**第一章**

1-3.最大公约数为1。快1414倍。

程序1-2的while循环体做了10次，程序1-3的while循环体做了14141次（14142~2循环）

**第二章**

2-8.（1）画线语句的执行次数为。。

（2）画线语句的执行次数为。。

（3）画线语句的执行次数为。。

（4）当n为奇数时，画线语句的执行次数为：

，

当n为偶数时，画线语句的执行次数为：

。

渐进时间复杂度为。

**2-10.试用定义证明下列的渐进时间复杂度。**

**(1)**当时，，所以，可选**，**。对于，，这里𝑔 (*n*)=*n*2，所以，。

**(2)**当时，，所以，可选，。对于，，这里𝑔 (*n*)=*n*2，所以，。

注：当*n*=*n*0=3时，𝑓(𝑛)=5×32−8×3+2=23，𝑔(𝑛)=3×32=27，𝑓(𝑛)<𝑔(𝑛)；

当*n*=*n*0=4时，𝑓(𝑛)=5×42−8×4+2=50，𝑔(𝑛)=3×42=48，𝑓(𝑛)>𝑔(𝑛)。

**(3)**由（1）、（2）可知，取，，，当时，有，这里𝑔 (*n*)=*n*2，所以。

**2-11. 设有*f*(*n*)和*g*(*n*)如下所示，分析*f*(*n*)为O(*g*(*n*))、Ω(*g*(*n*))还是Θ(*g*(*n*))。**

**（1）*f*(*n*)=20*n*+log*n*，*g*(*n*)= *n*+log3*n***

当*n*≥1时，log*n*≤log3*n*，故*f*(*n*)=20*n*+log*n*≤20*n*+20log*n*=20(*n*+log*n*)≤20(*n*+log3*n*)=20*g*(*n*)

取*n*0=1，c=20，当*n*≥*n*0时，*f*(*n*)=O(*g*(*n*))。

当*n*≥1时，10*n*≥log3*n*，故*f*(*n*)=20*n*+log*n*≥20*n*≥*n*+10n ≥n+log3*n*=*g*(*n*)

取*n*0=1，c=1，当*n*≥*n*0时，*f*(*n*)=Ω(*g*(*n*))。

因此，取c1=1，c2=20，当*n*≥*n*0时，c1*g*(*n*)≤*f*(*n*)≤c2*g*(*n*)，即*f*(*n*)=Θ(*g*(*n*))。

**注：**原来的做法，是在当*n*≥3时，*n*≤log3*n*的条件下推得的，但是实际上当*n*>982后，*n*≥log3*n*。但是也给了我们启示，即我们可以从log进行化简，正如上面的*f*(*n*)=20*n*+log*n*≤20*n*+20log*n*=20(*n*+log*n*)。

**（2）*f*(*n*)=*n*2/log*n*，*g*(*n*)= *n*log2*n***

当*n*≥2时，*n*≥1/8\*log3*n*，即*n*/log*n*≥1/8\*log2*n*，故*f*(*n*)=*n*2/log*n*≥1/8\**n*log2*n*≥1/8\**g*(*n*)

取*n*0=2，c=1/8，当*n*≥*n*0时，*f*(*n*)=Ω(*g*(*n*))。

当*n*→∞时，*n*不可能小于log3*n*，*f*(*n*)不可能小于c*g*(*n*)，故*f*(*n*)≠O(*g*(*n*))，进而*f*(*n*)≠Θ(*g*(*n*))，所以取*n*0=2，c=1/8，当*n*≥*n*0时，*f*(*n*)=Ω(*g*(*n*))。

**（3）*f*(*n*)=(log*n*)log*n*，*g*(*n*)= *n*/log*n***

**方法一：**

令*x*=log*n*，a=2，即*n*=a*x*，则(log*n*)log*n*=*xx*，*n*/log*n*=a*x*/*x*。

当*x*>a同时a>1时，*xx*>a*x*。又当*x*>1时，a*x*>a*x*/*x*，因此*xx*>a*x*>a*x*/*x*。

将*x*=log*n*，a=2回代，可以得到：

*f*(*n*)=(log*n*)log*n*>2log*n*/log*n*=*n*/log*n*=*g*(*n*)，即取*n*0=2，c=1，当*n*>*n*0时，*f*(*n*)≥*g*(*n*)，*f*(*n*)=Ω(*g*(*n*))。

当然，反过来就是*g*(*n*)=O(*f*(*n*))。

**方法二：**

将*f*(*n*)和*g*(*n*)两边取log，得log*f*(*n*)=log*n*\*(loglog*n*)，log*g*(*n*)=log*n*-loglog*n*。

从上述两式可以看出，当n→∞，前一个式子明显快于后一个式子，因此*f*(*n*)≥*g*(*n*)，*f*(*n*)=Ω(*g*(*n*))。

从上述两式还可以看出，*f*(*n*)≠O(*g*(*n*))，进而*f*(*n*)≠Θ(*g*(*n*))。

**方法三：**

因为，。当时，，。所以，可选，，对于，，即。

注：这种证明方法是基于O((log*n*)log*n*)=O(*n*log(log(*n*))，即它们是同一个数量级的，但是在写法上不太正规。

2-12．

注： (换底公式：基本性质： )

2-13 证明：因为*f*1(*n*) =O(*g*1(*n*))，所以存在正常数c1和自然数*n*1，当*n*≥*n*1时，有*f*1(*n*)≤c1*g*1(*n*) 。

同理，因为*f*2(*n*) =O(*g*2(*n*))，所以存在正常数c2和自然数*n*2，当*n*≥*n*2时，有*f*2(*n*)≤c2*g*2(*n*) 。

取c=max{c1, c2}，*n*0=max{*n*1, *n*2}，则当*n*≥*n*0时，

有*f*1(*n*) +*f*2(*n*)

 ≤c1*g*1(*n*) + c2*g*2(*n*)

 ≤c*g*1(*n*) +c*g*2(*n*)

=c(*g*1(*n*) +g2(*n*))

 ≤2c(max{*g*1(*n*) , *g*2 (*n*)})

= O(max{𝑔1(𝑛),𝑔2(𝑛)})

证毕！

2-16 *n*的阶乘公式为：*n*!=1×2×…(*n*-1)×*n*，0!=1,递归定义：*n*!= (*n*-1)!×*n*

根据*n*的阶乘公式，可得下列递归函数：

int f(int n)/\*递归函数\*/

{ if(n>0)return f(n-1)\*n;

return 1;

}

分析递归算法的时间一般需要列出递推式。根据上述程序，可以得到求*n*的阶乘的递推式为

(1)用替换法求

T(0)=2=0×2+2, T(1)=1×2+2=4, T(2)=2×2+2=6,……, 猜测T(*n*)=2*n*+2。

用归纳法证明

当*n*=0时，T(0)=2，结论正确。

假设当*k*<*n*时，T(*k*)=2*k*+2成立，

那么当*k*=*n*时，T(*n*)=T(*n*-1)+2=2(*n*-1)+2+2=2*n*+2。

因此，对所有*n*≥0，有T(*n*)=2*n*+2=Θ(*n*)。

(2)用迭代法求

T(*n*)=T(*n*-1)+2=T(*n*-2)+2+2=T(*n*-3)+2+2+2=……=T(0)+2+2+…..+2(*n*个2相加)

=2*n*+2=Θ(*n*)。

因此，对所有*n*≥0，有T(*n*)=2*n*+2=Θ(*n*)

2-17. 证明：设，则。

























当时，。所以，。

2-20使用下列数据计算主定理的递推式：

(1)*a*=1，*b*=2，*f*(*n*)=*cn*

解：logba=log21=0, *n*logba=1, f(n)=O(n), 取ε=1，f(n)=Ω(nlogba+ε)=Ω(n), 且af(n/b)=n/2,

为使af(n/b)≤cf(n), 即n/2≤cn, 取c=2/3, 满足主定理2-5（3），

故递推式为 T(n)=aT(n/b)+f(n)=T(n/2)+2/3\*n

(2)*a*=5，*b*=4，*f*(*n*)=*cn*2

解：logba=log45，f(n)=O(n2)=O(nlog45+ε),ε=2-log45>0,

af(n/b)=5f(n/4)=5\*n2/16≤cn2, 取c=1/2(=8/16)<0，满足主定理2-5（3）

故递推式为 T(n)=aT(n/b)+f(n)=5T(n/4)+n2

**第五章**

5-4.SolutionType DandC1(intleft,int right)

{ while(!Small(left,right)&&left<right)

{ int m=Divide(left,right);

if(x<P(m) right=m-1;

else if(x>P[m]) left=m+1;

else return S(P)

}

}

**5-5将程序5-5求最大、最小元问题算法转换成计算上等价的迭代过程。并分析你的算法的执行步数。**

**提示：实现PPT上的方法四。**

template<class T>

voidSortableList<T>::MaxMin\_4(T& max, T& min)

{ //前置条件：n>1

int k;

k=(n+1)/2; //两两一组，n个元素共有k组，编号0~k-1。/主要是考虑n奇数或偶数的问题以及赋值给

//整型变量k时取整的问题，所以用(n+1)后除以2。数学表示上，n个元素有n/2组。

if(l[0]<l[1]) { max=l[1]; min=l[0]; } //处理0号组两个数据，比较1次

else{ max=l[0]; min=l[1]; }

if(n%2) k--; //如果n是奇数，最后一组就只有1个元素，下面的循环就不处理最后一组，故k--。

for(inti=1; i<k; i++) //n为偶数时，做了n/2-1组，每组比较3次，共比较3(n/2-1)次；

{ if(l[2\*i]<l[2\*i+1]) //n为奇数时，做了n/2-2组，每组比较3次，共比较3(n/2-2)次。

{ if(l[2\*i]<min) min=l[2\*i];

if(l[2\*i+1]>max) max=l[2\*i+1];

}

else

{ if(l[2\*i]>max) max=l[2\*i];

if(l[2\*i+1]<min) min=l[2\*i+1];

}

}

//n是奇数时，最后一组只有一个元素，不需要组内元素两两比较，直接与max或min比较。

if(n%2) //n为奇数时，只比较1次；n为偶数时，不做下面的比较，即比较0次。

{ if(l[2\*i]>max) max=l[2\*i];

else if(l[2\*i]<min) min=l[2\*i];

}

} //该方法，当n为偶数时，总的比较次数为1+3(n/2-1)=3n/2-2次；

// 当n为奇数时，总的比较次数为1+3(n/2-2)+1=3n/2-4次。

//故该方法最坏情况下总的比较次数为3n/2-2次。

**5-7设计一个二分搜索算法。它将原集合分成1/3和2/3大小的两个子集合。并证明你的算法的正确性。**

template<class T>

intSortableList<T>::BSearch(constT&x) const

{

returnBSearch(x,0,n-1);

}

template<class T>

intSortableList<T>::BSearch(constT&x,intleft,int right) const

{ if (left<=right)

{ int m=left+(right-left+1)/3; //①

//int m=(left+right)/3; //②公式是错的

if (x<l[m]) return BSearch(x,left,m-1);

else if (x>l[m]) return BSearch(x,m+1,right);

else return m;

}

return -1;

}

**注意**：应该用m=left+(right-left+1)/3计算m，不能用m=(left+right)/3 ，两者不同。

同学们可能是受对半搜索的影响，但是m=(left+right)/2和m=left+(right-left+1)/2是一样的（因为要取整）。

将上述程序中的语句①去掉，换成语句②，用归纳法可以证明该程序是错的。

当left=right时，m=(left+right)/3=(2/3)left<left，而m应该满足：left≤m≤right，这样求出的m不能落在[left, right]中（其实当left=right时，m应该等于left，即m=left），所以使用m=(left+right)/3是错误的，进而程序也是不正确的。而在相同情况下（left=right时），m=left+(right-left+1)/3=left+1/3，取整后m=left，落在[left, right]中。

我们对a[3]={1,2,3}, 搜索x=3时，/程序出错，搜索不到3。在对右边子序列尤其是搜索最右边一个元素时出错。原因是：搜索过程执行到left=right=2时，m=(left+right)/3=1，由于x=3>l[1]=2, 所以继续执行 return BSearch2(x,m+1,right)，进入BSearch2(x,m+1,right)后，计算出来的left和right都还是2，而m计算出来还是等于1，由此进入到死循环，导致系统栈出现“Stack Overflow”错误，程序结束不了。说明m=(left+right)/3计算公式有问题。

**5-8 三分搜索算法的做法是：它先将待查元素*x*与*n*/3处的元素比较，然后将*x*与2*n*/3处的元素比较。比较的结果或者是找到*x*，或者将搜索范围缩小到原来的*n*/3。**

**（1）编写C++程序实现算法；**

**（2）分析算法时间复杂度。**

template<class T>

intSortableList<T>::Search(T x)const //返回搜索的元素*x*的下标

{ return Search(0, n-1, x);

}

template<class T>

intSortableList<T>::Search(int left, int right, T x)const //平均时间复杂度为

{ int m1,m2;

if (left==right) { if (l[left]==x) return left; }

if (left<right)

{ m1=left+(right-left)/3;//左分界点

m2=(int)ceil((double)(right-(right-left)/3));//右分界点

if (x==l[m1]) return m1;

else if (x<l[m1]) return Search(left,m1-1,x);

else if (x<l[m2]) return Search(m1+1,m2-1,x);

else if(x==l[m2]) return m2;

else return Search(m2+1,right,x);

}

else return -1;

}

5-9



4

5

3

1

2

6

7

8

0

1

2

3

4

5

6

0

证明：因为该算法在成功搜索的情况下，关键字之间的比较次数至少为，至多为。在不成功搜索的情况下，关键字之间的比较次数至少为，至多为。所以，算法的最好、最坏情况的时间复杂度为。

假定查找表中任何一个元素的概率是相等的，为，那么，

不成功搜索的平均时间复杂度为，

成功搜索的平均时间复杂度为。

其中，是二叉判定树的内路径长度，是外路径长度，并且。

**5-11.对两组数据：(1,1,1,1,1)和(5,5,8,3,4,3,2)执行程序5-12的快速排序，按照表5-11的格式分别列表表示执行过程。**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 步数 | 0 | 1 | 2 | 3 | 4 | 5 |
| 初始时 |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 排序结果 |  |  |  |  |  |  |

解答：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 步数 | 0 | 1 | 2 | 3 | 4 | 5 |
| 初始时 | 1(0) | 1(1) | 1(2) | 1(3) | 1(4) | ∞ |
| 1 | [1(3) | 1(4)] | 1(0) | [1(2) | 1(1)] | ∞ |
| 2 | [1(4)] | 1(3) | 1(0) | [1(2) | 1(1)] | ∞ |
| 3 | 1(4) | 1(3) | 1(0) | [1(2) | 1(1)] | ∞ |
| 4 | 1(4) | 1(3) | 1(0) | [1(1)] | 1(2) | ∞ |
| 排序结果 | 1(4) | 1(3) | 1(0) | 1(1) | 1(2) | ∞ |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 步数 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 初始时 | 5 | 5 | 8 | 3 | 4 | 3 | 2 | ∞ |
| 1 | [4 | 2 | 3 | 3] | 5 | [8 | 5] | ∞ |
| 2 | [3 | 2 | 3] | 4 | 5 | [8 | 5] | ∞ |
| 3 | [3 | 2] | 3 | 4 | 5 | [8 | 5] | ∞ |
| 4 | [2] | 3 | 3 | 4 | 5 | [8 | 5] | ∞ |
| 5 | 2 | 3 | 3 | 4 | 5 | [5] | 8 | ∞ |
| 排序结果 | 2 | 3 | 3 | 4 | 5 | 5 | 8 | ∞ |

5-12.（1）证明：当或或时，程序显然正确。

当n=right-left+1>2时，程序执行下面的语句：

int k=(right-left+1)/3;

StoogeSort(left,right-k);

StoogeSort(left+k,right);

StoogeSort(left,right-k);

①首次递归StoogeSort(left,right-k);时，序列的前2/3的子序列有序。

②当递归执行StoogeSort(left+k,right);时，使序列的后2/3的子序列有序，经过这两次递归排序，使原序列的后1/3的位置上是整个序列中较大的数，即序列后1/3的位置上数均大于前2/3的数，但此时，前2/3的序列并不一定是有序的。

③再次执行StoogeSort(left,right-k);使序列的前2/3有序。

经过三次递归，最终使序列有序。

所以，这一排序算法是正确的。

（2）最坏情况发生在序列按递减次序排列。

，，。

设，则。















冒泡排序最坏时间复杂度为，堆排序最坏时间复杂度为，快速排序最坏时间复杂度为。所以，该算法不如冒泡排序，堆排序，快速排序。

**5-13.设有两个长度分别为m和n的已排序的表，设计一个有效算法在两个表的全部元素中求第k小元素，使得时间复杂度为O(log(max(m,n)))。**

//利用两个数组都是有序的条件，用二分查找。

//**算法思想**：

// 1）当a[k/2-1] == b[k/2-1]，则返回a[k/2-1]或者b[k/2-1] （允许有相同元素）

// 2）当a[k/2-1] > b[k/2-1]，则b在[0,k/2-1]内的元素一定比a、b合并后第k个元素小，故不用考虑b在[0,k/2-1]、

// 内的元素（去掉b的前k2个元素，故b=b+k2）

// 3）当a[k/2-1]< b[k/2-1]时，则a在[0,k/2-1]内的元素一定比a、b合并后第k个元素小，故不用考虑a在[0,k/2-1]

// 内的元素（去掉a的前k1个元素，故a=a+k1）

//**算法的递归形式**为：

// 1）a或者b为空时，返回 a[k-1]或者b[k-1]

// 2）k==1时，返回min(a[0],b[0])

// 3）若a[k/2-1] > b[k/2-1]时，递归调用Search(a,aL,lena-1,b+k2,0,lenb-k2-1,k-k2)；

// 若a[k/2-1] < b[k/2-1]时，递归调用 Search(a+k1,0,lena-k1-1,b,bL,lenb-1,k-k1)；

// 若a[k/2-1] == b[k/2-1]时，返回a[k/2-1]。

// 时间复杂度O(log(m+n))=O(log(max(m,n)))

template<class T>

intSortableList<T>::Search(int k) //非递归函数

{ if(m+n<k||k<=0)

{ cout<<"Out Of Bounds! ";

return -1;

}

return Search(a,0,m-1,b,0,n-1,k);

}

template<class T>

intSortableList<T>::Search(T \*a, intaL, intaR, T \*b, intbL, intbR, int k)

{//前置条件：m<n, 即要求a[]元素个数<b[]元素个数

int lena=aR-aL+1;

int lenb=bR-bL+1;

if(lena>lenb) //本次递归调用交换a、b数组及其下标，目的是使其满足前置条件lena<lenb

return Search(b,bL,bR,a,aL,aR,k);

if(lena==0) return b[k-1]; //a数组为空时

if(k==1) return a[0]<b[0]?a[0]:b[0]; //剩余的k==1时，输出a[0]与b[0]中较小的一个

/\*\*将k分为两部分，分别在a和b数组上查找\*\*/

int k1=k/2<lena?k/2:lena; //在a[]中查找k1个元素。

int k2=k-k1; //剩余的k2个元素在b[]中查找。

if(a[k1-1]>b[k2-1]) //b[k2-1]小，故跳过b数组中的k2个元素，即以b+k2为参数递归调用。

return Search(a,aL,lena-1,b+k2,0,lenb-k2-1,k-k2);

else if(a[k1-1]<b[k2-1]) //a[k1-1]小，故跳过a数组中的k1个元素，即以a+k1为参数递归调用。

return Search(a+k1,0,lena-k1-1,b,bL,lenb-1,k-k1);

else //当第k小元素在a[]、b[]中都存在且位于相同位置时，会出现a[k1-1]==b[k2-1]的情况。

return a[k1-1];

}

5-18 通过手算证明斯特拉森矩阵乘法的正确性。

**题解**：一般的分治法，是将n×n的A、B矩阵分别分成4个(n/2)×(n/2)的子矩阵。

=

则AB=C表示为，

其中C11＝A11B11+A12B21，C12＝A11B12+A12B22，C21＝A21B11+A22B21，C22＝A21B12+A22B22

斯特拉森分治法也是将n×n的A、B矩阵分别分成4个(n/2)×(n/2)的子矩阵，它首先计算如下7个子矩阵：

P=(A11+A22)(B11+B22)，Q=(A21+A22)B11，R=A11(B12-B22)，S=A22(B21-B11)，T=(A11+A12)B22，U=(A21-A11)(B11+B12)

V=(A12-A22)(B21+B22)。

然后计算C**’**11=P+S-T+V，C**’**12=R+T，C**’**21=Q+S，C**’**22=P+R-Q+U。

要证明C**’**11= C11，C**’**12= C12，C**’**21= C21，C**’**22= C22。

**证明**C**’**11= C11

∵C**’**11=P+S-T+V

=(A11+A22)(B11+B22)+ A22(B21-B11)- (A11+A12)B22+(A12-A22)(B21+B22)

= A11 B11+ A11 B22+ A22 B11+ A22 B22+A22 B21- A22 B11-A11B22-A12B22+A12B21+A12B22*-*A22 B21- A22 B22

=A11 B11+A12B21

= C11

∴C**’**11= C11，证毕。

同理可以证明C**’**12= C12，C**’**21= C21，C**’**22= C22。（同学们可以自己证明）

由此可得，斯特拉森矩阵乘法是正确的。

**第六章**

**6-1 设有背包问题实例*n*=7, *M*=15, (*x*0,*x*1,…,*x*6)=(2,3,5,7,1,4,1)，物品装入背包的收益为：(*p*0,*p*1,…,*p*6)=(10,5,15,7,6,18,3)。求这一实例的最优解和最大收益。**

01 2 3 4 5 601 2 3 4 5 6

解：由题可得：=(5, 1.67, 3, 1, 6, 4.5, 3)，

0 123456

所以，最优解为，或(*x*0,*x*1,*x*2,*x*3,*x*4,*x*5,*x*6)=(1, 1, 1, 1, 1, 2/3, 0)。

最大收益为=166/3=55.33

**6-8设*W*={3,7,8,9,15,16,18,20,23,25,28}，构造一棵最优3路合并树。**

**题解：**因**t**(n-1)%(K-1)=(11-1)%(3-1)=0, 故无需补充虚游程。

172

56

76

40

18

15

16

9

25

28

23

7

8

3

18

20

**6-9图G=(*V*,*E*)是一个无向连通图，*n*=|*V*|，*m*=|*E*|，且*m*=O(*n*1.99)，试问选择何种算法求最小代价生成树，是普里姆还是克鲁斯卡尔算法？**

**题解：**用普里姆算法。

因为图G是一个无向连通图。所以*n*-1≤*m*≤*n* (*n*-1)/2; O(*n*)≤O(*m*)≤O(*n*2);

克鲁斯卡尔对边数较少的带权图有较高的效率，而，此图边数较多，接近完全图，故选用普里姆算法。

**6-10令T是带权无向连通图G=(*V*,*E*)的一棵最小代价生成树，如果将图G的每条边的代价都增加相同的值*c*。试问T是否是新图的最小代价生成树（树的代价可以不同）？请证明你的结论。**

**题解：** T仍是新图的最小代价生成树。

证明：假设T不是新图的最小代价生成树，T’是新图的最小代价生成树，那么cost(T’)<cost(T)。有cost(T’)-*c*(*n*-1)<cost(T)-*c*(*n*-1)，即在原图中存在一棵生成树，其代价小于T的代价，这与题设中T是原图的最小代价生成树矛盾。所以假设不成立。证毕。

**6-11 证明若无向连通图不包含具有相同代价的边，则该图的最小代价生成树是唯一的。**

**证明**：假设该图的最小代价生成树不是唯一的，存在两棵不同的最小代价生成树与。设的边是，其中是图的顶点个数。

若与不同，则在中至少有一条边，使得不是中的边，但是是的边。因为是树，我们在中加上边，必有一条回路，而是树，所以中必存在某条边不在中。对于树，若以边置换则得到新的一棵树，但树的权，因为是最小代价生成树，故，由此推出，即。

因为是的边，且在中没有回路，故不可能成立，因为否则在中，自之后将取而不能取，与题设矛盾。于是，因此也是图的一棵最小代价生成树，但是与的公共边数比与的公共边数多1，用置换，重复上述论证，直到与有条公共边的最小代价生成树，这时就说明与是同一棵最小代价生成树。

**方法二：**

假设存在两棵不同的最小代价生成树与，它们的边分别是与。

设与的前边相同，与不同，即。

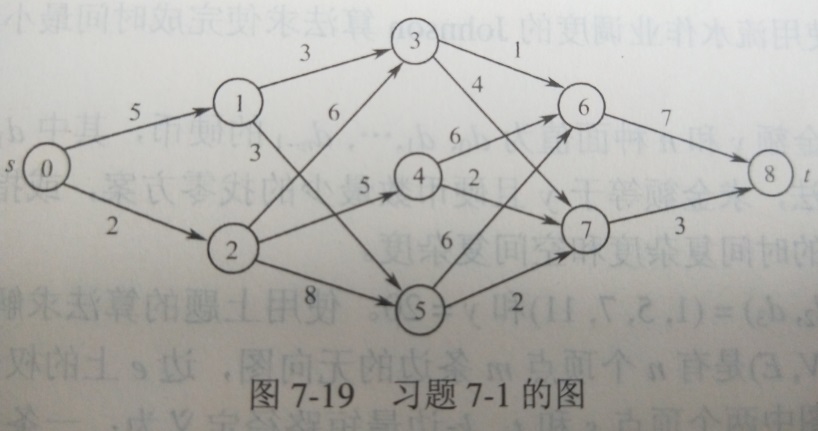
又因，

由于与都是最小代价生成树，因此，即，与矛盾，故与相同，是同一条边。

重复上述论证，直到与的边均为同一条边，这就说明与是同一棵最小代价生成树。

**第七章**

**7-1. 写出对图7-19所示的多段图采用向后递推动态规划算法求解时的计算过程。**

****

向后递推过程如下：

Bcost(1,0)=0 d(1,0)=-1;

Bcost(2,1)=c(1,1)+Bcost(1.0)=5 d(2,1)=0;

Bcost(2,2)=c(1,2)+Bcost(1,0)=2 d(2,2)=0;

Bcost(3,3)=min{c(2,3)+Bcost(2,2),c(1,3)+Bcost(2,1)}=min{6+2,3+5}=8 d(3,3)=2;

Bcost(3,4)=c(2,4)+Bcost(2,2)=5+2=7 d(3,4)=2;

Bcost(3,5)=min{c(1,5)+Bcost(2,1),c(2,5)+Bcost(2,2)}=min{3+5,8+2}=8 d(3,5)=1;

Bcost(4,6)=min{c(3,6)+Bcost(3,3),c(4,6)+Bcost(3,4),c(5,6)+Bcost(3,5)}=min{1+8,6+7,6+8}=9 d(4,6)=3;

Bcost(4,7)=min{c(3,7)+Bcost(3,3),c(4,7)+Bcost(3,4),c(5,7)+Bcost(3,5)}=min{4+8,2+7,6+8}=9 d(4,7)=4;

**Bcost(5,8)**=min{c(6,8)+Bcost(4,6),c(7,8)+Bcost(4,7)}=min{7+9,3+9}=**12**d(5,8)=7;

最短路径长度为Bcost(5,8)=**12**。

最短路径为（0，2，4，7，8）。

**7-2.使用向前递推和向后递推两种方法，求图7-19所示的多段图中从s到t的最短路径及路径长度。**

向后递推的计算过程如上题所示，

最短路径长度为Bcost(5,8)=**12**。

注意，d[i,j]存储最短路径上第i阶段j顶点前的一个顶点号。

从汇点s(即8)开始，向前查找。

取d(5,**8**)(即**7)**，即8的前一个顶点事7，取d(4,**7**)(即**4)**；取d(3,**4**)(即**2)**，取d(2,**2**)(即**0)，**到达源点s(即0)。

故从s到t的最短路径为 (d(2,2)=**0**, d(3,4)=**2**, d(4,7)=**4**, d(5,8)=**7, 8**)，即（0，2，4，7，8）。

向前递推过程如下：

cost(5,8)=0 d(5,8)=-1

cost(4,7)=3 d(4,7)=8, cost(4,6)=7 d(4,6)=8,

cost(3,5)=min{6+cost(4,6),2+cost(4,7)}=5 d(3,5)=7,

cost(3,4)=min{6+cost(4,6),2+cost(4,7)}=5 d(3,4)=7,

cost(3,3)=min{1+cost(4,6),4+cost(4,7)}=7 d(3,3)=7,

cost(2,2)=min{6+cost(3,3),8+cost(3,5),5+cost(3,4)}=10 d(2,2)=4,

cost(2,1)=min{3+cost(3,3),3+cost(3,5)}=8 d(2,1)=5,

**cost(1,0)**=min{5+cost(2,1),2+cost(2,2)}=**12** d(1,0)=2,

最短路径长度为**cost(1,0)**=**12**。

从源点s(即0)开始，向后查找。

取d(1,0)(即2)，即0的下一个顶点是2，取d(2,2)(即4); 取d(3,4)(即7)，取d(4,7)(即8)，到达汇点t(即8)，所以最短路径是**(0, 2, 4, 7, 8)**。

**7-3. 设多段图采用邻接矩阵存储，编写多段图问题的向后递推动态规划算法。**

template<class T>

T MGraph<T>::FMultiGraph(int k, int \*p) //多段图的向后递推算法

{//采用程序6-8的邻接矩阵存储图G

T c,\*Bcost=new T[n];

int q, \*d=new T[n]; //d[i]用于记录最短路径上i的前一个顶点

Bcost[0]=0, d[0]=-1; //设置向后递推的初值

for (int j=1; j<n; j++) //按1,2 …,n-1的次序计算cost和d

{ int min=65535; //按式（7-1）计算最小值为cost[j]

**for(inti=0; i<n; i++)** //如果使用**for(inti=n-1; i>=0; i--)**语句，对图7-1的数据，可以得到

{ if(i!=j&&a[i][j]!=noedge) //长度也是16的另一条最短路径0-2-5-9-11。

{ int v=i;

if(a[i][j]+Bcost[i]<min)

{ min=a[i][j]+Bcost[v];

q=v;

}

}

}

Bcost[j]=min; d[j]=q; //q是j在最短子路径上的后继结点

}

p[0]=0; p[k-1]=n-1; c=Bcost[n-1]; //p[0]和p[n-1]是源点和汇点

for(j=k-2; j>=1; j--) p[j]=d[p[j+1]]; //p[i]是最短路径上第i阶段的结点

delete []Bcost; delete []d;

return c;

}

**7-7 动态规划法可用于计算二项式的系数：。试设计一个动态规划算法计算二项式系数。分析算法的时间复杂度和空间复杂度。将这一算法与其它计算二项式系数的算法进行比较。（不做）**

**补充题：编写函数intcombinat(int m, int n)，用动态规划法求解组合数值。递推公式如下：**

**题解：**

intcombinat(int m, int n)

{ int i,j, \*\*C;

C=new int\*[m-n+1+1];

for(i=0; i<m-n+1+1; i++)

C[i]=new int[n+1];

if(n==0) return 1;

else

{ for(j=1; j<=n;j++)

{ C[1][j]=1;

for(i=2; i<=m-n+1; i++)

if(j==1) C[i][j]=i;

else C[i][j]=C[i-1][j]+C[i][j-1];

}

return C[m-n+1][n];

}

}

**7-9. 给定字符串A=“xzyzzyx”，B=“zxyyzxz”，使用LCS算法求最长公共子串，并给出一个最长公共子串。**

char A[8]={‘0’,’x’,’z’,’y’,’z’,’z’,’y’,’x’ }

B[8]={‘0’,’z’,’x’,’y’,’y’,’z’,’x’,’z’}



(a) c[i][j] （b）s[i][j]

所以，最长公共字串为 (x,y,z,z)。

**7-11. 写一个类似于LCS的算法，它在O(*m*+*n*)时间内构造最长公共子序列。该算法仅使用数组c，而不使用数组s，从而节省了存储空间。**

void LCS::CLCS ( int i,int j )

{ if ( i = = 0 || j = = 0) return;

if (**x[i]==y[j]**)

{ CLCS ( i-1,j-1);

Cout<<a[i];

}

else if ( **c[i-1][j]>=c[i][j-1]**) CLCS (i-1,j);

else CLCS (i,j-1);

}

**7-12. 如果只计算最长公共子序列的长度，而无须构造最优解，则只需保存两行元素就可以计算最长公共子序列的长度。请改写程序7-9得到改进算法，使算法的空间复杂度为Θ(min(*m*,*n*))。**

LCS::LCS(intnx, intny, char \*a, char \*b) //修改的构造函数

{ m=nx; n=ny;

x=a; y=b;

c=new int\*[2];

for(inti=0; i<2; i++) c[i]=new int [n+1];

}

int LCS::LCSLength()

{ c[1][0]=0; //1行0列置0

for(inti=0; i<=n; i++) c[0][i]=0; //0行置0

for(i=1; i<=m; i++)

{ for(int j=1; j<=n; j++)

if(x[i]==y[j]) c[1][j]=c[0][j-1]+1; //x[i]==y[j]时，左上格+1

else if(c[0][j]>=c[1][j-1]) c[1][j]=c[0][j]; //上格≥左格，则取上格

else c[1][j]=c[1][j-1]; //否则取左格

for(j=0; j<=n; j++) c[0][j]=c[1][j]; //直接使用c[0]=c[1]也可以完成将1行的数据赋值给0行。

} //还有孙杨使用%、卞云海使用&，交替计算c[0]和c[1]，

//也可以省略将1行的数据赋值给0行。

return c[1][n]; //或return c[0][n]也行

}

**7-15. 设有0/1背包实例和*M*=32。试计算*Si*，0≤i<4，给出最优解值和最优解（要有计算过程）。另使用启发式方法再计算一次。(题意要求用教材P165的计算步骤计算，由于没有讲，故用P162的步骤计算)**

题解：, 在*S*-1上加上，得,

合并: ，得。

在*S*0上加上，得,

合并: ，得。

在*S*1上加上，得，

合并: ，得，其中被支配，故被舍弃。

在*S*2上加上，得,

合并: ，得，其中有三个阶跃点已被舍弃，被支配，被支配，而，由于40>M(=32)，故此三个点被舍弃。

最优解值(最大收益)为15（注意取得最大收益15时，背包装了10+15+6=31，即背包没有装满，还有剩余量1）。

确定最优解：从开始，看它属于还是*S*2。由于，故。

(不用减去)，看属于还是*S*1。由于，故。

减去，得，看它属于还是*S*0。由于，故。

减去，得，看它属于还是*S*-1。由于，故。

因此最优解(*x*0,*x*1,*x*2,*x*3)=(1, 1, 1, 0)。

**7-16 设有0/1背包问题，所有*wi*、*pi*和*M*均为正整数。试设计一个动态规划算法求解这一问题，使得算法的时间复杂度为Θ(*nM*)。**

**提示：**

**一、求最大收益**

可以定义一个二维数组*m*[*n*][*M*]，称之为状态数组(记录当前最大收益)，共有(*n*+1)×(*M*+1)个元素，每个元素代表一个状态，m[i][j]表示前i个物品放入容量为 j 的背包所能获得的最大价值。状态数组m[i][j]的递推公式如下：

说明：1.首先置0列元素和0行元素为0；

2.然后从下标1行开始，从左至右计算的值，再从左至右计算下标2行的值，以此类推；

3. 时，取上格元素；

4. 时，取（上格元素）与（上格元素左边第个元素+）之间较大的一个；

5. 当计算完成后，右下角元素值就是最大收益。

**例子：**假设有*n*=6个物品，收益数组*p*=(8, 10, 6, 3, 7, 2)，重量数组*w* =(4, 6, 2, 2, 5, 1)，求背包容量M = 12时对应的数组。

该算法的时间主要是花在计算上，因此其时间复杂度是Θ(*nM*)。

**注意**：应用上述递推式时，*p*和*w*数组的前面要放置一个0，即*p*=(0, 8, 10, 6, 3, 7, 2)，*w* =(0, 4, 6, 2, 2, 5, 1)。

根据上述的*p*和*w*计算出的状态数组的值如下：

*w p i*\*j* 0 1 2 3 4 5 6 7 8 9 10 11 12

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

4 8 1 0 0 0 0 8 8 8 8 8 8 8 8 8

**6 10** 2 0 [0 0 0 8 8] 10 10 10 10 18 18 18

2 6 3 0 0 6 6 8 8 14 14 16 16 18 18 24

2 3 4 0 0 6 6 9 9 14 14 17 17 19 19 24

5 7 5 0 0 6 6 9 9 14 14 17 17 19 21 24

1 2 6 0 2 6 8 9 11 14 16 17 19 19 21 **24**

例如，计算*i*=2这一行的元素时，由于*w*[2]=6，且*w*[2]>j时取上格元素，故j=1~5都是取上格元素的值，见上图黑色底纹部分的元素（即方括号中的元素）。当计算*j*=6~12的*m*的值时，由于满足，所以其后的*m*值都是取上格元素与上格元素左边第*w*[2]=6个元素的值加的值中较大的一个值。比如当计算的元素时，就是取其上格值8与8的左边第6个元素0加10中较大的，即8与10较大的一个值10，即；计算时，也是取其上格元素8与上格元素8左边第6个元素8+10中较大的一个，即。

最大收益为。

由于算法时间主要用在计算状态数组*m*[*n*][*M*]上，故其时间复杂度是Θ(*nM*)。

**二、求最优解**

在求得状态数组后，可以从右下角元素开始，回溯找最优解x[]。

从*i*=*n*，*j*=*M*开始回溯，按照下列确定*x*[*i*]的值：

直到*i*=0结束。

该算法的时间主要就是回溯的时间，可以看出，主要是下标*i*的变化，*n*~0，因此该算法的时间复杂度为Θ(*n*+*M*)。

*w p i*\*j* 0 1 2 3 4 5 6 7 8 9 10 11 12

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

4 8 1 0 0 0 0 8 8 8 8 8 8 8 8 8

6 10 2 0 0 0 0 8 8 10 10 10 10 18 18 18

2 6 3 0 0 6 6 8 8 14 14 16 16 18 18 24

2 3 4 0 0 6 6 9 9 14 14 17 17 19 19 24

5 7 5 0 0 6 6 9 9 14 14 17 17 19 21 24

1 2 6 0 2 6 8 9 11 14 16 17 19 19 21 **24**

从开始回溯。由于，故，回溯到；

从开始回溯。由于，故，回溯到；

从开始回溯。由于，故，回溯到；

从开始回溯。由于，而是，故，回溯到；

从开始回溯。由于，而是，故，回溯到；

从开始回溯。由于，而是，故，回溯到；此时，算法结束。红色箭头线给出了回溯的路径。

因此最优解(*x*1,*x*2,*x*3,*x*4,*x*5,*x*6)=(1, 1, 1, 0, 0, 0)。

该算法的时间复杂度是Θ(*n*+*M*)。

**三、相关程序**

1.计算状态矩阵m

template<class T>

T Knapsack<T>::Cal\_m() //计算状态矩阵m，返回最大收益

{ inti,j;

for(i=0; i<=n; i++) m[i][0]=0; //把m的0列元素置0

for(j=0; j<=M; j++) m[0][j]=0; //把m的0行元素置0

for(i=1; i<=n; i++)

{ for(j=1; j<=M; j++)

if(w[i]>j) m[i][j]=m[i-1][j]; //取上格元素值

else m[i][j]=m[i-1][j]>=m[i-1][j-w[i]]+p[i]?m[i-1][j]:m[i-1][j-w[i]]+p[i];

} //取max**(**上格元素m[i][j], 上格元素左边第w[i]个元素值+p[i]**)**

return m[n][M];

}

2.求最优解x[]

template<class T>

void Knapsack<T>::Traceback1()

{ Traceback1(n, M);

}

template<class T>

void Knapsack<T>::Traceback1(inti,int j) //递归法求最优解x[]

{ if(i==0) return;

if(m[i][j]==m[i-1][j]) //若m[i][j]等于上格值，

{ x[i]=0; //则第i件物品未装入，置为0

Traceback1(i-1,j); //回溯到上格，继续递归

}

else if(j-w[i]>=0&&m[i][j]==m[i-1][j-w[i]]+p[i]) //若m[i][j]等于上格的左边第w[i]个元素的值，

{ x[i]=1; //则第i件物品已装入，置为1

Traceback1(i-1,j-w[i]); //回溯到装包之前的位置，即回溯到上格的左边第w[i]个元素的位置，继续递归

}

}

**第八章**

**8-1简述下列名词：状态空间、显示约束、隐式约束、问题状态、解状态、答案状态、活结点、扩展结点、约束函数、剪枝函数。**

状态空间：描述问题的各种可能的情况，一种情况对呀状态空间的一个状态。

显示约束：用于规定每个*xi*取值的约束条件称为显示约束。

隐式约束：用于判定一个候选解是否为可行解的条件。

问题状态：在状态空间树中的每个节点称为一个问题状态。

解状态：如果从根到树中某个状态的路径代表一个作为候选解的元组，则该状态为解状态。

答案状态：如果从根到树中某个状态的路径代表一个作为可行解的元组，则该状态为解状态。

活结点：回溯法从开始结点出发，以深度优先的方式搜索整个解空间，这个开始结点就成为一个活结点。未检测的结点称为活结点。

扩展结点：算法从*x*出发，访问*x*的摸个后继结点*y*，则*x*被称为扩展结点。

约束函数：一个约束函数是关于部分向量的函数*Bk*(*x*0,*x*1.....*xk*),它被定义为：如果可以判定Y的子树上不含任何答案状态，则*Bk*(*x*0,*x*1.....*xk*)为false,否则为true。

剪枝函数：约束函数和限界函数的目的相同，都是为了剪去不必要搜索的子树，减少问题求解所需实际生成的状态节点数，它们统称为剪枝函数。

**8-2程序8-4求*n*-皇后问题的全部可行解，请修改这一程序，使之在求得第一个可行解后算法终止。**

下面的递归函数NQueens可以输出N(可以指定N=1)个解：

int count=0; int N=1;

voidNqueens(int k,int n,int \*x)

{ for(inti=0;i<n&&count<N;i++)

if(place(k,i,x))

{ x[k]=i;

if(k==n-1)

{ for(i=0;i<n;i++)cout<<x[i];

cout<<endl;

count++;

}

else Nqueens(k+1,n,x)

}

}

voidNqueens(int n,int \*x)

{ Nqueens(0,n,x);

}

下面的递归函数NQueens只输出一个解：

voidNQueens(int k, int n, int \*x, bool &flag)

{ for(int i=0; i<n&&flag;i++)

{ if(Place(k,i,x))

{ x[k]=i;

if(k==n-1)

{ for(i=0;i<n;i++) cout<<x[i]<<" ";

cout<<endl; flag=false;

}

else NQueens(k+1,n,x,flag);

}//end if(Place)

}//end for

}

voidNqueens(int n,int \*x)

{ bool flag=true;

NQueens(0,n,x,flag);

}

**8-4图8-4所示的两个可行解是对称的。观察表明*n*-皇后问题的解的确存在这种对偶性。修改算法NQeens，令x[0]=0,1,…,，使得只求其中不对称的那些解。**

voidNQueens(int k, int n, int \*x ,int M, int &nn)

{ int LL;

LL=((k==0)||(n%2==1&&k==1&&x[0]==M-1))?M:n; //当k==0时，控制x[0]<M=(N+1)/2.

for(inti=0; i<LL; i++) //当n为奇数、k==1和x[0]=M-1时，控制x[1]<M=(N+1)/2

if(Place(k,i,x))

{ x[k]=i;

if(k==n-1)

{ nn++;

for(i=0; i<n; i++) cout<<setw(3)<<x[i];

cout<<endl;

}

else NQueens(k+1,n,x,M,nn);

}

}

intNQueens(int n,int \*x)

{ int M, nn=0;

M=(n+1)/2; //M确定x[0]的取值范围，即x[0]<M

NQueens(0,n,x,M,nn);

return nn;

}

**8-6设有子集和数问题的实例*W*=(*w*0,*w*1,…,*w*6)=(5,7,10,12,15,18,20)和*M*=35。求W中元素之和等于*M*的所有子集。画出对于这一实例由SumOfSub算法实际生成的那部分状态空间树。**

状态空间树如下：



**图 只包含了可行解的分枝**

*x*0=0

*x*0=1

0,0,87

*x*1=0

*x*1=1

*x*1=0

*x*1=1

0,1,82

0,2,75

0,3,65

10,3,65

12,4,53

5,1,82

7,2,75

5,2,75

12,2,75

*x*2=0

*x*2=1

*x*2=0

*x*2=1

*x*2=0

*x*2=1

*x*2=0

*x*2=1

17,4,53

17,3,65

7,3,65

5,3,65

12,3,65

15,3,65

5,4,53

15,4,53

15,5,38

15,6,20

22,3,65

*x*3=0

*x*3=0

*x*3=0

*x*3=0

*x*3=0

*x*3=1

*x*3=0

*x*3=1

*x*3=1

*x*3=0

7,4,53

19,4,53

10,4,53

0,4,53

12,4,53

17,4,53

*x*4=0

*x*4=0

*x*4=0

*x*4=0

*x*4=0

*x*4=0

*x*4=0

*x*4=0

*x*4=1

*x*4=0

17,5,38

7,5,38

12,5,38

10,5,38

17,5,38

5,5,38

15,5,38

0,5,38

12,5,38

*x*5=0

*x*5=0

*x*5=1

*x*5=1

15,6,20

*x*6=1

*x*6=1

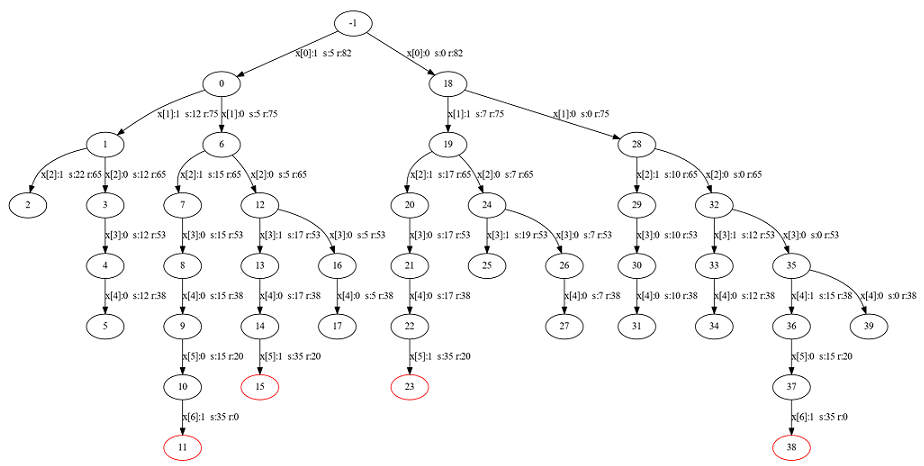
35,6,20

35,6,20

35,7,0

35,7,0

**图 执行程序8-5（改进）得到**



**图 包含了全部生成的结点（该图由丁玉青同学提供，采用python编程，graphviz软件绘图）**

可行解：（1，0，0，1，0，1），（0，1，1，0，0，1），（1，0，1，0，0，0，1），（0，0，0，0，1，0，1）

或（5，12，18）， (7，10，18)， （5，10，20）， （15，20）

**8-7 设*M*=35，对于下列三组数据执行SumOfSub，观察在计算时间上有何显著差别？**

**（1）**

**（2）**

**（3）**

解：对程序8-5进行修改，使之能够生成状态空间树的各结点。运行改进的程序，得可行解为：

（1）（1，0，1，0，0，0，1），（1，0，0，1，0，1），（0，1，1，0，0，1），（0，0，0，0，1，0，1）

（2）（0，1，0，0，1，1）

（3）（1，0，1），（0，1，0，0，1，1），（0，0，0，1，1，0，1）

**各组数据的特点、可行解个数、求得的可行解个数、生成和搜索的结点数见下表：**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **数据组号** | **数据特点** | **可行解数** | **求得的可行解数** | **生成的结点数** | **搜索的结点数** |
| 1 | 有序 | 4 | 4 | 37 | 41 |
| 2 | 逆序 | 4 | 1 | 10 | 11 |
| 3 | 无序 | 4 | 3 | 21 | 24 |

**分析上表可以看出，**由于**SumOfSub**函数要求初始序列有序排列，而后面2组数据无序，故后面2组不能得到全部可行解。而第2组数据只能求出1个可行解，生成10个结点为最少，因此执行时间上第2组时间最短；同理，第3组执行时间长于第2组，第1组执行时间最长。（方月彤同学提供了计算时间）。

**详细分析：参见“程序\习题8-7-子集和数问题的回溯算法-生成状态空间树-固定元组.docx”**

**8-8 设计一个回溯算法，使用可变长度状态空间树求解子集和数问题。**

解：int kk=0; //可行解的个数

struct Node //状态空间树结点结构

{ Node(int num, Node \*par, int jj)

{ NodeNum=num; parent=par; j=jj;

}

int NodeNum; //本结点编号

Node \*parent; //指向双亲结点的指针

int j; //增加的元素号

};

void SumOfSub(float s,int k,float r,int\* x,float m,float\* w,Node \*E, int n, int &num) //递归函数

{ Node \*p; int jj;

for(jj=E->j+1; jj<n; jj++)

{ if(s+w[jj]==m) //到达答案结点，输出

{ kk++; x[k]=jj; //答案结点实际不用生成，也没有生成

cout<<" 可行解"<<char(64+kk)<<"：(";

for (int j=0; j<k; j++) cout<<x[j]<<','; //输出一个解

cout<<x[j]<<')'<<endl;

}

else if( (s+w[jj]+w[jj+1]<=m) && ((s+r-w[jj]>=m)&&(s+w[jj+1]<=m)) )

{ x[k]=jj;

p=new Node(++num, E, jj); //生成一个孩子

SumOfSub(s+w[jj],k+1,r-w[jj],x,m,w,p,n,num); //搜索孩子

}

}//end for

}

void SumOfSub(int\* x, int n, float m, float\* w, int &num)

{ Node \*E;

float r=0;

for(int i=0; i<n; i++) r=r+w[i]; //计算r

E=new Node(++num, NULL, -1);

if(r>=m && w[0]<=m) SumOfSub(0,0,r,x,m,w,E,n,num);

}

**注意：**当n=6时，将最后一个元素后面的元素置成大于最后一个元素的数，防止jj=n-1时，取w[jj+1]超界取到很小的数而不能剪枝。

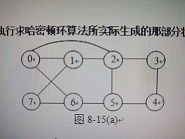
float w0[]={5,10,12,13,15,18,INF};

**8-9使用大小分别为*n*=2, 3, 4, 5, 6, 7的完全图作为输入实例执行*m*-着色算法mColoring。令*m*=*n*和*m*=*n*/2为颜色数。请对*n*和*m*的每个值列表表示算法所需的计算时间。**

总的执行时间=n\*(mn+1-m)/(m-1)

|  |  |  |
| --- | --- | --- |
| n | m=n | m=n/2  注意：该表是在n/2向上取整时得到的结果；如果向下取整，则结果有不同，有可能无法着色。 |
| 2 | 12 | 无法着色 |
| 3 | 117 | 21.38 |
| 4 | 1360 | 120.00 |
| 5 | 19525 | 805.47 |
| 6 | 335916 | 6552.00 |
| 7 | 6725593 | 63042.71 |

**8-11 请画出对图8-15(a)所示的图执行求哈密顿环算法所实际生成的那部分状态空间树。**

 图8-15(a)

7

1

6

0

5

3

4

2

解：

ans

ans

**哈密顿算法实际生成的状态空间树(根据算法得到)-含B结点**

ans

ans

**哈密顿算法实际生成的状态空间树(执行程序得到)-不含B结点**

**ans**

**ans**

B

B

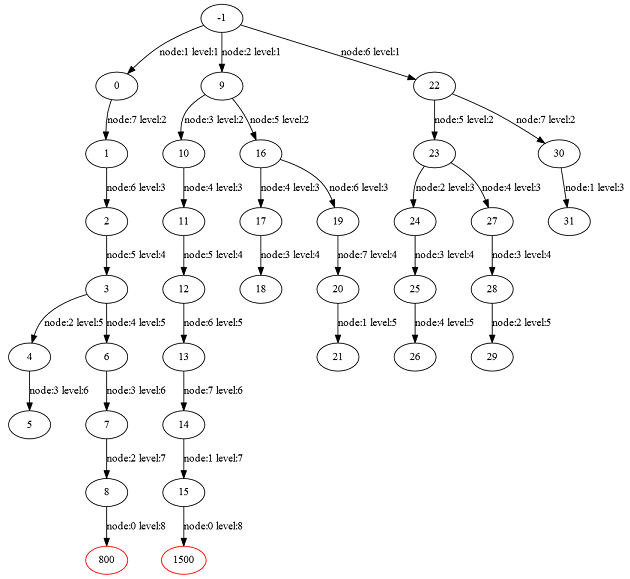
B

B

B

B

**哈密顿算法实际生成的状态空间树(执行程序得到)-含B结点**

****

**8-13 设计一个回溯算法，它使用可变大小元组的状态空间树求解0/1背包问题的最优解。并以例8-4的实例为输入运行你所设计的算法，画出由此生存的那部分状态空间树。**

解：

template<class T>

void Knapsack<T>::BKnapsack(int k, T cp, T cw, T &fp, int \*x, int \*y, Node \*E)

//cp是当前收益，cw是当前背包已装物品重量，k是当前待考察的物品编号,要求p[i]/w[i]≥p[i+1]/w[i+1]

//fp是当前最大收益

{ int j; Node \*Child; T bp; //存放bp上限函数值

for(int jj=E->j+1; jj<n; jj++) //考察孩子

{ bp=Bound(k,cp,cw); //计算上限函数值

if(bp>=fp&&(cw+w[jj]<=m))

{ y[k]=jj;

num++; //增加的语句

Child=new Node(num,E,jj);

BKnapsack(k+1,cp+p[jj],cw+w[jj],fp,x,y,Child);

if(cp+p[jj]>fp)

{ fp=cp+p[jj]; //找到一个收益更高的0/1背包可行解

for(j=0; j<=k; j++) x[j]=y[j]; //x[0], x[1],…, x[k]中保存对应于fp的可行解

Num\_Op=k+1; //记下该组可行解的长度

}

}

else //增加的语句

{ num++; B\_num++;

}

}//end for

}

template<class T>

T Knapsack<T>:: BKnapsack() //一维数组x中返回0/1背包的最优解，函数返回最优解值

{ int \*y; T fp=0;

y=new int[n];

for(int i=0; i<n; i++) y[i]=0;

Node \*root=new Node(num, NULL,-1);

BKnapsack(0,0,0,fp,x,y,root);

return fp;

}

struct Node //状态空间树结点结构

{ Node(int num, Node \*par, int jj)

{ NodeNum=num; parent=par; j=jj;

}

int NodeNum; //本结点编号

Node \*parent; //指向本结点的双亲结点的指针

int j; //增加的本结点的物品号

};

template<class T>

class Knapsack

{ public:

Knapsack(int mSize,T cap,T \*wei,T \*prof);

T BKnapsack();

~Knapsack();

void Optimal\_Solution(); //增加的，输出最优解的函数

protected:

void BKnapsack(int k, T cp, T cw, T &fp, int\*x, int \*y, Node \*E);

T Bound(int k,T cp, T cw); //计算上界函数

T m,\*w,\*p; //要求p[i]/w[i]≥p[i+1]/w[i+1]

int n; //物品个数

int \*x; //增加的存放最优解

int Num\_Op; //增加的可行解实际个数，最后是最优解实际个数

int num; //增加的结点编号

int B\_num; //增加的变量，B结点个数

};

**ans**

4

3

7

6

5

7

6

5

7

4

6

5

6

5

5

*y*4=4

*y*3=3

*y*2=2

*y*1=1

*y*3=4

*y*0=0

**图 求解0/1背包问题的可变大小元组的状态空间树**

**第九章**

**9.1实现带时限的作业排序的函数GenerateAns。**

template<class T>

void JS<T>::GenerateAns(int \*x, int &k)

{ Node \*ans1=ans;

k=0;

while(ans1->j>0)

{ k++;

ans1=ans1->parent;

}

ans1=ans;

for(inti=k-1; i>=0; i--)

{ x[i]=ans1->j;

ans1=ans1->parent;

}

}

**9.2 设有带时限的作业排序问题实例，和**

**。求问题的最优解及对应于最优解的收益损失。画出JSFIFOBB算法实际生成的那部分状态空间树。**

0 1 2 3 4

p={3,8,6,4,5} total=26 U=7

d={1,2,3,4,4}最优解值=19

t={1,1,2,2,1}最优解X={1,2,4} (p,d,t)= (8,2,1) , (6,3,2) , (5,4,1)

**←ans**

*x*0=0

*x*0=3

*x*0=2

*x*0=1

*x*1=1

*x*1=3

*x*1=2

*x*1=3

*x*1=2

*x*1=3

*x*2=3

*x*2=4

*x*2=4

ans

***c*=17**

**^**

***c*=0**

**^**

***c*=3**

**^**

***c*=11**

**^**

***c*=0**

**^**

***c*=8**

**^**

***c*=14**

**^**

***c*=3**

**^**

***c*=9**

**^**

***c*=11**

**^**

***c*=6**

**^**

***c*=10**

**^**

***c*=7**

**^**

**9.3画出当采用LC分枝限界算法求上题中给定的带时限作业排序实例时，实际生成的那部分状态空间树。**

0 1 2 3 4

p={3,8,6,4,5} total=26 U=7

d={1,2,3,4,4} 最优解值=19

t={1,1,2,2,1} 最优解X={1,2,4} (p,d,t)= (8,2,1) , (6,3,2) , (5,4,1)

c(X)<U, 入队 U(X)<U, 修正U

C(X)>=U, 不入队（不生成结点） U(X)>=U, 不修正U

**c(2)=0**

**u(2)=23**

**U=23**

**U=26**

**c(3)=3**

**u(3)=18**

**U=18**

**入队**

**入队**

**入队**

**入队**

**c(4)=11**

**u(4)=18**

**c(5)=17**

**u(5)=22**

**c(6)=21**

**u(6)=21**

**不生成，不入队**

**c(7)=3**

**u(7)=15**

**U=15**

**入队**

**入队**

**入队**

**1. 2出队7**

**c(10)=18**

**u(10)=18**

**不生成**

**c(8)=8**

**u(8)=17**

**c(9)=14**

**u(9)=19**

**2. 7出队**

**3. 3出队**

**←ans**

**4. 14出队**

**c(14)=3**

**u(14)=12**

**U=12**

**c(15)=9**

**u(15)=14**

**入队**

**入队**

**c(18)=7<U=10, 故入队**

**u(18)=7**

**因cost=total-ec.prof=26-19=7<U=10, ans指向结点18，**

**同时置U=7**

**5. 18出队，由于ep.loss(=7)==U(=7)，**

**故算法结束，返回total=U=7.**

**注：忽略了中间过程结点12、13.**

**x0=0**

**x1=1**

**x2=4**

**x1=2**

**1**

**2**

**3**

**4**

图 按照所有孩子结点编号

**1**

**←ans**

**x0=0**

**x1=1**

**x2=4**

**x1=2**

**2**

**3**

图 按照所有孩子结点编号

*x*0=0

*x*0=3

*x*0=2

*x*0=1

*x*1=1

*x*1=3

*x*1=2

*x*1=2

*x*1=3

*x*2=3

*x*2=4

*x*2=4

ans

***c*=17**

**^**

***c*=0**

**^**

***c*=3**

**^**

***c*=11**

**^**

***c*=0**

**^**

***c*=8**

**^**

***c*=14**

**^**

***c*=3**

**^**

***c*=9**

**^**

***c*=6**

**^**

***c*=10**

**^**

***c*=7**

**^**

图 按照所有不受限制的孩子结点编号

**9.4 改写程序9-4，带时限的作业排序程序为LC分枝限界算法。**

template<class T>

T JS<T>::JSLCBB()

{ Node \*E,\*child;

PrioQueue< qNode<T> > q(mSize); //生成一个优先权队列实例q

int num=1; //结点编号num

E=root=new Node(NULL,-1,num); //构造状态空间树的根结点root

qNode<T> ep(0,0,0,-1,root); //ep为扩展结点，根结点的ep.j的值是-1

qNode<T> ec; //ec为活结点（入队的结点）

T U=total; //上界变量U赋初值，total为作业收益和

while(1)

{ T loss=ep.loss, prof=ep.prof; E=ep.ptr; //loss为已造成的损失，prof为已获收益

for(int j=ep.j+1; j<n; j++) //考察所有孩子

{ num++; //按照所有孩子结点编号

if(ep.d+t[j]<=d[j]&&loss<U)

{ //num++; //按照所有不受限制的孩子结点编号

child=new Node(E,j,num); //构造E的孩子结点，构造的孩子结点的ep.j的值是j

ec.prof=prof+p[j]; ec.d=ep.d+t[j];

ec.ptr=child; ec.loss=loss; ec.j=j;

q.Append(ec); //活结点进优先权队列

T cost=total-ec.prof; //计算上界函数值

if(cost<U)

{ U=cost; ans=child; //修改上界变量U，置child为当前的答案结点，

} //当函数返回时，ans指向答案结点

}

loss=loss+p[j];

}

if(!q.IsEmpty())

{ q.Serve(ep); //选择下一个扩展结点

if(ep.loss>=U) return total=U;

}

else return total=U; //若优先权队列为空，则返回最小损失值，隐含着返回ans结点

}

}

**9-5 写一个LIFO分枝限界算法实现带时限的作业排序，假定采取固定大小元组的状态空间树。**

template<class T>

T JS<T>::JSLCBB()

{ Node \*E,\*child;

PrioQueue< qNode<T> > q(mSize); //生成一个优先权队列实例q

int num=1; //结点编号num

cout<<" 结点号 双亲结点号 c^(x) u(x) U ans"<<endl;

E=root=new Node(NULL,-1,num,-1); //构造状态空间树的根结点root

cout<<setw(6)<<root->NodeNum<<setw(9)<<root->j<<setw(21)<<total<<endl;

qNode<T> ep(0,0,0,-1,root); //ep为扩展结点，根结点的ep.j的值是-1

qNode<T> ec; //ec为活结点（入队的结点）

T U=total; //上界变量U赋初值，total为作业收益和

while(1)

{ T loss=ep.loss, prof=ep.prof; E=ep.ptr; k=ep.j+1; //loss为已造成的损失，prof为已获收益

for(int j=1; j>=0; j--) //考察2个孩子

{ //num++; //编号较大，跳号

if(ep.d+j\*t[k]<=d[k]&&loss<U) //前者为约束条件(可行)，loss<U为限界条件(剪枝)。j=1，要加上时间t[k]。

{ num++; //编号较小，按照生成的顺序编号，不跳号。

child=new Node(E,k,num,j); //构造E的孩子结点，构造的孩子结点的ec.j的值是作业号k

ec.prof=prof+j\*p[k]; ec.d=ep.d+j\*t[k]; ec.ptr=child;

ec.loss=loss+(1-j)\*p[k]; ec.j=k;

cout<<setw(6)<<child->NodeNum<<setw(9)<<E->NodeNum<<setw(9)<<ec.loss;

q.Append(ec); //活结点入队

T cost=total-ec.prof; //计算上界函数值

cout<<setw(7)<<cost;

if(cost<U)

{ U=cost; //修改上界变量U，置child为当前的答案结点，

ans=child; //当函数返回时，ans指向答案结点。最后一个ans即是答案结点。

cout<<setw(5)<<U<<setw(4)<<ans->NodeNum;

}

else cout<<setw(5)<<U;

cout<<endl;

}

}

if(!q.IsEmpty())

{ q.Serve(ep); //选择下一个扩展结点

//cout<<"出队结点："<<ep.ptr->NodeNum<<endl;

if(ep.loss>=U) return total=U;

}

else return total=U; //若队列为空，则返回最小损失值，隐含着返回ans结点

}

}

template<class T>

void JS<T>::GenerateAns(int \*x)

{ Node \*ans1=ans;

int i;

for(i=n-1; i>=0; i--)

{ x[i]=ans1->ChildTag; //ans1->ChildTag=1表示是双亲的左孩子，ans1->ChildTag=0表示是双亲的右孩子

ans1=ans1->parent;

}

}

完整程序如下：

#include " PrioQueue.h"

#include "iomanip.h"

#define mSize 30

struct Node //状态空间树结点结构

{ Node(Node \*par, int k, int num, int tag)

{ parent=par; j=k; NodeNum=num; ChildTag=tag;

}

Node \*parent; //指向该结点的双亲结点

int j; //该结点代表的解分量x[i]=j

int ChildTag; //增加的变量，1表示是双亲的左孩子，0表示是双亲的右孩子

int NodeNum; //增加的变量，表示结点的编号，可以不要

};

template<class T>

struct qNode //活结点表中的活结点结构

{ qNode(){}

qNode(T p, T los, int sd, int k, Node \*pt)

{ prof=p; loss=los; d=sd; ptr=pt; j=k;

}

operator T()const {return loss;} //增加的，没有这句，编译出错

T prof, loss; //当前结点X的下界函数c(X)=loss，上界函数u(X)=total-prof

int j,d; //j是作业号, d是迄今为止的时间

Node \*ptr; //指向状态空间树中相应的结点

};

template<class T>

class JS

{ public:

JS(T \*prof, int \*de, int \*time, int size);

T JSLCBB(); //构造状态空间树，求最优解值

void GenerateAns(int \*x); //一维数组x为最优解向量，k中返回x的分量解

private:

T \*p,total; //p为收益数组，total初始为n个作业收益之和

int \*t, \*d, n; //t为作业处理时间数组，d为按非减次序排列的作业时间数组

Node \*ans,\*root; //root指向状态空间树的根，ans指向最优解答案结点

int k; //结点的作业号

};

template<class T>

JS<T>::JS(T \*prof, int \*de, int \*time, int size)

{ n=size;

p=new T[n]; d=new int[n]; t=new int[n];

total=0; k=-1;

for(int i=0; i<n; i++)

{ p[i]=prof[i]; total+=p[i];

d[i]=de[i]; t[i]=time[i];

}

}

template<class T>

T JS<T>::JSLCBB()

{ 省略；

}

template<class T>

void JS<T>::GenerateAns(int \*x)

{ 省略；

}

//要求作业按时限的非减次序排列

void main()

{ float prof1[]={3,8,6,4,5}; int de1[]={1,2,3,4,4}; int time1[]={1,1,2,2,1}; //数据来源习题9-2

float total=0,U;

int i,x[10],n;

cout<<"数据来源习题9-2: "<<endl;

n=5;

JS<float> work1(prof1,de1,time1,n);

U=work1.JSLCBB();

for(i=0; i<n; i++) total+=prof1[i];

cout<<endl;

cout<<" total="<<total<<" U="<<U<<endl;

cout<<" 最优解值="<<total-U<<endl;

cout<<" 最优解X[]=(";

work1.GenerateAns(x);

for(i=0; i<n-1; i++) cout<<setw(2)<<x[i]<<",";

cout<<setw(2)<<x[i]<<" )"<<endl;

for(i=0; i<n; i++)

if(x[i]) cout<<" 作业"<<i<<":("<<setw(2)<<prof1[i]<<","<<setw(2)<<de1[i]<<","<<setw(2)<<time1[i]<<")" <<endl;

cout<<endl;

//数据来源图9-8

cout<<"数据来源图9-8: "<<endl;

n=4; total=0;

float prof2[]={5,3,6,10}; int de2[]={1,1,2,3}; int time2[]={1,1,1,2};

JS<float> work2(prof2,de2,time2,n);

U=work2.JSLCBB();

for(i=0; i<n; i++) total+=prof2[i];

cout<<endl;

cout<<" total="<<total<<" U="<<U<<endl;

cout<<" 最优解值="<<total-U<<endl;

cout<<" 最优解X[]=(";

work2.GenerateAns(x);

for(i=0; i<n-1; i++) cout<<setw(2)<<x[i]<<",";

cout<<setw(2)<<x[i]<<" )"<<endl;

for(i=0; i<n; i++)

if(x[i]) cout<<" 作业"<<i<<":("<<setw(2)<<prof2[i]<<","<<setw(2)<<de2[i]<<","<<setw(2)<<time2[i]<<")" <<endl;

}

*x*4=1

ans

*x*0=1

*x*0=0

*x*1=1

*x*1=0

*x*1=0

*x*1=1

*x*2=1

*x*2=0

*x*2=0

*x*2=1

*x*2=0

*x*3=1

*x*3=0

*x*3=0

*x*3=0

*x*3=0

*x*3=0

*x*3=1

*x*3=1

*x*3=1

*x*4=1

*x*4=1

*x*4=1

*x*4=0

U=23

u=23

c=0

^

U=26

u=26

c=3

^

U=18

u=18

c=3

^

u=26

c=11

^

u=26

c=17

^

u=20

c=11

^

u=26

c=21

^

u=22

c=17

^

U=17

u=17

c=17

^

U=16

u=16

c=11

^

u=20

c=11

^

U=15

u=15

c=15

^

u=16

c=16

^

U=12

u=12

c=3

^

u=18

c=9

^

u=18

c=13

^

u=14

c=9

^

U=9

u=9

c=9

^

U=7

u=7

c=7

^

u=12

c=7

^

u=15

c=0

^

u=23

c=8

^

u=15

c=6

^

u=11

c=6

^

u=15

c=10

^

u=11

c=11

^

**(a) 习题9-2数据的状态空间树**

ans

*x*0=1

*x*0=0

*x*1=1

*x*1=0

*x*1=0

*x*2=0

*x*2=1

*x*2=0

*x*2=1

*x*2=0

*x*2=1

*x*3=0

*x*3=1

*x*3=1

*x*3=1

U=19

u=19

c=0

^

u=21

c=19

^

u=24

c=5

^

u=24

c=8

^

u=24

c=14

^

U=18

u=18

c=8

^

U=14

u=14

c=14

^

U=8

u=8

c=8

^

u=15

c=5

^

u=21

c=11

^

u=15

c=15

^

u=19

c=3

^

u=13

c=3

^

u=19

c=9

^

u=13

c=13

^

U=24

**(b) 图9-8数据的状态空间树**

**图 JSLIFOBB算法的状态空间树(固定长度元组，按生成结点的顺序编号，不跳号)**

**9-6 画出对例9-2的0/1背包实例执行FIFO分枝限界法所实际生产的那部分状态空间树。**

解：数据来源例9-2(已按单位重量非增次序排列)

n=4; p[]={10,10,12,18}; w[]={2,4,6,9}; M=15；

L=38

最优解值=38；最优解x[]={1,1,0,1}，即x[]={0,1,3}

*x*1=1

*x*0=1

*x*2=0

**ans**

*x*3=1

e(15,0,38,0,1)

(UBB,LBB)=(38,32)

*L*=32-ε

e(13,10,38,1,2)

(38,32)

L=32-ε

e(9,20,38,2,4)

(38,32)

*L*=32-ε

e(3,32,38,3,7)

(38,32)

*L*=32-ε

e(0,38,38,4,9)

(38,38)

*L*=38-ε

**←修正*L*=38-ε**

*x*2=1

*x*0=0

e(11,10,32,2,6)

(32,22)

*L*=32-ε

e(15,0,32,1,3)

(32,22)

*L*=32-ε

e(13,10,36,2,5)

(36,22)

*L*=32-ε

*x*1=0

*x*1=1

e(9,20,38,3,8)

(38,38)

*L*=32-ε

**←到达答案结点，修正*L*=38**

结点入队的顺序是：2,3,4,5,6,7,8,9

结点出队列的顺序是：2,3,4,5,6,7,8,9。结点9出队后，队列为空，算法结束。

**图 习题9-6（即例9-2数据）的FIFO分枝限界法状态空间树**

**9-7 假定采用可变大小元组解结构，画出对例9-2的0/1背包实例执行LC分枝限界法所实际生产的那部分状态空间树。**

解：数据来源例9-2(已按单位重量非增次序排列)

n=4; p[]={10,10,12,18}; w[]={2,4,6,9}; M=15；

L=38

结点入队的顺序是：2,3,4,5,6,7

结点出队列的顺序是：2,4,6,7,5。结点5时，其UBB(36)<L(38-ε)，算法结束，返回L=prof=38。

最优解值=38；最优解x[]={0,1,3}

*x*1=1

*x*0=0

ans

*x*2=3

e(15,0,38,-1)

(UBB,LBB)=(38,32)

*L*=32-ε

e(13,10,38,0)

(38,32)

*L*=32-ε

e(9,20,38,1)

(38,32)

*L*=32-ε

e(0,38,38,3)

(38,38)

*L*=38-ε

*x*0=1

e(11,0,32,1)

(32,22)

*L*=32-ε

*x*1=2

e(7,22,36,2)

(36,22)

*L*=32-ε

*x*2=2

e(3,32,38,2)

(38,32)

*L*=32-ε

(28,28)

*L*=32-ε

(30,30)

*L*=32-ε

(18,18)

*L*=32-ε

(38,32)

*L*=38-ε

*x*3=3

此时UBB<*L*，故不生成这3个结点

此时UBB<*L*，故不生成这3个结点

此时UBB<*L*，故不生成这3个结点

*L*最终被修正为38-ε

**图 习题9-7（即例9-2数据）的LC分枝限界法状态空间树**

**9-8 设有0/1背包问题实例n=5，对于以下两种情况：**

**（1）{*p*1, *p*2,…, *p*5}={10,15,6,8,4}，{ *w*1, *w*2,…, *w*5}={4,6,3,4,2}和*m*=12；**

**（2）{ *p*1, *p*2,…, *p*5}={ *w*1, *w*2,…, *w*5}={4,4,5,8,9}和*m*=15。**

**分别求问题的最优解和最优解值，并画出采用LC分枝限界算法实际生成的那部分状态空间树。**

解：

**（1）**

*x*1=1

*x*0=1

*x*2=0

ans

*x*3=0

*x*4=1

e(12,0,29,0,1)

(UBB,LBB)=(29,29)

L=32-ε

e(8,10,29,1,2)

(29,29)

L=29-ε

e(2,25,29,2,3)

(29,29)

L=29-ε

e(2,25,29,3,4)

(29,29)

L=29-ε

e(2,25,29,4,5)

(29,29)

L=29-ε

e(0,29,29,5,6)

(29,29)

L=29-ε

**（2）**

e(15,0,15,0,1)

(UBB,LBB)=(15,13)

*L*=32-ε

e(11,4,15,1,2)

(15,13)

*L*=13-ε

e(7,8,15,2,4)

(15,13)

*L*=13-ε

e(11,4,15,2,5)

(15,9)

*L*=13-ε

e(15,0,15,1,3)

(15,13)

*L*=13-ε

e(11,4,15,2,6)

(15,9)

*L*=13-ε

e(15,0,15,2,7)

(15,13)

*L*=13-ε

e(11,4,15,3,9)

(15,12)

*L*=13-ε

e(6,9,15,3,8)

(15,9)

*L*=13-ε

e(15,0,15,3,11)

(15,8)

*L*=13-ε

e(10,5,15,3,10)

(15,13)

*L*=13-ε

e(11,4,13,4,13)

(13,13)

*L*=13-ε

e(3,12,15,4,12)

(15,12)

*L*=13-ε

e(7,8,15,4,14)

(15,8)

*L*=13-ε

e(7,8,15,3,16)

(15,8)

*L*=13-ε

e(2,13,15,3,15)

(15,13)

*L*=13-ε

e(6,9,15,4,17)

(15,12)

*L*=13-ε

e(2,13,15,4,18)

(15,13)

*L*=13-ε

e(2,13,**13**,5,19)

(13,13)

*L*=13-ε

e(7,8,15,4,20)

(15,8)

*L*=13-ε

e(10,5,14,4,22)

(14,14)

*L*=14-ε

e(2,13,15,4,21)

(15,13)

*L*=13-ε

e(11,4,15,3,24)

(15,12)

*L*=14-ε

e(9,6,15,3,23)

(15,9)

*L*=14-ε

e(9,6,15,4,25)

(15,9)

*L*=14-ε

e(3,12,15,4,26)

(15,12)

*L*=14-ε

ans

**修正*L*=14→**

e(1,14,**14**,5,27)

(14,14)

*L*=14-ε

e(1,14,**14**,5,27)

(14,14)

*L*=14-ε

**图 习题9-8数据的LC分枝限界法状态空间树-固定元组**