WARSHALL算法与动态规划

软件学院《离散数学1》课程组 王聪

WARSHALL

- 算法目标：
  - 求解关系R的传递闭包
- 背景
  - 当有限集合A的元素较多时，用矩阵算法求A上的关系R的传递闭包仍很复杂。
  - 1962年Warshall提出了一种有效的算法
- 算法复杂度
  - O(N^3)

WARSHALL算法伪代码

Algorithm 1 Warshall

Input: Relation R
Output: $M(R^+)$

```
1:  Matrix B = M(R)
2:  for i = 1; i ≤ n; i++ do
3:      for j = 1; j ≤ n; j++ do
4:          if B[j, i] == 1 then
5:              for k = 1; k ≤ n; k++ do
6:                  B[j, k] = B[j, k] ∨ B[i, k]
7:              end for
8:          end if
9:      end for
10: end for
11: return B
```

如何理解？

1. 算法求解的正确性？
- ✓ 遍历i的情况下，再遍历j
- ✓ 如果j可以到达i，那么只要是i可以到达的，j也可以到达
- ✓ 也就是这里的k

WARSHALL算法伪代码

Algorithm 1 Warshall

Input: Relation R
Output: $M(R^+)$

```
1:  Matrix B = M(R)
2:  for i = 1; i ≤ n; i++ do
3:      for j = 1; j ≤ n; j++ do
4:          if B[j, i] == 1 then
5:              for k = 1; k ≤ n; k++ do
6:                  B[j, k] = B[j, k] ∨ B[i, k]
7:              end for
8:          end if
9:      end for
10: end for
11: return B
```

如何理解？

1. 算法求解的正确性？
   - ✓ 遍历i的情况下，再遍历j
   - ✓ 如果j可以到达i，那么只要是i可以到达的，j也可以到达
   - ✓ 也就是这里的k

2. 算法求解的完全性？
   - ✓ 我们假设任有一条路径是p1→p2→...pn-1→pn
   - ✓ 那么这个算法的每次执行完毕变量i的一个循环过程，就对应的将这条路径中i出现的前后两项连通了
   - ✓ 而这个算法本身遍历了所有n个点，因此这条路径必然是会走通的

咋用呢？

- 口诀：列a行b有一，行a加到行b
- R={〈a,a〉,〈a,b〉,〈b,d〉,〈d,b〉}.

动态规划：数字三角形（先来看个例子）

- 在数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。
- 路径上的每一步都只能往左下或右下走。
- 只需要求出这个最大和即可，不必给出具体路径。

7

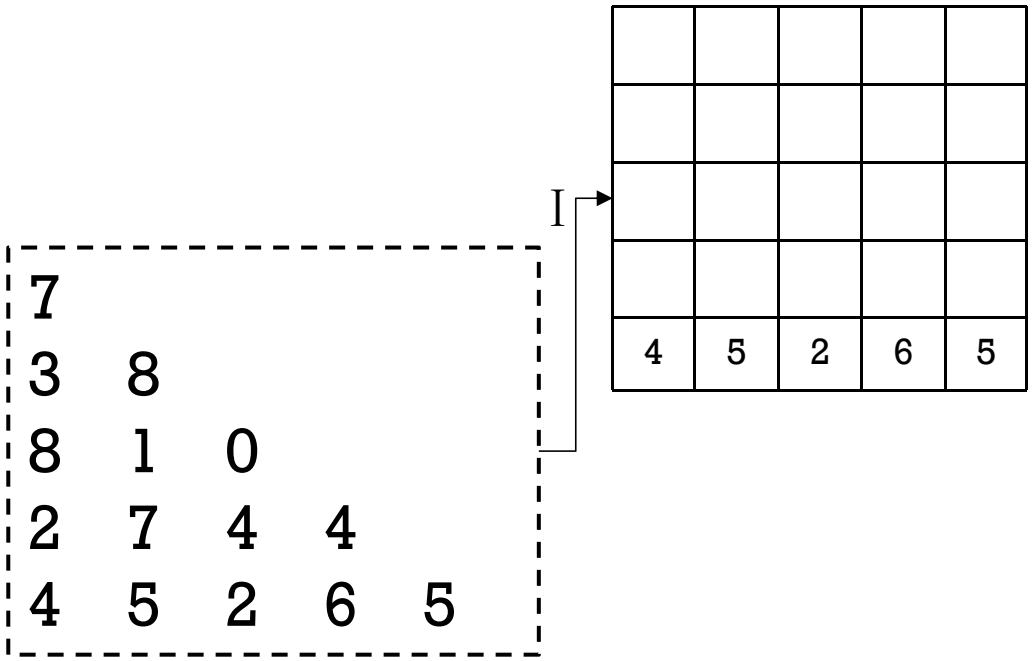
3 8

8 1 0

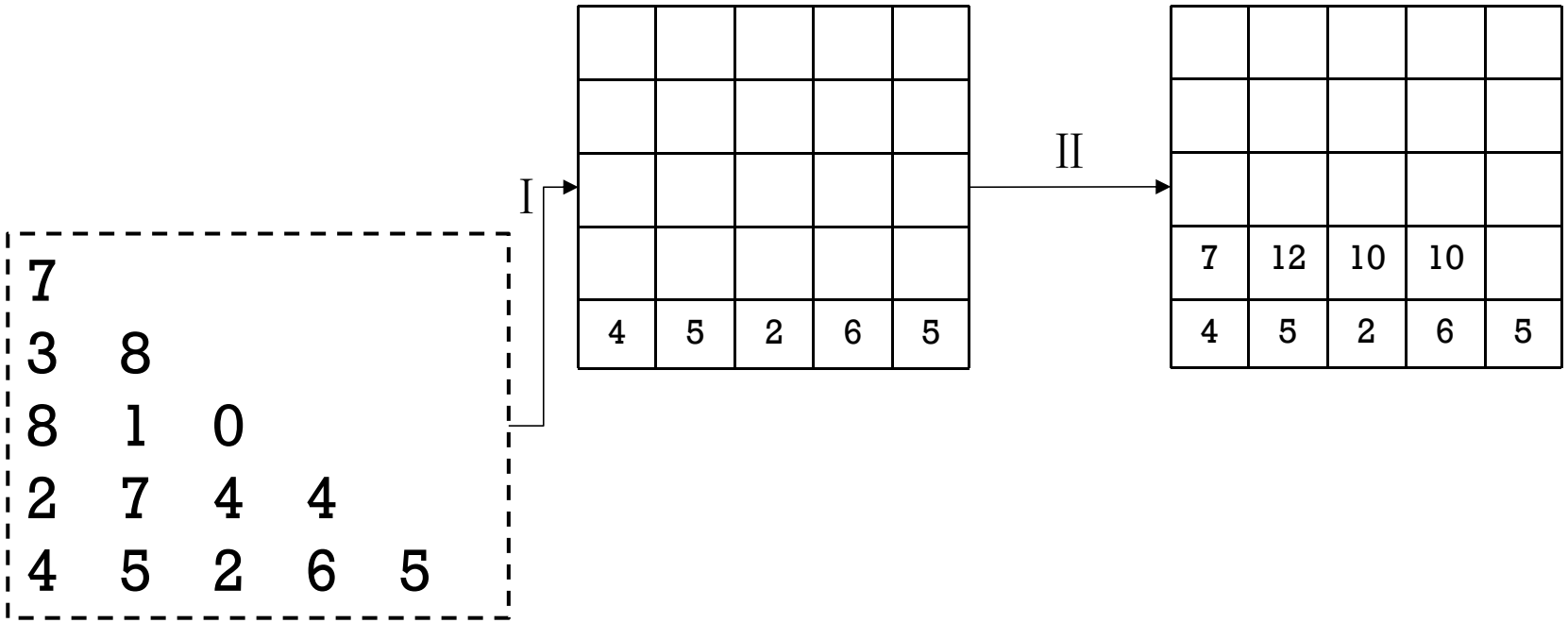
2 7 4 4

4 5 2 6 5

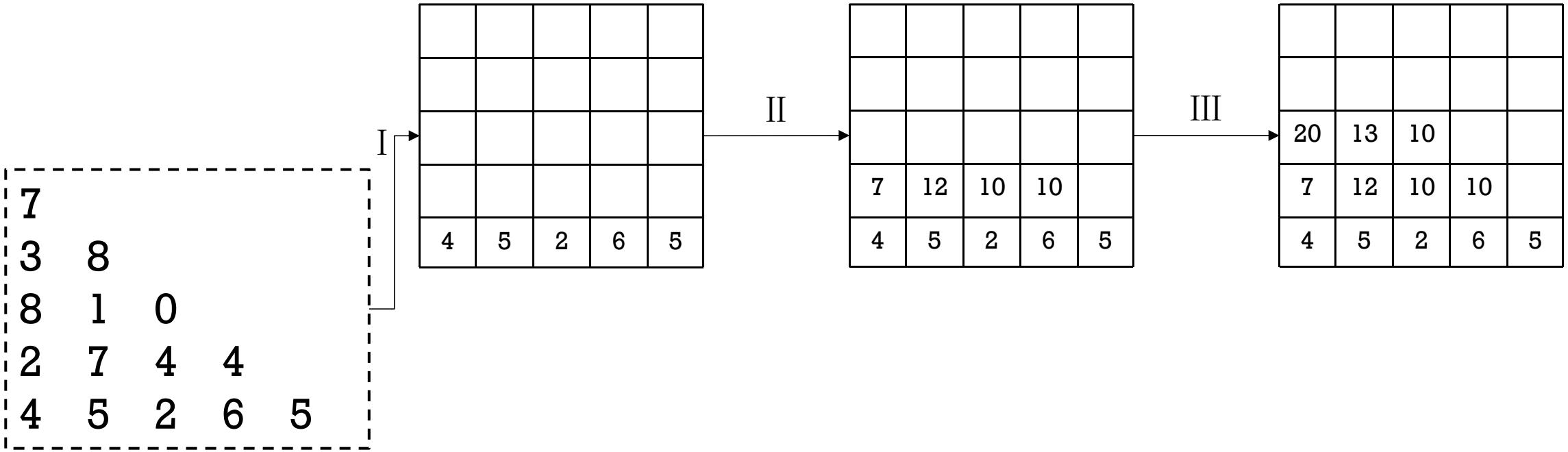
动态规划：数字三角形



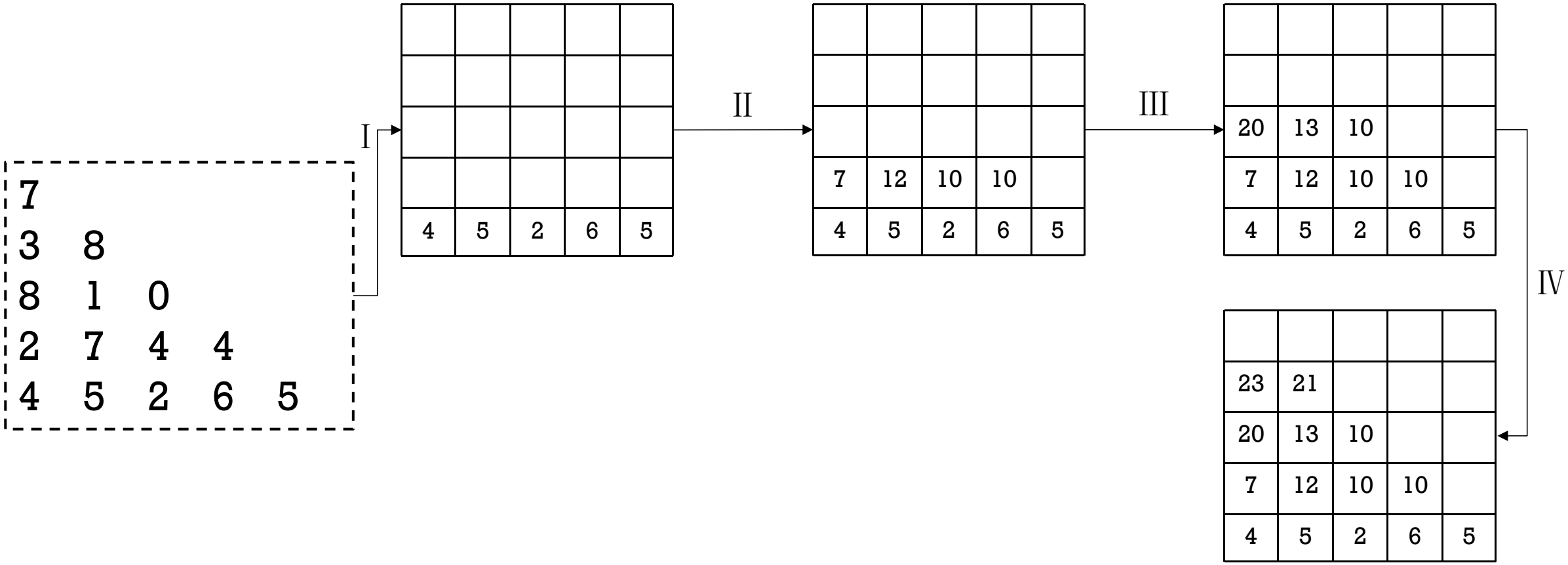
动态规划：数字三角形



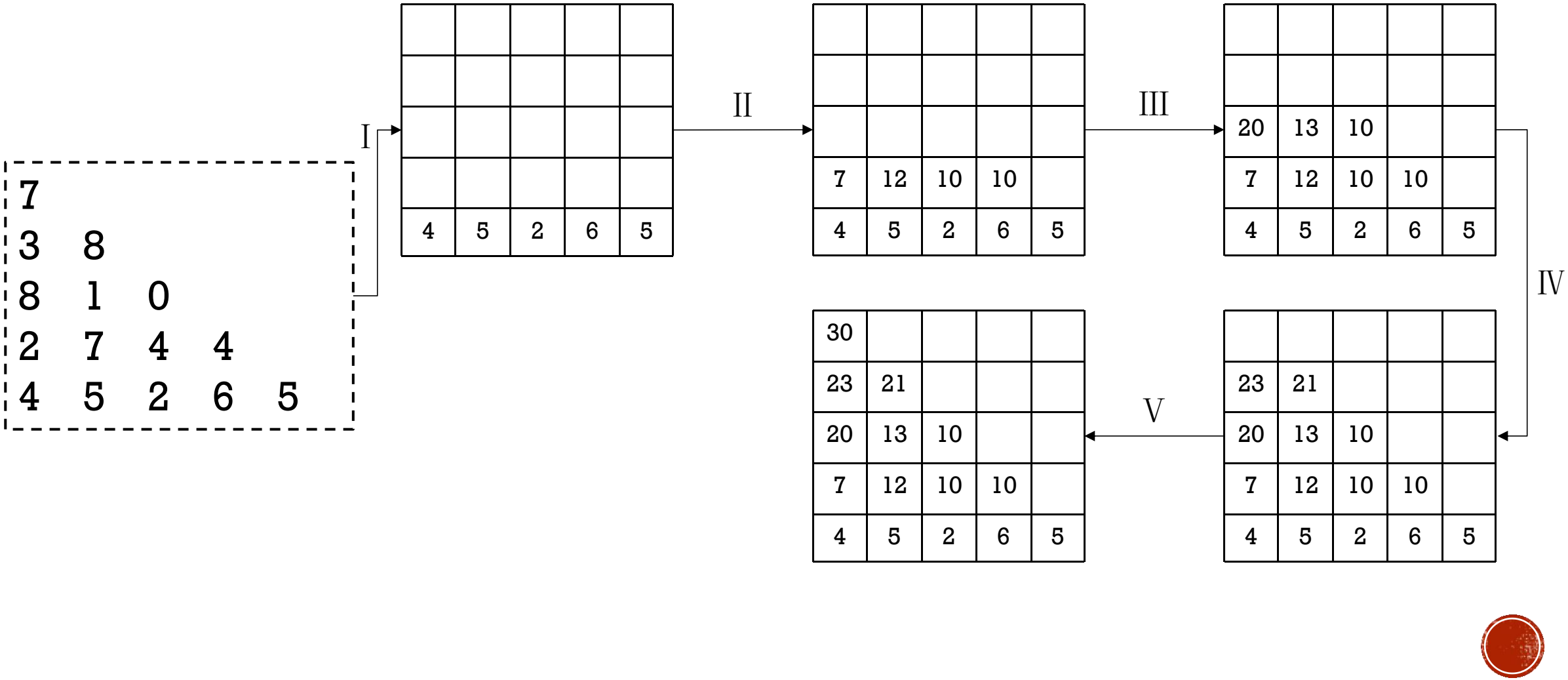
动态规划：数字三角形


7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
I
II
III
4 5 2 6 5
7 12 10 10
4 5 2 6 5
20 13 10
7 12 10 10
4 5 2 6 5

动态规划：数字三角形



动态规划：数字三角形


7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
I
II
III
IV
V
4 5 2 6 5
7 12 10 10
4 5 2 6 5
20 13 10
7 12 10 10
4 5 2 6 5
23 21
20 13 10
7 12 10 10
4 5 2 6 5
30
23 21
20 13 10
7 12 10 10
4 5 2 6 5

动态规划

- 适用情况
  - 最优子结构性质。如果问题的最优解所包含的子问题的解也是最优的，我们就称该问题具有最优子结构性质（即满足最优化原理）。最优子结构性质为动态规划算法解决问题提供了重要线索。
  - 无后效性。即子问题的解一旦确定，就不再改变，不受在这之后、包含它的更大的问题的求解决策影响。
  - 子问题重叠性质。子问题重叠性质是指在用递归算法自顶向下对问题进行求解时，每次产生的子问题并不总是新问题，有些子问题会被重复计算多次。动态规划算法正是利用了这种子问题的重叠性质，对每一个子问题只计算一次，然后将其计算结果保存在一个表格中，当再次需要计算已经计算过的子问题时，只是在表格中简单地查看一下结果，从而获得较高的效率。

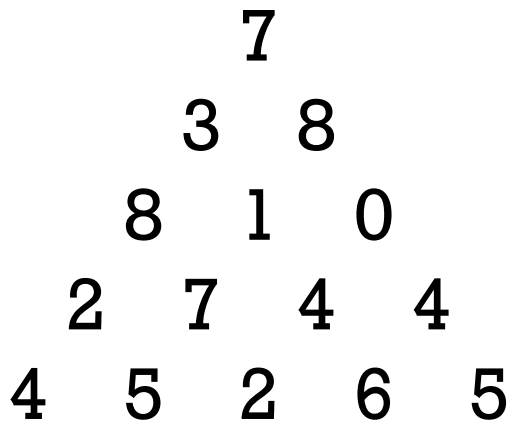
动态规划

- 算法介绍
  - 动态规划是通过组合子问题的解而解决整个问题的
  - 动态规划算法对每个子问题只求解一次，将其结果保存在一张表中，从而避免每次遇到各个子问题时重新计算答案
- 算法步骤：
  - 描述最优解的结构
  - 递归定义最优解的值
  - 按自底向上的方式计算最优解的值
  - 由计算出的结果构造一个最优解

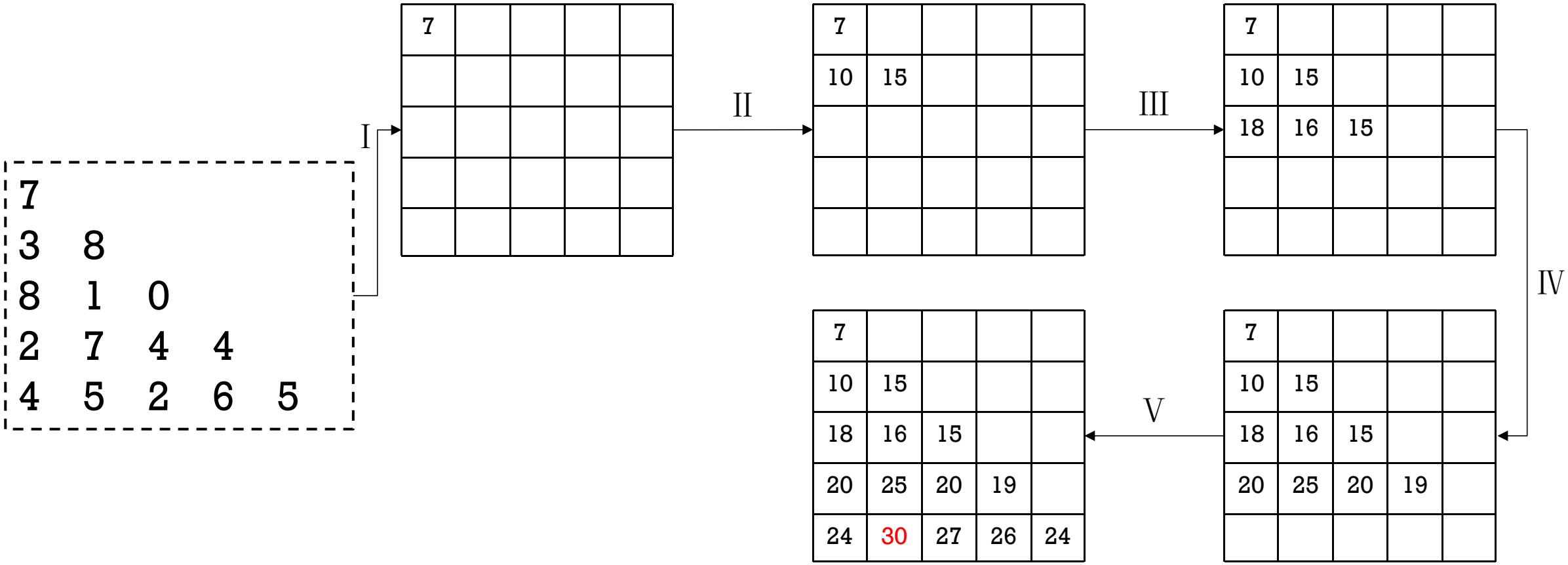
动态规划：数字三角形（再来看这个例子）

- 在上面的数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。
- 路径上的每一步都只能往左下或右下走。
- 只需要求出这个最大和即可，不必给出具体路径。三角形的行数大于1小于等于100，数字为0-99

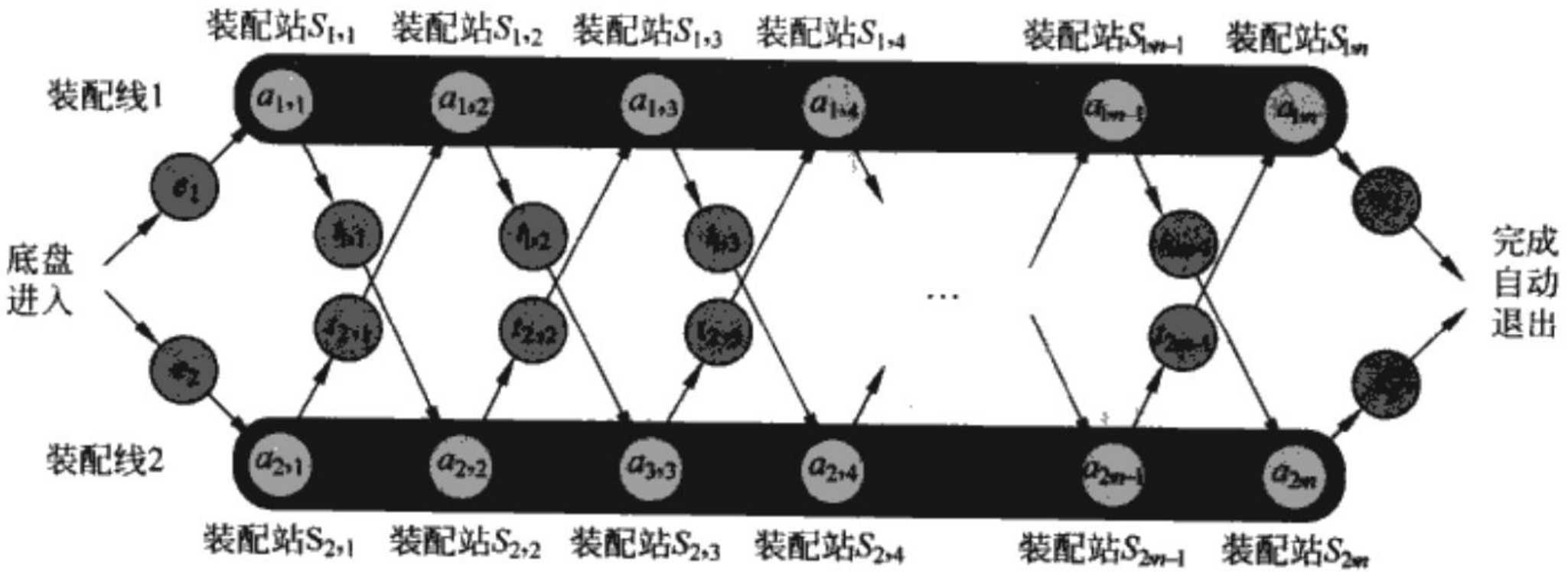
思考：如果还是用动态规划，还可以怎么做？



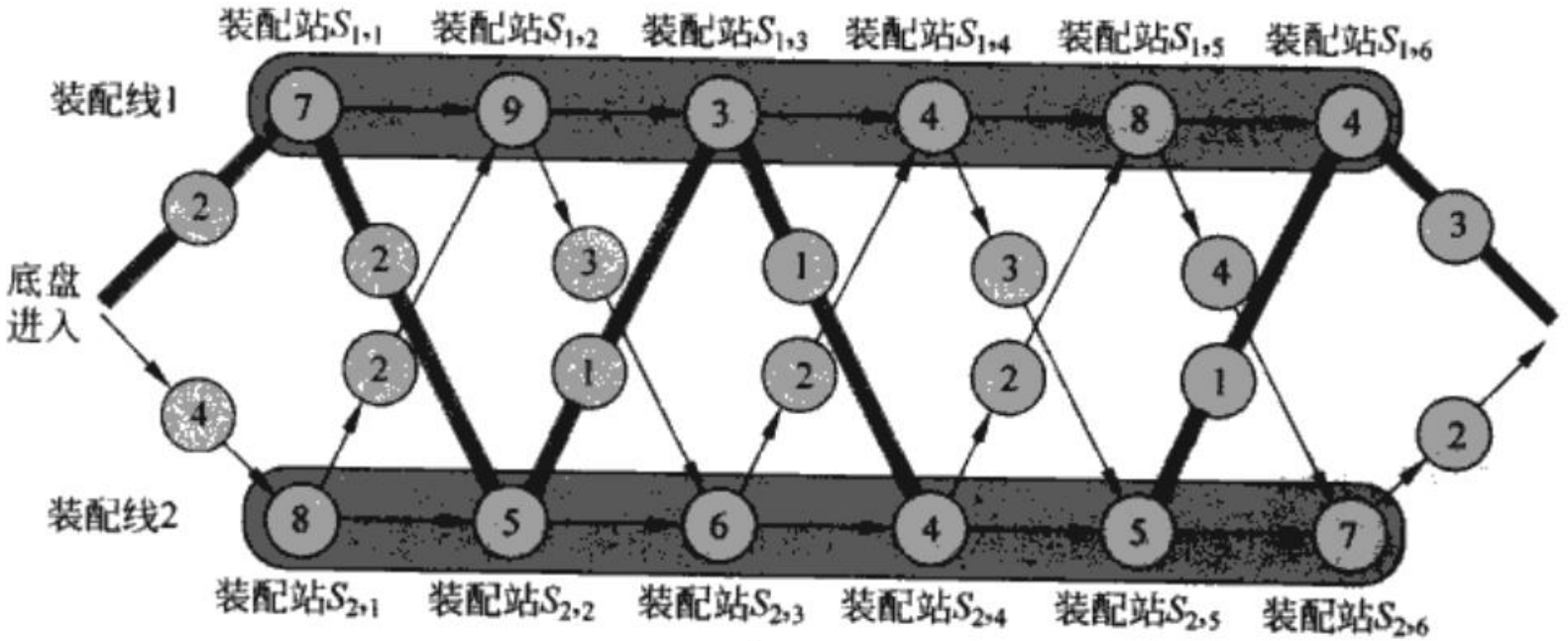
动态规划：数字三角形


7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
I
II
III
IV
V
7
10 15
18 16 15
20 25 20 19
24 30 27 26 24

动态规划：装配线调度


装配站$S_{1,1}$
装配站$S_{1,2}$
装配站$S_{1,3}$
装配站$S_{1,4}$
装配站$S_{1,n-1}$
装配站$S_{1,n}$
装配线1
底盘
进入
装配线2
完成
自动
退出
装配站$S_{2,1}$
装配站$S_{2,2}$
装配站$S_{2,3}$
装配站$S_{2,4}$
装配站$S_{2,n-1}$
装配站$S_{2,n}$

动态规划：装配线调度



j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

```
1   f1[1] ← e1 + a1,1
2   f2[1] ← e2 + a2,1
3   for j ← 2 to n
4       do if f1[j−1] + a1,j ≤ f2[j−1] + t2,j−1 + a1,j
5             then f1[j] ← f1[j−1] + a1,j
6                  l1[j] ← 1
7             else f1[j] ← f2[j−1] + t2,j−1 + a1,j
8                  l1[j] ← 2
9           if f2[j−1] + a2,j ≤ f1[j−1] + t1,j−1 + a2,j
10            then f2[j] ← f2[j−1] + a2,j
11                 l2[j] ← 2
12            else f2[j] ← f1[j−1] + t1,j−1 + a2,j
13                 l2[j] ← 1
14  if f1[n] + x1 ≤ f2[n] + x2
15    then f* ← f1[n] + x1
16         l* ← 1
17    else f* ← f2[n] + x2
18         l* ← 2
```

红包题

- 做出本题的同学，可以在微信群里抢红包，得红包最高者来给大家分享解题思路~非常简单的一道题目。

- 给定一个整数数组 *nums*，找到一个具有最大和的连续子数组（子数组最少包含一个元素），返其最大和。
- 输入: [-2,1,-3,4,-1,2,1,-5,4,-7,-3,5,-1,3]
- 输出: ?