动态规划

讲授者 王爱娟

aijuan321@foxmail.com

重庆理工大学 计算机科学与工程学院

August 22, 2024

目录

回顾：分治法

分治法的基本思想

1. 将规模较大的问题分解为规模较小的问题
2. 解决这些小问题
3. 然后将小问题的解合并成为原来大问题的解

回顾：分治法

分治法的基本思想

1. 将规模较大的问题分解为规模较小的问题
2. 解决这些小问题
3. 然后将小问题的解合并成为原来大问题的解

分治法的基础

1. 小问题比大问题更容易解决
2. 将小问题的解组装成大问题的解比直接求解大问题成本更低
3. 小问题又可以按照同样的方法分解为更小的问题，便于将问题的规模进一步缩小

分治法面临的问题

问题：大量重复的小问题

1. 在分解原问题为小问题的过程中，有时会产生大量重复的小问题
2. 对这些重复问题的计算，产生了巨大的时间开销。

分治法面临的问题

问题：大量重复的小问题

1. 在分解原问题为小问题的过程中，有时会产生大量重复的小问题
2. 对这些重复问题的计算，产生了巨大的时间开销。

例子：Fibonacci序列问题

$$f(n)=\begin{cases}1 & n=1,2\\ f(n-1)+f(n-2) & n\geq 3\end{cases}$$

可用分治法解决该问题

分治法面临的问题

问题：大量重复的小问题

1. 在分解原问题为小问题的过程中，有时会产生大量重复的小问题
2. 对这些重复问题的计算，产生了巨大的时间开销。

例子：Fibonacci序列问题

$$f(n) = \begin{cases} 1 & n = 1, 2 \\ f(n-1) + f(n-2) & n \geq 3 \end{cases}$$

可用分治法解决该问题

1. 规模为n 的问题$f(n)$ 被划分为规模更小的两个子问题$f(n-1)$ 和$f(n-2)$
2. 求解这两个子问题的解之后，通过简单组合(加法操作)，即可求原问题解
3. 子问题可以采用同样地方式进一步缩减规模(直至为1或2)

基于分治法的Fibonacci序列算法伪代码

Fibonacci(n)

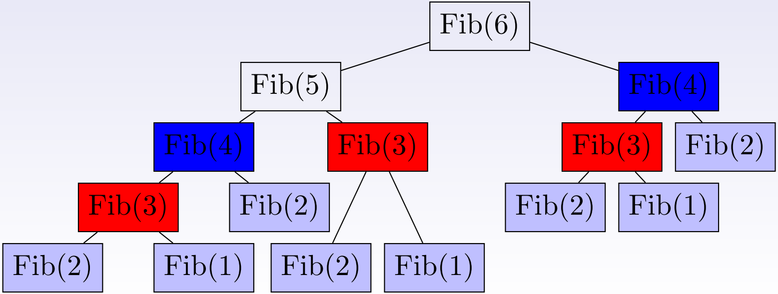
```
if n == 1 or n == 2 then
    return n
else
    return Fibonacci(n-1) + Fibonacci(n-2)
end if
```

基于分治法的Fibonacci序列算法伪代码

Fibonacci(n)

```
if n == 1 or n == 2 then
    return n
else
    return Fibonacci(n-1) + Fibonacci(n-2)
end if
```

Fib(6)的计算过程



分治法面临的问题

Fib(6) 的问题

1. Fib(4) 计算了两次，Fib(3) 计算了三次
2. 采用分治法计算Fib(n)，n 越大，分解出的相同子问题就越多

分治法面临的问题

Fib(6) 的问题

1. Fib(4) 计算了两次，Fib(3) 计算了三次
2. 采用分治法计算Fib(n)，n 越大，分解出的相同子问题就越多

定义 (交叠子问题)

在划分的一组子问题中，存在若干或大量的雷同子问题，它们称为交叠子问题

分治法面临的问题

Fib(6) 的问题

1. Fib(4) 计算了两次，Fib(3) 计算了三次
2. 采用分治法计算Fib(n)，n 越大，分解出的相同子问题就越多

定义 (交叠子问题)

在划分的一组子问题中，存在若干或大量的雷同子问题，它们称为交叠子问题

解决办法：避免相同子问题的重复计算

1. 对子问题只求解一次，并把它的解记录在表中。
2. 当再次需要求解该子问题时，直接从记录表中查询获取它的解

基于动态规划的Fibonacci序列算法伪代码

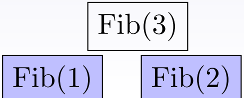
Fibonacci(n)

```
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n do
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)
```

基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

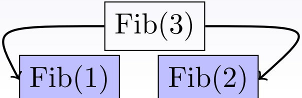
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

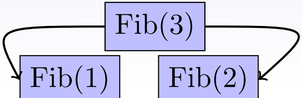
```
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n do
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)
```



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

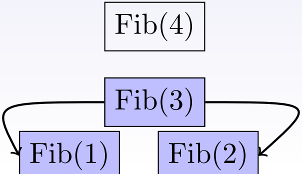
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

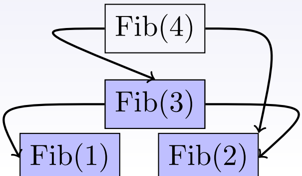
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

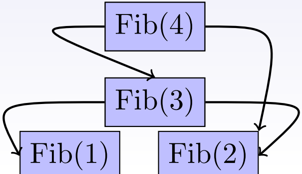
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

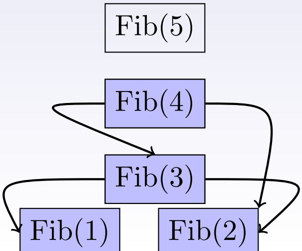
```
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n do
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)
```



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

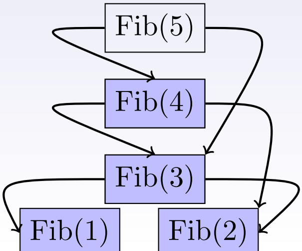
```
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n do
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)
```



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

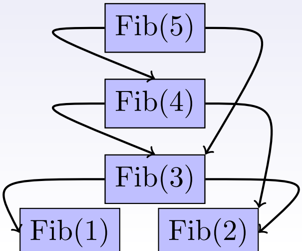
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

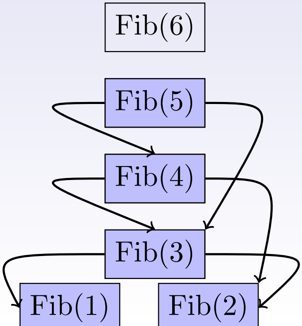
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

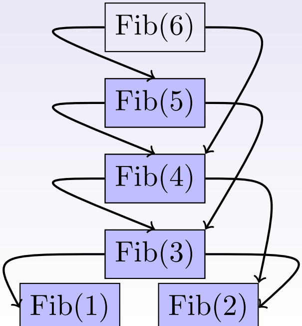
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
  Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

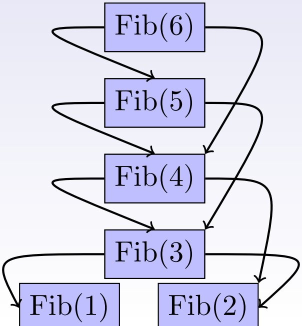
Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
  Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



基于动态规划的Fibonacci序列算法伪代码

Fibonacci(n)

Fibonacci(1) = 1;
Fibonacci(2) = 2;
for i = 3 to n **do**
    Fibonacci(i) = Fibonacci(i-1) + Fibonacci(i-2)
end for
return Fibonacci(n)



动态规划与分治思想的比较

动态规划与分治思想的比较

1. 二者思想类似：将大问题划分为若干小问题
2. 分治算法是采用自上而下的方式求值，导致了不止一次的递归调用
3. 动态规划法是采取自下向上的方式递推求值，并把中间结果存储起来，避免重复运算，从而降低时间复杂度。

动态规划与分治思想的比较

动态规划与分治思想的比较

1. 二者思想类似：将大问题划分为若干小问题
2. 分治算法是采用自上而下的方式求值，导致了不止一次的递归调用
3. 动态规划法是采取自下向上的方式递推求值，并把中间结果存储起来，避免重复运算，从而降低时间复杂度。

动态规划的主要目的

动态规划提出来的主要目的是优化，即不只是解决一个问题，而是**以最优的方式解决这个问题**，或者说，针对特定问题寻求最优解。

动态规划与分治思想的比较

动态规划与分治思想的比较

1. 二者思想类似：将大问题划分为若干小问题
2. 分治算法是采用自上而下的方式求值，导致了不止一次的递归调用
3. 动态规划法是采取自下向上的方式递推求值，并把中间结果存储起来，避免重复运算，从而降低时间复杂度。

动态规划的主要目的

动态规划提出来的主要目的是优化，即不只是解决一个问题，而是**以最优的方式解决这个问题**，或者说，针对特定问题寻求最优解。

发明者

动态规划由理查德.贝尔曼(Richard E. Bellman)于1957年在其著作《动态规划（Dynamic Programming）》一书中提出。

币值最大问题

问题

给定一排硬币（n 个），其面值均为正整数$c_1, c_2, \cdots, c_n$。如何选择硬币，使得在其原位置互不相邻的条件下，所选硬币的总金额最大。

币值最大问题

问题

给定一排硬币（n 个），其面值均为正整数$c_1, c_2, \cdots, c_n$。如何选择硬币，使得在其原位置互不相邻的条件下，所选硬币的总金额最大。

实例

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	–	5	1	2	10	6	2
F	0	5					

币值最大问题

问题

给定一排硬币（n 个），其面值均为正整数$c_1, c_2, \cdots, c_n$。如何选择硬币，使得在其原位置互不相邻的条件下，所选硬币的总金额最大。

实例

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	–	5	1	2	10	6	2
F	0	5					

选择方法的表示

假设$F(n)$表示在n个可选硬币下，所选硬币的最大金额，则

$$F(n) = \begin{cases} \max\{c_n + F(n-2), F(n-1)\}, & n > 1 \\ c_1, & n = 1 \\ 0, & n = 0 \end{cases}$$

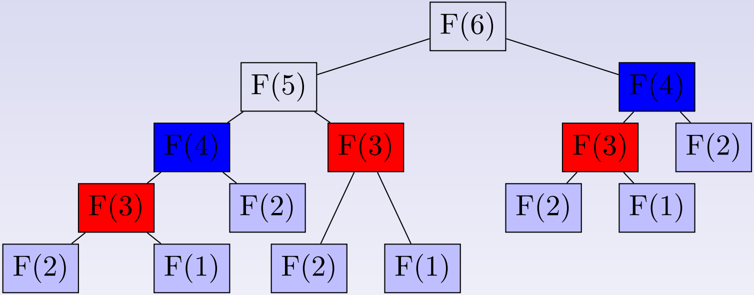
基于递归分治的币值最大化问题算法

F(n)

数组C[1..n] 保存n个硬币的面值
if n=1 **then**
  return C_1
else if n=0 **then**
  return 0
end if
if n $\geq$ 2 **then**
  $F(n) = \max\{C_n + F(n-2), F(n-1)\}$
end if
return F(n)

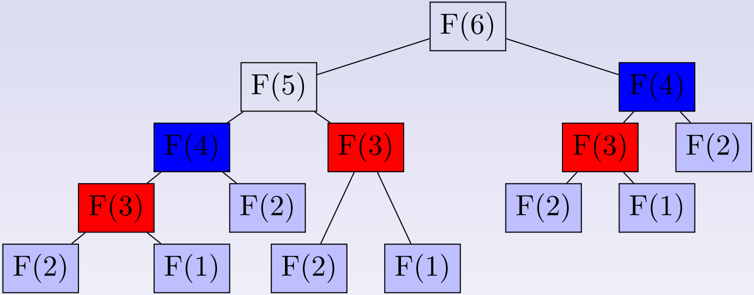
基于递归分治的币值最大化问题算法

F(6)的计算过程



基于递归分治的币值最大化问题算法

F(6)的计算过程



问题

存在大量的相同子问题，导致大量的重复计算

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式

$$F(2) = \max\{C_2 + F(2-2), F(2-1)\}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式

$$\begin{aligned} F(2) &= \max\{C_2 + F(2-2), F(2-1)\} \\ &= \max\{1 + F(0), F(1)\} \end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式

$$\begin{aligned} F(2) &= \max\{C_2 + F(2-2), F(2-1)\} \\ &= \max\{1 + F(0), F(1)\} \\ &= \max\{1, 5\} = 5 \end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式

$$\begin{aligned} F(2) &= \max\{C_2 + F(2-2), F(2-1)\} \\ &= \max\{1 + F(0), F(1)\} \\ &= \max\{1, 5\} = 5 \\ F(3) &= \max\{C_3 + F(3-2), F(3-1)\} \end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式

$$\begin{aligned} F(2) &= \max\{C_2 + F(2-2), F(2-1)\} \\ &= \max\{1 + F(0), F(1)\} \\ &= \max\{1, 5\} = 5 \\ F(3) &= \max\{C_3 + F(3-2), F(3-1)\} \\ &= \max\{2 + F(1), F(2)\} \end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式

$$\begin{aligned}
F(2) &= \max\{C_2 + F(2-2), F(2-1)\} \\
&= \max\{1 + F(0), F(1)\} \\
&= \max\{1, 5\} = 5 \\
F(3) &= \max\{C_3 + F(3-2), F(3-1)\} \\
&= \max\{2 + F(1), F(2)\} \\
&= \max\{7, 5\} = 7
\end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$
$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式

$$\begin{aligned}
F(2) &= \max\{C_2 + F(2-2), F(2-1)\} \\
&= \max\{1 + F(0), F(1)\} \\
&= \max\{1, 5\} = 5 \\
F(3) &= \max\{C_3 + F(3-2), F(3-1)\} \\
&= \max\{2 + F(1), F(2)\} \\
&= \max\{7, 5\} = 7 \\
F(4) &= \max\{C_4 + F(4-2), F(4-1)\}
\end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$
$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式

$$\begin{aligned}
F(2) &= \max\{C_2 + F(2-2), F(2-1)\} \\
&= \max\{1 + F(0), F(1)\} \\
&= \max\{1, 5\} = 5 \\
F(3) &= \max\{C_3 + F(3-2), F(3-1)\} \\
&= \max\{2 + F(1), F(2)\} \\
&= \max\{7, 5\} = 7 \\
F(4) &= \max\{C_4 + F(4-2), F(4-1)\} \\
&= \max\{10 + F(2), F(3)\} = \max\{15, 7\} = 15
\end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式(续)

$$F(5) = \max\{C_5 + F(5-2), F(5-1)\}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式(续)

$$\begin{aligned} F(5) &= \max\{C_5 + F(5-2), F(5-1)\} \\ &= \max\{6 + F(3), F(4)\} \end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式(续)

$$\begin{aligned} F(5) &= \max\{C_5 + F(5-2), F(5-1)\} \\ &= \max\{6 + F(3), F(4)\} \\ &= \max\{13, 15\} = 15 \end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式(续)

$$\begin{aligned} F(5) &= \max\{C_5 + F(5-2), F(5-1)\} \\ &= \max\{6 + F(3), F(4)\} \\ &= \max\{13, 15\} = 15 \\ F(6) &= \max\{C_6 + F(6-2), F(6-1)\} \end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式(续)

$$\begin{aligned} F(5) &= \max\{C_5 + F(5-2), F(5-1)\} \\ &= \max\{6 + F(3), F(4)\} \\ &= \max\{13, 15\} = 15 \\ F(6) &= \max\{C_6 + F(6-2), F(6-1)\} \\ &= \max\{2 + F(4), F(5)\} \end{aligned}$$

币值最大问题

coin's index(i)	0	1	2	3	4	5	6
coin's value(c_i)	-	5	1	2	10	6	2

$$F(n) = \max\{C_n + F(n-2), F(n-1)\}, n > 1$$

$$F(0) = 0, F(1) = 5$$

自下而上求解:动态规划的币值最大问题求解方式(续)

$$\begin{aligned} F(5) &= \max\{C_5 + F(5-2), F(5-1)\} \\ &= \max\{6 + F(3), F(4)\} \\ &= \max\{13, 15\} = 15 \\ F(6) &= \max\{C_6 + F(6-2), F(6-1)\} \\ &= \max\{2 + F(4), F(5)\} \\ &= \max\{17, 15\} = 17 \end{aligned}$$

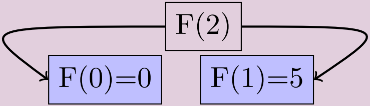
自下而上求解过程图示

$\boxed{F(2)}$

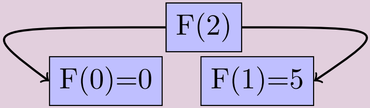
$\boxed{F(0)=0}$ $\boxed{F(1)=5}$

币值最大问题

自下而上求解过程图示

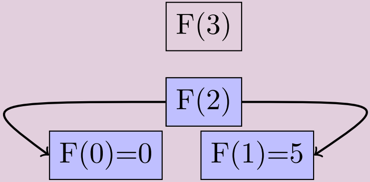


自下而上求解过程图示



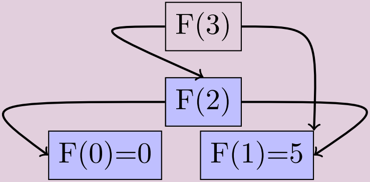
币值最大问题

自下而上求解过程图示



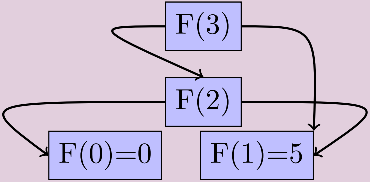
币值最大问题

自下而上求解过程图示



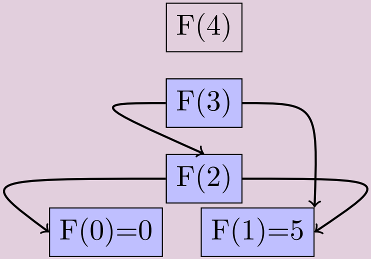
币值最大问题

自下而上求解过程图示

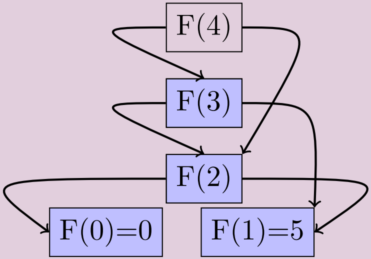


币值最大问题

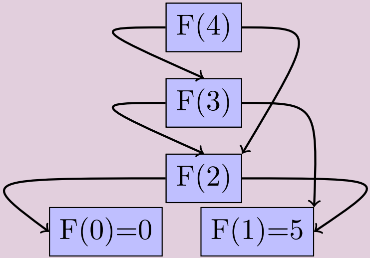
自下而上求解过程图示



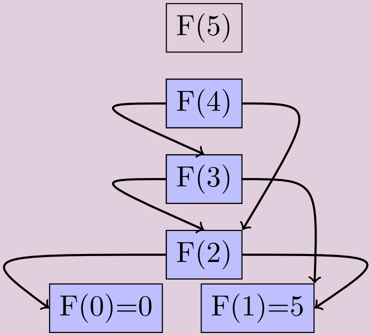
自下而上求解过程图示



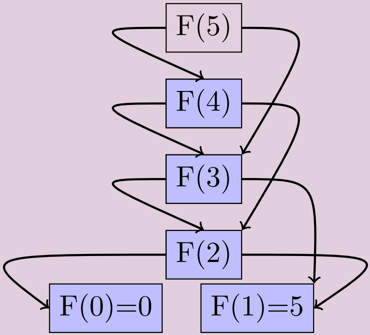
自下而上求解过程图示



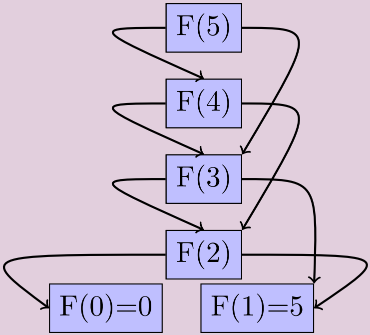
自下而上求解过程图示



自下而上求解过程图示

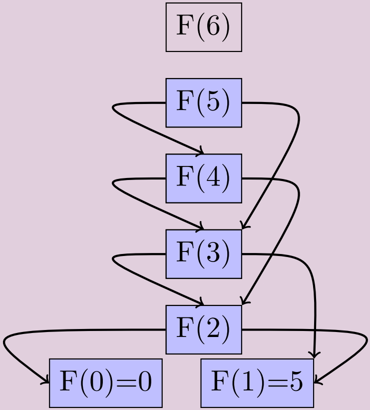


自下而上求解过程图示

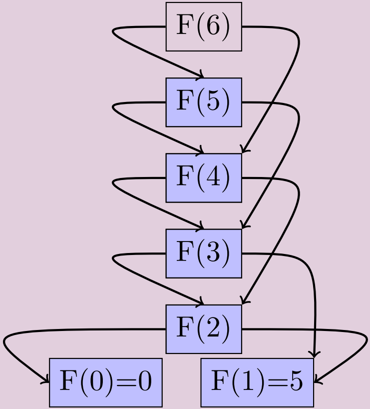


币值最大问题

自下而上求解过程图示

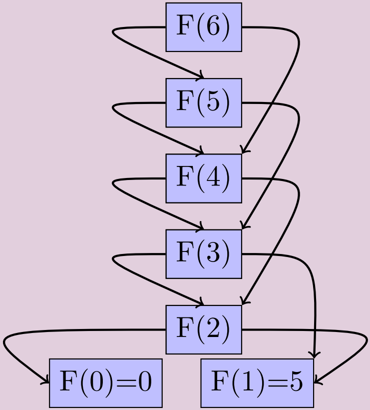


自下而上求解过程图示



币值最大问题

自下而上求解过程图示



基于动态规划的币值最大问题算法

CoinRow(C[1⋯n])

//算法基础：$F(n) = \max\{C_n + F(n-2), F(n-1)\} \quad n > 1$

//输入: C[1⋯n]表示n 个硬币及其对应的值

//输出：能选取的最大币值

```
F[0] = 0; F[1] = 1
for i = 2 to n do
    F[i] = max{C[i] + F[i-2], F[i-1]}
end for
return F[n]
```

找零问题

找零问题

有m种规格的硬币，其面值依次为：$d_1 < d_2 < \cdots < d_m$。现需要找零给客户，其找零金额为n，请问最少需要多少个硬币？

找零问题

找零问题

有m种规格的硬币，其面值依次为：$d_1 < d_2 < \cdots < d_m$。现需要找零给客户，其找零金额为n，请问最少需要多少个硬币？

实例

假设有3个硬币，它们的面值依次为1，3，6

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–						

找零问题

找零问题

有m种规格的硬币，其面值依次为：$d_1 < d_2 < \cdots < d_m$。现需要找零给客户，其找零金额为n，请问最少需要多少个硬币？

实例

假设有3个硬币，它们的面值依次为1，3，6

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–						

假设F(n)表示找零金额为n所需的最小硬币数

- 如果第一个硬币被用于找零，则$F(n) = F(n-d_1) + 1$
- 如果第二个硬币被用于找零，则$F(n) = F(n-d_2) + 1$
- 如果第三个硬币被用于找零，则$F(n) = F(n-d_3) + 1$
- 比较这三种选择，最优选择应该如下：
  $F(n) = \min\{F(n-d_1)+1, F(n-d_2)+1, F(n-d_3)+1\}$

找零问题

找零问题

有m种规格的硬币，其面值依次为：$d_1 < d_2 < \cdots < d_m$。现需要找零给客户，其找零金额为n，请问最少需要多少个硬币？

实例

假设有3个硬币，它们的面值依次为1，3，6

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–						

假设F(n)表示找零金额为n所需的最小硬币数

- 如果第一个硬币被用于找零，则$F(n) = F(n-d_1) + 1$
- 如果第二个硬币被用于找零，则$F(n) = F(n-d_2) + 1$
- 如果第三个硬币被用于找零，则$F(n) = F(n-d_3) + 1$
- 比较这三种选择，最优选择应该如下：
  $F(n) = \min\{F(n-d_1)+1, F(n-d_2)+1, F(n-d_3)+1\}$

找零问题

有m种规格的硬币，其面值依次为：$d_1 < d_2 < \cdots < d_m$。现需要找零给客户，其找零金额为n，请问最少需要多少个硬币？

实例

假设有3个硬币，它们的面值依次为1，3，6

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–						

假设F(n)表示找零金额为n所需的最小硬币数

- 如果第一个硬币被用于找零，则$F(n) = F(n\text{-}d_1) + 1$
- 如果第二个硬币被用于找零，则$F(n) = F(n\text{-}d_2) + 1$
- 如果第三个硬币被用于找零，则$F(n) = F(n\text{-}d_3) + 1$
- 比较这三种选择，最优选择应该如下：$F(n) = \min\{F(n-d_1)+1, F(n-d_2)+1, F(n-d_3)+1\}$

找零问题

找零问题

有m种规格的硬币，其面值依次为：$d_1 < d_2 < \cdots < d_m$。现需要找零给客户，其找零金额为n，请问最少需要多少个硬币？

实例

假设有3个硬币，它们的面值依次为1，3，6

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–						

假设F(n)表示找零金额为n所需的最小硬币数

- 如果第一个硬币被用于找零，则$F(n) = F(n-d_1) + 1$
- 如果第二个硬币被用于找零，则$F(n) = F(n-d_2) + 1$
- 如果第三个硬币被用于找零，则$F(n) = F(n-d_3) + 1$
- 比较这三种选择，最优选择应该如下：
  $F(n) = \min\{F(n-d_1)+1, F(n-d_2)+1, F(n-d_3)+1\}$

找零问题

找零问题

有m种规格的硬币，其面值依次为：$d_1 < d_2 < \cdots < d_m$。现需要找零给客户，其找零金额为n，请问最少需要多少个硬币？

实例

假设有3个硬币，它们的面值依次为1，3，6

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–						

假设F(n)表示找零金额为n所需的最小硬币数

- 如果第一个硬币被用于找零，则$F(n) = F(n-d_1) + 1$
- 如果第二个硬币被用于找零，则$F(n) = F(n-d_2) + 1$
- 如果第三个硬币被用于找零，则$F(n) = F(n-d_3) + 1$
- 比较这三种选择，最优选择应该如下：
  $F(n) = \min\{F(n-d_1)+1, F(n-d_2)+1, F(n-d_3)+1\}$

找零问题

等式扩展

上述等式的前提是只有3个硬币可用于找零。若有m个硬币可用于找零，则上述等式可表示如下，

$$F(n) = \begin{cases} \min\limits_{n \geq d_i}\{F(n-d_i)+1\}, & n > 1, 1 \leq i \leq m \\ 0, & n = 0 \end{cases}$$

找零问题

实例

假设有3个硬币，它们的面值依次为1，3，4，现需要找零6，最少需要多少硬币?

$$F(n) = \begin{cases} \min\limits_{n \geq d_i}\{F(n-d_i)+1\}, & n > 1, 1 \leq i \leq m \\ 0, & n = 0 \end{cases}$$

找零问题

实例

假设有3个硬币，它们的面值依次为1，3，4，现需要找零6，最少需要多少硬币？

$$F(n) = \begin{cases} \min\limits_{n \geq d_i}\{F(n-d_i)+1\}, & n>1, 1 \leq i \leq m \\ 0, & n=0 \end{cases}$$

自下向上：基于动态规划的计算过程

- $\mathrm{F}(1) = \min\{F(1-\mathrm{d}_1)+1\} = \min\{F(0)+1\} = 1$
- $\mathrm{F}(2) = \min\{\mathrm{F}(2\text{-}\mathrm{d}_1)+1\} = \min\{\mathrm{F}(1)+1\} = 2$
- $\mathrm{F}(3) = \min\{\mathrm{F}(3\text{-}\mathrm{d}_1)+1,\ \mathrm{F}(3\text{-}\mathrm{d}_2)+1\} = \min\{\mathrm{F}(2)+1,\ \mathrm{F}(0)+1\} = 1$
- $\mathrm{F}(4) = \min\{\mathrm{F}(4\text{-}\mathrm{d}_1)+1,\ \mathrm{F}(4\text{-}\mathrm{d}_2)+1,\ \mathrm{F}(4\text{-}\mathrm{d}_3)+1\} = \min\{\mathrm{F}(3)+1,\ \mathrm{F}(1)+1,\ \mathrm{F}(0)+1\} = 1$

找零问题

实例

假设有3个硬币，它们的面值依次为1，3，4，现需要找零6，最少需要多少硬币?

$$F(n)=\begin{cases}\min\limits_{n\geq d_i}\{F(n-d_i)+1\}, & n>1, 1\leq i\leq m\\ 0, & n=0\end{cases}$$

自下向上：基于动态规划的计算过程

- $\mathrm{F}(1)=\min\{F(1-\mathrm{d}_1)+1\}=\min\{F(0)+1\}=1$
- $\mathrm{F}(2)=\min\{\mathrm{F}(2\text{-}\mathrm{d}_1)+1\}=\min\{\mathrm{F}(1)+1\}=2$
- $\mathrm{F}(3)=\min\{\mathrm{F}(3\text{-}\mathrm{d}_1)+1, \mathrm{F}(3\text{-}\mathrm{d}_2)+1\}=\min\{\mathrm{F}(2)+1, \mathrm{F}(0)+1\}=1$
- $\mathrm{F}(4)=\min\{\mathrm{F}(4\text{-}\mathrm{d}_1)+1, \mathrm{F}(4\text{-}\mathrm{d}_2)+1, \mathrm{F}(4\text{-}\mathrm{d}_3)+1\}=\min\{\mathrm{F}(3)+1, \mathrm{F}(1)+1, \mathrm{F}(0)+1\}=1$

找零问题

实例

假设有3个硬币，它们的面值依次为1，3，4，现需要找零6，最少需要多少硬币？

$$F(n) = \begin{cases} \min\limits_{n \geq d_i}\{F(n-d_i)+1\}, & n > 1, 1 \leq i \leq m \\ 0, & n = 0 \end{cases}$$

自下向上：基于动态规划的计算过程

- $\mathrm{F}(1) = \min\{F(1-\mathrm{d}_1)+1\} = \min\{F(0)+1\} = 1$
- $\mathrm{F}(2) = \min\{\mathrm{F}(2\text{-}\mathrm{d}_1)+1\} = \min\{\mathrm{F}(1)+1\} = 2$
- $\mathrm{F}(3) = \min\{\mathrm{F}(3\text{-}\mathrm{d}_1)+1,\ \mathrm{F}(3\text{-}\mathrm{d}_2)+1\} = \min\{\mathrm{F}(2)+1,\ \mathrm{F}(0)+1\} = 1$
- $\mathrm{F}(4) = \min\{\mathrm{F}(4\text{-}\mathrm{d}_1)+1,\ \mathrm{F}(4\text{-}\mathrm{d}_2)+1,\ \mathrm{F}(4\text{-}\mathrm{d}_3)+1\} = \min\{\mathrm{F}(3)+1,\ \mathrm{F}(1)+1,\ \mathrm{F}(0)+1\} = 1$

找零问题

实例

假设有3个硬币，它们的面值依次为1，3，4，现需要找零6，最少需要多少硬币？

$$F(n) = \begin{cases} \min\limits_{n \geq d_i}\{F(n-d_i)+1\}, & n > 1, 1 \leq i \leq m \\ 0, & n = 0 \end{cases}$$

自下向上：基于动态规划的计算过程

- $\mathrm{F}(1) = \min\{F(1-\mathrm{d}_1)+1\} = \min\{F(0)+1\} = 1$
- $\mathrm{F}(2) = \min\{\mathrm{F}(2\text{-}\mathrm{d}_1)+1\} = \min\{\mathrm{F}(1)+1\} = 2$
- $\mathrm{F}(3) = \min\{\mathrm{F}(3\text{-}\mathrm{d}_1)+1,\ \mathrm{F}(3\text{-}\mathrm{d}_2)+1\} = \min\{\mathrm{F}(2)+1,\ \mathrm{F}(0)+1\} = 1$
- $\mathrm{F}(4) = \min\{\mathrm{F}(4\text{-}\mathrm{d}_1)+1,\ \mathrm{F}(4\text{-}\mathrm{d}_2)+1,\ \mathrm{F}(4\text{-}\mathrm{d}_3)+1\} = \min\{\mathrm{F}(3)+1,\ \mathrm{F}(1)+1,\ \mathrm{F}(0)+1\} = 1$

找零问题

实例

假设有3个硬币，它们的面值依次为1，3，4，现需要找零6，最少需要多少硬币

$$F(n)=\begin{cases}\min\limits_{n\geq d_i}\{F(n-d_i)+1\}, & n>1, 1\leq i\leq m\\ 0, & n=0\end{cases}$$

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–	1	2	1	1		

找零问题

实例

假设有3个硬币，它们的面值依次为1，3，4，现需要找零6，最少需要多少硬币

$$F(n)=\begin{cases}\min\limits_{n\geq d_i}\{F(n-d_i)+1\}, & n>1, 1\leq i\leq m\\ 0, & n=0\end{cases}$$

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–	1	2	1	1		

自下向上：基于动态规划的计算过程(续)

- F(5) = min{F(5-d_1)+1, F(5-d_2)+1, F(5-d_3)+1} = min{F(4)+1, F(2)+1, F(1)+1} = 2
- F(6) = min{F(6-d_1)+1,F(6-d_2)+1,F(6-d_3)+1} = min{F(5)+1,F(3)+1,F(2)+1} = 2

找零问题

实例

假设有3个硬币，它们的面值依次为1，3，4，现需要找零6，最少需要多少硬币

$$F(n) = \begin{cases} \min\limits_{n \geq d_i}\{F(n-d_i)+1\}, & n > 1, 1 \leq i \leq m \\ 0, & n = 0 \end{cases}$$

找零金额n	0	1	2	3	4	5	6
所需硬币数F	–	1	2	1	1		

自下向上：基于动态规划的计算过程(续)

- F(5) = min{F(5-d_1)+1, F(5-d_2)+1, F(5-d_3)+1} = min{F(4)+1, F(2)+1, F(1)+1} = 2
- F(6) = min{F(6-d_1)+1,F(6-d_2)+1,F(6-d_3)+1} = min{F(5)+1,F(3)+1,F(2)+1} = 2

自下而上求解过程图示

$\boxed{F(0)=0}$

找币问题

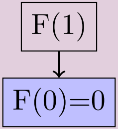
自下而上求解过程图示

F(1)

F(0)=0

找币问题

自下而上求解过程图示



自下而上求解过程图示

$$
\begin{tikzcd}
\boxed{F(1)=1} \arrow[d] \\
\boxed{F(0)=0}
\end{tikzcd}
$$

找币问题

自下而上求解过程图示

F(2)

F(1)=1

↓

F(0)=0

自下而上求解过程图示

$$\begin{array}{c}\boxed{F(2)}\\ \downarrow \\ \boxed{F(1)=1}\\ \downarrow \\ \boxed{F(0)=0}\end{array}$$

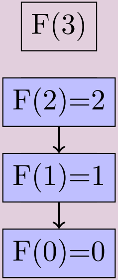
找币问题

自下而上求解过程图示

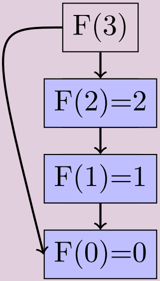
$$\boxed{F(2)=2} \downarrow \boxed{F(1)=1} \downarrow \boxed{F(0)=0}$$

找币问题

自下而上求解过程图示

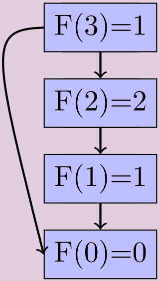


自下而上求解过程图示



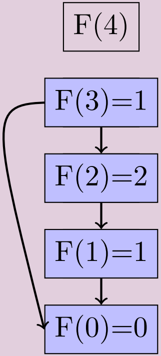
找币问题

自下而上求解过程图示



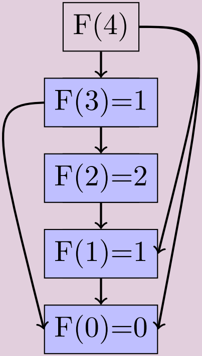
找币问题

自下而上求解过程图示

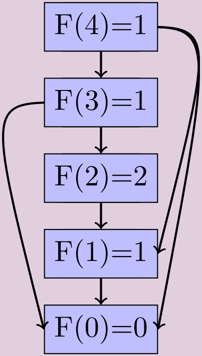


找币问题

自下而上求解过程图示

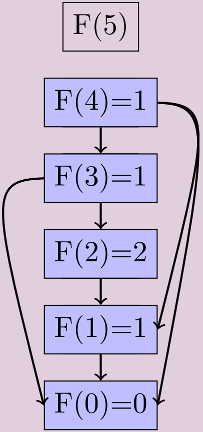


自下而上求解过程图示



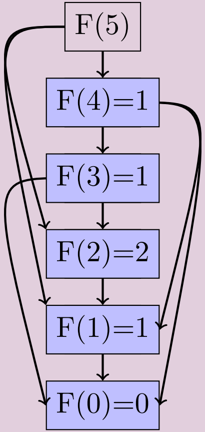
找币问题

自下而上求解过程图示



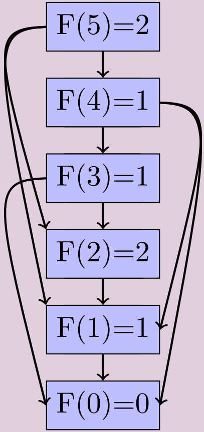
找币问题

自下而上求解过程图示



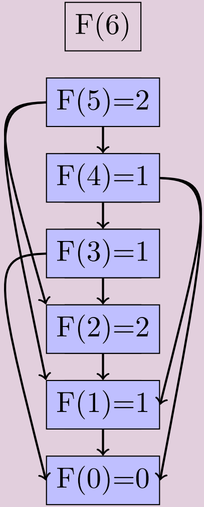
找币问题

自下而上求解过程图示



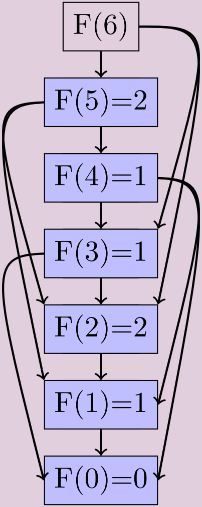
找币问题

自下而上求解过程图示

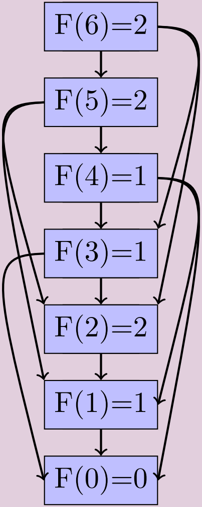


找币问题

自下而上求解过程图示



自下而上求解过程图示



ChangeMaking(C[1···m], n)

//算法基础：$F(n) = \min_{n \geq d_i}\{F(n-d_i)+1\} = \min_{n \geq d_i}\{F(n-d_i)\}+1$

//输入: C[1···m]表示用于找零的m个硬币及其对应的值

//输入：n表示找零金额

//输出：最少的用币数

```
F[0] = 0
for i = 1 to n do
    temp = ∞; j = 1
    while j ≤ m and i ≥ C[j] do
        temp = min(F(i - C[j]), temp)
        j = j + 1
    end while
    F[i] = temp + 1
end for
return F[n]
```

最长递增子序列

最长递增子序列–问题描述

在一个给定的数值序列中，找到一个子序列，使得这个子序列元素的数值依次递增，并且这个子序列的长度尽可能地大。最长递增子序列中的元素在原序列中不一定是连续的。

示例

1. 原始序列：$\{0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15\}$
2. 最长递增子序列：$\{0, 2, 6, 9, 11, 15\}$

✓ 注意：

$\{0, 2, 6, 9, 13, 15\}, \{0, 4, 6, 9, 11, 15\}, \{0, 4, 6, 9, 13, 15\}$ 也是问题的解。

最长递增子序列

求解思路

- 假设$A[0,1,...,n-1]$表示输入序列，其中n表示问题序列的长度；
- L_i表示以$A[i]$结尾的最长递增子序列的长度；
- 求最长递增子序列问题：$max\{L_i\}, 0 \le i \le n-1$;
- 对于$\forall k$，$k < i$，$L_i = max\{L_k | A[k] < A[i]\} + 1$;
- 对于$\forall k$，$k < i$，若$A[k] \ge A[i]$，则$L_i = 1$;
- 初始条件：$L_0 = 1$;

实例

- $A = \{0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15\}$
- $L_0 = 1, L_{15} = ?$

最长递增子序列

求解过程

- $A = \{0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15\}$
- $i = 0; \Rightarrow L_0 = 1$
- $i = 1; k = 0; \Rightarrow L_1 = max\{L_k\} + 1 = 2$
- $i = 2; k = 0; \Rightarrow L_2 = max\{L_k\} + 1 = 2$
- $i = 3; k = 0, 1, 2; \Rightarrow L_3 = max\{L_k\} + 1 = 3$
- $i = 4; k = 0; \Rightarrow L_4 = max\{L_k\} + 1 = 2$
- $i = 5; k = 0, 1, 2, 4; \Rightarrow L_5 = max\{L_k\} + 1 = 3$
- $i = 6; k = 0, 2, 4; \Rightarrow L_6 = max\{L_k\} + 1 = 3$
- $i = 7; k = 0, 2, 4, 6; \Rightarrow L_7 = max\{L_k\} + 1 = 4$

最长递增子序列

求解过程

- $A = \{0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15\}$
- $i = 8; k = 0; \Rightarrow L_8 = max\{L_k\} + 1 = 2$
- $i = 9; k = 0, 1, 2, 4, 6, 8; \Rightarrow L_9 = max\{L_k\} + 1 = 4$
- $i = 10; k = 0, 2, 4, 8; \Rightarrow L_{10} = max\{L_k\} + 1 = 3$
- $i = 11; k = 0, ..., 6, 8, 9, 10; \Rightarrow L_{11} = max\{L_k\} + 1 = 5$
- $i = 12; k = 0, 4, 8; \Rightarrow L_{12} = max\{L_k\} + 1 = 3$
- $i = 13; k = 0, 1, 2, 4, 5, 6, 8, 9, 10, 12; \Rightarrow L_{13} = max\{L_k\} + 1 = 5$
- $i = 14; k = 0, 2, 4, 6, 8, 10, 12; \Rightarrow L_{14} = max\{L_k\} + 1 = 4$
- $i = 15; k = 0, ..., 14; \Rightarrow L_{15} = max\{L_k\} + 1 = 6$

最长递增子序列

求解过程—回溯求解

- $A = \{0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15\}$
- $L_{15} = 6; \Rightarrow \{15\}$
- $L_{11}, L_{13}; \Rightarrow \{13, 15\}, \{11, 15\}$
- $L_{11} : L_9; \Rightarrow 9, 13, 15$
- $L_{11} : L_9 : L_6; \Rightarrow \{6, 9, 13, 15\}$
- $L_{11} : L_9 : L_6 : L_2, L_4; \Rightarrow \{4, 6, 9, 13, 15\}, \{2, 6, 9, 13, 15\}$
- $L_{11} : L_9 : L_6 : L_2, L_4 : L_0; \Rightarrow \{0, 4, 6, 9, 13, 15\}, \{0, 2, 6, 9, 13, 15\}$
- $L_{13} : L_9; \Rightarrow \{9, 11, 15\}$
- $L_{13} : L_9 : L_6 : L_2, L_4 : L_0; \Rightarrow \{0, 4, 6, 9, 11, 15\}, \{0, 2, 6, 9, 11, 15\}$

编辑距离问题

编辑距离问题—问题描述

对于序列S和T，求从S变为T最少需要几步，其中每一步只能进行以下三个操作：
1、删除一个字符；
2、插入一个字符；
3、改变一个字符。
将S变为T的最小操作计数就是两者的编辑距离。

示例

1. $S=ABC$；$T=CBCD$
2. S 需要执行的操作：替换A为C，末尾插入D，距离为2；
3. $S=ABC$；$T=DCB$
4. S 需要执行的操作：替换A为D，删除B，末尾插入B，距离为3；

求解思路

1. $D[i,j]$ 表示$S[0,..,i] \rightarrow T[0,...,j]$的编辑距离。
2. $S[i]=T[j]$ 时，不需要操作，即$D[i,j]=D[i-1,j-1]$；
3. $S[i]\neq T[j]$ 时，可以执行如下的操作：
   1. $S[0,..,i] \rightarrow T[0,...,j-1]$，$T[j]$插入到$S$ 末尾，$D[i][j-1]+1$
   2. 删除$S[i]$，$S[0,..,i-1] \rightarrow T[0,...,j]$，$D[i-1][j]+1$
   3. $S[0,..,i-1] \rightarrow T[0,...,j-1]$，替换$S[i]$ 为$T[j]$，$D[i-1][j-1]+1$
4. $D[i][j]=min(D[i][j-1]+1, D[i-1][j]+1, D[i-1][j-1]+1)$

实例

计算$S(algorithm)$和$T(altruistic)$ 的编辑距离。

编辑距离问题

求解过程

$D[i,j]$ 矩阵

	T	1	2	3	4	5	6	7	8	9	10
S		a	l	t	r	u	i	s	t	i	c
1	a	0	1	2	3	4	5	6	7	8	9
2	l	1	0	1	2	3	4	5	6	7	8
3	g	2	1	1	2	3	4	5	6	7	8
4	o	3	2	2	2	3	4	5	6	7	8
5	r	4	3	3	2	3	4	5	6	7	8
6	i	5	4	4	3	3	3	4	5	6	7
7	t	6	5	4	4	4	4	4	4	5	6
8	h	7	6	5	5	5	5	5	5	5	6
9	m	8	7	6	6	6	6	6	6	6	6

时间复杂度

假如S的长度为n，T的长度为m，计算$D[i,j]$的复杂度为$m*n$。

背包问题

问题回顾

- 求能够放入给定承重量的背包的最大价值的物品集合;
- 穷举查找法可以得到最优解，但是计算复杂度较高(2^n);
- 贪心算法计算较快，但是只能得到近似解;

动态规划求解

- $F(i,j)$为前i 个物品放进承重量为j的背包问题的最优解;
- 前i个物品构成的子集分为两类：含第i个物品和不含第i个物品;
- $F(i,j) = \begin{cases} \max\{F(i-1,j), v_i + F(i-1,j-w_i)\}, j-w_i \geq 0 \\ F(i-1,j), \ j-w_i \leq 0 \end{cases}$
- $F(0,j)=0, j\geq 0; F(i,0)=0, i\geq 0;$
- 一共需要计算nW 个F 值，复杂度为$\Theta(n)$;

背包问题

实例——递推关系

$$F\left(i,j\right)=\begin{cases}\max\left\{F\left(i-1,j\right),v_i+F\left(i-1,j-w_i\right)\right\},j-w_i\geq 0\\ F\left(i-1,j\right),\ j-w_i\leq 0\end{cases}$$

$$F(0,j)=0,j\geq 0;F(i,0)=0,i\geq 0$$

实例

物品	重量（w）	价值（v）
1	2	12
2	1	10
3	3	20
4	2	15
	背包承重量W=5	

背包问题

实例

$F(i,j)$	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

回溯求装入背包的物品

- $F(4,5) > F(3,5)$: 物品4装入背包;
- $F(3,5-2) = F(2,5-2)$: 物品3 不是最优子集一部分;
- $F(2,3) > F(1,3)$: 物品2 装入背包;
- $F(1,3-1) > F(0,3-1)$: 物品1 装入背包;

动态规划优化

记忆化

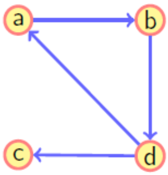
- 自顶向下的递归调用求解会多次求解子问题；
- 自下向上的动态规划过程会求解一些不必要的子问题；
- 结合两种方法，只求解一次必要的子问题（记忆化）；

传递闭包问题

传递闭包问题—定义

一个n个顶点的有向图的传递闭包可以定义为一个n阶矩阵$T = t_{ij}$，如果从第i个顶点到第j个顶点之间存在一条有效的有向路径，则$t_{ij} = 1$，否则为0。

示例



邻接矩阵

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

传递闭包

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

传递闭包问题

Warshall算法——算法要点

- 顶点v_i到v_j路径上顶点的编号都不大于k，则$R^{(k)}(i,j)=1$;
- 路径$a(1)-b(2)-d(3)$: $R^{(2)}(1,3)=1, R^{(3)}(1,3)=1$;
- $R^{(k)}(i,j)=1$ 时，v_i到v_j路径有两种情况：
  v_i-{编号$<k$的顶点}-v_j
  v_i-{编号$<k$的顶点}-v_k{编号<k的顶点}-v_j（假定v_k只出现1次）
- 即$R^{(k-1)}(i,j)=1$或$R^{(k-1)}(i,k)=1, R^{(k-1)}(k,j)=1$

传递闭包问题

Warshall算法——算法要点

- 顶点v_i到v_j路径上顶点的编号都不大于k，则$R^{(k)}(i,j)=1$;
- 路径$a(1)-b(2)-d(3)$: $R^{(2)}(1,3)=1, R^{(3)}(1,3)=1$;
- $R^{(k)}(i,j)=1$ 时，v_i到v_j路径有两种情况：
  v_i-{编号$<k$的顶点}-v_j
  v_i-{编号$<k$的顶点}-v_k{编号¡k的顶点}-v_j（假定v_k只出现1次）
- 即$R^{(k-1)}(i,j)=1$或$R^{(k-1)}(i,k)=1, R^{(k-1)}(k,j)=1$

递推关系

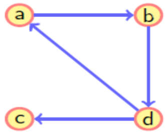
$R^{(k-1)}(i,j)=1$
或$\{R^{(k-1)}(i,k)=1, R^{(k-1)}(k,j)=1\} \Rightarrow R^{(k)}(i,j)=1$

传递闭包问题问题

递推关系

$$R^{(k-1)}(i,j)=1$$
$$\text{或}\{R^{(k-1)}(i,k)=1, R^{(k-1)}(k,j)=1\} \Rightarrow R^{(k)}(i,j)=1$$

实例



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

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

$$\mathbf{R}^{(2)} \quad \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & \mathbf{1} \end{bmatrix} \end{array}$$

$$\mathbf{R}^{(3)} \quad \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$\mathbf{R}^{(4)} \quad \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} \mathbf{1} & 1 & \mathbf{1} & 1 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

完全最短路径问题

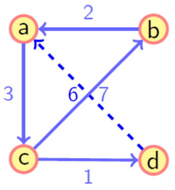
单源最短路径问题—回顾

给定一个起点，计算起点到所有其他顶点之间的最短路径。

完全最短路径问题

找到从每个顶点到其他所有顶点之间的最短路径。

示例



邻接矩阵**W**

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

距离矩阵**D**

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

Floyd算法

- $D(k)(i,j)$ 表示第i个顶点到第j个顶点的最短路径长度，并且路径的中间顶点编号不大于k;
- 第i个顶点到第j个顶点的最短路径有两种情况:
  v_i-{编号$<k$的顶点}-v_j
  v_i-{编号$<k$的顶点}-v_k{编号$<k$的顶点}-v_j
- 第i个顶点到第j个顶点的最短路径为两种情况下的最小值;

完全最短路径问题

Floyd算法

- $D(k)(i,j)$ 表示第i个顶点到第j个顶点的最短路径长度，并且路径的中间顶点编号不大于k;
- 第i个顶点到第j个顶点的最短路径有两种情况：
  v_i-{编号$<k$的顶点}-v_j
  v_i-{编号$<k$的顶点}-v_k{编号$<k$的顶点}-v_j
- 第i个顶点到第j个顶点的最短路径为两种情况下的最小值;

递推关系

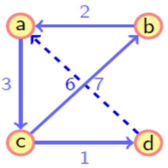
$$D^{(k)}(i,j) = min\{D^{(k-1)}(i,k) + D^{(k-1)}(k,j), D^{(k-1)}(i,j)\}$$

完全最短路径问题

递推关系

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

实例



$$\mathbf{D}^{(0)}\quad \begin{array}{c|c} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & - & 3 & - \\ 2 & 0 & - & - \\ - & 7 & 0 & 1 \\ 6 & - & - & 0 \end{bmatrix} \end{array}$$

$$\mathbf{D}^{(1)}\quad \begin{array}{c|c} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & - & 3 & - \\ 2 & 0 & \mathbf{5} & - \\ - & 7 & 0 & 1 \\ 6 & - & \mathbf{9} & 0 \end{bmatrix} \end{array}$$

$$\mathbf{D}^{(2)}\quad \begin{array}{c|c} & \begin{matrix} a & b & c & d \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \end{matrix} & \begin{bmatrix} 0 & - & 3 & - \\ 2 & 0 & 5 & - \\ \mathbf{9} & 7 & 0 & 1 \\ 6 & - & 9 & 0 \end{bmatrix} \end{array}$$

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

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

矩阵链乘计算问题

矩阵链乘计算问题——问题描述

- 设$A_1, A_2, ..., A_n$ 为n 个矩阵的序列，其中A_i 为$P_{i-1} * P_i$ 阶矩阵，可用向量$P =< P_0, P_1, P_2, ..., P_n >$ 表示矩阵链的输入规模；
- 计算$A_1, A_2, ..., A_n$ 的乘积，不同的计算顺序，计算量有所不同，选择计算量最少的次序计算矩阵乘积；

示例

- 若A_1, A_2分别是$i * j$, $j * k$ 的矩阵，计算$A_1 A_2$ 需要$i * j * k$次乘法；
- 如有$P =< 10, 100, 5, 50 >$，计算$A_1 A_2 A_3$ 的乘法次数分别为：
  - $(A_1 A_2)A_3$：$10 * 100 * 5 + 10 * 5 * 50 = 7500$;
  - $A_1(A_2 A_3)$：$10 * 100 * 50 + 100 * 5 * 50 = 75000$;

矩阵链乘计算问题

动态规划求解–递推关系

- $A_iA_{i+1}...A_j=(A_iA_{i+1}...A_k)(A_{k+1}A_{k+2}...A_j)$;
- 假设$m[i,j]$为计算乘积$A_iA_{i+1}...A_j$ 所用的最少运算次数，有：
  - $i=j, m[i,j]=0, i=j$
  - $i<j, m[i,j]=min_{i\le k<j}\{m[i,k]+m[k+1,j]+P_{i-1}P_kP_j\}$

实例：$P=<30,35,15,5,10,20>$

- $A_1:30*35, A_2:35*15, A_3:15*5, A_4:5*10, A_5:10*20$
- $m[1,2]=30*35*15=15750, m[2,3]=35*15*5=2625$
- $m[3,4]=15*5*10=750, m[4,5]=5*10*20=1000$
- $m[1,3]=min\{m[1,2]+30*15*5, m[2,3]+30*35*5\}=7875$
- $m[2,4]=min\{m[2,3]+35*5*10, m[3,4]+35*15*10\}=4375$

动态规划求解

实例：$P =< 30, 35, 15, 5, 10, 20 >$

- $m[3,5] = min\{m[3,4]+15*10*20, m[4,5]+15*5*20\} = 2500;$
- $m[1,4] = min\{m[2,4]+30*35*10, m[1,2]+m[3,4]+30*15*10, m[1,3]+30*5*10\} = 9375$
- $m[2,5] = min\{m[3,5]+35*15*20, m[2,3]+m[4,5]+35*5*20, m[2,4]+35*10*20\} = 7125$
- $m[1,5] = min\{m[2,5]+30*35*20, m[1,2]+m[3,5]+30*15*20, m[1,3]+m[4,5]+30*5*20, m[1,4]+30*10*20\} = 11875$

回溯求解

最佳计算次序：$A_1A_2A_3A_4A_5 = (A_1(A_2A_3))(A_4A_5)$

动态规划算法步骤小结

动态规划算法步骤小结

1. 动态规划方法是对一种具有交叠子问题进行求解的技术，与递归调用不同，动态规划对较小的子问题只求解一次；
2. 动态规划方法要求最优问题满足的法则：该问题的任何实例的最优解是由该实例的子实例的最优解组成。

作业

1. 8.1:1 Page224
2. 8.1:5 Page224
3. 8.1:6 Page225
4. 8.1:11 Page225