装 订 线

“工大出版社杯”第十八届西北工业大学数学

建模竞赛暨全国大学生数学建模竞赛选拔赛题目

A题

|  |  |  |
| --- | --- | --- |
| 密封号 |  | 2017年5月1日 |

剪 切 线

|  |  |  |
| --- | --- | --- |
| 密封号 |  | 2017年5月1日 |

自动化 学院 第 79 队

|  |  |  |  |
| --- | --- | --- | --- |
|  | 队员1 | 队员2 | 队员3 |
| 姓名 | 顾铭 | 瞿铸枫 | 张教福 |
| 班级 | 09091501 | 09091501 | 09091501 |

摘 要

分酒问题即在已经知道瓶子数与各瓶容量，并给定各个瓶子装酒量的初始状态和目标状态的情况下，求解不借助其他工具实现这两个状态转化的最短途径。在分酒问题规模较小时，我们可以手工实现。但当问题规模扩大，我们需要建立模型通过计算机求解。

本文通过建立模型，将求解分酒问题转化为求解有向无权图的最短路径问题，采用广度优先算法的思想编写程序，遍历求解输出所有的最短路径。

针对问题一，先建立了分酒问题与有向无权图求最短路径问题的对应关系，建立解题模型。然后编写程序，从初始状态开始，运用广度优先搜索算法的思想，求出了所有可由初态经过n步倒酒实现的状态，并记录所有状态之间的转换关系。将这些中间状态作为图的节点，将可转化关系作为节点间的有向边，所有边的权值设为1，使其转化成一张有向无权图。然后，先后尝试使用Dijkstra算法、Floyd算法和改进后的广度优先算法，求解初态到末态的最短路径。若求出最短路径则可达，否则不可达。

其中，我们提出了一种新的求解有向无权图最短路径且输出所有最短路径的算法，我们称之为“层次树”法。该方法可以同时求出最短路径长及所有最短路径，总时间复杂度为O(n2)，可以接受。

问题二，在问题一的基础上扩大了规模，并且可随机给定初始状态和目标状态，但是问题的本质未发生变化。我们采用同样的模型求解，针对问题规模与求解时空复杂度的扩大，使我们进一步优化了算法，提出了“层次树”法。

问题三，题目要求设计最少完成步数不少于13步的实例并给出详细的实现步骤，我们通过选择适当的问题规模并利用已有的模型与算法求解出了几组结果。

最后我们对模型与算法的优缺点进行了客观的评价并将分酒问题在实际生活中的应用进行了推广。

关键词：分酒问题 有向无权图 最短路径 广度优先 “层次树”

目录

[1. 问题重述 1](#_Toc15492)

[2． 符号及名词说明 2](#_Toc8184)

[3． 问题分析 3](#_Toc24034)

[3.1 问题一 3](#_Toc6208)

[3.2 问题二 3](#_Toc17922)

[3.3 问题三 3](#_Toc15004)

[4． 模型建立 4](#_Toc26075)

[4.1 一般模型：求解有向带权图的最短路径问题 4](#_Toc230)

[4.1 模型具体化：分酒问题转化为有向无权图的最短路径问题 5](#_Toc29084)

[5． 求解结果 6](#_Toc20836)

[5.1问题一求解结果： 6](#_Toc30205)

[5.2问题二求解结果： 7](#_Toc28500)

[5.2.1模型分析： 8](#_Toc29982)

[5.2.2算法分析： 8](#_Toc22719)

[5.3问题三求解结果： 8](#_Toc12172)

[6． 算法实现与分析 10](#_Toc17418)

[6.1 算法实现 10](#_Toc8723)

[6.1.1 找出所有可能的装酒状态 10](#_Toc17114)

[6.1.2 建立“层次树”T 13](#_Toc18319)

[6.1.3 遍历查找所有最短路径 15](#_Toc32068)

[6.2 算法复杂度分析 16](#_Toc20326)

[6.3 编程实现： 16](#_Toc10116)

[6.4 算法改进过程： 17](#_Toc9433)

[7． 模型评价 18](#_Toc4093)

[7.1模型优缺点分析 18](#_Toc2203)

[7.2 模型推广 19](#_Toc1619)

[8． 参考文献 19](#_Toc21859)

[9． 附录 20](#_Toc21182)

[9.1 算法的C++代码实现： 20](#_Toc4477)

[9.2 长度为13的最短路径： 29](#_Toc3457)

[9.3 长度为17的最短路径： 30](#_Toc6816)

# 问题重述

分酒问题是一个十分著名的智力问题。该问题是：有一只装满8斤酒的瓶子和两只分别装5斤和3斤酒的空瓶，如何才能将这8斤酒分成两等份。这个问题规模小，手工就可以完成。当你学习了这个问题后，是否考虑过更大规模和一般性问题呢？如果规模扩大，手工无法完成，如何设计算法和建立模型求解呢？对一个一般性的问题又该如何建立模型和设计算法进行求解呢？请你完成下面问题：

问题1 现有一只装满12斤酒的瓶子和三只分别装10斤、6斤和3斤酒的空瓶，如何才能将这12斤酒分成三等份。如果进行四等份呢，结果如何？如果4个瓶子分别要求装5斤、3斤、2斤、2斤，又能否实现？试建立数学模型并设计算法，求最少经过多少步操作完成，且有多少种方式可采用最少步数完成。要求对实现方式给出详细操作步骤。

问题2 一般问题:设有个瓶子，每个瓶子最多装酒数量用向量表示为。现在初始各瓶子装酒为。现要实现将各瓶子装酒为。要求不凭借任何其它工具，问能否实现？若能实现，给出实现的方法，并给出充分理由说明是否是最少步数。并对你所使用的模型和算法进行分析说明。

问题3 你能否自己设计一个实例，要求最少完成步数不少于13步。给出从初始状态到目标状态的详细实现步骤。

# 符号及名词说明

|  |  |
| --- | --- |
| **符号或名词** | **说明** |
| 有向无权图 | 一张图由点，及连接各点的边组成，若各边有固定走向，且没有权重，则称为有向无权图，如下图：  QQ截图  图 1 有向无权图举例 |
| 有序实数对  (a1,a2,a3,…,an) | 一个有序实数对(a1,a2,a3,…,an)对应一个装酒状态(同样对应有向图中一个节点及其编号），a1到an代表各瓶中装酒量，如（8,5,3）代表1号瓶装8L酒，2号装5L，3号装3L，下标表示瓶子编号。 |
| 序偶  <b1，b2> | b1，b2代表某个装酒状态的编号，<b1，b2>代表编号为b1的状态，经一步倒酒可以转化为编号为b2的状态(同样对应有向图中一条有向边），b1、b2顺序不可颠倒，b1称为b2的前驱，b2称为b1的后继。 |

# 问题分析

分酒问题，即是在已经知道瓶子数与各瓶容量的情况下，给定各个瓶子装酒量的初始状态和目标状态，求解：

1. 能否不借助其他工具，从初态转化为末态；
2. 如能转化，则求解最少分酒步数；
3. 求解所有可满足最少步数的分酒过程。

## **3.1 问题一**

问题一中给了四个容器，分别为12斤、10斤、6斤、3斤的瓶子。问题一首先要求将12斤酒三等分、四等分，即通过n步有限操作（经过n-1个中间状态)实现（12,0,0,0）到（4,4,4,0）的转换和（12，,0,0,0）到（3,3,3,0）的转换。后来不再平分，即改变了目标状态，要求从（12,0，0,0）变为（5,3,2,2）。

## **3.2 问题二**

问题二在问题一的基础上将规模扩大，初始状态和目标状态也更具随机性，题目要求我们实现。

在给出实现步骤后，题目要求给出充分理由说明是最少步数，对所使用的模型和算法进行分析说明。

## **3.3 问题三**

在实现问题二的求解的基础上，题目要求自己设计一个实例，并要求步数不少于13步，给出详细的操作步骤。我们可以通过扩大问题的规模来达到题目的要求。

通过对分酒问题的分析研究，我们发现可以用图论的知识建立有向无权图求解最短路径。考虑是无权图，且只需求解一对状态的最短转化路径，而无需求出所有点到其他点的最短路径（Floyd算法），我们决定用广度优先算法结合深度优先来实现对图的遍历与查找路径。具体解题思路如流程图所示：

由给定初始状态开始，列举求出所有经过n步变化可以得到的不同状态(a1,a2,a3,…,an)，即图所有节点。

求出所有的两个节点之间的可转化关系<b1，b2>，以有序实数对记录，即为图的所有有向边。

求解最短路径，将所有的最短路径输出，如果不存在可达路径输出，则无法实现初态到目标状态的转化。

图 2 解题流程

# 模型建立

## QQ截图20170430212333**4.1 一般模型：求解****有向带权图的最短路径问题**

图 3 有向带权图举例

一张有向图由点及连接各点的边组成，若各边有固定走向，且有权重，则称为有向无权图，如图3。

求带权图的最短路径问题即求两个顶点间长度最短的路径。其中：路径长度不是指路径上边数的总和，而是指路径上各边的权值总和，如图3中路径A->D->F的长度为18+30=48。

## **4.1 模型具体化：分酒问题转化为有向无权图的最短路径问题**

1. 首先在每瓶最多装酒量为，初始装酒量状态为的条件下，求出不借助其他工具只在这些瓶子间相互倒酒可以实现的所有可能的中间状态i，当状态i可经过倒一次酒成为状态j时，记录下序偶。
2. 考虑将每一种各瓶装酒的情况i，对应画为有向图中的一个节点，所有的装酒情况组成节点集合V；
3. 若情况1经一步可以到达情况2，则画一条到的有向线段，箭头表示指向，如此划出所有可达关系的序偶对应的有向线段集合E；
4. 再将所有线段的权重置为1；
5. 如此，就将分酒问题的条件转化为了一张有向无权图G=<V,E>；
6. 求解是否可达及可达最短路径，即为求解G=<V,E>图中对应节点与对应节点是否可达及可达最短路径。

我们选择C++语言（软件DEV-C++）进行上述过程的编程实现，其中：

1. 使用广度优先搜索算法，实现所有装酒状态与一步可达性序偶的遍历与记录，将装酒状态编号为节点Vi存入集合V，序偶集合存入集合E，并转化为邻接矩阵，V与E构成有向图无权图G；
2. 再一次广度优先遍历G并生成至目标节点为止的“层次树”（该“层次树”在查找最短路径时使用，具体含义见6.算法实现与分析）；
3. 在所有节点集合V中查找目标状态，若未找到，则此状态不可达；
4. 若找到目标状态，则使用广度优先搜索算法在“层次树”中结合邻接矩阵遍历查找输出所有最短路径。

# 求解结果

## **5.1问题一求解结果：**

将12斤酒三等分时，最少经过10步完成，共12种最短路径。

所有最短路径如下所示：

表 1 （12,0,0,0）三等分所有最短路径

|  |  |
| --- | --- |
| **路径序号** | **具体操作步骤** |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |

将12斤酒四等分时，最少经过4步完成，共4种最短路径。

所有最短路径如下所示：

表 2 （12,0,0,0）四等分所有最短路径

|  |  |
| --- | --- |
| **路径序号** | **具体操作步骤** |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |

（5,3,2,2）不可实现。当目标状态要求为（5,3,2,2）时，在生成的所有可达状态中并不存在此状态，即目标状态（5,3,2,2）并不可以由初始状态（12,0,0,0）在每瓶最大装酒（12,10,6,3）时经过有限步的到达。

## **5.2问题二求解结果：**

初始状态到目标状态,要求不使用任何其他工具实现转换。

可以列出所有由初始状态经n步操作能实现的所有状态并生成有向无权图，如果目标状态在生成的有向无权图中，则存在从源节点到达当前节点的路径，即可以实现初始状态到目标状态的转化。

我们使用的建立“层次树”搜索最短路径的显然寻找出的最短路径必然是所有可达通路里最短的。

### 5.2.1模型分析：

可靠性：将所有的可以经由初始状态做n步变换可得到的状态都包罗在有向无权图中，所以如果问题有解，那么求解的路径一定包含在创建的有向无权图中，而且所有的可达路径都包含在该有向无权图中，不会有遗漏。

简易性：有向无权图容易建立，有向无权图中的求解最短路径有前人总结的算法可以供我们参考。

程序中输入瓶子数目、瓶子容量、初始状态、目标状态后即可求解，十分方便。

### 5.2.2算法分析：

（1）可靠性：本文的算法是基于广度优先算法，是对所有可通路径的遍历，所以如果能从初始状态到目标状态，那么我们一定可以找到所有的路径，从这当中所找到的最短路径必然完备且一定是最短路径(具体分析见6.算法实现与分析），不会有遗漏或错误。

（2）时间复杂度：，具体分析见6.算法实现与分析。

（3）空间复杂度：复杂度也为,但当瓶子数目N超过5时，装酒状态总数n很大，成指数级增长，空间占用仍然很大。

## **5.3问题三求解结果：**

**实例一：**现有5个瓶子容量为（16，8，6，14，3）时，初始状态（15，0，0，0，0），最少经过13步操作可以实现（9，1，1，1，3）。

程序运行求解出来共有147条最少操作13步的方法，方法见附录，下面列举其中的三条：

路径一：



路径二：



路径三：



**实例二：**现有6个瓶子容量为（20，16，8，14，3）时，初始状态（15,0,0,0,0,0），最少经过17步操作从（15,0,0,0,0,0）实现（4,4,1,2,4,0）。

附录附有20条最少操作17步的路径，下面列举其中三条：

路径一：

路径二：



路径三：



# 算法实现与分析

## **6.1 算法实现**

### 6.1.1 找出所有可能的装酒状态

取一个状态，找出经一步倒酒可实现的所有后继状态。其中，如果新得到的状态在以前没出现过，则将当前状态存储到节点集合V中，并将序偶<,>存储到有向边集合E中。如果状态已经在之前出现过 ，则不再存储到V中，仅储存序偶<,>到有向边集E中。

遍历V中所有没被查找过后继的状态节点，循环此过程，当所有新出现的状态都已存在在V中时结束循环。此时，所有可能的状态必都已经在V中。

以瓶子容量(8,5,3)，初始状态（8,0,0)为例说明我们找出所有可能的装酒状态的算法。

步骤一：

由已给定的初始状态建立一个节点，此时V={}，E=Φ。

求出所有经一步操作可以实现的状态为,。遍历V 发现，都不在V中，故将其存入V中，并将序偶<,><,>存入E。

查找完后，

V={，，}

E={<,>，<,>}

步骤二：

然后，按节点编号顺序，从V中取出没被查找过后继的节点，求出所有经一步操作可以实现的状态为，，。遍历V发现已存在在V中，故只将，存入V中，并将这一步得到的所有序偶<,><,><,>都存入E。

查找完后，

V={，，，，}

E={<,>，<,>，<,>，<,>，<,>}

步骤三：

再按节点编号顺序，从V中取出没被查找过后继的节点，求出所有经一步操作可以实现的状态为，，。遍历V发现，已存在在V中，故只将存入V中，并将这一步得到的所有序偶<,><,><,>都存入E。

查找完后，

V={，，，，，}

E={<,>，<,>，<,>，<,>，

<,>，<,>，<,>，<,>}

步骤四及往后：

再按节点编号顺序，从V中取出没被查找过后继的节点，重复上述步骤，再取没被查找过后继的节点……直到V中所有节点都查找过后继，不在有新节点加入为止，此时所有可能的状态必都已经在V中且所有一步可达序偶都在E中，将E转化为邻接矩阵，便于计算机操作。步骤如下图示：

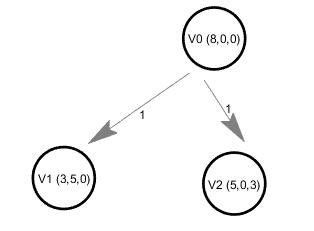


图 4 步骤一图示

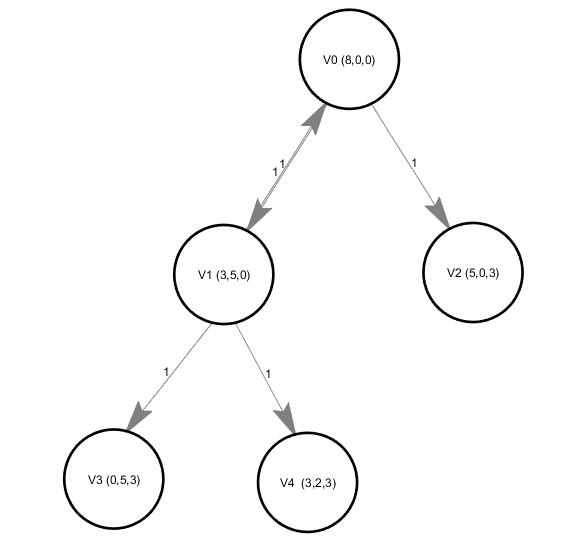
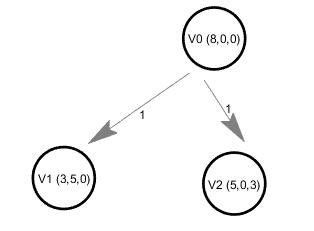


图 5 步骤二图示

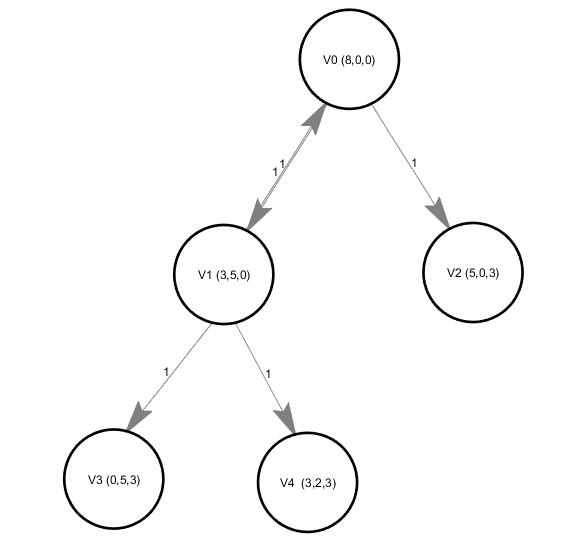
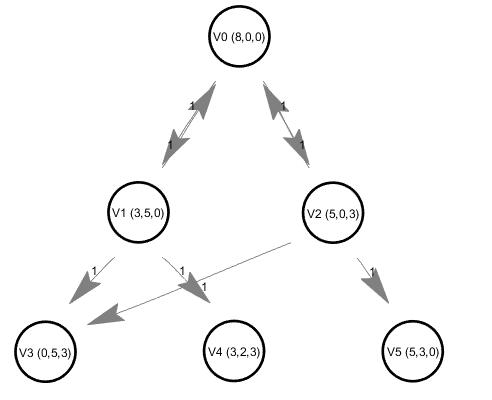


图 6 步骤三图示

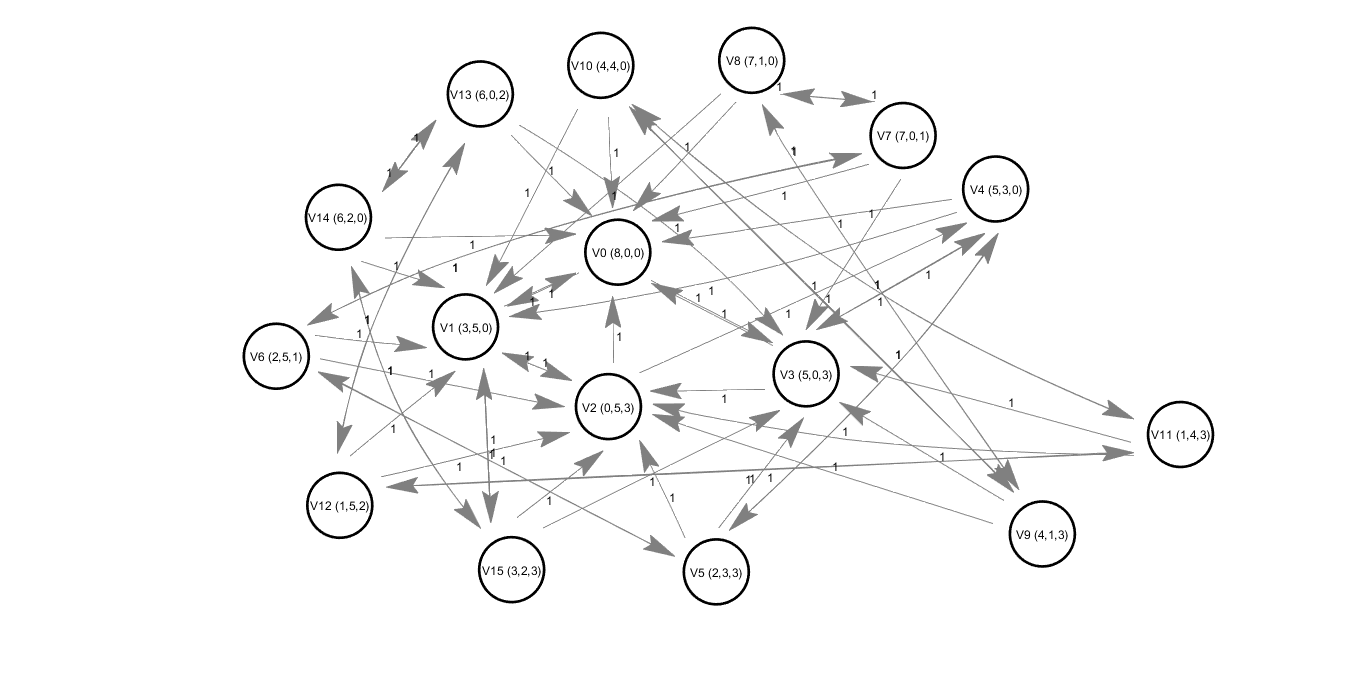


图 7 所有装酒状态生成的有向无权图G={V,E}（与图4,5,6节点编号有变化，但无影响）

### 6.1.2 建立“层次树”T

步骤一： 以初始状态作为根节点，约定所在层为层次树T第0层，向下建立一棵层次树T。取第0层中所有状态（即为），放入集合V中。将集合V中所有元素经过一次操作可到达且不在当前层次树T中的新状态（即为)记为集合V’存入树T第1层。

V= T[0]={}

T[1]=V’={,}

步骤二：清空集合V，将集合V’中所有状态存入集合V，清空集合V’。

V=Φ

V=V’={,}

V’=Φ

步骤三：将集合V中所有元素（,）经过一次操作可到达且不在当前层次树T中的新状态（,,）记为集合V’存入树T新的一层。

V’={,,}

T[2]=V’={,,}

步骤四：重复步骤二至步骤三，直至层次树T中已经包含所有当前初始状态可到达状态或树最底层包含了目标状态。

**层0**

**层1**

**层2**

**层3**

**层4**

**层5**

**层6**

**层7**

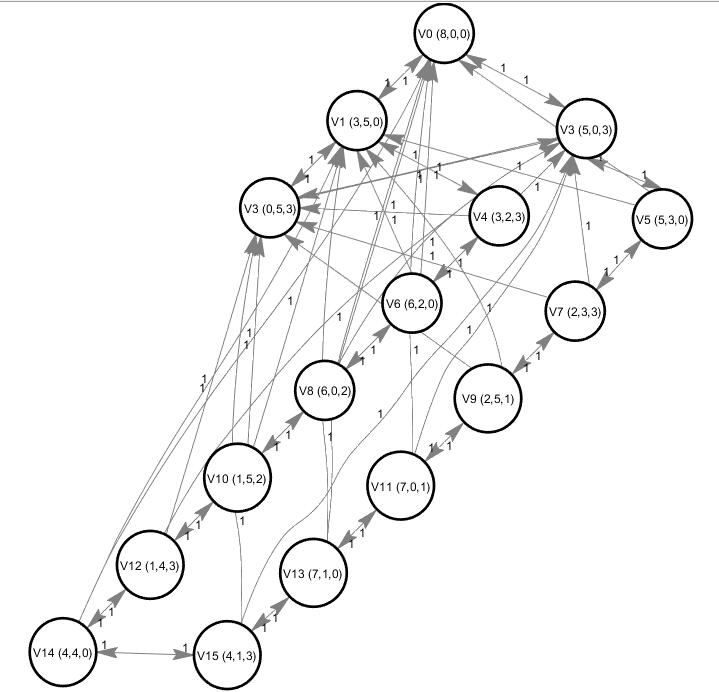


图 8 由图7转化为的“层次树”（实际仍为有向图，节点编号同图4,5,6编号）

容易发现，如此生成的“层次树”：

1. 任意两个层数差大于1的状态之间不存在直接由低层到高层的有向线段，
2. i+1层包含了i层节点的所有不在i层上层的后继，

所以，上层状态到下层状态的最短路径必为一层一层向下走。若路径有从下向上的边，则必有经过多余的点，不为最短。故以为起始状态到的最短路径长即为所在层数，从一层一层向下走到所有的不同通路即为所有最短路径。

### 6.1.3 遍历查找所有最短路径

步骤一：

定义一个一维数组trace记录最短路径，将序号为num的目标状态在“层次树”中的所在层数floor存入trace数组第floor位，然后结合有向边集合E（邻接矩阵）在“层次树”中第floor-1层搜索所有为目标状态前驱的节点，作为集合Vfloor中的元素。

Trace={0,0,0,0,0,0,0,15}

V7={V12}

Trace={0,0,0,0,0,0,12,15}

V6={V10}

依次以集合V中元素为新的目标状态，递归操作上述步骤，直至遍历结束。

当递归操作至顶层时trace中记录了从初态到目标状态的一条最短路径。

步骤二：

遍历trace数组根据Map表将序号转换成对于状态输出，然后回溯至下一层搜索集合Vfloor中的下一个状态，覆写trace对应位，继续向上搜索，直到每个Vfloor中的状态均被搜索过。

这棵根据广度优先搜索算法生成的层次树，任意两个层数差大于1的状态之间不存在直接的由层次低到层次高的直接有向线段。因此该算法依据层次树生成的路径trace必定为由初始状态至目标状态的最短路径，且目标状态所在层数即为最短路径长度。

因为算法遍历了所有由初始状态至目标状态的长度等于目标状态层数的路径，所以，算法列举的所有路径即为所有从初始状态至目标状态的最短路径。

程序递归遍历完每个当前状态的每个前驱后，程序结束，即全部输出了从初始状态到目标状态的所有最短路径方案。

## **6.2 算法复杂度分析**

设瓶子数目为N，由初始状态经n步操作可以实现的状态总数目为num，层次树层数为Num,最短路径长为Length,最短路径数为n。

算法第一部分找出所有状态基于广度优先算法，则算法的第一部分求出所有状态，生成Map表和Couple表过程时间复杂度最坏情况下为，因为num远大于N，故时间复杂度为；

算法的第二部分同样基于广度优先，在最坏的情况下，生成“层次树”的时间复杂度为,

第三部分遍历找所有最短路径最坏情况下，时间复杂度为，由于大规模时，num远大于Length和n，所以综合复杂度为。

综上，程序的综合时间复杂度，但是由于num基数非常大，我们计算发现，当六个瓶子时num就可以达到级别，程序运行仍然需要大量时间。

## **6.3编程实现：**

算法的第一部分实现生成有向无权图，用一个点的集合（存储在线性链表Map）和一个边的集合（存储在线性链表Couple）来表示图，点的集合中存储每个瓶子中酒的含量且Map表中初始化存入初始状态，边的集合中存储的是有通路的两点之间的序偶关系。建立一个链式队列，初始化队列当中存放着一个元素0。

在给定瓶子的数目N、每个瓶子的容量、初始状态每个瓶子内所装酒的量的情况下，首先判断队列是否为空，如果不空，执行出队操作，从Map表中读入数据并生成新的状态，如果队列中没有与这些状态相同的状态则给它们附上编号先放入Map表，再执行入队操作；如果队列中已经有与新的状态相同的状态，则无需把这个新生成的状态放入Map表，也无需执行入队操作。将这个由一个节点转换为另一个节点的过程以序偶关系存入Couple表。重复上述操作，直至队列为空。

算法的第二部分基于广度优先算法，首先根据Couple表生成邻接矩阵。

然后访问Map表中的第一个节点V0，将V0的状态设置为1，其余节点的状态设置为0，将这些状态放入层次树dictfloor中的第0层。根据邻接矩阵中的邻接关系，访问所有的与V0直接相邻且未被访问的后继邻接点Vi，将节点Vi的状态设置为1，其余节点设置为0，将这些状态放入dictfloor中的第一层。依次从这些邻接点出发，根据邻接关系访问它们所有的未被访问的后继邻接点Vk，将Vk的状态设置为1，其余节点状态设置为0，放入dictfloor中的下一层。重复上述过程，直到图中所有以V0为起始点可到达的点都被访问到或是这一层中包含了目标状态。如果是访问了所有的可到达点，那么在N瓶状态下该初始状态到任意可到达状态的最短路径最长为层数；如果是这一层包含了目标状态，那么层数即为初始状态到目标状态的最短路径。

算法第三部分与深度优先算法。定义一个一维数组trace记录最短路径，将序号为num的目标状态所在层floor存入trace数组第floor位，将num状态上一层第（floor-1）层数据存入一维数组dict，将dict与邻接矩阵第[num]列数据取与运算，数据存入dict数组，及将目标状态num存在于它上一层的前驱存入dict数组。遍历dict数组，依次取dict数组中值为1的位号作为新的num递归做上述操作。直至全部遍历完。当递归操作至顶层时，trace中记录了从初始状态到目标状态的一条最短路径，遍历trace数组根据Map表将序号转换成对于状态输出，然后回溯至下一层搜索下一个num，覆写trace对应位，继续向上搜索。当程序递归遍历完每个当前状态的每个前驱后，程序结束，即全部输出了从初始状态到目标状态的所有最短路径方案。

按照讨论三个瓶子时的思路编写算法，实现对规模为n的一般求解方法。

## **6.4 算法改进过程：**

分析题目联想到图论知识，将其分酒问题转化为求图中的两个节点之间最短通路的问题。求图中两点之间的最短通路，我们首先想到了Dijkstra算法，它可以求解无向赋权图中给定两点间的最短通路，其时间复杂度是，但是由于分酒问题中的操作从一个瓶子倒入另一个瓶子要么是一个瓶子倒空，要么是一个瓶子倒满，比如（1,5,2）可以通过一步操作变为（3,5,0），但是（3,5，0）无法一步逆操作变为（1,5,2）。所以两个操作之间不一定可逆，用无向图来表示会将一些本不可能出现的状况也算进来，增加了操作难度和求解时出错的概率。

由于某些操作是不可逆的，所以我们决定建立的模型为有向图，采用Floyd算法，Floyd算法可以求解任意两点之间的最短通路，其时间复杂度为 ，其时间复杂度大，运算耗费时间，而且只能给出一种解，无法给出所有的解，这种算法并不适合于此处进行求解。

考虑到分酒问题中需要求解所有的最短路径，而且生成的有向图中任意两个直接相邻节点间的路径的权值为1，相当于有向无权图，我们决定采用广度优先算法。广度优先算法的时间复杂度为，比Floyd算法低，而且可以没有遗漏的给出了所有的最短路径，很好地符合了题目要求。另外我们借鉴了Floyd中算法的邻接矩阵用来存储图，使算法操作更简单。

在用算法生成Map表和Couple表的过程中，刚开始我们采用了递归算法，后来改进为非递归算法降低了时空的消耗。

# 模型评价

## **7.1模型优缺点分析**

优点：

能够一次性求出所有最短路径，且复杂度不高。有向无权图求解最短路径有很多前人已经总结的思想与算法，所以建立了这个模型之后使问题的难度大大降低，主要集中在算法的实现上。

缺点：

本模型的主要缺点是，当瓶子数目N较大时，图中的节点数目的规模成指数级增长， 当六个瓶子时num就可以达到级别，这个问题难以解决，算法运行时间仍然很大。

## **7.2 模型推广**

分酒问题所建立的求解有向无权图中给定两点间的所有最短路径的模型，在网络传输、建筑施工、货物运输中都可以有应用之处，出来求解两点间的最短路径以供决策者使用。例如，在建筑施工中，修建供暖供气的管道时求最短距离便可以应用到这个模型，另外如果考虑通路的权值，可以建立有向有权图，用Floyd算法求解。

# 参考文献

1. 郭俊杰，毕双燕，雷冠丽 分油问题的网络最优化解法 长春邮电学院学报 1989年第七卷第一期
2. 傅彦，顾小丰，王庆光，刘启和 离散数学及其应用 高等教育出版社,2007
3. 严蔚敏，吴伟民 数据结构（C语言版） 清华大学出版社，2007
4. 杨智明 图的广度优先搜索遍历算法的分析与实现 农业网络信息2009年第12期

# 附录

## **9.1 算法的C++代码实现：**

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include<vector>

#include<fstream>

/\*Final version V2.5\*/

using namespace std;

#define N 4//瓶个数

int Cap[N];//容量

int Init[N];//初态

int Fin[N];//终态

static int i=0;

const int &INF=0;

struct Node//状态以及Map表中序号

{

int status[N];

int i;

Node \* nextNode;

};

struct Couple//序偶对

{

int relate[2];

Couple \* nextCouple;

};

struct Qnode

{

int status[N];

Qnode \* nextQueue;

};

struct Queue

{

Qnode \* front;

Qnode \* rear;

};

void ArrayInput(int A[N],int B[N])//将数组B赋值给数组A

{

for(int j=0;j<N;j++)

{

A[j]=B[j];

}

}

Node \* InitMap()//创建Map表，存放由初始状态经过若干步操作能到达的新状态

{

Node \* Map=NULL;

Map=new Node;

Map->i=i++;

ArrayInput(Map->status,Init);

Map->nextNode=NULL;

return Map;

}

Couple \* InitCouple()//创建并初始化C表，存放连通序偶对，在初始化时认为加入一个<0,0>序偶对，在后续使用中跳过

{

Couple \* C=NULL;

C=new Couple;

C->relate[0]=0;

C->relate[1]=0;

C->nextCouple=NULL;

return C;

}

void InitQueue(Queue &Q)

{

Qnode \* p=NULL;

p=new Qnode;

//ArrayInput(p->status,Init);

Q.front=p;

p->nextQueue=new Qnode;

Q.rear=p->nextQueue;

ArrayInput(Q.rear->status,Init);

Q.rear->nextQueue=NULL;

}

int QueueEmpty(Queue &Q)

{

return Q.front==Q.rear;

}

void EnQueue(Queue &Q,int status[])

{

Qnode \* p=NULL;

p=new Qnode;

ArrayInput(p->status,status);

Q.rear->nextQueue=p;

Q.rear=p;

Q.rear->nextQueue=NULL;

}

int DeQueue(Queue &Q,int status[])

{

if(Q.front==Q.rear)return 0;

Qnode \* p=NULL;

p=Q.front->nextQueue;

ArrayInput(status,p->status);

delete Q.front;

Q.front=p;

return 1;

}

int ArrayCheck(int A[N],int B[N])//判断项是否相等

{

for(int j=0;j<N;j++)

{

if(A[j]!=B[j])return 0;

}

return 1;

}

int Check(Node \*Map,int status[N])//查找项是否已存在

{

Node \* p;

p=Map;

while(p)

{

if(ArrayCheck(p->status,status))return p->i;//存在回i

p=p->nextNode;

}

return -1;//不存在回-1

}

void AntiCheck(Node \* Map,int num)//根据序号在Map表中查找对应状态，根据程序整体可知，当调用该函数时查找的序号必定在Map表中，因此无需考虑若循环完毕之后未搜索到对应状态

{

Node \* p;

p=Map;

while(p)

{

if(p->i==num)

{

cout<<'('<<p->status[0];

for(int j=1;j<N;j++)

{

cout<<','<<p->status[j];

}

cout<<')';

break;

}

p=p->nextNode;

}

}

void print(Node \* Map,const int &beg,const int &end,

const vector<vector<int> > &path)//传引用，避免拷贝，不占用内存空间

//也可以用栈结构先进后出的特性来代替函数递归

{

if(path[beg][end]>=0)

{

print(Map,beg,path[beg][end],path);

print(Map,path[beg][end],end,path);

}

else

{

cout<<"->";

AntiCheck(Map,end);

}

}

Node \* AppEnd(Node \* Map,int status[],int newnum)//将新项加入Map末尾

{

//cout<<'.';

Node \* p;

p=Map;

if(p==NULL)

{

p=new Node;

}

else

{

while(p->nextNode!=NULL)p=p->nextNode;

p->nextNode=new Node;

p=p->nextNode;

}

ArrayInput(p->status,status);

p->i=newnum;

p->nextNode=NULL;

return Map;

}

Couple \* Insert(Couple \*C,int num,int newnum)//将邻接关系加入C表

{

//cout<<'+';

int relate[]={num,newnum};

Couple \* p;

p=C;

while(p->nextCouple)p=p->nextCouple;

p->nextCouple=new Couple;

p=p->nextCouple;

ArrayInput(p->relate,relate);

p->nextCouple=NULL;

return C;

}

int checkQueue(Queue Q,int status[])

{

Qnode \*p=NULL;

p=Q.front;

while(p!=Q.rear)

{

if(ArrayCheck(p->status,status))return 1;

p=p->nextQueue;

}

return 0;

}

void Search(Node \* Map,Couple \*C,Queue &Q)//寻找该项的下一项

{

int status[N];

int n=0;

while(!(Q.rear==Q.front))

{

DeQueue(Q,status);

n=i;

for(int a=0;a<N;a++)

{

if(status[a]==0)continue;

for(int b=0;b<N;b++)

{

if(b==a||status[b]==Cap[b])continue;

else

{

int NewStatus[N];//新状态

/\*对新状态进行赋值\*/

ArrayInput(NewStatus,status);

if(NewStatus[a]<(Cap[b]-NewStatus[b]))

{

NewStatus[b]+=NewStatus[a];

NewStatus[a]=0;

}

else

{

NewStatus[a]=NewStatus[a]-Cap[b]+NewStatus[b];

NewStatus[b]=Cap[b];

}

/\*-------------\*/

if(Check(Map,NewStatus)==-1)

{

AppEnd(Map,NewStatus,i++);//新状态若不在Map内则入Map

if(checkQueue(Q,NewStatus)==0)EnQueue(Q,NewStatus);//新状态若不在Map表中且不在队列里，则入队

}

C=Insert(C,Check(Map,status),Check(Map,NewStatus));

}

}

}

}

}

void print1(Couple \* C)//C表遍历并打印序偶

{

fstream file;

file.open("Couple.txt");

Couple \* p;

p=C->nextCouple;

while(p)

{

file<<"<"<<p->relate[0]<<","<<p->relate[1]<<">\n";

p=p->nextCouple;

}

file.close();

}

void print2(Node \* Map)//Map表遍历并打印状态+序号

{

fstream file;

file.open("Map.txt");

Node \* p;

p=Map;

while(p)

{

file<<'('<<p->status[0];

for(int j=1;j<N;j++)

{

file<<','<<p->status[j];

}

file<<")i="<<p->i<<"\n";

p=p->nextNode;

}

file.close();

}

int FloorSearch\_1(int dict[],int end,int floor,int n\_num,vector<vector<int> > distmap,vector<vector<int> > &distmap1)

{

int floordict[n\_num]={0};//第floor层的全部后继，包括重复

for(int a=0;a<n\_num;a++)

{

if(distmap1[floor][a])

{

for(int b=0;b<n\_num;b++)

{

floordict[b]=floordict[b]||distmap[a][b];

}

}

}

//此时的floordict记录了floor层所有的后继

//floordict与dict对比剔除重复项，得到当前层项

for(int a=0;a<n\_num;a++)

{

if(dict[a])floordict[a]=0;

if(floordict[a])

{

distmap1[floor+1][a]=1;

dict[a]=1;

}

}

if(!dict[end])

{

floor++;

floor=FloorSearch\_1(dict,end,floor,n\_num,distmap,distmap1);

}

return floor;

}

int DictFull(int dict[],int n\_num)

{

for(int a=0;a<n\_num;a++)

{

if (!dict[a])return 0;

}

return 1;

}

int FloorSearch\_2(int dict[],int floor,int n\_num,vector<vector<int> > distmap,vector<vector<int> > &distmap1)

{

int floordict[n\_num]={0};

for(int a=0;a<n\_num;a++)

{

if(distmap1[floor][a])

{

for(int b=0;b<n\_num;b++)

{

floordict[b]=floordict[b]||distmap[a][b];

}

}

}

for(int a=0;a<n\_num;a++)

{

if(dict[a])floordict[a]=0;

if(floordict[a])

{

distmap1[floor+1][a]=1;

dict[a]=1;

}

}

for(int a=0;a<n\_num;a++)

{

if(floordict[a])cout<<a<<' ';

}

cout<<endl;

if(!DictFull(dict,n\_num))

{

floor++;

floor=FloorSearch\_2(dict,floor,n\_num,distmap,distmap1);

}

return floor;

}

void AntiCheck1(Node \* Map,int num)

{

fstream file;

file.open("result.txt",ios::app);

Node \* p;

p=Map;

while(p)

{

if(p->i==num)

{

cout<<'('<<p->status[0];

file<<'('<<p->status[0];

for(int j=1;j<N;j++)

{

cout<<','<<p->status[j];

file<<','<<p->status[j];

}

cout<<')';

file<<')';

break;

}

p=p->nextNode;

}

file.close();

}

void Output(Node \* Map,int trace[],int length)

{

fstream file;

for(int a=0;a<length;a++)

{

AntiCheck1(Map,trace[a]);

cout<<"->";

file.open("result.txt",ios::app);

file<<"->";

file.close();

}

AntiCheck1(Map,trace[length]);

cout<<endl;

file.open("result.txt",ios::app);

file<<"\n";

file.close();

}

void ReverseSearch(Node \* Map,int length,int end,int n\_num,int floor,vector<vector<int> > distmap,vector<vector<int> > distmap1,int trace[])

{

trace[floor]=end;

floor--;

if(!floor)

{

Output(Map,trace,length);

}

else

{

int floordict[n\_num]={0};//floordict记录end的前驱

for(int a=0;a<n\_num;a++)

{

if(distmap[a][end])floordict[a]=(floordict[a]||distmap[a][end])&&distmap1[floor][a];

}

for(int a=0;a<n\_num;a++)

{

if(floordict[a])

{

ReverseSearch(Map,length,a,n\_num,floor,distmap,distmap1,trace);

}

}

}

}

int main()

{

int n\_num=0,//点数

end;//终止状态序号

/\*输入瓶子容量\*/

cout<<"请输入"<<N<<"个瓶子的容量:"<<endl;

for(int j=0;j<N;j++)

{

scanf("%d",&Cap[j]);

}

/\*输入起始状态\*/

cout<<"请输入起始点:"<<endl;

for(int j=0;j<N;j++)

{

scanf("%d",&Init[j]);

}

/\*根据瓶子数量，容量，初始状态生成Map表与C表\*/

Node \* Map=NULL;

Couple \* C=NULL;

Queue Q;

InitQueue(Q);

Map=InitMap();

C=InitCouple();

Search(Map,C,Q);

/\*遍历Map表获取点数\*/

Node \* p1;

p1=Map;

while(p1)

{

n\_num++;

p1=p1->nextNode;

}

/\*默认初始化邻接矩阵\*/

vector<vector<int> >distmap(n\_num,vector<int>(n\_num,INF)),distmap1(n\_num,vector<int>(n\_num,INF));

/\*遍历C表获取边数并给邻接矩阵赋值\*/

Couple \* p;

p=C->nextCouple;

int a1,b1;

while(p)

{

a1=p->relate[0];

b1=p->relate[1];

distmap[a1][b1]=1;

p=p->nextCouple;

}

distmap1[0][0]=1;

print1(C);

print2(Map);

cout<<"计算完毕，可以开始查询:"<<endl;

/\*若存在则打印路径\*/

int floor=0;

int dict[n\_num]={0};

dict[0]=1;

int mode=0;

while(1)

{

cout<<"选择工作模式（1为求全部最短路径长度，0为求最大可达最短生成路径）:"<<endl;

cin>>mode;

/\*以下部分为列举所有最短路径可能\*/

if(mode==1)

{

cout<<"请输入终点："<<endl;

/\*-输入终止状态-\*/

for(int j=0;j<N;j++)

{

scanf("%d",&Fin[j]);

}

/\*查找终止状态是否存在于Map表中若不存在输出-1，反之输出Map表中序号\*/

end=Check(Map,Fin);

if(end>0)

{

floor=FloorSearch\_1(dict,end,floor,n\_num,dis

tmap,distmap1);

cout<<"最短路径长度为："<<++floor<<endl;

ofstream file1;

file1.open("result.txt");

file1<<"最短路径长度为："<<floor<<"\n";

file1.close();

//floor输出最底层层号=层数-1

int trace[floor]={0};

int length=floor;

ReverseSearch(Map,length,end,n\_num,floor,distmap,distmap1,trace);

}

/\*若初始状态与终止状态相同则提示两者相同\*/

else if(end==0)cout<<"起点与终点为同一点！"<<endl;

/\*若终止状态不在Map表中，即终止状态不可达到，则提示不可到达\*/

else cout<<"该终点无法到达!"<<endl;

}

/\*以下部分为输出生成的层次树的线性表表示\*/

else if(mode==0)

{

for(int a=0;a<n\_num;a++)

{

if(dict[a])cout<<a;

}

cout<<endl;

floor=FloorSearch\_2(dict,floor,n\_num,distmap,distmap1);

cout<<"最大可达最短路径为："<<++floor<<endl;

}

else cout<<"模式选择错误！";

}

return 0;

}

## **9.2 长度为13的最短路径：**

每个酒瓶最多装酒量为（16,14,8,6,3）初始状态为（15,0,0,0,0）目标状态为（9,1,1,1,3）的情况下，遍历得到的147条长度为13的最短路径，此处列举前20条：

1. 1.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,6,0,8,0)->(1,6,6,2,0)->(1,6,3,2,3)->(1,8,3,2,1)->(1,0,3,10,1)->(0,1,3,10,1)->(3,1,0,10,1)->(3,1,6,4,1)->(9,1,0,4,1)->(9,1,1,4,0)->(9,1,1,1,3)
2. 2.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,6,0,8,0)->(1,6,6,2,0)->(1,6,3,2,3)->(1,8,3,2,1)->(1,0,3,10,1)->(1,3,0,10,1)->(1,3,6,4,1)->(1,8,1,4,1)->(9,0,1,4,1)->(9,1,1,4,0)->(9,1,1,1,3)
3. 3.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,0,11,3)->(1,0,6,5,3)->(1,6,0,5,3)->(1,8,0,5,1)->(1,0,0,13,1)->(0,1,0,13,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
4. 4.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,0,6,5,3)->(1,6,0,5,3)->(1,8,0,5,1)->(1,0,0,13,1)->(0,1,0,13,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
5. 5.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,6,0,8,0)->(1,6,0,5,3)->(1,8,0,5,1)->(1,0,0,13,1)->(0,1,0,13,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
6. 6.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,0,11,3)->(1,0,6,5,3)->(1,6,0,5,3)->(1,8,0,5,1)->(1,0,0,13,1)->(1,0,6,7,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
7. 7.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,0,6,5,3)->(1,6,0,5,3)->(1,8,0,5,1)->(1,0,0,13,1)->(1,0,6,7,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
8. 8.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,6,0,8,0)->(1,6,0,5,3)->(1,8,0,5,1)->(1,0,0,13,1)->(1,0,6,7,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
9. 9.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,0,11,3)->(1,0,6,5,3)->(1,6,0,5,3)->(1,8,0,5,1)->(1,2,6,5,1)->(1,0,6,7,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
10. 10.(15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,0,6,5,3)->(1,6,0,5,3)->(1,8,0,5,1)->(1,2,6,5,1)->(1,0,6,7,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
11. (15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,6,0,8,0)->(1,6,0,5,3)->(1,8,0,5,1)->(1,2,6,5,1)->(1,0,6,7,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,1,7,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
12. (15,0,0,0,0)->(1,0,0,14,0)->(1,8,0,6,0)->(1,2,6,6,0)->(0,2,6,6,1)->(0,2,0,12,1)->(0,0,2,12,1)->(0,8,2,4,1)->(0,4,6,4,1)->(6,4,0,4,1)->(6,4,1,4,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
13. (15,0,0,0,0)->(1,0,0,14,0)->(0,0,0,14,1)->(0,8,0,6,1)->(0,2,6,6,1)->(0,2,0,12,1)->(0,0,2,12,1)->(0,8,2,4,1)->(0,4,6,4,1)->(6,4,0,4,1)->(6,4,1,4,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
14. (15,0,0,0,0)->(1,0,0,14,0)->(1,8,0,6,0)->(0,8,0,6,1)->(0,2,6,6,1)->(0,2,0,12,1)->(0,0,2,12,1)->(0,8,2,4,1)->(0,4,6,4,1)->(6,4,0,4,1)->(6,4,1,4,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
15. (15,0,0,0,0)->(1,0,0,14,0)->(1,8,0,6,0)->(1,2,6,6,0)->(1,2,0,12,0)->(0,2,0,12,1)->(0,0,2,12,1)->(0,8,2,4,1)->(0,4,6,4,1)->(6,4,0,4,1)->(6,4,1,4,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
16. (15,0,0,0,0)->(1,0,0,14,0)->(1,8,0,6,0)->(1,2,6,6,0)->(1,2,0,12,0)->(1,0,2,12,0)->(0,0,2,12,1)->(0,8,2,4,1)->(0,4,6,4,1)->(6,4,0,4,1)->(6,4,1,4,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
17. (15,0,0,0,0)->(1,0,0,14,0)->(1,8,0,6,0)->(1,2,6,6,0)->(1,2,0,12,0)->(1,0,2,12,0)->(1,8,2,4,0)->(0,8,2,4,1)->(0,4,6,4,1)->(6,4,0,4,1)->(6,4,1,4,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
18. (15,0,0,0,0)->(1,0,0,14,0)->(1,8,0,6,0)->(1,2,6,6,0)->(1,2,0,12,0)->(1,0,2,12,0)->(1,8,2,4,0)->(1,4,6,4,0)->(0,4,6,4,1)->(6,4,0,4,1)->(6,4,1,4,0)->(6,1,1,4,3)->(9,1,1,4,0)->(9,1,1,1,3)
19. (15,0,0,0,0)->(1,0,0,14,0)->(1,0,0,11,3)->(1,0,6,5,3)->(1,6,0,5,3)->(1,8,0,5,1)->(1,0,0,13,1)->(0,1,0,13,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,6,1,1)->(12,1,0,1,1)->(12,1,1,1,0)->(9,1,1,1,3)
20. (15,0,0,0,0)->(1,0,0,14,0)->(1,0,6,8,0)->(1,0,6,5,3)->(1,6,0,5,3)->(1,8,0,5,1)->(1,0,0,13,1)->(0,1,0,13,1)->(0,1,6,7,1)->(6,1,0,7,1)->(6,1,6,1,1)->(12,1,0,1,1)->(12,1,1,1,0)->(9,1,1,1,3)

## **9.3 长度为17的最短路径：**

6个瓶子容量为（20，16，8，14，3）时，初始状态（15,0,0,0,0,0），最少经过17步操作从（15,0,0,0,0,0）实现（4,4,1,2,4,0），因程序规模大，我们没有等程序运行完，至我们停止程序共输出了2030条最短路径，且仍有在输出，下面列举20条路径。

1. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,8,0,3,3)->(1,0,2,6,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
2. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,0,0,11,3)->(1,0,8,0,3,3)->(1,0,2,6,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
3. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,2,6,6,0)->(1,0,2,6,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
4. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,8,0,3,3)->(1,8,0,0,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
5. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,0,0,11,3)->(1,0,8,0,3,3)->(1,8,0,0,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
6. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,8,0,0,6,0)->(1,8,0,0,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
7. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,2,6,6,0)->(1,2,0,6,6,0)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
8. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,8,0,0,6,0)->(1,2,0,6,6,0)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
9. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,2,6,6,0)->(1,2,0,6,6,0)->(1,2,6,0,6,0)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
10. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,8,0,0,6,0)->(1,2,0,6,6,0)->(1,2,6,0,6,0)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,2,2,6,3,1)->(1,2,2,0,9,1)->(1,2,0,2,9,1)->(1,2,8,2,1,1)->(1,10,0,2,1,1)->(1,10,1,2,1,0)->(1,7,1,2,1,3)->(4,7,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
11. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,8,0,3,3)->(1,0,2,6,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
12. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,0,0,11,3)->(1,0,8,0,3,3)->(1,0,2,6,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
13. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,2,6,6,0)->(1,0,2,6,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
14. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,8,0,3,3)->(1,8,0,0,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
15. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,0,0,11,3)->(1,0,8,0,3,3)->(1,8,0,0,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
16. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,8,0,0,6,0)->(1,8,0,0,3,3)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
17. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,2,6,6,0)->(1,2,0,6,6,0)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
18. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,8,0,0,6,0)->(1,2,0,6,6,0)->(1,2,0,6,3,3)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
19. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,0,2,6,6,0)->(1,2,0,6,6,0)->(1,2,6,0,6,0)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)
20. (15,0,0,0,0,0)->(1,0,0,0,14,0)->(1,0,8,0,6,0)->(1,8,0,0,6,0)->(1,2,0,6,6,0)->(1,2,6,0,6,0)->(1,2,6,0,3,3)->(1,2,8,0,3,1)->(1,10,0,0,3,1)->(1,4,0,6,3,1)->(1,4,0,0,9,1)->(1,4,8,0,1,1)->(1,4,2,6,1,1)->(7,4,2,0,1,1)->(7,4,0,2,1,1)->(7,4,1,2,1,0)->(4,4,1,2,1,3)->(4,4,1,2,4,0)