# 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

// 类似树的字底向上的写法

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

题目描述：

给出 n 代表生成括号的对数，请你写出一个函数，使其能够生成所有可能的并且有效的括号组合。

例如，给出 n = 3，生成结果为：

[

"((()))",

"(()())",

"(())()",

"()(())",

"()()()"

]

/\*这里step的定义就是放一个括号，直到放满2\*box个盒子；option是放左还是放右;

放左的条件是只要左边的没放完，就可以一直放;

\*/

void step(char\* ans, int left, int right, int box)

{

if (left + right == box) {

memcpy(res[g\_count], ans, strlen(ans) + 1);

g\_count++;

return;

}

// 只要左括号没有达到最大值就都可以放

if (left \* 2 < box) {

ans[left + right] = '(';

step(ans, left +1, right, box);

}

// 只要右括号比左括号少就都可以放

if (right < left) {

ans[left +right] = ')';

step(ans, left, right +1, box);

}

}

char \*\* generateParenthesis(int n, int\* returnSize){

char\* ans;

int i;

g\_count = 0;

if (n == 0) {

\*returnSize = 0;

return res;

}

if (n == 1) {

\*returnSize = 1;

res = (char\*\*)malloc(sizeof(char\*)\*2);

res[0] = (char\*)malloc(sizeof(char) \* 3);

memset(res[0], 0, 3);

memcpy(res[0], "()",3);

return res;

}

ans = (char\*)malloc(sizeof(char) \* (2 \* n) + 1);

memset(ans, 0, sizeof(char) \* (2 \* n) + 1);

res = malloc(sizeof(char\*) \* RES\_SIZE\_MAX);

for (i = 0; i < RES\_SIZE\_MAX; i++) {

res[i] = malloc(sizeof(char) \* (2 \* n + 1));

memset(res[i], 0 , sizeof(char) \* (2 \* n + 1));

}

step(ans, 0, 0 , 2 \* n);

\*returnSize = g\_count;

free(ans);

ans = NULL;

return res;

}

### 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);

}

### 7. [字母大小写全排列](https://leetcode-cn.com/problems/letter-case-permutation/)【回溯递归】

给定一个字符串S，通过将字符串S中的每个字母转变大小写，我们可以获得一个新的字符串。返回所有可能得到的字符串集合。

**示例:**

**输入:** S = "a1b2"

**输出:** ["a1b2", "a1B2", "A1b2", "A1B2"]

本题其实也是回溯类问题的盒子模型。

思路：将S中的字母个数num算出，就可以得到结果集的总数（2^num）。num其实就是盒子，那么在这里step的定义就是填一个盒子，而option就是2个，分别大小写字母。

1. vis[i][0] 表示字母本身，为1表示字母已经排列过，vis[i][1]表示对应的大或小写
2. res用于报存整个题目的结果，ans用于保存每次step过程中的结果，如果直接用res[g\_count][i]可能会造成越界，因此以后尽量step过程中另外使用数组保存过程的结果
3. option中用index从start开始第一是可以优化性能，第二有时会导致问题重复（比如二进制时钟问题中必须用start开始）

#define MAX 13

int vis[MAX][2] = {0};

int g\_count = 0;

void Step(char\*\* res, char\* S, char\* ans, int box, int index, int len)

{

    int i;

    if (box == 0) {

        memcpy(res[g\_count], ans, len + 1);

        g\_count++;

        return;

    }

    for (i = index; i < len; i++) {

        if (vis[i][0] == 1 && vis[i][1] == 1) {

            ans[i] = S[i];

            continue;

        }

        if (vis[i][0] != 1) {

            vis[i][0] = 1;

            ans[i] = S[i];

            Step(res, S, ans, box - 1, i + 1, len);

            vis[i][0] = 0;

        }

        if (vis[i][1] != 1) {

            vis[i][1] = 1;

            ans[i] = (S[i] >= 'a' && S[i] <= 'z') ? (S[i] - 32) : (S[i] + 32);

            Step(res, S, ans, box - 1, i + 1, len);

            vis[i][1] = 0;

        }

    }

}

char \*\* letterCasePermutation(char \* S, int\* returnSize){

    int len, i;

    int box = 0;

    int sum = 1;

    char \*\*res;

    char\* ans ;

    len = strlen(S);

    g\_count = 0;

    // 计算总共的大小，并标记非字母的已经处理

    for (i = 0; i < len; i++) {

        if ((S[i] >= 'A' && S[i] <= 'Z') || (S[i] >= 'a' && S[i] <= 'z')) {

            sum \*= 2;

            box++;

            vis[i][0] = 0;

            vis[i][1] = 0;

        }

        else {

            vis[i][0] = 1;

            vis[i][1] = 1;

        }

    }

    // 如果没有字母直接返回原串

    if(box == 0) {

        \*returnSize = 1;

        res = (char\*\*)malloc(sizeof(char\*) \* (\*returnSize));

        res[0] = (char\*)malloc(sizeof(char) \* (len + 1));

        memcpy(res[0], S, len + 1);

        return res;

    }

    // 分配空间

    \*returnSize = sum;

    res = (char\*\*)malloc(sizeof(char\*) \* (\*returnSize));

    for (i = 0; i < \*returnSize; i++) {

        res[i] = (char\*)malloc(sizeof(char) \* (len + 1));

        memset(res[i], 0 , sizeof(char) \* (len + 1));

    }

    // ans用于保存每次step的结果，如果直接用res[g\_count][i]会导致越界。这里赋值，可以规避像1ab2的串第一次只返回1ab的问题

    ans = (char\*)malloc(len + 1);

    memcpy(ans, S, len + 1);

    Step(res, S, ans, box, 0, len);

    return res;

}

### 8. [401. 二进制手表](https://leetcode-cn.com/problems/binary-watch/)【回溯递归】

题目：二进制手表顶部有 4 个 LED 代表**小时（0-11）**，底部的 6 个 LED 代表**分钟（0-59）**。每个 LED 代表一个 0 或 1，最低位在右侧。

思路：其实也是属于盒子问题，这里step的定义其实就是num个灯，option就是9个选择，当然要用index为start开始搜索。

/\*这里的step定义为放一个灯；option可以定义为从0-10个灯选择位置\*/

/\*这里的step定义为放一个灯；option可以定义为从0-10个灯选择位置\*/

1. 每个结果最多6个字符串，并且用全局的res减少step函数的参数
2. cal[]数组可以减少计算，并且可以在参数中用hour与min减少计算
3. 注意

// 用于计数

int g\_count = 0;

char \*\*res = NULL;

int cal[] = {8,4,2,1,32,16,8,4,2,1};

#define MAX 0xfffff

// 每个字符串最长为6 如12:00

#define COL\_MAX 6

void step(int\* vis, int hour, int min, int start, int num)

{

    int i;

    if (num == 0) {

        sprintf(res[g\_count], "%d:%02d", hour,min);

        g\_count++;

        return;

    }

    for (i = start; i <= 9; i++) {

        if (vis[i] != 0) {

            continue;

        }

        vis[i] = 1;

        if (i >= 6 && hour + cal[i] >= 12) {

            vis[i] = 0;

            continue;

        }

        if (i < 6 && min + cal[i] >= 60) {

            vis[i] = 0;

            continue;

        }

        if (i >= 6) {

            step(vis, hour + cal[i], min, i + 1, num - 1);

            vis[i] = 0;

        }

        else {

            step(vis, hour, min + cal[i], i + 1, num - 1);

            vis[i] = 0;

        }

    }

}

char \*\* readBinaryWatch(int num, int\* returnSize){

    int vis[10] = {0};

    int i;

    res = (char\*\*)malloc(sizeof(char\*) \* MAX);

    for (i = 0; i < MAX; i++) {

        res[i] = (char\*)malloc(sizeof(char) \* COL\_MAX);

        memset(res[i], 0, sizeof(char) \* COL\_MAX);

    }

    g\_count = 0;

    step(vis, 0 ,0, 0, num);

    \*returnSize = g\_count;

    return res;

}

### 9. [17. 电话号码的字母组合](https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number/)【深搜递归不算回溯】

题目描述：给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。



**输入：**"23"

**输出：**["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].

思路：在本题其实也是盒子模型。盒子的大小其实就是输入字符串的长度。因此step的定义就是放一个盒子（字母），option就是从3或者4个字母中选择。

可以制作一个表：

static char\* g\_table[] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

方便计算放第box个盒子时有多少种选择strlen(g\_table[digits[box] – ‘0’])。

很好理解的是本地不需要回溯，或者说标记。如果标记了，那么对于有重复数字的输入就会少了几种选择。

#define MAX 9

static char\* g\_table[] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};

char \*\*res = NULL;

char\* ans = NULL;

int g\_count = 0;

int vis[10][4] = {0};

void step(char \*digits, int box, int box\_max)

{

    int i;

    if (box == box\_max) {

        memcpy(res[g\_count], ans, strlen(ans));

        g\_count++;

        return;

    }

    for (i = 0; i < strlen(g\_table[digits[box] - '0']); i++) {

        //if (vis[digits[box] - '0'][i] == 0) {

            //vis[digits[box] - '0'][i] = 1;不能做标记

            ans[box] = g\_table[digits[box] - '0'][i];

            step(digits, box + 1, box\_max);

            //vis[digits[box] - '0'][i] = 0;

        //}

    }

}

char \*\* letterCombinations(char \* digits, int\* returnSize){

    int sum = 1;

    int i, j, len;

    g\_count = 0;

    len  = strlen(digits);

    if (len == 0) {

        res = (char\*\*)malloc(sizeof(char\*));

        res[0] = malloc(sizeof(char) \* 1);

        memset(res[0], 0, sizeof(char) \* MAX);

        \*returnSize = 1;

        return res;

    }

    // 计算总共可以返回多少个字符串

    for (i = 0; i < len; i++) {

        sum \*= strlen(g\_table[digits[i] - '0']);

    }

    \*returnSize = sum;

    // 分配空间

    res = (char\*\*)malloc(sum \* sizeof(char\*));

    for (i = 0; i < sum; i++) {

        res[i] = malloc(sizeof(char) \* MAX);

        memset(res[i], 0, sizeof(char) \* MAX);

    }

    for (i = 0; i < 10; i++) {

        for (j = 0; j < 4; j++) {

            vis[i][j] = 0;

        }

    }

    ans = (char\*)malloc(MAX);

    memset(ans, 0 , MAX);

    step(digits, 0, len);

    return res;

}

### 10. [78. 子集](https://leetcode-cn.com/problems/subsets/)【递归深搜】

题目描述：给定一组不含重复元素的整数数组 nums，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

示例:

输入: nums = [1,2,3]

输出:

[

[3],

  [1],

  [2],

  [1,2,3],

  [1,3],

  [2,3],

  [1,2],

  []

]

解题思路：本题也是属于盒子类题目。

(1)Step的定义其实是放一个盒子，option是对应的数字选还是不选。

(2)Box的数目其实就是nums的长度。选或者不选，都认为是搞定了一个盒子,即box+1,当box==numsSize时，认为得到子集中的一个解。

(3)用ans数组来保存每个step的结果，如果当前的数字当选，则递归到下一层时ans\_index+1.否则直接传递ans\_index

#define MAX 1000

int\*\* res = NULL;

int g\_count = 0;

void step(int\* nums, int numsSize, int\* ans, int box, int\* res\_col, int ans\_index, int start)

{

int i;

if (box == numsSize) {

memcpy(res[g\_count], ans, sizeof(int) \* (ans\_index));

res\_col[g\_count] = ans\_index;

g\_count++;

return;

}

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

// 先记录当前数字

ans[ans\_index] = nums[i];

// 当前的数字当选了，box+1,ans\_index+1

step(nums, numsSize, ans, box + 1, res\_col, ans\_index + 1, i + 1);

// 当前的数字不当选，box仍然+1，认为处理了一个盒子，但是ans\_index不能加

step(nums, numsSize, ans, box + 1,res\_col, ans\_index, i + 1);

}

}

int\*\* subsets(int\* nums, int numsSize, int\* returnSize, int\*\* returnColumnSizes){

int i =0;

int sum = 1;

int\* res\_col = NULL;

int\* ans = NULL;

g\_count = 0;

// 计算结果数

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

sum \*= 2;

}

\*returnSize = sum;

// 为结果分配空间

res = (int\*\*)malloc(sizeof(int\*) \* (\*returnSize));

res\_col = (int\*)malloc(sizeof(int) \* (\*returnSize));

memset(res\_col, 0, sizeof(int) \* (\*returnSize));

for (i = 0; i < \*returnSize; i++) {

res[i] = malloc(sizeof(int) \* (numsSize));

memset(res[i], 0, sizeof(int) \* numsSize);

}

// 保存单次的结果

ans = (int\*)malloc(sizeof(int) \* numsSize);

memset(ans, 0, numsSize);

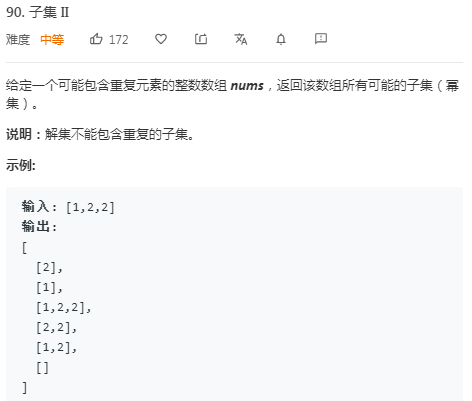
step(nums, numsSize, ans, 0, res\_col, 0 ,0);

\*returnColumnSizes = res\_col;

return res;

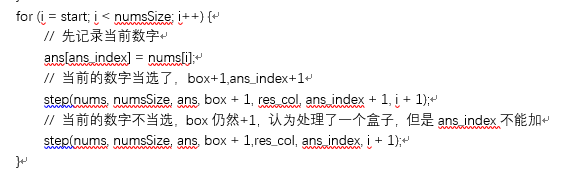
}

### 11.子集II【递归深搜】



思路：本题与子集问题类似，关键在与排序去重。

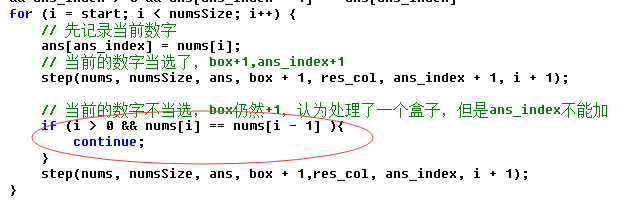
如1，2，2的输入按照选与不选的思想，重复的地方在于此：



（1）当1，2，2都被选上时，栈回到上层，即box = 1时，第二个2不选，此时[1,2]会被打印。

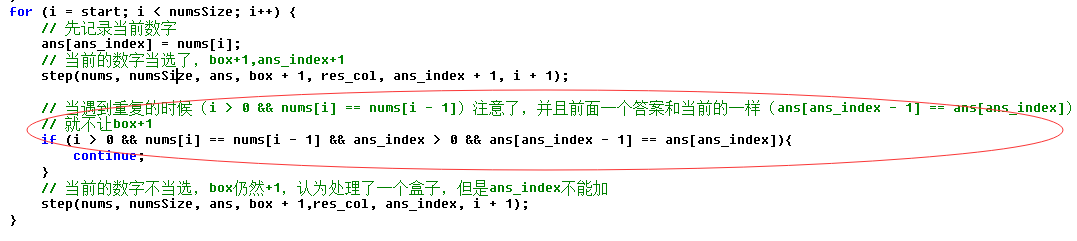
（2）当box = 0时，1当选，第一个2不当选，第二个当选，此时[1,2又被打印，因此关键是如何避免这2种情况。

很容易想到是判断（1）的时候当发现nums[i]与nums[i-1]相等时不去step。即如下：



但是这样的话，还有一种情况是，当1当选，第二、三个2都不当选的情况，1无法打印了，因为此时box没办法为3。

因此，既要让第一个2当选，第二个2不当选的时候不把[1,2]给打出来，又要在2个2不当选的时候把1给打出来，只能用如下方式。



实现：

#define MAX 1000

int\*\* res = NULL;

int g\_count = 0;

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

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

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

return \*pa - \*pb;

}

void step(int\* nums, int numsSize, int\* ans, int box, int\* res\_col, int ans\_index, int start)

{

int i;

if (box == numsSize) {

memcpy(res[g\_count], ans, sizeof(int) \* (ans\_index));

res\_col[g\_count] = ans\_index;

//memset(ans, 0, numsSize);

g\_count++;

return;

}

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

// 先记录当前数字

ans[ans\_index] = nums[i];

// 当前的数字当选了，box+1,ans\_index+1

step(nums, numsSize, ans, box + 1, res\_col, ans\_index + 1, i + 1);

// 当遇到重复的时候（i > 0 && nums[i] == nums[i - 1]）注意了，并且前面一个答案和当前的一样（ans[ans\_index - 1] == ans[ans\_index]）

// 就不让box+1

if (i > 0 && nums[i] == nums[i - 1] && ans\_index > 0 && ans[ans\_index - 1] == ans[ans\_index]){

continue;

}

// 当前的数字不当选，box仍然+1，认为处理了一个盒子，但是ans\_index不能加

step(nums, numsSize, ans, box + 1,res\_col, ans\_index, i + 1);

}

}

int\*\* subsetsWithDup(int\* nums, int numsSize, int\* returnSize, int\*\* returnColumnSizes){

int i =0;

int sum = 1;

int\* res\_col = NULL;

int\* ans = NULL;

g\_count = 0;

// 计算结果数

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

sum \*= 2;

}

\*returnSize = sum;

// 为结果分配空间

res = (int\*\*)malloc(sizeof(int\*) \* (\*returnSize));

res\_col = (int\*)malloc(sizeof(int) \* (\*returnSize));

memset(res\_col, 0, sizeof(int) \* (\*returnSize));

for (i = 0; i < \*returnSize; i++) {

res[i] = malloc(sizeof(int) \* (numsSize));

memset(res[i], 0, sizeof(int) \* numsSize);

}

// 保存单次的结果

ans = (int\*)malloc(sizeof(int) \* numsSize);

memset(ans, 0, numsSize);

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

step(nums, numsSize, ans, 0, res\_col, 0 ,0);

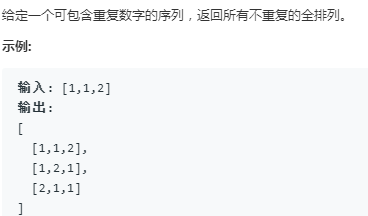
\*returnColumnSizes = res\_col;

\*returnSize = g\_count;

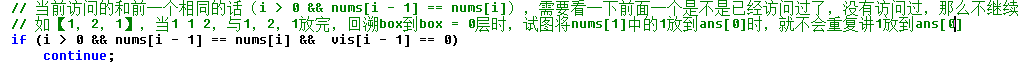
return res;

}

### 12.全排列II【重复元素排序去重】



思路：也是回溯类“盒子类型”，共numsSize个box.这里step定义是放一个数字到盒子，option是有3个选择。主要对比nums[]中无重复数字中这里需要排序qosrt+去重

。

代码：

#define N 100

int vis[N] = {0};

int ans[N] = {0};

int count = 0;

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

{

    int\* pa = (int\*)a;

    int\* pb = (int\*)b;

    return \*pa - \*pb;

}

void step(int \*nums, int numsSize, int\*\* res, int box)

{

    int i;

    if (box == numsSize) {

        memcpy(res[count], ans, sizeof(int) \* box);

        count++;

        return;

    }

    for (i = 0; i < numsSize; i++) {

        // 访问过的不继续

        if (vis[i] == 1) {

            continue;

        }

        // 当前访问的和前一个相同的话（i > 0 && nums[i - 1] == nums[i]），需要看一下前面一个是不是已经访问过了，没有访问过，那么不继续

        // 如【1，2，1】，当1 1 2，与1，2，1放完，回溯box到box = 0层时，试图将nums[1]中的1放到ans[0]时，就不会重复讲1放到ans[0]

        if (i > 0 && nums[i - 1] == nums[i] &&  vis[i - 1] == 0)

            continue;

        vis[i] = 1;

        ans[box] = nums[i];

        step(nums, numsSize, res, box +1);

        vis[i] = 0;

    }

}

int\*\* permuteUnique(int\* nums, int numsSize, int\* returnSize, int\*\* returnColumnSizes){

    int i = 0,j = 0;

    int \*\*res = NULL;

    int \*resCol = NULL;

    \*returnSize = 1;

    count = 0;

    // 计算空间大小

    for (i = numsSize; i > 0; i--) {

        \*returnSize \*= i;

    }

    res = (int\*\*)malloc(sizeof(int\*) \* (\*returnSize));

    memset(res, 0, sizeof(int\*) \* (\*returnSize));

    resCol = (int\*)malloc(sizeof(int) \* (\*returnSize));

    memset(resCol, 0, sizeof(int) \* (\*returnSize));

    for (i = 0; i < \*returnSize; i++) {

        res[i] = (int\*)malloc(sizeof(int) \* numsSize);

        memset(res[i], 0, sizeof(int) \* numsSize);

        resCol[i] = numsSize;

    }

    for (i = 0; i < 100; i++) {

        vis[i] = 0;

        ans[i] = 0;

    }

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

    step(nums, numsSize, res, 0);

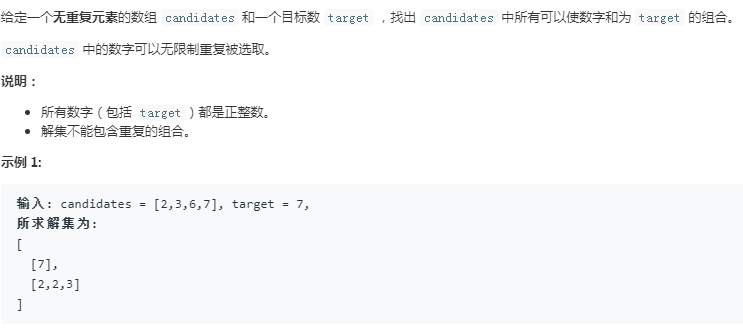
    \*returnColumnSizes = resCol;

    \*returnSize = count;

    return res;

}

### 13. [39. 组合总和](https://leetcode-cn.com/problems/combination-sum/)



思路：回溯，属于盒子类题目。Step的定义是网上叠加一个数，option是candidate中的元素选择，每次都有多种选择。目标是凑成target。

代码：

#define N 200

int g\_count = 0;

// 记录res[i]的大小

int res\_col[N] = {0};

// 保存每一个结果

int ans[N] = {0};

// 记录当前一组结果的个数

int g\_ans\_count = 0;

void Step(int\* candidates, int candidatesSize, int target, int\*\* res, int startIndex, int curSum)

{

int i;

if (curSum > target) {

return;

}

if (curSum == target) {

memcpy(res[g\_count], ans, sizeof(int) \* g\_ans\_count);

res\_col[g\_count] = g\_ans\_count;

g\_count++;

return;

}

for (i = startIndex; i < candidatesSize; i++) {

if (candidates[i] + curSum > target) {

continue;

}

ans[g\_ans\_count] = candidates[i];

g\_ans\_count++;

Step(candidates, candidatesSize, target, res, i, curSum + candidates[i]);

g\_ans\_count--;

}

}

int\*\* combinationSum(int\* candidates, int candidatesSize, int target, int\* returnSize, int\*\* returnColumnSizes)

{

int i, j;

int\*\* res = NULL;

res = (int\*\*)malloc(sizeof(int\*) \* N);

g\_count = 0;

g\_ans\_count = 0;

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

res\_col[i] = 0;

ans[i] = 0;

res[i] = (int\*)malloc(sizeof(int) \* N);

memset(res[i], 0, sizeof(int) \* N);

}

Step(candidates, candidatesSize, target, res, 0, 0);

\*returnSize = g\_count;

\*returnColumnSizes = res\_col;

return (int\*\*)res;

}

int main()

{

int i ,j;

int nums[] = { 2, 3, 6, 7, 8 };

int\*\* result = NULL;

int \*\*resCol = (int\*)malloc(sizeof(int\*));

int size = 0;

\*resCol = NULL;

result = combinationSum(nums, sizeof(nums) / sizeof(nums[0]), 7, &size, resCol);

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

for (j = 0; j < (\*resCol)[i]; j++) {

printf("%d ",result[i][j]);

}

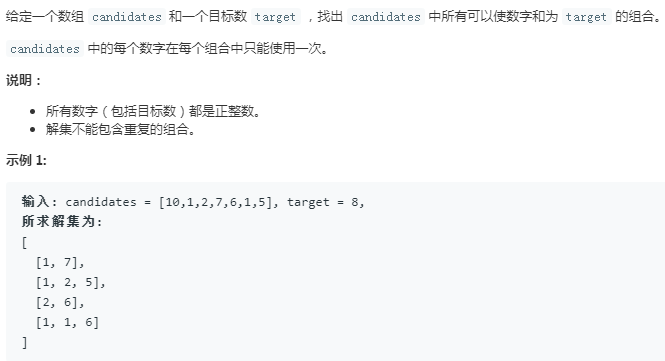
printf("\n");

}

printf("g\_count:%d\n",g\_count);

}

### 14：[40. 组合总和 II](https://leetcode-cn.com/problems/combination-sum-ii/)【排序去重】



思路：与题目39，组合总和，同属一类，加上排序、去重即可。



代码：

#define N 200

int ans[N] = {0};

int g\_ans\_count = 0;

int g\_count = 0;

int g\_resCol[N] = {0};

int vis[N] = {0};

void Step(int\* candidates, int candidatesSize, int target, int\*\* res, int startIndex, int curSum)

{

    int i;

    if (target == curSum) {

        memcpy(res[g\_count], ans, sizeof(int) \* g\_ans\_count);

        g\_resCol[g\_count] = g\_ans\_count;

        g\_count++;

    }

    for (i = startIndex; i < candidatesSize; i++) {

        if (vis[i] == 1) {

            continue;

        }

        if (i > 0 && candidates[i - 1] == candidates[i] && vis[i - 1] == 0) {

            continue;

        }

        if (candidates[i] + curSum > target) {

            continue;

        }

        vis[i] = 1;

        ans[g\_ans\_count] = candidates[i];

        g\_ans\_count++;

        Step(candidates, candidatesSize, target, res, i + 1, curSum + candidates[i]);

        vis[i] = 0;

        g\_ans\_count--;

    }

}

int\*\* combinationSum2(int\* candidates, int candidatesSize, int target, int\* returnSize, int\*\* returnColumnSizes){

    int\*\* res = NULL;

    int i;

    res = (int\*\*)malloc(sizeof(int\*) \* N);

    g\_count = 0;

    for (i = 0; i < N; i++) {

        res[i] = (int\*)malloc(sizeof(int) \* N);

        memset(res[i], 0 , sizeof(int) \* N);

        ans[i] = 0;

        vis[i] = 0;

        g\_resCol[i] = 0;

    }

    qsort(candidates, candidatesSize, sizeof(int), MyCmp);

    Step(candidates, candidatesSize, target, res, 0, 0);

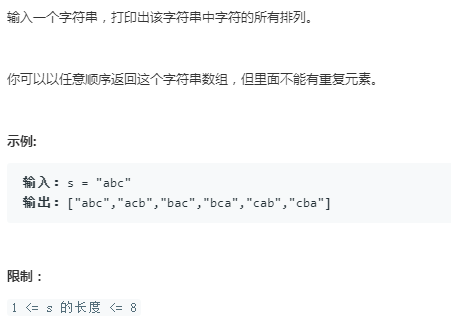
    \*returnSize = g\_count;

    \*returnColumnSizes = g\_resCol;

    return res;

}

### 15. [面试题38. 字符串的排列](https://leetcode-cn.com/problems/zi-fu-chuan-de-pai-lie-lcof/)【字符串排序并去重+递归回溯】



思路：盒子类问题，strlne(s)个盒子，step定义是填充一个盒子，即选定一个字母，option这里定义是可以从strlen(s)个字母选择。

注意qsort对字符串排序，要求第一个参数是可以改的，如直接对参数“dcf”进行调用会失败。

代码：

#define N 10

int g\_count = 0;

int vis[N] = {0};

int CountSize(int n) {

    if (n == 0 || n == 1) {

        return 1;

    }

    return n \* CountSize(n - 1);

}

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

{

    char\* pa = (char\*)a;

    char\* pb = (char\*)b;

    return \*pa - \*pb;

}

void Step(char\*\* res, char\* s, char\* ans, int box)

{

    int i;

    int len = strlen(s);

    if (box == len) {

        memcpy(res[g\_count], ans, box);

        res[g\_count][box] = '\0';

        g\_count++;

        return;

    }

    for  (i = 0; i < len; i++) {

        if (vis[i] == 1) {

            continue;

        }

        if (i > 0 && s[i - 1] == s[i] && vis[i - 1] == 0) {

            continue;

        }

        vis[i] = 1;

        ans[box] = s[i];

        Step(res, s, ans, box + 1);

        vis[i] = 0;

    }

}

char\*\* permutation(char\* s, int\* returnSize){

    char\*\* res = NULL;

    char\* ans = NULL;

    char\* temp = NULL;

    int i;

    int len = strlen(s);

    g\_count = 0;

    \*returnSize = CountSize(len);

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

    for (i = 0; i <\*returnSize; i++){

        res[i] = (char\*)calloc(len + 1, sizeof(char));

    }

    ans = (char\*)calloc(len + 1, sizeof(char));

    temp = (char\*)calloc(len + 1, sizeof(char));

    memcpy(temp, s, len + 1);

    qsort(temp, len, sizeof(char), MyCmp);

    //printf("%s\n", temp);

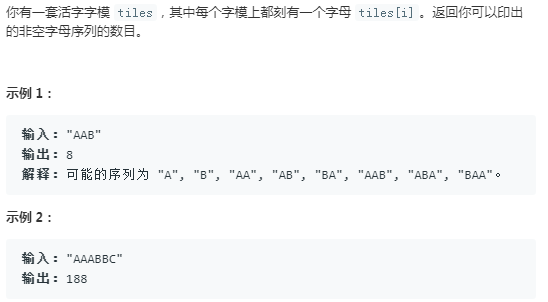
    Step(res, temp, ans, 0);

    \*returnSize = g\_count;

    return res;

}

### 16. [1079. 活字印刷](https://leetcode-cn.com/problems/letter-tile-possibilities/)【递归回溯+for循环+去重类131&1239】



思路：

For循环控制maxBox,step就是选择一个元素，option是有0-strlen(tiles)个选择，过程中注意去重。

代码：

    int g\_count = 0;

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

    {

        char\* pa = (char\*)a;

        char\* pb = (char\*)b;

        return \*pa - \*pb;

    }

    void Dfs(char\* tiles, char\* ans, int\* vis, int maxBox, int box)

    {

        int i;

        int len = strlen(tiles);

        if (maxBox == box) {

            g\_count++;

            return;

        }

        for (i = 0; i < len; i++) {

            if (vis[i] == 1) {

                continue;

            }

            if (i > 0 && tiles[i - 1] == tiles[i] && vis[i - 1] == 0) {

                continue;

            }

            vis[i] = 1;

            ans[box] = tiles[i];

            Dfs(tiles, ans, vis, maxBox, box + 1);

            vis[i] = 0;

        }

    }

    int numTilePossibilities(char \* tiles)

    {

        int i = 0;

        int len  = strlen(tiles);

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

        char\* tmpStr = (char\*)calloc(len + 1, sizeof(char));

        char\* ans = (char\*)calloc(len, sizeof(char));

        memcpy(tmpStr, tiles, len + 1);

        qsort(tmpStr, len, sizeof(char), MyCmp);

        //printf("%s\n", tmpStr);

        g\_count = 0;

        for (i = 1; i <= len; i++) {

            memset(vis, 0, len \* sizeof(int));

            memset(ans, 0, len \* sizeof(char));

            Dfs(tmpStr, ans, vis, i, 0);

        }

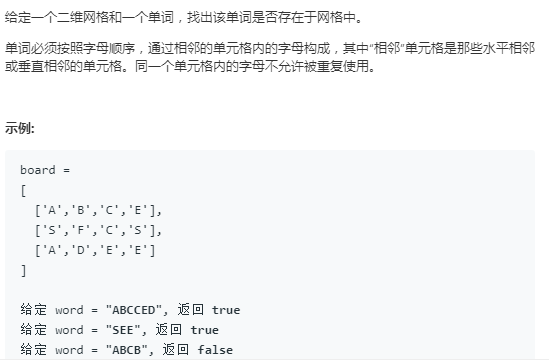
        free(vis);

        free(tmpStr);

        return g\_count;

    }

### 17. [79. 单词搜索](https://leetcode-cn.com/problems/word-search/)【迷宫类型/for循环+递归】



思路：for循环控制每个数组元素都当作起点，进行回溯递归。属于迷宫类型，结束条件是走了word长度的步数，并且每一步要走对。

#define N 200

int vis[N][N] = {0};

static int dir[][2] = {

{0, -1},

{0,1},

{1,0},

{-1,0},

};

bool Step(char\*\* board, int boardSize, int colSize, char\* word, int box, int curX, int curY)

{

int i = 0,j = 0;

int nextX, nextY;

int res = 0;

// 保证每一步走对

if (board[curX][curY] != word[box]) {

return false;

}

if (box == strlen(word) - 1) {

return true;

}

for (i = 0; i < 4; i++) {

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

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

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

continue;

}

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

continue;

}

vis[nextX][nextY] = 1;

res = Step(board, boardSize, colSize, word, box + 1, nextX, nextY);

if (res) {

return true;

}

vis[nextX][nextY] = 0;

}

return false;

}

bool exist(char\*\* board, int boardSize, int\* boardColSize, char\* word){

int i, j, res, k;

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

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

vis[i][j] = 0;

}

}

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

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

// 每次递归前进行对vis清零

for (k = 0; k < boardSize; k++) {

memset(vis[k], 0 , sizeof(int) \* boardColSize[0]);

}

vis[i][j] = 1;

res = Step(board, boardSize, boardColSize[0], word, 0, i, j);

if (res) {

return res;

}

}

}

return false;

}

int main()

{

int i, j;

char \*\*board = NULL;

/\*char board1[][4] = {

{'A','B','C','E'},

{'S','F','C','S'},

{'A','D','E','E'},

};\*/

char board1[][2] = {

{'a', 'b'},

{'c', 'd'},

};

/\*char board1[][6] = {

{'b','a','a','b','a','b'},

{'a','b','a','a','a','a'},

{'a','b','a','a','a','b'},

{'a','b','a','b','b','a'},

{'a','a','b','b','a','b'},

{'a','a','b','b','b','a'},

{'a','a','b','a','a','b'},

};\*/

int\* colSize = NULL;

int res = 0;

board = (char\*\*)malloc(sizeof(char\*) \* sizeof(board1) / sizeof(board1[0]));

colSize = (int\*)malloc(sizeof(int) \* sizeof(board1) / sizeof(board1[0]));

for (i = 0; i < sizeof(board1) / sizeof(board1[0]); i++) {

board[i] = (char\*)malloc(sizeof(char) \* sizeof(board1[0]));

memcpy(board[i], board1[i], sizeof(board1[0]));

colSize[i] = sizeof(board1[0]);

}

for (i = 0; i < sizeof(board1) / sizeof(board1[0]); i++) {

for (j = 0; j < sizeof(board1[0]); j++) {

printf("%c ",board[i][j]);

}

printf("\n");

}

res = exist(board, sizeof(board1) / sizeof(board1[0]), colSize, "cdba");//aabbbbabbaababaaaabababbaaba

printf("%d\n",res);

for (i = 0; i < sizeof(board1) / sizeof(board1[0]); i++) {

free(board[i]);

board[i] = NULL;

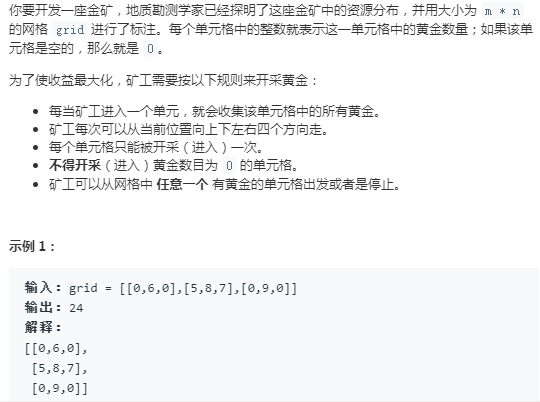
}

free(colSize);

free(board);

}

### 18. [1219. 黄金矿工](https://leetcode-cn.com/problems/path-with-maximum-gold/)【迷宫类型+递归回溯全搜索】



思路：

其实按题目的意思可以知道需要从每个非0的起点开始搜索，截止条件不固定，需要搜索到不能继续下一个节点为止。并且过程中对当前的cur进行回溯，有最大的往g\_max上即可。

代码：

int g\_max = -1;

int g\_count = 0;

int g\_dir[][2] = {

    {0, -1},

    {0, 1},

    {-1,0},

    {1,0},

};

void Dfs(int\*\* grid, int\*\* vis, int maxX, int maxY, int x, int y, int cur)

{

    int i, j;

int nextX, nextY;

//不管哪个循环，一路搜出最大的就行

g\_max = (g\_max < cur) ? (cur) : (g\_max);

// 四个方向

    for (i = 0; i < sizeof(g\_dir) / sizeof(g\_dir[0]); i++) {

        nextX = x + g\_dir[i][0];

        nextY = y + g\_dir[i][1];

        // 非法的不用继续

        if (nextX < 0 || nextY < 0 || nextX >= maxX || nextY >= maxY) {

            continue;

        }

        // 访问过的不挖

        if (vis[nextX][nextY] == 1) {

            continue;

        }

        // 为0的不去挖，并标记为永不访问

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

            vis[nextX][nextY] = 1;

            continue;

        }

        vis[nextX][nextY] = 1;

        Dfs(grid, vis, maxX, maxY, nextX, nextY, cur + grid[nextX][nextY]);

        vis[nextX][nextY] = 0;

    }

}

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

{

    int row = gridSize, col = gridColSize[0];

    int i, j, k;

    int \*\*vis = NULL;

    int\*\*\* router =NULL;//(int\*\*\*)calloc(row , sizeof(int\*\*));

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

    g\_max = -1;

    for(i = 0; i < row; i++) {

        vis[i] = (int\*)calloc(col, sizeof(int));

}

// 递归查找非0起点

    for (i = 0; i < row; i++) {

        for (j = 0; j < col; j++) {

            if (grid[i][j] == 0) {

                continue;

            }

            for (k = 0; k < row; k++) {

                memset(vis[k], 0, col \* sizeof(int));

            }

// 每次起点置为已访问

            vis[i][j] = 1;

            if (i == 4 && j == 2) {

                printf("%d %d:%d\n",i,j,vis[3][5]);

            }

            Dfs(grid, vis, row, col, i , j, grid[i][j]);

        }

    }

    for(i = 0; i < row; i++) {

        free(vis[i]);

    }

    free(vis);

    return g\_max;

}

int main()

{

    int res = 0;

    int\* colSize = NULL;

    /\*int tmpGrid[][3] = {

        {0,6,0},

        {5,8,7},

        {0,9,0},

    };\*/ //24

    int tmpGrid[][6] = {

        {1,0,7,0,0,0},

        {2,0,6,0,1,0},

        {3,5,6,7,4,2},

        {4,3,1,0,2,0},

        {3,0,5,0,20,0},

    };//从(4,2)->(3,2)->(3,1)->(3,0)->(2,0)->(2,1)->(2,2)->(2,3)->(2,4)->(3,4)->(4,4)

int i ,j;

// 开一个grid数组，模拟leetcode输入

    int\*\* grid = (int\*\*)calloc(sizeof(tmpGrid) / sizeof(tmpGrid[0]), sizeof(int\*));

    colSize = (int\*)calloc(sizeof(tmpGrid) / sizeof(tmpGrid[0]), sizeof(int));

    memset(colSize, 0, sizeof(tmpGrid[0]) / sizeof(tmpGrid[0][0]) \* sizeof(int));

    colSize[0] = sizeof(tmpGrid[0]) / sizeof(tmpGrid[0][0]);

    for (i = 0; i < sizeof(tmpGrid) / sizeof(tmpGrid[0]); i++) {

        grid[i] = (int\*)calloc(sizeof(tmpGrid[0]) / sizeof(tmpGrid[0][0]), sizeof(int));

        memcpy(grid[i], tmpGrid[i], sizeof(tmpGrid[0]) / sizeof(tmpGrid[0][0]) \* sizeof(int));

}

    for (i = 0; i < sizeof(tmpGrid) / sizeof(tmpGrid[0]); i++) {

        for (j = 0; j < sizeof(tmpGrid[0]) / sizeof(tmpGrid[0][0]); j++) {

            printf("%d ",grid[i][j]);

        }

        printf("\n");

    }

    res = getMaximumGold(grid, sizeof(tmpGrid) / sizeof(tmpGrid[0]), colSize);

    for (i = 0; i < sizeof(tmpGrid) / sizeof(tmpGrid[0]); i++) {

        free(grid[i]);

    }

    free(grid);

    grid = NULL;

    free(colSize);

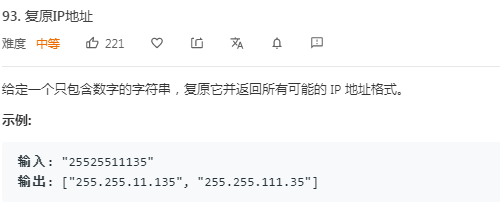
    colSize = NULL;

    printf("%d\n", res);

    return 0;

}

### 19. [93. 复原IP地址](https://leetcode-cn.com/problems/restore-ip-addresses/)【字符串操作+递归回溯】



/\* 思路：回溯+剪枝

1.采用盒子的方法，也就是放四个小数点

2.step的定义是放一个小数点

3.option是有三个选择，如果当前的索引是startIndex,那么小数点可以从startIndex-strlen(s)选择, i < startIndex +3属于剪枝操作

4.每一个option都从s的startIndex位置截取字符，判断值如果大于255就退出；如果截取下来的首字母为0，但 如果长度不是1，也是非法情况

5.结束条件是box为4，即放了四个小数点；并且需要保证startIndex不能小于strlen(s),即情况255.2.5.5.11133

\*/

#define RES\_MAX 50

int g\_count = 0;

void MySubstr(char \*str, char\* dst, int startIndex, int len)

{

    if (str == NULL) {

        dst[0] = 0;

        return;

    }

    memcpy(dst, (char\*)&str[startIndex], sizeof(char) \* len);

    dst[len] = '\0';

    return;

}

void step(char\* s, char\*\* res, char\* ans, int ans\_index, int box, int startIndex)

{

    int i;

    int len = strlen(s);

    char temp[50] = {0};

    if (box == 4) {

        if (startIndex < strlen(s)) {

            return;

        }

        // ans\_index - 1是为了不把最后一个小数点给带上

        memcpy(res[g\_count], ans, ans\_index - 1);

        g\_count++;

        return;

    }

    // option是从startIndex到startIndex + 3中插入

    for (i = startIndex; i < len && i < startIndex + 3; i++) {

        MySubstr(s, temp, startIndex, i - startIndex + 1);

        if (atoi(temp) > 255) {

            return;

        }

        // 规避00. 及01.及001.的情况

        if (temp[0] == '0' && strlen(temp) != 1) {

            return;

        }

        sprintf((char\*)&ans[ans\_index], "%s%s", temp, ".");

        // ans\_index需要从ans\_index + temp的长度的后一个开始

        step(s, res, ans, ans\_index + i - startIndex + 2, box + 1, i + 1);

    }

}

char \*\* restoreIpAddresses(char \* s, int\* returnSize){

    char\*\* res = NULL;

    char\* ans = NULL;

    int i;

    int len;

    g\_count = 0;

    len = strlen(s);

    if (len >= 13) {

        \*returnSize = 0;

        res = (char\*\*)malloc(sizeof(char\*));

        res[0] = (char\*)malloc(sizeof(char));

        res[0][0] = '\0';

        return res;

    }

    res = (char\*\*)malloc(sizeof(char\*) \* RES\_MAX);

    for (i = 0; i < RES\_MAX; i++) {

        res[i] = (char\*)malloc(sizeof(char) \* (len + 5));

        memset(res[i], 0, sizeof(char) \* (len + 5));

    }

    ans = (char\*)malloc(sizeof(char) \* (len + 5));

    memset(ans, 0, sizeof(char) \* (len + 5));

    step(s, res, ans, 0, 0, 0);

    free(ans);

    \*returnSize = g\_count;

    return res;

}

int main()

{

    char\*\* res = NULL;

    int size = 0;

    char s[] = "010010";//"25525511135";

    int i;

    res = restoreIpAddresses(s, &size);

    for (i = 0; i < size; i++) {

        printf("%s\n", res[i]);

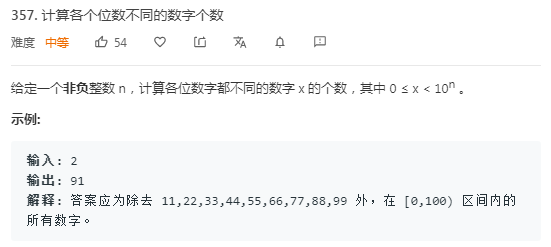
    }

    printf("%d\n", size);

    return 0;

}

### 20.[357. 计算各个位数不同的数字个数](https://leetcode-cn.com/problems/count-numbers-with-unique-digits/)【简单总结加递归方法】



/\* 思路：

把n位数到1位数的每位不同数字的个数累加可以得到结果。

对于n位数，如3位数，每位都不同的数字其实是（10-1）\*（10-1）\*（10-2），最高位10中选择，但是不能选0因此减掉1；

因此一般写法和递归写法都可以

\*/

int CountN(int n)

{

    int sum = 1;

    int i;

    if (n == 0) {

        return 1;

    }

    for (i = 0; i < n; i++) {

        if (i == 0) {

            sum \*= 9;

        } else {

            sum \*= (10 - i);

        }

    }

    return sum + CountN(n -1);

}

int countNumbersWithUniqueDigits(int n){

    //return  CountN(n);

}

一般写法

int CountN1(int n)

{

    int i;

    int sum = 1;

    for (i = 0; i < n; i++) {

        if (i == 0) {

            sum \*= 9;

        }

        else {

            sum \*= (10 -i);

        }

    }

    return sum;

}

int countNumbersWithUniqueDigits(int n){

    //return  CountN(n);

    int i;

    int sum = 0;

    if (n == 0) {

        return 1;

    }

    if (n == 1) {

        return 10;

    }

    sum = 10;

    for (i = 2; i <= n; i++) {

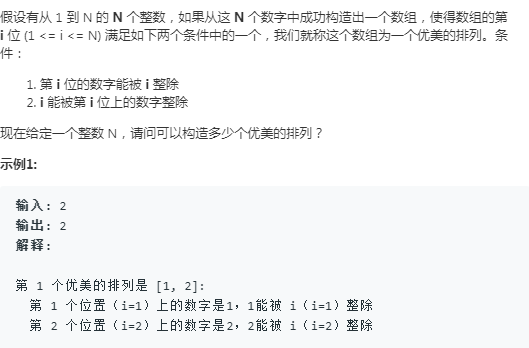
        sum += CountN1(i);

    }

    return sum;

}

### 21. [526. 优美的排列](https://leetcode-cn.com/problems/beautiful-arrangement/)【回溯递归】



思路：step就是选取元素，option就是从1-N去选择。判断条件是满足1或2，否则就不去访问。

#define MAX 100

int g\_count = 0;

int vis[MAX] = {0};

void Step(int N, int box)

{

    int value;

    if (box == N + 1) {

        g\_count++;

        return;

    }

    for (value = 1; value <= N; value++) {

        if (vis[value] == 1) {

            continue;

        }

        if (value % box == 0 || box % value == 0) {

            vis[value] = 1;

            Step(N, box + 1);

            vis[value] = 0;

        }

    }

}

int countArrangement(int N)

{

    int i;

    g\_count = 0;

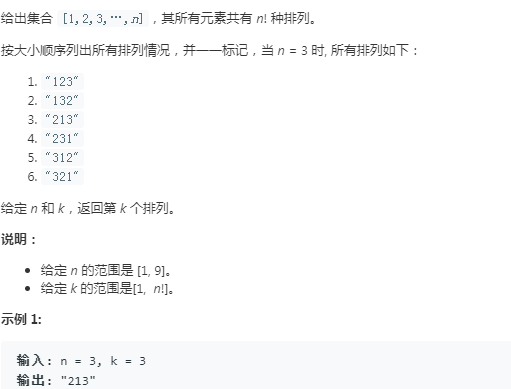
    memset(vis, 0, sizeof(int) \* MAX);

    Step(N, 1);

    return g\_count;

}

### 22. [60. 第k个排列](https://leetcode-cn.com/problems/permutation-sequence/)【递归回溯】



#define MAX 100

int g\_count = 0;

int vis[MAX] = {0};

void step(char \*res, char\* ans, int n, int k, int box)

{

    int i;

    if (g\_count >= k) {

        return;

    }

    if (box == n) {

        ans[box] = '\0';

        g\_count++;

        if (g\_count == k) {

            memcpy(res, ans, strlen(ans));

        }

        return;

    }

    for (i = 1; i <= n; i++) {

        if (vis[i] == 1) {

            continue;

        }

        vis[i] = 1;

        ans[box] = i + '0';

        step(res, ans, n, k, box + 1);

        vis[i] = 0;

    }

}

char \* getPermutation(int n, int k)

{

    char \*res = NULL;

    char \*ans = NULL;

    g\_count = 0;

    res = (char\*)calloc(n + 1, sizeof(char));

    ans = (char\*)calloc(n + 1, sizeof(char));

    memset(vis, 0, sizeof(int) \* MAX);

    step(res, ans, n, k, 0);

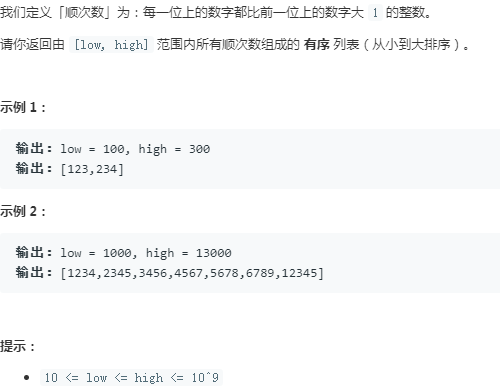
    free(ans);

    ans = NULL;

    return res;

}

### 23. [1291. 顺次数](https://leetcode-cn.com/problems/sequential-digits/)【for循环+递归】



思路：容易想到，对于数字1，如果让他进行连续增加1，并且需要累计成值与low和high比较。很容易想到for循环+递归方式。外层For循环控制数字从1遍历到9，每个遍历递归方式去计算累计和，并控值index是递增且梯度为1（curSum \* 10 + (curSum) % 10 + 1）。

代码一：

#define MAX 100

int g\_count = 0;

// 循环加递归

void Dfs(int \*res, int low, int high, int curSum)

{

    if (curSum > high) {

        return;

    }

    if (curSum >= low && high <= high) {

        res[g\_count] = curSum;

        g\_count++;

    }

    // 先预判

    if (curSum % 10 == 9) {

        return;

    }

    // curSum % 10 + 1 控制1234是连续的

    Dfs(res, low, high, curSum \* 10 + curSum % 10 + 1);

}

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

{

    int \*pa = (int\*)a;

    int \*pb = (int\*)b;

    return \*pa - \*pb;

}

int\* sequentialDigits(int low, int high, int\* returnSize){

    int \*res = NULL;

    int i;

    g\_count= 0;

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

    //ForValue(res, low, high);

    for (i = 1; i <= 9; i++) {

        Dfs(res, low, high, i);

    }

    \*returnSize = g\_count;

    qsort(res, g\_count, sizeof(int), MyCmp);

    return res;

}

方法二：

#define MAX 100

int g\_count = 0;

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

{

    int \*pa = (int\*)a;

    int \*pb = (int\*)b;

    return \*pa - \*pb;

}

void ForValue(int \*res, int low, int high)

{

    int i, j;

    int curSum = 0;

    for (i = 1; i < 10; i++) {

        curSum = 0;

        for (j = i; j < 10; j++) {

            curSum = curSum \* 10 + j;

            if (curSum < low) {

                continue;

            } else if (curSum > high) {

                break;

            } else {

                res[g\_count] = curSum;

                g\_count++;

            }

        }

    }

}

int\* sequentialDigits(int low, int high, int\* returnSize){

    int \*res = NULL;

    g\_count= 0;

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

    ForValue(res, low, high);

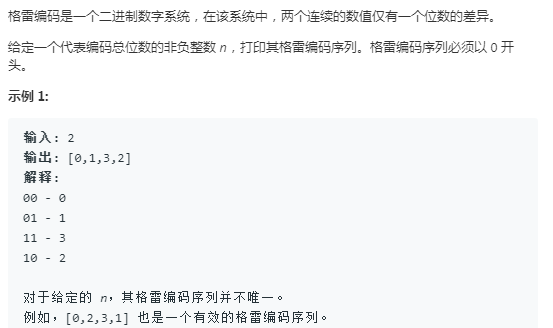
    \*returnSize = g\_count;

    qsort(res, g\_count, sizeof(int), MyCmp);

    return res;

}

### 24. [89. 格雷编码](https://leetcode-cn.com/problems/gray-code/)【只改变一位用异或】



思路：

首先某个数的下个格雷的编码，其实只要报证下个数在位上与该数上只有一位差异即可，而只改变某一位用异或（0 ^ 1 = 1; 1 ^ 1 = 0）.

因此本题容易step的定义就是求出一个格雷编码，option就是bit位的变化从0 – n选择。直到找到2 << n – 1个步骤截止。

注意：在这题中，每个bit进行step之后，这里一定要return，不能在同层继续去移位，否则会进行大量的重复计算。

void Step(int \*res, int \*vis, int n, int box)

{

    int i;

    int next;

    if (box == ((1 << n) -1)) {

        return;

    }

    for (i = 0; i < n; i++) {

        next = res[box] ^ (1 << i);

        if (vis[next] == 1) {

            continue;

        }

        vis[next] = 1;

        res[box + 1] = next;

        Step(res, vis, n, box + 1);

        vis[next] = 0;

        return;  // 这一句一定不能少，否则找到了一组，还会继续for循环往下

    }

}

int\* grayCode(int n, int\* returnSize)

{

    int \*res = NULL;

    int \*vis = NULL;

    \*returnSize = 1 << n;

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

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

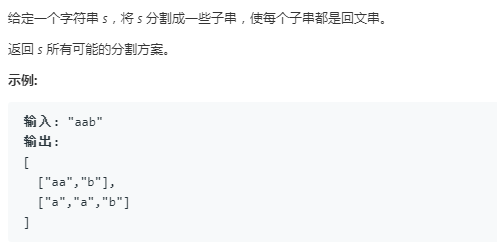
    vis[0] = 1;

    Step(res, vis , n, 0);

    return res;

}

### 25. [131. 分割回文串](https://leetcode-cn.com/problems/palindrome-partitioning/)【递归回溯+for循环+组合变形】



思路：

可以将他看成是这种问题：和为整数len的，有多少种组合方式。用递归回溯很好理解

如长度为3 （如串aba）有如下几种方式：

1+1+1 对应"a","b","a"

1+2 对应“a”,"ba" 舍弃

2+1对应“aa”，“b”

3对应“aba”。

其中：每次递归都是去凑curSum 使其等于len,过成中去判断当前一小步即当前的串（&s[curSum] 到 &s[curSum + i - 1]）是不是回文的。

代码:

#define N 250

int g\_count = 0;

char\*\*\* res = NULL;

int\* res\_col = NULL;

int IsLoopBackStr(char \*s, int start, int end)

{

    while(start < end && start < strlen(s) && end >= 0) {

        if (s[start] != s[end]) {

            return 0;

        }

        start++;

        end--;

    }

    return 1;

}

void Step(char\* s, int len, char\*\* tmpStr, int curSum, int box)

{

    int i;

    int temp;

    if (curSum == len) {

        for (i = 0; i < box; i++) {

            temp = strlen(tmpStr[i]);

            memcpy(res[g\_count][i], tmpStr[i], temp);

            res[g\_count][i][temp] = '\0';

        }

        res\_col[g\_count] = box;

        g\_count++;

        return;

    }

    for (i = 1; i <= len && i + curSum <= len; i++) {

        // 看当前的步进值是不是回文的，是回文，才允许进去临时结果tmpStr中

if (!IsLoopBackStr(s, curSum, curSum + i - 1)) {

            continue;

        }

        memcpy(tmpStr[box], &s[curSum], i);

        tmpStr[box][i] = '\0';

        Step(s, len, tmpStr, curSum + i, box + 1);

    }

}

char \*\*\* partition(char \* s, int\* returnSize, int\*\* returnColumnSizes)

{

    int i, j;

    int len = strlen(s);

    char\*\* tmpStr = NULL;

    g\_count = 0;

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

    res\_col = (int\*)calloc(N, sizeof(int));

    tmpStr = (char\*\*)calloc(N, sizeof(char\*));

    for (i = 0; i < N; i++) {

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

        tmpStr[i] = calloc(N, sizeof(char));

        for (j = 0; j < N; j++) {

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

        }

    }

    Step(s, len, tmpStr, 0, 0);

    free(tmpStr);

    tmpStr = NULL;

    \*returnSize = g\_count;

    \*returnColumnSizes = res\_col;

    return res;

}

### 25. [1239. 串联字符串的最大长度](https://leetcode-cn.com/problems/maximum-length-of-a-concatenated-string-with-unique-characters/)【递归回溯+for循环+排列变形】



思路：

可以用排列的思想，也就是盒子。for循环控制组合的个数，dfs控制组合个数为n时的排列种类。

如["cha","r","act","ers"]：

组合个数为4：

其实只有一种。即元素0，1，2，3都选上 每次选取时新选取的都要去和当前ans比较如果有重复不继续

组合个数为3：同样每次选取时都和现有的ans比较，如果有重复立即停止

0，1，2 cha,r,act 选act时就发现不行了

0，1，3

0，2，3

1，2，3

注意：除了新选取的和现有ans比，还要判断当前元素是否自重复，比如yy，这个可以在dfs前对

输入数组处理，把重复的元素的下表直接设置为“已访问”

int g\_max = -1;

int\* vis =NULL;

int\* visBack = NULL;

// 判断str2中的字符是否在str1中出现了

int IsStr2SomeInStr1(char\* str1, char\* str2)

{

int i;

int len = strlen(str2);

if (str1 == NULL) {

return 0;

}

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

if (strchr(str1, str2[i]) != NULL) {

return 1;

}

}

return 0;

}

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

{

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

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

return \*pa - \*pb;

}

// 判断自重复

int IsSelfCom(char\* s)

{

char temp[50] = {0};

int i;

int len = strlen(s);

if (len < 2) {

return 0;

}

memcpy(temp, s, len);

qsort(temp, len, sizeof(char), MyCmp);

for (i = 0; i < len - 1; i++) {

if (temp[i] == temp[i + 1]) {

return 1;

}

}

return 0;

}

void Step(char\*\* arr, int size, int maxBox, int startIndex, char\* ans, int ansIndex, int box)

{

int i;

int tmp;

if (box == maxBox) {

tmp = strlen(ans);

g\_max = (g\_max < tmp) ? tmp : g\_max;

return;

}

for (i = startIndex; i < size; i++) {

// vis[1] == 1表示已访问或者自重复 || 新加入的不能与当前的重复

if (vis[i] == 1 || IsStr2SomeInStr1(ans, arr[i])) {

continue;

}

vis[i] = 1;

memcpy(&ans[ansIndex], arr[i], strlen(arr[i]) + 1);

Step(arr, size, maxBox, i + 1, ans, ansIndex + strlen(arr[i]), box + 1);

// 要把ans中的新增段给复原

memset(&ans[ansIndex], 0, strlen(arr[i]) + 1);

ans[ansIndex] = '\0';

vis[i] = 0;

}

}

int maxLength(char\*\* arr, int arrSize)

{

int i;

int maxLen = 0;

vis = calloc(arrSize, sizeof(int));

visBack = calloc(arrSize, sizeof(int));

char\* ans = NULL;

g\_max = 0;

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

// 对于自重复直接置为已访问，并且长度也只加

if (IsSelfCom(arr[i])) {

visBack[i] = 1;

continue;

}

maxLen += strlen(arr[i]);

}

ans = calloc(maxLen + 1, sizeof(char));

// for循环控制组合的个数

for (i = arrSize; i > 0; i--) {

memset(ans, 0, maxLen + 1);

memcpy(vis, visBack, arrSize \* sizeof(int));

Step(arr, arrSize, i, 0, ans, 0, 0);

}

return g\_max;

}

int main()

{

char\* arr[] = {"yy","bkhwmpbiisbldzknpm"};//{"cha","r","act","ers"};

int len = maxLength(arr, sizeof(arr) / sizeof(arr[0]));

printf("%d\n", len);

}

### 26. [306. 累加数](https://leetcode-cn.com/problems/additive-number/)【递归回溯+组合变形】



思路：同样可以将长度len拆分为和为len的加法公式有多少种组合。这个题靠后的用例数字比较大，因此整形需要用unsigned long long 即atoll()

在拆的过程中注意判断条件：

1. 当box>2也即解集中数组为2个整数以上，就要判断规则是否可可行
2. 当前拆分下来的字符串长度为1并且首字符如果为0为非法
3. 当发现当前拆分下来的字符串转化的数字比前2个和都大那么就要返回false
4. 注意判断条件，这个写法是当找到一种解集，递归就不会继续了，因为函数栈的上面都在等这个rs.

rs = Step(num, ans, res, box + 1, curSum + i);

if (rs == 1)

return 1;

代码：

void GetFromStr(char\* source, int start, int size, char\* target)

{

    int len = strlen(source);

    if (start > len || size > len) {

        return;

    }

    memcpy(target, &source[start], size);

    target[size] = '\0';

}

int Step(char\* num, char\* ans, unsigned long long\* res, int box, int curSum)

{

    int i;

    unsigned long long temp;

    int len = strlen(num);

    int rs = 0;

    if (curSum == len && box > 2) {

        for (i = 0; i < box; i++) {

            printf("%l ",res[i]);

        }

        return 1;

    }

    for (i = 1; i <= len && i + curSum <= len; i++) {

        // 截取字符串

        GetFromStr(num, curSum, i, ans);

        // 如果此次截取的长度非1，并且首字母是0则不符合

        if (ans[0] == '0' && i != 1) {

            continue;

        }

        temp = atoll(ans);

        // 如果res中已经有2个结果了，那么需要判断是否符合2个之和是否为当前这个数

        if (box > 1 && res[box - 2] + res[box - 1] < temp) {

            return 0;

        }

        if (box > 1 && res[box - 2] + res[box - 1] > temp) {

            continue;

        }

        res[box] = temp;

        // 这个写法可以在找到一个结果的时候，也让上层函数栈返回结果，因此上层直接在下面判断这个rs，就不会继续

        rs = Step(num, ans, res, box + 1, curSum + i);

        if (rs == 1) {

            return 1;

        }

    }

    return 0;

}

int main()

{

int rs = 0;

rs=isAdditiveNumber("121474836472147483648");

//2461016264268110179;121474836472147483648

printf("\n%d\n", rs);

}