1. **在課程投影片提到了一個被稱為Prim’s Algorithm的Minimum Spanning Tree演算法。請證明其正確性。**

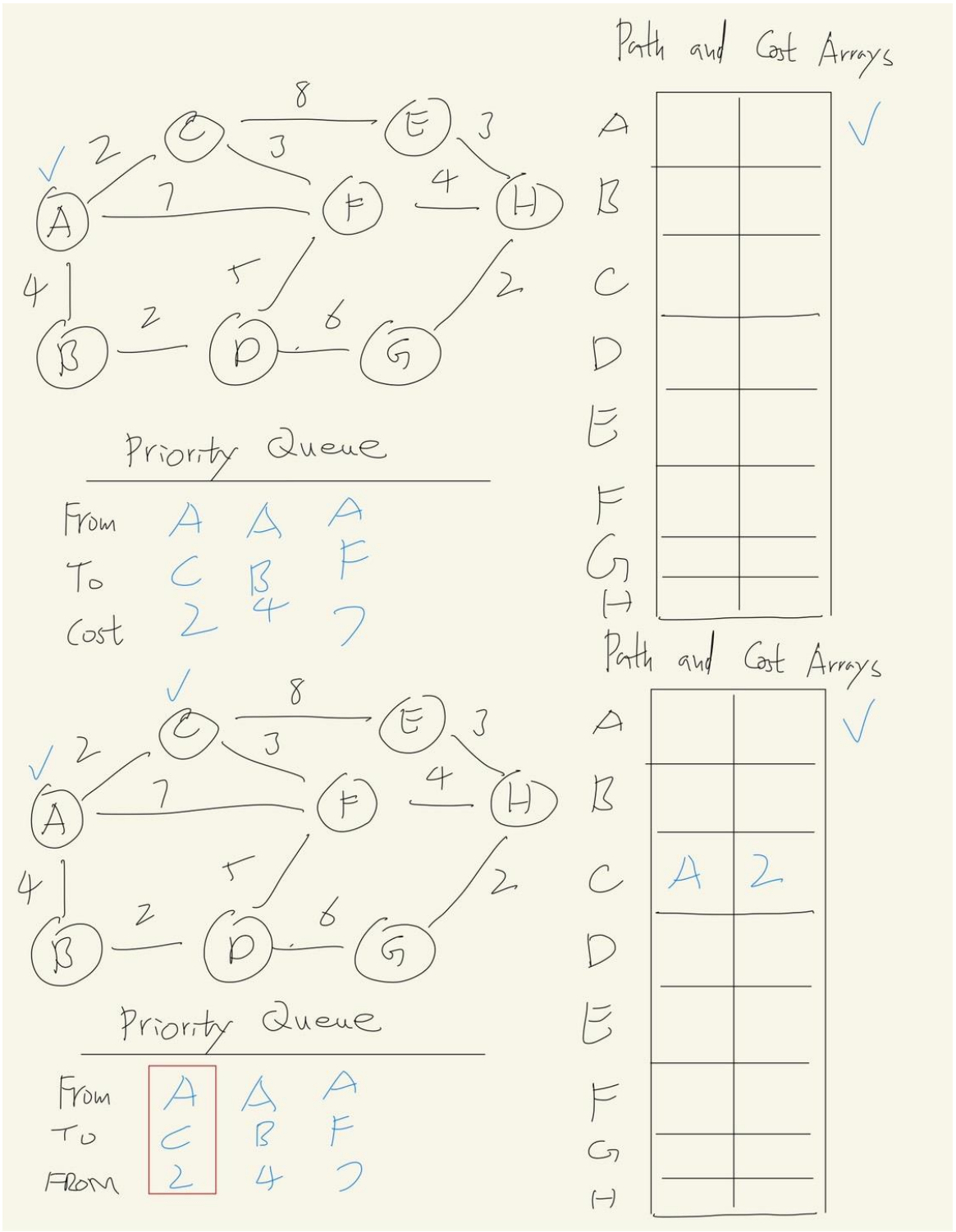
設Tmin為圖像G的MST，而T為Prim生成的Tree
設e1為存在於Tmin但不存在於T的連結(u,v)的邊，設e2為存在於T但不存在於Tmin的連結(u,v)的邊
因為Tmin為MST，故w(e1)<w(e2)
但是Prim演算法透過不斷連接與目前子樹權重最小的邊來吸收外部node v
借助於cut property可得知，對於任一node n，可連接到此node的最小edge必然會屬於MST
所以產生矛盾，w(e1)=w(e2)。

2. **在課程投影片提到了使用如heap的priority queue來加速Dijkstra’s Algorithm的方法，但是當時講課時只提及可以這麼做，卻沒有詳盡解釋怎麼做的細節。請盡量搭配圖例解釋如果使用如heap的priority queue來加速Dijkstra’s Algorithm使之達到O((|V|+|E|)log|V|)的time complexity。附帶一提，若是sparse graph的話；也就是|E|=O(|V|)，上述time complexity大約等於O(|V|log|V|)，比原先的O(|V|2)來得快。可以參考https://tinyurl.com/2p89tbtd的Chapter 4.4或是任何其他你偏好的資源來撰寫這題。**

以下為使用heap的priority queue的Dijkstra演算法:

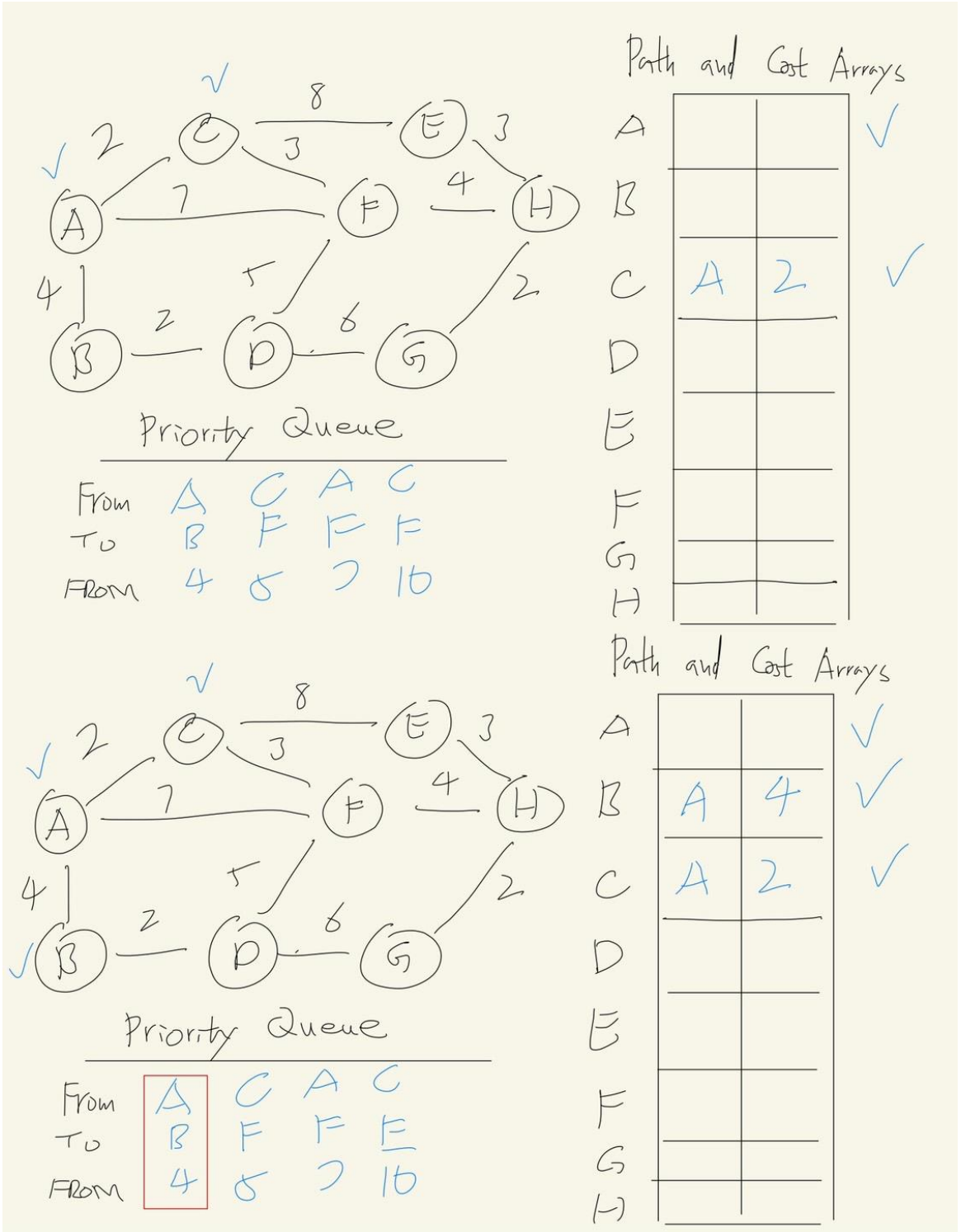
1. 首先由A點開始，挑選cost最低的C

   Visited={A,C}



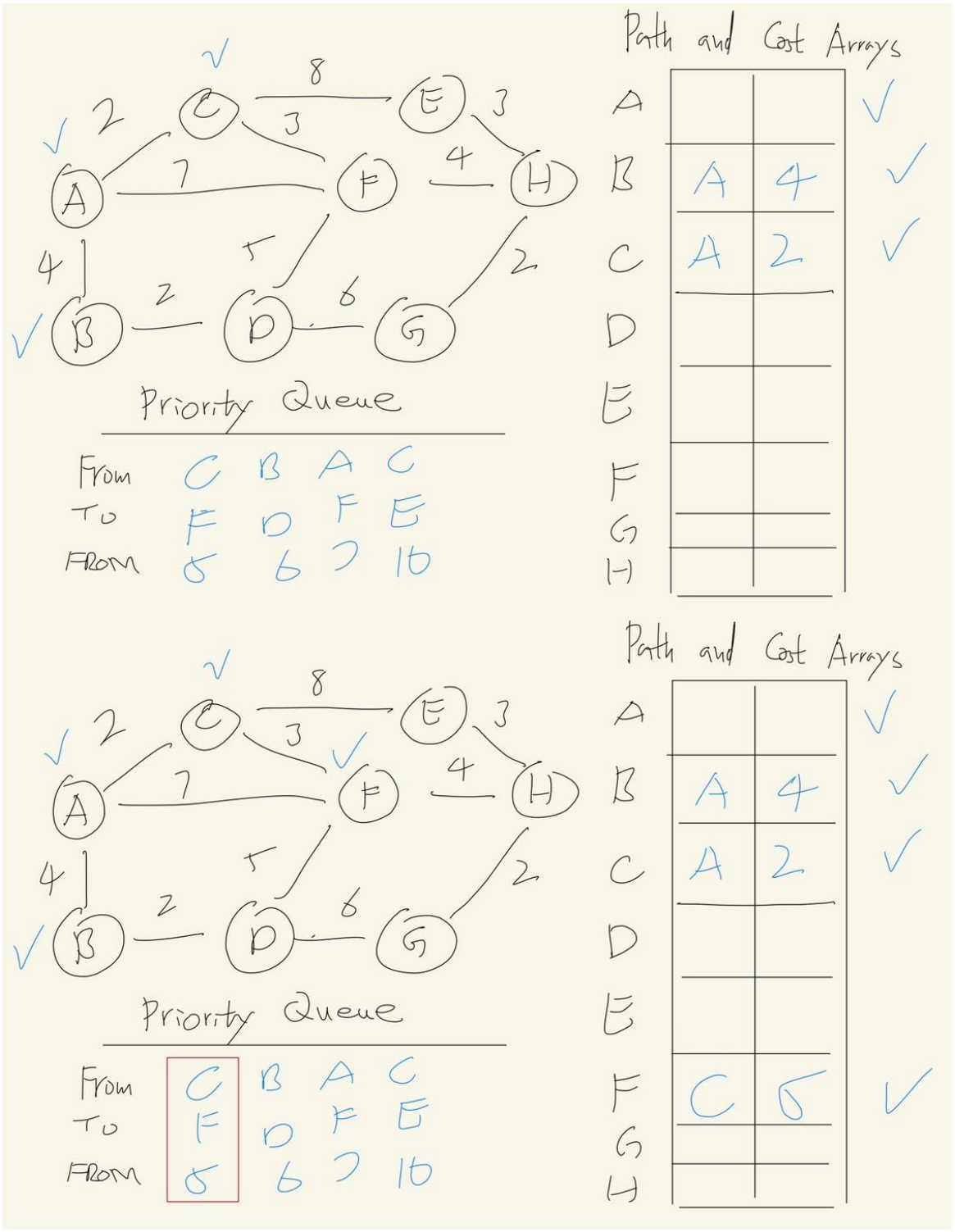
2. 接著挑選cost最低的B

   Visited={A,C,B}



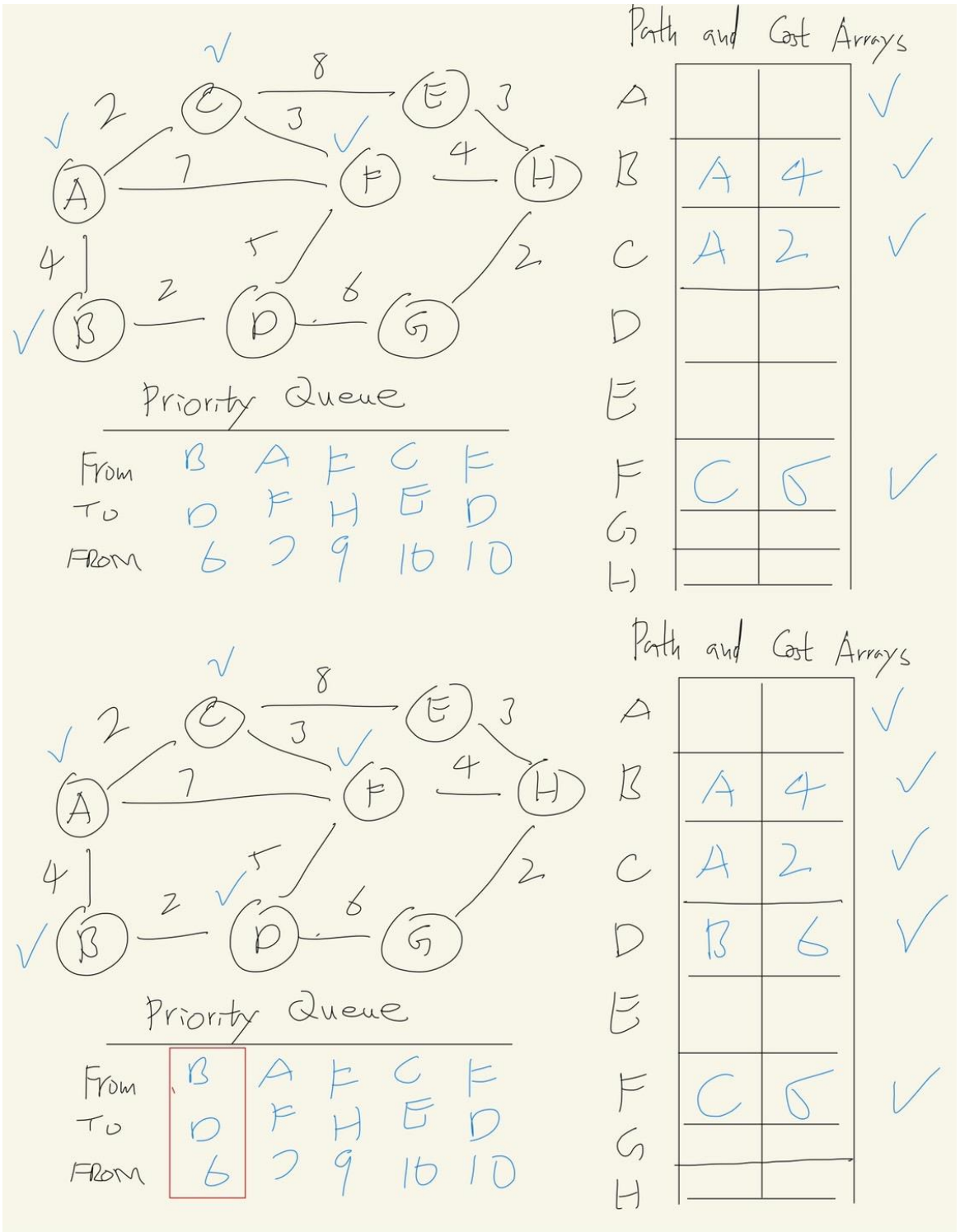
3. 接著挑選cost最低的F

Visited={A,C,B,F}

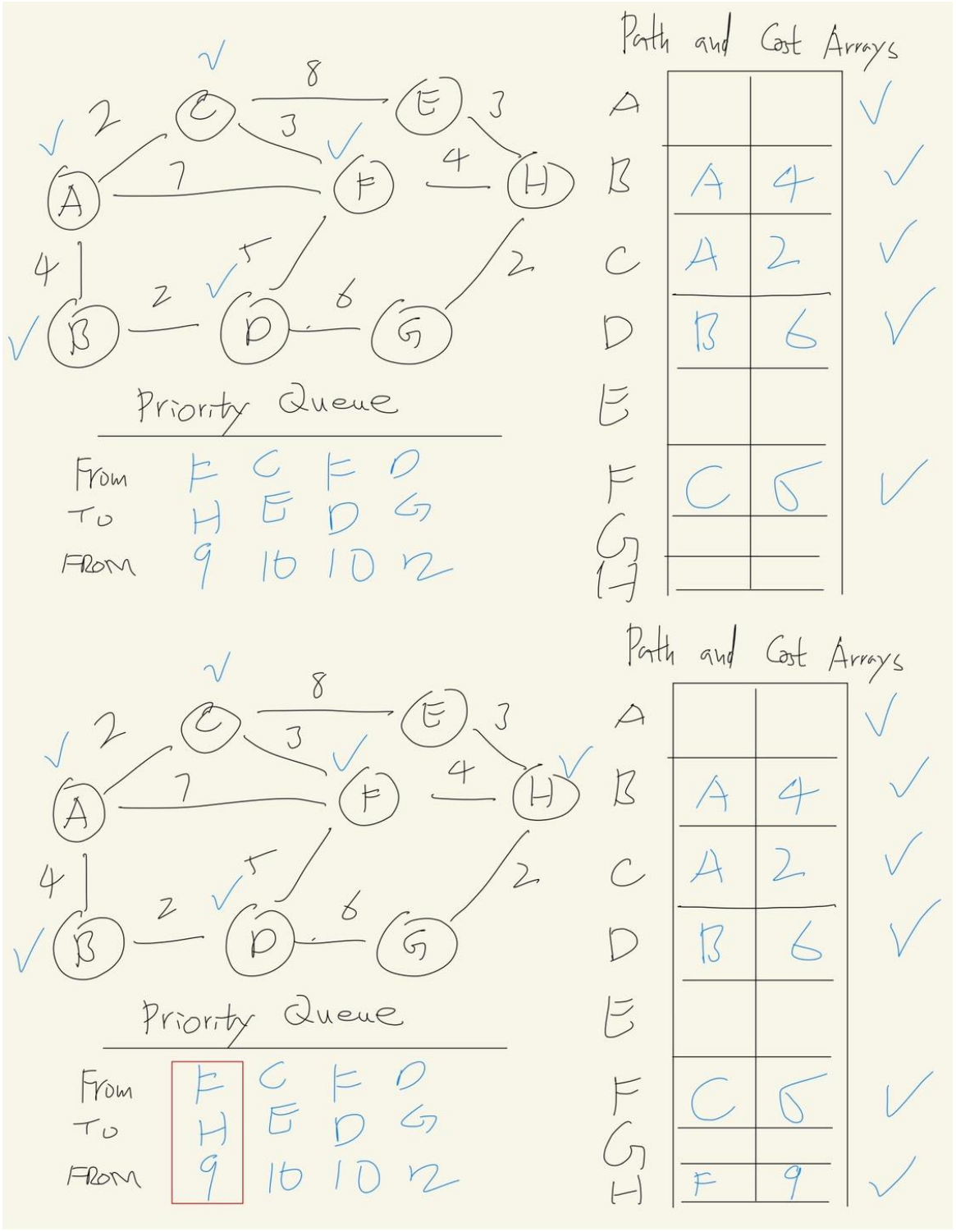


4. 接著挑選cost最低的D

   Visited={A,C,B,F,D}

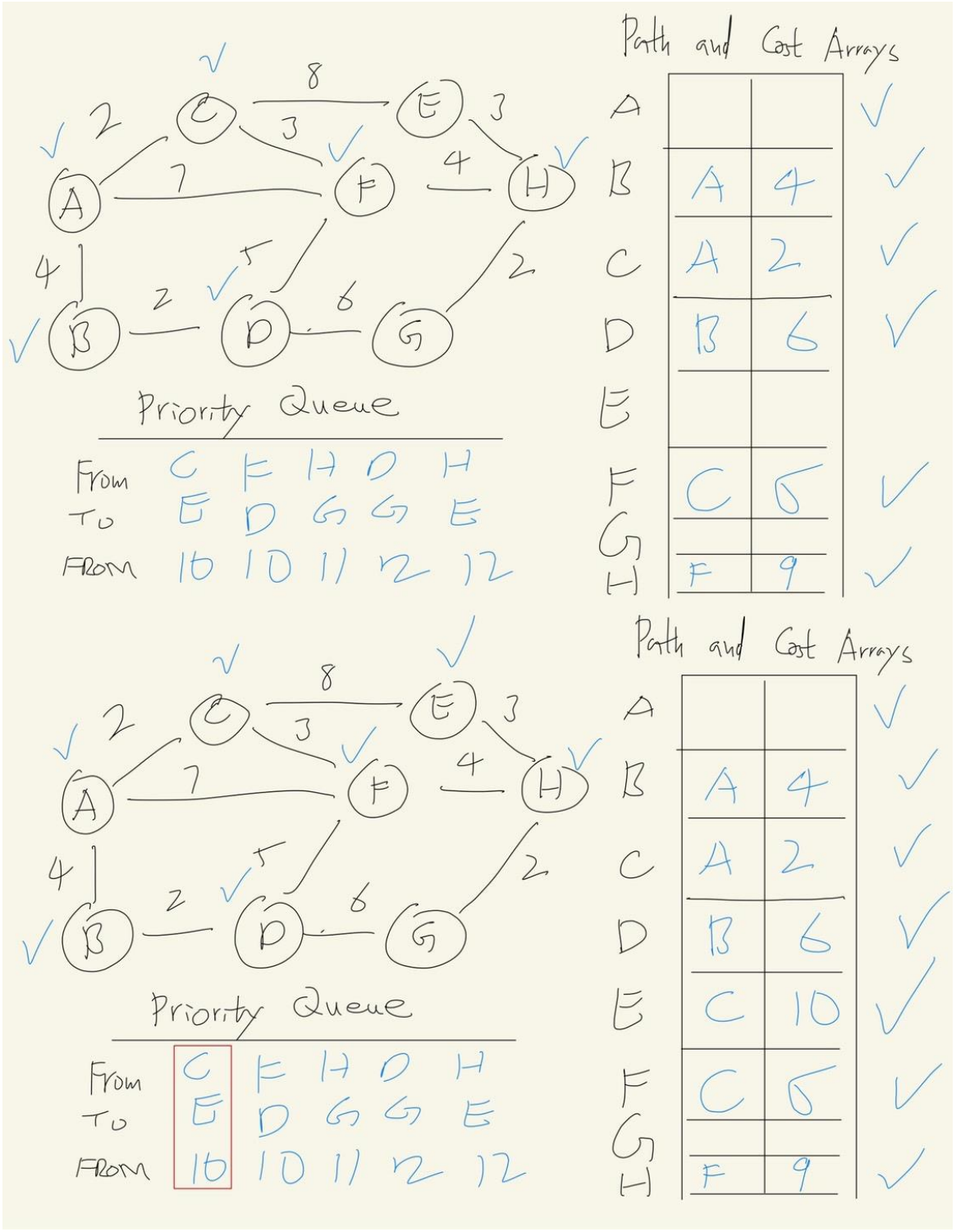


5. 雖然距離F的cost最低，但此點已走過，接著挑選cost最低的H
   Visited={A,C,B,F,D,H}

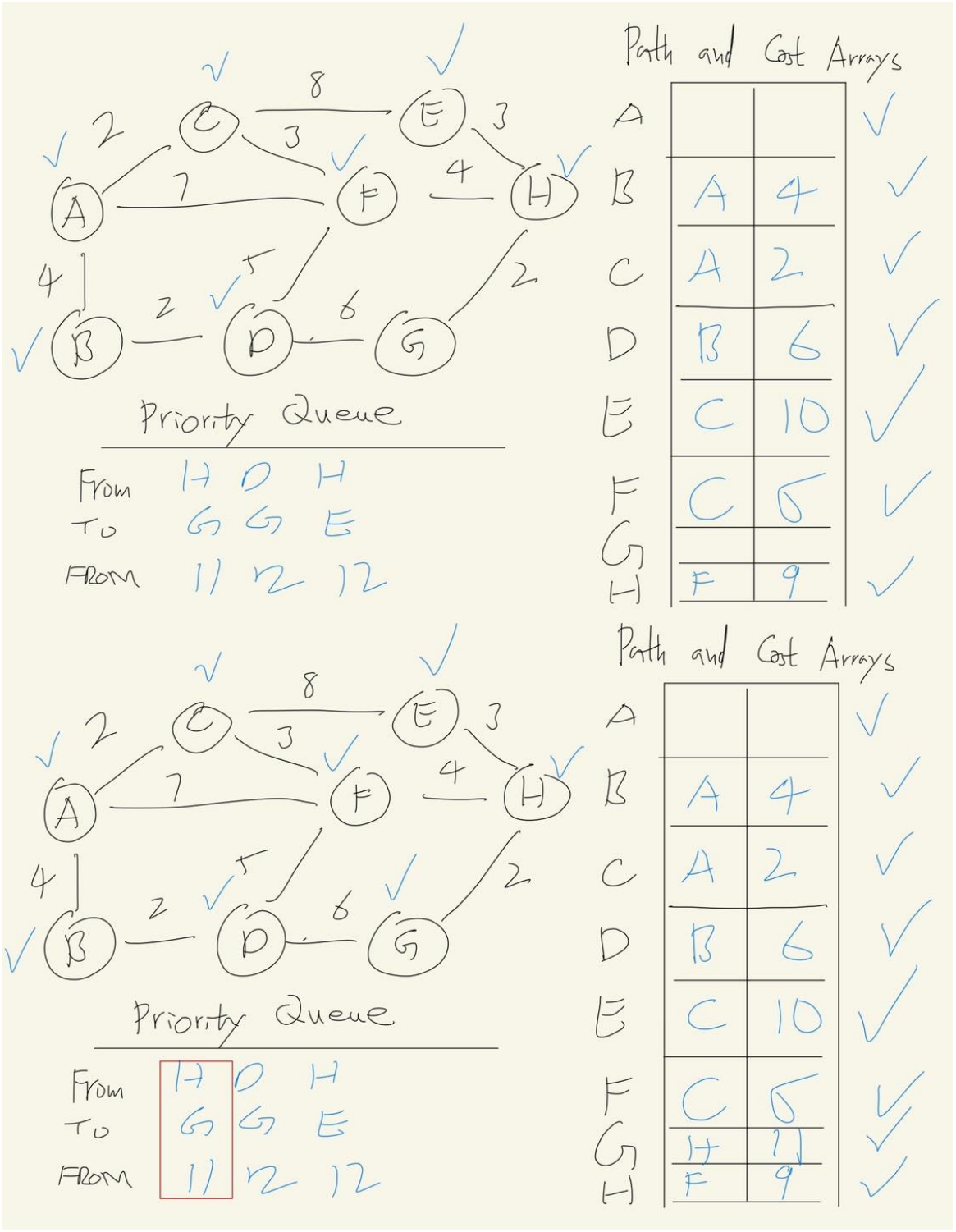


6. 接著挑選cost最低的E

   Visited={A,C,B,F,D,H,E}



7. 雖然距離D的cost最低，但此點已走過，接著挑選cost最低的G
   Visited={A,C,B,F,D,H,E,G}，於是走遍所有點完成了MST



如以上所示，透過 heap 的 priority queue 來執行 Dijkstra 演算法，探尋每個點與邊花費 O(|V|+|E|)，每次需調整 heap 構成的 priority queue 來尋找最小 cost 的點花費 O(log|V|)，總共需要 O((|V|+|E|)log|V|)