**Assignment 7/8**

**Problem 1:**

**My algorithm:**

I decided to use Dynamic programing

Complexity -> O(n^2)

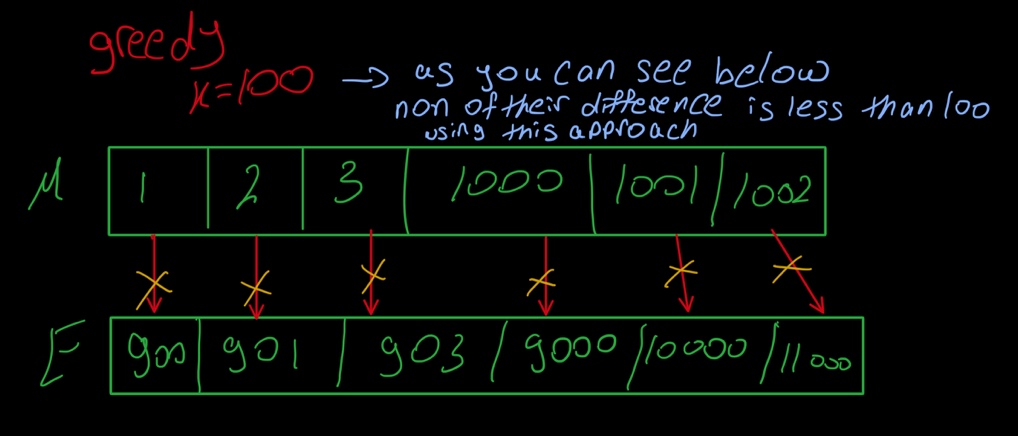
**Problem 2:**

**This problem is like either the Halloween problem or the shortest path problem in a 1 D, I have 2 approaches to solve this problem**

**First approach is greedy algorithm:**

* As we have both ethical and malicious in one array
* So, we have to separate them in two different arrays
* Second, assume that the zones are in ascending order in the x axis, so sort both arrays (ethical and malicious) according to the zone number
* Then loop over the two arrays and assigning ith ethical to the ith malicious hacker if their difference is less than K
* Complexity can be [O (n logn+n) =O(n)]

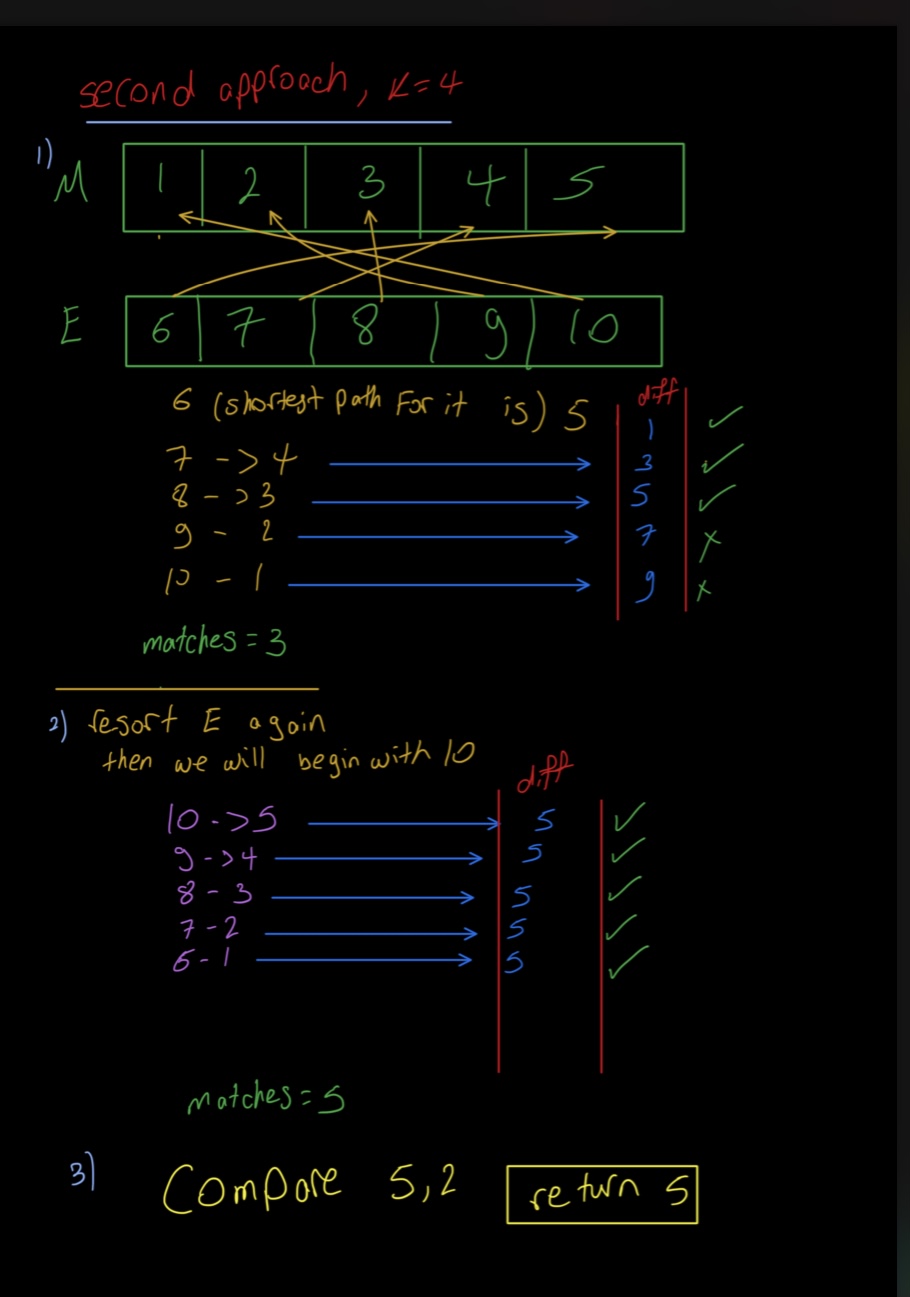
This approach won’t be the most optimal way: see the below example



**Second approach is greedy algorithm:**

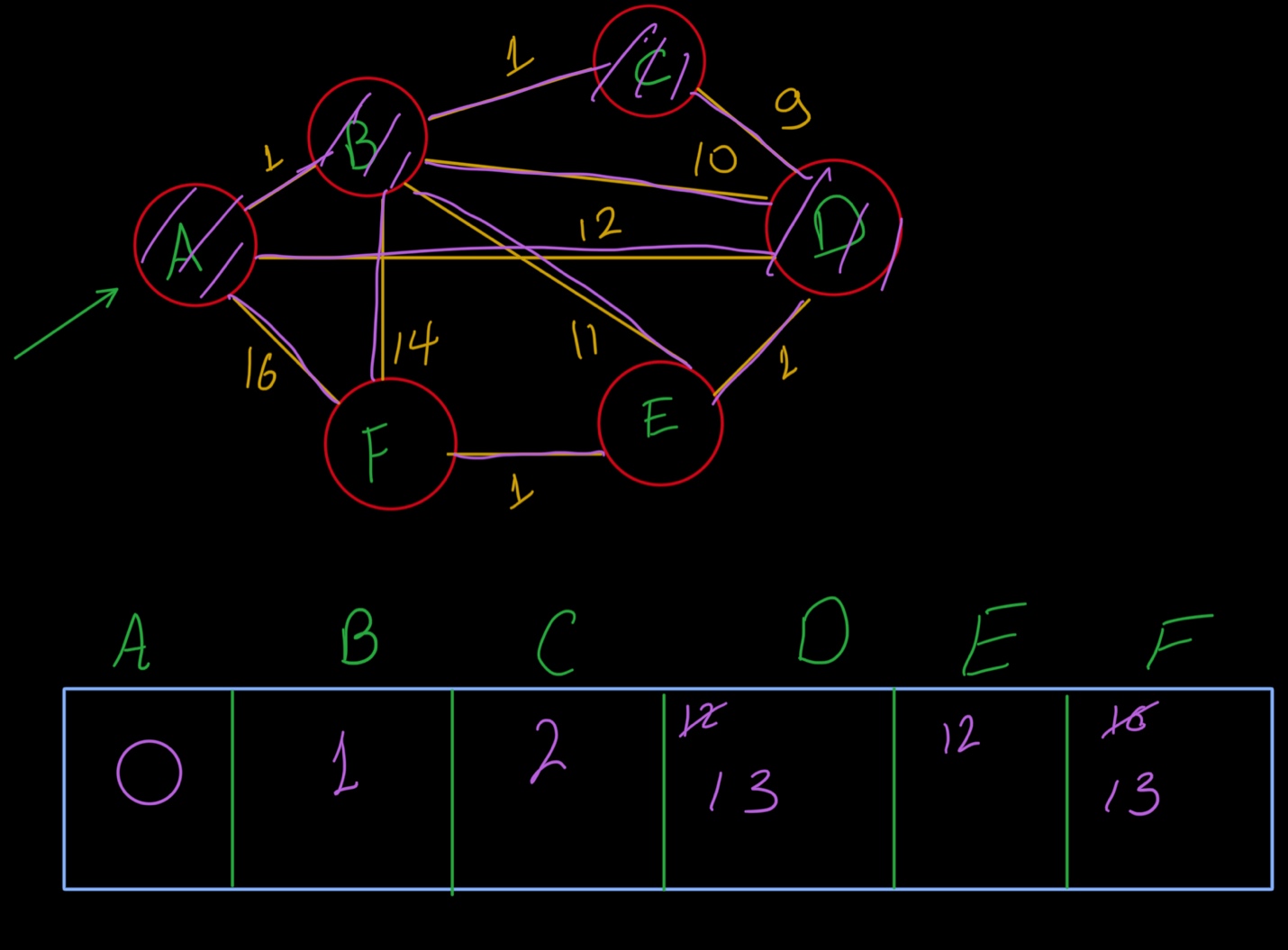
* also divide E and M in two different arrays
* then, also sort both of them
* now check the closest M to ith **E**
* then mark this M visited
* repeat this to assign all the ethical hacker to M
* and then put the result of the number of matches in a variable
* did we get the optimal thing now, not yet?
* we need to repeat all of these comparisons (closest M to every E), but re-sort the E array descending and re-initialize everything, and repeat all of previous algorithm and put the result in another variable and compare both variable and check which is bigger and return it
* Complexity here -> O(n^2 +nlogn) ->O(n^2)

see the below example



**Problem 3:**

**a)**



b) Getpath(X s, X e) function in the dikstra.cpp

**Problem 4:**

The problem is optimal merge algorithm, where I used priority queue, where the it pops always the smallest two numbers and then push the sum of them

Complexity -> pushing and popping from the priority queue (logn)  
so pushing and popping n time make the complexity ->O(nlogn)

**Problem 5:**

Implementation: Covid.cpp which also contains a detailed comments and description for the code

Complexity **->** O(n)

**Problem 6:**

When dividing a tree into many trees across many processors, every processor will take probably logn to compute the MST, and in the process of merging them again it will merge n spanning trees into one tree, which is O(logn) complexity

Kruskal’s algorithm: when using min heaps its complexity can be O(nlogn)

So, there is no difference in the complexity, so it is not better nor worst

**Problem 7:**

**Please check the small pdf for this Question where there is a detailed algorithm and pseudo code**

|  |
| --- |
| #include <iostream>  **using** **namespace** std;  **void** **count**(**int** Grid[][**5**], **int** rows, **int** cols){  **int** count=**0**;  **for**(**int** i=**0**;i<rows;i++)  **for**(**int** j=**0**;j<cols;j++)  **if**(Grid[i-**1**][j+**1**]!=**1**&&Grid[i][j-**1**]!=**1**&&Grid[i-**1**][j]!=**1**&&Grid[i][j]==**1**)  {  // cout<<Grid[i][j-1]<<" "<<Grid[i-1][j]<<" "<<i<<" "<<j<<" ";  count++;  cout<<endl;  }  cout<<count;  }  **int** **main**()  {    // int Grid[5][8] =  // { 0, 0, 0, 0, 0, 0, 0, 0,  // 0, 1, 1, 1, 1, 0, 0, 0,  // 0, 1, 0, 1, 0, 0, 1, 0,  // 0, 1, 1, 1, 1, 0, 1, 0,  // 0, 0, 0, 0, 0, 0, 0, 0, };  //i ignored the blue X here in my grid but it is supposed to be 1's and i dont loop over them  **int** Grid[**3**][**5**] =  {**0**,**0**,**0**,**0**,**0**,  **0**,**0**,**0**,**1**,**0**,  **0**,**1**,**1**,**0**,**0** };  count(Grid,**3**,**4**);  **return** **0**;  } |