INT10 2W6 _dp

定义： 一种解决最优化问题的方法，效率高速度快，最简单的一个例子就是找最长递增子序列的长度，比如nums=【1,5,2，4，3】要找到他的最长递增序列的长度，首先我们经常会想到到的是一个暴力枚举和搜索，这样诚然是能得到答案的，但是时间太长，所以就需要我们使用动态规划来解决。如何解决呢？这里就需要我们用到记忆的做法了。他的基本思想是解答问题的不同部分来以此得到解，乍一听与分治类似但是其实区别满大的，区别在于动态规划的子问题是存在重叠的，所以需要我们提前储存起来避免重复计算

设计一个动态规划算法，通常可按以下几个步骤进行：

(1)分析最优解的性质，并刻画其结构特征。

(2)递归地定义最优值(每个解都有一个值，代价)。

(3)根据递归方程分解子问题，直到不能分为止。

(4)自底向上的方式计算最优值，并记录构造最优解所需的信息。

(5)根据计算最优值时得到的信息，构造一个最优解。

应用场景： 各类背包问题，流水线问题。Dp的许多题都可以视作是改动后的背包

优点： **优化重复计算**：DP 可以避免重复计算，通过存储中间结果来提高效率。

缺点： **空间复杂度高**：DP 需要存储中间结果，因此可能需要较大的内存空间。

例子： 斐波那契数列，代码如下

```
public static int fibonacci(int n) {
        if (n <= 1) {
            return n;
        }
        int[] dp = new int[n + 1];
        dp[0] = 0;
        dp[1] = 1;

        for (int i = 2; i <= n; i++) {
            dp[i] = dp[i - 1] + dp[i - 2];
        }
```

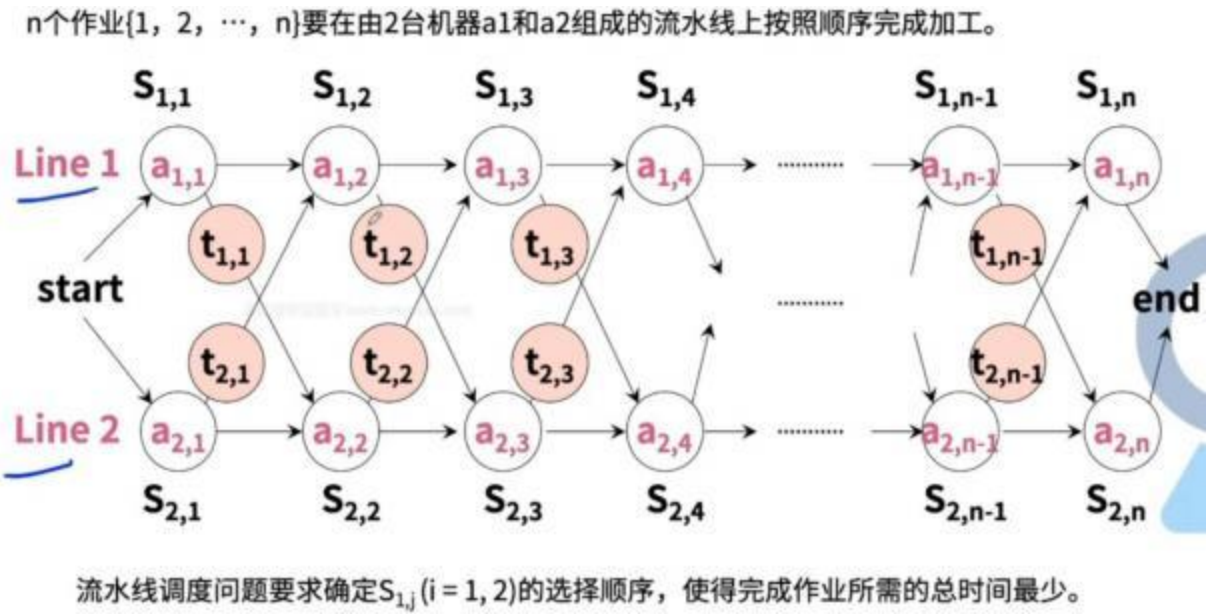
```
        return dp[n];
    }
```

这里我们可以看到，对于我们的斐波那契而言区别于以往的递归，我们使用了新数组DP来存储计算的值，通过这一过程我们减少了不必要的重复计算，由此来优化我们的时间复杂度。我们可以通过伪代码来观察

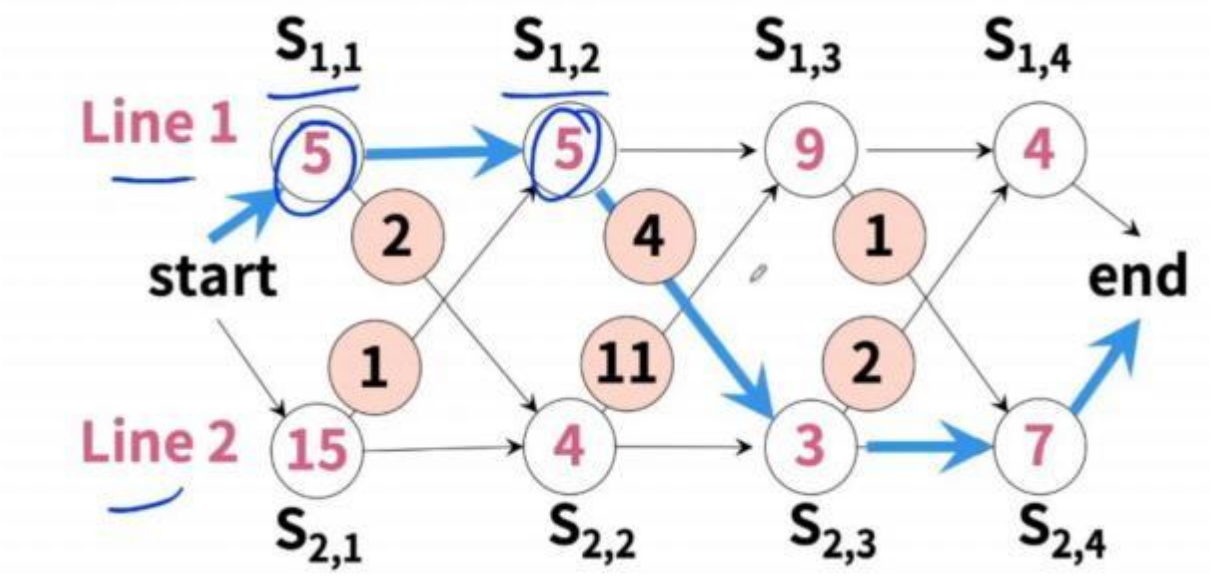
```
Procedure F(n)
    Set A[0] = 0
    Set A[1] = 1
    for i = 2 to n do
            A[i] = A[i-1] + A[i-2]
    return A[n]
```

区别于原始的递归O（2^n），我们这里的时间复杂度只是O（n），由此可以看到他时间复杂度上面 的优越性。

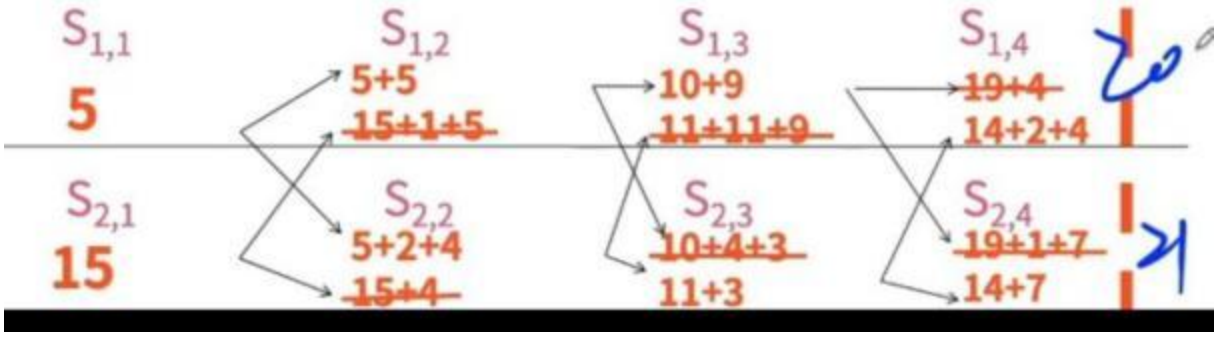
流水线调度是dp的另外一个重要例子，他也被叫做线性dp，具体来说我个人认为可以用如下内容来展示



首先我们需要明确其中含义，对于两条流水线共同完成作业的情况来说，（tx，j）代表第j个任务从x号线转移到另外一条的时间，（sx，j）则是代表第j个作业在x线的第j个工作站完成，这一流程要求我们确定选择工作站的顺序来让他时间最短，现在我们可以正式进入例子了

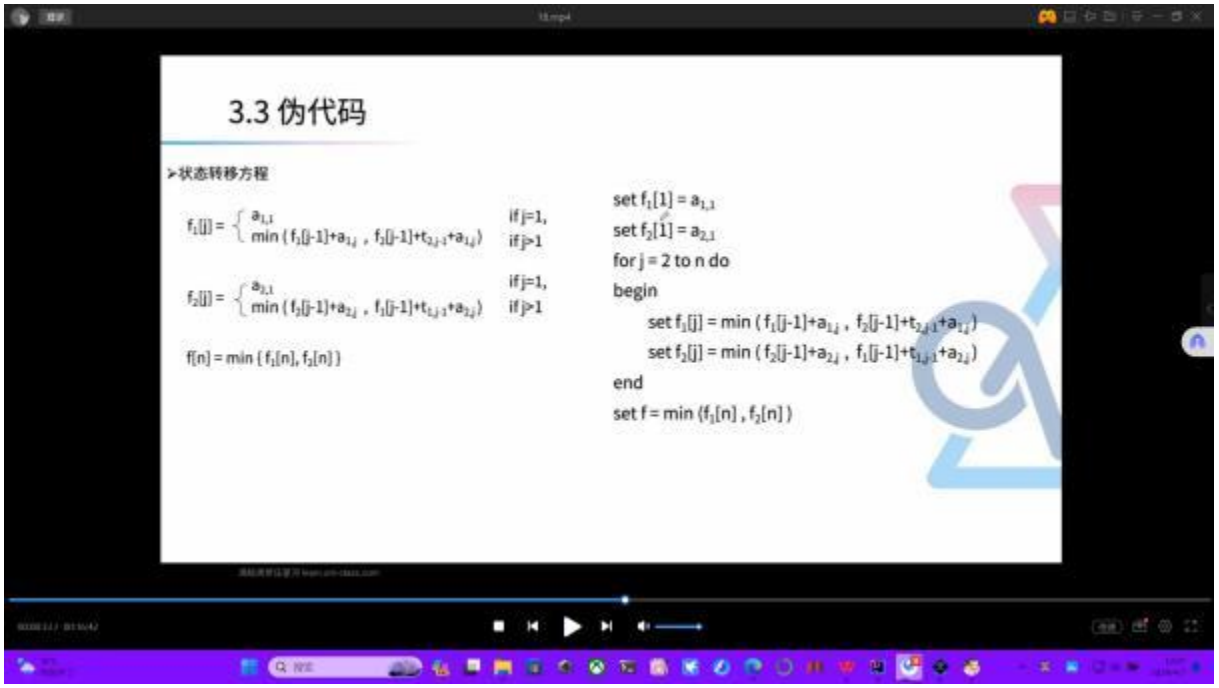


对于上图，我们采用的就是动态规划来解决了，首先我们对应分析，每一步到达两个工作站的时间那么我们可以类比迪杰斯特拉算法列表来分析



这样，我们最终就可以得到该流水线的最短作业时间是20了。他的思路仍然是拆分，给定到达每一个顶点的最短距离时间由此得到最后的最短时间，思路上是类似迪杰斯特拉的，但是过程不同，因为动态规划会储存每一步进行时的数据，而迪杰斯特拉更侧重的是锁定点最后回溯时再累加。

下面让我们看看伪代码来进行分析，如下是他的状态转移方程和伪代码



从图中我们可以分析对应每一个值他应该如何取，然后对应我们的第j个任务，分别用两个流水线所代表的集合来进行规划，得到对应集合中最短的路径长度，最后再对二者的总计时长进行比较。值得注意的是他得到的只是长度而不是具体的路径 。

背包问题暂时没有讲我就先不补充了，等他讲解后我会更新。