

人工智能第一次大作业设计报告

自 45 柳荫 2014011858

注：评分模板选择模板 A。

可执行程序是 H1. jar

我做的是第 3 题，数字游戏题。

用了 java 开发环境，编译环境要求用户机器上安装了 JDK，并且配置了所需要的编译 java 的环境变量。

共设计了 5 个类：PalyNumbers, LargestNum, NearestNum, NearestProduct, SeveralMultipleBiggestNum。其中 PlayNumbers 是搭建整体框架包括 UI 设计等的类，接着 LargestNum 是求解给出的第一题的类，NearestNum 是求解给出的第二题的类，SeveralMultipleBiggestNum 是求解自己出的难度分别为” Simple”，“Normal”，“Hard” 三道题目的类，NearestProduct 是求解自己出的难度为” Extreme” 的题目的类。

UI 设计：

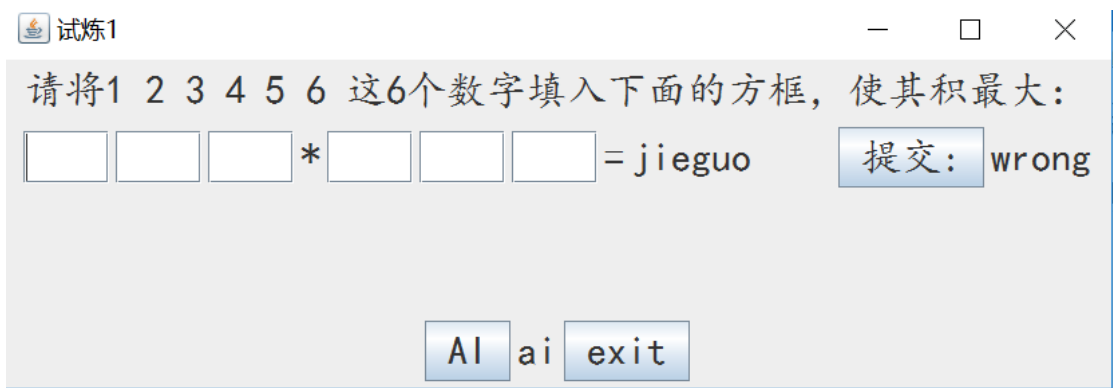
全部在 **PlayNumbers** 里面实现。开启游戏，有一个如下的启动界面：



游戏名为 NumberPlate, 配上一幅背景图片, src 要和 a1. jpg 放在一个目录下编译后才能显示图片, 可执行程序 H1. jar 也要和 a1. jpg 放在一个目录下运行时才能显示图片。

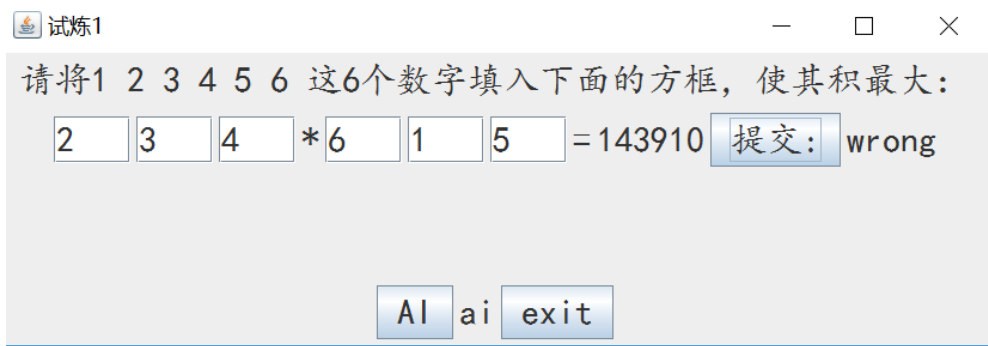
加上少许游戏选项——“试炼 1”对应原题中第一题, “试炼 2”对应原题中第二题, “跟多难度”对应自己出的更难的题目, “我要退出”会关闭所有游戏衍生的界面并将整个游戏退出。

点击”试炼 1”,弹出一个新界面：

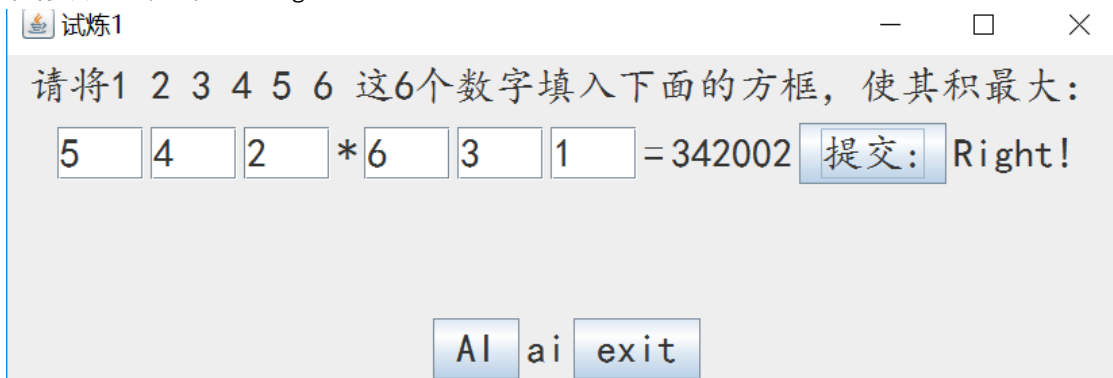


界面由一行规则，一行与用户交互的数字填写并检验，一行计算机求解以及退出的交互。
游戏默认玩家的填写结果状态为“wrong”。

游戏规则讲得很清楚，所以只管填 6 个数字，比如你填了 2 3 4 6 1 5，然后点击”提交”按钮，会出现：



算出你的结果，显示出来，但由于错，所以结果状态还是“wrong”。
若你填对了，则显示“Right!”如下图：



另外，只要你填得不符合要求，无论是每空的字符填多了、少了、不是 123456 六个数字、或是并没有每个数字仅用一次，那么提交后都会提示你输入错误如下面的一些截图：

试炼1

请将1 2 3 4 5 6 这6个数字填入下面的方框，使其积最大：

* = 342002

输入不符合要求！请重新输入！

试炼1

请将1 2 3 4 5 6 这6个数字填入下面的方框，使其积最大：

* = 342002

输入不符合要求！请重新输入！

试炼1

请将1 2 3 4 5 6 这6个数字填入下面的方框，使其积最大：

* = 342002

输入不符合要求！请重新输入！

下面是最下面的 2 个按钮：
点击"AI"，会显示应填的数，以及应有的结果：

试炼1

请将1 2 3 4 5 6 这6个数字填入下面的方框，使其积最大：

* = 342002

输入不符合要求！请重新输入！

应填的数： 631 与 542 其结果为： 342002 .

而点击"exit"会关闭"试炼 1"窗口。

点击启动界面的”试炼 2”,同样会弹出窗口：

试炼2

— □ ×

请将1 2 3 4 5 6 这六个数填入以下6个方框中，使其离400000最近：

jieguo

提交：

与400000的距离：差值wrong

AI

ai

exit

与试炼 1 类似，只有当填的 6 个数是 123456 的某一个排列时输入才合理，提交后才有正确和错误之分，以及与 400000 的距离。

试炼2

— □ ×

请将1 2 3 4 5 6 这六个数填入以下6个方框中，使其离400000最近：

1

3

5

4

6

2

135462

提交：

与400000的距离：264538wrong

AI

ai

exit

试炼2

— □ ×

请将1 2 3 4 5 6 这六个数填入以下6个方框中，使其离400000最近：

4

1

2

3

5

6

412356

提交：

与400000的距离：12356Right!

AI

ai

exit

“AI”的按钮点击也是显示一个正确答案，

试炼2

— □ ×

请将1 2 3 4 5 6 这六个数填入以下6个方框中，使其离400000最近：

4

1

2

3

5

6

412356

提交：

与400000的距离：12356Right!

AI

应填的数：412356，与400000的距离：12356 .

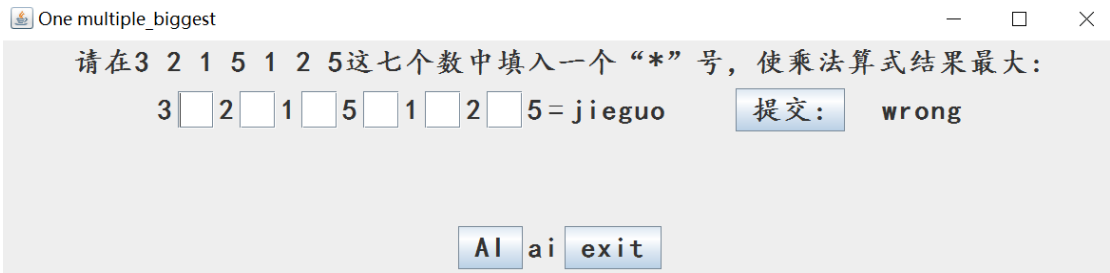
exit

点”exit“会关闭窗口。

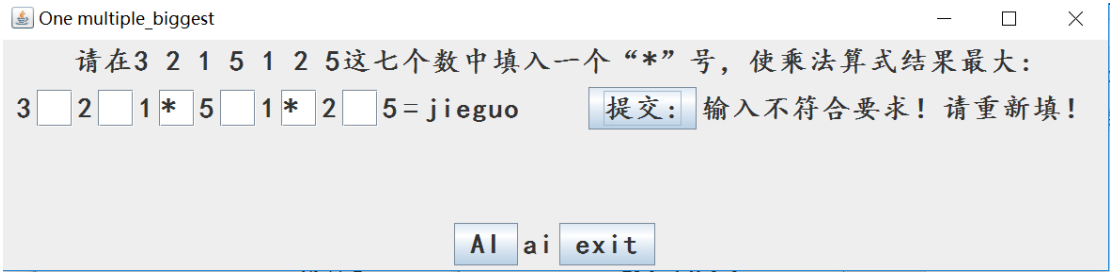
在“更多难度”中，有一个难度系数窗口，可以选择不同的题目难度

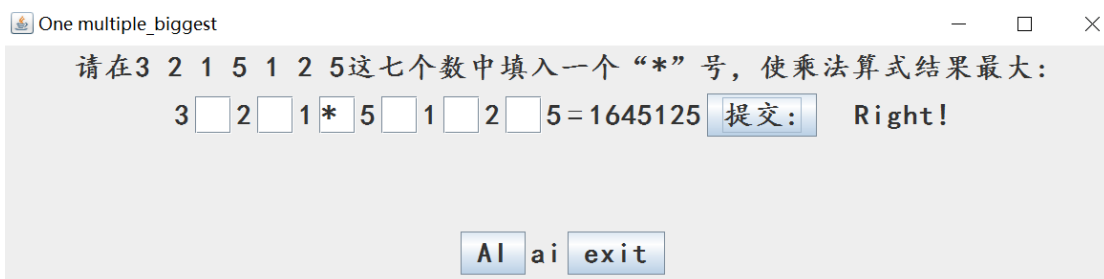
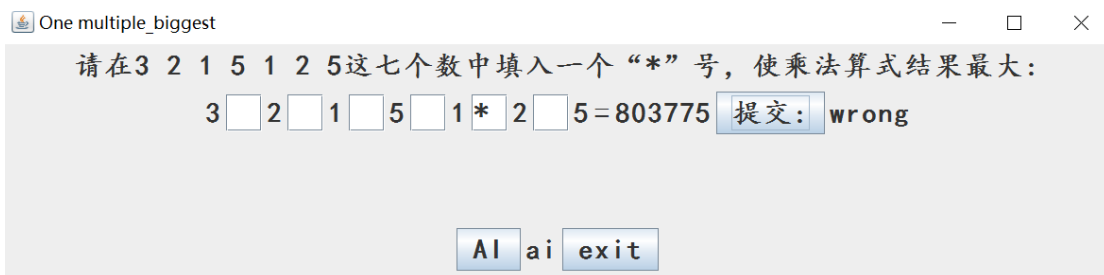


点击“simple“ 会有简单的一个题目：



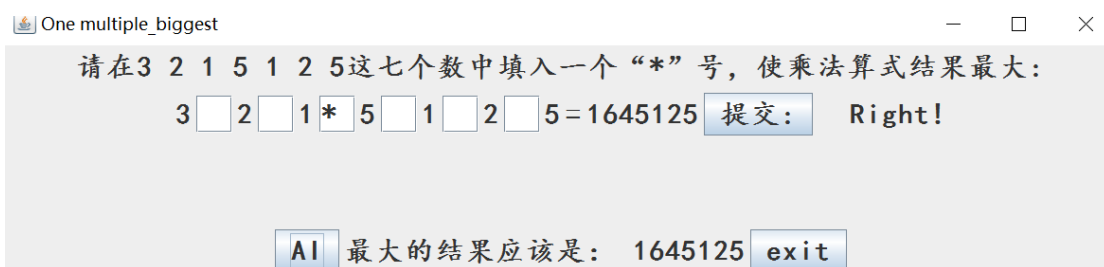
One multiple biggest，只能在 6 个框中选一个填“*”号，否则会报错。





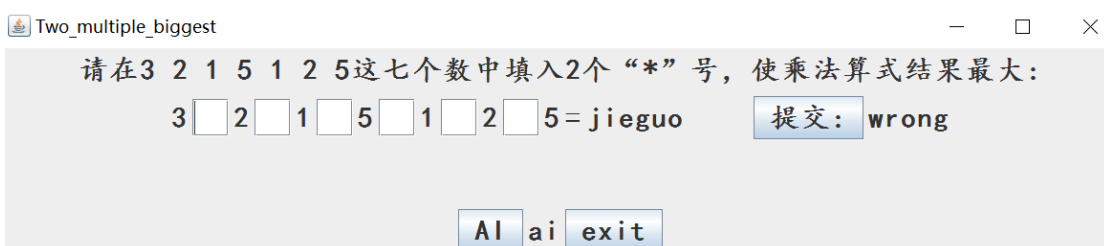
上面分别显示了输入不合法、答案错误、答案正确的提交后的界面。

点击“AI”会自动解题：

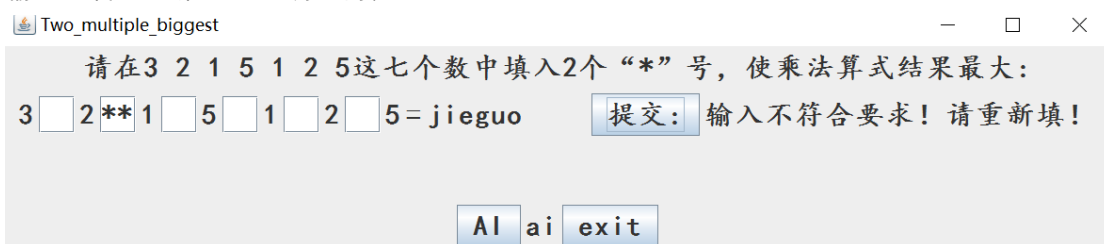


点击 exit 会退出。

在“难度系数”界面点击“Normal”也会有一个类似的题目
Two_multiple_biggest，即填入 2 个“*”号使结果最大：



输入不合法、错误、正确同前，显示如下：



Two_multiple_biggest

请在3 2 1 5 1 2 5这七个数中填入2个“*”号，使乘法算式结果最大：

3 2 * 1 5 * 1 2 5 = 60000 提交: wrong

AI ai exit

Two_multiple_biggest

请在3 2 1 5 1 2 5这七个数中填入2个“*”号，使乘法算式结果最大：

3 2 1 * 5 1 2 * 5 = 821760 提交: Right!

AI ai exit

点击“AI”按钮会调用算法解题，显示最大结果：

Two_multiple_biggest

请在3 2 1 5 1 2 5这七个数中填入2个“*”号，使乘法算式结果最大：

3 2 1 * 5 1 2 * 5 = 821760 提交: Right!

AI 最大的结果应该是: 821760 exit

点击“exit”会关闭界面。

在“难度系数”界面点击“Hard”也会有一个类似的题目
Three_multiple_biggest，即填入 3 个“*”号使结果最大：

Three_multiple_biggest

请在3 2 1 5 1 2 5这七个数中填入3个“*”号，使乘法算式结果最大：

3 2 1 5 1 2 5 = jieguo 提交: wrong

AI ai exit

输入不合法、错误、正确同前，显示如下：

Three_multiple_biggest

请在3 2 1 5 1 2 5这七个数中填入3个“*”号，使乘法算式结果最大：

3 2 1 ** 5 1 * 2 5 = jieguo 提交: 输入不符合要求！请重新填！

AI ai exit

Three_multiple_biggest

请在3 2 1 5 1 2 5这七个数中填入3个“*”号，使乘法算式结果最大：

3 * 2 * 1 * 5 * 1 * 2 * 5 = 80325 提交: wrong

AI ai exit

Three_multiple_biggest

请在3 2 1 5 1 2 5这七个数中填入3个“*”号，使乘法算式结果最大：

3 * 2 * 1 * 5 * 1 * 2 * 5 = 163710 提交: Right!

AI ai exit

点击“AI”按钮会调用算法解题，显示最大结果：

Three_multiple_biggest

请在3 2 1 5 1 2 5这七个数中填入3个“*”号，使乘法算式结果最大：

3 * 2 * 1 * 5 * 1 * 2 * 5 = 163710 提交: Right!

AI 最大的结果应该是: 163710 exit

点击“exit”会关闭界面。

在难度系数界面点击“Extreme！”则会弹出界面“近水楼台”：

近水楼台

请将1 2 3 4 5 6六个数分别填入下面方框中，使乘法算式结果最接近给定值：314054

* * * * * = jieguo 提交: wrong

AI ai exit

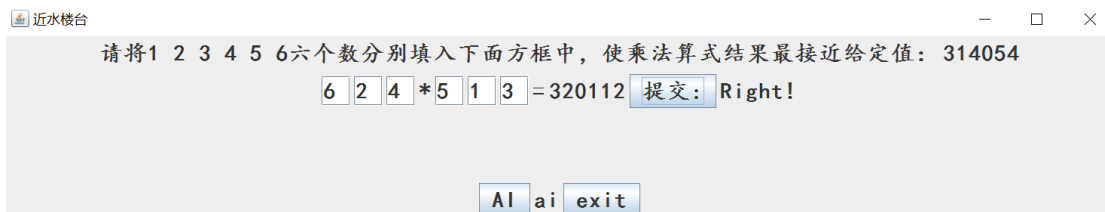
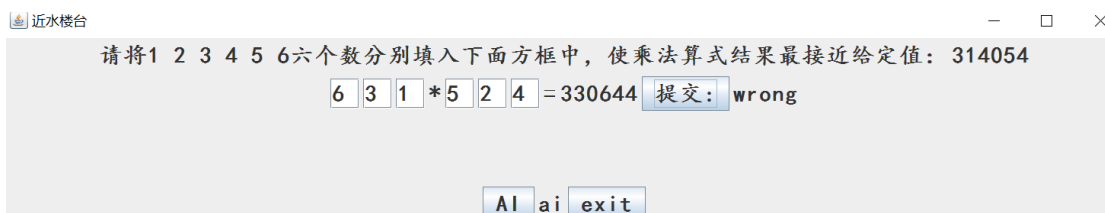
对应的是如上所示的题目，这其中给定值是生成的 0~350000 内的随机数。同样如果输入不合法的话，也会报错，只有输入合法后提交才能显示对错：

近水楼台

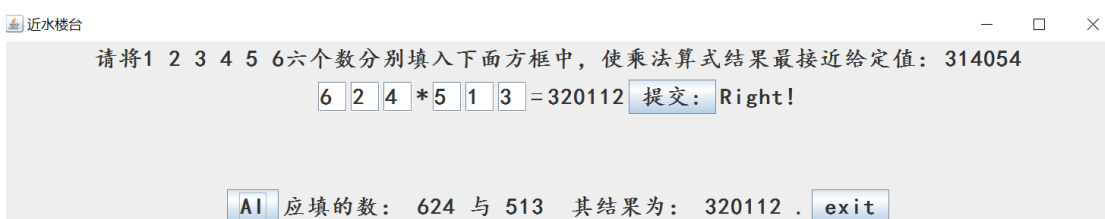
请将1 2 3 4 5 6六个数分别填入下面方框中，使乘法算式结果最接近给定值：314054

6 3 1 * 5 2 2 = 330644 提交: 输入不符合要求！请重新输入！

AI ai exit



同样，点击"AI"也会显示计算机做出来的结果，



点击"exit"退出界面。

算法设计：

1. 求取最大的数，即试炼 1，在 LargestNum 类中我采用的是直接搜索，先尝试一下最简单粗暴的方法。
2. 求取最接近 400000 的数，即试炼 2，我在 NearestNum 中用了深度优先搜索的思想，用了递归的算法：

```

36
37
38     private void fill(int index){
39         //         if(index == 7) System.out.println(constructNum());
40         if(index == 7) {
41             count[h] = constructNum();
42             dif[h] = Math.abs(count[h] - 400000);
43             ++h;
44         }
45         for(int i = 1; i < 7; ++ i) {
46             if (find[i]) continue;
47             a[index] = i;
48             find[i] = true;
49             fill(index+1);
50             find[i] = false;
51         }
52     }

```

结果可以从小到大得到 6 个数的全部组合六位数，方便接下来继续求解。

以上是最简单的 2 个题目的算法设计。

3. 对于更多难度里的前三题（一类题），我采用了动态规划的算法思想同样用了递归的方法，在 SeveralMultipleBiggestNum 中：

```

25
26     public int P(int l, int r, int k){
27         if(k == 0) return d[l][r];
28         int x,ans = 0;
29         for(int q = 1; q <= r - k; q++){
30             x = d[l][q] * P(q + 1, r, k - 1);
31             if(x > ans) ans = x;
32         }
33         return ans;
34     }
35 }
36

```

定义了一个 P 函数，3 个参数 l r k 的意思是，一个若干位数，从第 l 位到第 r 位，加入 k 个乘号的乘积值，P 函数返回其最大值，函数体中 d[l][q] 表示从 l 到 q 的这个数。对于自己题目中采用的 7 位数，返回加入 k 个乘号的最大值，就是 l = 0, r = 6, 对题目“simple”，k = 1, 对题目“normal”，k = 2, 对题目“hard”，k = 3。

假设我们考虑 hard 即加入 3 个乘号的求最大值，将 S 分为 2 段，第一段为第一个乘号左边的，是一个数；第二段为第一个乘号右边的，是包含 2 个乘号的部分，若第一个乘号固定，则要求出第二段的最大值，是一个子问题，因此

$$P(l, r, k) = \max_q \{ d(l, q) * P(q+1, r, k-1) \}.$$

从 q+1 到 r 之间包含的数字个数大于 k-1（乘号个数） 因此

$$q < r - k + 1$$

用这种思路，对本题，就可以写出：

$$P(0, 6, 3) = \max \{ 3 * P(1,6,2), 32 * P(2,6,2), 321 * P(3,6,2), 3215 * P(4,6,2) \}.$$

其中递归的边界是

$$P(u, r, 0) = d(u, r) = S_u S_{u+1} \cdots S_r.$$

对每个子问题用同样的方法展开，得到最大的值再回带比较，动态规划 + 递归 可以最终得到最大值。

这种解法对于任何多位数及若干个乘号求最大值均非常实用有效，并不局限于本题中的那个 3215125。

4. 对于 Extreme 的题目，给定的目标数是生成的随机数（0~340000），摒弃了直接暴力搜索比较的方法，在 NearestProduct 类中采取了多重剪枝的手段。

```

28      public int[] search(int givenNum){...}
78
79      public int[] search2(int m, int n, int givenNum){...}
155
156      public int[] search2x(int m, int n, int givenNum){...}
226
227      public int[] search4(int m, int n, int p, int q, int givenNum){...}
265
266      public int[] search4x(int m, int n, int p, int q, int givenNum){...}
304

```

定义了 5 个搜索子函数。

其中 **search4** 是给定 2 个 3 位数各自的前两位，用 6 个数中剩下的 2 个数分别填补后一位，会有 2 个乘积结果，1 大 1 小，若给定目标数在此范围内，则返回 2 个乘积离得近的其 2 个乘数，若不在此范围内，大则返回[1,1],小则返回[0,0]。在上层函数调用时，若遇到超出范围的，则会调用 **search4x**，这是不管在不在范围内，都返回离得近的乘积的 2 个乘数。

search2 类似是给定 2 个 3 位数的各自的首位，用 6 个数中剩下的 4 个数分别填入后面，会有若干个乘积结果，容易证明，若剩下的 4 个数是 $c_3 > c_2 > c_1 > c_0$, 2 个乘数的首位分别是 a,b, ($a > b$)，则 其最大结果是 $[a c_2 c_0] * [b c_3 c_1]$ ，最小结果是 $[a c_1 c_3] * [b c_0 c_2]$ ，（只要考虑到**十位尽可能取大数**，然后**2 个数和一定的情况下越靠近其乘积越大** 这 2 条数学规则就行）。同样的，若给定目标数在此范围内，则返回乘积离得近的其 2 个乘数，若不在此范围内，大则返回[1,1],小则返回[0,0]。在上层函数调用时，若遇到超出范围的，则会调用 **search2x**，这是不管在不在范围内，都返回离得近的乘积的 2 个乘数。

最后是上层的 *search* 函数，只给定一个目标数，然后调用 *search2*，一般来说 0 ~ 340000 中大多数数都在某 1 个或某几个 *search2* 的范围内，即返回不是 [1,1] 或 [0,0]，

这下就剪了不少枝，很少会调用到 *search2x* 的（只有给定数极小或极大时可能在某些范围的“代沟”中）。

然后这几个选中的 *search2* 每一个又调用下面的 *search4*，若存在某几个 *search4* 范围符合，（其他的返回 [1,1] 或 [0,0] 的 *search4* 就被“剪掉了”，）则返回最小的，若所有 *search4* 都返回 [1,1] 或 [0,0]，则再调用 *search4x*。

这样下来层层剪枝，效率会提高很多，不在范围内的果断摒弃，只有迫不得已（都不在范围内），才会调用“*x*”函数。

整个算法设计思路就是这样。