Single Source Shortest Path Problem

Outline for Today

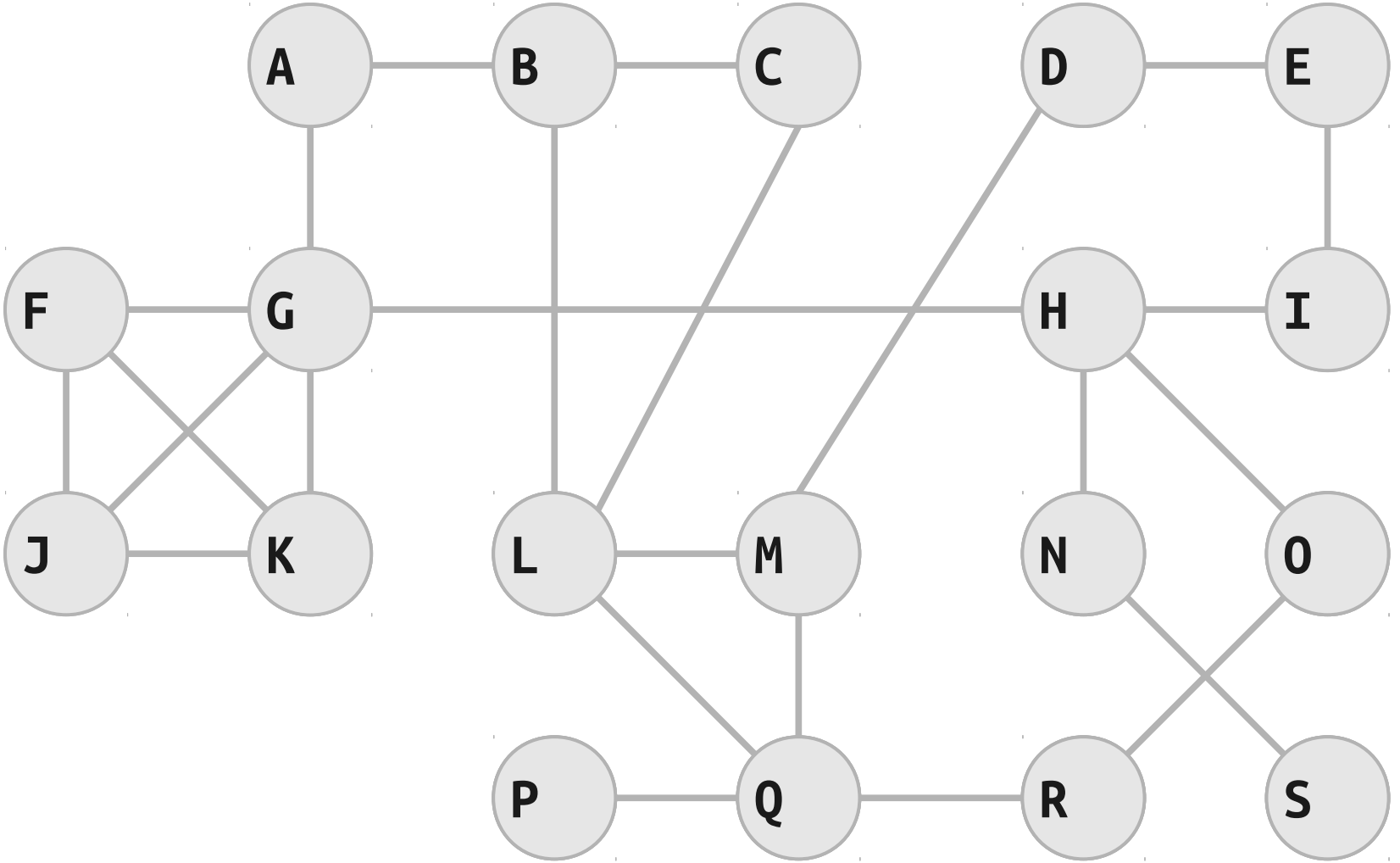
- **BFS/DFS**
- **Dijkstra's Algorithm**
  - An algorithm for finding shortest paths in more realistic settings
- **Bellman-Ford**
- **Practical Concerns**

Breadth-First Search

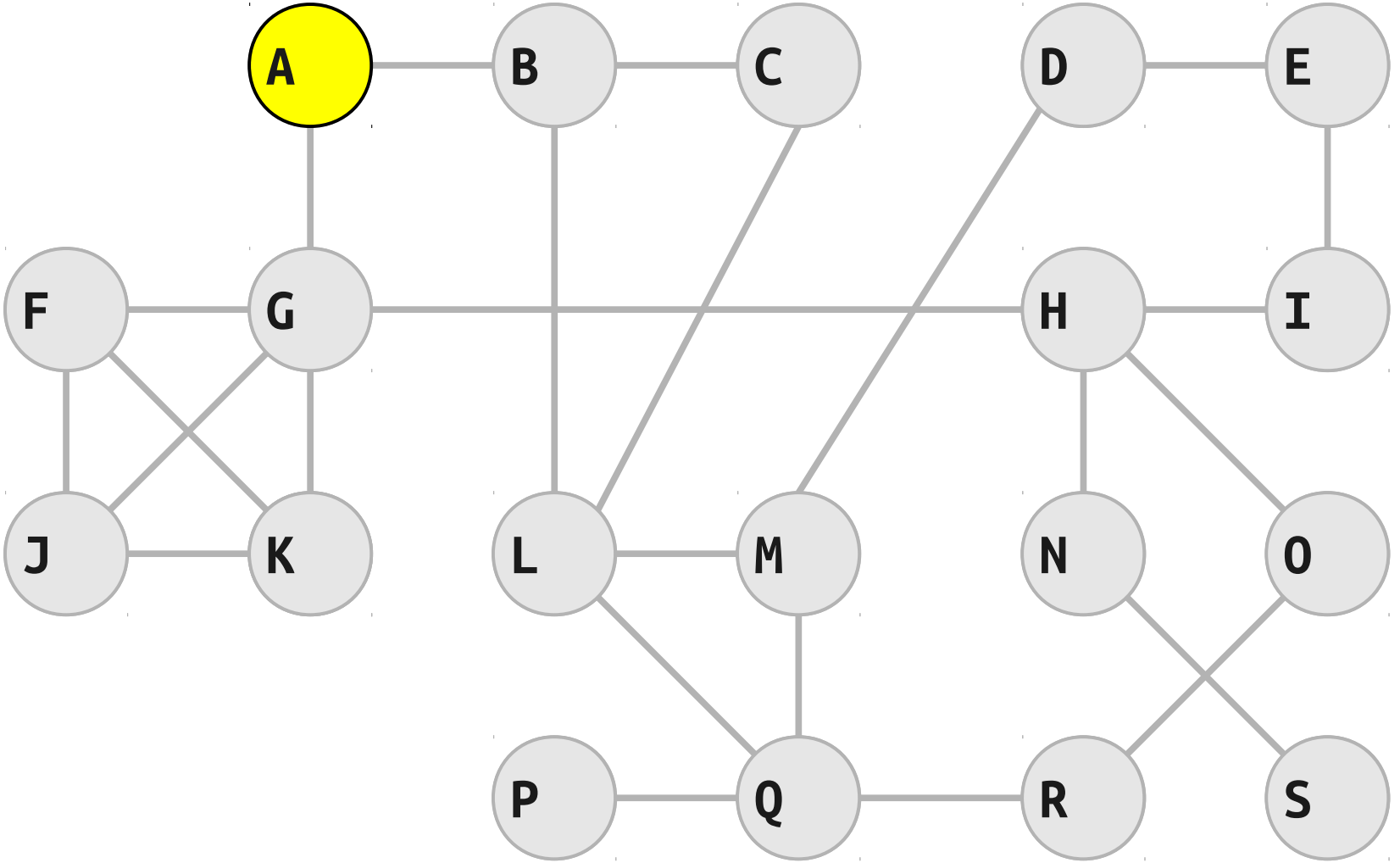
- Given an arbitrary graph $G = (V, E)$ and a starting node $s \in V$, **breadth-first search** finds shortest paths from s to each reachable node v.
- When implemented using an adjacency list, runs in $O(m + n)$ time, which we defined to be linear time on a graph.
- One correctness proof worked in terms of "layers:" the algorithm finds all nodes at distance 0, 1, 2, ... in order.

A Second Intuition for BFS

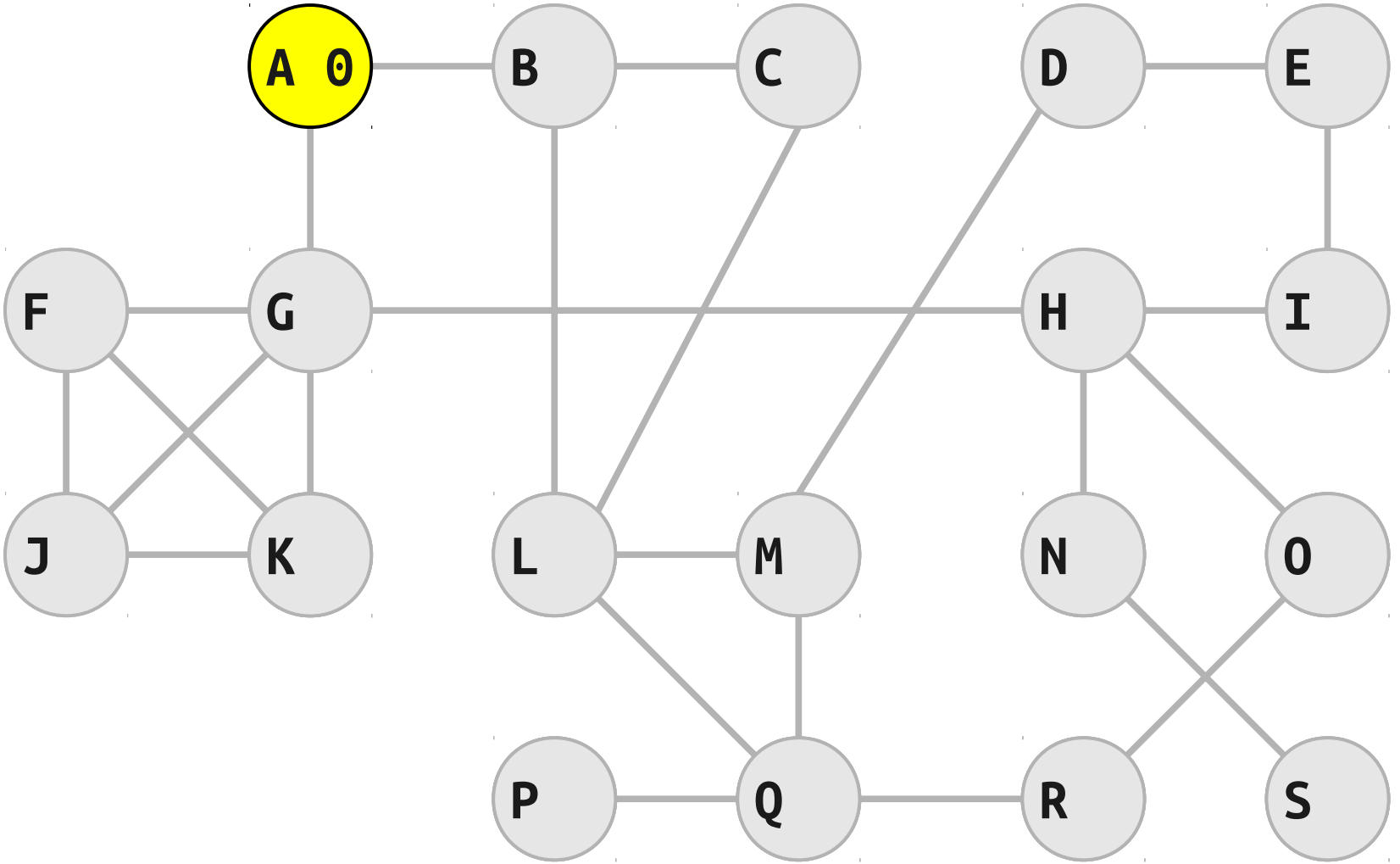
Breadth-First Search



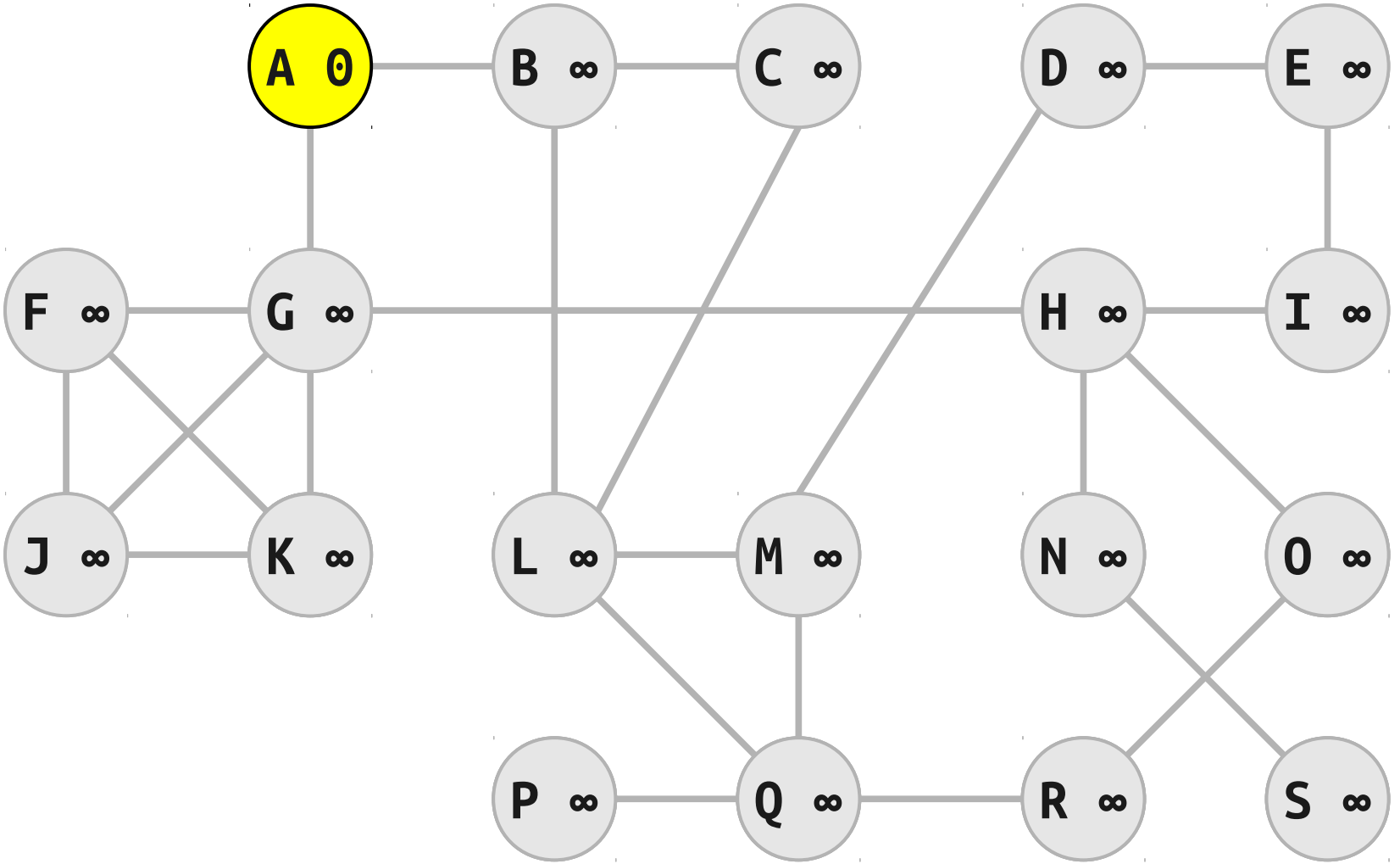
Breadth-First Search



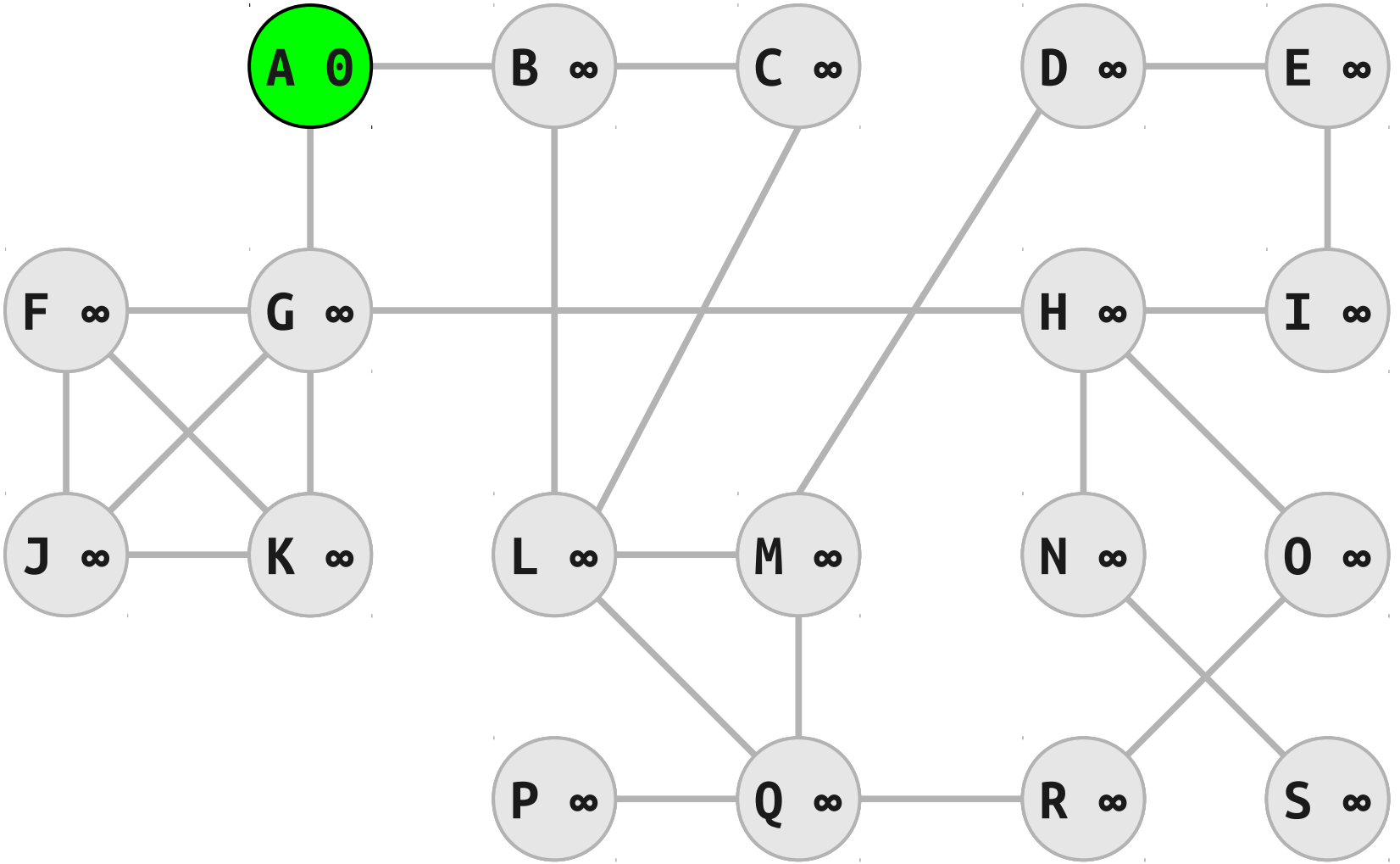
Breadth-First Search



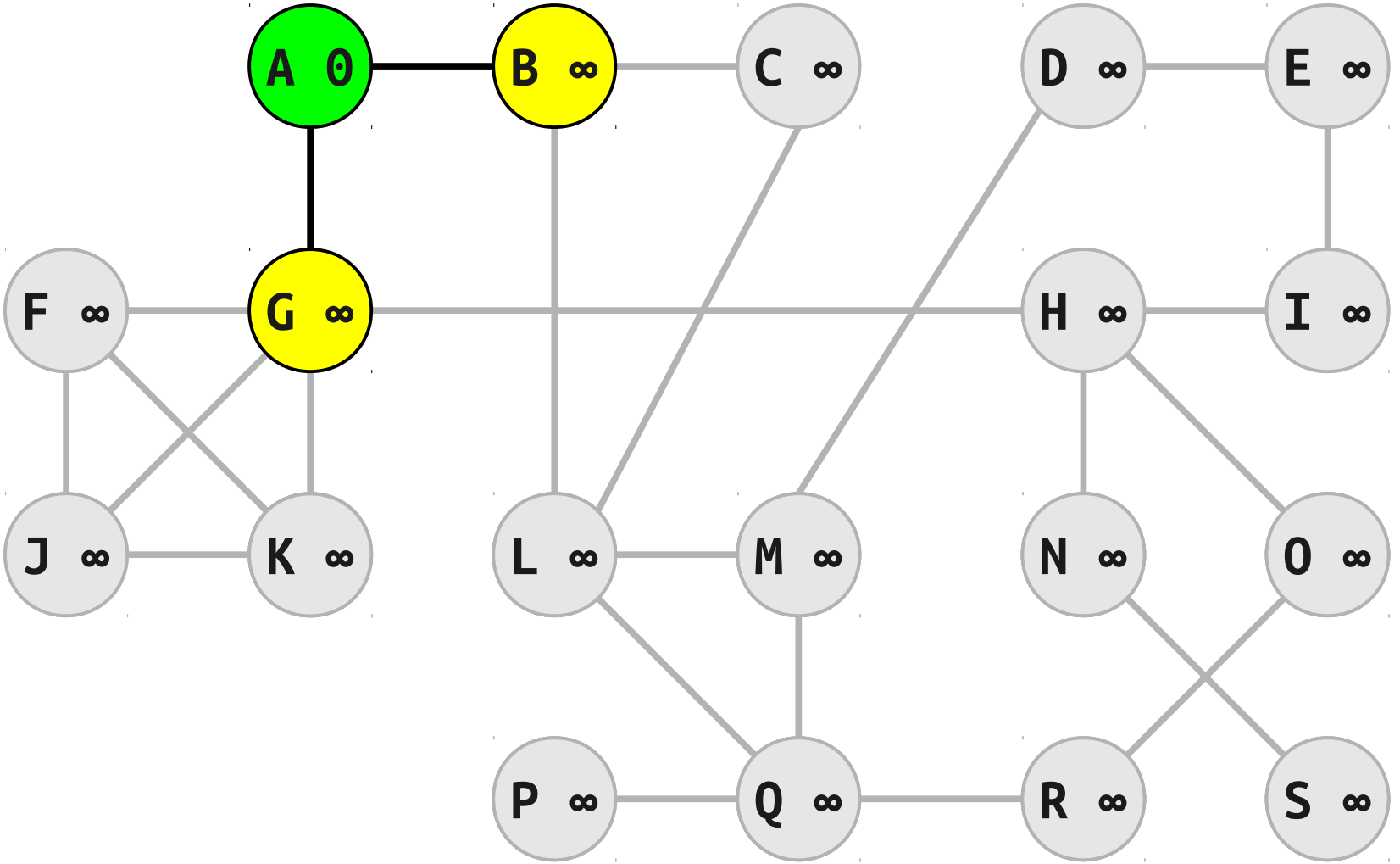
Breadth-First Search



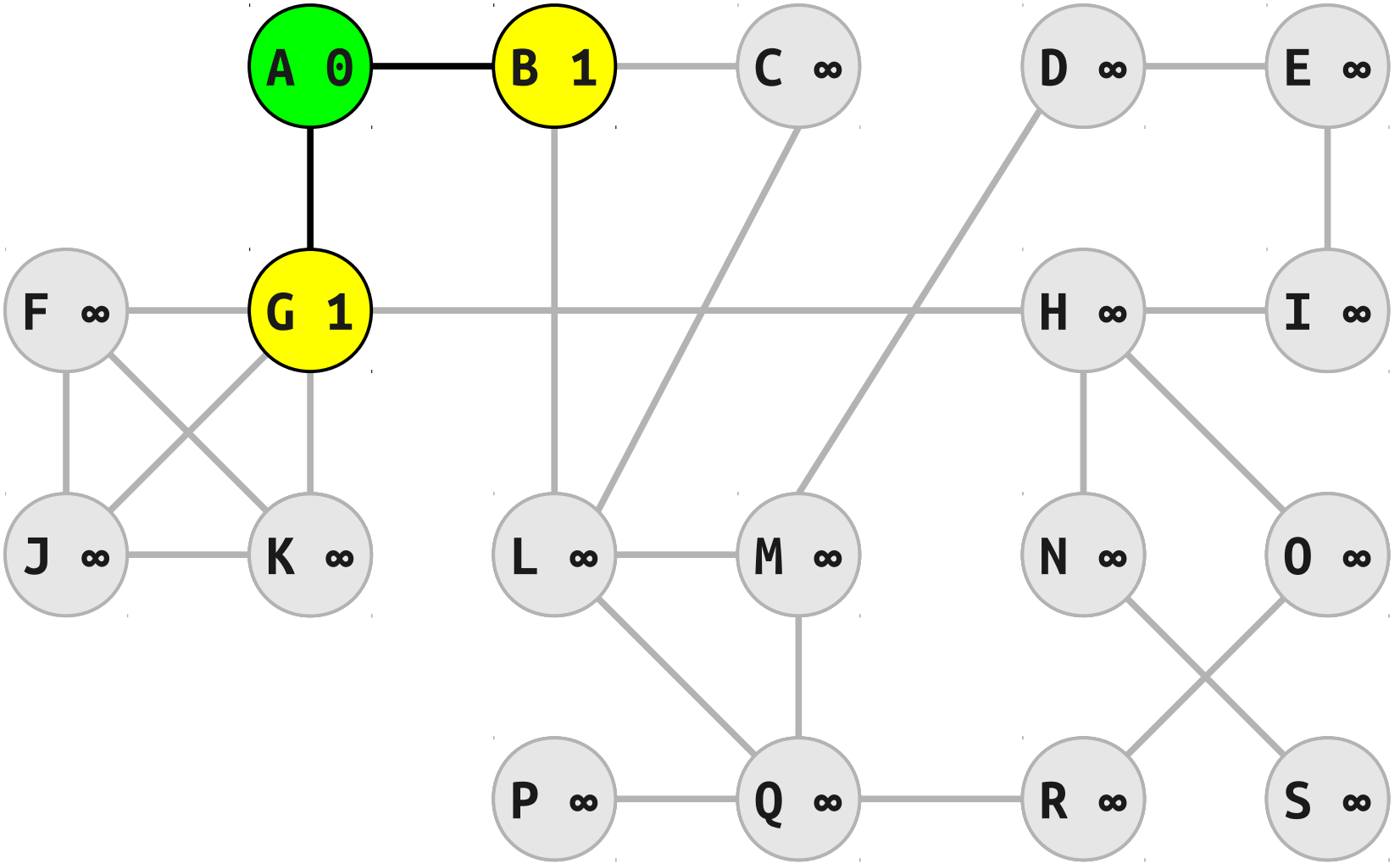
Breadth-First Search



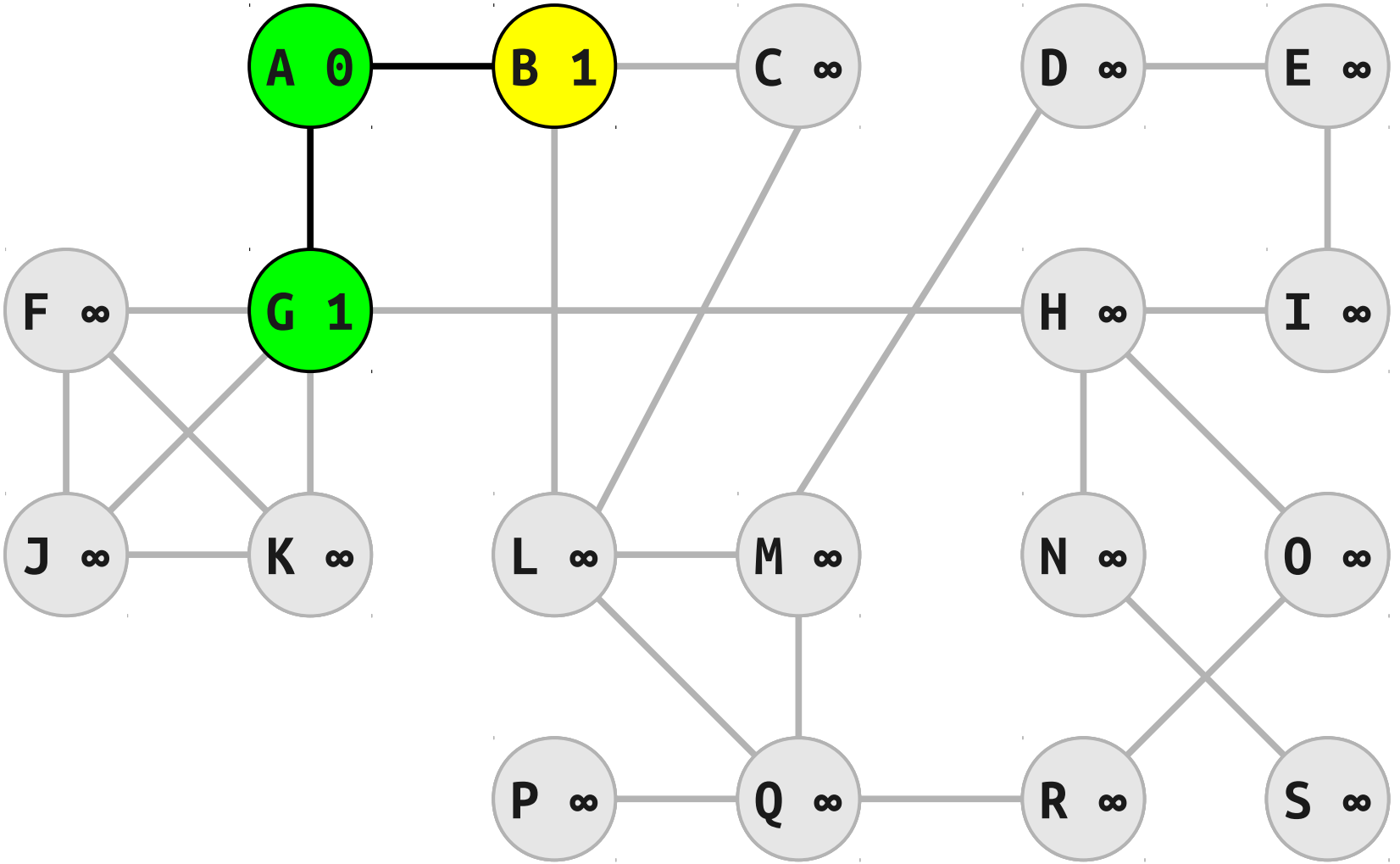
Breadth-First Search



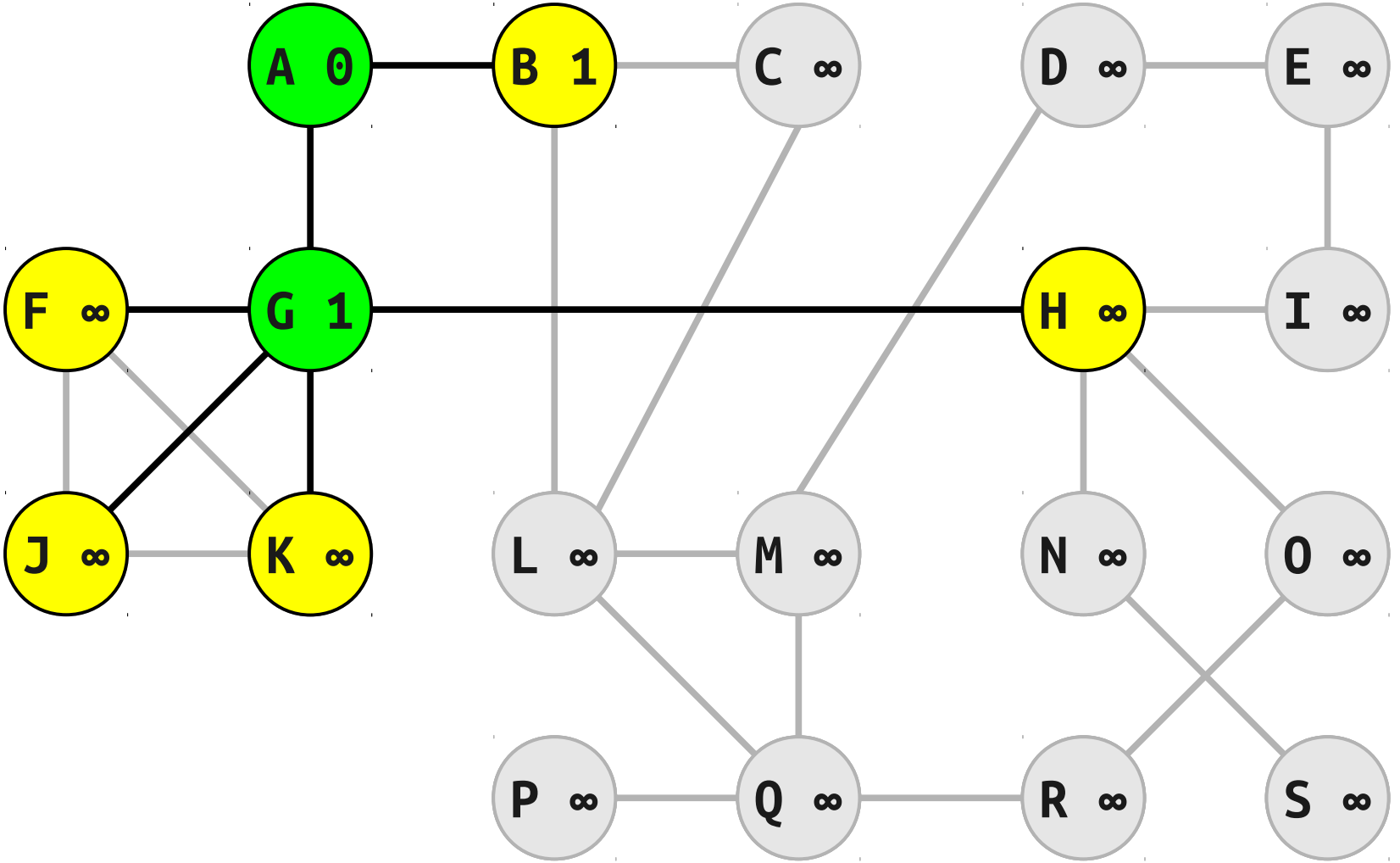
Breadth-First Search



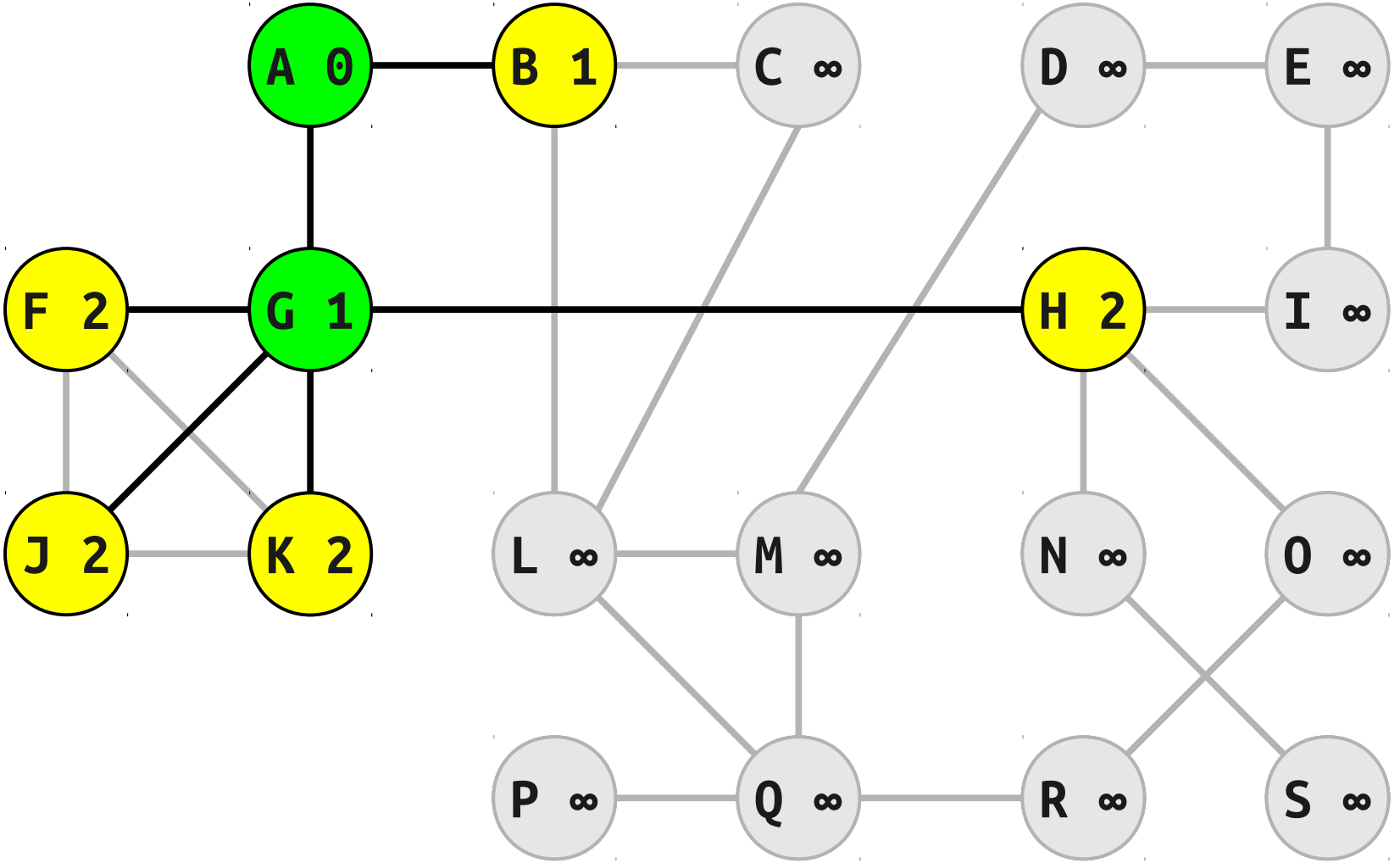
Breadth-First Search



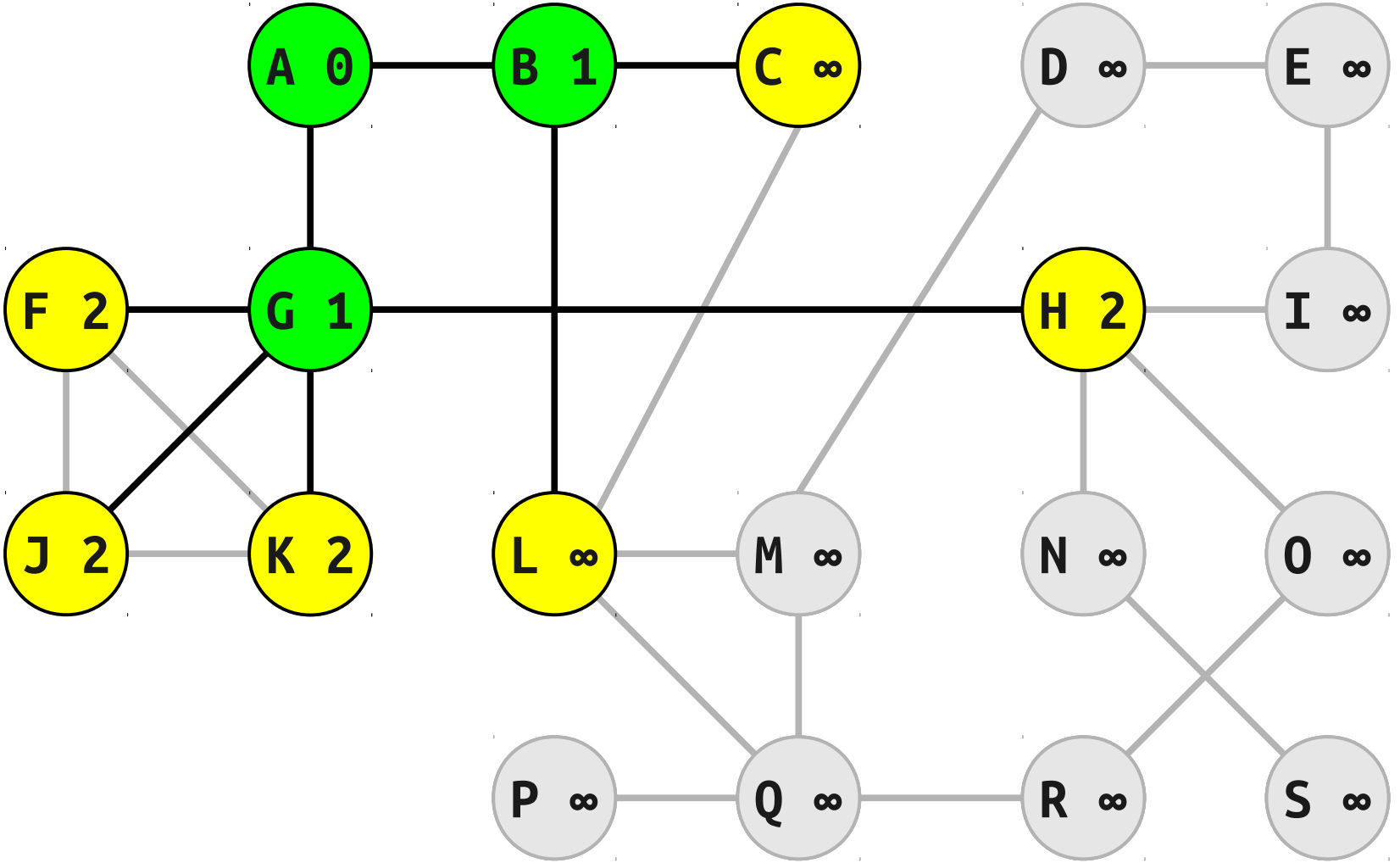
Breadth-First Search



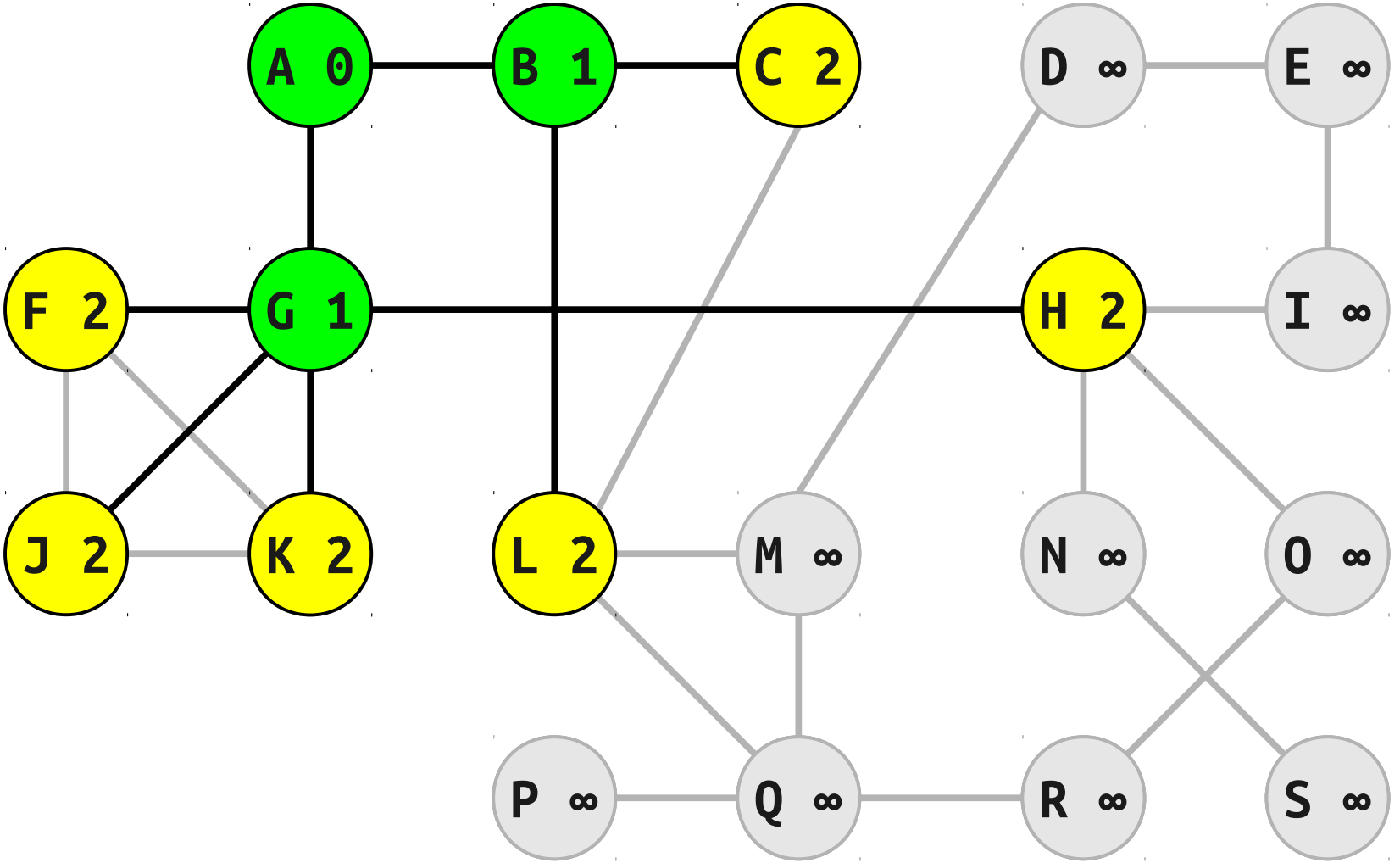
Breadth-First Search



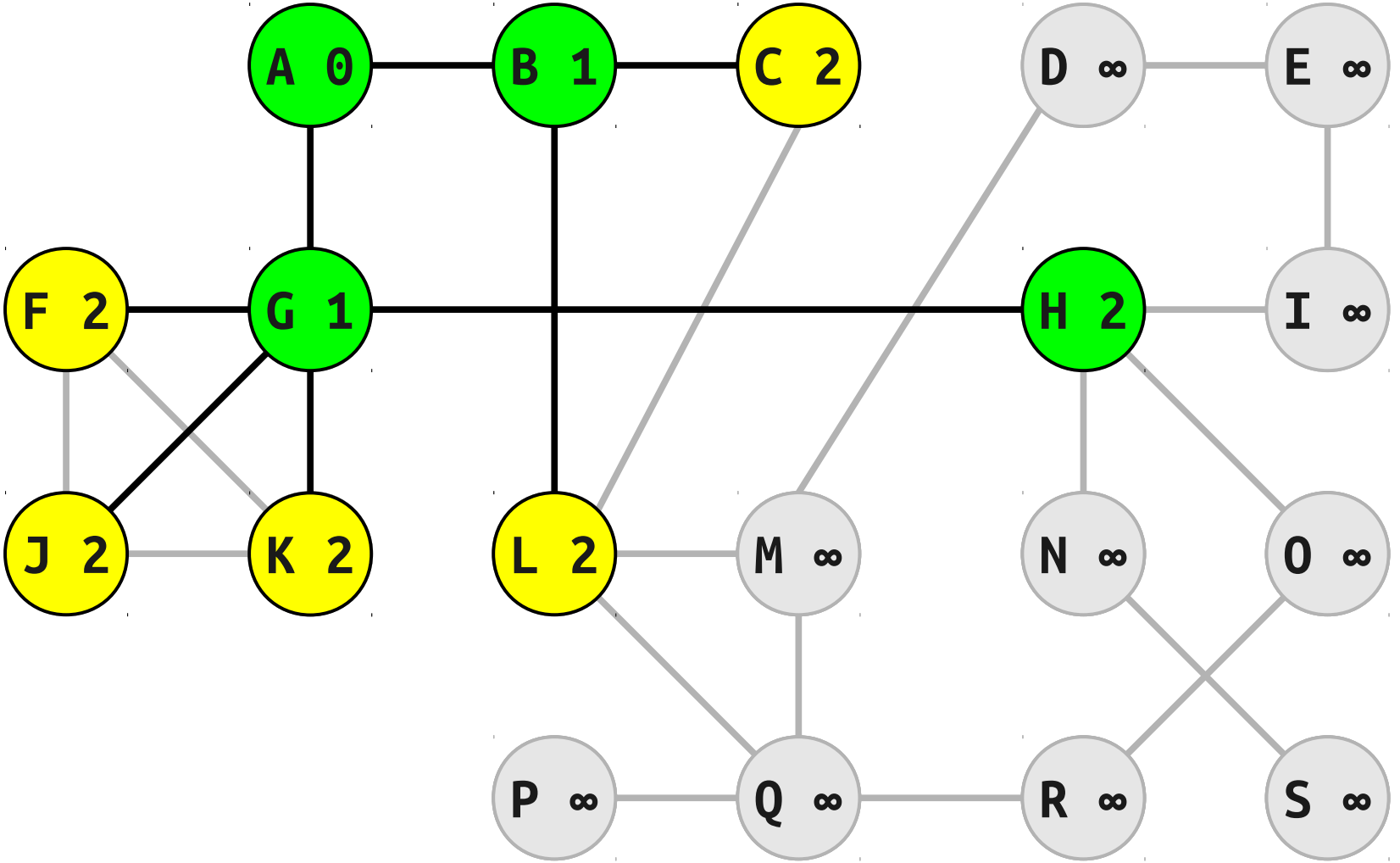
Breadth-First Search



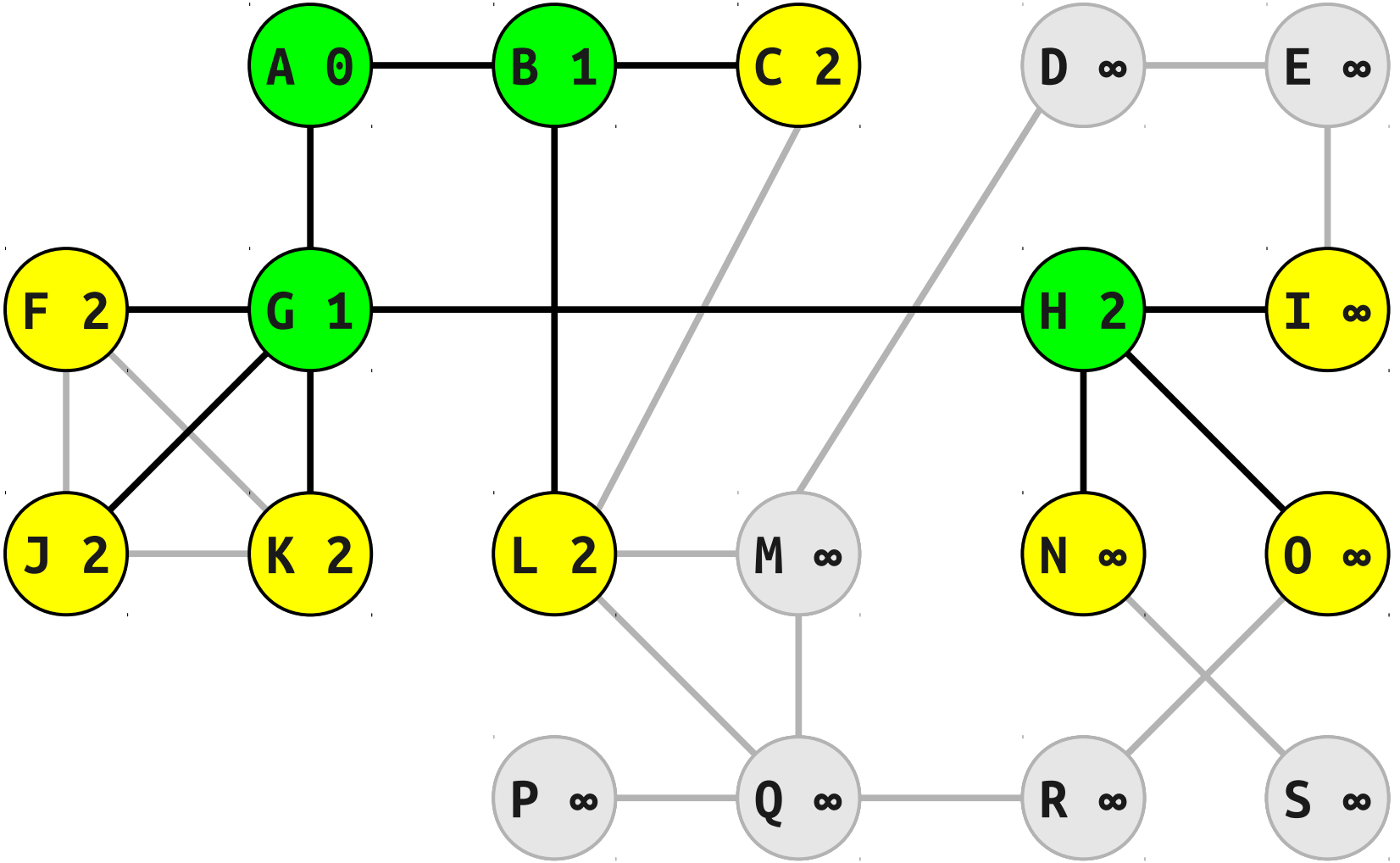
Breadth-First Search



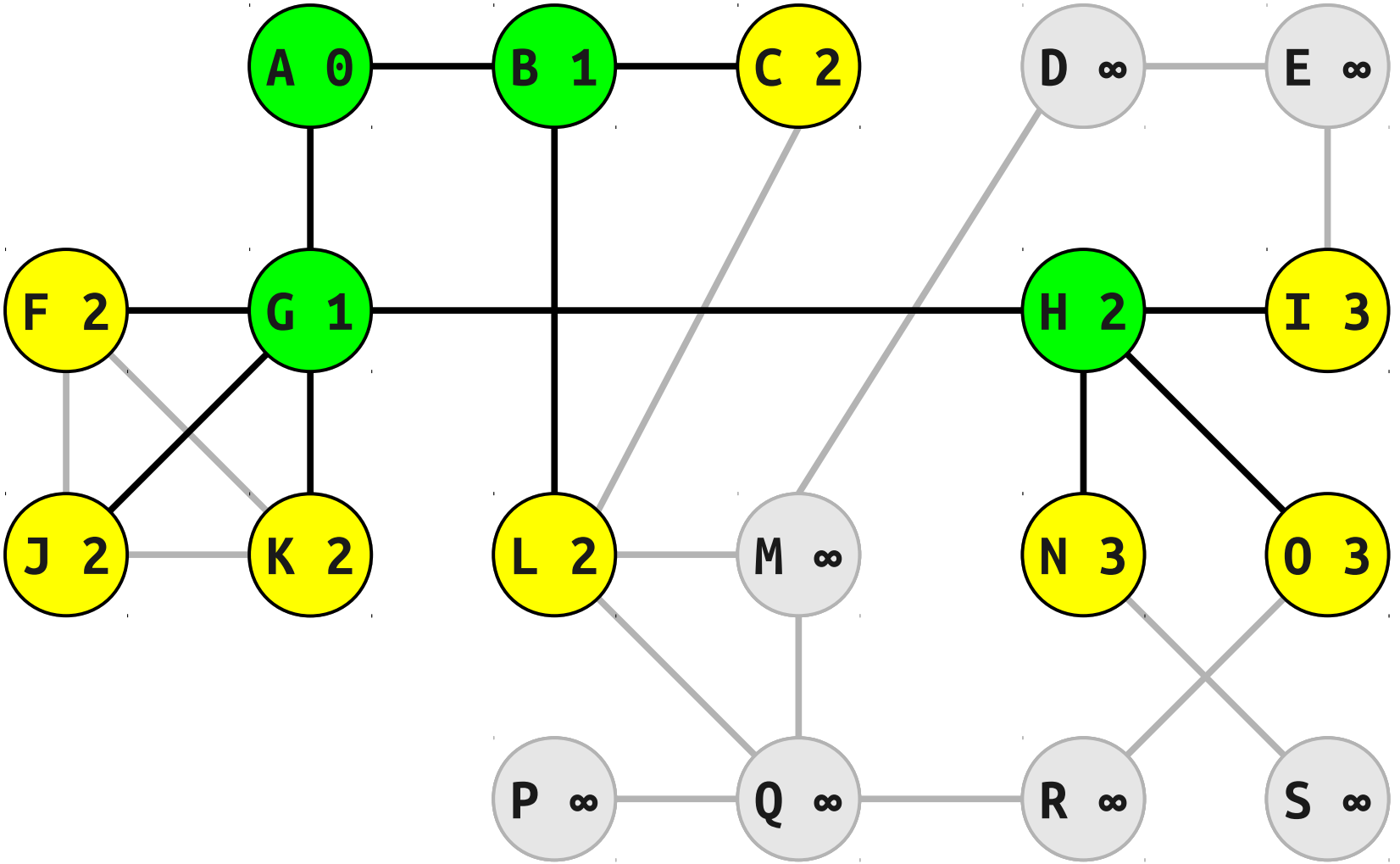
Breadth-First Search



Breadth-First Search



Breadth-First Search



```
procedure breadthFirstSearch(s, G):
    let q be a new queue.
    for each node v in G:
      dist[v] = ∞

    dist[s] = 0
    enqueue(s, q)

    while q is not empty:
       let v = dequeue(q)
       for each neighbor u of v:
          if dist[u] = ∞:
             dist[u] = dist[v] + 1
             enqueue(u, q)
```

O(1)

O(*n*)

O(1)

O(*n*)

O(*m* + *n*)

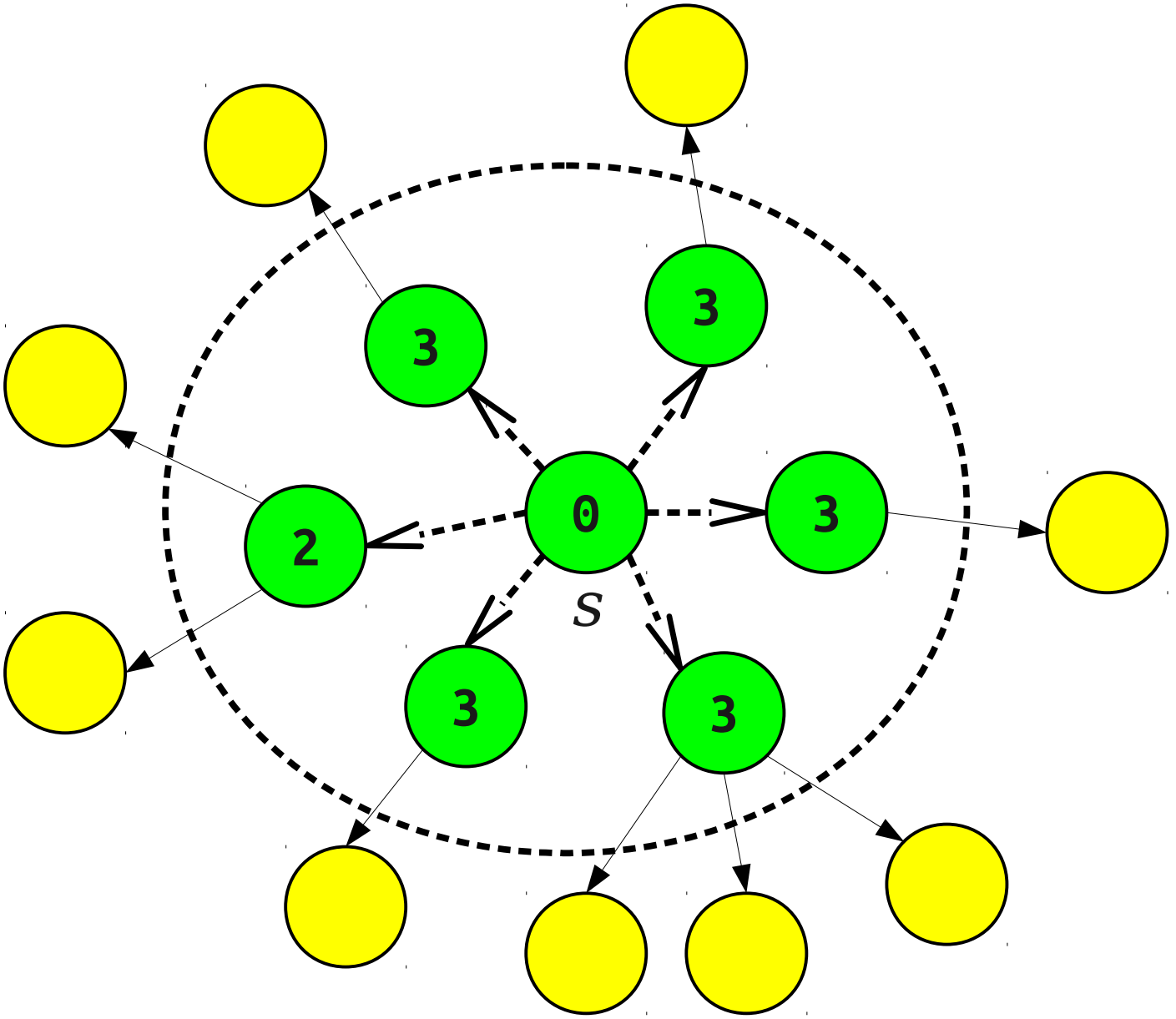
A Second Intuition

- At each point in the execution of BFS, a node v is either
  - green, and we have the shortest path to v;
  - yellow, and it is connected to some green node; or
  - gray, and v is undiscovered.
- Each iteration, we pick a yellow node with minimal distance from the start node and color it green. So what is the cost of the lowest-cost yellow node?
- If v is yellow, it is connected to a green node u by an edge.
- The cost of getting from s to v is then $d(s, u) + 1$.
- BFS works by picking the yellow node v minimizing

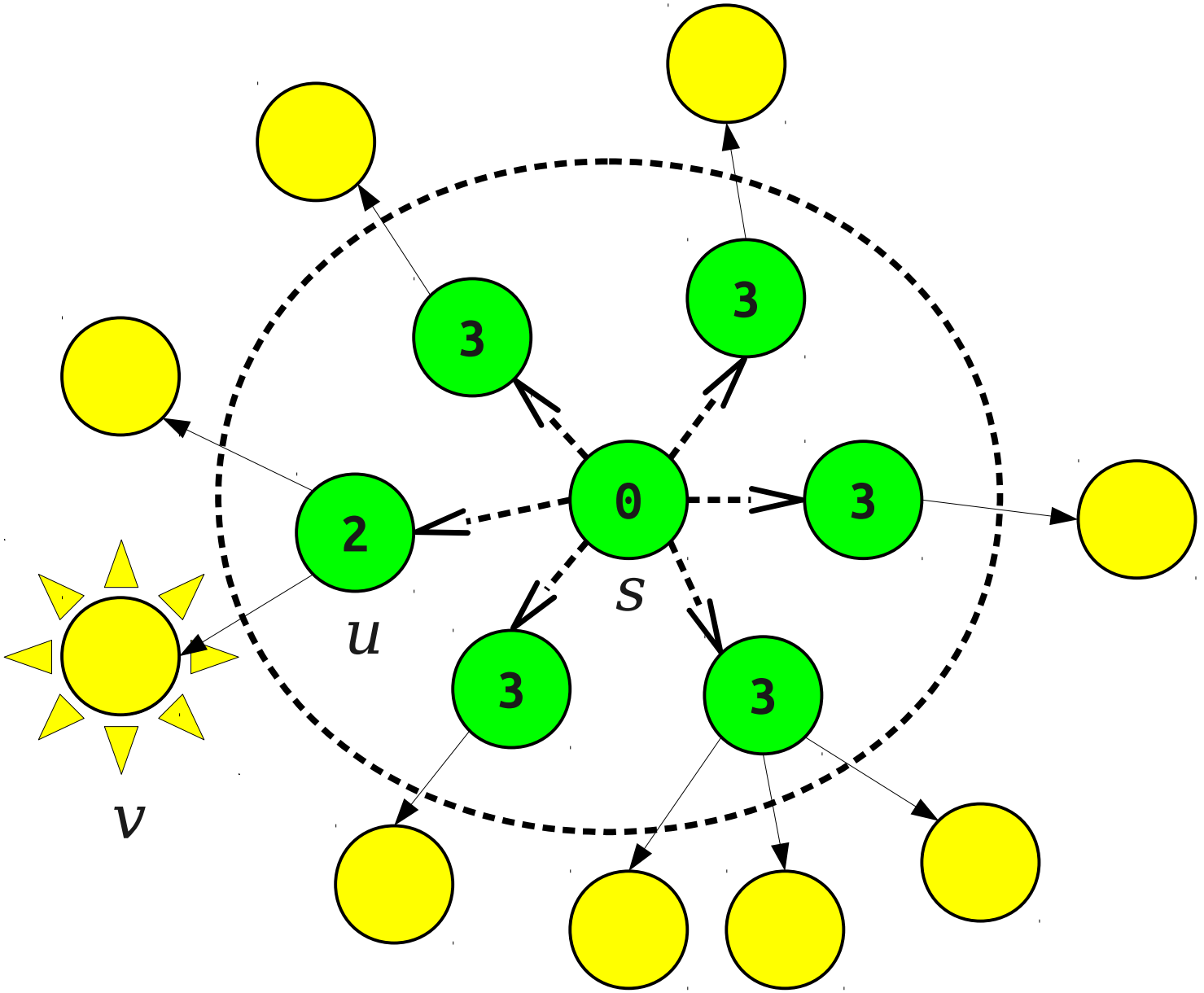
$$\mathbf{d(s, u) + 1}$$

where (u, v) is an edge and u is green.

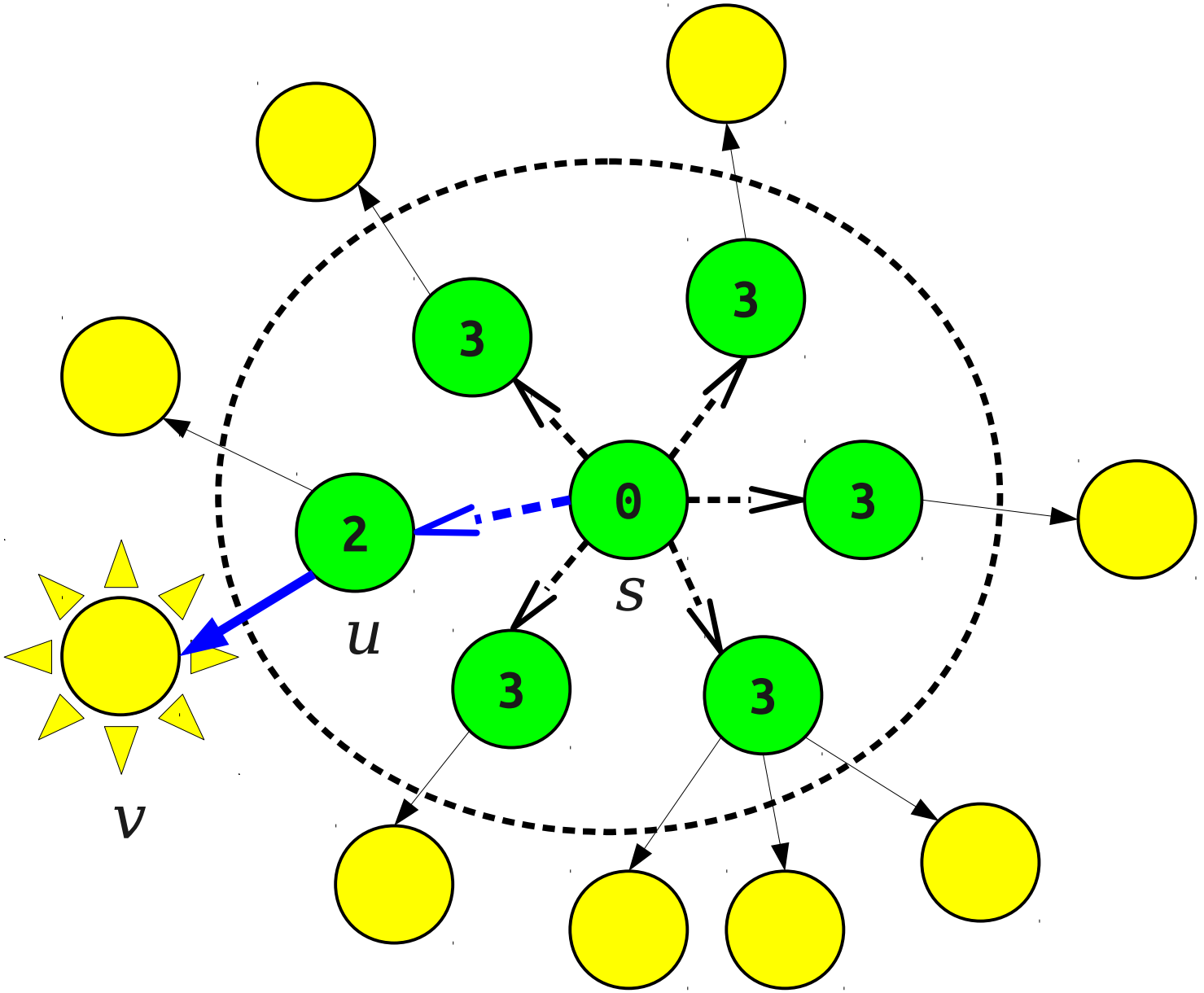
Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.



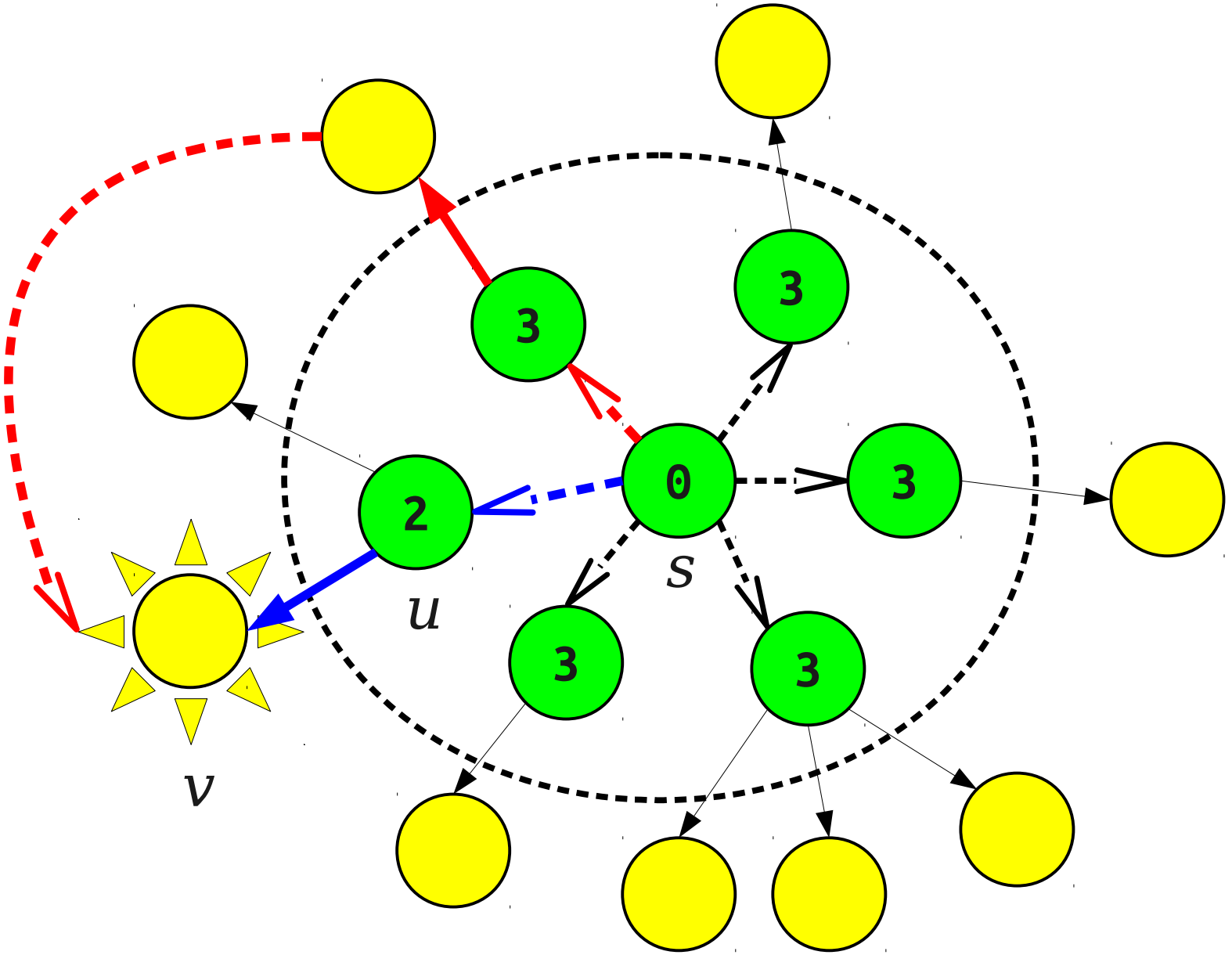
Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.



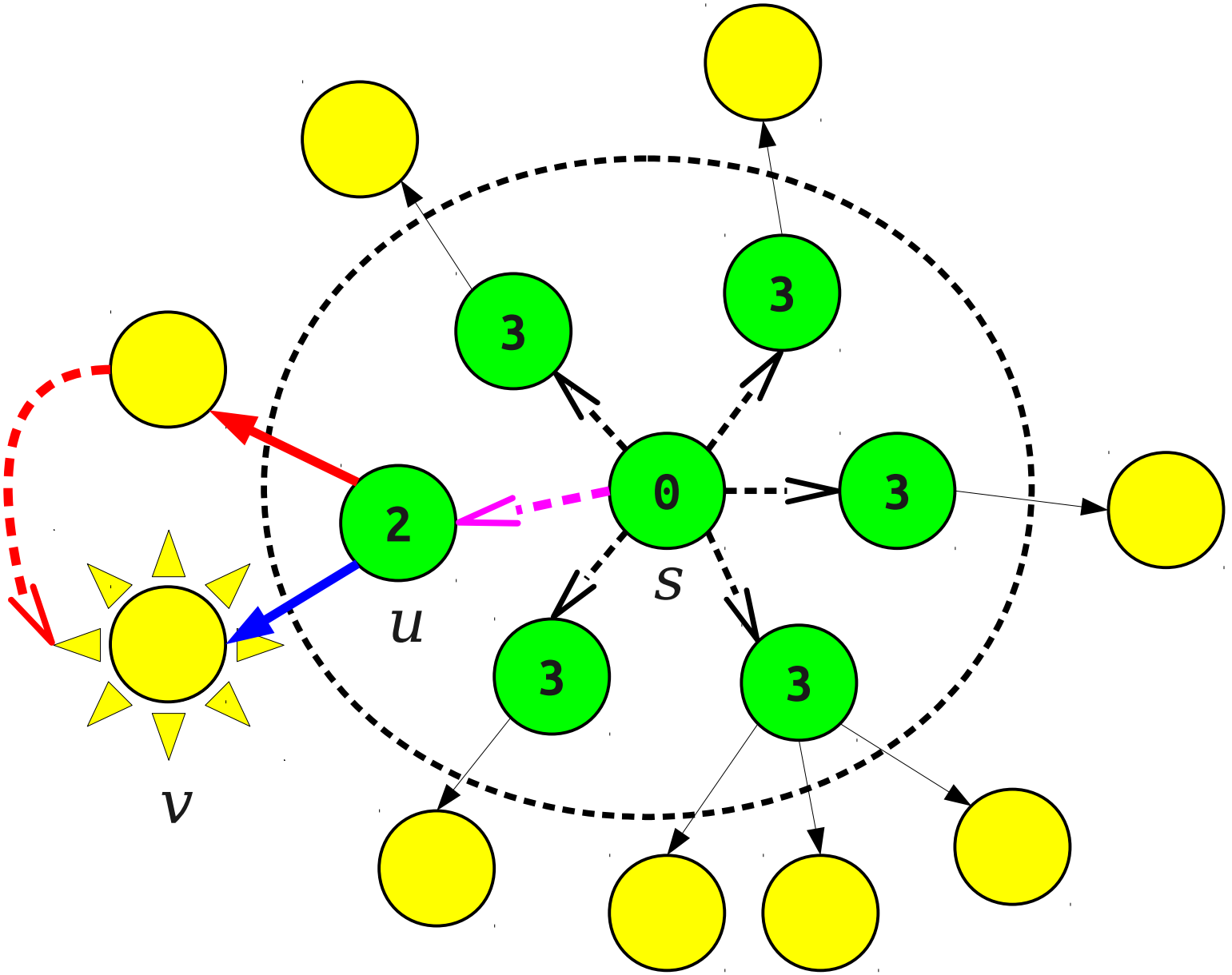
Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.



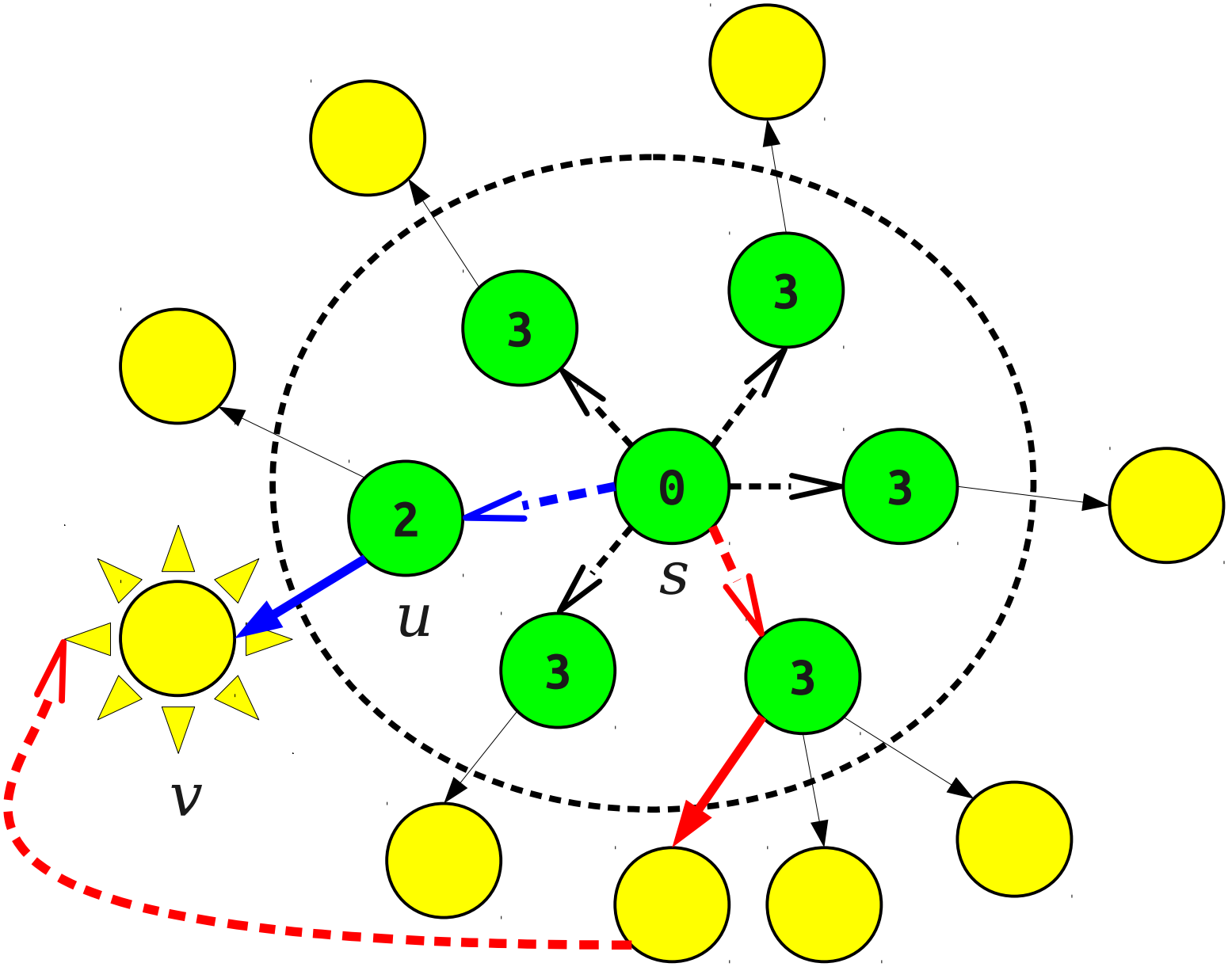
Pick yellow node v minimizing d(s, u) + 1,
where (u, v) is an edge and u is green.



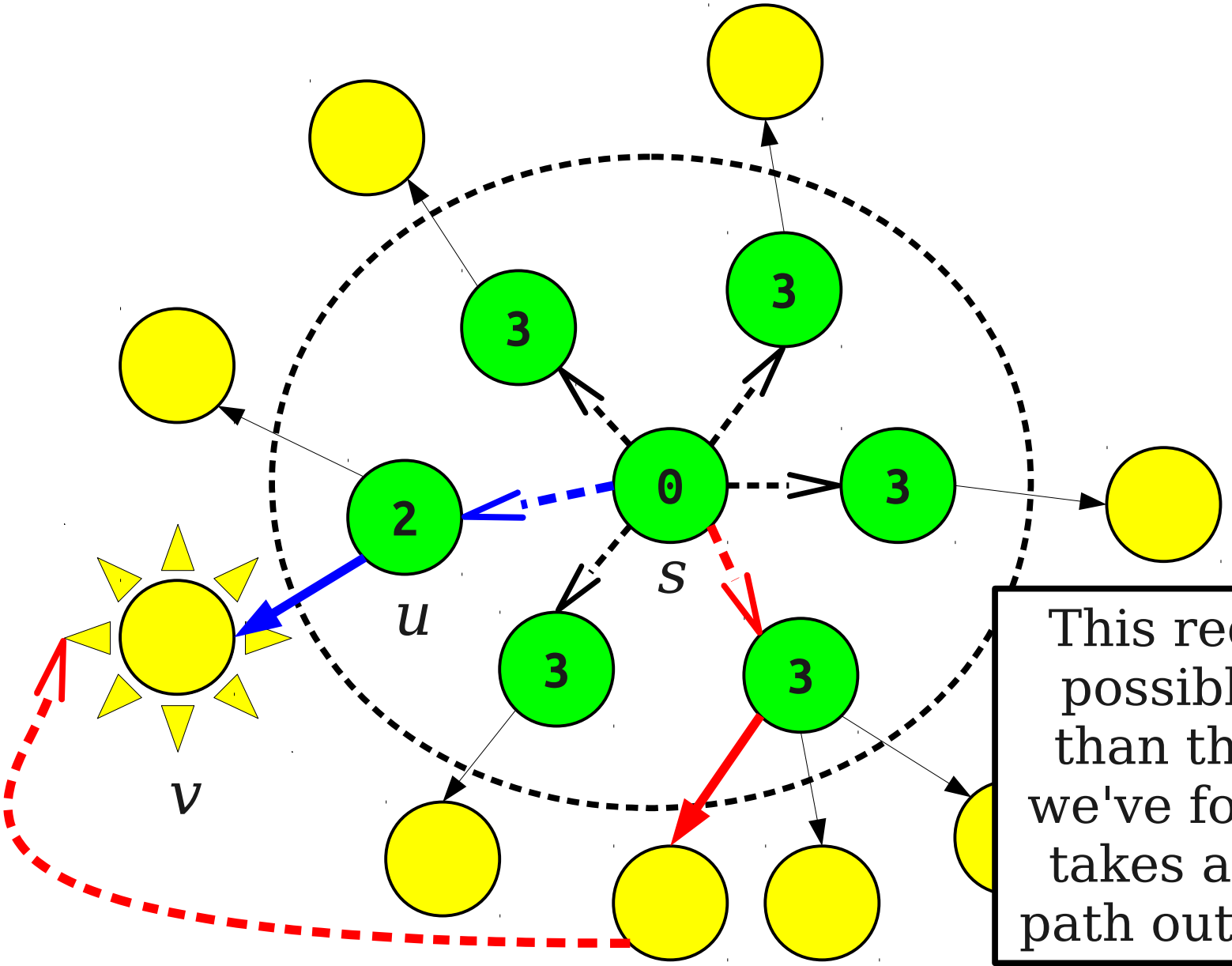
Pick yellow node v minimizing d(s, u) + 1,
where (u, v) is an edge and u is green.



Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.

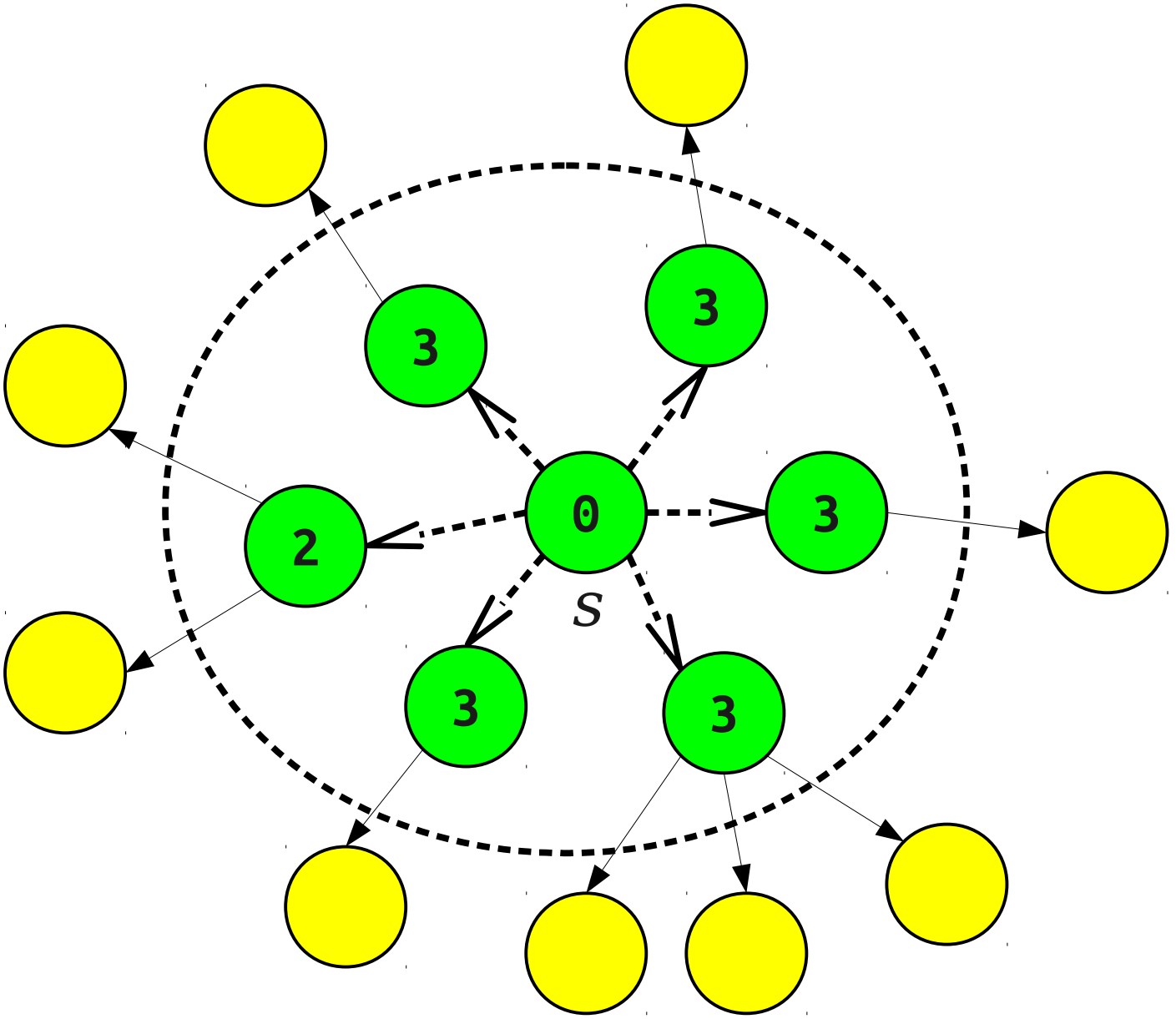


Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.

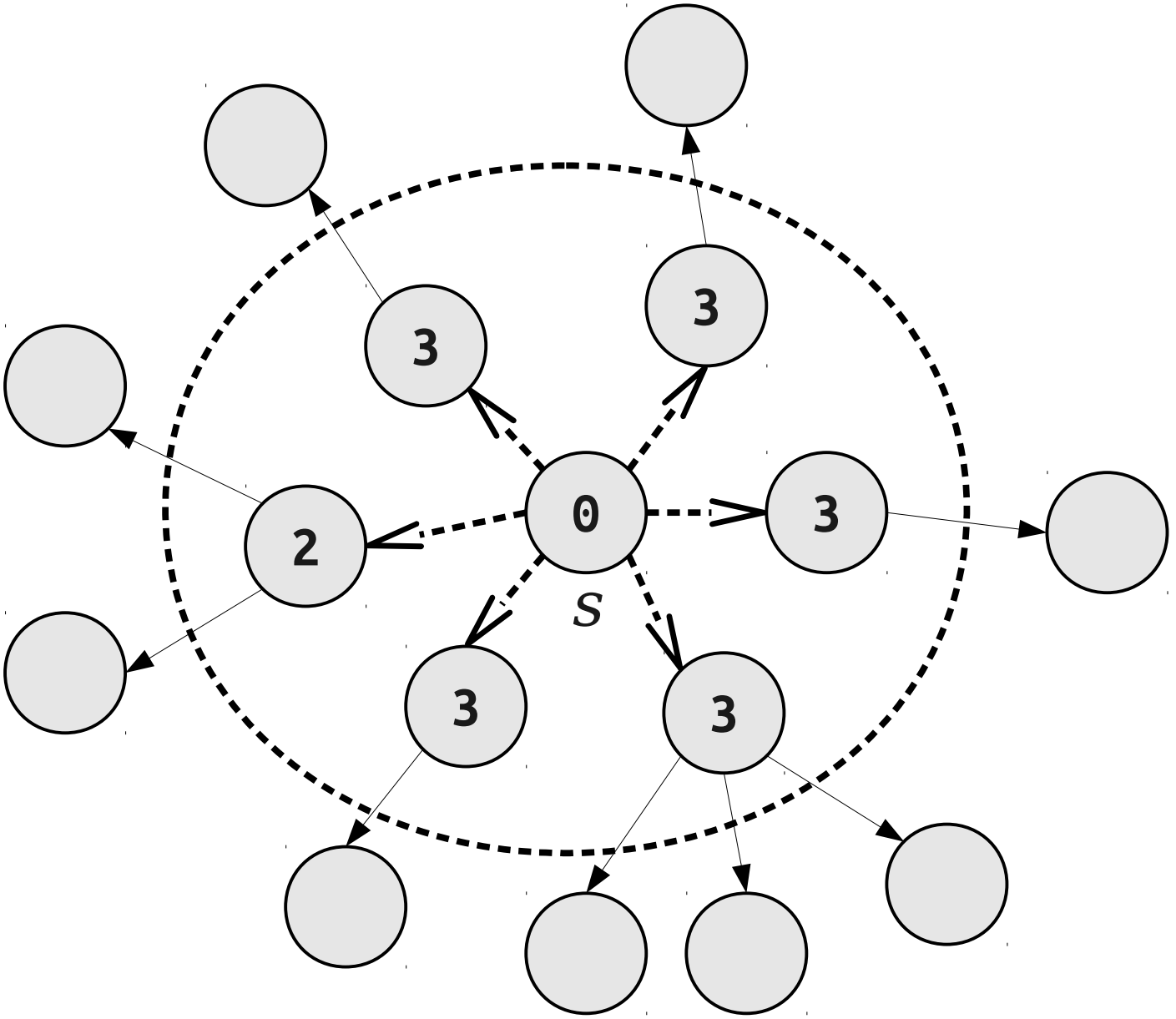


This red path can't possibly be better than the blue path we've found, since it takes a suboptimal path out of the circle!

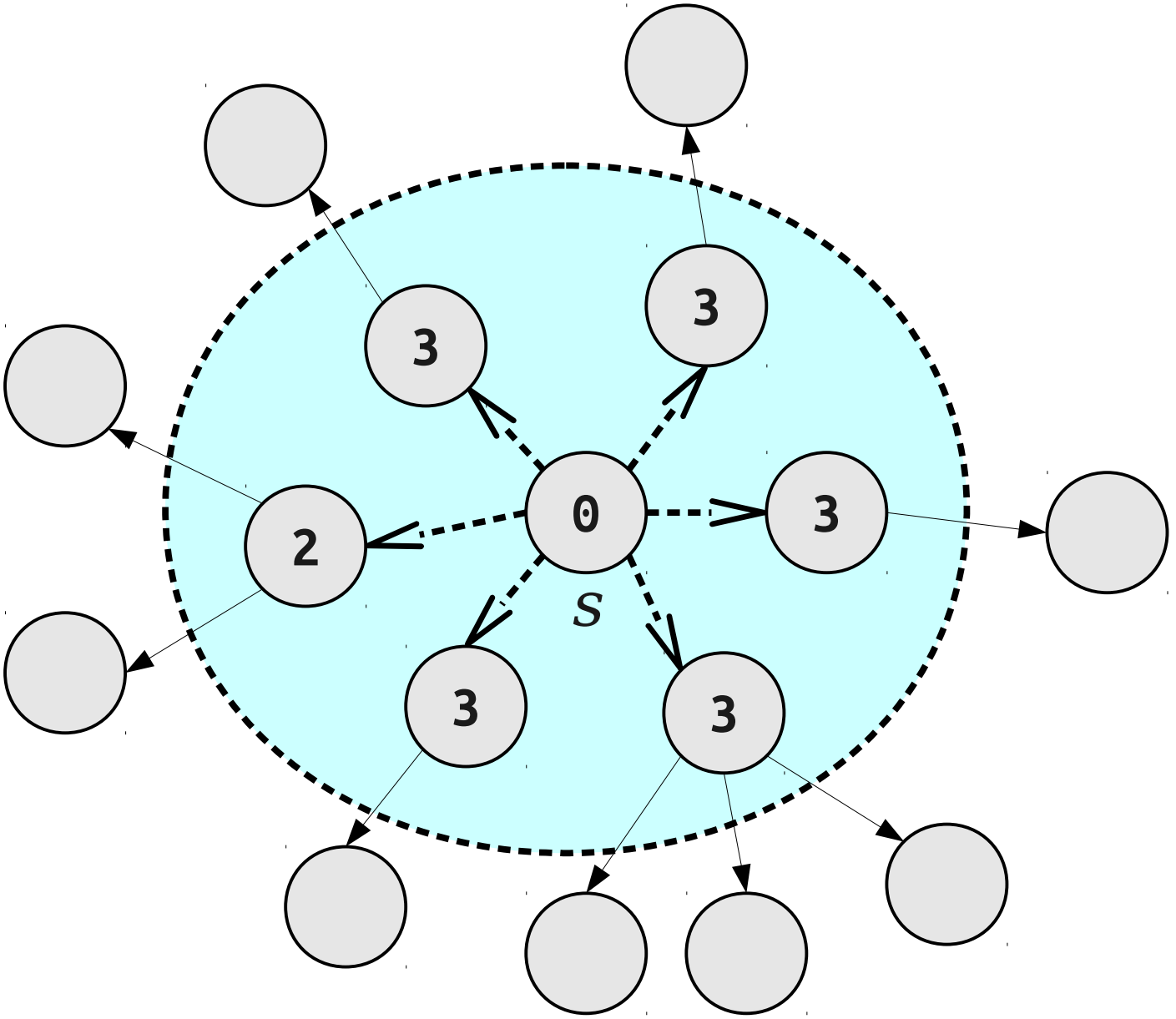
Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.



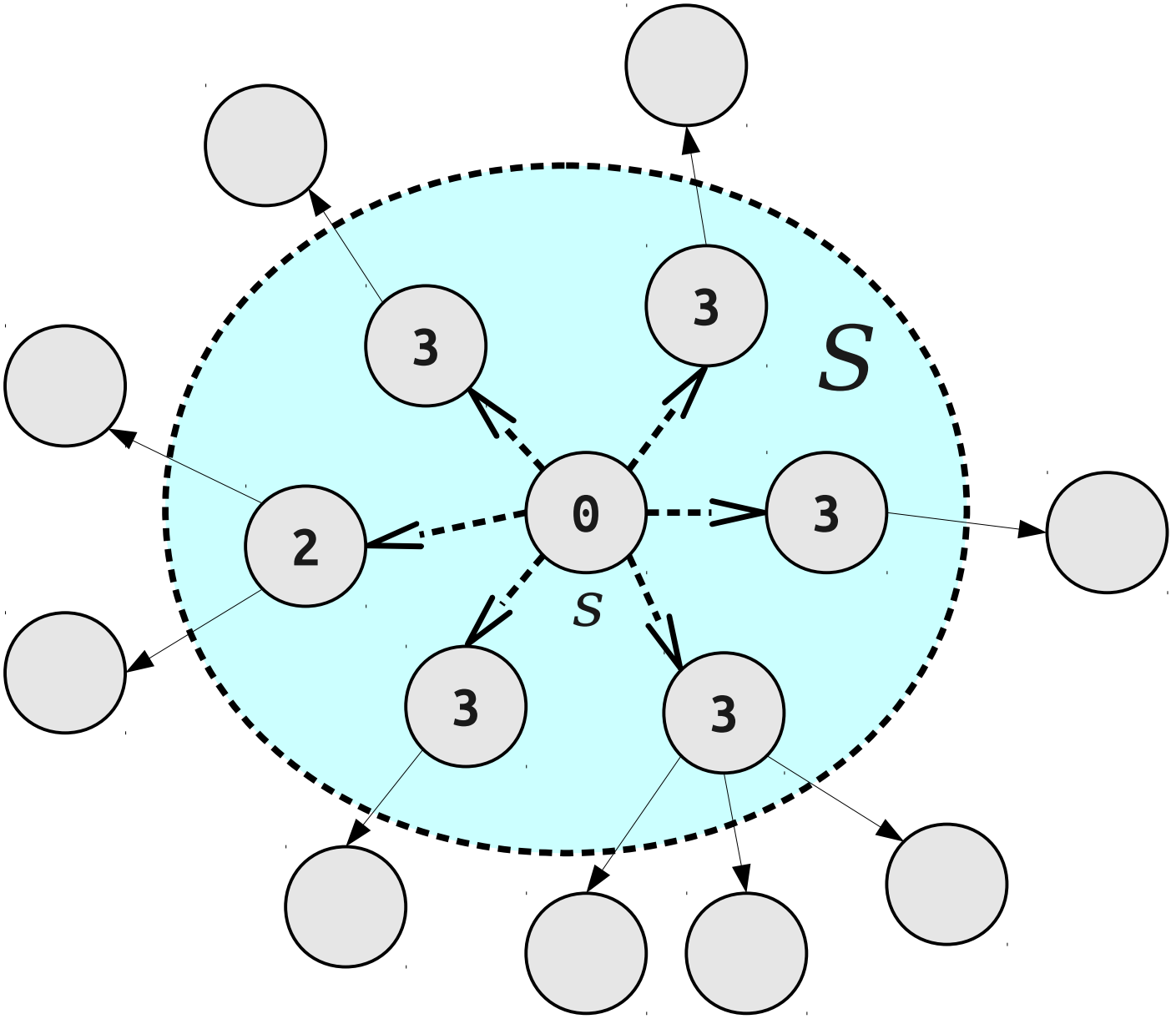
Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.



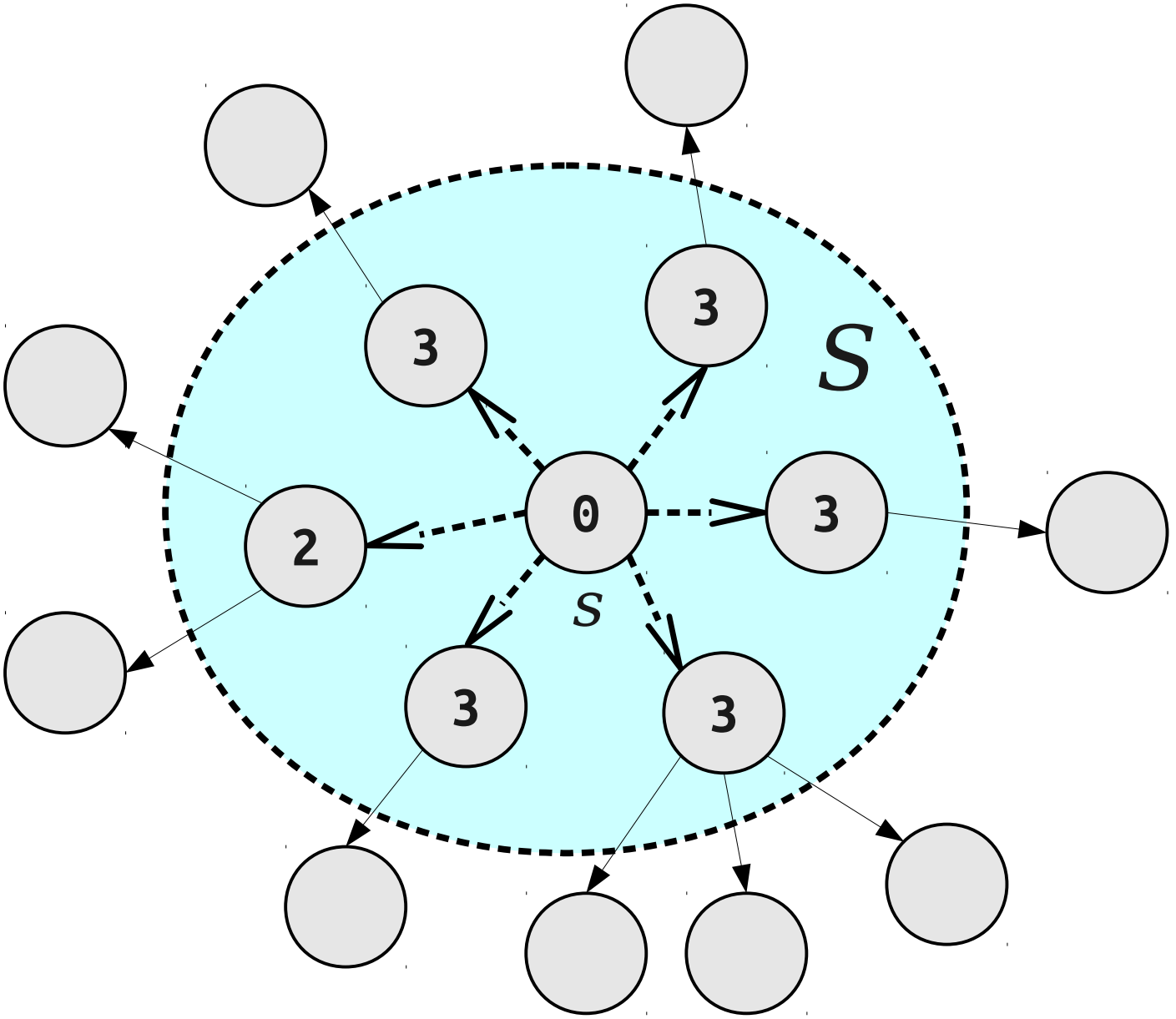
Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.



Pick yellow node v minimizing d(s, u) + 1, where (u, v) is an edge and u is green.



Pick node $v \notin S$ minimizing d(s, u) + 1,
where (u, v) is an edge and $u \in S$



Lemma: Suppose we have shortest paths computed for nodes $S \subseteq V$, where $s \in S$. Consider a node v where $(u, v) \in E$, $u \in S$, and the quantity $\mathrm{d}(s, u) + 1$ is minimized. Then $d(s, v) = d(s, u) + 1$.

Proof: There is a path to v of cost $\mathrm{d}(s, u) + 1$: follow the shortest path to u (which has cost $\mathrm{d}(s, u)$), then follow one more edge to v for total cost $\mathrm{d}(s, u) + 1$.

Now suppose for the sake of contradiction that there is a shorter path P to v. This path must start in S (since $s \in S$) and leave S (since $v \notin S$). So consider when P leaves S. When this happens, P must go from s to some node $x \in S$, cross an edge (x, y) to some node y, then continue from y to v. This means that $|P|$ is at least $\mathrm{d}(s, x) + 1$, since the path goes from s to x and then follows at least one more edge.

Since v was picked to minimize $\mathrm{d}(s, u) + 1$ for any choice of $u \in S$ adjacent to an edge (u, v), we know

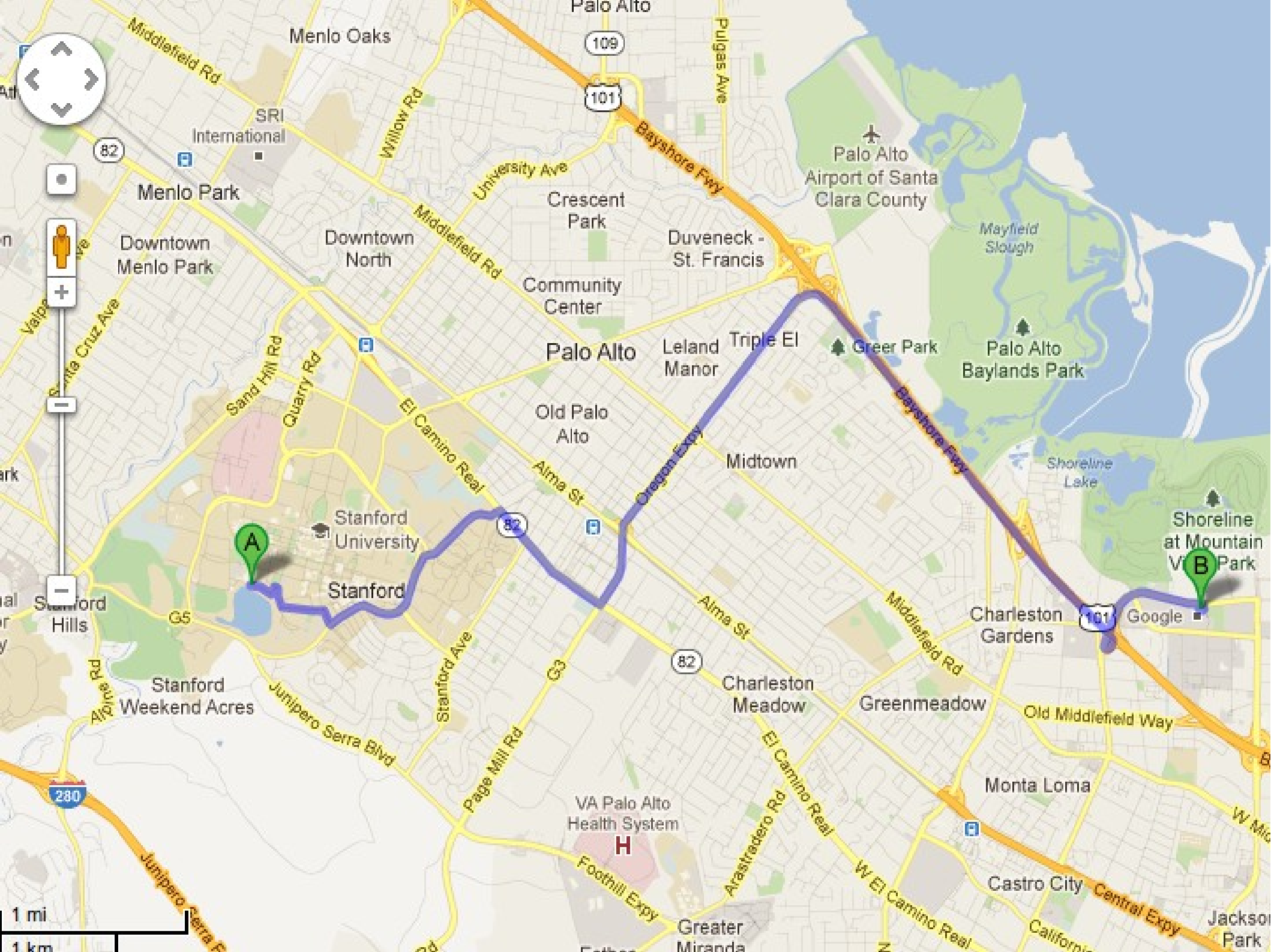
$$\mathrm{d}(s, u) + 1 \leq d(s, x) + 1 \leq |P|$$

contradicting the fact that $|P| < \mathrm{d}(s, u) + 1$. We have reached a contradiction, so our assumption was wrong and no shorter path exists.

Since there is a path of length $\mathrm{d}(s, u) + 1$ from s to v and no shorter path, this means that $\mathrm{d}(s, v) = \mathrm{d}(s, u) + 1$. ■

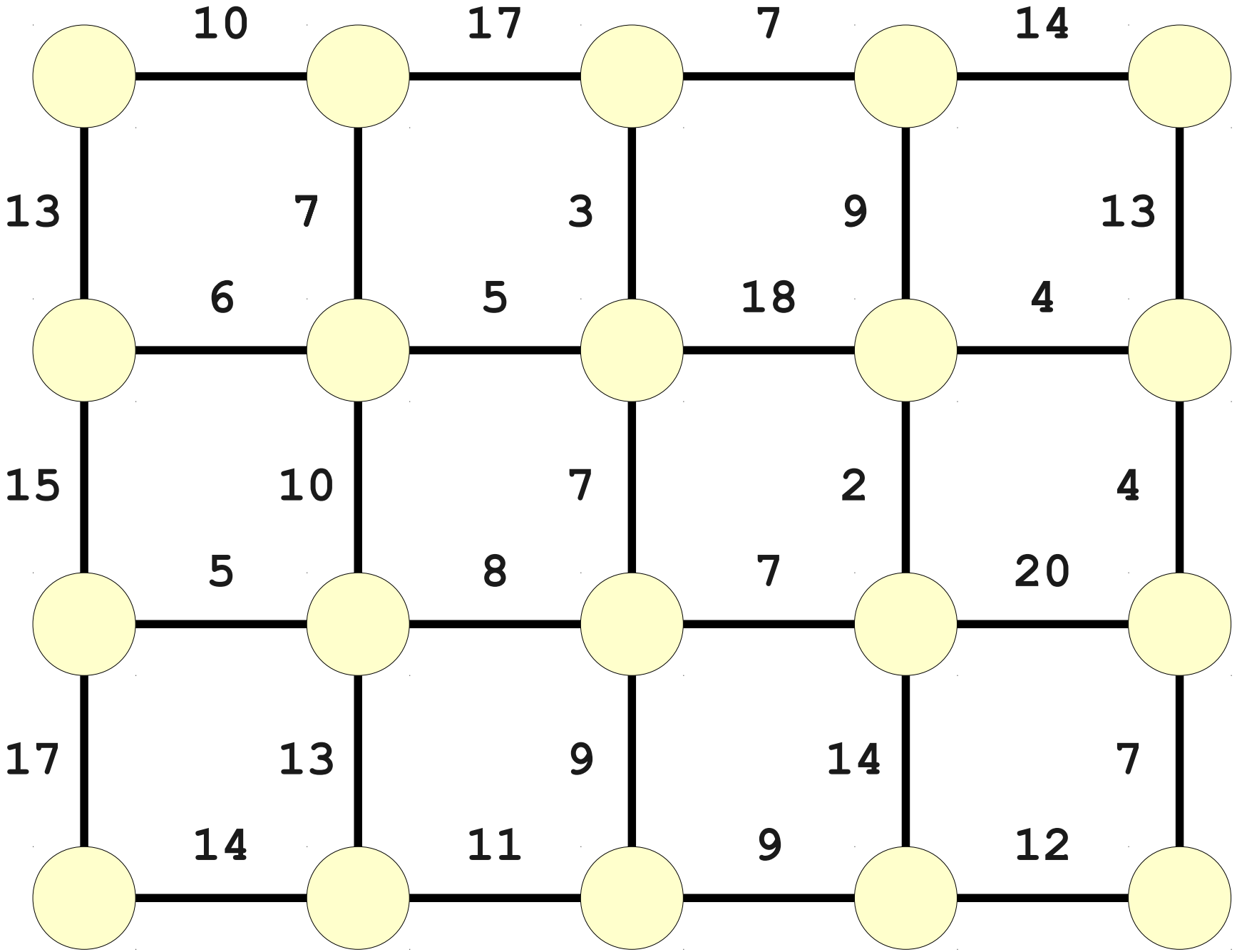
Why These Two Proofs Matter

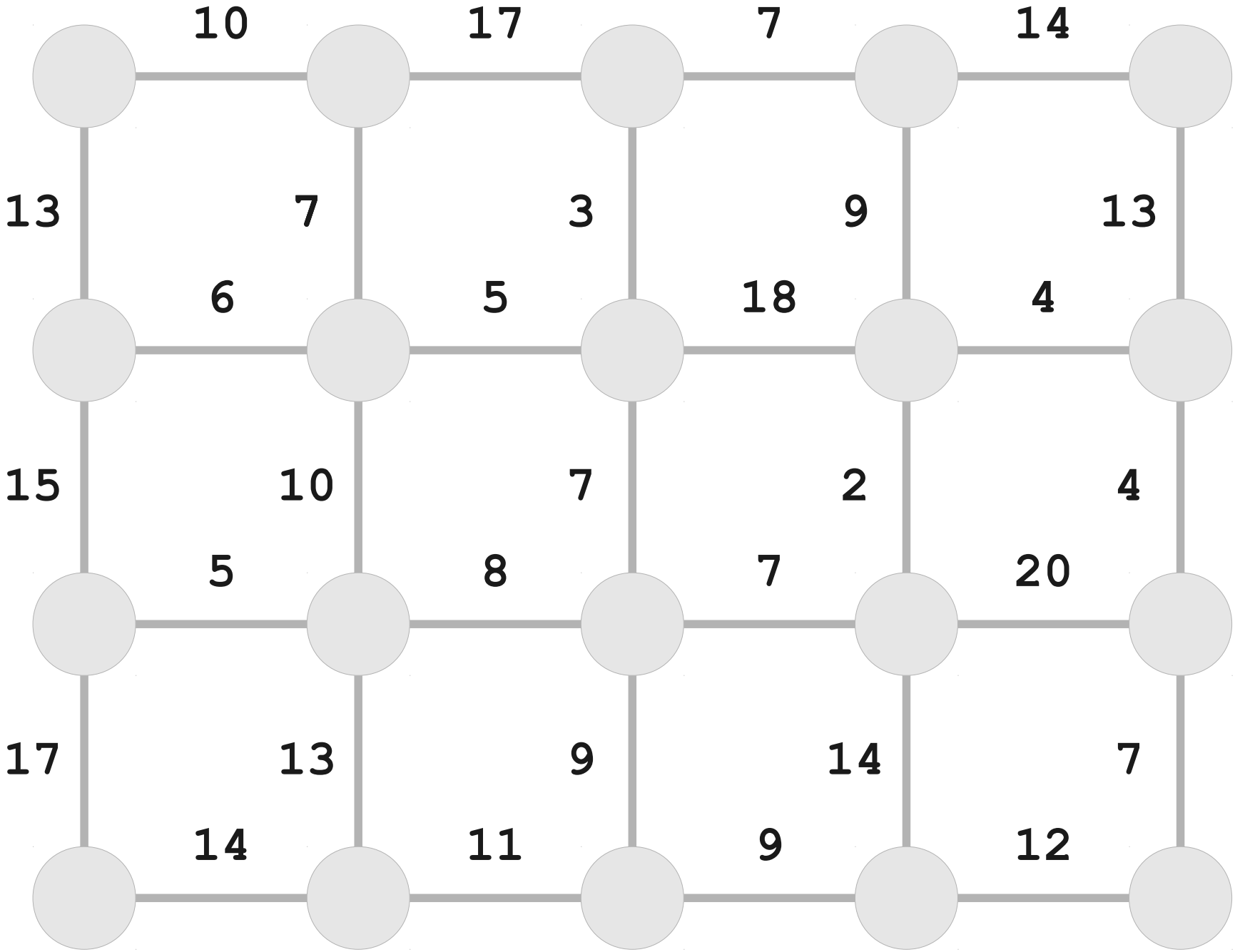
- The first proof of correctness (based on layers) is based on our first observation: the nodes visited in BFS radiate outward from the start node in ascending order of distance.
- The second proof of correctness (based on picking the lowest yellow node) is based on our second observation: picking the lowest-cost yellow node correctly computes a shortest path.
- Interestingly, this second correctness proof can be generalized to a larger setting...

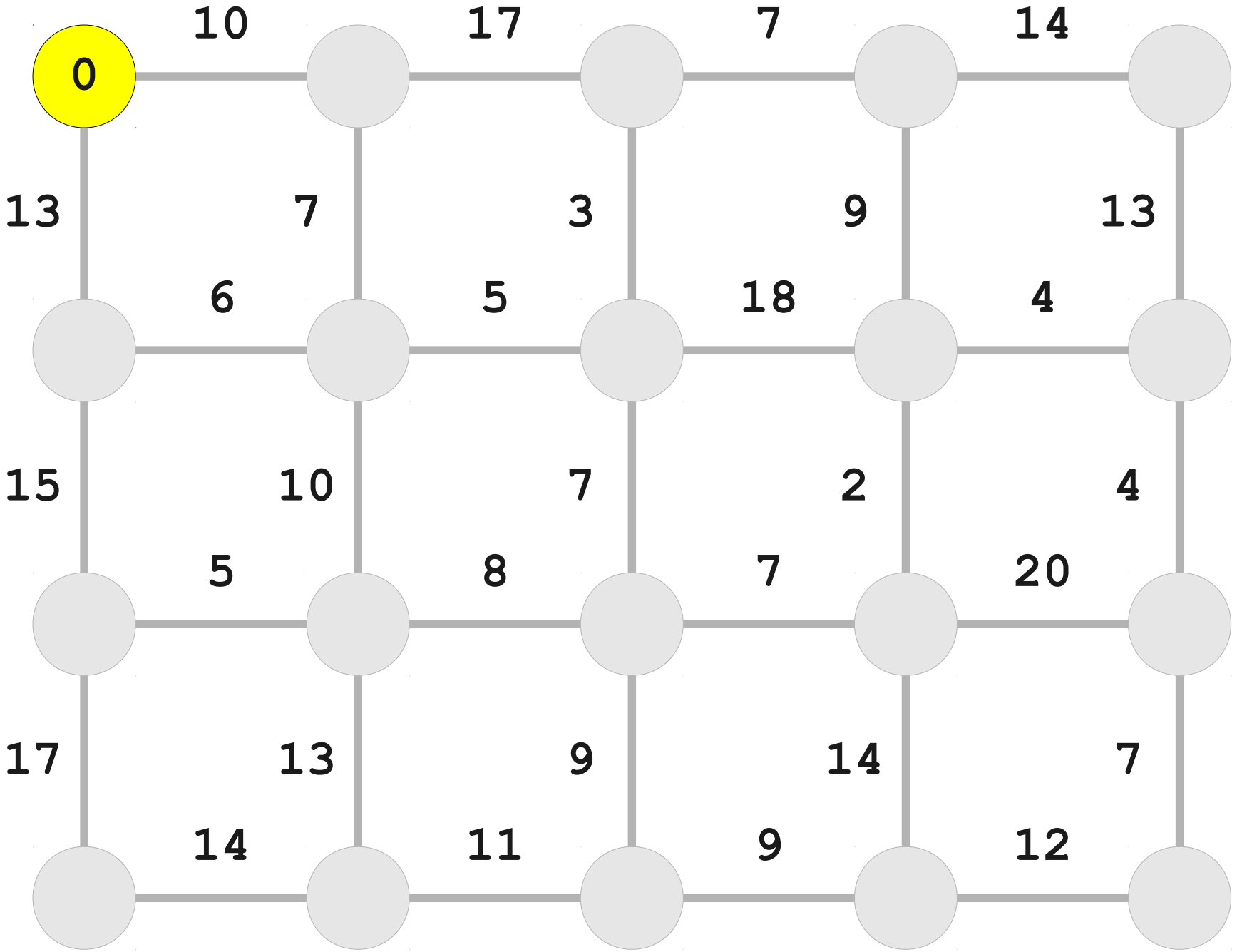

Palo Alto
Menlo Oaks
Middlefield Rd
109
101
Bayshore Fwy
Pulgas Ave
Willow Rd
SRI International
82
Menlo Park
University Ave
Crescent Park
Palo Alto Airport of Santa Clara County
Mayfield Slough
Downtown Menlo Park
Downtown North
Middlefield Rd
Duveneck - St. Francis
Community Center
Palo Alto
Leland Manor
Triple El
Greer Park
Palo Alto Baylands Park
Santa Cruz Ave
Sand Hill Rd
Quarry Rd
El Camino Real
Old Palo Alto
Alma St
Oregon Expy
Midtown
Bayshore Fwy
Shoreline Lake
Stanford University
A
B
Shoreline at Mountain View Park
Stanford
Stanford Hills
G5
Alma St
Middlefield Rd
Charleston Gardens
101
Google
Alpine Rd
Stanford Weekend Acres
Junipero Serra Blvd
Stanford Ave
G3
82
Charleston Meadow
Greenmeadow
Old Middlefield Way
Page Mill Rd
El Camino Real
280
Monta Loma
VA Palo Alto Health System
H
Arastradero Rd
W El Camino Real
Foothill Expy
Castro City
Central Expy
Jackson Park
Greater Miranda
1 mi
1 km

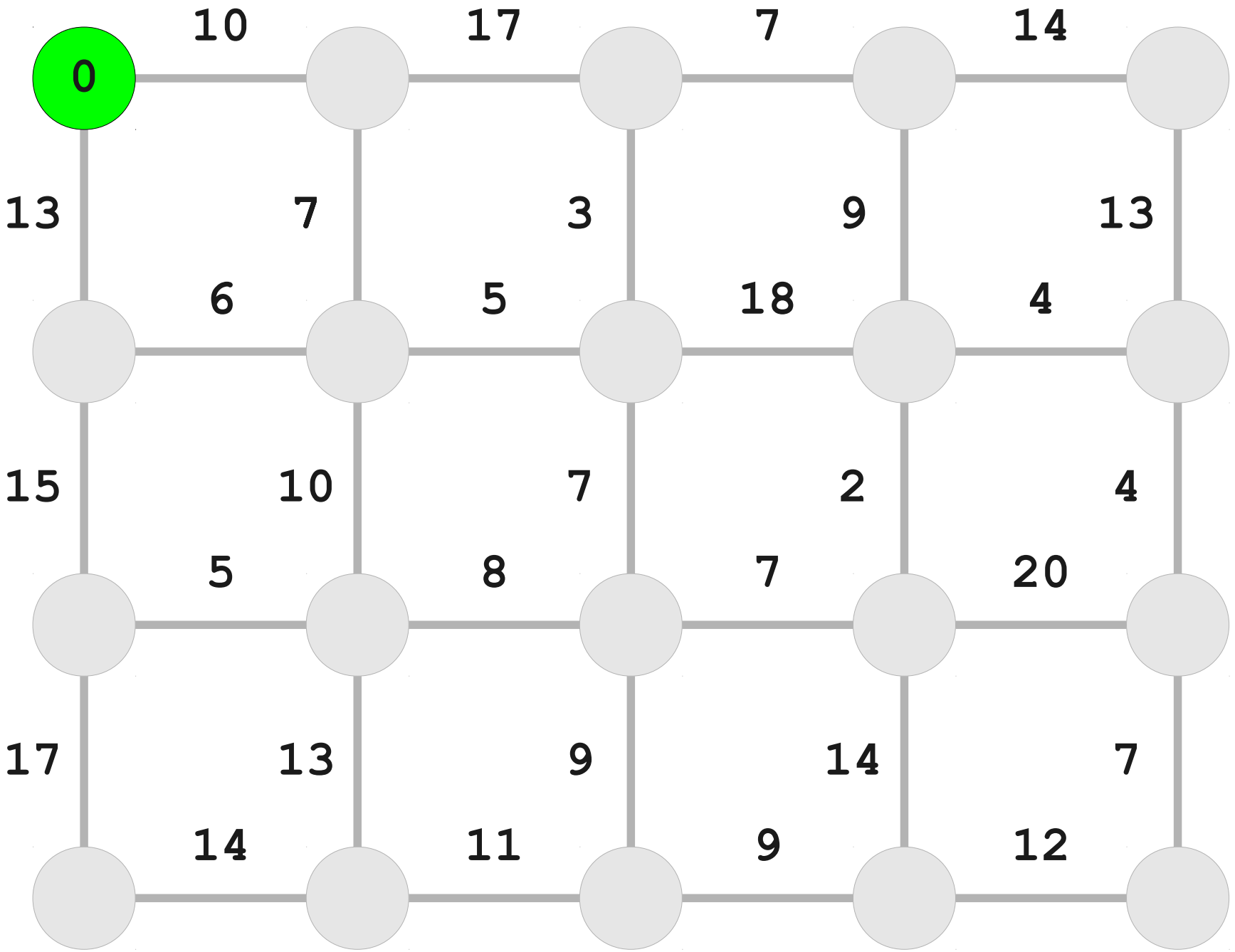
Edges with Costs

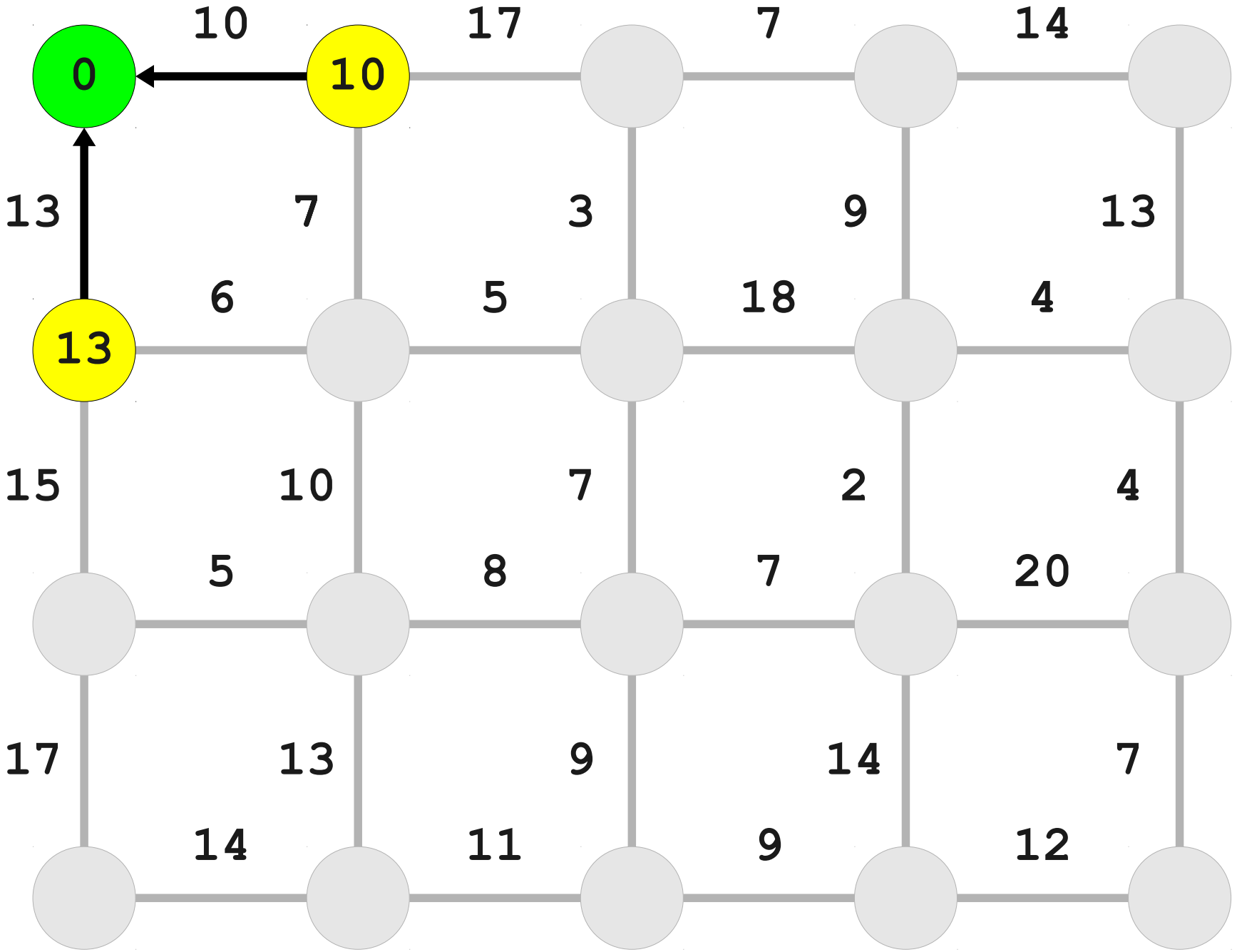
- In many applications, edges have an associated length (or cost, weight, etc.), denoted ***l*(*u*, *v*)**.
- **Assumption**: Lengths are nonnegative. (We'll revisit this later in the quarter.)
- Let's say that the length of a path P (denoted ***l*(*P*)**) is the sum of all the edge lengths in the path P.
- Goal: find the shortest path from s to every node in V, taking costs into account.

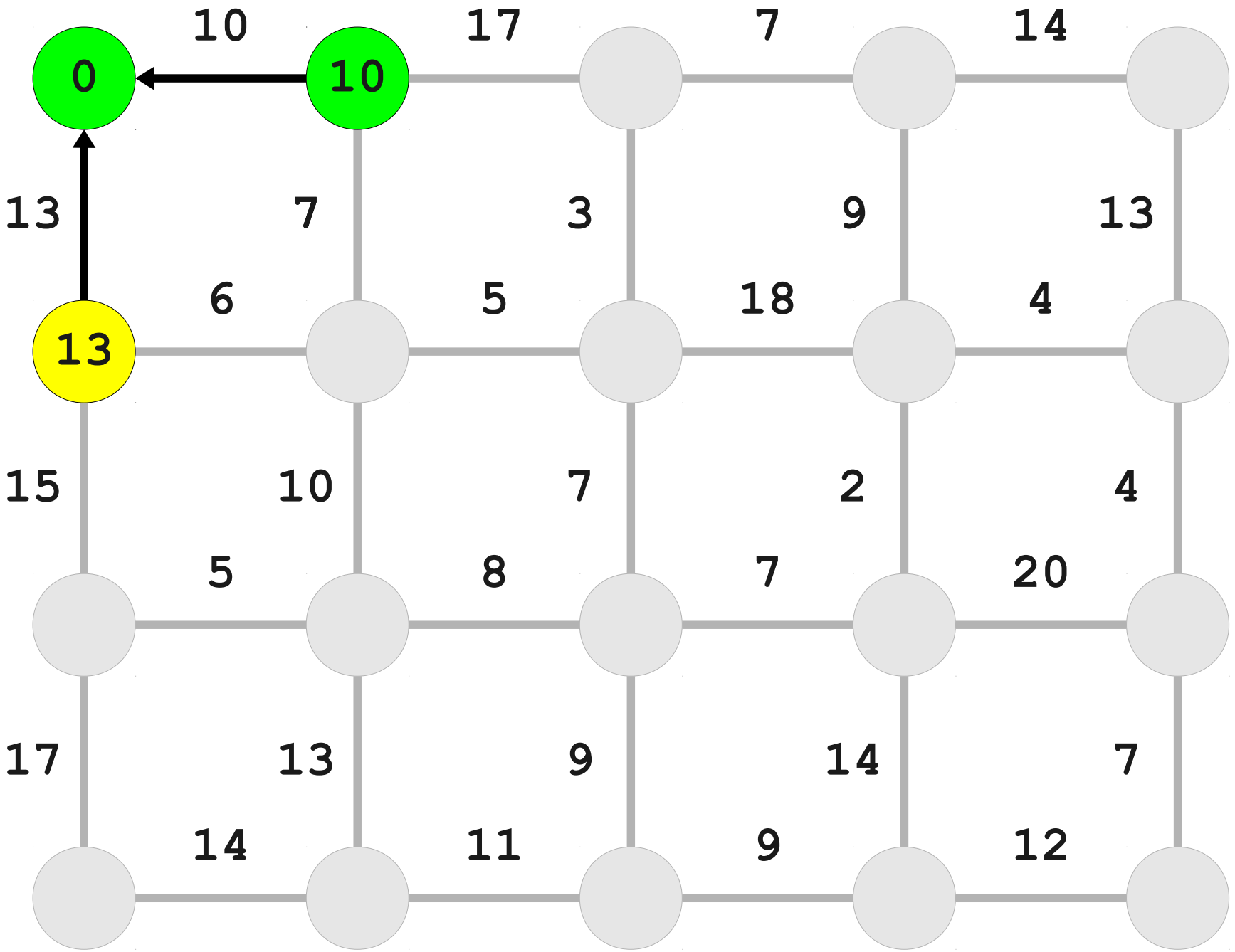

10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

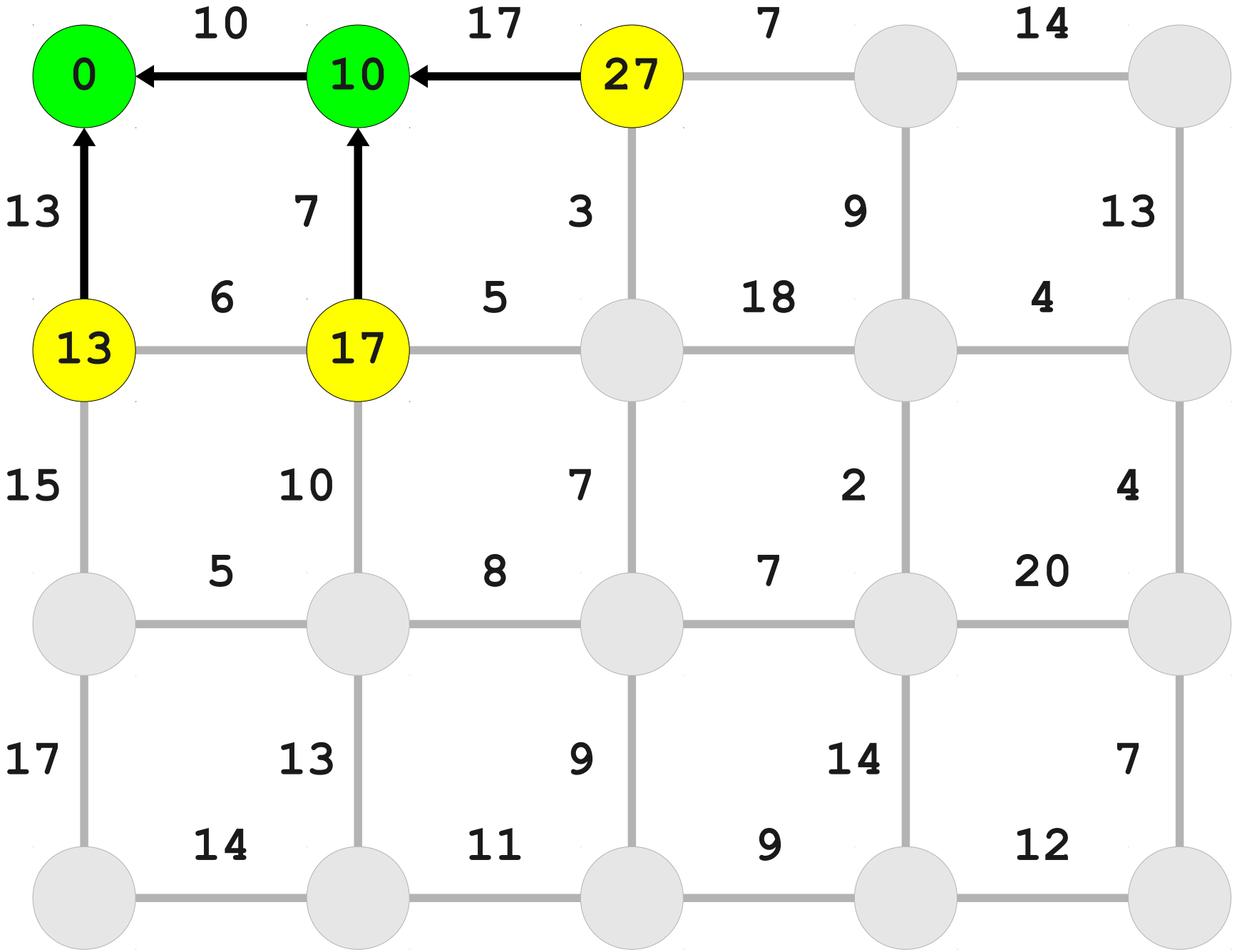

10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

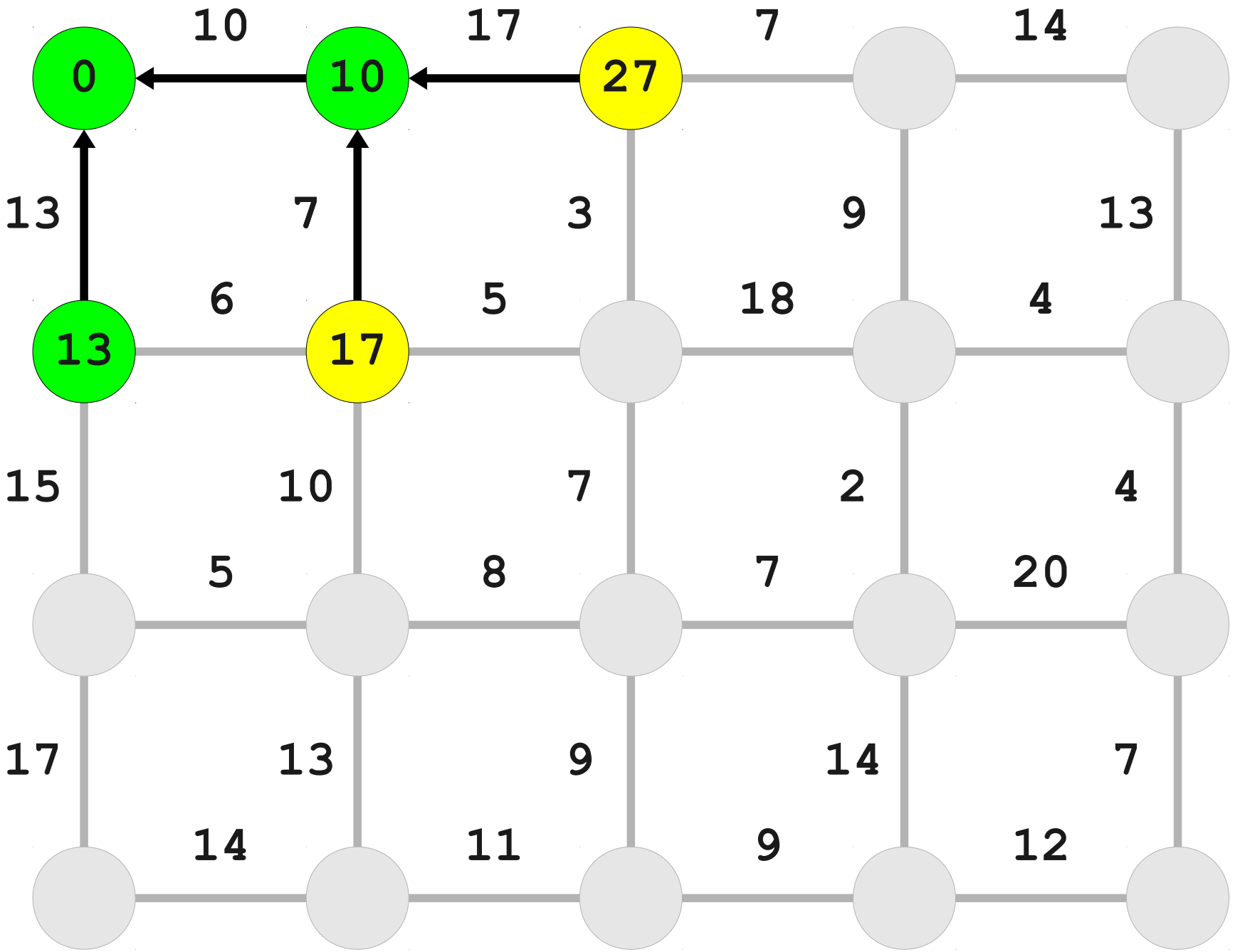

0
10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

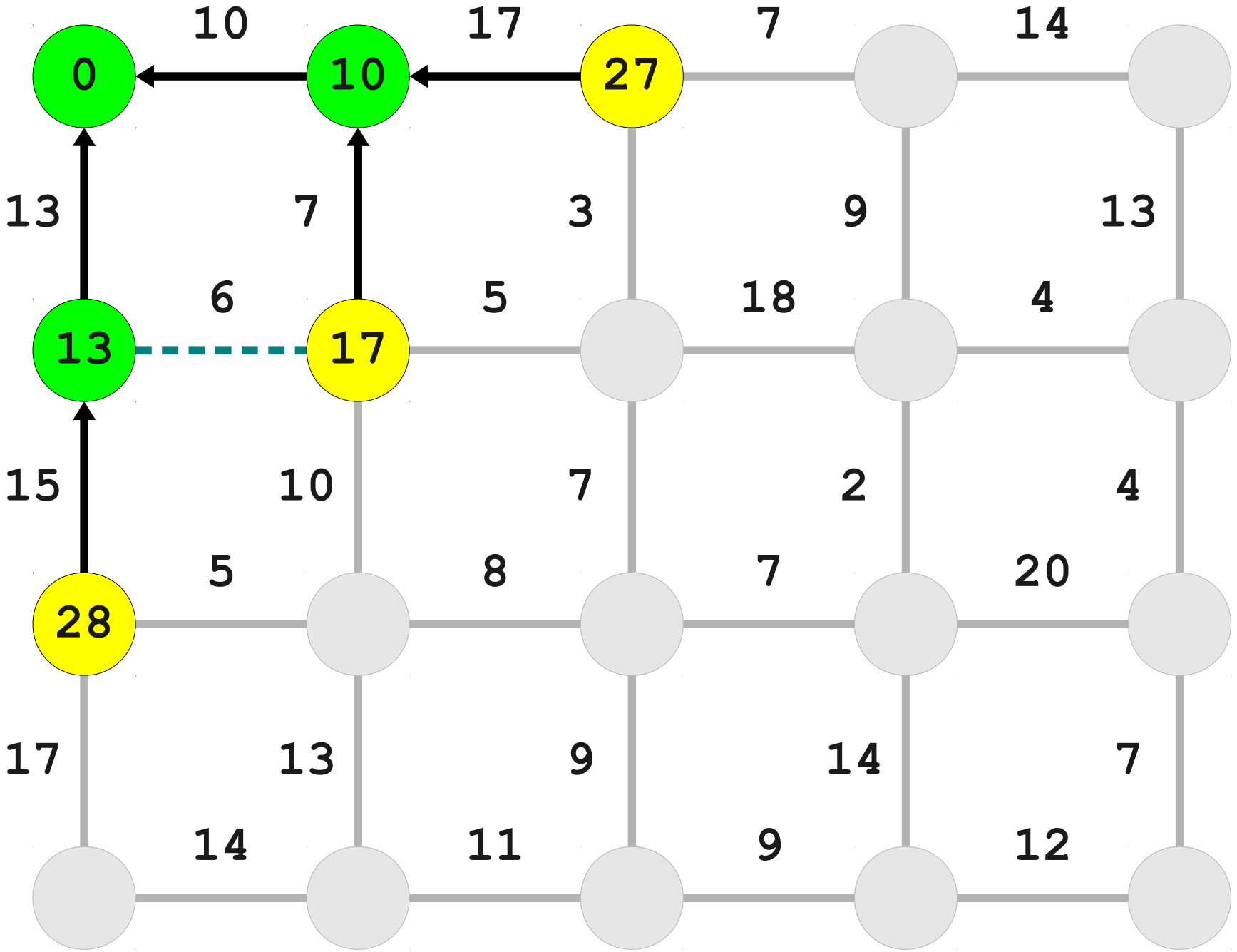

0
10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

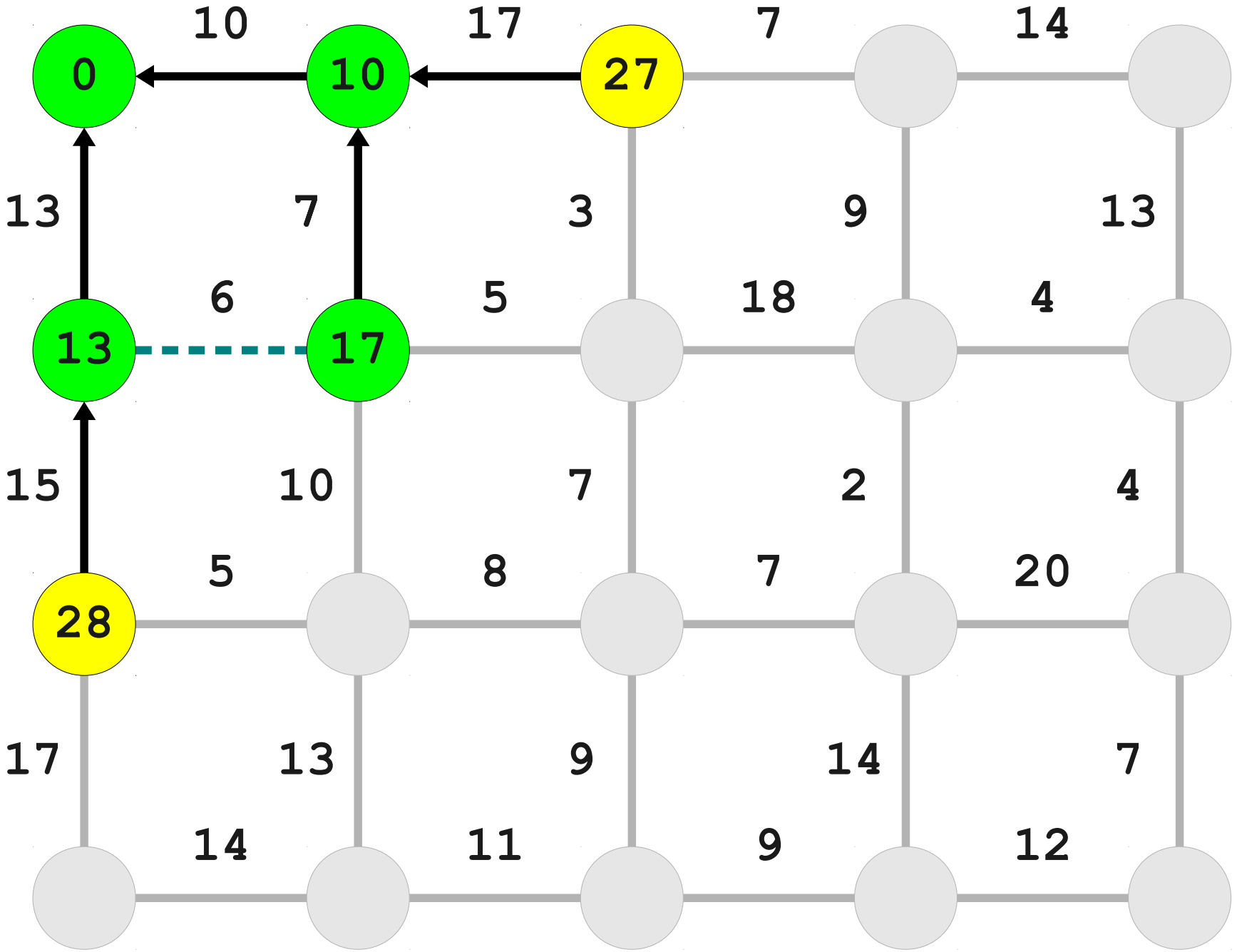

0
10
10
17
7
14
13
7
3
9
13
6
5
18
4
13
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

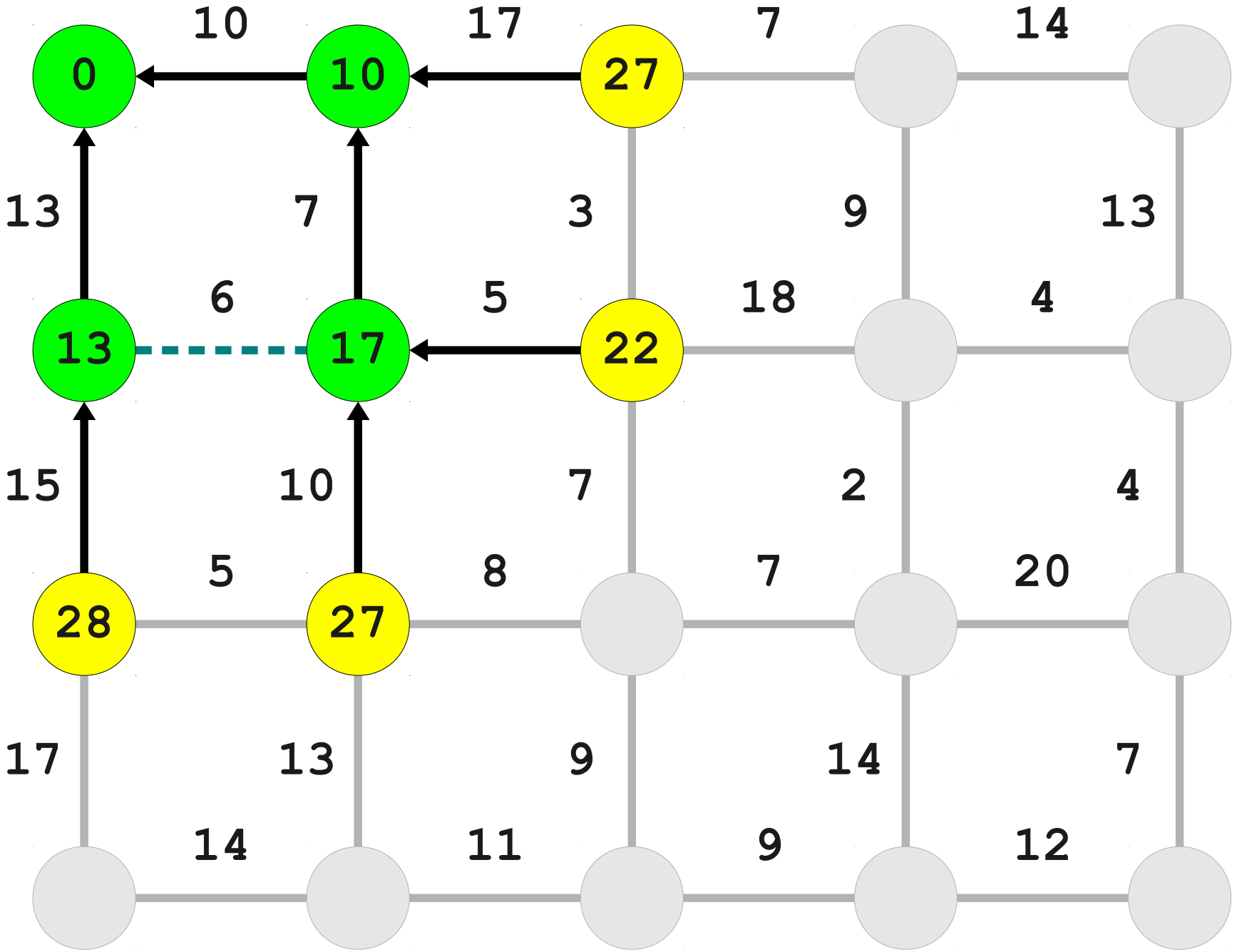

10
17
7
14
0
10
13
7
3
9
13
6
5
18
4
13
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

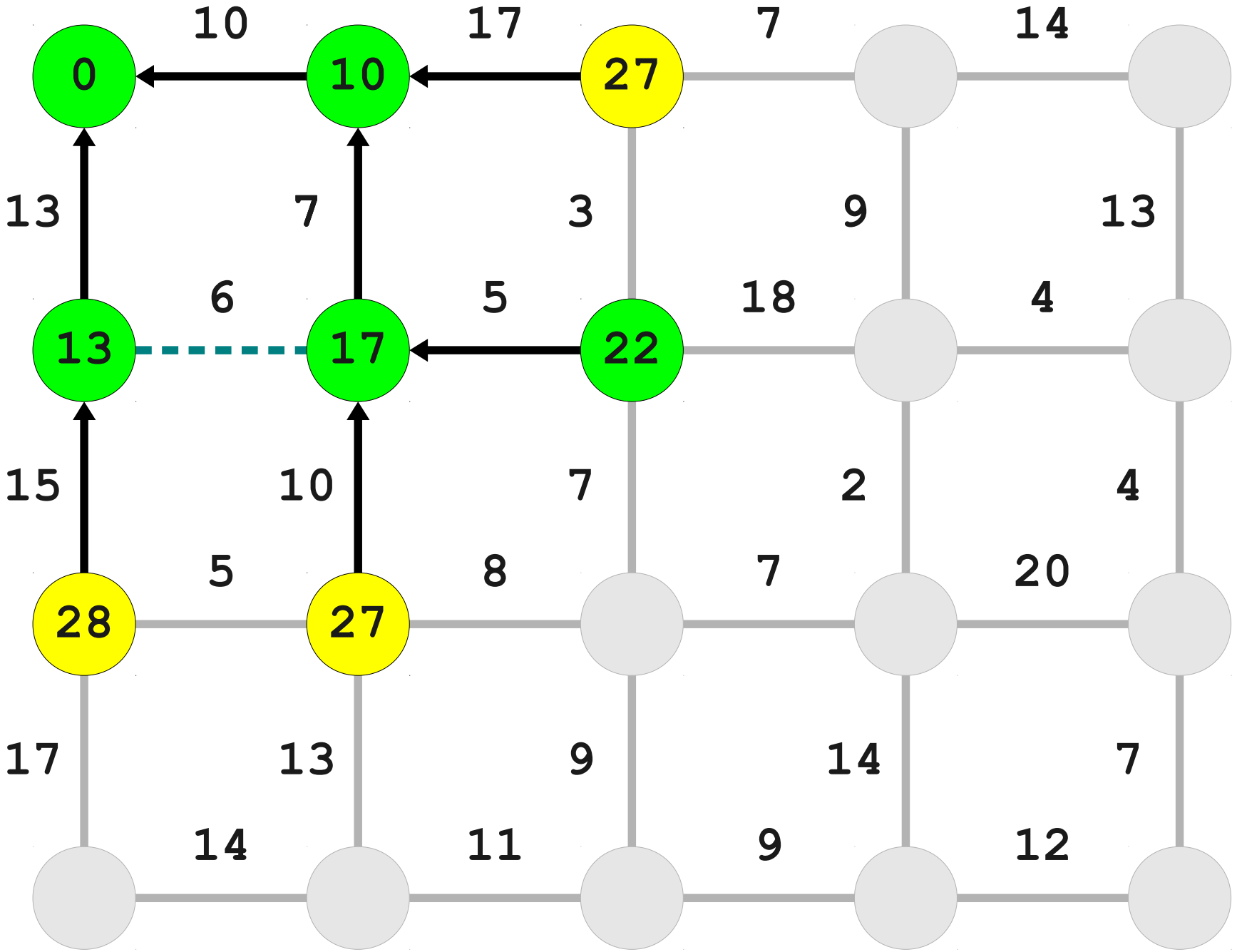

10
17
7
14
0
10
27
13
7
3
9
13
6
5
18
4
13
17
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

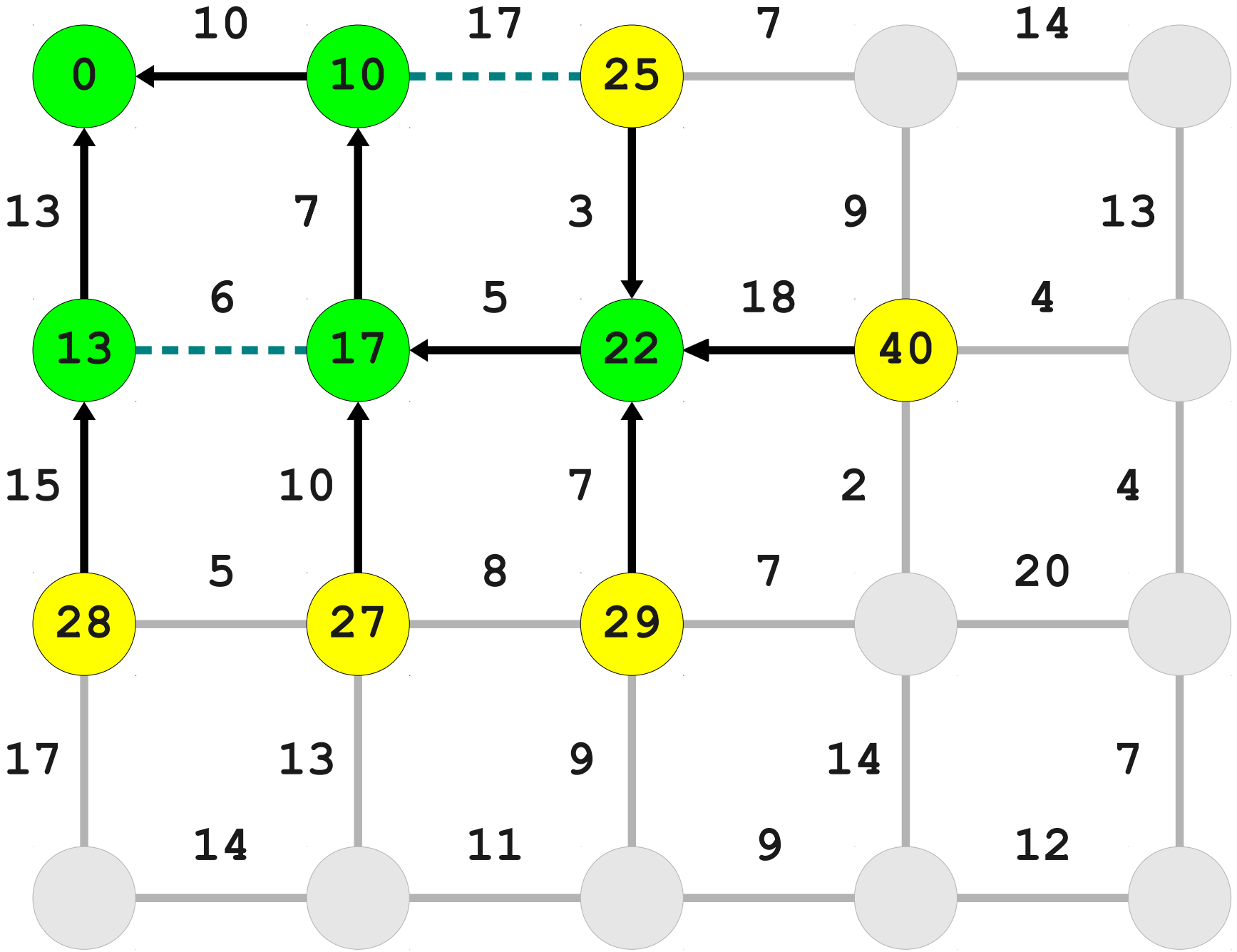

10
17
7
14
0
10
27
13
7
3
9
13
6
5
18
4
13
17
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

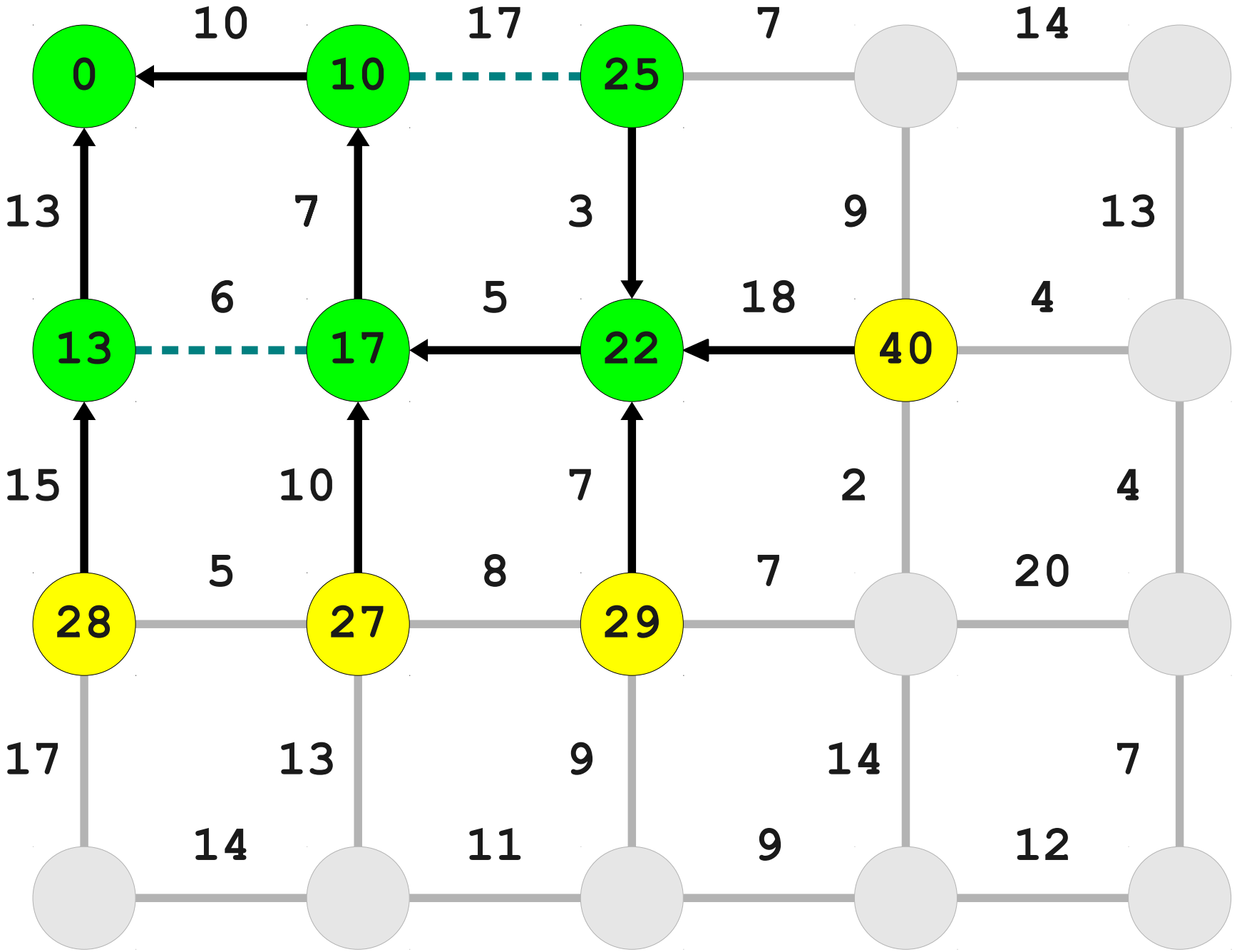

10
17
7
14
0
10
27
13
7
3
9
13
6
5
18
4
13
17
15
10
7
2
4
5
8
7
20
28
17
13
9
14
7
14
11
9
12

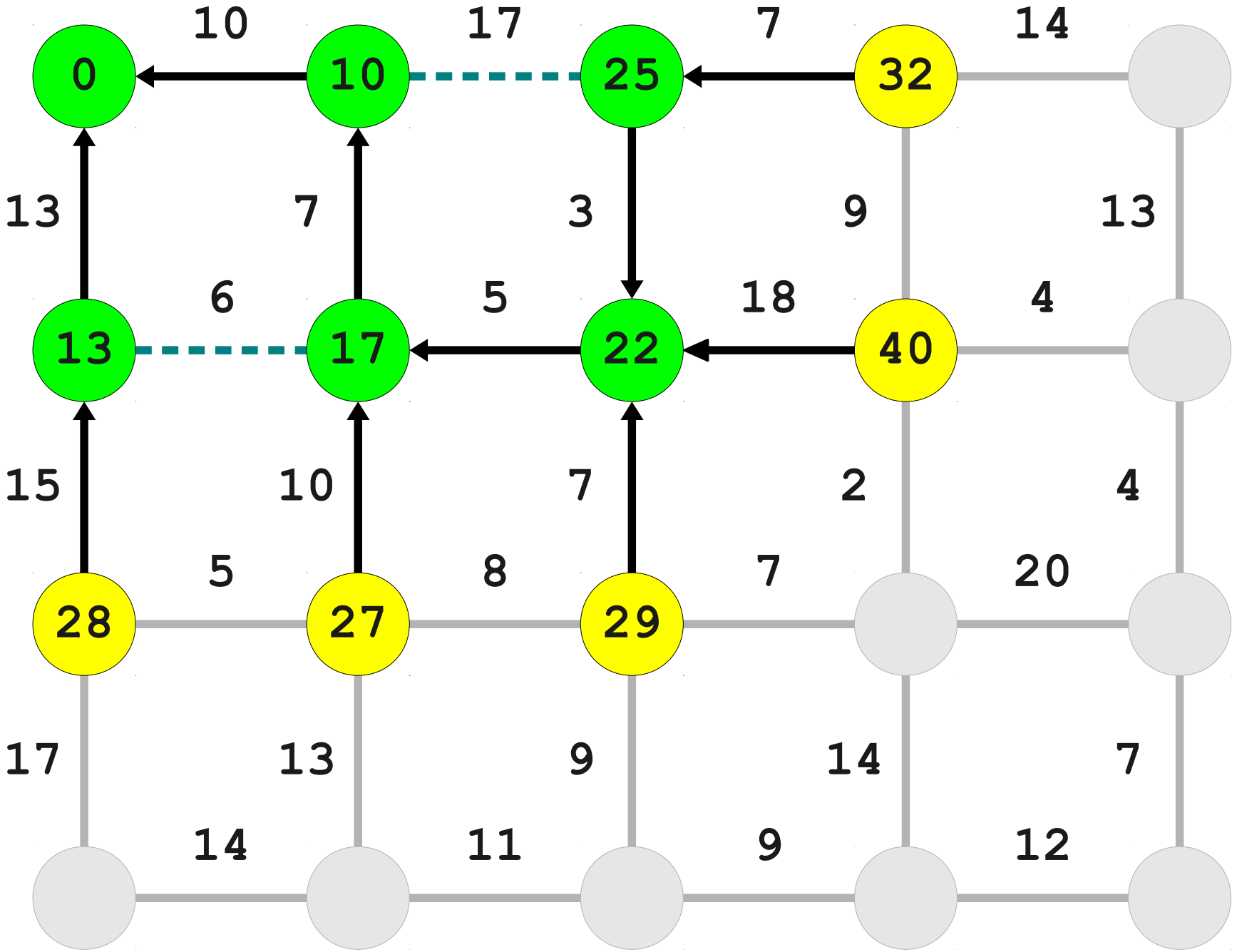

10
17
7
14
0
10
27
13
7
3
9
13
6
5
18
4
13
17
15
10
7
2
4
5
8
7
20
28
17
13
9
14
7
14
11
9
12

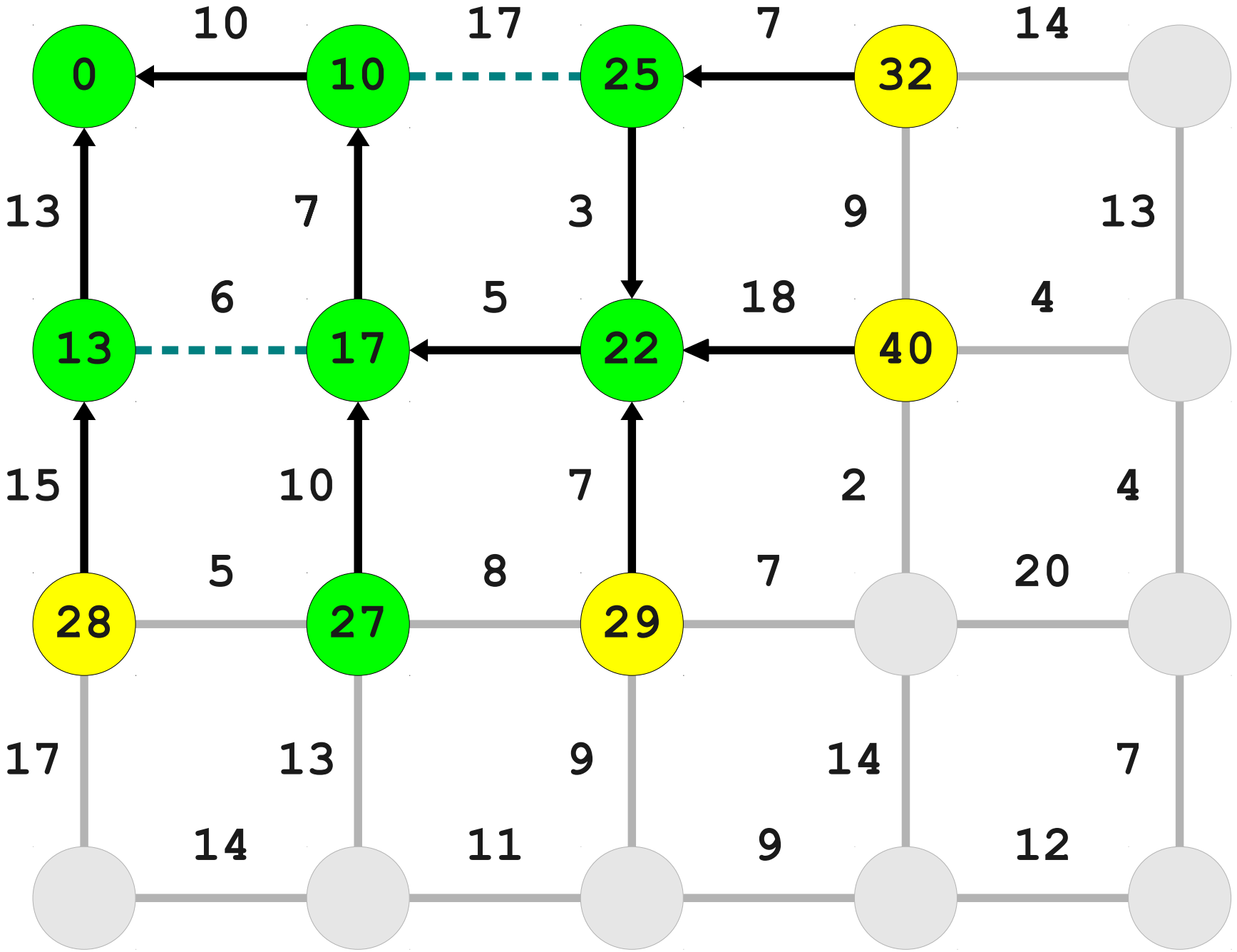

10
17
7
14
0
10
27
13
7
3
9
13
6
5
18
4
13
17
22
15
10
7
2
4
5
8
7
20
28
27
17
13
9
14
7
14
11
9
12

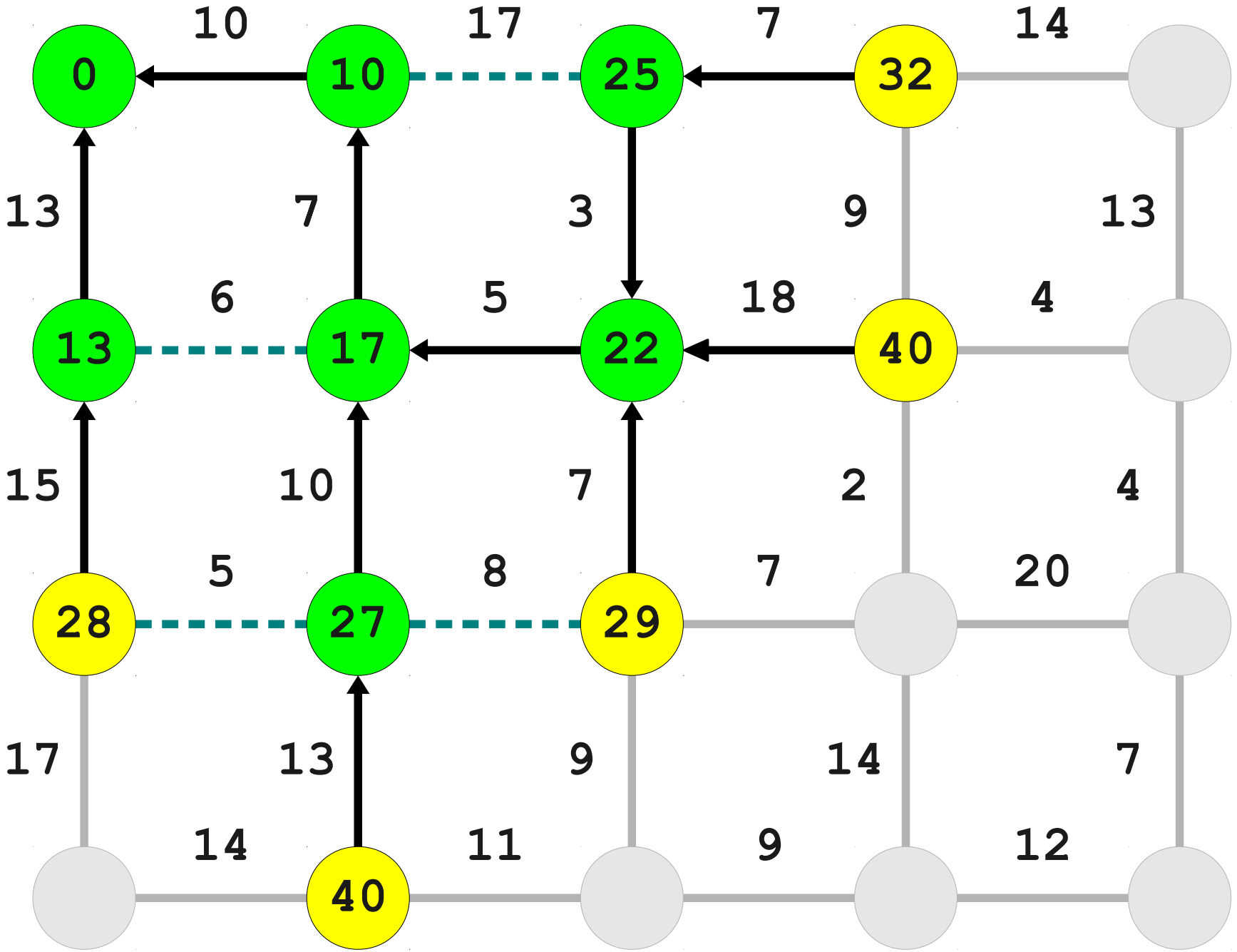

10
17
7
14
0
10
27
13
7
3
9
13
6
5
18
4
13
17
22
15
10
7
2
4
5
8
7
20
28
27
17
13
9
14
7
14
11
9
12

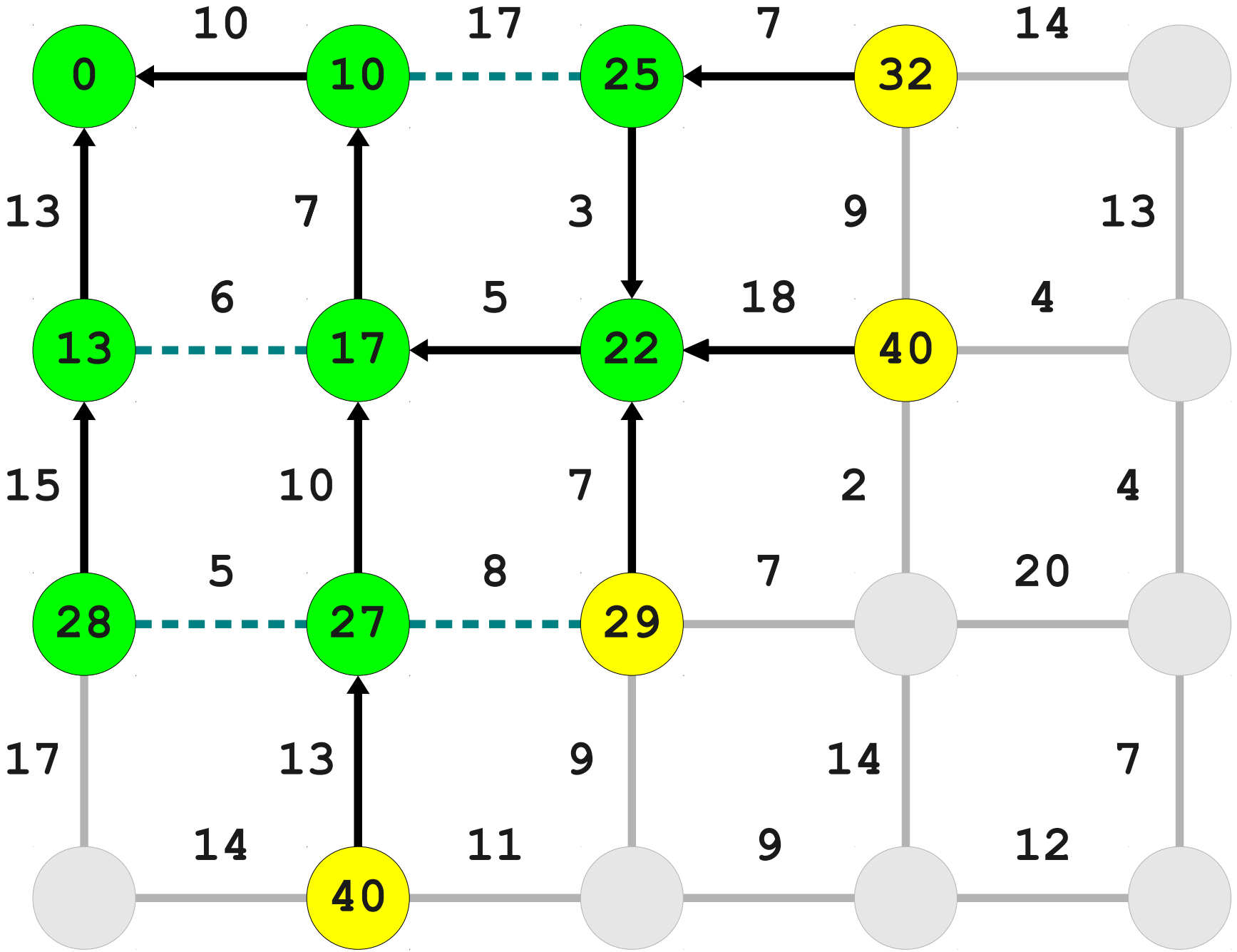

10
17
7
14
0
10
25
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
17
13
9
14
7
14
11
9
12

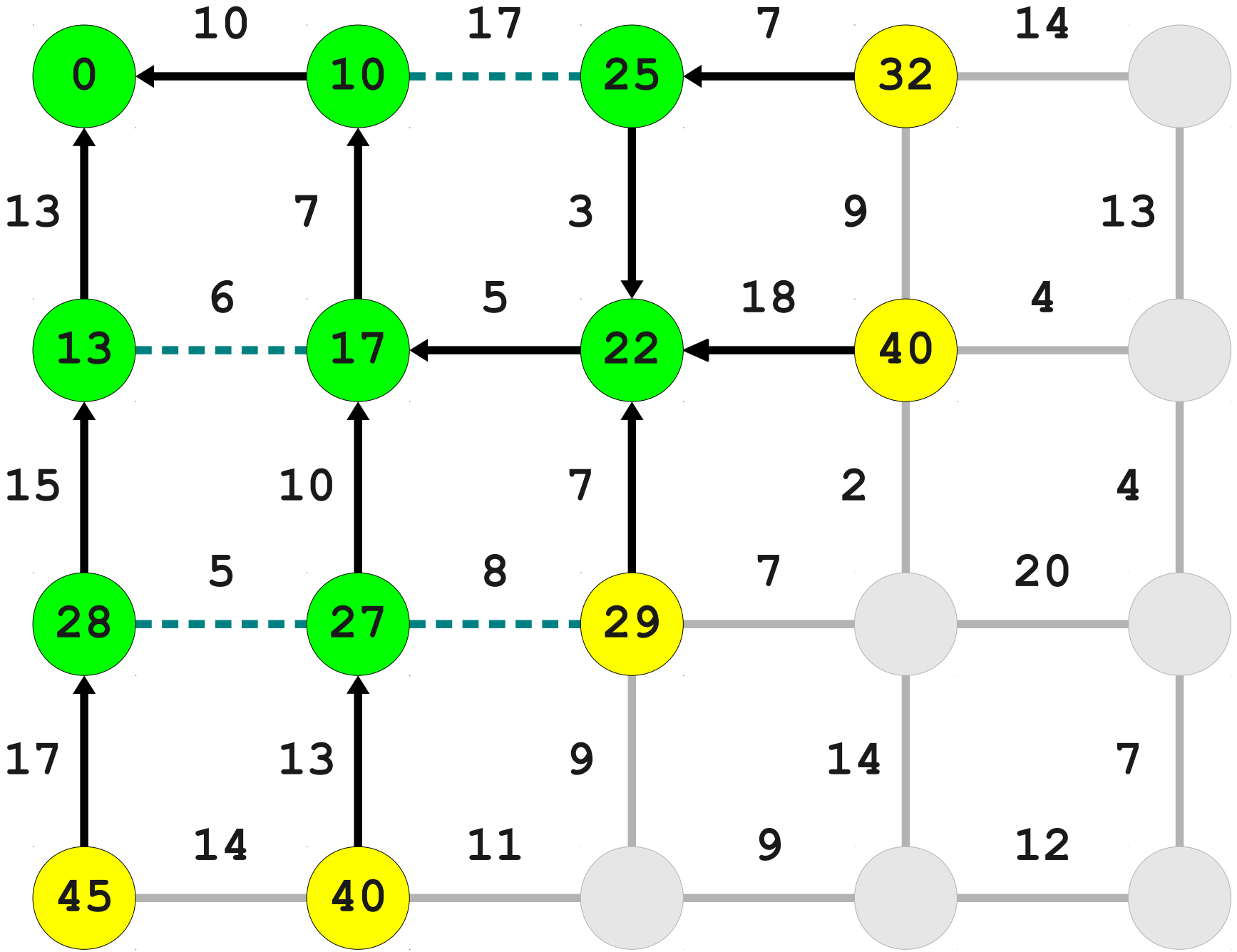

10
17
7
14
0
10
25
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
17
13
9
14
7
14
11
9
12

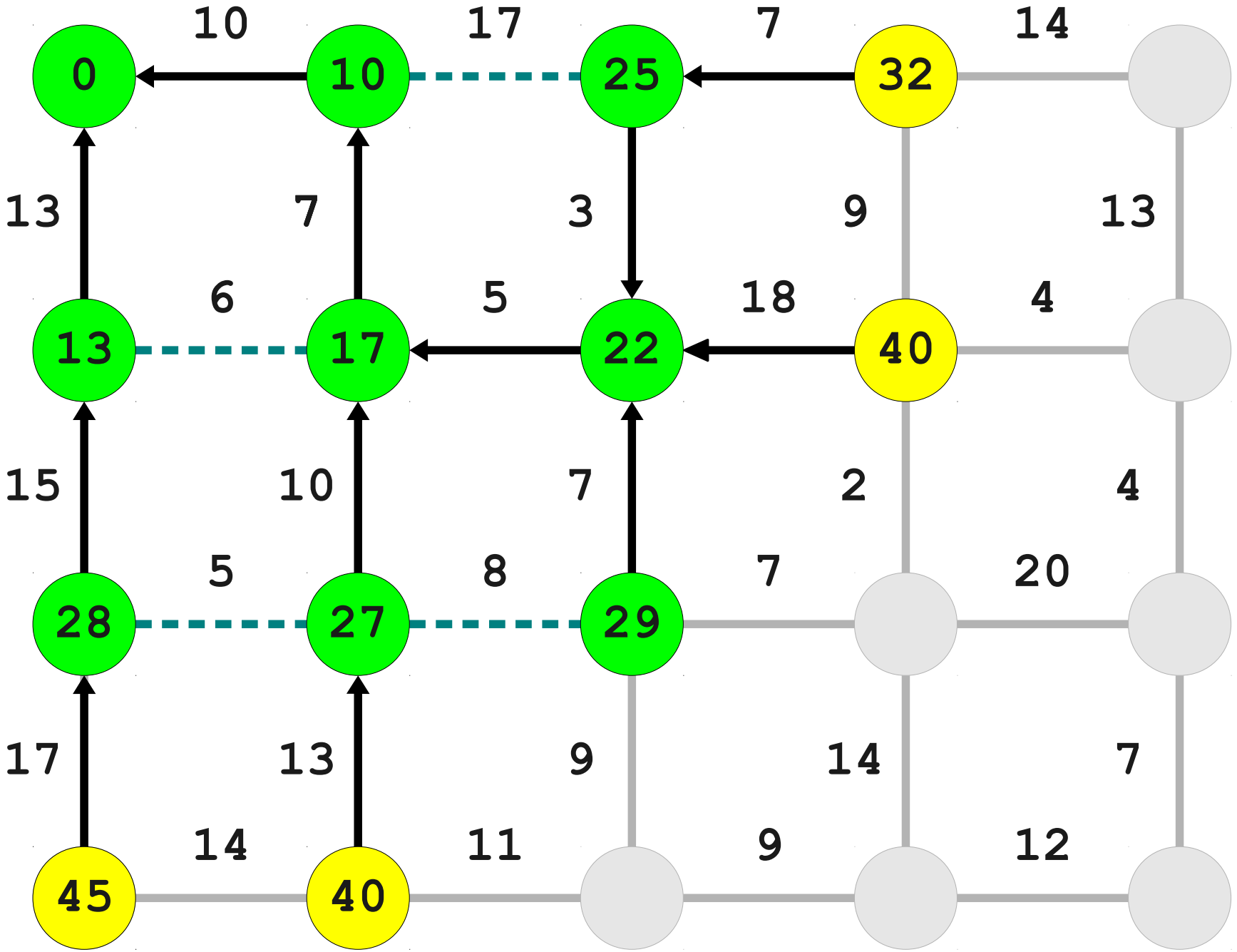

10
17
7
14
0
10
25
32
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
17
13
9
14
7
14
11
9
12

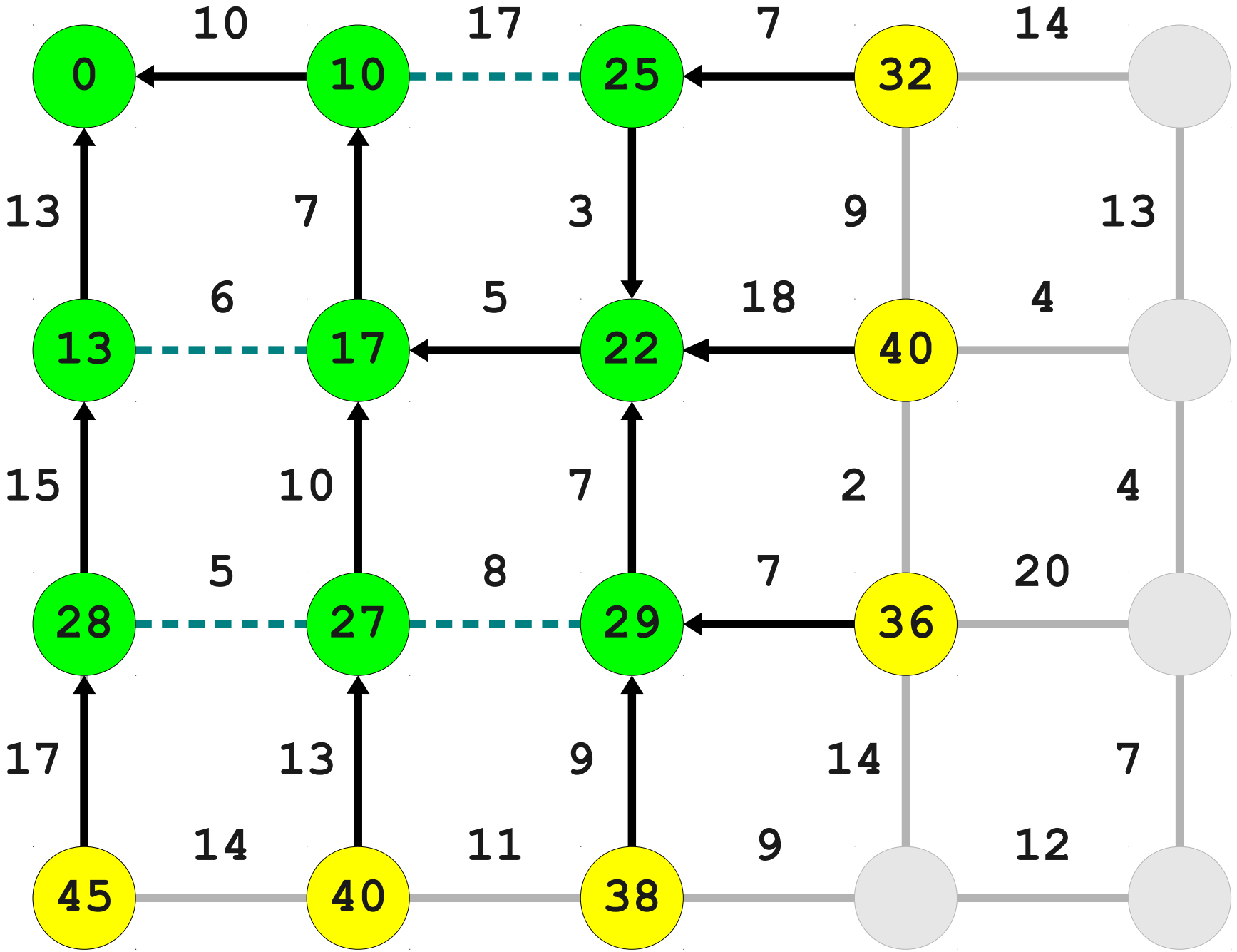

10
17
7
14
0
10
25
32
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
17
13
9
14
7
14
11
9
12

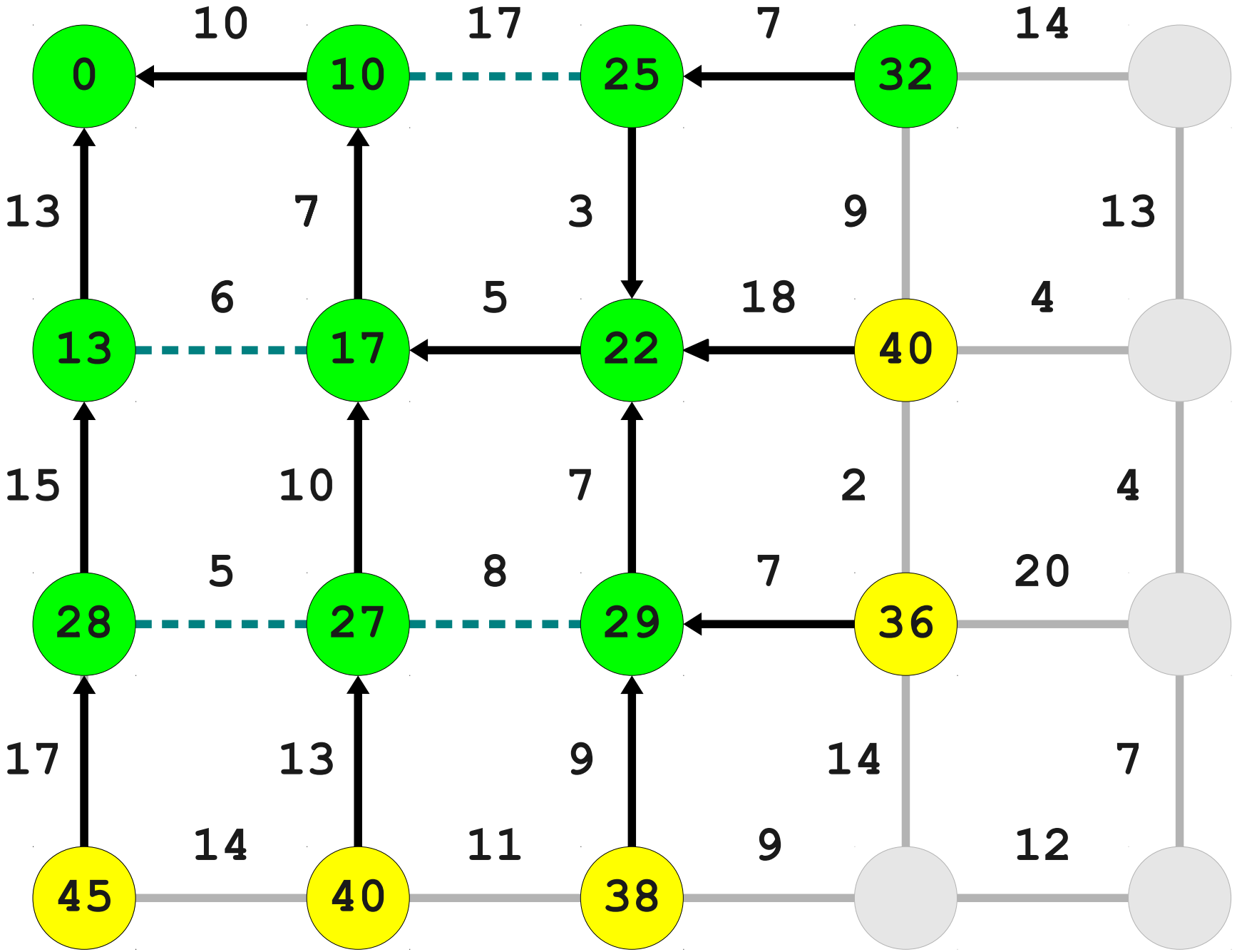

10
17
7
14
0
10
25
32
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
17
13
9
14
7
14
11
9
12
40

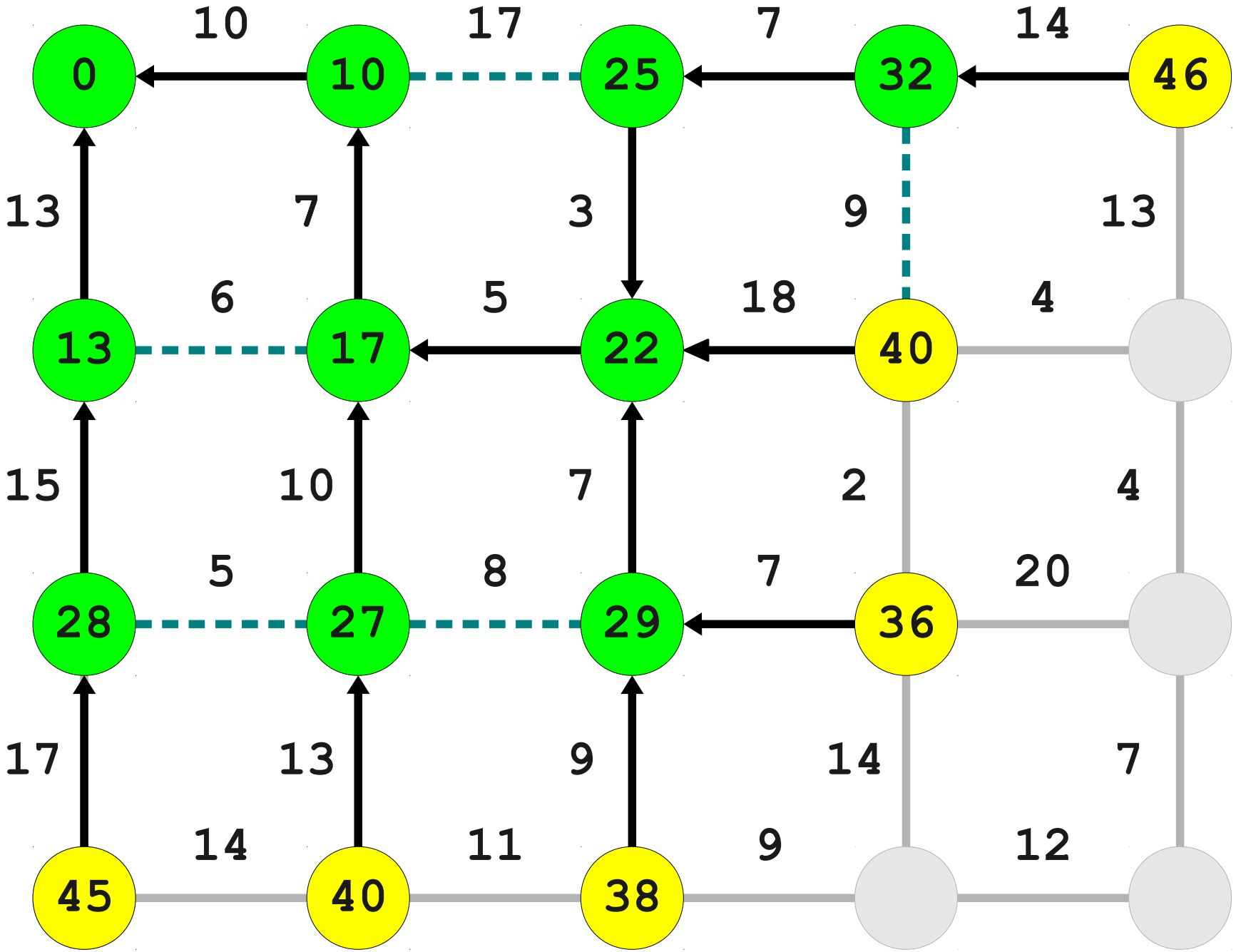

10
17
7
14
0
10
25
32
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
17
13
9
14
7
14
11
9
12
40

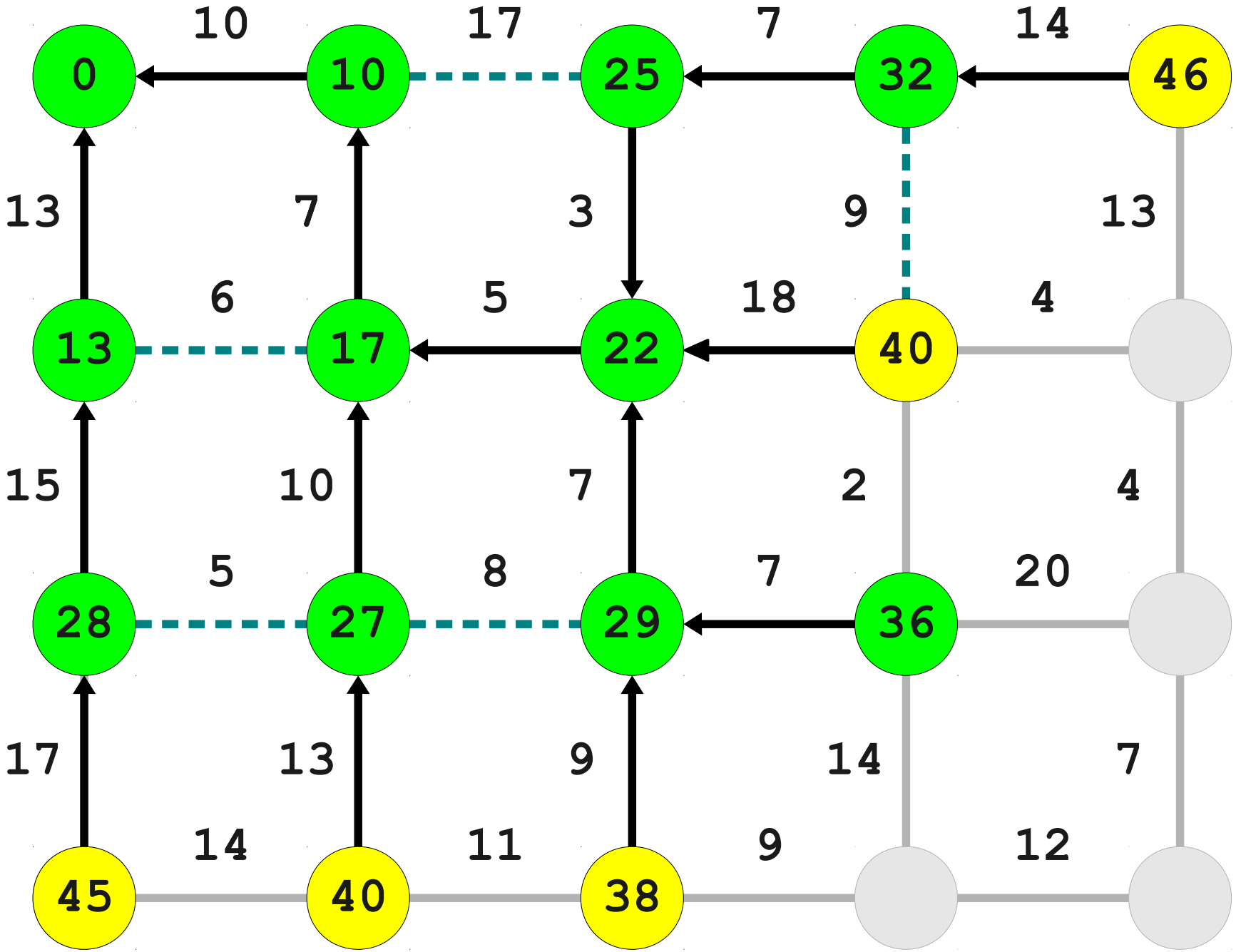

10
17
7
14
0
10
25
32
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
17
13
9
14
7
14
11
9
12
45
40

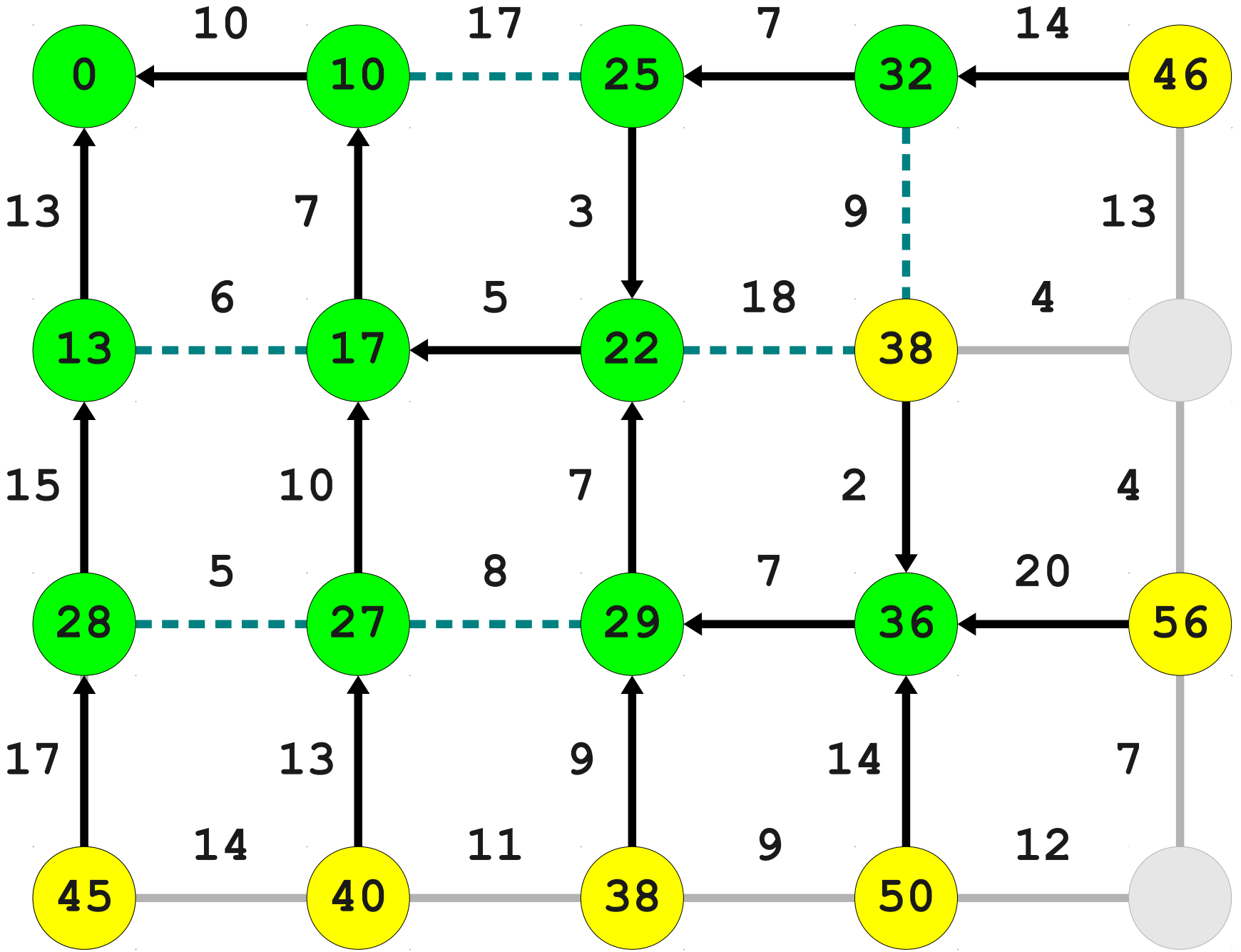

10
17
7
14
0
10
25
32
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
17
13
9
14
7
14
11
9
12
45
40

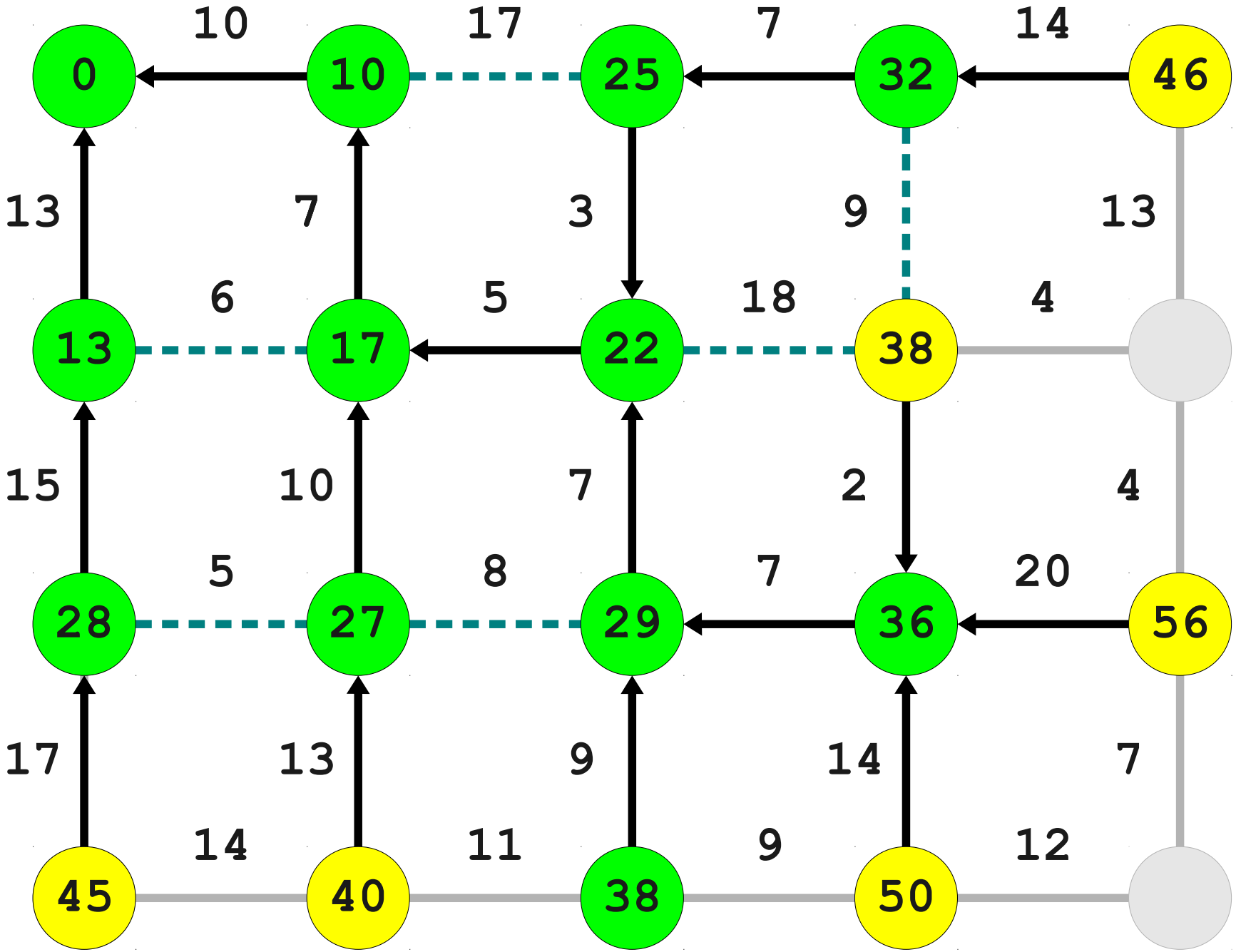

10
17
7
14
0
10
25
32
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
36
17
13
9
14
7
14
11
9
12
45
40
38

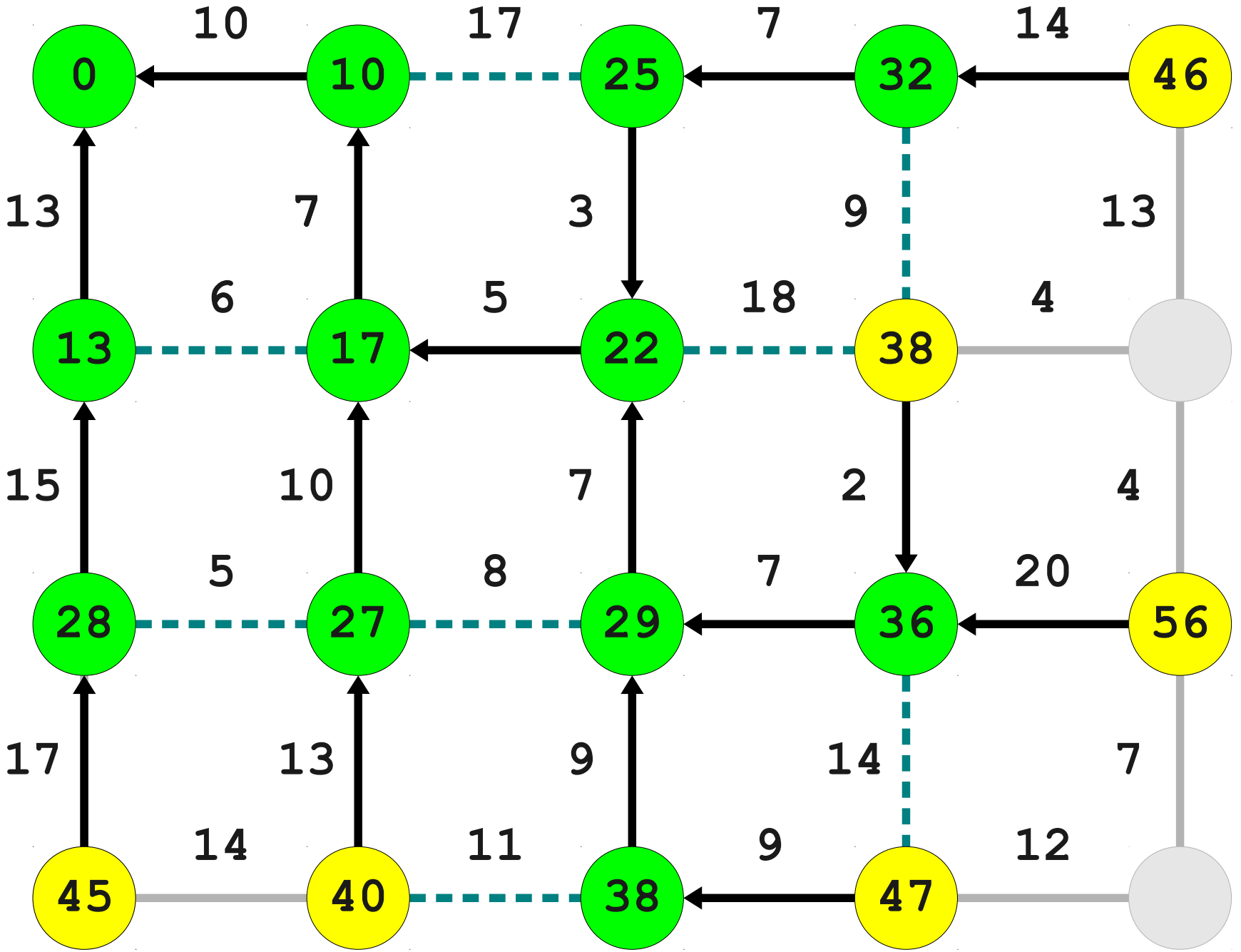

10
17
7
14
0
10
25
32
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
36
17
13
9
14
7
14
11
9
12
45
40
38

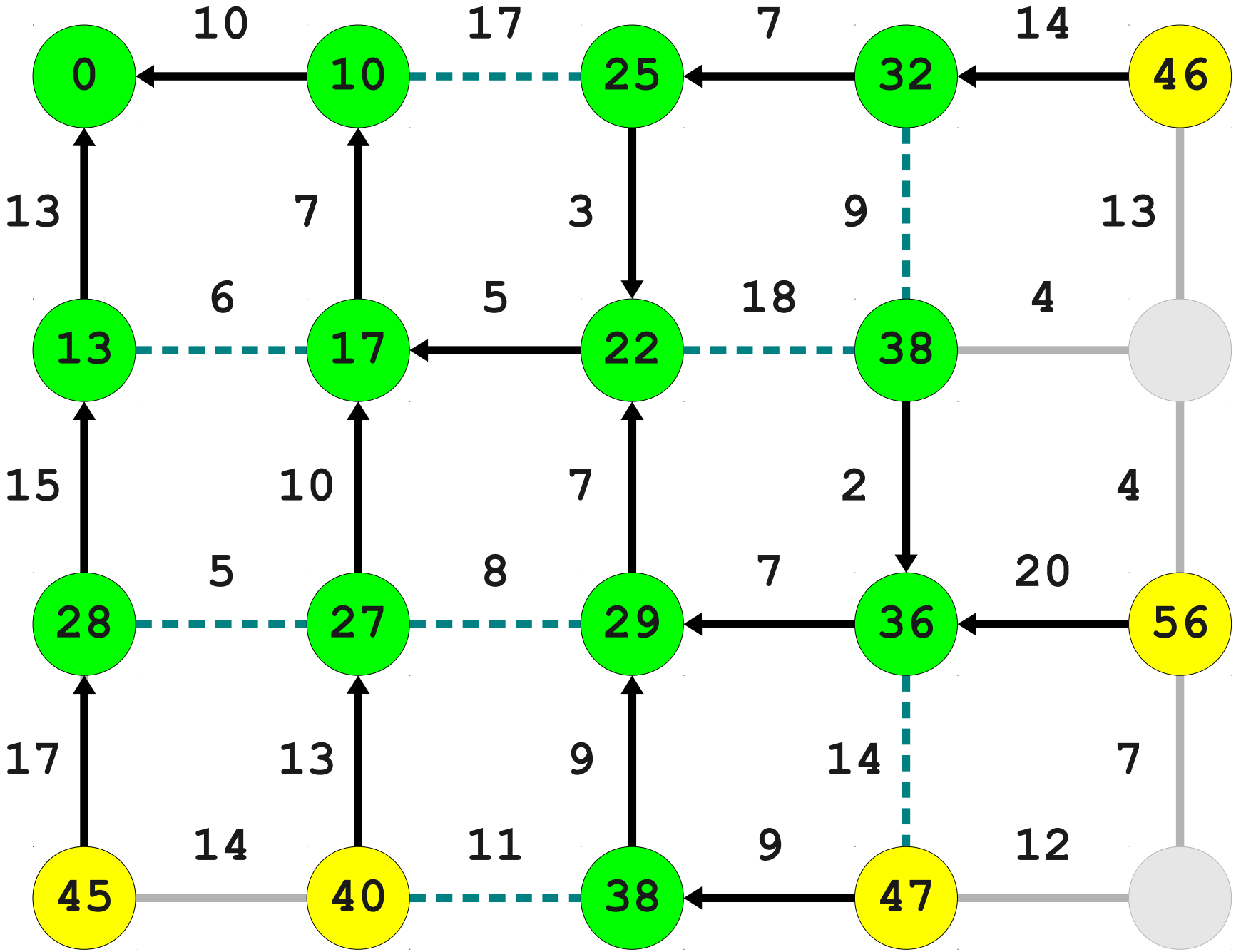

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
36
17
13
9
14
7
14
11
9
12
45
40
38

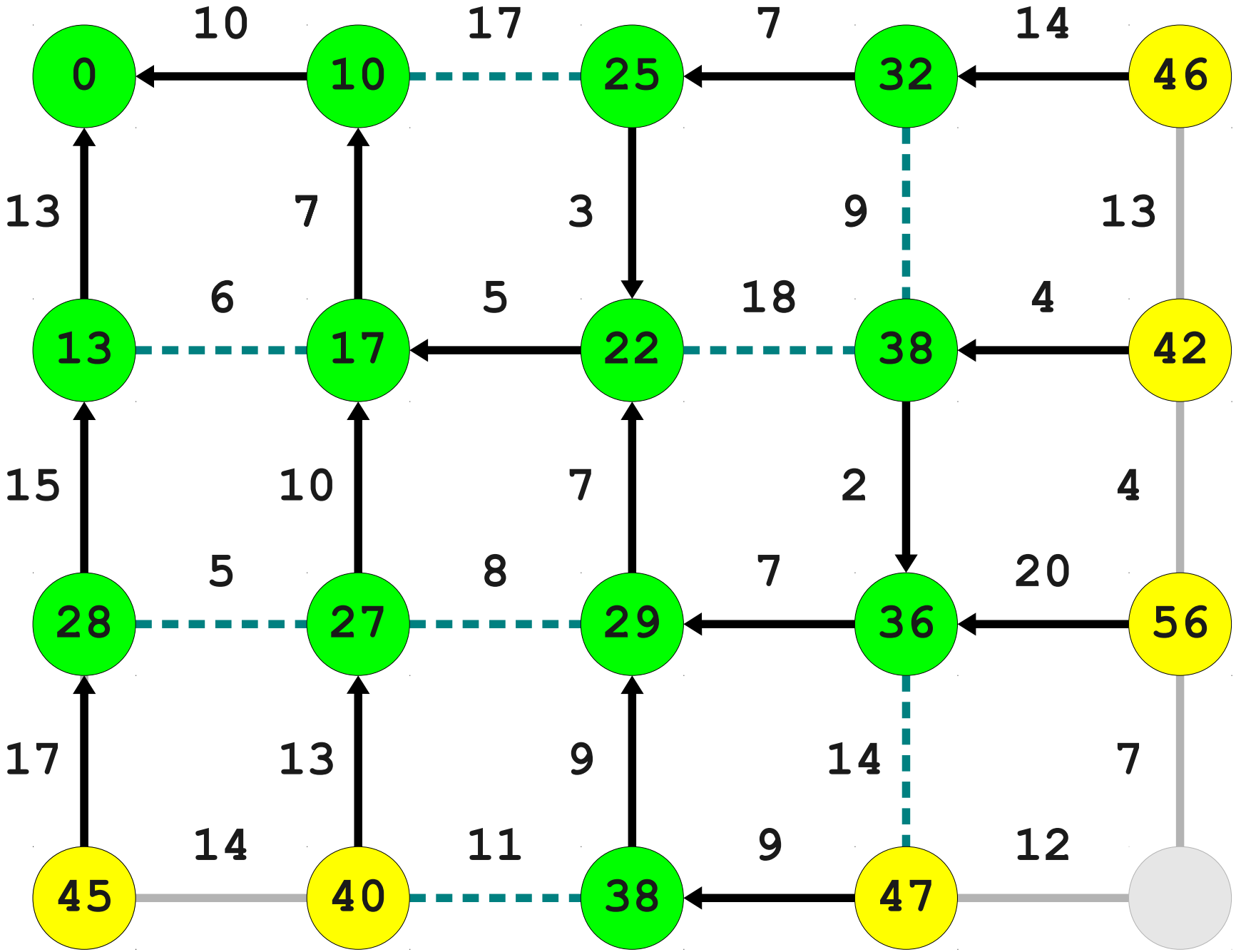

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
40
15
10
7
2
4
5
8
7
20
28
27
29
36
17
13
9
14
7
14
11
9
12
45
40
38

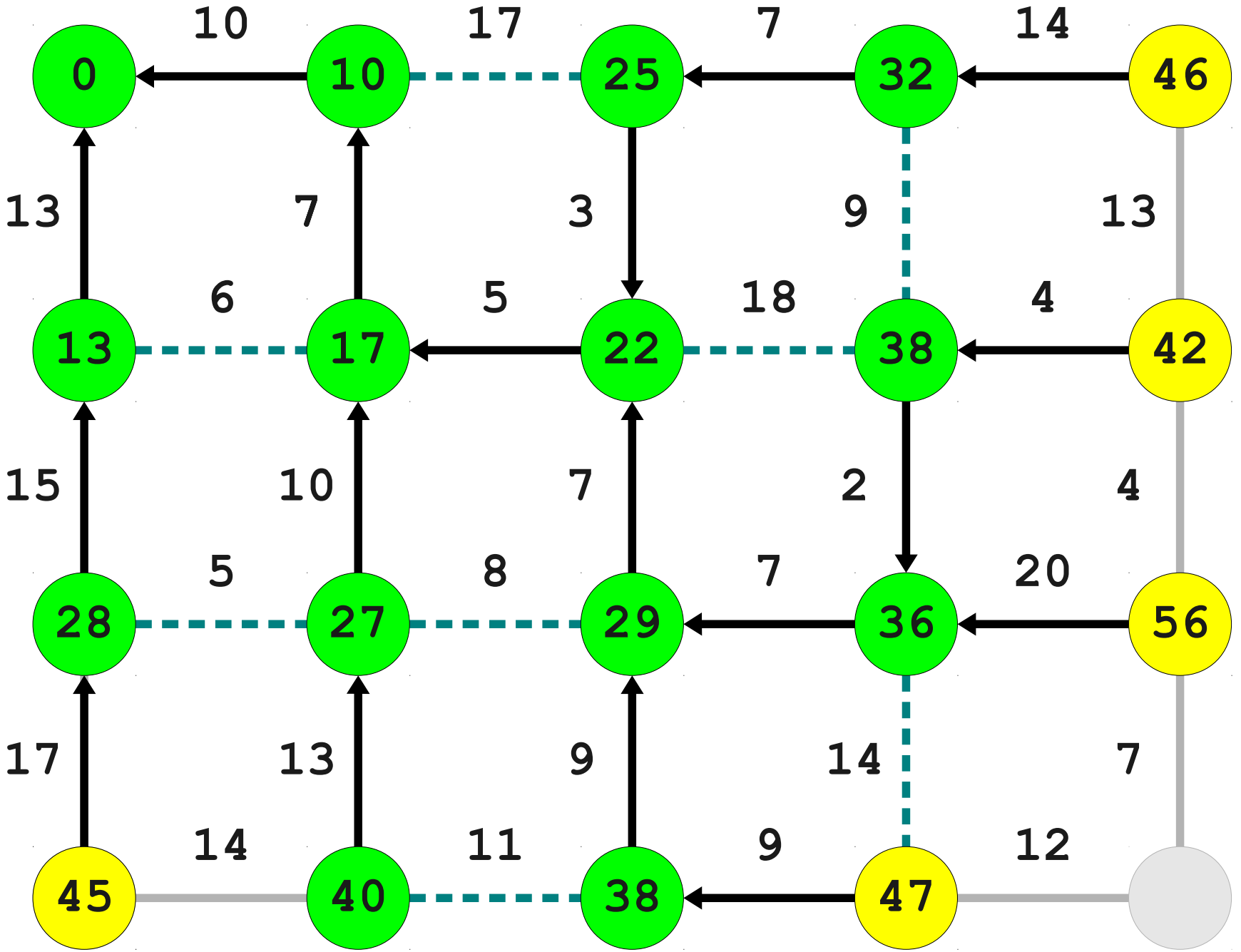

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
15
10
7
2
4
5
8
7
20
28
27
29
36
56
17
13
9
14
7
14
11
9
12
45
40
38
50

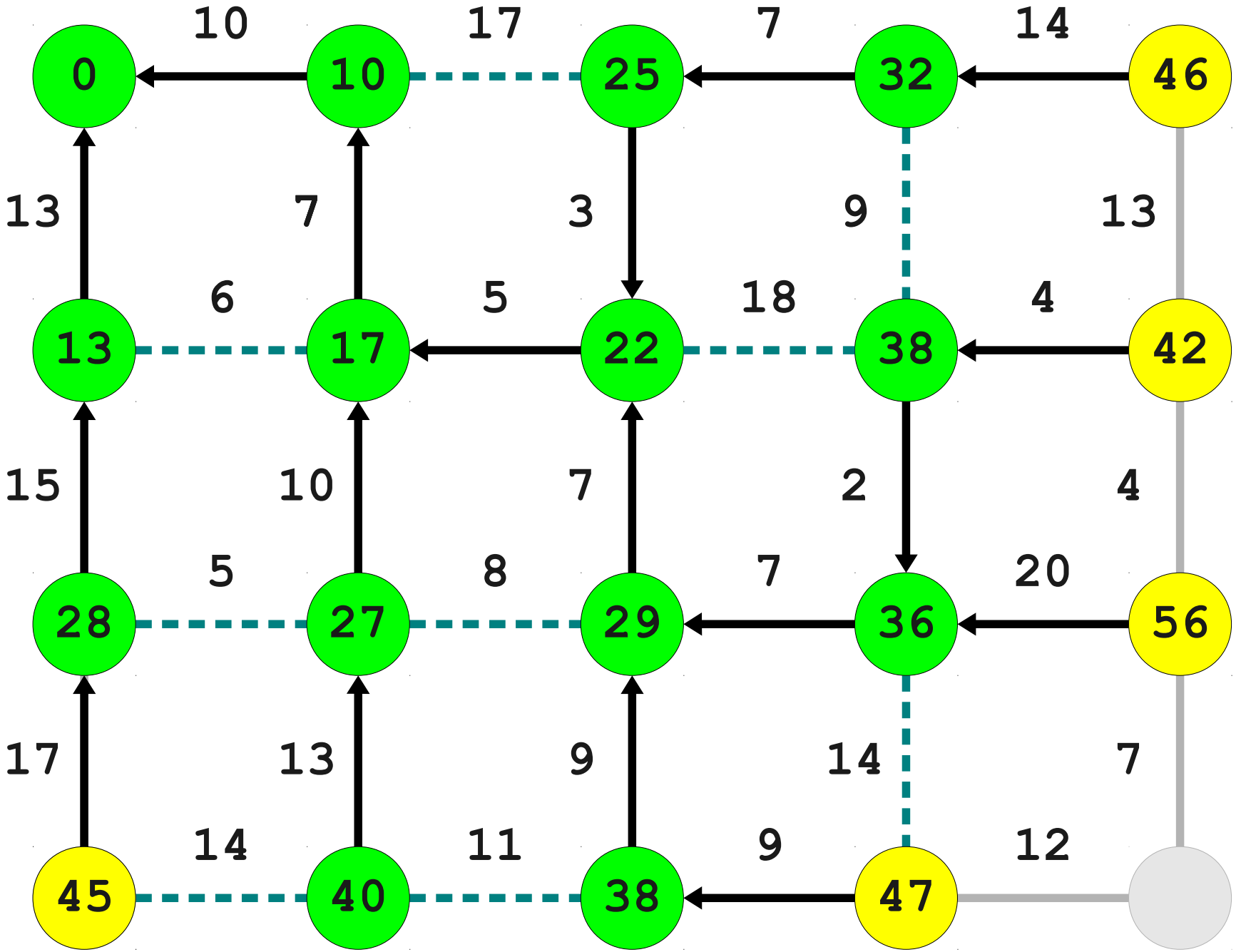

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
15
10
7
2
4
5
8
7
20
28
27
29
36
56
17
13
9
14
7
14
11
9
12
45
40
38
50

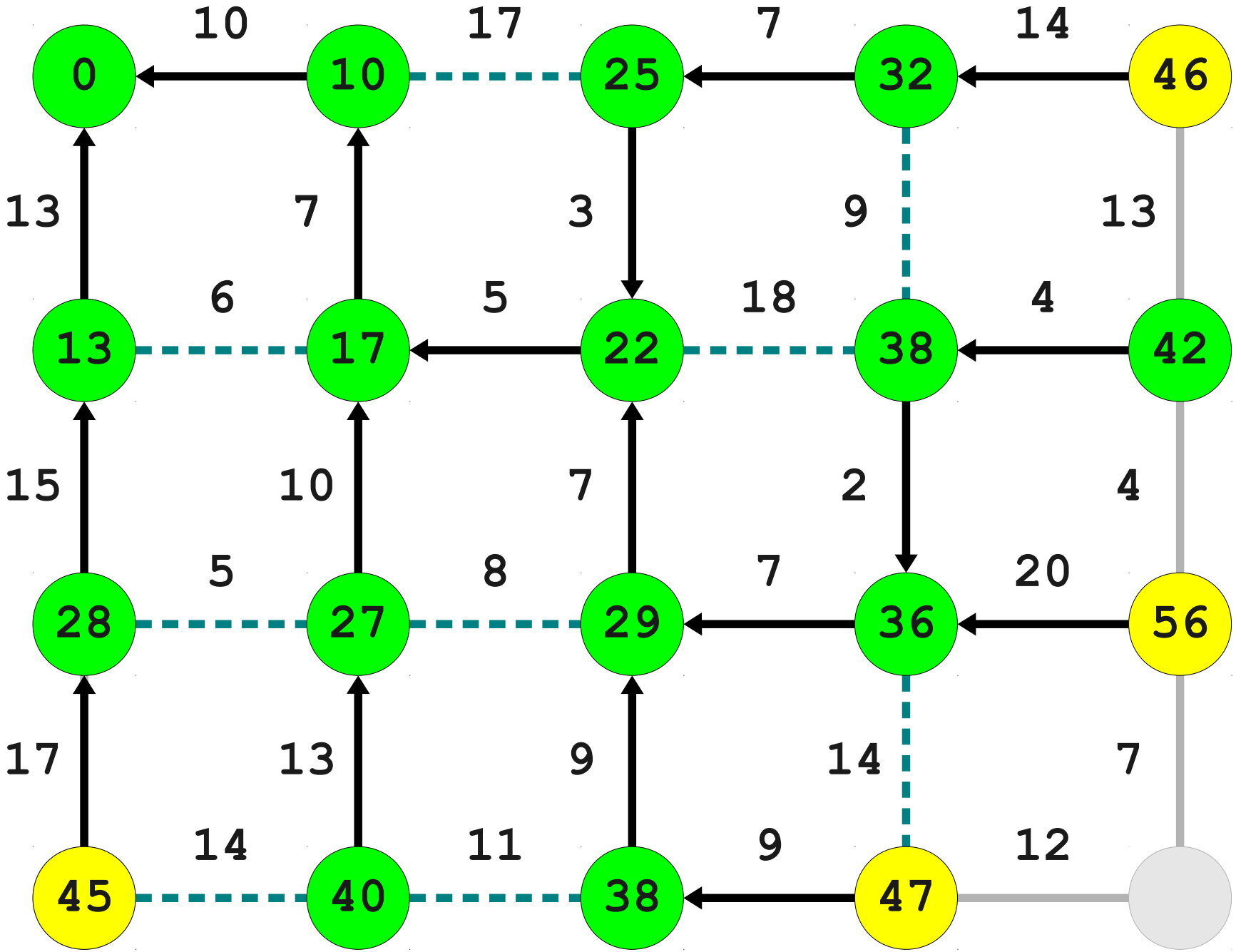

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
15
10
7
2
4
5
8
7
20
28
27
29
36
56
17
13
9
14
7
14
11
9
12
45
40
38
47

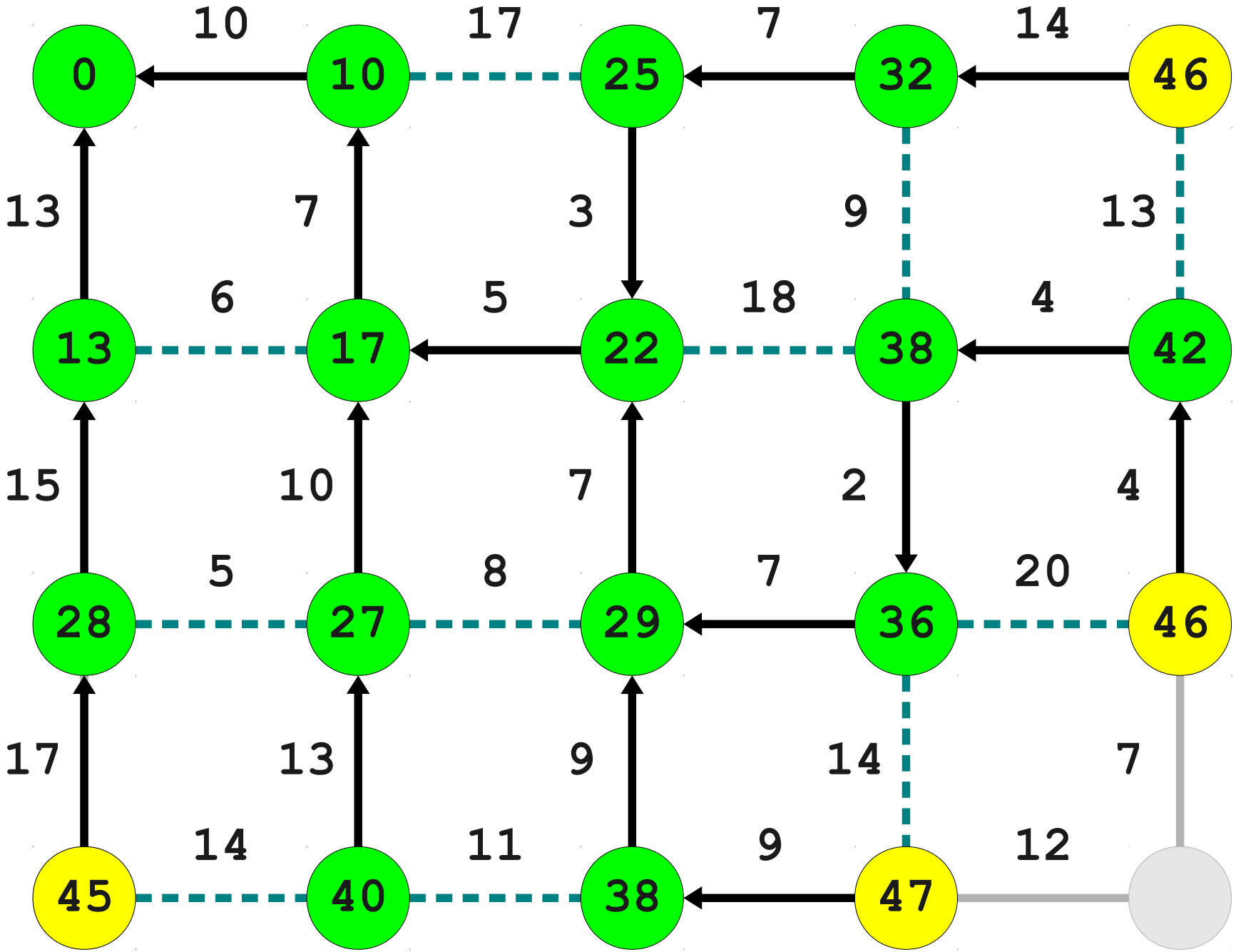

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
15
10
7
2
4
5
8
7
20
28
27
29
36
56
17
13
9
14
7
14
11
9
12
45
40
38
47

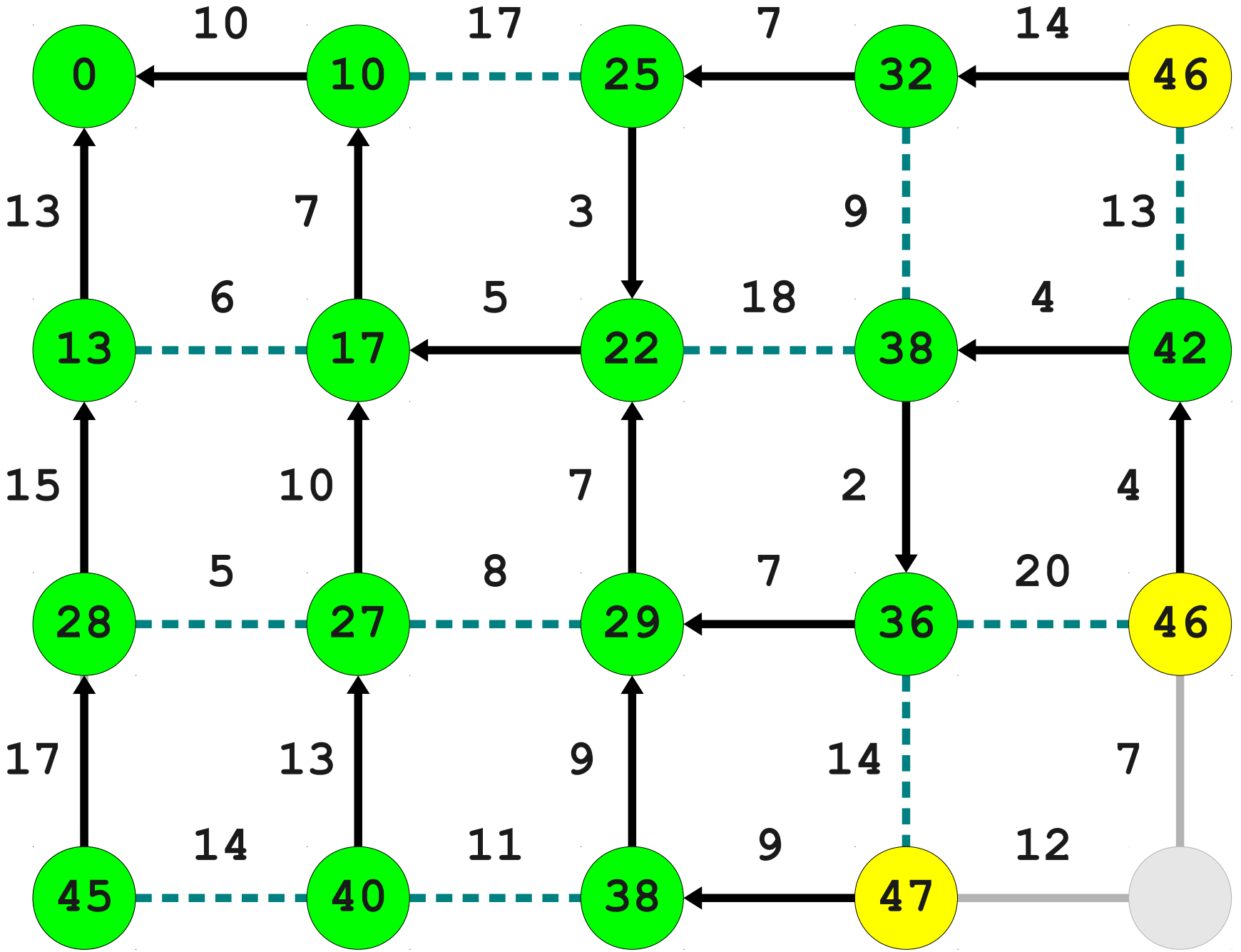

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
56
17
13
9
14
7
14
11
9
12
45
40
38
47

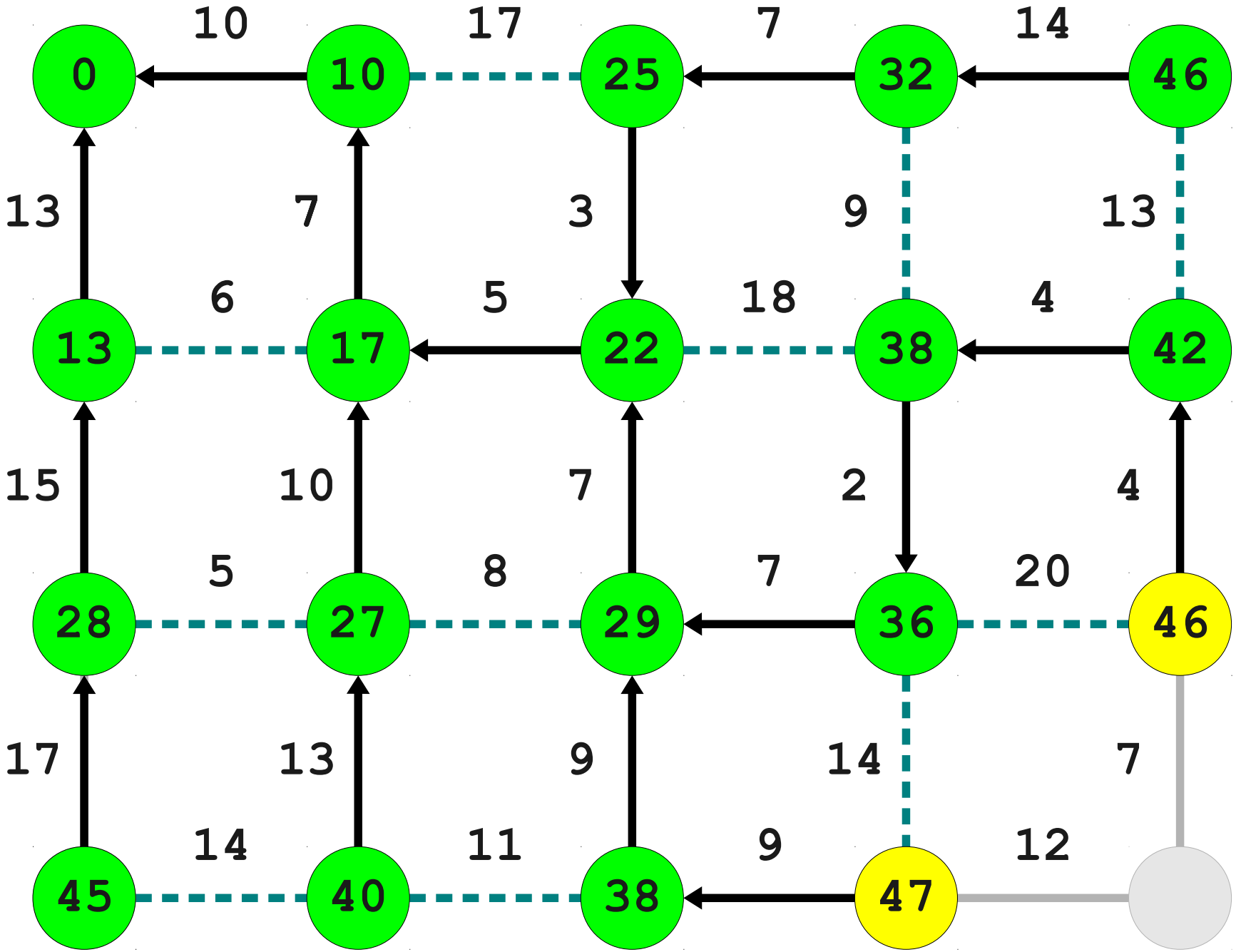

0
10
25
32
46
13
17
22
38
42
28
27
29
36
56
45
40
38
47
10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

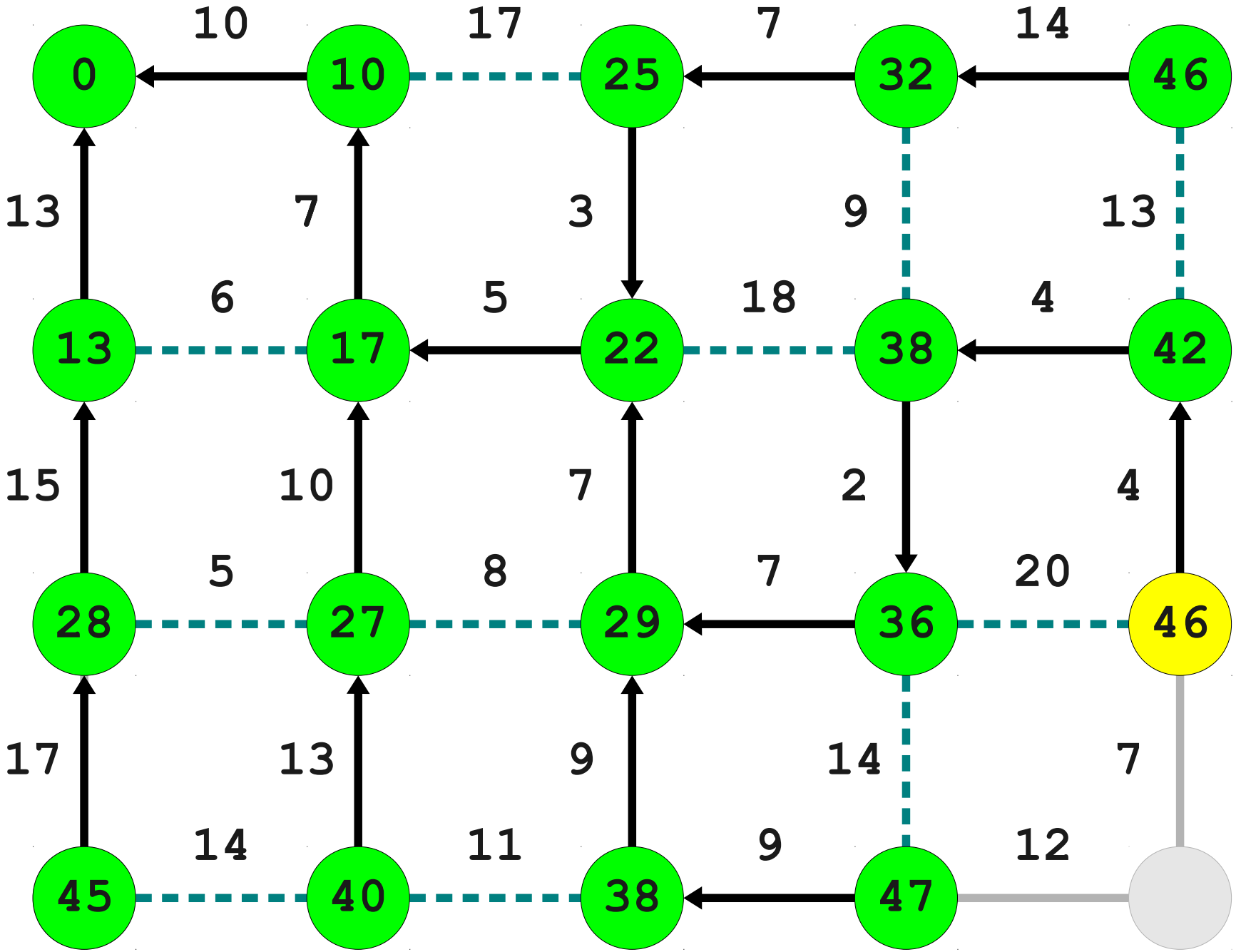

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
56
17
13
9
14
7
14
11
9
12
45
40
38
47

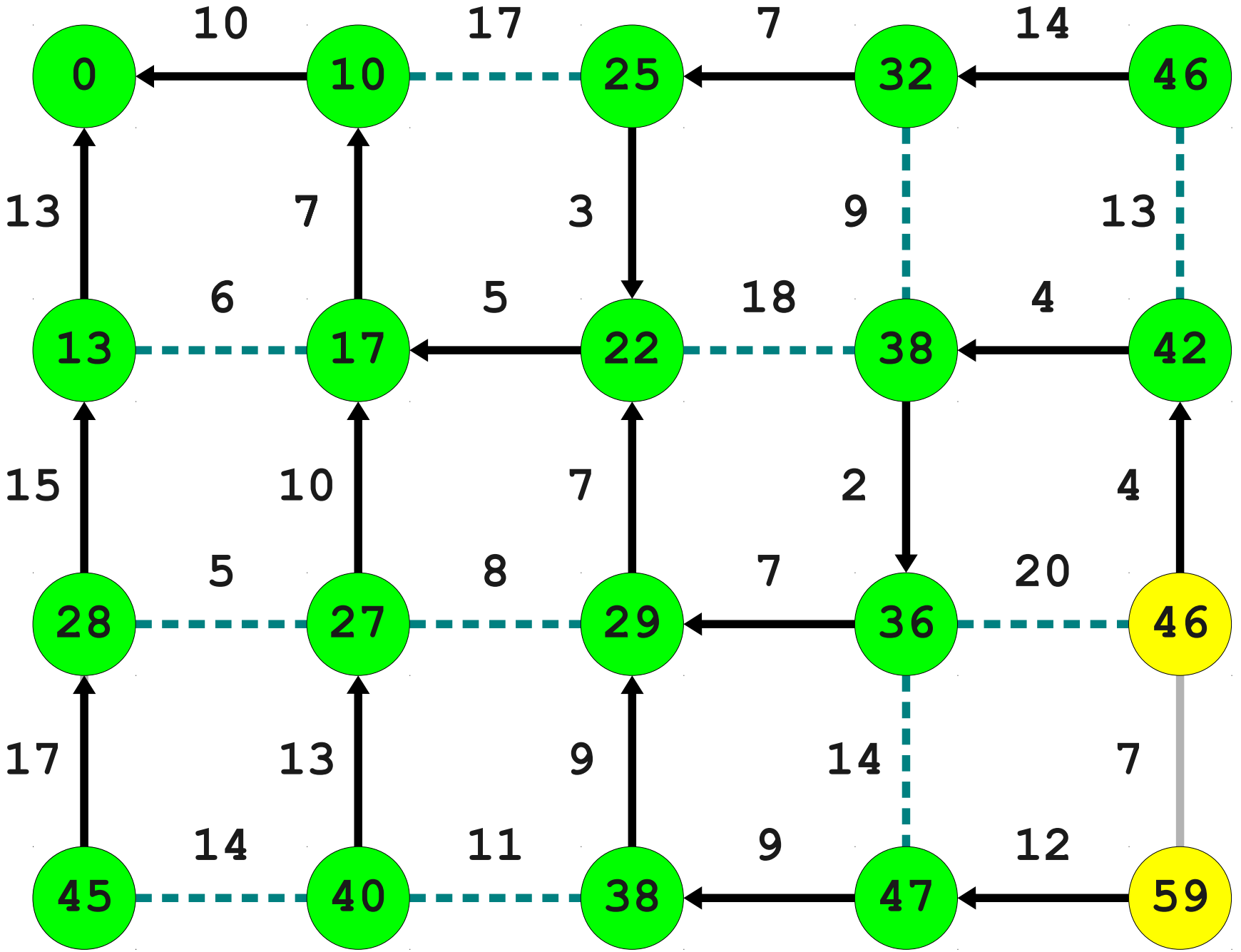

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
56
17
13
9
14
7
14
11
9
12
45
40
38
47

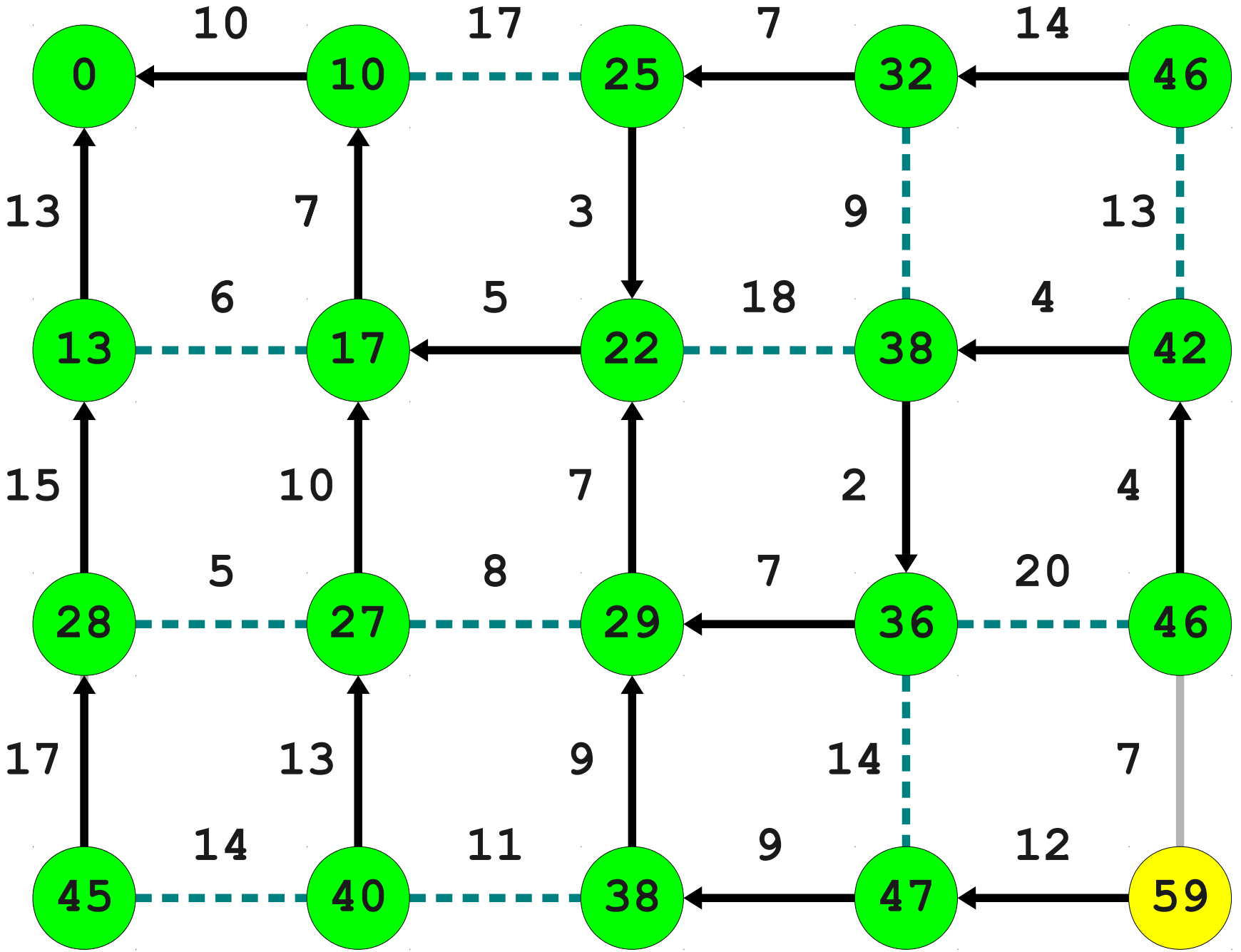

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
46
17
13
9
14
7
14
11
9
12
45
40
38
47

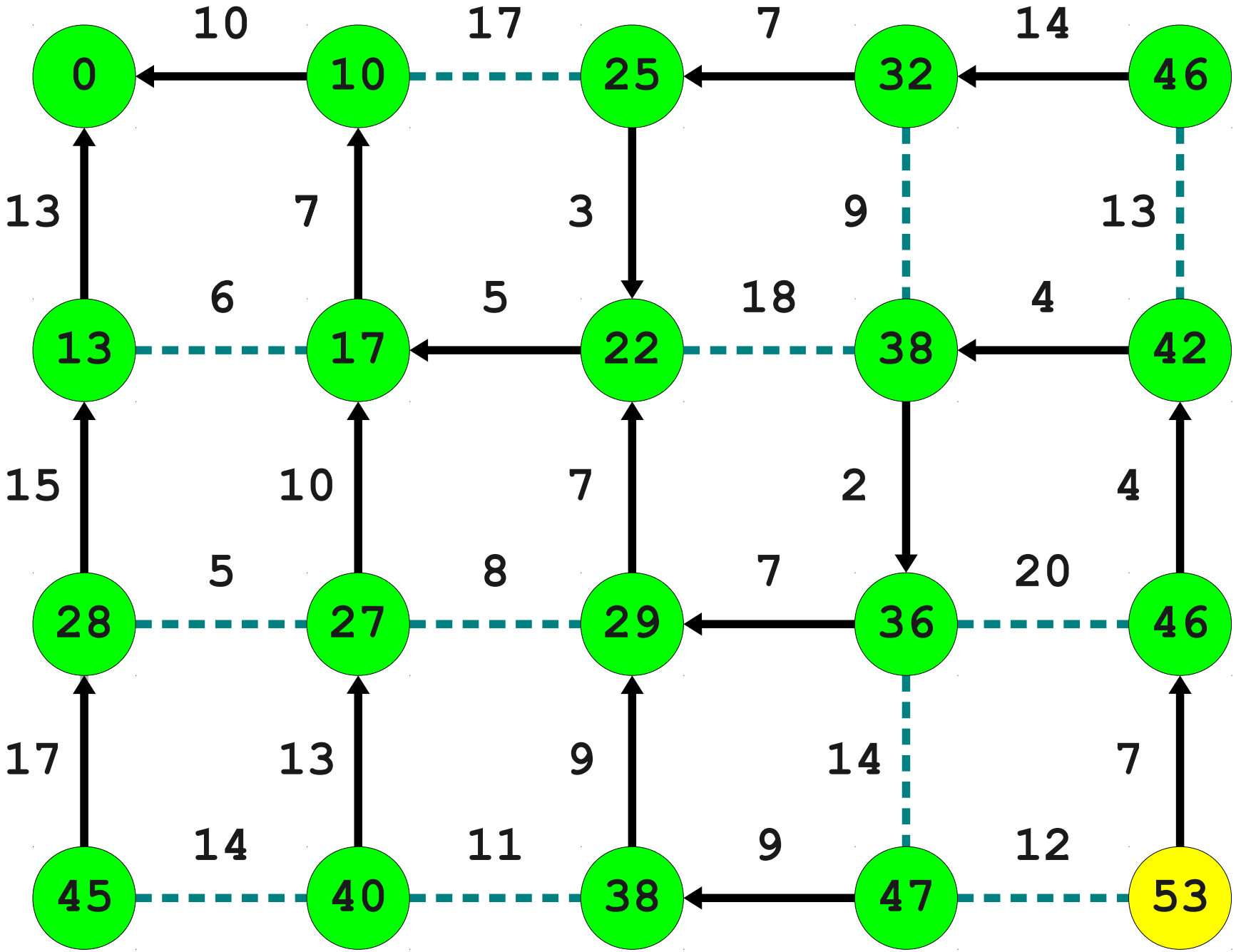

0
10
25
32
46
13
17
22
38
42
28
27
29
36
46
45
40
38
47
10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

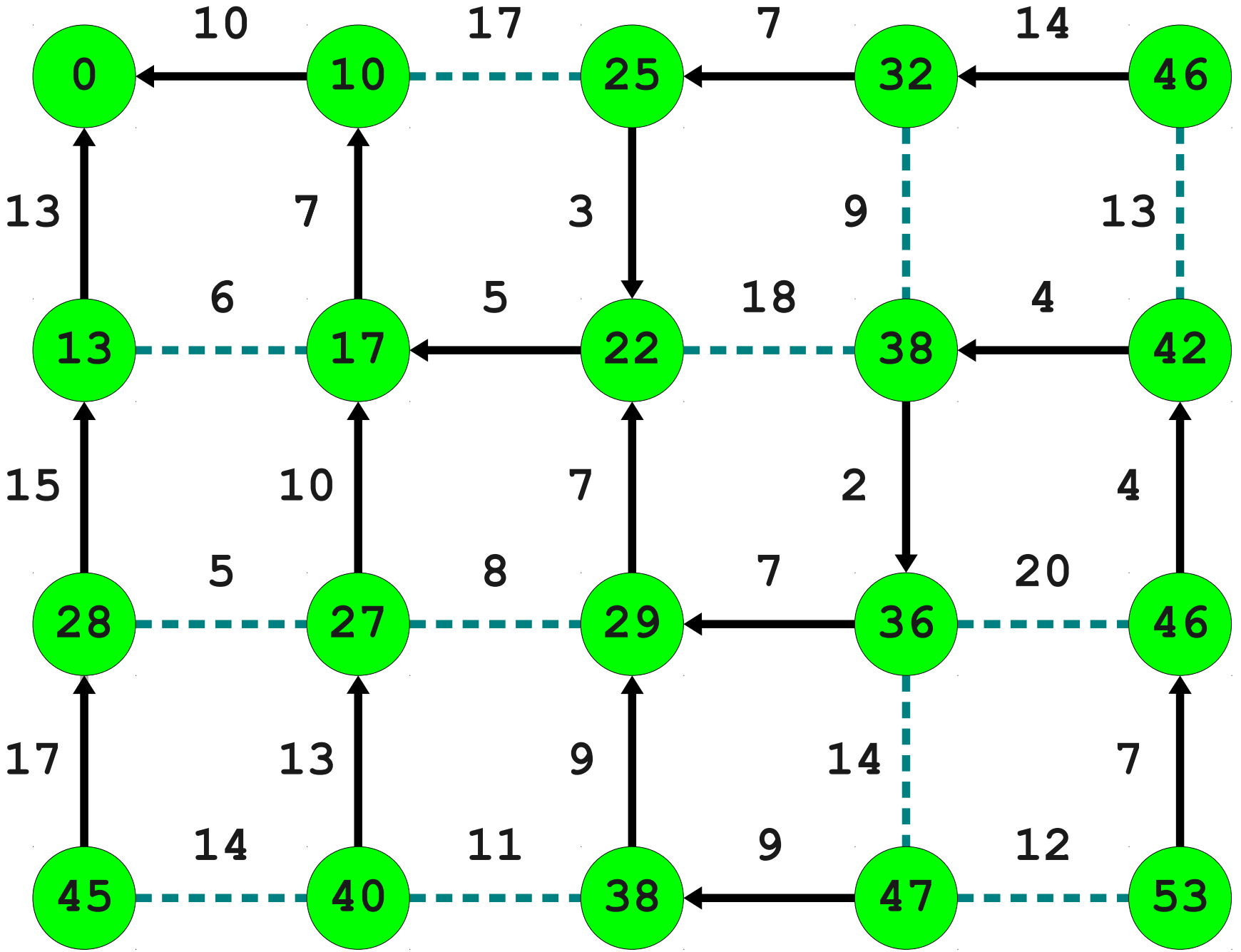

0
10
25
32
46
13
17
22
38
42
28
27
29
36
46
45
40
38
47
10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12

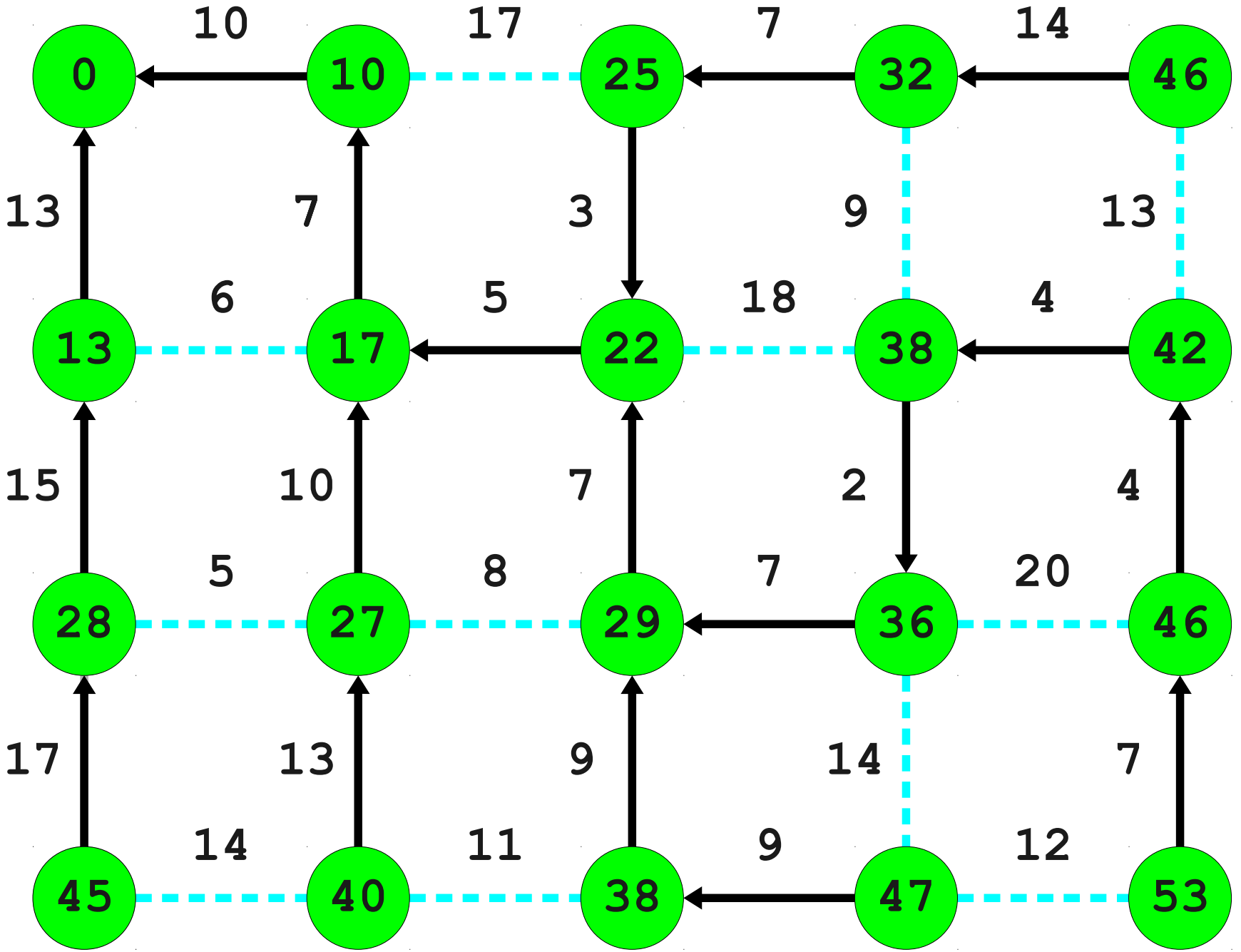

10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
46
17
13
9
14
7
14
11
9
12
45
40
38
47

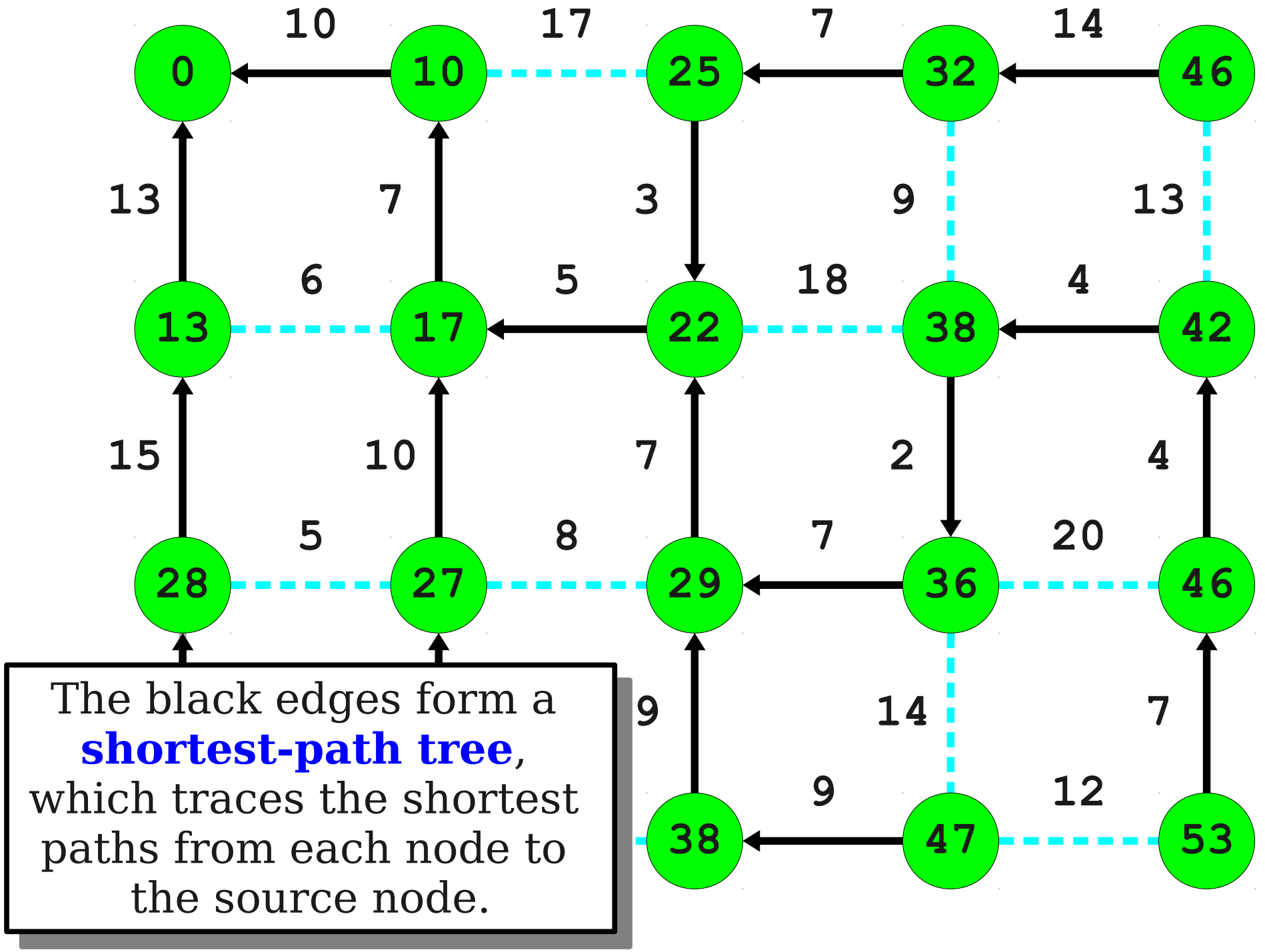

0
10
25
32
46
13
17
22
38
42
28
27
29
36
46
45
40
38
47
59
10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12


10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
46
17
13
9
14
7
14
11
9
12
45
40
38
47
59


0
10
25
32
46
13
17
22
38
42
28
27
29
36
46
45
40
38
47
53
10
17
7
14
13
7
3
9
13
6
5
18
4
15
10
7
2
4
5
8
7
20
17
13
9
14
7
14
11
9
12


10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
46
17
13
9
14
7
14
11
9
12
45
40
38
47
53


10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
46
17
13
9
14
7
14
11
9
12
45
40
38
47
53


10
17
7
14
0
10
25
32
46
13
7
3
9
13
6
5
18
4
13
17
22
38
42
15
10
7
2
4
5
8
7
20
28
27
29
36
46
9
14
7
9
12
38
47
53
The black edges form a shortest-path tree, which traces the shortest paths from each node to the source node.

```
procedure dijkstrasAlgorithm(s, G):
 let q be a new queue
 for each v in V:
    dist[v] = ∞

 dist[s] = 0
 enqueue(s, q)

 while q is not empty:
    let v be a node in q minimizing dist[v]
    remove(v, q)

    for each node u connected to v:
       if dist[u] > dist[v] + l(u, v):
          dist[u] = dist[v] + l(u, v)
          if u is not enqueued into q:
             enqueue(u, q)
```

Dijkstra's Algorithm

- Assuming nonnegative edge lengths, finds the shortest path from s to each node in G.
- Correctness proof sketch is based on the second argument for breadth-first search:
  - Always picks the node v minimizing $\mathrm{d}(s, u) + l(u, v)$ for yellow v and green u.
  - If a shorter path P exists to v, it must leave the set of green nodes through some edge (x, y).
  - But then $l(P)$ is at least $\mathrm{d}(s, x) + l(x, y)$, which is at least $\mathrm{d}(s, u) + l(u, v)$.
  - So the “shorter” path costs at least as much as the path we found.

```
procedure dijkstrasAlgorithm(s, G):
 let q be a new queue
 for each v in V:
    dist[v] = ∞

 dist[s] = 0
 enqueue(s, q)

 while q is not empty:
    let v be a node in q minimizing dist[v]
    remove(v, q)

    for each node u connected to v:
       if dist[u] > dist[v] + l(u, v):
          dist[u] = dist[v] + l(u, v)
          if u is not enqueued into q:
             enqueue(u, q)
```

```
procedure dijkstrasAlgorithm(s, G):
 let q be a new queue
 for each v in V:
    dist[v] = ∞

 dist[s] = 0
 enqueue(s, q)

 while q is not empty:
    let v be a node in q minimizing dist[v]
    remove(v, q)

    for each node u connected to v:
       if dist[u] > dist[v] + l(u, v):
          dist[u] = dist[v] + l(u, v)
          if u is not enqueued into q:
             enqueue(u, q)
```

```
procedure dijkstrasAlgorithm(s, G):
 let q be a new queue
 for each v in V:
    dist[v] = ∞

 dist[s] = 0
 enqueue(s, q)

 while q is not empty:
    let v be a node in q minimizing dist[v]
    remove(v, q)

    for each node u connected to v:
       if dist[u] > dist[v] + l(u, v):
          dist[u] = dist[v] + l(u, v)
          if u is not enqueued into q:
             enqueue(u, q)
```

$O(m + n)$

```
procedure dijkstrasAlgorithm(s, G):
 let q be a new queue
 for each v in V:
    dist[v] = ∞

 dist[s] = 0
 enqueue(s, q)

 while q is not empty:
    let v be a node in q minimizing dist[v]
    remove(v, q)

    for each node u connected to v:
       if dist[u] > dist[v] + l(u, v):
          dist[u] = dist[v] + l(u, v)
          if u is not enqueued into q:
             enqueue(u, q)
```

```
procedure dijkstrasAlgorithm(s, G):
 let q be a new queue
 for each v in V:
    dist[v] = ∞

 dist[s] = 0
 enqueue(s, q)

 while q is not empty:
    let v be a node in q minimizing dist[v]
    remove(v, q)

    for each node u connected to v:
       if dist[u] > dist[v] + l(u, v):
          dist[u] = dist[v] + l(u, v)
          if u is not enqueued into q:
             enqueue(u, q)
```

$O(n^2)$

Dijkstra Runtime

- Using a standard implementation of a queue, Dijkstra's algorithm runs in time **$O(n^2)$**.
  - $O(n + m)$ time processing nodes and edges, plus $O(n^2)$ time finding the lowest-cost node.
  - Since $m = O(n^2)$, $O(n + m + n^2) = O(n^2)$.
- Using a slightly fancier data structure (a binary heap), can be made to run in time **$O(m \log n)$**.
  - Is this necessarily more efficient?
  - More on how to do this later this quarter.
- Using a *much* fancier data structure (the *Fibonacci heap*), can be made to run in time **$O(m + n \log n)$**.
  - Take CE354 for details!

Shortest Path Algorithms

- If all edges have the same weight, can use breadth-first search to find shortest paths.
  - Takes time $O(m + n)$.
- If edges have nonnegative weight, can use Dijkstra's algorithm.
  - Takes time $O(n^2)$, or less using more complex data structures.
- What about the case where edges can have negative weight?
  - More on that later in the quarter...

```
INITIALIZE-SINGLE-SOURCE(G, s)
1   for each vertex v ∈ G.V
2       v.d = ∞
3       v.π = NIL
4   s.d = 0
```

```
DIJKSTRA(G, w, s)
 1  INITIALIZE-SINGLE-SOURCE(G, s)
 2  S = ∅
 3  Q = ∅
 4  for each vertex u ∈ G.V
 5      INSERT(Q, u)
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      S = S ∪ {u}
 9      for each vertex v in G.Adj[u]
10          RELAX(u, v, w)
11          if the call of RELAX decreased v.d
12              DECREASE-KEY(Q, v, v.d)
```

```
RELAX(u, v, w)
1   if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```

Dijkstra's algorithm: which priority queue?

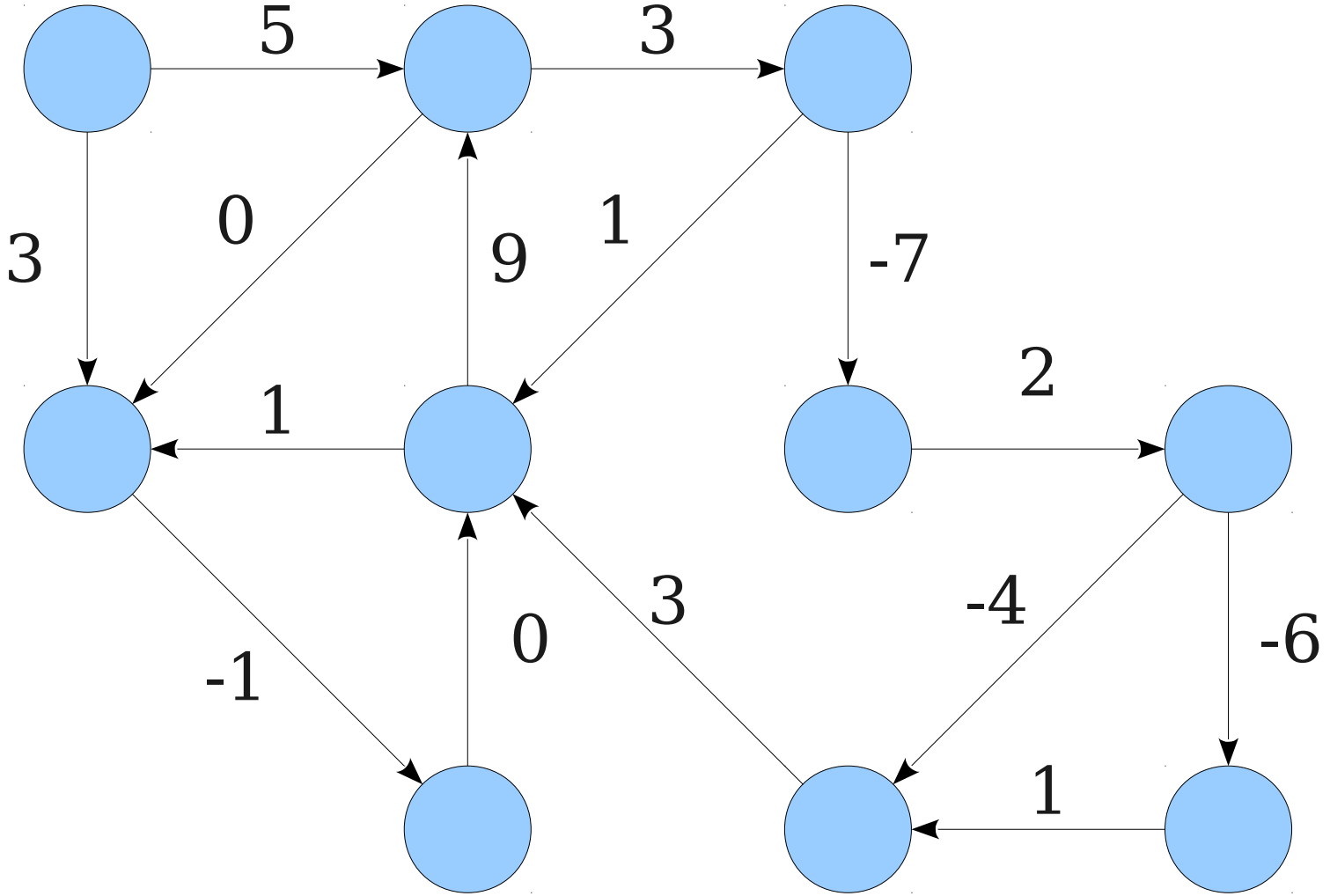
Performance. Depends on PQ: n INSERT, n DELETE-MIN, $\leq m$ DECREASE-KEY.

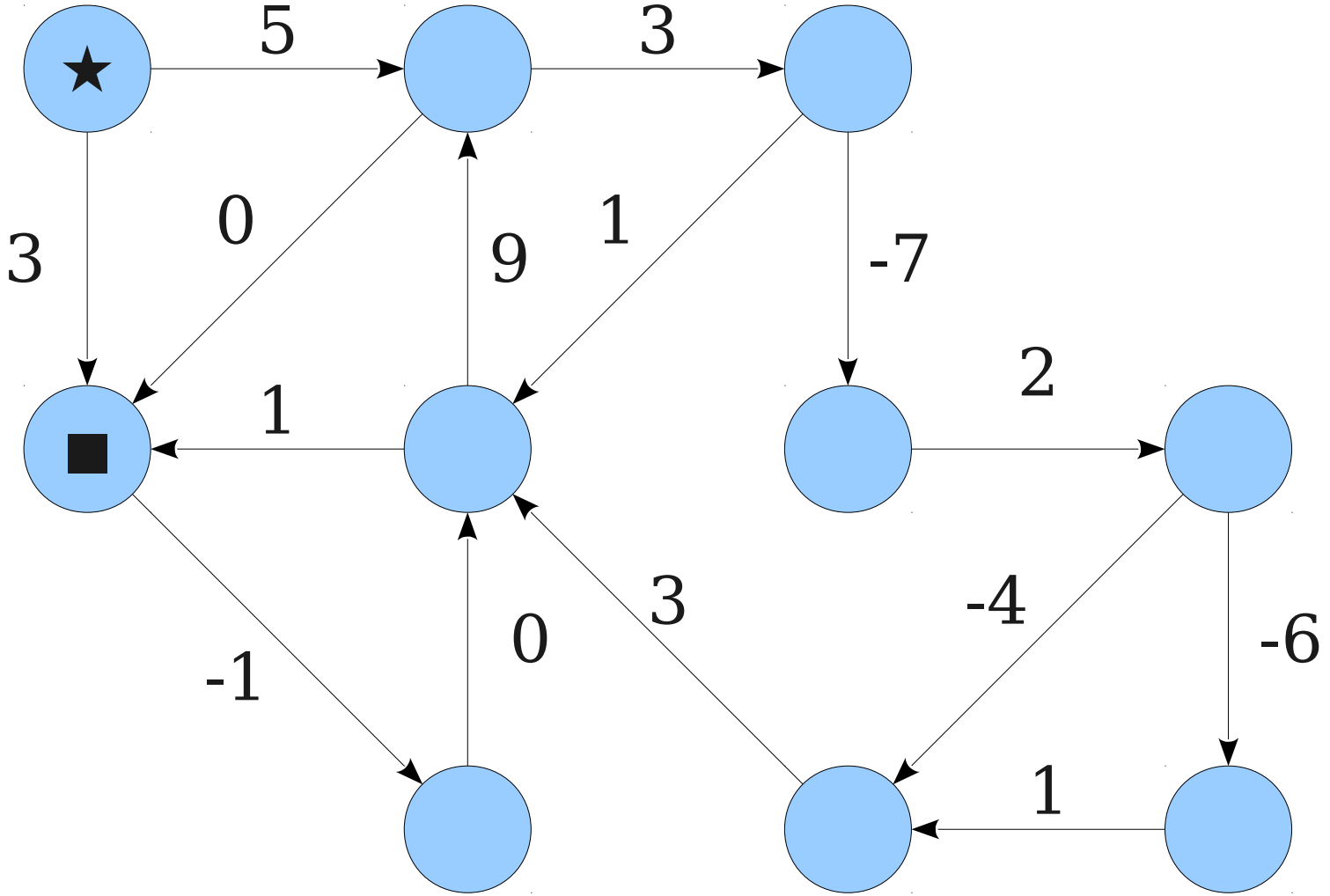
- Array implementation optimal for dense graphs. ⟵ $\Theta(n^2)$ edges
- Binary heap much faster for sparse graphs. ⟵ $\Theta(n)$ edges
- 4-way heap worth the trouble in performance-critical situations.

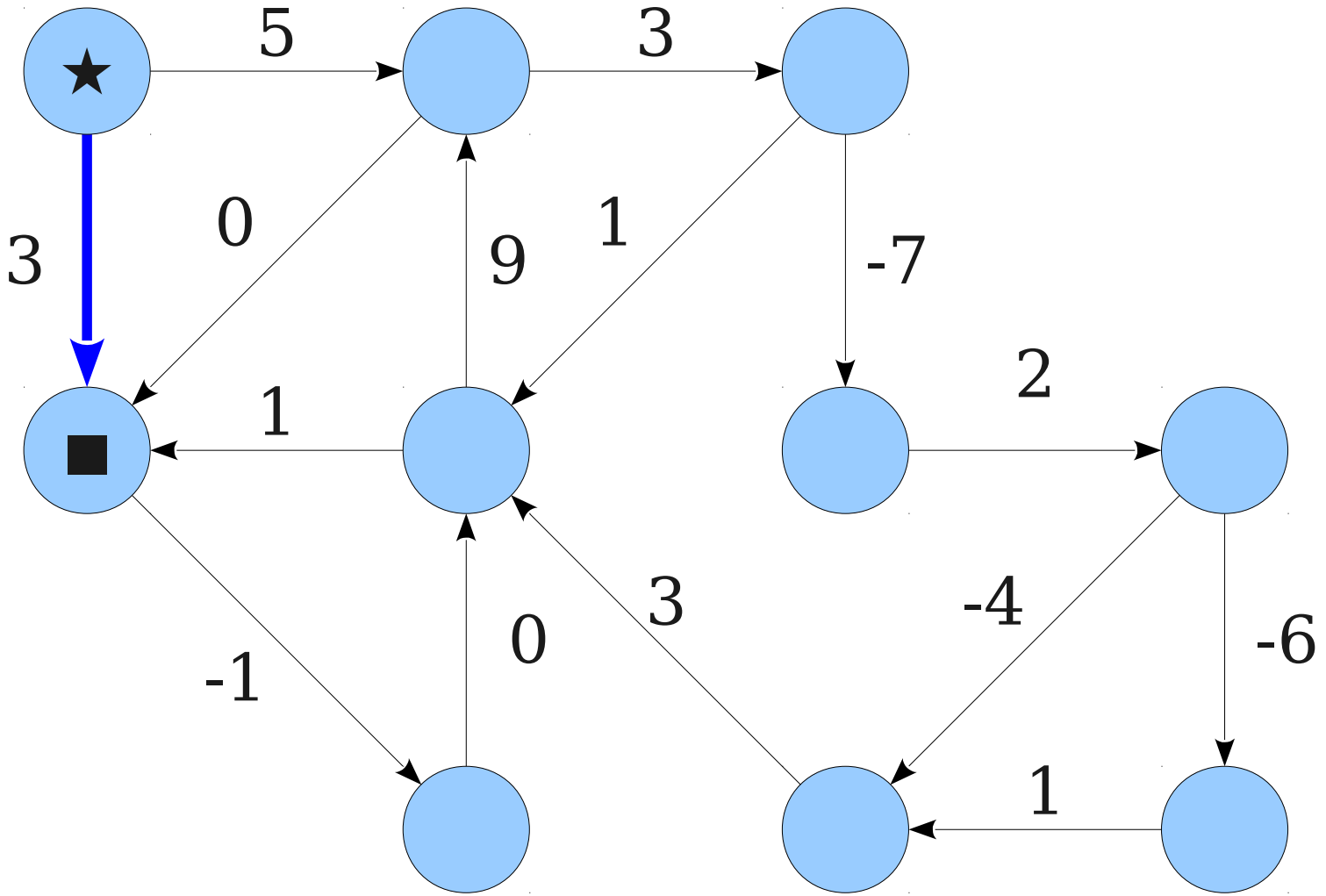
priority queue	INSERT	DELETE-MIN	DECREASE-KEY	total
node-indexed array (A[i] = priority of i)	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
binary heap	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(m \log n)$
d-way heap (Johnson 1975)	$O(d \log_d n)$	$O(d \log_d n)$	$O(\log_d n)$	$O(m \log_{m/n} n)$
Fibonacci heap (Fredman–Tarjan 1984)	$O(1)$	$O(\log n)$ †	$O(1)$ †	$O(m + n \log n)$
integer priority queue (Thorup 2004)	$O(1)$	$O(\log \log n)$	$O(1)$	$O(m + n \log \log n)$

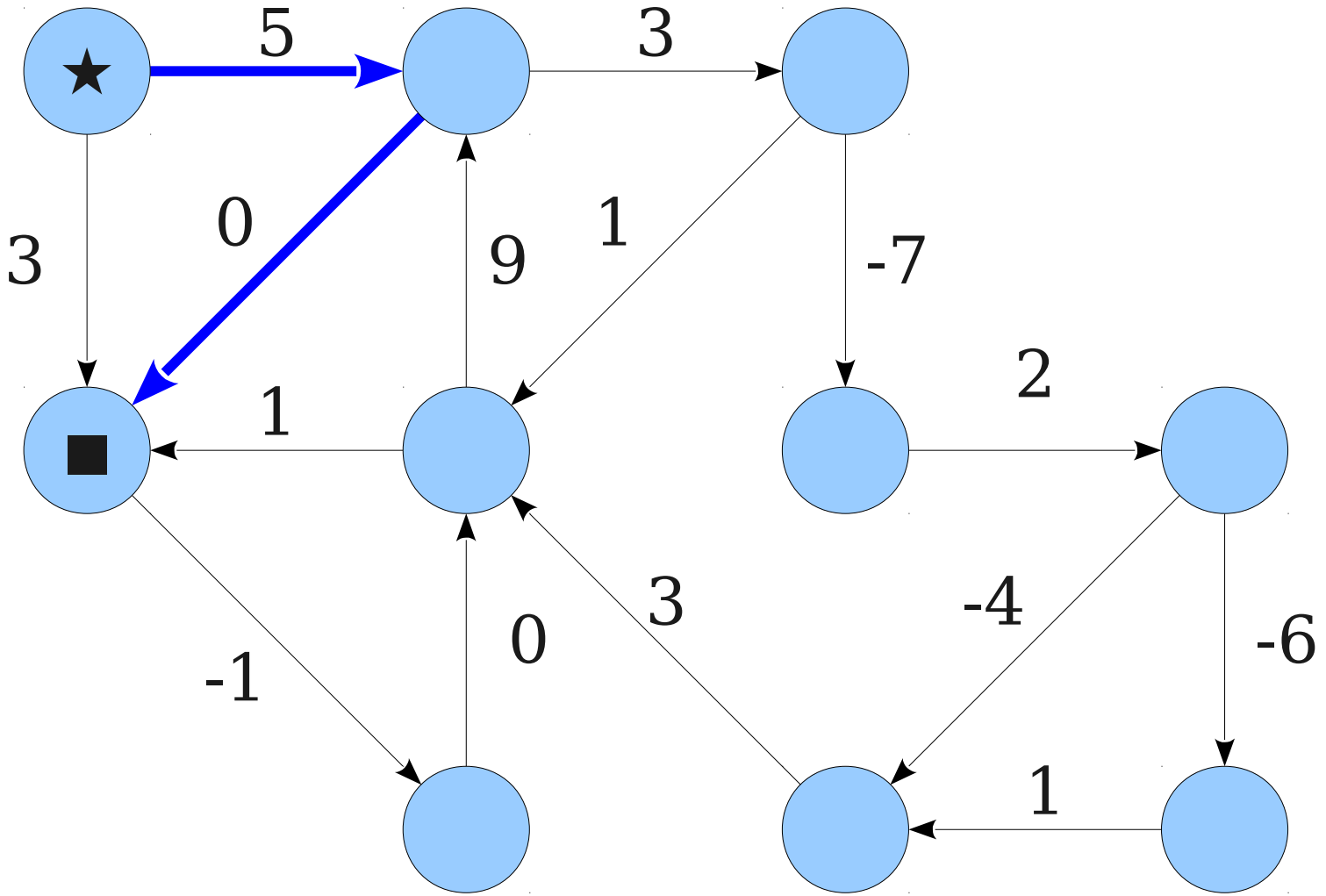
assumes $m \geq n$ † amortized

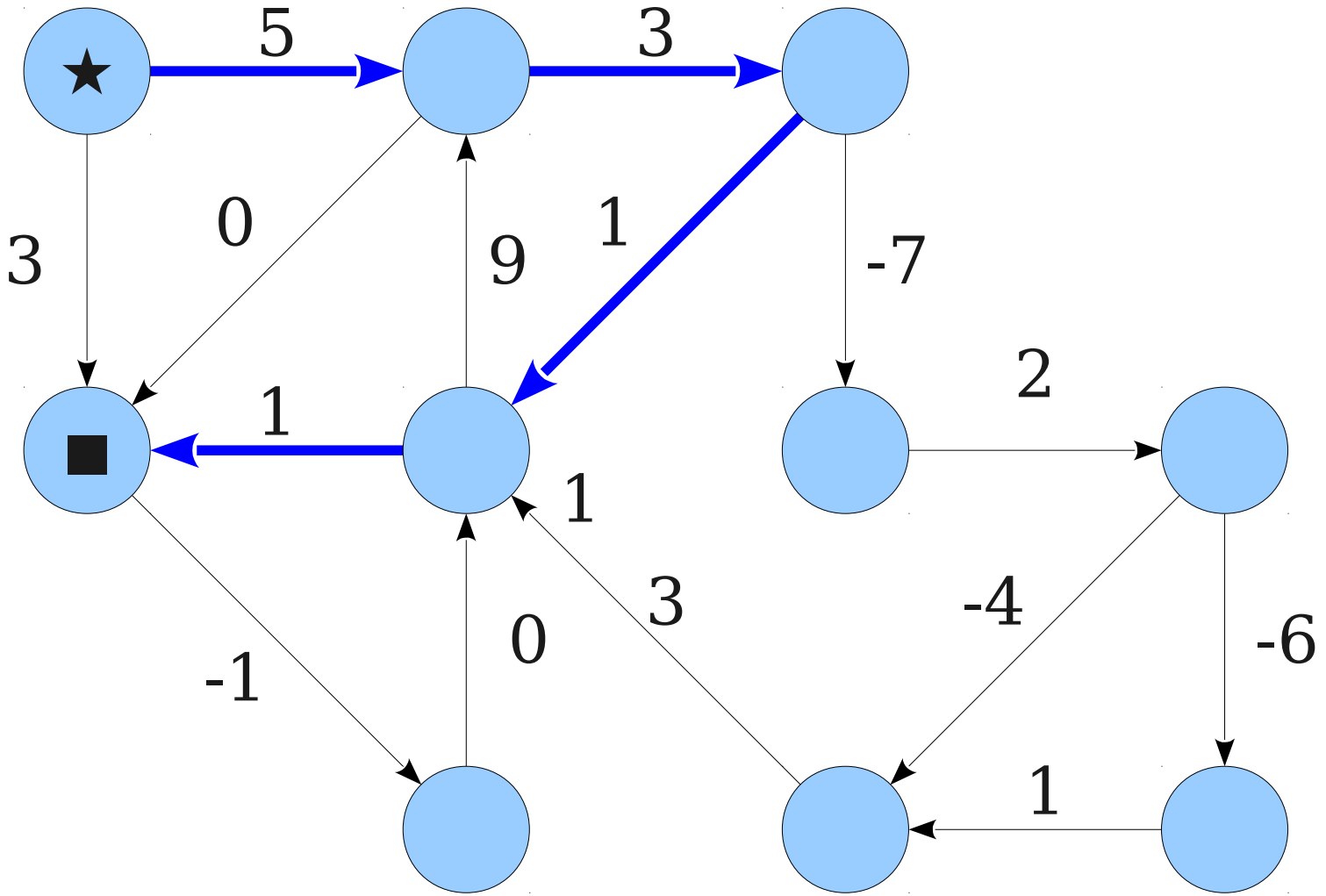
Negative Edge Weights

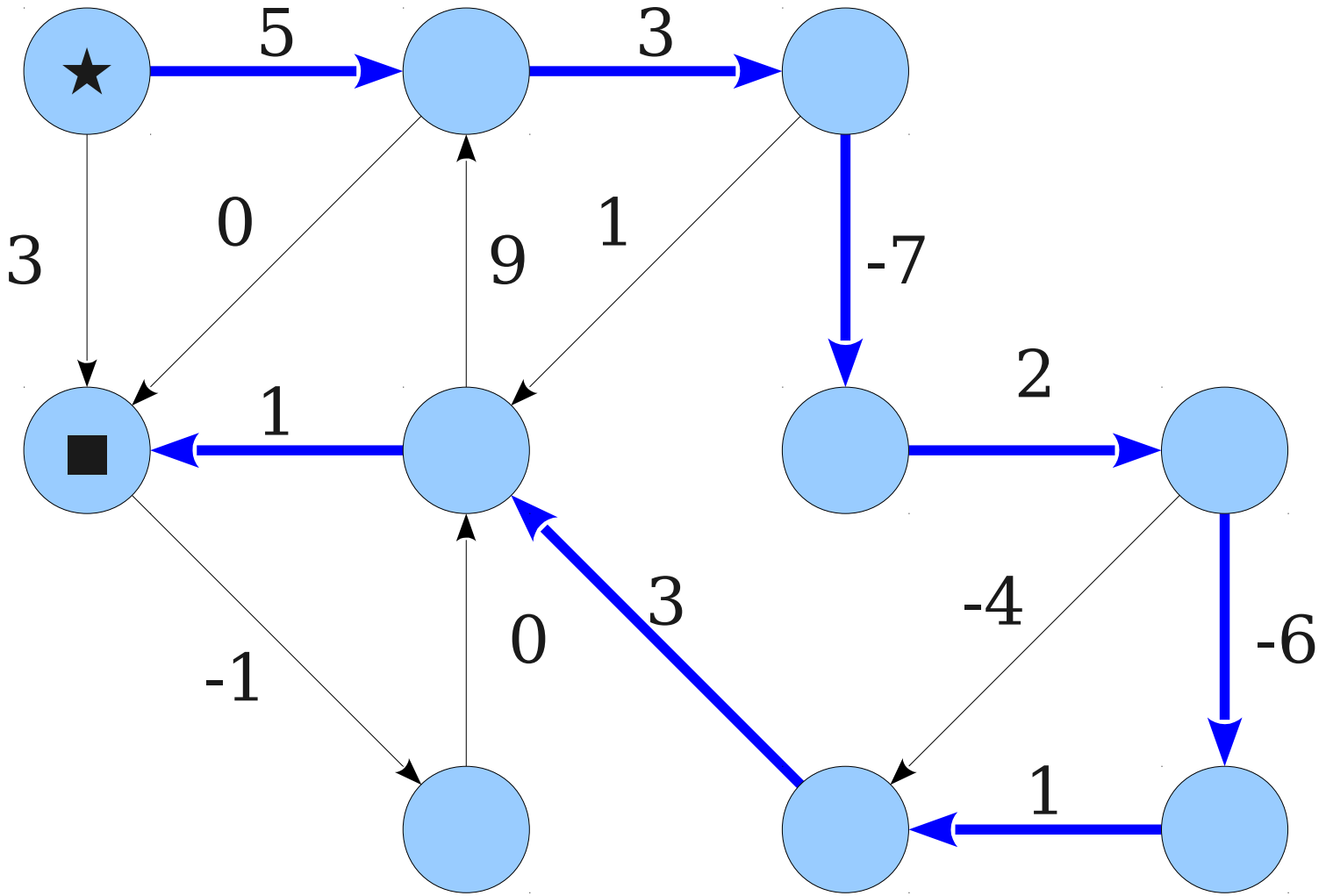

5
3
3
0
9
1
-7
1
2
-1
0
3
-4
-6
1


5
3
3
0
9
1
-7
1
2
-1
0
3
-4
-6
1


5
3
3
0
9
1
-7
1
2
-1
0
3
-4
-6
1

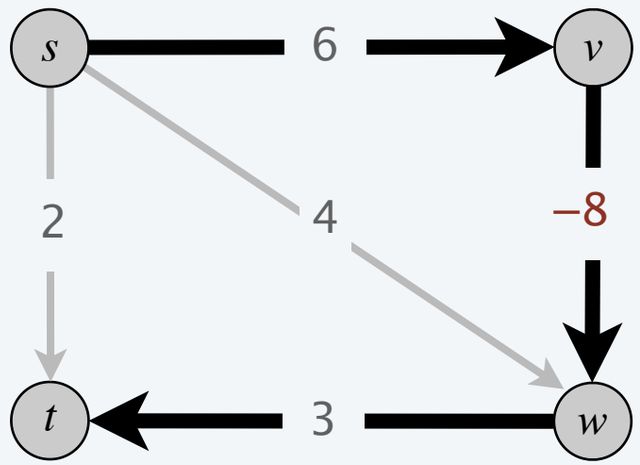

5
3
3
0
9
1
-7
1
2
-1
0
3
-4
-6
1


5
3
3
0
9
1
-7
1
2
1
3
0
-4
-6
-1
1


5
3
3
0
9
1
-7
1
2
-1
0
3
-4
-6
1

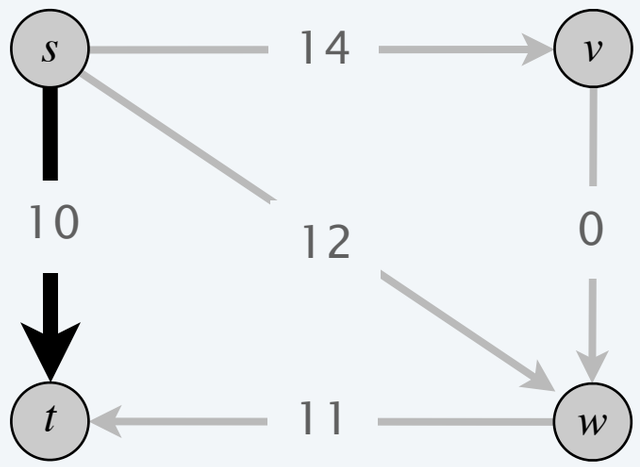
Shortest paths with negative weights: failed attempts

Dijkstra. May not produce shortest paths when edge lengths are negative.



Dijkstra selects the vertices in the order s, t, w, v
But shortest path from s to t is $s \rightarrow v \rightarrow w \rightarrow t$.

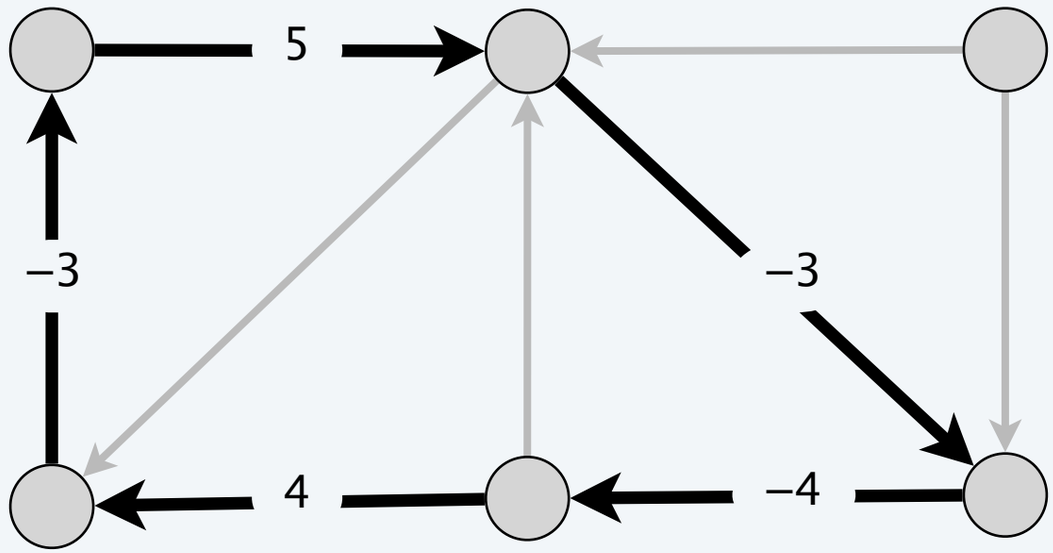
Reweighting. Adding a constant to every edge length does not necessarily make Dijkstra's algorithm produce shortest paths.



Adding 8 to each edge weight changes the shortest path from $s \rightarrow v \rightarrow w \rightarrow t$ to $s \rightarrow t$.

Negative cycles

Def. A negative cycle is a directed cycle for which the sum of its edge lengths is negative.

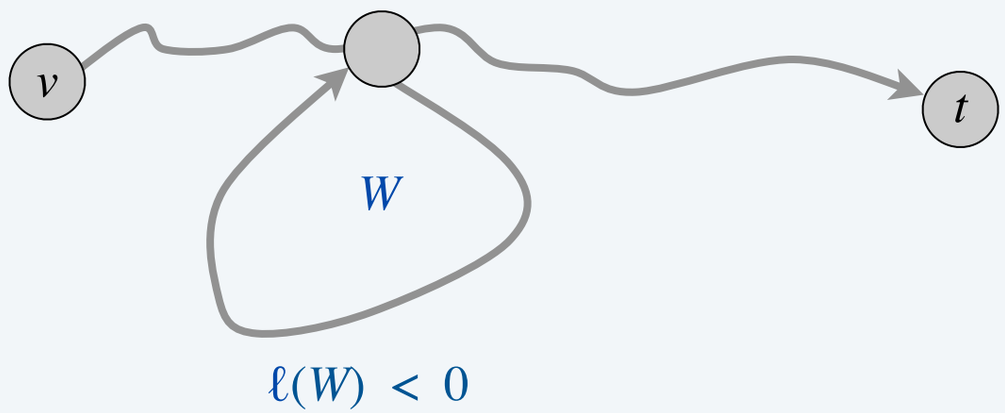


a negative cycle W : $\ell(W) = \sum_{e \in W} \ell_e < 0$

Shortest paths and negative cycles

Lemma 1. If some $v \rightsquigarrow t$ path contains a negative cycle, then there does not exist a shortest $v \rightsquigarrow t$ path.

Pf. If there exists such a cycle W, then can build a $v \rightsquigarrow t$ path of arbitrarily negative length by detouring around W as many times as desired. ▪

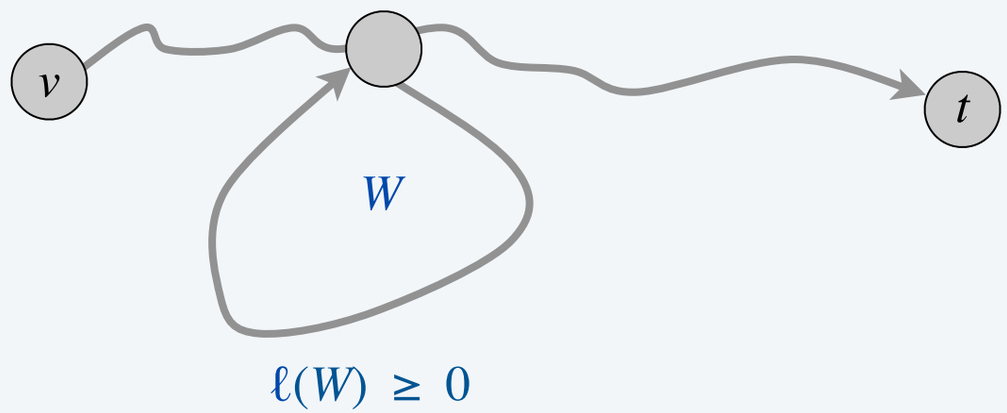


Shortest paths and negative cycles

Lemma 2. If G has no negative cycles, then there exists a shortest $v \rightsquigarrow t$ path that is simple (and has $\leq n - 1$ edges).

Pf.

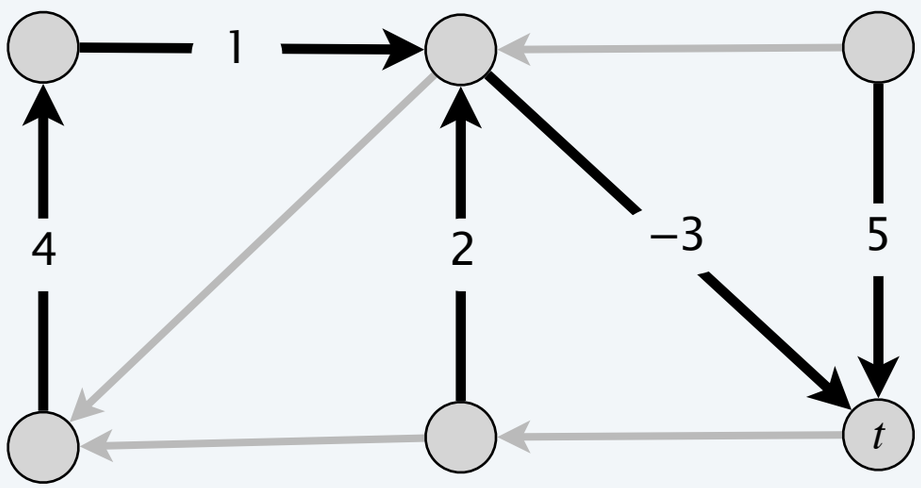
- Among all shortest $v \rightsquigarrow t$ paths, consider one that uses the fewest edges.
- If that path P contains a directed cycle W, can remove the portion of P corresponding to W without increasing its length. ▪



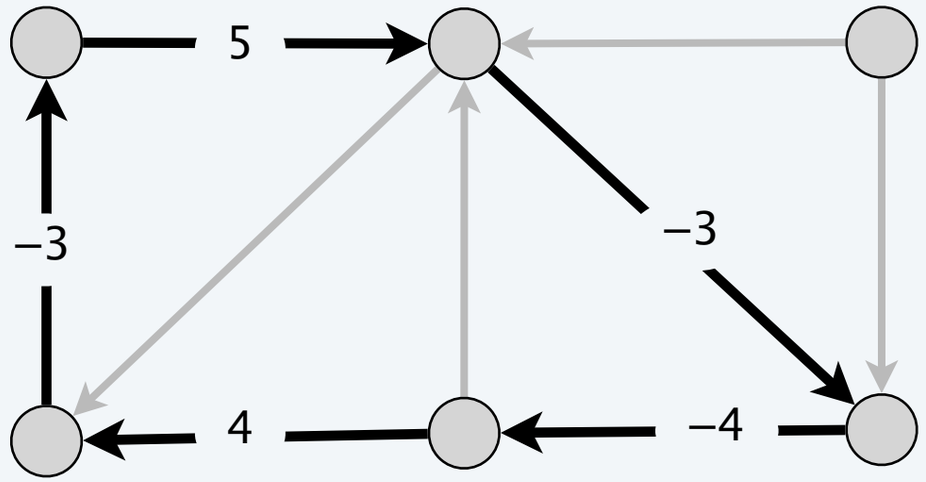
Shortest-paths and negative-cycle problems

Single-destination shortest-paths problem. Given a digraph $G = (V, E)$ with edge lengths ℓ_{vw} (but no negative cycles) and a distinguished node t, find a shortest $v \rightsquigarrow t$ path for every node v.

Negative-cycle problem. Given a digraph $G = (V, E)$ with edge lengths ℓ_{vw}, find a negative cycle (if one exists).

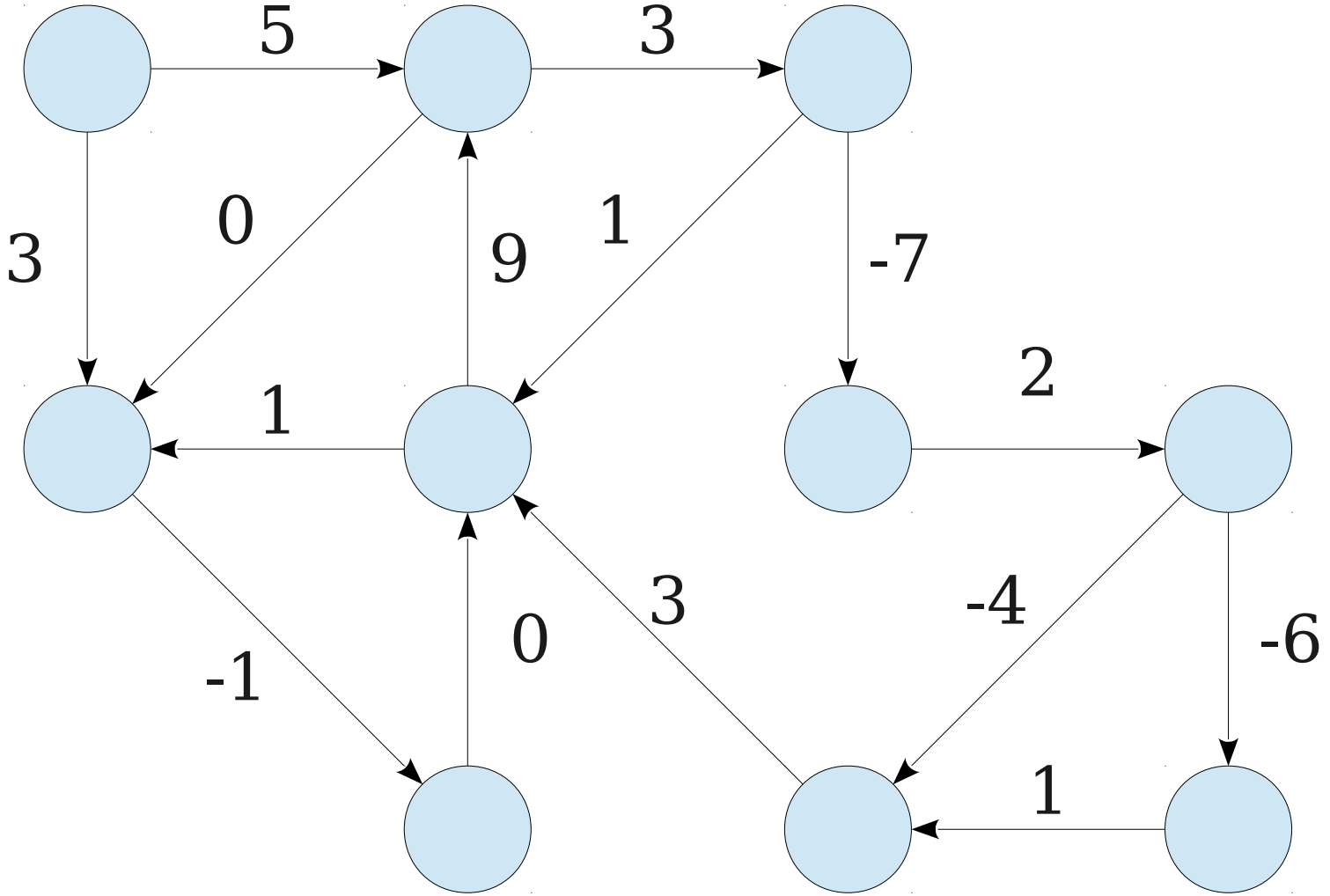


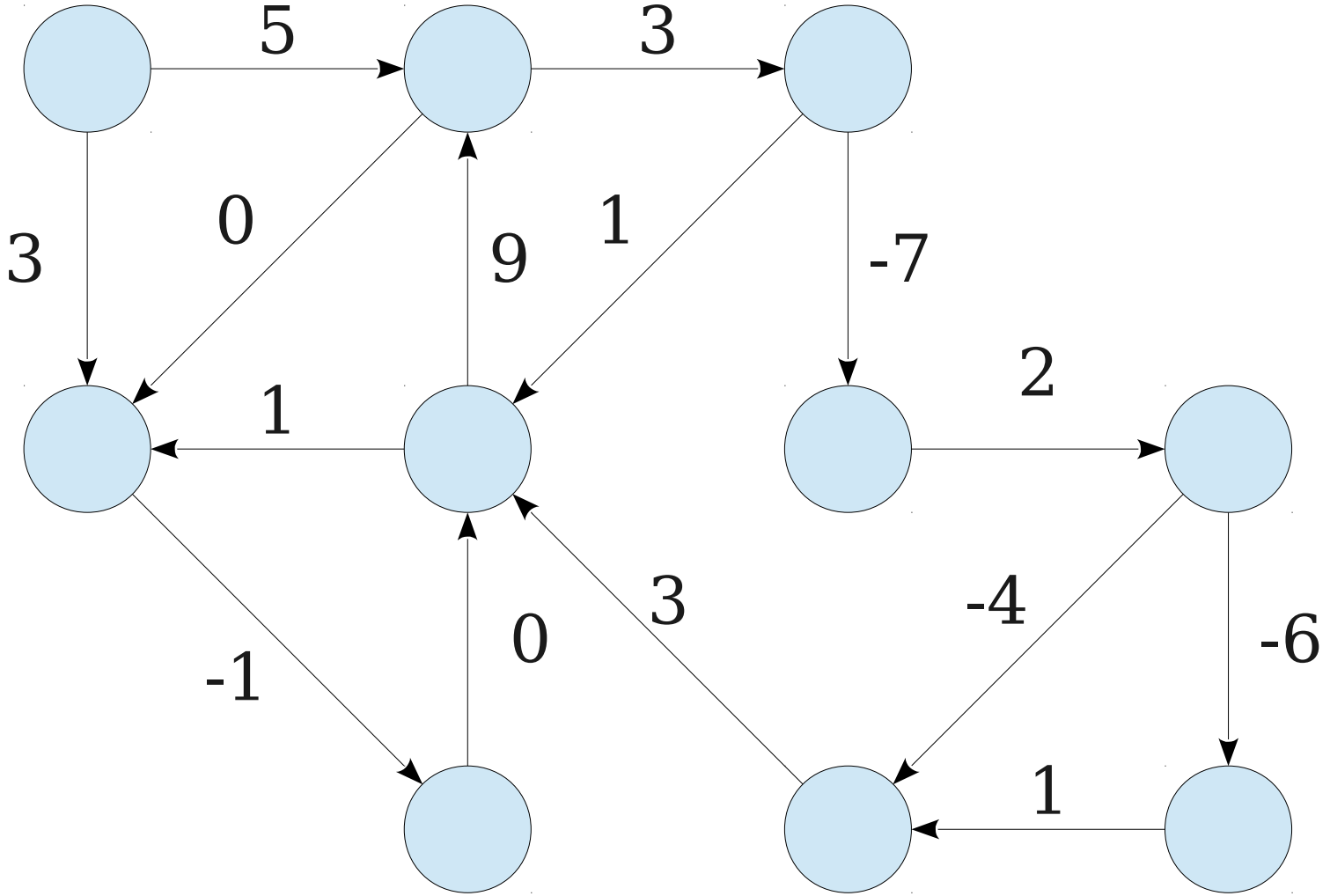
shortest-paths tree



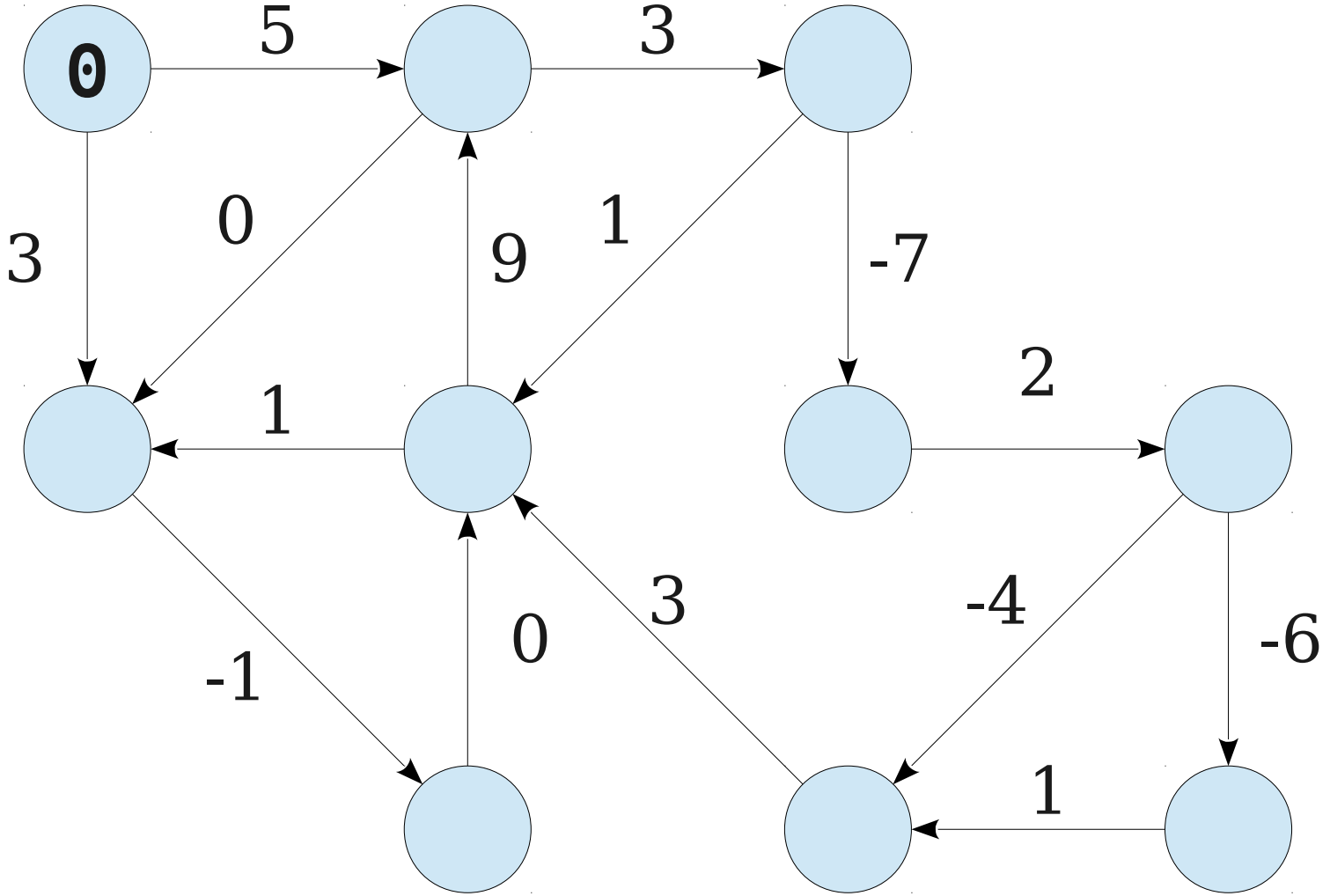
negative cycle

A Different Intuition

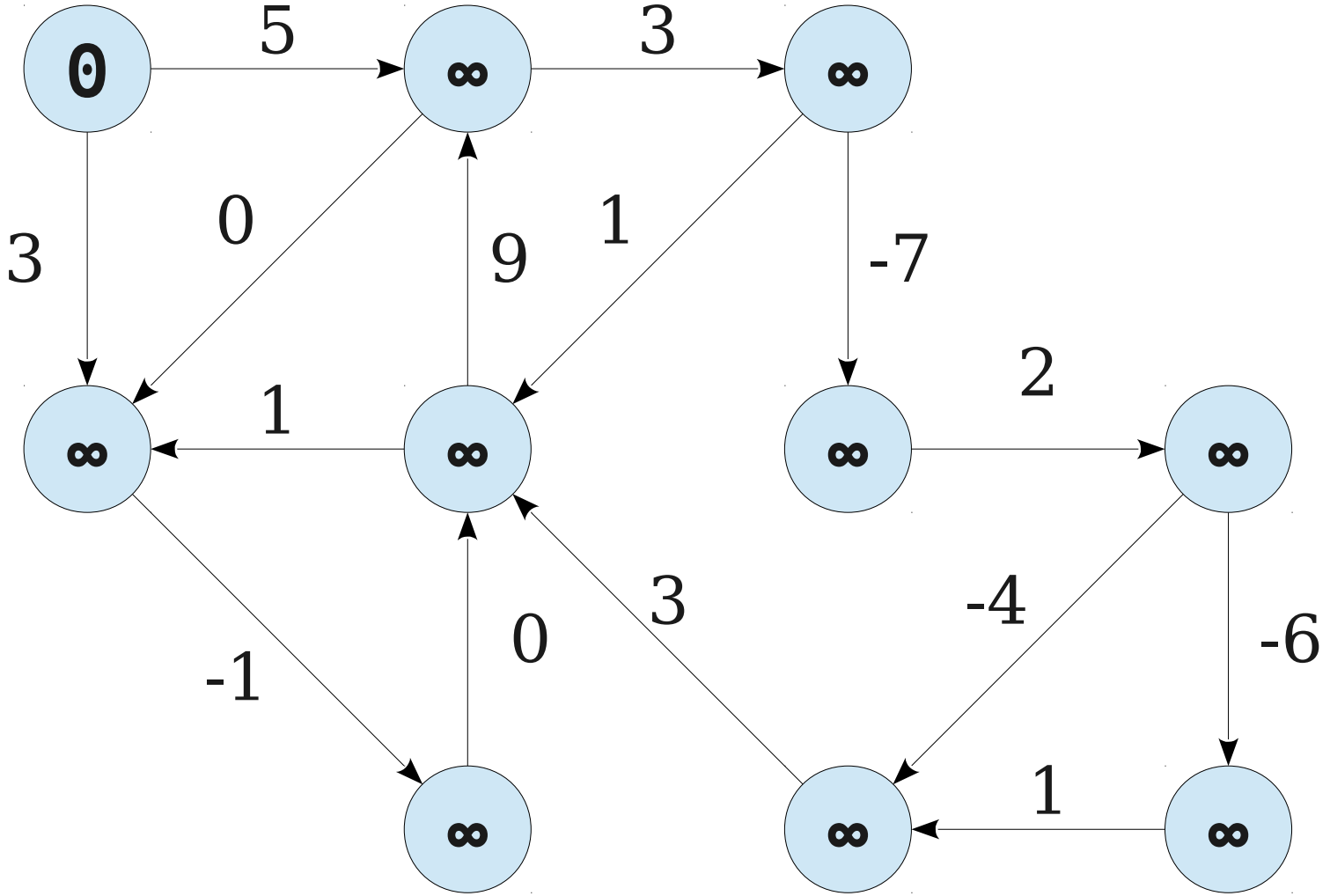

5
3
3
0
9
1
-7
1
2
-1
0
3
-4
-6
1



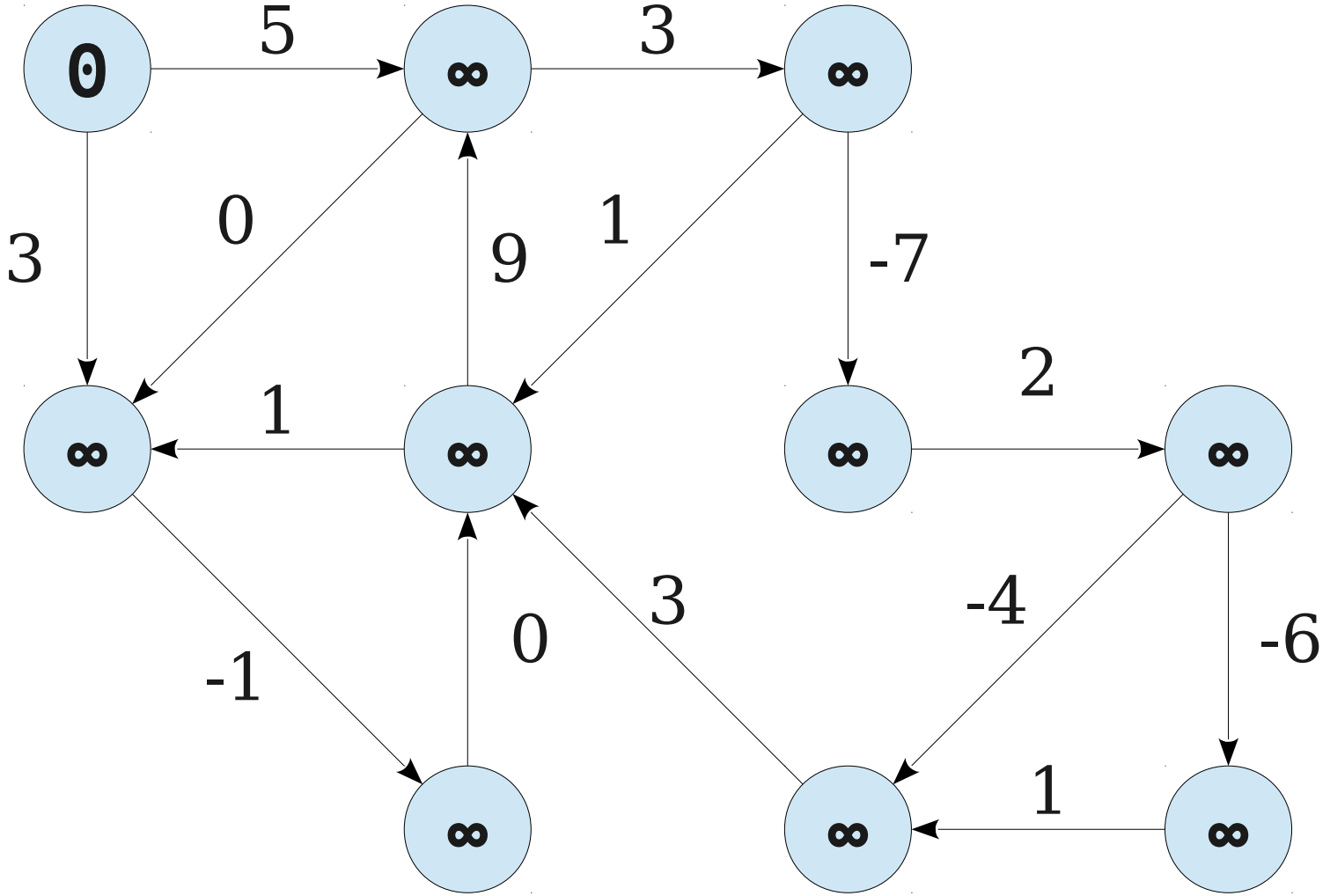
Consider paths of length **0**.



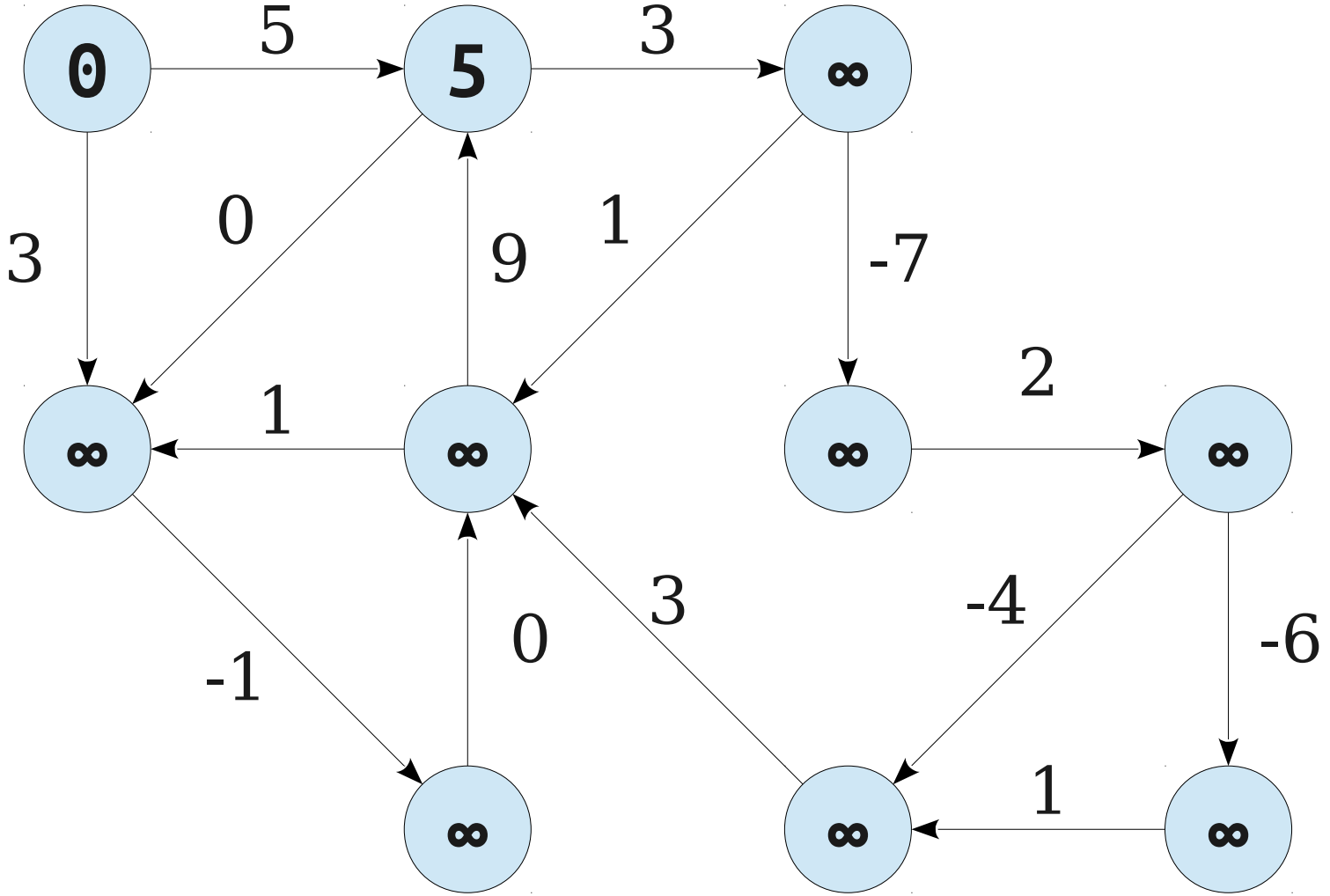
Consider paths of length **0**.



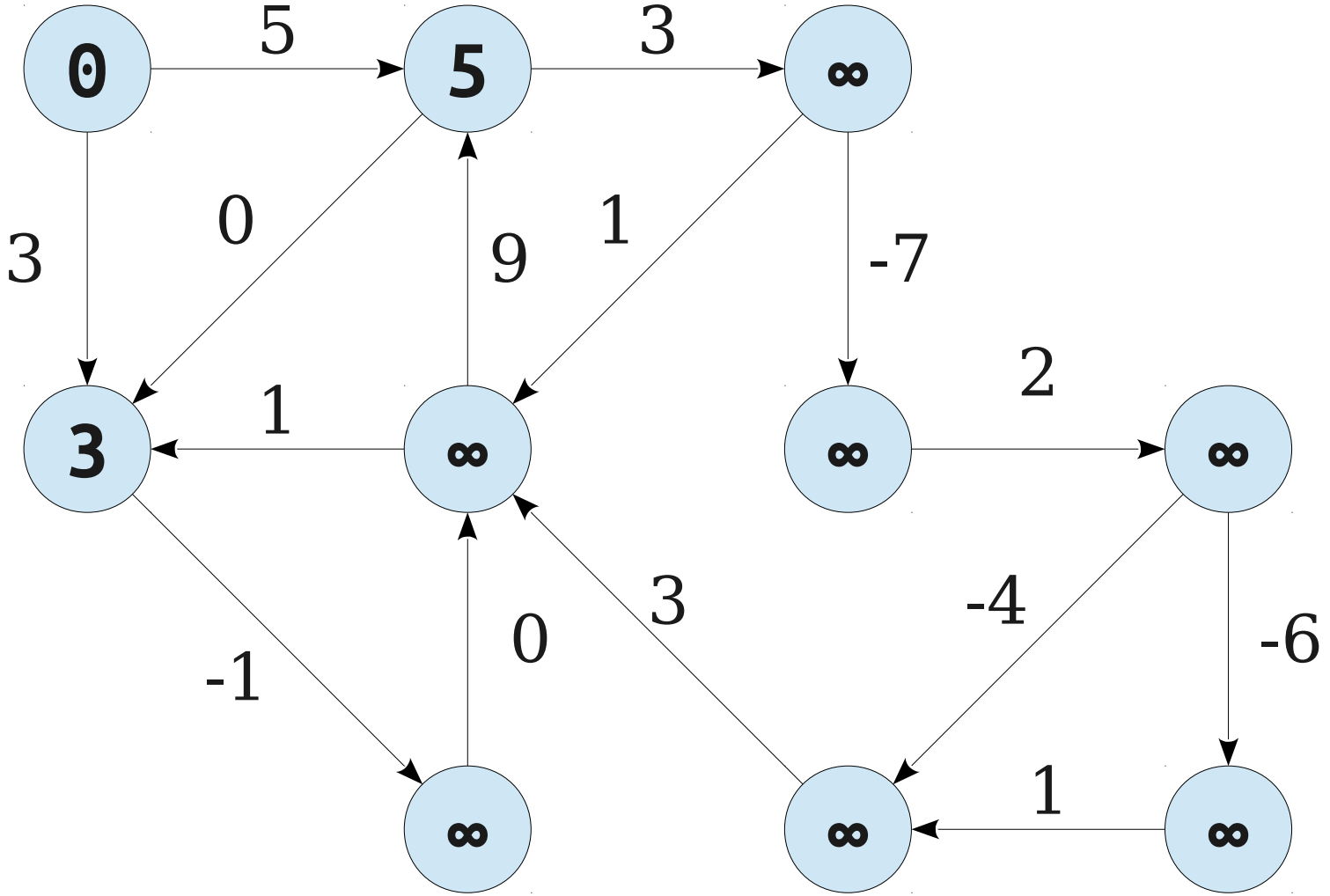
Consider paths of length **0**.



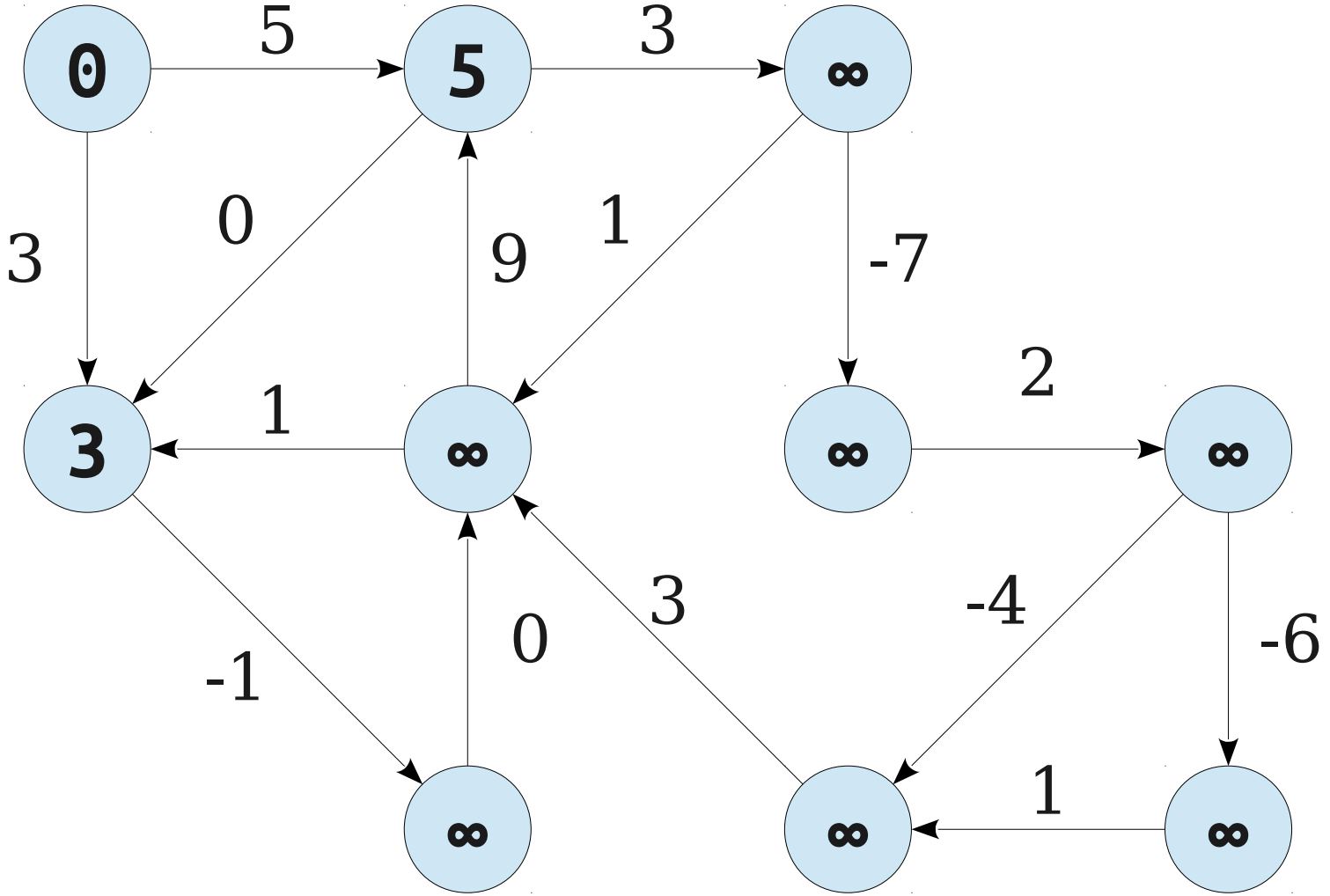
Consider paths of length **1**.



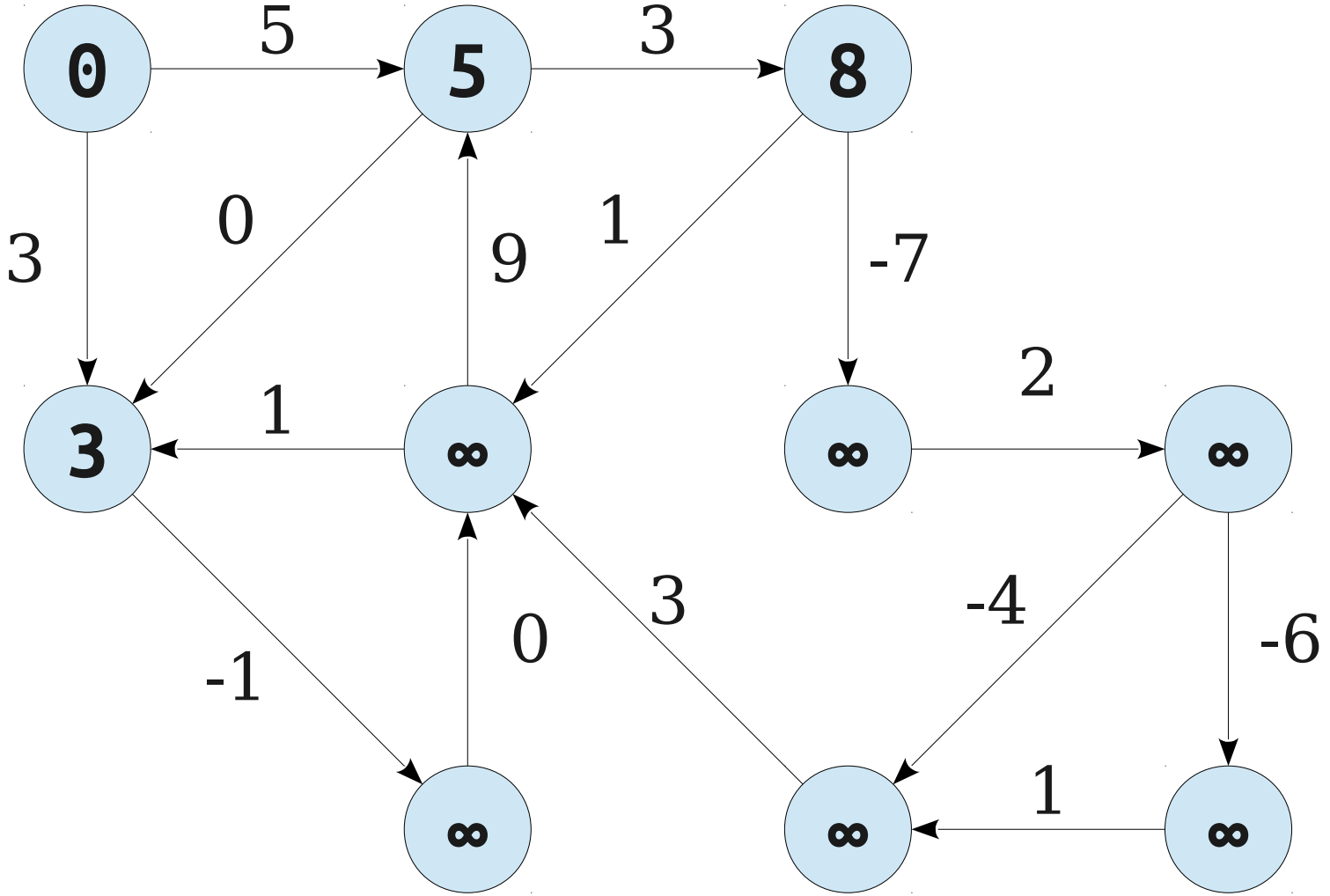
Consider paths of length **1**.



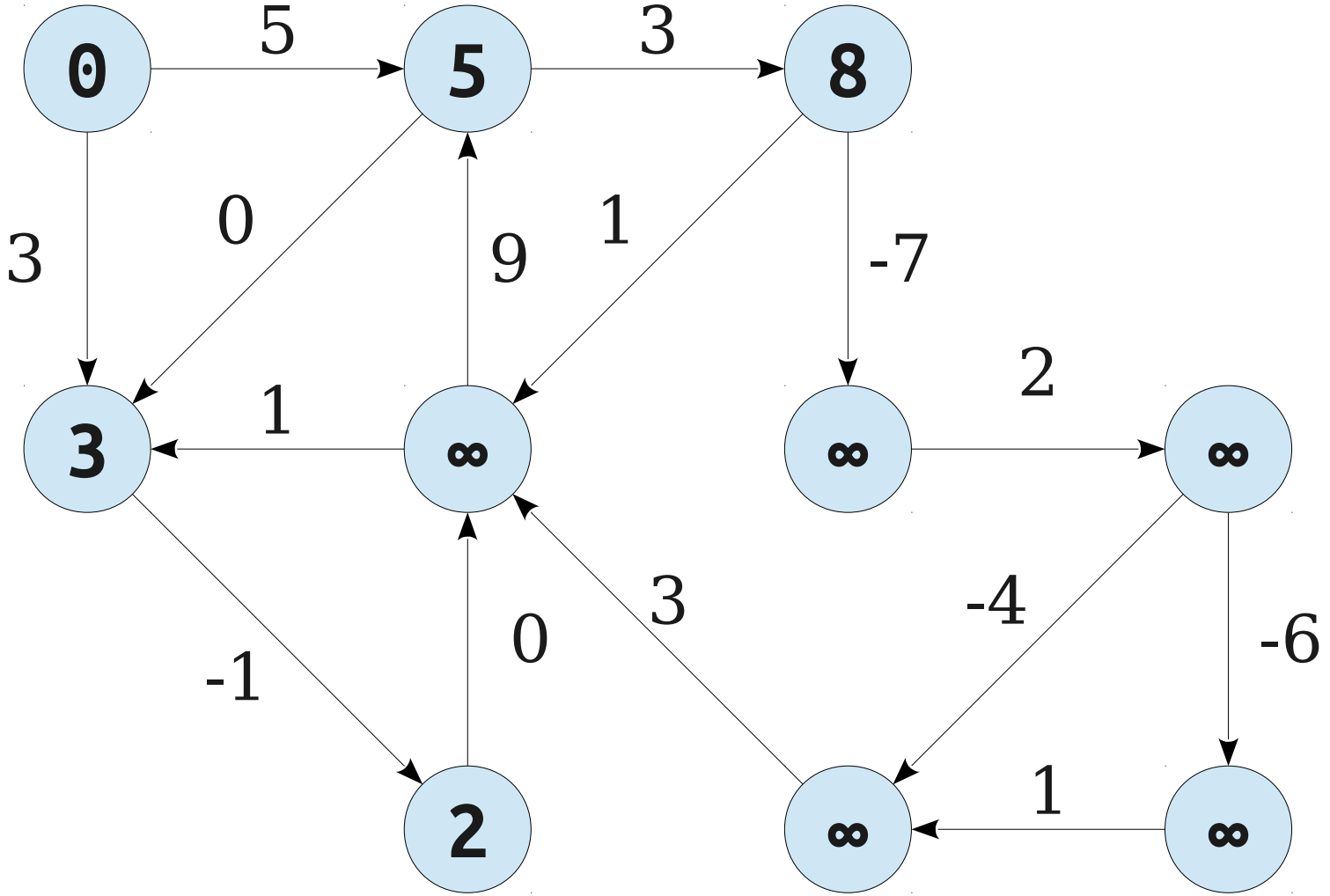
Consider paths of length **1**.



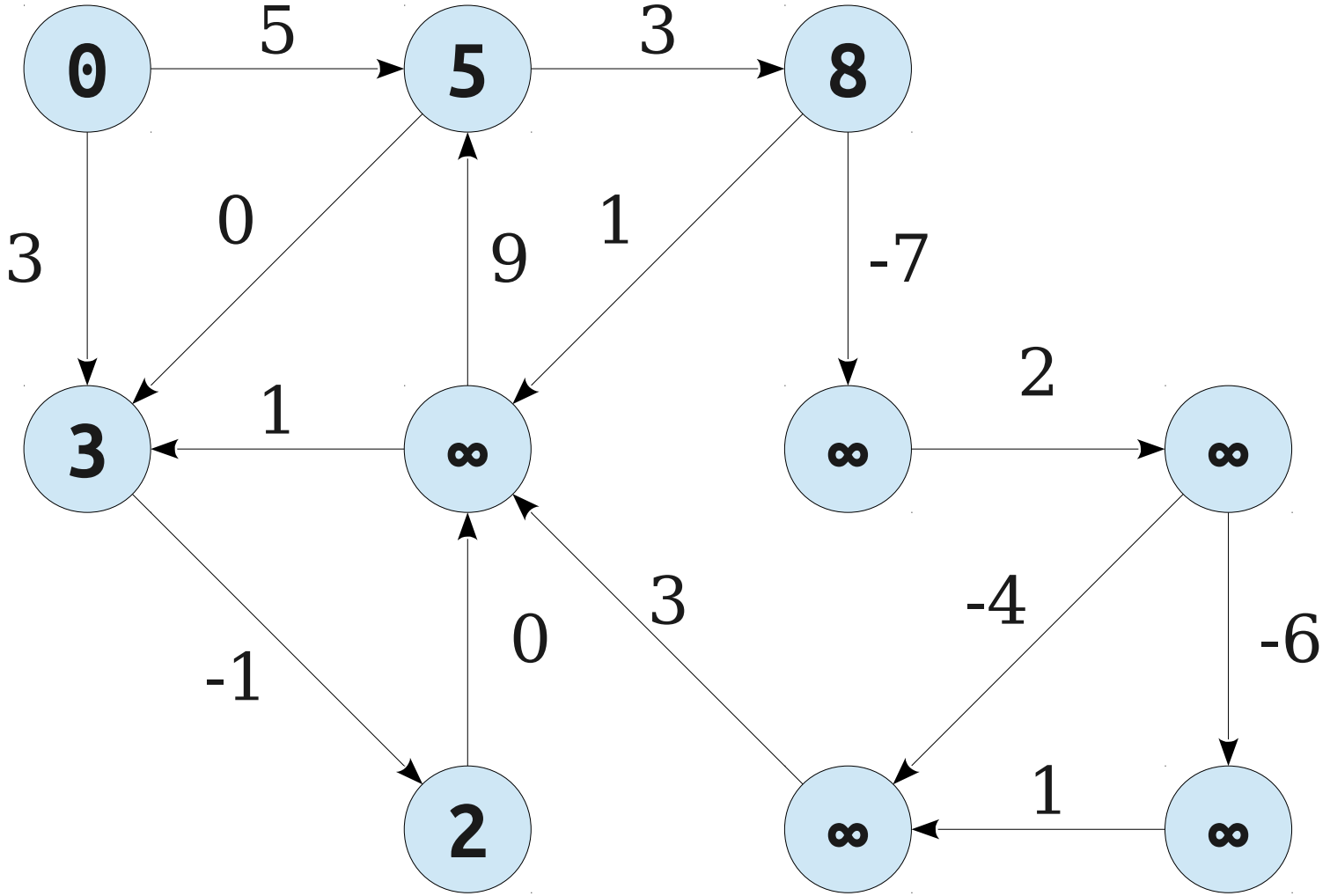
Consider paths of length **2**.



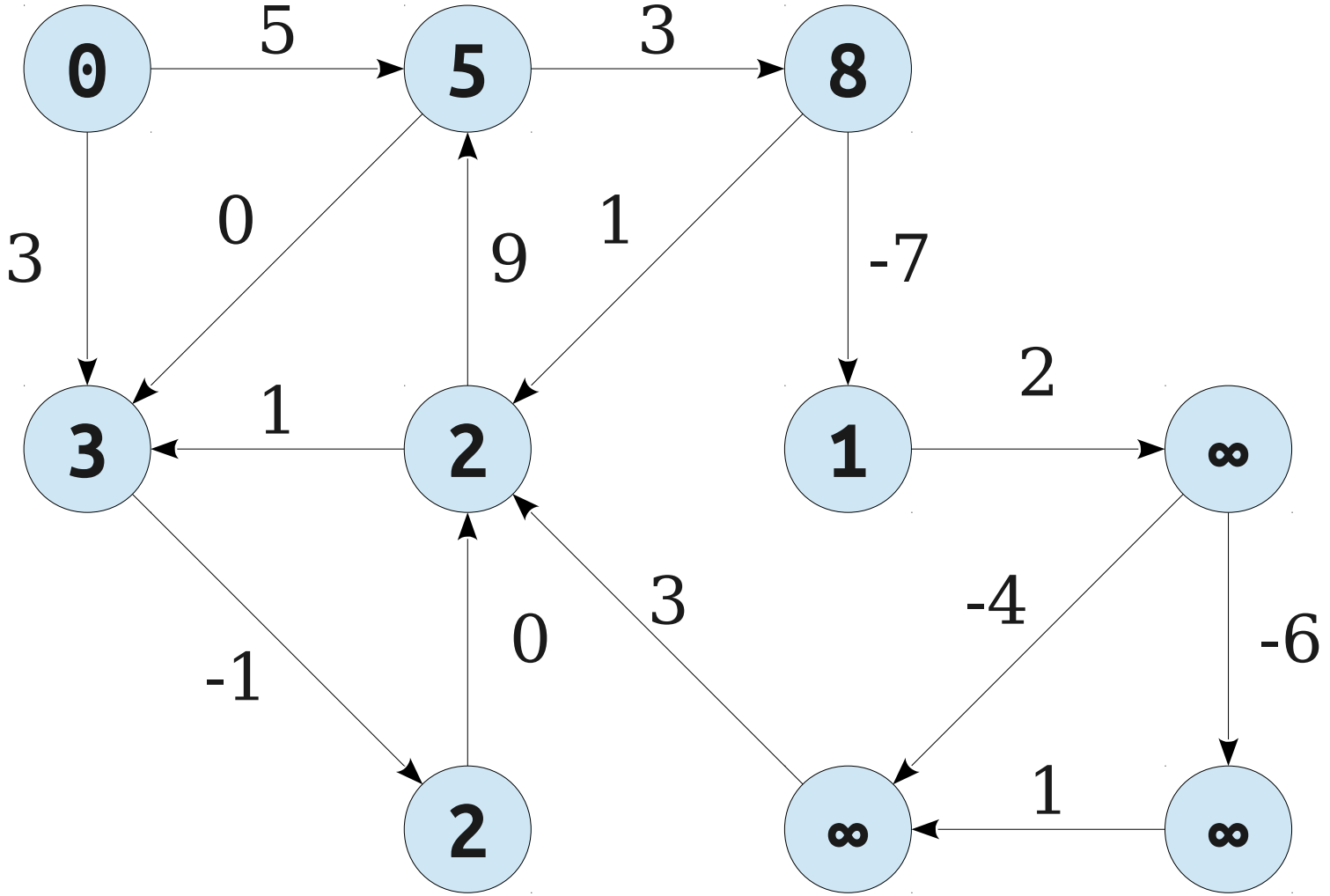
Consider paths of length **2**.



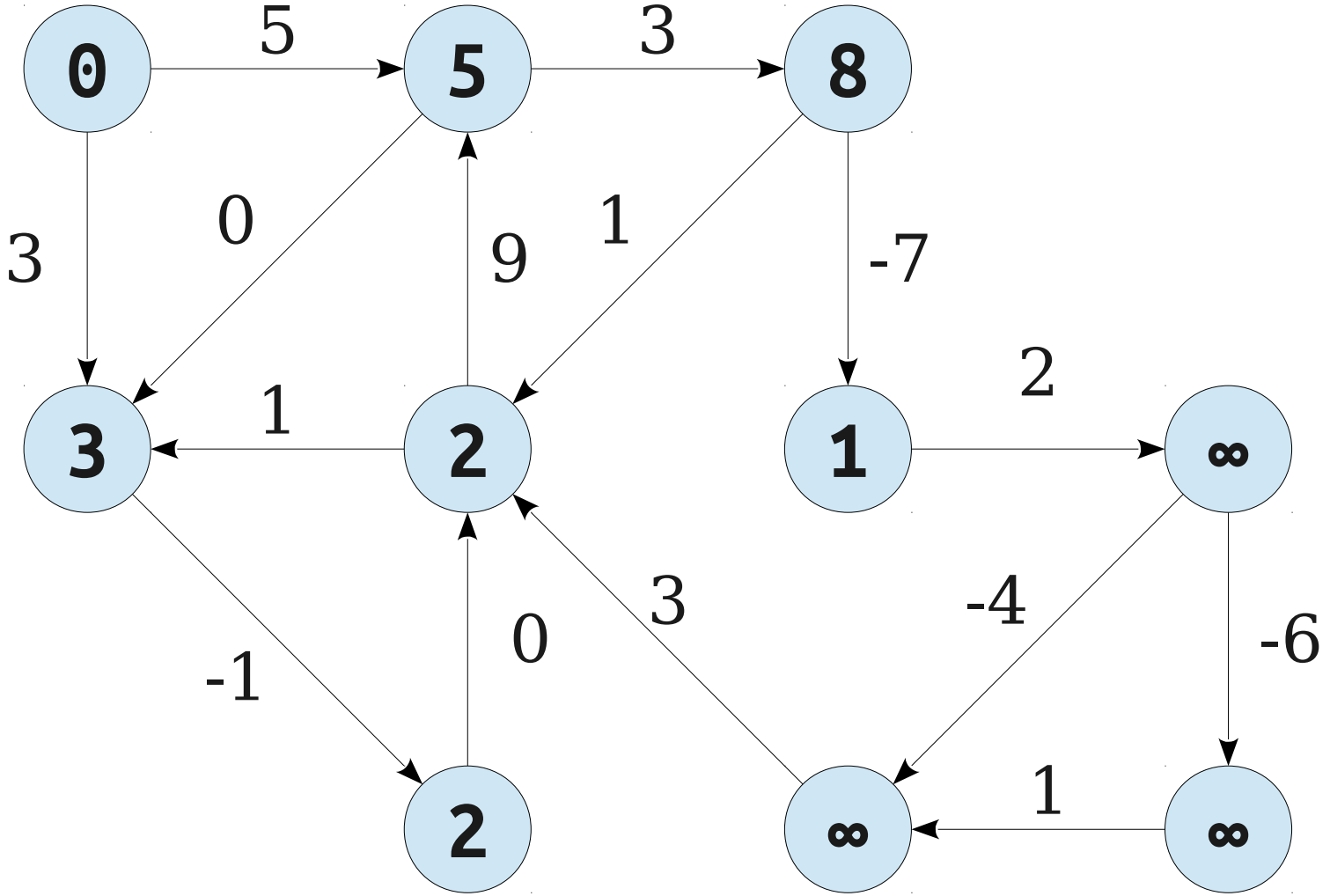
Consider paths of length **2**.



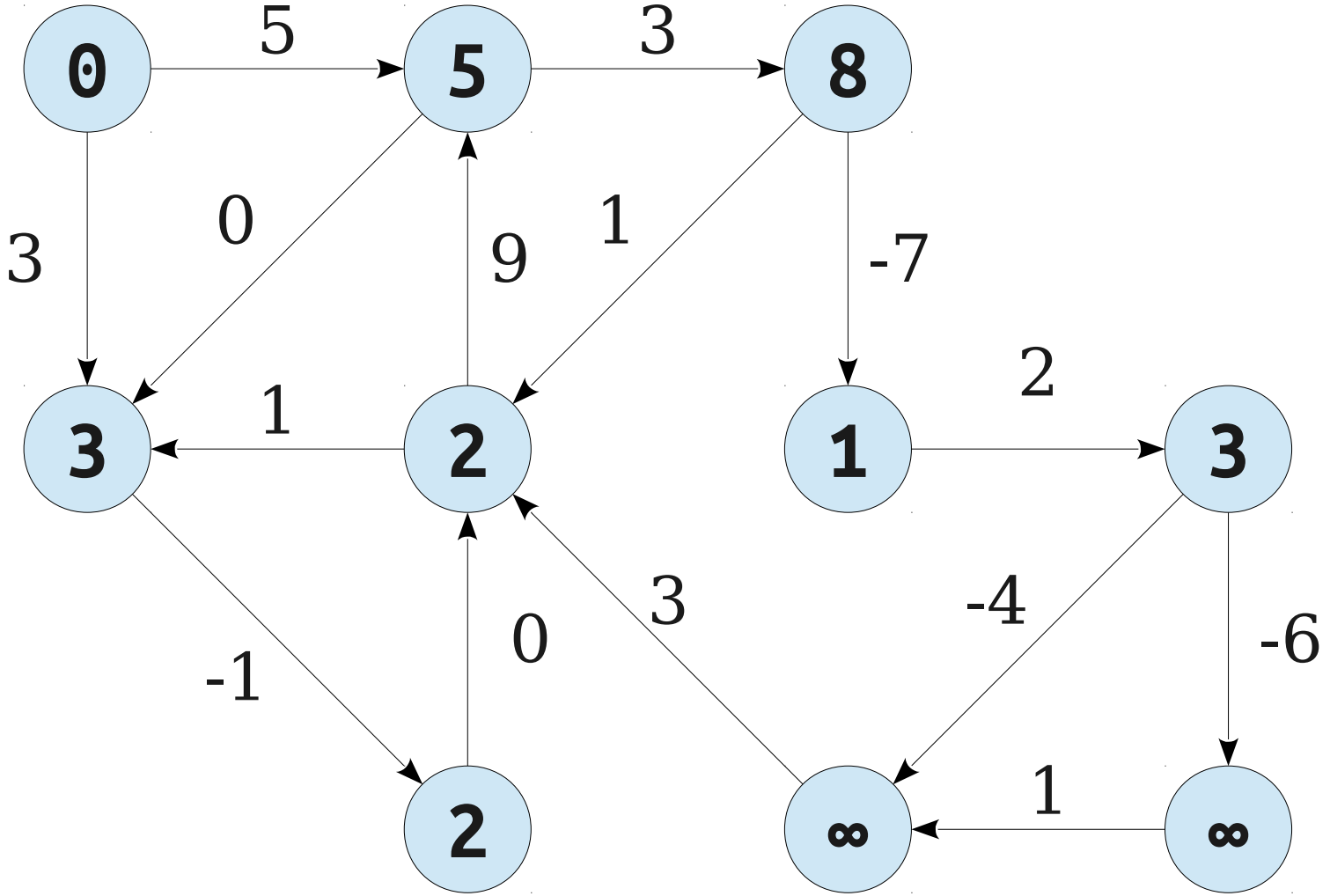
Consider paths of length **3**.



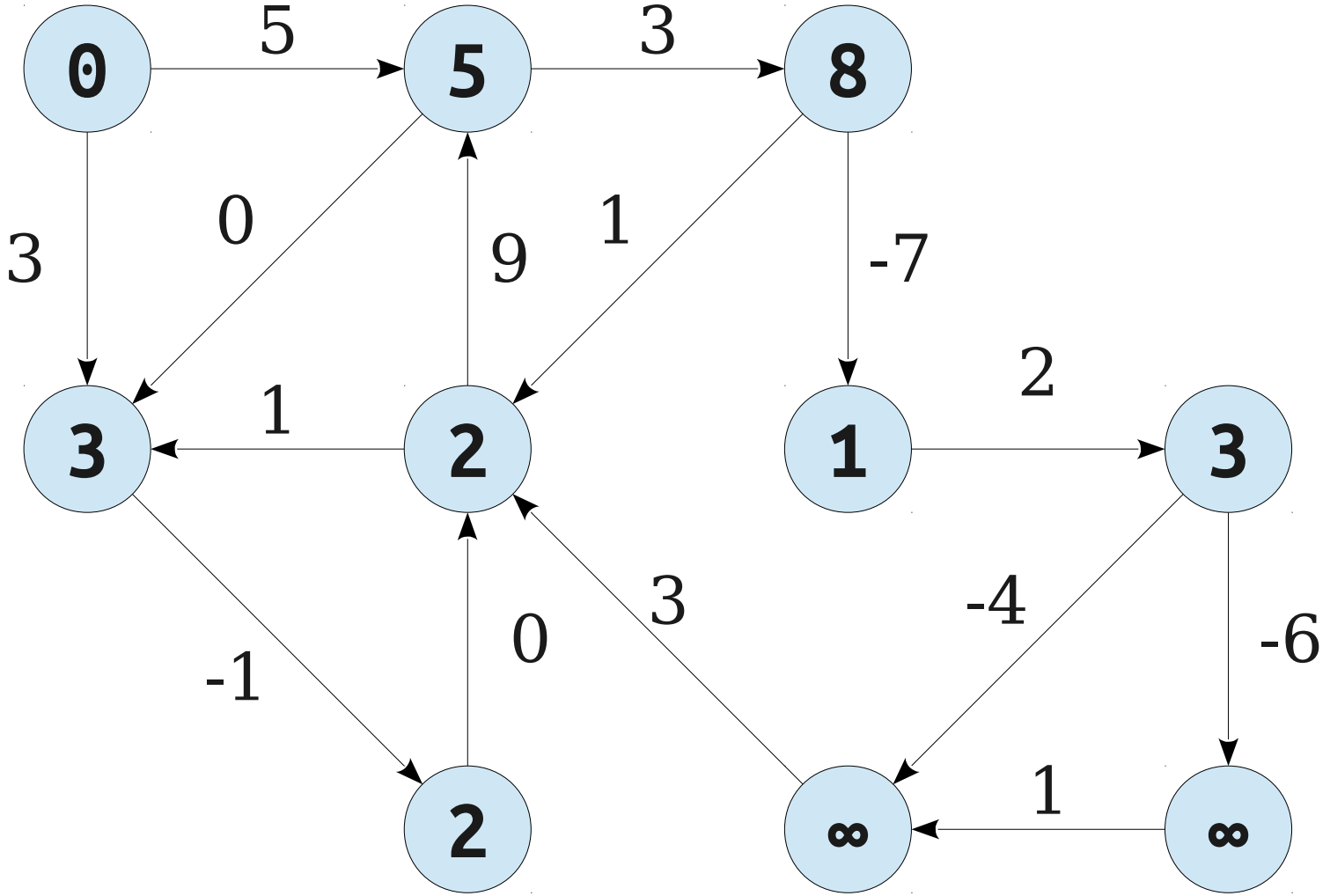
Consider paths of length **3**.



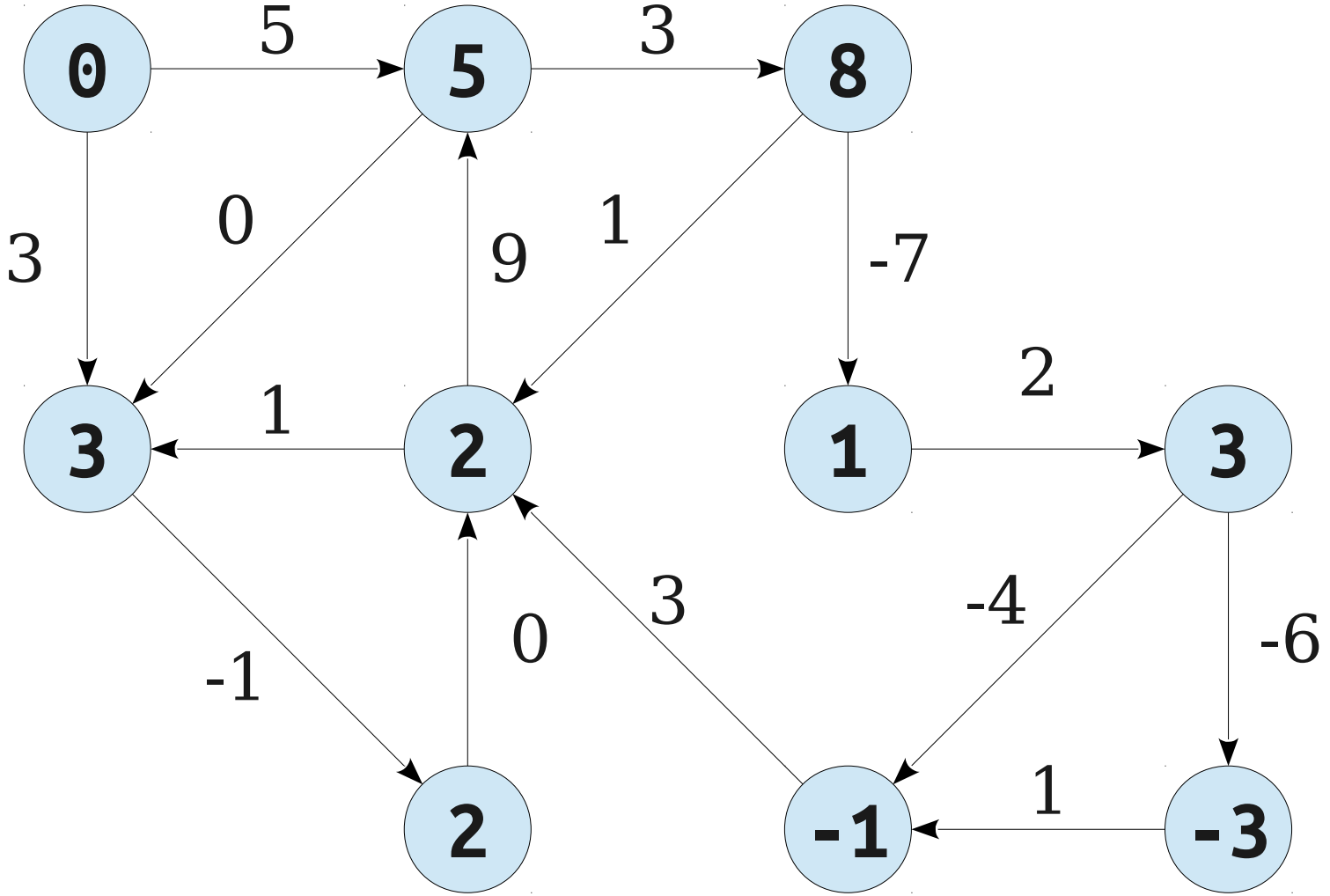
Consider paths of length **4**.



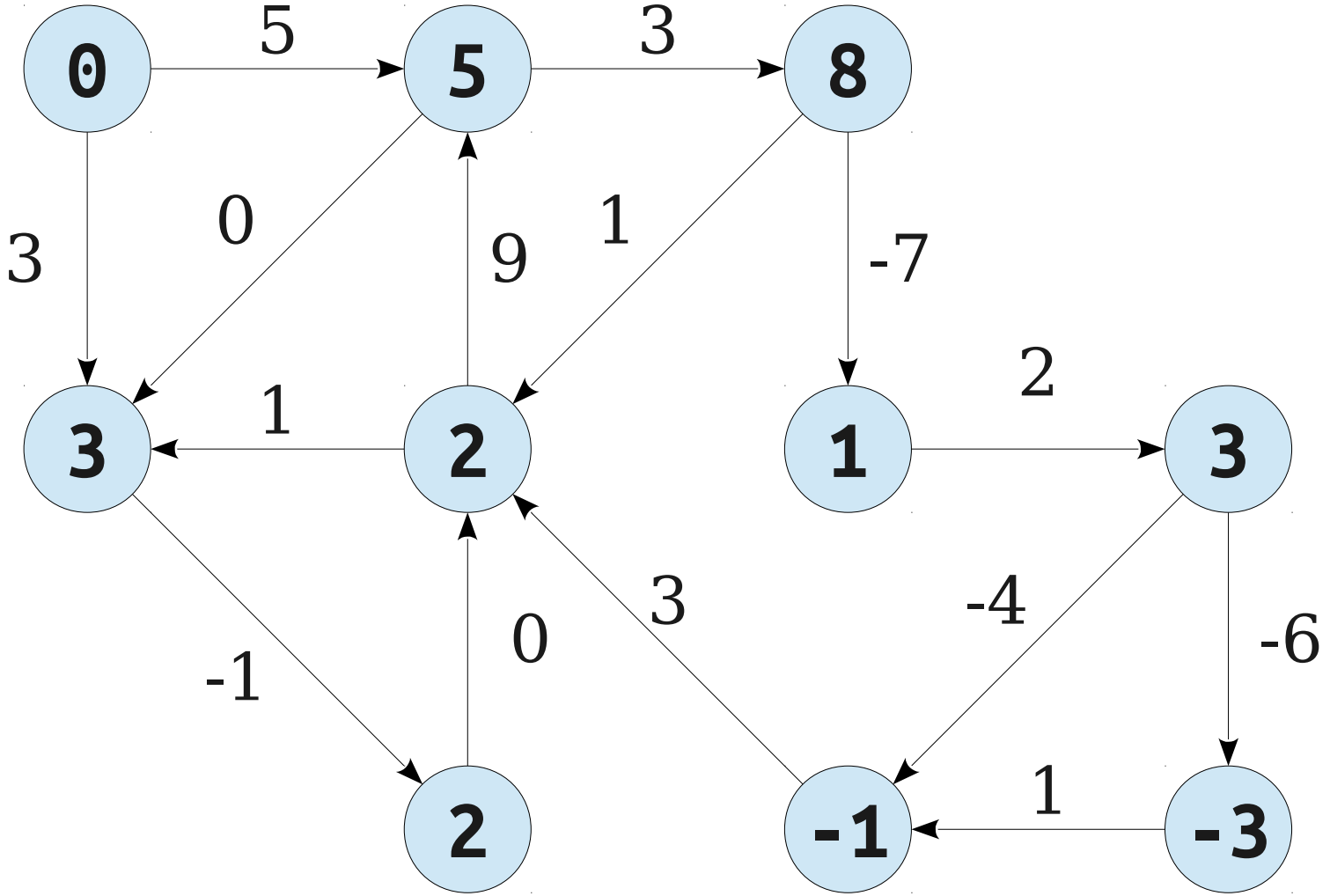
Consider paths of length **4**.



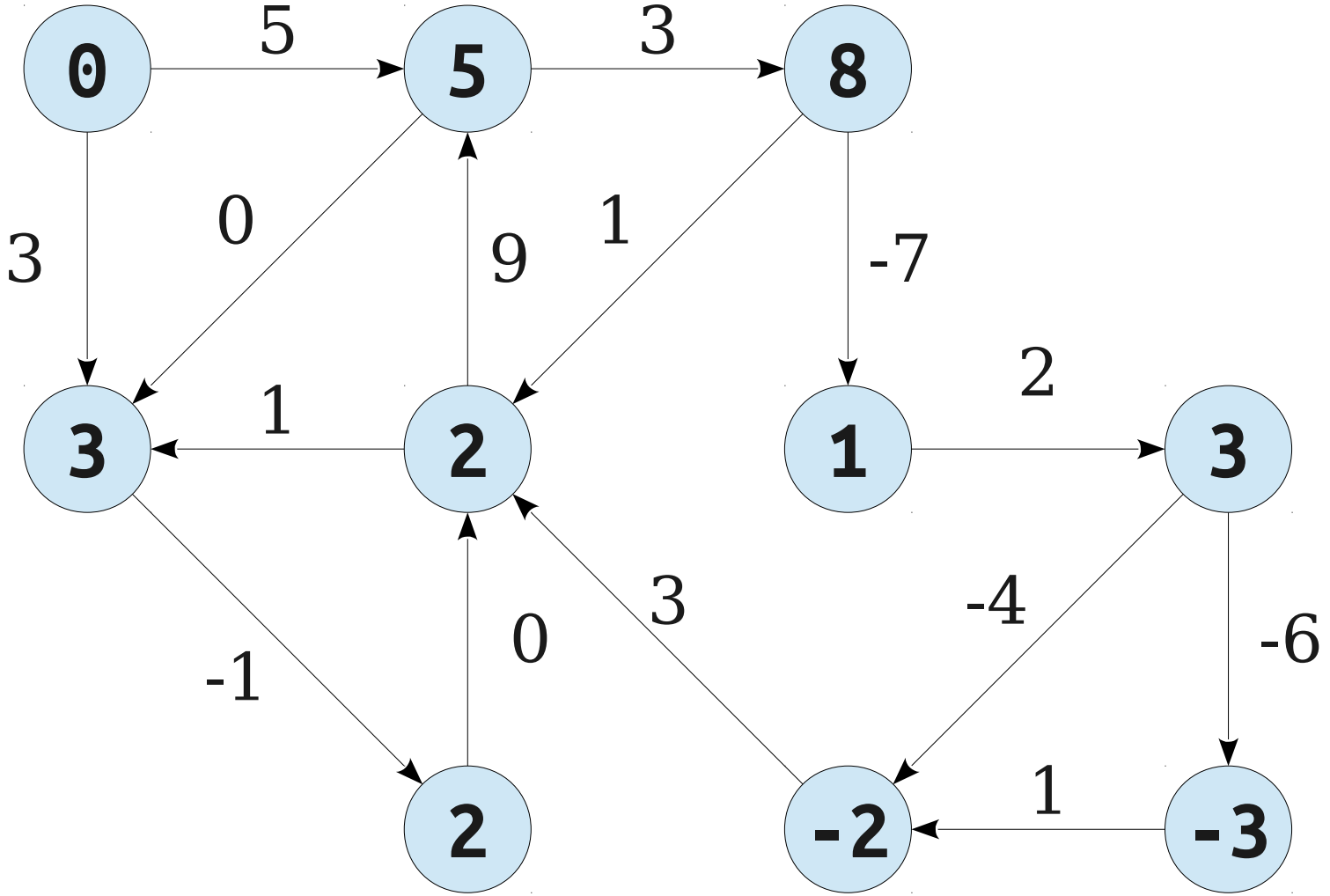
Consider paths of length **5**.



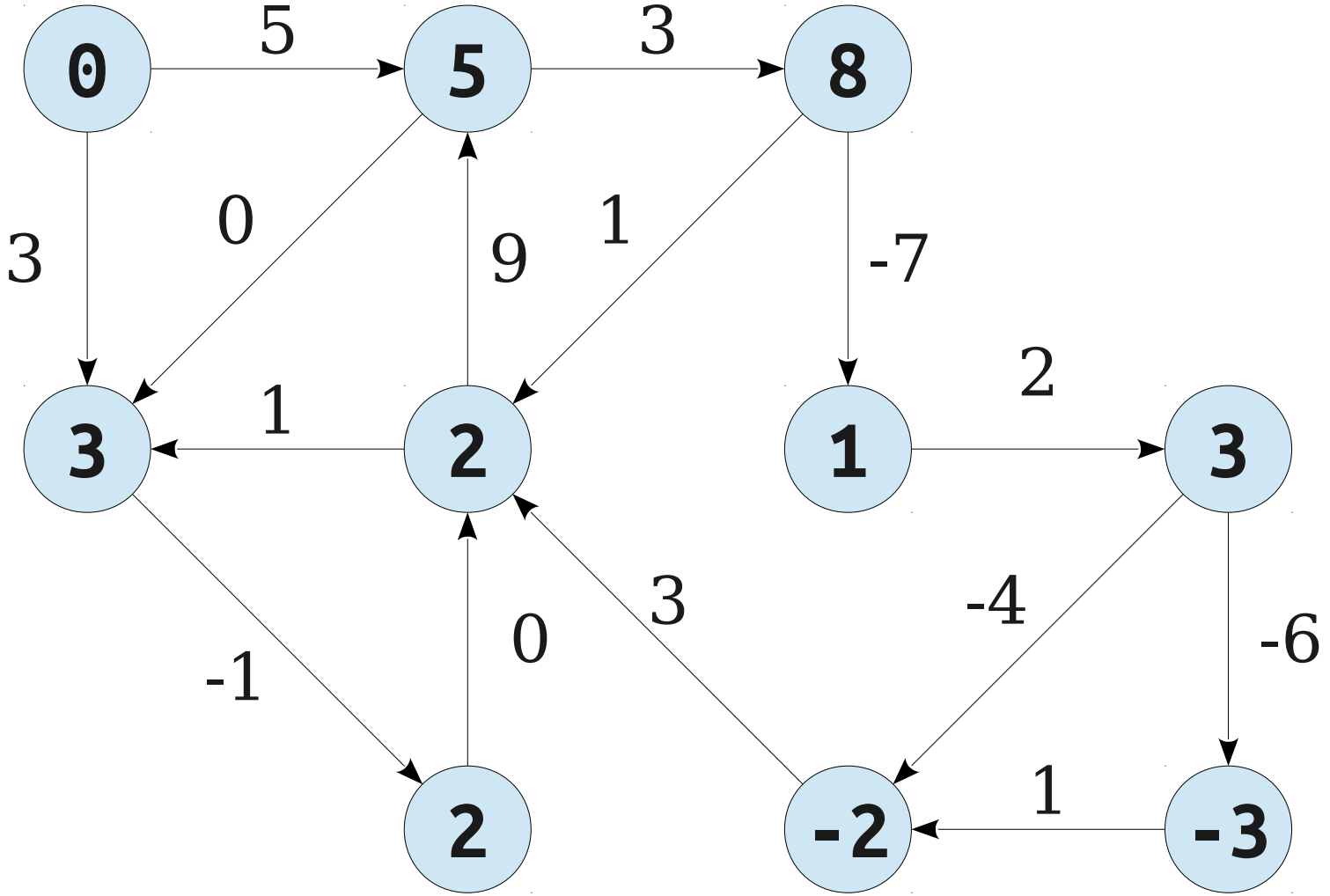
Consider paths of length **5**.



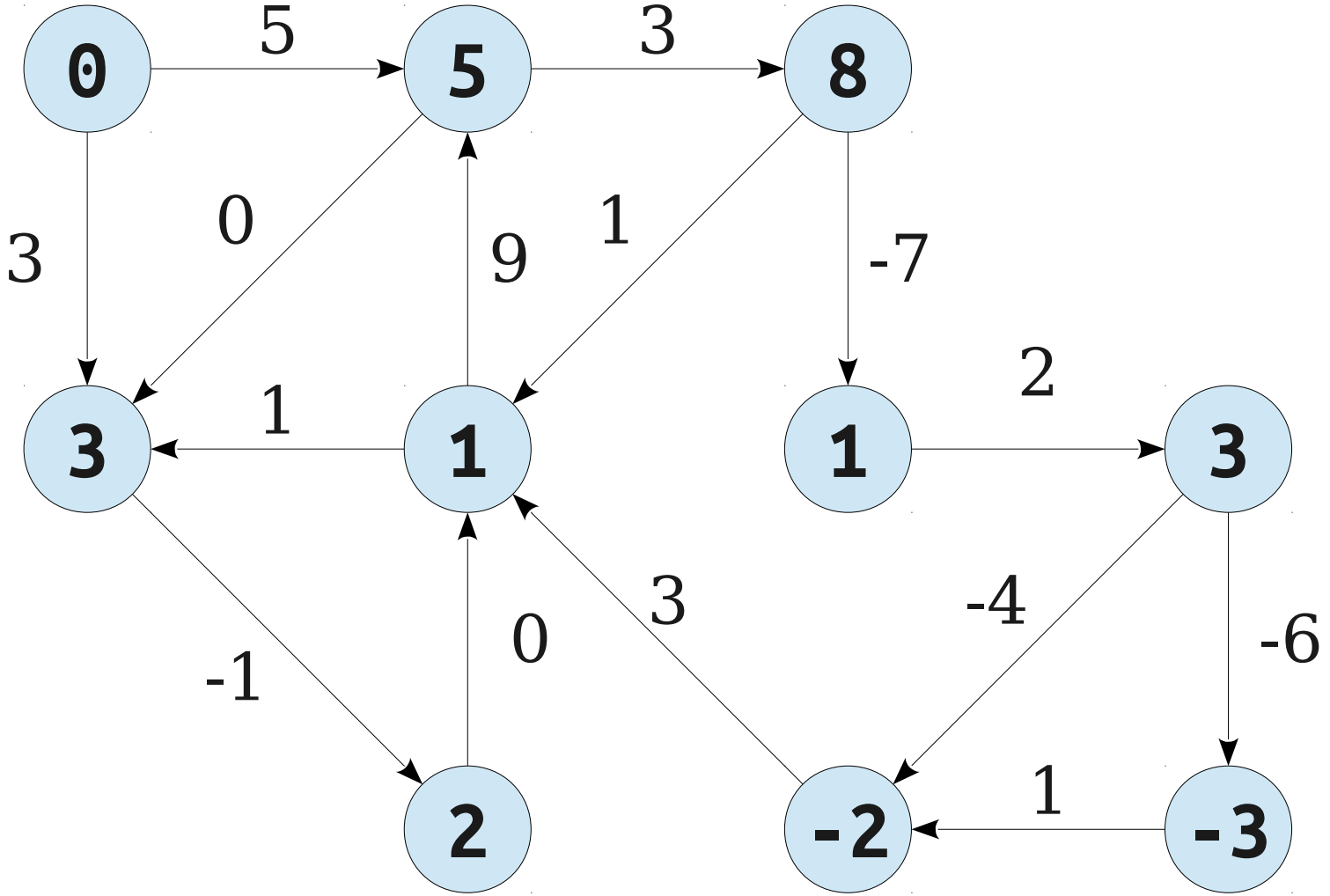
Consider paths of length **6**.



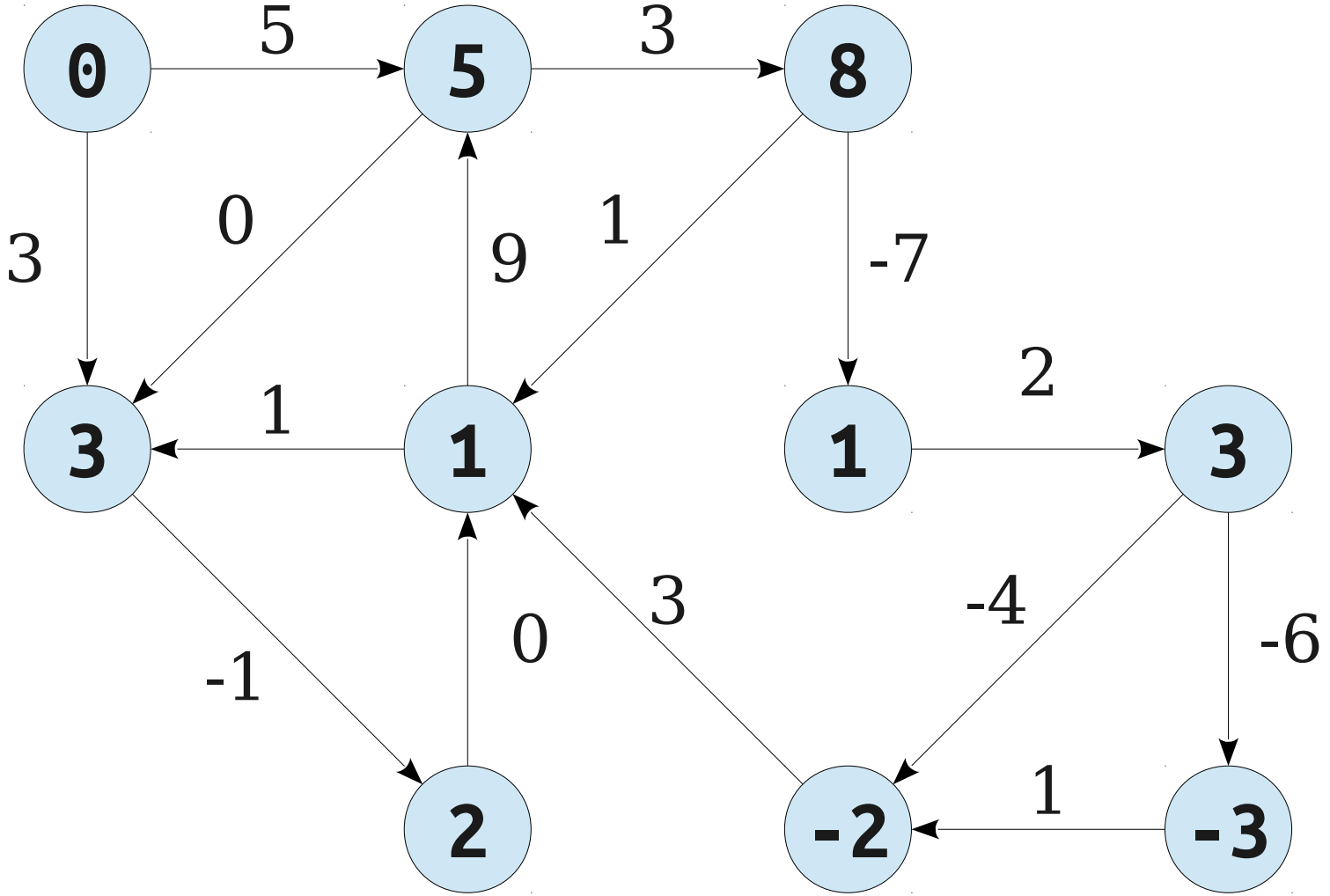
Consider paths of length **6**.



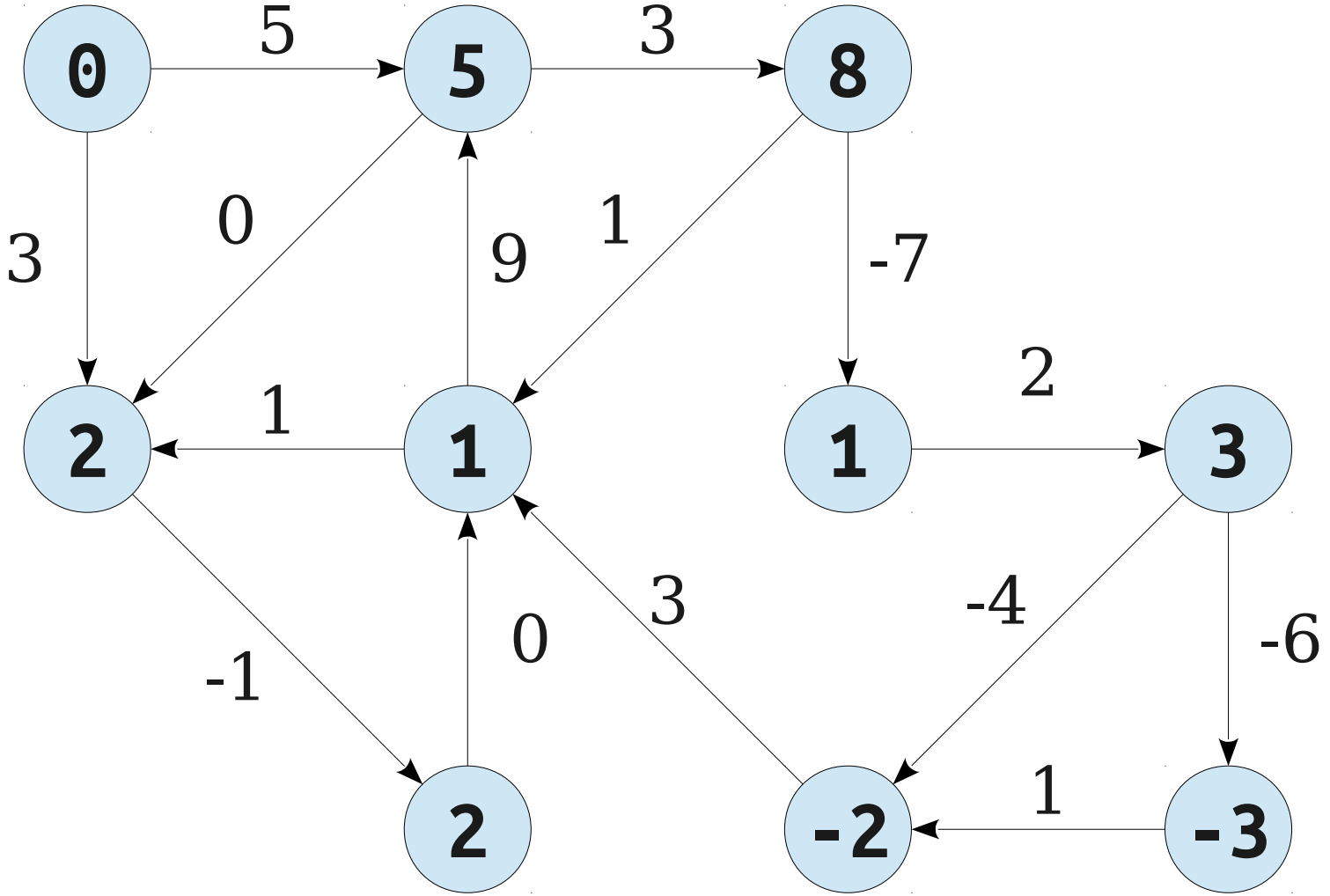
Consider paths of length 7.



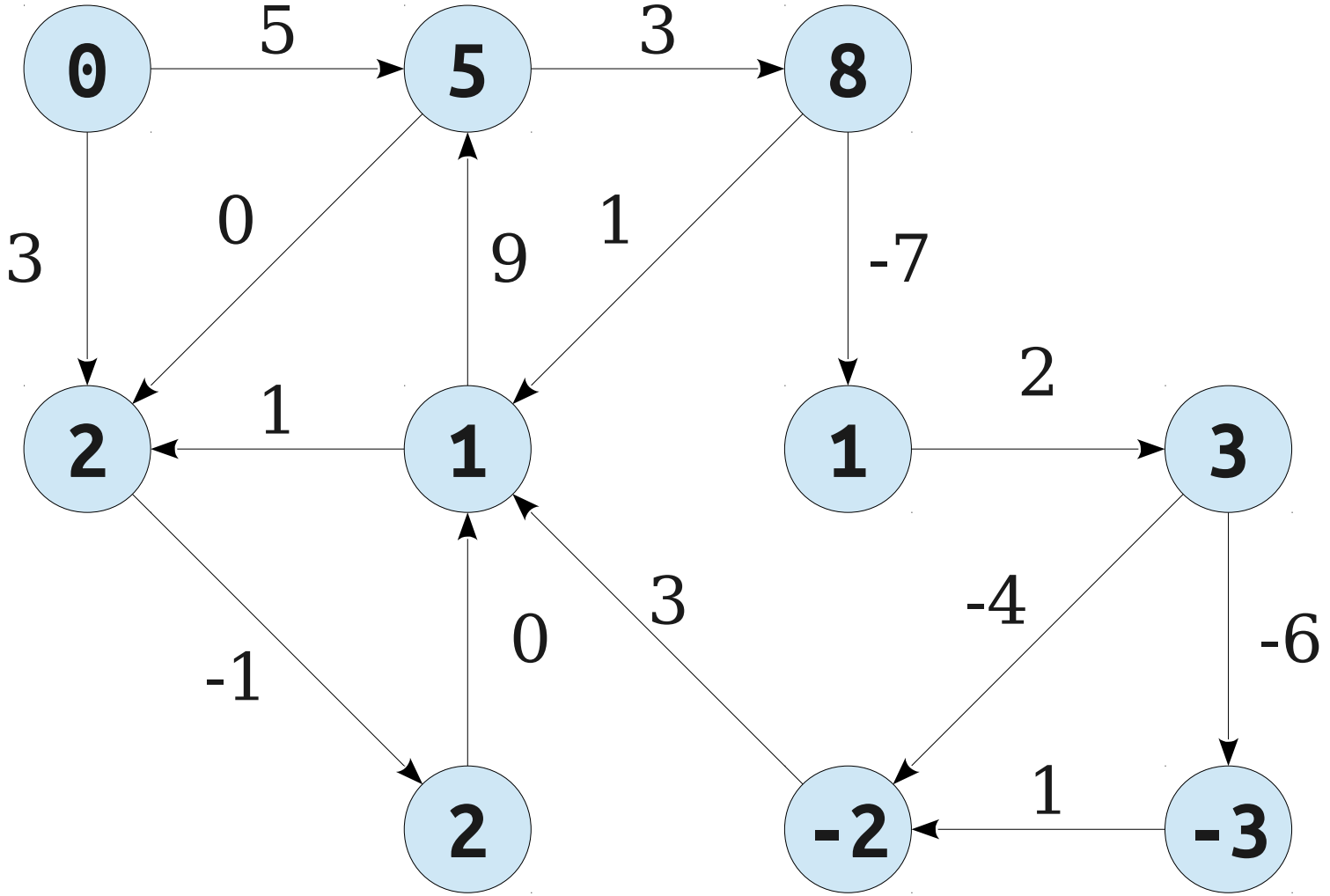
Consider paths of length 7.



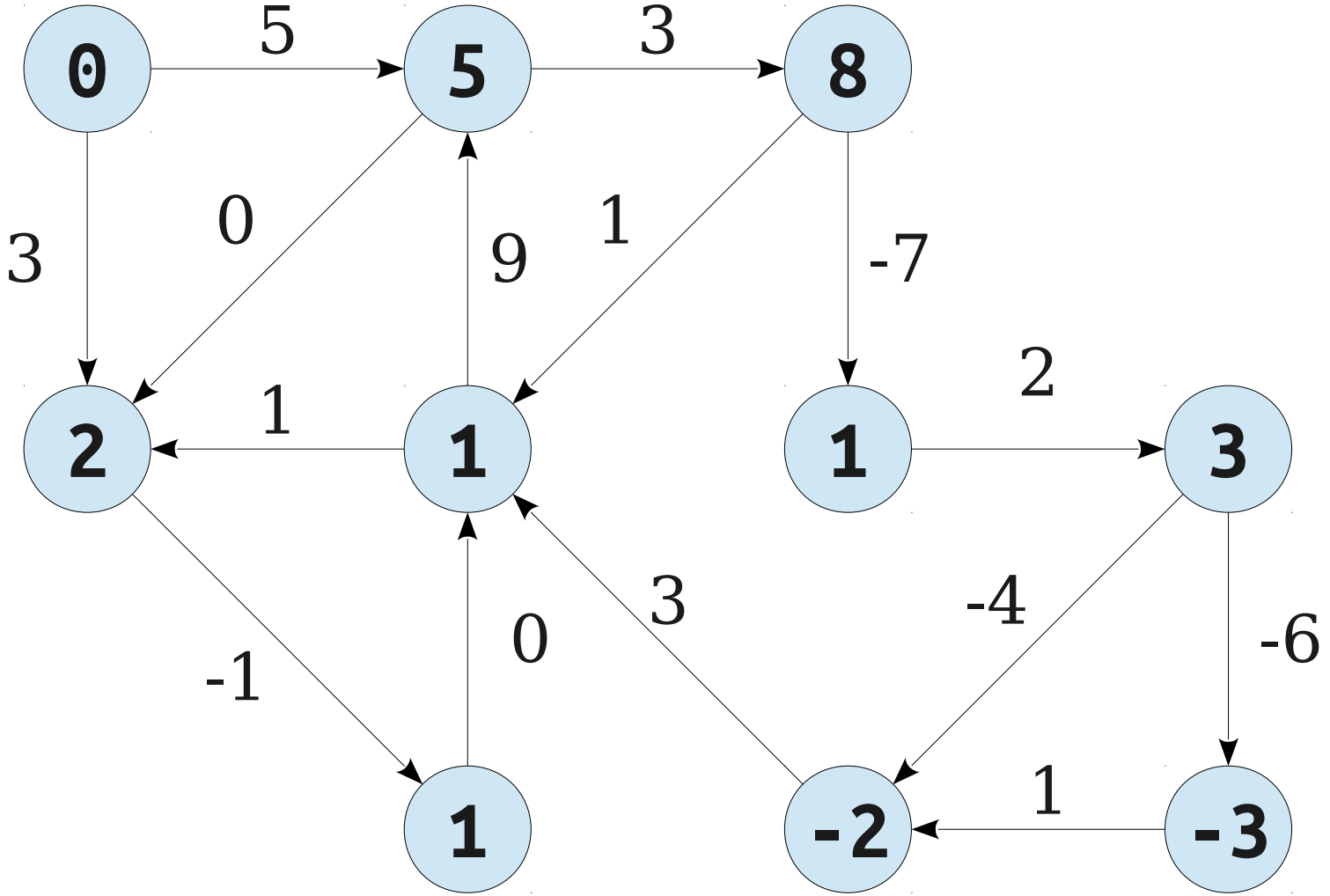
Consider paths of length **8**.



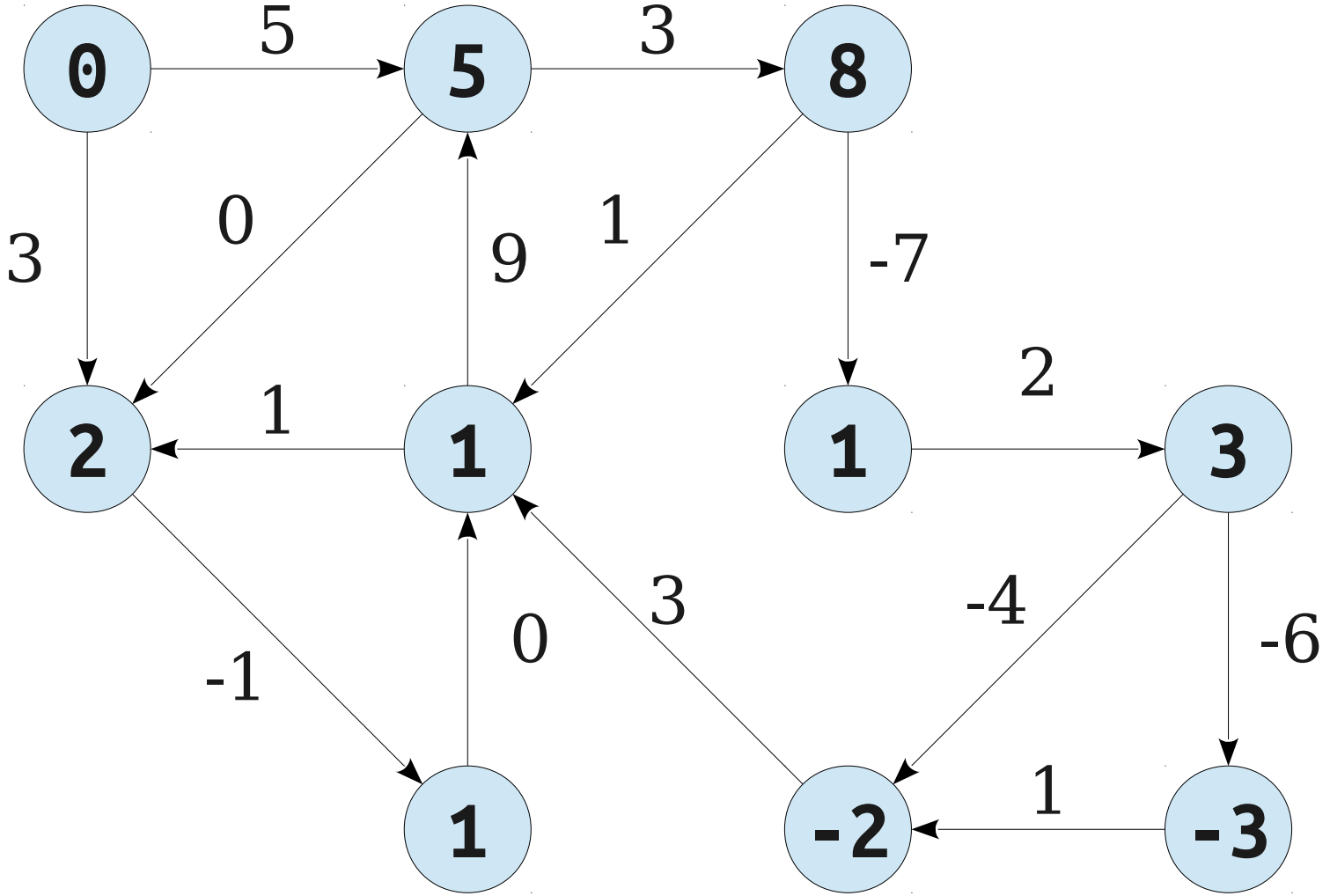
Consider paths of length **8**.



Consider paths of length **9**.



Consider paths of length **9**.



Consider paths of length **10+**.

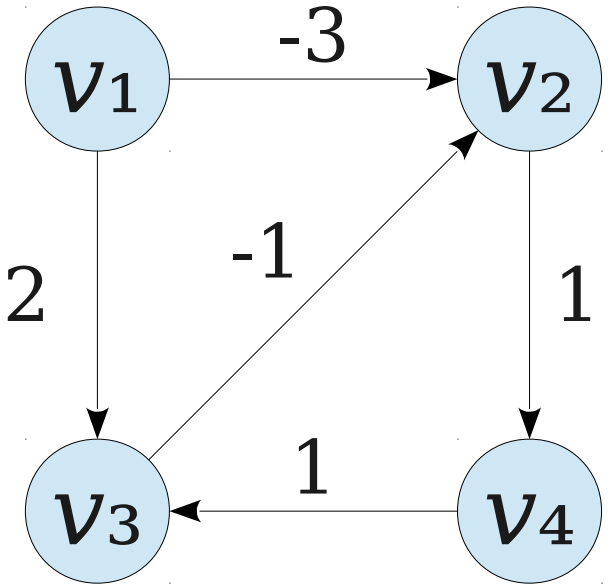
The Recurrence

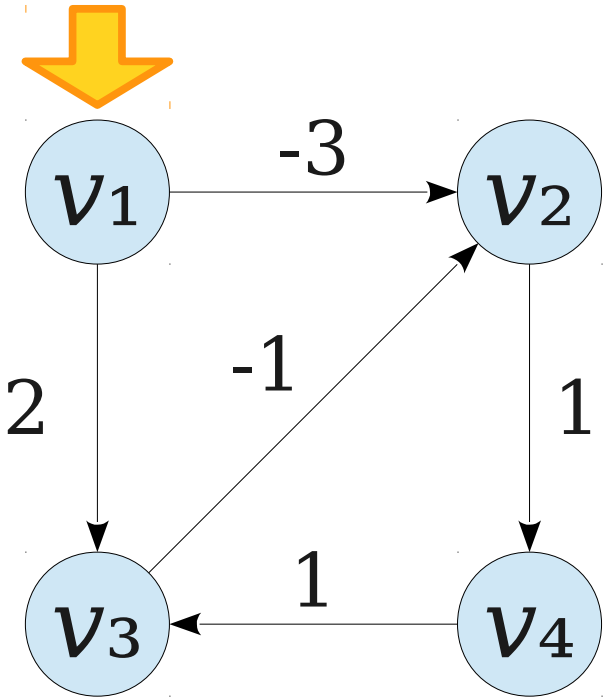
- **Idea:** Find paths of lengths at most 0, 1, 2, …, n.
- Let $w(u, v)$ denote the weight of edge (u, v).
- Let s be our start node. Let OPT(v, i) be the length of the shortest s – v path whose length is at most i, or ∞ if no path exists.
- **Claim:** OPT(v, i) satisfies the following recurrence:

$$\text{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ \min\left\{ \begin{array}{l} \text{OPT}(v,i-1), \\ \min\limits_{(u,v)\in E}\{\text{OPT}(u,i-1)+w(u,v)\} \end{array} \right\} & \text{otherwise} \end{cases}$$

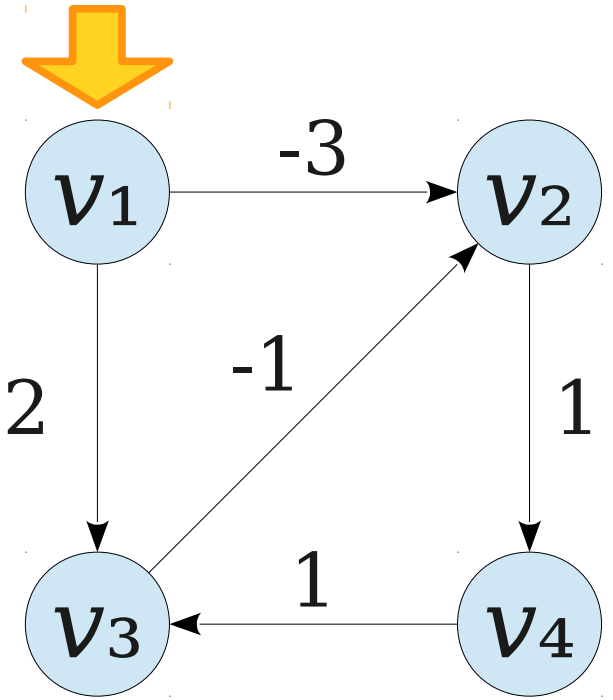
The Bellman-Ford Algorithm

- The **Bellman-Ford algorithm** evaluates this recurrence bottom-up:
  - Create a table DP of size $n \times n$.
  - Set DP[v][0] = ∞ for all $v \neq s$.
  - Set DP[s][0] = 0
  - For $i = 1$ to $n - 1$, for all $v \in V$:
    - Set DP[v][i] =
      min {
        DP[v][$i - 1$],
        min { DP[u][$i - 1$] + $w(u, v)$ } (where $(u, v) \in E$)
      }
  - Return row n of DP.

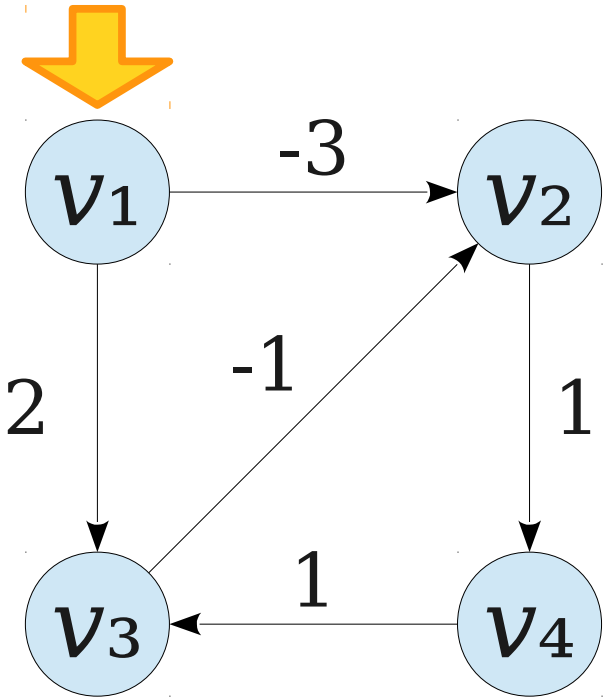

v_1
v_2
v_3
v_4
-3
2
-1
1
1


v_1
v_2
v_3
v_4
-3
2
-1
1
1

	v_1	v_2	v_3	v_4
3				
2				
1				
0				

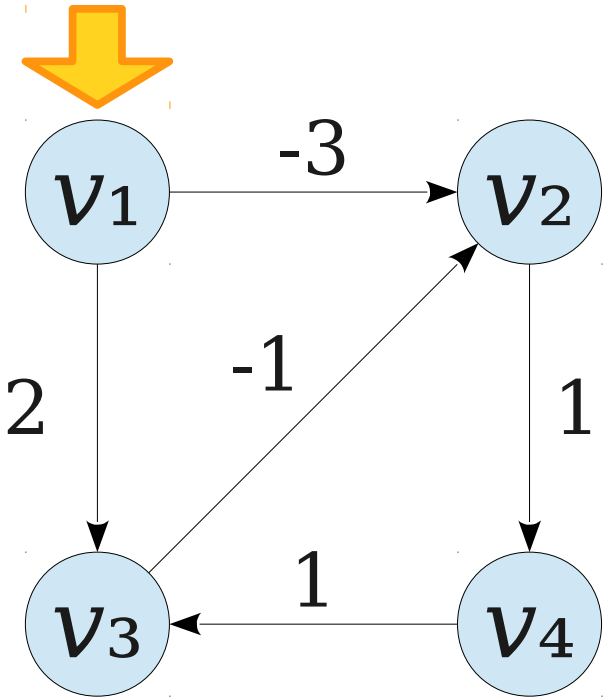


	v_1	v_2	v_3	v_4
3				
2				
1				
0				



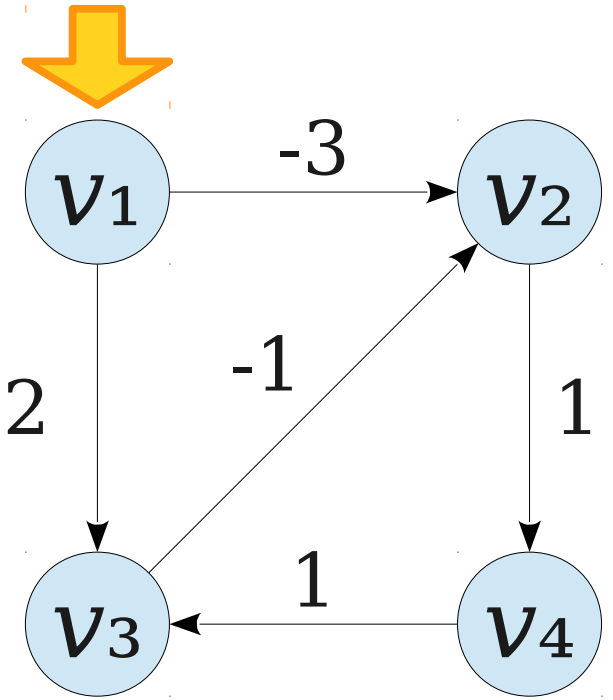
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2				
1				
0	**0**	∞	∞	∞



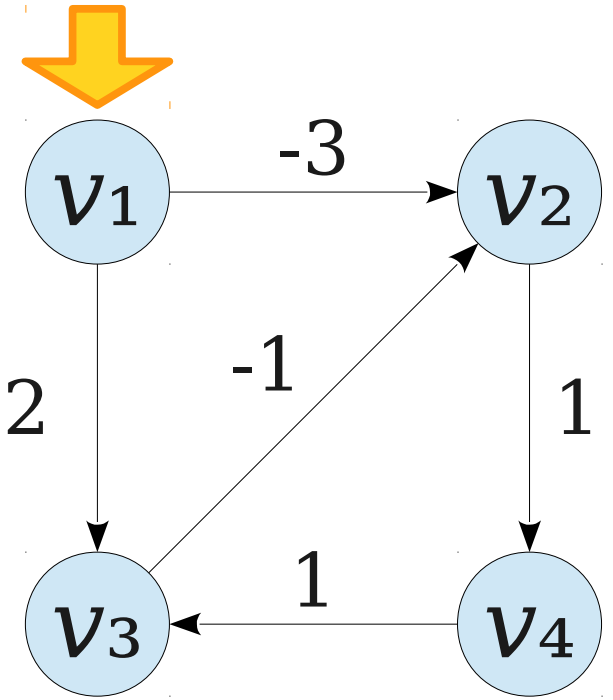
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2				
1				
0	**0**	∞	∞	∞



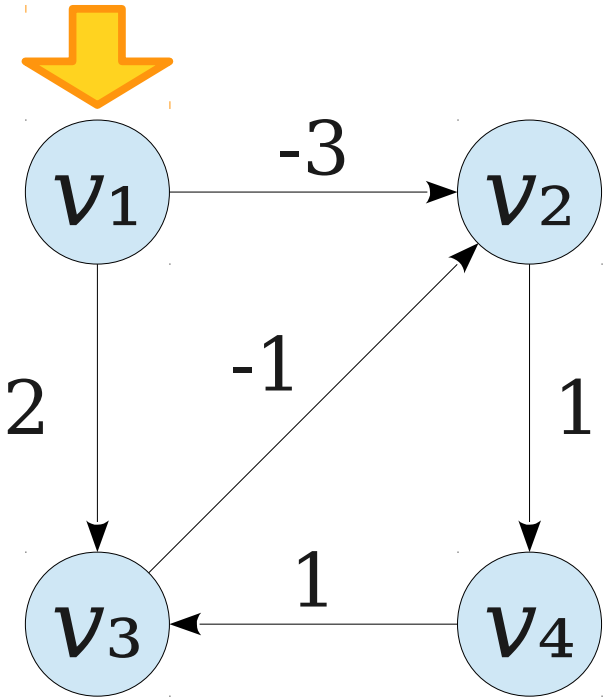
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ \min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \min\limits_{(u,v)\in E}\{\mathrm{OPT}(u,i-1)+w(u,v)\} \end{array} \right\} & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2				
1	**0**			
0	**0**	∞	∞	∞



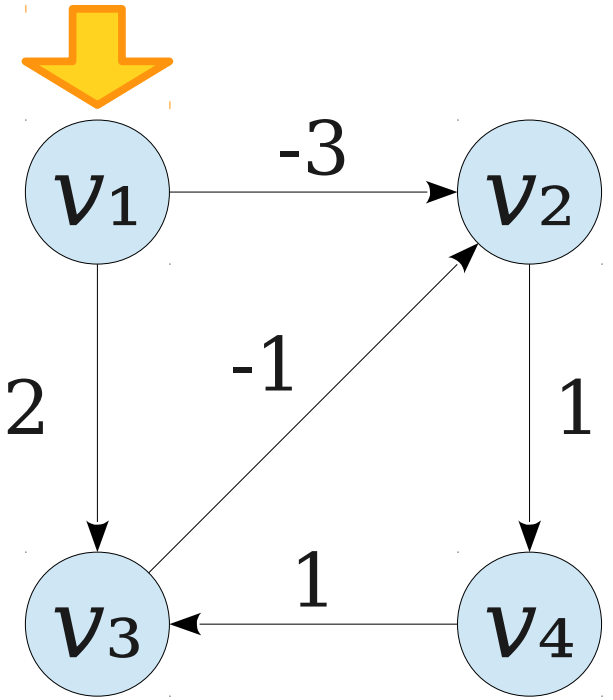
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2				
1	**0**	**-3**		
0	**0**	∞	∞	∞



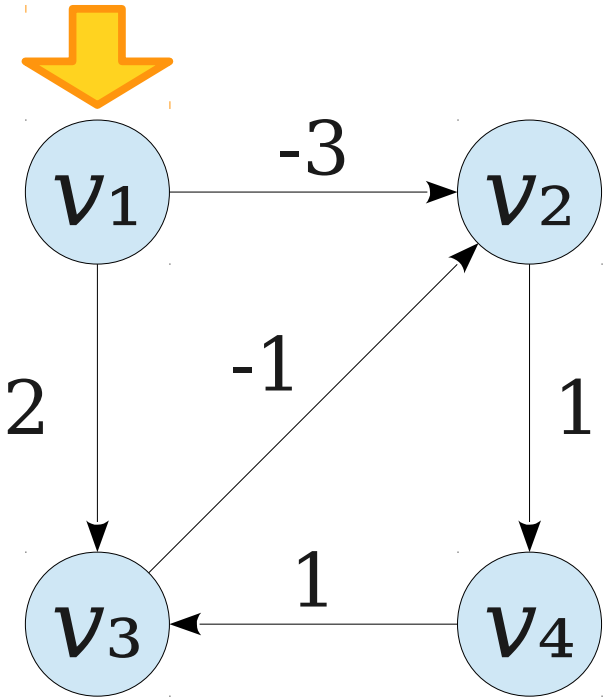
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2				
1	**0**	**-3**	**2**	
0	**0**	∞	∞	∞



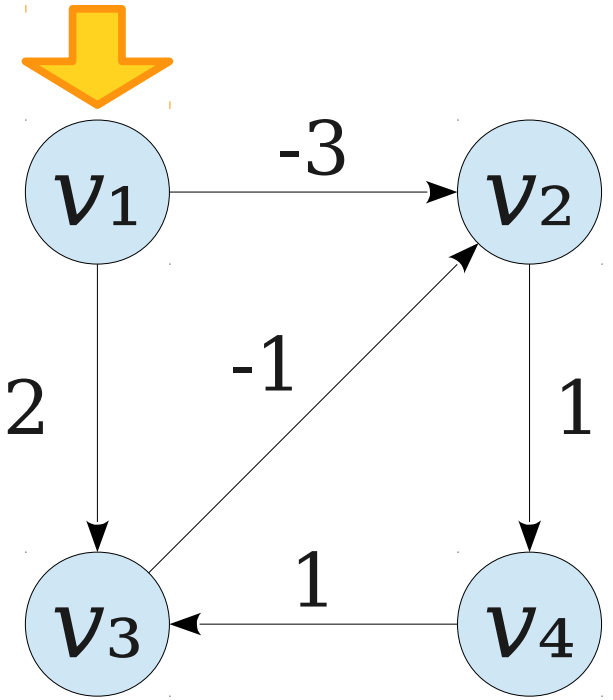
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2				
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



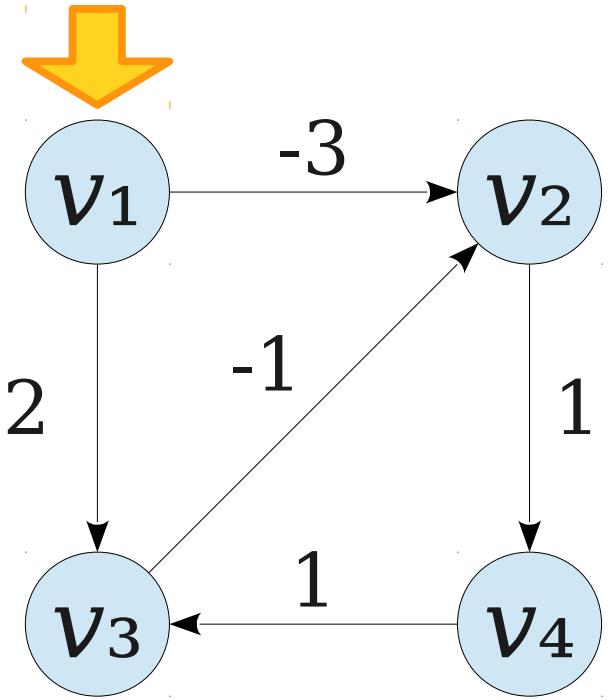
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ \min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \min\limits_{(u,v)\in E}\{\mathrm{OPT}(u,i-1)+w(u,v)\} \end{array} \right\} & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2				
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



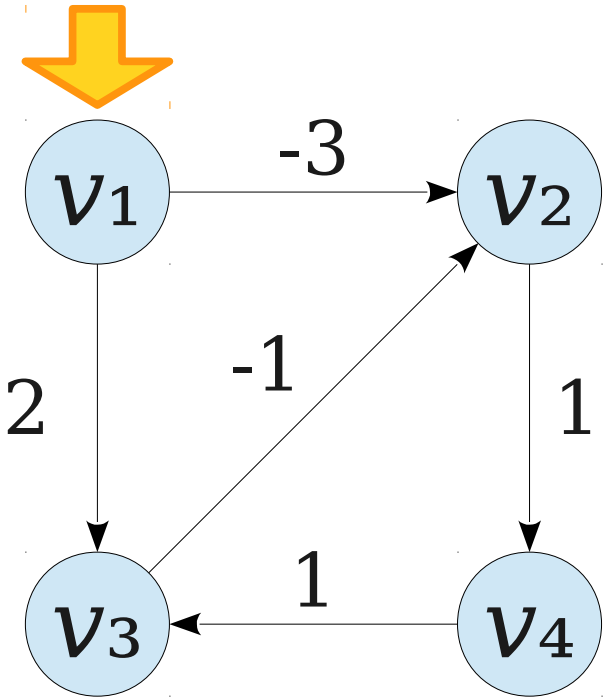
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right\} & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2	**0**			
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



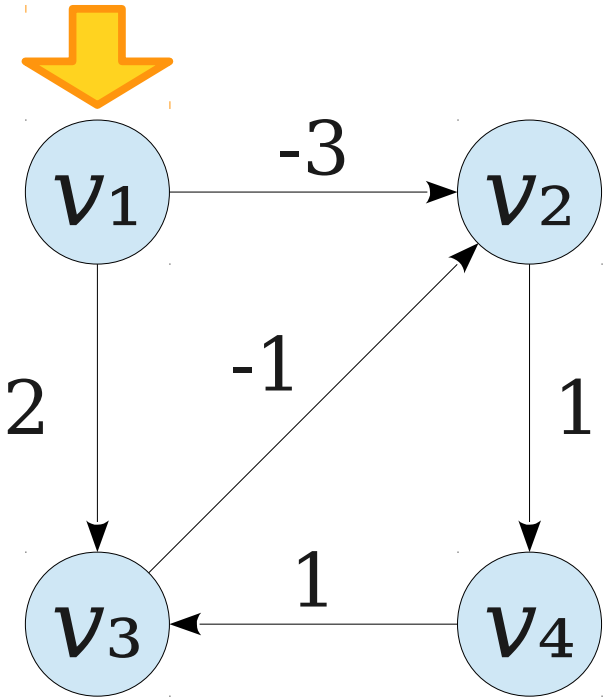
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2	**0**	**-3**		
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



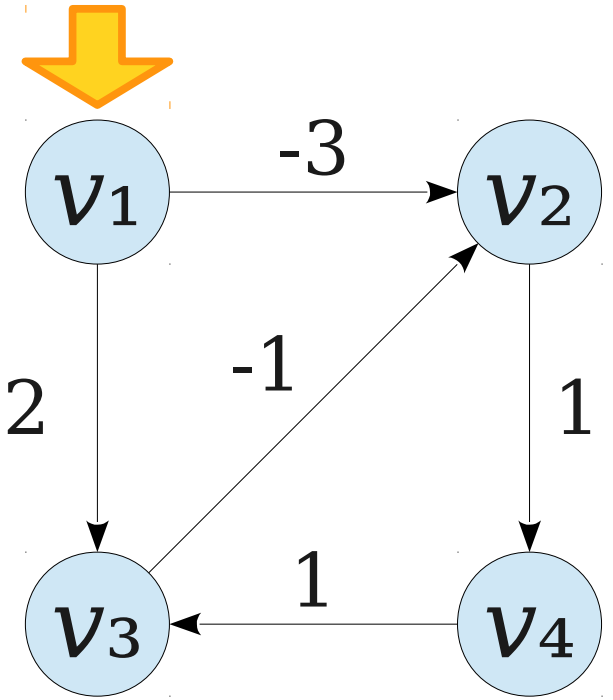
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{\begin{array}{l}\mathrm{OPT}(v,i-1),\\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\}\end{array}\right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2	**0**	**-3**	**2**	
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



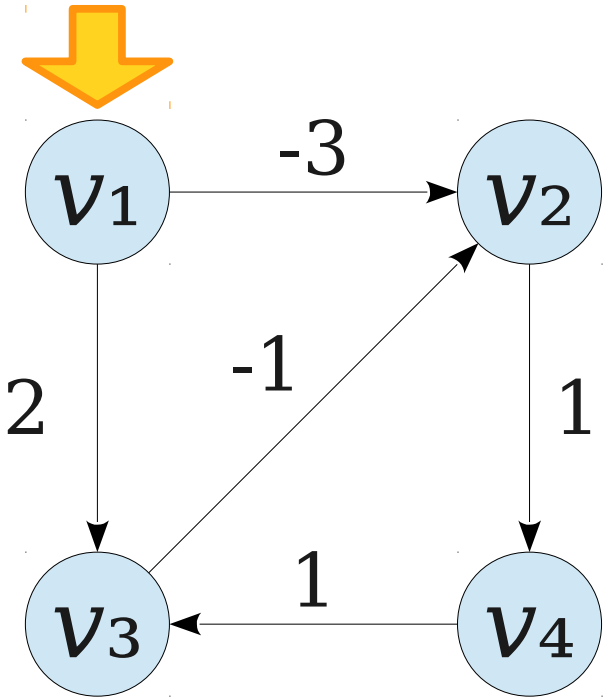
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2	**0**	**-3**	**2**	**-2**
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



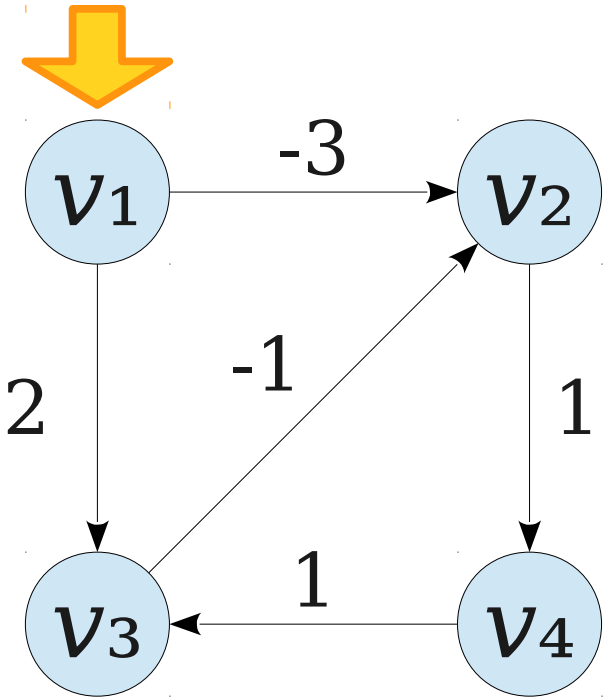
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ \min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \min\limits_{(u,v)\in E}\{\mathrm{OPT}(u,i-1)+w(u,v)\} \end{array} \right\} & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3				
2	**0**	**-3**	**2**	**-2**
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



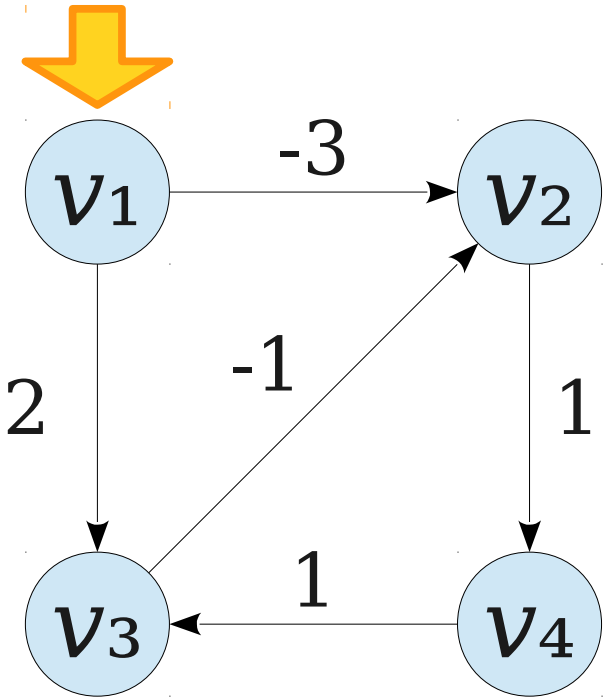
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3	**0**			
2	**0**	**-3**	**2**	**-2**
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



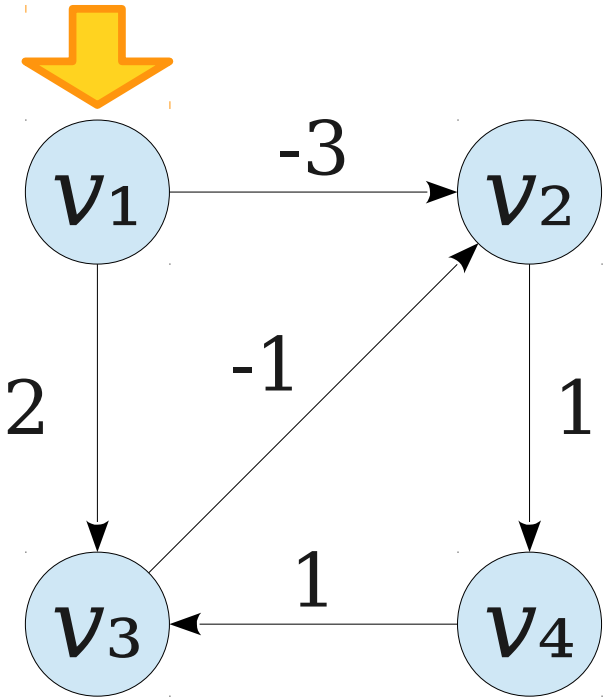
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right\} & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3	**0**	**-3**		
2	**0**	**-3**	**2**	**-2**
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



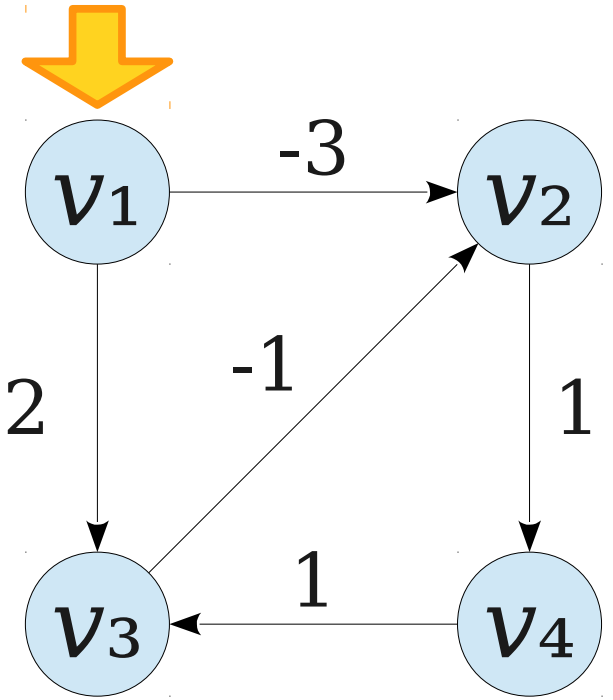
$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3	**0**	**-3**	**-1**	
2	**0**	**-3**	**2**	**-2**
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

	v_1	v_2	v_3	v_4
3	**0**	**-3**	**-1**	**-2**
2	**0**	**-3**	**2**	**-2**
1	**0**	**-3**	**2**	**∞**
0	**0**	**∞**	**∞**	**∞**



$$\mathrm{OPT}(v,i)=\begin{cases} 0 & \text{if } i=0 \text{ and } v=s \\ \infty & \text{if } i=0 \text{ and } v\neq s \\ min\left\{ \begin{array}{l} \mathrm{OPT}(v,i-1), \\ \underset{(u,v)\in E}{min}\left\{\mathrm{OPT}(u,i-1)+w(u,v)\right\} \end{array} \right. & \text{otherwise} \end{cases}$$

Analyzing Time Complexity

- What is the time complexity of this algorithm?
  - Create a table DP of size $n \times n$.
  - Set DP[v][0] = ∞ for all $v \neq s$.
  - Set DP[s][0] = 0
  - For $i = 1$ to $n - 1$, for all $v \in V$:
    - Set DP[v][i] =
      min {
        DP[v][$i - 1$],
        min { DP[u][$i - 1$] + $w(u, v)$ } *(where $(u, v) \in E$)*
      }
  - Return row n of DP.
- Answer: **O(*mn*)**, i you reverse G prior to running the algorithm.

Analyzing Space Complexity

- What is the *space* complexity of this algorithm?
  - Create a table DP of size $n \times n$.
  - Set DP[v][0] = ∞ for all $v \neq s$.
  - Set DP[s][0] = 0
  - For $i = 1$ to $n - 1$, for all $v \in V$:
    - Set DP[v][i] =
      min {
        DP[v][$i - 1$],
        min { DP[u][$i - 1$] + $w(u, v)$ } *(where $(u, v) \in E$)*
      }
  - Return row n of DP.
- Answer: **$O(n^2)$**. *(Can we reduce this?)*

Shortest paths with negative weights: practical improvements

Space optimization. Maintain two 1D arrays (instead of 2D array).

- $d[v]$ = length of a shortest $v \rightsquigarrow t$ path that we have found so far.
- $successor[v]$ = next node on a $v \rightsquigarrow t$ path.

Performance optimization. If $d[w]$ was not updated in iteration $i - 1$, then no reason to consider edges entering w in iteration i.

Bellman–Ford–Moore: efficient implementation

BELLMAN–FORD–MOORE(V, E, c, t)

FOREACH node $v \in V$:

  $d[v] \leftarrow \infty$.

  $successor[v] \leftarrow null$.

$d[t] \leftarrow 0$.

FOR $i = 1$ TO $n - 1$

  FOREACH node $w \in V$:

    IF ($d[w]$ was updated in previous pass)

      FOREACH edge $(v, w) \in E$:

        IF ($d[v] > d[w] + \ell_{vw}$)

          $d[v] \leftarrow d[w] + \ell_{vw}$.

          $successor[v] \leftarrow w$.

  IF (no $d[\cdot]$ value changed in pass i) STOP.

pass i
$O(m)$ time

Bellman–Ford–Moore: analysis

Lemma 3. For each node v : $d[v]$ is the length of some $v \rightsquigarrow t$ path.

Lemma 4. For each node v : $d[v]$ is monotone non-increasing.

Lemma 5. After pass i, $d[v] \leq$ length of a shortest $v \rightsquigarrow t$ path using $\leq i$ edges.

Pf. [by induction on i]

- Base case: $i = 0$.
- Assume true after pass i.
- Let P be any $v \rightsquigarrow t$ path with $\leq i + 1$ edges.
- Let (v, w) be first edge in P and let P' be subpath from w to t.
- By inductive hypothesis, at the end of pass i, $d[w] \leq \ell(P')$ because P' is a $w \rightsquigarrow t$ path with $\leq i$ edges.
- After considering edge (v, w) in pass $i + 1$:

and by Lemma 4, $d[w]$ does not increase

$$\begin{aligned} d[v] &\leq \ell_{vw} + d[w] \\ &\leq \ell_{vw} + \ell(P') \\ &= \ell(P) \quad \blacksquare \end{aligned}$$

and by Lemma 4, $d[v]$ does not increase

Bellman–Ford–Moore: analysis

Theorem 2. Assuming no negative cycles, Bellman–Ford–Moore computes the lengths of the shortest $v \rightsquigarrow t$ paths in $O(mn)$ time and $\Theta(n)$ extra space.

Pf. Lemma 2 + Lemma 5. ▪

shortest path exists and has at most $n-1$ edges

after i passes, $d[v] \leq$ length of shortest path that uses $\leq i$ edges

Remark. Bellman–Ford–Moore is typically faster in practice.

- Edge (v, w) considered in pass $i+1$ only if $d[w]$ updated in pass i.
- If shortest path has k edges, then algorithm finds it after $\leq k$ passes.

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

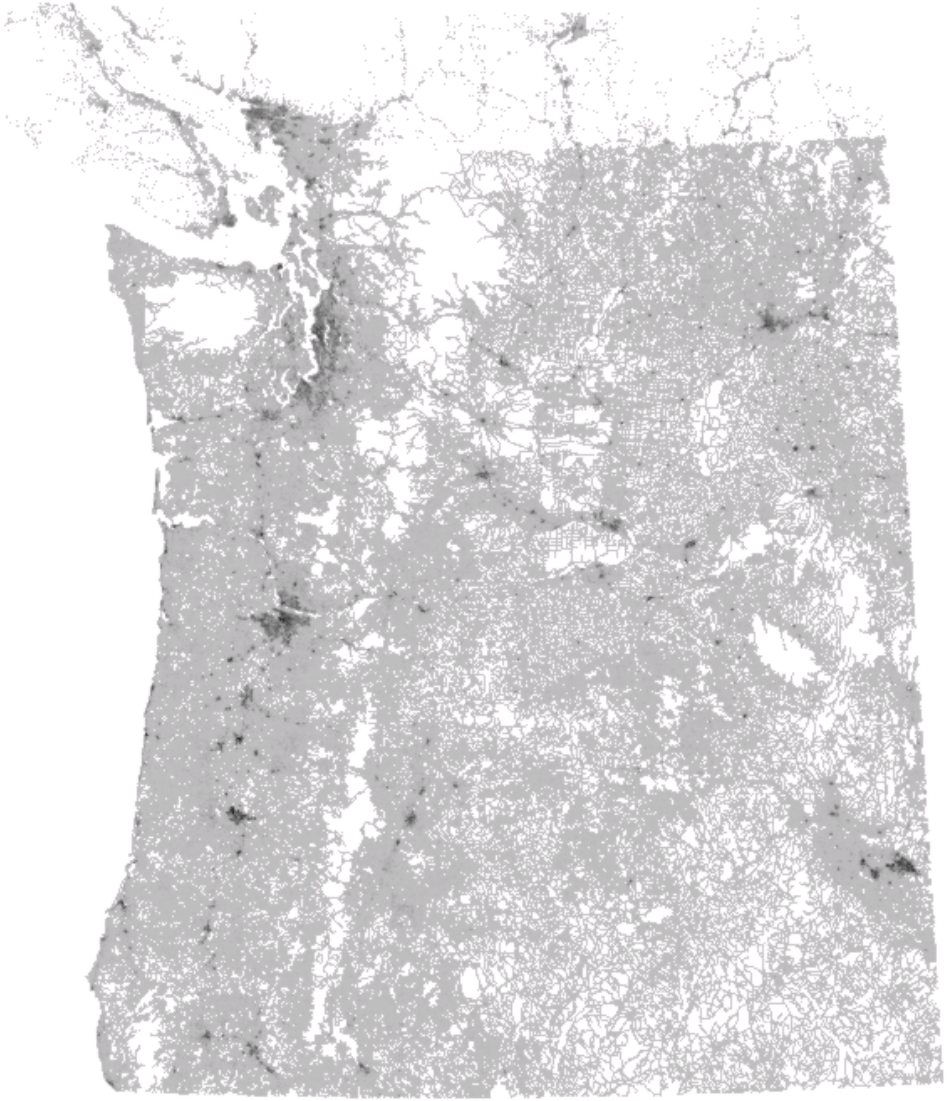
Single-source shortest paths with negative weights

year	worst case	discovered by
1955	$O(n^4)$	Shimbel
1956	$O(m\, n^2\, W)$	Ford
1958	$O(m\, n)$	Bellman, Moore
1983	$O(n^{3/4}\, m \log W)$	Gabow
1989	$O(m\, n^{1/2} \log(nW))$	Gabow–Tarjan
1993	$O(m\, n^{1/2} \log W)$	Goldberg
2005	$O(n^{2.38}\, W)$	Sankowsi, Yuster–Zwick
2016	$\tilde{O}(n^{10/7} \log W)$	Cohen–Mądry–Sankowski–Vladu
20xx		

single-source shortest paths with weights between –W and W

Practical Concerns

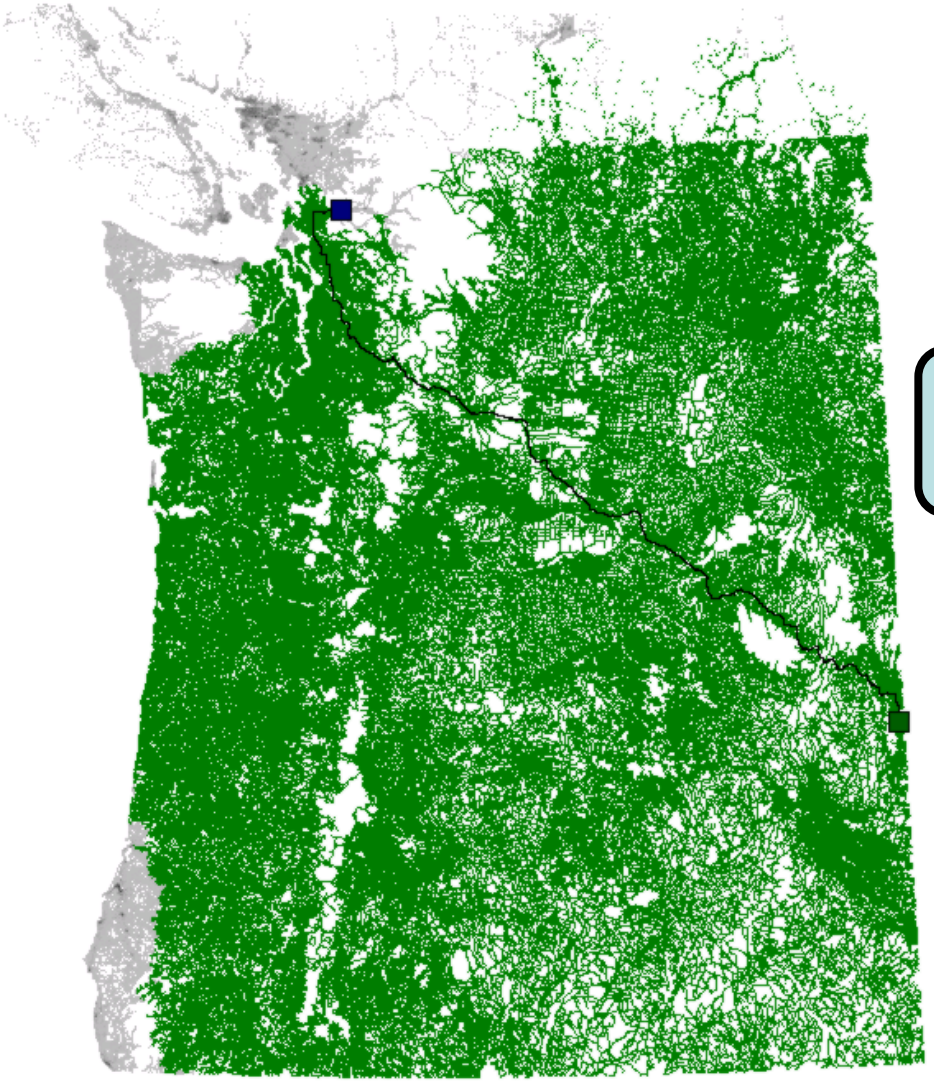
Dijkstra Example



1.6 million vertices
3.8 million edges
Distance = travel time.

Images comes from A.V. Goldberg

Dijkstra Example

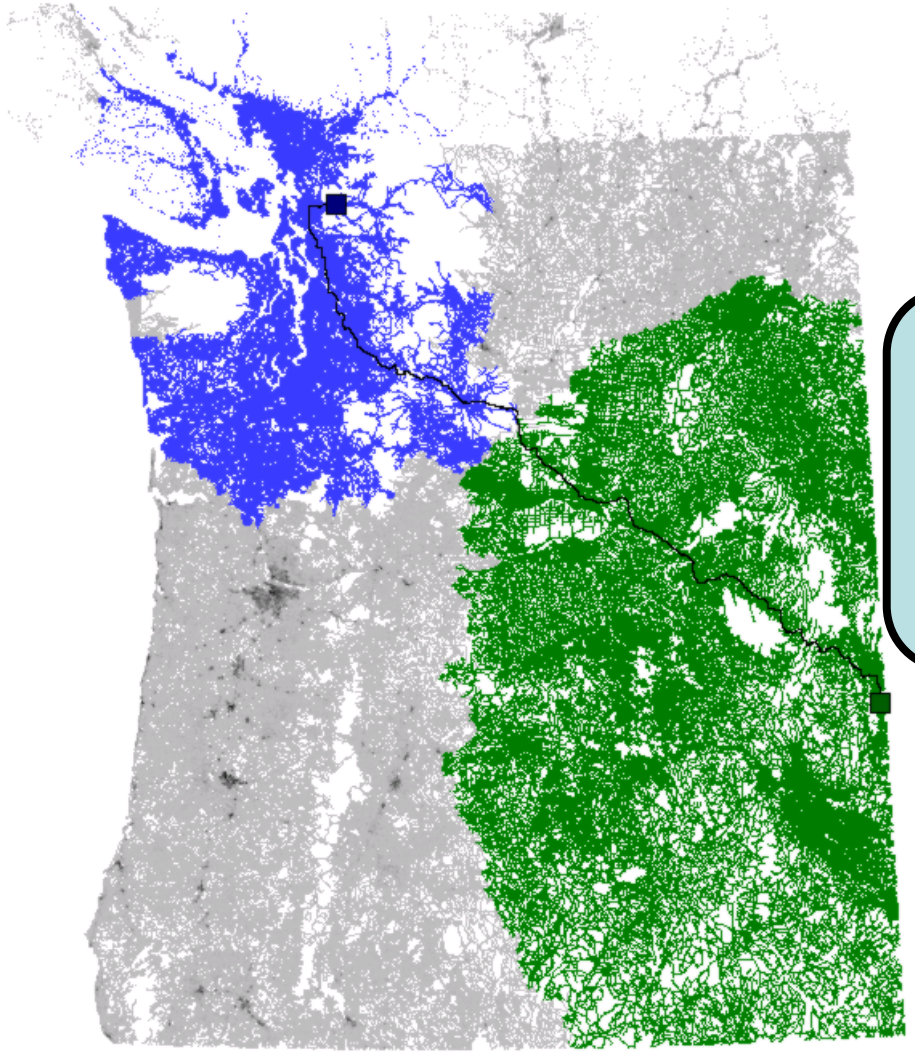


Searched Area
(starting from green point)

Problem of Dijkstra:
Didn't take account of where is t

340ms

Bidirectional Dijkstra



Forward search
Backward search

Problem of Bidirectional Dijkstra:
Forward search did not take account of t
Backward search did not take account of s.

A^* Search

```
AStar(G, c, s, t) {
    Initialize set of explored nodes S ← {s}

    // Maintain distance from s to each vertices in S
    d[s] ← 0

    while (S ≠ V)
    {
        Pick an edge (u, v) such that u ∈ S and v ∉ S and
                d[u] + c(u,v) + h(v) is as small as possible.

        Add v to S and define d[v] = d[u] + c(u,v) .
        Parent(v) ← u .
    }
```

BFS

Dijkstra

A^*

- $h(v)$ is the estimate of distance from v to t
- If $h(v)$ is exactly the shortest distance from v to t, then the algorithm would go directly to t.

Routing Problem in Large Graphs

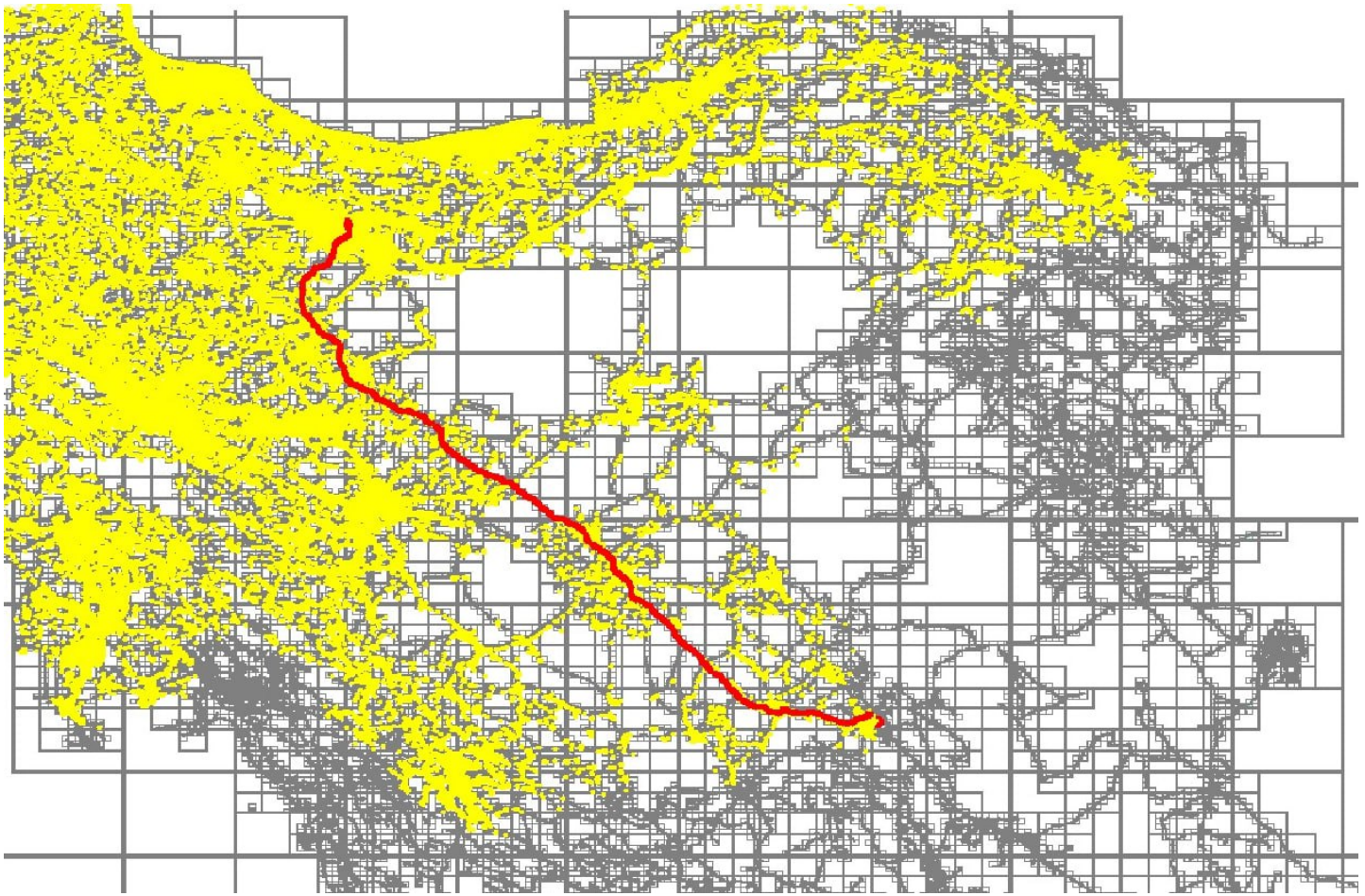
- Iran's Graph has 2,052,067 vertices and 2707945 edges.

- Consider Routing From Azadi Gate to Kerman
- The shortest path contains 3437 many edges.
- Dijkstra algorithm visits 1722445 many nodes!

Algorithms & Photo By: A. R. Tofighi Mohammadi

Dijkstra Performance

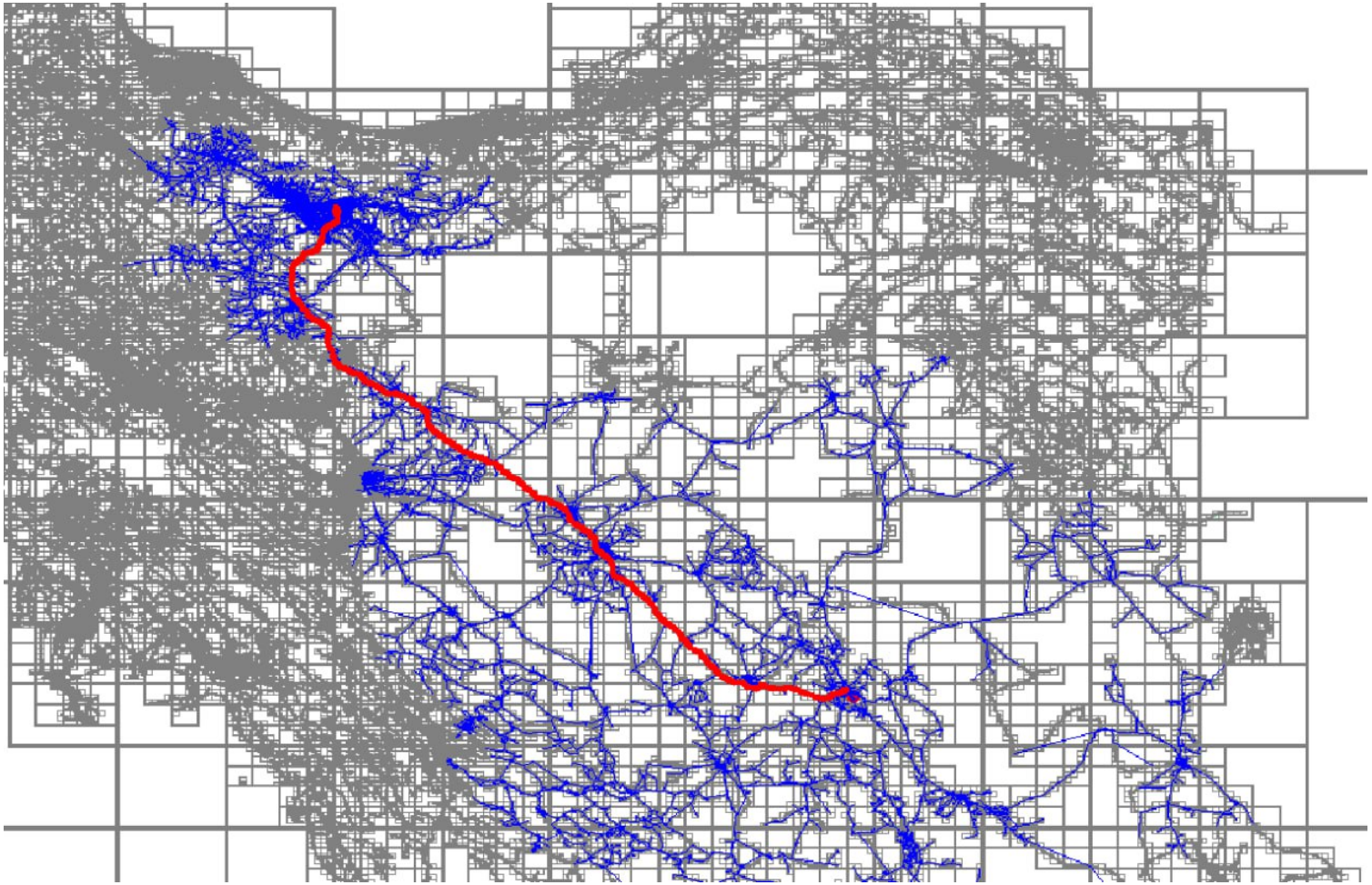
#Visited Nodes: 1,722,445



Algorithms & Photo By: A. R. Tofighi Mohammadi

Bi-Directional Dijkstra

#Visited Nodes: 660,564



Algorithms & Photo By: A. R. Tofighi Mohammadi

Landmark Selection

Preprocessing

- Random selection is fast.
- Many heuristics find better landmarks.
- Local search can find a good subset of candidate landmarks.
- We use a heuristic with local search.

Preprocessing/query trade-off.

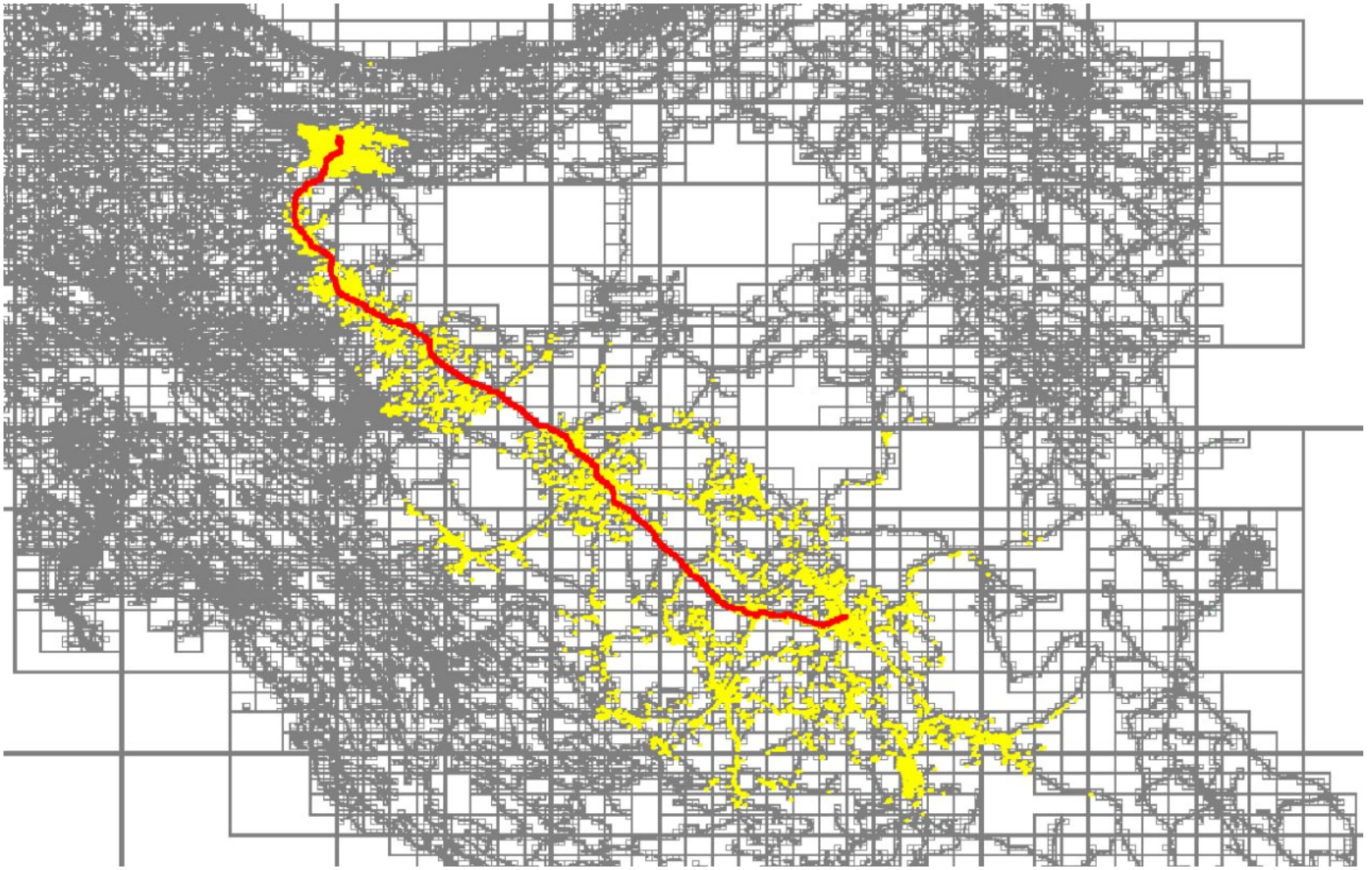
Query

- For a specific s, t pair, only some landmarks are useful.
- Use only active landmarks that give best bounds on dist(s, t).
- If needed, dynamically add active landmarks (good for the search frontier).
- Only three active landmarks on the average.

Allows using many landmarks with small time overhead.

Dijkstra + Landmark

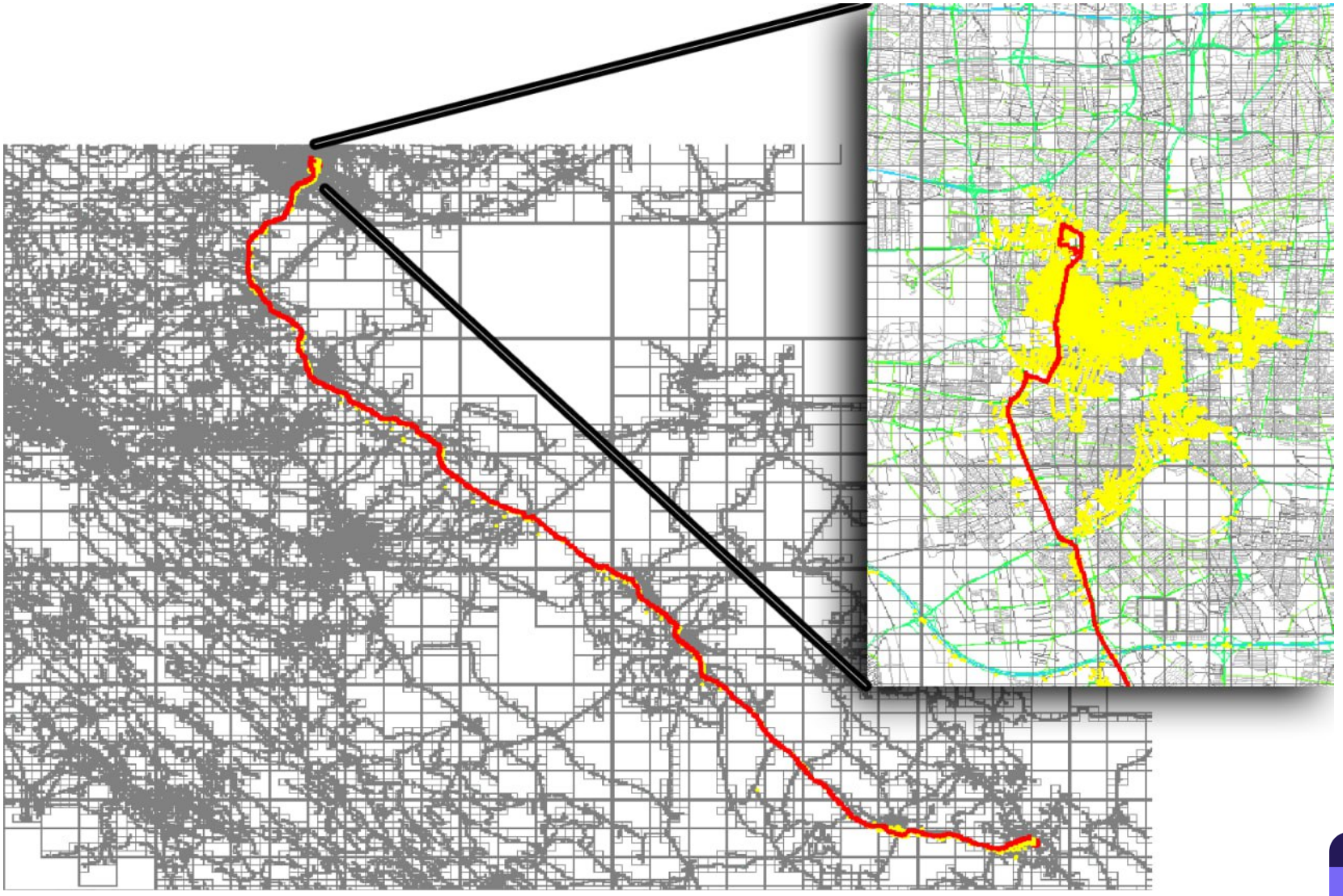
#Visited Nodes: 405,784



Algorithms & Photo By: A. R. Tofighi Mohammadi

Bi-directional Dijkstra + Landmark

#Visited Nodes: 19,056



Algorithms & Photo By: A. R. Tofighi Mohammadi