# 基本理论:

## (1)递归是一种编程技巧，方法是分治，推荐使用递归法好理解，推荐在几十到百层的数量级推荐使用递归。

## (2)使用cache（缓存）进行减少重复计算,很多题都可以用递归+cache的方法。

## (3)广度(BFS)一般用的少，深度(DFS)可以解决很多问题。与深度强相关的是回溯，属于一种编程技巧或者思路。

A:回溯有2种写法，循环控制回溯路径，但是回溯推荐用递归+cache的方法，因为特别容易看懂。visit记得要抹除

B:套路话但很容易的写法，深度搜索上其实也有2个概念。深度对于每个题不一样，广度也不同。

如在井字格上深度（step）定义是往前走一步，广度(option)是怎么走，广度是必须在一个step中需要全部完成的。

### 1.基本井字格

4行3列的格子，从（0，0）到端点的（4，3）不走远路的话可以有多少种走法？



（1）分治法，使用的是递归编程技巧

设f(m,n)为到达（m,n）的走法种树则：f(m,n) = f(m-1,n) + f(m,n-1)

int f(m,n)

{

if (m < 0 || n < 0) {

return 0;

}

if (m == 0 && n == 0) {

return 1;

}

return f(m-1, n) + f(m, n-1);

}

缓存减少重复计算，解决性能问题

int cache[100][100] = {0};

int f(m,n)

{

if (m < 0 || n < 0) {

return 0;

}

if (m == 0 && n == 0) {

return 1;

}

if (cache[m][n] != 0 ) {

return cache[m][n];

}

cache[m][n] = f(m-1, n) + f(m, n-1);

return cache[m][n];

}

（2）动态规划法

A:把显而易见的东西写上去

B:按照路径去计算

int dp[100][100] = {0};

int f\_dp(int m,int n)

{

int i = 0, j = 0;

dp[0][0] = 1;

for (i = 1; i <= n; i++) {

dp[0][i] = 1;

}

for (i = 1; i <= m; i++) {

dp[i][0] = 1;

}

for (i = 1; i <= m; i++) {

for (j = 1; j <= n; j++) {

dp[i][j] = dp[i][j - 1] + dp[i - 1][j];

}

}

return dp[m][n];

}

4行3列的格子，从（0，0）到端点的（4，3）可以走远路的话可以有多少种走法？

（3）深度搜索（DFS）

广度(BFS)一般用的少，深度(DFS)可以解决很多问题。与深度强相关的事回溯，属于一种编程技巧或者思路。

A:回溯有2种写法，循环控制回溯路径，但是推荐用递归+cache的方法，因为特别容易看懂。

B:套路话但很容易的写法，深度搜索上其实也有2格概念。深度对于每个题不一样，广度也不同。

如在井字格上深度（step）是往前走一步，广度(option)是怎么走，广度是必须在一个step中需要全部完成的。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*也是DFS模板,可用与其他DFS的问题\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int step(int x, int y, int m, int n)

{

int sum = 0;

if (x == m && y == n) {

return 1; // 走到目的地

}

if (x < 0 || y <0 || x > COL || y > ROW) {

return 0; // 判断边界

}

if (visited[x][y] == 1) {

return 0;

}

if (visited[x][y] == 1) {

return 0;

}

visited[x][y] = 1; //踩脚印

sum += step(x - 1,y,m,n); //向左

sum += step(x + 1,y,m,n); //向右

sum += step(x,y + 1,m,n); //向上

sum += step(x,y - 1,m,n); //向下

visited[x][y] = 0; //抹脚印

return sum;

}

### 2.八/N皇后问题

描述：4\*4的格子，皇后要在格子上，不能互相攻击。可以攻击的规则是左右上下，及45度角。求有多少中状态？



其实不用二维数组，用一个queue\_pos的一维数组，表示每一列皇后所在的行号。那么用深度搜索，step在这里定义就是部署一个皇后；广度是必需在每个step必须完成的，在这里就是部署在index列的某一行。

#define N 8

int queue\_pos[N] = {0};

static int g\_kind = 0;

void step(int index)

{

int i = 0;

int j = 0;

//printf("?\n");

// 部署到第index个皇后，就认为是是一种OK的状态

if (index == N + 1) {

g\_kind++;

}

// option广度上需要判断是否合理,从index列的0行到N行都试一下

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

//尝试部署一个

queue\_pos[index] = i;

// 判断是否符合不攻击的状态，第一个判断是这个位置不能与前面的位置出现在同一行

for (j = 0; j < index; j++) {

// 不能在同一行

if (queue\_pos[j] == queue\_pos[index]) {

break;

}

// 不能在45度

if (abs(queue\_pos[index] - queue\_pos[j]) == abs(index - j)) {

break;

}

}

if (j >= index) {

step(index + 1);

}

}

}

### 3.数独问题

9\*9的格子，即9个9宫格。

这里step的定义其实是填格子，options是填1-9的9格数字。探索只管前面。

### 4.括号匹配问题

给定一个数字N，表示几对括号“（”“）”，把所有的可能打印出来，非法的除外。

在这里深度上step的定义是放一个符号，option是放左括号或者右括号。

#define N 2

void step(char\* buff, int index, int left, int right)

{

if (index == 2 \* N) {

buff[index + 1] = '\0';

printf("%s\n",buff);

}

if (left < N)

{

buff[index] = '(';

step(buff,index + 1, left + 1,right);

}

if (right < left) {

buff[index] = ')';

step(buff,index + 1, left, right + 1);

}

}

### 5.通配符号匹配问题

text = "ab.....def" pattern = "a?\*bf",写一个程序判断text是否与patter你是否匹配。？可以匹配任意单个字符，\*可以匹配0个或多个任意字符。

这个问题上step定义为在patter方向，一个字符在text匹配上了，如step一步匹配了一个pattern中的a;

自己没做出来，用上面的办法。

### 6. 划分为k个相等的子集【回溯递归】

给定一个整数数组  nums 和一个正整数 k，找出是否有可能把这个数组分成 k 个非空子集，其总和都相等。

示例 1：

输入： nums = [4, 3, 2, 3, 5, 2, 1], k = 4

输出： True

说明： 有可能将其分成 4 个子集（5），（1,4），（2,3），（2,3）等于总和。

回溯问题分为“盒子”问题和“迷宫”问题。

本题即为盒子问题。

本题思路：这里的step其实是去往盒子里放可以和为target的数，盒子放完即可。Option其实是从len(nums)个数中选择。

，

#define true 1

#define false 0

#define bool int

int my\_cmp(const void\* a, const void\* b)

{

int \*pa = (int\*)a;

int \*pb = (int\*)b;

return \*pa - \*pb;

}

int dfs(int\* nums, int numsSize, int target, int\* flag, int start, int index, int k)

{

int i = 0;

if (k == 0)

return true;

for (i = start; i < numsSize && (nums[i] + index <= target); ++i) {

if (flag[i] == 0 ) {

flag[i] = 1;

if (nums[i] + index == target) {

if (dfs(nums, numsSize, target, flag, 0, 0, k - 1))

return true;【不能直接写成return dfs(nums, numsSize, target, flag, 0, 0, k - 1)否则构不成回溯，因为一旦失败后从这里返回就无法对flag清0了】

}

else if (dfs(nums, numsSize, target, flag, i + 1,nums[i] + index, k))

return true;

flag[i] = 0;

}

}

return false;

}

bool canPartitionKSubsets(int\* nums, int numsSize, int k)

{

int sum = 0;

int i, target;

int flag[40] ={0};

int max = -1;

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

sum += nums[i];

if (max < nums[i]) {

max = nums[i];

}

}

if (sum % k != 0) {

return false;

}

target = sum / k;

if ( max > target) {

return false;

}

qsort(nums, numsSize, sizeof(int), my\_cmp);

return dfs(nums, numsSize, target,flag,0, 0, k);

}