

# ALGORITHM DESIGN 1

classmate

Date \_\_\_\_\_

Page \_\_\_\_\_

ASSIGNMENT: 1 → Introduction to AD;  
Heaps; Graph & Related Algorithms; Greedy Method

## SECTION - A

Method of induction:-

① Prove :- for  $n \geq 5$ ,  $4n < 2^n$

**Ans** Proof (induction)

Base case :- For  $n = 5$ , we have  $2^5 = 32 > 20 = 4(5)$

Inductive Hypothesis :-

Suppose that  $2^k > 4k$  for some integer,  $k \geq 5$

Induction step :- We wish to show that  $2^{k+1} > 4(k+1)$

$$2^{k+1} = 2 \cdot 2^k = 2^k + 2^k > 4k + 4k > 4(k+1)$$

We have supposed true for  $k$  and shown true for  $k+1$ . Thus, by induction,  $2^n > 4n$ , if  $n \geq 5$  (proved).

② Prove :- for all  $n > 0$ ,  $n^3 \leq n^2$

**Ans** Let represent the above func,  $f(n) = n^3 \leq n^2$

Base case,  $n=1$ ;  $f(1) = 1^3 \leq 1^2$ ;  
 $1 \leq 1$  (true)

$f(n)$  is true for  $n=1$ .

Inductive Hypothesis for  $n=k$ ; so,  $f(k) = k^3 \leq k^2$  (true)

Now for ( $n=k+1$ )

$$\begin{aligned} f(k+1) &= (k+1)^3 = k^3 + 1 + 3k^2 + 3k \\ &= k^2 + 1 + 2k + 2k^2 + k + k^3 \\ &= (k+1)^2 + 2k^2 + k + k^3 \leq (k+1)^2 \end{aligned}$$

so,  $f(k+1) = (k+1)^3 \leq (k+1)^2$  is true; i.e. inductive step

$n^3 \leq n^2$ , for all  $n > 0$  (thus proved)

Name → Sanjib Kumar Mohanty Sec → CS IT - A

Reg → 2041013276

## COUNTER EXAMPLES

- ① P: a & b are even integers  
 Q: a + b is an even integer  
 $P \rightarrow Q$  (according to statement)  
 However,  $(Q \rightarrow P)$  converse is not necessarily  
 be true if we take a & b as odd integers.  
 Let's see an counter example:-  
 $a = 3$  &  $b = 9$  (a & b are odd)  
 $a + b = 3 + 9 = 12$  (which is even)  
 Hence, converse is not true.

- ② If  $(a+b)^2 = c^2$ , such that  $a+b = c$   
 $P = a, b, c$  (real no.'s)  $\rightarrow$   
 $Q = (a+b)^2 = c^2$ ;  $P \rightarrow Q$ .  
 If  $(a+b)^2 = c^2$ , then a, b & c are real integers  
 such that  $Q \rightarrow P$  (converse of statement)  
 This converse is always true even if we  
 take away real no.'s as counter example,  
 True statement can't be proved false.

- ③  $P = a, b, c$ ;  $Q = a/bc$ ;  $P \rightarrow Q$   
 converse  $\Rightarrow$   ~~$Q \rightarrow P$~~   
 $a/bc \Rightarrow a = 6$ ;  $b = 8$ ;  $c = 3$ ;  $b \times c = 24$   
 $\Rightarrow \frac{24}{a} \times \frac{24}{b} = 4$ . But the converse is  
 not true, because 8 is not divisible by 6.

(4) P :- a & b are rational  
& Q :- ab is rational  
 $P \rightarrow Q \Rightarrow$  A/T statement

Converse is  $Q \rightarrow P$ , i.e. if product is rational,  
then a & b are rational.

However, the converse is not necessarily true if we  
take any two irrational no.s,

Counter Example such that product is rational.

Let  $a = 3/\sqrt{2}$

### PROOF OF CORRECTNESS

(1) Interchange (a, b)

$$\text{Let } a = 10; b = 5$$

$$a = 10 + 5 = 15$$

$$b = 15 - 5 = 10$$

$$a = 15 + 10 = 25$$

$\therefore$  ALGORITHM is NOT CORRECT

(2) To show :-  $a+b < \min(a, b)$   $\Rightarrow$  let  $a=1, b=2$

$$a+b = 1+2=3$$

$$\min(1, 2) = 1$$

$3 < 1$  (not possible)

$\therefore$  ALGORITHM is NOT CORRECT

(3) Loop invariant,  $m \geq 0$

$$a = bx + m$$

(a) Before the 1<sup>st</sup> iteration;

$$m = a, \text{ if } x = 0; a = 0 + m \\ \Rightarrow a = a \quad (\text{TRUE})$$

(b)  $m' = m + b$  ( $L$ -times)       $(K \leq Lm) \quad m > 0$

Let  $x' = a + L$

Let  $L = K ; m = a$   
 $a = K(b + m)$   
 $a = Kb + m - b - b$       ( $K$ -times)  
 $a = m ; a = a ; m'' = m' - b$   
 $c = K + 1 = a'' ; u = (m+1)b + m' - b$   
 $\frac{a}{a} = \frac{m}{a}$       ( $\rightarrow$  true)

(c) Post cond<sup>n</sup> :  $m = a \bmod b$   
 $0 \leq m < b$   
 $a = bn + m$

for  $n-1$  cond<sup>n</sup> ;  $m = a \bmod (b-1)+1$   
 $= a \bmod b$

Hence,

M counts m no. after post cond<sup>n</sup> of loops.

(d) Since, 1<sup>st</sup> element is considered to be smallest element of array.  
 $i_{\min} = 0 \Rightarrow K = 1$ .

(b) Let's assume j iteration is true.

T.P.  $\rightarrow$   $(j+1)$  iteration is true.

$\Rightarrow$  upto  $A[j-1]$  is sorted.

Now, it finds  $A[j]$  which is smaller than  $j-1$ , it means element are sorted in correct place.

$\therefore$  invariant is at 1<sup>st</sup> iteration of inner loop ;  $i_{\min} = 0, K = 1, i < n$ , true if  $[A[i] < A[0]]$ ,  $A[0]$  &  $A[i]$  is smaller.

- ① For  $j^{th}$  iteration,  $K = j+1$  & compression made b/w inserted & sorted array.  
 Now element is placed in context with  $A[1:m:n]$   
 'if  $A[K] < A[K']$  before  $A[1:m:n]$  is sorted.

②  $J = n+1$   
 $\rightarrow A[1] \leq A[i] \leq A[3] \dots A[A:m:n]$   
 is already sorted.

⑤ Input  $m$  &  $n$

Output product of  $m$  &  $n$

$$n = m; y = n; t = 0.$$

Before 1<sup>st</sup> iteration;

$$Z = mn - ny; Z - m - mn = 0.$$

Term  $x = 0$  prod. will be 0.

$$Z = mn - ny$$

$$Z = mn$$

⑥ Let  $y = 2; Z = 3$

$$x = 1$$

$$\therefore Z = 3 \text{ (True)} \rightarrow n = 2, Z = 2$$

$$Z = 2 \text{ (n)} \rightarrow x = 4, Z = 1$$

$$Z = 1 \text{ (n)} \rightarrow x = 8; Z = 0,$$

$$Z = 0 \text{ false } n = 8 = 2^3$$

$\therefore$  Algo. is correct

⑦  $y = 5; Z = 4$

Part 1 ( $Z = 4$ ) while ( $2 > 0$ )

Part 2 ( $Z = 2$ ) while ( $2 > 0$ )

if ( $Z \bmod 2 = 1$ )  $\rightarrow 2 \% 2 = 0 \Rightarrow$  false

$$y = 2 \times 10 \rightarrow y = 20$$

$$Z = 2/2 \Rightarrow Z = 1.$$

Part 3 ( $x=1$ ) while ( $x \geq 0$ )

if ( $x \neq 1, x = 1$ )  $\rightarrow$  1.  $y = 0 \rightarrow$  false.  
 $y = 2 \times 2^0 = 40$   
 $x = \frac{1}{2} = 0.$

Q) ~~Ques~~

$$A = [1, 2, 4] \quad B = [3, 10, 12, 14]$$

for  $i = 1$  to 7

$$1 < 3 \Rightarrow C[1] = 1$$

$i++$

$$2 < 3 \Rightarrow C[2] = 2$$

$i++$

$$4 | < 3 \Rightarrow \text{else} \Rightarrow C[3] = 3 \quad i++$$

$$4 < 10 \rightarrow C[4] = 4 \quad i++$$

now,  $i = 5$ , but  $A[5]$  is out of index

$\therefore$  This program isn't working correctly.

Q)  ~~$(\log n)^2 \times \sqrt{n} + 17n \times \text{mcbgns}$~~   
 ~~$n^2 \log n \times n^2 - 20n \leq n^2 - n^3 + n^4$~~   
 ~~$< 2n - 20n$~~ .

Q)  ~~$2^{\log n} + 2^{\log n} = n^2$~~

~~$2^{\log n} + 2^{\log n} < n^2$~~

~~$\Rightarrow \text{Let } n=2$~~

~~$\Rightarrow 2 = 1$~~

~~$2^{0.002} = 1.0000000000000002 \approx 1.00$~~  P10.

$$9. (\log n)^2 < \sqrt{n} < 17n < m \log n < n^2 \log n < n^2 - 20n < n^2 - n^3 + n^4 < 2^n - 20n$$

10.

$$(a) 2^{\lceil \log n \rceil} + \lfloor \log n \rfloor$$

$$= 2^{\lceil \log n + \log n \rceil} = 2^{\lceil 2 \log n \rceil}$$

$$= 2^{\lceil \log n^2 \rceil} = n^2$$

$$f(n) = n^2 \quad g(n) = n^2$$

$$f(n) = O(g(n))$$

$$n^2 = Cn^2 \quad c = 1 \neq n \geq 0$$

Hence proved.

$$(b) 2^{\lceil \log n \rceil} = O(2^{\lceil \log n \rceil})$$

$$f(n) = 2^{\lceil \log n \rceil} = n \quad g(n) = 2^{\lceil \log n \rceil} = n$$

$$f(n) = g(n)$$

$$n = Cn \quad \text{put } c=1 \neq n \geq 0$$

$$n = n$$

Hence proved.

$$(c) 2^{\lceil \log \log n \rceil} = O(2^{\lceil \log \log n \rceil})$$

$$f(n) = 2^{\lceil \log \log n \rceil} = 2^{\lceil \log \log n \rceil} = (\log n)^2$$

$$g(n) = 2^{\lceil \log \log n \rceil} = 2^{\lceil \log \log n \rceil} = (\log n)^2$$

$$f(n) = g(n)$$

$$(\log n)^2 = C \cdot (\log n)^2 \text{ for } C=1 \neq n \geq 0$$

$$\text{Hence } 2^{\lceil \log \log n \rceil} = O(2^{\lceil \log \log n \rceil})$$

proved.

$$(d) \text{ Given: } f(n) = O(g(n))$$

such that  $n > n_0$  and  $c > 0$

$$f(n) \leq c(g(n))$$

Taking log both sides,

$$\log f(n) \leq \log c + \log g(n) + n > n_0$$

since  $n_0$  and  $c$  are constants there be constant  $d$  such

$$\text{that } d \geq \frac{\log c}{\log(g(n_0))} + 1$$

$$(d-1) \log(g(n_0)) \geq \log c$$

$$d \log(g(n_0)) \geq \log c + \log(g(n_0))$$

$$\log f(n) \leq \log c + \log(g(n)) \leq d \log(g(n))$$

$$\log(f(n)) = O(\log(g(n))) \quad \underline{\text{proved.}}$$

(e) Given  $f(n) = O(g(n))$

$$f(n) \leq c g(n)$$

Multiply 2 on both side

$$2f(n) \leq c 2g(n)$$

$$2f(n) = O(2g(n))$$

proved.

(f) We have to prove  $\log^k n = O(n^{1/k})$  for any positive integer  $k$

put  $k=2$  we get

$$\log_2 n \leq (n)^{1/2}$$

put  $k=e \Rightarrow \log_e n = n^{1/e}$  which is true.

$$\text{put } n = e+1 \Rightarrow \log_{e+1} n \leq n^{1/e+1}$$

Hence for any  $k > 0$

$$\log^k n = O(n^{1/k})$$

11. (a)  $A(n) = A\left(\frac{n}{3} + 5 + \lfloor \log n \rfloor\right) + n \log \log n$

$$a=1 \quad b=3 \quad k=1$$

$$a < b^k$$

$$A(n) = O(n \log \log n)$$

(b)  $B(n) = \min_{0 \leq k \leq n} (B + B(k) + B(n-k))$

$$B(n) + B(n-k) = \frac{k(n+1)(n-k)(n-k+1)}{2}$$

(c)  $D(n) = n/(n-5) D(n-1) + 1$   $\underbrace{\quad}_{2} \underbrace{\quad}_{2} - k(n-k) = O(n^2)$

$$D(n) = n/(n-5) \cdot D(n-1) + 1$$

$$D(n-1) = \frac{n-1}{n-6} \quad D(n-2) + 1$$

$$D(n) = \frac{n}{n-5} \left[ \frac{n-1}{n-6} D(n-2) + 1 \right] + 1$$

$$\frac{n(n-1)(n-2) \dots D(0)}{(n-5)(n-6)(n-7) \dots} = \frac{n!}{(n-5)!}$$

$$= O(n!)$$

(d)  $E(n) = E(\lfloor 3n/4 \rfloor) + 1 / \sqrt{n}$

$$a = 1, b = 4/3, k = -1/2$$

$$b^k = (4/3)^{-1/2} = 0.866, \quad (> 0.866)$$

$$E(n) = O(n^{\log_{4/3} 1})$$

$$= O(n^{\log 1/\log 4/3}) = O(n^0) = O(1)$$

(e)  $F(n) = F(\lfloor \log_2 n \rfloor) + \log n$

$$= F(\log \log n) + \log n + \log \log n$$

$$= F(\log \log \log n) + \log n + \log \log n + \log \log \log n$$

$$= O(\log n)$$

(f)  $G(n) = n + \lceil \sqrt{n} \rceil \cdot G(\lfloor \sqrt{n} \rfloor)$

$$\text{let } G(2) = 1$$

so,

$$G(n) = n + \lceil \sqrt{n} \rceil \times G(\lfloor \sqrt{n} \rfloor)$$

$$= n + \lceil \sqrt{n} \rceil (\sqrt{n} + \lceil \sqrt{n} \rceil G(\lfloor \sqrt{n} \rfloor))$$

$$= n + \sqrt{n} + \lceil \sqrt{n} \rceil^{3/2} \cdot G(\lfloor \sqrt{n} \rfloor)$$

$$= n + \sqrt{n} + \lceil \sqrt{n} \rceil^2 + \lceil \sqrt{n} \rceil^{7/8} \cdot G(\lfloor \sqrt{n} \rfloor)$$

⋮

$$= n + \sqrt{n} + \lceil \sqrt{n} \rceil^2 + \dots + \lceil \sqrt{n} \rceil^{4-1} + \lceil \sqrt{n} \rceil^{\frac{(2^k-1)}{2^k}}$$

$$= n \left( \frac{\lceil \sqrt{n} \rceil^k - 1}{\lceil \sqrt{n} \rceil - 1} \right) + \lceil \sqrt{n} \rceil^k \cdot n^{\frac{2^k-1}{2^k}} \cdot G(\lfloor \sqrt{n} \rfloor)$$

$$= n \left( \frac{\lceil \sqrt{\log n} \rceil^k - 1}{\lceil \sqrt{\log n} \rceil - 1} \right) + \lceil \sqrt{\log n} \rceil^k \cdot n^{\left( \frac{\log n - 1}{\log n} \right)}$$

$$= O(n \log \log n)$$

$$(g) H(n) = \log_2 (H(n-1)) + 1$$

$$H(0) = 1$$

$$H(1) = 1 + H(n-1) + 1$$

$$H(n-1) = H(n-2) + 1$$

$$H(n-2) = H(n-3) + 1$$

!

$$H(n) = H(n-1) + 1$$

$$= H(n-2) + 1 + 1$$

!

$$H(n-k) + k \quad [n-k=0]$$

$$= H(0) + n$$

$$= 1 + n$$

$$H(n) = \mathcal{O}(n)$$

$$(h) I(n) = I(\lfloor n^{1/4} \rfloor) + 1$$

$\Rightarrow$  Base case  $\rightarrow I(4) = 1$

$$I(n) = I(n^{1/4}) + 1$$

$$= I(n^{1/16}) + 1 + 1$$

$$= I(n^{1/64}) + 1 + 1 + 1$$

!

$$= I(n^{1/4^k}) + k$$

$$= I(4) + \frac{1}{4} \log \log k$$

$$= 1 + \frac{1}{4} \log \log k$$

$$I(n) = \mathcal{O}(\log \log n)$$

$$(i) J(n) = J(n - \lfloor n/\log n \rfloor) + 1$$

$$\text{Here, } n \gg \lfloor n/\log n \rfloor$$

$$\therefore n^{\log_b a} = n^{\log 1} = 1$$

$$\therefore f(n) = O(1)$$

By Masters theorem,

$$J(n) = \mathcal{O}(\log n)$$

$$(j) C(n) = 3C(\lceil n/2 \rceil) + n/\log n$$

$$= 3C\left(\frac{n}{2}\right) + n/\log n \quad \left[\frac{n}{2} \gg 5\right]$$

$$\Rightarrow a=3, b=2 \text{ and } f(n) = n/\log n \quad \therefore \frac{n}{2} \gg n^{\frac{1}{b}}$$

$$\text{Now, } n^{\log_2 3} = n^{1.58}$$

So,

$$C(n) = \mathcal{O}(n^{1.58})$$

12. for  $i = 1$  to  $n$  do

    for  $j = 1$  to  $2^i - 1$  do

        output "ALGORITHMS"

(a)  $T(n) = \sum_{i=1}^n \sum_{j=1}^{2^i} 1$

(b)  $T(n) = \sum_{i=1}^n \sum_{j=1}^{2^i} 1 = \sum_{i=1}^n (2^i - j + 1)$

$$= \sum_{i=1}^n (i+1) = \sum_{i=1}^n i + \sum_{i=1}^n 1$$

$$= \frac{n(n+1)}{2} + n$$

13. sum ( $n$ )

$s \leftarrow 0$

for  $i \leftarrow 1$  to  $n$  do

$s \leftarrow s + i$

return  $s$

(a) sum of  $n$  numbers

(b) Basic operation is addition

(c)  $n+1$  times

(d)  $O(n)$

(e)  $O(n)$

$$s \leftarrow 0 \quad s = \frac{n(n+1)}{2} \text{ returned efficiency } O(1).$$

14. function multiply ( $y, z$ )

if  $z = 0$  then

    return 0

else

    if  $z$  is odd then

        return (multiply ( $2y, z/2$ ) +  $y$ )

    else return (multiply ( $2y, z/2$ ))

Ans If  $z = 0$  multiply ( $y, 0$ ) = 0 irrespective of what  $y$  is  
if  $z \neq 0$  multiply ( $y, z$ ) which depend on whether  $z$  is even or odd.

If  $z+1$  is odd then product will be

    multiply ( $(2y, z/2) + y$ )

If  $z+1$  is even, product will be multiply ( $2y, z/2$ ). Hence proved.

### 15. Initialization:

let  $y=1, z=2$

$$n=0 \quad z \bmod 2 = 0+1$$

$$y=2, z=1 \quad n = x+y = 0+2 = 2$$

$$\text{and } 1 \times 2 = 2$$

Hence this initialization is correct in 1<sup>st</sup> iteration of the loop.

### Maintenance:

Let there be  $j$  natural numbers product of  $j$  natural numbers  $= \prod_{i=1}^j i$  which is true for any number  $i$ .

This statement is also true for ' $j+1$ ' natural numbers.

### Termination:

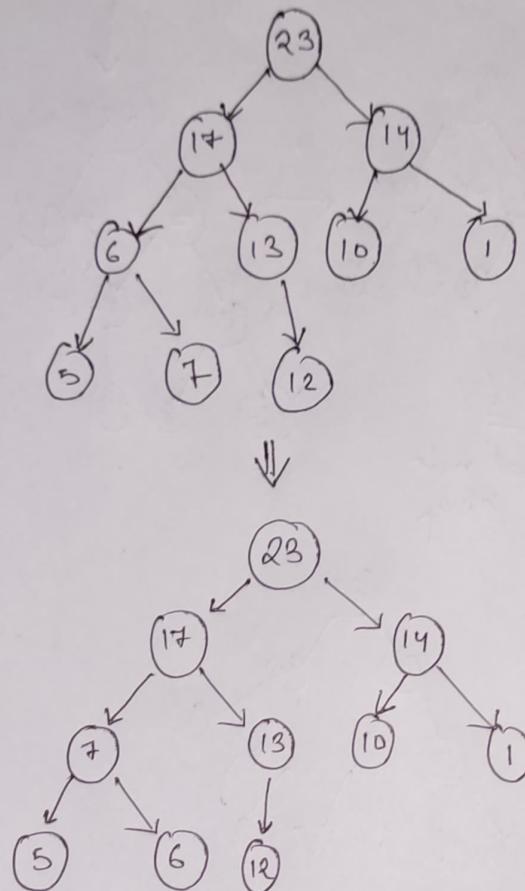
Let  $y=n-1, z=n$  process continues until  $z=1$  irrespective of whether  $n$  is even or odd. This statement is true.

Hence this statement is proved to be correct.

\* HEAPS; GRAPHS AND RELATED ALGORITHMS

1. (a) The array is not a max heap.

The property of max heap failed at  $A[4] = 6 < 80$ ,  
 $A[4]$  is swapped with  $A[9]$  to get max heap.



(b) def isHeap(arr, i, n):

if  $i \geq \text{int}((n-2)/2)$ :

return True

if (arr[i]  $\geq$  arr[2 \* i + 1] and  
 $arr[i] \geq arr[2 * i + 2]$  and  
 $\text{isHeap}(arr, 2 * i + 1, n)$  and  
 $\text{isHeap}(arr, 2 * i + 2, n))$ :

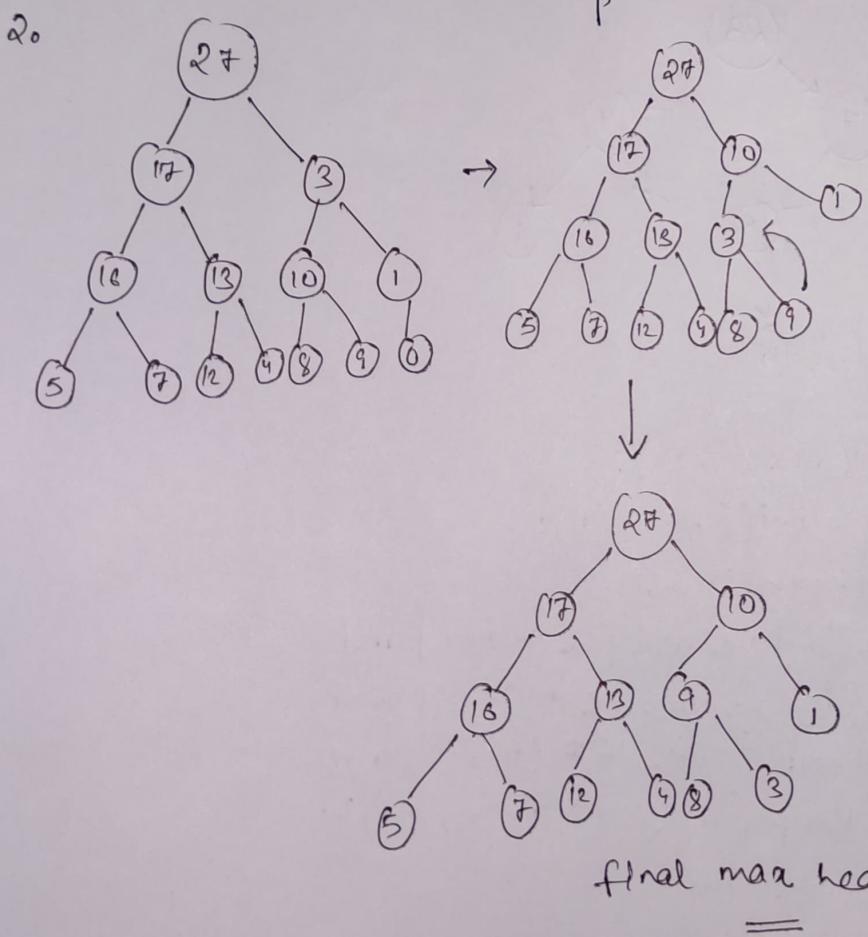
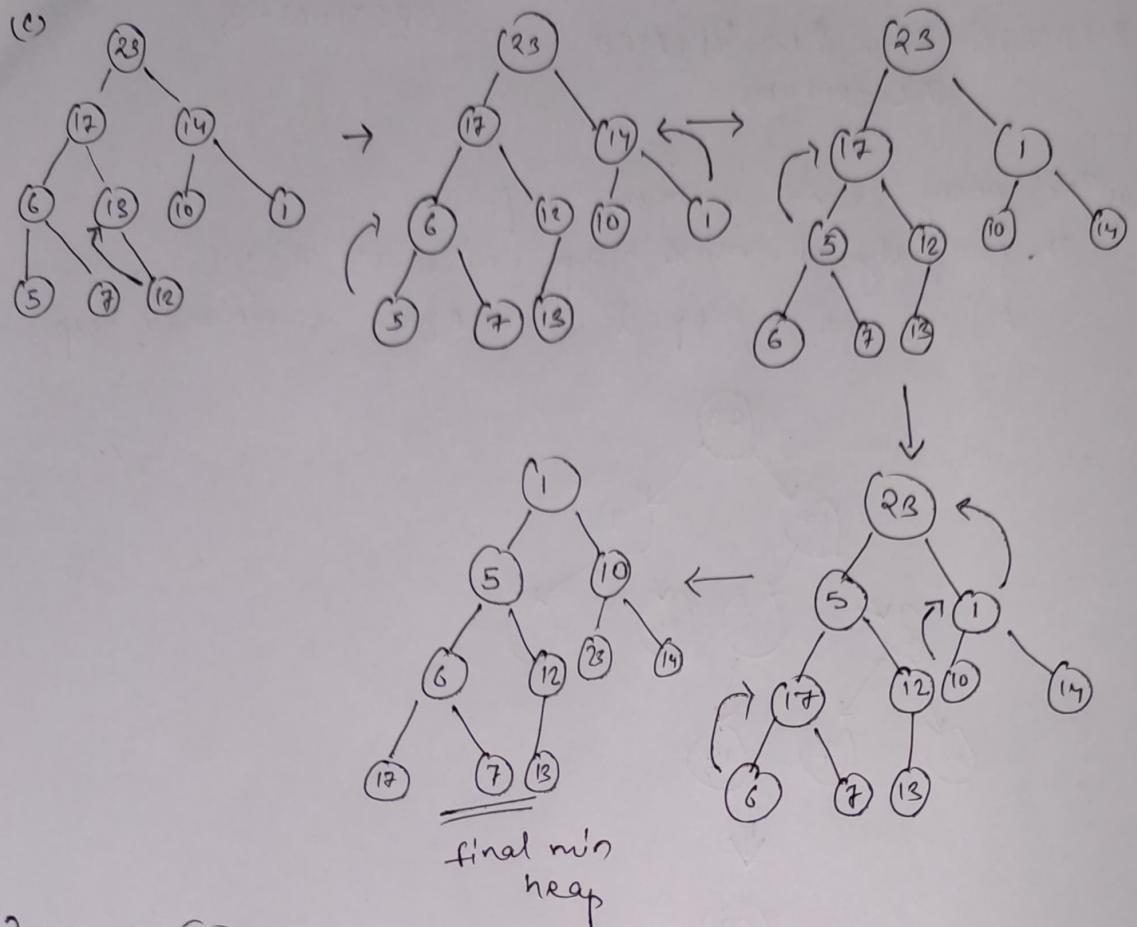
return True

return False

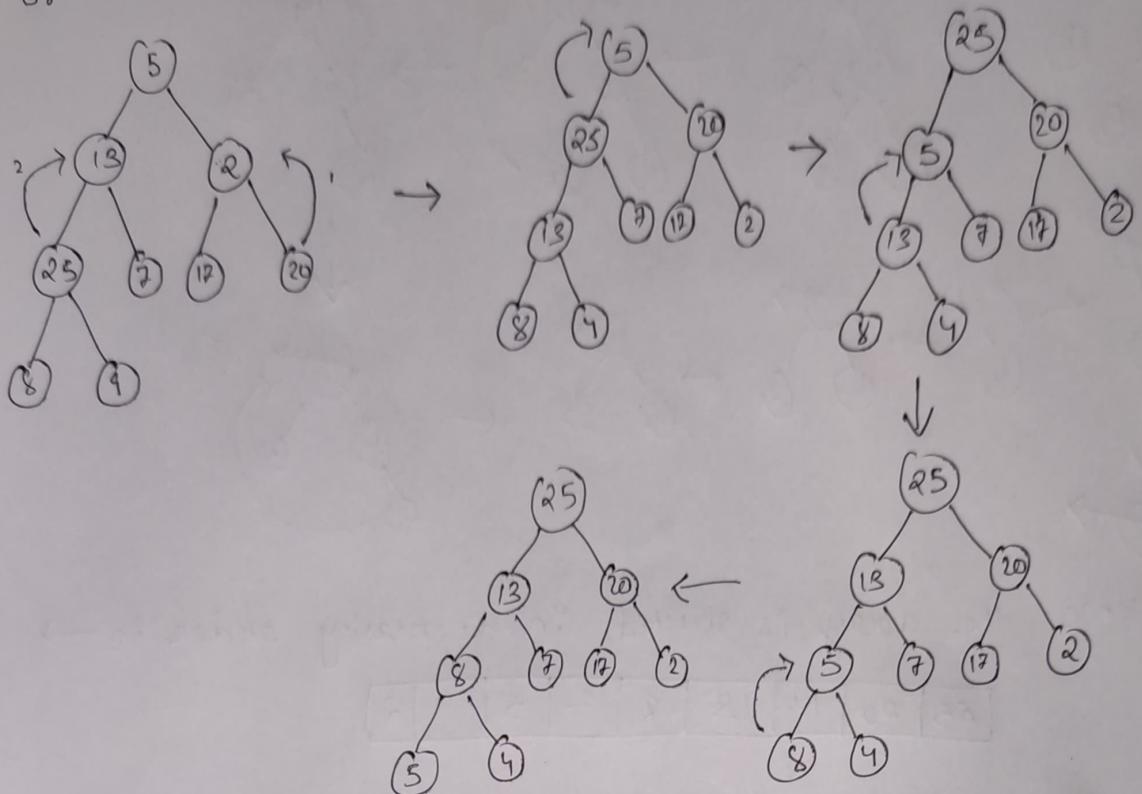
# arr is the array

# i is the starting index

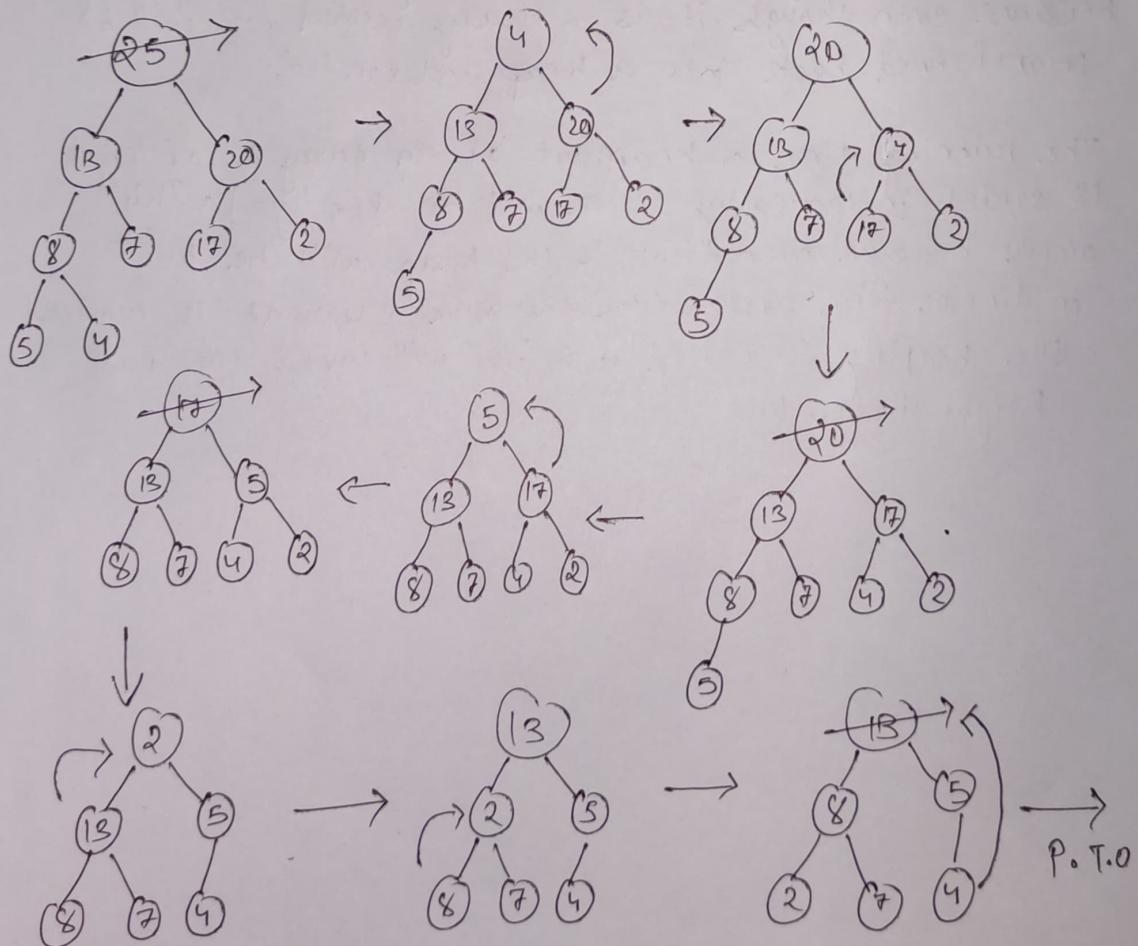
# n is the length of the array.

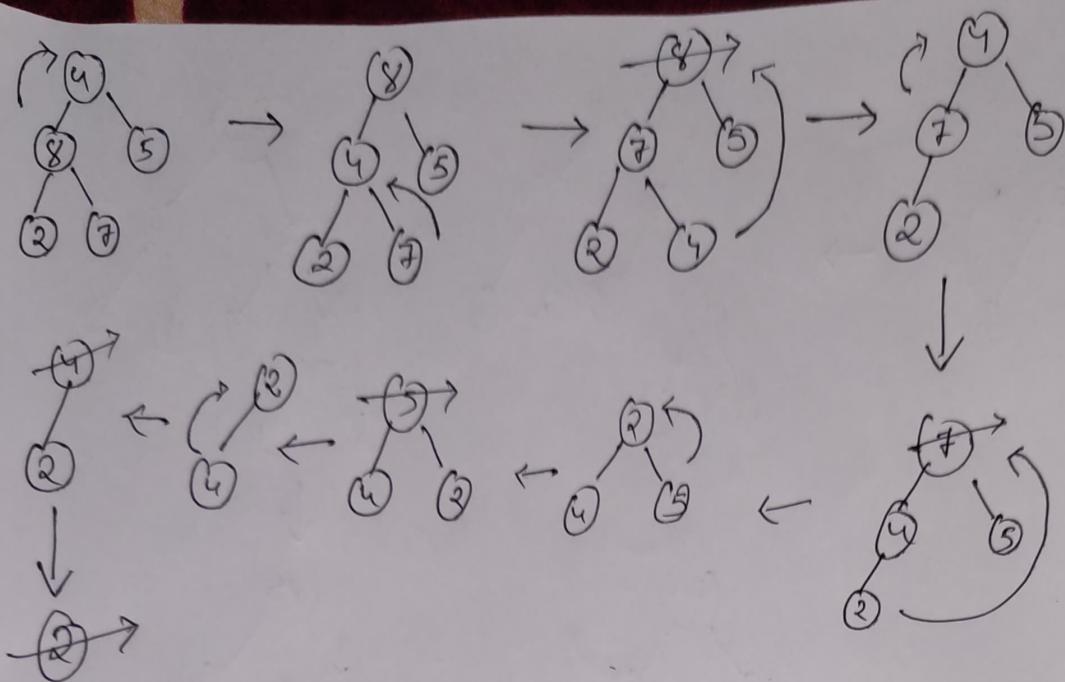


3.



## Performing Heap Sort





Hence the array is sorted in decreasing order  $\rightarrow$

25	20	17	13	8	7	5	4	2
----	----	----	----	---	---	---	---	---

4. The running time of heap sort of an array A of length n that is already sorted in increasing order is  $O(n \log n)$  because even though it is already sorted, it will be transformed back into a heap and sorted.

The running time of heapsort of an array A of length n sorted in decreasing order will be  $O(n \log n)$ . This occurs because even though the heap will be built in linear time every time the max element is removed the heapify is called and it will cover the full height of the tree.

```
BFSTraversal.java x
1 package assignment1;
2 import java.io.*;
3 import java.util.*;
4 public class BFSTraversal
5 {
6     private int node;          /* total number number of nodes in the graph */
7     private LinkedList<Integer> adj[];      /* adjacency list */
8     private Queue<Integer> que;           /* maintaining a queue */
9     BFSTraversal(int v)
10    {
11        node = v;
12        adj = new LinkedList<Integer>[node];
13        for (int i=0; i<v; i++)
14        {
15            adj[i] = new LinkedList<Integer>();
16        }
17        que = new Queue<Integer>();
18    }
19    void insertEdge(int v,int w)
20    {
21        adj[v].add(w);      /* adding an edge to the adjacency list (edges are bidirectional in this example) */
22    }
23    void BFS(int n)
24    {
25        boolean nodes[] = new boolean[node];      /* initialize boolean array for holding the data */
26        int a = 0;
27        nodes[n]=true;
28        que.add(n);      /* root node is added to the top of the queue */
29        while (que.size() != 0)
30        {
31            n = que.poll();      /* remove the top element of the queue */
32            System.out.print(n+" ");
33            for (int i = 0; i < adj[n].size(); i++) /* iterate through the linked list and push all neighbors into queue */
34            {
35                a = adj[n].get(i);
36                if (!nodes[a])      /* only insert nodes into queue if they have not been explored already */
37                {
38                    nodes[a] = true;
39                    que.add(a);
40                }
41            }
42        }
43        public static void main(String args[])
44        {
45            BFSTraversal graph = new BFSTraversal(6);
46            graph.insertEdge(0, 1);
47            graph.insertEdge(0, 3);
48            graph.insertEdge(0, 4);
49            graph.insertEdge(4, 5);
50            graph.insertEdge(3, 5);
51            graph.insertEdge(1, 2);
52            graph.insertEdge(1, 0);
53            graph.insertEdge(2, 1);
54            graph.insertEdge(4, 1);
55            graph.insertEdge(3, 1);
56            graph.insertEdge(5, 4);
57            graph.insertEdge(5, 3);
58            System.out.println("Breadth First Traversal for the graph is:");
59            graph.BFS(0);
60        }
61    }
}
```

```
Problems Javadoc Declaration Search Console x
<terminated> BFSTraversal [Java Application] C:\Users\LENOVO\Down
Breadth First Traversal for the graph is:
0 1 3 4 2 5
```

```
DepthFirstSearch.java x
1 package assignment1;
2 import java.io.*;
3 class DepthFirstSearch {
4     private int V; // No. of vertices
5     private LinkedList<Integer> adj[];
6     // Constructor
7     @SuppressWarnings("unchecked") DepthFirstSearch(int v)
8     {
9         V = v;
10        adj = new LinkedList[v];
11        for (int i = 0; i < v; ++i)
12            adj[i] = new LinkedList();
13    }
14    // Function to add an edge into the graph
15    void addEdge(int v, int w)
16    {
17        adj[v].add(w); // Add w to v's list.
18    }
19
20    // A function used by DFS
21    void DFSUtil(int v, boolean visited[])
22    {
23        // Mark the current node as visited and print it
24        visited[v] = true;
25        System.out.print(v + " ");
26
27        // Recur for all the vertices adjacent to this
28        // vertex
29        Iterator<Integer> i = adj[v].listIterator();
30        while (i.hasNext()) {
31            int n = i.next();
32            if (!visited[n])
33                DFSUtil(n, visited);
34        }
35    }
36    void DFS(int v)
37    {
38        // Mark all the vertices as not visited(set as false by default in java)
39        boolean visited[] = new boolean[V];
40        // Call the recursive helper function to print DFS traversal
41        DFSUtil(v, visited);
42    }
43
44    // Driver Code
45    public static void main(String args[])
46    {
47        DepthFirstSearch g = new DepthFirstSearch(4);
48        g.addEdge(0, 1);
49        g.addEdge(0, 2);
50        g.addEdge(1, 2);
51        g.addEdge(2, 0);
52        g.addEdge(2, 3);
53        g.addEdge(3, 3);
54        System.out.println(
55            "Following is Depth First Traversal "
56            + "(starting from vertex 2)");
57        g.DFS(2);
58    }
59 }
```

Problems Javadoc Declaration Search Console

<terminated> DepthFirstSearch [Java Application] C:\Users\LENOVO\

Following is Depth First Traversal (starting from vertex 2)

2 0 1 3

```
graphjava x
9 class KruskalMSTalgorithm {
10 // A class to represent a graph edge
11 class Edge implements Comparable<Edge>
12 {
13     int src, dest, weight;
14
15     // Comparator function used for
16     // sorting edges based on their weight
17     public int compareTo(Edge compareEdge)
18     {
19         return this.weight - compareEdge.weight;
20     }
21 }
22
23 // A class to represent a subset for
24 // union-find
25 class subset
26 {
27     int parent, rank;
28 }
29
30 int V, E; // V-> no. of vertices & E->no. of edges
31 Edge edge[]; // collection of all edges
32
33 // Creates a graph with V vertices and E edges
34 KruskalMSTalgorithm(int v, int e)
35 {
36     V = v;
37     E = e;
38     edge = new Edge[E];
39     for (int i = 0; i < e; ++i)
40         edge[i] = new Edge();
41 }
42
43 // A utility function to find set of an
44 // element i (uses path compression technique)
45 int find(subset subsets[], int i)
46 {
47     // find root and make root as parent of i
48     // (path compression)
49     if (subsets[i].parent != i)
50         subsets[i].parent
51             = find(subsets, subsets[i].parent);
52
53     return subsets[i].parent;
54 }
55
56 // A function that does union of two sets
57 // of x and y (uses union by rank)
58 void Union(subset subsets[], int x, int y)
59 {
60     int xroot = find(subsets, x);
61     int yroot = find(subsets, y);
62
63     // Attach smaller rank tree under root
64     // of high rank tree (Union by Rank)
65     if (subsets[xroot].rank
66         < subsets[yroot].rank)
67         subsets[xroot].parent = yroot;
68     else if (subsets[xroot].rank
69         > subsets[yroot].rank)
70         subsets[yroot].parent = xroot;
71
72     // If ranks are same, then make one as
73     // root and increment its rank by one
74     else {
75         subsets[yroot].parent = xroot;
76         subsets[xroot].rank++;
77     }
78
79 // The main function to construct MST using Kruskal's
80 // algorithm
81 void KruskalMST()
82 {
83     // This will store the resultant MST
84     Edge result[] = new Edge[V];
85
86     // An index variable, used for result[]
87     int e = 0;
88
89     // An index variable, used for sorted edges
90     int i = 0;
91     for (i = 0; i < V; ++i)
92         result[i] = new Edge();
93
94     // Step 1: Sort all the edges in non-decreasing
95     // order of their weight. If we are not allowed to
96     // change the given graph, we can create a copy of
97     // array of edges
98     Arrays.sort(edge);
99
100    // Allocate memory for creating V subsets
101    subset subsets[] = new subset[V];
102    for (i = 0; i < V; ++i)
103        subsets[i] = new subset();
104
105    // Create V subsets with single elements
106    for (int v = 0; v < V; ++v)
107    {
108        subsets[v].parent = v;
109        subsets[v].rank = 0;
110    }
111
112    i = 0; // Index used to pick next edge
113
114    // Number of edges to be taken is equal to V-1
115    while (e < V - 1)
116    {
117        // Step 2: Pick the smallest edge. And increment
118        // the index for next iteration
119        Edge next_edge = edge[i++];
120
121        int x = find(subsets, next_edge.src);
122        int y = find(subsets, next_edge.dest);
123
124        // If including this edge doesn't cause
125        // a cycle, then include it in result and increment
126        // the index for next iteration
127        if (x != y) {
128            result[e++] = next_edge;
129            Union(subsets, x, y);
130        }
131        // Else discard the next_edge
132    }
133
134
135    // print the contents of result[] to display
136    // the built MST
137    System.out.println("Following are the edges " +
138                      + "the constructed MST");
139
140    int minimumCost = 0;
141    for (i = 0; i < e; ++i)
142    {
143        System.out.println(result[i].src + " -- "
144                            + result[i].dest
145                            + " == " + result[i].weight);
146        minimumCost += result[i].weight;
147    }
148
149    System.out.println("Minimum Cost Spanning Tree "
150                      + minimumCost);
151
152 // Driver Code
153 public static void main(String[] args)
154 {
155     /* Let us create following weighted graph
156      10
157      0-----1
158      | \ |
159      6| 5\|15
160      |   \
161      2-----3
162      4 */
163
164     int V = 4; // Number of vertices in graph
165     int E = 5; // Number of edges in graph
166     KruskalMSTalgorithm graph = new KruskalMSTalgorithm(V, E);
167
168     // add edge 0-1
169     graph.edge[0].src = 0;
170     graph.edge[0].dest = 1;
171     graph.edge[0].weight = 10;
172
173     // add edge 0-2
174     graph.edge[1].src = 0;
175     graph.edge[1].dest = 2;
176     graph.edge[1].weight = 6;
177
178     // add edge 0-3
179     graph.edge[2].src = 0;
180     graph.edge[2].dest = 3;
181
182     // add edge 1-3
183     graph.edge[3].src = 1;
184     graph.edge[3].dest = 3;
185     graph.edge[3].weight = 15;
186
187     // add edge 2-3
188     graph.edge[4].src = 2;
189     graph.edge[4].dest = 3;
190     graph.edge[4].weight = 4;
191
192     // Function call
193     graph.KruskalMST();
194 }
195 }
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564

```

```

Prism_MST.java x
1 package assignment1;
2
3 import java.util.*;
4
5 class Prism_MST {
6     private static final int V = 5;
7
8     int minKey(int key[], Boolean mstSet[])
9     {
10        int min = Integer.MAX_VALUE, min_index = -1;
11
12        for (int v = 0; v < V; v++)
13            if (mstSet[v] == false && key[v] < min) {
14                min = key[v];
15                min_index = v;
16            }
17
18        return min_index;
19    }
20
21    void printMST(int parent[], int graph[][])
22    {
23        System.out.println("Edge \tWeight");
24        for (int i = 1; i < V; i++)
25            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
26    }
27
28    void primMST(int graph[][])
29    {
30        // Array to store constructed MST
31        int parent[] = new int[V];
32
33        // Key values used to pick minimum weight edge in cut
34        int key[] = new int[V];
35
36        // To represent set of vertices included in MST
37        Boolean mstSet[] = new Boolean[V];
38
39        // Initialize all keys as INFINITE
40        for (int i = 0; i < V; i++) {
41            key[i] = Integer.MAX_VALUE;
42            mstSet[i] = false;
43        }
44
45    }

```

```

Prism_MST.java x
42         for (int i = 0, i < v, i++)
43             key[i] = Integer.MAX_VALUE;
44             mstSet[i] = false;
45         }
46
47         key[0] = 0;
48         parent[0] = -1;
49
50         for (int count = 0; count < V - 1; count++) {
51             int u = minKey(key, mstSet);
52
53             mstSet[u] = true;
54
55             for (int v = 0; v < V; v++)
56
57                 if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
58                     parent[v] = u;
59                     key[v] = graph[u][v];
60                 }
61             }
62
63             printMST(parent, graph);
64         }
65
66     public static void main(String[] args)
67     {
68         /* Let us create the following graph
69          2 3
70          (0)--(1)--(2)
71          | / \ |
72          6| 8/ \5 | 7
73          | /       \ |
74          (3)-----(4)
75          9           */
76         Prism_MST t = new Prism_MST();
77         int graph[][] = new int[][] { { 0, 2, 0, 6, 0 },
78                                     { 2, 0, 3, 8, 5 },
79                                     { 0, 3, 0, 0, 7 },
80                                     { 6, 8, 0, 0, 9 },
81                                     { 0, 5, 7, 9, 0 } };
82
83         // Print the solution
84         t.primMST(graph);
85     }
86 }
87

```

Problems Javadoc Declaration Search Console x

<terminated> Prism\_MST [Java Application] C:\Users\LENOVO\Downloads\eclipse\plugins\org.e

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

DijkstraShortestPath.java

```
1 package assignment1;
2 import java.lang.*;
3 import java.io.*;
4
5 class Dijistras_ALG {
6     static final int V = 9;
7     int minDistance(int dist[], Boolean sptSet[])
8     {
9         int min = Integer.MAX_VALUE, min_index = -1;
10
11        for (int v = 0; v < V; v++)
12            if (sptSet[v] == false && dist[v] <= min) {
13                min = dist[v];
14                min_index = v;
15            }
16
17        return min_index;
18    }
19
20    void printSolution(int dist[], int n)
21    {
22        System.out.println("Vertex Distance from Source");
23        for (int i = 0; i < V; i++)
24            System.out.println(i + " \t " + dist[i]);
25    }
26
27    void dijkstra(int graph[][], int src)
28    {
29        int dist[] = new int[V];
30
31        Boolean sptSet[] = new Boolean[V];
32
33        for (int i = 0; i < V; i++) {
34            dist[i] = Integer.MAX_VALUE;
35            sptSet[i] = false;
36        }
37
38        dist[src] = 0;
39
40        for (int count = 0; count < V - 1; count++) {
41            int u = minDistance(dist, sptSet);
42            sptSet[u] = true;
43            for (int v = 0; v < V; v++)
44
45                if (!sptSet[v] && graph[u][v] != 0 &&
46                    dist[u] != Integer.MAX_VALUE && dist[u] + graph[u][v] < dist[v])
47                    dist[v] = dist[u] + graph[u][v];
48        }
49
50        printSolution(dist, V);
51    }
52
53    public static void main(String[] args)
54    {
55        /* Let us create the example graph discussed above */
56        int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
57                                    { 4, 0, 8, 0, 0, 0, 0, 11, 0 },
58                                    { 0, 8, 0, 7, 0, 4, 0, 0, 2 },
59                                    { 0, 0, 7, 0, 9, 14, 0, 0, 0 },
60                                    { 0, 0, 0, 9, 0, 10, 0, 0, 0 },
61                                    { 0, 0, 4, 14, 10, 0, 2, 0, 0 },
62                                    { 0, 0, 0, 0, 2, 0, 0, 1, 6 },
63                                    { 0, 11, 0, 0, 0, 0, 1, 0, 7 },
64                                    { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };
65        Dijistras_ALG t = new Dijistras_ALG();
66        t.dijkstra(graph, 0);
67    }
68 }
```

Problems Javadoc Declaration Search Console

```
<terminated> Prism_MST [Java Application] C:\Users\LENOVO\Down
Edge  Weight
0 - 1  2
1 - 2  3
0 - 3  6
1 - 4  5
```

CheckBipartitenessByBFS.java x

```
1 package assignment1;
2 import java.util.*;
3 import java.lang.*;
4 import java.io.*;
5
6 class Bipartite
7 {
8     final static int V = 4; // No. of Vertices
9
10    // This function returns true if
11    // graph G[V][V] is Bipartite, else false
12    boolean isBipartite(int G[][],int src)
13    {
14        // Create a color array to store
15        // colors assigned to all vertices.
16        // Vertex number is used as index
17        // in this array. The value '-1'
18        // of colorArr[i] is used to indicate
19        // that no color is assigned
20        // to vertex 'i'. The value 1 is
21        // used to indicate first color
22        // is assigned and value 0 indicates
23        // second color is assigned.
24        int colorArr[] = new int[V];
25        for (int i=0; i<V; ++i)
26            colorArr[i] = -1;
27
28        // Assign first color to source
29        colorArr[src] = 1;
30
31        // Create a queue (FIFO) of vertex numbers
32        // and enqueue source vertex for BFS traversal
33        LinkedList<Integer>q = new LinkedList<Integer>();
34        q.add(src);
35
36        // Run while there are vertices in queue (Similar to BFS)
37        while (q.size() != 0)
38        {
39            // Dequeue a vertex from queue
40            int u = q.poll();
41
42            // Return false if there is a self-loop
43            if (G[u][u] == 1)
```

CheckBipartitenessByBFS.java x

```
44                return false;
45
46                // Find all non-colored adjacent vertices
47                for (int v=0; v<V; ++v)
48                {
49                    if (G[u][v]==1 && colorArr[v]==-1)
50                    {
51                        // Assign alternate color to this adjacent v of u
52                        colorArr[v] = 1-colorArr[u];
53                        q.add(v);
54                    }
55                    // An edge from u to v exists and destination
56                    // v is colored with same color as u
57                    else if (G[u][v]==1 && colorArr[v]==colorArr[u])
58                        return false;
59                }
60            }
61            // If we reach here, then all adjacent vertices can
62            // be colored with alternate color
63            return true;
64        }
65        // Driver program to test above function
66        public static void main (String[] args)
67        {
68            int G[][] = {{0, 1, 0, 1},
69                         {1, 0, 1, 0},
70                         {0, 1, 0, 1},
71                         {1, 0, 1, 0}};
72            Bipartite b = new Bipartite();
73            if (b.isBipartite(G, 0))
74                System.out.println("Yes");
75            else
76                System.out.println("No");
77        }
78    }
```

Problems Javadoc Declaration Search Console x

<terminated> Prism\_MST [Java Application] C:\Users\LENOVO\Downloads\eclipse\plugins\org.eclipse.justj.

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

```

1 package assignment1;
2 import java.io.*;
3 import java.util.*;
4
5 // This class represents a directed graph
6 // using adjacency list representation
7 class Graph {
8     // No. of vertices
9     private int V;
10
11    // Adjacency List as ArrayList of ArrayList's
12    private ArrayList<ArrayList<Integer>> adj;
13
14    // Constructor
15    Graph(int v)
16    {
17        V = v;
18        adj = new ArrayList<ArrayList<Integer>>(v);
19        for (int i = 0; i < v; ++i)
20            adj.add(new ArrayList<Integer>());
21    }
22
23    // Function to add an edge into the graph
24    void addEdge(int v, int w) { adj.get(v).add(w); }
25
26    // A recursive function used by topologicalSort
27    void topologicalSortUtil(int v, boolean visited[],
28                             Stack<Integer> stack)
29    {
30        // Mark the current node as visited.
31        visited[v] = true;
32        Integer i;
33        Iterator<Integer> it = adj.get(v).iterator();
34        while (it.hasNext()) {
35            i = it.next();
36            if (!visited[i])
37                topologicalSortUtil(i, visited, stack);
38        }
39        stack.push(new Integer(v));
40    }
41
42    void topologicalSort()
43    {

```

```

30
31        stack.push(new Integer(v));
32    }
33
34    void topologicalSort()
35    {
36        Stack<Integer> stack = new Stack<Integer>();
37
38        // Mark all the vertices as not visited
39        boolean visited[] = new boolean[V];
40        for (int i = 0; i < V; i++)
41            visited[i] = false;
42
43        // Call the recursive helper
44        // function to store
45        // Topological Sort starting
46        // from all vertices one by one
47        for (int i = 0; i < V; i++)
48            if (visited[i] == false)
49                topologicalSortUtil(i, visited, stack);
50
51        // Print contents of stack
52        while (stack.empty() == false)
53            System.out.print(stack.pop() + " ");
54
55        // Driver code
56        public static void main(String args[])
57        {
58            // Create a graph given in the above diagram
59            Graph g = new Graph(6);
60            g.addEdge(5, 2);
61            g.addEdge(5, 0);
62            g.addEdge(4, 0);
63            g.addEdge(4, 1);
64            g.addEdge(2, 3);
65            g.addEdge(3, 1);
66
67            System.out.println("Following is a Topological "
68                               + "sort of the given graph");
69            // Function Call
70            g.topologicalSort();
71        }
72
73    }
74
75    void topologicalSort()
76    {
77        Stack<Integer> stack = new Stack<Integer>();
78
79        // Mark all the vertices as not visited
80        boolean visited[] = new boolean[V];
81        for (int i = 0; i < V; i++)
82            visited[i] = false;
83
84        // Call the recursive helper
85        // function to store
86        // Topological Sort starting
87        // from all vertices one by one
88        for (int i = 0; i < V; i++)
89            if (visited[i] == false)
90                topologicalSortUtil(i, visited, stack);
91
92        // Print contents of stack
93        while (stack.empty() == false)
94            System.out.print(stack.pop() + " ");
95
96        // Driver code
97        public static void main(String args[])
98        {
99            // Create a graph given in the above diagram
100           Graph g = new Graph(6);
101           g.addEdge(5, 2);
102           g.addEdge(5, 0);
103           g.addEdge(4, 0);
104           g.addEdge(4, 1);
105           g.addEdge(2, 3);
106           g.addEdge(3, 1);
107
108           System.out.println("Following is a Topological "
109                              + "sort of the given graph");
110           // Function Call
111           g.topologicalSort();
112       }
113
114   }
115
116   void topologicalSort()
117   {
118       Stack<Integer> stack = new Stack<Integer>();
119
120       // Mark all the vertices as not visited
121       boolean visited[] = new boolean[V];
122       for (int i = 0; i < V; i++)
123           visited[i] = false;
124
125       // Call the recursive helper
126       // function to store
127       // Topological Sort starting
128       // from all vertices one by one
129       for (int i = 0; i < V; i++)
130           if (visited[i] == false)
131               topologicalSortUtil(i, visited, stack);
132
133       // Print contents of stack
134       while (stack.empty() == false)
135           System.out.print(stack.pop() + " ");
136
137       // Driver code
138       public static void main(String args[])
139       {
140           // Create a graph given in the above diagram
141           Graph g = new Graph(6);
142           g.addEdge(5, 2);
143           g.addEdge(5, 0);
144           g.addEdge(4, 0);
145           g.addEdge(4, 1);
146           g.addEdge(2, 3);
147           g.addEdge(3, 1);
148
149           System.out.println("Following is a Topological "
150                          + "sort of the given graph");
151           // Function Call
152           g.topologicalSort();
153       }
154
155   }
156
157   void topologicalSort()
158   {
159       Stack<Integer> stack = new Stack<Integer>();
160
161       // Mark all the vertices as not visited
162       boolean visited[] = new boolean[V];
163       for (int i = 0; i < V; i++)
164           visited[i] = false;
165
166       // Call the recursive helper
167       // function to store
168       // Topological Sort starting
169       // from all vertices one by one
170       for (int i = 0; i < V; i++)
171           if (visited[i] == false)
172               topologicalSortUtil(i, visited, stack);
173
174       // Print contents of stack
175       while (stack.empty() == false)
176           System.out.print(stack.pop() + " ");
177
178       // Driver code
179       public static void main(String args[])
180       {
181           // Create a graph given in the above diagram
182           Graph g = new Graph(6);
183           g.addEdge(5, 2);
184           g.addEdge(5, 0);
185           g.addEdge(4, 0);
186           g.addEdge(4, 1);
187           g.addEdge(2, 3);
188           g.addEdge(3, 1);
189
190           System.out.println("Following is a Topological "
191                          + "sort of the given graph");
192           // Function Call
193           g.topologicalSort();
194       }
195
196   }
197
198   void topologicalSort()
199   {
200       Stack<Integer> stack = new Stack<Integer>();
201
202       // Mark all the vertices as not visited
203       boolean visited[] = new boolean[V];
204       for (int i = 0; i < V; i++)
205           visited[i] = false;
206
207       // Call the recursive helper
208       // function to store
209       // Topological Sort starting
210       // from all vertices one by one
211       for (int i = 0; i < V; i++)
212           if (visited[i] == false)
213               topologicalSortUtil(i, visited, stack);
214
215       // Print contents of stack
216       while (stack.empty() == false)
217           System.out.print(stack.pop() + " ");
218
219       // Driver code
220       public static void main(String args[])
221       {
222           // Create a graph given in the above diagram
223           Graph g = new Graph(6);
224           g.addEdge(5, 2);
225           g.addEdge(5, 0);
226           g.addEdge(4, 0);
227           g.addEdge(4, 1);
228           g.addEdge(2, 3);
229           g.addEdge(3, 1);
230
231           System.out.println("Following is a Topological "
232                          + "sort of the given graph");
233           // Function Call
234           g.topologicalSort();
235       }
236
237   }
238
239   void topologicalSort()
240   {
241       Stack<Integer> stack = new Stack<Integer>();
242
243       // Mark all the vertices as not visited
244       boolean visited[] = new boolean[V];
245       for (int i = 0; i < V; i++)
246           visited[i] = false;
247
248       // Call the recursive helper
249       // function to store
250       // Topological Sort starting
251       // from all vertices one by one
252       for (int i = 0; i < V; i++)
253           if (visited[i] == false)
254               topologicalSortUtil(i, visited, stack);
255
256       // Print contents of stack
257       while (stack.empty() == false)
258           System.out.print(stack.pop() + " ");
259
260       // Driver code
261       public static void main(String args[])
262       {
263           // Create a graph given in the above diagram
264           Graph g = new Graph(6);
265           g.addEdge(5, 2);
266           g.addEdge(5, 0);
267           g.addEdge(4, 0);
268           g.addEdge(4, 1);
269           g.addEdge(2, 3);
270           g.addEdge(3, 1);
271
272           System.out.println("Following is a Topological "
273                          + "sort of the given graph");
274           // Function Call
275           g.topologicalSort();
276       }
277
278   }
279
280   void topologicalSort()
281   {
282       Stack<Integer> stack = new Stack<Integer>();
283
284       // Mark all the vertices as not visited
285       boolean visited[] = new boolean[V];
286       for (int i = 0; i < V; i++)
287           visited[i] = false;
288
289       // Call the recursive helper
290       // function to store
291       // Topological Sort starting
292       // from all vertices one by one
293       for (int i = 0; i < V; i++)
294           if (visited[i] == false)
295               topologicalSortUtil(i, visited, stack);
296
297       // Print contents of stack
298       while (stack.empty() == false)
299           System.out.print(stack.pop() + " ");
300
301       // Driver code
302       public static void main(String args[])
303       {
304           // Create a graph given in the above diagram
305           Graph g = new Graph(6);
306           g.addEdge(5, 2);
307           g.addEdge(5, 0);
308           g.addEdge(4, 0);
309           g.addEdge(4, 1);
310           g.addEdge(2, 3);
311           g.addEdge(3, 1);
312
313           System.out.println("Following is a Topological "
314                          + "sort of the given graph");
315           // Function Call
316           g.topologicalSort();
317       }
318
319   }
320
321   void topologicalSort()
322   {
323       Stack<Integer> stack = new Stack<Integer>();
324
325       // Mark all the vertices as not visited
326       boolean visited[] = new boolean[V];
327       for (int i = 0; i < V; i++)
328           visited[i] = false;
329
330       // Call the recursive helper
331       // function to store
332       // Topological Sort starting
333       // from all vertices one by one
334       for (int i = 0; i < V; i++)
335           if (visited[i] == false)
336               topologicalSortUtil(i, visited, stack);
337
338       // Print contents of stack
339       while (stack.empty() == false)
340           System.out.print(stack.pop() + " ");
341
342       // Driver code
343       public static void main(String args[])
344       {
345           // Create a graph given in the above diagram
346           Graph g = new Graph(6);
347           g.addEdge(5, 2);
348           g.addEdge(5, 0);
349           g.addEdge(4, 0);
350           g.addEdge(4, 1);
351           g.addEdge(2, 3);
352           g.addEdge(3, 1);
353
354           System.out.println("Following is a Topological "
355                          + "sort of the given graph");
356           // Function Call
357           g.topologicalSort();
358       }
359
360   }
361
362   void topologicalSort()
363   {
364       Stack<Integer> stack = new Stack<Integer>();
365
366       // Mark all the vertices as not visited
367       boolean visited[] = new boolean[V];
368       for (int i = 0; i < V; i++)
369           visited[i] = false;
370
371       // Call the recursive helper
372       // function to store
373       // Topological Sort starting
374       // from all vertices one by one
375       for (int i = 0; i < V; i++)
376           if (visited[i] == false)
377               topologicalSortUtil(i, visited, stack);
378
379       // Print contents of stack
380       while (stack.empty() == false)
381           System.out.print(stack.pop() + " ");
382
383       // Driver code
384       public static void main(String args[])
385       {
386           // Create a graph given in the above diagram
387           Graph g = new Graph(6);
388           g.addEdge(5, 2);
389           g.addEdge(5, 0);
390           g.addEdge(4, 0);
391           g.addEdge(4, 1);
392           g.addEdge(2, 3);
393           g.addEdge(3, 1);
394
395           System.out.println("Following is a Topological "
396                          + "sort of the given graph");
397           // Function Call
398           g.topologicalSort();
399       }
400
401   }
402
403   void topologicalSort()
404   {
405       Stack<Integer> stack = new Stack<Integer>();
406
407       // Mark all the vertices as not visited
408       boolean visited[] = new boolean[V];
409       for (int i = 0; i < V; i++)
410           visited[i] = false;
411
412       // Call the recursive helper
413       // function to store
414       // Topological Sort starting
415       // from all vertices one by one
416       for (int i = 0; i < V; i++)
417           if (visited[i] == false)
418               topologicalSortUtil(i, visited, stack);
419
420       // Print contents of stack
421       while (stack.empty() == false)
422           System.out.print(stack.pop() + " ");
423
424       // Driver code
425       public static void main(String args[])
426       {
427           // Create a graph given in the above diagram
428           Graph g = new Graph(6);
429           g.addEdge(5, 2);
430           g.addEdge(5, 0);
431           g.addEdge(4, 0);
432           g.addEdge(4, 1);
433           g.addEdge(2, 3);
434           g.addEdge(3, 1);
435
436           System.out.println("Following is a Topological "
437                          + "sort of the given graph");
438           // Function Call
439           g.topologicalSort();
440       }
441
442   }
443
444   void topologicalSort()
445   {
446       Stack<Integer> stack = new Stack<Integer>();
447
448       // Mark all the vertices as not visited
449       boolean visited[] = new boolean[V];
450       for (int i = 0; i < V; i++)
451           visited[i] = false;
452
453       // Call the recursive helper
454       // function to store
455       // Topological Sort starting
456       // from all vertices one by one
457       for (int i = 0; i < V; i++)
458           if (visited[i] == false)
459               topologicalSortUtil(i, visited, stack);
460
461       // Print contents of stack
462       while (stack.empty() == false)
463           System.out.print(stack.pop() + " ");
464
465       // Driver code
466       public static void main(String args[])
467       {
468           // Create a graph given in the above diagram
469           Graph g = new Graph(6);
470           g.addEdge(5, 2);
471           g.addEdge(5, 0);
472           g.addEdge(4, 0);
473           g.addEdge(4, 1);
474           g.addEdge(2, 3);
475           g.addEdge(3, 1);
476
477           System.out.println("Following is a Topological "
478                          + "sort of the given graph");
479           // Function Call
480           g.topologicalSort();
481       }
482
483   }
484
485   void topologicalSort()
486   {
487       Stack<Integer> stack = new Stack<Integer>();
488
489       // Mark all the vertices as not visited
490       boolean visited[] = new boolean[V];
491       for (int i = 0; i < V; i++)
492           visited[i] = false;
493
494       // Call the recursive helper
495       // function to store
496       // Topological Sort starting
497       // from all vertices one by one
498       for (int i = 0; i < V; i++)
499           if (visited[i] == false)
500               topologicalSortUtil(i, visited, stack);
501
502       // Print contents of stack
503       while (stack.empty() == false)
504           System.out.print(stack.pop() + " ");
505
506       // Driver code
507       public static void main(String args[])
508       {
509           // Create a graph given in the above diagram
510           Graph g = new Graph(6);
511           g.addEdge(5, 2);
512           g.addEdge(5, 0);
513           g.addEdge(4, 0);
514           g.addEdge(4, 1);
515           g.addEdge(2, 3);
516           g.addEdge(3, 1);
517
518           System.out.println("Following is a Topological "
519                          + "sort of the given graph");
520           // Function Call
521           g.topologicalSort();
522       }
523
524   }
525
526   void topologicalSort()
527   {
528       Stack<Integer> stack = new Stack<Integer>();
529
530       // Mark all the vertices as not visited
531       boolean visited[] = new boolean[V];
532       for (int i = 0; i < V; i++)
533           visited[i] = false;
534
535       // Call the recursive helper
536       // function to store
537       // Topological Sort starting
538       // from all vertices one by one
539       for (int i = 0; i < V; i++)
540           if (visited[i] == false)
541               topologicalSortUtil(i, visited, stack);
542
543       // Print contents of stack
544       while (stack.empty() == false)
545           System.out.print(stack.pop() + " ");
546
547       // Driver code
548       public static void main(String args[])
549       {
550           // Create a graph given in the above diagram
551           Graph g = new Graph(6);
552           g.addEdge(5, 2);
553           g.addEdge(5, 0);
554           g.addEdge(4, 0);
555           g.addEdge(4, 1);
556           g.addEdge(2, 3);
557           g.addEdge(3, 1);
558
559           System.out.println("Following is a Topological "
560                          + "sort of the given graph");
561           // Function Call
562           g.topologicalSort();
563       }
564
565   }
566
567   void topologicalSort()
568   {
569       Stack<Integer> stack = new Stack<Integer>();
570
571       // Mark all the vertices as not visited
572       boolean visited[] = new boolean[V];
573       for (int i = 0; i < V; i++)
574           visited[i] = false;
575
576       // Call the recursive helper
577       // function to store
578       // Topological Sort starting
579       // from all vertices one by one
580       for (int i = 0; i < V; i++)
581           if (visited[i] == false)
582               topologicalSortUtil(i, visited, stack);
583
584       // Print contents of stack
585       while (stack.empty() == false)
586           System.out.print(stack.pop() + " ");
587
588       // Driver code
589       public static void main(String args[])
590       {
591           // Create a graph given in the above diagram
592           Graph g = new Graph(6);
593           g.addEdge(5, 2);
594           g.addEdge(5, 0);
595           g.addEdge(4, 0);
596           g.addEdge(4, 1);
597           g.addEdge(2, 3);
598           g.addEdge(3, 1);
599
600           System.out.println("Following is a Topological "
601                          + "sort of the given graph");
602           // Function Call
603           g.topologicalSort();
604       }
605
606   }
607
608   void topologicalSort()
609   {
610       Stack<Integer> stack = new Stack<Integer>();
611
612       // Mark all the vertices as not visited
613       boolean visited[] = new boolean[V];
614       for (int i = 0; i < V; i++)
615           visited[i] = false;
616
617       // Call the recursive helper
618       // function to store
619       // Topological Sort starting
620       // from all vertices one by one
621       for (int i = 0; i < V; i++)
622           if (visited[i] == false)
623               topologicalSortUtil(i, visited, stack);
624
625       // Print contents of stack
626       while (stack.empty() == false)
627           System.out.print(stack.pop() + " ");
628
629       // Driver code
630       public static void main(String args[])
631       {
632           // Create a graph given in the above diagram
633           Graph g = new Graph(6);
634           g.addEdge(5, 2);
635           g.addEdge(5, 0);
636           g.addEdge(4, 0);
637           g.addEdge(4, 1);
638           g.addEdge(2, 3);
639           g.addEdge(3, 1);
640
641           System.out.println("Following is a Topological "
642                          + "sort of the given graph");
643           // Function Call
644           g.topologicalSort();
645       }
646
647   }
648
649   void topologicalSort()
650   {
651       Stack<Integer> stack = new Stack<Integer>();
652
653       // Mark all the vertices as not visited
654       boolean visited[] = new boolean[V];
655       for (int i = 0; i < V; i++)
656           visited[i] = false;
657
658       // Call the recursive helper
659       // function to store
660       // Topological Sort starting
661       // from all vertices one by one
662       for (int i = 0; i < V; i++)
663           if (visited[i] == false)
664               topologicalSortUtil(i, visited, stack);
665
666       // Print contents of stack
667       while (stack.empty() == false)
668           System.out.print(stack.pop() + " ");
669
670       // Driver code
671       public static void main(String args[])
672       {
673           // Create a graph given in the above diagram
674           Graph g = new Graph(6);
675           g.addEdge(5, 2);
676           g.addEdge(5, 0);
677           g.addEdge(4, 0);
678           g.addEdge(4, 1);
679           g.addEdge(2, 3);
680           g.addEdge(3, 1);
681
682           System.out.println("Following is a Topological "
683                          + "sort of the given graph");
684           // Function Call
685           g.topologicalSort();
686       }
687
688   }
689
690   void topologicalSort()
691   {
692       Stack<Integer> stack = new Stack<Integer>();
693
694       // Mark all the vertices as not visited
695       boolean visited[] = new boolean[V];
696       for (int i = 0; i < V; i++)
697           visited[i] = false;
698
699       // Call the recursive helper
700       // function to store
701       // Topological Sort starting
702       // from all vertices one by one
703       for (int i = 0; i < V; i++)
704           if (visited[i] == false)
705               topologicalSortUtil(i, visited, stack);
706
707       // Print contents of stack
708       while (stack.empty() == false)
709           System.out.print(stack.pop() + " ");
710
711       // Driver code
712       public static void main(String args[])
713       {
714           // Create a graph given in the above diagram
715           Graph g = new Graph(6);
716           g.addEdge(5, 2);
717           g.addEdge(5, 0);
718           g.addEdge(4, 0);
719           g.addEdge(4, 1);
720           g.addEdge(2, 3);
721           g.addEdge(3, 1);
722
723           System.out.println("Following is a Topological "
724                          + "sort of the given graph");
725           // Function Call
726           g.topologicalSort();
727       }
728
729   }
730
731   void topologicalSort()
732   {
733       Stack<Integer> stack = new Stack<Integer>();
734
735       // Mark all the vertices as not visited
736       boolean visited[] = new boolean[V];
737       for (int i = 0; i < V; i++)
738           visited[i] = false;
739
740       // Call the recursive helper
741       // function to store
742       // Topological Sort starting
743       // from all vertices one by one
744       for (int i = 0; i < V; i++)
745           if (visited[i] == false)
746               topologicalSortUtil(i, visited, stack);
747
748       // Print contents of stack
749       while (stack.empty() == false)
750           System.out.print(stack.pop() + " ");
751
752       // Driver code
753       public static void main(String args[])
754       {
755           // Create a graph given in the above diagram
756           Graph g = new Graph(6);
757           g.addEdge(5, 2);
758           g.addEdge(5, 0);
759           g.addEdge(4, 0);
760           g.addEdge(4, 1);
761           g.addEdge(2, 3);
762           g.addEdge(3, 1);
763
764           System.out.println("Following is a Topological "
765                          + "sort of the given graph");
766           // Function Call
767           g.topologicalSort();
768       }
769
770   }
771
772   void topologicalSort()
773   {
774       Stack<Integer> stack = new Stack<Integer>();
775
776       // Mark all the vertices as not visited
777       boolean visited[] = new boolean[V];
778       for (int i = 0; i < V; i++)
779           visited[i] = false;
780
781       // Call the recursive helper
782       // function to store
783       // Topological Sort starting
784       // from all vertices one by one
785       for (int i = 0; i < V; i++)
786           if (visited[i] == false)
787               topologicalSortUtil(i, visited, stack);
788
789       // Print contents of stack
790       while (stack.empty() == false)
791           System.out.print(stack.pop() + " ");
792
793       // Driver code
794       public static void main(String args[])
795       {
796           // Create a graph given in the above diagram
797           Graph g = new Graph(6);
798           g.addEdge(5, 2);
799           g.addEdge(5, 0);
800           g.addEdge(4, 0);
801           g.addEdge(4, 1);
802           g.addEdge(2, 3);
803           g.addEdge(3, 1);
804
805           System.out.println("Following is a Topological "
806                          + "sort of the given graph");
807           // Function Call
808           g.topologicalSort();
809       }
810
811   }
812
813   void topologicalSort()
814   {
815       Stack<Integer> stack = new Stack<Integer>();
816
817       // Mark all the vertices as not visited
818       boolean visited[] = new boolean[V];
819       for (int i = 0; i < V; i++)
820           visited[i] = false;
821
822       // Call the recursive helper
823       // function to store
824       // Topological Sort starting
825       // from all vertices one by one
826       for (int i = 0; i < V; i++)
827           if (visited[i] == false)
828               topologicalSortUtil(i, visited, stack);
829
830       // Print contents of stack
831       while (stack.empty() == false)
832           System.out.print(stack.pop() + " ");
833
834       // Driver code
835       public static void main(String args[])
836       {
837           // Create a graph given in the above diagram
838           Graph g = new Graph(6);
839           g.addEdge(5, 2);
840           g.addEdge(5, 0);
841           g.addEdge(4, 0);
842           g.addEdge(4, 1);
843           g.addEdge(2, 3);
844           g.addEdge(3, 1);
845
846           System.out.println("Following is a Topological "
847                          + "sort of the given graph");
848           // Function Call
849           g.topologicalSort();
850       }
851
852   }
853
854   void topologicalSort()
855   {
856       Stack<Integer> stack = new Stack<Integer>();
857
858       // Mark all the vertices as not visited
859       boolean visited[] = new boolean[V];
860       for (int i = 0; i < V; i++)
861           visited[i] = false;
862
863       // Call the recursive helper
864       // function to store
865       // Topological Sort starting
866       // from all vertices one by one
867       for (int i = 0; i < V; i++)
868           if (visited[i] == false)
869               topologicalSortUtil(i, visited, stack);
870
871       // Print contents of stack
872       while (stack.empty() == false)
873           System.out.print(stack.pop() + " ");
874
875       // Driver code
876       public static void main(String args[])
877       {
878           // Create a graph given in the above diagram
879           Graph g = new Graph(6);
880           g.addEdge(5, 2);
881           g.addEdge(5, 0);
882           g.addEdge(4, 0);
883           g.addEdge(4, 1);
884           g.addEdge(2, 3);
885           g.addEdge(3, 1);
886
887           System.out.println("Following is a Topological "
888                          + "sort of the given graph");
889           // Function Call
890           g.topologicalSort();
891       }
892
893   }
894
895   void topologicalSort()
896   {
897       Stack<Integer> stack = new Stack<Integer>();
898
899       // Mark all the vertices as not visited
900       boolean visited[] = new boolean[V];
901       for (int i = 0; i < V; i++)
902           visited[i] = false;
903
904       // Call the recursive helper
905       // function to store
906       // Topological Sort starting
907       // from all vertices one by one
908       for (int i = 0; i < V; i++)
909           if (visited[i] == false)
910               topologicalSortUtil(i, visited, stack);
911
912       // Print contents of stack
913       while (stack.empty() == false)
914           System.out.print(stack.pop() + " ");
915
916       // Driver code
917       public static void main(String args[])
918       {
919           // Create a graph given in the above diagram
920           Graph g = new Graph(6);
921           g.addEdge(5, 2);
922           g.addEdge(5, 0);
923           g.addEdge(4, 0);
924           g.addEdge(4, 1);
925           g.addEdge(2, 3);
926           g.addEdge(3, 1);
927
928           System.out.println("Following is a Topological "
929                          + "sort of the given graph");
930           // Function Call
931           g.topologicalSort();
932       }
933
934   }
935
936   void topologicalSort()
937   {
938       Stack<Integer> stack = new Stack<Integer>();
939
940       // Mark all the vertices as not visited
941       boolean visited[] = new boolean[V];
942       for (int i = 0; i < V; i++)
943           visited[i] = false;
944
945       // Call the recursive helper
946       // function to store
947       // Topological Sort starting
948       // from all vertices one by one
949       for (int i = 0; i < V; i++)
950           if (visited[i] == false)
951               topologicalSortUtil(i, visited, stack);
952
953       // Print contents of stack
954       while (stack.empty() == false)
955           System.out.print(stack.pop() + " ");
956
957       // Driver code
958       public static void main(String args[])
959       {
960           // Create a graph given in the above diagram
961           Graph g = new Graph(6);
962           g.addEdge(5, 2);
963           g.addEdge(5, 0);
964           g.addEdge(4, 0);
965           g.addEdge(4, 1);
966           g.addEdge(2, 3);
967           g.addEdge(3, 1);
968
969           System.out.println("Following is a Topological "
970                          + "sort of the given graph");
971           // Function Call
972           g.topologicalSort();
973       }
974
975   }
976
977   void topologicalSort()
978   {
979       Stack<Integer> stack = new Stack<Integer>();
980
981       // Mark all the vertices as not visited
982       boolean visited[] = new boolean[V];
983       for (int i = 0; i < V; i++)
984           visited[i] = false;
985
986       // Call the recursive helper
987       // function to store
988       // Topological Sort starting
989       // from all vertices one by one
990       for (int i = 0; i < V; i++)
991           if (visited[i] == false)
992               topologicalSortUtil(i, visited, stack);
993
994       // Print contents of stack
995       while (stack.empty() == false)
996           System.out.print(stack.pop() + " ");
997
998       // Driver code
999       public static void main(String args[])
1000      {
1001          // Create a graph given in the above diagram
1002          Graph g = new Graph(6);
1003          g.addEdge(5, 2);
1004          g.addEdge(5, 0);
1005          g.addEdge(4, 0);
1006          g.addEdge(4, 1);
1007          g.addEdge(2, 3);
1008          g.addEdge(3, 1);
1009
1010          System.out.println("Following is a Topological "
1011                         + "sort of the given graph");
1012          // Function Call
1013          g.topologicalSort();
1014      }
1015
1016  }

```

Edge	Weight
0 - 1	2
1 - 2	3
0 - 3	6
1 - 4	5

ConnectedComponents.java

```
1 package assignment1;
2 // Java program for the above approach
3 import java.util.*;
4 class ConnectedComponents{
5
6     // Stores the parent of each vertex
7     static int []parent = new int[1000000];
8
9     // Function to find the topmost
10    // parent of vertex a
11    static int root(int a)
12    {
13
14        // If current vertex is
15        // the topmost vertex
16        if (a == parent[a])
17        {
18            return a;
19        }
20
21        // Otherwise, set topmost vertex of
22        // its parent as its topmost vertex
23        return parent[a] = root(parent[a]);
24    }
25
26    // Function to connect the component
27    // having vertex a with the component
28    // having vertex b
29    static void connect(int a, int b)
30    {
31
32        // Connect edges
33        a = root(a);
34        b = root(b);
35
36        if (a != b) {
37            parent[b] = a;
38        }
39    }
40
41    // Function to find unique top most parents
42    static void connectedComponents(int n)
43    {
```

ConnectedComponents.java

```
44        // Function to find unique top most parents
45        static void connectedComponents(int n)
46        {
47            HashSet<Integer> s = new HashSet<Integer>();
48
49            // Traverse all vertices
50            for (int i = 0; i < n; i++)
51            {
52                s.add(parent[i]);
53            }
54            System.out.println(s.size());
55
56            static void printAnswer(int N,int [][] edges)
57            {
58                for (int i = 0; i <= N; i++)
59                {
60                    parent[i] = i;
61                }
62                for (int i = 0; i < edges.length; i++)
63                {
64                    connect(edges[i][0], edges[i][1]);
65                }
66            }
67
68            // Driver Code
69            public static void main(String[] args)
70            {
71
72                // Given N
73                int N = 8;
74
75                // Given edges
76                int [][]edges = {{ 1, 0 }, { 0, 2 },
77                                { 5, 3 }, { 3, 4 },
78                                { 6, 7 }};
79
80                // Function call
81                printAnswer(N, edges);
82            }
83        }
```

Problems Javadoc Declaration Search Console

<terminated> ConnectedComponents [Java Application] C:\Users\LEN

3

## \* GREEDY METHOD

## SECTION - C

1. No. of objects =  $n$

max capacity of knapsack =  $m$

weight of object  $i$  =  $w_i$

fraction value:  $x_i$ , where  $0 \leq x_i \leq 1$

Profit earned after placing  $x_i$  =  $p_i x_i$

Goal: To maximize profit subject to capacity constraint.

(a) Maximize

$$\sum_{i=1}^n p_i x_i$$

Subject to

$$\sum_{i=1}^n w_i x_i \leq m$$

let

$$n=3, m=20$$

$$(p_1, p_2, p_3) = (25, 4, 15)$$

$$(w_1, w_2, w_3) = (17, 15, 10)$$

if

$x_1$	$x_2$	$x_3$	$\sum_{i=1}^n w_i x_i$	$\sum_{i=1}^n p_i x_i$
$1/2$	$1/3$	$1/4$	16.5	24.85
1	$2/15$	0	20	28.02
0	1	$1/2$	20	31.05

Strategy - 1

maximize objective function

Put the object with the highest profit in the knapsack.  
Then use a function of the last to fill the sack.

$$\sum_{i=1}^n p_i x_i = 25 \times 1 + 24 \times 2/15 + 0 = 28.05$$

$$\sum_{i=1}^n w_i x_i = 18 \times 1 + 15 \times 2/15 + 0 = 20$$

strategy doesn't yield an optimal, so as the capacity used quickly exhausted which constrained the profit attained.

### Strategy-2

maximize capacity

Choose objects according to least weight.

The approach is that the more no. of objects in knapsack, more is the profit.

$$\sum_{i=1}^2 p_i x_i = 0 + 24 \times 2/3 + 15 \times 1 = 31$$

$$\sum_{i=1}^2 w_i x_i = 0 + 15 \times 2/3 + 10 \times 1 = 20$$

∴ Solution is not optimal as rate of % change in profit not high enough.

### Strategy-3

maximizing profit per weight/unit ( $p_i/w_i$ )

$$\sum_{i=1}^2 p_i x_i = 0 + 24 \times 1 + 15 \times 1/2 = 31.5$$

$$\sum_{i=1}^2 w_i x_i = 0 + 15 \times 1 + 10 \times 1/2 = 20$$

∴ Solution is optimal.

### (b) Fractional Knapsack Greedy Algorithm

Algorithm Knapsack ( $p, w, x$ : arrays;  $m, n$ : int) {

// Input: the objects are in increasing order

// so that  $p[i]/w[i] \geq p[i+1]/w[i+1]$ ;

// Output: optimal solution vector  $x$

for ( $i=0$ ;  $i < n$ ;  $i++$ )

$x[i] = 0$ ;

    capacity =  $m$ ;

    for ( $i=1$ ;  $i < n$ ;  $i++$ ) {

        if  $w[i] > capacity$  then exit() // cause overflow.

        else

$x[i] = 1$ ; // greedy choice.

            capacity = capacity -  $w[i]$ ;

        if  $i < n$  then  $x[i] = capacity / w[i]$ ; // choose a fraction.

}

$$(c) n = 7$$

$$m = 15$$

$$(P_1 \ P_2 \ P_3 \ P_4 \ P_5 \ P_6 \ P_7) = (10, 5, 15, 7, 6, 18, 3)$$

$$(w_1 \ w_2 \ w_3 \ w_4 \ w_5 \ w_6 \ w_7) = (2, 3, 3, 5, 7, 14, 1)$$

$$\frac{P[i]}{w[i]} = [5, 1.3, 3, 1, 6, 4.5, 3]$$

$$x_i = (\underbrace{1}_{\text{1-2/3}} \ \underbrace{1}_{\text{0}} \ \underbrace{1}_{\text{0}} \ \underbrace{1}_{\text{0}} \ \underbrace{1}_{\text{0}})$$

$$\sum x_i w_i = 1 \times 2 + \frac{2}{3} \times 3 + 1 \times 5 + 0 \times 7 + 1 \times 1 + 1 \times 4 + 1 \times 1 \\ = 15$$

$$\sum x_i p_i = 1 \times 10 + 2 \times 5 + 1 \times 15 + 0 \times 7 + 1 \times 6 + 1 \times 18 + 1 \times 3 \\ = 54.6$$

2. Algorithm Cover ( $V, E$ )

```

 $\{$ 
   $V := \emptyset$ 
  repeat
     $\}$ 
  
```

$q$  be a vertex from  $V$  of maximum degree;

Add  $q$  to  $V$ ; Eliminate  $q$  from  $V$ .

$E := E - \{(x, y) \text{ such that } x = q \text{ or } y = q\}$ ;

until ( $E = \emptyset$ ); //  $V$  is the node cover

}

Ans

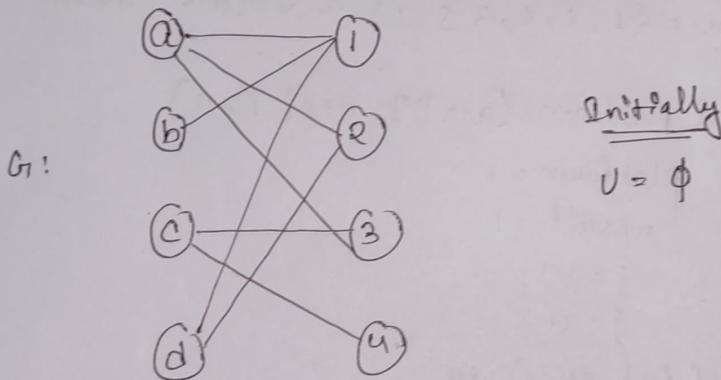
The algorithm considers an empty set  $V$  and the algorithm operates on the graph until edge set is  $\emptyset$ .

The algorithm takes the vertex with highest degree and adds into set  $V$  and eliminates the same from  $V$ .

By this greedy technique at each step the algorithm is trying to remove as much as edges as possible from set  $E$ .

At the end no edges are left and set  $V$  is ensured to be the edge cover.

Eg.



Initially

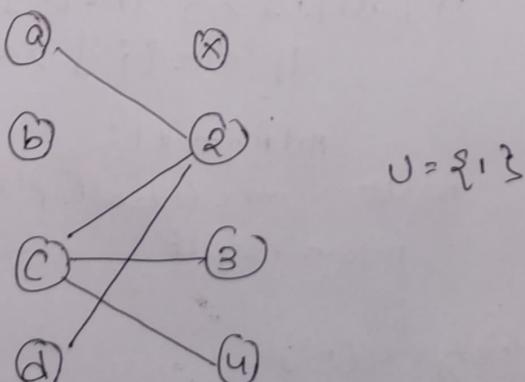
$$V = \emptyset$$

Step 1

$$V = \{\emptyset\}$$

delete  $\sim(\{1, 3\})$ ,  $E(\{a, 1\}, \{b, 1\}, \{c, 1\})$

$G_1:$



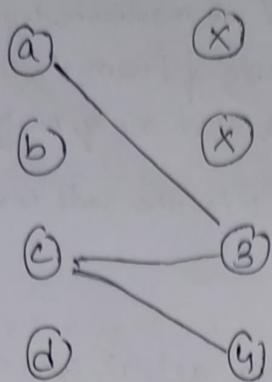
$$V = \{1\}$$

## Step - 2

$$U = \{1, 2\}$$

delete [v{2}, B({2,a3, {2,c3}, {2,d3}})]

9



### Step-3

$$U = \{1, 2\}^*$$

delete ⓒ

## Step-4

$$U = \{1, 2, c, 3\}$$

so,  $u = \{1, 2, 4, 3\}$  is a vertex cover.

3.

(a) function find\_platform (arr[], dep[], n)

platform = 1

result = 1

i = 1

$$j = 0$$

```
for ( i=1 to n ) do
```

platform = 1

(b)

(b)  $f(i) = i+1 + n$   
 checking for overlap } if  $\text{arr}[i] \geq \text{arr}[i] \text{ & } \text{arr}[i] \leq \text{dep}[j]$   
 $\text{if } \text{arr}[j] \geq \text{arr}[i] \text{ & } \text{arr}[j] \geq \text{dep}[i]$

result = max(result, platform);

Return result.

(c) Time complexity =  $O(n^2)$ .