Algorithm 1 Brute Force

Input: $grid \in \mathbb{R}^{m \times n}$
Output: $dist \in \mathbb{R}^{m \times n}$
for each $\boldsymbol{p} \in grid$ **do**
    for each $\boldsymbol{o} \in grid$ **and** $grid[\boldsymbol{o}]$ **is** 0 **do**
        $dist[\boldsymbol{p}] \leftarrow \min(dist[\boldsymbol{p}], \|\boldsymbol{p} - \boldsymbol{o}\|_2)$
    end for
end for

Algorithm 2 Forward BFS

Input: $grid \in \mathbb{R}^{m \times n}$

Output: $dist \in \mathbb{R}^{m \times n}$

for each $\boldsymbol{p} \in grid$ **do**

  $open$.push($\boldsymbol{p}$), $step \leftarrow 0$, $found \leftarrow false$

  while not $open$.empty() **and not** $found$ **do**

    for each $\boldsymbol{c} \in open$ **do**

      $open$.pop()

      $closed$.insert($\boldsymbol{c}$)

      if $grid[\boldsymbol{c}]$ **is** 0 **then**

        $dist[\boldsymbol{c}] \leftarrow step$, $found \leftarrow true$

        break

      end if

      for each $\boldsymbol{n} \in \boldsymbol{c}.neighbours$ **not in** $closed$ **do**

        $open$.push($\boldsymbol{n}$)

      end for

    end for

    $step \leftarrow step + 1$

  end while

end for

Algorithm 3 Backward BFS

Input: $grid \in \mathbb{R}^{m \times n}$
Output: $dist \in \mathbb{R}^{m \times n}$
for each $\boldsymbol{p} \in grid$ **do**
if $grid[\boldsymbol{p}]$ **is** 0 **then**
$open$.push($\boldsymbol{p}$)
end if
end for
$step \leftarrow 0$
while not $open$.empty() **do**
for each $\boldsymbol{c} \in open$ **do**
$open$.pop()
$closed$.insert($\boldsymbol{c}$)
$dist[\boldsymbol{c}] \leftarrow step$
for each $\boldsymbol{n} \in \boldsymbol{c}.neighbours$ **not in** $closed$ **do**
$open.push(\boldsymbol{n})$
end for
end for
$step \leftarrow step + 1$
end while

Algorithm 4 L1 DP

```
Input: grid ∈ ℝ^{m×n}
Output: dist ∈ ℝ^{m×n}
for each p ∈ grid do
    if grid[p] is 0 then
        dist[p] ← 0
    end if
end for
for x = 0 to m − 1 do
    for y = 0 to n − 1 do
        dist[x][y] ← min(dist[x][y], dist[x − 1][y] + 1)
        dist[x][y] ← min(dist[x][y], dist[x][y − 1] + 1)
    end for
end for
for x = m − 1 to 0 do
    for y = n − 1 to 0 do
        dist[x][y] ← min(dist[x][y], dist[x + 1][y] + 1)
        dist[x][y] ← min(dist[x][y], dist[x][y + 1] + 1)
    end for
end for
```

Algorithm 5 Backward BFS with COC

```
Input: grid ∈ ℝ^{m×n}
Output: dist ∈ ℝ^{m×n}
open ← PriorityQueue()
for each p ∈ grid do
    if grid[p] is 0 then
        p.dis ← 0,  p.coc ← p
        open.push(p)
    end if
end for
while not open.empty() do
    c ← open.front(),  open.pop()
    closed.insert(c)
    dist[c] ← c.dis
    for each n ∈ c.neighbours not in closed do
        if getDist(n, c.coc) < n.dis then
            n.dis ← getDist(n, c.coc),  n.coc ← c.coc
            open.push(n)
        end if
    end for
end while
```

Algorithm 6 Distance Transform 1D

Input: $f : \mathbb{R} \to \mathbb{R}$
Output: $dist \in \mathbb{R}^n$
$k \leftarrow 0$ ▷ Index of rightmost parabola in lower envelope
$v[0] \leftarrow 0$ ▷ Locations of parabolas in lower envelope
$z[0] \leftarrow -\infty,\ z[1] \leftarrow +\infty$ ▷ Locations of boundaries between parabolas
for $q = 1$ **to** $n - 1$ **do**
  $s \leftarrow ((f(q) + q^2) - (f(v[k]) + v[k]^2))/(2q - 2v[k])$
  while $s \leq z[k]$ **do**
    $k \leftarrow k - 1$
    $s \leftarrow ((f(q) + q^2) - (f(v[k]) + v[k]^2))/(2q - 2v[k])$
  end while
  $k \leftarrow k + 1$
  $v[k] \leftarrow q$
  $z[k] \leftarrow s,\ z[k+1] \leftarrow +\infty$
end for
$k \leftarrow 0$
for $q = 0$ **to** $n - 1$ **do**
  while $z[k+1] < q$ **do**
    $k \leftarrow k + 1$
  end while
  $dist[q] \leftarrow (q - v[k]^2) + f(v[k])$
end for

Algorithm 7 Incremental Backward BFS

```
Input: grid ∈ ℝ^{m×n}, insertQueue, deleteQueue
Output: dist ∈ ℝ^{m×n}
while not insertQueue.empty() do
    p ← insertQueue.front()
    insertQueue.pop()
    deleteFromDLL(p.coc, p)
    p.dis ← 0, p.coc ← p
    insertToDLL(p.coc, p)
    updateQueue.push(p)
end while
while not deleteQueue.empty() do
    p ← deleteQueue.front()
    deleteQueue.pop()
    for each c ∈ p.dll do
        deleteFrom(c.coc, c)
        c.dis ← ∞, c.coc ← IP
        for each n ∈ c.neighbours do
            if n.coc exists and getDist(n.coc, c) < c.dis then
                c.dis ← getDist(n.coc, c), c.coc ← n.coc
            end if
        end for
        insertToDLL(c.coc, c)
        if c.coc is not IP then
            updateQueue.push(c)
        end if
    end for
end while
```

Algorithm 8 Incremental Backward BFS Part II

Input: $grid \in \mathbb{R}^{m\times n},\ insertQueue,\ deleteQueue$
Output: $dist \in \mathbb{R}^{m\times n}$
while not $updateQueue$.empty() **do**
  $\boldsymbol{c} \leftarrow updateQueue.\text{front}(),\ updateQueue.\text{pop}()$
  for each $\boldsymbol{n} \in \boldsymbol{c}.neighbours$ **do**
    if $\text{getDist}(\boldsymbol{c}.coc, \boldsymbol{n}) < \boldsymbol{n}.dis$ **then**
      $\boldsymbol{n}.dis \leftarrow \text{getDist}(\boldsymbol{c}.coc, \boldsymbol{n})$
      $\text{deleteFromDLL}(\boldsymbol{n}.coc, \boldsymbol{n})$
      $\boldsymbol{n}.coc \leftarrow \boldsymbol{c}.coc$
      $\text{insertToDLL}(\boldsymbol{n}.coc, \boldsymbol{n})$
      $updateQueue.\text{push}(\boldsymbol{n})$
    end if
  end for
end while