UNIT-VI

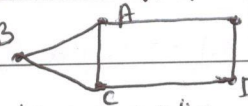
Syllabus :-

Dictionaries: definition, ADT, linear list representation, operations - insertion, deletion and searching, Hash table representation/ Hash Function - Devision Method, Collision resolution techniques, Separate chaining, Open addressing - linear Probing, quadratic Probing double hashing, Rehashing.
Graphs :- Graph terminology, representation of graphs, Graph Traversals : BFS (Breadth First Search) - DFS (Depth First Search). minimum Spanning Tree.

Graph :- A graph is a non-linear data structure. A graph 'G' consists of two things:
1) A set 'V' of elements called 'nodes' (or Points or Vertices)
2) A set 'E' of edges such that each edge e in E is identified with a unique (unordered) pair [u,v] of nodes in V, denoted by e=[u,v]

Sometimes we indicate the graph using G=(V,E)

Terminologies :-



V = {A, B, C, D, E}
E = {AB, BC, CD, DE, AE, CE, AC}

1. The degree of a node u, written as deg(u), is the number of edges containing u.
If deg(u)=0 i.e u does not belong to any edge. Then it is called an "isolated node".

---

2. The Path P of length n from a node u to a node v is defined as a sequence of n+1 nodes.
$P = (V_0, V_1, V_2 \ldots V_n)$
The path P is said to be closed if $V_0 = V_n$.
The path P is said to be simple if all the nodes are distinct.
A cycle is a closed simple path with length 3 or more. A cycle of length k is called a k-cycle.
3. A graph G is said to be connected if there is a path between any two of its nodes.
4. A graph G is said to be labelled if its edges are assigned data.
G is said to be weighted if each edge 'e' in G is assigned a non-negative numerical value w(e) called the weight or length of 'e'.
5. Multiple Edges: Distinct edges e and e' are called multiple edges if they connect the same end points. ie e=[u,v] and e'=[u,v]
6. An edge 'e' is called a "loop" if it has identical end points
i.e e=[u,u]
7. A graph is called "multigraph" if it ~~does not~~ allow either multiple edges or loops
8. A graph is called "finite" if it has a finite number of nodes and finite number of edges.

Directed Graph:- A graph G is called a digraph or graph is same as a multi-graph except that ~~edge~~ each edge e in G is assigned a direction. Or in other words, each edge 'e' is identified with an ordered pair (u,v) of nodes in G rather than an unordered pair [u,v].

The "outdegree" of a node u in G, written outdeg(u), is the number of edges beginning at u.
The "indegree" of u, written indeg(u), is the number of edges ending at u.

A node 'u' is called a "source" if it has a positive outdegree but zero indegree. Similarly, u is called a "sink" if it has a zero outdegree, but a positive indegree.
A node 'v' is said to be reachable from a node 'u' if there is a path from u to v.

A directed graph G is said to be connected or strongly connected, if for each pair u,v of nodes in G, there is a path from u to v and also exists a path from v to u. This is also called as unilaterally connected graph.

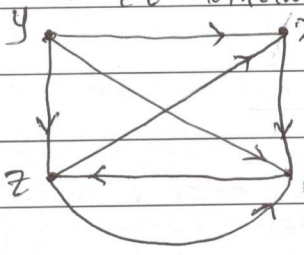
Directed Acyclic Graph (DAG): It is a directed graph with no cycles

Representation of Graphs:-

1. Adjacency Matrix:- Suppose G is a simple directed graph with 'm' nodes and suppose the nodes of G have been ordered and are called $V_1, V_2, \ldots, V_m$. Then the adjacency matrix $A = (a_{ij})$ of the graph G is the $m \times m$ matrix defined as follows:

$$a_{ij} = \begin{cases} 1 & \text{if } V_i \text{ is adjacent to } V_j \text{ i.e if there is an edge } (V_i, V_j) \\ 0 & \text{otherwise} \end{cases}$$

Example



Nodes of G Data = {x, y, z, w}

ordering of the nodes in G as follows:

$V_1 = x$; $V_2 = y$; $V_3 = z$; $V_4 = w$

$$\therefore A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

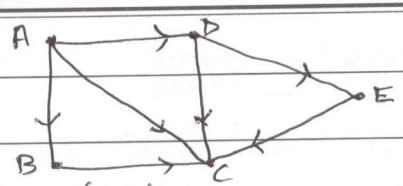
Problems in Adjacency Matrix:-
1) Insertion is difficult.
2) Space wastage

NOTE:- This method is used when the graph is dense.
i.e if in the graph every node connects to more number of other nodes in the graph

linked representation of a graph:- (lists)

This is also called as "adjacency structure".

Consider the following graph G:



Graph-G

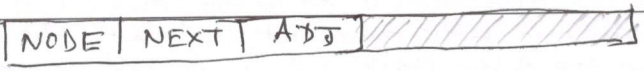
NODE	Adjacency list
A	B, C, D
B	C
C	
D	C, E
E	C

Adjacency list of G.

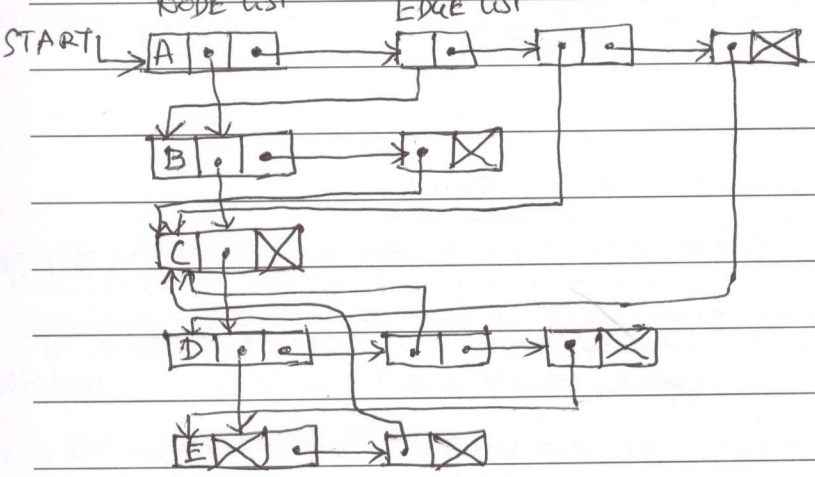
The linked representation of the graph include two lists

1) A node list
2) An Edge list as shown as follows:

1) A Node list:-



Here "NODE" will be the name or key value of the node;
"NEXT" will be a pointer to the next node in the list Node.
"ADJ" will be a pointer to the first element in the adjacency list of the node which is maintained in the EDGE.
shaded part can be used to maintain INDEG, OUTDEG, STATUS ... etc.



NOTE:
This method is used when the graph is sparce.
i.e the graph includes less no of edges between the nodes

2. EDGE List:-

Where "DEST" will be point to the location in the list NODE
"LINK" will link together the edges with the same initial node

Graph Traversals: It is a technique used for searching a vertex in a graph. It is also used to decide the order of vertices to be visited in the search process.

Two types of graph traversals { used DS for implementation

1. BFS (Breadth First Search) — Queue
2. DFS (Depth First Search) — Stack

1) DFS (Depth first Search): This traversal of a graph, produces "spanning tree" as final result. i.e a graph without loops. We use Stack data structure with max size of total number of vertices in the graph to implement

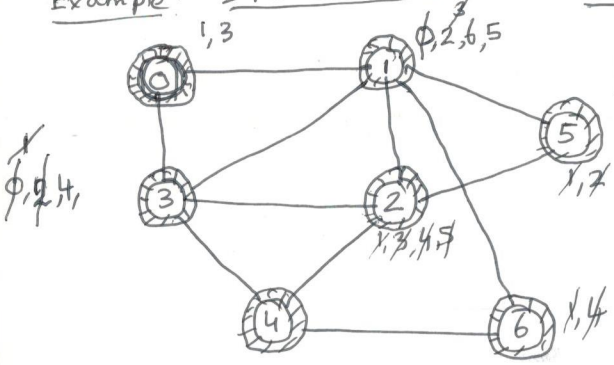
Algorithm:-

Step 1: Define a stack of size total no of vertices in graph.
2. Select any vertex as starting point for traversal. Visit that vertex and push it on to the stack.
3. Visit any one of the adjacent vertex of the vertex which is at the top of the stack, which is not visited and push it on to the stack.
4. Repeat step-3 until there are no new vertex to be visited from the vertex on the top of the stack.
5. When there is no new vertex to be visited then use backtracking and pop one vertex from the stack.
6. Repeat steps 3,4,5 until stack becomes empty.
7. When stack becomes empty, then produce final spanning tree by removing unused edges from the graph.

2) BFS (level order traversal): Algorithm:

Step 1: Define a queue size equal to the no of vertices of graph.
2: Select any vertex as starting point for traversal. Visit that vertex and insert it into the queue.
3: Visit all the adjacent vertices of the vertex which is at front of the queue, which is not visited and insert them into the queue.
4: When there is no new vertex to be visited from the vertex at front of the queue then delete that vertex from the queue.
5: Repeat steps 3 and 4 until queue becomes empty.
6. When queue becomes empty, then produce final spanning tree by removing unused edges from the graph.

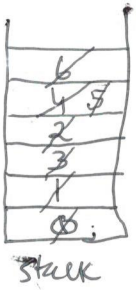
Example: BFS Traversal:-



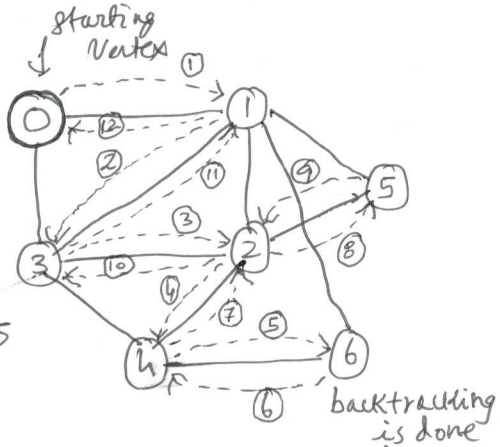
Queue | 0 | 1 | 3 | 2 | 5 | 6 | 4 |

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

DFS Traversal: (Stack)



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

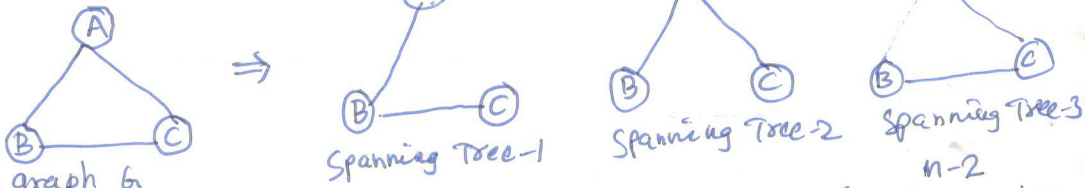


Spanning Tree:- A spanning tree is a subset of graph 'G', which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it can't be disconnected.

NOTE:- 1) Every connected and undirected graph 'G' has at least one spanning tree.

2) A disconnected graph does not have any spanning tree, as it can't be spanned to all its vertices.

Ex:- The following is the graph 'G' with the three possible Spanning trees:



$n-2$

∴ A complete undirected graph can have maximum of n number of spanning trees; where n is the number of vertices in the graph 'G'.

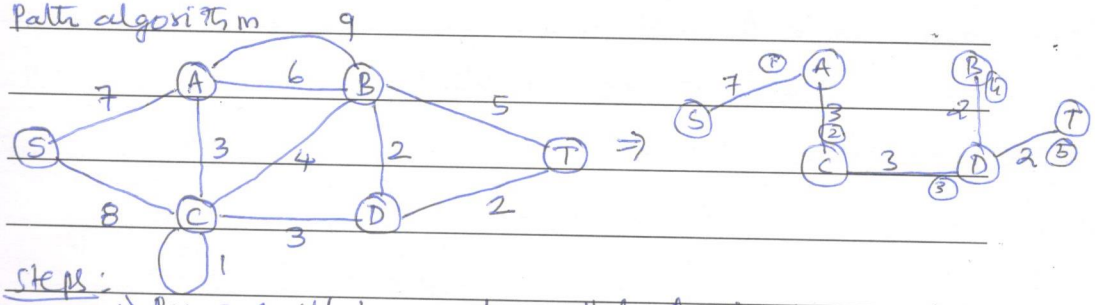
2. Spanning tree has n-1 edges
3. From a complete graph, by removing e-n+1 edges, we can construct a spanning tree.

Applications :-

1) Civil network planning
2) Computer network routing protocol
3) Cluster analysis

Minimum Spanning Tree :- In a weighted graph, a minimum spanning tree ~~of the graph~~ is a spanning tree that has minimum weight than all other spanning trees of the same graph.

Prim's Alg to find minimum spanning :- This is also called as shortest path algorithm → Edge based approach



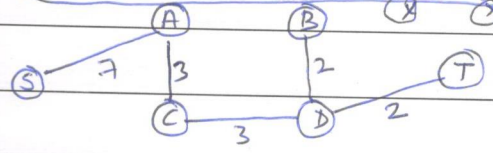
Steps :
1) Remove all loops and parallel edges in the graph
2) Choose any arbitrary node as the root node
3) Check outgoing edges and select the one with less cost.

Kruskal's Alg :- Node/Vertex based approach.

Steps : 1) Remove all loops and parallel edges
NOTES- In case of parallel edges, keep the edge which has less cost.
2) Arrange all edges in their increasing order of weight
3) Add the edge which has the least weightage.
NOTE :- In case, by adding one edge if a cycle/circuit forms then ignore that edge.

The algorithm starts by treating the graph as a forest where every node is an individual tree.

B,D	D,T	A,C	C,D	C,B	B,T	A,B	S,A	S,C
2	2	3	3	4	5	6	7	8
				X	X	X		X



Dictionaries :-

A "dictionary" is a general purpose data structure for storing a group of objects. A dictionary has a set of keys and each key has a single associated value. When represented with a key, the dictionary will return the associated value.

Example:- To represent the results of a classroom test:

results = {'0561': 81, '0562': 70, '0563': 60, ... ; '05C0': 75}

Instead of using the numerical index of the data, we can use dictionary names to return the values

Ex:- results['0562'] returns 70
results['05C0'] returns 75

A dictionary is also called "a hash", "a map" "a hashmap"
The concept of a key-value store is widely used in various computer systems such as caches and high-performance databases.
Keys in a dictionary must be unique

A dictionary is an ADT (Abstract Data Type)

Basic Operations:

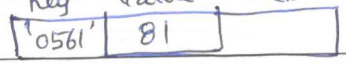
1. Insert(x, D) : Insertion of element x (Key & Value) in dictionary D.
2. Delete(x, D) : Deletion of element x (Key & Value) in D with the help of Key 'k'.
3. Search(x, D) : Search for x in D
4. member(x, D) : Returns true if $x \in D$ else return false
5. size(D) : Returns the count of elements in D

Implementation of Dictionary :-

1) Using Arrays — sorted list
2) Linked lists — Skip lists
3) Hashing
4) Trees — BST (Binary Search Tree)
   Balanced Tree — AVL Tree, Red-Black Tree
   Splay Trees
   Multi-way search Trees

Using lists :-

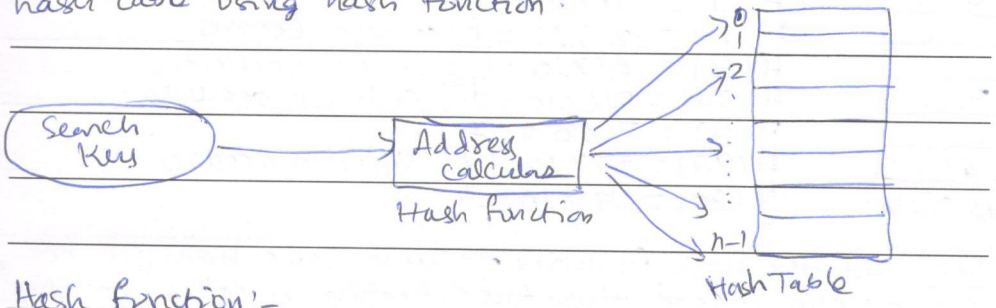
NODE structure:




```
Struct node
{
  char key[30];
  int value;
  struct node *next;
} head;
```

Hash Table:- Hash table (or hash map) is a data structure used to store and retrieve data very quickly.
Insertion of data in the hash table is based on the Key-value. Every entry in the hash table is associated with some Key.

Hashing:- It is the process of mapping large amount of data items to a small table with the help of "Hash function".
i.e we can place the dictionary entries (Key, Value) in the hash table using hash function.



Hash function:-

A hash function is any function that can be used map a dataset of any arbitrary size to a data set of a fixed size which falls into the hash table.

The value returned by the hash function is called "hash-key".

One common method for determining the hash key is "division method" or "Hashing"

Syntax:-

Hash key = key % size of the table

Ex:- Consider the table with 8 entries to insert. Insert the following elements into the table using the following Hash function

$H(x) = x \% \text{ size of the table}$

input elements : {36, 18, 72, 43, 6}

0	72
1	
2	18
3	
4	36
5	
6	6
7	

$H(36) = 36 \% 8 = 4$
$H(18) = 18 \% 8 = 2$
$H(72) = 72 \% 8 = 0$
$H(6) = 6 \% 8 = 6$

Example: 2 Collision!-

Data elements: {131, 3, 4, 21, 61, 24, 7, 97, 89}

Size of the table: $t_s = 10$

0	
1	131
2	
3	3
4	4
5	
6	
7	7
8	
9	89

Hash Table

Hash Function $H(Key) = Key \,\%\, Size$.

$H(131) = 131 \,\%\, 10 = 1$

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

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

$H(21) = 21 \,\%\, 10 = [1]$ collision occures

$H(61) = 61 \,\%\, 10 = [1]$ collision occures

$H(24) = 24 \,\%\, 10 = [4]$ collision occures

$H(7) = 7 \,\%\, 10 = 7$

$H(97) = 97 \,\%\, 10 = [7]$ collision occures

$H(89) = 89 \,\%\, 10 = 9$

Definition!- If the hash function returns same hash key for more than one record, then the situation is called "collision" when there is no room (memory location) for new pair (Key, Value) in the hash table then that situation is called "overflow". Some times when we handle collision in the hash table it may lead to "overflow".

Collision and overflow situations in the hashing leads to a poor "hash function design"

Collision resolution techniques: The techniques used to resolve the problem of collision in hashing procedure is called collision resolution techniques. They are classified as:

methods

- open Hashing (closed Addressing) — Ex:- Separate chaining
- closed Hashing (open Addressing) — Ex:- 1. Linear Probing, 2. Quadratic Probing, 3. Double Hashing

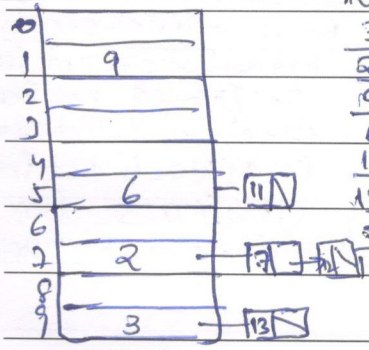
Hashing is a search technique takes $O(1)$

Search techniques like linear or binary search takes the time of $O(n)$ and $O(\log n)$ respectively. It is purely depends on the value of n.

We want to develop a technique which does not depend on 'n' but which takes constant amount of time. Here we introduce "hashing" which takes constant amount of time irrespective of the size of 'n'.

1. Chaining Method:

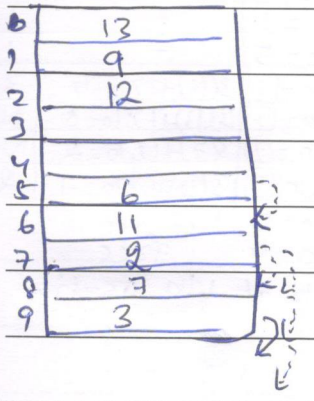
Key values: {3, 2, 9, 6, 11, 13, 7, 12} Use division method and closed addressing (open Hashing) to store these values.
Hash function: $h(k) = 2k+3$ where $m=10$ (size of Hash table)



Key	Location (u)	Prob	→ times of search for place/inquiry
3	$(2(3)+3)\%10 = 9$	1	
2	$(2(2)+3)\%10 = 7$	1	
9	$(2(9)+3)\%10 = 1$	1	
6	$(2(6)+3)\%10 = 5$	1	
11	$(2(11)+3)\%10 = [5]$	2	
13	$(2(13)+3)\%10 = [9]$	2	
7	$(2(7)+3)\%10 = [7]$	2	
12	$(2(12)+3)\%10 = [7]$	3	

2. Linear Probing Method. [During collision insert K_i at first free location from $(u+i)\%m$ where $i = 0$ to $(m-1)$]

Key values: {3, 2, 9, 6, 11, 13, 7, 12} Use division Method and open addressing (closed Hashing) to store these values
$h(k) = 2k+3$ $m=10$



Key	Location (u)	Prob	for 11 (K)
3	$2(3)+3 \%10 = 9$	1	
2	$2(2)+3 \%10 = 7$	1	
9	$2(9)+3 \%10 = 1$	1	
6	$2(6)+3 \%10 = 5$	1	
11	$2(11)+3 \%10 = [5]$	2	$(5+0)\%10 = 5$
13	$2(13)+3 \%10 = [9]$	2	$(5+1)\%10 = 6$
7	$2(7)+3 \%10 = [7]$	2	6 is free location
12	$2(12)+3 \%10 = [7]$	6	∴ insert 11 at 6 in Hash table

$16/8 = 2$ (Average Prob)

order of elements stored in Hash table is: 13, 9, 12, _, _, 6, 11, 2, 7, 3

Quadratic Probing:-

During collision insert K_i at first free location from $(u+i^2)\%m$ where $i = 0$ to $(m-1)$

input elements $A = \{3,2,9,6,11,13,7,12\}$ Use division method & quadratic probing to store these values with $H(K) = 2K+3$

0	13
1	9
2	
3	12
4	
5	6
6	11
7	2
8	7
9	3

For 11 collision occures
$(5+0)\%10 = 5$ Not free
$(5+1)\%10 = 6$ free
∴ allocate 11 in 6th loc

For 13 collision occures
$(9+0)\%10 = 9$ not free
$(9+1)\%10 = 0$ free
∴ allocate 13 at 0th position

For 7 collision occures
∴ $(7+0)\%10 = 7$ not free
$(7+1)\%10 = 8$ free.
∴ allocate 7 at 8th position

Key	location(u)	Prob
3	$2(3)+3\%10 = 9$	1
2	$2(2)+3\%10 = 7$	1
9	$2(9)+3\%10 = 1$	1
6	$2(6)+3\%10 = 5$	1
11	$2(11)+3\%10 = 5$	2
13	$2(13)+3\%10 = 9$	2
7	$2(7)+3\%10 = 7$	2
12	$2(12)+3\%10 = 7$	5
		15

For 12 collision occures
$(7+0)\%10 = 7$ not free
$(7+1)\%10 = 8$ not free
$(7+4)\%10 = 1$ not free
$(7+9)\%10 = 6$ not free
$(7+16)\%10 = 3$ free
∴ Allocate 12 at 3rd position

Order of elements: 13, 9, –, 12, –, 6, 11, 2, 7, 3

Double Hashing:-

$A = \{3,2,9,6,11,13,7,12\}$ Use division method & Double Hashing technique to insert these elements where $h(K) = 2K+3$ and $h_2(K) = 3K+1$ & $M = 10$

During collision insert K_i at the first free location from $(u+v*i)\%m$ where $i = 0$ to $(m-1)$

0	
1	9
2	
3	11
4	12
5	5
6	
7	2
8	
9	3

(11) $(5+4*0)\%10 = 5$ not free
$(5+4*1)\%10 = 9$ not free
$(5+4*2)\%10 = 3$ free

(13) $(9+0*0)\%10 = 9$
$(9+0*1)\%10 = 9$
∴ not possible to insert

(7) $(7+2*0)\%10 = 7$ not free
$(7+2*1)\%10 = 9$ not free
$(7+2*2)\%10 = 1$ not free
$(7+2*3)\%10 = 3$ not free
$(7+2*4)\%10 = 5$ not free
$(7+2*5)\%10 = 7$ not free
6, 7, 8
∴ regenerates same key values ∴
not possible to insert

Key	location(u)	location(v)	Probs
3	$2(3)+3\%10 = 9$	–	1
2	$2(2)+3\%10 = 7$	–	1
9	$2(9)+3\%10 = 1$	–	1
6	$2(6)+3\%10 = 5$	–	1
11	$2(11)+3\%10 = 5$	$3(11)+1\%10 = 4$	3
(13)	$2(13)+3\%10 = 9$	$3(13)+1\%10 = 0$	NA
(7)	$2(7)+3\%10 = 7$	$3(7)+1\%10 = 2$	NA
12	$2(12)+3\%10 = 7$	$3(12)+1\%10 = 7$	2

(12) $(7+7*0)\%10 = 7$ Not free
$(7+7*1)\%10 = 4$ free
∴ insert 12 at 4th location

Rehashing:- It is another technique to resolve the collisions in hash table. The relationship between the number of elements (n) and the size of Hash table (M) is represented with load factor (λ) as $\lambda = \frac{n}{M}$

The ideal value of λ should be "0.75" or "<1"

when λ is > 0.75 or >1 then use rehashing technique. Rehashing means "hashing again". When the load factor λ is greater than the threshold value it leads to more time complexity and also leads to more collisions. Hence we need to increase the size of hash table that can prevent collisions as well it maintain good time complexity equal to O(1).

During rehashing the following things will be done.

1) Increase the size of the hash table from M to M'
2) Modify the hash function from H(x) to H'(x)
3) Apply H'(x) on the elements to insert

Generally M' should be the number closest Prime No to 2M

Ex:- Insert the elements {6, 7, 8} into Hash table of size 3 with $H(x) = x \% \text{size}$

	Key
0	6
1	7
2	8

$H(6) = 6 \% 3 = 0$
$H(7) = 7 \% 3 = 1$
$H(8) = 8 \% 3 = 2$

Here $\lambda = 1 \left(\because \frac{3}{3} = 1\right)$

Select M' close to 2M i.e 7

$\therefore M' = 7 \quad H'(x) = x \bmod 7$

	Key
0	7
1	8
2	
3	
4	
5	
6	6

$H(6) = 6 \% 7 = 6$
$H(7) = 7 \% 7 = 0$
$H(8) = 8 \% 7 = 1$

load factor $= \frac{3}{7} = 0.42$