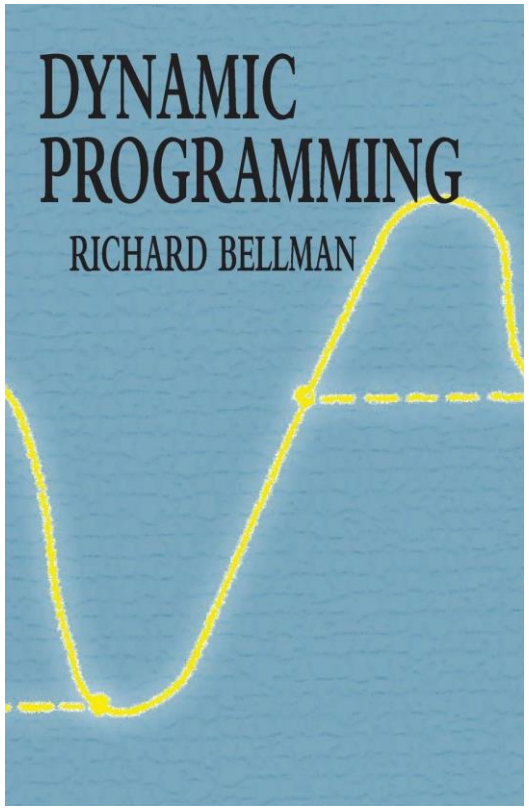
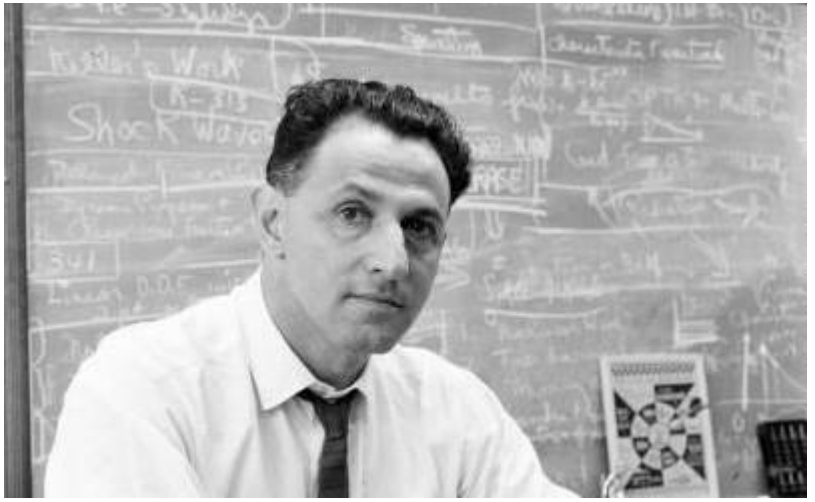
数据分析与算法设计
第8章 动态规划
（Dynamic Programming）

李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn



动态规划：
多阶段决策过程最优的通用方法



发明人：
美国数学家Richard Bellman

RICHARD BELLMAN ON THE BIRTH OF DYNAMIC PROGRAMMING

STUART DREYFUS

University of California, Berkeley, IEOR, Berkeley, California 94720, dreyfus@ieor.berkeley.edu

CHOICE OF THE NAME DYNAMIC PROGRAMMING

"I spent the Fall quarter (of 1950) at RAND. My first task was to find a name for multistage decision processes.

"An interesting question is, 'Where did the name, dynamic programming, come from?' The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, 'programming.' I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying—I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities" (p. 159).

动态规划

- 基本思想
  - 如果求解的问题由交叠的子问题构成，且这些子问题出现在求解的递推关系中，那么与其对交叠的子问题一次又一次地求解，还不如对每个较小的子问题只求解一次并把结果记录在表中，并从表中得出原问题的解
    - 可看成一种特殊的时空权衡技术
    - 对比分治法：分治法将原问题分解为不重叠的子问题
- 简单例子：斐波那契数列的计算

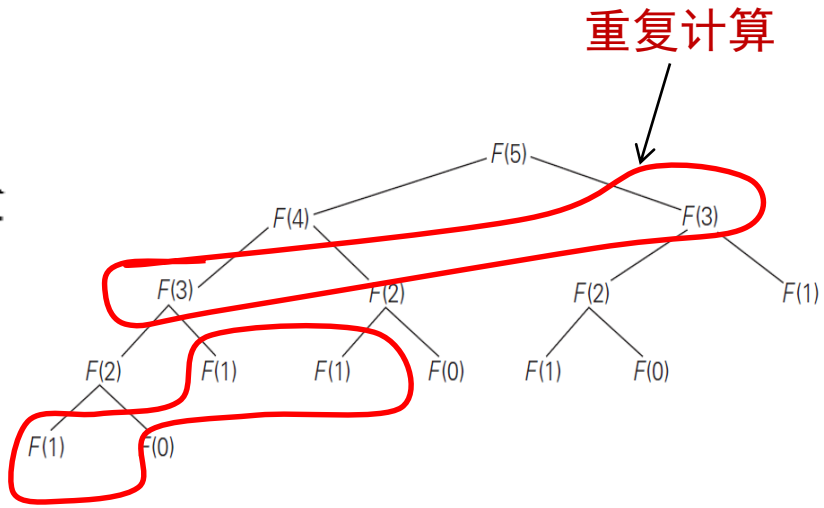
简单例子：斐波那契数列

- 数列的递推式和初始条件

$$F(n) = F(n-1) + F(n-2) \quad \text{for } n > 1$$
$$F(0) = 0, \qquad F(1) = 1.$$

- 自顶向下递归求解

算法 $F(n)$
//根据定义，递归计算第 n 个斐波那契数
//输入：一个非负整数 n
//输出：第 n 个斐波那契数
if $n \leqslant 1$ **return** n
else return $F(n-1) + F(n-2)$



简单例子：斐波那契数列

- 自底向上动态规划

$F(0) = 0$
$F(1) = 1$
$F(2) = 0+1 = 1$
$F(3) = 1+1 = 2$
$F(4) = 1+2 = 3$
$F(5) = 2+3 = 5$
.
.
.
$F(n\text{-}2) =$
$F(n\text{-}1) =$
$F(n) = F(n\text{-}1) + F(n\text{-}2)$

用表格记录子问题的解

算法 Fib(n)
//根据定义，迭代计算第 n 个斐波那契数
//输入：一个非负整数 n
//输出：第 n 个斐波那契数
$F[0] \leftarrow 0$; $F[1] \leftarrow 1$
for $i \leftarrow 2$ **to** n **do**
  $F[i] \leftarrow F[i-1] + F[i-2]$
return $F(n)$

Those who cannot remember the past are condemned to repeat it.

-Dynamic Programming

目录

- 三个基本例子
  - 币值最大化问题
  - 找零问题
  - 硬币收集问题
- 背包问题
- Warshall算法和Floyd算法
- 最优二叉查找树

币值最大化问题

- 问题描述

币值最大化问题 给定一排 n 个硬币，其面值均为正整数 $c_1, c_2, \cdots, c_n$，这些整数并不一定两两不同。请问如何选择硬币，使得在其原始位置互不相邻的条件下，所选硬币的总金额最大。

- 递推方程

包括最后一枚硬币的组合　　不包括最后一枚硬币的组合

$$F(n) = \max\{c_n + F(n-2), F(n-1)\}, \quad n > 1$$
$$F(0) = 0, F(1) = c_1$$

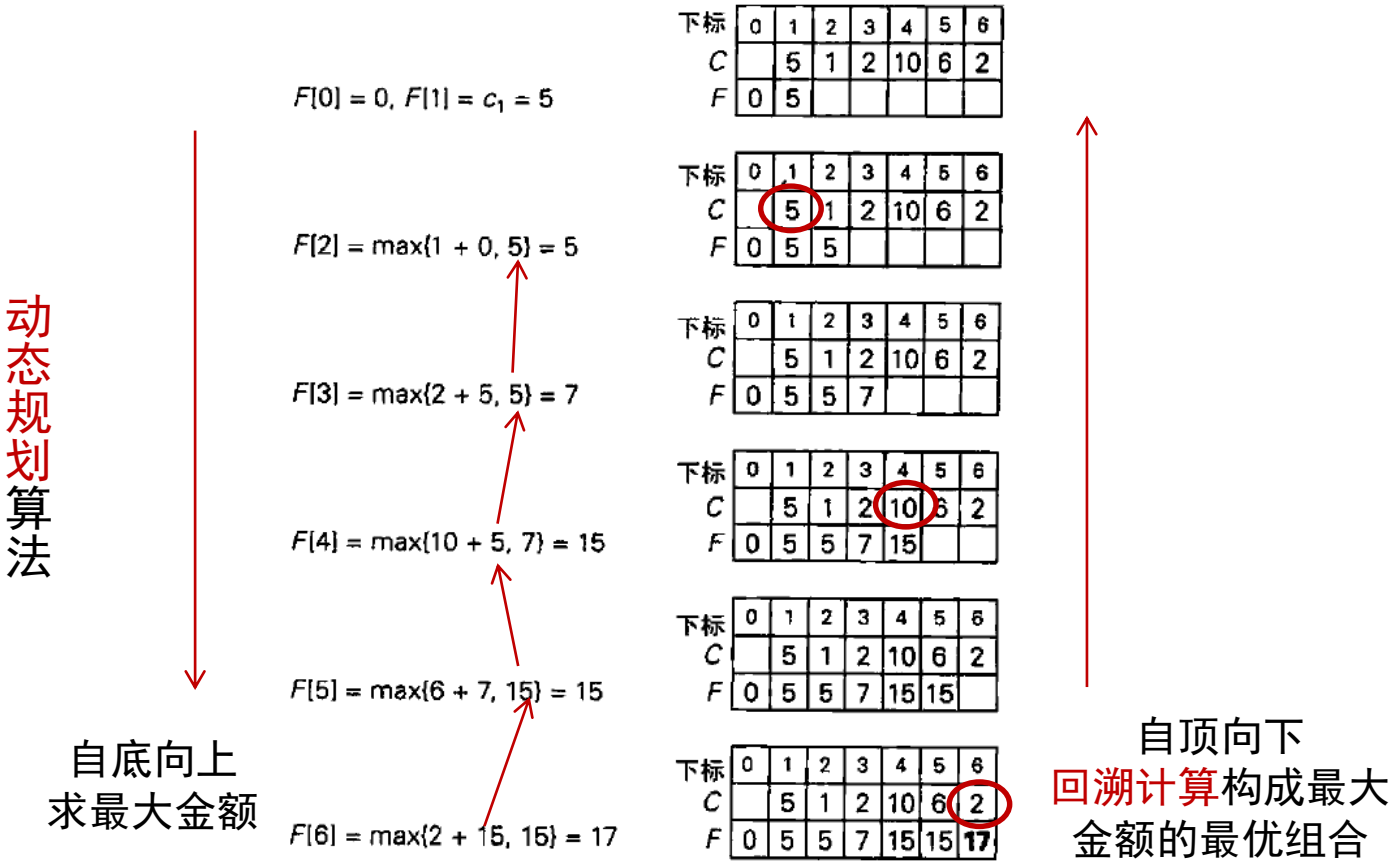
可选硬币的最大金额

算法 CoinRow($C[1..n]$)
//应用公式(8.3)，自底向上求最大金额
//输入：数组 $C[1..n]$保存 n 个硬币的面值
//输出：可选硬币的最大金额
$F[0] \leftarrow 0$; $F[1] \leftarrow C[1]$
for $i \leftarrow 2$ **to** n **do**
　$F[i] \leftarrow \max(C[i] + F[i-2], F[i-1])$
return $F[n]$

$\Theta(n)$ 时间和空间效率

币值最大化问题

- 举例：假设有6枚硬币，面值分别为5，1，2，10，6，2



找零问题

- 问题描述
  - 需找零金额为n，最少要用多少面值为$d_1 < d_2 < \cdots d_m$的硬币？

- 递推式

$$\begin{cases} \text{当} n>0, \quad F(n)=\min\limits_{j:n\geq d_j}\{F(n-d_j)\}+1 \\ F(0)=0 \end{cases}$$

在总金额为$(n-d_j)$的硬币基础上再加1枚硬币

举例：n=6, 可用的硬币面值为1，3，4

$F[0] = 0$

n	0	1	2	3	4	5	6
F	0						

$F[1] = \min\{F[1-1]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1					

$F[2] = \min\{F[2-1]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2				

$F[3] = \min\{F[3-1], F[3-3]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1			

$F[4] = \min\{F[4-1], F[4-3], F[4-4]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1	1		

$F[5] = \min\{F[5-1], F[5-3], F[5-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	

$F[6] = \min\{F[6-1], F[6-3], F[6-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	2

最优解为两枚面值为3的硬币

找零问题

算法 ChangeMaking($D[1..m]$, n)

//应用动态规划算法求解找零问题，找出使硬币加起来等于 n 时所需最少的硬币数目

//其中币值为 $d_1 < d_2 < \cdots < d_m$，$d_1 = 1$

//输入：正整数 n，以及用于表示币值的递增整数数组 $D[1...m]$，$D[1] = 1$

//输出：总金额等于 n 的硬币最少的数目

$F[0] \leftarrow 0$

for $i \leftarrow 1$ **to** n **do**

  $temp \leftarrow \infty$; $j \leftarrow 1$

  while $j \leqslant m$ **and** $i \geqslant D[j]$ **do**

    $temp \leftarrow \min(F[i - D[j]], temp)$

    $j \leftarrow j + 1$

  $F[i] \leftarrow temp + 1$

return $F[n]$

$O(nm)$ 时间效率

$\Theta(n)$ 空间效率

硬币收集问题

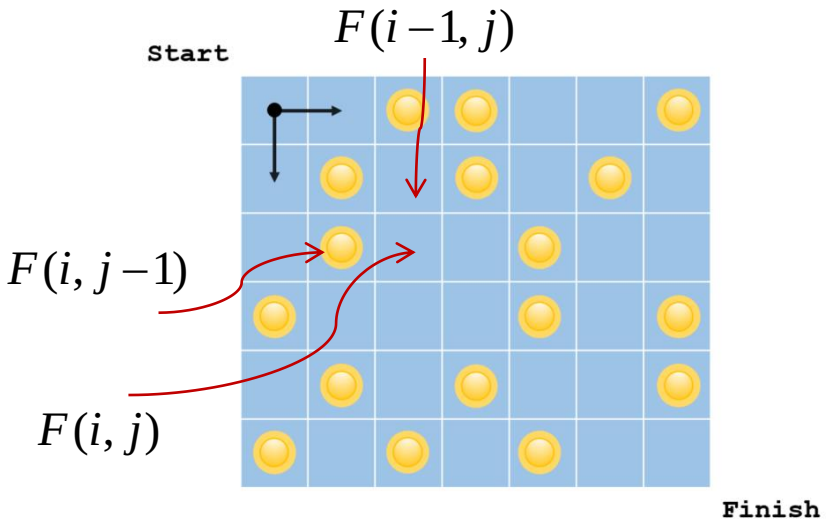
- 问题描述

硬币收集问题 在 $n \times m$ 格木板中放有一些硬币，每格的硬币数目最多为一个。在木板左上方的一个机器人需要收集尽可能多的硬币并把它们带到右下方的单元格。每一步，机器人可以从当前的位置向右移动一格或向下移动一格。当机器人遇到一个有硬币的单元格时，就会将这枚硬币收集起来。设计一个算法找出机器人能找到的最大硬币数并给出相应的路径。

- 递推式

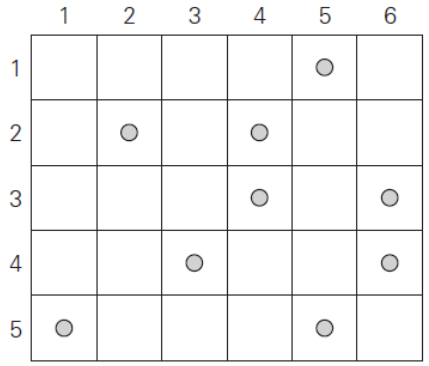
$$\begin{cases} F(i,j) = \max\{F(i-1,j), F(i,j-1)\} + c_{ij}, & 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m \\ F(0,j) = 0, \quad 1 \leqslant j \leqslant m; \quad F(i,0) = 0, \quad 1 \leqslant i \leqslant n \end{cases}$$

若单元格(i,j)中有硬币存在，则 c_{ij} 为 1，否则为 0。



硬币收集问题

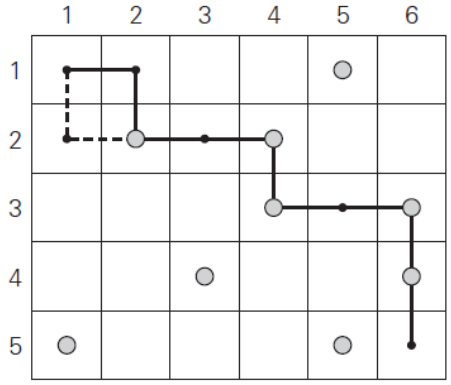
- 举例



(a)木板上初始的硬币布局

	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	5

(b)动态规划算法的计算结果



(c)收集到 5 个硬币(最大值)的两条路径

硬币收集问题

算法 RobotCoinCollection($C[1..n, 1..m]$)

//利用动态规划算法计算机器人在 $n \times m$ 木板上所能收集的最大硬币数

//机器人从(1, 1)出发，每次向右或向下移动，从左上方移动到右下方

/输入：矩阵 $C[1..n, 1..m]$，矩阵元素为 1 或者 0 分别表示单元格中有一枚硬币或者没有

//输出：机器人在单元格(n,m)中收集到的最大硬币数

$F[1, 1] \leftarrow C[1, 1]$; **for** $j \leftarrow 2$ **to** m **do** $F[1, j] \leftarrow F[1, j-1] + C[1, j]$

for $i \leftarrow 2$ **to** n **do**

    $F[i, 1] \leftarrow F[i-1, 1] + C[i, 1]$

    for $j \leftarrow 2$ **to** m **do**

        $F[i, j] \leftarrow \max(F[i-1, j], F[i, j-1]) + C[i, j]$

return $F[n, m]$

$\Theta(nm)$ 时间和空间效率

动态规划的要素

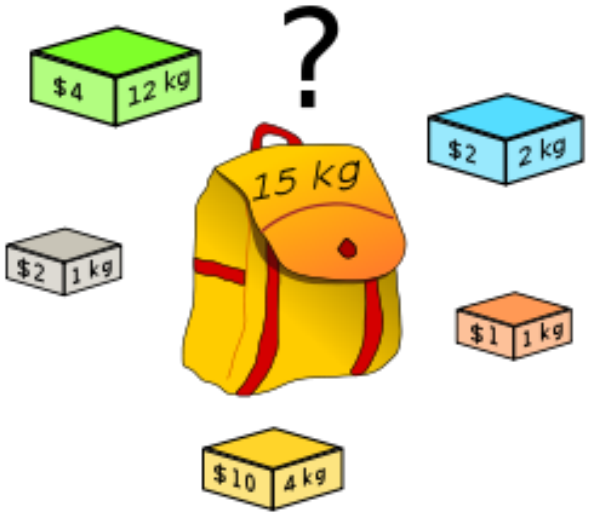
- 最优化原理（最优子结构性质）
  - 一个最优化策略的子策略总是最优的。因此，可以采用自底向上的方式从子问题的最优解逐步构造出整个问题的最优解
- 无后向性
  - 将动态规划中各阶段按照一定的次序排列好后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响未来的决策，而只能通过当前的状态来影响。换句话说，每个状态都是过去历史的一个完整总结，即无后向性
- 子问题的重叠性
  - 该条件为非必要条件，但缺少此条件，动态规划法相比其它方法没有太多的优势

目录

- 三个基本例子
  - 币值最大化问题
  - 找零问题
  - 硬币收集问题
- 背包问题
- Warshall算法和Floyd算法
- 最优二叉查找树

背包问题

- 回顾：问题描述（Knapsack Problem）
  - 给定n个重量为W_1、W_2、…W_n，价值为V_1、V_2、…V_n的物品和一个承重为W的背包，要求选择一些物品装入背包，使得背包内物品的价值之和达到最大

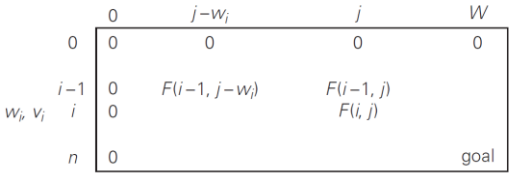


背包问题

- 动态规划求解
  - 考虑一个由前$i(1 \le i \le n)$个物品定义的问题实例，假定背包的承重量为$j(1 \le j \le W)$，$F(i,j)$为该实例的最优解的物品总价值。则可把前i个物品能够放进重量为j的背包中的子集分成两个类别：
    - 不包含第i个物品，则该子集的总价值为$F(i-1,j)$
    - 包含第i个物品，则该子集的总价值为$F(i-1,j-W_i)+V_i$
  - 递推式

$$F(i,\ j) = \begin{cases} \max\{F(i-1,\ j),\ v_i + F(i-1,\ j-w_i)\} & \text{if } j - w_i \ge 0, \\ F(i-1,\ j) & \text{if } j - w_i < 0. \end{cases}$$

初始条件：$F(0,\ j) = 0 \text{ for } j \ge 0 \quad \text{and} \quad F(i,\ 0) = 0 \text{ for } i \ge 0.$



背包问题

- 举例

物　品	重　量	价值/美元
1	2	12
2	1	10
3	3	20
4	2	15

承重量 $W = 5$

$w_1 = 2,\ v_1 = 12$

$w_2 = 1,\ v_2 = 10$

$w_3 = 3,\ v_3 = 20$

$w_4 = 2,\ v_4 = 15$

i \ 承重量 j	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	**37**

$F(2,3) > F(1,3)$

⟹ 包含物品2

⟹ 包含物品1

$F(4,5) > F(3,5)$

⟹ 包含物品4

$F(2,3) = F(3,3)$

⟹ 不包含物品3

背包问题

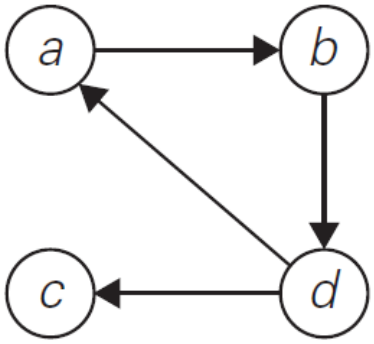
```
算法  MFKnapsack (i, j)
//对背包问题实现记忆功能方法
//输入：一个非负整数 i 表示先考虑的物品数量，一个非负整数 j 表示背包的承重量
//输出：前 i 个物品的最优可行子集的价值
//注意：我们把输入数组 Weights[1..n]，Values[1..n]和表格 F[0..n, 0..W]作为全局变量
//除了行 0 和列 0 用 0 初始化以外，F 的所有单元格都用-1 初始化
if  F[i, j] < 0
    if  j < Weights[i]
        value ← MFKnapsack(i − 1, j)
    else
        value ← max(MFKnapsack(i − 1, j),
                Values[i] + MFKnapsack(i − 1, j − Weights[i]))
    F[i, j] ← value
return  F[i, j]
```

目录

- 三个基本例子
  - 币值最大化问题
  - 找零问题
  - 硬币收集问题
- 背包问题
- Warshall算法和Floyd算法
- 最优二叉查找树

求解有向图的传递闭包

- 传递闭包：用1个n阶的布尔矩阵$T = \{t_{i,j}\}$表示，当存在一条有效的有向路径时，$t_{i,j}$=1；否则，$t_{i,j}$=0



有向图

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

邻接矩阵

$$T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

传递闭包

- 广泛应用
  - 物流管理、公交查询、软件测试及计算机网络等方面

Warshall算法

- 基本思想：通过一系列n阶的布尔矩阵构造传递闭包

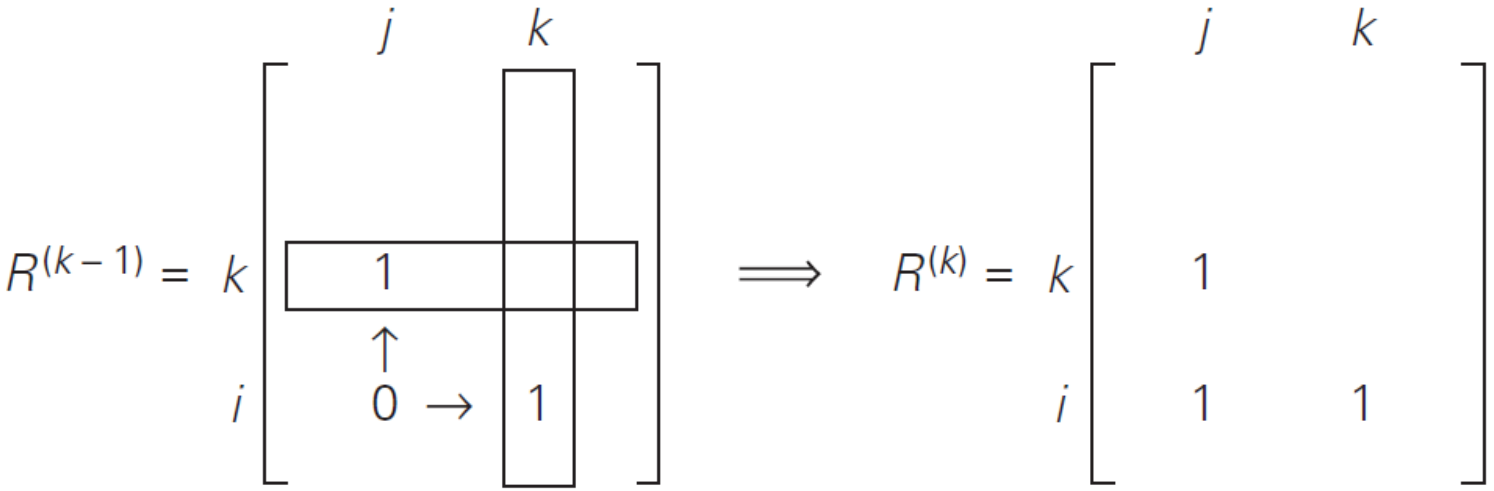
$$\boldsymbol{R}^{(0)}, \ldots, \boldsymbol{R}^{(k-1)}, \boldsymbol{R}^{(k)}, \ldots, \boldsymbol{R}^{(n)}$$

  - 矩阵$\boldsymbol{R}^{(0)}$为邻接矩阵
  - 矩阵$\boldsymbol{R}^{(k)}$：$[\boldsymbol{R}^{(k)}]_{i,j} = 1$，当且仅当第i个顶点到第j个顶点存在一条有效的有向路径，并且路径中间的每一个节点编号不大于k

- 如何迭代地从矩阵$\boldsymbol{R}^{(k-1)}$来构造$\boldsymbol{R}^{(k)}$呢？

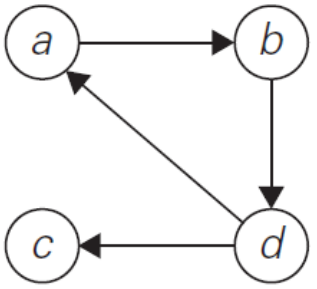
Warshall算法

- 从矩阵$\boldsymbol{R}^{(k-1)}$构造$\boldsymbol{R}^{(k)}$的规则：
  - 所有$[\boldsymbol{R}^{(k-1)}]_{i,j}=1$的位置仍为1
  - 对于$[\boldsymbol{R}^{(k-1)}]_{i,j}=0$的位置，当且仅当$[\boldsymbol{R}^{(k-1)}]_{i,k}=1$ & $[\boldsymbol{R}^{(k-1)}]_{k,j}=1$时，该位置更新为 $[\boldsymbol{R}^{(k)}]_{i,j}=1$



Warshall算法

- 举例



有向图

$$R^{(0)}=\begin{array}{c|cccc} & a & b & c & d \\ a & 0 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 0 & 1 & 0 \end{array} \Rightarrow R^{(1)}=\begin{array}{c|cccc} & a & b & c & d \\ a & 0 & 1 & 0 & 0 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & \mathbf{1} & 1 & 0 \end{array} \Rightarrow$$

$$R^{(2)}=\begin{array}{c|cccc} & a & b & c & d \\ a & 0 & 1 & 0 & \mathbf{1} \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & \mathbf{1} \end{array} \Rightarrow R^{(3)}=\begin{array}{c|cccc} & a & b & c & d \\ a & 0 & 1 & 0 & 1 \\ b & 0 & 0 & 0 & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 1 \end{array} \Rightarrow R^{(4)}=\begin{array}{c|cccc} & a & b & c & d \\ a & \mathbf{1} & 1 & \mathbf{1} & 1 \\ b & \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \\ c & 0 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 1 \end{array}$$

Warshall算法

- 算法伪代码

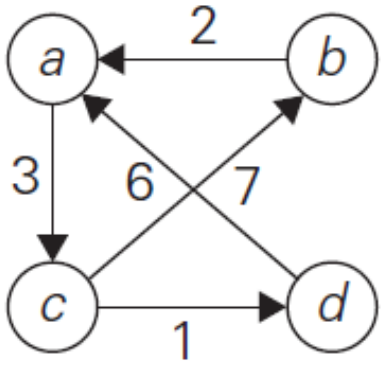
ALGORITHM *Warshall*$(A[1..n, 1..n])$

//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph
$R^{(0)} \leftarrow A$
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 $R^{(n)}$

- 效率类型： $\Theta(n^3)$. 是否可以改进？

求解完全最短路径问题

- 问题描述：给定一个加权连通图（无向或有向），求解每个顶点到其它所顶点之间的最短路径的长度（该长度可以记录在一个距离矩阵中）



有向图

$$W = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

权重矩阵

$$D = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 7 & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

距离矩阵

- 广泛应用
  - 通信、交通、运筹学等

Floyd算法

- 基本思想：通过一系列n阶矩阵来构造距离矩阵

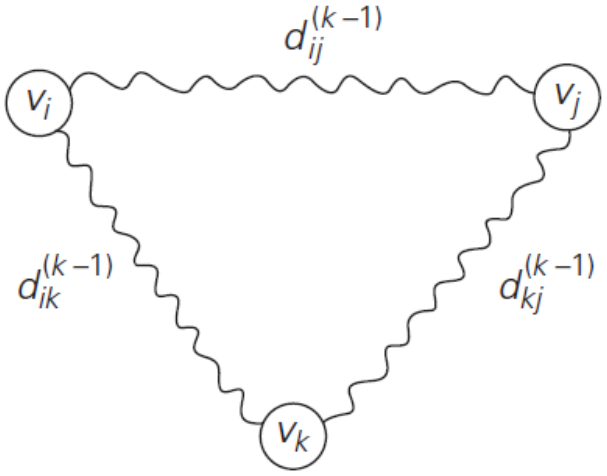
$$\boldsymbol{D}^{(0)},\ldots,\boldsymbol{D}^{(k-1)},\boldsymbol{D}^{(k)},\ldots,\boldsymbol{D}^{(n)}$$

  - 矩阵$\boldsymbol{D}^{(0)}$为权重矩阵
  - 矩阵$\boldsymbol{D}^{(k)}$：$[\boldsymbol{D}^{(k)}]_{i,j}=d_{ij}^{(k)}$表示第i个顶点到第j个顶点之间所有路径中一条最短路径的长度，并且路径的每个中间节点编号不大于k

- 如何迭代地从矩阵$\boldsymbol{D}^{(k-1)}$来构造$\boldsymbol{D}^{(k)}$呢？

Floyd算法

- 节点v_i到节点v_j间可能的两种路径形式

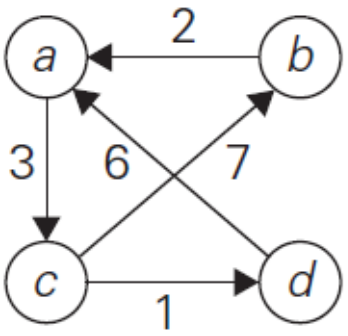


- 递推式

$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \quad \text{for } k \geq 1,\ d_{ij}^{(0)} = w_{ij}.$$

Floyd算法

- 举例



$$D^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array} \Rightarrow D^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \mathbf{5} & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \mathbf{9} & 0 \end{bmatrix} \end{array} \Rightarrow$$

$$D^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array} \Rightarrow D^{(3)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \mathbf{10} & 3 & \mathbf{4} \\ 2 & 0 & 5 & \mathbf{6} \\ 9 & 7 & 0 & 1 \\ 6 & \mathbf{16} & 9 & 0 \end{bmatrix} \end{array} \Rightarrow D^{(4)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ \mathbf{7} & 7 & 0 & 1 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

Floyd算法

- 算法伪代码

ALGORITHM *Floyd*$(W[1..n, 1..n])$

//Implements Floyd's algorithm for the all-pairs shortest-paths problem
//Input: The weight matrix W of a graph with no negative-length cycle
//Output: The distance matrix of the shortest paths' lengths
$D \leftarrow W$ //is not necessary if W can be overwritten
for $k \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** n **do**
        for $j \leftarrow 1$ **to** n **do**
            $D[i, j] \leftarrow \min\{D[i, j], D[i, k] + D[k, j]\}$
return D

- 效率类型： $\Theta(n^3)$

目录

- 三个基本例子
  - 币值最大化问题
  - 找零问题
  - 硬币收集问题
- 背包问题
- Warshall算法和Floyd算法
- 最优二叉查找树

最优二叉查找树

- 问题描述：设$a_1, a_2, \ldots, a_n$是从小到大排列的互不相等的键，$p_1, p_2, \ldots, p_n$是它们的查找频率，则构造一棵最优二叉树，使在树中成功查找键的平均查找次数最少
- 蛮力法：考虑所有可能的包含n个键的二叉树，做评估和比较
  - 总数量 = 第n个卡塔兰数C(2n,n)/(n+1)，指数级增长
- 分析该问题是否适合动态规划法：
  - 是否具有最优解性质？如果该树是一棵最优二叉查找树，那么其子树也必然是最优查找二叉树
  - 是否有重复子问题？递推关系式是？

最优二叉查找树

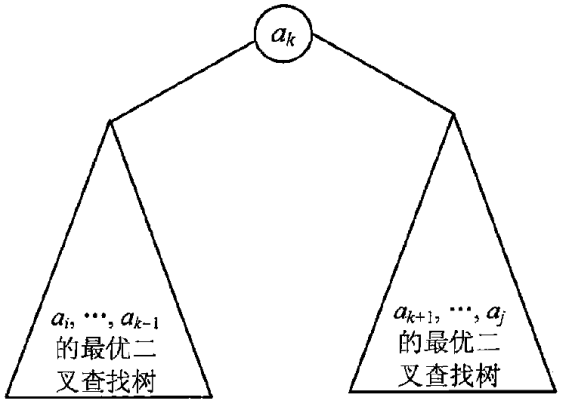
- 假设T_i^j是由键$a_i, a_{i+1}, \ldots, a_j$构成的二叉树，$C(i,j)$是这棵树中成功查找的最小的平均查找次数，则有

根（成功）　左子树（成功）　右子树（成功）

$$C(i,j) = \min_{i \le k \le j} \{p_k \cdot 1 + \sum_{s=i}^{k-1} p_s \cdot (\text{level of } a_s \text{ in } T_i^{k-1} + 1) + \sum_{s=k+1}^{j} p_s \cdot (\text{level of } a_s \text{ in } T_{k+1}^{j} + 1)\}$$

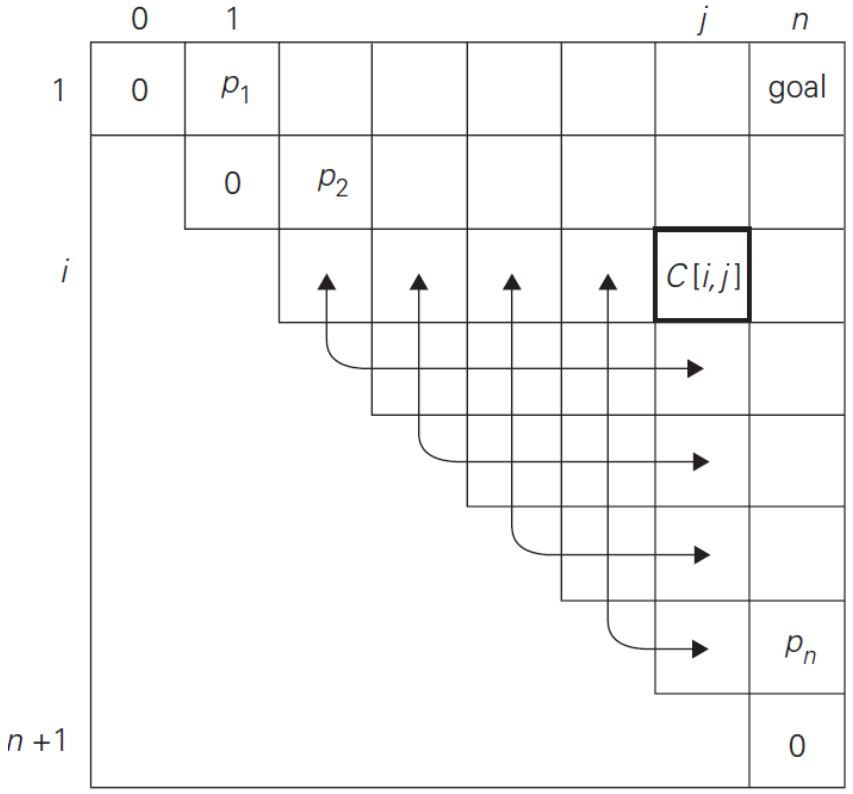
$$= \min_{i \le k \le j} \{\sum_{s=i}^{k-1} p_s \cdot \text{level of } a_s \text{ in } T_i^{k-1} + \sum_{s=k+1}^{j} p_s \cdot \text{level of } a_s \text{ in } T_{k+1}^{j} + \sum_{s=i}^{j} p_s\}$$

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



最优二叉查找树

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



动态规划算法的表格

- 主对角线全部填0
- 给定概率p_i填在对角线的右上方
- 箭头指出了计算$C(i,j)$时所需要的求和对的单元格

最优二叉查找树

- 举例

key	A	B	C	D
probability	0.1	0.2	0.4	0.3

j

主表

i	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					

$$C(1,2)=\min\{C(1,0)+C(2,2),C(1,1)+C(3,2)\}+\sum_{s=1}^{2}p_s$$
$$=\min\{0+0.2,0.1+0\}+0.3=0.4$$

此时k=2

最优二叉查找树

- 举例

key	A	B	C	D
probability	0.1	0.2	0.4	0.3

j　　主表　　根表

i

	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

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

$$C(2,3)=\min\{C(2,1)+C(3,3),C(2,2)+C(4,3)\}$$
$$+\sum_{s=2}^{3} p_s=\min\{0+0.4,0.2+0\}+0.6=0.8$$ 此时k=3

最优二叉查找树

- 举例

key	A	B	C	D
probability	0.1	0.2	0.4	0.3

j

i

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					

$$C(3,4)=\min\{C(3,2)+C(4,4),C(3,3)+C(5,4)\}$$

$$+\sum_{s=3}^{4} p_s=\min\{0+0.3,0.4+0\}+0.7=1.0$$

此时k=3

最优二叉查找树

- 举例

key	A	B	C	D
probability	0.1	0.2	0.4	0.3

主表

i \ j	0	1	2	3	4
1	0	0.1	0.4	1.1	
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	
2			2	3	3
3				3	3
4					4
5					

$$C(2,4)=\min\{C(2,1)+C(3,4),C(2,2)+C(4,4),C(2,3)+C(5,4)\}$$

$$+\sum_{s=2}^{4} p_s=\min\{0+1.0,0.2+0.3,0.8+0\}+0.9=1.4$$ 此时k=3

最优二叉查找树

- 举例

key	A	B	C	D
probability	0.1	0.2	0.4	0.3

j

i

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

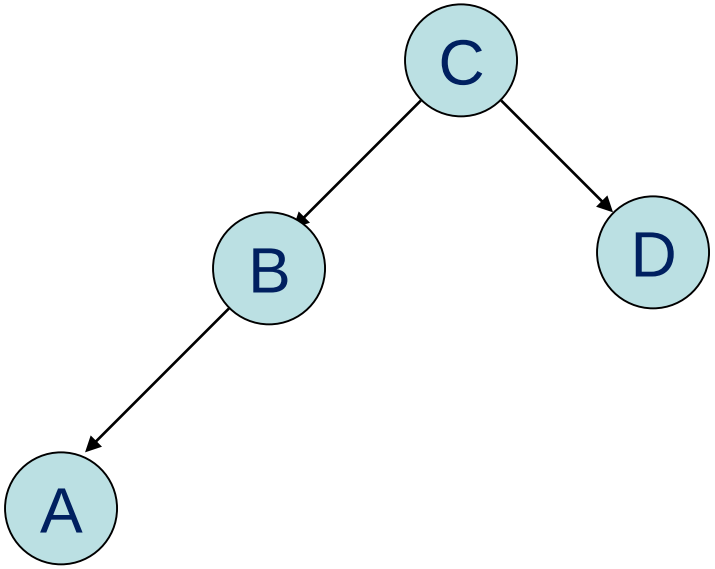
根表

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

$$C(1,4)=\min\{C(1,0)+C(2,4),C(1,1)+C(3,4),C(1,2)+C(4,4)$$
$$C(1,3)+C(5,4)\}+\sum_{s=1}^{4}p_s=\min\{0+1.4,0.1+1.0,$$
$$0.4+0.3,1.1+0\}+1=1.7$$

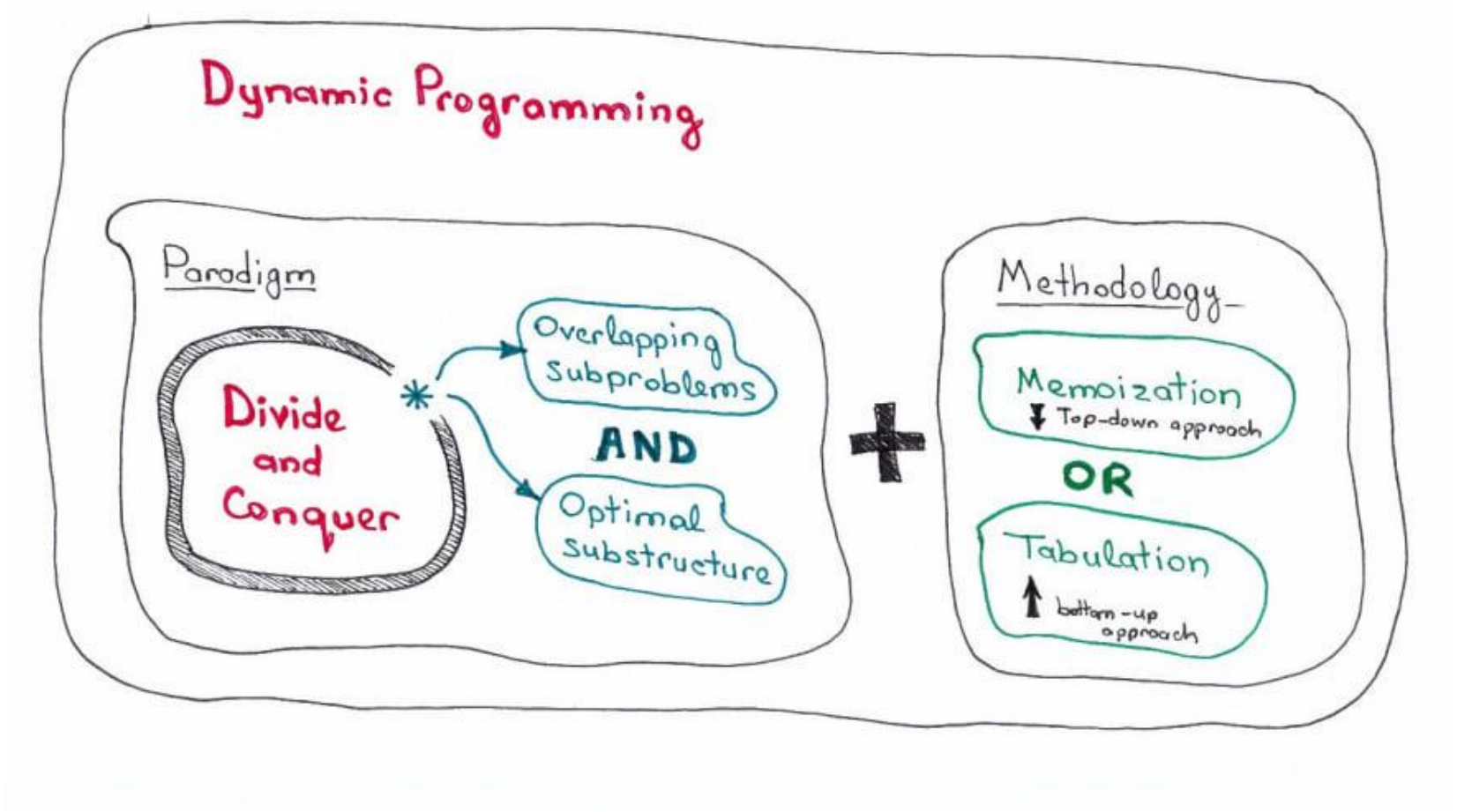
此时k=3

最优二叉查找树



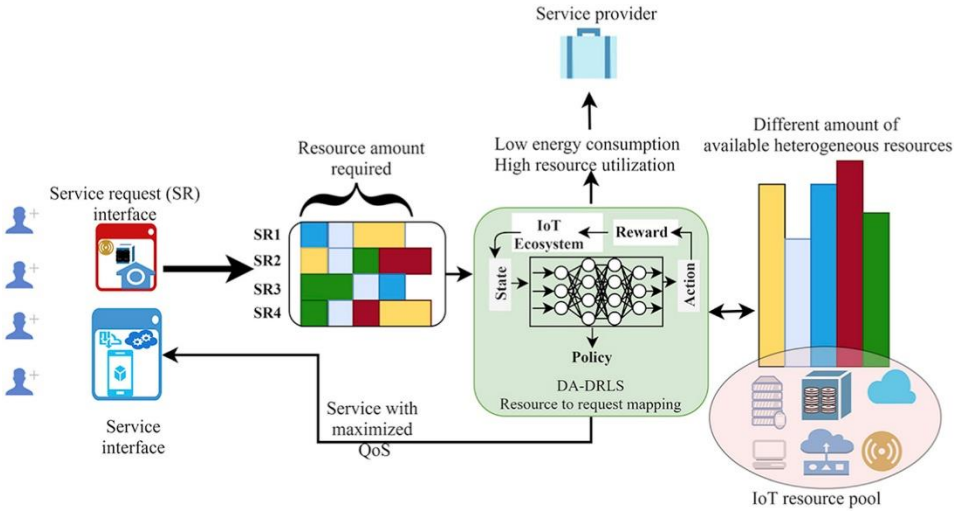
最终的最优二叉查找树

小结


Dynamic Programming
Paradigm
Divide and Conquer
Overlapping Subproblems
AND
Optimal Substructure
Methodology
Memoization
Top-down approach
OR
Tabulation
bottom-up approach

强化学习：应用举例

AlphaGo

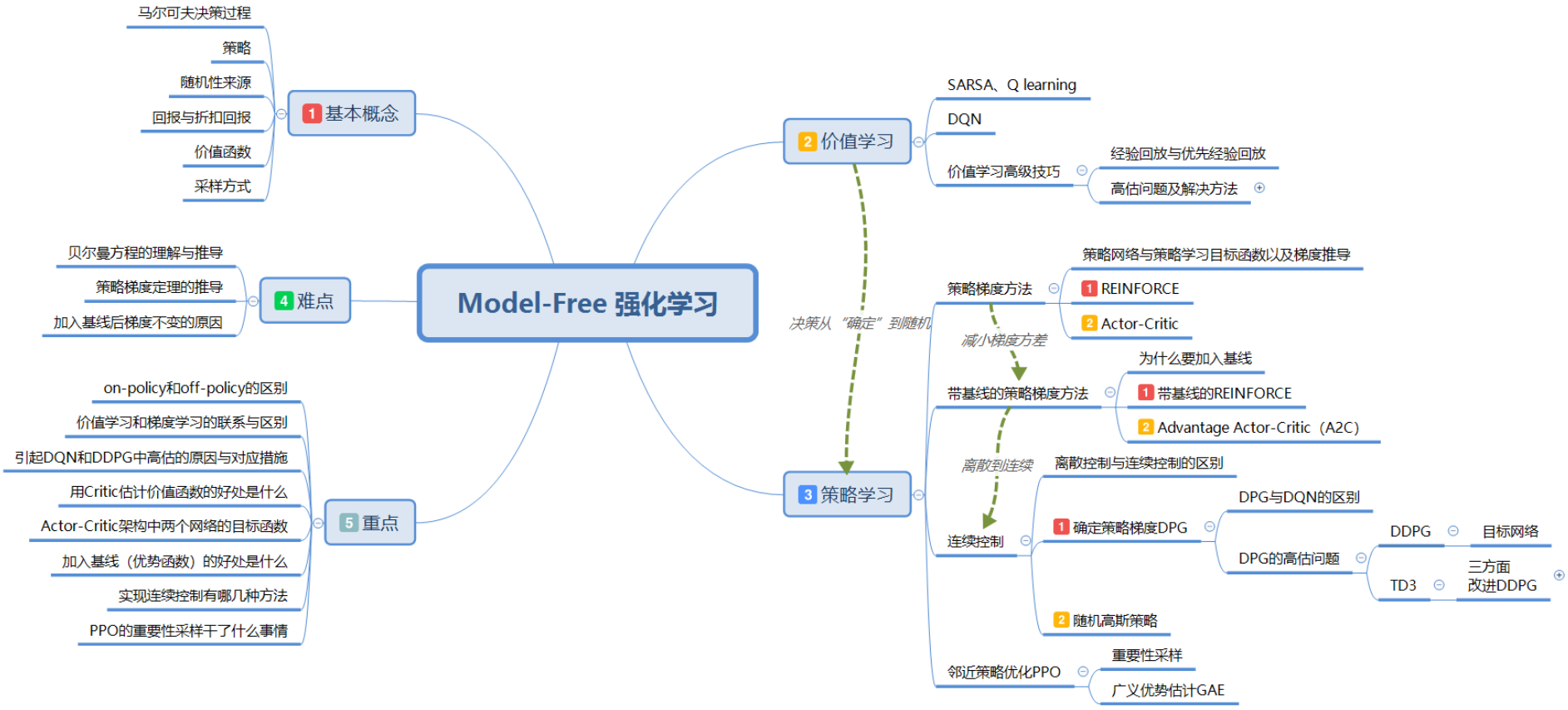


无线网络资源管理

动态规划 vs 强化学习

- 相似之处：
  - 都涉及在一个序列或者状态空间中做出一系列决策，以达到最优化的目标
  - 都涉及在未来可能发生的事件上进行考虑，即考虑长期回报或长期利益
- 区别：
  - **反馈类型**：在动态规划中，系统的环境通常是已知的、静态的，并且对智能体的行为没有直接的反馈。而在强化学习中，智能体与环境之间存在着交互，通过与环境的交互获得反馈信号，这些信号指示了智能体行为的好坏
  - **知识先验**：在动态规划中，通常假设有关环境动态的完全知识，即转移概率和奖励函数是已知的。而在强化学习中，通常并不假设对环境的先验知识，智能体需要通过与环境的交互来学习
  - **问题类型**：动态规划通常用于解决离散状态和动作空间的问题，比如最短路径问题、背包问题等。而强化学习更适用于连续状态和动作空间的问题，比如机器人控制、游戏玩法等

（单智能体）强化学习


Model-Free 强化学习
1 基本概念
马尔可夫决策过程
策略
随机性来源
回报与折扣回报
价值函数
采样方式
2 价值学习
SARSA、Q learning
DQN
价值学习高级技巧
经验回放与优先经验回放
高估问题及解决方法
3 策略学习
决策从“确定”到随机
策略梯度方法
策略网络与策略学习目标函数以及梯度推导
1 REINFORCE
2 Actor-Critic
减小梯度方差
带基线的策略梯度方法
为什么要加入基线
1 带基线的REINFORCE
2 Advantage Actor-Critic（A2C）
离散到连续
连续控制
离散控制与连续控制的区别
1 确定策略梯度DPG
DPG与DQN的区别
DPG的高估问题
DDPG
目标网络
TD3
三方面改进DDPG
2 随机高斯策略
邻近策略优化PPO
重要性采样
广义优势估计GAE
4 难点
贝尔曼方程的理解与推导
策略梯度定理的推导
加入基线后梯度不变的原因
5 重点
on-policy和off-policy的区别
价值学习和梯度学习的联系与区别
引起DQN和DDPG中高估的原因与对应措施
用Critic估计价值函数的好处是什么
Actor-Critic架构中两个网络的目标函数
加入基线（优势函数）的好处是什么
实现连续控制有哪几种方法
PPO的重要性采样干了什么事情

强化学习：基本概念（1）

1. 马尔可夫决策过程 MDP

$$(S, A, P, R, \gamma)$$

$$P_{ss'}^{a} = P\left[S_{t+1} = s' | S_t = s, A_t = a\right]$$

2. 策略

A. 随机策略

$$\pi(a|s) = \mathbb{P}(A = a \,|\, S = s)$$

$$\pi(\text{左} \,|\, s) = 0.2,$$
$$\pi(\text{右} \,|\, s) = 0.1,$$
$$\pi(\text{上} \,|\, s) = 0.7.$$

B. 确定策略

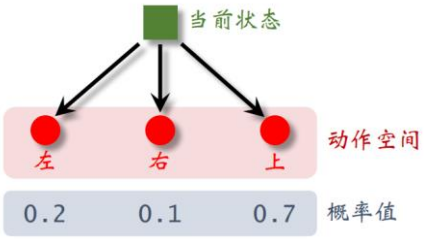
$$\mu : \mathcal{S} \mapsto \mathcal{A}$$

$$\pi(a \,|\, s) = \begin{cases} 1, & \text{if } \mu(s) = a; \\ 0, & \text{otherwise.} \end{cases}$$

强化学习：基本概念（2）

3. 随机性来源

A. 动作随机性--随机决策



B. 状态随机性--转移随机性

马尔可夫性

$$\mathbb{P}(S_{t+1} \mid S_t, A_t) = \mathbb{P}(S_{t+1} \mid S_1, A_1, S_2, A_2, \cdots, S_t, A_t).$$

C. 奖励随机性

$$r_t = r(s_t, a_t) \Longrightarrow R_t = r(s_t, A_t) \quad \text{或} \quad R_t = r(S_t, A_t)$$

D. 轨迹

$$\cdots \; s_t \to a_t \to \begin{matrix} s_{t+1} \\ r_t \end{matrix} \to a_{t+1} \to \begin{matrix} s_{t+2} \\ r_{t+1} \end{matrix} \to a_{t+2} \to \begin{matrix} s_{t+3} \\ r_{t+2} \end{matrix} \cdots$$

已经观测到的值： $s_1, a_1, r_1, \; s_2, a_2, r_2, \; \cdots, \; s_{t-1}, a_{t-1}, r_{t-1}, \; s_t$

尚未被观测到的值：$A_t, R_t, \; S_{t+1}, A_{t+1}, R_{t+1}, \; S_{t+2}, A_{t+2}, R_{t+2}$

强化学习：基本概念（3）

4. 回报与折扣回报

从 t 时刻起，未来所有奖励的（加权）和

A. 回报--累计奖励

$$U_t = R_t + R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_n$$

B. 折扣回报

$$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \cdots$$

5. 价值函数--回报的期望--评判好坏的工具

A. 动作价值函数

回报 U_t 在t时刻仍是随机变量，判断局势好坏需要对其求期望，消除随机性

$$Q_\pi(s_t, a_t) = \mathbb{E}_{S_{t+1}, A_{t+1}, \cdots, S_n, A_n}\left[U_t \,\middle|\, S_t = s_t, A_t = a_t\right]$$

B. 最优动作价值函数

$$Q_\star(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t), \quad \forall\, s_t \in \mathcal{S}, \quad a_t \in \mathcal{A}.$$

$$\pi^\star = \operatorname*{argmax}_\pi Q_\pi(s_t, a_t), \quad \forall\, s_t \in \mathcal{S}, \quad a_t \in \mathcal{A}.$$

$$Q_\star(s_t, \text{左}) = 130, \quad Q_\star(s_t, \text{右}) = -50, \quad Q_\star(s_t, \text{上}) = 296$$

C. 状态价值函数

评估当前状态的胜算（好坏）

$$V_\pi(s_t) = \mathbb{E}_{A_t \sim \pi(\cdot|s_t)}\left[Q_\pi(s_t, A_t)\right] = \sum_{a \in \mathcal{A}} \pi(a|s_t) \cdot Q_\pi(s_t, a).$$

强化学习：基本概念（4）

6. 采样方式--用样本近似价值函数

（计算值函数）

数据收集
策略评估

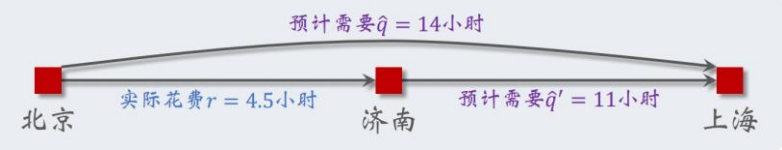
策略改进
与更新

on-policy
off-policy

（依据计算结果更新策略）

求值函数是求期望（离散时要遍历所有情况加权求和、连续时求定积分），难以获得足够数据且计算量大

因而需要使用蒙特卡洛方法对值函数进行采样，近似估计值函数的大小（数学原理：大数定律）



A. 蒙特卡洛采样法（MC）无偏，方差大 $u_t=\sum_{k=t}^{n}\gamma^{k-t}\cdot r_k$

B. 时间差分采样法（TD） 有偏，方差小

$$V(S_t)\leftarrow V(S_t)+\alpha\left(R_{t+1}+\gamma V(S_{t+1})-V(S_t)\right)$$

$$R_{t+1}+\gamma V(S_{t+1})$$

课后作业

章 X	节 X. Y	课后作业题 Z	思考题 Z
8	8.1	2,4,5	6,7,11
	8.2	5	9
	8.3	1,3	11
	8.4	1,7	11

注：只需上交“课后作业题”；以“学号姓名_chX. pdf ”规范命名，提交到“学在浙大”指定文件夹。DDL：2024年4月23日