ACM算法与程序设计
（三）
搜索入门

杜育根

ygdu@sei.ecnu.edu.cn

搜索入门

- 在这一章里，我们将介绍搜索算法基础，包括深度优先搜索和广度优先搜索。对于数据规模较小的一类问题，搜索算法是解决它们非常有用的工具。现在，让我们一起搜索吧！

递归

- 递归是计算机编程中应用最广泛的一个技巧，也是比较难理解的一个技巧，所以我们打算在这章花大量的时间来理解递归。所谓递归，就是函数调用函数自身 ，一个过程或者函数在其定义或说明中有直接或者间接调用自身都叫递归。而递归一般都用来解决有重复子问题的问题。
- 我们先来理解直接递归，间接递归非常复杂，用的比较少。下面通过求解f(n)=n!（！代表阶乘）的问题来理解直接递归。递归的一个难点——边界条件。所谓边界条件，就是什么情况下，函数不应该再继续调用自身了。那么很显然，按照阶乘的定义factorial函数对应的边界条件是n == 1，如果n == 1，函数应该立即返回 1。
- 1 int factorial(int n) {
- 2 if (n == 1) {
- 3 return 1;
- 4 }
- 5 return n * factorial(n - 1);
- 6 }

Fibonacci 数列

- Fibonacci 数列的定义是：

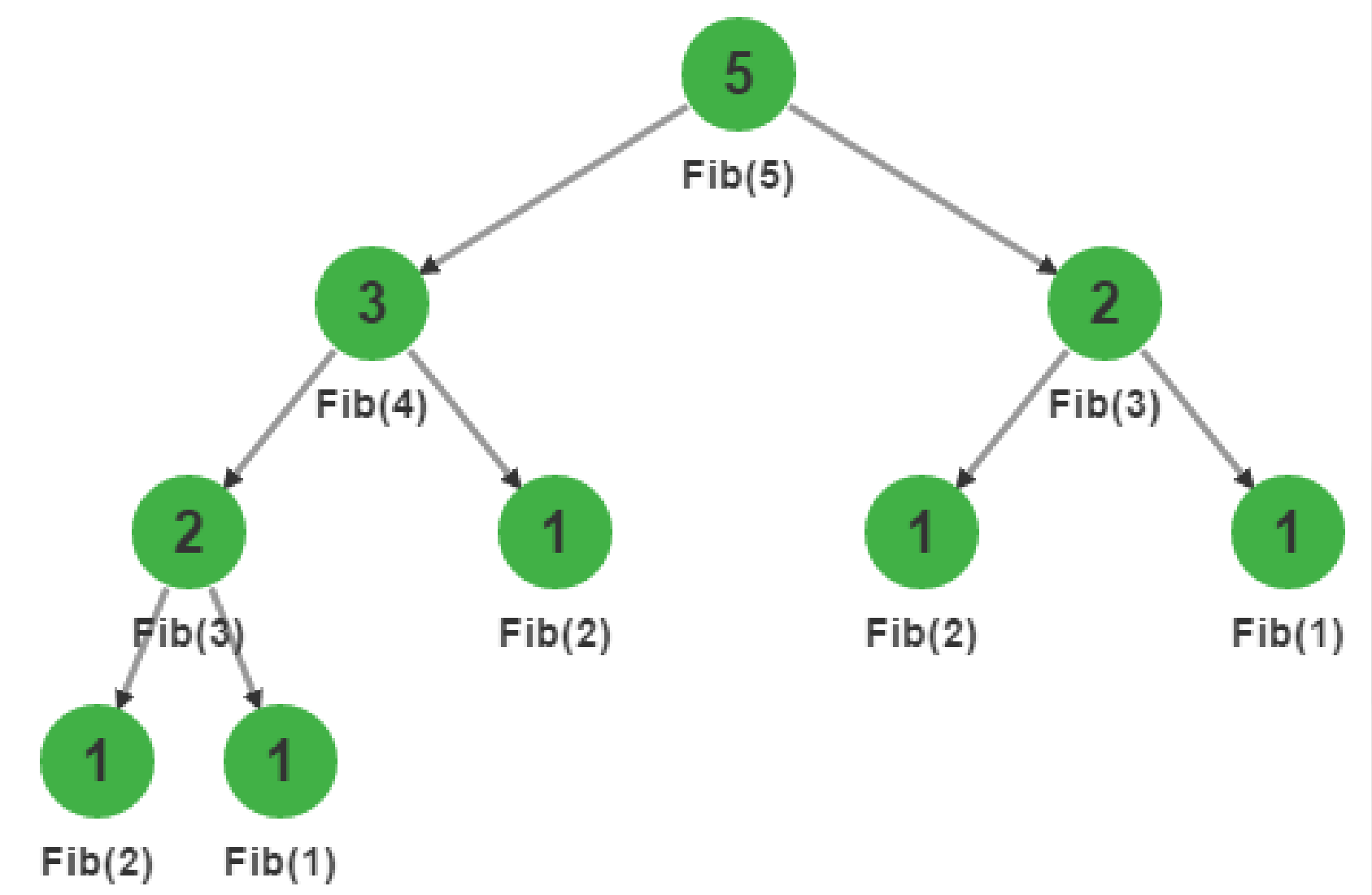
$$f(n) = \begin{cases} 1 & n = 1 \\ 1 & n = 2 \\ f(n-1) + f(n-2) & n > 2 \end{cases}$$

- 如果用递归写 Fibonacci 数列，按照定义可以很简单地写出来。

```
1 int fib(int n) {
2     if (n == 1 || n == 2) {
3         return 1;
4     }
5     return fib(n - 1) + fib(n - 2);
6 }
```

- 下一节我们将用动画来演示fib(5)递归调用的过程。

fib(5) 的演示



习题：小明吃桃

- 小明买了一堆桃子不知道个数，第一天吃了一半的桃子，还不过瘾，有多吃了一个。以后他每天吃剩下的桃子的一半还多一个，到 n 天只剩下一个桃子了。小明想知道一开始买了多少桃子。
- 输入格式
- 输入一个整数 n(2≤n≤60)，代表第 n只剩了一个桃子。
- 输出格式
- 输出买的桃子的数量。
- 样例输入1
- 2
- 样例输出1
- 4
- 样例输入2
- 3
- 样例输出2
- 10

习题：汉诺塔1

- 汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着 64 片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。
- 现在小明开始玩汉诺塔游戏， 他放了 n 片黄金圆盘在第一根柱子上，从上到下依次编号为 1,2,...,n，1 号圆盘最小，n 号圆盘最大。现在小明想把圆盘全部移动到第2 根柱子上，移动过程中小明必须准守游戏规则。 现在小明想知道他完成游戏的每一步和总步数。
- 输入格式：输入一个正整数n(1≤n≤10) 表示黄金圆盘的个数
- 输出格式：输出每一步的移动方向以及总移动步数，见样例格式。
- 样例输入
- 3
- 样例输出
- 第1步:将1号盘子A---->C
- 第2步:将2号盘子A---->B
- 第3步:将1号盘子C---->B
- 第4步:将3号盘子A---->C
- 第5步:将1号盘子B---->A
- 第6步:将2号盘子B---->C
- 第7步:将1号盘子A---->C
- 最后总的步数为7

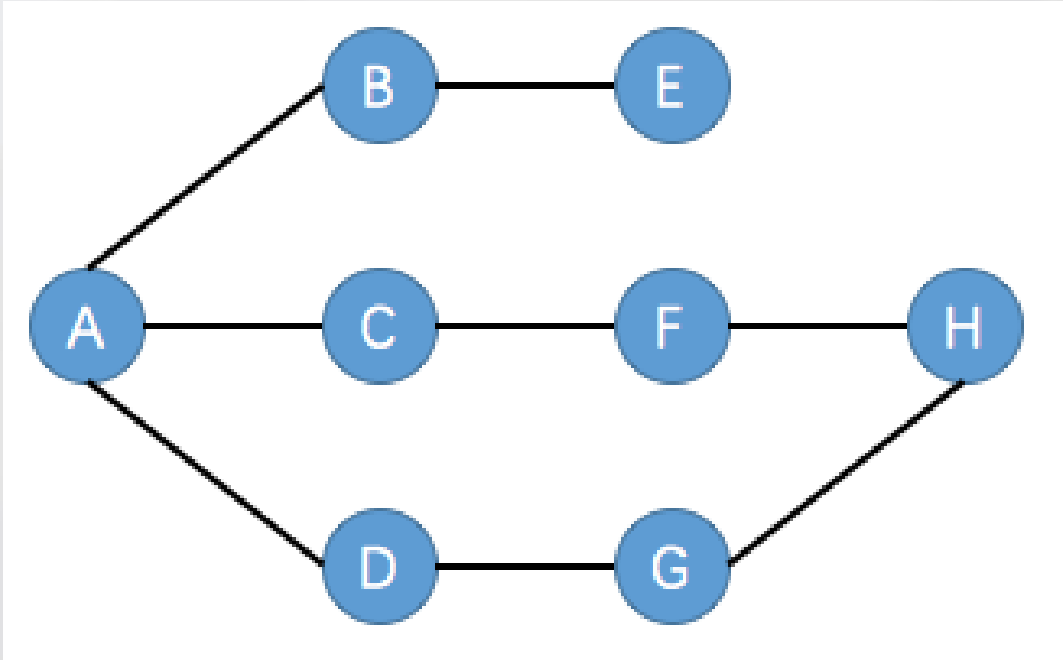
习题：汉诺塔2

- 汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着 64 片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。
- 现在小明开始玩汉诺塔游戏， 他放了 n 片黄金圆盘在第一根柱子上，从上到下依次编号为 1,2,...,n， 1 号圆盘最小，n 号圆盘最大。小明移动第 i 号圆盘的时候需要花费 i 点体力。现在小明想把圆盘全部移动到第2 根柱子上，移动过程中小明必须准守游戏规则。
- 现在小明想知道他完成游戏的最小移动次数和最少消耗的体力。。
- 输入格式
- 输入一个正整数n(1≤n≤60) 表示黄金圆盘的个数
- 输出格式
- 一行输出 2 个数，表示最小移动次数和最小消耗的体力，中间用一个空格隔开。
- 样例输入
- 3
- 样例输出
- 7 11

深度优先搜索

- 深度优先搜索（depth-first-search），简称 dfs，应该算是应用最广泛的搜索算法，属于图算法的一种。dfs 按照深度优先的方式搜索，通俗的说就是“一条路走到黑”。dfs 是一种穷举的手段，实际上就是把所有的可行方案列举出来，不断去试探，直到找到问题的解，其过程是对每一个可能的分支路径深入到不能再深入为止，而且每个顶点只能访问一次。

举例说明：右图是一个无向图，如果我们从 A点开始深度优先搜索（以下的访问次序并不是唯一的，第二个点既可以是 B也可以是C、D），则我们可能得到如下的一个访问过程：A->B->E，回溯到A，继续访问C->F->H->G->D，回溯到A，A此时已经没有未访问的相邻顶点，本次搜索结束。最终通过 A->B-> E->C->F->H->G->D的顺序搜索了图中的所有顶点。



dfs 一般借助递归来实现：

```
1 void dfs(int deep) {
2     if (到达边界) {
3       // 做一些处理后返回
4     } else {
5         for(所有可能的选择) {
6             dfs(deep + 1);
7         }
8     }
9 }
```

- 上面这段代码是 dfs 的一般写法。
- dfs 和递归的区别是，dfs 是一种算法，注重的是思想，而递归是一种基于编程语言的实现方式。我们可以通过递归来实现 dfs。
- 上一节我们讲到的那几个递归的例子里，每个顶点最多只有两个分支，而在接下来的 dfs 中，每个顶点可以扩展出很多个分支。

走迷宫

- 下面我们通过一个实际问题来理解 dfs 到底是怎么一回事。 相信大家都玩过走迷宫。用二维数组来表示一个迷宫：

1	S	#	#	.
2	.	.	.	.
3	#	#	#	T

- 其中'S'表示起点，'T'表示终点，'#'表示墙壁，'.'表示平地。你需要从'S'出发走到'T'，每次只能向上下左右相邻的点移动，并且不能走出地图，也不能走进墙壁，每个点只能通过一次。求一共有多少种走的方案。
- 用 dfs 来求解这个问题。先找到起点，每个点按照右、上、左、下的顺序尝试，从起点‘S’开始，走到下一个点以后，把这个点再当做起点‘S’继续按照顺序尝试，如果某个点上下左右能走的点都尝试走过了以后，便把起点回到走到这个点之前的点。继续尝试其他方向。直到所有点都尝试走了上下左右。如果这条路不行，就回头，尝试下一条路，我们来讨论通过程序来完成这个过程。

走迷宫代码框架

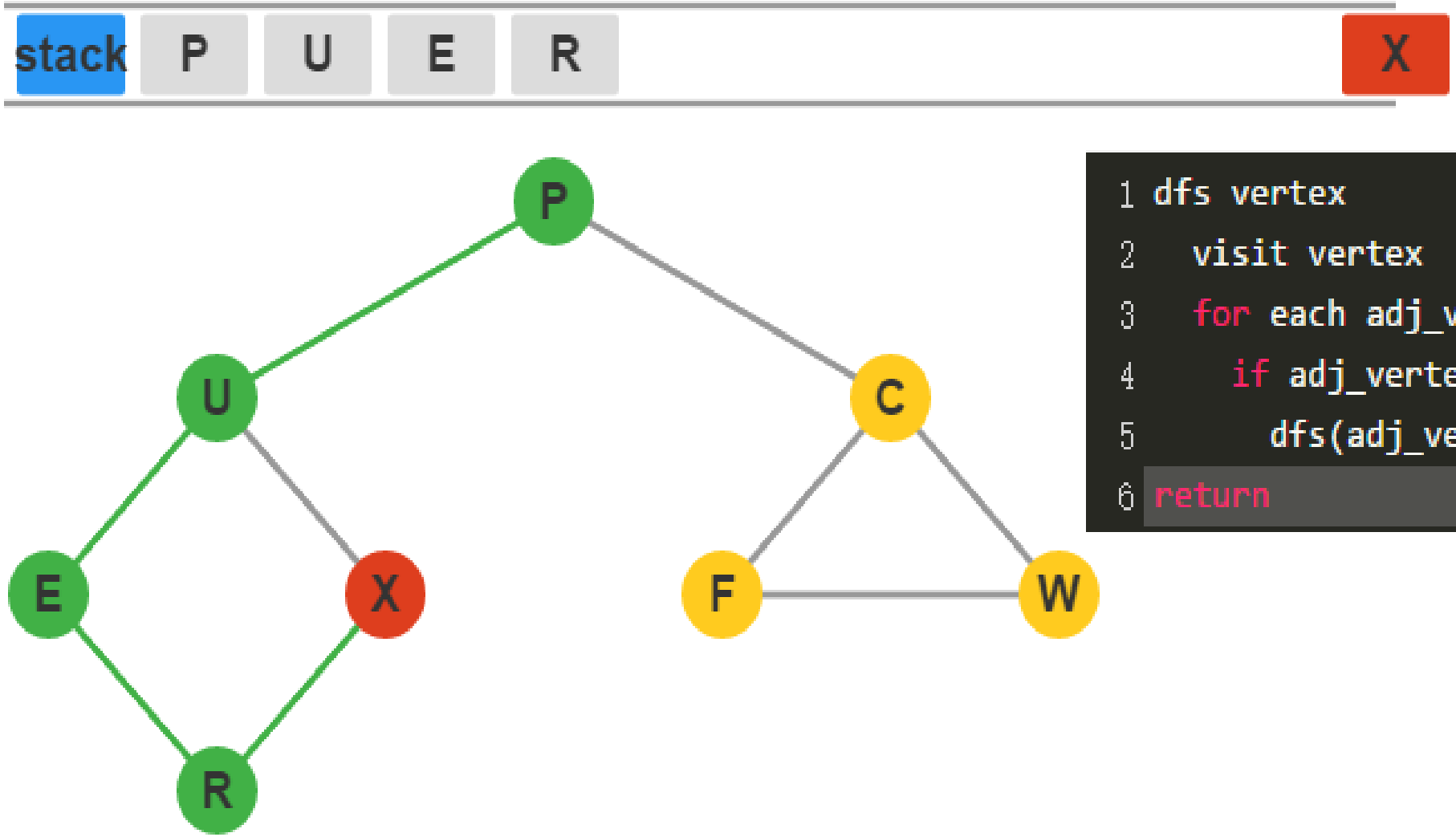
下面给出了代码的框架。

```
1 // 对坐标为(x, y)的点进行搜索
2 void dfs(int x, int y) {
3     if (x,y) 是终点 {
4         方案数增加
5         return;
6     }
7     标记(x, y)已经访问
8     for (x, y) 能到达的格子(tx, ty) {
9         if (tx, ty) 没有访问 {
10            dfs(tx, ty);
11        }
12    }
13    取消(x, y)访问标记
14 }
```

对于这道题目，我们可以通过如下的方式，来根据格子 (x, y) 求出可以到达的格子 (tx, ty)。

```
1 int x, y;
2 int xx[4] = {1, 0, -1, 0}; //横向位移
3 int yy[4] = {0, 1, 0, -1}; //纵向位移
4 for (int i = 0; i < 4; ++i) {
5    int tx = x + xx[i]; //计算到达的点横坐标
6    int ty = y + yy[i]; //计算到达的点纵坐标
7     //做一些处理
8  }
```

深度优先搜索的演示




```
1 dfs vertex
2   visit vertex
3   for each adj_vertex ∈ adj[vertex]
4     if adj_vertex has not visited
5       dfs(adj_vertex)
6 return
```

习题：走迷宫

- 给一个n行m列的2维的迷宫，'S'表示迷宫额起点，'T'表示迷宫的终点，'#'表示不能通过的点，'.' 表示可以通过的点。你需要从'S'出发走到'T'，每次只能上下左右走动，并且只能进入能通过的点，每个点只能通过一次。现在要求你求出有多少种通过迷宫的的方案。
- 输入格式: 第一行输入 n, m(1≤n,m≤10) 表示迷宫大小，接下来输入n行字符串表示迷宫
- 输出格式：输入通过迷宫的方法数
- 样例输入1
- 2 3
- S.#
- ..T
- 样例输出1
- 2
- 样例输入2
- 3 3
- S..
- .#.
- ..T
- 样例输出2
- 2

习题：踏青

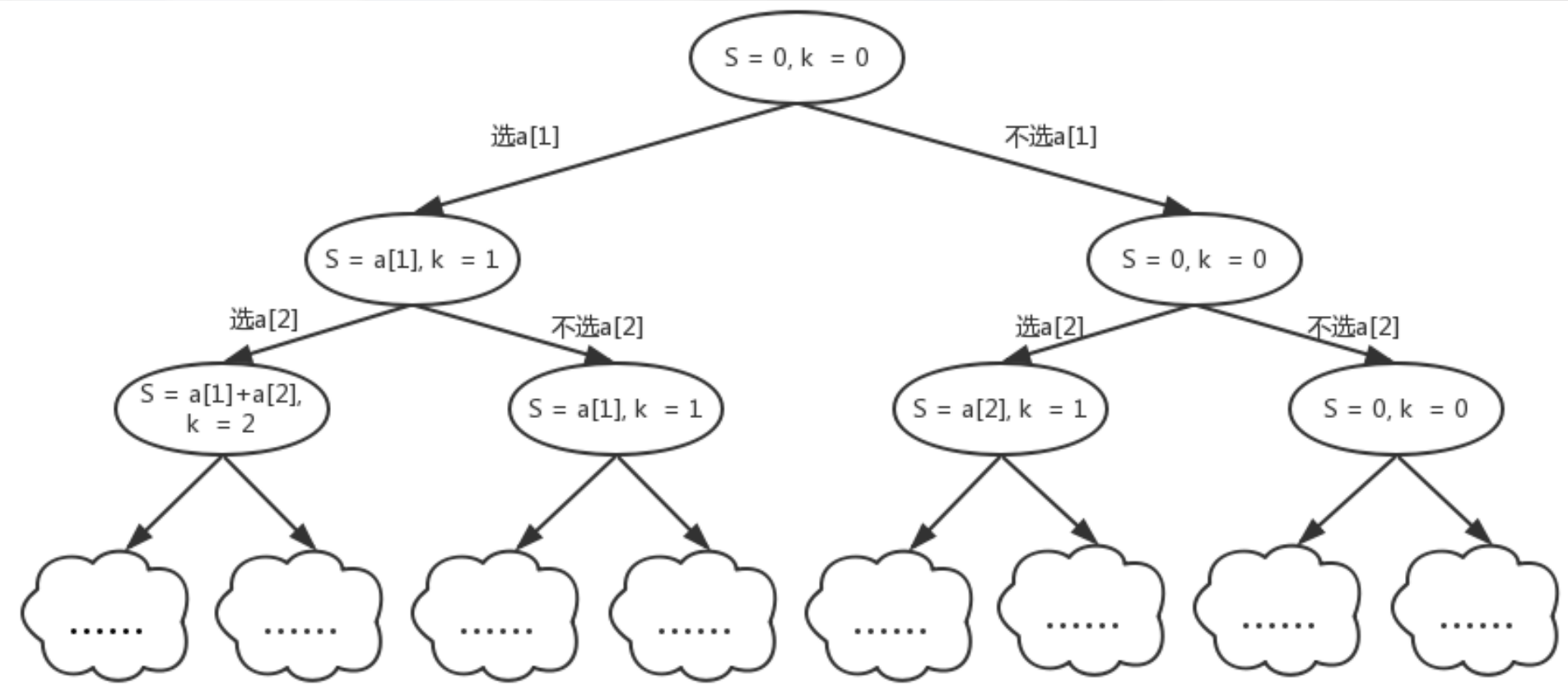
- 小明和他的朋友周末相约去召唤师峡谷踏青。他们发现召唤师峡谷的地图是由一块一块格子组成的，有的格子上是草丛，有的是空地。草丛通过上下左右 4个方向扩展其他草丛形成一片草地，任何一片草地中的格子都是草丛，并且所有格子之间都能通过上下左右连通。如果用'#'代表草丛，'.'代表空地，下面的峡谷中有 2片草地。
- 1 ##..
- 2 ..##
- 处在同一个草地的 2个人可以相互看到，空地看不到草地里面的人。他们发现有一个朋友不见了，现在需要分头去找，每个人负责一片草地，小明想知道他们至少需要多少人。
- 输入格式：第一行输入n，m(1≤n,m≤100) 表示峡谷大小。接下来输入 nn 行字符串表示峡谷的地形
- 输出格式：输出至少需要多少人
- 样例输入　　样例输出
- 5 6　　5
- .#....
- ..#...
- ..#..#
- ...##.
- .#....

抽象形式的dfs和剪枝

- 前面用到的 dfs 算法都是比较容易想象出搜索过程的，接下来我们看看一些需要抽象成 dfs 的问题。实际上，我们所遇到的大多数问题都是需要抽象成 dfs 的形式才能应用 dfs 进行搜索的。
- 例题：给出n个整数，要求从里面选出K个整数，使的选出来的数的和为 SUM。枚举每一个数选或者不选，我们可以用dfs来抽象这样的枚举过程。我们在搜索的过程中用S来记录当前选择的数值总和，k来记录选择的数的个数，在第一层 dfs 的时候，我们可以枚举是否选第一个数，如果选第一个数则让S加上第一个数且 k加一，dfs 进入到下一层，否则 dfs 直接进入到下一层，在第二层，对第二个数做同样的处理，dfs 的过程中记录已经选取的数的个数，如果已经选取了 k个数，判断 S值是否是SUM。对于每一层，我们都有两个选择，选和不选。不同的选择，都会使得搜索进入两个完全不同的分支继续搜索。

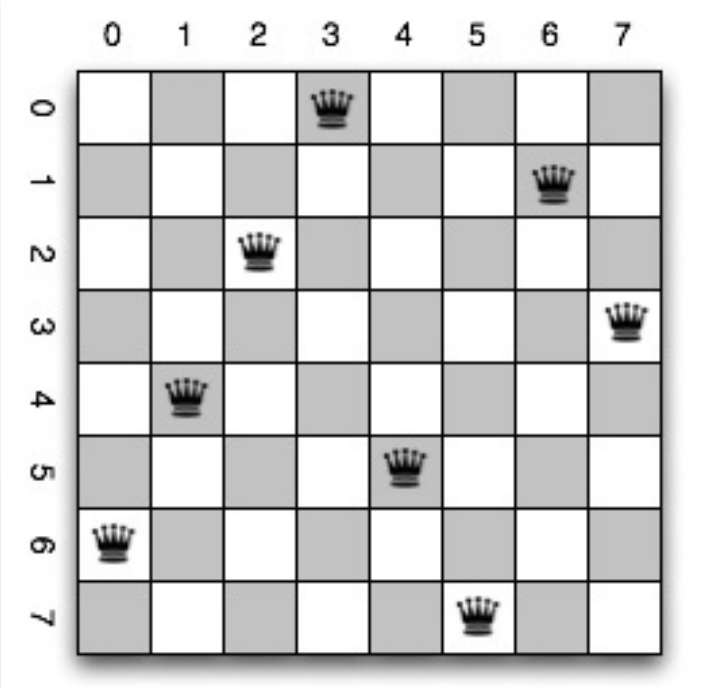
dfs 搜索树例子

- 图片显示的是这个例题的 dfs 搜索树例子，搜索树中的每个结点代表一个状态。具体地，通过记录当前的 S和 k值来表示我们搜索到的状态，初始状态是S=0,k=0。对于不同的决策，会在搜索树上对应的分支进行跳转。



例题：N皇后问题

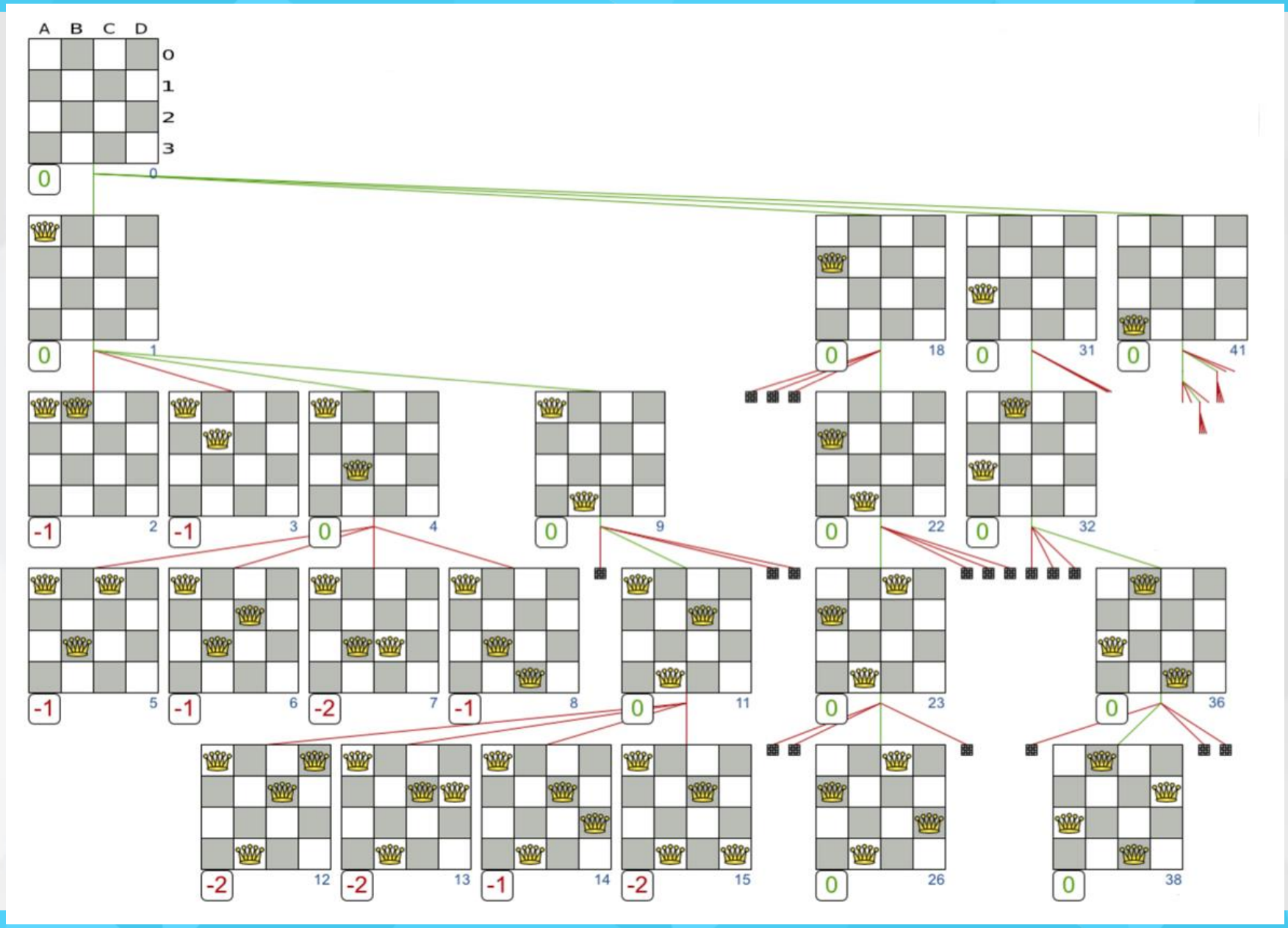
- N皇后问题是一个经典的问题，在一个 N×N 的棋盘上放置 N个皇后，每行刚好放置一个并使其不能互相攻击（同一行、同一列、同一斜线上的皇后都会自动攻击）
- 下图就是一个合法的 八皇后的解。N皇后问题是指：计算一共有多少个 N皇后的解。



解析

- 这是一个经典的 dfs 的应用。在搜索时，一共有四个状态需要记录：当前搜索到第几行、每列的占用情况、每个左上-右下对角线的占用情况、每个右上-左下对角线的占用情况。
- 下图是 N=4时搜索树的局部形态：
- 图中每个状态下的数如-2,-1,0 表示的是该状态的冲突次数（在多少列或斜线发生冲突），只有当冲突次数为 0时才是合法状态。 由于我们是逐行搜索的，所以可以确保每一行至多有一个皇后，因此在搜索过程中只需要确保每列、两个方向的所有对角线上都至多有一个皇后就可以了。直到最后一行的皇后也放置完毕，则说明找到了一个合法解，用一个计数器来累加总方案个数即可。

N=4时搜索树的局部形态



C++ 代码如下：

```
int n;  // 棋盘大小
int ans = 0;  // 解的个数
bool a[20];  // 列占用情况，若第 i 列被占用，则 a[i] = true，否则为 false
bool x1[20];  // 左下-右上 对角线的占用情况
bool y1[20];  // 左上-右下 对角线的占用情况
void dfs(int deep) {  // deep 表示当前搜索到第几行
    if (deep >= n) {  // 搜索完最后一行，说明找到了一组合法解
        ans++;
        return;  // 回溯
    }
    for (int i = 0; i < n; i++) {  // 枚举当前行的皇后放置到第 i 列
        if (x1[i + deep] == false && y1[i - deep + n] == false && a[i] == false) {
                // 判断该皇后是否与已放置的皇后发生冲突
            // 放置皇后 (deep, i)，一共需要修改三个标记数组
            x1[deep + i] = true;
            y1[i - deep + n] = true;
            a[i] = true;
            dfs(deep + 1);  // 当前行枚举完毕，搜索下一行
            // 恢复放置皇后 (deep, i) 前的状态
            a[i] = false;
            x1[deep + i]=false;
            y1[i - deep + n] = false;
        }
    }
}
```

```
int main() {
    scanf("%d", &n);
    memset(a, false, sizeof(a));
    memset(x1, false, sizeof(x1));
    memset(y1, false, sizeof(y1));
    dfs(0);
    printf("%d", ans);
    return 0;
}
```

剪枝

- 我们了解到，搜索是从起点出发，遍历整张图。而搜索的顺序，就对应着一棵搜索树。 剪枝，顾名思义，就是通过一些判断，砍掉搜索树上不必要的子树。这些子树可能是不可达的，也可能是可达但显然不是最优的，去掉它们对最终答案的求解没有影响，所以我们称为“剪枝”。
- 把常用的剪枝分成以下两类：
- 1.可行性剪枝。
- 2.最优性剪枝。

Betsy 的旅行

- 一个正方形的小镇被分成 N^2个小方格，Betsy 要从左上角的方格到达左下角的方格，并且经过每个方格恰好一次。对于给定的 N，计算出 Betsy 能采用的所有的旅行路线的数目。
- 可以用 DFS 来解决这个问题：Betsy 从左上角出发，每一步可以从一个格子移动到相邻且没有经过的格子中，遇到死胡同则回溯，当移动了 N^2-1步并达到左下角时，即得到了一条新的路径，继续回溯搜索，直至枚举完所有可行的道路。
- 这个解法的时间复杂度极高，对N=6 的情况都无法很快地解决，所以，必须要借助剪枝来优化。

可行性剪枝

- 一般是通过某种计算，判断出当前状态能否到达目标状态，如果不能到达目标状态则退出。
- 首先从“必要条件”（即合法的解所应当具备的特征）的角度分析：
- 1. 对于一条合法的路径，除出发点和目标格子外，每一个中间格子都必然有“一进一出”的过程。所以在搜索过程中，必须保证每个尚未经过的格子都与至少两个尚未经过的格子相邻（除非当时 Betsy 就在它旁边）。
- 2. 在一个合法的移动方案的任何时刻，都不可能有孤立的区域（区域内所有点都无法到达目标格子）存在。虽然孤立区域中的每一个格子也可能都有至少两个相邻的空的格子，但它们作为一个整体，Betsy 已经不能达到。如果发现这种情况。
- 我们在搜索的同时维护状态。
- 对于第一条剪枝策略，可以设一个整型标记数组，分别保存与每个格子相邻的没被经过的格子的数目，Betsy 每移动到一个新位置，都只会使与之相邻的至多 4 个格子的标记值发生变化，只要检查它们的标记值即可。
- 对于第二条剪枝策略，处理就稍稍麻烦一些。但我们仍然可以使用局部分析的方法，即只通过对 Betsy 附近的格子进行判断，就确定是否应当剪枝。

最优性剪枝

- 定义一个抽象估价函数（对于具体的问题，通常是当前状态，加上尽可能多的改变量）。
- 当我们处在搜索树的一个子树的根结点时，可以通过一些方法来估算该子树所有解的估价函数的上界。如果最终解的上界没有当前答案优，则该子树不用搜索下去了，省去了大量冗余的计算，这就是“最优性剪枝”。

习题：买书

- 小明去书店买书，他有 m 元钱，书店里面有 n 本书，每本书的价格为 pi 元。小明很爱学习，想把身上钱都用来买书，并且刚好买 k 本书。请帮小明计算他是否能刚好用 m 元买 k 本书。
- 输入格式：第一行输入 3 个整数 m(1≤m≤100000000)，n(1≤n≤30)，k(1≤k≤min(8,n))。接下来一行输入 n 个整数，表示每本书的价格 pi(1≤pi≤100000000)。
- 输出格式：如果小明能刚好用 m 元买 k 本书，输入一行"Yes"，否则输出"No"。
- 样例输入1
- 10 4 4
- 1 2 3 4
- 样例输出1
- Yes
- 样例输入2
- 10 4 3
- 1 2 3 4
- 样例输出2
- No

习题：方程的解数

- 小明在求解一个 n元的高次方程：

$$k_1 x_1^{p1} + k_2 x_2^{p2} + \ldots + k_n x_n^{pn} = 0$$

- 其中：x1,x2,...,xn 是未知数，k1,k2,...,kn 是系数，p1,p2,...,pn 是指数。方程中所有数都一定是整数。
- 假设未知数1≤xi≤M,i=1...n。你能帮小明算出这个方程的整数解个数吗?
- 输入格式：第一行输入一个整数 n(1≤n≤4)。第二行输入一个整数 M(1≤M≤150)。第3行到第n+2 行，每行输入两个整数，分别表示ki(|ki|≤20) 和pi(1≤pi≤4)。两个整数之间用一个空格隔开。
- 输出格式：输出一行，输出一个整数，表示方程的整数解的个数。

样例输入	样例输出
3 100 1 2 -1 2 1 2	104

习题：等边三角形

- 小明手上有一些小木棍，它们长短不一，小明想用这些木棍拼出一个等边三角形，并且每根木棍都要用到。 例如，小明手上有长度为 1，2，3，3 的4根木棍，他可以让长度为1，2 的木棍组成一条边，另外 2根分别组成 另2条边，拼成一个边长为3的等边三角形。小明希望你提前告诉他能不能拼出来，免得白费功夫。
- 输入格式：首先输入一个整数n(3≤n≤20)，表示木棍数量，接下来输入n根木棍的长度 pi (1≤pi≤10000)。
- 输出格式：如果小明能拼出等边三角形，输出"yes"，否则输出"no"。
- 样例输入1
- 5
- 1 2 3 4 5
- 样例输出1
- yes
- 样例输入2
- 4
- 1 1 1 1
- 样例输出2
- no

习题：正方形

- 小明手上有一些小木棍，它们长短不一，小明想用这些木棍拼出一个等边三角形，并且每根木棍都要用到。 例如，小明手上有长度为 1，2，3，3，3 的5根木棍，他可以让长度为1，2 的木棍组成一条边，另外 3根分别组成 另3条边，拼成一个边长为3的正方形。小明希望你提前告诉他能不能拼出来，免得白费功夫。
- 输入格式：首先输入一个整数n(4≤n≤20)，表示木棍数量，接下来输入n根木棍的长度 pi (1≤pi≤10000)。
- 输出格式：如果小明能拼出正方形，输出"yes"，否则输出"no"。
- 样例输入1
- 4
- 1 1 1 1
- 样例输出1
- Yes
- 样例输入2
- 5
- 10 20 30 40 50
- 样例输出2
- no

习题：八皇后问题

- 小明在和朋友下国际象棋，下的时候突发奇想，在国际象棋棋盘的每个格子上写下1到99内的数字，又拿出了珍藏已久的 8个皇后棋子。国际象棋中的皇后可以将同一行、同一列和同一对角线上的对方棋子吃掉。小小明在想，怎么摆放这 8个皇后的位置才能让她们不能互相攻击，同时这8个皇后占的格子上的数字总和最大。
- 小明来求助热爱算法的你了，你能帮她算出答案吗?
- 输入格式：每个棋盘有 64个数字，分成8行 8列输入，就如样例所示。棋盘上每一个数字均小于100。
- 输出格式：输出一个最大的总和
- 样例输入
- 1 2 3 4 5 6 7 8
- 9 10 11 12 13 14 15 16
- 17 18 19 20 21 22 23 24
- 25 26 27 28 29 30 31 32
- 33 34 35 36 37 38 39 40
- 41 42 43 44 45 46 47 48
- 48 50 51 52 53 54 55 56
- 57 58 59 60 61 62 63 64
- 样例输出
- 260

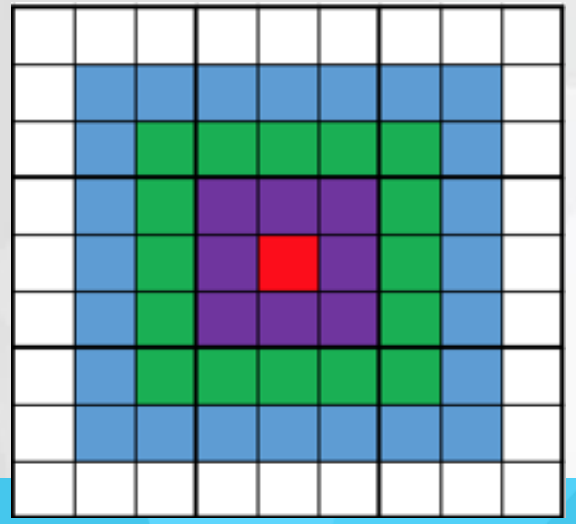
习题：金字塔数独

- 小明天资聪颖，酷爱数学，尤其擅长做数独游戏。不过普通的数独游戏已经满足不了小明了，于是他发明了一种“金字塔数独”：
- 下图即为金字塔数独。和普通数独一样，在 9 × 9 的大九宫格中有 9 个 3 ×3 的小九宫格（用粗黑色线隔开的）。要求每个格子上都有一个 1 到 9 的数字，每个数字在每个小九宫格内不能重复出现，每个数字在每行、每列也不能重复出现。
- 但金字塔数独的每一个格子都有一个分值，类似金字塔的俯视图。如图所示，白色格子为 6 分，蓝色格子为 7 分，绿色格子为 8 分，紫色格子为 9 分，红色格子为 10 分。颜色相同的格子分值一样，离中心越近则分值越高。
- 金字塔数独的总分等于每个格子上的数字和对应的分值乘积之和。现在小明给定金字塔数独的若干数字，请问如何填写，可以使得金字塔数独的总分最高。
- 输入格式：输入一共 9 行。每行输入 9 个整数（每个数都在 0—9 的范围内），每两个整数之间用一个空格隔开，“0”表示该格子为空。
- 输出格式：输出为一行，输出一个整数，代表金字塔数独的最高总分。
- 如果数独无解，则输出 –1。
- 样例输入

```
0 0 0 0 0 0 0 0 0
0 0 3 0 0 0 9 0 0
7 9 0 0 0 2 0 1 3
0 0 9 1 5 0 0 0 0
0 7 4 0 2 6 1 3 9
0 0 6 0 0 0 0 0 0
6 0 0 0 0 7 0 0 0
3 1 0 4 0 5 7 9 6
0 0 7 0 0 1 0 4 0
```

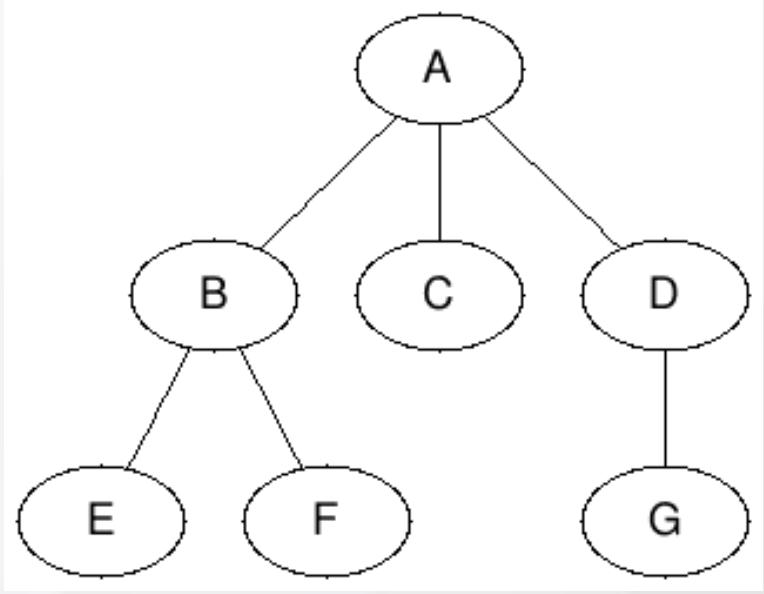
样例输出

```
2864
```



广度优先搜索

- 广度优先搜索(Breadth-first-search),又称宽度优先搜索,简称 BFS,是图的搜索算法之一。与深度优先搜索不同的是，广度优先搜索会先搜索到与起始点距离较近的点，而深搜却是沿着一个分支递归到最后。



- 对上图进行深搜按照顶点访问顺序会得到序列：A-B-E-F-C-D-G
- 对上图进行广搜按照顶点访问顺序会得到序列：A-B-C-D-E-F-G
- 广搜可以理解为，从起点开始一层一层地往外扩展，内层一定会在外层前面被访问到。

BFS的一般算法

- 与深度优先搜索的对比 深度优先搜索用栈（stack）来实现： 把起始顶点压入栈中。 每次从栈顶取出一个顶点，搜索所有它的未访问相邻顶点，把这些顶点压入栈中。 重复执行第二步操作，直至找到所要找的顶点或者栈为空时结束程序。
- 广度优先搜索使用队列（queue）来实现：把起始顶点放到队列中。 每次从队首取出一个顶点，查看这个顶点所有的未访问相邻顶点，把它们放到队尾。 重复执行第二步操作，直至找到所要找的顶点或者队列为空时结束程序。
- BFS的一般写法：

```
1 void bfs(起始点) {
2     将起始点放入队列中;
3     while (如果队列不为空) {
4         访问队列中队首元素x;
5         删除队首元素;
6         for (x 所有相邻点) {
7             if (该点未被访问过且合法) {
8                 将该点加入队列末尾;
9             }
10        }
11    }
12    队列为空，广搜结束;
13 }
```

走迷宫-最少步数

- 在深搜的章节讲解了走迷宫问题，现在再来看一个走迷宫问题： 相信大家都玩过走迷宫。用矩阵来表示一个迷宫
- 1 S...
- 2 .##T
- 3 ....
- 'S'表示起点，'T'表示终点，'#'表示墙壁，'.'表示平地。你需要从'S'出发走到'T'，每次只能上下左右走动，并且不能走出地图和走进墙壁，请你计算出走到终点需要走的最少步数。
- 之前问题是求出所有的方案数，需要遍历整张图所有的可能，现在需要求解最少步数，我们依然可以用深搜来遍历所有可能性，然后从中选取最优的步数。但是广搜则不需要搜索所有的路径，由于它的层数具备由小到大的特点，第一次访问到终点时，记录的步数一定是最少的，也就是最优解。所以在“求解最少”这一类问题上，广搜比深搜更有优势。

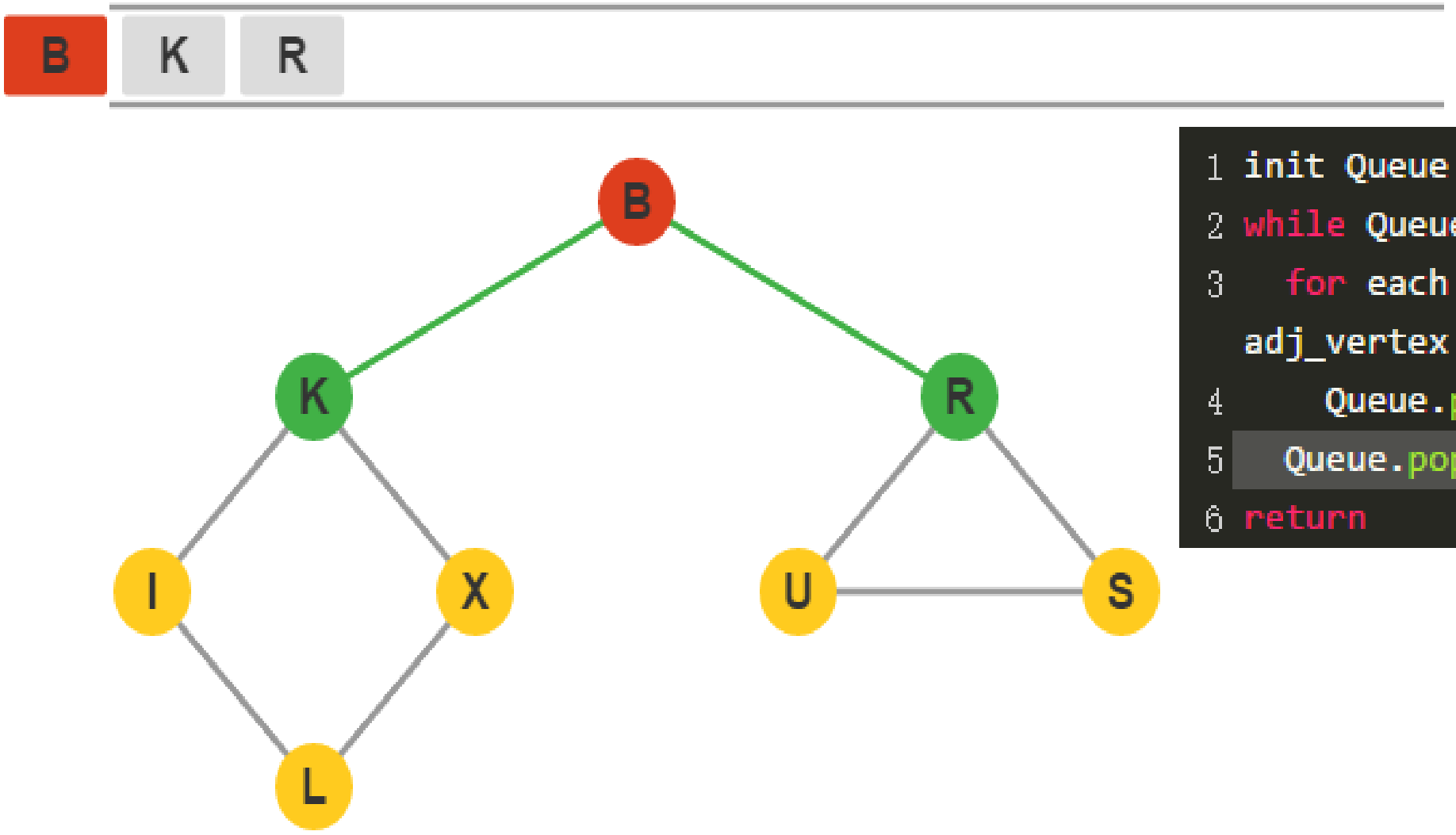
走迷宫问题的广搜

- 对于走迷宫问题的广搜，我们把一个格子的横纵坐标用一个结构体（或class）来存储，并放入队列。这里给出一段体现核心思路的代码（C++ 示例代码）：

```
1 struct point {
2     int x, y;
3     point(int xx, int yy) {
4         x = xx;
5         y = yy;
6     }
7 };
```

```
8 void bfs(int sx, int sy) {  // (sx, sy) 为搜索的起点
9     queue<point> q;
10    q.push(point(sx, sy));  // 将起始点放入队列中
11    用vis数组标记(x, y)已经被访问过;
12    while(队列非空) {
13        取出队列元素(x, y);
14        for (枚举(x, y)能到达的所有格子(tx, ty)) {
15            if (点(tx, ty)合法，可以搜索) {
16                标记(tx, ty)已经被访问过;
17                记录走到(tx, ty)的步数 = 走到(x, y)的
步数 + 1;
18                q.push(point(tx, ty))
19            }
20        }
21    }
22    return;
23 }
```

广度优先搜索的演示




```
1 init Queue with a element vertex0
2 while Queue not empty
3   for each not visited
  adj_vertex∈adj[Queue.head]
4     Queue.push adj_vertex
5   Queue.pop
6 return
```

习题：走迷宫2-最少步数

- 有一个二维迷宫，n行m列，‘s’表示迷宫的起点，‘T’表示迷宫的终点，‘#’表示围墙，‘.’表示通路。现在从S出发，你不能穿墙，问到达终点T最少需要多少步?
- 输入格式：第一行输入n,m（1<=n,m<=50）表示迷宫的行列大小。
- 接下来输入n行字符串表示迷宫。
- 输出格式：一个整数，表示走出迷宫所需的最小步数，若走不出迷宫则输出 -1。
- 样例输入1
- 2 3
- S.#
- ..T
- 样例输出1
- 3
- 样例输入2
- 3 3
- S.#
- .#.
- .#T
- 样例输出2
- -1

习题：一维坐标的移动

- 在一个长度为 n的坐标轴上，小明想从 A 点 移动到 B 点。他的移动规则如下：
- 向前一步，坐标增加 1。
- 向后一步，坐标减少 1
- 跳跃一步，使得坐标乘 2
- 小明不能移动到坐标小于 0 或大于 n 的位置。小明想知道从 A 点移动到 B 点的最少步数是多少，你能帮他计算出来么?
- 输入格式
- 第一行输入三个整数 n，A，B，分别代表坐标轴长度，起始点坐标，终点坐标。（$0 \le A,B \le n \le 5000$）
- 输出格式
- 输出一个整数占一行，代表小明要走的最少步数。
- 样例输入
- 10 2 7
- 样例输出
- 3

习题：小明回家

- 小明要回家，但是他家的钥匙在他的朋友花椰妹手里，他要先从花椰妹手里取得钥匙才能回到家。花椰妹告诉他：“你家的钥匙被我复制了很多个，分别放在不同的地方。”
- 小明希望能尽快回到家中，他需要首先取得任意一把钥匙，请你帮他计算出回家所需要的最短路程。
- 小明生活的城市可以看做是一个 n×m 的网格，其中有道路有障碍，钥匙和家所在的地方可以看做是道路，可以通过。小明可以在城市中沿着上下左右 4 个方向移动，移动一个格子算做走一步。
- 输入格式：第一行有两个整数 n，m。城市的地图是 n 行 m 列。(1≤n,m≤2000)。接下来的 n 行，每行 m 个字符，代表城市的地图。’.’ 代表道路，’#’ 代表障碍物，’S’ 代表小明所在的位置，’T’ 代表小明家的位置，’P’代表钥匙的位置。除了障碍物以外，别的地方都可以通过。(题目保证小明至少有一条路径可以顺利拿到钥匙并且回家)
- 输出格式：输出小明回家要走的最少步数，占一行。
- 样例输入
- 8 10
- P.####.#P#
- ..#..#...#
- ..#T##.#.#
- .....
- ..##.#####
- .....
- #####...##
- ###....S##
- 样例输出
- 21

习题：逃跑

- 小明被困在了一个 n+1 行 m+1 列的迷宫当中，小明所在位置为左上角的 (0,0)，他需要逃跑到位于右下角(n,m) 的出口位置。在逃跑的过程中，小明只可以向东南西北四个方向移动，当然也可以选择停留在某一位置，他每移动一个单位距离需要 1 秒的时间，小明初始时刻的能量为d，小明在迷宫当中每过 1 秒需要消耗 1 单位能量。在迷宫中有 k 个士兵，他们会朝着某一方向周期性地射击，子弹只会在整点位置射中小明。当小明被子弹射中、和士兵相遇或者能量消耗完时，他将死去。求小明逃离迷宫的最短时间。
- 输入格式 :第一行包含四个整数n,m,k,d(2≤n,m≤100;0≤k≤100;m+n≤d≤1000)。 接下来 k 行每行包含一个字符 c 和四个整数 t,v,x,y(t,v≤100;0≤x≤n;0≤y≤m)，其中 c 表示每个士兵的射击方向，使用N、S、W、E表示上下左右四个方向，射击的周期为 t，子弹的速度为 v，士兵所在位置为(x,y)。
- 输出格式 :输出一个整数，表示小明逃跑的最短时间，如果小明不能成功的逃离迷宫，输出Bad luck!。
- 样例输入1
- 4 5 3 10
- N 1 1 1 1
- W 1 1 3 2
- W 2 1 2 4
- 样例输出1
- 10

- 样例输入2
- 4 5 3 10
- N 1 1 1 1
- W 1 1 3 2
- W 1 1 2 4
- 样例输出2
- Bad luck!