

542、01矩阵

我们把每个非0的数都进行 bfs 搜索，那么时间复杂度是 $n*n*O(bfs)$ ，超时了。

```
1 struct {
2     int i;
3     int j;
4     int count;
5 }queue[1000];
6 int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
7 int bfs(int **mat, int m, int n, int a, int b)
8 {
9     int rear = 0, front = 0;
10    queue[front].i = a;
11    queue[front].j = b;
12    queue[front++].count = 0;
13    while (front != rear) {
14        int x = queue[rear].i;
15        int y = queue[rear].j;
16        int count = queue[rear].count;
17        rear = (rear + 1) % 1000;
18        for (int i = 0; i < 4; i++) {
19            int x1 = x + dir[i][0];
20            int y1 = y + dir[i][1];
21            if (0 <= x1 && x1 < m && 0 <= y1 && y1 < n) {
22                if (mat[x1][y1] == 0)
23                    return count + 1;
24                else {
25                    queue[front].i = x1;
26                    queue[front].j = y1;
27                    queue[front].count = count + 1;
28                    front = (front + 1) % 1000;
29                }
30            }
31        }
32    }
33    return 0;
34 }
35 int** updateMatrix(int** mat, int matSize, int* matColSize, int* returnSize,
36 int** returnColumnSizes){
37     *returnSize = matSize;
38     *returnColumnSizes = (int *)malloc(sizeof(int) * matSize);
39     for (int i = 0; i < matSize; i++)
40         (*returnColumnSizes)[i] = matColSize[0];
41     int **ans = (int *)malloc(sizeof(int *) * matSize);
42     for (int i = 0; i < matSize; i++) {
43         int *temp = (int *)malloc(sizeof(int) * matColSize[0]);
44         for (int j = 0; j < matColSize[0]; j++) {
45             if (mat[i][j] == 0)
46                 temp[j] = 0;
47             else
48                 temp[j] = bfs(mat, matSize, matColSize[0], i, j);
49         }
50         ans[i] = temp;
51     }
```

```

50     }
51     return ans;
52 }

```

那么有什么办法吗？我们可以用0来 bfs 搜索1，而且是所有的0一起来搜索，先把所有的0都入队，然后查找1。因为我们的1是特定的，但是随便哪个0都是可行的。

```

1  struct {
2      int i;
3      int j;
4      int count;
5  }queue[10000];
6  int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
7  int** updateMatrix(int** mat, int matSize, int* matColSize, int* returnSize,
8  int** returnColumnSizes){
9      int rear = 0, front = 0;
10     *returnSize = matSize;
11     *returnColumnSizes = (int *)malloc(sizeof(int) * matSize);
12     for (int i = 0; i < matSize; i++)
13         (*returnColumnSizes)[i] = matColSize[0];
14     int sign[matSize][matColSize[0]];
15     memset(sign, 0, sizeof(sign));
16     int **ans = (int **)malloc(sizeof(int *) * matSize);
17     for (int i = 0; i < matSize; i++)
18         ans[i] = (int *)malloc(sizeof(int) * matColSize[0]);
19     for (int i = 0; i < matSize; i++) {
20         for (int j = 0; j < matColSize[0]; j++) {
21             if (mat[i][j] == 0) {
22                 sign[i][j] = 1;
23                 ans[i][j] = 0;
24                 queue[rear].i = i;
25                 queue[rear].j = j;
26                 queue[rear].count = 0;
27             }
28         }
29     }
30     while (front < rear) {
31         int x = queue[rear].i;
32         int y = queue[rear].j;
33         int count = queue[rear].count;
34         for (int i = 0; i < 4; i++) {
35             int x1 = x + dir[i][0];
36             int y1 = y + dir[i][1];
37             if (0 <= x1 && x1 < matSize && 0 <= y1 && y1 < matColSize[0] &&
38             sign[x1][y1] == 0) {
39                 ans[x1][y1] = count + 1;
40                 sign[x1][y1] = 1;
41                 queue[rear].i = x1;
42                 queue[rear].j = y1;
43                 queue[rear].count = count + 1;
44             }
45         }
46     }
47     return ans;
48 }

```

是动态规划它不香了吗？我们设 $dp[i][j]$ 表示位置 (i, j) 到最近 0 的距离，那么 $dp[i][j] = \min(dp[i][j - 1], dp[i - 1][j], dp[i + 1][j], dp[i][j + 1]) + 1$ 。

```
1  int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
2  int** updateMatrix(int** mat, int matSize, int* matColSize, int* returnSize,
3  int** returnColumnSizes){
4      *returnSize = matSize;
5      *returnColumnSizes = (int *)malloc(sizeof(int) * matSize);
6      int **dp = (int **)malloc(sizeof(int *) * matSize);
7      for (int i = 0; i < matSize; i++) {
8          dp[i] = (int *)malloc(sizeof(int) * matColSize[0]);
9          (*returnColumnSizes)[i] = matColSize[0];
10     }
11     for (int i = 0; i < matSize; i++) {
12         for (int j = 0; j < matColSize[0]; j++) {
13             dp[i][j] = INT_MAX - 1;
14         }
15     }
16     // 从左上到右下
17     for (int i = 0; i < matSize; i++) {
18         for (int j = 0; j < matColSize[0]; j++) {
19             if (mat[i][j] == 0)
20                 dp[i][j] = 0;
21             else {
22                 for (int k = 0; k < 4; k++) {
23                     int x = i + dir[k][0];
24                     int y = j + dir[k][1];
25                     if (0 <= x && x < matSize && 0 <= y && y <
26                         matColSize[0])
27                         dp[i][j] = fmin(dp[i][j], dp[x][y] + 1);
28                 }
29             }
30         }
31     }
32     // 从右下到左上
33     for (int i = matSize - 1; i >= 0; i--) {
34         for (int j = matColSize[0] - 1; j >= 0; j--) {
35             for (int k = 0; k < 4; k++) {
36                 int x = i + dir[k][0];
37                 int y = j + dir[k][1];
38                 if (0 <= x && x < matSize && 0 <= y && y <
39                     matColSize[0])
40                     dp[i][j] = fmin(dp[i][j], dp[x][y] + 1);
41             }
42         }
43     }
44     return dp;
45 }
```

为什么要从左上到右下然后又从右下到左上呢？我们求 $dp[0][0]$ 的时候，很显然得不到正确的值，因为它周围的值还没正确呢，要一直到遇见一个 0，这个值就正确了，然后才能推出后面的全都正确。那么我们再反过来，又能把前面的全部都搞正确。

994、腐烂的橘子

我们和上面一题一样，把所有腐烂的都入队，因为它们可以同时来腐烂橘子。然后找和这个腐烂橘子整体最远的，也就是腐烂需要的天数。最后再遍历一下，还有没腐烂的就返回-1。

```
1 struct {
2     int i;
3     int j;
4     int day;
5 }queue[1000];
6 int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
7 int orangesRotting(int** grid, int gridSize, int* gridColSize){
8     int rear = 0, front = 0;
9     // 将所有已经腐烂的橘子入队
10    for (int i = 0; i < gridSize; i++) {
11        for (int j = 0; j < gridColSize[0]; j++) {
12            if (grid[i][j] == 2) {
13                queue[front].i = i;
14                queue[front].j = j;
15                queue[front++].day = 0;
16            }
17        }
18    }
19    int day;
20    // 开始腐烂橘子
21    while (front > rear) {
22        int x = queue[rear].i;
23        int y = queue[rear].j;
24        day = queue[rear++].day;
25        for (int i = 0; i < 4; i++) {
26            int x1 = x + dir[i][0];
27            int y1 = y + dir[i][1];
28            // 如果周围的在范围内并且是没有腐烂的橘子，就入队（代表腐烂）
29            if (0 <= x1 && x1 < gridSize && 0 <= y1 && y1 < gridColSize[0]
&& grid[x1][y1] == 1) {
30                grid[x1][y1] = 2;
31                queue[front].i = x1;
32                queue[front].j = y1;
33                queue[front++].day = day + 1;
34            }
35        }
36    }
37    // 遍历是否有没有腐烂的橘子
38    for (int i = 0; i < gridSize; i++) {
39        for (int j = 0; j < gridColSize[0]; j++) {
40            if (grid[i][j] == 1)
41                return -1;
42        }
43    }
44    return day;
45 }
```

