

$$S = (4, 2)$$

$$F = (1, 1)$$

Visited

$$\text{cur} = (4, 1)$$

$$\text{queue} = [(5, 2) (4, 3) (3, 2)]$$

$$(5, 1) (3, 1) (4, 0) (4, 2)$$

$$(4, 2) \rightarrow (4, 1) \rightarrow (4, 2)$$

```

private List<FullMapNode> continuousPathBFS(
    Map<String, FullMapNode> parent,
    Map<String, Integer> score) {
    parent.put(startKey, value: null);
    score.put(startKey, value: 0);

    while (!queue.isEmpty()) {
        FullMapNode current = queue.poll();
        String cKey = keyOf.apply(current);

        if (cKey.equals(finishKey)) {
            break;
        }

        for (FullMapNode nb : gameHelper.getNeighbours4(current)) {
            String nKey = keyOf.apply(nb);
            if (!isPassable(nb)) continue;
            if (parent.get(nKey) == null)
            int newScore = score.get(cKey);
            if (goals.contains(nb)) {
                newScore++; // reward if path passes through goal
            }
            // if neighbor not visited OR new path has better score - explore
            if (newScore > score.getOrDefault(nKey, -1)) {
                parent.put(nKey, cKey);
                parent.put(nKey, cKey);
                score.put(nKey, newScore);
                queue.add(nb);
            }
        }
    }

    if (!visited.contains(finishKey)) throw new IllegalArgumentException();

    LinkedList<FullMapNode> path = new LinkedList<>();
    String walk = finishKey;
    while (walk != null) {
        FullMapNode node = byKey.get(walk);
        path.addFirst(node);
        walk = parent.get(walk);
    }
}

```

$\rightarrow \text{cur} = B$

$\text{neighbours} = [D, A]$

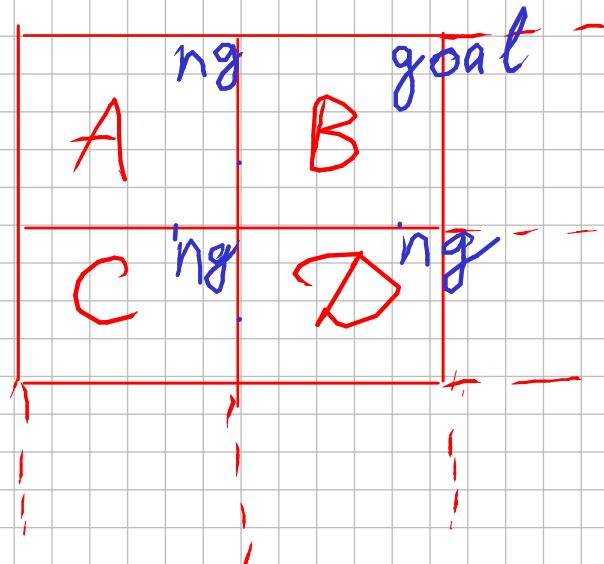
$nB = D$

$\text{new Score} = 1 + 0 = 1$

Start = D Finish = A

```
// if neighbor not visited OR new path has better score - explore  
if (!visited.contains(nKey), newScore > score.getOrDefault(nKey, -1)) {  
    1) visited.add(nKey);  
    2) parent.put(nKey, cKey);  
    3) score.put(nKey, newScore);  
    4) queue.add(nb);  
}
```

$$P = \begin{bmatrix} D: \text{null} & B:D \\ C:D & A:C \end{bmatrix}$$



$$Q = [\quad A \quad]$$

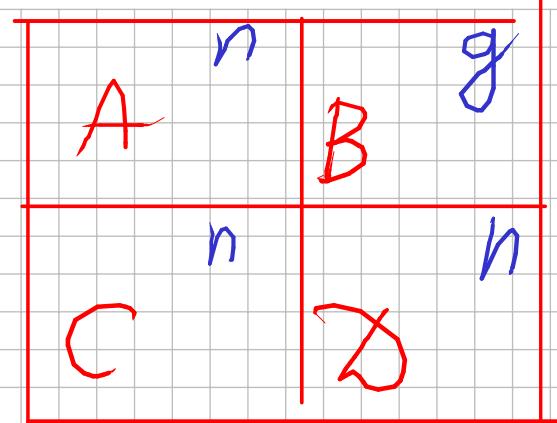
$$V = [D, C, B, A]$$

$$\text{Score} = [(D, 0), (C, 0), (B, 1), (A, 0)]$$

$$\text{cur} = B \quad nb = D$$

Start = D Finish = A

```
228     String startKey = keyOf.apply(start);
229     String finishKey = keyOf.apply(finish);
230     queue.add(start);
231     visited.add(startKey);
232     parent.put(startKey, value: null);
233     score.put(startKey, value: 0);
234
235
236
237     while (!queue.isEmpty()) {
238         FullMapNode current = queue.poll();
239         String cKey = keyOf.apply(current); = A
240
241         if (cKey.equals(finishKey)) {
242             break;
243         } nbs = [ ]
244
245         for (FullMapNode nb : gameHelper.getNeighbours4(current)) {
246             String nKey = keyOf.apply(nb); =
247             if (!isPassable(nb)) continue;
248             if (visited.contains(nKey)) continue;
249             int newScore = score.get(cKey);
250             if (goals.contains(nb)) {
251                 newScore++; // reward if path passes through goal
252             }
253             // if neighbor not visited OR new path has better score - explore
254             if [newScore > score.getOrDefault(nKey, -1)] {
255                 visited.add(nKey);
256                 parent.put(nKey, cKey);
257                 score.put(nKey, newScore);
258                 queue.add(nb);
259             }
260         }
261     }
```



$$Q = []$$

$$V = [D, C, B, A]$$

$$\text{Score} = D:0$$

$$C:0 \quad B:1 \quad A:0$$

$$P = [D:\text{null}, C:D, B:D]$$

$$A:C$$

D → C → A