Homework1

Zric

January 8, 2026

1 Problem1: solving the equation $a^5 + b^5 + c^5 + d^5 = e^5$

1.1 problem description

give all the possible solution of the equation below, N stands for natural numbers (including 0)

$$a^5 + b^5 + c^5 + d^5 = e^5 \qquad a, b, c, d, e \in N \quad a, b, c, d, e \in [0, 200]$$

1.2 algorithm description

First split the quaternion into two pairs. Enumerate all pairs (a, b) satisfying $0 \leqslant a \leqslant b \leqslant n = 200$, calculate the value of $a^5 + b^5$, store them in the array `sums`, and store them pairwise with (a, b).

Then enter the subroutine `mergesort_pairs` to sort `sums` in ascending order according to the value of $a^5 + b^5$, and (a, b) are sorted synchronously. Next, enter the main loop. For each e^5 and each `sum_ab=sums[i]`, calculate `need` $= e^5 -$ `sum_ab`. Then enter the subroutine `equal_range` and use binary search to locate the indices of all items in `sums` equal to `need`, which can then be paired to obtain all (c, d).

The binary search location steps are: first input $[low, high]$ as the initial upper and lower bounds of the index. Take $index = (low + high)/2$ according to the bisection method, judge the size of `sum[index]` and `need`, and then use $(low + high)/2$ to update the upper or lower bound of the index. Iterate repeatedly to finally determine the index. The complexity of the main loop is $O(n^3 \log(n))$.

In the output step, sort a, b, c, d in ascending order, eliminate duplicates in the solutions obtained from the main loop, and then output them, implemented through the subroutine `output_unique_simple`.

1.3 pseudo code

below is the pseudo code description:

```
Input:
  n = 200

1. Precompute fifth powers:
   for i = 0..n:
       pow5[i] = i^5 (64-bit integer)

```

```
2. Generate all pairs (a,b) with a  b and their sums:
   npairs = (n+1)*(n+2)/2
   allocate arrays pa[1..npairs], pb[1..npairs], sums[1..npairs]
   j = 0
   for a = 0..n:
     for b = a..n:
       j = j + 1
       pa[j] = a
       pb[j] = b
       sums[j] = pow5[a] + pow5[b]

3. Sort by sums ascending while permuting (a,b) in the same order:
   mergesort_pairs(sums, pa, pb, 1, npairs)
   minSum = sums[1]; maxSum = sums[npairs]

4. Main search (O(n^3) + outputs):
   for e = 0..n:
     e5 = pow5[e]
     for i = 1..npairs:
       need = e5 - sums[i]

       // de-dup and range pruning
       if need < sums[i]: continue
       if need < minSum or need > maxSum: continue

       // binary search for all j with sums[j] == need
       (L, R) = equal_range(sums, npairs, need)  // uses lower_bound
          + upper_bound

       if L <= R:
         if need == sums[i]:
            j0 = max(L, i)   // enforce i <= j to avoid duplicates
               when sums equal
         else
            j0 = L
         for j = j0..R:
            output solution: (a=pa[i], b=pb[i], c=pa[j], d=pb[j], e)

Helper routines:
- mergesort_pairs: stable merge sort on sums and co-reorder pa/pb.
- lower_bound(s, n, value): binary search for first index with s[idx]
      value.
- upper_bound(s, n, value): binary search for last index with s[idx]
     value.
- equal_range: combines both to return [L, R] of all indices with s[
    idx] == value;
               returns empty range (L=1, R=0) if not found.
```

2 Problem 2: 24 Point Game Solver

2.1 Problem Description

The 24 Game is a classic mathematical puzzle game. Given numbers on four playing cards (usually 1 to 13, corresponding to A, 2-10, J, Q, K), players rearrange the order of the numbers and use different operators (addition, subtraction, multiplication, division) and parentheses between the numbers to calculate the target number 24. For example, 2, 3, 4, 6 can be calculated as (((4+6)-2)*3)=24.

2.2 Program Description

This program first validates the input of 4 numbers, ensuring they are integers and within the range of 1-13. Then, the exhaustive method is used to process all possible combinations. Therefore, the time complexity of the program is $O[(n_{number}!)*n_{operator}^{n_{number}-1}*n_{structure}]$, which is only applicable when the count of numbers is small:

- For numbers, $n_{number} = 4$, there are $4! = 24$ combinations.
- For operators, $n_{operator} = 4$, there are $4^3 = 64$ combinations.
- When numbers and operators are fixed, there are 5 parenthesis structures in total, $n_{structure} = 5$.

To obtain all possible final expressions, every case must be considered. The logic is as follows:

1. First consider the permutation of 4 numbers, using the `permut` subroutine in the program to traverse the numbers.

2. Then consider 64 combinations of 3 operators, using the `try_opt` subroutine in the program to traverse the operators.

3. Then fix the positions of "numbers + operators" obtained from the above two steps and input them into the `check_forms` subroutine. It lists all 5 parenthesis structures and verifies whether the value of the expression equals 24. If a solution exists, define boolean `found=.true.` and output the satisfying expression directly.

If there is no output, then `found=.false.`, output "no solution found.", and the program ends.

2.3 Pseudo Code

Below is the pseudo code description:

```
Input:
  - Four integers x[1..4], each in {1..9}
Output:
  - All valid infix expressions that evaluate to 24; otherwise "No
     solution found"

Constants:
  - OPS = { '+', '-', '*', '/' }
  - EPS = 1e-6  (numeric tolerance)

1. Read x[1..4] and validate that each value is in [1, 9].
```

```
11
12 2. Generate:
13    - PERMS = all 4! permutations of x
14    - OPCOMBS = all length-3 operator tuples from OPS (Cartesian
          product)
15
16 3. found = false
17
18 4. For each permutation p = (a, b, c, d) in PERMS:
19      For each (op1, op2, op3) in OPCOMBS:
20        For each parenthesis form F in:
21          F1: ((a op1 b) op2 c) op3 d
22          F2: (a op1 (b op2 c)) op3 d
23          F3: (a op1 b) op2 (c op3 d)
24          F4: a op1 ((b op2 c) op3 d)
25          F5: a op1 (b op2 (c op3 d))
26          Do:
27            value ← Evaluate(F with (a,b,c,d) and (op1,op2,op3))
28            If |value - 24|   EPS:
29              Print the concrete expression string for F, p, (op1,op2,
                    op3)
30              found = true
31
32 5. If found = false:
33      Print "No solution found"
```

3 Input and Output

The gfortran compiler version is tdm64-gcc-10.3.0-2.

3.1 Problem 1

The source code is `equation.f90`. Enter `gfortran equation.f90 -o equation.exe` in the terminal to compile, and enter `./equation.exe` to run.

Only partial output examples are included. Most trivial solutions $(a, b, c, d, e) = (0, 0, 0, k, k), k \in N$ are omitted. Here, (a, b, c, d, e) are sorted in ascending order for output. There is only 1 non-trivial solution $(27, 84, 110, 133, 144)$ within the range of the problem.

```
a=          0  b=          0  c=          0  d=        141  e=        141
a=          0  b=          0  c=          0  d=        142  e=        142
a=          0  b=          0  c=          0  d=        143  e=        143
a=          0  b=          0  c=          0  d=        144  e=        144
a=         27  b=         84  c=        110  d=        133  e=        144
a=          0  b=          0  c=          0  d=        145  e=        145
a=          0  b=          0  c=          0  d=        146  e=        146
a=          0  b=          0  c=          0  d=        147  e=        147
a=          0  b=          0  c=          0  d=        148  e=        148
a=          0  b=          0  c=          0  d=        149  e=        149
a=          0  b=          0  c=          0  d=        150  e=        150
```

3.2 Problem 2

The source code is `24-points.f90`. Enter `gfortran 24-points.f90 -o 24-points.exe` in the terminal to compile, and enter `./24-points.exe` to run.

Input Example:

```
 Enter 4 numbers (1-9):
8
3
3
1
```

Output Example:

```
(((8+1)*3)-3 = 24
(((8-1)*3)+3 = 24
(((8+1)*3)-3 = 24
(((8-1)*3)+3 = 24
3+((8-1)*3)) = 24
(3*(8+1))-3 = 24
(3*(8-1))+3 = 24
3+(3*(8-1)) = 24
3-(3*(1-8)) = 24
3-((1-8)*3)) = 24
(3*(1+8))-3 = 24
3+((8-1)*3)) = 24
(3*(8+1))-3 = 24
(3*(8-1))+3 = 24
3+(3*(8-1)) = 24
3-(3*(1-8)) = 24
3-((1-8)*3)) = 24
(3*(1+8))-3 = 24
(((1+8)*3)-3 = 24
(((1+8)*3)-3 = 24
```