# 一、深度优先搜索

深度优先搜索算法(Depth First Search)，是图论中的经典算法。

深度优先搜索算法是**沿着树的深度遍历树的节点，尽可能深的搜索树的分支**。当结点所有子结点那一层都被搜索过，再回溯返回到当前结点的邻结点，继续搜索，直到遍历完整棵树。**一般采用的是前序遍历，先根然后再左右结点的方式进行。**

一些经典的问题，比如八皇后、马走日、迷宫等，都可以通过深度优先搜索算法来解决。

为了方便描述，下文用DFS来做为深度优先搜索算法的简称。

# 二、对DFS的认识

对于DFS，我相信很多人第一次接触很难设计出相应的算法，即便是有不错的编程经验。我第一次几乎没办法设计出解决八皇后的算法，即便是想了很久。最后没办法只好参照别人写的递归式的DFS。之后，虽然对这个算法有一点了解，但由于了解不够深度，过了几天就记不得了，下次又完全不知道怎么入手。然后需要再到网上搜下代码，看一遍后大概才双知道。而且发现每次写代码的时候心里总觉得不踏实，一开始总有错误的地方，并且每次写的代码都有些不同。总之，写过很多次后，依然是停留到了解的阶段，没办法进一步提升，特别是非递归式的DFS一直都停留到靠脑力记忆而不是理解的阶段。

今天周末有点时间，觉得有必要解决这些问题，试着花时间去归纳总结DFS的本质，看能否做到一劳永逸。

## 我设定的目标是：

1、不仅停留到理解阶段，而是要知道这个算法每一步的实现

2、捉住其中的本质，给出这个算法的设计框架。

3、在1与2的基础中，可以熟练写出递归与非递归两种实现方式 。

**经过一个下午的研究，我发现任何DFS只需要通过下面几步就可以实现，无论是递归还是非递归方式。我给这几步分别做了一个命名，分别是find、forward、done、back。**

如下：

1、find(right):在树的当前层，横向遍历，尝试找到ok的节点。（这一步通常被叫做剪枝，只留下ok的。）

2、forward(down):若在当前层找到ok的结点，并且当前层不是最后一层:把ok的节点放到当前层；进入下一层第一个结点。跳到find

3、done(right):若在当前层找到ok的结点，并且当前层是最后一层：打印出结果；进入当前层的下一个结点。跳到find

4、back(up):在当前层没有找到ok的节点：返回上一层当前结点的下一个兄弟节点。跳到find

其实最重要的是find。然后后面的forward、done、back只是用来控制搜索走向。这四步可以进一步总结成两步。 为了了解算法，我想最好的切入方式是从一些实例开始。下面分别从八皇后以及马走日等问题做为切入点来分析DFS

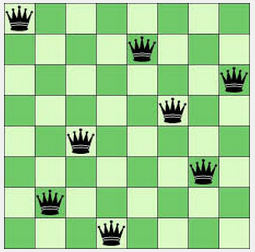
# 三、用DFS解八皇后

## 1、问题描述

八皇后问题是一个以国际象棋为背景的问题：如何能够在8×8 的国际象棋棋盘上放置八个皇后，使得任何一个皇后都无法直接吃掉其他的皇后？

也就是说，使得棋盘中每个横向、纵向、左上至右下斜向、右上至左下斜向均只有一枚皇后。

八皇后有92组解，下面给出其中一种解的图例：



## 2、 问题分析

规则是每一个皇后与前面的所有皇后不能在同一行、同一列、同一对角线。我们可以从第0行，第0列开始摆放，然后按照深度优先的原则，按照规则往更下面的行摆放皇后，直到摆放完8行。因为解不只一个，当某一行（包括最后一行跟最后一行之前的所有行）的所有列都被尝试过，再回溯返回到上一行，继续深度优先，直到遍历完整个棋盘的所有情况。得出所有的解。 八皇后问题可以看成是在深度为8的8叉树中，找出所有的解。

## 3、代码实现

递归算法：[dfs\EightQueenUsingRecursive.cpp](dfs/EightQueenUsingRecursive.cpp)

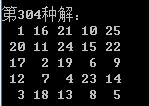
非递归算法：[dfs\EightQueenUsingStack.cpp](dfs/EightQueenUsingStack.cpp)

# 四、马走日

## 1、问题描述

在n\*n的棋盘中，马只能走"日"字。马从位置(0,0)出发，把棋盘的每一格都走一次且只走一次。找出所有路径。 5\*5的棋盘上，有304种解。

下面是其中一种路径的图例：



## 2、问题分析

搜索过程是从(0,0)出发，按照深度优先的原则，从8个方向中尝试一个可以走的点，直到尝试过所有的方向，走完棋盘上的所有点，得出所有的解。

马走日问题可以看成是在层数为n\*n的8叉树中，找出所有的解。

## 3、代码实现

同样的，也可以把上面的算法框架，套用于马走日的身上。

递归算法：[dfs\HorseUsingRecursive.cpp](dfs/HorseUsingRecursive.cpp)

非递归算法：[dfs\HorseUsingStack.cpp](dfs/HorseUsingStack.cpp)

# 五、回溯法的根源

## 基本概念

深度优先搜索算法（Depth First Search，简称DFS）：一种用于遍历或搜索树或图的算法。 沿着树的深度遍历树的节点，尽可能深的搜索树的分支。当节点v的所在边都己被探寻过或者在搜寻时结点不满足条件，搜索将回溯到发现节点v的那条边的起始节点。整个进程反复进行直到所有节点都被访问为止。属于盲目搜索,最糟糕的情况算法时间复杂度为O(!n)。

## 基本步骤

1.搜索当前节点的所有相邻节点

2.1若有未访问的相邻节点，进入该节点，重复1操作

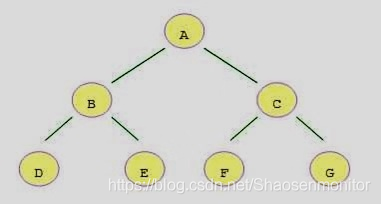
2.2若没有相邻节点，退回上一个节点

2.3若访问完所有相邻节点，退回上一节点

## 总的来说

DFS三个步骤  
1.判断是否终止条件  
2.搜索相邻节点，判断是否访问，没有则执行3操作  
3.进入该节点，重复2操作，直到所有相邻节点访问完

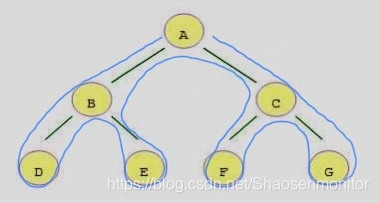
## DFS过程



模拟过程  
1.从根节点A开始搜索，发现有相邻节点B，进入B

2.进入B后，开始搜索，发现有相邻节点D，进入D  
3.进入D后，开始搜索，发现没有相邻节点，退回B  
4.回到B后，继续搜索，发现有相邻节点E，进入E  
5.进入E后，开始搜索，发现没有相邻节点，退回B  
6.回到B后，继续搜索，发现没有相邻节点，退回A  
7.回到A后，继续搜索，发现有相邻节点C，进入C

8.进入C后，开始搜索，发现有相邻节点F，进入F  
9.进入F后，开始搜索，发现没有相邻节点，退回C  
10.回到C后，继续搜索，发现有相邻节点G，进入G  
11.进入G后，开始搜索，发现没有相邻节点，退回C  
12.回到C后，继续搜索，发现没有相邻节点，退回A



相信模拟完过程之后，你应该了解DFS的过程了，就是从根节点开始走，走到底，例如从A到D的过程，就是一直走到底。然后再回头访问其他节点。

这是遍历图的一个算法，当然是遍历图，可以解决ACM的迷宫类问题，排列组合，八皇后等经典问题等等。

## 算法思想

回溯法（探索与回溯法）是一种选优搜索法，又称为试探法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。

## 基本模板1

1. int check(参数)
2. {
3. if(满足条件)
4. return 1;
5. return 0;
6. }
8. void dfs(int step)
9. {
10. 判断边界
11. {
12. 相应操作
13. }
14. 尝试每一种可能
15. {
16. 满足check条件
17. 标记
18. 继续下一步dfs(step+1)
19. 恢复初始状态（回溯的时候要用到）
20. }
21. }

## 基本模板2

1. void dfs(int x){
2. if(判断终止条件){
3. 根据题目条件处理
4. return;
5. }
6. if(判断是否不可能状态){
7. 减枝
8. return;
9. }
10. if(判断是否越界){
11. return;
12. }
13. for(搜索相邻节点){
14. if(没有访问过){
15. 根据题目条件处理
16. v[i]=1;*//标记访问过*
17. dfs(x+1);*//继续深搜*
18. v[i]=0;*//恢复标记位*
19. }
20. }
21. return;
22. }

## 问题示例

### 全排列问题

1. *//全排列问题*
2. #include<stdio.h>
3. #include<string.h>
5. int n;
6. char  a[15];
7. char re[15];
8. int vis[15];
9. *//假设有n个字符要排列，把他们依次放到n个箱子中*
10. *//先要检查箱子是否为空，手中还有什么字符，把他们放进并标记。*
11. *//放完一次要恢复初始状态，当到n+1个箱子时，一次排列已经结束*
12. void dfs(int step)
13. {
14. int i;
15. if(step==n+1)*//判断边界*
16. {
17. for(i=1;i<=n;i++)
18. printf("%c",re[i]);
19. printf("\n");
20. return ;
21. }
22. for(i=1;i<=n;i++)*//遍历每一种情况*
23. {
24. if(vis[i]==0)*//check满足*
25. {
26. re[step]=a[i];
27. vis[i]=1;*//标记*
28. dfs(step+1);*//继续搜索*
29. vis[i]=0;*//恢复初始状态*
30. }
31. }
32. return ;
33. }
35. int main(void)
36. {
37. int T;
38. scanf("%d",&T);
39. getchar();
40. while(T--)
41. {
42. memset(a,0,sizeof(a));
43. memset(vis,0,sizeof(vis));*//对存数据的数组分别初始化*
44. scanf("%s",a+1);
45. n=strlen(a+1);
46. dfs(1);*//从第一个箱子开始*
47. }
48. return 0;
49. }

### 2、素数圈问题

一个环由个圈组成，把自然数1，2，…，N分别放在每一个圆内，数字的在两个相邻圈之和应该是一个素数。 注意：第一圈数应始终为1。

input: N(0~20)

output:输出格式如下所示的样品。每一行表示在环中的一系列圆号码从1开始顺时针和按逆时针方向。编号的顺序必须满足上述要求。打印解决方案的字典顺序。

1. *//Prime Ring Problem*
2. *//与上面的全排列问题其实思路差不多，只是需要判断的条件比较多*
3. *//化大为小*
5. #include<stdio.h>
6. #include<string.h>
7. #include<stdlib.h>
9. int book[25];
10. int result[25];
11. int n;
12. int num;
13. *//判断是否为素数*
14. int prime(int n)
15. {
16. if(n<=1)
17. return 0;
18. int i;
19. for(i=2;i\*i<=n;i++)
20. {
21. if(n%i==0)
22. break;
23. }
24. if(i\*i>n)
25. return 1;
26. return 0;
27. }
28. *//判断是否能将当前的数字放到当前的圈内*
29. int check(int i,int step)
30. {
31. if((book[i]==0) && prime(i+result[step-1])==1)
32. {
33. if(step==n-1)
34. {
35. if(!prime(i+result[0]))
36. return 0;
37. }
38. return 1;
39. }
40. return 0;
41. }
43. void dfs(int step)
44. {
45. if(step==n)*//判断边界*
46. {
47. int a;
48. printf("%d",result[0]);
49. for(a=1;a<n;a++)
50. {
51. printf(" %d",result[a]);
52. }
53. printf("\n");
54. return ;
55. }
56. int i;
57. for(i=2;i<=n;i++)*//遍历每一种情况*
58. {
59. if(check(i,step))*//check是否满足*
60. {
61. book[i]=1;*//标记*
62. result[step]=i;*//记录结果*
63. dfs(step+1);*//继续往下搜索*
64. book[i]=0;*//恢复初始状态*
65. }
66. }
67. }
69. int main(void)
70. {
72. while(scanf("%d",&n)!=EOF)
73. {
74. num++;
75. memset(result,0,sizeof(result));
76. memset(book,0,sizeof(book));
77. result[0]=1;
78. printf("Case %d:\n",num);*//格式比较容易出错*
79. dfs(1);
80. printf("\n");
81. }
82. return 0;
83. }

### 3、油田问题

问题：GeoSurvComp地质调查公司负责探测地下石油储藏。 GeoSurvComp现在在一块矩形区域探测石油，并把这个大区域分成了很多小块。他们通过专业设备，来分析每个小块中是否蕴藏石油。如果这些蕴藏石油的小方格相邻，那么他们被认为是同一油藏的一部分。在这块矩形区域，可能有很多油藏。你的任务是确定有多少不同的油藏。

input: 输入可能有多个矩形区域（即可能有多组测试）。每个矩形区域的起始行包含m和n，表示行和列的数量，

1<=n,m<=100，如果m =0表示输入的结束，接下来是n行，每行m个字符。每个字符对应一个小方格，并且要么是’\*’，代表没有油，要么是’@’，表示有油。

output: 对于每一个矩形区域，输出油藏的数量。两个小方格是相邻的，当且仅当他们水平或者垂直或者对角线相邻（即8个方向）。

1. *//A - Oil Deposits*
2. #include<stdio.h>
3. #include<string.h>
4. #include<stdlib.h>
6. char a[105][105];
7. int n,m,result;
8. int dir[8][2]={{1,0},{-1,0},{0,1},{0,-1},{1,1},{-1,-1},{1,-1},{-1,1}};*//表示8个方向*
10. int check(int x,int y)*//检查是否有油田*
11. {
12. if(x>=0&&x<m&&y>=0&&y<n&&a[x][y]=='@')
13. return 1;
14. return 0;
15. }
17. int dfs(int x, int y)
18. {
19. int i,xx,yy;
20. if(check(x,y))
21. {
22. a[x][y]='.'; *//统计之后就可以把该油田标记，且不用恢复（要不会重复），*
23. *//也可以用一个数组来存每个点的访问情况，但是感觉没必要，浪费空间*
24. for(i=0;i<8;i++)
25. {
26. xx=x+dir[i][0];
27. yy=y+dir[i][1];
28. dfs(xx,yy);*//依次检查8个方向*
29. }
30. return 1;
31. }
32. return 0;
33. }
35. int main(void)
36. {
37. int i,j;
38. while(scanf("%d %d",&m,&n)==2)
39. {
40. if(m==0&&n==0)
41. break;
42. result = 0;
43. memset(a,0,sizeof(a));
44. for(i=0;i<m;i++)
45. scanf("%s",a[i]);
46. for(i=0;i<m;i++)*//在每一个点都搜索一次*
47. {
48. for(j=0;j<n;j++)
49. {
50. if(dfs(i,j))*//找到油田就可以将结果加1*
51. result++;
52. }
53. }
54. printf("%d\n",result);
55. }
56. return 0;
57. }

### 4、棋盘问题

问题：在一个给定形状的棋盘（形状可能是不规则的）上面摆放棋子，棋子没有区别。要求摆放时任意的两个棋子不能放在棋盘中的同一行或者同一列，请编程求解对于给定形状和大小的棋盘，摆放k个棋子的所有可行的摆放方案C。

input： 输入含有多组测试数据。 每组数据的第一行是两个正整数，n k，用一个空格隔开，表示了将在一个n\*n的矩阵内描述棋盘，以及摆放棋子的数目。 n <= 8 , k <= n 当为-1 -1时表示输入结束。 随后的n行描述了棋盘的形状：每行有n个字符，其中 # 表示棋盘区域， . 表示空白区域（数据保证不出现多余的空白行或者空白列）。

output：对于每一组数据，给出一行输出，输出摆放的方案数目C （数据保证C<2^31）。

1. #include<stdio.h>
2. #include<string.h>
3. #include<stdlib.h>
5. int n, k, ans;
6. char str[10][10];
7. int vis[100];
9. void dfs(int r, int k)
10. {
11. if(k==0)*//判断边界，此时棋子已经放完*
12. {
13. ans++;
14. return;
15. }
17. for(int i=r; i<n; i++)*//每次都从放过棋子下一行开始搜索，保证不重复*
18. {
19. for(int j=0; j<n; j++)
20. {
21. *//循环保证行不重复，check保证列不重复*
22. if(str[i][j]=='.' || vis[j]==1)
23. continue;*//不满足条件直接跳过*
24. vis[j] = 1;*//标记*
25. dfs(i+1, k-1);*//继续下一次标记*
26. vis[j] = 0;*//恢复初始状态*
27. }
28. }
29. }
31. int main(void)
32. {
33. while(1)
34. {
35. scanf("%d %d", &n, &k);
36. getchar();
37. if(n==-1 && k==-1)
38. break;
39. memset(str, '\0', sizeof(str));
40. memset(vis, 0, sizeof(vis));
41. ans = 0;
43. for(int i=0; i<n; i++)
44. {
45. for(int j=0; j<n; j++)
46. str[i][j] = getchar();
47. getchar();
48. }
50. dfs(0, k);*//从第0行开始放，此时手中还剩k个棋子*
51. printf("%d\n", ans);
52. }
53. return 0;
54. }