第3章

枚举算法

算法设计技术是求解问题的有效策略，也是算法解题的一般性方法，用于解决不同计算领域的多种问题。常用算法设计技术有枚举算法、递推算法、贪心算法、分治算法、动态规划算法、网络流算法、随机算法、近似算法、启发式算法。枚举算法从所有候选答案中搜索正确解，是一种暴力求解算法，也是最常想到和使用的算法。本章介绍蛮力算法和枚举算法，枚举算法的优化方法，排列和子集的生成方法。

视频讲解

3.1 枚举与优化

3.1.1 蛮力算法

蛮力算法是基于问题描述和定义，简单直接地解决问题的方法，又称暴力求解。这里的“力”指的是计算机的计算能力，而不是人的智力。蛮力算法是任何问题都有的简单算法，枚举算法、模拟算法和仿真算法是其常用的方法。例如，计算 $a^n(a>0,n$ 为非负整数)的蛮力算法是直接做乘法 $n-1$ 次，时间复杂度是 $O(n)$，但本书第 2 章学过时间复杂度为 $O(\log n)$ 的快速幂算法。顺序查找是查找问题的蛮力算法，更好的算法是折半查找和分块查找。矩阵相乘的时间复杂度为 $O(n^3)$ 的算法是蛮力算法，本书后面会学到时间复杂度为 $O(n^{\log 7})$ 的算法。

例 1：给定 n 个元素的序列 A，将 A 排序为 $\{a_1,a_2,\cdots,a_n\}$，使 $i<j$ 时 $a_i \leqslant a_j$。

前面学过的插入排序是蛮力算法，后面的直接选择排序算法和冒泡排序算法也是蛮力算法。

直接选择排序算法 SelectSort：

输入：数组 A，n。

输出：数组 A，使 $i<j$ 时 $a_i \leqslant a_j$，$0 \leqslant i<j \leqslant n-1$。

```
1.  for i = 0 to n - 2 do
2.        k = i                                  //选择具有最小排序码的对象
3.       for j = i + 1 to n - 1 do
```

```
4.            if (A[j] < A[k]) then k = j            //当前具有最小排序码的对象
5.        if ( k <> i ) then Swap (A[i],A[k])        //对换到第 i 个位置
6.   return A
```

直接选择排序的比较次数为 $\sum_{i=0}^{n-2}(n-i-1)=((1+n-1)/2)\cdot(n-1)=(n(n-1))/2$，交换次数最多为 $n-1$ 次，因此时间复杂度为 $\Theta(n^2)$。不管序列是正序还是反序，时间复杂度都为 $\Theta(n^2)$。

冒泡排序算法 BubbleSort：

输入：数组 A，n。

输出：数组 A，使 $i<j$ 时 $a_i \leqslant a_j$，$0 \leqslant i<j \leqslant n-1$。

```
1.   for i = 1 to n - 1 do
2.        for j = 1 to n - i do
3.            if ( a[j - 1] > a[j] ) then Swap ( a[j],a[j - 1] )   //交换
4.   return a
```

冒泡排序的比较次数为 $\sum_{i=1}^{n-1}(n-i)=((1+n-1)/2)\cdot(n-1)=(n(n-1))/2$，交换次数最多为 $\sum_{i=1}^{n-1}(n-i)=((1+n-1)/2)\cdot(n-1)=(n(n-1))/2$，因此时间复杂度为 $\Theta(n^2)$。但如果序列为正序，实际上不需要交换。据此进行改进，设置 exchange 变量，如果没有交换，则说明已经排好序，算法终止。

改进的冒泡排序算法：

输入：数组 A，n。

输出：数组 A，使 $i<j$ 时 $a_i \leqslant a_j$，$0 \leqslant i<j \leqslant n-1$。

```
1.   i = 1
2.   exchange = 1
3.   while(exchange and i < n) do
4.        exchange = 0
5.        for j = 1 to n - i do
6.            if (A[j - 1] > A[j]) then
7.                Swap (A[j],A[j - 1])            //元素交换
8.                exchange = 1
9.        i = i + 1
10.  return A
```

当序列为正序时，需进行 1 次循环，$n-1$ 次比较，不用交换，算法结束，因此时间复杂度为 $\Omega(n)$。当序列为反序时，算法的时间复杂度为 $O(n^2)$。

例 2：给定长度为 n 的字符串 A，在其中查找是否有长度为 m 的字符串 B。

字符串匹配算法：

输入：字符串 A 和 B。

输出：true or false。

```
1.   for s = 0 to n - m do
```

```
2.       if (B[1:m] == A[s+1:s+m])            //子串相等,B[1:m]=B
3.           then return true
4.   return false
```

算法循环 $n-m+1$ 次，每次循环需要比较 m 次，因此蛮力算法的时间复杂度为$(n-m+1)m=O(nm)$。而字符串匹配的 KMP(由 D. E. Knuth、J. H. Morris 和 V. R. Pratt 同时发现，因此以其名字首字母命名)算法的时间复杂度为 $O(n+m)$。

蛮力算法的优点是：简单、应用广泛；对于一些重要问题具有实用性，且不必限制实例规模；对于规模较小的实例，比实现复杂算法的代价低；一般用于解决小规模实例。其缺点是：很少有高效的算法，不像其他算法设计得那么巧妙、有效。

3.1.2 枚举算法概述

枚举(穷举)算法属于简单的蛮力方法，从问题所有可能的解集中一一枚举各元素，用题目中给定的约束条件判断是否符合条件，查找具有特殊属性(目标函数最优)的元素。满足约束条件的解称为问题的可行解，所有可行解构成问题的解空间，使目标函数最优的可行解称为问题的最优解。枚举算法一般涉及排列、组合或子集等对象。

枚举算法的一般步骤如下。

(1) 找出解的表示形式，构造问题的所有可能解(解空间)。

(2) 逐个评价解，选出满足约束条件的元素。

(3) 查找具有特殊属性的元素。

例 3：旅行商问题，给定 n 个城市相互间的距离，设置一条经过所有城市一次且仅有一次，然后回到起始城市的最短路径。

问题分析：如图 3-1 所示，从城市 a 出发，经过 b、c、d 这 3 个城市再回到 a 的回路，如表 3-1 所示。$n=4$，枚举 6 条路径，得到最短路程为 a→b→c→d→a 和 a→d→c→b→a，路径长度为 17。

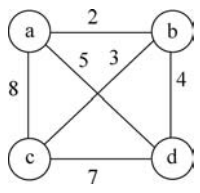

图 3-1 旅行商问题实例

实际上每条回路，除了起点和终点，刚好是 b、c、d 这 3 个城市的一个排列。因此，n 个城市的解空间有$(n-1)!$条回路。算法首先找出$(n-1)!$条回路，然后计算每条回路的长度和，从这些长度和中选取具有最小长度和的路径。

表 3-1 旅行商问题求解示例

回　路	路　程	回　路	路　程
a→b→c→d→a	2+3+7+5=17	a→c→d→b→a	8+7+4+2=21
a→b→d→c→a	2+4+7+8=21	a→d→b→c→a	5+4+3+8=20
a→c→b→d→a	8+3+4+5=20	a→d→c→b→a	5+7+3+2=17

旅行商问题枚举算法：

输入：图 G 的权值矩阵，n。

输出：最短路径长度。

```
1.  L=0
2.  for 每条回路 do
```

```
3.        计算回路的长度和 p
4.        if (p < L) then L = p
5.   return L
```

计算每条回路的长度和需要 n 次加法，共有$(n-1)!$条回路，因此算法的时间复杂度为$O(n(n-1)!)=O(n!)$。

枚举算法从本质上说是一种搜索算法，需要确定枚举变量及范围，确定约束条件，不要漏项或枚举不必要的项。如果问题的解为 $(A_1,A_2,\cdots,A_n)$，解变量 A_i 为枚举变量，$\{a_{i1},a_{i2},\cdots,a_{iK}\}$ 为 A_i 的取值范围，则问题的解空间为 $\Pi|A_i|$。旅行商实例中解变量为(A_1,A_2,A_3)，$|A_1|=3$，$|A_2|=2$，$|A_3|=1$，因此问题有 6 种可能解，最小的解元素即为问题的解。

例 4：0-1 背包问题。给定 n 个物品和 1 个背包，物品 i 有重量 w_i 和价值 v_i，背包容量为 W。如果物品不允许切分，求装入背包物品的最大价值和。问题的实例如表 3-2 所示。

表 3-2　0-1 背包问题实例($W=16$)

物　品	重　量	价值/元
1	2	20
2	5	30
3	10	50
4	5	10

背包中装入的物品是 n 个物品的 1 个子集，因此解空间为 2^n 个子集。算法首先找出 2^n 个子集；然后计算每个子集的价值和，并检查其重量和是否超过背包容量；最后在符合约束条件的价值和中找出最大值。$n=4$，除空集外 15 个子集的结果如表 3-3 所示，最大价值为 80，对应物品 2 和 3。

表 3-3　0-1 背包问题求解结果

子集	{1}	{2}	{3}	{4}	{1,2}	{1,3}	{1,4}	{2,3}	{2,4}	{3,4}	{1,2,3}	{1,2,4}	{1,3,4}	{2,3,4}	{1,2,3,4}
总重量	2	5	10	5	7	12	7	15	10	15	17	12	17	20	22
总价值	20	30	50	10	50	70	30	80	40	60	不可行	60	不可行	不可行	不可行

0-1 背包问题枚举算法：

输入：n 个物品的重量 w_i 和价值 v_i，$1\leqslant i\leqslant n$，背包容量 W。

输出：最大价值和。

```
1.   V = 0
2.   for n 个物品的每个子集 do
3.        if(子集的重量和<= W)then
4.             if (子集的价值和 v > V) then V = v
5.   return V
```

算法计算每个子集的总重量和总价值的时间复杂度为 $O(n)$，共有 2^n 个子集，因此算法的时间复杂度为 $O(n2^n)$。

3.1.3 枚举优化

常用的枚举优化方法如下。

(1) 减少枚举变量。使用枚举算法前,首先考虑解元素之间的关联,将一些非枚举不可的解元素列为枚举变量,其他元素通过计算得出解元素的可能值。

(2) 减少枚举变量的值域(取值范围)。

(3) 优化算法的模型和数据结构。

例 5:巧妙填数。将 1～9 这 9 个数字填入 9 个空格中,每一横行的 3 个数字组成一个三位数。如果要使第 2 行的三位数是第 1 行的 2 倍,第 3 行的三位数是第 1 行的 3 倍,应怎样填数?算法实例如表 3-4 所示。

表 3-4 巧妙填数实例

1	9	2
3	8	4
5	7	6

问题分析:问题的解是每格 1 个数字,可以设置 9 个变量:A_1～A_9,第 1 个格有 9 个数字可选,$|A_1|=9$;第 2 格有 8 个数字可选,$|A_2|=8$;…;第 9 格只有 1 个数字可选,$|A_9|=1$。共有 9!=362 880 种方案,在这些方案中符合条件的即为问题的解。

考虑最后一行最大为 987,显然第 1 行的三位数不会超过 987/3,因此第 1 格只有{1,2,3}3 个数字可选。实际上只要确定第 1 行的数就可以根据倍数条件计算出其他两行的数,然后检查是否 9 个数字都不重复,如果满足则得到问题的解。这样仅需设置 3 个枚举变量,$|A_1|=3$、$|A_2|=8$、$|A_2|=7$,共有 168 个方案。算法通过减少枚举变量和枚举变量的值域,减少了计算量。

例 6:中国古代数学家张丘建在他的《算经》中提出了著名的"百钱百鸡问题"。鸡翁一,值钱五;鸡母一,值钱三;鸡雏三,值钱一;百钱买百鸡,翁、母、雏各几何?

问题分析:设 x,y,z 分别为公鸡、母鸡、小鸡的数量,则 $x+y+z=100$ 且 $5x+3y+z/3=100$。x 的取值范围为 1～20;y 的取值范围为 1～34,z 的取值范围为 1～100,因此,最多共有 $20\times34\times100$ 种方案。

实际上,公鸡 x 和母鸡 y 确定后,小鸡 $z=100-x-y$,则共有 $20\times34=680$ 种方案。约束条件为:$5x+3y+z/3=100$,$z \bmod 3=0$。这样,算法通过减少枚举变量,减少了计算量。

百钱百鸡问题枚举算法:

```
1.  for x = 1 to 20 do
2.      for y = 1 to 34 do
3.          z = 100 - x - y
4.          if (z mod 3 == 0 and 100 == 5 * x + 3 * y + z/3) then
5.              print x,y,z
```

例 7:输入正整数 n,顺序输出形如 $abcde/fghij=n$ 的表达式。a～j 为 0～9 的一个

排列。例如，79 546/01 283＝62。

问题分析：设置 10 个变量 $a\sim j$，$a\sim j$ 为 0～9 的一个排列，所有排列数为 10!＝3 628 800。

实际上，n 是给定的，通过枚举分母 $fg\,hij$，可以算出 $ab\,cde$，然后判断所有数字是否不同，这样减少了枚举变量，方案数减少为 $10\times9\times8\times7\times6$。另外，还可以减少枚举变量取值范围，如果 $62\times fg\,hij>100\,000$(超出 5 位数)，可以直接删除，减少计算。

例 8：分数拆分。输入正整数 k，找所有正整数 $x\geqslant y$，使 $1/k=1/x+1/y$。例如，1/2＝1/6＋1/3，1/2＝1/4＋1/4。

问题分析：因为 $x\geqslant y$，所以 $1/k=1/x+1/y\leqslant2/y$，$y\leqslant2k$。在 $2k$ 范围内枚举 y，通过 $1/k=1/x+1/y$，可以计算 x，这样减少了枚举变量 x 和 y 的取值范围。

例 9：约瑟夫问题。现在有 $2k$ 个人围坐成一个圈，其中前 k 个人是好人，后 k 个人是坏人。从第 1 个人开始循环报数，报数为 m 的人被处决，之后的人接着从 1 开始报数。设计一个最小的 m，使 k 轮处决之后留下的 k 个人都是好人。

问题分析：第 1 种方法可以使用模拟法，根据问题的描述进行求解。使用 now＋m－1 模拟每次报数为 m 被处决人的编号。now＋m－1 mod bad 通过模运算，保证编号≤总人数。如果报数 m 的编号＞k，则是坏人，算法继续运行；否则是好人，算法终止，开始 m＋1 的测试。最终 bad＝k，处决了 k 个坏人，得到问题的解。

约瑟夫问题模拟算法：

输入：k。

输出：m。

```
1.  m = 0
2.  while (1) do
3.      m ++                                    //枚举 m
4.      bad = 2 * k                             //总人数
5.      now = 0                                 //起始编号
6.      while(1) do
7.          now = (now + m - 1) % bad + 1       //编号
8.          if(now > k) then bad -- now --      //坏人报数为 m 被处决,总人数减少
9.          else break
10.     if(bad == k) then
11.         sign[k] = m
12.         break
```

第 1 个 while 循环 m 次，第 2 个 while 最多循环 k 次，且 $n=2k$，因此算法的时间复杂度为 $O(nm)$。问题还可以通过循环链表模拟实现。

第 2 种方法使用枚举算法。考虑最后处决的坏人，如果倒数第 2 个处决的坏人在其后面，则 $m=x(k+1)$ 报数为最后处决的坏人；如果倒数第 2 个处决的坏人在其前面，则 $m=1+x(k+1)$报数为最后处决的坏人。x 表示转过 x 圈回到最后删除的坏人，如图 3-2 所示。这样操作优化了枚举变量的取值范围，枚举次数大大减少。

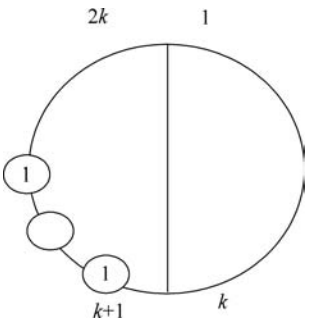

图 3-2　约瑟夫问题

枚举算法虽然简单、容易编程、容易分析复杂度和证明正确性，但速度慢，仅对很小的实例有合理的运行时间，在许多规模较大的实例中有更好的替代算法。枚举算法要求所解问题的可能解是有限的、固定的，不会产生组合爆炸、容易枚举的。在某些实例中，枚举算法是唯一的解决方法。枚举算法多用于决策类问题，因为这类问题不易进行问题的分解，只能整体来求解。

本节思考题

计算 $a^n \bmod m$，正整数 $a>1$，正整数 $n>0$。当 n 很大时，如何处理 a^n 的巨大的数量级问题？


视频讲解

3.2 组合与排列

枚举算法一般涉及排列、组合或子集等对象。本节主要讲述子集与排列的生成方法。

3.2.1 排列

给定 n 个元素的数组 P，生成 P 的全排列，可以按字典序、最小变化等要求输出全排列。

1. 按字典序输出全排列

字典序输出全排列算法：

输入：P，n。

输出：按字典序输出 P 的所有排列。

```
call permutation (n,A,0,P)
permutation (n,A,cur,P) {
1.   if(cur == n) then                              //递归边界,输出排列
2.        print A
3.   else for i = 0 to n - 1 do
4.             ok = 1                               //初值未使用过
5.             for j = 0 to cur - 1 do              //选择前面没有使用过的元素
6.                  if (A[j] == P[i]) then ok = 0   //i 在前面使用过,不能再使用
7.             if(ok) then
8.                  A[cur] = P[i]
9.                  permutation (n,A,cur + 1,P)
10. }
```

按字典序输出全排列的示例如图 3-3 所示。首先 cur=0，$A[0]=P[0]=1$，然后 cur=1 进入第 1 层。因为 $A[0]=P[0]=1$，1 已经被使用过，所以这里选择 $A[1]=P[1]=3$，然后 cur=2 进入第 2 层。因为 1 和 3 已经被使用过，所以这里选择 $A[2]=P[2]=5$，然后 cur=3 进入第 3 层。因为 1、3 和 5 已经被使用过，所以选择 $A[3]=P[3]=9$，然后 cur=4 进入第 4 层。因为 cur=n，所以输出排列 1359。

回到第 2 层，选择 $A[2]=P[3]=9$，然后 cur=3 进入第 3 层。因为 1、3 和 9 已经被使用过，所以选择 $A[3]=P[2]=5$，然后 cur=4 进入第 4 层。因为 cur=n，所以输出排

列 1395。

回到第 1 层，选择 $A[1]=P[2]=5$，然后 cur=2 进入第 2 层。因为 1 和 5 已经被使用过，所以选择 $A[2]=P[1]=3$，然后 cur=3 进入第 3 层。因为 1、3 和 5 已经被使用过，所以选择 $A[3]=P[3]=9$，然后 cur=4 进入第 4 层。因为 cur=n，输出排列 1539。回到第 2 层，选择 $A[2]=P[3]=9$，第 3 层选择 $A[3]=P[1]=3$，第 4 层输出 1593。同理输出 1935 和 1953。

回到第 0 层，cur=0，选择 $A[0]=P[1]=3$，进入第 1 层选择 $A[1]=P[0]=1$，进入第 2 层选择 $A[2]=P[2]=5$，进入第 3 层选择 $A[3]=P[3]=9$，进入第 4 层输出 3159……

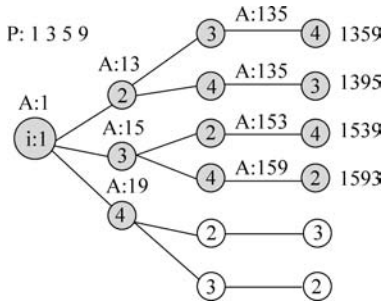

图 3-3　排列示例

计算排列的每位元素时，需要检查该元素是否已经被使用过，因此每个排列的检查量是 $1+2+\cdots+n-1=O(n^2)$。n 个元素的排列数为 $n!$，因此算法的时间复杂度为 $O((n+2)!)$。

2. 按照最小变化输出全排列

Johnson-Trotter 算法引入可移动元素的概念，按照最小变化输出全排列。设排列 $p=\overleftarrow{p_1}\ \overleftarrow{p_2}\ \overleftarrow{p_3}\cdots\overleftarrow{p_n}$，元素 p_k 的箭头指向的相邻元素小于 p_k，则 p_k 是可移动元素。

Johnson-Trotter 算法：

输入：n。

输出：按最小变化输出$\{1,2,\cdots,n\}$的所有排列。

```
1.  初始化排列 p = 1← 2← 3← … n←
2.  while 存在移动元素 k do
3.      求 p 中最大的可移动元素 k
4.      p = p 中元素 k 和箭头指向的相邻元素互换
5.      p = p 中调转所有大于 k 的元素的方向
6.      print p
```

当 $n=3$ 时，从初始排列 $p=\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3}$ 开始，查找到可移动元素 2 和 3，选择最大可移动元素 3 和箭头指向的相邻元素 2 互换，得到 $\overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2}$。同理，下一步得到 $\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2}$。然后选择可移动元素 2 和 1 互换，并将 3 反向，得到 $\overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1}$……最后输出的全排列为$\{\overleftarrow{1}\ \overleftarrow{2}\ \overleftarrow{3},\overleftarrow{1}\ \overleftarrow{3}\ \overleftarrow{2},\overleftarrow{3}\ \overleftarrow{1}\ \overleftarrow{2},\overrightarrow{3}\ \overleftarrow{2}\ \overleftarrow{1},\overleftarrow{2}\ \overleftarrow{3}\ \overleftarrow{1},\overleftarrow{2}\ \overleftarrow{1}\ \overleftarrow{3}\}$。

语句 3、4 和 5 的时间复杂度为 $O(n)$，n 个元素的排列数为 $n!$，需要循环 $n!$ 次，因此算法的时间复杂度为 $O((n+1)!)$。

3. 库函数输出全排列

C++的 STL 中提供库函数 next_permutation，可以使用下面代码实现全排列。

```
1.  int a[n];
2.  /* 输入数组 a */
3.  sort(a,a+n);                                    //必须先排序
4.  do
5.      {//在这里进行相应操作}
6.  while(next_permutation(a,a+n));
```

3.2.2 子集

给定一个集合，枚举其所有子集，常用的方法有增量构造法、位向量法和二进制法。

1. 增量构造法

增量构造法一次选出一个元素放到集合中。$A=\{1,5,9\}$，其子集构造过程如图 3-4 所示。增量构造前需要定序，使集合中元素编号从小到大排序，避免同一集合输出两次。例如，{1,2}和{2,1}。增量构造法伪代码如下。

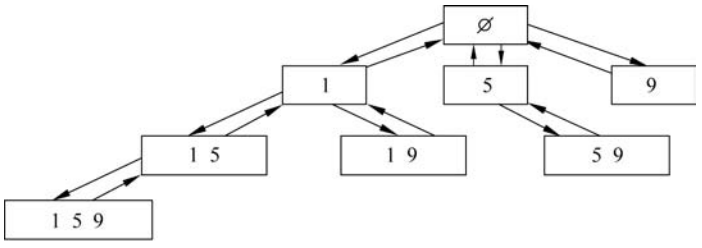

图 3-4 增量构造法示例

输入：A，n。

输出：A 的所有子集。

```
call subset (n,B,0)
subset(n,B,cur) {
1.  for i = 0 to cur - 1 do
2.      print  A[B[i]]                              //打印当前集合
3.  if (cur > 0) then s = B[cur - 1] + 1
4.  else s = 0
5.  for i = s to n - 1 do                           //向下递归
6.      B[cur] = i
7.      subset(n,B,cur + 1)
8.  }
```

n 个元素的集合有 2^n 个子集，每次调用的时间为 $O(n)$，因此算法的时间复杂度为 $O(n2^n)$。

2. 位向量法

位向量法构造一个位向量 $\boldsymbol{b}$，而不是直接构造子集 A 本身。向量 $\boldsymbol{b}$ 有 0 和 1 两种选择，代表子集中是否选择该元素，当所有元素是否被选择确定后构成一个子集。位向量法伪代码如下。

输入：A，n。

输出：A 的所有子集。

```
call subset(n,B,0)     A={1,5,9}      B={0,0, 0)
subset(n,B,cur) {
1.   if(cur==n) then
2.        for i=0 to cur-1 do
3.            if(B[i]) then print A[i]              //打印当前集合
4.   else B[cur]=1                                  //选第 cur 个元素
5.         subset(n,B,cur+1)
6.         B[cur]=0                                 //不选第 cur 个元素
7.         subset(n,B,cur+1)
8.   }
```

位向量法的判定树是一棵完全二叉树，在判定树中叶子节点为一个子集，中间节点不构成子集。生成节点数为 $O(2^{n+1})$，输出每个子集的时间为 $O(n)$，因此时间复杂度为 $O(n2^{n+1})$。位向量法的任何节点都是一个子集，因此位向量法生成的节点多，效率低。

3. 二进制法

二进制法用二进制表示一个子集，从右向左，第 i 位表示元素 i 是否在集合中。例如，给定数组 $A=\{1,5,9\}$，二进制 011 表示选择低位的 1 和 5，未选择高位的 9，因此对应子集为{1,5}。

输入：A，n。

输出：A 的所有子集。

```
1.   for j=0 to (1<<n)-1 do          //(1<<n)-1 表示 2^n-1,正好是 2^n 个子集
2.        call subset(n,j)           //从小到大调用 000 001 010 011…
subset(n,s) {                       //打印子集 S,传入一个十进制数,代表一个子集
1.   for i=0 to n-1 do
2.        if ( s and (1<<i) ) then   //判断 s 的第 i 位是否为 1
3.            print A[i]
4.   }
```

主程序调用子程序的次数为 $O(2^n)$，子程序的运行时间为 $O(n)$，因此二进制法的运行时间为 $O(n2^n)$。

使用二进制法可以实现如下集合的操作。

(1) 交集 $S\cap T$：位与运算 (S AND T)。

(2) 并集 $S\cup T$：位或运算(S OR T)。

(3) 对称差集$(S\cup T)-(S\cap T)$：位异或运算(S XOR T)。

例如，$A=\{1,5,9,10,11\}$，则 10110 表示{5,9,11}，01100 表示{9,10}。位与运算得到 00100 表示{9}，正好是两者的交集。位或运算得到 11110 表示{5,9,10,11}，正好是两者的并集。位异或运算得到 11010 表示{5,10,11}，正好是两者的对称差集。

本节思考题

1. 如何实现集合的操作(插入、删除和查找)?

2. 并查集的实现方式和复杂度?

本章习题

1. 编程实现排列算法(POJ 1256、POJ 1147)。

2. 编程实现尺取法算法(POJ 3061)。

3. 编程实现枚举算法(POJ 2785、3977、2549)。

4. 编程实现并查集算法(POJ 2492)。

5. 编程实现深度优先搜索(DFS)算法(POJ 1562、POJ 1753)。

6. 编程实现广度优先搜索(BFS)算法(POJ 2935、POJ 1465)。

7. 简述常用枚举算法的优化方法。

8. $O(n^2)$时间复杂度的排序算法有哪些?

9. 人有 3 个生理周期:体力周期、情感周期和智力周期。它们的周期长度分别是 23 天、28 天和 33 天。每个周期中有一天是高峰。在高峰这天,在相应的方面会表现出色。但是,3 个周期的高峰通常不会是同一天。对于每个周期,给定当年某一天是高峰期,计算当年 3 个高峰同时出现的时间。

10. 给定正整数 a,b,c,求不定方程 $ax+by=c$ 的所有非负整数解的组数。

11. 大于 1 的正整数 n 可以分解为:$n=x_1\times x_2\times\cdots\times x_m$。例如,当 $n=12$ 时,共有 8 种不同的分解式:12=12;12=6×2;12=4×3;12=3×4;12=3×2×2;12=2×6;12=2×3×2;12=2×2×3。对于给定的正整数 n,共有多少种不同的分解式?

12. 有重复元素的排列问题:给定 n 个元素,元素值可能相同,请列出所有不同的排列。

13. 给定 n 个物品的重量,给定常数 k,从中选择 3 个物品,使其重量之和与 k 的差最小。

14. 给定 n 个物品的重量,给定一个背包和背包的载重量,编写算法,使放入背包的物品重量之和最大。

15. 有 1000 桶酒,其中 1 桶有毒。而一旦吃了,毒性会在 1 周内发作。现在用小老鼠做实验,要在 1 周时间找出那桶毒酒,最少需要多少老鼠?

16. Latin 方问题:给定 4×4 棋盘,每个方格可填入 1、2、3 或 4,使每个数字在每行、每列恰好都出现一次,请给出安置方案。

17. 鸡兔同笼:一个笼子里有若干鸡和兔子,有 n 个头、m 只脚,则笼子里有多少只鸡,多少只兔子?

18. 完美立方:$a^3=b^3+c^3+d^3$,$1<a,b,c,d\leqslant n$。给定正整数 n,求满足要求的四元组。

19. 折半枚举:给定 n 个整数的 4 个序列 A、B、C 和 D,从 4 个序列中各取 1 个数,使其和等于 0,求这样的组合的个数。如果一个数列中有多个相同数字时,作为不同数字看待。