动态规划

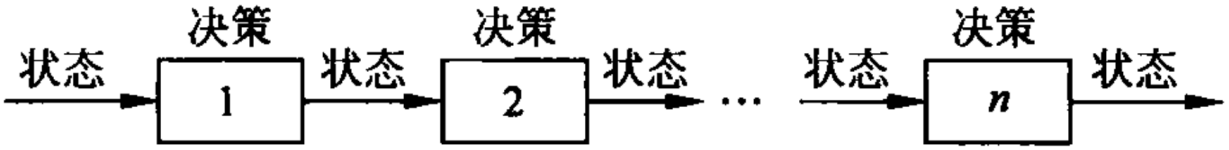
Dynamic Programming

动态规划

- 动态规划是解决多阶段决策过程最优化的一种方法，由美国数学家贝尔曼（R. Bellman）等人在20世纪50年代初提出。
- 1957 年,R.Bellman 发表了该分支领域的第一本专著《动态规划》(Dynamic Programming)
- 针对多阶段决策问题的特点，提出了解决这类问题的最优化原理，并成功地解决了生产管理、工程技术等方面的许多实际问题。

多阶段决策问题

- 可将过程划分为若干互相联系的阶段；
- 在它的每一个阶段都需要作出决策，并且一个阶段的决策确定以后，常影响下一个阶段的决策，从而影响整个过程的活动路线；
- 各个阶段所确定的决策就构成一个决策序列，通常称为一个策略；
- 每一个阶段可供选择的决策往往不止一个，对应于一个策略就有确定的活动效果；
- 多阶段决策问题，就是要在允许选择的那些策略中间，选择一个最优策略，使在预定的标准下达到最好的效果。

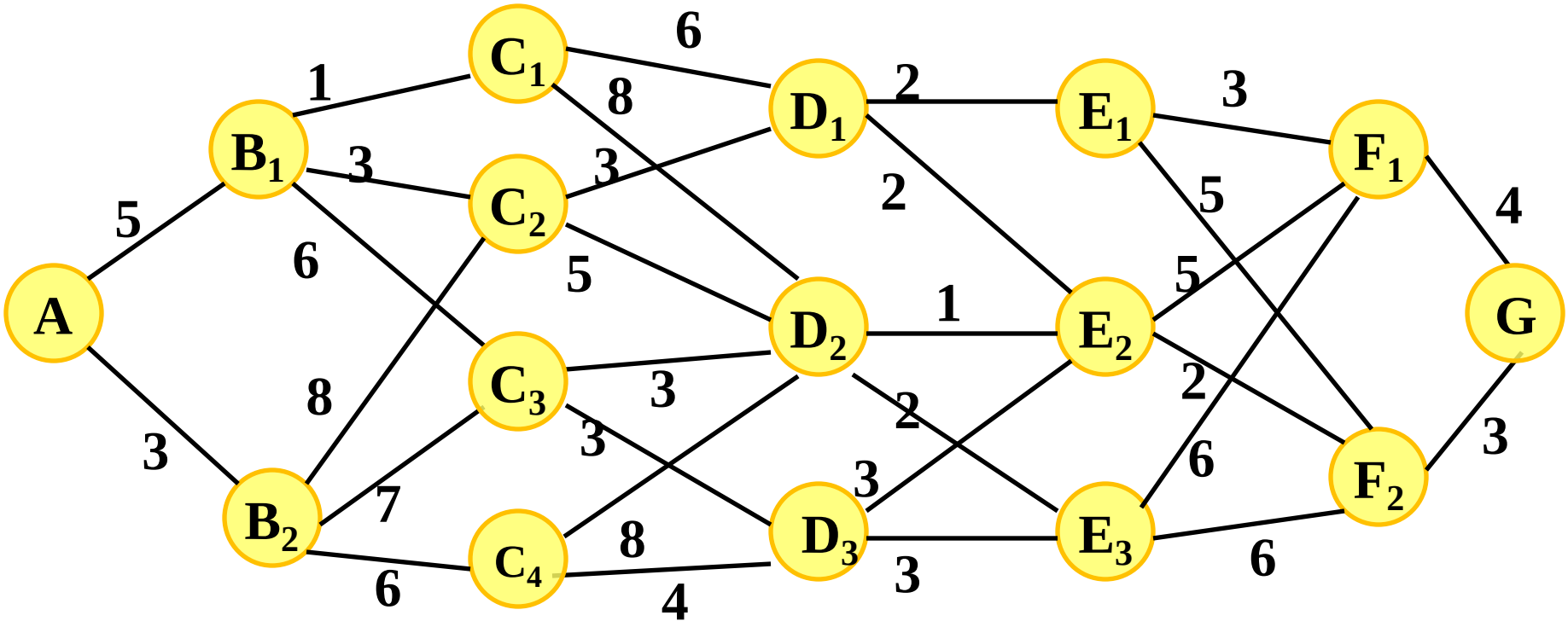


举例说明

例1　最短路线问题

给出一个线路网络，从 A 点要铺设一条管道到 G 点，其两点之间连线上的数字表示两点间的距离；

要求选择一条由 A 到 G 的铺管线路，使总距离最短。



最短路线问题

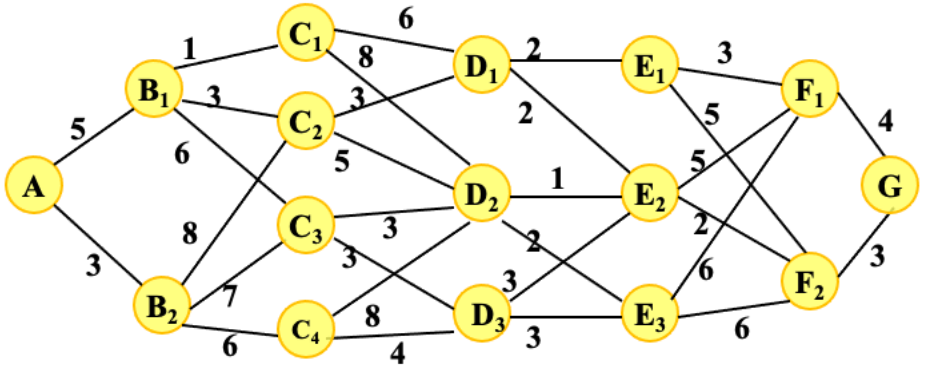
- 从A到G可以分为6个阶段。各个阶段的决策不同，铺管路线就不同。
- 当某段的始点给定时，它直接影响着后面阶段的引进路线和整个路线的长短，而后面各阶段的路线的发展不受这点以前各段路线的影响。
- 问题的要求是：在各个阶段选取一个恰当的决策，使由这些决策组成的一个策略所决定的一条路线，其总路程最短——最优策略

穷举法：共有2× 3 × 2 × 2 × 2 × 1=48 条路线

动态规划的基本概念

- 阶段(Stage)
  - 把所给问题的过程，恰当地划分成若干个相互联系的阶段，以便于求解。
  - 一般是根据时间和空间的自然特征来划分，但要便于把问题的过程能转化为多阶段决策过程。
  - 阶段变量：描述阶段的变量，常用 ***k*** 表示。

如例1可分为6个阶段来求解，***k*** 分别等于1, 2, 3, 4, 5, 6。



动态规划的基本概念

- 状态（State）：表示每个阶段开始所处的自然状况或客观条件，描述了研究问题过程的状况。

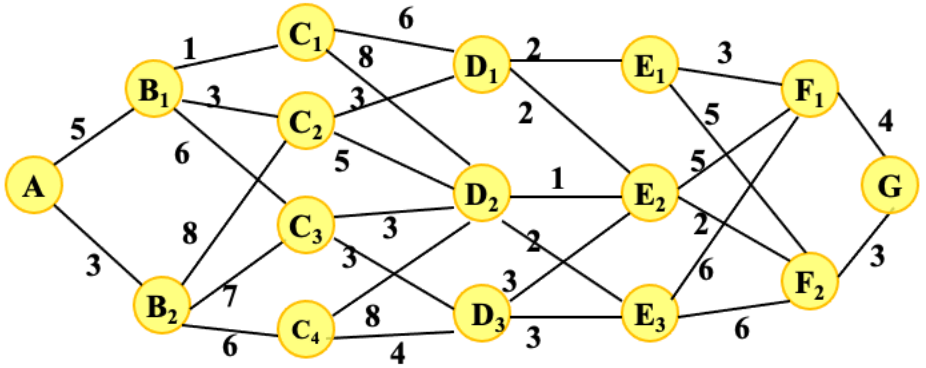
例1中，状态就是某阶段的出发位置，既是该段某支路的始点，也是前一段某支路的终点。

- 通常一个阶段包含若干个状态。

如例 1中，第一阶段有一个状态，就是点 A

第二个阶段有两个状态，B_1, B_2

第三个阶段有四个状态，C_1, C_2, C_3, C_4



动态规划的基本概念

- 状态（State）：表示每个阶段开始所处的自然状况或客观条件，描述了研究问题过程的状况。

例1中，状态就是某阶段的出发位置，既是该段某支路的始点，也是前一段某支路的终点。

- 通常一个阶段包含若干个状态。
- 状态变量：描述过程状态的变量，常用 s_k 表示在第 k 阶段的某一状态。

第 k 阶段的状态集合可表示为

$$S_k = \{ s_k^{(1)}, s_k^{(2)}, \ldots, s_k^{(i)}, \ldots, s_k^{(r)} \}$$

例1中第三阶段的状态集合就可记为

$$S_3 = \{ s_3^{(1)}, s_3^{(2)}, s_3^{(3)}, s_3^{(4)} \} = \{C_1, C_2, C_3, C_4\}$$

动态规划的基本概念

- 状态（State）：表示每个阶段开始所处的自然状况或客观条件，描述了研究问题过程的状况。

例1中，状态就是某阶段的出发位置，既是该段某支路的始点，也是前一段某支路的终点。

- 通常一个阶段包含若干个状态。
- 状态变量：描述过程状态的变量，常用 s_k 表示在第 k 阶段的某一状态。
- 状态需满足无后效性（马尔科夫性）：

如果某阶段状态给定后，则在这阶段以后过程的发展不受这阶段以前各段状态的影响。

动态规划的基本概念

- 决策（Decision）：当过程处于某一阶段的某个状态时，可以作出不同的决定（或选择），从而确定下一阶段的状态。
  - 决策变量：描述决策的变量，可以用一个数、一组数或一向量来描述、

常用 $u_k(s_k)$ 表示第 k 阶段当状态处于s_k时的决策变量。

  - 允许决策集合：决策变量的取值限制的范围。

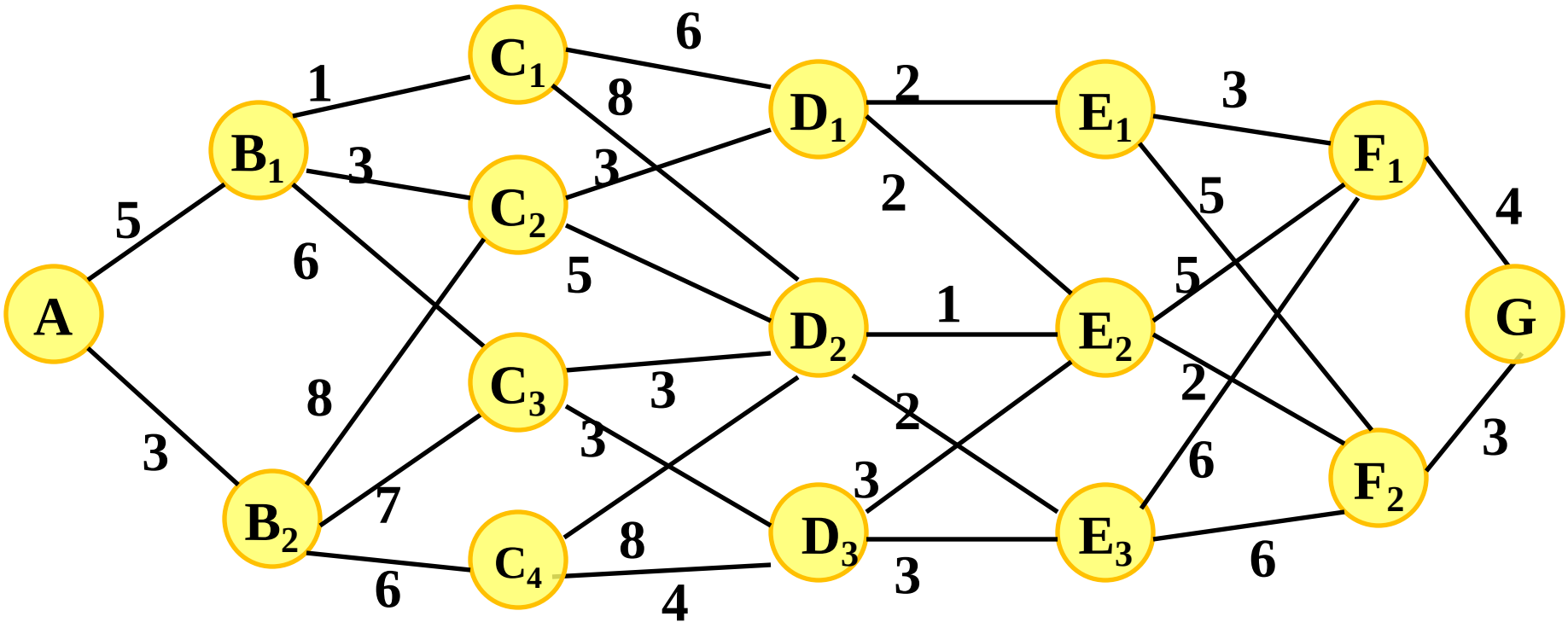
通常以$D_k(s_k)$表示第 k 阶段从状态处于s_k 时的允许决策集合，显然有 $u_k(s_k) \in D_k(s_k)$。

动态规划的基本概念

例1中第2阶段状态集合$S_2=\{B_1, B_2\}$;

则从 B_1出发的允许决策集合$D_2(B_1)=\{C_1, C_2, C_3\}$;

若选择点C_2， 则$u_2(B_1)=C_2$。



动态规划的基本概念

- 策略（Policy）：一个按顺序排列的决策组成的集合
  - 全过程策略：由每段的决策 $u_i(s_i)$ (i=1,2……,n) 组成的决策函数序列，简称策略，记为$p_{1,n}$，即

$$p_{1,n}(s_1) = \{u_1(s_1), u_2(s_2), \ldots\ldots, u_n(s_n)\}$$

  - 后部子过程（或k子过程）：由第k段开始到终点的过程，其决策函数序列$\{u_k(s_k), \ldots, u_n(s_n)\}$称为 k 子过程策略，简称子策略。即

$$p_{k,n}(s_k) = \{u_k(s_k), u_{k+1}(s_{k+2}), \ldots\ldots, u_n(s_n)\}$$

- 在实际问题中，可供选择的策略有一定的范围，此范围称为允许策略集合，用 P 表示。
- 最优策略: 从允许策略中找出达到最优效果的策略

动态规划的基本概念

- 状态转移方程：确定过程由一个状态到另一个状态的演变过程。
  - 若给定第 k 阶段状态变量 s_k 的值，如果该阶段的决策变量 u_k 一经确定，第 $k+1$ 阶段的状态变量 s_{k+1} 的值也就完全确定，记为

$$s_{k+1} = T_k(s_k, u_k)$$

称为状态转移方程，描述了由 k 阶段到 $k+1$ 阶段的状态转移规律。T_k 称为状态转移函数。

如例1中，状态转移方程为 $s_{k+1} = u_k(s_k)$。

动态规划的基本概念

- 指标函数：用来衡量过程的优劣的一种数量指标
  - 定义在全过程和所有后部子过程上的确定的数量函数，常用 $V_{k,n}$ 表示，即

    $V_{k,n}=V_{k,n}(s_k, u_k, s_{k+1},\ldots, s_{n+1})$，$k=1, 2, \ldots, n$
- 对于要构成动态规划模型的指标函数，应具有可分离性，并满足递推关系。

  即 $V_{k,n}$ 可以表示为 $s_k, u_k, V_{k+1,n}$ 的函数。记为

$V_{k,n}(s_k, u_k, s_{k+1},\ldots, s_{n+1})$

$=\psi_k(s_k, u_k, V_{k+1,n}(s_{k+1}, u_{k+1},\ldots, s_{n+1}))$

动态规划的基本概念

- **指标函数**：用来衡量过程的优劣的一种数量指标
  - 定义在全过程和所有后部子过程上的确定的数量函数，常用 $V_{k,n}$ 表示。即

    $V_{k,n}=V_{k,n}(s_k, u_k, s_{k+1},\cdots, s_{n+1})$，$k=1, 2, \ldots, n$
  - 不同问题中，指标含义也不同：距离、利润、成本、产量、资源消耗等。

例 1中，$V_{k,n}$表示第 k 阶段由点 s_k 至终点 G 的距离；

- 第 k 阶段由点 s_k 至 $u_k(s_k)$ 的距离为阶段指标(阶段效益)，记为 $d_k(s_k, u_k)$。

如：$d_5(E_1, F_1)=3$。

动态规划的基本概念

- 常见的指标函数
  - 过程和它的任一子过程的指标是它所包含的各阶段的指标的和，即

$$V_{k,n}(s_k, u_k, s_{k+1}, \ldots, s_{n+1}) = \sum_{j=k}^{n} v_j(s_j, u_j)$$

其中，$v_j(s_j, u_j)$ 表示第 j 阶段的阶段指标。

上式可写成

$$V_{k,n}(s_k, u_k, s_{k+1}, \ldots, s_{n+1})$$

$$= v_k(s_k, u_k) + V_{k+1,n}(s_{k+1}, u_{k+1}, s_{k+2}, \ldots, s_{n+1})。$$

动态规划的基本概念

- 常见的指标函数
  - 过程和它的任一子过程的指标是它所包含的各阶段的指标的乘积，即

$$V_{k,n}(s_k, u_k, s_{k+1}, \ldots, s_{n+1}) = \prod_{j=k}^{n} v_j(s_j, u_j).$$

上式可写成

$$V_{k,n}(s_k, u_k, s_{k+1}, \ldots, s_{n+1})$$

$$= v_k(s_k, u_k) V_{k+1,n}(s_{k+1}, u_{k+1}, s_{k+2}, \ldots, s_{n+1})$$

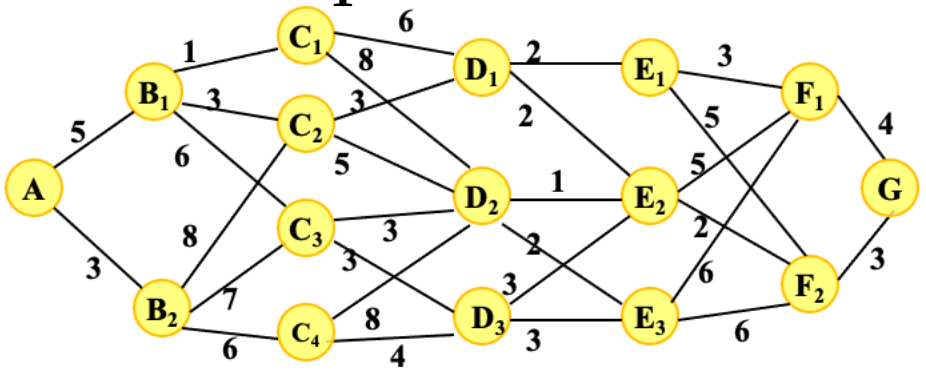
动态规划的基本概念

- **最优值函数**：指标函数的最优值，记为 $f_k(s_k)$
  - 表示从第 k 阶段的状态 s_k 开始到第 n 阶段的终止状态的过程，采取**最优策略**所得到的指标函数值，即

$$f_k(s_k) = \text{opt}_{\{u_k,\ldots,u_n\}} V_{k,n}(s_k, u_k, s_{k+1}, \ldots, u_n s_{n+1})$$

其中，opt可取 min或 max。

例 1 中，$f_k(s_k)$ 表示从第 k 段 s_k 点到终点G的最短距离。
如，$f_4(D_1)$ 就表示从第4段中的D_1点到G点的最短距离。

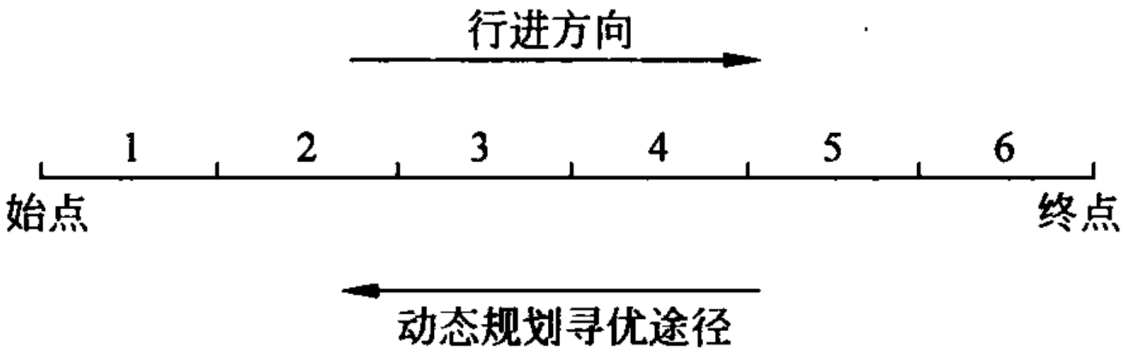


动态规划的基本思想和基本方程

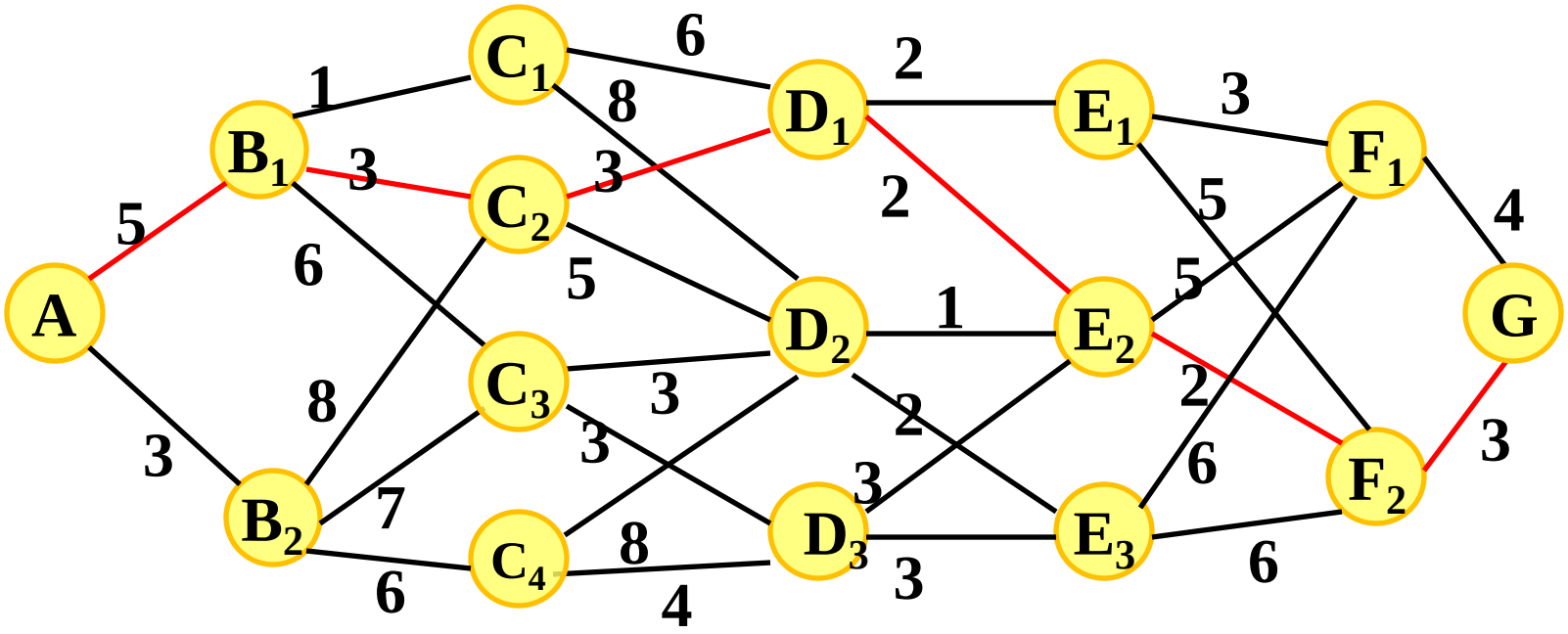
- 例：最短路线的重要特性

  如果最短路线在第 k 阶段通过点 P_k，则该最短路线中由点P_k出发到达终点的子路线，对于从点P_k出发到达终点的所有可能选择的不同路线来说，必定也是最短路线。

- 动态规划的方法是从终点逐段向始点方向寻找最短路线的一种方法。



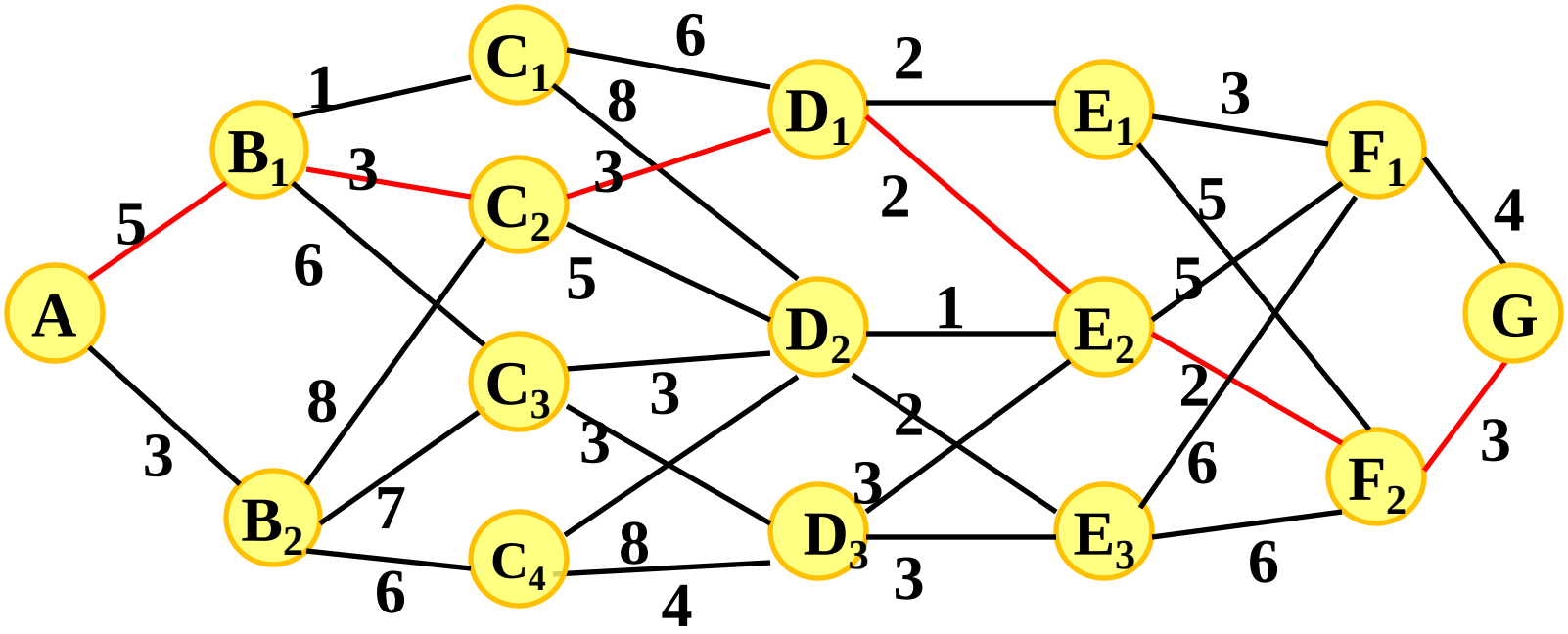
最短路径问题



- 最短路径的子路径也是最短路径

例：$A \to B_1 \to C_2 \to D_1 \to E_2 \to F_2 \to G$ 是从A到G的最短路径，则$D_1 \to E_2 \to F_2 \to G$ 是从D_1到G的最短路径。

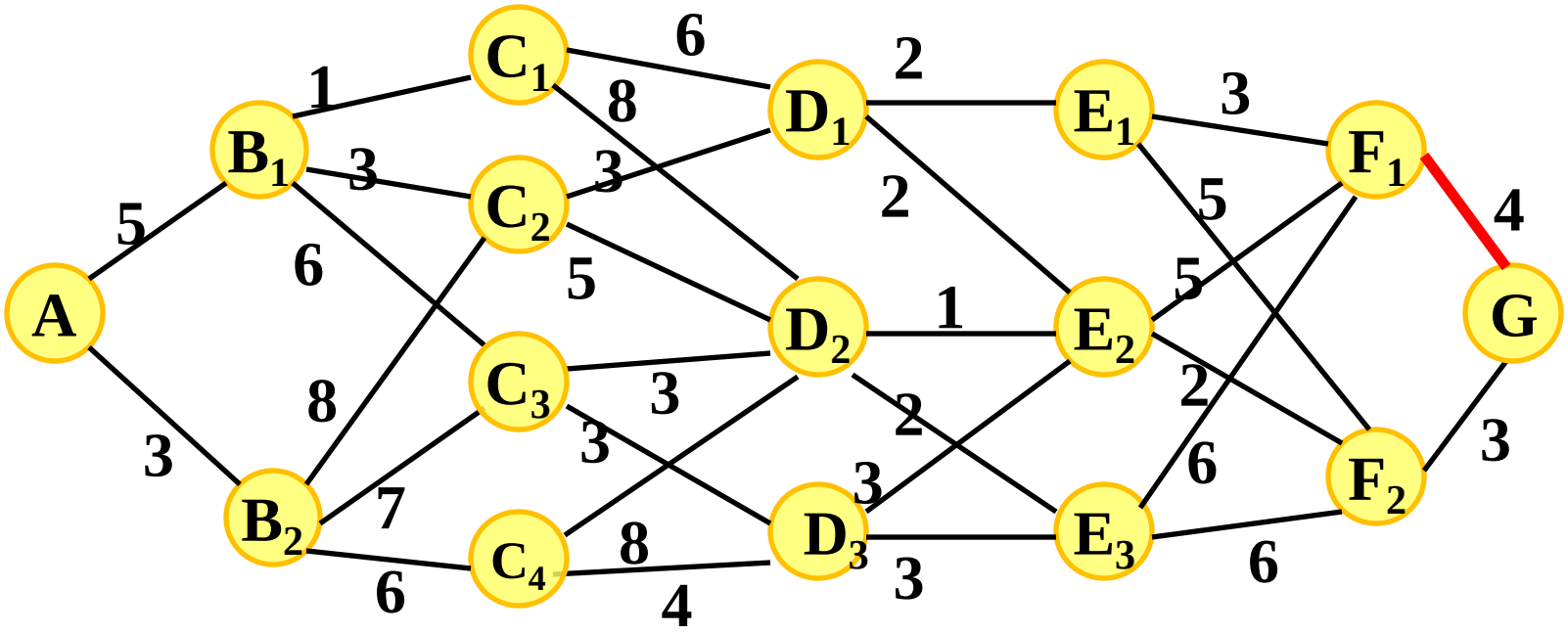
最短路径问题



- 从最后一段开始，用由后向前逐步递推的方法，求出各点到 G 点的最短路径，最后求得由A点到G点的最短路径。

$$V_{k,n}(s_k, u_k, s_{k+1}, \ldots, s_{n+1})$$
$$= v_k(s_k, u_k) + V_{k+1,n}(s_{k+1}, u_{k+1}, s_{k+2}, \ldots, s_{n+1})$$

最短路径问题



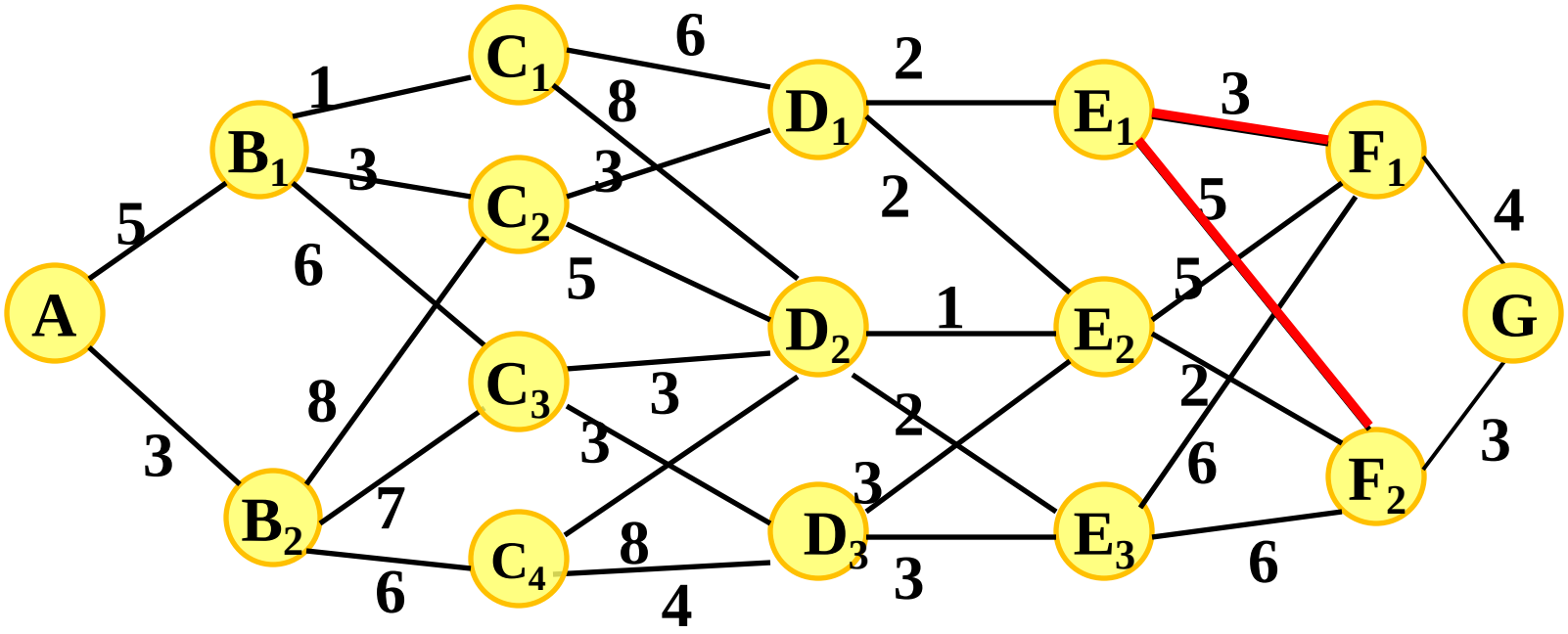
当$k = 6$时，

$f_6(F_1)$：第6阶段由F_1至G的最短距离，

故 $f_6(F_1) = 4$。

同理，$f_6(F_2) = 3$。

最短路径问题



当$k = 5$时，出发点有 E_1, E_2, E_3。

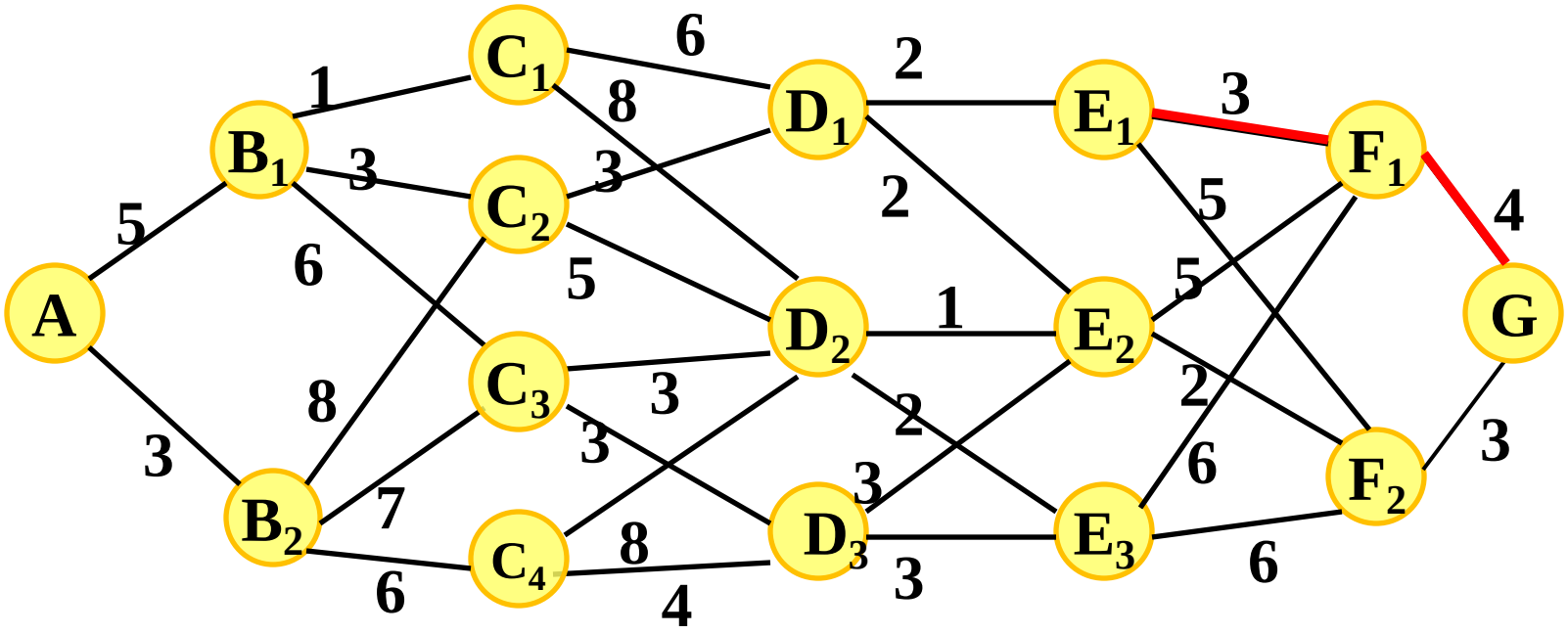
若从E_1出发，则有两个选择，至F_1或 至F_2。

则 $f_5(E_1)= \min\{d_5(E_1, F_1)+f_6(F_1), d_5(E_1, F_2)+f_6(F_2)\}$

$= \min\{3+4, 5+3\}= \min\{7, 8\} =7$

相应决策为 $u_5(E_1)=F_1$。

最短路径问题



当$k = 5$时，出发点有 E_1, E_2, E_3。

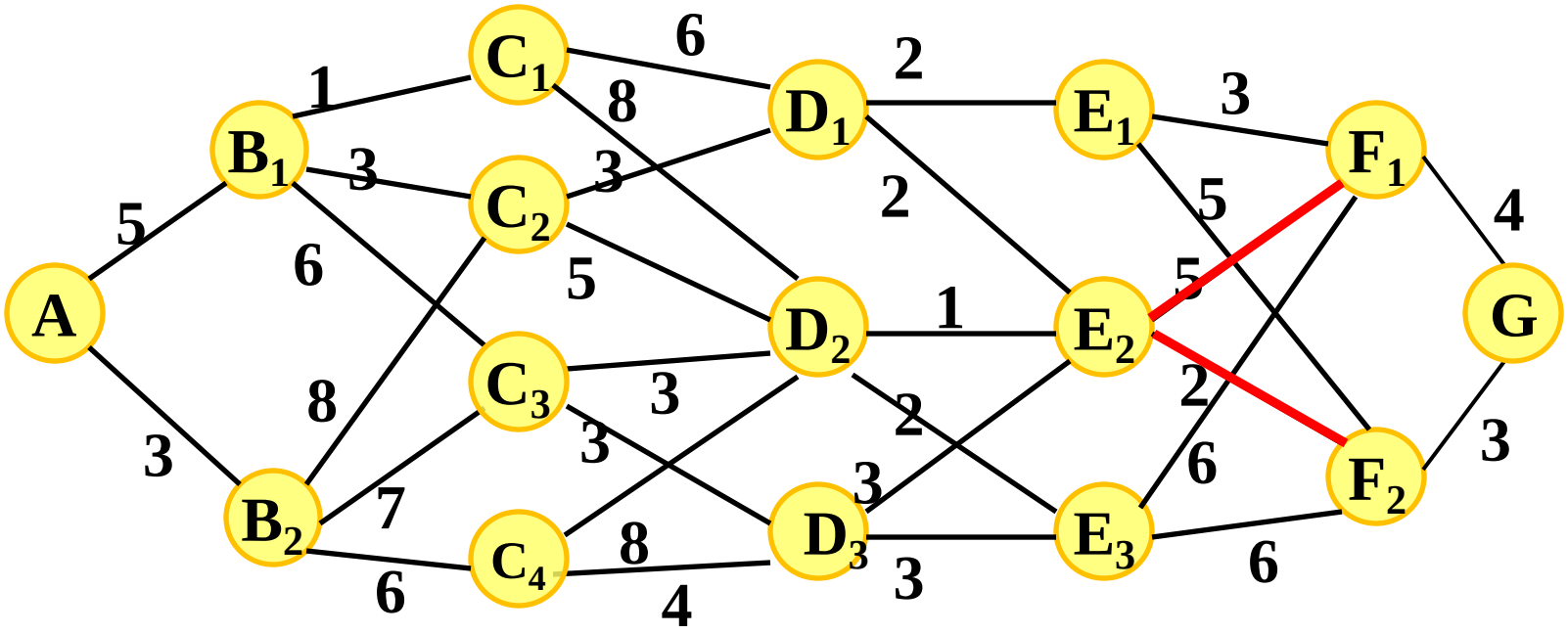
若从E_1出发，则有两个选择，至F_1或 至F_2。

则 $f_5(E_1)= \min\{d_5(E_1, F_1)+f_6(F_1), d_5(E_1, F_2)+f_6(F_2)\}$

$= \min\{3+4, 5+3\}= \min\{7, 8\} =7$

相应决策为 $u_5(E_1)=F_1$。

最短路径问题



当$k = 5$时，出发点有 E_1, E_2, E_3。

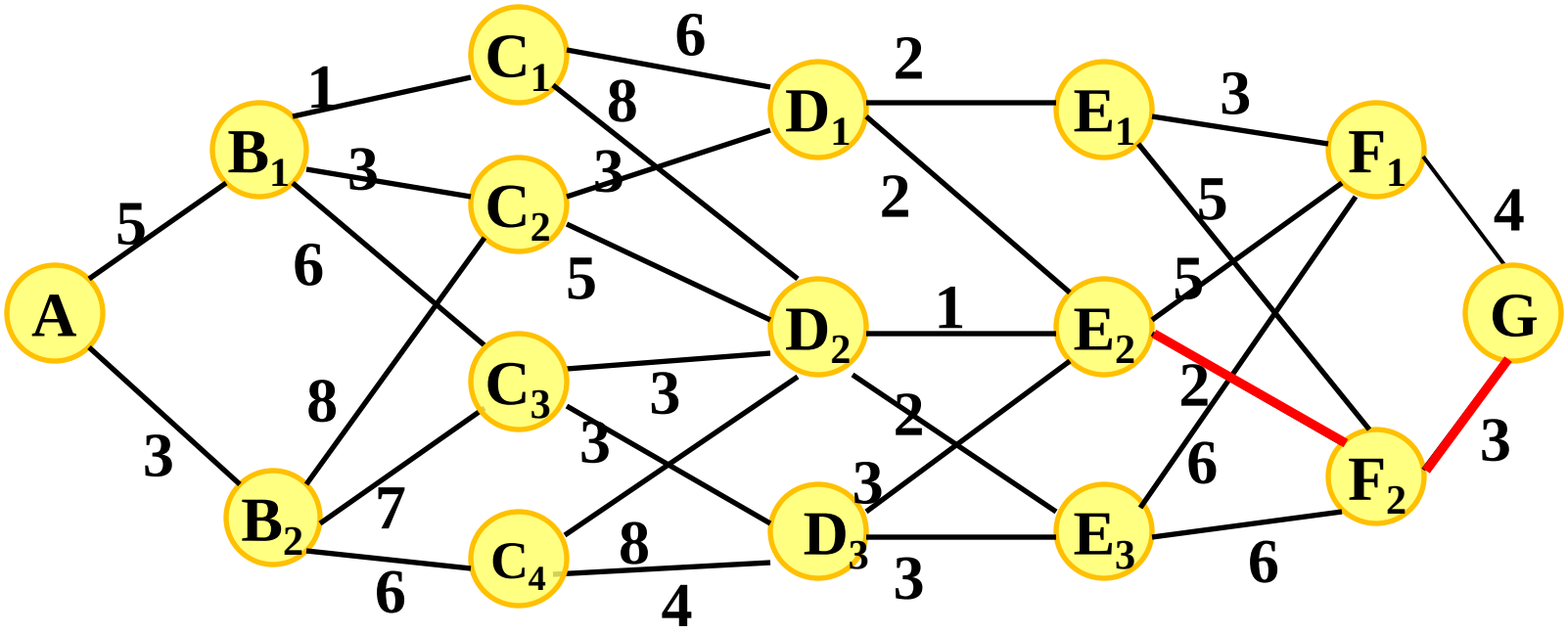
若从E_2出发，则有两个选择，至F_1或 至F_2。

则 $f_5(E_2)= \min\{d_5(E_2, F_1)+f_6(F_1), d_5(E_2, F_2)+f_6(F_2)\}$

$= \min\{5+4，2+3\}=\min\{9, 5\} =5$

相应决策为 $u_5(E_2)=F_2$。

最短路径问题



当$k = 5$时，出发点有 E_1, E_2, E_3。

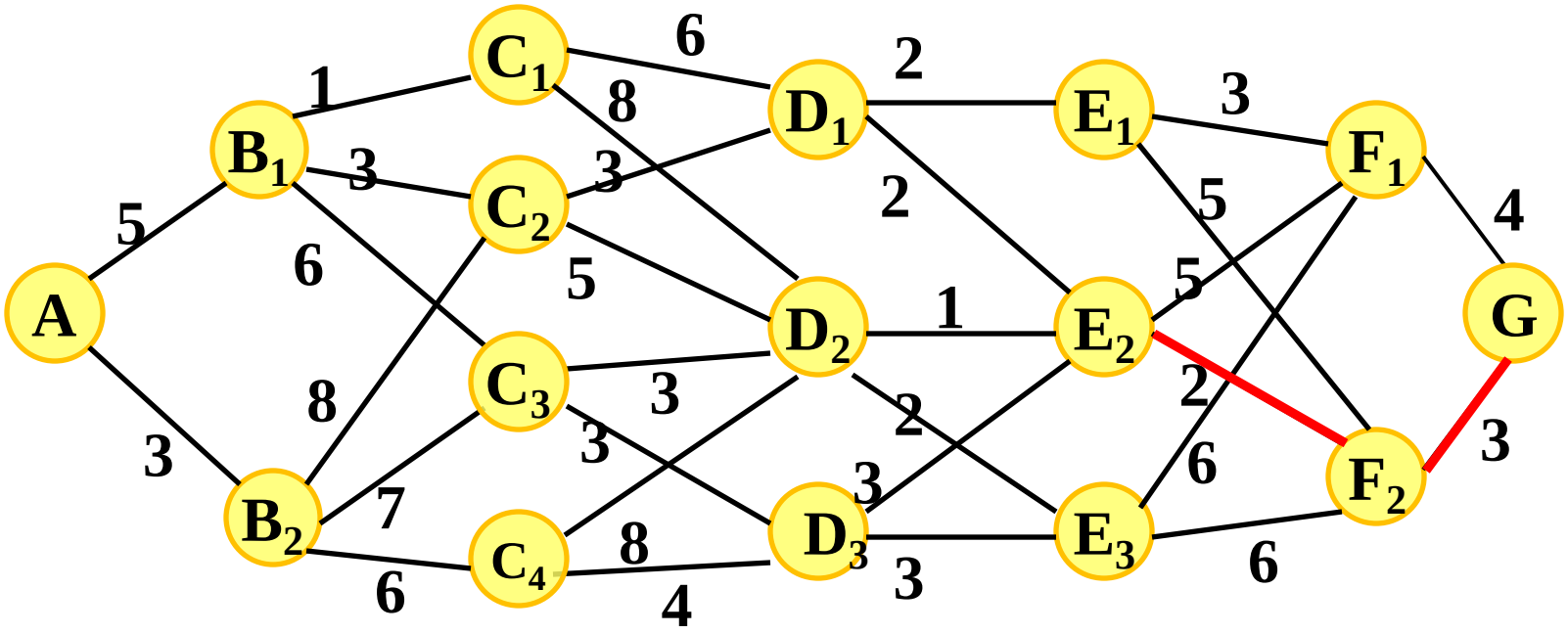
若从E_2出发，则有两个选择，至F_1或 至F_2。

则 $f_5(E_2)= \min\{d_5(E_2, F_1)+f_6(F_1), d_5(E_2, F_2)+f_6(F_2)\}$

$= \min\{5+4，2+3\}=\min\{9, 5\} =5$

相应决策为 $u_5(E_2)=F_2$。

最短路径问题



当$k = 5$时，出发点有 E_1, E_2, E_3。

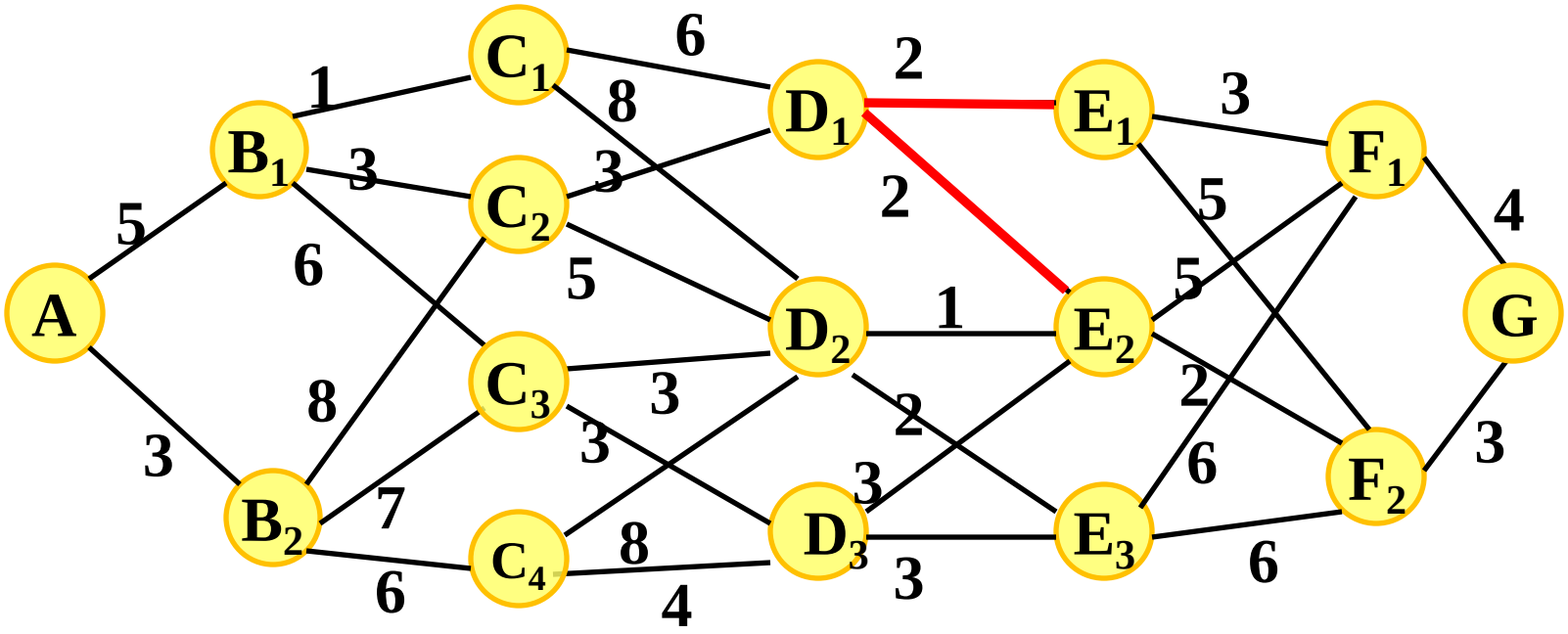
若从E_2出发，则有两个选择，至F_1或 至F_2。

则 $f_5(E_2)= \min\{d_5(E_2, F_1)+f_6(F_1), d_5(E_2, F_2)+f_6(F_2)\}$

$= \min\{5+4，2+3\}=\min\{9, 5\} =5$

相应决策为 $u_5(E_2)=F_2$。同理可得，$f_5(E_3)=9$, $u_5(E_3)=F_2$

最短路径问题



当$k = 4$时，出发点有 D_1, D_2, D_3。

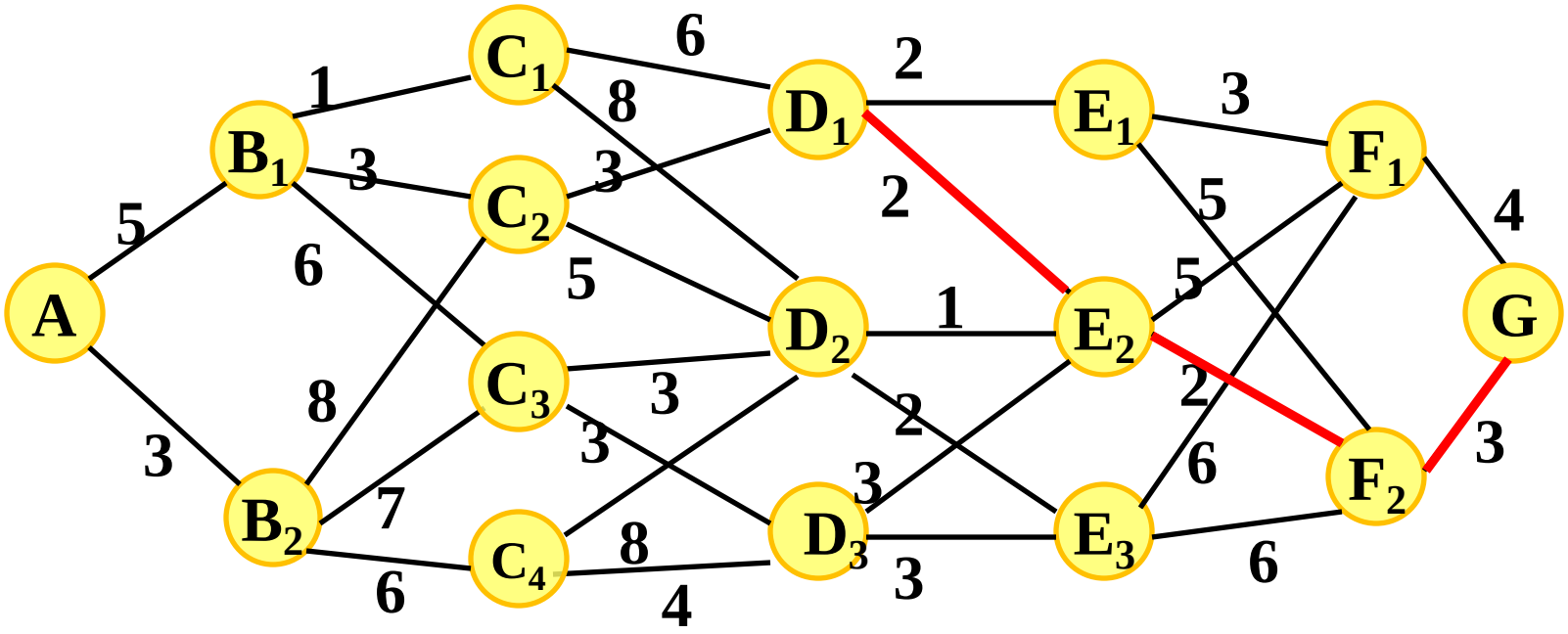
若从D_1出发，则有两个选择，至E_1或 至E_2。

则 $f_4(D_1)= \min\{d_4(D_1, E_1)+f_5(E_1), d_4(D_1, E_2)+f_5(E_2)\}$

$= \min\{ 2+7, 2+5\} = 7$

相应决策为 $u_4(D_1)=E_2$。

最短路径问题



当$k = 4$时，出发点有 D_1, D_2, D_3。

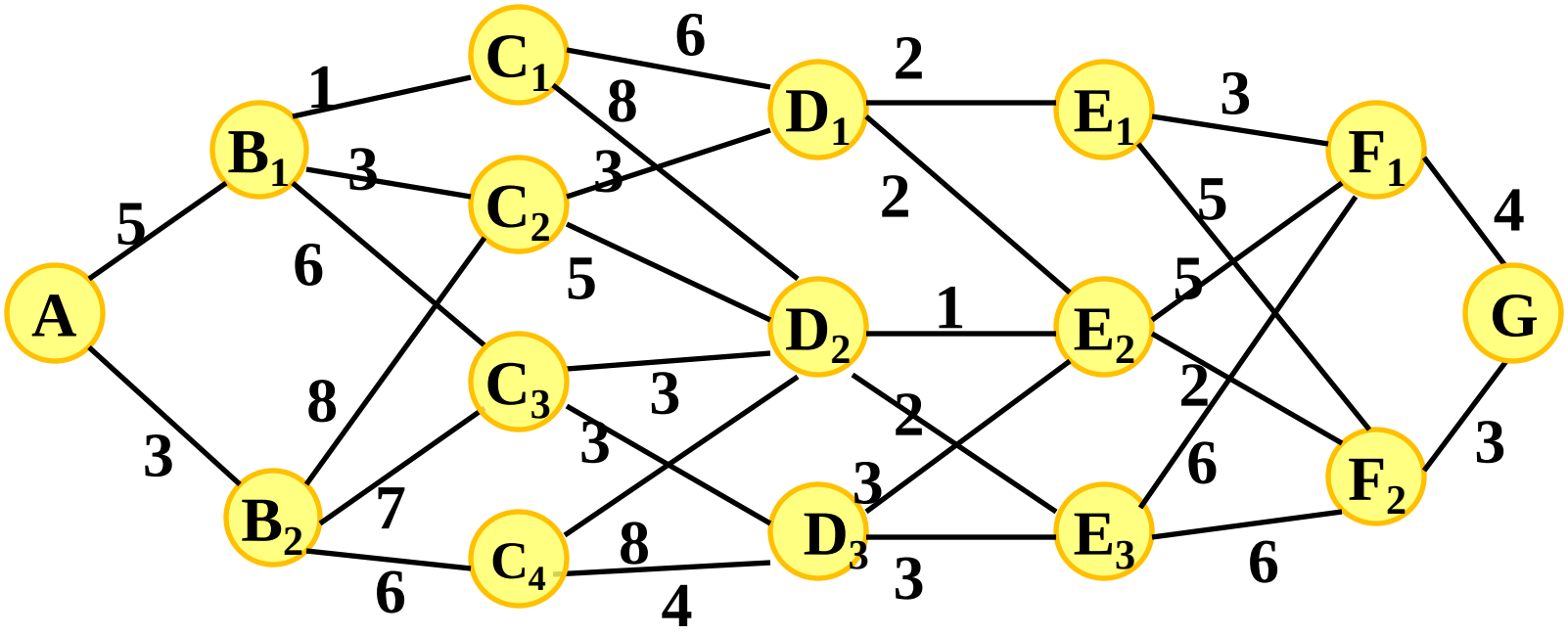
若从D_1出发，则有两个选择，至E_1或 至E_2。

则 $f_4(D_1)= \min\{d_4(D_1, E_1)+f_5(E_1), d_4(D_1, E_2)+f_5(E_2)\}$

$= \min\{ 2+7, 2+5\} = 7$

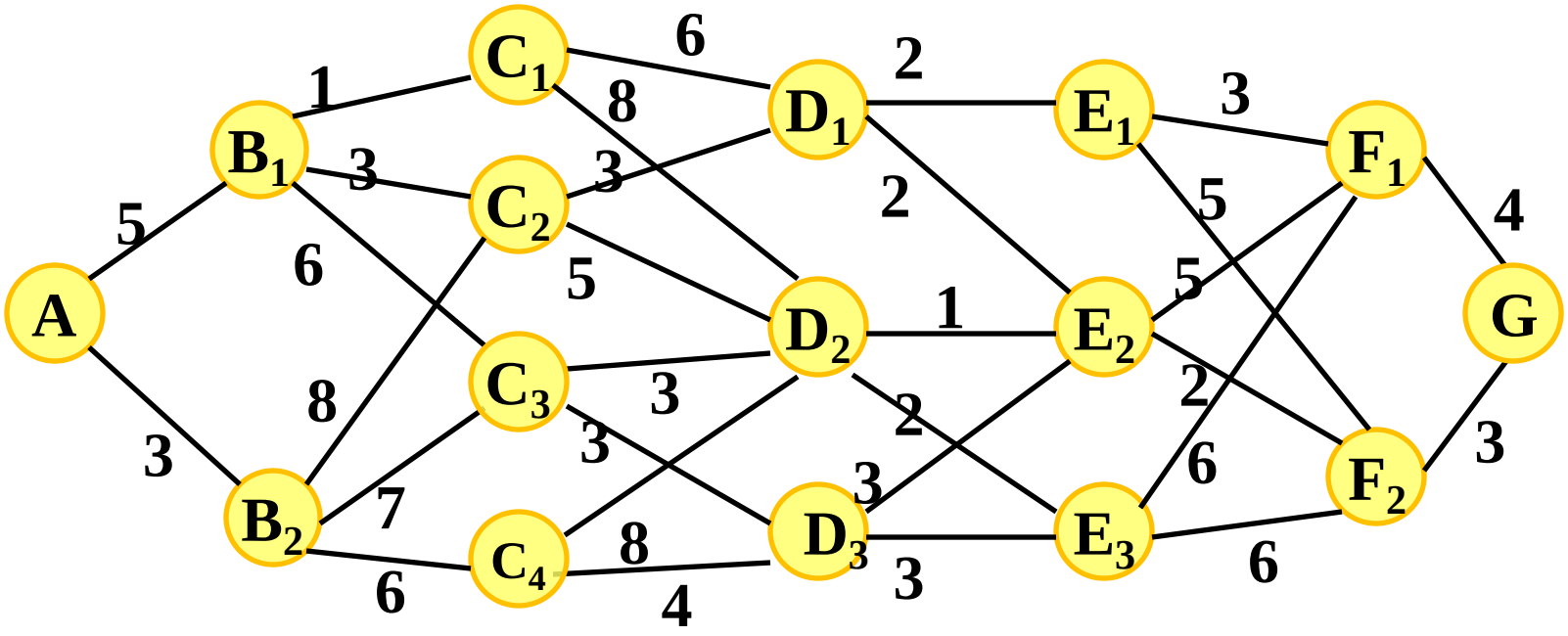
相应决策为 $u_4(D_1)=E_2$。

最短路径问题



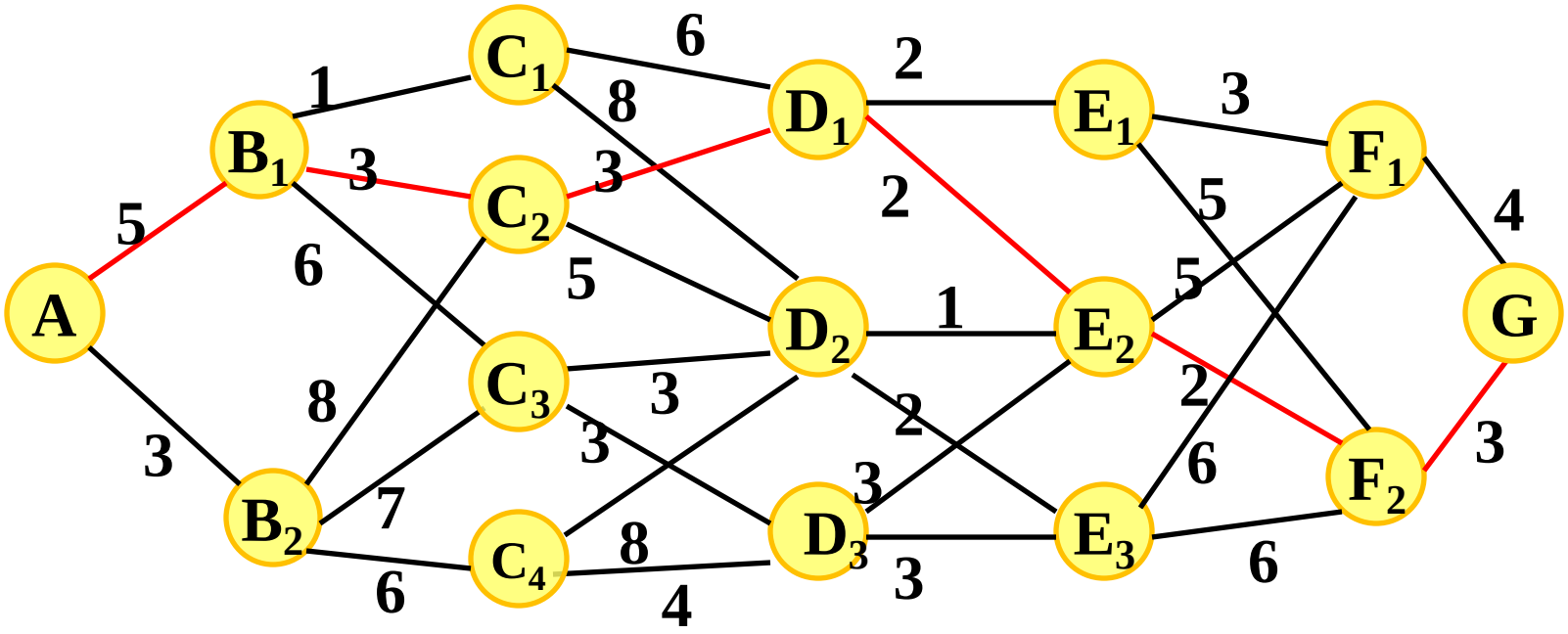
	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
f							12	7	6	8	7	5	9	4	3	0
u							D_3	E_2	E_2	E_2	F_1	F_2	F_2	G	G	0

最短路径问题



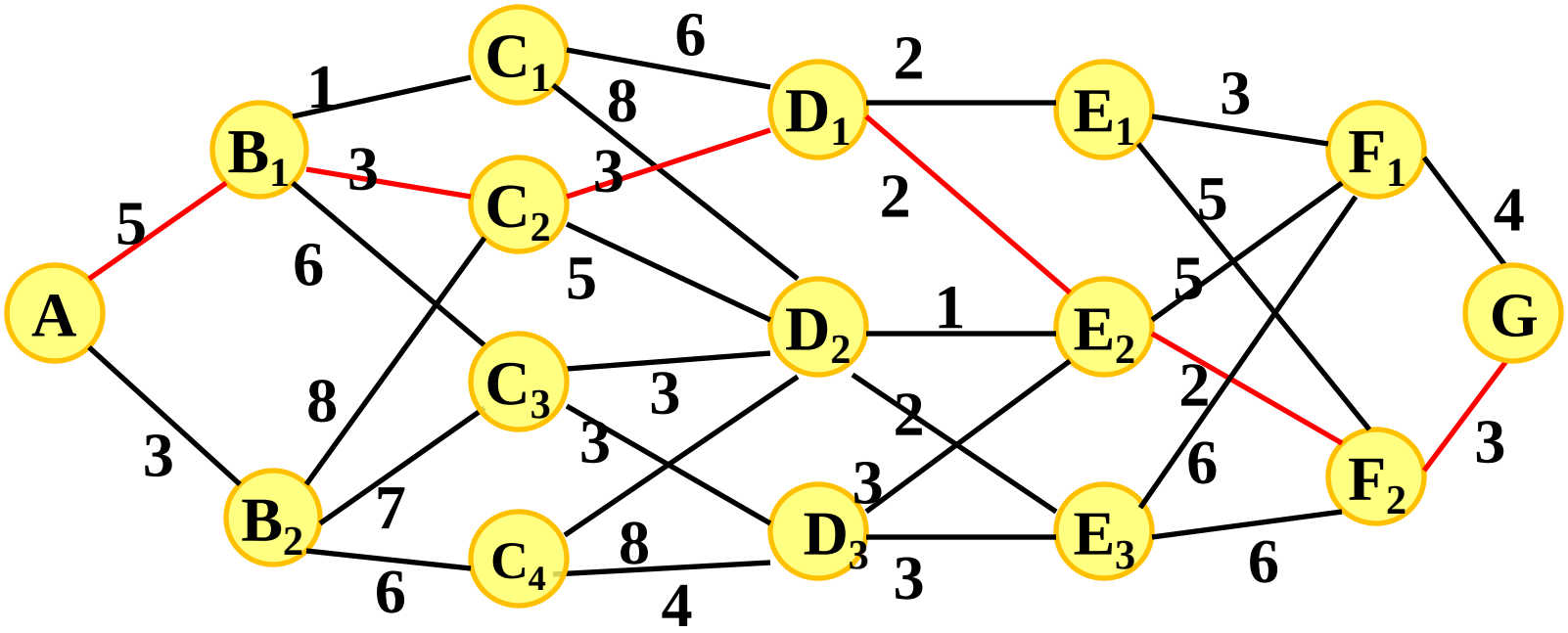
	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
f	18	13	16	13	10	9	12	7	6	8	7	5	9	4	3	0
u	B_1	C_2	C_3	D_1	D_1	D_2	D_3	E_2	E_2	E_2	F_1	F_2	F_2	G	G	0

最短路径问题



	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
f	18	13	16	13	10	9	12	7	6	8	7	5	9	4	3	0
u	B_1	C_2	C_3	D_1	D_1	D_2	D_3	E_2	E_2	E_2	F_1	F_2	F_2	G	G	0

动态规划的函数基本方程



- 在求解的各个阶段，利用了k阶段与k+1阶段之间的递推关系：

$$f_k(s_k) = \min_{u_k \in D_k(s_k)} \{d_k(s_k, u_k(s_k)) + f_{k+1}(u_k(s_k))\},$$

$$k = 6, 5, 4, 3, 2, 1$$

$f_7(s_7)=0$ （边界条件）

动态规划的递推关系

- 一般情况下，k 阶段与 k+1阶段的递推关系式可写为

$$f_k(s_k) = \mathbf{opt}_{u_k \in D_k(s_k)} \{ v_k(s_k, u_k(s_k)) + f_{k+1}(u_k(s_k)) \}$$

边界条件为 $f_{n+1}(s_{n+1})=0$。

以上递推关系式称为动态规划的基本方程。

动态规划的基本思想

- 关键在于正确写出基本递推关系式和恰当边界条件。
- 将问题的过程分成几个相互联系的阶段，从而把一个大问题转化成一族同类型的子问题，然后逐个求解。
- 从边界条件开始，逐段递推寻优，在每一个子问题的求解中，均利用了它前面的子问题的最优化结果。
- 最后一个子问题的最优解，就是整个问题的最优解。

逆序解法与顺序解法

- 规定从A点到G点为顺行方向，由G点到A点为逆行方向
  - 逆序解法：以A为始端，以G为终端，从G到A的解法（如例1）
  - 顺序解法：以A为始端，以G为终端，从A到G的解法
- 顺序解法和逆序解法只表示行进方向的不同或始端的颠倒。
- 但动态规划方法求最优解时，一般都是在行进方向规定后，均要逆着这个规定的行进方向，从最后一段向前逆推计算，逐段找出最优途径。

与穷举法的对比

- 减少了计算量

穷举法：

- 要对48条路线进行比较，比较运算要进行47次；
- 求各条路线的距离（288次加法），即使使用逐段累加方法，也要进行0+6+12+32+48+48=146次加法运算。

动态规划方法：

- 比较运算（从k=5开始向前算）共进行3+3+4+4+1=15次。
- 每次比较运算对应两次加法运算，再去掉中间重复两次（即B1→C1， B2→C4各多算了一次），实际只有28次加法运算。

与穷举法的对比

- 丰富了计算结果
  - 得到的不仅仅是由 A点出发到 G点的最短路线及相应的最短距离，而且得到了从所有各中间点出发到G点的最短路线及相应的距离。
  - 求出的不是一个最优策略，而是一族最优策略，有利于帮助分析所得结果。

	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
f	18	13	16	13	10	9	12	7	6	8	7	5	9	4	3	0
u	B_1	C_2	C_3	D_1	D_1	D_2	D_3	E_2	E_2	E_2	F_1	F_2	F_2	G	G	0

与穷举法的对比

- 丰富了计算结果
  - 得到的不仅仅是由 A点出发到 G点的最短路线及相应的最短距离，而且得到了从所有各中间点出发到G点的最短路线及相应的距离。
  - 求出的不是一个最优策略，而是一族最优策略，有利于帮助分析所得结果。

	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
f	18	13	16	13	10	9	9	7	6	8	7	5	9	4	3	0
u	B_1	C_2	C_3	D_1	D_1	D_2	D_2	E_2	E_2	E_2	F_1	F_2	F_2	G	G	0

与穷举法的对比

- 丰富了计算结果
  - 得到的不仅仅是由 A点出发到 G点的最短路线及相应的最短距离，而且得到了从所有各中间点出发到G点的最短路线及相应的距离。
  - 求出的不是一个最优策略，而是一族最优策略，有利于帮助分析所得结果。

	A	B_1	B_2	C_1	C_2	C_3	C_4	D_1	D_2	D_3	E_1	E_2	E_3	F_1	F_2	G
f	18	13	16	13	10	9	9	7	6	8	7	5	9	4	3	0
u	B_1	C_2	C_3	D_1	D_1	D_2	D_2	E_2	E_2	E_2	F_1	F_2	F_2	G	G	0

动态规划的建模

- 将问题的过程划分成恰当的阶段;
- 正确选择状态变量 s_k，使它既能描述过程的演变，又要满足无后效性;
- 确定决策变量 u_k 及每阶段的允许决策集合 $D_k(s_k)$;
- 正确写出状态转移方程;
- 正确写出指标函数 $V_{k,n}$的关系，它应满足三个性质
  - 是定义在全过程和所有后部子过程上的数量函数;
  - 具有可分离性，并满足递推关系

$$V_{k,n}(s_k, u_k, s_{k+1},\ldots, s_{n+1}) = \psi_k(s_k, uk, V_{k+1,n}(s_{k+1}, u_{k+1},\ldots, s_{n+1}))$$

  - 函数$\psi_k(s_k, uk, V_{k+1,n})$对于变量$V_{k+1,n}$要严格单调。

动态规划方程求解

- 设指标函数是取各阶段指标的和的形式，即

$$V_{k,n}(s_k, u_k, s_{k+1}, \ldots, s_{n+1}) = \sum_{j=k}^{n} v_j(s_j, u_j)$$

其中 $v_j(s_j, u_j)$表示第 j 段的指标。上式可写成

$$V_{k,n}(s_k, u_k, s_{k+1}, \ldots, s_{n+1}) = v_k(s_k, u_k) + V_{k+1,n}(s_{k+1}, u_{k+1}, \ldots, s_{n+1})。$$

当初始状态给定时，过程的策略就被确定，则指标函数也就确定了。

因此，指标函数是初始状态和策略的函数，可记为

$$V_{k,n}[s_k, p_{k,n}(s_k)]$$

故上面递推关系又可写成

$$V_{k,n}[s_k, p_{k,n}] = v_k(s_k, u_k) + V_{k+1,n}[s_{k+1}, p_{k+1,n}]$$

动态规划方程求解

故上面递推关系又可写成

$$V_{k,n}[s_k, p_{k,n}]=v_k(s_k,u_k)+V_{k+1,n}[s_{k+1}, p_{k+1,n}]$$

其子策略 $p_{k,n}(s_k)$ 可看作是由决策$u_k(s_k)$ 和$p_{k+1,n}(s_{k+1})$组成而成，即 $p_{k,n}=\{u_k(s_k), p_{k+1,n}(s_{k+1})\}$。

用 $p^*_{k,n}(s_k)$表示初始状态为 s_k 的后部子过程所有子策略中的最优子策略，　则最优值函数为：

$$f_k(s_k)= V_{k,n}[s_k, p^*_{k,n}(s_k)] = \mathrm{opt}_{p_{k,n}}\{V_{k,n}[s_k, p_{k,n}(s_k)]\},$$

$$=\mathrm{opt}_{\{u_k,\, p_{k+1,n}\}}\{v_k(s_k,u_k) + V_{k+1,n}[s_{k+1}, p_{k+1,n}]\}$$

$$= \mathrm{opt}_{u_k}\{v_k(s_k,u_k) + \mathrm{opt}_{p_{k+1,n}} V_{k+1,n}[s_{k+1}, p_{k+1,n}]\}$$

动态规划方程求解

$f_k(s_k) = V_{k,n}[s_k, p^*_{k,n}(s_k)]$

$= \mathrm{opt}_{u_k}\{v_k(s_k, u_k) + \mathrm{opt}_{p_{k+1,n}}\{V_{k+1,n}[s_{k+1}, p_{k+1,n}]\}\}.$

但 $f_{k+1}(s_{k+1}) = \mathrm{opt}_{p_{k+1,n}}\{V_{k+1,n}[s_{k+1}, p_{k+1,n}]\}$,

因此， $f_k(s_k) = \mathrm{opt}_{u_k \in D_k(s_k)}\{v_k(s_k, u_k) + f_{k+1}(s_{k+1})\}$

$k = n, n\text{-}1, \ldots, 1$

边界条件为 $f_{n+1}(s_{n+1}) = 0$。 逆序求解 顺序求解？

其中 $s_{k+1} = T_k(s_k, u_k)$。

其求解过程，根据边界条件，从$k= n$开始，由后向前逆推，从而逐步可求得各段的最优决策和相应的最优值，最后求出$f_1(s_1)$，就得到整个问题的最优解。

动态规划最优性原理

- 动态规划最优性原理

  “作为整个过程的最优策略具有这样的性质：即无论过去的状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略。”

  — **R.Bellman**等人于**20**世纪 **50**年代提出

  - 一个最优策略的子策略总是最优的
  - 但不是动态规划的理论基础

- 反映动态规划基本方程的最优性定理，是策略的最优性的充分必要条件，而最优性原理仅仅是策略最优性的必要条件。

动态规划的最优性定理

设阶段数为 n 的多阶段决策过程，其阶段编号为 $k = 0, 1, 2, \ldots, n\text{-}1$。允许策略 $p_{0,n-1}^{*} = (u_0^{*}, u_1^{*}, \ldots, u_{n-1}^{*})$ 为最优策略的充要条件是对任意一个 k，$0<k<n\text{-}1$ 和 $s_0 \in S_0$，有

$$V_{0,n\text{-}1}(s_0, p_{0,n-1}^{*}) =$$

$$\mathrm{opt}_{p_{0,k-1} \in p_{0,k-1}(s_0)} \{V_{0,k-1}(s, p_{0,k-1}) + \mathrm{opt}_{p_{k,n-1} \in p_{k,n-1}(\tilde{s}_k)} V_{k,n-1}(\tilde{s}_k, p_{k,n-1})\}$$

其中，$p_{0,n-1}^{*} = (p_{0,k-1}, p_{k,n-1})$，$\tilde{s}_k = T_{k\text{-}1}(s_{k\text{-}1}, u_{k\text{-}1})$ 是由给定初始状态 s_0 和子策略 $p_{0,k\text{-}1}$ 所确定的 k 段状态。

当 V 是效益函数时，opt 取max; 当 V 是损失函数时，opt 取min。

推论—最优性原理

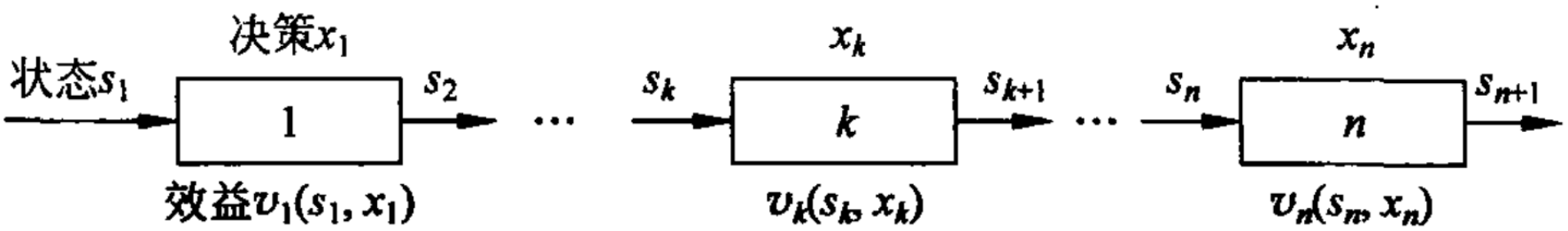
若允许$p^*_{0,n-1}$是最优策略，则对任意的 k, $0<k<n-1$，它的子策略 $p^*_{k,n-1}$对于以 $s^*_k = T_{k-1}(s^*_{k-1}, u^*_{k-1})$为起点的 k 到n-1子过程来说，必是最优策略。

（注意：k 阶段状态 s^*_k 是由 s_0 和 $p^*_{0,k-1}$ 所确定的）

- 从最优性定理可以看出，如果一个决策问题有最优策略，则该问题的最优值函数一定可用动态规划的基本方程来表示，反之亦真。
- 为用动态规划方法去处理决策问题提供了理论依据和指明方法，就是要充分分析决策问题结构，使它满足动态规划的条件，正确写出动态规划基本方程。

动态规划和静态规划

- 如线性规划、非线性规划所研究的问题通常是与时间无关的，故又称它们为静态规划。
- 对于某些静态的问题，可以人为地引入时间因素，看做是按阶段进行的一个动态规划问题，这就使得动态规划成为求解某些线性、非线性规划的有效方法。



动态规划应用举例

资源分配问题

- 资源分配问题：将供应量有限的一种或若干种资源（例如原材料、资金、机器设备、劳力、食品等等），分配给若干个使用者，而使目标函数为最优。
- 问题描述

设有某种原料，总数量为 a，用于生产 n 种产品。若分配数量 x_i 用于生产第 i 种产品，其收益为 $g_i(x_i)$。问应如何分配，才能使生产 n 种产品的总收入最大？

可写成静态规划问题：

$$\begin{cases} \max z = g_1(x_1) + g_2(x_2) + \cdots + g_n(x_n) \\ x_1 + x_2 + \cdots + x_n = a \\ x_i \geq 0, i = 1, 2, \ldots, n \end{cases}$$

资源分配问题

- 可写成静态规划问题：

$$\begin{cases} \max z = g_1(x_1) + g_2(x_2) + \cdots + g_n(x_n) \\ x_1 + x_2 + \cdots + x_n = a \\ x_i \geq 0, i = 1, 2, \ldots, n \end{cases}$$

- 应用动态规划处理这类静态规划问题
  - 阶段：把资源分配给一个或几个使用者的过程
  - 决策变量：规划问题中的变量
  - 状态变量：将累计的量或随递推过程变化的量

资源分配问题

- 可写成静态规划问题：

$$\begin{cases} \max z = g_1(x_1) + g_2(x_2) + \cdots + g_n(x_n) \\ x_1 + x_2 + \cdots + x_n = a \\ x_i \geq 0, i = 1, 2, \ldots, n \end{cases}$$

- 应用动态规划处理这类静态规划问题
  - 阶段：把资源分配给一个或几个使用者的过程
  - 决策变量：规划问题中的变量

u_k：分配给生产第 k 种产品的原料数。

  - 状态变量：将累计的量或随递推过程变化的量

s_k：分配用于生产第 k 种产品至第 n 种产品的原料数量。

  - 状态转移方程：$s_{k+1} = s_k - u_k = s_k - x_k$

资源分配问题

- 可写成静态规划问题：

$$\begin{cases} \max z = g_1(x_1) + g_2(x_2) + \cdots + g_n(x_n) \\ x_1 + x_2 + \cdots + x_n = a \\ x_i \geq 0, i = 1, 2, \ldots, n \end{cases}$$

- 应用动态规划处理这类静态规划问题
  - 允许决策集合：$D_k(s_k) = \{ u_k | \ 0 \leq u_k = x_k \leq s_k \}$

$f_k(s_k)$：以数量为 s_k 的原料分配给第 k 种产品至第 n 种产品所得到的最大总收入。

可写出动态规划的递推关系式：

$$f_k(s_k) = \max_{0 \leq x_k \leq s_k} \{ g_k(x_k) + f_{k+1}(s_k - x_k) \}, \ k = n-1, \ldots, 2, 1$$

$$f_n(s_n) = \max_{0 \leq x_n \leq s_n} g_n(x_n)$$

资源分配问题

- 可写成静态规划问题：

$$\begin{cases} \max z = g_1(x_1) + g_2(x_2) + \cdots + g_n(x_n) \\ x_1 + x_2 + \cdots + x_n = a \\ x_i \geq 0, i = 1, 2, \ldots, n \end{cases}$$

- 应用动态规划处理这类静态规划问题
  - 允许决策集合：$D_k(s_k) = \{ u_k | 0 \leq u_k = x_k \leq s_k \}$

$f_k(s_k)$：以数量为 s_k 的原料分配给第 k 种产品至第 n 种产品所得到的最大总收入。

可写出动态规划的递推关系式：

$$f_k(s_k) = \max_{0 \leq x_k \leq s_k} \{ g_k(x_k) + f_{k+1}(s_k - x_k) \}, \quad k = n, \ldots, 2, 1$$

$$f_{n+1}(s_{n+1}) = 0$$

例：现将某种高效率设备五台，分配给甲、乙、丙三个工厂，各工厂若获得这种设备之后的赢利如下表所示。问：这五台设备如何分配给各工厂，才能赢利最大？

设备台数	甲	乙	丙
0	0	0	0
1	3	5	4
2	7	10	6
3	9	11	11
4	12	11	12
5	13	11	12

例：现将某种设备n台，分配给m个工厂，若工厂 k 获得 x_k 台设备后的赢利为$P_k(x_k)$。问：这n台设备如何分配给各工厂，才能赢利最大？

例：现将某种高效率设备五台，分配给甲、乙、丙三个工厂，各工厂若获得这种设备之后的赢利如下表所示。
问：这五台设备如何分配给各工厂，才能赢利最大？

解： 将问题按工厂分为 m 个阶段。

令 s_k：分配给第 k 个工厂至第 n 个工厂的设备台数；

x_k：分配给第 k个工厂的设备台数。

则 $s_{k+1} = s_k - x_k$： 分配给第 k+1 个工厂至第n 个工厂的设备台数。

$P_k(x_k)$：x_k 台设备分配到第 k 个工厂所得的赢利值。

$f_k(s_k)$：s_k 台设备分配给第 k个工厂至第 n 个工厂时所得的最大赢利值。

解： 因而可写出递推关系式为：

$$f_k(s_k) = \max_{0 \le x_k \le s_k} \{P_k(x_k) + f_{k+1}(s_k - x_k)\},\ k = 3, 2, 1$$

$$f_4(s_4) = 0$$

下面从最后一个阶段开始向前逆推计算。

第 3阶段：将 s_3台设备全部分配给工厂丙，最大赢利值为 $f_3(s_3) = \max_{x_3}\{P_3(x_3)\}$, 其中， $x_3 = s_3 = 0, 1, 2, 3, 4, 5.$

s_3	$P_3(x_3)$ x_3						$f_3(s_3)$	x_3^*
	0	1	2	3	4	5		
0	0						0	0
1		4					4	1
2			6				6	2
3				11			11	3
4					12		12	4
5						12	12	5

此时只有一个工厂，因此全部分配给工厂丙，故它的赢利值就是该段的最大赢利值。

其中， x_3^*表示使$f_3(s_3)$为最大值时最优决策。

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0								
1								
2								
3								
4								
5								

解：（续）第2阶段：将 s_2台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0							
1	0							
2	0							
3	0							
4	0							
5	0							

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0							
1	0+4							
2	0+6							
3	0+11							
4	0+12							
5	0+12							

解：（续）第2阶段：将 s_2台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0							
1	0+4	5						
2	0+6	5						
3	0+11	5						
4	0+12	5						
5	0+12	5						

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0							
1	0+4	5+0						
2	0+6	5+4						
3	0+11	5+6						
4	0+12	5+11						
5	0+12	5+12						

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0							
1	0+4	5+0						
2	0+6	5+4	10					
3	0+11	5+6	10					
4	0+12	5+11	10					
5	0+12	5+12	10					

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0							
1	0+4	5+0						
2	0+6	5+4	10+0					
3	0+11	5+6	10+4					
4	0+12	5+11	10+6					
5	0+12	5+12	10+11					

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0							
1	0+4	5+0						
2	0+6	5+4	10+0					
3	0+11	5+6	10+4	11+0				
4	0+12	5+11	10+6	11+4	11+0			
5	0+12	5+12	10+11	11+6	11+4	11+0		

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0						0	
1	0+4	5+0						
2	0+6	5+4	10+0					
3	0+11	5+6	10+4	11+0				
4	0+12	5+11	10+6	11+4	11+0			
5	0+12	5+12	10+11	11+6	11+4	11+0		

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，
则对每个s_2值，有一种最优分配方案，
使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，
$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.
给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，
则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0						0	
1	0+4	5+0					5	
2	0+6	5+4	10+0					
3	0+11	5+6	10+4	11+0				
4	0+12	5+11	10+6	11+4	11+0			
5	0+12	5+12	10+11	11+6	11+4	11+0		

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，则对每个s_2值，有一种最优分配方案，使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.

给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0						0	
1	0+4	5+0					5	
2	0+6	5+4	10+0				10	
3	0+11	5+6	10+4	11+0				
4	0+12	5+11	10+6	11+4	11+0			
5	0+12	5+12	10+11	11+6	11+4	11+0		

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，则对每个s_2值，有一种最优分配方案，使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，$x_2, s_2 = 0, 1\ldots, 5, x_2 \le s_2$.

给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0						0	
1	0+4	5+0					5	
2	0+6	5+4	10+0				10	
3	0+11	5+6	10+4	11+0			14	
4	0+12	5+11	10+6	11+4	11+0			
5	0+12	5+12	10+11	11+6	11+4	11+0		

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，则对每个s_2值，有一种最优分配方案，使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，$x_2, s_2 = 0, 1\ldots, 5.$

给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0						0	
1	0+4	5+0					5	
2	0+6	5+4	10+0				10	
3	0+11	5+6	10+4	11+0			14	
4	0+12	5+11	10+6	11+4	11+0		16	
5	0+12	5+12	10+11	11+6	11+4	11+0	21	

解：（续）第2阶段：将 s_2 台设备全部分配给乙和丙，则对每个s_2值，有一种最优分配方案，使最大赢利值为 $f_2(s_2) = \max_{x_2}\{P_2(x_2) + f_3(s_2 - x_2)\}$, 其中，$x_2, s_2 = 0, 1\ldots, 5$.

给乙工厂 x_2 台，赢利为$P_2(x_2)$，余下的$s_2 - x_2$台给工厂丙，则它的赢利最大值为 $f_3(s_2 - x_2)$。

s_2	$P_2(x_2) + f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	x_2							
	0	1	2	3	4	5		
0	0+0						0	0
1	0+4	5+0					5	1
2	0+6	5+4	10+0				10	2
3	0+11	5+6	10+4	11+0			14	2
4	0+12	5+11	10+6	11+4	11+0		16	1, 2
5	0+12	5+12	10+11	11+6	11+4	11+0	21	2

解：（续）第1阶段：将 s_1（这里只有 $s_1=5$）台设备分配给甲、乙和丙，则最大赢利值为

$f_1(s_1)=\max_{x_1}\{P_1(x_1)+f_2(5-x_1)\}$, 其中，$x_1=0,1\ldots,5.$

给甲工厂 x_1台，赢利为$P_1(x_1)$，余下的s_1-x_1台给乙和丙，则赢利最大值为 $f_2(5-x_1)$。

s_1	$P_1(x_1)+f_2(5-x_1)$						$f_1(5)$	x_1^*
	x_1							
	0	1	2	3	4	5		
5								

解：（续）第1阶段：将 s_1（这里只有 $s_1=5$）台设备分配给甲、乙和丙，则最大赢利值为

$f_1(s_1)=\max_{x_1}\{P_1(x_1)+f_2(5-x_1)\}$, 其中，$x_1=0,1\ldots,5.$

给甲工厂 x_1台，赢利为$P_1(x_1)$，余下的s_1-x_1台给乙和丙，则赢利最大值为 $f_2(5-x_1)$。

s_1	$P_1(x_1)+f_2(5-x_1)$						$f_1(5)$	x_1^*
	x_1							
	0	1	2	3	4	5		
5	0	3	7	9	12	13		

解：（续）第1阶段：将 s_1（这里只有 s_1=5）台设备分配给甲、乙和丙，则最大赢利值为

$f_1(s_1)=\max_{x_1}\{P_1(x_1)+f_2(5-x_1)\}$, 其中，$x_1=0,1\ldots,5$.

给甲工厂 x_1台，赢利为$P_1(x_1)$，余下的s_1-x_1台给乙和丙，则赢利最大值为 $f_2(5-x_1)$。

s_1	$P_1(x_1)+f_2(5-x_1)$						$f_1(5)$	x_1^*
	x_1							
	0	1	2	3	4	5		
5	0+21	3+16	7+14	9+10	12+5	13+0		

解：（续）第1阶段：将 s_1（这里只有 $s_1=5$）台设备分配给甲、乙和丙，则最大赢利值为

$f_1(s_1)=\max_{x_1}\{P_1(x_1)+f_2(5-x_1)\}$, 其中，$x_1=0, 1\ldots, 5.$

给甲工厂 x_1台，赢利为$P_1(x_1)$，余下的s_1-x_1台给乙和丙，则赢利最大值为 $f_2(5-x_1)$。

s_1	$P_1(x_1)+f_2(5-x_1)$						$f_1(5)$	x_1^*
	x_1							
	0	1	2	3	4	5		
5	0+21	3+16	7+14	9+10	12+5	13+0	21	0, 2

s_1	$P_1(x_1)+ f_2(5 - x_1)$						$f_1(5)$	x_1^*
	0	1	2	3	4	5		
5	0+21	3+16	7+14	9+10	12+5	13+0	21	0, 2

s_2	$P_2(x_2)+ f_3(s_2 - x_2)$						$f_2(s_2)$	x_2^*
	0	1	2	3	4	5		
0	0						0	
1	0+4	5+0					5	1
2	0+6	5+4	10+0				10	2
3	0+11	5+6	10+4	11+0			14	2
4	0+12	5+11	10+6	11+4	11+0		16	1, 2
5	0+12	5+12	10+11	11+6	11+4	11+0	21	2

	$P_3(x_3)$						$f_3(s_3)$	x_3^*
	0	1	2	3	4	5		
0	0						0	
1		4					4	1
2			6				6	2
3				11			11	3
4					12		12	4
5						12	12	5

s_1	$P_1(x_1)+f_2(5-x_1)$						$f_1(5)$	x_1^*
	0	1	2	3	4	5		
5	0+21	3+16	7+14	9+10	12+5	13+0	21	0, 2

s_2	$P_2(x_2)+f_3(s_2-x_2)$						$f_2(s_2)$	x_2^*
	0	1	2	3	4	5		
0	0						0	
1	0+4	5+0					5	1
2	0+6	5+4	10+0				10	2
3	0+11	5+6	10+4	11+0			14	2
4	0+12	5+11	10+6	11+4	11+0		16	1, 2
5	0+12	5+12	10+11	11+6	11+4	11+0	21	2

	$P_3(x_3)$						$f_3(s_3)$	x_3^*
	0	1	2	3	4	5		
0	0						0	
1		4					4	1
2			6				6	2
3				11			11	3
4					12		12	4
5						12	12	5

最优分配方案1：

$x_1^*=0$，

$s_2=s_1-x_1^*=5-0=5$，

此时，$x_2^*=2$，

$s_3=s_2-x_2^*=5-2=3$，

故$x_3^*=s_3=3$.

甲分配 0台，乙分配 2台，丙分配 3台，总赢利为 21万元。

s_1	$P_1(x_1)+f_2(5-x_1)$						$f_1(5)$	x_1^*
	0	1	2	3	4	5		
5	0+21	3+16	7+14	9+10	12+5	13+0	21	0, 2

s_2	$P_2(x_2)+f_3(s_2-x_2)$						$f_2(s_2)$	x_2^*
	0	1	2	3	4	5		
0	0						0	
1	0+4	5+0					5	1
2	0+6	5+4	10+0				10	2
3	0+11	5+6	10+4	11+0			14	2
4	0+12	5+11	10+6	11+4	11+0		16	1, 2
5	0+12	5+12	10+11	11+6	11+4	11+0	21	2

s_3	$P_3(x_3)$						$f_3(s_3)$	x_3^*
	0	1	2	3	4	5		
0	0						0	
1		4					4	1
2			6				6	2
3				11			11	3
4					12		12	4
5						12	12	5

最优分配方案2：

$x_1^*=2$，

$s_2=s_1-x_1^*=5-2=3$，

此时，$x_2^*=2$，

$s_3=s_2-x_2^*=3-2=1$，

故$x_3^*=s_3=1$.

甲分配 2台，乙分配 2台，丙分配 1台，总赢利为 21万元。

s_1	$P_1(x_1)+f_2(5-x_1)$					$f_1(5)$	x_1^*
	0	1	2	3	4		
4	0+16	3+14	7+10	9+5	12+0	17	1, 2

s_2	$P_2(x_2)+f_3(s_2-x_2)$						$f_2(s_2)$	x_2^*
	0	1	2	3	4	5		
0	0						0	
1	0+4	5+0					5	1
2	0+6	5+4	10+0				10	2
3	0+11	5+6	10+4	11+0			14	2
4	0+12	5+11	10+6	11+4	11+0		16	1, 2
5	0+12	5+12	10+11	11+6	11+4	11+0	21	2

s_3	$P_3(x_3)$						$f_3(s_3)$	x_3^*
	0	1	2	3	4	5		
0	0						0	
1		4					4	1
2			6				6	2
3				11			11	3
4					12		12	4
5						12	12	5

当设备台数为4台时，

$x_1^*=1$，

$s_2=s_1-x_1^*=4-1=3$，

此时，$x_2^*=2$，

$s_3=s_2-x_2^*=3-2=1$，

故$x_3^*=s_3=1$.

甲分配 1台，乙分配 2台，丙分配 1台，总赢利为 17万元。

s_1	$P_1(x_1)+f_2(5-x_1)$					$f_1(5)$	x_1^*
	0	1	2	3	4		
4	0+16	3+14	7+10	9+5	12+0	17	1, 2

s_2	$P_2(x_2)+f_3(s_2-x_2)$						$f_2(s_2)$	x_2^*
	0	1	2	3	4	5		
0	0						0	
1	0+4	5+0					5	1
2	0+6	5+4	10+0				10	2
3	0+11	5+6	10+4	11+0			14	2
4	0+12	5+11	10+6	11+4	11+0		16	1, 2
5	0+12	5+12	10+11	11+6	11+4	11+0	21	2

s_3	$P_3(x_3)$						$f_3(s_3)$	x_3^*
	0	1	2	3	4	5		
0	0						0	0
1		4					4	1
2			6				6	2
3				11			11	3
4					12		12	4
5						12	12	5

当设备台数为4台时，

$x_1^*=2$，

$s_2=s_1-x_1^*=4-2=2$，

此时，$x_2^*=2$，

$s_3=s_2-x_2^*=2-2=0$，

故$x_3^*=s_3=0$.

甲分配 2台，乙分配 2台，丙分配 0台，总赢利为 17万元。

资源连续分配问题

- 资源分配问题是决策变量取离散值的一类分配问题
  - 销售店分配问题、投资分配问题、货物的分配问题
  - 只分配不考虑回收，又称为资源平行分配问题
- 资源连续分配问题：考虑资源回收利用
  - 决策变量为连续值

资源连续分配问题

设有数量为 s_1 的某种资源，可入 A 和B 两种生产。第一年若以数量 u_1 投入生产A，剩下的量 $s_1 - u_1$ 就投入生产 B，则可得收入为 $g(u_1)+h(s_1 - u_1)$，其中g 和 h为已知函数，且 $g(0)=h(0)=0$。

这种资源在投入 A、B 生产后，年终还可回收再投入生产。设年回收率分别为 $0<a<1$ 和 $0< b<1$，则在第一年生产后，回收的资源量合计为 $s_2 =au_1 +b(s_1 - u_1)$。第二年再将资源数量 s_2 中的 u_2 和$s_2 - u_2$分别再投入 A、B两种生产，第二年又可得到收入为 $g(u_2)+h(s_2 - u_2)$。如此继续进行 n 年，应当如何决定每年投入 A 生产的资源量 $u_1, u_2, \ldots, u_n$，才使总的收入最大?

资源连续分配问题

- 此问题写成静态规划问题为

$$\max z = g(u_1)+h(s_1-u_1)+ g(u_2)+h(s_2-u_2)+\ldots+g(u_n)+h(s_n-u_n)$$

$$s_2 = au_1+ b(s_1 - u_1)$$

$$s_3 = au_2+ b(s_2 - u_2)$$

$$\ldots\ \ldots$$

$$s_n = au_{n-1}+ b(s_{n-1} - u_{n-1})$$

$$0\leq u_i \leq s_i,\ i = 1, 2, \ldots, n$$

资源连续分配问题

下面用动态规划方法来处理。

- 状态变量 s_k：在第 k 阶段(第 k 年)可投入 A、B 两种生产的资源量。
- 决策变量u_k：在第 k 阶段(第 k年)用于A生产的资源量，则 $s_k - u_k$ 表示用于 B 生产的资源量。
- 状态转移方程：$s_{k+1} = au_k + b(s_k - u_k)$
- 最优值函数 $f_k(s_k)$：有资源量 s_k，从第 k 阶段至第 n 阶段采取最优分配方案进行生产后所得到的最大总收入。

动态规划递推关系式

- 因此，可写出动态规划的逆推关系式为

$$f_n(s_n) = \max_{0 \le u_n \le s_n} \{g(u_n) + h(s_n - u_n)\}$$

$$f_k(s_k) = \max_{0 \le u_k \le s_k} \{g(u_k) + h(s_k - u_k) + f_{k+1}[au_k + b(s_k - u_k)]\}$$

$$k = n\text{-}1, \ldots, 2, 1$$

最后求出 $f_1(s_1)$，即为所求问题的最大总收入。

问题举例

例2. 某种机器可在高低两种不同的负荷下进行生产。(1) 设机器在高负荷下生产的产量函数为 $g=8u_1$，其中 u_1 为投入生产的机器数量，年完好率为 $a=0.7$；

(2) 在低负荷下生产的产量函数为 $h=5y$，其中 y 为投入生产的机器数量，年完好率为 $b=0.9$。

假定开始生产时完好的机器数量 $s_1=1000$ 台，试问每年如何安排机器在高、低负荷下的生产，使在五年内生产的总产量最高。

	产量函数	年完好率
高负荷	$g=8u$	$a=0.7$
低负荷	$h=5y$	$b=0.9$

动态规划建模

	产量函数	年完好率
高负荷	$g=8u$	$a=0.7$
低负荷	$h=5y$	$b=0.9$

- 问题的动态规划模型：
  - 设阶段序数 k 表示年度。
  - 状态变量 s_k：第 k 年度初拥有的完好机器数量，亦为第 k-1年度末时的完好机器数量。
  - 决策变量 u_k：第 k 年度中分配高负荷下生产的机器数量，则 s_k-u_k为该年度中分配低负荷下生产的机器数量。
- 状态转移方程：

$$s_{k+1} = au_k + b(s_k - u_k) = 0.7u_k + 0.9(s_k - u_k)\ ,\quad k=1, 2, \ldots, 5$$

第 k 年度的产量$v_k(s_k, u_k)$为：

$$v_k = 8u_k + 5(s_k - u_k)$$

动态规划建模

- 状态转移方程为

$$s_{k+1} = au_k + b(s_k - u_k)$$

$$= 0.7u_k + 0.9(s_k - u_k)\ ,\quad k=1, 2, \ldots, 5$$

- k 段允许决策集合为

$$D_k(x_k) = \{ u_k \mid 0 \le u_k \le x_k \}$$

- 第 k 年度的产量 $v_k(x_k, u_k)$为：

$$v_k = 8u_k + 5(x_k - u_k)$$

故，指标函数为

$$V_{1,5} = \sum_{k=1}^{5} v_k(x_k, u_k)$$

动态规划递推式

- 令 $f_k(s_k)$表示由资源量 s_k 出发，从第k年开始到第 5 年度结束时所产生的产品的总产量的最大值。

因而有逆推关系式：

$$f_6(s_6) = 0$$

$$f_k(s_k) = \max_{u_k \in D_k(s_k)} \{ 8u_k + 5(s_k - u_k) + f_{k+1}[0.7u_k + 0.9(s_k - u_k)] \}$$

$$k = 1, 2, 3, 4, 5$$

$$f_6(s_6) = 0$$

$$f_k(s_k) = \max_{u_k \in D_k(s_k)} \{ 8u_k + 5(s_k - u_k) + f_{k+1}[0.7u_k + 0.9(s_k - u_k)] \}$$

$$k = 1, 2, 3, 4, 5$$

$k = 5$ 时，有

$$f_5(s_5) = \max_{0 \le u_5 \le s_5} \{ 8u_5 + 5(s_5 - u_5) + f_6[0.7u_5 + 0.9(s_5 - u_5)] \}$$

$$= \max_{0 \le u_5 \le s_5} \{ 8u_5 + 5(s_5 - u_5) \}$$

$$= \max_{0 \le u_5 \le s_5} \{ 3u_5 + 5s_5 \}$$

因为 f_5 是u_5的线性单调增函数，

得最大解 $u_5^* = s_5$，

故相应有 $f_5(s_5) = 8s_5$。

$k=4$ 时，有

$$f_4(s_4)=\max_{0\le u_4\le s_4}\{8u_4+5(s_4-u_4)+f_5[0.7u_4+0.9(s_4-u_4)]\}$$

$$=\max_{0\le u_4\le s_4}\{8u_4+5(s_4-u_4)+8[0.7u_4+0.9(s_4-u_4)]\}$$

$$=\max_{0\le u_4\le s_4}\{1.4u_4+12.2s_4\}$$

得最大解 $u_4^*=s_4$，故相应有 $f_4(s_4)=13.6s_4$。

依次类推，可求得

$u_3^*=s_3$，故相应有 $f_3(s_3)=17.5s_3$；

$u_2^*=0$，故相应有 $f_2(s_2)=20.8s_2$；

$u_1^*=0$，故相应有 $f_1(s_1)=23.7s_1$.

由 $s_1=1000$, 得 $f_1(s_1)=23700$台。

最优策略为$u_1^*=0, u_2^*=0, u_3^*=s_3, u_4^*=s_4, u_5^*=s_5$,

即前两年应把年初全部完好机器投入低负荷生产，

后三年应把年初全部完好机器投入高负荷生产，

这样所得的产量最高，其最高产量为23700台。

在得到整个问题的最优指标函数值和最优策略后，还需反过来确定每年年初的状态，即从始端向终端递推计算出每年年初完好机器数。

已知 $s_1 = 1000$，得

$s_2 = 0.7\,u_1^* + 0.9(s_1 - u_1^*) = 0.9s_1 = 900$台；

$s_3 = 0.7\,u_2^* + 0.9(s_2 - u_2^*) = 0.9s_2 = 810$台；

$s_4 = 0.7\,u_3^* + 0.9(s_3 - u_3^*) = 0.7s_3 = 567$台；

$s_5 = 0.7\,u_4^* + 0.9(s_4 - u_4^*) = 0.9s_4 = 397$台；

$s_6 = 0.7\,u_5^* + 0.9(s_5 - u_5^*) = 0.7s_5 = 278$台。

$$s_{k+1} = au_k + b(s_k - u_k)$$
$$= 0.7u_k + 0.9(s_k - u_k)\ ,\quad k = 1, 2, \ldots, 5$$

复合系统工作可靠性问题

- 若某种机器的工作系统由*N*个部件串联组成，只要有一个部件失灵，整个系统就不能正常工作。为提高系统工作的可靠性，在每一个部件上均装有主要元件的备用件，并且设计了备用元件自动投入装置。
- 显然，备用元件越多，整个系统正常工作的可靠性越大，但整个系统的成本、重量、体积均相应加大，工作精度也降低。
- 最优化问题是在考虑上述限制条件下，应如何选择各部件的备用元件数，使整个系统的工作可靠性最大？

问题定义

设部件 i (i=1, 2, …, n)上装有 u_i个备用元件时，正常工作的概率为 $p_i(u_i)$。

则整个系统正常工作的可靠性，可用它正常工作的概率衡量，即 $P=\prod_{i=1}^{n} p_i(u_i)$。

设一个部件 i 的备用元件费用为 c_i，重量为 w_i，要求总费用不超过c，总重量不超过w，

则静态规划模型为：

$$\max P = \prod_{i=1}^{n} p_i(u_i)$$

$$\sum_{i=1}^{n} c_i u_i \le c$$

$$\sum_{i=1}^{n} w_i u_i \le w$$

$u_i \ge 0$ 且为整数，i =1, 2, …, n

动态规划模型

- 状态变量：两个约束条件，选二维状态变量，采用两个状态变量符号 x_k, y_k 来表达，其中

  x_k：由第 k 个到第 n 个部件所容许使用的总费用。

  y_k：由第 k 个到第 n 个部件所容许具有的总重量。
- 决策变量：u_k 为部件 k 上装的备用元件数
- 状态转移方程：

$$x_{k+1} = x_k - u_k c_k,\quad y_{k+1} = y_k - u_k w_k \qquad (1 \le k \le n)$$

- 允许决策集合：

$$D_k(x_k, y_k) = \{u_k \mid 0 \le u_k \le \min([x_k / c_k], [y_k / w_k])\}$$

- 最优值函数 $f_k(x_k, y_k)$：由状态 x_k 和 y_k 出发，从部件 k 到部件 n 的系统的最大可靠性。

动态规划基本方程

- 整机可靠性的动态规划基本方程为：

$$f_k(x_k, y_k) = \max_{u_k \in D_k(x_k, y_k)} \{p_k(u_k) f_{k+1}(x_k - c_k u_k, y_k - w_k u_k)\}$$

$$(k = n, n\text{-}1, \ldots, 1)$$

$$f_{n+1}(x_{n+1}, y_{n+1}) = 1$$

- 边界条件为1：x_{n+1}，y_{n+1} 均为零时，装置不工作，故可靠性为1。
- 最后计算得 $f_1(c, w)$即为所求问题的最大可靠性。

复合系统工作可靠性问题

- 该问题的特点是：
  - 指标函数为连乘积形式，而不是连加形式，但仍满足递推关系；
  - 边界条件为1而不是零，这是由研究对象的特性所决定的。
- 在该问题中，如果静态模型的约束条件增加为三个，例如总体积不超过v，则状态变量就要选为三维的（x_k, y_k, z_k）
  - 说明静态规划问题的约束条件增加时，对应的动态规划的状态变量维数也需要增加，而决策变量维数可以不变。

回顾：动态规划

动态规划递推式：

$V_{k,n}[s_k, p_{k,n}]$

$=v_k(s_k, u_k)+V_{k+1,n}[s_{k+1}, p_{k+1,n}]$，$k = n, n\text{-}1, \ldots, 1$

动态规划求解：

$$f_k(s_k)= \mathrm{opt}_{u_k \in D_k(s_k)}\{v_k(s_k, u_k) + f_{k+1}(s_{k+1})\}$$

$$k = n, n\text{-}1, \ldots, 1$$

边界条件为 $f_{n+1}(s_{n+1}) = 0$。

其中 $s_{k+1}= T_k(s_k, u_k)$。

其求解过程，根据边界条件，从$k = n$开始，由后向前逆推，从而逐步可求得各段最优决策和相应的最优值，最后求出$f_1(s_1)$，就得到整个问题的最优解。

排序问题

- 设有n个工件需要在机床A、B 上加工，每个工件都必须经过先A而后B的两道加工工序。

设工件 i $(1 \le i \le n)$在A、B上的加工时间分别为a_i和b_i。

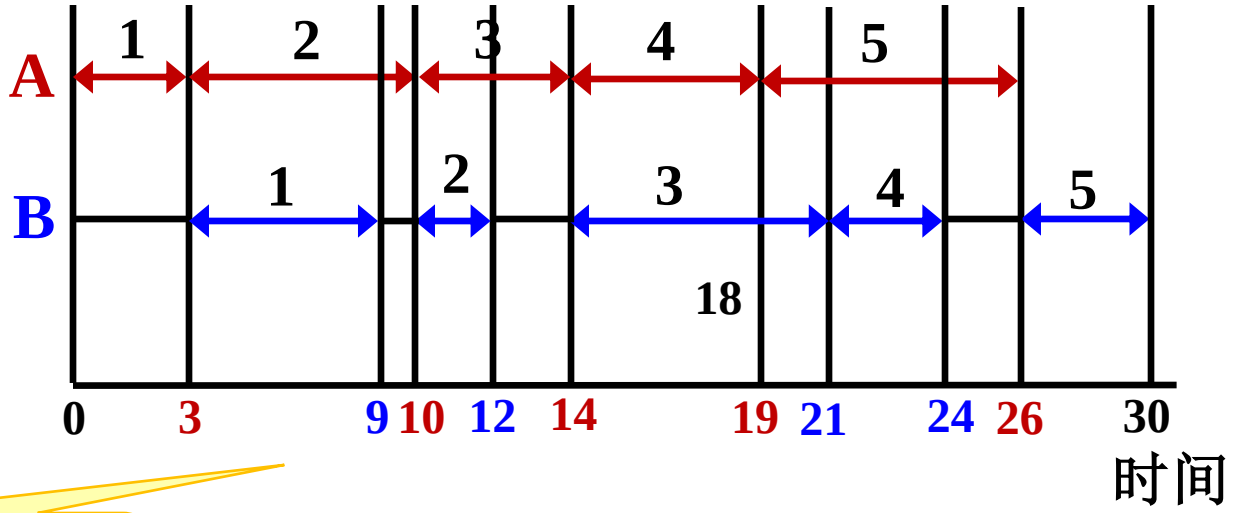
- 问：应如何在两机床上安排各工件加工的顺序，使在机床A上加工第一个工件开始到在机床 B上将最后一个工件加工完为止，所用的加工总时间最少？

机床A　　机床B

排序问题分析

- 当加工顺序取定之后，工件在 A 上加工时没有等待时间，而在 B 上则常常等待。

机床＼工件	1	2	3	4	5
A	3	7	4	5	7
B	6	2	7	3	4

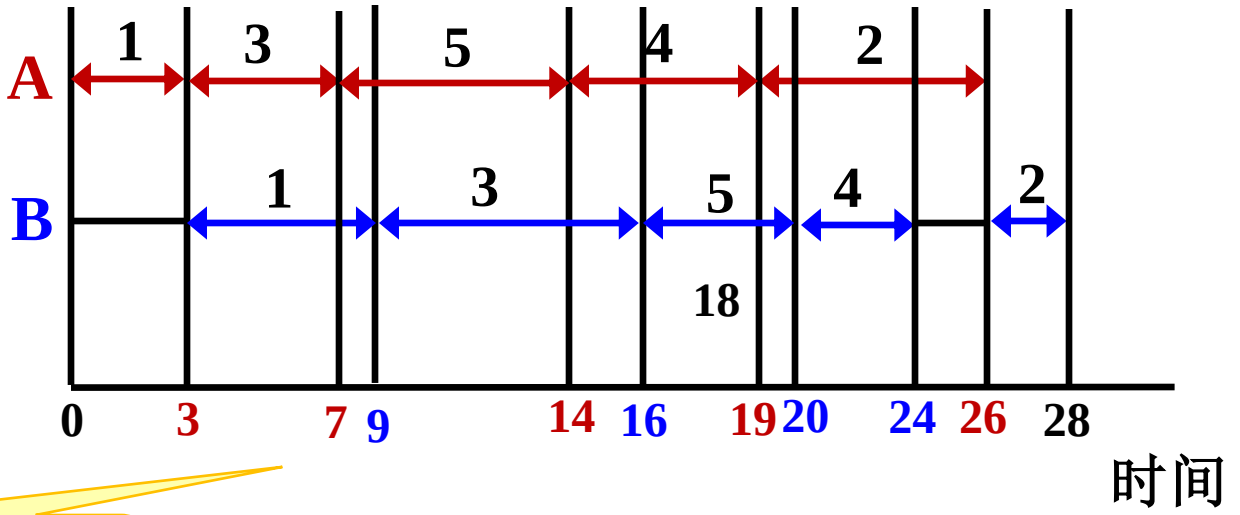


工件在B上等待时间：
3+1+2+2 = 8 小时

排序问题分析

- 当加工顺序取定之后，工件在 A 上加工时没有等待时间，而在 B 上则常常等待。
- 寻求最优排序方案只有尽量减少在 B上等待时间，才能使总加工时间最短。

工件 / 机床	1	2	3	4	5
A	3	7	4	5	7
B	6	2	7	3	4



工件在B上等待时间：
3+2 = 5 小时

排序问题分析

- 当加工顺序取定之后，工件在 A 上加工时没有等待时间，而在 B 上则常常等待。
- 寻求最优排序方案只有尽量减少在 B上等待时间，才能使总加工时间最短。
- 设第 *i* 个工件在机床 A上加工完毕以后，在B上要经过若干时间才能加工完, 故对同个工件来说，在 A, B上总是出现加工完毕的时间差，可用于描述加工状态.

机床A　　机床B

排序问题建模

- 以在机床 A上更换工件的时刻作为时段。
- X：在机床 A 上等待加工的按取定顺序排列的工件集合。
- x：不属于 X 的在 A上最后加工完的工件。
- t：在 A上加工完 x 的时刻算起到在 B上加工完 x 所需的时间。
  - x 在 B上的等待时间 + 加工时间。

因此，在A上每加工完一个工件，就有(X, t)与之对应。

- (X, t)：作为描述机床A、B在加工过程中的状态变量，则当 X包含有 s 个工件时，过程尚有s段，其时段数已隐含在状态变量之中。

排序问题建模

由状态 (X, t) 出发，

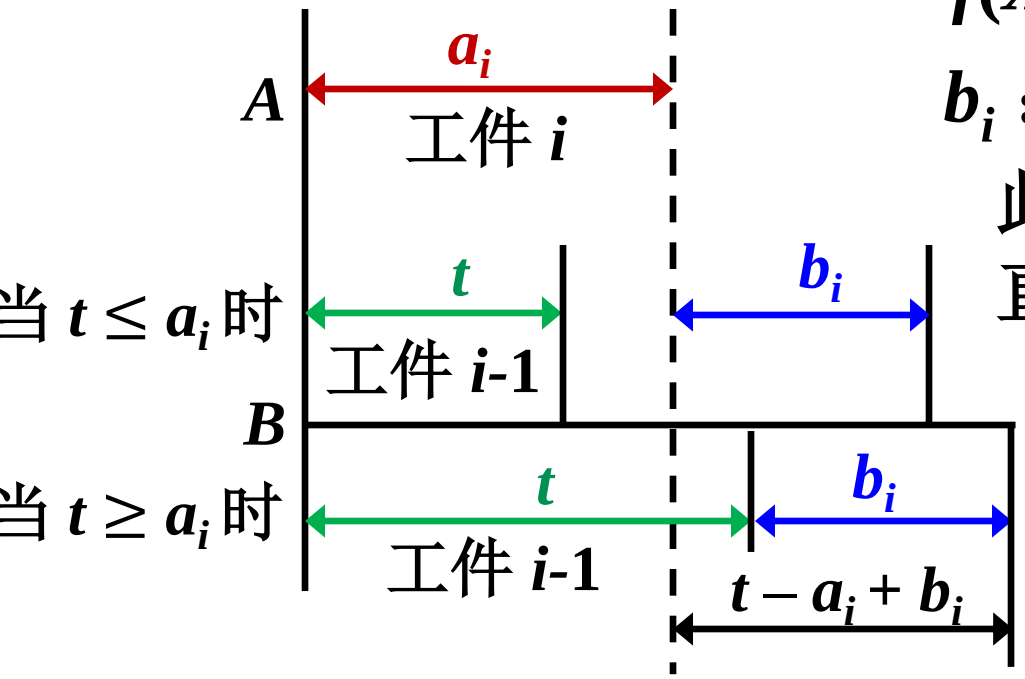
- $f(X, t)$：对未加工的工件采取最优加工顺序后，将 X 中所有工件加工完所需时间。
- $f(X, t, i)$：在A上加工工件 i，然后再对以后的加工工件采取最优顺序后，把 X中工件全部加工完所需要的时间。
- $f(X, t, i, j)$：在A上相继加工 i 与 j 后，对以后的加工工件采取最优顺序后，将 X 中工件全部加工完所需要的时间。

- **(*X*, *t*)**：机床*A*上等待加工的工件集合为*X*，其中*X*中第一个被执行的为 *i*；当*A*上加工完 *i*-1 后，到在*B*上加工完 *i*-1 所需时间为 *t*（*B*上的等待+加工时间），
- 当 $t \le a_i$（工件 *i* 在*A*上的加工时间）时，
  当 *A*上加工完 *i* 后，此时，*i*-1 在*B*上加工完成，*i* 将在 *B*上执行b_i时间。

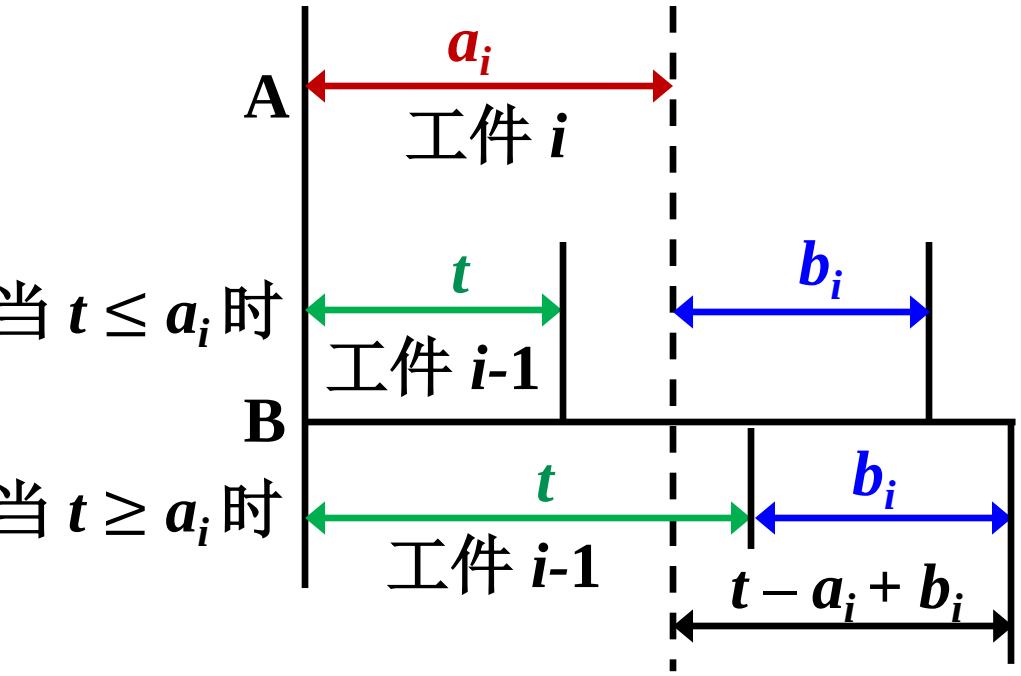
$$f(X, t, i) = a_i + f(X - \{i\}, b_i)$$

b_i：工件 *i* 在*B*上的加工时间

此时，*A*上加工完*i*后，直接在*B*上执行，无需等待



- **(*X*, *t*)**：机床*A*上等待加工的工件集合为*X*，其中*X*中第一个被执行的为 *i*；当*A*上加工完 *i*-1 后，到在*B*上加工完 *i*-1 所需时间为 *t*（*B*上的等待+加工时间）
- 当 $t \geq a_i$（工件 *i* 在*A*上的加工时间）时，
  当 A上加工完 *i* 后，此时，*i*-1 在*B*上未加工完成。
  i 将在B上等待 $t - a_i$时间后，执行 b_i 时间。



$$f(X, t, i) = a_i + f(X-\{i\}, t - a_i + b_i)$$

动态规划递推式

- 因此，有递推式

$$\text{则 } f(X,t,i)=\begin{cases} a_i+f(X-\{i\},\ t-a_i+b_i), & \text{当} t\geq a_i \text{时} \\ a_i+f(X-\{i\},\ b_i), & \text{当} t\leq a_i \text{时} \end{cases}$$

记 $z_i(t)=\max(t-a_i,0)+b_i$，

上式可合并为 $f(X,t,i)=a_i+f(X-\{i\},z_i(t))$。

由定义，可得

$$f(X,t,i,j)=a_i+a_j+f(X-\{i,j\},z_{ij}(t))$$

其中 $z_{ij}(t)$ 是在机床 A 上从 X 出发相继加工工件 i 和 j，并从 A 将 j 加工完的时刻算起，至在 B 上相继加工工件 i 和 j 并将所有工件加工完所需时间。

排序问题

- 故 $(X-\{i, j\}, z_{ij}(t))$ 是在 A 加工 i、j后所形成的新状态。

即在 A上加工 i, j后由状态 (X, t) 转移至 $(X-\{i, j\}, z_{ij}(t))$。

- 仿照 $z_i(t)$的定义，
  - 以 $X-\{i, j\}$代替$X-\{i\}$
  - 以 $z_i(t)$ 代替 t
  - 以 b_j代替 b_i，以 a_j代替 a_i

$$z_i(t) = \max(t - a_i, 0) + b_i$$

可得：
$$z_{ij}(t) = \max(z_i(t) - a_j, 0) + b_j$$
$$= \max[\max(t - a_i, 0) + b_i - a_j, 0] + b_j$$
$$= \max[\max(t - a_i - a_j + b_i, b_i - a_j), 0] + b_j$$
$$= \max[t - a_i - a_j + b_i + b_j, b_i + b_j - a_j, b_j]$$

排序问题

$$f(X, t, i, j) = a_i + a_j + f\,(X - \{ i, j\}, z_{ij}(t))$$

将 i、j 对调，可得

$$f(X, t, j, i) = a_i + a_j + f\,(X - \{ i, j\}, \; z_{ji}(t))$$

$$z_{ji}(t) = \max[t - a_i - a_j + b_i + b_j, \; b_i + b_j - a_i, \; b_i]$$

由于$f(X, t)$ 为 t 的单调上升函数，故当 $z_{ij}(t) \le z_{ji}(t)$时，

$$f(X, t, i, j) \le f(X, t, j, i)$$

由此，对任意 t，当 $z_{ij}(t) \le z_{ji}(t)$ 时， 工件 i 放在工件 j 之前加工可以使总的加工时间更短。

排序问题

$$z_{ij}(t) = \max[\, t - a_i - a_j + b_i + b_j\,,\ b_i + b_j - a_j,\ b_j\,]$$
$$z_{ji}(t) = \max[\, t - a_i - a_j + b_i + b_j,\ b_i + b_j - a_i,\ b_i\,]$$

- 由 $z_{ij}(t)$ 和 $z_{ji}(t)$ 的表示可知，若

$$\max(b_i + b_j - a_j,\ b_j) \le \max(b_i + b_j - a_i,\ b_i)$$

则有 $z_{ij}(t) \le z_{ji}(t)$。

将上式两边同减去 b_i 与 b_j，得

$$\max(-a_j,\ -b_i) \le \max(-a_i,\ -b_j)$$

即有 $\min(a_i,\ b_j) \le \min(a_j,\ b_i)$，

以上条件就是工件 i 应该排在工件 j 之前的条件。

最优排序规则

- 根据以上条件，得到最优排序规则如下：

1. 找出$a_1,a_2,\ldots\ldots,a_n,b_1,b_2,\ldots\ldots,b_n$中的最小数；
2. 若最小者为a_i，则将工件 i 排在第一位，并从工件集合中去掉这个工件；
3. 若最小者为 b_i，则将工件 i 排在最后一位，并从工件集合中去掉这个工件；
4. 对剩下的工件重复上述操作，直至工件集合为空。

- 主要思想：尽量减少在机床B上等待加工的时间，把在机床B上加工时间长的工件先加工，在B上加工时间短的工件后加工。

例：设有5个工件需在机床 A、B上加工，加工的顺序是先A后B，每个工件所需加工时间(单位:小时)如下表所示。问如何安排加工顺序，使机床连续加工完所有工件的加工总时间最少？并求出总加工时间。

工件号码	加工时间（小时）	
	机床 A	机床 B
1	3	6
2	7	2
3	4	7
4	5	3
5	7	4

根据最优排序规则，最优加工顺序为

例：设有5个工件需在机床 A、B上加工，加工的顺序是先A后B，每个工件所需加工时间(单位:小时)如下表所示。问如何安排加工顺序，使机床连续加工完所有工件的加工总时间最少？并求出总加工时间。

工件号码	加工时间（小时）	
	机床 A	机床 B
1	3	6
2	7	2
3	4	7
4	5	3
5	7	4

根据最优排序规则，最优加工顺序为

2

例：设有5个工件需在机床 A、B上加工，加工的顺序是先A后B，每个工件所需加工时间(单位:小时)如下表所示。问如何安排加工顺序，使机床连续加工完所有工件的加工总时间最少？并求出总加工时间。

工件号码	加工时间（小时）	
	机床 A	机床 B
1	3	6
2	7	2
3	4	7
4	5	3
5	7	4

根据最优排序规则，最优加工顺序为

1　　2

例：设有5个工件需在机床 A、B上加工，加工的顺序是先A后B，每个工件所需加工时间(单位:小时)如下表所示。问如何安排加工顺序，使机床连续加工完所有工件的加工总时间最少？并求出总加工时间。

工件号码	加工时间（小时）	
	机床 A	机床 B
1	3	6
2	7	2
3	4	7
4	5	3
5	7	4

根据最优排序规则，最优加工顺序为

1　　4　2

例：设有5个工件需在机床 A、B上加工，加工的顺序是先A后B，每个工件所需加工时间(单位:小时)如下表所示。问如何安排加工顺序，使机床连续加工完所有工件的加工总时间最少？并求出总加工时间。

工件号码	加工时间（小时）	
	机床 A	机床 B
1	3	6
2	7	2
3	4	7
4	5	3
5	7	4

根据最优排序规则，最优加工顺序为

1 3 4 2

例：设有5个工件需在机床 A、B上加工，加工的顺序是先A后B，每个工件所需加工时间(单位:小时)如下表所示。问如何安排加工顺序，使机床连续加工完所有工件的加工总时间最少？并求出总加工时间。

工件号码	加工时间（小时）	
	机床 A	机床 B
1	3	6
2	7	2
3	4	7
4	5	3
5	7	4

根据最优排序规则，最优加工顺序为

1 3 5 4 2

设备更新问题

- 在工业和交通运输企业中，经常碰到设备陈旧或损坏需要更新的问题：一种设备应该用多少年后进行更新为最恰当，即更新的最佳策略应该如何，从而使在某一时间内的总收入达到最大（或总费用达到最小）
- 一台设备的使用情况：
  - 随着使用年限的增加，机器的使用效率降低，收入减少，维修费用增加
  - 使用年限越长，机器本身的价值就越小，因而更新时所需的净支出费用就越多

问题描述

令 t 为机器已经使用过的年数(役龄)。

- $I_j(t)$：在第 j 年机器役龄为 t 年的机器运行所得收入。
- $O_j(t)$：在第 j 年机器役龄为 t 年的一台机器运行时所需的运行费用。
- $C_j(t)$：在第 j 年机器役龄为 t 年的一台机器更新时所需更新净费用。
- $g_j(t)$：在第 j 年开始使用一个役龄为 t 年的机器时，从第 j 年至第 n 年内的最佳收入。
- $x_j(t)$：给出 $g_j(t)$ 时，在第 j 年开始时的决策（保留或更新）。

问题描述

令 t 为机器已经使用过的年数（役龄）。

- α：折扣因子($0 \leq \alpha \leq 1$)，表示一年后的单位收入的价值视为现年的 α 单位
- T：在第一年开始时，正在使用的机器的役龄。
- n：计划的年限总数

问题描述

- 分为两种情况：
  - 若在第 j 年开始时购买了新机器，则有

第 j 年至第 n 年的总收入 =

在第 j 年由新机器获得的收入 $I_j(0)$

– 在第 j 年中的运行费用 $O_j(0)$

– 在第 j 年开始时役龄为 t 年的机器的更新净费用 $C_j(t)$

+ 在第 $j+1$ 年开始使用役龄为 1 年的机器从第 $j+1$ 年至第 n 年的最佳收入 $\alpha g_{j+1}(1)$

表示为：$I_j(0) - O_j(0) - C_j(t) + \alpha g_{j+1}(1)$

问题描述

- 分为两种情况：
  - 若在第 j 年开始时继续使用役龄为 t 年的机器，则

从第 j 年至第 n 年的总收入 =

在第 j 年由役龄为 t 年的机器获得的收入 $I_j(t)$

– 在第 j 年中役龄为 t 年的机器的运行费用 $O_j(t)$

+ 在第 $j+1$ 年开始使用役龄为 $t+1$ 年的机器从第 $i+1$ 年至第 n 年的最佳收入$\alpha g_{j+1}(t+1)$。

表示为： $I_j(t) - O_j(t) + \alpha g_{j+1}(t+1)$

- 比较两种情况的大小，选取大的，并相应得出是更新还是保留的决策。

动态规划递推式

- 即递推式为：

$$g_j(t) = \max\{ I_j(0) - O_j(0) - C_j(t) + \alpha g_{j+1}(1),\ I_j(t) - O_j(t) - + \alpha g_{j+1}(t+1) \}$$

$$j = 1, 2, \ldots, n;\ t = 1, 2, \ldots, j\text{-}1,\ j + T\text{-}1.$$

由于研究的是今后 n 年的计划，故还要求

$$g_{n+1}(t) = 0$$

可基于动态规划思想求解的问题与算法

- 最长公共子序列
- 矩阵链相乘
- 所有点对的最短路径
- 有向图的传递闭包
- 最优二叉查找树问题
- 0/1背包问题

最长公共子序列

- 给定字母表 Σ上的两个长度为 n和 m的字符串A和B

  令 $A= a_1a_2\ldots a_n$, $B=b_1b_2\ldots b_m$。

  - A的一个子序列是一个形如 $a_{i_1}a_{i_2}\ldots a_{i_k}$的字符串，其中$1\leq i_1<i_2<\ldots<i_k \leq n$。
  - 若一个字符串既是 A的子序列又是 B的子序列，则称它是A与B的公共子序列。

例： A=z x y x y z，B= x y y z x，

最长公共子序列

- 给定字母表 Σ上的两个长度为 n和 m的字符串A和B

  令 $A= a_1a_2\ldots a_n$, $B=b_1b_2\ldots b_m$。

  - A的一个子序列是一个形如 $a_{i_1}a_{i_2}\ldots a_{i_k}$的字符串，其中$1\leq i_1<i_2<\ldots<i_k \leq n$。
  - 若一个字符串既是 A的子序列又是 B的子序列，则称它是A与B的公共子序列。

例： $A=$z x y x y z，$B=$ x y y z x，

z 是A与B的长度为 1的公共子序列

最长公共子序列

- 给定字母表 Σ上的两个长度为 n和 m的字符串A和B

  令 $A= a_1a_2\ldots a_n$, $B=b_1b_2\ldots b_m$。

  - A的一个子序列是一个形如 $a_{i_1}a_{i_2}\ldots a_{i_k}$的字符串，其中$1\leq i_1<i_2<\ldots<i_k \leq n$。
  - 若一个字符串既是 A的子序列又是 B的子序列，则称它是A与B的公共子序列。

例： A=z x y x y z， B= x y y z x，

z 是A与B的长度为 1的公共子序列

xyy 是A与B的长度为 3的公共子序列

公共子序列

- 给定字母表 Σ上的两个长度为 n和 m的字符串A和B

  令 $A= a_1a_2\ldots a_n$, $B=b_1b_2\ldots b_m$。

  - A的一个子序列是一个形如 $a_{i_1}a_{i_2}\ldots a_{i_k}$的字符串，其中$1\leq i_1<i_2<\ldots<i_k \leq n$。
  - 若一个字符串既是 A的子序列又是 B的子序列，则称它是A与B的公共子序列。

例： A=z x y x y z， B= x y y z x，

z 是A与B的长度为 1的公共子序列

xyy 是A与B的长度为 3的公共子序列

$xyyz$ 是A与B的长度为 4的公共子序列

最长公共子序列问题

- 输入：字母表 Σ 上的两个字符串 $A=a_1a_2\ldots a_n$，$B=b_1b_2\ldots b_m$
- 输出：A 和 B的最长的公共子序列

例：$A=z\ x\ y\ x\ y\ z$，$B=x\ y\ y\ z\ x$，

z 是A与B的长度为 1的公共子序列

xyy 是A与B的长度为 3的公共子序列

$xyyz$ 是A与B的长度为 4的公共子序列，也是A与B的最长公共子序列

最长公共子序列问题算法

- 蛮力搜索的方法

  令$A= a_1a_2…a_n$和 $B= b_1b_2…b_m$。

  例举 A所有的2^n个子序列，

  对于每一个子序列在$\Theta(m)$ 时间内来确定它是否也是B的子序列。

  此方法的时间复杂性是 $\Theta(m2^n)$ 。

动态规划的递推式

- 令$L[i, j]$表示$A=a_1a_2\ldots a_i$和$B=b_1b_2\ldots b_j$的最长公共子序列的长度。

$L[i, j]$

A	a_1	a_2	…	a_{i-1}	a_i
B	b_1	b_2	…	b_{j-1}	b_j

- $a_i = b_j$

$L[i, j]$

$=\max\{ L[i-1, j-1]+1, L[i, j -1], L[i-1, j] \}$

$= L[i-1, j-1]+1$

$L[i - 1, j -1]+1$

A	a_1	a_2	…	a_{i-1}	a_i
B	b_1	b_2	…	b_{j-1}	b_j

$L[i, j -1]$

A	a_1	a_2	…	a_{i-1}	a_i
B	b_1	b_2	…	b_{j-1}	b_j

$L[i-1, j]$

A	a_1	a_2	…	a_{i-1}	a_i
B	b_1	b_2	…	b_{j-1}	b_j

动态规划的递推式

- 令$L[i, j]$表示$A=a_1a_2\ldots a_i$和$B=b_1b_2\ldots b_j$的最长公共子序列的长度。

$L[i, j]$

A	a_1	a_2	…	a_{i-1}	a_i
B	b_1	b_2	…	b_{j-1}	b_j

$L[i, j-1]$

A	a_1	a_2	…	a_{i-1}	a_i
B	b_1	b_2	…	b_{j-1}	b_j

- $a_i \neq b_j$

$L[i-1, j]$

A	a_1	a_2	…	a_{i-1}	a_i
B	b_1	b_2	…	b_{j-1}	b_j

$L[i, j]$

$=\max\{L[i, j-1], L[i-1, j]\}$

动态规划的递推式

- 令$L[i, j]$表示$a_1a_2\ldots a_i$和$b_1b_2\ldots b_j$的最长公共子序列的长度。
  - 如果i和j都大于0

  若$a_i = b_j$，$L[i, j] = L[i\text{-}1, j\text{-}1] + 1$；

  若$a_i \neq b_j$，$L[i, j] = \max\{ L[i, j\text{-}1], L[i\text{-}1, j] \}$。
  - 如果i或j等于0，$L[i, j] = 0$
- 计算A和B的最长公共子序列长度的递推式为：

$$L[i, j] = \begin{cases} 0, & \text{若} i = 0 \text{ 或 } j=0 \\ L[i\text{-}1, j\text{-}1] + 1, & \text{若} i > 0, j > 0, a_i = b_j \\ \max\{L[i, j\text{-}1], L[i\text{-}1, j]\}, & \text{若} i>0, j > 0, a_i \neq b_j \end{cases}$$

动态规划思想

- 令$L[i, j]$ 表示 $a_1a_2\ldots a_i$ 和 $b_1b_2\ldots b_j$ 的最长公共子序列的长度。
- 计算 A和B的最长公共子序列长度的递推式为：

$$L[i, j] = \begin{cases} 0, & \text{若} i = 0 \text{ 或 } j=0 \\ L[i\text{-}1, j\text{-}1] + 1, & \text{若} i > 0, j > 0, a_i = b_j \\ \max\{L[i, j\text{-}1], L[i\text{-}1, j]\}, & \text{若} i>0, j > 0, a_i \neq b_j \end{cases}$$

- 动态规划算法思想

对于任意 i 和j ， $0 \le i \le n$和 $0 \le j \le m$, 用一个 $(n+1)\times(m+1)$ 表计算$L[i, j]$的值，按照递推式逐行地填表 $L[0\ldots n, 0\ldots m]$。

最长公共子序列算法LCS

输入：字母表 Σ上的两个字符串A和B，长度分别为n 和m。
输出：A和B最长公共子序列的长度。

```
1.  for i←0 to n
2.      L[i, 0] ←0
3.  for j←0 to m
4.      L[0, j] ←0
5.  for i←1 to n
6.      for j←1 to m
7.          if a_i = b_j then L[i, j] ←L[i-1, j-1] + 1
8.          else L[i, j] ←max{ L[i, j-1], L[i-1, j]}
9.          end if
10.     end for
11. end for
12. return L[n, m]
```

算法复杂性正好是表的大小$\Theta(nm)$

最长公共子序列算法LCS

输入：字母表 Σ上的两个字符串A和B，长度分别为n 和m。
输出：A和B最长公共子序列的长度。

			j	
		$L[i-1, j-1]$		
i			$L[i, j]$	

1. for $i \leftarrow 0$ to n
2. $L[i, 0] \leftarrow 0$
3. for $j \leftarrow 0$ to m
4. $L[0, j] \leftarrow 0$
5. for $i \leftarrow 1$ to n
6. for $j \leftarrow 1$ to m
7. if $a_i = b_j$ then $L[i, j] \leftarrow L[i-1, j-1] + 1$
8. else $L[i, j] \leftarrow \max\{ L[i, j-1], L[i-1, j]\}$
9. end if
10. end for
11. end for
12. return $L[n, m]$

算法复杂性正好是表的大小$\Theta(nm)$

最长公共子序列算法LCS

输入：字母表 Σ上的两个字符串A和B，长度分别为n 和m。
输出：A和B最长公共子序列的长度。

			j	
			$L[i-1, j]$	
i		$L[i, j-1]$	$L[i, j]$	

1. for $i \leftarrow 0$ to n
2. $L[i, 0] \leftarrow 0$
3. for $j \leftarrow 0$ to m
4. $L[0, j] \leftarrow 0$
5. for $i \leftarrow 1$ to n
6. for $j \leftarrow 1$ to m
7. if $a_i = b_j$ then $L[i, j] \leftarrow L[i-1, j-1] + 1$
8. else $L[i, j] \leftarrow \max\{ L[i, j-1], L[i-1, j]\}$
9. end if
10. end for
11. end for
12. return $L[n, m]$

算法复杂性正好是表的大小$\Theta(nm)$

例：$A= x\ y\ x\ x\ z\ x\ y\ z\ x\ y$，$B = z\ x\ z\ y\ \ y\ z\ x\ x\ y\ x\ x\ z$

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

例：$A= x\ y\ x\ x\ z\ x\ y\ z\ x\ y$，$B = z\ x\ z\ y\ \ y\ z\ x\ x\ y\ x\ x\ z$

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

例：$A= x\ y\ x\ x\ z\ x\ y\ z\ x\ y$，$B = z\ x\ z\ y\ \ y\ z\ x\ x\ y\ x\ x\ z$

i \ j	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3	3	3
4	0	0	1	1	2	2	2	3	4	4	4	4	4
5	0	1	1	2	2	2	3	3	4	4	4	4	5
6	0	1	2	2	2	2	3	4	4	4	5	5	5
7	0	1	2	2	3	3	3	4	4	5	5	5	5
8	0	1	2	3	3	3	4	4	4	5	5	5	6
9	0	1	2	3	3	3	4	5	5	5	6	6	6
10	0	1	2	3	4	4	4	5	5	6	6	6	6

例：A= x y x x z x y z x y，B = z x z y y z x x y x x z

i \ j	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3	3	3
4	0	0	1	1	2	2	2	3	4	4	4	4	4
5	0	1	1	2	2	2	3	3	4	4	4	4	5
6	0	1	2	2	2	2	3	4	4	4	5	5	5
7	0	1	2	2	3	3	3	4	4	5	5	5	5
8	0	1	2	3	3	3	4	4	4	5	5	5	6
9	0	1	2	3	3	3	4	5	5	5	6	6	6
10	0	1	2	3	4	4	4	5	5	6	6	6	6

可基于动态规划思想求解的问题与算法

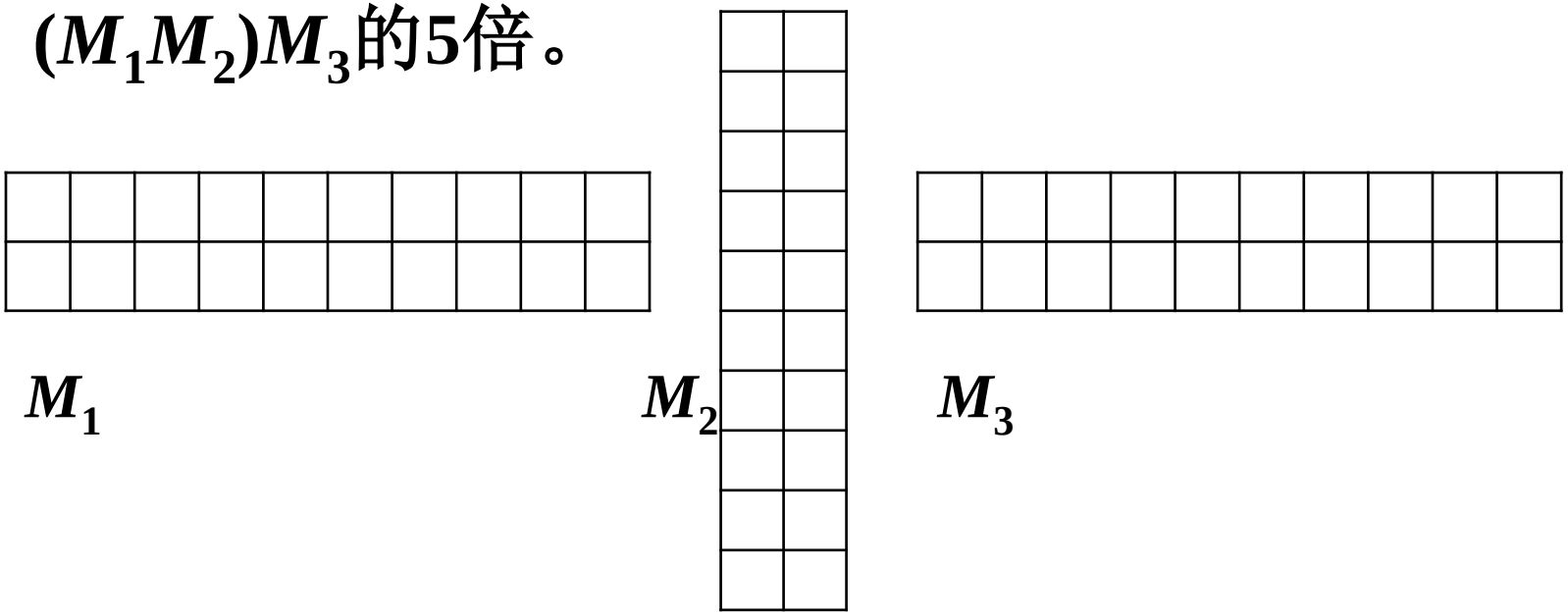
- 最长公共子序列
- 矩阵链相乘
- 所有点对的最短路径
- 有向图的传递闭包
- 最优二叉查找树问题
- 0/1背包问题

矩阵链相乘

例：给定维数分别是2×10, 10×2 和 2×10 的三个矩阵 M_1，M_2，M_3，用标准矩阵乘法计算乘积$M_1M_2M_3$。

- $(M_1M_2)M_3$：2×10×2 +2×2×10 = 80次乘法；
- $M_1(M_2M_3)$：10×2×10 + 2×10×10 = 400次乘法；

- 执行乘法 $M_1(M_2M_3)$ 耗费的时间是执行乘法 $(M_1M_2)M_3$的5倍。



矩阵链相乘

- 一般来说，n个矩阵$M_1, M_2, \ldots, M_n$ 链乘法的耗费，取决于 n-1个乘法执行的顺序
- 顺序数等于乘这n个矩阵时用每一种可能的途径放置括弧的方法数。

例： $(M_1 (M_2 (M_3 M_4)))$

$(M_1 ((M_2 M_3) M_4))$

$((M_1 (M_2 M_3)) M_4)$

$((M_1 M_2) (M_3 M_4))$

$(((M_1 M_2) M_3) M_4)$

矩阵链相乘

- 设 $f(n)$是求 n个矩阵乘积的所有放置括弧的方法数，
  假定要进行以下的乘法： $1 \le k \le n-1$

$$(M_1 M_2 \ldots M_k)\,(M_{k+1} M_{k+2} \ldots M_n)$$

则，对于前 k 个矩阵有 $f(k)$ 种方法放置括弧，

对于后 $n-k$ 个矩阵有 $f(n-k)$ 种方法放置括弧。

总共有 $f(k)f(n-k)$ 种放置括弧的方法。

因此， n个矩阵放置括弧的所有方法数为

$$f(n) = \sum_{k=1}^{n-1} f(k) f(n-k)$$

可以证明 $f(n) = \frac{1}{n}\binom{2n-2}{n-1}$为第$n$-1个Catalan数。

凸多边形三角形剖分方法计数

- 设 h_n表示用下面方法把凸多边形区域分成三角形区域的方法数：

  在有$n+1$条边的凸多边形区域内通过插入不相交的对角线，而把它分成三角形区域。

$n+1$个点

定义 $h_1=1$。

则 h_n满足如下递推关系：

$$h_n= h_1h_{n-1}+h_2h_{n-2}+\ldots+ h_{n-1}h_1 = \sum_{k=1}^{n-1} h_k h_{n-k} \ (n\geq 2)$$

该递推关系解为：$h_n= \frac{1}{n}\binom{2n-2}{n-1}$ $(n=1, 2, 3,\ldots)$

Catalan数 C_{n-1}

矩阵链相乘–递推式

$$f(n) = \frac{1}{n}\binom{2n-2}{n-1}$$

- 由Stirling公式：

$$n! \approx \sqrt{2\pi n}\left(n/e\right)^n \quad \text{其中} e=2.718\ 28\ldots$$

有

$$f(n) = \frac{1}{n}\binom{2n-2}{n-1} = \frac{(2n-2)!}{n((n-1)!)^2} \approx \frac{4^n}{4\sqrt{\pi}n^{1.5}}$$

- 因此 $f(n) = \Omega\left(\frac{4^n}{n^{1.5}}\right)$

矩阵链相乘中数量乘法的最少次数

- 对于每个 i，$1 \le i < n$，矩阵 M_i 的列数一定等于矩阵 M_{i+1} 的行数。
  - 令 $n+1$ 维数 $r_1, r_2, \ldots, r_{n+1}$，其中，$r_i$ 与 r_{i+1} 分别是 M_i 的行数和列数，$1 \le i \le n$
  - 用 $M_{i,j}$ 表示乘积 $M_i M_{i+1} \ldots M_j$，$1 \le i < j \le n$
  - 计算 $M_{i,j}$ 的耗费用数量乘法的次数 $C[i, j]$ 来测度

$$\underbrace{(M_i \quad \ldots \quad M_{k-1}}_{C[i,k-1]} \underbrace{(M_k}_{r_i r_k r_{j+1}} \underbrace{\quad \ldots \quad M_j)}_{C[k,j]}$$

$$C[i,j] = \min_{i<k\le j}\{C[i,k-1] + C[k,j] + r_i r_k r_{j+1}\}$$

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	7
1	C[1, 1]	C[1, 2]	C[1, 3]	C[1, 4]	C[1, 5]	C[1, 6]
2		C[2, 2]	C[2, 3]	C[2, 4]	C[2, 5]	C[2, 6]
3			C[3, 3]	C[3, 4]	C[3, 5]	C[3, 6]
4				C[4, 4]	C[4, 5]	C[4, 6]
5					C[5, 5]	C[5, 6]
6						C[6, 6]

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	6
1	C[1, 1]	C[1, 2]	C[1, 3]	C[1, 4]	C[1, 5]	C[1, 6]
2		C[2, 2]	C[2, 3]	C[2, 4]	C[2, 5]	C[2, 6]
3			C[3, 3]	C[3, 4]	C[3, 5]	C[3, 6]
4				C[4, 4]	C[4, 5]	C[4, 6]
5					C[5, 5]	C[5, 6]
6						C[6, 6]

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	6
1	C[1, 1]	C[1, 2]	C[1, 3]	C[1, 4]	C[1, 5]	C[1, 6]
2		C[2, 2]	C[2, 3]	C[2, 4]	C[2, 5]	C[2, 6]
3			C[3, 3]	C[3, 4]	C[3, 5]	C[3, 6]
4				C[4, 4]	C[4, 5]	C[4, 6]
5					C[5, 5]	C[5, 6]
6						C[6, 6]

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	6
1	C[1, 1]	C[1, 2]	C[1, 3]	C[1, 4]	C[1, 5]	C[1, 6]
2		C[2, 2]	C[2, 3]	C[2, 4]	C[2, 5]	C[2, 6]
3			C[3, 3]	C[3, 4]	C[3, 5]	C[3, 6]
4				C[4, 4]	C[4, 5]	C[4, 6]
5					C[5, 5]	C[5, 6]
6						C[6, 6]

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	6
1	C[1, 1]	C[1, 2]	C[1, 3]	C[1, 4]	C[1, 5]	C[1, 6]
2		C[2, 2]	C[2, 3]	C[2, 4]	C[2, 5]	C[2, 6]
3			C[3, 3]	C[3, 4]	C[3, 5]	C[3, 6]
4				C[4, 4]	C[4, 5]	C[4, 6]
5					C[5, 5]	C[5, 6]
6						C[6, 6]

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	6
1	C[1, 1]	C[1, 2]	C[1, 3]	C[1, 4]	C[1, 5]	C[1, 6]
2		C[2, 2]	C[2, 3]	C[2, 4]	C[2, 5]	C[2, 6]
3			C[3, 3]	C[3, 4]	C[3, 5]	C[3, 6]
4				C[4, 4]	C[4, 5]	C[4, 6]
5					C[5, 5]	C[5, 6]
6						C[6, 6]

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	6
1	C[1, 1]	C[1, 2]	C[1, 3]	C[1, 4]	C[1, 5]	C[1, 6]
2		C[2, 2]	C[2, 3]	C[2, 4]	C[2, 5]	C[2, 6]
3			C[3, 3]	C[3, 4]	C[3, 5]	C[3, 6]
4				C[4, 4]	C[4, 5]	C[4, 6]
5					C[5, 5]	C[5, 6]
6						C[6, 6]

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	6
1	C[1, 1]	C[1, 2]	C[1, 3]	C[1, 4]	C[1, 5]	C[1, 6]
2		C[2, 2]	C[2, 3]	C[2, 4]	C[2, 5]	C[2, 6]
3			C[3, 3]	C[3, 4]	C[3, 5]	C[3, 6]
4				C[4, 4]	C[4, 5]	C[4, 6]
5					C[5, 5]	C[5, 6]
6						C[6, 6]

只有一个矩阵

数量乘法的最少次数的递推式

- 执行矩阵乘法$M_1M_2\ldots M_n$所需的数量乘法的最小次数满足递推式：

$$C[1,n]=\min_{1<k\le n}\{C[1,k-1]+C[k,n]+r_1r_kr_n\}$$

$$C[i,j]=\min_{i<k\le j}\{C[i,k-1]+C[k,j]+r_ir_kr_{j+1}\},\ 1\le i<j\le n$$

	1	2	3	4	5	6	
1	0	$C[1,2]$	$C[1,3]$	$C[1,4]$	$C[1,5]$	$C[1,6]$	d_5
2		0	$C[2,3]$	$C[2,4]$	$C[2,5]$	$C[2,6]$	d_4
3			0	$C[3,4]$	$C[3,5]$	$C[3,6]$	d_3
4				0	$C[4,5]$	$C[4,6]$	d_2
5					0	$C[5,6]$	d_1
6						0	d_0

矩阵链相乘动态规划算法

算法MATCHAIN

输入： n个矩阵的链的维数对应于正整数数组$r[1\ldots n+1]$，其中$r[1\ldots n]$是n个矩阵的行数，$r[n+1]$是M_n的列数。

输出： n个矩阵相乘的数量乘法的最小次数。

	1	2	3	4	5	6	
1	0	$C[1,2]$	$C[1,3]$	$C[1,4]$	$C[1,5]$	$C[1,6]$	d_5
2		0	$C[2,3]$	$C[2,4]$	$C[2,5]$	$C[2,6]$	d_4
3			0	$C[3,4]$	$C[3,5]$	$C[3,6]$	d_3
4				0	$C[4,5]$	$C[4,6]$	d_2
5					0	$C[5,6]$	d_1
6						0	d_0

```
1. for i←1 to n        \\ 填充对角线d0
2.       C[i, i] ←0
3. end for
4. for d ←1 to n-1  \\ 填充对角线 d1到dn-1
5.       for i←1to n-d   \\ 填充对角线 di 的项目
6.             j←i+d    \\ 对角线上第 i 行元素的列数
7.             C[i, j] ←∞
8.             for k←i+1 to j       \\  i，j之间的索引
9.                   C[i, j] ←min{ C[i, j], C[i, k-1] + C[k, j] +r[i]r[k]r[j+1]}
10.            end for
11.      end for
12. end for
13. return C[1,n]
```

$\Theta(n^3)$时间

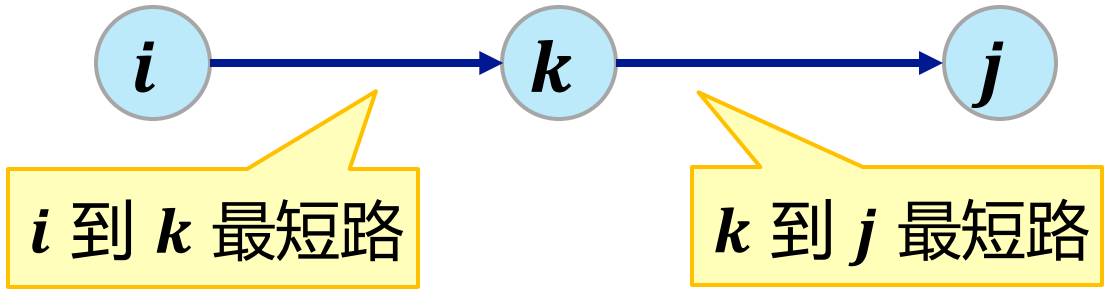
可基于动态规划思想求解的问题与算法

- 最长公共子序列
- 矩阵链相乘
- 所有点对的最短路径
- 有向图的传递闭包
- 最优二叉查找树问题
- 0/1背包问题

所有点对的最短路径问题

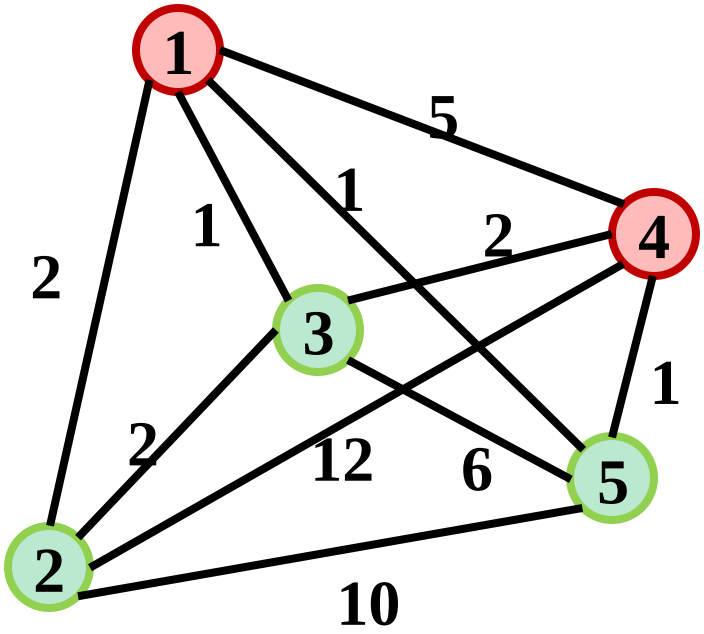
- 输入：有向图$G = (V, E)$，其中每条边(i, j)有一个非负的长度$l[i, j]$；如果从顶点i到顶点j没有边，则$l[i, j] = \infty$。
- 输出：每个顶点到其他所有顶点的距离（从顶点i到顶点j的距离是指从i 到j的最短路径的长度）。

- 性质：i到j的一条经过k的最短路径一定由i到k的一条最短路径和k到j的一条最短路径组成



最短路径松驰

例：假设 V ={1, 2, 3, 4, 5}

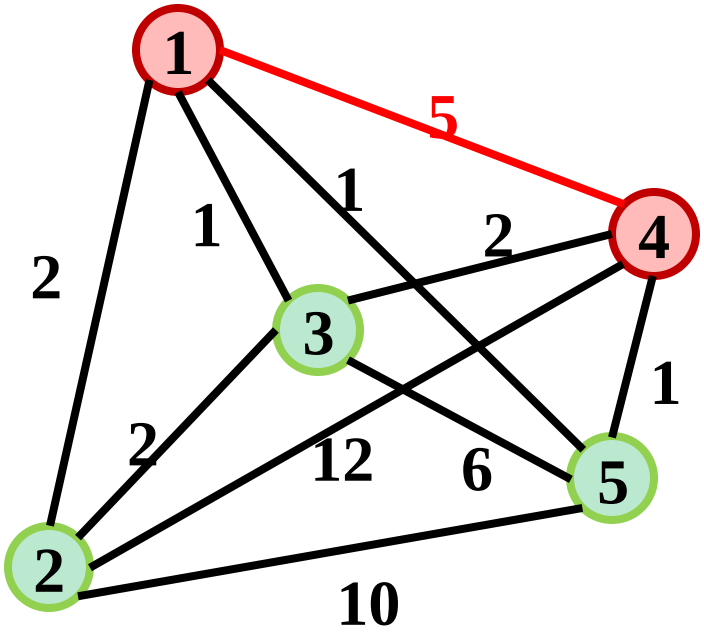


考虑顶点 1到 4的最短路径

可经过的点	路径	最短距离

最短路径松驰

例：假设 V ={1, 2, 3, 4, 5}

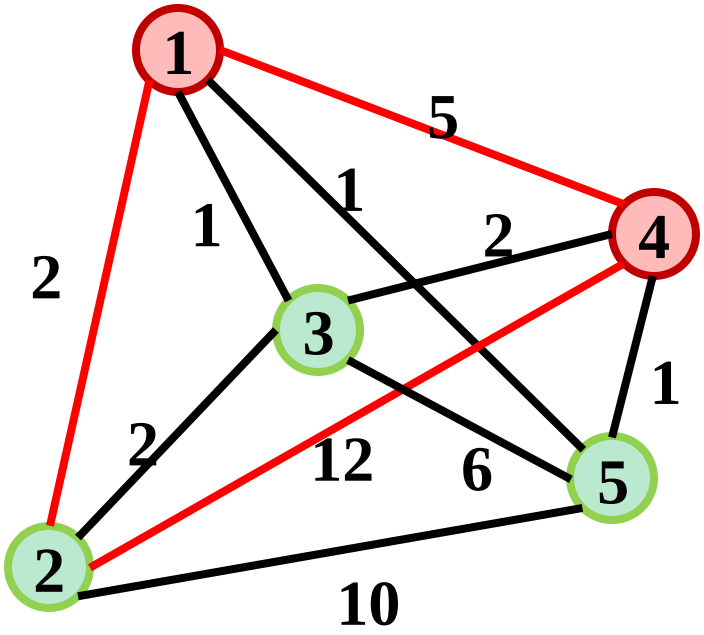


考虑顶点 1到 4的最短路径

可经过的点	路径	最短距离
{1}	1-4	5

最短路径松驰

例：假设 V ={1, 2, 3, 4, 5}

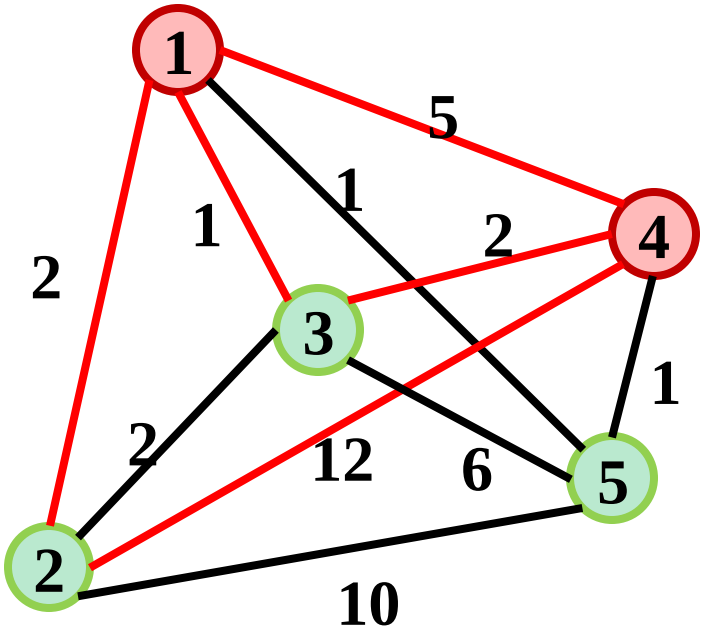


考虑顶点 1到 4的最短路径

可经过的点	路径	最短距离
{1}	1-4	5
{1, 2}	1-4, 1-2-4	5

最短路径松驰

例：假设 V ={1, 2, 3, 4, 5}

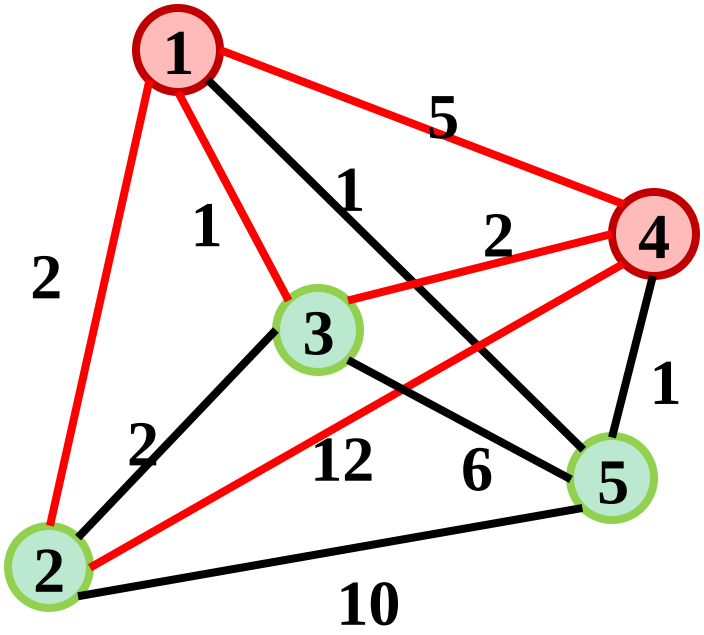


考虑顶点 1到 4的最短路径

可经过的点	路径	最短距离
{1}	1-4	5
{1, 2}	1-4, 1-2-4	5
{1, 2, 3}	1-4, 1-2-4, 1-3-4	3

最短路径松驰

例：假设 V ={1, 2, 3, 4, 5}

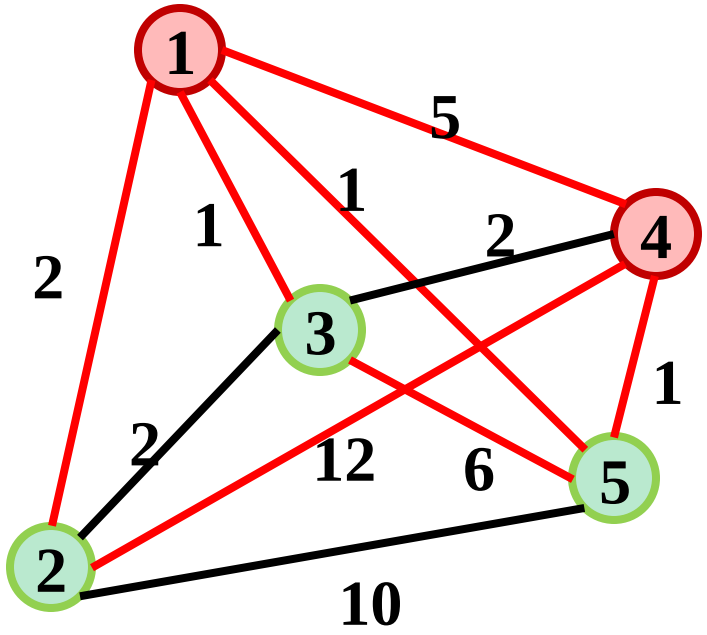


考虑顶点 1到 4的最短路径

可经过的点	路径	最短距离
{1}	1-4	5
{1, 2}	1-4, 1-2-4	5
{1, 2, 3}	1-4, 1-2-4, 1-3-4	3
{1, 2, 3, 4}	1-4, 1-2-4, 1-3-4,	3

最短路径松驰

例：假设 V ={1, 2, 3, 4, 5}



考虑顶点 1到 4的最短路径

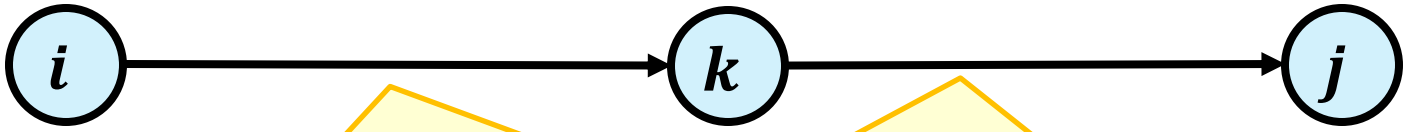
可经过的点	路径	最短距离
{1}	1-4	5
{1, 2}	1-4, 1-2-4	5
{1, 2, 3}	1-4, 1-2-4, 1-3-4	3
{1, 2, 3, 4}	1-4, 1-2-4, 1-3-4,	3
{1, 2, 3, 4, 5}	1-4, 1-2-4, 1-3-4, 1-5-4	2

随着可经过的顶点增加，最短路径长度会减少或保持不变

所有点对的最短路径问题

- 假设 $V=\{1, 2, \ldots, n\}$，设 i 和 j 是 V 中两个不同的顶点。

从 i 到 j 可经过前 k 个点的最短路径



从 i 到 k 可经过前 $k-1$ 个点的最短路

从 k 到 j 可经过前 $k-1$ 个点的最短路

- 定义 $d_{i,j}^{k}$ 是从 i 到 j，并且只经过 $\{1, 2, \ldots, k\}$ 中顶点的最短路径的长度。

  $d_{i,j}^{0} = l(i,j)$　　$d_{i,j}^{n}$ 为 顶点 i 到 j 的最短路径的长度

  - 如果最短路径不经过 k：$d_{i,j}^{k} = d_{i,j}^{k-1}$
  - 如果最短路径经过 k：$d_{i,j}^{k} = d_{i,k}^{k-1} + d_{k,j}^{k-1}$

动态规划递推式

- 定义 $d_{i,j}^{k}$ 是从 i 到 j，并且只经过 $\{1, 2, \ldots, k\}$ 中顶点的最短路径的长度。

则可递归地计算 $d_{i,j}^{k}$：

$$d_{i,j}^{k} = \begin{cases} l[i,j], & \text{若} k = 0 \\ \min\left\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\right\}, & \text{若} 1 \le k \le n \end{cases}$$

- Floyd-Warshall算法（1962年分别提出）

Floyd-Warshall算法

算法FLOYD

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

输出： 矩阵 D, 其中 $D[i,j]$为 i 到 j 的距离。

```
1.      D←l                 {将输入矩阵 l 复制到D}
2.      for k←1 to n
3.              for i←1 to n
4.                      for j←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
```

算法的运行时间是 $\Theta(n^3)$.

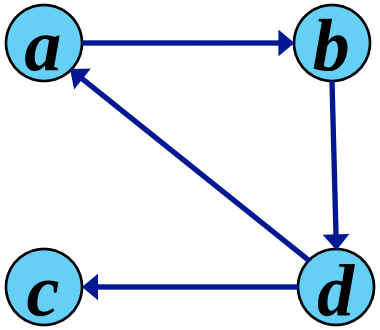
可基于动态规划思想求解的问题与算法

- 最长公共子序列
- 矩阵链相乘
- 所有点对的最短路径
- 有向图的传递闭包
- 最优二叉查找树问题
- 0/1背包问题

有向图的传递闭包

- 一个 n 顶点有向图的传递闭包可以定义为一个 n 阶布尔矩阵 $T=\{t_{ij}\}$，满足：

  如果从第 i 个顶点到第 j 个顶点之间存在一条有向路径，则矩阵 T 的第 i 行（$1 \le i \le n$）第 j 列（$1 \le j \le n$）的元素为 1，即 $t_{ij}=1$；否则，t_{ij} 为 0。



$$A=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&0&1&0\end{bmatrix}$$

邻接矩阵

$$T=\begin{bmatrix}1&1&1&1\\1&1&1&1\\0&0&0&0\\1&1&1&1\end{bmatrix}$$

传递闭包

计算有向图的传递闭包

- 一个 n 顶点有向图的传递闭包可以定义为一个 n 阶布尔矩阵 $T=\{t_{ij}\}$，满足：

  如果从第 i 个顶点到第 j 个顶点之间存在一条有向路径，则矩阵 T 的第 i 行（$1 \le i \le n$）第 j 列（$1 \le j \le n$）的元素为 1，即 $t_{ij}=1$；否则，t_{ij} 为 0。

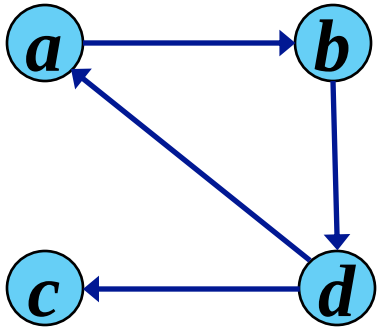
- 可以通过深度优先查找和广度优先查找生成有向图的传递闭包。
- 从顶点 i 为 起始点都能访问到可达的所有顶点，从而传递闭包的第 i 行的相应列置为 1。
- 但对同一个有向图遍历了多次（需以每个顶点为起点做一次遍历）。

Warshall 算法

- 1962年，由史蒂芬. 沃舍尔（S. Warshall）提出
  - 算法思想与所有点对的最短路径的Floyd-Warshall算法类似

设有向图 G的个顶为 1, 2, …, n，对任意两个顶点 i, j，从顶点 i 到顶点 j 有向路径 ρ，可以有以下情况：

- ρ 不经过中间顶点，即ρ是从 i到 j 的边；
- ρ的中间顶点仅为{1, 2, …, k }中的顶点， k=1, 2, …, n



Warshall 算法

- 定义 n行 n列的矩阵 $R^{(k)} = (r_{ij}^{(k)})_{n\times n}$ 满足：

$r_{ij}^{(k)} = 1$ 当且仅当从顶点 i到顶点 j存在一条长度大于0的有向路径且路径的每一个中间顶点的编号不大于k。

  - $R^{(0)}$ 就是有向图 G 的邻接矩阵。
  - $R^{(1)}$包含允许使用顶点 1作为中间顶点的路径信息
  - $R^{(2)}$包含允许使用顶点 1, 2 作为中间顶点的路径信息
  - $R^{(k)}$包含允许使用顶点 1, 2, …, k 作为中间顶点的路径信息, k=1, 2, …, n
  - $R^{(n)}$即为有向图 G 的传递闭包。

Warshall 算法

- **Warshall算法通过一系列 n 阶布尔矩阵来构造一个给定的 n 个顶点有向图的传递闭包：**

$$R^{(0)}, \ldots, R^{(k-1)}, R^{(k)}, \ldots, R^{(n)}$$

$r_{ij}^{(k)}=1 \Leftrightarrow$ 存在一条从顶点 i 到顶点 j 的路径，且路径中每一个中间顶点的编号都不大于 k

两种情况：

(1) 存在顶点 i 到顶点 j 的一条路径，且每个中间顶点的编号都不大于 k-1：$r_{ij}^{(k-1)}=1$，或者

(2) 顶点 i 到顶点 j 的路径的中间顶点的确包含顶点 k：

$$r_{ik}^{(k-1)}=1 \text{ 且 } r_{kj}^{(k-1)}=1$$

Warshall 算法

- 如何从矩阵$R^{(k-1)}$的元素生成矩阵$R^{(k-1)}$的元素？

$r_{ij}^{(k-1)}=1$ 当且仅当 $r_{ij}^{(k-1)}=1$，或 $r_{ik}^{(k-1)}=1$ 且 $r_{kj}^{(k-1)}=1$

得 $r_{ij}^{(k-1)} = r_{ij}^{(k-1)} \vee (r_{ik}^{(k-1)} \wedge r_{kj}^{(k-1)})$

- 规则：
  - 如果一个元素 r_{ij} 在 $R^{(k-1)}$ 中是1，它在$R^{(k)}$中仍然是1。
  - 如果一个元素 r_{ij} 在$R^{(k-1)}$中是 0，它在$R^{(k)}$中为 1当且仅当矩阵中第 i 行第 k列的元素和第 k行第 j列的元素都是1。

Warshall 算法

//输入：n个顶点有向图的邻接矩阵A
//输出：该有向图的传递闭包
$R^{(0)} \leftarrow A$
for $k \leftarrow 1$ **to** n **do**
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
      $R^{(k)}[i, j] \leftarrow R^{k-1}[i, j]$ **or** $R^{k-1}[i, k]$ **and** $R^{k-1}[k, j]$
return $R^{(n)}$

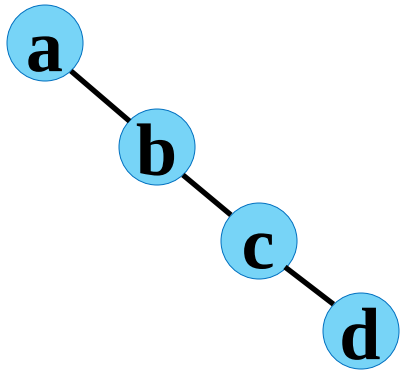
$\Theta(n^3)$; sparse graph? DFS? BFS?
比特串，位操作语句

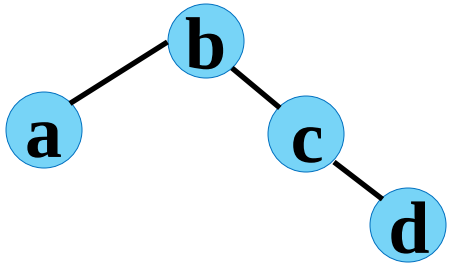
可基于动态规划思想求解的问题与算法

- 最长公共子序列
- 矩阵链相乘
- 所有点对的最短路径
- 有向图的传递闭包
- 最优二叉查找树问题
- 0/1背包问题

二叉查找树

- 二叉查找树 *T* 是一棵二叉树，且满足：对*T* 的任意一个顶点 *x*
  - *x* 的键值一定大于其左树中任一顶点的键值
  - *x* 的键值一定小于其右树中任一顶点的键值

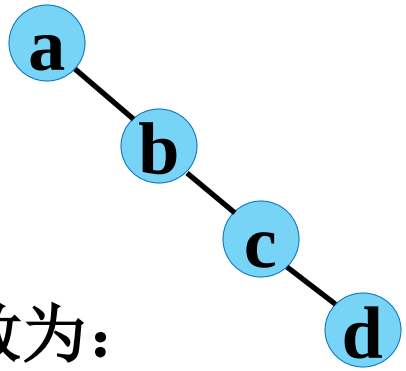




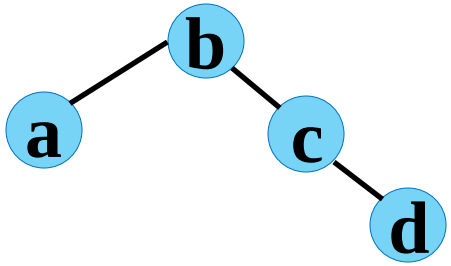
最优二叉查找树问题

- 输入：从小到大排列的互不相等的键 $a_1,\ldots a_n$，其查找概率分别为 $p_1, \ldots, p_n$。
- 输出：由键 $a_1,\ldots a_n$构成的最优二叉查找树 T_1^n，即在查找中的平均键值比较次数最低。

例：分别以概率 0.1, 0.2, 0.4, 0.3来查找4个键 a, b, c, d



平均键值比较次数为：
0.1×1+0.2× 2+ 0.4×3+0.3× 4
=2.9



平均键值比较次数为：
0.1×2+0.2× 1+ 0.4×2+0.3× 3
=2.1

最优二叉查找树:穷举法

- 穷举法不现实
  - 包含 n个键的二叉查找树的总数量等于第n个Catalan数

$$C(n) = \frac{1}{n+1}\binom{2n}{n},\ C(0) = 0$$

  - 以$4^n/n^{1.5}$的速度逼近无穷大。

1	2	3	4	5	6	7	8	9	10
1	2	5	14	42	132	429	1430	4682	16796

最优二叉查找树:递推式

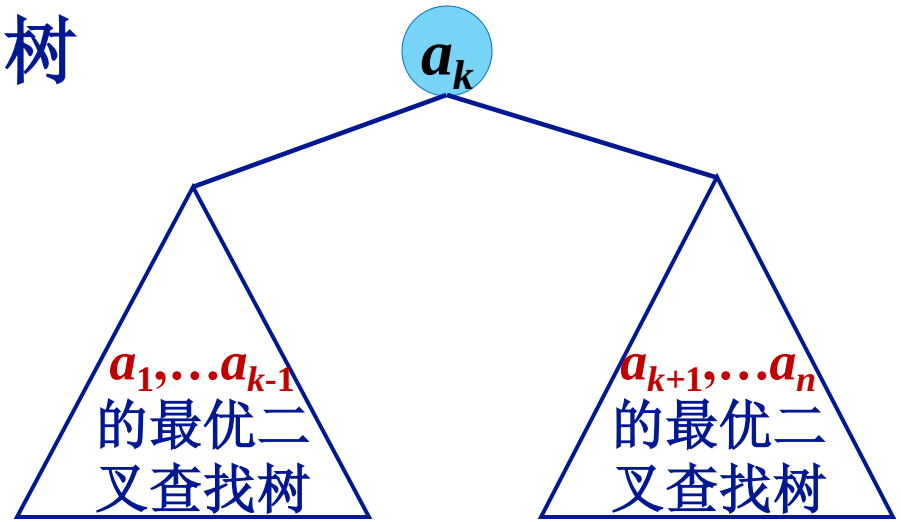
- 穷举法不现实
  - 包含 n个键的二叉查找树的总数量等于第n个Catalan数

$$C(n) = \frac{1}{n+1}\binom{2n}{n},\ C(n) = 0$$

  - 以$4^n/n^{1.5}$的速度逼近无穷大。

(1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796…)

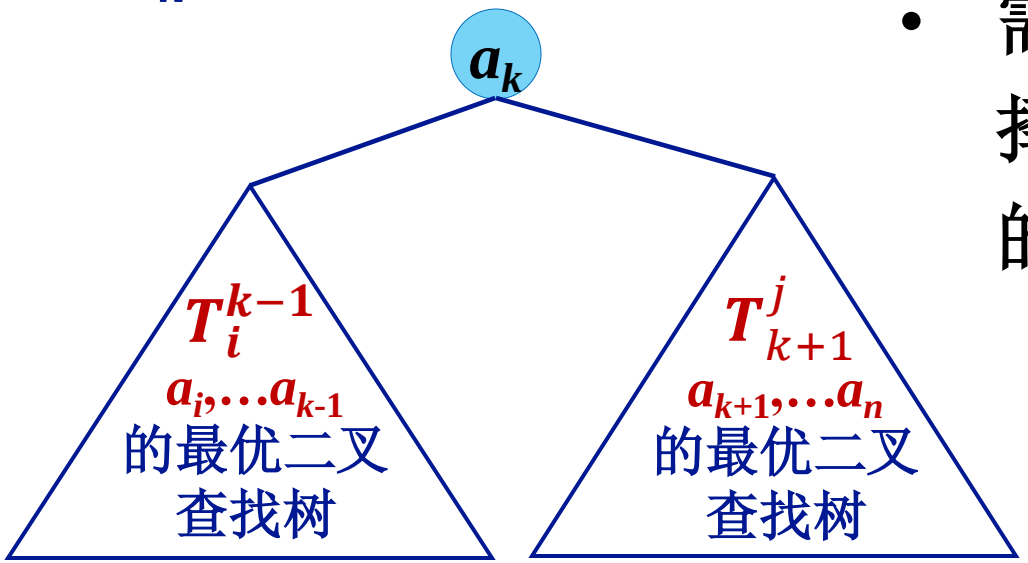
以 a_k 为根的二叉查找树



最优二叉查找树-递推式

- 设 T_i^j 是由键 $a_i, \ldots, a_j$ 构成的二叉树，$C[i, j]$是在这棵树中成功查找的最小的平均查找次数，$1 \le i \le j \le n$.
  - T_1^n 是键 $a_1,\ldots a_n$ 构成的最优二叉查找树
  - $C[1, n]$ 是T_1^n 中成功查找的最小的平均查找次数

以 a_k 为根的二叉查找树



- 需考虑从键 $a_i, \ldots, a_j$ 中选择一个根 a_k 的所有可能的方法

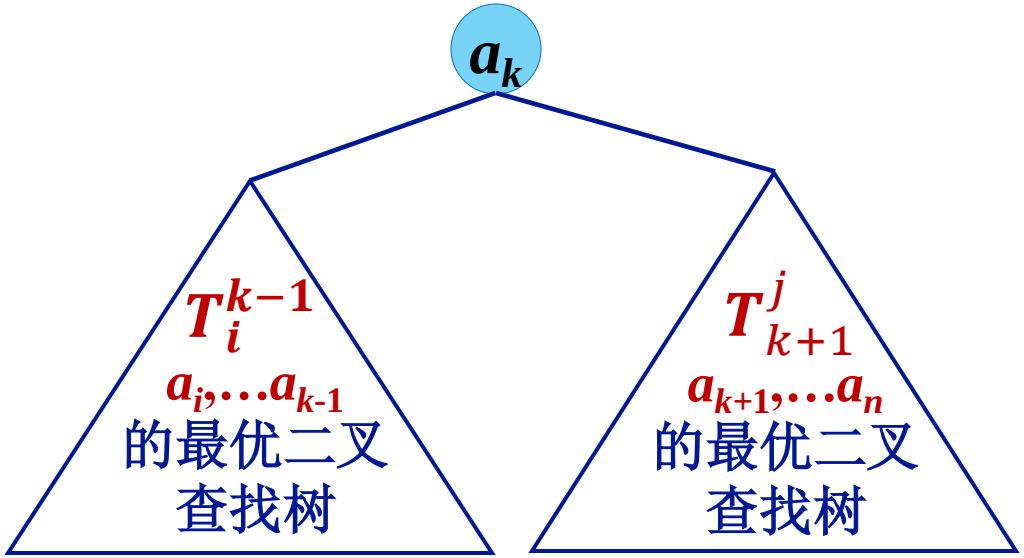
最优二叉查找树-递推式

- 令根节点 a_k 的层数为 0，有下列递推关系式：

设α_s 为 a_s 在T_i^{k-1}中的层数， β_s 为a_s在T_{k+1}^{j} 中的层数。

$$C(i,j) = \min_{i \le k \le j}\{p_k \times 1 + \sum_{s=i}^{k-1} p_s(\alpha_s + 1) + \sum_{s=k+1}^{j} p_s(\beta_s + 1)\}$$

以 a_k 为根的二叉查找树



最优二叉查找树-递推式

- 令根节点 a_k 的层数为 0，有下列递推关系式：

设α_s 为 a_s 在T_i^{k-1}中的层数， β_s 为a_s在T_{k+1}^{j} 中的层数）

$$C(i,j) = \min_{i\le k\le j}\{p_k \times 1 + \sum_{s=i}^{k-1} p_s(\alpha_s + 1) + \sum_{s=k+1}^{j} p_s(\beta_s + 1)\}$$

$$= \min_{i\le k\le j}\{\sum_{s=i}^{k-1} p_s\alpha_s + \sum_{s=k+1}^{j} p_s\beta_s + \sum_{s=i}^{j} p_s\}$$

$$= \min_{i\le k\le j}\{C(i,k-1) + C(k+1,j)\} + \sum_{s=i}^{j} p_s$$

因此，有下列递归式：当$1\le i \le j \le n$时，

$$C(i,j) = \min_{i\le k\le j}\{C(i,k-1) + C(k+1,j)\} + \sum_{s=i}^{j} p_s$$

最优二叉查找树-递推式

因此，有下列递归式：当$1 \le i \le j \le n$时，

$$C(i,j) = \min_{i \le k \le j}\{C(i,k-1) + C(k+1,j)\} + \sum_{s=i}^{j} p_s\}$$

- 假设 $1 \le i \le n+1$ 时， $C[i, i-1]=0$

  此时，为空树的比较次数。

- 当$1 \le i \le n$ 时， $C[i, i] = p_i$

  此时，为包含 a_i 的单节点二叉树。

最优二叉查找树-算法

算法*OptimalBST(P*[1..n])
输入：一个*n*个键的有序列表的查找概率数组*P*[1..*n*]
输出：最优BST中成功查找的平均比较次数，以及最优BST中子树的根表 *R*

```
for i← 1 to n do
    C[i, i-1] ←0
    C[i, i] ←P[i]
    R[i, i] ←i
C[n+1,n] ←0
for d←1 to n-1 do //对角线计数
    for i←1to n-d do
        j←i+d
        minval←∞
        for k←i to j do
            if C[i, k-1]+C[k+1, j]<minval
                minval←C[i,k-1]+C[k+1,j];kmin←k
            R[i, j] ←kmin
            sum←P[i];
            for s←i+1 to j do sum←sum+P[s]
            C[i,j] ←minval+sum
return C[1,n], R
```

$O(n^3)$

$$C(i,j) = \min_{i \le k \le j}\{C(i,k-1) + C(k+1,j)\} + \sum_{s=i}^{j} p_s\}$$

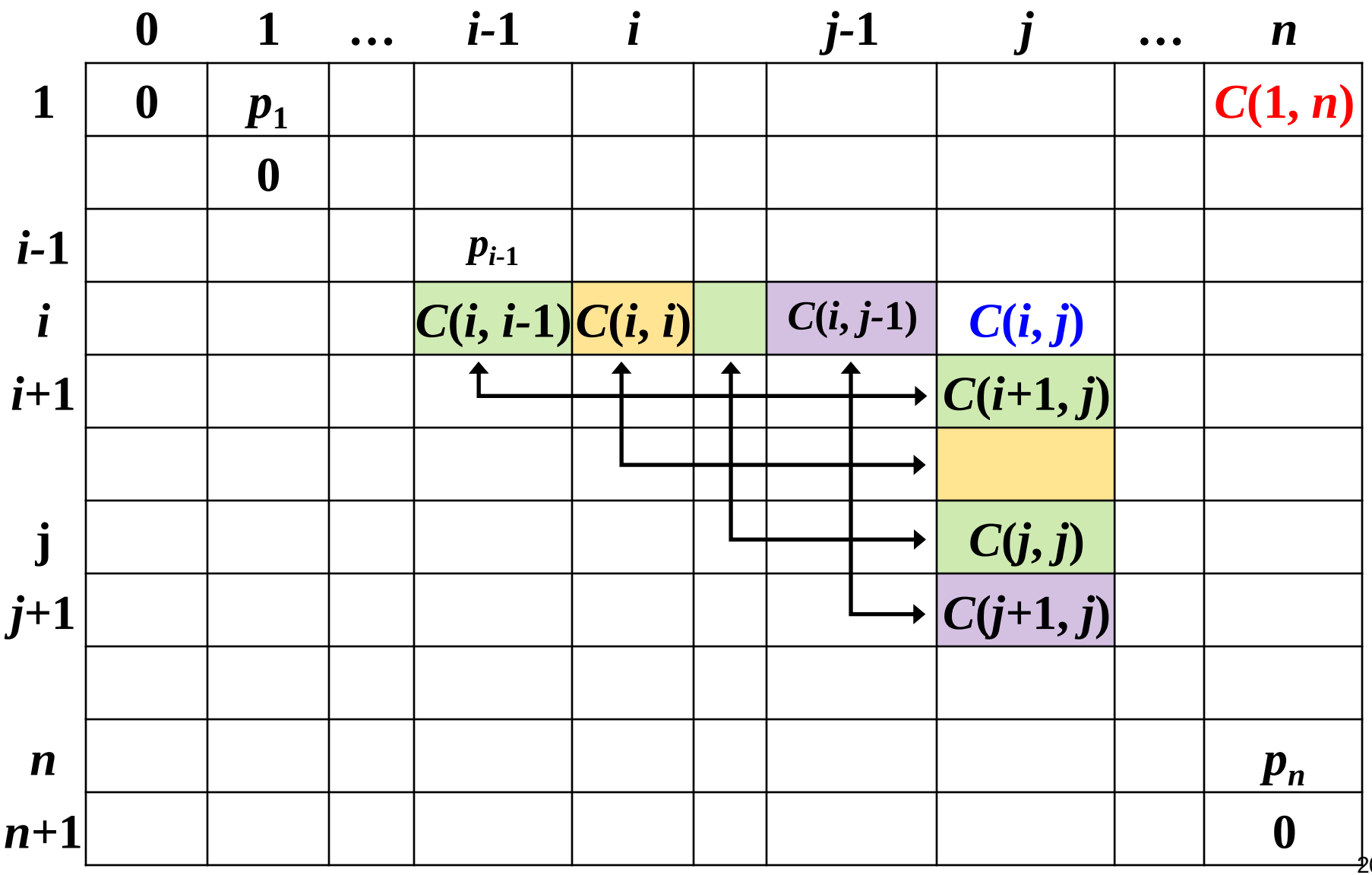
	0	1	…	i-1	i		j-1	j	…	n
1	0	p_1								$C(1, n)$
		0								
i-1				p_{i-1}						
i										
i+1										
j										
j+1										
n										p_n
n+1										0

$$C(i,j) = \min_{i \le k \le j}\{C(i,k-1) + C(k+1,j)\} + \sum_{s=i}^{j} p_s\}$$

	0	1	…	*i*-1	*i*		*j*-1	*j*	…	*n*
1	0	p_1								$C(1, n)$
		0								
i-1				p_{i-1}						
i				$C(i, i-1)$	$C(i, i)$		$C(i, j-1)$	$C(i, j)$		
i+1								$C(i+1, j)$		
j								$C(j, j)$		
j+1								$C(j+1, j)$		
n										p_n
n+1										0

计算$C(i, j)$需要 *i* 行位于 *j* 列左边的列的值以及 *j* 列中在 *i* 第下边的行中的值

$$C(i,j) = \min_{i \le k \le j}\{C(i,k-1) + C(k+1,j)\} + \sum_{s=i}^{j} p_s\}$$



举例说明

$$C(i,j)=\min_{i\le k\le j}\{C(i,k-1)+C(k+1,j)\}+\sum_{s=i}^{j}p_s\}$$

例：

键	a	b	c	d
查找概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	?		
2		0	0.2		
3			0	0.4	
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1			
2			2		
3				3	
4					4
5					

k=1: $C(1,0)+C(2,3)+p_1+p_2$= 0+0.2+0.1+0.2=0.5

k=2: $C(1,1)+C(3,2)+p_1+p_2$= 0.1+0+0.1+0.2=0.4

$C(1,2)=\min\{0.5, 0.4\}=0.4$

举例说明

$$C(i,j)=\min_{i\le k\le j}\{C(i,k-1)+C(k+1,j)\}+\sum_{s=i}^{j}p_s\}$$

例：

键	a	b	c	d
查找概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	0.4		
2		0	0.2	?	
3			0	0.4	
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	2		
2			2		
3				3	
4					4
5					

k=2: $C(2,1)+C(3,3)+p_2+p_3$= 0+0.4+0.2+0.4= 1.0

k=3: $C(2,2)+C(4,3)+p_2+p_3$= 0.2+0+0.2+0.4=0.8

$C(2,3)$ = min{1.0, 0.8} = 0.8

举例说明

$$C(i,j)=\min_{i\le k\le j}\{C(i,k-1)+C(k+1,j)\}+\sum_{s=i}^{j}p_s\}$$

例：

键	a	b	c	d
查找概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	0.4		
2		0	0.2	0.8	
3			0	0.4	
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	2		
2			2	3	
3				3	
4					4
5					

k=2: $C(2,1)+C(3,3)+p_2+p_3$= 0+0.4+0.2+0.4= 1.0

k=3: $C(2,2)+C(4,3)+p_2+p_3$= 0.2+0+0.2+0.4=0.8

$C(2,3)$ = min{1.0, 0.8} = 0.8

举例说明

$$C(i,j)=\min_{i\le k\le j}\{C(i,k-1)+C(k+1,j)\}+\sum_{s=i}^{j}p_s\}$$

例：

键	a	b	c	d
查找概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	0.4	?	
2		0	0.2	0.8	
3			0	0.4	1.0
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	2		
2			2	3	
3				3	3
4					4
5					

k=1: $C(1,0)+C(2,3)+p_1+p_2+p_3$= 0+0.8+0.1+0.2+0.4 = 1.5

k=2: $C(3,1)+C(3,3)+p_1+p_2+p_3$= 0.1+0.4+0.1+0.2+0.4=1.2

k=3: $C(1,2)+C(4,3)+p_1+p_2+p_3$= 0.4+0+0.1+0.2+0.4=1.1

举例说明

$$C(i,j)=\min_{i\le k\le j}\{C(i,k-1)+C(k+1,j)\}+\sum_{s=i}^{j}p_s\}$$

例：

键	a	b	c	d
查找概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	0.4	1.1	
2		0	0.2	0.8	
3			0	0.4	1.0
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	2	3	
2			2	3	
3				3	3
4					4
5					

k=1: $C(1,0)+C(2,3)+p_1+p_2+p_3$= 0+0.8+0.1+0.2+0.4 = 1.5

k=2: $C(3,1)+C(3,3)+p_1+p_2+p_3$= 0.1+0.4+0.1+0.2+0.4=1.2

k=3: $C(1,2)+C(4,3)+p_1+p_2+p_3$= 0.4+0+0.1+0.2+0.4=1.1

举例说明

$$C(i,j)=\min_{i\le k\le j}\{C(i,k-1)+C(k+1,j)\}+\sum_{s=i}^{j}p_s\}$$

例：

键	a	b	c	d
查找概率	0.1	0.2	0.4	0.3

主表

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2		0	0.2	0.8	1.4
3			0	0.4	1.0
4				0	0.3
5					0

根表

	0	1	2	3	4
1		1	2	3	3
2			2	3	3
3				3	3
4					4
5					

- 分别以概率 0.1, 0.2, 0.4, 0.3来查找4个键 a, b, c, d 的最优二叉树的平均键值比较次数等于 1.7

举例说明

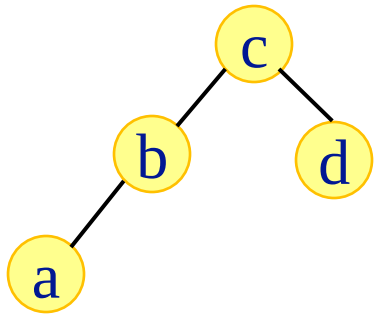
$$C(i,j)=\min_{i\le k\le j}\{C(i,k-1)+C(k+1,j)\}+\sum_{s=i}^{j}p_s\}$$

例：

键	a	b	c	d
查找概率	0.1	0.2	0.4	0.3

根表

	0	1	2	3	4
1		1	2	3	3
2			2	3	3
3				3	3
4					4
5					



$R(1,4)=3$, $R(1,2)=2$, $R(4,4)=4$

可基于动态规划思想求解的问题与算法

- 最长公共子序列
- 矩阵链相乘
- 所有点对的最短路径
- 有向图的传递闭包
- 最优二叉查找树问题
- **0/1背包问题**

0/1 背包问题

- 输入：n 项物品的集合$U=\{u_1, u_2,\ldots, u_n\}$，其中物品 u_j 的体积为 s_j，价值为 $v_j(1\leq j\leq n)$，以及背包容量 C。

- 输出：物品子集 $S\subseteq U$，满足 S 中的物品的总体积不超过 C，即 $\sum_{u_j\in S} s_j \leq C$，且总价值 $\sum_{u_j\in S} v_j$ 最大。

0/1 背包问题递推式

设有 n 项物品的集合$U=\{u_1, u_2,\ldots, u_n\}$，其中物品 u_j 的体积为 s_j，价值为 v_j $(1\le j\le n)$，以及背包容量 C

- $V[i, j]$: 从前 i 项 $\{u_1, u_2, \ldots, u_i\}$ 中取出来的装入体积为 j 的背包的物品的最大价值，$0\le i \le n$，$0\le j \le C$
  - 要寻求的是值$V[n, C]$。
  - $V[0, j]= 0$：当背包中没有物品，$0\le j \le C$
  - $V[i, 0]=0$：没有物品可放到容积为0的背包里，

    $0\le i \le n$

0/1 背包问题递推式

设有 n 项物品的集合$U=\{u_1, u_2,\ldots, u_n\}$，其中物品 u_j 的体积为 s_j，价值为 v_j $(1\le j\le n)$，以及背包容量 C

- $V[i, j]$: 从前 i 项 $\{u_1, u_2, \ldots, u_i\}$ 中取出来的装入体积为 j 的背包的物品的最大价值，$0\le i \le n$，$0\le j \le C$
  - 当 i 和 j 都大于0时

0/1 背包问题递推式

设有 n 项物品的集合$U=\{u_1, u_2,\ldots, u_n\}$，其中物品 u_j 的体积为 s_j，价值为 $v_j(1\le j\le n)$，以及背包容量 C

- $V[i, j]$: 从前 i 项 $\{u_1, u_2, \ldots, u_i\}$ 中取出来的装入体积为 j 的背包的物品的最大价值，$0\le i \le n$，$0\le j \le C$
  - 当 i 和 j 都大于0时，分为两种情况：

$V[i-1, j]$： $\{u_1, u_2, \ldots, u_{i-1}\}$的物品去装入体积为 j 的背包所得到的价值最大值；

$V[i-1, j-s_i]+v_i$： $\{u_1, u_2, \ldots, u_{i-1}\}$的物品去装入体积为 $j - s_i$ 的背包所得到的价值最大值加上物品 u_i 的价值。

取两者最大值：$V[i, j]=\max\{V[i-1, j], V[i-1, j-s_i]+v_i\}$

0/1 背包问题递推式

- 得递推式：

$$V[i,j]=\begin{cases}0, & \text{若 } i=0 \text{ 或 } j=0\\ V[i-1,j], & \text{若 } j<s_i\\ \max\{V[i-1,j],V[i-1,j-s_i]+v_i\}, & \text{若 } j\geq s_i\end{cases}$$

- 用动态规划来求解。
  - 用一个$(n+1)\times(C+1)$的表来计算$V[i,j]$的值，只需利用上面的公式逐行地填表$V[0\ldots n, 0\ldots C]$即可。

0/1 背包问题

算法KNAPSACK

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

输出：$\sum_{u_j \in S} v_j$的最大总价值，且满足$\sum_{u_j \in S} s_j \leq C$，其中$S \subseteq U$。

```
1. for i←0 to n
2.         V[i, 0] ←0
3. for j←0 to C
4.         V[0, j] ←0
5. for i←1 to n
6.         for j←1 to C
7.                 V[i, j] ←V[i-1, j]
8.                 if s_i ≤ j then V[i, j] ←Max{V[i, j], V[i-1, j-s_i]+v_i}
9. return V[n, C]
```

伪多项式时间算法

$\Theta(nC)$

总结

- 多阶段决策问题
- 动态规划的最优性定理与最优性原理
- 动态规划的递推关系式
- 重复子问题与最优子问题结构