计算物理作业 1

杨远青 22300190015

2024 年 9 月 27 日

1 题目 1：五次幂丢番图方程

1.1 题目描述

Find all integer solutions to the **Diophantine equation** $a^5 + b^5 + c^5 + d^5 = e^5$ within the range $[0, 200]$.

1.2 程序描述

1967 年，Lander 和 Parkin 使用 CDC 6600 计算机首次找到方程

$$a^5 + b^5 + c^5 + d^5 = e^5$$

的一个解，即

$$27^5 + 84^5 + 110^5 + 133^5 = 144^5$$

这个解也推翻了欧拉的幂和猜想（即至少需要 k 个 k 次幂才能表示另一 k 次幂的和，且 $k \geq 2$）。相关参考资料可参见 Diophantine Equation—5th Powers。2004 年，J. Frye 使用分布式并行计算找到了第二个解：

$$55^5 + 3183^5 + 28969^5 + 85282^5 = 85359^5$$

题目要求我们找到所有满足 $0 \leq a \leq b \leq c \leq d < e \leq 200$ 的整数解

为了解决这个问题，我们首先采用暴力搜索的方法，遍历所有可能的 a, b, c, d, e 组合，并逐一检查是否满足方程。在 `brute_force.f90` 中实现了这一方法，其伪代码见算法 1。该程序的运行时间达到了 8.953 秒，速度较为缓慢。

在网络上寻找更优解法时，我偶然发现了一个专门讨论整数幂方程的论坛：Euler Free。该论坛提到可以通过数论优化的方法。注意到方程中的 x^5 是齐次的，首先注意到到费马小定理：

$$a^{p-1} \equiv 1 \pmod{p}, \quad \gcd(a, p) = 1.$$

基于此，在搜索时可以将步长扩大到 5。然而，进一步查阅 Lander 和 Parkin 的论文后，未能找到有效的进一步启发。在 Rosetta Code 网站上，我发现了 C 语言中使用的Mod30 Trick，这让我意识到可以进一步扩展费马小定理的应用范围。即我们可以证明：

$$a^5 \equiv a \pmod{30}$$

这意味着在搜索时可以将步长扩展到 30。(昨天使用这个问题测试过了 ChatGPT o1 pre，它枚举证明了所有情况。) 具体而言，对于 $n = 2, 3, 5$，我们可以分别讨论奇偶性和模数的同余关系：- 对于 $n = 2$，由于奇偶性，显然有 $a^5 \equiv a \pmod 2$。

- 对于 $n = 3$：- 若 $\gcd(a, 3) = 1$，则 $a^2 \equiv 1 \pmod 3$，因此

$$a^5 = a \cdot (a^2)^2 \equiv a \cdot 1^2 \equiv a \pmod 3$$

- 若 $3 \mid a$，则 $a \equiv 0 \pmod 3$，因此 $a^5 \equiv 0 \pmod 3$。- 对于 $n = 5$：- 若 $\gcd(a, 5) = 1$，则 $a^4 \equiv 1 \pmod 5$，因此

$$a^5 = a \cdot a^4 \equiv a \cdot 1 \equiv a \pmod 5$$

- 若 $5 \mid a$，则 $a \equiv 0 \pmod 5$，因此 $a^5 \equiv 0 \pmod 5$。

根据剩余定理，我们立马可以得出：$a^5 \equiv a \pmod{30}$

利用这个技巧，我们得到了算法 2，经测试，其 Fortran 实现的运行时间缩短至 0.797 秒。此外，通过反向查找，即从 e 开始，结合 Mod30 Trick，我们进一步优化了算法，得到了算法 3，运行时间进一步缩短至 0.438 秒。

尽管运行时间有所改善，但 mod30 技巧只是常数级别的优化，不能显著加快找到第二个非平凡解的速度。为了进一步降低算法的复杂度，我们尝试将暴力搜索的时间复杂度从 $\mathcal{O}(N^5)$ 降低至 $\mathcal{O}(N^3)$ 。最自然的想法是使用哈希表和分治法，即借助 Hash Table 存储单个元素的幂和双元素幂和的键对，将线性搜索的 $\mathcal{O}(N)$ 成本转嫁到建表操作上，再分治匹配。

遗憾的是，Fortran 对哈希表的支持较为有限，通常需要依赖外部库，而我对其实现原理尚不完全理解。在 C++ 中借助 llu(unsigned long long int) 构造大质数，似乎可以减少碰撞并建立一个简易的 Hash Table, 并借助拓展的牛顿法快速求五次根, 据此对 a, b, c, d 的搜索剪枝, 可惜这两个技巧我并没成功在 Fortran 中复现。

不过我决定使用 Python 进行关于哈希表加速的对比研究，因为其内置了优化后的字典类型，并通过 gpt 建议的多线程进一步优化，最终将运行时间从 3.929s 加速到了 0.056s。

`./Codes/Problem 1` 中有算法 1-3 的 Fortran 实现，其目录内使用 `gfortran brute_force brute_force.f90` `gfortran mod30 mod30_trick.f90` `gfortran mod30v2 mod30_trick_reverse.f90` 等命令可以编译。编译选项 $-o$ 后面的程序名可自选，再执行即可。如果是在 Windows 下，可以直接使用`brute_force.exe`、`mod30.exe`、`mod30v2.exe`运行。

以及算法 4、5 的 Python 实现，其目录内使用`python -u brute_force.py`、`python -u hash_quick.py`编译后运行。

1.3 伪代码

暴力破解算法伪代码如下, 实际 Fortran 实现时没有存储 *solutions*，而是逐一打印。

Algorithm 1: Brute-force solution to the Diophantine equation

Input: N: Integer (the upper bound, $N = 200$)

Output: *solutions*: List of tuples (a, b, c, d, e) where $0 \le a \le b \le c \le d < e \le N$

```
for a ← 0 to N do
    for b ← a to N do
        for c ← b to N do
            for d ← c to N do
                for e ← d + 1 to N do
                    if a^5 + b^5 + c^5 + d^5 = e^5 then
                        solutions.append((a, b, c, d, e)) ;          // Store the solution tuple
                    end
                end
            end
        end
    end
end
return solutions
```

使用 Mod30 Trick 的正向算法 ($a \to e$) 与逆向算法 ($e \to a$) 的伪代码如 2、3 所示，Python 中实现的是构建单幂表和双幂表的算法，伪代码如 4、5 所示。

Algorithm 2: Mod30 trick for solving the Diophantine equation

Input: N: Integer (the upper bound, $N = 200$)
Output: *solutions*: List of tuples (a, b, c, d, e) where $1 \le a \le b \le c \le d < e \le N$

```
for a ← 1 to N do
    for b ← a to N do
        for c ← b to N do
            for d ← c to N do
                r_left ← mod(a + b + c + d, 30) ;
                 // Compute remainder for e
                e_start ←
                 d + ((r_left − d) mod 30) ;
                 // Ensure e ≥ d and
                 a + b + c + d ≡ e (mod 30)
                for e ← e_start to N do
                    if (e − e_start) mod 30 = 0
                     then
                        // Increase the step
                           size to 30
                        if a^5 + b^5 + c^5 + d^5 = e^5
                         then
                            solutions.append((a, b, c, d, e))
                        end
                    end
                end
            end
        end
    end
end
return solutions
```

Algorithm 3: Reverse Mod30 trick for solving the Diophantine equation

Input: N: Integer (the upper bound, $N = 200$)
Output: *solutions*: List of tuples (a, b, c, d, e) where $1 \le a \le b \le c \le d < e \le N$

```
for e ← N to 1 do
    for d ← e to 1 do
        for c ← d to 1 do
            for b ← c to 1 do
                a_min ← (e − d − c − b) mod 30 ;
                 // Compute minimal a using
                 modular arithmetic
                if a_min ≤ 0 then
                    a_min ← a_min + 30 ;
                     // Adjust a_min to fit
                     within the modulus
                end
                for a ← a_min to b do
                    if (a − a_min) mod 30 = 0
                     then
                        if a^5 + b^5 + c^5 + d^5 = e^5
                         then
                            solutions.append((a, b, c, d, e))
                        end
                    end
                end
            end
        end
    end
end
return solutions
```

Algorithm 4: Brute-force solution using a single hash

Input: *limit*: Upper bound

Output: *solution*: Tuple (a, b, c, d, e) or `None`

```
pow_5 ← [n^5 for n in [0, limit]] ;                          // List of n^5 values
pow5_to_n ← {n^5 : n for n in [0, limit]} ;                  // Dictionary mapping n^5 to n
for a ← 1 to limit do
    for b ← a to limit do
        for c ← b to limit do
            for d ← c to limit do
                pow_5_sum ← pow_5[a] + pow_5[b] + pow_5[c] + pow_5[d] if pow_5_sum in pow5_to_n
                 then
                    e ← pow5_to_n[pow_5_sum] return (a, b, c, d, e)
                end
            end
        end
    end
end
return None
```

Algorithm 5: Parallel brute-force solution using double hash

Input: *limit*: Upper bound

Output: *solution*: Tuple (a, b, c, d, e) or `None`

```
power_5 ← {i^5 : i in [1, limit]} ;                          // Dictionary of fifth powers
sum2 ← {} ;                                  // Dictionary to store sums of two fifth powers
for i ← 1 to limit                                           // Parallelized computation do
    a5 ← i^5 for j ← i to limit do
        sum2[a5 + j^5] ← (i, j)
    end
end
sk ← sorted keys of sum2 foreach p in sorted(power_5.keys()) do
    foreach s in sk do
        if p ≤ s then
            break ;                  // Exit the loop early if no further solutions are possible
        end
        if p − s in sum2 then
            return (power_5[p], sum2[s], sum2[p − s])
        end
    end
end
return None
```

1.4 结果示例

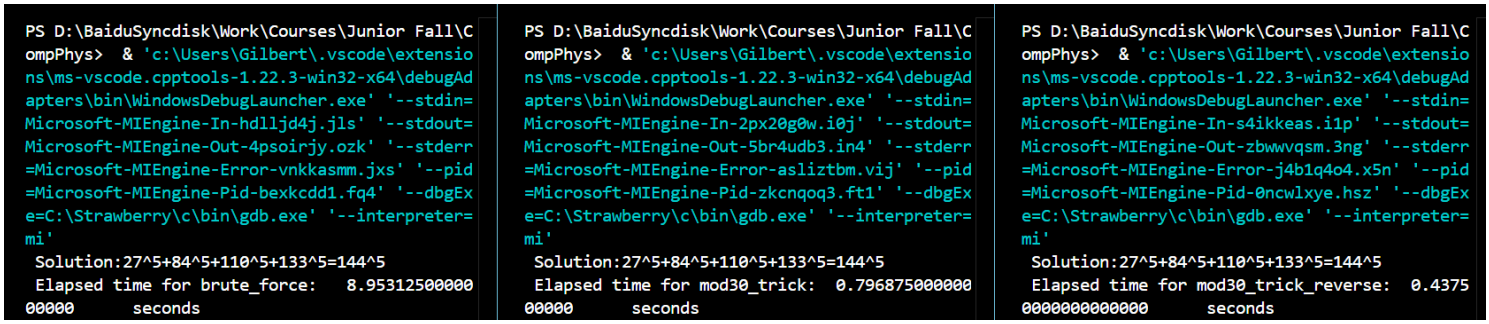


图 1: Fortran 运行结果对比

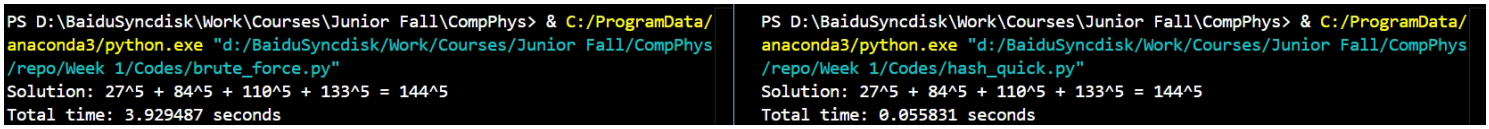


图 2: Python 运行结果对比

2 题目 2：24 点问题

2.1 题目描述

The 24-point game is one of the main puzzle games we played as a child. The game involves drawing four cards from a deck of playing cards and using any of the operations addition, subtraction, multiplication, and division on the four cards to make the result 24. For example, using the numbers 2, 3, 4, and 6, the calculation $(((4+6)-2)*3)=24$ yields 24, and the fastest to figure it out wins. Please use Fortran90 programming to solve the solutions for the 24-point game.

2.2 程序描述

本题作为递归回溯基本功的经典考题 (LeetCode 679), 兼具考察算法知识与编程语言掌握情况的双重作用。递归回溯最暴力的思路是：从给定的 n 个数中有序取 2 个数进行四则运算，得到 8 种运算结果与剩余元素组成 8 种新的序列，然后对新序列递归调用自身，直到序列中只剩下一个数，判断是否为 24。不难发现，这样的路径数为

$$T_1(n) = 8^{n-1}\binom{n}{2}\binom{n-1}{2}\cdots\binom{2}{2} = 4^{n-1}\cdot n!(n-1)!$$

对于 $T_1(4)=9216$，搜索空间尚可接受，若是 n 再大些，不剪枝实在难受。

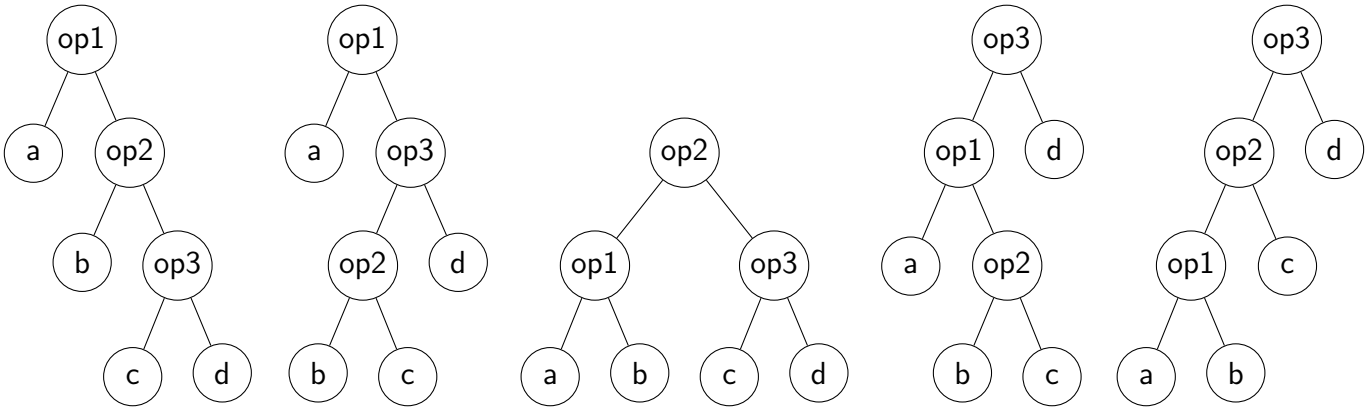
第一个朴素的想法是借助表达式树，即 n 个数作为 n 片叶子的二叉树结构种类，可以用 $Catalan$ 数 C_n 表示。更具体说，为 $n-1$ 个运算符所连接的 n 个数表达式添加括号，构成的合法表达式种数为

$$C_{n-1} = \binom{2n-2}{n-1}/n.$$

例如，对于 $n=4$，$C_3=5$：

$$(a\,\mathrm{op1}\ (b\,\mathrm{op2}\ (c\,\mathrm{op3}\ d))),\quad (a\,\mathrm{op1}\ ((b\,\mathrm{op2}\ c)\,\mathrm{op3}\ d))$$

$$((a\,\mathrm{op1}\ b)\,\mathrm{op2}\ (c\,\mathrm{op3}\ d)),\quad ((a\,\mathrm{op1}\ (b\,\mathrm{op2}\ c))\,\mathrm{op3}\ d),\quad (((a\,\mathrm{op1}\ b)\,\mathrm{op2}\ c)\,\mathrm{op3}\ d)$$



再考虑 n 个数的全排列与各个数之间的运算符种类，可以得到总路径数

$$T_2(n)=4^{n-1}\cdot n!\cdot C_{n-1}=4^{n-1}\cdot n!\cdot\binom{2n-2}{n-1}/n=4^{n-1}\cdot(2n-2)!/(n-1)!$$

对于 $T_2(4)=7680$，搜索空间已经减少。

另一种剪枝策略是从加法和乘法的交换律入手，即挑选两个数进行四则运算时，实际上只要考虑 6 种运算结果，这可以将路径数目降至 $T_3(n)=3^{n-1}\cdot n!(n-1)!$，对于 $T_3(4)=3888$，搜索空间更小。但考虑到两个阶乘复杂度的后发优势，$n=8$ 时 $T_3(8)\approx 4\times10^{11}>T_2(8)\approx 3\times10^{11}$.

自然还会联想到使用分治法，且对于 4 个数的运算，可以先挑选出一个与 24 进行四则逆运算，再与前述 $n-1$ 个数的运行结果进行对比，这样的" 路径数" 为 $T_3(n)=n\cdot4\cdot T_2(n-1)=4\cdot3^{n-1}\cdot(n!)^2$，对于 $T_3(4)=1296$，" 路径数" 的确更少，但考虑到查找比对的开销，实际不一定划算。

本程序最终设计为允许用户最多输入 8 张牌，允许输入整数或者 A/J/Q/K(含小写)，综合考虑复杂度与 Fortran 特性，采用了第 3 种剪枝策略。对于结果的输出，可以将所有解存储再一并输出，但这涉及到未知数目的存储，且在 $n>4$ 时极大影响运行效率（多个数的运算更有可能在较浅搜索区域寻找到解并提前结束），最终决定只输出第一个找到的解。不过 `game24_ultra.f90` 中，对 $n\geq6$ 引入了进度条与 `OpenMP` 多线程并行优化，可能同时输出不止一个解。

判断结果正误的精度最初设置为 1×10^{-6}，比 Fortran 内置的单精度 `epsilon(1.0)` 大些，但仍不能正确判定 $8/(3-8/3)=24$，便放宽为了 1×10^{-4}, 对于 4 个 $[1,13]$ 的运算尚未发现误判，但在更大规模的测试中已经发现漏洞，且转为双精度的运算开销实在过大，故在最终结果输出时加注括号与机器运算值，供用户自行检验。

`./Codes/Problem 2` 中有 `game24_promax.f90` 与 `game24_ultra.f90` 两个版本，下文的代码分析以前者为例，后者仅是在 GPT o1 建议下引入了多线程优化与进度条（实现原理为：提前存储 $n=6,7,8$ 路径数，与已调用次数进行比对）。多线程优化在无解情况测试中被验证是显著有效的。在终端进入其目录，分别运行

`gfortran game24_promax.f90 -o promax_test`、`gfortran -O -fopenmp game24_ultra.f90 -o ultra_test`

可以编译运行。文件夹中还有 `game24_promax.exe` 与 `game24_ultra.exe`, 在大多数 Windows 系统下可以直接运行。如果是 `x86_64` 架构，建议使用 `_x64.exe` 版本. 为了达到最优的运行速度，**建议使用**

`gfortran -O3 -march=native -fopenmp game24_ultra.f90 -o ultra_local` 命令编译 `Ultra` 的本地版本，但在移植到其它架构时可能性能下降。

2.3 伪代码

主程序与用户交互，调用 `Recursive Subroutine solve_24`，而 `create_new_arrays` 生成下一轮数字与表达式。

Algorithm 6: Main Program: game24_promax

Input: n: int, a: Array(float, len=n)

Output: *sol*: bool

```
repeat
    repeat
        read n;                                  // Enter number of cards between 1 and max_limit
    until 1 ≤ n ≤ max_limit;
    allocate numbers, expressions;               // Allocate memory for arrays
    for i ← 1 to n do
        read a[i];                               // Enter card value i
        call convert_to_number;                  // Convert card to number
        write expressions[i], numbers[i];        // Store as string
        call remove_decimal_zeros;               // Clean up decimal
    end
    sol ← false;
    call solve_24;                               // Solve the 24-point game
    if sol == false then
        print No solution found.;
    end
    deallocate numbers, expressions;             // Deallocate memory
    read play_again ;                            // Play again? (y/n)
until play_again ≠ 'y' and ≠ 'Y';
```

Algorithm 7: Subroutine: create_new_arrays

Input: `nums`: Array(float), `exprs`: Array(string), `idx1, idx2`: int, `result`: float, `new_expr`: string

Output: `new_nums`: Array(float), `new_exprs`: Array(string)

```
n ← size(nums);                                  // Get the size of input arrays
allocate new_nums(n − 1), new_exprs(n − 1);      // Allocate memory for new arrays
j ← 0;
for i ← 1 to n do
    if i ≠ idx1 and i ≠ idx2 then
        j ← j + 1;
        new_nums(j) ← nums(i);                   // Copy remaining numbers
        new_exprs(j) ← exprs(i);                 // Copy remaining expressions
    end
end
new_nums(n − 1) ← result;                        // Add result to new arrays
new_exprs(n − 1) ← new_expr;                     // Add new expression to new arrays
```

程序的核心逻辑是下面的 **Recursive Subroutine solve_24**，

Algorithm 8: Recursive Subroutine: solve_24

Input: *nums*: Array(float, len=n), *exprs*: Array(string, len=n), *found*: bool

Output: *found*: bool

$n \leftarrow$ `size`(*nums*); `// Get the size of the array`

if *found* **then**

  return ; `// If solution is already found, exit recursion`

end

if $n = 1$ ***and*** `abs`*(nums*$[1] - 24.0) < 1 \times 10^{-4}$ **then**

  print *Solution found: exprs*$[1]$*, set found* $\leftarrow$ **true**; `// Base case: check if number equals 24`

  return;

end

for $i \leftarrow 1$ ***to*** $n - 1$ **do**

  for $j \leftarrow i + 1$ ***to*** n **do**

    $a, b \leftarrow nums[i], nums[j]$;

    $expr\_a, expr\_b \leftarrow exprs[i], exprs[j]$; `// Get numbers and their expressions`

    for $op \in \{+, -, *, /\}$ **do**

      if $op = /$ ***and*** `abs`*(b)* $< 1 \times 10^{-6}$ **then**

        continue ; `// Skip division by zero`

      end

      $result \leftarrow a\ op\ b$; `// Apply operator on` a `and` b

      $new\_expr \leftarrow$ (`trim`$(expr\_a) + op +$ `trim`$(expr\_b)$); `// Build new expression`

      call *create_new_arrays*; `// Update arrays with` i`,` j`, result, and new expression`

      call *solve_24(new_nums, new_exprs, found)*; `// Recursive call with updated arrays`

      if *found* **then**

        return;

      end

      if $op \in \{-, /\}$ ***and*** *(*$op \neq /$ ***or*** `abs`*(*$a) \geq 1e - 6$*)* **then**

        $result \leftarrow b\ op\ a$; `// Apply reverse operation`

        $new\_expr \leftarrow$ (`trim`$(expr\_b) + op +$ `trim`$(expr\_a)$);

        call *create_new_arrays*; `// Update arrays with` i`,` j`, result, and new expression`

        call *solve_24(new_nums, new_exprs, found)*; `// Recursive call with reverse operation`

        if *found* **then**

          return;

        end

      end

    end

  end

end

`convert_to_number`,`remove_decimal_zeros` 仅用于输入输出格式化处理，原理简单，具体实现不再赘述。下面给出一些输入输出实例。

2.4 输入输出实例

对于本程序，用户需要先输入牌数 $n \in [1, 8]$，再输入 n 张牌，可以是整数或者 A/J/Q/K(含小写)。程序会输出第一个找到的解 `Solution found:xxx = 24(24.000xxxx)`，如果没有解则输出 `No valid solution found.`，之后程序会询问是否继续，输入 `Y` 或 `y` 则继续，否则退出。过程中任何错误输入会被拒绝并要求重新输入。在 `Ultra` 版本中，若 $n \geq 6$ 则会启用多线程优化并显示进度条，每 1% 更新一次进度条。下列表格为在相应输入的计算结果

	Input		Output
Index	n	Cards	Solution
①	4	(A, 2, 3, 4)	(4*(3+(1+2)))= 24 (24.0000000)
②	4	(3, 3, 8, 8)	(8/(3-(8/3)))= 24 (24.0000057)
③	6	(174, 985, 244, 192, 784, 454)	No valid solution found.
④	7	(174, 985, 244, 192, 784, 454,520)	(((454*(520-244))-(192*754))/(174-985))= 24 (24.0000000)
⑤	8	(17495, 3, -7, q, Q, a, A, 74)	((12+12)+((1+1)/((74–7)*(17495+3))))= 24 (24.0000019)

表 1: 计算 24 点问题的结果实例

通过对比表格中的 ① 与 ②，可以清楚地看到，尽管得到的结果在数学上接近 24，但由于单精度浮点运算的特性，仍然出现了微小但显著的误差。这种误差凸显了我们在前文中提到的误差判定问题。在 ③ 中，尽管未能找到接近 24 的有效解，但经过极限优化后的 `Ultra` 版本能够在几秒钟内完成整个搜索，而 `Promax` 版本则略微显得缓慢。

对于 ④，令人意外的是，程序成功找到了一个精确的解，且搜索速度极快。推测这可能是因为该解位于搜索空间的浅层区域，因而耗时较短。在 ⑤ 中，程序输出了一个错误的解，问题仍然源于浮点数误差判定的影响。目前，尚未找到解决这一问题的有效方法，仍需进一步优化与探索。下面给出程序实际运行其它一些组合的截图。

很久没有这么开心地探索喜欢的问题了，上一次或许追溯到和大一室友玩 *Stable Diffusion* 的时候，怀念他和那张 *3080ti*，给那段特殊日子带来了一些亮色。这次的 *24_point* 也已经忘了只是个 *Fortran* 练手作业，但写得很开心，尽管接下来我不得不去面对恐怖的量力和电动作业。往后这样的机会恐怕越来越少吧，下次问问在东京的乙桑，那家伙鬼点子多。

```
Enter the number of numbers (1 to 8): 1.1
Invalid input. Please enter an integer between 1 and 8.
Enter the number of numbers (1 to 8): 4
Enter 4 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: 3
Enter value for card 2: 8
Enter value for card 3: 8
Enter value for card 4: 3
Solution found:(8/(3-(8/3)))= 24 ( 24.000006 )
Play again? (Enter y/n to continue or any other key to exit): y
Enter the number of numbers (1 to 8): 6
Enter 6 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: a
Enter value for card 2: j
Enter value for card 3: q
Enter value for card 4: Q
Enter value for card 5: K
Enter value for card 6: A
Solution found:(((1+11)+(12+12))-(13-1))= 24 ( 24.000000 )
Play again? (Enter y/n to continue or any other key to exit): y
Enter the number of numbers (1 to 8): 8
Enter 8 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: -5
Enter value for card 2: a
Enter value for card 3: 17
Enter value for card 4: q
Enter value for card 5: 23
Enter value for card 6: 25
Enter value for card 7: 13
Enter value for card 8: 58
Solution found:((-5+1)*((13+58)-((17+12)+(23+25))))= 24 ( 24.000000 )
Play again? (Enter y/n to continue or any other key to exit): y
Enter the number of numbers (1 to 8): 1
Enter 1 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: 24
Solution found:24= 24 ( 24.000000 )
Play again? (Enter y/n to continue or any other key to exit): y
Enter the number of numbers (1 to 8): 3
Enter 3 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: 3.3
 Invalid input. Please enter an integer or valid card symbol (A, J, Q, K).
Enter value for card 1: 3.7
 Invalid input. Please enter an integer or valid card symbol (A, J, Q, K).
Enter value for card 1: 4
Enter value for card 2: 97
Enter value for card 3: 582
Solution found:(582*(4/97))= 24 ( 24.000000 )
Play again? (Enter y/n to continue or any other key to exit): 柜子动了我不玩了
 Exiting the game...
```

图 3: Promax 运行实例

```
Enter the number of numbers (1 to 8): 8
Enter 8 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: 4
Enter value for card 2: 64
Enter value for card 3: 3213
Enter value for card 4: 465
Enter value for card 5: 31
Enter value for card 6:
 Invalid input. Please enter an integer or valid card symbol (A, J, Q, K).
Enter value for card 6: 54
Enter value for card 7: 13
Enter value for card 8: 45
[==================================================] 100.0%
 Solution found:(((4-64)+(465-3213))/((13-45)-(31+54)))= 24 (   24.0000000      )
Play again? (Enter y/n to continue or any other key to exit): y
Enter the number of numbers (1 to 8): 4
Enter 4 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: 3
Enter value for card 2: 3
Enter value for card 3: 8
Enter value for card 4: 8
 Solution found:(8/(3-(8/3)))= 24 (   24.0000057      )
Play again? (Enter y/n to continue or any other key to exit): y
Enter the number of numbers (1 to 8): 1.1
Invalid input. Please enter an integer between 1 and 8.
Enter the number of numbers (1 to 8): 6
Enter 6 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: 13
Enter value for card 2: 15
Enter value for card 3: 3135
Enter value for card 4: 12
Enter value for card 5: 35
Enter value for card 6: 45
[==================================================] 100.0%
 Solution found:(((3135+45)-(12*(13*15)))/35)= 24 (   24.0000000      )
Play again? (Enter y/n to continue or any other key to exit): y
Enter the number of numbers (1 to 8): 8
Enter 8 numbers or card values (A=1, J=11, Q=12, K=13).
Enter value for card 1: 13
Enter value for card 2: 54
Enter value for card 3: 12
Enter value for card 4: 123
Enter value for card 5: 45
Enter value for card 6: 54
Enter value for card 7: 54
Enter value for card 8: 65
[==================================================] 100.0%
 Solution found:(((45+54)-(54+65))-((13+54)+(12-123)))= 24 (   24.0000000      )
[==================================================] 100.0%
 Solution found:(((54-13)-(45-54))-((54*65)/(12+123)))= 24 (   24.0000000      )
[==================================================] 100.0%
 Solution found:(((45+54)-(65+(13+54)))-(54+(12-123)))= 24 (   24.0000000      )
[==================================================] 100.0%
 Solution found:((65+(13-54))+((54-54)*(45+(12+123))))= 24 (   24.0000000      )
[==================================================] 100.0%
 Solution found:(((123+54)-(54+65))-((54-45)+(13+12)))= 24 (   24.0000000      )
```

图 4: Ultra 运行实例