第二章 搜索技术

- 内容提要：
- 状态空间法
- 问题归约法

问题描述

- 盲目搜索
- 启发式搜索
- 与或图搜索

问题求解

1

搜索技术

- 搜索技术是人工智能的基本技术之一，在人工智能各应用领域中被广泛地使用。

- 早期的人工智能程序与搜索技术联系就更为紧密，几乎所有的早期的人工智能程序（智力难题、棋类游戏、简单数学定理证明）都是以搜索为基础的。

- 现在，搜索技术渗透在各种人工智能系统中，可以说没有哪一种人工智能的应用不用搜索方法，在专家系统、自然语言理解、自动程序设计、模式识别、机器人学、信息检索和博弈都广泛使用搜索技术。

搜索技术

- 什么是搜索
  - 根据问题的实际情况不断寻找可利用的知识，构造出一条代价较少的推理路线，使问题得到圆满解决的过程称为搜索
  - 包括两个方面：

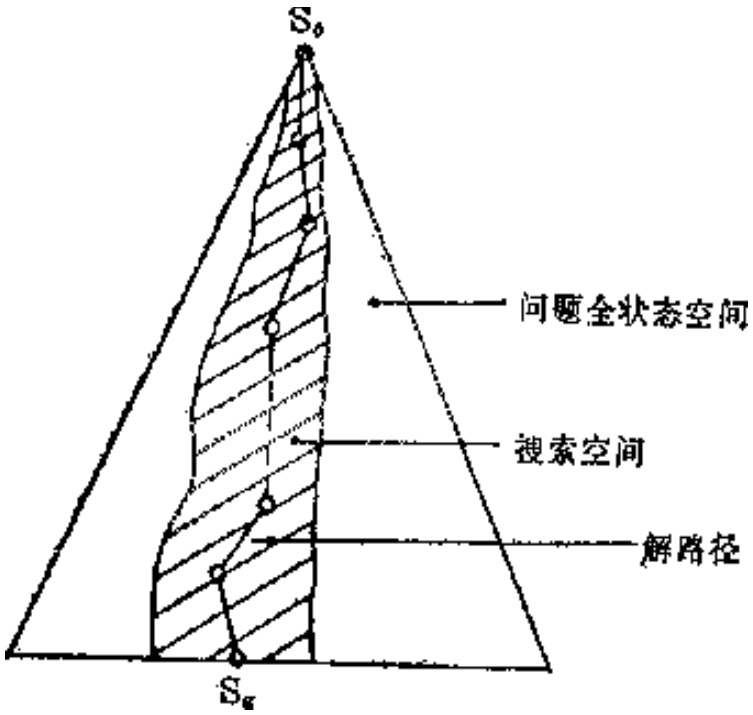
    ——— 找到从初始事实到问题最终答案的一条推理路径

    ——— 找到的这条路径在时间和空间上复杂度最小

搜索技术

搜索方法好的标准，一般认为有两个：

- (1) 搜索空间小
- (2) 解最佳



- 图搜索策略
- 一种在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**.

图的搜索过程

状态：(城市名)

算子：常德→益阳

益阳→常德

益阳↔汨罗

益阳↔宁乡

益阳↔娄底

...

深度优先搜索

图的搜索过程

 必须记住下一步还可以走哪些点

OPEN表(记录还没有扩展的点

用于存放刚生成的节点)

 必须记住哪些点走过了

CLOSED表(记录已经扩展的点

用于存放已经扩展或将要扩展的节点)

 必须记住从目标返回的路径

每个表示状态的节点结构中必须有指向父节点的指针

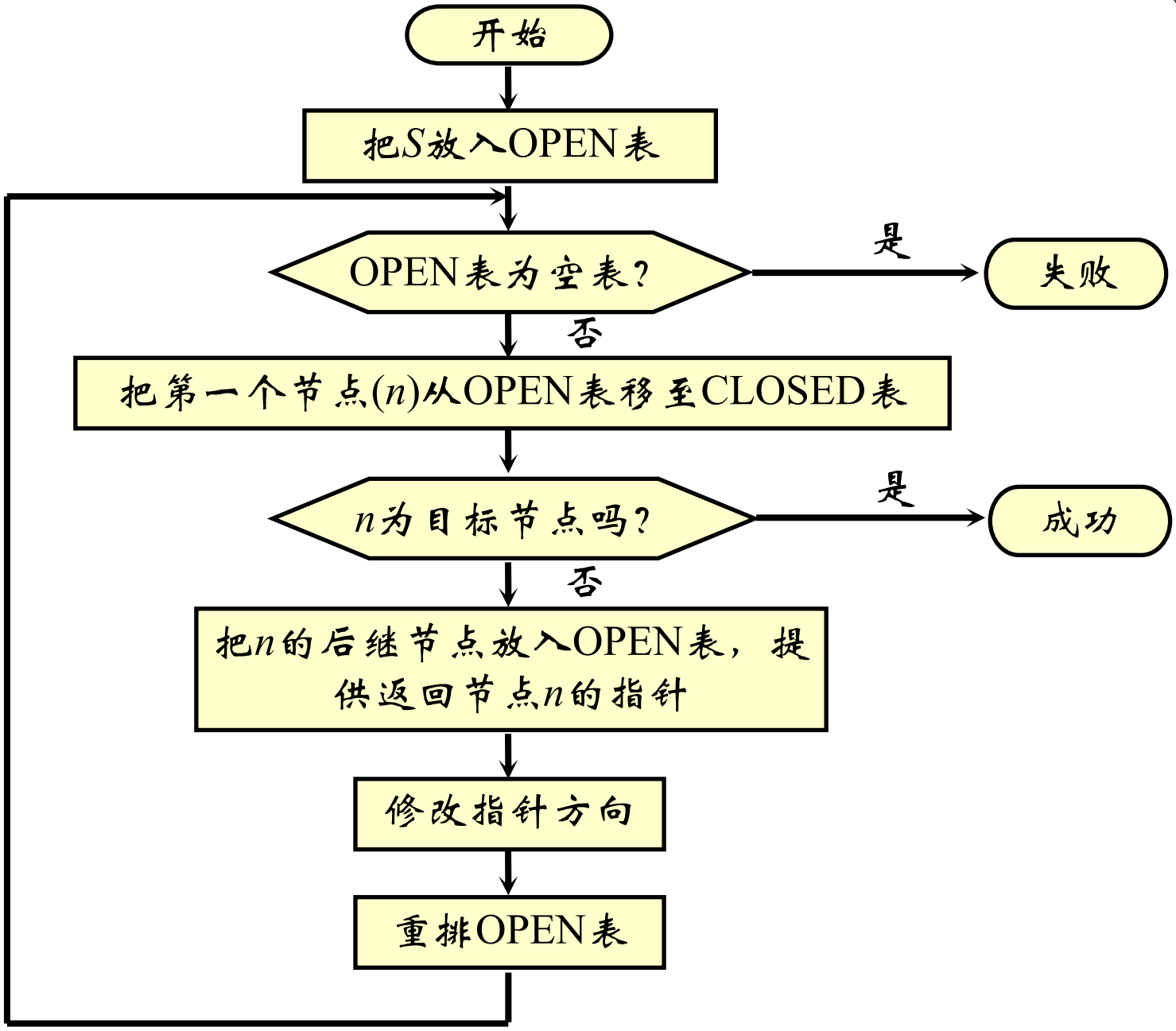
图的搜索过程

- 状态空间搜索的基本思想
- 先把问题的初始状态作为当前扩展节点对其进行扩展，生成一组子节点，然后检查问题的目标状态是否出现在这些子节点中。若出现，则搜索成功，找到了问题的解；若没出现，则再按照某种搜索策略从已生成的子节点中选择一个节点作为当前扩展节点。重复上述过程，直到目标状态出现在子节点中或者没有可供操作的节点为止。所谓对一个节点进行“扩展”是指对该节点用某个可用操作进行作用，生成该节点的一组子节点。

图搜索过程要点

- 搜索过程的要点如下：
  - 起始节点：对应于初始状态描述
  - 后继节点：把适用于某个节点状态描述的一些算符用来推算该节点而得到的新节点，称为该节点的后继节点
  - 指针：从每个后继节点返回指向其父辈节点
  - 考察各后继节点看是否为目标节点.
- 搜索过程扩展后继节点的次序
  - 如果搜索是以接近起始节点的程度（由节点之间连结弧线的数目来衡量）依次扩展节点，称为广（宽）度优先搜索
  - 如果搜索时首先扩展最新产生的节点，称为深度优先搜索

图的一般搜索策略


开始
把S放入OPEN表
OPEN表为空表？
是
失败
否
把第一个节点(n)从OPEN表移至CLOSED表
n为目标节点吗？
是
成功
否
把n的后继节点放入OPEN表，提供返回节点n的指针
修改指针方向
重排OPEN表

图搜索过程

(1) 把初始节点S0放入OPEN表，并建立目前只包含S0的图，记为G

(2) 检查OPEN表是否为空，若为空则问题无解，退出

(3) 把OPEN表的第一个节点取出放入CLOSED表，记该节点为n

(4) 考察n是否为目标节点. 如是，则问题得解，退出

(5) 扩展节点n，生成一组子节点. 把其中不是节点n先辈的那些节点记作集合M，并把这些子节点作为节点n的子节点加入G中

(6) 针对M中子节点的不同情况，分别进行处理

1. 对于那些未曾在G中出现过的M成员设置一个指向父节点(即n)的指针，并把它们放入OPEN表
2. 对于那些先前已在G中出现过的M成员，确定是否需要修改它指向父节点的指针
3. 对于那些先前已在G中出现并且已经扩展了的M成员，确定是否需要修改其后继节点指向父节点的指针

(7) 按某种搜索策略对OPEN表中的节点进行排序

(8) 转第(2)步

图搜索过程

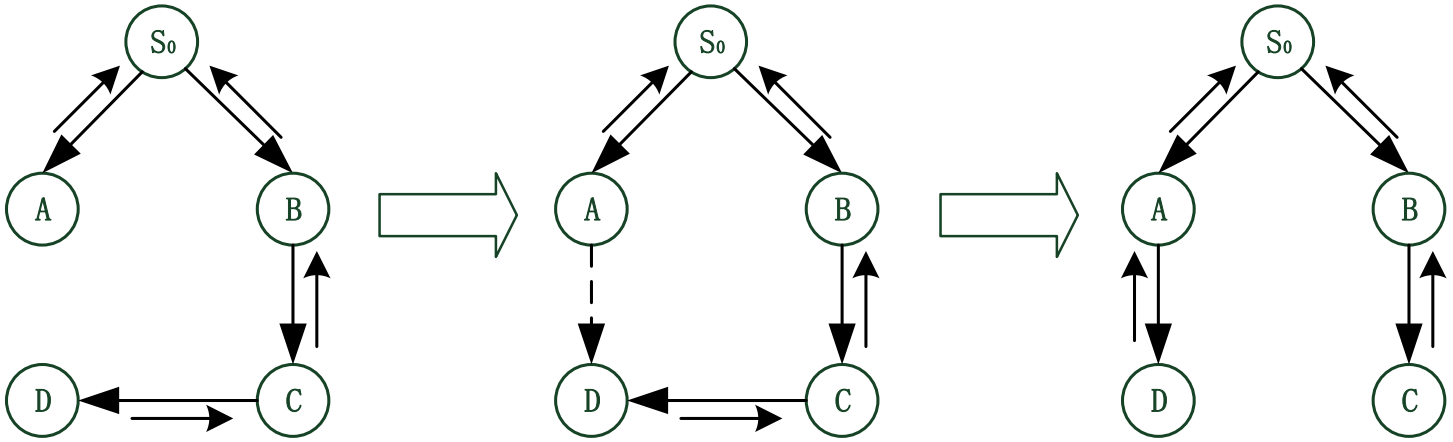
- 注:这是一个通用的搜索过程,后面讨论的状态空间各种搜索策略都是其特例.各种搜索策略的主要区别就是对OPEN表中节点排序准则不同
- 算法结束后，将生成一个图G，称为搜索图。同时由于每个节点都有一个指针指向父节点，这些指针指向的节点构成G的一个支撑树，称为搜索树。
- 从目标节点开始，将指针指向的状态回串起来，即找到一条解路径.

(5) 扩展节点n, 生成一组子节点. 把其中不是节点n先辈的那些节点记作集合M, 并把这些子节点作为节点n的子节点加入G中

(6) 针对M中子节点的不同情况, 分别进行处理

1. 对于那些未曾在G中出现过的M成员设置一个指向父节点(即n)的指针, 并把它们放入OPEN表
2. 对于那些先前已在G中出现过的M成员, 确定是否需要修改它指向父节点的指针
3. 对于那些先前已在G中出现并且已经扩展了的M成员, 确定是否需要修改其后继节点指向父节点的指针

- **说明:**
- (1) n的先辈节点。
- (2) 对已存在于OPEN表的节点(如果有的话)删除之; 但删除之前要比较其返回初始节点的新路径与原路径, 如果新路径“短”, 则修改这些节点在OPEN表中的原返回指针, 使其沿新路返回。
- (3) 对已存在于CLOSED表的节点(如果有的话), 做与(2)同样的处理, 并且再将其移出CLOSED表, 放入OPEN表重新扩展(为了重新计算代价)。
- (4) 对其余子节点配上指向n的返回指针后放入OPEN表中某处。



修改返回指针示例

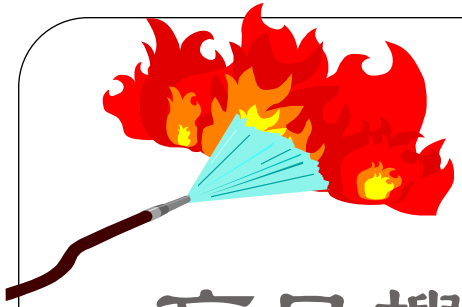
进一步说明：

(1) 步6中修改返回指针的原因是，因为这些节点又被第二次生成，所以它们返回初始节点的路径已有两条，但这两条路径的“长度”可能不同。 那么，当新路短时自然要走新路。

(2) 这里对路径的长短是按路径上的节点数来衡量的，后面我们将会看到路径的长短也可以其“代价”(如距离、费用、时间等)衡量。

图搜索方法分析

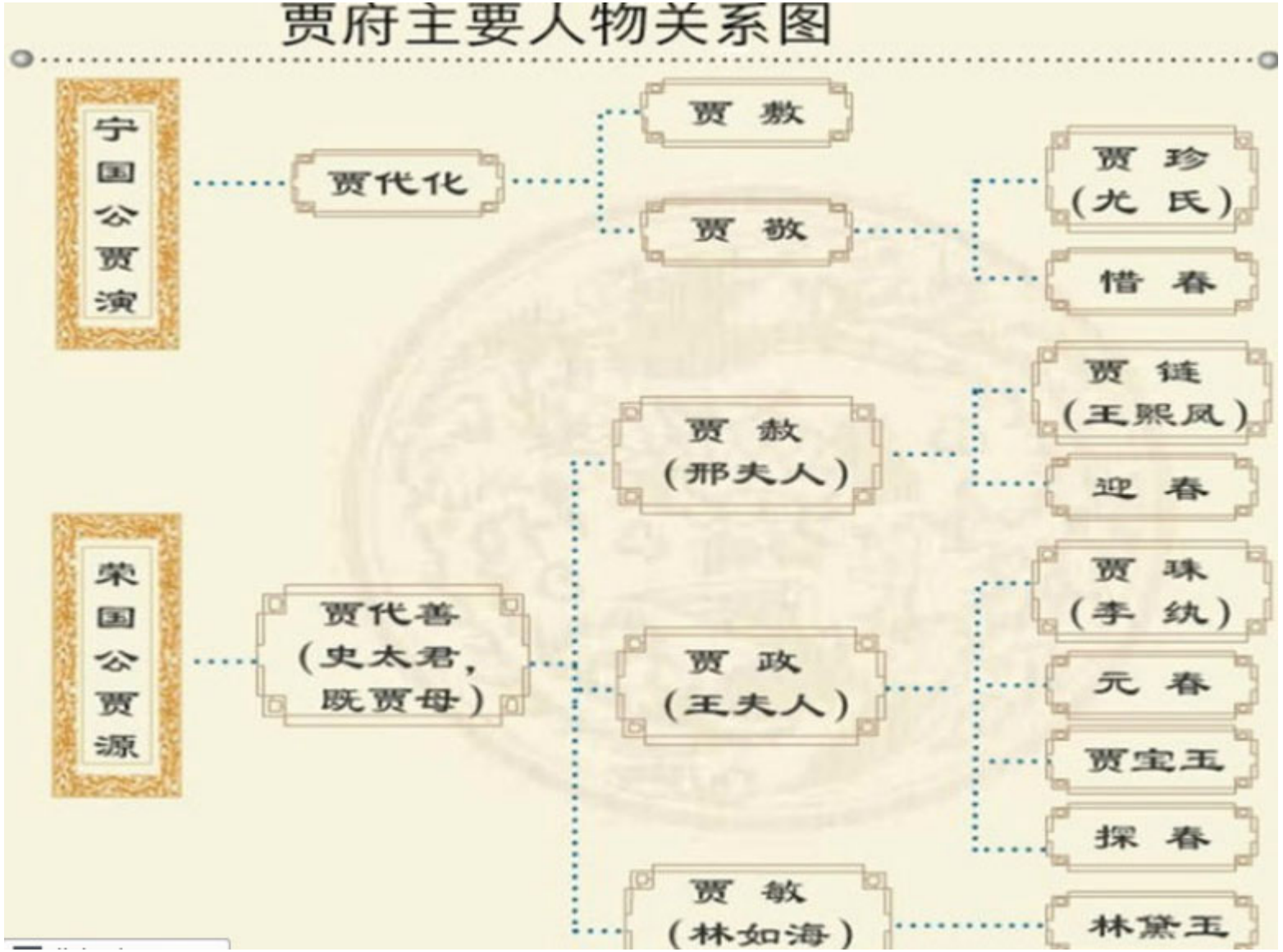
- 每当被选作扩展的节点为目标节点时，这一过程就宣告成功结束。这时，能够重现从起始节点到目标节点的这条成功路径，其办法是从目标节点按指针向S返回追溯。
- 当搜索树不再剩有未被扩展的端节点时，过程就以失败告终(某些节点最终可能没有后继节点，所以OPEN表可能最后变成空表)。在失败终止的情况下，从起始节点出发，一定达不到目标节点。
- 图搜索过程的第7步对OPEN表上的节点进行排序，以便能够从中选出一个“最好”的节点作为第3步扩展用。
- 这种排序可以是任意的即盲目的(属于盲目搜索)，也可以用以后要讨论的各种启发思想或其它准则为依据(属于启发式搜索)。

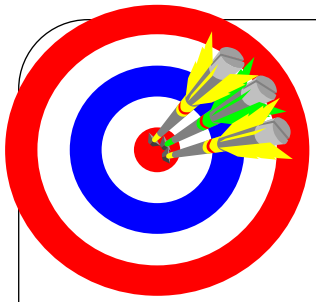


盲目搜索

- 概念
  - ◇没有启发信息的一种搜索形式(按预定的控制策略进行搜索，在搜索过程中获得的中间信息不用来改进控制策略)
  - ◇一般只适用于求解比较简单的问题
- 特点
  - ◇不需重排OPEN表
- 种类
  - ◇宽度优先
  - ◇深度优先
  - ◇等代价搜索

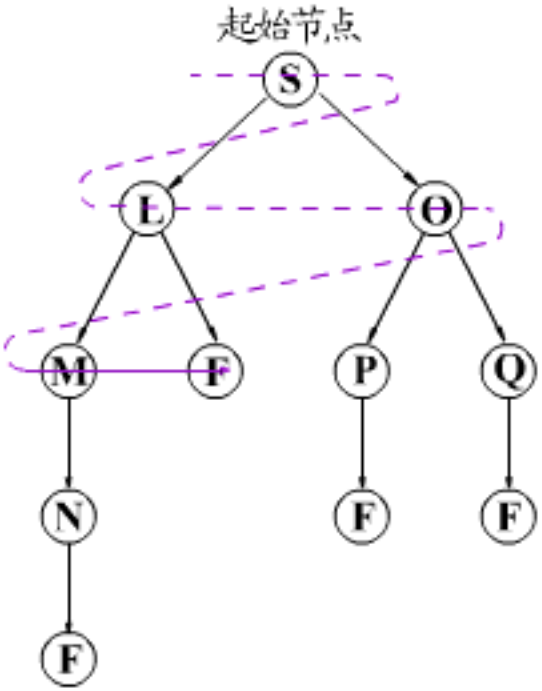
盲目搜索


贾府主要人物关系图
宁国公贾演
贾代化
贾敷
贾敬
贾珍（尤氏）
惜春
荣国公贾源
贾代善（史太君，既贾母）
贾赦（邢夫人）
贾政（王夫人）
贾敏（林如海）
贾琏（王熙凤）
迎春
贾珠（李纨）
元春
贾宝玉
探春
林黛玉

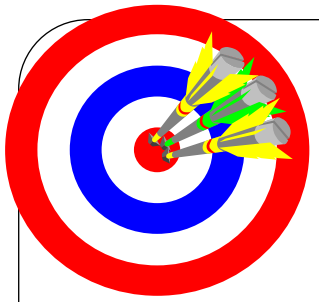


宽度优先搜索策略

- 宽度优先搜索(breadth-first search)的定义：如果搜索是以接近起始节点的程度依次扩展节点的，那么这种搜索就叫做宽度优先搜索(breadth-first search).
- 这种搜索是逐层进行的；在对下一层的任一节点进行搜索之前，必须搜索完本层的所有节点



宽度优先搜索示意图



宽度优先搜索策略算法

(1) 把初始节点S。放入OPEN表；

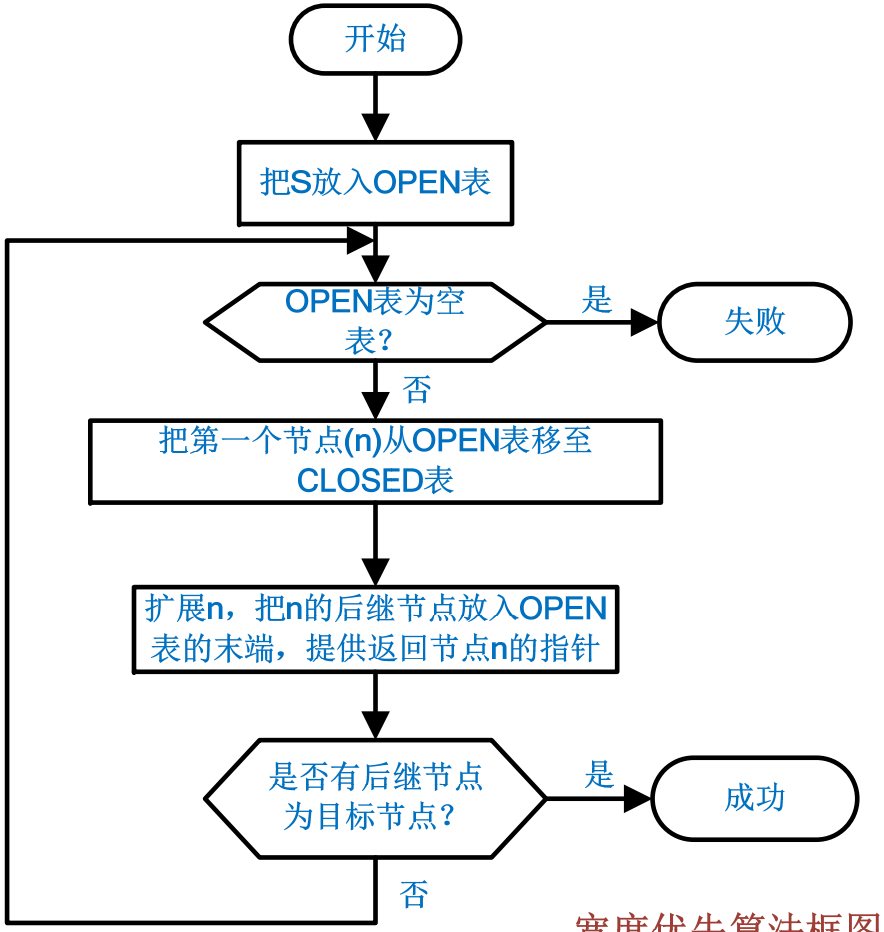
(2) 如果OPEN表为空，则问题无解，退出；

(3) 把OPEN表的第一个节点（记为n）取出放入CLOSED表；

(5) 若节点n不可扩展，则转第(2)步；

(6) 扩展节点n，将其子节点放入OPEN表的末端，并为每一个子节点都配置指向父节点的指针，然后转第(2)步；

宽度优先搜索策略框图



宽度优先算法框图

八数码难题

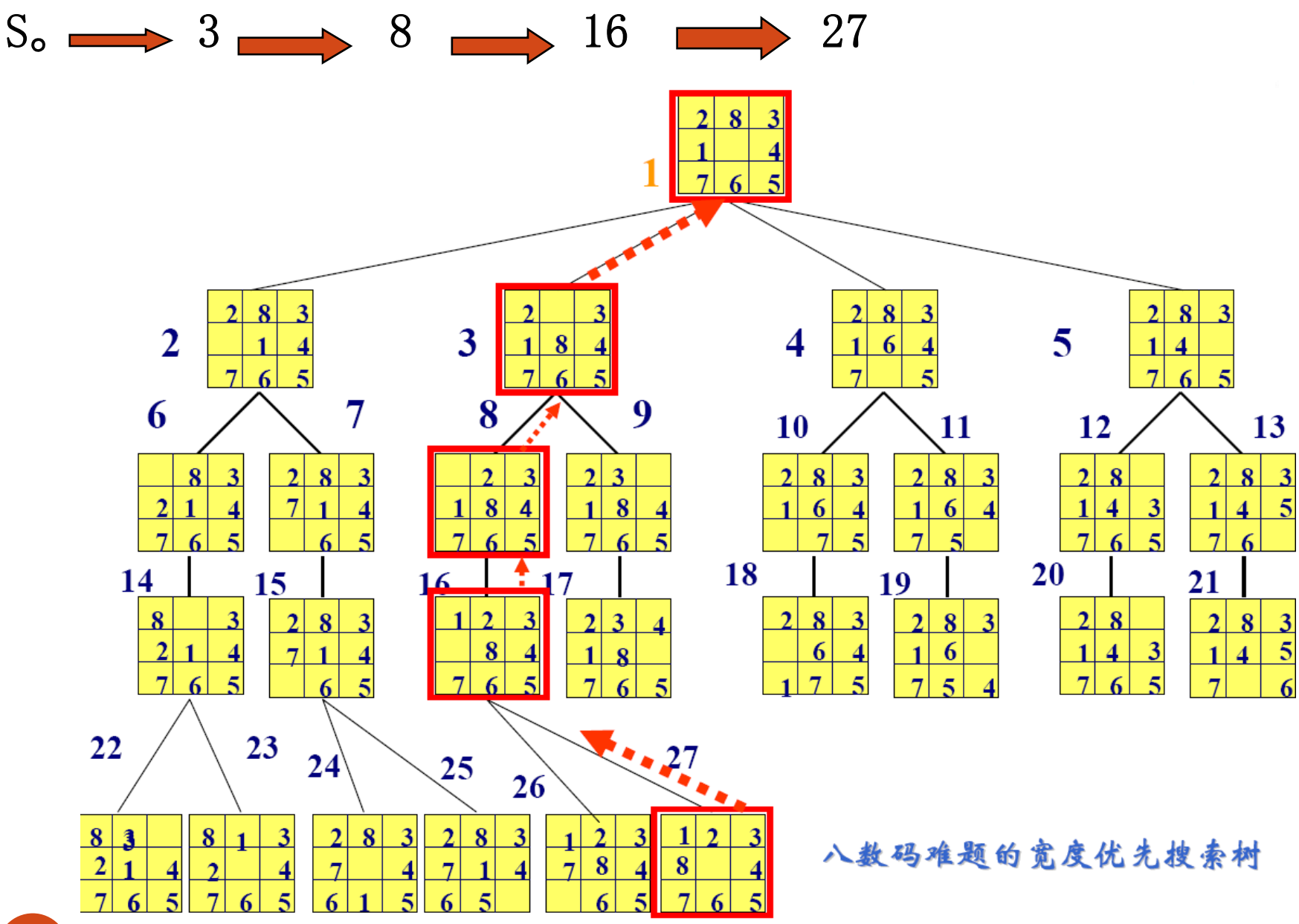
___8--puzzle problem

（初始状态）　　（目标状态）

要求寻找从初始状态到目标状态的路径。

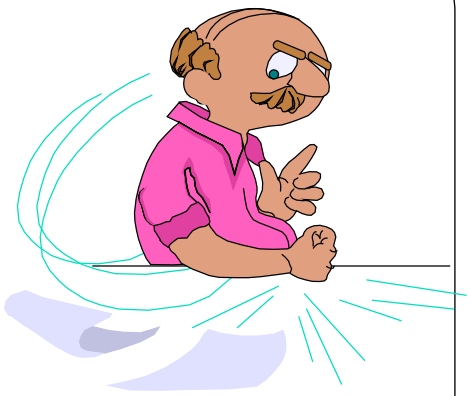
解：　应用宽度优先搜索，可得到如下图所示的搜索树，并可得到解的路径是：

$S_0 \rightarrow 3 \rightarrow 8 \rightarrow 16 \rightarrow 27$


S。
3
8
16
27
八数码难题的宽度优先搜索树

深度优先搜索

(depth-first search)



- 定义

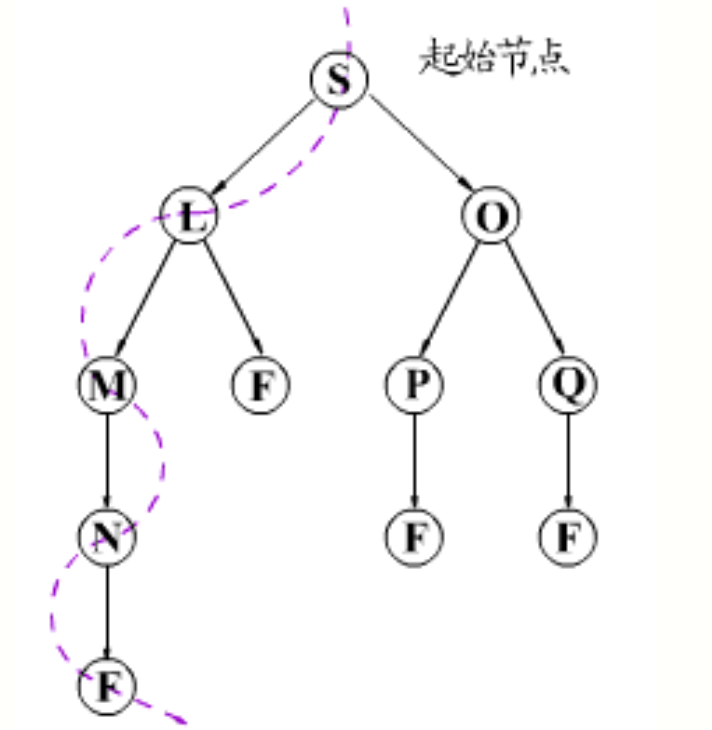
  ◇首先扩展最新产生的（即最深的）节点。

  ◇深度相等的节点可以任意排列。

- 特点

  ◇首先，扩展最深的节点的结果使得搜索沿着状态空间某条单一的 首先路径从起始节点向下进行下去；

  ◇仅当搜索到达一个没有后裔的状态时，才考虑另一条替代的路径。



深度优先搜索示意图

算法

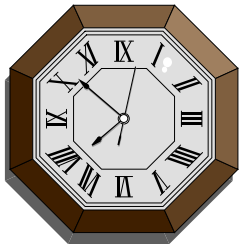
□ ◇防止搜索过程沿着无益的路径扩展下去，往往给出一个节点扩展最大深度 ——深度界限 。任何节点如果达到了深度界限，那么都将把它们作为没有后继节点来处理。

定义-节点的深度：

(1) 起始节点的深度为0。

(2) 任何其他节点的深度等于其父辈节点的深度加1。

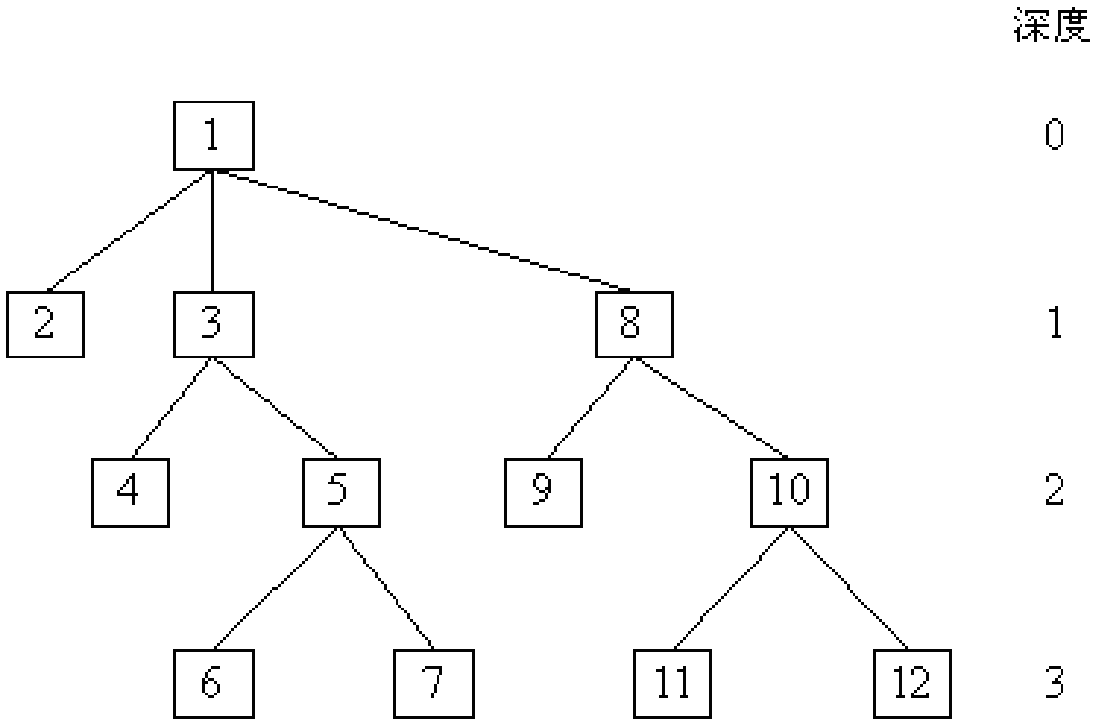
◇与宽度优先搜索算法最根本的不同：将扩展的后继节点放在OPEN表的前端。



算法的具体过程

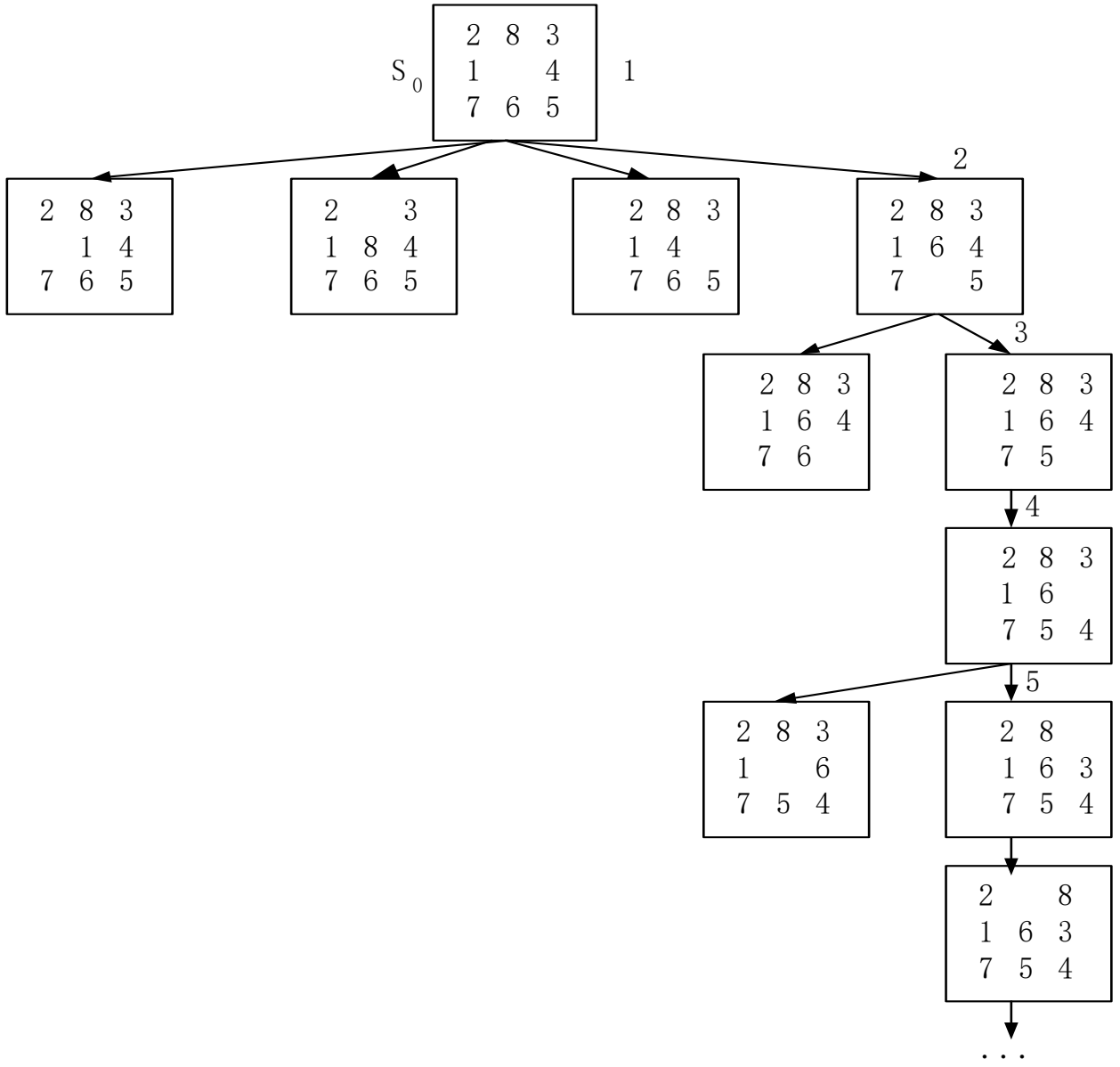
1. 把起始节点 S放到未扩展节点 OPEN表中。如果此节点为一目标节点，则得到一个解。
2. 如果 OPEN为一空表，则失败退出。
3. 把第一个节点（节点n）从 OPEN表移到CLOSED表。
4. 如果节点 n的深度等于最大深度，则转向(2)
5. 扩展节点n，产生其全部后裔，并把它们放入OPEN表的前端，如果没有后裔，则转向(2)
6. 如果后继节点中有任一个为目标节点，则求得一个解，成功退出，否则，转向(2)

深度优先搜索基本思想



深度优先搜索的性质

- 一般不能保证找到最优解
- 当深度限制不合理时，可能找不到解，可以将算法改为可变深度限制
- 最坏情况时，搜索空间等同于穷举


S0
2 8 3
1 4
7 6 5
1
2 8 3
1 4
7 6 5
2 3
1 8 4
7 6 5
2 8 3
1 4
7 6 5
2
2 8 3
1 6 4
7 5
3
2 8 3
1 6 4
7 6
2 8 3
1 6 4
7 5
4
2 8 3
1 6
7 5 4
5
2 8 3
1 6
7 5 4
2 8
1 6 3
7 5 4
2 8
1 6 3
7 5 4
. . .

有界深度优先搜索

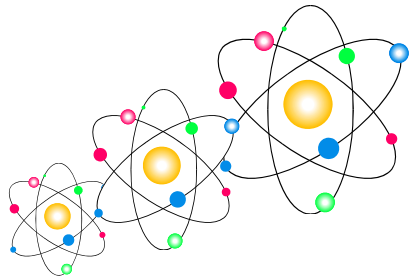
- 基本思想：

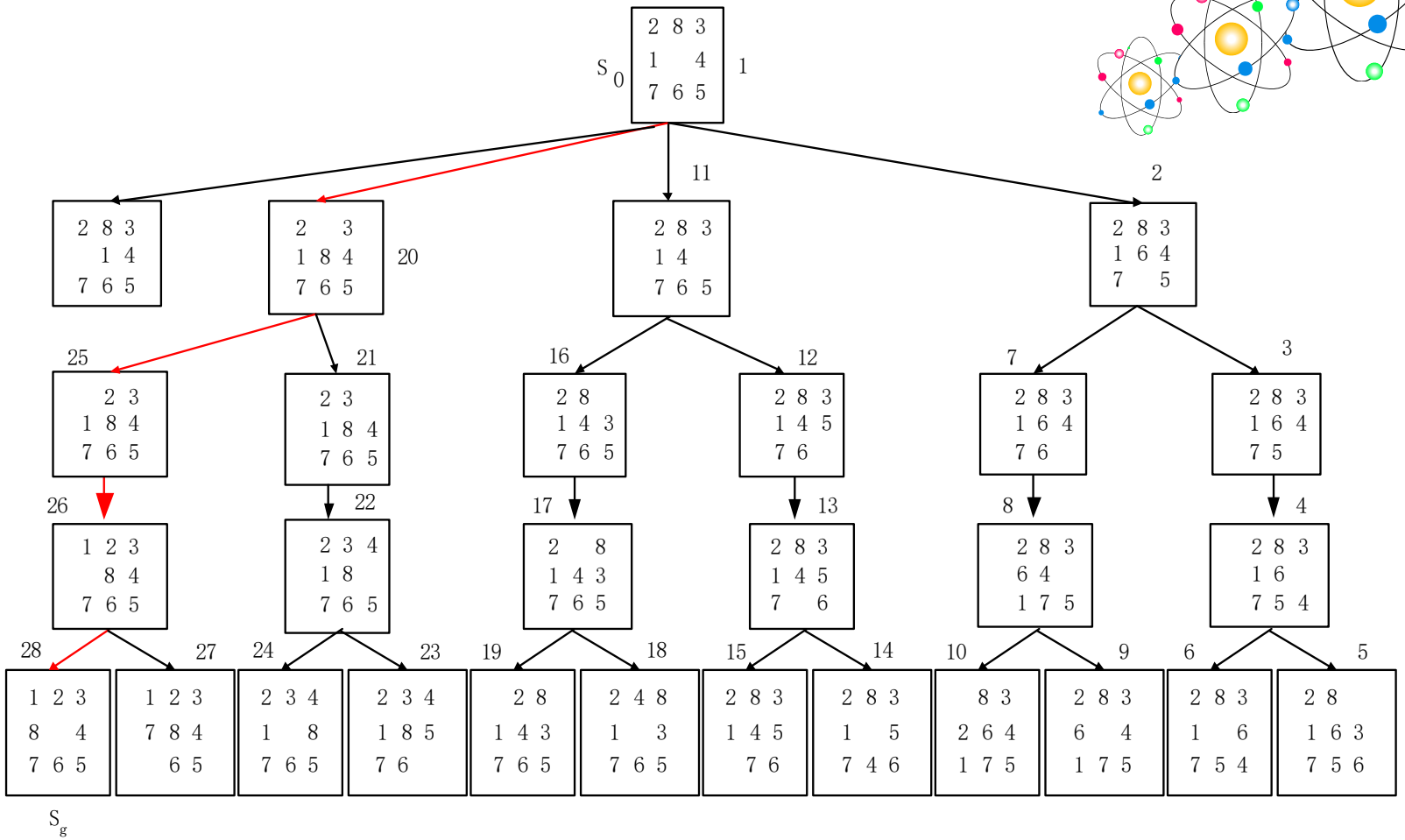
  对深度优先搜索引入搜索深度的界限（设为d_m），当搜索深度达到了深度界限，而仍未出现目标节点时，就换一个分支进行搜索。

有界深度优先搜索

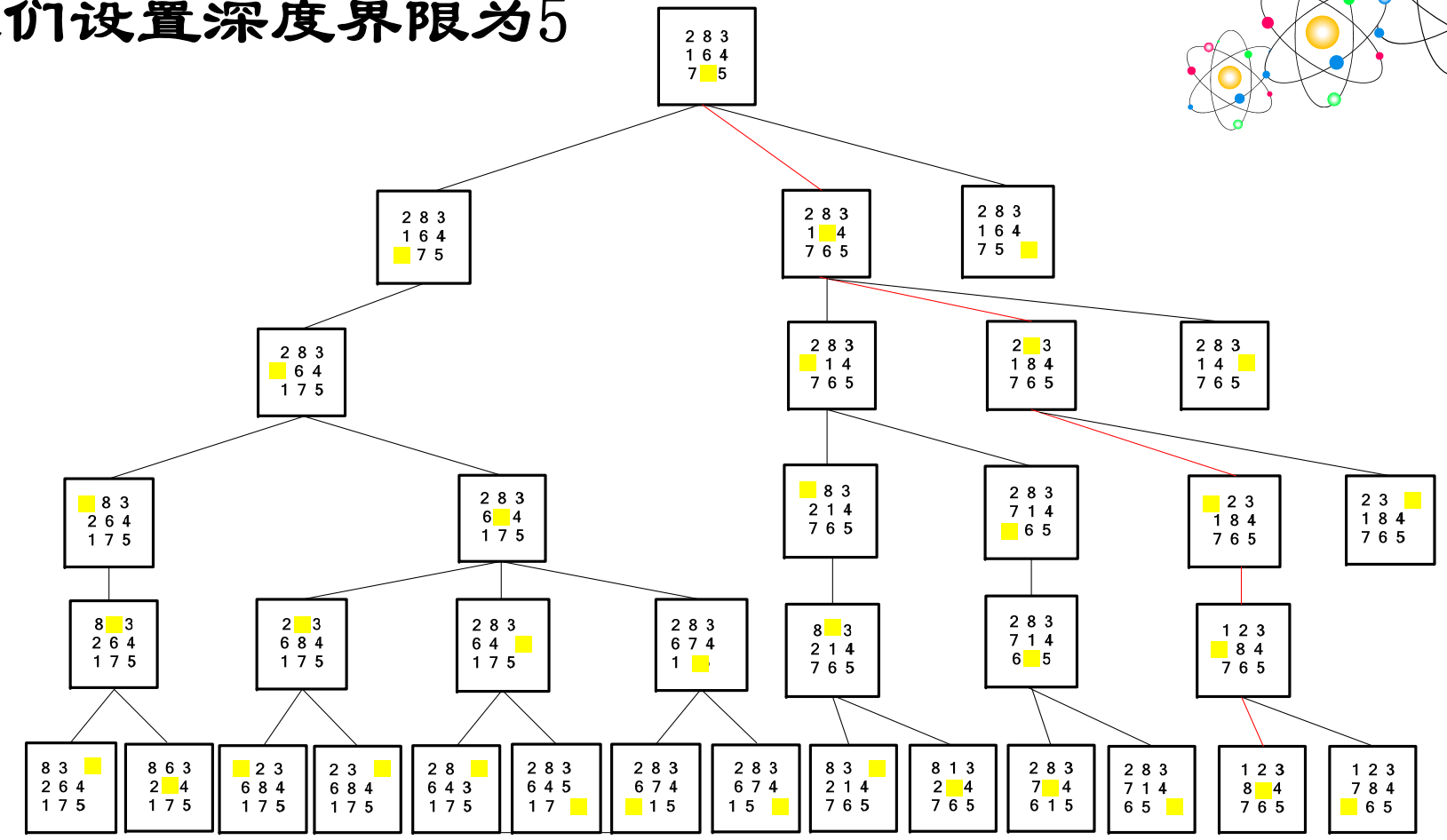
- 如果问题有解，且其路径长度≤d_m，则上述搜索过程一定能求得解。但是，若解的路径长度>d_m，则上述搜索过程就得不到解。这说明在有界深度优先搜索中，深度界限的选择是很重要的。
- 要恰当地给出d_m的值是比较困难的。即使能求出解，它也不一定是最优解。

例：按深度优先搜索生成的八数码难题搜索树，我们设置深度界限为4





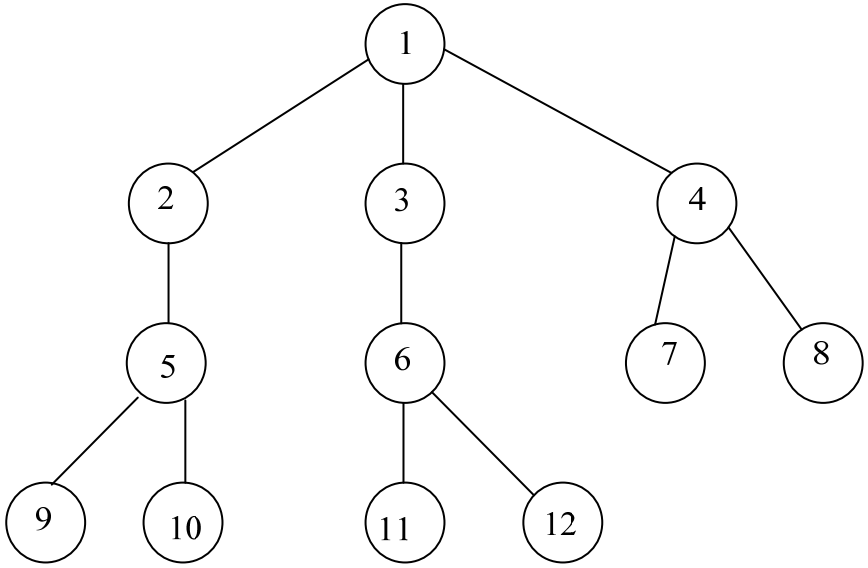
例：按深度优先搜索生成的八数码难题搜索树，我们设置深度界限为5



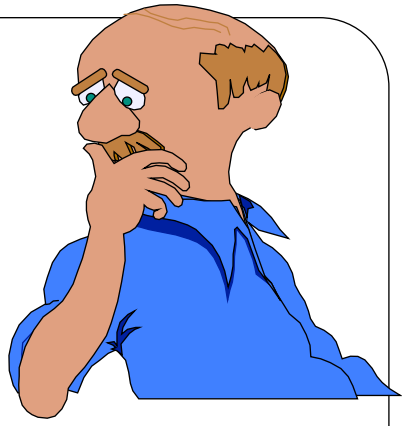
练习

(1)请根据深度优先搜索策略列出图中树的节点的访问序列(在目标节点为节点8，所有情况中都选择最左分支优先访问)。

(2)请写出根据深度优先搜索策略搜索时从初始状态迭代至搜索结束的过程中OPEN表和CLOSED表所存储的内容。



等代价搜索



- 概念

  ◇宽度优先搜索的一种推广。

  ◇不是沿着等长度路径断层进行扩展，而是沿着等代价路径断层进行扩展。

  ◇搜索树中每条连接弧线上的有关代价，表示时间、距离等花费。

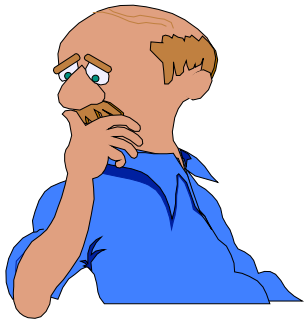
- 等代价搜索中的几个记号

  ◇起始节点记为 S；

  ◇从节点i到它的后继节点j的连接弧线代价记为 c(i, j)

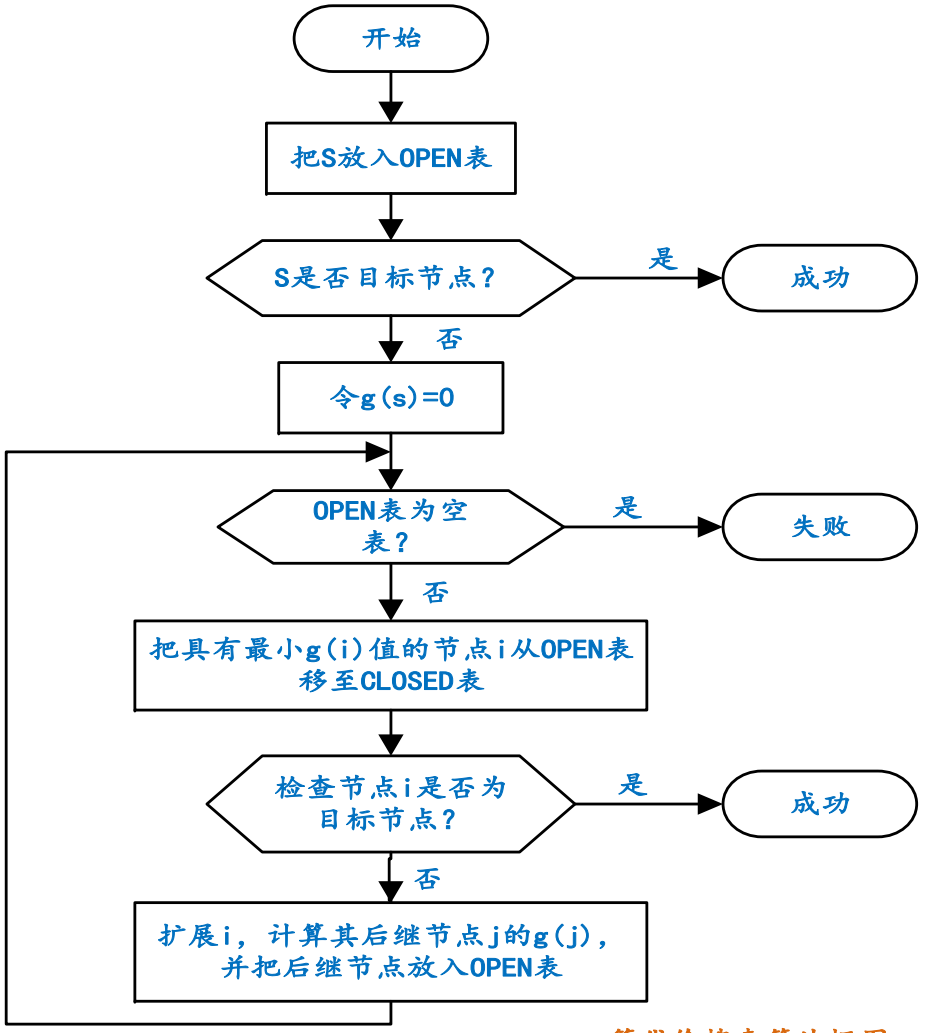
  ◇从起始节点S到任一节点i的路径代价记为g(i) 。

等代价搜索算法



1. 把起始节点S放到未扩展节点表OPEN中。如果此起始节点为一目标节点,则求得一个解；否则令g(S)=0;
2. 如果OPEN是个空表，则没有解而失败退出;
3. 从OPEN表中选择一个节点i，使其g(i)为最小.如果有几个节点都合格，那么就要选择一个目标节点作为节点i(要是有目标节点的话)；否则,就从中选一个作为节点i。把节点i从OPEN表移至扩展节点表CLOSED中;
4. 如果节点i为目标节点，则求得一个解;
5. 扩展节点i。如果没有后继节点，则转向第(2)步;
6. 对于节点i的每个后继节点j,计算 g(j)=g(i)+c(i,j)，并把所有后继节点j放进OPEN表. 提供回到节点i的指针;
7. 转向第(2)步。

等代价搜索算法框图



等代价搜索算法框图

代价树的广度优先搜索

- 基本思想：

  每次从OPEN表中选择节点往CLOSED表传送时，总是选择其中代价最小的节点。也就是说，OPEN表中的节点在任一时刻都是按其代价从小到大排序的。代价小的节点排在前面，代价大的节点排在后面。

- 如果问题有解，代价树的广度优先搜索一定可以求得解，并且求出的是最优解。

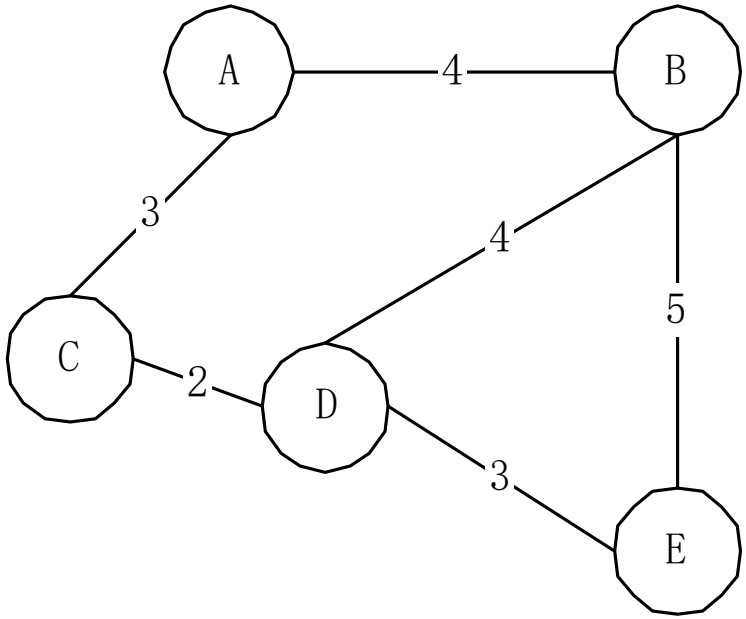
代价树的广度优先搜索

1. 把初始节点S_0放入OPEN表，令$g(S_0)=0$。
2. 如果OPEN表为空，则问题无解，退出。
3. 把OPEN表的第一个节点（记为节点n）取出放入CLOSED表。
4. 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点n不可扩展，则转第2步。
6. 扩展节点n，为每一个子节点都配置指向父节点的指针，计算各子节点的代价，并将各子节点放入OPEN表中。按各节点的代价对OPEN表中的全部节点进行排序（按从小到大的顺序），然后转第2步。

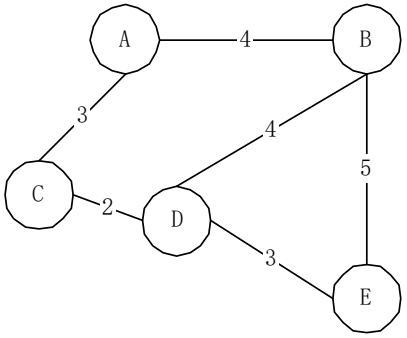
代价树的深度优先搜索

- 搜索过程：
  1. 把初始节点S_0放入OPEN表，令$g(S_0)=0$。
  2. 如果OPEN表为空，则问题无解，退出。
  3. 把OPEN表的第一个节点（记为节点n）取出放入CLOSED表。
  4. 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
  5. 若节点n不可扩展，则转第2步。
  6. 扩展节点n，将其子节点按代价从小到大的顺序放到OPEN表中的前端，并为每一个子节点都配置指向父节点的指针，然后转第2步。

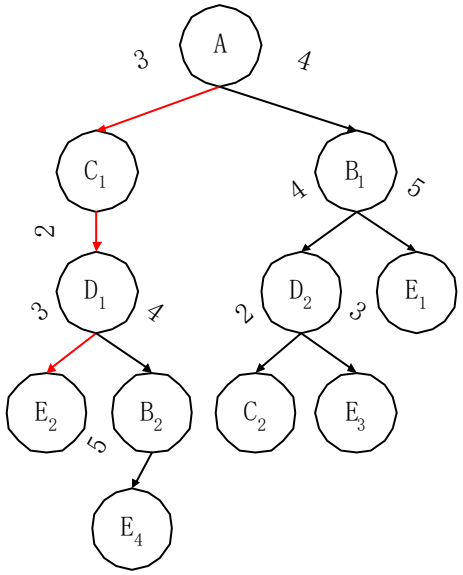
代价树搜索举例



交通图



交通图



交通图的代价树

在图所示的代价树中，首先对A进行扩展，得到B1和C1，由于C1的代价小于的B1代价，所以首先把C1送入CLOSED表进行考察。此时代价树的宽度优先搜索与代价树的深度优先搜索是一致的。但往下继续进行时，两者就不一样了。对进行C1扩展得到D1，D1的代价为5。此时OPEN表中有D1和B1，B1的代价为4。若按代价树的宽度优先搜索方法进行搜索，应选B1送入CLOSED表，但按代价树的深度优先搜索方法，则应选D1送入CLOSED表。扩展后D1，再选E2，到达了目标节点。

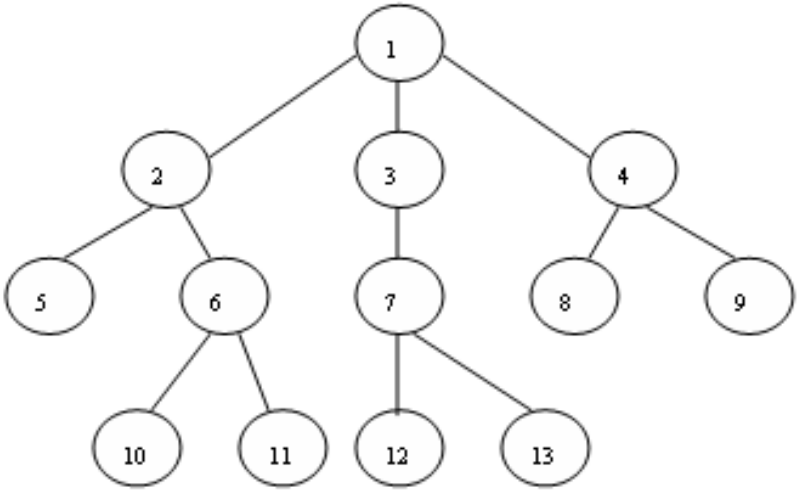
所以，按代价树的深度优先搜索方法，得到的解路径是A→C1→D1→E2代价为8，**这与宽度优先搜索得到的结果相同，但这只是巧合，一般情况下这两种方法中得到的结果不一定相同**。

在**代价树的宽度优先搜索**中，每次都是从OPEN表的**全体节点中选择一个代价最小的节点**送入CLOSED表进行观察，而**代价树的深度优先搜索是从刚扩展出的子节点中选一个代价最小的节点**送入CLOSED表进行观察。

盲目搜索

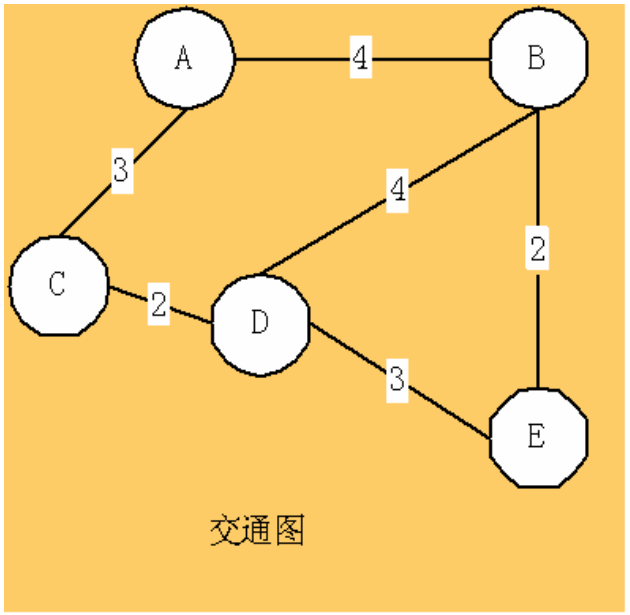
- 按照事先规定的路线进行搜索
  - 广度优先搜索是按“层”进行搜索的，先进入OPEN 表的节点先被考察
  - 深度优先搜索是沿着纵深方向进行搜索的，后进入OPEN表的节点先被考察
- 按已经付出的代价决定下一步要搜索的节点
  - 代价树的广度优先
  - 代价树的深度优先

作业题



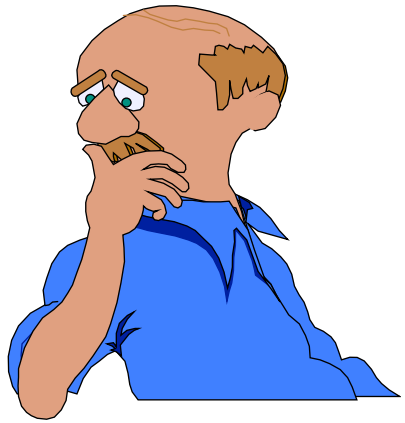
- 1 补充题：列出图中树的节点访问序列以满足下面的两个搜索策略，并写出其搜索过程中的open和closed表（在所有情况中都选择最左分枝优先访问，设节点12为目标节点）：
- （1）深度优先搜索；
- （2）宽度优先搜索。

- 2 补充题：根据右边的交通图，分别利用代价树的广度优先搜索和代价树的深度优先搜索求出从A点出发到达E点的路径。



45

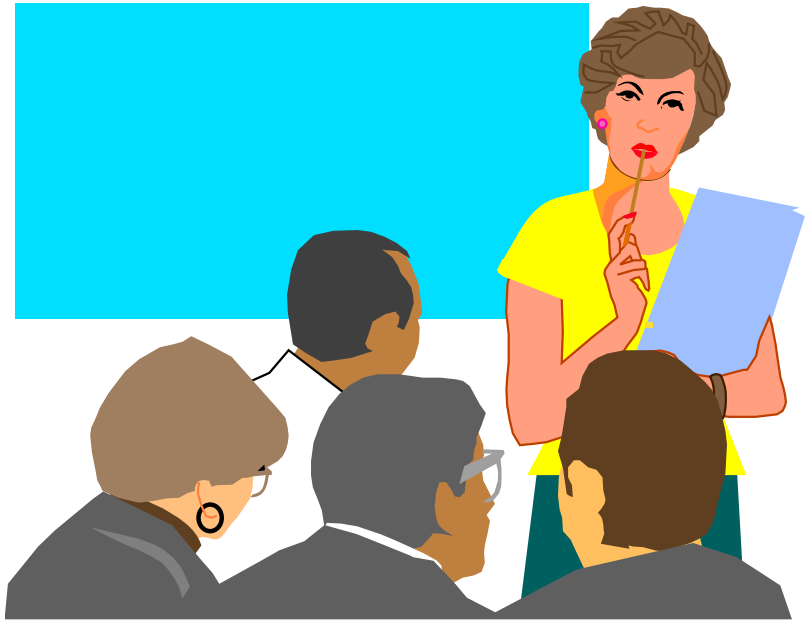
启发式搜索



- 为什么需要启发式搜索
- ◇盲目搜索的不足
- 效率低，耗费过多的计算空间与时间
- 可能带来组合爆炸
- 分析前面介绍的宽度优先、深度优先搜索，或等代价搜索算法，其主要的差别是OPEN表中待扩展节点的顺序问题。人们就试图找到一种方法用于排列待扩展节点的顺序，即选择最有希望的节点加以扩展，那么，搜索效率将会大为提高。
- ◇什么可以做为启发信息？
- 进行搜索技术一般需要某些有关具体问题的领域的特性信息，把此种信息叫做启发信息。

启发式搜索

- ◇特点
- 重排OPEN表，选择**最有希望**的节点加以扩展
- ◇种类
- 有序搜索、A*算法等



启发式搜索的估价函数

- 定义

  ◇ 估价函数（evaluation function），估算节点希望程度的量度

  ◇ 表示方法

- f(n)—表示节点n的估价函数值

建立估价函数的一般方法

- ◇提出任意节点与目标集之间的距离量度或差别量度
- ◇在棋盘式的博弈和难题中根据棋局的某些特点来决定棋局的得分数。
- 这些特点被认为与向目标节点前进一步的希望程度有关

有序搜索

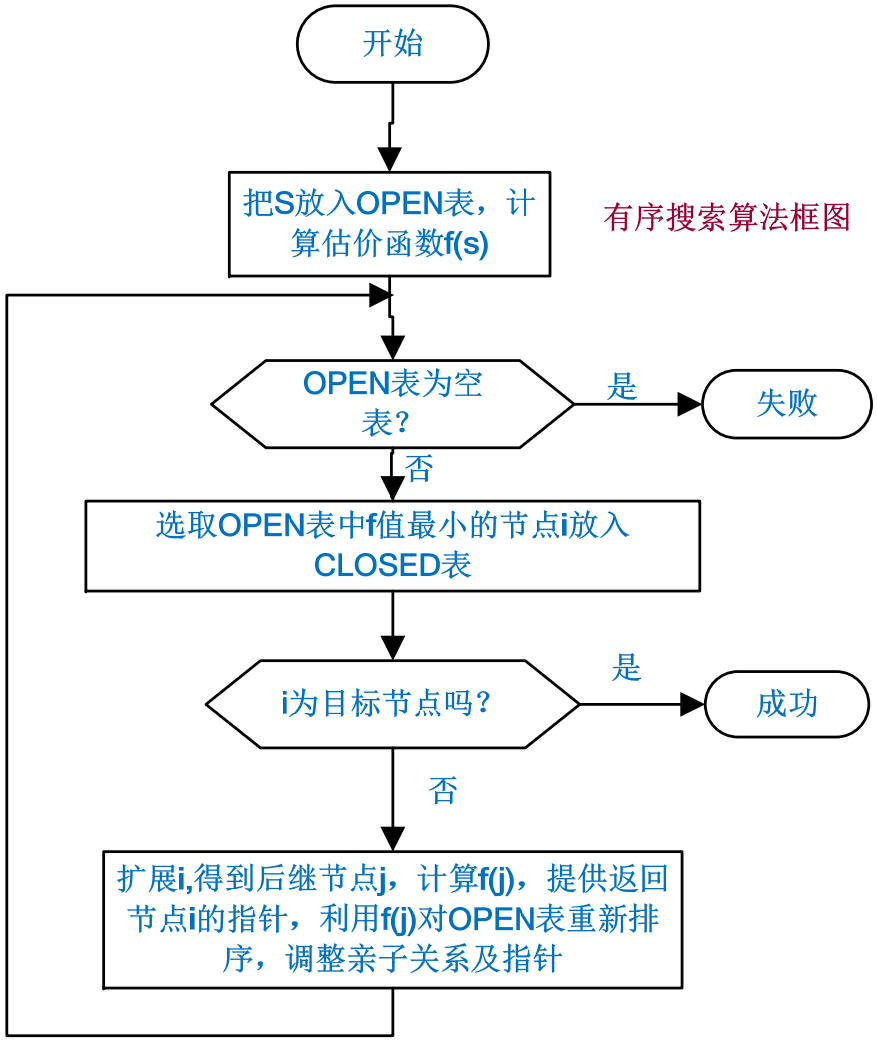
- 概念
- ◇有序搜索（ordered search），即最好优先搜索（best-first search）。
- ◇选择OPEN表上具有最小f值的节点作为下一个要扩展的节点。

有序搜索算法

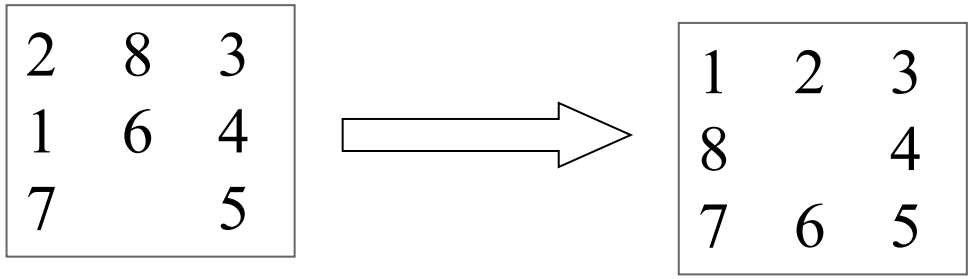
1. 把起始节点S放在OPEN表中，计算f(s)并把其值与节点S联系起来。
2. 如果OPEN是个空表，则失败退出，无解。
3. 从OPEN表中选择一个f值最小的节点i。结果有几个节点合格，当其中有一个为目标节点时，则选择此目标节点，否则就选择其中任一个节点作为节点i。
4. 把节点i从OPEN表中移出，并把它放入CLOSED的扩展节点表中。
5. 如果i是个目标节点，则成功退出，求得一个解。
6. 扩展节点i，生成其全部后继节点，对于i的每一个后继节点j:
    1. 计算f(j)。
    2. 如果j既不在OPEN表中，又不在CLOSED表中，则用估价函数f把它添入OPEN表，从j加一指向其父辈节点i的指针，以便一旦找到目标节点时记住一个解答路径。
    3. 如果j已在OPEN表上或CLOSED表上，则比较刚刚对j计算过的f值和前面计算过的该节点在表中的f值。

    如果新的f值较小，则
    1. 以此新值取代旧值。
    2. 从j指向i，而不是指向它的父辈节点。
    3. 如果节点j在CLOSED表中，则把它移回OPEN表

有序搜索算法框图


开始
把S放入OPEN表，计算估价函数f(s)
有序搜索算法框图
OPEN表为空表？
是
失败
否
选取OPEN表中f值最小的节点i放入CLOSED表
i为目标节点吗？
是
成功
否
扩展i,得到后继节点j，计算f(j)，提供返回节点i的指针，利用f(j)对OPEN表重新排序，调整亲子关系及指针

一个算法的例子



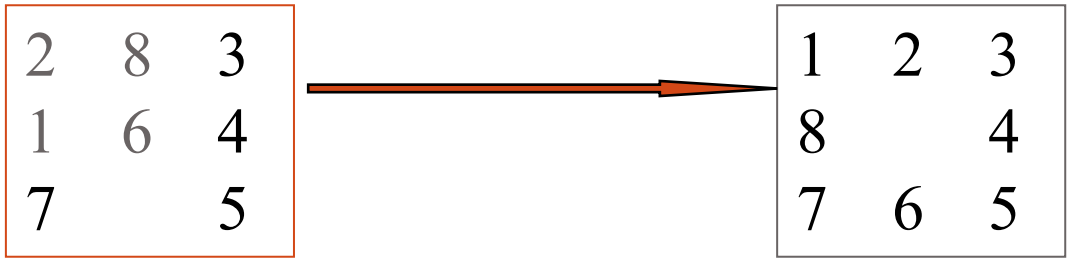
定义评价函数：

$f(n) = g(n) + h(n)$

$g(n)$为从初始节点到当前节点的路径长度（深度）

$h(n)$为当前节点“不在位”的将牌数

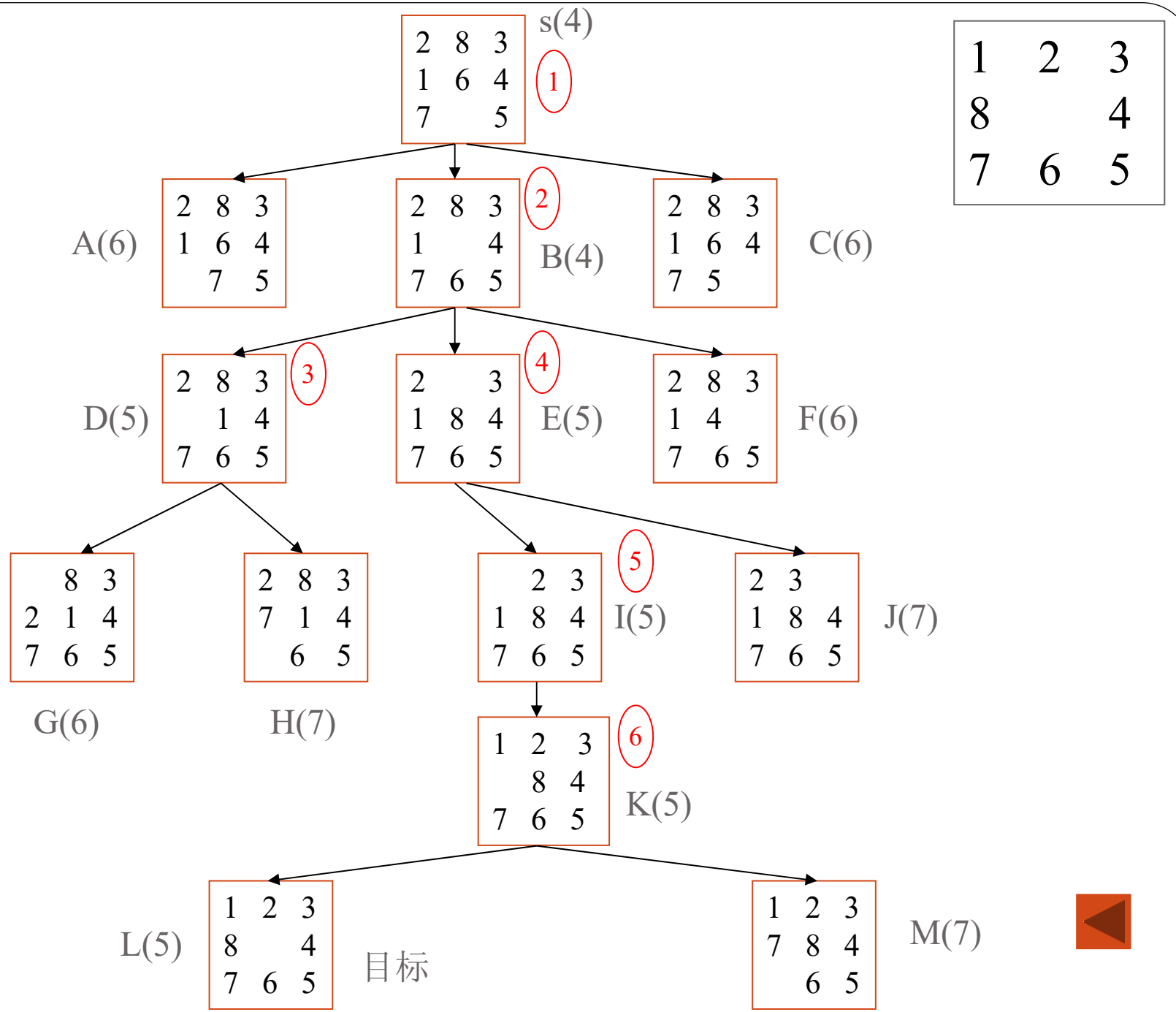
h计算举例



h(n) =4

g(n)=0

f(n)=4+0=4


s(4)
1
A(6)
2
B(4)
C(6)
D(5)
3
4
E(5)
F(6)
G(6)
H(7)
5
I(5)
J(7)
6
K(5)
L(5)
目标
M(7)

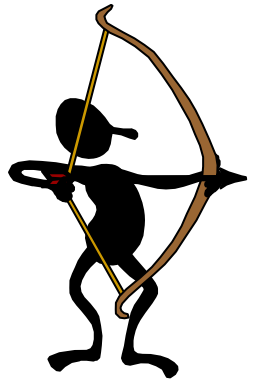
A*算法

- 算法特点
- ◇隶属于有序搜索算法的一种；□
- ◇其特点在于对估价函数的定义上.
- 算法符号
- $k(n_i, n_j)$
- ◇表示任意两个节点ni和nj之间最小代价路径的实际代价；
- ◇对于两节点间没有通路的节点，函数k没有定义；
- ◇从节点n到某个具体的目标节点t_i，某一条最小代价路径的代价可由$k(n, t_i)$给出。

◇ h*（n）表示整个目标节点集合$\{t_i\}$上所有$k(n, t_i)$中最小的一个；

◇对于任何不能到达目标节点的节点n，函数h*没有定义。

◇ h*（n）就是从n到目标节点最小代价路径的代价，而且从n到目标节点能够获得到h*（n）的任一路径就是一条从n到某个目标节点的最佳路径；

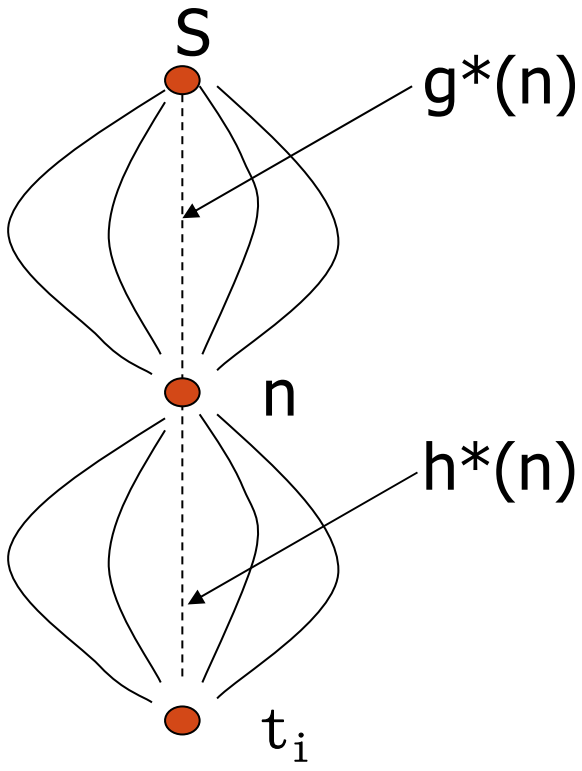


估价函数的设计

h*(n)的估计h(n)依赖于有关问题的领域的启发信息。h叫做启发函数

- 定义g*： g*(n)=k(S, n)；
- 定义f*： f*(n)=g*(n)+h*(n)；
- 希望估价函数f是f*的一个估计：
  f(n)=g(n)+h(n)
- （g是g*的估计, h是h*的估计）.

对于g(n)来说，一个明显的选择就是搜索树中从S到n这段路径的代价，这一代价可以由从n到S寻找指针时，把所遇到的各段弧线的代价加起来给出(这条路径就是到目前为止用搜索算法找到的从S到n的最小代价路径))。这个定义包含了g(n)≥g*(n)


S
g*(n)
n
h*(n)
t_i

A*算法

定义1

- 在GRAPHSEARCH过程中，如果第8步的重排OPEN表是依据$f(n)=g(n)+h(n)$进行的，则称该过程为A算法；

定义2

- 在A算法中，如果对所有的n存在$h(n) \leqslant h^*(n)$，则称$h(n)$为$h^*(n)$的下界，它表示某种偏于保守的估计；

定义3

采用$h^*(n)$的下界$h(n)$为启发函数的A算法，称为A*算法。

A*算法过程

(1) 把S放入OPEN表，记f=h，令CLOSED为空表。

(2) 重复下列过程，直至找到目标节点止。若OPEN为空表，则宣告失败。

(3) 选取OPEN表中未设置过的具有最小f值的节点为最佳节点BESTNODE，并把它放入CLOSED表。

(4) 若BESTNODE为一目标节点，则成功求得一解。

(5) 若BESTNODE不是目标节点，则扩展之，产生后继节点SUCCSSOR。

(6) 对每个SUCCSSOR进行下列过程：

(a) 建立从SUCCSSOR返回BESTNODE的指针。

(b) 计算g(SUC)=g(BES)+k(BES, SUC)。

(c) 如果SUCCSSOR∈OPEN，则称此节点为OLD，并把它添至BESTNODE的后继节点表中。

(d) 比较新旧路径代价。如果g(SUC)＜g(OLD)，则重新确定OLD的父辈节点为BESTNODE，记下较小代价g(OLD)，并修正f(OLD)值。

(e) 若至OLD节点的代价较低或一样，则停止扩展节点。

(f) 若SUCCSSOR不在OPEN表中，则看其是否在CLOSED表中。

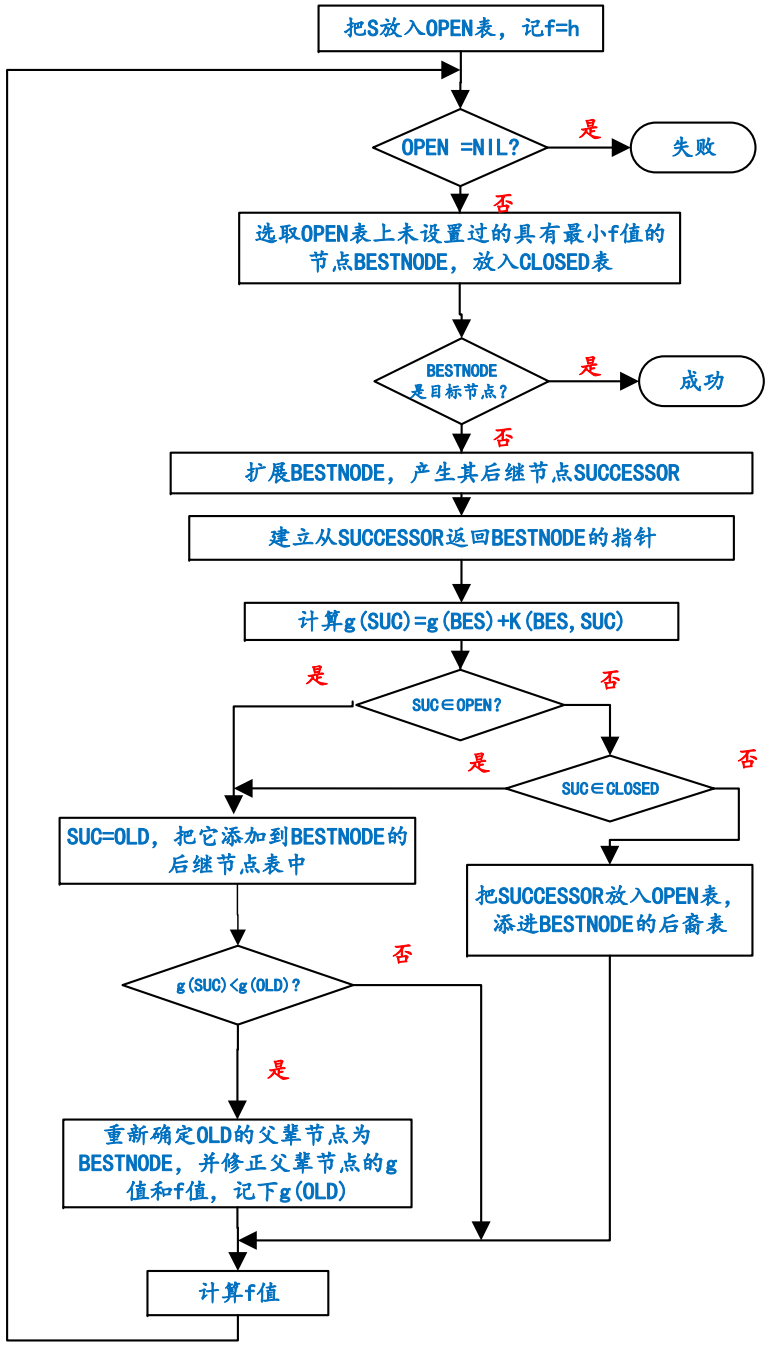
(g) 若SUCCSSOR在CLOSE表中，则转向c。

(h) 若SUCCSSOR既不在OPEN表中，又不在CLOSED表中，则把它放入OPEN表中，并添入BESTNODE后裔表，然后转向(7)

(7) 计算f值。

(8) GO LOOP

A*算法


把S放入OPEN表，记f=h
OPEN =NIL?
是
失败
否
选取OPEN表上未设置过的具有最小f值的节点BESTNODE，放入CLOSED表
BESTNODE是目标节点?
是
成功
否
扩展BESTNODE，产生其后继节点SUCCESSOR
建立从SUCCESSOR返回BESTNODE的指针
计算g(SUC)=g(BES)+K(BES,SUC)
是
SUC∈OPEN?
否
是
SUC∈CLOSED
否
SUC=OLD，把它添加到BESTNODE的后继节点表中
把SUCCESSOR放入OPEN表，添进BESTNODE的后裔表
g(SUC)<g(OLD)?
否
是
重新确定OLD的父辈节点为BESTNODE，并修正父辈节点的g值和f值，记下g(OLD)
计算f值

启发式搜索小结

- 启发式策略就是利用与问题有关的启发信息进行搜索的策略。
- 利用估价函数来估计待搜索节点的希望程度，并以此排序。估价函数一般由两部分组成：
- $$f(n)=g(n)+h(n)$$
- $g(n)$是从初始节点到节点n已付出的实际代价，$h(n)$是从节点n到目的节点的最佳路径的估计代价。$h(n)$体现了搜索的启发信息。
- 在$f(n)$中，$g(n)$的比重越大，越倾向于宽度优先搜索，而$h(n)$的比重越大，表示启发性越强。

启发式搜索小结

- $f(n)=g(n)+h(n)$
- 在$f(n)$中，$g(n)$的比重越大，越倾向于宽度优先搜索，而$h(n)$的比重越大，表示启发性越强。
- $g(n)$的作用一般是不可忽略的，保持$g(n)$项就保持了搜索的宽度优先成分，这有利于搜索的完备性，但会影响搜索的效率。

启发式搜索小结

- A算法
- 在GRAPHSEARCH过程中，如果第8步的重排OPEN表是依据 $f(n)=g(n)+h(n)$ 进行的，则称该过程为A算法；
- 显然，A算法对$h(n)$没有明确的限制。
- A*算法
- 采用 $h^*(n)$的下界$h(n)$为启发函数的A算法，称为A*算法。
- A*算法要求$h(n) \leqslant h^*(n)$。它表示某种偏于保守的估计。
- 如果算法有解，A*算法一定能够找到最优的解答。
- 一般说来，在满足$h(n) \leqslant h^*(n)$的前提下，$h(n)$的比重越大越好，$h(n)$的比重越大表示启发性越强。

例题

- 对于如图所示的八数码问题，给出满足A*算法的启发函数，并给出相应的搜索图。

$S_0 =$

2	8	3
1	6	4
7		5

$S_g =$

1	2	3
8		4
7	6	5

66

$S_0=$

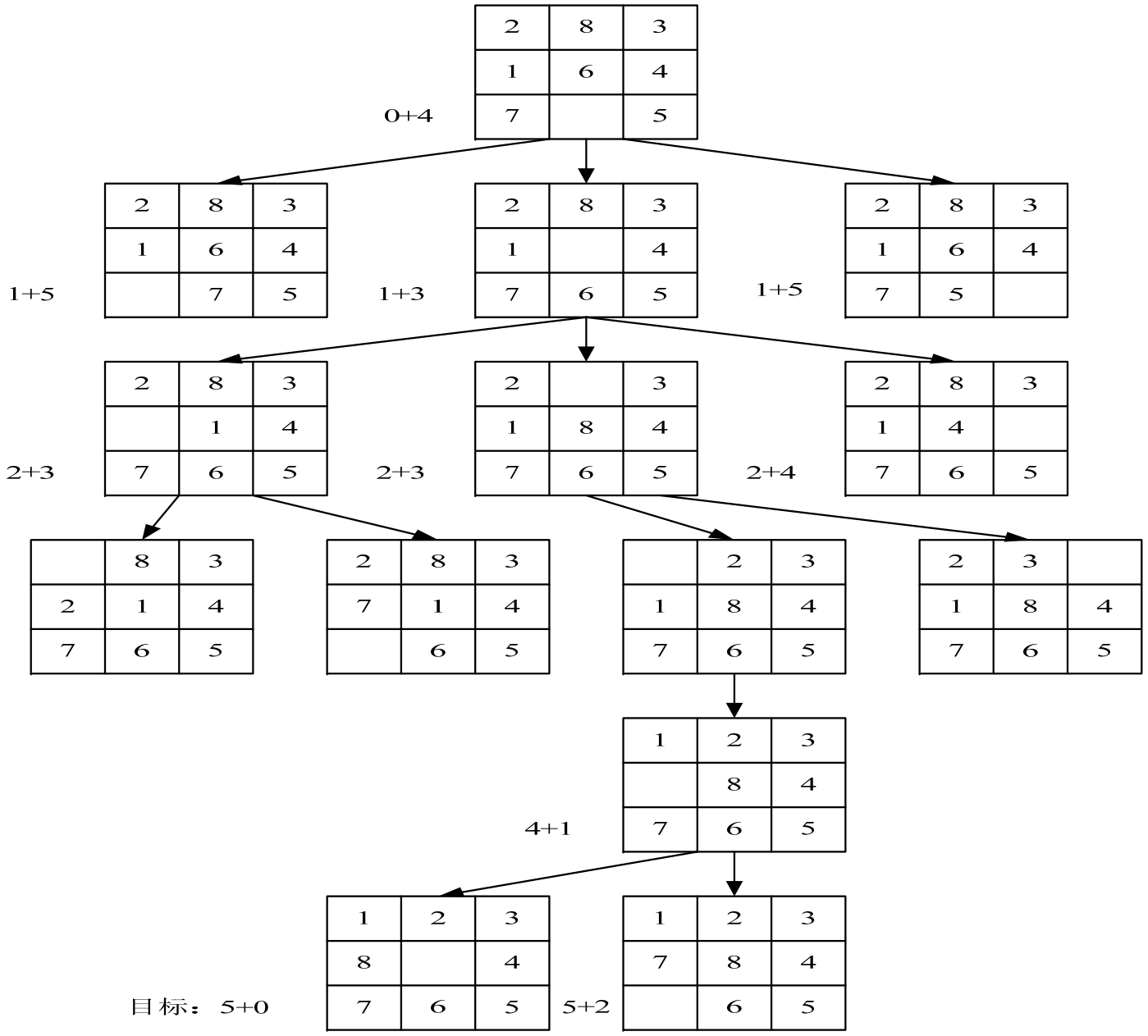
2	8	3
1	6	4
7		5

$S_g=$

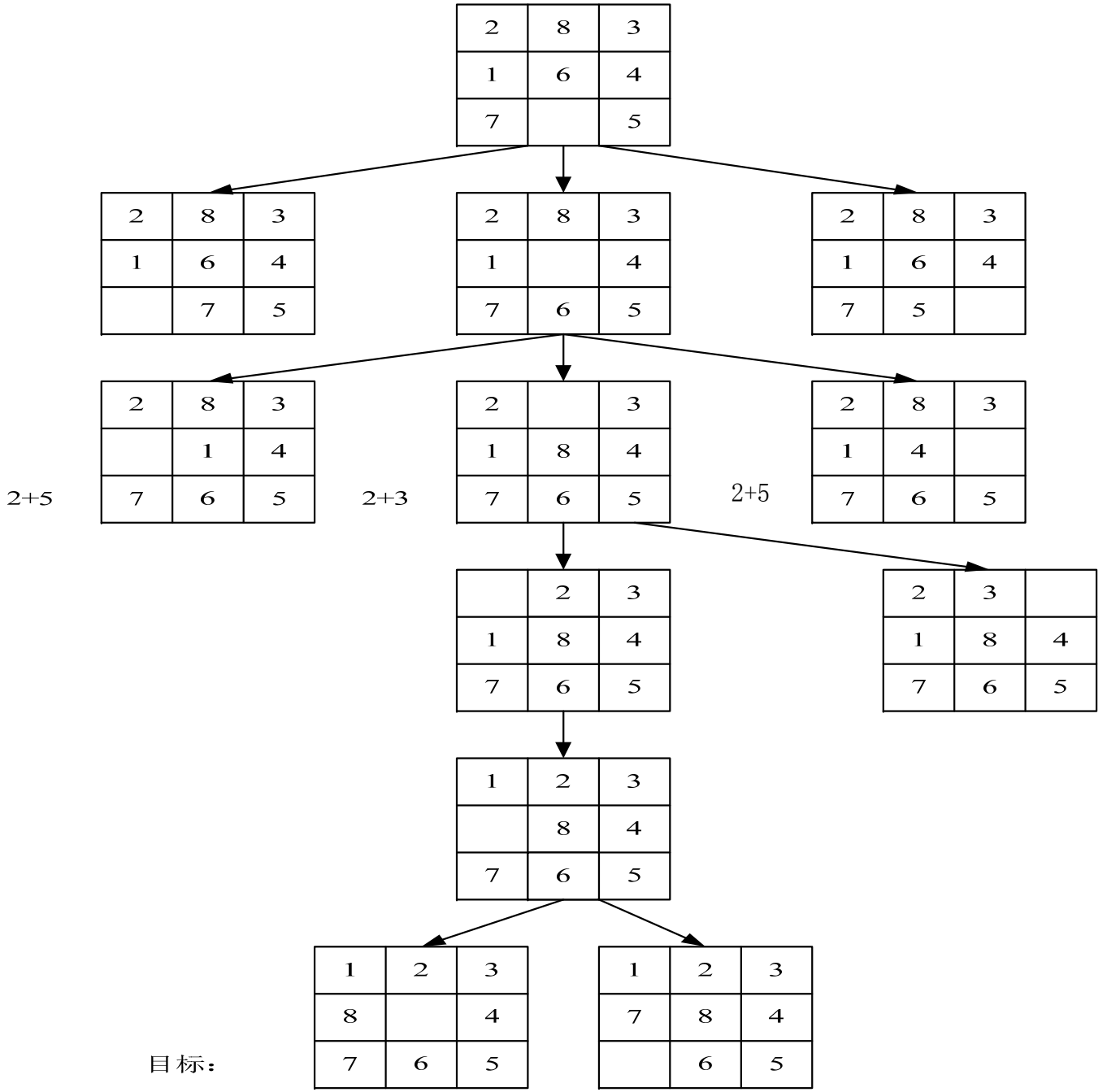
1	2	3
8		4
7	6	5

- 解：
- 启发函数的选取如下：$g(n)$表示节点n在搜索树中的深度，$h(n)=\omega(n)$表示节点n中不在目标状态中相应位置的数码个数，则根据

  $f(n)=\omega(n)+g(n)$，可以得到如图所示搜索过程。
- 在上面确定$h(n)$时，尽管并不知道$h^*(n)$具体为多少，但当采用单位代价时，通过对“不在目标状态中相应位置的数码个数”的估计，可以得出至少需要移动$h(n)$步才能够到达目标，显然$h(n)\leqslant h^*(n)$。因此它满足A*算法的要求。


0+4
1+5
1+3
1+5
2+3
2+3
2+4
4+1
目标：5+0
5+2

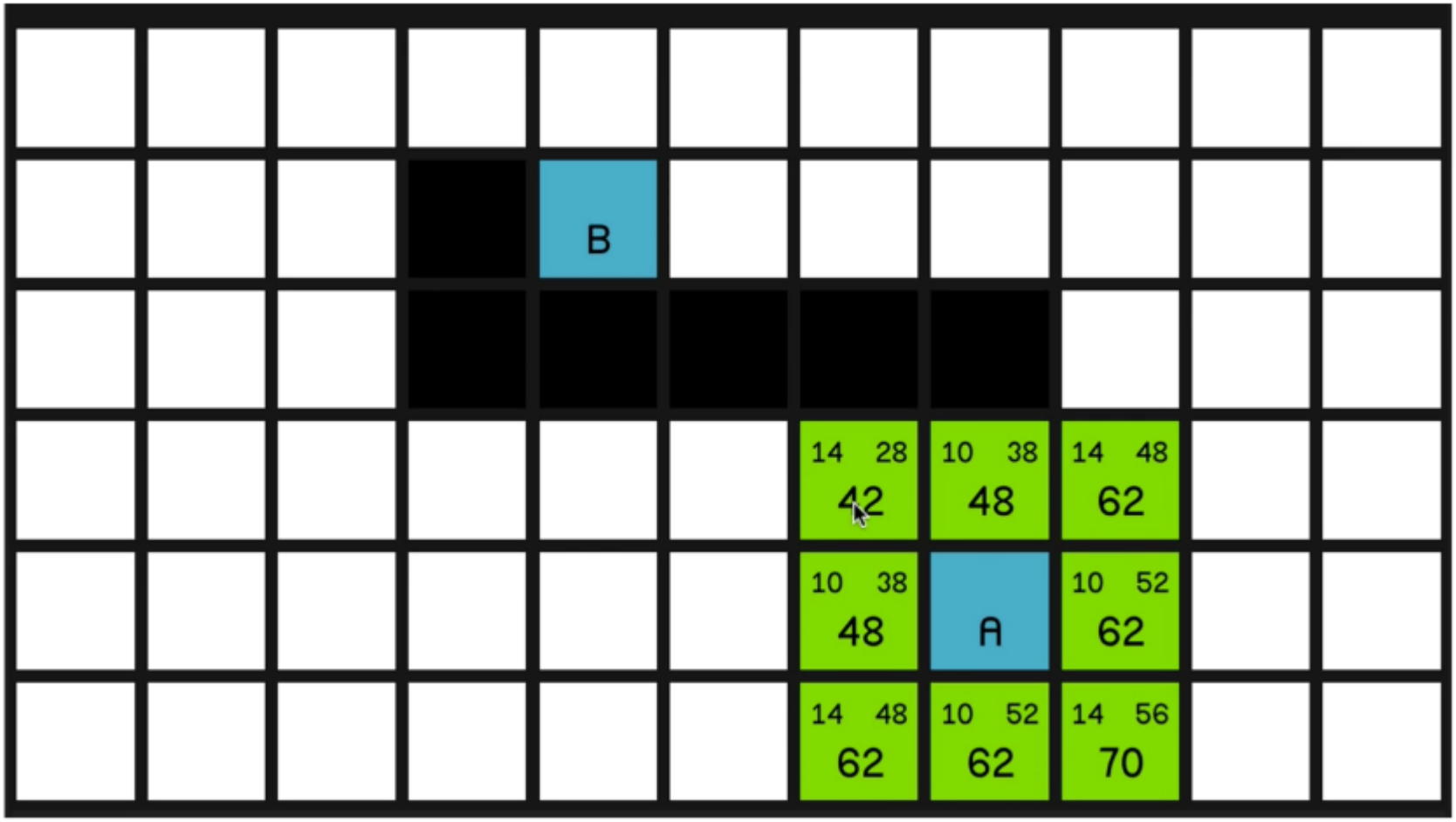
- 另解：
- 定义启发函数h(n)=p(n)为节点n的每一数码与其目标位置之间的距离总和。
- 显然有$\omega(n) \leqslant p(n) \leqslant h^*(n)$，相应的搜索过程也是A*算法。
- 然而，p(n)比$\omega(n)$有更强的启发性信息，由h(n)=p(n)构造的启发式搜索树，比$h(n)=\omega(n)$构造的启发式搜索树节点数要少。


2+5
2+3
2+5
目标：

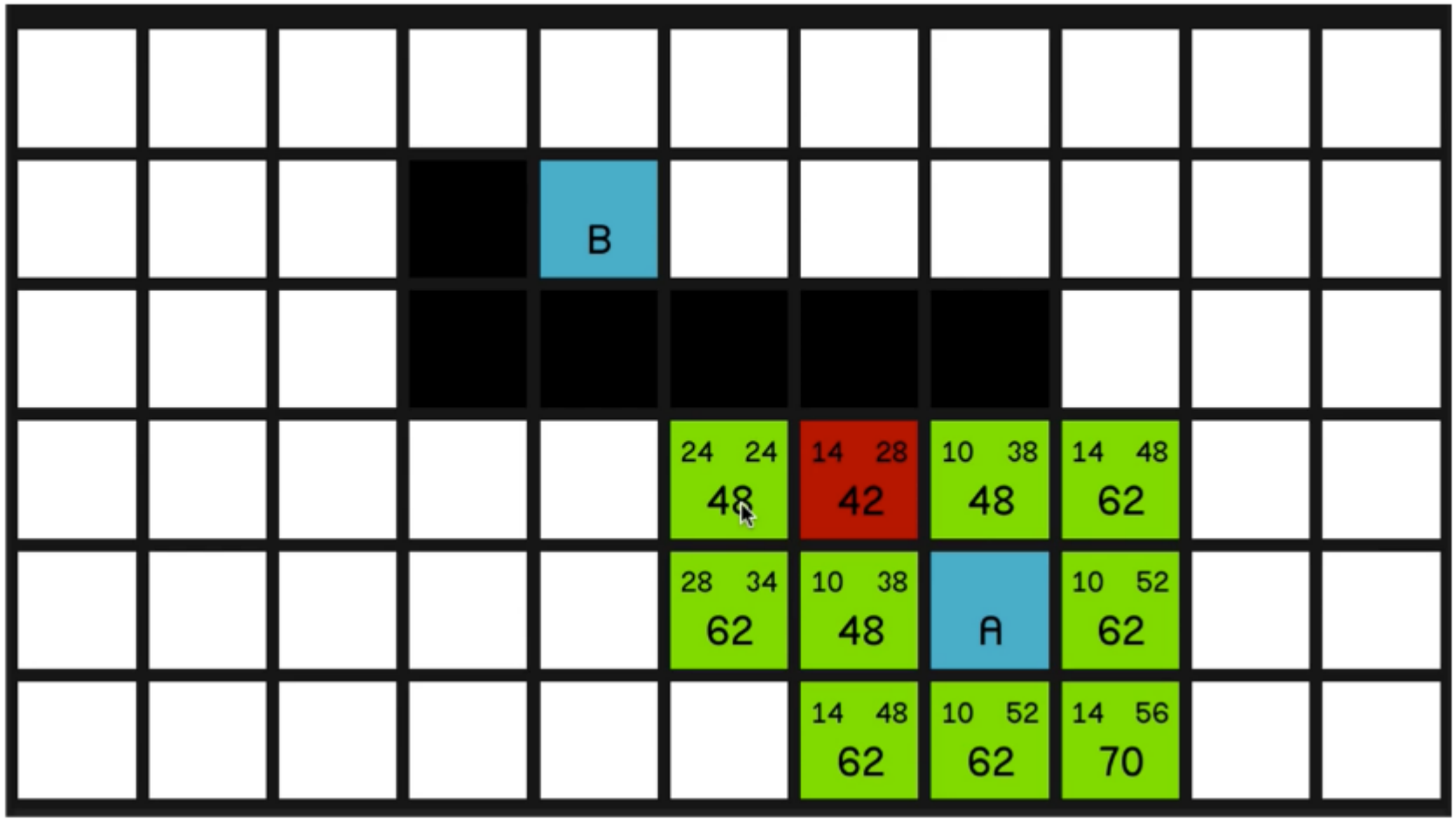
下面给出本题的启发函数的比较结果。

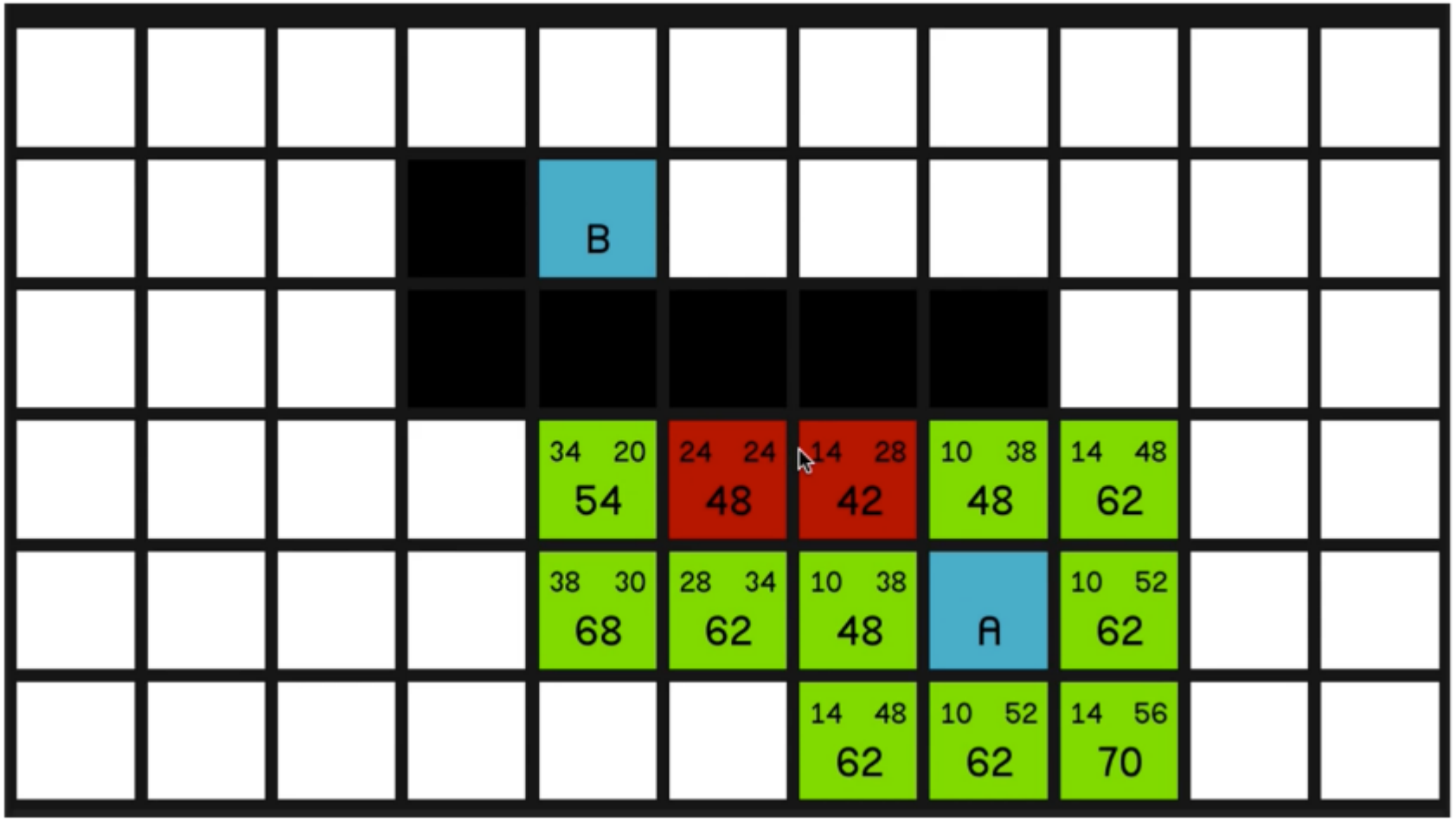
启发函数	h(n)=0	h(n)=ω(n)	h(n)=p(n)
扩展节点	26	6	5
生成节点	46	13	11

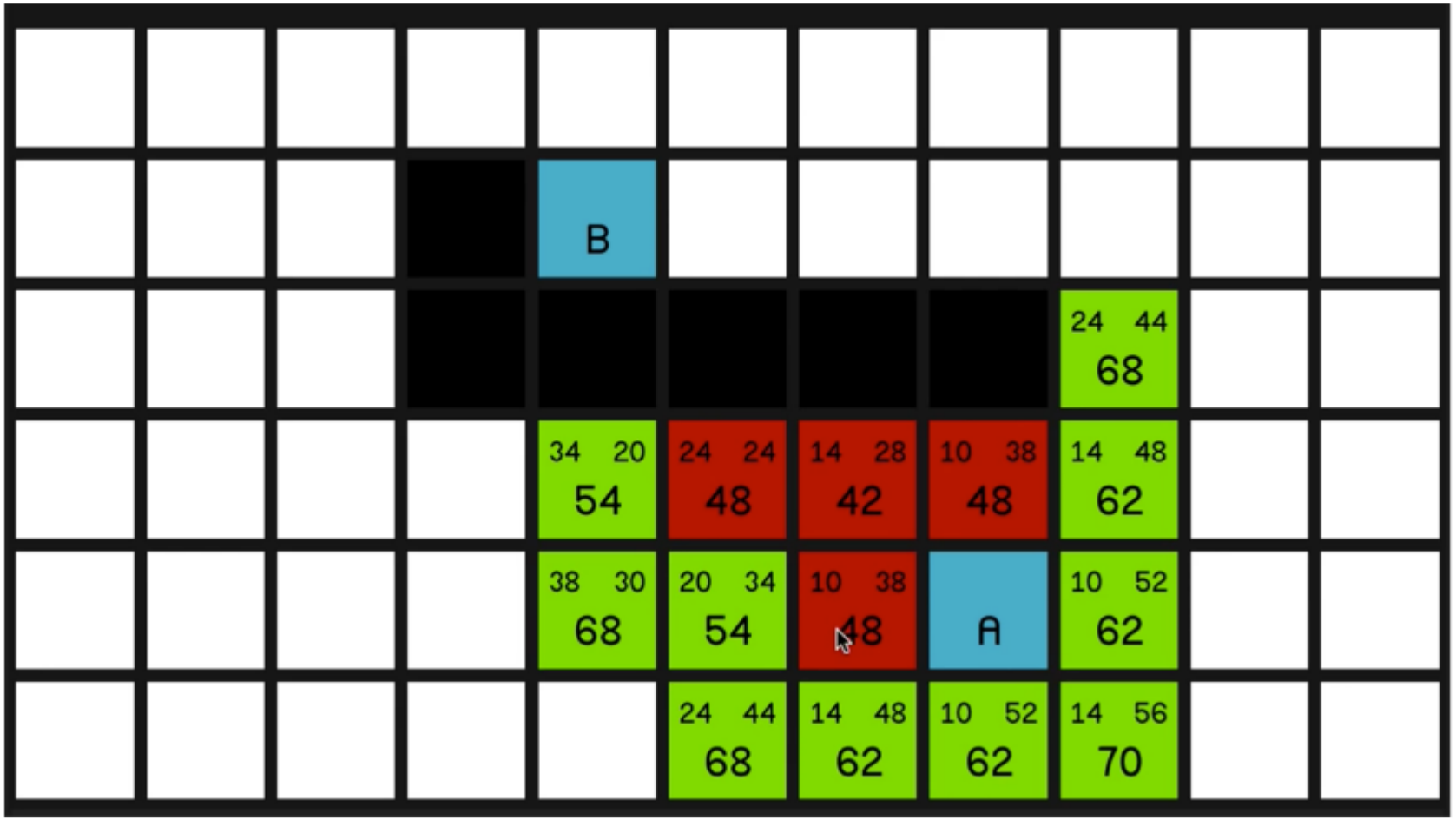
例2

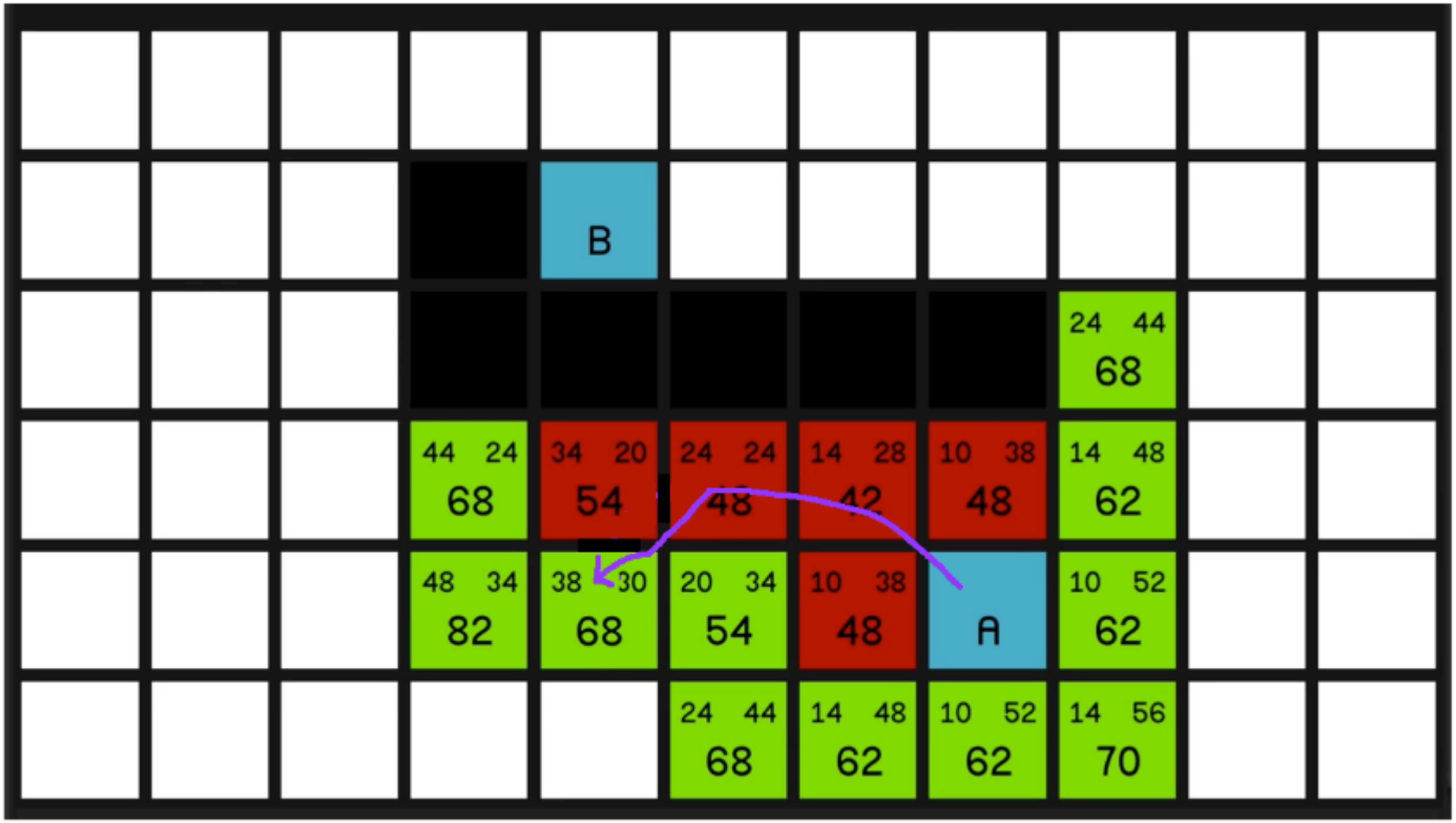


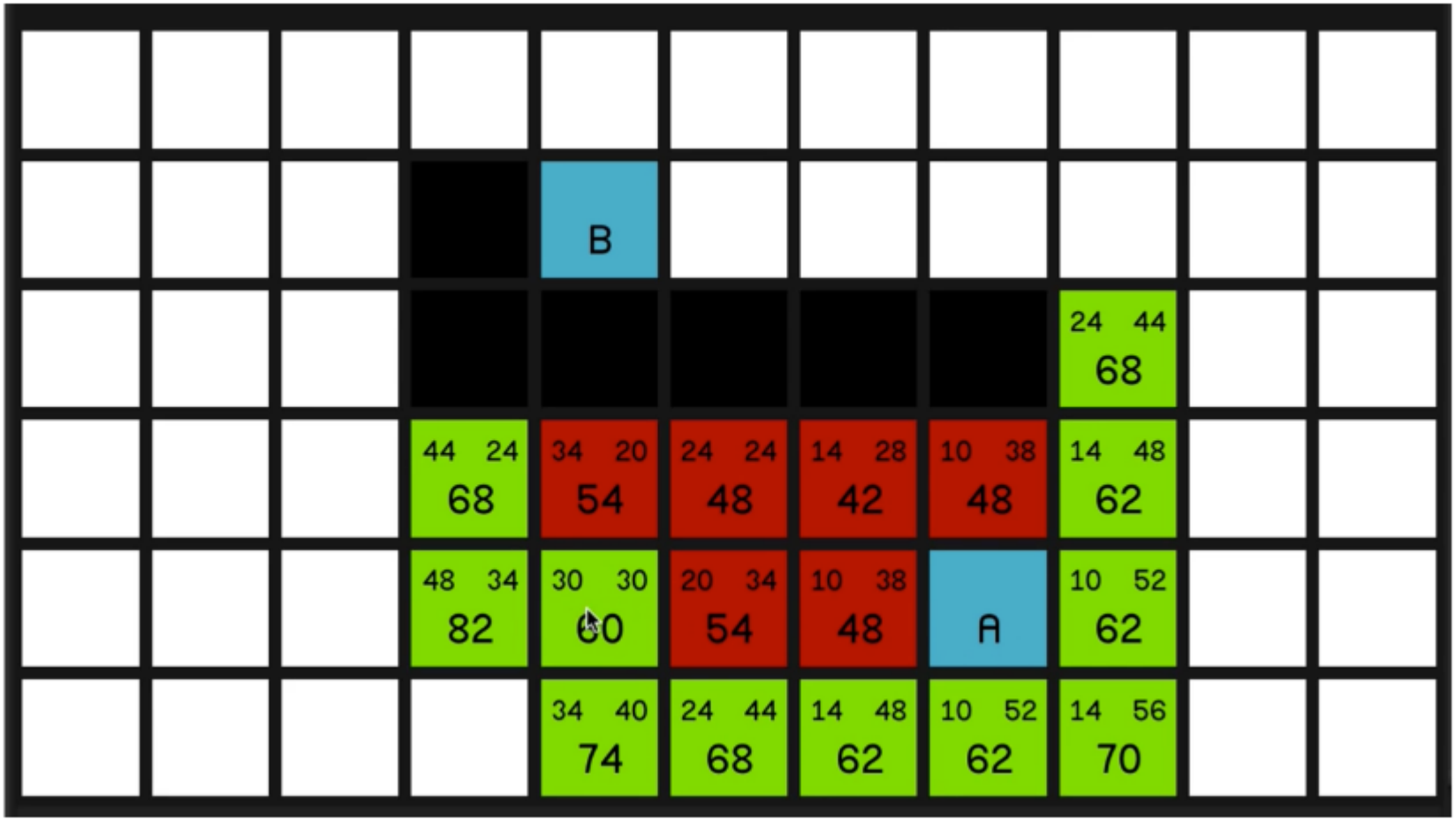
- 每个方块左上角的值记为G，表示该点到A的距离，右上角值为H，H取其到B的距离。F=H+G。

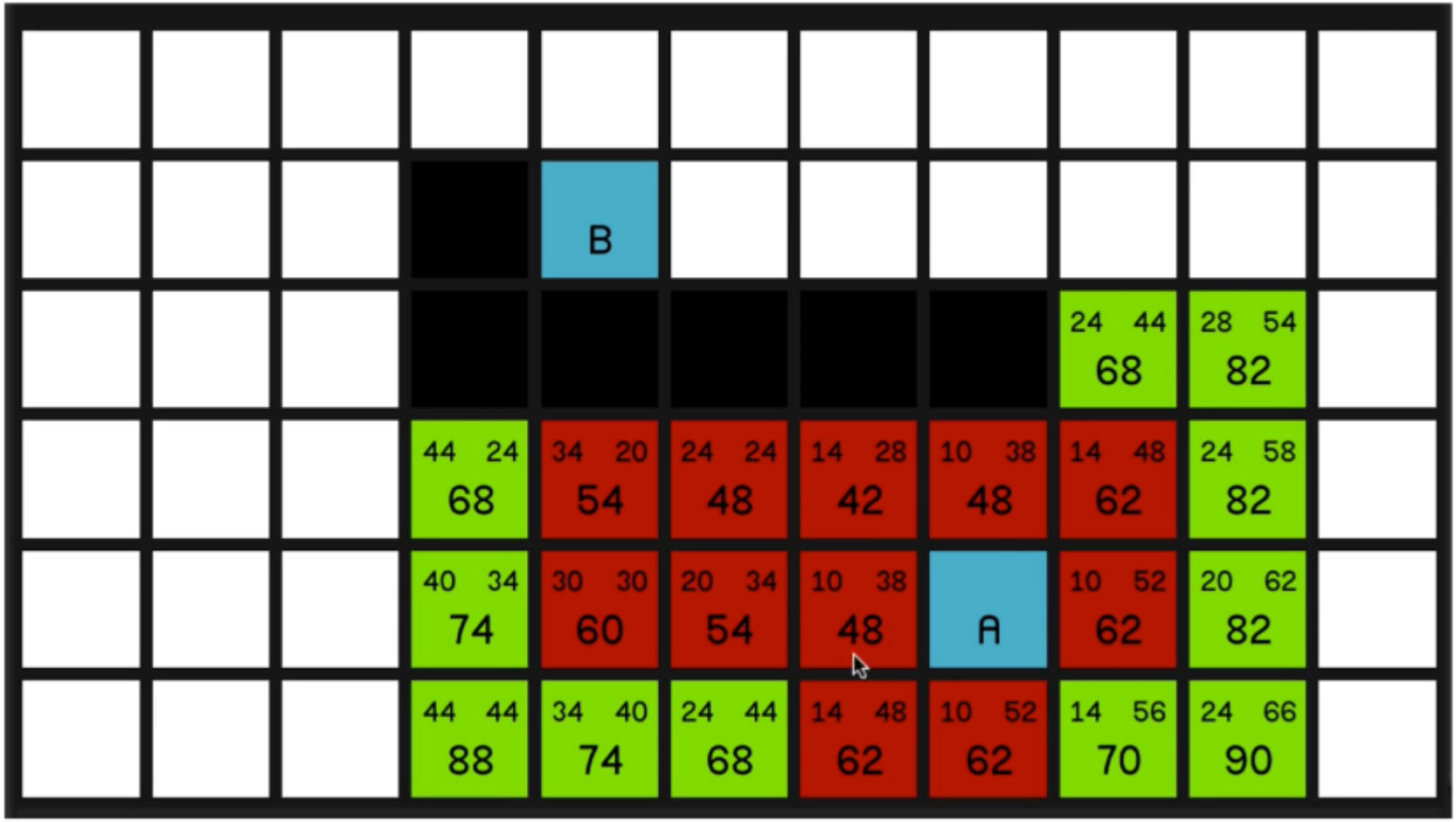

B
24 24
48
14 28
42
10 38
48
14 48
62
28 34
62
10 38
48
A
10 52
62
14 48
62
10 52
62
14 56
70

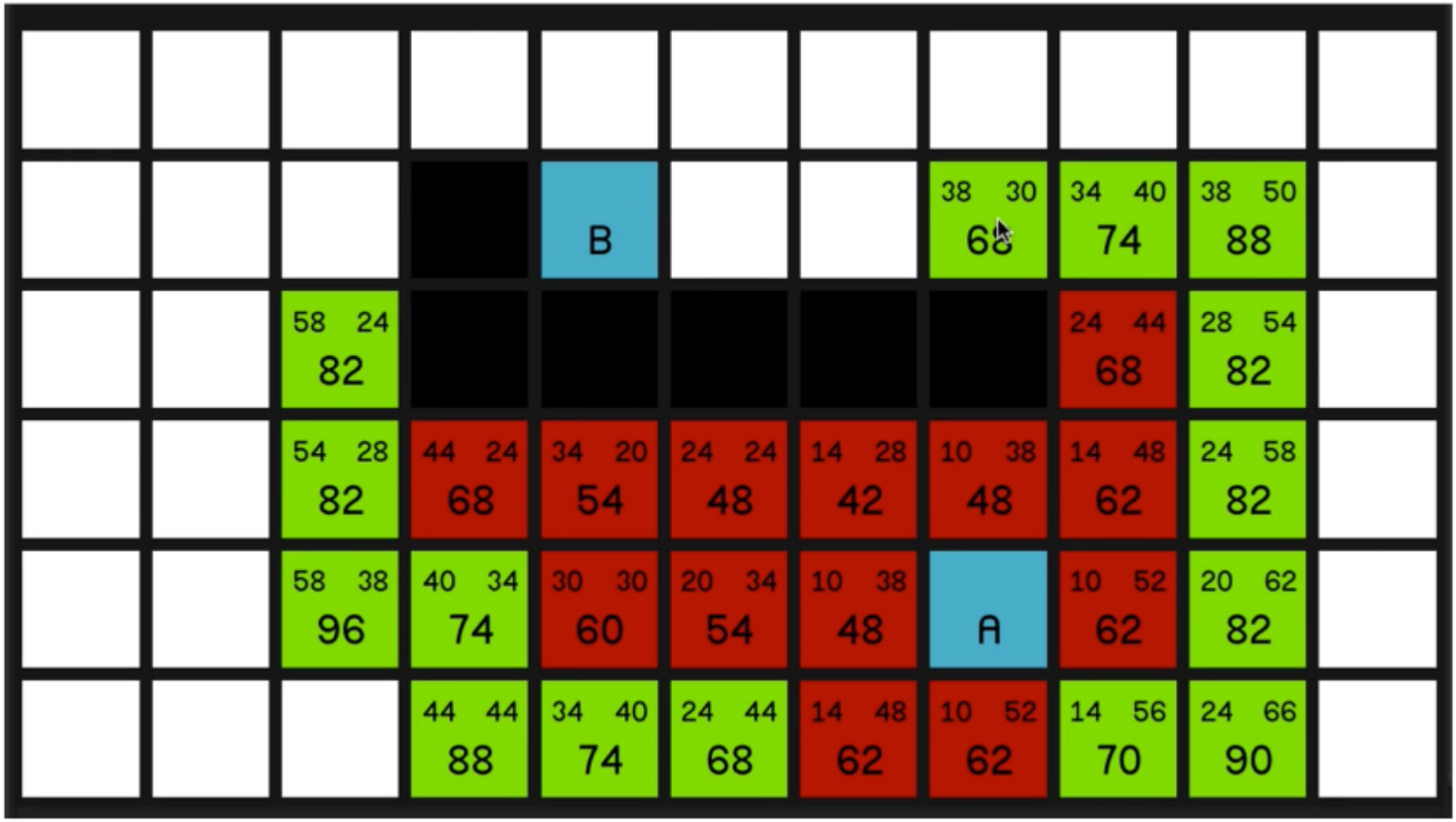

B
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
38 30
68
28 34
62
10 38
48
A
10 52
62
14 48
62
10 52
62
14 56
70

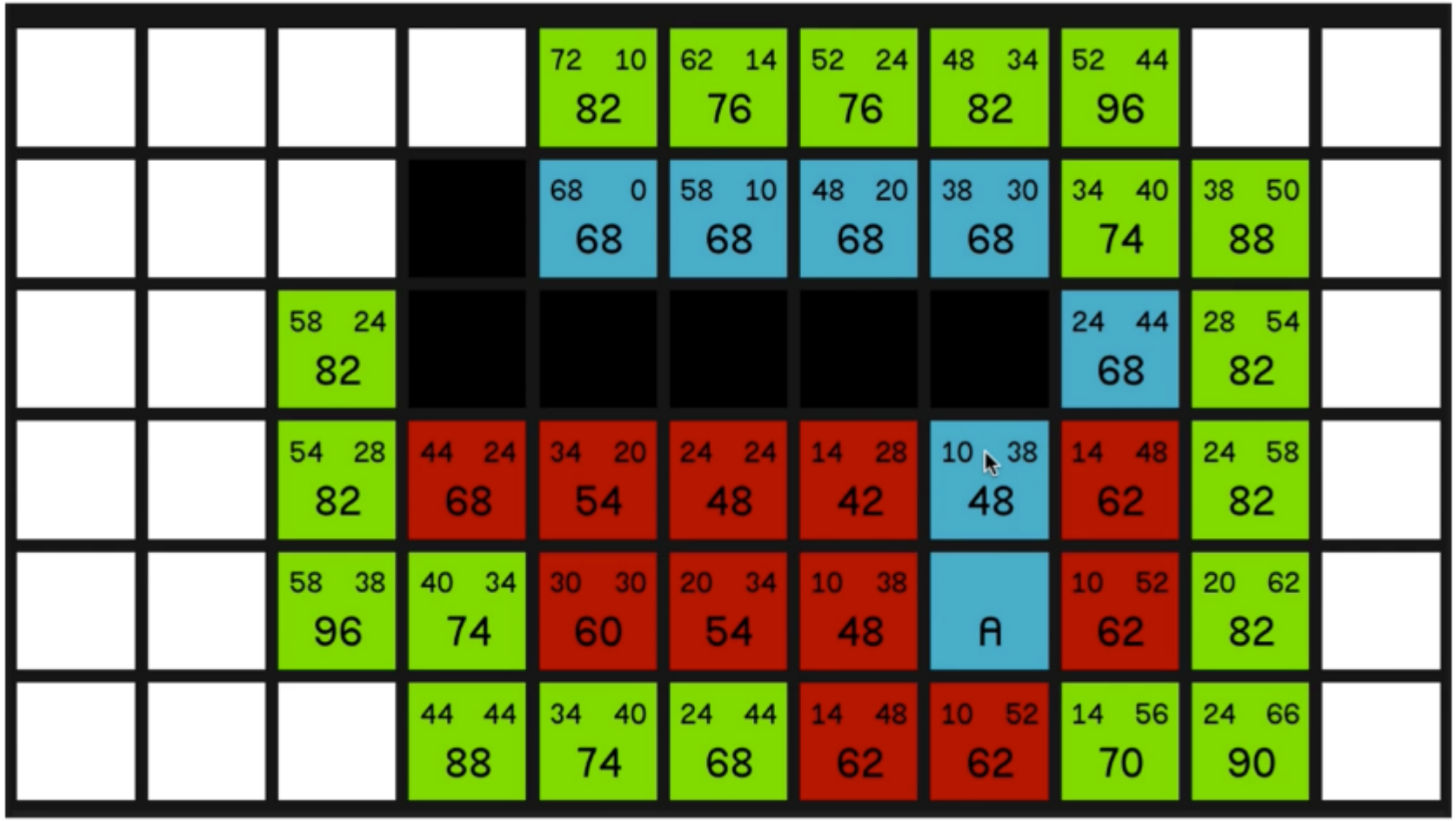

B
24 44
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
38 30
68
20 34
54
10 38
48
A
10 52
62
24 44
68
14 48
62
10 52
62
14 56
70


B
24 44
68
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
48 34
82
38 30
68
20 34
54
10 38
48
A
10 52
62
24 44
68
14 48
62
10 52
62
14 56
70


B
24 44
68
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
48 34
82
30 30
60
20 34
54
10 38
48
A
10 52
62
34 40
74
24 44
68
14 48
62
10 52
62
14 56
70


B
24 44
68
28 54
82
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
24 58
82
40 34
74
30 30
60
20 34
54
10 38
48
A
10 52
62
20 62
82
44 44
88
34 40
74
24 44
68
14 48
62
10 52
62
14 56
70
24 66
90

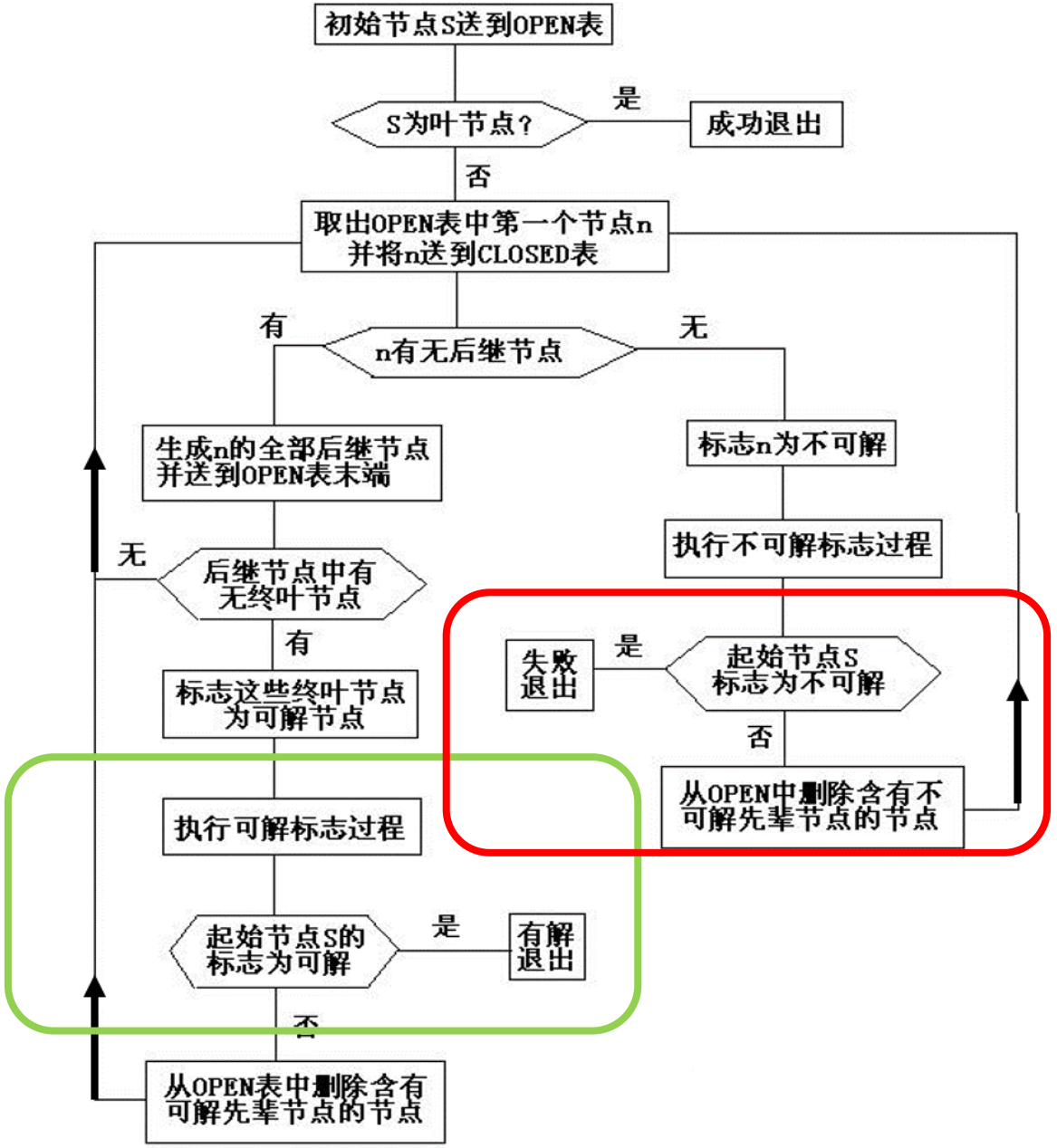

B
38 30
68
34 40
74
38 50
88
58 24
82
24 44
68
28 54
82
54 28
82
44 24
68
34 20
54
24 24
48
14 28
42
10 38
48
14 48
62
24 58
82
58 38
96
40 34
74
30 30
60
20 34
54
10 38
48
A
10 52
62
20 62
82
44 44
88
34 40
74
24 44
68
14 48
62
10 52
62
14 56
70
24 66
90


72 10 82
62 14 76
52 24 76
48 34 82
52 44 96
68 0 68
58 10 68
48 20 68
38 30 68
34 40 74
38 50 88
58 24 82
24 44 68
28 54 82
54 28 82
44 24 68
34 20 54
24 24 48
14 28 42
10 38 48
14 48 62
24 58 82
58 38 96
40 34 74
30 30 60
20 34 54
10 38 48
A
10 52 62
20 62 82
44 44 88
34 40 74
24 44 68
14 48 62
10 52 62
14 56 70
24 66 90

与/或图搜索

- 与/或图的搜索策略
  - 盲目式搜索
    - 与/或图的一般搜索
    - 宽度优先搜索
    - 深度优先搜索
  - 启发式搜索
    - 希望树
    - 机器博弈
      - Max-Min搜索
      - α–β剪枝

与或树的宽度优先搜索


初始节点S送到OPEN表
S为叶节点？
是
成功退出
否
取出OPEN表中第一个节点n
并将n送到CLOSED表
有
n有无后继节点
无
生成n的全部后继节点
并送到OPEN表末端
标志n为不可解
执行不可解标志过程
无
后继节点中有
无终叶节点
有
标志这些终叶节点
为可解节点
失败
退出
是
起始节点S
标志为不可解
否
从OPEN中删除含有不
可解先辈节点的节点
执行可解标志过程
起始节点S的
标志为可解
是
有解
退出
否
从OPEN表中删除含有
可解先辈节点的节点

与或树的宽度优先搜索

(1) 起始节点S送OPEN表

(2) 若S为叶节点，则成功结束，否则，继续

(3) 取出OPEN表的第一个节点（记作n），并送到CLOSED表

(4) 扩展节点n，生成其全部后继节点，送OPEN表末端，并设置指向n的指针
说明：此时可能出现三种情况
- 节点 n 无后继节点
- 节点 n 有后继节点、并有叶节点
- 节点 n 有后继节点、但无叶节点

(5)
- 若 n 无后继节点，标志 n 为不可解，并转9（10、11）；
- 若后继节点中有叶节点，则标志这些叶节点为可解节点，并继续6、7、8
- 否则转第3步

(6) 实行可解标志过程

(7) 若起始节点S标志为可解，则找到解而结束，否则继续

(8) 从OPEN表中删去含有可解先辈节点的节点，并转第3步

(9) 实行不可解标志过程

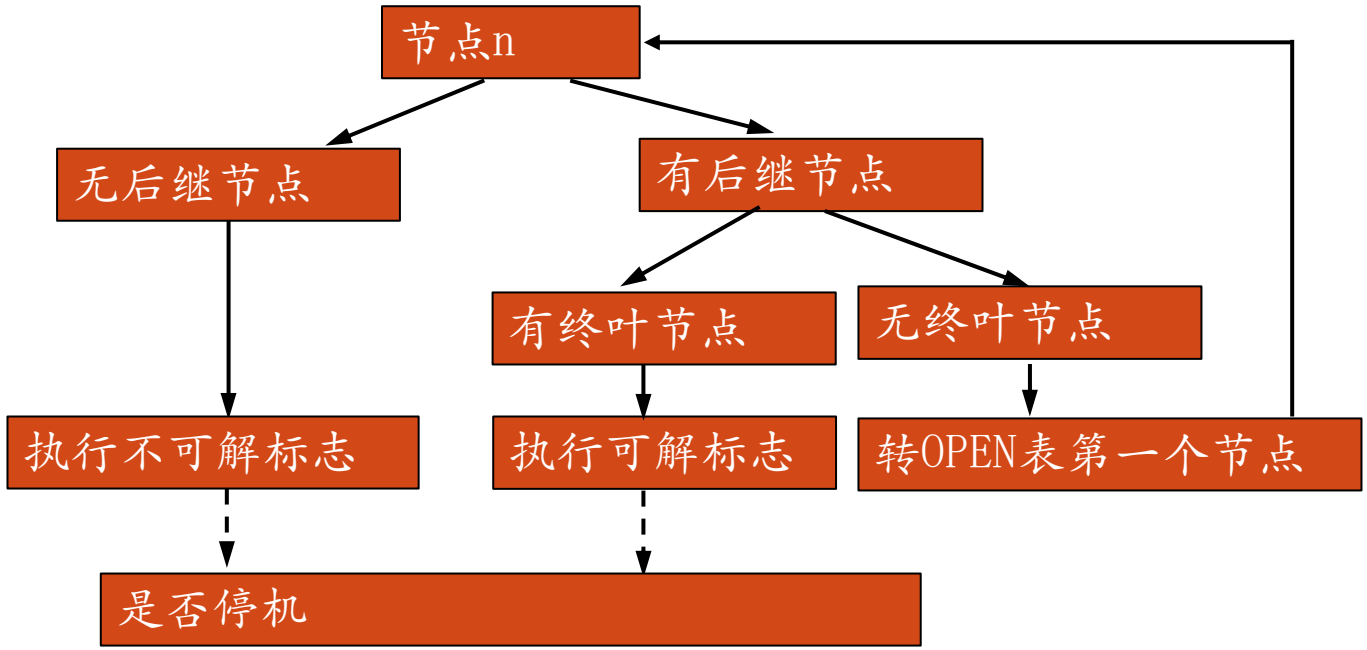
(10) 若起始节点S标志为不可解，则失败而结束，否则继续

(11) 从OPEN表中删去含有不可解先辈节点的节点

(12) 转第3步

与或树的宽度优先搜索

- **关键步骤**：扩展节点n，生成其全部后继节点，送到OPEN表末端，并设置指向n的指针。



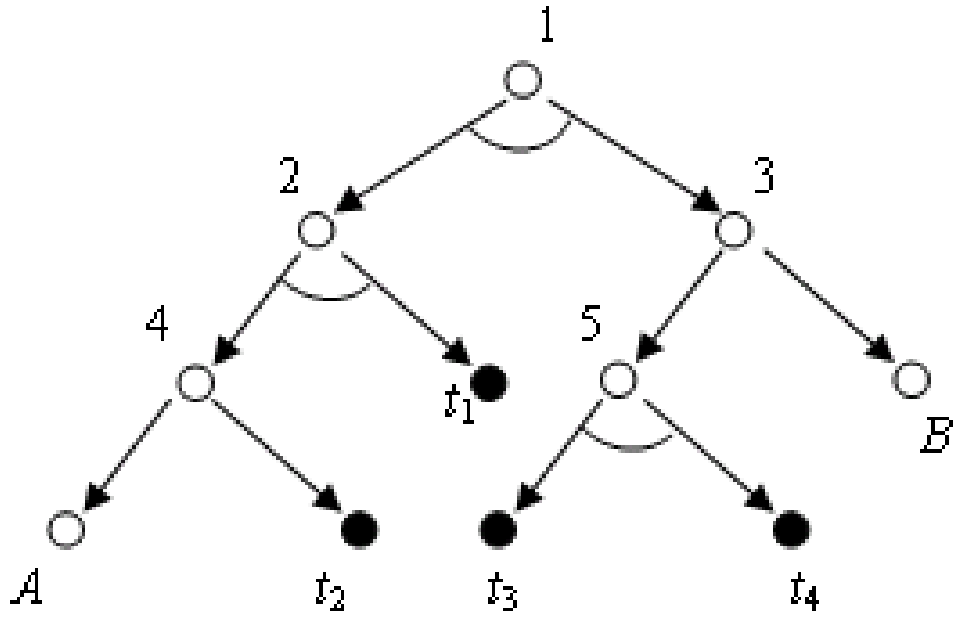
算法结束的条件

- 若初始节点被标示为可解节点，算法成功结束（有解）
- 若初始节点被标示为不可解结点，则搜索失败结束（无解）

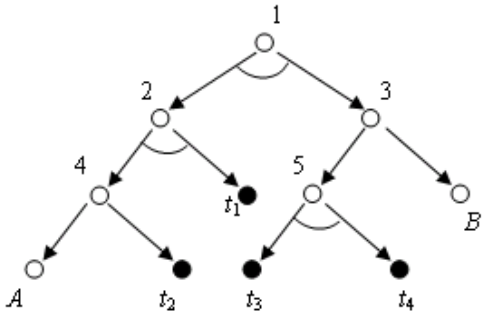
与/或树的宽度优先搜索

例1：设有如图所示的与/或树，其中t_1, t_2, t_3, t_4均为终叶节点，A和B是不可解的端节点。

用与/或树的宽度优先搜索法对该图进行搜索。



例：与/或树的宽度优先搜索



初始化：

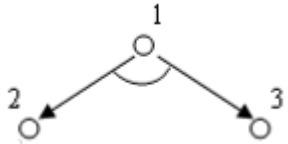
节点1送open表，且不为叶节点

open	close
1	[]

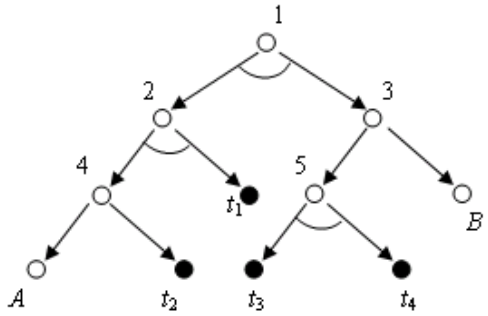
STEP1：

- 扩展节点1，得到节点2、3；
- 节点2，3都不是终叶节点，接着扩展节点2，此时open表只剩节点3.

open	close
1	
2, 3	1
3	1, 2



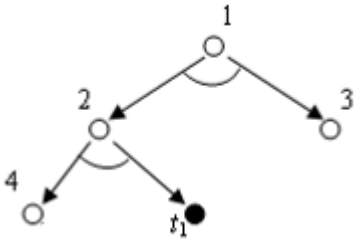
例：与/或树的宽度优先搜索



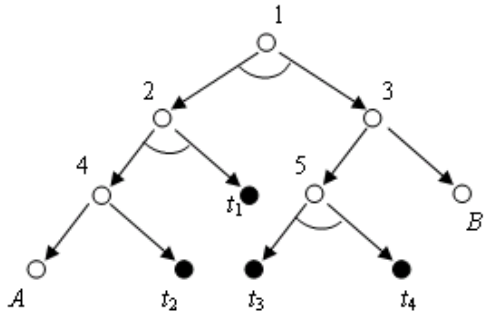
STEP2：

- 扩展节点2后，得到节点4、t1，此时open表中的节点有3，4，t1；
- 节点t1是终叶节点且为可解节点，对其先辈节点进行标示；
- t1的父节点是“与”节点，无法判断节点2是否可解，接着扩展节点3.

open	close
1	
2, 3	1
3, 4, t1	1, 2
4, t1	1, 2, 3



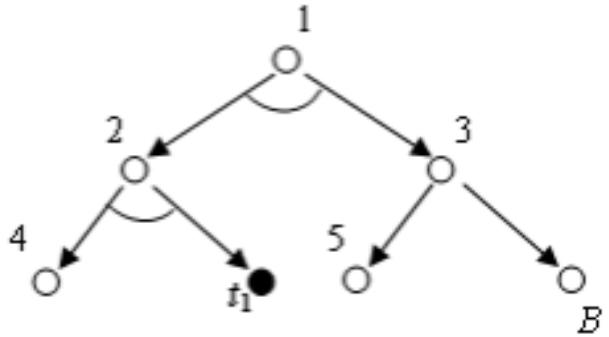
例：与/或树的宽度优先搜索



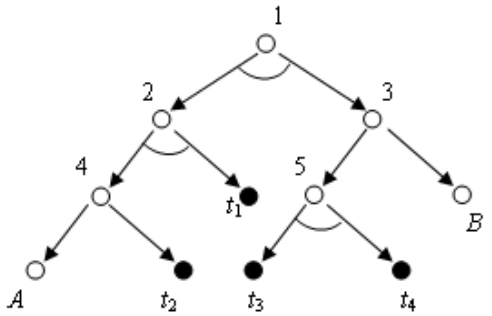
STEP3：

- 扩展节点3后，得到节点5、B；
- 节点5，B都不是终叶节点，接着扩展节点4.

open	close
1	
2, 3	1
3, 4, t1	1, 2
4, t1, 5, B	1, 2, 3
t1, 5, B	1, 2, 3, 4



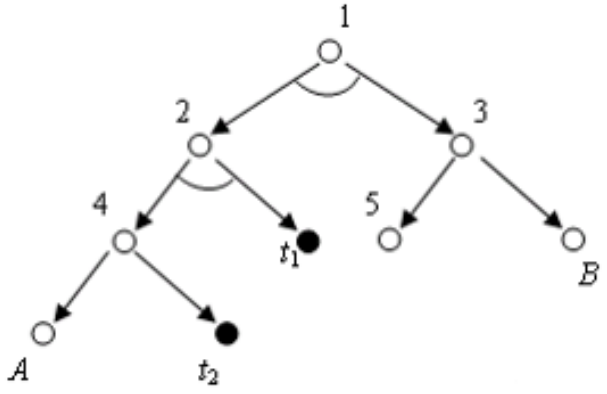
例：与/或树的宽度优先搜索



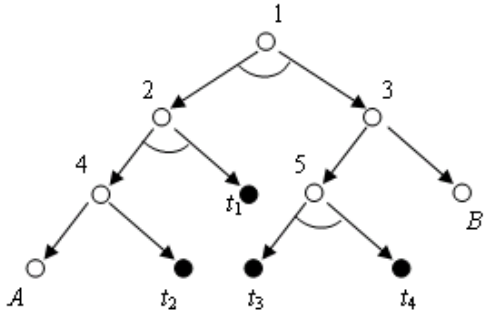
STEP4:

- 扩展节点4后，得到节点A、t2；
- 节点t2是终叶节点且为可解节点，对其先辈节点进行标示；
- t2的父节点是“或”节点，推出节点4可解，推出节点2可解，但不能确定1是否可解；
- 此时节点5是open表中第一个待考察的节点，下一步将扩展节点5.

open	close
1	
2, 3	1
3, 4, t1	1, 2
4, t1, 5, B	1, 2, 3
t1, 5, B, A, t2	1, 2, 3, 4
5, B	1, 2, 3, 4



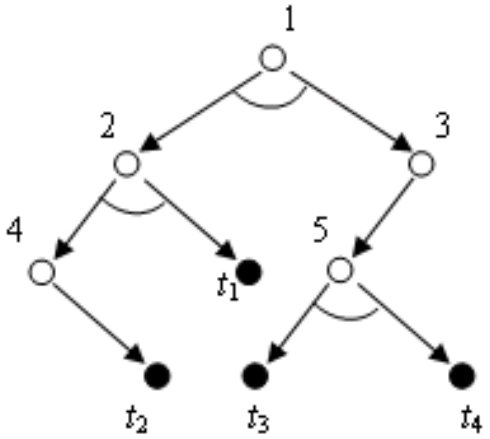
例：与/或树的宽度优先搜索



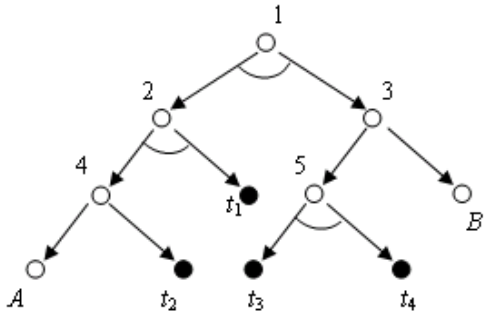
STEP5：

- 扩展节点5后，得到节点t3、t4；
- 节点t3、t4都是终叶节点，且为可解节点，对其先辈节点进行标示；
- 节点5可解，接着推出节点3可解，节点1可解.

open	close
1	
2, 3	1
3, 4, t1	1, 2
4, t1, 5, B	1, 2, 3
5, B	1, 2, 3, 4
B, t3, t4	1, 2, 3, 4, 5

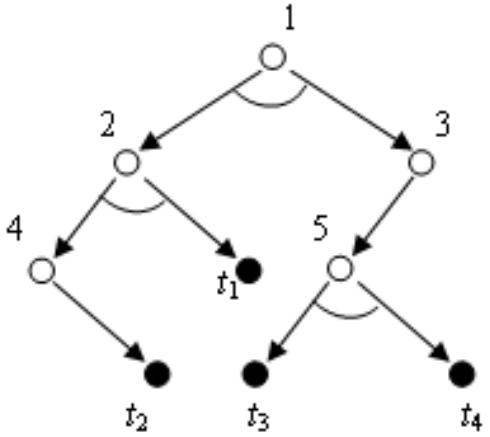


例：与/或树的宽度优先搜索

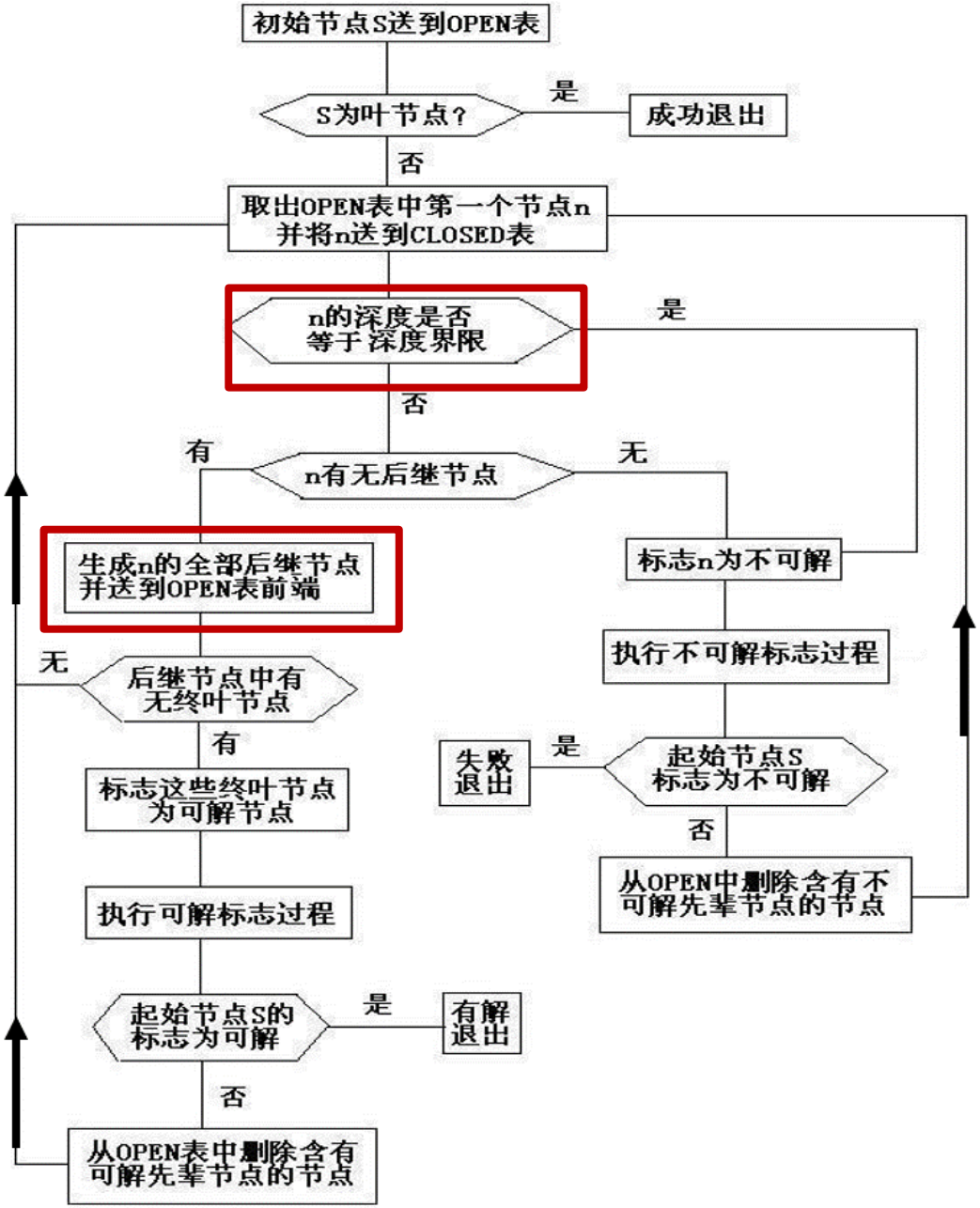


得到节点1可解，算法终止

open	close
1	[]
2, 3	1
3, 4, t1	1, 2
4, t1, 5, B	1, 2, 3
5, B	1, 2, 3, 4
B, t3, t4	1, 2, 3, 4, 5



与或树的深度优先搜索


初始节点S送到OPEN表
S为叶节点？
是
成功退出
否
取出OPEN表中第一个节点n
并将n送到CLOSED表
n的深度是否
等于深度界限
是
否
有
n有无后继节点
无
生成n的全部后继节点
并送到OPEN表前端
标志n为不可解
执行不可解标志过程
无
后继节点中有
无终叶节点
失败
退出
是
起始节点S
标志为不可解
否
有
标志这些终叶节点
为可解节点
从OPEN中删除含有不
可解先辈节点的节点
执行可解标志过程
起始节点S的
标志为可解
是
有解
退出
否
从OPEN表中删除含有
可解先辈节点的节点

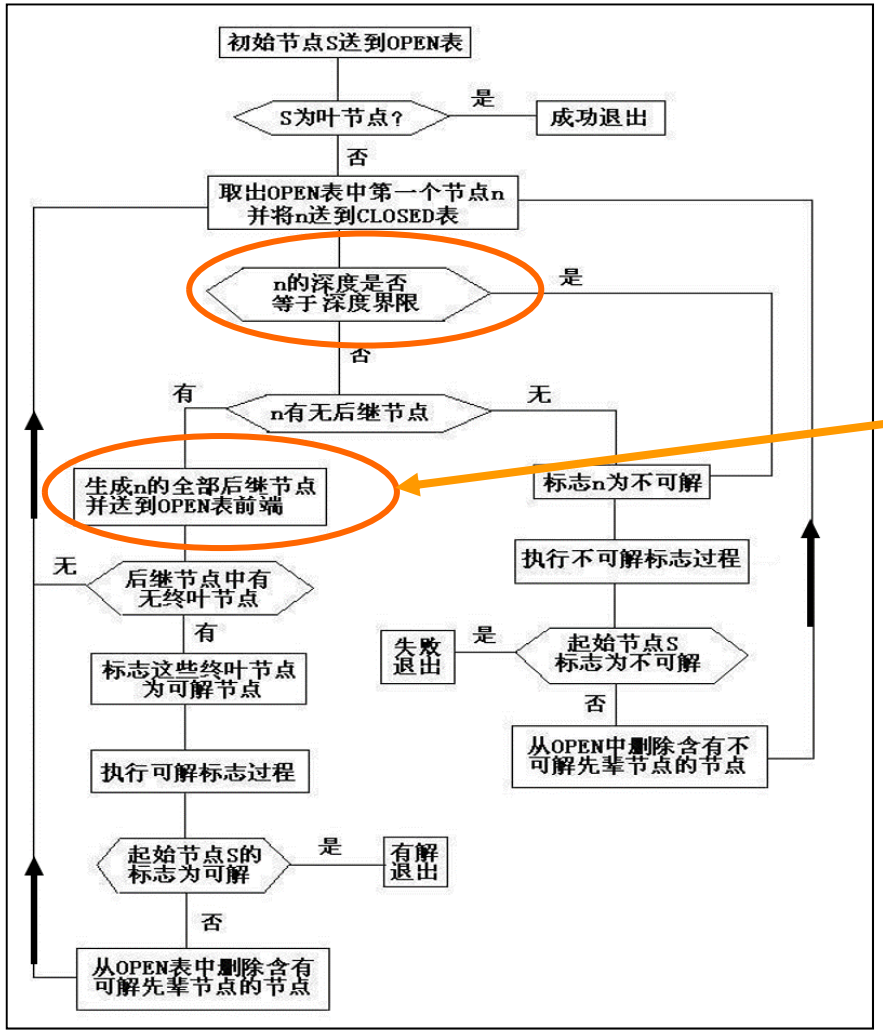
与或树深度优先搜索算法

与宽度优先算法相比，深度优先算法的特殊之处：

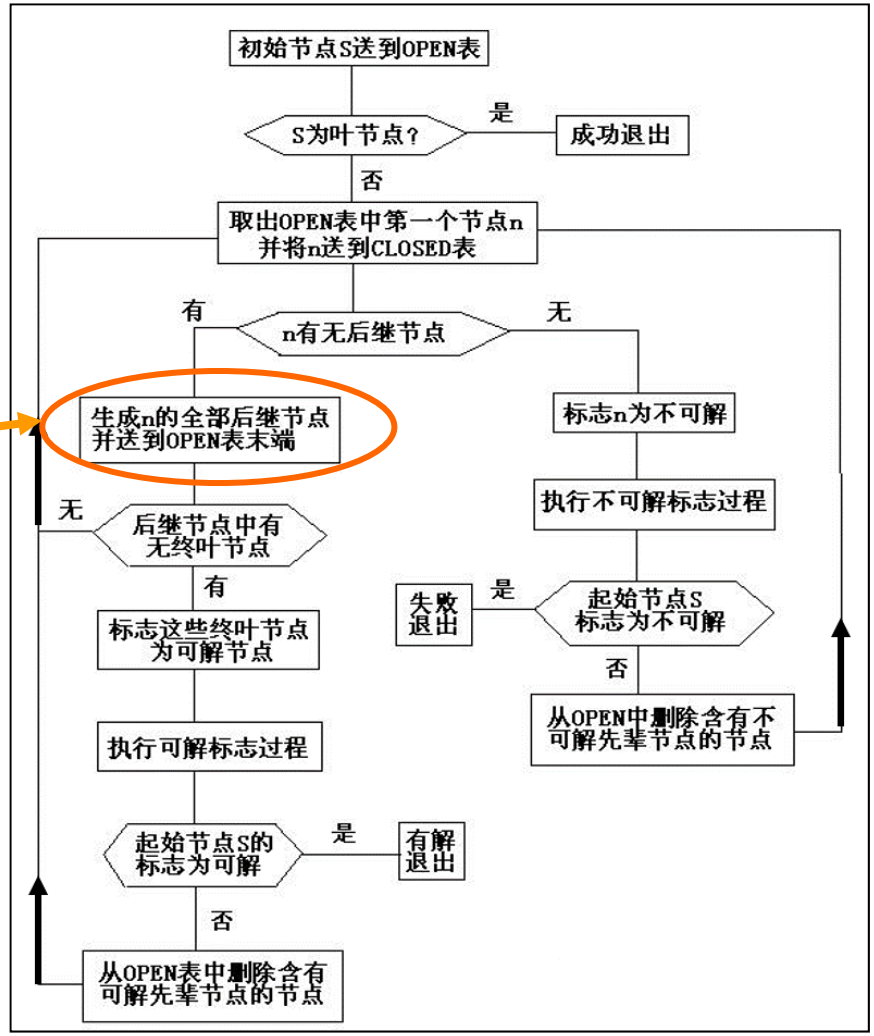
- 第4步要判断从open表取出来的节点的深度。如果等于深度界限，认定它为不可解节点

- 第5步将扩展出来的节点放到open的前端，即open是堆栈

与或树深度优先搜索算法



与或树宽度优先搜索算法



深度优先搜索算法流程

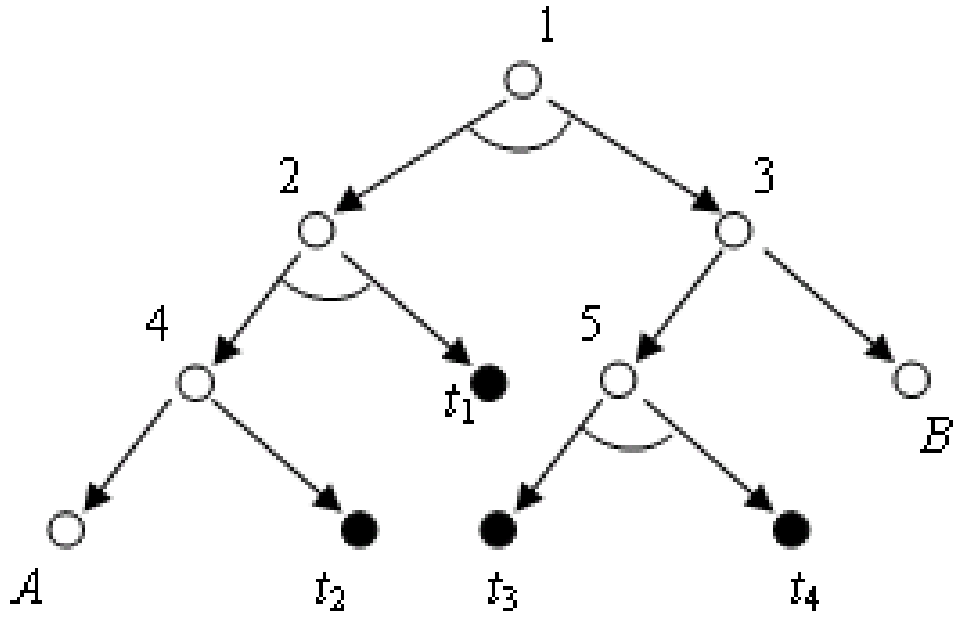
(1) 起始节点S送OPEN表
(2) 若S为叶节点，则成功结束，否则继续
(3) 取出OPEN表第一个节点（记作n），送到CLOSED表
(4) 若节点n的深度等于深度界限，则将n标志为不可解节点，并转10；否则继续
(5) 扩展节点n，生成全部后继节点，置于OPEN表前面，并设置指向n的指针
(6) （分三种情况）
- 如果 n 无后继节点，则标志为不可解节点，并转10，否则继续
- 若有后继节点为叶节点，则将这些叶节点标志为可解节点，并继续；
- 否则转3

(7) 实行可解标志过程
(8) 若起始节点为可解节点，则算法成功结束；否则，继续下一步
(9) 从OPEN表中删除含有可解先辈节点的节点，并转3
(10) 实行不可解标志过程
(11) 若起始节点为不可解，则失败结束，否则，继续下一步
(12) 从OPEN表中删去含有不可解先辈节点的节点
(13) 转3

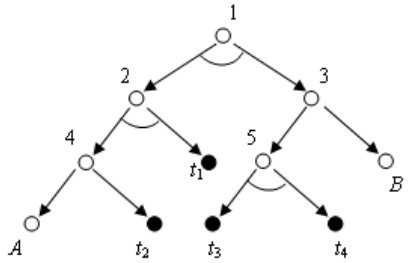
与/或树的深度优先搜索

例2：设有如图所示的与/或树，其中t_1, t_2, t_3, t_4均为终叶节点，A和B是不可解的端节点。

采用与/或树的深度优先搜索法对该图进行搜索。（规定深度界限为4）



与/或树的深度优先搜索



初始化：

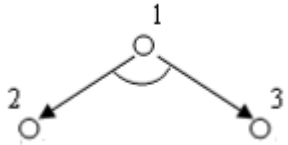
节点 1 送open表，且不为叶节点

open	close
1	[]

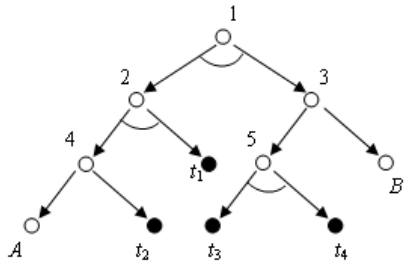
STEP1：

- 扩展节点1，得到节点2、3；
- 节点2，3都不是终叶节点，接着扩展节点2，此时open表只剩节点3.

open	close
1	
2, 3	1
3	1, 2



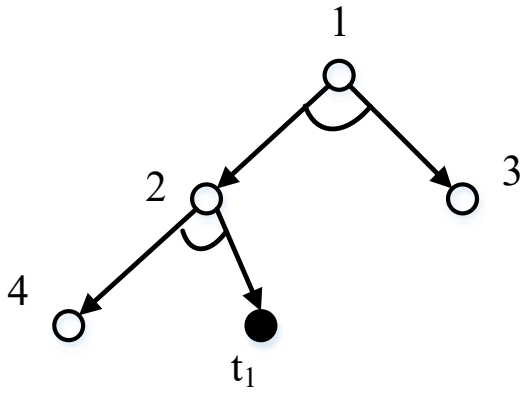
与/或树的深度优先搜索



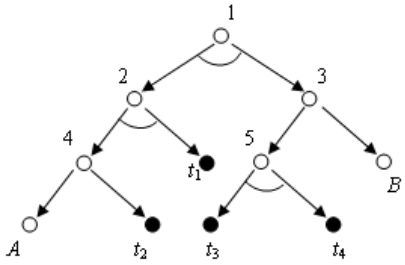
STEP2：

- 扩展节点2后，得到节点4, t1；
- 节点t1不是终叶节点，对其先辈节点进行标示；
- 没有先辈节点被标记成可解节点，不需要调整OPEN表。

open	close
1	
2, 3	1
4, t1, 3	1, 2
t1, 3	1, 2, 4



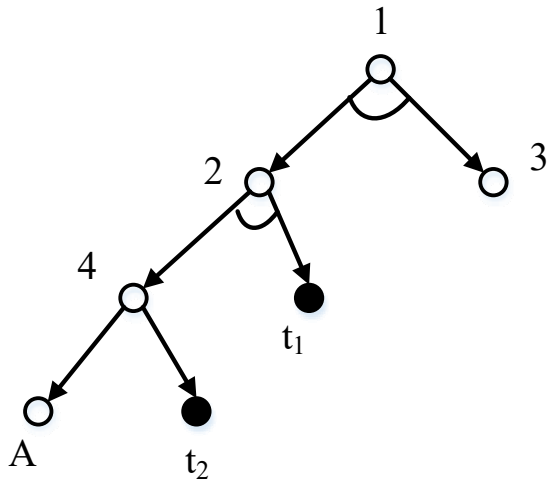
与/或树的深度优先搜索



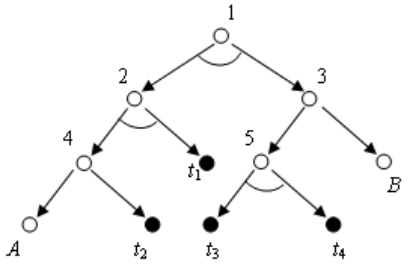
STEP3：

- 扩展节点4后，得到节点A, t2；
- 节点t2是终叶节点，A节点为不可解点，对其先辈节点进行标示；
- 先辈节点2被标记为可解节点，删除OPEN表中的A, t2, t1；

open	close
1	
2, 3	1
4, t1, 3	1, 2
A, t2, t1, 3	1, 2, 4
3	1, 2, 4
	1, 2, 4, 3



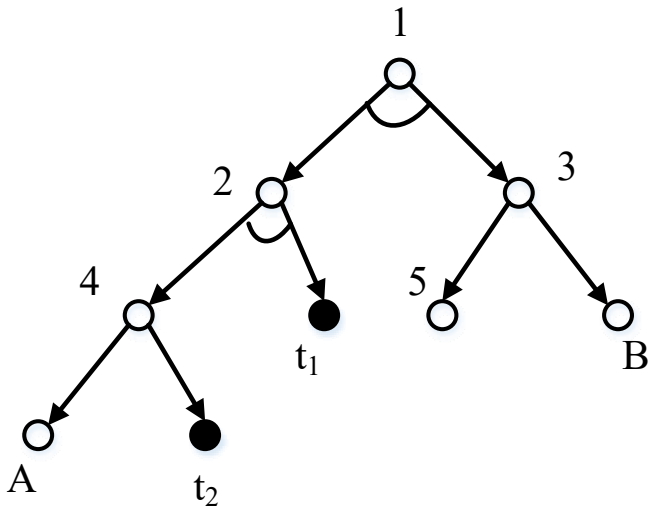
与/或树的深度优先搜索



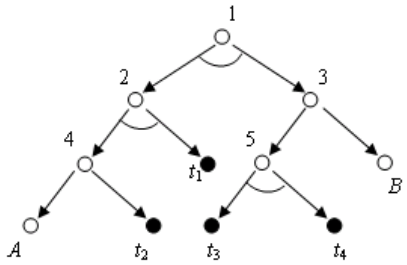
STEP4:

- 扩展节点3后，得到节点5、B；
- 节点B是不可解节点，对其先辈节点进行标示；

open	close
1	
2, 3	1
4, t1, 3	1, 2
3	1, 2, 4
5, B	1, 2, 4, 3
B	1, 2, 4, 3, 5



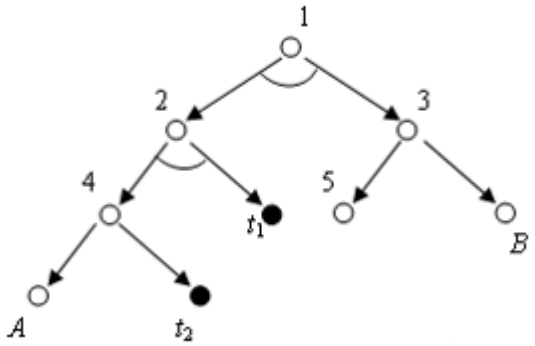
与/或树的深度优先搜索



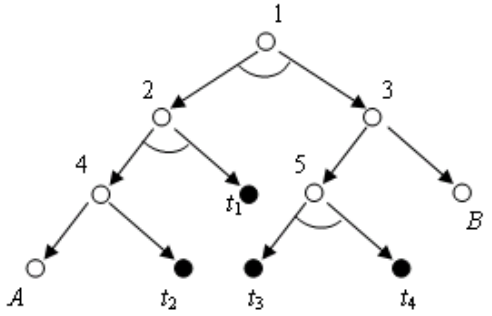
STEP5：

- 扩展节点5后，得到节点t3、t4；
- 节点t3, t4是终叶节点且标示为可解节点，对其先辈节点进行标示；
- 节点1被标记为可解节点

open	close
1	
2, 3	1
4, t1, 3	1, 2
3	1, 2, 4
5, B	1, 2, 4, 3
t3, t4, B	1, 2, 4, 3, 5

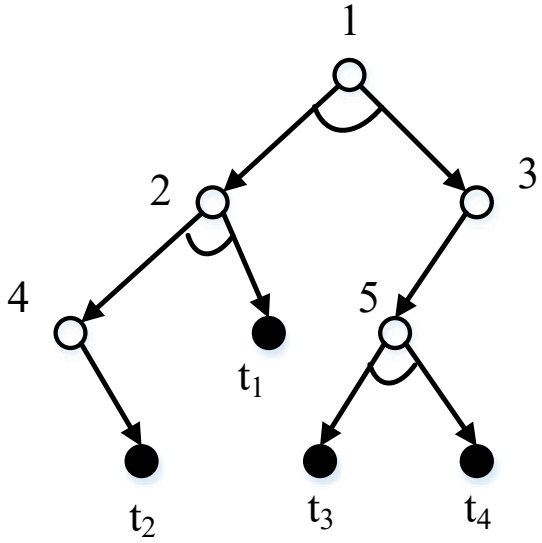


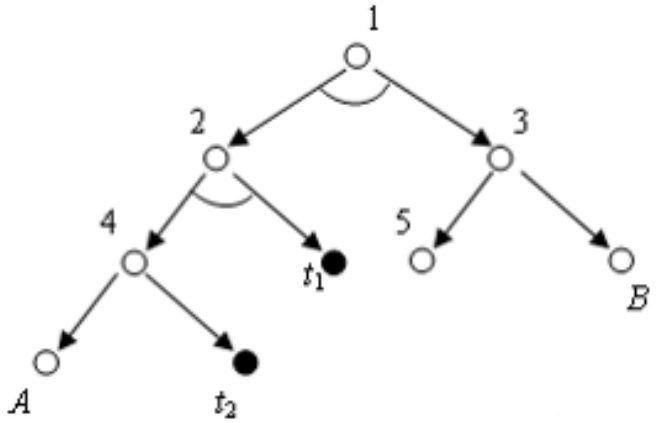
与/或树的深度优先搜索



得到节点1可解，算法终止

open	close
1	
2, 3	1
4, t1, 3	1, 2
3	1, 2, 4
5, B	1, 2, 4, 3
t3, t4, B	1, 2, 4, 3, 5





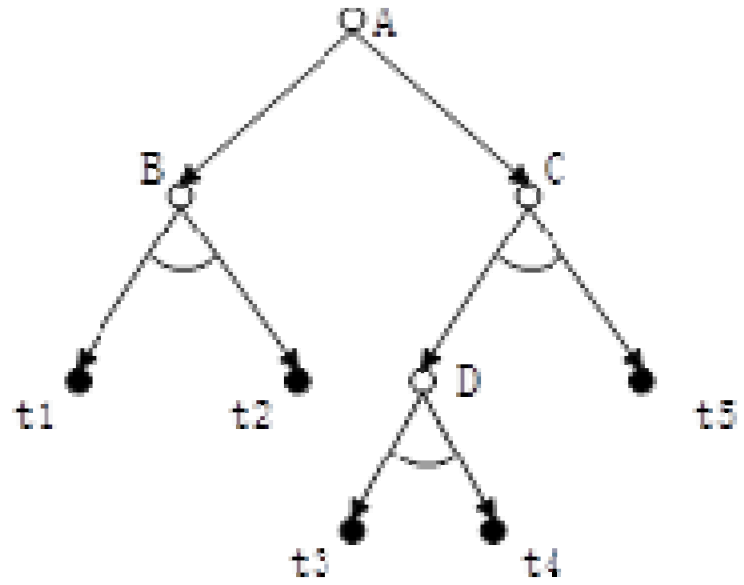
open	close
1	[]
2, 3	1
3, 4, t1	1, 2
4, t1, 5, B	1, 2, 3
5, B	1, 2, 3, 4
B, t3, t4	1, 2, 3, 4, 5

open	close
1	
2, 3	1
4, t1, 3	1, 2
3	1, 2, 4
5, B	1, 2, 4, 3
t3, t4, B	1, 2, 4, 3, 5

与/或树的宽/深度优先搜索

作业

设有如下所示的与或树，请分别用与或树的宽度优先搜索和深度优先搜索求解树。（课堂）



机器博弈

你可曾听说过“深蓝”？

1997年5月11日，IBM开发的“深蓝”击败了国际象棋冠军卡斯帕罗夫。

卡氏何许人也？

- 6岁下棋
- 13岁获得全苏青年赛冠军
- 16岁获世界青年赛第一名
- 1980年他获得世界少年组冠军 (17岁)
- 1982年他并列夺得苏联冠军 (19岁)
- 1985年22岁的卡斯帕罗夫成为历史上最年轻的国际象棋冠军

电脑棋手：永不停歇的挑战！

- 1988年“深思”击败了丹麦特级大师拉森。
- 1993年“深思”第二代击败了丹麦世界优秀女棋手小波尔加。
- 2001年“更弗里茨” 击败了除了克拉姆尼克之外的所有排名世界前十位的棋手。
- 2002年10月“更弗里茨”与世界棋王克拉姆尼克在巴林交手，双方以4比4战平。
- 2003年1至2月“更年少者”与卡斯帕罗夫在纽约较量，3比3战平。

许多人在努力

机器博弈

- 20世纪50年代，有人设想利用机器智能来实现机器与人的对弈。
- 1997年IBM的“深蓝”战胜了国际象棋世界冠军卡斯帕罗夫，惊动了世界。
- 加拿大阿尔伯塔大学的奥赛罗程序Logistello和西洋跳棋程序Chinook也相继成为确定的、二人、零和、完备信息游戏世界冠军
- 西洋双陆棋这样的存在非确定因素的棋类也有了美国卡内基梅隆大学的西洋双陆琪程序BKG这样的世界冠军。
- 2017年5月，在中国乌镇围棋峰会上，AlphaGo与排名世界第一的世界围棋冠军柯洁对战，以3比0的总比分获胜。
- 中国象棋、桥牌、扑克等许多种其它种类游戏博弈的研究也正在进行中。

博弈搜索

- 特征：智力竞技

机器博弈，意味着机器参与博弈，参与智力竞技。

我们这里的博弈只涉及双方博弈，常见的是棋类游戏，如：中国象棋，军旗，围棋等。

- 目标：取胜

取胜的棋局如同状态空间法中的目标状态。

与八数码游戏一样，游戏者需要对棋局进行操作，以改变棋局，使其向目标棋局转移。

然而，八数码游戏只涉及一个主体，不是博弈。

博弈涉及多个主体，他们按规则，依次对棋局进行操作，并且目标是击败对手。

- 方法：Max-Min搜索、α–β剪枝

博弈问题为何可以使用与或图来表示？

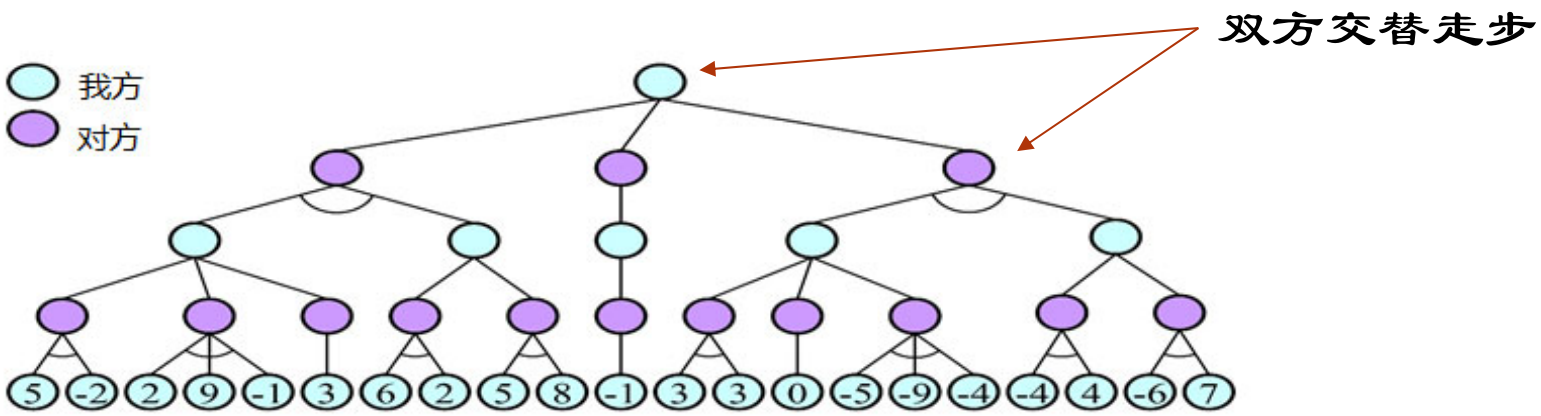
- 当轮到我方走棋时，只需要从若干个可以走的棋中选择一个棋走就可以，从这个意义上来说，这若干个可以走的棋是“或”的关系。
- 而轮到对手走棋时，对于我方来说，必须能够应付对手的每一步走棋，相当于这些棋是“与”的关系。

因此，博弈问题可以看作是一个与或图，但是与一般的与或图不同，是一种具有特殊结构的与或图。

博弈树

博弈树

- 如何根据当前的棋局，选择对自己最有利的一步棋 ？
- 博弈的问题表示：博弈树表示
  - 一种特殊的与或树。
  - 节点：博弈的格局（即棋局），相当于状态空间中的状态，反映了博弈的信息， 并且与节点、或隔层交替出现。



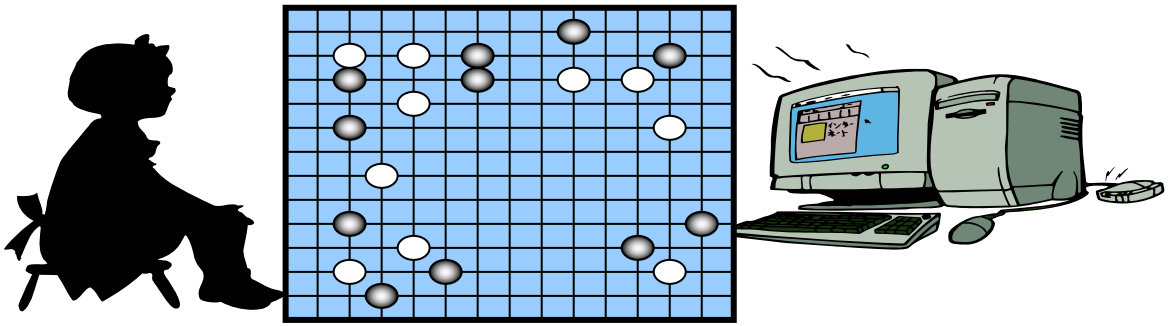
博弈树

- **描述博弈过程的与或树为博弈树，特点为：**
  - 博弈的初始格局是初始节点；
  - 博弈树中，或节点与与节点逐层交替出现，自己一方扩展节点是“或”关系，对方扩展节点是“与” 关系；
  - 所有能使自己一方获胜的终局是本原问题，相应节点为可解节点，所有使对方获胜的终局都是不可解节点

- 博弈树搜索
  - **极大极小（Max-Min）搜索**
  - **α–β剪枝**

双方博弈实例

以围棋为例，竞技的双方分为黑方和白方，由黑方开棋，双方轮流行棋，最终，谁占据的地盘大，谁就成为获胜方。



Max-Min搜索

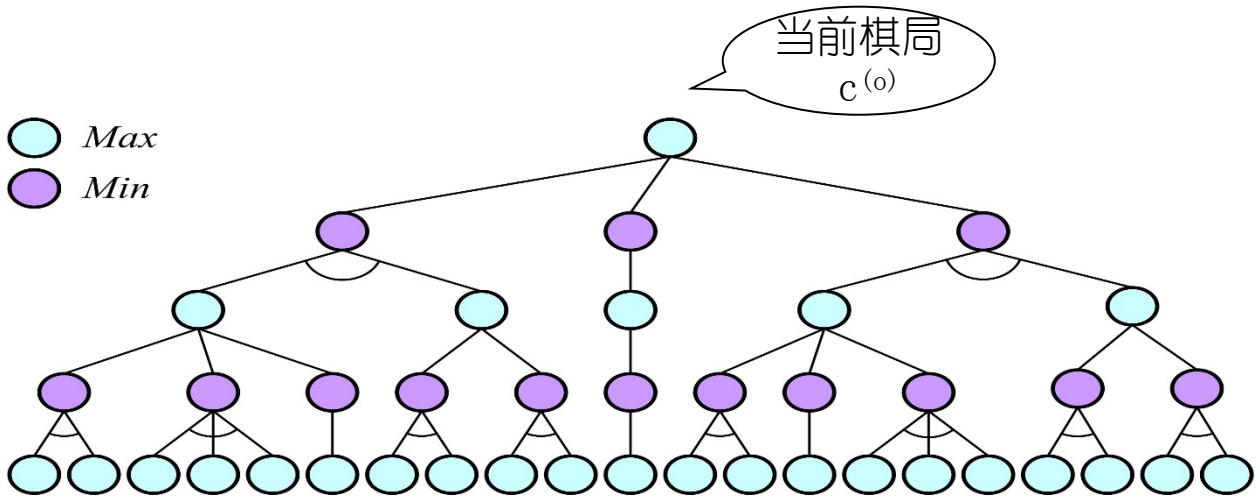
基本思想：

- 目的是为博弈的双方中的一方寻找一个最优行动方案；
- 要寻找这个最优方案，就要通过计算当前所有可能的方案来进行比较；
- 方案的比较是根据问题的特征来定义一个估价函数，用来估算当前博弈树端节点的得分；
- 当计算出端节点的估值后，再推算出父节点的得分（即计算倒推值）；
  - 对或节点，选其子节点中一个最大得分作为父节点的得分
  - 对与节点，选其子节点中一个最小得分作为父节点的得分
- 如果一个行动方案能获得较大的倒推值，则它就是当前最好的行动方案。

Max-Min搜索

Step1.生成k-步博弈树

- Max 代表机器一方 / Min 代表敌方
- 设 Max 面对的当前棋局为 $c^{(o)}$，以 $c^{(o)}$为根，生成 k-步博弈树：



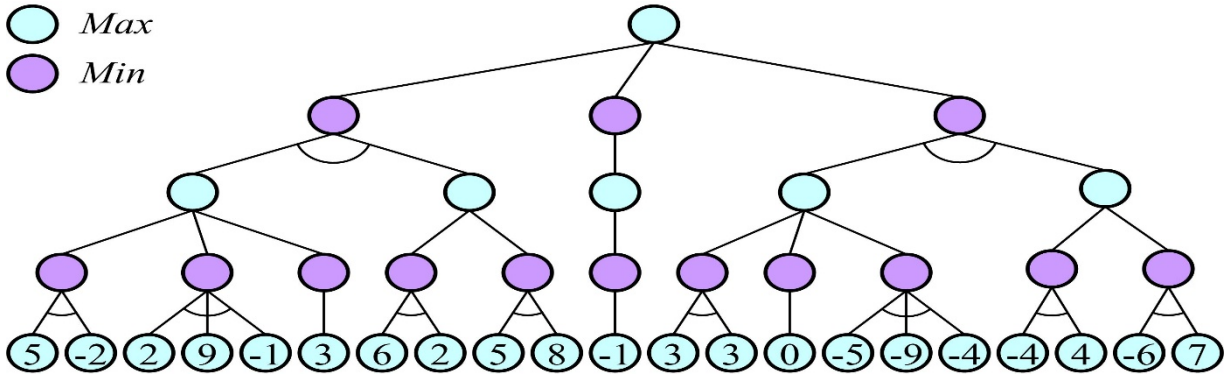
Max-Min搜索

Step2. 评估棋局(博弈状态)

- 估价函数

✓ 为特定的博弈问题定义一个估价函数est(c)，用以评估k-步博弈树叶节点对应的棋局c

✓ est(c)的值越大，意味着棋局c对Max越有利。

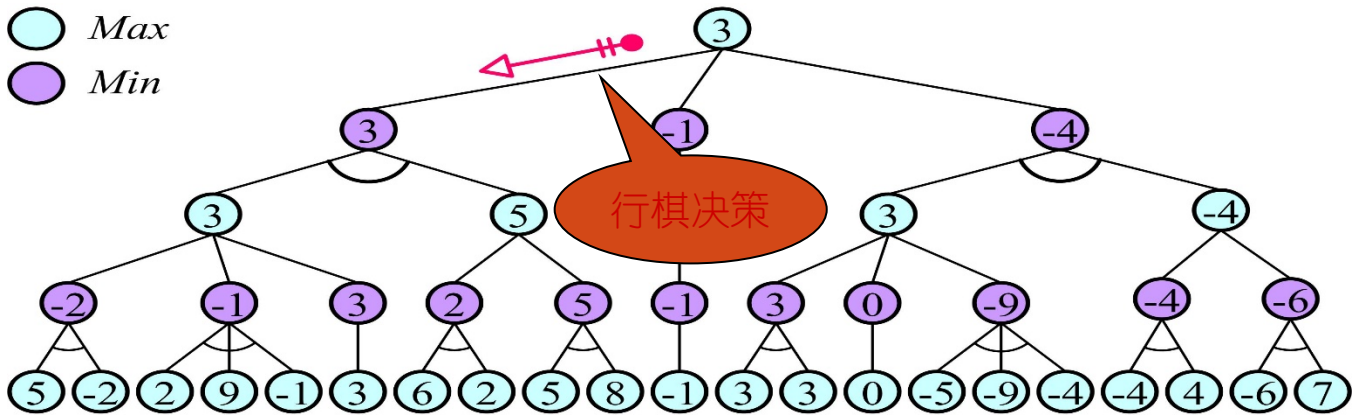


Max-Min搜索

Step3. 回溯评估

- 极大极小运算

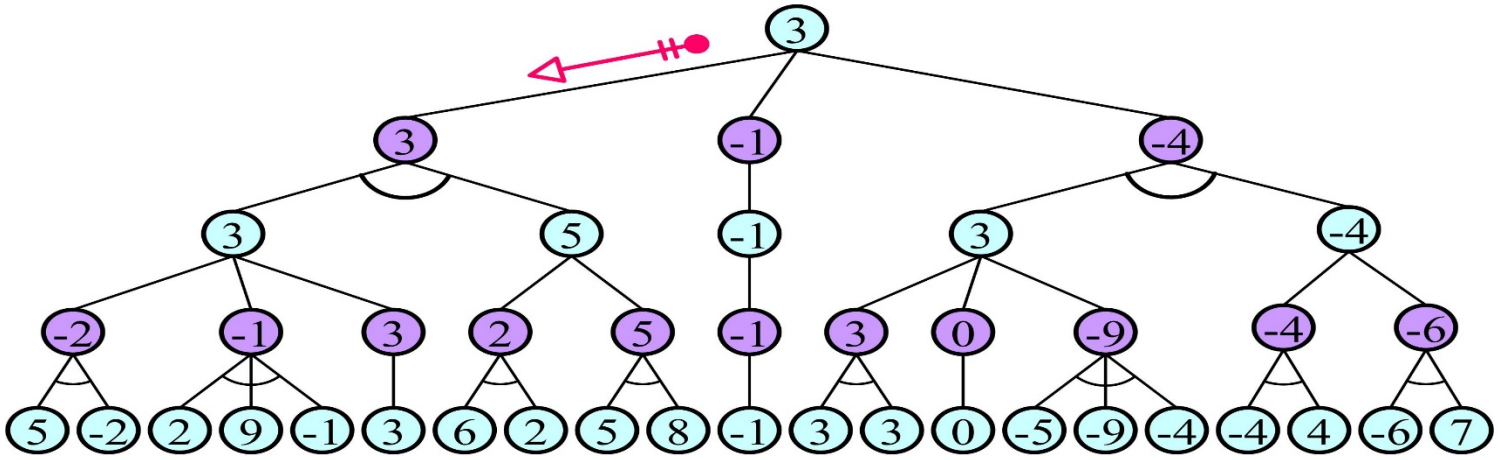
由叶节点向根节点方向回溯评估，在Max处取最大评估值(或运算)，在Min处取最小评估值(与运算)。



Max-Min搜索

Step3. 回溯评估

- 极大极小运算

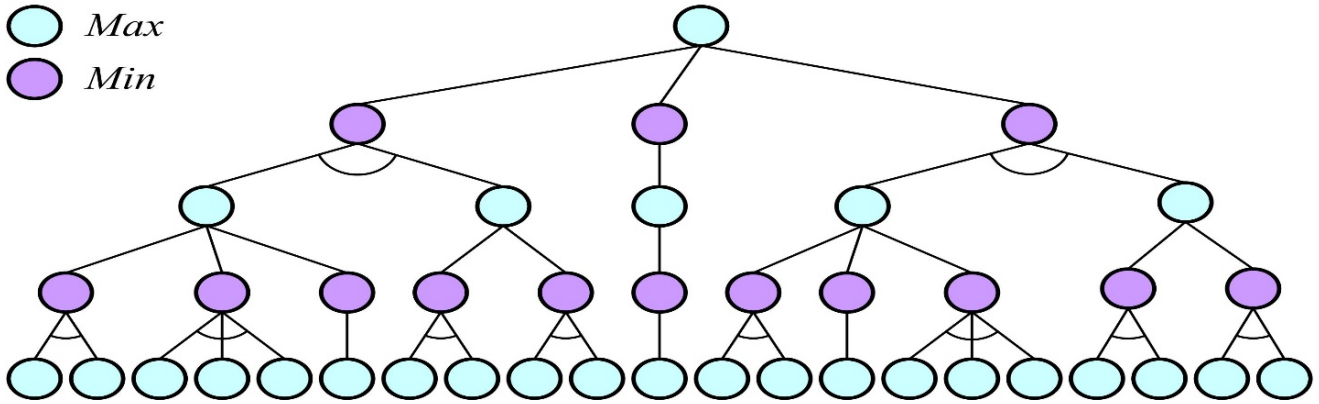


- Max 按取最大评估值的方向行棋

Max-Min搜索

Step4. 递归循环

- Max 行棋后，等待 Min 行棋;
- Min 行棋后，即产生对于 Max 而言新的当前棋局 $c^{(o)}$;
- 返回 Step1，开始下一轮博弈。



Max-Min搜索

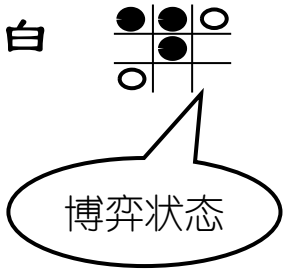
Max-Min搜索流程总结：

- Step1：以 c(o) 为根，生成 k-步博弈树；
- Step2：评估博弈树叶节点对应的博弈状态(棋局)；
- Step3：进行极大极小运算 (Max-Min 运算)；
- Step4：等待 Min 行棋，产生新的 c(o)，返回 step1.

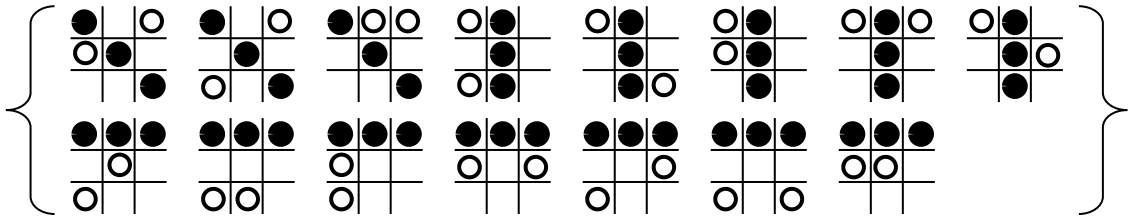
Max-Min搜索

一字棋：

设有 3×3 棋格，Max 与 Min 轮流行棋，黑先白后，先将 3 颗棋子连成一线的一方获胜。



- 一字棋博弈空间：共有 9！种可能的博弈状态
- 一字棋算子空间：博弈规则集合
- 一字棋博弈目标集合 (对 Max而言)：

$\sum c(g) =$ 

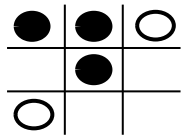
Max-Min搜索

一字棋：

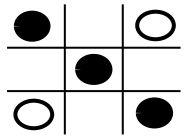
- 定义估价函数：*est*(c)

(1) 对于非终局的博弈状态c，估价函数为：

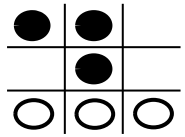
$est(c)$=(所有空格都放上黑色棋子之后，3颗黑色棋子连成的直线总数)－(所有空格都放上白色棋子之后，3颗白色棋子连成的直线总数)

est(c)=3–2=1　　例如：c= 

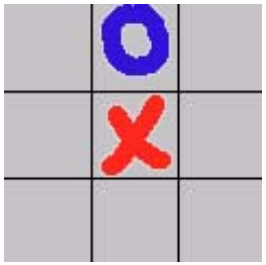
(2) 若 c 是 Max 的胜局，则：

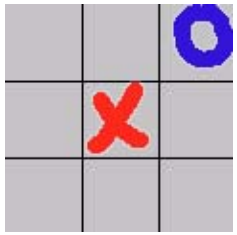
est(c) = +∞　　例如：c= 

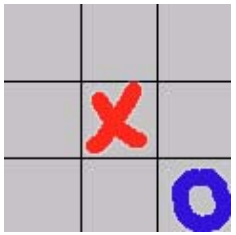
(3) 若 c 是 *Min* 的胜局，则：

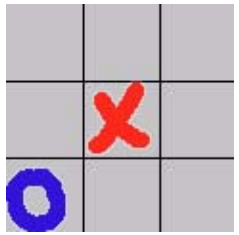
est(c) = –∞　　例如：c= 

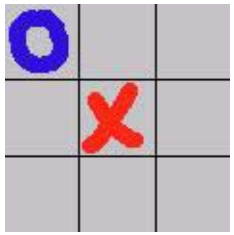
需要说明的是，等价的 (如具有对称性的) 棋局被视为相同棋局。





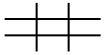


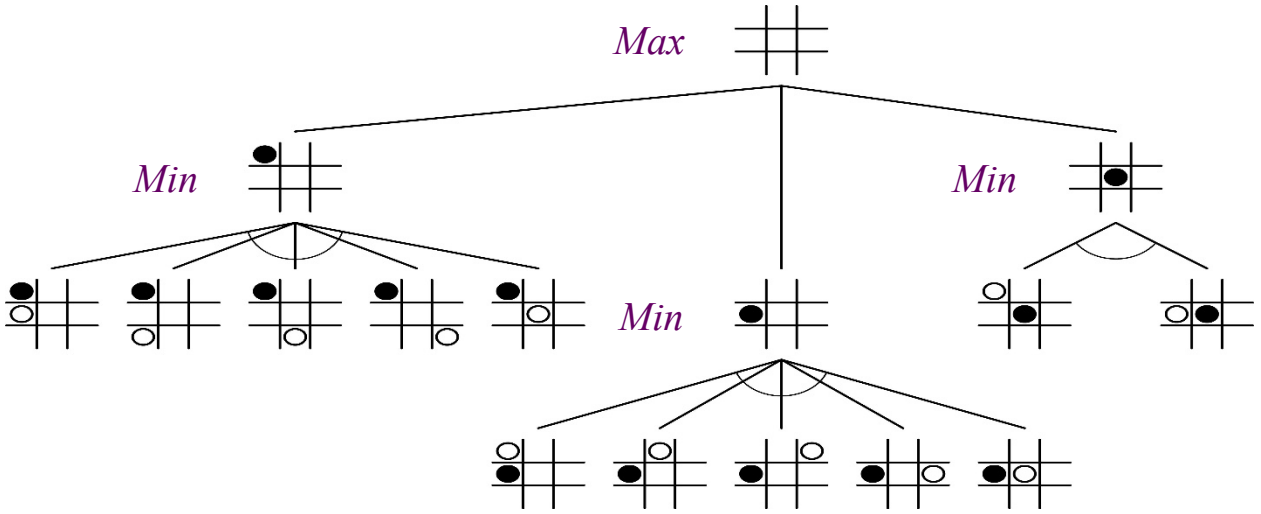




用叉号表示MAX，用圆圈代表MIN。

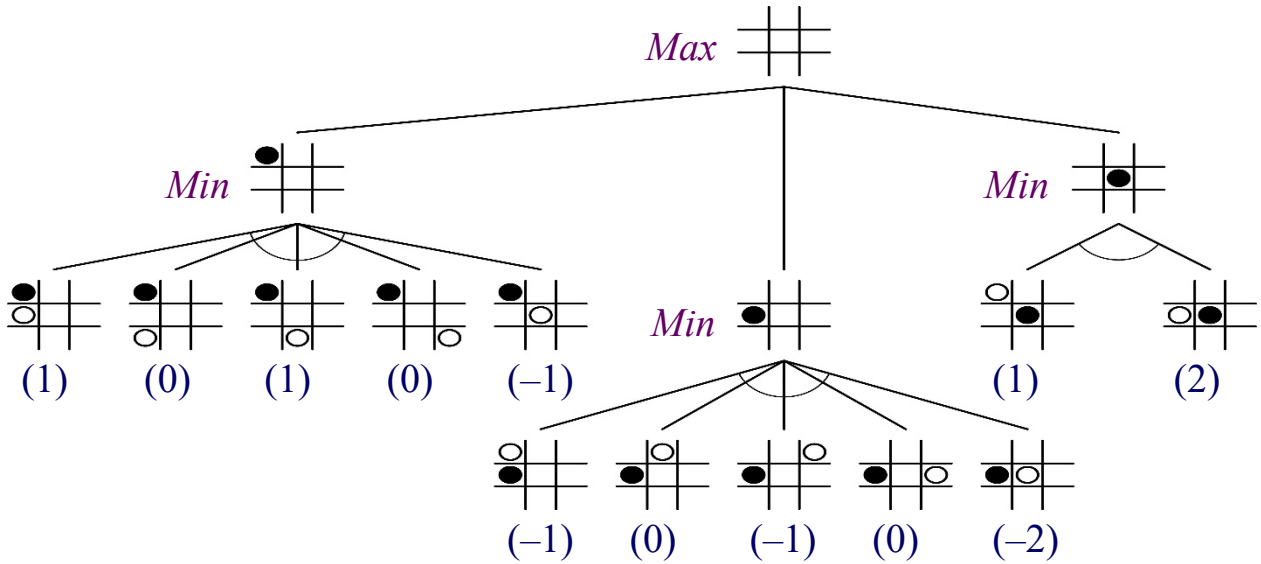
Max-Min搜索

step1. 以 $c^{(0)}$ =  为根，生成2-步博弈树：



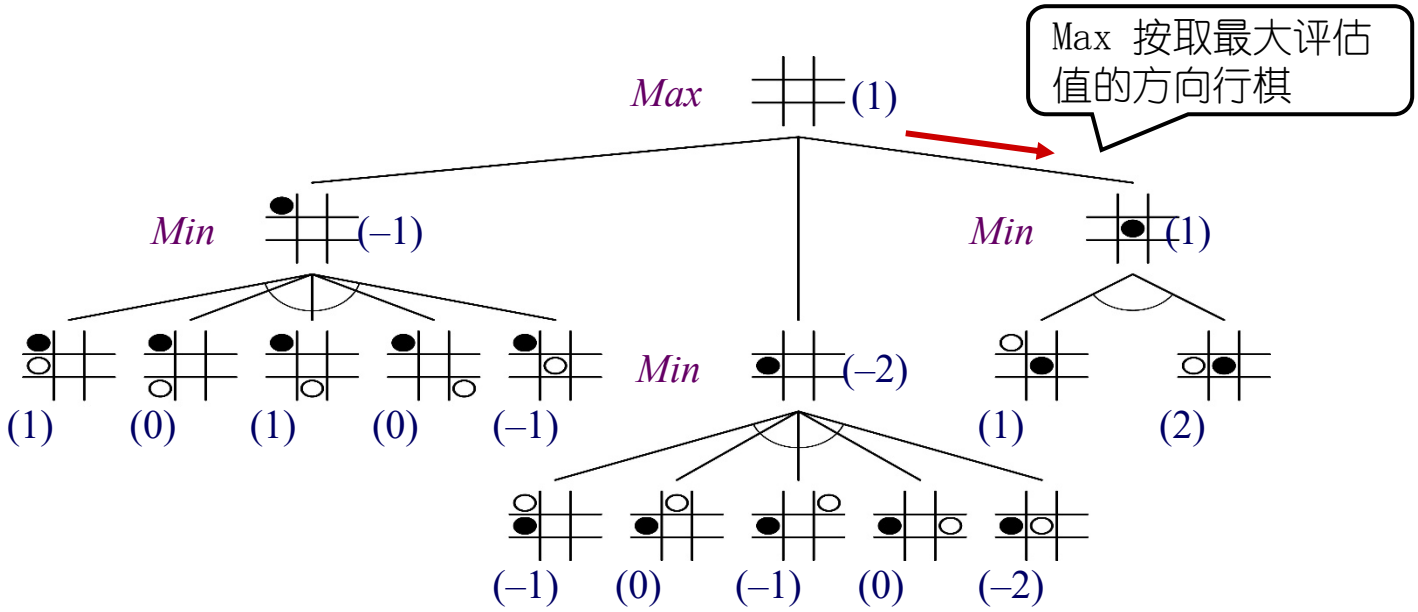
Max-Min搜索

step2. 评估博弈树叶节点对应的博弈状态



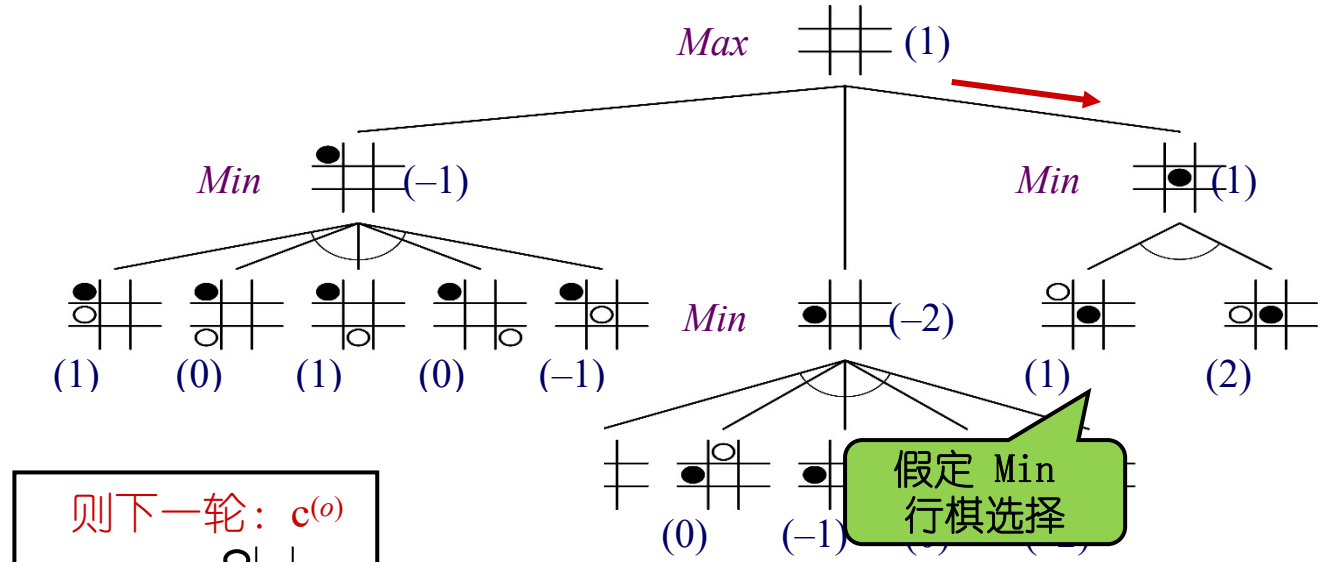
Max-Min搜索

step3. 进行极大极小运算(Max-Min运算)

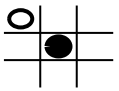


Max-Min搜索

step4. 等待 Min 行棋，产生新的 $c^{(o)}$，返回 step1.



则下一轮：$c^{(o)}$ =



α–β 剪枝

首先分析极小极大分析法效率，上述的极小极大分析法，实际是先生成一棵博弈树，然后再计算其倒推值，至使极小极大分析法效率较低。于是在极小极大分析法的基础上提出了α-β剪枝技术。

α-β剪枝技术的基本思想，边生成博弈树边计算评估各节点的倒推值，并且根据评估出的倒推值范围，及时停止扩展那些已无必要再扩展的子节点，即相当于剪去了博弈树上的一些分枝，从而节约了机器开销，提高了搜索效率。

α–β 剪枝

- 实际上，就博弈而言，人类棋手的思维模式更多地表现出深度优先的特征，而不是宽度优先。
- 因此，采用深度优先搜索策略进行k-步博弈搜索，更符合AI模拟人类智能的原则，这里的k是深度优先搜索的一个自然的深度界限。
- 深度优先搜索策略产生的k-步博弈树是可以剪枝的，因此，搜索空间较小。重要的是，这正是人类棋手约束搜索空间的特征。

α–β 算法的剪枝规则

对于一个**与**节点来说，它取当前子节点中的最**小**倒推值作为它倒推值的**上界**，称此为β值；（β <= 最小值 ）

对于一个**或**节点来说，它取当前子节点中的最**大**倒推值作为它倒推值的**下界**，称此为α值. （α >= 最大值）

规则一 （α 剪枝规则）：

任何**与**节点x的β值如果不能升高其父节点的α值，则对节点x以下的分支可停止搜索，并使x的倒推值为β

规则二 （β 剪枝规则）：

任何**或**节点x的α值如果不能降低其父节点的β值，则对节点x以下的分支可停止搜索，并使x的倒推值为α

由规则一形成的剪枝被称为 **“α 剪枝”**，而由规则二形成的剪枝被称为 **“β 剪枝”**。

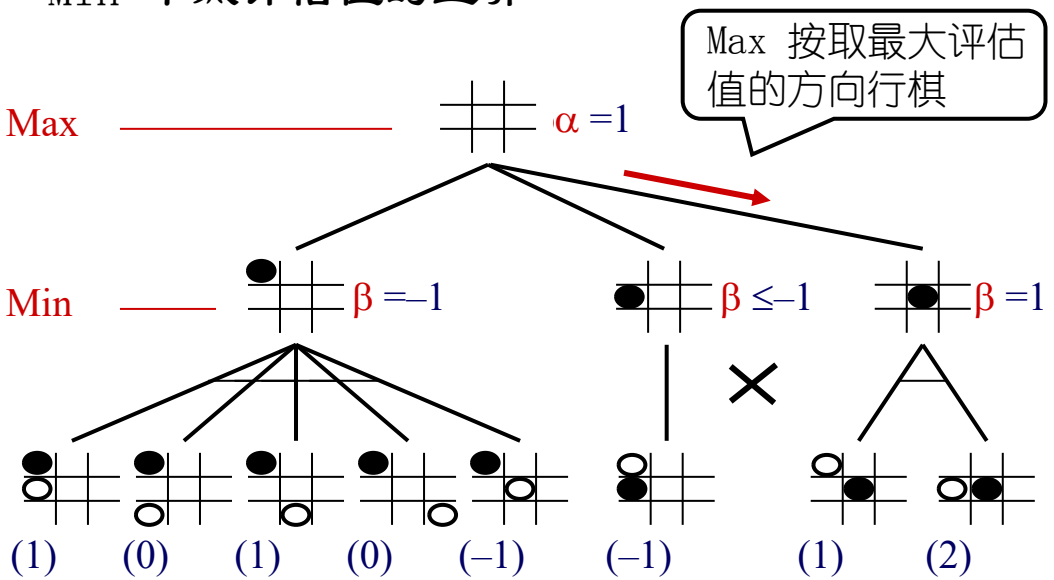
α–β 剪枝

一字棋：

- 搜索策略：k-步博弈；深度优先；每次扩展一个节点；一边扩展一边评估。
- 在 α –β 算法中：

  α：Max 节点评估值的下界

  β：Min 节点评估值的上界



一字棋的 α–β 搜索过程

作业

- 设有如下图所示的博弈树，其中最下面的数字是假设的估值。该博弈树做如下工作：

①计算各节点的倒推值。

②利用 $\alpha-\beta$ 剪枝技术剪去不必要的分枝。