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。

```c
struct {
    int i;
    int j;
    int count;
}queue[1000];
int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
int bfs(int **mat, int m, int n, int a, int b)
{
    int rear = 0, front = 0;
    queue[front].i = a;
    queue[front].j = b;
    queue[front++].count = 0;
    while (front != rear) {
        int x = queue[rear].i;
        int y = queue[rear].j;
        int count = queue[rear].count;
        rear = (rear + 1) % 1000;
        for (int i = 0; i < 4; i++) {
            int x1 = x + dir[i][0];
            int y1 = y + dir[i][1];
            if (0 <= x1 && x1 < m && 0 <= y1 && y1 < n) {
                if (mat[x1][y1] == 0)
                    return count + 1;
                else {
                    queue[front].i = x1;
                    queue[front].j = y1;
                    queue[front].count = count + 1;
                    front = (front + 1) % 1000;
                }
            }
        }
    }
    return 0;
}
int** updateMatrix(int** mat, int matSize, int* matColSize, int* returnSize, int** returnColumnSizes){
    *returnSize = matSize;
    *returnColumnSizes = (int *)malloc(sizeof(int) * matSize);
    for (int i = 0; i < matSize; i++)
        (*returnColumnSizes)[i] = matColSize[0];
    int **ans = (int *)malloc(sizeof(int *) * matSize);
    for (int i = 0; i < matSize; i++) {
        int *temp = (int *)malloc(sizeof(int) * matColSize[0]);
        for (int j = 0; j < matColSize[0]; j++) {
            if (mat[i][j] == 0)
                temp[j] = 0;
            else
                temp[j] = bfs(mat, matSize, matColSize[0], i , j);
        }
        ans[i] = temp;
    }
    return ans;
}
```

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。

```
struct {
    int i;
    int j;
    int count;
}queue[10000];
int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
int** updateMatrix(int** mat, int matSize, int* matColSize, int* returnSize, int** returnColumnSizes){
    int rear = 0, front = 0;
    *returnSize = matSize;
    *returnColumnSizes = (int *)malloc(sizeof(int) * matSize);
    for (int i = 0; i < matSize; i++)
        (*returnColumnSizes)[i] = matColSize[0];
    int sign[matSize][matColSize[0]];
    memset(sign, 0, sizeof(sign));
    int **ans = (int **)malloc(sizeof(int *) * matSize);
    for (int i = 0; i < matSize; i++)
        ans[i] = (int *)malloc(sizeof(int) * matColSize[0]);
    for (int i = 0; i < matSize; i++) {
        for (int j = 0; j < matColSize[0]; j++) {
            if (mat[i][j] == 0) {
                sign[i][j] = 1;
                ans[i][j] = 0;
                queue[front].i = i;
                queue[front].j = j;
                queue[front++].count = 0;
            }
        }
    }
    while (front > rear) {
        int x = queue[rear].i;
        int y = queue[rear].j;
        int count = queue[rear++].count;
        for (int i = 0; i < 4; i++) {
            int x1 = x + dir[i][0];
            int y1 = y + dir[i][1];
            if (0 <= x1 && x1 < matSize && 0 <= y1 && y1 < matColSize[0] && sign[x1][y1] == 0) {
                ans[x1][y1] = count + 1;
                sign[x1][y1] = 1;
                queue[front].i = x1;
                queue[front].j = y1;
                queue[front++].count = count + 1;
            }
        }
    }
    return ans;
}
```

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`。

```c
int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
int** updateMatrix(int** mat, int matSize, int* matColSize, int* returnSize, int** returnColumnSizes){
    *returnSize = matSize;
    *returnColumnSizes = (int *)malloc(sizeof(int) * matSize);
    int **dp = (int **)malloc(sizeof(int *) * matSize);
    for (int i = 0; i < matSize; i++) {
        dp[i] = (int *)malloc(sizeof(int) * matColSize[0]);
        (*returnColumnSizes)[i] = matColSize[0];
    }
    for (int i = 0; i < matSize; i++) {
        for (int j = 0; j < matColSize[0]; j++) {
            dp[i][j] = INT_MAX - 1;
        }
    }
    // 从左上到右下
    for (int i = 0; i < matSize; i++) {
        for (int j = 0; j < matColSize[0]; j++) {
            if (mat[i][j] == 0)
                dp[i][j] = 0;
            else {
                for (int k = 0; k < 4; k++) {
                    int x = i + dir[k][0];
                    int y = j + dir[k][1];
                    if (0 <= x && x < matSize && 0 <= y && y < matColSize[0])
                        dp[i][j] = fmin(dp[i][j], dp[x][y] + 1);
                }
            }
        }
    }
    // 从右下到左上
    for (int i = matSize - 1; i >= 0; i--) {
        for (int j = matColSize[0] - 1; j >= 0; j--) {
            for (int k = 0; k < 4; k++) {
                    int x = i + dir[k][0];
                    int y = j + dir[k][1];
                    if (0 <= x && x < matSize && 0 <= y && y < matColSize[0])
                        dp[i][j] = fmin(dp[i][j], dp[x][y] + 1);
                }
        }
    }
    return dp;
}
```

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。

```c
struct {
    int i;
    int j;
    int day;
}queue[1000];
int dir[4][2] = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
int orangesRotting(int** grid, int gridSize, int* gridColSize){
    int rear = 0, front = 0;
    // 将所有已经腐烂的橘子入队
    for (int i = 0; i < gridSize; i++) {
        for (int j = 0; j < gridColSize[0]; j++) {
            if (grid[i][j] == 2) {
                queue[front].i = i;
                queue[front].j = j;
                queue[front++].day = 0;
            }
        }
    }
    int day;
    // 开始腐烂橘子
    while (front > rear) {
        int x = queue[rear].i;
        int y = queue[rear].j;
        day = queue[rear++].day;
        for (int i = 0; i < 4; i++) {
            int x1 = x + dir[i][0];
            int y1 = y + dir[i][1];
            // 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）
            if (0 <= x1 && x1 < gridSize && 0 <= y1 && y1 < gridColSize[0] && grid[x1][y1] == 1) {
                grid[x1][y1] = 2;
                queue[front].i = x1;
                queue[front].j = y1;
                queue[front++].day = day + 1;
            }
        }
    }
    // 遍历是否有没有腐烂的橘子
    for (int i = 0; i < gridSize; i++) {
        for (int j = 0; j < gridColSize[0]; j++) {
            if (grid[i][j] == 1)
                return -1;
        }
    }
    return day;
}
```