994. Rotting Oranges

You are given an m x n grid where each cell can have one of three values:

- 0 representing an empty cell,
- 1 representing a fresh orange, or
- 2 representing a rotten orange.

Every minute, any fresh orange that is **4-directionally adjacent** to a rotten orange becomes rotten.

Return *the minimum number of minutes that must elapse until no cell has a fresh orange*. If *this is impossible, return* -1.

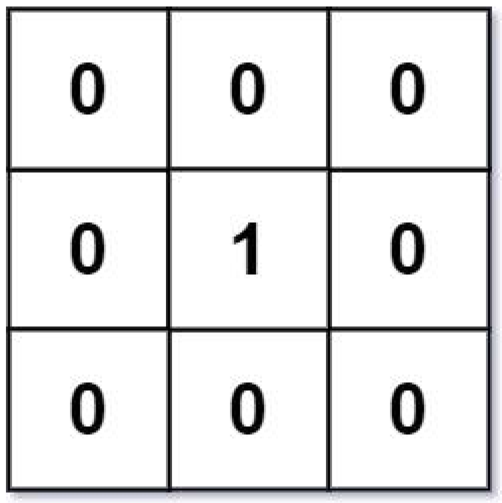
C/C++
```cpp
class Solution {
public:
    bool isValidCase(vector<vector<int>>& grid,int x,int y){
        int n = grid.size();
        int m = grid[0].size();
        return (x<n && x>=0 && y<m && y>=0);
    }
    void fillQueueWithRotten(vector<vector<int>>&
grid,queue<pair<int,int>> &que){
        int n = grid.size();
        int m = grid[0].size();
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(grid[i][j]==2) que.push({i,j});
            }
        }
    }
    bool isEdgeCase(vector<vector<int>>& grid){
        int n = grid.size();
        int m = grid[0].size();
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(grid[i][j]==1) return true;
            }
        }
        return false;
    }
    int orangesRotting(vector<vector<int>>& grid) {
        queue<pair<int,int>> que;
        fillQueueWithRotten(grid,que);
        if(!isEdgeCase(grid)) return 0;
        int ans = 0;
        vector<int> dir = {-1,0,1,0,-1};
        while(!que.empty()){
            auto [x,y] = que.front();que.pop();
            ans = max(ans,grid[x][y]);
            for(int i=1;i<5;i++){
                int _x = x + dir[i];
                int _y = y + dir[i-1];
                if(isValidCase(grid,_x,_y) && grid[_x][_y]==1){
                    grid[_x][_y] = grid[x][y]+1;
                    que.push({_x,_y});
                }
            }
        }
        if(isEdgeCase(grid)) return -1;
        return ans-2;
    }
};
```

542. 01 Matrix

Given an m x n binary matrix mat, return *the distance of the nearest* 0 *for each cell*.

The distance between two cells sharing a common edge is 1.

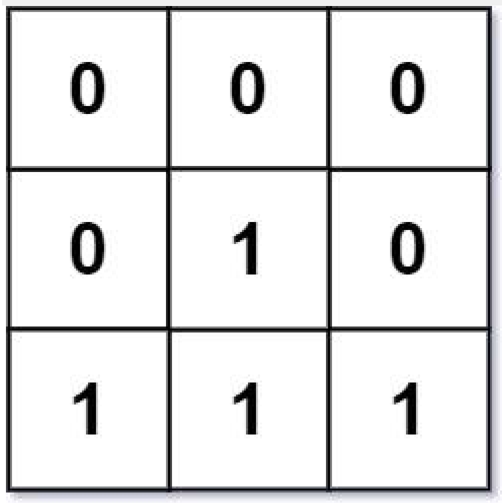
Example 1:



0	0	0
0	1	0
0	0	0

Input: mat = [[0,0,0],[0,1,0],[0,0,0]]
Output: [[0,0,0],[0,1,0],[0,0,0]]

Example 2:



0	0	0
0	1	0
1	1	1

Input: mat = [[0,0,0],[0,1,0],[1,1,1]]
Output: [[0,0,0],[0,1,0],[1,2,1]]

Constraints:

- m == mat.length
- n == mat[i].length
- 1 <= m, n <= 10_4
- 1 <= m * n <= 10_4
- mat[i][j] is either 0 or 1.
- There is at least one 0 in mat.

Note: This question is the same as 1765: https://leetcode.com/problems/map-of-highest-peak/

C/C++

```
class Solution {
public:
    bool isEdgeCase(vector<vector<int>>& mat){
        int n = mat.size();
        int m = mat[0].size();
        if(n==1 && m==1) return true;
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(mat[i][j]==1) return false;
            }
        }
        return true;
    }
    bool isboundaryCase(vector<vector<int>>& mat,int i,int j){
        int n = mat.size(),m=mat[0].size();
        return (i<0 || i>=n || j<0 || j>=m || mat[i][j]!=1);
    }
    void fillQueueWithZeros(vector<vector<int>>&
mat,queue<pair<int,int>> &que){
        int n = mat.size(),m=mat[0].size();
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(mat[i][j]==0) que.push({i,j});
            }
        }
    }
    void removeUnnecessaryTime(vector<vector<int>>& mat){
        int n = mat.size(),m=mat[0].size();
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(mat[i][j]!=0) mat[i][j]--;
            }
        }
    }
    void printMatrix(vector<vector<int>>& mat){
        int n = mat.size(),m=mat[0].size();
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                cout<<mat[i][j]<<" ";
            }
            cout<<endl;
        }
    }
    vector<vector<int>> updateMatrix(vector<vector<int>>& mat) {
        if(isEdgeCase(mat)) return mat;
        queue<pair<int,int>> que;
        fillQueueWithZeros(mat,que);
        vector<int> dir = {-1,0,1,0,-1};
        while(!que.empty()){
            auto [x,y] = que.front();que.pop();
            for(int i=1;i<5;i++){
                int _x = x + dir[i];
                int _y = y + dir[i-1];
                if(!isboundaryCase(mat,_x,_y)){
                    mat[_x][_y] = mat[_x][_y] + (mat[x][y]==0 ? 1 :
mat[x][y]);
                    que.push({_x,_y});
                }
            }
        }
        // printMatrix(mat);
        removeUnnecessaryTime(mat);
        return mat;
    }
};
```

752. Open the Lock

You have a lock in front of you with 4 circular wheels. Each wheel has 10 slots: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'. The wheels can rotate freely and wrap around: for example we can turn '9' to be '0', or '0' to be '9'. Each move consists of turning one wheel one slot.

The lock initially starts at '0000', a string representing the state of the 4 wheels.

You are given a list of deadends dead ends, meaning if the lock displays any of these codes, the wheels of the lock will stop turning and you will be unable to open it.

Given a target representing the value of the wheels that will unlock the lock, return the minimum total number of turns required to open the lock, or -1 if it is impossible.

Example 1:

Input: deadends = ["0201","0101","0102","1212","2002"], target = "0202"
Output: 6
Explanation:
A sequence of valid moves would be "0000" -> "1000" -> "1100" -> "1200" -> "1201" -> "1202" -> "0202".
Note that a sequence like "0000" -> "0001" -> "0002" -> "0102" -> "0202" would be invalid, because the wheels of the lock become stuck after the display becomes the dead end "0102".

Example 2:

Input: deadends = ["8888"], target = "0009"
Output: 1
Explanation: We can turn the last wheel in reverse to move from "0000" -> "0009".

Example 3:

Input: deadends = ["8887","8889","8878","8898","8788","8988","7888","9888"], target = "8888"
Output: -1
Explanation: We cannot reach the target without getting stuck.

Constraints:

- 1 <= deadends.length <= 500
- deadends[i].length == 4
- target.length == 4
- target **will not be** in the list deadends.
- target and deadends[i] consist of digits only.

C/C++
```
class Solution {
public:
    int openLock(vector<string>& deadends, string target) {
        queue<pair<string,int>> que;
        unordered_set<string> vis,rest;
        for(string &s : deadends) rest.insert(s);
        que.push({"0000",0});
        while(que.size()>0){
            auto [node,ops] = que.front();
            que.pop();
            if(node==target) return ops;
            if(rest.count(node)) continue;
            if(vis.count(node)) continue;
            vis.insert(node);
            for(int i=0;i<4;i++){
                char temp = node[i];
                if(node[i]=='0'){
                    node[i] = '1';
                    que.push({node,ops+1});
                    node[i] = '9';
                    que.push({node,ops+1});
                }else if(node[i]=='9'){
                    node[i] = '0';
                    que.push({node,ops+1});
                    node[i] = '8';
                    que.push({node,ops+1});
                }else{
                    node[i]++;
                    que.push({node,ops+1});
                    node[i]-=2;
                    que.push({node,ops+1});
                }
                node[i]=temp;
            }
        }
        return -1;
    }
};
```

127. Word Ladder

A **transformation sequence** from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s_1 -> s_2 -> ... -> s_k such that:

- Every adjacent pair of words differs by a single letter.
- Every s_i for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
- s_k == endWord

Given two words, beginWord and endWord, and a dictionary wordList, return *the* ***number of words*** *in the* ***shortest transformation sequence*** *from* beginWord *to* endWord*, or* 0 *if no such sequence exists.*

Example 1:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is 5 words long.

Example 2:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.

Constraints:

- 1 <= beginWord.length <= 10
- endWord.length == beginWord.length
- 1 <= wordList.length <= 5000
- wordList[i].length == beginWord.length
- beginWord, endWord, and wordList[i] consist of lowercase English letters.
- beginWord != endWord
- All the words in wordList are **unique**

C/C++
```cpp
class Solution {
public:
    bool isEdgeCase(string beginWord, string endWord, vector<string>&
wordList){
        return
find(begin(wordList),end(wordList),beginWord)==end(wordList) ||
find(begin(wordList),end(wordList),endWord)==end(wordList);
    }
    void handleBeginWordNotFound(string beginWord,vector<string>&
wordList){

if(find(begin(wordList),end(wordList),beginWord)==end(wordList)){
            wordList.push_back(beginWord);
        }
    }
    bool handleEndWordNotFound(string endWord,vector<string>& wordList){
        if(find(begin(wordList),end(wordList),endWord)==end(wordList))
return true;
        return false;
    }
    bool isDifferByOne(string &a,string &b){
        int cnt = 0;
        for(int i=0;i<a.size();i++){
            if(a[i]!=b[i]) cnt++;
            if(cnt>1) return false;
        }
        return cnt==1;
    }
    int ladderLength(string beginWord, string endWord, vector<string>&
wordList) {
        if(handleEndWordNotFound(endWord,wordList)) return 0;
        handleBeginWordNotFound(beginWord,wordList);
        int n = wordList.size();
        vector<vector<int>> adj(n);
        for(int i=0;i<n;i++){
            for(int j=i+1;j<n;j++){
                if(isDifferByOne(wordList[i],wordList[j])){
                    adj[i].push_back(j);
                    adj[j].push_back(i);
                }
            }
        }
        auto it = find(begin(wordList),end(wordList),beginWord);
        int root = distance(begin(wordList),it);
        auto itend = find(begin(wordList),end(wordList),endWord);
        int target = distance(begin(wordList),itend);
        queue<int> que;
        vector<int> vis(n,0);
        que.push(root);
        vis[root] = 1;
        while(!que.empty()){
            int x = que.front();que.pop();
            if(x==target) return vis[x];
            for(auto &e : adj[x]){
                if(vis[e]==0){
                    vis[e]=vis[x]+1;
                    que.push(e);
                }
            }
        }
        return 0;
    }
};
```