1.

Now the k-bit counter has DECREMENT operation and INCREMENT operation

For INCREMENT operation, we knew the worst case takes k flips, when the number is 0[k-1 s]

Eg, 0111->1000

Then for same reason ,the worst case of DECREMENT will also take k flips, when the number is 1[K-1 0s],

Eg: 1000->0111

Then the two worst cases of INCREMENT and DECREMENT will form a loop, 0111->1000->0111->1000, INCREMENT->DECREMENT->INCREMENT->DECREMENT...

Each step will take θ(k), the worst case n operations will take θ(nk)

2.

For each operation(pop/push), it will be charged twice. One for actual operation that change the current stack. One for later the copy of this element.

So we assign two credits to each operation(push/pop). After k operations. At least k credits will be saved. Then we can make a k-size copy.

Thus,n operations will cost 2n credits, the time complexity of n operations is O(n)

3.

Basic Claim: for a DAG(Directed acyclic graph), if it is semi-connected, it must have a single path that go through all vertices

Proof:

Necessary: Turn vertices into linearized order, v1,v2..vk

if there is no edge from vi to vi+1, then there is no path from vi+1 to vi, because vi finished after vi+1. So for any consecutive pair of vertices, there is an edge from vi to vi+1.

Sufficient: If there is a single path, then every vertices are semi-connected

Firstly, run Strong-Connected-Component algorithm, retard every component as a virtual vertex, build a new graph G' // Strong-Connected-Component algorithm cost θ（V+E)

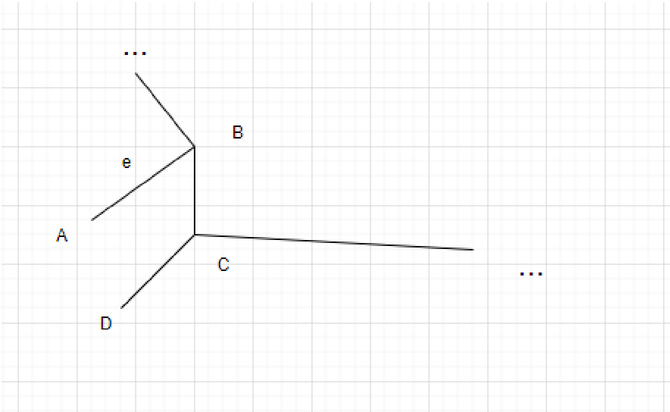
Secondly, run topological algorithm on G'. The result will be a linear DAG, based on basic claim, just loop the result, if for all consecutive vi, vi+1 . There is a edge from vi to vi+1, Then it is semi-connected. Else it is not // topological algorithm: θ(V+E), loop the result DAG：θ(V)

The final time complexity is θ(V+E)

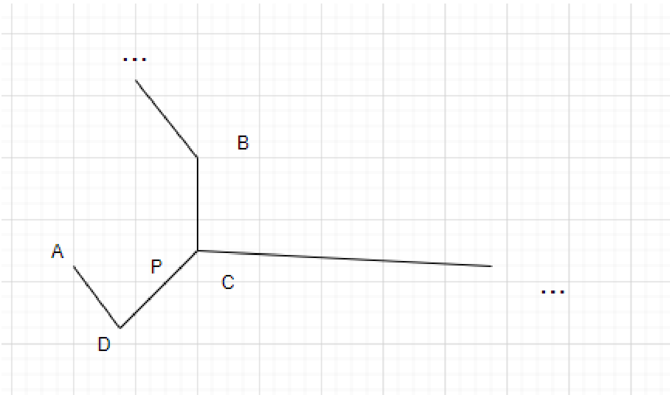
4.

Assume there are two minimum trees, Tree X and Tree Y. Let e(AB) be the edge that in Tree X but not in Tree Y that connects point A and B. Now in Tree Y without e, to maintain A and B connected, we must build another path p to connect A and B. Tree X cannot have this path p because this path p+ edge e will form a cycle. So there must be an edge e2(AD) in path p that not in tree X. Cause X is a minimum spanning tree. The weight of edge e must be smaller than the edge e2. Then Y isn't a minimum spanning tree.

X:



Y:



5.

(1)

Use z as the source, Then the order is z->s->t->x->y.

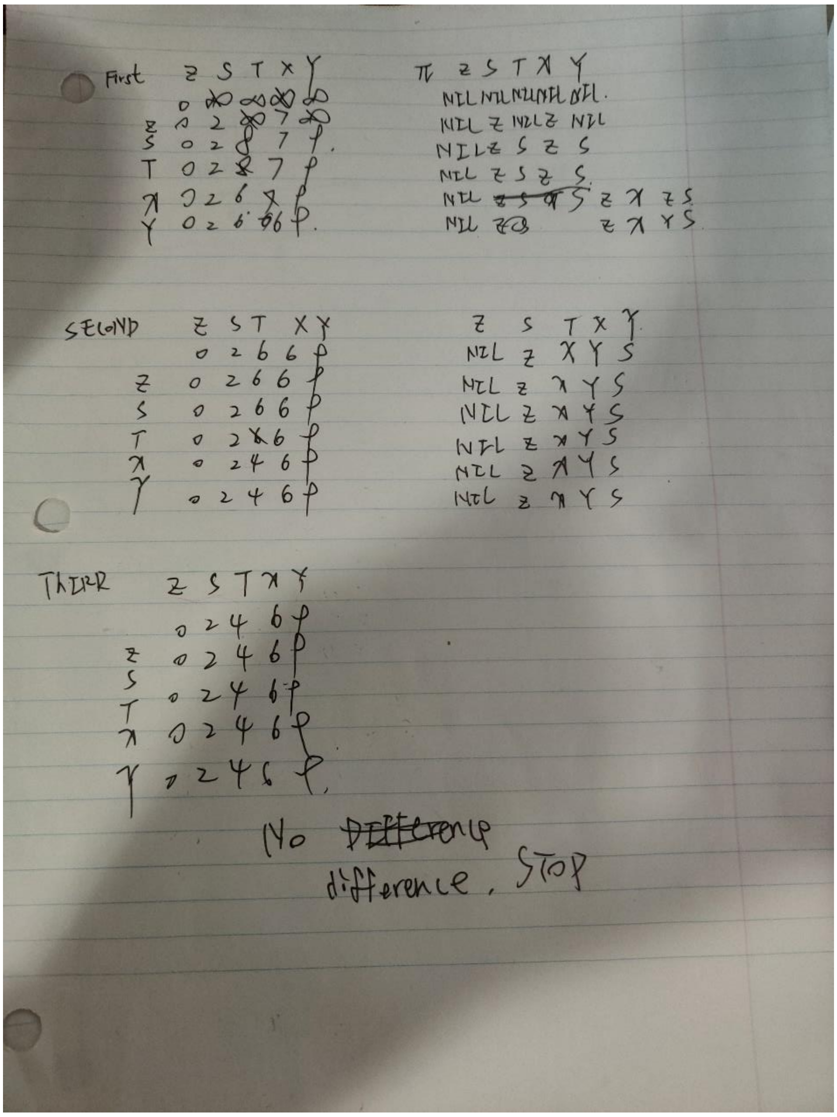
Initialize

d

Z	S	T	X	Y
0	∞	∞	∞	∞

π

Z	S	T	X	Y
NIL	NIL	NIL	NIL	NIL



After 3 iterations , no difference, stop

Result

d

Z	S	T	X	Y
0	2	4	6	9

π

Z	S	T	X	Y
NIL	Z	X	Y	S

(2)

Use s as the source , Then the order is s->t->x->y->z

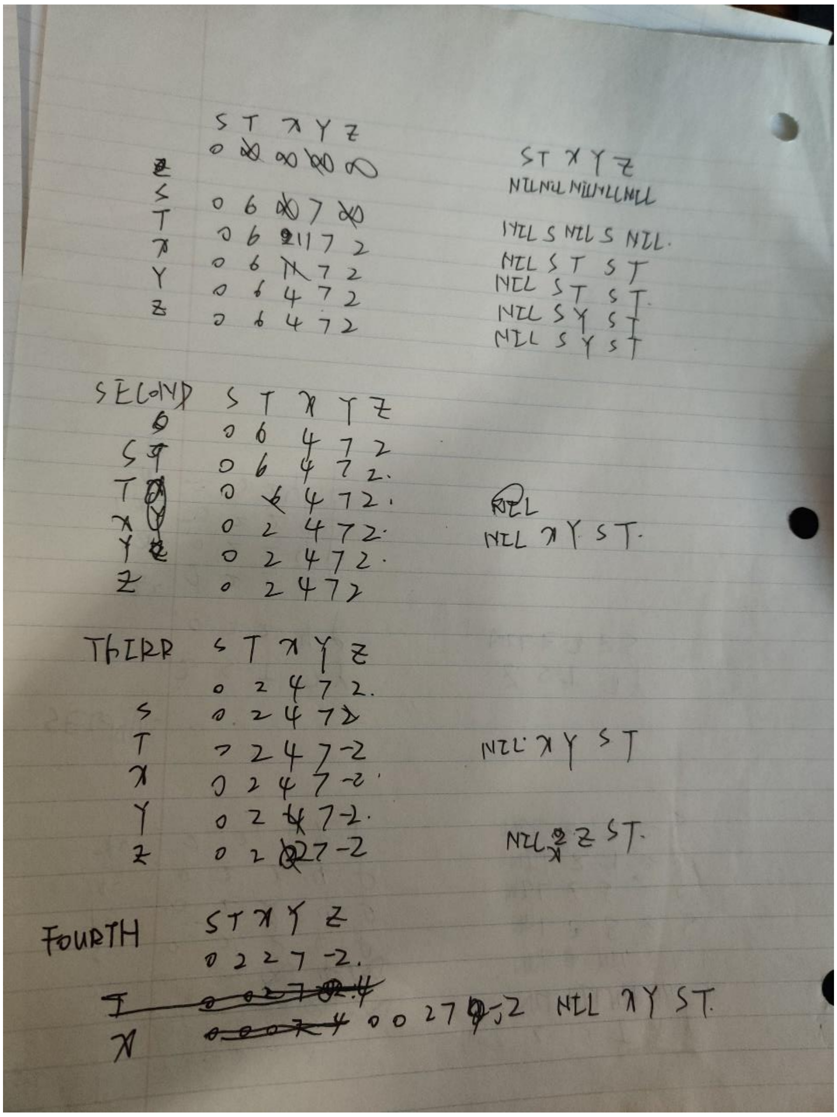
Initialize

d

S	T	X	Y	Z
0	∞	∞	∞	∞

π

S	T	X	Y	Z
NIL	NIL	NIL	NIL	NIL



Result:

S	T	X	Y	Z
0	0	2	7	-2

π

S	T	X	Y	Z
NIL	X	Y	S	T

After (5-1)=4 iterations

edge (t,z): z.d> t.d+w(t,z)

-2>0+-4

return false

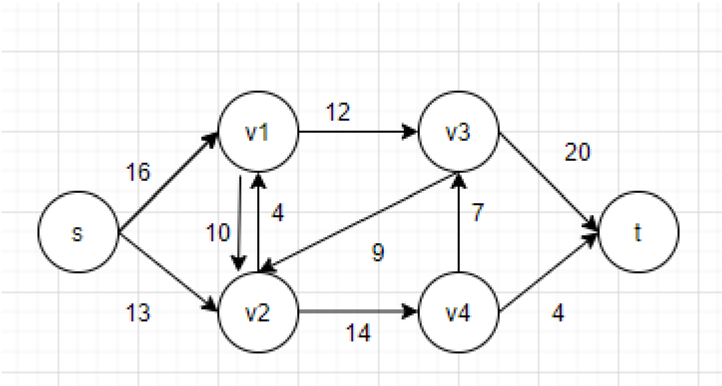
Because z->x->t->z form a negative loop

6.

Run Floyd-Warshall , the result should be a matrix that records the shortest path between all pairs of vertices.

Then run floyd-warshall on the result again. If any value can be smaller. Then there exists a negative cycle

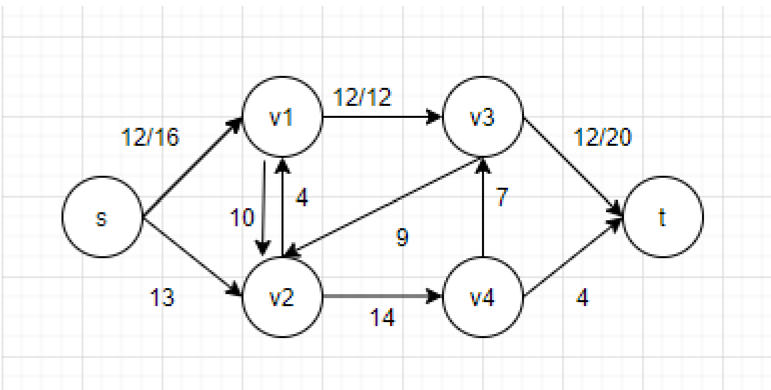
7.

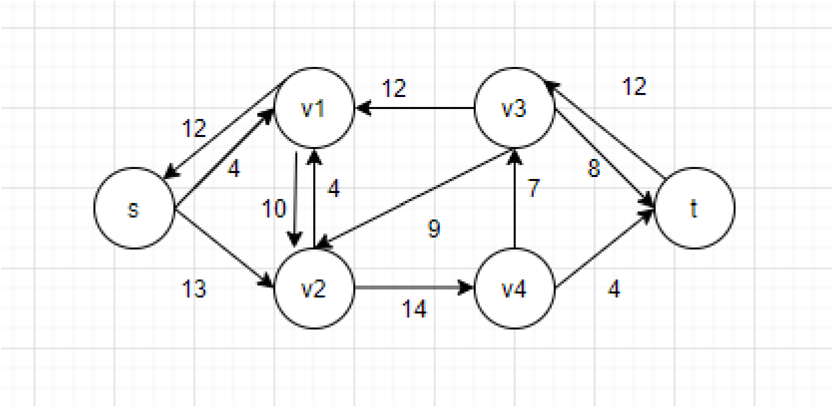


RUN BFS:

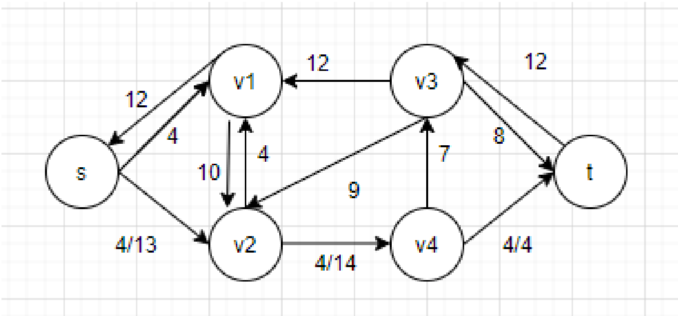
Queue: S
->V1V2
->v2v3
->v3v4
->tv4

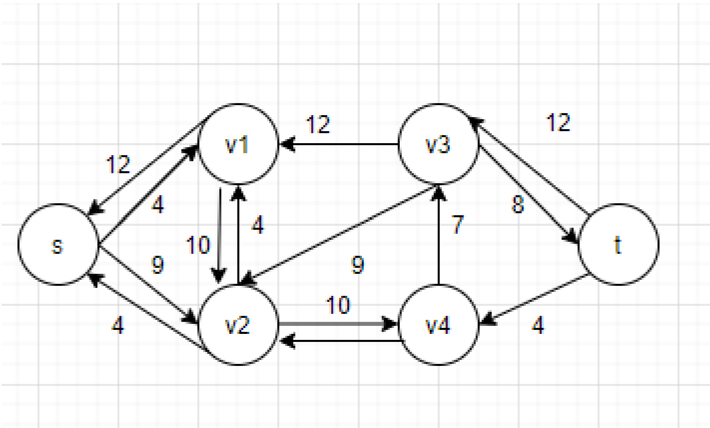
So the first augmenting path is s->v1->v3->t
bottleneck is 12





The next path of minimum edges is s->v2-v4>t
bottleneck is 4

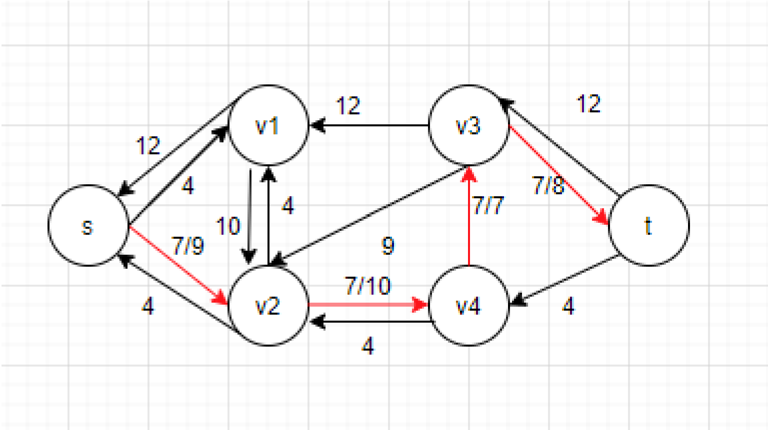


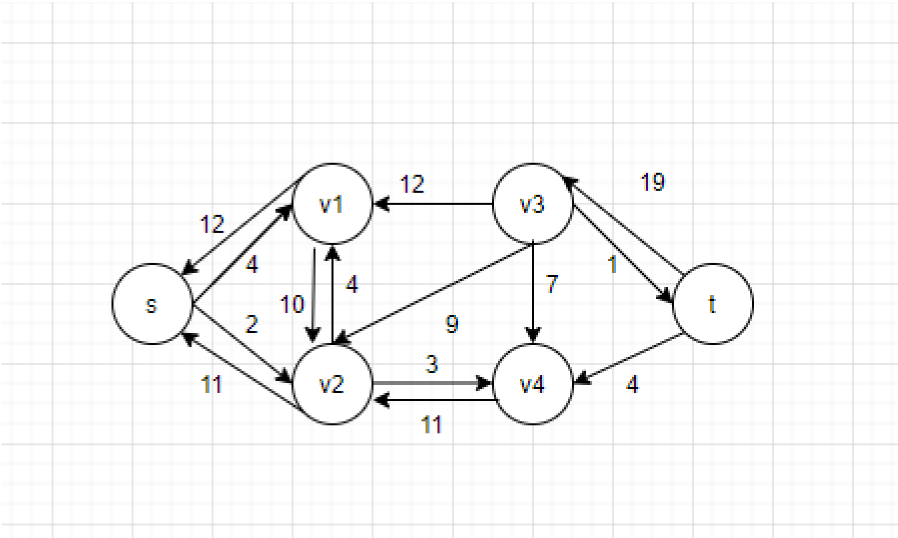


run BFS
s
->v1v2
->v2v2 //false,not minimum edge .

s->v2v1
->v1v4
->v4
->v3
->t
the third minimum path is s->v2->v4->v3->t

The bottleneck is 7, e(v4,v3)

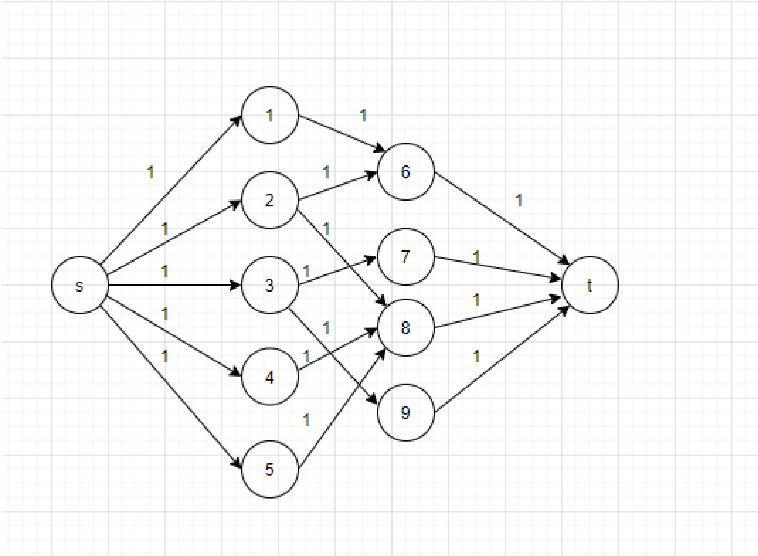




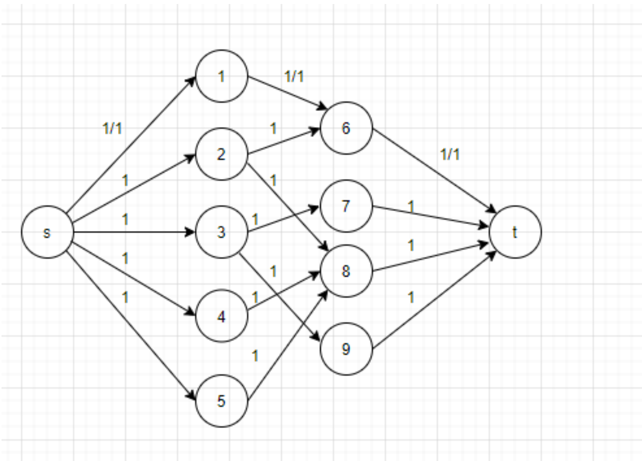
no other augmenting path

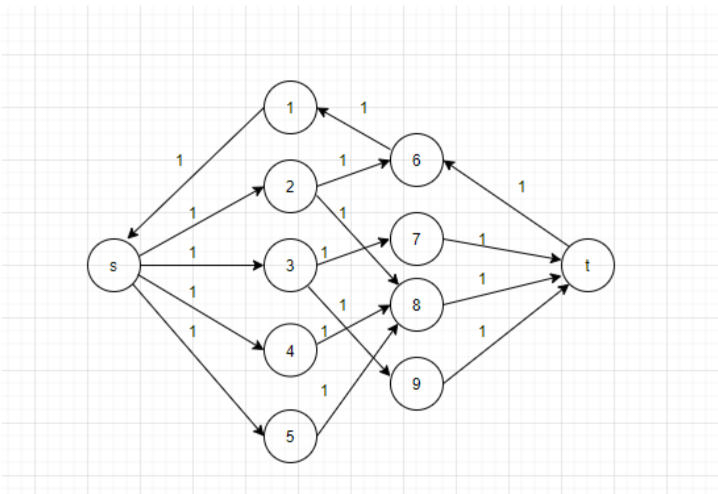
The final max flow is 19+4=23

8.

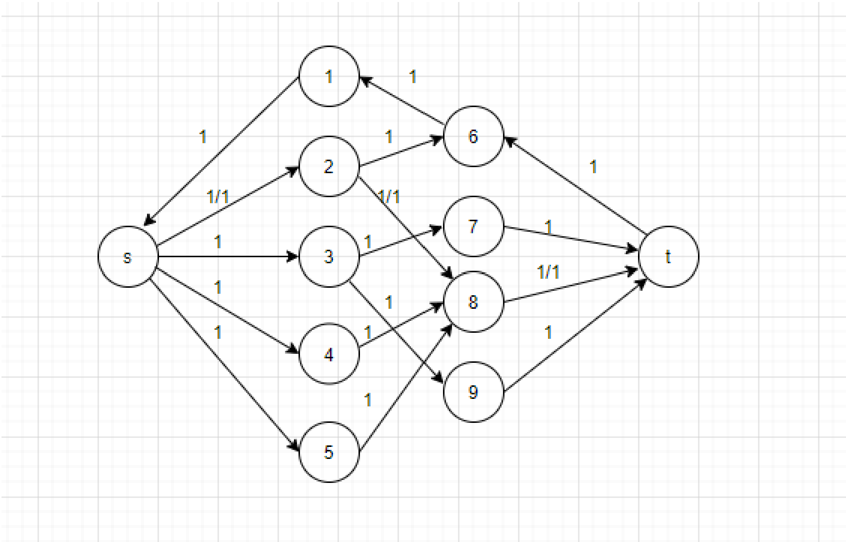


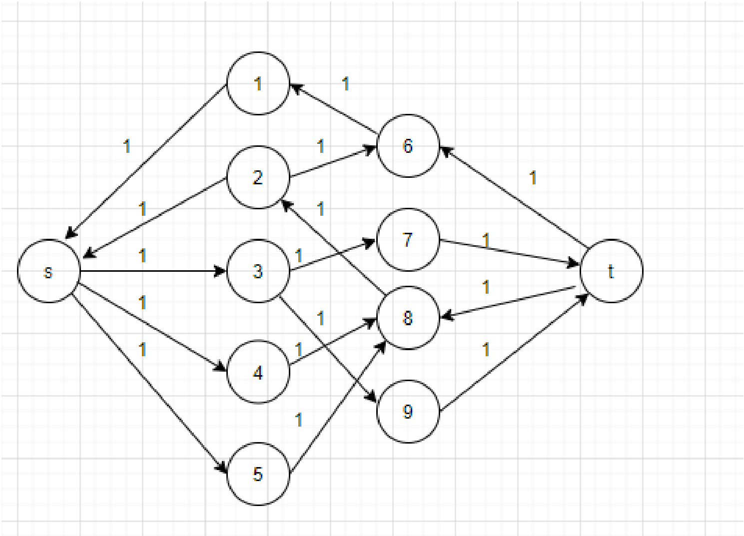
First augmenting path: s->1->6->t



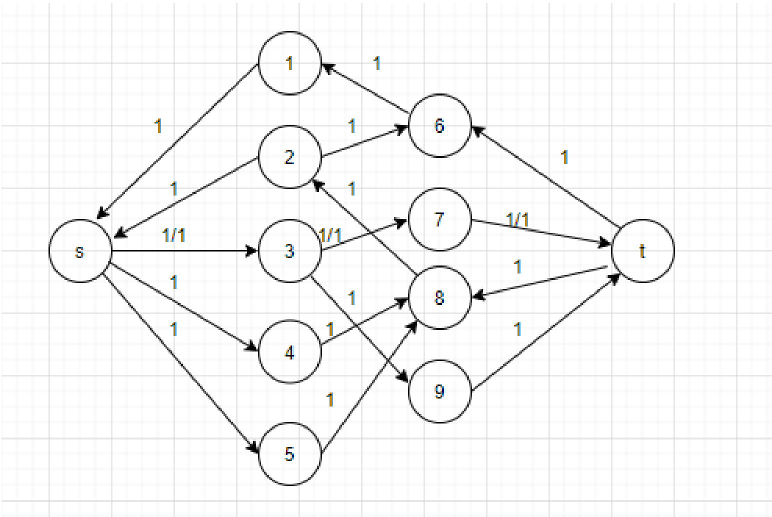


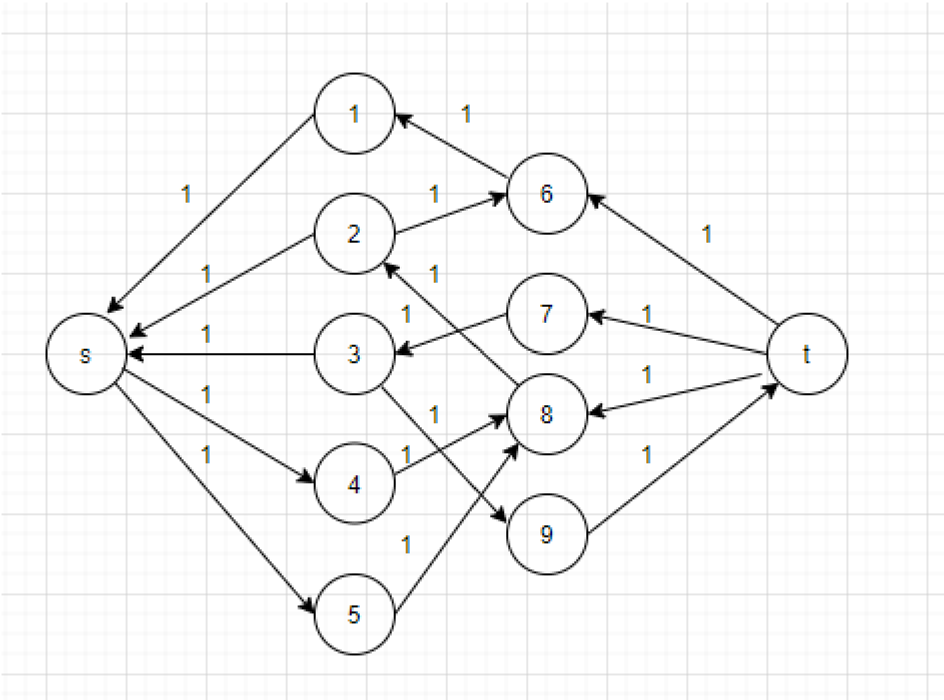
Second augmenting path:s->2->8->t





Third augmenting path: s-3->7->t





cause there is no path from s to 9 after 3 iterations
The final max flow=3