BFS(广度优先搜索算法)

出现场景:

就是让你在一幅”**图**“中找到**从Start到终点target**的最近距离.(**注意里面还具有一定的约束条件**)

算法的基本框架:

```
int BFS(Node start,Node target){
 Queue<Node>q;//核心数据结构
 Set<Node> visited;//避免走回头路
 int step=0;//记录扩散的步数

 q.offer(start);//将起点加入队列

 while(q is not empty){
 int qsize=q.size();
  /****把队列中的所有节点都往外扩散******/
  for(int i=0;i<qsize;i++){
  int curNode=q.pull();
  /*****************判断是否达到终点******************/
  if(curNode==targetNode){
  return step;
  }
  /*********将cur相邻未访问过的节点加入队列中********/
  for(Node x:cur.adj()){
  if(x not in visited){
  q.offer(x);
  visited.add(x);
  }
  }
  }
  /***这里更新步数*************/
  step++;
  }
}
```

判断叶子节点的条件是：没有左右子树

BFS和DFS的区别与联系:

BFS和DFS都可以计算最短距离，但是BFS的空间复杂度远远高于DFS，但BFS的时间复杂度低于DFS。

解开密码锁的最少次数:

```
int openLock(String[] deadends,String target){
 HashMap<String> dead=new HashMap<String>();//存放死亡密码和已经试过的密码
 Queue<String> q=new LinkedList<String>();//存放队列
 int step=0;//解开密码锁的次数
 for(int i=0;i<deadends.size();i++){
 dead.add(deadends[i]);
 }
 q.add("0000");
 dead.add("0000");
  while(q is not empty){
  int size=q.size();
  for(int i=0;i<size;i++){
  String top=q.pop();
  /****判断该节点是否合法**/
  if(dead.contains(top))continue;
  /*****判断该节点是否是最终的节点***/
  if(top.equals(top)) return step;

  /**将该节点的相邻节点向外扩散*****/
  for(int j=0;j<4;j++){
  String up=plusOne(cur,j);
  if(!dead.contains(up)){
  q.add(up);
  dead.add(up);
  }
  String down=minusOne(cur,j);
  if(!dead.contains(down)){
  q.add(down);
  dead.add(down);
  }
  }
  }
  step++;
  }
  return -1;

}
```

双向BFS算法框架:

双向BFS则**从起点和终点**同时开始扩散，当**两边有交集的时候**停止。

出现场景：寻找从起点start到终点target的最短路径，必须知道起点start和终点target的位置才能同时扩散。

```
int openLock(String[] deadends,String target){
 Set<String> deads=new HashSet<>();
 for(String s:deadends) deadends.add(s);
 //用集合不用队列，可以快速判断元素是否存在
 Set<String> q1=new HashSet<String>();
 Set<String> q2=new HashSet<String>();
 Set<String> visited=new HashSet<String>();

 //起始化起点和终点
  q1.add("0000");
  q2.add(target);
  int step=0;

  while(!q1.isEmpty()&&!q2.isEmpty()){
  //在遍历过程中不能修改哈希集合
  //用temp存储q1的扩散结果
  Set<String> temp=new HashSet<String>();

  /*****将q1中的所有节点向周围扩散*********/
  for(String cur:q1){
  if(deads.contains(cur)) return false;
  if(q2.contains(cur)) return step;
  visited.add(cur);

  /**********将一个节点的未遍历相邻节点加入集合********/
  for(int j=0;j<4;j++){
  String up=plusOne(cur,j);
  if(!dead.contains(up)){
  temp.add(up);
  dead.add(up);
  }
  String down=minusOne(cur,j);
  if(!dead.contains(down)){
  temp.add(down);
  dead.add(down);
  }
  }
```

```
38  /***********增加步数*******************/
39  step++;
40  /******************在这里交换q1,q2,下一轮while会扩散q2***************/
41  q1=q2;
42  q2=temp;
43  }
44  }
45  return -1;
46 }
```