Pauls Zahlenreihen

Test:

Setzen Sie die folgenden Zahlenreihen sinnvoll fort. Sie haben dafür 10 Minuten Zeit.

1	3	6	9	____
2	3	2	1	____
3	8	6	4	____
4	2	5	8	____
5	0	2	4	____
6	3	6	12	____
7	3	9	27	____
8	2	4	8	____
9	8	12	16	____
10	– 4	16	– 64	____
11	25	22	19	____
12	54	48	42	____
13	24	58	92	____
14	– 2	– 5	– 8	____
15	17	19	21	____
16	– 3	9	– 27	____
17	129	272	415	____

18	4	2	5	3	____
19	0	1	3	7	____

20	1	1	2	3	____
21	7	10	9	12	____
22	1	4	9	16	____
23	2	3	5	9	____
24	5	6	8	9	____
25	3	6	11	18	____
26	5	11	19	29	____
27	4	11	15	26	____
28	0	1	4	9	____
29	3	7	15	31	____
30	1	2	6	24	____
31	2	4	7	11	____
32	6	8	5	7	____
33	3	2	7	6	____
34	7	8	10	13	____
35	12	15	8	11	____
36	15	13	16	12	____
37	17	15	18	14	____
38	2	12	21	29	____
39	9	21	39	63	____
40	3	2	6	5	____
41	5	7	6	8	____
42	2	5	9	19	____
43	9	20	6	17	____
44	2	8	18	32	____

45	18	36	60	90	____
46	3	12	27	48	____
47	8	15	26	41	____
48	2	6	14	26	____
49	– 4	– 16	– 36	– 64	____
50	12	48	108	192	____
51	149	286	419	548	____
52	28	15	6	1	____
53	– 1	– 17	– 41	– 73	____
54	28	33	31	36	____

55	0	1	2	1	4	____	
56	3	5	10	12	24	____	
57	8	12	10	16	12	____	
58	0	– 2	– 2	0	4	____	
59	0	0	1	1	0	____	
60	8	10	14	18	26	____	
61	4	5	6	8	10	____	
62	6	9	18	21	42	____	
63	3	6	8	16	18	____	
64	0	1	1	2	3	____	
65	0	0	1	0	0	1	____
66	9	8	11	10	14	13	____

67	10	13	14	13	16	17	16	____			
68	14	5	15	7	21	14	42	____			
69	16	7	21	13	39	32	96	____			
70	2	3	5	6	7	9	10	____			
71	20	19	22	23	22	25	26	____			
72	239	1945	6507	15569	30007	53001	80819	_____			
73	3	870	3225	7830	15495	27174	44253	_____			
74	148	84	52	36	28	____					
75	4	5	3	5	4	6	4	____			
76	5	3	6	3	9	5	20	____			
77	14	11	33	30	10	7	21	____			
78	9	7	14	11	33	29	116	____			
79	19	15	18	9	13	16	8	____			
80	5	2	6	2	8	3	15	____			
81	12	10	20	17	51	47	188	____			
82	12	10	13	17	12	18	25	____			
83	1	2	2	3	3	3	4	4	4	4	____

ENDE

Auflösung:

1) 12	13) 126	25) 27	37) 19	49) -100	61) 13	73) 69414
2) 0	14) -11	26) 41	38) 36	50) 300	62) 45	74) 24
3) 2	15) 23	27) 41	39) 93	51) 673	63) 36	75) 5
4) 11	16) 81	28) 16	40) 9	52) 0	64) 5	76) 15
5) 6	17) 558	29) 63	41) 7	53) -113	65) 0	77) 18
6) 24	18) 6	30) 120	42) 37	54) 34	66) 18	78) 111
7) 81	19) 15	31) 16	43) 3	55) 1	67) 19	79) 4
8) 16	20) 5	32) 4	44) 50	56) 26	68) 36	80) 9
9) 20	21) 11	33) 11	45) 126	57) 20	69) 90	81) 180
10) 256	22) 25	34) 17	46) 75	58) 10	70) 11	82) 17
11) 16	23) 17	35) 4	47) 60	59) 0	71) 25	83) 5
12) 36	24) 11	36) 17	48) 42	60) 34	72) 130465	

Erklärung:

Zahlenreihen gehörten schon immer zur Standardausrüstung von Intelligenztests. Dahinter liegt die Vorstellung, dass das Erkennen von Regelmäßigkeiten keine mechanische Tätigkeit ist, sondern mitunter das fordert, was wir unter „Intelligenz“ verstehen.

Genau diese Intelligenz versuchen Programmierer seit jeher in ihren Algorithmen erfassen. Als Ergebnis meiner bescheidenen Bemühungen möchte ich nun Paul vorstellen, ein Programm, dass darauf spezialisiert ist, Zahlenreihen in allen Formen in die Unendlichkeit fortzuführen. Dabei hat Paul ein Niveau erreicht, dass sich nicht vor den menschlichen Mitstreitern verstecken braucht: Paul kann selbstständig die ersten 73 Aufgaben des Tests korrekt lösen. Arbeitszeit: 2 Sekunden. Zu finden ist Paul unter dal-research.de/code/2016/04/02/paul/.

Paul nutzt dazu keine Magie. Er braucht keine besondere Rechenleistung oder große Datenbanken. Das Programm besteht aus 230 Zeilen C#-Code (kompletter Code findet sich im Anhang), die ausführbare Datei beläuft sich auf 10kB. Sein Trick besteht darin, dass er das Problem aus einer cleveren Perspektive sieht. Schauen wir zu, wie er eine Zahlenfolge analysiert:

```
1       4       9       16      25      36
  +3      +5      +7      +9      +11
    (i +2)  (i +2)  (i +2)  (i +2)
```

Wenn Paul versucht, die Folge der Quadratzahlen fortzusetzen, betrachtet er zuerst jeweils zwei aufeinanderfolgende Zahlen. Hier erkennt er, dass immer etwas hinzugefügt wurde. Nachdem er sich seine Beobachtungen notiert hat, schaut er sich zwei seiner Beobachtungen an und erkennt: Es wird immer eine Addition ausgeführt, aber der Summand steigt immer um zwei an. Das notiert er sich wieder und schaut sich diese Beobachtungen an. Er stellt nun fest, dass alle Einträge in der Zeile gleich sind. Er nimmt also an, dass sich dieses Muster immer so fortsetzt.

Paul ist in der Lage, Quadratzahlen, Fakultäten, Fibonaccizahlen, Exponentialfunktionen, uvm zu erkennen, doch er weiß von diesen Begriffen nichts. Er sieht nur Zahlen, die sich verändern, er sieht darin Regelmäßigkeiten und sieht in diesen Regelmäßigkeiten wieder ihre besonderen Muster. So ähnlich gehen Menschen auch an diese Aufgaben heran. Wer solche Aufgaben löst, wird sich sicher

die Differenz der Zahlen notiert haben und dann versuchen, darin ein Muster zu entdecken.

Was Paul vom Menschen abhebt ist seine Geschwindigkeit und Präzision. Weil er systematisch alle Möglichkeiten durchsucht, kann er sehr schnell sagen, ob er eine Lösung gefunden hat oder nicht. Falls es eine Lösung gibt, dann findet er sie auch. Findet er keine Lösung, ist es ein Zeichen, dass sich dieses Problem nicht auf eine solche Abfolge reduzieren lässt. Das hat einen sehr menschlichen Moment: Paul kann über nichts nachdenken, dass den Umfang seiner Sprache überschreitet.

Die Aufgaben 74 – 83 verstoßen dabei jeweils gegen einen Aspekt von Pauls Ausdrucksfähigkeit. Bei der Aufgabe 74 zum Beispiel, dass Paul keine Division erkennen kann. Denn die Differenz der Zahlen entspricht der Folge 64, 32, 16, 8, usw. Diese Begrenzung ist bewusst gewählt worden: Das gesamte Programm ist speziell für Ganzzahlen entwickelt worden und ignoriert damit Zahlenfolgen, die Dezimalzahlen enthalten.

Bei den Aufgaben 75 bis 82 ist Paul nicht in der Lage, unabhängig den Wechsel von Operator und Operand zu unterscheiden. Er kann eine Folge wie

+ 2 – 3 – 4 + 5 – 6 – 7 + 8 …

nicht erkennen. Im Grunde müsste man eine Methode einfügen, die eben eine solche Trennung ermöglicht. Der Versuch ist allerdings daran gescheitert, dass der Suchraum plötzlich viel größer geworden ist und das Programm selbst für einfache Probleme sehr lange gebraucht hat.

Aufgabe 83 sprengt den Erwartungshorizont auf wieder andere Art. Für Menschen leicht zu sehen, aber für Paul unverständlich ist die Möglichkeit, die Zahlen zu gruppieren. Hier müsste sich Paul mehr in die Richtung einer allgemeinen Mustererkennung entwickeln.

Trotz all dieser Einschränkungen ist die Leistung von Paul eindrucksvoll – vor allem im Hinblick auf seine Einfachheit. Darin liegt meine Überzeugungen, dass das Wesen unserer Intelligenz im Grunde *einfach* ist. Es sind keine erschreckenden Formeln notwendig oder unverständliche Code. Das Wesen der Intelligenz liegt in der Fähigkeit, die Welt aus der Perspektive zu betrachten, in der die meisten Regelmäßigkeiten am deutlichsten zu Tage treten.

Paul ist ein statisches Programm. Es ist nicht in der Lage, zu lernen. Wie geht es also weiter? Mustererkennungen sind für viele Bereiche des Lebens von Bedeutung und viel mehr dieser Muster können von Paul erkannt werden, wenn sie davor in Zahlen gefasst wurden. Das Anwendungsfeld von Paul lässt sich erweitern, wenn man geeignete Filter vor Paul schaltet. Für Aufgabe 83 könnte ein solcher Filter darin bestehen, die Zahlen zuerst zu gruppieren. Oder unterschiedliche Perioden zu trennen. Oder Messwerte aus der Umgebung für Paul aufzuarbeiten.

Weiterentwickeln kann man auch die Tatsache, dass sich Paul seine Gedanken notieren kann. Paul notiert sich seine Beobachtungen und behandelt diese Notizen, als wären das selber Eingabewerte. Damit kann Paul tatsächlich sich selber beim Denken zuschauen. Dies ist schonmal ein Schritt in Richtung bewusster Reflektion. Pauls nutzt dieses Potenzial natürlich noch nicht voll aus und darin könnte sich für die Zukunft ein Feld ergeben.

Ein Gedanke zum Schluss: Wenn Paul eine Zahlenfolge fortsetzen kann, dann kann man sagen, dass er das Prinzip dahinter verstanden hat. Denn er ist in der Lage, die Reihe ins Unendliche forzusetzen. Das Ergebnis seiner Berechnung ist also nicht nur ein Ergebnis, sondern – und das darf man Paul und damit auch dem Computer eingestehen – ein Stück Wahrheit.

Code (C#):

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace Paul
{
    class MainClass
    {
        private static string s_identity = "i";
        private static string s_add = "+";
        private static string s_minus = "-";
        private static string s_mult = "*";
        private static string s_empty = "x";
        private static string s_matchin = "&";

        private static int ctoi(char c) { return ((int)c) - ((int)'a'); }
        private static int Ctoi(char C) { return ((int)C) - ((int)'A'); }
        private static char itoc(int i) { return ((char)(i + ((int)'a'))); }
        private static char itoC(int i) { return ((char)(i + ((int)'A'))); }

        private static bool run = true;
        private static List<List<string>> layers;
        private static List<List<string>> resultState;
        private const int maxMatchInDepth = 2;
        private static int maxMatchInLength() { return layers[0].Count / 2; }
        private static string[] splitPair(string pair) {
            int spaceIndex = -1;
            char[] input = pair.ToCharArray();
            int s = 0;
            for (int i = 1; i < pair.Length; i++) {
                if (input[i] == '(')
                    s++;
                if (input[i] == ')')
                    s--;
                if (s == 0 && input[i] == ' ') {
                    spaceIndex = i;
                    break;
                }
            }
            return new string[] {
                pair.Substring(1, spaceIndex - 1),
                pair.Substring(spaceIndex + 1, pair.Length - spaceIndex - 2)
            }; }
        private static void printLayer(List<string> layer) {
            layer.ForEach((x) => { Console.Write(x + " "); });
            Console.WriteLine();
        }
        public static void Main(string[] args) {
            Console.WriteLine("Paul says hello!");
            while (run) {
                if (Prepare(args)) {
                    if(Search(1, 0))
                        Output(resultState);
                    else
                        Console.WriteLine("@");
                }
            }
        }
        private static bool Prepare(string[] cmdArgs) {
            string text = "";
            if (cmdArgs.Length == 1) {
                text = cmdArgs[0];
            } else {
                Console.Write(">");
                text = Console.ReadLine();
            }
            if (cmdArgs.Length == 1) run = false;
            if (text == "q") return run = false;
            List<string> input = text.Trim().Split(new char[] { ' ' },
                                                   StringSplitOptions.RemoveEmptyEntries).ToList();
            foreach(string num in input) {
```

```
            long x;
            if (!long.TryParse(num, out x)) {
                Console.WriteLine("'" + num + "' is not a number.");
                return false;
            }
        }
        if (input.Count < 2) {
            Console.WriteLine("Enter at least two numbers.");
            return false;
        }
        layers = new List<List<string>>();
        layers.Add(input);
        for (int row = input.Count - 1; row > 0; row--) {
            var curLayer = new List<string>();
            for (int col = 0; col < row; col++)
                curLayer.Add("");
            layers.Add(curLayer);
        }
        return true;
    }
    private static List<string> Check2(string a, string b) {
        var rules = new List<string>();
        if (a == s_empty || b == s_empty)
            return rules;
        if (a == b)
            rules.Add(s_identity);
        long longA, longB;
        if (long.TryParse(a, out longA) && long.TryParse(b, out longB)) {
            rules.Add("(" + s_add + " " + (longB - longA) + ")");
            rules.Add("(" + s_minus + " " + (-(longA + longB)) + ")");
            if (longA != 0 && longB != 0 && (longB / longA) * longA == longB)
                rules.Add("(" + s_mult + " " + (longB / longA) + ")");
        }
        return rules;
    }
    private static List<string> CheckTree(string a, string b, int row, int col) {
        var rules = new List<string>();
        int pRow = row - 1;
        for (int matchRow=pRow;matchRow>pRow-maxMatchInDepth&&matchRow>=0; matchRow--) {
            for (int matchCol=col;matchCol>col-maxMatchInLength()&&matchCol>=0; matchCol--) {
                if (b != s_empty && layers[matchRow][matchCol] == b) {
                    char rowC = itoC(pRow - matchRow);
                    char colc = itoc(col - matchCol);
                    if (!(rowC == 'A' && colc == 'a'))
                        rules.Add("(" + s_matchin + " " + rowC + colc + ")");
                }
            }
        }
        if (a.StartsWith("(") && b.StartsWith("(")) {
            string[] partsA = splitPair(a), partsB = splitPair(b);
            if (partsA.Length == 2 && partsB.Length == 2) {
                var r1 = CheckTree(partsA[0], partsB[0], row, col);
                var r2 = CheckTree(partsA[1], partsB[1], row, col);
                foreach (var e1 in r1) {
                    foreach (var e2 in r2) {
                        if (e1.IndexOf(s_matchin) < 0)
                            rules.Add("(" + e1 + " " + e2 + ")");
                    }
                }
            }
        } else
            rules.AddRange(Check2(a, b));
        rules.Reverse();
        return rules;
    }
    private static bool isId(string input) {
        var parts = input.Split(new char [] { ' ', '(', ')' });
        bool result = true;
        foreach (var part in parts) {
            if (part.Length > 0 && !(part == s_identity))
                result = false;
        }
        return result;
    }
```

```
        private static bool Search(int row, int col) {
            if (col >= layers.Count - row) { // out of pyramid - lets do some checking
                bool isRegular = true;
                int emptyCount = 0;
                for (int i = 0; i < col; i++) {
                    if (layers[row][i] == s_empty)
                        ++emptyCount;
                    else if (!isId(layers[row][i]))
                        isRegular = false;
                }
                string pattern = layers[row - 1].Last();
                int tolerateEmptys = -1;
                int index = pattern.IndexOf(s_matchin);
                if (index >= 0) {
                    string substr = pattern.Substring(index);
                    tolerateEmptys = ctoi(splitPair(substr)[1].ToCharArray()[1]);
                }
                if (tolerateEmptys < 0 || emptyCount > tolerateEmptys)
                    if (emptyCount > 0) isRegular = false;
                if (isRegular) {
                    resultState = new List<List<string>>();
                    for (int i = 0; i < row; i++)
                        resultState.Add(layers[i].Take(col).ToList());
                    return true;
                } else if (row < layers.Count - 1)
                    return Search(row + 1, 0);
                return false;
            } // proceed next element
            List<string> choices = CheckTree(layers[row - 1][col], layers[row-1][col+1], row, col);
            if (choices.Count==0&& col < maxMatchInLength() - 1 && col < layers[row].Count() - 1) {
                layers[row][col] = s_empty;
                return Search(row, col + 1);
            }
            foreach (var choice in choices) {
                layers[row][col] = choice;
                if (Search(row, col + 1))
                    return true;
            }
            return false;
        }
        private static string ApplyRule(string rule,string input,List<List<string>> state,int row) {
            if (rule == s_identity) return input;
            string[] op = splitPair(rule);
            if (op[0] == s_add) return (long.Parse(input) + long.Parse(op[1])).ToString();
            if (op[0] == s_mult) return (long.Parse(input) * long.Parse(op[1])).ToString();
            if (op[0] == s_minus) return (-(long.Parse(input) + long.Parse(op[1]))).ToString();
            if (op[0] == s_matchin) {
                char[] offsets = op[1].ToCharArray();
                int rowIndex = row - Ctoi(offsets[0]);
                int colIndex = state[row].Count - 1 - ctoi(offsets[1]);
                return state[rowIndex][colIndex];
            }
            string[] val = splitPair(input);
            return "("+ApplyRule(op[0],val[0],state,row)+" "+ApplyRule(op[1],val[1],state,row)+")";
        }
        private static void Grow(List<List<string>> state) {
            var topLayer = state.Last();
            topLayer.Add(topLayer.Last());
            for (int i = state.Count - 2; i >= 0; i--)
                state[i].Add(ApplyRule(state[i+1][state[i+1].Count-2], state[i].Last(), state, i));
        }
        private static void Output(List<List<string>> state) {
            for (int i = state.Count - 1; i > 0; i--)
                printLayer(state[i]);
            while (state.First().Count < 20) Grow(state);
            state.First().ForEach((val) => { Console.Write(val + " "); });
            Console.WriteLine();
        }
    }
}
```