岛屿系列问题主要是用DFS/BFS来求解。

```
// 二维矩阵遍历框架
void dfs(std::vector<std::vector<int>> &grid, int i, int j,
std::vector<std::vector<int>> &visited) {
    int row = grid.size();
    int col = grid[0].size();
    // 超出边界
    if(i < 0 || i >= row || j < 0 || j >= col) {
        return ;
    }

    if(visited[i][j]) {
        return ; // 已访问过
    }

    // 前序
    visited[i][j] = true;
    dfs(nums, i-1,j,visited); // 上
    dfs(nums, i+1, j , visited); // 下
    dfs(nums, i, j-1, visited); // 左
    dfs(nums, i, j+1, visited); // 右
    // 后序
    // visited[i][j] = true;
}
```

岛屿个数问题

```
class Solution {
public:
  int numIslands(std::vector<std::vector<char>>& grid) {
    int row = grid.size(), col = 0;
    if (row != 0) {
      col = grid[0].size();
    } else {
      return 0;
    }

    int res = 0;
    for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
        if (grid[i][j] == '1') {
          res++;
          dfs(grid, row, col, i, j);
        }
      }
    }
    return res;
  }
```

```
private:
  bool isInArea(int row, int col, int i, int j) {
    return i < 0 || i >= row || j < 0 || j >= col;
  }

  void dfs(std::vector<std::vector<char>>& grid,
           int                             row,
           int                             col,
           int                             i,
           int                             j) {
    grid[i][j] = '0';
    for (auto item : dir) {
      int dx = i + item[0];
      int dy = j + item[1];
      if (!isInArea(row, col, dx, dy) && grid[dx][dy] == '1') {
        dfs(grid, row, col, dx, dy);
      }
    }
  }

  std::vector<std::vector<int>> dir = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
};
```

求封闭岛屿数

分析： 首先用dfs将边界处的岛屿全部换成海水，计算余下岛屿数即可。

```
class Solution {
public:
  int closedIsland(std::vector<std::vector<int>>& grid) {
    int row = grid.size(), col = 0;
    if (0 == row) {
      return 0;
    } else {
      col = grid[0].size();
    }

    // 首先将边界处的岛屿全部替换为海水
    for (int i = 0; i < row; i++) {
      if (0 == grid[i][0]) {
        dfs(grid, row, col, i, 0);
      }

      if (0 == grid[i][col - 1]) {
        dfs(grid, row, col, i, col - 1);
      }
    }

    for (int i = 0; i < col; i++) {
      if (0 == grid[0][i]) {
```

```
        dfs(grid, row, col, 0, i);
      }

      if (0 == grid[row - 1][i]) {
        dfs(grid, row, col, row - 1, i);
      }
    }

    int res = 0;  // 查找剩余的岛屿数
    for (int i = 1; i < row - 1; i++) {
      for (int j = 1; j < col - 1; j++) {
        if (0 == grid[i][j]) {
          res++;
          dfs(grid, row, col, i, j);
        }
      }
    }
    return res;
  }

private:
  std::vector<std::vector<int>> dir = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
  bool                          isInArea(int row, int col, int x, int y) {
    return x >= 0 && x < row && y >= 0 && y < col;
  }

  void dfs(std::vector<std::vector<int>>& grid,
           int                            row,
           int                            col,
           int                            x,
           int                            y) {
    grid[x][y] = 1;
    for (auto item : dir) {
      int dx = item[0] + x;
      int dy = item[1] + y;
      if (isInArea(row, col, dx, dy) && grid[dx][dy] == 0) {
        dfs(grid, row, col, dx, dy);
      }
    }
  }
};
```

封闭岛屿面积

```
class Solution {
public:
  int numEnclaves(std::vector<std::vector<int>>& grid) {
    int row = grid.size(), col = 0;
    if (row == 0) {
      return 0;
    } else {
```

```
        col = grid[0].size();
      }

      // 处理边界
      for (int i = 0; i < row; i++) {
        if (1 == grid[i][0]) {
          dfs(grid, row, col, i, 0);
        }

        if (1 == grid[i][col - 1]) {
          dfs(grid, row, col, i, col - 1);
        }
      }

      for (int i = 0; i < col; i++) {
        if (1 == grid[0][i]) {
          dfs(grid, row, col, 0, i);
        }

        if (1 == grid[row - 1][i]) {
          dfs(grid, row, col, row - 1, i);
        }
      }

      int res = 0;
      for (int i = 1; i < row - 1; i++) {
        for (int j = 1; j < col - 1; j++) {
          if (1 == grid[i][j]) {
            res++;
          }
        }
      }

      return res;
    }

  private:
    std::vector<std::vector<int>> dir = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

    bool isInArea(int row, int col, int x, int y) {
      return x >= 0 && x < row && y >= 0 && y < col;
    }

    void dfs(std::vector<std::vector<int>>& grid,
             int                            row,
             int                            col,
             int                            x,
             int                            y) {
      grid[x][y] = 0;
      for (auto item : dir) {
        int dx = x + item[0];
        int dy = y + item[1];
        if (isInArea(row, col, dx, dy) && 1 == grid[dx][dy]) {
          dfs(grid, row, col, dx, dy);
```

```
      }
    }
  }
};
```

岛屿最大面积

```
class Solution {
public:
  int maxAreaOfIsland(std::vector<std::vector<int>>& grid) {
    int row = grid.size(), col = 0;
    if (row == 0) {
      return 0;
    } else {
      col = grid[0].size();
    }

    // 遍历
    int res = 0;
    for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
        if (1 == grid[i][j]) {
          res = std::max(res, dfs(grid, row, col, i, j));
        }
      }
    }

    return res;
  }

private:
  std::vector<std::vector<int>> dir = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

  bool isInArea(int row, int col, int x, int y) {
    return x >= 0 && x < row && y >= 0 && y < col;
  }

  int dfs(std::vector<std::vector<int>>& grid, int row, int col, int x,
int y) {
    grid[x][y] = 0;
    int res    = 1;
    for (auto item : dir) {
      int dx = x + item[0];
      int dy = y + item[1];
      if (isInArea(row, col, dx, dy) && 1 == grid[dx][dy]) {
        res += dfs(grid, row, col, dx, dy);
      }
    }

    return res;
```

```
    }
};
```

子岛屿问题

分析： 将grid2中为岛屿，grid1中非岛屿的部分全部淹掉，剩余的即为岛屿数。

```
class Solution {
public:
  int countSubIslands(vector<vector<int>>& grid1, vector<vector<int>>&
grid2) {
    int row = grid1.size(), col = 0;
    if (0 == row) {
      return 0;
    } else {
      col = grid1[0].size();
    }

    // 将在grid1中非岛屿，grid2中岛屿部分淹掉
    for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
        if (0 == grid1[i][j] && 1 == grid2[i][j]) {
          dfs(grid2, row, col, i, j);
        }
      }
    }

    // 统计grid2中剩余岛屿
    int res = 0;
    for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
        if (1 == grid2[i][j]) {
          res++;
          dfs(grid2, row, col, i, j);
        }
      }
    }

    return res;
  }

private:
  vector<vector<int>> dir = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

  bool isInArea(int row, int col, int i, int j) {
    return i >= 0 && i < row && j >= 0 && j < col;
  }

  void dfs(vector<vector<int>>& grid, int row, int col, int x, int y) {
    grid[x][y] = 0;
    for (auto item : dir) {
```

```
      int dx = x + item[0];
      int dy = y + item[1];
      if (isInArea(row, col, dx, dy) && 1 == grid[dx][dy]) {
        dfs(grid, row, col, dx, dy);
      }
    }
  }
};
```

不同的岛屿数量

记录岛屿遍历顺序，如果岛屿的遍历顺序相同，那么其结果形状一定相同。 记录遍历每个岛屿的遍历次序，即可得到形状相同的岛屿数。

```
class Solution {
public:
  int numDistinctIslands(std::vector<std::vector<char>>& grid) {
    int row = grid.size(), col = 0;
    if (0 == row) {
      return 0;
    } else {
      col = grid[0].size();
    }

    std::set<std::string> set;

    for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
        if ('1' == grid[i][j]) {
          std::string res;
          dfs(grid, row, col, i, j, res);
          set.insert(res);
        }
      }
    }

    return set.size();
  }

private:
  std::vector<std::vector<int>> dir = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

  bool isInArea(int row, int col, int x, int y) {
    return x >= 0 && x < row && y >= 0 && y < col;
  }

  void dfs(std::vector<std::vector<char>>& grid,
           int                             row,
           int                             col,
           int                             x,
           int                             y,
```

```
            std::string&                    track) {
    grid[x][y] = '0';
    track.push_back(0 + '0');
    for (int i = 0; i < 4; i++) {
      int dx = x + dir[i][0];
      int dy = y + dir[i][1];
      if (isInArea(row, col, dx, dy) && 1 == grid[dx][dy]) {
        track.push_back(i + 1 + '0');
        dfs(grid, row, col, dx, dy, track);
        track.append(std::to_string(-(i + 1)));
      }
    }
  }
};
```