



中国科学技术大学 计算机科学与技术系
University of Science and Technology of China
DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

算法设计与分析

Design and Analysis of Algorithms

主讲人 徐云

Fall 2018, USTC

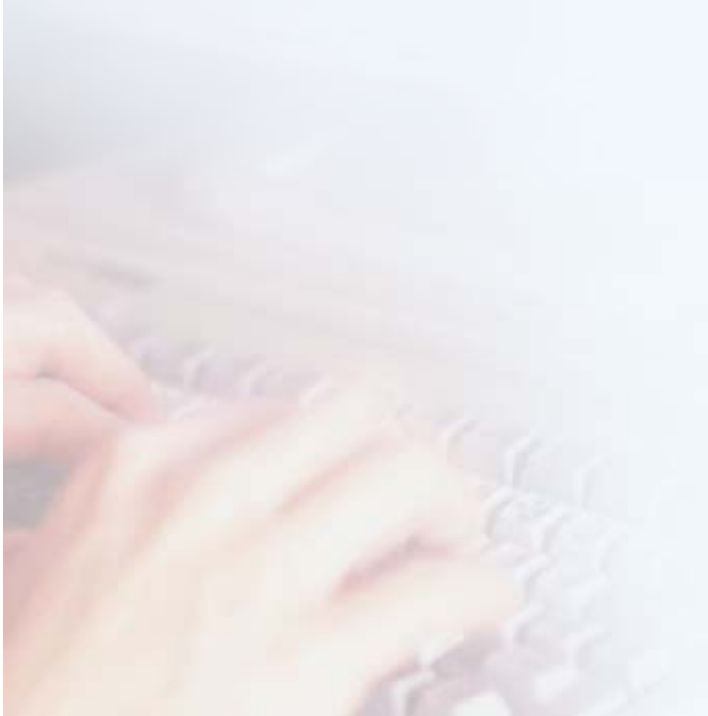


第3章(补充) 分支限界法

3.1 方法概述

3.2 8谜问题

3.3 装载问题



3.1 方法概述

- 基本思想
- 与回溯法的区别
- 求解步骤
- 两种常见的活结点扩充方式
- 示例1和示例2

基本思想

- 在解空间树中,以广度优先BFS或最佳优先方式搜索最优解,利用部分解的最优信息,裁剪那些不能得到最优解的子树以提高搜索效率。
- 搜索策略是:在扩展结点处,先生成其所有的儿子结点(分支),然后再从当前的活结点表中选择下一个扩展结点。为了有效地选择下一扩展结点,以加速搜索的进程,在每一活结点处,计算一个函数值(优先值),并根据这些已计算出的函数值,从当前活结点表中选择一个最有利的结点作为扩展结点,使搜索朝着解空间树上有最优解的分支推进,以便尽快地找出一个最优解。

与回溯法的区别

- 求解目标不同：

一般而言，回溯法的求解目标是找出解空间树中满足约束条件的所有解，而分支限界法的求解目标则是尽快地找出满足约束条件的一个解；

- 搜索方法不同：

回溯算法使用深度优先方法搜索，而分支限界一般用宽度优先或最佳优先方法来搜索；

- 对扩展结点的扩展方式不同：

分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点；

- 存储空间的要求不同：

分支限界法的存储空间比回溯法大得多，因此当内存容量有限时，回溯法成功的可能性更大；

求解步骤

- ① 定义解空间(对解编码);
- ② 确定解空间的树结构;
- ③ 按BFS等方式搜索:
 - a. 每个活结点仅有一次机会变成扩展结点;
 - b. 由扩展结点生成一步可达的新结点;
 - c. 在新结点中, 删除不可能导出最优解的结点; // 限界策略
 - d. 将余下的新结点加入活动表(队列)中;
 - e. 从活动表中选择结点再扩展; // 分支策略
 - f. 直至活动表为空;

两种常见的活结点扩充方式

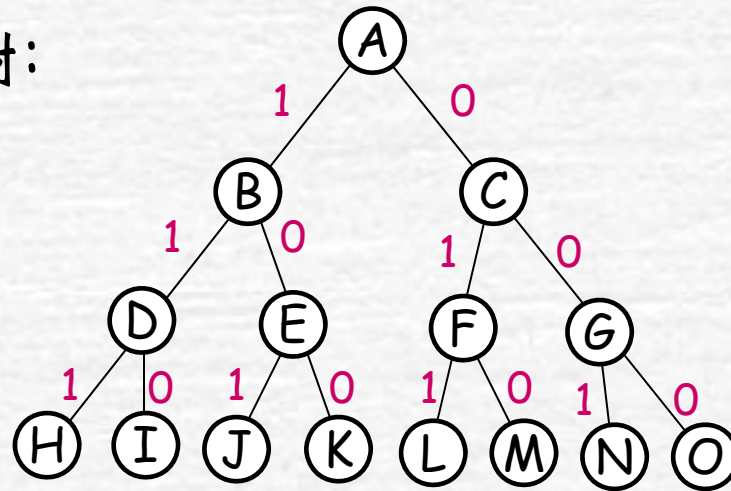
- 先进先出队列 (F I F O) : 从活结点表中取出结点的顺序与加入结点的顺序相同, 因此活结点表的性质与队列相同;
- 优先队列 (耗费用小根堆, 受益用大根堆): 每个结点都有一个对应的耗费或收益。
 - 如果查找一个具有最小耗费的解, 则活结点表可用小根堆来建立, 下一个扩展结点就是具有最小耗费的活结点;
 - 如果希望搜索一个具有最大收益的解, 则可用大根堆来构造活结点表, 下一个扩展结点是具有最大收益的活结点。

示例1 (1)

- 示例1 (FIFO队列分支限界法)
- 问题： 0-1背包问题:物品数 $n=3$, 重量 $w=(20,15,15)$, 价值 $v=(40,25,25)$, 背包容量 $c=30$, 试装入价值和最大的物品?
- 求解:

①解空间: $\{(0,0,0), (0,0,1), \dots, (1,1,1)\}$

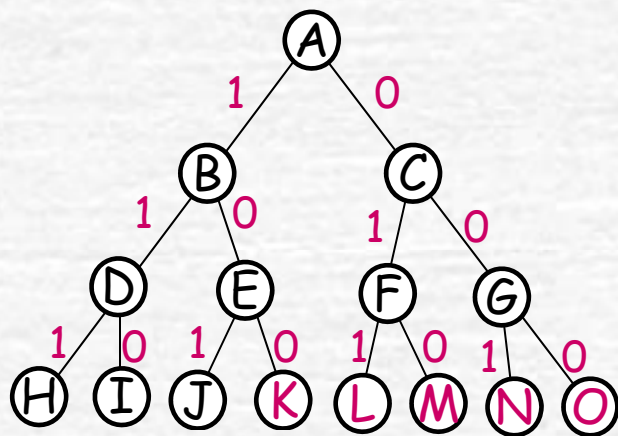
②解空间树:



示例1 (2)

③BFS搜索 (FIFO队列)

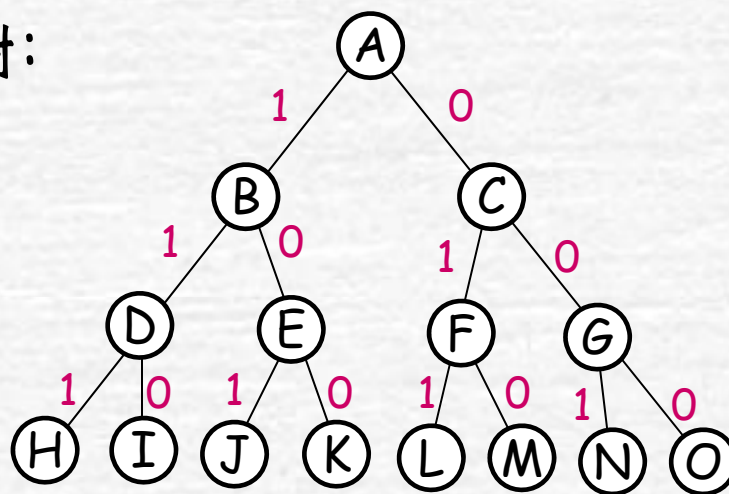
扩展结点	活结点	队列(可行结点)	可行解(叶结点)	解值
A	B,C	BC		
B	D,E(D死结点)	CE		
C	F,G	EFG		
E	J,K(J死结点)	FG	K	40
F	L,M	G	L,M	50,25
G	N,O	\varnothing	N,O	25,0



\therefore 最优解为L, 即(0,1,1); 解值为50

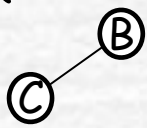
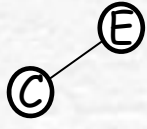

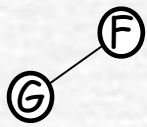

示例2 (1)

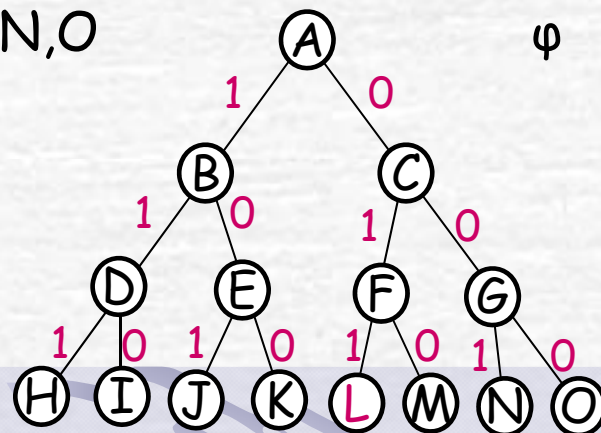
- 示例2 (优先队列分支限界法)
- 问题: 0-1背包问题: 物品数 $n=3$, 重量 $w=(20,15,15)$, 价值 $v=(40,25,25)$, 背包容量 $c=30$, 试装入价值和最大的物品?
- 求解:
 - ① 解空间: $\{(0,0,0), (0,0,1), \dots, (1,1,1)\}$
 - ② 解空间树:



示例2 (2)

- **BFS搜索** (优先队列: 按价值率优先)

扩展结点	活结点	队列(可行结点)	可行解(叶结点)	解值
A	B,C			
B	D,E(D死结点)			
E	J,K(J死结点)		K	40
C	F,G			
F	L,M		L	50(最优)
G	N,O	\varnothing		





第3章(补充) 分枝限界法

3.1 方法概述

3.2 8谜问题

3.3 装载问题

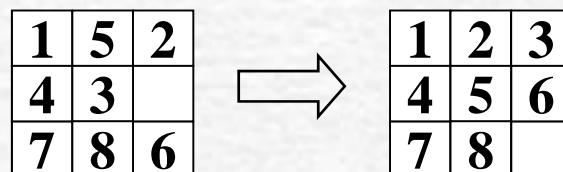


3.2 8谜问题

- 问题定义
- 代价函数
- 代价最小优先搜索

问题定义和代价函数

1. 问题定义



2. 代价函数 $f(x)$

设 $f(x)$ 为经 x 到解的代价， x 是当前结点：

(1) 如果 x 是死结点或 x 的子树不含解，使 $f(x) = \infty$

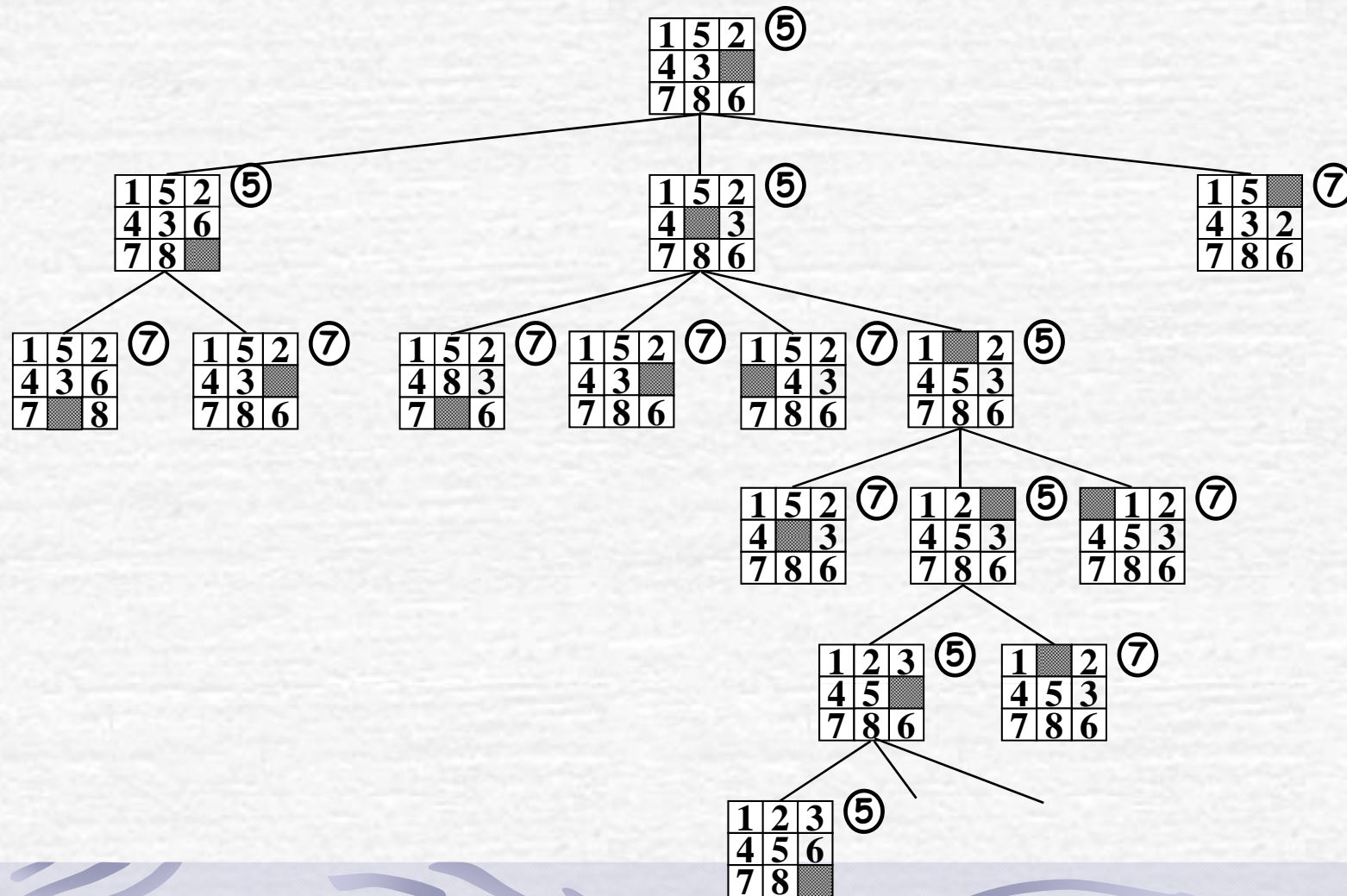
(2) 如果 x 是活结点： $f(x) = g(x) + h(x)$

$g(x)$: 从根到 x 的代价, $h(x)$: 由 x 产生解的代价估计

这里我们取: $h(x) = \sum_{i=1 \sim 8} (|x_1^i - x_0^i| + |y_1^i - y_0^i|)$

代价最小优先搜索

3. 代价最小优先搜索：结点旁的圆圈内的数字为 $f(x)$





第3章(补充) 分枝限界法

3.1 方法概述

3.2 8谜问题

3.3 装载问题



3.3 装载问题

- 问题定义
- 解空间树
- FIFO队列分支限界

问题定义 (1)

- 有一批共 n 个集装箱要装上2艘载重量分别为 c_1 和 c_2 的轮船，其中集装箱 i 的重量为 w_i ，且

$$\sum_{i=1}^n w_i \leq c_1 + c_2$$

- 装载问题要求确定是否有一个合理的装载方案可将这 n 个集装箱装上这2艘轮船。
- 可以证明：如果一个给定的装载问题有解，则采用下面的策略可得到一个最优装载方案：
 - ① 首先，将第1艘轮船尽可能装满；
 - ② 将剩余的集装箱装上第2艘轮船。

问题定义 (1)

因此,第1艘轮船的装载问题可定义为:

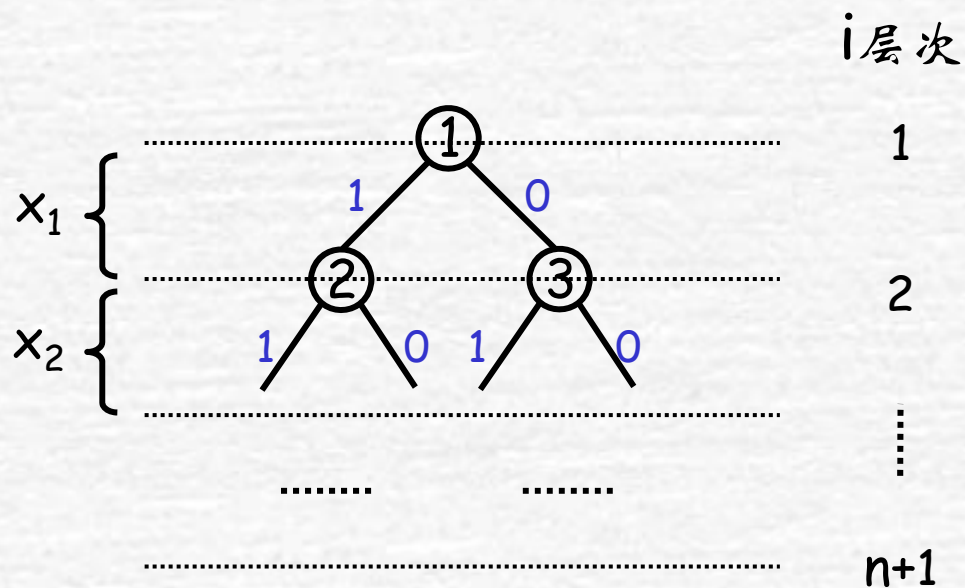
轮船载重为 c ,集装箱重量为 w_i ($i=1,2,\dots,n$),在装载体积不受限制的情况下,尽可能重的集装箱装上轮船。

问题的形式化定义:

$$\begin{aligned} & \max \sum_{i=1}^n w_i x_i \\ & s.t. \begin{cases} \sum_{i=1}^n w_i x_i \leq c & w_i > 0 \\ x_i \in \{0,1\} & i = 1,2,\dots,n \end{cases} \end{aligned}$$

解空间树

- 解表示和解空间: $\{(x_1, x_2, \dots, x_n) \mid x_i \in \{0, 1\}, i=1 \sim n\}$
- 解空间:



FIFO分支限界 (1)

- 全局变量的设计
 - $x[1..n]$ 保存搜索到的解;
 - bestw保存当前最优解值;

- 结点变量的设计

```
Typedef struct Treenode{  
    float wt;    //搜索到该结点时的载重量  
    int level;   //结点所处的层次  
    struct Treenode *parent; //指向父结点的指针  
}*qnode;
```

FIFO分支限界 (2)

- 算法

MaxLoading(w[], c, n)

{//返回最优值bestw和解向量x[1..n]}

inqueue(Q); //建立空队列Q, Q中元素为qnode类型

p=new qnode; //生成一个qnode结点, 装入①结点

p->wt=0; p->level=1; p->parent=NIL;

enqueue(Q, p); //入队

float bestw=0; qnode bestp=p;

FIFO分支限界 (3)

```
while( !empty(Q) ) do { //BFS遍历
    p=dequeue(Q); //出队, p结点成为扩展结点
    //扩展左孩子结点
    if p->wt+w[p->level]<=c then //p的左孩子是可行结点
    { q=new qnode;
      q->wt=p->wt+w[p->level]; q->level=p->level+1; q->parent=p;
      enqueue(Q, q);
      if( bestw<q->wt )
      { bestw=q->wt; bestp=q;
      }
    }
    //扩展右孩子结点
    { q=new qnode;
      q->wt=p->wt; q->level=p->level+1; q->parent=p;
      enqueue(Q, q);
    }
}
```


FIFO分支限界 (4)

//构造解

```
for i=1 to n do x[i]=0;
p=bestp;
while( p!=NIL ) do
{ tempw=p->wt;
  p=p->parent;
  if p->wt != tempw then x[p->level]=1;
}
} //end
```



End of SChap3

