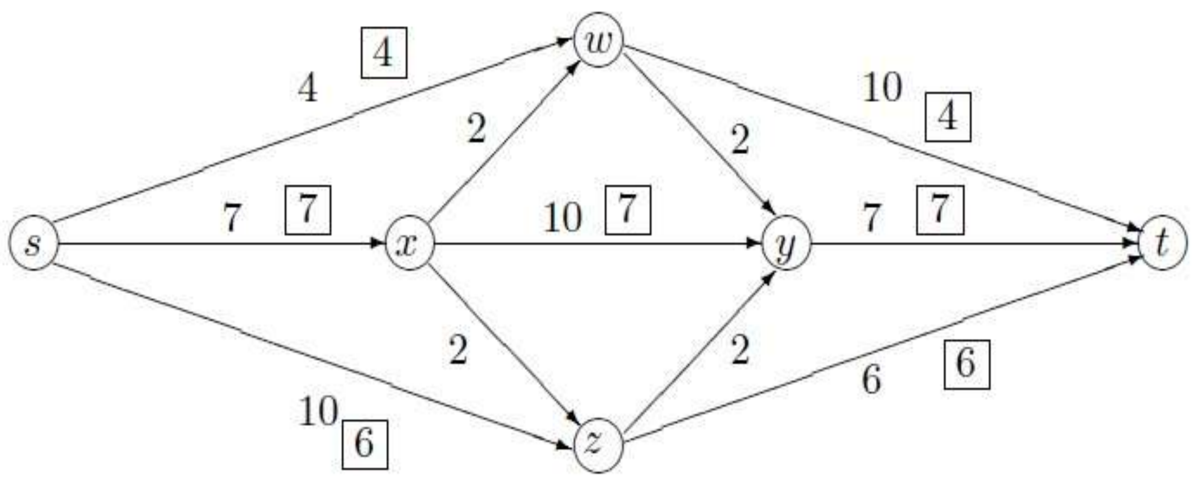
ASSIGNMENT 2:

SUBMITTED BY : ADRITA DUTTA (axd172930)



The figure shows a flow network on which an s-t flow is shown.

The capacity of each edge appears as a label next to the edge, and the numbers in boxes give the amount of flow sent on each edge.

(Edges without boxed numbers have no flow being sent on them.)

(a) What is the value of this flow?

=> The value of the flow is swt(4) + sxyt(7) + szt(6) = 17

(b) Is this a maximum s-t flow in this graph? If not, find a maximum s-t flow.

=> This is not the maximum value,

The maximum flow is :

swt(4) + sxyt(7) + szt(6) + szyxwt(2) =19 (this uses backward flow, residual flow)

(c)Find a minimum s-t cut. (Specify which vertices belong to the sets of the cut.)

=> The minimum cut in this case has sets : {s,z} and {x,w,y,t}

Capacity of this cut is 19, it contains the edges : sw , sx, zy, zt

Answer the above question using two methods.

1. formulate it as an LP problem.   Then use lp-solve (free tool on the internet) to solve th LP problem.

⇨ GIVEN VALUES:

```
/* Objective function */
max: TS ;

/* Variable bounds */
0 <= sw <= 4 ;
0 <= sx <= 7 ;
0 <= sz <= 6 ;
0 <= wt <= 4;
0 <= wy ;
0 <= xw ;
0 <= xy <= 7 ;
0 <= xz ;
0 <= zy ;
0 <= zt <= 6 ;
0 <= yt <= 7 ;
0 <= TS ;

/*node variables */
TS = sw + sx + sz ;
sw = wt ;
sx = xy ;
xy = yt ;
sz = zt ;
wt + yt + zt = TS ;
```

LP formulation:

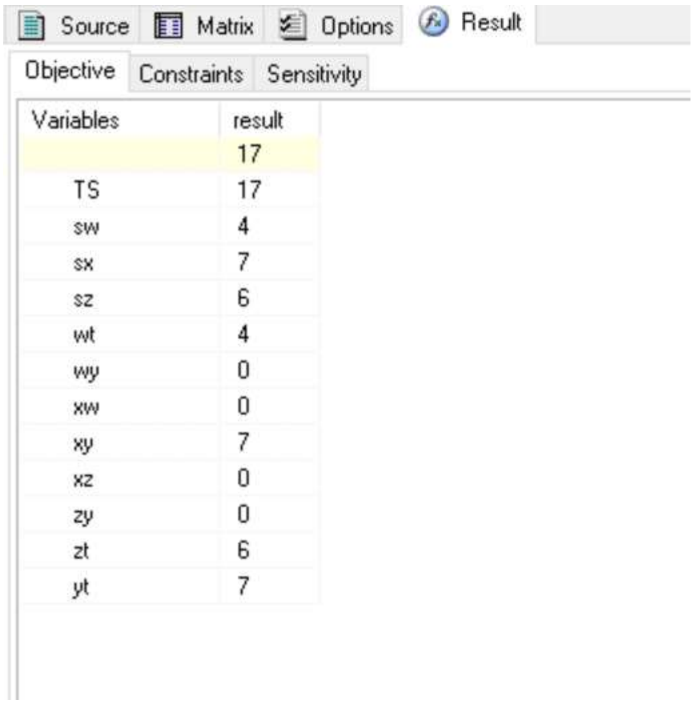
Source | Matrix | Options | Result

```
/* Objective function */
max: TS ;

/* Variable bounds */
0 <= sw <= 4 ;
0 <= sx <= 7 ;
0 <= sz <= 6 ;
0 <= wt <= 4;
0 <= wy ;
0 <= xw ;
0 <= xy <= 7 ;
0 <= xz ;
0 <= zy ;
0 <= zt <= 6 ;
0 <= yt <= 7 ;
0 <= TS ;

/*node variables */
TS = sw + sx + sz ;
sw = wt ;
sx = xy ;
xy = yt ;
sz = zt ;
wt + yt + zt = TS ;
```

Results:



Variables	result
	17
TS	17
sw	4
sx	7
sz	6
wt	4
wy	0
xw	0
xy	7
xz	0
zy	0
zt	6
yt	7

⇨ MAX VALUE:

```
/* Objective function */
max: TS ;

/* Variable bounds */
0 <= sw <= 4 ;
0 <= sx <= 7 ;
0 <= sz <= 14 ;
0 <= wy <= 2 ;
0 <= wt <= 10;
0 <= xw <= 2 ;
0 <= xy <= 10 ;
0 <= xz <= 2 ;
0 <= yt <= 7 ;
0 <= zy <= 2 ;
0 <= zt <= 6 ;
0 <= TS ;

/*node variables */
TS = sw + sx + sz ;
```

sw + xw = wt + wy ;
sx = xw + xz + xy ;
wy + zy + xy = yt ;
sz + xz = zy + zt ;
wt + yt + zt = TS ;

LP formulation:

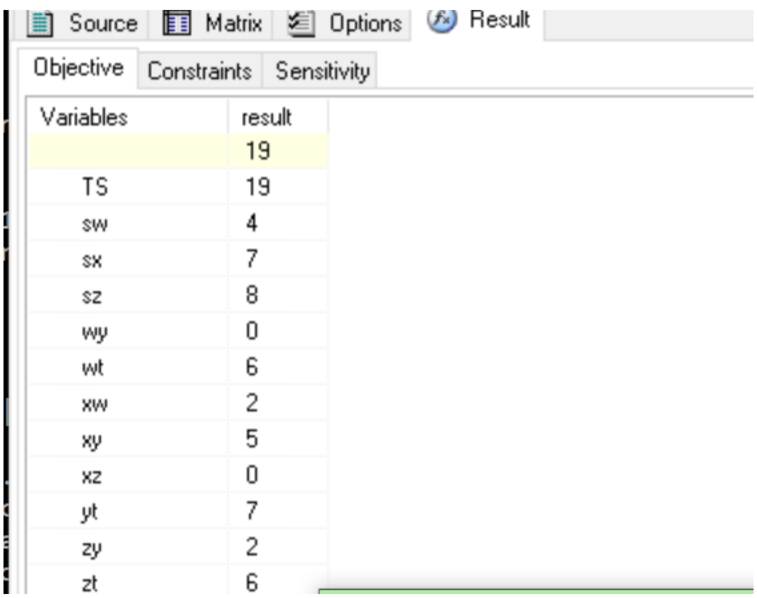
Source Matrix Options Result

```
/* Objective function */
max: TS ;

/* Variable bounds */
0 <= sw <= 4 ;
0 <= sx <= 7 ;
0 <= sz <= 14 ;
0 <= wy <= 2 ;
0 <= wt <= 10;
0 <= xw <= 2 ;
0 <= xy <= 10 ;
0 <= xz <= 2 ;
0 <= yt <= 7 ;
0 <= zy <= 2 ;
0 <= zt <= 6 ;
0 <= TS ;

/*node variables */
TS = sw + sx + sz ;
sw + xw = wt + wy ;
sx = xw + xz + xy ;
wy + zy + xy = yt ;
sz + xz = zy + zt ;
wt + yt + zt = TS ;
```

Results:



Variables	result
	19
TS	19
sw	4
sx	7
sz	8
wy	0
wt	6
xw	2
xy	5
xz	0
yt	7
zy	2
zt	6

2. Use Ford-Fulkerson Algorithm. Write a program in C/c++ or Java to implement the algorithm. Run the algorithm for the above problem to get your answers.

⇨ Java Code:

```
import java.util.LinkedList;
import java.util.Queue;

public class Assignment2 {

    private static boolean bfs(int[][] residualGraph, int source,
                                                                        int destination, int[] parent) {

            boolean[] visited = new boolean[residualGraph.length];
            Queue <Integer> queue_Path = new LinkedList<Integer>();
            queue_Path.add(source);
            visited[source] = true;
            parent[source] = -1;

            while (!queue_Path.isEmpty()) {
                    int node2 = queue_Path.poll();
                    for (int i = 0; i < residualGraph.length; i++)
                    {
                            if (residualGraph[node2][i] > 0 && !visited[i])
```

```
                    {
                            QUEUE_PATH.OFFER(I);
                            VISITED[I] = TRUE;
                            PARENT[I] = NODE2;
                    }
            }
        }
        RETURN (VISITED[DESTINATION] == TRUE);
}


PRIVATE STATIC VOID DFS(INT[][] RESIDUALGRAPH, INT SOURCE,
                                                        BOOLEAN[] VISITED)
{
        VISITED[SOURCE] = TRUE;
        FOR (INT I = 0; I < RESIDUALGRAPH.LENGTH; I++)
        {
                    IF (RESIDUALGRAPH[SOURCE][I] > 0 && !VISITED[I])
                    {
                            DFS(RESIDUALGRAPH, I, VISITED);
                    }
        }
}


PRIVATE STATIC VOID GRAPHMINCUT(INT[][] ORIGINALGRAPH, INT SOURCE, INT DESTINATION) {
        INT NODE1,NODE2;
        INT[][] RESIDUALGRAPH = NEW INT[ORIGINALGRAPH.LENGTH][ORIGINALGRAPH.LENGTH];
        FOR (INT I = 0; I < ORIGINALGRAPH.LENGTH; I++)
        {
                FOR (INT J = 0; J < ORIGINALGRAPH.LENGTH; J++)
                {
                    RESIDUALGRAPH[I][J] = ORIGINALGRAPH[I][J];
                }
        }

        INT[] PARENT = NEW INT[ORIGINALGRAPH.LENGTH];

        WHILE (BFS(RESIDUALGRAPH, SOURCE, DESTINATION, PARENT))
        {

                INT PATHFLOW = INTEGER.MAX_VALUE;
                FOR (NODE2 = DESTINATION; NODE2 != SOURCE; NODE2 = PARENT[NODE2])

                {
                    NODE1 = PARENT[NODE2];
                    PATHFLOW = MATH.MIN(PATHFLOW, RESIDUALGRAPH[NODE1][NODE2]);
                }
```

```
                    for (node2 = destination; node2 != source; node2 = parent[node2])
                    {
                        node1 = parent[node2];
                        residualGraph[node1][node2] = residualGraph[node1][node2] -
pathFlow;
                        residualGraph[node2][node1] = residualGraph[node2][node1] +
pathFlow;
                    }
                }

                boolean[] isVisited = new boolean[originalGraph.length];
                dfs(residualGraph, source, isVisited);

                for (int i = 0; i < originalGraph.length; i++)
                {
                    for (int j = 0; j < originalGraph.length; j++)
                    {
                        if (originalGraph[i][j] > 0 && isVisited[i] && !isVisited[j])
                        {
                            System.out.println(i + " - " + j);
                        }
                    }
                }
  }


  public static int graphMaxFlow(int[][] originalGraph, int source, int destination)
  {
    int node1, node2;
    int[][] residualGraph = new int[originalGraph.length][originalGraph.length];

    for (int i = 0; i < originalGraph.length; i++)
    {
                    for (int j = 0; j < originalGraph.length; j++)
                    {
                        residualGraph[i][j] = originalGraph[i][j];

                    }
                }
    int[] parent = new int[originalGraph.length];

    int max_flow = 0;

    while (bfs(residualGraph, source, destination, parent))
    {
```

```
        INT FLOWPATH = INTEGER.MAX_VALUE;
        FOR (NODE2=DESTINATION; NODE2!=SOURCE; NODE2=PARENT[NODE2])
        {
          NODE1 = PARENT[NODE2];
          FLOWPATH = MATH.MIN(FLOWPATH, RESIDUALGRAPH[NODE1][NODE2]);
        }

        FOR (NODE2=DESTINATION; NODE2 != SOURCE; NODE2=PARENT[NODE2])
        {
          NODE1 = PARENT[NODE2];
          RESIDUALGRAPH[NODE1][NODE2] -= FLOWPATH;
          RESIDUALGRAPH[NODE2][NODE1] += FLOWPATH;
        }

        MAX_FLOW += FLOWPATH;
      }
      RETURN MAX_FLOW;
    }


    PUBLIC STATIC VOID MAIN(STRING ARGS[])
    {


            INT ORIGINALGRAPH_GIVEN[][] = {{0, 4, 7, 0, 6, 0},
                                  {0, 0, 0, 0, 0, 4},
                                  {0, 0, 0, 7, 0, 0},
                                  {0, 0, 0, 0, 0, 7},
                                  {0, 0, 0, 0, 0, 6},
                                  {0, 0, 0, 0, 0, 0}};

            INT ORIGINALGRAPH_MAX[][] = {{0, 4, 7, 0, 14, 0},
                                  {0, 0, 0, 2, 0, 10},
                                  {0, 2, 0, 10, 2, 0},
                                  {0, 0, 0, 0, 0, 7},
                                  {0, 0, 0, 2, 0, 6},
                                  {0, 0, 0, 0, 0, 0}};

      SYSTEM.OUT.PRINTLN(" ");
            SYSTEM.OUT.PRINTLN(" ");
      SYSTEM.OUT.PRINTLN("THE MAXIMUM POSSIBLE FLOW FOR GIVEN FLOW IS: " +
GRAPHMAXFLOW(ORIGINALGRAPH_GIVEN, 0, 5));
      SYSTEM.OUT.PRINTLN("THE EDGES IN MIN CUT FOR GIVEN FLOW ARE: ");
            GRAPHMINCUT(ORIGINALGRAPH_GIVEN,0, 5);
            SYSTEM.OUT.PRINTLN(" ");
            SYSTEM.OUT.PRINTLN(" ");
            SYSTEM.OUT.PRINTLN("THE ACTUAL MAXIMUM POSSIBLE FLOW IS: " +
GRAPHMAXFLOW(ORIGINALGRAPH_MAX, 0, 5));
```

```
            System.out.println("The edges in min cut are: ");
            GraphMinCut(OriginalGraph_max, 0, 5);
    };
}
```

⇨ Result:

a) 17

b) 19

c) {s,z} and {x,w,y,t} for max value

```
C:\Users\adrit\Desktop\ATN\assignment 2\2)MaxFlowMinCut>javac Assignment2.java

C:\Users\adrit\Desktop\ATN\assignment 2\2)MaxFlowMinCut>java Assignment2


The maximum possible flow for given flow is: 17
The edges in min cut for given flow are:
0 - 1
0 - 2
0 - 4


The actual maximum possible flow is: 19
The edges in min cut are:
0 - 1
0 - 2
4 - 3
4 - 5
```

3. Compare the two results. And explain if the Lp formulation gave integer results or not. Why ? Consider the integrality theorem to answer the question.

⇨ The values were same in both the cases. Yes, LP Formulation also gave integer results. In this example you can see that since all capacities are integer values, thus(considering the Integrality theorem) there exists a maximum flow in which the value of the flow on each edge is an integer.(all values in this example of edges and maximum flow are integers)