算法设计

——动态规划

陈卫东

chenwd@scnu.edu.cn

华南师范大学计算机学院

2021-10

动态规划

- 动态规划的基本模式
  - 5.1 引言
  - 5.2 带权的区间调度问题
  - 5.3 动态规划的基本模式
- 应用
  - 5.4 分段的最小二乘法
  - 5.5 0-1背包问题
  - 5.6 所有点对间的最短路问题

5.1 引言

- 【例1】 计算Fibonacci数列的第n项。

该数列的定义为：

$$f(n)=\begin{cases}1 & \text{若 } n=1 \text{ 或 } n=2 \\ f(n-1)+f(n-2) & \text{若 } n\geqslant 3\end{cases}$$

- **算法1（直接递归法）**

  根据上述定义可导出如下递归算法:

```
int  F(int n)
{  //输入为正整数n
   if (n==1||n==2) return 1;
   return F(n-1)+F(n-2);
}
```

- 算法特点：子问题的求解有大量的重复计算。

  (图示$F(7)$)

- **算法1（直接递归法）**

  - 时间复杂度分析:

$$T(n)=\begin{cases}0, & \text{若 } n=1 \text{ 或 } n=2; \\ T(n-1)+T(n-2)+1, & \text{若 } n\geq 3。\end{cases}$$

  - $T(n)=\Theta(((1+5^{1/2})/2)^n)=O(1.618^n)$

- **算法2（备忘录方法）**

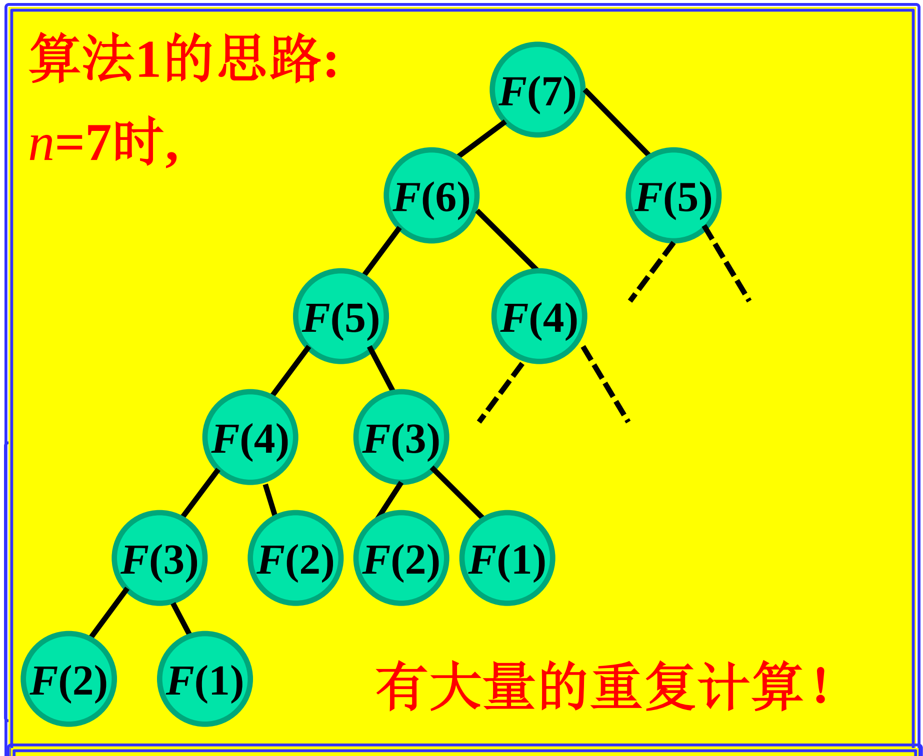
  - 算法特点：从上到下计算子问题，每个子问题只计算一次。
  - 时间复杂度：**O(*n*)** 。

- **算法3（动态规划法）**

  - 算法特点：从下到上计算子问题，每个子问题只计算一次。
  - 时间复杂度：**O(*n*)** 。

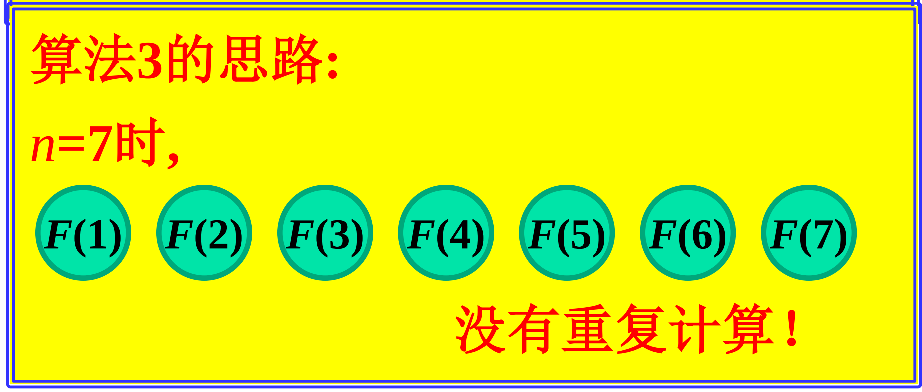
◆ 算法1（递归方法）

```
int  F (int n) //根据定义可导出递归算法
{
    if (n = =1||n = =2) return 1;
    return F(n-1)+F(n-2);
}
```



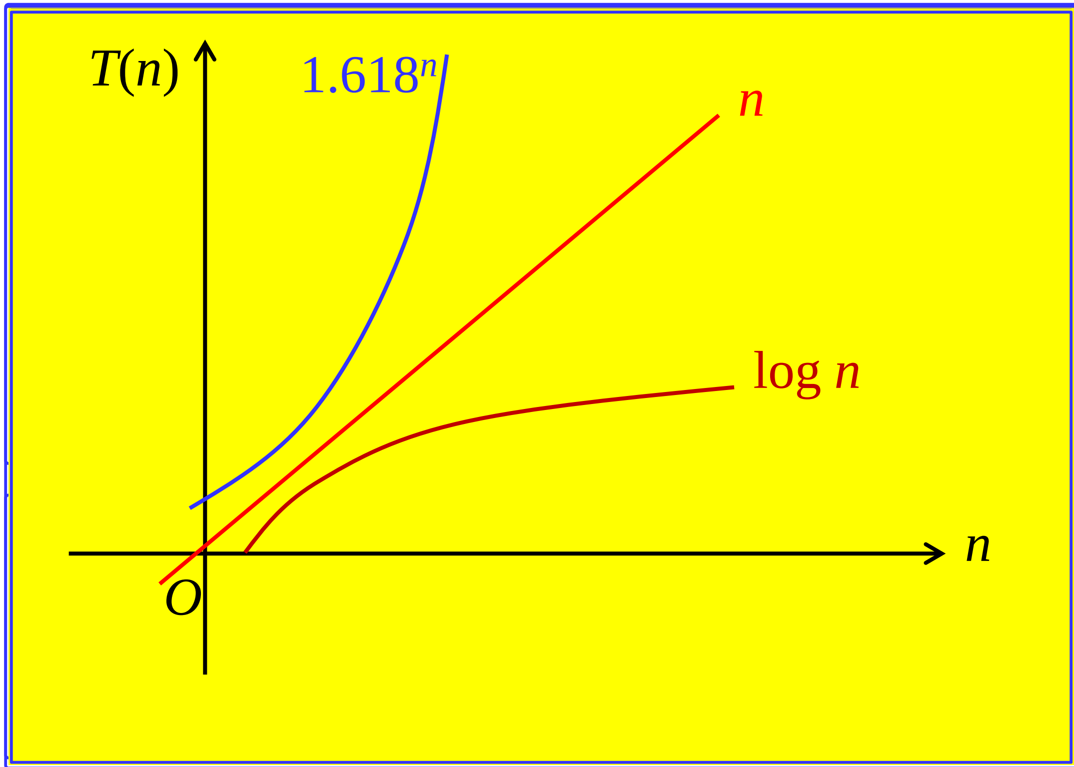
◆ 算法3（动态规划法）

```
int  F (int n)
{
    if( n = =1|| n = =2)  return 1;
    int  f1,  f2,  f;
     f1 = 1;  f2 = 1;
    for(int  i=3; i<=n; i++)  { f = f1+f2;  f1 = f2;  f2 = f; }
    return  f ;
}
```



◆ 算法1（递归方法）

```
int  F (int n)
{
    if (n = =1||n = =2) return 1;
    return F(n-1)+F(n-2);
}
```



◆ 算法3（动态规划法）

```
int  F (int n)
{
    if( n = =1|| n = =2)  return 1;
    int  f1,  f2,  f;
     f1 = 1;  f2 = 1;
    for(int  i=3; i<=n; i++)  { f = f1+f
    return  f ;
}
```

$1.618^{40} \approx 228633935$ ---约1秒

$1.618^{76} \approx 7627238257976215$ ---约1年

$1.618^{100} \approx 7904087286752943259 24$ ---约103630年

注：1年 = 365×24 ×3600 = 31536000 秒

✓ **思考：能否获得O(log*n*)**时间的算法?

- **算法4（O(log*n*)时间的算法）**

  算法特点：$(f(n),f(n-1))=\boldsymbol{B}(f(n-1),(f(n-2)))=\ldots.=\boldsymbol{B}^{n-2}(f(2),f(1))$，其中$\boldsymbol{B}$是一个2阶矩阵。既然$O(\log n)$时间可以计算$\boldsymbol{B}^{n-2}$，由此可得到一个时间复杂度为$\mathbf{O}(\log n)$的算法来计算$f(n)$。

◆ 算法4（O(log n)时间的算法）

基本思路:

根据 $f(n)$的定义，我们有

$$
\begin{aligned}
(f(n), f(n-1)) &= (f(n-1), f(n-2)) \times \boldsymbol{B} \\
&= (f(n-2), f(n-3)) \times \boldsymbol{B}^2 \\
&\cdots \\
&= (f(2), f(1)) \times \boldsymbol{B}^{n-2}
\end{aligned}
$$

，其中 $\boldsymbol{B} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

B^{n-2} 的计算方法:

n=8:	n=9:	n=10:	n=11:
$\boldsymbol{B}^2=\boldsymbol{B}\times\boldsymbol{B}$,	$\boldsymbol{B}^2=\boldsymbol{B}\times\boldsymbol{B}$,	$\boldsymbol{B}^2=\boldsymbol{B}\times\boldsymbol{B}$,	$\boldsymbol{B}^2=\boldsymbol{B}\times\boldsymbol{B}$
$\boldsymbol{B}^4=\boldsymbol{B}^2\times\boldsymbol{B}^2$	$\boldsymbol{B}^4=(\boldsymbol{B}^2)^2$	$\boldsymbol{B}^4=(\boldsymbol{B}^2)^2\times\boldsymbol{B}$	$\boldsymbol{B}^5=(\boldsymbol{B}^2)^2\times\boldsymbol{B}$
$\boldsymbol{B}^8=\boldsymbol{B}^4\times\boldsymbol{B}^4$	$\boldsymbol{B}^9=(\boldsymbol{B}^4)^2\times\boldsymbol{B}$	$\boldsymbol{B}^{10}=(\boldsymbol{B}^5)^2$	$\boldsymbol{B}^{11}=(\boldsymbol{B}^5)^2\times\boldsymbol{B}$

计算$\boldsymbol{B}^{n-2}$的时间为$O(\log n)$。由此可得运行时间为$O(\log n)$的算法计算$f(n)$。

- 【例2】计算二项式系数C_n^k。
  - 算法1（直接递归计算）

$$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}, \quad 0<k<n$$

$$C_n^0 = C_n^n = 1$$

- 算法特点：子问题的求解有大量的重复(图示C_{10}^5)
- 算法时间复杂度分析：

$$T(n,k)=T(n-1,k)+T(n-1,k-1)+1, \ 0<k<n;$$

$$T(n,0)=T(n,n)=0.$$

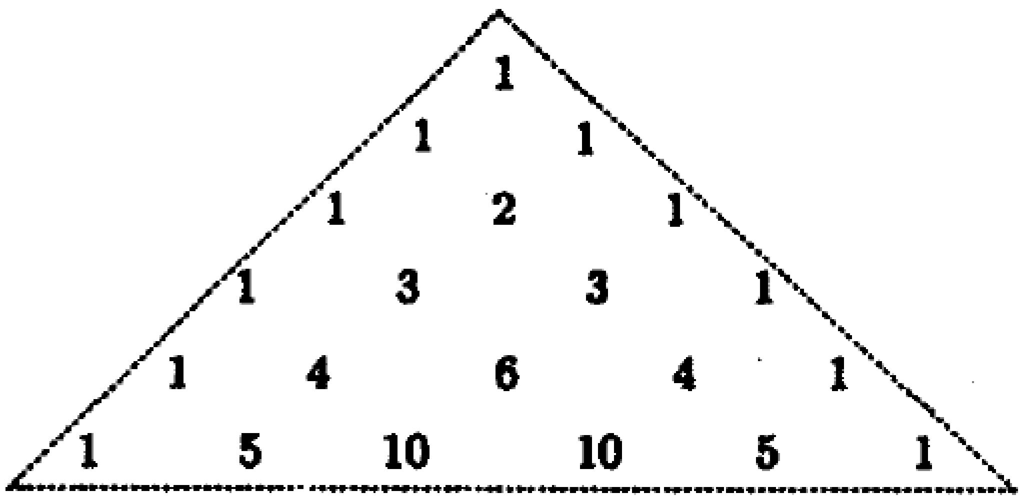
$T(n,k)$正比于C_n^k

$$C_n^{n/2} \geqslant \frac{2^n}{\sqrt{\pi n}}$$

- **算法2（动态规划法）**

➢ 算法特点：用帕斯卡三角形(杨辉三角形)从下到上来计算，每个子问题只算一次。

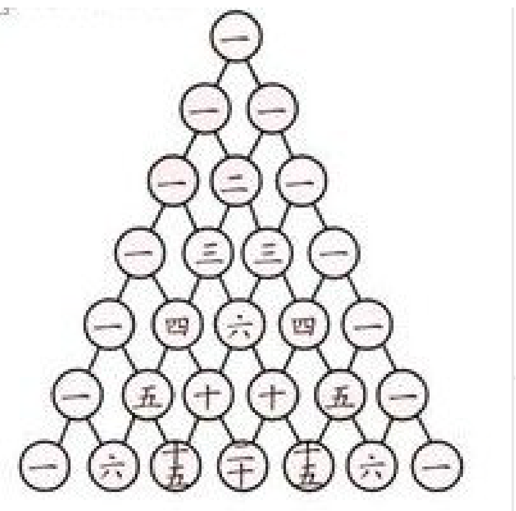
➢ 时间复杂度： **$O(n^2)$**



帕斯卡三角形的前 6 行

（百度百科:https://baike.so.com/doc/5391358-5628080.html）

杨辉三角形，又称**贾宪三角形**，**帕斯卡三角形**，是二项式系数在三角形中的一种几何排列。在我国南宋数学家杨辉所著的《详解九章算术》（1261年)一书中用如图的三角形解释二项和的乘方规律。



历史

北宋人贾宪约1050年首先使用“贾宪三角”进行高次开方运算。

杨辉，字谦光，南宋时期杭州人。在他1261年所著的《详解九章算法》一书中，辑录了如上所示的三角形数表，称之为“开方作法本源”图，并说明此表引自11世纪前半贾宪的《释锁算术》，并绘画了“古法七乘方图”。故此，杨辉三角又被称为“贾宪三角”。

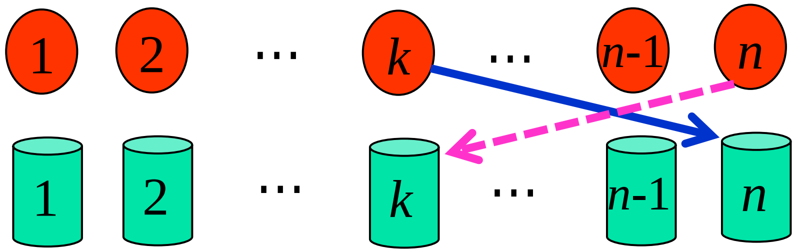
元朝数学家朱世杰在《四元玉鉴》（1303年）扩充了“贾宪三角”成“古法七乘方图”。

意大利人称之为“塔塔利亚三角形”（Triangolo di Tartaglia）以纪念在16世纪发现一元三次方程解的塔塔利亚。

在欧洲直到1623年以后，法国数学家帕斯卡在13岁时发现了“帕斯卡三角”。

布莱士·帕斯卡的著作Traité du triangle arithmétique（1655年）介绍了这个三角形。帕斯卡搜集了几个关于它的结果，并以此解决一些概率论上的问题，影响面广泛，Pierre Raymond de Montmort（1708年）和亚伯拉罕·棣·美弗（1730年）都用帕斯卡来称呼这个三角形。

近年来国外也逐渐承认这项成果属于中国，所以有些书上称这是“中国三角形”（Chinese triangle）



5.1 引言

- **思考题**
  - n只球和n个盒子，要求每个盒子里恰好放一只球且第k只球不能放入第k个盒子里($1 \le k \le n$)。试求一共有多少种不同的放法?

✓ **解题思路：**

设有$D(n)$中放法。则有

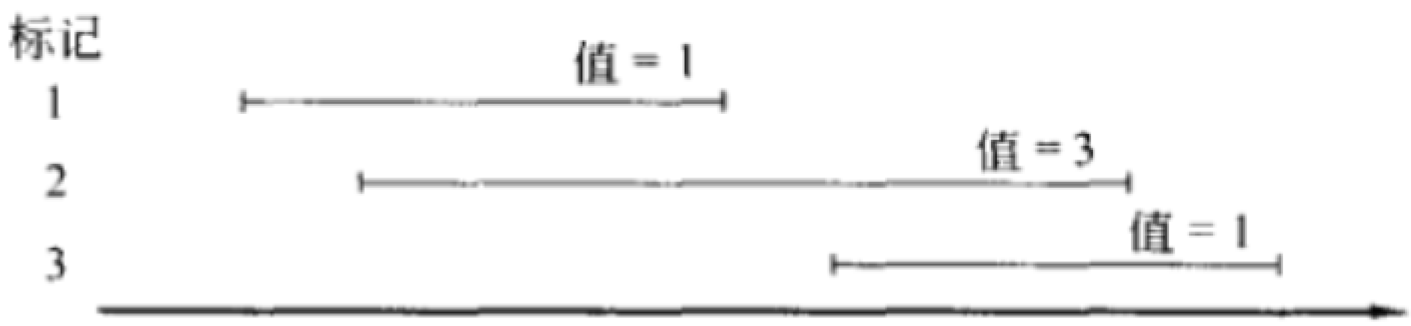
$$D(n)=(n-1)*(D(n-1)+D(n-2)),$$

其中，$D(1)=0$，$D(2)=1$。

然后使用动态规划法求解即可。

5.2 带权区间调度问题

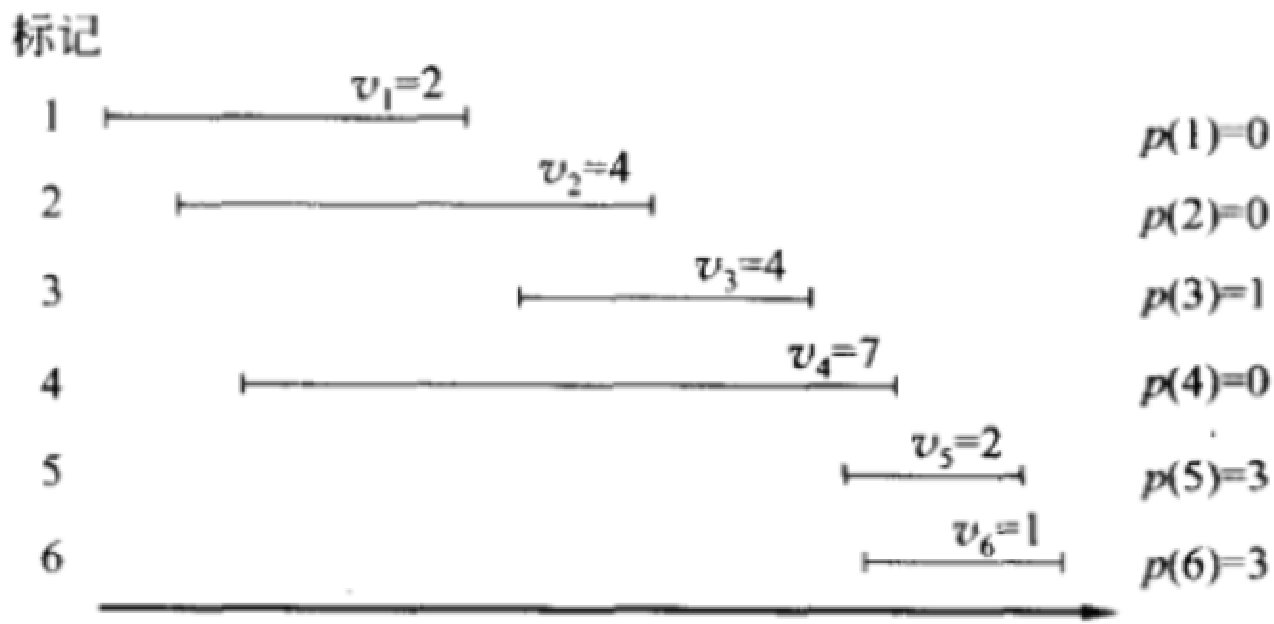
【问题】 带权区间调度问题如下：我们有 n 个需求，标记为 $1,2,\cdots,n$，每个需求指定一个开始时间 s_i 与一个结束时间 f_i. 每个区间 i 现在还有一个值或者权 v_i. 如果两个区间不重叠，那么它们是相容的. 当前问题的目标是选择一个两两相容的区间的子集 $S\subseteq\{1,\cdots,n\}$，以使得所选区间的值之和 $\sum_{i\in S} v_i$ 最大.



一个带权区间调度的简单实例

动态规划算法

让我们假设需求按照结束时间非降的次序排列：$f_1 \leqslant f_2 \leqslant \cdots \leqslant f_n$. 如果 $i<j$，我们说需求 i 在需求 j 之前来到. 这将是自然的从左到右的次序，我们将按照这个次序考虑区间. 为了有助于提及这个次序，我们对区间 j 定义 $p(j)$ 为使得区间 i 与 j 不相交的最大的标记 $i<j$. 换句话说，i 是最右边的在 j 开始之前结束的区间. 如果没有需求 $i<j$ 与 j 不相交，那么我们定义 $p(j)=0$. $p(j)$ 定义的一个例子如下图所示.



对于每个区间 j 具有被定义函数 $p(j)$ 的一个带权区间调度的实例

动态规划算法

所有这些都使人想到求区间$\{1,2,\cdots,n\}$上的最优解包括查看形如$\{1,2,\cdots,j\}$的较小问题的最优解。于是，对于任意在1与n之间的j值，令O_j表示对于由需求$\{1,\cdots,j\}$所组成问题的最优解，并且令$\mathrm{OPT}(j)$表示这个解的值(基于对于区间的空集上的最优解的约定，我们定义$\mathrm{OPT}(0)=0$)。我们正在寻找的最优解正好是O_n，具有值$\mathrm{OPT}(n)$。对于$\{1,2,\cdots,j\}$上的最优解O_j，我们上面的推理(从$j=n$的情况加以推广)说，或者$j\in O_j$，在这种情况下，$\mathrm{OPT}(j)=v_j+\mathrm{OPT}(p(j))$，或者$j\notin O_j$，在这种情况下，$\mathrm{OPT}(j)=\mathrm{OPT}(j-1)$。由于这些正好是两种可能的选择($j\in O_j$或$j\notin O_j$)，我们可以进一步说：

定理 6.1 $\mathrm{OPT}(j)=\max(v_j+\mathrm{OPT}(p(j)),\mathrm{OPT}(j-1))$.

我们怎样决定j是否属于最优解O_j？这也是容易的：它属于最优解当且仅当上述第一个最优至少与第二个一样好；换句话说，

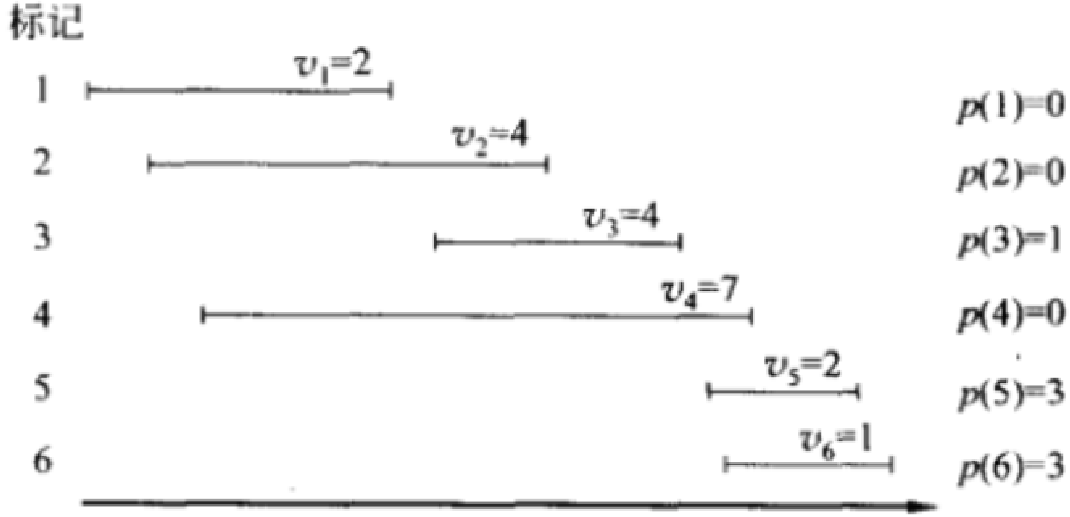
定理 6.2 需求j属于集合$\{1,2,\cdots,j\}$上的最优解当且仅当

$$v_j+\mathrm{OPT}(p(j))\geqslant \mathrm{OPT}(j-1).$$

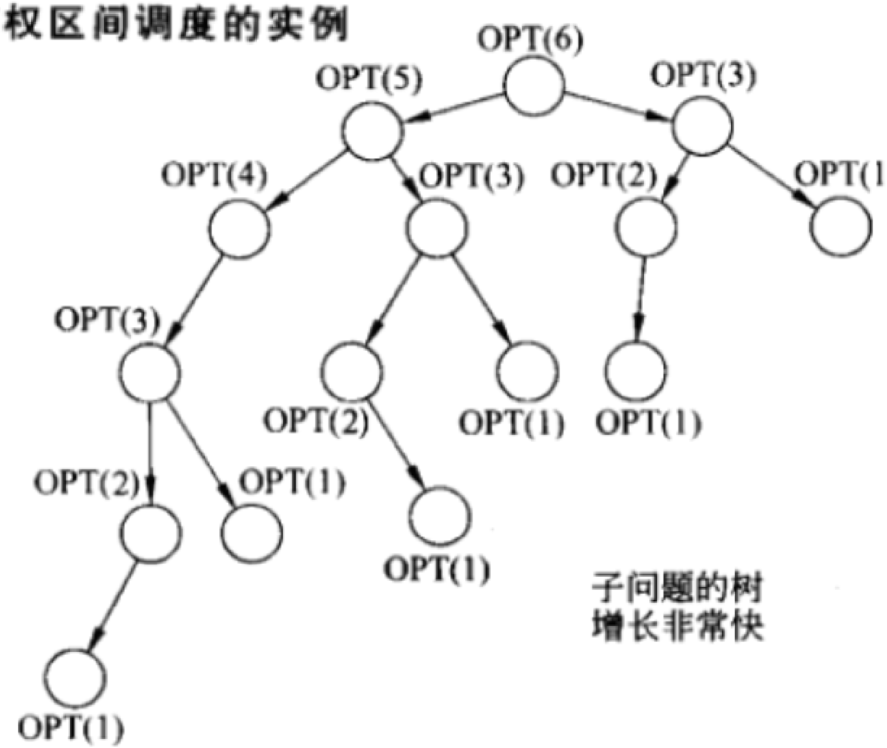
◆ 算法1（简单递归形式）

```
Compute-Opt(j)
  If j=0 then
    返回 0
  Else
    返回 max(v_j + Compute-Opt(p(j)), Compute-Opt(j-1))
  Endif
```

➢ 该算法具有指数型复杂度。



对于每个区间 j 具有被定义函数 $p(j)$ 的一个带权区间调度的实例



由 Compute-Opt 在 上图 的问题实例上所调用的子问题树

◆ 算法2（递归的备忘录形式）——O(*n*)时间

我们怎样能删除所有这些冗余？我们可以在第一次计算它时就把 Compute-Opt 值存在一个大家都可访问的地方，然后在所有后来的递归调用中只是使用这些预先算好的值. 这个存储已算好的值的技术叫做备忘录.

我们用一个更“聪明”的过程 M-Compute-Opt 改进上述策略. 这个过程将用到一个数组 $M[0\cdots n]$；开始 $M[j]$将具有值“空”，但是一旦 Compute-Opt(j)的值被确定，就把它保留起来. 为确定 OPT(n)，我们调用 M-Compute-Opt(n).

```
M-Compute-Opt(j)
  If j=0 then
    返回 0
  Else if M[j]不空 then
    返回 M[j]
  Else
    定义 M[j]=max(v_j+M-Compute-Opt(p(j)),M-Compute-Opt(j-1))
    返回 M[j]
  Endif
```

◆ 算法2（递归的备忘录形式）——O(n)时间

我们从定理 6.2 知道 j 属于一个关于区间集合 $\{1,\cdots,j\}$ 的最优解当且仅当 $v_j+\text{OPT}(p(j))\geqslant \text{OPT}(j-1)$. 使用这个观察，我们得到下面的简单过程，它通过数组 M“反向追踪”来找出在最优解中的区间集合.

```
Find-Solution(j)
  If j=0 then
    什么也不输出
  Else
    If v_j + M[p(j)] ≥ M[j-1] then
      输出 j 与 Find-Solution(p(j))的结果
    Else
      输出 Find-Solution(j-1)的结果
    Endif
  Endif
```

由于 Find-Solution 只在确实更小的值上调用自己，它总的产生 $O(n)$ 次递归调用；并且由于它每次调用用常数时间，我们有：

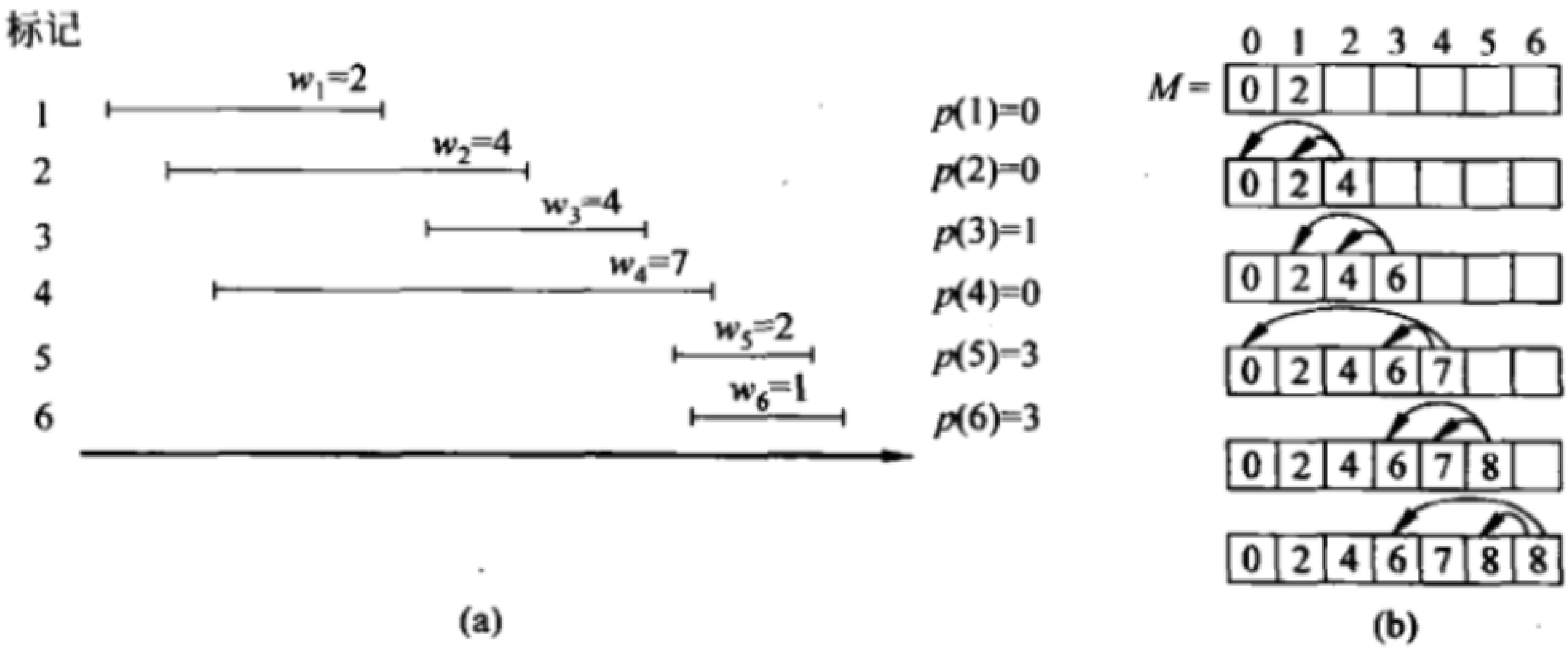
定理 6.5 给定这些子问题的最优值的数组 M，Find-Solution 在 $O(n)$ 时间内返回一个最优解.

◆ 算法3（动态规划迭代形式）——O(n)时间

那么实现的关键就是我们可以不使用备忘录式的递归，而通过迭代算法直接计算 M 中的项. 我们就从 $M[0]=0$ 开始，并且保持 j 增长；每一次我们需要确定一个值 $M[j]$，就由定理 6.1 提供答案. 算法看起来如下.

```
Iterative-Compute-Opt
M[0]=0
For j=1,2,…,n
   M[j]=max(v_j+M[p(j)],M[j-1])
Endfor
```

◆ 算法3（动态规划迭代形式）——O(n)时间



(b)部分说明了部分(a)中所描述的带权区间调度的样本实例上的 Iterative-Compute-Opt 迭代

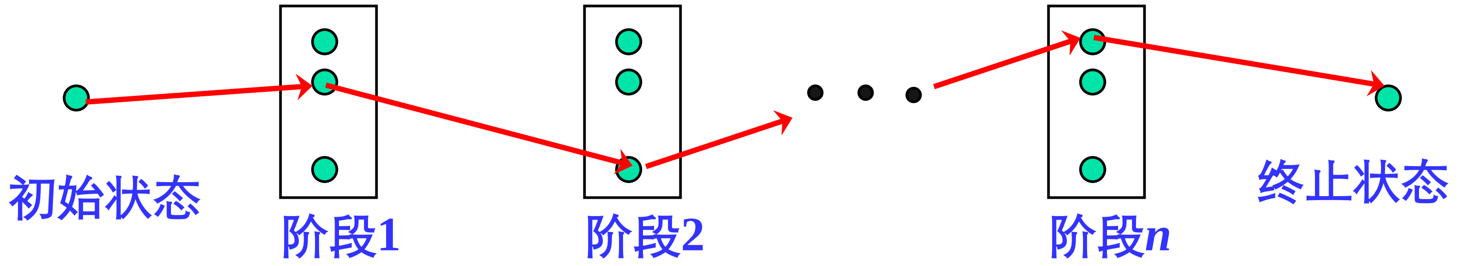
5.3 动态规划的基本模式

- 适用范围
  - 多阶段决策的最优化问题
  - 最优解满足最优性原理
  - 子问题的重叠性
- 基本思想
- 设计一个动态规划算法有四个步骤

5.3 动态规划的基本模式

- **适用问题**
  - **多阶段决策的最优化问题**

问题的求解过程分阶段来完成，在每阶段需要做出一个决策，形成一个决策序列。每个决策序列对应一个目标函数值。要求求出目标值最优（最大或最小）的决策序列，即最优决策序列（最优解），所对应的目标值为最优值。



5.3 动态规划的基本模式

- **最优性原理**

  当问题的最优解包含有子问题的最优解时，称该问题满足最优性原理。

  因此，问题的最优解可在其子问题的最优解中寻找。这就决定了计算过程是：首先将问题分解为子问题，求得子问题的最优解，由此再构造得到原问题的最优解。

- **子问题重叠性**

  原问题的子问题中由于可能有大量重复的子问题，而相同的子问题只求解一次，因而其效率往往高于枚举法。且子问题重复的越多，其效率越高。

5.3 动态规划的基本模式

- **动态规划法的基本思想**

　　将原问题转化为若干个子问题来解决。但是每个子问题只计算一次。因此，一般需要将子问题的解保存起来以避免重复计算。

　　对所考虑的每个子问题都求出最优解，然后由子问题的最优解递推地构造原问题的最优解。

　　但是，在求解过程中由于需要将子问题的解保存下来以备将来使用，所以该方法往往需要较多的附加空间。

5.3 动态规划的基本模式

- **动态规划法的基本步骤**

  1. 证明满足最优性原理

     实际上即是说明大问题的最优解可从子问题的最优解答中找。这就决定了计算是从下而上地进行根据子问题的最优解逐步构造出整个问题的最优解的过程。

  2. 递归定义最优解的值

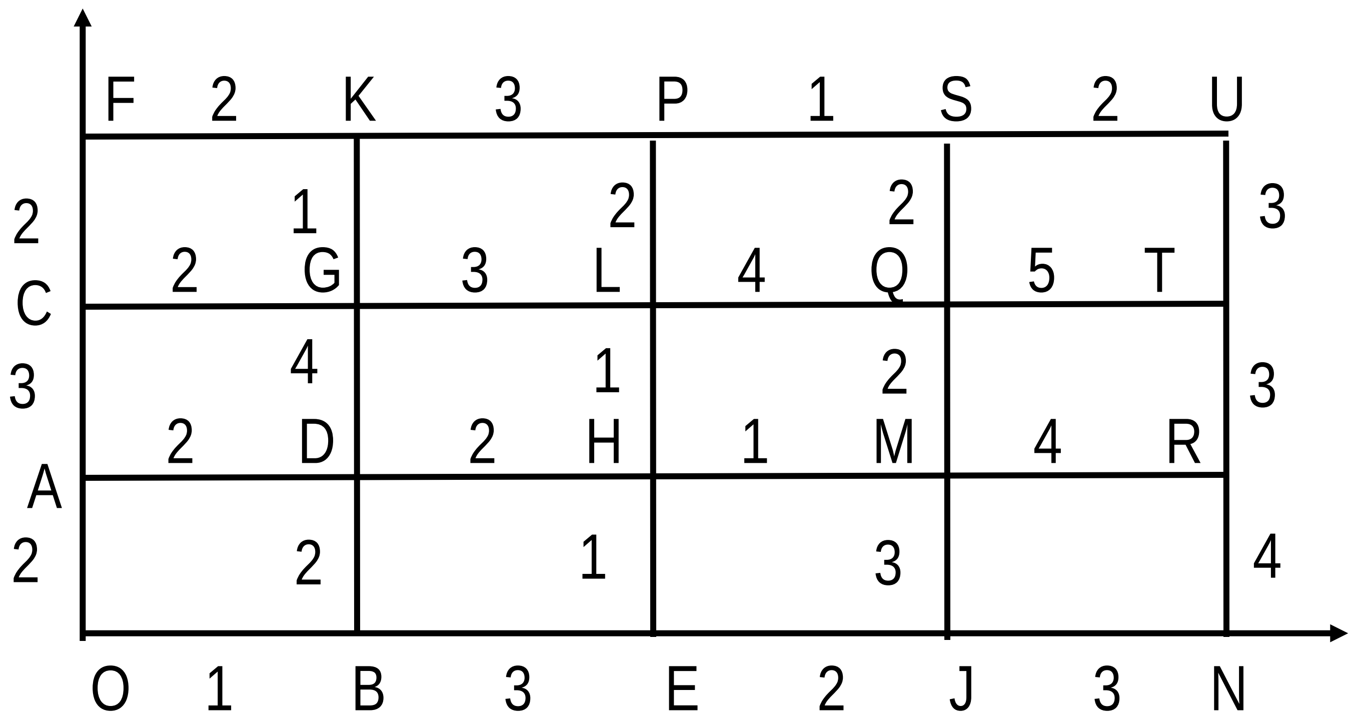
     即找出如何由子问题的最优解得到原问题的最优解的关系式。

  3. 按自下而上的方式计算最优解的值
  4. 由计算的结果构造出一个最优解

多段图问题

【问题实例】

考虑下面的有向图，水平边的方向为自左向右，纵向边的方向为自下而上，边上的数值为边的长度（权值）。问题：求从O到U点的最短路径及其长度。



多段图问题

- **穷举法**

从O到U点的每一条路径都要走4个横边和3个纵边。因而一共要考虑C_7^3=35条不同路径，计算每一条路经长度要6次加法。用穷举法一共要210次加法和34次比较。

- **动态规划法**
  - **求解该问题可看成是一个多阶段决策过程**

  第i阶段确定由从O经过 i步能到达的结点，i=1,2,3,…,7。

  可表示如下：

  $O \Rightarrow \{A,B\} \Rightarrow \{C,D,E\} \Rightarrow \{F,G,H,J\} \Rightarrow \{K,L,M,N\}$

  $\Rightarrow \{P,Q,R\} \Rightarrow \{S,T\} \Rightarrow U$
  - **验证满足最优性原理**

多段图问题

- **列出递推进行计算公式**

用dist(X)表示由O到达结点X的最短路径的长度，该路径上X结点的前一结点用pred(X)来表示。假定X是由O经过i步能到达的结点，则递推计算公式如下：

1）当i>1时，

dist(X) = ***min***{dist(Y)+YX|Y是由O经过i-1步能到达的任一结点}

pred(X)=上式取最小值的那个Y。

2）当i=1时，dist(X)=边OX长度，pred(X)=O。

dist(A)=2, pred(A)=O; dist(B)=1, pred(B)=O

dist(C)=dist(A)+3=5, pred(C)=A;

dist(E)=dist(B)+3=4, pred(E)=B;

dist(D)=min{dist(A)+2, dist(B)+2}=3, pred(D)=B;

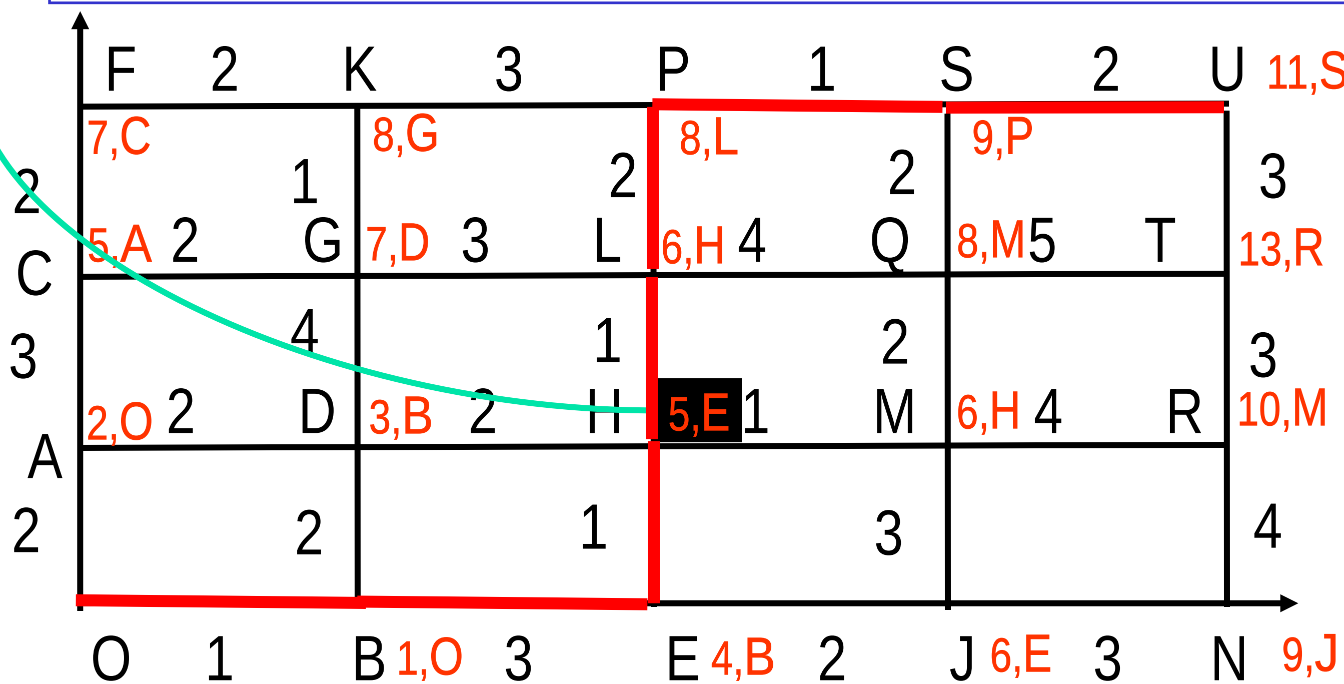
dist(H)=min{dist(E)+1, dist(D)+2}=5, pred(H)=E;

⋮ ⋮ ⋮

dist(S)=min{dist(P)+1, dist(Q)+2}=9, pred(S)=P;

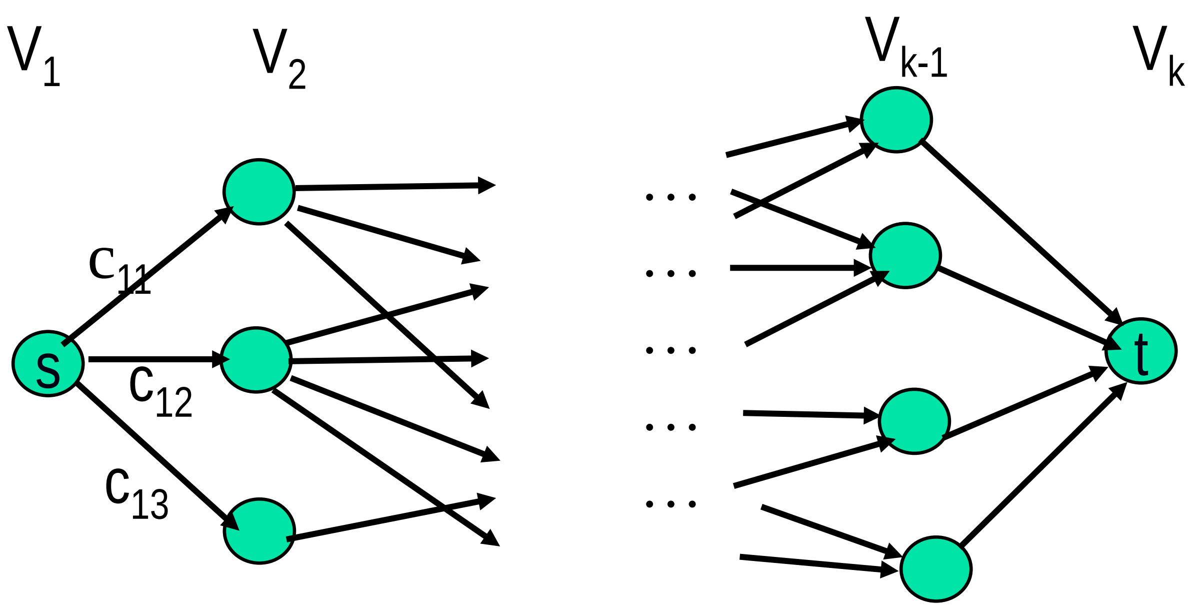
dist(T)=min{dist(Q)+5, dist(R)+3}=13, pred(T)=R;

dist(U)=min{dist(S)+2, dist(T)+3}=11, pred(U)=S;

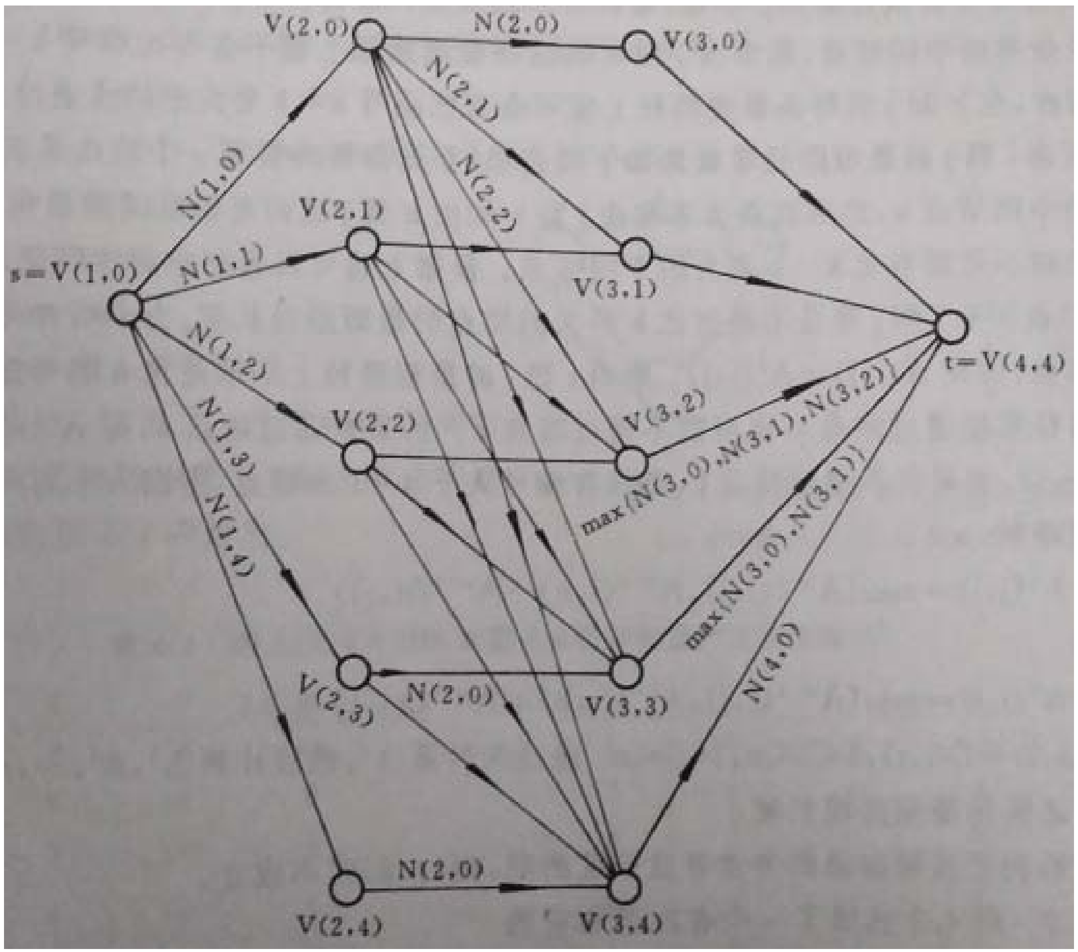


多段图问题

上述问题实质上即为多段图问题（见下图）。用动态规划法能高效地求解多段图问题。还有其它的一些问题，比如**资源分配问题**等均可转化为多段图问题，然后再用动态规划法来求解。



【资源分配问题】某厂根据计划安排，拟将n台相同的设备分配给m个车间，各车间获得这种设备后，可以为国家提供盈利$N[i,j]$(j台设备提供给i号车间将得到的利润，$1 \le i \le n$，$1 \le j \le m$)。问如何分配，才使国家得到最大的盈利？



给3个车间分配4台设备的多段图，最优分配方案由s到t的一条最大成本路径确定。

- **动态规划算法**

  设$C[i,j]$表示j台设备分配给$1,2,\ldots,i$号车间将得到的利润。则

  $C[i,0]=0;$

  $C[0,j]=0;$

  若$i>0$且$j>0$，则有

  $C[i,j]=\mathbf{max}\{C[i-1,j-k]+N[i,k] \mid 1 \le k \le j\}.$

  特别地，

  $C[m,n]=\mathbf{max}\{C[m-1,n-k]+N[m,k] \mid 1 \le k \le n\}.$

5.4 分段的最小二乘法

【问题】

当看绘制在一组二维数轴上的科学或统计数据时，人们常常试图用一条“最佳拟合”的线穿过这些数据，如图 6.6 所示.

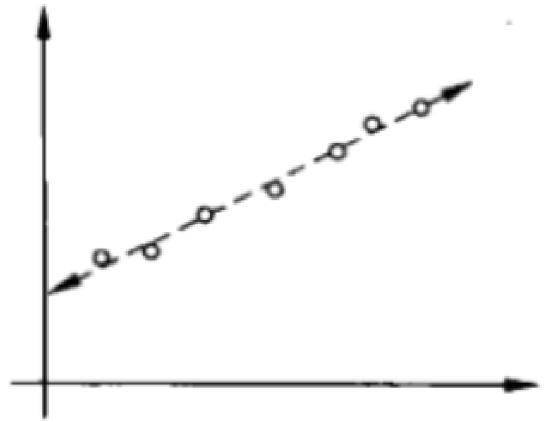


图 6.6　一条“最佳拟合”的线

5.4 分段的最小二乘法

这是统计学与数值分析中的基本问题，形式化如下．假设我们的数据由平面上的 n 个点的集合 P 组成，表示为 $(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)$；并且假设 $x_1<x_2<\cdots<x_n$．给定由方程 $y=ax+b$ 定义的直线 L，我们说 L 相对于 P 的误差是它对于 P 中的点的"距离"的平方和：

$$\mathrm{Error}(L,P)=\sum_{i=1}^{n}(y_i-ax_i-b)^2$$

那么一个自然的目标是找到具有最小误差的直线；这原来有一个好的近似解，它可以很容易地用计算导出．这里跳过推导过程，我们只叙述结果：最小误差的直线是 $y=ax+b$，其中：

$$a=\frac{n\sum_i x_iy_i-\left(\sum_i x_i\right)\left(\sum_i y_i\right)}{n\sum_i x_i^2-\left(\sum_i x_i\right)^2}\quad 和 \quad b=\frac{\sum_i y_i-a\sum_i x_i}{n}$$

5.4 分段的最小二乘法

现在，这里有一类问题，这些公式不是为覆盖它们而设计的. 我们有的数据常常看起来像图 6.7 中的情况. 在这种情况下，我们希望有类似下面的一个陈述：“这些点大致位于连续的两条直线上.”我们怎样把这个概念形式化？

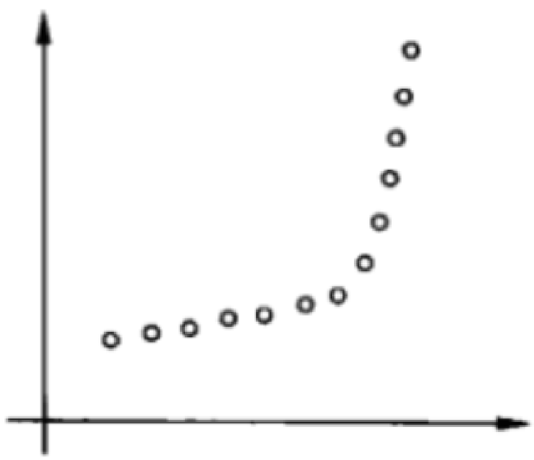


图 6.7　一组近似位于两条线上的点

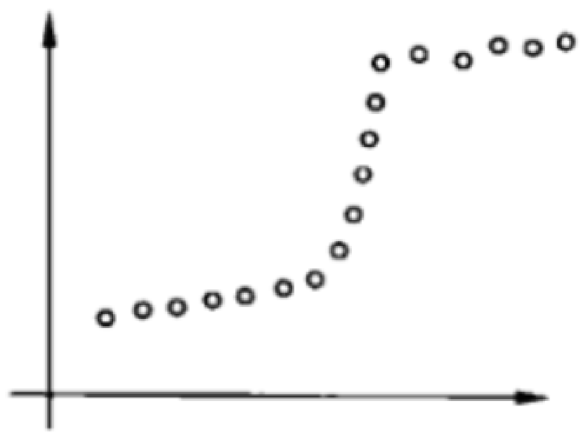


图 6.8　一组近似位于三条线上的点

问题形式化 如上面讨论的，给定一组点 $P=\{(x_1,y_1),(x_2,y_2),\cdots,(x_n,y_n)\}$，具有 $x_1<x_2<\cdots<x_n$. 我们将使用 P_i 表示点 (x_i,y_i). 我们必须首先把 P 划分成若干段. 每段是 P 的一个子集，它表示连续的 x 坐标的集合；即对某个下标 $i\leqslant j$，它是一个形如 $\{p_i,p_{i+1},\cdots,p_{j-1},p_j\}$ 的子集. 那么，对于在 P 的划分中的每段 S，我们按上面的公式计算相对于 S 中点的误差最小的直线.

一个划分的罚分定义为下面的项之和.

(i) 我们划分 P 的段数乘以一个给定的不变因子 $C>0$.

(ii) 对每段，通过那个段的最优直线的误差值.

我们在**分段最小二乘问题**的目标是找一个最小罚分的划分. 这个最小化抓住了我们前面所讨论的权衡问题. 我们可以考虑划分成任意多个段；由于增加了段数，我们将会减少定义(ii)部分中的罚分项，但是我们增加了(i)部分中的项(因子 C 由输入提供，并且通过调整 C 我们可以对多使用线的惩罚定到一个较大或较小的范围).

P 的划分有指数多种可能，初始对我们能有效找到最优的一个并不清楚. 我们现在显示怎样使用动态规划在 n 的多项式时间内找到一个最小罚分的划分.

动态规划算法

对于分段最小二乘，下面的观察是非常有用的：最后一个点 p_n 在最优划分中属于一段，并且那个段在某个比较早的点 p_i 开始。这是典型的能提出正确的子问题集合的观察：如果我们能挑出最后一段 $p_i,\cdots,p_n$（见图 6.9），那么我们可以从考虑中排出这些点并且递归地求解在剩下的点 $p_1,\cdots,p_{i-1}$ 上的问题。

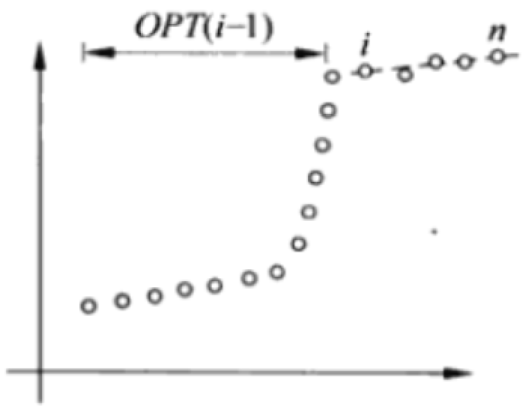


图 6.9　一个可行解：一条线段拟合点 $p_i, p_{i+1},\cdots,p_n$，然后对于剩下的点 $p_1,\cdots,p_{i-1}$ 找一个最优解

假设我们令 OPT(i) 表示对于点 $p_1,\cdots,p_i$ 的最优解，并且令 $e_{i,j}$ 表示关系到 $p_i, p_{i+1},\cdots,p_j$ 的任何直线的最小误差（作为边界情况我们写 OPT(0) = 0)。那么我们上述的观察可以叙述如下。

定理 6.6　如果最优划分的最后一段是 $p_i,\cdots,p_n$，那么最优解的值是：

$$\mathrm{OPT}(n) = e_{i,n} + C + \mathrm{OPT}(i-1).$$

动态规划算法

对于点 $p_1,\cdots,p_j$ 组成的子问题使用同样的观察，我们看到为得到 OPT(j)我们应该找到最好的方式产生最后的一段 $p_i,\cdots,p_j$——偿付误差加上关于这段的 C——加上对剩下的点的一个最优解 $OPT(i-1)$. 换句话说，我们已经证明了下面的递推式.

定理 6.7 对于点 $p_1,\cdots,p_j$ 上的子问题，

$$\mathrm{OPT}(j)=\min_{1\leqslant i\leqslant j}(e_{i,j}+C+\mathrm{OPT}(i-1))$$

并且段 $p_i,\cdots,p_j$ 被用于一个关于子问题的最优解，当且仅当这个最小是使用下标 i 得到的.

在设计这个算法中的困难部分现在已经被我们克服了. 从现在起，我们只需按照 i 增加的次序建立解 OPT(i).

动态规划算法

```
Segmentd-Least-Squares(n)
  数组 M[0,…,n]
  置 M[0]=0
  For 所有的对 i≤j
    计算对于段 p_i,…,p_j 的最小二乘误差 e_{i,j}
  Endfor
  For j=1,2,…,n
    使用递推式 6.7 计算 M[j]
  Endfor
  返回 M[n]
```

动态规划算法

正如在带权区间调度算法一样，我们可以通过数组 M 向回追踪来计算最优的划分.

```
Find-Segments(j)
  If j=0 then
    不用输出
  Else
    找一个使得 e_{i,j}+C+M[i-1]最小的 i
    输出这个段{p_i,…,p_j}以及 Find-Segments(i-1)的结果
  Endif
```

算法复杂度

最后我们考虑 Segmented-Least-Squares 的运行时间. 首先我们需要计算所有最小二乘的误差 $e_{i,j}$ 的值. 为了对它的运行时间实行简单的核算，我们注意到有 $O(n^2)$ 个对 (i,j) 需要这个计算；并且对每个对 (i,j)，我们可以在 $O(n)$ 时间内使用这一节开始给出的公式来计算

$e_{i,j}$. 于是计算所有的 $e_{i,j}$ 值的总运行时间是 $O(n^3)$.

此后算法对于值 $j=1,\cdots,n$ 有 n 次迭代. 对每个 j 的值，我们必须确定在递归式 6.7 中的最小值来填入数组 $M[j]$；这对每个 j 要用 $O(n)$ 时间，总计 $O(n^2)$ 时间. 于是，一旦所有的 $e_{i,j}$ 值被确定以后，运行时间是 $O(n^2)$ 时间①.

5.5 0-1背包问题

- 【问题】

设 $U=\{u_1, u_2, \ldots, u_n\}$ 是n个待放入容量为C的背包的物品集. 任意i: $n \geq i \geq 1$, 设s_i和v_i分别是第i种物品的重量和价值。每个物品要么整个放入背包，要么不放。且已知物品i放入背包能产生v_i的价值($n \geq i \geq 1$)。其中C和 s_i、v_i（$n \geq \geq 1$）均为正整数。

如何装包才能获得最大价值?

- **动态规划算法**
  - 设 $V[i,j]$表示从 $\{u_1, u_2, \dots, u_i\}$挑选物件装入容量为j的背包中的最优装包的价值。则

    $V[i,0]=0$

    $V[0,j]=0$

    若 $i>0$且$j<s_i$， $V[i,j]=V[i-1,j]$

    若 $i>0$且$j\geq s_i$, $V[i,j]=\boldsymbol{max}\{V[i-1,j], V[i-1,j-s_i]+v_i\}$

- **【实例】**

设0-1背包问题中n=4,C=9，n个物品的重量分别为{2,3,4,5}，其价值分别{3,4,5,7}。使用动态规划法求解最优装包方法的过程即是填写如下表格：

	0	1	2	3	4	5	6	7	8	9
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**0**	0	3	3	3	3	3	3	3	3
2	**0**	0	3	4	4	7	7	7	7	7
3	**0**	0	3	4	4	7	8	9	9	12
4	**0**	0	3	4	5	7	8	10	11	12

算法　　KNAPSACK

输入： 物品集合 $U=\{u_1,u_2,\cdots,u_n\}$，体积分别为 $s_1,s_2,\cdots,s_n$，价值分别为 $v_1,v_2,\cdots,v_n$，容量为 C 的背包。

输出： $\sum_{u_i \in S} v_i$ 的最大总价值，且满足 $\sum_{u_i \in S} s_i \leqslant C$，其中 $S \subseteq U$。

```
for i ← 0 to n
    V[i,0] ← 0
end for
for j ← 0 to C
    V[0,j] ← 0
end for
for i ← 1 to n
    for j ← 1 to C
        V[i,j] ← V[i-1,j]
        if s_i ≤ j then V[i,j] ← max{V[i,j], V[i-1,j-s_i] + v_i}
    end for
end for
return V[n,C]
```

- **算法复杂度**
  - 时间复杂度: $\Theta(nC)$
  - 空间复杂度: $\Theta(C)$

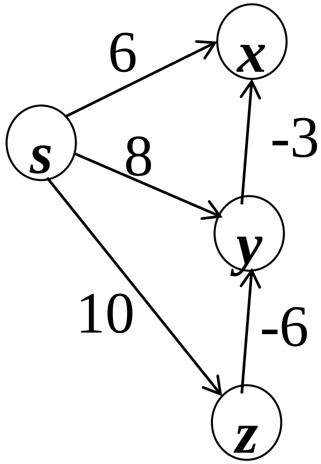
定理 背包问题的最优解能够在 $\Theta(nC)$ 的时间内和 $\Theta(C)$ 的空间内得到。

5.6 所有结点对间的最短路问题

【单源最短路径问题】 设 $G=<V,E>$ 是一个有向图，图中每一条边都有一个非负长度。单源最短路径问题就是要求出从图中一定点s（称为源点）到其它各点的长度最短的路径。

- Dijkstra算法（贪心法）
  - 贪心准则

    按照从源点s到各点最短路径的长度由小到大依次构造s到各点的最短路径。



【扩展问题】

设$G=<V,E>$ 是一个有向图，图中每一条边都有一个长度(长度可能为负)，但图中不存在负长度的回路。s是图G中任意点，如何求从点s到其它所有点的最短路径?

——思路：动态规划法设计的算法（Bellman-Ford算法）

当图用邻接表表示时，时间复杂度为：$O(mn)$

当图用邻接矩阵表示时，时间复杂度为：$O(n^3)$

➢ Bellman-Ford算法（SPFA：一种快速实现）

设$L(v,k)$表示s到v且中间经过最多k条边的最短路径的长度。则对于任意点v ，

k=0时,

$$L(s,0)=0,\ L(v,0)=\infty\ (v \neq s\ );$$

k=1,2,…,n–1时,

$$L(v,\ k)=\mathbf{min}_{<x,v>\in E}\{L(v,k\text{-}1),L(x,k\text{-}1)+l(x,v)\}.$$

最后计算得出 $L(v,\ n-1)$即可.

➢ 算法分析

当图用邻接表表示时，时间复杂度为: $O(mn)$

当图用邻接矩阵表示时，时间复杂度为: $O(n^3)$

(如果图是有向无回路图(DAG)时，图中存在拓扑序列，此时可按照图的拓扑序列的次序来计算公式，时间复杂度可降为$O(m+n)$或$O(n^2)$)

5.6 所有结点对间的最短路问题

- 【问题】

设G=<V,E>是一个有向图，图中每条边<i,j>都有一个成本（长度）$l[i,j]$，若从结点i到结点j没有边，则令$l[i,j]=\infty$。要求找出从每个结点到所有其它结点的长度最短的路。

假定图中没有负长度的回路。

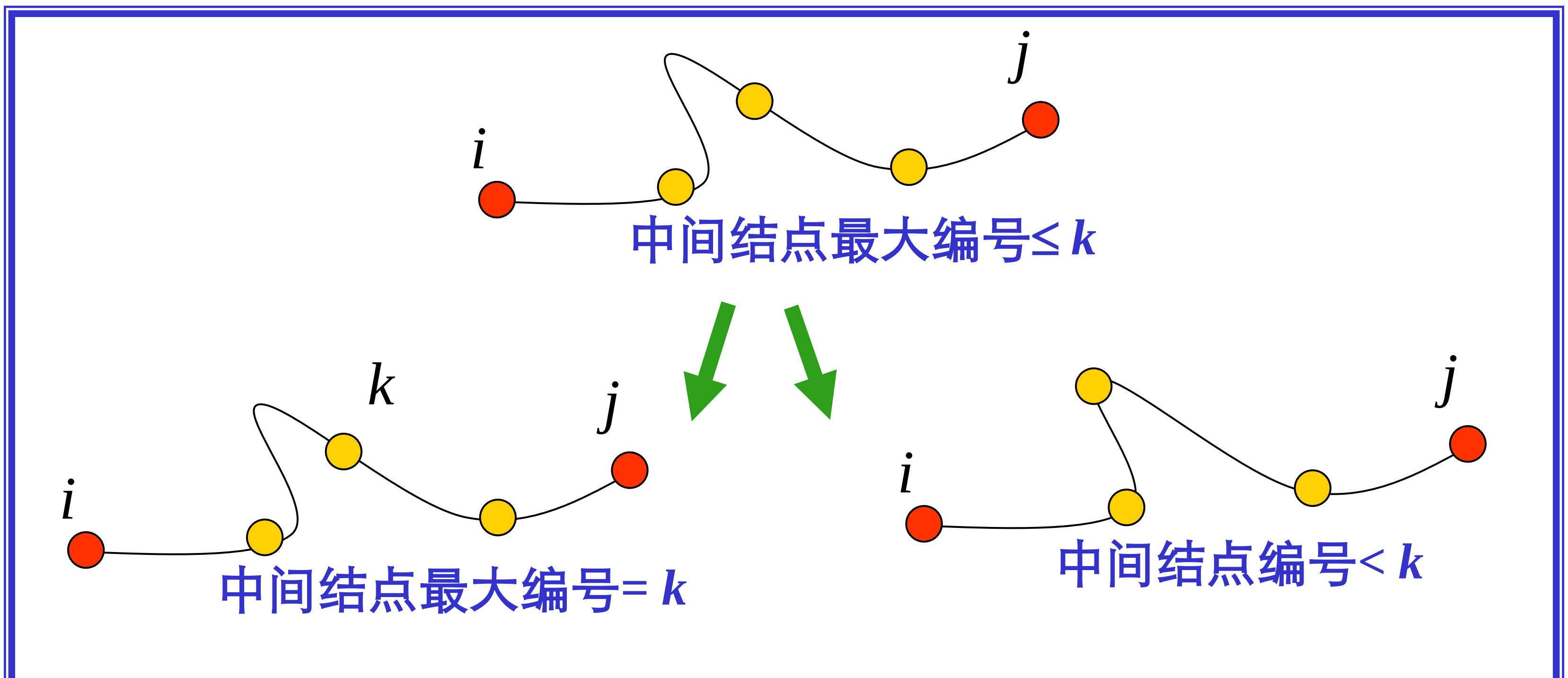
- **动态规划算法**
  - 设 $d^k(i,j)$表示从i到j的一条中间不经过比k大的结点的最短路径的长度。则

$$d^0(i,j) = l[i,j],$$

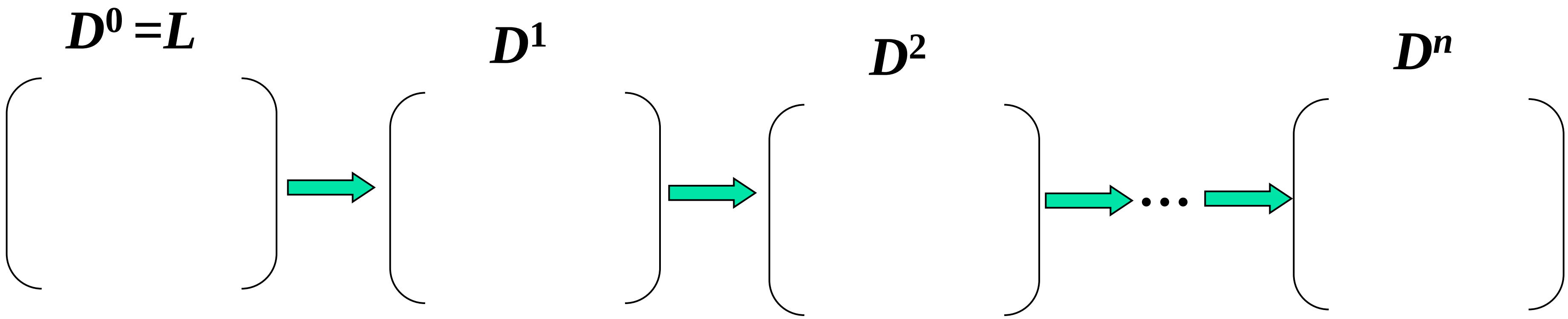
当$n\ge k\ge 1$时, $d^k(i,j) = \mathbf{min}\ \{d^{k-1}(i,j),\quad d^{k-1}(i,k)+d^{k-1}(k,j)\}$

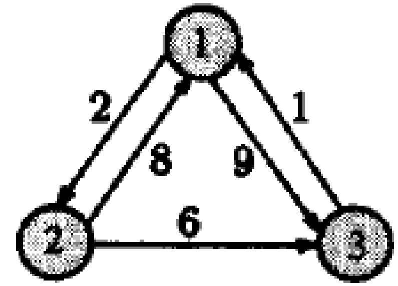
特别地，

$$d^n(i,j) = \mathbf{min}\ \{d^{n-1}(i,j),\quad d^{n-1}(i,n)+d^{n-1}(n,j)\}$$



- **动态规划算法**
  - 计算次序：需要n个矩阵





所有点对最短路径问题的实例

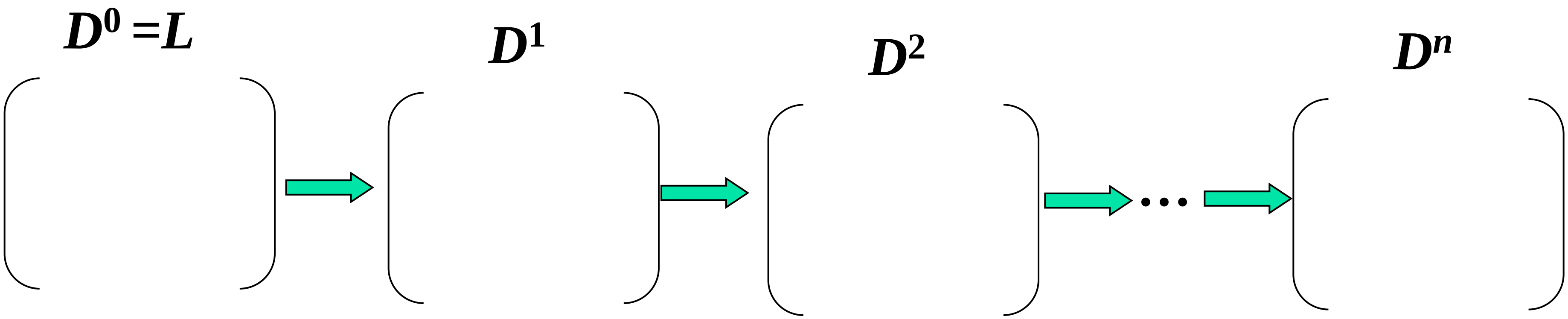
矩阵 D_0, D_1, D_2 和 D_3 是

$$D_0=\begin{bmatrix}0 & 2 & 9\\ 8 & 0 & 6\\ 1 & \infty & 6\end{bmatrix}\qquad D_1=\begin{bmatrix}0 & 2 & 9\\ 8 & 0 & 6\\ 1 & 3 & 0\end{bmatrix}$$

$$D_2=\begin{bmatrix}0 & 2 & 8\\ 8 & 0 & 6\\ 1 & 3 & 0\end{bmatrix}\qquad D_3=\begin{bmatrix}0 & 2 & 8\\ 7 & 0 & 6\\ 1 & 3 & 0\end{bmatrix}$$

最后计算的矩阵 D_3 存有所要求的距离。

- **动态规划算法**
  - 计算次序：需要n个矩阵

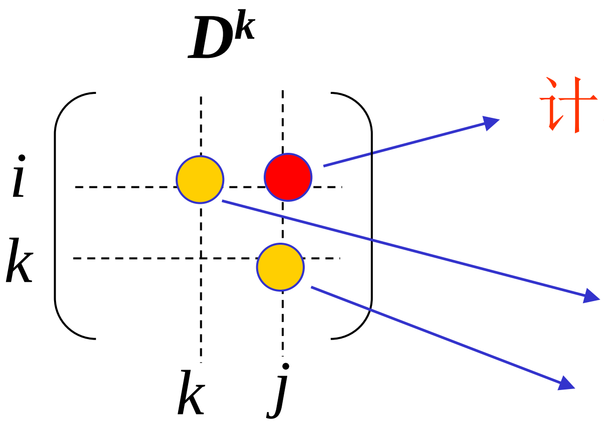


✓思考：能否只需要一个矩阵就能完成计算?

■ 动态规划算法

- 事实

$$d^k(i,j) = \mathbf{min}\ \{d^{k-1}(i,j),\quad d^{k-1}(i,k)+d^{k-1}(k,j)\}$$



计算 $d^k(i,j)=?$

$d^k(i,k) = d^{k-1}(i,k)$

$d^k(k,j) = d^{k-1}(k,j)$

✓**结论：** 只需要一个矩阵就能完成计算！

- FLOYD算法
  - 时间复杂度: $\Theta(n^3)$
  - 空间复杂度: $\Theta(n^2)$

算法 FLOYD

输入: $n \times n$ 维矩阵 $l[1\cdots n,1\cdots n]$，以便对于有向图 $G=(\{1,2,\cdots,n\},E)$中的边(i,j)的长度为 $l[i,j]$。

输出: 矩阵 D，使得 $D[i,j]$等于 i 到 j 的距离。

1. $D \leftarrow l$ {将输入矩阵 l 复制到 D}
2. **for** $k \leftarrow 1$ *to* n
3.   **for** $i \leftarrow 1$ *to* n
4.   **for** $j \leftarrow 1$ *to* n
5. $D[i,j]=\min\{D[i,j],D[i,k]+D[k,j]\}$
6.   **end for**
7.   **end for**
8. **end for**

- **Warshall算法**

1. $D \leftarrow l$ {将输入矩阵 l 复制到 D}
2. **for** $k \leftarrow 1$ *to* n
3. **for** $i \leftarrow 1$ *to* n
4. **for** $j \leftarrow 1$ *to* n
5. $D[i,j] = \min\{D[i,j], D[i,k] + D[k,j]\}$
6. **end for**
7. **end for**
8. **end for**

$$\boxed{D[i,j] = D[i,j] \vee (D[i,k] \wedge D[k,j])}$$