Date: / /

一使用数组 m 和 s

一需要空间 $O(n^2)$

案例分析4：0-1背包问题

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

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

案例分析5：凸多边形最优三角剖分

· 问题描述

一给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得该三角剖分中诸三角形上权之和为最小。

· 提炼问题三要素

一输入：凸多边形 $P=\{v_0,v_1,\cdots,v_n\}$，$v_0=v_{n+1}$

一输出：凸多边形的最优权值三角剖分

一约束：三角形权之和最小

动态规划步骤1：分析最优子结构性质

动态规划步骤2：建立最优三角剖分的递推关系

$$t[i][j]=\begin{cases}0 & i=j\\ \min\limits_{i\leq k<j}\{t[i][k]+t[k+1][j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

» 表达式的语法树

· 一个表达式的完全加括号方式相应于一棵完全二叉树

· 凸多边形 $\{v_0,v_1,\cdots,v_n\}$ 的三角剖分也可以用语法树表示

» 转换与匹配

矩阵 A_i 对应于凸 $(n+1)$ 边形中的一条边 $v_{i-1}v_i$

Date: / /

矩阵连乘积 $A[i+1:j]$ 三角剖分中的一条弦 i,j $i<j$

动态规划步骤4：获取构造最优解的信息

```
Matrix-Chain-Order(p)
n=length(p)-1;
FOR i=1 TO n DO
    m[i,i]=0;
FOR l=2 TO n DO
    FOR i=1 TO n-l+1 DO
        j=i+l-1;
        m[i,j]=∞;
        FOR k=i TO j-1 DO
            q=m[i,k]+m[k+1,j]+p_{i-1}p_k p_j
            IF q<m[i,j] THEN m[i,j]=q, s[i,j]=k;
Return m and s.
```

$s[i,j]$ 记录 $A_iA_{i+1}\cdots A_j$ 的最优划分处在 A_k 与 A_{k+1} 之间

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 $A_i\cdots A_j$ 的最优划分处；

$s[i,s[i,j]]$ 记录 $A_i\cdots A_{s[i,j]}$ 的最优划分处

$s[s[i,j]+1,j]$ 记录 $A_{s[i,j]+1}\cdots A_j$ 的最优划分处。

调用 Print-Optimal-Parens(s,1,n) 即可输出 $A_{1\sim n}$ 的优化计算次序

算法复杂性

·时间复杂性

-计算代价的时间

·(l,i,k) 三层循环，每层至多 n-1 步

Date: / /

- $O(n^3)$
  - 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 $O(n^2)$

动态规划算例 6：多边形游戏

- 问题描述：对于给定的多边形，按如下游戏规则，计算最高得分。

游戏规则：

✓ 首先，将一条边删除。随后 n-1 步如下：

✓ 选择一条边 E 以及由 E 连接着的 2 个顶点 V1 和 V2

✓ 用一个新的顶点取代边 E 以及由 E 连接着的 2 个顶点 V1 和 V2。

将由顶点 V1 和 V2 的整数值经过边 E 上由运算得到的结果赋予新顶点。

✓ 最后，所有边都被删除，游戏结束

✓ 游戏的得分就是所剩顶点上的整数值。

- 提交问题三要素

输入：含 n 个顶点的多边形，顶点有整数值，边有运算符

输出：最高得分

约束：游戏规则

动态规划步骤 1：分析最优子结构性质

(1) 当 op[i+s] = '+' 时，显然有 a+c ≤ m ≤ b+d

(2) 当 op[i+s] = '*' 时有 $\min\{ac, ad, bc, bd\} \le m \le \max\{ac, ad, bc, bd\}$

即：主链的最大值和最小值可由子链的最大值和最小值得到。

动态规划步骤 2：建立递推关系

设 m[i,j,0] 是链 p(i,j) 合并的最小值，而 m[i,j,1] 是最大值。

Date: / /

若最优合并在 $op[i+s]$ 处，将 $p(i,j)$ 分为两个长度小于 j 的子链的最大值和最小值均已计算出。

即：$a=m[i,s,0]$ $b=m[i,s,1]$ $c=m[i+s,j-s,0]$ $d=m[i+s,j-s,1]$

(1) 当 $op[i+s]$ = '+' 时，$m[i,j,0]=a+c$; $m[i,j,1]=b+d$

(2) 当 $op[i+s]$ = '*' 时 $m[i,j,0]=\min\{ac,ad,bc,bd\}$; $m[i,j,1]=\max\{ac,ad,bc,bd\}$

动态规划步骤 3.4 略.

贪心算法

案例分析1：活动安排问题

· 提取问题要素（问题形式化定义）

—输入：$S=\{1,2,\cdots,n\}$, $F=\{[s_i,f_i]\}$, $n \ge i \ge 1$

—输出：S 的最大相容集合

—约束条件：相容、最大

贪心思想：

为了选择最多的相容活动，每次选 f_i 最小的活动，使我们能够选更多的活动。

[算法思路] 将 n 个活动按结束时间非减序排列，依次考虑活动 i，若 i 与已选择的活动相容，则添加此活动到相容活动子集。

[例] 设待安排的 11 个活动起止时间按结束时间的非减序排列

i	1	2	3	4	5	6	7	8	9	10	11
s[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

最大相容活动子集 (1, 4, 8, 11),

也可表示为等长 n 元数组：(1,0,0,1,0,0,0,1,0,0,1)

案例分析2：背包问题

[最优化描述] 找一个 n 元向量 $(x_1,\cdots,x_n)$ $0 \le x_i \le 1$ 使得 $\sum_{i=1}^{n} w_i x_i \le C$ 且 $\max \sum_{i=1}^{n} v_i x_i$，其中 $C, w_i, v_i > 0$, $1 \le i \le n$.

Date: / /

背包问题 VS 0-1背包问题

这两类问题都具有最优子结构性质，极为相似，但背包问题可以用贪心算法求解，而0-1背包问题却不能用贪心算法求解。

- 考虑下列情况的背包问题

  $n=3$, $C=20$, $(v_1,v_2,v_3)=(25,24,15)$, $(w_1,w_2,w_3)=(18,15,10)$

  其中的4个可行解是：

	(x_1,x_2,x_3)	$\sum w_i x_i$	$\sum v_i x_i$
①	$(1/2,1/3,1/4)$	16.5	24.25
②	$(1,2/15,0)$	20	28.2
③	$(0,2/3,1)$	20	31
④	$(0,1,1/2)$	20	31.5

贪心方法的数据选择策略：

1. 效益大的优先：将效益值的非增次序将物品放入背包。即：$(v_1,v_2,v_3)=(25,24,15)$
2. 重量小的优先：将物品重量的非降次序将物品放入背包，即 $(w_3,w_2,w_1)=(10,15,18)$
3. 单位效益大的优先：即将物品的 v_i/w_i 比值的非增次序装入背包。

   $(v_2/w_2, v_3/w_3, v_1/w_1)=(24/15, 15/10, 25/18)$

案例分析3：最优装载问题

问题描述：有一批集装箱要装上一艘载重量为C的轮船。其中集装箱i的重量为 w_i。最优装载问题要求确定在载体允许的情况下，将尽可能多的集装箱装上轮船。

[数学模型] 输入：$(x_1,x_2,\cdots,x_n)$，$x_i=0$，货箱i不装船；$x_i=1$，货箱i装船。

可行解：满足约束条件 $\sum_{i=1}^{n} w_i x_i \le C$ 的输入

优化函数：$\sum_{i=1}^{n} x_i$

最优解：使优化函数达到最大值的一种输入。

[算法思路] 将装船过程划分为多步选择，每步装一个货箱，首次从剩下的货箱中选择重量最轻的货箱，如此下去直到所有货箱为装上船或船上不能再容纳其他任何一个货箱。

Date: / /

```
template <class Type>
void Loading(int x[], Type w[], Type c, int n)
{
    int *t = new int[n+1];
    Sort(w, t, n);
    for (int i=1; i<=n; i++) x[i]=0;
    for (int i=1; i<=n && w[t[i]]<=c; i++) {x[t[i]]=1; c-=w[t[i]];}
}
```

算法Loading的主要计算量在于将集装箱依其重量从小到大排序，故算法所需的计算时间为$O(n\log n)$。

案例分析4：最优编码树问题

贪心思想：每次把选择具有最小频率的两个结点生成一棵子树，直至形成树。

$$T(n)=O(n)+O(n\log n)=O(n\log n)$$

练习1：