算法分析与设计 II

2022-2023-2

数学与计算机学院
数据科学与大数据技术

Last Modified: 2023.1.16

4. 动态规划

4.1 基础动态规划问题

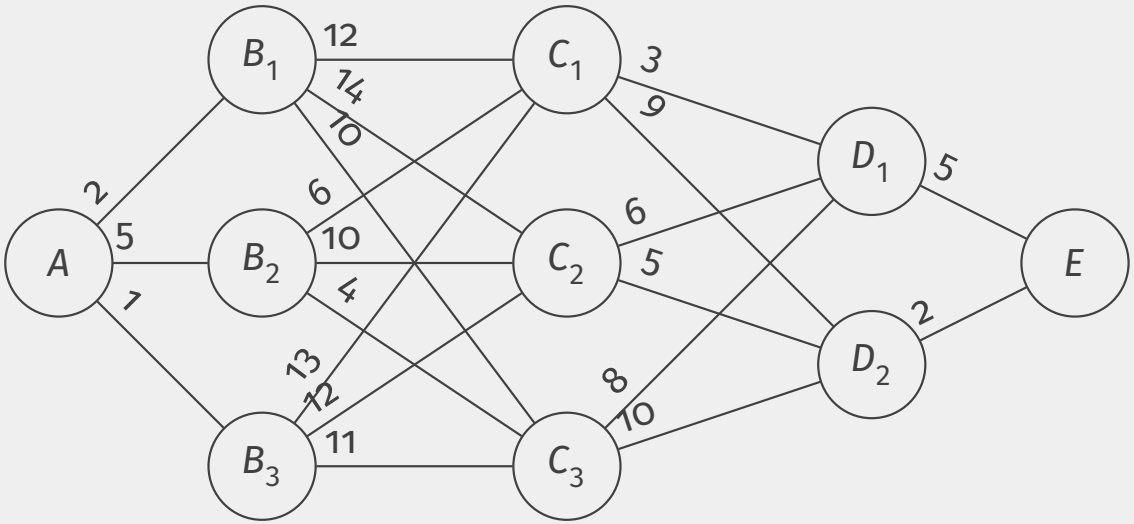
1. 动态规划的一个特征是问题有重叠的子问题，与分治算法类似，可以将原问题分解为子问题，使用相同的规则进行求解

2. 动态规划的另一个重要特征是最优子结构(Optimal substructure)，依据的是贝尔曼最优化原理 (Bellman's principle of optimality)：作为整个过程的最优策略具有这样的性质，无论过去的状态和决策如何，对先前决策所形成的状态而言，余下的诸决策必构成最优策略

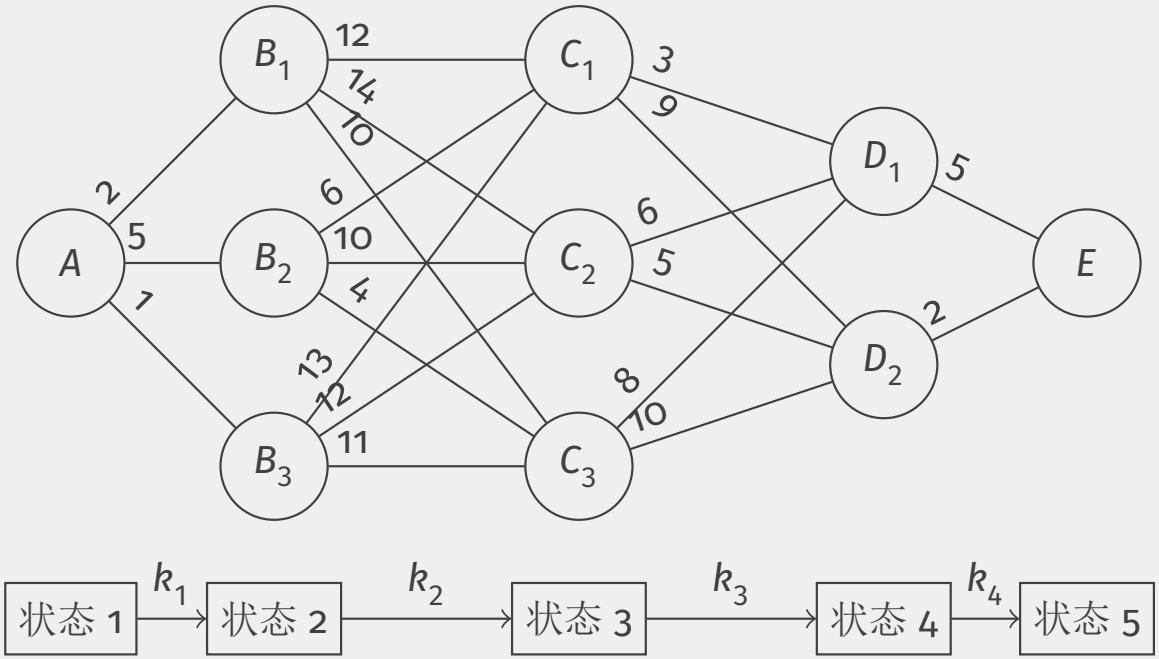
- 以上两点也是判断一个问题是否适用动态规划的主要依据

解最短路问题

(1) 阶段 (stage): k



解最短路问题

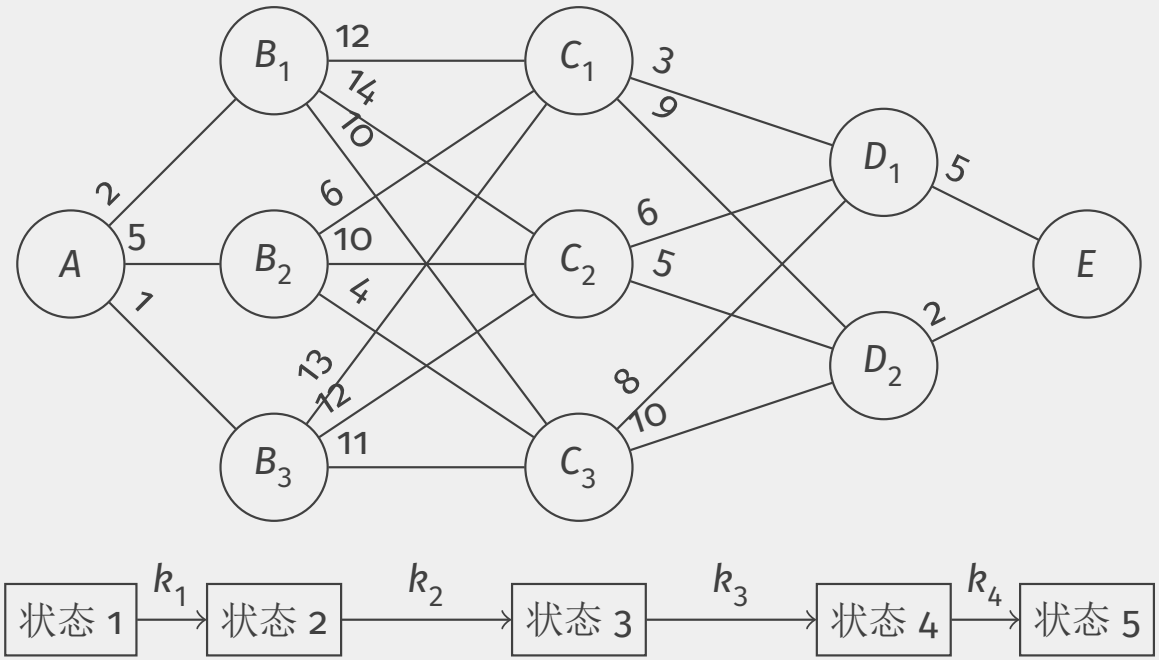


(1) 阶段 (stage): k

(2) 状态 (state): s_k

阶段 1	$s_1 = [A]$
阶段 2	$s_2 = [B_1, B_2, B_3]$
阶段 3	$s_3 = [C_1, C_2, C_3]$
阶段 4	$s_4 = [D_1, D_2]$
阶段 5	$s_5 = [E]$

解最短路问题



(1) 阶段 (stage): k

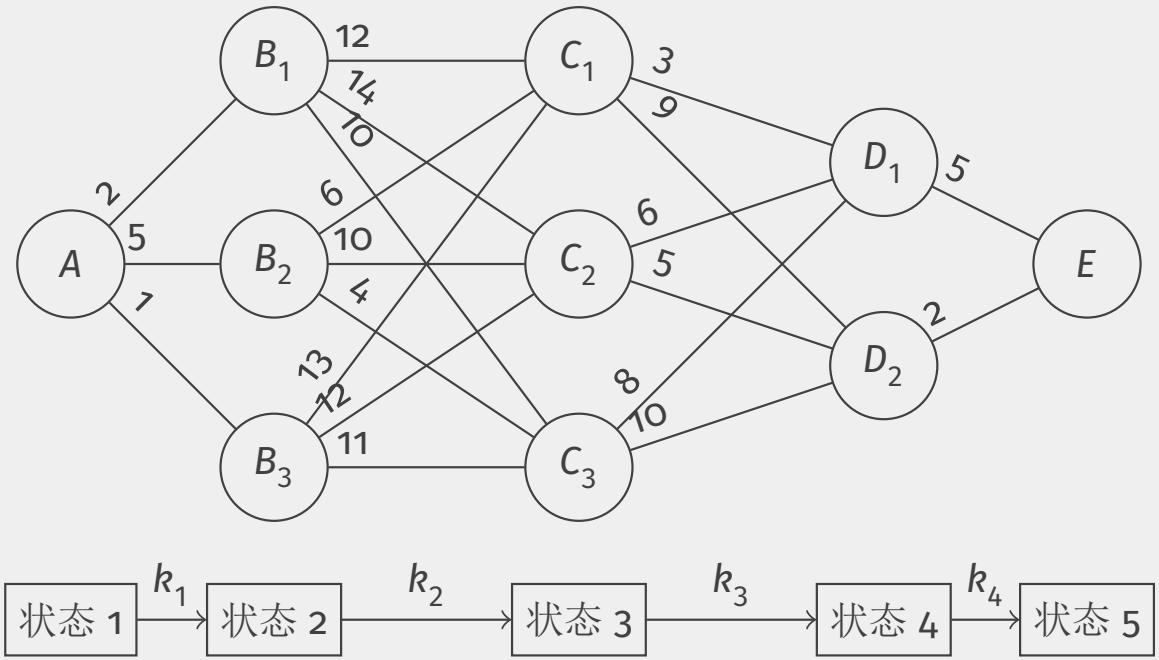
(2) 状态 (state): s_k

阶段 1	$s_1 = [A]$
阶段 2	$s_2 = [B_1, B_2, B_3]$
阶段 3	$s_3 = [C_1, C_2, C_3]$
阶段 4	$s_4 = [D_1, D_2]$
阶段 5	$s_5 = [E]$

(3) 决策 (decision)

- 决策变量 $x_k(s_k)$ 表示第 k 阶段状态为 s_k 时对方案的选择. $D_k(s_k)$ 表示 k 阶段状态为 s_k 时决策允许的取值集合
- $D_2(B_1) = [C_1, C_2, C_3]$

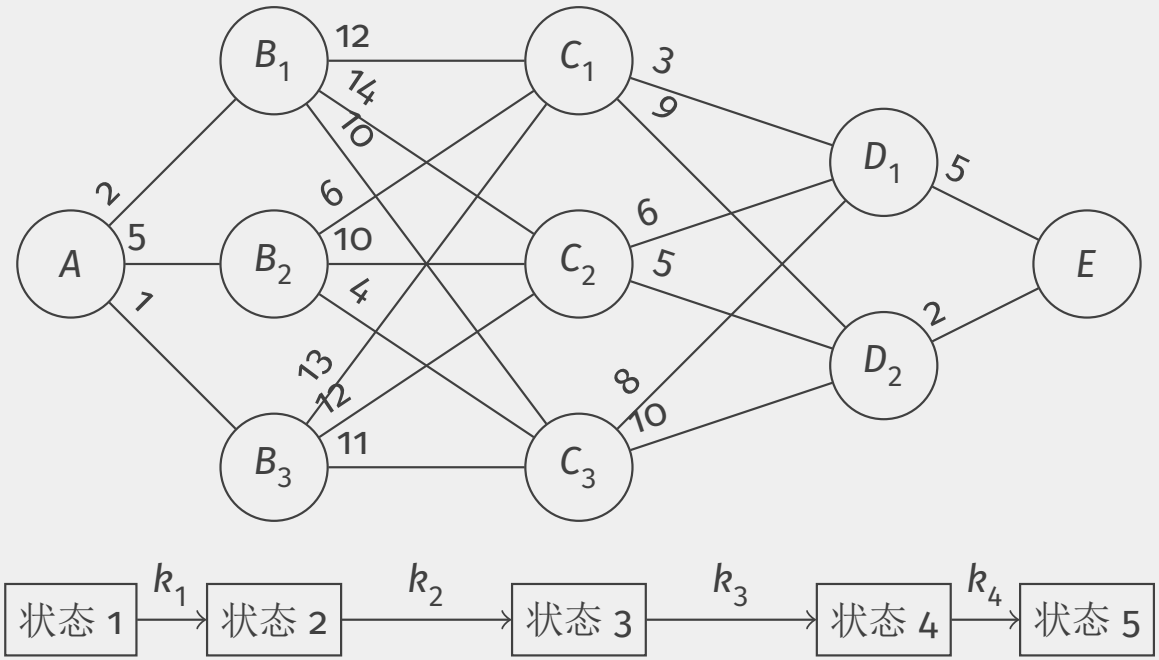
解最短路问题



(4) 策略 (policy) 和子策略 (subpolicy)

- 动态规划问题各阶段决策组成的序列总体称为一个策略
- $[x_1(s_1), x_2(s_2), \ldots, x_n(s_n)]$ 是 n 阶段 DP 一个策略

解最短路问题



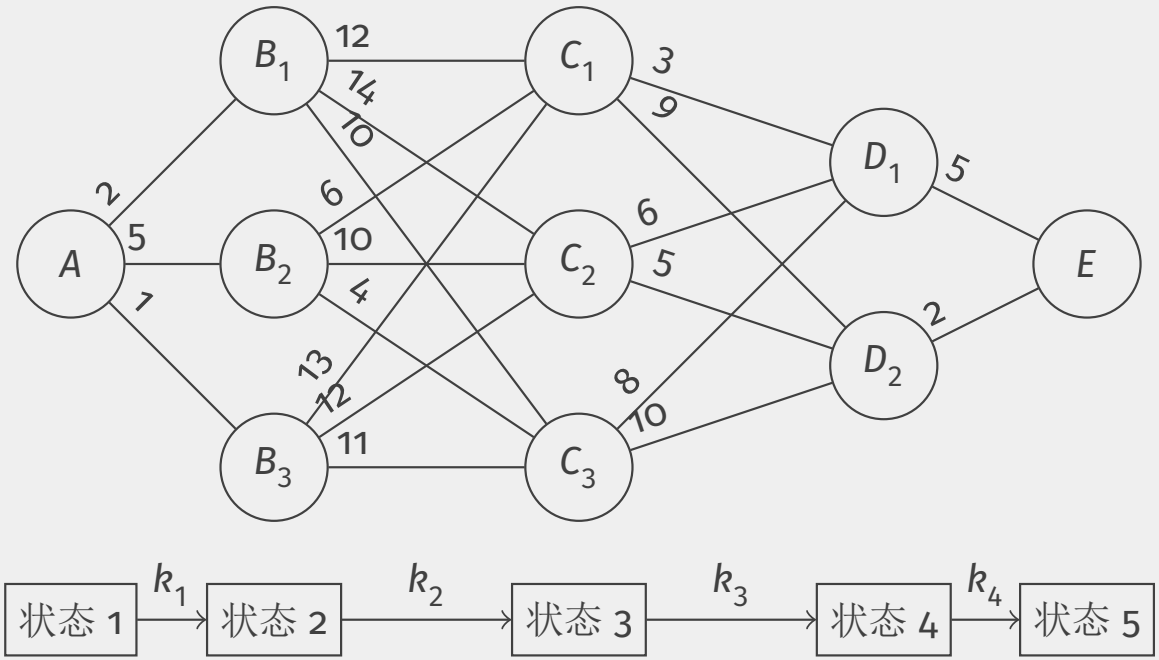
(4) 策略 (policy) 和子策略 (subpolicy)

- 动态规划问题各阶段决策组成的序列总体称为一个策略
- $[x_1(s_1), x_2(s_2), ..., x_n(s_n)]$ 是 n 阶段 DP 一个策略

(5) 状态转移

- 从 s_k 的某一状态值出发, 当决策变量 $x_k(s_k)$ 的取值决定后, 下一阶段状态变量 s_{k+1} 的取值也随之确定. 这种从上一阶段的某一状态到下一阶段某一状态称为状态转移
- $s_{k+1} = T(s_k, x_k(s_k))$
- 或 $s_{k+1} = T(s_k, x_k)$

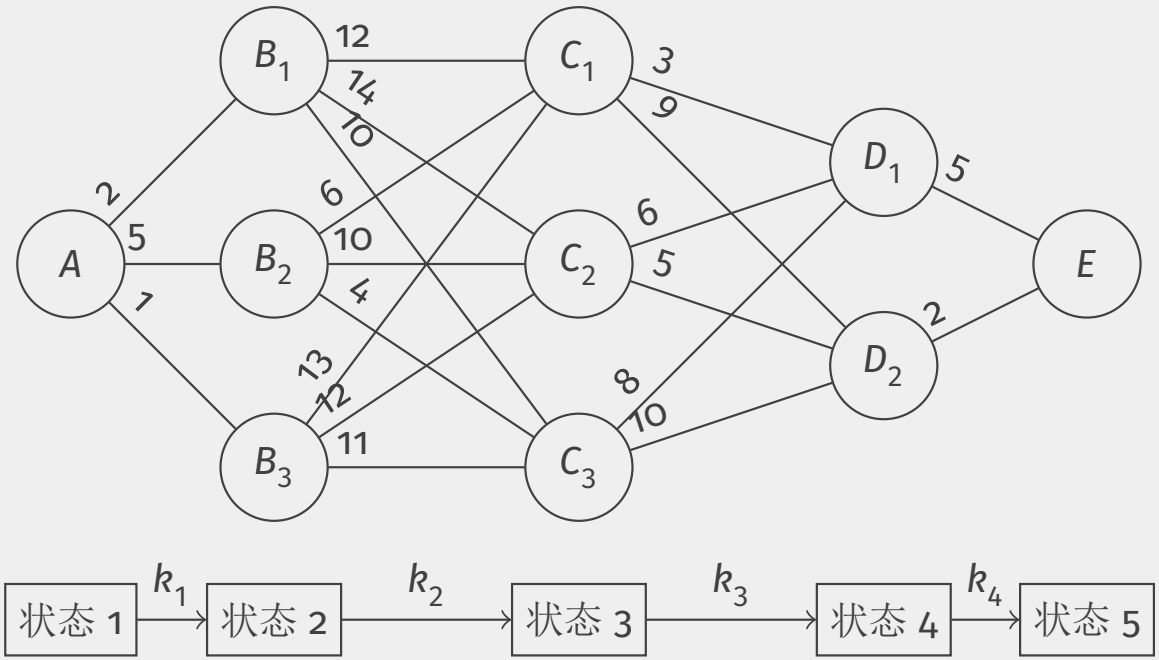
解最短路问题



(6) 目标函数

- 目标函数是用来衡量实现过程优劣的一种数量指标. 它是从状态 s_k 出发至过程最终, 当采取某种策略时, 按预定标准得到的效益值
- $V_{k,n}(s_k, x_k, s_{k+1}, x_{k+1}, \ldots, s_n)$

解最短路问题

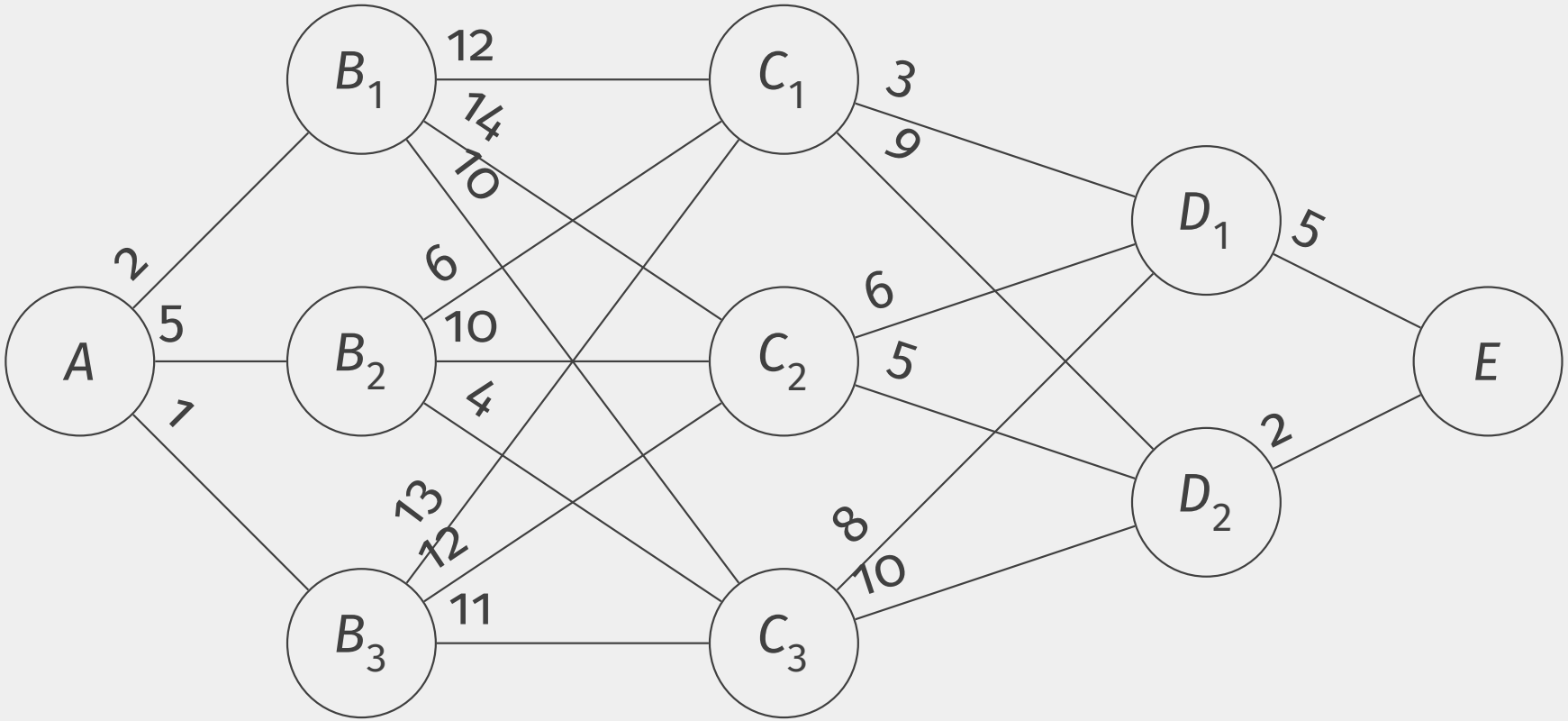


(6) 目标函数

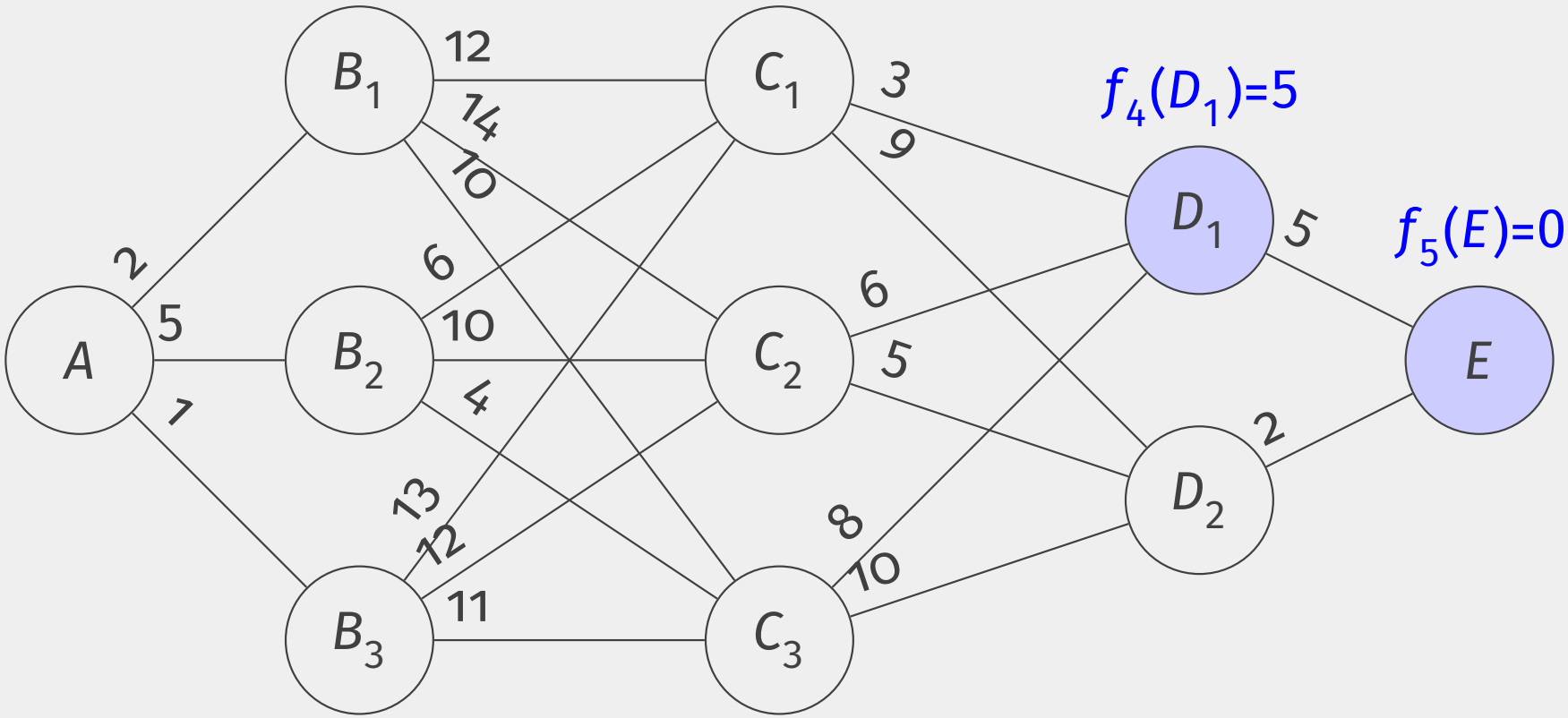
- 目标函数是用来衡量实现过程优劣的一种数量指标. 它是从状态 s_k 出发至过程最终, 当采取某种策略时, 按预定标准得到的效益值
- $V_{k,n}(s_k, x_k, s_{k+1}, x_{k+1}, \ldots, s_n)$

(7) 最优目标函数

- 对某一确定状态选取最优策略后得到的指标函数值, 也就是对应某一最优子策略的某种效益度量
- $f_k(s_k) = optV_{k,n}$

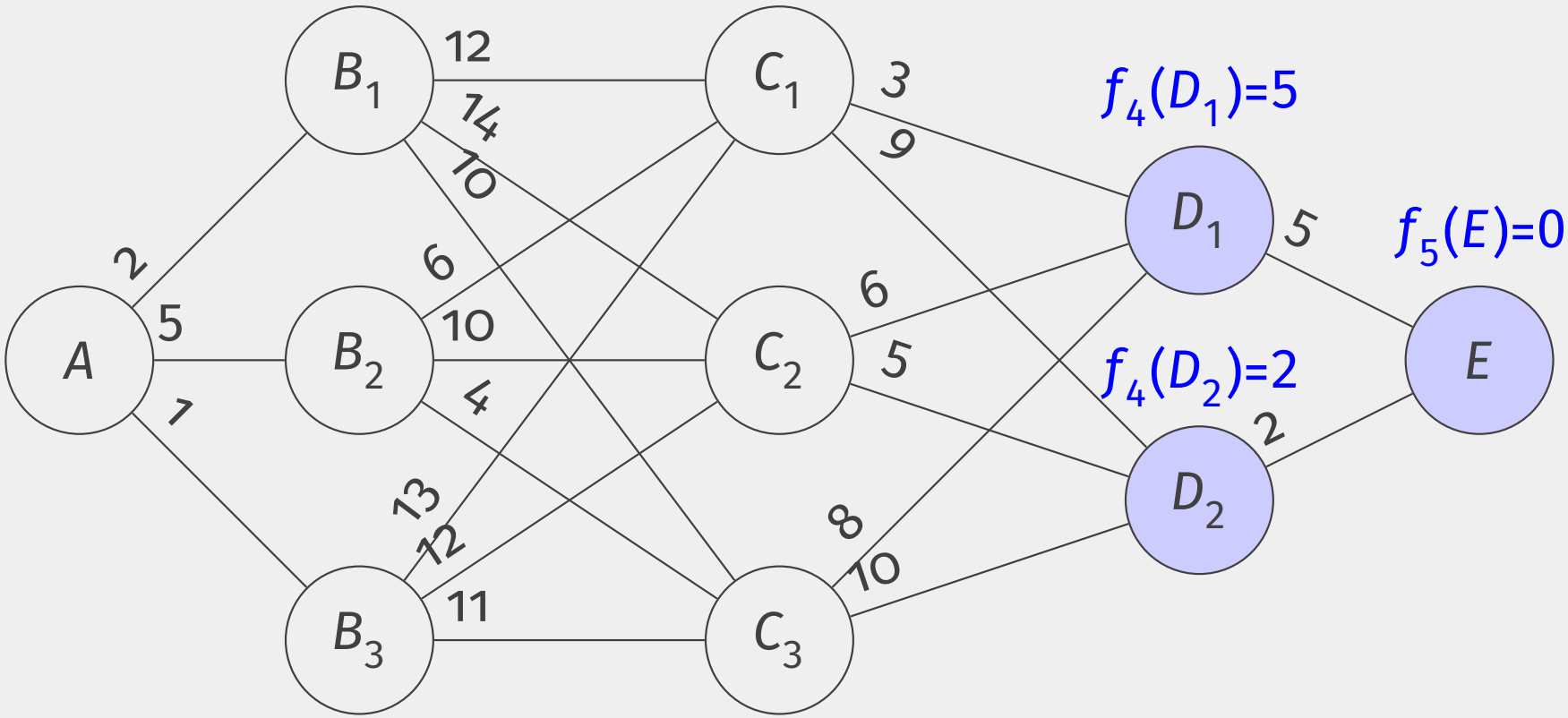


$$f_k(s_k) = \min_{x_k \in D_k(s_k), k=4,3,2,1} [d_k(s_k, x_k) + f_{k+1}(s_{k+1})]$$



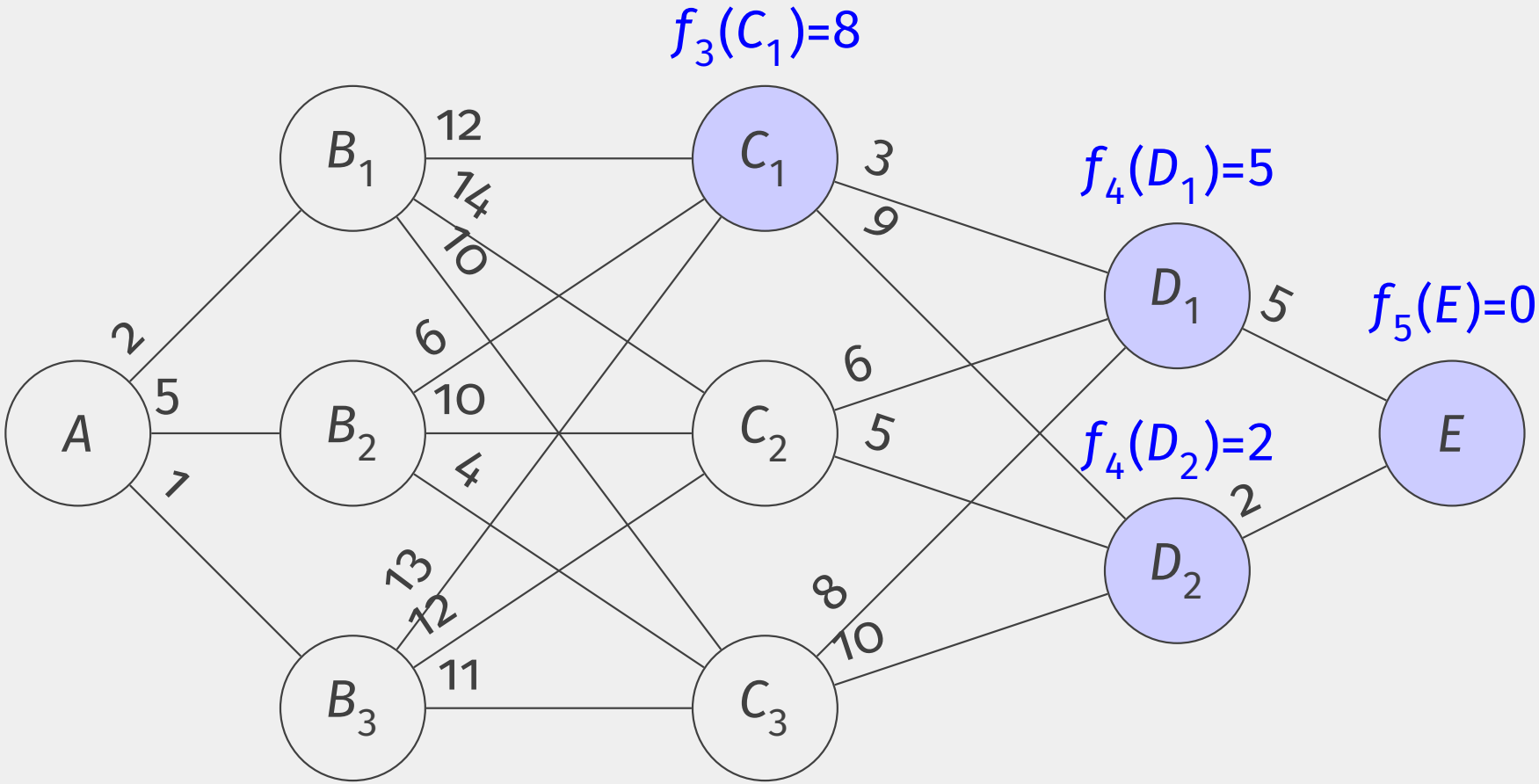
$$f_4(D_1) = d(D_1 \rightarrow E) + f_5(E) = 5 + 0 = 5$$

最优决策：$D_1 \rightarrow E$



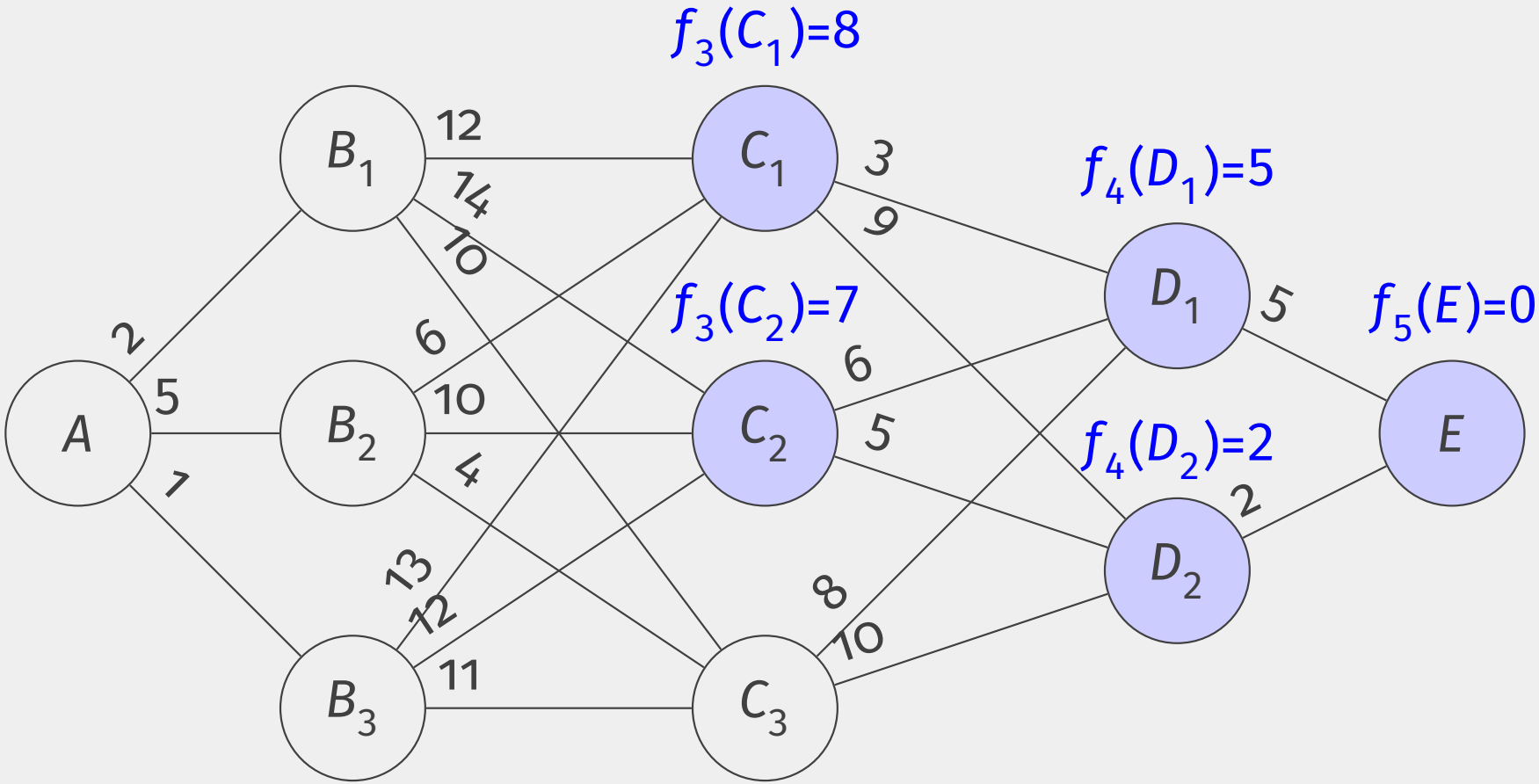
$$f_4(D_2) = d(D_2 \rightarrow E) + f_5(E) = 2 + 0 = 5$$

最优决策：$D_2 \rightarrow E$



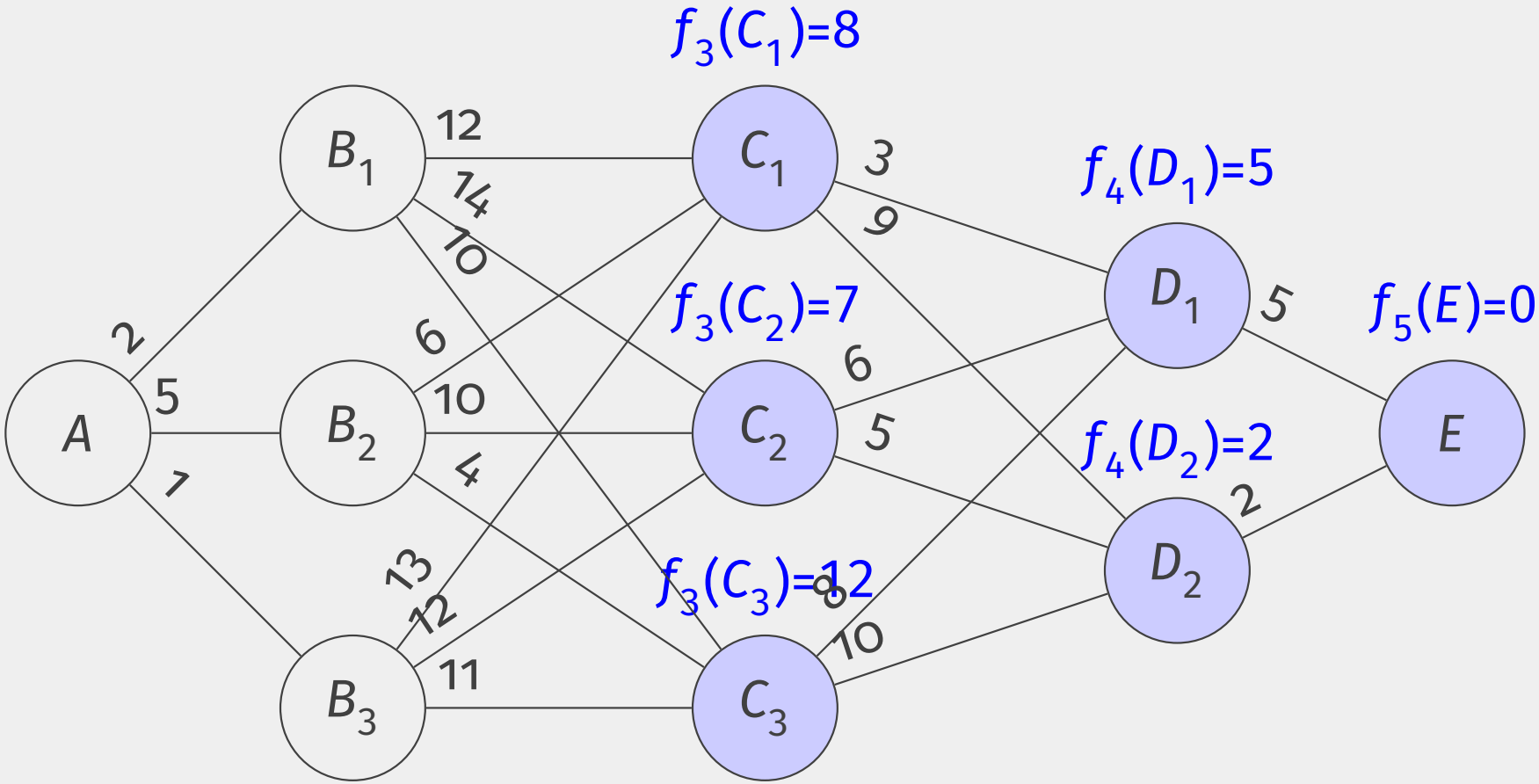
$$f_3(C_1) = min\begin{Bmatrix}(C_1, D_1) + f_4(D_1)\\(C_1, D_2) + f_4(D_2)\end{Bmatrix} = min\begin{Bmatrix}3 + 5\\9 + 2\end{Bmatrix} = 8$$

最优决策：$C_1 \rightarrow D_1$



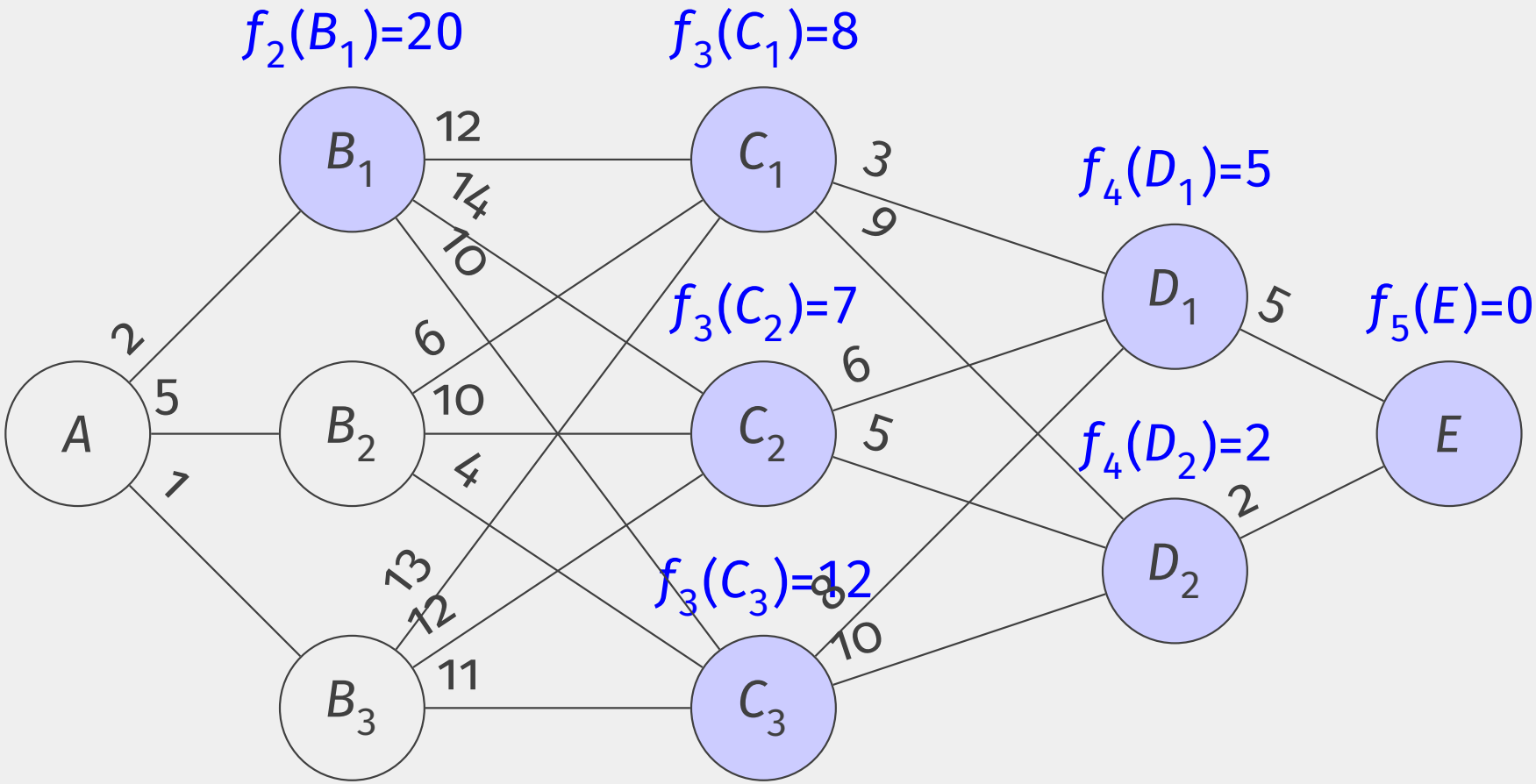
$$f_3(C_2) = min\begin{Bmatrix}(C_2, D_1) + f_4(D_1)\\(C_2, D_2) + f_4(D_2)\end{Bmatrix} = min\begin{Bmatrix}6 + 5\\5 + 2\end{Bmatrix} = 7$$

最优决策：$C_2 \rightarrow D_2$



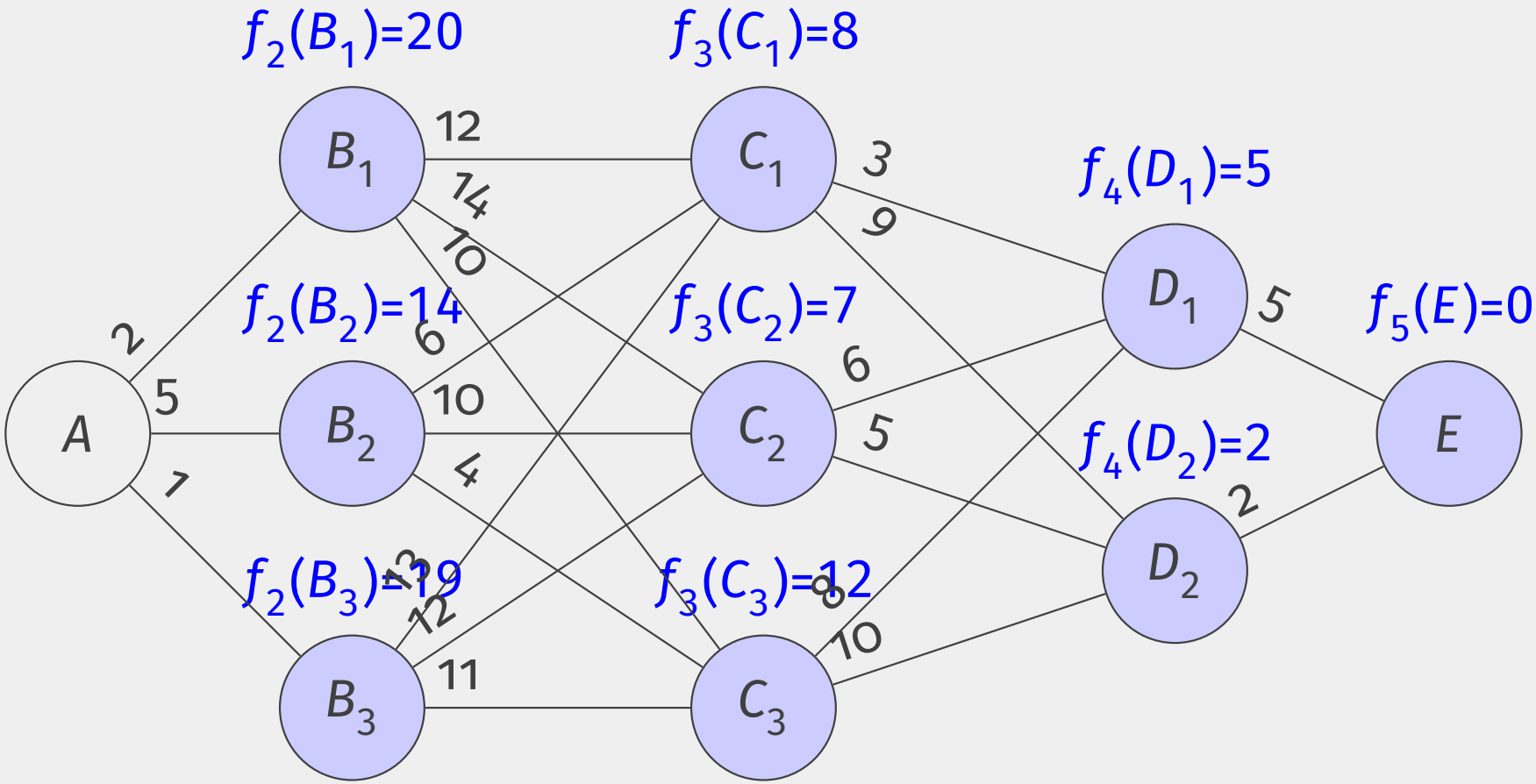
$$f_3(C_3) = min\left\{\begin{matrix}(C_3, D_1) + f_4(D_1)\\ (C_3, D_2) + f_4(D_2)\end{matrix}\right\} = min\left\{\begin{matrix}8 + 5\\ 10 + 2\end{matrix}\right\} = 12$$

最优决策：$C_3 \rightarrow D_2$



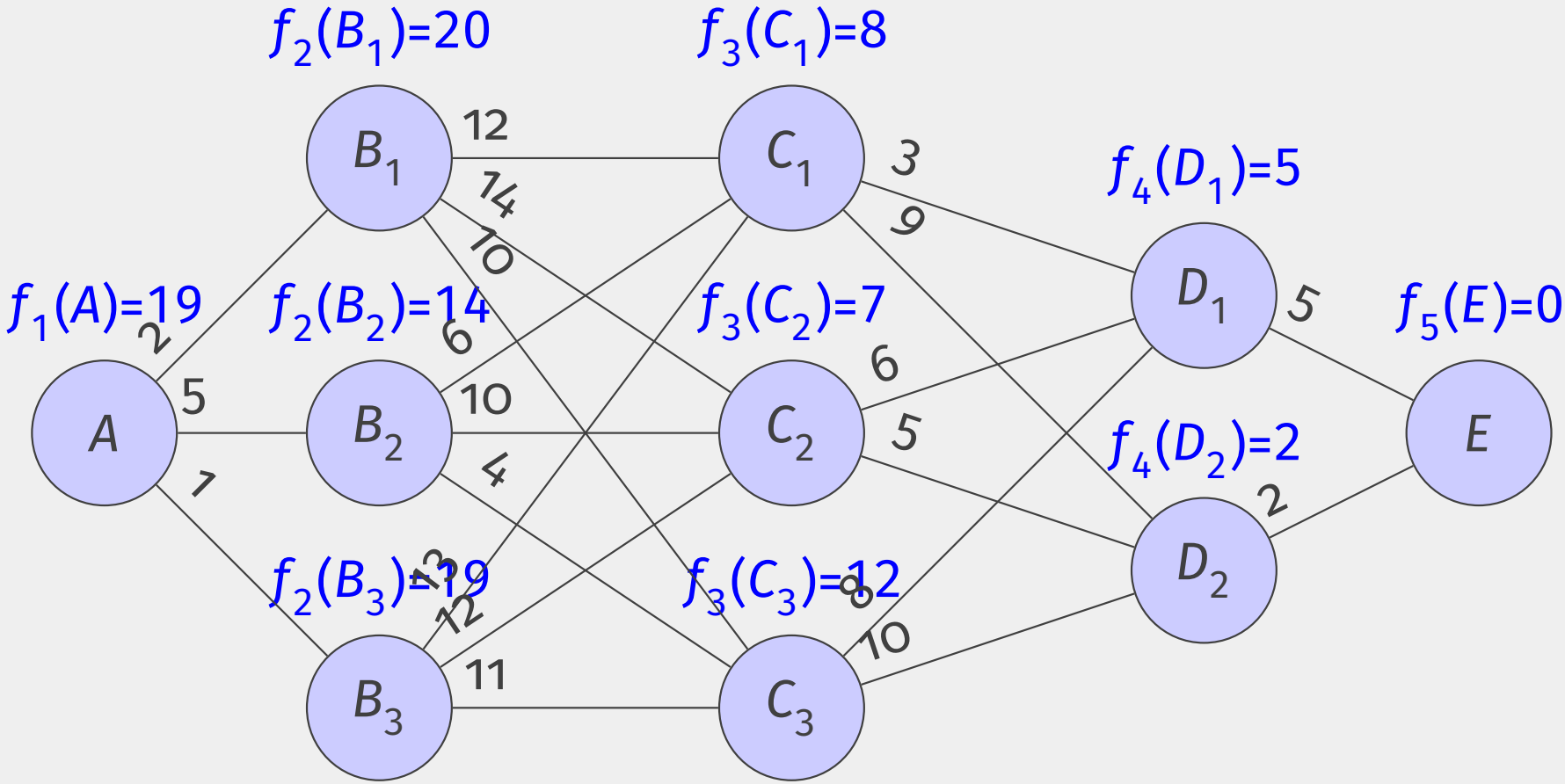
$$f_2(B_1) = min\left\{\begin{array}{l}(B_1, C_1) + f_3(C_1)\\(B_1, C_2) + f_3(C_2)\\(B_1, C_3) + f_3(C_3)\end{array}\right\} = min\left\{\begin{array}{c}12 + 8\\14 + 7\\10 + 12\end{array}\right\} = 20$$

最优决策：$B_1 \rightarrow C_1$



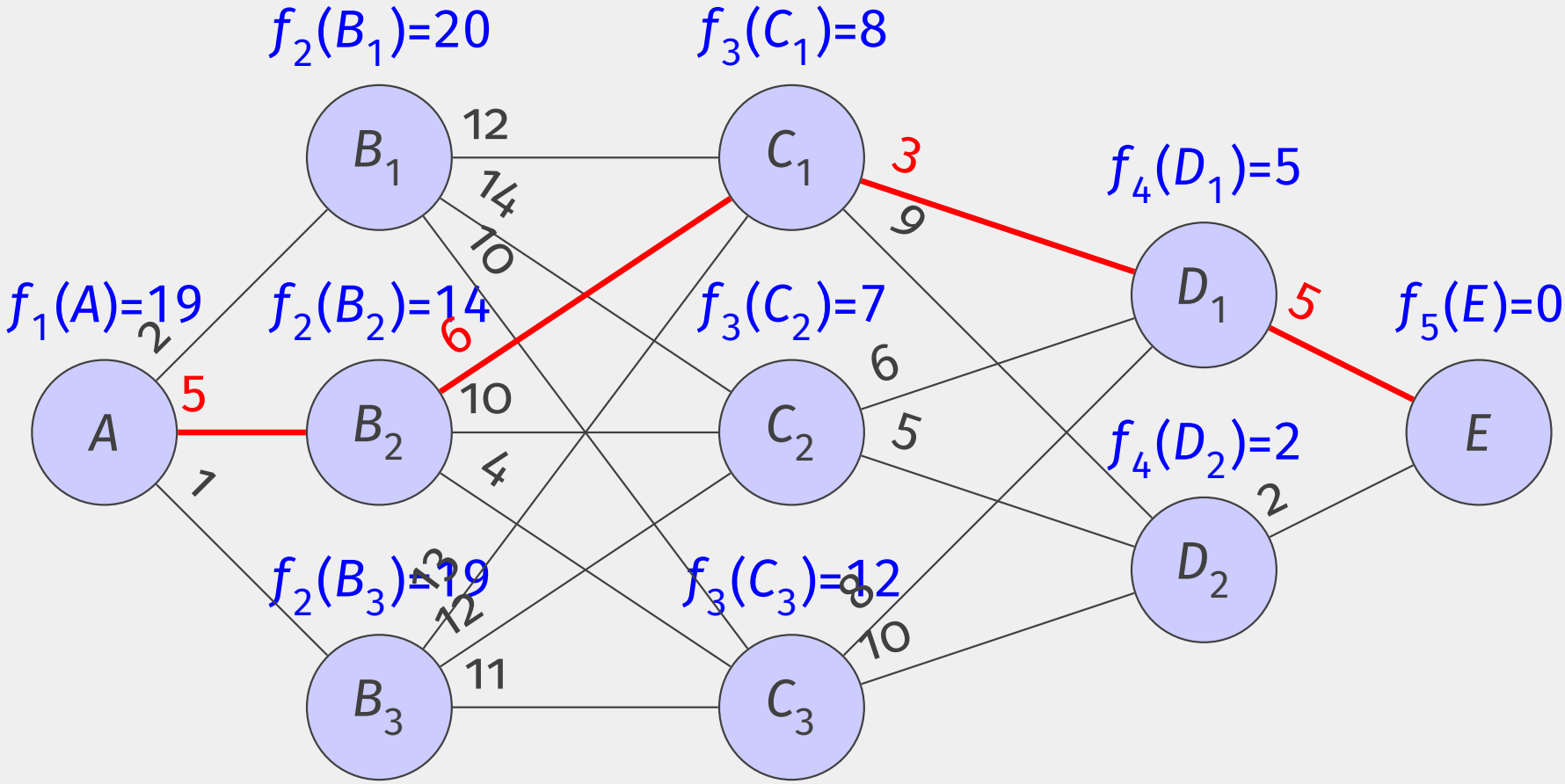
$$f_2(B_3) = min\left\{\begin{array}{l}(B_3, C_1) + f_3(C_1)\\(B_3, C_2) + f_3(C_2)\\(B_3, C_3) + f_3(C_3)\end{array}\right\} = min\left\{\begin{array}{c}13 + 8\\12 + 7\\11 + 12\end{array}\right\} = 19$$

最优决策：$B_3 \rightarrow C_2$



$$f_1(A) = min\begin{Bmatrix}(A,B_1)+f_2(B_1)\\(A,B_2)+f_2(B_2)\\(A,B_3)+f_2(B_3)\end{Bmatrix} = min\begin{Bmatrix}2+21\\5+14\\1+19\end{Bmatrix} = 19$$

最优决策：$A \rightarrow B_2$

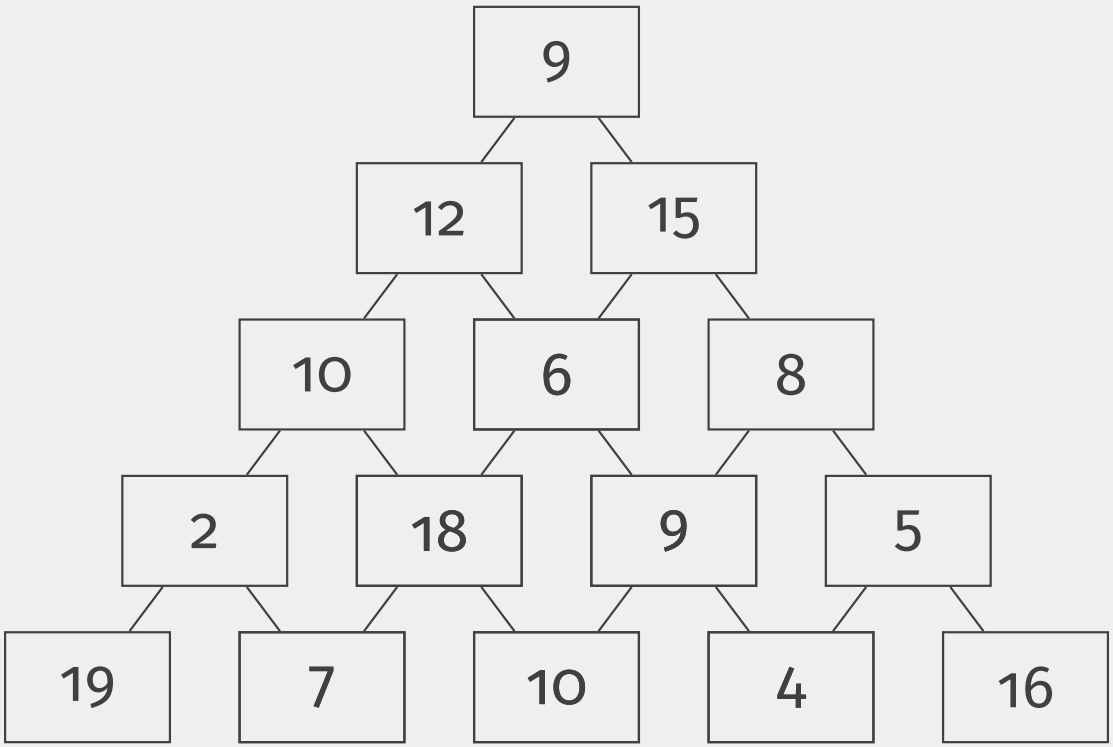


$$A \to B_2 \to C_1 \to D_1 \to E$$

为 A 到 E 的最短路径，长度 19

1163 – The Triangle (poj.org)

- 如图的数字三角形，从顶部出发，每一个结点可以选择向左或向右走，要求找到一个从上到下的路径，路径上数字的和最大



1651 – Multiplication Puzzle (poj.org)

- n 个数排成一排，从中间位置取走一个数，和它前后两个数相乘得到一个乘积值，再取一个数，继续计算，直到剩下的最后三个数相乘，将所有乘积相加，求这个和的最小值

例：10 1 50 20 5

- 依次取 1,20,50:

  $10 \times 1 \times 50 + 50 \times 20 \times 5 + 10 \times 50 \times 5 = 500 + 5000 + 2500 = 8000$

- 依次取 50,20,1:

  $1 \times 50 \times 20 + 1 \times 20 \times 5 + 10 \times 1 \times 5 = 1000 + 100 + 50 = 1150$

分析

- 区间动态规划方法
  (1) 定义二维数组 D[i,j],i,j 为区间左右两个边界
  (2) 子区间的长度逐渐递增，直到原问题的区间长度为止
  (3) 将子区间内的各种情况进行计算，保存最优值
  (4) 由子问题的值推导上一层问题的值，直到计算结束

- 用 a[1...n] 保存 n 个数,D[i,j] 为 i 到 j 区间计算出来的和的最小值，区间长度 s=j-i, 在区间内取一个位置 k，则 D[i,j] 为所有 D[i,k]+D[k,j]+a[i]×a[k]×a[j] 的最小值

4.2 子序列问题

- 最长递增子序列(Longest Increasing Subsequence, LIS)

1. 动态规划解法，复杂度为 $O(n^2)$
   - 用数组 $D[i]$ 储存到 $a[i]$ 为止包含 $a[i]$ 的最长递增子序列的长度
   - $D[i] = max(D[j]) + 1, 1 \le j < i$ 且 $a[j] < a[i]$

i	0	1	2	3	4	5	6
$a[i]$	1	7	3	5	9	4	8
$D[i]$	1	2	2	3	4	3	4

2. 二分查找，复杂度为 $O(nlogn)$

2533 – Longest Ordered Subsequence (poj.org)

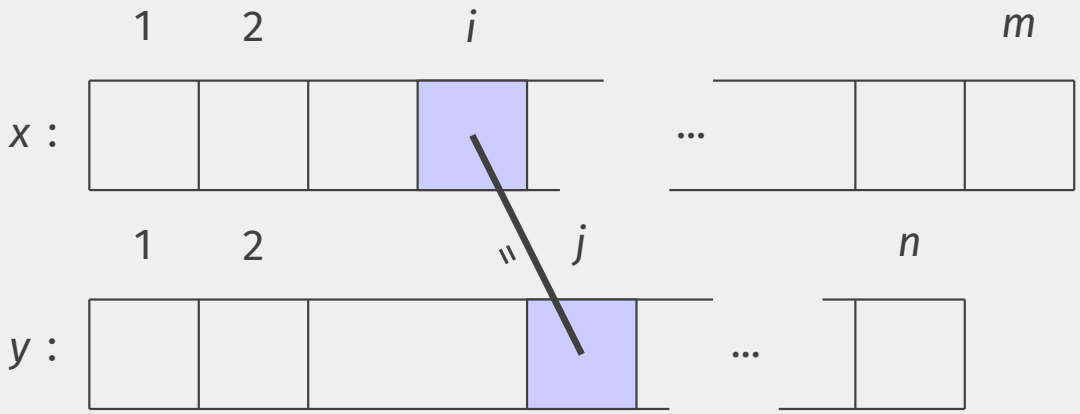
- A numeric sequence of a_i is ordered if $a_1 < a_2 < \ldots < a_N$. Let the subsequence of the given numeric sequence $(a_1, a_2, \ldots, a_N)$ be any sequence $(a_{i_1}, a_{i_2}, \ldots, a_{i_K})$, where $1 \leq i_1 < i_2 < \ldots < i_K \leq N$.
- For example, sequence (1, 7, 3, 5, 9, 4, 8) has ordered subsequences, e. g., (1, 7), (3, 4, 8) and many others. All longest ordered subsequences are of length 4, e. g., (1, 3, 5, 8).
- Your program, when given the numeric sequence, must find the length of its longest ordered subsequence.

最长公共子序列

最长公共子序列 (Longest Common Subsequence, LCS) 是在一个序列集合中 (通常为两个序列) 查找所有序列中最长子序列的问题
两个序列通常为两个字符串，此类问题也称为求最长公共子串问题

- $c[i,j]$ 表示从序列起点到 $x[i]$ 和 $y[j]$ 时最长子序列的长度，则

$$c[i,j] = \begin{cases} c[i-1,j-1]+1, x[i]=y[j] \\ max(c[i-1,j], c[i,j-1]), others \end{cases}$$



1458 – Common Subsequence (poj.org)

- A subsequence of a given sequence is the given sequence with some elements (possible none) left out.
- Given a sequence $X =< x_1, x_2, ..., x_m >$ another sequence $Z =< z_1, z_2, ..., z_k >$ is a subsequence of X if there exists a strictly increasing sequence $< i_1, i_2, ..., i_k >$ of indices of X such that for all $j = 1, 2, ..., k, x_{i_j} = z_j$
- For example, $Z =< a, b, f, c >$ is a subsequence of $X =< a, b, c, f, b, c >$ with index sequence $< 1, 2, 4, 6 >$
- Given two sequences X and Y the problem is to find the length of the maximum-length common subsequence of X and Y

背包问题

一组物品具有重量和价格，在规定的总重量内，如何选择，使得价格最高，被称为背包问题[1](Knapsack problem)

- 0-1 背包问题: 每种物品仅有一件，可以选择放或不放
  - 用动态规划来求解背包问题，通过子问题定义状态：即 $F[i,v]$ 表示前 i 件物品恰放入一个容量为 v 的背包可以获得的最大价值，则当前最优解要么包含第 i 种物品，要么不包含第 i 种物品，状态转移方程:

$$F[i,v] = \max \begin{cases} F[i-1,v] \\ F[i-1,v-c[i]+w[i]] \end{cases}$$

  - 时空复杂度均为 NV，只用一维数组 $F[j]$ 可以降低空间复杂度
  - 恰好装满 $F[0]=0$，$F[1\ldots V]=-\infty$; 不要求装满 $F[1\ldots V]=0$

[1] https://github.com/tianyicui/pack

3624 – Charm Bracelet (poj.org)

- Bessie has gone to the mall's jewelry store and spies a charm bracelet. Of course, she'd like to fill it with the best charms possible from the $N(1 \leq N \leq 3,402)$ available charms.
- Each charm i in the supplied list has a weight $W_i(1 \leq W_i \leq 400)$, a 'desirability' factor $D_i(1 \leq D_i \leq 100)$, and can be used at most once. Bessie can only support a charm bracelet whose weight is no more than $M(1 \leq M \leq 12,880)$.
- Given that weight limit as a constraint and a list of the charms with their weights and desirability rating, deduce the maximum possible sum of ratings.

分析

- n 个物品，每个物品有 w 和 d 属性，要求选出一定的物品，在 $\sum w$ 不超过 m 的情况下，使得 $\sum d$ 最大

输入：

4	6
1	4
2	6
3	12
2	7

n	m
w_1	d_1
w_2	d_2
w_3	d_3
w_4	d_4

完全背包问题

- 完全背包问题: 有 n 种物品和一个容量为 V 的背包，每种物品都有无限件可用

- 在 0-1 背包基础上，将添加第 i 件物品的情况考虑到已经选择过第 i 件物品的情况
  - 0-1 背包：$v = V \to 0$, 保证每件物品选一次
  - 完全背包：$v = 0 \to V$, 每种物品可选无限件

- 恰好装满 $F[0] = 0$，$F[1 \dots V] = -\infty$; 不要求装满 $F[1 \dots V] = 0$

1384 – Piggy-Bank (poj.org)

- n 种不同的硬币，有两个属性，p 代表金额，w 代表重量，钱罐空时重 e，满时重 f，计算满时钱罐里最小可能的总金额是多少

输入：

10	110
2	
1	1
30	50

e	f
n	
p_1	w_1
p_2	w_2

多重背包问题

- 多重背包问题: 一种物品有 m 个 (既不是固定的 1 个，也不是无数个)

- 多重背包 c 为费用，w 为价值，m 为数量
  - 当 $c \times m$ 超过 V，直接采用完全背包
  - 否则将 m 进行拆分，拆分成

$$1, 2, 4, \ldots, 2^k - 1, m - (2^k - 1)$$

  - 然后进行 0-1 背包

- 例如 $m = 13$，拆成 $1, 2, 4, 6$

1276 – Cash Machine (poj.org)

- 有 N 台提款机，每台提款机提供不同数量 n_i 和面值 D_i 的货币，求这些机器能提取的不超过指定限额 $cash$ 的最大金额

Example

$cash$	N	n_1	D_1	n_2	D_2	n_3	D_3	n_N	D_N
735	3	4	125	6	5	3	350		
633	4	500	30	6	100	1	5	0	1

输出	n_1	D_1	n_2	D_2	n_3	D_3	n_N	D_N
735	3	125	2	5	1	350		
633	1	30	6	100	0	5	0	1