My Turing Machine

161220126 王森

Contents

1 实验内容

1.1 任务一：多带图灵机程序解析器

以符合语法描述的图灵机程序作为解析器的输入；该解析器能够正确解析读入的图灵机程序；图灵机程序解析完成后，得到与之对应的一个图灵机模拟器。

1.2 任务二：多带图灵机模拟器

图灵机程序经过解析器解析后，得到一个对应的图灵机模拟器。模拟器读入一个字符串作为图灵机的输入。要求：

1. 判断输入字符串的合法性 (也即判断所有字符是否均属于输入符号集，输入符号集的定义参见语法描述)。若字符串不合法，则不对其进行模拟运行，并按以下格式报告输入错误：

```
Input: XXXXXXXXXXX
==================== ERR ====================
The input "XXXXXXXXXXX" is illegal
==================== END ====================
```

若字符串合法，按以下格式输出提示信息，并继续运行：

```
Input: XXXXXXXXXXX
==================== RUN ====================
```

2. 模拟器对合法的输入串进行模拟运行。对于图灵机上的每一次转移，按以下格式给出图灵机的一个瞬时描述：若字符串合法，按以下格式输出提示信息，并继续运行：

```
Step : 0
Index0 : 0 1 2 3 4 5 6
Tape0 : 1 0 0 1 0 0 1
Head0 : ^
Index1 : 0
Tape1 : _
Head1 : ^
State : 0
-----------------------------------------------------
```

3. 模拟运行结束后，将第一条纸带上的内容 (首尾分别为纸带上第一个和最后一个非空格符号) 作为最后的输出，若接收输入的字符串则最后纸带上的内容为 True，反之为 False。并按以下格式输出信息：

```
Result: XXXX
==================== END ====================
```

1.3 任务三：多带图灵机程序

使用规定的图灵机程序语法，实现识别以下两个语言的图灵机对应的图灵机程序。

1. $\{0^k \mid k$ 是一个斐波那契数 $\}$
2. $\{ww \mid w \in \{0,1\}^*\}$

2 实验分析（数据结构）

2.1 图灵机纸带

图灵机拥有若干条纸带，每条纸带无限长度，每条纸带都有一个读写头指向当前正在操作的纸带单元。设图灵机拥有 N 条纸带，可使用二维 vector 模拟，每一条纸带是一个包含 char 和 index 的复合类型的 vector，因而可以双向扩展，理论上无限长度。另设置长度为 N 的数组存储每一条纸带的读写头位置（iterator 类型）。

```
struct Tape {
    char ch;
    int index;
};
vector<vector<Tape> > tapes;
vector<vector<Tape>::iterator> heads;
```

2.2 图灵机初始化

图灵机状态集 Q，字母表集合 S，纸带字母表集合 G，终止状态集合 F，采用集合存储。起始状态 q0 和空白符 B，采用 string 类型存储，纸带个数 N 采用 int 类型存储。

```
set<string> Q;
set<char> S;
set<char> G;
string q0;
string B;
set<string> F;
int N;
```

图灵机的转移函数采用符合数据结构存储，需要涉及当前图灵及所属状态，目标转移状态，当前所有纸带读写头位置字符情况，将当前所有纸带读写头位置修改的目标情况，修改完读写头内容后的读写头移动情况。

```
struct Node {
    string currentState;
    string nextState;
    string currentTape;
    string nextTape;
    string direction;
};
vector<Node> Transition;
string currentState;
```

3 图灵机模拟程序设计思路

3.1 图灵机解析器

读取图灵机描述文本，跳过空行与注释文本，逐一将 Q，S，G，q0，B，F，N 和转移函数存储到对应的数据结构中去，具体数据结构在上面已经叙述过。

3.2 图灵机初始化

读取 input.txt 文件中的输入字符串，复制到第一条纸带上，并将纸带读写头指向字符串头部。其余纸带（如果有的话）上只有一个空白字符，图灵机的当前状态设为起始状态。

```
void init(string input) {
    for(int i = 0; i < N; i++) {
        tapes[i].clear();
        heads[i] = tapes[i].begin();
    }
    step = 0;
    if(input.length() == 0) {
        Tape tape;
        tape.index = 0;
        tape.ch = B[0];
        tapes[0].push_back(tape);
    }
    else {
        for(int i = 0; i < input.length(); i++) {
            Tape tape;
            tape.index = i;
            tape.ch = input[i];
            tapes[0].push_back(tape);
        }
    }
    heads[0] = tapes[0].begin();
    for(int i = 1; i < N; i++) {
        Tape tape;
        tape.index = 0;
        tape.ch = B[0];
        tapes[i].push_back(tape);
        heads[i] = tapes[i].begin();
    }
    currentState = q0;
}
```

3.3 选择转移函数

对于某一个状态的图灵机来说，遍历转移函数的集合，可能有多个相匹配的转移函数（因为通配符的存在），也有可能没有相匹配的转移函数。对于前者，将所有的相匹配的转移函数都存储在 candidates 中，遍历完毕，对 candidates 中所有相匹配的转移函数进行比较，选取通配符最少的转移函数作为最终选定的转移函数。对于后者，遍历所有转移函数之后，candidates 中没有候选转移函数，则此时图灵机 halt，模拟结束。

```
vector<Node> candidates;
for(it = Transition.begin(); it != Transition.end(); it++) {
    if((*it).currentState == currentState) {
        for(j = 0; j < N; j++) {
            if((*it).currentTape[j] == '*' || tapes[j].size() != 0 && (*it).currentTape[j] ==
                (*(heads[j])).ch) {
                ;
            } else {
                break;
            }
        }
        if(j == N) {
            candidates.push_back(*it);
        }
    }
}
if(candidates.size() == 0) {
    break; // halt
} else {
    if(candidates.size() == 1) {
        takeAction(candidates[0]);
    } else {
        int minNum = -1;
        int index = 0;
        int counter; // 通配符个数
        for(int k = 0; k < candidates.size(); k++) {
            counter = 0;
            for(int m = 0; m < N; m++) {
                if(candidates[k].currentTape[m] == '*') {
                    counter++;
                }
                if(candidates[k].nextTape[m] == '*') {
                    counter++;
                }
                if(candidates[k].direction[m] == '*') {
                    counter++;
                }
            }
            if(minNum == -1 || minNum > counter) {
```

```
                minNum = counter;
                index = k;
            }
        }
        takeAction(candidates[index]);
    }
}
```

3.4 图灵机状态变化

寻找到最佳匹配的转移函数之后，根据转移函数，对图灵机的运行情况进行修改，包括当前状态，纸带情况，读写头移动情况。但是会遇到边界情况，如超出当前纸带的长度，或者已经到达了当前纸带的最左端，则需要对纸带进行扩张，初始化为空字符。

```
void takeAction(Node node) {
    printCurrent();
    step++;
    currentState = node.nextState;
    for(int i = 0; i < N; i++) {
        if(node.nextTape[i] != '*') { // 纸带内容不是保持不变
            heads[i]->ch = node.nextTape[i];
        }
        if(node.direction[i] != '*') { // 读写头不是不变
            if(node.direction[i] == 'l') {// 左移
                if(heads[i] == tapes[i].begin()) { // 边际情况
                    Tape tape;
                    tape.ch = B[0];
                    tape.index = tapes[i].begin()->index-1;
                    tapes[i].insert(tapes[i].begin(), tape);
                    heads[i] = tapes[i].begin();
                } else
                    heads[i]--;
            } else { // 右移
                // 边际情况
                if(heads[i]-tapes[i].begin()+1 == tapes[i].size()) {
                    Tape tape;
                    tape.ch = B[0];
                    tape.index = heads[i]->index+1;
                    tapes[i].push_back(tape);
                    heads[i] = --tapes[i].end();
                } else
                    heads[i]++;
            }
        }
    }
```

```
}
```

3.5 图灵机 halt 并打印结果

当找不到相匹配的转移函数时，将第一条纸带上的内容打印出来，并清空其他纸带上的内容。

4 图灵机设计思路

4.1 斐波那契数

使用三条纸带，初始状态第二条和第三条纸带上只有一个 0，在这两条纸带上模拟斐波那契数列的迭代过程。每次先分别比较第一条纸带上的 0 的个数，若第一条纸带上的 0 都是略多的一方，则继续进行斐波那契数列的迭代。把第二条纸带上的 0 全都拼接到第三条纸带后面，再把第三条纸带上的内容拼接到第二条纸带的后面，这里进行了两步斐波那契数列的迭代，a(n+1) = a(n-1)+a(n) 以及 a(n+2) = a(n+1)+a(n)，然后继续比较 0 的个数。直到后两条纸带上的 0 的个数与第一条纸带上的 0 个数相等，则 accept 打印 True，或者第一条纸带上的 0 个数小于另两个或者介于两者之间则 reject 打印 False。

4.2 WW 字符串

使用三条纸带，对于输入的字符串，若为空串，则接收。若不为空串，则从第二个位置开始逐一作为第二个 W 的起始位置，并比较拆分出来的两个字符串是否相等。若相等则接收，若不相等则进行下一次划分。直到划分之后，前一段字符串的长度比后一段的字符串长度大，则拒绝该字符串。

5 实验完成度

整个实验按照实验要求全部完成，并构造了样例对任务三的两个图灵机程序进行了测试。

6 遇到的问题与解决方案

6.1 遇到的问题

Windows 下书写的图灵机描述文本，在 linux 下使用程序解析失败。

6.2 解决方案

Windows 下换行为 \n\r，linux 下换行为 \n，因此将 Windows 下的文本文件放进 linux 下每行的行尾会多出 \r 的符号，因此需要采用 dos2unix 工具对文本进行转码。解决了问题。

7 编译运行说明

使用 make 对程序进行编译，生成 Turing 可执行文件，然后输入./Turing case1 和./Turing case2 即可运行。使用 make clean 清除可执行文件。

```
make clean
make
```

```
./Turing case1
./Turing case2
```

8 演示

8.1 斐波那契数

测试样例如下：

```
0
000
000000000
0000000000000
```

console 部分输出如下：

```
Input: 0
==================== RUN ====================
Step  : 0
Index0 : 0
Tape0 : 0
Head0 : ^
Index1 : 0
Tape1 : _
Head1 : ^
Index2 : 0
Tape2 : _
Head2 : ^
State : 0
-----------------------------------------------------
Step  : 1
Index0 : 0
Tape0 : 0
Head0 : ^
Index1 : 0
Tape1 : 0
Head1 : ^
Index2 : 0
Tape2 : _
Head2 : ^
State : p1
-----------------------------------------------------
```

图灵机最终输出如下：

```
True
```

```
True
False
True
```

8.2 WW 字符串

测试样例如下：

```
010010
000
001001
101010
```

console 部分输出如下：

```
Input: 010010
==================== RUN ====================
Step  : 0
Index0 : 0 1  2  3  4  5
Tape0 :  0  1  0  0  1  0
Head0 :  ^
Index1 : 0
Tape1 :  _
Head1 :  ^
Index2 : 0
Tape2 :  _
Head2 :  ^
State :  0
-----------------------------------------------------
Step  : 1
Index0 : 0 1  2  3  4  5
Tape0 :  0  1  0  0  1  0
Head0 :     ^
Index1 : 0
Tape1 :  _
Head1 :  ^
Index2 : 1
Tape2 :  _
Head2 :  ^
State :  pre
-----------------------------------------------------
```

图灵机最终输出如下：

```
True
False
True
```

```
False
```

9 总结感想

9.1

图灵机模型引入了读写与算法与程序语言的概念，其计算能力十分强大。

9.2

通配符可以极大地提高图灵机程序的设计自由度。

9.3

计算理论初步是一门优秀的计算机科学课程，让我学到了很多很重要的知识。