Haskell大作业报告

软件52班 张迁瑞 2015013226

算法设计过程

总体思路

经过相关资料的调研与阅读，我发现猜数字（Bulls and Cows）问题已经是一个相对古老的问题了。这个问题的解决方法很多，但都没有脱离最基本的refine的框架。即：

1. 将所有可能值作为candidate集合。
2. 从candidate中随机选择一个数字进行猜测。
3. 得到返回结果(p,q)后，遍历candidate，将和之前选择的数字距离不是candidate的数字从candidate中删除。
4. 在新的candidate集合中再选择一个数字进行猜测。（不同算法的区别在于如何选择这个数字）
5. 重复2-4直到找到正确结果。

不同算法的区别在于第四步，如何根据已有信息选择下一步猜测的数字。经测试，如果在这一步不进行任何选择，而是随机选取，那么5040个数字的总猜测次数大概是27200左右。

在我的实验中，我使用的是Lamouth的启发式策略。

Lamouth算法

Lamouth算法的主要思想是，每次选择猜测时使得不同回答等价响应类的数量最大。

等价响应类(response class)的定义为，对于同一个猜测，得到的(b,c)相同的那些数字构成一个等价响应类。

在上节算法要选择第四步要使用的数字时，计算如下函数：

```
∑|t_r| * log|t_r| - 2 * log(2) * |t_BBBB|
```

其中|t_r|为等价响应类r中元素的数量，对于candidate集合中的所有元素，找到使得这个函数值最小的那个数字，这个数字即是下一次应该猜测的数字。

实验结果

算法实现过程

算法的实现过程比较平凡，我先使用C++对Lamouth算法进行了测试，之后将其用haskell实现。实现时的主要难点在于等价类的生成，由于在haskell中无法像C++那样方便地实现循环遍历，所以导致haskell的运行速度要远慢于C++。

实验结果

使用Lamouth的启发式算法后，总的猜测次数可以降低为26569次，每一次猜出数字的时间大概在10秒左右。根据参数的不同（比如第一次猜测时不使用0123），结果可能会略有差别。

关于剪枝

在实验中可以使用剪枝的方法来减少运算量，具体思路为，在计算上文中函数时，一旦该函数的值超过了某个阈值，就直接终止运算，不再继续计算其他的等价类。但由于在C++的实现中此种剪枝没有起到多大效果，故没有在haskell算法中实现这一剪枝方法。

参考文献

- Strategies for playing MOO, or "Bulls and Cows" John Francis http://www.jfwaf.com/Bulls%20and%20Cows.pdf
- R.W . Irving, Towards an Optimal Mastermind Strategy, J.Recreational Mathematics