图灵机解释器

该作业为图灵机作业的第一次小作业，我们将关注以下几件事情

- 图灵机是什么?
- 图灵机的定义是什么?
- 如何构造一个图灵机?
- 如何在图灵机和字符串之间相互转换?

1. 实验背景

正在浏览网页的这台电脑为什么可以正常运行？为什么将CPU、内存、硬盘等等硬件恰当的组合在一起就能让计算机跑起来？计算机究竟可以解决哪些问题？学过计基的话，我们知道可以把计算机抽象成冯诺伊曼模型， 如果我们再抽象一次呢？计算机的本质到底是什么?

有限状态机

从另一个角度看计算机

考虑一下计算机中所有设备（除了IO设备）的用途，基本上可以归结成两类，一类是存储数据，一类是在数据上进行运算，我们能否将计算机看成只有CPU和内存的机器呢？学过DLX的大家应该可以很容易的理解这件事情，在编写DLX代码的时候，你其实是在一个有限的内存空间和寄存器空间中存储读取数据，并且有一个能够执行你的DLX代码的计算设备从内存中取出指令，然后运行，再修改内存和寄存器。

寄存器是必要的吗？其实不是必要的，因为我们可以将寄存器当成内存的一块特定的区域，这样，我们眼中的计算机就是一个CPU加上一个内存，再次抽象，其实计算机就是一个计算设备与一个存储设备的结合，在现实世界中的计算机，他们所拥有的存储空间往往是有限的，也就是说，一个计算机的存储设备中能出现的0,1bit的情况是**有限的**，如果我们将所有的情况看成是计算机不同的状态，那么计算设备做的事情，其实就是根据计算机现在的状态，执行一系列计算，然后将计算机的状态修改成下一个状态。这种具有现态、动作、次态的模型通常被称为状态机，状态机中的信息都是有限的，也就是说，状态的数量和能做的动作都是有限的。

什么是图灵机?

图灵机就是一个拥有一个计算设备和一个存储设备的有限状态机。当然有限状态机还有很多，比如DFA，PDA等等。在这一页我们先非形式化的描述一下图灵机是什么， 下面这段是Wiki上给予的描述

> A **Turing machine** is a mathematical model of computation that defines an abstract machine, which manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, given any computer algorithm, a Turing machine capable of simulating that algorithm's logic can be constructed.
>
> The machine operates on an infinite memory tape divided into discrete "cells". The machine positions its "head" over a cell and "reads" or "scans"the symbol there. Then, as per the symbol and the machine's own present state in a "finite table"of user-specified instructions, the machine (i) writes a symbol (e.g., a digit or a letter from a finite alphabet) in the cell (some models allow symbol erasure or no writing),then (ii) either moves the tape one cell left or right (some models allow no motion, some models move the head),then (iii) (as determined by the observed symbol and the machine's own state in the table) either proceeds to a subsequent instruction or halts the computation.

简单来说，图灵机有一条水平的无限长的磁带(Tape)，这个是图灵机的内存(Memory)，同时图灵机具有有限多个状态，这些状态和内存的总和共同构成了图灵机的存储设备。图灵机有一个磁头(Head)，它所在的位置即为图灵机当前能够读取到的磁带上的符号的位置。图灵机还有一个计算设备，它能够根据当前磁头指示的符号，当前图灵机的状态来做出一个动作，动作会产生三个结果

- 当前磁头指示的位置的符号发生变化（有可能不变）
- 磁头的位置会发生变化（向左或者向右移动一个"cells"，或者不动）
- 图灵机的状态发生变化（可能会从状态A走到状态A，也就是状态没发生改变）

以上就是对图灵机的非形式化描述，其实图灵机的概念还是比较简单的，但是这么一个简单的机器却是计算理论中重要的组成部分。

图灵机的定义

图灵机由以下七个集合共同描述

1. 有穷的状态集合 Q
2. 一个输入字母表(input alphabet) Σ
3. 磁带上的字母表(tape alphabet) $\Gamma(\text{constains } \Sigma)$
4. 迁移函数(transition function) δ
5. 开始状态(start state) $q_0 (q_0 \in Q)$
6. Blank Symbol B $(\in \Gamma - \Sigma)$
    1. 磁带上所有除了输入以外的字符都被初始化为Blank
7. 终止状态(final state) $F \subseteq Q$

符号约定

我们先约定一下后文中可能出现的符号的一般含义，如果没有特殊说明，那么$a, b, \ldots$通常用来表示输入符号，$\ldots, X, Y, Z$常用来表示磁带上的符号，$\ldots, w, x, y, z$表示输入符号组成的字符串，$\alpha, \beta, \ldots$是磁带上的符号组成的字符串。

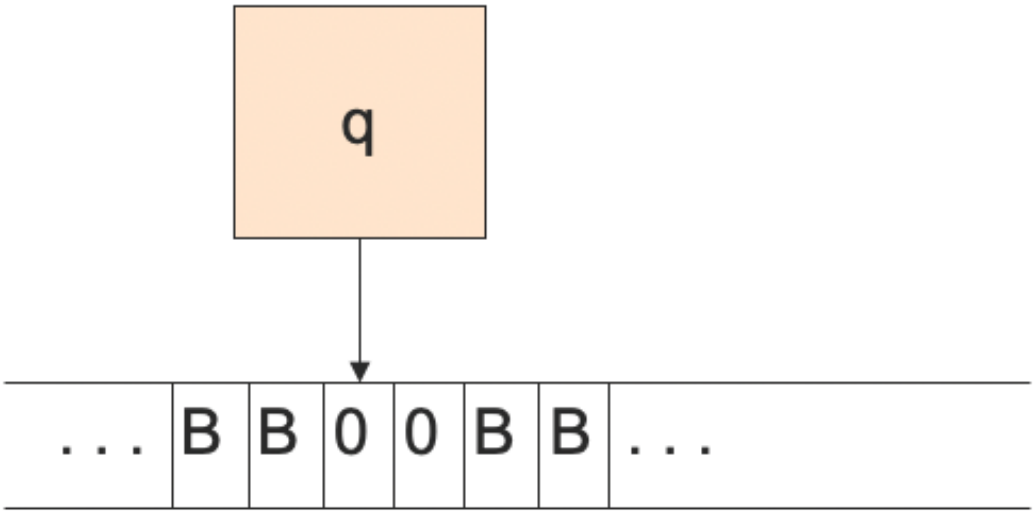
函数与磁带的进一步定义

1. 函数接受两个参数，一个是状态q，另一个是磁带上的符号Z, $\delta(q, Z)$的结果是一个三元组(p, Y, D)，表示在状态 q 读到字符 Z 的时候，转移到状态 p，并且把 Z 修改成 Y，最后向 D（L or R）方向移动一格。在图灵机中如果想进行状态转移，现在的状态和符号必须和某个转移函数相符合，如果不符合就会进入停机。
2. 最初，TM有一个由一串**输入符号组成的磁带**，他就是图灵机的**输入**，它被两个方向上无限多的空格包围着。TM处于初始状态时，磁头位于最左边的输入符号处。

例子

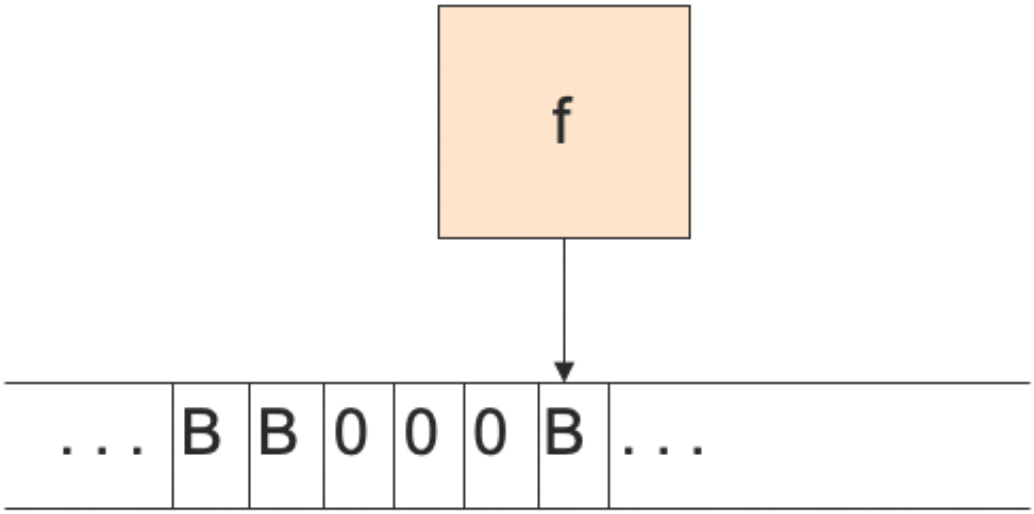
$$
\begin{gathered}
Q = \{q(start), f(final)\} \\
\Sigma = \{0, 1\} \\
\Gamma = \{0, 1, B\} \\
\delta(q, 0) = (q, 0, R) \\
\delta(q, 1) = (f, 0, R) \\
\delta(q, B) = (q, 1, L)
\end{gathered}
$$

假设我们的输入是下面的磁带



初始的时候

那么结果应该是



结束的时候

会顺次调用函数 $\delta(q,0),\delta(q,0),\delta(q,B),\delta(q,0),\delta(q,1)$，我们也称上述的过程为推导。

一个 ID 就是对图灵机和当前磁带情况的一次快照，它记录了当前所有的状态，我们用一个字符串 $\alpha q\beta$ 来表示，其中 $\alpha\beta$ 表示了磁带上最左边的和最右边的 $nonblanks$ 的所有字符，就拿上图举例， $\alpha\beta$ 表示的就是 000 。而 q 表示的是图灵机当前的状态，它紧跟在当前磁带被扫描到的符号的左侧。$\vdash$ 是推导的意思，表示使用了一次迁移函数。$\vdash^*$表示使用了零次，一次或者多次迁移函数。

上述推导的过程用快照和 $\vdash$ 表示就是 $q00\vdash 0q0\vdash 00q\vdash 0q01\vdash 00q1\vdash 000f$

在**(确定性)图灵机**推导的过程中，要么有一个迁移函数可以选择，要么没有任何迁移函数可以选择。我们默认图灵机指的是确定性图灵机

接受(Accept)

注意到，一个图灵机的定义中是不包含对于磁带具体的定义的，仅仅只是定义了磁带上能够出现的字符，对于一个图灵机来说，具体的磁带是它的输入。有了输入后，图灵机会开始不停的运转，什么时候图灵机会停下来呢?

对于图灵机来说，停止意味着接受了这个磁带，下面我们给出**接受**的定义

图灵机有两种接受的方式，一种是通过 $FinalState$接受，也就是说我们在推导的过程中路过了$FinalState$ ，那么这个语言就可以被图灵机所接受

$$L(M) = \{w|q_0w \vdash^* I, where\ I\ is\ an\ ID\ with\ a\ final\ state\}$$

另一种则是用过 $Halt$ 来接受，也就是说，我们在推导的过程中无路可走了，那么图灵机就会停机

$$H(M) = \{w|q_0w \vdash^* I, and\ there\ is\ no\ move\ possible\ from\ ID\ I\}$$

以上是完成图灵机解释器需要了解的图灵机基础知识，下面我们开始介绍本次实验的内容

2. 实验要求

本次实验中，同学们需要实现两个方法

```
//TODO
public TuringMachine(String s)
{

}

//TODO
@Override
public String toString()
{
  return "todo";
}
```

第一个方法，需要同学们完成一个图灵机的构造函数，函数的参数是一个**给定格式**的字符串，第二个方法是toString方法，需要同学们能够将图灵机打印出来，变成一个给定格式的字符串，具体的格式在最后附上。toString函数做的事情也叫**序列化**。

对于构造方法，我们要求他能够做到以下几件事情

- 正确识别文末给定格式的字符串，并且生成图灵机对象
- 若输入的字符串有错误，需要在标准错误流中输出`Error: [line]`，其中，`line`是错误的行数，你需要报告出所有的错误
    - 在本次实验中，错误仅包含以下情况
        - 漏写括号
        - 出现了无法解析的内容，即不属于任何最后给出的格式的内容

    - 某个需要的七元组信息不在输入当中（此时不需要输出错误行数，需要输出`Error: lack [info]`，info处可能出现的情况为Q,S,G,q0,F,B,N,Delta）
    - 输入的Delta函数读取的符号和写回的符号长度不同
        - 如果Delta函数相关的输入全都错误，也相当于没有输入Delta的任何信息，标准错误流中还需要输出`Error: lack D`

我们的输入一定会遵守以下假设

- 空格符号B只有一个字符。
- 和状态相关的信息，均是不定长的字符串，比如#Q中可以有一个状态为"abcasd"，也可以是"0"。
- 七元组中的任何一个信息（除了Delta以外）只会在输入中出现一次，也就是说不会有两行都是以#Q开头。
- 信息并不一定会在一行的开头给出，#Q的前面可能会有很多空格。
- Input symbols 和 tape symbols集合中的元素长度均只有一个字符
- 磁带数不会超过Int能表示的范围

参考用例

```
; This example program checks if the input string is a a_nb_n.
; Input: a string of a's and b's, e.g. 'aaabbb'
; the finite set of states
#Q = {0, 1, 2, 3, 4}

; the finite set of input symbols
#S = {a, b}

; the complete set of tape symbols
#G = {a, b, _}

; the start state
#q0 = 0

; the set of final states
#F = {4}

#B = _

 #N = 1

; the transition functions

; State 0: start state
#D 0 a _ r 1
#D 0 _ _ r 4

; State 1:
#D 1 a a r 1
#D 1 b b r 1
#D 1 _ _ l 2
```

```
; State 2:
#D 2 b _ l 3

; State 3:
#D 3 a a l 3
#D 3 b b l 3
#D 3 _ _ r 0
```

输入这些内容或者我们预定义好的几个集合的时候，会得到一个TM对象，当调用该对象的toString方法的时候，我们不对里面的内容的顺序有任何要求，只需要不同的信息都在不同行即可，比如对于上面输入得到的图灵机，调用他的toString方法时，你输出的信息可以是下面这样，不要输出任何多余的内容，包括注释。

还有需要注意的是，集合中任意两项之间不要留有空格，每一行的信息需要在一行的开头就给出，也就是说#Q需要在所在的行的最开头，不要留有空格，最后一行不要有换行符，#{Q\S\G\N\F\q0\B}和=号，=号和{之间都需要留出一个空格。

```
#Q = {0,1,2,3,4}
#S = {a,b}
#G = {a,b,_}
#N = 1
#D 3 b b l l
#D 1 b b r r
#D 1 a a r r
#D 0 a _ r r
#F = {4}
#q0 = 0
#B = _
#D 3 _ _ r r
#D 3 a a l l
#D 0 _ _ r r
#D 1 _ _ l l
#D 2 b _ l l
```

如果对于以下输入

```
; This example program checks if the input string is a a_nb_n.
; Input: a string of a's and b's, e.g. 'aaabbb'
; the finite set of states
#Q = {0, 1, 2, 3, 4

; the finite set of input symbols



; the complete set of tape symbols
#G = a, b, _}

; the start state
```

```
#q0 = 0

; the set of final states
#F = {4}

#B = _

; the transition functions

; State 0: start state
0 a _ r 1
0 _ _ r 4

; State 1:
1 a a r 1
1 b b r 1
1 _ _ l 2

; State 2:
2 b _ l 3

; State 3:
3 a a l 3
3 b b l 3
3 _ _ r 0
```

需要给出输出

```
Error: 4
Error: 10
Error: lack S
Error: lack N
```

输出没有顺序要求

代码指导

```
.
├── README.md
├── README.pdf
├── TuringMaching.iml
├── pom.xml
└── src
    ├── main
    │   ├── java
    │   │   └── edu
    │   │       └── nju
    │   │           ├── Tape.java
```

```
    │   │           ├── Track.java
    │   │           ├── TransitionFunction.java
    │   │           └── TuringMachine.java
    │   └── resources
    └── test
        └── java
            └── TMTest.java
```

这个是框架代码的结构，我们已经给大家准备了几个类，大家可以把解析字符串的任务分散到不同的函数中，十分不建议大家全都写在构造函数里面，这样会增加一个方法内部的复杂性。具体每个类的作用，大家需要自己进行理解。

对于公共的部分，如果能够抽象出来会更好，同样功能的代码散落在不同的位置，会给调式和后续的修改带来一定的难度。

因为输入的是一个字符串，自然免不了很多和字符串相关的操作，相比于Python来说，java的字符串操作稍微困难一些，但是也是可以比较轻松的完成本次任务的。建议大家先把字符串中的注释删掉，然后按照给定的格式一个个解析出来。本次实验应该不会频繁的拼接字符串，但是会对字符串做分割，大家可以研究一下`String`类已经提供好的方法，比如`subString`、`replaceAll\replaceFirst\replace`方法等。

在解析集合的时候，大家应该会用到Java中的`Collection`库，这个库提供了ArrayList、LinkedList、Map、Set等等一系列非常有用的数据结构，能够帮助大家更加高效的构建TM对象。大家可以了解一下Java中的函数式编程，比如stream(),forEach()等等，包括consumer，producer等，就算现在用不到，以后也是会用到的。

这里给大家添加了一个常用的jar包`common-lang3`，它是开源组织apache给Java的lang库做的一个增强，里面提供了类型StringUtil类等可以辅助大家编程的方法，当然，自己造轮子也是一件趣味无穷的事情。

本次作业的难度不大，只需要按照给定的格式把信息全部解析出来就行，解析的时候不要忘记我们需要判断的几个错误，如果没有框定错误的范围，需要做的工作会非常的繁琐，但是框定了错误的情况后，大家只需要按照给出情况一点点加在代码里就行。

平时大家的println语句都是在标准输出流中打印信息，想要在标准错误流中打印信息，需要调用`System.err.println()`即可。

生存指南

及时测试

> “任何没有测试过的代码都是错误的“

在大家编写完某个函数或者某个功能后，我强烈建议大家先对这个功能进行一次测试，而不是草草提交，根据我们OJ的结果来判断这个函数是否正确，在很多时候你是需要自己编写很多测试用例来测试自己的代码是否正确的，如果你只使用我们的OJ来进行判断，极有可能出现下述的情况

> 你编写的函数A后通过了测试A，但是测试A没有测试到函数A的边界情况A，你为测试B编写了函数B和C，但是测试B和测试A并不是毫无关联，因此导致触发了函数A的边界情况A，但是你因为你新编写了函数B和C，这时候你需要判断问题究竟出在函数A、B、C的哪一个地方

对于一段代码来说，你进行的测试越详尽，你对它就越有信心，越敢在后续的代码中使用它。

善用版本控制

在写代码的过程中，如果一开始没有想好该怎么完成一个模块，那么在实际编写的时候可能会磕磕碰碰，遇到很多问题，遇到问题难免就需要各种修修补补，这个修补的过程也是极容易出错的，可能你想到了解决方案A，然后实践出来发现不对，想要回溯到实现A之前，这个时候有一个版本控制工具就会极大的提高你的coding效率。

建议大家每次完成一个模块之后就进行一次commit，每次对代码做改动一定要保证上一次的修改已经被commit过了，一般来说，只要是进行了commit，之前的内容就都会被保留下来，接着你就可以进行时光回溯了。

图灵机序列化字符串格式

1. **状态集 Q**：以 `#Q` 开头。**占据 (且仅占据) 图灵机程序的一行** (即不包括回车换行)；`'#'` 为该行的第一个字符 (以下2-7同)；各状态之间以英文逗号 `','` 分隔。状态用一个或多个**非空白字符** (字母`a-z,A-Z`、数字`0-9`和下划线`_`) 表示，称为该状态的**标签**，如 `"10"`、`"a"`、`"state1"` 等。语法格式为 `#Q = {q1,q2,...,qi}`。`'='` **两边各有一个空格** `' '` (以下2-7同)。

   示例：

   ```
   #Q = {0,cp,cmp,mh,accept,accept2,accept3,accept4,halt_accept,reject,reject2,reject3,reject4,reject5,halt_reject}
   ```

2. **输入符号集 S**：以 `#S` 开头。各符号之间以英文逗号 `','` 分隔。输入符号的可取值范围为除 `' '(space)`、`','`、`';'`、`'{'`、`'}'`、`'*'`、`'_'` 外的所有 **ASCII 可显示字符** (定义参见 维基百科)。语法格式为 `#S = {s1,s2,...,sj}`。

   示例：

   ```
   #S = {0,1}
   ```

3. **纸带符号集 G**：以 `#G` 开头。各符号之间以英文逗号 `','` 分隔。纸带符号的可取值范围为除 `' '(space)`、`','`、`';'`、`'{'`、`'}'`、`'*'` 外的所有 **ASCII 可显示字符**，规定用 `'_'` 表示空格符号。语法格式为 `#G = {s1,s2,...,sk}`。

   示例：

   ```
   #G = {0,1,_,t,r,u,e,f,a,l,s}
   ```

4. **初始状态 q0**：以 `#q0` 开头。语法格式为 `#q0 = q`。

   示例：

   ```
   #q0 = 0
   ```

5. **空格符号 B**：以 `#B` 开头。语法格式为 `#B = b`。在本实验中，**空格符号规定为** `'_'`。

   示例：

   ```
   #B = _
   ```

6. **终结状态集 F**：以 `#F` 开头。各状态之间以英文逗号 `','` 分隔。语法格式为 `#F = {f1,f2,...,fn}`。

   示例：

   ```
   #F = {halt_accept}
   ```

7. **磁带数 N**：以 `#N` 开头。语法格式为 `#N = n`。

   示例：

   ```
   #N = 2
   ```

8. **转移函数 delta**：每个转移函数占用 (且仅占用) 图灵机程序的一行，行内容为一个五元组。五元组格式为："<旧状态> <旧符号组> <新符号组> <方向组> <新状态>"，元组各部分之间以一个空格分隔。以`#D`开头

   - **(新、旧) 状态** 定义见 **状态集**。
   - **(新、旧) 符号组** 均由 `n` 个**纸带符号**组成的字符串表示。
   - **方向组** 由 `n` 个以下三个符号之一组成的字符串表示：`'l'`、`'r'` 或者 `'*'`，分别表示向左移动、向右移动和不移动。

   示例：

   ```
   #D cmp 01 __ rl reject ; 当前处于状态 cmp
                          ; 两个带头下符号分别为 '0' 和 '1'
                          ; 将要写入的新符号为 '_' 和 '_'
                          ; 下一步第一个带头向右移动，第二个带头向左移动
                          ; 下一个状态为 reject
   ```

9. **注释和空行**：你只需要处理行注释，行注释以英文分号 `';'` 开头。在解析程序时应当忽略注释和空行。

   示例：

   ```
   ; State cmp: compare two strings

   0 __ __ ** accept ; empty input
   ```

> 本格式来自 图灵机程序语法