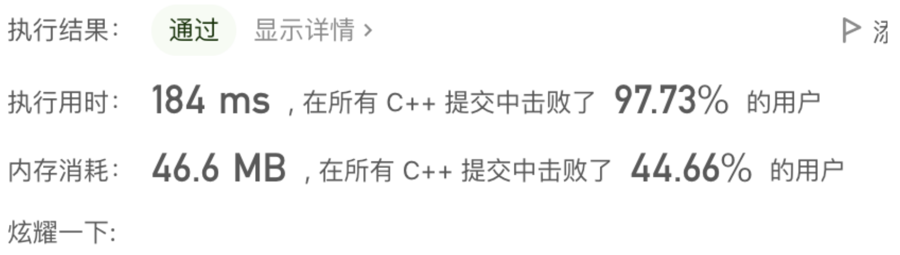
1108 Invalid IP Address

```go

func defangIPaddr(address string) string {
  res := strings.Replace(address, ".", "[.]", -1)
  return res
}

```

1135 Connecting Cities With Minimum Cost



执行结果：通过

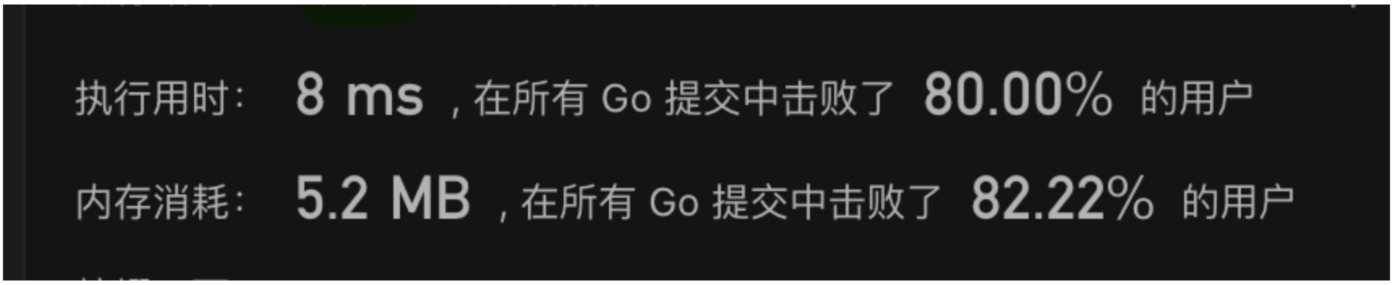
执行用时：184 ms，在所有 C++ 提交中击败了 97.73% 的用户

内存消耗：46.6 MB，在所有 C++ 提交中击败了 44.66% 的用户

炫耀一下：

```cpp
/*
  Kruskal 算法

*/
class WeightedUnionFind{
public:
    vector<int> sz;
    vector<int> id;

    WeightedUnionFind(int N) : sz(vector<int>(N, 1)) , id(vector<int>(N, 0)){
        for(int i = 0; i < N; i++)
            id[i] = i;
    }

    void uni(int p, int q){
        int pRoot = find(p);
        int qRoot = find(q);

        if(pRoot == qRoot)
            return;

        if(sz[pRoot] >= sz[qRoot]){
            sz[pRoot] += sz[qRoot];
            id[qRoot] = id[pRoot];
        }else{
            sz[qRoot] += sz[pRoot];
            id[pRoot] = id[qRoot];
        }
    }

    int find(int p){
        while(p != id[p]){
            id[p] = id[id[p]];
            p = id[p];
        }
        return p;
    }

    bool connected(int p, int q){
        return find(p) == find(q);
    }
};



class Edge{
public:
    int val;
    int thisEnd;
    int anotherEnd;

    Edge(int v, int t, int a) : val(v), thisEnd(t), anotherEnd(a){}
};

class Compare{
public:
    bool operator()(Edge e1, Edge e2){
        return e1.val > e2.val;
    }
};


class Solution {
public:
    int minimumCost(int n, vector<vector<int>>& connections) {
        priority_queue<Edge, vector<Edge>, Compare> pq;

        WeightedUnionFind wuf(n + 1);
        for(vector<int>& edge : connections){
            Edge myEdge(edge[2], edge[1], edge[0]);
            pq.push(myEdge);
        }

        queue<Edge> myQueue;

        int money = 0;
        while(!pq.empty() && myQueue.size() != n - 1){
            Edge edge = pq.top();
            pq.pop();

            int thisEnd = edge.thisEnd;
            int thatEnd = edge.anotherEnd;
            int val = edge.val;

            if(wuf.connected(thisEnd, thatEnd))
                continue;

            wuf.uni(thatEnd, thisEnd);

            myQueue.push(edge);
            money += val;
        }

        return myQueue.size() == n - 1? money : -1;
    }
};

```

1123 Lowest Common Ancestor of Deepest Leaves

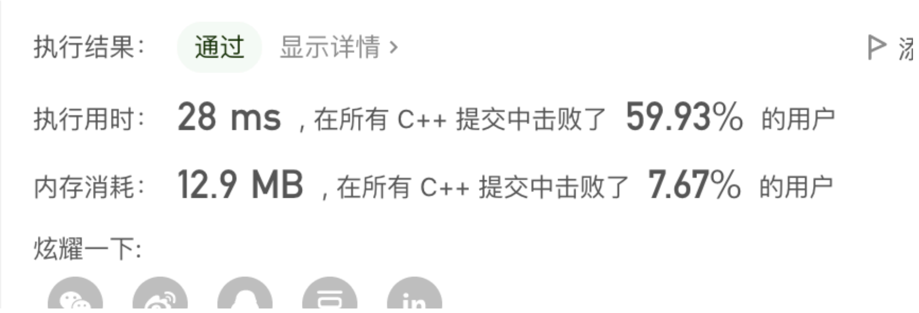


执行用时： 8 ms ，在所有 Go 提交中击败了 80.00% 的用户

内存消耗： 5.2 MB ，在所有 Go 提交中击败了 82.22% 的用户

```go
/*
  朴素的想法，求出左右的最大深度，如果一致，那就是它了！
  but still duplicated process, needs to improve on that
*/
func lcaDeepestLeaves(root *TreeNode) *TreeNode {
  if root == nil || (root.Left == nil && root.Right == nil){
    return root
  }

  left  := getMaxDepth(root.Left)
  right := getMaxDepth(root.Right)

  if left == right{
    return root
  }else{
    if left > right{
      return lcaDeepestLeaves(root.Left)
    }else{
      return lcaDeepestLeaves(root.Right)
    }
  }
}

func getMaxDepth(root *TreeNode) int{
  if root == nil{
    return 0
  }

  return 1 + max(getMaxDepth(root.Left), getMaxDepth(root.Right))
}

func max(a int, b int) int{
  if a < b{
    return b
  }

  return a
}
```

1143 Longest Common Subsequence



执行结果： 通过 显示详情 ›

执行用时： 28 ms ，在所有 C++ 提交中击败了 59.93% 的用户

内存消耗： 12.9 MB ，在所有 C++ 提交中击败了 7.67% 的用户

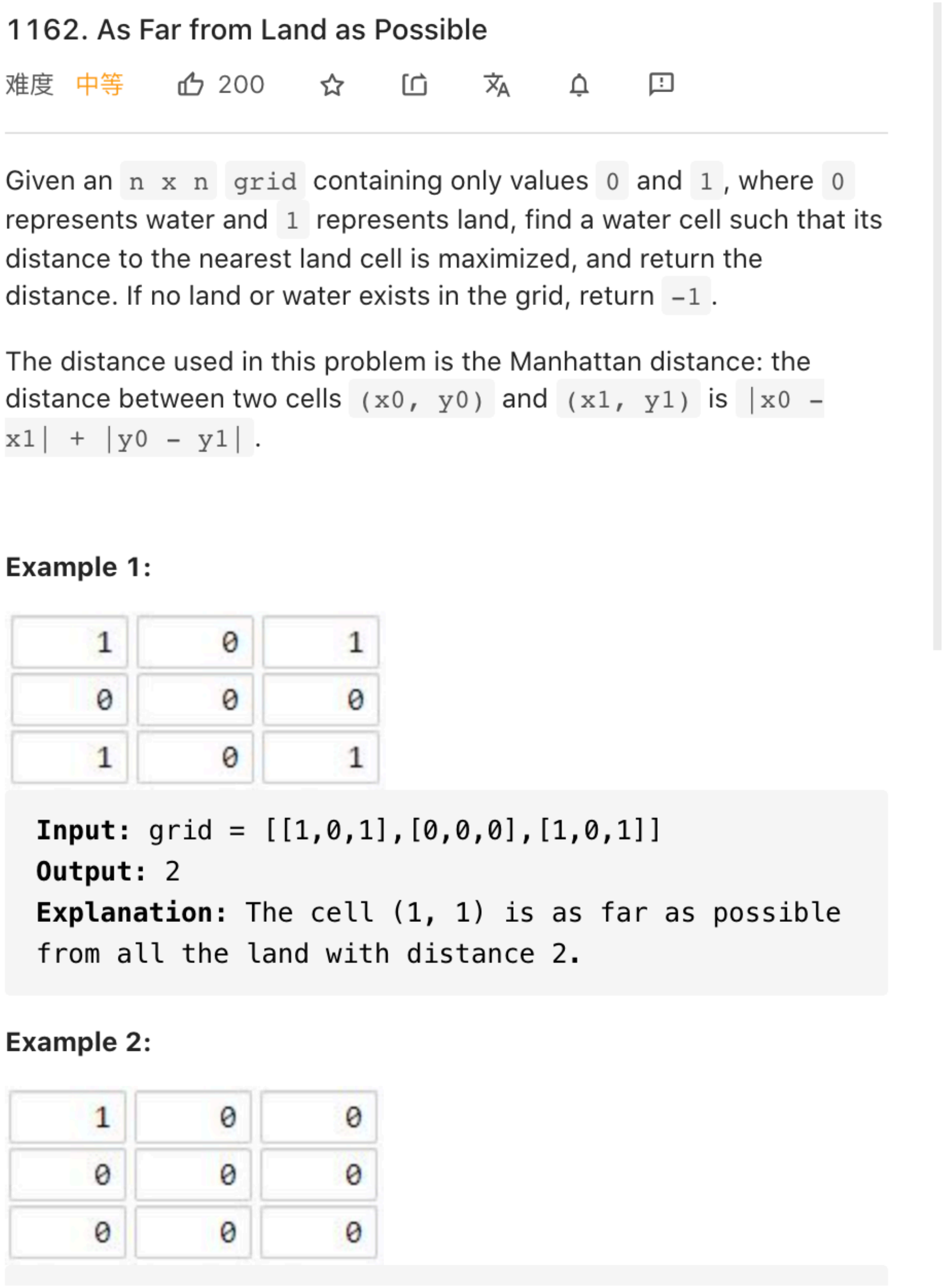
炫耀一下：

```cpp
class Solution {
public:
    int longestCommonSubsequence(string text1, string text2) {
        int len1 = text1.size();
        int len2 = text2.size();

        vector<vector<int> > dp(len1 + 1, vector<int>(len2 + 1, 0));
        for(int i = 1; i <= len1; i++){
            for(int j = 1; j <= len2 ; j++){
                if(text1[i - 1] == text2[j - 1]){
                    dp[i][j] = 1 + dp[i - 1][j - 1];
                }else{
                    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }

        return dp[len1][len2];
    }

    int max(int i ,int j){
        return i >= j ? i : j;
    }
};
```

1153 Strubg Transforms Into Another String

```
/*
    本题目的思想就是，  类似常规映射的思想
    但是 如果 string 2 中有 26个字母，那么一定无法映射
    因为如下原因

    abcdefghigklmnopqrstuvwxyz
    bcdefghigklmnopqrstuvwxyza

    对于上面的变换，我们一定要找到一个 中间 的temp 字母，否则就会一起变换
    比如 abc
        bca

  如果说 a-> b
  bbc
  bca

  那么就screw 的


  注意一定得是 ch2 的 set
  因为比如这个例子
  "abcdefghijklmnopqrstuvwxyz"
  "bcadefghijklmnopqrstuvwxzz"

  实际上是可以被转换的，比如
  先让 y 变成 z 然后 a b c 通过 y 的过渡，  变成 对应的字母
  具体变换过程是
  abc......xyz
  ->
  abc......xzz
  ->
  ybc......xzz
  yac......xzz
  bac......xzz
  byc......xzz
  bya......xzz
  bca......xzz -> 最终的结果

*/
class Solution {
public:
    bool canConvert(string str1, string str2) {
        int size1 = str1.size();
        int size2 = str2.size();

        if(str1 == str2)
            return true;
        if(size1 != size2)
            return false;

        unordered_map<char, char> map;
        unordered_set<char> set;
        for(int i = 0; i < size1; i++){
            char ch1 = str1[i];
            char ch2 = str2[i];

            if(map.count(ch1) != 0){
                if(map[ch1] != ch2)
                    return false;
            }else{
                map.insert({ch1, ch2});
            }

            set.insert(ch2);
        }

        return set.size() < 26;
    }
};
```

1162 As far From Land As Possible



1162. As Far from Land as Possible

Given an `n x n` `grid` containing only values `0` and `1`, where `0` represents water and `1` represents land, find a water cell such that its distance to the nearest land cell is maximized, and return the distance. If no land or water exists in the grid, return `-1`.

The distance used in this problem is the Manhattan distance: the distance between two cells `(x0, y0)` and `(x1, y1)` is `|x0 - x1| + |y0 - y1|`.

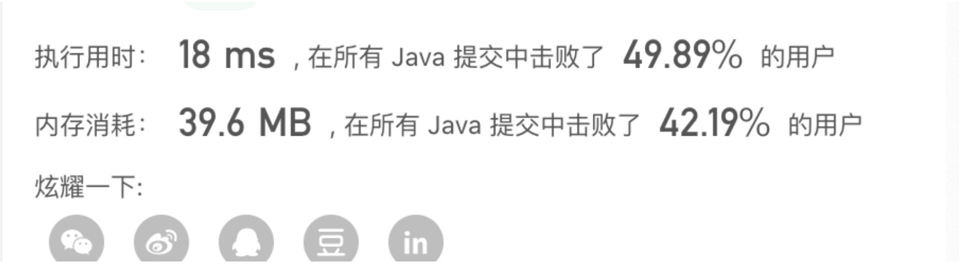
Example 1:

1	0	1
0	0	0
1	0	1

```
Input: grid = [[1,0,1],[0,0,0],[1,0,1]]
Output: 2
Explanation: The cell (1, 1) is as far as possible
from all the land with distance 2.
```

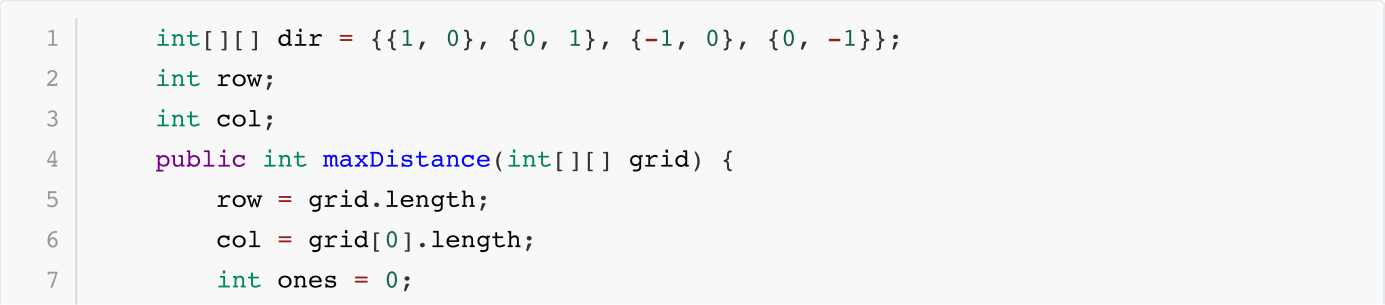
Example 2:

1	0	0
0	0	0
0	0	0



执行用时： 18 ms ，在所有 Java 提交中击败了 49.89% 的用户

内存消耗： 39.6 MB ，在所有 Java 提交中击败了 42.19% 的用户



```
    int[][] dir = {{1, 0}, {0, 1}, {-1, 0}, {0, -1}};
    int row;
    int col;
    public int maxDistance(int[][] grid) {
        row = grid.length;
        col = grid[0].length;
        int ones = 0;
```

```java
        int zeros = 0;

        Deque<int[]> queue = new ArrayDeque<>();
        for(int i = 0; i < row; i++){
            for(int j = 0; j < col; j++){
                if(grid[i][j] == 1) {
                    queue.add(new int[]{i, j});
                    ones++;
                }
            }
        }

        zeros = row * col - ones;
        int maxDistance = 0;

        while(!queue.isEmpty()){
            int size= queue.size();
            for(int i = 0; i < size; i++){
                int[] cur = queue.removeFirst();

                for(int k = 0; k < 4; k++){
                    int newX = cur[0] + dir[k][0];
                    int newY = cur[1] + dir[k][1];

                    if(isInRange(newX, newY)  && grid[newX][newY] == 0){
                        grid[newX][newY] = grid[cur[0]][cur[1]] + 10;
                        zeros--;

                        if(zeros == 0)
                            break ;

                        queue.addLast(new int[]{newX, newY});
                    }
                }

                if(zeros == 0)
                    break;
            }

            if(zeros == 0)
                break;
        }

        for(int i = 0; i < row; i++){
            for(int j =0; j < col; j++){
                if(grid[i][j] > 1)
                    maxDistance = Math.max(maxDistance, grid[i][j] / 10);
            }
        }

        return maxDistance == 0 ? -1 : maxDistance;
    }

    private boolean isInRange(int i, int j){
        return i >= 0 && j >= 0 && i < row && j < col;
    }
```

```java
    public int maxDistance(int[][] grid) {
        int row = grid.length;
        int col = grid[0].length;

        List<int[]> list = new ArrayList<>();

        for(int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                if(grid[i][j] == 1)
                    list.add(new int[]{i, j});
            }
        }


        int maxDistance = 0;
        for(int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                if (grid[i][j] == 0) {
                    int distance = Integer.MAX_VALUE;
                    for (int[] arr : list) {
                        distance = Math.min(distance, Math.abs(arr[0] - i) + Math.abs(arr[1] - j));
                    }

                    if(distance != Integer.MAX_VALUE)
                        maxDistance = Math.max(maxDistance, distance);
                }
            }
        }

        return maxDistance == 0 ? -1 : maxDistance;
    }

```