1. Explain Dijkstra's algorithm with examples.

- Dijkstra's algorithm is the most efficient algorithm for finding the shortest path between a specified vertex pair.

Description of the algorithm: Dijkstra's algorithm labels the vertices of the given digraph. At each stage in the algorithm, some vertices have permanent labels and others temporary labels. This algorithm begins by assigning a permanent label 0 to the starting vertex s, and a temporary label ∞ to the remaining n-1 vertices. From then on, in each iteration another vertex gets a permanent label, according to the foll. rules:

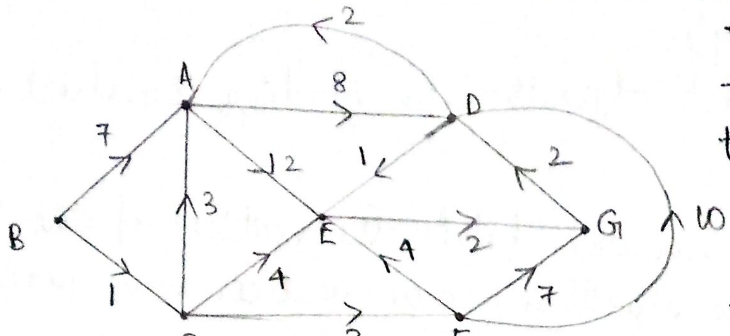
1. Every vertex j that is not yet permanently labeled gets a new temporary label whose value is given by min[old label of j, (old label of i + d_{ij})] — ① where i is the latest vertex permanently labeled, in previous iteration, and d_{ij} is the direct distance between vertices i and j. If i and j are not joined by an edge, then $d_{ij} = \infty$.

2. The smallest value among all the temporary labels is found, and this becomes the permanent label of the corresponding vertex. In case of a tie, select any one of the candidates for permanent labeling.

Steps 1 & 2 are repeated alternately until the destination vertex t gets a permanent label.

The 1st vertex to be permanently labeled is ~~the~~ at a distance of 0 from s. The 2nd vertex to get a permanent label is the vertex closest to s, and so on. The permanent label of each vertex is the shortest distance of that vertex from s. We shall use a vector of length 7 to show the temporary & permanent labels of the vertices. The permanent labels will be shown enclosed in a square, and the most recently assigned permanent label in the vector is indicated by a tick □✓.

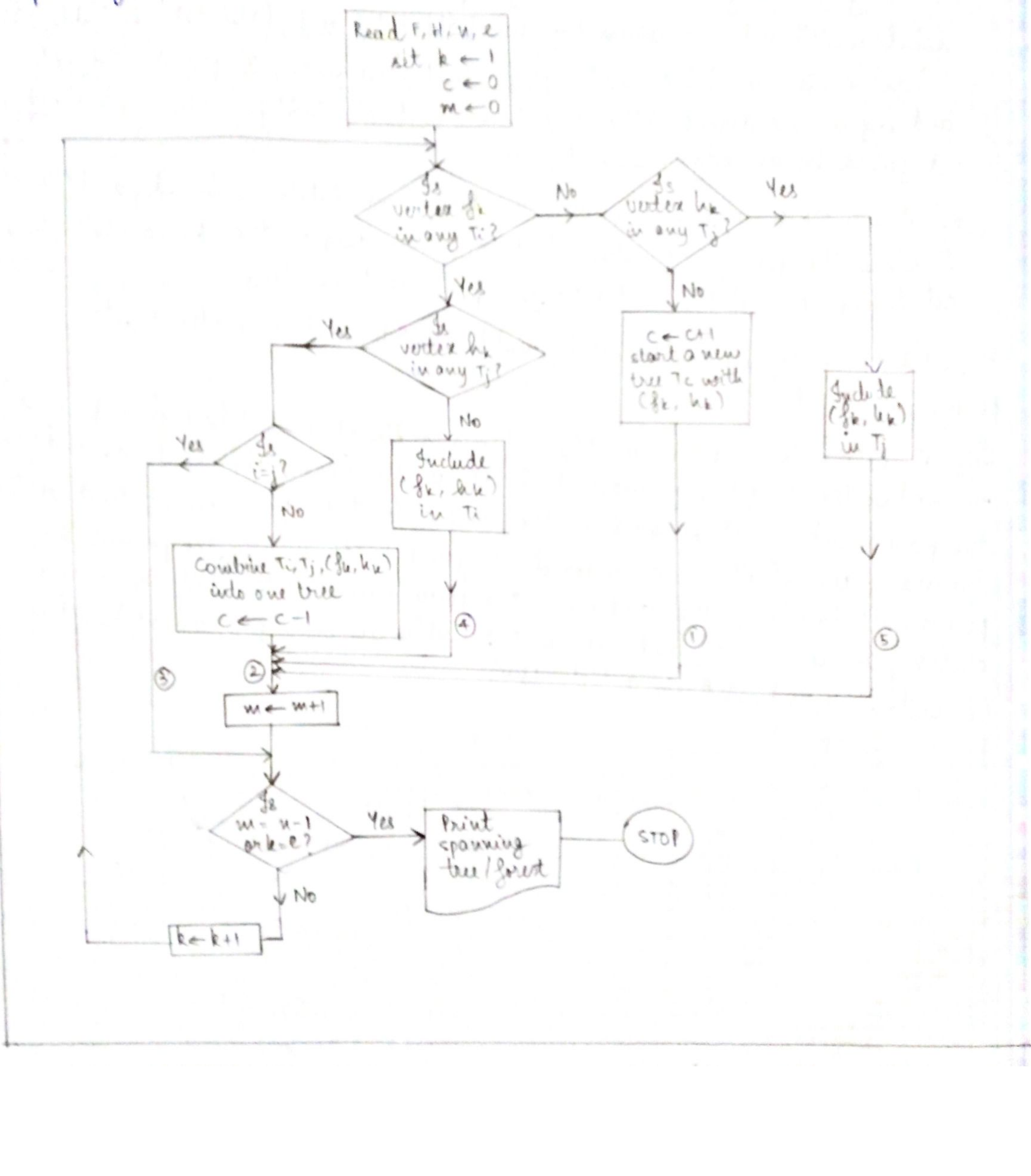
A	B	C	D	E	F	G	
∞	$\boxed{0}$✓	∞	∞	∞	∞	∞	: starting vertex B is labeled 0.
7	$\boxed{0}$	1	∞	∞	∞	∞	: All successors of B get labeled
7	$\boxed{0}$	$\boxed{1}$✓	∞	∞	∞	∞	: Smallest label becomes permanent
4	$\boxed{0}$	$\boxed{1}$	∞	5	4	∞	: Successors of C get labelled.
4	$\boxed{0}$	$\boxed{1}$	∞	5	$\boxed{4}$✓	∞	
4	$\boxed{0}$	$\boxed{1}$	14	5	$\boxed{4}$	11	
$\boxed{4}$	$\boxed{0}$	$\boxed{1}$	14	5	$\boxed{4}$	11	
$\boxed{4}$	$\boxed{0}$	$\boxed{1}$	12	5	$\boxed{4}$	11	
$\boxed{4}$	$\boxed{0}$	$\boxed{1}$	12	$\boxed{5}$✓	$\boxed{4}$	11	
$\boxed{4}$	$\boxed{0}$	$\boxed{1}$	12	$\boxed{5}$	$\boxed{4}$	7	
$\boxed{4}$	$\boxed{0}$	$\boxed{1}$	12	$\boxed{5}$	$\boxed{4}$	$\boxed{7}$✓	: Destination vertex gets labeled permanently



simple weighted digraph

The algorithm described does not actually list the shortest path from the starting vertex to the terminal vertex; it only gives the shortest distance. The shortest path can be easily constructed by working backward from the terminal vertex such that we go to that predecessor whose label differs exactly by the length of the connecting edge.

2. Draw the flow chart of spanning tree algorithm and write the algorithm that clearly mentions the five conditions to be tested in connection with the spanning tree construction.



Let the given undirected self-loop-free graph G contain n vertices and e edges. Let the vertices be labeled $1, 2, \ldots, n$ and the graph be described by 2 linear arrays F and H such that $f_i \in F$ and $h_i \in H$ are the end vertices of the ith edge in G.

At each stage in the algorithm, a new edge is tested to see if either or both of its end vertices appear in any tree formed so far. At the kth stage, $1 \le k \le e$, in examining the edge (f_k, h_k), 5 different conditions may arise:

1. If neither vertex f_k nor h_k is included in any of the trees constructed so far in G, the kth edge is named as a new tree and its end vertices f_k, h_k are given the component number c, after incrementing the value of c by 1.
2. If vertex f_k is in some tree T_i $(i = 1, 2, \ldots, c)$ and h_k is in tree T_j $(j = 1, 2, \ldots, c$ and $i \neq j)$, the kth edge is used to join these 2 trees; therefore, every vertex in T_j is now given the component number of T_i. The value of c is decremented by 1.
3. If both vertices are in the same tree, the edge (f_k, h_k) forms a fundamental circuit and is not considered any further.
4. If vertex f_k is in a tree T_i and h_k is in no tree, the edge (f_k, h_k) is added to T_i by assigning the component number of T_i to h_k also.
5. If vertex f_k is in no tree and h_k is in a tree T_j, the edge (f_k, h_k) is added to T_j by assigning the component number of T_j to f_k also.