搜索入门之路

邓丝雨

2016 年 3 月 9 日

仅以此篇阐述我个人对搜索入门的一点看法，希望能对群巨以及新人们有所帮助!
理解得不到位的地方还望大家多多指点交流!
本文部分图文来自于申尚昆学长的课件，在此说明版权并表示感谢!
另外：本文和15年选修课本人的ppt部分配套……（其实是我懒，很多文字直接复制粘贴了）

目录

1 预备知识——递归和相关数据结构介绍

1.1 递归的理解

递归是什么？刘汝佳老师的书上说：递归的定义为——**参见递归**！这个说法乍一听很逗，但是请大家先记住它，一会儿你就会发现，这个说法还是蛮有道理的！

递归的标准定义（来自百度百科）：**程序调用自身的编程技巧**。

递归必须满足以下2个条件：一、会反复调用自身；二、一定能够跳出反复调用自身的过程，不会一直递归下去（有递归出口）。

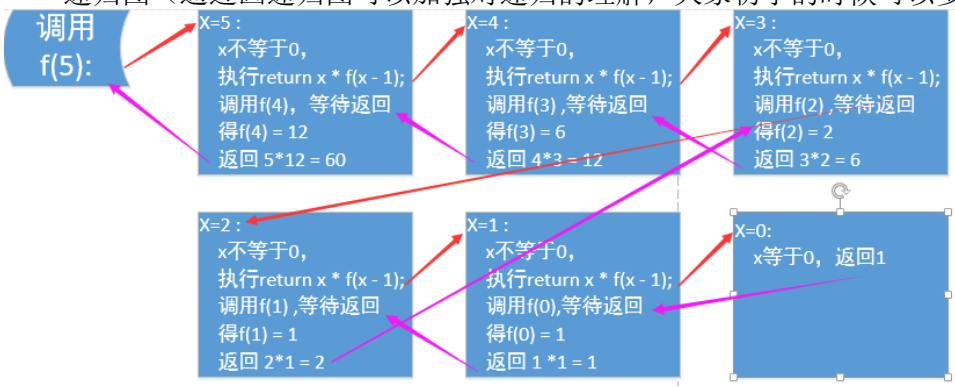
下面来看几个例子：

1.求n的阶乘n!

```
1        int f(int x)
2        {
3          if (x == 0) return 1;
4          else return x * f(x - 1);
5        }
```

在x不等于0的时候，会反复调用f(x-1)以减小规模，当x为0时则会终止递归一层一层得再返回回去。

递归图（通过画递归图可以加强对递归的理解，大家初学的时候可以多画）：



(红色箭头为调用，粉色箭头为返回)

2.求为斐波那契数列第n项! $f(1) = f(2) = 1, f(n) = f(n-1) + f(n-2)$;

```
1        int fib(int x)
2        {
3          if (x <= 2) return 1;
4          else return fib(x - 1) + fib(x - 2);
5        }
```

递归图略……

3.汉诺塔问题(Hanoi)

问题描述：古代有一个梵塔，塔内有三个座A、B、C，A座上有n个盘子，盘子大小不等，大的在下，小的在上。有一个和尚想把这n个盘子从A座移到B座，但每次只能允许移动一个盘子，并

且在移动过程中，3个座上的盘子始终保持大盘在下，小盘在上。在移动过程中可以利用B座，输出将n个盘子从A移到B的步骤!

思路分析：

我们可以将问题进行分解，把n个盘子从所在塔移动至目标塔可以分解成三个部分——将上面的n-1个盘子从所在塔移动至中转塔，将第n个盘子从所在塔移动到处目标塔，将n-1个盘子从中转塔移动至目标塔。

可以看到“将第n个盘子从所在塔移动到处目标塔”是一步到位的，而另外两个部分则是一个实际上和原问题相同的问题（三个塔是一样的，地位可以问题变换过程中互换），这样相同的问题是可以用递归解决的!

代码：

```
1   void hanoi(int n, int cur, int tmp, int target)
2   {
3     if (n == 1)
4     {
5       write(cur, target);
6       step++;
7     }
8     else
9     {
10      hanoi(n-1, cur, target, tmp);
11      write(cur, target);
12      ++step;
13      hanoi(n-1, tmp, cur, target);
14    }
15  }
```

几点说明：n个盘子，当前在cur塔上，中转塔为tmp，目标塔为target，step为移动步数，write函数用于输出移动。

如果你能理解这个问题，那么递归大家就理解了!

最后我想说明一下我对递归的定义为**“参见递归”**这一说法的理解，它至少说明了两点，第一是递归不停重复调用自身，第二是使用递归的问题必须具有相同子问题。但是他有一点很重要的问题没有说明，递归的有返回的!!

在知乎上看到一个说法：如果我们用开门求答案分别来说明递归和循环，那么循环是你拿着一把钥匙打开了第一扇门，然后看到第二扇门，手中的钥匙仍然能打开那扇门，一直这样直到打开最后一扇门，并在过程中找到答案，整个过程结束得出最终答案；递归则是拿着钥匙打开第一扇门，从第一扇门里面的人手上拿到新的钥匙打开第二扇门，一直到最后一扇门，拿到部分答案，再往回退，每进过一扇门，告诉相应的人，并得到相应对答案的加工，从而在退出的时候得到问题的全部答案。

蛮有意思的大家参考!

原帖地址：https://www.zhihu.com/question/20507130/answer/15551917

1.2 相关数据结构

1.2.1 ACM竞赛对于数据结构的要求——From 前ceo大人shen学长

在编写方面，C++的标准库提供了绝大部分的数据结构，诸如堆栈队列优先队列甚至是更高级的平衡二叉搜索树红黑树也有。

所以对于算法竞赛而言，要求是：

1、自己可以熟练编写简单的链表，堆栈，队列；

2、自己可以熟练地使用C++标准库中的现成的堆栈与队列；

3、对于一些更高级的数据结构，不要求能够随手就默写出来，但是要求对于原理和核心思想核心操作能够脱口而出；par 4、对于模板，必须保证自己能顾熟练地使用模板，而且对模板的每一个细节都要十分了解，并且要有能够在比赛时修改模板以适应题目变化的能力

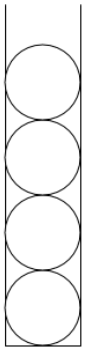
1.2.2 堆栈和队列

栈的定义：栈是限定仅在表头进行插入和删除操作的线性表（先进后出）

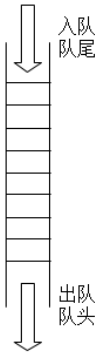
队列的定义：队列是一种特殊的线性表，特殊之处在于它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作，和栈一样，队列是一种操作受限制的线性表。进行插入操作的端称为队尾，进行删除操作的端称为队头。（先进先出）

队列和我们生活中排队的队列没有什么区别（当然，不能插队哈）……而堆栈呢，我喜欢把它比喻成一端封闭的球筒，筒的直径和球的直径差不多，所以球只能从一边放入也只能从一边拿出来。

示意图如下：



栈：



队列：

一般我们比赛的时候对于栈和队列有两种写法——数组模拟或者STL的写法。

数组模拟栈的写法：一个数组表示栈，一个int类型的变量为栈顶指针（这里的指针是“广义”的，只是表示下标的int类型变量）。注意c语言中区间一般默认是左闭右开，所以栈顶指针习惯性指向栈顶的上面一个位置，插入元素的时候先将元素放入指针指向的位置，然后指针上移，删除元素的时候指针下移。

队列的写法类似，但是注意队首队尾指针按左闭右开写时，都是先取值然后再移动指针（请大家自行思考）。

队列和栈的stl容器：

stack 入栈stack.push() 出栈stack.pop()

queue 入队queue.push() 出队queue.pop()

更多用法参考：http://www.cplusplus.com/

1.2.3 树的相关概念及起遍历

树是一种数据结构，它是由n($n \geq 1$)个有限节点组成一个具有层次关系的集合。把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。

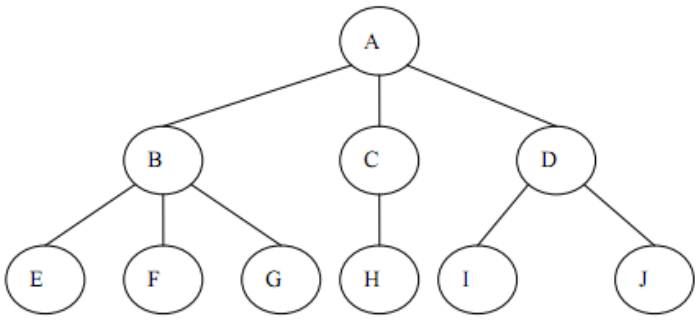
它具有以下的特点:

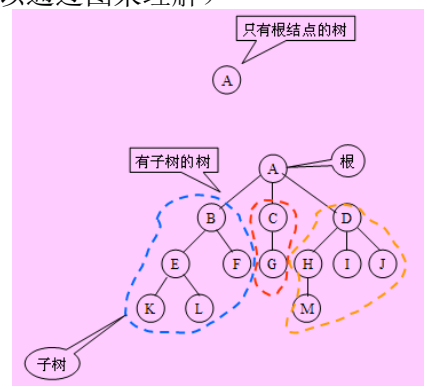
1.每个节点有零个或多个子节点

2.没有父节点的节点称为根节点

3.没有儿子节点的称为叶子节点

如图：（学习数据结构要多画图，很多定义和概念也可以通过图来理解）



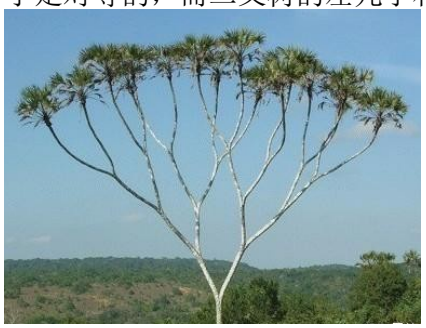


每一个非根节点有且只有一个父节点；除了根节点外，每个子节点可以分为多个不相交的子树

可以看到树是一种递归定义的结构（树的儿子还是树），所以树上的问题用递归来解决是得天独厚的!

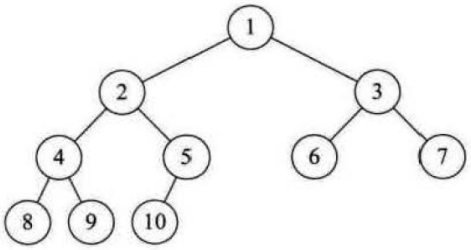
以下还有几个重要的概念需要介绍：

二叉树：每个节点最多含有两个子树的树称为二叉树，注意：一般的树任意一个节点的每个儿子是对等的，而二叉树的左儿子和右儿子是必须严格区分的。



满二叉树：除最后一层无任何子节点外，每一层上的所有结点都有两个子结点的二叉树。

完全二叉树：一颗满二叉树去掉最后一行最右边的若干个叶子节点，都是完全二叉树。（不同的教材有各种说法，但是意思都一样，大家结合图理解）



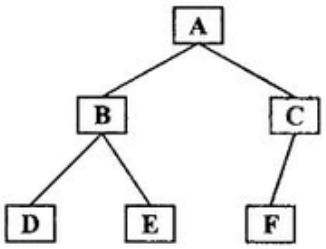
下面介绍二叉树的四种遍历：先序遍历、中序遍历、后序遍历和层序遍历!

先序遍历也叫做先根遍历、前序遍历，首先访问根结点然后遍历左子树，最后遍历右子树。在遍历左、右子树时，仍然先访问根结点，然后遍历左子树，最后遍历右子树，如果二叉树为空则返回。

中序遍历首先遍历左子树，然后访问根结点，最后遍历右子树。在遍历左、右子树时，仍然先遍历左子树，再访问根结点，最后遍历右子树。

后序遍历首先遍历左子树，然后遍历右子树，最后访问根结点，在遍历左、右子树时，仍然先遍历左子树，然后遍历右子树，最后遍历根结点。

层序遍历就是按二叉树的每一层的顺序来遍历，也就是先访问根，然后访问第一层，接着访问第二层。（从上到下，从左到右）



上图先序遍历结果：ABDECF

中序遍历结果：DBEAFC

后序遍历结果：DEBFCA

层序遍历结果：ABCDEF

你是不是还要问我这四种遍历代码咋写啊？

简单的给个提示：递归，将遍历以当前节点为根的子树分解为：访问当前点，遍历其左子树，遍历其右子树!

本节最后一个问题：已知先序中序求后序怎么做？

在先序中第一个是根，这样就可以在中序中找到根的位置，然后中序中左儿子在根的左边，右儿子在根的右边，从而区分出左子树和右子树，然后在左右子树中分别再重复如上判断即可，最终就能恢复出树的形状，从而得到后序遍历的结果。

Q：已知先序后序能求出中序么？已知后序中序呢？请大家思考!

2 搜索介绍

百度百科说：

“搜索算法是利用计算机的高性能来有目的的穷举一个问题解空间的部分或所有的可能情况，从而求出问题的解的一种方法。

搜索算法实际上是根据初始条件和扩展规则构造一棵‘解答树’并寻找符合目标状态的节点的过程。所有的搜索算法从最终的算法实现上来看，都可以划分成两个部分——控制结构（扩展节点的方式）和产生系统（扩展节点），而所有的算法优化和改进主要都是通过修改其控制结构来完成的。其实，在这样的思考过程中，我们已经不知不觉地将一个具体的问题抽象成了一个图论的模型——树，即搜索算法的使用第一步在于搜索树的建立。”

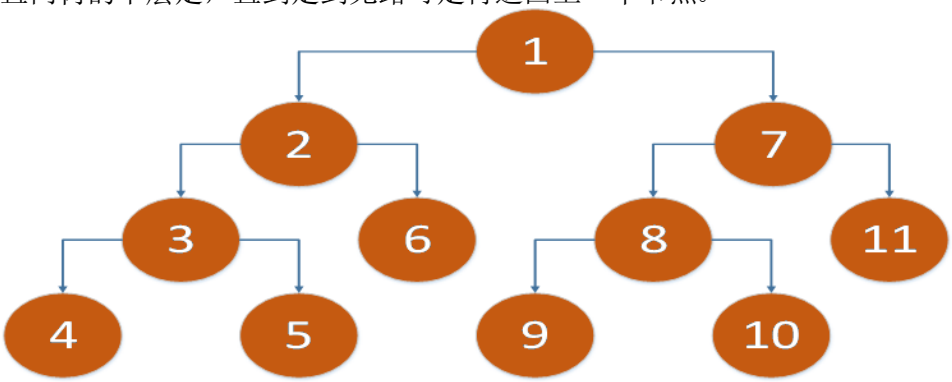
总的来说，搜索其实就是通过不停的试探去寻找解，而最坏情况下，搜索会把所有可能的办法全部穷举一遍!

一般说来，基础的方法有暴力的搜索法（最暴力的，比如直接多重循环），深搜，广搜三种。更高级的则有IDDFS，DBFS，$A\star$，$IDA\star$等等。

在这份入门指南里，我主要讲解一般深搜和广搜!

由于搜索出来的结构是一棵树，所以我们先来看树上的搜索!

深度优先搜索，顾名思义，是要优先往深的地方走的，通俗的来说就是“一条道走到黑!”一直向树的下层走，直到走到无路可走再返回上一个节点。



(图上数字为搜索访问的顺序)

算法过程：

VOID DFS(状态A)

判断当前的节点是否能走。合能走则继续执行，否则则回到上次调用（上一层）。

记录当前的决策Δ

向下走一层，也就是调用DFS(状态$A+\Delta$)

撤销当前的决策Δ

结束

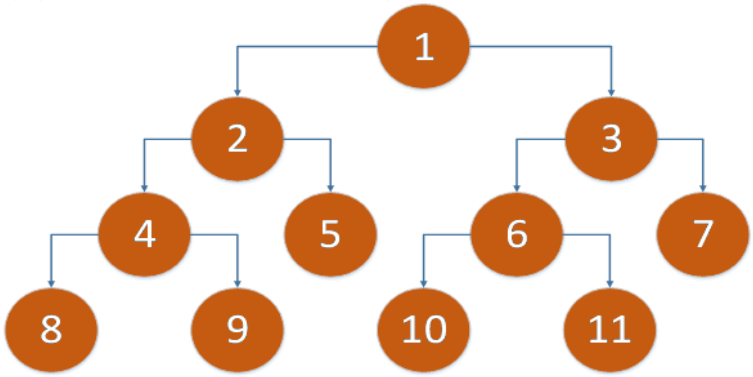
我们可以看出，深度优先搜索实际上是不断的试探和撤销回退的过程——如果可以往前走就往前走，如果不能，就撤销掉这一步退回上一步继续走。

在具体代码实现时，由于当前层的决策是要进行记录的，所以代码里会有一个非常明显的记

录决策部分，而在调用结束以后，为了向另外的子节点走，也就会有一个与之对应的撤销决策部分!（这两段代码非常对称，可以用于检测代码的正确性）

广度优先搜索，就是一层一层的走，下面问题来了，一层一层走代码要怎么样实现呢？

这里需要用到我们之前提到过的队列这种数据结构。广搜总是每次都把离上一状态最近的状态用一个队列记录下来；记录之后，检查队列是否为空，如果不为空，就讲队首元素弹出，并且以这个状态为“根节点”进行广度优先搜索，直到整个队列为空为止。



算法过程：

VOID BFS()

初始状态入队

循环开始

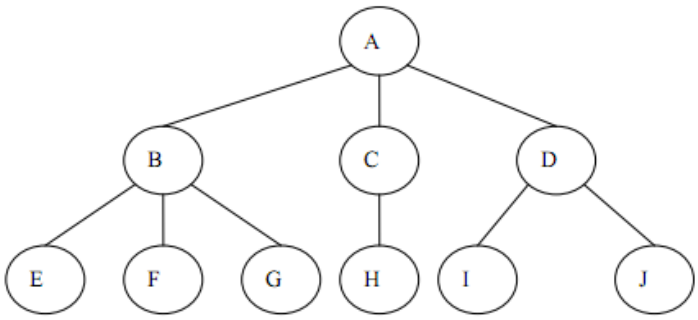
队首出队

队首的下一层节点入队（如果该节点存在并可以访问的话），同时进行标记访问等操作

队列为空时结束循环

结束

这样广搜就可以一层一层的进行下去了，可以看出广搜过程中，每个节点都只走了一次，不像深搜还有退回去的步骤!



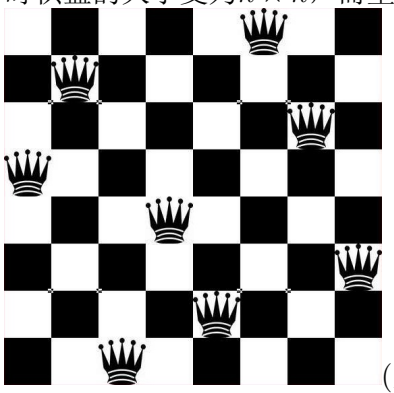
上图深度优先搜索的结果是：ABEFGCHDIJ

上图广度优先搜索的结果是：ABCDEFGHIJ

3 深度优先搜索解决问题——以八皇后为例

3.1 八皇后问题

在8 × 8格的国际象棋上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。八皇后问题可以推广为更一般的n皇后摆放问题：这时棋盘的大小变为$n \times n$，而皇后个数也变成n。当且仅当$n = 1$ 或$n \geq 4$ 时问题有解。


(八皇后问题的一个解)

3.2 解法

由于皇后的摆放位置不能通过某种公式来确定（但是方案数是有公式的其实），因此对于每个皇后的摆放位置都要进行试探和纠正，这就是“回溯”的思想。

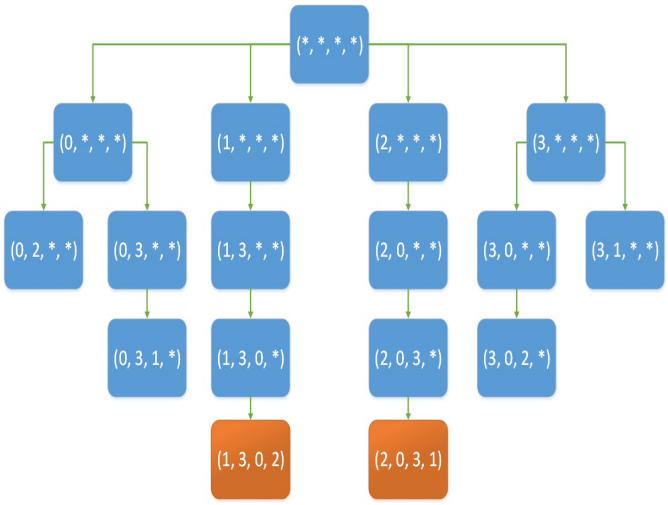
在N个皇后未放置完成前，摆放第i个皇后和第i+1个皇后的试探方法是相同的，因此完全可以采用递归的方法来处理。

由于皇后本身的特殊性质，即一行一列只能有一个皇后，所以我们所要做的就是，从第0行开始摆放，一直摆到第n –1行为止。

对于每一行皇后的摆放，大概是这样的步骤：枚举本行所有位置，如果可以放，1.放置皇后并标记该行该列该对角线被占领；2.放置下一行（递归）3.把这个皇后拿走（撤销占领标记）

下面我们以4皇后为例做一个说明

如果我们用四个数分别表示第0行第1行第2行第3行的放置状态的话，整个搜索的情况可以表示成如下图所示：



最开始的时候，每一行都没有放置皇后（1），然后在第0行的第0个位置放置皇后（2），在第一行的第2 个位置放置皇后（3），接下来发现第2行没法放置了，说明前面放错了，退回去，撤销掉（3）回到（2）的状态，在第一行的第3个放置皇后（4），在第二行的第1个位置放置皇后（5），第三行无法放置，撤销掉（5）回到（4），此时没有别的方案的，所以继续退回去，撤销掉（4）回到（2）……（以此类推，大家自行分析）

还有一个问题：怎么判断当前皇后可不可以放？或者说怎样记录皇后的放置情况？

Of course 你可以直接一个数组记下每个皇后放哪里，然后判断可不可以放的时候把前面的皇后都循环一遍就好了……但是如果这是最好的办法，我还会问这个问题么？

首先，我们是一行一行的放置皇后，所以不需要判断行冲突；

判断列冲突时，可以通过设置一个布尔数组，如果已经有皇后放在那里，就把布尔值设为FALSE，如果可以放置并且没有冲突，就放置当前这个皇后，且设置为TRUE；

判断对角线冲突时，有一个特殊的技巧：

由于每一条主对角线(x-y)的值是一定的，每一条副对角线(x+y)的值是一定的!!

所以就可以用(x + y)的值表示副对角线，(x - y)的值表示主对角线；

（于是就和处理列的情况一样了!）

（x-y有可能为负，所以要加上一个固定值，把它们变成正的，加多少我就不说了，这么简单的问题……）

假设我们把第cur个皇后放在了pos[cur]（pos[cur]储存了这个值），那么只需判断所检查的从前往后数第k 个皇后有没有冲突就行了!

可以看到，深度优先搜索就是试探然后不行就撤销，想明白这一点后，代码也是极好写的哦!

(代码暂时使用shen学长代码截图)(对，我就是懒，不想写一遍)

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MaxN = 50;
int n, tot, pos[MaxN];
bool vis[3][MaxN];

void dfs(int cur)
{
    int i, j;
    if (cur == n) tot++;
    else for (i = 0; i < n; i++)
    {
        if (!vis[0][i] && !vis[1][cur + i] && !vis[2][cur - i + n])
        {
            pos[cur] = i;
            vis[0][i] = vis[1][cur + i] = vis[2][cur - i + n] = 1;
            dfs(cur + 1);
            vis[0][i] = vis[1][cur + i] = vis[2][cur - i + n] = 0;
        }
    }
}

int main()
{
    #define QUERY(X) { \
                n = X, tot = 0; \
                memset(vis, 0, sizeof(vis)); \
                dfs(0); cout << tot << endl; \
                }
    QUERY(8);
    return 0;
}
```

3.3 深搜小结

深度优先搜索通过不停的试探和撤销将所有的可能解都遍历到，然后得出想要的结果，满足相同子问题性质（问题和问题的子问题是一样的结构，比如，在放完前i个皇后之后放置第i+1个皇后和在放完前i+1个皇后后放置第i+2个皇后是用的一样的办法）（只有满足相同子问题性质才能用递归啊!!）

深搜代码一般的结构在上面介绍深搜的时候就说过了，再更详细的写一遍（按理来说是不能这样讲的，这只是给初学者一个引导，但请不要迷信它）

VOID DFS(状态A)

判断是否已经求出全部结果，是的话，输出方案或者统计方案数等操作（按题目定）

判断当前的节点是否能走。能走则继续执行，否则则回到上次调用（上一层）。

枚举当前层的决策（循环）

{

可能会有的一个if，判断这个决策是否可行

{

记录当前的决策Δ

向下走一层，也就是调用DFS(状态$A+\Delta$)

撤销当前的决策Δ

}

}

结束

4 宽度优先搜索解决问题——以走迷宫为例

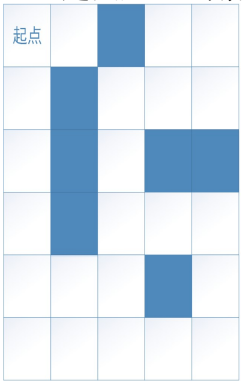
4.1 走迷宫问题

设有一个n*m方格的迷宫，即n行m列的迷宫。

迷宫格子中分别放有0和1，1表示可通，0表示不能，迷宫走的规则为：

从某点开始，有四个方向可走（上下左右），前进方格中数字为1时表示可通过，为0时表示不可通过，要另找路径。

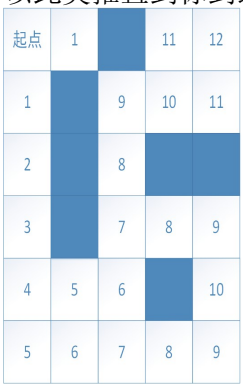
起点是左上角的点，输入终点坐标，给出最短的路径



4.2 解法

如果让你手工算最短路你会怎么算？如果地图比较大的话怎样才能又快又好的得到结果呢？

首先把一步能到的格子标成1，然后把两步能到的格子标成2，然后把三步能到的格子标成3，以此类推直到标到终点（或者没有新的点可以标）为止……



让代码做这件事也是这样的！可是怎么写呢？

——建立一个等待处理的节点的队列。

先把一步以内能够走到的节点加进来。

然后对这个队列的元素进行处理：即从队列中删除队首节点A，然后再把这个节点A的能够一步走到的节点加入队列。

直到这个队列空为止。

这就是广度优先搜索啦!!!

一个小技巧：

我们注意到每一个点是一个坐标(x, y)，我们希望能够用一个数来表示这个数对（为了方便），如果所有的编号都从0开始的话，第x行y列的点编号是i*m+j（m为列数），反之，编号为u的节点，其行号和列号分别为u/m，u%m。

对于这个题还有一个问题，方案怎么记录??

大家要习惯递归的思维方式——对一个节点x，记录他的父亲节点就行了！然后通过不断的找父亲就可以将整个路径找出来！输出的时候也递归输出！代码大概如下：

```
1  void print_path(int x, int y)
2  {
3    int fx = fa[x][y] / m, fy = fa[x][y] % m;
4    if (fx != x || fy != y)
5    {
6      print_path(fx, fy);
7      printf("(%d,%d)", fx, fy);
8    }
9  }
```

至于广搜咋个写嘛，这个问题大家先自己想想……希望你们能够在看广搜总结之前把这个代码写出来!!

5 在解空间里进行状态转移，直到到达目标状态——我对搜索的一点理解

6 状压只是一种手段——以hdu 5025拯救唐僧为例具体理解搜索是在进行状态转移

7 搜索剪枝

7.1 奇偶剪枝：hdu 1010

7.2 其他题目要求有关的剪枝：hdu 2437

7.3 经典的小木棍问题：poj1011