第一章 基础知识

1. 通过初等操作进行 T(n)的估算

例：

```
void  childen_question(int n,int &k,int g[],int m[],
      int s[])
{                                          赋值等运算（初等操作）   算法1.1
    int  a,b,c;
    k=0;                                          //1
    for(a=0; a<=n; a++)                           //1+2(n+1)
       for(b=0; b<=n; b++)                        // n+1+2 (n+1)²
          for(c=0; c<=n; c++)                     // (n+1)² +2(n+1)³
            if(!(c%3)&&a+b+c==n &&
 //14(n+1)³                    5*a+3*b+c/3==n)
            {
                g[k]=a;  m[k]=b;  s[k]=c;
                k++;                              //4(n+1)³
            }
}
```

T(n) ≤1+ 1+2(n+1) + n+1+2 (n+1)2+(n+1)2 +2(n+1)3+14(n+1)3+4(n+1)3

= 20n3 +64n2 +72n +31

2. 上界和下界

设 f(n)是某算法的运行时间函数，g(n)是某一简单函数，当且仅当存在正的常数 c 和非负整数 n0，当 n≥n0 时：

如果有 f(n)≤cg(n)，则称 f(n)当 n 充分大时上有界；且 g(n)是它的一个上界，记作 f(n) = O(g(n))。或称 f(n)的阶不高于 g(n)的阶。

如果有 f(n)≥cg(n)，则称 f(n)当 n 充分大时下有界；且 g(n)是它的一个下界，记作 f(n) = Ω(g(n))。或称 f(n)的阶不低于 g(n)的阶。

3. 时间复杂度例题

求下列函数的渐近表达式：

$3n^2+10n$；$n^2/10+2^n$；$21+1/n$；$\log n^3$；$10\log 3^n$。

分析与解答：

$3n^2+10n=O(n^2)$；　　$\log n^3=O(\log n)$；

$n^2/10+2^n=O(2^n)$；　　$10\log 3^n=O(n)$。

$21+1/n=O(1)$；

第二章 递归算法（考试比例稍小）

1. **会写递归的形式、边界（参看 ppt）**

2. **有些递归问题会写过程**

第三章 分治策略

1. **分治法的适用条件（选择题）**

   1）该问题的规模缩小到一定的程度就可以容易地解决；
   2）该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质
   3）利用该问题分解出的子问题的解可以合并为该问题的解；
   4）该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

2. **合并排序、快速排序会写算法和时间复杂度**

   合并排序

```
void MergeSort(Type a[], int left, int right)
    {
        if (left<right) {//至少有 2 个元素
        int i=(left+right)/2;   //取中点
        mergeSort(a, left, i);
        mergeSort(a, i+1, right);
        merge(a, b, left, i, right);   //合并到数组 b
        copy(a, b, left, right);       //复制回数组 a
        }
    }
```

   最坏时间复杂度：O(nlogn)
   平均时间复杂度：O(nlogn)
   辅助空间：O(n)

   快速排序

```
template<class Type>
int RandomizedPartition (Type a[], int p, int r)
{
            int i = Random(p,r);
            Swap(a[i], a[p]);
            return Partition (a, p, r);
}
```

   最坏时间复杂度：O(n2)
   平均时间复杂度：O(nlogn)
   辅助空间：O(n)或 O(logn)

3. **哪些问题可以使用分治法（参看课本）**

第四章 贪心策略（重点）

1. **贪心策略两基本要素：最优子结构、贪心选择**

2. **什么是最优子结构**

   当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。问题的最优子结构性质是该问题可用动态规划算法或贪心算法求解的关键特征。

3. **活动安排问题（一定掌握）**

应用实例

```
template<class Type>
void GreedySelector(int n, Type s[],
Type f[], bool A[]) {
    A[1] = true;
    int j = 1;
    for (int i=2;i<=n;i++) {
        if (s[i]>=f[j]) {
            A[i]=true;
            j=i;
        }
        else A[i]=false;
    }
}
```

各活动起始时间与结束时间分别存储于数组s和f中，且f中活动按结束时间非递减排序，即$f_1 \le f_2 \le \ldots \le f_n$。

时间复杂度O(n)
预排序O(nlogn)

该算法的贪心选择的意义是**使剩余的可安排时间段极大化**，以便安排尽可能多的相容活动。

- 活动安排问题—实例计算过程

设待安排的11个活动的开始时间和结束时间按结束时间的非减序排列如下：

i	1	2	3	4	5	6	7	8	9	10	11
s[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

首先选定活动1，其结束时间f[1]=4。
然后因s[4] = 5≥4，选定活动4，这时f[4] = 7。
又因s[8] = 8≥7，选定活动8，这时f[8] = 11。
最后因s[11] = 12≥11，选定活动11。

4. 背包问题（会写程序和时间复杂度）

背包问题的贪心算法

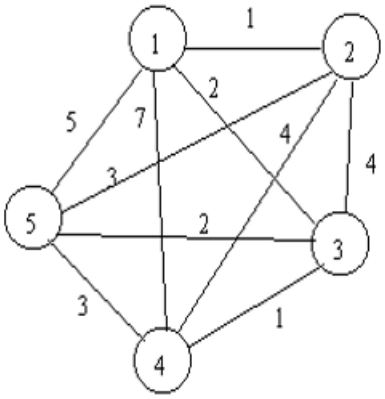
```
void Knapsack(int n, float M, float v[], float
w[], float x[]) {
    Sort (n, v, w);
    int i;
    for (i=1; i<=n; i++) x[i]=0;
    float c = M;
    for (i=1; i<=n; i++) {
        if (w[i] > c) break;
        x[i]=1;
        c -= w[i];
    }
    if (i<=n) x[i]=c/w[i];
}
```

对于0-1**背包问题**，贪心选择之所以不能得到最优解是因为在这种情况下，它无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。
动态规划

算法knapsack的主要计算时间在于将各种物品依其单位重量的价值从大到小排序。因此，算法的计算时间上界为O(nlogn)。
为了证明算法的正确性，还必须证明背包问题具有贪心选择性质。

5. 旅行商问题（会方法和时间复杂度）

旅行商算法举例



从节点1出发：
Path=<1,2,5,3,4,1> ;cost=14;

不难看出，从节点2出发：
Path=<2,1,3,4,5,2> ;cost=10;
且此为最优解！

因此最临近法不保证求得旅行商问题的精确解，只能得到问题地近似解。一般地，贪心选择只依赖于前面选择步骤地最优性，因此是局部最优的，所以贪心法不能够确保求出问题的最优解。

改进的旅行商算法

- 算法Tray_Greedy1：如果分别从节点i出发（i=1,2,..n）执行算法Tray_Greedy，通过结果比较，取最小代价回路，可以求得更接近于最佳解的近似解。

- Tray_Greedy算法有两层循环，外层循环为n次，内层循环也是n次，因此Tray_Greedy算法的时间复杂度为 $O(n^2)$

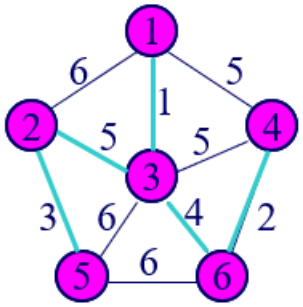
- Tray_Greedy1 算法分别从n个节点出发计算最小代价回路，其时间复杂度为$O(n^3)$

6. **装载问题**
   将集装箱按从轻到重排序，轻者先装。

7. **构造最小生成树（必会）**
   Prim 算法的做法：在保证连通的前提下依次选出权重较小的 n－1 条边（在实现中体现为 n 个顶点的选择）。

Prim算法的示例

- 给定一个连通带权图如下：

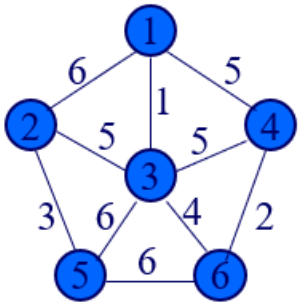


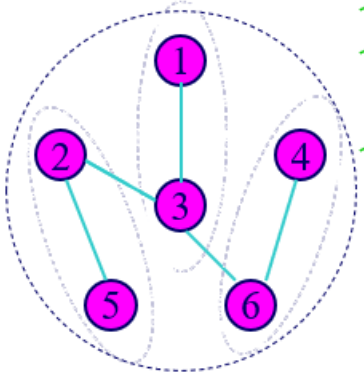
- 初始时S={1}，T= Φ;
- 第五次选择：
  ∵(5, 2)权最小
  ∴ S={1, 3, 6, 4, 2, 5}，
  T= {(1, 3), (3, 6), (6, 4), (3, 2) (2, 5)} ；

Kruskal 算法的做法：在保证无回路的前提下依次选择权重较小的 n－1 条边。

Kruskal算法的例子

已经选择边了n–1条边，算法结束。结果如图所示。





2、√ / √ √ × √

u	v	w
1	3	1
4	6	2
2	5	3
3	6	4
1	4	5
2	3	5
3	4	5
1	2	6
3	5	6
5	6	6

8. 单源最短路径问题

Dijkstra 做法 O(n2)

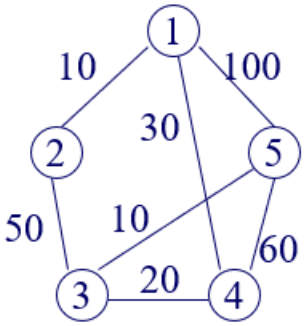
1）由近到远逐步计算，每次最近的顶点的距离就是它的最短路径长度。

2）然后再从这个最近者出发。即依据最近者修订到各顶点的距离，然后再选出新的最近者。

3）如此走下去，直到所有顶点都走到。

Dijkstra算法举例

赋权图G



迭代	S	u	dis[2]	dis[3]	dis[4]	dis[5]
初始	{1}	--	**10**	∞	30	100
1	{1,2}	2	10	60	**30**	100
2	{1,2,4}	4	10	**50**	30	90
3	{1,2,4,3}	3	10	50	30	**60**
4	{1,2,4,3,5}	5	10	50	30	60

由数组dis[i]可知：从顶点1到顶点2、3、4、5的最短通路的长度分别为10、50、30和60。

9. 哈夫曼编码

1)以 n 个字母为结点构成 n 棵仅含一个点的二叉树集合,字母的频率即为结点的权 .

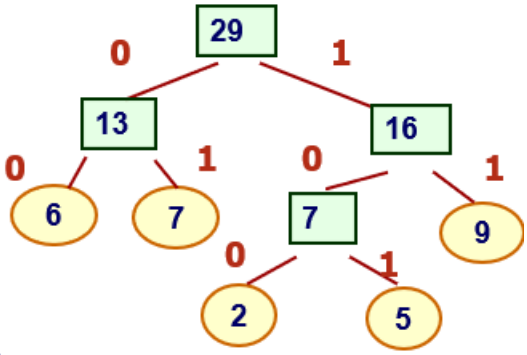
2)每次从二叉树集合中找出两个权最小者合并为一棵二叉树:增加一个根结点将这两棵树作为左右子树.新树的权为两棵子树的权之和.
3)反复进行步骤 2)直到只剩一棵树为止.

例：假设有5个符号以及它们的频率：

A	B	C	D	E
6	7	2	5	9

求前缀编码。

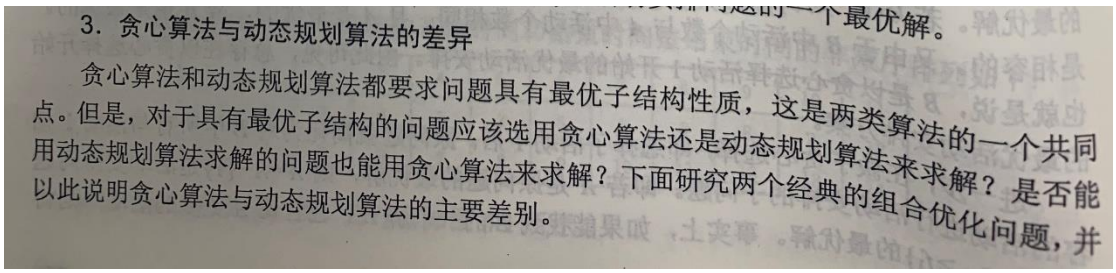
1、建立赫夫曼树
2、对边编号
3、求出叶子结点的路径
4、得到字符编码



6	7	2	5	9
A	B	C	D	E
00	01	100	101	11

10. 哪些问题可以使用贪心算法（参看课本）

11. 贪心算法与动态规划的异同（课本 P98）



3. 贪心算法与动态规划算法的差异

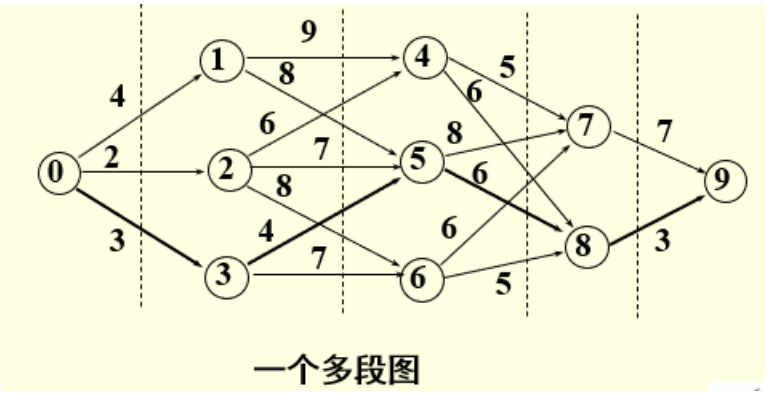
贪心算法和动态规划算法都要求问题具有最优子结构性质，这是两类算法的一个共同点。但是，对于具有最优子结构的问题应该选用贪心算法还是动态规划算法来求解？是否能用动态规划算法求解的问题也能用贪心算法来求解？下面研究两个经典的组合优化问题，并以此说明贪心算法与动态规划算法的主要差别。

第五章 动态规划

1. 哪些问题可以使用动态规划（参看课本）

2. 多段图的最短路径问题

例题：



一个多段图

过程：

对多段图的边(u, v)，用 cuv 表示边上的权值，将从源点 s 到终点 t 的最短路径记为 d(s, t)，则从源点 0 到终点 9 的最短路径 d(0, 9)由下式确定：

d(0, 9)=min{c01+d(1, 9), c02+d(2, 9), c03+d(3, 9)}

这是最后一个阶段的决策，它依赖于 d(1, 9)、d(2, 9)和 d(3, 9)的计算结果，而

d(1, 9)=min{c14+d(4, 9), c15+d(5, 9)}

d(2, 9)=min{c24+d(4, 9), c25+d(5, 9), c26+d(6, 9)}

d(3, 9)=min{c35+d(5, 9), c36+d(6, 9)}

这一阶段的决策又依赖于 d(4, 9)、d(5, 9)和 d(6, 9)的计算结果：

d(4, 9)=min{c47+d(7, 9), c48+d(8, 9)}

d(5, 9)=min{c57+d(7, 9), c58+d(8, 9)}

d(6, 9)=min{c67+d(7, 9), c68+d(8, 9)}

这一阶段的决策依赖于 d(7, 9)和 d(8, 9)的计算，而 d(7, 9)和 d(8, 9)可以直接获得（括号中给出了决策产生的状态转移）：

d(7, 9)=c79 = 7(7→9)

d(8, 9)=c89 = 3(8→9)

再向前推导，有：

d(6, 9)=min{c67+d(7, 9), c68+d(8, 9)}=min{6+7, 5+3}=8(6→8)

d(5, 9)=min{c57+d(7, 9), c58+d(8, 9)}=min{8+7, 6+3}=9(5→8)

d(4, 9)=min{c47+d(7, 9), c48+d(8, 9)}=min{5+7, 6+3}=9(4→8)

d(3, 9)=min{c35+d(5, 9), c36+d(6, 9)}=min{4+9, 7+8}=13(3→5)

d(2, 9)=min{c24+d(4, 9), c25+d(5, 9), c26+d(6, 9)}=min{6+9, 7+9, 8+8}=15(2→4)

d(1, 9)=min{c14+d(4, 9), c15+d(5, 9)}=min{9+9, 8+9}=17(1→5)

d(0, 9)=min{c01+d(1, 9), c02+d(2, 9), c03+d(3, 9)}=min{4+17, 2+15, 3+13}=16(0→3)

最后，得到最短路径为 0→3→5→8→9，长度为 16。

3. 基本要素：最优子结构、重叠子问题

4. **数字三角形问题（会递推公式）**

$$aMaxSum[r][j]=\begin{cases}D[r][j] & r=N\\ Max(aMaxSum[r+1][j],\\ \quad aMaxSum[r+1][j+1])+D[r][j] & \text{其他}\end{cases}$$

5. **投资分配问题**

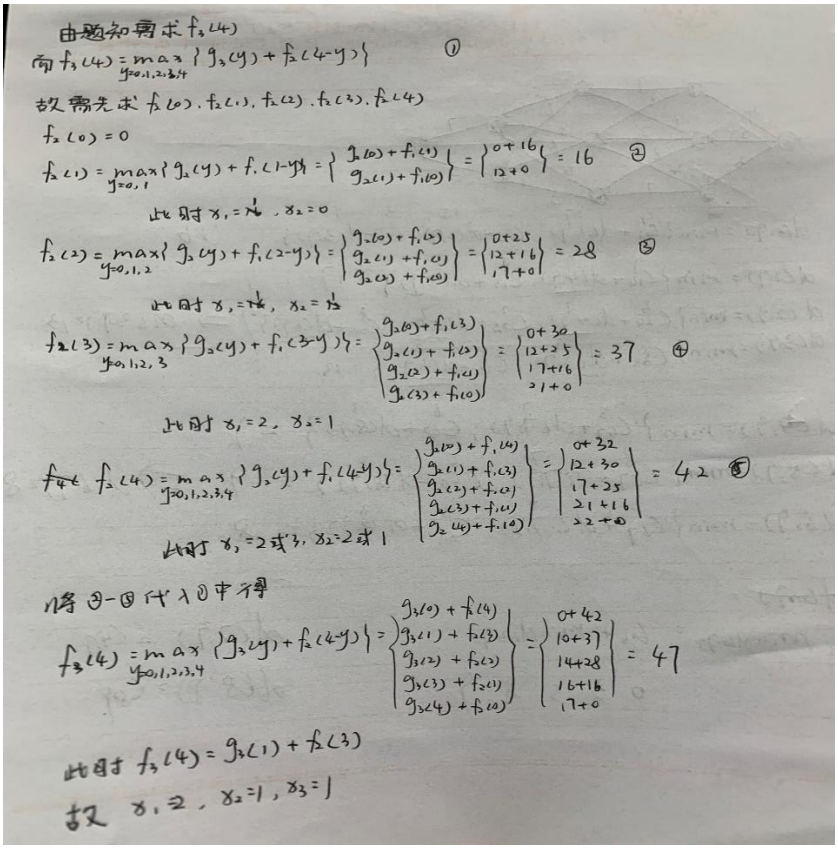
习题：

某公司打算在 3 个不同的地区设置 4 个销售点，根据市场部门估计，在不同地区设置不同数量的销售点每月可得到的利润如表所示。试问在各地区如何设置销售点可使每月总利润最大。

地区	销售点				
	0	1	2	3	4
1	0	16	25	30	32
2	0	12	17	21	22
3	0	10	14	16	17

$x_1=2$，$x_2=1$，$x_3=1$，$f_3(4)=47$

过程：



6. 0-1 背包问题（掌握并会写程序）

实例： 有5个物品，其重量分别是{2, 2, 6, 5, 4}，价值分别为{6, 3, 5, 4, 6}，背包的容量为10。

根据动态规划函数，用一个(n+1)×(C+1)的二维表V，V[i][j]表示把前i个物品装入容量为j的背包中获得的最大价值。

$$V(i,j)=\begin{cases} V(i-1,j) & j < w_i \\ \max\{V(i-1,j),\ V(i-1,j-w_i)+v_i\} & j \ge w_i \end{cases}$$

		0	1	2	3	4	5	6	7	8	9	10	
	0	0	0	0	0	0	0	0	0	0	0	0	
w_1=2 v_1=6	1	0	0	6	6	6	6	6	6	6	6	6	x_1=1
w_2=2 v_2=3	2	0	0	6	6	9	9	9	9	9	9	9	x_2=1
w_3=6 v_3=5	3	0	0	6	6	9	9	9	9	11	11	14	x_3=0
w_4=5 v_4=4	4	0	0	6	6	9	9	9	10	11	13	14	x_4=0
w_5=4 v_5=6	5	0	0	6	6	9	9	12	12	15	15	15	x_5=1

算法实现：

设 n 个物品的重量存储在数组 w[n]中，价值存储在数组 v[n]中，背包容量为 C，数组 V[n+1][C+1]存放迭代结果，其中 V[i][j]表示前 i 个物品装入容量为 j 的背包中获得的最大价值，数组 x[n]存储装入背包的物品，动态规划法求解 0/1 背包问题的算法如下：

```
int KnapSack(int n, int w[ ], int v[ ]) {
    for (i=0; i<=n; i++)     //初始化第 0 列
        V[i][0]=0;
    for (j=0; j<=C; j++)     //初始化第 0 行
        V[0][j]=0;
  for (i=1; i<=n; i++)     //计算第 i 行，进行第 i 次迭代
        for (j=1; j<=C; j++)
            if (j<w[i])     V[i][j]=V[i-1][j];
            else     V[i][j]=max(V[i-1][j], V[i-1][j-w[i]]+v[i]);

    j=C;      //求装入背包的物品
    for (i=n; i>0; i--){
        if (V[i][j]>V[i-1][j]) {
            x[i]=1;
            j=j-w[i];
        }
        else x[i]=0;
    }
```

```
        return V[n][C];      //返回背包取得的最大价值
}
```

7. 最长公共子序列问题

例：序列X=(a, b, c, b, d, b)，Y=(a, c, b, b, a, b, d, b, b)，建立两个(m+1)×(n+1)的二维表L和表S，分别存放搜索过程中得到的子序列的长度和状态。

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1
2	0	1	1	2	2	2	2	2	2	2
3	0	1	2	2	2	2	2	2	2	2
4	0	1	2	3	3	3	3	3	3	3
5	0	1	2	3	3	3	3	4	4	4
6	0	1	2	3	4	4	4	4	5	5

(a) 长度矩阵L

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	2	2	1	2	2	2	2
2	0	3	2	1	1	2	1	2	1	1
3	0	3	1	2	2	2	2	2	2	2
4	0	3	3	1	1	3	1	3	1	1
5	0	3	3	3	2	2	2	1	2	2
6	0	3	3	1	1	3	1	2	1	1

(b) 状态矩阵S

$$L[0][0]=L[i][0]=L[0][j]=0(1\le i\le m,1\le j\le n) \quad \text{（式1）}$$

$$L[i][j]=\begin{cases} L[i-1][j-1]+1 & x_i=y_j, i>1, j>1 \\ \max\{L[i][j-1],L[i-1][j]\} & x_i\neq y_j, i>1, j>1 \end{cases} \quad \text{（式2）}$$

第七章 1 回溯法

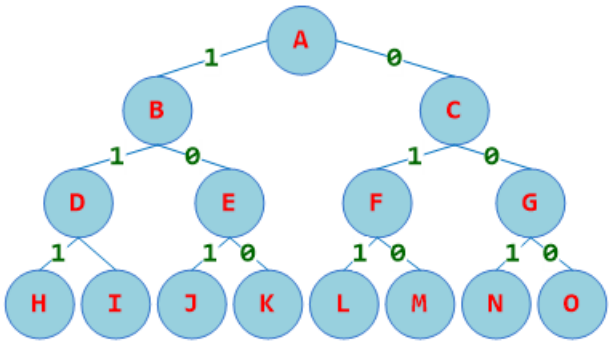
1. 哪些问题可以使用回溯法（参照课本）

2. 回溯法解 0/1 背包问题（解空间、可行解、最优解）

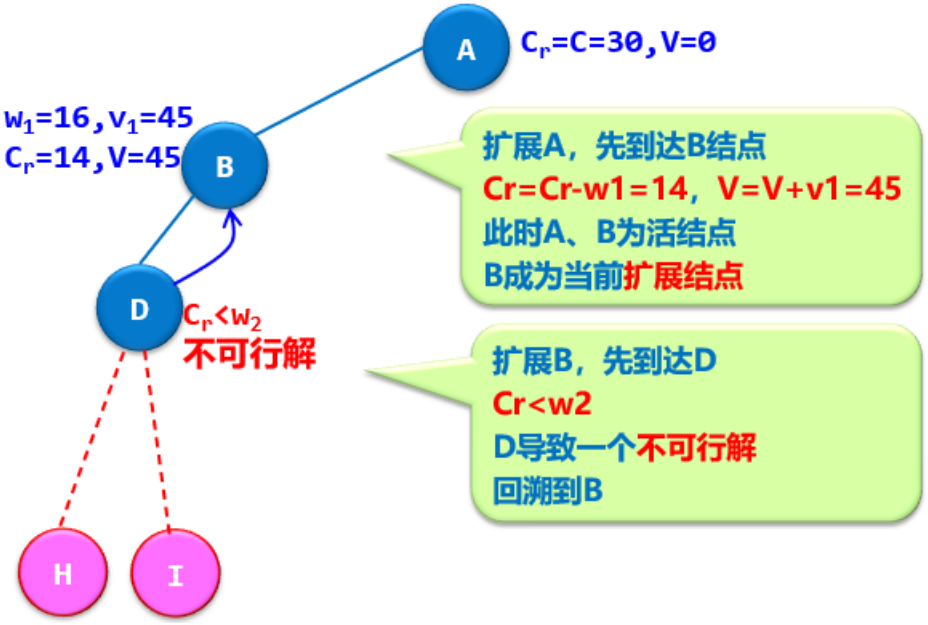
- 对于有n种可选物品的0-1背包问题，其解空间由2^n个长度为n的0-1向量组成。
- n=3时，解空间为
  {(0,0,0),(0,0,1),(0,1,0),(0,1,1),
  (1,0,0),(1,0,1),(1,1,0),(1,1,1)}

0-1背包问题

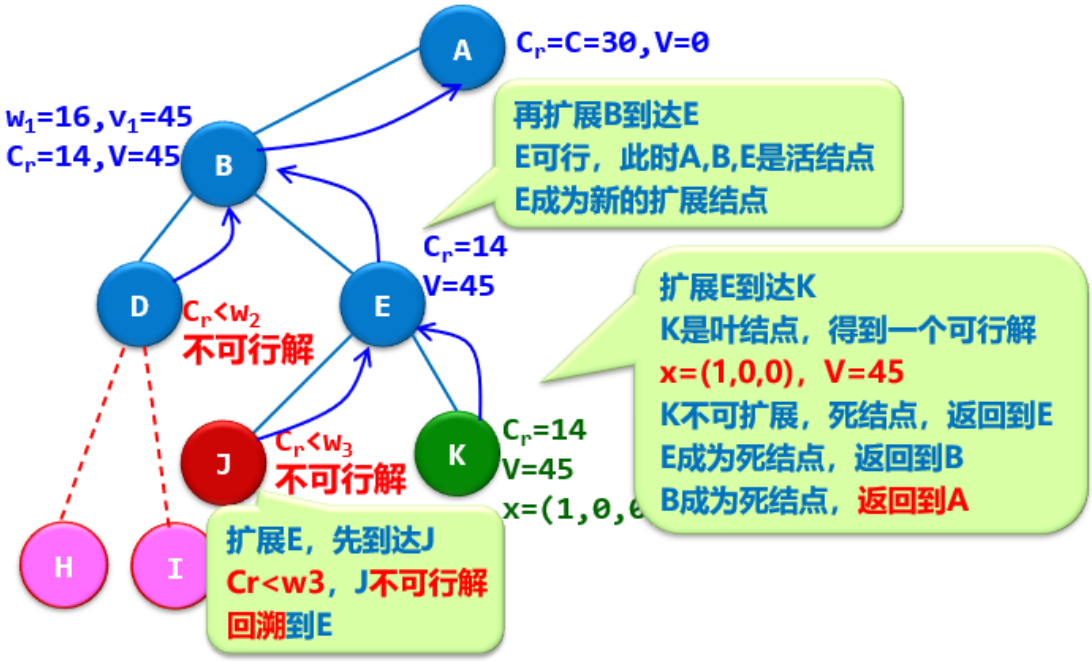
- n=3, C=30, w={16, 15, 15}, v={45,25,25}
- 开始时，
  - **Cr=C=30, V=0**
    - C为容量，Cr为剩余空间，V为价值。
  - A为唯一活结点，也是当前扩展结点。



- n=3, C=30, w={16,15,15}, v={45,25,25}

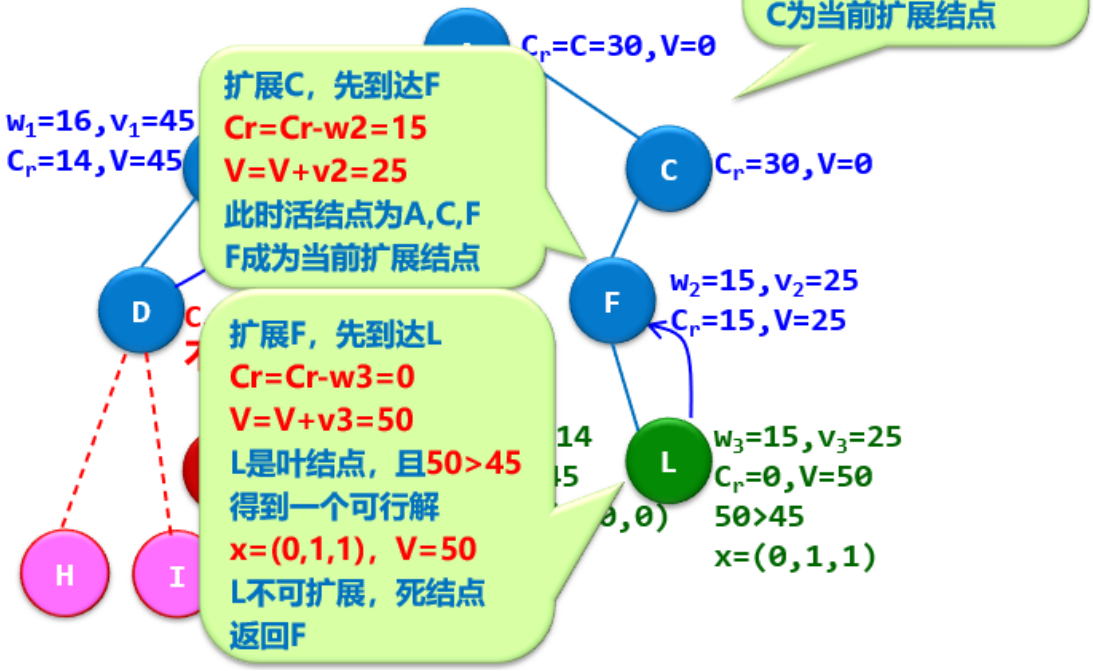


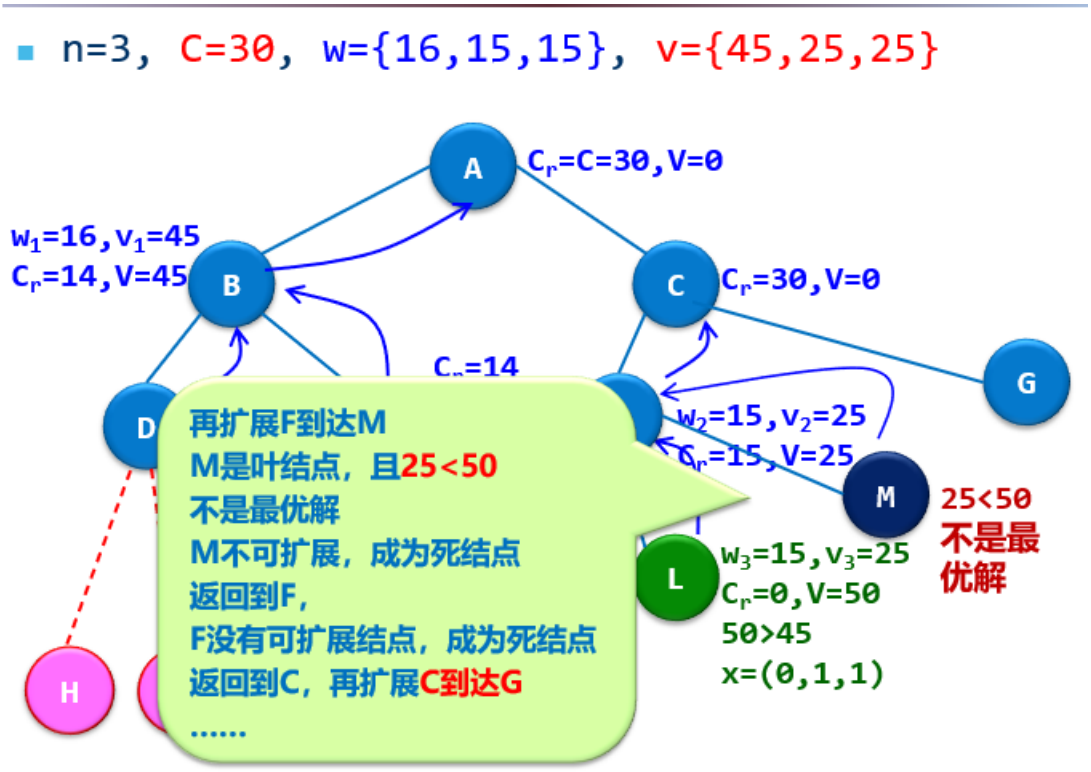
- n=3, C=30, w={16,15,15}, v={45,25,25}



6.1 回溯法的算法框架

- n=3, C=30, w={16,15,15}, v={4





3. **子集树和排列树（区分什么问题是子集树，什么问题是排列树）**

   0-1 背包问题的解空间是一棵子集树

   旅行售货员问题的解空间是一颗排列树

4. **n 皇后问题（知道过程）**

5. **装载问题（必会）**

装载问题

- 4 算法描述

```
public static void Backtrack(int i)
{
    if (i >= n)//到达叶结点
    {
        if (cw > bestw)//当前解最优，更新
        { for (int j = 0; j < n; j++) bestx[j] = x[j];//更新解
          bestw = cw;}
        return;
    }
    r -= w[i];//剩余货箱重量之和
    if (cw + w[i] <= c)//搜索左子树，前这件可以装进来
    {
        x[i] = 1; cw += w[i];//更新当前重量
        Backtrack(i + 1);//搜索下一结点
        cw -= w[i];//回溯到上一结点
    }
    //当前+剩余有可能大于最优解，搜索右子树
    if (cw + r > bestw)
    { x[i] = 0; Backtrack(i + 1); }
    r += w[i];
}
```

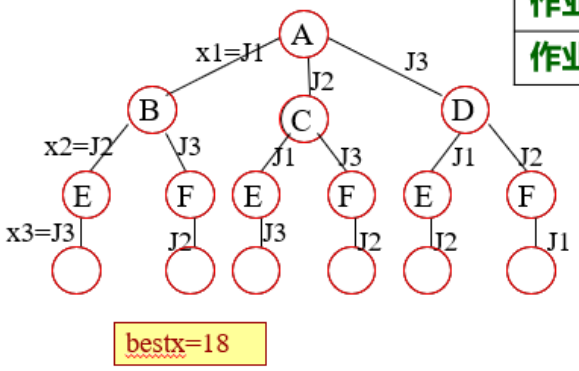
6. 作业调度问题（会画图）

算法设计与分析 >回溯法 >作业调度

例题

t_{ji}	机器1	机器2
作业1	2	1
作业2	3	1
作业3	2	3

F_{ij}	机器1	机器2
作业1	$F_{11}=2$	$F_{21}=3$
作业3	$F_{13}=4$	$F_{23}=7$
作业2	$F_{12}=7$	$F_{22}=8$



bestx=18

7. 着色问题（过程看一下）

第七章 2 分支限界法

1. 回溯法和分支限界法的异同

相同点

1）均需要先定义问题的解空间，确定的解空间组织结构一般都是树或图。
在问题的解空间树上搜索问题解。

2）搜索前均需确定判断条件，该判断条件用于判断扩展生成的结点是否为可行结点。

3）搜索过程中必须判断扩展生成的结点是否满足判断条件，如果满足，则保留该扩展生成的结点，否则舍弃。

不同点

1）搜索目标：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找出满足约束条件的一个解，或是在满足约束条件的解中找出在某种意义下的最优解。

2）搜索方式不同：回溯法以深度优先的方式搜索解空间树，而分支限界法则以广度优先或以最小耗费优先的方式搜索解空间树

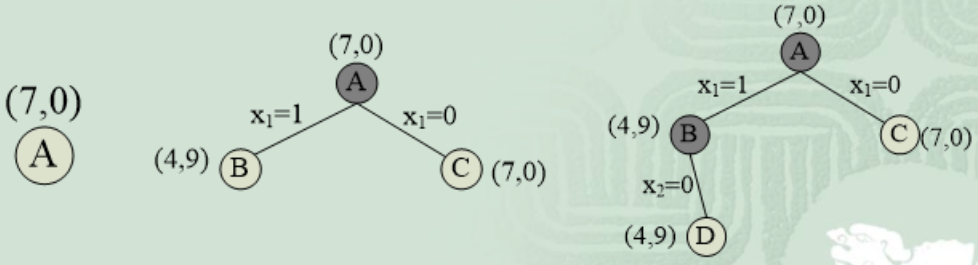
3）扩展方式不同：在回溯法搜索中，扩展结点一次生成一个孩子结点，而在分支限界法搜索中，扩展结点一次生成它所有的孩子结点。

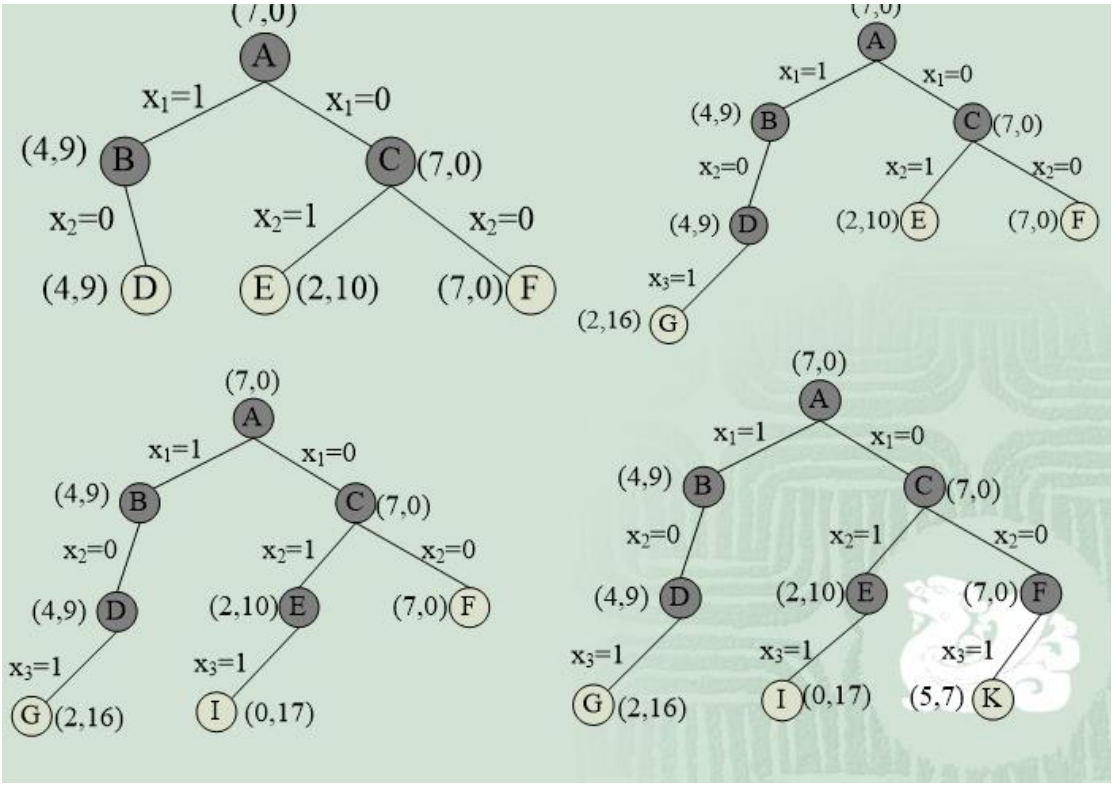
2. 0-1 背包问题

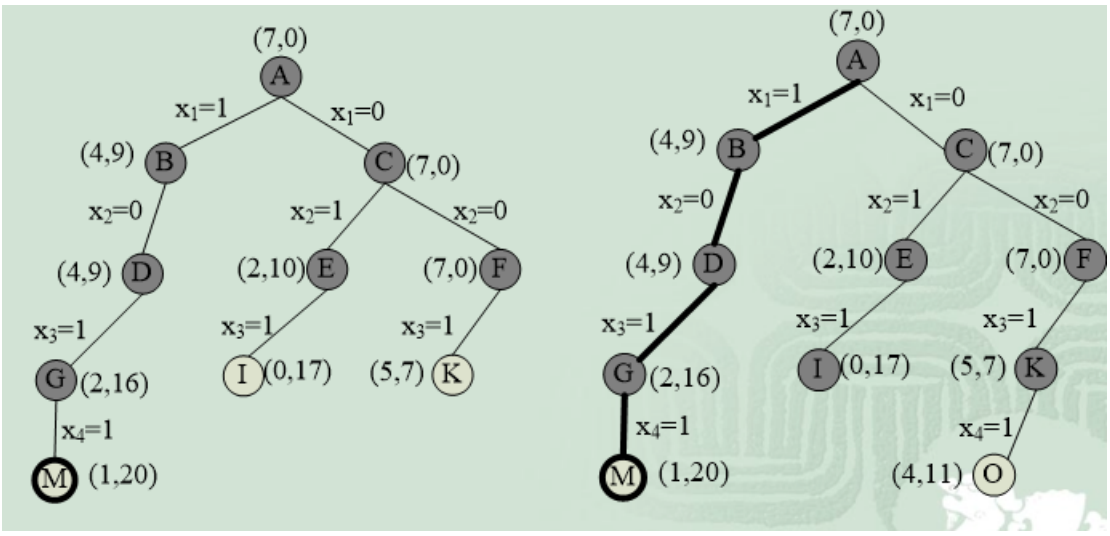
示例1：0-1背包问题

- 考虑实例n=4，w=[3,5,2,1]，v=[9,10,7,4]，C=7。
- 定义问题的解空间
  - 该实例的解空间为（x1,x2,x3,x4），xi=0或1(i=1,2,3,4)。
- 确定问题的解空间组织结构
  - 该实例的解空间是一棵子集树，深度为4。
- 搜索解空间
  - 约束条件 $\sum_{i=1}^{n} w_i x_i \le C$
  - 限界条件 cp+rp>bestp

- cp初始值为0；rp初始值为所有物品的价值之和；bestp表示当前最优解，初始值为0。
- 当cp>bestp时，更新bestp为cp。




(7,0)
A
x1=1
x1=0
(4,9)
B
C
(7,0)
x2=0
x2=1
x2=0
(4,9)
D
E
(2,10)
(7,0)
F
x3=1
(2,16)
G
I
(0,17)
(5,7)
K

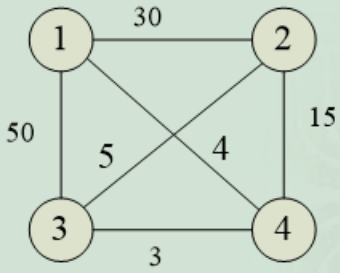

(7,0)
A
x1=1
x1=0
(4,9)
B
C
(7,0)
x2=0
x2=1
x2=0
(4,9)
D
E
(2,10)
F
(7,0)
x3=1
G
(2,16)
I
(0,17)
K
(5,7)
x4=1
M
(1,20)
O
(4,11)

3. 旅行商问题

示例2： 旅行售货员问题

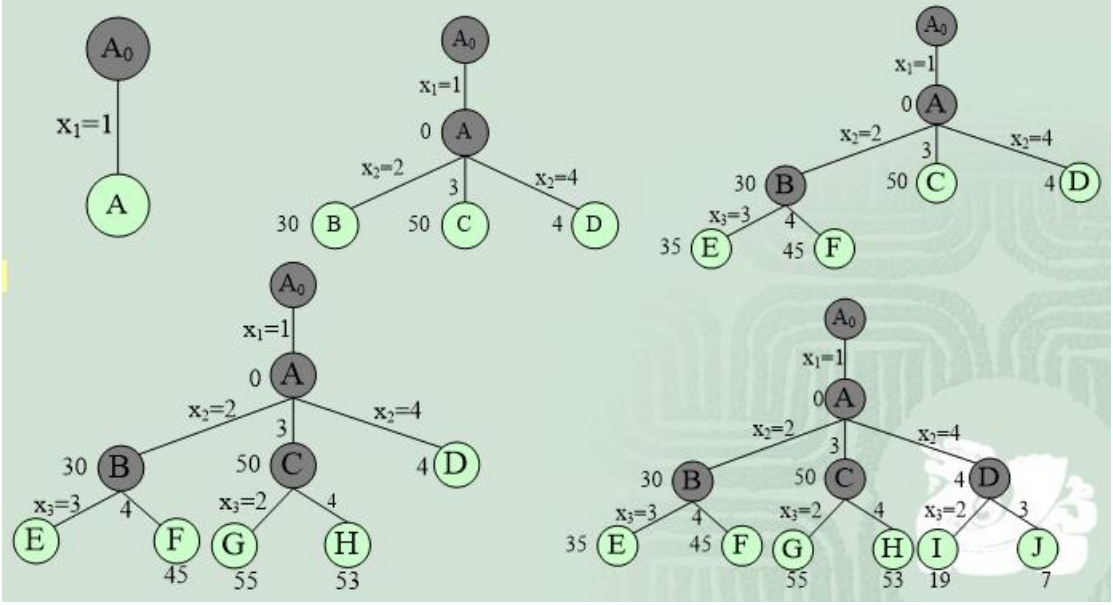
问题描述

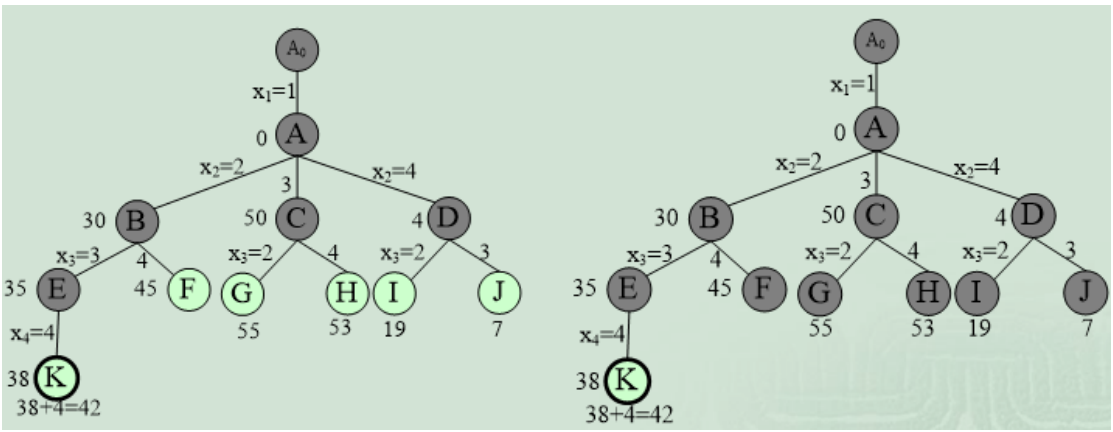
某售货员要到若干城市去推销商品，已知各城市之间的路程(或旅费)。他要选定一条从驻地出发，经过每个城市一次，最后回到驻地的路线，使总的路程(或总旅费)最小。



- 问题的解空间（**x1,x2,x3,x4**），其中令**S={1,2,3,4}， x1=1，x2∈S-{x1}，x3∈S-{x1,x2}，x4 ∈ S-{x1,x2,x3}**。
- 解空间的组织结构是一棵深度为**4**的排列树。
- 搜索
  - 约束条件**g[i][j]!=∞**，其中**g**是该图的邻接矩阵；
  - 限界条件：**cl<bestl**，其中**cl**表示当前已经走的路径长度，初始值为**0**；**bestl**表示当前最短路径长度，初始值为∞。

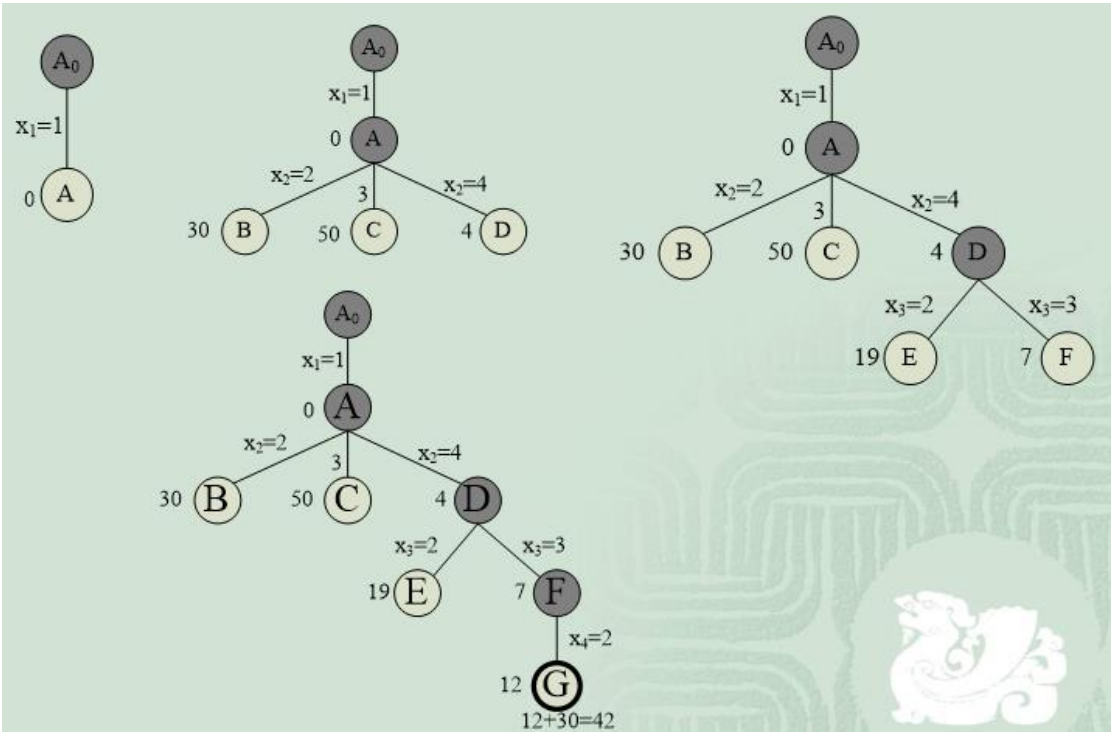
队列式分支限界法

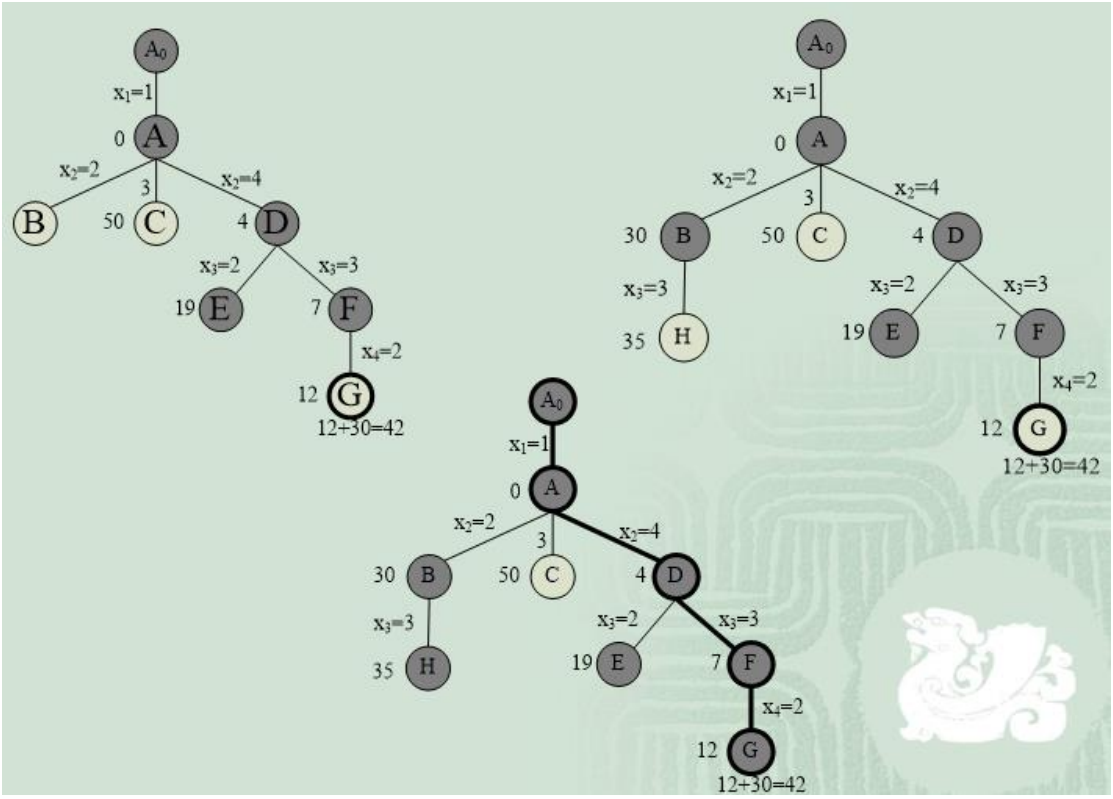




优先队列式分支限界法

- 优先级：活结点所对应的已经走过的路径长度cl，长度越短，优先级越高。


A0
x1=1
A
0
x2=2
3
x2=4
B
30
C
50
D
4
x3=2
x3=3
E
19
F
7
x4=2
G
12
12+30=42


A0
x1=1
A
0
x2=2
3
x2=4
B
30
C
50
D
4
x3=3
H
35
x3=2
E
19
F
7
x4=2
G
12
12+30=42

算法优化

- 估计路径长度的下界 用zl表示，zl=cl+rl
- 优先级：活结点的zl，zl越小，优先级越高；
- 约束条件：同上
- 限界条件：zl=cl+rl>bestl，