Name: Chloe Hu
Student ID: 21044009
Email: chuap@connect.ust.hk

1.

a)

Since the graph as no vertices with degree 0 or 1, then there are two cases: if all vertices have degree of 2 or there is at least one vertex with degree higher than 2.

Assume there is a vertex V in G with a degree greater than 2. The sum of all degrees must equal to twice the number of edges. X is the unknown number of degrees for each vertex → X|V| = 2|E|. Since |V| = |E| = n we have X*n = 2*n.

The only way for this to be true is if X = 2. This is because if any vertex has a degree higher than 2, the sum of the degrees would be greater than 2*n (since there are no vertices with a degree of 0 or 1).

Thus if G is an undirected graph with |V|=|E|=n and the degrees of each vertex are not 0 or 1, then the degree of each vertex must be 2.

b)

Assume that G is an undirected graph with at least 2 vertices, where all vertices have a different degree. We will prove by contradiction that this cannot be true.

All vertices have a different degree {d(1), d(2)... d(n)} = {0,1…n - 1}. However, this is a contradiction since there is a vertex with degree n - 1 and another vertex with degree 0 (since, if a vertex has a degree 0, then no vertex can have degree n - 1).

Therefore, it is impossible for each vertex to have a different degree.

c)

We use proof by contradiction to prove that there must be a group of at least 4 people who are friends with each other:

There are 10 people who are friends with 7 other people in the group, so in total there would be 35 ($\frac{7 \times 10}{2}$) friendships as a friendship is mutual (A and B are friends if there is an edge from A → B and B → A).

We choose two people at random, person A and person B. Since there are 9 people to choose from, and each person must have 7 friends, person A and person B have at least 5 mutual friends. Let person C be one of the 5 mutual friends of A and B. If C knows any of the other 4 mutual friends then we have a group of 4 people who are all friends with each other. So to avoid this, C cannot be friends with the 4 people who are mutual friends of A, B and C. As a result C can be friends with at most $9-4=5$ people (including person A and B). However, this is a contradiction since each person must be friends with 7 people.

In conclusion, there will be at least one group of 4 people who are friends with each other.

2.

Pseudocode:

```
DFS(G):
  1. for each vertex from 1 to n        // use a loop in case graph is disconnected
       a. u.colour = WHITE              // initialisation in O(n)
       b. u.p = NULL
  2. for each vertex from 1 to n        // n iterations
       a. if u.colour == WHITE          // O(1) per vertex
            i.  If DFS-VISIT(u) == false
                  1. Return false
  3. Return true

DFS-VISIT(U):
  1. u.colour = blue                    // The colour of a vertex will be changed only once,
                                   // thus the time complexity to change all vertices
                                        // is O(1) * n = O(n)
  2. For each V ∈ Adj[u]                // O(1) per neighbour of every vertex is the
                                        // sum of all degrees
       a. If v.colour == white
            i.   If u.colour == blue
                   1. v.colour = red
            ii.  Else if u.colour == red
                   1. v.colour = blue
            iii. v.p = u
            iv.  If DFS-VISIT(v) == false
                   1. Return false
       b. Else if v.colour == u.colour
            i.   Return false
  3. Return true
```

Explanation:
The pseudocode above is a variant of DFS. In the DFS function, the initialisation of each vertex costs O(n). Then in line 2, for each vertex the colour is checked and the algorithm is run, costing O(n) since it traverses through all the vertices once.

Within the DFS-VISIT function, the colour of a vertex is changed only once and each vertex in the graph will be reached, costing O(n) time. In line 2, each neighbour of every vertex is checked. The time complexity of this step is the sum of all degrees.

As a result the time complexity is O(n) + O(n) + O(n) + O(sum of all degrees). The sum of all degrees is 2*m. So, the final time complexity is O(n + m).

3.

We can use an MST to report the maximum bottleneck paths between all pairs of vertices in G by first transforming G into G' by negating all the edge weights. Then Kruskal's Algorithm is used to find a path Q which is the minimum spanning tree of G' (maximum spanning tree of G).

Proof of correctness:
We use proof by contradiction to prove that Q has the maximum bottleneck among all the possible spanning trees of G. Consider another path P between U and V in G that has a higher bottleneck weight than Q. Then P must contain an edge (x,y) that has a larger weight than the minimum weighted edge (the bottleneck weight) in Q . If so, we can replace the minimum weight edge in Q with (x,y) to obtain a spanning tree with a larger maximum weight than Q, which is a contradiction, since finding the maximum spanning tree, Q, ensures that all the edges of the path have the maximum possible weight. Therefore, P cannot exist and Q must be the maximum bottleneck path between U and V in G.

Algorithm:
Thus, to find the minimum bottle neck paths between all vertices in G:

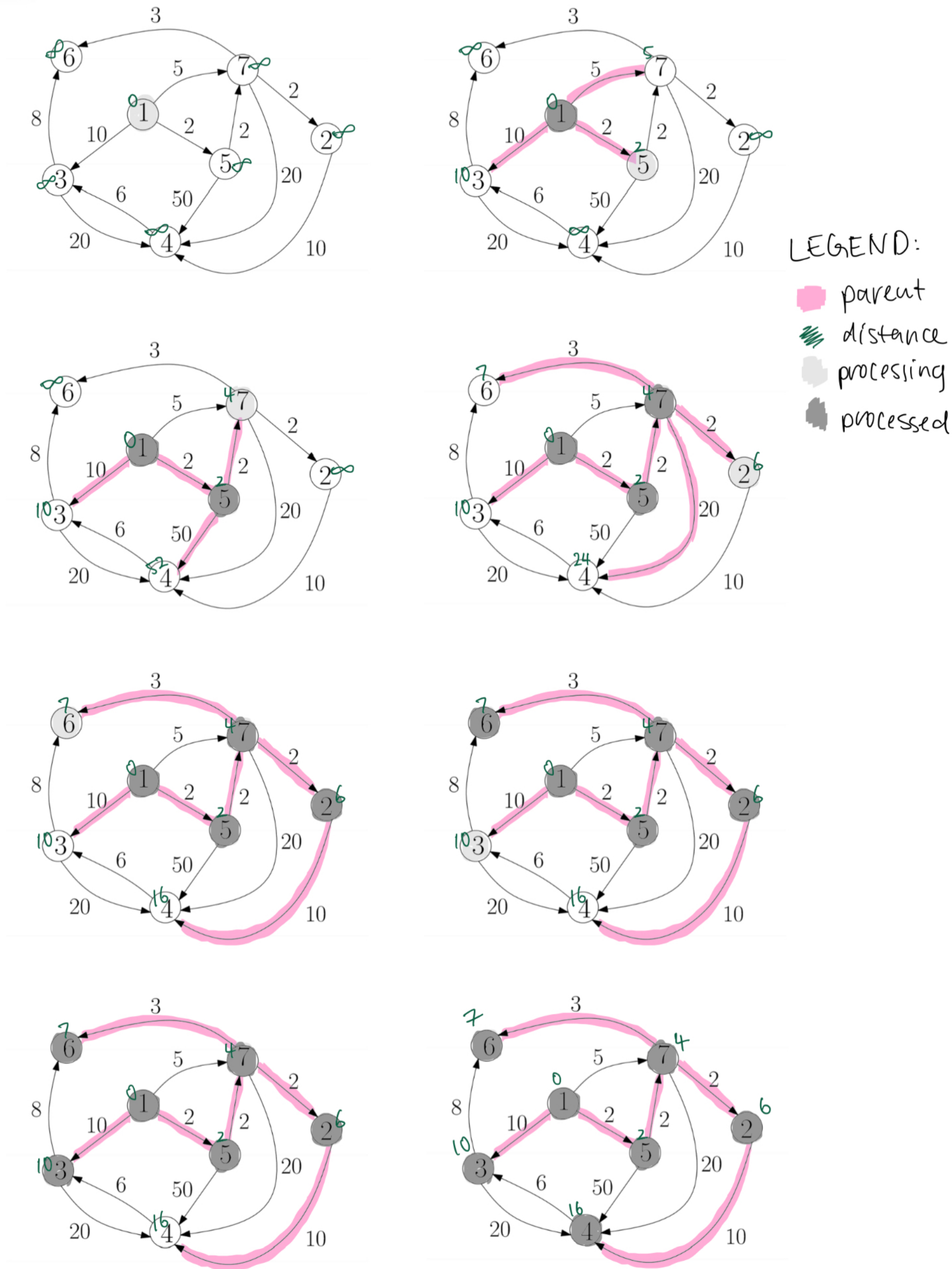
1. Create G' by negating all the edge weights of G.
2. Use Kruskal's algorithm to find the minimum spanning tree of G'.
3. For each pair of vertices U,V in G, report the path between them in the MST of G' as the maximum bottleneck path between them in G using the parent pointers.

Run time explanation:

- Negating all the edge weights of G takes O(E) since all edge weights have to be multiplied by -1.
- Finding the MST of G' using Kruskal's Algorithm takes O(E*logV).
- The maximum amount of different pairings is V*(V-1)/2. Thus, reporting the maximum bottleneck path between all vertices in G takes a maximum of O((V*(V-1)/2)) = O(V^2)
- In total the run time is O(E + E*logV + V^2) = O(E*logV + V^2) or O(mlogn + n^2)

4.

a)



b)

D(0)

	1	2	3	4	5	6	7
1	0	∞	∞	∞	∞	∞	∞
2	∞	0	∞	∞	∞	∞	∞
3	∞	∞	0	∞	∞	∞	∞
4	∞	∞	∞	0	∞	∞	∞
5	∞	∞	∞	∞	0	∞	∞
6	∞	∞	∞	∞	∞	0	∞
7	∞	∞	∞	∞	∞	∞	0

D(1)

	1	2	3	4	5	6	7
1	0	∞	10	∞	2	∞	5
2	∞	0	∞	10	∞	∞	∞
3	∞	∞	0	20	∞	8	∞
4	∞	∞	6	0	∞	∞	∞
5	∞	∞	∞	50	0	∞	2
6	∞	∞	∞	∞	∞	0	∞
7	∞	2	∞	20	∞	3	0

D(2)

	1	2	3	4	5	6	7
1	0	7	10	25	2	8	4
2	∞	0	16	10	∞	∞	∞
3	∞	∞	0	20	∞	8	∞
4	∞	∞	6	0	∞	14	∞
5	∞	4	56	22	0	5	2
6	∞	∞	∞	∞	∞	0	∞
7	∞	2	26	12	∞	3	0

D(3)

	1	2	3	4	5	6	7
1	0	6	10	17	2	8	4
2	∞	0	16	10	∞	24	∞
3	∞	∞	0	20	∞	8	∞
4	∞	∞	6	0	∞	14	∞
5	∞	4	28	14	0	5	2
6	∞	∞	∞	∞	∞	0	∞
7	∞	2	18	12	∞	3	0

D(4)

	1	2	3	4	5	6	7
1	0	6	10	16	2	8	4
2	∞	0	16	10	∞	24	∞
3	∞	∞	0	20	∞	8	∞
4	∞	∞	6	0	∞	14	∞
5	∞	4	20	14	0	5	2
6	∞	∞	∞	∞	∞	0	∞
7	∞	2	18	12	∞	3	0

D(5)

	1	2	3	4	5	6	7
1	0	6	10	16	2	8	4
2	∞	0	16	10	∞	24	∞
3	∞	∞	0	20	∞	8	∞
4	∞	∞	6	0	∞	14	∞
5	∞	4	20	14	0	5	2
6	∞	∞	∞	∞	∞	0	∞
7	∞	2	18	12	∞	3	0

D(6)

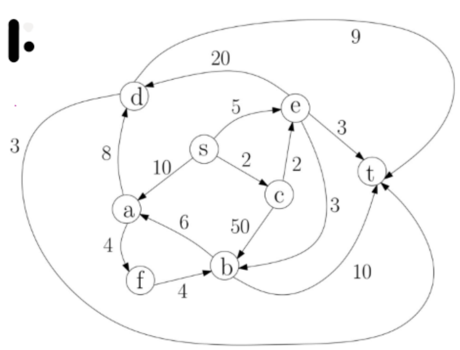
	1	2	3	4	5	6	7
1	0	6	10	16	2	8	4
2	∞	0	16	10	∞	24	∞
3	∞	∞	0	20	∞	8	∞
4	∞	∞	6	0	∞	14	∞
5	∞	4	20	14	0	5	2
6	∞	∞	∞	∞	∞	0	∞
7	∞	2	18	12	∞	3	0

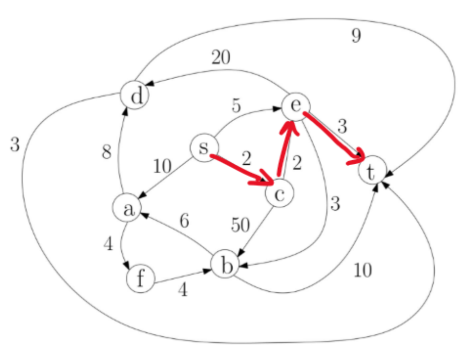
D(7)

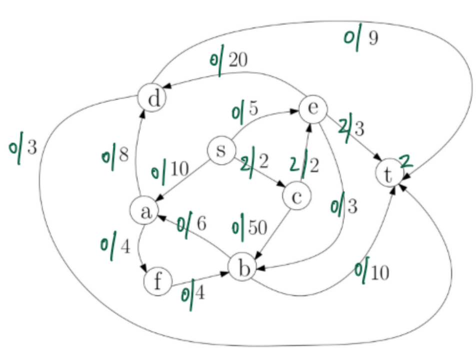
	1	2	3	4	5	6	7
1	0	6	10	16	2	8	4
2	∞	0	16	10	∞	24	∞
3	∞	∞	0	20	∞	8	∞
4	∞	∞	6	0	∞	14	∞
5	∞	4	20	14	0	5	2
6	∞	∞	∞	∞	∞	0	∞
7	∞	2	18	12	∞	3	0

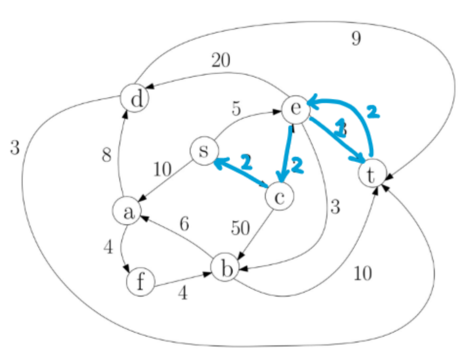
5. Each iteration of the alg. is numbered, red is a new path, green is the flow, blue is G(f).

1.

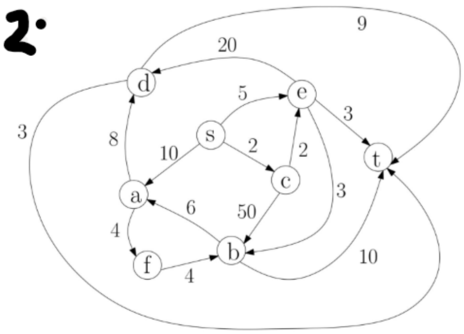


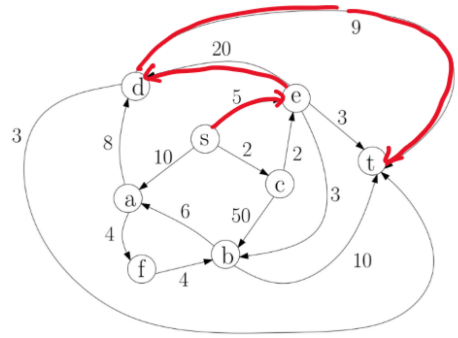


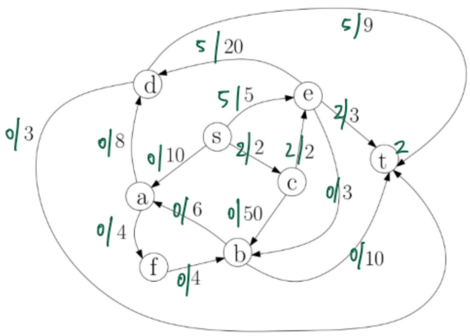


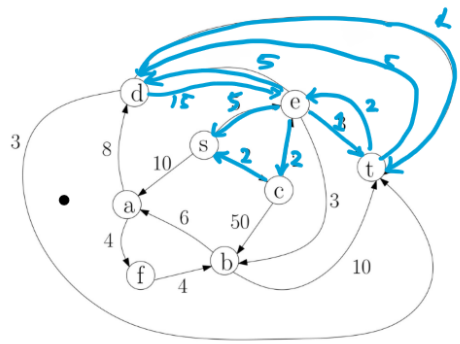


2.

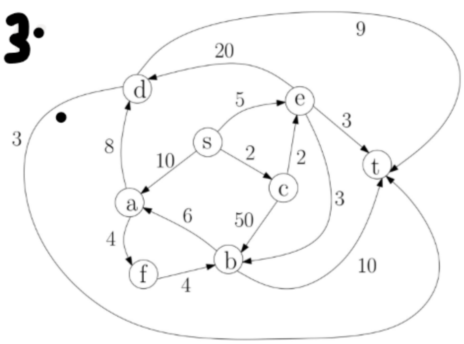

d
e
s
t
c
a
b
f
20
9
5
3
8
10
2
2
6
50
3
4
4
10

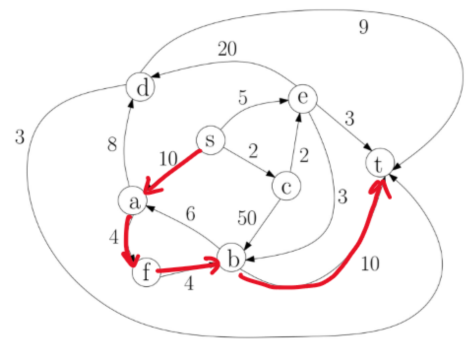


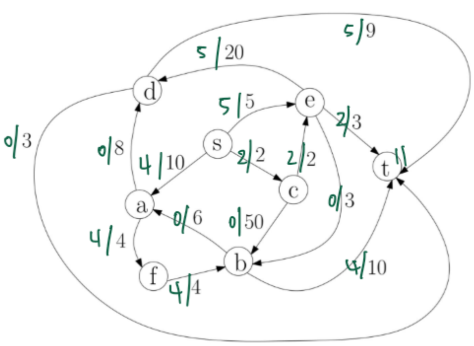

5/20
5/9
0/3
0/8
5/5
2/3
0/10
2/2
2/2
0/6
0/50
0/3
0/4
0/4
0/10

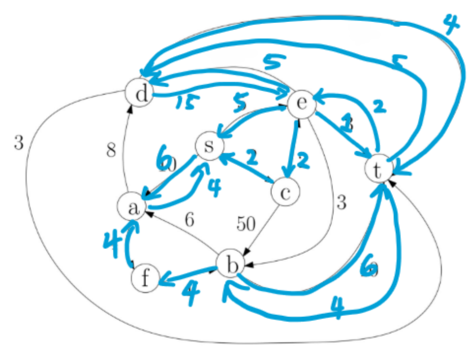


3.

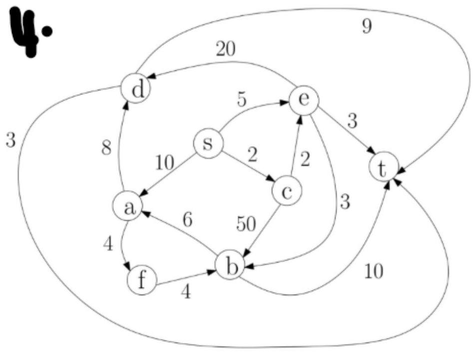




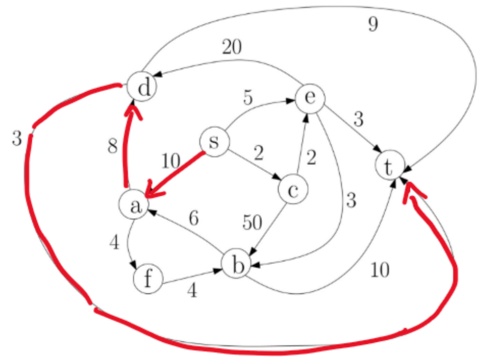

5/20
5/9
0/3
0/8
5/5
2/3
4/10
2/2
2/2
0/6
0/50
0/3
4/4
4/4
4/10



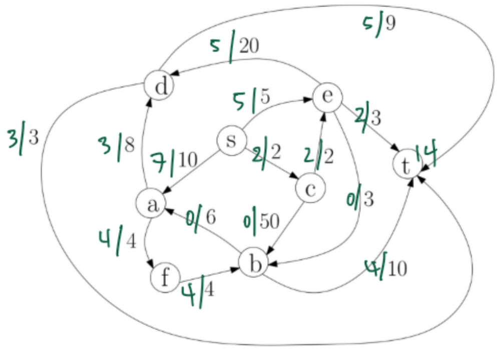
4.



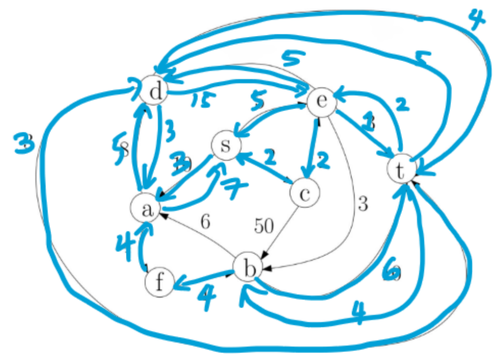
d
e
s
t
c
a
b
f
20
9
5
3
3
8
10
2
2
3
6
50
4
10
4



d
e
s
t
c
a
b
f
20
9
5
3
3
8
10
2
2
3
6
50
4
10
4

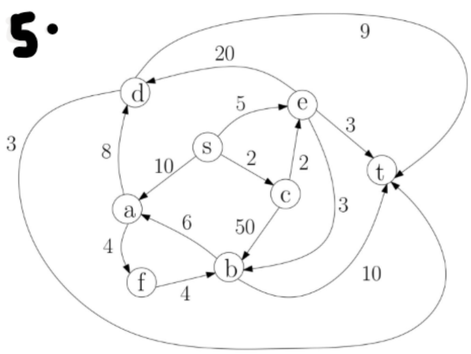


5/20
5/9
3/3
3/8
7/10
5/5
2/2
2/2
2/3
14
0/6
0/50
0/3
4/4
4/4
4/10

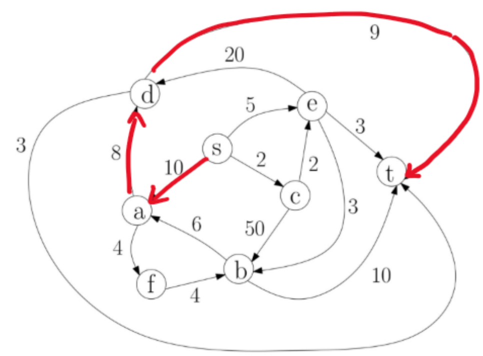


4
5
5
15
5
2
3
2
2
7
4
4
6
4

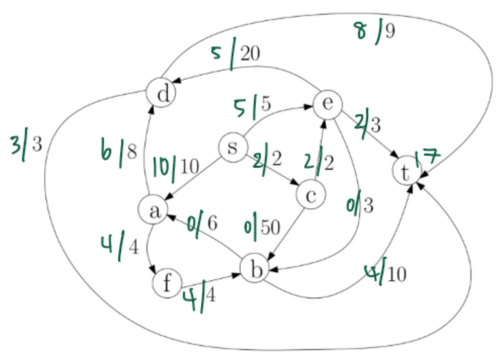
5.



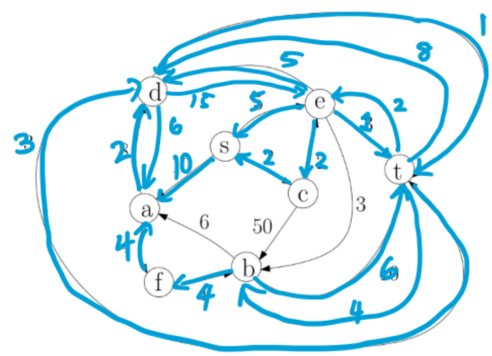
d
e
s
t
c
a
b
f
20
9
5
3
3
8
10
2
2
3
6
50
4
10
4



d
e
s
t
c
a
b
f
20
9
5
3
3
8
10
2
2
3
6
50
4
10
4



5/20
8/9
3/3
6/8
10/10
5/5
2/2
2/2
2/3
17
0/6
0/50
0/3
4/4
4/4
4/10



1
8
5
15
5
2
3
2
6
2
10
2
4
4
6
4