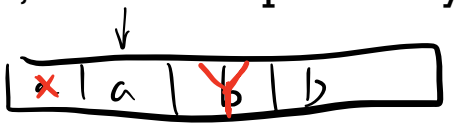
TM DESIGN

COMP 4200 – Formal Language

Design a TM to accept the language L = $\{a^n b^n, n \geq 1\}$. Show an ID for the string 'aaabbb' with tape symbols.

When the leftmost 'a' is traversed, that 'a' is replaced by X and the head moves to one right.

$$\delta(q0, a) \rightarrow (q1, X, R)$$



Then, the machine needs to search for the leftmost 'b'. Before that 'b', there exist (n – 1) numbers of 'a'. Those 'a' are traversed by

loop $\delta(q1, a) \rightarrow (q1, a, R)$

When the leftmost 'b' is traversed, the state is q1. That 'b' is replaced by Y, the state is changed to q2, and the head is moved to one left. The transitional functional is

$$\delta(q1, b) \rightarrow (q2, Y, L)$$

Then, it needs to search for the second 'a' starting from the left. The first 'a' is replaced by X, which means the second 'a' exists after X. So, it needs to search for the rightmost 'X'. After traversing the leftmost 'b', the head moves to the left to find the rightmost X. Before that, it has to traverse 'a'. The transitional function is

$$\delta(q2, a) \rightarrow (q2, a, L)$$

After traversing all the ‘a’ we get the rightmost ‘X’. Traversing the X the machine changes its state from q2 to q0 and the head moves to one right. The transitional function is

$$\delta(q2, X) \rightarrow (q0, X, R)$$

Similarly, after traversing ‘b’, the machine has to traverse some Y to get the rightmost ‘X’.

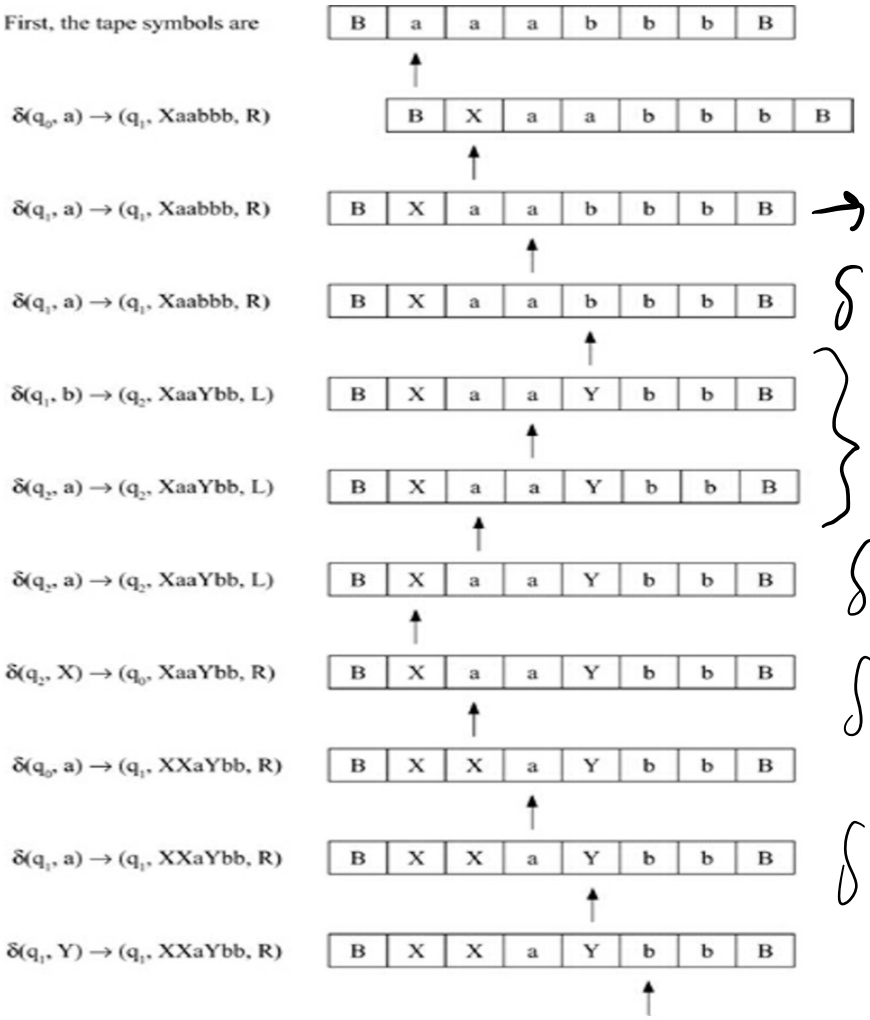
$$\delta(q2, Y) \rightarrow (q2, Y, L)$$

When all the ‘a’s are traversed, the state is q0, because before that state was q2 and the input was X. The head moves to one right and gets a Y. Getting a Y means that all the ‘a’s are traversed and the same number of ‘b’s are traversed. Traversing right, if at last the machine gets no ‘b’ but a blank ‘B’, then the machine halts. The transitional functions are

$$\delta(q0, Y) \rightarrow (q3, Y, R)$$

$$\delta(q3, Y) \rightarrow (q3, Y, R)$$

$$\delta(q3, B) \rightarrow (q4, B, H)$$



$\delta(q_0, a) = (q_1, X, R)$

$\delta(q_1, a) = (q_1, a, R)$ do nothing

$\delta(q_1, b) = (q_2, Y, L)$

$\delta(q_2, a) = (q_2, a, L)$

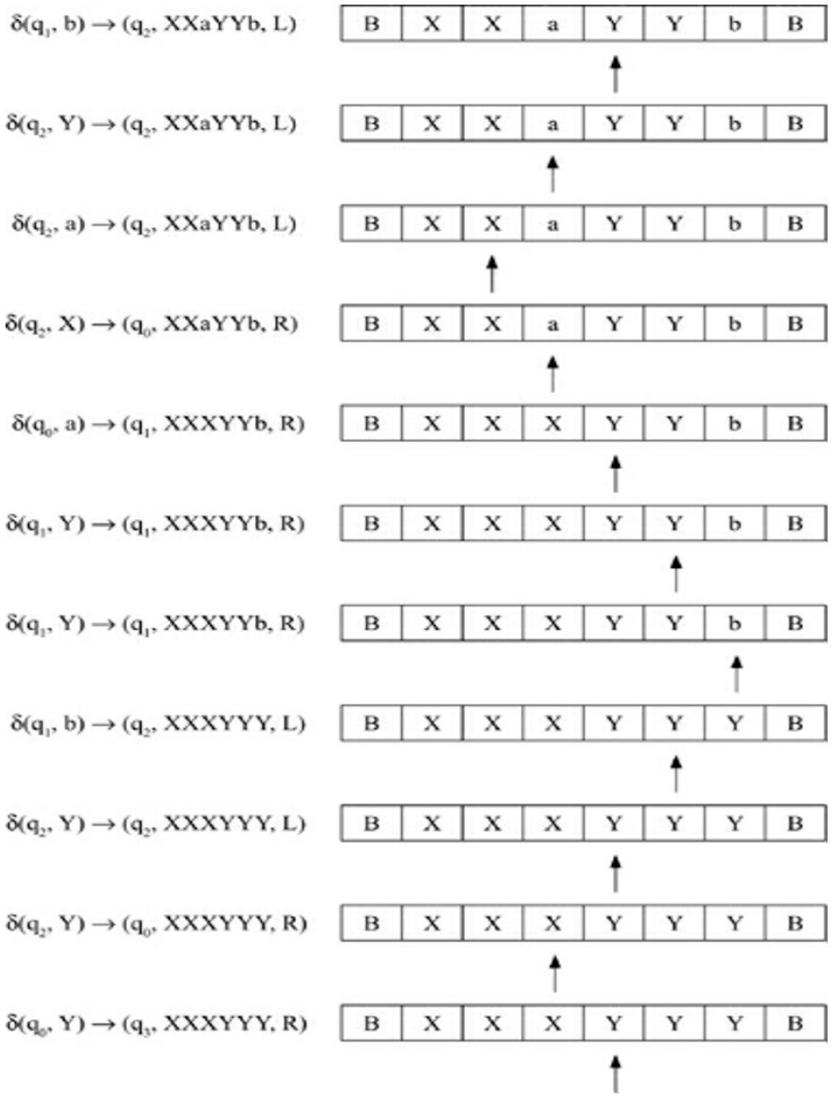
ID for the String 'aaabbb'

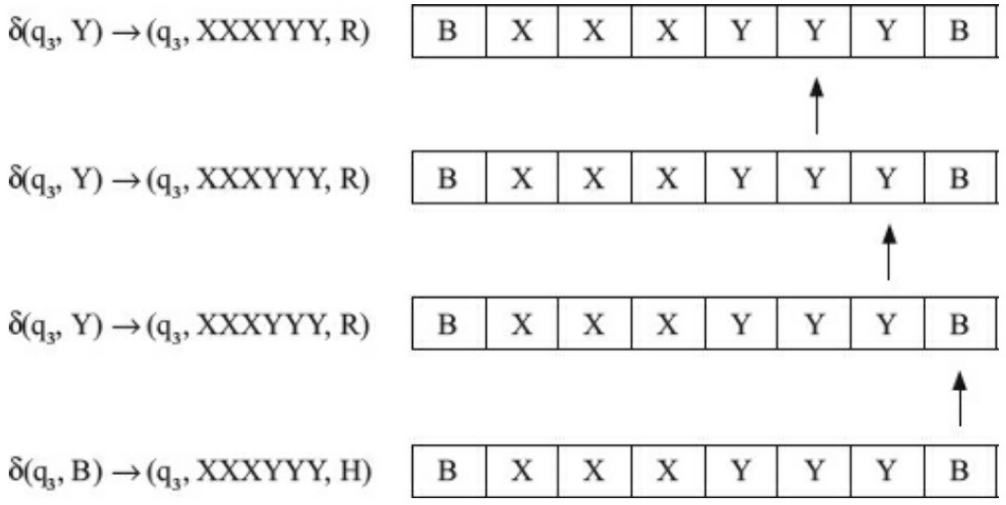
$\delta(q_2, X) = (q_0, X, R)$

$\delta(q_2, Y) = \delta(q_2, Y, R)$

$\delta(q_2\ b) = (q_2, Y, L)$

all a's & all b's match



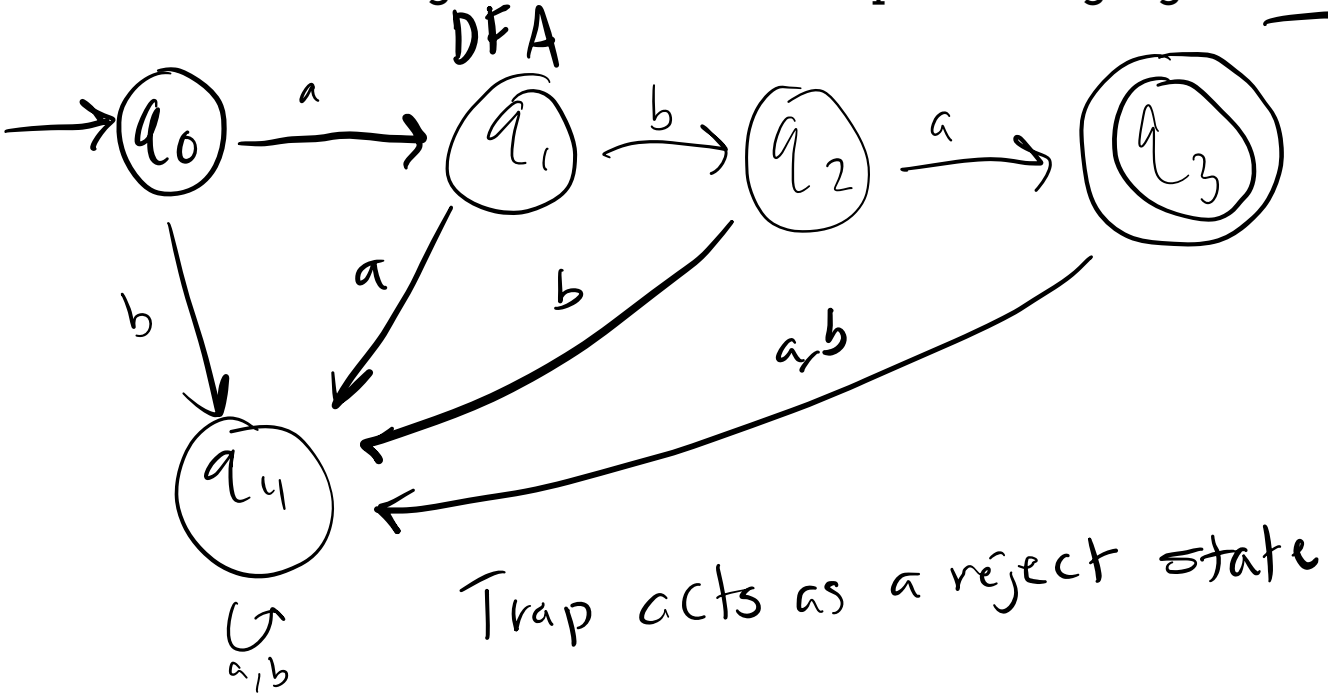


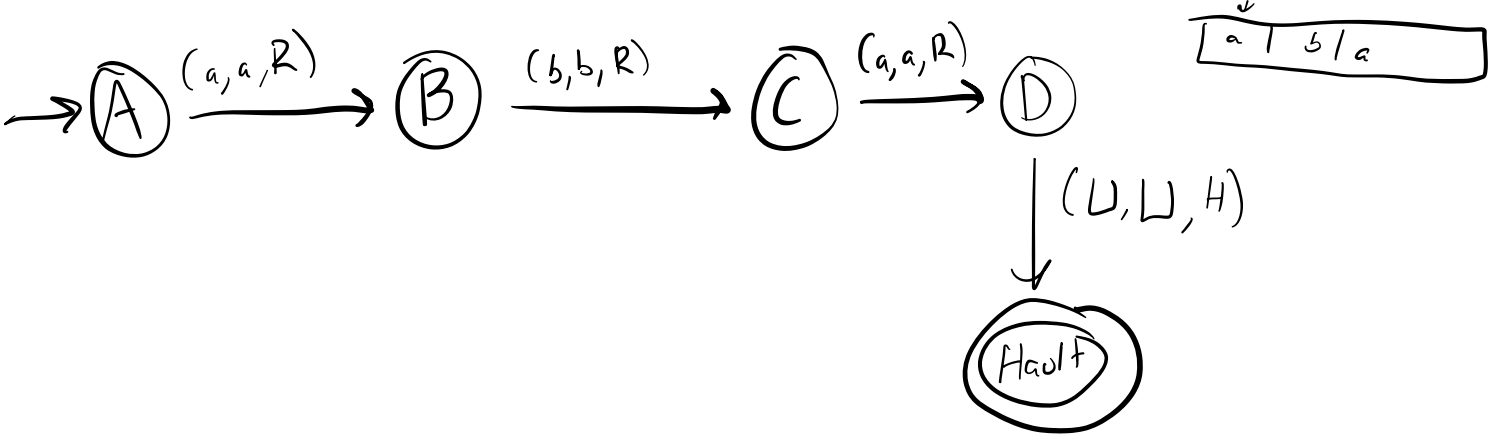
$\delta(q_3, B) = (q_3, B, H)$

ID for the String 'aaabbb'

EXAMPLE

Construct a turing machine which accepts the language of aba over {a b}.

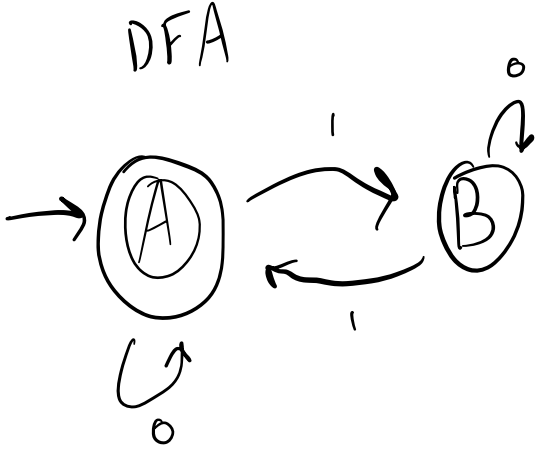


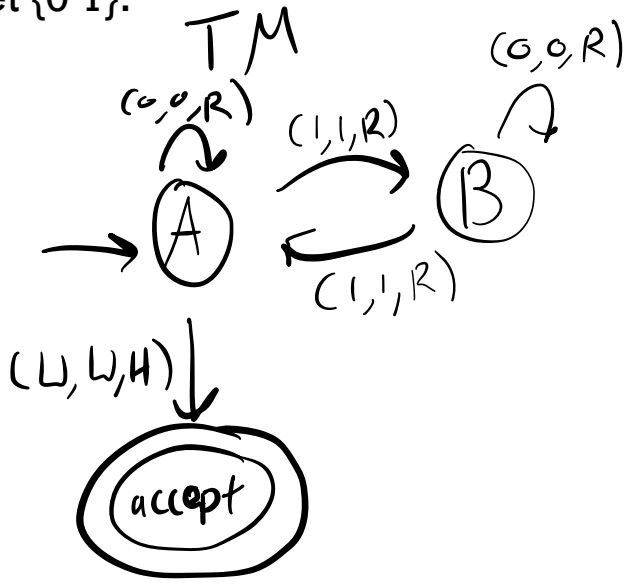


Don't have to show reject states

EXAMPLE

Construct TM for language consisting of strings having any number of 0's and only even number of 1's over the input set {0 1}.





TRY YOURSELF!

- Design a TM that recognizes the language L = $\{0^n 1^n \mid n>0\}$

␣ 0011 ␣

$\delta(q_0, 0) = (q_1, X, R)$

$\delta(q_1, 0) = (q_1, 0, R)$

$\delta(q_1, 1) = (q_2, Y, L)$

$\delta(q_2, 0) = (q_2, 0, L)$

$\delta(q_2, X) = (q_0, X, R)$

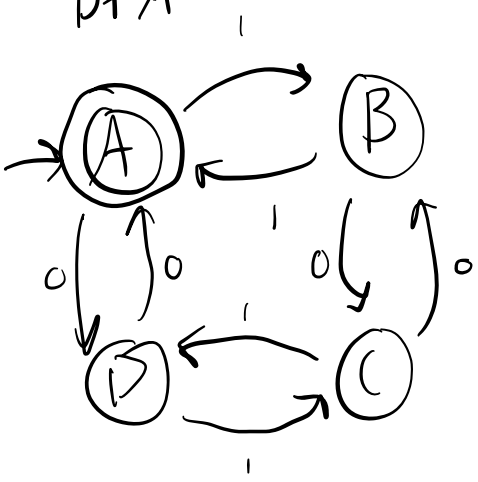
$\delta(q_1, Y) = (q_1, Y, R)$

$\delta(q_0, Y) = (q_0, Y, R)$

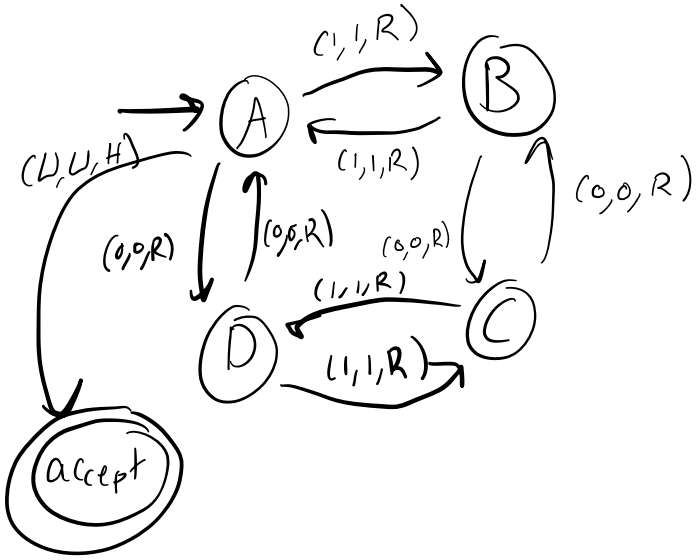
$\delta(q_0, ␣) = (q_3, ␣, H)$

Even # of 1's & Even # of 0's

DFA



TM



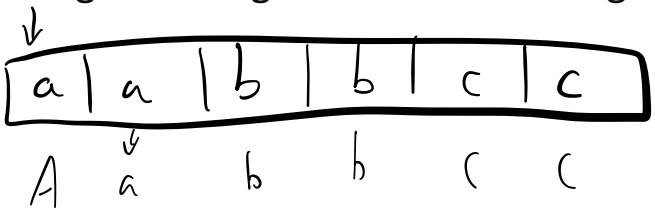
EXAMPLE

Construct Turing Machine for L = $\{a^n b^n a^n \mid n \geq 1\}$ Solution: The transition

3 variables

EXAMPLE

Design a Turing machine that recognizes whether a given input string is in the form of $a^n b^n c^n$



a to A move right

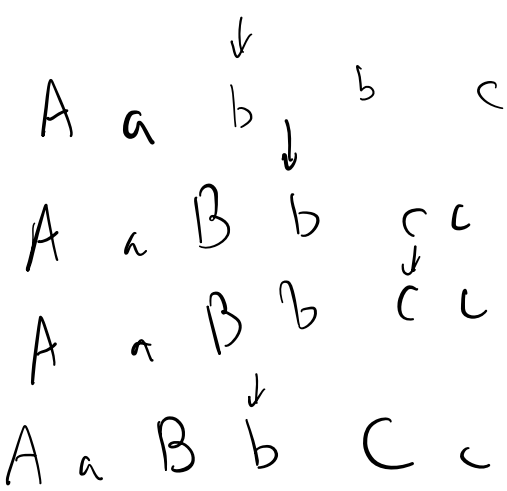
move right up to c

convert b to B move right

move right

convert c to C move left

move left



A a B b C c move left

A a B b C c move left

A a B b C c move right

A a B b C c a to A move right

A A B b C c move right

A A B b C c b to B move right

A A B B C c move right

A A B B C c C to C move left

A A B B C C move left

A A B B C C move left

A A B B C C move left

A A B B C C move right

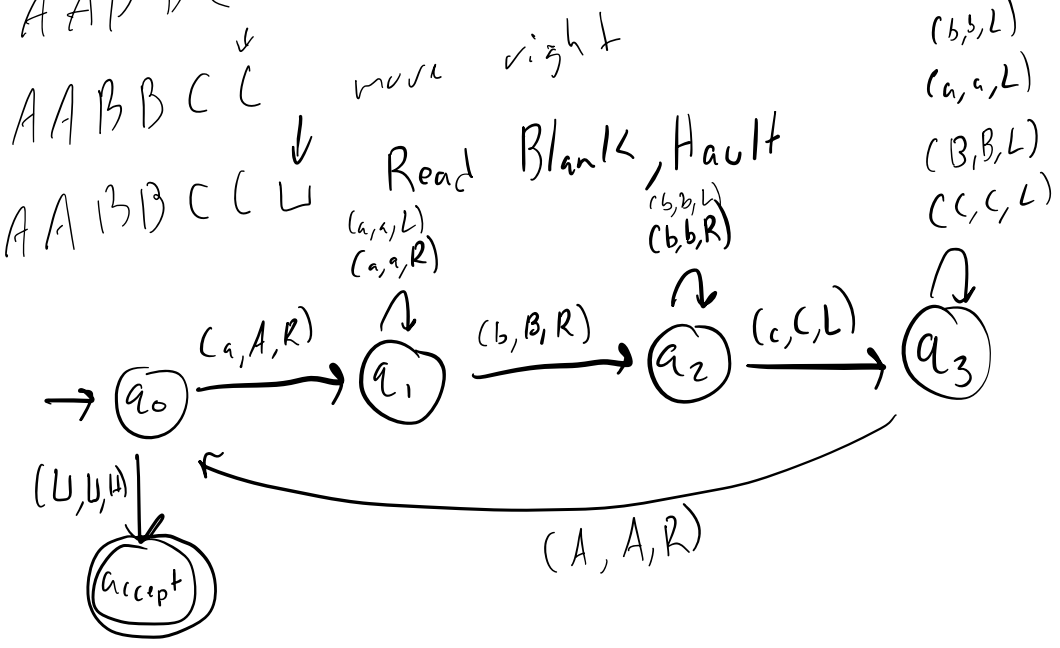
A A B B C C move right

A A B B C C move right

A A B B C C move right

A A B B C C move right

A A B B C C L Read Blank, Hault



Turing machine M_3 to decide language $C = \{ a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1 \}$.

Idea: If i collections of j things each, then i × j things total.

TM: for each 'a', cross off j c's by matching each b with a 'c'.

M_3 = "On input string w:

- Scan the input from left to right to make sure that it is a member of L(a∗b∗c∗) and reject if it isn't.
- Return the head to the left-hand end of the tape
- Cross off an a and scan to the right until a b occurs. Shuttle between the b's and the c's, crossing off each until all b's are gone. If all c's have been crossed off and some b's remain, reject.
- Restore the crossed off b's and repeat stage 3 if there is another a to cross off. If all a's are crossed off, check whether all c's also are crossed off. If yes, accept; otherwise, reject."

Clarity above all: high-level description of TMs is allowed

M = "On input string w:

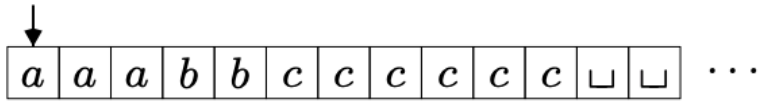
1. Scan input …"

but it should not be used as a trick to hide the important details of the program.

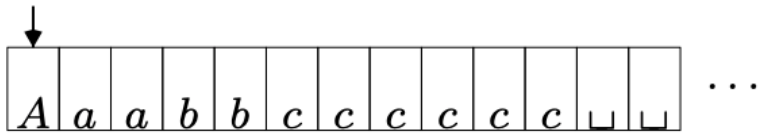
- Standard tools:
  - Expanding tape alphabet Γ with
    - separator "#"
    - dotted symbols $\dot{0}$, $\dot{a}$, to indicate "activity,"
    - Typical example: $\Gamma = \{0, 1, \#, \sqcup, \dot{0}, \dot{1}\}$

Running TM M_3 on Input $a^3b^2c^6 \in C$

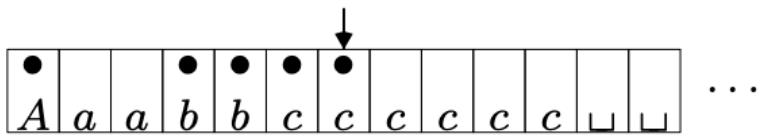
- Tape head starts over leftmost symbol



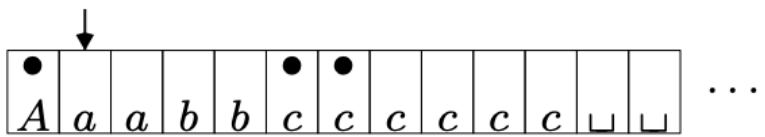
- Stage 1: Mark leftmost symbol and scan to see if input $\in L(a^*b^*c^*)$



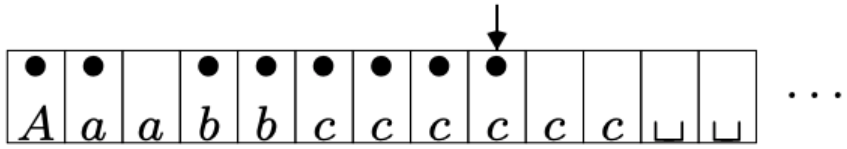
- Stage 3: Cross off one a and cross off matching b's and c's



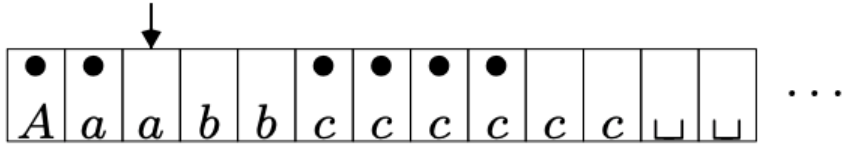
- Stage 4: Restore b's and return head to first a not crossed off



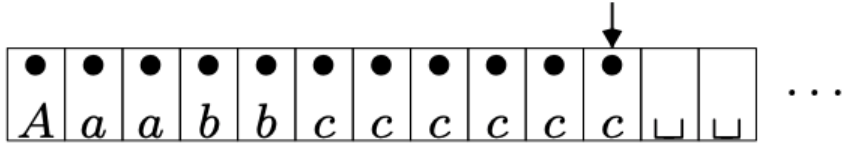
- Stage 3: Cross off one a and cross off matching b's and c's



- Stage 4: Restore b's and return head to first a not crossed off



- Stage 3: Cross off one a and cross off matching b's and c's

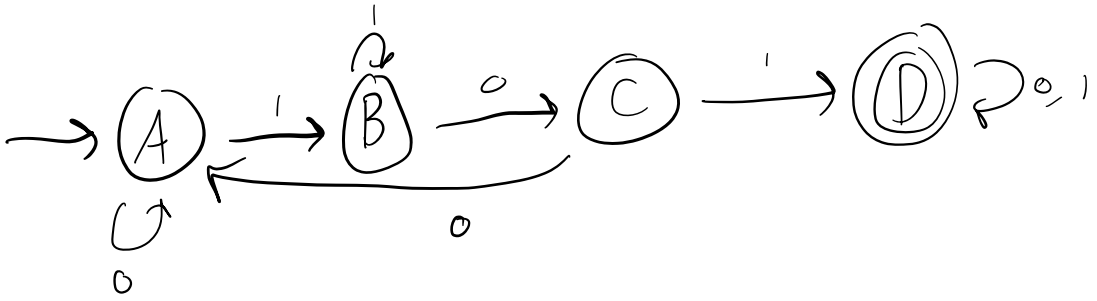


- Stage 4: If all a's crossed off, check if all c's crossed off.
- *accept*

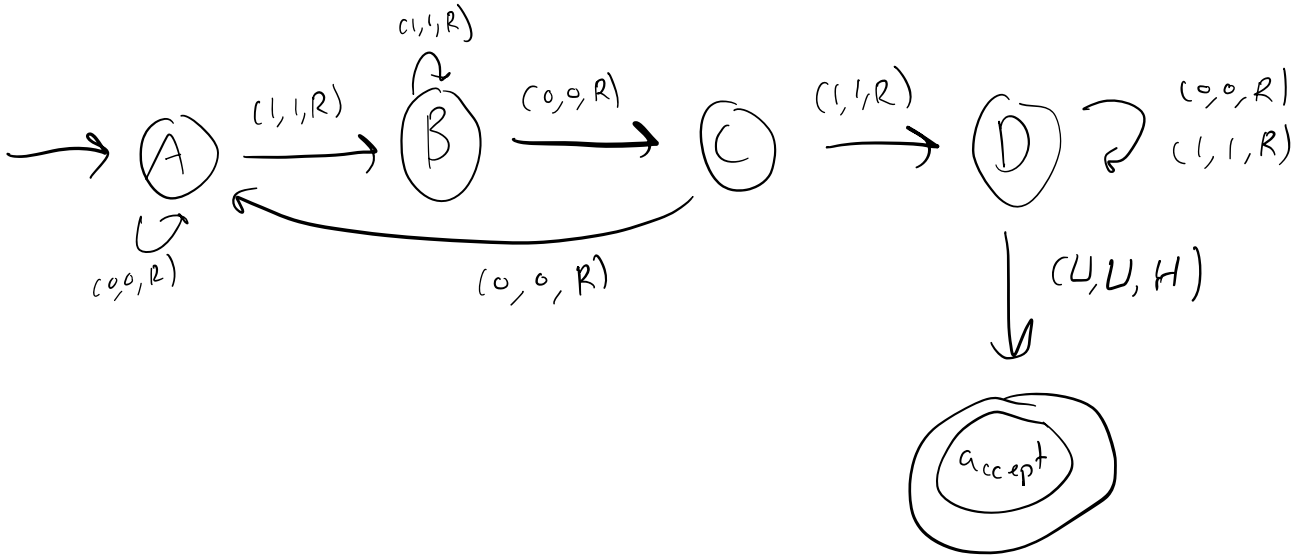
How to tell when a TM is at the left end of the tape?

- One Approach: Mark it with a special symbol.
- Alternative method:
  - remember current symbol
  - overwrite it with special symbol
  - move left
  - if special symbol still there, head is at start of tape
  - otherwise, restore previous symbol and move left.

Substring "101" DFA



TM



VARIANTS OF TM

- Multitape TM or Non-deterministic TM
- The original model and its reasonable variants all have the same power—they recognize the same class of languages.
- There are multiple ways to check the robustness of TM, one way is to change the transition function by including the tape head to Stay put.
- Might this feature allow Turing machines to recognize additional languages, thus adding to the power of the model? Of course not, because we can convert any TM with the "stay put" feature to one that does not have it.
- This small change shows how TM are robust in nature.

MULTI-TAPE TM

- TM with more than one tape.
- Each tape has its own head for reading and writing.
- Initially the input appears on tape 1, and the others start out blank.
- The transition function is changed to allow for reading, writing, and moving the heads on some or all of the tapes simultaneously.

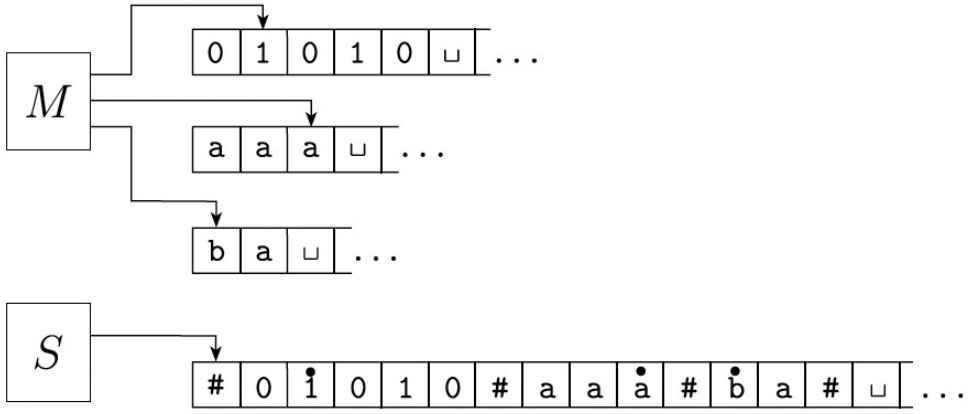
$$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}^k$$

where k is the number of tapes.

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \mathrm{L}, \mathrm{R}, \ldots, \mathrm{L})$$

means that if the machine is in state qi and heads 1 through k are reading symbols a1 through ak, the machine goes to state qj, writes symbols b1 through bk, and directs each head to move left or right, or to stay put, as specified.

Multi - tape Turing machines appear to be more powerful than ordinary Turing machines, but we can show that they are equivalent in power.



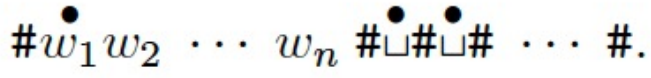
Every multitape Turing machine has an equivalent single-tape Turing machine.

- Say that M has k tapes.
- Then S simulates the effect of k tapes by storing their information on its single tape.
- It uses the new symbol # as a delimiter to separate the contents of the different tapes.
- In addition to the contents of these tapes, S must keep track of the locations of the heads.
- It does so by writing a tape symbol with a dot above it to mark the place where the head on that tape would be.
- Think of these as "virtual" tapes and heads.
- As before, the "dotted" tape symbols are simply new symbols that have been added to the tape alphabet.

S = “On input w = w1 · · · wn:

- First S puts its tape into the format that represents all k tapes

of M. The formatted tape contains



To simulate a single move, S scans its tape from the first #,

- which marks the left-hand end, to the (k + 1)st #, which marks the right-hand end, in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to the way that M’s transition function dictates.
- If at any point S moves one of the virtual heads to the right onto a #, this action signifies that M has moved the corresponding head onto the previously unread blank portion of that tape. So S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost #, one unit to the right. Then it continues the simulation as before.”

NONDETERMINISTIC TURING MACHINES

- A non-deterministic Turing machine is defined in the expected way. At any point in a computation, the machine may proceed according to several possibilities.
- The transition function for a nondeterministic Turing machine has the form

$$\delta: Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}).$$

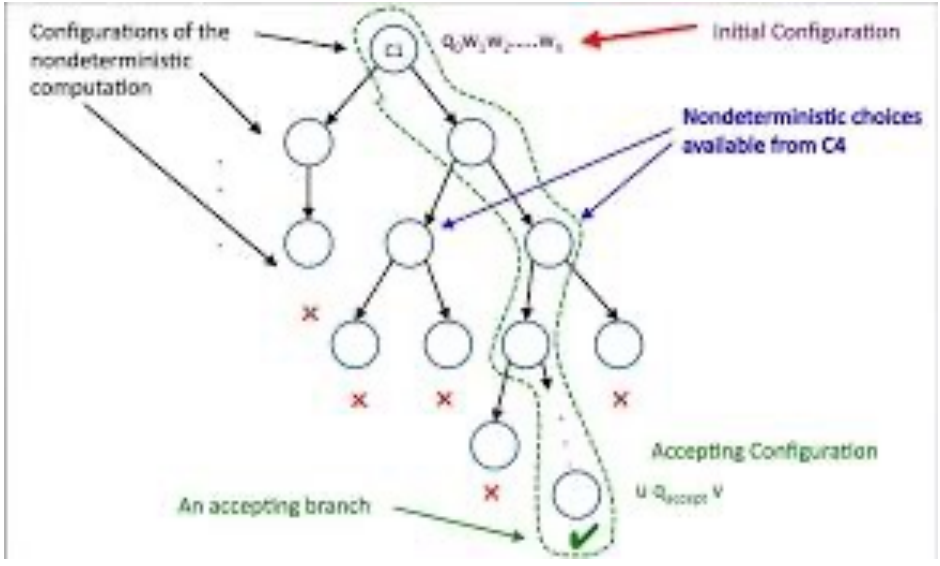
- The computation of a nondeterministic Turing machine is a tree whose branches correspond to different possibilities for the machine.
- If some branch of the computation leads to the accept state, the machine accepts its input.

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

Idea:

- We can simulate any nondeterministic TM N with a deterministic TM D.
- The idea behind the simulation is to have D try all possible branches of N's nondeterministic computation.
- If D ever finds the accept state on one of these branches, D accepts.
- Otherwise, D's simulation will not terminate.

- We represent N's computation on an input w as a tree.
- Each branch of the tree represents one of the branches of the nondeterminism.
- Each node of the tree is a configuration of N.
- The root of the tree is the start configuration.
- The TM D searches this tree for an accepting configuration.
- What type of search will we use?



MORE ON TM

- The depth-first search strategy goes all the way down one branch before backing up to explore other branches.
- If D were to explore the tree in this manner, D could go forever down one infinite branch and miss an accepting configuration on some other branch.
- So best way to explore the tree by using breadth-first search.
- This strategy explores all branches to the same depth before going on to explore any branch to the next depth.
- This method guarantees that D will visit every node in the tree until it encounters an accepting configuration.

Every nondeterministic Turing machine has an equivalent deterministic Turing machine.

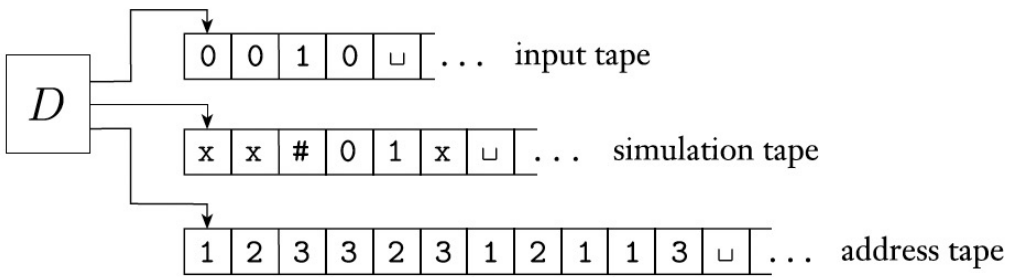
The simulating deterministic TM D has three tapes. The arrangement is equivalent to having a single tape.

The machine D uses its three tapes in a particular way.

Tape 1 always contains the input string and is never altered.

Tape 2 maintains a copy of N's tape on some branch of its nondeterministic computation.

Tape 3 keeps track of D's location in N's nondeterministic computation tree.



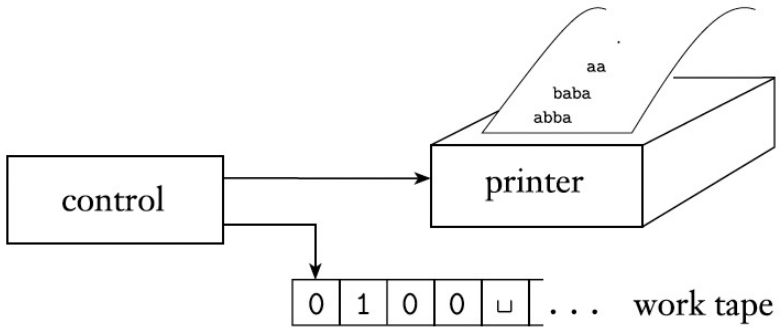
- Consider the data representation on tape 3. Every node in the tree can have at most b children, where b is the size of the largest set of possible choices given by N's transition function.
- To every node in the tree we assign an address that is a string over the alphabet $\Gamma_b = \{1, 2, \ldots, b\}$.
- We assign the address 231 to the node we arrive at by starting at the root, going to its 2nd child, going to that node's 3rd child, and finally going to that node's 1st child.
- Each symbol in the string tells us which choice to make next when simulating a step in one branch in N's nondeterministic computation.
- Sometimes a symbol may not correspond to any choice if too few choices are available for a configuration.
- In that case, the address is invalid and doesn't correspond to any node.
- Tape 3 contains a string over Γ_b. It represents the branch of N's computation from the root to the node addressed by that string unless the address is invalid.
- The empty string is the address of the root of the tree.

Working of D:

- Initially, tape 1 contains the input w, and tapes 2 and 3 are empty.
- Copy tape 1 to tape 2 and initialize the string on tape 3 to be ε.
- Use tape 2 to simulate N with input w on one branch of its nondeterministic computation. Before each step of N, consult the next symbol on tape 3 to determine which choice to make among those allowed by N's transition function.
- If no more symbols remain on tape 3 or if this nondeterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, accept the input.
- Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of N's computation by going to stage 2.

ENUMERATORS

- The term, recursively enumerable language originates from a type of Turing machine variant called an enumerator.
- Loosely defined, an enumerator is a Turing machine with an attached printer.
- The Turing machine can use that printer as an output device to print strings.
- Every time the Turing machine wants to add a string to the list, it sends the string to the printer.



ENUMERATORS

- An enumerator E starts with a blank input on its work tape.
- If the enumerator doesn't halt, it may print an infinite list of strings.
- The language enumerated by E is the collection of all the strings that it eventually prints out.
- Moreover, E may generate the strings of the language in any order, possibly with repetitions.

A language is Turing-recognizable if and only if some enumerator enumerates it.

PROOF: First we show that if we have an enumerator E that enumerates a language A, a TM M recognizes A. The TM M works in the following way.

M = "On input w:

1. Run E. Every time that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept."

Clearly, M accepts those strings that appear on E's list.

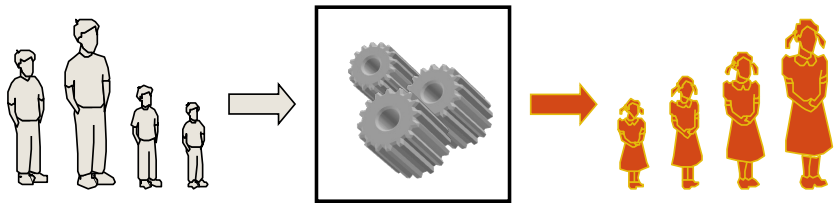
Now we do the other direction. If TM M recognizes a language A, we can construct the following enumerator E for A. Say that s1, s2, s3, . . . is a list of all possible strings in Σ^*.

E = "Ignore the input.

1. Repeat the following for i = 1, 2, 3, ...
2. Run M for i steps on each input, s1, s2, . . . , si.
3. If any computations accept, print out the corresponding sj ."

If M accepts a particular string s, eventually it will appear on the list generated by E. In fact, it will appear on the list infinitely many times because M runs from the beginning on each string for each repetition of step 1. This procedure gives the effect of running M in parallel on all possible input strings.

ALGORITHM

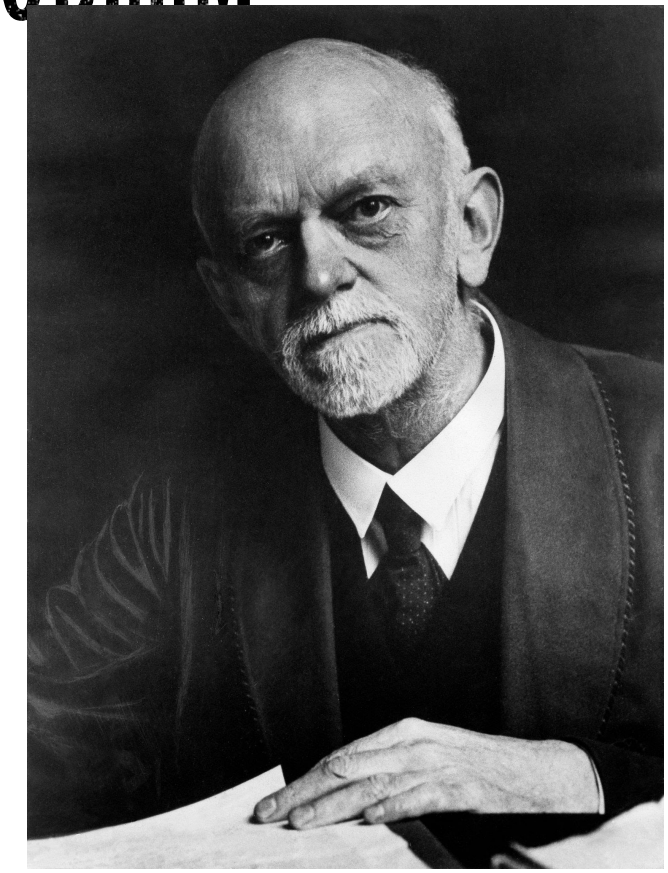


Input Algorithm Output

An **algorithm** is a step-by-step procedure for solving a problem in a finite amount of time.

HILBERT'S PROBLEM

- In 1900, mathematician David Hilbert delivered a now-famous address at the International Congress of Mathematicians in Paris.
- In his lecture, he identified 23 mathematical problems and posed them as a challenge for the coming century.
- The tenth problem on his list concerned algorithms.



HILBERT'S 10TH PROBLEM

- A polynomial is a sum of terms, where each term is a product of certain variables and a constant, called a coefficient. For example,

$$6 \cdot x \cdot x \cdot x \cdot y \cdot z \cdot z = 6x^3yz^2$$

is a term with coefficient 6, and

$$6x^3yz^2 + 3xy^2 - x^3 - 10$$

is a polynomial with four terms, over the variables x, y, and z.

For this discussion, we consider only coefficients that are integers.

HILBERT'S 10TH PROBLEM

- A root of a polynomial is an assignment of values to its variables so that the value of the polynomial is 0.
- This polynomial has a root at $x = 5$, $y = 3$, and $z = 0$.
- This root is an integral root because all the variables are assigned integer values.
- Some polynomials have an integral root and some do not.
- Hilbert's tenth problem was to devise an algorithm that tests whether a polynomial has an integral root.

HILBERT'S 10TH PROBLEM

- No algorithm exists for this task; it is algorithmically unsolvable.
- For mathematicians of that period to come to this conclusion with their intuitive concept of algorithm would have been virtually impossible.
- The intuitive concept may have been adequate for giving algorithms for certain tasks, but it was useless for showing that no algorithm exists for a particular task.
- Proving that an algorithm does not exist requires having a clear definition of algorithm.

CHURCH TURING THESIS

- The definition came in the 1936 papers of Alonzo Church and Alan Turing.
- Church used a notational system called the λ-calculus to define algorithms.
- Turing did it with his "machines."
- These two definitions were shown to be equivalent.
- This connection between the informal notion of algorithm and the precise definition has come to be called the Church–Turing thesis.

CHURCH TURING THESIS

- So, what is Church Turing Thesis in simple words?
  - It states that any function that can be computed algorithmically can be computed by a Turing machine, and vice versa.
  - The Church-Turing thesis suggests that any problem that can be solved by an algorithm can also be solved by a Turing machine, which serves as a theoretical model of a general-purpose computer.

HILBERT'S 10TH PROBLEM

Let D = {p| p is a polynomial with an integral root}

Hilbert's tenth problem → the set D is decidable. The answer is negative.

In contrast, we can show that D is Turing-recognizable.

Considering a simpler problem, polynomials that have only a single variable, such as $4x^3 - 2x^2 + x - 7$.

Let D1 = {p| p is a polynomial over x with an integral root}

Here is a TM M1 that recognizes D1:

M1 = "On input ⟨p⟩: where p is a polynomial over the variable x.

- Evaluate p with x set successively to the values 0, 1, −1, 2, −2, 3,

−3, ... If at any point the polynomial evaluates to 0, accept ."

- If p has an integral root,M1 eventually will find it and accept. If p does not have

an integral root, M1 will run forever.

For the multivariable case, we can present a similar TMM that recognizes D.

Here, M goes through all possible settings of its variables to integral values.

Both M1 and M are recognizers but not deciders.

We can convert M1 to be a decider for D1 because we can calculate bounds within which the roots of a single variable polynomial must lie and restrict the search to these bounds.

We can show that the roots of such a polynomial must lie between the values

$$\pm k \frac{c_{\max}}{c_1},$$

where k is the number of terms in the polynomial, c_{max} is the coefficient with the largest absolute value, and c_1 is the coefficient of the highest order term.

If a root is not found within these bounds, the machine rejects.

HOW DO WE DESCRIBE A TM?

What is the right level of detail to give when describing such algorithms?

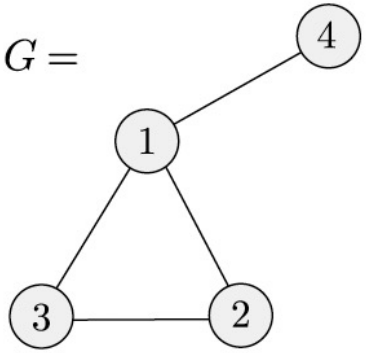
There are three possibilities.

- The first is the formal description that spells out in full the Turing machine's states, transition function, and so on.
- the lowest → most detailed level of description.
- higher level of description → implementation description, in which we use English prose to describe the way that the Turing machine moves its head and the way that it stores data on its tape. At this level we do not give details of states or transition function.
- The high-level description, we use English prose to describe an algorithm, ignoring the implementation details. At this level we do not need to mention how the machine manages its tape or head.

- The input to a Turing machine is always a string.
- If we want to provide an object other than a string as input, we must first represent that object as a string.
- Strings can easily represent polynomials, graphs, grammars, automata, and any combination
- of those objects.
- A Turing machine may be programmed to decode the representation so that it can be interpreted in the way we intend.
- Our notation for the encoding of an object O into its representation as a string is ⟨O⟩.
- If we have several objects O1,O2, . . .,Ok, we denote their encoding into a single
- string ⟨O1,O2, . . .,Ok⟩.
- The encoding itself can be done in many reasonable ways.

GRAPH ENCODING

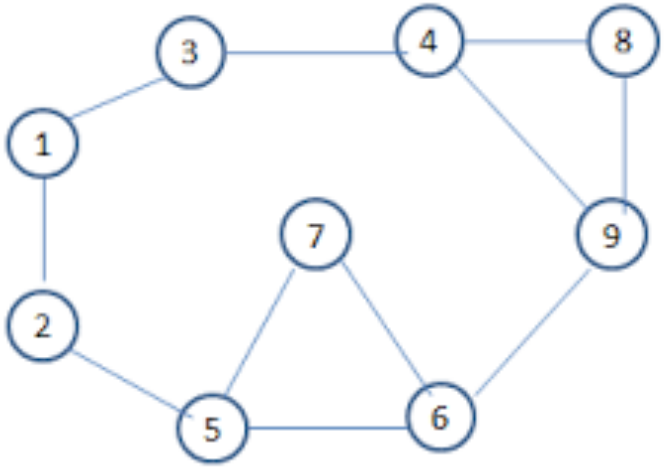
$G =$



$\langle G \rangle =$

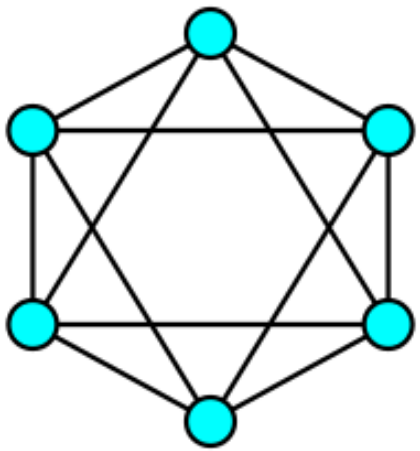
`(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`

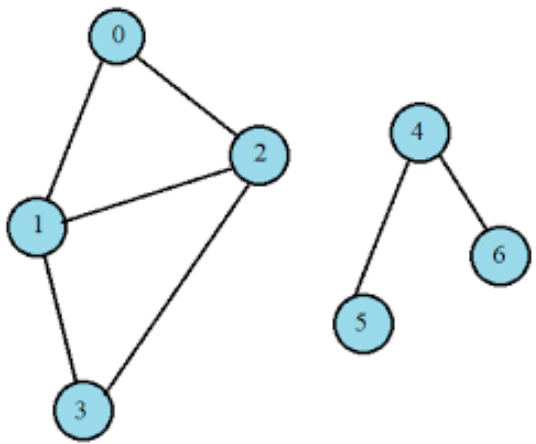
EXAMPLE



Let A be the language consisting of all strings representing undirected graphs that are connected. Here the graph is connected if every node can be reached from every other node by traveling along the edges of the graph.

A = {⟨G⟩ | G is a connected undirected graph}.





- A high-level description of a TM M that decides A.

M = "On input ⟨G⟩, the encoding of a graph G:

1. Select the first node of G and mark it.
2. Repeat the following stage until no new nodes are marked:
3. For each node in G, mark it if it is attached by an edge to a node that is already marked.
4. Scan all the nodes of G to determine whether they all are marked. If they are, accept; otherwise, reject ."