Sean Brandon
spbrando@ucsc.edu
11/5/2020

CSE13s Fall 2020
Assignment 4: Bit Vectors and Primes
Design Document

Pre-lab Questions

1. Pseudo-code shown below
2. Pseudo-code shown below
3. Functions implemented in bv.c
4. When freeing allocated memory for my BitVector ADT, I avoid any memory leaks by first freeing the memory in the BitVector ADT and then freeing the BitVector ADT itself.
5. Well I'm sure that the line of code "bv_set_bit(v, 2);" is redundant since the first thing we do in this function is set ALL bits so we wouldn't need to set this one twice.

Introduction

A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers.

A Fibonacci number is a sequence of numbers that are the sum of the two previous numbers in the sequence, starting with 0 and 1. A Fibonacci prime is a number that is a Fibonacci number and also prime.

A Lucas number is a sequence of numbers that are the sum of the two previous numbers in the sequence, starting at 2 and 1. A Lucas prime is a number in the Lucas sequence that is also prime.

A Mersenne number is a number in a sequence such that 2^x - 1, with x being incrementally increased. A Mersenne prime is a number in the Mersenne sequence that is also prime.

A sieve is a way to find prime numbers by eliminating the numbers that are multiples of lower primes (we call these multiples, composite numbers).

A palindrome is a word or number that reads the same forward as it does backward. A palindromic prime number is a number that is both a prime number and palindrome.

In this lab I use the sieve of Eratosthenes to generate a list of prime numbers up to a specified number and determine whether it is a Fibonacci Prime (F), Lucas Prime (L) and/or a Mersenne Prime (M). I also determine whether each prime number is a palindrome for bases 2, 9, 10, and 12.

The Inputs for this program are:

1. -s : Prints out all primes and identifies whether or not they are Lucas, Mersenne and/or Fibonacci.
2. -p : Prints out palindromic primes in bases 2, 9, 10, and 12.
3. -n <value> " Specifies the largest value to consider, inclusively, for the prime number sieve. By default this value is 1000.

Top Level (pseudo-code)

Sequence.c

```
Global variables to track my most recent fibonacci, lucas, and mersenne numbers
Int current_f = 1, last_f = 0, current_l = 1, last_l = 2, last_m = 2

main(getopt arguments) {
    Read in program arguments with getopt & switch (set end_num in switch)
    This if statement below is for special primes
    If s {
        Create bitvector(end_num)
        Sieve bitvector
        Loop through v {
            If i is prime {
                Bool F = fib(i)
                Bool L = luc(i)
                Bool M = mar(i)
                print newline
            }
        }
        Delete bitvector
    }
    This if statement below is for palindromes
    if p {
        Create bitvector(end_num) named bv
        Sieve(bv)
        palindrome(bv, 2)
        Print newline
        palindrome(bv, 9)
        Print newline
        palindrome(bv, 10)
        Print newline
        palindrome(bv, 12)
        Delete bitvector
    }
    Return 0;
}

This function prints fibonacci if the number is a fibonacci prime
Void fib(int prime) {
    While current_f < prime {
        Temp = current_f
        Current_f += last_f
        Last_f = temp
    }
    If current_f == prime {
        Print fibonacci
        Return
    }
    Else {
        Return
    }
}
```

This function prints lucas if the number is a lucas prime

```
Void luc(int prime) {
    While current_l < prime {
        Temp = current_l
        Current_l += last_l
        Last_l = temp
    }
    If current_l == prime {
        Print lucas
        Return
    }
    Else {
        Return
    }
}
```

This function prints mersenne if the number is a mersenne prime

```
void mer(int prime) {
    While last_m < prime + 1 {
        Last_m *= 2
    }
    If last_m == (prime + 1) {
        Print mersene
        return
    }
    Else {
        Return
    }
}
```

This function loops through the prime numbers in a bit vector, converts them to the given base, determines if they are palindromes in that base, and prints them if they are. I separated this function into 2 other functions to help with organizing and clarity for myself.

```
void palindrome(BitVector *bv, int base) {
    Print base header
    For i in bv {
        If i is prime {
            Str = base_change(i, base) base changing function described below
            if (is_palindrome(str)) { palindrome checking function described below
                Print ($number_{base\ 10}$ = $number_{base\ x}$)
            }
        }
    }
    return
}
```

This function changes the base of a given number to the given base and converts it to a string

```
char* base_change(int num, int base) {
    Char str1[32]
    Char str2[32]
    Char letter;
    Int remainder
    do {
        Remainder = num % base
        Num = num / base
        This if is for all the numbers that need to be letters (10 = a, 11 = , b, etc.)
        If (remainder > 9) {
            Letter = (char)(87 + remainder)
            Letter += str2
            Str2 = letter
        }
        This is for all the numbers that don’t need to be letters
        Else {
            Str1 = (string)remainder
            Str1 += str2
            Str2 = str1
        }
    } while( num != 0);
    Return str2
}
```

This function determines if the given string is a palindrome

Pseudo-code given in lab doc but I modified it to make it my own

```
bool is_palindrome(char* str) {
    Int length
    for (length = 0; str[length] != ‘\0’; ++length); this for loop only finds the length of the string
    Length -=1;
    For i in length / 2 { for the first half of the string
        If str[i] != str[length - i] { checks if each char in the string is the same as its mirror on the other half of the sting. If it is not the function returns false
            Return false
        }
    }
    If it gets through the for loop then it is a palindrome so return true
    Return true
}
```

bv.c

```
BitVector *bv_create(uint32_t bit_len) {
    Malloc memory for bitvector
    Check that malloc worked
    Calloc memory for bitvector array inside of bitvector
    Return bitvector
}

Void bv_delete(BitVector *v) {
    free(memory in v)
    free(v)
}

Uint32_t bv_get_len(BitVector *v) {
    return v -> length
}

Void bv_set_bit(BitVector *v, uint32_t i) {
    Sets the bit at index i in v to true/prime/1
    Byte_position |= byte mask regarding bit position
}

Void bv_clr_bit(BitVector *v, uint32_t i) {
    Sets the bit at index i in v to false/composite/0
    Byte_position &= byte mask regarding bit position
}
Uint8_t bv_get_bit (BitVector *v, uint32_t i) {
    Returns the bit at index i in v
    Return (Byte_position |= mask of byte) shifted right back into the 1s place so its either a 0 or 1 returned
}

Uint8t bv_set_all_bits(BitVector *v) {
        Sets all bits in v to true/prime/1
        Loops through all bytes in bitvector and “or” masks them with 0xff
}
```

sieve.c (code given in lab document)

Bit vector is the perfect data type for this because the numbers we are dealing with can be described only 1 of 2 ways which makes them binary (prime or composite) so we only need one bit and that bits position in the bitvector to know what numbers are prime and composite in a given range.

The sieve sets all the bits to true (describing as prime) and then clears (describing as composite) 0 and 1. It the takes all the multiples of 2 and clears them. Then all the multiples of 3, and so on until it has cleared all multiples of numbers in the given range leaving only the prime numbers to be set to true/1.

```
Void  sieve(BitVector *v) {
        bv_set_all_bits(v);      // Sets all bits to be represented as prime (at first)
        bv_clr_bit(v, 0);
        bv_clr_bit(v, 1):
        bv_set_bit(v, 2);          // Unnecessary??
        For (starting at 2 loop incrementally through all bits of v) {  // Looping through v
                If bit is set to prime {
                        For ( all multiples of i in v) {
                                bv_clr_bit(v, the multiple of i)
                        }
                }
        }
}
```

Design Process

Over the course of this lab, I modified my design in the following ways:

- Well figuring out how to manipulate the bits in a bit vector took awhile and was a challenge. Because of this I had to edit my code in bv.c a few times and now that I know more about what is done I have added more top level details to my bv.c pseudo-code.
- Originally the functions I had to evaluate whether the given prime number was mersenne, lucas, or fibonacci prime as well returned boolean true or false and I was going to print the message using those boolean values in the loop for each prime number. However, I realized it was simpler to just have the functions that determine fibonacci, lucas, and mersene be void and print the output “tags” inside of those functions.

- I also originally thought I could use one function for lucas and fibonacci (which is definitely still an option if I were to write my code differently) but I decided to split these up to simplify the printing and because if I wanted to use global variables to keep track of the values then this would be the easiest way.
- I originally planned to pass pointers to my fib luc and mer functions to modify them that way but eventually decided to just have each function modify their own respective global variables and only take the prime number being evaluated as an input.
- I originally had a method for -s but since that would only be called once per program run I decided to get rid of it and just throw the contents of it into the if statement where the function was originally called in main.
- At first I forgot to convert numbers greater than 9 in bases above 10 to letters so I had to adjust my code when I remembered