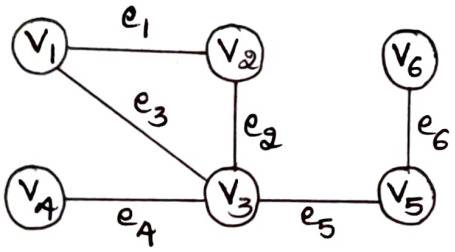
Graphs :-

- Graphs in data structure are non-linear data structures made up of a finite number of nodes or vertices and the edges that connect them.
- A graph G can be defined as an ordered set of G(V, E) where
  V(G) → represents the set of vertices
  E(G) → represents the set of edges which are used to connect these vertices.

Eg :-

1)

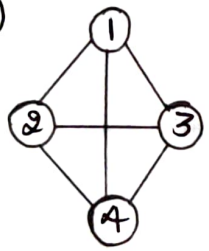


$V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6\}$
$|V| = 6$
$E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6\}$
$|E| = 6$

$E(G) = \{(v_1, v_2)\ (v_2, v_3), (v_1, v_3)\ (v_3, v_4)\ (v_3, v_5)\ (v_5, v_6)\}$

- There are 6 edges and 6 vertex in a graph.

2)



$V(G) = \{1, 2, 3, 4\}$
$E(G) = \{(1,2)\ (1,3)\ (1,4)\ (2,3)\ (2,4)\ (3,4)\}$
$|V| = 4$ and $|E| = 6$

3)

$V(G) = \{1, 2, 3\}$
$E(G) = \{(1,2)\ (2,1)\ (2,3)\}$
$|V| = 3$ and $|E| = 3$

Applications :-

- Graphs in data structures are used to address real-world problems in which it represents the problem area as a network like telephone networks, circuit networks and social networks.

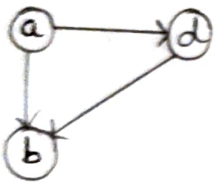
Eg :- It can represent a single user as nodes or vertices in a telephone network, which links between them via telephone represents the edges.

Types of graphs :-

1) Directed graph
2) Undirected graph

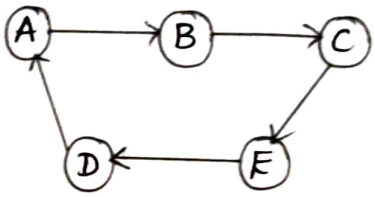
1) Directed graph :-

- In directed graph, edges form an ordered pair.
- Edges represent a specific path from some vertex A to another vertex B.
- Vertex A is called as initial vertex and vertex B is called as terminal vertex
- Directed graph is denoted by the directed pair as $\langle A, B \rangle$ within the angular brackets.



$V(G) = \{a, b, d\}$

$E(G) = \{ \langle a,b \rangle \langle a,d \rangle \langle d,b \rangle \}$

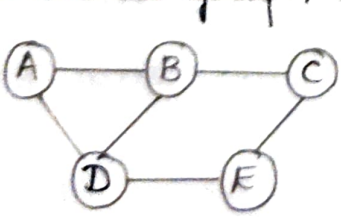


$V(G) = \{ A, B, C, D, E \}$

$E(G) = \{ \langle A, B \rangle \langle B, C \rangle \langle B, D \rangle \langle C, E \rangle \langle E, D \rangle \langle D, A \rangle \}$

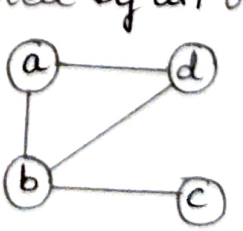
2) Undirected graph :-

- In undirected graph, edges are not associated with the direction with them.
- If an edge exists between vertex A and B then the vertices can be traversed from B to A as well as A to B since its edges are not attached with any of the directions.
- Undirected graph edges are denoted by an ordered pair as (A,B)



$V(G) = \{ A, B, C, D, E \}$

$E(G) = \{ (A,B) (A,D) (B,C) (B,D) (C,E) (E,D) \}$



$V(G) = \{ a, b, c, d \}$

$E(G) = \{ (a,b) (a,d) (b,d) (b,c) \}$

Terminologies :-

1) **Vertex** :- It is a synonym for a node. A vertex is represented by circle.

Eg :- ① ② ③ Nodes 1, 2, 3 are vertices.

2) **Edge** :- An arc / line joining 2 vertices say U & V is called as an edge.

Eg :- ① —— ② U —— V

a) **Directed edge** :- Direction exists between 2 vertices and denoted by the directed pair $\langle 1,2 \rangle$ where 1 is called tail of the edge and 2 is the head of the edge.

Eg :- ① ⟶ ②

$\langle 2,1 \rangle$ is not same as $\langle 1,2 \rangle$

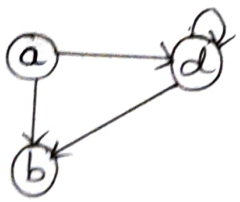
b) **Undirected edge** :- Direction does not exists between 2 vertices and denoted by an ordered pair $(1,2)$.

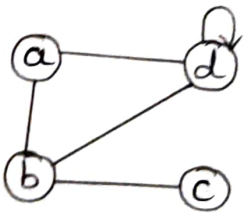
Eg :- ① —— ②

$(1,2)$ is same as $(2,1)$

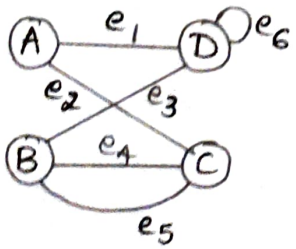
3) **Self loop / Self edge** :- An edge which starts and ends on the same vertex.

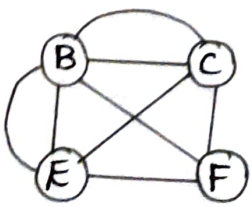
Eg :- ①





4) **Multigraph** :- It is a graph with multiple occurrences of edges between the same 2 vertices or a pair of vertices.

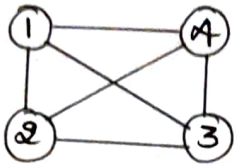




5) Complete graph :-

- A graph $G = (V, E)$ is said to be a complete graph if every node is connected with all other nodes.
- For a complete graph with 'n' vertices, there will be $n(n-1)/2$ edges.

Eg :-

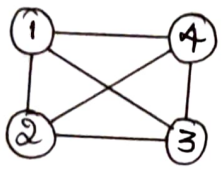


$n = 4$ vertices

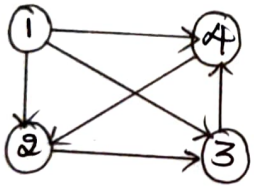
$n(n-1)/2$ edges = 6 edges.

6) Path :- A path is denoted using sequence of vertices and there exists an edge from one vertex to another vertex.

Eg :-



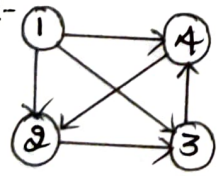
Path from vertex 1 to 4 is denoted as $(1, 2, 3, 4)$ which can also be written as $(1, 2)\ (2, 3)\ (3, 4)$ (or) $(1, 3)\ (3, 2)\ (2, 4)$ $(1, 2)\ (2, 4)$ (or) $(1, 2)\ (2, 3)(3, 1)(1, 4)$



Path from vertex 1 to 3 is denoted by $\langle 1, 4, 2, 3 \rangle$ which can also be written as $\langle 1, 4 \rangle\ \langle 4, 2 \rangle\ \langle 2, 3 \rangle$

7) Simple path :- A path in which all the vertices are distinct

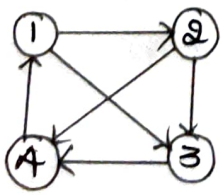
Eg :-



Path $\langle 1, 4, 2, 3 \rangle$ is a simple path, since each node in the sequence is distinct i.e, appears only once.

8) closed path :- A path in which the initial node is same as terminal node. closed path is also called as cycle.

Eg :-

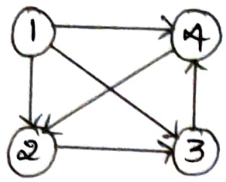


Path $\langle 1, 2, 3, 4, 1 \rangle$ is a closed path where initial = 1 and terminal = 1.

Path $\langle 1, 2, 4, 1 \rangle$ is a closed path.

9) Length of the path :- Number of edges in the path gives the length of the path.

Eg :-



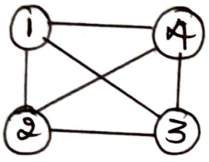
$\langle 1, 4, 2, 3 \rangle = 3$

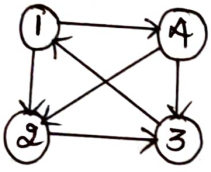
Length of the path from vertex 1 to 3 is 3 i.e, there exists 3 edges.

10) Connected graph :- A graph G is said to be connected if and only if there exists a path between every pair of vertices.

- Both directed and undirected graphs are connected graphs
- There should be atleast 1 incoming path for all the vertex.

Eg :-





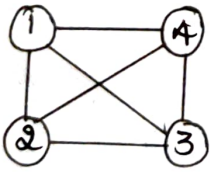
- From Vertex 1, we have the path as $1 \to 4$, $1 \to 2$ & $1 \to 2 \to 3$.

i.e, from Vertex 1, every other vertices are connected directly as well as indirectly through some other vertex.

Eg :- Vertex 3 is connected indirectly from 1 through 2.

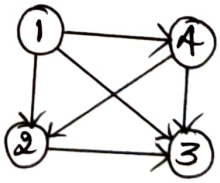
11) Disconnected graph :- A graph G is said to be disconnected if there exists atleast one vertex in a graph that cannot be reached from other vertices.

Eg :-



5

Vertex 5 is not connected.



Vertex 1 is not reachable from 4
Vertex 1 is not reachable from 3
Vertex 1 is not reachable from 2

Representation of graphs :-

There are 2 different methods to represent a graph.

1) Adjacency matrix
2) Adjacency linked list

1) Adjacency matrix :-

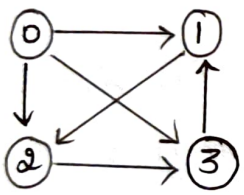
- Let $G=(V, E)$ be a graph, where $V \rightarrow$ set of vertices and $E \rightarrow$ set of edges.
- Let N be the number of vertices in graph G. The adjacency matrix A graph G is defined as

$$A[i][j] = \begin{cases} 1 & \text{if there is an edge from vertex } i \text{ to vertex } j \\ 0 & \text{if there is no edge from vertex } i \text{ to vertex } j \end{cases}$$

- It is a boolean square matrix with 'n' rows and 'n' column with entries 1's and 0's.

Eg :-

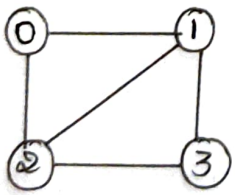
Directed graph



Adjacency matrix

	0	1	2	3
0	0	1	1	1
1	0	0	1	0
2	0	0	0	1
3	0	1	0	0

Undirected graph



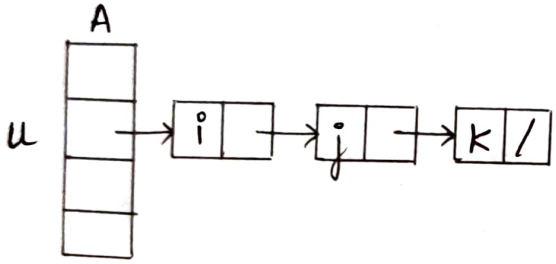
Adjacency matrix

	0	1	2	3
0	0	1	1	0
1	1	0	1	1
2	1	1	0	1
3	0	1	1	0

2) Adjacency linked list :-

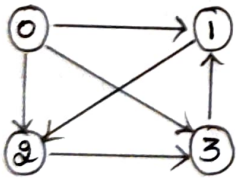
- Let $G = (V, E)$ be a graph, where $V \rightarrow$ set of vertices and $E \rightarrow$ set of edges.
- An adjacency linked list is an array of 'n' linked list where 'n' is the number of vertices in graph G.
- Each location of the array represents a vertex of the graph
- For each vertex $u \in V$, a linked list consisting of all the vertices adjacent to u is created and stored in A[u] and the resulting array A is an adjacency list.

i.e, if i, j and k are the vertices adjacent to vertex u, then i, j & k are stored in a linked list and starting address of linked list is stored in A[u]

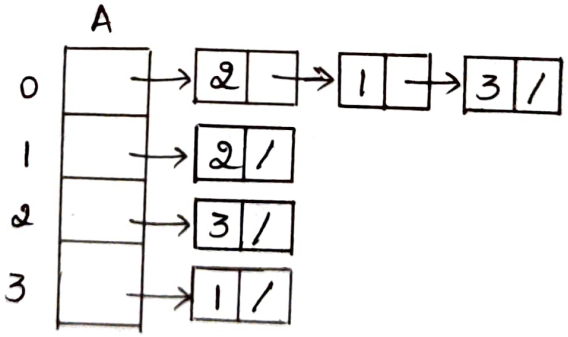


Eg :-

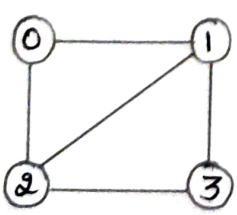
Directed graph



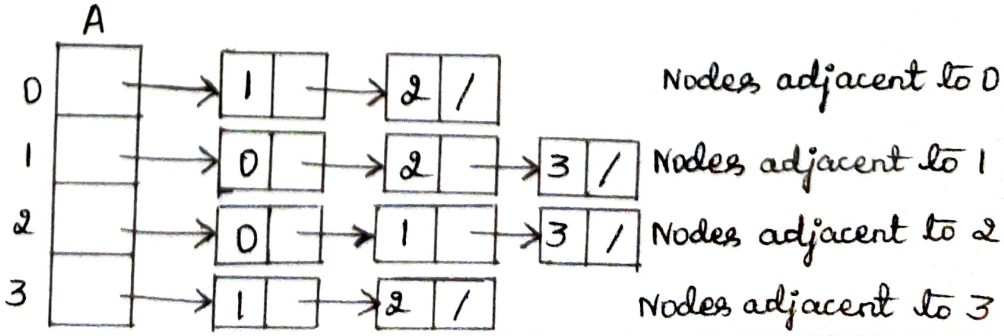
Adjacency linked list



Undirected graph

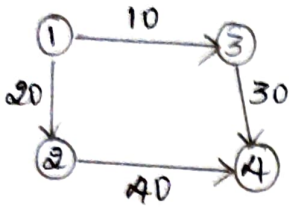


Adjacency linked list



Weighted graphs :-

- A graph in which a number is assigned to each edge in a graph is called weighted graph.
- These numbers are called as costs or weights.
- Weights may represent the cost involved or length or capacity depending on the problem.



- In the graph shown, values 10, 20, 30 & 40 are the weights associated with 4 edges (1,3) (3,4) (1,2) and (2,4).

Weighted graph representation :-

- Weighted graphs can be represented using adjacency matrix as well as adjacency linked list.
- Adjacency matrix consisting of costs/weights can also be called as cost adjacency matrix.
- Adjacency linked list consisting of costs/weights can also be called as cost adjacency linked list.

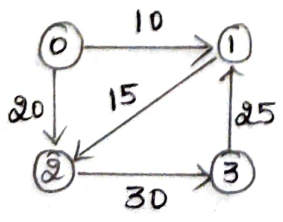
Cost adjacency matrix :-

- Let G = (V, E) be the graph where V → set of vertices & E → set of edges with 'n' number of vertices. The cost adjacency matrix 'A' of a graph G is formally defined as

$$A[i][j] = \begin{cases} w & \text{if there is a weight with edge from vertex } i \text{ to vertex } j \\ \infty & \text{if there is no edge from vertex } i \text{ to vertex } j \end{cases}$$

Eg :-

weighted directed graph



cost adjacency matrix

	0	1	2	3
0	∞	10	20	∞
1	∞	∞	15	∞
2	∞	∞	∞	30
3	∞	25	∞	∞

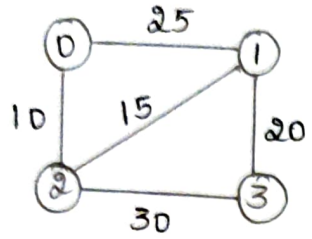
Diagonal values can be replaced by zero.

	0	1	2	3
0	∞	10	20	∞
1	∞	∞	15	∞
2	∞	∞	∞	30
3	∞	25	∞	∞

$\Rightarrow$

	0	1	2	3
0	0	10	20	∞
1	∞	0	15	∞
2	∞	∞	0	30
3	∞	25	∞	0

Weighted undirected graph



Cost adjacency matrix

	0	1	2	3
0	∞	25	10	∞
1	25	∞	15	20
2	10	15	∞	30
3	∞	20	30	∞

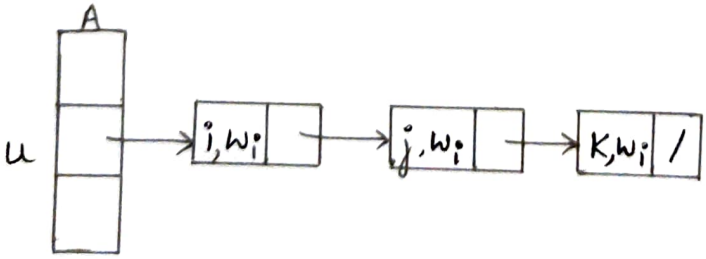
Diagonal values can be replaced by 0's

$\Downarrow$

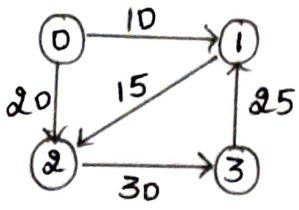
	0	1	2	3
0	0	25	10	∞
1	25	0	15	20
2	10	15	0	30
3	∞	20	30	0

Cost adjacency linked list :-

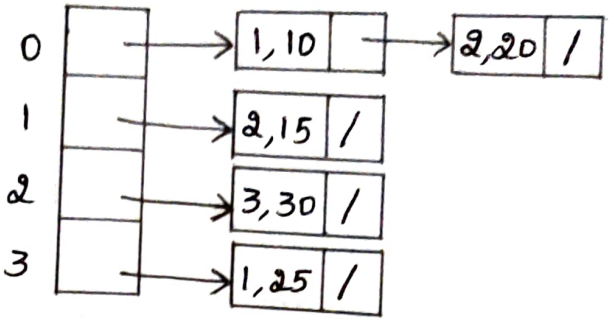
- It is an array of 'n' linked lists. For each vertex $u \in V$, A[u] contains the address of a linked list.
- All the vertices which are adjacent from vertex 'u' are stored in the form of a linked list and the starting address of the 1st node is stored in A[u].
- If i, j and k are the vertices adjacent to the vertex u, then i, j & k are stored in a linked list along with the weights in A[u] as



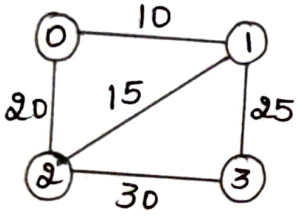
Weighted directed graph



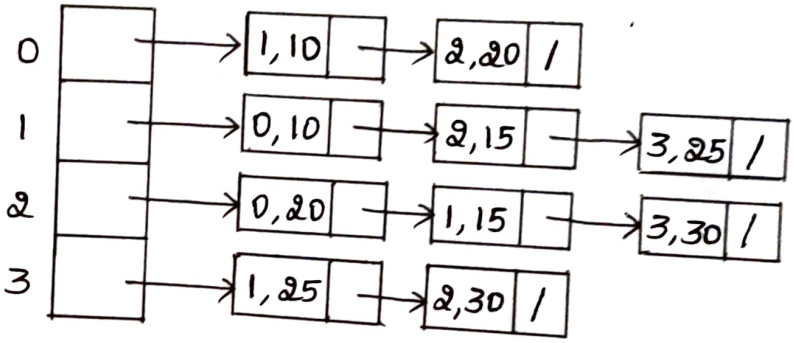
cost adjacency linked list



Weighted Undirected graph



cost adjacency linked list



Graph Traversal :-

- It is a process of visiting each node of a graph systematically in some order.
- There are 2 types of traversal technique
  1) Breadth First Search (BFS)
  2) Depth First Search (DFS)
- During the execution of the traversal techniques algorithms, each node N of graph G will be in one of 3 states, called the status of N, as follows:

STATUS = 1 : (Ready state) The initial state of the node N.

STATUS = 2 : (Waiting state) The node N is on the queue or stack, waiting to be processed.

STATUS = 3 : (Processed state) The node N has been processed.

1) Breadth First Search :-

- BFS is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes.
- Then it selects the nearest node and explores all the unexplored nodes.
- In BFS, for traversal any node in the graph can be considered as the root node.
- BFS is a recursive algorithm, which is used to search all the vertices of a tree or graph data structure.
- BFS puts every vertex of the graph into 2 categories i.e, Visited and Non-Visited
- It selects a single node in a graph and after that visits all the nodes adjacent to the selected node.
- BFS uses 'Queue' data structure for implementation.
- It uses 2 queues, namely QUEUE1 and QUEUE2.

QUEUE1 → holds all the nodes that are to be processed. i.e, nodes to be visited.

QUEUE2 → holds all the nodes that are processed and deleted from QUEUE1. i.e, already visited and deleted nodes.

Algorithm :-

The steps involved in the BFS algorithm to explore a graph are given as follows:

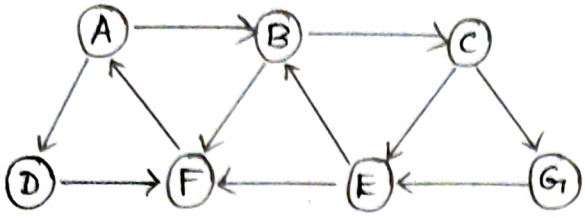
Steps :-

1) Set STATUS = 1 (ready state) for each node in G.
2) Enqueue the starting node A and set its STATUS = 2 (waiting state)
3) Repeat steps 4 & 5 until QUEUE is empty.
4) Dequeue a node N. Process it and set its STATUS = 3 (processed state)
5) Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

End of LOOP. 6) Exit.

Eg :-

1) Find the nodes that are reachable from node 'A' by considering 'A' as root node using BFS.



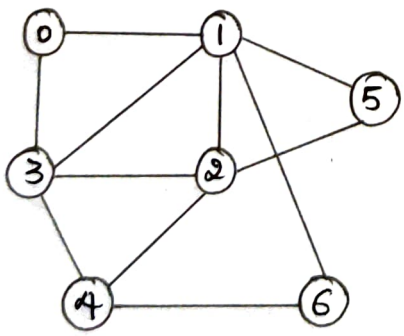
Adjacency lists

A : B, D
B : C, F
C : E, G
G : E
F : A
D : F
E : B, F

QUEUE1 = U Nodes to be visited	V = Adjacent to U	QUEUE2 Nodes that are visited & deleted
A	B, D	A
B	D, C, F	A, B
D	C, F	A, B, D
C	F, E, G	A, B, D, C
F	E, G	A, B, D, E, F
E	G	A, B, D, C, F, E
G	–	A, B, D, C, F, E, G

∴ Nodes that are reachable from source 'A' are
A, B, D, C, F, E, G

2) Find the minimum path from '0' to '6' using BFS.



Adjacency lists

0 : 1, 3
1 : 0, 2, 3, 5, 6
2 : 1, 3, 4, 5
3 : 0, 1, 2
4 : 2, 3, 6
5 : 1, 2
6 : 1, 4

Source : 0
Destination : 6

QUEUE1 = u	v = adj to u	QUEUE2
0	1, 3	0
1	3, 2, 5, 6	0, 1
3	2, 5, 6,	0, 1, 3
2	5, 6, 4	0, 1, 3, 2
5	6, 4	0, 1, 3, 2, 5
6	4	0, 1, 3, 2, 5, 6

∴ Minimum path from 0 to 6 is 0, 1, 3, 2, 5, 6

2) Depth First Search :-

- DFS is a recursive algorithm to search all the vertices of a tree or graph data structure.
- DFS algorithm starts with the initial node of graph G and goes deeper until we find the goal node with no children.
- DFS uses stack data structure for implementation.
- Step by step process to implement DFS traversal is given as follows:

1) First, create a stack with the total number of vertices in the graph.
2) Now, choose any vertex as the starting point of traversal and push that vertex into the stack.
3) After that, push a non-visited vertex (adjacent to the vertex on the top of the stack) to the top of the stack.
4) Now repeat steps - 3 & 4 until no vertices are left to visit from the vertex on the stack's top.
5) If no vertex is left, go back and pop a vertex from the stack
6) Repeat steps - 2, 3 & 4 until stack is empty.

Algorithm :-

The steps involved in the DFS algorithm to explore a graph are given as follows:

Steps :-

1) Set STATUS = 1 (ready state) for each node in G.
2) Push the starting node A on the stack and set it STATUS = 2 (waiting state)

3) Repeat steps 4 & 5 until STACK is empty.
4) Pop the top node N. Process it and set its STATUS = 3 (processed state)
5) Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
End of LOOP
6) Exit.

Applications of BFS & DFS :-

BFS :-

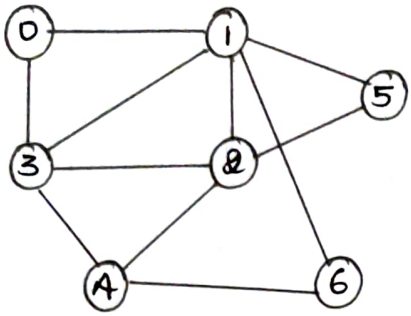
- BFS can be used to find the neighboring locations from a given source location.
- BFS is used to determine the shortest path & minimum spanning tree.
- BFS can be used in web crawlers to create web page indexes. It is one of the main algorithms that can be used to index web pages. It starts traversing from the source page & follows the links associated with the page.
  Here, every web page is considered as a node in the graph.
- It can be used in ford-Fulkerson method to compute the maximum flow in a flow network.

DFS :-

- DFS algorithm can be used to implement the topological sorting
- It can be used to find the paths between 2 vertices.
- It can also be used to detect cycles in the graph.
- It is also used for one solution puzzles.
- DFS is used to determine if a graph is bipartite or not.

Eg :-

1) Solve using DFS algorithm and find the path between all the vertices by considering '0' as a source node.

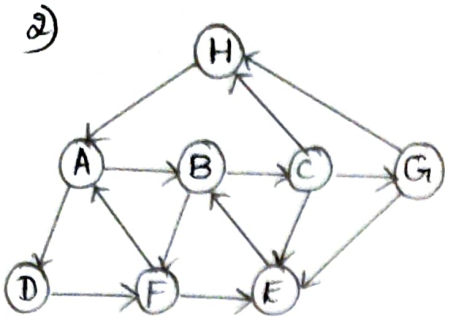


Stack = u	V = Adjacent to u i.e, s[top]	S = Nodes visited	Popped from stack
0	1	0	–
0, 1	3	0, 1	–
0, 1, 3	2	0, 1, 3	–
0, 1, 3, 2	4	0, 1, 3, 2	–
0, 1, 3, 2, 4	6	0, 1, 3, 2, 4	–
0, 1, 3, 2, 4, 6	–	0, 1, 3, 2, 4, 6	6
0, 1, 3, 2, 4	–	0, 1, 3, 2, 4, 6	4
0, 1, 3, 2	5	0, 1, 3, 2, 4, 6	–
0, 1, 3, 2, 5	–	0, 1, 3, 2, 4, 6, 5	–
0, 1, 3, 2, 5	–	0, 1, 3, 2, 4, 6, 5	5
0, 1, 3, 2	–	0, 1, 3, 2, 4, 6, 5	2
0, 1, 3	–	0, 1, 3, 2, 4, 6, 5	3
0, 1	–	0, 1, 3, 2, 4, 6, 5	1
0	–	0, 1, 3, 2, 4, 6, 5	0
–	–	0, 1, 3, 2, 4, 6, 5	

∴ Path between all the vertices from source node '0' is

0, 1, 3, 2, 4, 6, 5

2)



Consider 'H' as a root node and find the path between all the vertices using DFS.

Stack = u	V = adj to u s[top]	S = Nodes Visited	Popped from stack
H	A	H	–
H, A	D	H, A	–
H, A, D	F	H, A, D	–
H, A, D, F	–	H, A, D, F	F
H, A, D	–	H, A, D, F	D
H, A	B	H, A, D, F	–
H, A, B	C	H, A, D, F, B	–
H, A, B, C	E	H, A, D, F, B, C	–
H, A, B, C, E	–	H, A, D, F, B, C, E	E
H, A, B, C	G	H, A, D, F, B, C, E	C
H, A, B, C, G	–	H, A, D, F, B, C, E, G	G
H, A, B, C	–	H, A, D, F, B, C, E, G	C
H, A, B	–	H, A, D, F, B, C, E, G	B
H, A	–	H, A, D, F, B, C, E, G	A
H	–	H, A, D, F, B, C, E, G	H
–	–	H, A, D, F, B, C, E, G	–

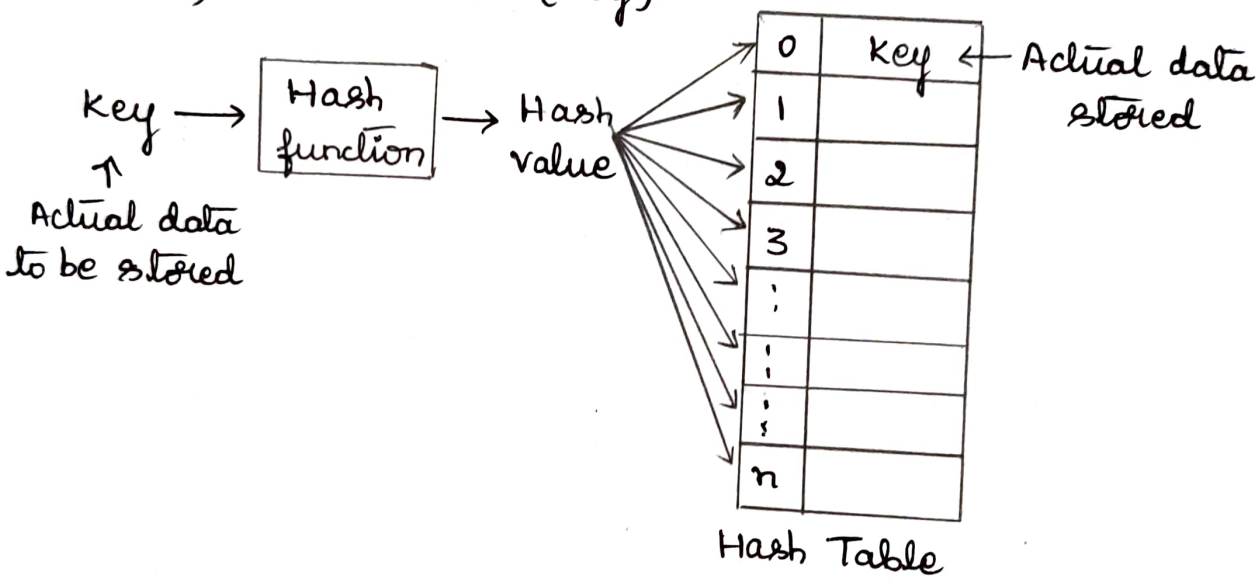
∴ Path between all the vertices from source node 'H' is

H, A, D, F, B, C, E, G

Hashing :-

- Hashing in the data structure is a technique that is used to quickly identify a specific value within a given array.
- Also, it is a technique or a process of mapping keys and values into the hash table by using a hash function for faster access of elements.
- Main idea behind the hashing is to create the key/value pairs.
- If the key is given, then algorithm computes the index at which the value would be stored.

  i.e, Index = hash(key)



Hash Table Organization :-

Hash table :-

- It is a data structure that stores some information and the information has basically 2 main components

  i.e, key and value
- It is used for storing and retriciving data very quickly.
- Insertion, deletion or retrieival operation takes place with the help of hash value.
- Hence, every entry in the hash table is associated with some key.
- Using the hash key the required piece of data can be searched in the hash table by few or more key comparisons.

- Searching time is dependant upon the size of the hash table.

Hash Function :-

- Function used to push the data in the hash table.
- Integer returned by the hash function is called hash key
- Efficiency of mapping the key depends on the efficiency of the hash function used.
- Using hash function, we calculate the address at which the value can be stored.

A good hash function should satisfy 2 criteria :-

1) Hash function should generate the hash addresses such that all the keys are distributed as evenly as possible among the various cells of the hash table.
2) Computation of a key should be simple.

Eg :-

Let a hash function H(x) maps the value 'x' at the index X % 10 in an array.

If the list of values is {11, 12, 13, 14, 15}. It will be stored at positions {1, 2, 3, 4, 5} in the hash table.

$H(X) = X \% 10$ so,

$H(11) = 11 \% 10 = 1$

$H(12) = 12 \% 10 = 2$

$H(13) = 13 \% 10 = 3$

$H(14) = 14 \% 10 = 4$

$H(15) = 15 \% 10 = 5$

$\Rightarrow$

0	
1	11
2	12
3	13
4	14
5	15
⋮	

← Hash Table with Keys & Hash values

Here {1, 2, 3, 4, 5} are called hash values and $H(X) = X \% 10$ is the Hash function used.

Need for Hashing :-

- Gives more secure & adjustable method for retrieving data compared to any other data structure.
- It is an efficient way to store & retrieve data in data structure because it avoids the need for comparisons between elements.
- Main use of hashing is to compare 2 files for equality. i.e, without opening 2 document files to compare them word to word, calculated hash values of these files will allow the owner to know immediately if they are different.
- Hashing is an effective technique to calculate the direct location of a data record on the disk without using index structure.
- Hashing uses hash function with search keys as parameters to generate the address of a data record.

Drawback of Hashing :-

- Hashing in data structure falls into a collision if 2 keys are assigned the same index number in the hash table. collision creates a problem because each index in a hash table is supposed to store only one value.

Hashing Functions :-

There are 4 types of hashing functions, namely :

1) Division method
2) Multiplication method
3) Mid-square method
4) Folding method

1) Division method :-

- Most simple method of hashing an integer 'x'
- This method divides 'x' by 'm' and then uses the remainder obtained.
- This type depends on the remainder of division and divisor is the hash table length.
- This type of hash function can be given as

$$h(x) = x \bmod m$$

where $x \rightarrow$ keys to be stored

$m \rightarrow$ length of the hash table.

Eg :-

If the record to be stored is 54, 72, 89, 37 in the hash table and size of the table is 10

$X = \{54, 72, 89, 37\}$ $\quad m = 10$

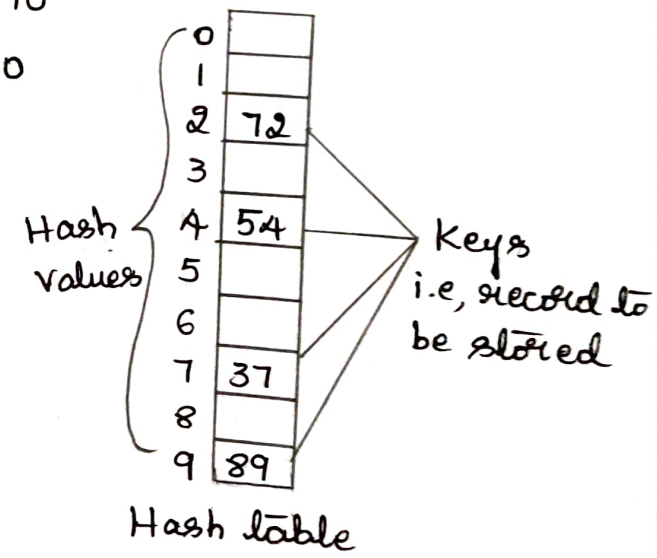
$h(x) = x \bmod m$

$h(54) = 54 \bmod 10 = 4$

$h(72) = 72 \bmod 10 = 2$

$h(89) = 89 \bmod 10 = 9$

$h(37) = 37 \bmod 10 = 7$



2) Multiplication method :-

The steps involved in the multiplication method are as follows

a) Choose a constant A such that $0 < A < 1$

b) Multiply the key 'K' with A.

c) Extract the fractional part of KA.

d) Multiply the result of step-3 by the size of hash table 'm'

e) Hash value will be the floor value i.e, only integer part.

Hence, hash function can be given as

$$h(K) = \lfloor m(KA \bmod 1) \rfloor$$

where $A = 0.618033$

Eg :-

1) Given a hash table of size 1000, map the key 12345 to an appropriate location in hash table.

$A = 0.618033 \quad m = 1000 \quad K = 12345$

$h(k) = \lfloor m(KA \bmod 1) \rfloor$

$h(12345) = \lfloor 1000 (12345 \times 0.618033 \bmod 1) \rfloor$

$= \lfloor 1000 (7629.617385 \bmod 1) \rfloor$

$= \lfloor 1000 (0.617385) \rfloor$

$= \lfloor 617.385 \rfloor$

$= 617$

∴ Key 12345 can be mapped at the location 617

2) When $K = 1234 \quad m = 100 \quad A = 0.56$

$h(k) = \lfloor m (KA \bmod 1) \rfloor$

$h(1234) = \lfloor 100 (1234 \times 0.56 \bmod 1) \rfloor$

$= \lfloor 100 (691.04 \bmod 1) \rfloor$

$= \lfloor 100 (0.04) \rfloor$

$= \lfloor 4 \rfloor$

$= 4$

3) Mid-Square method :-

The steps involved in mid-square method are as follows:

a) Square the value of the key i.e, find K^2.

b) Extract the middle 'r' digits of the result obtained in step-1.

- Same number of 'r' digits must be chosen from all the keys.

Hence, Hash function can be given as

$h(k) = s$

where 's' is obtained by selecting 'r' digits from K^2.

Eg :-

1) Calculate the hash values for keys 1234 and 5642 using mid-square method. The hash table has 100 memory location

Note :- Hash table has 100 memory location whose indices values varies from 0 to 99. This means that only 2 digits are needed to map the key to a location in hash table. So r = 2 digits

When K = 1234, K^2 = 1522756

h(K) = S

h(1234) = 27 — 27 is the middle element obtained from the value of K^2.

When K = 5642, K^2 = 31832164

h(K) = S

h(5642) = 32 — 32 is the middle element obtained from the value of K^2.

When K = 2345, K^2 = 5499025

h(K) = S

h(2345) = 90 — 90 is the middle element obtained from the value of K^2

Note :- If the hash table size is 1000 then index value varies from 0-999. This means that only 3 digits are needed to map the key to a location in hash table. SO, r = 3 digits

4) Folding method :-

The steps involved in the folding method are as follows:

a) Divide the key value into number of parts. That is divide K into parts $K_1, K_2 \ldots K_n$ where each part has the same number of digits except the last part which may have lesser digits than the other parts.

b) Add the individual parts. That is obtain the sum of $K_1 + K_2 + \cdots + K_n$. The Hash value is produced by ignoring the last carry if any.

c) Then perform the division method on the sum of the parts using hash table size

i.e, $h(k) = (K_1 + K_2 + \cdots + K_n) \bmod m$

where $m \rightarrow$ size of the hash table.

Note :- Number of digits in each part of the key will vary depending on the size of the hash table.

i.e, If the hash table has a size of 1000, then there are 1000 locations in the hash table. To address these 1000 locations indices varies from 0-999, so we need at least 3 digits.

Therefore, each part of the key must have 3 digits except the last part which may have lesser digits.

Eg :-

1) Given the hash table of 100 location, calculate the hash value using folding method for keys 5678, 321 & 34567

Note :- As there are 100 locations to address, break the key into parts where each part must contain atleast 2 digits as the index values vary from 0-99 and the last part can contain lesser digits.

when $k = 5678$ $m = 100$

$K_1 = 56$ $K_2 = 78$

$h(k) = (K_1 + K_2) \bmod m$

$h(5678) = (56 + 78) \bmod 100$

$= 134 \bmod 100$

$= 34$

when $k = 321$ $m = 100$

$K_1 = 32$ $K_2 = 1$

$h(k) = (K_1 + K_2) \bmod m$

$h(321) = (32 + 1) \bmod 100$

$= 33 \bmod 100$

$= 33$

when $K = 34567$ $\quad k_1 = 34 \quad k_2 = 56 \quad k_3 = 7$

$h(k) = (k_1 + k_2 + k_3) \bmod m$

$h(34567) = (34 + 56 + 7) \bmod 100$

$= 97 \bmod 100$

$= 97$

② when $K = 123987234876 \quad m = 1000$

$k_1 = 123 \quad k_2 = 987 \quad k_3 = 234 \quad k_4 = 876$

$h(k) = (k_1 + k_2 + k_3 + k_4) \bmod m$

$h(123987234876) = (123 + 987 + 234 + 876) \bmod 1000$

$= 2220 \bmod 1000$

$= 220$

Collisions :-

- Collision occurs when the hash function maps to different keys to the same location as the 2 records cannot be stored in same location.
- Phenomenon of 2 or more keys being hashed to the same location of the hash table is called collision.

Eg :- The data elements to be placed are : 44, 73, 17, 77

$h(x) = x \bmod m$ where $m = 10$

- If we try to place 77 in the hash table, we get the hash value 7 and at hash value 7, already 17 is placed. This situation is called collision.
- Methods used to resolve collision are called collision resolution technique.

0	
1	
2	
3	73
4	44
5	
6	
7	17
8	

- 2 Most popular methods of resolving collisions are
  1) Open addressing or closed hashing
  2) Chaining.

1) Open Addressing :-

- It computes new positions using a probe sequence and the next record is stored in that position.
- All the values are stored in hash table.
- Hash table contains 2 types of values
  - Sentinal values i.e, -1
  - Data values
- Presence of sentinel values indicates that the location contains no data value at present but can be used to hold a value. i.e, the location is free and the data value can be stored in it.
- If the location already has some data value stored in it, then other slots are examined in order to find a free slot.
- Process of examining memory location in the hash table is called probing.
- Open addressing technique can be implemented using
  a) Linear probing
  b) Quadratic probing
  c) Double hashing
  d) Rehashing

a) Linear Probing :-

- Simplest approach to resolve a collision.
- In this technique, if a value is already stored at a location generated by $h(k)$, then the following hash function is used to resolve collision.

$$h(k, i) = [h'(k) + i] \bmod m$$

where $m \rightarrow$ size of the hash table

$$h'(k) = k \bmod m$$

$i \rightarrow$ probe number that varies from 0 to $m-1$.

- For a given key k, first the location generated by $[h'(k) \bmod m]$ is probed because for the first time $i = 0$.
- If the location is free, the value is stored in it, else the second probe generates the address of the location given by $[h'(k) + 1] \bmod m$.

Eg :- Consider a hash table of size 10. using linear probing, insert the keys 72, 27, 36, 24, 63, 81, 92 & 101 into the table.

$h(k, i) = [h'(k) + i] \bmod m$ where $h'(k) = k \bmod m$

$m = 10$

Initially, the hash table can be given as:

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

i) Key = 72

$h(72, 0) = [72 \bmod 10 + 0] \bmod 10$

$= [2 + 0] \bmod 10$

$= 2 \bmod 10$

$= 2$

- Since T[2] is vacant, insert key 72 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	-1	-1	-1

ii) Key = 27

$h(27, 0) = [27 \bmod 10 + 0] \bmod 10$

$= 7 \bmod 10$

$= 7$

- Since T[7] is vacant, insert key 27 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	27	-1	-1

iii) Key = 36

$h(36, 0) = [36 \bmod 10 + 0] \bmod 10$

$= 6 \bmod 10$

$= 6$

- Since T[6] is vacant, insert key 36 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	36	27	-1	-1

iv) Key 24

$h(24,0) = [24 \bmod 10 + 0] \bmod 10$

$= 4 \bmod 10$

$= 4$

- Since T[4] is vacant, insert key 24 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	24	-1	36	27	-1	-1

v) Key 63

$h(63,0) = [63 \bmod 10 + 0] \bmod 10$

$= 3 \bmod 10$

$= 3$

- Since T[3] is vacant, insert key 63 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	63	24	-1	36	27	-1	-1

vi) Key 81

$h(81,0) = [81 \bmod 10 + 0] \bmod 10$

$= 1 \bmod 10$

$= 1$

- Since T[1] is vacant, insert key 81 at this location.

0	1	2	3	4	5	6	7	8	9
-1	81	72	63	24	-1	36	27	-1	-1

vii) Key 92

$h(92,0) = [92 \bmod 10 + 0] \bmod 10$

$= 2 \bmod 10$

$= 2$

- T[2] is occupied, so we cannot store the key 92 in T[2]. Therefore, try again for the next location. Thus, probe i=1

$h(92,1) = [92 \bmod 10 + 1] \bmod 10$

$= [2+1] \bmod 10$

$= 3 \bmod 10$

$= 3$

- T[3] is occupied, so we cannot store the key 92 in T[3]. Therefore, try again for the next location. Thus, probe i = 2

$h(92, 2) = [92 \bmod 10 + 2] \bmod 10$

$= [2 + 2] \bmod 10$

$= 4 \bmod 10$

$= 4$

- T[4] is occupied, so we cannot store the key 92 in T[4]. Therefore, try again for the next location. Thus, probe i = 3

$h(92, 3) = [92 \bmod 10 + 3] \bmod 10$

$= [2 + 3] \bmod 10$

$= 5 \bmod 10$

$= 5$

- Since T[5] is vacant, insert key 92 at this location.

0	1	2	3	4	5	6	7	8	9
-1	81	72	63	24	92	36	27	-1	-1

viii) Key 101

$h(101, 0) = [101 \bmod 10 + 0] \bmod 10$

$= 1 \bmod 10$

$= 1$

- T[1] is occupied, so we cannot store the key 101 in T[1]. Therefore, try again for the next location. Thus, probe i = 1

$h(101, 1) = [101 \bmod 10 + 1] \bmod 10$

$= [1 + 1] \bmod 10$

$= 2 \bmod 10$

$= 2$

- T[2] is also occupied, so we cannot store the key in this location.
- Procedure will be repeated untill the hash function generates the address of location 8 which is vacant and can be used to store the value in it. i.e, at probe i = 7

$h(101, 7) = [101 \bmod 10 + 7] \bmod 10$

$= [1 + 7] \bmod 10$

$= 8 \bmod 10$

$= 8$

0	1	2	3	4	5	6	7	8	9
-1	81	72	63	24	92	36	27	101	-1

- Since T[8] is vacant, insert key 101 at this location.

b) Quadratic probing :-

- In this technique, if a value is already stored at a location generated by $h(k)$, then the following hash function is used to resolve the collision

$$h(k,i) = [h'(k) + c_1 i + c_2 i^2] \bmod m$$

where $m \rightarrow$ size of the table

$h'(k) = k \bmod m$

$i \rightarrow$ probe number that varies from 0 to $m-1$

c_1 & $c_2 \rightarrow$ constants such that c_1 & $c_2 \neq 0$

- Quadratic probing eliminates the primary clustering phenomenon of linear probing because instead of doing a linear search, it does a quadratic search.
- For a given key K, first the location generated by $h'(k) \bmod m$ is probed.
- If the location is free, the value is stored in it, else subsequent locations probed are offset by factors that depend in a quadratic manner on the probe number i.
- In order to maximize the utilization of the hash table, the values of c_1, c_2 & m need to be constrained.

Eg :- Consider a hash table of size 10. Using quadratic probing insert the keys 72, 27, 36, 24, 63, 81 & 101 into the table. Take $c_1 = 1$ & $c_2 = 3$

$$h(k,i) = [h'(k) + c_1 i + c_2 i^2] \bmod m \quad \text{where } h'(k) = k \bmod m$$

$m = 10$

$c_1 = 1$

$c_2 = 3$

Initially, the hash table can be given as

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

i) Key 72

$$h(72,0) = [72 \bmod 10 + 1 \times 0 + 3 \times 0] \bmod 10$$
$$= [72 \bmod 10] \bmod 10 = 2 \bmod 10$$
$$= 2$$

- Since T[2] is vacant, insert the key 72 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	-1	-1	-1

ii) Key 27

$h(27, 0) = [27 \bmod 10 + 1 \times 0 + 3 \times 0] \bmod 10$

$= [27 \bmod 10] \bmod 10$

$= 27 \bmod 10$

$= 7$

- Since T[7] is vacant, insert the key 27 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	27	-1	-1

iii) Key 36

$h(36, 0) = [36 \bmod 10 + 1 \times 0 + 3 \times 0] \bmod 10$

$= [36 \bmod 10] \bmod 10$

$= 6 \bmod 10$

$= 6$

- Since T[6] is vacant, insert the key 36 at this location

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	36	27	-1	-1

iv) Key 24

$h(24, 0) = [24 \bmod 10 + 1 \times 0 + 3 \times 0] \bmod 10$

$= [24 \bmod 10] \bmod 10$

$= 4 \bmod 10$

$= 4$

- Since T[4] is vacant, insert the key 24 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	24	-1	36	27	-1	-1

v) Key 63

$h(63, 0) = [63 \bmod 10 + 1 \times 0 + 3 \times 0] \bmod 10$

$= [63 \bmod 10] \bmod 10$

$= 63 \bmod 10 = 3$

• Since T[3] is vacant, insert the key 63 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	63	24	-1	36	27	-1	-1

vi) key 81

$h(81, 0) = [81 \bmod 10 + 1 \times 0 + 3 \times 0] \bmod 10$

$= [81 \bmod 10] \bmod 10$

$= 1 \bmod 10$

$= 1$

• Since T[1] is vacant, insert the key 81 at this location.

0	1	2	3	4	5	6	7	8	9
-1	81	72	63	24	-1	36	27	-1	-1

vii) key 101

$h(101, 0) = [101 \bmod 10 + 1 \times 0 + 3 \times 0] \bmod 10$

$= [101 \bmod 10] \bmod 10$

$= 1 \bmod 10$

$= 1$

• T[1] is already occupied, the key 101 cannot be stored in T[1]. Therefore, try again for next location. Thus, probe i=1

$h(101, 1) = [101 \bmod 10 + 1 \times 1 + 3 \times 1] \bmod 10$

$= [1 + 1 + 3] \bmod 10$

$= 5 \bmod 10$

$= 5$

• Since T[5] is vacant, insert the key 101 at this location.

0	1	2	3	4	5	6	7	8	9
-1	81	72	63	24	101	36	27	-1	-1

c) Double Hashing :-

- Double hashing uses one hash value and then repeatedly steps forward an interval until an empty location is reached.
- The interval is decided using a second, independent hash function, hence the name double hashing.
- In double hashing, we use 2 hash functions rather than a single function.
- The hash function in the case of double hashing can be given as:

$$h(k, i) = [h_1(k) + ih_2(k)] \bmod m$$

where $m \rightarrow$ size of the hash table

$h_1(k)$ and $h_2(k)$ are 2 hash functions

$h_1(k) = k \bmod m$

$h_2(k) = k \bmod m'$

$i \rightarrow$ probe number that varies from 0 to $m-1$

$m' \rightarrow$ chosen to be less than m such as $m' = m-1$ or $m-2$

- When we have to insert a key k in the hash table, we first probe the location given by applying $[h_1(k) \bmod m]$ because during the first probe, $i = 0$.
- If the location is vacant, the key is inserted into it, else subsequent probes generate locations that are at an offset $[h_2(k) \bmod m]$ from the previous location.

Eg :- Consider a hash table of size = 10. Using double hashing insert the keys 72, 27, 36, 24, 63, 81, 92 and 101 into the table. Take $h_1 = (k \bmod 10)$ and $h_2 = (k \bmod 8)$

$h_1 = k \bmod 10$

$h_2 = k \bmod 8$

$m = 10$

$h(k, i) = [h_1(k) + ih_2(k)] \bmod m$

Initially, the hash table can be given as

0	1	2	3	4	5	6	7	8	9
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

i) Key 72

$h(72,0) = [72 \bmod 10 + (0 \times 72 \bmod 8)] \bmod 10$

$= [2 + (0 \times 0)] \bmod 10$

$= 2 \bmod 10$

$= 2$

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	-1	-1	-1

- Since T[2] is vacant, insert the key 72 at this location.

ii) Key 27

$h(27,0) = [27 \bmod 10 + (0 \times 27 \bmod 8)] \bmod 10$

$= [7 + (0 \times 3)] \bmod 10$

$= 7 \bmod 10$

$= 7$

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	-1	27	-1	-1

iii) Key 36

$h(37,0) = [36 \bmod 10 + (0 \times 36 \bmod 8)] \bmod 10$

$= [6 + (0 \times 4)] \bmod 10$

$= 6 \bmod 10$

$= 6$

- Since T[6] is vacant, insert the key 36 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	-1	-1	36	27	-1	-1

iv) Key 24

$h(24,0) = [24 \bmod 10 + (0 \times 24 \bmod 8)] \bmod 10$

$= [4 + (0 \times 0)] \bmod 10$

$= 4 \bmod 10$

$= 4$

- Since T[4] is vacant, insert the key 24 at this location.

0	1	2	3	4	5	6	7	8	9
-1	-1	72	-1	24	-1	36	27	-1	-1

v) Key 63

$h(63, 0) = [63 \bmod 10 + (0 \times 63 \bmod 8)] \bmod 10$

$= [3 + (0 \times 7)] \bmod 10$

$= (3 + 0) \bmod 10$

$= 3 \bmod 10$

$= 3$

0	1	2	3	4	5	6	7	8	9
-1	-1	72	63	24	-1	36	27	-1	-1

- Since T[3] is vacant, insert the key 63 at this location.

vi) Key 81

$h(81, 0) = [81 \bmod 10 + (0 \times 81 \bmod 8)] \bmod 10$

$= [1 + (0 \times 1)] \bmod 10$

$= 1 \bmod 10$

$= 1$

- Since T[1] is vacant, insert the key 81 at this location.

0	1	2	3	4	5	6	7	8	9
-1	81	72	63	24	-1	36	27	-1	-1

vii) Key 92

$h(92, 0) = [92 \bmod 10 + (0 \times 92 \bmod 8)] \bmod 10$

$= [2 + (0 \times 4)] \bmod 10$

$= (2 + 0) \bmod 10$

$= 2$

- T[2] is already occupied, so we cannot store the key 92 at T[2]. Therefore, try again for the next location. Thus, probe i = 1.

$h(92, 1) = [92 \bmod 10 + (1 \times 92 \bmod 8)] \bmod 10$

$= [2 + (1 \times 4)] \bmod 10$

$= 6 \bmod 10$

$= 6$

- T[6] is already occupied, so we cannot store the key 92 at T[6]. Therefore, try again for the next location. Thus, probe i = 2

$$h(92, 2) = [92 \bmod 10 + (2 \times 92 \bmod 8)] \bmod 10$$
$$= [2 + (2 \times 4)] \bmod 10$$
$$= 10 \bmod 10$$
$$= 0$$

- Since T[0] is vacant, insert the key 92 at this location.

0	1	2	3	4	5	6	7	8	9
92	81	72	63	24	-1	36	27	-1	-1

viii) Key 101

$$h(101, 0) = [101 \bmod 10 + (0 \times 101 \bmod 8)] \bmod 10$$
$$= [1 + (0 \times 5)] \bmod 10$$
$$= 1 \bmod 10$$
$$= 1$$

- T[1] is already occupied, so we cannot store the key 101 in T[1]. Therefore, try again for the next location. Thus probe i=1

$$h(101, 1) = [101 \bmod 10 + (1 \times 101 \bmod 8)] \bmod 10$$
$$= [1 + (1 \times 5)] \bmod 10$$
$$= 6 \bmod 10$$
$$= 6$$

- T[6] is already occupied, so we cannot store the key 101 at T[6]. Therefore, try again for the next location with probe i=2.
- Repeat the entire process until a vacant location is found
- We can see that, we have to probe many times to insert the key 101 in hash table.
- Although double hashing is a very efficient algorithm, it always requires 'm' to be a prime number.
- In our case, m=10 which is not a prime number, hence the degradation in performance.
- We can say that, the performance of the technique is sensitive to the value of 'm'.

d) Rehashing :-

- When the hash table becomes full, number of collisions increases, thereby degrading the performance of insertion and search operations.
- In such cases, a better option is to create a new hash table with size double of the original hash table.
- All the entries in the original hash table will then have to be moved to the new hash table.
- This is done by taking each entry, computing its new hash value and then inserting it in the new hash table.
- Hash function that can be used is

$$H(K) = K \bmod m$$

where $m \rightarrow$ size of the hash table

Eg :- Consider the hash table of size = 5. Hash table looks as follows given below.

0	1	2	3	4
	26	31	43	17

- Rehash the keys 26, 31, 43, 17 into a new hash table using rehashing technique.
- New hash table is of 10 locations, double the size of the original table.
- Rehash the key values from the old hash table into the new one using hash function, $h(K) = K \bmod 10$

$H(26) = 26 \bmod 10 = 6$

$H(31) = 31 \bmod 10 = 1$

$H(43) = 43 \bmod 10 = 3$

$H(17) = 17 \bmod 10 = 7$

Therefore, new hash table of size 10, looks like as follows

0	1	2	3	4	5	6	7	8	9
	31		43			26	17		

2) Chaining :-

- Chaining technique avoids collision using an array of linked lists
- If more than one key has same hash value, then all the keys will be inserted at the end of the list one by one and thus collision is avoided.
- In chaining, each location in a hash table stores a pointer to a linked list that contains all the key values that were hashed to that location.

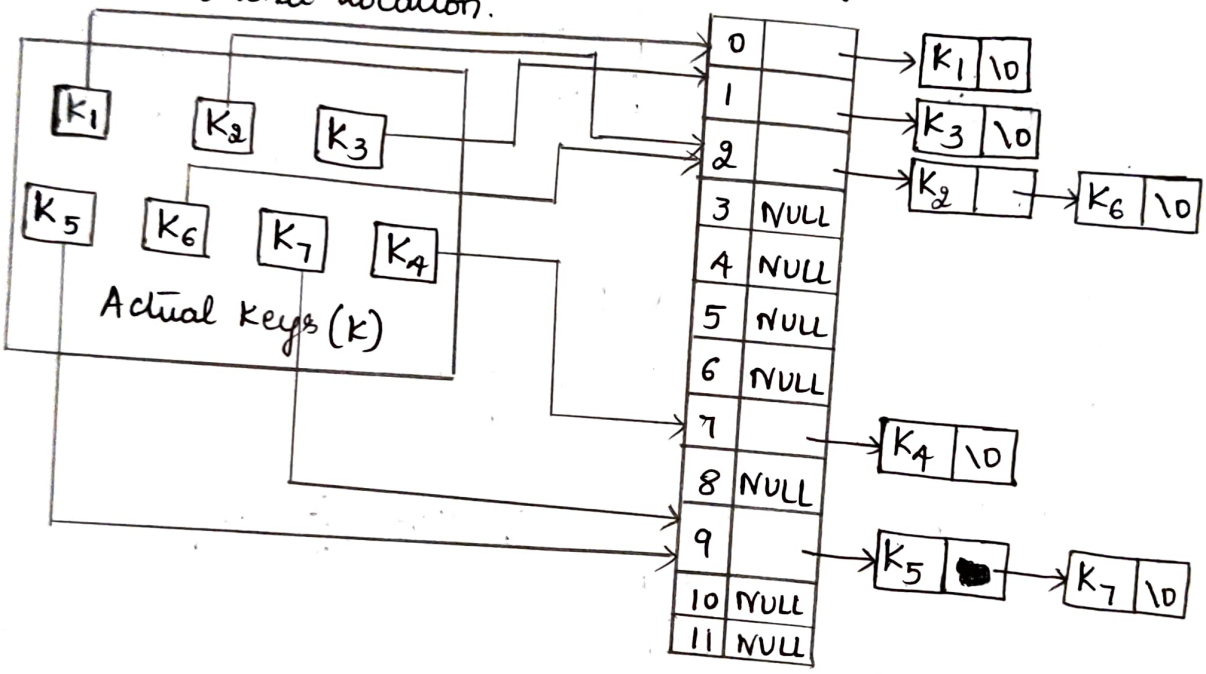


Fig :- Keys being hashed to a chained hash table

Eg :-

1) Insert the keys 7, 24, 18, 52, 36, 54, 11 and 23 in a chained hash table of 9 memory location. Use $h(k) = k \bmod m$

$h(k) = k \bmod m$
where $m = 9$

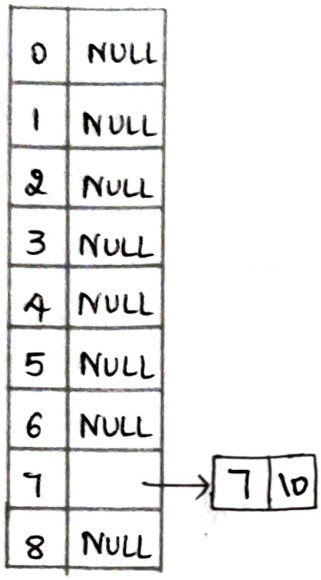
Initial hash table is as follows

0	NULL
1	NULL
2	NULL
3	NULL
4	NULL
5	NULL
6	NULL
7	NULL
8	NULL

i) Key 7

$h(7) = 7 \bmod 9$

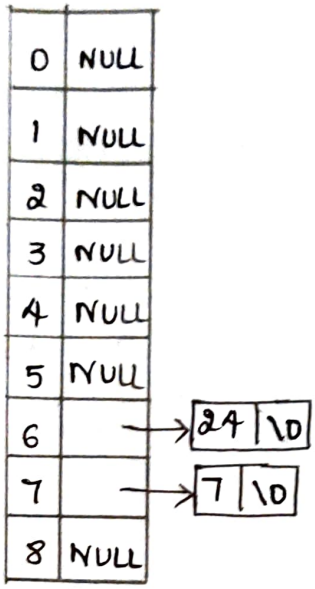
$= 7$



ii) Key 24

$h(24) = 24 \bmod 9$

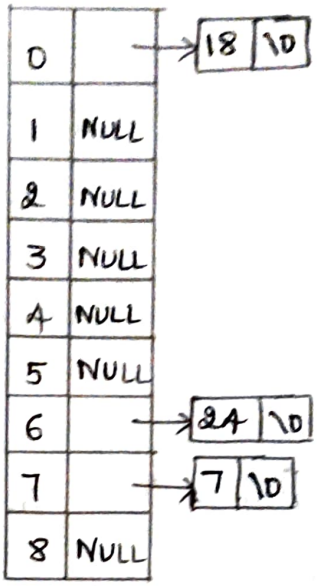
$= 6$



iii) Key 18

$h(18) = 18 \bmod 9$

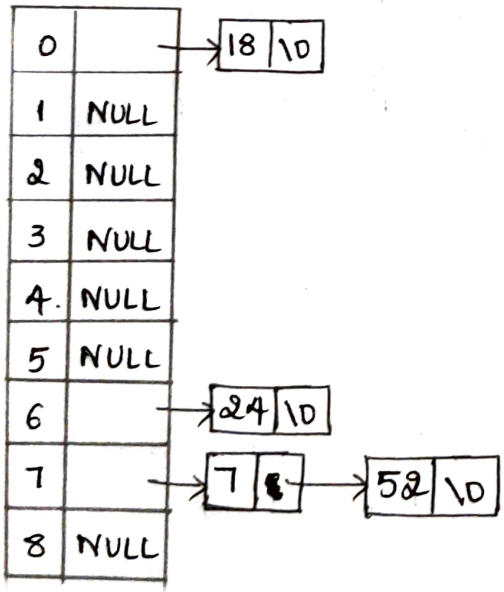
$= 0$



iv) Key 52

$h(52) = 52 \bmod 9$

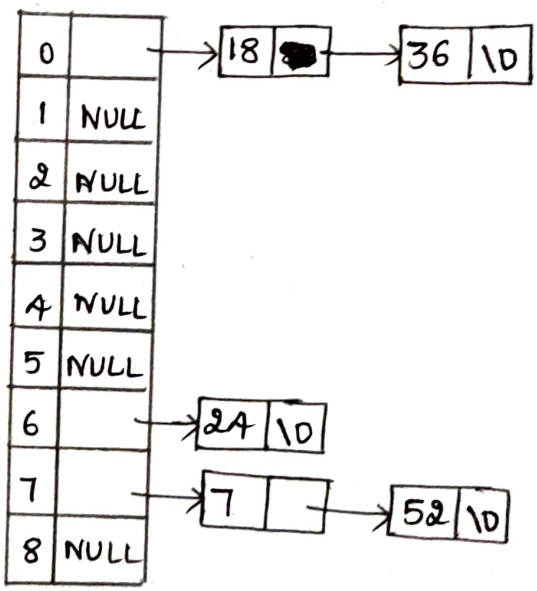
$= 7$



v) Key 36

$h(36) = 36 \bmod 9$

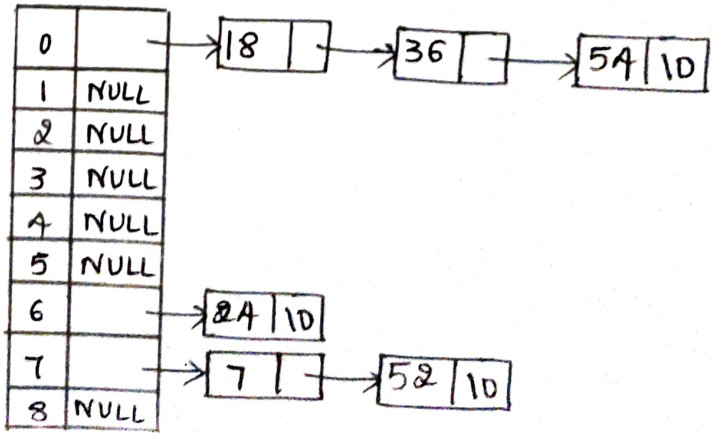
$= 0$



vi) Key 54

$h(54) = 54 \bmod 9$

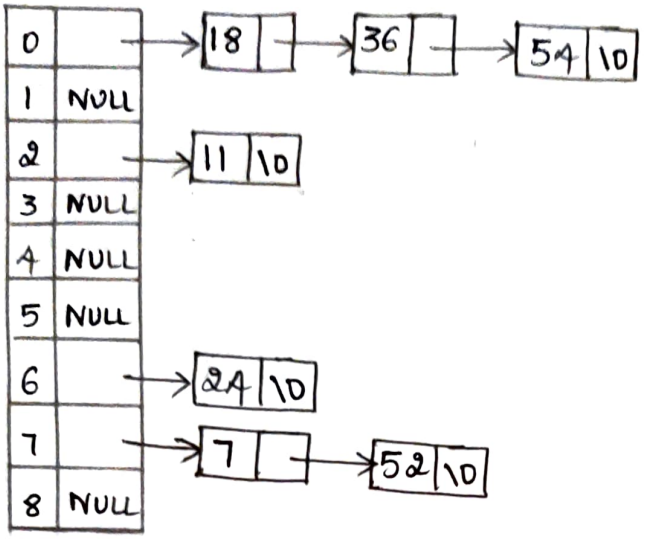
$= 0$



vii) key 11

$h(11) = 11 \bmod 9$

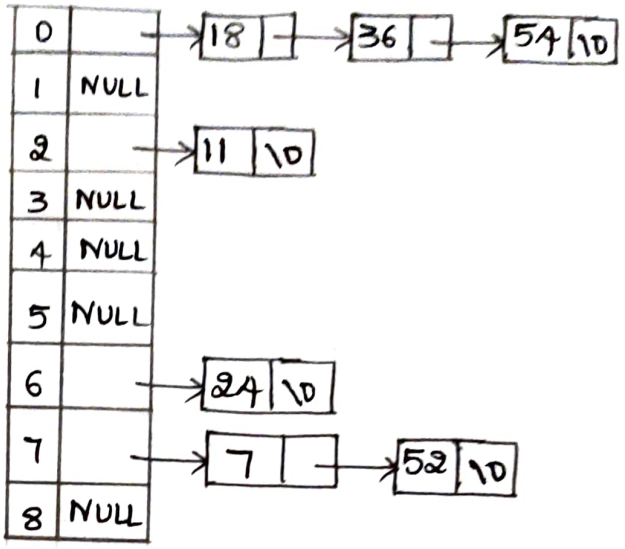
$= 2$



viii) key 23

$h(23) = 23 \bmod 9$

$= 5$



2) Construct a hash table of size 5 using chaining technique & store the following words. Use $h(K) = \text{sum} \% m$

like, a, tree, you, first, place, to.

$h(K) = \text{sum} \% m$ where $m = 5$,

Initial hash table looks as follows.

0	NULL
1	NULL
2	NULL
3	NULL
4	NULL

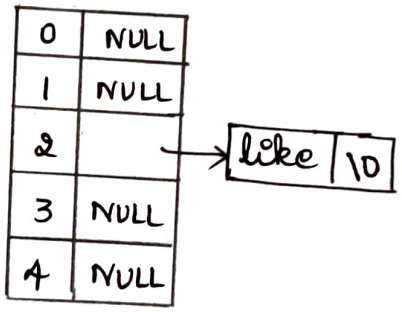
i) key like

$l + i + k + e = 12 + 9 + 11 + 5$

$= 37$

$h(\text{like}) = 37 \% 5$

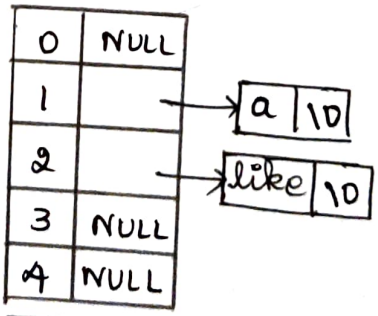
$= 2$



ii) key a

$a = 1$

$h(a) = 1 \% 5 = 1$



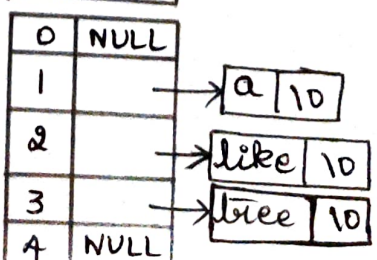
iii) key tree

$t + r + e + e = 20 + 18 + 5 + 5$

$= 48$

$h(\text{tree}) = 48 \% 5$

$= 3$



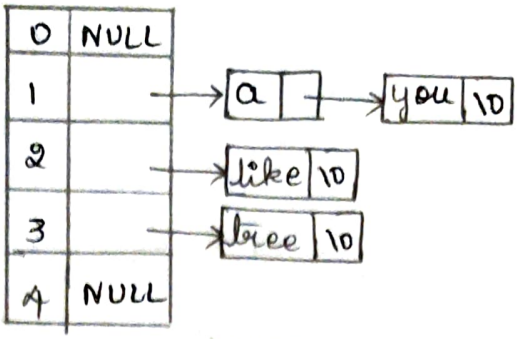
iv) key you

$y+o+u = 25+15+21$

$= 61$

$h(you) = 61 \% 5$

$= 1$



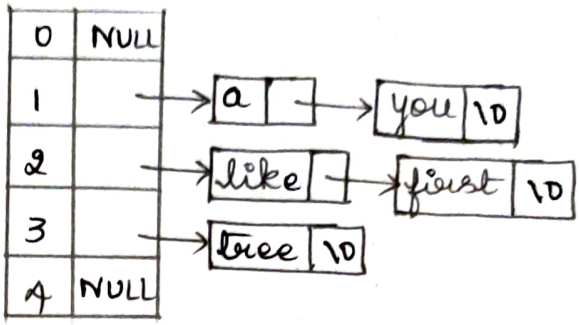
v) key first

$f+i+r+s+t = 6+9+18+19+20$

$= 72$

$h(first) = 72 \% 5$

$= 2$



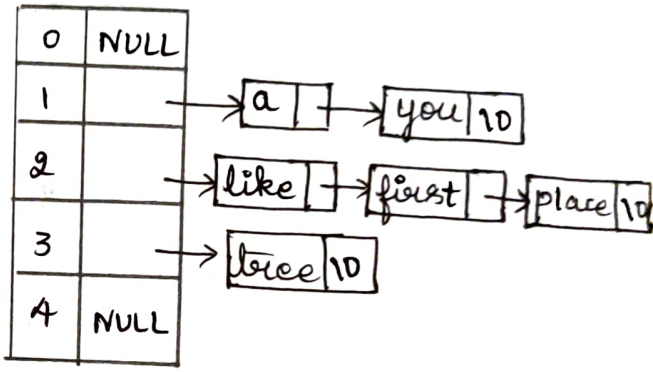
vi) key place

$p+l+a+c+e = 16+12+1+3+5$

$= 37$

$h(place) = 37 \% 5$

$= 2$



vii) key to

$t+o = 20+15$

$= 35$

$h(to) = 35 \% 5$

$= 0$