第八章 分支限界法

目录

- 8.1 一般方法
- 8.2 LC-检索
- 8.3 15-谜问题
- 8.4 求最小成本的分支限界法
- 8.5 带有期限的作业调度问题
- 8.6 货郎担问题
- 8.7 小结

8.1 一般方法

- 方法适用的问题特点
- 分支限界法的基本思想
- 算法8.1 分支限界法的抽象化描述
- 分支限界法的不同检索方式
- 理解FIFO-分支限界法

方法适用的问题特点

- 与回溯法类似，分支限界法同样适用于求解组合数较大的问题，特别是组合优化问题(求最优解)。
- 分支限界法中，解向量的表达、显式约束条件、隐式约束条件、解空间和解空间树等概念均与回溯法相同
- 两者主要区别在于E-结点(即扩展结点)处理方式不同

清华大学出版社出版的屈婉玲等编著的《算法设计与分析》中认为：“分支限界是回溯算法的变种”

分支限界法的基本思想

- 针对问题定义解空间树结构：元组、显式约束条件、隐式约束条件。
- 检验问题满足多米诺性质。 找一个可行解
- 假设当前寻找一个答案结点，按下列方式搜索解空间树：
  - 如果根结点T是答案结点，输出T，操作结束；否则令T是当前扩展结点E。
  - 生成E的所有儿子结点，判断每个儿子结点X：
    - 如果X是答案结点，输出到根的路径，操作结束；
    - 如果X满足限界函数B，则将X添加到活结点表中；否则舍弃X。
  - 从活结点表中选出下一个结点成为新的E-结点，重复上述操作。如果活结点表为空，则算法以失败结束。

限界函数剪枝作用：避免生成那些不包含可行解的子树。

算法8.1 分支限界法的抽象化描述

找一个可行解

- ADD(X):将X添加到活结点表中
- LEAST(E):从活结点表中选中一个结点赋值给E，并从表中删除该结点

```
procedure BB(T)
if T是答案结点 then 输出T; return endif
E←T
将活结点表初始化为空
loop
    for E的每个儿子X do
        if X是答案结点 then 输出从X到T的那条路径; return; endif
        if B(X) then call ADD(X);PARENT(X)←E endif
    repeat
    if 表中不再有活结点 then print(“no answer node”); return; endif
    call LEAST(E)
repeat
end BANDB
```

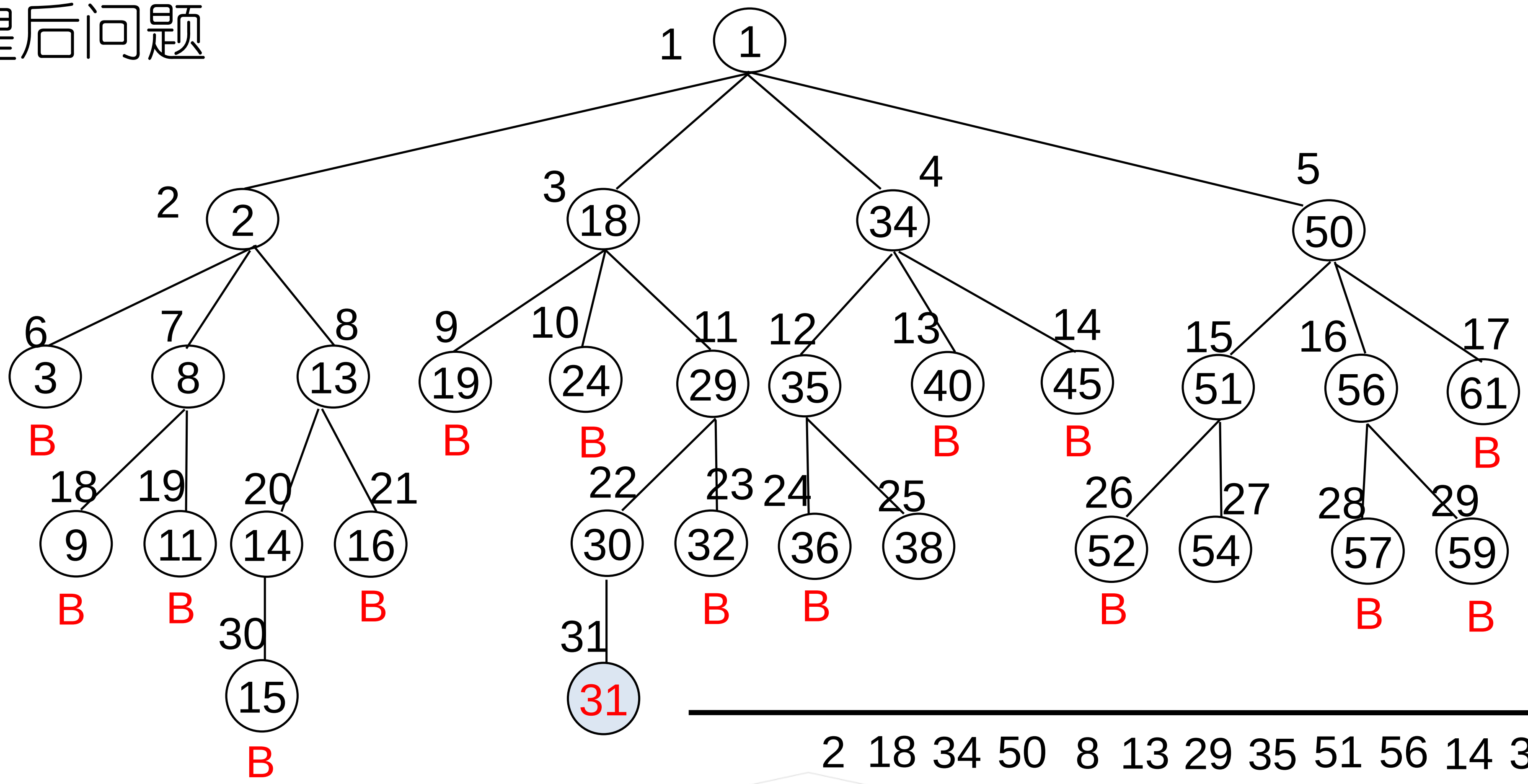
思考：如果想获得所有可行解，算法怎样设计？

分支限界法的不同检索方式

- 根据活结点检索次序，分支限界策略可以分为：
  - 顺序队列式
    - 活结点依赖进入活结点表的顺序从活结点表中被选出
    - 常见先进先出方式(FIFO)，活结点表采用队列实现。
  - 优先队列式
    - 活结点依赖成本估计函数ĉ，ĉ最小/最大的活结点优先从活结点表中被选出。
    - 活结点表采用极小堆/极大堆实现。
- 若LEAST遵循：
  - FIFO-顺序队列式：算法即为FIFO-检索
  - 极小堆优先队列式：算法即为LC-检索

理解FIFO-检索

- 4-皇后问题



2 18 34 50 8 13 29 35 51 56 14 30 38 54

FIFO-分支限界法一共处理了31个结点，回溯法一共处理了16个结点。

8.2 LC-检索(least cost)

- LC-检索的优点
- 成本函数c的量化方法
- 区分状态空间树中结点X
- 成本函数c定义
- 成本估计函数ĉ定义
- LC-检索总结

LC-检索的优点

- 在FIFO-分支限界法中，对下一个E-结点的选择是死板的、盲目的，对于可能快速检索到答案结点的结点没有给出任何优先权。
- 理想状态下，对活结点表使用一个“有智力的”成本函数c来选取下一个E-结点，从而加快到达答案结点的检索速度。

给那些可能导致答案结点的活结点赋以优先次序

成本函数c的量化方法

- 令X是当前结点，c(X)定义为以X为根的子树中的最小成本值
  - 方法1：寻找生成结点数目最少的答案结点
    - 基于X在生成一个答案结点之前需要生成的结点数定义
  - 方法2：寻找路径长度最短的答案结点
    - 基于距离X最近的那个答案结点的路径长度定义
  - 方法3：寻找使目标函数取极值的答案节点(最优解)
    - 基于问题描述中的目标函数定义

本章中，问题不存在目标函数时，采用方法2定义c函数

成本函数c定义

- 对状态空间树里的解状态结点X定义真实成本函数

- cost(X)= $\begin{cases} \text{根到达X的真实成本/代价} & \text{X是答案结点} \\ \infty & \text{X不是答案结点} \end{cases}$

如路径长度、目标函数值等

- 从上帝视角对状态空间树里的任意结点X定义成本函数

- c(X)= $\begin{cases} \text{cost(X)} & \text{叶节点X是答案结点} \\ \min\{\text{cost(P)}\mid\text{答案结点P} \in \text{子树X}\} & \text{子树X中包含答案结点} \\ \infty & \text{子树X不包含答案结点} \end{cases}$

k-元组表达

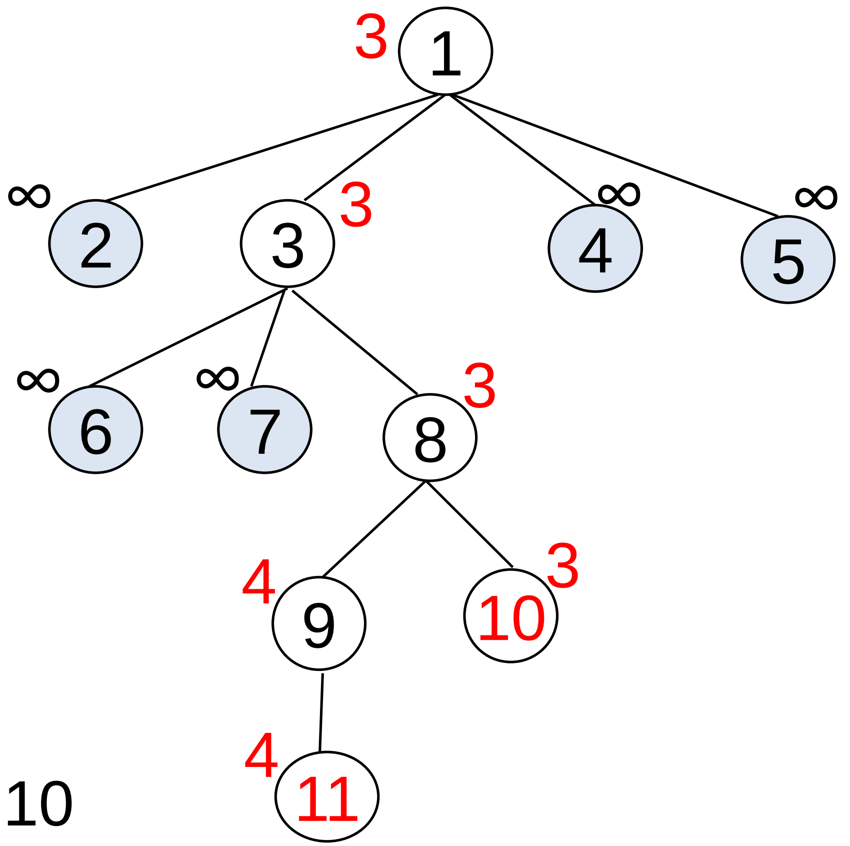
思考：在什么样的状态空间树中会出现答案结点X不是叶结点的情况？

注意：在我们探讨c(X)时，我们假设答案节点已经找到，检索树已经生成，一切都是在已知的情况下进行讨论。

换句话说，c(X)是节点X的真实成本函数

成本函数c的例子

- 设c(X)=到达答案结点的最短路径长度
- 图中
  - 蓝色结点表示不能到达一个答案结点
  - 叶结点10和11是答案结点
  - c(10)=3，c(11)=4
- 基于c对活结点表检索，可快速找到结点10



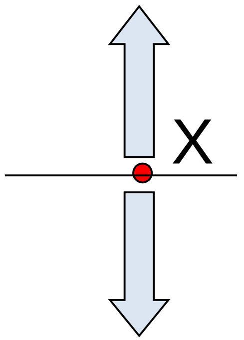
叶节点X是答案结点时，c(X)=根结点到X的真实成本

子树X中包含答案结点时，c(X)=子树X中最小成本答案结点的真实成本

成本函数c的问题

- 要得到函数c很困难
  - c(X)基于答案结点的真实成本定义，为求出该值，需要生成状态空间树。
  - c的计算工作量与原问题具有相同的复杂度。

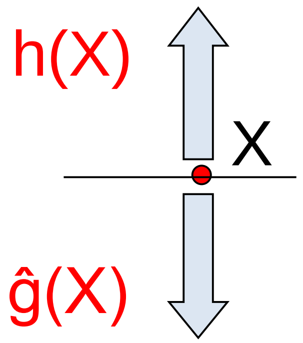
- 基于c，定义一个易于计算的成本估计函数ĉ，来替代c对活结点表进行检索。定义ĉ(X)时：
  - 很容易确定根到X已经产生的成本
  - 很容易估计X到一个答案结点可能会产生的成本



这是一个富于创造性的工作

成本估计函数ĉ定义

- 成本估计函数 $\hat{c}(X)=f(h(X))+\hat{g}(X)$
  - $h(X)$：根结点到结点X的成本
  - $\hat{g}(X)$：子树X中，X到最小成本答案结点的估计成本
  - f：为调整h和ĝ在成本估计函数ĉ中的影响比例而定义的非负函数



- 对成本估计函数ĉ的要求
  - 易于计算，能够替代c对活结点表进行检索
  - $\hat{c}(X)\le c(X)$
  - 当叶节点X是答案节点时，$c(X)=\hat{c}(X)$

LC-检索

$\hat{c}(X)=f(h(X))+\hat{g}(X)$

- LC-检索(Least Cost search)：选取成本估计函数ĉ的值最小的活结点作为下一个E-结点。
- BFS-检索(广度优先)：f(h(X))=根到结点X的路径长度，$\hat{g}(X)=0$

BFS是LC-检索的特殊情况

思考：只用$\hat{g}(X)$为结点排序是否适合? 即f(h(X))=0。

$\hat{c}(X)=\hat{g}(X)$时

- 假设子树X的儿子结点是Y，容易存在$\hat{g}(Y) \le \hat{g}(X)$
  - 若活结点表中其他结点成本估计值均大于$\hat{g}(X)$，则X成为当前E-结点，Y在X之后成为新的E-结点。该过程直到子树X全部检索完毕才可能生成子树X以外的结点。
- 后果
  - 只考虑未来的可能的代价，导致算法偏向于纵深方向检索，可能会丢失掉更靠近根的答案结点，导致结果不理想。

乐观的冒进

LC-检索总结

从T经X到达

- T是状态空间树， X 是T中任意结点
- c(X)表示以X为根的子树中最小成本的答案结点的成本。找到这样一个易于计算的c是很困难的，因此基于c设计一个成本估计函数ĉ
- ĉ(X)=f(h(X))+ ĝ(X)要易于计算，且ĉ(X)≤c(X)；当叶节点X是答案节点时，c(X)=ĉ(X)
- 在LC-检索中，算法利用ĉ对活结点表进行检索。即优先选择更靠近答案结点，同时又离根结点较近的结点

LC-检索的优点

8.3 15-谜问题

- 问题描述
- 状态空间树
- 函数定义
- 判定定理
- FIFO-检索
- 深度优先检索
- LC-检索

问题描述

- 在一个分成16格的方形棋盘上放有15块编了号码的牌，如(a)所示，要求通过一系列合法的移动转换成(b)所示那样的目标排列。

若当前牌邻接有空位置，则可将牌移动到空位置。

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

(a)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(b)

状态空间树

- 问题状态
  - 棋牌布局状态
  - 初始排列称为初始状态
  - 目标排列称为目标状态
- 状态空间树
  - 棋牌每移动一次，就会产生一个新的布局状态
  - 由所有可从初始状态经过一系列合法移动到达的状态构成
  - 当前结点X的儿子结点是X通过一次合法移动到达的布局状态。

初始状态满足某些条件时，才能达到目标状态(b)

函数定义

- 对棋盘的方格位置编上1~16的号码，编号顺序如图(b)所示，空格位是位置16
- 设POSITION(i)是棋牌i在初始状态时的位置号，1≤i≤16，POSITION(16)表示空格的位置
- 设LESS(i)是牌面上j<i且POSITION(j)>POSITION(i)的j的数目，即反序的数目

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(b)

位置16

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

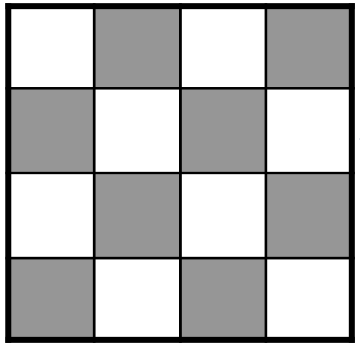
(a)

牌面i	8	9	10	13	14	15	16
LESS(i)	1	1	1	1	1	1	9

其他牌面的LESS值为0

判定定理

状态(a)有∑LESS(i)+X=15+1 是偶数，可达状态(b)



(c)

在初始状态下，如果空格在(c)的阴影位置中，则令X=1；否则令X=0。

1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

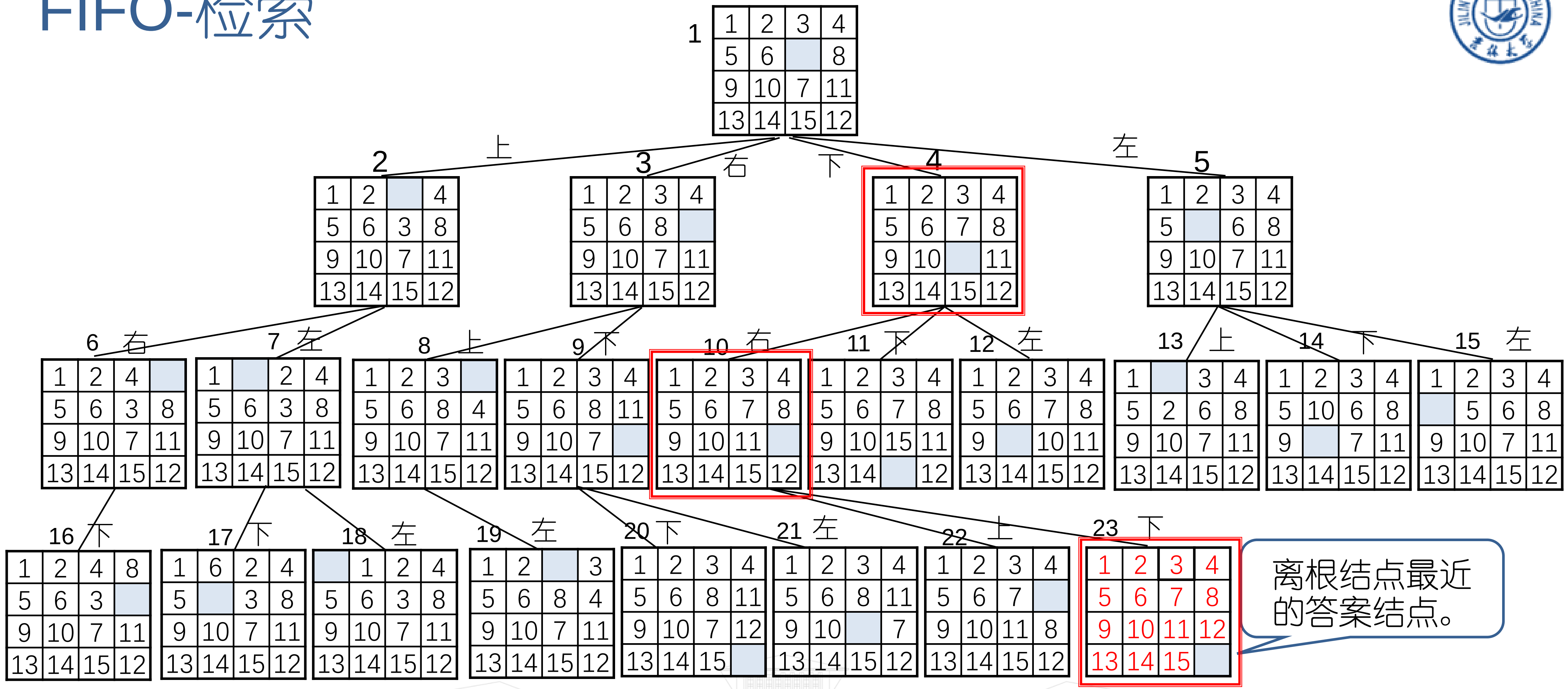
(a)

初始状态判定定理：

当且仅当初始状态的$\sum_{1\le i\le 16} LESS(i)+X$是偶数时，图(b)所示的目标状态可由此状态到达

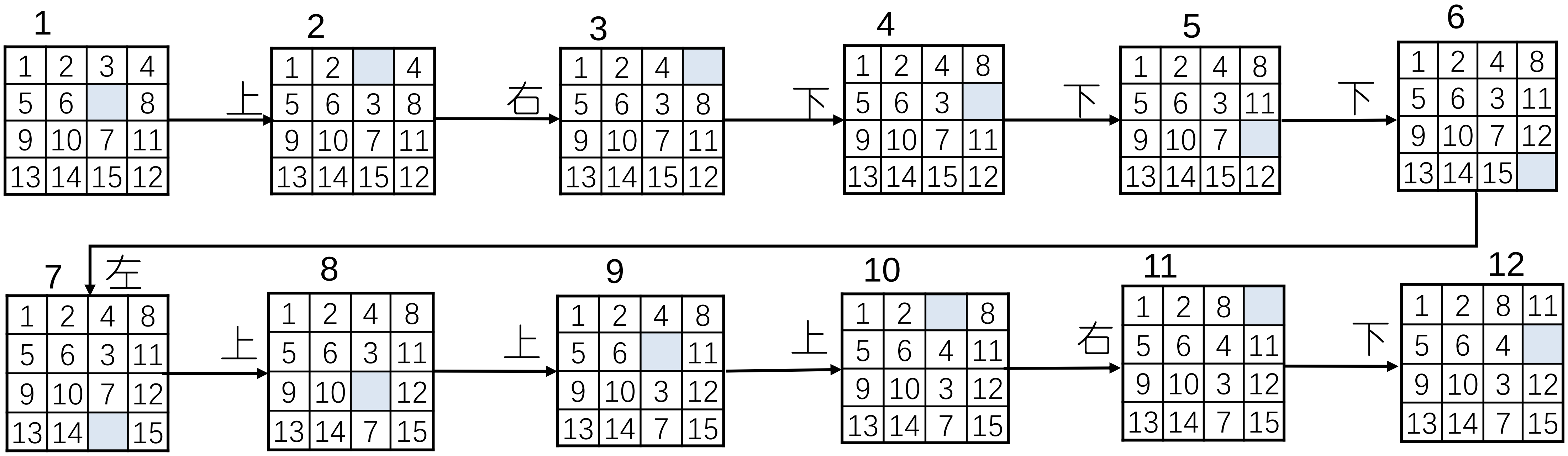
由于移动牌与移动空格等效，因此状态空间树中，边表示为空格的一次合法移动，按上、右、下、左的顺序进行。

FIFO-检索



能找到离根最近的答案结点，不管开局如何（不关心问题的具体实例），总是按千篇一律的顺序移动。

深度优先检索



不管开局如何（不关心问题的具体实例），总是采取由根开始的最左路径，呆板而盲目。搜索过程中有可能远离目标。

LC-检索

离根结点最近的答案结点

- 15谜问题的c(X)和ĉ(X)定义
  - 定义c(X)：从初始排列到达目标排列时，棋牌最少移动次数
  - 基于c定义ĉ(X)：ĉ(X)=f(X)+ĝ(X)
    - f(X)：从初始排列到X时，棋牌已经移动的次数
    - ĝ(X)：不在其目标位置的非空白棋牌数目

例如：右图(a)中，不在目标位置的非空白棋牌有{7,11,12}，所以ĝ(X)=3

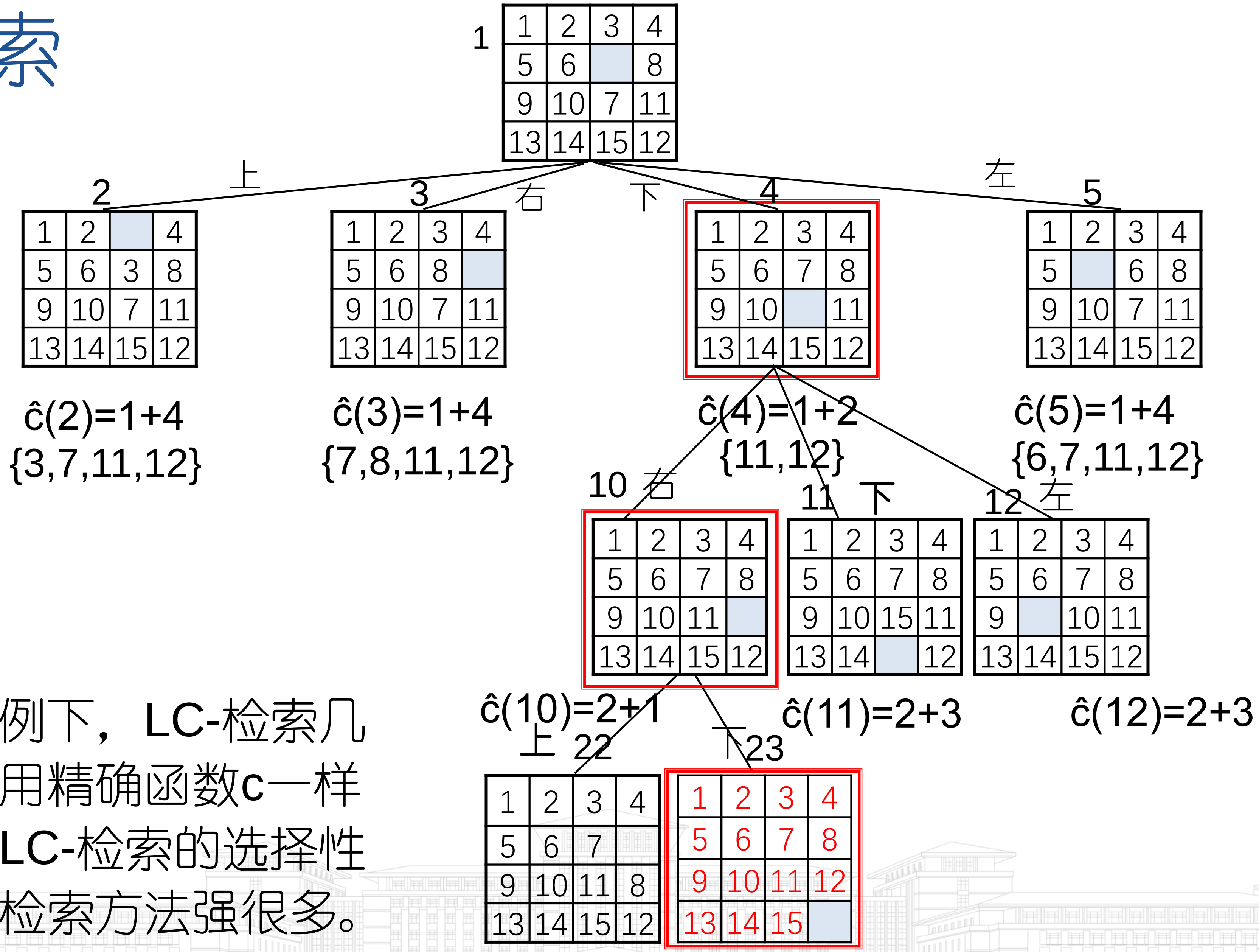
1	2	3	4
5	6		8
9	10	7	11
13	14	15	12

(a)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(b)

LC-检索



当前实例下，LC-检索几乎和使用精确函数c一样有效，LC-检索的选择性比很多检索方法强很多。

8.4 求最小成本的分支限界法

- 寻找最小成本
- 算法8.2 基于ĉ求最小成本的LC-检索算法
- 求最小成本的分支限界法基本思想
- 最小成本上界U
- 算法8.3 界函数UB
- 算法8.4 求最小成本的FIFO-分支限界算法
- 算法8.5 求最小成本的LC-分支限界算法
- 求极大化问题
- 总结求最优解问题的分支限界法

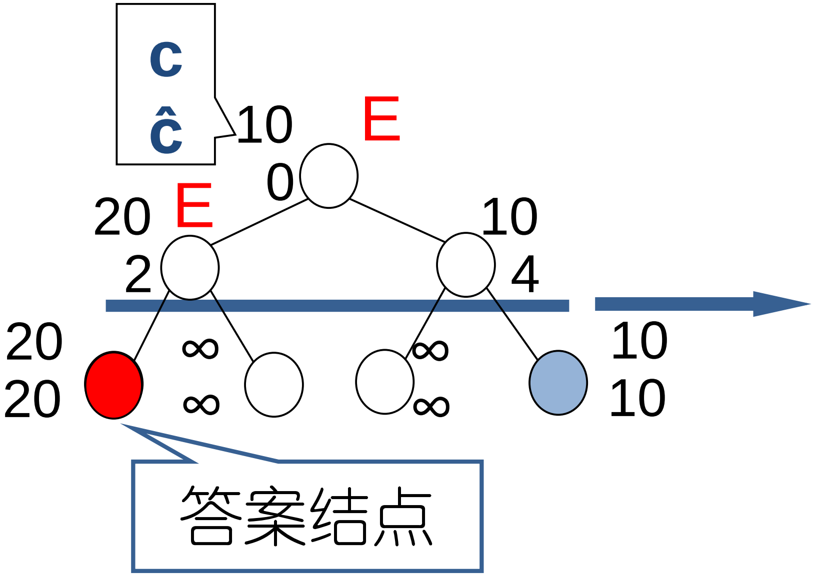
可行解：即答案结点

最优解：使目标函数取极值的答案结点

寻找最小成本

BB算法中：算法一旦判断出儿子结点X是答案结点，则打印路径，操作结束。

- 基于LC-检索的算法BB是否一定能找到具有最小成本的答案结点G呢？



$c(X)>c(Y)$
$\hat{c}(X)<\hat{c}(Y)$

难实现，改进算法

- 对ĉ追加要求：对于每一对结点X、Y，当$c(X)<c(Y)$时有$\hat{c}(X)<\hat{c}(Y)$

寻找最小成本

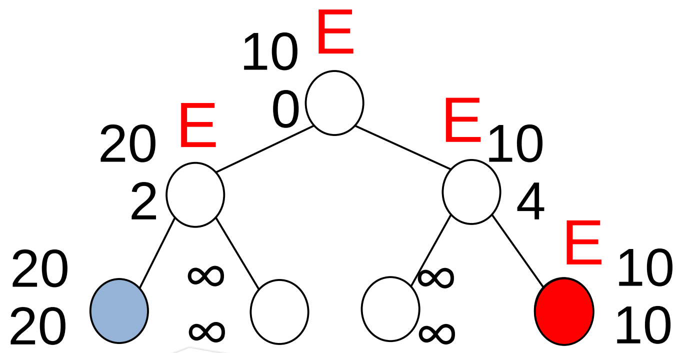
- 算法BB基于LC-检索寻找具有最小成本的答案结点，则ĉ要满足：
  - 易于计算
  - 对于每一个结点X，　ĉ(X) ≤ c(X)
  - 对于答案结点X，有ĉ(X) = c(X)
  - 追加：c(X)<c(Y)时，有ĉ(X)< ĉ(Y)

难实现

- 一般只能找到满足前三项要求的ĉ

改进算法：

算法从活结点表中选出E结点时，再判断E是否是答案结点，若是则打印路径，操作结束。



思考：如何证明改进后找到的答案结点一定是最小成本？

算法8.2 基于ĉ求最小成本的LC-检索算法

```
procedure LC(T, ĉ) //为找出最小成本答案结点
E←T, 将活结点表初始化为空
loop
    if E是答案结点 then 输出从E到T的那条路径; return; endif
    for E的每个儿子X do
        if B(X) then call ADD(X); PARENT(X)←E; endif
    repeat
    if 表中不再有活结点 then print(“no answer node”); return; endif
    call LEAST(E)
repeat
end LC
```

满足ĉ(E)=c(E)=cost(E)

选择ĉ最小的

- 对于活结点表中的每一个结点L，一定有ĉ(E)≤ĉ(L)。由ĉ定义知，E是答案结点时c(E)=ĉ(E)，则c(E)=ĉ(E)≤ĉ(L) ≤c(L)
- 因此，算法LC选中成本最小的答案结点

加速寻找最小成本

- 基本思想：
  - 满足约束条件的答案结点并不一定成本最小，因此在寻找最优解过程中，不止判断B(X)，若发现结点X不能导致最小成本答案结点，也不必再搜索子树X，子树X被剪枝。
- 设计方案：
  - 设置一个最小成本上界U，问题的最小成本不会大于这个上界U。如果$\hat{c}(X)>(或\geq)U$，则算法无需检索以X为根的子树。
  - 算法借助U和$\hat{c}(X)$，进一步提高剪枝能力。

两种剪枝：可行解剪枝；最优解剪枝

基于ĉ和U求最小成本的分支限界法基本思想

分支限界法求极小化问题时，问题转化为在解空间树中寻找最小成本答案节点。

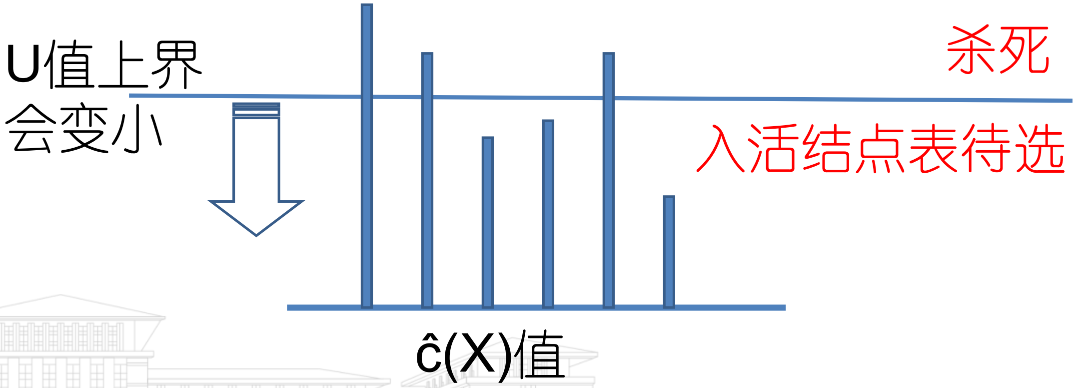
- 1) 对于极小化问题，把目标函数作为成本函数c
- 2) 约束条件作为约束函数B
- 3) 问题转化为寻找解空间树中最小成本答案结点
- 4) 设计成本估计函数ĉ(X)，ĉ(X)≤c(X)
- 5) 还可以设计最小成本的上界U，c(X)≤U
- 6) 基于ĉ(X)和U进行分支限界搜索

U是界

思考：极大化问题如何转化？

最小成本上界U

- U的含义
  - U是当前算法生成的所有状态结点对最小成本上界估计的最小值
- U的取值
  - 初值∞，或通过启发式方法得到；初值≥最小成本答案结点的成本
  - 随着结点的访问不断改小
- U的作用
  - 界定当前结点的死活
  - 通过剪枝加快找到最优解



成本上界函数u(X)

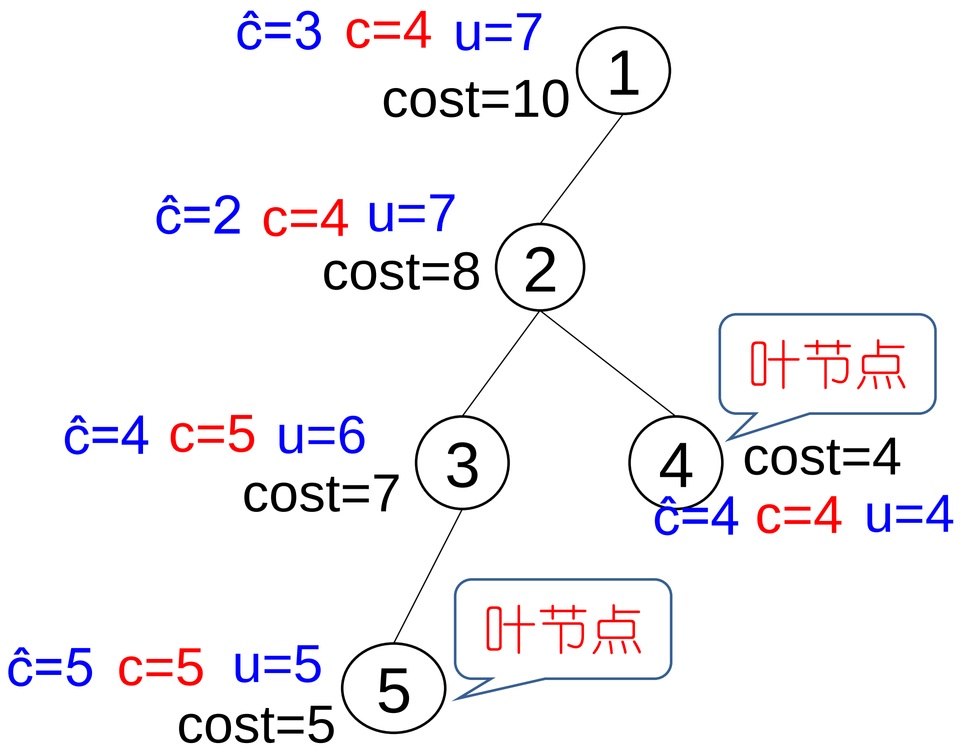
为提高U的剪枝能力，希望能不断减小U值。U可以通过u(X)加速减小。

- u(X)是对成本c(X)的上界估计，以下简称成本上界函数
- u(X)可仿照ĉ(X)定义，或其他方式：
  - 根到X的成本
  - 子树X中，X到最小成本答案节点的成本上界估计
- 对成本上界函数u的要求：
  - 易于计算
  - $c(X) \le u(X)$
  - 当叶结点X是答案结点时，$c(X)=u(X)$

对于结点X，有$\hat{c}(X) \le c(X) \le u(X)$

区分函数cost,ĉ,c和u

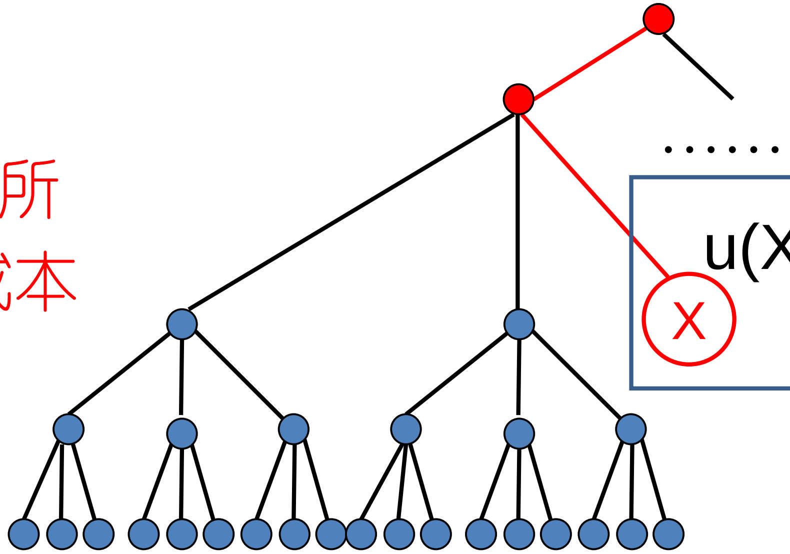
- cost(X)：根到答案结点X的真实成本
- c(X)：子树X中所有答案结点的最小成本值，即最小cost值
- ĉ(X)和u(X)：对子树X的c值估计，满足ĉ(X)≤c(X)≤u(X)
- 设图中，设当前每个结点都满足约束条件
- 当cost(X)>u(X)时，意味着以X为根的子树中一定有更优解。



区分上界值U和函数u(X)

算法当前生成的状态空间树如图所示，X是当前正在生成的结点

U是当前算法生成的所有状态结点对最小成本上界估计的最小值



u(X)是从根出发到X的这条指定路径下的最小成本上界估计值。

U会检查x的ĉ值，界定X结点的死活；也会利用u(X)值改小自己

细谈界U的改值和剪枝

- U改值
  - 发现一个答案结点X，$cost(X)<U$时，考虑$U \leftarrow cost(X)$
  - 发现一个状态结点X，$u(X)<U$时，考虑$U \leftarrow u(X)$
- U剪枝
  - 若U值来自一个真实成本，$\hat{c}(X) \geq U$的活结点X都可以被杀死
    - 此时U记录当前最小成本的答案结点，想找更小成本的答案结点
  - 若U值来自一个成本上界，$\hat{c}(X) > U$的所有活结点X都可以被杀死
    - 此时能确定存在，但还没有找到一个成本小于等于U的答案结点

界定当前结点的死活

思考：答案结点X是非叶节点时，如果cost(X)和u(X)都优于当前U值，该怎样选择?

选值小的

改进界U的剪枝

目前的问题：U值来源不同，剪枝策略不同。
为方便剪枝，当U将从一个u(X)获取数值时，U值调高一点。

- 一个改进技巧
  - 定义一个足够小的正常数ε，对任意结点X,Y，当u(X)<u(Y)时，有 u(X)<u(X)+ε<u(Y)
  - 当U从u(X)获得数值时，再追加一个ε。即U←u(X)+ε
  - 当U从cost(X)获得数值时，无需调整。即U←cost(X)
- 改进后的U剪枝：
  - 当前结点X若满足ĉ(X)≥U，X被杀死

算法8.3 界函数UB

- 一个足够小的正常数ε：对任意结点X,Y，如果$u(X)<u(Y)$，$u(X)<u(X)+\varepsilon<u(Y)$

```
procedure UB(X,ε,U,ans)
//X是当前活结点，U是当前上界估计值，ans是当前最小成本的答案结点。
//结点X满足ĉ(X)≤c(X)≤u(X)；当X是答案节点时，cost(X)表示根到X的真实成本。
   if ĉ(X)≥U then return false
   if X是解结点 and cost(X)<U
        then U←min(cost(X), u(X)+ε); ans←X
        else  if u(X)+ε<U then U←u(X)+ε endif
   endif
   return true
end UB
```

界定杀死，否则探查是否能够更改U值

U←min(cost(X), u(X)+ε) 较 U←cost(X) 更能提高剪枝

算法8.4 求最小成本的FIFO-分支限界算法

假定状态空间树T至少包含一个解结点，不可行结点的估计值ĉ(X)=∞

```
procedure FIFOBB(T,ĉ,u,ε,cost)
  E←T; PARENT(E)←0; U←∞， ans←0
   UB(E,ε,U,ans)
  将队列初始化为空
  loop
    for E的每个儿子X do
        if B(X) and UB(X,ε,U,ans) then call ADDQ(X); PARENT(X)←E endif
    repeat
    loop
        if 队列为空 then print ('least cost = ', U); 输出从ans到T的路径; return endif
        call DELETEQ(E); if ĉ(E)<U then exit endif
   repeat
  repeat
end FIFOBB
```

思考：请验证算法的正确性

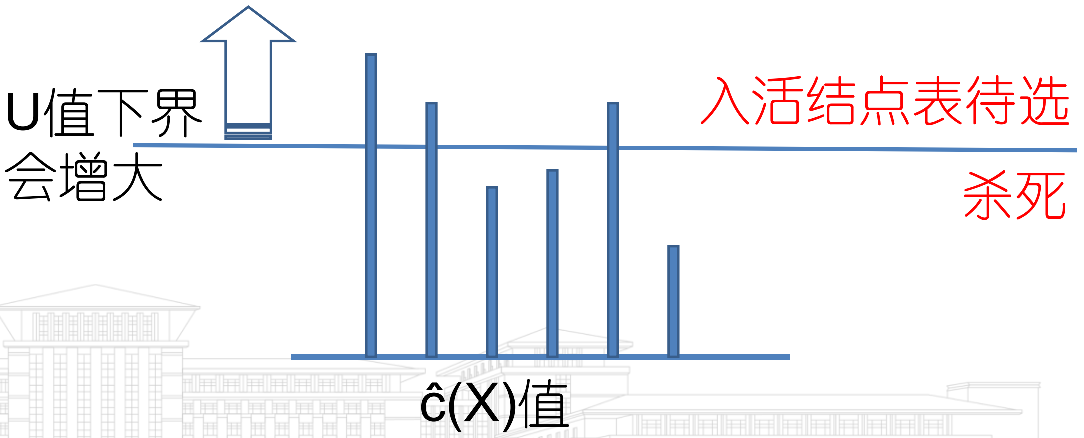
算法8.5 求最小成本的LC-分支限界法

假定状态空间树T至少包含一个解结点，不可行结点的估计值ĉ(X)=∞

```
procedure LCBB(T,ĉ,u,ε,cost)
//函数ADD：加一个结点到min-堆中；函数LEAST：从min-堆中删去堆顶结点
  E←T; PARENT(E)←0; U←∞， ans←0
  UB(E,ε,U,ans)
  将活结点表初始化为空
  loop
      for E的每个儿子X do
         if B(X) and UB(X,ε,U,ans) then call ADD(X); PARENT(X)←E endif
      repeat
      if 表中不再有活结点 or 堆顶结点ĉ值≥U
         then print (‘least cost = ‘ ,U); 输出从ans到T的那条路径; return endif
      call LEAST(E)
  repeat
end LCBB
```

求极大化问题

- 很多极值问题求目标函数极大值，可以采用以下两种方法解决：
- 方法1：修改目标函数，将问题转化为极小化问题
  - 取目标函数的相反数作为成本函数c
- 方法2：对照极小化问题做镜像修改
  - 把目标函数作为成本函数c，问题转化为寻找解空间树中最大成本答案结点，此时$u(X) \le c(X) \le \hat{c}(X)$。



总结求最优解问题的分支限界法

- 算法剪枝的依据
  - 约束函数B：限定是否存在可行解
  - 成本估计函数ĉ(X)和界U：界定是否存在最优解
- 算法剪枝的发生点
  - X入活结点表时，接受B检验和U检验
  - X出活结点表时，接受U检验
- 算法终止的条件
  - 活结点表为空
  - 活结点表中再没有通过U检验的活结点

能

思考：用回溯法解决最优解问题时，ĉ(X)和界U能否用于提高算法效率？

8.5 带有期限的作业调度问题

- 问题描述
- 一个问题实例
- 约束函数B
- 成本估计函数ĉ
- 成本上界U
- FIFO-分支限界法实例运行

问题描述

- 假设有n个作业和一台处理机，每个作业i由一个三元组(p_i,d_i,t_i)表示，表示作业需要t_i个时间处理完毕，如果在期限d_i之前没有完成则要交付p_i的罚款。
- 问题目标：从这n个作业中选取一个子集合J，使J中作业都能在相应的期限内完成，而不在J中的作业罚款总数最小。

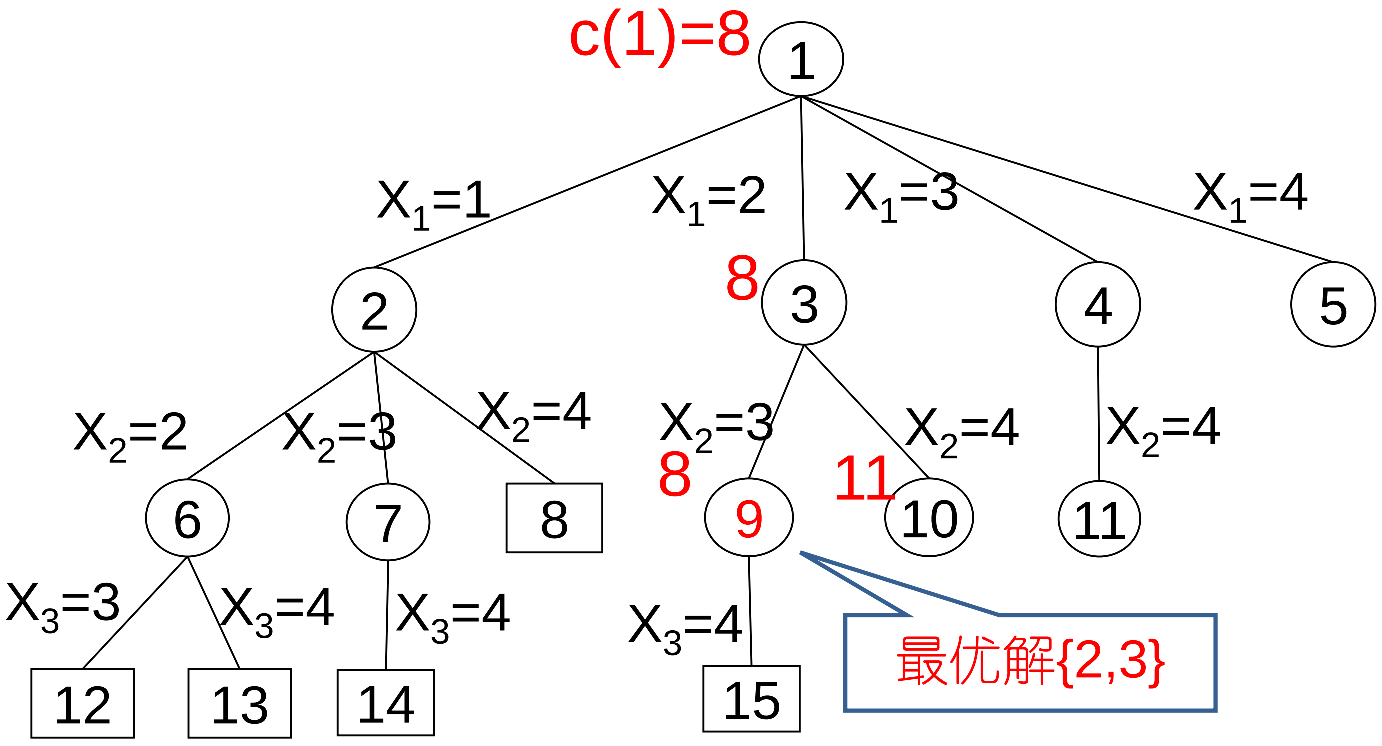
极小化问题

一个问题实例

- $n=4$, $(p_1,d_1,t_1)=(5,1,1)$; $(p_2,d_2,t_2)=(10,3,2)$; $(p_3,d_3,t_3)=(6,2,1)$;$(p_4,d_4,t_4)=(3,1,1)$;
- 解空间的表示方法
  - 不定长的k-元组$(X_1,..X_k)$, $k\le n$
  - 显式约束条件：X_j表示选中的作业下标, $x_i<x_{i+1}$, $1\le i<k$
  - 隐式约束条件：作业能在期限前完成
  - 目标函数：未选中的作业罚款总数最小
- 状态空间树
  - 共计$2^n=16$个结点
  - 圆形结点表示满足约束条件的结点
  - 方形结点表示不可行结点

约束函数B

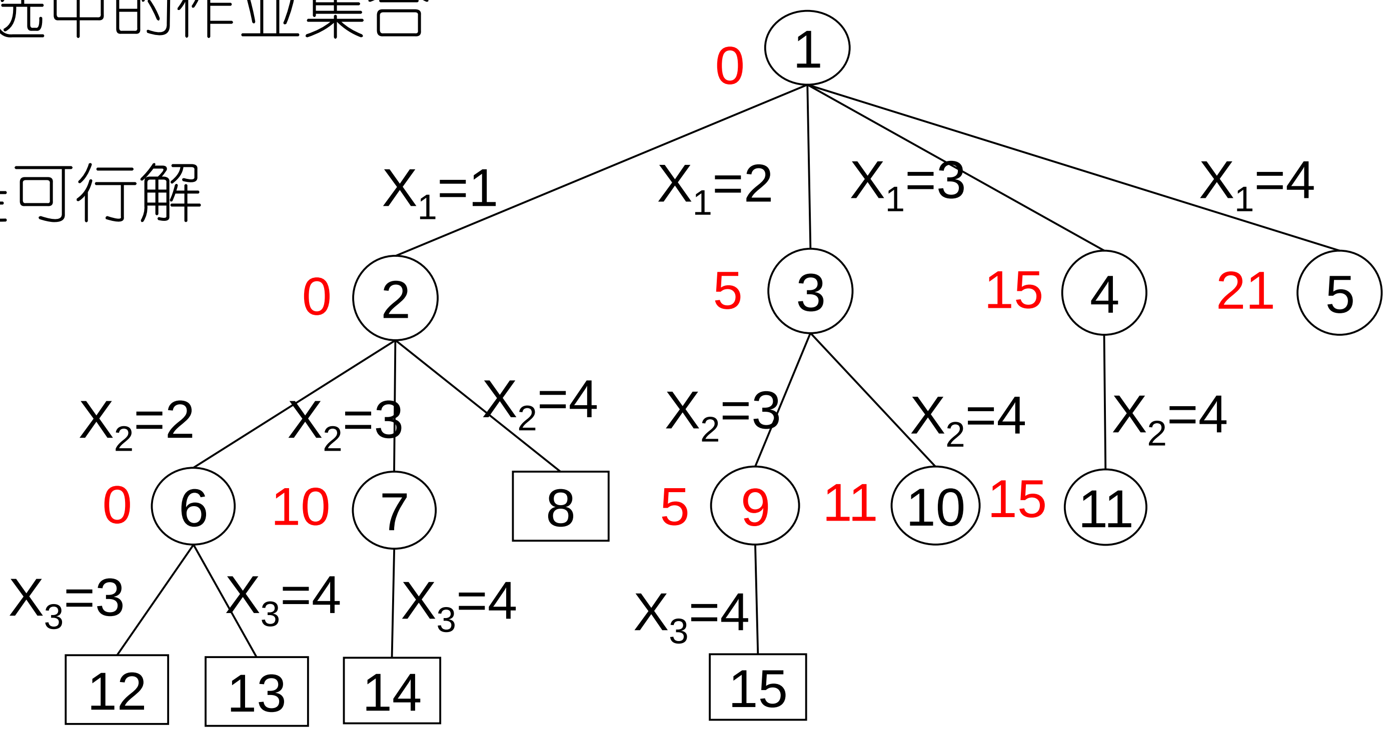
- n=4
- k-元组表示状态空间树
- 按层次遍历为结点编号
- 作业实例
  - $(p_1,d_1,t_1)=(5,1,1)$
  - $(p_2,d_2,t_2)=(10,3,2)$
  - $(p_3,d_3,t_3)=(6,2,1)$
  - $(p_4,d_4,t_4)=(3,1,1)$



10个活结点通过B检验

成本估计函数ĉ

- 定义成本估计函数ĉ(X)，使得ĉ(X)≤c(X)：
  - 设Sx是根结点到达结点X时选中的作业集合
  - 令$m=\max\{i|i\in Sx\}$
  - $\hat{c}(X)=\sum p_i$, $i<m$, $i\notin Sx$，若X是可行解
  - $\hat{c}(X)=\infty$，若X是不可行解
- 作业实例
  - $(p_1,d_1,t_1)=(5,1,1)$
  - $(p_2,d_2,t_2)=(10,3,2)$
  - $(p_3,d_3,t_3)=(6,2,1)$
  - $(p_4,d_4,t_4)=(3,1,1)$



成本上界U

- 上界估计函数$u(X)=\sum p_i,\ i \notin S_X$
- 作业实例
  - $(p_1,d_1,t_1)=(5,1,1)$
  - $(p_2,d_2,t_2)=(10,3,2)$
  - $(p_3,d_3,t_3)=(6,2,1)$
  - $(p_4,d_4,t_4)=(3,1,1)$

成本上界$U=\min\{u(X)\}$

$U \leftarrow 24$

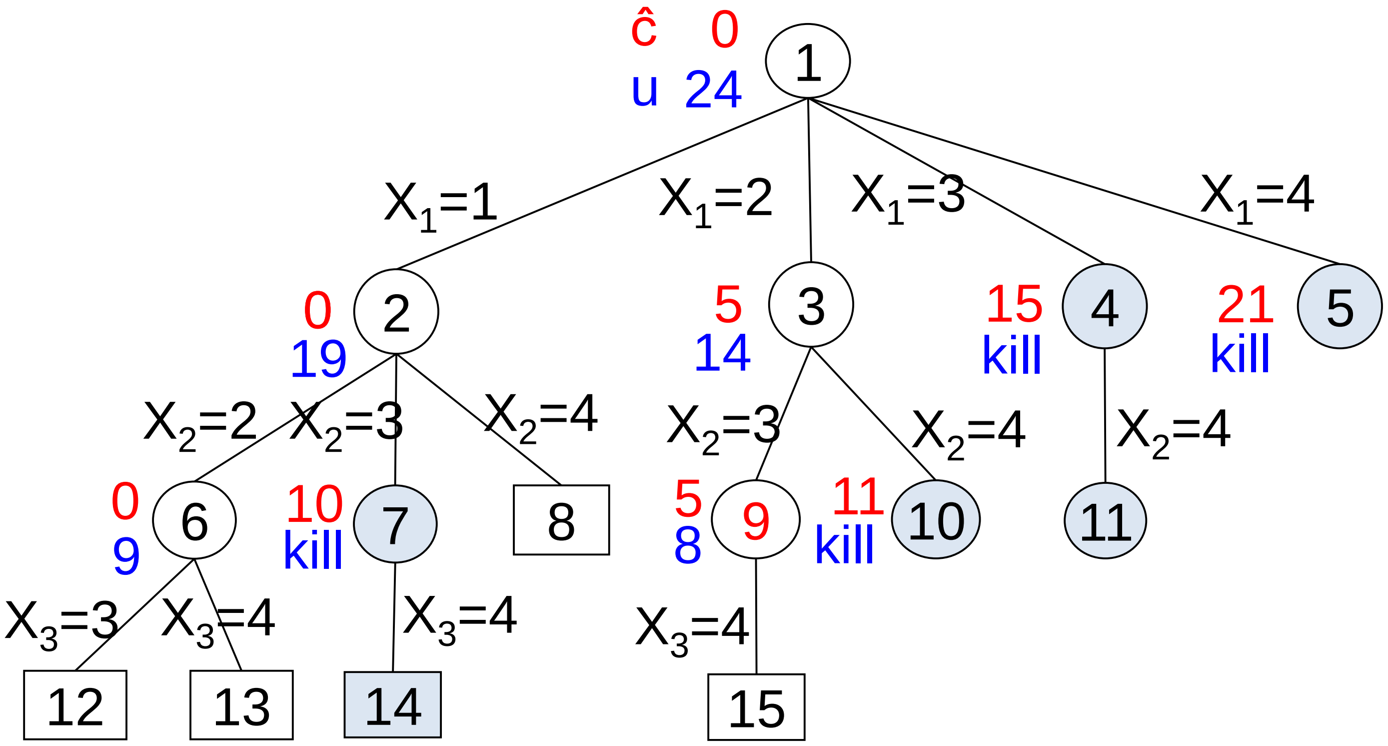
$U \leftarrow 19$

$U \leftarrow 14$

$U \leftarrow 9$

$U \leftarrow 8$

FIFO-检索下考虑U的剪枝作用



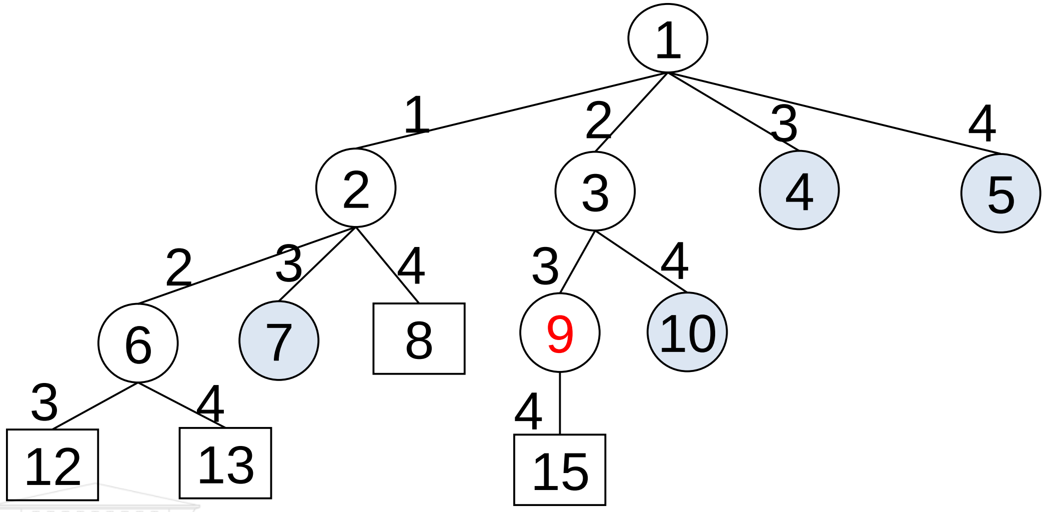
5个活结点通过U检验

FIFO-分支限界法实例运行

E	儿子X	U	ans	队列
1		24	1	
	2	19	2	
	3	14	3	2 3
	4弃			
	5弃			
2		14	3	
	6	9	6	
	7弃			
	8弃			3 6
3		9	6	
	9	8	9	
	10弃			6 9
6		8	9	
	12弃			
	13弃			9
9		8	9	
	15弃			

ε=0.1，U不断减小；ĉ(X)≥U时，X被杀死

编号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ĉ	0	0	5	15	21	0	10	∞	5	11	∞	∞	∞	∞	∞
cost	24	19	14	18	21	9	13	∞	8	11	∞	∞	∞	∞	∞
u	24	19	14	18	21	9	13	∞	8	11	∞	∞	∞	∞	∞



队列为空，算法结束。此时U等于8，ans等于9。

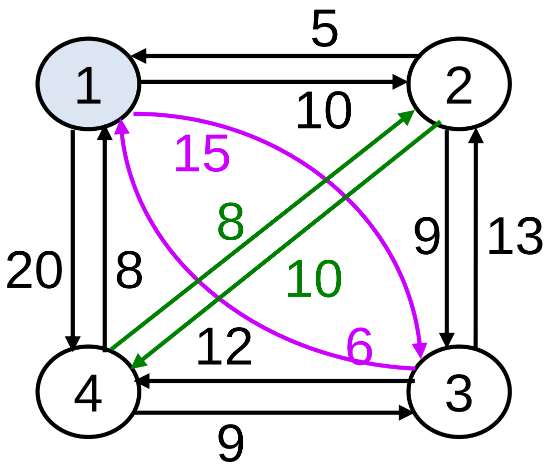
8.6 货郎担问题

- 问题描述
- 一个问题实例
- 成本函数c
- 成本下界函数ĉ
- 成本上界U
- LC-分支限界法实例运行

问题描述

- TSP问题：某售货员要到若干个城市销售货物，已知各城市之间的距离，要求售货员从某一城市出发并选择旅行路线，使每个城市经过一次，最后回到原出发城市，而总路程最短。
- 问题形式化描述:设$G(V,E)$是一个有向图，$|V|=n$，$c_{ij}>0$表示边$<i, j>\in E$的成本。寻找一条最小成本的周游路线，不失一般性，考虑从结点1开始，在结点1结束。
- 动态规划方法求解的时间复杂度是$\Theta(n^2 2^n)$
- 分支限界法最坏情况也是$O(n^2 2^n)$，但对许多具体实例，能花费较少的时间

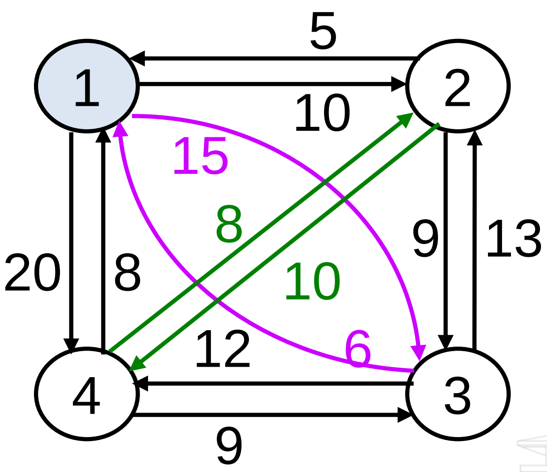
一个问题实例



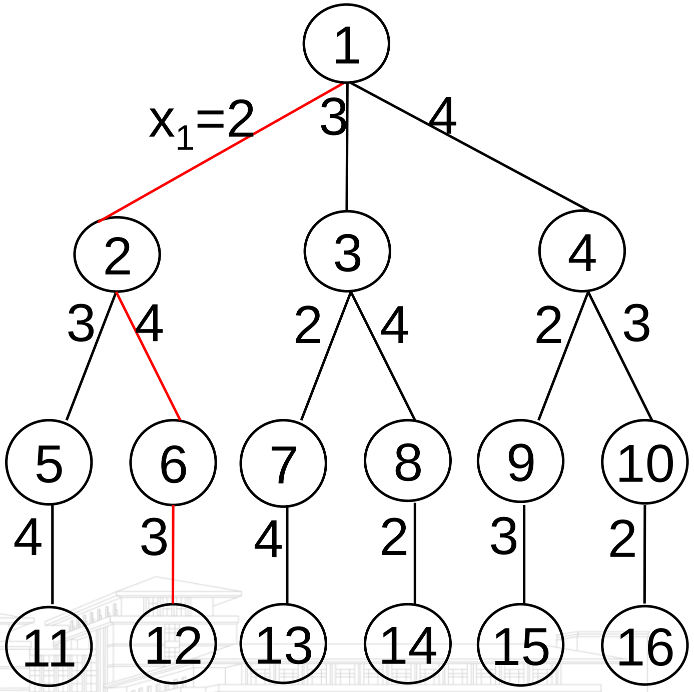
- n=4, 从1经过V-{1}={2,3,4}, 再到达1。
- 问题的解是S中结点的一种排列(X_1,X_2,X_3)，使$1,X_1,X_2,X_3,1$的成本最小
- 解空间的表示方法
  - 设计为固定长3-元组$(X_1,..X_{n-1})$，n=4
  - 显示约束条件：$X_i \in \{2,3,4\}$, 且不相同, $1 \le i \le n-1$
  - 隐式约束条件：存在周游路线$1,X_1,X_2,X_3,1$，即相邻两点有边存在
  - 目标函数：周游路线成本最小
- 状态空间树
  - 形如n-皇后问题，共计3!个叶结点，根结点到叶结点的一条路线是{2,3,4}的一个排列形式

成本函数c

- 成本函数c(X)：目标函数作为成本函数
  - c(X)=根到X的的路径的周游路线成本，X是叶结点
  - c(X)=子树X中最小成本叶结点的成本，X非叶结点
- 例如c(12)=$c_{12}+c_{24}+c_{43}+c_{31}$=10+10+9+6=35



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0



成本下界函数ĉ

- 成本下界函数$\hat{c}(X) \le c(X)$，X表示$(X_1,..X_k)$到达的结点， $\hat{c}(X)= f(X)+ \hat{g}(X)$
- $f(X)$表示已选定的路线$X_0=1,X_1,\ldots X_k$的成本和
  - 设a_i表示边$<X_i,X_{i+1}>$的成本, $0\le i<k$
  - $f(X)= \sum a_i$
- $\hat{g}(X)$表示经过剩余结点回到1的最短距离估计
  - 设$S=V-\{1,X_1,\ldots X_k\}$ 是剩余结点
  - 设l_d表示结点d出发的最小边成本，$d\in S$
  - $\hat{g}(X)=l_{Xk}+ \sum_{d\in S} l_d$

不关心到达的点是什么

$$\hat{c}(X)=\begin{cases} \sum_{0\leqslant i<k} a_i + l_{Xk}+ \sum_{d\in S} l_d & k<n-1\text{时} \\ \sum_{0\leqslant i<k} a_i + \text{边}<X_{n-1},1>\text{的成本} & k=n-1\text{时} \end{cases}$$

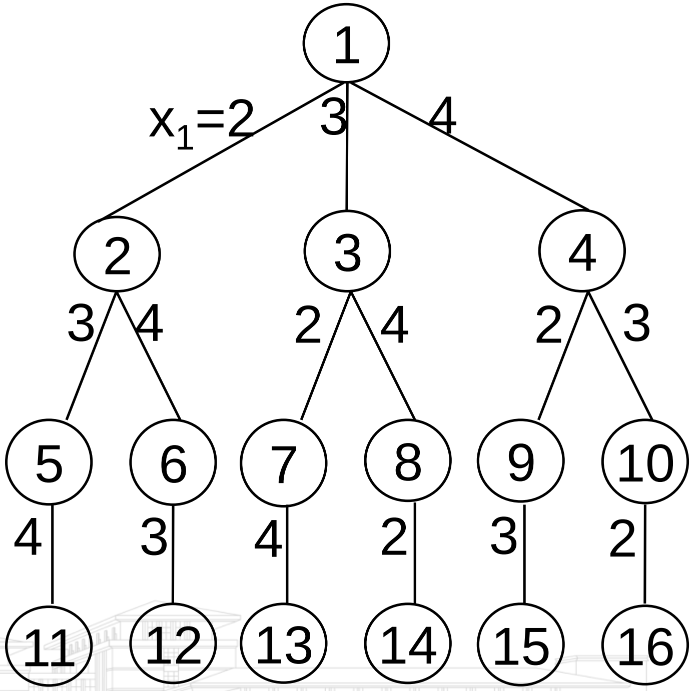
$$\underbrace{1,X_1,\ldots X_k}_{f(X)}\underbrace{X_k}_{l_{Xk}}\underbrace{X_{k+1},\ldots,X_{n-1},1}_{\sum_{d\in S} l_d}$$

实例ĉ值

部分结点编号	路径	f(X)	S	l_{Xk}	$\sum l_d$	ĉ(X)
1	空	0	{2,3,4}	10	19	29
2	1-2	10	{3,4}	5	14	29
3	1-3	15	{2,4}	6	13	34
4	1-4	20	{2,3}	8	11	39
5	1-2-3	19	{4}	6	8	33
6	1-2-4	20	{3}	8	6	34
11	1-2-3-4		空			39
12	1-2-4-3		空			35

=C

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0



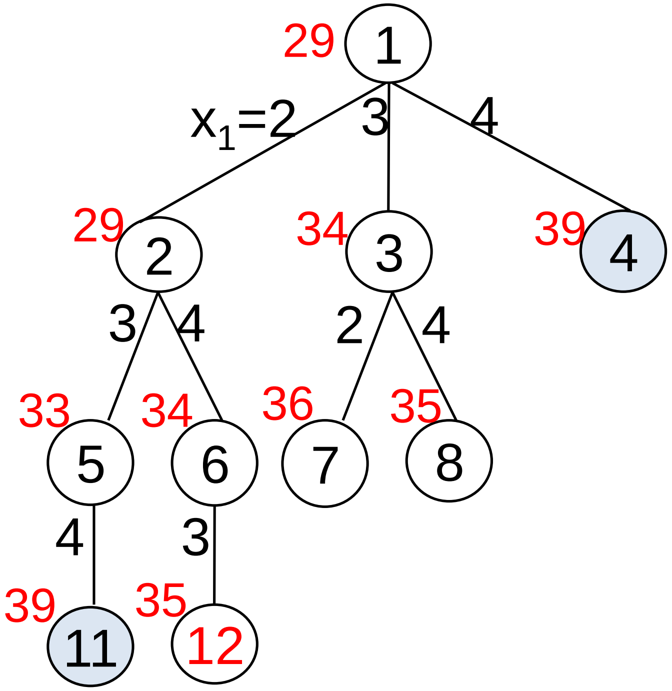
成本上界U

- 成本上界U≥c(X)
  - 当前得到的最短周游路线长度
  - 初始值可以通过贪心法获得
- 问题实例从1出发，贪心法求初始U=39
- 当前没有设计u(X)

	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

LC-分支限界法实例运行

E	儿子X	U	ans	活结点表
1	2 3 4弃	39		2,3
2	5 6	39		3,5,6
5	11弃	39		3,6
3	7 8	39		6,7,8
6	12	35	12	7,8,12
8		35	12	



堆顶结点ĉ(8) ≥ U，算法结束。此时U等于35， ans等于12

8.7 小结

- 分支限界法与回溯法的相同点：
  - 同样适用于求解组合数较大的问题(/多阶段决策问题)
  - 都是在解空间树上搜索答案结点
  - 都会借助约束函数B进行剪枝
- 分支限界法与回溯法的不同点：
  - 最本质的区别在于E-结点(即扩展结点)处理方式不同，见第七章；分支限界法还可以基于ĉ选择，因此求最优解问题时效率更高
  - 存储空间上，分支限界法需要额外维护活结点表，回溯法不需要

- 分支限界法求极小化问题，即寻找状态空间树中最小成本的答案结点
  - 目标函数作为成本函数c
  - 约束条件作为约束函数B
  - 设计成本估计函数ĉ(X)，ĉ(X)≤c(X)
  - 设计最小成本的上界U，c(X)≤U
  - 基于ĉ(X)和U进行分支限界搜索
- 分支限界法求极大化问题
  - 目标函数取相反数，转化为极小化问题
  - 或对照极小化问题做镜像修改
- 分支限界法中的剪枝
  - 约束函数B：限定是否存在可行解
  - 成本估计函数ĉ(X)和界U：界定是否存在最优解

8.7 小结

- 8.1 一般方法
  - 掌握分支限界法适用的问题特点，掌握分支限界法求解问题的设计思想和一般方法，掌握分支限界法的不同检索方式的差异性
- 8.2 LC-检索
- 8.3 15-谜问题
  - 掌握结点成本函数c定义、成本估计函数ĉ定义和LC-检索定义
  - 以15-谜为例理解LC-检索的优势，掌握15-谜问题判定定理和ĉ设计
- 8.4 求最小成本的分支限界法
  - 掌握基于ĉ(X)和U优化算法的一般思想，掌握分支限界法求最优解问题的一般算法FIFOBB和LCBB

8.7 小结

- 8.5 带有期限的作业调度问题
- 8.6 货郎担问题
  - 掌握经典问题的解空间构造方法、约束函数B的设计思想，掌握$\hat{c}(X)$和U的设计方法，掌握求最优解问题的分支限界算法

能够识别出适合分支限界法的可计算性问题、独立设计算法和分析算法复杂度。

本章结束