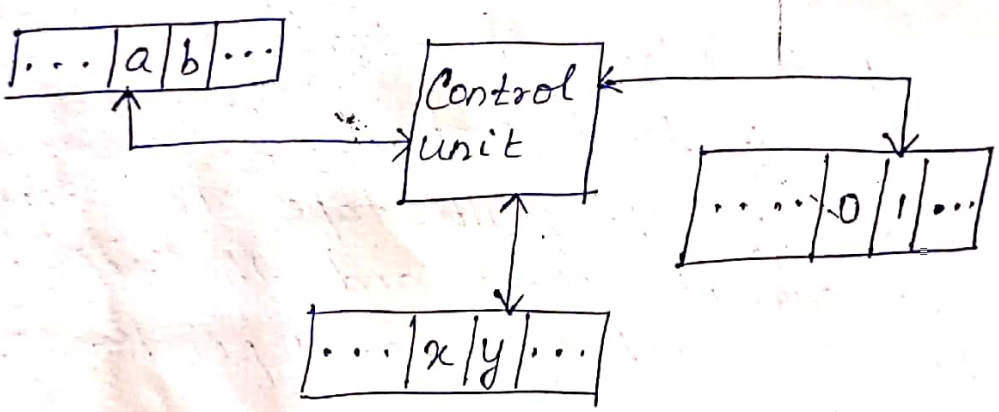
Module_5

Variants of Turing Machine

i) Multi tape Turing Machine

A multi-tape Turing Machine is nothing but a standard Turing Machine with more number of tapes. Each tape is controlled independently with independent read-write head. The pictorial representation of multi-tape turing machine is shown in figure below.



Various components of multi-tape TM are

a) Finite Control

b) Each tape have its own symbols and R/W head.

Each tape is divided into cells which can hold any symbol from the given alphabet. To start with TM should be in start state. If R/W head pointing to tape 1 moves towards right, the read/write head pointing

to tape 2 and tape 3 may move towards right or left depending on the transition.

Defn: The multi-tape TM is an n-tape m/c

$M = Q, \Sigma, \Gamma, \delta, q_0, B, F$

where

Q is set of finite states
Σ is i/p alphabets
Γ is tape symbols
δ is transition fn from $Q \times \Gamma^n$ to $Q \times \Gamma^n \times D$
q_0 is start state
B is Blank character
F is final state.

The move of the multi-tape TM depends on the current state and the scanned symbol by each of the tape heads.

For eg: If number of tapes in TM is 3 as shown in the fig and if there is a δ

$$\delta(q, a, b, c) = (p, x, y, z, L, R, S)$$

where q is the current state.

The transition can be interpreted as follows.

The TM in state q will be moved to state p only when the first read/write head points to a, the second R/W head to

to b and third R/W head to c and the R/W head will be moved to left in first case and right in second case and its stationary head in third case. At the same time, the symbols a, b and c will be replaced by x, y and z.

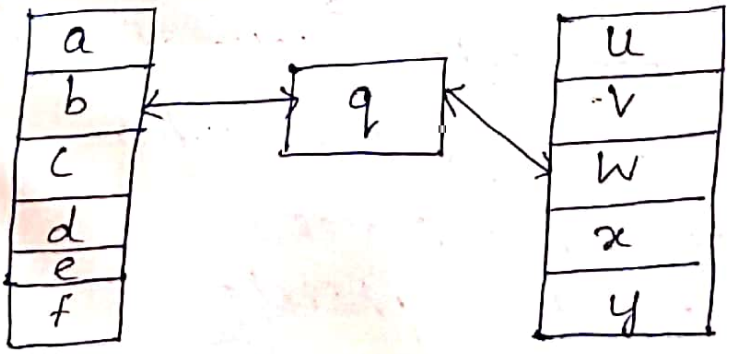
It can be easily shown that that the n-tape TM in fact is equivalent to the single tape standard TM as shown below.

Equivalence of Single tape and Multi-tape TM

Theorem: Every language accepted by a multi-tape TM is recursively enumerable.

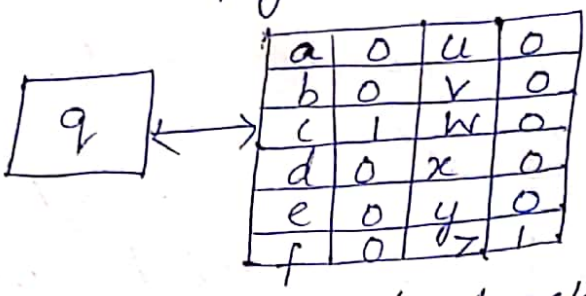
Proof: This can be shown by simulation.

For eg: Consider a TM with two tapes as shown below.



The above 2-tape TM can be simulated using single tape TM which has four tracks

as shown in fig below.



Single tape multiple tracks.

The first and third tracks consists of symbols from first and second tape respectively. The second and fourth track consists of the positions of the read/write head with respect to first and second tape.

So whatever transitions have been applied for multi-tape TM, if we apply the same transitions for the new m/c that we have constructed then the two m/cs are equivalent.

ii) Non deterministic Turing Machines:

The difference b/w non-deterministic TM and deterministic TM lies only in the definition of transition δ.

Defn: The non-deterministic Turing m/c $M = Q, \Sigma, \Gamma, \delta, q_0, B, F$ where

Q is set of finite states

Σ is set of i/p alphabets.

Γ is set of tape symbols.

δ is transition fn from $Q \times \Gamma$ to $2^{Q \times \Gamma \times (L,R)}$

$$\delta(q, X) = \{(q_1, X_1, D), (q_2, X_2, D) \ldots\ldots (q_i, X_i, D)\}$$

The m/c can choose any of the triples as the next move.

The language accepted by a non-deterministic TM is defined as follows.

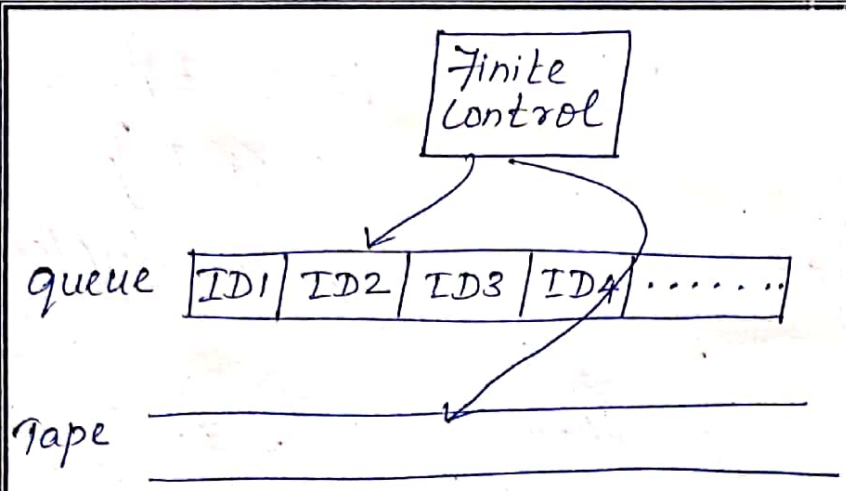
$$L(M) = \{w / q_0 w \vdash^* \alpha_1 p \alpha_2 \text{ where } w \in \Sigma, p \in F \text{ and } \alpha_1, \alpha_2 \in \Gamma^*\}$$

Theorem: For every NTM there exists a DTM such that $L(NTM) = L(DTM)$.

Proof: Given a string w, NTM starts at the initial configuration (initial ID) and goes through a sequence of IDs until it reaches one of the conditions:

*) final state is reached and the m/c halts
*) The transition is not defined and the m/c halts
*) Goes into an infinite loop.

The way NTM is simulated by DTM is shown using the figure shown below.



Each ID is associated with a state and the next I/P symbol.

It is clear from the fig, that all the IDs of NTM (represented by the nodes of the tree) are placed in the queue one after the other. Each ID inserted into the queue is also associated with the state and the next I/P symbol to be scanned. The first ID to be explored and examined will be at the front end of the queue.

Given any ID in the queue, all IDs to the left of designated ID are assumed to be deleted (marked) and the IDs towards right are assumed to be explored or to be examined.

The steps carried out by DTM are shown below:

1) The current ID is examined based on the state and the scanned symbol. If the state in the current ID is final state

then DTM accepts the string and the m/c halts.

2) If the state is non final state, the current ID is explored and various IDs obtained are inserted at the end of the queue.

3) Steps 1 and 2 are repeated until no more ID is examined or queue is empty.

The above steps can be clearly explained using tree representation.

The root node corresponds to the initial configuration (initial ID) and it is the only vertex of level 0. All the configurations (IDs) that are obtained by applying the transition fn of NTM only once will be the children of the initial configuration (ID). These new vertices which are derived from the root are at level 1. Since the configurations are finite, the number of children at various levels is finite.

The easiest way to simulate NTM using DTM is to traverse this tree using BFS from the root until the halt is reached.

It can be easily seen that

DTM accepts a string if and only if NTM accepts it. Thus any language accepted by a NTM is also accepted by a DTM.

The Model of Linear Bounded Automaton:

This model is important because
a) the set of context-sensitive languages is accepted by the model.
b) the infinite storage is restricted in size but not in accessibility to the storage in comparison with the TM model.

It is called the linear bounded Automaton (LBA) because a linear function is used to restrict (to bound) the length of the tape.

A linear bounded automaton is a nondeterministic TM which has a single tape whose length is not infinite but bounded by a linear function of the length of the i/p string. The models can be described by

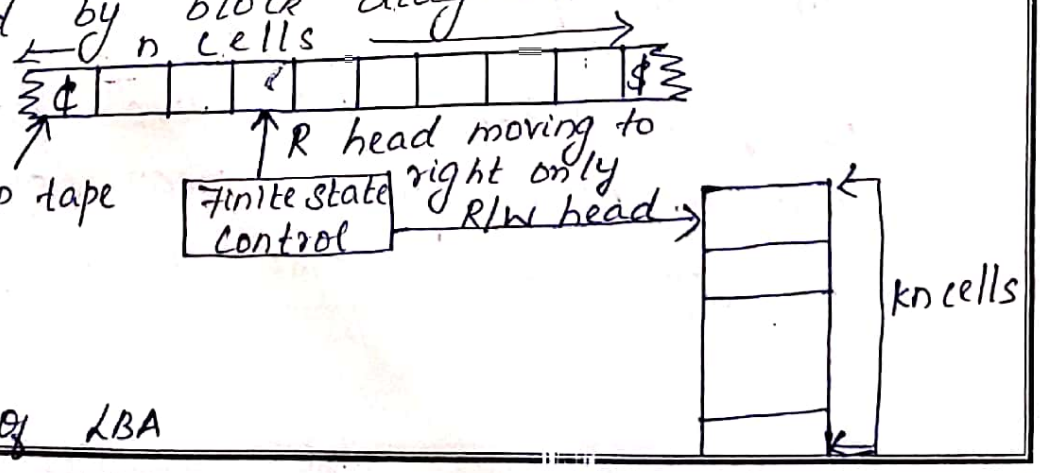
$$M = (Q, \Sigma, \Gamma, \delta, q_0, b, \phi, \$, F)$$

All the symbols have the same meaning as in the basic model of TM with the

difference that the i/p alphabet Σ contains two special symbols ¢ and $. ¢ is called the left-end marker which is entered in the left-most cell of the i/p tape and prevents the R/W head from getting off the left end of the tape.

$ is called the right-end marker which is entered in the right-most cell of the i/p tape and prevents the R/W head from getting off the right end of the tape. Bot the endmarkers should not appear on any other cell within the i/p tape, and the R/W head should not print any other symbol over both endmarkers.

Whenever we process any string in LBA, we shall assume that the i/p string is enclosed within the endmarkers ¢ and $. The above model of LBA can be represented by block diagram



Model of LBA

There are two tapes :

- input tape
- working tape

On the input tape head never prints and never moves to the left.

On the working tape the head can modify the contents in any way, without any restriction.

In case of LBA, an ID is denoted by (q, w, k) where $q \in Q$, $w \in \Gamma$ and k is some integer between 1 and n.

The transition of IDs is similar except that k changes to $k-1$ if the R/W head moves to the left and $k+1$ if the head moves to the right.

The language accepted by LBA is defined as the set

$$\{w \in (\Sigma - \{¢, \$\})^* \mid (q_0, ¢ w \$, 1) \vdash^* (q, \alpha, i)$$

for some $q \in F$ and for some integer i b/w 1 and n

If L is a context Sensitive Language, then L is accepted by a linear bounded automaton. The converse is also true.

Decidability and Recursively Enumerable Languages

The Definition of an Algorithm:

Algorithm is a procedure (finite sequence of instructions which can be mechanically carried out) that terminates after a finite number of steps for any i/p.

The formal defn. of alg. emerged after the works of Alan Turing and Alanzo church in 1936. The Church Turing thesis states that any algorithmic procedure that can be carried out by a human or a computer, can also be carried out by a TM. Thus the TM arose as an ideal theoretical model of an algorithm.

The TM provided a machinery to mathematicians for attacking the Hilbert's tenth problem.

The problem can be restated as follows; does there exist a TM that can accept a polynomial over n variables if it has an integral root and reject the polynomial if it does not have one.

In 1970 Yuri Matijasevic, after studying the work of Martin Davis,

Hilary Putnam and Julia Robinson showed that no such algorithm (Turing Machine) exists for testing whether a polynomial over n variables has integral roots.

Now it is universally accepted by computer scientists that TM is a mathematical model of an algorithm.

Decidability:

Defn 1: A language $L \subseteq \Sigma^*$ is recursively enumerable if there exists a TM M such $L = T(M)$

Defn 2: A language $L \subseteq \Sigma^*$ is recursive if there exists some TM M that satisfies the following two conditions.

i) If $w \in L$ then M accepts w and halts.

ii) If $w \notin L$ then M halts, without reaching an accepting state.

Defn 3: A problem with two answers (Yes/No) is decidable if the corresponding language is recursive. In this case, the language L is also called decidable

Defn 4: A problem / language is undecidable if it is not decidable.

Decidable Languages:

Defn 5: $A_{DFA} = \{(B, w) \mid B \text{ accepts the input string } w\}$

Theorem: A_{DFA} is decidable.

Proof: To prove the theorem, we have to construct a TM that always halts and also accepts A_{DFA}. We define a TM M as follows

1. Let B be a DFA and w an i/p string. (B, w) is an i/p for the Turing m/c M.
2. Simulate B and input w in the TM
3. If the simulation ends in an accepting state of B then M accepts w. If it ends in a nonaccepting state of B the M rejects w.

M accepts (B, w) if and only if w is accepted by the DFA B.

Defn 6: $A_{CFG} = \{(G, w) \mid \text{the context-free grammar } G \text{ accepts the i/p string } w\}$

Theorem: A_{CFG} is decidable.

Now we design a TM that halts as follows

1) Let G be a CFG in CNF and w an i/p

string. (G, w) is an input for M.

2) If $k=0$, list all the single step derivations. If $k \neq 0$ list all the derivations with $2k-1$ steps.

3) If any of the derivations in step 2 generates the given string w, M accepts (G, w) Otherwise M rejects.

Defn 7: $A_{CSG} = \{(G, w) \mid$ the context-sensitive grammar G accepts the input string $w\}$

Theorem: A_{CSG} is decidable.

We can design a TM M as follows:

1. Let G be a context-sensitive grammar and w an input string of length n. Then (G, w) is an input for TM.
2. Construct $W_0 = \{S\}$. $W_{i+1} = W_i \cup \{\beta \in (V_N \cup \Sigma)^* \mid \alpha_i \in W_i$ such that $\alpha \Rightarrow \beta$ and $|\beta| \leq n\}$. Continue until $W_k = W_{k+1}$ for some k.
3. If $w \in W_k$, $w \in L(G)$ and M accepts (G, w); Otherwise M rejects (G, w)

Note: If L_d denotes the class of all decidable languages over Σ then

$$L_{rl} \subseteq L_{cfl} \subseteq L_{csl} \subseteq L_d$$

Halting problem:

Given a Turing machine M and an input string w with the initial configuration q_0 after some (or all) computations do the machine M halts?

In other words we have to identify whether (M, w) where M is the Turing m/c, halts or does not halt when w is applied as the input. The domain of the problem is to be taken as the set of all TM and all w

It is not possible to find the answer for Halting problem by simulating the action of M on w by a universal TM, because there is no limit on the length of the computation. If M enters into an infinite loop, then no matter how long we wait, we can never be sure that M is in fact in a loop. The machine may be in a loop because of a very long computation.

Theorem: $HALT_{TM} = \{(M,w) \mid$ The Turing m/c halts on i/p w$\}$ is undecidable.

Proof: We assume that $HALT_{TM}$ is decidable and get a contradiction. Let M_1 be the TM

such that $T(M_1) = HALT_{TM}$ and let M_1 halt eventually on all (M, w). We construct a TM M_2 as follows:

1) For M_2, (M, w) is an input
2) The TM M_1 acts on (M, w)
3) If M_1 rejects (M, w) then M_2 rejects (M, w)
4) If M_1 accepts (M, w) simulate the TM M on the i/p string w until M halts.
5) If M has accepted w, M_2 accepts (M, w); otherwise M_2 rejects (M, w).

$$T(M_2) = \{(M, w) \mid \text{The TM accepts } w\}$$
$$= A_{TM}$$

This is a contradiction since A_{TM} is undecidable

The Post Correspondence problem:

The PCP was first introduced by Emil Post in 1946. Later the problem was found to have many applns in theory of formal languages. The problem over an alphabet Σ belongs to a class of yes/no problems and is stated as follows.

Consider the two lists $x = (x_1, \ldots x_n)$ $y = (y_1 \ldots y_n)$ of nonempty strings over an alphabet $\Sigma = \{0, 1\}$. The PCP is to determine

Whether or not there exist $i_1, \ldots i_n$ where $1 \leq ij \leq n$ such that

$$x_{i_1} \ldots x_{im} = y_{i_1} \ldots y_{im}$$

Does the PCP with two lists $x = (b, bab^3, ba)$ and $y = (b^3, ba, a)$ have a solution

Required sequence is given by

$i_1 = 2, i_2 = 1, i_3 = 1, i_4 = 3$ ie $(2,1,1,3)$ and $m = 4$

The corresponding strings are

x_2	x_1	x_1	x_3	=	y_2	y_1	y_1	y_3
bab^3	b	b	ba		ba	b^3	b^3	a

Thus the PCP has a solution.

Growth Rate of Functions.

When we have two algorithms for the same problem, we may require a comparison b/w the running time of these alg. This leads to the study of growth rate of fns defined on the set of natural numbers

eg1:

Let $f(n) = 4n^3 + 5n^2 + 7n + 3$ Prove that $f(n) = O(n^3)$

Solution:

Take $c = 5$ and $N_0 = 10$ then

$f(n) = 4n^3 + 5n^2 + 7n + 3 \leq 5n^3$ for $n \geq 10$

when $n = 10$,

$5n^2 + 7n + 3 = 573 < 10^3$

For $n > 10$

$5n^2 + 7n + 3 < n^3$

$\therefore f(n) = O(n^3)$

Theorem: If $p(n) = a_k n^k = a_k n^k + a_{k-1} n^{k-1} + \ldots$ $\ldots a_1 n + a_0$ is a polynomial of degree k over Z and $a_k > 0$ then $p(n) = O(n^k)$

Proof:

$p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots a_1 n + a_0$

As a_k is an integer and positive $a_k \geq 1$

As $a_{k-1}, a_{k-2} \ldots a_1, a_0$ and k are fixed integers, choose N_0 such that for all $n \geq N_0$ each of the members

$\frac{|a_{k-1}|}{n}, \frac{|a_{k-2}|}{n^2} \ldots \frac{|a_1|}{n^{k-1}}, \frac{|a_0|}{n^k}$ is less than $\frac{1}{k}$

Hence

$$\left| \frac{a_{k-1}}{n} + \frac{a_{k-2}}{n^2} + \ldots \frac{a_0}{n^k} \right| < 1$$

As $a_k \geq 1$, $\frac{p(n)}{n^k} = a_k + \frac{a_{k-1}}{n} + \ldots \frac{a_1}{n^{k-1}} + \frac{a_0}{n^k} > 0$

for all $n \geq N_0$

Also

$$\frac{p(n)}{n^k} = a_k + \left(\frac{a_{k-1}}{n} + \cdots \frac{a_1}{n^{k-1}} \frac{+a_0}{n^k}\right)$$

$$\leq a_k + 1$$

So $p(n) \leq Cn^k$ where $C = a_k + 1$

$\therefore$ $p(n) = O(n^k)$

Corollary: The order of a polynomial is determined by its degree.

An exponential function is a function $q: N \to N$ defined by $q(n) = a^n$ for some fixed $a > 1$

When n increases each of n, n^2, 2^n increases. But a comparison of these functions for specific values of n will indicate the vast difference b/w the growth rate of these fns

Growth rate polynomial and Exponential fn

n	$f(n) = n^2$	$g(n) = n^2 + 3n + 9$	$q(n) = 2^n$
1	1	13	2
5	25	49	32
10	100	139	1024
50	2500	2659	$(1.13)10^{15}$

It is easy to see that the function $g(n)$ grows at a very fast rate when compared to $f(n)$ or $g(n)$. In particular the exponential function grows at a very fast rate when compared to any polynomial of large degree.

The classes P and NP:

A Turing machine M is said to be of time complexity $T(n)$ if the following holds: Given an i/p w of length n, M halts after making atmost $T(n)$ moves.

In this case, M eventually halts.

The standard TM is called a deterministic TM.

Quantum Computers:

A quantum bit or qubit can be described mathematically as $|\psi| = \alpha|0\rangle + \beta|0\rangle$

The qubit can be explained as follows.

A classical bit has two states, a 0 and a 1.

Two possible states for a qubit are the states $|0\rangle$ and $|1\rangle$.

Unlike a classical bit, a qubit can be in infinite number of states other than $|0\rangle$ and $|1\rangle$

It can be in a state $|\psi| = \alpha|0\rangle + \beta|0\rangle$ where

α and β are complex numbers such that $|\alpha|^2+|\beta|^2=1$. The 0 and 1 are called the computational basis states and $|\psi|$ is called a superposition. We can call $|\psi|=\alpha|0\rangle+\beta|0\rangle$ a quantum state.

It is not possible to determine the quantum state on observation. When we measure/observe a qubit, we get either the state $|0\rangle$ with probability $|\alpha|^2$ or the state $|1\rangle$ with probability $|\beta|^2$.

Multiple qubits can be defined in a similar way. For eg, a two-qubit system has four computational basis states, $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ and quantum states

$|\psi\rangle=\alpha_{00}|00+\alpha_{01}|01+\alpha_{10}|10\rangle+\alpha_{11}|11\rangle$

with $|\alpha_{00}|^2+|\alpha_{01}|^2+|\alpha_{10}|^2+|\alpha_{11}|^2=1$

Qubit NOT gate can be described by

$$\begin{bmatrix}\alpha\\ \beta\end{bmatrix}\rightarrow\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}=\begin{bmatrix}\beta\\ \alpha\end{bmatrix}$$

Qubit gate corresponding to NOR is the controlled-NOT or CNOT gate. It can be described by

$$|A,B\rangle\rightarrow|A,B\oplus B\rangle$$

where ⊕ denotes addition modulo 2. The action on computational basis is $|00\rangle \rightarrow |00\rangle$, $|01\rangle \rightarrow |01\rangle$, $|01\rangle, |10\rangle \rightarrow |11\rangle, |11\rangle \rightarrow |10\rangle$. It can be described by the following 4×4 unitary matrix

$$U_{CN} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Quantum computer is a system built from quantum circuits, containing wires and elementary quantum gates, to carry out manipulation of quantum information.

Church-Turing Thesis.

Church-Turing thesis which asserts that any algorithm that can be performed on any computing machine can be performed on a Turing machine as well.

Any algorithmic process can be simulated efficiently by a Turing m/c. But a challenge to the strong Church-Turing thesis arose from analog computation.

Certain types of analog computers solved some problems efficiently whereas these problems had no efficient solution on a Turing m/c. But when the presence of noise was taken into account, the power of the analog computers disappeared.

In mid-1970, Robert Solovay and Volker Strassen gave a randomized algorithm for testing the primality of a number. This led to the modification of the Church thesis.

Strong Church–Turing Thesis:

Any algorithmic process can be simulated efficiently using a nondeterministic TM. In 1985, David Deutsch tried to build computing devices using quantum mechanics.

" Computers are physical objects and computations are physical processess. What computers can or cannot compute is determined by the law of physics alone and not by pure mathematics".

In 1994, Peter Shor proved that finding the prime factors of a composite

number and the discrete logarithm problem could be solved efficiently by a quantum computer. This may be a pointer to proving that quantum computers are more efficient than TM.