Explain Dijkstra's algorithm with examples

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

<u>Description of the Algorithm</u>: Dijkstra's algorithm labels the vertices of the given digraph. At each stage in the algorithm some vertices have permanent labels and others temperory labels. The algorithm begins by assigning.

If the given digraph is not simple, it can be simplified by discarding all self-loops and replacing every set of parallel edges by the shortest (least-weight edge among them. Also, the graph need not be directed. For an undirected graph $d_{ij} = d_{ji}$ and effectively each undirected edge is replaced by 2 oppositely directed edges of the same weight. If the graph is not weighted, assume $d_{ij} = 1$ and the adjacency matrix becomes the distance matrix, a permanent label 0 to the starting vertex s, and a tempory label ∞ to the remaining $n-1$ vertices. From then on, in each iteration another vertex get a permanent label, according to the following rules:

1. Every vertex j that is not yet permanently labeled gets a new temporary label whose value is given by

$$\min [\text{old label of } j, (\text{old label of } i + d_{ij})],$$

where i is the ~~shortest~~ latestest vertex permanently labeled, in the previous iterations and d_{ij} is the direct distance between vertices i and j. If i and j are not joined by an edge, then $d_{ij} = \infty$.

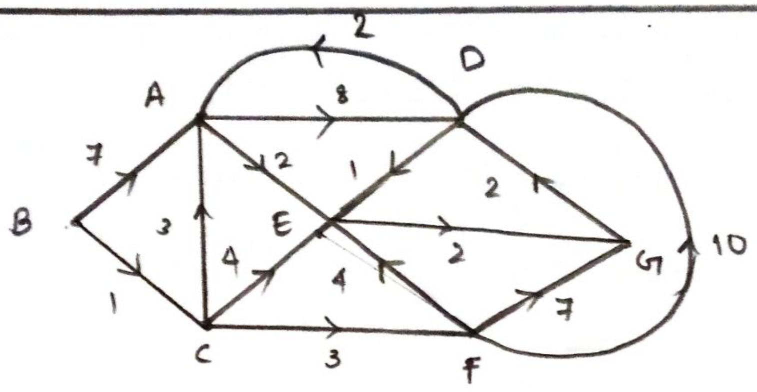
2. The smallest value among all the temporary labels is found, and this becomes the permanent label of the corresponding vertex. In case of a tie, select any one of the candidates for permanent labeling.

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

The first vertex to be permanently labeled is at a distance of zero from s. The second vertex to get a permanent label (out of the remaining n-1 vertices) is the vertex closest to s. From the remaining n-2 vertices, the next one to be permanently labeled is the second closest vertex to s. And so on. The permanent label of each vertex is the shortest distance from vertex B to G in the digraph shown in figure.

We shall use a vector of length seven to show the temporary and permanent labels of the vertices as we go through the solution. 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 $\boxed{\ }$✓. The labeling proceeds as follows:

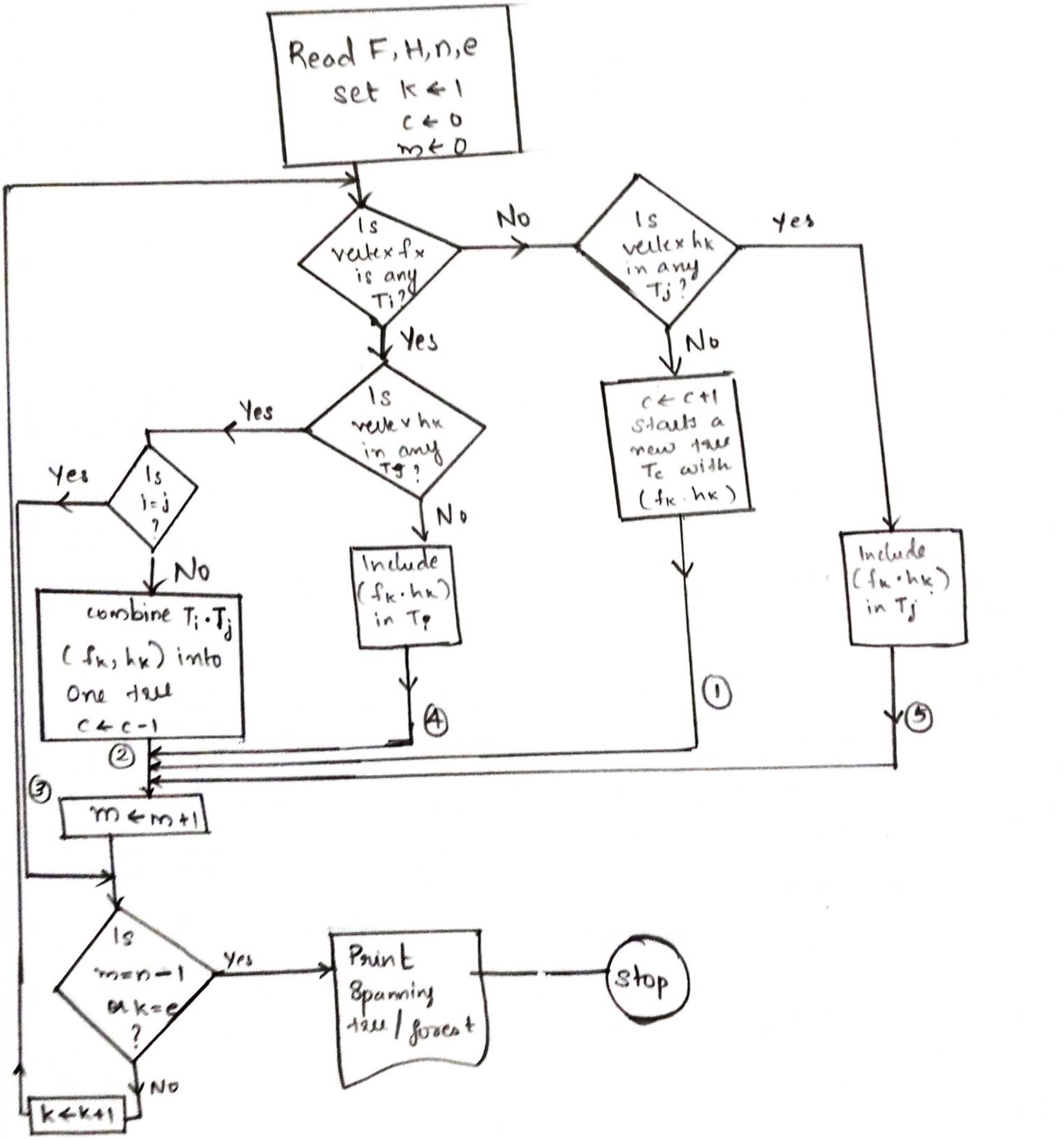
A	B	C	D	E	F	G		
∞	$\boxed{0}$	∞	∞	∞	∞	∞	:	Starting vertex B is labeled 0
7	$\boxed{0}$	1 ✓	∞	∞	∞	∞	:	All successors of B get labeled
7	$\boxed{0}$	$\boxed{1}$	∞	∞	∞	∞	:	Smallest label becomes permanent
4	$\boxed{0}$	$\boxed{1}$	∞	5	4	∞	:	Successors of C get labeled
4	$\boxed{0}$	$\boxed{1}$	∞	5	$\boxed{4}$ ✓	∞		
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 permanently labeled!



Simple weighted digraph.

The algorithms 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.

Draw the flowchart of spanning tree algorithm and also write the algorithm that clearly mention the five conditions to be tested in connection with the spanning tree construction.



Algorithm:

Let the given undirected self-loop-free (if the graph has any self loops, they may be discarded) graph

G contains n vertices and e edges. Let the vertices be labeled 1, 2, ... n and the graph be described by two linear arrays F and H [i.e., in the form (d) of section] such that $f_i \in F$ and $h_i \in H$ are the end vertices of the i^{th} 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 k^{th} stage, $1 \le k \le e$ in examining the edge (f_k, h_k) five 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 k^{th} edge is named as a new tree and its vertices f_k, h_k are ~~done~~ 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 in tree T_j $(j = 1, 2, \ldots c$, and $i \neq j)$, the k^{th} 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 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.