# 一：基本思想

基本套路和回溯是一样的，模板基本一致。其实都属于穷举类方法。

但又不是回溯，明显的特征是一般没有取消标记，也没有通过参数+1的操作。

注意也才采用step和option的思想。

# 二：实例

## 1. [695. 岛屿的最大面积](https://leetcode-cn.com/problems/max-area-of-island/)【深搜或者并查集】



思路：

同样step是踩一个点，option是有4中选择。

求最大的岛屿面积，考虑从非1的点作为起点进行dfs，寻找可以猜过的最多点数，

这里dfs功能是踩一点。这个地方其实不同于回溯的，这里求最大的必须使用全局变量，不能使用参数+1的形式。

并且每次进行一个点的深度搜索之后，比较本次的最大g\_curMax和最终的结果比较。

int dir[][2] = {

    {0,-1},

    {0,1},

    {-1,0},

    {1,0},

};

int g\_curMax = 0;

void LongestDepthDfs(int\*\* grid, int rowSize, int colSize, int curX, int curY)

{

    int k;

    int nextX, nextY;

    for (k = 0; k < 4; k++) {

        nextX = curX + dir[k][0];

        nextY = curY + dir[k][1];

        if (nextX < 0 || nextY < 0 || nextX >= rowSize || nextY >= colSize) {

            continue;

        }

        if (grid[nextX][nextY] == 0) {

            continue;

        }

        /\*\*标记已经被访问\*/

        grid[nextX][nextY] = 0;

        /\*\*踩过的脚印增加\*/

        g\_curMax++;

        LongestDepthDfs(grid, rowSize, colSize, nextX, nextY);

    }

}

int maxAreaOfIsland(int\*\* grid, int gridSize, int\* gridColSize)

{

    int i, j;

    int maxArea = 0;

    for (i = 0; i < gridSize; i++) {

        for (j = 0; j < gridColSize[0]; j++) {

            if (grid[i][j] == 0) {

                continue;

            }

            /\*\*标记起点已经被踩,这里可以直接采用grid当vis使用\*/

            grid[i][j] = 0;

            g\_curMax = 1;

            LongestDepthDfs(grid, gridSize, gridColSize[0], i ,j);

            // g\_curMax表示本地深搜的最大点数

            maxArea = maxArea > g\_curMax ? maxArea : g\_curMax;

        }

    }

    return maxArea;

}

并查集思想：

1.遍历grid并初始化并查集

2.遍历grid,对非0的进行四周合并

3.遍历grid,对非0的节点寻根，用map统计相同根节点的最大值

int dir[][2] = {

    {0,-1},

    {0,1},

    {-1,0},

    {1,0},

};

int g\_curMax = 0;

void LongestDepthDfs(int\*\* grid, int rowSize, int colSize, int curX, int curY)

{

    int k;

    int nextX, nextY;

    for (k = 0; k < 4; k++) {

        nextX = curX + dir[k][0];

        nextY = curY + dir[k][1];

        if (nextX < 0 || nextY < 0 || nextX >= rowSize || nextY >= colSize) {

            continue;

        }

        if (grid[nextX][nextY] == 0) {

            continue;

        }

        /\*\*标记已经被访问\*/

        grid[nextX][nextY] = 0;

        /\*\*踩过的脚印增加\*/

        g\_curMax++;

        LongestDepthDfs(grid, rowSize, colSize, nextX, nextY);

    }

}

int maxAreaOfIsland(int\*\* grid, int gridSize, int\* gridColSize)

{

    int i, j;

    int maxArea = 0;

    for (i = 0; i < gridSize; i++) {

        for (j = 0; j < gridColSize[0]; j++) {

            if (grid[i][j] == 0) {

                continue;

            }

            /\*\*标记起点已经被踩,这里可以直接采用grid当vis使用\*/

            grid[i][j] = 0;

            g\_curMax = 1;

            LongestDepthDfs(grid, gridSize, gridColSize[0], i ,j);

            // g\_curMax表示本地深搜的最大点数

            maxArea = maxArea > g\_curMax ? maxArea : g\_curMax;

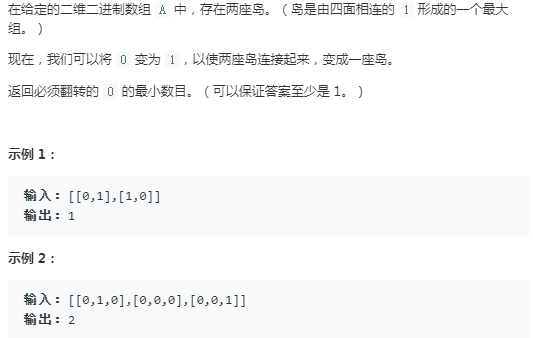
        }

    }

    return maxArea;

}

## 2. [934. 最短的桥](https://leetcode-cn.com/problems/shortest-bridge/)【DFS染色+BFS扩散搜索最短路径】



思路：

任意找一个岛屿DFS进行染色，由于是不用的回溯找更多路径的，因此可以直接利用染色的标记2 替代vis来避免对同一节点重复访问。

之后对染色岛屿的所有坐标为起点进行BFS，寻找当前的最小步数，然后记录整体的最小步数。因为要记录层数，因此采用了 将同一层的所有队列节点都pop之后对层数+1。

int dir[][2] = {

    {0,-1},

    {0,1},

    {-1,0},

    {1,0},

};

int g\_find = 0;

#define MAX 200

typedef struct Queue {

    int head;

    int tail;

    int size;

    int A[MAX][2];

}QUEUE\_T;

void QueueReInit(QUEUE\_T\* queue)

{

    queue->head = 0;

    queue->tail = 0;

    queue->size = 0;

    memset(queue->A, 0 , MAX \* 2\* sizeof(int));

}

int QueueIsFull(QUEUE\_T\* queue)

{

    return queue->size == MAX ? 1 : 0;

}

int QueueIsEmpty(QUEUE\_T\* queue)

{

    return queue->size == 0 ? 1 : 0;

}

void QueuePush(QUEUE\_T\* queue, int x, int y)

{

    if (QueueIsFull(queue) == 1) {

        return;

    }

    queue->A[queue->tail][0] = x;

    queue->A[queue->tail][1] = y;

    queue->size++;

    queue->tail = (++queue->tail) % MAX;

}

void QueuePop(QUEUE\_T\* queue, int\* x, int\* y)

{

    if (QueueIsEmpty(queue) == 1) {

        return;

    }

    \*x = queue->A[queue->head][0];

    \*y = queue->A[queue->head][1];

    queue->size--;

    queue->head = (++queue->head) % MAX;

}

void ColorDfs(int\*\* A, int rowSize, int colSize, int curX, int curY)

{

    int i, j;

    int nextX, nextY;

    for (i = 0; i < 4; i++) {

        nextX = curX + dir[i][0];

        nextY = curY + dir[i][1];

        if (nextX < 0 || nextY < 0 || nextX >= rowSize || nextY >= colSize) {

            continue;

        }

        if (A[nextX][nextY] != 1) {

            continue;

        }

        /\*\*染色2\*/

        A[nextX][nextY] = 2;

        ColorDfs(A, rowSize, colSize, nextX, nextY);

    }

}

int g\_BfsLevel = 0;

void ExpandBfs(int\*\* A, int rowSize, int colSize, int startX, int startY)

{

    QUEUE\_T queue = {0};

    int vis[MAX][MAX] = {0};

    int curX = 0, curY = 0;

    int nextX = 0, nextY = 0;

    int levelSize;

    int i,j;

    QueueReInit(&queue);

    vis[startX][startY] = 1;

    QueuePush(&queue, startX, startY);

    while (QueueIsEmpty(&queue) != 1) {

        levelSize = queue.size;

        for (i = 0; i < levelSize; i++) {

            QueuePop(&queue, &curX, &curY);

            for (j = 0; j < 4; j++) {

                nextX = curX + dir[j][0];

                nextY = curY + dir[j][1];

                if (nextX < 0 || nextX >= rowSize || nextY < 0 || nextY >= colSize) {

                    continue;

                }

                if (A[nextX][nextY] == 2 || vis[nextX][nextY] == 1) {

                    continue;

                }

                // 找到

                if (A[nextX][nextY] == 1) {

                    g\_find = 1;

                    return;

                }

                // 标记

                vis[nextX][nextY] = 1;

                QueuePush(&queue, nextX, nextY);

            }

        }

        g\_BfsLevel++;

    }

}

int shortestBridge(int\*\* A, int ASize, int\* AColSize)

{

    int i, j;

    int min = INT\_MAX;

    int color = 0;

    /\*\*随便染色一个岛屿\*/

    for (i = 0; i < ASize; i++) {

        for (j = 0; j < AColSize[0]; j++) {

            if (A[i][j] == 1) {

                A[i][j] = 2;

                ColorDfs(A, ASize, AColSize[0], i ,j);

                break;

            }

        }

        if (j < AColSize[0]) {

            break;

        }

    }

    for (i = 0; i < ASize; i++) {

        for (j = 0; j < AColSize[0]; j++) {

            printf("%d ", A[i][j]);

        }

        printf("\n");

    }

    /\*\*染色之后，选取一个岛屿的所有点进行都进行一遍扩散，寻找最小的\*/

    for (i = 0; i < ASize; i++) {

        for (j = 0; j < AColSize[0]; j++) {

            if (A[i][j] == 2) {

                g\_BfsLevel = 0;

                g\_find = 0;

                ExpandBfs(A, ASize, AColSize[0], i ,j);

                if (g\_find == 1) {

                    min = min > g\_BfsLevel ? g\_BfsLevel : min;

                }

            }

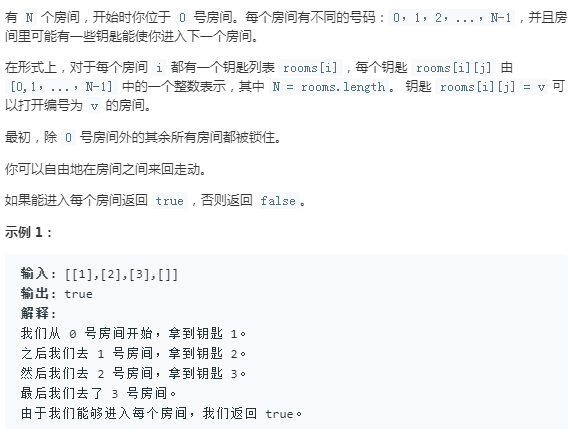
        }

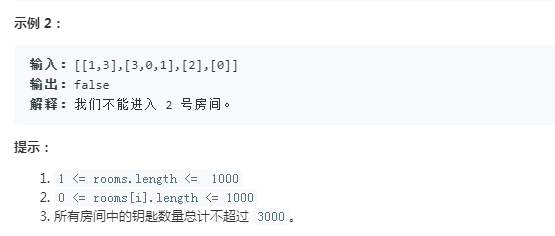
    }

    return min;

}

## 3.[841. 钥匙和房间](https://leetcode-cn.com/problems/keys-and-rooms/)【深搜】





思路：step是每次访问一个房间，option是从n把钥匙中选着择，对于访问过的房间用vis标记。当深度搜索之后，如果vis中有未标记的则说明false.

或者对访问的房间进行计数，就是做标记同时计数。

void VisOneRoomDfs(int\*\* rooms, int roomsSize, int\* roomsColSize, int curRoom, int\* vis)

{

    int i;

    for (i = 0; i < roomsColSize[curRoom]; i++) {

        if (vis[rooms[curRoom][i]] == 1) {

            continue;

        }

// 对访问的房间进行标记，可以引入全局变量从而避免最后去遍历vis

        vis[rooms[curRoom][i]] = 1;

        VisOneRoomDfs(rooms, roomsSize, roomsColSize, rooms[curRoom][i], vis);

    }

}

bool canVisitAllRooms(int\*\* rooms, int roomsSize, int\* roomsColSize)

{

    int\* vis = (int\*)calloc(roomsSize, sizeof(int));

    int i, j;

    vis[0] = 1;

    VisOneRoomDfs(rooms, roomsSize, roomsColSize, 0 ,vis);

    for (i = 0; i < roomsSize; i++) {

        if (vis[i] != 1) {

            return false;

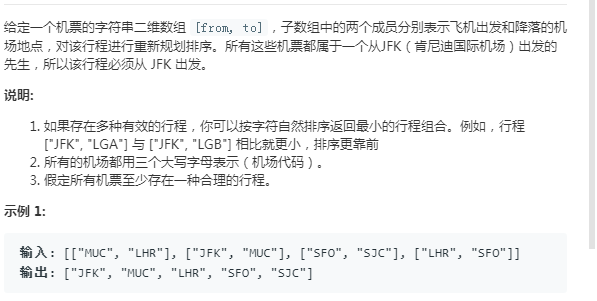
        }

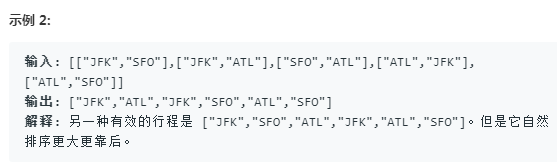
    }

    return true;

}

## 4.[332. 重新安排行程](https://leetcode-cn.com/problems/reconstruct-itinerary/)【深搜&回溯】





思路：

首先，路径排序最后考虑，那么可以dfs将所有的符合题意的结果保存，最后选取排序最小的路径其实就可以。

那既然也需要用到回溯，因为加入当前城市加入之后后面又不符合，需要重新考虑，因此可以直接先把tickets按照最后一个城市升序排序，那么在做选择时，只要找到出发城市的并且没有访问过的票就进一步递归，如果发现票用完了即符合题意的路径找到了就立即停止搜索的方法显然会节省时间。

int MyCmp(const void\* a, const void\* b)

{

    char\*\*\* pa = (char\*\*\*)a;

    char\*\*\* pb = (char\*\*\*)b;

    int ret = strcmp((\*pa)[1], (\*pb)[1]);

    return ret;

}

int g\_curIndex = 0;

bool OneTicketDfs(char \*\*\* tickets, int ticketsSize, char\*\* res, int box, int\* vis, char\* start)

{

    int i;

int ret;

// 截止条件是票用完了，城市个数为票数+1

    if (box == ticketsSize + 1) {

        return true;

    }

    for (i = 0; i < ticketsSize; i++) {

        if (vis[i] == 1) {

            continue;

        }

        if (strcmp(tickets[i][0], start) == 0) {

            vis[i] = 1;

            strcpy(res[box], tickets[i][1]);

// 一旦找到立即停止搜索

            if (OneTicketDfs(tickets, ticketsSize, res, box + 1, vis, tickets[i][1]) == true) {

                return true;

            }

            vis[i] = 0;

        }

    }

    return false;

}

char \*\* findItinerary(char \*\*\* tickets, int ticketsSize, int\* ticketsColSize, int\* returnSize)

{

    int i;

    char\*\* res = NULL;

    res = (char\*\*)calloc(500, sizeof(char\*));

    int\* vis = (int\*)calloc(ticketsSize, sizeof(int));

    for (i = 0; i < 500; i++) {

        res[i] = (char\*)calloc(10, sizeof(char));

    }

    qsort(tickets, ticketsSize, sizeof(char\*\*), MyCmp);

    for (i = 0; i < ticketsSize; i++) {

        //printf("%s %s\n", tickets[i][0], tickets[i][1]);

    }

    strcpy(res[0], "JFK");

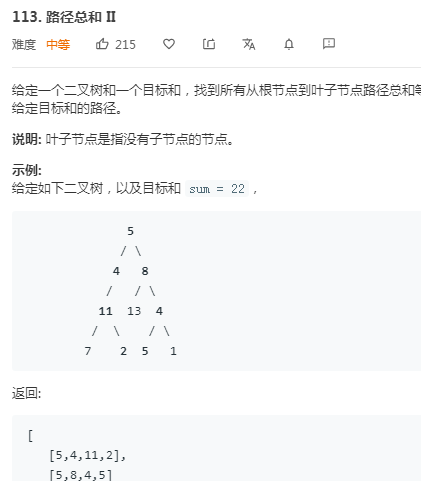
    OneTicketDfs(tickets, ticketsSize, res, 1, vis, "JFK");

    \*returnSize = ticketsSize + 1;

    return res;

}

## 5. [113. 路径总和 II](https://leetcode-cn.com/problems/path-sum-ii/)【树的先序遍历】



#define MAX 1000

int g\_count = 0;

void PreOrderDfs(struct TreeNode\* root, int sum, int curSum, int\*\* res, int\* resColSize, int\* ans, int level)

{

    int i;

    if (root == NULL) {

        return;

    }

    ans[level] = root->val;

    if (root->left == NULL && root->right == NULL) {

        if (curSum + root->val == sum) {

            memcpy(res[g\_count], ans, (level + 1) \* sizeof(int));

            resColSize[g\_count++] = level + 1;

            return;

        }

    }

    PreOrderDfs(root->left, sum, curSum + root->val, res, resColSize, ans, level + 1);

    PreOrderDfs(root->right, sum, curSum + root->val, res, resColSize, ans, level + 1);

}

int\*\* pathSum(struct TreeNode\* root, int sum, int\* returnSize, int\*\* returnColumnSizes){

    int \*\*res = NULL;

    int\* ans = NULL;

    int\* resColSize = NULL;

    int i;

    g\_count = 0;

    res = (int\*\*)calloc(MAX, sizeof(int\*));

    ans = (int\*)calloc(MAX, sizeof(int));

    resColSize = (int\*)calloc(MAX, sizeof(int));

    for (i = 0 ; i < MAX; i++) {

        res[i] = (int\*)calloc(MAX, sizeof(int));

    }

    PreOrderDfs(root, sum, 0, res, resColSize, ans, 0);

    \*returnSize = g\_count;

    \*returnColumnSizes = resColSize;

    return res;

}