广度优先搜索（BFS）

广度优先搜索

- 广度优先搜索是一种分层的查找过程，每向前一层可能会访问一批顶点，不像深度优先搜索有回溯的过程。
- 同时与深搜用栈来维护不同，广搜一般是用先进先出队列来进行维护的。
- 实际上我们从初始状态开始，搜索第k层的意义就是，搜索需要k步才能到达的状态。

广度优先搜索——具体操作、特点

- 具体操作：
- 它是先将起始状态加入队列，然后每次从队列中取出一个状态，将其后继状态加入队列，后继状态指的是由当前状态一步操作可以到达的状态，直到所有状态均被访问为止。

广度优先搜索——具体操作、特点

- 特点：
- 1. 它并不考虑结果的可能位置，而是彻底地搜索所有状态，所以很少有基于 BFS 的启发式算法，也很少对 BFS 进行剪枝。
- 2. 相对于 DFS，BFS 更加难于保存当前节点的状态，所以 BFS **在爆搜中的应用较少**。
- 3. 在某一层还没有搜索完时，是不会进入下一层的，也就是说在队列中所有同一深度的状态，是**连续的一段**。
- （这个性质在之后会用到！）

广度优先搜索——算法框架

• 广搜主要解决的是最优问题，比如最短路径，最少步数等。

```
void bfs(v){
        int queue(q);
        visit(v); visited[v]=true; insert_queue(q,v)；
        while not empty(q) do {
                取出队首元素 v
                delete_queue(q,v);//队首元素出队
                for 对所有v扩展出来的元素w
                        if (not visited[w] ) {
                                visit(w);visited[w]=true;
                                insert_queue(q,w)}
                        }
}
```

经典问题1——数联通块

- 给出一个n*m的网格，每一个有一个颜色，两个格子之间相连当且仅当，两个格子相连且颜色相同。
- 求联通块的数量。

- n*m<=10^5

经典问题1——数联通块

- 举个例子：
- n=4，m=5
- 1 2 1 2 3
- 1 1 1 2 3
- 2 2 2 2 3
- 3 3 3 3 3
- 在这个问题中联通块的数量为4。

经典问题1——数联通块

- 我们从上到下，从左到右，依次遍历每一个点，如果某一个点没有遍历到过，那么ans++，并遍历整个相邻的联通块。

- 代码怎么实现？

经典问题1——数联通块——代码实现

```
for (int i=1;i<=n;i++)
    for (int j=1;j<=m;j++) if (!b[i][j])
    {
        ans++;
        head=tail=1;
        x[1]=i,y[1]=j;//初始加入队列
        b[i][j]=true;
        while (head<=tail)
        {
            int tx=x[head],ty=y[head];head++;
            for (int k=0;k<4;k++) if (pan(tx+qx[k],ty+qy[k],a[tx][ty]))  //遍历所有相邻点
            {
                ++tail;
                x[tail]=tx+qx[k];
                y[tail]=ty+qy[k];
                b[x[tail]][y[tail]]=true;//将同种颜色的相邻点，加入队列。
            }
        }
    }
```

经典问题1——数联通块——一个代码技巧

- 有一个非常经典的编码技巧。
- 对于当前所在的格子，你怎么遍历所有的相邻格子？
- QAQ：“4个if，枚举4个方向”

- 其实有更简单办法，且拓展性很强的方法。

经典问题1——数联通块——一个代码技巧

图1：

```
const int qx[]={0,0,1,-1};
const int qy[]={1,-1,0,0};
```

图2：

```
for (int k=0;k<4;k++)
if (pan(x+qx[k],y+qy[k]))
```

经典问题1——数联通块——一个代码技巧

我们发现，这样写，就可以用一个for循环，代替4个if。

其实如果需要遍历8联通的格子，也是一样的。它的可拓展性非常强。

经典问题2——走迷宫

- 给出一个n*m的网格，其中有一些格子是障碍“*”，不能经过，其他位置是空地“.”，初始时Bob被放在一个位置，求Bob最少走多少步可以走出网格。
- 如果永远都不能走出网格，输出 -1 。
- n,m<=1000

经典问题2——走迷宫

- 举个例子：一个5*5的迷宫，初始在S：(4,3)
- * * * * *
- * . . . *
- * * . . *
- * . S * .
- * * * * *
- 红色的为走出迷宫的路线。

经典问题2——走迷宫

- 还是套用最开始给出的bfs的方法。
- **最开始**将初始坐标加入队列，**然后**每一次取出队首，扩展相邻没有遍历到过的点，**然后**加入队列中。
- 如果某一时刻我们发现走出了网格了，那么直接输出当前的步数即可。
- 如果队列空了，还是没有走出网格，说明无解，输出-1。

经典问题2——走迷宫——代码实现

```
while (head<=tail)
{
    int tx=x[head],ty=y[head];head++;
    for (int px,py,k=0;k<4;k++) //遍历相邻状态
    {
        px=tx+qx[k],py=ty+qy[k];
        if (px<1||px>n||py<1||py>m) //如果找到了答案，  直接输出
        {
            printf("%d\n",ans[tx][ty]+1);
            return 0;
        }
        if (b[px][py] || s[px][py]=='*') continue;

        ++tail;
        x[tail]=px;
        y[tail]=py;
        ans[px][py]=ans[tx][ty]+1;//统计答案
        b[px][py]=true;//标记已经遍历到过
    }
}
```

广搜深搜的区别与各自适用范围

探讨内容

- 两种搜索方式有什么区别？

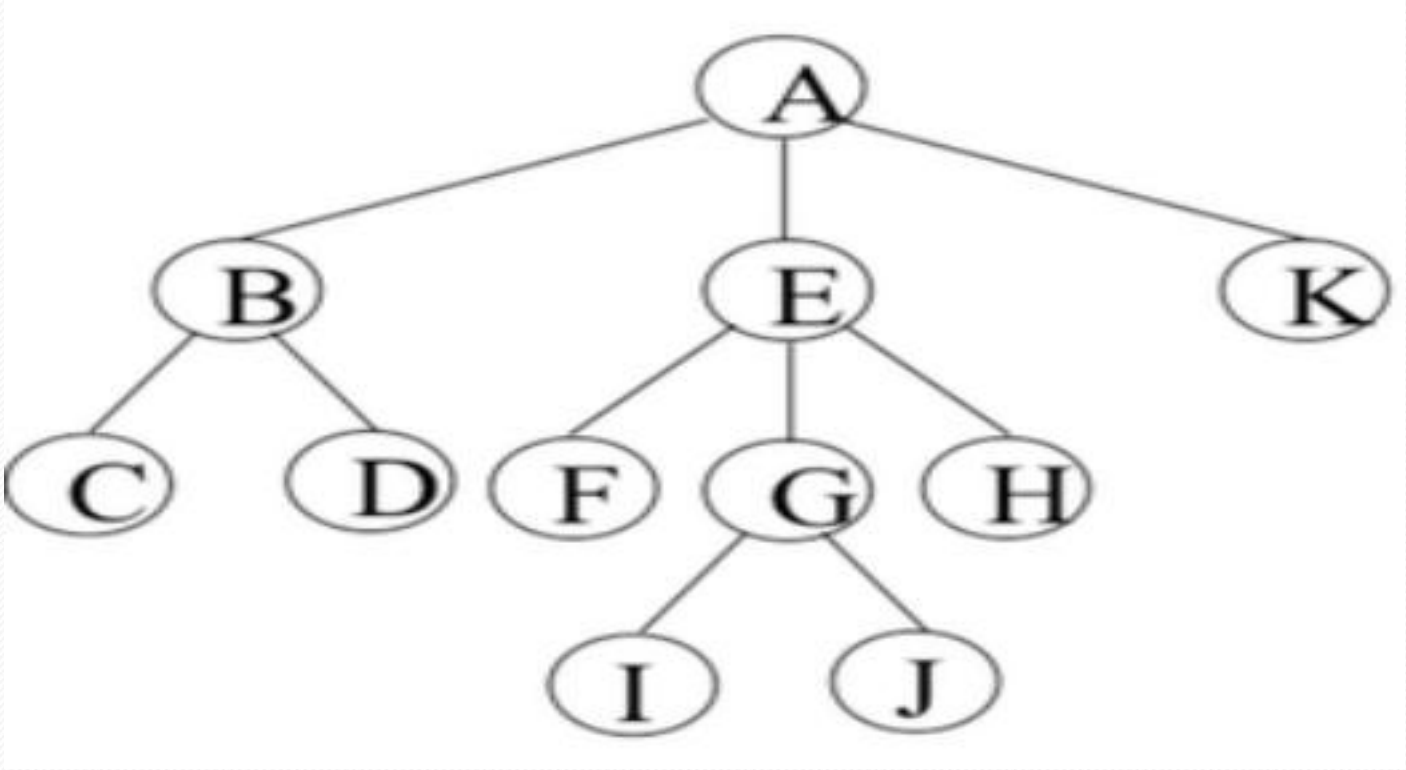
- 正是这种区别，使它们分别适合什么题？
- （**结构决定性质！**）

广度优先搜索与深度优先搜索——区别

- 最直观的：
- 深度优先搜索一般总是一搜搜到底，也就是所谓的**深度优先**。
- 广度优先搜索不一样，是按深度从小到大**一层一层搜索**，不会一次搜到底。

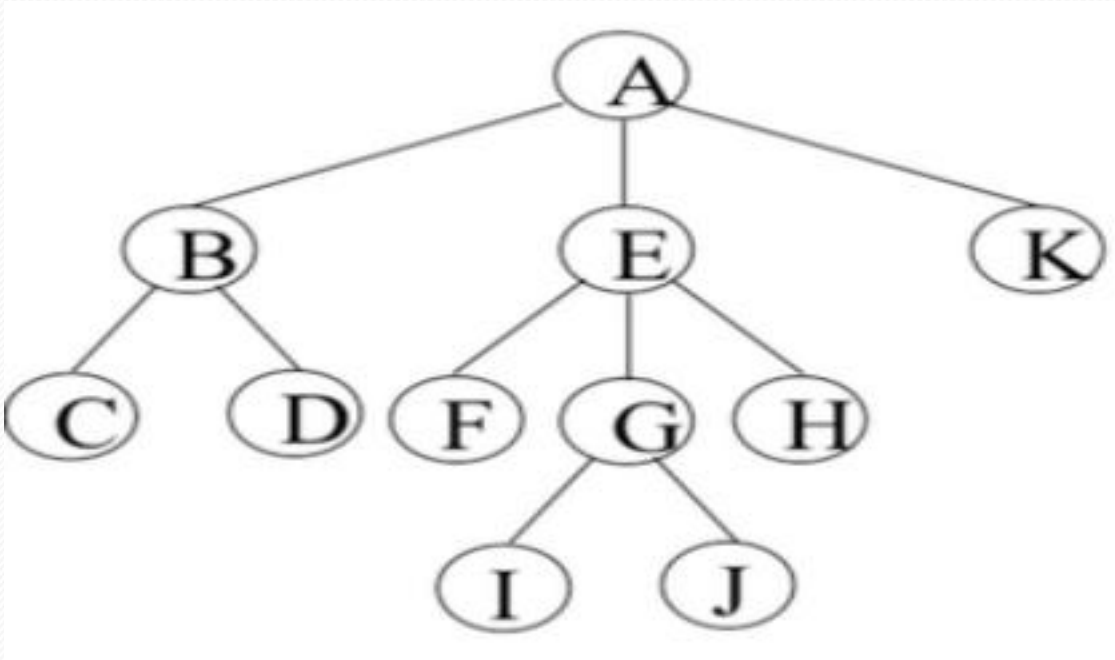
以树的遍历为例

我们分别以深度优先搜索和广度优先搜索来遍历这一棵树。



深度优先搜索

A->B->C->D->E->F->G->I->J->H->K 每一次搜到底！



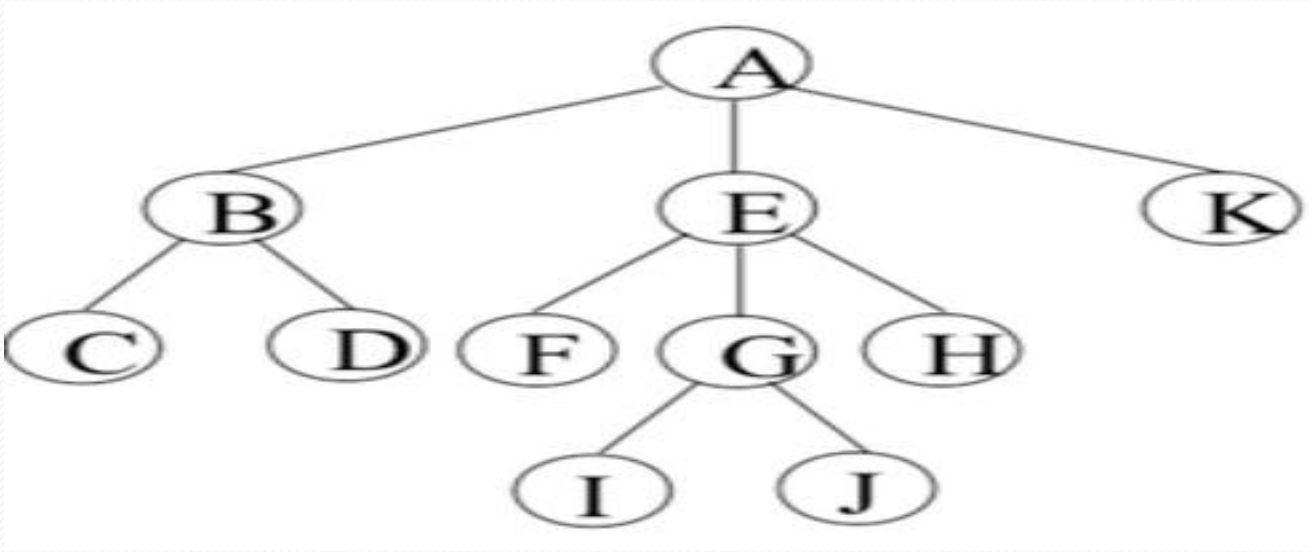
广度优先搜索

A-> （第一层）

B->E->K-> （第二层）

C->D->F->G->H （第三层）

I->J （第四层） 一层一层搜索！



广搜

- 广搜是一层一层，一步一步，所以很多求**最短方案**，最少的步骤数的搜索问题，考虑广搜。

- 这种时候若用深搜，往往会没有一个**明确的终止范围**，容易搜索的过深，导致效率低下。

深搜

- 深搜往往是把**所有可行的情况**列举出来了，找一个**最优解**，不一定是最短的步数。

- 不是有的深搜也是求最短步数吗？

- 注意广搜是**一般不好加剪枝**，但是深搜的剪枝使用比较方便，当搜索的界限比较明确，你也有一个比较成熟剪枝方案时，深搜是可以代替广搜找最短步数的。

速度问题

- 深搜一般是利用系统的栈，通过**回溯来实现**的。
- 而广搜只用**一个队列**。

- 所以广搜的速度一般是优于深搜。
- 同时相比深搜，广搜没有爆栈的风险。

速度问题

- 在一些问题中如果深搜广搜都能做，推荐用广搜。

一句话总结

- 求**最少步数**（每一步代价相同）考虑BFS。
- 求**最优解**（把每一种情况列举出来，通过计算找最优情况）用dfs。

Thanks for listening !