|  |  |
| --- | --- |
| **Rotten Oranges in C++** | |
| #include<bits/stdc++.h>  using namespace std;  class Solution {  public:  //Function to find minimum time required to rot all oranges.  int orangesRotting(vector < vector < int >> & grid) {  // figure out the grid size  int n = grid.size();  int m = grid[0].size();  // store {{row, column}, time}  queue < pair < pair < int, int > , int >> q;  int vis[n][m];  int cntFresh = 0;  for (int i = 0; i < n; i++) {  for (int j = 0; j < m; j++) {  // if cell contains rotten orange  if (grid[i][j] == 2) {  q.push({{i, j}, 0});  // mark as visited (rotten) in visited array  vis[i][j] = 2;  }  // if not rotten  else {  vis[i][j] = 0;  }  // count fresh oranges  if (grid[i][j] == 1) cntFresh++;  }  }  int tm = 0;  // delta row and delta column  int drow[] = {-1, 0, +1, 0};  int dcol[] = {0, 1, 0, -1};  int cnt = 0;  // bfs traversal (until the queue becomes empty)  while (!q.empty()) {  int r = q.front().first.first;  int c = q.front().first.second;  int t = q.front().second;  tm = max(tm, t);  q.pop();  // exactly 4 neighbours  for (int i = 0; i < 4; i++) {  // neighbouring row and column  int nrow = r + drow[i];  int ncol = c + dcol[i];  // check for valid cell and  // then for unvisited fresh orange  if (nrow >= 0 && nrow < n && ncol >= 0 && ncol < m &&  vis[nrow][ncol] == 0 && grid[nrow][ncol] == 1) {  // push in queue with timer increased  q.push({{nrow, ncol}, t + 1});  // mark as rotten  vis[nrow][ncol] = 2;  cnt++;  }  }  }  // if all oranges are not rotten  if (cnt != cntFresh) return -1;  return tm;  }  };  int main() {  vector<vector<int>>grid{{0,1,2},{0,1,2},{2,1,1}};  Solution obj;  int ans = obj.orangesRotting(grid);  cout << ans << "\n";  return 0;  } |  |
| **Output:-**  1 | |