第10章
算法优化策略

算法设计策略的比较与选择

最大子段和问题

给定由n个整数(可能为负整数)组成的序列$a_1,a_2,\ldots,a_n$，求该序列形如$\sum_{k=i}^{j} a_k$的子段和的最大值。当所有整数均为负整数时定义其最大子段和为 0 。依此定义，所求的最优值为:

$$\max\left\{0, \max_{1\le i\le j\le n}\sum_{k=i}^{j} a_k\right\}$$

例如:

A=(-2，11，-4，13，-5，-2)

最大子段和为 $\sum_{k=2}^{4} a_k = 20$

简单算法

```
public static int maxSum()
  {
    int n=a.length-1;
    int sum=0;
    for (int i=1;i<=n;i++) {
      int thissum=0;
      for (int j=i;j<=n;j++) {
        thissum+=a[j];
        if (thissum>sum) {
          sum=thissum;
          besti=i;
          bestj=j;
          }
        }
    return sum;
  }
```

注意到 $\sum_{k=i}^{j} a_k = a_j + \sum_{k=i}^{j-1} a_k$，则可将算法中的最后一个for循环省去，避免重复计算只需要$O(n^2)$的计算时间。

分治算法

如果将所给的序列a[1:n]分为长度相等的2段a[1:n/2]和a[n/2+1:n]，分别求出这2段的最大子段和，则a[1

(1)a[1

(2)a[1

(3)a[1:n]的最大子段和为 $\sum_{k=i}^{j} a_k$，且1≤i≤n/2，n/2+1≤j≤n。

对

列

a[n

情

治算法。

复杂度分析

$$T(n)=\begin{cases} O(1) & n \le c \\ 2T(n/2)+O(n) & n > c \end{cases}$$

T(n)=**O(nlogn)**

定理 4.1(主定理) 设 $a \geqslant 1$ 和 $b>1$ 为常数，设 $f(n)$为一函数，$T(n)$由递归式

$$T(n) = aT(n/b) + f(n)$$

对非负整数定义，其中 n/b 指 $\lfloor n/b \rfloor$ 或 $\lceil n/b \rceil$。那么 $T(n)$可能有如下的渐近界：

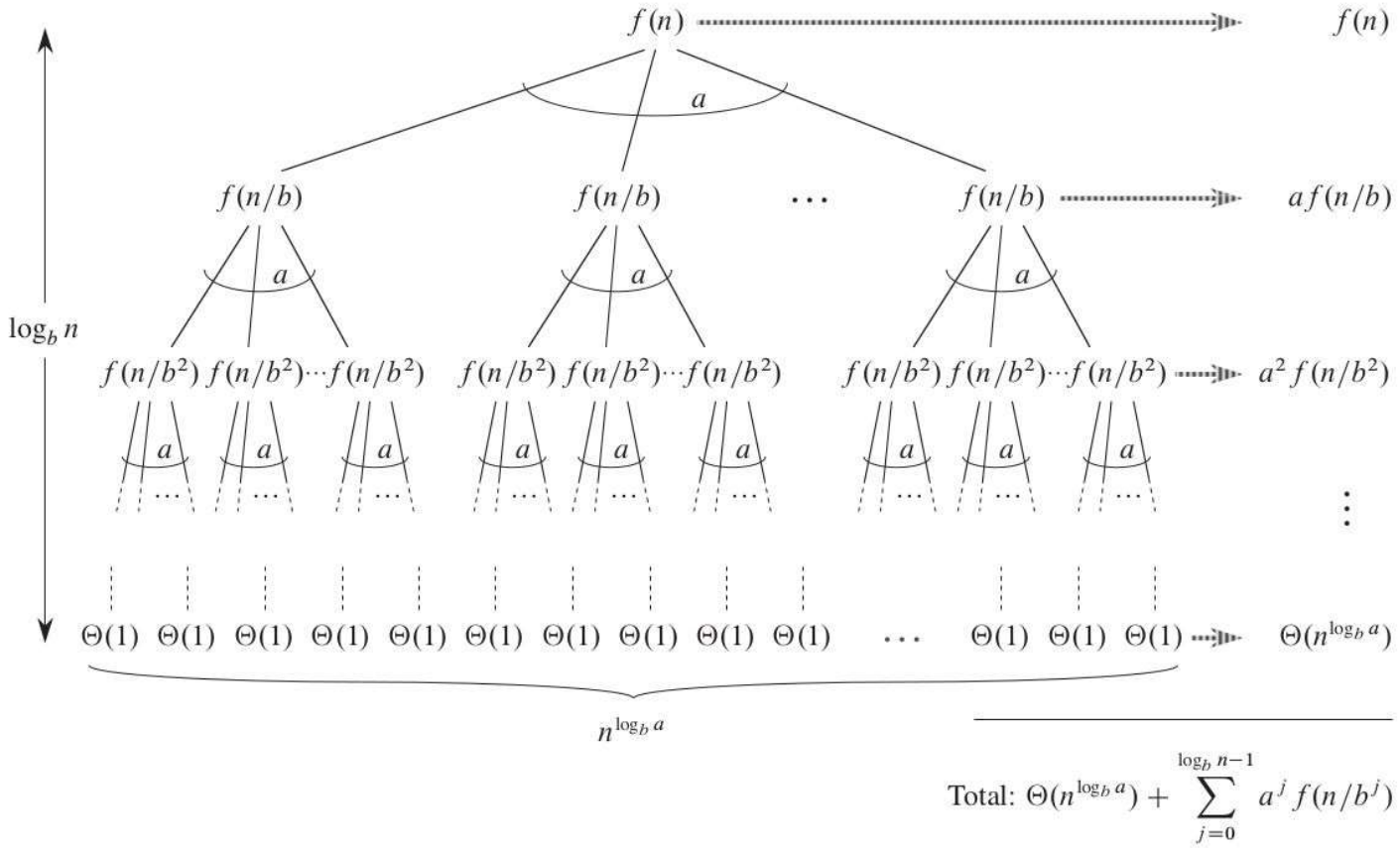
1)若对于某常数 $\varepsilon>0$，有 $f(n)=O(n^{\log_b a-\varepsilon})$，则 $T(n)=\Theta(n^{\log_b a})$；

2)若 $f(n)=\Theta(n^{\log_b a})$，则 $T(n)=\Theta(n^{\log_b a}\lg n)$；

3)若对某常数 $\varepsilon>0$，有 $f(n)=\Omega(n^{\log_b a+\varepsilon})$，且对常数 $c<1$ 与所有足够大的 n，有 $af(n/b)\leqslant cf(n)$，则 $T(n)=\Theta(f(n))$。 ■

分治递归式

$$T(n) = aT(n/b) + f(n)$$



主定理

定理 4.1(主定理) 设 $a \geqslant 1$ 和 $b > 1$ 为常数，设 $f(n)$为一函数，$T(n)$由递归式

$$T(n) = aT(n/b) + f(n)$$

对非负整数定义，其中 n/b 指 $\lfloor n/b \rfloor$ 或 $\lceil n/b \rceil$。那么 $T(n)$可能有如下的渐近界：

1)若对于某常数 $\varepsilon > 0$，有 $f(n) = O(n^{\log_b a - \varepsilon})$，则 $T(n) = \Theta(n^{\log_b a})$；

2)若 $f(n) = \Theta(n^{\log_b a})$，则 $T(n) = \Theta(n^{\log_b a} \lg n)$；

3)若对某常数 $\varepsilon > 0$，有 $f(n) = \Omega(n^{\log_b a + \varepsilon})$，且对常数 $c < 1$ 与所有足够大的 n，有 $af(n/b) \leqslant cf(n)$，则 $T(n) = \Theta(f(n))$。 ■

$(2) f(n) = \Theta(n^{\log_b a})$

设$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(\frac{n}{b^j})$

$f(\frac{n}{b^j}) = \Theta((\frac{n}{b^j})^{\log_b a})$

则$\exists c_1, c_2, n_0, \forall n \geq n_0, c_1(\frac{n}{b^j})^{\log_b a} \leq f(\frac{n}{b^j}) \leq c_2(\frac{n}{b^j})^{\log_b a}. \Rightarrow a^j c_1(\frac{n}{b^j})^{\log_b a} \leq a^j f(\frac{n}{b^j}) \leq a^j c_2(\frac{n}{b^j})^{\log_b a}.$

$c_1 n^{\log_b a}(\frac{a}{b^{\log_b a}})^j \leq a^j f(\frac{n}{b^j}) \leq c_2 n^{\log_b a}(\frac{a}{b^{\log_b a}})^j \Rightarrow c_1 n^{\log_b a}(1)^j \leq a^j f(\frac{n}{b^j}) \leq c_2 n^{\log_b a}(1)^j$

$c_1 n^{\log_b a} \log_b n \leq g(n) \leq c_2 n^{\log_b a} \log_b n \Rightarrow c_1 n^{\log_b a} \lg n \leq g(n) \leq c_2 n^{\log_b a} \lg n$

所以$g(n) = \Theta(n^{\log_b a} \lg n)$

$T(n) = \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \lg n) = \Theta(n^{\log_b a} \lg n)$

$$(1) f(n) = O(n^{(\log_b a)-\varepsilon})$$

$$设 g(n) = \sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})$$

$$f(\frac{n}{b^j}) = O((\frac{n}{b^j})^{(\log_b a)-\varepsilon})$$

$$则 \exists c, n_0, \forall n \geq n_0, f(\frac{n}{b^j}) \leq c(\frac{n}{b^j})^{(\log_b a)-\varepsilon} . \Rightarrow a^j f(\frac{n}{b^j}) \leq c a^j (\frac{n}{b^j})^{(\log_b a)-\varepsilon} .$$

$$a^j f(\frac{n}{b^j}) \leq c n^{(\log_b a)-\varepsilon} (\frac{a}{b^{(\log_b a)-\varepsilon}})^j \Rightarrow a^j f(\frac{n}{b^j}) \leq c n^{(\log_b a)-\varepsilon} (\frac{a b^\varepsilon}{b^{(\log_b a)}})^j \Rightarrow a^j f(\frac{n}{b^j}) \leq c n^{(\log_b a)-\varepsilon} (b^\varepsilon)^j$$

$$g(n) \leq c n^{(\log_b a)-\varepsilon} \sum_{j=0}^{\log_b n-1} (b^\varepsilon)^j \Rightarrow g(n) \leq c n^{(\log_b a)-\varepsilon} \frac{1(1-(b^\varepsilon)^{\log_b n})}{1-b^\varepsilon} \Rightarrow g(n) \leq c n^{(\log_b a)-\varepsilon} \frac{(1-n^\varepsilon)}{1-b^\varepsilon}$$

$$\Rightarrow g(n) \leq c \frac{(n^{(\log_b a)-\varepsilon} - n^{\log_b a})}{1-b^\varepsilon}$$

$$所以 g(n) = O(n^{\log_b a})$$

$$T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a}) = \Theta(n^{\log_b a})$$

$(3) f(n)=\Omega(n^{(\log_b a)+\varepsilon})$存在$c<1$,使得$af(\frac{n}{b})\le cf(n)$

$$g(n)=\sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})$$

$$af(\frac{n}{b})\le cf(n)\Rightarrow a^j f(\frac{n}{b^j})\le c^j f(n)\Rightarrow g(n)\le \sum_{j=0}^{\log_b n-1} c^j f(n)$$

$$\Rightarrow g(n)\le \sum_{j=0}^{\infty} c^j f(n)\Rightarrow g(n)\le \frac{1}{1-c} f(n)$$

因此$g(n)=O(f(n))$

又因为$g(n)=f(n)+af(\frac{n}{b})+a^2 f(\frac{n}{b^2})+.....+a^{\log_b n-1} f(\frac{n}{b^{\log_b n-1}})\ge f(n)$

因此$g(n)=\Omega(f(n))$

所以$g(n)=\Theta(f(n))$

$T(n)=\Theta(n^{\log_b a})+\Theta(f(n))$,又因为$f(n)=\Omega(n^{(\log_b a)+\varepsilon})$,所以$T(n)=\Theta(f(n))$

动态规划算法

记 $b[j]=\max\limits_{1\le i\le j}\{\sum\limits_{k=i}^{j}a[k]\}$ ，1≤ j ≤n，则所求的最大子段和为

$$\max_{1\le i\le j\le n}\sum_{k=i}^{j}a[k]=\max_{1\le j\le n}\max_{1\le i\le j}\sum_{k=i}^{j}a[k]=\max_{1\le j\le n}b[j]$$

当b[j-1]>0时b[j]=b[j-1]+a[j]，否则b[j]=a[j]。由此可得计算b[j]的动态规划递归式: b[j]=max{b[j-1]+a[j]，a[j]}， 1≤ j ≤n

```
public static int maxSum()
  {
    int n=a.length-1;
    int sum=0,
       b=0;
    for (int i=1;i<=n;i++) {
      if (b>0) b+=a[i];
      else b=a[i];
      if (b>sum)sum=b;
      }
    return sum;
  }
```

算法显然需要O(n)计算时间和O(n)空间。

最大子矩阵和问题

给定一个m行n列的整数矩阵a，试求矩阵a的一个子矩阵，使其各元素之和为最大。

记 $$s(i1,i2,j1,j2)=\sum_{i=i1}^{i2}\sum_{j=j1}^{j2}a[i][j]$$

最大子矩阵和问题的最优值为 $$\max_{\substack{1\le i1\le i2\le m\\ 1\le j1\le j2\le n}} s(i1,i2,j1,j2)$$

由于 $$\max_{\substack{1\le i1\le i2\le m\\ 1\le j1\le j2\le n}} s(i1,i2,j1,j2)=\max_{1\le i1\le i2\le m}\{\max_{1\le j1\le j2\le n} s(i1,i2,j1,j2)\}=\max_{1\le i1\le i2\le m} t(i1,i2)$$

其中，$$t(i1,i2)=\max_{1\le j1\le j2\le n} s(i1,i2,j1,j2)=\max_{1\le j1\le j2\le n}\sum_{j=j1}^{j2}\sum_{i=i1}^{i2}a[i][j]$$

设 $b[j]=\sum_{i=i1}^{i2}a[i][j]$，则 $$t(i1,i2)=\max_{1\le j1\le j2\le n}\sum_{j=j1}^{j2}b[j]$$

由于解最大子段和问题的动态规划算法需要时间O(n)，故算法的双重for循环需要计算时间$O(m^2n)$。

最大m子段和问题

给定由n个整数(可能为负整数)组成的序列a1, a2, …, an, 以及一个正整数m，要求确定序列的m个不相交子段，使这m个子段的总和达到最大。

设b(i，j)表示数组a的前j项中i个子段和的最大值，且第i个子段含a[j](1≤ i ≤m，i≤ j ≤n)。则所求的最优值显然为$\max_{m\le j\le n} b(m,j)$

与最大子段和问题类似地，计算b(i，j)的递归式为

$$b(i,j)=\max\{b(i,j-1)+a[j],\ \max_{i-1\le t<j} b(i-1,t)+a[j]\}\quad (1\le i\le m, i\le j\le n)$$

初始时，b(0，j)=0，(1≤ j ≤n);b(i，0)=0，(1≤ i ≤m)。

优化：注意到在上述算法中，计算b[i][j]时只用到数组b的第i-1行和第i行的值。因而算法中只要存储数组b的当前行，不必存储整个数组。另一方面，b(i-1，t)的值可以在计算第i-1行时预先计算并保存起来。计算第i行的值时不必重新计算，节省了计算时间和空间。

最长单调递增子序列（LIS）

问题： 在数组a[0..n-1]中存放有n个数，找出其中最长的单调递增子序列，如a[]={10, 4, 8, 5, 3, 6, 9, 7, 12}，其最长单调递增子序列为{4, 5, 6, 7, 12}。

问题分析： 用数组L[0..n-1]记录以a[i]为结尾元素的最长单调递增子序列的长度，则数组a的最长单调递增子序列的长度为max{L[i]|0≤i<n}。

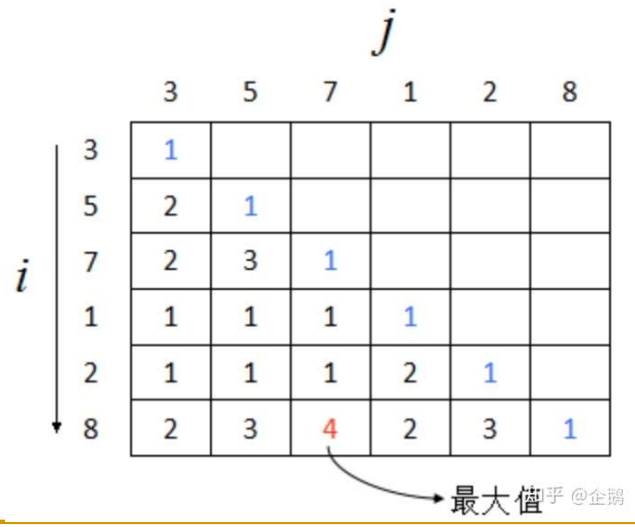
可以递归地定义：

对于j<i<n且a[i]>a[j], 有L[i]=max{L[j]+1,L[i]}

容易证明，L[i]满足最优子结构性质。

最长单调递增子序列

- 假设数组A={3，5，7，1，2，8}
  - 初始化n=6的数组L，L={1,1,1,1,1,1}
  - 遍历i，计算L[i]
  - 对于每个i，遍历j(0<j<i)
  - 如果a[i]>a[j]，则L[i]=max{L[j]+1,L[i]}



最长单调递增子序列

设计算法如下：

```
def get_lis_length(a)
{
 lis = np.ones(n)
  for (i=1;i<n; i++){
     for (j=0;j<i;j++)
         {
    if (a[i]>a[j]) lis[i]=max(lis[j]+1,lis[i]);
          }
                        }
return int(lis.max());
}
```

分析算法可知，算法的时间复杂度为$O(n^2)$。

最长单调递增子序列

二分查找：
原始数组为A，建立一个辅助数组B，变量end记录B数组末尾元素的下标

遍历A中的所有的元素x=A[i]

如果x>B的末尾元素，则将x追加到B的末尾，end+=1
如果x<B的末尾元素，则利用二分法查找，寻找B中第一个大于x的元素，并用x进行替换，例如x=4，B=[1,3,5,6]变为[1,3,4,6]

遍历结束之后，B的长度则为LIS的长度

最长单调递增子序列

INPUT	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	
初始化	B[0] = A[0] end=0	B : [2]
迭代		
1	将A[1] 放到B里面	因为 A[1] <= B[0], 替换
	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	B : [1]
2	将A[2] 放到B里面	因为 A[2] > B[0], B追加， 修改end
	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	B : [1, 5] end = 1
3	将A[3] 放到B里面	因为 A[3] <= B[1], 替换
	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	B : [1, 3]
4	将A[4] 放到B里面	因为 A[4] > B[1]， B追加， 修改end
	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	B : [1, 3, 6] end=2
5	将A[5] 放到B里面	因为 A[5] <= B[2]， 替换
	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	B : [1, 3, 4] end=2
6	将A[6] 放到B里面	因为 A[6] > B[2]， B追加， 修改end
	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	B : [1, 3, 4, 8] end=3
7	将A[7] 放到B里面	因为 A[7] > B[3]， B追加， 修改end
	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	B : [1, 3, 4, 8, 9] end=4
8	将A[8] 放到B里面	因为 A[8] <= B[4], 替换
	A： [2, 1, 5, 3, 6, 4, 8, 9, 7]	B : [1, 3, 4, 7, 9]
OUTPUT	end +1 = 5	

最长单调递增子序列

```
def get_lis_length(arr):
    """
    二分法获取最长递增子序列长度
    :param arr:
    :return:
    """
    temp = [arr[0]]
    end = 0

    for i in range(1, len(arr)):
        if arr[i] > temp[end]:
            end += 1
            temp.append(arr[i])
        else :
            pos = binary_search(temp,0, len(temp), arr[i])
            temp[pos] = arr[i]
    return end + 1
```

```
def binary_search(arr, start, end, value):
    l = start
    r = end-1
    while l <= r:
        m = (l + r) //2
        if arr[m] == value:
            return m
        elif arr[m] < value:
            l = m + 1
        else:
            r = m - 1
    return l
```

动态规划加速原理

货物储运问题

在一个铁路沿线顺序存放着n堆装满货物的集装箱。货物储运公司要将集装箱有次序地集中成一堆。规定每次只能选相邻的2堆集装箱合并成新的一堆，所需的运输费用与新的一堆中集装箱数成正比。 给定各堆的集装箱数，试制定一个运输方案，使总运输费用最少。

设合并a[i:j]，1≤i≤j≤n，所需的最少费用为m[i,j]，则原问题的最优值为m[1，n]。由最优子结构性质可知，

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i<k\le j}\{m[i,k-1]+m[k,j]+\sum\limits_{t=i}^{j}a[t]\} & i<j\end{cases}$$

根据递归式，按通常方法可设计计算m(i,j)的O(n³)动态规划算法

四边形不等式

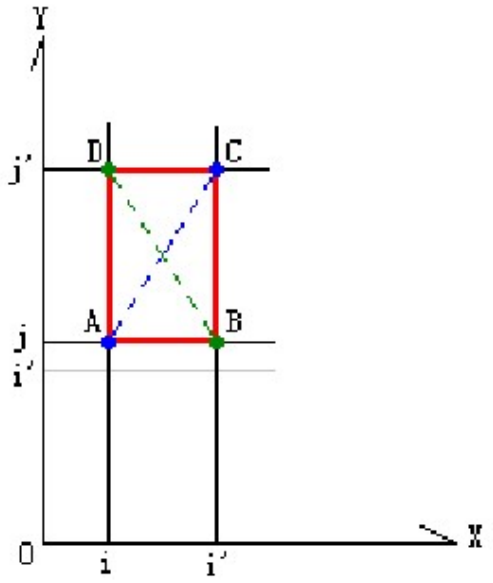
货物储运问题的动态规划递归式是下面更一般的递归计算式的特殊情形。

$$m[i,j]=\begin{cases}0 & i=j\\ w(i,j)+\min\limits_{i<k\le j}\{m[i,k-1]+m[k,j]\} & i<j\end{cases}$$

对于 $i \le i' \le j \le j'$,当函数w(i,j)满足

$$w(i,j)+w(i',j')\le w(i',j)+w(i,j')$$

时称w满足四边形不等式。



当函数w(i,j)满足

$$w(i',j)\le w(i,j')$$

时称W关于区间包含关系单调

对于满足四边形不等式的单调函数w，可推知由递归式定义的函数m(i,j)也满足四边形不等式，即

$$m(i,j)+m(i',j')\le m(i',j)+m(i,j')$$

四边形不等式

- 定义$m_k(i,j)=m(i,k)+m(k+1,j)+w(i,j)$表示分割为k时的m(i,j)值,对于i≤i'≤j≤j'时，记

u = $\operatorname{argmin}_{i\le k<j'} m_k(i,j')$, v = $\operatorname*{argmin}_{i'\le k<j} m_k(i',j)$分别表示m(i,j')和m(i',j)的最优分割点，对四边形的长度j'-i运动归纳法，当i=i'或是j=j'时,m的四边形不等式显成立，于是长度≤1时，m的四边形不等式成立

先考虑i'=j的情形

四边形不等式

- 对于i≤i'=j≤j'，若u<j,则

$$m(i,j) \le m(i,u) + m(u+1,j) + w(i,j)$$

由归纳法，当长度为j'-$(u+1)$时，四边形不等式成立，即

$$m(u+1,j) + m(i',j') \le m(u+1,j') + m(i',j), \text{于是}$$

$$m(i,j) + m(u+1,j) + m(i',j') \le m(i,u) + m(u+1,j) + w(i,j) + m(u+1,j') + m(i',j)$$

$$m(i,j) + m(i',j') \le m(i,u) + m(u+1,j') + m(i',j) + w(i,j)$$

对$w(i,j)$运用区间包含单调性$w(i,j) \le w(i,j')$,得到

$$\begin{aligned} m(i,j) + m(i',j') &\le m(i,u) + m(u+1,j') + w(i,j') + m(i',j) \\ &= m(i,j') + m(i',j) \end{aligned}$$

四边形不等式

- 若u≥j,则

$$m(i',j') \le m(i',u) + m(u+1,j') + w(i',j')$$

由归纳法，当长度为$u-i$时，四边形不等式成立，即

$$m(i,j) + m(i',u) \le m(i,u) + m(i',j),\text{于是}$$

$$m(i,j) + m(i',u) + m(i',j') \le m(i',u) + m(u+1,j') + w(i',j') + m(i,u) + m(i',j)$$

$$m(i,j) + m(i',j') \le m(i,u) + m(u+1,j') + w(i',j') + m(i',j)$$

对$w(i',j')$运用区间包含单调性$w(i',j') \le w(i,j')$,得到

$$m(i,j) + m(i',j') \le m(i,u) + m(u+1,j') + w(i,j') + m(i',j)$$

$$= m(i,j') + m(i',j)$$

四边形不等式

- 对于i≤i’<j≤j’(不需要区间包含单调性)，若u≤v，则i≤u≤v<j,i’≤v<j≤j’,即i≤u<j,i’≤v<j’,因此

$$m(i,j) \le m(i,u) + m(u+1,j) + w(i,j)$$

$$m(i',j') \le m(i',v) + m(v+1,j') + w(i',j')$$

由$u+1 \le v+1 \le j \le j'$,及归纳法，当长度为$j'-(u+1)$时，

四边形不等式成立，即

$$m(u+1,j) + m(v+1,j') \le m(u+1,j') + m(v+1,j),\text{于是}$$

$$\begin{aligned} &m(i,j) + m(i',j') + m(u+1,j) + m(v+1,j') \\ &\le m(i,u) + m(u+1,j) + w(i,j) \\ &+m(i',v) + m(v+1,j') + w(i',j') \\ &+m(u+1,j') + m(v+1,j) \end{aligned}$$

四边形不等式

消去$m(u+1,j)+m(v+1,j')$，得到

$$\begin{aligned} & m(i,j)+m(i',j') \\ \leq\ & m(i,u)+m(u+1,j')+w(i,j) \\ & +m(i',v)+m(v+1,j)+w(i',j') \end{aligned}$$

对$w(i,j)$运用四边形不等式$w(i,j)+w(i',j')\leq w(i,j')+w(i,j')$,得到

$$\begin{aligned} & m(i,j)+m(i',j') \\ \leq\ & m(i,u)+m(u+1,j')+w(i,j') \\ & +m(i',v)+m(v+1,j)+w(i',j) \\ =\ & m(i,j')+m(i',j) \end{aligned}$$

- 若u>v,则i≤v(<u)<j,i'≤(v<)u≤j',因此

$$m(i,j)\leq m(i,v)+m(v+1,j)+w(i,j)$$

$$m(i',j')\leq m(i',u)+m(u+1,j')+w(i',j')$$

四边形不等式

由$i \le i' \le v < u$,及归纳法，当长度为$u - i$时，

$$m(i,v) + m(i',u) \le m(i,u) + m(i',v),\text{于是}$$

$$\begin{aligned} & m(i,j) + m(i',j') \\ & \le m(i,v) + m(v+1,j) + w(i,j) \\ & + m(i',u) + m(u+1,j') + w(i',j') \\ & \le m(i,u) + m(u+1,j') + w(i,j) \\ & + m(i',v) + m(v+1,j) + w(i',j') \end{aligned}$$

对$w(i,j)$运用四边形不等式$w(i,j) + w(i',j') \le w(i,j') + w(i,j')$,得到

$$\begin{aligned} & m(i,j) + m(i',j') \\ & \le m(i,u) + m(u+1,j') + w(i,j') \\ & + m(i',v) + m(v+1,j) + w(i',j) \\ & = m(i,j') + m(i',j) \end{aligned}$$

四边形不等式

- 若m(i,j)满足四边形不等式，记 $s(i,j)=\underset{i\le k<j}{\operatorname{argmin}} m_k(i,j)$ 表示最优分割点，则有

$$s(i,j-1) \le s(i,j) \le s(i+1,j)$$

- 记$u=s(i,j), k_1=s(i,j-1), k_2=s(i+1,j)$
- 若$k_1>u$,则$u+1\le k_1+1\le j-1\le j$,根据m(i,j)四边形不等式有

$$m(u+1,j-1)+m(k_1+1,j) \le m(u+1,j)+m(k_1+1,j-1)$$

四边形不等式

跟据u是$m(i,j)$的最优分割点，得到

$$m(i,j) = m(i,u) + m(u+1,j) \le m(i,k_1) + m(k_1+1,j)$$

将两不等式相加，得到

$$\begin{aligned} & m(u+1,j-1) + m(k_1+1,j) + m(i,u) + m(u+1,j) \\ & \le m(u+1,j) + m(k_1+1,j-1) \\ & \quad + m(i,k_1) + m(k_1+1,j) \end{aligned}$$

即

$$m(i,u) + m(u+1,j-1) \le m(i,k_1) + m(k_1+1,j-1)$$

与k_1是$m(i,j-1)$的最优分割点矛盾，因此$s(i,j-1) \le s(i,j)$

四边形不等式

- 若u>k_2,则i≤i+1≤k_2≤u,由四边形不等式得

$$m(i,k_2)+m(i+1,u)\le m(i,u)+m(i+1,k_2)$$

跟据k_2是$m(i+1,j)$的最优分割点，得到

$$m(i+1,j)=m(i+1,k_2)+m(k_2+1,j)\le m(i+1,u)+m(u+1,j)$$

将两不等式相加，得到

$$\begin{aligned} m(i,k_2)+m(i+1,u)+m(i+1,k_2)+m(k_2+1,j) \\ \le m(i,u)+m(i+1,k_2) \\ +m(i+1,u)+m(u+1,j) \end{aligned}$$

即

$$m(i,k_2)+m(k_2+1,j)\le m(i,u)+m(u+1,j)$$

与u是$m(i,j)$的最优分割点矛盾，因此$s(i,j)\le s(i+1,j)$

四边形不等式

- 因此在计算状态m(i,j)的同时将其最优分割点s(i,j)记录下来，则对于最优分割点k的枚举量降为

$$\min_{i<k\le j}\{m(i,k-1)+m(k,j)\}=\min_{s(i,j-1)\le k\le s(i+1,j)}\{m(i,k-1)+m(k,j)\}$$

- 改进后动态规划算法求解m(1,n)=opt{m(i,j)}所需的时间为

$$\sum_{1\le i<j\le n}\{s(i+1,j)-s(i,j-1)+1\}=\sum_{j=1}^{n}\sum_{i=1}^{j-1}\{s(i+1,j)-s(i,j-1)+1\}$$

$$=\sum_{r=1}^{n-1}\sum_{q=1}^{n-r}s(q+1,q+r)-s(q,q+r-1)+1=\sum_{r=1}^{n-1}n-r+s(n-r,n)-s(1,r)$$

货物储运问题

- 对于货物储运问题，w(i,j)=cnt(j)-cnt(i-1),其中cnt(n)=a[1]+…+a[n]
- 对于i<j,要证明w(i,j)+w(i+1,j+1)<=w(i+1,j)+w(i,j+1)
  - 移项得w(i,j)-w(i+1,j)<=w(i,j+1)-w(i+1,j+1)
  - 带入cnt, w(i,j)-w(i+1,j)=cnt(i)-cnt(i-1)<=w(i,j+1)-w(i+1,j+1)
  - 且满足区间包含单调性

问题的算法特征

贪心策略

- 采用每次合并集装箱数最少的相邻2堆货物的贪心策略，并不能得到最优解。
- 例如：5，3，4，1，3，2，3，4

5，3，4，4，　　2，3，4

5，3，4，4，　　5，　　4

5，7，　　4，　　5，　　4

5，7，　　9，　　4

12，　　9，　　4

12，　　13

25

4+5+7+9+12+13+25=75

贪心策略

5，3，4，1，3，2，3，4

5，3，4，4，　2，3，4

5，3，4，4，　5，　4

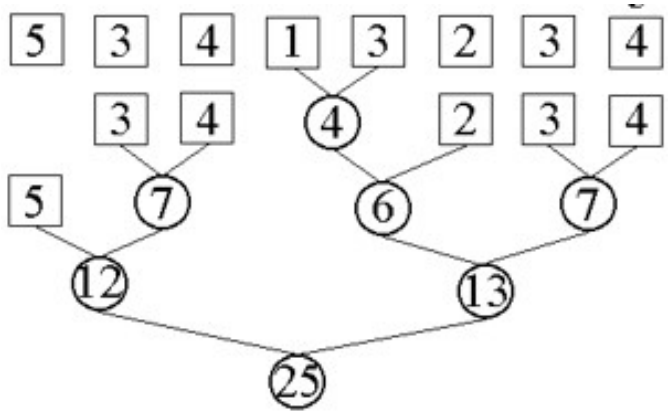
5，7，　4，　5，　4

5，7，　4，　9

5，11，　　9

16，　　9

　　25



4+5+7+9+11+16+25=77

4+6+7+7+12+13+25=74

贪心策略

- 适当放松相邻性约束，引入相容结点对概念。如图，原始结点用方形结点表示，合并生成的结点用圆形结点表示。

- 最小相容结点对a[i]和a[j] 是满足下面条件的结点对：

（1）结点a[i]和a[j] 之间没有方形结点；

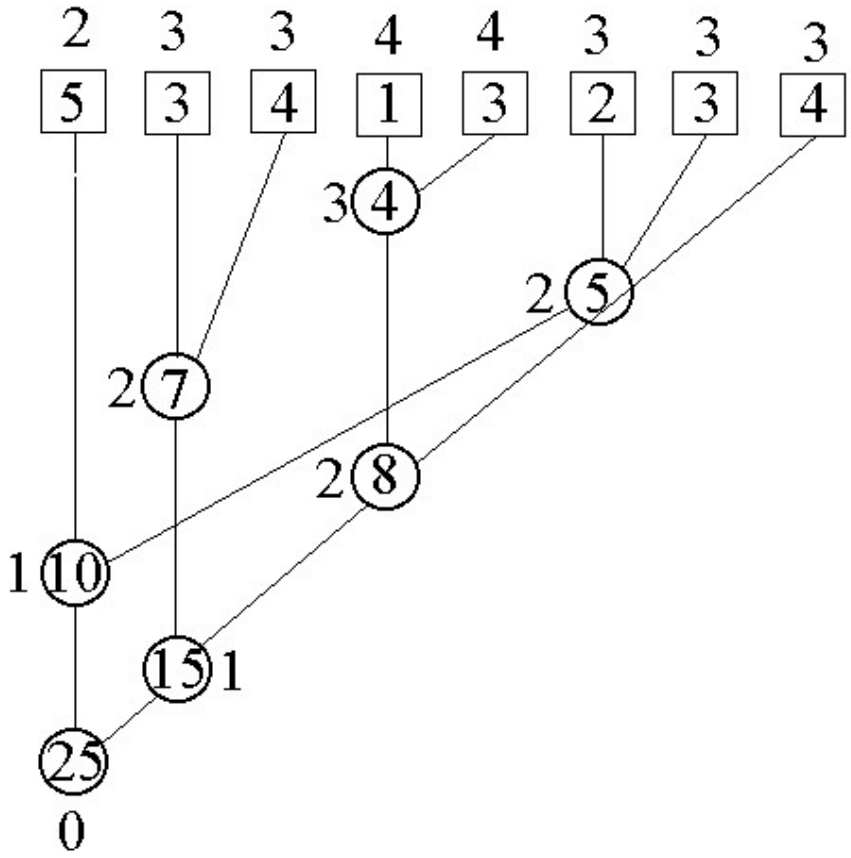
（2）在所有满足条件（1）的结点中a[i]+a[j]的值最小；

（3）在所有满足条件（1）和（2）的结点中下标 i 最小；

（4）在所有满足条件（1）~（3）的结点中下标 j 最小。

贪心策略

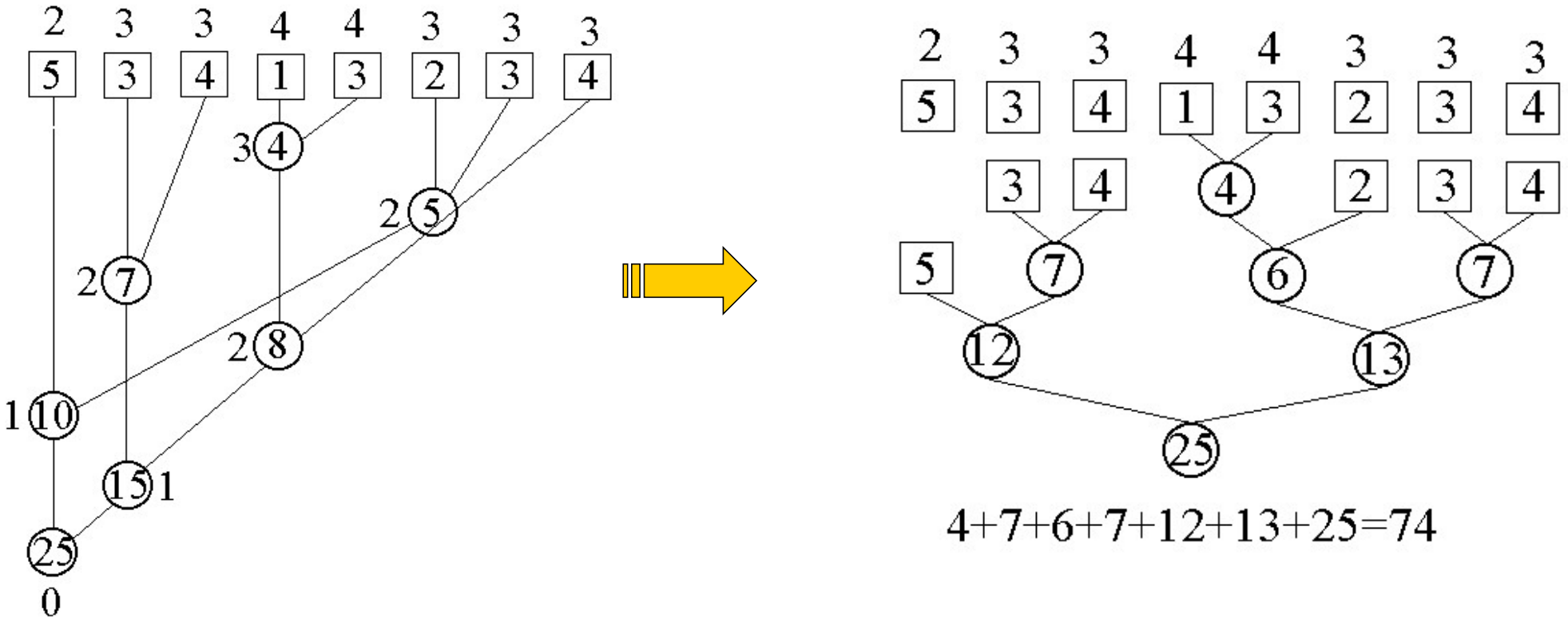
- 相应的最小相容合并树，如图所示。



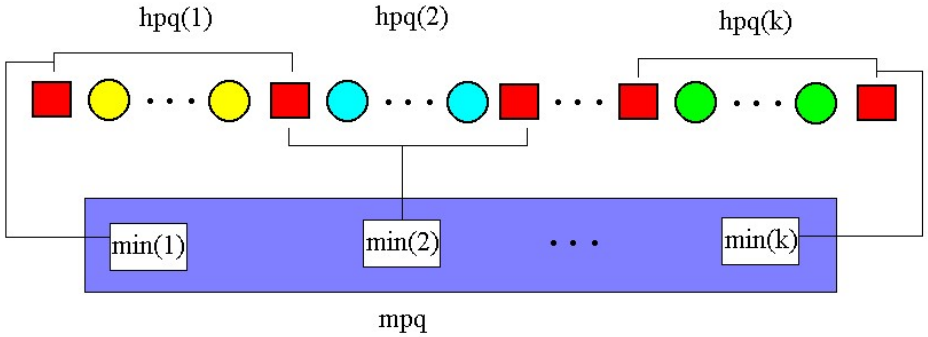
相同层序定理

相同层序定理：存在货物储运问题的最优合并树，其各原始结点在最优合并树中所处的层序与相应的原始结点在相容合并树中所处的层序相同。

根据上述定理，容易从各原始结点在相容合并树中所处的层序构造出相应的最优合并树，如图所示。



算 法



1. **组合阶段**: 将给定的n个数作为方形结点依序从左到右排列，a[1]，a[2]，…，a[n]。反复删除序列中最小相容结点对a[i]和a[j]，(i<j)，并在位置i处插入值为a[i]+a[j]的圆形结点，直至序列中只剩下1个结点。O(nlogn)
2. **标记层序阶段**: 将第一阶段结束后留下的惟一结点标记为第0层结点。然后以与第一阶段相反的组合顺序标记其余结点的层序。O(n)
3. **重组阶段**: 根据标记层序阶段计算出的各结点的层序，按下述规则重组。O(n)

   结点a[i]和a[j]重组为新结点应满足：

   （1）a[i]和a[j]在当前序列中相邻；

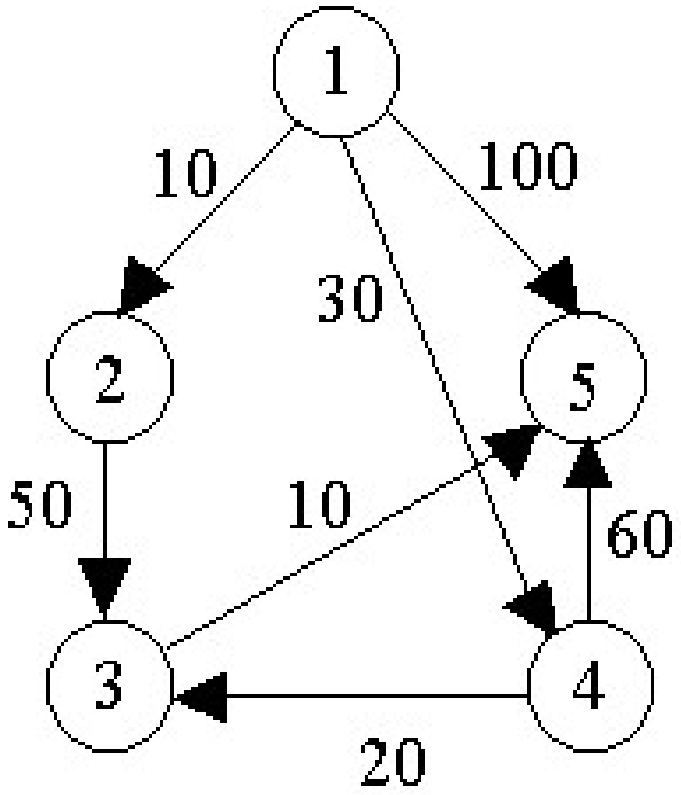
   （2）a[i]和a[j]均为当前序列中最大层序结点；

   （3）在所有满足条件（1）和（2）的结点中，下标 i 最小。

优化数据结构

带权区间最短路问题

例如，对右图中的有向图，应用Dijkstra算法计算从源顶点1到其他顶点间最短路径的过程列在下页的表中。



带权区间最短路问题

Dijkstra算法的迭代过程：

迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	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

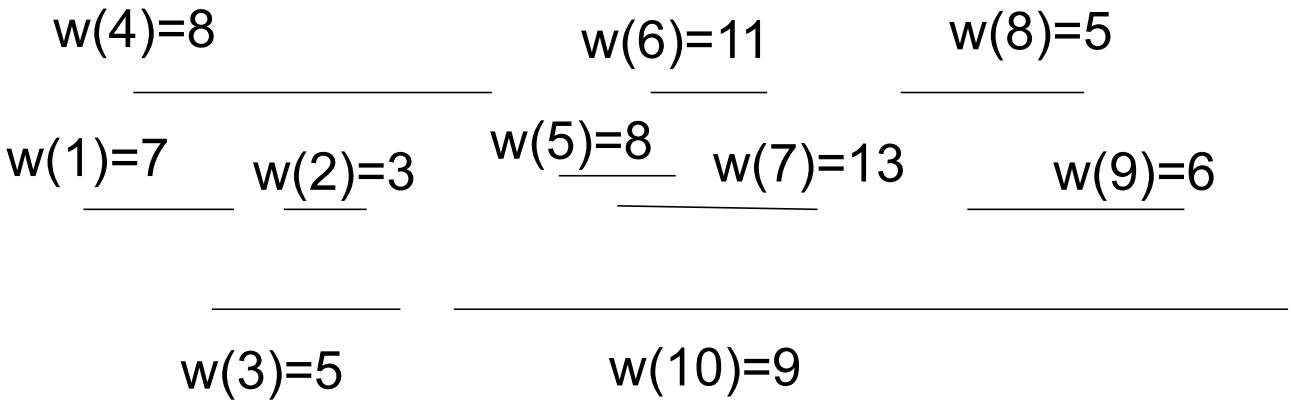
带权区间最短路问题

S是直线上n个带权区间的集合。从区间I∈S到区间J∈S的一条路是S的一个区间序列 J(1)，J(2)，…，J(k)，其中 J(1) = I，J(k) = J，且对所有1≤ i ≤ k-1, J(i)与J(i+1)相交。这条路的长度定义为路上各区间权之和。在所有从I到J的路中，路长最短的路称为从I到J的最短路。**带权区间图的单源最短路问题要求计算从S中一个特定的源区间到S中所有其他区间之间的最短路。**

区间集S(i)的扩展定义为：S(i)∪T，其中T是满足下面条件的另一区间集。T中任意区间I=[a,b]均有b>b(i)。

设区间I(k)(k≤i)是区间集S(i)中的一个区间，1≤ i ≤ n。如果对于S(i)的任意扩展S(i)∪T，当区间J∈T且在S(i)∪T中有从I(1)到J的路时，在S(i)∪T中从I(1)到J的任一最短路都不含区间I(k)，则称区间I(k)是S(i)中的**无效区间**。若S(i)中的区间I(k)不是无效区间则称其为S(i)中的**有效区间**。

带权区间最短路问题



- 区间I(2)是S(4)的无效区间，I(3)是S(3)的有效区间，I(5)是S(5)的无效区间，I(9)是S(10)的无效区间，I(10)是S(10)的有效区间

带权区间最短路问题

性质1： 区间I(k)是S(i)中的有效区间，则对任意k≤j≤i，区间I(k)是S(j)中的有效区间。另一方面，若区间I(k)是S(i)中的无效区间，则对任意j>i，区间I(k)是S(j)中的无效区间。

性质2： 集合S(i)中所有有效区间的并覆盖从a(1)到b(j)的线段，其中b(j)是S(i)的最右有效区间的右端点。

性质3： 区间I(i)是集合S(i)中的有效区间当且仅当在S(i)中有一条从I(1)到I(i)的路。

性质4： 当i>k且dist(i,i)<dist(k,i)时，I(k)是S(i)中的无效区间。

带权区间最短路问题

性质**5**：设I(j(1))，I(j(2))，…，I(j(k))是S(i)中的有效区间，且j(1)<j(2)<…<j(k)≤i，则dist(j(1),i) ≤ dist(j(2),i) ≤ … ≤ dist(j(k),i)。

性质**6**：如果区间I(i)包含区间I(k)（因此i>k），且dist(i,i)<dist(k,i)，则I(k)是S(i)中的无效区间。

性质**7**：当i>k且dist(i,i)<dist(k,i-1)时，I(k)是S(i)中的无效区间。

性质**8**：如果区间I(k)(k>1)不包含S(k-1)中任一有效区间I(j)的右端点b(j)，则对任意i≥k，I(k)是S(i)中的无效区间。

带权区间图的最短路算法

算法**shortestIntervalPaths**

步骤**1**：dist(1,1)←w(1);

步骤**2**：

```
for (i=2;i<=n;i++){
(2.1):
j=min{ k | a(i)<b(k);1≤k<i };
if (j不存在) dist(i,i)←+∞;
else dist(i,i)←dist(j,i-1)+w(i);
(2.2):
for (k<i){
if (dist(i,i)<dist(k,i-1)) dist(k,i)←+∞;
else dist(k,i)←dist(k,i-1);
}
}
```

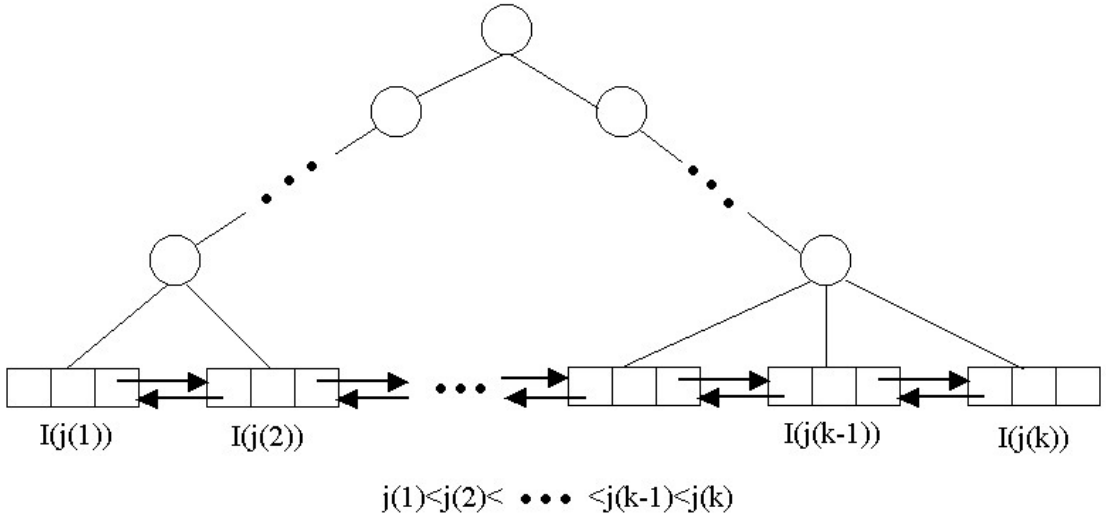
步骤**3**：

```
for (i=2;i<=n;i++){
   if (dist(i,n)=+∞) {
       j=min{ k | (dist(k,n)<+∞)
                  &&(a(i)<b(k)) };
           dist(i,n)=dist(j,n)+w(i);
}
}
```

上述算法的关键是有效地实现步骤(2.1)和(2.2)

实现方案1

用一棵平衡搜索树（2-3树）存储当前区间集S(i)中的有效区间。以区间的右端点的值为序。如图所示。



(2.1)的实现对应于平衡搜索树从根到叶的一条路径上的搜索，在最坏情况下需要时间O(logn)。 (2.2)的实现对应于反复检查并删除平衡搜索树中最右叶结点的前驱结点。在最坏情况下，每删除一个结点需要时间O(logn)。

综上，算法**shortestIntervalPaths**用平衡搜索树的实现方案，在最坏情况下的计算时间复杂性为O(nlogn)。

实现方案2

采用并查集结构。用整数k表示区间I(k),1≤k≤n。初始时每个元素k构成一个单元素集，即集合k是{k}，1≤k≤n。

(1) 每个当前有效区间I(k)在集合k中。

(2) 对每个集合S(i)，设L(S(i))={I(k)|I(k)是S(i)的无效区间，且I(k)与S(i)的任一有效区间均不相交} , L(S(i))中所有区间均位于S(i)的所有有效区间并的右侧。

(3) 用一个栈AS存放当前有效区间I(i(1))，I(i(2))，…，I(i(k))。I(i(k))是栈顶元素。该栈称为当前有效区间栈。

(4) 对于1≤k≤n，记prev(I(k))=min{j|a(k)<b(j)}。对给定的区间序列做一次线性扫描确定prev(I(k))的值。

(5) 对于当前区间集S(i)，用一维数组dist记录dist(j,i)的值。

(6) 用dist[k]=-1标记区间I(k)为无效区间。

优化搜索策略

最短加法链问题

给定一个正整数和一个实数，如何用最少的乘法次数计算出x^n。例如，可以用6次乘法逐步计算x^{23}如下：

$$x, x^2, x^3, x^5, x^{10}, x^{20}, x^{23}$$

可以证明计算最少需要6次乘法。计算的幂序列中各幂次1，2，3，5，10，20，23组成了一个关于整数23的加法链。在一般情况下，计算x^n的幂序列中各幂次组成正整数的一个加法链

$$1 = a_0 < a_1 < a_2 < \cdots < a_r = n$$

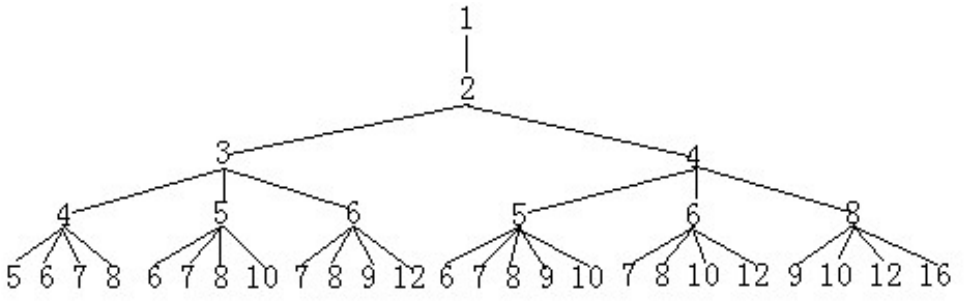
$$a_i = a_j + a_k, k \le j < i,\ i = 1,2,\cdots,r$$

上述最优求幂问题相应于正整数的最短加法链问题，即求n的一个加法链使其长度达到最小。正整数的最短加法链长度记为l(n)。

回溯法

问题的状态空间树如图所示。其中第i层结点a_i的儿子结点$a_{i+1}>a_i$由a_j+a_k, $k\le j\le i$所构成。

```
private static void backtrack(int step)
  {// 解最短加法链问题的标准回溯法
     int i,j,k;
     if (a[step]==n)   // 找到一条加法链
     { if (step<best)  更新最优值
        return;
     }
// 对当前结点a[step]的每一个儿子结点递归搜索
     for (i=step;i>=1;i--)
       if (2*a[i]>a[step])
         for (j=i;j>=1;j--){
           k=a[i]+a[j];
           a[step+1]=k;
           if ((k>a[step])&&(k<=n))
                backtrack(step+1);
           }
  }
```



由于加法链问题的状态空间树的每一个第k层结点至少有k+1个儿子结点，因此从根结点到第k层的任一结点的路径数至少是k!。用标准的回溯法只能对较小的构造出最短加法链。

迭代搜索法

- 深度优先搜索: 算法所搜索到的第一个加法链不一定是最短加法链。
- 广度优先搜索: 算法找到的第一个加法链就是最短加法链，但这种方法的空间开销太大。
- **迭代搜索算法**: 既能保证算法找到的第一个加法链就是最短加法链，又不需要太大的空间开销。其基本思想是控制回溯法的搜索深度d，从d=1开始搜索，每次搜索后使d增1，加深搜索深度，直到找到一条加法链为止。

迭代搜索法

```
private static void iterativeDeepening()
 {// 逐步深化的迭代搜索算法
    best=n+1;
    found=false;
    lb=2;  // 初始迭代搜索深度
    while (!found){
      a[1]=1;
      backtrack(1);
      lb++;  // 加深搜索深度
      }
 }
```

对于正整数，记λ(n)=⌊logn⌋，v(n)=n的2进制表示中1的个数。迄今为止所知道的l(n)的最好下界是l(n)≥lb(n)= λ(n)+⌈logv(n)⌉。利用这个下界，可以从深度lb(n)开始搜索，大大加快了算法的搜索进程。

剪枝函数

- 设a_i和a_j是加法链中的两个元素，且$a_i>2^m a_j$。由于加倍是加法链中元素增大的最快的方式，即$a_i \le 2a_{i-1}$，所以从a_j到a_i至少需要m+1步。如果预期在状态空间树T的第d层找到关于n的一条加法链，则以状态空间树第i层结点a_i为根的子树中，可在第d层找到一条加法链的必要条件是$2^{d-i}a_i \ge n$。
- 当n是奇数时，加法链最后一个元素$a_r=a_j+a_k$，$k \le j$，是奇数，于是k<j(否则a_r为偶数),由r最小可知,j=r-1.因此, $a_r=a_{r-1}+a_k$, k<r-1.如果在第d层找到最短加法链，即r=d，则$a_{d-1}+a_{d-2} \ge n$，由于$a_{d-1} \le 2a_{d-2}$,所以$3a_{d-2} \ge n$，由$a_{d-2} \le 2a_{d-3}$知$6a_{d-3} \ge n$，一般的，有$3*2^{d-(i+2)}a_i \ge n$, 当$3*2^{d-(i+2)}a_i < n$时，状态空间树种以ai为根节点的子树不可能在第d层之前找到最短加法链，因此可以剪去。

$$\begin{cases} \log(n/3a_i)+i+2>d & 0 \le i \le d-2 \\ \log(n/a_i)+i>d & d-1 \le i \le d \end{cases}$$

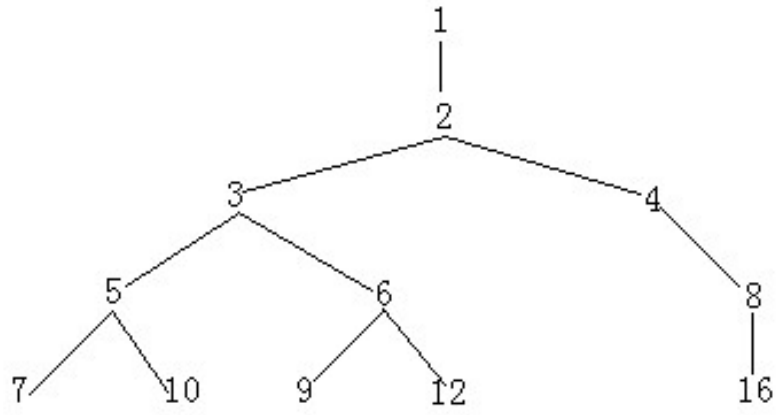
剪枝函数

- 对于n为偶数时，当n是2的幂，唯一最短加法链是1,2,4,8,…,2^m.当n不是2的幂，可将n表示为$n=2^t(2k+1),k\geqslant 1$
- 设在求正整数n的最短加法链的逐步深化迭代搜索算法中，当前搜索深度为d。且正整数可表示为$n=2^t(2k+1)$，$k>0$，则在状态空间树的第i层结点a_i处的一个剪枝条件是

$$\begin{cases} \log(n/3a_i)+i+2>d & 0\leq i\leq d-t-2 \\ \log(n/a_i)+i>d & d-t-1\leq i\leq d \end{cases}$$

最短加法链长的上界

与加法链问题密切相关的幂树给出了l(n)的更精确的上界。



假设已定义了幂树T的第k层结点，则T的第k+1层结点可定义如下。依从左到右顺序取第k层结点ak，定义其按从左到右顺序排列的儿子结点为ak+aj，0≤j≤k。其中a0,a1,…,ak，是从T的根到结点ak的路径。且ak+aj在T中未出现过。

含正整数n的部分幂树T容易在线性时间内用迭代搜索方式构造出来。

优化算法

综合前面的讨论，对构造最短加法链的标准回溯法作如下改进。

(1)采用逐步深化迭代搜索策略；

(2)利用l(n)的下界lb(n)对迭代深度作精确估计；

(3)采用剪枝函数对问题的状态空间树进行剪枝搜索，加速搜索进程；

(4)用幂树构造l(n)的精确上界ub(n)。

当lb(n)=ub(n)时，幂树给出的加法链已是最短加法链。

当lb(n)<ub(n)时，用改进后的逐步深化迭代搜索算法，从深度d=lb(n)开始搜索。