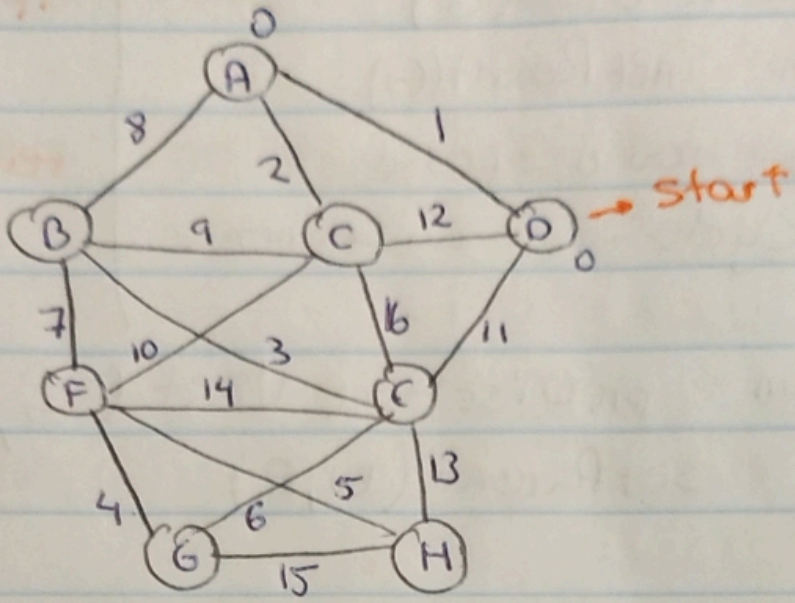
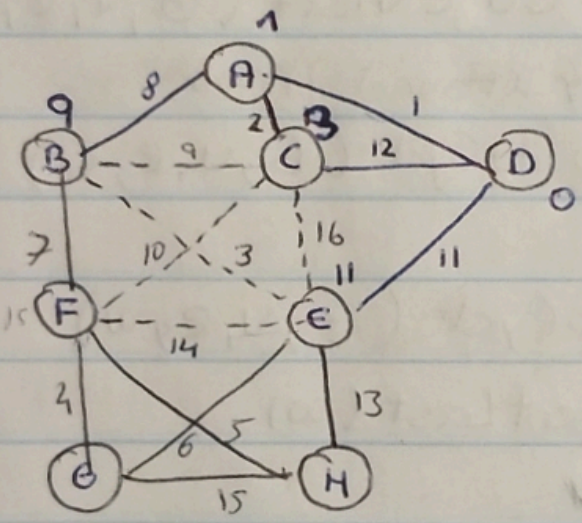
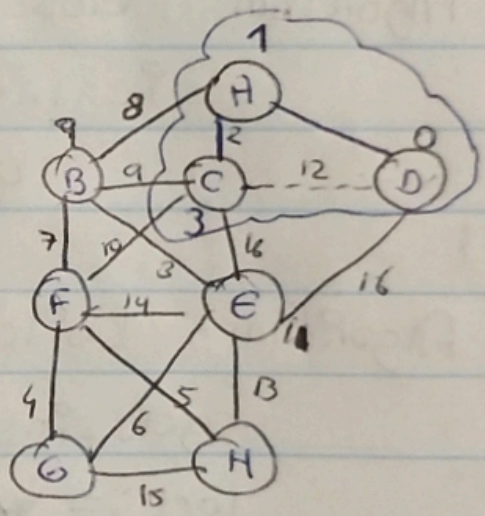
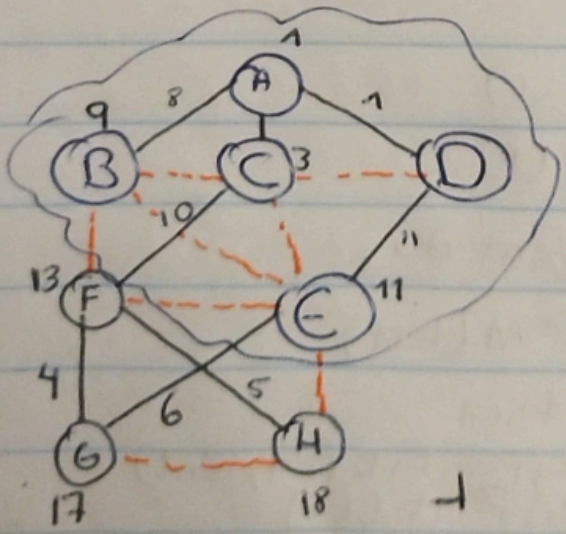
① Draw Shortest Path.

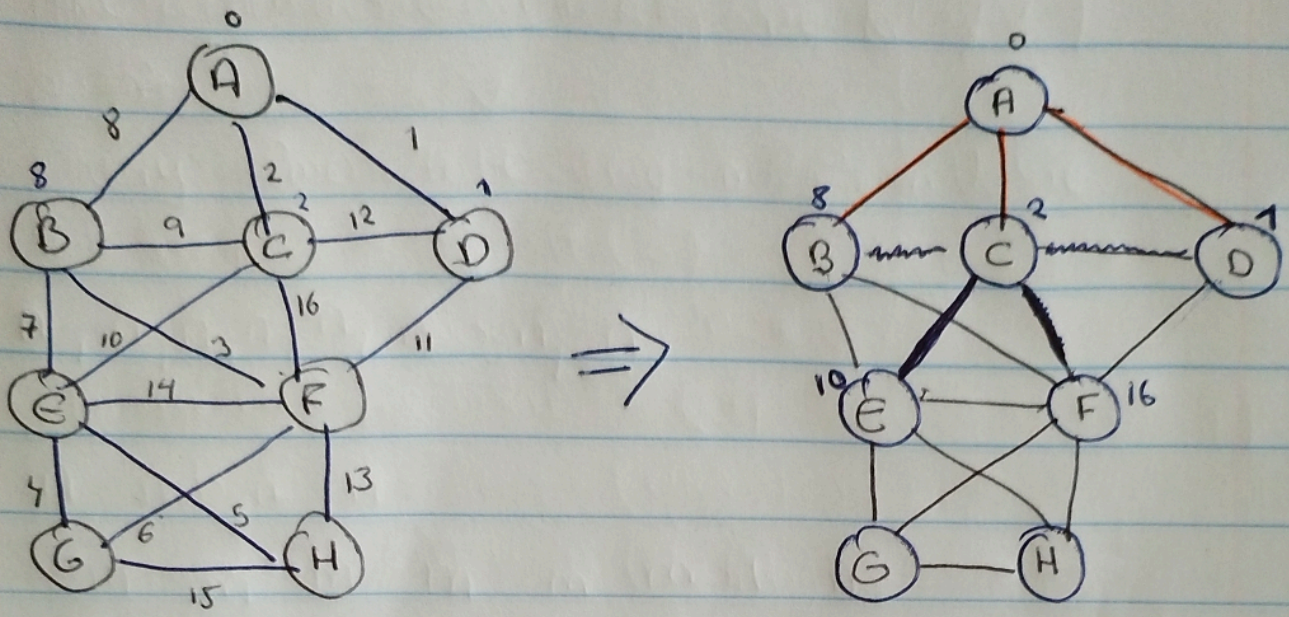


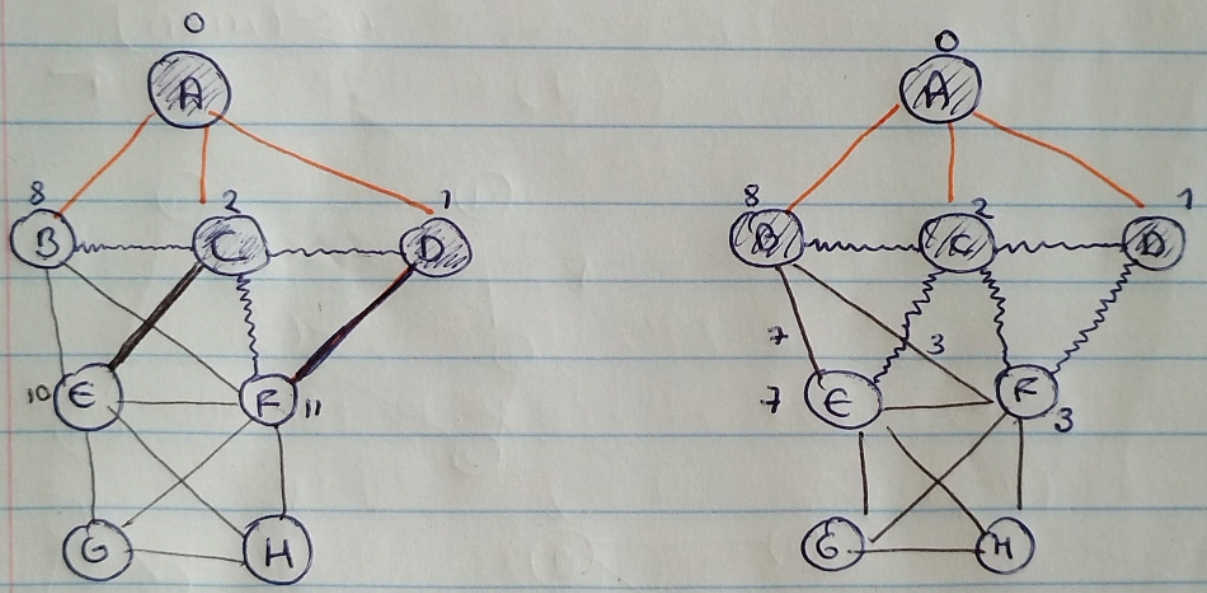


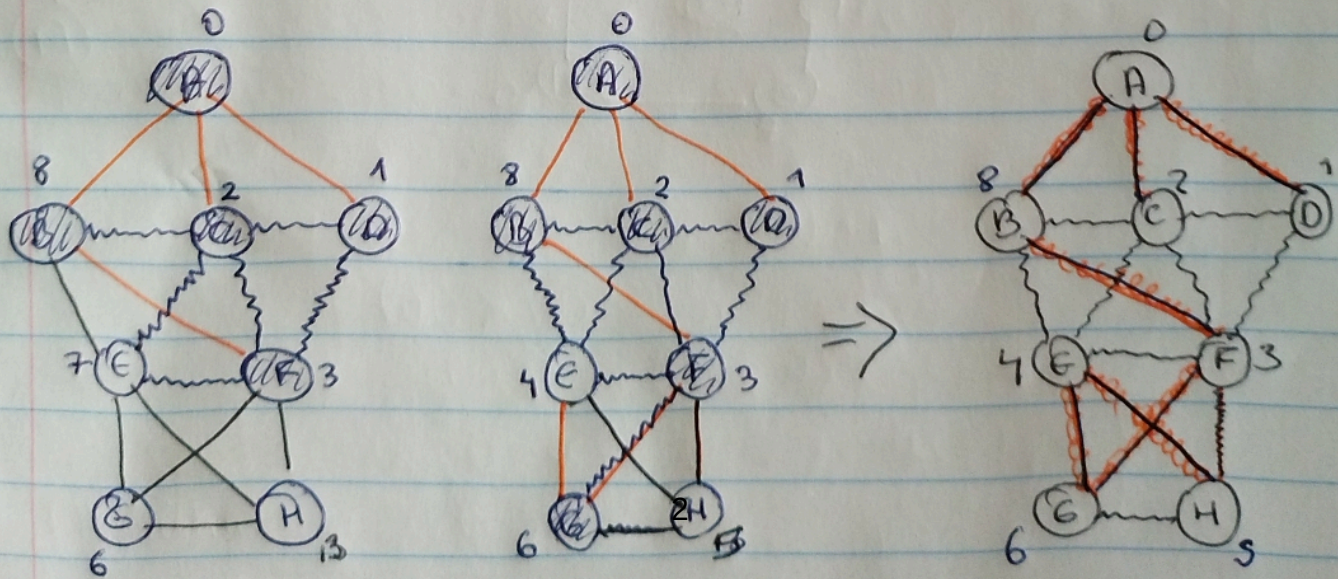




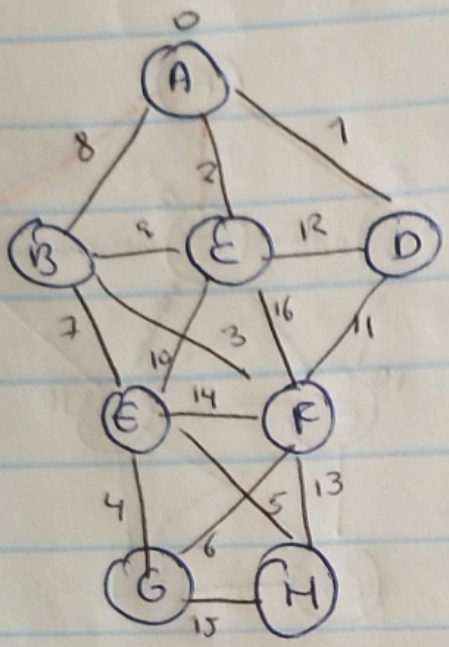
Minimum Spanning tree.

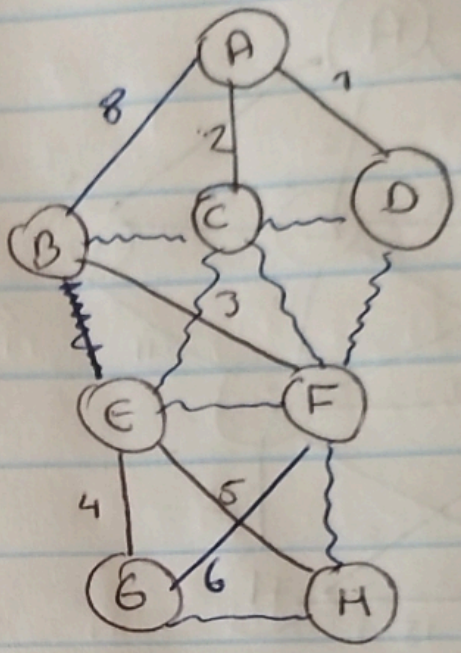


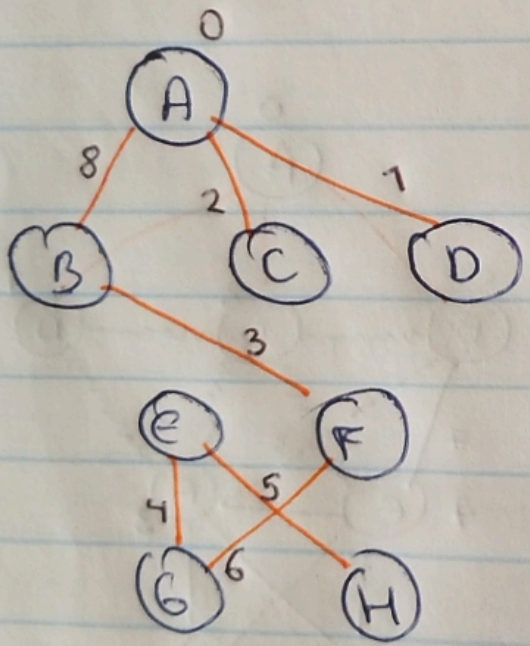




R.7.9 Algorithm Baruvka







~~Calculate~~ A: BFS Dijkstra

Algorithm calculatePaths (G, v)

    start := v
    return BFS(G)

Algorithm initResult (G)

    Q := new PriorityQueue

Algorithm passInitVertex (u)

    if u = start
        setDistance (u, 0)
    else setDistance(u, ∞)
    locator ← Q.insert (getDistance (u), u)
    setLocator (u, L)
    setParent (u, ∅)

Algorithm isNextComponent (G, u)

    return if u = start ~~then~~

Algorithm preDiscEdgeVisit (G, u, e, w)

    setDistance (getDistance(v)+1, w)
    setParent(w, e)
    Q.replaceKey (getLocator (w), getDistance(w))

Algorithm crossEdgeVisit(G, v, e, w)

    wD := getDistance(w)
    vD := getDistance(v) + 1
    if (wD > vD then
        setDistance (vD, w)
        Q.replaceKey (getLocator(w), vD)

Algorithm result (G)

    return Q

8.) SUBTREE

```
Algorithm isSubtree (G,T)
    counter := 0
    for each e in T.elements() do
        setEdgeOFT (e, YES)
    hasCycle ← BFS (G)
    if hasCycle or counter > 1 then
        return false
    return true
Algorithm initResult(G)
    isCycle := false
Algorithm postComponentVisit(G, v)
    counter := counter + 1
Algorithm result(G)
    return isCycle
Algorithm preEdgeVisit(G,v,e)
    if getEdgeOFT(e) ≠ YES
        setLabel(e, SKIP)
Algorithm crossEdgeVisit(G,v,e,w)
    isCycle := true
```