ACM 程序设计

计算机学院 刘春英

今天。

你 AC 了吗？

每周一星（7）：

NPEG-MP4 &&
OPPO MP4

第八讲

一招制敌之搜索题

统计信息：

根据“信息学初学者之家”网站的统计，Ural（俄罗斯的Ural州立大学的简称，那里设立了一个Ural Online Problem Set，并且支持Online Judge。）的题目类型大概呈如下的分布：

搜索	动态规划	贪心	构造	图论
约10%	约15%	约5%	约5%	约10%
计算几何	纯数学问题	数据结构	其它	
约5%	约20%	约5%	约25%	

搜索题特点分析:

- 题意容易理解
- 算法相对固定
- 编程有路可循
- 竞赛必备知识

引言

“算法中*最基本和常用*的是搜索，主要是回溯和分支限界法的使用。这里要说的是，有些初学者在学习这些搜索基本算法是不太注意剪枝，这是十分不可取的，因为所有搜索的题目给你的测试用例都不会有很大的规模，你往往察觉不出程序运行的时间问题，但是真正的测试数据一定能过滤出那些没有剪枝的算法。实际上参赛选手基本上都会使用常用的搜索算法，题目的区分度往往就是建立在诸如剪枝之类的优化上了。”

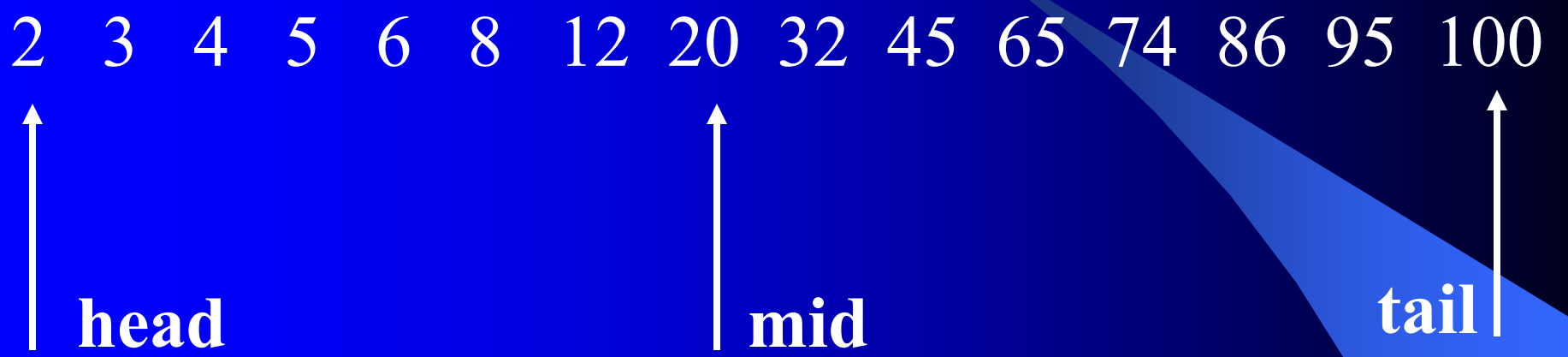
——摘自《ACM竞赛之新人向导》

什么是搜索算法呢？

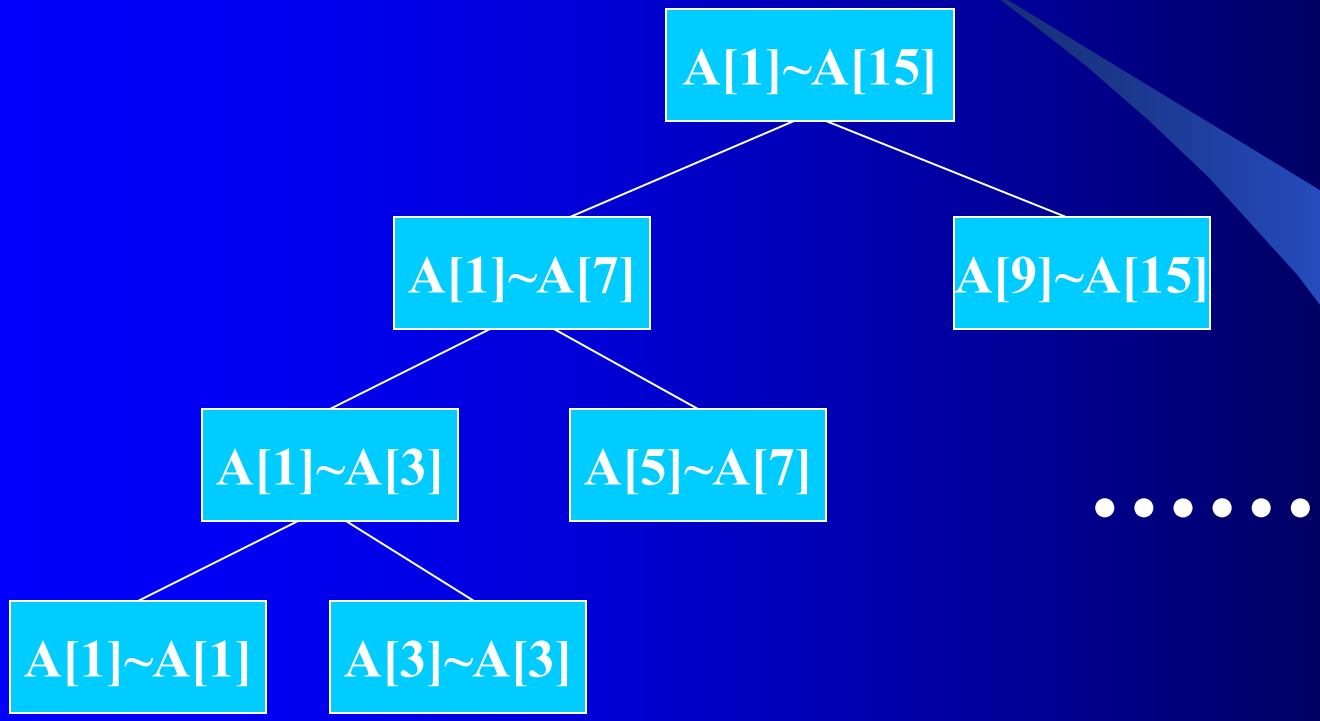
搜索算法是利用计算机的高性能来有目的地穷举一个问题的部分或所有的可能情况，从而求出问题的解的一种方法。

搜索过程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。

预热一下：二分查找



查找示意图：



思考：

- 1、在一百万个元素里查找某个元素大约需要比较多少次？
- 2、时间复杂度：O(logN)

举例分析

从简单的字符串搜索讲起

HDOJ_1238 Substrings

- 题目链接

- **Sample Input**

```
2
3
ABCD
BCDFF
BRCD
2
rose
orchid
```

- **Sample Output**

```
2
2
```

题目分析：

- 这是一道入门级别的搜索题，基本思想比较简单，但是如果用最朴素的算法，可能会超时如何降低算法的复杂度呢？

下面的算法如何：

先将字符串按长度从短到长排序，枚举最短的字符串的子串，判断是否都是别的字符串的子串，求出最大长度即可。

说明：

本题除了可以练习基本搜索算法，也是练习字符串处理的好题目，题中用到的相关知识点有：

- 求反串
- 求子串
- 字符串查找
- 求字符串长度

强烈推荐！！

再来一道数值型搜索题

HDOJ_1239

Calling Extraterrestrial Intelligence Again

- 题目链接

- **Sample Input**

```
5 1 2
99999 999 999
1680 5 16
1970 1 1
2002 4 11
0 0 0
```

- **Sample Output**

```
2 2
313 313
23 73
43 43
37 53
```

获取有用信息

a.给定整数m,a,b(4 < m <= 100000 and 1 <= a <= b <= 1000)

b.需要找到两个数(不妨设为p,q)满足以下条件：

- p,q均为质数；
- p*q<=m;
- a/b <= p/q <= 1;

c.输出所有满足以上条件的p,q中乘积最大的一对p,q

算法分析

1.典型的搜索

从所有可能的p,q中寻找满足条件的一对

2.p,q的要求

p,q均为质数,且p<=q<=100000;

3.按上述思想流程应为

a.从1—100000中搜出质数

b.两层循环，试遍所有的组合(p,q可能相等）

c.每种组合去判断是否符合条件，如是，将p*q与当前最大值比较，判断，保存

面临的问题：

超时！

从1—100000的质数运算约为1e+8,而这只是准备工作。

因此，如不加以分析简化此题无法在规定时间内出解

深入分析

p,q的范围其实可在2—50000(why?)

然而，这是最小的范围吗？

考虑大于10000的某个质数，不妨设为Q，另一个质数为P，则：

- 如果P<10，P/Q<0.001
- 如果P>10，P*Q>100000

而考虑到a,b的取值范围(1<=a<=b<=1000)

可知min(a/b)=0.001

同时，要求：p*q<=m<=100000

所以无论如何质数都不能超过10000。（事实上，不会超过9091）

搜索时的技巧：

- 搜索顺序很重要。建议从大往小搜

（ num:质数的个数 ）

```
for (i=num-1;i>=0;i--)
        for (j=i;j<=num-1;j++)
                ……
```

- 注意剪枝：

```
If ( a[j]>m || a[j]*a[i]>m || ( (double)a[i]/a[j])<s )
        ……
```

真正的搜索题

迷宫搜索

预备知识——树的遍历

树的遍历主要有如下四种方法：

1.先根/序遍历
2.中根/序遍历
3.后根/序遍历
4.层次遍历

分别有什么特点呢？

（1）先根遍历

对树的访问次序是:

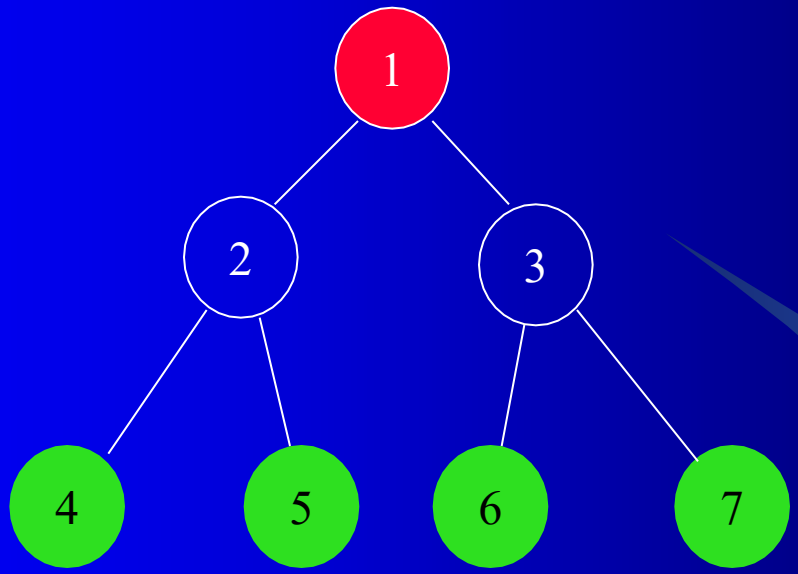
1.先访问根结点

2.再访问左子树

3.最后访问右子树

4.对于左右子树的访问也要满足以上规则

示例如下：



以上二叉树的先根遍历序列是：？？

1、2、4、5、3、6、7

（2）中根遍历

对树的访问次序是:

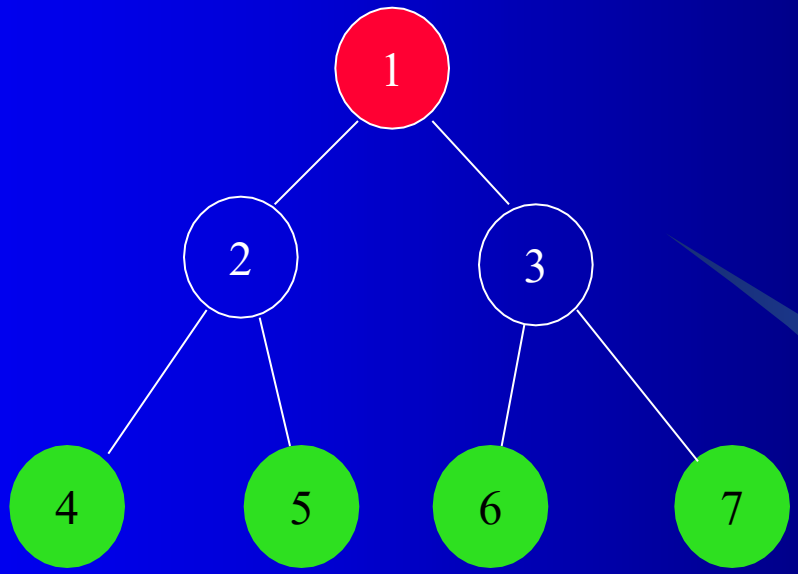
1.先访问左子树

2.再访问根结点

3.最后访问右子树

4.对于左右子树的访问也要满足以上规则

示例如下：



以上二叉树的中根遍历序列是：？？

4、2、5、1、6、3、7

（3）后根遍历

对树的访问次序是:

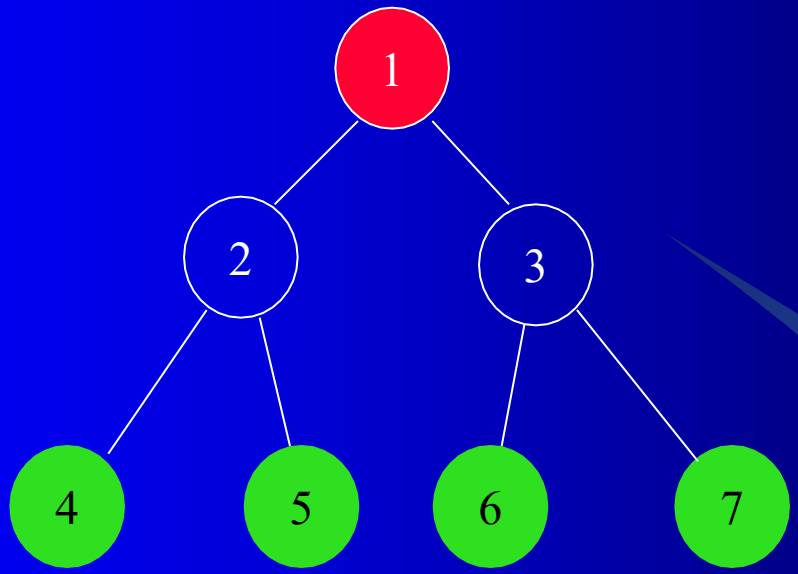
1.先访问左子树

2.再访问右子树

3.最后访问根结点

4.对于左右子树的访问也要满足以上规则

示例如下：



以上二叉树的后根遍历序列是：？？

4、5、2、6、7、3、1

（4）层次遍历

对树的访问次序是:

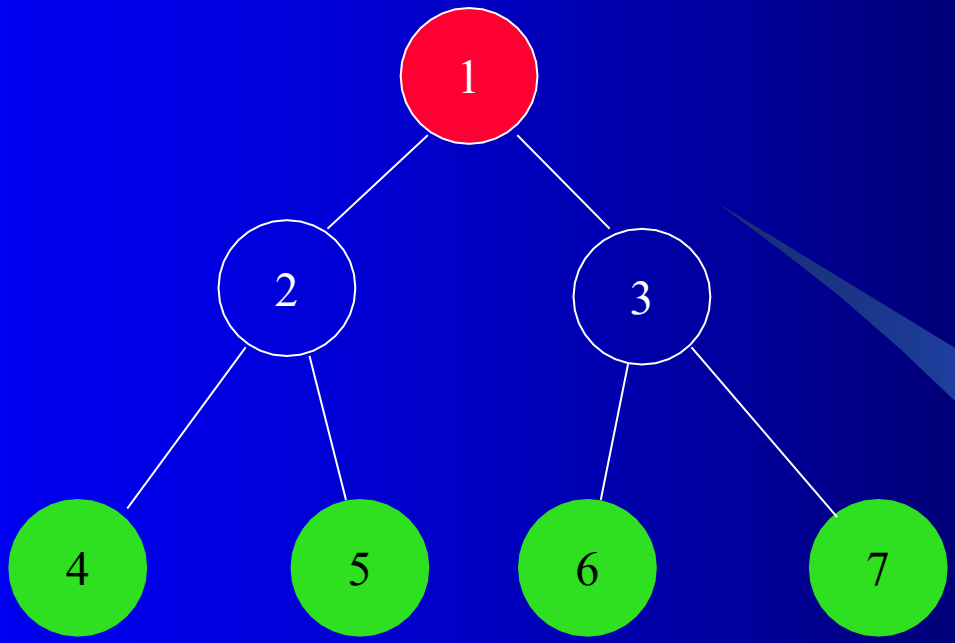
1.先访问根结点

2.再访问根结点的子节点（即第二层节点）

3.再访问第三层节点

4. ……

示例如下：



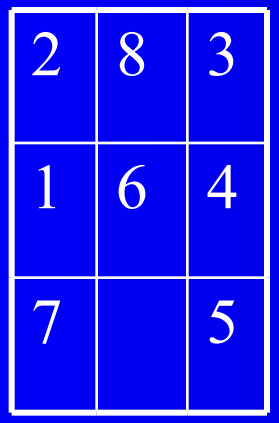
以上二叉树的层次遍历序列是：？？

1、2、3、4、5、6、7

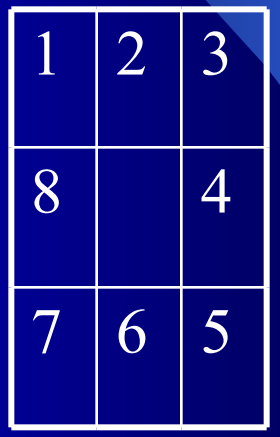
几个基本概念：

- 初始状态：略
- 目标状态：略
- 状态空间：由于求解问题的过程中分枝有很多（主要是求解过程中求解条件的不确定性、不完备性造成的），使得求解的路径很多，这就构成了一个图，我们说这个图就是状态空间。
- 状态空间搜索：就是将问题求解过程表现为从初始状态到目标状态寻找这个路径的过程。通俗点说，就是在解一个问题时，找到一条解题的过程，可以从求解的开始到问题的结果。

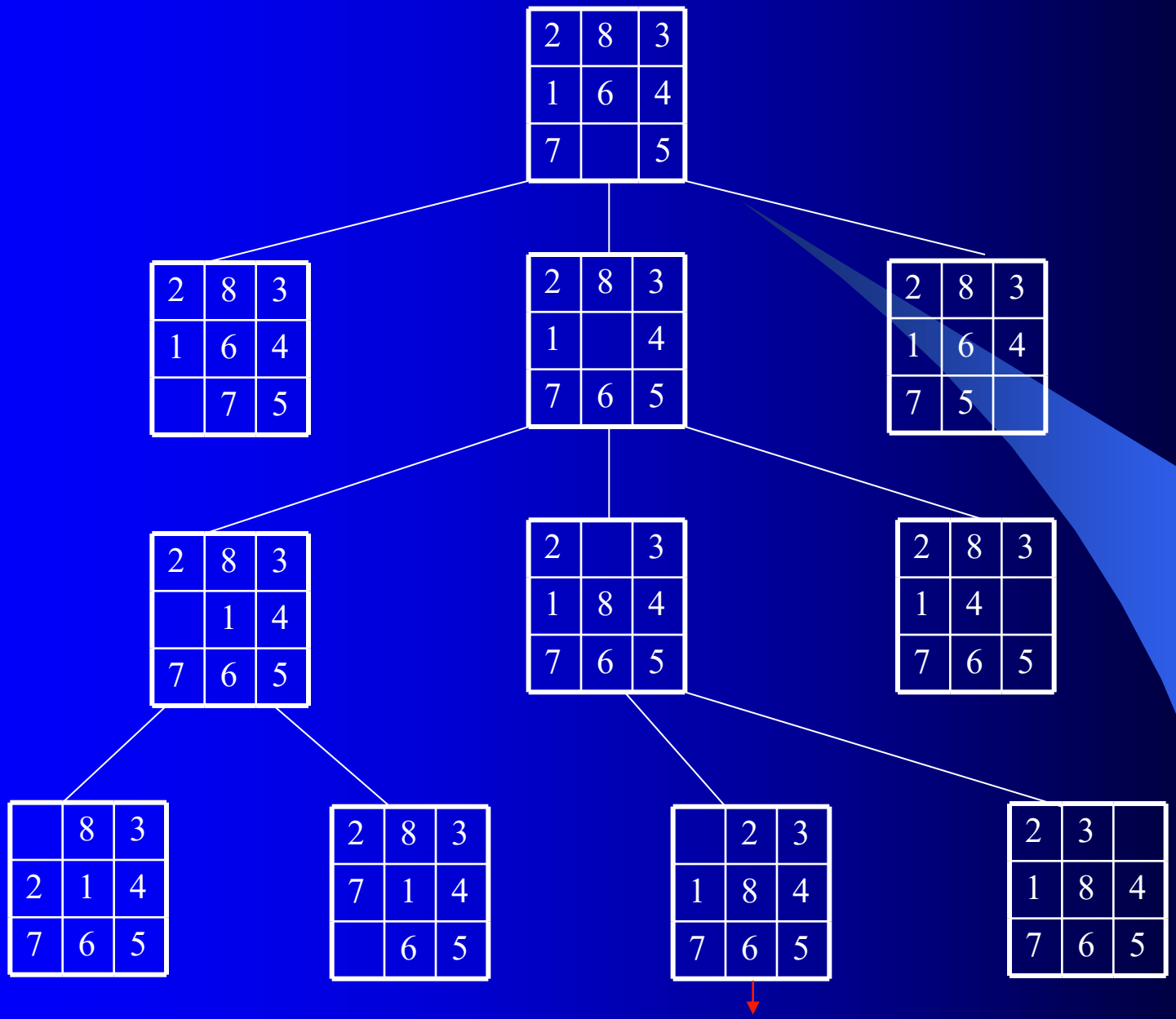
例 九宫重排问题

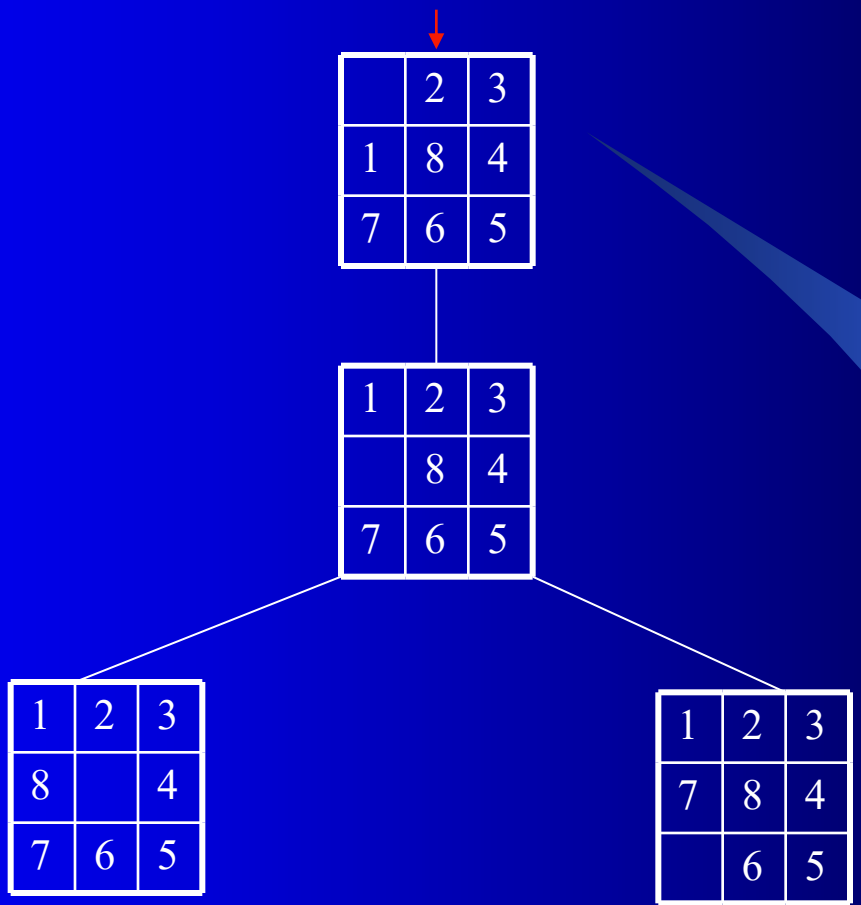


初始状态：



目标状态：


2 8 3
1 6 4
7 5
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 8 3
1 6 4
7 5
2 8 3
1 4
7 6 5
2 3
1 8 4
7 6 5
2 8 3
1 4
7 6 5
8 3
2 1 4
7 6 5
2 8 3
7 1 4
6 5
2 3
1 8 4
7 6 5
2 3
1 8 4
7 6 5


2 3
1 8 4
7 6 5
1 2 3
8 4
7 6 5
1 2 3
8 4
7 6 5
1 2 3
7 8 4
6 5

三、广度优先搜索

基本思想：从初始状态S开始，利用规则，生成所有可能的状态。构成树的下一层节点，检查是否出现目标状态G，若未出现，就对该层所有状态节点，分别顺序利用规则。生成再下一层的所有状态节点，对这一层的所有状态节点检查是否出现G，若未出现，继续按上面思想生成再下一层的所有状态节点，这样一层一层往下展开。直到出现目标状态为止。

BFS算法：

（1）把起始节点S线放到OPEN表中

（2）如果OPEN是空表，则失败退出，否则继续。

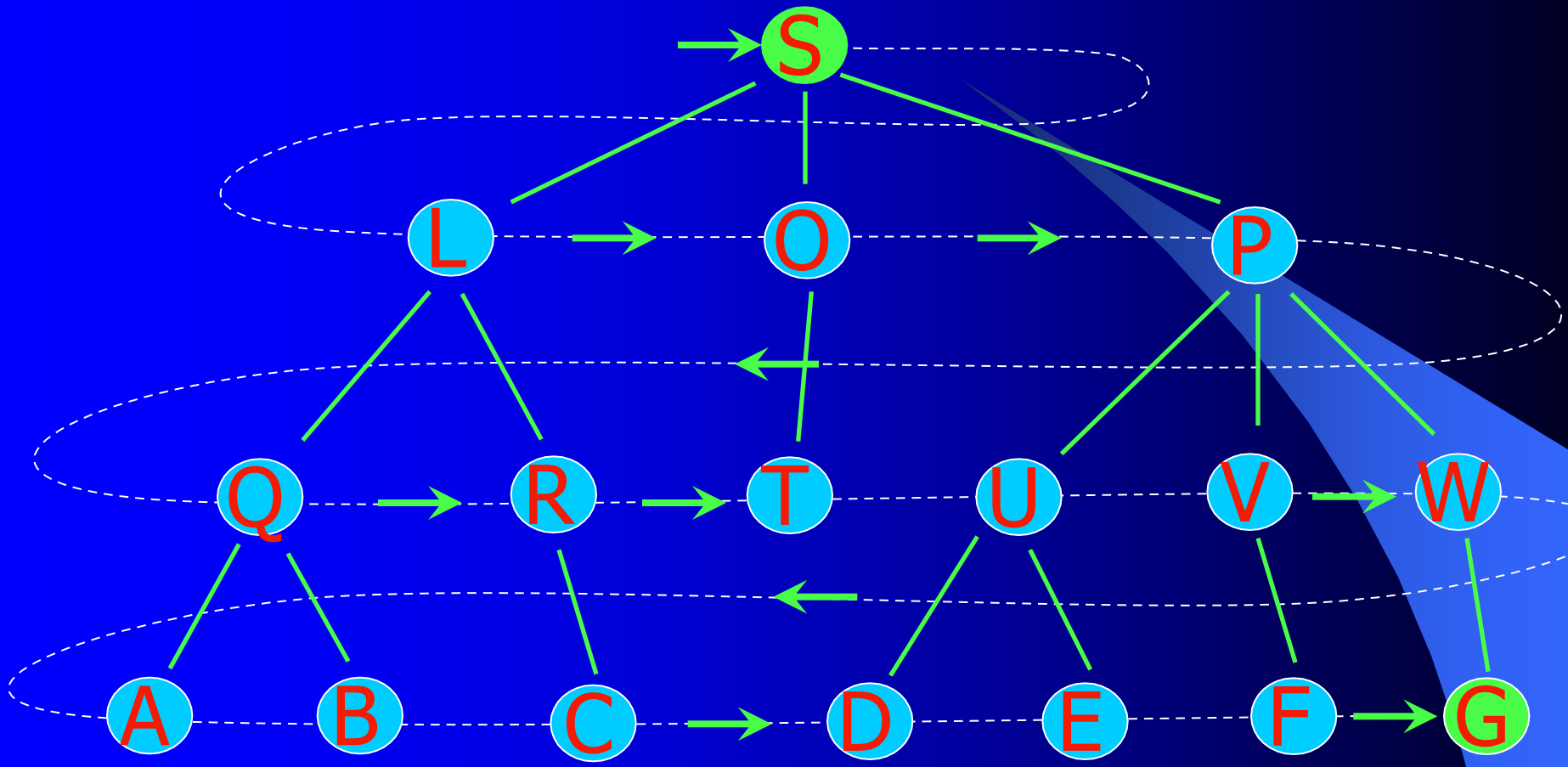
（3）在OPEN表中取最前面的节点node移到CLOSED 表中。

（4）扩展node节点。若没有后继（即叶节点），则转向（2）循环。

（5）把node的所有后继节点放在OPEN表的末端。各后继结点指针指向node节点。

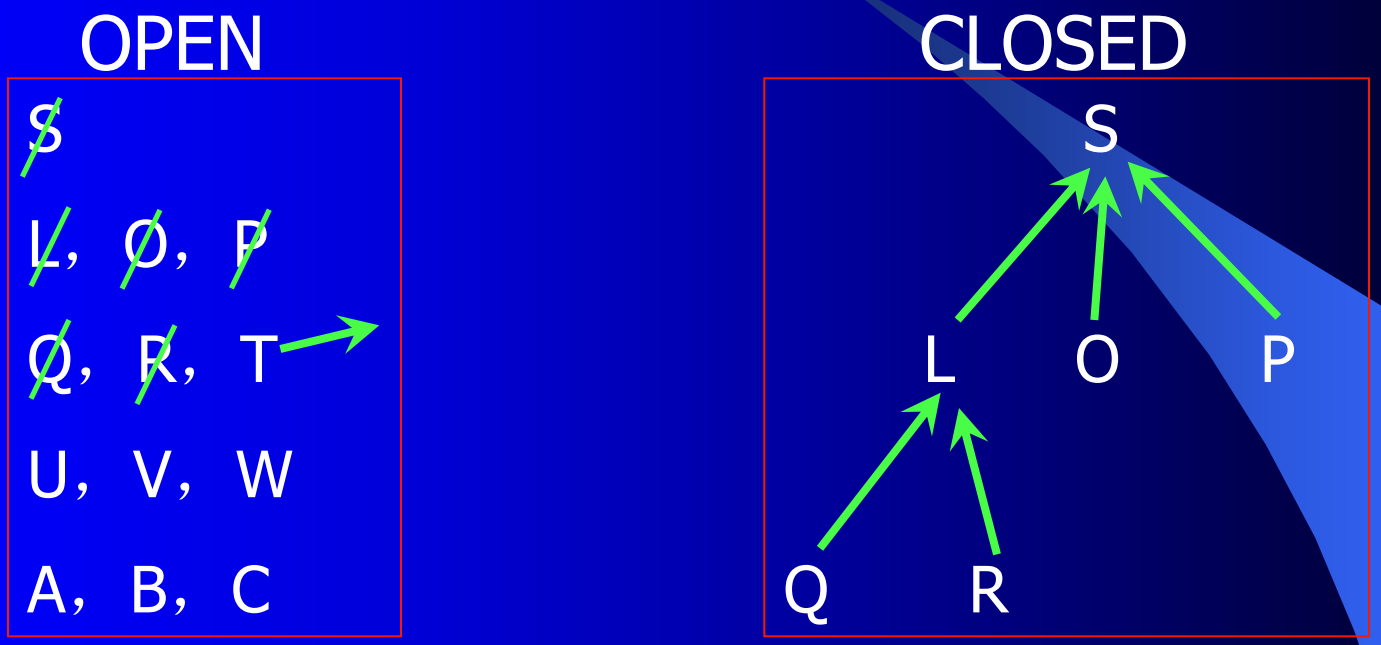
（6）若后继节点中某一个是目标节点，则找到一个解，成功退出。否则转向（2）循环。

搜索过程如下：



广度优先搜索示意图

例1、示意图节点的搜索

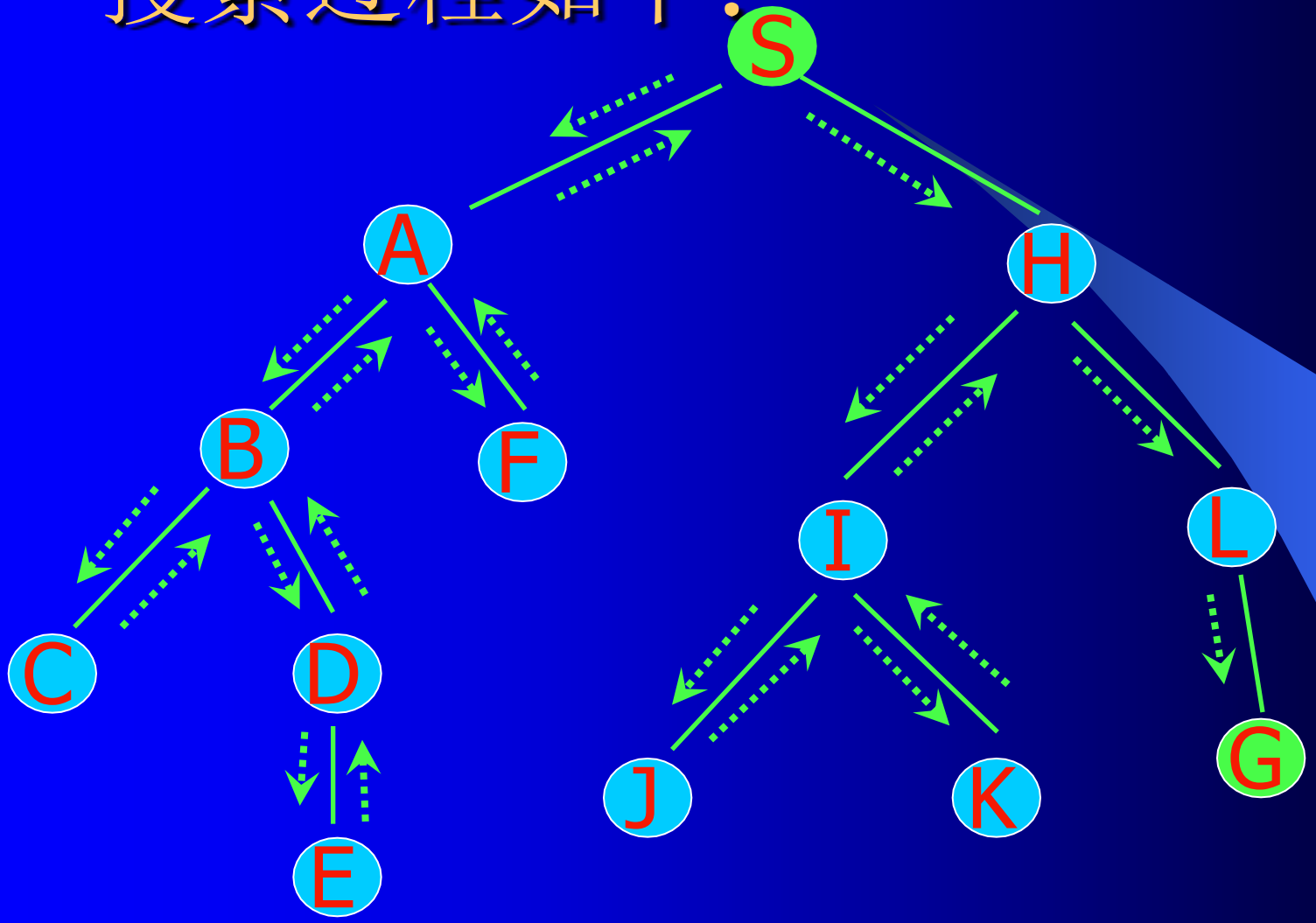


广度优先搜索过程中的OPEN表和CLOSED表

四、深度优先搜索

基本思想： 从初始状态S开始，利用规则生成搜索树下一层任一个结点，检查是否出现目标状态G，若未出现，以此状态利用规则生成再下一层任一个结点，再检查是否为目标节点G，若未出现，继续以上操作过程，一直进行到叶节点（即不能再生成新状态节点），当它仍不是目标状态G时，回溯到上一层结果，取另一可能扩展搜索的分支。生成新状态节点。若仍不是目标状态，就按该分支一直扩展到叶节点，若仍不是目标，采用相同的回溯办法回退到上层节点，扩展可能的分支生成新状态，…，一直进行下去，直到找到目标状态G为止。

搜索过程如下：



深度优先搜索示意图

DFS算法

（1）把起始节点S线放到OPEN表中。

（2）如果OPEN是空表，则失败退出，否则继续。

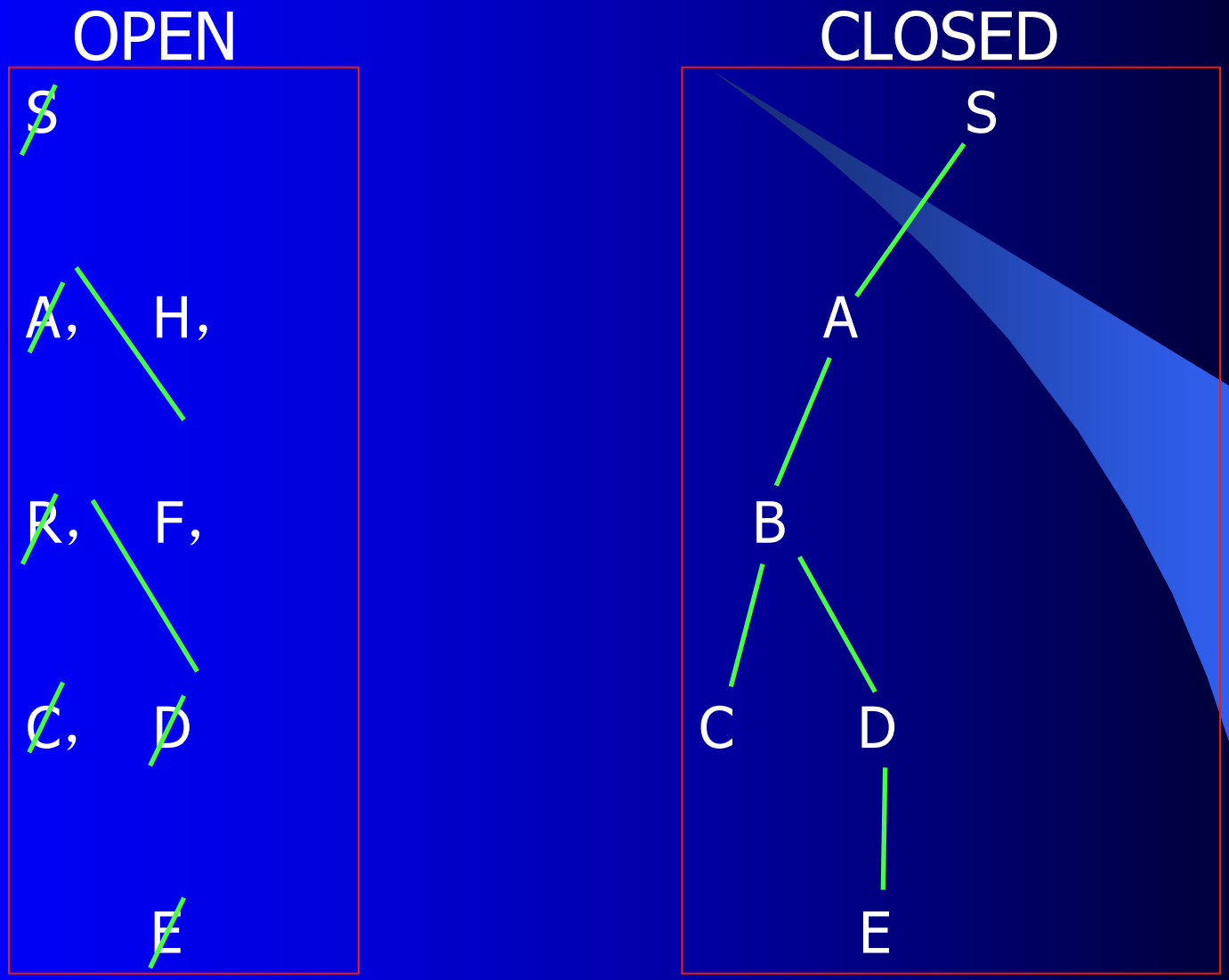
（3）从OPEN表中取最前面的节点node移到CLOSED 表中。

（4）若node节点是叶结点（若没有后继节点），则转向（2）。

（5）扩展node的后继节点，产生全部后继节点，并把他们放在OPEN表的前面。各后继结点指针指向node节点。

（6）若后继节点中某一个是目标节点，则找到一个解，成功退出。否则转向（2）循环。

例、节点搜索示意图



小结：

广度和深度优先搜索有一个很大的缺陷,就是他们都是在一个给定的状态空间中穷举。这在状态空间不大的情况下是很合适的算法，可是当状态空间十分大，且不预测的情况下就不可取了。他的效率实在太低，甚至不可完成。

所以，在这里再次强调**“剪枝”！**

HDOJ_1010 Tempter of the Bone

- **Sample Input**

```
4 4 5
S.X.
..X.
..XD
....
3 4 5
S.X.
..X.
...D
0 0 0
```

- **Sample Output**

NO

YES

要点分析：

- 典型的迷宫搜索，做出该题将具有里程碑式的意义！
- 每个block只能走一次
- 要求恰好某个给定的时间到达出口

想到了什么剪枝条件？

如果可走的block的总数小于时间，将会产生什么情况？

还想到什么剪枝？

奇偶性剪枝

- 可以把map看成这样：
- 0 1 0 1 0 1
- 1 0 1 0 1 0
- 0 1 0 1 0 1
- 1 0 1 0 1 0
- 0 1 0 1 0 1
- 从为 0 的格子走一步，必然走向为 1 的格子
- 从为 1 的格子走一步，必然走向为 0 的格子
- 即：
- 0 ->1或1->0 必然是奇数步
- 0->0 走1->1 必然是偶数步

结论：

所以当遇到从 0 走向 0 但是要求时间是奇数的，或者， 从 1 走向 0 但是要求时间是偶数的 都可以直接判断不可达！

这个题目没问题了吧？

思考：

- 求某给定时间以内能否找到出口
- 找到出口的最短时间
- 条件变为可以停留

附录：推荐搜索题：

- 1010、1240、1241、1242
- 1072、 1253 、 1312、1372

(以上题目类似于1010)

- 1238、1239、1015、1016
- 1401、1515、1548

课后一定要练习！

ACM，天天见！

附录：hdoj_1010月下版

```
# include <iostream.h>
# include <string.h>
# include <stdlib.h>
char map[9][9];
int n,m,t,di,dj;
bool escape;
int dir[4][2]={{0,-1},{0,1},{1,0},{-1,0}};
void dfs(int si,int sj,int cnt)
{    int i,temp;
  if(si>n||sj>m||si<=0||sj<=0) return;
  if(cnt==t&&si==di&&sj==dj) escape=1;
  if(escape) return;

  temp=(t-cnt)-abs(si-di)-abs(sj-dj);
  if(temp<0||temp&1) return;
  for(i=0;i<4;i++){
    if(map[si+dir[i][0]][sj+dir[i][1]]!='X')
          {
      map[si+dir[i][0]][sj+dir[i][1]]='X';
      dfs(si+dir[i][0],sj+dir[i][1],cnt+1);
      map[si+dir[i][0]][sj+dir[i][1]]='.';
    }
  }
  return;
}
```

```
int main()
{
        int i,j,si,sj;
        while(cin>>n>>m>>t)
        {

if(n==0&&m==0&&t==0) break;
          int wall=0;
          for(i=1;i<=n;i++)
      for(j=1;j<=m;j++)
                        {
        cin>>map[i][j];
        if(map[i][j]=='S') { si=i; sj=j; }
        else if(map[i][j]=='D') { di=i; dj=j; }
        else if(map[i][j]=='X') wall++;
      }
    if(n*m-wall<=t)
          {

cout<<"NO"<<endl;
                              continue;
          }
          escape=0;
    map[si][sj]='X';
    dfs(si,sj,0);
    if(escape) cout<<"YES"<<endl;
    else cout<<"NO"<<endl;
  }
  return 0;
}
```