CSCI 3302
Introduction to Robotics

Prof. Alessandro Roncone

aroncone@colorado.edu

https://hiro-group.ronc.one

Lab 5
Path planning with Dijkstra

Prof. Alessandro Roncone

aroncone@colorado.edu

https://hiro-group.ronc.one

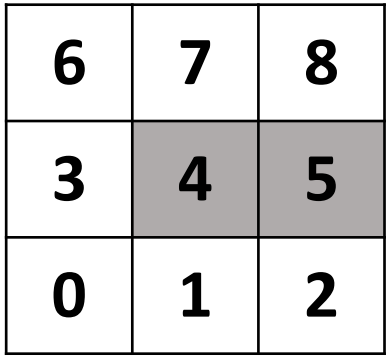
Dijkstra Lab In-Depth

Data Structures

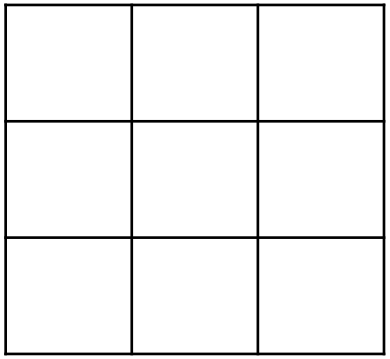
- int dist[num_vertices]
  - Final shortest-distance/lowest cost relative to source
- int prev[num_vertices]
  - Previous node on path back to source
- int Q_cost[num_vertices]
  - Queue of vertices with priority equal to the best-known cost to get to that vertex from source. Used for bookkeeping during Dijkstra function

Important Functions

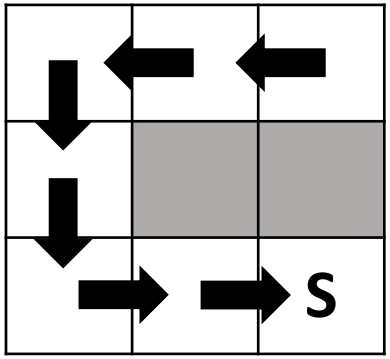
- Dijkstra(source_vertex)
  - Returns a populated ‘prev’ array, using source_vertex as origin of all paths
- Reconstruct_path(prev, vertex)
  - Iteratively follows prev[vertex] back to the source and returns a list of vertices on the (shortest) path
- get_travel_cost(source_vertex, dest_vertex)
  - Returns the cost (distance) of traveling between two vertices. 1 if neighbors, BIG_NUMBER if not.

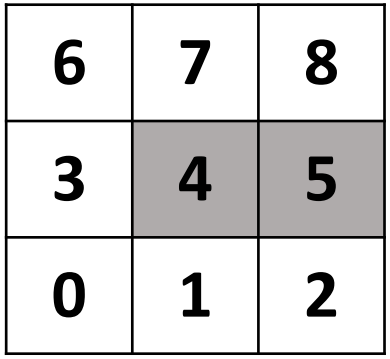



```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```



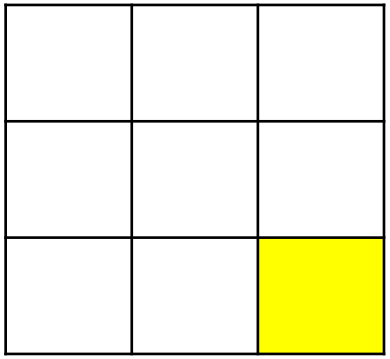
Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [inf, inf, inf, inf, inf, inf, inf, inf, inf]
prev = [-1, -1, -1, -1, -1, -1, -1, -1, -1]

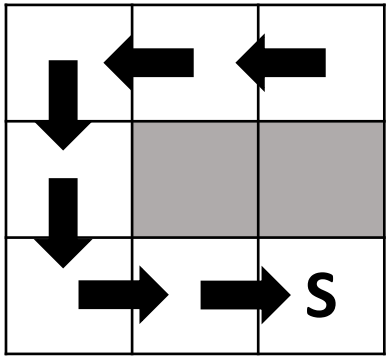





```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Initialize Source

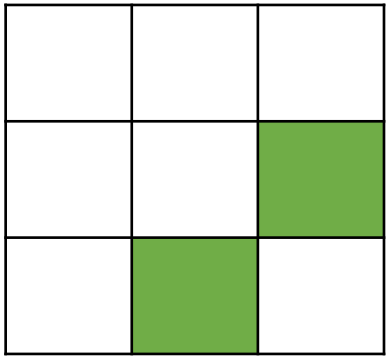


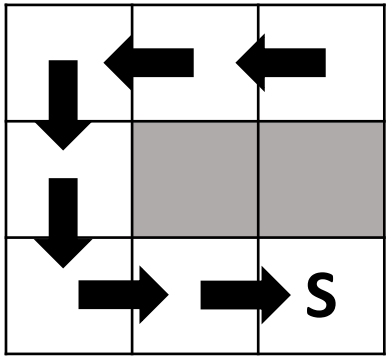


Q = [-1, -1, 0, -1, -1, -1, -1, -1, -1]
dist = [inf, inf, 0, inf, inf, inf, inf, inf, inf]
prev = [-1, -1, -1, -1, -1, -1, -1, -1, -1]

6	7	8
3	4	5
0	1	2

Expand Neighbors



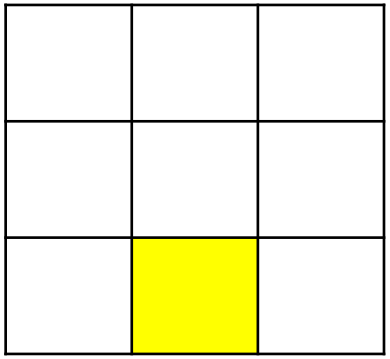


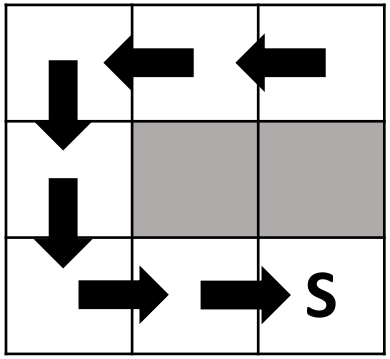

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [inf, inf, 0, inf, inf, inf, inf, inf, inf]
prev = [-1, -1, -1, -1, -1, -1, -1, -1, -1]

6	7	8
3	4	5
0	1	2

If new path is cheaper than old path, update and add to Q



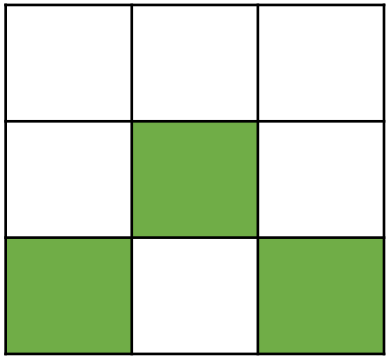


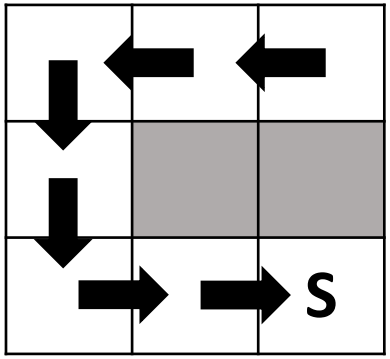

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, **1**, -1, -1, -1, -1, -1, -1, -1]
dist = [inf, **1**, 0, inf, inf, inf, inf, inf, inf]
prev = [-1, 2, -1, -1, -1, -1, -1, -1, -1]

6	7	8
3	4	5
0	1	2

Expand Neighbors



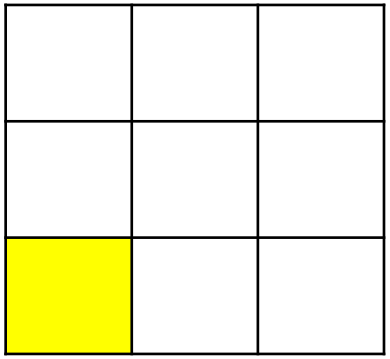



```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [inf, 1, 0, inf, inf, inf, inf, inf, inf]
prev = [-1, 2, -1, -1, -1, -1, -1, -1, -1]

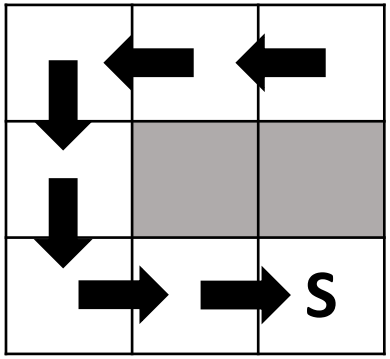
6	7	8
3	4	5
0	1	2

If new path is cheaper than old path, update and add to Q



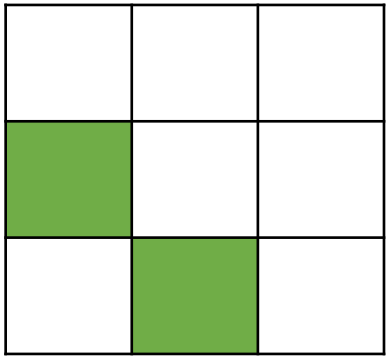
```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

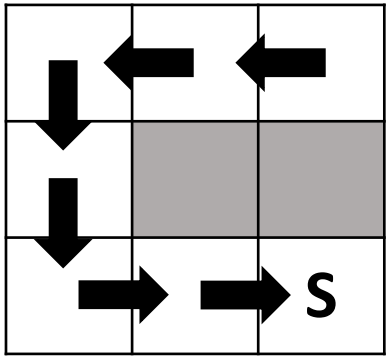
Q = [**2**, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [**2**, 1, 0, inf, inf, inf, inf, inf, inf]
prev = [1, 2, -1, -1, -1, -1, -1, -1, -1]



6	7	8
3	4	5
0	1	2

Expand Neighbors



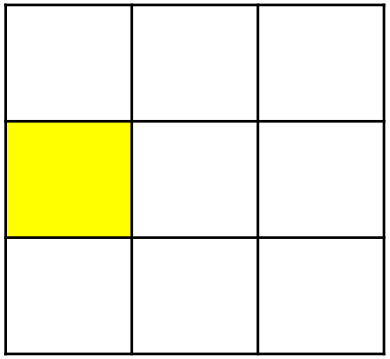


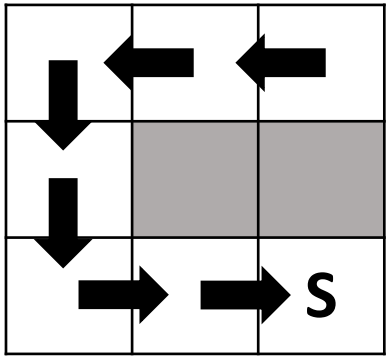

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [2, 1, 0, inf, inf, inf, inf, inf, inf]
prev = [1, 2, -1, -1, -1, -1, -1, -1, -1]

6	7	8
3	4	5
0	1	2

If new path is cheaper than old path, update and add to Q



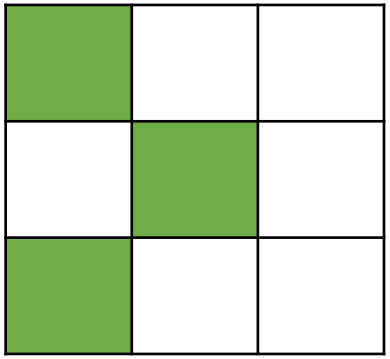


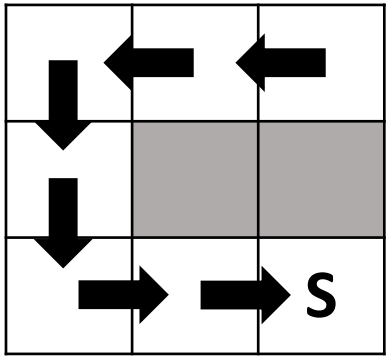

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, -1, -1, **3**, -1, -1, -1, -1, -1]
dist = [2, 1, 0, **3**, inf, inf, inf, inf, inf]
prev = [1, 2, -1, 0, -1, -1, -1, -1, -1]

6	7	8
3	4	5
0	1	2

Expand Neighbors



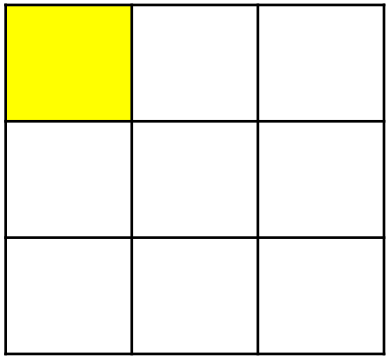


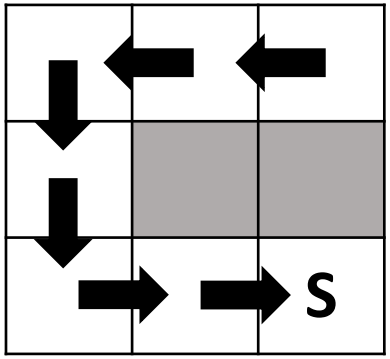

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [2, 1, 0, 3, inf, inf, inf, inf, inf]
prev = [1, 2, -1, 0, -1, -1, -1, -1, -1]

6	7	8
3	4	5
0	1	2

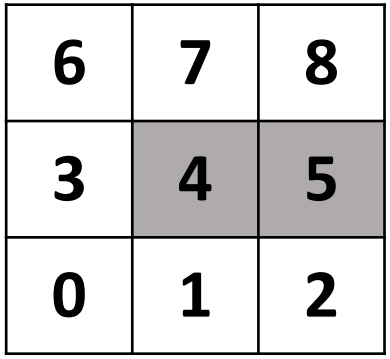
If new path is cheaper than old path, update and add to Q





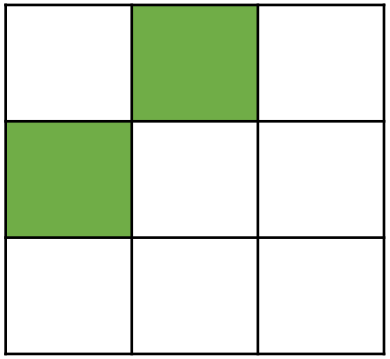

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

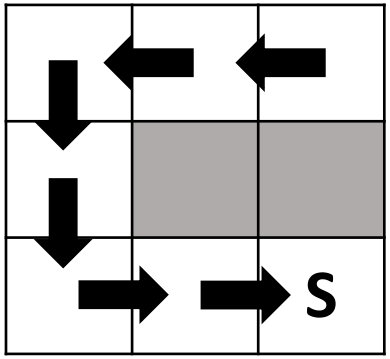
Q = [-1, -1, -1, -1, -1, -1, **4**, -1, -1]
dist = [2, 1, 0, 3, inf, inf, **4**, inf, inf]
prev = [1, 2, -1, 0, -1, -1, 3, -1, -1]




```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Expand Neighbors

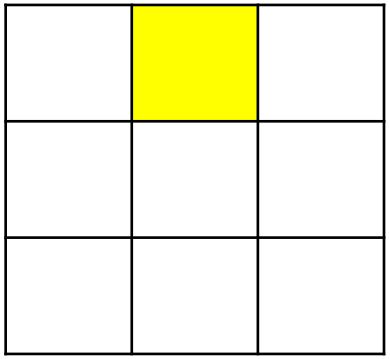


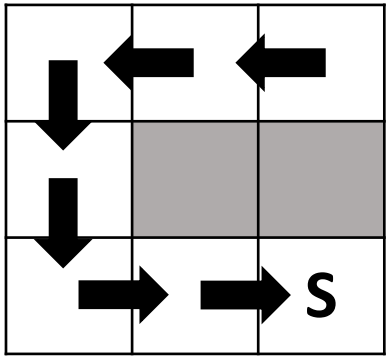


Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [2, 1, 0, 3, inf, inf, 4, inf, inf]
prev = [1, 2, -1, 0, -1, -1, 3, -1, -1]

6	7	8
3	4	5
0	1	2

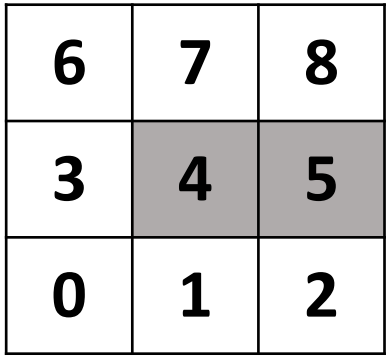
If new path is cheaper than old path, update and add to Q





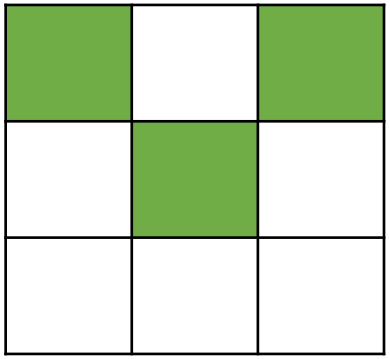

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

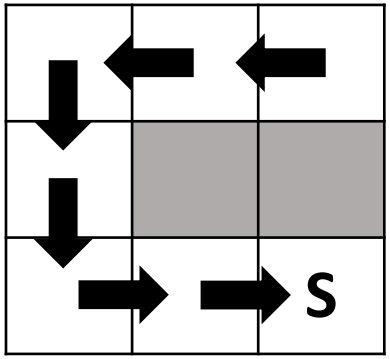
Q = [-1, -1, -1, -1, -1, -1, -1, **5**, -1]
dist = [2, 1, 0, 3, inf, inf, 4, **5**, inf]
prev = [1, 2, -1, 0, -1, -1, 3, 6, -1]



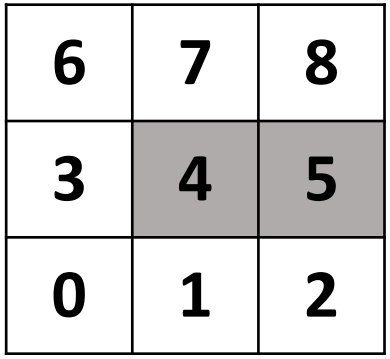

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Expand Neighbors

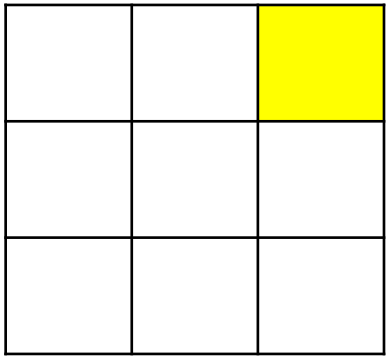




Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [2, 1, 0, 3, inf, inf, 4, 5, inf]
prev = [1, 2, -1, 0, -1, -1, 3, 6, -1]

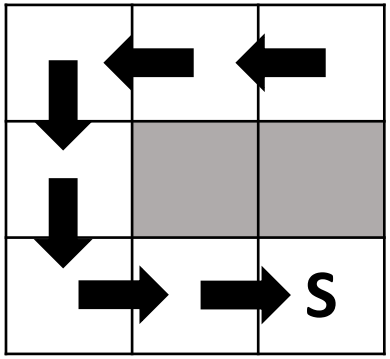


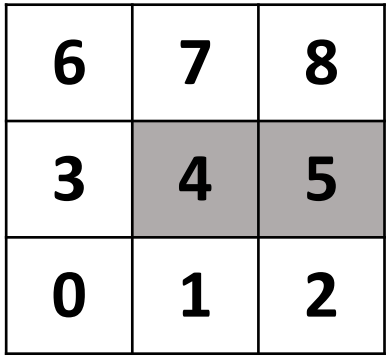
If new path is cheaper than old path, update and add to Q



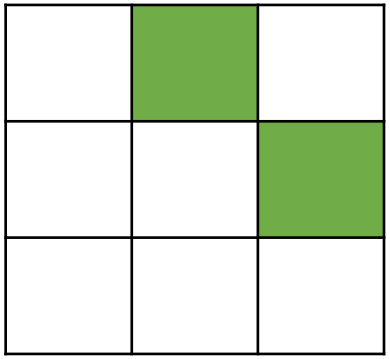
```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

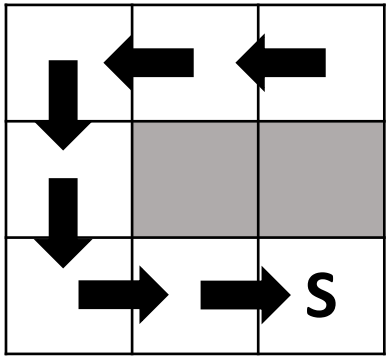
Q = [-1, -1, -1, -1, -1, -1, -1, -1, **6**]
dist = [2, 1, 0, 3, inf, inf, 4, 5, **6**]
prev = [1, 2, -1, 0, -1, -1, 3, 6, 7]





Expand Neighbors





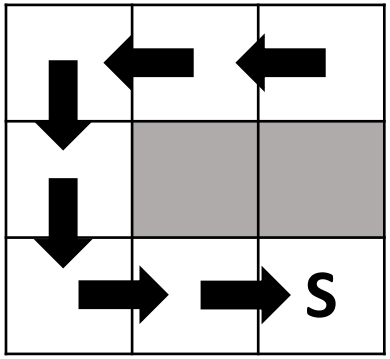

```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [2, 1, 0, 3, inf, inf, 4, 5, 6]
prev = [1, 2, -1, 0, -1, -1, 3, 6, 7]

6	7	8
3	4	5
0	1	2

Final Dist costmap

4	5	6
3	∞	∞
2	1	0




```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]

dist = [2, 1, 0, 3, inf, inf, 4, 5, 6]

prev = [1, 2, -1, 0, -1, -1, 3, 6, 7]

Dijkstra Lab In-Depth

Data Structures

- int dist[num_vertices] → [2, 1, 0, 3, inf, inf, 4, 5, 6]
  - Final shortest-distance/lowest cost relative to source
- int prev[num_vertices] → [1, 2, -1, 0, -1, -1, 3, 6, 7]
  - Previous node on path back to source
- int Q_cost[num_vertices] →
  - Queue of vertices with priority equal to the best-known cost to get to that vertex from source. Used for bookkeeping during Dijkstra function

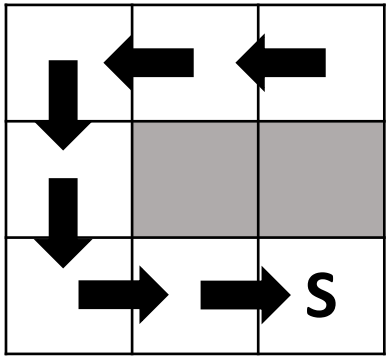
Important Functions

- Dijkstra(source_vertex)
  - Returns a populated 'prev' array, using source_vertex as origin of all paths
- Reconstruct_path(prev, vertex)
  - Iteratively follows prev[vertex] back to the source and returns a list of vertices on the (shortest) path
- get_travel_cost(source_vertex, dest_vertex)
  - Returns the cost (distance) of traveling between two vertices. 1 if neighbors, BIG_NUMBER if not.

6	7	8
3	4	5
0	1	2

Final Dist costmap

4	5	6
3	∞	∞
2	1	0




```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0                                  // Initialization
3
4      create vertex set Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY                        // Unknown distance from source to v
9              prev[v] ← UNDEFINED                       // Predecessor of v
10
11         Q.add_with_priority(v, dist[v])
12
13
14     while Q is not empty:                             // The main loop
15         u ← Q.extract_min()                           // Remove and return best vertex
16         for each neighbor v of u:                     // only v that is still in Q
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)
22
23     return dist[], prev[]
```

Q = [-1, -1, -1, -1, -1, -1, -1, -1, -1]
dist = [2, 1, 0, 3, inf, inf, 4, 5, 6]
prev = [1, 2, -1, 0, -1, -1, 3, 6, 7]

```
1  S ← empty sequence
2  u ← target
3  while prev[u] is defined:                  // Construct the shortest path with a stack S
4      insert u at the beginning of S         // Push the vertex onto the stack
5      u ← prev[u]                            // Traverse from target to source
6  insert u at the beginning of S             // Push the source onto the stack
```