Genetic Programming

Dr. Sabah Sayed

Department of Computer Science

Faculty of Computers and Artificial Intelligence

Cairo University

Egypt

THE CHALLENGE

"How can computers learn to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it?"

— Attributed to Arthur Samuel (1959)

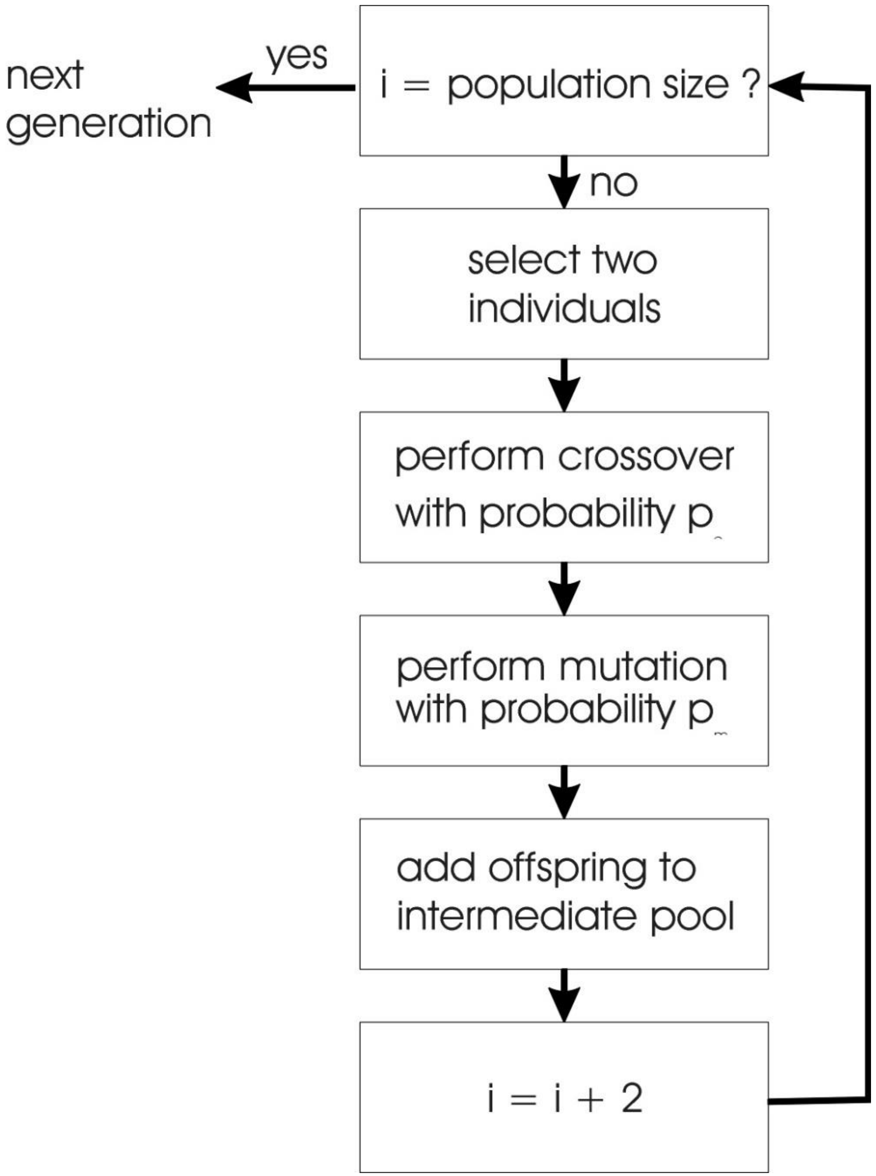
GENETIC PROGRAMMING (GP)

- One of the central problems in computer science is how to make computers solve problems without being explicitly programmed to do so.
- Genetic programming offers a solution through the **evolution of computer programs** by methods of **natural selection**. It is an extension of the **Genetic Algorithm**.
- Genetic programming is a recent development in the area of evolutionary computation. It was greatly stimulated in the 1990s by **John Koza**.
- According to Koza, genetic programming **searches the space of possible computer programs** for a **program** that is highly fit for solving the problem at hand.

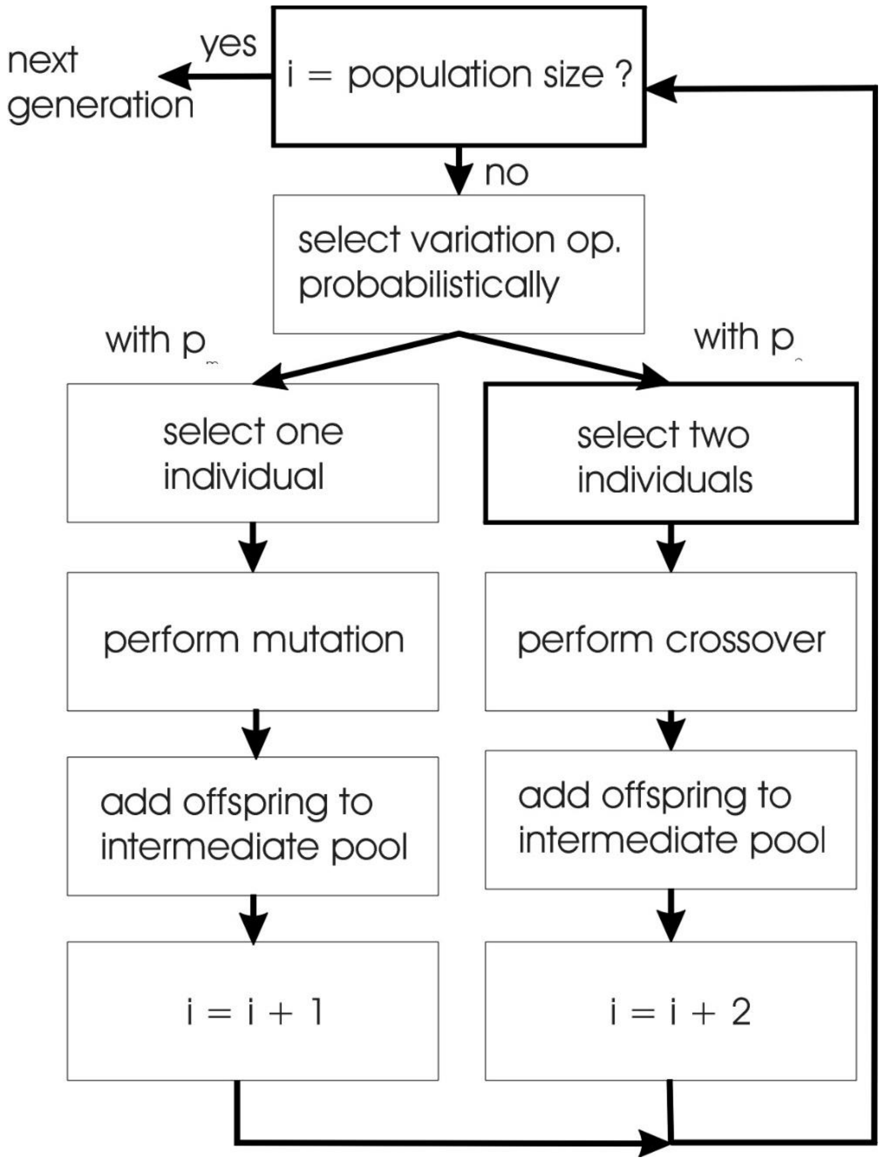
Why Genetic Programming?

- "For many problems in machine learning and artificial intelligence, the most natural representation for a solution is a computer program." [Koza, 1994]

- A **parse tree** is a good representation of a computer program for Genetic Programming



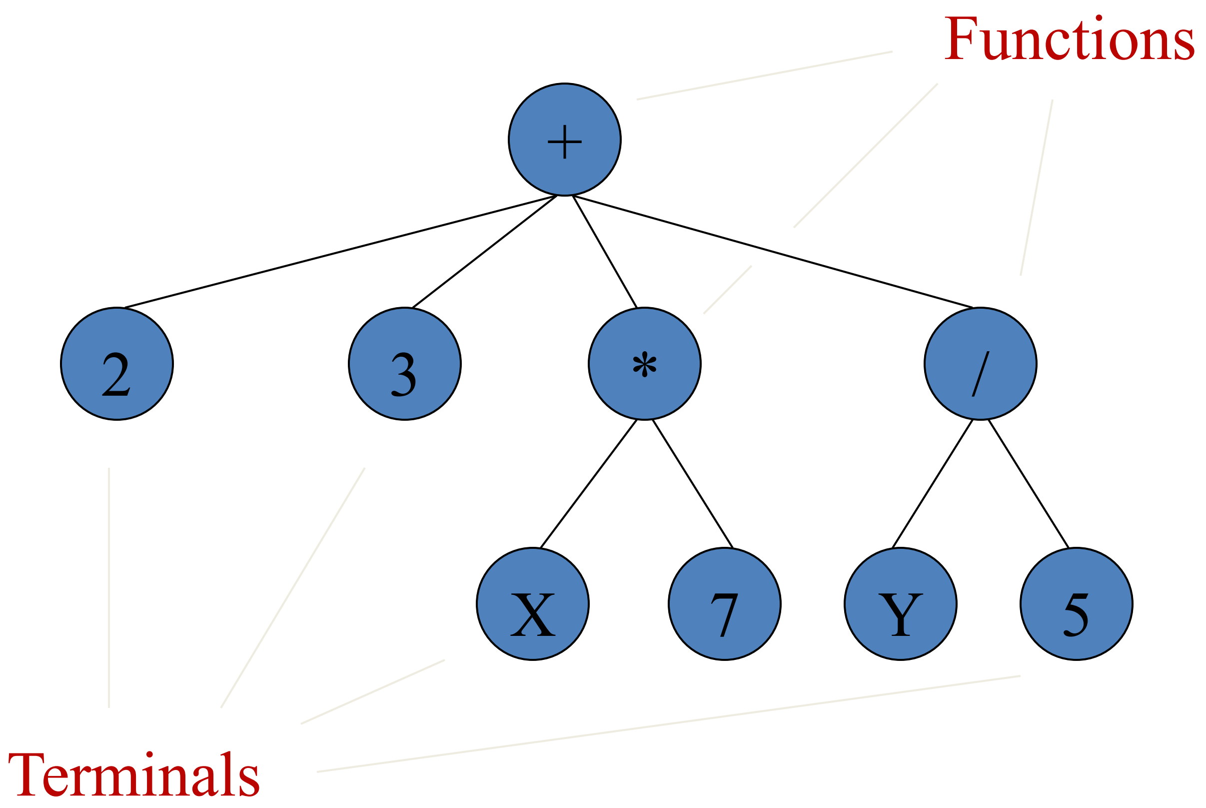
GA flowchart



GP flowchart

Using ***Trees*** To Represent Computer Programs

(+ 2 3 (* X 7) (/ Y 5))



GP Technical Summary

Representation	Tree structures
Recombination	Exchange of subtrees
Mutation	Random change in trees
Parent selection	Fitness proportional
Survivor selection	Generational replacement

Genetic Operations

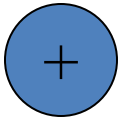
- Random Generation of the **initial population** of possible solutions
- **Mutation** of promising solutions to create new possible solutions
- Genetic **Crossover** of two promising solutions to create new possible solutions

Randomly Generating Programs

- Randomly generate a program that takes two arguments and uses basic arithmetic to return an answer
  - Function set = {+, -, *, /}
  - Terminal set = {integers, X, Y}
- Randomly select either a function or a terminal to represent our program
- If a function was selected, recursively generate random programs to act as arguments

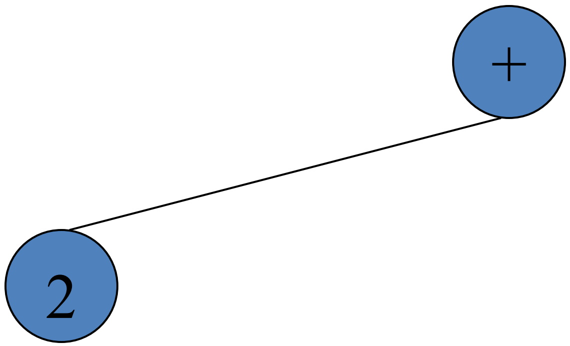
Randomly Generating Programs

(+ . . .)



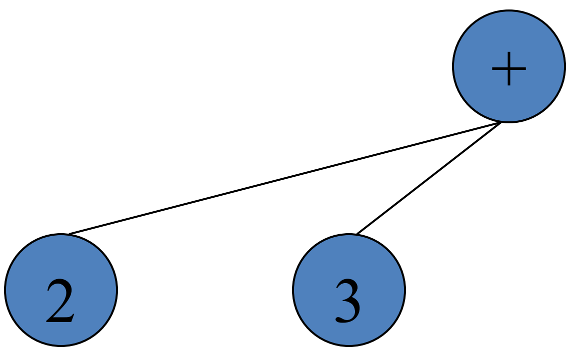
Randomly Generating Programs

(+ 2 …)



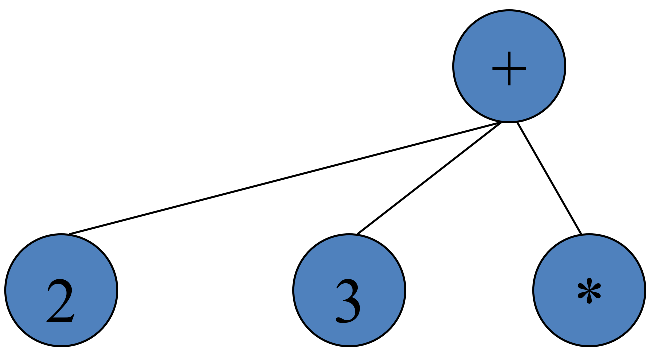
Randomly Generating Programs

(+ 2 3 …)



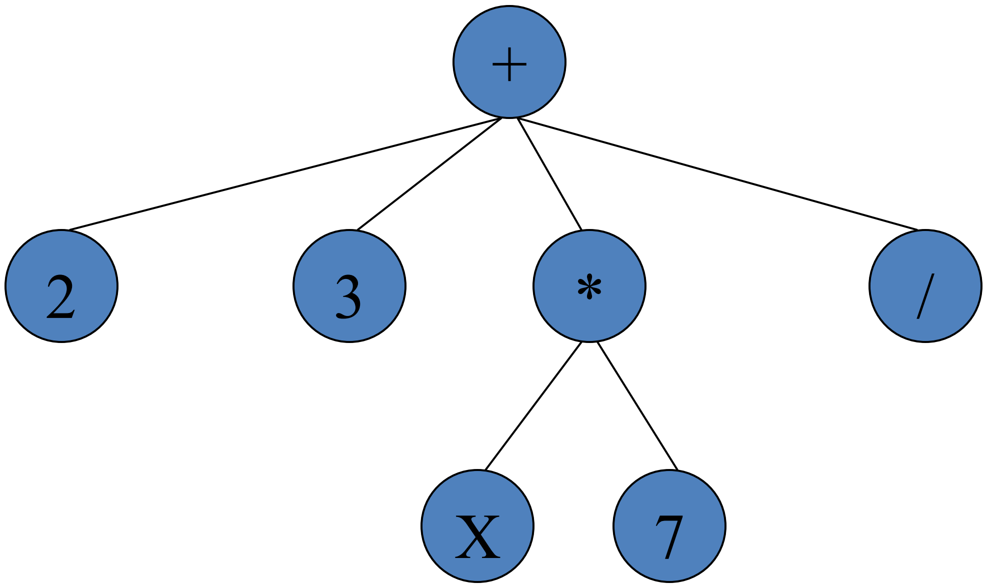
Randomly Generating Programs

(+ 2 3 (* …) …)



Randomly Generating Programs

(+ 2 3 (* X 7) (/ …))



Tree based representation

- Trees are a universal form, e.g. consider:
- Arithmetic formula

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

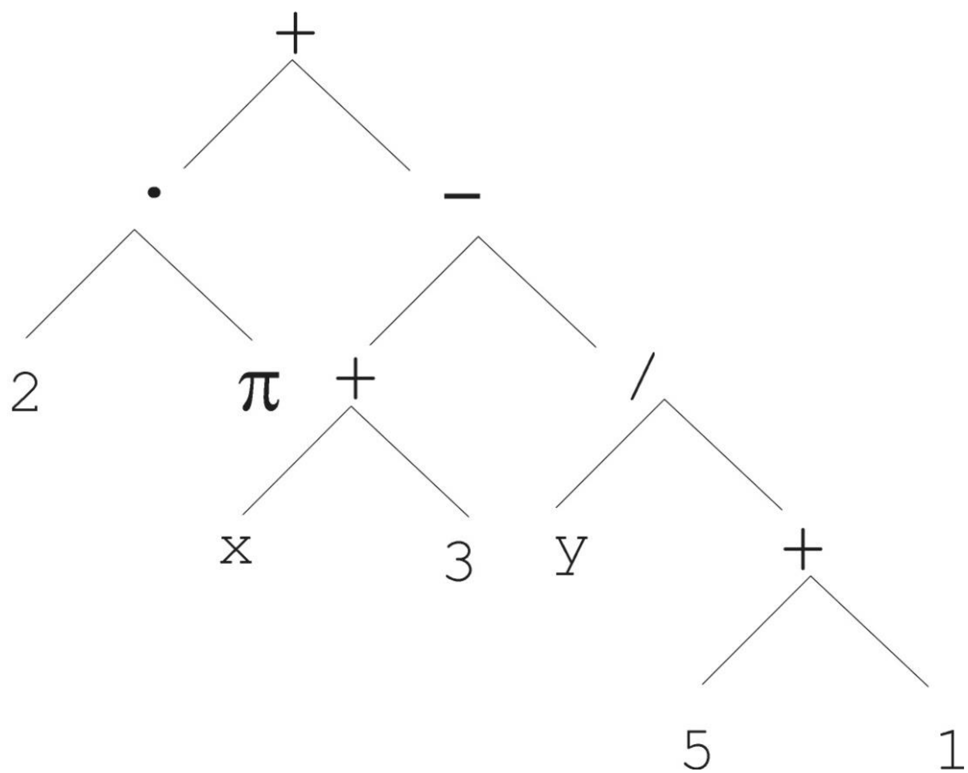
- Logical formula

$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

- Program

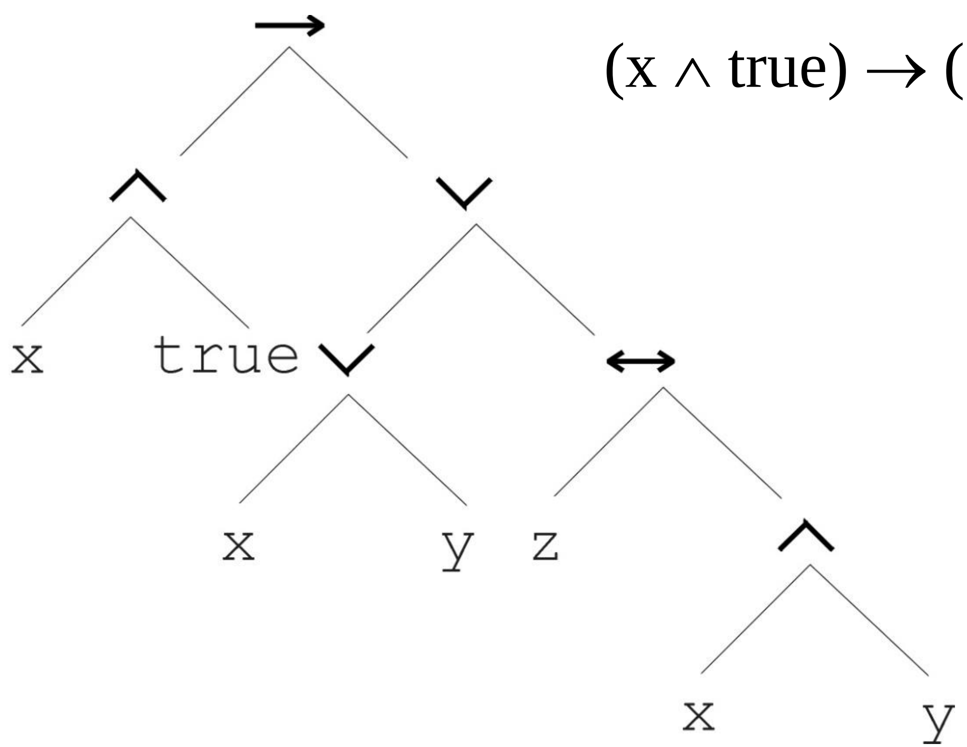
```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation



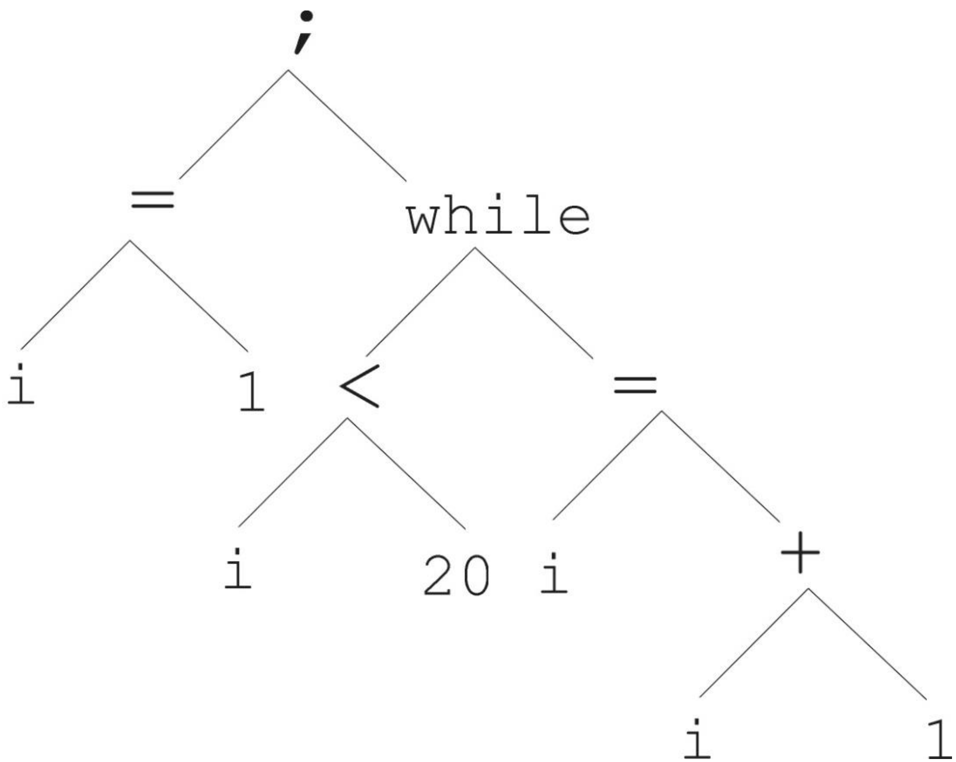
$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

Tree based representation



$$(x \wedge \text{true}) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$

Tree based representation



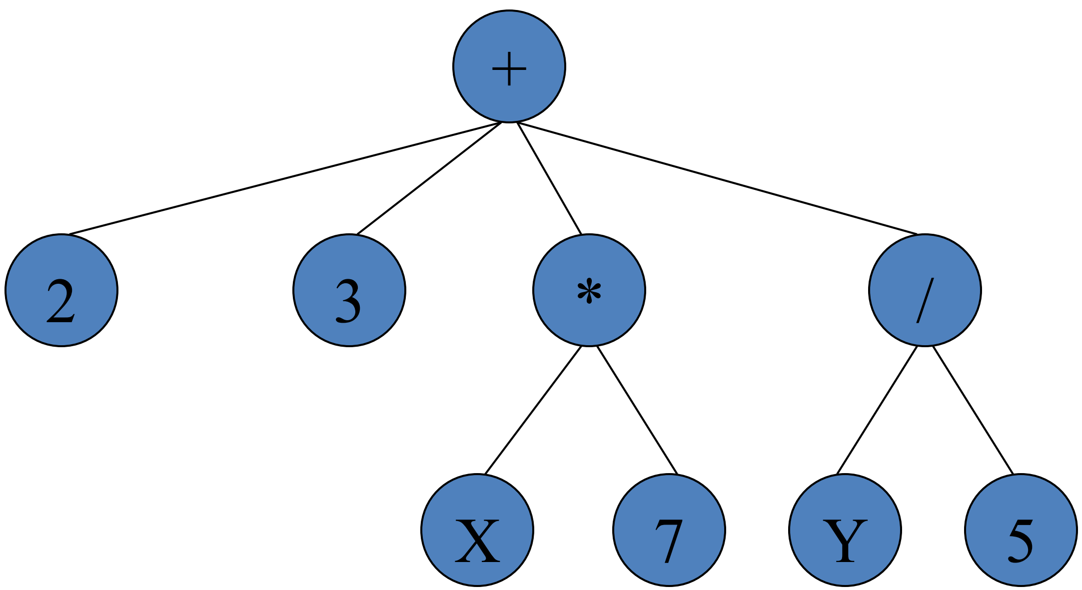

```
i =1;
while (i < 20)
{
        i = i +1
}
```

Tree based representation

- In GA, chromosomes are *linear* structures (bit strings, integer string, permutations)
- Tree shaped chromosomes are *non-linear* structures
- In GA, the size of the chromosomes is *fixed*
- Trees in GP may *vary* in depth and width

Randomly Generating Programs

(+ 2 3 (* X 7) (/ Y 5))

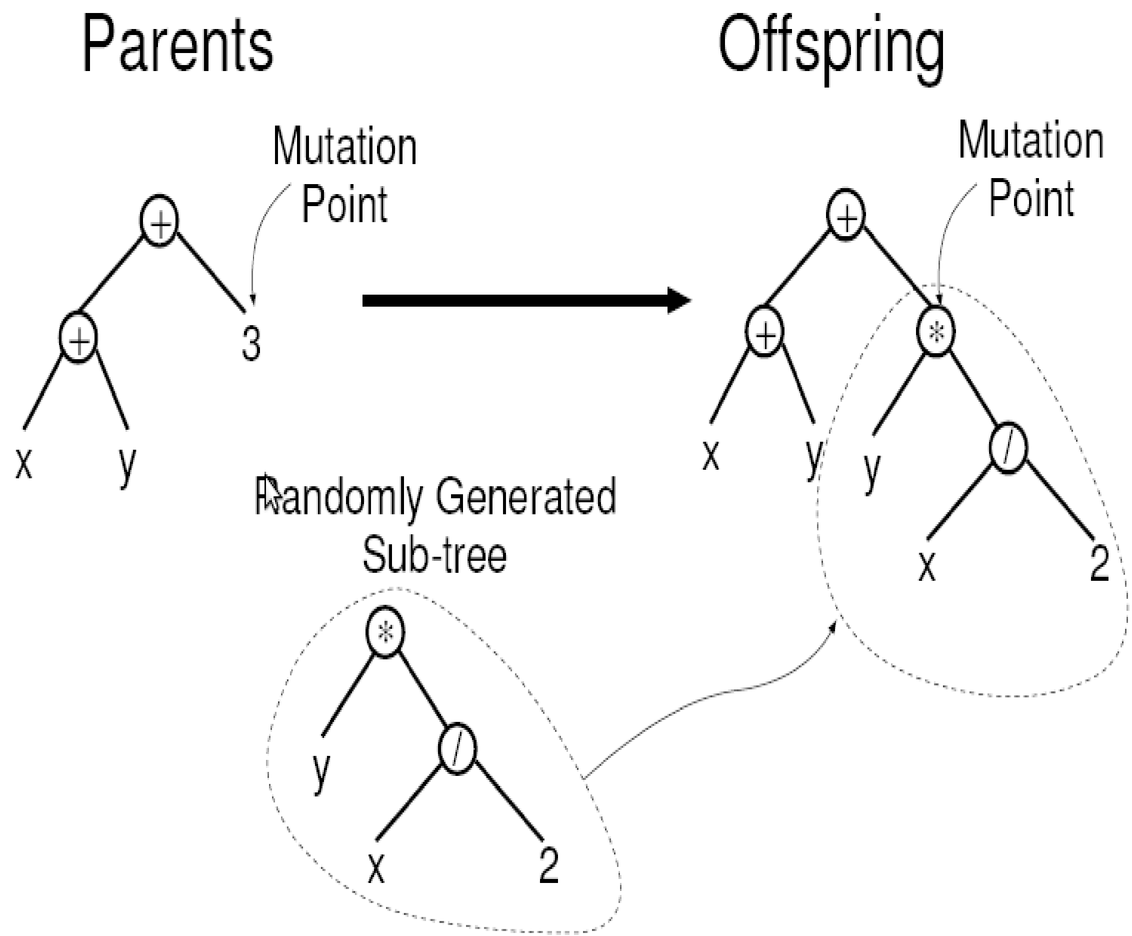


Mutation

- Mutation has two parameters:
    1. Probability p_m to choose mutation vs. recombination
    2. Probability to choose an internal point as the root of the subtree to be replaced
- Remarkably p_m is advised to be very small (Koza'92), like 0.05 (Banzhaf et al. '98)
- The size of the child can exceed the size of the parent

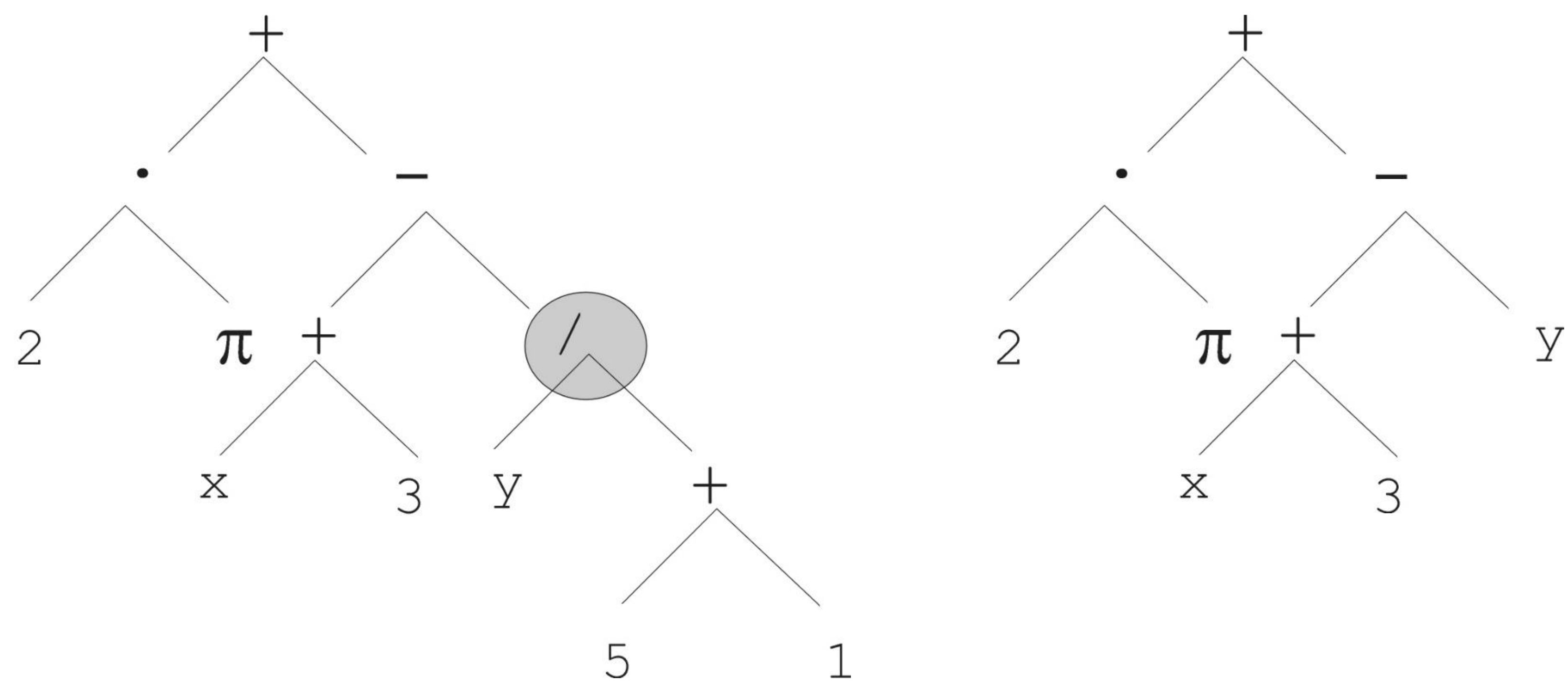
Mutation

- Subtree Mutation: Replace the mutation point by randomly generated tree.
- Point Mutation: Randomly select a node and replace it with another primitive.



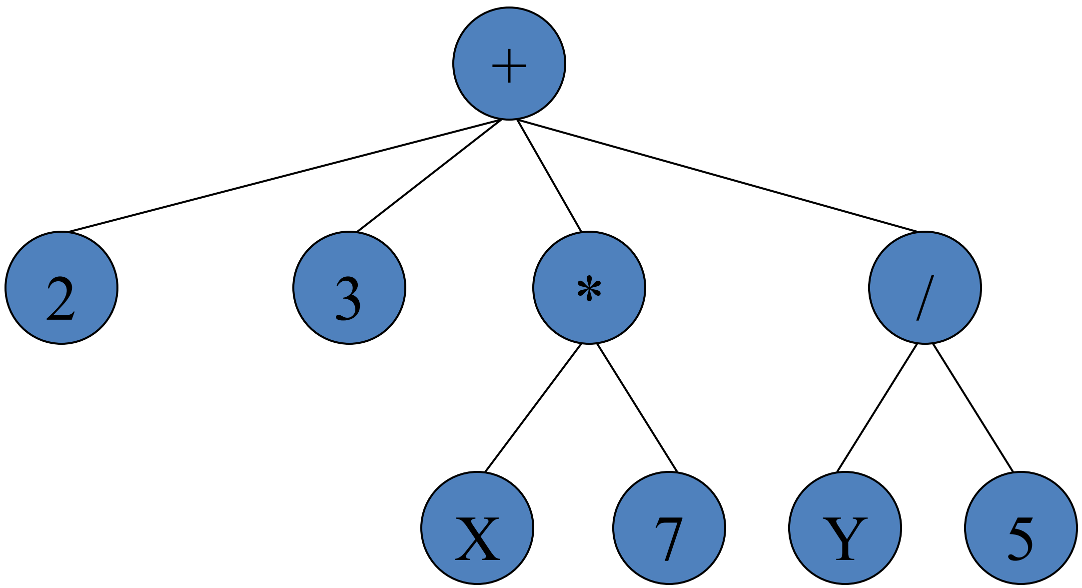
Mutation

- Most common mutation: replace randomly chosen subtree by randomly generated tree



Mutation

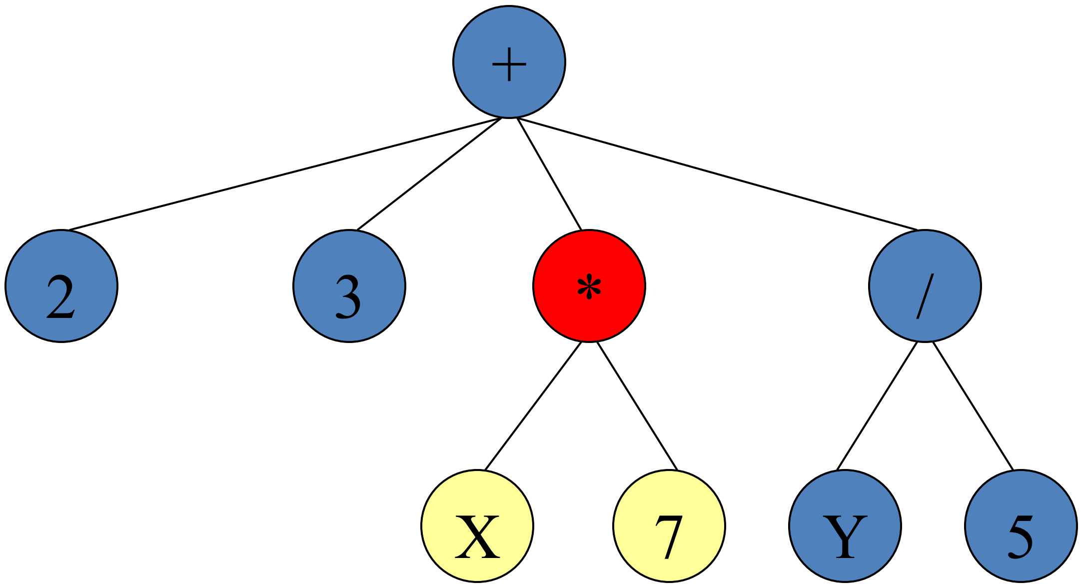
(+ 2 3 (* X 7) (/ Y 5))



Mutation

First pick a random node

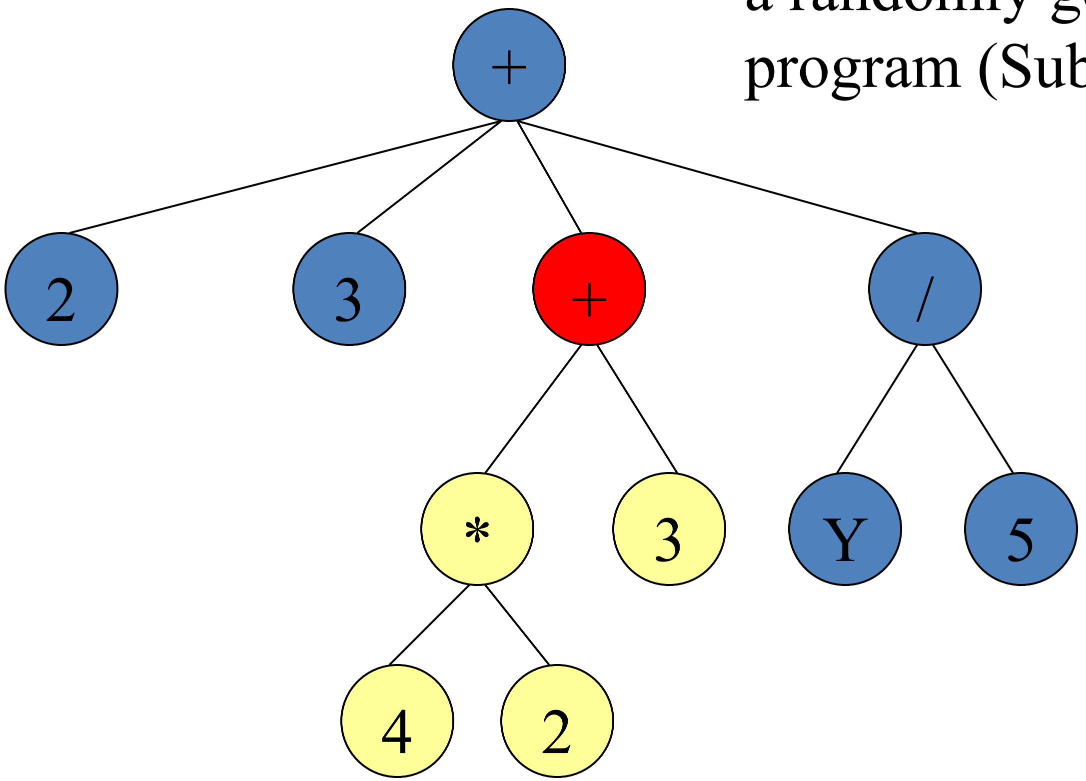
(+ 2 3 (* X 7) (/ Y 5))



Mutation

(+ 2 3 (+ (* 4 2) 3) (/ Y 5))

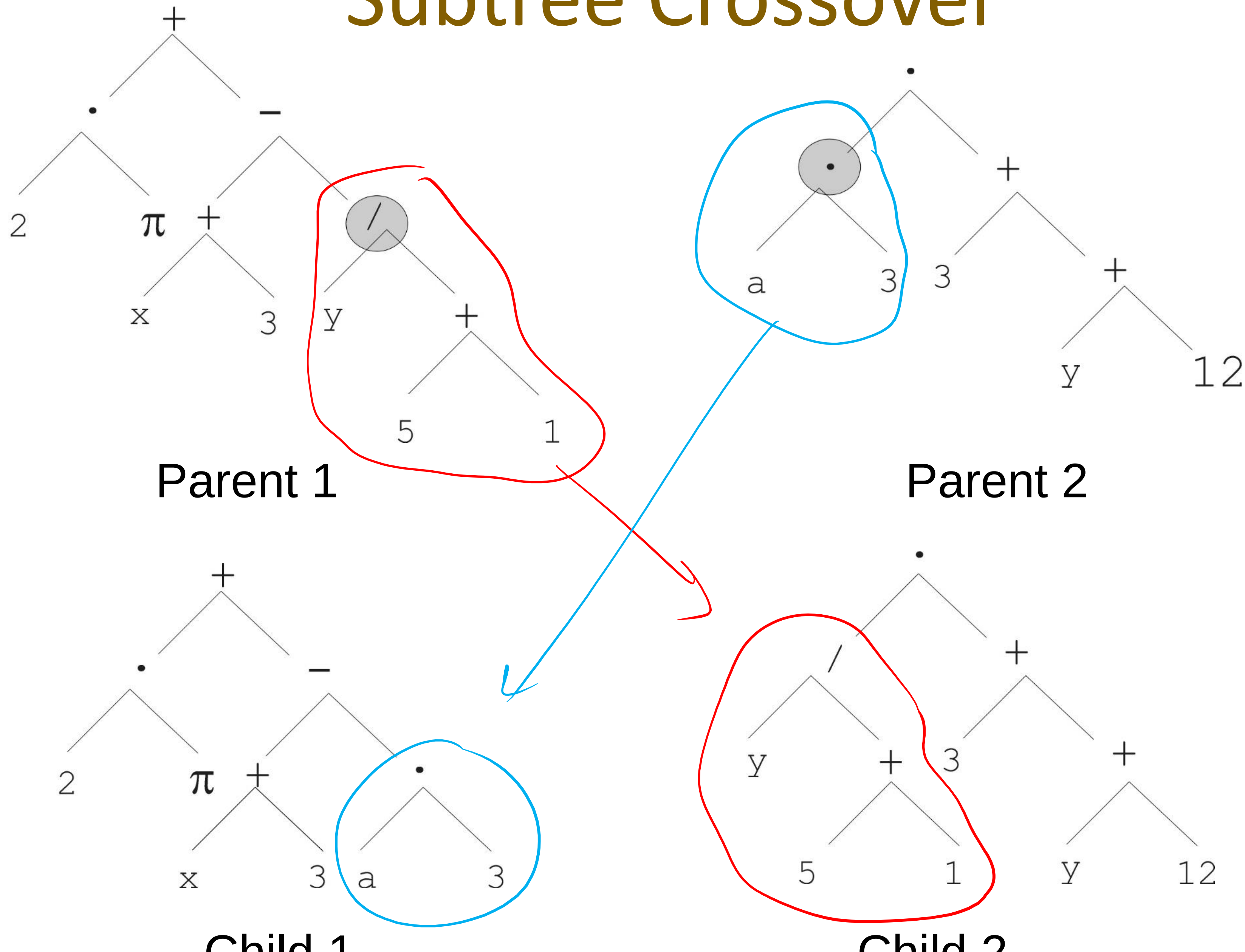
Delete the node and its children, and replace with a randomly generated program (Subtree)



Recombination = Crossover

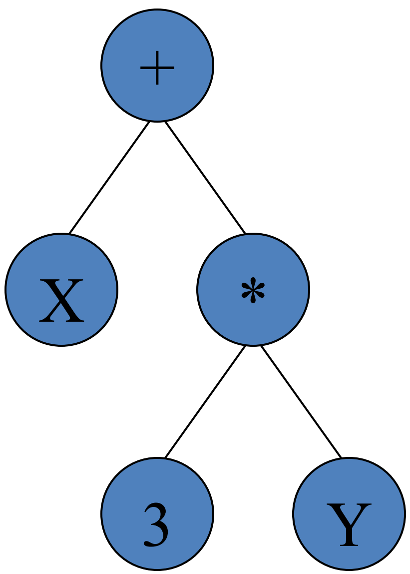
- Most common recombination: exchange two randomly chosen subtrees among the parents
- Recombination has two parameters:
    1. Probability p_c to choose recombination vs. mutation
    2. Probability to choose an internal point within each parent as crossover point
- The size of offspring can exceed that of the parents

Subtree Crossover

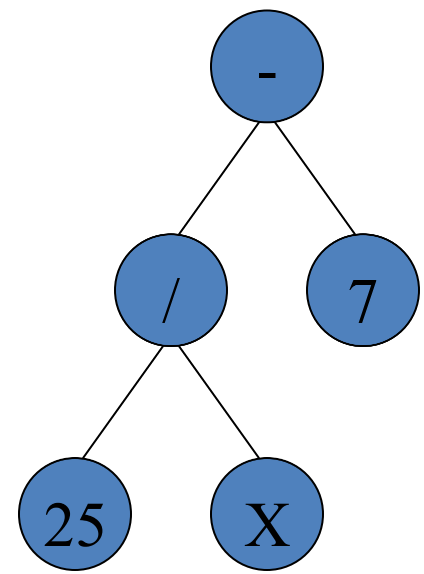


Crossover

(+ X (* 3 Y))

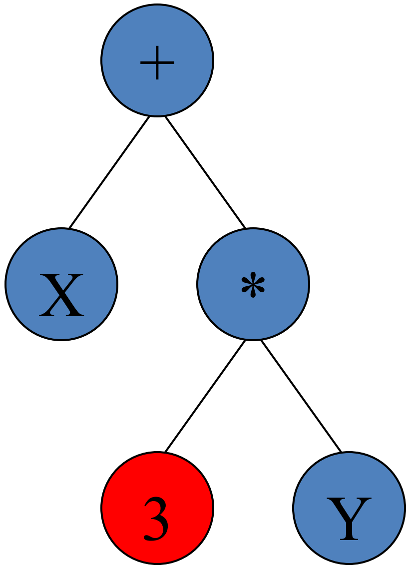


(- (/ 25 X) 7)



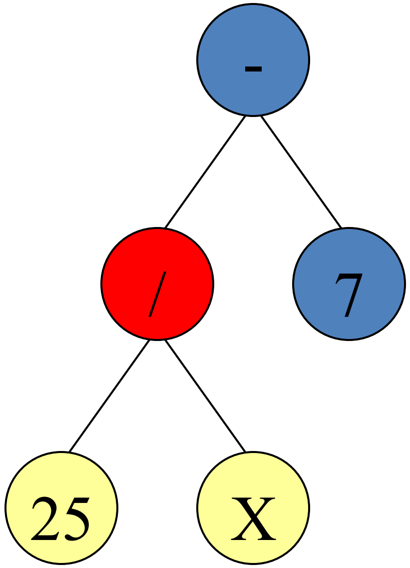
Crossover

(+ X (* 3 Y))



Pick a random node in each program

(- (/ 25 X) 7)

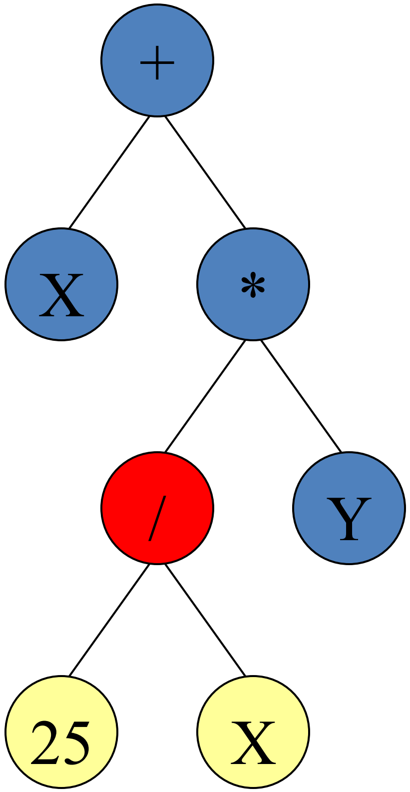


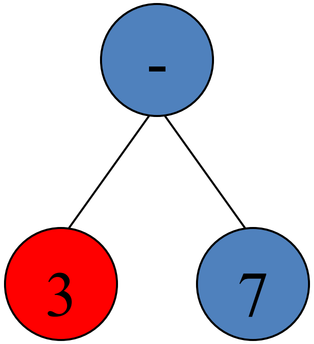
Crossover

(+ X (* (/ 25 X) Y))

(- 3 7)

Swap the two nodes





What About Just Randomly Generating Programs?

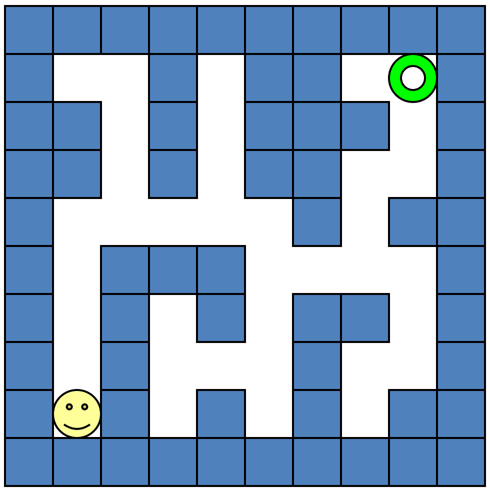
- Is Genetic Programming really better than just randomly creating new functions?
- Yes!
  - Pete Angeline compared the result of evolving a tic-tac-toe algorithm for 200 generations, with a population size of 1000 per generation, against 200,000 randomly generated algorithms
  - The best evolved program was found to be significantly superior to the best randomly generated program [Genetic Programming FAQ, 2002]
- The key lies in using a **fitness** measure to determine which functions survive to reproduce in each generation

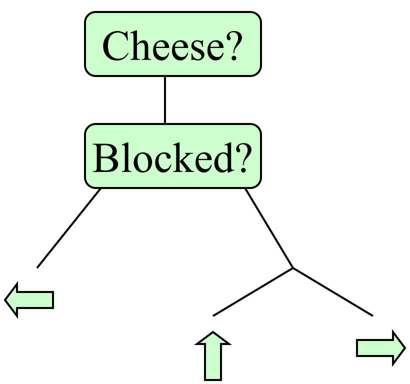
Building a Better Mouse

- Apply Genetic Programming to the problem of navigating a maze
- What are our terminal and function sets?
  - Function Set = {If-Movement-Blocked, While-Not-At-Cheese*}
  - Terminal Set = {Move-Forward, Turn-Left, Turn-Right}

* While-Not-At-Cheese will be used exclusively as the root node of the parse tree

Building a Better Mouse





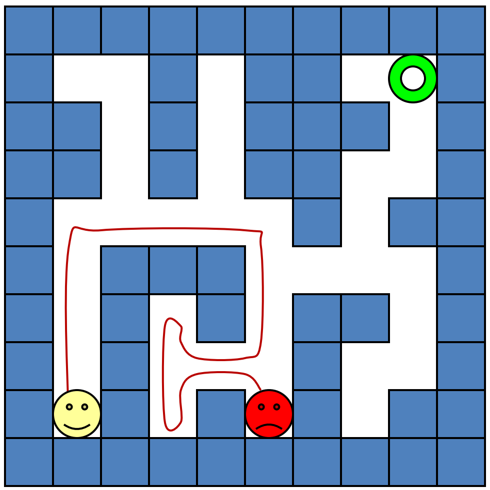
How to get the starving mouse to the cheese?

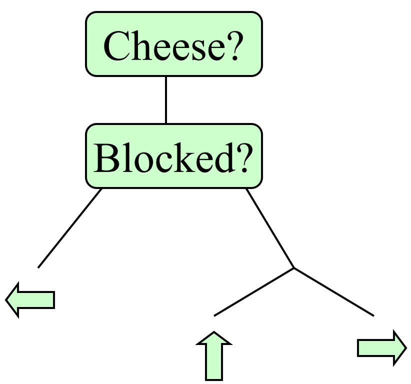
One possible solution:

```
While not at the cheese
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Move forward
    Turn right 90 degrees
```

Is there a better solution for this maze?
How good is this solution?

Building a Better Mouse



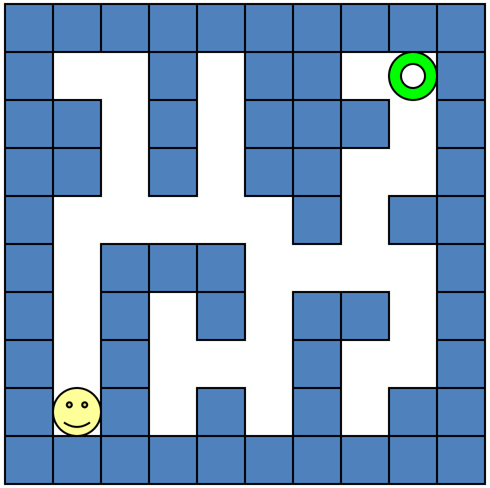


A fitness function:

- Each function and terminal other than the root node shall cost one unit to execute
- If the mouse spends more than 100 units, it dies of hunger
- The fitness measure for a program is determined by executing the program, then squaring the **sum** of the **total units spent** and **the final distance from the exit**
- A **lower fitness** measure is preferable to a higher fitness measure

Our mouse will die 10 moves from the exit after spending 100 units, so the fitness measure for our program is 12100

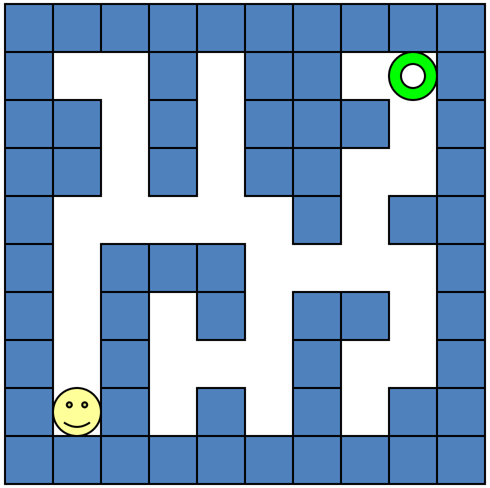
Building a Better Mouse



```
While not at the cheese
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Move forward one space

While not at the cheese
  Move forward one space
  Turn right 90 degrees
  Turn left 90 degrees
```

Building a Better Mouse



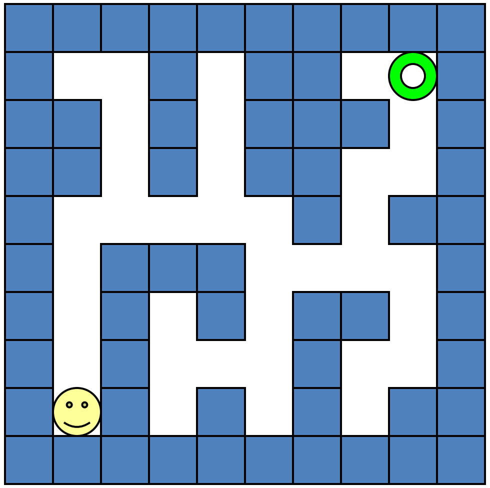
```
While not at the cheese
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Move forward one space

While not at the cheese
  Move forward one space
  Turn right 90 degrees
  Turn left 90 degrees
```

Mutation:

```
While not at the cheese
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Turn left 90 degrees
```

Building a Better Mouse



```
While not at the cheese
  If the way ahead is blocked
    Turn left 90 degrees
  Otherwise
    Move forward one space
```

```
While not at the cheese
  Move forward one space
  Turn right 90 degrees
  Turn left 90 degrees
```

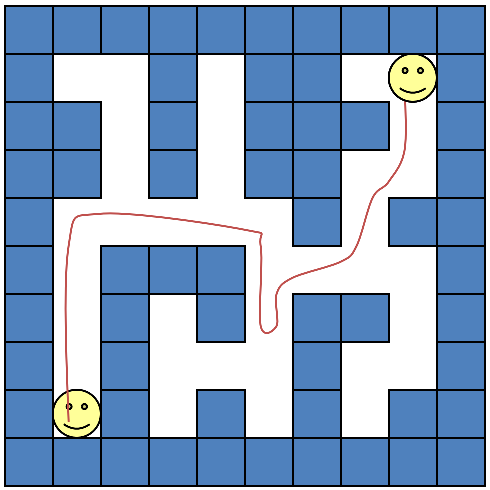
Crossover:

```
While not at the cheese
  If the way ahead is blocked
    Move forward one space
    Turn right 90 degrees
  Otherwise
    Move forward one space
```

```
While not at the cheese
  Turn left 90 degrees
  Turn left 90 degrees
```

Building a Better Mouse

(after 4202 generations, with 1000 programs per generation)



Is this better?

Fitness measure: 2809

```
While not at the cheese
  If the way ahead is blocked
    Turn right 90 degrees
    Move forward one space
    Move forward one space
    Move forward one space
  Otherwise
    Move forward one space
    Turn right 90 degrees
    Move forward one space
    Move forward one space
    Turn left 90 degrees
    If the way ahead is blocked
      Turn left 90 degrees
    Otherwise
      Move forward one space
```