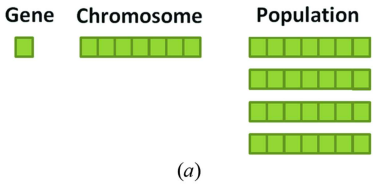
Genetic Algorithm Basics Using MIPs

October 12, 2018

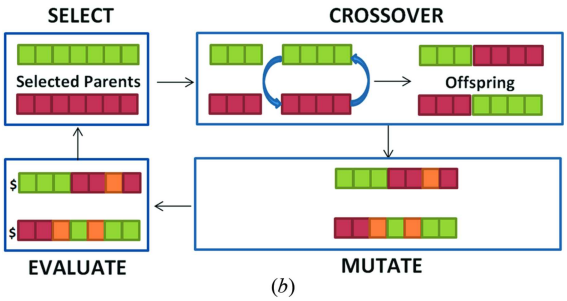
1 Genetic/Evolutionary Algorithms

Biological processes are often used as a basis for computer science algorithms. In computer science a Genetic Algorithm is a group of algorithms used to search large spaces to optimize a problem. There are a wide variety of variations in the implementations, all of which depend on the problem you're trying to solve. For this assignment, you will implement a very small element of a Genetic Algorithm, crossover, or "recombination."

Recombination is the process of constructing new chromosomes from *specific segments* of two selected parent chromosomes. The most common type of recombination involves making two new chromosomes from the sequences of two selected parents. This is done by choosing a point in the two parent sequences, copying each gene up to the point, then swapping all the remaining genes. For instance, if the chosen point splits the parents in half, then the first half of Parent A (green) is joined to the latter half of Parent B (red), while the first half of Parent B is joined to the latter half of Parent A. This creates two new offspring with the varying features of both parents. Random mutations are introduced to reduce the risk of getting stuck in a local min/max of your search space.



(*a*)



(*b*)

2 Description

In this assignment you will prompt the user to input from the keyboard two strings of characters, each of length 16, and an integer value representing the cross-over point (ranging from 0 thru 15). You will write code that performs the cross-over operation and then subsequently writes the output to the terminal. Use of at least 2 functions is required. An example of a possible code structure is shown below.

```
# Name:
# Email:
# Stu ID:

#Description:

.data

    parent1:    .space 17    #empty space to receive input from user  (a "buffer")
    parent2:    .space 17
    child1:     .space 17
    child2:     .space 17

    prompt1:    .asciiz "Enter your first string with no spaces (16 total): "
    prompt2:    .asciiz "\nEnter your second string with no spaces (16 total): "
    prompt3:    .asciiz "\nEnter an integer between 0 & 15:  "

################################# MAIN ####################
# (main method, calls macros, which call underlying methods)
.globl main
.text

    main:
        jal getInput            # get user inputs
        jal outputCtrl          # print out what the user entered in
        jal copySwapController  # run the hard logic loops
        jal printResults        # finally, print out the results
        j   end                 # jump to the end (don't repeat code below)

################################# get inputs ################# (Controls getInput1-3)

    getInput:

################################# copy/swap controller ########

    copySwapController:

################################# print output ####################

    outputCtrl:

#################################  Other functions ####################


################################# end ##################### (end of program)

    end:
        li SysCode, 10 # end the program
        syscall     # end the program
```

3 Syscalls

You are required to use the input/output and string storage features (syscall, .asciiz) to print out prompts asking the user for the necessary data and then read in the data. A number of system services, mainly for input and output, are available for use by your MIPS program. They are described in the table found under the syscalls tab of the MARS help page. MIPS register contents are not affected by a system call, except for result registers as specified in the help page.

How to use SYSCALL system services:

1. Load the service number in register $v0.
2. Load argument values, if any, in $a0, $a1, $a2, etc., per Mars Help.
3. Issue the SYSCALL instruction.
4. Retrieve return values, if any, from result registers as specified.

Example: Display the value stored in $t0 .

```
li    $v0, 1            # service 1 is print integer
add   $a0, $t0, $zero   # load desired value into argument register $a0
syscall
```

4 Example display on console

(your prompts should be exactly as follows)

```
Enter your first string with no spaces (16 total):  aaaabbbbccccdddd
Enter your second string with no spaces (16 total): 0000111100001111
Enter an integer between 0 & 15 (to be later used as an upper limit
for splitting the arrays): 6

You entered the following:
aaaabbbbccccdddd
0000111100001111
6

Creating newly evolved candidates from your specified inputs...
Child One and Child Two are shown below:

aaaabbb100001111
0000111bccccdddd
-- program is finished running --
```

5 Macros

Macros are a form of text replacement. They allow you to use one line of code to replace multiple lines of code.

```
 ####################   MACROS  ####################
.macro print(%string)
  la $a0, %string
  li $v0, 4
  syscall
.end_macro
.macro printInt(%int)
  la $a0, %int
  lb $a0, 0($a0)
  li $v0, 1
  syscall
.end_macro
.macro scanString(%string)
  la $a0, %string
  li $a1, 17
  li $v0, 8
  syscall
.end_macro
.macro scanInt(%int)
  li $v0, 5
  syscall

  la $a0, %int
  sb $v0, 0($a0)
.end_macro

.macro terminate (%termination_value)
  li $a0, %termination_value
  li $v0, 17
  syscall
.end_macro

####################   DATA SEGMENT  ####################
.data

prompt1: .asciiz "Enter your info here: "
prompt2: .asciiz "Enter more useful info here: "

outMsg: .asciiz "Here's some output: "

#Input buffers
```

```
buffer1:   .space 17
buffer2:  .space 17

#Constants
newline: .asciiz "\n"

####################   TEXT SEGMENT  ####################
.text
main:
  #Prompt user
  print(prompt1)
  scanString(buffer1)
  print(newline)

  #Print output to the screen
  print(outMsg)
  print(newline)

  #return or exit
  terminate(0)        #a macro to exit code using a syscall
```

6 Rubric

```
Name:
Email:
Stu ID:
---------------------

REQUIREMENTS:
1. prompts and accepts the user input              (1 pts)
2. writes input strings to memory                  (1 pts)
3. writes output strings to console                (1 pts)
4. correctly does crossover                        (4 pts)
5. sufficient comments                             (1 pts)
6. effective use of functions/modularization (uses at least 2 functions)  (2 pts)

Proj: Evolutionary Algorithm Teaser
=======================================

Programs that don't assemble/compile will receive a grade of 0.

 points given for above:
 -----------------
1. input prompt:                       x/1
2. writes to memory:                   x/1
3. writes output to console:           x/1
4. implements crossover successfully:  x/4
5. comments adequately  :              x/1
6. uses modularization:                x/2

            total:    xx/10
====================
Sample run:

Enter first string  (16 total):  1111000011110000
Enter second string (16 total):  aaaabbbbaaaabbbb
Enter an integer between 0 & 15:   3

You entereed the following:
1111000011110000
aaaabbbbaaaabbbb
3

Creating new offspring with your specified inputs...
Child One and Child Two are shown below:

1111bbbbaaaabbbb
aaaa000011110000

Instructor Comments:
====================
```