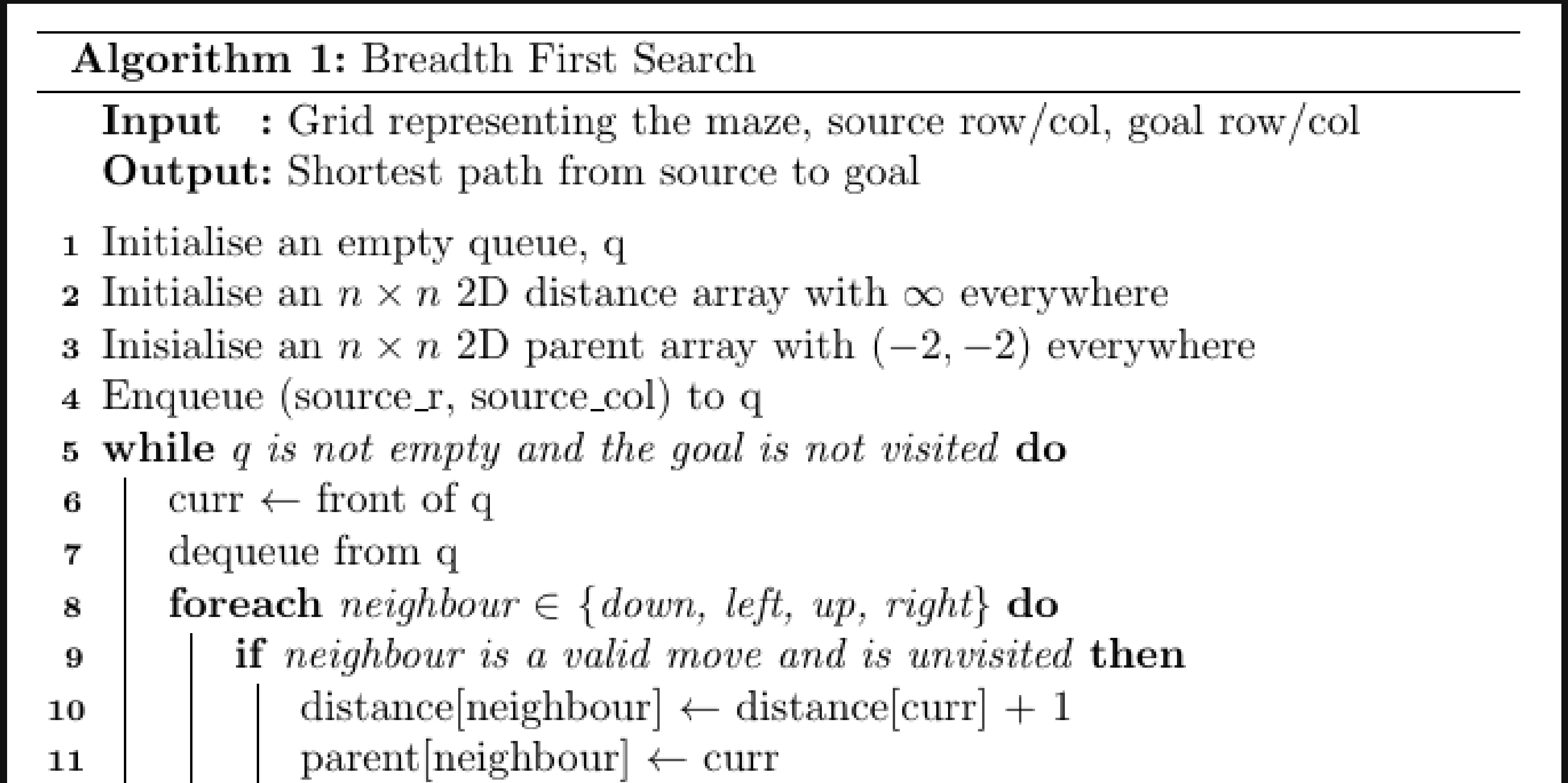
6.5.4 Pseudocode

When running on a grid where each step costs the same, the shortest path algorithm reduces to a normal breadth-first search. In the algorithm that follows, we make use of a queue to keep track of the fringe. We use a parent array to keep track of the row/column indices of the cell through which we travelled on the way to the current cell. The distance array keeps track of how long the path is to get from the source to the current cell. When we have found a path to the goal, we follow the parent array back to the source and reverse it to get the shortest path from the source to the goal.



Algorithm 1: Breadth First Search

```
Input  : Grid representing the maze, source row/col, goal row/col
Output: Shortest path from source to goal

Initialise an empty queue, q
Initialise an $n \times n$ 2D distance array with $\infty$ everywhere
Inisialise an $n \times n$ 2D parent array with $(-2, -2)$ everywhere
Enqueue (source_r, source_col) to q
while q is not empty and the goal is not visited do
    curr ← front of q
    dequeue from q
    foreach neighbour $\in$ {down, left, up, right} do
        if neighbour is a valid move and is unvisited then
            distance[neighbour] ← distance[curr] + 1
            parent[neighbour] ← curr
```