盲目式搜索

万永权

目录

CONTENTS

学习目标

- **理解搜索的基本概念。**
- **掌握盲目搜索算法：宽度优先搜索。**

搜索策略

- 在求解许多问题时都采用试探的搜索方法，**搜索**就是一个不断尝试和探索的过程。
- 为**模拟**这些试探性的**问题求解过程**而发展的一种技术，就称为**搜索技术**。
- **搜索技术**是人工智能中的一个核心技术，它直接关系到智能系统的性能和运行效率。
- 以**状态空间表示法**描述问题空间，已知**初始状态和目标状态**，**搜索问题**就是求解**一个操作序列**使得智能体能从初始状态转移到目标状态。
- **搜索问题的主要任务**是找到正确的搜索策略。
- **搜索策略**是指在搜索过程中确定扩展状态顺序的规则。
- 所求得的从初始状态转移到目标状态的**一个操作序列**就是问题的一个**解**，若某操作序列可以使总代价最低，则该方案称为**最优解**。

图搜索策略

- **一种在状态图中寻找路径的方法。**
- 图中每个节点对应一个状态，每条连线对应一个操作符。
- 图搜索涉及两个主要数据结构：

OPEN 表

CLOSED 表

图搜索策略

OPEN 表

OPEN表是一种动态数据结构，用于记录当前节点的 扩展节点，也称未扩展节点表 。

OPEN表，每个节点的信息还存储了指向父节点的返回地址（即父节点地址）。

OPEN表

节点	父节点编号

◆所谓**扩展节点**，是指用合适的操作（算子）对该节点进行操作，生成一组后继节点。

◆一个节点经过一个算子操作后，一般只生成一个后继节点，但对于一个节点，适用的算子可能有多个，故此时会生成一组后继节点。

图搜索策略

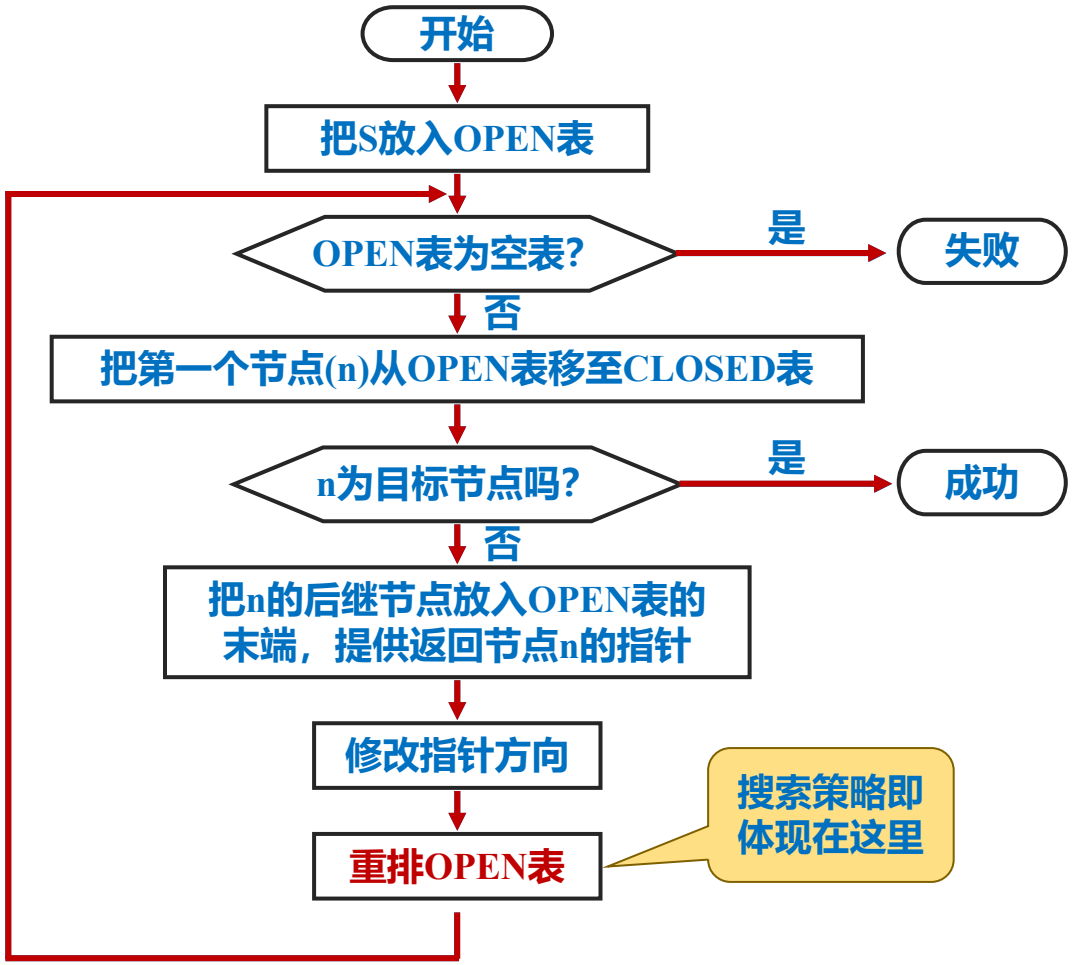
CLOSED 表

CLOSED表是一种动态数据结构，记录访问过的节点，也叫已扩展节点表，其初始为空表。

CLOSED表

编号	节点	父节点编号

图搜索过程



◆**需要注意的是**：在这些后继节点中，可能包含当前扩展节点的父节点、祖父节点，则这些**祖先节点不能作为当前扩展节点的子节点**。

◆图搜索**不允许重复访问节点**，即**OPEN表和CLOSED表的交集为空**。

基于状态空间图的搜索算法的步骤

1.G=G0 (G0=s), OPEN:=(s);

// G是生成的搜索图，**OPEN表**用来存储**待扩展的节点**，每次循环从OPEN表中取出一个节点加以扩展，并把新生成的节点加入OPEN表；

2. CLOSED:=( );

// **CLOSED表**用来存储**已扩展的节点**，其用途是检查新生成的节点是否已被扩展过。

// 图搜索不允许重复访问节点， 即 **OPEN表 ∩ CLOSED表 = ø.**

3. LOOP: IF OPEN=( ) THEN EXIT(FAIL);

4. n:=FIRST(OPEN), REMOVE(n, OPEN), ADD(n, CLOSED);

// 将n从OPEN中删除，加入CLOSED中，即将 n 归入已被扩展的节点

5. IF GOAL(n) THEN EXIT(SUCCESS);

6. EXPAND(n)→$\{m_i\}$, G:=ADD (m_i, G);

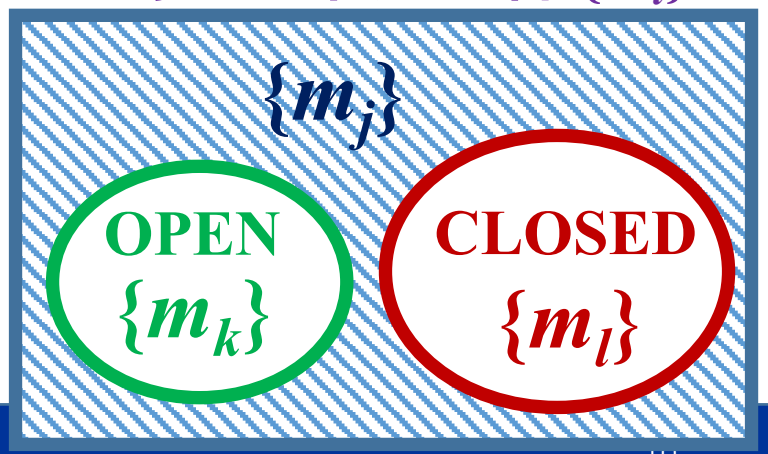
// 扩展节点 n，建立集合$\{m_i\}$, 使其包含n 的后继节点，而不包含n 的祖先，并将这些后继节点加入G中。

注：n 的后继节点有三类： $\{m_i\} = \{m_j\} \cup \{m_k\} \cup \{m_l\}$,

（1）n的后继节点m_j既不包含于OPEN，也不包含于CLOSED；

（2）n的后继节点m_k包含在OPEN中；

（3）n的后继节点m_l包含在CLOSED中；



7. For all nodes in $\{m_i\}$;

//对$\{m_i\}$中所有节点，标记和修改其前驱指针：

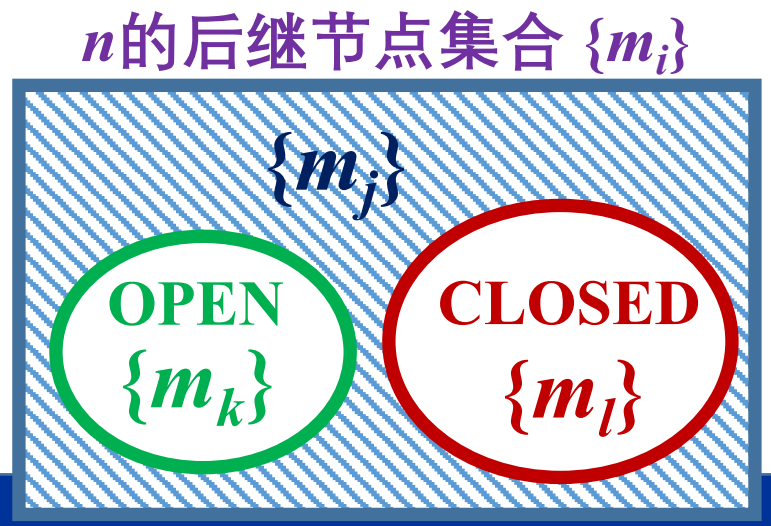
(1) ADD (m_j, OPEN），并令m_j的前驱指针指向n；

(2) 判断是否需要修改 m_k 或 m_l 的前驱指针（都已有前驱），使其指向指向n;

(3) 判断是否需要修改 m_l 后继节点的前驱指针，使其指向m_l本身;

8. 按某种搜索策略对OPEN中的节点重新排序;

9. GO LOOP（语句3）；

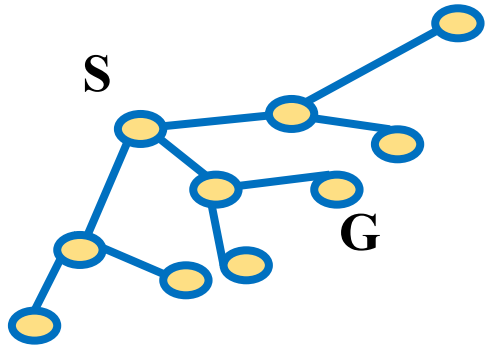


图搜索分类

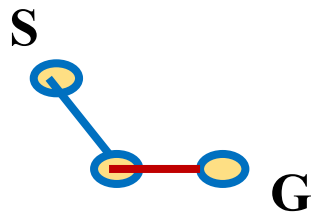
图式搜索 搜索过程中，搜索路径允许形成回路。

树式搜索 搜索过程中，搜索路径不允许形成回路。

线式搜索 搜索过程中，每次只扩展一个节点。



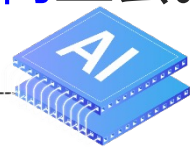
树式搜索



线式搜索

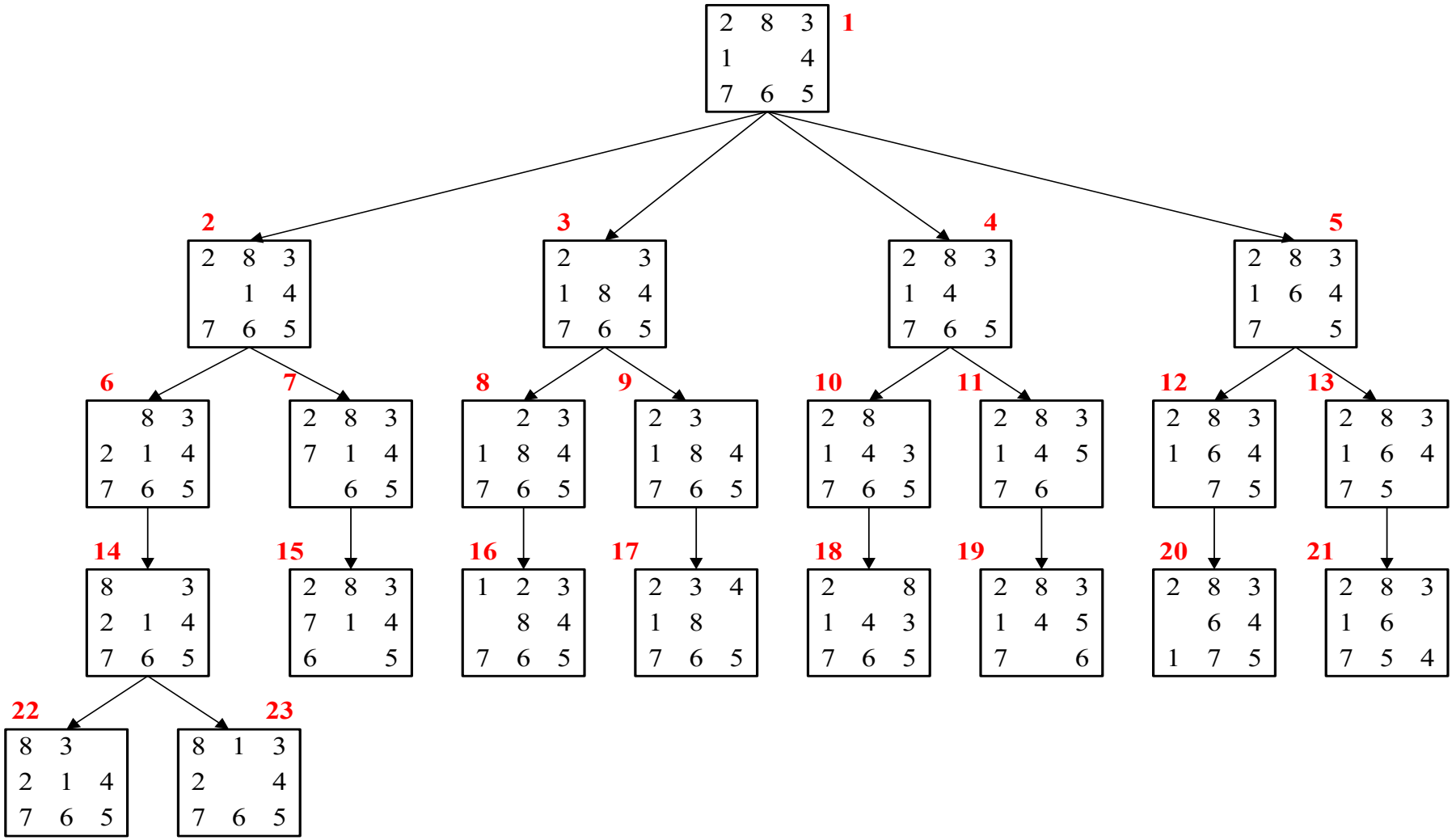
搜索树

一个可以搜索出某个可行解的问题，如“农夫、白菜、羊、狼”和“八数码难题”等，虽然从表面上看上去和“树”这种结构无关，但是整个搜索过程中的可能试探点所行成的搜索空间总可以对应到一颗搜索树上去。



将各类形式上不同的搜索问题抽象并统一成为搜索树的形式，为算法的设计与分析带来巨大的方便。

搜索树


1
2 8 3
1 4
7 6 5
2
2 8 3
1 4
7 6 5
3
2 3
1 8 4
7 6 5
4
2 8 3
1 4
7 6 5
5
2 8 3
1 6 4
7 5
6
8 3
2 1 4
7 6 5
7
2 8 3
7 1 4
6 5
8
2 3
1 8 4
7 6 5
9
2 3
1 8 4
7 6 5
10
2 8
1 4 3
7 6 5
11
2 8 3
1 4 5
7 6
12
2 8 3
1 6 4
7 5
13
2 8 3
1 6 4
7 5
14
8 3
2 1 4
7 6 5
15
2 8 3
7 1 4
6 5
16
1 2 3
8 4
7 6 5
17
2 3 4
1 8
7 6 5
18
2 8
1 4 3
7 6 5
19
2 8 3
1 4 5
7 6
20
2 8 3
6 4
1 7 5
21
2 8 3
1 6
7 5 4
22
8 3
2 1 4
7 6 5
23
8 1 3
2 4
7 6 5

两种方式的搜索策略

（1）在不具备任何与给定问题有关的知识或信息的情况下，系统按照某种固定的规则依次或随机地调用操作算子，这种搜索方法称为**盲目搜索策略**（Blind Search Strategy），又称为**无信息引导的搜索策略**（Uninformed Search Strategy）；

（2）可应用与给定问题有关的领域知识，动态地优先选择当前最合适的操作算子，这种搜索方法称为**启发式搜索策略**（Heuristic Search Strategy）或**有信息引导的搜索策略**（Informed Search Strategy）。

不同搜索策略的搜索性能也不同。

搜索策略性能的评价标准

◆完备性

- 当问题有解时，保证能找到一个解，具有完备性。
- 当问题有解，却找不到，就不具有完备性。

◆最优性

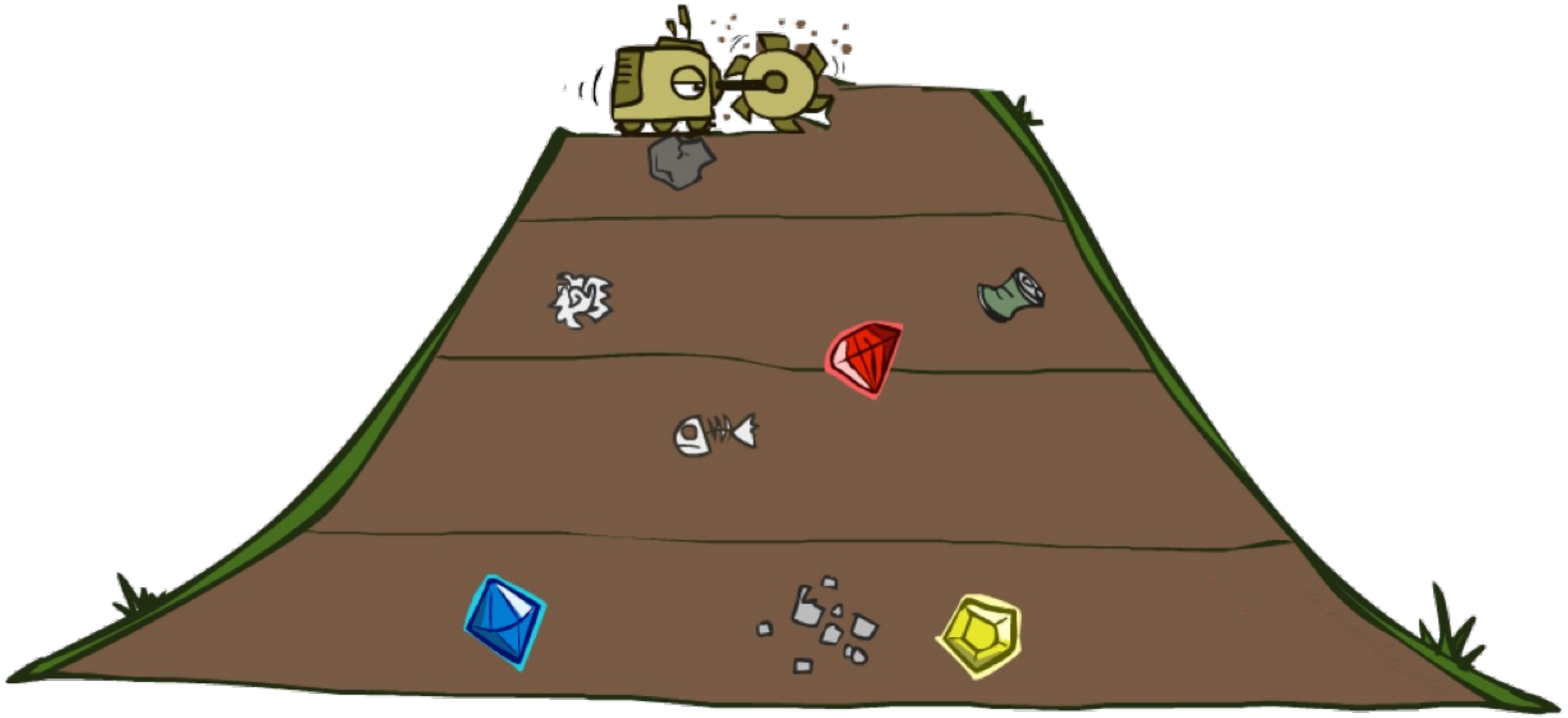
- 当问题有最优解时，保证能找到最优解（最小损耗路径），具有最优性。
- 当问题有最优解，但找不到，找到的只是次优解，则不具有最优性。

盲目搜索

- 盲目搜索（Blind Search）又叫**无信息**搜索（Uninformed Search）。在盲目搜索的过程中，没有任何与问题有关的先验知识或者启发信息可以利用，算法只能判断当前状态是否为目标状态，而无法比较两个非目标状态的好坏。
- 盲目搜索一般只适用于求解比较简单的问题。
- **常用的两种盲目搜索方法：**
  - **（1）深度优先搜索（Depth-first search，DFS）**
  - **（2）宽度优先搜索（Breadth-first search，BFS）**

宽度优先算法

BFS



盲目搜索

宽度优先搜索策略

优先搜索状态空间中离初始状态近的节点（状态）

特点　具有完备性，占用空间。

搜索中使用的数据结构

OPEN表　先进先出队列，存放待扩展的节点。

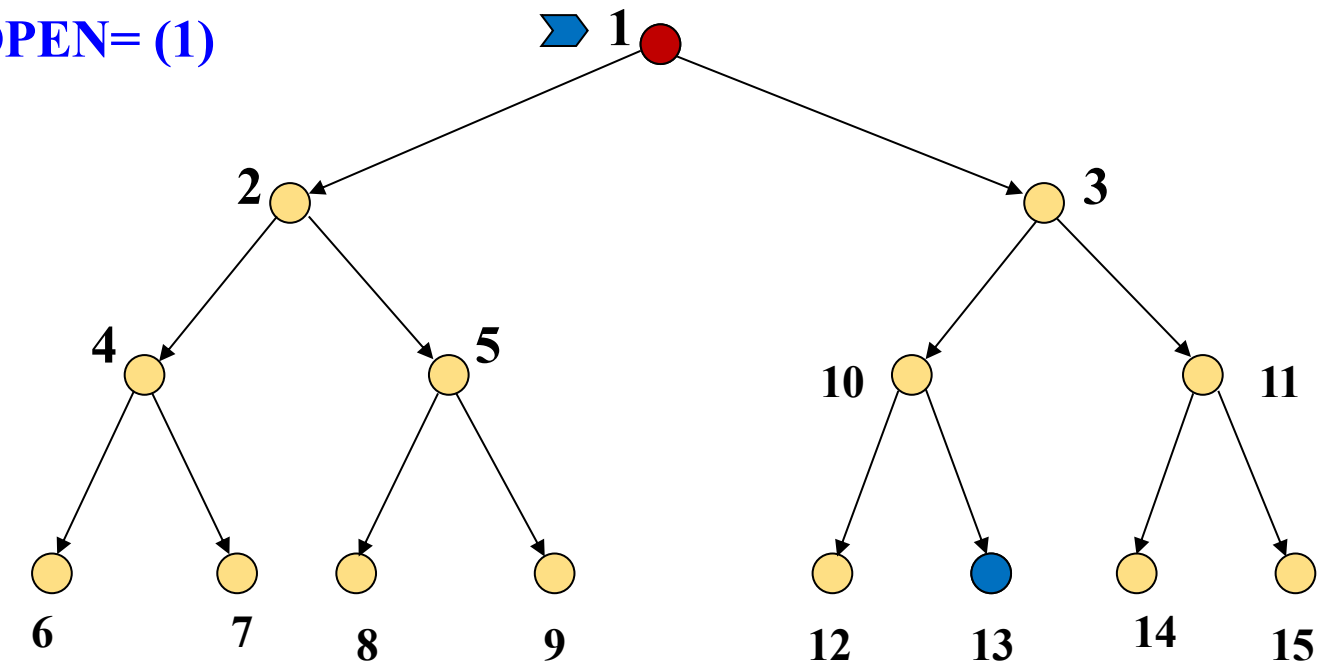
CLOSED表　存放已被扩展过的节点

宽度优先搜索

新的节点被插入到 OPEN 表的 最后

CLOSED=( )

OPEN= (1)

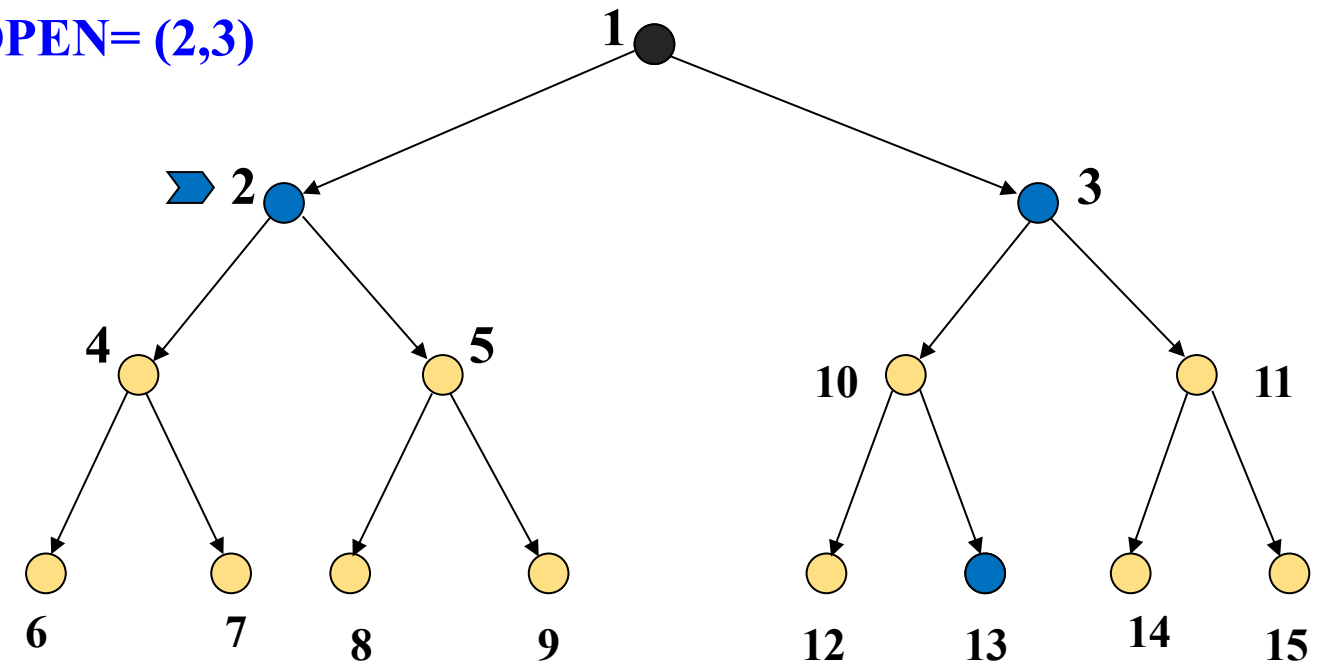


宽度优先搜索

新的节点被插入到 OPEN 表的 最后

CLOSED=(1)

OPEN= (2,3)

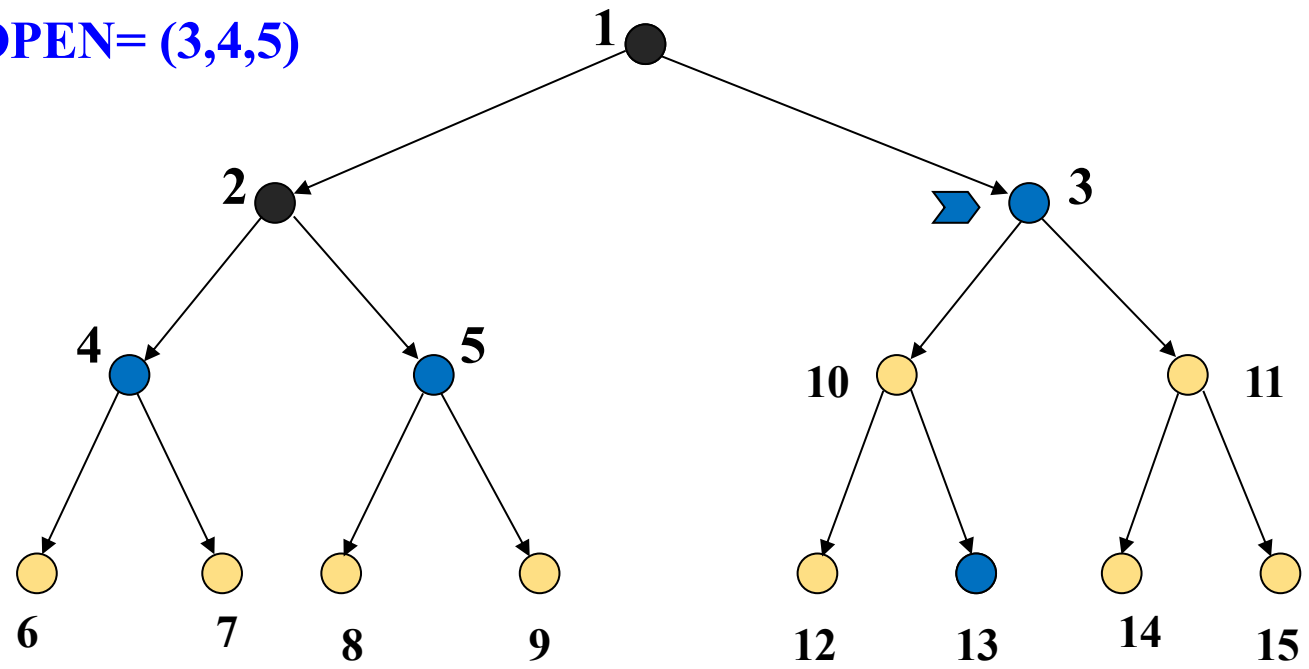


宽度优先搜索

新的节点被插入到 OPEN 表的 最后

CLOSED=(1,2)

OPEN= (3,4,5)

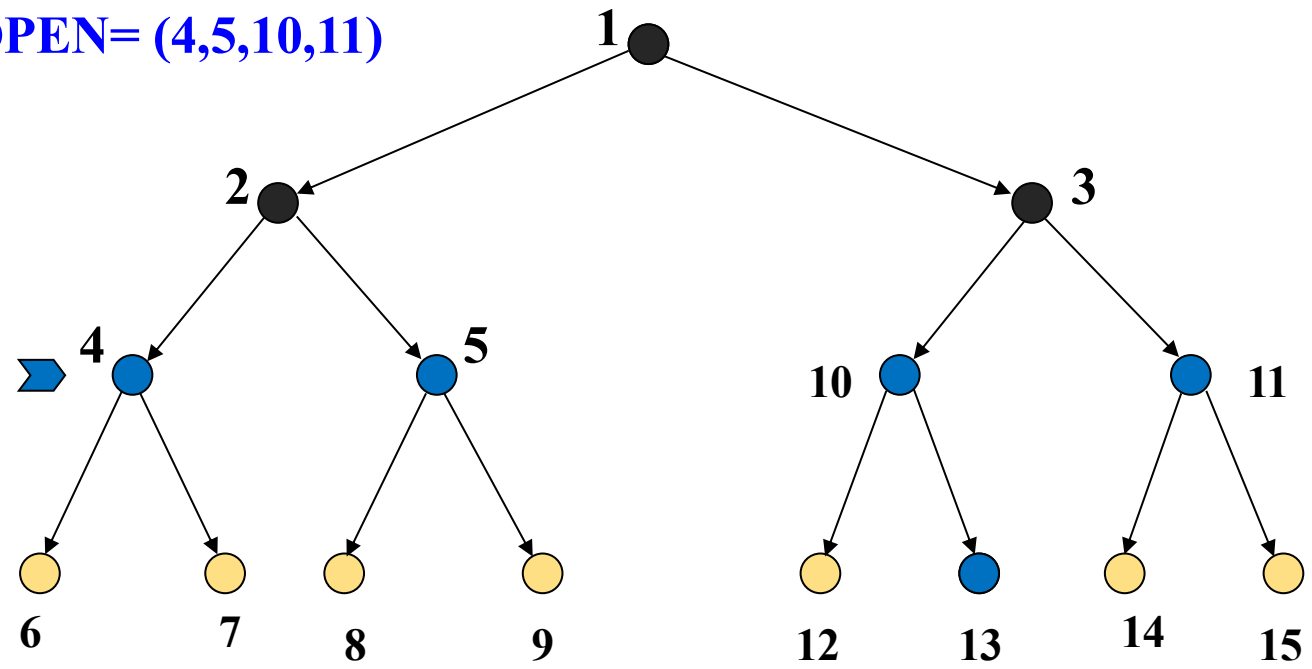


宽度优先搜索

新的节点被插入到 OPEN 表的 最后

CLOSED=(1,2,3)

OPEN= (4,5,10,11)

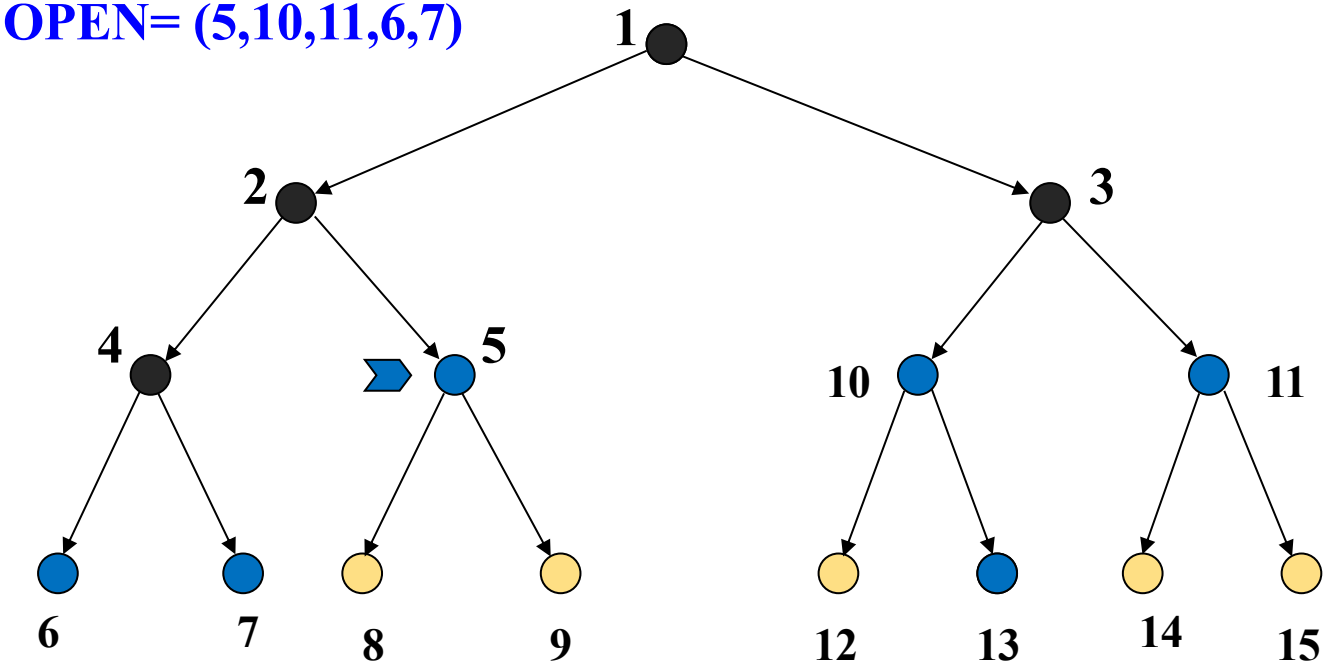


宽度优先搜索

新的节点被插入到 OPEN 表的 最后

CLOSED=(1,2,3,4)

OPEN= (5,10,11,6,7)

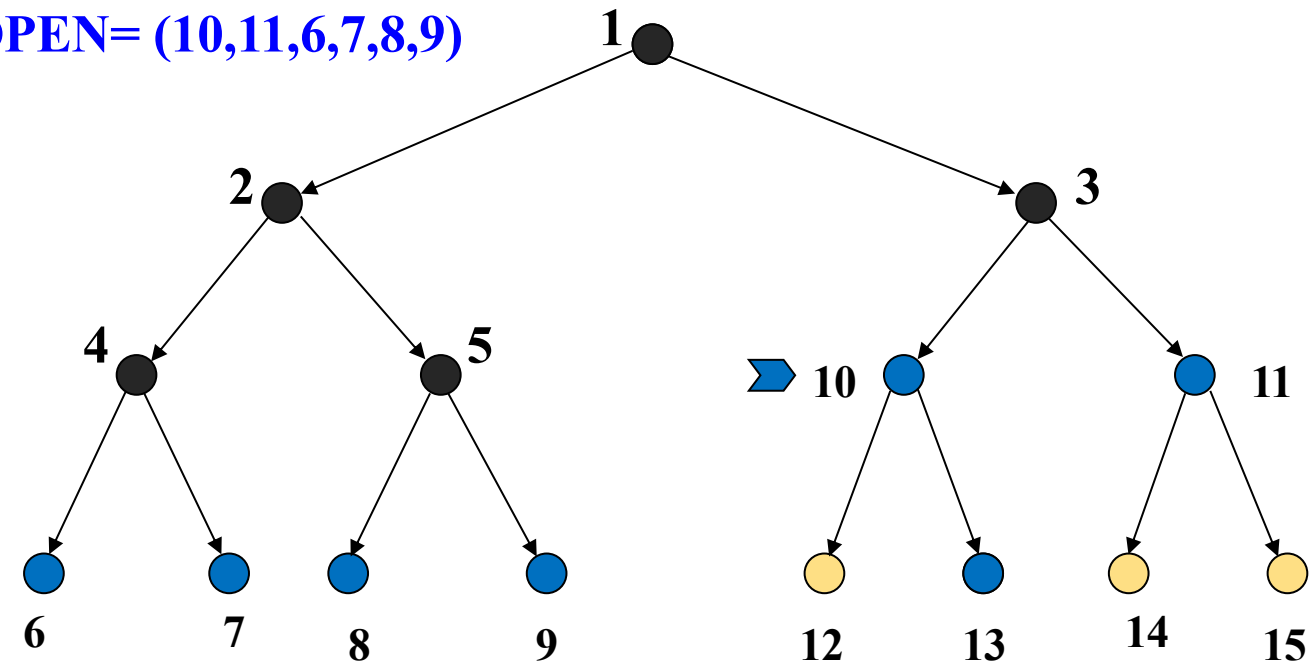


宽度优先搜索

新的节点被插入到 OPEN 表的 最后

CLOSED=(1,2,3,4,5)

OPEN= (10,11,6,7,8,9)

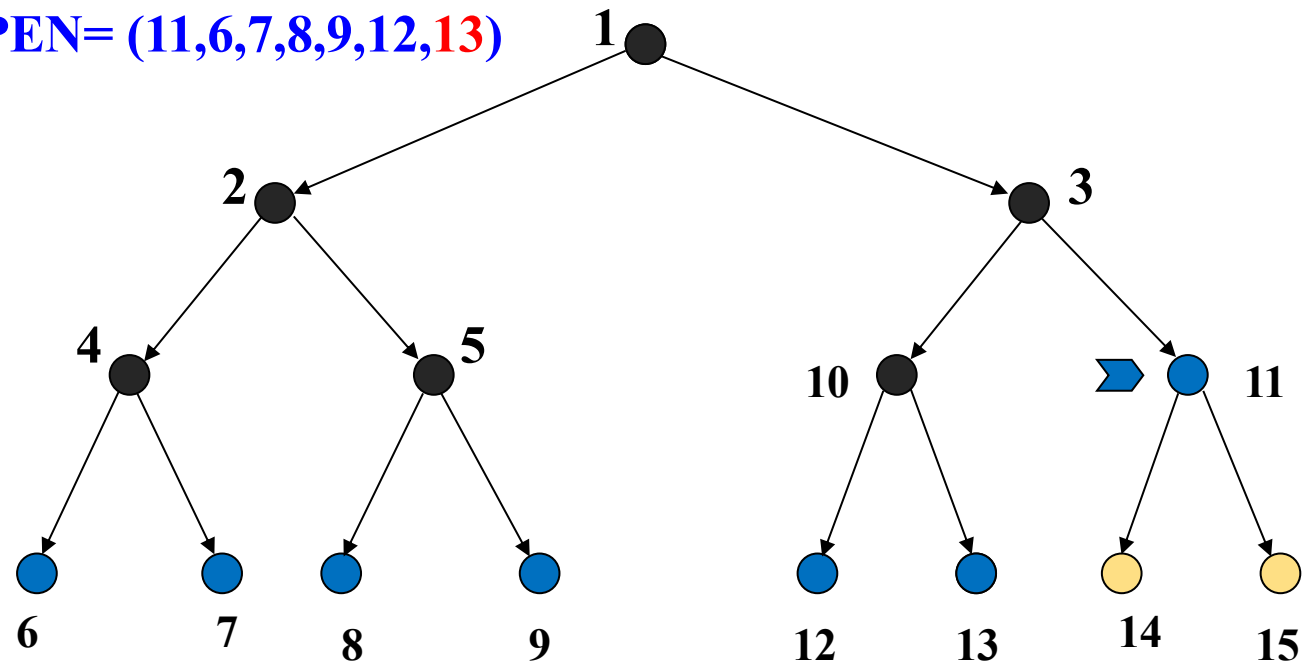


宽度优先搜索

新的节点被插入到 OPEN 表的 最后

CLOSED=(1,2,3,4,5,10)

OPEN= (11,6,7,8,9,12,13)



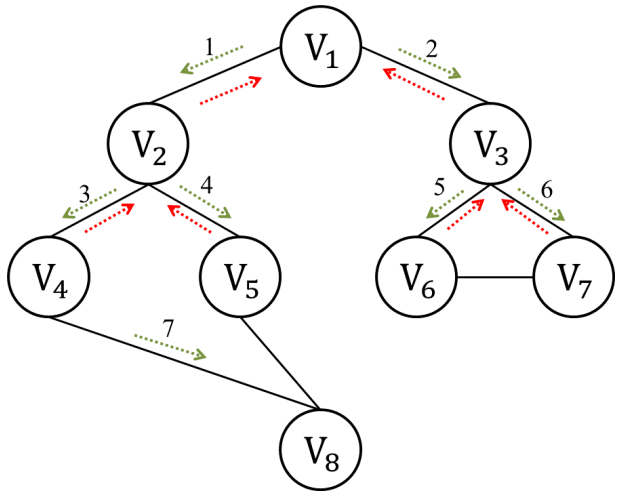
（2）宽度优先搜索---BFS

- **宽度优先搜索**也称为广度优先搜索。
- **基本思想**是：**优先扩展深度最浅的节点**。
  - 先扩展根节点，再扩展根节点的所有后继，然后再扩展它们的后继，依此类推。
  - 如果有**多个**节点深度是相同的，则按照事先约定的规则，从深度最浅的几个节点中选择一个，进行扩展。
- 一般地，在下一层的任何节点扩展之前，搜索树上本层深度的所有节点都应该已经扩展过。

宽度优先搜索，类似于树的层次遍历

所谓广度，就是一层一层的，向下遍历，层层堵截，如对下图3-4进行一次宽度优先遍历的话，我们的结果是：

$$V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \rightarrow V_6 \rightarrow V_7 \rightarrow V_8$$



宽度优先搜索示例图1

1、访问顶点V_i

2、访问V_i的所有未被访问的邻接点 V_1，V_2，…，V_k。

3、依次从这些邻接点（在步骤2中访问的顶点）出发，访问它们的所有未被访问的邻接点，依此类推，直到图中所有访问过的顶点的邻接点都被访问。

宽度优先搜索的实现方法

- 可以采用**FIFO** (First-In First-Out)的**队列**存储OPEN表。
  - **BFS**是将OPEN表中的节点按搜索树中节点**深度的增序**排序，**深度最浅**的节点排在最前面（**队头**），深度相同的节点可以任意排列。
  - 新节点（深度比其父节点深）总是加入到**队尾**，深度相同的节点可按某种事先约定的规则排列，这意味着浅层的老节点会在深层的新节点之前被扩展。

BFS算法

宽度优先搜索算法的过程如下：

（1）将初始节点S放入OPEN表的队头；

（2）若OPEN表为空，表示再也没有可扩展的节点，即未能找到问题的解，则算法结束；

（3）将OPEN表的**队头元素**（记为节点n）取出，放入CLOSED表中；

（4）若节点n是目标节点，则已求得问题的解，算法结束；

（5）若节点n不可扩展，即n没有后继节点，则转至步骤（2）。

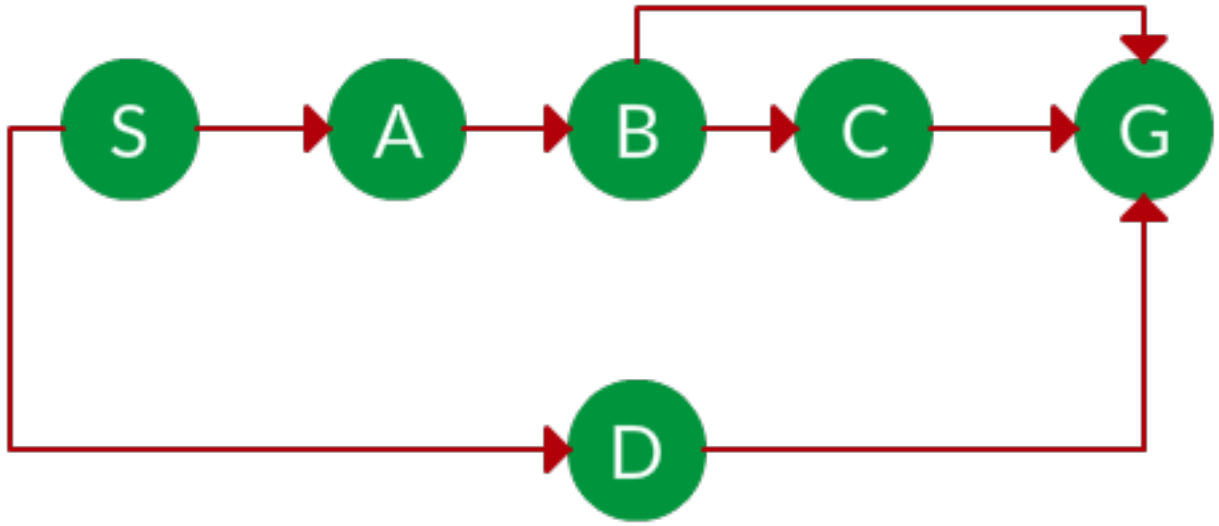
（6）扩展节点n，将其**所有未被访问过的子节点**依次放入**OPEN表的队尾**，并将这些子节点的前驱指针设为指向父节点n，然后转至步骤（2）。

BFS算法（伪代码）

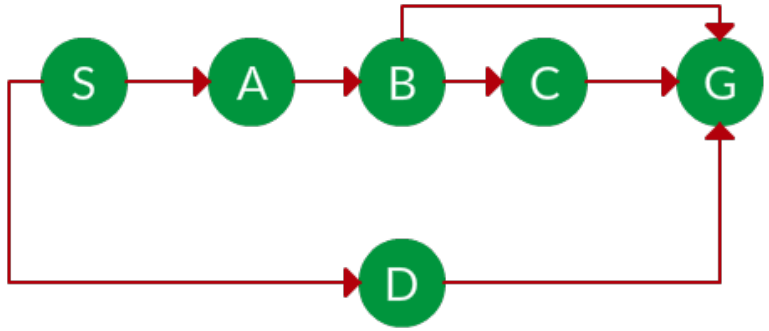
```
BFS 算法( S)：
        输入：  S，  起始节点
        返回：  解路径

         初始化Open表，Closed表，  将S节点加入Open表；
         v = Open表.pop（）  # 从Open表中取出第一个节点

         如果 节点v是目标节点，  则返回success
         将节点v加入Closed表；
         在节点v上执行所有可用的操作符，扩展得到新的节点，将子节点的parent设置为v；
         for each sub_node in v.child: # 遍历v的所有子节点
              if sub_node 未被访问过：
                   将其加入Open表

        if 找到解：
          从目标节点.parent反溯，得到解路径；
          返回解路径
```

BFS案例

- 问题： 如果在下图上运行，从节点 S 移动到节点 G， BFS 会找到哪种解决方案？
- 试着画出搜索树



step1



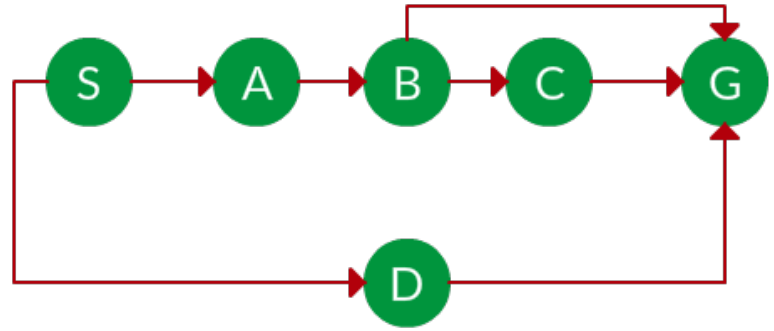
- BFS

最初，open表和Closed表都是空的

- Open

- Closed

step2



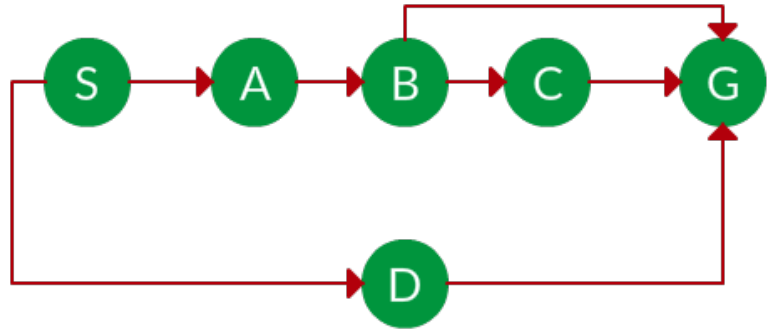
- 将初始节点放入open表

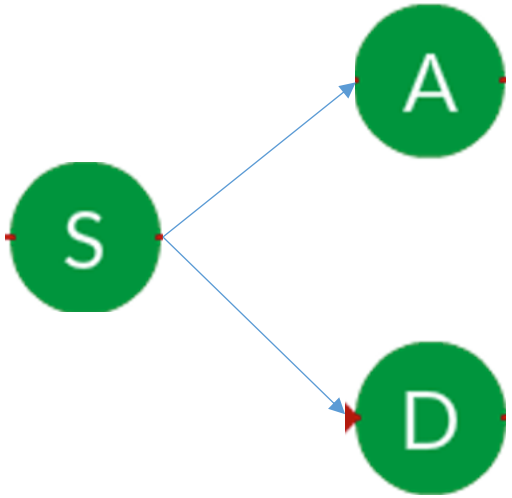
- Open

S			

- Closed

step3





- 从Open表取出第一个节点，判断节点S不是目标节点；存入Closed表
- 访问S的子节点，放入Open 表

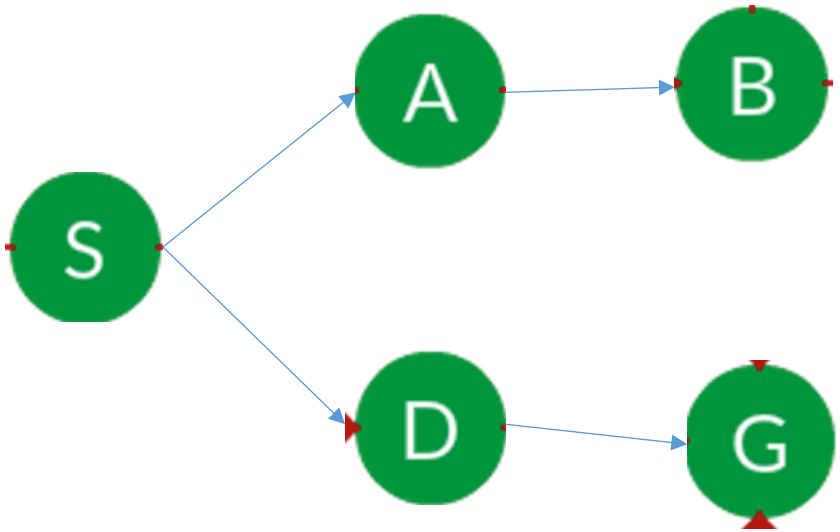
- Open

A	D		

- Closed

S			

step4



- 对A节点的处理和扩展
- Open

D	B		

- Closed

S	A		

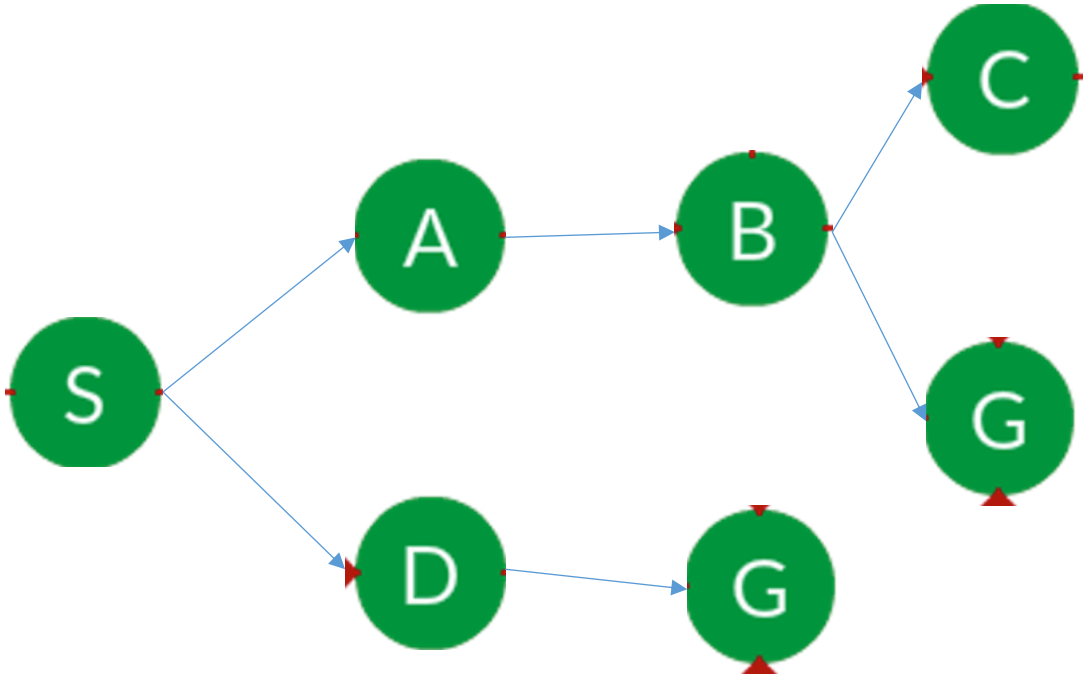
- **对D节点的处理和扩展**
- **Open**

B	G		

- **Closed**

S	A	D	

step5



- 对B节点的处理和扩展
- Open

G	C		

- Closed

S	A	D	B

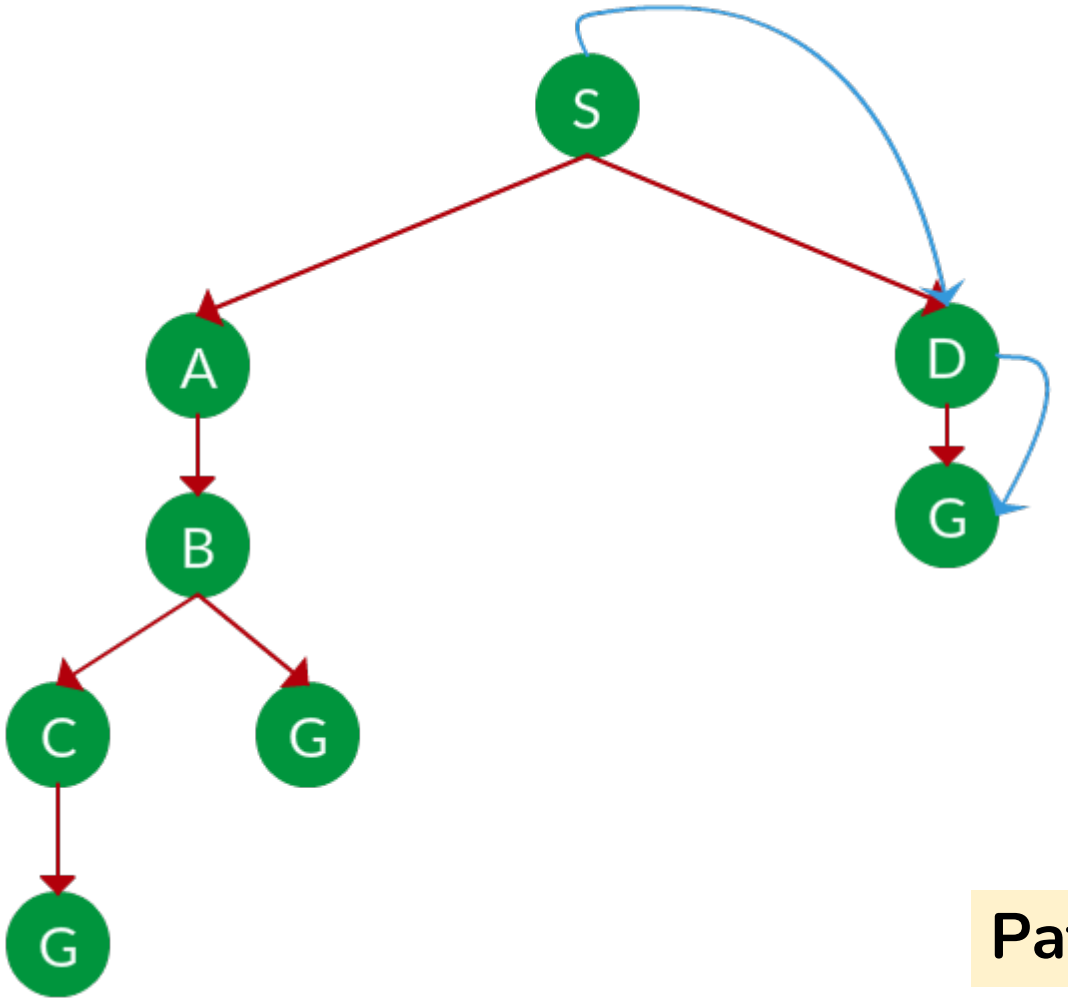
- **对G节点的处理和扩展**
- **Open**

C			

- **Closed**

S	A	D	B	G

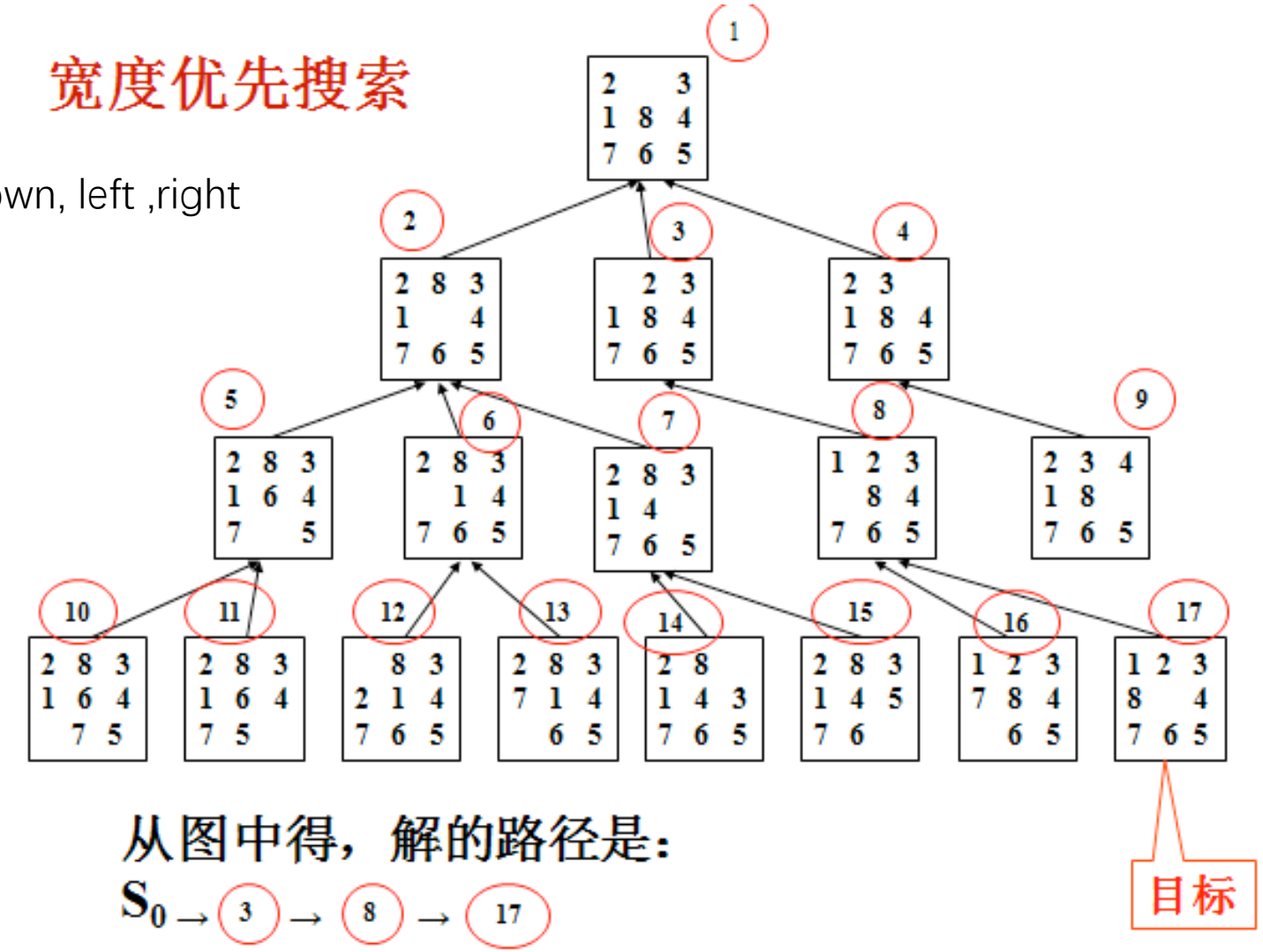
搜索树



Path: S -> D -> G

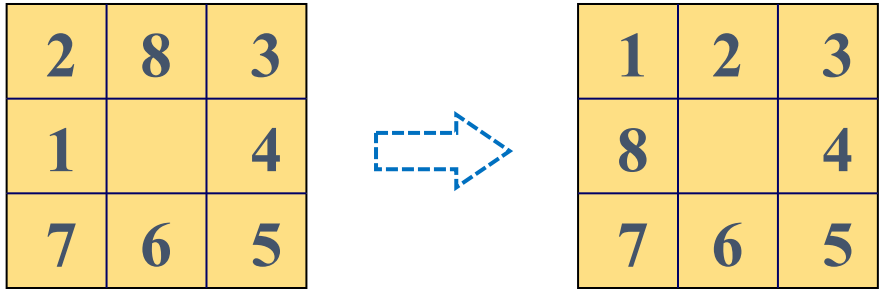
手动状态图求解法回顾

操作符顺序： up, down, left ,right



宽度优先搜索

八数码问题



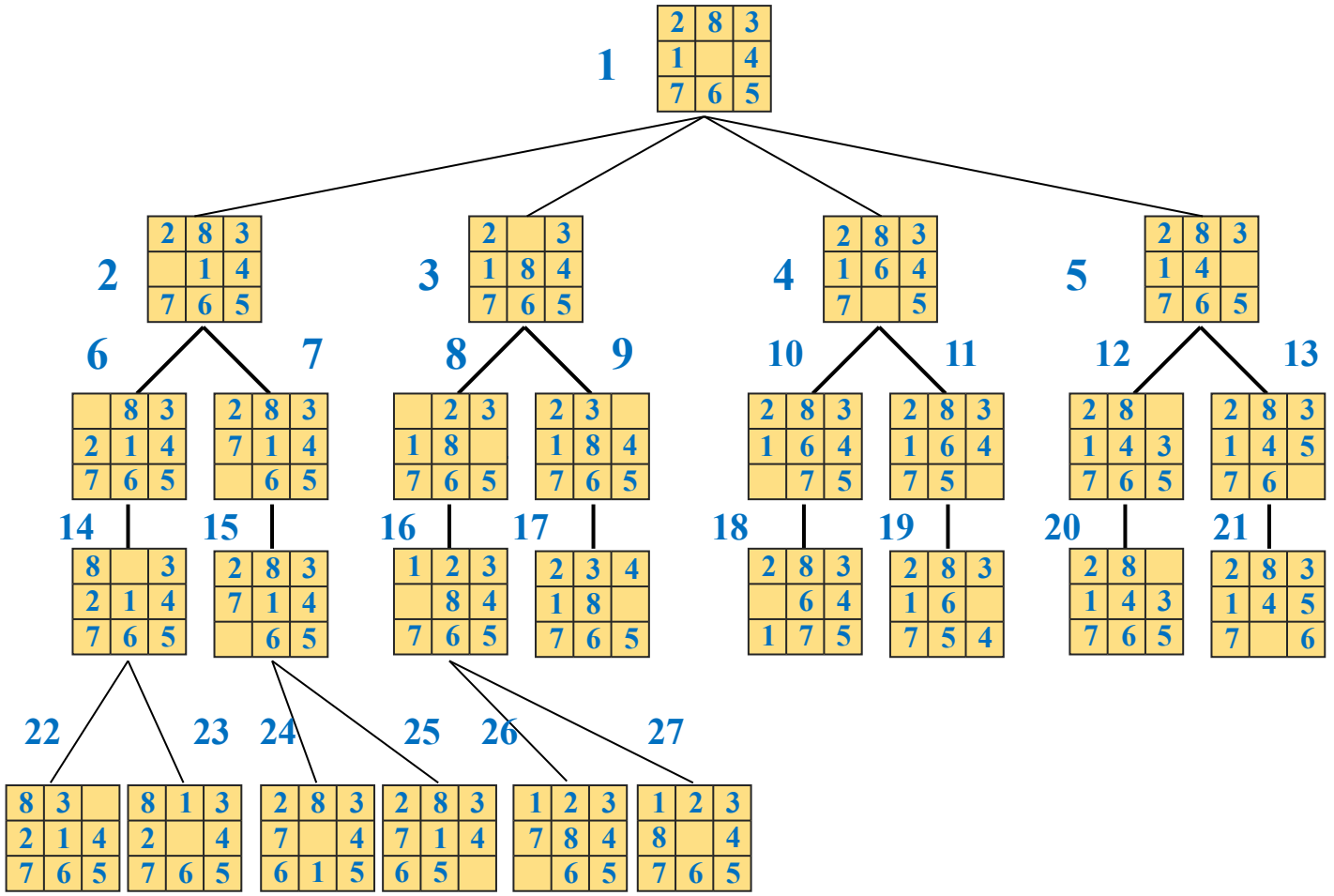
初始状态　　目标状态

规则

① 操作符顺序： 左、上、右、下。

② 不允许斜向移动。

宽度优先搜索



总结

- BFS是完备的（complete），即当问题有解时，一定能找到解。
- 若路径代价是节点深度的非递减函数，或者每步代价都相等，则宽度优先搜索一定能找到最优解，即BFS具有最优性。
- BFS是一个通用的、与问题无关的方法.
- 缺点：求解问题的效率较低。