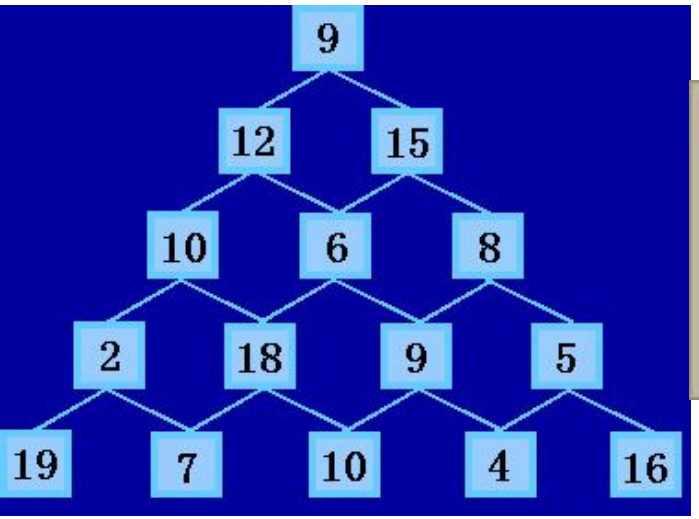
三 、基本的算法策略

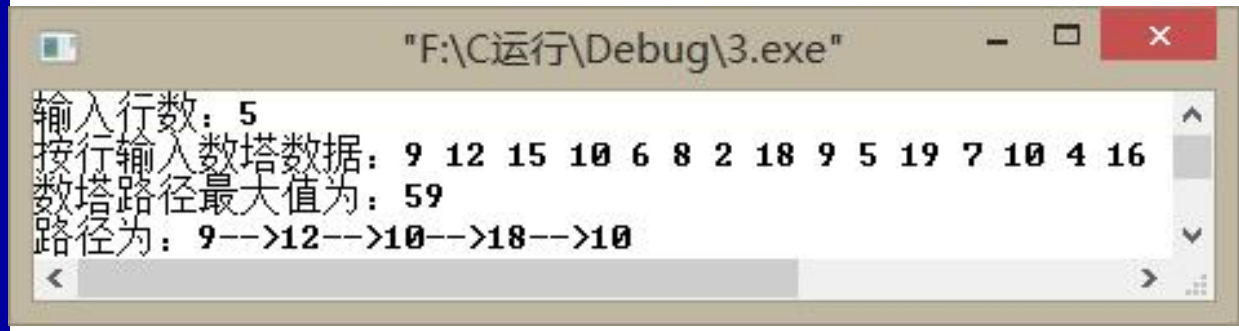
5、动态规划

- 在动态规划算法策略中，决策不是线性的而是全面考虑不同的情况分别进行，并通过多阶段决策来最终解决问题。在各个阶段采取决策后，会不断决策出新的数据,直到找到最优解。一个决策序列就是在变化的状态中产生出来的，故有“动态”的含义。这种多阶段决策最优化的解决问题的过程称为动态规划。

5、动态规划

- 通过一个简单的例子来说明动态规划的多阶段决策与贪婪算法的区别。
- 例：数塔问题：有如图所示的一个数塔，从顶部出发，在每一结点可以选择向左走或是向右走，一直走到底层，要求找出一条路径，使路径上的数值和最大。





5、动态规划

- 问题分析：
  - 这个问题用贪婪算法有可能会找不到真正的最大和。用贪婪的策略，则路径和分别为：

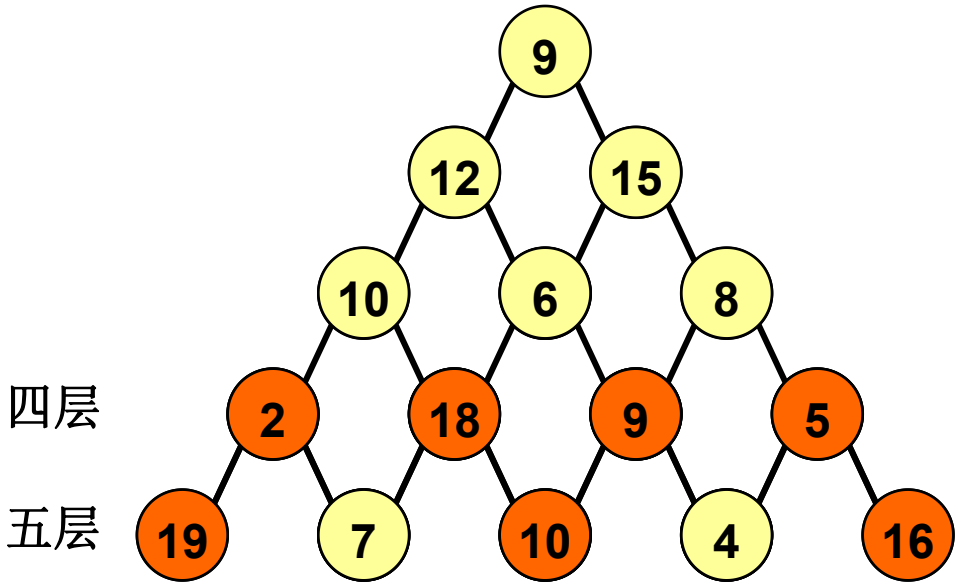
    9+15+8+9+10=51 （自上而下）

    19+2+10+12+9=52（自下而上）
  - 都得不到最优解，真正的最大和是：

    9+12+10+18+10=59。
  - 在知道数塔的全貌的前提下，可以用枚举法或图的搜索算法来完成。

5、动态规划

- 算法设计：
  - 动态规划设计过程如下：

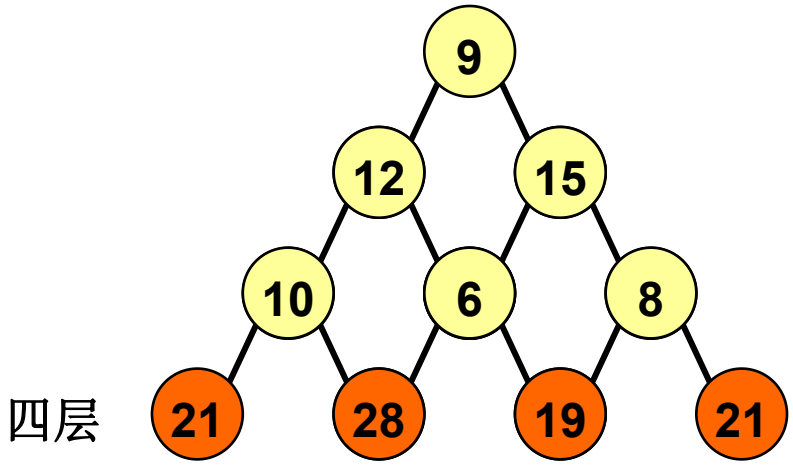


  - 1. 阶段划分：
    - 第一步对于第五层的数据，如下五次决策：
    - 对经过第四层2的路径选择第五层的19，
    - 对经过第四层18的路径选择第五层的10，
    - 对经过第四层9的路径也选择第五层的10，
    - 对经过第四层5的路径选择第五层的16。

5、动态规划

- 算法设计：
  - 以上的决策结果将五阶数塔问题变为4阶子问题，递推出第四层与第五层的和为:

    21(2+19), 28(18+10), 19(9+10), 21(5+16)。



  - 用同样的方法可以将4阶数塔问题,变为3阶数塔问题。
  - …… 最后得到的1阶数塔问题，就是整个问题的最优解。

5、动态规划

- 算法设计：
  - 2．存储、求解：
  - （1）原始信息存储
    - 原始信息有层数和数塔中的数据，层数用一个整型变量n存储，数塔中的数据用二维数组data，存储成如下的下三角矩阵：

9

12 15

10 6 8

2 18 9 5

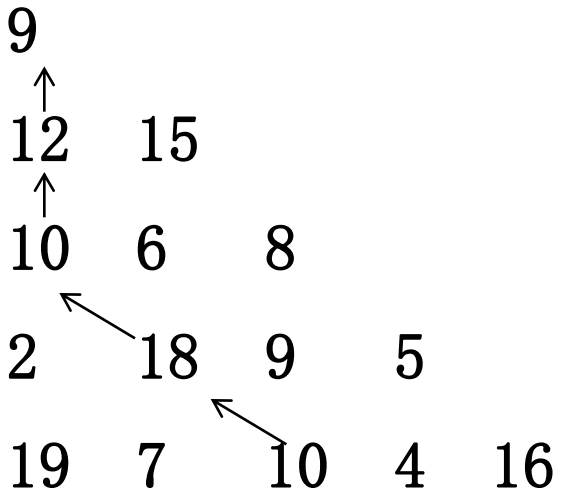
19 7 10 4 16

5、动态规划

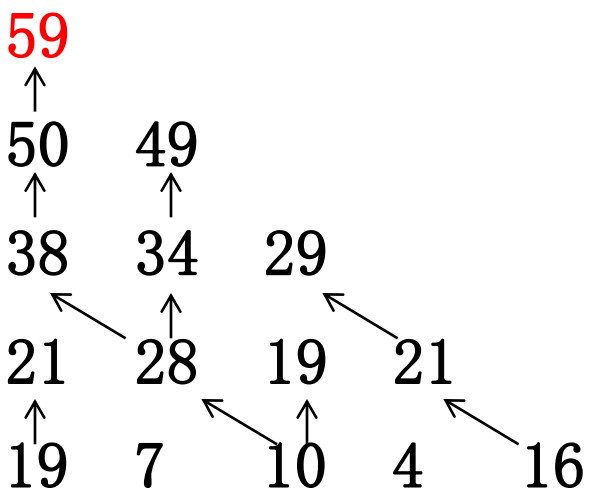
- 算法设计：
  - （2）动态规划过程存储
    - 用二维数组d存储各阶段的决策结果。二维数组d的存储内容如下：
    - d[n][j]=data[n][j]　　j=1,2,……,n;
    - d[n-1][j]=max(d[n][j], d[n][j+1])+data[n-1][j]
    - d[n-2][j]=max(d[n-1][j], d[n-1][j+1])+data[n-2][j]
    - ……
    - 最后d[1][1]存储的就是问题的解。

5、动态规划

- 算法设计：
  - （2）动态规划过程存储



数组data



数组d

5、动态规划

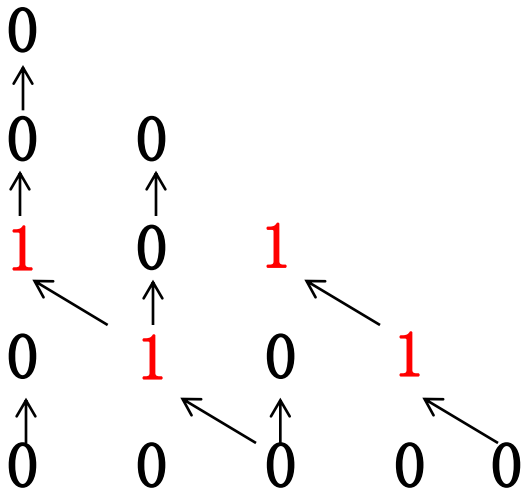
- 算法设计：
  - （3）最优解路径求解及存储
    - 根据数组data和数组d可以找到最优解的路径，但需要自顶向下比较数组data和数组d才可以找到，处理过程比较麻烦。
    - 输出data[1][1]　″9″
    - c=d[1][1]-data[1][1]=59-9=50　c与d[2][1],d[2][2] 比较
    - c与d[2][1]相等则输出data[2][1]　″12″
    - c=d[2][1]-data[2][1]=50-12=38　c与d[3][1],d[3][2] 比较
    - c与d[3][1]相等则输出data[3][1]　″10″

5、动态规划

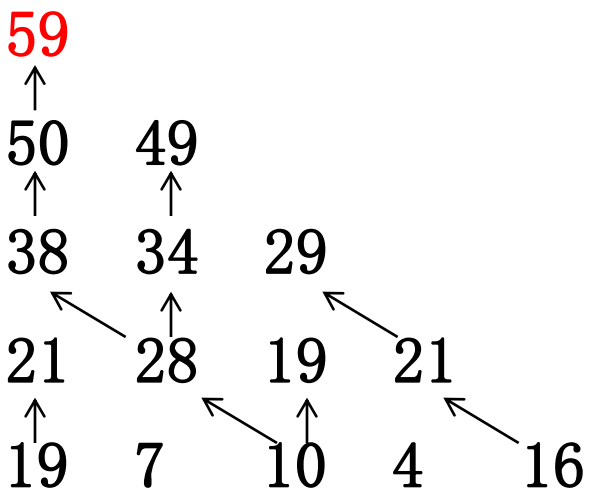
- 算法设计：
  - （3）最优解路径求解及存储
    - c=d[3][1]-data[3][1]=38-10=28　　c与d[4][1],d[4][2] 比较
    - c与d[4][2]相等则输出data[4][2]　″18″
    - c=d[4][2]-data[4][2]=28-18=10　　c与d[5][2],d[5][3] 比较
    - c与d[5][3]相等则输出data[5][3]　″10 ″

5、动态规划

- 算法设计：
  - 为了设计简洁的算法，用数组c记录解路径，解为左路径则为0，解为右路径则为1



数组c



数组d

5、动态规划

- 算法设计：

0				
0	0			
1	0	1		
0	1	0	1	
0	0	0	0	0

数组c

9					输出9，j+0
12	15				输出12，j+0
10	6	8			输出10，j+1
2	18	9	5		输出18，j+1
19	7	10	4	16	输出10

数组data

5、动态规划

- 算法实现：

```
int main()
{
    int data[50][50],d[50][50],c[50][50],i,j,n;
    printf("输入行数：");
    scanf("%d",&n);
    printf("按行输入数塔数据：");
    for(i=1;i<=n;i++)
        for(j=1;j<=i;j++)
        {
                scanf("%d",&data[i][j]);  //data存储数塔数据
                d[i][j]=data[i][j];  //d存储各阶段的决策结果
                c[i][j]=0;  //c记录解路径，数据初始为0
        }
```

5、动态规划

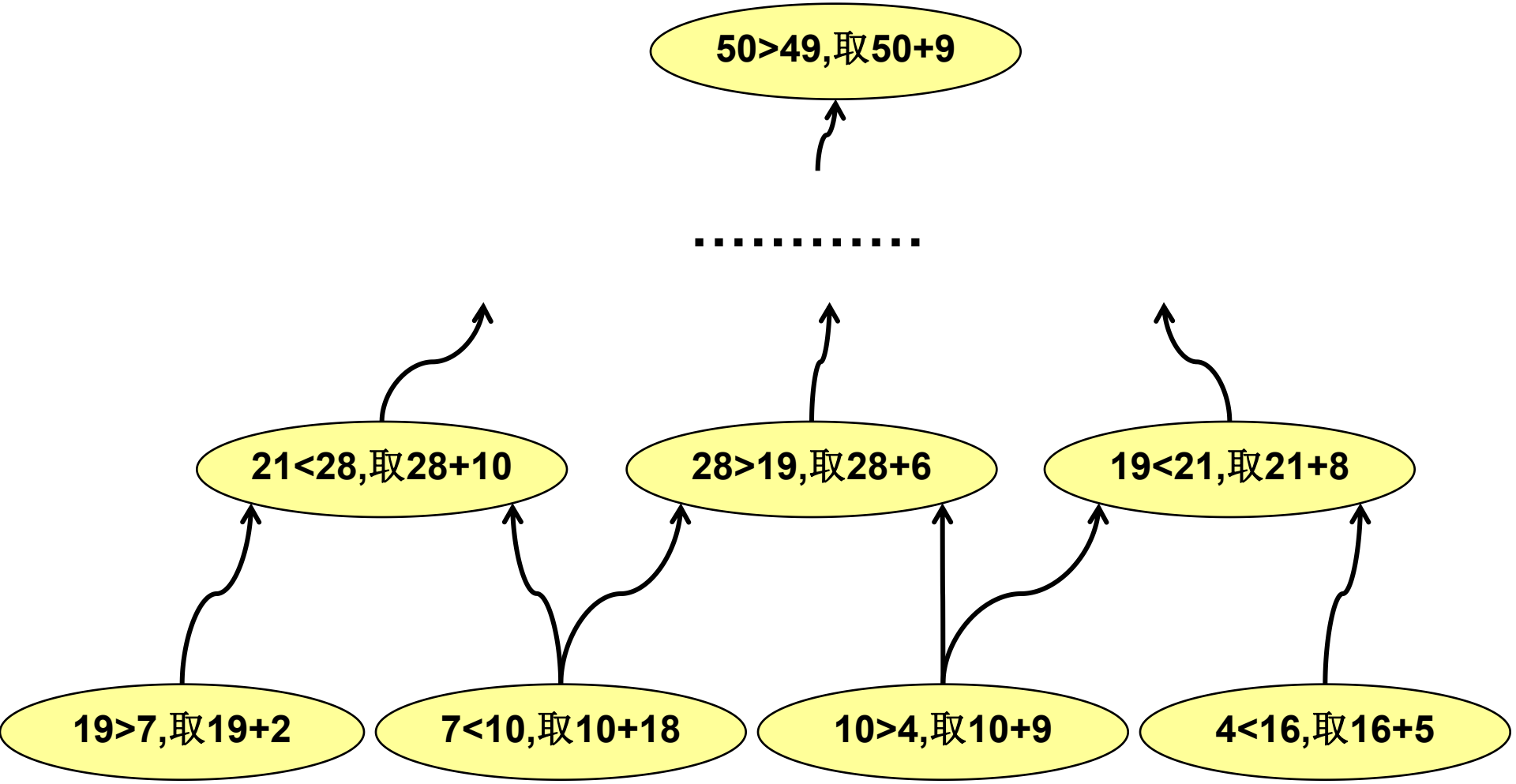
```
for(i=n-1;i>=1;i--)
      for(j=1;j<=i;j++)
              if(d[i+1][j]>d[i+1][j+1]) //取下一层两路径中较大的数
              {
                      d[i][j]=d[i][j]+d[i+1][j];  //与上一层路径中的数求和
                      c[i][j]=0;  //解为左路径则为0
              }
              else
              {
                      d[i][j]=d[i][j]+d[i+1][j+1];
                      c[i][j]=1;       //解为右路径则为1
              }
```

5、动态规划

```
    printf("数塔路径最大值为：%d\n",d[1][1]);
    j=1;
    printf("路径为：");
    for(i=1;i<=n-1;i++)
    {
        printf("%d-->",data[i][j]);  //依次输出每行中的解
        j=j+c[i][j];  //为左路径则j+0，为右路径则j+1
    }
    printf("%d\n",data[n][j]);
    return 0;
}
```

5、动态规划

- 算法分析：
  - 动态规划=贪婪策略+递推(降阶)+存储递推结果
  - 贪婪策略、递推算法都是在“线性”地解决问题，而动态规划则是全面分阶段地解决问题。可以说动态规划是“带决策的多阶段、多方位的递推算法”。
  - 动态规划的基本思想是把求解的问题分成许多阶段或多个子问题，然后按顺序求解各子问题。最后一个子问题就是初始问题的解。
  - 由于动态规划有重叠子问题的特点，为了减少重复计算，将其不同阶段的不同状态保存在一个二维数组中。



- 动态规划=贪婪策略+递推(降阶)+存储递推结果
- 把求解的问题分成许多阶段或多个子问题，最后一个子问题就是初始问题的解。

5、动态规划

- 例：一个小偷拿了一个背包出来活动，背包最多可以装50斤的东西。他发现了四件宝贝a，b，c，d。其中：

  a重10斤，价值60元；

  b重20斤，价值100元；

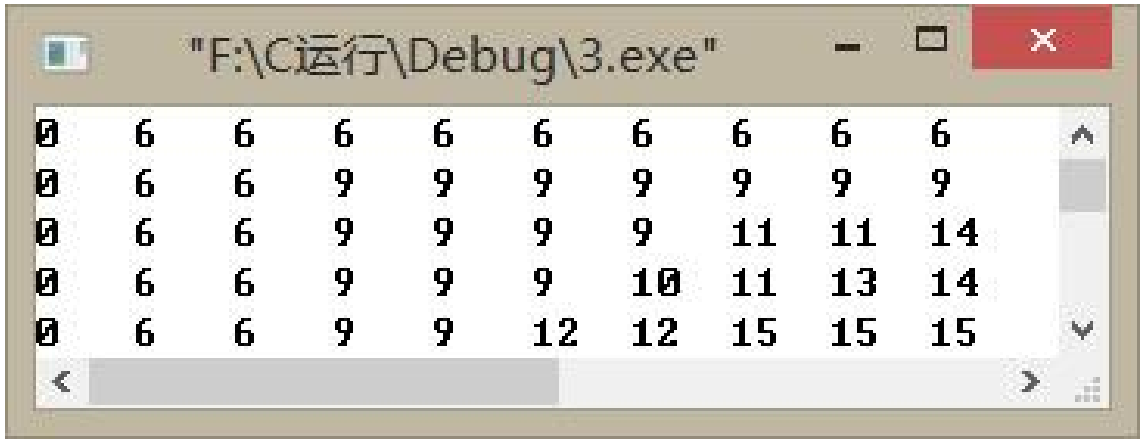
  c重20斤，价值120元；

  d重30斤，价值150元。

问：在背包允许的范围内，小偷最多能偷到多少钱？

5、动态规划

- 例：背包问题：给定n种物品和一背包。物品i的重量是wi，其价值为vi，背包的容量为C。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?
- 例如：物品共5个，物品重量分别为2kg，2kg，6kg，5kg，4kg，物品价值分别为6元，3元，5元，4元，6元，背包的最大容量为10kg。列出背包容量为j（1<=j<=10）的情况下，前i（1<=i<=5）件物品最大价值的动态规划过程。



5、动态规划

i	物品		背包容量										j
	重量 W[i]	价值 V[i]	1	2	3	4	5	6	7	8	9	10	
1	2	6	0	6	6	6	6	6	6	6	6	6	
2	2	3	0	6	6	9	9	9	9	9	9	9	
3	6	5	0	6	6	9	9	9	9	11	11	14	
4	5	4	0	6	6	9	9	9	10	11	13	14	
5	4	6	0	6	6	9	9	12	12	15	15	15	

背包容量为j时选入前i件物品获得的最大价值

5、动态规划

- 根据分析过程，得出规划过程中的两种情况：
  - （1）若背包容量j<当前要加入的物品i的重量W[i]时，最大价值与W[i-1]保持一致。

物品		背包容量j									
重量 W[i]	价值 V[i]	1	2	3	4	5	6	7	8	9	10
2	6	0	6	6	6	6	6	6	6	6	6
2	3	0	6	6	9	9	9	9	9	9	9
6	5	0	6	6	9	9	9	9	11	11	14
5	4	0	6	6	9	9	9	10	11	13	14
4	6	0	6	6	9	9	12	12	15	15	15

5、动态规划

- 根据分析过程，得出规划过程中的两种情况：
  - （2）当背包容量j>=当前要加入的物品i的重量W[i]时，则判断加入i物品后的价值与不加入的价值，取大者。

物品		背包容量j									
重量 W[i]	价值 V[i]	1	2	3	4	5					
2	6	0	6	6	6	6					
2	3	0	6	6	9	9					
6	5	0	6	6	9	9	9	9	11	11	14
5	4	0	6	6	9	9	9	10	11	13	14
4	6	0	6	6	9	9	12	12	15	15	15

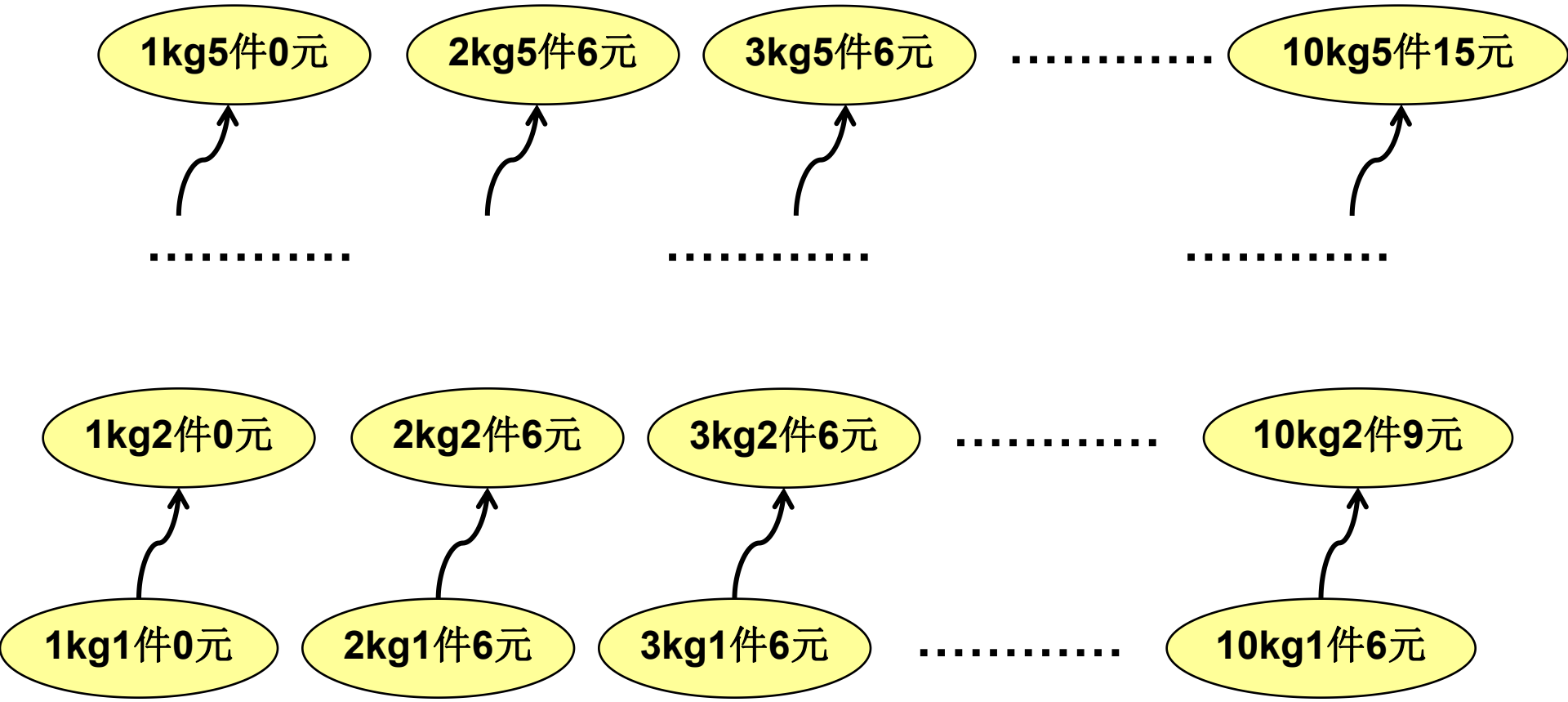
当背包容量为j-w[i]=4-2时取前i-1件物品的最大价值为6，
6+V[i]=6+3=9,
9>当背包容量为4时取前i-1件物品的最大价值6，则最大价值取9

5、动态规划

- 根据分析过程，得出规划过程中的两种情况：
  - （2）当背包容量j>=当前要加入的物品i的重量W[i]时，则判断加入i物品后的价值与不加入的价值，取大者。

物品		背包容量j									
重量 W[i]	价值 V[i]	1	2	3	4	5	6	7	8	9	10
2	6	0	6							6	6
2	3	0	6							9	9
6	5	0	6							11	14
5	4	0	6	6	9	9	9		11	13	14
4	6	0	6	6	9	9	12	12	15	15	15

当背包容量为j-w[i]=8-4时取前i-1件物品的最大价值为9，
9+V[i]=9+6=15,
15>当背包容量为8时取前i-1件物品的最大价值11，则最大价值取15



- 动态规划=贪婪策略+递推(降阶)+存储递推结果
- 把求解的问题分成许多阶段或多个子问题，最后一个子问题就是初始问题的解。

5、动态规划

- 算法实现：

```
#define N 5  //物品种类
#define C 10 //背包能容纳的总重量
main()
{
    int V[N+1]={0,6,3,5,4,6}; // 每种物品价值
    int W[N+1]={0,2,2,6,5,4}; // 每种物品重量
    int a[N+1][C+1]={0};      // a[i][j]表示在背包容量为j的情况下，前i件物品的最大价值
    int i=1;
    int j=1;
    int x,y;
```

5、动态规划

```
    for(i=1;i<=N;i++)
    {
        for(j=1;j<=C;j++)
        {
            if(j<W[i])
                a[i][j]=a[i-1][j];
            else
            {
                x=a[i-1][j];  //不取当前物品
                y=a[i-1][j-W[i]]+V[i];  //取当前物品
                a[i][j]=x<y?y:x;
            }
            printf("%-4d",a[i][j]);
        }
        printf("\n");
    }
}
```