

## Pathfinding Algorithm

Input size:

n x n matrix

Pseudo code:

```
#BFS

bfs(arr[0..n-1][0..m-1])
    currX = startX
    currY = startY
    fail = False
    while True:
        if arr[currX - 1][currY] != "*":
            if arr[currX - 1][currY].visited == False:
                arr[currX - 1][currY].visited = True
                arr[currX - 1][currY].parentX = currX
                arr[currX - 1][currY].parentY = currY

            if arr[currX - 1][currY] == "e":
                eposX = currX - 1
                eposY = currY
                break

            queueX.append(currX - 1)
            queueY.append(currY)

        if arr[currX + 1][currY] != "*":
            if arr[currX + 1][currY].visited == False:
                arr[currX + 1][currY].visited = True
                arr[currX + 1][currY].parentX = currX
                arr[currX + 1][currY].parentY = currY

            if arr[currX + 1][currY] == "e":
                eposX = currX + 1
                eposY = currY
                break
```

```

        queueX.append(currX + 1)
        queueY.append(currY)

    if arr[currX][currY - 1] != "*":
        if arr[currX][currY - 1].visited == False:
            arr[currX][currY - 1].visited = True
            arr[currX][currY - 1].parentX = currX
            arr[currX][currY - 1].parentY = currY

            if arr[currX][currY - 1] == "e":
                eposX = currX
                eposY = currY - 1
                break

        queueX.append(currX)
        queueY.append(currY - 1)

    if arr[currX][currY + 1] != "*":
        if arr[currX][currY + 1].visited == False:
            arr[currX][currY + 1].visited = True
            arr[currX][currY + 1].parentX = currX
            arr[currX][currY + 1].parentY = currY

            if arr[currX][currY + 1] == "e":
                eposX = currX
                eposY = currY + 1
                break

        queueX.append(currX)
        queueY.append(currY + 1)

    currX = queueX.pop()
    currY = queueY.pop()

    if currX == NULL
        fail = True
        break

    if fail == False:
        return eposX, eposY
    else:
        return False

```

Basic Operation:

comparison (checking if a position is traversable)

Efficiency Class:

The image shows a handwritten derivation on lined paper. At the top, there are several expressions for a double summation of a constant value 4 over a grid of size n by n. The first row contains four expressions:  $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 4$ ,  $4 \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1$ ,  $4 \sum_{i=0}^{n-1} (n - \cancel{j - 0})$ , and  $4 \sum_{i=0}^{n-1} n$ . The second row shows  $4n \sum_{i=0}^{n-1} 1$  and  $4n (n - \cancel{j - 0}) = 4n * n = 4n^2$ . The third row shows the limit as n approaches infinity of the expression  $\frac{4n^2}{n^2} = 4$ , followed by the complexity class  $\Theta(n^2)$ .

$$\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 4 \quad 4 \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} 1 \quad 4 \sum_{i=0}^{n-1} (n - \cancel{j - 0}) \quad 4 \sum_{i=0}^{n-1} n$$
$$4n \sum_{i=0}^{n-1} 1 \quad 4n (n - \cancel{j - 0}) = 4n * n = 4n^2$$
$$\lim_{n \rightarrow \infty} \frac{4n^2}{n^2} = 4 \quad \Theta(n^2)$$