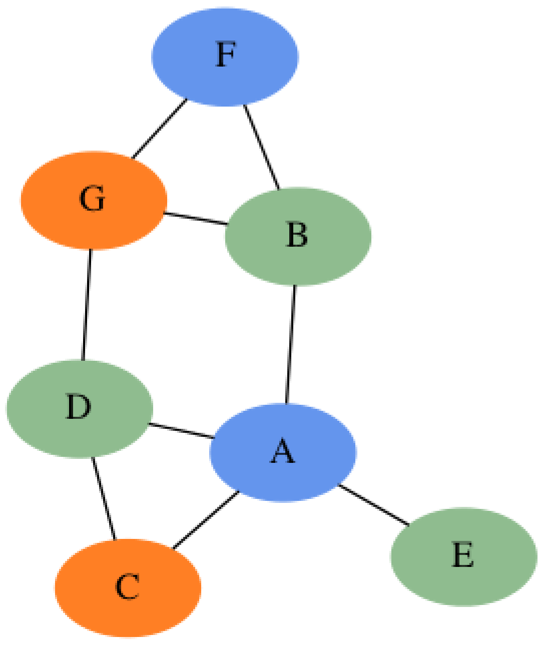
7/1)No the graph will not be directed.



7/2)

Result of inserting the input numbers using linear probing :-

0	22
1	88
2	
3	
4	4
5	15
6	28
7	17
8	59
9	31
10	10

Result of inserting using quadratic probing:-

0	22
1	
2	88
3	17
4	4
5	59
6	28
7	
8	15
9	31
10	10

Result of inserting using double hashing:-

0	22
1	
2	59
3	17
4	4
5	15
6	28
7	
8	88
9	31
10	10

7/3) h(k , i)=(h1(k)+ih2(k))%m $\longrightarrow$ (i)

Let us suppose that h2(k) and m has greatest common divisor d>=1. Then dividing m by d will give us non-common factor of m between m and d. And hence searching the table for (m/d)th number of times means h2(k) will add up (m/d)th times giving the lcm(least common multiple) of h2(k) and m. After that value of h2(k) will start from initial as if value of I is started from 1.

So, in mean time we have checked only (m/d) number of blocks of hash table and we have not checked any of the rest of the blocks and they will not be checked until or unless value of d is 1. Because in that case we will check all the blocks of hash table.

Example: m = 48, h2(k) = 24 ,GCD(48,24) = 24 , only 2 of cells of the table would be addressed.

There are two guidelines on choosing m and step function so that GCD(h2(k),m)=1:

1. The size of the table m is a prime number, and h2(k) < m for every k.

2. The size of the table m is 2x, and the step function returns only odd values.

7/4) Suppose in rehashing, each time the load factor reaches 0.5, we double the size of hash table and hence the load factor becomes 0.25. In the same way as soon as load factor reaches 0.5, we repeat the previous steps. And hence the load factor oscillates between 0.25 and 0.5. So the average value is 0.375.

$$\alpha=n/m$$

n->number of blocks occupied

m-> total number of blocks

If we double the size of table when α reaches 0.5, then α becomes

$$\alpha=n/(2*m)$$

n is constant, therefore α becomes 0.25

7/6)

```
Dijkstra(G, w, s)
1. INITIALIZE-SINGLE-SOURCE(G, s)
2. S .. {}
3. Q .. V[G]
4. while Q != {}
    a. do u .. EXTRACT-MIN(Q)
    b. S .. S.{u}
    c. For each vertex v.Adj[u]
        i. Do RELAX(u, v, w)
```

The running time of Dijkstra's algorithm depends on the implementation of the min-priority queue. In Dijkstra's algorithm, we process those vertices closest to the source vertex first. Because each edge has at most weight W, we know that the maximum possible value of the longest path in the graph is (V-1)W. We can prioritize the vertices based on their d[] values. Remember, d[v] is the shortest path from the source to vertex v.

The queue consists of (V – 1)W + 2 buckets. Vertex v can be found in bucket d[v]. Since all other than the source have d[v] values between 1 and (V – 1)W, so they can be found in buckets 1, …, (V – 1)W. If s is the source vertex, d[s] = 0. So, s can be found in bucket 0. INITIALIZE-SINGLE-SOURCE ensures that for all vertices v other than the root, d[v] is initialized to 8. The final bucket holds all vertices whose d[]-values are infinity (all undiscovered vertices).

After initializing all of the vertices, we scan the buckets from 0 to (V – 1)W. When a non-empty bucket is encountered, the first vertex is removed, and all adjacent vertices are relaxed. This step is repeated until we have reached the end of the queue—in O(WV) time. Since we relax a total of E edges, the total running time for this algorithm is O(VW + E).

7/7) For G to be acyclic, undirected and connected graph, there should be no more than n-1 edges.

7/9) Suppose G has a Hamilton path. Any topological ordering must respect all the edges on this path, and must therefore put the vertices in exactly the same order as they appear on the path.

On the other hand, if G does not have a Hamilton path, the algorithm will find some topological ordering v1, . . . , vn, but one of the edges (vi, vi+1) will not be in E.