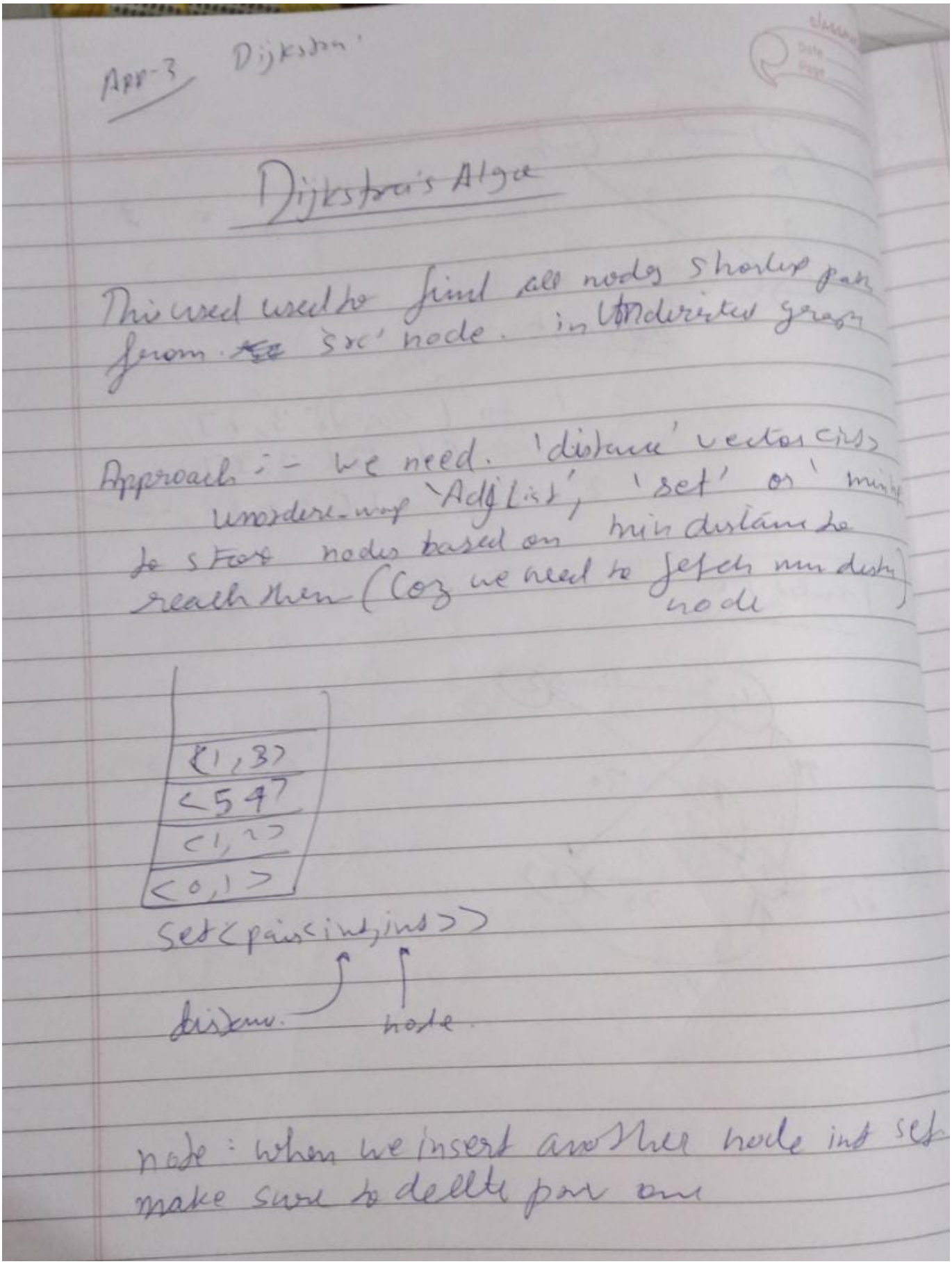
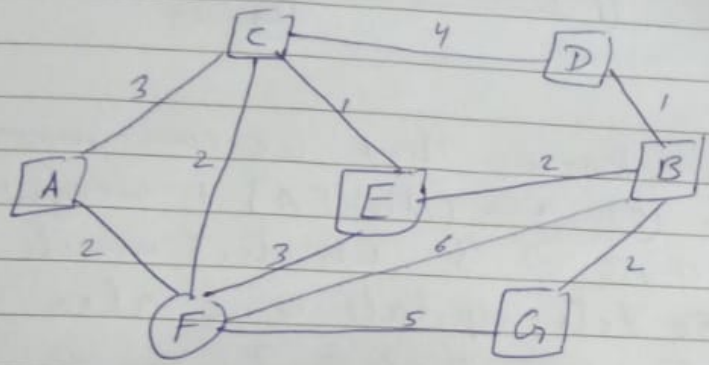
Dijkstra's Algo



Dijkstra's Algo

- works on 2. basic steps
  - (i) relaxation.
  - (ii). choose new next.

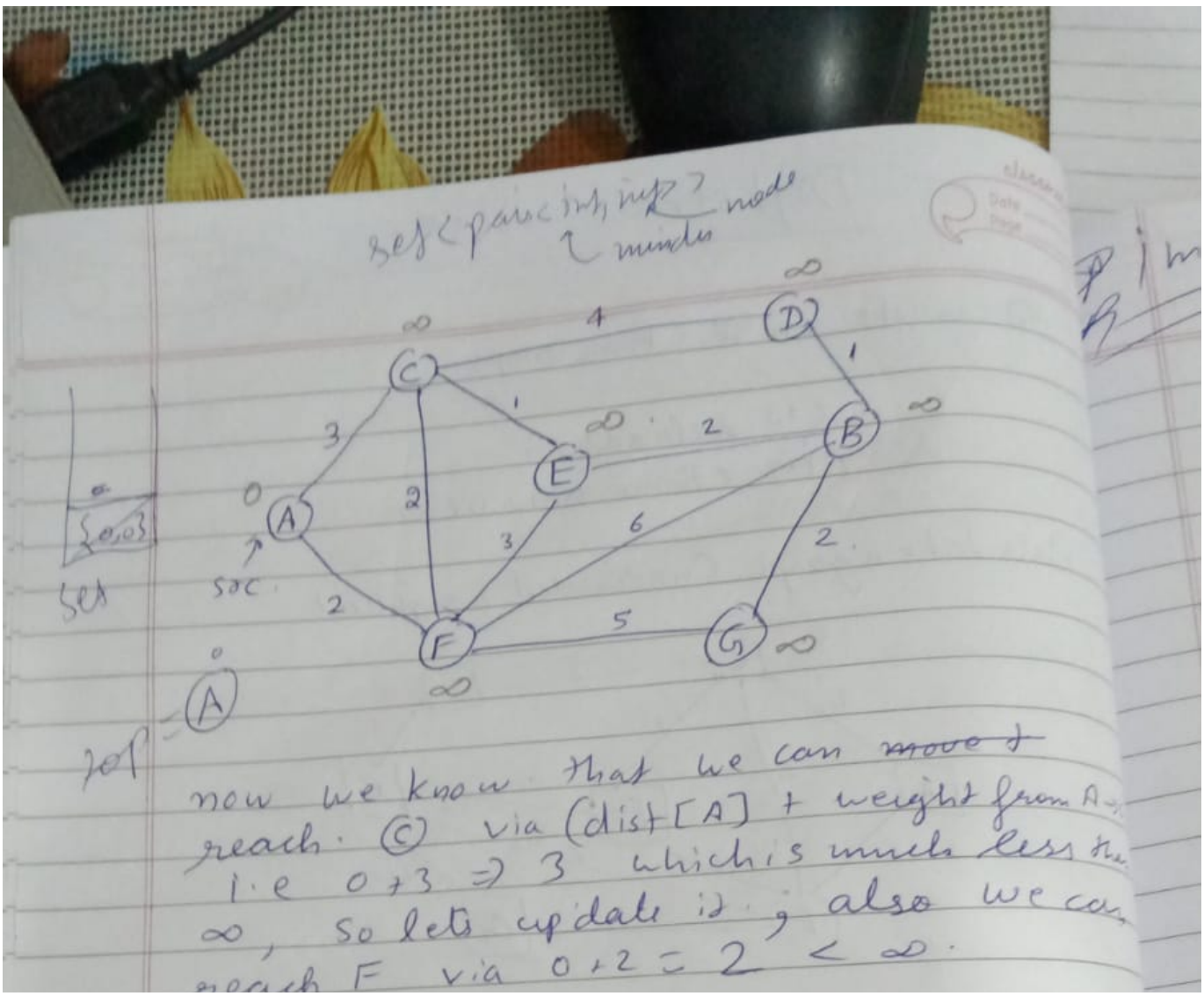
Lets take a graph. (undirected weighted).



Lets say we need to determine the shortest distance as well as shortest path from A to B., but for that we need to know the shortest distances to reach other nodes from A. & if we have it we can easily determine the final ans

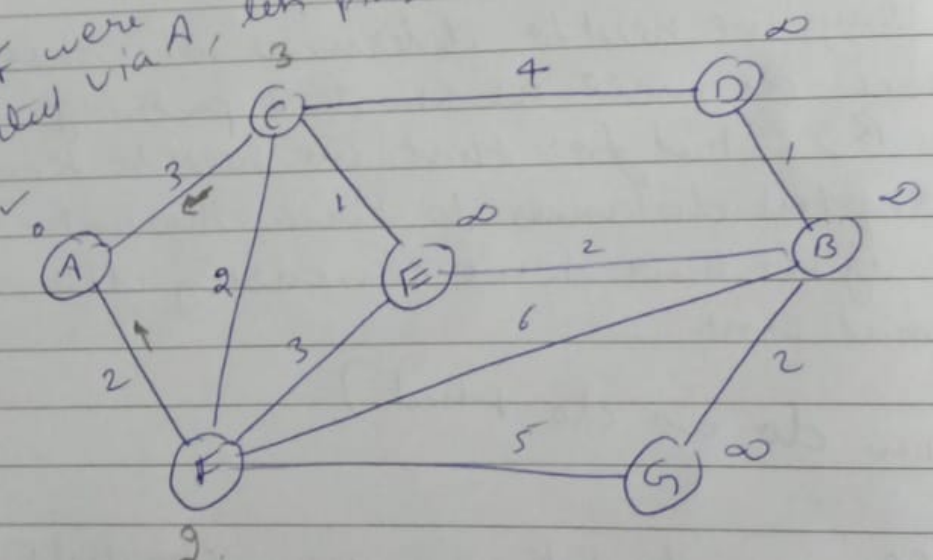
But how do we do that?

Initially we don't know any [illegible]
for any node except start, so lets
take all the other nodes distance as
'∞' & src's dist a '0'



now we know that we can ~~move~~ reach C via (dist[A] + weight from A→C)
i.e 0+3 ⇒ 3 which is much less than
∞, so lets update it; also we can
reach F via 0+2 = 2 < ∞.

C & F were updated via A, let's pick is low



(2, F)
(3, C)

now let's pick the next node (with min dist) to run which is node F (dis(F) < dis[C]).

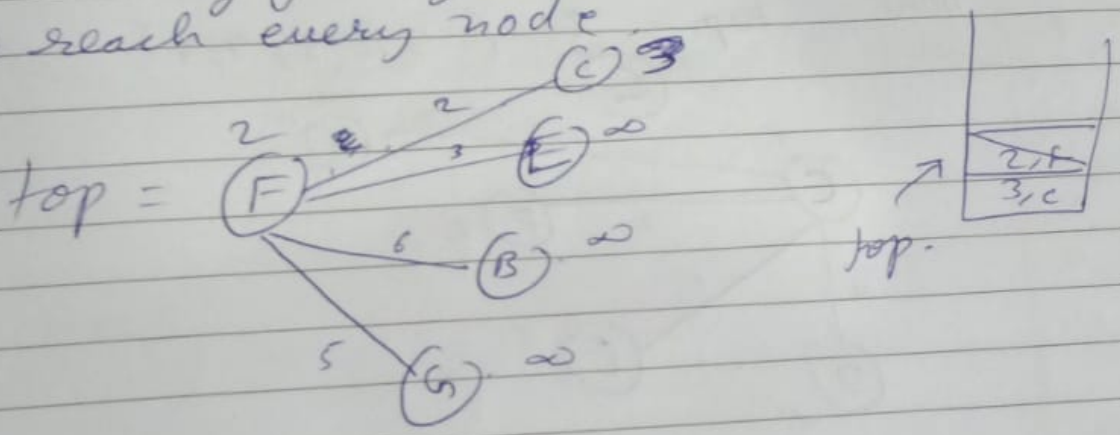
~~State C is~~

imp note { We need to remove any PAIR entry for a given 'node' & update new entry pair for it because the min distance for that node is not the same

note: Here we picked F rather than C. now because currently are to our knowledge we can reach next stop with least dist of 2.

This does not mean that our shortest path is A → F
, no.

we are only finding the min distance required to reach every node.

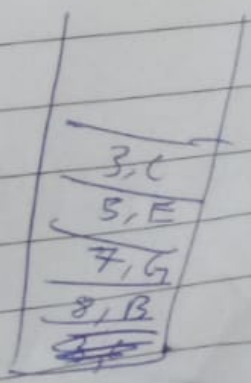


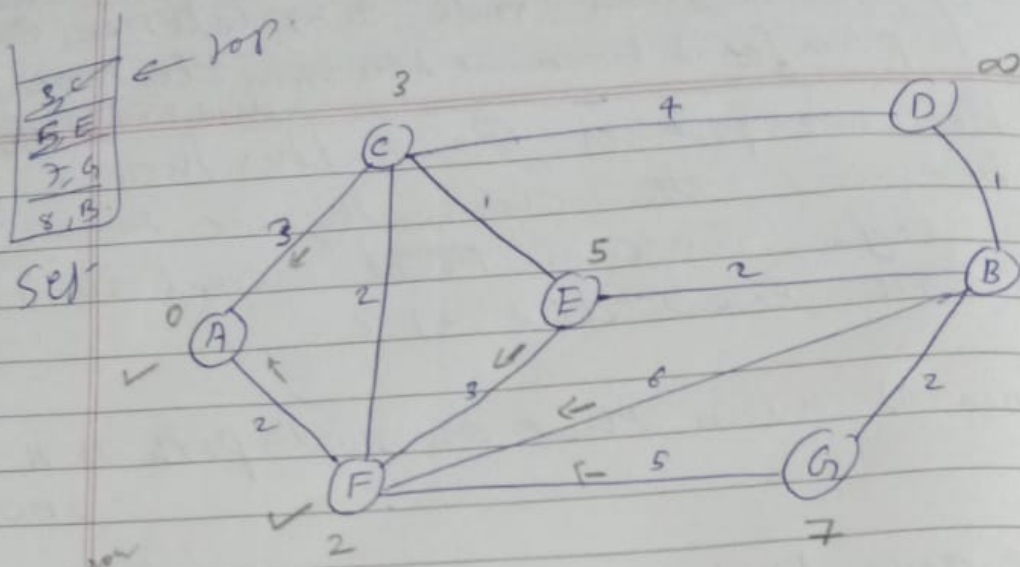
$2+2 < 3$ ✗

$2+3 < E(\infty)$ ✓ updated E.

$2+6 < B(\infty)$ ✓ " B

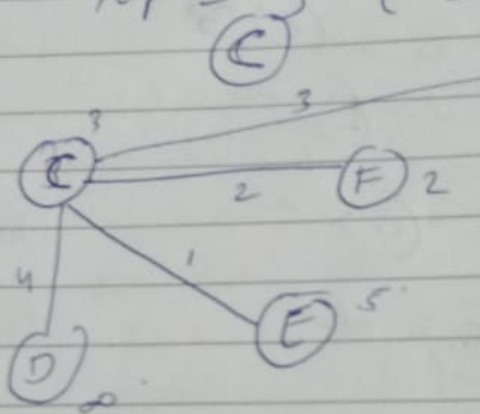
$2+5 < G(\infty)$ ✓ " G



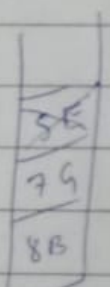


mark arrow from B, E, G to F

now top = 3 (3 < 5, 7, 8)



cons B, F, G are updated with prev F



deleted

3 + 3 < 0 (A) ✗

3 + 2 < F (2) ✗

3 + 1 < F (5) ✓ update

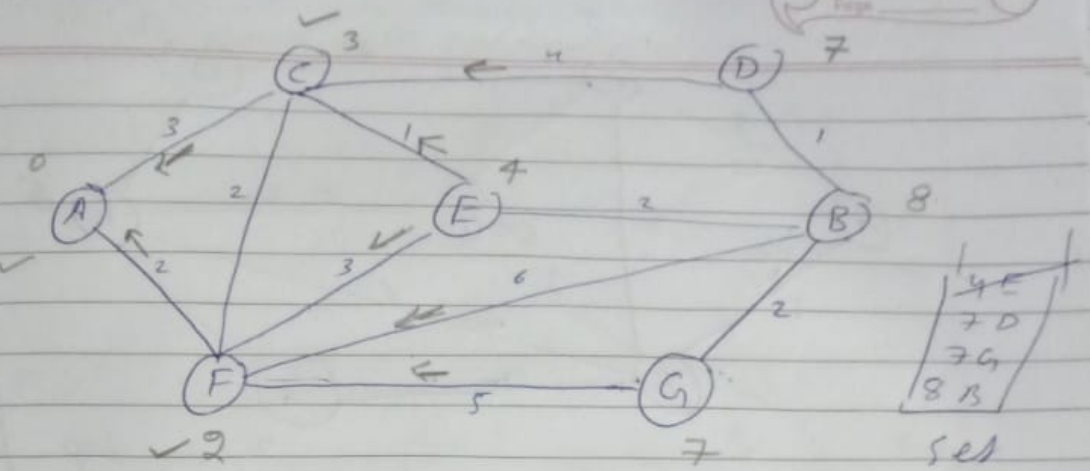
3 + 4 < D (∞) ✓ to D

F removed

4,E
~~4,E~~
7 D
7 G
8 B

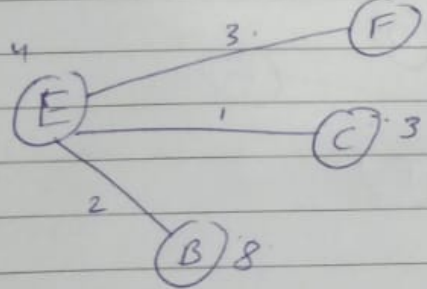
new set

for E (5,E)



E, D are updated via C

now, top = 4 (E) (4 < 7, 8, 7)



4 + 3 < F (2) ✗

4 + 1 < C (3) ✗

→D
→G
~~8 B~~ deleted

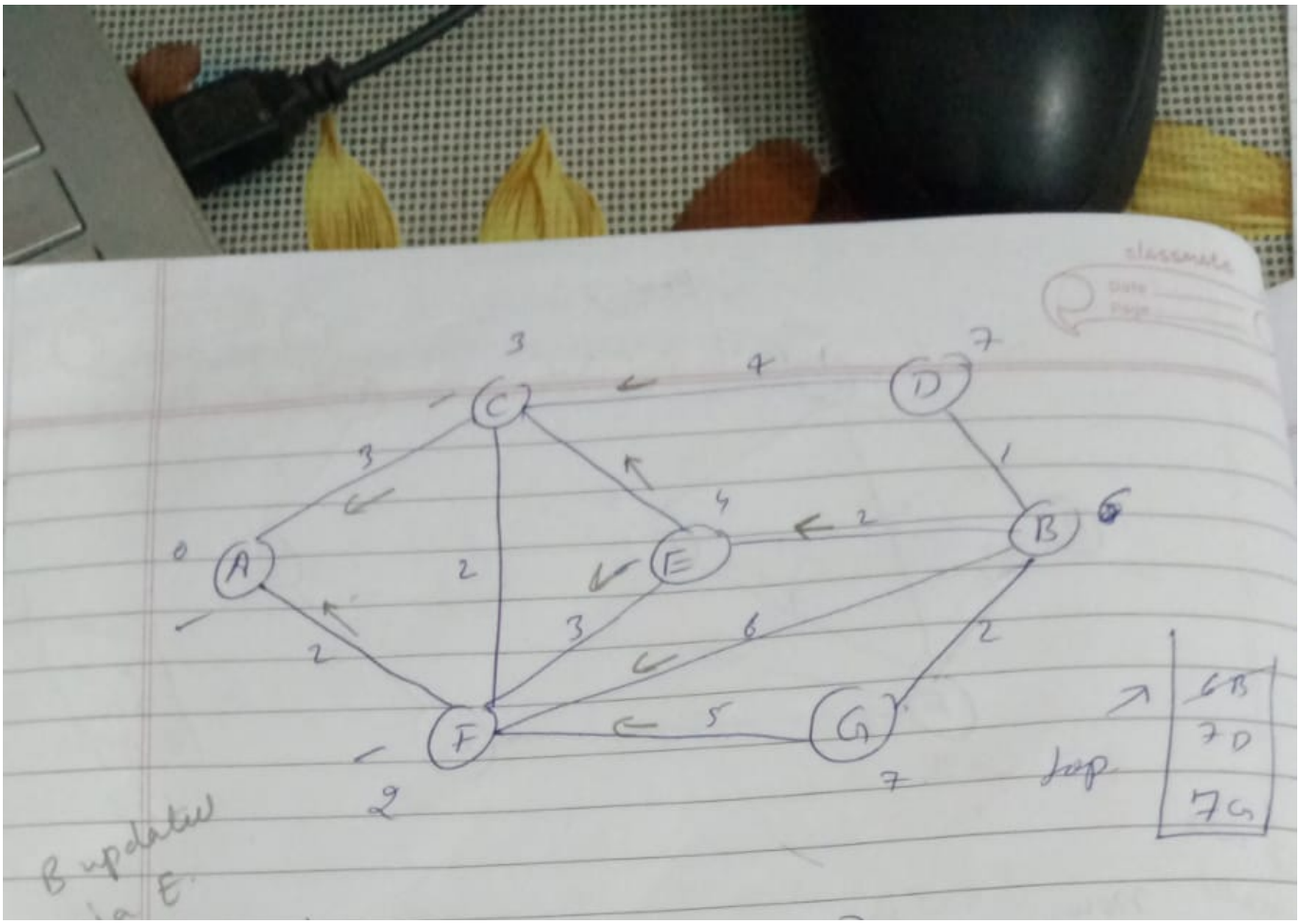
4 + 2 < B(8) ✓ update B.

old set

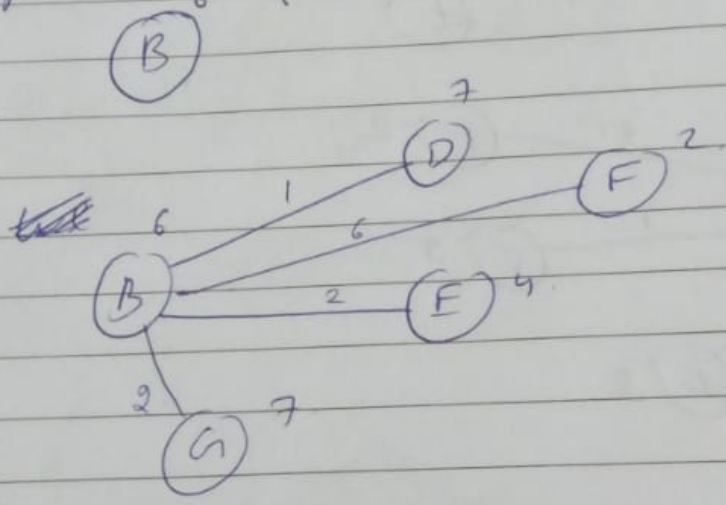
6, B
→D
→G

increments for B. (6, B deleted)

new set.



top = 6 (6 < 7, 7).



6 + 1 < D(7) ✗

6 + 6 < F(2). ✗

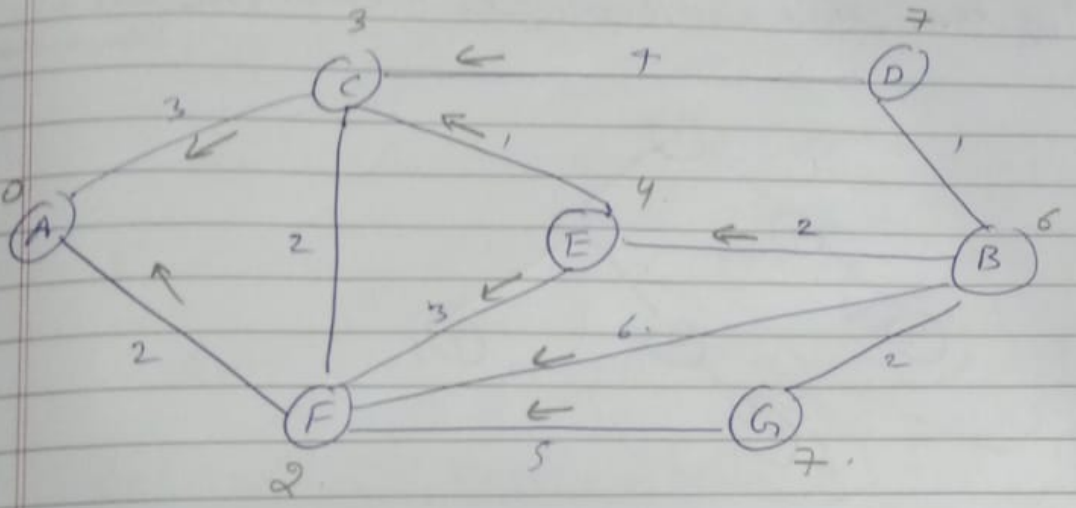
6 + 2 < E(4) ✗

6 + 2 < G(7) ✓.

T. O(E log V)

S. O(E + V).

Finally we have graph as:



∴ (A) to (B) min dist = 6 (shortest distance)

now back way back from (B) to (A) via parent

B ⟶ E ⟶ C ⟶ A (Shortest path)

Code :-

```cpp
#include <bits/stdc++.h>

/*[✔️ ⭐ [App-1.1] more faster then prv] App-1

    Title = Dijkstra's Algo Using Set<pair(minDis to reach node ,node) >

    explanation :- what we do here is we use a set(minDis to reach node,node) and using this every time we pop the front
                   of this set, we will get the pair with least 'minDis'.
                   (we can also use minHeap instead of this), so every time we will follow a 2 step process after fetching the front
                        step 1 -> Relaxation :- here if the minDis[front] + distance between neigh and front is lesser then the minDist[neigh] then we will update the minDist[neigh]
                                                  and also insert this new pair for node -> 'neigh' into the set, make sure to delete the prv entry of the 'neigh' from the set if exists any

                        step 2 -> find the next node to do the same process (for this we will choose the node with the min Distance, which is not yet explored i.e present in the set)

    note: .find() .insert() .erase() takes O(logN) times to perform because we are using 'ordered_set' not an UNordered_set

    ☑T : O(E*log*V) - traversing each edge takes E and inserting the verticies into set will take logV
    ☑S : O(E+V)

*/

// fun.2

void createAdjList(vector<vector<int>> &vec, unordered_map<int,list<pair<int,int>>> &AdjList){

    for(int i=0; i < vec.size(); i++){
        int node1 = vec[i][0];
        int node2 = vec[i][1];
        int weight = vec[i][2];

        // undirected graph
        AdjList[node1].push_back({node2, weight});
        AdjList[node2].push_back({node1, weight});

    }
}

// fun.3
void dijkstraMinDis(unordered_map<int,list<pair<int,int>>> &AdjList,
```

```
set<pair<int,int>> &set, vector<int> &minDist, int src){

    // initially we can reach the src itself in 0 distance, and insert its pair{0,src}
into the set also
    minDist[src] = 0;
    set.insert({0,src});

    // run a loop while set is non empty
    while(!set.empty()){

        // fetch the top node of set (the one with the minimum distance)
        pair<int,int> frontPair = *(set.begin());
        set.erase(set.begin()); // pop the front

        int frontMinDis = frontPair.first;
        int frontNode = frontPair.second;

        // explore all the neighbours of this frontnode
        for(auto neighPair:AdjList[frontNode]){

            int neigh = neighPair.first;
            int weight = neighPair.second; // this weight is the 'distance from
frontNode to neigh'

            // now update the min distance to reach the 'neigh' node if and only if
the new path distance is smaller then the prv.
            if(minDist[frontNode] + weight < minDist[neigh]){

                // now after a node's distance to reach is updated, we need to update
that 'neigh''s min distance in the set also, so for that (find if there already exist
a node 'neigh' in the set, if yes then delete it) and then insert the new pair into
the set
                auto prvEntry = set.find({minDist[neigh] ,neigh});
                if(prvEntry != set.end()){ // another entry for 'neigh' exists
                    set.erase(prvEntry);
                }

                // updation of minDist of 'neigh'
                minDist[neigh] = minDist[frontNode] + weight;

                // insertion of new pair in set
                set.insert({minDist[neigh],neigh});
            }
        }

    }
}

// main function
vector<int> dijkstra(vector<vector<int>> &vec, int vertices, int edges, int source) {
```

```
    // we need a 'AdjList', a 'minDist' (to return at the end), and a
set<pair<int,int>> to store the {minDis to reach node, node}
    unordered_map<int,list<pair<int,int>>> AdjList;
    vector<int> minDist(vertices,INT_MAX); // initially all nodes can be reached only
in inf distance
    set<pair<int,int>> set; // order this on basis of the pair.first i.e min dist

    // create AdjList by fun.2
    createAdjList(vec, AdjList);

    // now to apply dijkstra call fun.3
    dijkstraMinDis(AdjList, set, minDist, source);

    return minDist;



}
```

---------- END ----------