

Assignment 4 - Bit Vectors and Primes Program Design

Pre-Lab Answers

Part 1

1. Pseudocode to determine if number is Fibonacci/Lucas/Mersenne prime

Begin is_fib module

 Declare found_fib as boolean initialized to false

 Begin For

 For (integer i initialized to 0, i < 20, increment i)

 Declare integer fibnum, assigning value of fib(in i as integer)

 Begin if

 If (fibnum == current)

 Return true

 End if

 End for

End is_fib

Begin is_lucas module

 Begin For

 Declare found_lucas as boolean initialized to false

 For (integer i initialized to 0, i < 20, increment i)

 Declare integer lucasnum, assigning value of lucas(in i as integer)

 Begin if

 If (lucasnum == current)

 Return true

 End if

 End for

End is_lucas

Being is_mersee module

 Begin for

 Declare found_mersenne as boolean initialized to false

 For (integer i initialized to 0, i <= 20, increment i)

 Declare integer mersennenum, assigning value of mersee(in i as integer)

 Begin if

 If (mersennenum == current)

 Return true

 End if

 End for

End is_mersee

2. Pseudocode to determine if number in any base is pseudocode
Begin is_Palindrome module
 Declare f as bool initialized to true
 Declare length as integer initialized to strlen(in s as string)
 Begin for
 For (integer i initialized to 0, i < length, i++)
 Begin if
 If (s[i] != s[length -(i+1)])
 Assign to f value of false
 End if
 End for
 Return f
End is_Palindrome

Part 2

1. The implementation of each BitVector ADT function:

bv_create module design

```
Begin bv_create(pass in unsigned integer bit_len)
    Dynamically allocate v of type BitVector structure
    Begin if
        if(v == NULL)
            Return 0
        End if
        Begin If
            If (bit_len < 1)
                bit_len= 1
            End if
            v->length = bit_len
            Dynamically allocate vector of v, of type int with size of bit_len
            Begin if
                if(v->vector == NULL)
                    Return 0
                End if
            Return v
        End bv_create module
```

bv_delete module design

```
Begin bv_delete ( BitVector *v)
    Call free(v->vector)
```

```
    Call free(v)
    return
End bv_delete module
```

bv_get_len module design

```
Begin bv_get_len(BitVector *v)
    Return v->length
End bv_get_len module
```

bv_set_bit module design

```
Begin bv_set_bit module (BitVector *v, uint32_t i)
    Declare bucket as integer assign value of i / 8
    Declare thebit as integer assign value of i % 8

    uint8_t thebyte = v->vector[bucket]
    uint8_t shiftbyte = (00000001 << thebit)
    uint8_t newresult = thebyte | shiftbyte
    v->vector[bucket] = newresult
End bv_set_bit module
```

bv_clr_bit module design

```
Begin bv_clr_bit(BitVector *v, uint32_t i)
    Declare bucket as integer assign value of i / 8
    Declare thebit as integer assign value of i % 8

    uint8_t thebyte = v->vector[bucket]
    uint8_t shiftbyte = ~(00000001 << thebit)
    uint8_t newresult = (thebyte & shiftbyte)
    v->vector[bucket] = newresult
End bv_clr_bit module
```

bv_get_bit module design

```
Begin bv_get_bit(BitVector *v, uint32_t i)
    Declare bucket as integer assign value of i / 8
    Declare thebit as integer assign value of i % 8
    uint8_t thebyte = v->vector[bucket]

    uint8_t shiftbyte = (00000001 << thebit)
    uint8_t newresult = thebyte & shiftbyte
    uint8_t valueinbit = newresult >> thebit
```

```
        return valueinbit  
End bv_get_bit module
```

bv_set_all_bit module design

```
Begin bv_set_all_bits(BitVector *v)  
    Begin for  
        for (Declare i as integer of 32 bits initialized to 0, i < v->length, increment i)  
            bv_set_bit(v, i)  
        End for  
    End bv_set_all_bits module
```

2. Memory leaks were avoided when allocated memory for the BitVector ADT was freed due to allowing new contiguous arrays or data to be created dynamically on the heap and thus used. It resulted in prevention of accidental out of bounds accessing and errors when out of the address, that might be used by the next array or data afterward. So freeing up the memory ensures that the system is not polluted with an overflow as well which might cause segmentation faultation.
3. To improve the runtime of the sieve() function a change that could be added to the code might be enumerating each value from 0 to the total input of numbers and then checking taking all non even numbers besides 2, and checking if they have any other numbers that divide into them besides 1 and that num itself, if yes then mark them as composite. Or keep finding the composites and remove them while doubling the rest of the numbers being found to get primes.

The program is an implementation of a bitvector that stores prime numbers from 0 to a total num specified by the user, using a sieve to mark all the bits in the vector with 1 if they are prime, and 0 if they are composite. Once this is created the sieve is run through and each bit is checked if it is prime then, on it, are done multiple checks such as checking if that prime is a special prime number that is in the fibonacci, lucas, or merseness series. A lucas series is the same as fibonacci just with a base case of 2 when num is 0 and 1 when the num is 1. A mersenne prime starts off with num as 2 then computes it to $2^{\text{num}} - 1$, for each num that is odd following after. When the prime number in the bitvector matches either of these three cases, the list of prime is printed in tabular format with the prime number and its feature of being a lucas, mersenne, or fibonacci to the right side. Next if specified the user can find all palindromes of the prime number that get converted to base of 2, 9, 10 and 12. The prime number is converted to proper string displaying the according base its in, then gets checked if palindromic, if it is: the current base along with the prime number and its palindromic form in that base is outputted side by side.

The following functional decomposition has been implemented, along with the supporting Pseudocode:

3.0 Bit Vectors and Primes Program

- 3.1 fib(in num as integer)
- 3.2 lucas(in num as integer)
- 3.3 mersenne(in num as integer)
- 3.4 is_mersenne(in current as integer)
- 3.5 is_lucas(in current as integer)
- 3.6 is_fib(in current as integer)
- 3.7 prime_printer(in bv as pointer to BitVector structure)
- 3.8 decimal_to_base(in num as integer, in base as integer)
- 3.9 isPalindrome(in s as string)
- 3.91 palindrome_primeprint(in bv as pointer to BitVector structure, in b as integer)

Data design

Define OPTIONS as string constant "spn:"

Declare next_input as string initialized to NULL

Declare default_num as integer initialized to 0

Declare s, p, n as bool initialized to false

Declare c as integer initialized to 0

Main module design

Begin Main (pass in argc as integer, in argv as string)

 Begin While

 While (c = getopt(pass in argc as integer, in argv as string, in OPTIONS as string)) does not equal -1

 Begin switch (c)

 Case 's'

 Assign value of true to s

 Break statement

 Case 'p'

 Assign value of true to p

 Break statement

 Case 'n'

 Assign value of true to n

 Assign value of optarg to next_input

 Assign value of next_input converted to integer, to default_num

```
                Break statement
            Default Case
                Display "Character not defined in the string"
                Return with exit status fail
        End switch
    End While

    Begin if
    If (argc == 1)
        Display "Error: no arguments supplied!"
        Return with exit status fail
    End If
    Begin if
    If(s == true)
        Begin If
        if(n == false)
            Assign to default_num value of 1000
        End if
        BitVector *bv = bv_create(default_prime)
        Call sieve(bv)
        Call prime_printer(bv)
        Call bv_delete(bv)
    End if
    Begin if
    If(r == true)
        Begin If
        if(n == false)
            Assign to default_num value of 1000
        End if

        BitVector *bv2 = bv_create(default_prime)
        Call sieve(bv2)
        palindrome_primeprint(bv2, 2)
        Call palindrome_primeprint(bv2, 9)
        Call palindrome_primeprint(bv2, 10)
        Call palindrome_primeprint(bv2, 12)
        Call bv_delete(bv2)
    End if
End Main
```

prime_printer module design

```
Begin prime_printer(BitVector *bv)
  Begin for
    for (uint32_t i = 2; i < bv->length; i++)
      bool isfib = is_fib(i)
      bool islucas = is_lucas(i)
      bool ismers = is_mersenne(i)

      if (bv_get_bit(bv, i) == 1)
        if (ismers == 1 && islucas == 1 && isfib == 1)
          Display " prime, mersenne, lucas, fibonacci, i"
          if (islucas == 1 && isfib == 0 && ismers == 0)
            Display " prime, lucas, i"
          if (islucas == 0 && isfib == 1 && ismers == 0)
            Display " prime, fibonacci, i"
          if (islucas == 0 && isfib == 0 && ismers == 1)
            Display "prime, mersenne, i"
          if (islucas == 0 && isfib == 1 && ismers == 1)
            Display "prime, mersenne, fibonacci, i"
          if (islucas == 1 && isfib == 0 && ismers == 1)
            Display "prime, mersenne, lucas, i"
          if (islucas == 1 && isfib == 1 && ismers == 0)
            Display " prime, lucas, fibonacci, i"
          if (islucas == 0 && isfib == 0 && ismers == 0)
            Display "prime, i"
        End if
      End for
    End for
  End prime_printer module
```

fib module design

```
Begin fib(int num) module
  Begin if
    if (num == 0 || num == 1)
      return num
    End if
  Begin Else
    Else return fib(num - 1) + fib(num - 2)
  End Else
End fib module
```

lucas module design

```
Begin lucas(int num) module
  Begin if
    if (num == 0)
      return 2
    End if
    else if (num == 1)
      return num
    else return lucas(num - 1) + lucas(num - 2)
    End else
  End lucas
```

mersenne module design

```
Begin mersenne(int num) module
  Declare val as integer initialized to 0
  Begin if
    if (num == 2)
      return 3
    End if
    if (num % 2 != 0)
      val = (pow(2, num) - 1)
    End if
    return val
  End mersenne module
```

decimal_to_base module design

```
Begin decimal_to_base module(int num, int base)
  Declare hex as character
  Declare base_string as string initialize to NULL
  Declare c as character
  Declare thestring as string initialize to NULL
  Declare array string of character size 10 // 10 bits fit 1000 (default num primes)
  Declare array newstring of character size 10
  Declare counter as integer initialize to 0

  While (num > 0)
    Declare x as integer initialize to num
    Assign to num, (num / base)
    Assign to remainder, x % base
    if (base > 10)
```



```
    if (remainder == 10)
        hex = 'a'
        c = hex
        string[counter] = c
        Increment counter
    else if (remainder == 11)
        hex = 'b'
        c = hex
        string[counter] = c
        Increment counter
    else if (remainder == 12)
        hex = 'c'
        c = hex
        string[counter] = c
        Increment counter
    else if (remainder == 13)
        hex = 'd'
        c = hex
        string[counter] = c
        Increment counter
    else if (remainder == 14)
        hex = 'e'
        c = hex
        string[counter] = c
        Increment counter
    else if (remainder == 15)
        hex = 'f'
        c = hex
        string[counter] = c
        Increment counter
    else
        c = remainder + '0';
        Assign to string[counter], c
        increment counter
Else
    c = remainder + '0';
    Assign to string[counter], c
    increment counter
End while
string[counter] = '\0'
```

```
thestring = string
Decrement counter
for (int i = counter, k = 0; i >= 0; i--, k++)
    newstring[k] = thestring[i];
    if (i == 0)
        newstring[k + 1] = '\0'
    End if
End for
base_string = newstring
Return base_string
End decimal_to_base module
```

palindrome_primeprint module design

```
Begin palindrome_primeprint (BitVector *bv, int b)
    int base = b
    Display "Base , base"
    Display (---- --)
    for (uint32_t i = 2; i < bv->length; i++)

        if (bv_get_bit(bv, i) == 1)
            char *newbase = decimal_to_base(i, base)
            bool palindrome = isPalindrome(newbase)

            if (palindrome == 1)
                Display "i = newbase"
            End if
        End if
    End for
End palindrome_primeprint
```