End Semester Examination 2020

(Design & Analysis of Algorithms)

Puja Mondal
Roll : CRS1905
Email-Id : pujamondal9779@gmail.com
Phone Number : 8240254291
pujamondal9779.wixsite.com
github.com/CRS-1905

1.(a) Describe an efficient algorithm to calculate the Least Common Multiple of two positive integers, each of size n bits.

Ans:
Let a and b be two positive integers each of size n bits.
Then Least Common Multiple of two positive integers a and b is given by lcm(a,b) = $\frac{a\times b}{gcd(a,b)}$.
The pseducode for lcm is given by:

```
Input: two positive integers a and b of size n bits.
Method:
1. $r_0 \leftarrow a$
2. $r_1 \leftarrow b$
3. $i \leftarrow 1$
4. while $r_i \neq 0$ do
5.     $q_i \leftarrow \frac{r_{i-1}}{r_i}$
6.     $r_{i+1} \leftarrow r_{i-1} - q_i r_i$
7.     $i \leftarrow i+1$
8. end while
9. $gcd \leftarrow r_{i-1}$
10. $p \leftarrow \frac{a\times b}{gcd}$.
11. return p
```

(b) Analyse the time complexity of your algorithm. You should clearly explain the time complexity of multiplication/division of integers in case you exploit such a routine.

Ans:
In the step 5. we do division of two integers of size atmost n.
So we now find the time complexity of division of two integers of size atmost n:
Let n = $a_{n-1}a_{n-2}...a_0$ and d =$b_{m-1}b_{m-2}...b_0$, where d is divisor and n is divident.
Then we do as follows:
If m > n then return quotient q = 0 and remainder R = n.
Else d goes to first m bits of the divident one time and subtracted from the devident. If the subtraction is positive
then we put q = q << 1 | 1. and we modify the divident as the subtraction . If the subtraction is negative, then we rightshift the divisor one time and put q = q << 1 and then continue the above steps untill the bits in the divident have been exhausted.

Here in each step we do subtraction of two m bits number and check whether it is greater than 0 or less than 0 . So in each step it take $O(m)$ time. Each step size of divident is decrease by atleast 1, so the total number of step is atmost n. So the total time complexity is $O(mn)$.
So here in step5 the time complexity is $O(n^2)$.

In the step 6 we do multiplication and subtraction:
Suppose we want to multiply a k-bit integer n by a l-bit integer m.
We do as follows:
Take temp=0.
In the ith step, if m[i] = 1, then temp = temp + (n<<i).
We repeat the process untill the bits in the integer m have been exhausted.
We copy down that many right-most bits of the partial sum, and then add to n the integer formed from the rest of the partial sum. So in each step it takes $O(k)$ bit operations.
There are atmost l many step, So the total bit operation is atmost $O(kl)$. So the time complexity is $O(mn)$.
So here in step 6 the time complexity is $O(n^2)$.

Now we have to find atmost how many number the while loop is running.
Now claim that in the euclidean algorithm r$_{j+2}$ < $\frac{r_j}{2}$.
Proof of the claim: If r$_{j+1}$ ≤ $\frac{r_j}{2}$,
then we have r$_{j+2}$ < r$_{j+1}$ ≤ $\frac{r_j}{2}$. So we are done.
If r$_{j+1}$ > $\frac{r_j}{2}$,
Then $\frac{r_j}{2} < r_{j+1} < r_j$
i.e., $r_{j+1} < r_j < 2r_{j+1}$.
Now $r_j = r_{j+1}q_{j+1} + r_{j+2}$ implies $q_{j+1} = 1$,
because if $q_{j+1} \geq 2$, then $r_j \geq 2r_{j+1} + r_{j+2}$. i.e., $r_j \geq 2r_{j+1}$, which is a contradiction.
So $q_{j+1} = 1$, then $r_{j+2} = r_j$ -$r_{j+1} < r_j$ - $\frac{r_j}{2} = \frac{r_j}{2}$.
So proof our claim.
So atmost 2× n times the while loop can running.
So the time complexity upto finding the gcd is $O(n) \times (O(n^2))$ i.e., $O(n^3)$.
In the 9th step we do multiplication and then divide it by gcd(a,b)
In this step the time complexity is $O(n^2) + O(n^2)$ i.e., $O(n^2)$.
So the total time complexity of this algorithm is $O(n^3)$.

(c) Will your algorithm always work correctly in case you implement it with C programming language? Explain with pieces of C code.

Ans: No this algorithm does not work correctly in case we implement it with C programming language.
For example: We write the C code to finding lcm of two positive integers with 32 bits.
The C-code is given by:

```
#include<stdio.h>
#include<stdlib.h>
int main(){
        unsigned int a,n,q,a1,n1,r,lcm;
        printf("Enter Two Numbers :\t");
        scanf("%u%u",&n1,&a1);
        a=a1;
        n=n1;
        if(a == 0)
                printf("The gcd is %u\n",n);
        else{
                while(a!=0){
                        q=n/a;
                        r=n%a;
                        n=a;
                        a=r;
                }
                printf("The gcd is %u\n",n);
```

```
            }
            lcm=(a1*n1)/n;
            printf("lcm of %u and %u is %u\n",n1,a1,lcm);
    }
```

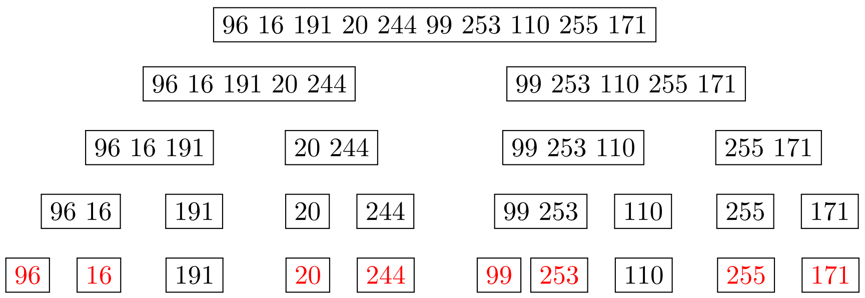
Here if we take a1 = 2^{20} = 1048576 and n1 = 1048575, then lcm have to be be a1 $\times$ n1 ,i.e., 1099510579200. But here the output of this program is 4293918720.
Because two numbers are of 21 bits and 20 bits respectively and the multiplication of them will be 41 bits which is greater than 32 bits. Now we have used multiplication operation of two 32 bits integers and saved the value in a 32 bits integer so there will be integer overflow and that's why we are having incorrect result.

3.(a) Write down a C program to implement the merge sort algorithm.
Ans: The program is as follows:

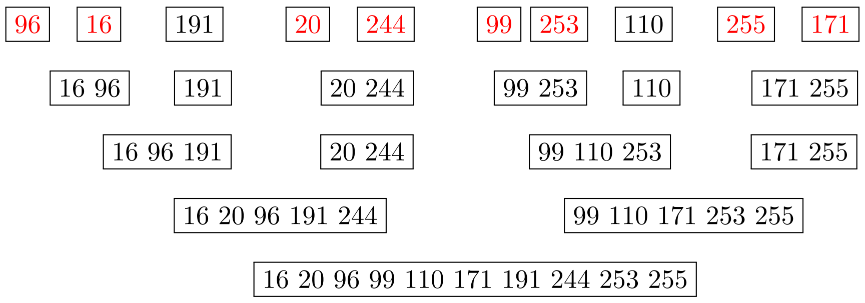
```
#include<stdio.h>
#include<stdlib.h>
int *num,n;
void Merge_sort(int l,int r);
void Merge(int l,int m,int r);
int main(){
        int i;
        printf("Enter The No Of Numbers :\t");
        scanf("%d",&n);
        num=(int *)malloc(n*sizeof(int));
        printf("Enter The %d Numbers \t",n);
        for(i=0;i<n;i++)
                scanf("%d",&num[i]);
        Merge_sort(0,n-1);
        printf("The Sorting Numbers Are Given Bellow\n");
        for(i=0;i<n;i++)
                printf("%d\t",num[i]);
        printf("\n");
}
void Merge_sort(int l,int r){
        if(l<r){
                int i;
                int m=l+((r-l)/2);
                Merge_sort(l,m);
                Merge_sort(m+1,r);
                Merge(l,m,r);
        }
}
void Merge(int l,int m,int r){
        int n1,n2,i,j,k;
        n1=m-l+1;
        n2=r-m;
        int L[n1],R[n2];

        for(i=0;i<n1;i++)
                L[i]=num[l+i];
        for(j=0;j<n2;j++)
                R[j]=num[m+1+j];
        i=0;j=0;k=l;
        while(i<n1 && j<n2){
                if(L[i]<=R[j]){
                        num[k]=L[i];
                        i++;
                }
                else{
                        num[k]=R[j];
                        j++;
                }
                k++;
        }
        while(i<n1){
                num[k]=L[i];
                i++;
                k++;
        }
        while(j<n2){
                num[k]=R[j];
                j++;
                k++;
        }
}
```

(b) Let $x_{-1} = 1$ and x_0 be your day of birth ($1 \le x \le 31$). Create 16 integers as $x_i = (x_{i-1} + 1)^2 \bmod 97 + x_{i-2}$.
Demonstrate how your implementation executes on x_1 , x_2 , . . . , x_{10} to sort them in ascending order?
Ans: Here $x_{-1} = 1$, $x_0 = 16$.
From the relation $x_i = (x_{i-1} + 1)^2 \bmod 97 + x_{i-2}$,
we get $r_1 = 96$, $r_2 = 16$, $r_3 = 191$, $r_4 = 20$, $r_5 = 244$, $r_6 = 99$, $r_7 = 253$, $r_8 = 110$, $r_9 = 255$, $r_{10} = 171$.
We have to show that how this implementation executes on x_1 , x_2 , . . . , x_{10} to sort them in ascending order:
In this program Merge_sort(l,r),where l is the starting index and r is the last index. we done as follows:
1. check if l<r or not.if yes we do
2. Find the middle index of the array to divide it in two halves: m=$\frac{l+2}{2}$
3. Call the Merge_sort for 1st half: Merge_sort(l,m);
4. Call the Merge_sort for 2nd half: Merge_sort(m+1,r);
5. Merge two sorted array: Merge(l,m,r);
In the given case our input array is: 96 16 191 20 244 99 253 110 255 171

First we divide the given array into 2 half. Then we divide each part recursively until all the partitions has exactly 1 element.



First we merge the elements which are highlighted. Then all the parts are merged together in the order, it was devided before:



So we now compute the sorted array is 16 20 96 99 110 171 191 244 253 255.

4. Let M be an (n×n) matrix where each element is a distinct positive integer. Construct another matrix M′ by permuting the rows and/or permuting the columns, such that the elements of at least one row appear in increasing order (while looking from left to right) and those of at least one column appear in decreasing order (while looking from top to bottom).

(a) Describe an algorithm for constructing M′ . Justify the correctness of your pro- posal.
Ans: Let R_1, R_2,..., R_n be the n rows of the matrix M and C_1, C_2,..., C_n be the n columns of the matrix M. We have to find a matrix M′ such that atleast one row appear in increasing order and atleast one column appear in decreasing order. The algorithm is as follows:

```
1. Given M = $(m_{ij})_{n\times n}$.
2. Let A be a array of size n and A = $\phi$
3. for i ← 1 to n
4.        A$_i$ = $m_{1i}$
5. Sorting the elements of A in decreasing order
6. Let Π be a permutation such that Π($j$)= position of $m_{1j}$ in A
7. Now we copy the elements of M to M′ such that C′$_{\Pi(j)}$ = C$_j$
8. M← $(M')^T$
9. A$_i$ = $m_{1i}$
10. Sorting the elements of A in increasing order
11. Let Π be a permutation such that Π($j$)= position of $m_{1j}$ in A
12.Now we copy the elements of M to M′ such that C′$_i$ = C$_{\Pi(i)}$
14. return M′
```

Here 1st we take the 1st row of the given matrix. we sort the row in decresing order. Then we permute the column of M w.r.t the permutation Π. Then in the modifying matrix the 1st row is in decreasing order. So after transforming the matrix the 1st column is in decreasing order. Now again we sort the 1st row in increasing order. Then we find Π. Then we permute the column wrt the permutation Π. Then in the modifying matrix we get the 1st row in increasing order and since the element of some column is unchange after the permutation, so we already have a column in decreasing order. So we must get the matrix M ′ by using the above algorithm.

For example : if we take a matrix M as follows: M = $\begin{bmatrix} 10 & 2 & 20 & 8 \\ 6 & 5 & 16 & 21 \\ 7 & 15 & 11 & 25 \\ 14 & 1 & 30 & 3 \end{bmatrix}$
Then we take the 1st row of the matrix M and put it to a array A
So A = $\begin{bmatrix} 10 & 2 & 20 & 8 \end{bmatrix}$.
Thenb sorting the elements of A in decreasing order we get
A = $\begin{bmatrix} 20 & 10 & 8 & 2 \end{bmatrix}$.
Now we have to find such permutation Π.
Here Π(1) = 2, Π(2) = 4, Π(3) = 1, Π(4) = 3.
So M′ = $\begin{bmatrix} C_3 & C_1 & C_4 & C_2 \end{bmatrix}$.

$$= \begin{bmatrix} 20 & 10 & 8 & 2 \\ 16 & 6 & 21 & 5 \\ 11 & 7 & 25 & 15 \\ 30 & 14 & 3 & 1 \end{bmatrix}$$

So we modify M as M = $(M')^T$.

I.e., M = $\begin{bmatrix} 20 & 16 & 11 & 30 \\ 10 & 6 & 7 & 14 \\ 8 & 21 & 25 & 3 \\ 2 & 5 & 15 & 1 \end{bmatrix}$

We now continue the similar process as follows:
1st take the 1st row of M in A i.e., A = $\begin{bmatrix} 20 & 16 & 11 & 30 \end{bmatrix}$.
Then after sorting this array in increasing order we get A = $\begin{bmatrix} 11 & 16 & 20 & 30 \end{bmatrix}$.
We find such Π as $\Pi(1) = 3$, $\Pi(2) = 2$, $\Pi(3) = 1$, $\Pi(4) = 4$.

So M$'$ = $\begin{bmatrix} 11 & 16 & 20 & 30 \\ 7 & 6 & 10 & 14 \\ 25 & 21 & 8 & 3 \\ 15 & 5 & 2 & 1 \end{bmatrix}$

Here 1st row is in increasing order and the 3rd column is in decreasing order.

(b) What is the time complexity of your algorithm? Do you think this is optimal?
Ans: In the step 3-step 4 we copy n elements in a array, so its takes $O(n)$ operations.
In step 5 we sorting the array of n elements, so its takes time $O(nlogn)$.
In step 6 we have to find permutation so it takes time $O(nlogn)$
In step 7 we copy n^2 elements, so it takes time $O(n^2)$
In step 8 we copy n^2 elements, so it takes time $O(n^2)$
In the step 9 we copy n elements in a array, so its takes $O(n)$ operations.
In step 10 we sorting the array of n elements, so its takes time $O(nlogn)$.
In step 11 we have to find permutation so it takes time $O(nlogn)$
In step 12 we copy n^2 elements, so it takes time $O(n^2)$
So the total time complexity is $O(n^2)$.

Yes, this is optimal because if we just fill up the elements in M$'$, then it takes at least $O(n^2)$ times and we did it in $O(n^2)$ complexity.

(c) Propose a data structure that supports your algorithm. Clearly explain how much additional storage, other than the matrix itself, is required in your algorithm.

Ans: We use two 1 dimensional array of integers, one is use for sorting and another is use for storing the permutation and we use one 2-dimentional array of integers to store the matrix. Here we use two 1 dimensional arrays each of size n and one two dimensional array which is of size n^2. So the total required space is $O(n^2)$.

5.(a) Describe an algorithm to solve the All Pair Shortest Path problem in a directed weighted graph.
Ans: Let G = (V, E) be a directed weighted graph and W =(w_{ij}) be the weight matrix of the graph. Let D$^{(k)}$ =(d$_{ij}{}^{(k)}$), where d$_{ij}{}^{(k)}$ be the weight of the shortest path from i to j where the all intermediate vertices are in the set {1, 2, 3, ..., k}.
i.e., d$_{ij}{}^{(k)}$ = w$_{ij}$, if k = 0, because the shortest path where there is no intermediate vertex, is the edge (i, j).
= min(d$_{ij}{}^{(k-1)}$, d$_{ik}{}^{(k-1)}$+d$_{kj}{}^{(k-1)}$), if k $\geq$ 1
Because we know that the weight of the shortest path from i to j is d$_{ij}{}^{(k-1)}$ where the intermediate vertex are {1, 2, 3, ...,k-1}. We want to find the weight of the shorted path from i to j where the intermediate vertices are {1, 2, 3,...,k}. Now if the weight of the shortest path from i to k where the intermediate vertices are {1, 2, .., k-1} + the weight of the shortest path from k to j where the intermediate vertices are {1, 2, .., k-1} is less than the weight of the shortest path from i to j where the intermediate vertices are {1, 2, .., k-1}, then we take the shortest path from i to k and from k to j. Otherwise the path will remain same.
We now initialize a predecessor matrix $\Pi^{(0)}$ as follows:

$$\Pi_{ij}^{(0)} = \begin{cases} NIL, \text{ if } i=j \text{ or } w_{ij} = \infty \\ 0, \text{ othewise.} \end{cases}$$

Since if w$_{ij}$ = ∞ or i = j , that means (i, j) is not a edge. So there is no path from i to j where there is no intermediate vertex. So in that case the predecessor is NIL.
Else (i, j) is an edge, so the shortest path is that edge and the predecessor of j is i.

And for k$\geq$1,

$$\Pi_{ij}^{(k)} = \begin{cases} \Pi_{ij}^{(k-1)}, \text{ if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \Pi_{kj}^{(k-1)}, \text{ otherwise} \end{cases}$$

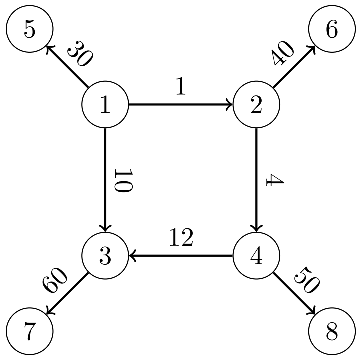
if d$_{ij}{}^{(k-1)}$ $\leq$d$_{ik}{}^{(k-1)}$+d$_{kj}{}^{(k-1)}$, then $d_{ij}{}^{(k-1)}$ is the weight of the shortest path from i to j where intermediate vertices are {1, 2, ...,k}. i.e., path remains unchange although we add k as a intermediate vertex. So the predecessor of j will remain unchanged.
But if d$_{ij}{}^{(k-1)}$ >d$_{ik}{}^{(k-1)}$+d$_{kj}{}^{(k-1)}$, then the shortest path from i to j is i to k and k to j, where the intermediate vertices are {1, 2, .., k}. So here the predecessor of j is the predecessor of j of the path from k to j.
So $\Pi_{ij}{}^{(k)} = \Pi_{kj}{}^{(k-1)}$.
Now we consider the following algorithm :

FLOYD-WARSHALL (W)

```
1. n← $W.row$
2. for i ← 0 to n
3.     for j ← 0 to n
4.         d$_{ij}{}^{(0)}$ = w$_{ij}$
5.         if i = j or w$_{ij}$ = $\infty$
6.             $\Pi_{ij}{}^{(0)}$ = NIL
7.         else
8.             $\Pi_{ij}{}^{(0)}$ = i
9. for k← 1 to n
10.    for i ← 0 to n
11.        for j ← 0 to n
12.            if d$_{ij}{}^{(k-1)}$ ≤d$_{ik}{}^{(k-1)}$+d$_{kj}{}^{(k-1)}$ then
13.                d$_{ij}{}^{(k)}$ = d$_{ij}{}^{(k-1)}$
14.                $\Pi_{ij}{}^{(k)}$ = $\Pi_{ij}{}^{(k-1)}$
15.            else
16.                d$_{ij}{}^{(k)}$ = d$_{ik}{}^{(k-1)}$+d$_{kj}{}^{(k-1)}$
17.                $\Pi_{ij}{}^{(k)}$ = $\Pi_{kj}{}^{(k-1)}$
18. return (D$^n$ and $\Pi^n$)
```

(b) What is the time complexity of your proposed algorithm?
Ans: We have to find the complexity of the above algorithm.
From the step2 to the step 8 we copy the matrix W into the matrix D$^{(0)}$, and check whether i = j or w$_{ij}$ = ∞, then put the value in Π_{ij}, so it takes $O(n^2)$ oprations.
Then for each k,i and j we check d$_{ij}{}^{(k-1)}$ $\leq$d$_{ik}{}^{(k-1)}$+d$_{kj}{}^{(k-1)}$, and write elemnents of two matrices D$^{(k)}$ and $\Pi^{(k)}$
So for each case we have done $O(n^2)$ operations.
So from the step 9 to 17 we have done $O(n^3)$ oprations.
So the total time complexity is $O(n^3)$.

(c) Explain the algorithm taking a suitable graph of 8 vertices as an example.
Ans: We consider the following graph with 8 vertices:



Here the weighted graph is given by:

$$W = \begin{bmatrix} 0 & 1 & 10 & \infty & 30 & \infty & \infty & \infty \\ \infty & 0 & \infty & 4 & \infty & 40 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & \infty & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

So here

$$D_0 = \begin{bmatrix} 0 & 1 & 10 & \infty & 30 & \infty & \infty & \infty \\ \infty & 0 & \infty & 4 & \infty & 40 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & \infty & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_0 = \begin{bmatrix} NIL & 1 & 1 & NIL & 1 & NIL & NIL & NIL \\ NIL & NIL & NIL & 2 & NIL & 2 & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & NIL & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

$$D_1 = \begin{bmatrix} 0 & 1 & 10 & \infty & 30 & \infty & \infty & \infty \\ \infty & 0 & \infty & 4 & \infty & 40 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & \infty & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_1 = \begin{bmatrix} NIL & 1 & 1 & NIL & 1 & NIL & NIL & NIL \\ NIL & NIL & NIL & 2 & NIL & 2 & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & NIL & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 1 & 10 & 5 & 30 & 41 & \infty & \infty \\ \infty & 0 & \infty & 4 & \infty & 40 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & \infty & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_2 = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 & 2 & NIL & NIL \\ NIL & NIL & NIL & 2 & NIL & 2 & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & NIL & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 0 & 1 & 10 & 5 & 30 & 41 & 70 & \infty \\ \infty & 0 & \infty & 4 & \infty & 40 & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & 72 & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_3 = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 & 2 & 3 & NIL \\ NIL & NIL & NIL & 2 & NIL & 2 & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 1 & 10 & 5 & 30 & 41 & 70 & 55 \\ \infty & 0 & 16 & 4 & \infty & 40 & 76 & 54 \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & 72 & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_4 = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 & 2 & 3 & 4 \\ NIL & NIL & 4 & 2 & NIL & 2 & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

$$D_5 = \begin{bmatrix} 0 & 1 & 10 & 5 & 30 & 41 & 70 & 55 \\ \infty & 0 & 16 & 4 & \infty & 40 & 76 & 54 \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & 72 & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_5 = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 & 2 & 3 & 4 \\ NIL & NIL & 4 & 2 & NIL & 2 & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

$$D_6 = \begin{bmatrix} 0 & 1 & 10 & 5 & 30 & 41 & 70 & 55 \\ \infty & 0 & 16 & 4 & \infty & 40 & 76 & 54 \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & 72 & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_6 = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 & 2 & 3 & 4 \\ NIL & NIL & 4 & 2 & NIL & 2 & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

$$D_7 = \begin{bmatrix} 0 & 1 & 10 & 5 & 30 & 41 & 70 & 55 \\ \infty & 0 & 16 & 4 & \infty & 40 & 76 & 54 \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & 72 & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_7 = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 & 2 & 3 & 4 \\ NIL & NIL & 4 & 2 & NIL & 2 & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

$$D_8 = \begin{bmatrix} 0 & 1 & 10 & 5 & 30 & 41 & 70 & 55 \\ \infty & 0 & 16 & 4 & \infty & 40 & 76 & 54 \\ \infty & \infty & 0 & \infty & \infty & \infty & 60 & \infty \\ \infty & \infty & 12 & 0 & \infty & \infty & 72 & 50 \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix} \text{ and } \Pi_8 = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 & 2 & 3 & 4 \\ NIL & NIL & 4 & 2 & NIL & 2 & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & 3 & NIL \\ NIL & NIL & 4 & NIL & NIL & NIL & 3 & 4 \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \\ NIL & NIL & NIL & NIL & NIL & NIL & NIL & NIL \end{bmatrix}$$

6.(a) Explain the Longest Common Subsequence Problem with a suitable example.
Ans: Let A and B be two sequences. Then the Longest Common Subsequence Problem is to find the common subsequence of A and B which is longest.
For example: Let A = ”pujamondal” and B = ”priyasamanta”. Then the subsequences are ”pam”, ”amn”, ”mna” are of length 3 and ”pamna” is of length 5. There is no common subsequence of length greater than 5. So the longest common subsequence of A and B is ”pamna”.

(b) Write an algorithm in dynamic programming approach to solve this problem.
Ans:
Let X and Y be two sequence of length m and n respectively.

```
LCS(X,Y):
1. for i = 1 to m
2.        C[i,0] = 0
3. end for
4. for j = 1 to n
5.        C[0,j] = 0
6. for i = 1 to m
7.        for j = 1 to n
8.            if X[i] = Y[j]
9.                   C[i,j] = C[i-1,j-1] + 1
10.            end if
11.            Else
12.                   C[i,j] = max{C[i-1,j], C[i,j-1]}
13.            end Else
14.        end for
15. end for
16. index ← C[m][n]
17. while i,j>0 and index > 1
18.        while C[i,j] = C[i, j-1]
19.             j← j-1
20.        end while
21.        while C[i,j] = C[i-1, j]
22.             i← i-1
23.        end while
24.        index← index-1
25.        lcs[index] = X[i]
26.        i←i-1
27.        j←j-1
28. end while
29 return lcs
```

Here C[i,j] = length of LCS(X_i, Y_i), where X_i and Y_j are the substring of X and Y with length i and j respectively and it starting from beginnig.
If $X_i = Y_j$ then C[i,j] = 1 + C[i-1, j-1], because then LCS(X_i, X_j) = LCS(X_{i-1}, Y_{j-1}) followed by A.
Else C[i,j] = max{C[i-1, j-1], C[i-1, j]}, beacause LCS(X_i, X_j) = LCS(X_{i-1}, Y_j) when X_i is not in the LCS.
LCS(X_i, X_j) = LCS(X_i, Y_{j-1}) when Y_j is not in the LCS.
So we take the maximum value.

(c) What is the time complexity of your algorithm?

Ans: In the above algorithm first we find the LCS-length and then we find a LCS.
The time complexity to find the length of LCS:
When we find the i, j^{th} element C[i,j], then we check three thing i. whether $X_i = Y_j$, then put C[i,j] =

C[i-1,j-1] else we check the which value between C[i-1,j] and C[i,j-1] is maximum and C[i,j] is equal to that maximum value, here we did atmost three step which take $O(1)$ time. This loop run mn times, so total time taken to find the length of LCS is $O(mn)$

Time Complexity to recover the LCS:

Here in each step either we walk up or walk diagonaly or left in the $m\times$ n array. Here from the C[m,n] to C[i,0] or C[0,j] there are atmost m+n many steps. So the total time complexity to recover the LCS is $O(m+n)$.

So the total time complexity is $O(mn)$+$O(m+n)$ = $O(mn)$.

7. (a) Explain the basic idea of Greedy algorithms.

Ans: Greedy is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. So the problems where choosing locally optimal also leads to global solution are best fit for Greedy.

Activity Selection, Job Scheduling, Huffman Coding, Prims Algorithm, Krushkal Algorithm are greedy algorithms that works but Knapsack problem are greedy algorithm that does not work.

(b) Using this approach show how one can obtain the minimum spanning tree given an undirected weighted graph.

Ans: We use a greedy algorithm Krushkal algorithm to find the minimum spanning tree of the given undirected graph:

Let G = (V,E) be a undirected weighted connected graph where V is the set of vertices, E is the set of weighted edges.

Krushkal Algorithm (V, E):

```
1. Sort all the edges in non-decreasing order of their weights.
Let $e_1 = \{u_1, v_1\}, e_2 = \{u_2, v_2\}, ..., e_m$ be the sorted weighted edges.
2. Let E'=$\phi$
3. j$\leftarrow$ 0
4. for each vertex v$\in$ V
5.       Make-Set(v)
6. for each i$\leftarrow$ 0 to m
7.          if find-Set($u_i$)$\neq$Find-Set($v_i$)
8.                   E'=E'$\cup$ $\{\{u_i, v_i\}\}$
9.                   j$\leftarrow$j+1
10.                  if j=$|V|-1$ then
11.                       stop here
12. return (V,E')
```

In the 1st step we sorting the edges in non-decreasing order with respect to their weights.

Then we initialize a empty tree.

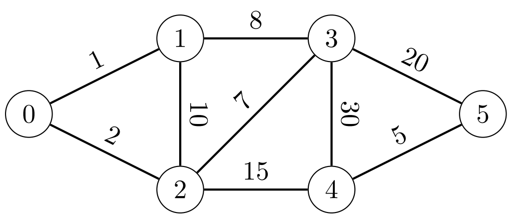
In the step2, step4 and 5 we initialize a trivial forest where each vertex in its own tree in the vorest.(This happen by usig Make-Set).

Then for each edge $\{u_i, v_i\}$, check weather the end points $u_i and V_i$ are in the same tree or not. If they are not in the same tree then we join the edge in the E'.

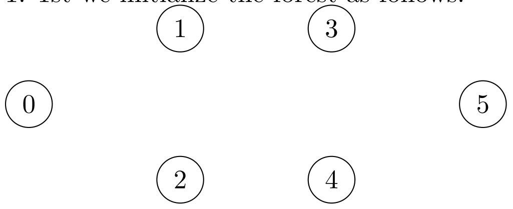
We continue the process untill we get upto $|V|-1$ edges in E'.

The algorithm is a greedy algorithm. Each step choose the smallest edge that does not cause a cycle in the MST is a greedy choice.

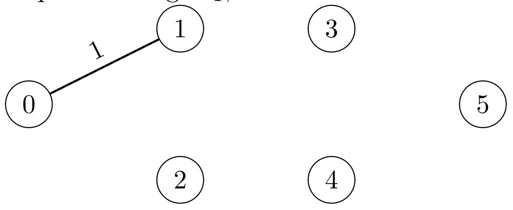
For example take the given graph:



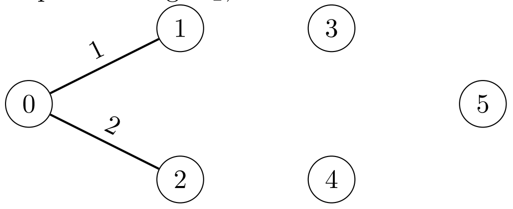
After sorting the weighted we get:
$e_1 = (\{0, 1\}, 1)$
$e_2 = (\{0, 2\}, 2)$
$e_3 = (\{4, 5\}, 5)$
$e_4 = (\{2, 3\}, 7)$
$e_5 = (\{1, 3\}, 8)$
$e_6 = (\{1, 2\}, 10)$
$e_7 = (\{2, 4\}, 15)$
$e_8 = (\{3, 5\}, 20)$
$e_9 = (\{3, 4\}, 30)$
Now pick the edges one by one from sorted list of edges as follows:
1. 1st we initialize the forest as follows:



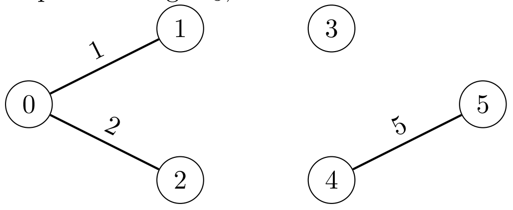
2. pick the edge e_1, Since 0 and 1 does not belong to the same tree so we join these edge.



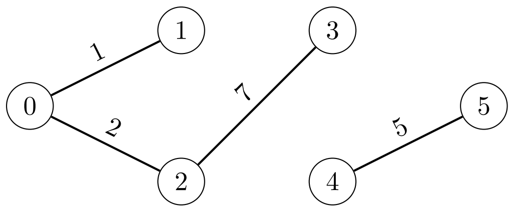
3. pick the edge e_2, Since 0 and 2 does not belong to the same tree so we join these edge.



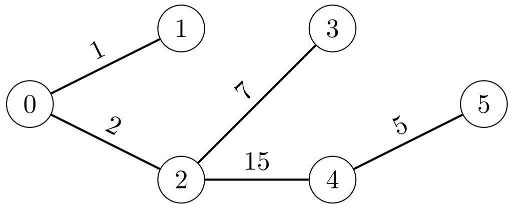
4. pick the edge e_3, Since 4and 5 does not belong to the same tree so we join these edge.



5. pick the edge e_4, Since 2 and 3 does not belong to the same tree so we join these edge.



5. pick the edge e_5, Since 1 and 3 belong to the same tree so we can't join these edge.
6.pick the edge e_6, Since 1 and 2 belong to the same tree so we can't join these edge.
7. pick the edge e_7, Since 2 and 4 does not belong to the same tree so we join these edge.



Since here number of edges is 5 = 6-1 = $|V| - 1$, so we stop it.
These is the minimum spanning tree.

(c) Write down the proof that it indeed provides a minimum spanning tree.
Ans: Here we use the Lemma:
Let S be a set of edges and consider a cut with respect to S and there is an MST containing S. Let (u,v) be a light edge. Then there is a MST containing S∪{(u, v)}.

- Assume that our choices S is so far are safe. i.e., there exits a tree containing S.
- Now we want to add the edge {u,v} to joint the trees T1 and T2, where E(T1), E(T2) ⊂ S.
- Consider the cut with respect to (E(T1), E(V-T1)) and here ({u,v} is a light edge for the cut.

By these above lemma, there is a MST containing S ∪ $\{\{u, v\}\}$. So it does not ruled out the success. So we greedy choices does not ruled out the success. This proves that it indeed provides a minimum spanning tree.

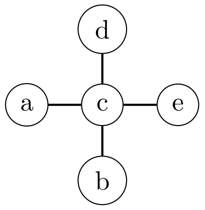
(d) What is the time complexity of your algorithm?
Ans: In the 1st step we sort all the edges in non decreasing order of their weights, Since the number of edges is $|E|$, so it takes $O(|E|log|E|)$ operations.
In the step 4-5, for each vertex v we call Make-Set(v). Each Make-Set(v) takes $O(1)$ time, so for all vertex v it takes $O(v)$ oprations.
For each edge $e_i = \{u, v\}$ we check find-set(u)≠ find-set(v), it takes time $O(log|V|)$.
From 6-11, it takes $O(|E|log|V|)$ operation.
So in this algorithm the total time complexity is given by:
$O(|E|log|E| + O(v) + O(|E|log|V|)$.
=$O(|E|log|V|)$, since $|E| < |V|^2 => log|E| < 2log|V| => O(log|E|) = O(log|V|)$.

9. (a) Explain the 3-SAT and Vertex Cover problems.
Ans:
Vertex Cover Problem:
Let G = (V,E) be a undirected graph with V is the set of vertices and E is the set of edges.
Let S be a subset of V. Then S is called a vertex cover of G if for any e = {u,v}∈ E, either u ∈ S or v ∈ S.
Given an undirected graph G and a positive integer K, the vertex cover problem is to find a vertex cover S of size K or less for G, i.e., a subset S⊂V such that $|S|$ ≤K and, for each (u,v)∈E, at least one of u or v ∈S.
3-SAT Problem:
Let $\phi = C_1 \Lambda C_2 .... \Lambda C_m$, where C_i are clauses and each C_i is in 3-CNF form i.e., C_i has atmost 3 literals and C_i contains literals from the variables x_1, x_2,..., x_n and their complements.

The problem is to check whether ϕ is satisfiable or not.

(b) Considering 3-SAT as NP-Complete, prove that the Vertex Cover problem is also NP-Complete.
Ans: To show that Vertex Cover (VC) problem is NP-Complete problem.
i.e., To show that i. VC$\in$ NP
ii. We can reduce a known NP-complete problem to VC in polynomial time.
i. VC$\in$ NP: Let G = (V,E) be a undirected graph and an integer K .
Given a set of vertices S $\subset$ V, we can check if it is a vertex cover by looking at every edge e$\in$ E and then,checking if the size of the set is at most K. This can be done in polymonial time. Beacause it takes time $O(|S||E|)$, therefore, VC is in NP.
ii.We can reduce a known NP-complete problem to VC in polynomial time:
We have to prove that 3SAT $\leq_p$ VC.
Let $\phi = \Lambda C_i \in$ 3-SAT where i runs from 1 to k.
Let x_1, x_2,..., x_n are the variables.
We now construct a graph G = (V,E). as follows:
V =$\{x_1, \neg x_1, x_2, \neg x_2, ..., x_n, \neg x_n\} \cup \{a_{11}, a_{12}, a_{13}, ..a_{k1}, a_{k2}, a_{k3}\}$.
Let $C_i = \{x_{i1}, x_{i2}, x_{i3}\}$ for i= 1,2,..,k.
Then E = $\cup\{x_i, \neg x_i\} \cup \{\{a_{i1}, a_{i2}\}, \{a_{i2}, a_{i3}\}, \{a_{i3}, a_{i1}\}, \{x_{i1}, a_{i1}\}, \{x_{i2}, a_{i2}\}, \{x_{i3}, a_{i3}\}\}$.
Then$|V|$ = 2n+3k and $|E|$ = n+3k+3k.
The transformation can be done in polynomial time (based on input size n and k)
Here we choose K=n+2k.
if there is a vertex cover set $|S| \leq |K|$ = n+2m
To show that the 3SAT problem has a solution if and only if the VC problem has a solution:
Let ϕ is satisfiable. We construct the vertex cover S as follows:
i. For i=1,2, .., n, if x_i = TRUE, then $x_i \in S$ Else $\neg x_i \in S$.
ii. For i=1,2, .., k, We take any two vertices from $\{a_{i1}, a_{i2}, a_{i3}\}$
Then S is a vertex cover with $|S| = n + 2k$.
Let G has a vertex cover S such that $|S| = n + 2k$,then we show that ϕ is satisfiable.
If S is a vertex cover, then S must contains either x_i or $\neg x_i$ and exactly two vertices from two vertices from $\{a_{j1}, a_{j2}, a_{j3}\}$ for all i=1,2, .., n and For j=1,2, .., k.
Then ϕ contains true assignment $(x_1, x_2, ..., x_n)$ where x_i= T if $x_i \in S$ and x_i = F if $x_i \notin S$.

(c) Consider the following algorithm A towards obtaining a vertex cover. "Pick an arbitrary vertex. Then, throw out all edges covered and repeat. Keep going until there are no uncovered edges left." Given an instance, if there actually exists a vertex cover of size k, what will be the output of algorithm A? Explain.
Ans: If we take star graphs with 5 vertices as follows:



Then there actually exits a vertex cover S = {c} of size 1, but if the arbitary vertex cover is other than c ,then we always get vertex cover whose cardinality is always greater than 1.