Quantum Algorithms
Lecture 1
Turing machines

Zhejiang University

Introduction

Abstract notion of computability

We only need a few basic properties.

Primitive programming language - for Turing machine (TM). A little of programming experience (in any language) suffices.

Algorithm

A set of instructions.

"We need only to carry out what is prescribed as if we were robots: neither understanding, nor cleverness, nor imagination is required of us".

Input --(algorithm)-> Output

Computation may also never terminate.

Input and output

Represented by strings.

We call a finite set of symbols to be used to define strings an alphabet.

String – finite sequence of symbols, where symbols are taken from some alphabet. We can also call an alphabet as our encoding.

Example: 15 -> 5,3

Here 15 is our input, and 5,3 is our output, representing two separate numbers – 5 and 3.

Input and output

It is important to pick alphabet properly. It should be possible for a string to distinguish each separate symbol of which it consists.

For example, {0, 1} and {00, 11} are alphabets, but {1, 11} is not an alphabet because 11 can be formed by either 11 or (1 and 1). So, in the second case we are not sure which symbols formed our string “11”.

More details about strings is on pages 6 and 7 of the Book.

Importance of encoding

Large integers as bit strings (binary) - easy to compare, difficult to multiply.

Integers represented by remainders modulo different primes $p_1, p_2, \dots, p_n$- easy to multiply, but comparison is difficult.

One more remark – input length can be different if we use unary encoding instead of binary (exponential difference):

9 = 1001 = 111111111

Definition of a Turing machine

Definition

Components of TM:

- a finite set S called the alphabet;
- an element $\sqcup \in S$ (blank symbol);
- a subset $A \subset S$ called the external alphabet; we assume that the blank symbol does not belong to A;
- a finite set Q whose elements are called states of the TM;
- an initial state $q_0 \in Q$;
- a transition function, specifically, a partial function

$$\delta\colon Q \times S \to Q \times S \times \{-1, 0, 1\}.$$

TM = Program

There are infinitely many Turing machines, each representing a particular algorithm.

Mentioned components may be considered as a computer program.

We provide this algorithm as instructions to the Turing machine, and these instructions are executed depending on the current situation during the process of computation.

TM = Program

Turing Machine to Add Two Integers

Input Tape =B0110110B

Transition Functions

$(q_0, 0) = \{q_1, 0, R\}$

$(q_1, 1) = \{q_1, 1, R\}$

$(q_1, 0) = \{q_2, 0, R\}$

$(q_2, 1) = \{q_3, 0, L\}$

$(q_2, 0) = \{q_5, B, R\} = \{F\}$

$(q_3, 0) = \{q_4, 1, R\}$

$(q_4, 1) = \{q_1, 0, L\}$

$(q_4, 0) = \{q_2 0, R\}$

Computation Trace

$q_0\,0110110 \vdash 0q_1\,110110 \vdash 01\,q_1\,10110 \vdash 011\,q_1\,0110$

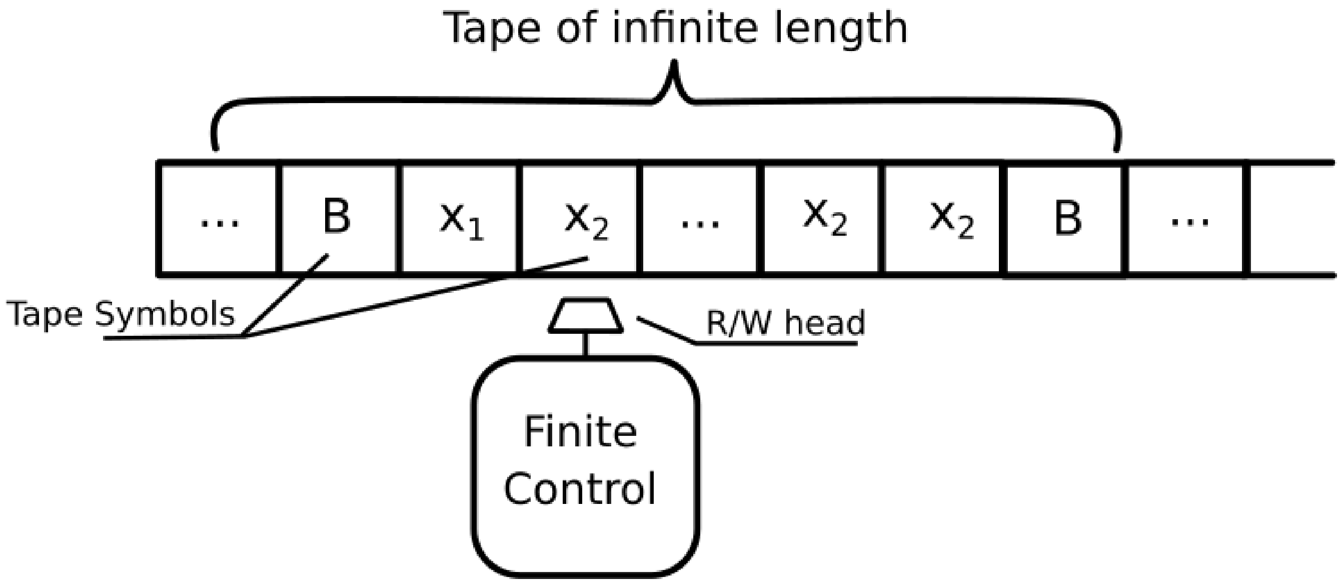
$\vdash 0110\,q_2\,110 \vdash 011\,q_3\,0010 \vdash 0111\,q_4\,010 \vdash 01110\,q_2\,10$

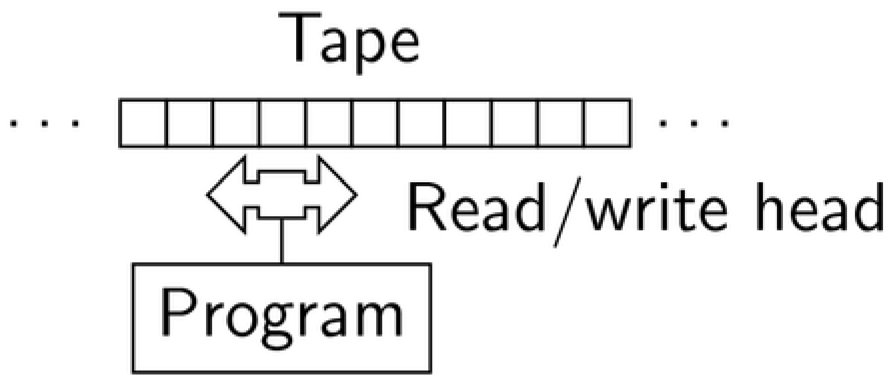
$\vdash 0111\,q_3\,000 \vdash 01111\,q_4\,00 \vdash 011110\,q_2\,0 \vdash 011110B\,q_5 \vdash \{F\}$

In this example numbers are given in unary and delimited by 0. So B0110110B means 11+11, and result is 1111.

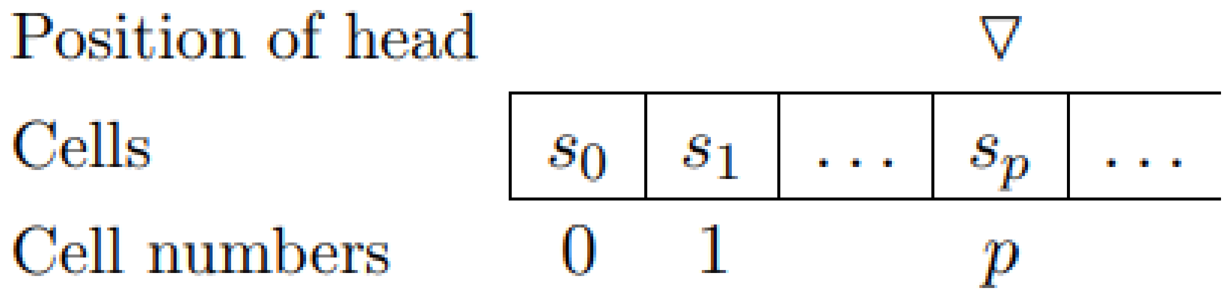
Here R=right, L=left, F=finish, B=blank symbol

TM hardware

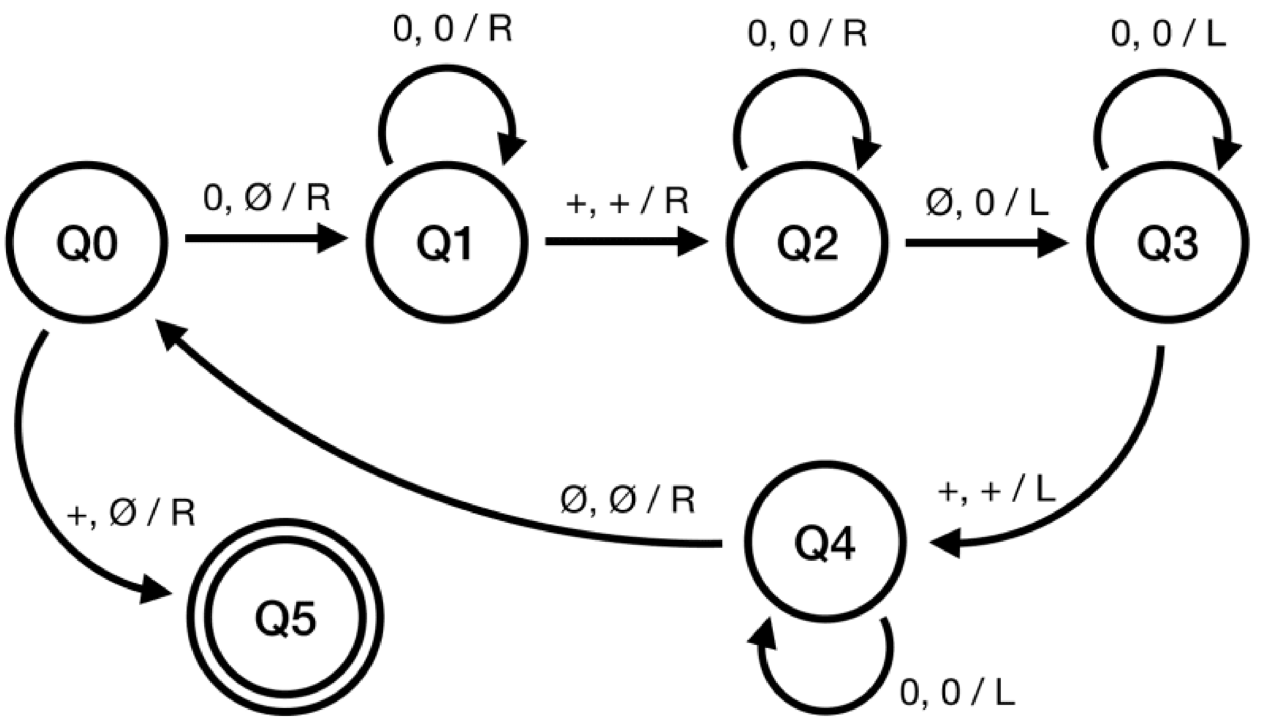

Tape of infinite length
...
B
x_1
x_2
...
x_2
x_2
B
...
Tape Symbols
R/W head
Finite Control


Tape
. . .
. . .
Read/write head
Program

TM tape


Position of head
∇
Cells
s_0
s_1
$\dots$
s_p
$\dots$
Cell numbers
0
1
p

Control device



Here R=right, L=left, ø=blank symbol

Configuration of a TM

A triple $< \sigma;\ p;\ q >$:

- σ is an infinite sequence $s_0, \dots, s_n, \dots$ of elements of S - contents of the tape;
- p is a nonnegative integer - position of the head;
- $q \in Q$ - internal state.

Computational step of TM

- TM reads the symbol s_p.
- TM computes the value of the transition function: $\delta(q, s_p) = (q', s', \Delta p)$; if step is not defined, TM stops.
- TM writes the symbol s in cell p of the tape, moves the head by Δp, and passes to state q'.

New configuration is

$$< s_0, \dots, s_{p-1}, s', s_{p+1}, \dots ;\ p + \Delta p;\ q' >$$

Computational step of TM

For example, suppose position of head is marked with green color and machine has the following program (transition function):

- $\delta(q_0, 0) \rightarrow \delta(q_1, 1, R)$
- $\delta(q_0, 1) \rightarrow \delta(q_1, 0, R)$
- $\delta(q_1, 0) \rightarrow \delta(q_0, 0, R)$
- $\delta(q_1, 1) \rightarrow \delta(q_0, 1, R)$

Current state: q_0
Tape: **0**101

In such case machine should apply $\delta(q_0, 0) \rightarrow \delta(q_1, 1, R)$

Then we have:

Current state: q_1
Tape: 1**1**01

Computational step of TM

For example, suppose position of head is marked with green color and machine has the following program (transition function):

- $\delta(q_0, 0) \rightarrow \delta(q_1, 1, R)$
- $\delta(q_0, 1) \rightarrow \delta(q_1, 0, R)$
- $\delta(q_1, 0) \rightarrow \delta(q_0, 0, R)$
- $\delta(q_1, 1) \rightarrow \delta(q_0, 1, R)$

Current state: q_1

Tape: 1**1**01

In such case machine should apply $\delta(q_1, 1) \rightarrow \delta(q_0, 1, R)$

Then we have:

Current state: q_0

Tape: 11**0**1

Overall computation

Inputs and outputs are strings over A.

Configuration in the beginning: an input string α is written on the tape and is padded by blanks; the head is at the left end of the tape; the initial state is q_0. Initial configuration:

$$\langle \alpha \sqcup \sqcup \ldots ; 0; q_0 \rangle$$

Overall computation

The configuration is transformed step by step using the rules of transition, and we get the sequence of configurations:

$\langle \alpha \sqcup \sqcup \ldots; 0; q_0 \rangle, < \sigma_1; p_1; q_1 >, < \sigma_2; p_2; q_2 >, \ldots$

Overall computation

The process terminates if δ is undefined or the head bumps into the (left) boundary of the tape.

The string before the symbol that does not belong to A will be the output of the TM. In other words, the content of work tape after the computation is the output of the algorithm.

Computable functions and decidable predicates

Computing a function

Every Turing machine M computes a partial function ϕ_M: $A^* \to A^*$, where A^* is the set of all strings over A.

α --(computation)-> $\phi_M(\alpha)$

$\phi_M(\alpha)$ is undefined if the computation never terminates.

Computable function

A partial function f from A^* to A^* is computable if there exists a Turing machine M such that $\phi_M = f$. In this case we say that f is computed by M.

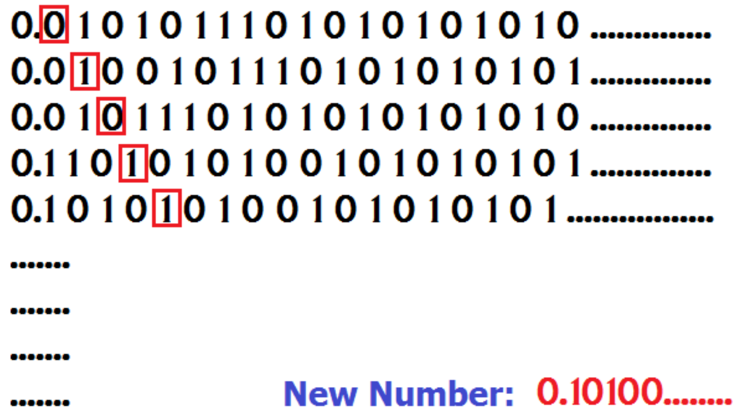
Countable set

The set of all Turing machines is countable.

0	$\to$	1
1	$\to$	2
00	$\to$	3
01	$\to$	4
10	$\to$	5
11	$\to$	6
000	$\to$	7

$$\aleph_0$$

The set of all functions of type $A^* \to A^*$ is uncountable.



$0.\boxed{0}\,1\,0\,1\,0\,1\,1\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0$ ....

$0.0\,\boxed{1}\,0\,0\,1\,0\,1\,1\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1$ ....

$0.0\,1\,\boxed{0}\,1\,1\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0$ ....

$0.1\,1\,0\,\boxed{1}\,0\,1\,0\,1\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1$ ....

$0.1\,0\,1\,0\,\boxed{1}\,0\,1\,0\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1$ ....

....

....

....

....

New Number: 0.10100........

Predicate

A function with Boolean values: 1 (true) or 0 (false). A predicate P corresponds to the set $\{x \colon P(x)\}$, i.e., the set of strings x for which $P(x)$ is true. Subsets of A^* are also called languages over A.

Formal languages, e.g., recognizable by finite automata, can be called predicates as well.

Predicate P is a function $A^* \to \{0,1\}$.

Decidable predicate

A predicate P is decidable if there exists a Turing machine that answers question "is $P(\alpha)$ true?" for any $\alpha \in A^*$, giving either 1 (yes) or 0 (no). Note that this machine must terminate for any $\alpha \in A^*$.

Changing notions

The notions of a computable function and a decidable predicate can be extended to functions and predicates in several variables in a natural way.

$\phi_{M,n}(\alpha_1, \ldots, \alpha_n)$ = output of M for the input $\alpha_1 \# \alpha_2 \# \cdots \# \alpha_n$.

Computable - definitions

A partial function f from $(A^*)^n$ to A^* is computable if there is a Turing machine M such that $\phi_{\mathrm{M,n}} = f$.

The definition of a decidable predicate can be given in the same way.

Computational complexity

We say that a Turing machine works in time $T(n)$ if it performs at most $T(n)$ steps for any input of size n. Analogously, a Turing machine M works in space $s(n)$ if it visits at most $s(n)$ cells for any computation on inputs of size n.

Time – number of computational steps.

Space – number of cells visited by a TM.

Turing’s thesis and universal machines

Turing thesis

Any algorithm can be realized by a Turing machine.

The Church-Turing thesis is not a mathematical theorem, but rather a statement about our informal notion of algorithm, or the physical reality this notion is based upon.

Cannot be proved, but it is supported by empirical evidence.

Different definitions of algorithm

Turing machine, Post machine, Church’s lambda-calculus, Gödel's theory of recursive functions – turned out to be equivalent.

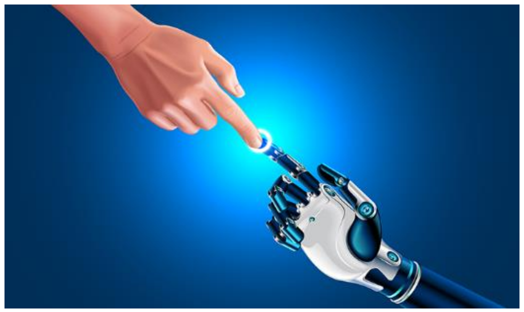
Capabilities of TMs

Example – human instead of a TM. A person with a restricted memory, pencil, eraser, and a notebook with an infinite number of pages.

Pages are of fixed size - finitely many variants of filling a single page (e.g., symbol of alphabet).

The person can work with one page at a time but can then move to the previous or to the next page.

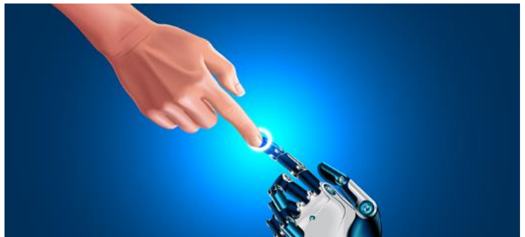
When turning a page, the person has a finite amount of information in the mind.



Process of TM

The input string is written on several first pages of the notebook (one letter per page); the output should be written in a similar way.

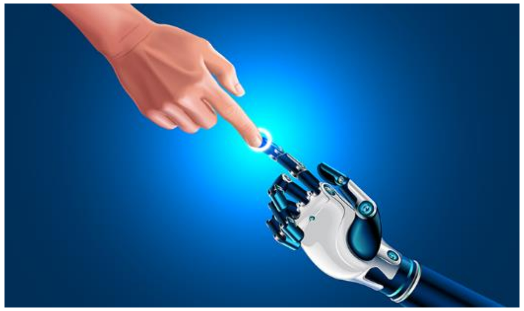
The computation terminates when the person closes the notebook or when the person does not know what to do next.



Recreating TM

The amount of data that a person remembers is limited, but we can store unlimited amount of data on the tape. We can remember something, but can also reference information on the tape.

E.g., auxiliary strings can be processed by "subroutines". In particular, auxiliary strings can be used to implement counters. Using counters, we can address a memory cell by its number.



Extending alphabet

In the definition of computability for functions of type $A^* \to A^*$ we restrict neither the number of auxiliary tape symbols (the set of symbols S could be much bigger than A) nor the number of states. One auxiliary symbol (the blank) is enough. We can represent each letter from $S \backslash A$ by a combination of blanks and nonblanks.

How? We can use more than one cell as one symbol, and remember more cells (by increasing the number of internal states).

TM as a string

Fact 1.1. All Turing machines with arbitrary numbers of states and alphabets of any size can be encoded by strings over a fixed alphabet.

Additional thought about this fact – we can describe any Turing machine on our computer. Computer uses some encoding, which is later translated into machine code that consists of bits. Therefore, there is always a way to translate description of a TM into a binary string.

More details on pages 12 and 13 of the Book.

Universal TM

TM is a finite object, and it can be encoded by a string. We can consider a universal TM U. Its input is a pair $([M], x)$, where $[M]$ is the encoding of a machine M with external alphabet A, and x is a string over A.

The output of U is $\phi_{\mathrm{M}}(x)$. Thus U computes the function u defined as follows:

$$u([M], x) = \phi_{\mathrm{M}}(x)$$

Universal function

The function $u([M], x) = \phi_{\mathrm{M}}(x)$ is universal for the class of computable functions of type $A^* \to A^*$ in the following sense: for any computable function $f: A^* \to A^*$, there exists some M such that $u([M], x) = f(x)$ for all $x \in A^*$.

Universal as a consequence

The existence of a universal machine U is a consequence of the Church-Turing thesis since our description of Turing machines was algorithmic. But, unlike the Church-Turing thesis, this is also a mathematical theorem. The construction is straightforward but boring.

Complexity classes

Motivation

An algorithm computing a function can require too much time or space. So from the practical viewpoint we are interested in effective algorithms. The idea of an effective algorithm can be formalized in different ways, leading to different complexity classes.

Polynomial growth

A function $f(n)$ is of polynomial growth if $f(n) \leq cn^d$ for some constants c, d and for all sufficiently large n.

$$f(n) = poly(n)$$

Poly-time computation

A function F on B^* is computable in polynomial time if there exists a Turing machine that computes it in time $T(n) = poly(n)$, where n is the length of the input.

If F is a predicate, we say that it is decidable in polynomial time.

$B = \{0,1\}$; B^* - binary string.

Class P

The class of all functions computable in polynomial time, or all predicates decidable in polynomial time, is denoted by P. If F is computable in polynomial time, then $|F(x)| = poly(|x|)$.

$|t|$ stands for the length of the string t

P – theoretic notion

If the degree of the polynomial is large (or the constant c is large), an algorithm running in polynomial time may be quite impractical.

n	n^{100}
1	1
2	10^{30}
5	$> 10^{69}$
10	10^{100}

Other models, e.g., we may use a usual programming language dealing with integer variables.

Importance of the encoding

It is easy to see that the predicate that is true for all unary representations of prime numbers is polynomial.

The obvious algorithm that tries to divide N by all numbers $\leq \sqrt{N}$.

On the other hand, we do not know whether the predicate P(x) = "x is a binary representation of a prime number" is polynomial or not.

Unary strings - 11...11. For binary - poly(n), where $n = \lfloor \log_2 N \rfloor$.

Class PSPACE

A function (predicate) F on B^* is computable (decidable) in polynomial space if there exists a Turing machine that computes F and runs in space $s(n) = poly(n)$, where n is the length of the input.

Class PSPACE

Note that any machine that runs in polynomial time also runs in polynomial space, therefore P $\subseteq$ PSPACE. Most experts believe that this inclusion is strict, i.e., P $\neq$ PSPACE, although nobody has succeeded in proving it so far. This is a famous open problem.

Halting problem

There is no algorithm that determines, for given Turing machine and input string, whether the machine terminates at that input or not.

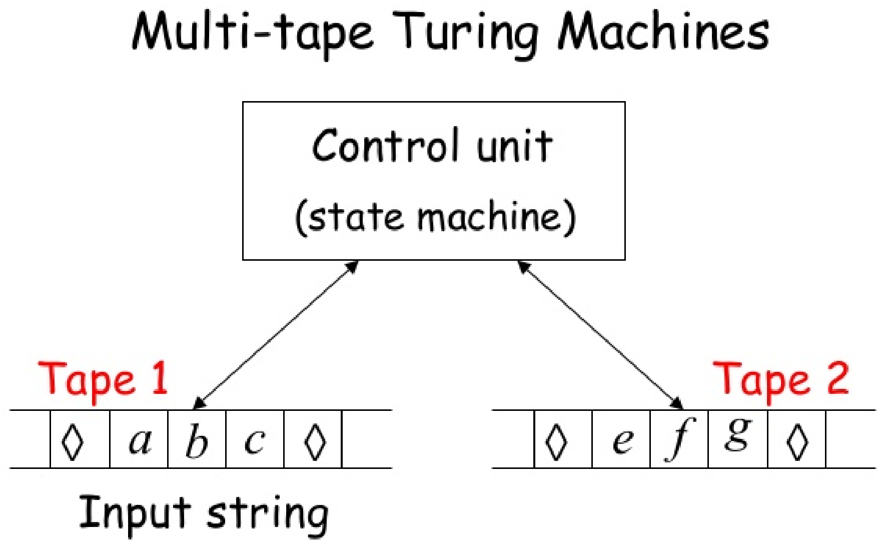
Enumeration

Enumeration is a process which produces one element of a set after another so that every element is included in this list.

Exact definition: a set $X \subseteq A^*$ is called enumerable if it is the set of all possible outputs of some (including universal) Turing machine E.

Extending TM

For example, one can consider multitape Turing machines that have a finite number of tapes. Each tape has its own head that can read and write symbols on the tape.



TM demo

Source: https://turingmachinesimulator.com/

Here you can check different examples to see different TMs in action.

Multitape TM properties

Transition function takes all tapes into consideration – contents of cell of each tape affects the transition, heads change contents of the cell of the work tapes.

More tapes allow Turing machine to work faster; however, the difference is not so great.

Multitape TM properties

2-tape TM working in time $T(n)$ can be simulated by 1-tape TM in time $O(T^2(n))$.

3-tape TM working in time $T(n)$ can be simulated by 2-tape TM in time $O(T(n) log T(n))$.

Overall, k-tape TM with time $T(n)$ can be simulated by 1-tape TM in time $O(T^2(n))$.

Thank you for your attention!