**汉诺塔递归与非递归算法研究**

作者1，作者2，作者33

(陕西师范大学 计算机科学学院，陕西 西安 710062)

**摘 要**: 摘要内容(包括目的、方法、结果和结论四要素) 摘要又称概要,内容提要.摘要是以提供文献内容梗概为目的,不加评论和补充解释,简明,确切地记述文献重要内容的短文.其基本要素包括研究目的,方法,结果和结论.具体地讲就是研究工作的主要对象和范围,采用的手段和方法,得出的结果和重要的结论,有时也包括具有情报价值的其它重要的信息.摘要应具有独立性和自明性,并且拥有与文献同等量的主要信息,即不阅读全文,就能获得必要的信息.

**关键词:** 关键词1; 关键词2；关键词3；……（一般可选3~8个关键词，用中文表示，不用英文

Title

**如**：XIN Ming-ming , XIN Ming

(1.Dept. of \*\*\*\*, University, City Province Zip Code, China；2.Dept. of \*\*\*\*, University, City Province Zip Code, China；3.Dept. of \*\*\*\*, University, City Province Zip Code, China)

**Abstract:** abstract（第三人称叙述，尽量使用简单句；介绍作者工作（目的、方法、结果）用过去时，简述作者结论用一般现在时）

**Key words:** keyword1;keyword2; keyword3;**……（**与中文关键词对应，字母小写(缩略词除外) **）**；

**正文部分用小5号宋体字，分两栏排，其中图表宽度不超过8cm.。设置为A4页面**

**1 引言 （一级标题四号黑体加粗）**

引言的作用就是引出为什么要写这篇文章，主要有以下几个方面：

（1）如果以采用新方法新理论，就要引出为什么要采用这种方法；

（2）如果是为了阐明某个观点，就要引出目前观点和目前对所研究领域的现状；

（3）为什么要研究“XXX”算法（为什么要研究它，背景及必要性）

**如：（文中举例内容仅供参考）汉诺塔问题的描述：**汉诺塔(Tower of Hanoi)问题又称“世界末日问题”有这样一个故事[1]。古代有一个焚塔，塔内有3个基座A,B,C，开始时A基座上有64个盘子，盘子大小不等，大的在下，小的在上。有一个老和尚想把这64个盘子从A座移到B座，但每次只容许移动一个盘子，且在移动过程中，3个基座上的盘子都始终保持大盘在下，小盘在上。移动过程中可以利用C基座做辅助。如图1所示：



**图1 汉诺塔问题**

这个问题当时老和尚和众僧们，经过计算后，预言当所有的盘子都从基柱A移到基座B上时，世界就将在一声霹雳中消灭，而梵塔、庙宇和众生也都将同归于尽。其实，不管这个传说的可信度有多大，如果考虑把64个盘子，由一个塔柱上移到另一根塔柱上，并且始终保持上小下大的顺序。假设有n个盘子，移动次数是f(n).显然f(1)=1,f(2)=3,f(3)=7,且f(k+1)=2\*f(k)+1。此后不难证明f(n)=2n-1。n=64时，

f(64)= 2^64-1=18446744073709551615

假如每秒钟一次，共需多长时间呢？一年大约有 31536926 秒，计算表明移完这些金片需要5800多亿年，比地球寿命还要长，事实上，世界、梵塔、庙宇和众生都早已经灰飞烟灭。

对传统的汉诺塔问题，目前还有不少的学者继续研究它的非递归解法，本文通过对递归算法的研究…….

**提示：（1）可以定义问题的规模n,如盘子的数量；（2）塔柱的数量（目前有部分理论可以支撑，不妨用计算机实现）分析规模的变化与算法的复杂度比较。（3）可以对经典的汉诺塔问题条件放松、加宽，如在经典的汉诺塔问题中大盘只能在小盘下面，放松其他条件可以定义相邻两个盘子必须满足大盘只能在小盘下面。其它盘子不作要求。**

**2 算法设计**

**2.1**  **汉诺塔递归算法描述（二级标题小五黑体加粗）**

用人类的大脑直接去解3，4或5个盘子的汉诺塔问题还可以，但是随着盘子个数的增多，问题的规模变的越来越大。这样的问题就难以完成，更不用说吧问题抽象成循环的机器操作。所以类似的问题可用递归算法来求解。下面n个盘的汉诺塔问题可用如下递归方法实现。

如果n=1，则将圆盘从A直接移动到B。  
如果n=2，则：  
（1）将A上的n-1（等于1）个圆盘移到C上；  
（2）再将A上的一个圆盘移到B上；

（3）最后将C上的n-1（等于1）个圆盘移到B上。  
如果n=3，则：  
A）将A上的n-1（等于2）个圆盘移到C（借助于B），步骤如下：  
（1）将A上的n-2（等于1）个圆盘移到B上。  
（2）将A上的一个圆盘移到C。  
（3）将B上的n-2（等于1）个圆盘移到C。  
B）将A上的一个圆盘移到B。  
C）将C上的n-1（等于2）个圆盘移到B（借助A），步骤如下：  
 （1）将C上的n-2（等于1）个圆盘移到A。  
 （2）将C上的一个盘子移到B。  
 （3）将A上的n-2（等于1）个圆盘移到B。到此，完成了三个圆盘的移动过程。  
从上面分析可以看出，当n大于等于2时， 移动的过程可分解为三个步骤：  
第一步 把A上的n-1个圆盘移到C上；  
第二步 把A上的一个圆盘移到B上；  
第三步 把C上的n-1个圆盘移到B上；  
其中第一步和第三步是类同的。

**算法如下：（伪码描述、自然语言描述、流程图）**

Main

1: { int n ;

2: Input(n);

3: Hanoi(n,”A”,”B”,”C”) ; }

4: Hanoi(n,char a,char b,char c)

5: { if (n>0)

6: { hanoi ( n - 1, a, c, b) ;

7: printf “( %d %a - > %c \n”, n , a, c) ;

8: hanoi ( n - 1,b, a, c) ;}

}

递归算法结构清晰，可读性强，而且很容易用数学归纳法证明算法的正确性，然而它的运行效率较低，它的时间复杂度主要在程序嵌套调用所用的时间。T(N)=2T(N-1)+1,容易计算出T(N)=2N-1.若在程序中消除递归调用，使其转化为非递归调用算法。通常，消除递归采用一个用户定义的栈来模拟系统的递归调用工作栈，从而达到递归改为非递归算法的目的。

**2.2 汉诺塔非递归算法描述**

**2.2.1** 非递归1：遍历二叉树搜索解空间（三级标题小五楷体）

通过定义MAXSTACK栈，可将递归算法转化为非递归调用算法。具体程序如下：

#define MAXSTACK 100 /\* 栈的最大深度 \*/

int N = 3; /\* N阶问题/\*

int c = 1; /\* 一个全局变量，表示目前移动的步数 \*/

struct hanoi { /\* 存储汉诺塔的结构，包括盘的数目和三个盘的名称 \*/

int n; char a, b, c;};

struct hanoi p[MAXSTACK];

void move(char a, int n, char c) /\* 移动函数，表示把某个盘从某根针移动到另一根针 \*/

{ printf("%d. Move disk %d from %c to %c\n", c++, n, a, c);}

void push(struct hanoi \*p, int top, char a, char b, char c,int n)

{p[top+1].n = n - 1;

p[top+1].a = a;

p[top+1].b = b;

p[top+1].c = c; }

void unreverse\_hanoi(struct hanoi \*p) /\*汉诺塔的非递归算法\*/

{ int top = 0;

while (top >= 0) {

while (p[top].n > 1) { /\* 向左走到尽头 \*/

push(p, top, p[top].a, p[top].c, p[top].b, p[top].n);

top++; }

if (p[top].n == 1) { /\* 叶子结点 \*/

move(p[top].a, 1, p[top].b);

top--; }

if (top >= 0) { /\* 向右走一步 \*/

move(p[top].a, p[top].n, p[top].c);

top--;

push(p, top, p[top+1].b, p[top+1].a, p[top+1].c, p[top+1].n);

top++; } }}

int main(void)

{ printf("unreverse program:n");

c = 1; p[0].n = N;

p[0].a = 'a', p[0].b = 'b', p[0].c = 'c';

unreverse\_hanoi(p);

return 0;}

2.2.2非递归2：优化遍历二叉树搜索解空间

**如：**从汉诺塔的递归算法中可知，当盘子的个数大于2 时，汉诺塔的移动过程分为3步，第一步将n-1个盘从A 移到C；第二步将第n盘从A 移到B；第三步将n-1个盘从C移到B。如果把移动过程看作是二叉树的中序遍历，则可用二叉树与汉诺塔移动过程建立一个映射[2，3]。如图2所示，三阶盘数，所对应的移动过程共有3！=6种移动方法。即:A→B ,A→C, B→C, B→A, C→A, C→B 6种移动方法.



**图 2 移动过程的映射**

在构造解空间树的时候，遵循由初始塔→目标塔，分解为两部分：初始塔→和中转塔→目标塔。如图3所示构造n阶汉诺塔问题的解空间树与对应的解。依次类推一直到达叶节点生成满二叉树。最后对生成的二叉树中序遍历，每搜索一个结点，对应的输出它的映射值，例如：搜索到0号结点，则输出A→B, 搜索到3号结点，则输出B→A, 搜索到5号结点，则输出C→B.依次类推直到解空间树中所有结点搜索完成，算法结束。



**图3 3阶汉诺塔与所对应的解空间树**

下面给出它的中序遍历算法。将二叉树严格按照左子树在左，右子树在右画，中序遍历的结果应该是结点从左到右的一个排列。由于它是满二叉树，整个输出过程是，先输出最下层的一个结点，然后，输出上层中第一个左子树能包含该结点的结点，然后，再输出下层的下一个结点，再输出上层中第一个左子树能包含该结点的结点，直到下层的结点全部输出完为止。用一维数level\_position [] 存储某一层已输出的结点序号。由于该二叉树是满二叉树，上层第i个结点的左孩子一定是下层的第2i-1个结点，右孩子一定是下层的第2i个结点。这样，判断下层结点是否是上层结点的右孩子，只要判断上下层结点在其本层的编号是否构成2倍关系即可，整个算法程序实现如下：

void output (int present\_level, int position, int n)

//参数分别为：当前层号，在本层的序号，总层数

{ int val;

val= (position-1) %3;

if (present\_level%2= =1) val=val+3;

//如果是奇数层，其值为3, 4, 5

switch (val)

{case 0: printf ("%d from A--->B\n" ,n -present\_level+1) ; break;

case 1: printf (" %d from B--->C\n" ,n-present\_level+1) ; break;

case 2: printf (" % d from C--->A\n" ,n -present\_level+1);break;

case 3:printf ("% d from A --->C\n" ,n -present\_level+1) ; break;

case 4: printf (" %d from C--->B\n" ,n-present\_level+1) ; break;

case 5:printf ("% d from B--->A\n" ,n-present\_level+1); break;}}

main ()

{ int level\_position [100] ; //某层的已输出的结点序号

int n,i,sample\_nub,total\_sample;//最后一层已输个数、总个数

printf (" input n=") ; //盘的数量

scanf (" %d" ,&n) ;

printf (" \n") ;

sample\_nub=0;total\_sample=1;

for (i=1;i<n;i++) total\_sample\*=2; //最底层总样点数

for (i=0;i<=n;i++) level\_position [i] =0;

i=n;

level\_position [i] ++;

output (i,level\_position [n] ,n) ;//输出第i层某一序号的结点

sample\_nub++;

while (sample\_nub<total\_sample)

{ while (level\_position [i] ==2\*level\_position [i-1] )

i--; //寻找把该结点作为左子树的祖先结点

level\_position [i-1] ++;

output (i-1,level\_position [i-1] ,n) ;

i=n;

level\_position [i] ++;

output (i,level\_position [n] ,n) ;

sample\_nub++;} }

**3 算法分析**

**定理1**.算法1是可终止的。

证明：…..

**定理2**.算法1是有效的。

证明：…..

**定理3**.算法1的时间复杂度是O()。

证明：…..

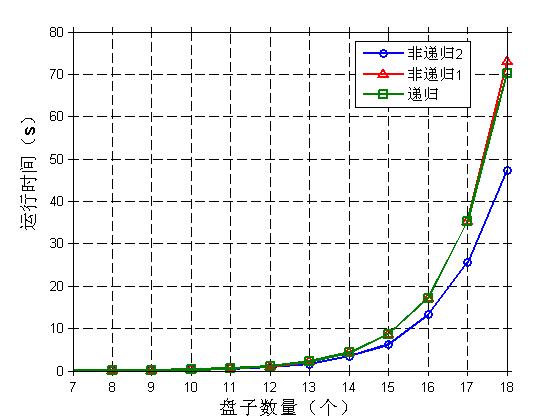
**定理4**.算法1的空间复杂度是S()。

证明：…..

**4 仿真实验与分析**

主要包括仿真平台（Matlab、C、C++、JAVA 、VC等）、仿真参数设置、实验分析

**如：**本文实验环境：CPU，Intel E6320。内存，DDR2；容量，1024MB；硬盘，160GB；缓存，8192KB；，PF使用率：47%-50%；集成开发工具：Microsoft Visual C++ 6.0，上对nanoi塔问题递归、非递归算法规模（盘子个数N）和执行时间进行仿真。图4表明递归算法与非递归算法随着塔柱上盘子个数的增多，所执行的时间均已指数级增长。同时递归算法与非递归算法1曲线变化不是很明显，从初始盘子为7到盘子个数为18均保持一致，主要原因是非递归算法1是在递归算法基础上消除递归调用，并没有进行智能优化。故它们在执行过程中所用的时间相同。即他们在时间复杂度均为2n-1。

****

**图4 塔数与3种算法运行时间**

而非递归算法2在盘子个数为9，就开始比递归算法与非递归算法1所执行时间要短。特别是随着盘子个数增加到15的时，非递归算法2明显比前两个算法时间上占有。说明非递归算法二在时间复杂度上要低于前两个算法的时间复杂度2n-1。这与我们预期分析一致。（文中实验仿真图用matlab7.0绘制）

**5 总 结**

总结是以研究成果为前提,经过严密的逻辑推理和论证所得出的最后结论.在结论中应明确指出论文研究的成果或观点,对其应用前景和社会,经济价值等加以预测和评价,并指出今后进一步在本研究方向进行研究工作的展望与设想.结论应写得简明扼要,精练完整,逻辑严谨,措施得当,表达准确,有条理。它是摘要的一部分，但要比摘要更详细。

**参考文献:** （参考文献示例参见下页）

［1］ 著者.题目［J］.刊名(全称,英文期刊名以黑体标志), 出版年,**卷号**(期号):起始页码. [期刊]

［2］ 著者.书名［M］.译者，译.版本项（初版不写） 出版地(城市名): 出版者, 出版年：起始页码.[书籍]

［3］ 著者.题目：文集实际完整名称，出版年[C].出版地：出版者，出版年：起止页码.[会议录(论文集、论文汇编等)]

［4］ 著者.析出题目[文献类型标志]//整本文献的编者姓名. 文集实际完整名称.版本项.出版地(城市名): 出版者,出版年:起止页码.[析出文献)]

［5］ 著者.题名[D]. 所在城市：学位授予单位, 出版年.[学位论文]

［6］ 著者.题名，报告号[R]. 出版地 (城市名): 出版者, 出版年.[科技报告、手册等]

［7］ 著者.准编号 标准名称［S］.出版地: 出版者,出版年.

［8］ 著者.题名［N］.报纸名，出版日期（版次）.[报纸文章] 出版日期按YY-MM-DD格式

［9］著者.题名［文献类型标志/电子文献载体标志］.(更新日期) [引用日期].获取和访问路径（如http://www.www.arocmag.com）.[电子文献]

［10］专利所有者.专利题名：专利国别，专利号［P］.公告日期.获取和访问路径.

**如：**

1. 吕国英,任瑞征,钱宇华.算法设计与分析（第二版）[M].北京:清华大学出版社，2009：57-59.
2. 邱宁. 汉诺塔问题的非递归算法分析[J].浙江树人大学学报，2005.5(02)：117-118.
3. 陈文. 汉诺塔非递归算法[J]. 电脑编程技巧与维护，2009，14：10-11.