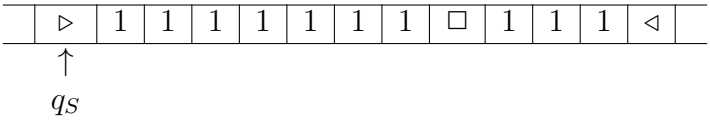
Lab10-Turing Machine

Algorithm and Complexity (CS214), Xiaofeng Gao, Spring 2020.

∗ If there is any problem, please contact TA Yiming Liu.
∗ Name:Haotian Xue Student ID:518021910506 Email: xavihart@sjtu.edu.cn

1. Design a one-tape TM M that computes the function $f(x, y) = x \mod y$, where x and y are positive integers ($x > y$). The alphabet is $\{1, 0, \square, \triangleright, \triangleleft\}$, and the inputs are x 1's, $\square$ and y 1's. Below is the initial configuration for input $x = 7$ and $y = 3$. The result $z = f(x, y)$ should also be represented in the form of z 1's on the tape with the pattern of $\triangleright 111 \cdots 111 \triangleleft$.

Initial Configuration



   (a) Please describe your design and then write the specifications of M in the form like $\langle q_S, \triangleright\rangle \to \langle q_1, \triangleright, R\rangle$. Explain the transition functions in detail.

   (b) Please draw the state transition diagram.

   (c) Show briefly and clearly the whole process from initial to final configurations for input $x = 7$ and $y = 3$. You may start like this:

   $$(q_s, \underline{\triangleright}1111111\square 111\triangleleft) \vdash (q_1, \triangleright\underline{1}111111\square 111\triangleleft) \vdash^* (q_1, \triangleright 1111111\underline{\square}111\triangleleft) \vdash (q_2, \triangleright 1111111\square\underline{1}11\triangleleft)$$

   (Note that for simplicity, we write $(q_1, \triangleright\underline{1}111111\square 111\triangleleft) \vdash^* (q_1, \triangleright 1111111\underline{\square}111\triangleleft)$ if the corresponding transaction repeats on multiple inputs with the same state.)

Solution. The solutions for this problem is as following.

(a) I implement the mod operation x *mod* $y$by repeatedly substract y from x untill x is not big enough to be substracted. And the Turing Machine process can be divided into three parts: the main substract loop and two conditional jumps. That is, in the main loop, we repeat the substract of x by turning some $1s$ in the end of x into $0s$. The first conditional jump is for the bits in y, when y become all-zeros, we turn it into all-ones to make the next substraction. The other jump happens when x is turned into all-zeros but y is not, which means we should turn some $0s$ of x into $1s$, and we call it resume operation.

We have many states in the implementation and they are:

- q_1: this is also the starting state, in this state we move right to the end of y.
- q_2: in this state we turn the last 1 of y into 0.
- q_3 :move to the end of x.
- q_4 : change the last 1 in x into 0.
- q_5: move to the starting point and repeat substraction.
- q_6: when y become all-zeros, make y all-ones.
- q_7: when x is all-zeros, move the end of y.
- q_8: move to the last 0 in y.
- q_9: move to the last 1 in x.

- q_{10}: resume operation.
- q_{11}: y is all-zeros and resume operation is done.
- q_{12}: move to the end of x.
- q_H: turn the fisrt 0 in x into the end notation.

Main substraction loop:$(q_1 \sim q_5)$, jump1(q_6), jump2$(q_7 \sim q_{12} + q_H)$.

M is designed as follows, and to make it more briefly to read, $\langle q, (a_1, a_2, ..., a_i)\rangle \to \langle q, (b_1, b_2, ..., b_i)), R\rangle$ means: $\langle q, a_k\rangle \to \langle q, b_k, R\rangle (k = 1, 2, ..., i)$.

$\langle q_1, (0, 1, \square, \triangleleft)\rangle \to \langle q_1, (0, 1, \square, \triangleleft), R\rangle$

$\langle q_1, \triangleleft\rangle \to \langle q_2, \triangleleft, L\rangle$ // move to the end of y

$\langle q_2, 1\rangle \to \langle q_3, 0, L\rangle$ // change the last 1 in y into 0

$\langle q_3, 1\rangle \to \langle q_3, 1, L\rangle$

$\langle q_3, \square\rangle \to \langle q_4, \square, L\rangle$

$\langle q_4, 1\rangle \to \langle q_5, 0, L\rangle$ // turn the last 1 of x into 0

$\langle q_5, \triangleright\rangle \to \langle q_5, /triangleright, R\rangle$ // return to starting point

$\langle q_2, \square\rangle \to \langle q_6, \square, R\rangle$ // jump then y is all $0s$

$\langle q_6, 1\rangle \to \langle q_6, 0, R\rangle$ // make y into all $1s$

$\langle q_6, \triangleleft\rangle \to \langle q_2, \triangleleft, L\rangle$ y is all make into 1

$\langle q_4, \triangleright\rangle \to \langle q_7, \triangleright, R\rangle$ // begin to resume

$\langle q_7, (1, 0, \square)\rangle \to \langle q_7, (1, 0, \square), L\rangle$

$\langle q_7, \triangleleft\rangle \to \langle q_8, \triangleleft, L\rangle$

$\langle q_8, 1\rangle \to \langle q_8, 1, L\rangle$ // turn to the last 0 in y

$\langle q_8, 0\rangle \to \langle q_9, 1, L\rangle$ // change the last 0 of y into 1

$\langle q_9, (0, \square)\rangle \to \langle q_9, (0, \square), L\rangle$

$\langle q_9, (0, \triangleright)\rangle \to \langle q_{10}, (0, \triangleright), L\rangle$ // move to the first 0 in x

$\langle q_{10}, 0\rangle \to \langle q_7, 1, R\rangle$ //write the first 0 in x into 1 and repeat the resume operation

$\langle q_8, \square\rangle \to \langle q_{11}, \square, L\rangle$

$\langle q_{11}, 0\rangle \to \langle q_{11}, 0, L\rangle$ // find the first 0 in x

$\langle q_{11}, 1\rangle \to \langle q_{12}, 1, L\rangle$

$\langle q_{12}, 0\rangle \to \langle q_H, \triangleleft, L\rangle$

(b) State transition diagram is shown is Fig 1.

(c) To make the answer of this part more briefly to read, I will divided the process into two main sections and in each section there are some subsections, and I will ommit some of the subsections since they are the same to work.

- **Substract** 3 **from** $7(7 - 3 = 4) \vdash (q_1, \underline{\triangleright}1111111\square111\triangleleft)$
  $\vdash^* (q_1, \triangleright1111111\square11\underline{1}\triangleleft)$
  $\vdash (q_2, \triangleright1111111\square11\underline{1}\triangleleft)$
  $\vdash (q_3, \triangleright1111111\square11\underline{0}\triangleleft)$
  $\vdash (q_4, \triangleright1111111\underline{1}\square110\triangleleft)$

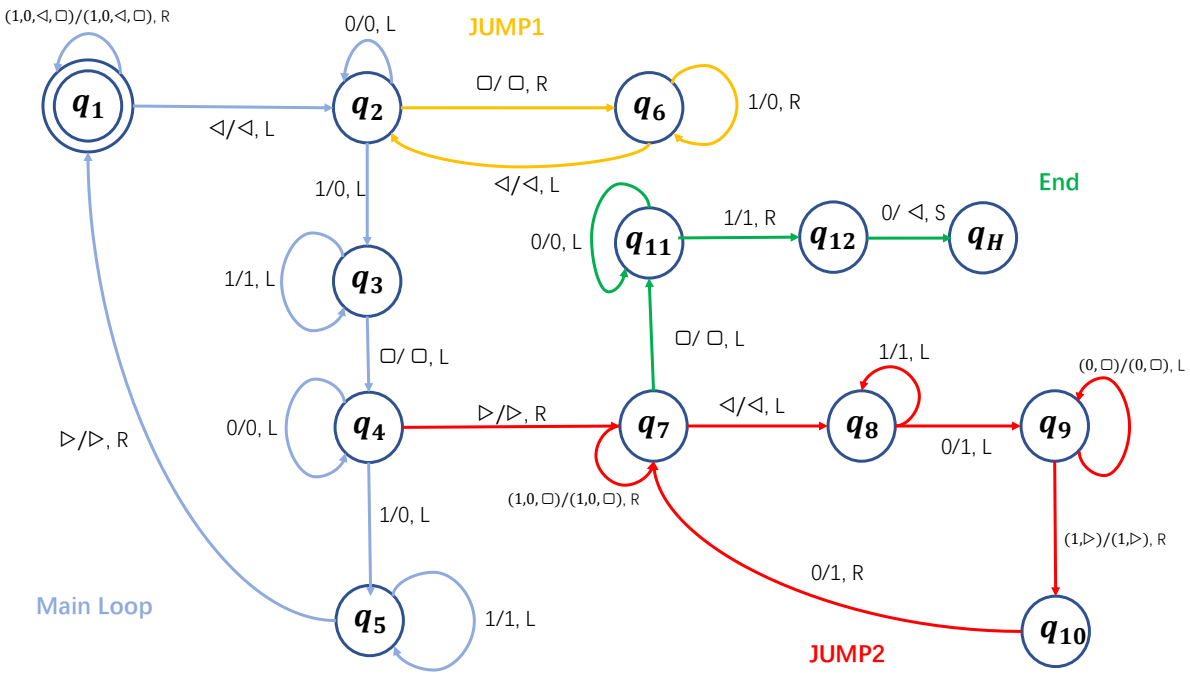


Fig. 1. Problem 1-(b)

$\vdash (q_5, \triangleright 1111110\underline{\square} 110\triangleleft)$
$\vdash^* (q_5, \triangleright \underline{1}111110\square 110\triangleleft)$
$\vdash (q_1, \underline{\triangleright} 1111110\square 110\triangleleft)$
we ommit the other two $sub-1$ operations and directly turn to the conditional jump:
$\vdash (q_1, \underline{\triangleright} 1111000\square 000\triangleleft)$
$\vdash^* (q_1, \triangleright 1111000\square 00\underline{0}\triangleleft)$
$\vdash (q_2, \triangleright 1111000\underline{\square} 000\triangleleft)$
$\vdash (q_6, \triangleright 1111000\square \underline{1}00\triangleleft)$
$\vdash^* (q_6, \triangleright 1111000\square 11\underline{1}\triangleleft)$
$\vdash (q_2, \triangleright 1111000\underline{\square} 111\triangleleft)$
So we successfully turn 3 into 111 and we continue the substraction untill 7 becomes 0000000:

- **Resume operation:**
$\vdash (q_1, \underline{\triangleright} 0000000\square 110\triangleleft)$
$\vdash^* (q_1, \triangleright 0000000\square 110\triangleleft)$
$\vdash (q_2, \triangleright 0000000\underline{\square} 110\triangleleft)$
$\vdash (q_3, \triangleright 0000000\underline{\square} 110\triangleleft)$
$\vdash (q_4, \triangleright 0000000\underline{0}\square 110\triangleleft)$
$\vdash^* (q_4, \underline{\triangleright} 0000000\underline{\square} 110\triangleleft)$
$\vdash (q_7, \triangleright 0000000\square 110\underline{\triangleleft})$
$\vdash (q_8, \triangleright 0000000\square 11\underline{0}\triangleleft)$
$\vdash^* (q_8, \triangleright \underline{0}000000\square 111\triangleleft)$
$\vdash (q_9, \triangleright \underline{0}000000\square 111\triangleleft)$
$\vdash (q_{10}, \triangleright \underline{1}000000\square 111\triangleleft)$
$\vdash (q_7, \triangleright \underline{1}000000\square 111\triangleleft)$

And then we turn to check if 3 is all ones and end the process:

$\vdash^* (q_7, \triangleright 1000000 \square 111 \underline{\triangleleft})$

$\vdash (q_8, \triangleright 1000000 \underline{\square} 111 \triangleleft)$

$\vdash (q_{11}, \triangleright 100000 \underline{0} \square 111 \triangleleft)$

$\vdash^* (q_{11}, \triangleright \underline{1} 000000 \square 111 \triangleleft)$

$\vdash (q_{12}, \triangleright 1 \underline{0} 00000 \square 111 \triangleleft)$

$\vdash (q_H, \triangleright 1 \underline{\triangleleft} 00000 \square 111 \triangleleft)$

□

2. Assume there's a Turing Machine M using alphabet $\Gamma : \{\triangleright, \square, a, b, \cdots, z\}$. We can simulate M by a Turing Machine $\tilde{M}$ using alphabet $\tilde{\Gamma} : \{\triangleright, \square, 0, 1\}$. Please transform the instruction $\langle q, i\rangle \to \langle q', j, R\rangle$ in M into its corresponding form in $\tilde{M}$.

Solution. The solutions for this problem is as following.

We decode $a, b, ..., z$ by using 5 bits of data in order from 00001 to 11010, and in the case referred to i and j, we can decode them as 01001 and 01010 respectively.

We have two kind of states in the solution and they are:

- $q_{read(x,y,z)}$: it is used to read the five bits to redecode the 5-bits, where x represents the number of bit read and $x \in \{5, 4, 3, 2, 1\}$; y represents the value read now and $y \in \{1, 2..., 26\}$, z is used to determine the current situation and in this case we name state q as $z = 1$ and $q^{'}$ as $z = 2$.
- $q_{retreat(x,y,z)}$: make state change and retreat to the beginning of the 5-bits to wait for write operation. x is used to represent the current position.
- $q_{write(x,y,z)}$: rewrite the 5-bits, in this example we should write them into 01010.

$\tilde{M}$ can be divided into three parts accordingly:

- **Read**: This part can be used in all conditions(not restricted to $i, j, q, q^{'}$ in this case):

$$\langle q_{read(x,y,z)}, p\rangle \to \langle q_{read(x,y+p\times 2^{x-1},z)}, p, R\rangle$$

  z is in $\{5, 4, 3, 2\}$, when z becomes 1, we read the last bit and to transformation, in this case we transform$z$ from 1 to 2:

$$\langle q_{read(1,y,1)}, p\rangle \to \langle q_{retreat(1,y+p,2)}, p, S\rangle$$

- **Retreat**: In this part we move back to the first bit, which is also of universal use:

$$\langle q_{retreat(x,y,z)}, p\rangle \to \langle q_{retreat(x-1,y,z)}, p, L\rangle$$

  where x is in $\{5, 4, 3, 2\}$, and when x becomes 1, we reach the beginning of i, we change state to $write$:

$$\langle q_{retreat(1,y,z)}, p\rangle \to \langle q_{write(1,y,z)}, p, S\rangle$$

- **Write**:
  Then we write the five bits into 01010, to make it more general we define 01010(j) as $a_1a_2a_3a_4a_5$ and 01001(i) as $b_1b_2b_3b_4b_5$:

$$\langle q_{retreat(i,y,2)}, b_i \rangle \rightarrow \langle q_{write(i+1,y,2)}, a_i, R \rangle$$

  where we have i is in $\{1, 2, 3, 4\}$, and in this case y is 9. When i is 5, we write the last bit, turn value into 0 and then change state to read the next word:

$$\langle q_{retreat(5,y,2)}, b_i \rangle \rightarrow \langle q_{read(i+1,0,2)}, a_i, R \rangle$$

□

3. **Wireless Data Broadcast System.** In a Wireless Data Broadcast System (WDBS), data items are repeatedly broadcasted in cycle on different channels. Denote $D = \{d_1, d_2, \cdots, d_k\}$ as data items, each d_i with length l_i (as time units), and $\mathbf{C} = \{C_1, C_2, \cdots, C_n\}$ as broadcasting channels. Fig. illustrates a WDBS with 25 data items and 4 channels. Once a channel finishes broadcasting current cycle, it will repeat these data again as a new cycle. E.g., a possible broadcasting sequence of C_1 could be $\{d_6, d_{12}, d_1, d_{18}, d_7, d_6, d_{12}, d_1, d_{18}, d_7, \cdots\}$

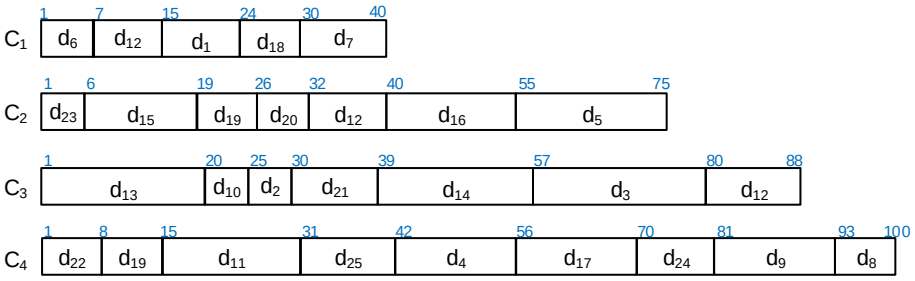


Fig. 2. An Example Scenario of Wireless Data Broadcast System.

If a mobile client requires a subset of data items $D_q \subseteq D$ from this WDBS, he/she must access onto one channel, wait for the appearance of one required item, and switch to another channel if necessary. Each "switch" requires one time slot. For example, Lucien wants to download $\{d_1, d_3, d_5\}$, as shown in Fig.. He firstly accesses onto C_1 at time slot 1, then download d_1, d_3 respectively during time slots 2 to 5, and then switch to C_3 at time slot 6 (note that he cannot download d_5 from C_2 because of the switch constraint), and download d_5 during time slots 7 to 8. We define *access latency* as the period when a client starts downloading, till the time he/she finishes. As a result, the overall access latency for Lucien is 7 in this example.

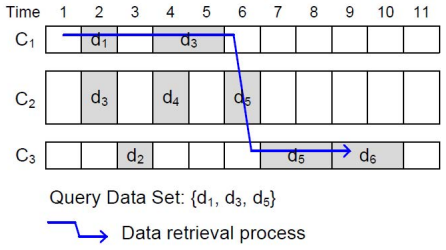


Fig. 3. An Example Scenario of Query of a Client.

Each operation (download/wait/switch) needs energy consumption. To conserve energy, a client hopes to use minimum amount of energy to download all required items in D_q, which

means that he/she waits to minimize both access latency and switch numbers. Unfortunately, these two objectives conflict with each other naturally. Fig. exhibits such a scenario. To download $D_q = \{d_1, d_2, d_3, d_4\}$, if we start from C_2, in Option 1 we can switch to C_1 for d_1 immediately after downloading d_3, return back to C_2 for d_4, and to C_1 again for d_2. Such option costs 3 switches and 7 access latency. While in Option 2, we stay at C_2 lazily for d_3 and d_4, and then switch to C_1 for d_2 and d_1. Such option costs 1 switches and 12 access latency.

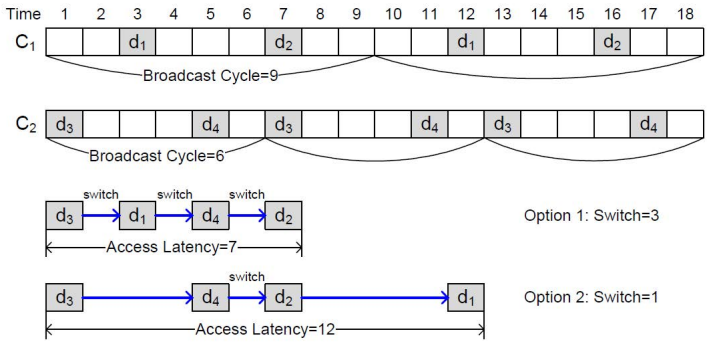


Fig. 4. Confliction between Access Latency and Switch Number.

Once we want to minimize two conflictive objectives simultaneously, we have three possible ways (similar as Segmented Least Squares told in Dynamic Programming Lecture). Now it is your turn to complete the formulation of this optimization, we name it as Minimum Constraint Data Retrieval Problem (MCDR), with the following sub-questions.

(a) If we add an additional switch parameter h, please define the MCDR (Version 1) completely as a search problem.

(b) If we add an additional latency parameter t, please define the MCDR (Version 2) completely as a search problem.

(c) If we set dimensional parameters α to switch number, and β to access latency, we can combine two objectives together linearly as a new concept "cost". Please define the Minimum Cost Data Retrieval Problem (MCDR, Version 3) correspondingly.

(d) Please give the decision versions of sub-questions (a), (b) and (c).

Solution. The solutions for this problem is as follows:

We first try to make a more mathematical abstraction of this problem:

We have n channels $C_1, C_2, ...C_n$ and each one is made up of certain circulating block of data: the i_th data item in C_j is $item_{ji}$. The requests is $D = \{d_1, d_2, ..., d_k\}$, what we try to find is a possible way to get the data we want from the channels in a specific sequence: $D^{'} = \{d_1^{'}, d_2^{'}, ..., d_k^{'}\}$.

Since we can fetch data from different channels in different time , we add two subindexs to define when and where and data is fetched, that is : $D_s = d_1^{t_1,c_1}, d_2^{t_2,c_2}, ..., d_k^{t_k,c_k}$, and we have additional requirements to make it more feasible:

- **Time sequence**: $t_i < t_j$ $(i < j)$
- **Switch constraint**: if c_i is not equal to c_{i+1}, then $t_i + 1 < t_{i+1}$
- **Data accuracy**: $d_{c_i t_i} = item_{c_i t_i}$
- **Data integrity**: D_s is a permutation of D

and we can calcualte the access latency(**AL**) and the switch numbers(**SN**) like this:

$$AL = t_k - t_1$$

$$SN = \sum_{i=1}^{k-1} \mathbb{I}(c_i, c_{i+1})$$

where $\mathbb{I}(a, b) = 1$ when $a = b$, otherwise it is 0.

(a) We define the cost for feasible solution x as $Cost(x) = AL(x) + hSN(x)$ and the search problem can be described like this:

- **Solution set:** X, $D_s \in X$
- **Instance:** solution s with the order D_s
- **Feasible solutions:** $s_x \in X$
- **Algorithem** : search for the s , s.t. $Cost(s) = min(Cost(s_x))$

(b) We define the cost for feasible solution x as $Cost(x) = tAL(x) + SN(x)$ and the search problem can be described like this:

- **Solution set:** X, $D_s \in X$
- **Instance:** solution s with the order D_s
- **Feasible solutions:** $s_x \in X$
- **Algorithem** : search for the s , s.t. $Cost(s) = min(Cost(s_x))$

(c) We define the cost for feasible solution x as $Cost(x) = \alpha AL(x) + \beta SN(x)$ and the search problem can be described like this:

- **Solution set:** X, $D_s \in X$
- **Instance:** solution s with the order D_s
- **Feasible solutions:** $s_x \in X$
- **Algorithem** : search for the s , s.t. $Cost(s) = min(Cost(s_x))$

(d)

(a), (b), (c) differ in the definition of cost, so we generalize the cost to describe the Decision Porbelm, which can be used to transform all the three questions:

- **Solution set:** X, $D_s \in X$ iff $Cost(D_s) < k$
- **Instance:** solution s with the order D_s
- **Algorithem A**: $A(s) = yes$ iff $s \in X$

□