目录

1.使用说明 2

1.1 实验环境 2

1.2 使用说明 2

2 设计说明 4

2.1 子程序 copy 4

2.2 子程序 multiply 5

2.3 子程序 add 5

2.4 主程序 sum 7

2.5 形式化表示 8

3.算法说明 8

3.1 main.ui 说明 8

3.2 statechange.py 说明 9

3.3 main.py 函数说明 9

4.总结 10

1.使用说明

1.1 实验环境

操作系统：Windows10

依赖环境：Python3.6，PyCharm

编程语言：Python

1.2 使用说明

运行 main.py 文件(需要安装 PySide2)，显示如下界面：

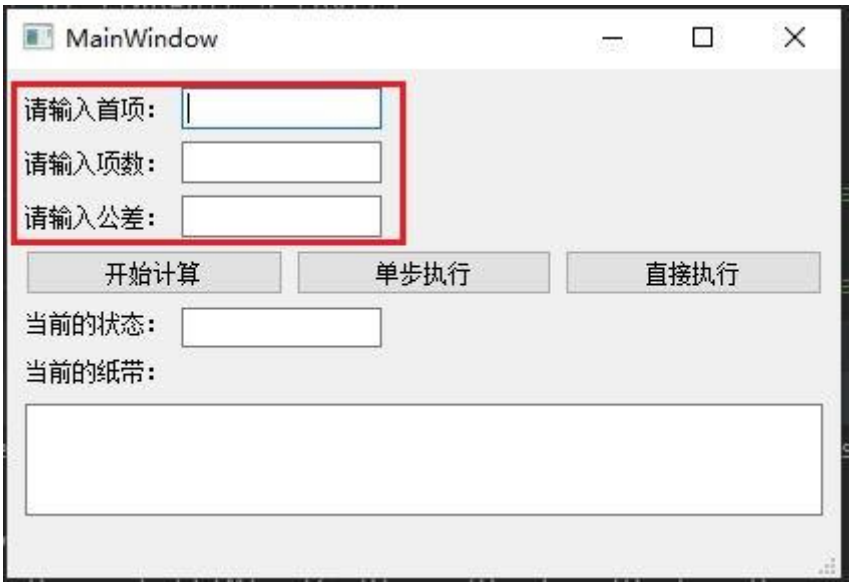

图 1 进入界面

在前三个输入框分别输入需要计算的等差数列的首项 a、项数 n 和公差 m，点击开始计算，输出当前为“o”，并且生成初始纸带为*B1^{a}B1^{n}B1^{m}BBB 的形式，*代表图灵机准备读取字符的位置。

图 2 执行按钮

点击单步执行，相当于执行一次状态转移函数，当前的状态、纸带和读取位置均发生相应变化。如下图所示，在点击若干次单步执行以后，图灵机的当前状态为“a”，纸带的状态如下。

图 3 执行过程

由于在本程序中实验跳转的步数太多，所以可以点击“直接执行”，得到最终计算结果，此时纸带上只剩下计算的结果且指针指向结果的第一个 1。

图 4 执行结果

2 设计说明

由于在本实验中，根据等差数列的求和的算法，在这里为了使程序简化，将重复多次的计算过程封装为子程序，方便调用和使代码简洁。

2.1 子程序 copy

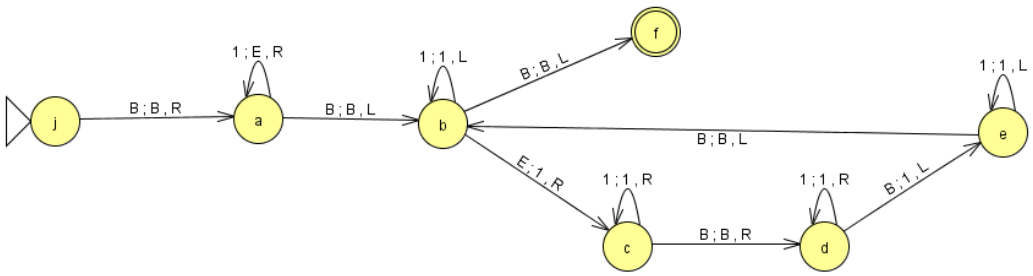

图 5 copy 子程序

在本次设计中，首先要实现图灵机最基本的复制字符的功能，所以设计子函数 copy 完成 1^m 的复制功能。经测试，本程序可以实现复制字符串 1^m 的功能。

2.2 子程序 multiply

在完成子程序 copy 之后，由于在计算$a_t = a_1 + m * n$中需要调用乘法运算，所以在这里多次调用 copy 完成乘法 multiply 的计算。

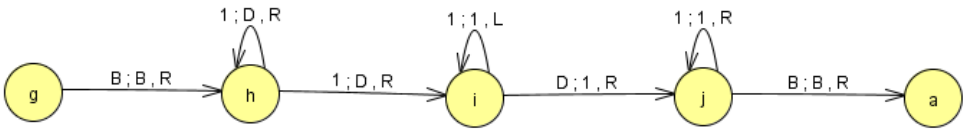

图 6 multiply 子程序的设计

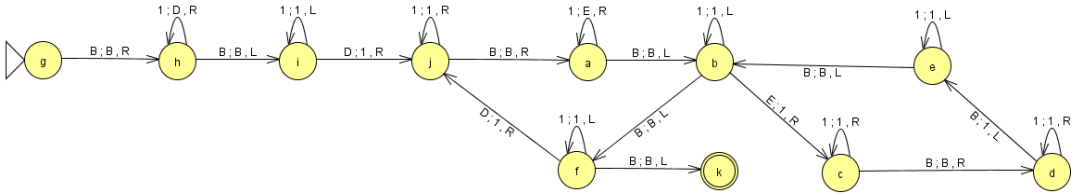

图 7 multiply 的完整图

2.3 子程序 add

为了图灵机设计的简洁性，在本实验中设计计算等差数列和时从a_n依次向前加到a_1。计算公式和递推公式如下：

$$S'_n = a_n + a_{n-1} + \cdots + a_2 + a_1$$

$$S'_{n-t+1} = S'_{n-t} + a_t$$

子程序 add 主要完成上面第二式的加法计算。根据等差数列的计算公式：

$$a_t = a_1 + (t-1)d = a + (t-1)m$$

子程序 add 的设计思路在S'_{n-t}的基础上先加上 a，再调用(t-1)次 copy 子程序，得到S'_{n-t+1}的结果。设计的图灵机模型如下图所示：

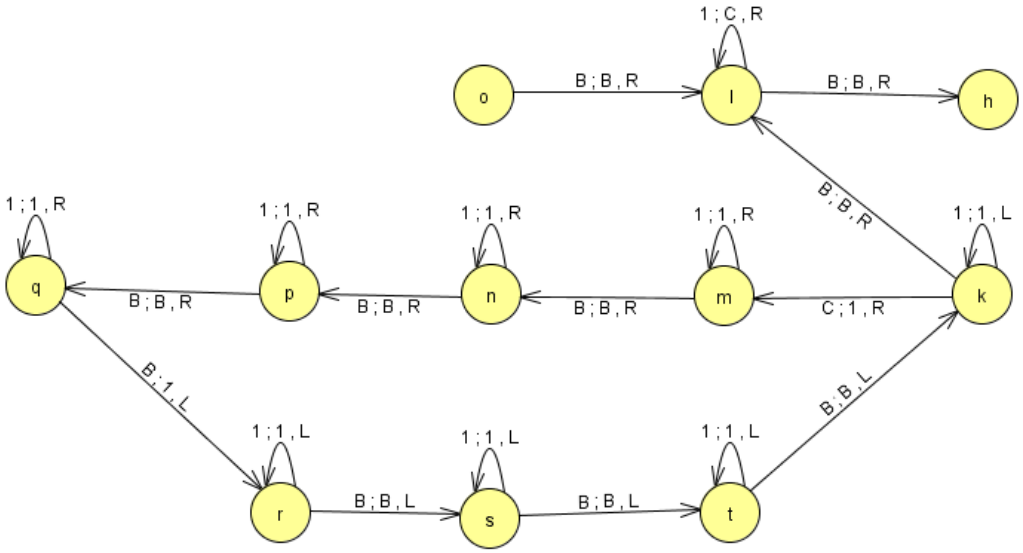

图 8 add 子程序的设计

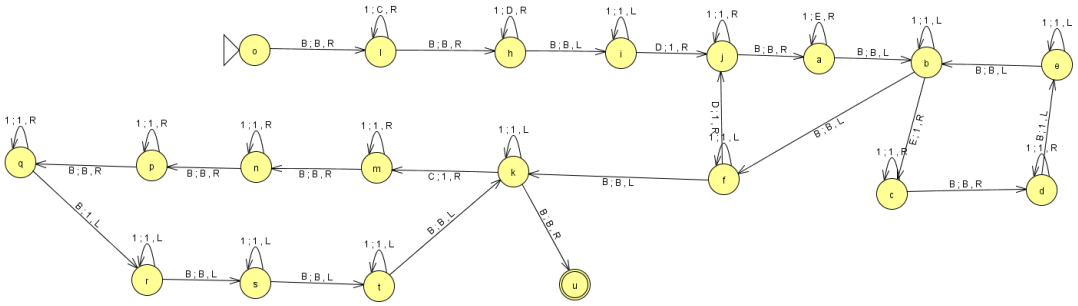

图 9 add 子程序的完整图

2.4 主程序 sum

在完成上面计算过程后，每调用一次 add 完成求和过程中的一次加法；通过多次循环嵌套，完成整个等差数列。整个图灵机的设计和子程序之间的嵌套关系见下图。

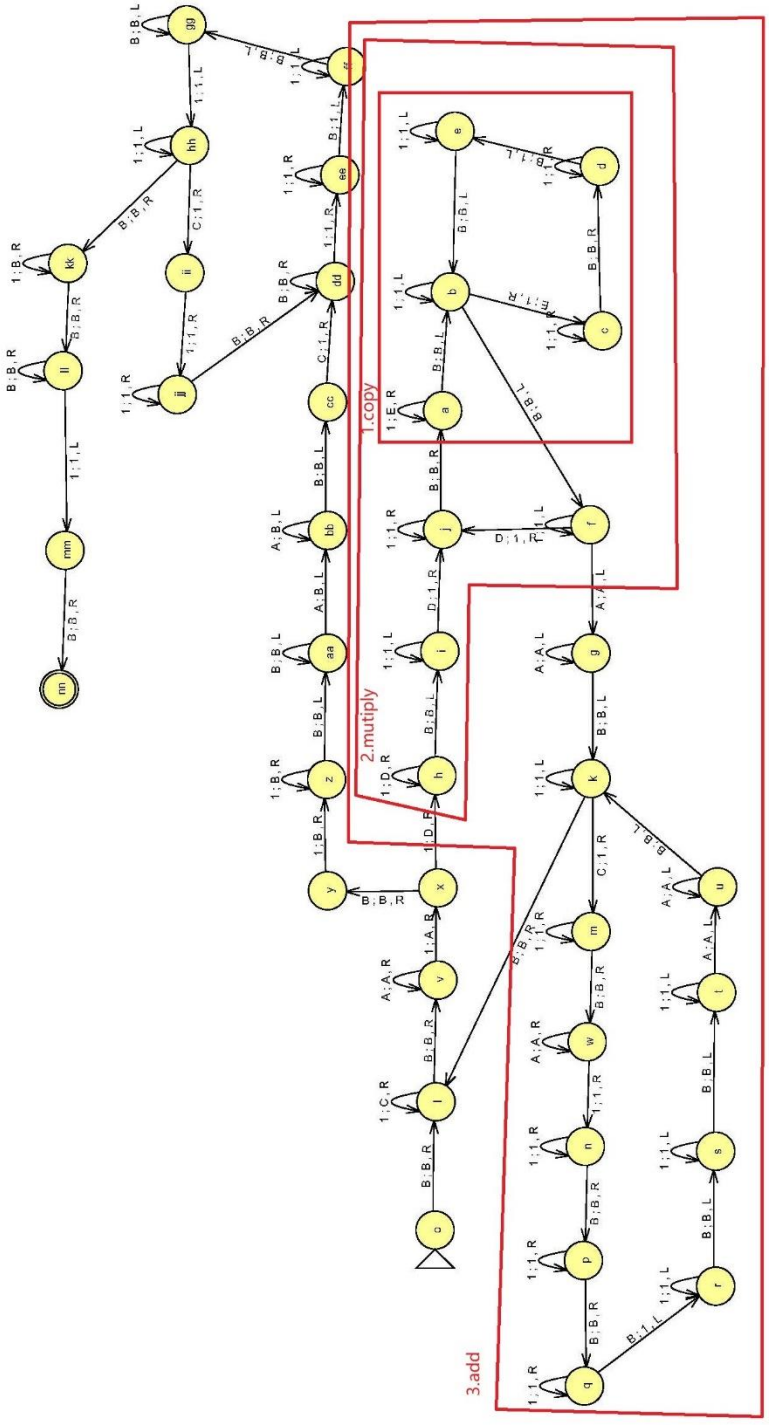

图 10 图灵机

2.5 形式化表示

完成上述设计过程后，将整个图灵机形式化表示为：$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- $Q = \{a, b, c, \ldots\ldots, x, y, z, aa, bb, cc, \ldots\ldots, ll, mm, nn\}$共 40 个状态
- $\Sigma = \{1\}$
- $\Gamma = \{1, A, B, C, D, E\}$
- $q_0 = o$
- $B = B$
- $S_f = \{nn\}$
- δ为转移函数，由于转移函数太多且已经给出图灵机的状态转移图，故不将其一一列出

3.算法说明

在本程序设计中，主要用 Python 语言写成，主要包括 main.py 和 statechange.py 的脚本文件，以及由 QtDesigner 设计的图形界面 main.ui 文件。

3.1 main.ui 说明

此文件为图形界面的生成文件，提供图灵机输入、输出的图形界面表示。具体如下图所示：

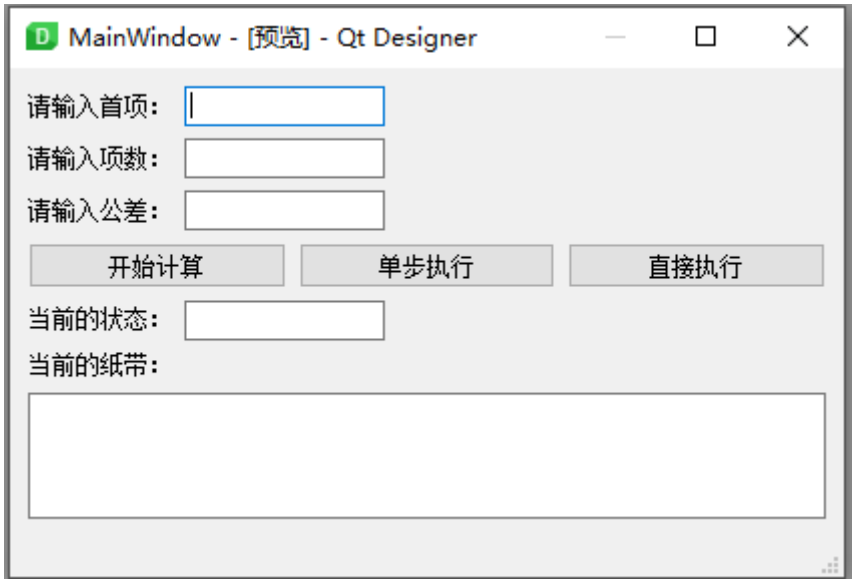

图 11 ui 界面设计

3.2 statechange.py 说明

statechange.py 主要是图灵机的状态转移函数。由于设计的图灵机是一个标准的图灵机，所以只需要根据状态转移图对应编写即可。利用两个 if…elif…循环完成状态转移下面以状态 h 为例，说明状态转移函数。

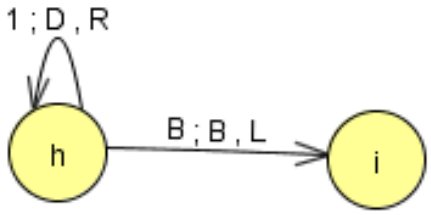

图 12 状态“h”图示

对应 h 的两个状态转移函数为：

$$\delta(\mathrm{h}, 1) = (\mathrm{h}, \mathrm{D}, \mathrm{R})$$

$$\delta(\mathrm{h}, \mathrm{B}) = (\mathrm{i}, \mathrm{B}, \mathrm{L})$$

对应的代码段为

`elif state == "h":`
`  if tape[point] == "1":`
`  tape[point] = "D"`
`  state = "h"`
`  point += 1`
`  elif tape[point] == "B":`
`  tape[point] = "B"`
`  state = "i"`
`  point -= 1`
	判断当前在状态“h”
	1.读取字符“1”
	2.写入字符“D”
	3.跳转到状态“h”
	右移
	1.读取字符“B”
	2.写入字符“B”
	3.跳转到状态“i”
	左移

可以看到，在上面所举的例子中，状态转移的过程与图灵机的作用完全相同，主要分为 3 个步骤：1.读取字符；2.写入字符；3.状态跳转、指针移动。这样的设计大大降低了编程的难度。

在整个计算过程中，由状态“o”开始，由状态转移函数决定转移状态；状态达到“nn”状态结束运行。

3.3 main.py 函数说明

主程序主要使用 Turing 类实现的计算等差数列和的计算，现将其主要属性的

方法介绍如下：

Turing	
属性	说明
a	等差数列的首项
n	等差数列的项数
m	等差数列的公差
tape	图灵机的纸带
state	图灵机的状态
point	图灵机读取纸带上的位置
方法	说明
start	获取参数并初始化图灵机
step_one	单步计算，完成一次状态转移
step_all	完成全部计算，输出结果

4.总结

本次实验设计了一个可以完成等差数列求和的图灵机的设计，其中使用了 1 条纸带，设计了 40 个状态，完成了从输入到输出的计算任务。界面显示了图灵机在计算过程中的状态转移、纸带的变化以及图灵机读取位置的变化。本本实验采用了 1 进制，因此纸带会比较长，而且状态数比较多，状态迁移也比较复杂。另外在图灵机的设计过程中，采用了子程序的设计思想，由难到易，完成了图灵机的设计。