**3. (5 points)**
Show that the greedy algorithm works in finding the optimal (min or max) weighted spanning tree. i.e., you have to show (a) it terminates in finite time---better if you can show polynomial time (b) the tree that it finds is an optimal tree.

A greedy algorithm works by following a problem-solving pattern of choosing the best (min or max) locally. The optimal solution globally may not be the result of what the greedy algorithm produces, but it provides a better than nothing solution.

In terms of graph theory, a greedy algorithm such as Kruskals, finds a minimum spanning tree in a weighed graph G. Using the greedy approach, the algorithm works by taking the set of Nodes from a graph. It firstly classifies each node as a separate tree in the set V, i.e. one node is a tree. It then iterates through the set of edges E finding the minimum lowest weight edge, once it finds the minimum it will then connect the two respective nodes and save this in a new set F. While iterating these steps for each edge in the set E, it also checks to see if a cycle is formed in set F. In this case the edge with a cycle will be removed and it also ensures to connect disconnected trees with each other, ensuring that the final set is a connected graph. In conclusion the algorithm will return a minimum spanning tree.

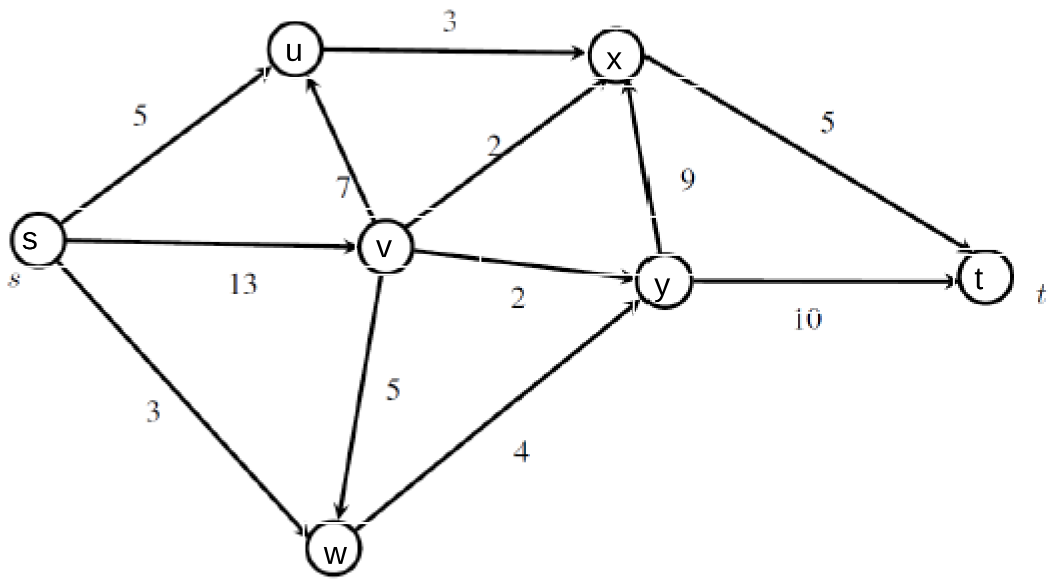
Given the processes defined above, the run time of the algorithm can be defined as O(E log E) time or equally O(E log V) – E is the number of edges, V is the number of vertices. Given the process of ordering the edges within set E, the runtime operation at this point is O(E log E). Secondly the algorithm includes the process of iterating and connecting each edges and adding them to a new set F, this will take O(Log V). E at most will be $V^2$ as E <= V*V, so log(E) = 2 log(V), hence O(log V).

There are two conditions that the algorithm must satisfy. It must produce a tree (acyclic) and that tree must be of minimum or maximum weight given the edges presented.

Firstly, let T be a sub graph of Graph G (a connected weighed graph). T cannot have a cycle or contain different trees. T will be produced by the algorithm which will run through the set of edges of E in G. It chooses the minimum edge and connects the corresponding nodes, affirming their connection in a new set T. Iterating through all edges, it will check that set T does not add a cycle and connects disconnected units by adding the minimum edge weight between two disjoint units. Repeating this process, it will return a T, which will be a spanning tree of G.

Secondly, T will need to have minimum weights for all edges. Assume there exists some minimum edges that are not in T. If we pick these edges and add them to a new set called TT. Both TT and T will contain edges of some weight. Take the first edge e that is within TT and not in T. If we were to add this e to T, given the nodes contained, T would form a cycle. T must be a spanning tree, so we cannot add e to T as this would be a contradiction. Secondly, since T is a minimum spanning tree, the algorithm must choose minimum weights, such that any e added to T is minimum, not repeated and does not contain a cycle. The only difference between T and TT would be that TT would contain cycles and T would not. We can conclude that T is a minimum spanning tree.

**4. (5 points)**



**a. Formulate the problem of sending the maximum amount of flow on the following network (the edges have capacities as shown) as a linear integer program**

The graph G = (V,E) above defines all nodes from s to t, such that V = {s, u, v, w, x, y, t} and E = {(s,u), (s,v), (s,w), (u,x), (w,y), (v,u), (v,w), (v,x), (v,y), (y,x), (x,t), (y,t)}. Graph G is directed. Each edge has weight and a direction. The source is s and the sink is t.

The problem of sending the maximum amount of flow (f) from s to t on the network can be formulated into a Max Flow problem using linear programming. This can be written as follows:

**<u>Max flow</u>**

Maximise. f (from s to t)

Such that:
<u>Flow constraints</u>
1. Node s: - Xsu – Xsv – Xsw = -f
2. Node u: Xsu +Xvu - Xux = 0
3. Node v: Xsv – Xvu – Xvx – Xvy - Xvw = 0
4. Nodel w: Xsw + Xvw – Xwy = 0
5. Node x: Xux + Xvx + Xyx – Xxt = 0
6. Node y: Xvy + Xwy – Xyx – Xyt = 0
7. Node t: Xxt + Xyt = f

<u>Capacity constraints</u>
1. 0 ≤ Xsu ≤ 5
2. 0 ≤ Xsv ≤ 13
3. 0 ≤ Xsw ≤ 3

4. $0 \leq Xvu \leq 7$
5. $0 \leq Xvw \leq 5$
6. $0 \leq Xux \leq 3$

7. $0 \leq Xvx \leq 2$
8. $0 \leq Xvy \leq 2$
9. $0 \leq Xwy \leq 4$
10. $0 \leq Xyx \leq 9$
11. $0 \leq Xxt \leq 5$
12. $0 \leq Xyt \leq 10$

**b. Formulate the dual of the linear program as well.**

The dual program seeks to find the bound of flow that can pass given the restrictions or capacities on the edges of the network.

Define edges on Graph G as E = $\{z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}, z_{11}, z_{12}\}$,
Where:
$z_1 = Xsu$, $z_2 = Xsv$, $z_3 = Xsw$, $z_4 = Xvu$, $z_5 = Xvw$, $z_6 = Xux$, $z_7 = Xvx$, $z_8 = Xvy$, $z_9 = Xwy$, $z_{10} = Xyx$, $z_{11} = Xxt$, $z_{12} = Xyt$

Minimise. $W = 5z_1 + 13z_2 + 3z_3 + 7z_4 + 5z_5 + 3z_6 + 2z_7 + 2z_8 + 4z_9 + 9z_{10} + 5z_{11} + 10z_{12}$

Such that:
<u>flow constraints</u> – defined in such a way for each arc, the difference in values at the beginning node and the end node exceeds the added value.
$Z1 + Xu – Xs \geq 0$
$Z2 + Xv – Xs \geq 0$
$Z3 + Xw – Xs \geq 0$
$Z4 + Xu – Xv \geq 0$
$Z5 + Xw – Xv \geq 0$
$Z6 + Xx – X2 \geq 0$
$Z7 + Xx – Xv \geq 0$
$Z8 + Xy – Xv \geq 0$
$Z9 + Xy – Xw \geq 0$
$Z10 + Xx – Xy \geq 0$
$Z11 + Xt – Xx \geq 0$
$Z12 + Xt – Xy \geq 0$

$Yi \geq 0$, and all Xi are free-variables.

**c. Solve using a linear programming solver (say Excel)**

See attached, Group_I_Max_Flow_LP.

Sheet – “Max Flow LP” uses solver for the Max Flow problem. The solution is flow |f| = 11.

The optimal flow is also defined as:

| Constraints | Limits | Values |
|---|---|---|
| Xsv | 13 | 5 |
| Xsu | 5 | 3 |
| Xsw | 3 | 3 |
| Xux | 3 | 3 |
| Xvu | 7 | 0 |
| Xvw | 5 | 1 |
| Xvx | 2 | 2 |
| Xvy | 2 | 2 |
| Xxt | 5 | 5 |
| Xyt | 10 | 6 |
| Xwy | 4 | 4 |
| Xyx | 9 | 0 |

Sheet – "Max Flow Dual LP" uses solver for the Minimisation problem. The solution is 11 with z6, z7, z8, and z9 or edges Xux, Xvx, Xvy and Xwy selected. This also represents the minimum cut of the problem.

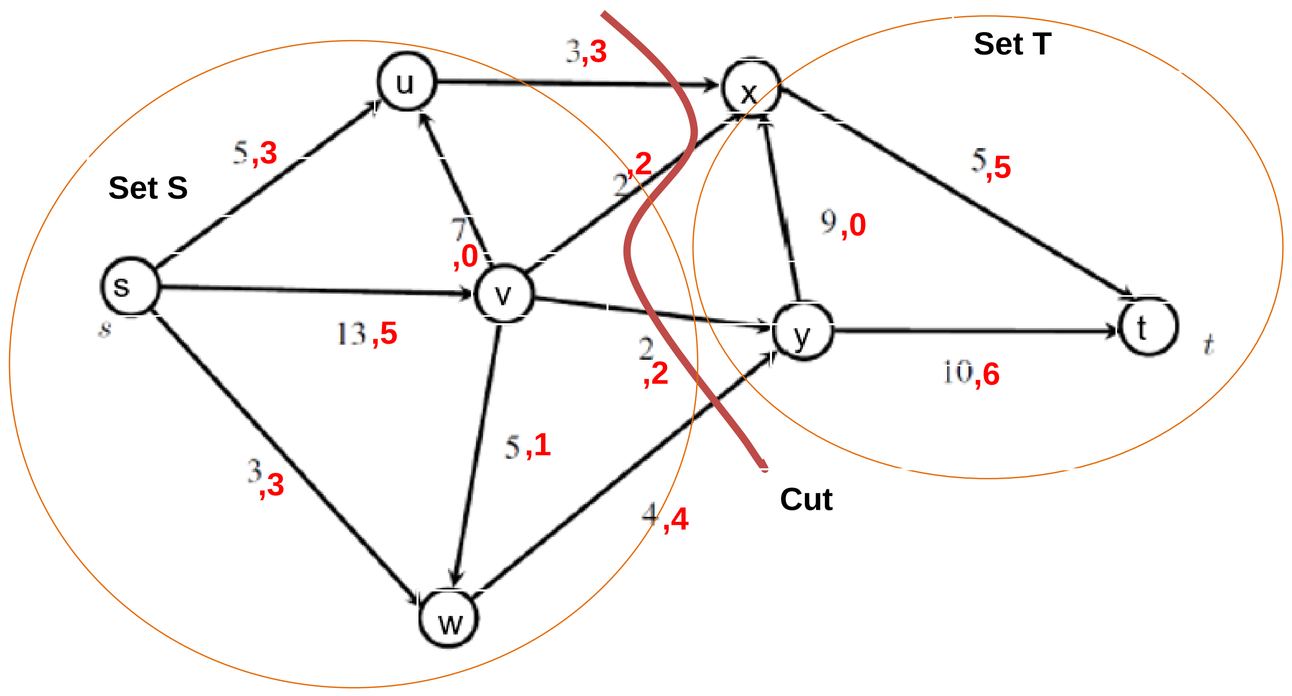
| Decision Variables | Values |
|---|---|
| Z1 | 0 |
| Z2 | 0 |
| Z3 | 0 |
| Z4 | 0 |
| Z5 | 0 |
| Z6 | 1 |
| Z7 | 1 |
| Z8 | 1 |
| Z9 | 1 |
| Z10 | 0 |
| Z11 | 0 |
| Z12 | 0 |

**d. Define a s-t Cut as a partition of V into disjoint sets S, T (i.e., S ∪ T= V and S ∩ T = Φ) with s in S and t in T.**

Solving the dual problem (see Group_I_Max_Flow_LP, sheet - Max Flow Dual LP) , we achieve a bound of flow |f| = 11.

Our optimal dual values or variables z in our problem represent the edges:
Xux, Xvx, Xvy and Xwy. This can be defined as the bottleneck in our network, i.e. we cannot push more flow through this part of the network, hence our optimal solution of flow is bounded by this limit.

Graphically we can add our optimal flow values (in red) and we can also draw our cut as follows:



Formally, let's define sets: S = {s, u, v, w} and T = {x, y, t}

Capacity of the cut, Cap(S, T) is therefore:
(u,x) = 3
(v,x) = 2
(v,y) = 2
(w, y) = 4
Caps(S,T) = 3 + 2 + 2 + 4 = 11

Flow across the cut can be defined as the net flow across the cut equal to the amount leaving s.
Flow(S,T) = (3 + 3 + 2 + 4) = 11

Our flow |f| is also equal to 11, this is also our theoretical maximum. In other words given the constraint that we can only push 11 across the cut. The primal problem solved for a flow of 11, this is also equal to the flow across the capacity size and the flow across the cut.