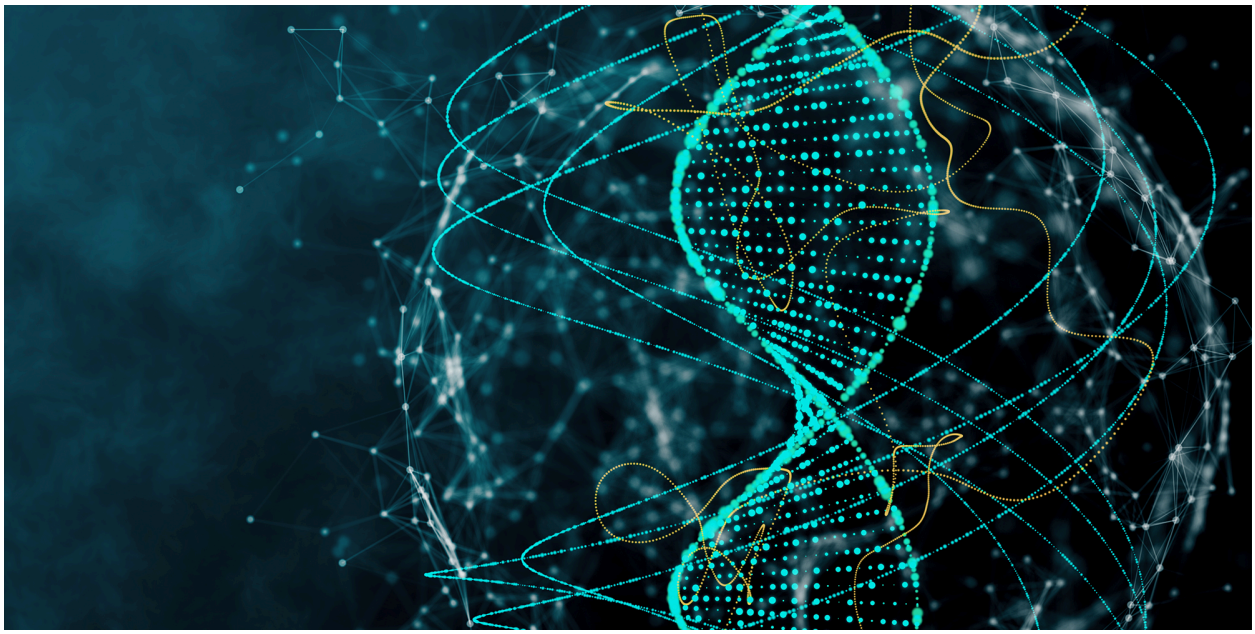
Programowanie Genetyczne

Finalizacja Projektu

Stopyra Jan, Wądrzyk Jakub



7.02.2024

Projekt Systemu

Gramatyka

```
ADD: '+';                EQ : '==';              LPAREN: '(';
SUB: '-';                NEQ: '!=';              RPAREN: ')';
MUL: '*';                LE : '<' ;              LBRACE: '{';
DIV: '/';                LEQ: '<=';              RBRACE: '}';
MOD: '%';                GE : '>' ;
ASS: '=';                GEQ: '>=';              IF: 'if';
                                                 WHILE: 'while';
                         AND: '&&';
                         OR : '||';              PRINT: 'print';
                         NOT: '!' ;              SCAN : 'scan' ;

NUMBER: [0-9]+ ('.' [0-9]+)?;                    TRUE : 'True' ;
NUM_VAR: 'X' [0-9];                              FALSE: 'False';
BOOL_VAR: 'L' [0-9];
```

```
program: expressions;

expressions
   :    ( if_statement
        | while_loop
        | print_call
        | scan_call
        | assignment) expressions?
   ;

if_statement: IF LPAREN bool_value RPAREN LBRACE expressions RBRACE;

while_loop  : WHILE LPAREN bool_value RPAREN LBRACE expressions RBRACE;

print_call
   : PRINT LPAREN numeric_value RPAREN #printNum
   | PRINT LPAREN  bool_value   RPAREN #printBool
   ;

scan_call
   : SCAN LPAREN NUM_VAR  RPAREN #scanNum
   | SCAN LPAREN BOOL_VAR RPAREN #scanBool
   ;
```

```
assignment
   : NUM_VAR  ASS numeric_value #AssignNum
   | BOOL_VAR ASS bool_value #AssignBool
   ;

comparisson_type            : EQ  | NEQ | LE  | LEQ | GE  | GEQ;
logic_operator              : AND | OR  ;
aritmetic_operator_strong   : MUL | DIV | MOD ;
aritmetic_operator_weak     : ADD | SUB ;

bool_value
   : BOOL_VAR                                                  #boolVarVal
   | TRUE                                                      #trueVal
   | FALSE                                                     #falseVal
   | NOT bool_value                                            #notVal
   | numeric_value comparisson_type numeric_value              #compVal
   | bool_value     logic_operator     bool_value              #logicVal
   | LPAREN bool_value RPAREN                                  #parenBoolVal
   ;

numeric_value
   : NUMBER                                                    #numVal
   | NUM_VAR                                                   #numVarVal
   | SUB numeric_value                                         #subVal
   | numeric_value aritmetic_operator_strong numeric_value     #aritStrongVal
   | numeric_value  aritmetic_operator_weak  numeric_value     #aritWeakVal
   | LPAREN numeric_value RPAREN                               #parenNumVal
   ;
```

Taka konstrukcja gramatyki pozwoliła na zbudowanie klasowej reprezentacji każdego węzła w drzewie stochastycznym.

Projekt klas

```
abs Node
├─ abs Expression
│    ├─ abs Assignment
│    │    ├─ NumericAssignment
│    │    └─ BooleanAssignment
│    │
│    ├─ IfStatement
│    ├─ PrintStatement
│    ├─ ScanStatement
│    └─ WhileStatement
│
├─ abs Value
│    ├─ IVariable
│    ├─ IConstant
│    │
│    ├─ abs BooleanValue
│    │    ├─ BooleanConstant : IConstant
│    │    ├─ BooleanNegation
│    │    ├─ BooleanVariable : IVariable
│    │    ├─ ComparisonOperation
│    │    └─ LogicalOperation
│    │
│    └─ abs NumericValue
│         ├─ ArithmeticOperation
│         ├─ NumericConstant : IConstant
│         ├─ NumericNegation
│         └─ NumericVariable : IVariable
│
└─ Program
```

Relacje między klasami są ściśle powiązane z projektem gramatyki.

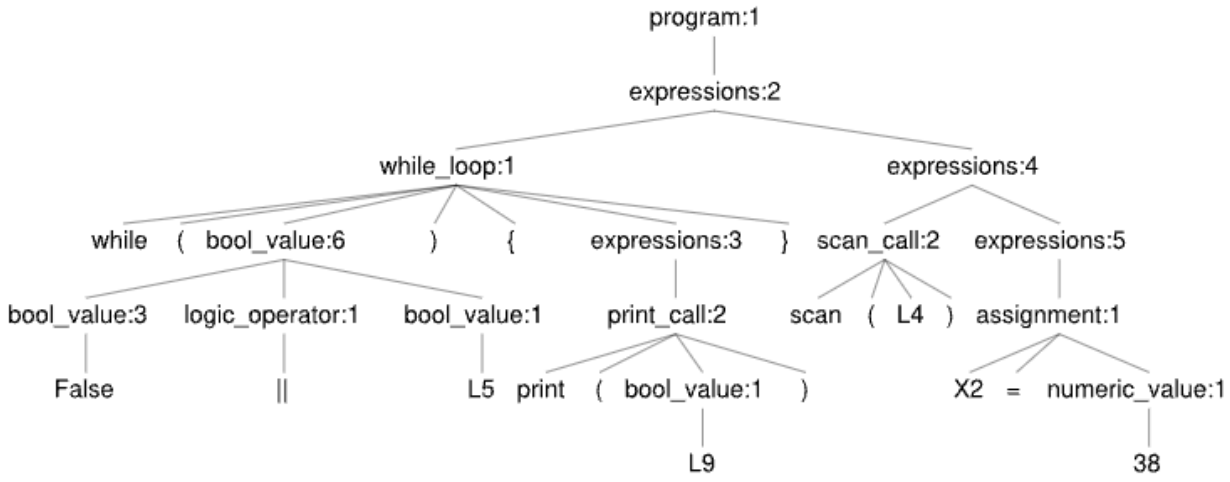
Generator programów

Funkcja grow

Dzięki zaimplementowaniu w każdej klasie statycznego pola opisującego minimalną odległość do liścia, możliwe było generowanie drzew o zadanej głębokości maksymalnej.

Oto przykładowy program wygenerowany przez Grow(6):

```
while (False || L5) {
  print(L9)
}
scan(L4)
X2 = 38
```



Błąd w wyświetlanej przez parser głębokości opiera się na sposobie przechowywania przez nas zmiennych, na wykresie *numeric_value* zajmuje dwa poziomy.

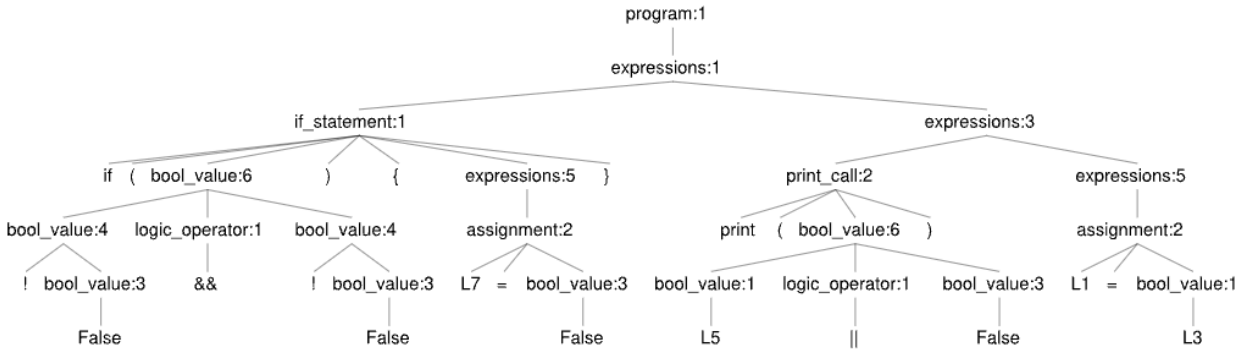
Funkcja grow-full

Metoda została zaimplementowana na podstawie metody grow z dodatkowym parametrem opisującym minimalną odległość do liścia. Gdy odległość minimalna i maksymalna do liścia są równe powstaje drzewo typu full.

Oto przykładowy program wygenerowany przez GrowFull(6):

```
if (!False && !False) {
 L7 = False
}
print(L5 || False)
```

```
L1 = L3
```



jak można zauważyć wszystkie liście są na jednym poziomie co oznacza poprawne działanie metody. (symbole terminalne takie jak *&&*, czy *print* nie są brane pod uwagę)

Interpreter

Interpreter został napisany z pomocą narzędzia Antlr4, które wygenerowało lexer, parser oraz bazowy visitor. Klasa GpVisitor dziedziczy po bazowym visitorze i nadpisuje metody visit dla poszczególnych reguł. Ustawiliśmy zero jako wartość domyślną dla typu liczbowego oraz false dla typu logicznego, aby rozwiązać problem odczytu z zmiennej która nie miała wcześniej przypisanej wartości. Jeżeli liczba użyć funkcji scan() w programie przekroczy długość wektora wejścia, to interpreter zacznie wczytywać wektor wejścia od początku. Interpreter zlicza liczbę wykonanych instrukcji i wywołuje wyjątek, jeżeli ich maksymalna liczba zostanie przekroczona. Jako pojedynczą operację liczymy wczytanie z wejścia, zapisanie na wektorze wyjścia, przypisanie wartości, sprawdzenie warunku pętli oraz sprawdzenie warunku instrukcji if. Klasa InOutWektor służy do komunikacji między biblioteką oraz interpreterem. Zawiera wektor wejścia i wyjścia.

Testy Interpretera

- Program obliczający silnię z 5:

```
X0 = 5
X9 = 1
while(X0 > 0) {
   X9 = X9 * X0
   X0 = X0 - 1
}
print(X9)
```

  Na wektorze output otrzymaliśmy: `double: 120,`

- Program z nieskończoną pętlą (limit operacji - 1000):

```
while(True) {
   X1 = X1 + 1
   print(X1)
}
```

  Wektor output: `double: 1, 2, 3, …, 332, 333,`

Funkcja fitnessu

Funkcja fitnessu opiera się na pliku w formacie WSL (*Whitespace Separated Values*), gdzie w pierwszej części linijki przechowujemy wartości które mają zostać przekazane na wejście, a w drugiej części (po znaku " : ") pożądany wektor wyjścia.

W klasie z funkcją fitnessu zostały zaimplementowane cztery flagi określające za co program ma być karany/nagradzany oraz odpowiadające im wartości kar:

- **restrictOutputLength** - *kara za przekroczenie pożądanej długości wyjścia*
- **restrictOutputPosition** - *rozpatruje daną kolejność wyjścia*
- **checkIfreadInput** - *na podstawie danych z interpretera, kara z nieprzeczytanie całego wejścia*
- **checkUsedInput** - *na podstawie danych z interpretera, kara za użycie niezainicjowanej zmiennej*

Domyślne kary za odpowiednie wykroczenia:

- **outputLengthPun** = *1e9*
- **valuePun** = *1e1*
- **inputNotReadPun** = *1e6*
- **notUsedInputPun** = *1e3*

Funkcja ewaluuje fitness programu poprzez zsumowanie obliczonych kar dla każdego uruchomienia go dla przypadków z pliku.

Największym problemem było odpowiednie zmodyfikowanie wartości kar, tak aby w procesie ewolucji program nie stwierdził, że "bardziej opłaca mu się" zwracać puste wyjście, niż zostać ukaranym za zwrot złej wartości. Podane powyżej wartości są wynikiem wielokrotnych prób i testów na różnych wartościach.

Kod funkcji:

```
class FitnessFunction {
    static readonly double _outputLenghtPunishment = 1e9;
    static readonly double _valuePunishment = 1e1;
    static readonly double _inputNotReadPunishment = 1e9;
    static readonly double _usedInputPunishment = 1e3;
    readonly bool restrictOutputLength = true;
    readonly bool restrictOutputPosition = true;
    readonly bool checkIfReadInput = true;
    readonly bool checkUsedInput = true;
    readonly List<Target> targets;// Lista przypadków testowych
    InOutVector? ioVector;
```

```
    public double Evaluate(Individual individual) {
        double fitness = 0;
        foreach (var target in targets) {
            ioVector = individual.Run(target.Inputs);
            fitness -= EvaluateTarget(target.ExpectedOutputs);
        }
        return fitness;
    }

    public double EvaluateTarget(List<Value> expectedOutput) {
        if (!restrictOutputPosition && expectedOutput.Count == 1) {
            if (ioVector!.output.Count == 0)
                return _outputLenghtPunishment;

            double minRes = double.MaxValue;
            foreach (var val in ioVector.output) {
                double _res = Math.Abs(val.GetNum() - expectedOutput[0].GetNum());
                if (_res < minRes)
                    minRes = _res;
            }
            return minRes;
        }

        double res = 0;
        for (int i = 0; i < expectedOutput.Count; i++) {
            if (i >= ioVector!.output.Count)
                res += _outputLenghtPunishment;
            else
                res += PunishValue(ioVector.output[i].GetNum(), expectedOutput[i].GetNum());
        }

        if (restrictOutputLength)
            res += PunichLength(ioVector!.output.Count, expectedOutput.Count);

        if (checkIfReadInput)
            res += PunishNotReadInput();

        if (checkUsedInput)
            res += PunishUsedInput();

        return res;
    }

    private static double PunishValue(double value, double expectedValue) {
        return _valuePunishment * Math.Abs(value - expectedValue);
    }

    private static double PunichLength(int length, int expectedLength) {
        return _outputLenghtPunishment * Math.Abs(length - expectedLength);
    }

    private double PunishNotReadInput() {
        var read = ioVector!.scanned.Where(x => x == true).Count();
        if (read < ioVector.input.Count)
            return _inputNotReadPunishment * (ioVector.input.Count - read);
        return 0;
    }

    private double PunishUsedInput() {
        return -_usedInputPunishment * ioVector!.used.Sum();
    }
}
```

Przykładowe zadania testowe systemu GP

✅ 1.1.A, B, C

*Program powinien wygenerować na wyjściu (na dowolnej pozycji w danych wyjściowych) liczbę **1**, **789**, **31415**. Poza liczbą może też zwrócić inne liczby.*

Parametry funkcji fitnessu:

```
restrictOutputLength    = false;
restrictOutputPosition  = false;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 0;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

Uczenie postanowiliśmy rozpocząć z powyższymi parametrami funkcji.

Dwie pierwsze flagi zostały ustawione na false, dzięki czemu funkcja mogła estymować fitness na podstawie najlepiej dopasowanej wartości z wektoru wyjścia do tej pożądanej.

Rezultaty:

Po odpowiednio 1, 49, 10 generacjach otrzymaliśmy funkcje spełniające podane wymagania:

A	program	`print(True)` `X1 = X5`
	output	**1**,

B	program	`print(-(99 % X2))`
		`print(99)`
		`print(11)`
		`print((-48 + ((((((-48 + (((((-(-94) + (76 * 11)) + (11 * 76)) / -48) / -94) * 11)) + 11) + (11 * 76)) / -94) / -94) * 11)) + (11 * 76))`

	output	-99, 99, 11, **789.0000406842254**,
C	program	`print(((-((X0 / -38) / X1) * -(X9 / X3)) * (15 / (X4 / X7))) + (99 * (((((((99 + 99) + 20) + 99) + 17) + 99) + 20) + 99)))` `print(((99 * (((99 + 99) + 20) + 99)) + 17) + 15)`
	output	54648, **31415**,

✅ 1.1.D, E

Program powinien wygenerować na pierwszej pozycji na wyjściu liczbę 1, 789. Poza liczbą może też zwrócić inne liczby.

Parametry funkcji fitnessu:

```
restrictOutputLength    = false;
restrictOutputPosition = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 0;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

W przeciwieństwie do poprzednich przykładów tutaj funkcja brała pod uwagę tylko pierwszą wartość z wyjścia, z powodu flagi `restrictOutputPosition` ustawionej na `true`.

Rezultaty:

Wymagania programu nie zmieniły się bardzo znacząco, dlatego równie szybko bo już po odpowiednio 1 i 3 generacjach, otrzymaliśmy programy spełniające zadane wymagania:

D	program	`print(!False)` `scan(X7)`
	output	`1,`
E	program	`scan(X7)` `print(((70 + ((70 + 53) + (((70 + (70 + 70)) + 70) + 53))) + 70) + ((70 + 53) + 70))`
	output	`789,`

✅ 1.1.F

Program powinien wygenerować na wyjściu liczbę jako jedyną liczbę 1. Poza liczbą 1 NIE powinien nic więcej wygenerować.

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e1;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

Ustawiając flagę `restrictOutputLength` na `true` karamy program za złą długość wyjścia dla każdej wypisanej dodatkowo lub niedopisanej wartości na wektor wyjścia.

Niestety nasze początkowe ustawienie kary ... na wartość ... okazało się błędem, ponieważ programy uczyły się nie wypisywać nic, aby w ten sposób otrzymać mniejszą karę. Dlatego początkowe parametry funkcji fitnessu zostały zmodyfikowane następująco:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e9;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

W ten sposób program otrzymuje znacznie większą karę za niezgodność w długości wektora wyjścia, dlatego niemożliwe jest aby najlepszy program wypisywał coś więcej.

Rezultaty:

Funkcję spełniającą podane wymagania uzyskaliśmy już w 1 generacji.

F	program	`print(!False)` `X2 = 17 / 79`
	output	`1,`

✅ 1.2.A

Program powinien odczytać dwie pierwsze liczy z wejścia i zwrócić na wyjściu (jedynie) ich sumę. Na wejściu mogą być tylko całkowite liczby dodatnie w zakresie [0,9].

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = true;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e3;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 1e6;
_usedInputPunishment    = 0;
```

Dla tego przypadku po raz pierwszy wprowadziliśmy flagę `checkIfReadInput`, która powoduje sprawdzenie w trakcie działania interpretera czy wszystkie wartości z wektora wejścia zostały zczytane, jeśli nie to za każdą niezczytaną wartość program jest karany o wartość `_inputNotReadPunishment`.

Fitness Cases:

Jako wartości kontrolne do uczenia funkcji podaliśmy 20 przypadków w postaci losowo wygenerowanych dwóch wartości na wektorze wejścia, a wektor wyjścia zawiera oczywiście ich sumę.

Rezultaty:

Niestety w bardziej zaawansowanych przypadkach rezultatów nie udało się uzyskać z każdym uruchomieniem systemu GP. Po kilku próbach i drobnych zmianach w wartościach kar, otrzymaliśmy funkcję spełniającą zadane warunki w 37 generacji:

```
scan(X7)
scan(X6)
print(X6 + (X7 - X8))
```

Dlatego, że jeszcze nie zostałą zimplementowana funkcja kary dla urzycia niezainicjowanych zmiennych, w wywołaniu funkcji `print` pojawiła się zmienna `X8`, która przyjmuje wartość 0.

Dla przykładowego wejścia: `3`, `4` - funkcja zwróci jego sumę: `7`.

✅ 1.2.B, C

*Program powinien odczytać dwie pierwsze liczy z wejścia i zwrócić na wyjściu (jedynie) ich sumę. Na wejściu mogą być tylko całkowite liczby w zakresie **B**-[-9,9], **C**-[-9999,9999].*

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = true;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e3;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 1e6;
_usedInputPunishment    = 0;
```

Parametry są identyczne jak w funkcji 1.2.A.

Fitness Cases:

Na początku stosowaliśmy wartości wygenerowane równomiernia na rozkładzie liniowym, jednak lepsze i szybsze rezultaty dawały wartości wygenerowane zupełnie losowo.

Rezultaty:

Wymagania programu nie zmieniły się bardzo znacząco, dlatego równie szybko, bo już po odpowiednio 9 i 52 generacjach, otrzymaliśmy programy spełniające zadane wymagania:

<table>
<tr><td rowspan="2">B</td><td>program</td><td colspan="3"><pre>scan(X2)
scan(X6)
scan(X0)
if (True) {
  print(X6 + X2)
}</pre></td></tr>
<tr><td>input</td><td><pre>-5, 2,</pre></td><td>output</td><td><pre>-3,</pre></td></tr>
<tr><td rowspan="2">C</td><td>program</td><td colspan="3"><pre>scan(X6)
scan(X9)
print(X6 + X9)</pre></td></tr>
<tr><td>input</td><td><pre>543, -123,</pre></td><td>output</td><td><pre>420,</pre></td></tr>
</table>

✅ 1.2.D, E

*Program powinien odczytać dwie pierwsze liczy z wejścia i zwrócić na wyjściu (jedynie) ich **D**-różnicę, **E**-iloczyn. Na wejściu mogą być tylko całkowite liczby w zakresie [-9999,9999].*

Te przypadki również nie wprowadzają dużych zmian w działaniu funkcji, jedynie zamiast dodawania chcemy otrzymać operację odejmowania lub mnożenia.

Rezultaty:

Dla przypadku **D** zajęło to 88 generacji, dla **E**-37:

<table>
<tr><td rowspan="2">B</td><td>program</td><td colspan="3"><pre>scan(X0)
scan(X4)
scan(X0)
print(X0 - X4)</pre></td></tr>
<tr><td>input</td><td><pre>543, -123,</pre></td><td>output</td><td><pre>666,</pre></td></tr>
<tr><td rowspan="2">C</td><td>program</td><td colspan="3"><pre>scan(X1)
scan(X1)
scan(X4)
print(X4 * (X1 / X6))
scan(L1)</pre></td></tr>
<tr><td>input</td><td><pre>543, -123,</pre></td><td>output</td><td><pre>-65682,</pre></td></tr>
</table>

❌ 1.3.A, B

*Program powinien odczytać dwie pierwsze liczby z wejścia i zwrócić na wyjściu (jedynie) większą z nich. Na wejściu mogą być tylko całkowite liczby dodatnie w zakresie **A**-[0,9], **B**-[-9999, 9999].*

Fitness Cases:

Dla zadania A przygotowaliśmy 100 przypadków uczących, gdzie na wejściu są różne kombinacje liczb całkowitych z zakresu [0, 9] z dokładnością do kolejności. Na wyjściu była większa liczba spośród tych podanych na wejściu. Analogicznie stworzyliśmy 960 przykładów dla zadania B.

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e1;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

Po uruchomieniu programu z powyższymi parametrami funkcji fitnessu problem nie został rozwiązany. Po 100 generacjach otrzymaliśmy następujące rezultaty:

Best fitness: 1650

Best individual:

```
scan(X4)
print(X4)
```

Jak widać nasz program nie wczytywał odpowiedniej ilości zmiennych. Z tego powodu zmieniliśmy parametry na:

```
restrictOutputLength    = true;
restrictOutputPosition = true;
checkIfReadInput        = true;
checkUsedInput          = true;
```

```
_outputLenghtPunishment = 1e1;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 1e1;
_usedInputPunishment    = 1e1;
```

W wyniku tego nasze generowane programy wczytywały co najmniej dwie zmienne i wypisywały co najmniej jedną z nich:

```
scan(X2)
scan(X9)
print(X9)
```

Dalsze próby poprawienia wyników nie spowodowały rozwiązania problemu. Próbowaliśmy modyfikować poszczególne kary do wartości 1e3 w różnych kombinacjach. Modyfikacja szansy na crossover również nie pomogła. Często nasze programy dla zadania A produkowały program, który wypisywał liczbę siedem czy programy posiadające głęboko zagnieżdżone warunki if z rozbudowanymi warunkami jak na przykład:

```
scan(X2)
if (X2 <= 7) {
 if (X2 <= X2) {
   if (X2 <= 7) {
     if (X2 <= 7) {
       scan(X2)
     }
   }
 }
}
print(X2)
```

Powyższy program osiągnął fitness wynoszący 850 co było najlepszym osiągniętym przez nas wynikiem. Co ciekawe zostało to osiągnięte po ponownym ustawieniu parametrów na checkIfReadInput oraz checkUsedInput na wartość false.

Rezultaty:

F	program	`scan(X2)`
		`if (X2 <= 7) {`
		`  if (X2 <= X2) {`
		`  if (X2 <= 7) {`
		`  if (X2 <= 7) {`
		`  scan(X2)`
		`  }`
		`  }`
		`  }`
		`}`
		`print(X2)`

	fitness	**850**

❌ 1.4.A

Program powinien odczytać dziesięć pięć pierwszych liczy z wejścia i zwrócić na wyjściu (jedynie) ich średnią arytmetyczną (zaokrągloną do pełnej liczby całkowitej). Na wejściu mogą być tylko całkowite liczby w zakresie [-99,99].

Fitness Cases:

Ograniczyliśmy długość wektora wejścia do potrzebnych 5 wartości, aby nieco ułatwić działanie programu. Wartości były generowane losowo w zakresie [-99,99], a pożądane wyjście zdefiniowane jako średnia liczb z wektora wejścia, (bez zaokrąglenia).

Parametry funkcji fitnessu:

```
restrictOutputLength    = false;
restrictOutputPosition = true;
checkIfReadInput        = true;
checkUsedInput          = true;
```

```
_outputLenghtPunishment = 1e3;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 1e1;
_usedInputPunishment    = 1e0;
```

Rezultaty:

Niestety mimo uproszczenia zadania do długości wejścia - 5, po przejściu 100 generacji nie uzyskaliśmy pożądanego programu. Najlepszym rezultatem jaki udało nam się znaleźć jest ten przedstawiony poniżej:

A	program	`if (True) { scan(X7) if (True) { scan(X3) scan(X5) } } print(-(X5 + X5) / -14) if (59 < 84) { scan(X3) scan(X2) } if (True) { scan(X2) if (54 < 96) { print(-(X5 + 88) / -27) if (54 < 96) { scan(X7) scan(X6) } }} if (54 < 96) { scan(X4) scan(X5) } print(True)`
	fitness	`3730`

Finalne testy systemu GP

✅ Suma int + float

Dla liczby całkowitej i liczby zmiennoprzecinkowej (int i float) program ma zwrócić ich sumę.

Parametry funkcji fitnessu:

```
restrictOutputLength    = false;
restrictOutputPosition = true;
checkIfReadInput        = true;
checkUsedInput          = true;
```

```
_outputLenghtPunishment = 1e3;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 1e6;
_usedInputPunishment    = 1e0;
```

Modyfikując funkcę dostosowania aby nie karała za wypisanie większej ilości wartości na wyjściu uzyskaliśmy bardziej regularne wyniki między uruchomieniami systemu (program nadal jest karany jeżeli wypisze za mało wartości).

Rezultaty:

Zadanie nie różniło się praktycznie niczym od przypadków 1.2.A, B, C z podstawowych testów systemu. Funkcję uzyskaliśmy już w 7 generacji:

```
scan(X2)
scan(X5)
print(X2 + X5)
```

Dla przykładowego wejścia: `4`, `3.14` - funkcja zwróci jego sumę: `7.14`.

❌ Suma kwadratów [1,n]

Funkcja powinna zczytać jedną wartość z wejścia i na wyjściu zwrócić sumę kwadratów liczb od 1 do n (wartości z wejścia).

Fitness Cases:

Jako wartości uczące przyjęliśmy 30 wartości od 0 do 29 (przeciwdziedzina: [0,8555]).

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = true;
checkUsedInput          = true;
```

```
_outputLenghtPunishment = 1e6;
_valuePunishment        = 1e0;
_inputNotReadPunishment = 1e6;
_usedInputPunishment    = 1e0;
```

Dużo eksperymentowaliśmy z wartościami kar... dla tego zbioru otrzymywaliśmy najbardziej stałe rezultaty.

Rezultaty:

Nie osiągnęliśmy sukcesu dla tego problemu. Oto najlepszy rezultat otrzymany po 100 generacjach:

<table>
<tr><td rowspan="3">A</td><td>program</td><td colspan="5"><pre>scan(X9)
print((((-((-(((-((-(-(28 * (69 - 21)) - (56 * -X9)) - -69) + 46) - -68) + (28 * (68 - 21))) - (69 * -X9)) - -69) + 46) - -X9) - -X9) - -X9) + (-(28 * (69 - 21)) - ((68 + 46) * -X9)))
scan(X9)
scan(X9)</pre></td></tr>
<tr><td>fitness</td><td colspan="5">25225</td></tr>
<tr><td>input</td><td>9,</td><td>output</td><td>644,</td><td>pożądany output</td><td>285,</td></tr>
</table>

❌ Mediana

Program powinien wczytać 3 liczby całkowite z wektora wejścia i wypisać ich medianę

Fitness Cases:

W każdym przypadku treningowym były losowane 3 liczby z zakresu [-20, 20] a następnie na wyjście przypisywano ich medianę. Stworzyliśmy 200 takich fitness caseów.

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e1;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

Po 100 generacjach z powyższymi parametrami nie udało się znaleźć rozwiązania. Programy wykonywały zazwyczaj jedną instrukcję którą było wczytanie pojedynczej liczby. W celu zapewnienia wczytania wszystkich 3 liczb zmieniliśmy wartość parametrów checkIfReadInput oraz checkUsedInput na true oraz powiązane z nimi zmienne na 1e1. Otrzymaliśmy w ten sposób poniższe rezultaty.

Best fitness: 6020

Best individual:

```
scan(X7)
scan(X1)
scan(X4)
```

Generowane najlepsze programy wczytywały za każdym razem 3 liczby, jednak żadnej nie wypisywały. Mogło to być spowodowane faktem, że program wolał nie wypisywać nic, niż wypisać wartość złej zmiennej i zostać za to ukaranym. Z tego powodu zmieniliśmy wartość parametrów na poniższe:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = true;
checkUsedInput          = true;
```

```
_outputLenghtPunishment = 1e3;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 1e3;
_usedInputPunishment    = 1e3;
```

Podwyższyliśmy wszystkie kary poza value aby system “nie bał się” próbować wypisywać jakiekolwiek wyniki. W wyniku modyfikacji była wypisywana jedna z wczytanych zmiennych, jednakże dalsze modyfikacje parametrów nie pozwoliły poprawić wyników.

Rezultaty:

F	program	`scan(X5)`
		`scan(X0)`
		`scan(X9)`
		`print(X9)`

	fitness	`10960`

❌ Boolean k=1

Program powinien być w stanie wygenerować funkcję boolowską dla k=1.

Fitness Cases:

Dla k=1 spróbowaliśmy odtworzyć funkcję NOT.

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e1;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

Z początku ustawiliśmy powyższe parametry, jednak zawsze powstawał program wypisujący prawdę lub fałsz. Modyfikację parametrów nie pomogły uzyskać innych efektów.

Rezultaty:

F	program	`print(False)`
	fitness	`10,`

✅ Boolean k=2

Program powinien być w stanie wygenerować funkcję boolowską dla k=2.

Fitness Case's:

Dla k=1 spróbowaliśmy odtworzyć funkcję XOR.

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e1;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

Nasz system był stanie wygenerować wynik dla powyższych parametrów za pierwszym razem. Końcowy program został wygenerowany w 76 generacji.

Rezultaty:

F	program	`scan(X8)`
		`scan(X9)`
		`scan(X1)`
		`print(X8 != X9)`
		`scan(X2)`

	fitness	`0,`

❌ Boolean k=3

Program powinien być w stanie wygenerować funkcję boolowską dla k=3.

Fitness Case's:

Dla k=3 spróbowaliśmy odtworzyć funkcję poniższą funkcję:

```
0 0 0 : 0
0 0 1 : 1
0 1 0 : 1
0 1 1 : 0
1 0 0 : 1
1 0 1 : 0
1 1 0 : 1
1 1 1 : 0
```

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e1;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

Ponownie zaczęliśmy z powyższymi parametrami, lecz znowu najlepsze programy polegały na wygenerowaniu wartość prawdy lub fałszu. Z tego powodu ustawiliśmy parametr checkIfReadInput na true oraz związaną z tą flagą karę na 1e1. Poskutkowało to osiągnięciem fitnessu o wartości 20, gdzie wcześniej najlepszym wynikiem było 40.

Rezultaty:

F	program	`if (!((63 * 41) <= (-77 * (X1 / (99 / X1))))) {`
		`  if (True) {`
		`  if (True) {`
		`  scan(X1)`
		`  }`
		`  }`
		`}`
		`scan(X4)`
		`scan(X7)`
		`if (True) {`
		`  if (True) {`
		`  print(X7 < X4)`
		`  }}`

	fitness	`20,`

❌ Boolean k=4

Program powinien być w stanie wygenerować funkcję boolowską dla k=4.

Fitness Case's:

Dla k=4 spróbowaliśmy odtworzyć funkcję poniższą funkcję:

```
0 0 0 0 : 0        0 0 0 1 : 1        0 0 1 0 : 0        0 0 1 1 : 0
0 1 0 0 : 1        0 1 0 1 : 0        0 1 1 0 : 1        0 1 1 1 : 1
1 0 0 0 : 0        1 0 0 1 : 0        1 0 1 0 : 1        1 0 1 1 : 0
1 1 0 0 : 1        1 1 0 1 : 0        1 1 1 0 : 1        1 1 1 1 : 0
```

Parametry funkcji fitnessu:

```
restrictOutputLength    = true;
restrictOutputPosition  = true;
checkIfReadInput        = false;
checkUsedInput          = false;
```

```
_outputLenghtPunishment = 1e1;
_valuePunishment        = 1e1;
_inputNotReadPunishment = 0;
_usedInputPunishment    = 0;
```

Powyższe parametry nie dawały dobrych rezultatów. Powtarzał się problem z wpisywaniem wartości fałsz bez warunków. Kolejne modyfikację parametrów takich jak checkIfReadInput, checkUsedInput czy crossover_chance nie pozwoliło uzyskać lepszych efektów.

Rezultaty:

F	program	`scan(X1)`
		`print(False)`
		`scan(X3)`
		`scan(X8)`
		`scan(X4)`

	fitness	`70,`