Lab 3: Coin Collection Robot

20226758 - Jingqi Fan

May 26, 2024

1 Algorithm

1.1 Solve Maze Coin

This method computes the maximum number of coins that can be collected using a dynamic programming approach. Firstly, a DP table of the same size as the maze is created. The value in each cell will represent the maximum coins that can be collected from that cell to the bottom-right of the maze. The bottom-right corner of the maze is initialized based on whether it contains a coin. Then the table is filled from the bottom-right to the top-left. If a cell is not a wall, it computes the maximum coins from the two possible moves. If it does not meet a wall and pass the coin place, the algorithm updates the number of coins.

```
49 @    public static int solveMazeCoins(MazeField[][] maze) {
50          int rows = maze.length;
51          int cols = maze[0].length;
52          dp = new int[rows][cols];
53          dp[rows-1][cols-1] = (maze[rows-1][cols-1] == MazeField.COIN) ? 1 : 0;
54          for (int row = rows-1; row >= 0; row--) {
55              for (int col = cols-1; col >= 0; col--) {
56                  if (maze[row][col] == MazeField.WALL) continue;
57                  int fromLeft = (col < cols - 1 && dp[row][col+1] != -1) ?
                     dp[row][col+1] : -1;
58                  int fromBelow = (row < rows - 1 && dp[row+1][col] != -1) ?
                     dp[row+1][col] : -1;
59                  if (fromLeft != -1)
60                      dp[row][col] = Math.max(dp[row][col], fromLeft + (maze[row][col] ==
                          MazeField.COIN ? 1 : 0));
61                  if (fromBelow != -1)
62                      dp[row][col] = Math.max(dp[row][col], fromBelow + (maze[row][col]
                         == MazeField.COIN ? 1 : 0));
63              }
64          }
65          return dp[0][0];
66      }
```

Figure 1: solveMazeCoins

1.2 Solve Maze Path

This method reconstructs the path taken to collect the maximum coins, using the information stored in the DP table. Firstly, starting from the top-left corner of the maze, the method determines the next best move (right or down) by comparing the values of the respective cells in the DP table. This continues until the bottom-right corner of the maze is reached. Finally, a list of MazeStep enums that describe the path taken to collect the maximum coins.

```java
    public static List<MazeStep> solveMazePath(MazeField[][] maze) {
        List<MazeStep> path = new ArrayList<>();
        int row = 0;
        int col = 0;
        while (row < maze.length - 1 || col < maze[0].length - 1) {
            if (col < maze[0].length - 1 && dp[row][col+1] >= dp[row+1][col]) {
                path.add(MazeStep.UP);
                col++;
            } else {
                path.add(MazeStep.LEFT);
                row++;
            }
        }
        return path;
    }
}
```

Figure 2: solveMazePath

2 Complexity Analysis

Theorem 1 *Given that the number of row n and the number of colomn m, if $n > 1$ and $m > 1$, the complexity of collecting coin robot is $\mathcal{O}(nm)$*

To prove Theorem 1, we need the following lemmas.

Lemma 1 *Given that the number of row n and the number of colomn m, the complexity of solveMazeCoin is $\mathcal{O}(nm)$*

Proof This method contains a nested loop structure where each loop iterates over the dimensions of the maze, which is $n \times m$. Each cell in the dp array is visited exactly once. Inside each loop iteration, the algorithm performs a constant amount of work, which includes checking boundary conditions, updating the dp array based on the presence of a coin, and comparing values to determine the maximum coins collectable from that point. Therefore, the time complexity of this method is $\mathcal{O}(nm)$. □

Lemma 2 *Given that the number of row n and the number of colomn m, the complexity of solveMazePath is $\mathcal{O}(n+m)$*

Proof This method involves iterating from the top-left corner of the maze to the bottom-right. At each step, the algorithm makes a decision in constant time based on the values in the dp table to move either right or down. In the worst case, this involves moving across all rows and all columns once. Therefore, this is bounded by $n+m$. □

Proof of Theorem 1 Combining Lemma 1 and Lemma 2, the comlexity of the whole algorithm is

$$\begin{aligned} \text{The complexity} &= \max\{\mathcal{O}(nm), \mathcal{O}(n+m)\} \\ &\leq \mathcal{O}(nm) \end{aligned} \tag{1}$$

The inequality holds because $n > 1$ and $m > 1$. Thus, we have proved the theorem. □