***1、问题描述：***

     n个作业{1，2，…，n}要在由2台机器M1和M2组成的流水线上完成加工。每个作业加工的顺序都是先在M1上加工，然后在M2上加工。M1和M2加工作业i所需的时间分别为ai和bi。***流水作业调度问题要求确定这n个作业的最优加工顺序，使得从第一个作业在机器M1上开始加工，到最后一个作业在机器M2上加工完成所需的时间最少***。  
     ***2、问题分析***

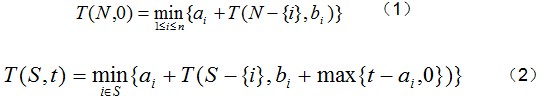
     直观上，一个最优调度应使机器M1没有空闲时间，且机器M2的空闲时间最少。在一般情况下，机器M2上会有机器空闲和作业积压2种情况。设全部作业的集合为N={1，2，…，n}。S是N的作业子集。在一般情况下，机器M1开始加工S中作业时，机器M2还在加工其他作业，要等时间t后才可利用。将这种情况下完成S中作业所需的最短时间记为T(S,t)。流水作业调度问题的最优值为T(N,0)。

     设*π*是所给n个流水作业的一个最优调度，它所需的加工时间为 a*π(1)*+T’。其中T’是在机器M2的等待时间为b*π(1)*时，安排作业*π(2)*，…，*π(n)*所需的时间。

      记S=N-{*π(1)*}，则有T’=T(S,b*π(1)*)。

      证明：事实上，由T的定义知T’>=T(S,b*π(1)*)。若T’>T(S,b*π(1)*)，设*π*’是作业集S在机器M2的等待时间为b*π(1)*情况下的一个最优调度。则*π(1)*，*π'(2)*，…，*π'(n)*是N的一个调度，且该调度所需的时间为a*π(1)*+T(S,b*π(1)*)<a*π(1)*+T’。这与*π*是N的最优调度矛盾。故T’<=T(S,b*π(1)*)。从而T’=T(S,b*π(1)*)。这就证明了流水作业调度问题具有最优子结构的性质。

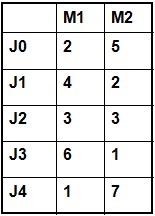
     由流水作业调度问题的最优子结构性质可知:



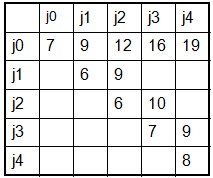
     从公式（1）可以看出，该问题类似一个排列问题，求N个作业的最优调度问题，利用其子结构性质，对集合中的每一个作业进行试调度，在所有的试调度中，取其中加工时间最短的作业做为选择方案。将问题规模缩小。公式（2）说明一般情况下，对作业集S进行调度，在M2机器上的等待时间，除了需要等该部件在M1机器上完成时间，还要冲抵一部分原来的等待时间，如果冲抵已成负值，自然仍需等待M1将作业做完，所以公式取max{t-ai,0}。

***3、动态规划法求解思路***

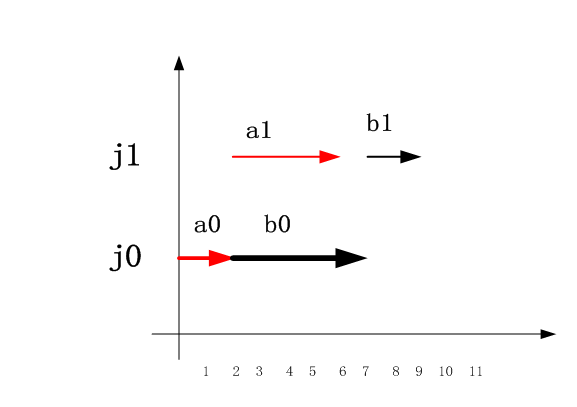
    假设有一组作业需要在M1和M2 两台机器上进行流水作业，他们在M1和M2上的作业时间如下表：



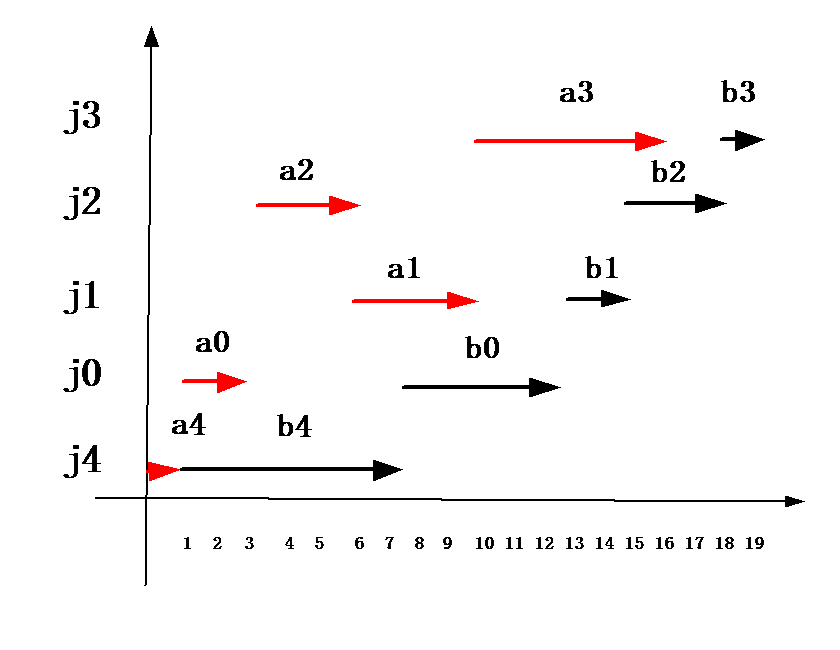
     问题是如何安排他们的加工顺序，使得，到最后一个作业在机器M2上加工完成所需要的时间最少。也就是所有作业在两台机器全部加工完成所需的时间最少。  
     ***思路如下***：考虑如果只有一个作业的情况，肯定所需时间就是它自身需要在M1和M2 上的加工时间总和；如果有两个作业就要考虑在两种不同的加工顺序下选取最优的一种作为候选，三个作业的时会出现三种组合情况(0,(1,2)); (1,(0,2)); (2,(0,1)),拿第一种为例,它表示先加工作业0,然后再按照作业1和作业2的优化顺序加工;将三种的作业时间计算出来,取最小值,即为三个作业的优化结果,同理可对更多的作业进行排序优化。具体做法是，用类似矩阵连乘的办法，自底向上将所有能的情况计算出来，并产生一个表，供后面的计算查用，减少重复计算的工作量。



     对于j1 作业M2 的等待时间为b0,实际上在M2加工j0作业的同时，M1 并行加工j1,实际它需要等待b1-a1时间。



      2+4+(5-4)+2=9

     从J0和J1两个作业的加工顺序，可以看出，先加工J0后J1，所用时间最短为9，将其填入表中，依此类推，即可得出最优解。  


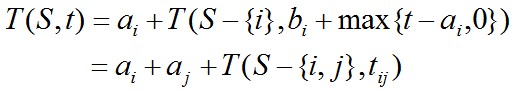
     a4+a0+a2+a1+a3+[(b4+b0+b1+b2)-(a0+a1+a2+a3)]+b3

     =1+2+3+4+6+[(7+5+2+3)-(2+4+3+6)]+1

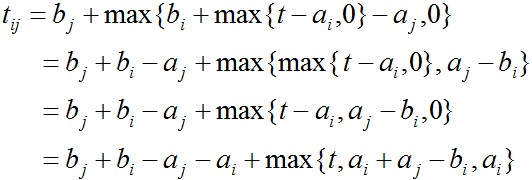
     =16+[17-15]+1=19

     选其中加工时间短的作为候选方案；在具体计算时非最优子集不必考虑，这样可以减少计算次数。

4、流水作业调度的***Johnson法则***

     设*兀*是作业集S在机器M2的等待时间为t时的任一最优调度。若在这个调度中，安排在最前面的两个作业分别是i 和j ，即*π*(1)=I,*π*(2)=j。则有动态规划递归式可得  


     其中

  
     如果作业i和j满足min{bi,aj} ≥min{bj,ai},则称作业i和j满足Johnson不等式。如果作业i和j 不满足Johnson不等式，则交换作业i和j满足Johnson不等式。

     证明 ：在作业集S中，对于机器M2 的等待时间为t的调度*π*，交换作业i和j 的加工顺序，得到作业集S 的另一个调度*π*’，它所需的加工时间为

http://img.my.csdn.net/uploads/201303/15/1363336020_5879.jpg

http://img.my.csdn.net/uploads/201303/15/1363336047_9386.jpg  
     当作业i和j 满足Johnson 不等式 min{bi,aj} ≥min{bj,ai}时，有

http://img.my.csdn.net/uploads/201303/15/1363336109_7186.jpg

从而

http://img.my.csdn.net/uploads/201303/15/1363336155_5487.jpg  
由此可得

http://img.my.csdn.net/uploads/201303/15/1363336221_8745.jpg  
因此，对任意t 有

http://img.my.csdn.net/uploads/201303/15/1363336286_3177.jpg

从而，tij≤tji,由此可见http://img.my.csdn.net/uploads/201303/15/1363336333_4858.jpg，换句话说，当作业i 和j不满足Johnson 不等式时，交换它们的加工顺序后，作业i和j满足Johnson 不等式，且不增加加工时间。由此可知，对于流水作业调度问题，必存在最优调度π，使得作业π(i)和π(i+1)满足Johnson 不等式：

http://img.my.csdn.net/uploads/201303/15/1363336398_1566.jpg  
这样的调度π称为满足Johnson 法则的调度。进一步还可以证明，调度满足Johnson 法则当且仅当对任意i<j 有：

http://img.my.csdn.net/uploads/201303/15/1363336438_9784.jpg  
     由此可知，任意两个满足Johnson 法则的调度具有相同的加工时间，从而所有满足Johnson 法则的调度均为最优调度。

5、流水作业调度问题***Johnson算法***

从上面的分析可知，流水作业调度问题一定存在满足Johnson法则的最优调度，且容易由下面的算法确定：

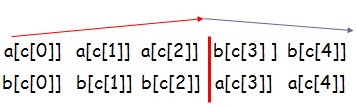
    流水作业调度问题的Johnson算法：

    (1)令N1={i|ai<bi},N2={i|ai>=bi};

    (2)将N1中作业按ai的非减序排序；将N2中作业按bi的非增序排序；

    (3)N1中作业接N2中作业构成满足Johnson法则的最优调度。

    Johnson算法中分类及排序的作用（验证不等式）设数组c[]为排序后的作业排列，排序结果如下：



     红线左侧满足 a[c[i]]<=b[c[i]] 和 a[c[i]]<=a[c[i+1]] 符合johnson 不等式，min(b[c[i]],a[c[i+1]])>=min(b[c[i+1]],a[c[i]])其调度顺序最优；  
     红线右侧满足 b[c[i]]<=a[c[i]] 和 b[c[i]]>=b[c[i+1]] 符合johnson 不等式，min(b[c[i]],a[c[i+1]])>=min(b[c[i+1]],a[c[i]])其调度顺序最优；

     中间过渡部分横向比较，左侧a[c[i]]< b[c[i]] 右侧b[c[i+1]]<=a[c[i+1] ]满足min(b[c[i]],a[c[i+1]])>=min(b[c[i+1]],a[c[i]])其调度顺序也最优；

     程序具体代码如下：

**[cpp]** [view plaincopy](http://blog.csdn.net/liufeng_king/article/details/8678316)

1. //3d9 动态规划 流水作业调度问题
2. #include "stdafx.h"
3. #include <iostream>
4. **using** **namespace** std;
6. **const** **int** N = 5;
8. **class** Jobtype
9. {
10. **public**:
11. **int** operator <=(Jobtype a) **const**
12. {
13. **return**(key<=a.key);
14. }
15. **int** key,index;
16. **bool** job;
17. };
19. **int** FlowShop(**int** n,**int** a[],**int** b[],**int** c[]);
20. **void** BubbleSort(Jobtype \*d,**int** n);//本例采用冒泡排序
22. **int** main()
23. {
24. **int** a[] = {2,4,3,6,1};
25. **int** b[] = {5,2,3,1,7};
26. **int** c[N];
28. **int** minTime =  FlowShop(N,a,b,c);
30. cout<<"作业在机器1上的运行时间为："<<endl;
31. **for**(**int** i=0; i<N; i++)
32. {
33. cout<<a[i]<<" ";
34. }
35. cout<<endl;
36. cout<<"作业在机器2上的运行时间为："<<endl;
37. **for**(**int** i=0; i<N; i++)
38. {
39. cout<<b[i]<<" ";
40. }
41. cout<<endl;
43. cout<<"完成作业的最短时间为："<<minTime<<endl;
44. cout<<"编号从0开始，作业调度的顺序为："<<endl;
45. **for**(**int** i=0; i<N; i++)
46. {
47. cout<<c[i]<<" ";
48. }
49. cout<<endl;
50. **return** 0;
51. }
53. **int** FlowShop(**int** n,**int** a[],**int** b[],**int** c[])
54. {
55. Jobtype \*d = **new** Jobtype[n];
56. **for**(**int** i=0; i<n; i++)
57. {
58. d[i].key = a[i]>b[i]?b[i]:a[i];//按Johnson法则分别取对应的b[i]或a[i]值作为关键字
59. d[i].job = a[i]<=b[i];//给符合条件a[i]<b[i]的放入到N1子集标记为true
60. d[i].index = i;
61. }
63. BubbleSort(d,n);//对数组d按关键字升序进行排序
65. **int** j = 0,k = n-1;
67. **for**(**int** i=0; i<n; i++)
68. {
69. **if**(d[i].job)
70. {
71. c[j++] = d[i].index;//将排过序的数组d，取其中作业序号属于N1的从前面进入
72. }
73. **else**
74. {
75. c[k--] = d[i].index;//属于N2的从后面进入，从而实现N1的非减序排序，N2的非增序排序
76. }
77. }
79. j = a[c[0]];
80. k = j+b[c[0]];
81. **for**(**int** i=1; i<n; i++)
82. {
83. j += a[c[i]];//M1在执行c[i]作业的同时，M2在执行c[i-1]号作业，最短执行时间取决于M1与M2谁后执行完
84. k = j<k?k+b[c[i]]:j+b[c[i]];//计算最优加工时间
85. }
87. **delete** d;
88. **return** k;
89. }
91. //冒泡排序
92. **void** BubbleSort(Jobtype \*d,**int** n)
93. {
94. **int** i,j,flag;
95. Jobtype temp;
97. **for**(i=0;i<n;i++){
98. flag = 0;
99. **for**(j=n-1;j>i;j--){
100. //如果前一个数大于后一个数，则交换
101. **if**(d[j]<=d[j-1]){
102. temp = d[j];
103. d[j] = d[j-1];
104. d[j-1] = temp;
105. flag = 1;
106. }
107. }
108. //如果本次排序没有进行一次交换，则break，减少了执行之间。
109. **if**(flag == 0){
110. **break**;
111. }
112. }
113. }

运行结果如下：

