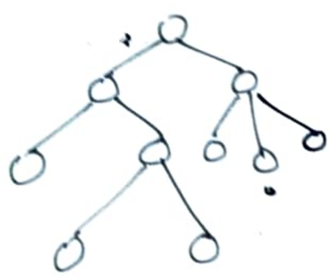
GRAPHS.

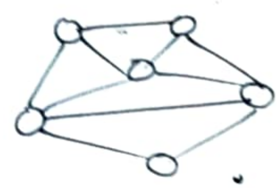
Introduction

→ Graph is another information important Non-linear Data Structure.

In tree structure, there is a hierarchial relationship b/w parent and children, that is, One parent and many children. On the other hand, in graph, relationship is less restricted.

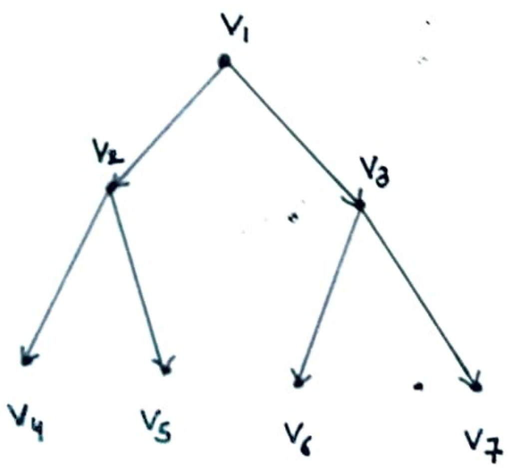


Tree

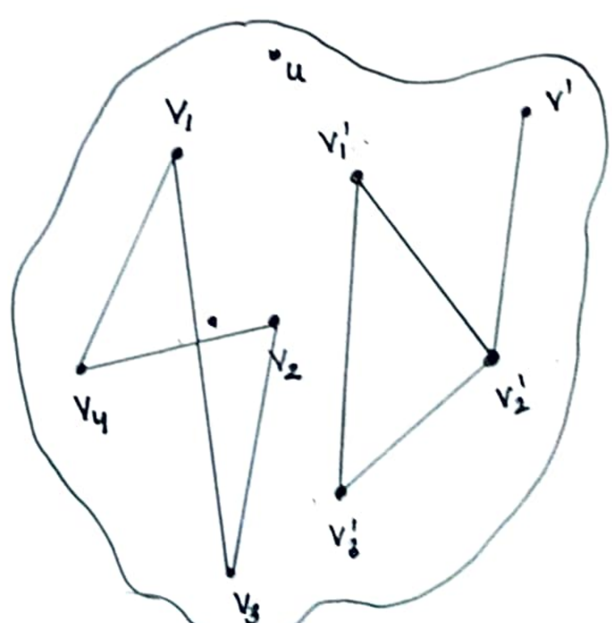


Graph

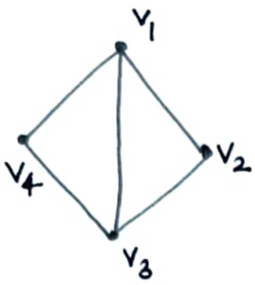
Malay Tripathi



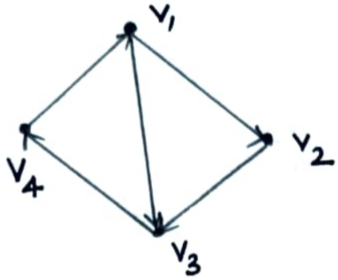
G_7



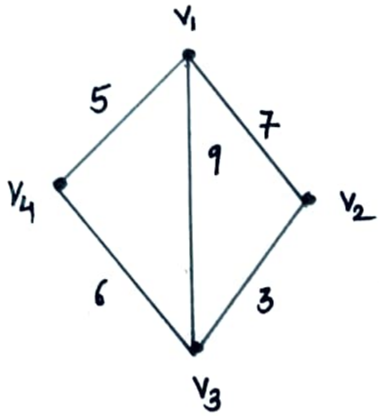
G_8



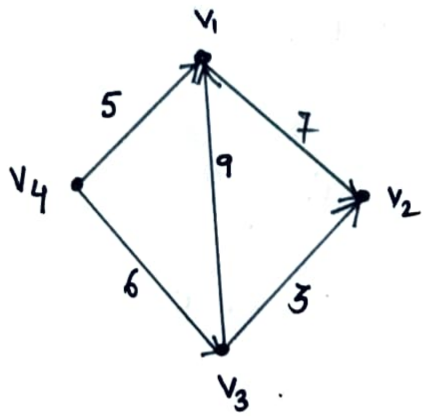
G_1



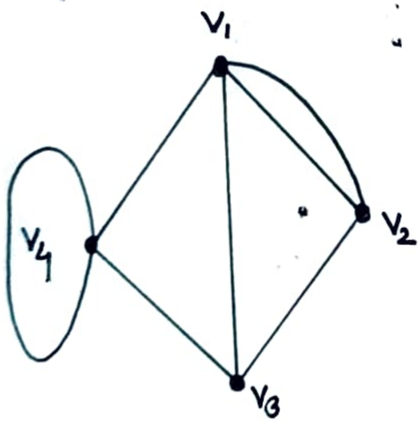
G_2



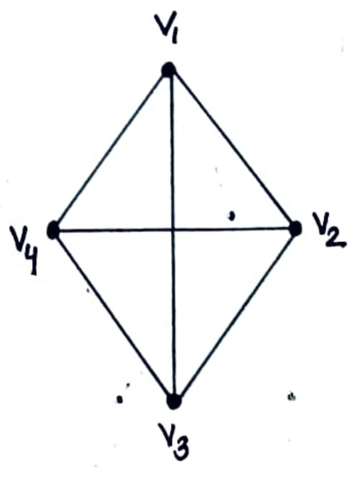
G_3



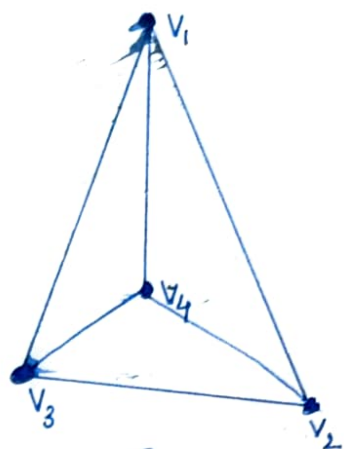
G_4



G_5



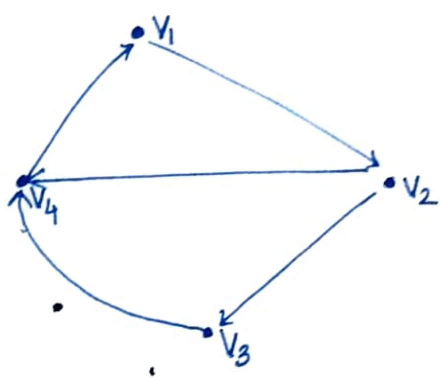
G_6



G_9

G_{10}

MALAY
TRIPATHI



G_4

graph terminology

(1). Graph → A graph G consists of two sets :→

(i) A set V, called the set of all vertices (or nodes)

(ii) A set E, called the set of all edges (or arcs). This set E is the set of all pairs of elements from V.

for example, let us consider the graph G_1 in figures of graphs

$V = \{V_1, V_2, V_3, V_4\}$ G_1

$E = \{(V_1, V_2), (V_1, V_3), (V_1, V_4), (V_2, V_3), (V_3, V_4)\}$

(2) here, if an order pair (V_i, V_j) is in E then there is an edge directed from V_i to V_j → it is known as DIRECTED GRAPH.

A diagraph is also called a DIRECTED GRAPH. It is a graph G, such that, $G = \langle V, E \rangle$, where V is the set of all vertices and E is the set of ordered pair of elements from V.

Malay

$V = \{V_1, V_2, V_3, V_4\}$ G_2

$E = \{(V_1, V_2), (V_1, V_3), (V_2, V_3), (V_3, V_4), (V_4, V_1)\}$

Weighted Graph:→ A graph (or diagraph) is termed weighted graph if all the edges in it are labelled with some weights.

for example, G_3 and G_4 are two weighted graphs.

Parallel edges→ If there is an edge, more than one, b/w the pair of vertices, then they are known as Parallel Edges. for example, there are two parallel edges b/w v_1 and v_2 in graph G_5.

A graph which has either self loop or parallel edges or both, is called Multigraph. G_5 is thus a multigraph.

Malay

→ Simple Graph (Digraph) → A graph (digraph) if it does not have any self loop or parallel edges is called a Simple Graph (Digraph). All graphs (digraphs) except G_5 and G_{10} are Simple.

→ Complete Graph ⟶ A graph (digraph) G is said to be complete if each vertex v_i is adjacent to every other vertex v_j in G.

In other words, There are edges from any vertex to all other vertices.

For example, G_6 and G_9 are two complete graph.

→ Connected Graph →

In a graph (not digraph) G, two vertices v_i and v_j are said to be connected, if there is a path in G from v_i to v_j (or v_j to v_i). A graph is said to be connected if for every pair of distinct vertices v_i, v_j in G, there is a path ~~and also from v_j to v_i~~. For ex G1, G3 and G6 are connected graphs but G8 is not.

- A digraph G is said to be strongly connected if for every pair of distinct vertices v_i, v_j in G, there is a directed path from v_i to v_j and also from v_j to v_i. for example, digraph G_{11} is strongly connected but digraphs G_{10} and G_{12} are not.
- If a digraph is not strongly connected but the underlying graph (without direction of the edges) is connected, then the graph is said to be weakly connected.

Malay

→ Acyclic Graph :→

If there is a path containing one or more edges which starts from a vertex v_i and terminates into the same vertex then the path is known as a cycle.

For example → There is a cycle in both G_1 and G_2.

If a graph (digraph) does not have any cycle then it is called Acyclic Graph. For example G_4 and G_7 are two acyclic graph.

→ Isolated vertex → A vertex is isolated if there is no edge connected from any other vertex to the vertex. For example, in G_8 the vertex u is an isolated vertex.

→ Degree of Vertex → The no. of edges connected with vertex v_i is called the degree of vertex v_i and is denoted by degree (v_i).
For example, degree $(v_i) = 3$, $\forall\, v_i \in G_6$.

But for a digraph, There are two degree → INDEGREE AND OUTDEGREE.

Indegree → v_i denoted as indegree (v_i) = no. of edges incident into v_i.

Outdegree → no. of edges emanating from v_i.

Malay

For ex → G_4

indegree $(v_1) = 2$ outdegree $(v_1) = 1$
indegree $(v_2) = 2$ outdegree $(v_2) = 0$
indegree $(v_3) = 1$ outdegree $(v_3) = 2$
indegree $(v_4) = 0$ outdegree $(v_4) = 2$.

→ Pendant Vertex → A vertex v_i is pendant if its indegree $(v_i) = 1$ and outdegree $(v_i) = 0$. For example in G_8 v' is a pendant vertex. In G_7, there are four pendant vertex v_4, v_5, v_6 and v_7.

Representation of Graphs

(1). Set Representation

(2). Linked Representation.

(3). Sequential (matrix) Representation.

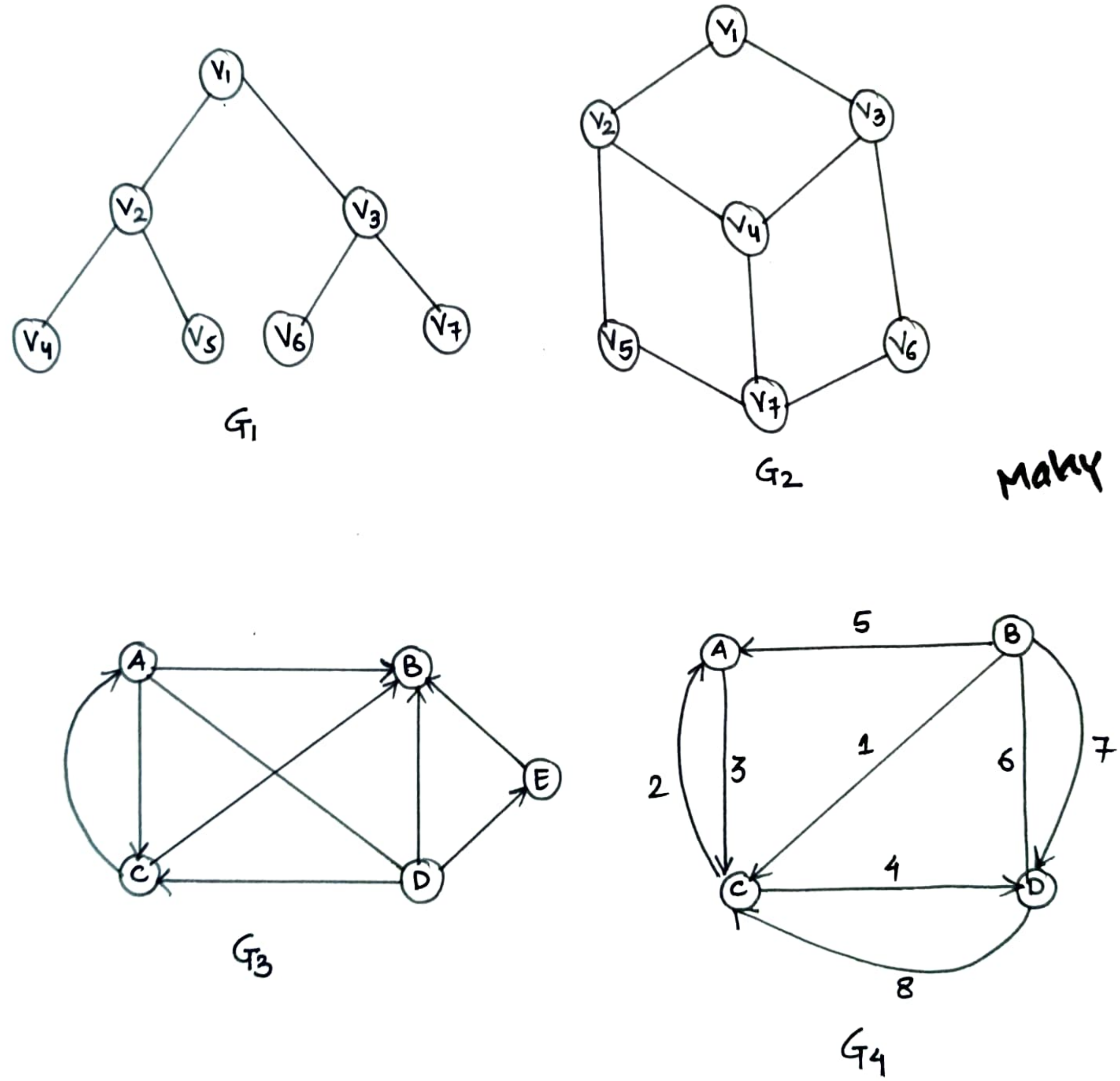


FIGURE 2. Types of Graph

Linked Representation

(a)

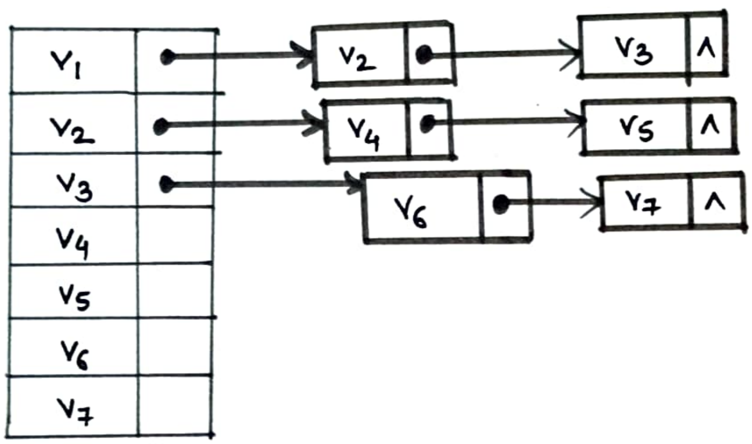
NODE_LABEL	ADJ_LIST

Node structure for non-weighted graph

(b).

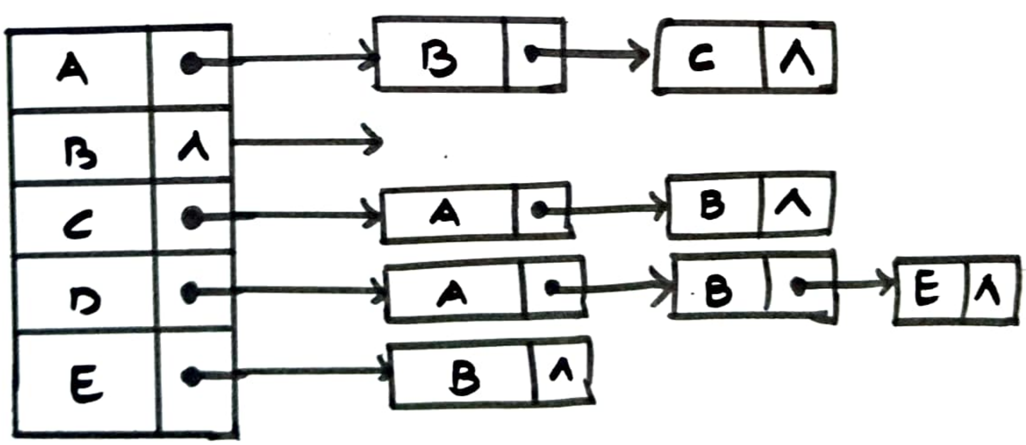
WEIGHT	NODE_LABEL	ADJ_LIST

Node structure for weighted graph



Malay

Representation of graph G_1



Representation of Graph G_3

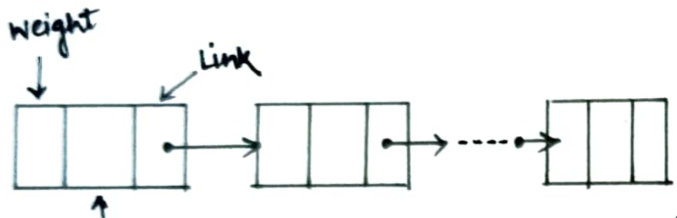
Operations on Linked List Rep. of Graphs

In order to generalize the implementation, we will assume two data structures in this representation.

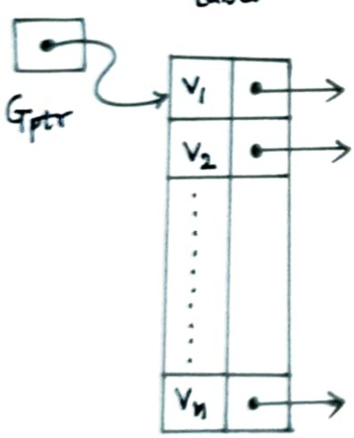
→ One is an Array of vertices. → has two fields: LABEL - label for the vertices. LINK → pointer to the linked list.

→ Second is a Single Linked List. → to maintain the list of all adjacent vertices for any vertex v_i, for which it is meant. So, if n vertices are there in the graph, then n linked lists have to be maintained.

• A node structure has two fields other than the field LINK. The first field weight is to store the weight of the edge & the second field LABEL to store the vertex's label.



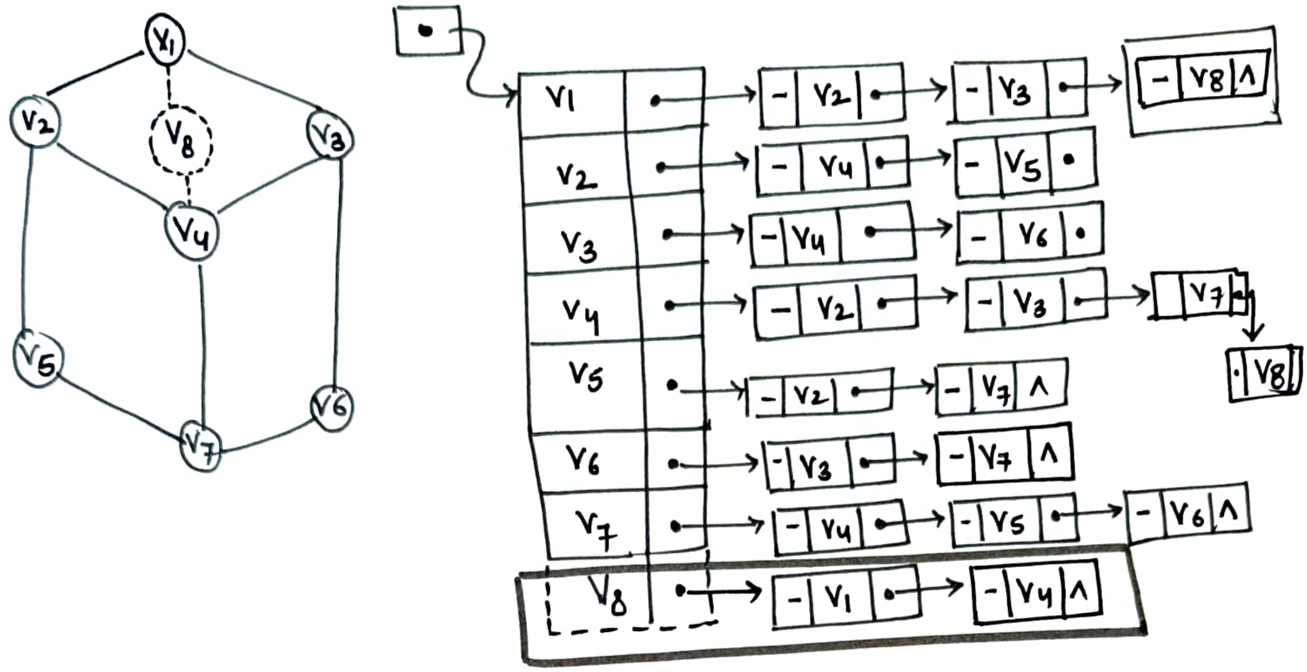
(b) Linked List



(a) An array of vertices

Malay

(1) Insertion : if we insert a vertex into a graph



LL-UG = Linked List in Undirected Graph.

Malay

Algorithm Insert Vertex_LL-UG

Input:- V_x, the new vertex that has to be inserted with $X = [V_{x1}, V_{x2}, \ldots, V_{xl}]$, l number of adjacent vertices to the vertex V_x

Let N be the number of vertices currently present in the graph.

Output:- A graph with new vertex V_x and its adjacent edges V_{xi}, $i = 1, 2, \ldots, l$, if the adjacent vertices exists

Data Structure :- Linked structure of Undirected Graph & UGPtr is the pointer to it.

Steps

1. N = N+1 , Vx = N // No. of vertices is increased by 1 and the label of the new vertex is N

/* To add the adjacency list of the new vertex, Vx in the graph */

2. For i = 1 to l do
3. Let j = X[i] // j is the label of ith adjacent vertex
4. If (j ≥ N) then // This label is not in graph.
5. print " No vertex labelled X[i] exists : Edge from Vx to X[i] is not established".
6. Else
7. InsertEnd-SL (UGptr[N] , X[i]) // Insert X[i] into the list of vertices.
8. InsertEnd-SL (UGptr[j], Vx) // To establish edges from X[i] to Vx
9. EndIf
10. EndFor
11. Stop.

Maly

Malay

Algorithm InsertEdge_LL-UG

Input: $\langle V_i, V_j \rangle$, the edge to be inserted b/w vertices V_i and V_j

Output: The graph with edge added b/w V_i and V_j

Data structure: Linked structure of Undirected Graph & UGptr is the pointer to it.

Steps:–

1. Let N = no. of vertices in the graph.
2. If (V_i > N) or (V_j > N) then
3. Print " Edge is not possible b/w V_i and V_j "
4. Else
5. InsertEnd-SL (UGptr[V_i], V_j) // Add V_j in the adjacency list of V_i
6. InsertEnd-SL (UGptr[V_j], V_i) // Add V_i in the adjacency list of V_j.
7. EndIf
8. Stop

Algorithm InsertEnd-SL. (node add. @ End of linked list).

Steps

Malay

1. new = GetNode (NODE)

2. If (new = NULL) then

3. Print "Memory is insufficient : Insertion not possible"

4. Exit

5. Else

6. ptr = HEADER

7. While (ptr → LINK ≠ NULL) do

8. ptr = ptr → LINK

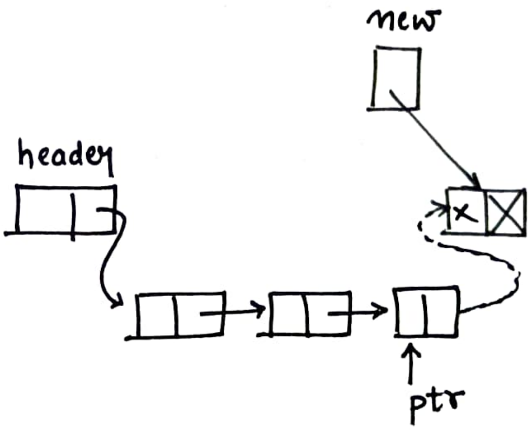
9. EndWhile

10. ptr → LINK = NEW

11. new → DATA = X

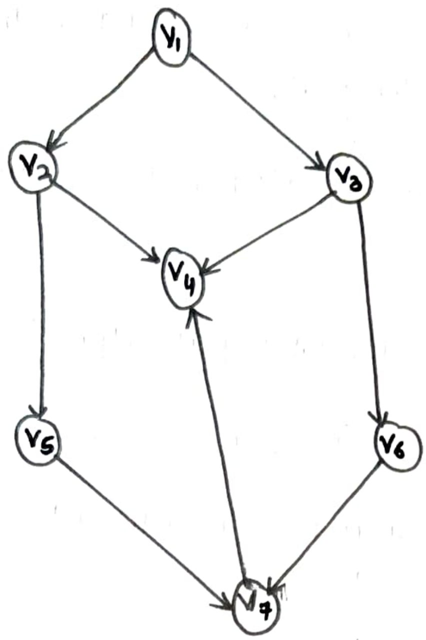
12. EndIf

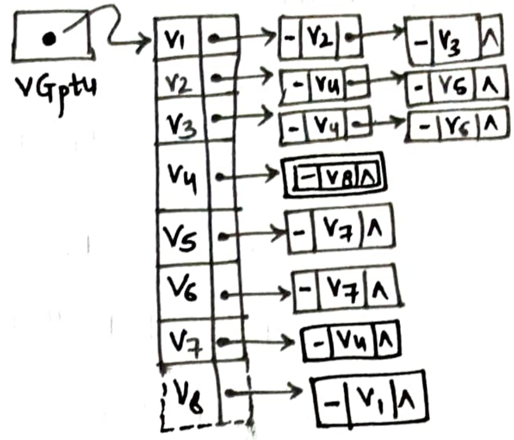
13. Stop



Inserting a node @ end of a single linked list

Insertion of vertex into a Directed Graph.





Algorithm InsertVertex

Malay

Input: V_x, the new vertex that has to be inserted.

$X = [V_{x1}, V_{x2}, V_{x3}, \ldots, V_{xm}]$, the list of adjacent vertices that the edges from V_x to V_{xi}, $i = 1, 2, \ldots, m$.

$Y = [V_{y1}, V_{y2}, V_{y3}, \ldots, V_{yn}]$, the list of adjacent vertices that has edges from V_{yi} $(i = 1, 2, \ldots, n)$ to V_x

Let N be the number of vertices currently present in the graph.

Output: A graph with new vertex V_x and directed edges from V_x to V_{xi} $i = 1, 2, \ldots,$ from V_{yi} $(i = 1, 2, \ldots, n)$ to V_x, if such V_{xi} and V_{yi} exists.

Steps:-

1\. N=N+1 , Vx = N // New Vertex Vx is counted as the next numbered in the graph.

/* To set the edge from Vx to Vxi , i = 1, 2, ..., m */

2\. for i=1 to m do

3\. Let j = X[i] // j is the label of ith adjacent vertex of Vx

4\. If j ≥ N then // j does not exists in the graph

5\. Print " No vertex labelled X[i] exists : Edge from Vx to X[i] is not established".

6\. Else → // Set the edge.

7\. InsertEnd_SL (DGptr [N], X [i])

Malay

8\. EndIf

9\. Endfor

/* To set edge from Vyi i = 1, 2, ..., n to Vx */

10\. For i=1 to n do

11\. Let j = Y [i] → // j is the label of ith adjacent vertex

12\. If j ≥ N then → // j does not exists in the graph

13\. Print " No vertex labelled Y[i] exists: Edge from Y[i] to Vx is not established" // set the edge

14\. Else

15\. InsertEnd_SL (DGptr[j], Vx)

16\. EndIf

17\. Endfor

18\. Stop.

Algorithm InsertEdge_LL_DG

Input : $\langle v_i, v_j \rangle$, the edge to be inserted from vertices v_i & v_j

Output: The graph with edge added b/w v_i and v_j

Data structure: Linked structure of Directed Graph & DGptr is the pointer to it.

Steps:

1. Let N = no. of vertices in the graph
2. If $(v_i > N)$ or $(v_j > N)$ then
3. Print "Edge is not possible b/w v_i and v_j"
4. Else
5. InsertEnd_SL (DGptr[v_i], v_j) // Add v_j in the adjacency list of v_i
6. EndIf
7. Stop

Malay

Deletion in [illegible] Graph.

① **Algorithm DeleteVertex_LL_UG**

Input: V_x, the label of vertex to be removed from the graph
let N be the no. of vertices presently available in the graph.

Output: The reduced graph w/o the vertex V_x & its associated edges.

Data Structure: Linked structure of Undirected graph & UGptr is the pointer to it.

Steps :→

Malay

```
1.  If (N=0) then
2.      Print "Graph is empty: No deletion"
3.      Exit
4.  EndIf
5.  ptr = UGptr[Vx] → LINK        // Move to the first node in the list of Vx
6.  While (ptr ≠ NULL) do         // for all nodes in the adjacency list
7.      j = ptr → LABEL           // Get the label of the vertex
8.      DeleteAny_SL (UGptr[j], Vx)  // Delete the node having label Vx
                                     from adjacency list.
9.      DeleteAny_SL (UGptr[Vx], j)  // Delete the node having label
                                     "j" from adjacency list of Vx.
10.     ptr = UGptr[Vx] → LINK
11. EndWhile
12. UGptr[Vx] → LABEL = NULL   ⟶  // Make entry NULL
13. UGptr[Vx] → LINK = NULL
14. ReturnNode(ptr)
15.     N = N-1
```

Algorithm - Delete Edge - LL - UG

Input : UGptr, the pointer to the graph. $\langle V_i, V_j \rangle$, the edge to be deleted b/w vertices V_i and V_j.

Output : The graph w/o edge b/w V_i and V_j.

Malay

Steps :→

1. Let N = no. of vertices in the graph.
2. If (V_i > N) or (V_j > N) then
3. Print " Vertex does not exists : Error in edge removal"
4. Else
5. Delete Any-SL (UGptr [V_i], V_j) // Delete V_j from the adjacency list of V_i
6. Delete Any-SL (UGptr [V_j], V_i) // Delete V_i from the adjacency list of V_j
7. EndIf
8. Stop.

Algorithm DeleteVertex_LL_DG

Input: DGptr, the pointer to the graph. V_x, the label of the vertex which has to be removed from the graph.

Let N be the no. of vertices presently available in the graph.

Output: The reduced graph w/o the vertex V_x & its associated edges.

Data structure: Linked list

Malay

Steps

1. If (N=0) then
2. Print "Graph is empty: No deletion"
3. Exit → // Exit the program
4. EndIf
5. ptr = DGptr[V_x] → LINK // Pointer to the adjacency list of V_x.
6. DGptr[V_x] → LINK = NULL // Remove adjacency list of vertex V_x
7. DGptr[V_x] → LABEL = NULL // Remove V_x from the list of vertices
8. N = N-1.
9. ReturnNode(ptr)

   /* for V_x in the adjacency list, deleting all existing vertices */
10. for i=1 to N do
11. DeleteAny_SL(DGptr[i], V_x)
12. EndFor
13. Stop.