $d[u] = \delta(s, u)$. Then after RELAX $u \to v$, $d[v] = \delta(s, v)$.

Proof: Follows from optimal subpath property.

If s to u is shortest path and s to v is a shortest-path, then u to v is a shortest-path

Single - source shortest-path algorithm

1. shortest - path estimate : $d\,[v]$

   $d\,[v] = \infty$ at source

   $d\,[v] \geq \delta(s, v)$ as algorithm progress

2. P $[v]$ = predecessor of v on a shortest path from source s

Relaxing an edge (u, v)

Tighter estimate of shortest-path distance of v from source

```
if (d[v] > (d[u] + w(u, v))) {

    d[v] = d[u] + w (u, v);
    P[v] = u;
}
```

Shortest-Path algorithm

Let $P = (v_0, v_1, v_2 \ldots\ldots\ldots\ldots, v_k)$ be a shortest-path from $s = v_0$ to v_k.
If we relax in order, (v_0, v_1), (v_1, v_2), ….. (v_{k-1}, v_k), even intermixed with other relaxations, then $d\,[v_k] = \delta\,(s,\ v_k)$

Proof by induction : show that $d\,[v_i] = \delta\,(s,\ v_i)$ after (v_{i-1}, v_i) is relaxed

Basic : $i = 0$, $d\,[v_0] = \delta\,(s,\ v_0) = \delta\,(s,\ s) = 0$

Induction Step : Assume that $d\,[v_{i-1}] = \delta\,(s,\ v_{i-1})$

When (v_{i-1}, v_i) is relaxed, $d\,[v_i] = \delta\,(s,\ v_i)$ (by convergence property)

Shortest - Path algorithm

Dijkstra's algorithm

- Weighted directed graph with non-negative weights
- Greedy but gives optimal solution

Bellman-Ford

- Weighted directed graph
- Weights may be negative

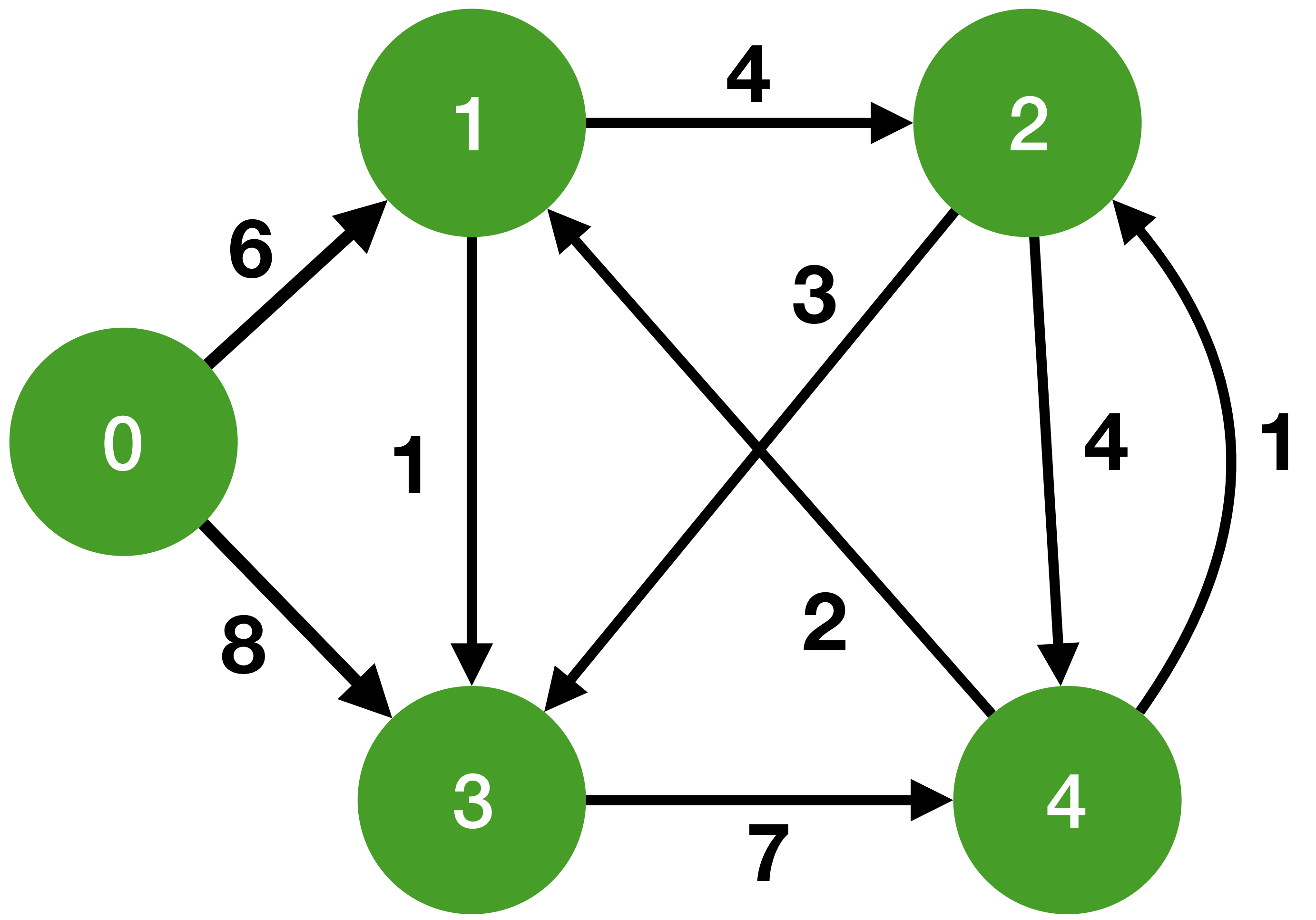
Dijkstra’s algorithm

1. Source vertex u has minimum shortest-path estimate

2. Perform a BFS starting from u and get shortest-path estimates of adjacent vertices

3. Pick new vertex u with minimum shortest-path estimate and repeat step 2

Dijkstra's algorithm example



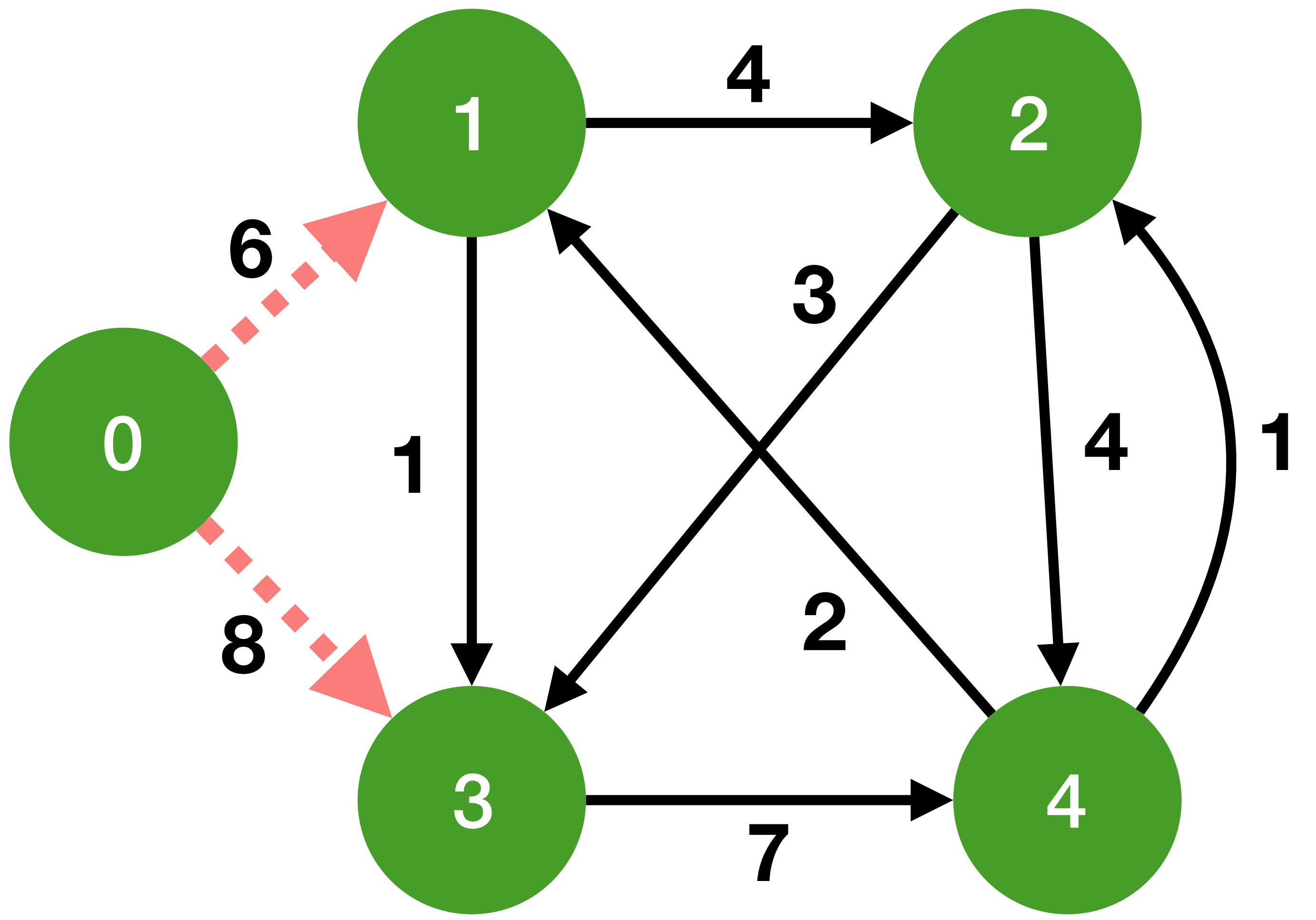
Q = V - SOL

id	**0**	1	2	3	4
d	**0**	∞	∞	∞	∞
p	**-1**	-1	-1	-1	-1

SOL = optimal solution set

id	
d	
p	

Dijkstra's algorithm example



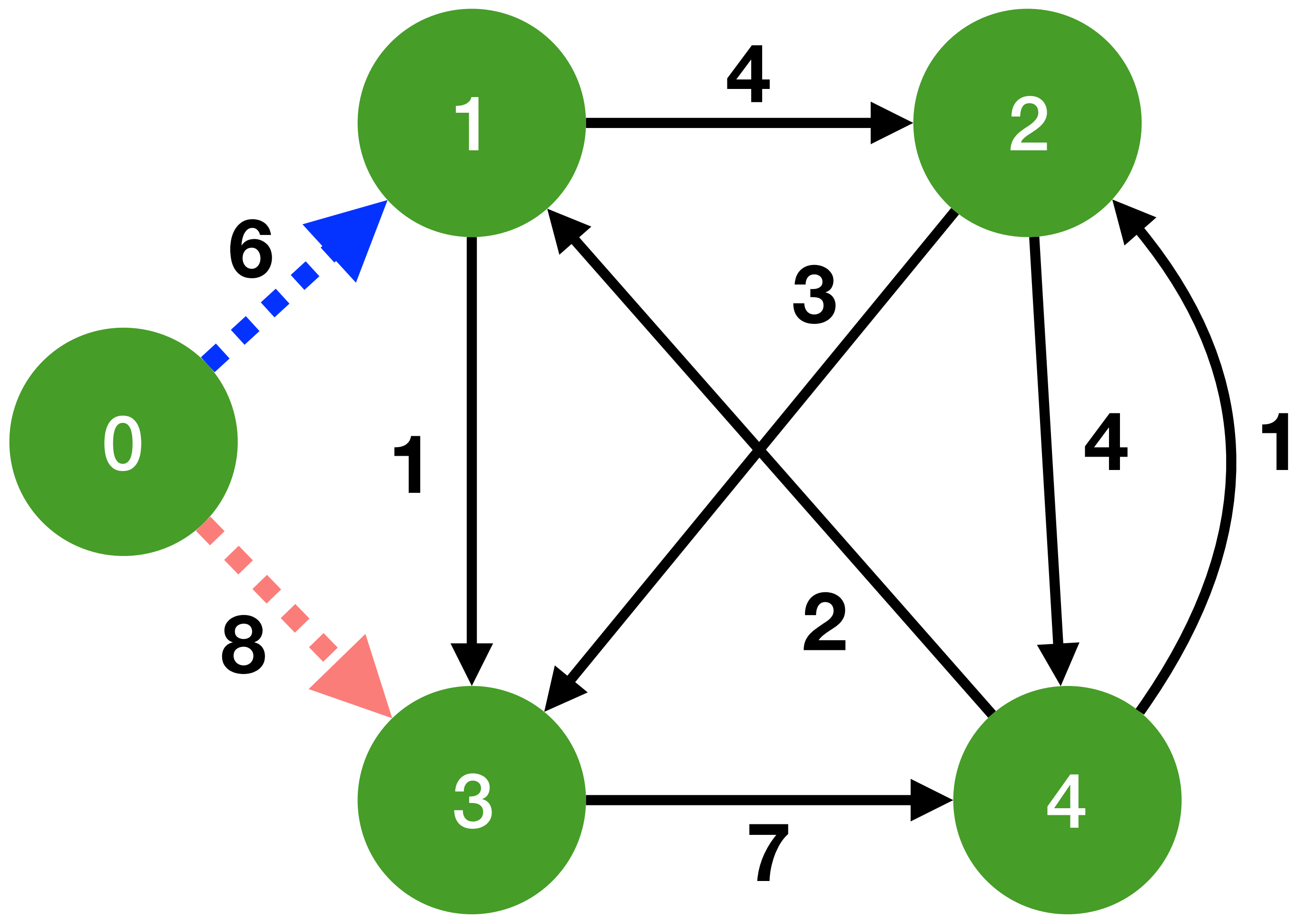
Q = V - SOL

id	1	2	3	4
d	6	∞	8	∞
p	0	-1	0	-1

SOL = optimal solution set

id	0
d	0
p	-1

Dijkstra's algorithm example



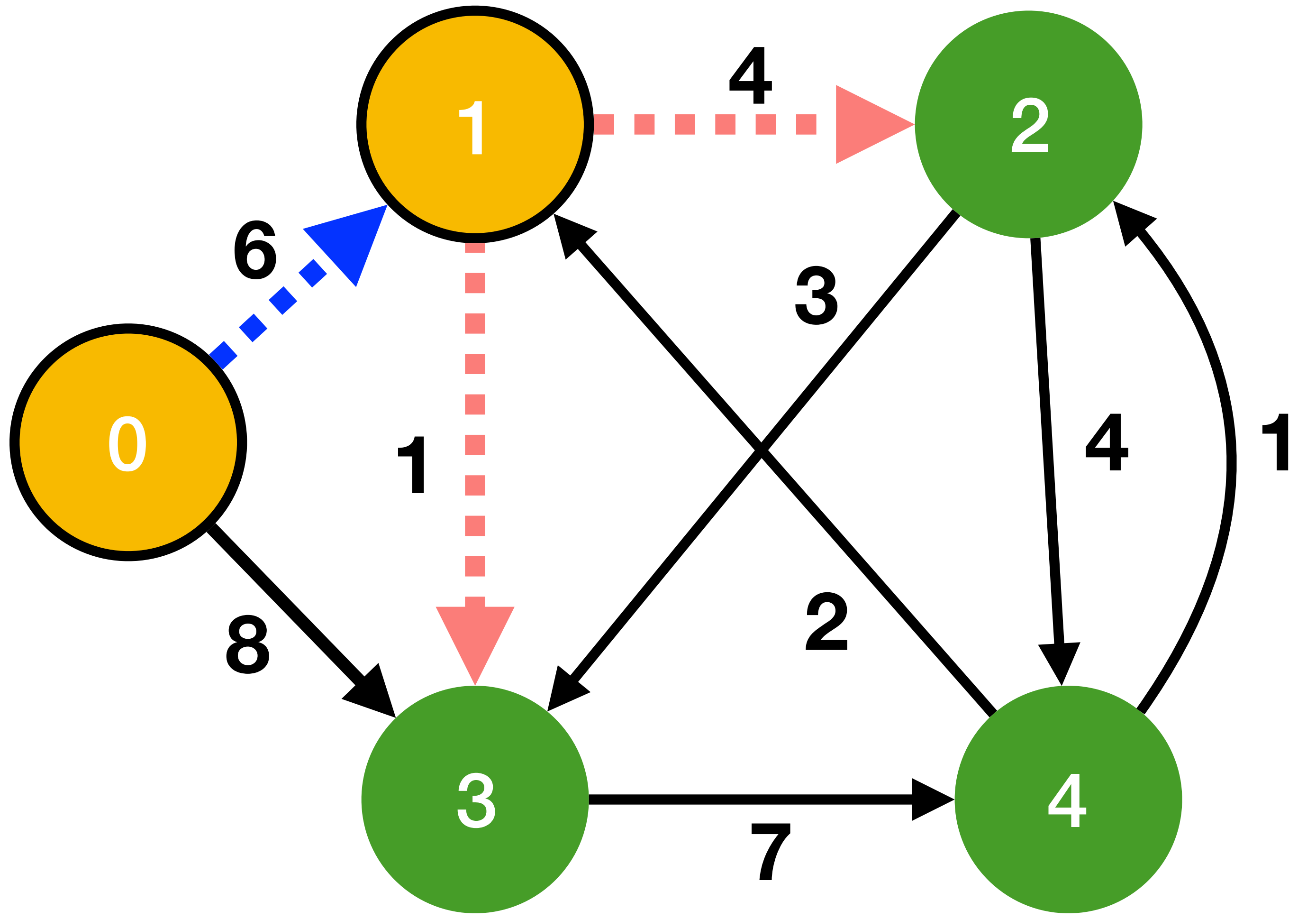
Q = V - SOL

id	1	2	3	4
d	6	∞	8	∞
p	0	-1	0	-1

SOL = optimal solution set

id	0
d	0
p	-1

Dijkstra’s algorithm example



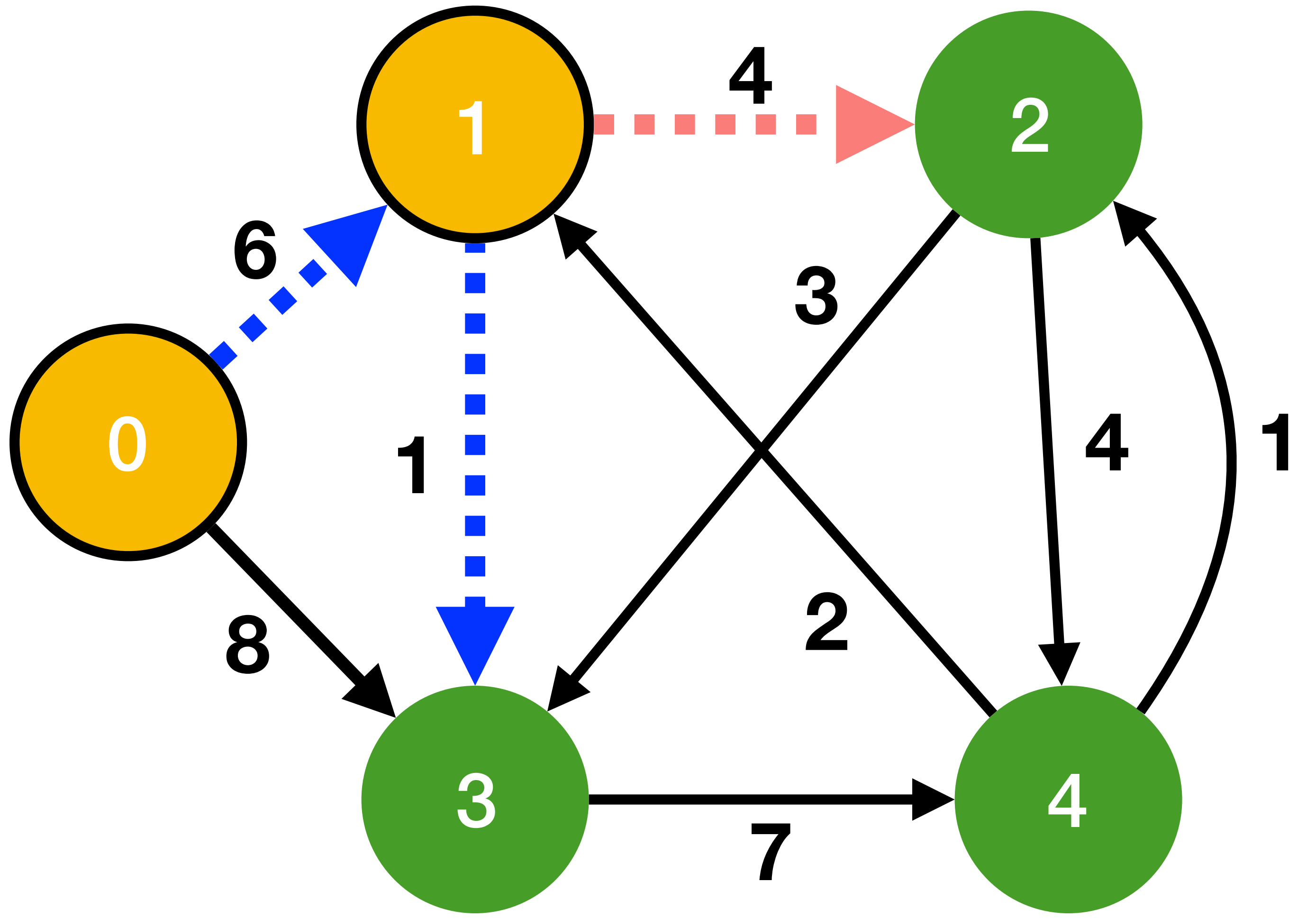
Q = V - SOL

id	2	3	4
d	10	7	∞
p	1	1	-1

SOL = optimal solution set

id	0	1
d	0	6
p	-1	0

Dijkstra’s algorithm example



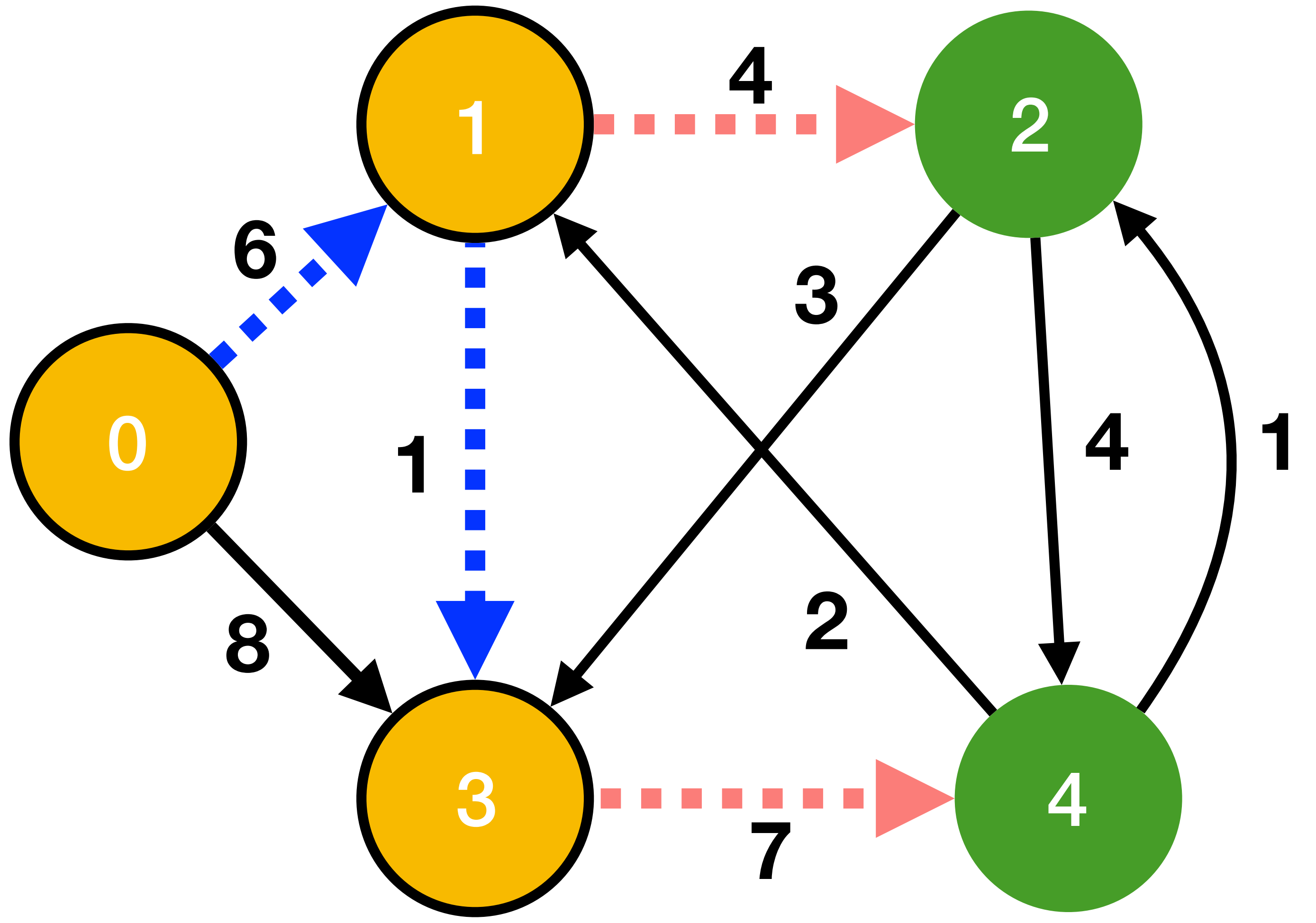
Q = V - SOL

id	2	3	4
d	10	7	∞
p	1	1	-1

SOL = optimal solution set

id	0	1
d	0	6
p	-1	0

Dijkstra's algorithm example



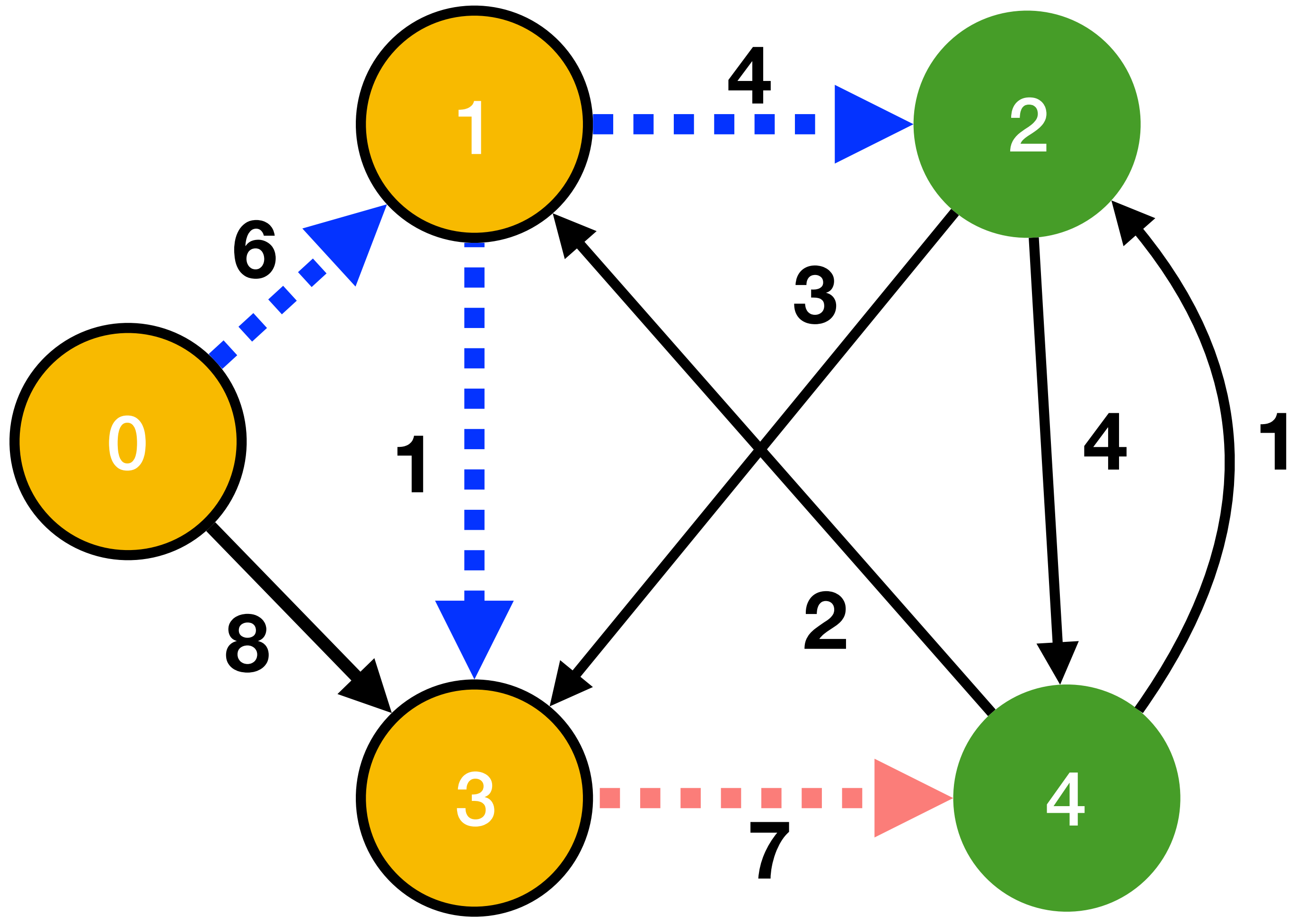
Q = V - SOL

id	2	4
d	10	14
p	1	3

SOL = optimal solution set

id	0	1	3
d	0	6	7
p	-1	0	1

Dijkstra's algorithm example



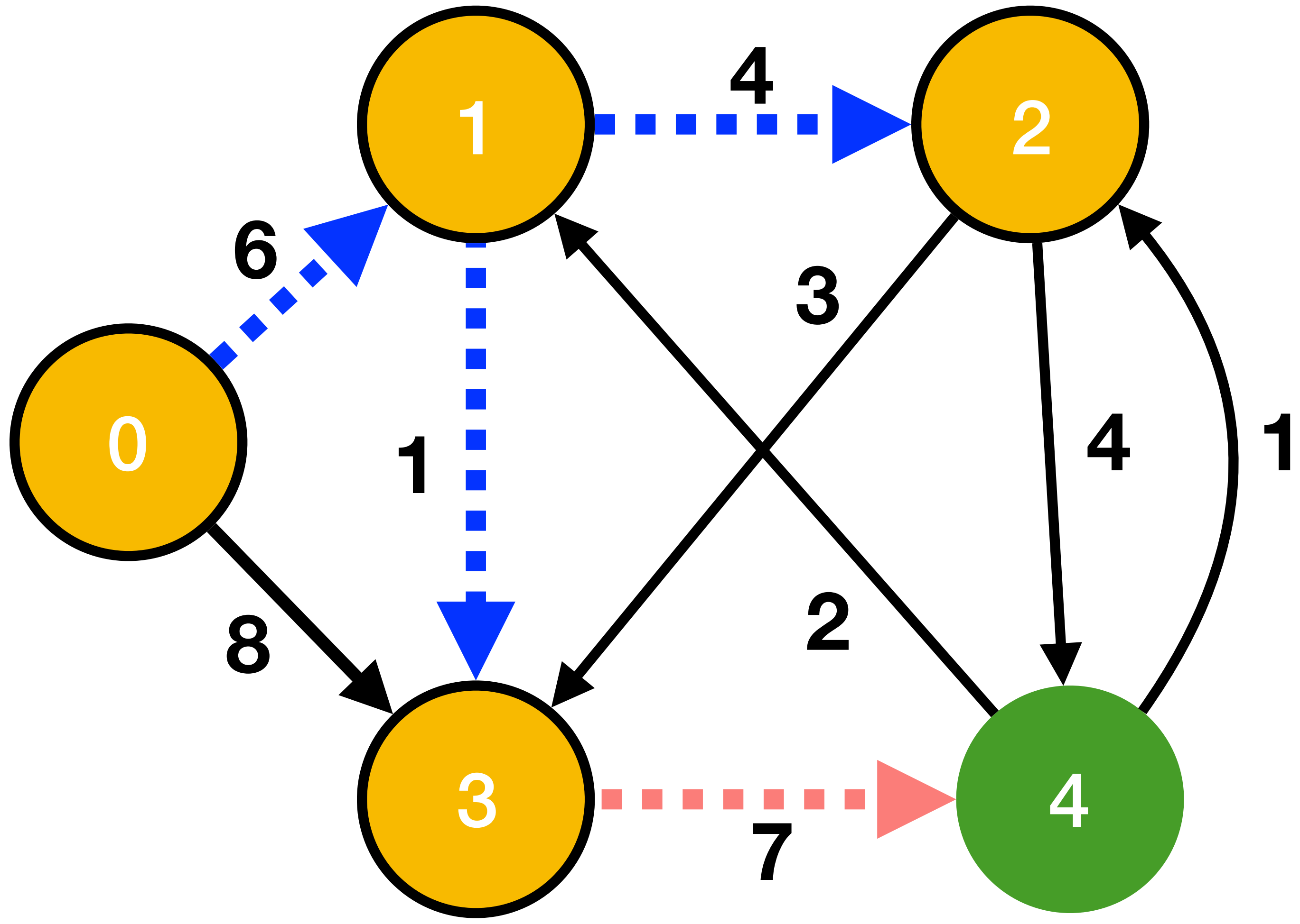
Q = V - SOL

id	2	4
d	10	14
p	1	3

SOL = optimal solution set

id	0	1	3
d	0	6	7
p	-1	0	1

Dijkstra’s algorithm example



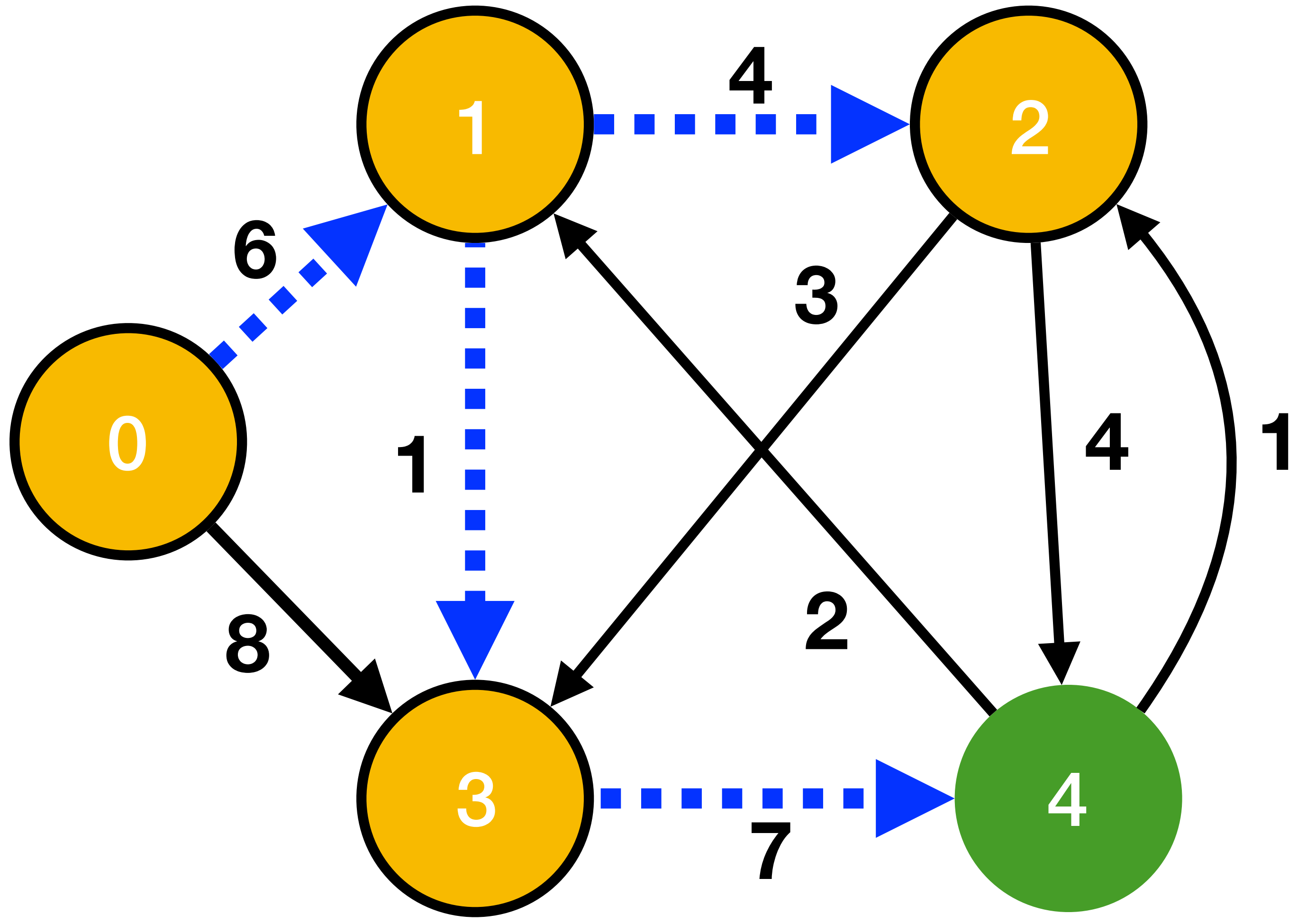
Q = V - SOL

id	4
d	14
p	3

SOL = optimal solution set

id	0	1	3	2
d	0	6	7	10
p	-1	0	1	1

Dijkstra's algorithm example



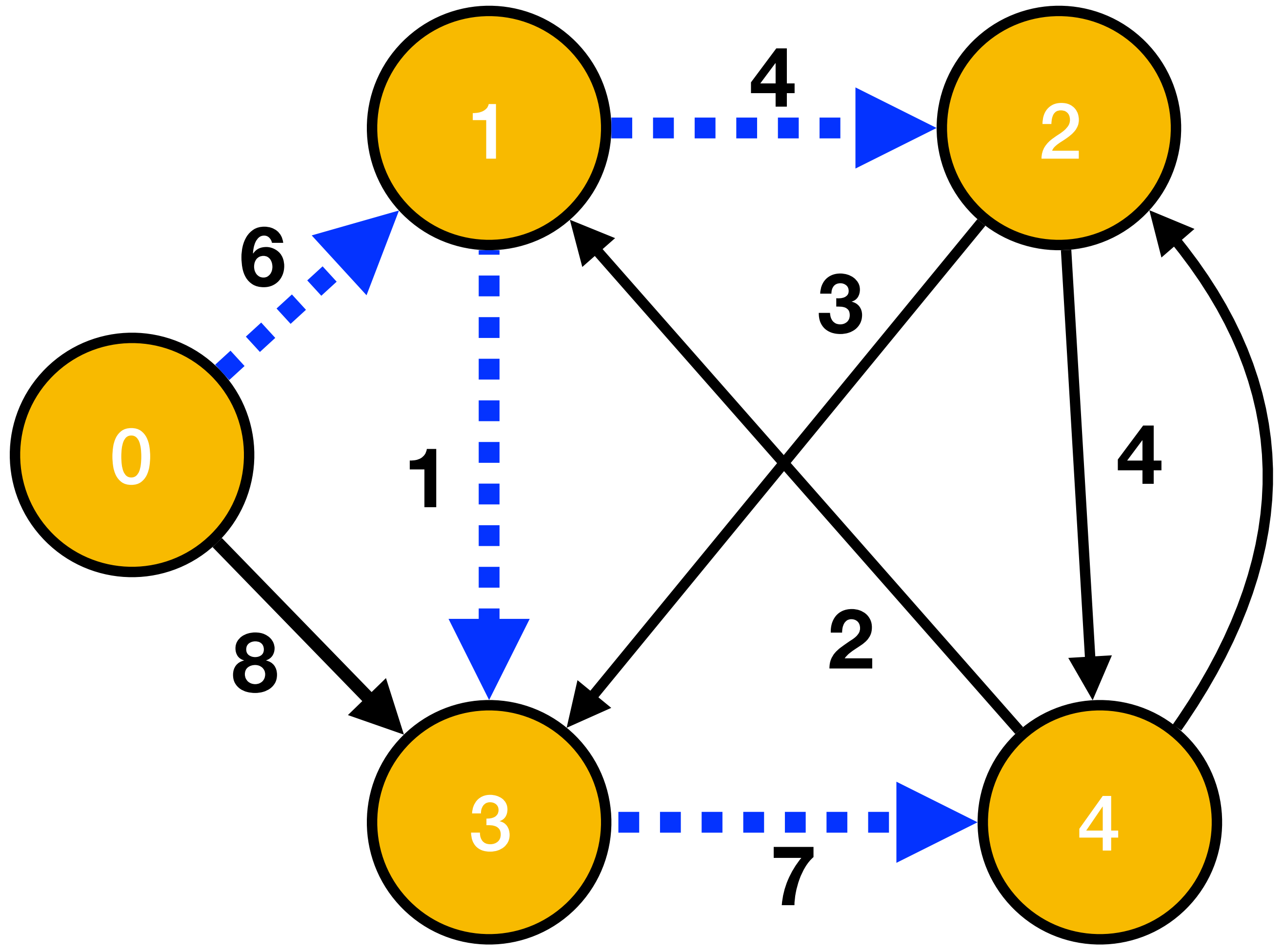
Q = V - SOL

id	4
d	14
p	3

SOL = optimal solution set

id	0	1	3	2
d	0	6	7	10
p	-1	0	1	1

Dijkstra’s algorithm example



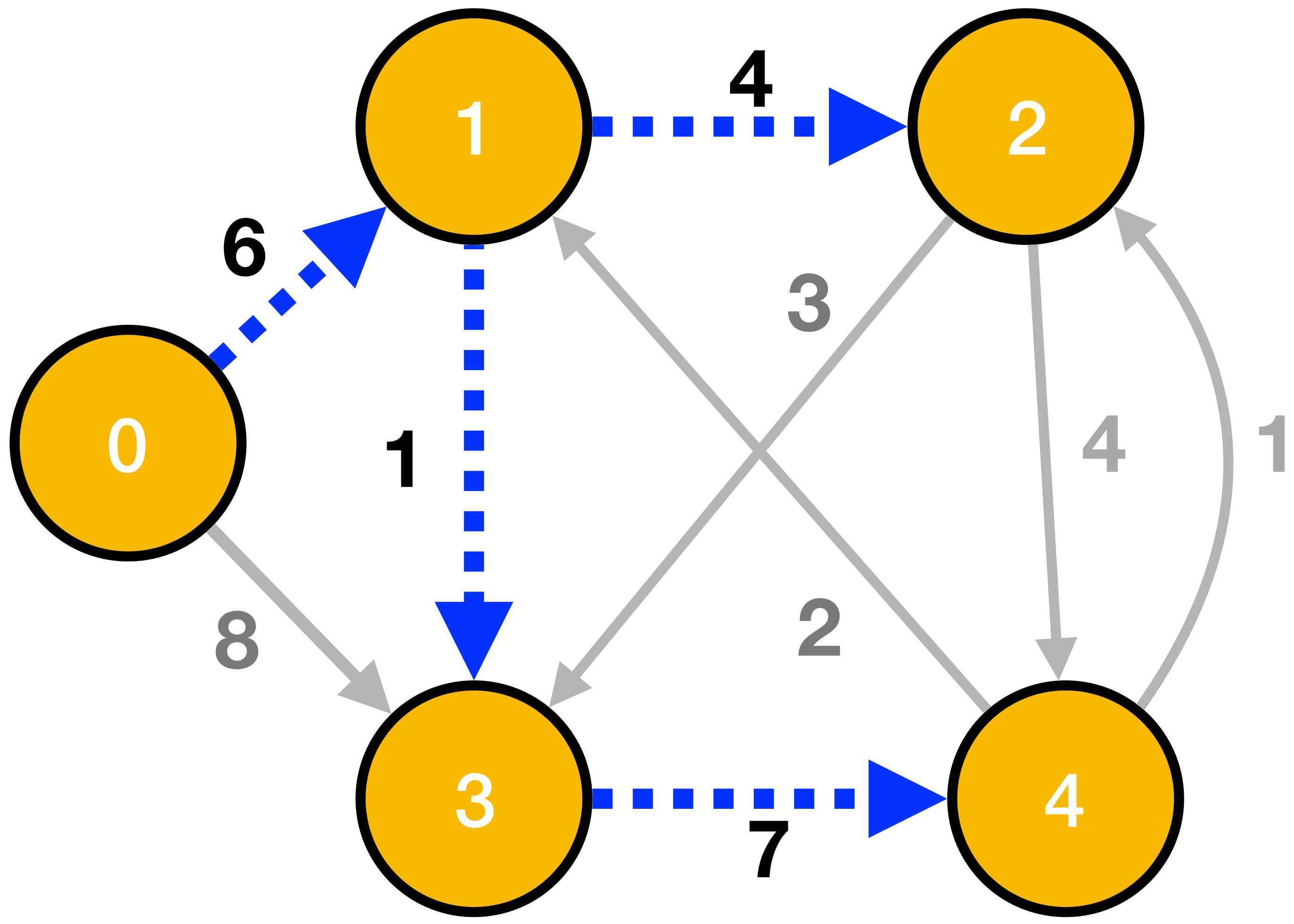
Q = V - SOL

id	
d	
p	

SOL = optimal solution set

id	0	1	3	2	4
d	0	6	7	10	14
p	-1	0	1	1	3

Dijkstra's algorithm solution



SOL = optimal solution set

id	0	1	3	2	4
d	0	6	7	10	14
p	-1	0	1	1	3

Dijkstra’s algorithm pseudocode

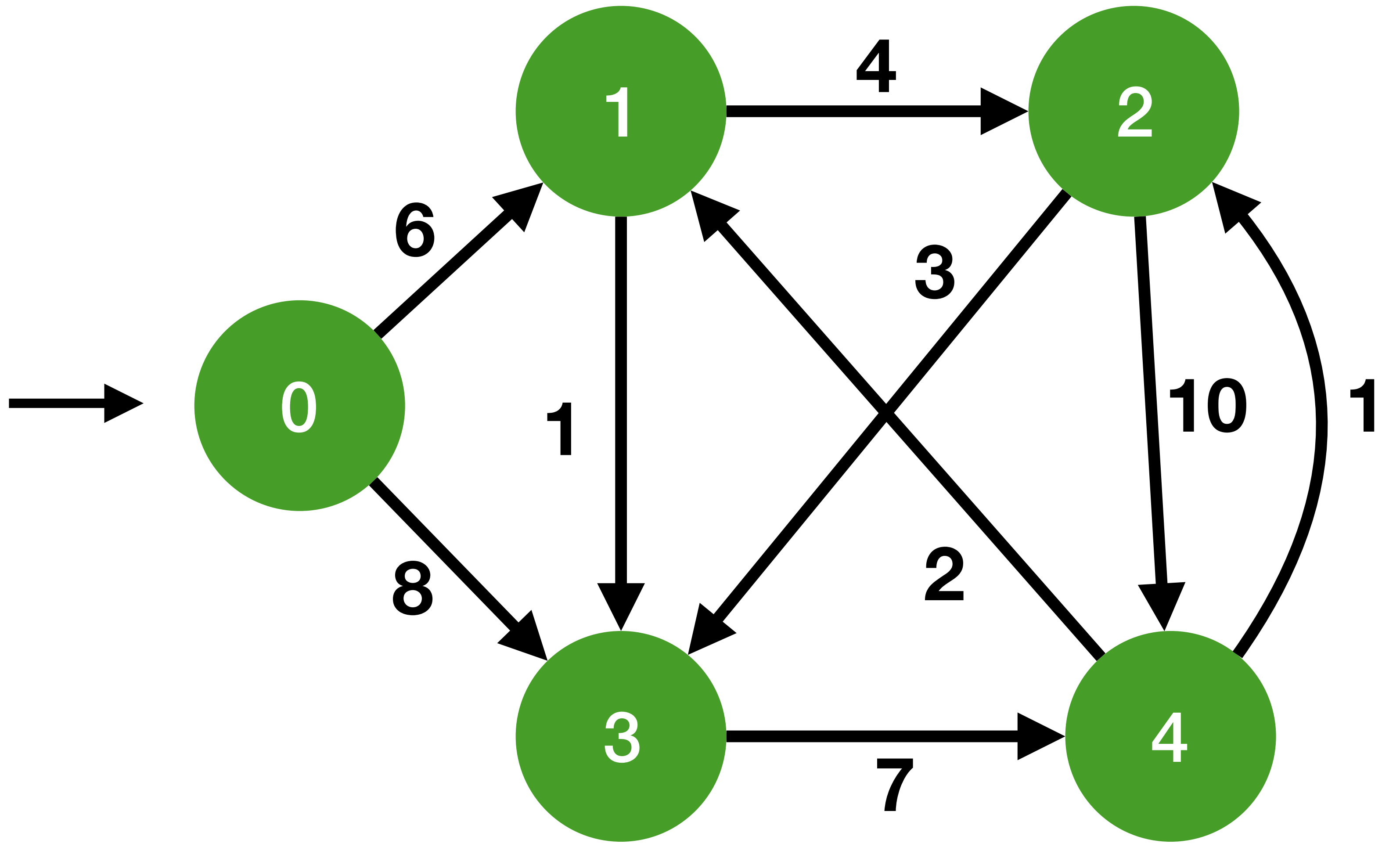
```
Dijkstra (G, w, s){   // G is graph, w is weight, s is source vertex
    for each v ∈ V {  // initialize vertices with d = ∞
       d[v] = ∞ ;
       p[v] = -1;
    }
    d[s] = 0;         //source has distance 0
    Sol = NULL ;      //store the solution set
    Q = v;            //initialize heap Q to all vertices in G
    while Q ≠ NULL {
        u = EXTRACT-MIN (Q); // remove vertex with smallest d from Q
        Sol = Sol U {u} ;    //put u is S
        for each vertex v adjacent to u {
              RELAX (u, v, w);
        }
    }
}
```

RELAX algorithm

```
RELAX (u, v, w){
   // Relaxes edge (u, v) by tightening shortest-
   distance estimate d of vertex v
   if(d[v] > d[u] + w(u, v)){
      d[v] = d[u] + w(u, v);
      p[v] = u;
   }

}
```

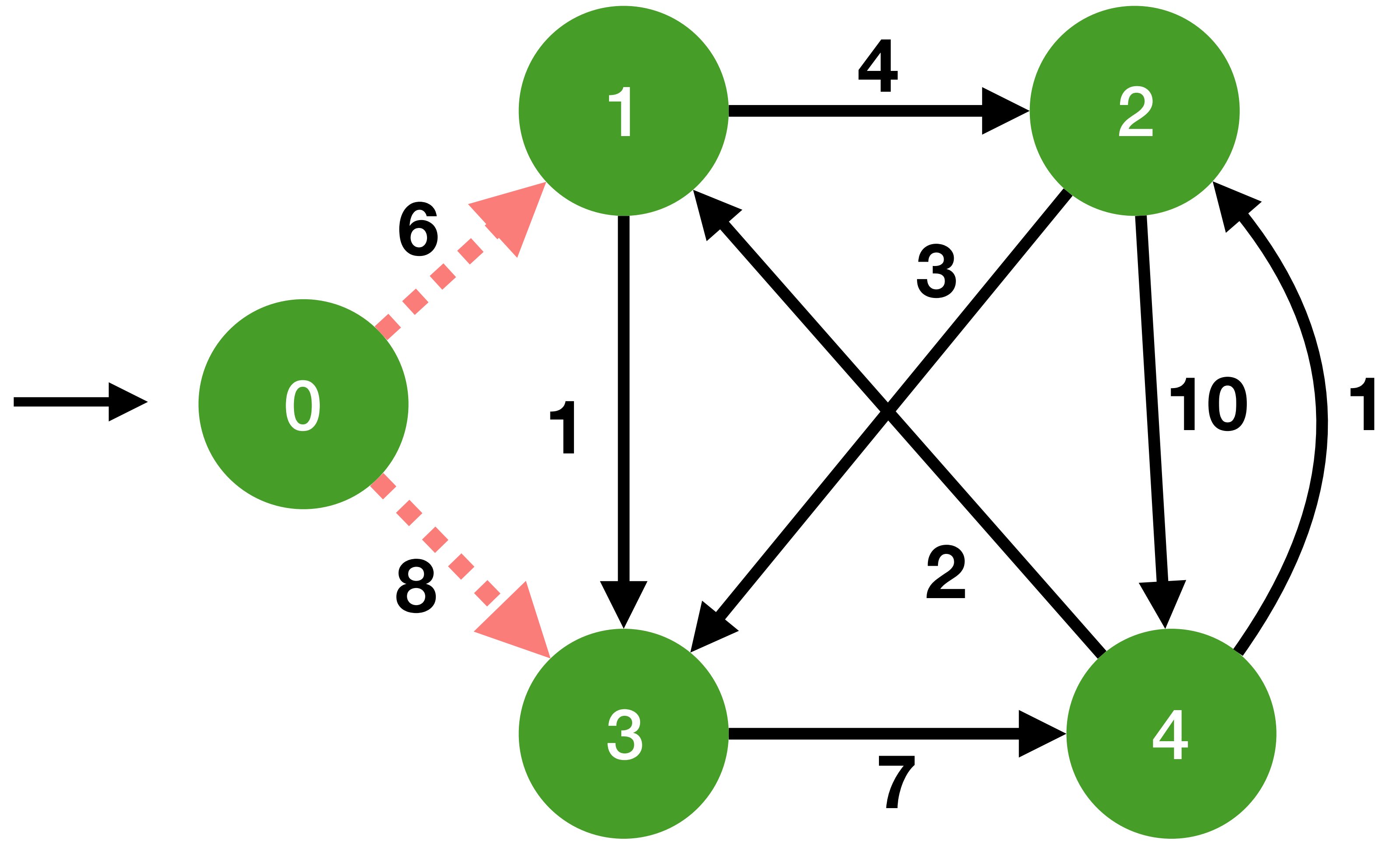
Breadth-first search (non - greedy approach)



V

id	0	1	2	3	4
d	0	∞	∞	∞	∞
p	-1	-1	-1	-1	-1

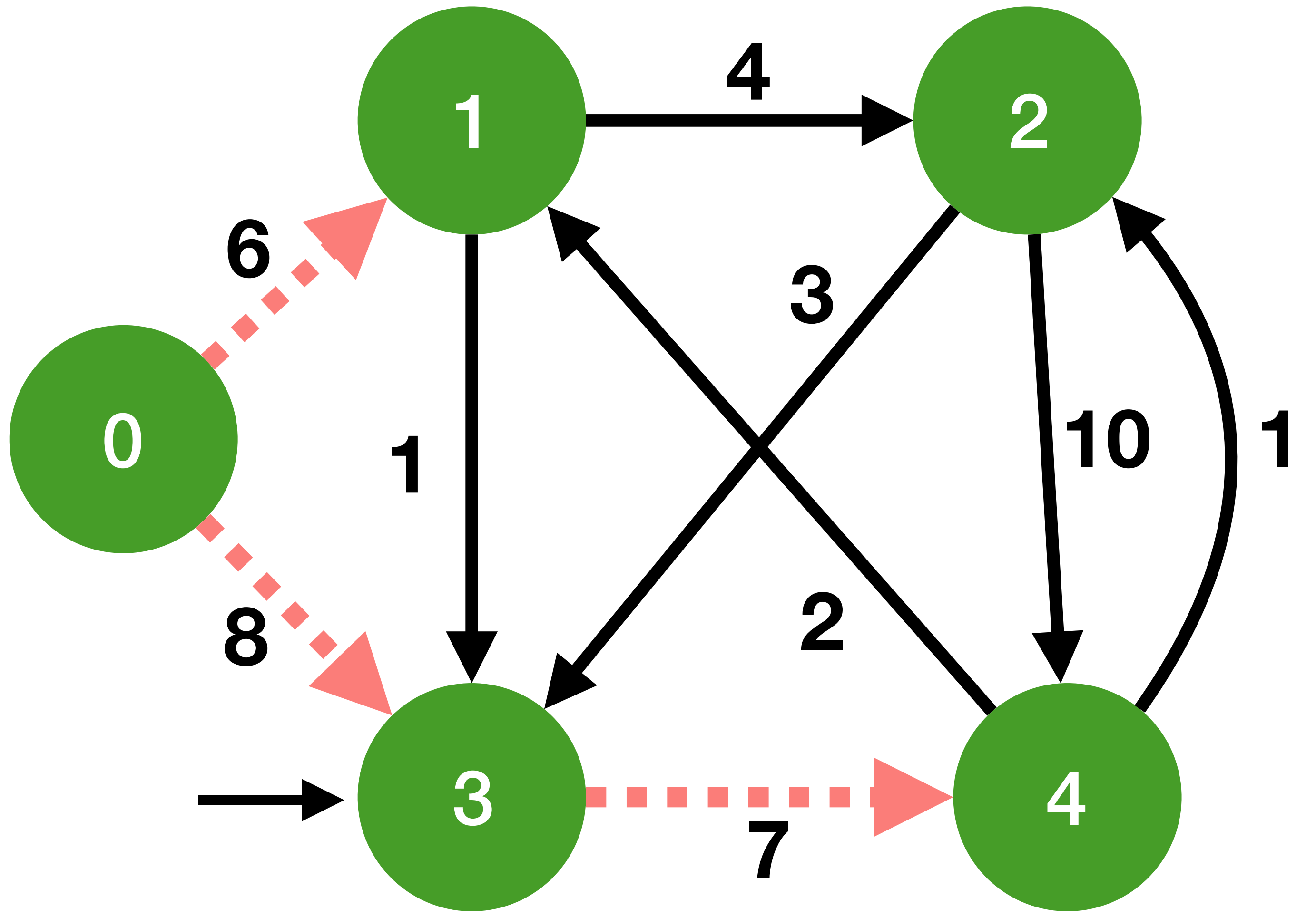
Breadth-first search (non - greedy approach)



V

id	0	1	2	3	4
d	0	6	∞	8	∞
p	-1	0	-1	0	-1

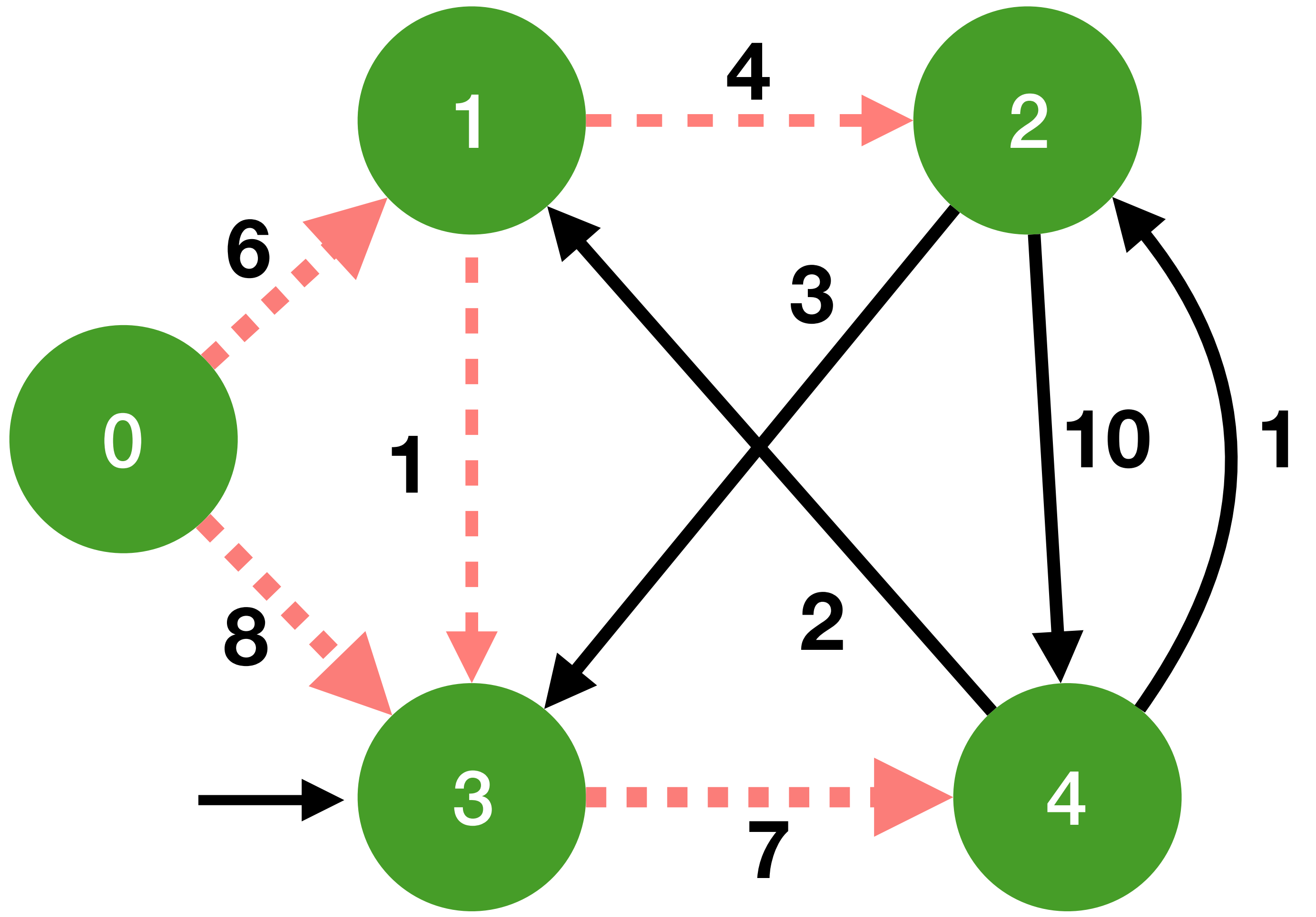
Breadth-first search (non - greedy approach)



V

id	0	1	2	3	**4**
d	0	6	∞	8	**15**
p	-1	0	-1	0	**3**

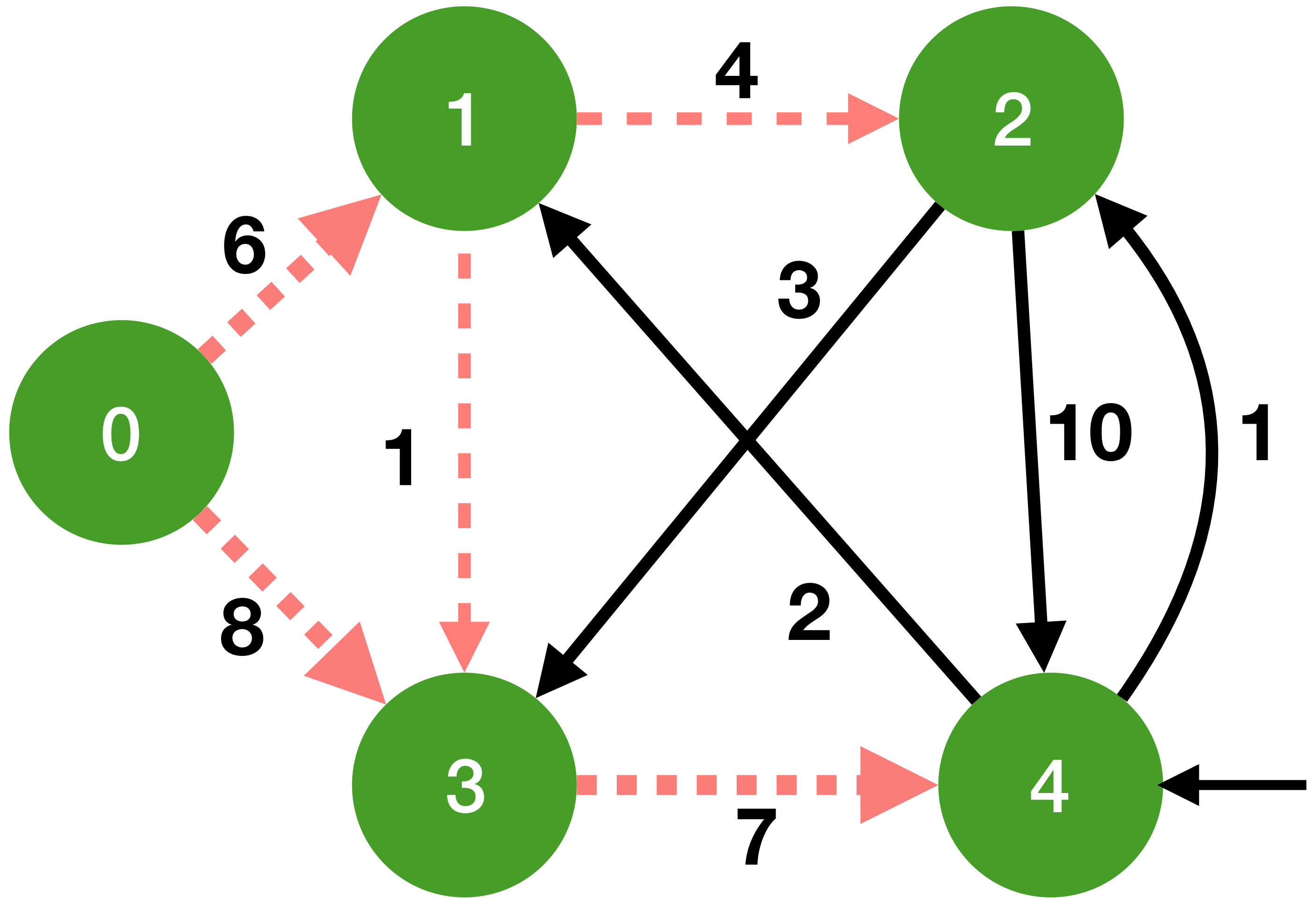
Breadth-first search (non - greedy approach)



V

id	0	1	**2**	**3**	**4**
d	0	6	**10**	**7**	**15**
p	-1	0	**1**	**1**	**3**

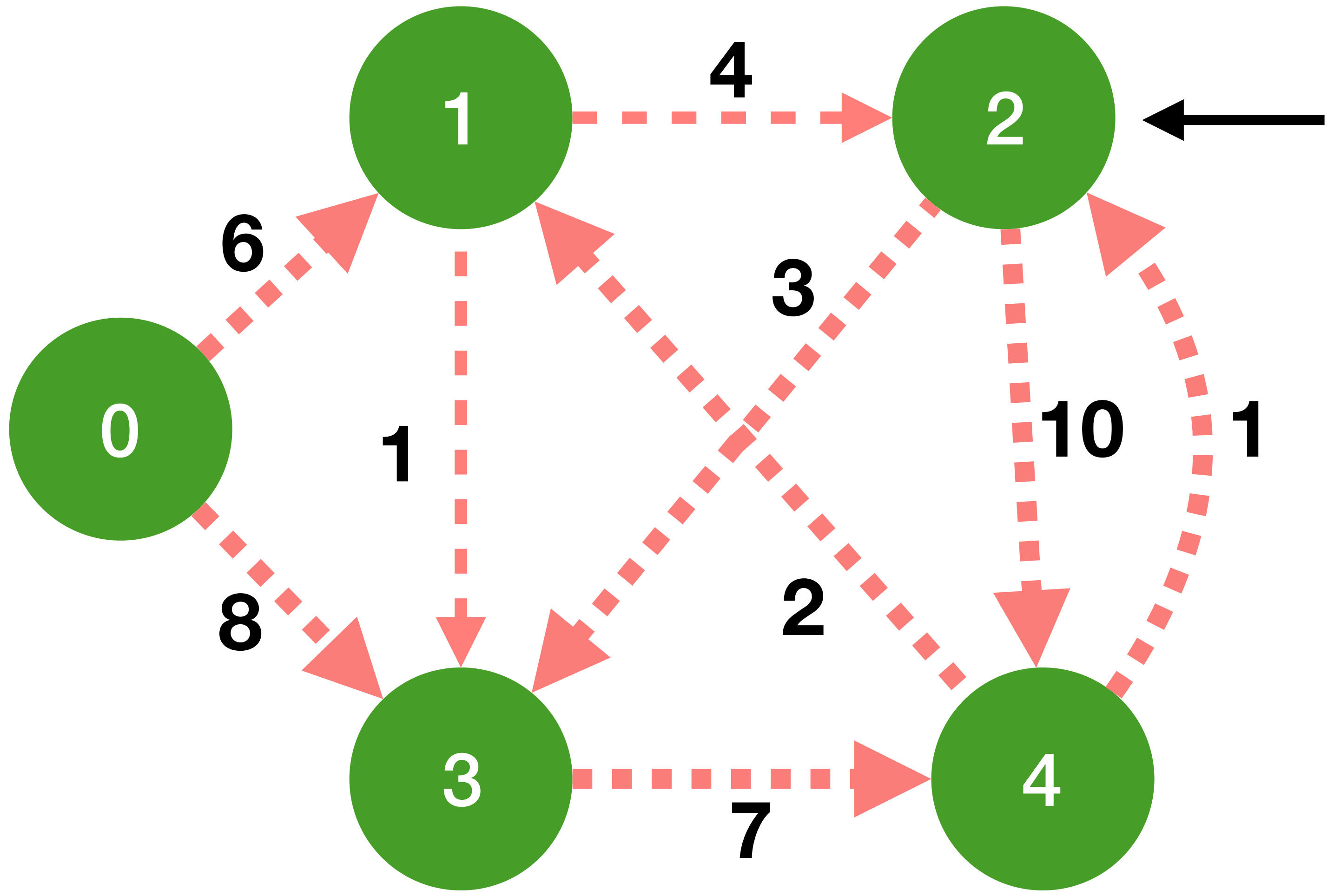
Breadth-first search (non - greedy approach)



V

id	0	1	2	3	4
d	0	6	10	7	15
p	-1	0	1	1	3

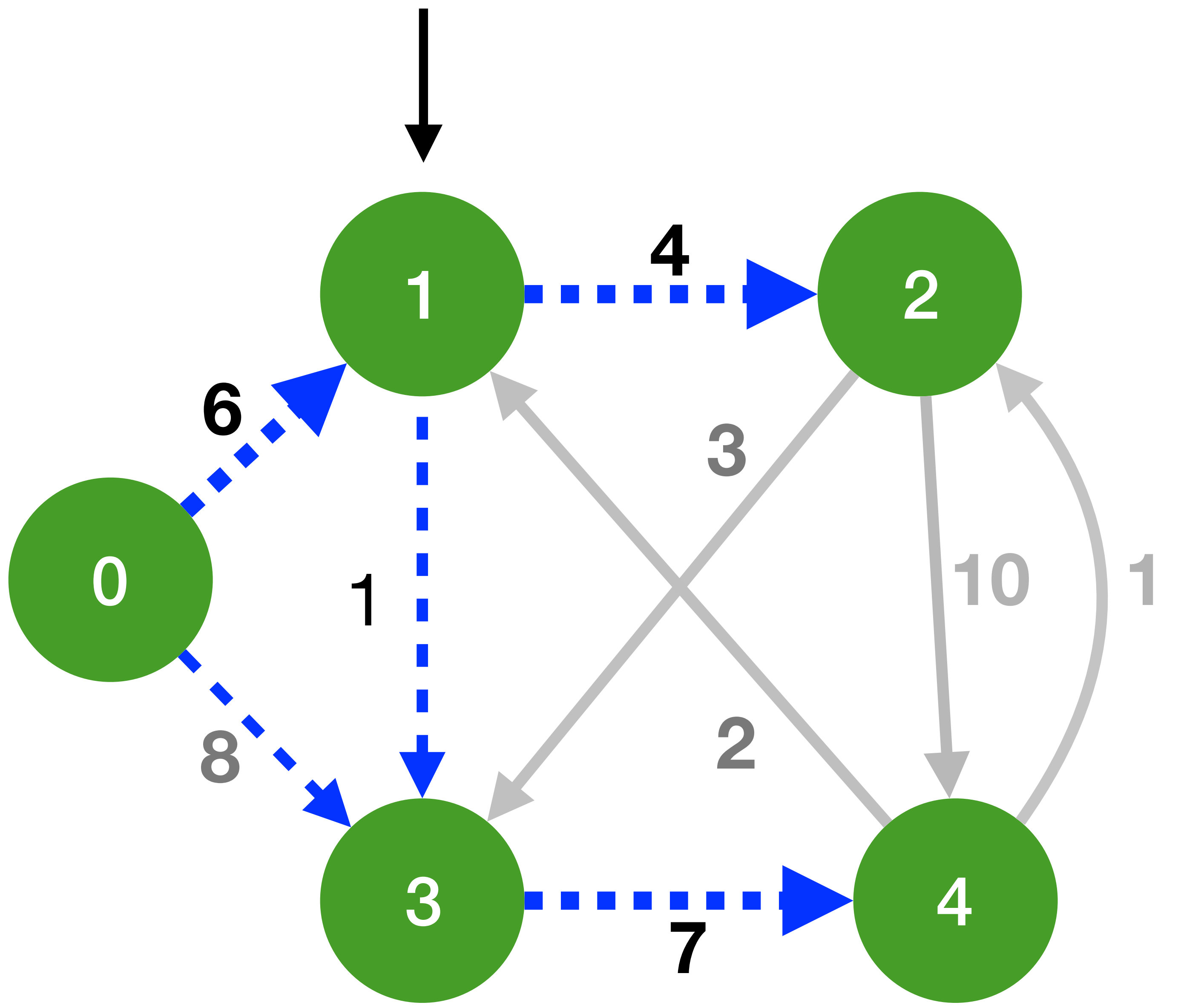
Breadth-first search (non - greedy approach)



V

id	0	1	**2**	**3**	**4**
d	0	6	**10**	**7**	**15**
p	-1	0	**1**	**1**	**3**

Solution with breadth-first search (non - greedy approach)

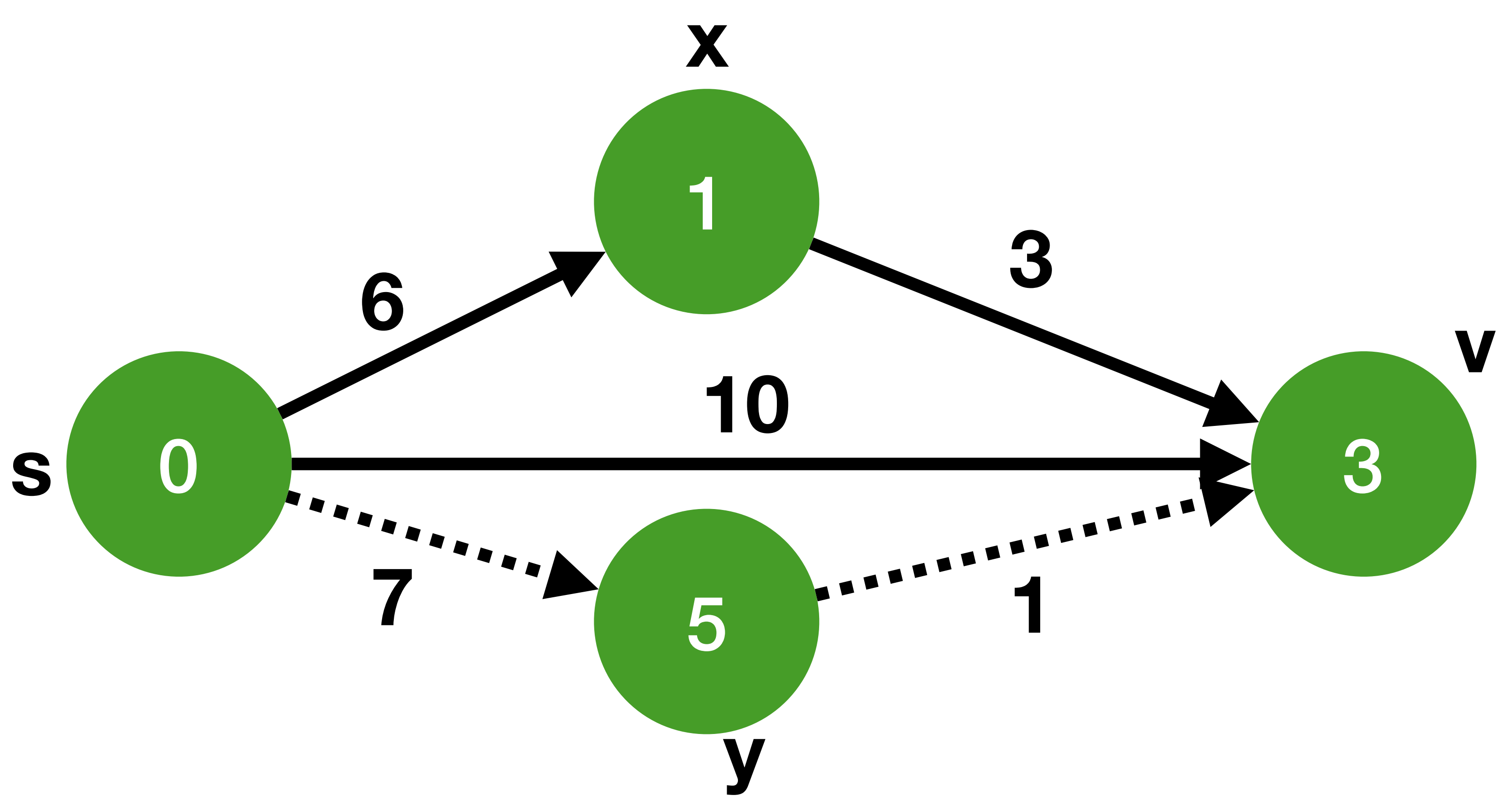


V

id	0	1	2	3	4
d	0	6	10	7	15
p	-1	0	1	1	3

Vertex 4 has incorrect path

Example



Could 0 -> 1 -> 3 be discovered as the shortest-path, when there is another shortest-path?

Step 1

id	0	1	2	3	4	5
d	6	∞	∞	9	∞	7
p	0	-1	-1	1	-1	0

Step 2

id	0	1	2	3	4	5
d		∞	∞		∞	
p						

No, 3 cannot be discovered in Dijkstra's algorithm because 5 must be explored first

Theorem

Given G = (V, E) with non negative weight paths, d[u] = δ (s, u) $\forall$ u $\in$ V

Proof by induction:

Base case: Add the source vertex to solution SOL, then this vertex has the shortest path.

Hypothesis. Assume true for SOL with k vertices, where k >=1.

Let v be the next node added to SOL, and let u-v be the corresponding edge.

The shortest s-u path plus (u,v) is an s-v path of length len(s, v).

Consider any other s-v path P - we need to show that is cannot be shorter than len(s, v)

Suppose that we have another path P: s->x ->y ->v

len(P) >= len(s, x)+ len(x, y) >= δ (s, x) + len(x, y) (by inductive hypothesis)

>= δ (s, y) >= δ (s, v) (Dijkstra chooses v instead of y)

Analysis (Dijkstra’s) - Array

1. Extract vertex v with minimum d from array O(V) - repeat V times

2. For each edge decrease weight d - O(1) repeat at most E times

Total time complexity = $O(E + V^2)$

Analysis (Dijkstra's) - Using binary heap

1. Extract vertex V with minimum d from heap O(log V) - repeat E times

Total time complexity = O(E log V)

Dijkstra's algorithm

- Requires all nodes to be inserted initially into priority queue Q
- RELAX operation performs a decrease key operation
    - Heaps don't efficiently support the find operation for arbitrary nodes

Improved Dijkstra's algorithm

- Only keep starting vertex initially in priority queue Q
- Insert a vertex into the priority queue every time it is updated by RELAX
  - Increases size of priority queue to E, but easier to implement
- Similar to Uniform Cost Search algorithm

Improved Dijkstra's algorithm

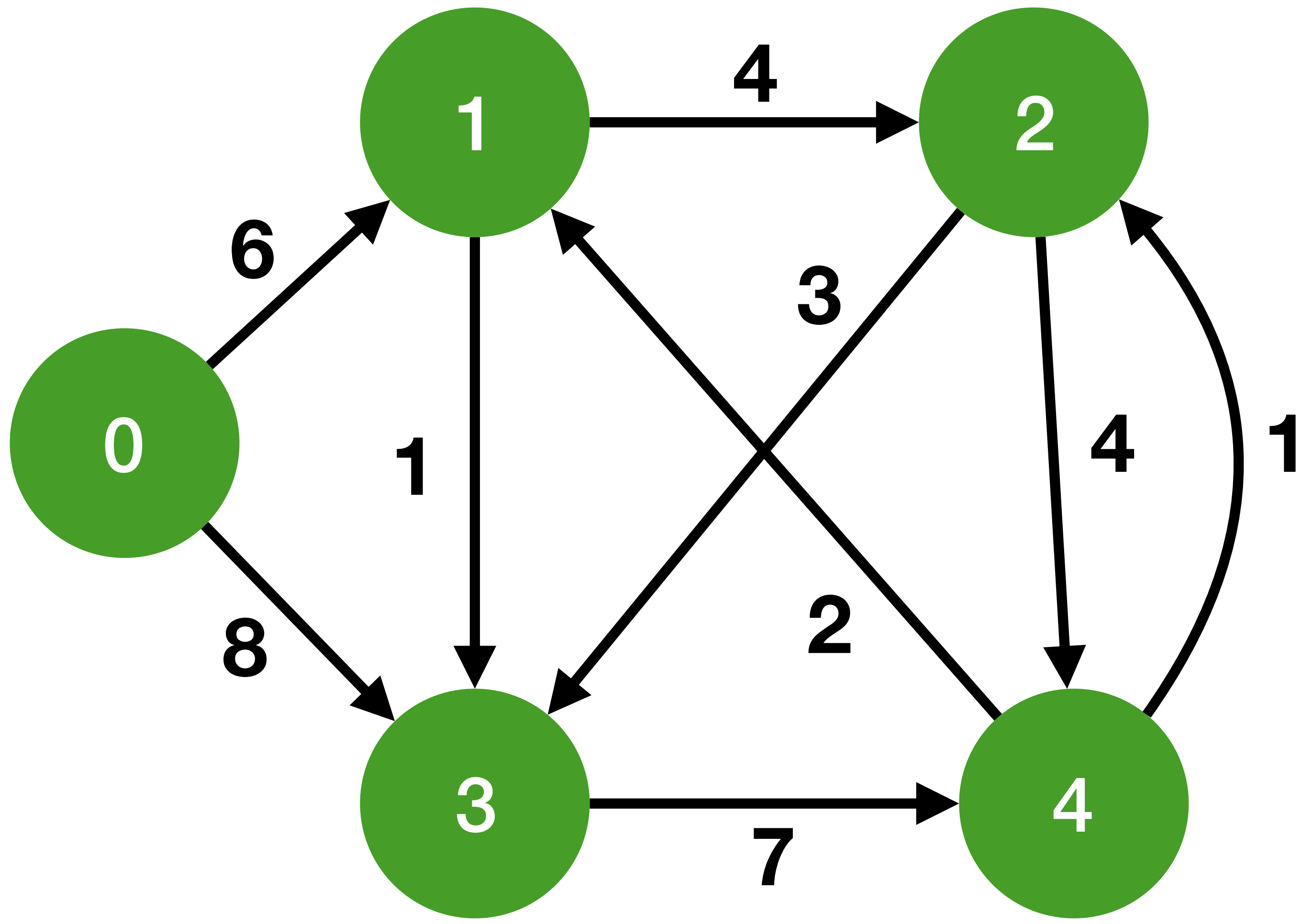
Arrays

known[i]: true if shortest distance to vertex i is known

path[i]: predecessor of vertex i on shortest path

d[i]: shortest-distance estimate of vertex i

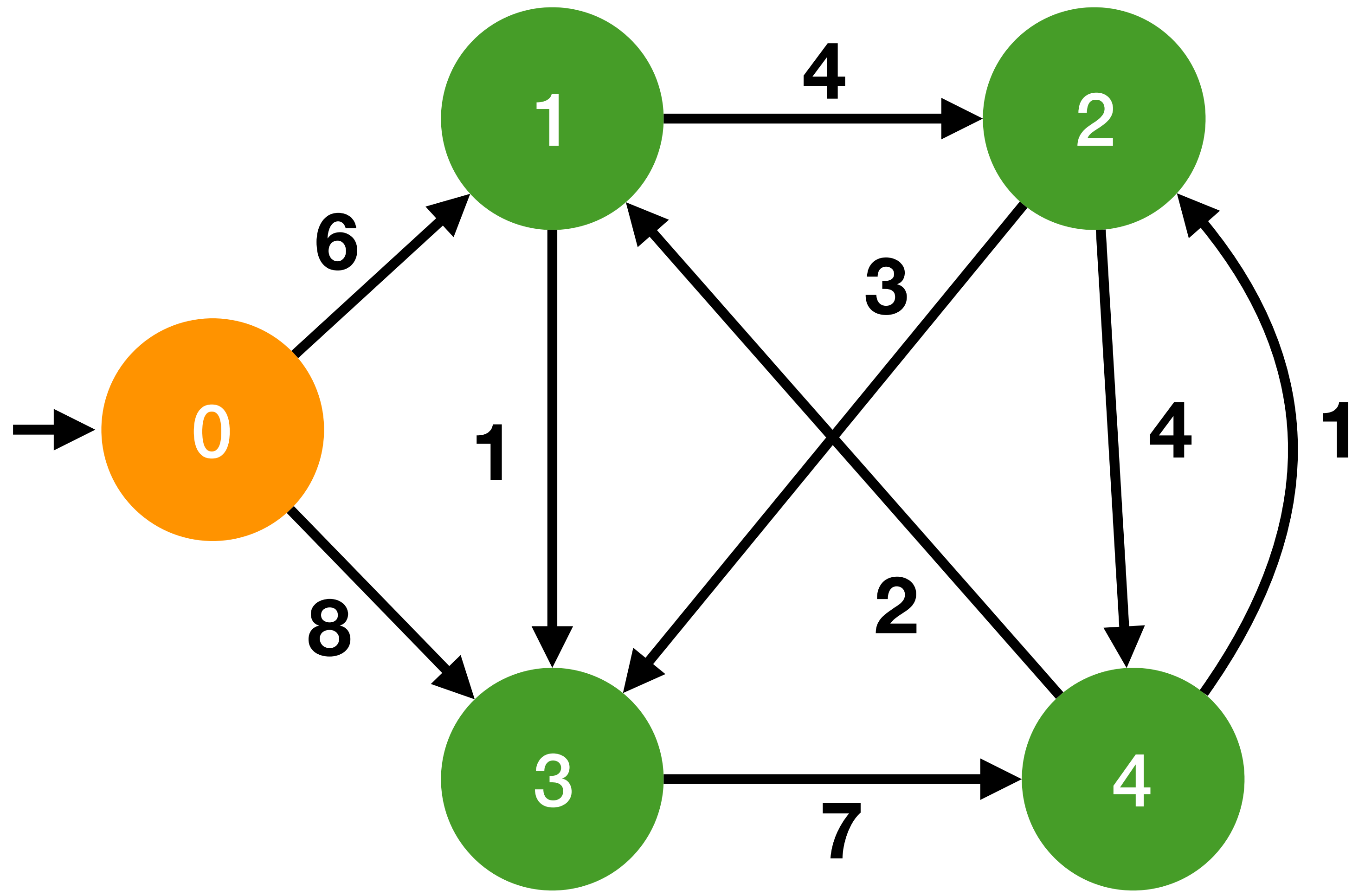
Improved Dijkstra's algorithm



	Q
id	
d	

	d	Path	Known
0	∞	-1	F
1	∞	-1	F
2	∞	-1	F
3	∞	-1	F
4	∞	-1	F

Improved Dijkstra's algorithm

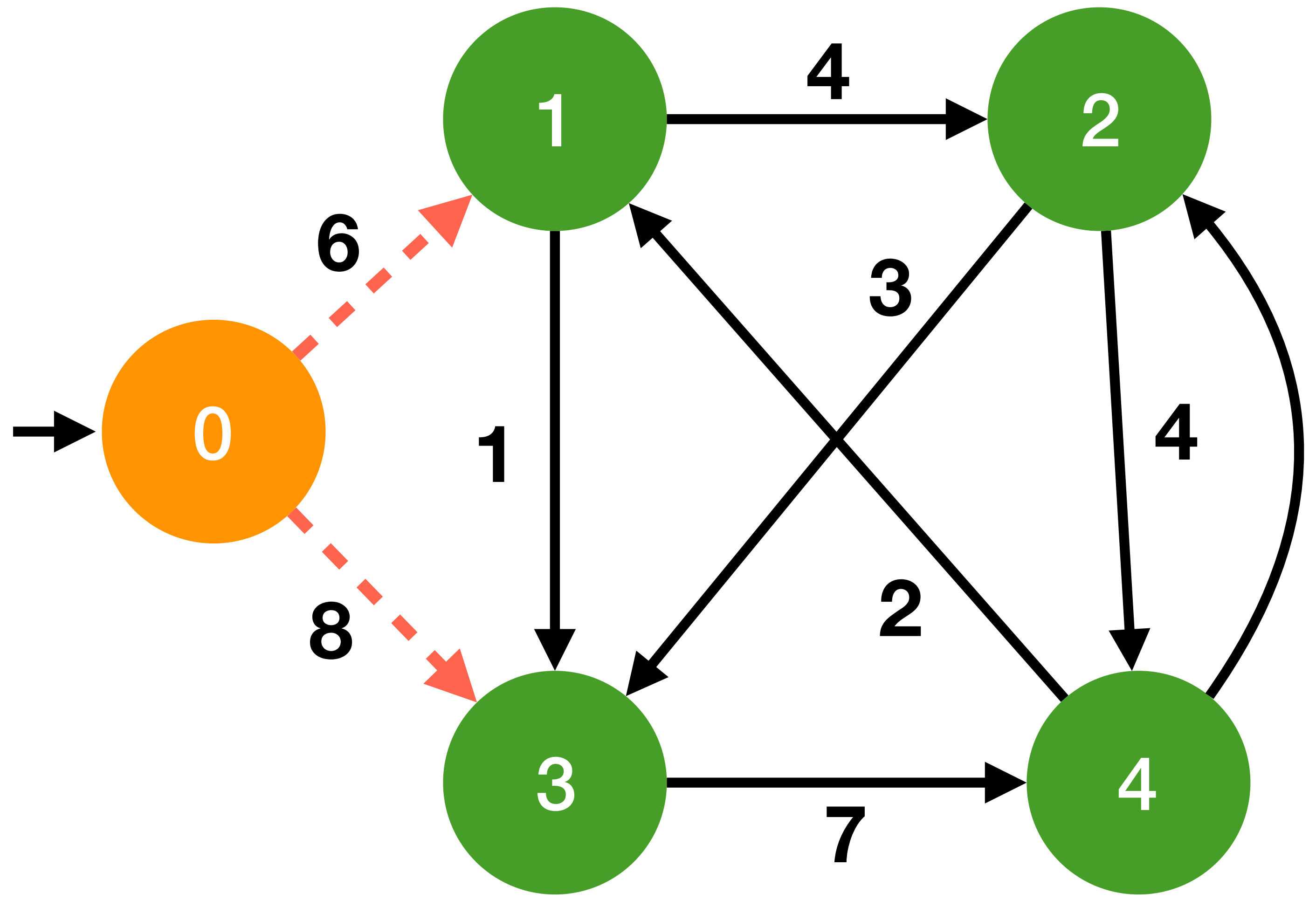


Q	
id	0
d	0

	d	Path	Known
0	**0**	**-1**	**T**
1	∞	-1	F
2	∞	-1	F
3	∞	-1	F
4	∞	-1	F

Remove vertex with smallest d from Q(0)

Improved Dijkstra's algorithm



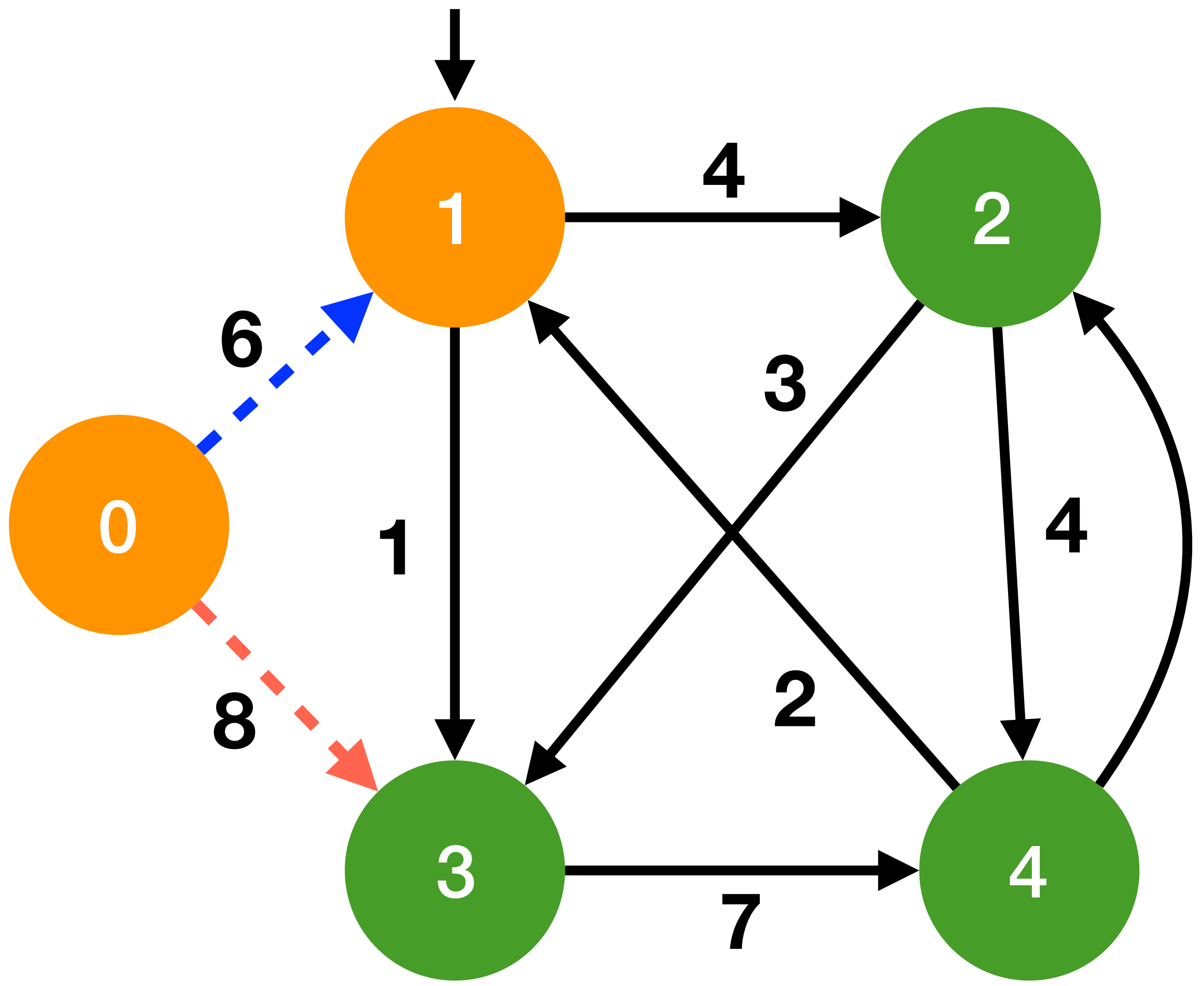
Q

id	1	3
d	6	8

	d	Path	Known
0	0	-1	T
1	**6**	**0**	**F**
2	∞	-1	F
3	**8**	**0**	**F**
4	∞	-1	F

Update each vertex v adjacent to 0 if known [v] is false

Improved Dijkstra's algorithm



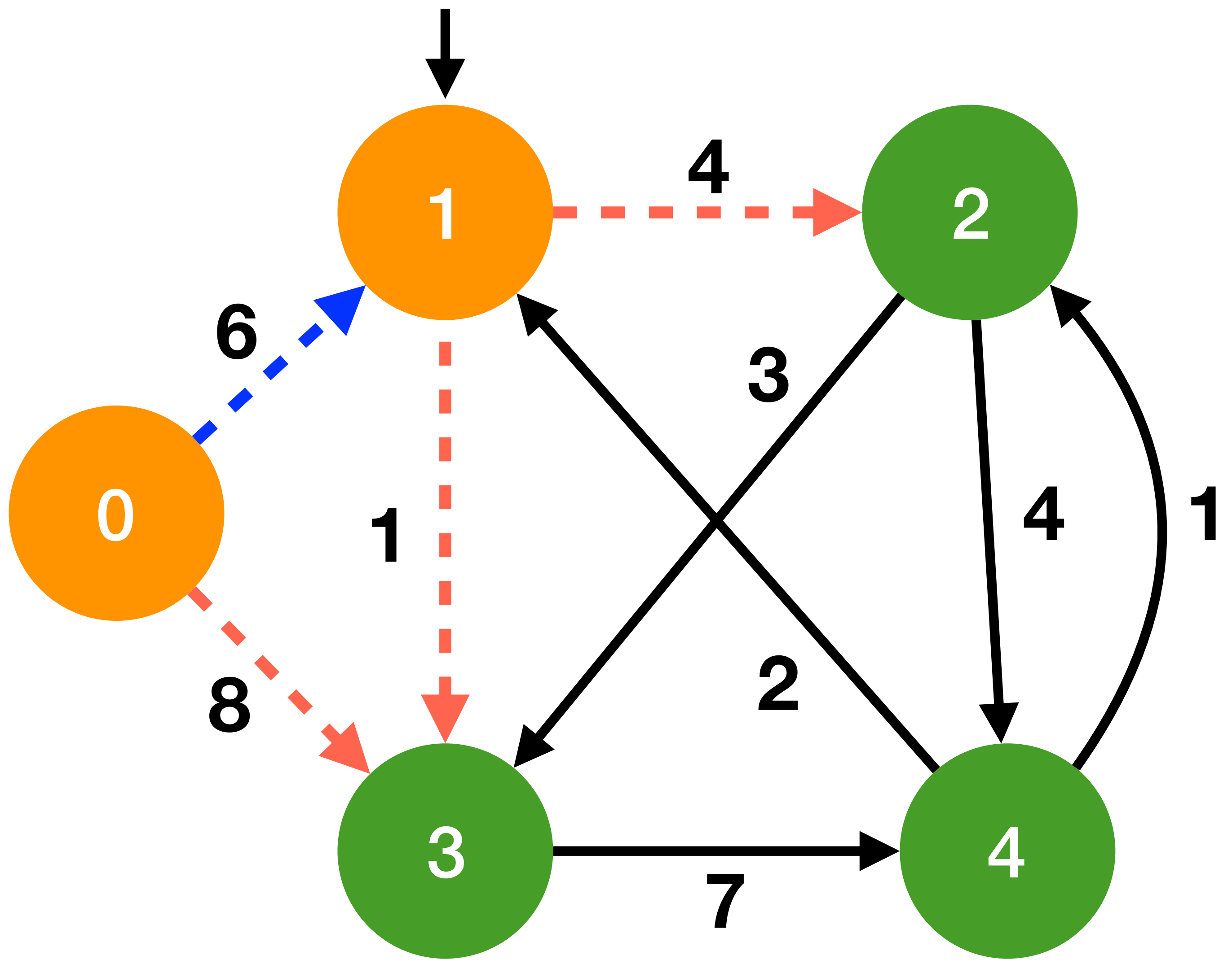
Q

id	3
d	8

	d	Path	Known
0	0	-1	T
1	6	0	T
2	∞	-1	F
3	8	0	F
4	∞	-1	F

Remove vertex with smallest d from Q

Improved Dijkstra's algorithm

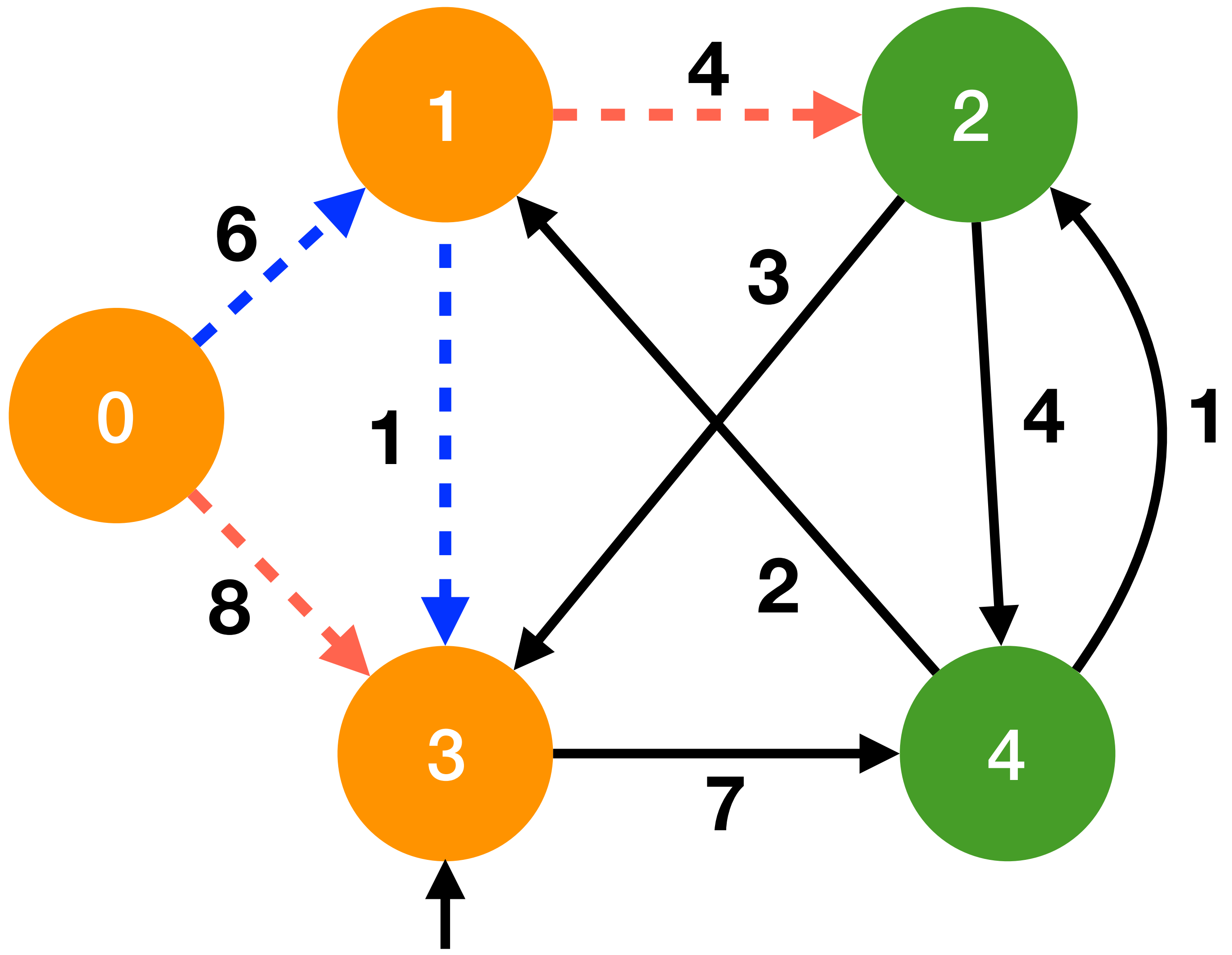


	Q		
id	3	3	2
d	7	8	10

	d	Path	Known
0	0	-1	T
1	6	0	T
2	10	1	F
3	7	1	F
4	∞	-1	F

Update each vertex v adjacent to 1 if known[v] is false

Improved Dijkstra's algorithm

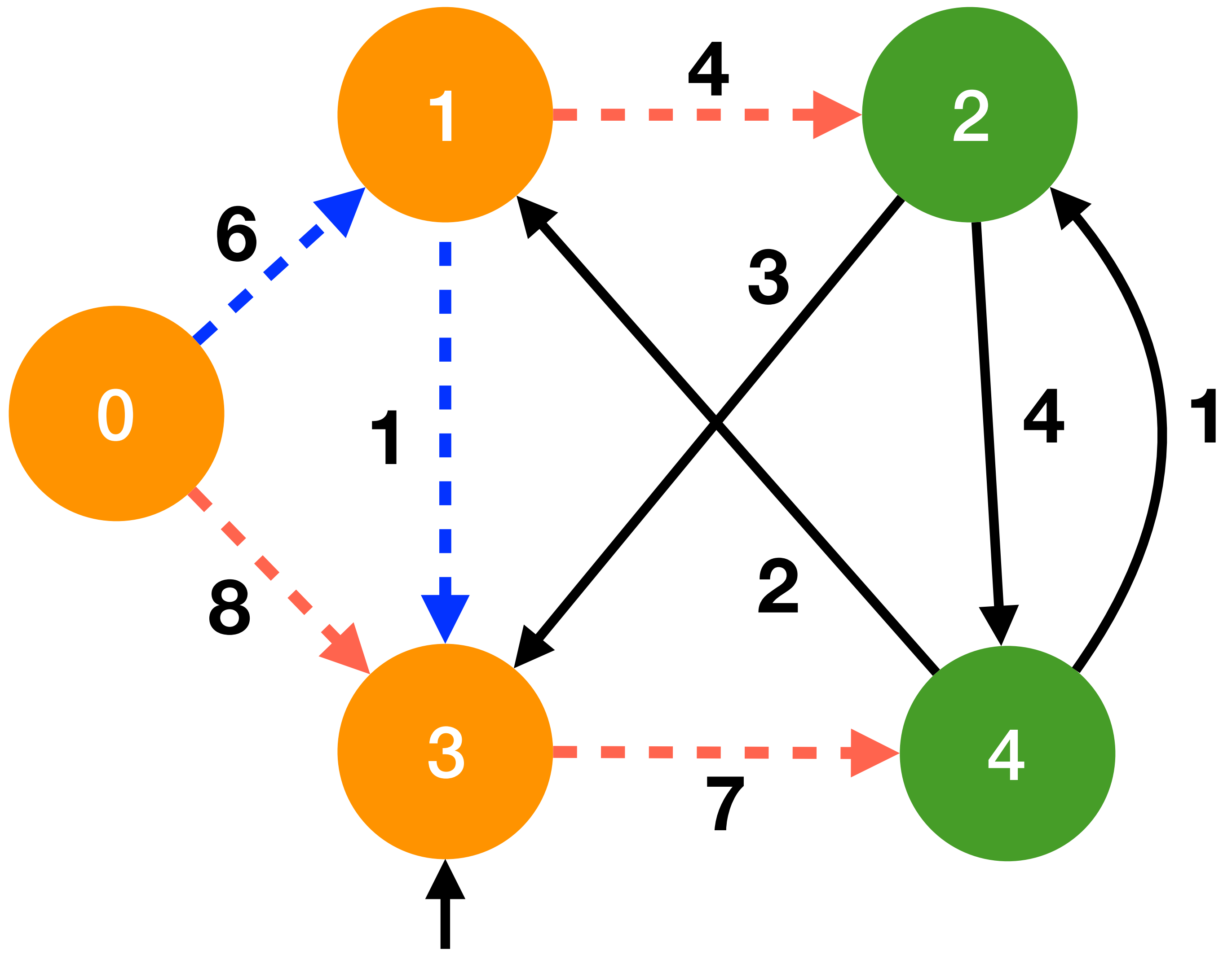


	Q	
id	3	2
d	8	10

	d	Path	Known
0	0	-1	T
1	6	0	T
2	10	1	F
3	7	1	T
4	∞	-1	F

Remove vertex with smallest d from Q(3)

Improved Dijkstra's algorithm

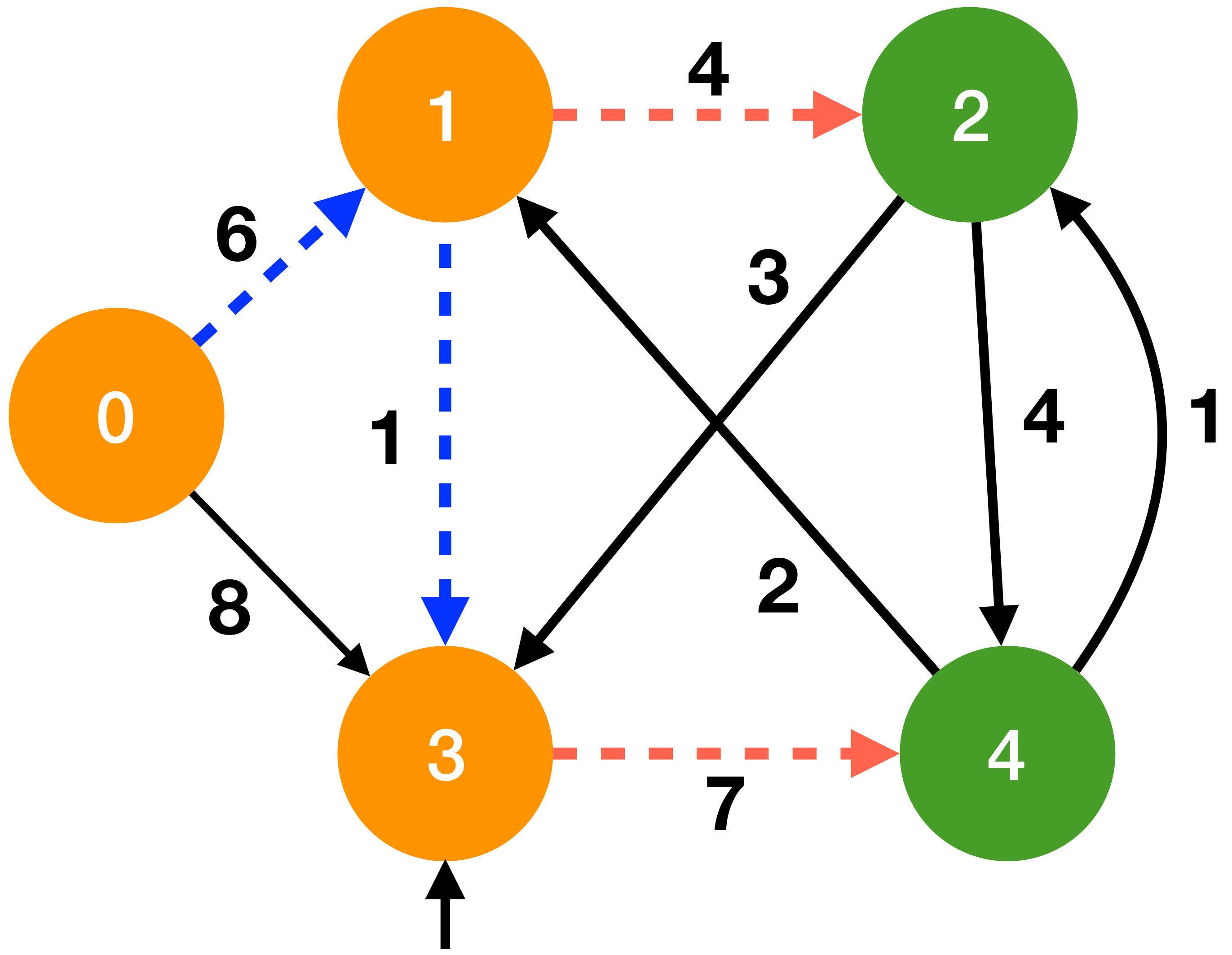


	Q		
id	3	2	4
d	8	10	14

	d	Path	Known
0	0	-1	T
1	6	0	T
2	10	1	F
3	7	1	T
4	**14**	**3**	**F**

Update each vertex v adjacent to 3 if known[v] is false

Improved Dijkstra's algorithm

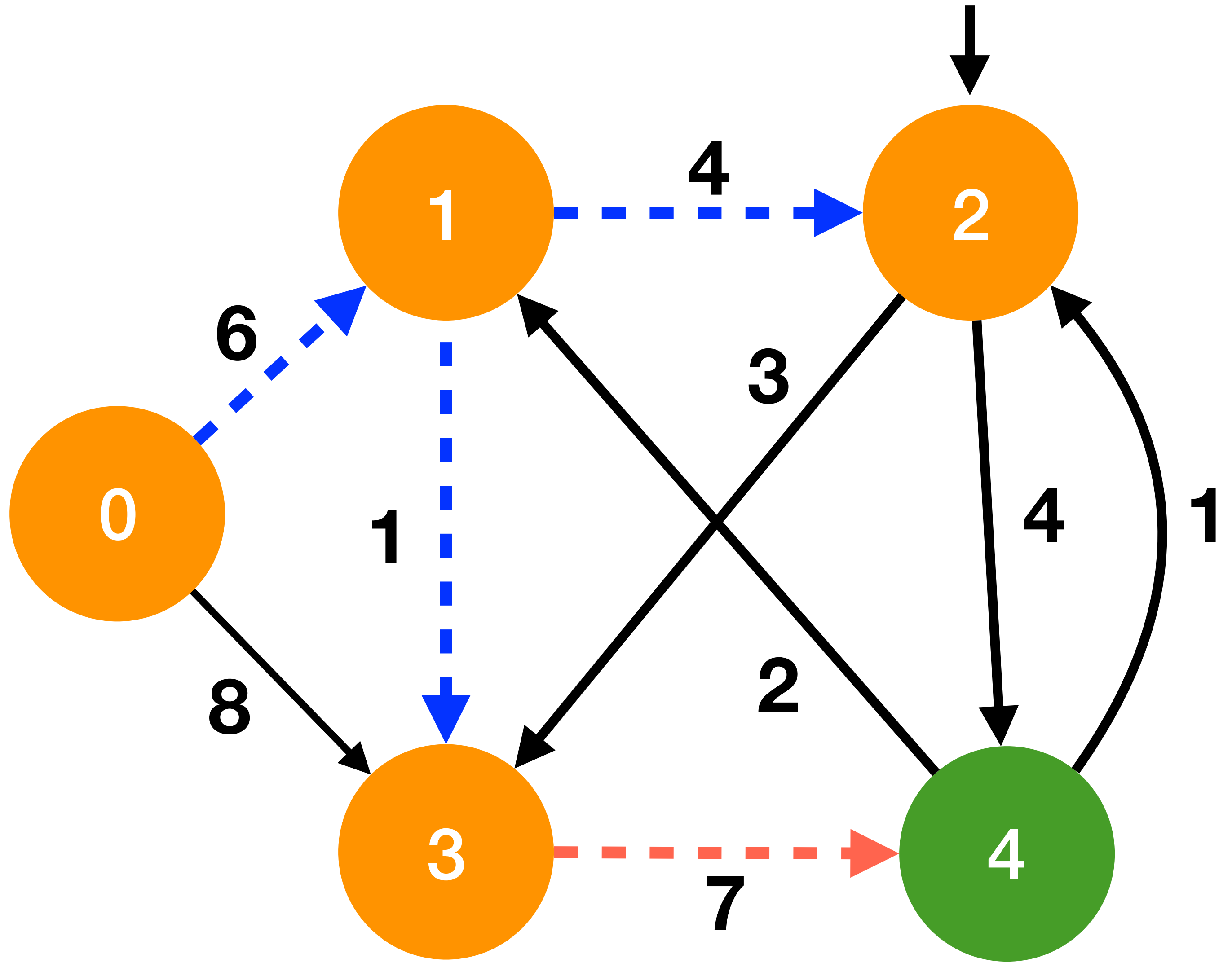


	Q	
id	2	4
d	10	14

	d	Path	Known
0	0	-1	T
1	6	0	T
2	10	1	F
3	7	1	T
4	**14**	**3**	**F**

Remove vertex with smallest d from Q(3) Known[3] is true, discard 3

Improved Dijkstra's algorithm

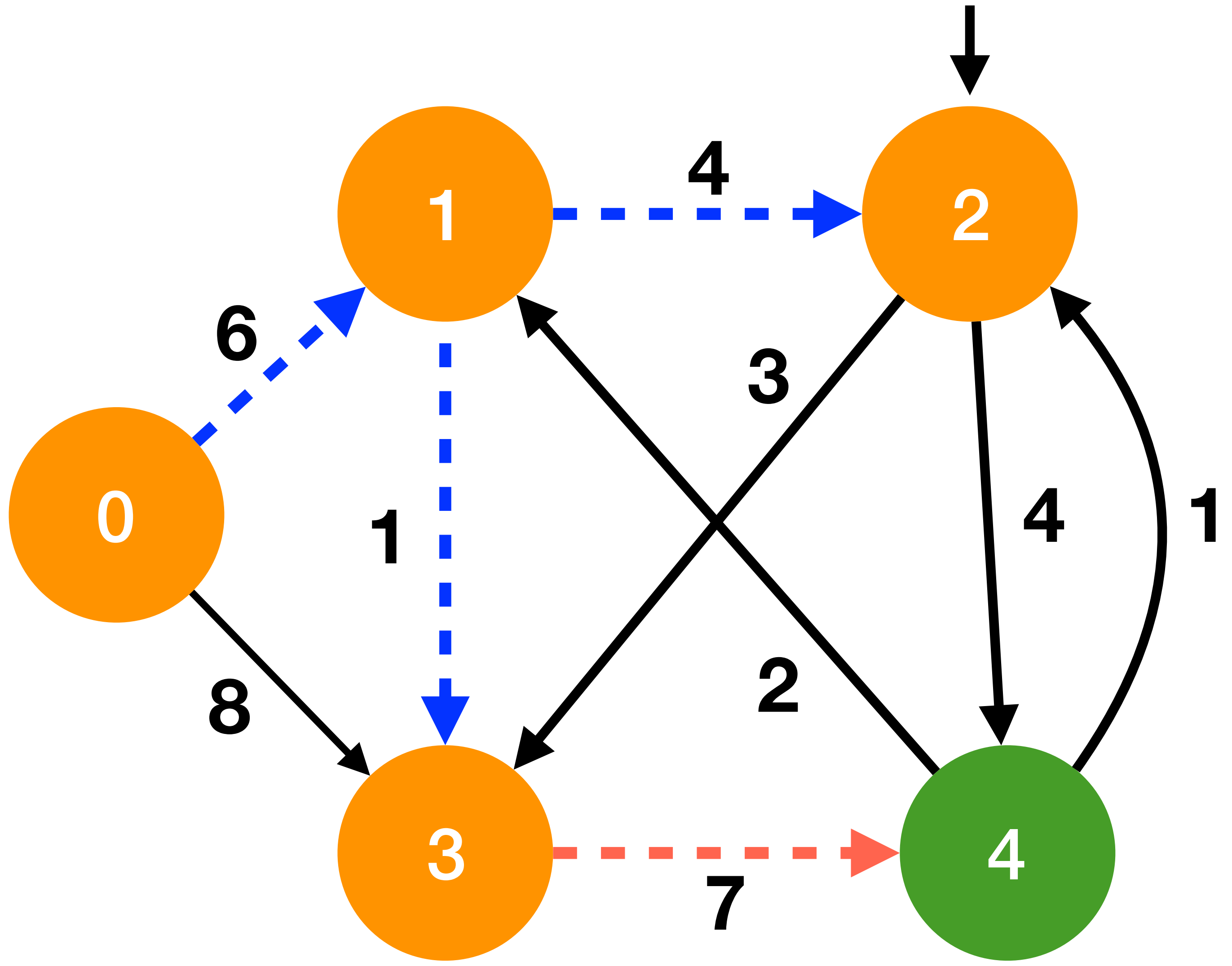


	Q
id	4
d	14

	d	Path	Known
0	0	-1	T
1	6	0	T
2	10	1	T
3	7	1	T
4	14	3	F

Remove vertex with smallest d from Q(2)

Improved Dijkstra's algorithm

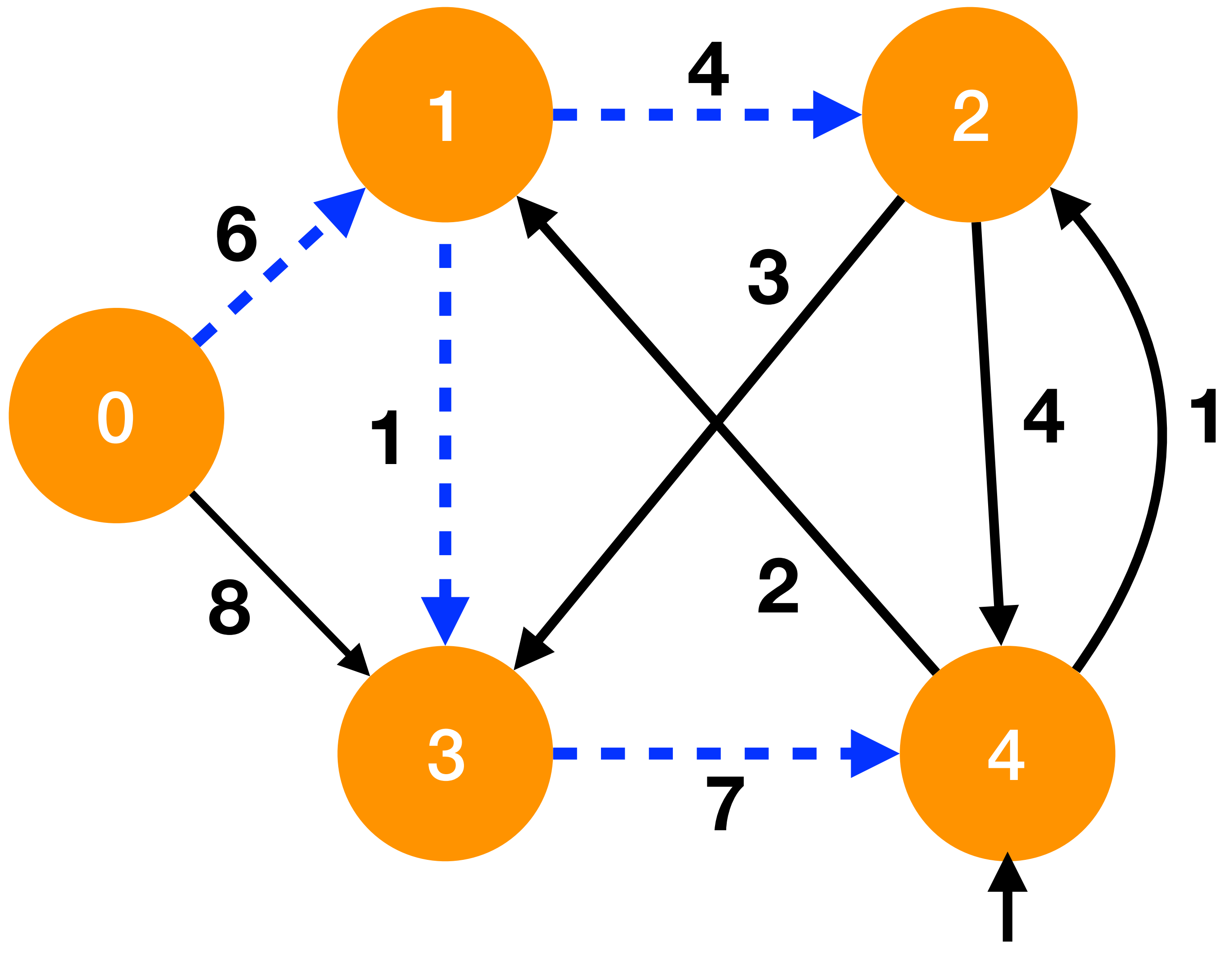


	Q
id	4
d	14

	d	Path	Known
0	0	-1	T
1	6	0	T
2	10	1	T
3	7	1	T
4	14	3	F

Update vertex 4 adjacent to 2, but new distance from 2 is not shorter :
so 4 is not updated

Improved Dijkstra's algorithm

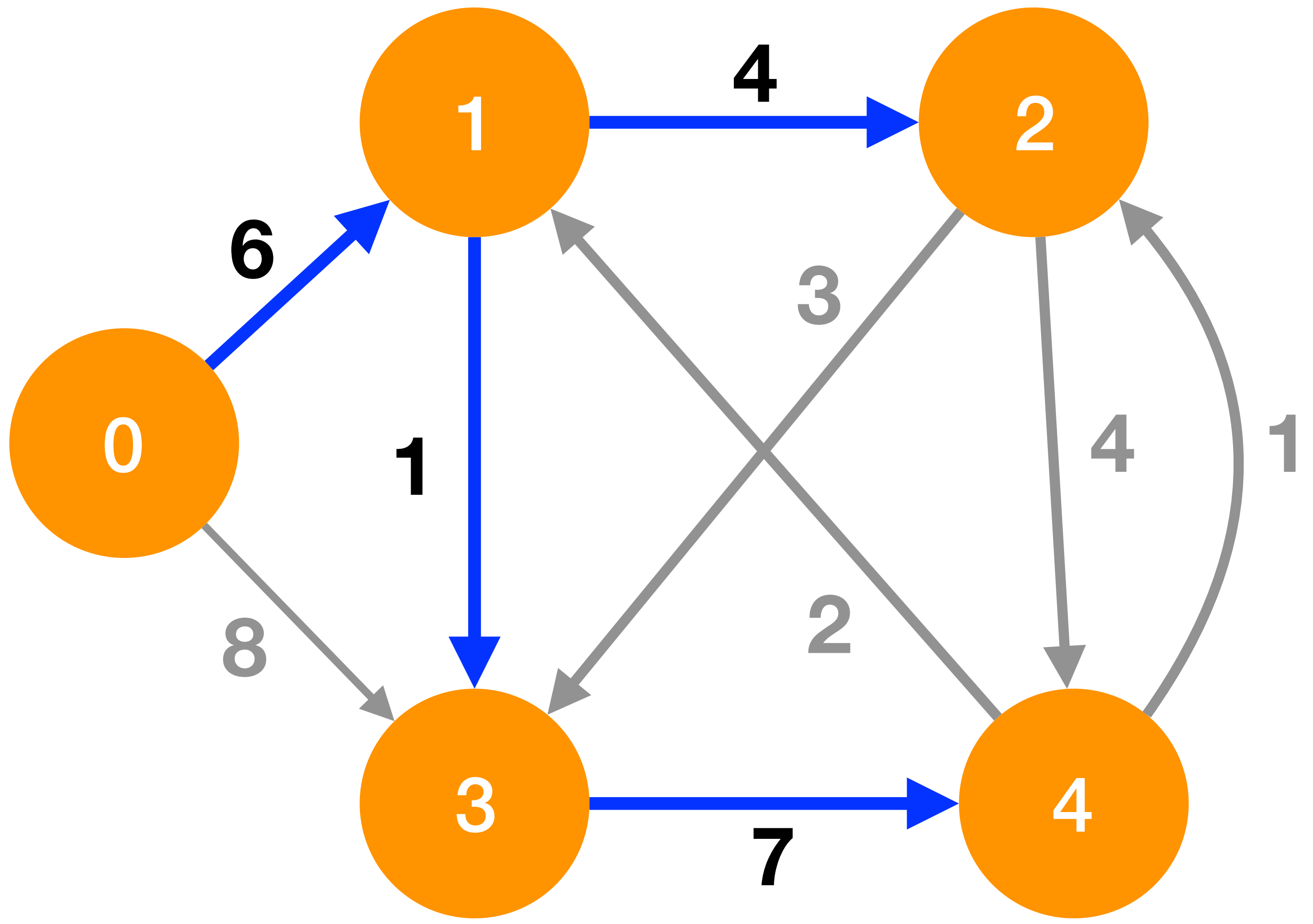


	Q
id	
d	

	d	Path	Known
0	0	-1	T
1	6	0	T
2	10	1	T
3	7	1	T
4	14	3	T

Remove vertex with smallest d from Q(4)

Improved Dijkstra's algorithm



	d	Path
0	0	-1
1	6	0
2	10	1
3	7	1
4	14	3

Solution

Dijkstra’s algorithm

```
void dijkstra(Graph myGraph, BinaryHeap heap, bool *known, int *distance, int *path) {
        cout << "Dijkstra's algorithm" <<  endl;

        while (!heap.isEmpty()) {
            Node * node = heap.deleteMin();
            int u = node->value; // vertex with minimum distance
            known[u] = true;    // shortest path to u has been found
            cout << "Deleted min key with id " << u << " and distance " << node->distance << " from heap" << endl;

            // check each vertex w adjacent to v in adjacency list
            ListNode *ptr = myGraph.getAdjListHead(u);
            while (ptr != NULL) {
                int v = ptr->value;

                if (!known[v]) {
                    cout << "Adjacent vertex of node with id " << u << " is " << v <<  endl;
                    if (distance[u]+ ptr->weight < distance[v]) {    // update distance[v] if this is shorter
                            distance[v] = distance[u] + ptr->weight;
                            path[v] = u;
                            cout << "updated distance of " << v << " to " << distance[v] << endl;
                            Node * newNode = new Node();
                            newNode->value = v;
                            newNode->distance = distance[v];
                            cout << "Inserting new node into heap " << newNode->value << "  " << newNode->distance << endl;
                            heap.insert(newNode);
                            heap.print();
                    }
                }
                ptr = ptr->next;
            }
        }
    }
```

Dijkstra's algorithm on grid

	0	1	2	3	4
0					
1			1		
2		1	S	1	
3			1		
4					D

Dijkstra's algorithm on grid

	0	1	2	3	4
0			2		
1		2	1	2	
2	2	1	S	1	2
3		2	1	2	
4			2		D

Dijkstra's algorithm on grid

	0	1	2	3	4
0		3	2	3	
1	3	2	1	2	3
2	2	1	S	1	2
3	3	2	1	2	3
4		3	2	3	D

Dijkstra’s algorithm on grid

	0	1	2	3	4
0	4	3	2	3	4
1	3	2	1	2	3
2	2	1	S	1	2
3	3	2	1	2	3
4	4	3	2	3	D

References

- Cormen, Thomas H., et al. *Introduction to Algorithms*. The MIT Press, 2014.
- https://en.wikipedia.org/wiki/A*_search_algorithm