|  |  |  |  |
| --- | --- | --- | --- |
| 实验地点 | 专业班级 | 学号 | 姓名 |
| 宿舍 | 软件工程三班 | 2220202929 | 蔡博宇 |

3-1 n同学关灯问题

一、实验环境

编程语言： Java/Python

IDE环境： IDEA/Pycharm

二、课设题目

有从1到n依次编号的n个同学和n盏灯。1号同学将所有的灯都关掉；2号同学将编号为2的倍数的灯都打开；3号同学则将编号为3的倍数的灯作相反处理(该号灯如打开的，则关掉；如关闭的，则打开)；以后的同学都将自己编号的倍数的灯，作相反处理。请设计程序计算n个同学操作后，哪些灯是打开的？

三、分析问题、问题的模型

题目问题是如何模拟开关灯，以及如何便利的找出倍数关系。

1. 在程序中使用数组来记录每一盏灯的情况，1为关，0为开，同时从2开始遍历，模拟开关灯。
2. 开关状态变换为1-当前状态

四、解题思路、方案

4.1方案实现

4.1.1 选用的算法策略、使用该策略的理由

程序模拟

4.1.2 算法的时间复杂度、空间复杂度分析

时间复杂度：O（n^2） 两层循环

空间复杂度：O（n）开辟空间记录开关灯情况

4.2 Java 实现程序源代码

|  |
| --- |
| public class Lighting Problem {  public static void main(String[] args) {  Scanner input = new Scanner(System.in);  int n = input.nextInt();  print(n);  }  static void print(int n){  int[] array = new int[n+1];  int k;  for (int i = 1; i <= n; i++) {  array[i] = 0;  }  for(int i = 2; i <= n; i++){  k = 1;  while (i \* k <= n){  array[i\*k] = 1 - array[i\*k];  k++;  }  }  for (int i = 1; i <= n; i++) {  System.out.print(array[i]+" ");  }  }  } |

4.2.1程序运行结果：



4.3 python 实现

|  |
| --- |
| def factorialLoop(n):  S = 1.0000  sign = 1  t = 1  for i in range(2, nn):  sign = -sign  t = t \* ((2 \* i - 2) \* (2 \* i - 1))  S += sign / t  for i in range(n):  array[i] = 0  for( i in range(n):  k = 1  while (i \* k <= n):  array[i\*k] = 1 - array[i\*k]  k++  for I in range(n):  print(array[i]+" ")  if \_\_name\_\_ == '\_\_main\_\_':  n = int(input())  print(factorialLoop(n)) |

4.3.1 程序运行结果



五、结论

本题考查对数组模拟开关的计算，同时也对如何巧妙应用下标来模拟开关的状态，以及1，0之间开关的转换来模拟状态转换。再一次感受到了标志位的妙用，也感受到了算法的魅力。

3-2找出次品问题

一、实验环境

编程语言： Java/Python

IDE环境： IDEA/Pycharm

二、课设题目

有10箱产品每箱有1000件，正品每件100克，其中的几箱是 次品，次品每件比正品轻10克。请设计程序用秤只称一次，就找出哪几箱是次品。

注：如果某箱是正品，整箱是正品；如果某箱是次品，整箱是次品。

三、分析问题、问题的模型

本体难点在于如何区分出找出的次品是哪一项，在计算机中用二进制可解决此问题，可用2的幂来表示每一箱，最后可累加得到相减得到缺少的质量，再通过分析得到缺省的箱号。

四、解题思路、方案

4.1方案实现

4.1.1 选用的算法策略、使用该策略的理由

循环

4.1.2 算法的时间复杂度、空间复杂度分析

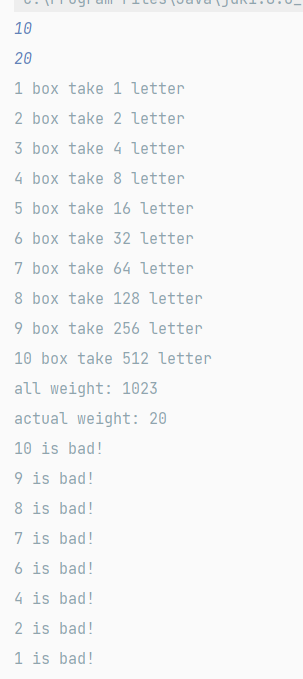
时间复杂度：O（n）

空间复杂度：O（0）

4.2 Java 实现程序源代码

|  |
| --- |
| public class box Question {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n = sc.nextInt();  int w2 = sc.nextInt();  print(n,w2);  }  static void print(int n,int w2){  int w1 = 0;  int t = 1;  int k;  for (int i = 1; i <= n; i++){  System.out.println(i+" box take " + t +" letter");  w1 = w1 + t;  t \*= 2;  }  System.out.println("all weight: " + w1);  System.out.println("actual weight: " + w2);  w2 = w1 - w2;  while(w2 != 0){  k = (int) (Math.log(w2)/Math.log(2));  System.out.println(k+1 + " is bad!");  w2 = (int) (w2 - Math.pow(2,k));  }  }  } |

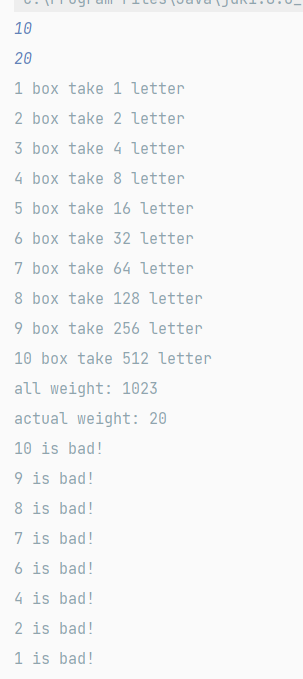
4.2.1程序运行结果：



4.3 python 实现

|  |
| --- |
| def box Question ():  for i in range(1000):  s = i  for j in range(1,i):  if i % j == 0:  s -= j  if s == 0 and i != 0:  print(i)  w2 = w1 - w2  while(w2 != 0):  k = (int) (Math.log(w2)/Math.log(2))  print(k+1 + " is bad!")  w2 = (int) (w2 - Math.pow(2,k))    if \_\_name\_\_ == '\_\_main\_\_':  box Question () |

4.3.1 程序运行结果



五、结论

本题本质上是区分问题，这里就巧妙运用了计算机自身二进制的设计，其实当今世界自然语言处理中的向量化或者计算机视觉中图片的张量化的思想一样。通过不同的累加数据来判断缺省箱号，也符合人的思考。

3-3兔子生兔子问题

一、实验环境

编程语言： Java/Python

IDE环境： IDEA/Pycharm

二、课设题目

一对免子从出生后第三个月开始，每月生一对小 免子。小免子到第三个月又开始生下一代小免子。假若免子 只生不死，一月份抱来一对刚出生的小免子，编程计算一年中每个 月各有多少只免子。

三、分析问题、问题的模型

迭代模型的使用，将问题的迭代公式找出即可解决。

1）将1年划分为5次循环，以每三个月为计算周期

2）初始1月，2月为1

3）利用迭代公式累加即可得到每月兔子对数

a = a + b;

b = a + b;

四、解题思路、方案

4.1方案实现

4.1.1 选用的算法策略、使用该策略的理由

迭代法

4.1.2 算法的时间复杂度、空间复杂度分析

时间复杂度：O（n）

空间复杂度：O（0）

4.2 Java 实现程序源代码

|  |
| --- |
| public class rabbit {  public static void main(String[] args) {  print();  }  static void print(){  int a = 1;  int b = 1;  int i = 2;  System.out.println(a);  System.out.println(b);  for (i=1;i<=5;i++){  a = a + b;  b = a + b;  System.out.println(a);  System.out.println(b);  }  }  } |

4.2.1程序运行结果：



4.3 python 实现

|  |
| --- |
| def rabbit(n, m):  if n == 1 or m == 1:  return 1  for i in range(5):  a = a + b  b = a + b  print(a)  print(b)  if \_\_name\_\_ == '\_\_main\_\_':  nn = input()  nn = int(nn)  print(divider(nn, nn)) |

4.3.1 程序运行结果



五、结论

本题对于迭代公式的考察，本来可以将问题划分为多种迭代模型，例如以三个月为一周期进行迭代c=a+b,a=c+b,b=a+c，而这里使用了c这个变量，会增加算法的空间复杂度，同时循环次数增多，也增加了时间复杂度，出于两方面的考虑将一年划分为五次循环计算，空间与时间复杂度都较低，此题12月兔子对数从1增加到144，也让我发现生物不加管控的繁殖的可怕程度。

3-4 沙漠穿越问题

一、实验环境

编程语言： Java/Python/C++

IDE环境： IDEA/Pycharm/sublime

二、课设题目

用一辆吉普车穿越1000公里的沙漠。吉普车的总装油量 为500加仑，耗油半为1加仑/公里。由于沙漠中没有油库，必须先用这辆车在沙漠中建立临时油库。该吉普车以最少的耗油量穿越沙漠，编程计算应在什么地方建油库，以及各处的贮油量。

三、分析问题、问题的模型

3.1假设：

1. 每次车都载满油

2. 到达油库时刚好没有油

3. 假设每次最优运输奇数次

3.2 解法：

1. 确保最后距离500米有一个油库有500 L油

2. 那倒数第二个油库设在哪会有最优解呢？

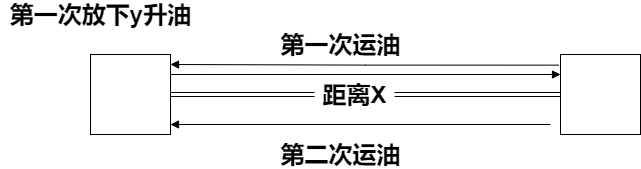
3. 可知至少得运两次，第一个油库才会有500 L油

3.3最优解法：

1. 设距离下一个油站为 $x$

2. 第一次放下油 $y$

3. 如图所示：



3.4列方程：

500-2x-y=0

y+500-x=500

\* 1式为第一次运油过程

\* 2式为第二次运油过程

可解得，x 与 y 相同，为 ${\Large \mathbf{\frac{500}{3}} } $

\* 并可知储油为 500 + 500 =1000 (两次运输)

\* 由此递推可知第三个站点的 x

四、解题思路、方案

4.1方案实现

4.1.1 选用的算法策略、使用该策略的理由

递归

4.1.2 算法的时间复杂度、空间复杂度分析

时间复杂度：O（n）

空间复杂度：O（0）

4.2 C++实现程序源代码

|  |
| --- |
| #include<iostream>  #include<cstdio>  using namespace std;  void jeep(){  int k = 1;  int oil = 500; //每轮载油量  int dis = 500; //距离从第一个站点开始计算  do{  printf("储油点 %d 距出发点距离 %d 储油量 %d\n",k,1000-dis,oil);  k++; // 从第二个站点开始计算  dis = dis + 500/(2\*k-1); // 计算总距离 dis = dis + x  oil = 500 \* k; // 计算储油量  }while(dis < 1000); // 超过1000则完成运油  oil = 500\*(k-1)+(1000-dis)\*(2\*k-1);  // 计算总储油量  // 第一次运输不算  // 由往返次数可知每个站点的油量 500 \* n (n为往返次数)  // 减去多余的距离，2 \* k -1 为原本第k个站点需要的运油量  printf("%d\n",oil);  return;  }  int main()  {  jeep();  return 0;  } |

4.2.1程序运行结果：

|  |
| --- |
| 储油点 1 距出发点距离 500 储油量 500  储油点 2 距出发点距离 334 储油量 1000  储油点 3 距出发点距离 234 储油量 1500  储油点 4 距出发点距离 163 储油量 2000  储油点 5 距出发点距离 108 储油量 2500  储油点 6 距出发点距离 63 储油量 3000  储油点 7 距出发点距离 25 储油量 3500  3380  [Finished in 436ms] |

4.3 python 实现

|  |
| --- |
| def jeep():  k = 1  oil = 500//每轮载油量  dis = 500 //距离从第一个站点开始计算  do{  print("储油点 %d 距出发点距离 %d 储油量 %d\n",k,1000-dis,oil)  k++// 从第二个站点开始计算  dis = dis + 500/(2\*k-1) // 计算总距离 dis = dis + x  oil = 500 \* k // 计算储油量  }while(dis < 1000)// 超过1000则完成运油  oil = 500\*(k-1)+(1000-dis)\*(2\*k-1)  // 计算总储油量  // 第一次运输不算  // 由往返次数可知每个站点的油量 500 \* n (n为往返次数)  // 减去多余的距离，2 \* k -1 为原本第k个站点需要的运油量  print("%d\n",oil)  return  if \_\_name\_\_ == '\_\_main\_\_':  jeep() |

4.3.1 程序运行结果



五、结论

本题是倒退法+数学分析的一种结合，利用极限最优解的思想以及分区解方程求最优的解法让本题变得十分有意思，本题也让我且身体会到算法对于数学分析的要求性，同时通过解方程的思想和画图的直观性体会到对于解决困难问题分析的重要性。