

算法设计与分析 (2011-2012 秋) 作业整理

授课老师: 陈玉福

周吕文
zhou.lv.wen@gmail.com

中国科学院力学研究所
2011 年 11 月 28 日

目 录

1	第一章 复杂性分析初步	2
1.1	习题 1.1	2
1.2	习题 1.5	2
1.3	习题 1.6	3
1.4	习题 1.8	4
2	第二章 图与遍历算法	5
2.1	习题 2.1	5
2.2	习题 2.5	5
2.3	习题 2.6	6
2.4	习题 2.7	7
2.5	习题 2.8	8
3	第三章 分治算法	10
3.1	习题 3.1	10
3.2	习题 3.4	10
4	第四章 贪心算法	11
4.1	习题 4.1	11
4.2	习题 4.2	11
4.3	习题 4.3	12
4.4	习题 4.4	13
5	第五章 动态规划算法	14
5.1	习题 5.1	14
5.2	习题 5.2	16
5.3	习题 5.3	17
5.4	习题 5.4	18
	附录	19
A	两种排序算法的 C++ 程序及其时间复杂性测试程序	19
B	Huffman 编码 C++ 程序	21

第一章 复杂性分析初步

习题 1.1 试确定下述程序的执行步数, 该函数实现一个 $m \times n$ 矩阵与一个 $n \times p$ 矩阵之间的乘法:

矩阵乘法运算

```

1  template<class T>
2  void Mult(T **a, T **b, int m, int n, int p)
3  {
4      for(int i=0; i<m; i++)
5          for(int j=0; j<p; j++){
6              T sum=0;
7              for(int k=0; k<n; k++)
8                  Sum+=a[i][k]*b[k][j];
9              C[i][j]=sum;
10     }
11 }

```

解: 表1是题中程序各行的执行步数统计. 因此该程序执行的总步数为 $\text{total step} = (m+1) + m(p+1) + mp + mp(n+1) + mpn + mp = 2mpn + 4mp + 2m + 1$ ■

表 1: 矩阵乘法运算执行步数统计

行数	s/e	频率	总步数	行数	s/e	频率	总步数
01	0	0	0	07	1	$mp(n+1)$	$mp(n+1)$
02	0	0	0	08	1	mpn	mpn
03	0	0	0	09	0	mp	mp
04	1	$m+1$	$m+1$	10	0	0	0
05	1	$m(p+1)$	$m(p+1)$	11	0	0	0
06	1	mp	mp				

习题 1.5 下面那些规则是正确的? 为什么?

1. $\{f(n) = O(F(n)), g(n) = O(G(n))\} \Rightarrow f(n)/g(n) = O(F(n)/G(n));$
2. $\{f(n) = O(F(n)), g(n) = O(G(n))\} \Rightarrow f(n)/g(n) = \Omega(F(n)/G(n));$
3. $\{f(n) = O(F(n)), g(n) = O(G(n))\} \Rightarrow f(n)/g(n) = \Theta(F(n)/G(n));$
4. $\{f(n) = \Omega(F(n)), g(n) = \Omega(G(n))\} \Rightarrow f(n)/g(n) = \Omega(F(n)/G(n));$
5. $\{f(n) = \Omega(F(n)), g(n) = \Omega(G(n))\} \Rightarrow f(n)/g(n) = O(F(n)/G(n));$
6. $\{f(n) = \Theta(F(n)), g(n) = \Theta(G(n))\} \Rightarrow f(n)/g(n) = \Theta(F(n)/G(n));$

解: 对以题中 6 个小命题, 错误的给出反例, 正确的给出证明, 具体如下:

1. **错误:** 在满足题设条件下, 不妨设 $f(n) = F(n) = g(n) = 1, G(n) = n$, 则显然不存在正常数 c , 当 n 足够大时使得: $1 = f(n)/g(n) \leq cF(n)/G(n) = c/n$.
2. **错误:** 在满足题设条件下, 不妨设 $f(n) = G(n) = g(n) = 1, F(n) = n$, 则显然不存在正常数 c , 当 n 足够大时使得: $1 = f(n)/g(n) \geq cF(n)/G(n) = cn$.
3. **错误:** 由 1,2 两小题中的反例, 可知在满足题设条件下, $f(n)/g(n)$ 和 $F(n)/G(n)$ 不一定同阶.
4. **错误:** 在满足题设条件下, 不妨设 $f(n) = F(n) = G(n) = 1, g(n) = n$, 则显然不存在正常数 c , 当 n 足够大时使得: $1/n = f(n)/g(n) \geq cF(n)/G(n) = c$.
5. **错误:** 在满足题设条件下, 不妨设 $F(n) = G(n) = g(n) = 1, f(n) = n$, 则显然不存在正常数 c , 当 n 足够大时使得: $n = f(n)/g(n) \leq cF(n)/G(n) = c$.
6. **正确:** 由题意知: 存在正常数 a_1 和 a_2 , 及 b_1 和 b_2 使得

$$a_1F(n) \leq f(n) \leq a_2F(n), \quad b_1G(n) \leq g(n) \leq b_2G(n)$$

则有

$$\frac{a_1F(n)}{b_2G(n)} \leq \frac{f(n)}{g(n)} \leq \frac{a_2F(n)}{b_1G(n)}$$

令 $c_1 = a_1/b_2, c_2 = a_2/b_1$ 则有

$$c_1 \frac{F(n)}{G(n)} \leq \frac{f(n)}{g(n)} \leq c_2 \frac{F(n)}{G(n)}$$

■

习题 1.6

$$4n^2, \log n, 3^n, 20n, n^{2/3}, n!$$

解: 按照渐近阶从低到高的顺序排列上述表达式:

$$\log n < n^{2/3} < 20n < 4n^2 < 3^n < n!$$

■

习题 1.7

1. 假设某算法在输入规模是 n 时为 $T(n) = 3 * 2^n$. 在某台计算机上实现并完成该算法的时间是 t 秒. 现有另一台计算机, 其运行速度为第一台的 64 倍, 那么, 在这台计算机上用同一算法在 t 秒内能解决规模为多大的问题?
2. 若上述算法改进后的新算法的计算为 $T(n) = n^2$, 则在新机器上用 t 秒时间能解决输入规模为多大的问题?
3. 若进一步改进算法, 最新的算法的时间复杂度为 $T(n) = 8$, 其余条件不变, 在新机器上运行, 在 t 秒内能够解决输入规模为多大的问题?

解:

1. 设该算法在原计算机和新计算机上每步运行所需的时间分别为 t_1, t_2 , 则根据题意得:

$$t_1 = \frac{t}{T(n)}, t_2 = \frac{t_1}{64}$$

设该算法在新计算机上能解决的规模为 n' , 则有

$$3 \times 2^{n'} \times t_2 = 3 \times 2^{n'} \times \frac{t_1}{64} = 3 \times 2^n \times t_1$$

解得 $n' = n + 6$, 因此在新计算机上用同一算法在 t 秒内能解决规模为 $n + 6$.

2. 算法改进后的新算法的计算为 $T(n) = n^2$. 则有

$$t = n'^2 \times t_1 = 3 \times 2^n \times t_1$$

解得 $n' = \sqrt{3 \cdot 2^{n+6}}$. 因此在新计算机上 t 秒内能解决规模为 $\sqrt{3 \cdot 2^{n+6}}$.

3. 最新的算法的时间复杂度为 $T(n) = 8$. 则有

$$t = 8 \times t_1 = 3 \times 2^n \times t_1$$

显然上式与 n' 无关, 即 $T(n)$ 不随 n 变化, 所以任意规模的 n' 都能在 t 秒内解决. ■

习题 1.8 Fibonacci 数有递推关系:

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

试求出 $F(n)$ 的表达.

解: 设当 $n > 3$ 时 $F(n), F(n-1), F(n-2)$ 满足:

$$F(n) = F(n-1) + F(n-2)$$

故其特征方程为

$$\lambda^2 = \lambda + 1$$

解得: $\lambda_1 = (1 + \sqrt{5})/2, \lambda_2 = (1 - \sqrt{5})/2$, 则可设

$$F(n) = a\lambda_1^n + b\lambda_2^n$$

由 $F(2) = 2, F(3) = 3$, 解得 $a = \frac{1+\sqrt{5}}{2\sqrt{5}}, b = -\frac{1-\sqrt{5}}{2\sqrt{5}}$ 因此有

$$F(n) = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right]$$

■

第二章 图与遍历算法

习题 2.1 证明下列结论:

1. 在一个无向图中, 如果每个顶点的度大于等于 2, 则该图一定含有圈;
2. 在一个有向图 D 中, 如果每个顶点的出度都大于等于 1, 则该图一定含有一个有向圈.

1. **证:** 设该无向图无圈, 有 m 条边, n 个顶点. 则该无向图为树或森林, 因此满足 $m = n - k$, 又由如果每个顶点的度大于等于 2 有

$$\sum d(v) = 2|E| = 2m \geq 2n$$

故有

$$m \geq n \implies n - k \geq n, (k = 1, 2, \dots) \quad (1)$$

显然式 (1) 是不成立的, 故假设该无向图无圈成立, 即该图一定含有圈.

2. **证:** 设该有向图中的最长有向迹为 $P = V_1 V_2 \cdots V_k$, 又因为每个顶点的度大于等于 1, 故存在 V' 为 V_k 连接到的点, 若 $V' \notin P$, 则 $V_1 V_2 \cdots V_k V'$ 比 P 更长, 与假设矛盾. 因此, $V' \in P$, 设 $V' = V_n (n \leq k)$, 故存在有向圈 $V_n V_{n+1} \cdots V_k V_n$

■

习题 1.5 实现图的 D-搜索算法. 要求用 ALGEN 语言写出算法的伪代码, 或者用一种计算机高级语言写出程序.

解: 先将起始顶点存入栈中. 搜索时, 取出栈顶元素, 遍历其邻点, 将未搜索的存入栈, 遍历其邻点后, 重复此过程, 直至栈变为空栈. 表2是图的 D-搜索算法伪代码. 表2中调用

表 2: 图的 D-搜索算法伪代码

```

Proc DBFT(G,m)//m 为不连通分支数
  count:=0;//计数器, 标示已经被访问的顶点个数
  for i to n do
    visited[i]:=0;//数组 visited 标示各顶点被访问的序数, 其元素初始化为 0.
  endfor
  for i to m do //遍历不连通分支的情况
    if visited[i]=0 then
      DBFS (i);
    end{if}
  end{for}
end{DBFT}

```

的 DBFS 为由一点出发的 D-搜索, 其伪代码见表3

表 3: 由一点出发的 D-搜索算法伪代码

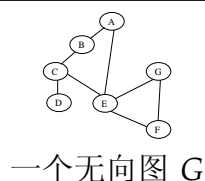
```

Proc DBFS(v)
  //数组 visited 标示各顶点被访问的序数, 其元素初始化为 0
  // 计数器 count 计数到目前为止已经被访问的顶点个数, 初始化为 0
  PushStack (v , S); //首先访问 v, 将 S 初始化为只含有一个元素 v 的栈
  count :=count +1; visited[v] := count;
  While S 非空 do
    u :=PullHead(S); //取出栈顶的元素 u, 并从栈中删除
    for 邻接于 u 的所有顶点 w do
      if visited[w] = 0 then
        PushStack(w,S); //将 w 存入栈 S
        count :=count +1; visited[w] := count;
      end{if}
    end{for}
  end{while}
end{DBFS}

```

■

习题 2.6 右图的无向图以邻接链表存储, 而且在关于每个顶点的链表中与该顶点相邻的顶点是按照字母顺序排列的. 试以此图为例描述讲义中算法 DFNL 的执行过程.



解: 题中所给的无向图所对应的邻接链表如图1所示. 此图讲义中算法 DFNL 的执行过

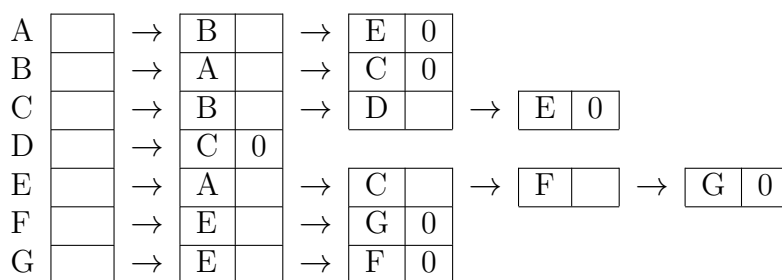


图 1: 无向图的邻接链表

程如下:

初始化数组 $DFN:=0$, $num:=1$;

A 为树的根节点, 对 A 计算 $DFNL(A, \text{null})$, $DFN(A):=num=1$; $L(A):=num=1$; $num:=1+1=2$.

从邻接链表查到 A 的邻接点 B,

因为 $DFN(B)=0$, 对 B 计算 $DFNL(B, A)$, $DFN(B):=num=2$; $L(B):=num=2$; $num:=2+1=3$.

查邻接链表得到 B 的邻接点 A, 因为 $DFN(A)=1$ 0, 但 $A=A$, 即是 B 的父节点, 无操作.

接着查找邻接链表得到 B 的邻接点 C, 因为 $DFN(C)=0$,

对 C 计算 $DFNL(C, B)$, $DFN(C):=num=3$; $L(C):=num=3$; $num:=3+1=4$.

查找 C 的邻接点 B, 因为 $DFN(B)=1$ 0, 但 $B=B$, 即是 C 的父节点, 无操作.

接着查找邻接链表得到 C 的邻接点 D, 因为 $DFN(D)=0$,

对 D 计算 $DFNL(D, C)$, $DFN(D):=num=4$; $L(D):=num=4$; $num:=4+1=5$.

查找得 D 邻接点 C, 而 $DFN(C)=3$ 0, 但 $C=C$, 为 D 的父节点, $L(D)$ 保持不变.

D 的邻接链表结束, $DFNL(D, C)$ 的计算结束.

返回到 D 的父节点 C, 查找邻接链表得到 C 的邻接点 E, 因为 $DFN(E)=0$,

对 E 计算 $DFNL(E, C)$, $DFN(E):=num=5$; $L(E):=num=5$; $num:=5+1=6$;

查找得 E 邻接点 A, 因 $DFN(A)=1$ 0, 又 $A \neq C$, 变换 $L(E)=\min(L(E), DFN(A))=1$.

查找得 E 邻接点 C, 因 $DFN(C)=3$ 0, 但 $C=C$, 无操作.

查找得 E 邻接点 F, 因 $DFN(F)=0$,

对 F 计算 $DFNL(F, E)$, $DFN(F):=num=6$; $L(F):=num=6$; $num:=6+1=7$;

查找得 F 邻接点 E, 因 $DFN(E)=5$ 0, 但 $E=E$, 无操作.

查找得 F 邻接点 G, 因 $DFN(G)=0$,

对 G 计算 $DFNL(G, F)$, $DFN(G):=num=7$; $L(G):=num=7$; $num:=7+1=8$;

查找 G 邻接点 E, 因 $DFN(E)=5$ 0, 又 $E \neq F$, $L(G)=\min(L(G), DFN(E))=5$

查找得 G 邻接点 F, 因 $DFN(F)=6$ 0, 但 $F=F$, 无操作.

G 的邻接链表结束, $DFNL(G, F)$ 的计算结束.

$L(F):=\min(L(F), L(G))=\min(6, 5)=5$

F 的邻接链表结束, $DFNL(F, E)$ 的计算结束.

$L(E):=\min(L(E), L(F))=\min(1, 5)=1$

E 邻接链表结束, $DFNL(E, C)$ 计算结束.

$L(C):=\min(L(C), L(E))=\min(3, 1)=1$

C 的邻接链表结束, $DFNL(C, B)$ 计算结束.

$L(B):=\min(L(B), L(C))=\min(2, 1)=1$

查找 B 的邻接链表结束, $DFNL(B, A)$ 计算结束.

$L(A):=\min(L(A), L(B))=1$

查找得 A 的邻接点 E, 因 $DFN(E)=0$, 又 $E \neq \text{null}$, 则 $L(A)=\min(L(A), DFN(E))=1$

查找 A 的邻接链表结束, $DFNL(A, \text{null})$ 计算结束.

■

习题 2.7 对图的另一种检索方法是 D-Search. 该方法与 BFS 的不同之处在于将队列换成栈, 即下一个要检测的结点是最新加到未检测结点表的那个结点.

1. 写一个 D-Search 算法;
2. 证明由结点 v 开始的 D-Search 能够访问 v 可到达的所有结点;
3. 你的算法的时, 空复杂度是什么?

1. **解:** D-Search 算法如习题 1.5 中的表3所示.

2. **证:** 设 v_n 为 v 可到达的某一节点, v 与 v_n 间的迹为 $P = vv_1v_2 \cdots v_n$. 由表3的算法可知, v 最先被放入栈, 并被访问. 其后取出栈顶元素并访问, 同时其邻接的顶点

被放入栈中, 重复此过程, 直至栈变空. 因此当 v 被取出时 v_1 将被放出栈中, 其后当 v_1 被取出时, v_1 被访问的同时, v_1 的邻点 v_2 被放入栈中. 假设 v_k 与 v_{k+1} 相邻, 若 v_k 可访问, 则 v_{k+1} 也可被访问. 下面用数学归纳法证明题设:

- 显然该假设对于 $v(v_0), v_1$ 成立.
- 当 v_k 可访问时, v_k 的所有邻点, 包括 v_{k+1} 将被放入栈中, 栈中的顶点将逐个被取出访问, 因此 v_{k+1} 也将被访问.

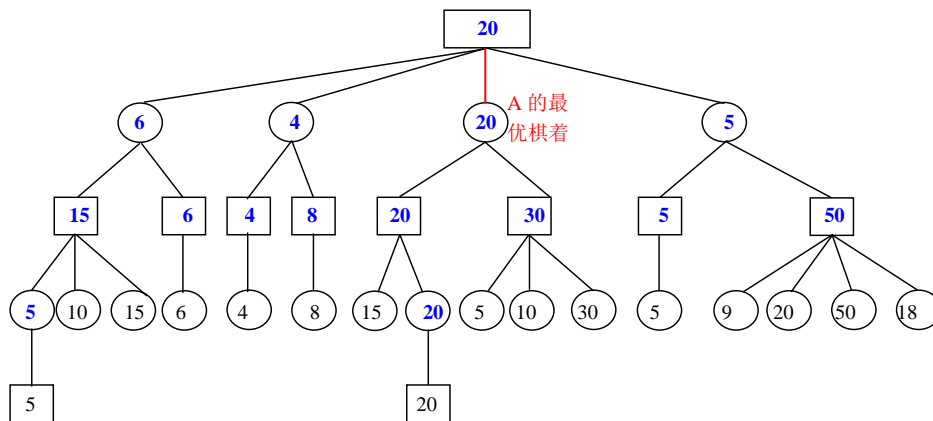
所以, D-Search 能够访问 v 可达到的结点 v_n . 由于 v_n 是任意假设的, 因此 D-Search 能够访问 v 可达到的所有结点.

3. 解: 图 G 的 D-搜索算法能够访问 G 中由 v 可能到达的所有顶点如果记 $t(v, \varepsilon)$ 和 $s(v, \varepsilon)$ 为 D-搜索算法在任意一个具有 v 个顶点和 ε 条边的连通图 G 上所花的最大时间和最大空间. 则可证明其时空复杂度:

- 除节点 v 外, 只有当结点 w 满足 $visited[w]$ 才被存入栈中, 因此每个节点至多有一次被存入栈中, 需要的栈空间至多是 $v - 1$; $visited$ 数组需要的空间为 v ; 其于变量所用空间为 $O(1)$, 所以该算法的空间复杂度为 $s(v, \varepsilon) = \Theta(v)$.
- 如果使用邻接链表, 语句 for 循环要做 $d(u)$ 次, 而语句 while 循环需要做 v 次, 因而整个循环做 $\sum_{u \in V} d(u) = 2\varepsilon$ 次 $O(1)$ 操作, 又 $visited$ 和 $count$ 的赋值都需要 v 次操作, 因此时间复杂度为 $t(v, \varepsilon) = \Theta(v + \varepsilon)$
- 如果采用邻接矩阵, 则语句 while 循环总共需要做 v^2 次操作, $visited$ 和 $count$ 的赋值都需要 v 次操作, 因而时间复杂度为 $t(v, \varepsilon) = \Theta(v^2)$.

■

习题 2.8 考虑下面这棵假想的对策树:



1. 使用最大最小方法 (2-4-2) 式获取各结点的值;
2. 弈者 A 为获胜应该什么棋着?
3. 列出算法 VEB 计算这棵对策树结点的值时各结点被计算的顺序;
4. 对树中每个结点 X , 用 (2-4-3) 式计算 $V(X)$;
5. 在取 X 根, $l = 10$, $LB = -\infty$, $D = \infty$ 的情况下, 用算法 AB 计算此树的根的值期间, 这棵树的那些结点没有计算?

1. 各结点的值已在题中所给图中标出;
2. 弈者 A 为获胜应该走的棋着已在题中所给图中标出;
3. 算法 VEB 计算这棵对策树结点的值时各结点被计算的顺序如图2中的红色数字;
4. 对树中每个结点 X, 用 (2-4-3) 式计算 $V(X)$ 所得的值, 已标在图2中的方框或圆圈中 (蓝色数字);

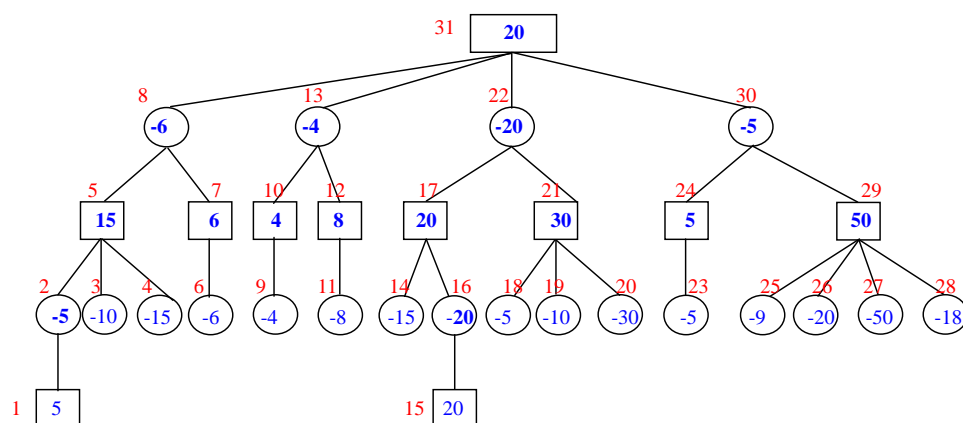


图 2: 第 (3)(4) 小问解

5. 在取 X 根, $l = 10$, $LB = -\infty$, $D = \infty$ 的情况下, 图2中标有以下红色数字中的结点没有被计算:

11, 12, 25, 26, 27, 28, 29

■

第三章 分治算法

习题 3.1 编写程序实现归并排序算法 MergeSortL 和快速排序算法 QuickSort;

解: 归并排序算法和快速排序算法的 C++ 程序见附录A(含时间复杂性测试). ■

习题 3.4 说明算法 PartSelect 的平均时间复杂性为 $O(n)$.

提示: 假定数组中的元素各不相同, 且第一次划分时划分元素 v 是第 i 小元素的概率为 $1/n$. 因为 Partition 中的 case 语句所要求的时间都是 $O(n)$, 所以, 存在常数 c , 使得算法 PartSelect 的平均时间复杂度 $C_A^k(n)$ 可以表示为

$$C_A^k(n) \leq cn + \frac{1}{n} \sum_{1 \leq i < k} C_A^{k-1}(n-i) + \sum_{k < i \leq n} (C_A^k(i-1))$$

令 $R(n) = \max(C_A^k(n))$, 取 $c \geq R(1)$ 试证明 $R(n) \leq 4cn$

证: 设 $C_A^k(n)$ 表示在含 n 个元素的数组 A 中搜索第 k 小元素的平均时间复杂度, 假定数组中的元素各不相同, 且第一次划分时划分元素 v 是第 i 小元素的概率为 $1/n$. 因为 Partition 中的 case 语句所要求的时间都是 $O(n)$, 所以, 存在常数 c , 使得算法 PartSelect 的平均时间复杂度 $C_A^k(n)$ 可以表示为

$$C_A^k(n) \leq cn + \frac{1}{n} \sum_{1 \leq i < k} C_A^{k-1}(n-i) + \sum_{k < i \leq n} (C_A^k(i-1))$$

令 $R(n) = \max(C_A^k(n))$, 取 $c \geq R(1)/4$. 当 $n = 2$ 时, 有

$$R(2) \leq 2c + \frac{1}{2}R(1) = 2c + 2c = 4c$$

设对于 $n > 2$ 的所有自然数, $R(n) \leq 4cn$, 则有

$$\begin{aligned} R(n) &\leq cn + \frac{1}{n} \left(\sum_{1 \leq i < k} R(n-i) + \sum_{k \leq i < n} R(i-1) \right) \\ &\leq cn + \frac{4c}{n} \left(\sum_{1 \leq i < k} (n-i) + \sum_{k \leq i < n} (i-1) \right) \\ &\leq cn + \frac{4c}{n} \left(\frac{(k-1)(2n-k)}{2} + \frac{(n-k)(k+n-1)}{2} \right) \\ &\leq cn - \frac{4c}{n} \frac{(k-1)(2n-k) + (n-k)(k+n-1)}{2} \\ &\leq cn - \frac{4c}{n} \left(k^2 - (n+1)k - \frac{n^2-3n}{2} \right) \\ &\leq cn - \frac{4c}{n} \left(-\left(\frac{n+1}{2}\right)^2 - \frac{n^2-3n}{2} \right) \\ &\leq cn + c \frac{3n^2-4n+1}{n} \\ &\leq cn + 3c(n-1) \\ &\leq 4cn \end{aligned}$$

因此, 平均时间复杂度为 $C_A^k(n) \leq 4cn$, 故有 $C_A^k(n) = O(n)$. ■

第四章 贪心算法

习题 4.1 设有 n 个顾客同时等待一项服务. 顾客 i 需要的服务时间为 t_i , $1 \leq i \leq n$. 应该如何安排 n 个顾客的服务次序才能使总的等待时间达到最小? 总的等待时间是各顾客等待服务的时间的总和. 试给出你的做法的理由 (证明).

解: 所需服务时间越短, 排在越前面. 即有 $t_1 \leq t_2 \leq \dots \leq t_n$. 其中下标为服务的顺序. 下面证明其为最优解.

证: 设 $T = [t_1, t_2, \dots, t_i, \dots, t_n]$, $t_i \leq t_{i+1}$ 不是最优解. $Y = [y_1, y_2, \dots, y_i, \dots, y_n]$ 是最优解, 则必存在 i 使得 $y_i > y_{i+1}$ 则对于解 Y , 各顾客等待服务的时间总和为

$$\begin{aligned} T_Y &= \underbrace{0}_1 + \underbrace{y_1}_2 + \underbrace{y_1 + y_2}_3 + \dots + \underbrace{y_1 + \dots + y_i}_{i+1} + \dots + \underbrace{y_1 + \dots + y_{n-1}}_n \\ &= (n-1)y_1 + (n-2)y_2 + \dots + (n-i)y_i + (n-(i+1))y_{i+1} + \dots + 2y_{n-2} + y_{n-1} \\ &> (n-1)y_1 + (n-2)y_2 + \dots + (n-i)y_{i+1} + (n-(i+1))y_i + \dots + 2y_{n-2} + y_{n-1} \end{aligned}$$

因此, $Y' = [y_1, y_2, \dots, y_{i+1}, y_i, \dots, y_n]$ 比 Y 更优, 因此 Y 不是最优解, 故假设不成立. 因此 $T = [t_1, t_2, \dots, t_i, \dots, t_n]$, $t_i \leq t_{i+1}$ 是最优解. ■

习题 4.2 字符 a-h 出现的频率分布恰好是前 8 个 Fibonacci 数, 它们的 Huffman 编码是什么? 将结果推广到 n 个字符的频率分布恰好是前 n 个 Fibonacci 数的情形. Fibonacci 数的定义为: $F_0 = 1, F_1 = 1, F_n = F_{n-2} + F_{n-1}$ if $n > 1$.

解: 根据 Fibonacci 数的定义求可求出的 a-h 出现的频率分布, 再由 Huffman 编码算法可得到 Huffman 编码树. Huffman 编码算法执行过程如图4 因此, 可得到 a-h 的

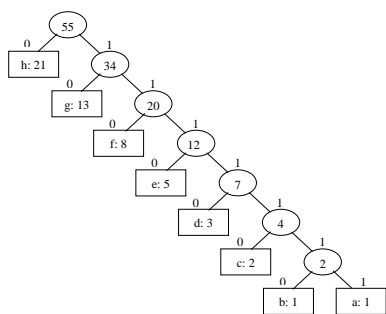


图 3: Huffman 编码树

1	a:1	b:1	c:2	d:3	e:5	f:8	g:13	h:21
2	c:2	a+b:2		d:3	e:5	f:8	g:13	h:21
3	d:3	(a+b)+c:4			e:5	f:8	g:13	h:21
4	e:5	((a+b)+c)+d:7				f:8	g:13	h:21
5	f:8	(((a+b)+c) + d)+e:12					g:13	h:21
6	g:13	((((a+b)+c) + d) + e) + f:20						h:21
7	h:21	(((((a+b)+c) + d) + e) + f) + g:34						
8	((((((a+b)+c) + d) + e) + f) + g) + h:55							

图 4: Huffman 编码算法执行过程

Huffman 编码, 如图3所示. 可进一步将结果推广到 n 个字符的频率分布恰好是前 n 个 Fibonacci 数的情形, 第 i 个数的 Huffman 编码为

$$\begin{cases} \underbrace{1 \dots 1}_{n-1 \text{ 个}} & i = 1 \\ \underbrace{1 \dots 10}_{n-i \text{ 个}} & 0 < i \leq n \end{cases}$$

习题 4.3 设 p_1, p_2, \dots, p_n 是准备存放到长为 L 的磁带上的 n 个程序, 程序 p_i 需要的带长为 a_i . 设 $\sum_{i=1}^n a_i > L$, 要求选取一个能放在带上的程序的最大子集合 (即其中含有最多个数的程序) Q . 构造 Q 的一种贪心策略是按 a_i 的非降次序将程序计入集合.

1. 证明这一策略总能找到最大子集 Q , 使得 $\sum_{p_i \in Q} a_i \leq L$.
2. 设 Q 是使用上述贪心算法得到的子集合, 磁带的利用率可以小到何种程度?
3. 试说明 1 问中提到的设计策略不一定得到使取最大值的子集合.

1. **证:** 设 $a_1 \leq a_2 \leq \dots \leq a_n$ 是按非降次序排列的. 设贪心策略构造的最大子集 $Q = [a_1, a_2, \dots, a_m]$ 使得

$$\sum_{i=1}^m a_i \leq L, \sum_{i=1}^{m+1} a_i > L$$

假设 Q 不是最大子集, 则至少存在 $m+1$ 个元素的集合 $P = [a_{i_1}, a_{i_2}, \dots, a_{i_{m+1}}]$ 使得

$$\sum_{j=1}^{m+1} a_{i_j} \leq L, \sum_{j=1}^{m+2} a_{i_j} > L$$

因为 $a_1 \leq a_2 \leq \dots \leq a_n$ 是按非降次序排列的, 所以有

$$L < \sum_{i=1}^{m+1} a_i < \sum_{j=1}^{m+1} a_{i_j}$$

这与假设是矛盾的, 因为不存在大于 m 个元素的集合, 即这一策略总能找到最大子集 Q , 使得 $\sum_{p_i \in Q} a_i \leq L$.

2. **解:** 磁带的利用率为

$$\alpha = \frac{\sum_{p_i \in Q} a_i}{L}$$

显然当 $a_1 > L$ 时 $\alpha = 0$, 因此磁带的利用率最小可为 0.

3. **解:** 贪心策略构造总能找到最大子集 Q , 只表示 Q 中的元素能达到最大 m , 并不表示能使得放在带上的程序最长. 如果存在

$$a_{m+1} + \sum_{i=1}^{m-1} a_i \leq L$$

显然其利用率比贪心策略构造出的 Q 大, 因此 1 问中提到的设计策略不一定得到使取最大值的子集合. ■

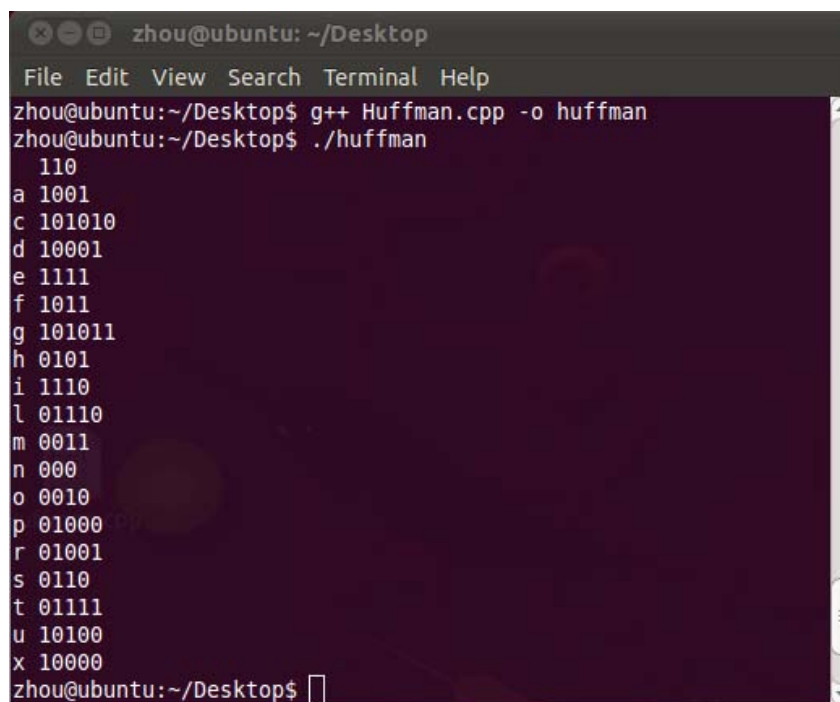
习题 4.4 写出 Huffman 编码的伪代码, 并编程实现.

解:Huffman 编码的伪代码见表4. 其中函数 Extract-Min(Q) 是从 Q 中取出频率最小的节点.

表 4: Huffman 编码的伪代码

```
Proc Huffman(A) //A 为需要编码的字符集
  n = |A|;
  Q = 按频率分布从小到大排序 A
  for i to n do
    z = 新节点
    left(z) = Extract-Min(Q); //取出 Q 中的第一个节点, 并从 Q 中删去
    right(z) = Extract-Min(Q); //取出 Q 中的第一个节点, 并从 Q 中删去
    frequency(z) = frequency(left(z)) + frequency(right(z))
    Insert(Q, z); //向 Q 中按频率顺序插入新节点
  end{for}
  return Extract-Min(Q)
end{Huffman}
```

实现的具体 C++ 代码见附录B. 在 linux 下经 g++ 编译并运行得到图5.



```
zhou@ubuntu: ~/Desktop
File Edit View Search Terminal Help
zhou@ubuntu:~/Desktop$ g++ Huffman.cpp -o huffman
zhou@ubuntu:~/Desktop$ ./huffman
110
a 1001
c 101010
d 10001
e 1111
f 1011
g 101011
h 0101
i 1110
l 01110
m 0011
n 000
o 0010
p 01000
r 01001
s 0110
t 01111
u 10100
x 10000
zhou@ubuntu:~/Desktop$
```

图 5: Huffman 编码 C++ 程序运行结果

■

第五章 动态规划算法

习题 5.1 最大子段和问题: 给定整数序列 a_1, a_2, \dots, a_n , 求该序列形如 $\sum_{k=i}^j a_k$ 的子段和的最大值: $\max\{0, \max_{1 \leq i \leq j \leq n} \sum_{k=i}^j a_k\}$.

1. 已知一个简单算法如下:

```

12 int Maxsum(int n, int a, int& besti, int& bestj)
13 { int sum = 0;
14   for(int i=1; i<=n; i++){
15     int suma = 0;
16     for(int j=i; j<=n; j++){
17       suma + = a[j];
18       if(suma > sum){
19         sum = suma;
20         besti = i;
21         bestj = j; }
22   }
23 }
24 return sum;
25 }
```

试分析该算法的时间复杂性.

2. 试用分治算法解最大子段和问题, 并分析算法的时间复杂性.
3. 试说明最大子段和问题具有最优子结构性质, 并设计一个动态规划算法解最大子段和问题. 分析算法的时间复杂度.
- (提示: 令 $b(j) = \max_{1 \leq i \leq j \leq n} \sum_{k=i}^j a_k, j = 1, 2, \dots, n$)

1. **解:** 该简单算法的执行步数统见表5. 由表5可知该简单算法执行的总步数为:

表 5: 执行步数统计表

行数	s/e	频率	总步数	行数	s/e	频率	总步数
01	0	0	0	08	1	$n(n-1)/2$	$n(n-1)/2$
02	0	0	0	09	1	$n(n-1)/2$	$n(n-1)/2$
03	1	$n+1$	$n+1$	10	1	$n(n-1)/2$	$n(n-1)/2$
04	1	$n+1$	$n+1$	11	0	0	0
05	1	$n(n+1)/2$	$n(n+1)/2$	12	0	0	0
06	1	$n(n-1)/2$	$n(n-1)/2$	13	0	0	0
07	1	$n(n-1)/2$	$n(n-1)/2$	14	0	0	0

$2 \times (n+1) + n(n+1)/2 + 5 \times n(n-1)/2 = 6n^2 + 4$ 所以该算法时间复杂度为 $O(n^2)$.

2. 解最大子段的分治算法: 对给定序列 $a[1:n]$ 对半划分为 $a[1:\lfloor n/2 \rfloor]$, $a[\lfloor n/2 \rfloor + 1:n]$, 并分别求出这两段的最大子段. 则原序列的最大子段有以下可能:

- 原序列的最大子段 = $a[1 : \lfloor n/2 \rfloor]$ 最大子段.
- 原序列的最大子段 = $a[\lfloor n/2 \rfloor + 1 : n]$ 最大子段.
- 原序列的最大子段 = $a[1 : \lfloor n/2 \rfloor]$ 最大子段与 $a[\lfloor n/2 \rfloor + 1 : n]$ 最大子段组合.

基于以上分析, 可得到解最大子段分治算法的伪代码, 见表6.

表 6: 解最大子段分治算法的伪代码

```

Proc MaxSubSum(a, left, right) // a 为整数序列
  if left < right then
    middle =  $\lfloor (\text{left} + \text{right}) / 2 \rfloor$ 
    MaxSumLeft = MaxSubSum(a, left, middle)
    MaxSumRight = MaxSubSum(a, middle+1, right)
    sumleft = 0; maxleft = 0;
    for i from middle to left
      sumleft = sumleft + a(i)
      maxleft = max(sumleft, maxleft)
    end{for}
    sumright = 0; maxright = 0;
    for i from middle+1 to right
      sumright = sumright + a(i)
      maxright = max(sumright, maxright)
    end{for}
    SumCombine = maxleft + maxright
    MaxLightRight = max(MaxSumLeft, MaxSumRight)
    MaxSum = max(SumCombine, MaxLightRight)
    return MaxSum
  else
    MaxSum = max(a(left), 0)
    return MaxSum
  end{Proc}

```

该算法的时间复杂度 $T(n)$ 有递归关系式 $T(n) = 2T(n/2) + O(n), n > 1, T(1) = O(1)$. 满足典型的分治算法递归关系式, 故 $T(n) = O(n \log n)$

3. 设 $S(j) = \max\{a_1 + a_2 + \cdots + a_j, a_2 + a_3 + \cdots + a_j, \cdots, a_{j-1} + a_j, a_j\} = \max_{1 \leq i \leq j} \{\sum_{k=i}^j a_k\}$.
 则子段和的最大值为 $\max\{S(1), S(2), \cdots, S(n)\}$. 由 $S(j)$ 的定义得

$$S(j) = \max_{1 \leq i \leq j-1} \{a_j + \sum_{k=i}^{j-1} a_k, a_j\} = \max\{a_j + S(j-1), a_j\}$$

因此, $S(j)$ 具有最优子结构. 其动态规划最优子结构公式为:

$$S(j) = \max\{S(j-1) + a_j, a_j\}, 1 \leq j \leq n$$

根据以上分析, 可得到解最大子段动态规划算法的伪代码, 见表7. 该算法的主要计算量取决于程序对 i 的 for 循环, 循环体内的计算量为 $O(1)$. 因此该算法的时间复杂度为 $O(n)$.

表 7: 解最大子段动态规划算法的伪代码

```

Proc MaxSum(a) // a 为整数序列
  n = |a|
  maxsum = 0; sum = 0;
  I = 0; J = 0; // 初始化 I, J. I, J 分别为最大子段的始终元素下标
  for i from 1 to n
    if sum > 0 then
      sum = sum + a[i];
    else
      sum = a[i];
      I = i;
    end{if}
    if sum > maxsum then
      maxsum = sum;
      J = i;
    end{if}
  end{for}
  return maxsum, I, J;
end{Proc}

```

■

习题 5.2 (双机调度问题) 用两台处理机 A 和 B 处理个作业. 设第 i 个作业交给机器 A 处理时所需要的时间是 a_i , 若由机器 B 来处理, 则所需要的时间是 b_i . 现在要求每个作业只能由一台机器处理, 每台机器都不能同时处理两个作业. 设计一个动态规划算法, 使得这两台机器处理完这个作业的时间最短 (从任何一台机器开工到最后一台机器停工的总的时间). 以下面的例子说明你的算法:

$$n = 6, (a_1, a_2, a_3, a_4, a_5, a_6) = (2, 5, 7, 10, 5, 2), (b_1, b_2, b_3, b_4, b_5, b_6) = (3, 8, 4, 11, 3, 4)$$

解: 设给定的作业标号为 $S = 1, 2, \dots, n$, $T(k)$ 为前 k 个作业用分给 A, B 机处理所需最小时间, 其中用 A 机处理的作业标号集合为 I , 用 B 机处理的作业标号集合为 J . 显然有 $I + J = S$. 则有

$$T(k) = \max\left\{\sum_{i \in I} a_i, \sum_{j \in J} b_j\right\}$$

$$T(k+1) = \min\left\{\max\left\{a_{k+1} + \sum_{i \in I} a_i, \sum_{j \in J} b_j\right\}, \max\left\{\sum_{i \in I} a_i, b_{k+1} + \sum_{j \in J} b_j\right\}\right\}$$

因此 $T(k)$ 具有最优子结构. 由以上分析可得出双机调度问题的动态规划算法, 其伪代码见表 8.

表 8: 双机调度问题的动态规划算法伪代码

```

Proc Schedule2Machine(a, b)
  Ta = 0; Tb = 0;
  n = |a|;
  Machine(1:n) = 0; //标记作业  $i$  由第 Machine(i) 个机器完成
  for i from 1 to n
    Tai = max{Ta+a[i], Tb};
    Tbi = max{Tb+b[i], Ta};
    if Tai < Tbi then
      Ta = Ta + a[i];
      Machine(i) = 1;
    else
      Tb = Tb + b[i];
      Machine(i) = 2;
    end{if}
  end{for}
  Tmin = max{Ta, Tb};
  return Tmin, Machine;
end{Schedule2Machine}

```

现以 $n = 6$, $a[1:6] = (2, 5, 7, 10, 5, 2)$, $b[1:6] = (3, 8, 4, 11, 3, 4)$ 为例说明该算法执行的每一步, 其中每步循环后, 程序中的各变量值见表9.

表 9: 每步循环后的各变量值

i	Ta _i	Tb _i	Ta	Tb	T=max{Ta,Tb}	Machine
0	0	0	0	0	0	[0,0,0,0,0,0]
1	2	< 3	3	0	3	[1,0,0,0,0,0]
2	7	< 8	7	0	7	[1,1,0,0,0,0]
3	14	> 7	7	4	7	[1,1,2,0,0,0]
4	17	> 15	7	15	15	[1,1,2,2,0,0]
5	15	< 18	12	15	15	[1,1,2,2,1,0]
6	15	< 19	14	15	15	[1,1,2,2,1,1]

最终可以算得最小时间 $T_{\min} = 15$; 各作业所有的机器 Machine = [1,1,2,2,1,1], Machine(i) = 1 表示第 i 个作业由 A 机完成, Machine(i) = 2 表示第 i 个作业由 B 机完成.

■

习题 5.3 考虑下面特殊的整数线性规划问题:

$$\begin{aligned}
 & \max \sum_{i=1}^n c_i x_i \\
 & \sum_{i=1}^n a_i x_i \leq b, x_i \in \{0, 1, 2\}, 1 \leq i \leq n
 \end{aligned}$$

试设计一个解此问题的动态规划算法, 并分析算法的时间复杂度.

解: 设 $p_{2i-1} = p_{2i} = c_i$, $y_{2i-1} = y_{2i} = x_i$, $w_{2i-1} = w_{2i} = a_i$ 则上述问题可转化经典的 0/1 背包问题.

$$\begin{aligned} \max \quad & \sum_{i=1}^n p_i y_i \\ \text{s.t.} \quad & \sum_{i=1}^{2n} w_i y_i \leq b, y_i \in \{0, 1\}, 1 \leq i \leq 2n \end{aligned}$$

用经典的 0/1 背包问题的动态规划算法就可解此问题. 经典的 0/1 背包问题时间复杂性是 $O(2^n)$, 因此该问题的时间复杂性为 $O(2^{2n})$ 即 $O(4^n)$ ■

习题 5.4 可靠性设计: 一个系统由 n 级设备串联而成, 为了增强可靠性, 每级都可能并联了不止一台同样的设备. 假设第 i 级设备 D_i 用了 m_i 台, 该级设备的可靠性是 $g_i(m_i)$, 则这个系统的可靠性是 $\prod g_i(m_i)$. 一般来说 $g_i(m_i)$ 都是递增函数, 所以每级用的设备越多系统的可靠性越高. 但是设备都是有成本的, 假定设备 D_i 的成本是 c_i , 设计该系统允许的投资不超过 c , 那么, 该如何设计该系统 (即各级采用多少设备) 使得这个系统的可靠性最高. 试设计一个动态规划算法求解可靠性设计问题.

解: 由于每一台设备至少需要一台, 故第 i 台设备至多可配置台数为

$$M_i = \left\lfloor \frac{c - \sum_{j=1}^n c_j}{c_i} \right\rfloor + 1$$

因此该问题的数学规划描述为

$$\begin{aligned} \max \quad & \prod_{i=1}^n g_i(m_i) \\ \text{s.t.} \quad & \sum_{i=1}^n m_i c_i \leq c, m_i \in \{1, 2, \dots, M_i\} \end{aligned}$$

其向后递推关系为

$$f_i(X) = \max_{1 \leq m_i \leq M_i} \{g_i(m_i) f_{i-1}(X - c_i m_i)\}$$

由 $f_0 \Rightarrow f_1 \Rightarrow \dots \Rightarrow f_n$, 回溯求出 m_1, m_2, \dots, m_n . ■

附录

A 两种排序算法的 C++ 程序及其时间复杂性测试程序

```

1  #include<iostream>
2  #include<cstdlib>
3  #include<ctime>
4  #include<time.h>
5  using namespace std;
6  template<class T>void MergeSort(T a[],int left,int right);
7  template<class T>void Merge(T c[],T d[], int l,int m,int r);
8  template<class T>void Copy(T a[],T b[],int l,int r);
9  template<class T>void QuickSort(T a[],int p,int r);
10 template<class T>int Partition(T a[],int p,int r);
11 int main()
12 {
13     int a[1000];
14     clock_t start,finish;
15     std::cout<<"====MergeSort===="<<endl;
16     std::cout<<"array size "<<" | "<<" time(us)"<<endl;
17     for(int n =1;n<11;n++){
18         start = clock();
19         for(int k=0;k<100000;k++){
20             srand(time(NULL));
21             for(int i=0;i<n*100;i++){
22                 a[i] = rand();
23             }
24             MergeSort(a,0,n-1);
25         }
26         finish = clock();
27         std::cout<<"          "<<n*100<<"          "<<(float(finish-start)
                /100000)<<endl;
28     }
29     ////////////////////////////////////////////
30     std::cout<<"====QuickSort===="<<endl;
31     std::cout<<"array size "<<" | "<<" time"<<endl;
32     for(int n =1;n<11;n++){
33         start = clock();
34         for(int k=0;k<100000;k++){
35             srand(time(NULL));
36             for(int i=0;i<n*100;i++){
37                 a[i] = rand();
38             }
39             QuickSort(a,0,n-1);
40         }
41         finish = clock();
42         std::cout<<"          "<<n*100<<"          "<<(float(finish-start)
                /100000)<<endl;
43     }

```

```
44         return 0;
45     }
46
47     template<class T>
48     void MergeSort(T a[],int left,int right) //
49     {
50         if(left<right)
51         {
52             int i=(left+right)/2;
53             T *b = new T[left+right];
54             MergeSort(a,left,i);
55             MergeSort(a,i+1,right);
56             Merge(a,b,left,i,right);
57             Copy(a,b,left,right);
58         }
59     }
60
61     template<class T>
62     void Merge(T c[],T d[],int l,int m,int r)
63     {
64         int i=l;
65         int j=m+1;
66         int k=l;
67         while((i<=m)&&(j<=r))
68         {
69             if(c[i]<=c[j])d[k++]=c[i++];
70             else d[k++]=c[j++];
71         }
72         if(i>m)
73         {
74             for(int q=j;q<=r;q++)
75                 d[k++]=c[q];
76         }
77         else
78             for(int q=i;q<=m;q++)
79                 d[k++]=c[q];
80     }
81
82     template<class T>
83     void Copy(T a[],T b[], int l,int r)
84     {
85         for(int i=l;i<=r;i++)
86             a[i]=b[i];
87     }
88
89     //////////////////////////////////////
90     template<class T>
91     void QuickSort(T a[],int p,int r)
92     {
93         if(p<r)
```

```
94     {
95         int q=Partition(a,p,r);
96         QuickSort(a,p,q-1);
97         QuickSort(a,q+1,r);
98     }
99 }
100
101 template<class T>
102 int Partition(T a[],int p,int r)
103 {
104     int i=p,j=r+1;
105     T x=a[p];
106     while(true)
107     {
108         while(a[++i]<x);
109         while(a[--j]>x);
110         if(i>=j)break;
111         Swap(a[i],a[j]);
112     }
113     a[p]=a[j];
114     a[j]=x;
115     return j;
116 }
117
118 template<class T>
119 inline void Swap(T &s,T &t)
120 {
121     T temp=s;
122     s=t;
123     t=temp;
124 }
```

B Huffman 编码 C++ 程序

```
1 #include <iostream>
2 #include <queue>
3 #include <map>
4 #include <climits> // for CHAR_BIT
5 #include <iterator>
6 #include <algorithm>
7
8 const int UniqueSymbols = 1 << CHAR_BIT;
9 const char* SampleString = "this is an example for huffman encoding"
10 ;
11
12 typedef std::vector<bool> HuffCode;
13 typedef std::map<char, HuffCode> HuffCodeMap;
```

```

14 class INode
15 {
16 public:
17     const int f;
18
19     virtual ~INode() {}
20
21 protected:
22     INode(int f) : f(f) {}
23 };
24
25 class InternalNode : public INode
26 {
27 public:
28     INode *const left;
29     INode *const right;
30
31     InternalNode(INode* c0, INode* c1) : INode(c0->f + c1->f), left(
        c0), right(c1) {}
32     ~InternalNode()
33     {
34         delete left;
35         delete right;
36     }
37 };
38
39 class LeafNode : public INode
40 {
41 public:
42     const char c;
43
44     LeafNode(int f, char c) : INode(f), c(c) {}
45 };
46
47 struct NodeCmp
48 {
49     bool operator()(const INode* lhs, const INode* rhs) { return lhs
        ->f > rhs->f; }
50 };
51
52 INode* BuildTree(const int (&frequencies)[UniqueSymbols])
53 {
54     std::priority_queue<INode*, std::vector<INode*>, NodeCmp> trees;
55
56     for (int i = 0; i < UniqueSymbols; ++i)
57     {
58         if(frequencies[i] != 0)
59             trees.push(new LeafNode(frequencies[i], (char)i));
60     }
61     while (trees.size() > 1)

```

```

62     {
63         INode* childR = trees.top();
64         trees.pop();
65
66         INode* childL = trees.top();
67         trees.pop();
68
69         INode* parent = new InternalNode(childR, childL);
70         trees.push(parent);
71     }
72     return trees.top();
73 }
74
75 void GenerateCodes(const INode* node, const HuffCode& prefix,
76 HuffCodeMap& outCodes)
77 {
78     if (const LeafNode* lf = dynamic_cast<const LeafNode*>(node))
79     {
80         outCodes[lf->c] = prefix;
81     }
82     else if (const InternalNode* in = dynamic_cast<const
83             InternalNode*>(node))
84     {
85         HuffCode leftPrefix = prefix;
86         leftPrefix.push_back(false);
87         GenerateCodes(in->left, leftPrefix, outCodes);
88
89         HuffCode rightPrefix = prefix;
90         rightPrefix.push_back(true);
91         GenerateCodes(in->right, rightPrefix, outCodes);
92     }
93 }
94
95 int main()
96 {
97     // Build frequency table
98     int frequencies[UniqueSymbols] = {0};
99     const char* ptr = SampleString;
100     while (*ptr != '\0')
101         ++frequencies[*ptr++];
102
103     INode* root = BuildTree(frequencies);
104
105     HuffCodeMap codes;
106     GenerateCodes(root, HuffCode(), codes);
107     delete root;
108
109     for (HuffCodeMap::const_iterator it = codes.begin(); it != codes
110         .end(); ++it)
111     {

```



```
109         std::cout << it->first << " ";
110         std::copy(it->second.begin(), it->second.end(),
111                 std::ostream_iterator<bool>(std::cout));
112         std::cout << std::endl;
113     }
114     return 0;
115 }
```