Theoritical Assignment 1

Sahil Dhull
160607

January 17, 2018

Solution 1:

The ADT for the universe of n digit natural number consists of an integer array that contains digits of an n-digit natural number as its elements.
Let us call this ADT as Num.
Note: int stands for integer.
Operations:

- CREATE: int × int × int → Num
  Creates the ADT.
  The first int defines the size of the ADT i.e. n.
  The second int defines the position in the ADT and the third int defines the digit to be placed at that position/index.
  Output is Num that was to be created.

- ADD: int × Num × Num → int × Num
  Adds two Num (n digit natural numbers).
  The value of n is given by the first int.
  Output includes the number of digits in the result (Num) after addition along with Num.

- SUB: int × Num × Num → int × Num
  Subtracts second Num from first Num digit by digit.
  The value of n is given by the first int.
  Output includes the number of digits in the result (Num) after subtraction along with Num.

- MUL: int × Num × Num → int × Num
  Multiplies the two Num given the number of digits (by int).
  Multiplication is done through digit by digit multiplication and then adding result in the output Num, maintaining the carry each time. int in the output determines the size of output Num.

- DIV: int × Num × Num → int × Num
  Divides the two Num given the number of digits (by int).
  Division is done by long division method in which multiple subtraction of divisor is done from the dividend.
  Output includes Quotient (Num) and number of digits in the quotient (int).

- SUCC: int × Num → Num
  Gives the successor of the n digit number in natural numbers.
  Takes number of digits (int) and Num as input and outputs the successor as Num.

- COMPARE: int × Num × Num → int
  Compares 2 n-digit numbers (Num) given the number of digits (int).
  Returns an integer: 1 if 1^{st} Num $>$ 2^{nd} Num; 0 if they are equal; -1 if 1^{st} Num $<$ 2^{nd} Num.

- CONSTANT: int → Num
  Takes number of digits as input (int) and outputs a constant n-digit number as Num.

- OUT: Num → Standard Output
  Prints the n-digit number Num.

Exceptions:

1. In SUB, if 1^{st} Num $<$ 2^{nd} Num, then output can't be Num. It will be negative integer.

2. In DIV, if 1^{st} Num $<$ 2^{nd} Num, then output can't be Num. It will be 0.

Solution 2:
PART 1: ADT Queue
Universe:
It consists of an array containing elements of type t. The count of elements is maintained through an integer.
Operations:

- CREATE: → Queue
  Creates Queue that is Empty.
  Count=0.
- ENQUEUE: t × Queue → Queue
  Adds the element at the end of the Queue.
  Increases the value of count by 1.
- DEQUEUE: t × Queue → Queue
  Removes the element present at the 0^{th} index.
  Shifts the remaining elements 1 place to the starting and decreases the count by 1.
- isEMPTY: Queue → int
  Checks if the Queue is empty or not.
  Returns 1 if Empty and 0 if not.

Exceptions:

1. Out of Space in ENQUEUE
2. In DEQUEUE, when the Queue is Empty.

PART 2: ADT Seq
Universe:
It consists of multi-dimensional array containing elements of type t. There will be n rows, each being the ADT Queue defined above and there will be n integers which maintain count of these n Queue.
Operations:

- CREATE: int → Seq
  Creates Seq that is Empty and contains number of rows (ADT Queue) given by int.
  Count=0 for all Queue.
- ENQUEUE: int × t × Seq → Seq
  Adds the element given by t at the end of the Queue at i^{th} position or i^{th} row where i is given by int.
  Increases the value of count of i^{th} Queue by 1.
- DEQUEUE: int × t × Seq → Seq
  Removes the element present at the 0^{th} index of the Queue at i^{th} position or i^{th} row where i is given by int.
  Shifts the remaining elements 1 place to the starting and decreases the count of i^{th} Queue by 1.

- isEMPTY: int × Seq → int
  Checks if the i^{th} Queue is empty or not.
  Returns 1 if Empty and 0 if not.

Exceptions:

1. Out of Space in ENQUEUE
2. In DEQUEUE, when the Queue is Empty.

Solution 3:

Psuedo Code of the program:

```
begin
    read n;
    read k;
    if(k=0)
        Write 1;
    y=1;
    while k>1 do begin
        if k is even
            n=n*n;
            k=k/2;
        else
            y=n*y;
            n=n*n;
            k=(k-1)/2;
    end;
    n=n*y;
    Write n;
end
```

RAM Program:

Label	Operation Code	Address	Comments	Label	Operation Code	Address	Comments
	READ	1	read n	Even:	LOAD	1	When k is even
	READ	2	read k		MUL	1	
	LOAD	2	if k=0		STORE	1	
	JZERO	Output			LOAD	2	
	LOAD	=1	y=1		DIV	=2	
	STORE	3			STORE	2	
While:	LOAD	2	if k=1		JUMP	While	
	SUB	=1		Odd:	LOAD	3	When k is odd
	JZERO	endwhile			MUL	1	
	LOAD	2	Checking Even or Odd		STORE	3	
	DIV	=2			LOAD	1	
	MUL	=2			MUL	1	
	SUB	2			STORE	1	
	ADD	=1			LOAD	2	
	JZERO	Odd			SUB	=1	
	JGTZ	Even			DIV	=2	
endwhile:	LOAD	1	When k=1		STORE	2	
	MUL	3			JUMP	While	
	STORE	1		Output:	WRITE	=1	When k=0
	WRITE	1			HALT		
	HALT						

Solution 4:
The TM program used here uses 2 tapes X and Y. The tape X is the the input tape that consists of k consecutive 1's and tape Y will be used in the program to first copy the content of tape X and then writing 2k consecutive 1's on tape X.
Following is the table showing Turing Machine Execution.

Current State	Symbol on:		(Next Symbol,Head Move)		New State
	Tape X	Tape Y	Tape X	Tape Y	
q_0	1	b	(1,R)	(0,R)	q_1
q_0	1	1	(1,R)	(1,S)	q_2
q_0	b	1	(1,R)	(1,S)	q_2
q_0	b	b	(b,S)	(b,S)	q_3
q_1	1	b	(1,R)	(1,R)	q_1
q_1	b	b	(b,L)	(b,L)	q_1
q_1	1	1	(1,L)	(1,L)	q_1
q_1	1	0	(1,S)	(1,S)	q_0
q_2	1	1	(1,R)	(1,R)	q_0
q_2	b	1	(1,R)	(1,R)	q_0

Where q_0 is the initial state, q_3 is the final state, b denotes blank cell, R denotes Right head move, S denotes Stationary head move.
How is the program implemented:

1. Initially Tape heads are at 1st blocks of X and Y respectively and TM is in state q_0. Tape Y contains 1st block as b and Tape X may contain 1st block as 1 or b.

2. If the tape heads read b on tape X and b on tape Y while in state q_0, then it will output b on both tapes, remain stationary and TM enters into state q_3 which indicates that program is executed.

3. If the tape heads read 1 on tape X and b on tape Y while in state q_0, then they will output 1 on tape X and 0 on tape Y (for marking purposes), and tape heads move right and TM enters state q_1.

4. Now in state q_1, basically tape X is being copied into tape Y and then tape heads return to the initial blocks of the tape.
In state q_1, if tape heads read 1 on tape X and b on tape Y, they will output 1 on both the tapes and move right, until at last when they encounter b on both the tapes, copying stops and tape heads move left and TM remains in state q_1.
Now in state q_1, if tape heads read 1 on both tapes, they will output 1 on both the tapes and move left, until at last when they encounter 1 on tape X and 0 on tape Y (i.e. the starting of the tapes is reached by the tape heads) and tape heads output 1 on both tapes and remain stationary and TM enters state q_0.

5. Now the tape heads will read 1 on both tapes, they will output 1 on both the tapes, move tape head on X to the right and tape head on Y remains stationary (because we have to print two 1s) and TM enters state q_2.

6. In q_2 state, tape heads will read 1 on Y (because tape head on Y was stationary after encountering 1 in state q_0) and 1 or b on X. In both the cases they will output 1 on both tapes, move right and enter state q_0.

7. Similarly the above 2 steps keep on repeating until tape heads read b on both tapes in state q_0 and then TM goes into state q_3 which indicates that program is executed.

And so for every single 1 in tape X, there are two 1s in the tape X finally and hence k consecutive 1s in input are replaced by 2k consecutive 1s.