

**数学与信息学院学生实验报告**

**实验课程名称：** 回溯算法设计与应用 **教师： \_\_**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验项目名称** | **实验三 贪心算法设计与应用** | | | **实验成绩** |  |
| **学生姓名** |  | **学 号** | **116052017163** | **年级专业班级** | **2017级软件工程** |
| **小组成员** |  | | | **实验日期** | **2019年5 月** |

# 1. 实验目的和要求

## 1.1 实验目的

① 理解回溯法的基本原理，掌握回溯法设计的基本方法及步骤，并应用于具体

　 问题的解决

② 用回溯算法设计求马的周游问题的算法，分析其复杂性，并实现

③ 用回溯算法设计求寻宝问题的算法，分析其复杂性，并实现

④ 用回溯算法设计求邮票问题的算法，分析其复杂性，并实现

## 1.2 实验软硬件环境

① 操作系统：win10

② 编译环境：JDK1.8，Eclipse Committers 2018-9

## 1.3 实验要求

### 1.3.1 马的周游问题

① 问题描述

　 在n×n棋盘（有n×n个格点的棋盘）的某个格点上有一个中国象棋马，马

　 走日字。求一条周游棋盘的路径，使得马能够从起始位置起沿着该路径每个

　 格点恰好走一次最后回到出发位置。

② 具体要求

　 用回溯法解决该问题。输入一个正整数n，输出一个解，解的输出形式尽可

　 能直观。

③ 设计与实现的提示

　 （1）注意可能无解。

　 （2）注意棋盘边界的处理

### 1.3.2 寻宝问题

① 问题描述

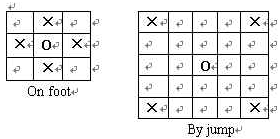
　 对于某个m\*n的字符串数组，相当于一个m行n列的平面形状的方格。里面

　 S表示起点，W表示障碍，B表示可走（但是不一定可以通）,X表示出口。

　 对于起点S，有8个方向可以走，当然前提是在没有障碍的情况之下，其中

　 可以分为单步走（on foot）和跳步走（by jump）两种情况，从起点S开始

　 追寻最短的出口路径count2。



② 具体要求

　 输入：The first line of the input is an integer n(0<=n<=20),

　　　　 Denothing the number of test cases.The following lines are

　　　　 the data of n test cases.The first line of each test case

　　　　 consists of two integer x and y (0<x<10,0<y<10).The next x

　　　　 lines describe the maze matrix,which contains x rows and y

　　　　 columns.In the matrix.’W’denotes the barrier,’B’denotes

　　　　 the enterable position,’S’denotes the starting position,

　　　　 and ‘X’denotes the position of the ark.There have and only

　　　　 have one ‘S’and one ‘X’in a maze matrix.

　 输出： For each test case,output one line.If there is a solution

　　　　 for the problem,output the number of the least steps to

　　　　 find the Ark.Otherwise output “NO ANSWER”.

③ 测试数据和结果

　 输入3

　 3 3

　 SWB

　 BWB

　　　　 XBB

　　　　 5 4

　　　　 BXWB

　　　　 BBWS

　　　　 BBWB

　　　　 BBWB

　　　　 BBBB

　　　　 3 2

　　　　 WX

　　　　 WW

　　　　 WS

　 输出：2

　　　　 2

　　　　 NO ANSWER

### 1.3.3 邮票问题

① 问题描述

　 设有n种不同面值a1，a2，...，an的邮票，规定每封信最多贴m张邮票。

　 对于给定的m，n，求出最大的邮资连续区间。例如，给定n=3，m=3，邮票

　 面值分别为2,3,5，则最大的邮票连续区间为[2,13]。

② 具体要求

　 输入：输入的第一行是一个正整数n，表示测试例个数。接下来几行是n个

　　　　 测试例的数据，每个测试例的数据由两行组成，其中第一行含两个正

　　　　 整数n个m（1<=n,m<=10），表示有n种邮票，每封信最多贴m张邮

　　　　 票；第二行含n个正整数，表示n种邮票的面值。同一行整数之间用

　　　　 一个空格隔开

　 输出：对于每个测试例输出一行，含两个整数，依次为最大的邮资连续区间

　　　　 的下界和上界，当存在多个最大的邮资连续区间时，输出下界值最小

　　　　 的一个。同一行两个整数之间用一个空格隔开

③ 测试数据和结果

　 输入：2

　　　　 3 3

　　　　 2 3 5

　　　　 4 5

　　　　 1 4 12 21

　 输出：2 13

　　　　 1 71

# 2. 实验记录

## 2.1 知识点归纳

### 2.1.1 回溯算法的基本原理

DFS+剪枝（在状态空间树上作带剪枝的DFS搜索）剪枝：若搜索到某结点

，其对应的部分解不满足解的约束条件且可断定以其为根的子树上不包含

答案结点，则不搜索该子树，直接回到其父结点，继续DFS。

利用回溯法可求问题的一个解，多个解，所有解，最优解，还可判断解的

存在性。

### 2.1.2 回溯算法的基本思路

其基本的解题思路为：

（1）定义给定问题的解空间；

（2）确定并表示解的约束条件和其他的剪枝条件；

（3）结合剪枝深度优先搜索相应的状态空间树。

注意：回溯法的一个特征是在搜索过程中动态的产生问题的状态空间树，

任何时候只存根到当前搜索的结点的路径。

### 2.1.3 回溯算法的基本概念

回溯算法实际上一个类似枚举的搜索尝试过程，主要是在搜索尝试过程中

寻找问题的解，当发现已不满足求解条件时，就“回溯”返回，尝试别的

路径。回溯法是一种选优搜索法，按选优条件向前搜索，以达到目标。但

当搜索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新

选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状

态的点称为“回溯点”。许多复杂的，规模较大的问题都可以使用回溯法

，有“通用解题方法”的美称。

### 2.1.4 回溯算法的基本思想

回溯算法的基本思想是：从一条路往前走，能进则进，不能进则退回来，

换一条路再试。八皇后问题就是回溯算法的典型，第一步按照顺序放一个

皇后，然后第二步符合要求放第2个皇后，如果没有位置符合要求，那么

就要改变第一个皇后的位置，重新放第2个皇后的位置，直到找到符合条

件的位置就可以了。回溯法在迷宫搜索中使用很常见，就是这条路走不通

，然后返回前一个路口，继续下一条路，回溯算法说白了就是穷举法。不

过回溯算法使用剪枝函数，剪去一些不可能到达最终状态（即答案状态）

的节点，从而减少状态空间树节点的生成。回溯法是一个既带有系统性又

带有跳跃性的搜索算法。它在包含问题的所有解的解空间树中，按照深度

优先的策略，从根节点出发搜索解空间树。算法搜索至解空间树的任一结

点时，总是先判断该结点是否肯定不包含问题的解。如果肯定不包含，则

跳过对以该节点为根的子树的系统搜索，逐层向其祖先结点回溯。否则，

进入该子树，继续按深度优先的策略进行搜索。回溯法在用来求问题的所

有解时，要回溯到根，且根节点的所有子树都已经被搜索遍才结束。而回

溯法在用来求问题的任一解时，只要搜索到问题的一个解就可以结束。这

种以深度优先的方式系统地搜索问题的解的算法成为回溯法，它适用于解

一些组合数较大的问题。

### 2.1.5 回溯算法的算法框架

pocedure try(i:integer);

var

begin

if i>n then 输出结界

else for j:=下界 to 上界 do

begin

x[i]:=h[j];

if 可行（满足界限函数和约束条件） then begin 置值;try(i+1);end;

end;

### 2.1.6 马的周游问题

**回溯法选择策略：**

马走日字，每一步有8个方向可以选择，利用回溯穷举8个方向进行尝试

即可，直到找到答案，搜索停止，exit（1）退出，不然结果不唯一，且指

数级的复杂度，分分钟爆掉。

至于书中给的N，M可能很大，这时候我们会用到分治法。

分治算法在计算过程中必须依赖一些事先计算好的少量数据，所以必须通

过回溯用O（1）的时间内计算6\*6、6\*8、8\*8、8\*10、10\*10、10\*12的结

构化棋盘。（因为这些棋盘的规模与N无关，可以事先计算好）。

**回溯法的一般思路：**

对于马所在其中一格时，它可以走的位置有以下8种情况：

⑧ ①

⑦ ②

⑥ ③

⑤ ④

所以对于每一个马所在的格子里，马可以走对应的8个方向。

用满8叉树，每一个子树对应马可跳的方向。

当要走下一子树（跳下一格）时，该子树可走（还没有走过并且在棋盘里

边），即沿该方向走下去，当不可以走的时候，即回溯到上一步，选择另

一方向往下走，当该子树的8个子棋都遍历完了（即8个方向都走过了）

，则回溯到它父亲那里。

重复一直做下去，到棋盘每个格子都走过一遍，而且回到出发点或者找不

到路径即结束。

### 2.1.7 寻宝问题

**算法分析**：

一看到这个题目，给人的第一感觉是用回溯算法，从面额1开始，每种面

额都用回溯进行判断，算法复杂度并不高，但是当m，n取到极限值100时

，程序明显超时，因此，回溯算法在取极限的时候并不可取。能否用递归

完成呢？我们有一个思路：从面额1开始，建立递推关系方程，就用范例

来说吧，面额1,2,4只用1张邮票就行了，面额3可以表示为面额1,2的

邮票和1+1=2，面额5有两种表示方式min（面额1+面额4，面额2+面额3

），照此类推，递推关系方程不难建立，就拿邮票问题来说，以下是递推

的一种方法：

这种递推方法虽然简单，由于1<=邮票面额<=255,1<=n<=100，因此MAX值

最多可以达到25500,25500次循环里必定还有嵌套循环，因此算法不加优

化，很难在规定时间内得出最优值。这就需要递推的算法优化。一味递推

不寻求算法优化，速度较之搜索提高不少，但一旦数据规模过大，很难在

规定时间内得出最优值。这种递推方法原理是：对于某种要求得到的面额

，判断它能否被题目给定面额整除，再寻找（1<=j<=i），使A[j]+A[i-j]

值最小，求出凑成某种面额最少邮票数，算法虽然简单，但还可以进一步

优化。何不将用m种面额邮票做循环，建立递推关系式：A[i]=MAX(A[l-C[

j]]+1），于是当取到极限值时，程序减少了约1.6\*pow(10,8)次循环，递

推优化作用不言而喻。

## 2.2 实验过程

### 2.2.1 实验思路

#### 2.2.1.1 马的周游问题

　　①从马周游最简单的问题描述开始做起：即马周游只需要遍历揍完棋盘

　　　中的所有格子即可，不要求最后回到起点位置。

　　　解释说明：其实很容易理解到，马周游棋盘，也就是要遍历棋盘中的

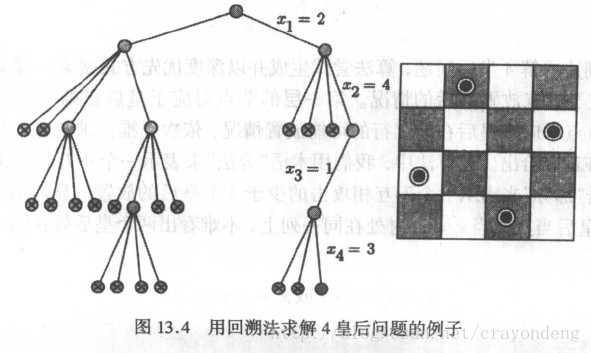
　　　所有格子有且只能一次，那么很显然就是一个图的遍历问题了。

　　　根据八皇后问题中，每个皇后都有8个位置的选择，那么对应来说就

　　　是一个满8叉树（也可以看做是下图）的遍历。下面给出一个四皇后

　　　问题的图解。在这个图中我们就可以清楚的看到图的遍历过程，而且

　　　是深度优先遍历。



　　　那么同样的，关于马周游问题，也是同样要进行图的深度优先遍历过

　　　程，那么深度优先遍历过程可以递归实现，可以非递归实现。

　　②完整解决马周游问题：既要遍历走完棋盘中所有的格子，最后还要回

　　　　　　　　　　　　　到起点

　　　有了上面第一个简化版马周游的解决经验，那么完整解决马周游问题

　　　，无非就是再添加一个限制条件：最后要回到起点。

　　　使用剪枝有3处：

　　　（1）使用Warnsdorff’s rule。在当前位置（Now）考虑下一个位置

　　　　　 (Next)走法最少的那个。作为当前位置（Now）的下一位置（

　　　　　 Next）。

　　　　　 譬如说：下图所示，当前位置现在要确定下一位置，那么就要所

　　　　　 有的下一个位置进行考察，看看假如走到下一个位置，它的下一

　　　　　 位置又有多少种走法，选择下一个位置可能走法最少的作为当前

　　　　　 位置（Now）的下一个位置（Next）。

　　　（2）在进行了第一点的剪枝后，如果可以优先选择的下一个位置不止

　　　　　 一个，则优先选择离中心位置较远的位置作为下一步（即靠近边

　　　　　 边的位置）。

　　　　　 通俗点理解，第一点的剪枝就是走那些位置可能走到机会较小的

　　　　　 ，反正走到的机会小，那么先走完总是好的，要不然你兜一圈回

　　　　　 来，还是要走这一个位置的。

　　　　　 第二点的剪枝就是走法尽量从边边走，然后是往中间靠。

　　　（3）第三点的剪枝，每次都从棋盘的中间点开始出发，然后求出一条

　　　　　 合法路径后再平移映射回待求路径。

　　　　　 所在马周游棋盘，最后还要回到起点，也就是在棋盘中找到一条

　　　　　 哈密顿回路。那么不管你是从哪里开始的，最后都是会在这个哈

　　　　　 密顿回路中的，那么选取的中点的位置也肯定是在这个回路上的

　　　（4）找到这个以中点为起点的哈密顿回路后，根据设定起点在这个回

　　　　　 路中的序号，映射回以这个位置为起点的马周游路线即可。



#### 2.2.1.2 邮票问题

　（1）搜索：每种邮票最多贴200张，总共50种，朴素的深搜规模将达

　　　 到pow（50,200）

　（2）动态规划：

　　　 ①阶段：能够构成每个面值为阶段。比如能构成的面值为1到V，

　　　　　　　 那么总共为V个阶段。

　　　 ②状态：dp[i]表示构成面值i所需要的最少邮票数。

　　　 ③决策：对于样例数据1和3两种面值的邮票：

　　　　　　　 构成邮资0：所需要邮票张数为0张，dp[0]=0；

　　　　　　　 构成邮资1：只能用1分的邮票，所需要邮票张数1张，

　　　　　　　 　　　　　　dp[1]=1；

　　　　　　　 构成邮资2：只能用1分的邮票，所需要邮票张数2张，

　　　　　　　　　　　　　 dp[2]=1；

　　　　　　　 构成邮资3：\*1.若选择使用一张1分的邮票，dp[3]=dp[

　　　　　　　　　　　　　2]+1=3........dp[3-1]+1

　　　　　　　　　　　　　\*2.若选择使用一张3分的邮票，dp[3]=dp[0]

　　　　　　　　　　　　　+1=1........dp[3-3]+1

　　　　　　　　　　　　　dp[3]=min{dp[2]+1,dp[0]+1}=1；

　　　　　　　 构成邮资4：\*1.若选择使用一张1分的邮票，dp[4]=dp[

　　　　　　　　　　　　　3]+1=2........dp[3-1]+1

　　　　　　　　　　　　　\*2.若选择使用一张3分的邮票，dp[4]=dp[1]

　　　　　　　　　　　　　+1=1........dp[3-3]+1

　　　　　　　　　　　　　dp[4]=min{dp[3]+1,dp[1]+1}=2；

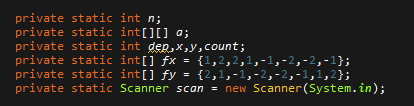
　　　 ④状态转移方程：dp[i]=min{dp[i-a[j]]+1}

　　　　　　　　　　　 i>=a[j] 1<=j<=n f[i]<=k；

### 2.2.2 程序代码

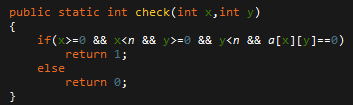
#### 2.2.2.1 马周游问题

　BoardHorse.java——初始化数据



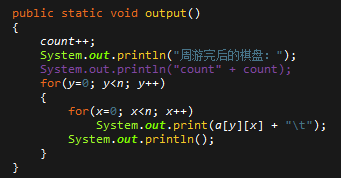
　功能：初始化全局的相关数据

　 BoardHorse.java——check函数



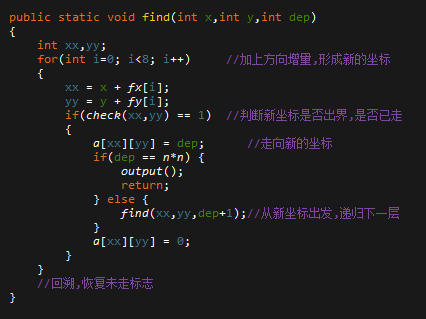
　 功能：判断是否越界

　　BoardHorse.java——output函数



　 功能：用于输出

　 BoardHorse.java——find函数



　 功能：对于求解的回溯算法函数

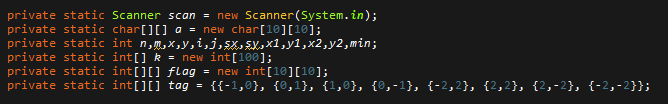
　 BoardHorse.java——main函数



　 功能：主函数，整个程序的开始，用于基础数据的输入

#### 2.2.2.2 寻宝问题

　　 Find.java——初始化数据



　　 功能：初始化全局相关参数

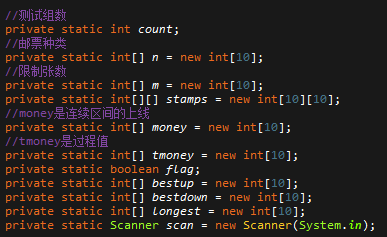
　　 Find.java——主函数



　　 功能：程序的入口以及输入初始化

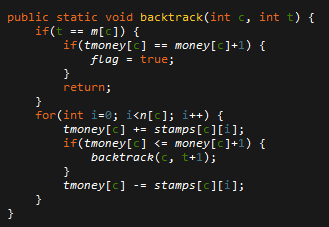
#### 2.2.2.3 邮票问题

　　　Tickit.java——初始化数据



　　 功能：初始化相关全局参数

　　 Tickit.java——backtrack函数



　　 功能：进行相关参数的回溯操作

　　 Tickit.java——main函数

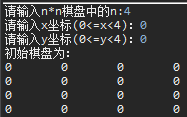


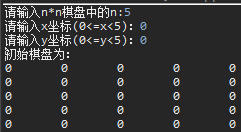
　　 功能：程序的开始以及相关参数的输入操作

## 2.3 实现结果

### 2.3.1 马的周游问题

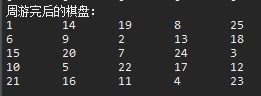
输入：





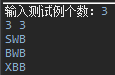
输出：





### 2.3.2 寻宝问题

输入：







输出：

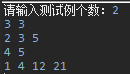






### 2.3.3 邮票问题

输入：



输出：



# 3. 实验总结

收获： ①学会了与回溯算法相关的一些基础知识，虽然了解比较浅显，但

　　　　 是我可以在日常的一些项目中稍微运用一点就足够了。

　　　 ②了解马的周游问题的一些相关概念以及怎么去应用解题之类的知

　　　　 识，可以更加快捷的帮助我提高自身程序算法的效率。也初步地

　　　　 掌握了马的周游问题的时间复杂度这一知识，也帮我更进一步地

　　　　 了解了时间复杂度该如何应用以及计算。

　　　 ③寻宝问题，深化了我对回溯算法的理解与应用，而且考虑算法也

　　　　 可以从多方面的点子去考虑而不是只有单一的一种（按所需要的

　　　　 去硬敲代码，而不去考虑时间复杂度，来精简算法），不同的算

　　　　 法有着不同的时间复杂度，为了提高程序运行的效率以及各个函

　　　　 数和项目之间的的容错率，肯定对其都要经过相对应的比对才可

　　　　 以去提高程序的运行效率和降低程序的容错率，使程序变得更加

　　　　 安全高效。

　　　 ④明白了回溯算法的原理以及相对应题目的解题步骤，根据解题步

　　　　 骤我可以更快更有效的对回溯算法这一类型的题目进行解题，以

　　　　 及回溯算法和动态规划一样遵循着一个原则：大事化小，小事化

　　　　 了。这是这两个算法之间的共性也可能是接下来的几种算法的共

　　　　 性。

　　　 ⑤该算法也提供我的耐心以及强化了我各方面对算法的理解以及应

　　　　 用，从而可以使我对于各个程序所需的功能可以进一步地进行一

　　　　 些强化；并且可以让我的思维更加清晰。

难处： 其实这三个实验都遇到了相对应的困难，虽然收获的很多，但是遇

　　　 到的困难也不少。毕竟作为一个对算法初步了解的我，不可能什么

　　　 都会，更不可能面面俱到，如若有写的不好的地方，请老师指出。

　　　 因为养成了惯性思维，都以为题目说的很简单，而且自己用笔手算

　　　 了一遍跟题目所给的测试数据的结果也一样，所以就很顺的把程序

　　　 一直往下写，直到马的周游问题测试数据的时候始终回不到原点，

　　　 才感觉到了自己应该漏掉了什么东西没有想到，后面经过上网查阅

　　　 资料更加深化地去了解题目才找出了攻破这个题目的关键所在：没

　　　 有考虑到最后结果的判断——是否回到原点。之后经过改良，加上

　　　 一个原点判断语句，整个问题就搞定了。

　　　 至此，我所遇到的难处，全部解决后，我重新回忆了一下做这些算

　　　 法题目的步骤以及思考后所写出来的代码，把二者统一结合在一起

　　　 发现了，不管是哪个算法题目都遵循着大事化小，小事化了这一不

　　　 变的原则，遵循这一原则，可以帮助我快速的理解问题，分析问题

# 4. 参考文献

①[回溯算法\_百度百科]：

<https://baike.baidu.com/item/%E5%9B%9E%E6%BA%AF%E7%AE%97%E6%B3%95/9258495?fr=aladdin#2>

②[回溯法（马周游问题） - 实验报告 – 百度文库]：

<https://wenku.baidu.com/view/49b9268be009581b6ad9eb65.html?re=view>

③[（4）邮票问题\_\_\_\_动态规划 – YzlCoder的记事本 CSDN博客]：

<https://blog.csdn.net/y1196645376/article/details/42197485>

④[回溯法 之 马周游 （马跳日） 问题 – 小邓笔记 – CSDN博客]：

<https://blog.csdn.net/crayondeng/article/details/17174951>