



提要

第四章 动态规划技术

船吉洲
计算机科学与技术学院

- 4.1 Elements of Dynamic Programming
- 4.2 Longest Common Subsequence
- 4.3 Matrix-chain multiplication
- 4.4 0/1 Knapsack Problem
- 4.5 The Optimal binary search trees
- 4.6 凸多边形的三角剖分



Why?

- 分治技术的问题
 - 子问题是相互独立的
 - 若果子问题是相互独立的，分治方法将重叠计算公共子问题，效率很低
- 例如，计算斐波那契数列的第 n 项
 - $F(0)=F(1)=1$
 - $F(n)=F(n-1)+F(n-2)$

3.1 动态规划技术的基本要素

Why?

What?

How?



Why?



Why?

• 分治技术的问题

- 子问题是相互独立的
- 若果子问题是相互独立的，分治方法将重叠计算公共子问题，效率很低

• 分治算法

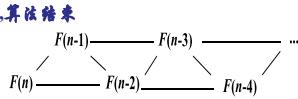
算法 $F(n)$

输入: 非负整数 n

输出: 斐波那契数列第 n 项

1. If $n=0$ 或 1 Then 输出 1, 算法结束
2. $f_1 \leftarrow F(n-1)$;
3. $f_2 \leftarrow F(n-2)$;
4. 输出 f_1+f_2 ;

时间复杂性
 $T(1)=T(0)=1$
 $T(n)=T(n-1)+T(n-2)$
 $T(n)$ 不是多项式有界的



提高效率的方法

- 从规模最小的子问题开始计算
- 用恰当数据结构存储子问题的解，供以后查用
- 确保每个子问题只求解一次

• 算法

算法 $F(n)$

输入: 非负整数 n

输出: 斐波那契数列第 n 项

1. $A[0] \leftarrow 1$; $A[1] \leftarrow 1$;
2. For $i=2$ To n
3. $A[i] \leftarrow A[i-1]+A[i-2]$;
4. 输出 $A[n]$;

时间复杂性

$$T(n)=\Theta(n)=\Theta(2^{\log n})$$

存在 $\Theta(\log n)$ 的分治算法

$$\begin{cases} F(n) \\ F(n-1) \end{cases} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \begin{cases} F(n-1) \\ F(n-2) \end{cases}$$

$$\begin{cases} F(n) \\ F(n-1) \end{cases} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$



Why?

- 动态规划的问题
 - 子问题是相互独立的
 - 如果子问题是相互独立的，动态方法将重複计算公共子问题，效率很低
- 优化问题
 - 给定一组约束条件和一个代价函数，在解空间中搜索具有最小或最大代价的优化解
 - 很多优化问题可分解为多个子问题，子问题相互关联，子问题的解被重复使用



What?

- 动态规划算法特点
 - 把原始问题划分成一系列子问题
 - 在解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接索取，不重复计算，节省计算时间
 - 自底向上地计算
- 适用范围
 - 一类优化问题：可分解为多个相关子问题，子问题的解被重复使用



How?

- 使用 Dynamic Programming 的条件
 - Optimal substructure (优化子结构)
 - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。
 - 缩小子问题集合，只需那些优化问题中包含的子问题，减低实现复杂性
 - 优化子结构使得我们能自下而上地完成求解过程
 - Subtleties (重叠子问题)
 - 在问题的求解过程中，很多子问题的解将被多次使用



• 动态规划算法的设计步骤

- 分析优化解的结构
- 追踪地定义最优解的代价
- 自底向上地计算优化解的代价保存之，并获取构造最优解的信息
- 根据构造最优解的信息构造优化解



4.2 Longest Common Subsequence

- 问题的定义
- 最长公共子序列 (LCS) 结构分析
- 建立求解 LCS 问题的递归方程
- 自底向上 LCS 问题的计算
- 构造优化解



问题的定义

- 子序列
 - $X = (A, B, C, B, D, B)$
 - $Z = (B, C, D, B)$ 是 X 的子序列
 - $W = (B, D, A)$ 不是 X 的子序列
- 公共子序列
 - Z 是序列 X 与 Y 的公共子序列如果 Z 是 X 的子序列也是 Y 的子序列。



最长公共子序列 (LCS) 问题

输入: $X = (x_1, x_2, \dots, x_n)$, $Y = (y_1, y_2, \dots, y_m)$

输出: $Z = X$ 与 Y 的最长公共子序列



HIT
CS&E

蛮力法

$X=ABCDBDB$
 $Y=BCDB$
 $Z_1=ABCDBDB$ 不是 Y 的子序列
 $Z_2=ACBDB$ 不是 Y 的子序列
 \dots
 $Z_5=ABCBD$ 不是 Y 的子序列
 $Z_6=BCBDB$ 不是 Y 的子序列
 \dots
 $Z_{11}=BCDB$ 是 Y 的子序列

基本操作: 字符比较

能够通过基本操作把原始问题处理成子问题的求解?



$X=ABCBD**B**$
 $Y=BCD**B**$

$LCS(ABCBD**B**, BCD**B**) = LCS(ABCBD, BCD) + <**B**>$

$X=ABCBD**B**$
 $Y=BCDA$ $LCS(X, Y)$ 的最末元素不能同时取 **A**, **B**

如果 $LCS(X, Y)$ 的最末元素不等于 **B**
 $LCS(ABCBD**B**, BCDA) = LCS(ABCBD, BCDA)$

如果 $LCS(X, Y)$ 的最末元素不等于 **A**
 $LCS(ABCBD**B**, BCDA) = LCS(ABCBD**B**, BCD)$

子问题



最长公共子序列结构分析

• 第 i 前缀

- 设 $X=(x_1, x_2, \dots, x_n)$ 是一个序列, X 的第 i 前缀 X_i 是一个序列, 定义为 $X_i=(x_1, \dots, x_i)$

例. $X=(A, B, D, C, A)$, $X_1=(A)$, $X_2=(A, B)$, $X_3=(A, B, D)$



• 优化子结构

定理1 (优化子结构) 设 $X=(x_1, \dots, x_m)$, $Y=(y_1, \dots, y_n)$ 是两个序列, $Z=(z_1, \dots, z_k)$ 是 X 与 Y 的 LCS, 我们有:

(1) 如果 $x_m=y_n$, 则 $z_k=x_m=y_n$, Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的 LCS, 即, $LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>$.

(2) 如果 $x_m \neq y_n$, 且 $z_k \neq x_m$, 则 Z 是 X_{m-1} 和 Y 的 LCS, 即

$LCS_{XY} = LCS_{X_{m-1}Y}$

(3) 如果 $x_m \neq y_n$, 且 $z_k \neq y_n$, 则 Z 是 X 与 Y_{n-1} 的 LCS, 即

$LCS_{XY} = LCS_{XY_{n-1}}$

证明:

(1). $X=<x_1, \dots, x_{m-1}, x_m>$, $Y=<y_1, \dots, y_{n-1}, y_n>$, 则

$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>$.

设 $z_k \neq x_m$, 则可加 $x_m=y_n$ 到 Z , 得到一个长于 $k+1$ 的 X 与 Y 的公共序列, 与 Z 是 X 和 Y 的 LCS 矛盾。于是 $z_k=x_m=y_n$ 。

现在证明 Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的 LCS。显然 Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的公共序列。我们需要证明 Z_{k-1} 是 LCS。

设不然, 则存在 X_{m-1} 与 Y_{n-1} 的公共子序列 W , W 的长大于 $k-1$ 。增加 $x_m=y_n$ 到 W , 我们得到一个长于 k 的 X 与 Y 的公共序列, 与 Z 是 LCS 矛盾。于是, Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的 LCS。

(2) $X = \langle x_1, \dots, x_{m-1}, x_m \rangle$, $Y = \langle y_1, \dots, y_{n-1}, y_n \rangle$,

$x_m \neq y_n$, $z_k \neq x_m$, 则 $LCS_{XY} = LCS_{X_{m-1}Y}$

由于 $z_k \neq x_m$, Z 是 X_{m-1} 与 Y 的公共子序列。我们来
证 Z 是 X_{m-1} 与 Y 的 LCS 。设 X_{m-1} 与 Y 有一个公共子序列
 W , W 的长大于 k , 则 W 也是 X 与 Y 的公共子序列, 与
 Z 是 LCS 矛盾。

(3) 同(2)可证。

X 和 Y 的 LCS 的优化解结构

$$\begin{aligned} LCS_{XY} &= LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle && \text{if } x_m = y_n \\ LCS_{XY} &= LCS_{X_{m-1}Y} && \text{if } x_m \neq y_n, z_k \neq x_m \\ LCS_{XY} &= LCS_{XY_{n-1}} && \text{if } x_m \neq y_n, z_k = y_n \end{aligned}$$

分治算法

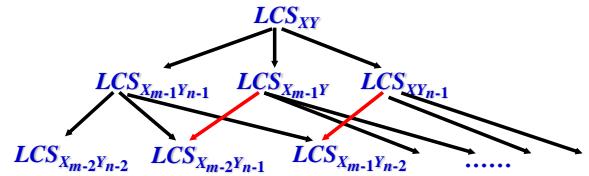
• 算法 SimpleLCS(X, Y)

输入: $X = (x_1, x_2, \dots, x_m)$, $Y = (y_1, y_2, \dots, y_m)$

输出: X, Y 的最长公共子序列

1. If $m=0$ 或 $n=0$ Then 输出空串, 算法结束;
2. If $x_n=y_m$ Then
 3. 输出 SimpleLCS(X_{n-1}, Y_{m-1}) + $\langle x_n \rangle$;
 4. Else
 5. $Z_1 \leftarrow \text{SimpleLCS}(X_{n-1}, Y);$
 6. $Z_2 \leftarrow \text{SimpleLCS}(X, Y_{m-1});$
 7. 输出 Z_1, Z_2 中较长者;

• 子问题重叠性



LCS 问题具有子问题重叠性

建立 LCS 长度的递归方程

• $C[i, j] = X_i$ 与 Y_j 的 LCS 的长度

• LCS 长度的递归方程

$$\begin{aligned} C[i, j] &= 0 && \text{if } i=0 \text{ 或 } j=0 \\ C[i, j] &= C[i-1, j-1] + 1 && \text{if } i, j > 0 \text{ and } x_i = y_j \\ C[i, j] &= \text{Max}(C[i, j-1], C[i-1, j]) && \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{aligned}$$

自底向上计算 LCS 的长度

• 基本思想

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

• 计算过程

C[0,0]	C[0,1]	C[0,2]	C[0,3]	C[0,4]
C[1,0]	C[1,1]	C[1,2]	C[1,3]	C[1,4]
C[2,0]	C[2,1]	C[2,2]	C[2,3]	C[2,4]
C[3,0]	C[3,1]	C[3,2]	C[3,3]	C[3,4]

• 计算LCS长度的算法

- 数据结构

C[0:m,0:n]: $C[i,j]$ 是 X_i 与 Y_j 的 LCS 的长度

B[1:m,1:n]: $B[i,j]$ 是指针，指向计算 $C[i,j]$

时所选择的子问题的优化解

所对应的 C 表的表项

```
LCS-length(X, Y)
m←length(X); n←length(Y);
For i←1 To m Do C[i,0]←0;
For j←1 To n Do C[0,j]←0;
For i←1 To m Do
    For j←1 To n Do
        If  $x_i = y_j$ 
            Then  $C[i,j] \leftarrow C[i-1,j-1] + 1$ ;  $B[i,j] \leftarrow "↖"$ ;
        Else If  $C[i-1,j] \geq C[i,j-1]$  Then
             $C[i,j] \leftarrow C[i-1,j]$ ;  $B[i,j] \leftarrow "↑"$ ;
        Else  $C[i,j] \leftarrow C[i,j-1]$ ;  $B[i,j] \leftarrow "←"$ ;
Return C and B.
```

• 基本思想

- 从 $B[m, n]$ 开始按指针搜索

- 若 $B[i, j] = "↖"$, 则 $x_i = y_j$ 是 LCS 的一个元素

- 由此找到的 “LCS” 是 X 与 Y 的 LCS 的 Inverse

```
Print-LCS(B, X, i, j)
IF i=0 or j=0 THEN Return;
IF B[i,j] = "↖"
THEN Print-LCS(B, X, i-1, j-1); Print  $x_i$ ;
ELSE If B[i,j] = "↑"
      THEN Print-LCS(B, X, i-1, j);
      ELSE Print-LCS(B, X, i, j-1).
```

4.3 矩阵链乘法

Print-LCS(B, X, length(X), length(Y))

可打印出 X 与 Y 的 LCS。



问题的定义

- 输入: $\langle A_1, A_2, \dots, A_n \rangle$, A_i 是矩阵
- 输出: 计算 $A_1 \times A_2 \times \dots \times A_n$ 的最小代价方法

$$A_{p \times q} \times B_{q \times r} = C_{p \times r}$$

$$c_{ij} = \sum_{k=1}^q a_{ik} b_{kj} \quad i=1, \dots, p, j=1, \dots, r$$

矩阵乘法的代价/复杂性: 乘法的次数
若 A 是 $p \times q$ 矩阵, B 是 $q \times r$ 矩阵, 则 $A \times B$
的代价是 $O(pqr)$



Motivation

矩阵链乘法的实现

- 矩阵乘法满足结合律: $A(BC) = (AB)C$
- 计算一个矩阵链的乘法可有多种方法:
例如, $(A_1 \times A_2 \times A_3 \times A_4)$
 $= (A_1 \times (A_2 \times (A_3 \times A_4)))$
 $= ((A_1 \times A_2) \times (A_3 \times A_4))$
 \dots
 $= ((A_1 \times A_2) \times A_3) \times A_4)$



- 矩阵链乘法的代价与计算顺序的关系
 - 设 $A_1 = 10 \times 100$ 矩阵, $A_2 = 100 \times 5$ 矩阵, $A_3 = 5 \times 50$ 矩阵
 $T((A_1 \times A_2) \times A_3) = 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$
 $T(A_1 \times (A_2 \times A_3)) = 100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$

结论: 不同计算顺序有不同的代价



矩阵链乘法优化问题的解空间

- 设 $p(n)$ = 计算 n 个矩阵乘积的方法数
- $p(n)$ 的递归方程

如此之大的解空间是无法用枚举方法求出最优解的!

$$p(n) = \sum_{k=1}^n p(k)p(n-k) \quad \text{if } n > 1$$

$$p(n) = C(n-1) = \text{Catalan 数} = \frac{1}{n} \binom{2(n-1)}{n-1} = \Omega(4^n/n^{3/2})$$



下边开始设计求解矩阵链乘法问题的 Dynamic Programming 算法

- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价保存之,
并获取构造最优解的信息
- 根据构造最优解的信息构造优化解



分析优化解的结构

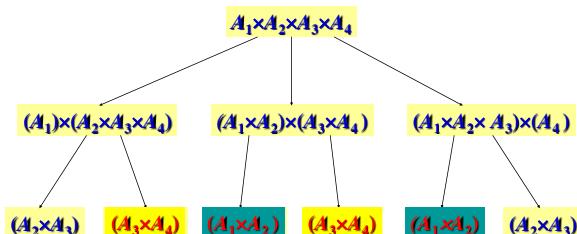
两个记号

- $A_{i:j} = A_i \times A_{i+1} \times \dots \times A_j$
- $\text{cost}(A_{i:j})$ = 计算 $A_{i:j}$ 的代价

优化解的性质

- 具有优化子结构:**
问题的优化解包括子问题优化解
即子问题 $A_{1:k}$ 的解必须是 $A_{1:k}$ 的优化解, 对应于子问题 $A_{k+1:n}$ 的解必须是 $A_{k+1:n}$ 的优化解

• 子问题重叠性



具有子问题重叠性



递归地定义最优解的代价

• 假设

- $m[i, j]$ = 计算 $A_{i:j}$ 的最小乘法数
- $m[1, n]$ = 计算 $A_{1:n}$ 的最小乘法数
- $A_1 \dots A_k A_{k+1} \dots A_n$ 是优化解 (k 实际上是不可预知)

• 代价方程

$m[i, i] = \text{计算 } A_{i:i} \text{ 的最小乘法数} = 0$

$m[i, j] = m[i, k] + m[k+1, j] + p_{i:k} p_k p_j$
 其中, $p_{i:k} p_k p_j$ 是计算 $A_{i:k} \times A_{k+1:j}$ 所需乘法数,
 $A_{i:k}$ 和 $A_{k+1:j}$ 分别是 $p_{i:k} \times p_k$ 和 $p_k \times p_j$ 矩阵.



自底向上计算优化解的代价

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i:k} p_k p_j \}$$

$m[1,1]$	$m[1,2]$	$m[1,3]$	$m[1,4]$	$m[1,5]$
$m[2,2]$	$m[2,3]$	$m[2,4]$	$m[2,5]$	
$m[3,3]$	$m[3,4]$	$m[3,5]$		
$m[4,4]$	$m[4,5]$			
			$m[5,5]$	

$$m[2,4] = \min \left\{ \begin{array}{l} m[2,2]+m[3,4] \\ m[2,3]+m[4,4] \end{array} \right\}$$

考虑到所有的 k , 最优解的代价方程为

$$\begin{aligned} m[i, j] &= 0 && \text{if } i=j \\ m[i, j] &= \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i:k} p_k p_j \} && \text{if } i < j \end{aligned}$$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i:k} p_k p_j \}$$

$m[1,1]$	$m[1,2]$	$m[1,3]$	$m[1,4]$	$m[1,5]$
$m[2,2]$	$m[2,3]$	$m[2,4]$	$m[2,5]$	
$m[3,3]$	$m[3,4]$	$m[3,5]$		
$m[4,4]$	$m[4,5]$			

Matrix-Chain-Order(p)

$n = \text{length}(p)-1;$

FOR $i=1$ TO n DO

$m[i, i]=0;$

FOR $l=2$ TO n DO /* 计算第 l 对角线 */

FOR $i=1$ TO $n-l+1$ DO

$j=i+l-1;$

$m[i, j]=\infty;$

FOR $k \leftarrow i$ To $j-1$ DO /* 计算 $m[i,j]$ */

$q=m[i, k]+m[k+1, j]+p_{i:k} p_k p_j$

IF $q < m[i, j]$ THEN $m[i, j]=q;$

Return m .

HIT
CS&E

获取构造最优解的信息

```

Matrix-Chain-Order( $p$ )
 $n = \text{length}(p)-1;$ 
FOR  $i=1$  TO  $n$  DO
     $m[i, i] = 0;$ 
FOR  $i=2$  TO  $n$  DO
    FOR  $i=1$  TO  $n-i+1$  DO
         $j=i+l-1;$ 
         $m[i, j] = \infty;$ 
        FOR  $k \leftarrow i$  To  $j-1$  DO
             $q = m[i, k] + m[k+1, j] + p_{i,1}p_kp_j$ 
            IF  $q < m[i, j]$  THEN  $m[i, j] = q$ ,  $s[i, j] = k$ ;
Return  $m$  and  $s$ .

```

$S[i, j]$ 记录 $A_i A_{i+1} \dots A_j$ 的最优划分处在 A_i 与 A_{i+1} 之间

HIT
CS&E

构造最优解

Print-Optimal-Parens(s, i, j)IF $j=i$ THEN Print “A” i ;

ELSE Print “(”

Print-Optimal-Parens($s, i, s[i, j]$)Print-Optimal-Parens($s, s[i, j]+1, j$)

Print “)”

$S[i, j]$ 记录 $A_i \dots A_j$ 的最优划分
子处;
 $S[i, S[i, j]]$ 记录 $A_i \dots A_{s[i, j]}$ 的最
优划分子处;
 $S[S[i, j]+1, j]$ 记录 $A_{s[i, j]+1} \dots A_j$
的最优划分子处

调用 Print-Optimal-Parens($s, 1, n$)即可输出 $A_{1 \dots n}$ 的优化计算顺序HIT
CS&E

算法复杂性

- 时间复杂性

- 计算代价的时间

- (l, i, k) 三重循环，每层至多 $n-1$ 步

- $O(n^3)$

- 构造最优解的时间: $O(n)$

- 总时间复杂性为: $O(n^3)$

- 空间复杂性

- 使用数组 m 和 S

- 需要空间 $O(n^2)$

HIT
CS&E

4.4 0/1 背包问题

HIT
CS&E

问题的定义

给定 n 种物品和一个背包，物品 i 的重量是 w_i ，价值 v_i ，背包承重量为 C ，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？

对于每种物品只能选择完全装入或不装入，一个物品至多装入一次。

HIT
CS&E

• 输入: $C > 0, w_i > 0, v_i > 0, 1 \leq i \leq n$

• 输出: $(x_1, x_2, \dots, x_n), x_i \in \{0, 1\}$, 满足

$$\sum_{1 \leq i \leq n} w_i x_i \leq C, \quad \sum_{1 \leq i \leq n} v_i x_i \text{ 最大}$$

等价的整数规划问题

$$\max \sum_{1 \leq i \leq n} v_i x_i$$

$$\sum_{1 \leq i \leq n} w_i x_i \leq C$$

$$x_i \in \{0, 1\}, \quad 1 \leq i \leq n$$



优化解结构的分析

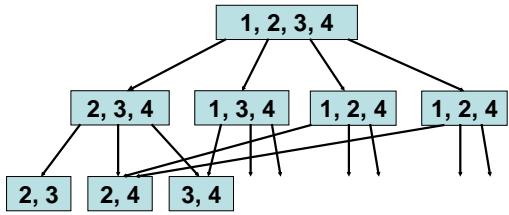
定理(优化子结构) 如果 (y_1, y_2, \dots, y_n) 是0-1背包问题的优化解，则 (y_2, \dots, y_n) 是此下子问题的优化解：

$$\begin{aligned} &\max \sum_{2 \leq i \leq n} v_i x_i \\ &\sum_{2 \leq i \leq n} w_i x_i \leq C - w_1 y_1 \\ &x_i \in \{0, 1\}, \quad 2 \leq i \leq n \end{aligned}$$

证明：如果 (y_2, \dots, y_n) 不是子问题优化解，则存在 (z_2, \dots, z_n) 是子问题更优的解。于是， (y_1, z_2, \dots, z_n) 是原问题比 (y_1, y_2, \dots, y_n) 更优解，矛盾。



子问题重叠性



建立优化解代价的递归方程

• $m(i, j)$:

背包容量为 j ，可选物品为 x_1, x_{i+1}, \dots, x_n 时，问题的最优解的代价是 $m(i, j)$ 。

• 形式地

$$\begin{aligned} \text{问题} \quad &\max \sum_{i \leq k \leq n} v_k x_k \\ &\sum_{i \leq k \leq n} w_k x_k \leq j \\ &x_k \in \{0, 1\}, \quad i \leq k \leq n \end{aligned}$$

的最优解代价为 $m(i, j)$ 。



• 递归方程

$$\begin{aligned} m(i, j) &= m(i+1, j) & 0 \leq j < w_i \\ m(i, j) &= \max\{m(i+1, j), m(i+1, j-w_i)+v_i\} & j \geq w_i \end{aligned}$$

$$\begin{aligned} m(n, j) &= 0 & 0 \leq j < w_n \\ m(n, j) &= v_n & j \geq w_n \end{aligned}$$



自底向上计算优化解的代价

$$\begin{aligned} m(i, j) &= m(i+1, j), \quad 0 \leq j < w_i \\ m(i, j) &= \max\{m(i+1, j), m(i+1, j-w_i)+v_i\}, \quad j \geq w_i \\ m(n, j) &= 0, \quad 0 \leq j < w_n \\ m(n, j) &= v_n, \quad j \geq w_n \end{aligned}$$

令 $w_i = \text{整数}$, $n=4$

$$\begin{array}{c} m(1, C) \\ m(2, C-w_1) \cdots m(2, C) \\ m(3, C-w_1-w_2) \cdots m(3, C-w_1) \cdots m(3, C) \\ \cdots m(4, C-w_1-w_2-w_3) \cdots m(4, C-w_1-w_2) \cdots m(4, C-w_1) \cdots m(4, C) \end{array}$$

$$\begin{array}{c} m(i, j) = m(i+1, j), \quad 0 \leq j < w_i \\ m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\}, \quad j \geq w_i \\ m(n, j) = 0, \quad 0 \leq j < w_n \\ m(n, j) = v_n, \quad j \geq w_n \\ \hline m(i, j) \\ m(i+1, j-w_i) \end{array}$$

令 $w_i = \text{整数}$, $n=4$

$$\begin{array}{ccccccccc} m(1, C) & & & & & & & & \\ \hline m(2, 0) & \cdots & m(2, w_2-1) & m(2, w_2) & \cdots & m(2, C-1) & m(2, C) & & \\ \hline m(3, 0) & \cdots & m(3, w_3-1) & m(3, w_3) & \cdots & m(3, C-1) & m(3, C) & & \\ \hline m(4, 0) & \cdots & m(4, w_4-1) & m(4, w_4) & \cdots & m(4, C-1) & m(4, C) & & \end{array}$$

• 算法

```

 $m(i, j) = m(i+1, j), 0 \leq j < w_i$ 
 $m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\}, j \geq w_i$ 
 $m(n, j) = 0, 0 \leq j < w_n$ 
 $m(n, j) = v_n, j \geq w_n$ 

For  $j=0$  To  $\min(w_n-1, C)$  Do
     $m[n, j] = 0;$ 
For  $j=w_n$  To  $C$  Do
     $m[n, j] = v_n;$ 
For  $i=n-1$  To  $2$  Do
    For  $j=0$  To  $\min(w_i-1, C)$  Do
         $m[i, j] = m[i+1, j];$ 
    For  $j=w_i$  To  $C$  Do
         $m[i, j] = \max\{m[i+1, j], m[i+1, j-w_i]+v_i\};$ 
If  $C < w_1$ 
Then  $m[1, C] = m[2, C];$ 
Else  $m[1, C] = \max\{m[2, C], m[2, C-w_1]+v_1\};$ 

```



构造优化解

1. $m(1, C)$ 是最优解代价值，相应解计算如下：

If $m(1, C) = m(2, C)$

Then $x_1 = 0;$

Else $x_1 = 1;$

2. 如果 $x_1=0$, 由 $m(2, C)$ 继续构造最优解；

3. 如果 $x_1=1$, 由 $m(2, C-w_1)$ 继续构造最优解。



算法复杂性



• 时间复杂性

- 计算代价的时间

$O(Cn)$

- 构造最优解的时间: $O(Cn)$

- 总时间复杂性为: $O(n)$

• 空间复杂性

- 使用数组 m

- 需要空间 $O(Cn)$

4.5 最优二叉搜索树



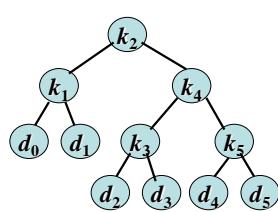
问题的定义



• 二叉搜索树 T

- 结点

- $K = \{k_1, k_2, \dots, k_n\}$
- $D = \{d_0, d_1, \dots, d_n\}$
- d_i 对应区间 (k_i, k_{i+1})
- d_0 对应区间 $(-\infty, k_1)$
- d_n 对应区间 $(k_n, +\infty)$

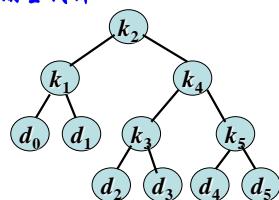


- 附加信息

- 搜索 k_i 的概率是 p_i
- 搜索 d_i 的概率是 q_i

$$\sum_{i=1}^n p_i + \sum_{j=0}^n q_j = 1$$

• 搜索树的期望代价



$$E(T) = \sum_{i=1}^n (DEP_T(k_i) + 1)p_i + \sum_{j=0}^n (DEP_T(d_j) + 1)q_j$$



• 问题的定义

输入: $K=\{k_1, k_2, \dots, k_n\}$, $k_1 < k_2 < \dots < k_n$,
 $P=\{p_1, p_2, \dots, p_n\}$, p_i 为搜索 k_i 的概率
 $Q=\{q_0, q_1, \dots, q_n\}$, q_i 为搜索值 d_i 的概率

输出: 构造 K 的二叉搜索树 T , 使得

$$E(T) = \sum_{i=1}^n (DEP_T(k_i) + 1)p_i + \sum_{j=0}^n (DEP_T(d_j) + 1)q_j$$

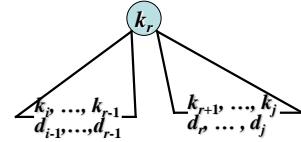
最小



优化二叉搜索树结构的分析

• 划分子问题

$K=\{k_i, k_{i+1}, \dots, k_j\}$ 的优化解的根结点 K 中某个 k_r



如果 $r=i$, 左子树 $\{k_p, \dots, k_{i-1}\}$ 仅包含 d_{i-1}
如果 $r=j$, 右子树 $\{k_{r+1}, \dots, k_j\}$ 仅包含 d_j



• 优化子结构

定理. 如果优化二叉搜索树 T 具有包含关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 的子树 T' , 则 T' 是关于关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 的子问题的优化解.

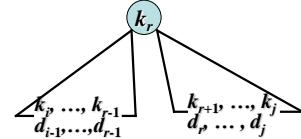
证明: 若不然, 必有关键字集 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 T'' , T'' 的期望搜索代价小于 T' . 用 T'' 替换 T 中的 T' , 可以得到一个期望搜索代价比 T 小的原始问题的二叉搜索树.

与 T 是最优解矛盾.



• 用优化子结构从子问题优化解构造优化解

$K=\{k_i, k_{i+1}, \dots, k_j\}$ 的优化解的根结点 K 中某个 k_r



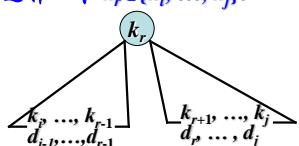
只要对于每个 $k_r \in K$, 确定 $\{k_i, \dots, k_{r-1}\}$ 和 $\{k_{r+1}, \dots, k_j\}$ 的优化解, 我们就可以求出 K 的优化解.

如果 $r=i$, 左子树 $\{k_p, \dots, k_{i-1}\}$ 仅包含 d_{i-1}
如果 $r=j$, 右子树 $\{k_{r+1}, \dots, k_j\}$ 仅包含 d_j



建立优化解的搜索代价递归方程

- 令 $E(i, j)$ 为 $\{k_p, \dots, k_j\}$ 的优化解 T_{ij} 的期望搜索代价
 - 当 $j=i-1$ 时, T_{ij} 中只有叶结点 d_{i-1} , $E(i, i-1)=q_{i-1}$
 - 当 $j \geq i$ 时, 选择一个 $k_r \in \{k_p, \dots, k_j\}$:



当把左右优化子树放进 T_{ij} 时, 每个结点的深度增加1

$$E(i, j) = P_r + E(i, r-1) + W(i, r-1) + E(r+1, j) + W(r+1, j)$$

• 计算 $W(i, r-1)$ 和 $W(r+1, j)$

$$\text{由 } E(LT+1) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 2)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 2)q_l$$

$$E(LT) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 1)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 1)q_l$$

$$\text{知 } W(i, r-1) = E(LT+1) - E(LT) = \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l$$

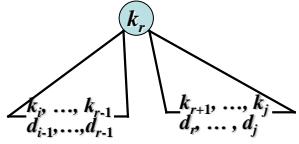
$$\text{同理, } W(r+1, j) = \sum_{l=r+1}^j p_l + \sum_{l=r}^j q_l$$

$$\begin{aligned} \text{令 } W(i, j) &= W(i, r-1) + W(r+1, j) + p_r = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l \\ &= W(i, j-1) + p_j + q_j \end{aligned}$$

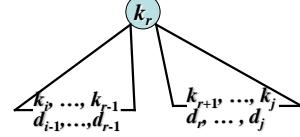
$$W(i, i-1) = q_{i-1}$$

$$W(i, j) = W(i, r-1) + W(r+1, j) + p_r = W(i, j-1) + p_j + q_j$$

$$E(i, j) = P_r + E(i, r-1) + W(i, r-1) + E(r+1, j) + W(r+1, j)$$



$$E(i, j) = E(i, r-1) + E(r+1, j) + W(i, j)$$



In Summary

$$E(i, j) = q_{i-1} \quad \text{If } j=i-1$$

$$E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\} \quad \text{If } j \geq i$$

$$W(i, i-1) = q_{i-1},$$

$$W(i, j) = W(i, j-1) + p_j + q_j$$

自下而上计算优化解的搜索代价

$$E(i, j) = q_{i-1} \quad \text{If } j=i-1$$

$$E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\} \quad \text{If } j \geq i$$

$$\begin{aligned} q_0 &= E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad \cancel{E(1,4)} \\ q_1 &= E(2,1) \quad E(2,2) \quad E(2,3) \quad \cancel{E(2,4)} \\ q_2 &= E(3,2) \quad E(3,3) \quad \cancel{E(3,4)} \\ q_3 &= E(4,3) \quad \cancel{E(4,4)} \\ q_4 &= \cancel{E(5,4)} \end{aligned}$$

$$\bullet \quad W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j + q_j$$

$$\begin{aligned} q_0 &= \cancel{W(1,0)} \quad \cancel{W(1,1)} \quad \cancel{W(1,2)} \quad \cancel{W(1,3)} \quad \cancel{W(1,4)} \\ q_1 &= \cancel{W(2,1)} \quad \cancel{W(2,2)} \quad \cancel{W(2,3)} \quad \cancel{W(2,4)} \\ q_2 &= \cancel{W(3,2)} \quad \cancel{W(3,3)} \quad \cancel{W(3,4)} \\ q_3 &= \cancel{W(4,3)} \quad \cancel{W(4,4)} \\ q_4 &= \cancel{W(5,4)} \end{aligned}$$

• 算法

• 数据结构

- $E[1:n+1; 0:n]$: 储备优化解搜索代价
- $W[1:n+1; 0:n]$: 储备代价增量
- $Root[1:n; 1:n]$: $Root(i, j)$ 记录子问题 $\{k_p, \dots, k_j\}$ 优化解的根

Optimal-BST(p, q, n)

For $i=1$ To $n+1$ Do

$$E(i, i-1) = q_{i-1};$$

$$W(i, i-1) = q_{i-1};$$

For $l=1$ To n Do

For $i=1$ To $n-l+1$ Do

$$j=i+l-1;$$

$$E(i, j)=\infty;$$

$$W(i, j)=W(i, j-1)+p_j+q_j;$$

For $r=i$ To j Do

$$t=E(i, r-1)+E(r+1, j)+W(i, j);$$

If $t < E(i, j)$

Then $E(i, j)=t$; $Root(i, j)=r$;

Return E and $Root$

• 时间复杂性

- (l, i, r) 三重循环，每层循环至多 n 步

- 时间复杂性为 $O(n^3)$

• 空间复杂性

- 二个 $(n+1) \times (n+1)$ 数组，一个 $n \times n$ 数组

- $O(n^2)$

4.6 凸多边形的三角剖分

• 线

多边形 P 上的任意两个不相邻顶点 v_i, v_{i+1} 所对应的线段 $v_i v_{i+1}$ 称为线

• 三角剖分

一个多边形 P 的三角剖分是将 P 划分为不相交三角形的线的集合

• 优化三角剖分问题

- 输入：多边形 P 和代价函数 W

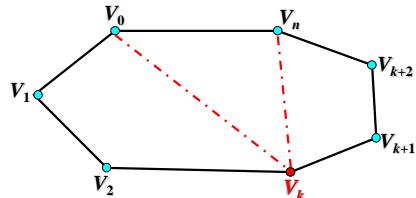
- 输出：在 P 的三角剖分 T ，使得代价 $\sum_{s \in S_T} W(s)$ 最小，其中 S_T 是 T 所对应的三角形集合

优化解结构的分析

• 线

- $P = (v_0, v_1, \dots, v_n)$ 是 $n+1$ 个顶点的多边形

- T_P 是 P 的优化三角剖分，包含三角形 $v_0 v_k v_n$





HIT
CS&E

Homework 1:

给出优化二元搜索树问题优化解的构造算法。

Homework 2:

给出求解下列问题的动态规划算法：

输入：平面上 n 个点： v_1, \dots, v_n, x — 坐标皆不同；

任意两点 (v_i, v_j) 之间的距离 d_{ij} , $i \neq j$.

输出：找到一个最短 bitonic tour:

从最左点出发，严格从左向右走到最右点；

再从最右点开始，严格从右向左回到开始点.