# CS571 - Graph Theory and Algorithms

## Take Home Final Exam

## Yi Ren (002269013)

**1. Show that in every simple graph there is a path from every vertex of odd degree to some other vertex of odd degree**

Suppose simple graph G (V,E), a vertex u$\in$V with odd degree. And u is not isolated since deg(u) > 0.
In the component containing u, there must exist at least two vertices with odd degree, because the number of vertices with odd degree should be even. In other words, there should be at least one odd-degree vertex in the same component with u. In this way, these odd-degree vertices could be connected by different paths.

**2.Let G be a simple graph with n vertices with n ≥ 3 and such that the degree of every vertex in the graph is at least (n - 1)/2.**
**(a) Show that G has a Hamilton circuit when n is even.**
When n is even, (n - 1)/2 will not be an integer.
Hence, the degree of every vertex in the graph should be at least n/2, which is the smallest integer larger than (n - 1)/2.
According to the Dirac's Theorem, G has a Halmilton circuit.

**(b) Does G have a circuit when n is odd? If not show at least one counter example.**
When n is odd, G may not have a Hamilton circuit.
For example, in figure a below, n is 5 and every vertex in G has at least 2 degrees, however, G only have Hamilton path but not circuit.

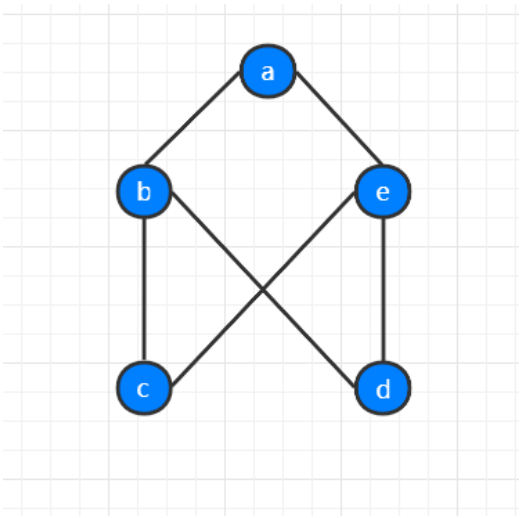
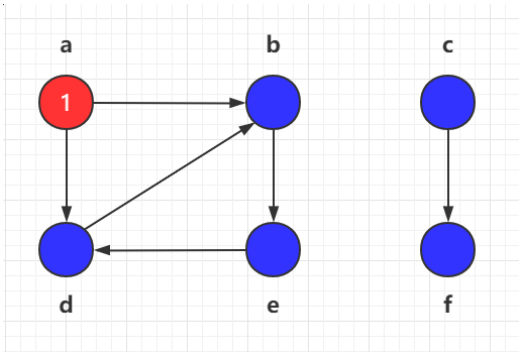
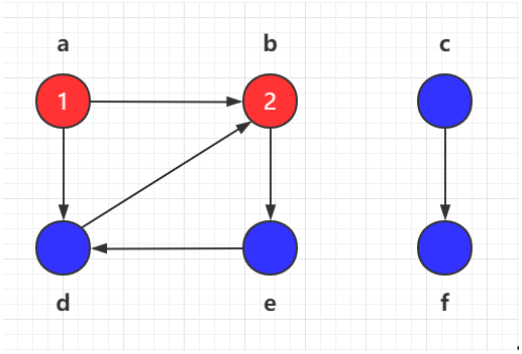


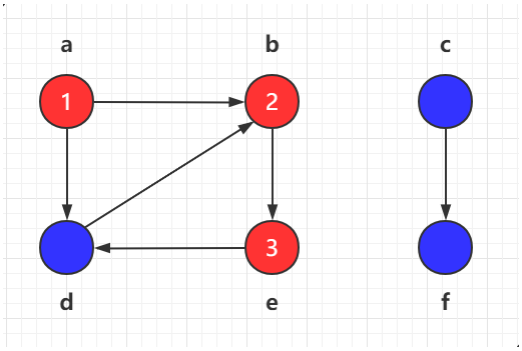
Figure a

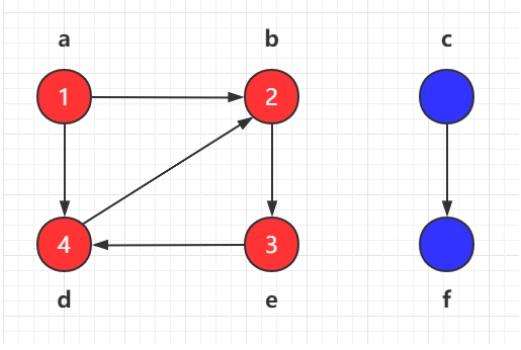
**3. Show how DFS works on the graph of Figure 1. Assume that the for loop of lines 6-10 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex, and show the DFS forest.**

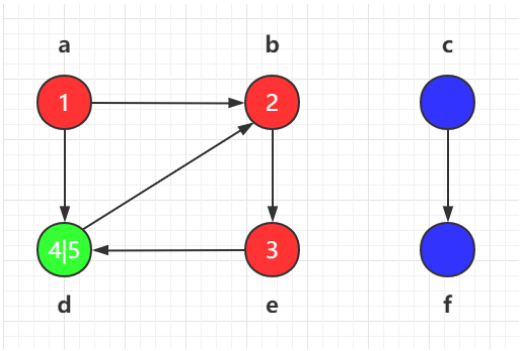
We got two DFS trees in this graph, one tree with root a contains 4 vertices a,b,e and d, the other tree with root c contains the rest vertices of graph.

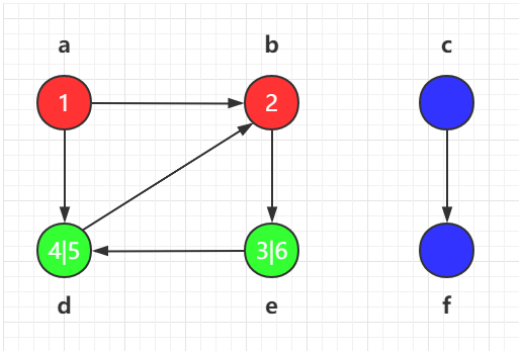


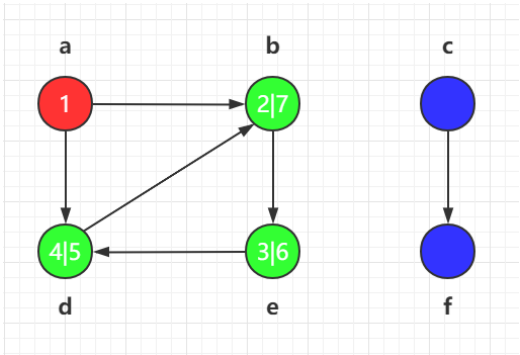



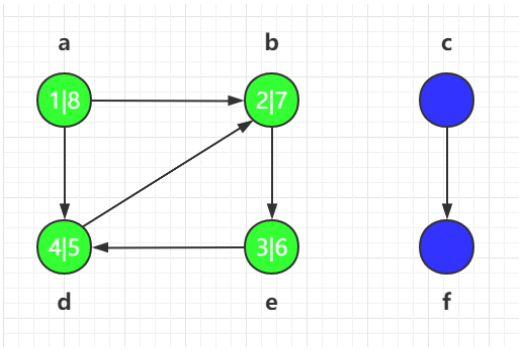



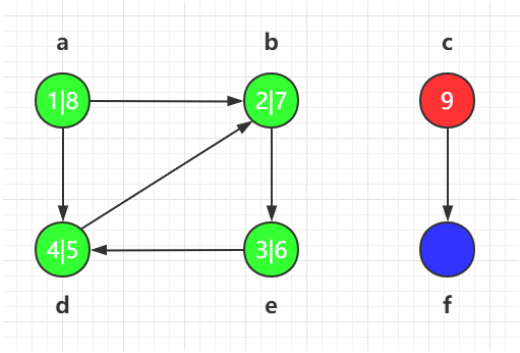



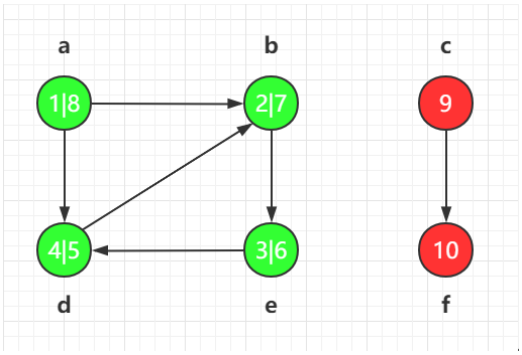

a
b
c
1
2
4|5
3
d
e
f

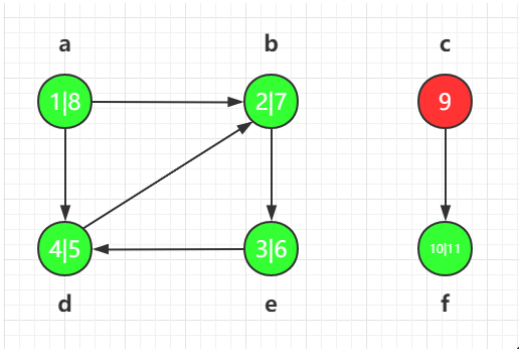

a
b
c
1
2
4|5
3|6
d
e
f

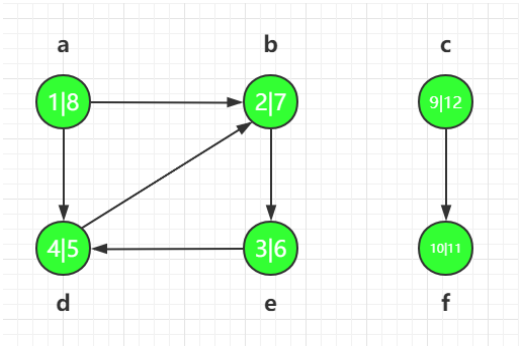

a
b
c
1
2|7
4|5
3|6
d
e
f


a
b
c
1|8
2|7
4|5
3|6
d
e
f


a
b
c
1|8
2|7
9
4|5
3|6
d
e
f




**4. Show that every DAG must have at least one vertex with no outgoing edges and at least one vertex without incoming edges.**

Suppose that DAG G has n vertices(n >= 2), and every vertex in graph G has outgoing edges.

We start to traverse graph G from an arbitrary vertex, define S to be the Set for all of the discovered vertices. When we choose the next vertex v, it should be a vertex out of Set S, to ensure a cycle wouldn't be formed with vertex v selected. Unfortunately, if vertex v is the last undiscovered vertex, it's impossible to choose a vertex out of Set S, which means a cycle will be formed with an outgoing edge of the last vertex. Hence, every DAG must have at least one vertex with No outgoing edges.

For the same reason(we need to change the traverse direction), if every vertex in graph G has incoming edges, a cycle must be formed. So every DAG must have at least one vertex with No incoming edges.

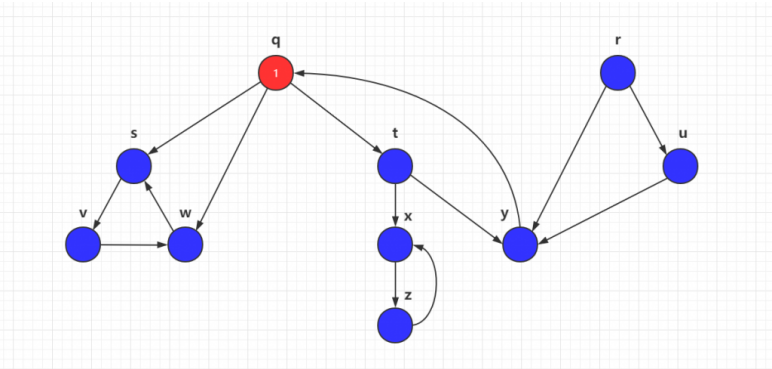
**5. Another way to perform topological sorting on a DAG G = (V, E) is to repeatedly find vertices of in-degree 0 (see previous question) and store them in a list (or queue) Lin, and as long as Lin is not empty, remove a vertex from Lin and add it to the front of a list Lout, and then remove it and all of its outgoing edges from the graph. Then, insert into Lin any vertex without incoming edges in the obtained subgraph. The list Lout will store all the vertices of G in a topological sorted order. Write the pseudocode of the algorithm that implements this idea and analyze its computational cost.**

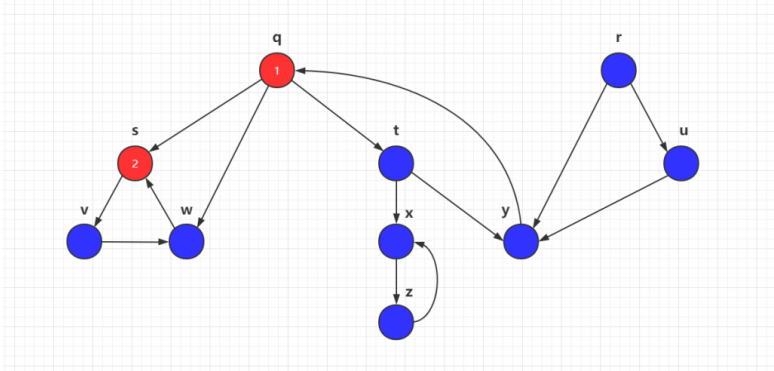
```
Graph G = (V,E);
Queue Q1 = null ,Q2 = null;     //define Q1 to be in-queue, Q2 to be out-queue
Set S1 = null;          //define S1 to be a Vertex set with in-degree 0

while(V != null){
    S1 = get in-degree 0 from V;
    ENQUEUE(Q1, S1);

    while(Q1 != null){
        v = DEQUEUE(Q1);
        ENQUEUE(Q2, v);
        REMOVE(v);      //remove vertex v and its outgoing edges
    }
}

return Q2;
```

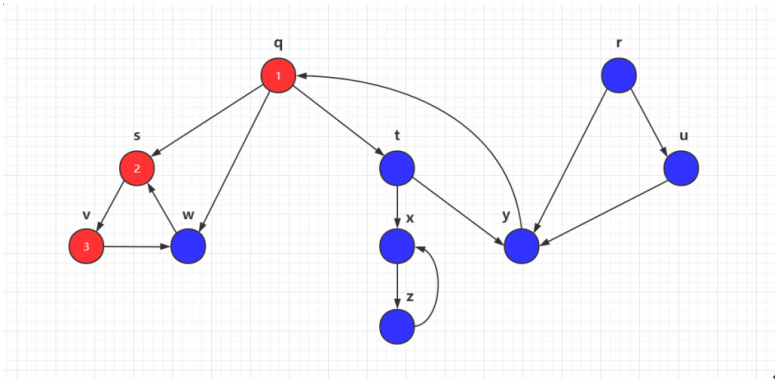
The computational cost:
SPACE: 4|V| + |E|
We defined 4 variables V,Q1,Q2 and S1, each of them need |V| space. Also the edge set of graph G needs |E| space.

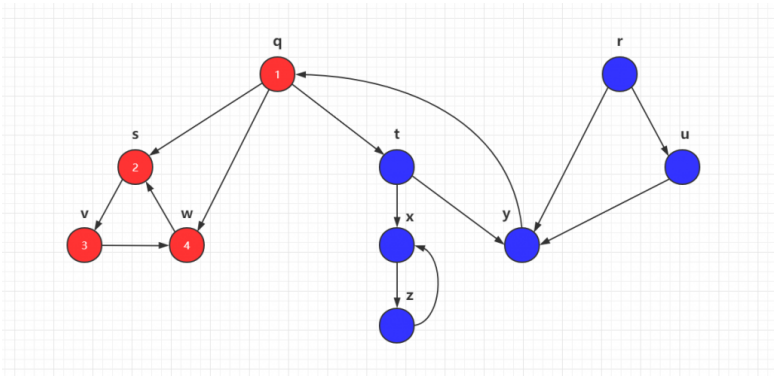
TIME: $\theta$(|E||V| + |V|)
In the outer while loop, every iteration will find the vertices of in-degree 0， which need to check all the edges of E. In this way, we need |V|* |E| time for degree checking.
Then, we need to do ENQUEUE, DEQUEUE and REMOVE for every vertex, so these operations will need 3|V| time.

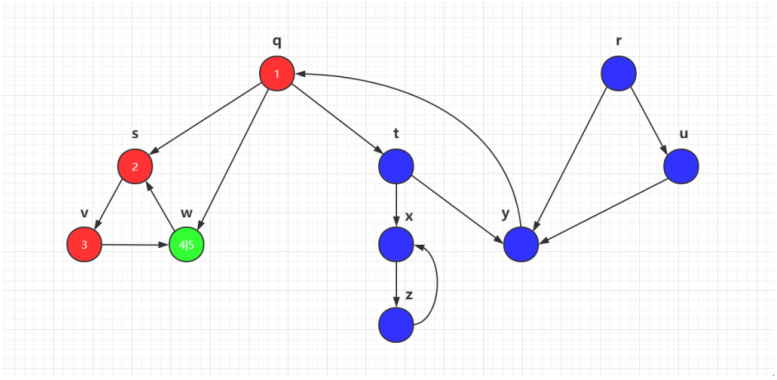
**Show how the procedure STRONGLY-CONNECTED-COMPONENTS works on the graph of Figure 2. Specifically, show the finishing times computed in line 1 and the forest produced in line 3. Assume that the loop of lines 6-10 of DFS considers vertices in alphabetical order and that the adjacency lists are in alphabetical order.**
The finishing time is labeled on the vertices, and we got two trees in this graph. One with root q contains 8 vertices q,s,t,v,w,x,y and z, the other tree with root r contains the rest vertices of graph.

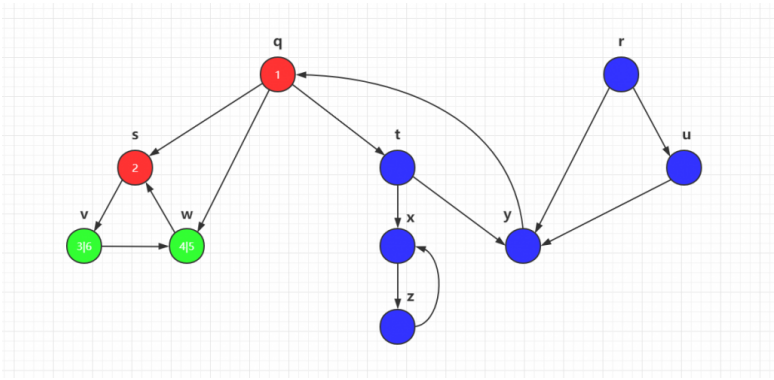


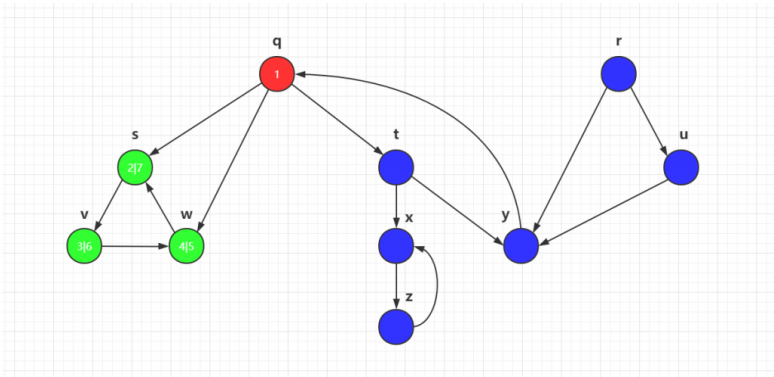

q
1
r
s
2
t
u
v
w
x
y
z

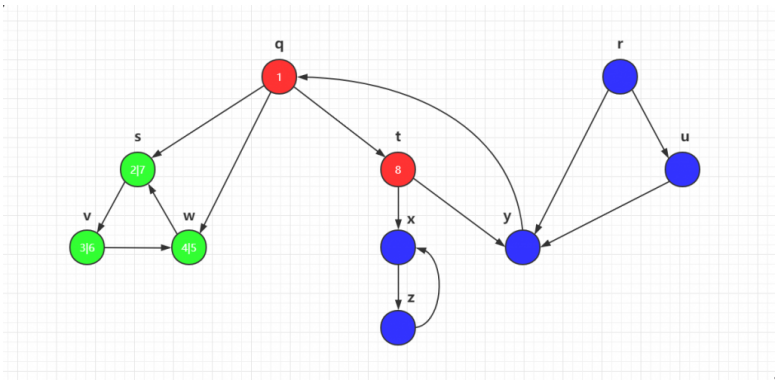

q
1
r
s
2
t
u
v
3
w
x
y
z

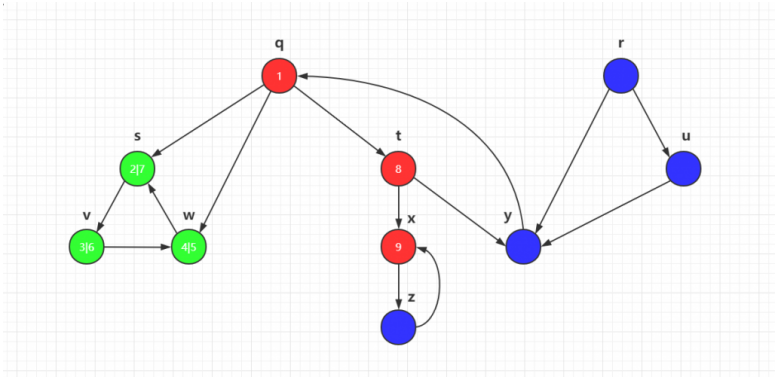

q
1
r
s
2
t
u
v
3
w
4
x
y
z

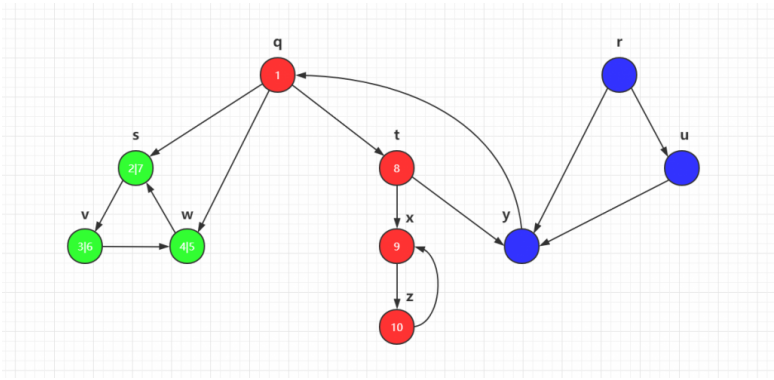

q
1
r
s
2
t
u
v
3
w
4|5
x
y
z

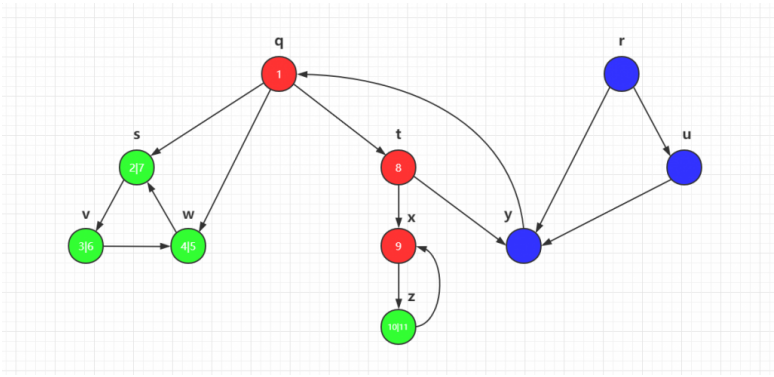

q
1
r
s
2
t
u
v
w
3|6
4|5
x
y
z

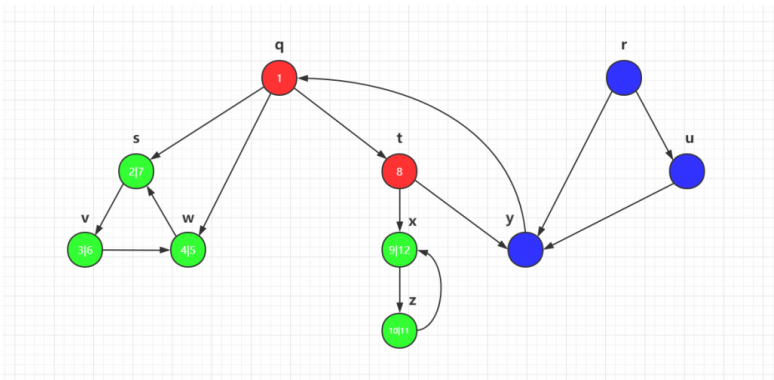

q
1
r
s
2|7
t
u
v
w
3|6
4|5
x
y
z

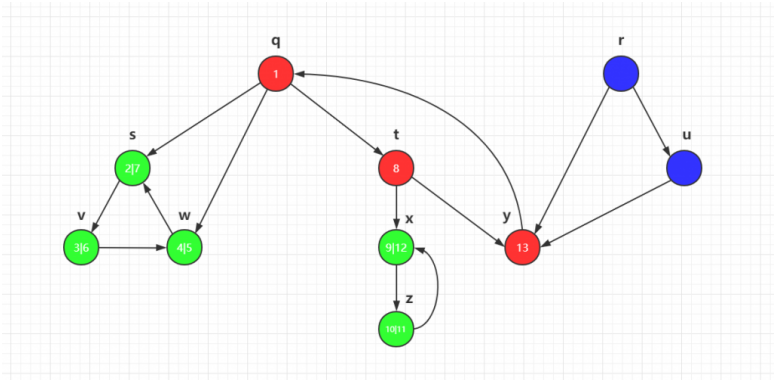

q
1
r
s
2|7
t
8
u
v
w
3|6
4|5
x
y
z

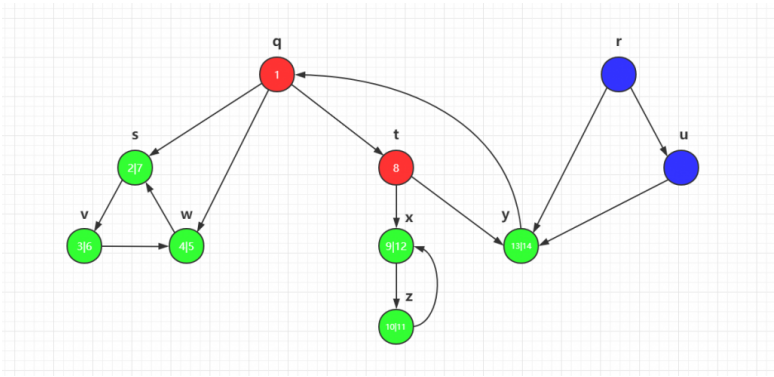

q
1
r
s
2|7
t
8
u
v
w
3|6
4|5
x
9
y
z

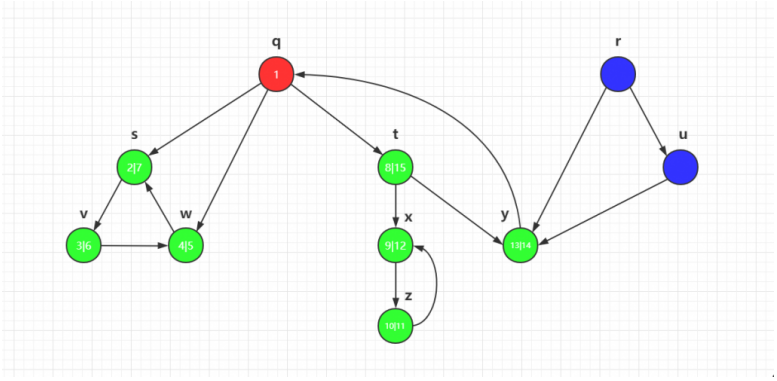

q
r
1
s
t
u
2|7
8
v
w
x
y
3|6
4|5
9
z
10

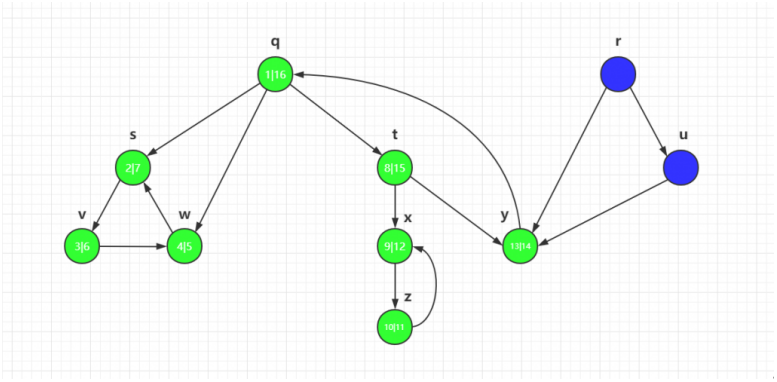

q
r
1
s
t
u
2|7
8
v
w
x
y
3|6
4|5
9
z
10|11

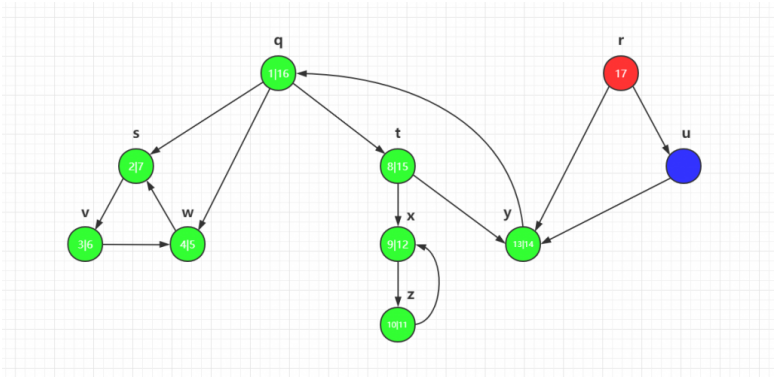

q
r
1
s
t
u
2|7
8
v
w
x
y
3|6
4|5
9|12
z
10|11

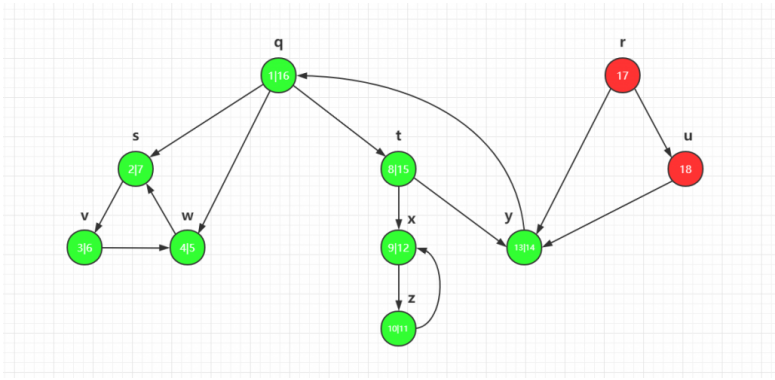

q
r
1
s
t
u
2|7
8
v
w
x
y
3|6
4|5
9|12
13
z
10|11

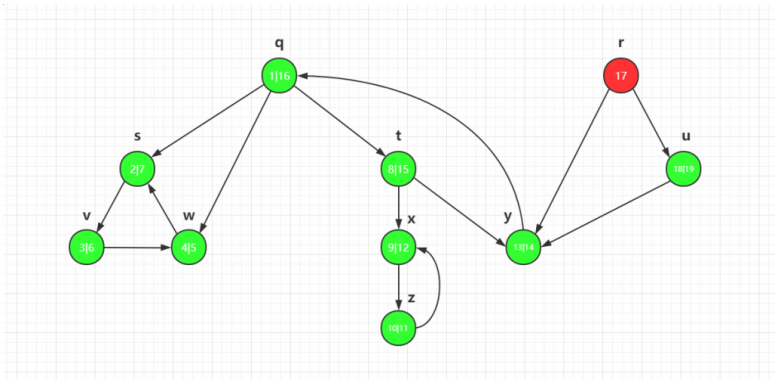

q
r
1
s
t
u
2|7
8
v
w
x
y
3|6
4|5
9|12
13|14
z
10|11

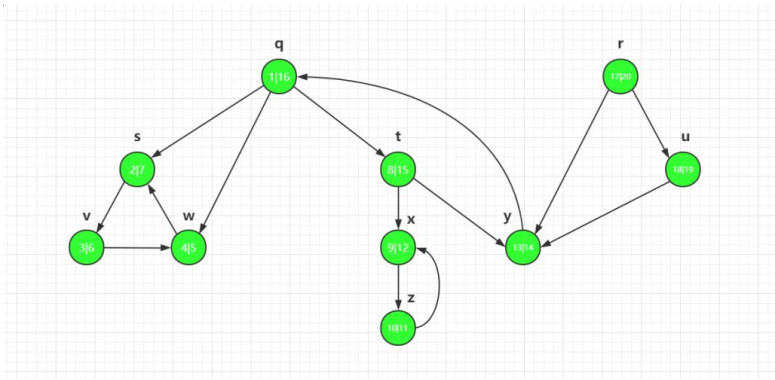

q
r
1
s
t
u
2|7
8|15
v
w
x
y
3|6
4|5
9|12
13|14
z
10|11

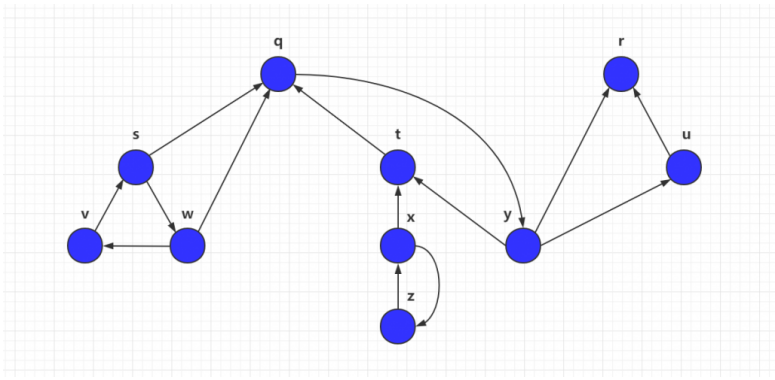

q
r
1|16
s
t
u
2|7
8|15
v
w
x
y
3|6
4|5
9|12
13|14
z
10|11

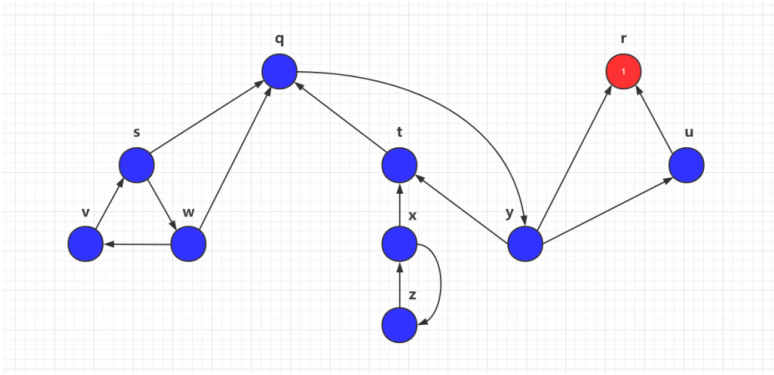

q
r
1|16
17
s
t
u
2|7
8|15
v
w
x
y
3|6
4|5
9|12
13|14
z
10|11

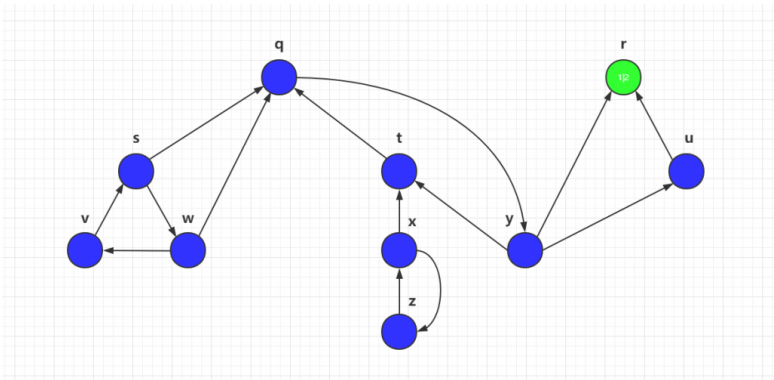

q
1|16
r
17
s
2|7
t
8|15
u
18
v
3|6
w
4|5
x
9|12
y
13|14
z
10|11

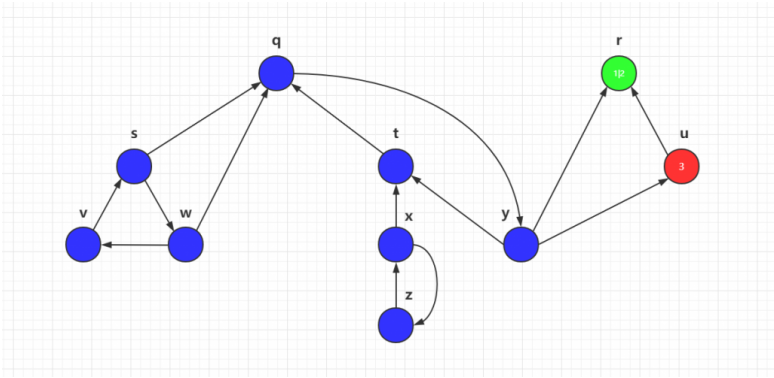

q
1|16
r
17
s
2|7
t
8|15
u
18|19
v
3|6
w
4|5
x
9|12
y
13|14
z
10|11

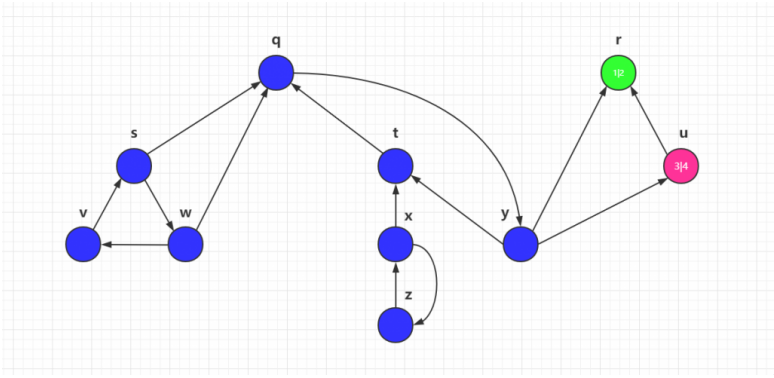

q
1|16
r
17|20
s
2|7
t
8|15
u
18|19
v
3|6
w
4|5
x
9|12
y
13|14
z
10|11

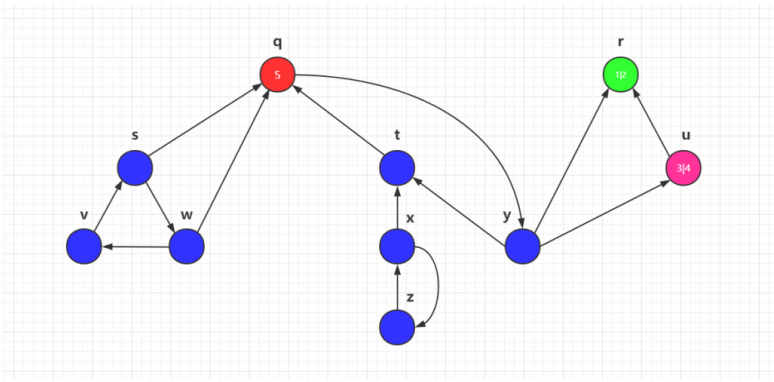

q
r
s
t
u
v
w
x
y
z

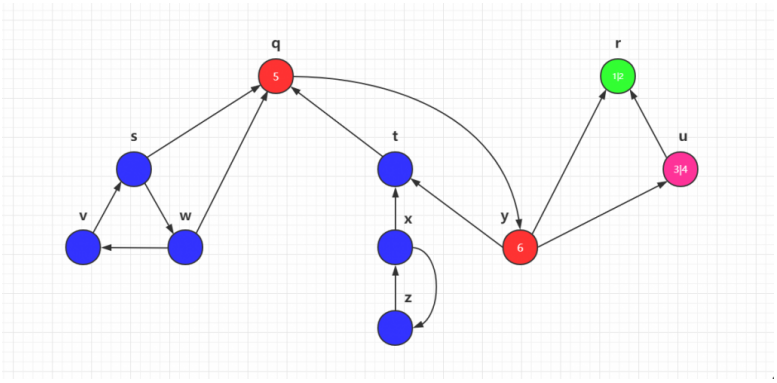

q
r
1
s
t
u
v
w
x
y
z

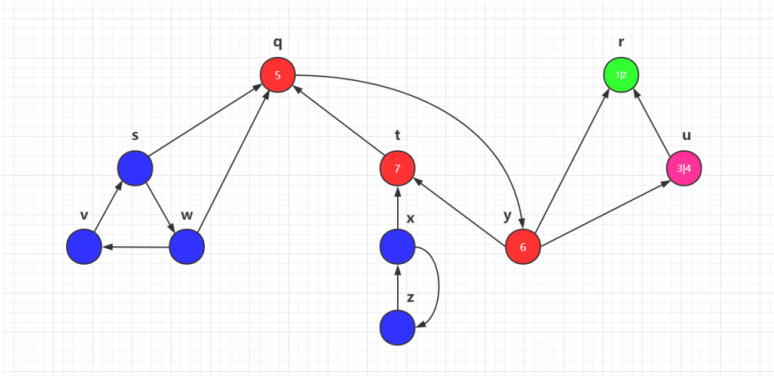

q
r
1|2
s
t
u
v
w
x
y
z

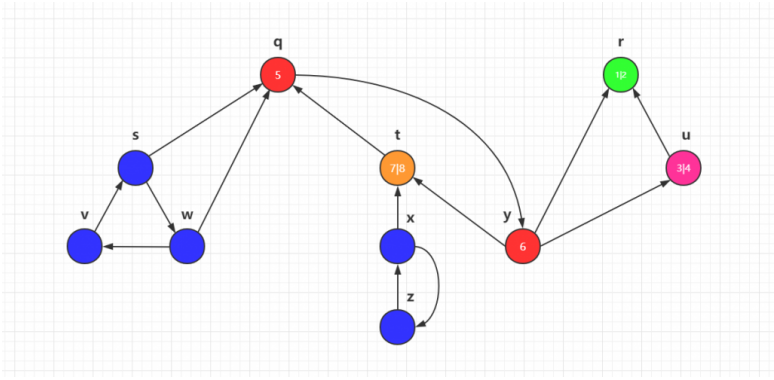

q
r
1|2
s
t
u
3
v
w
x
y
z

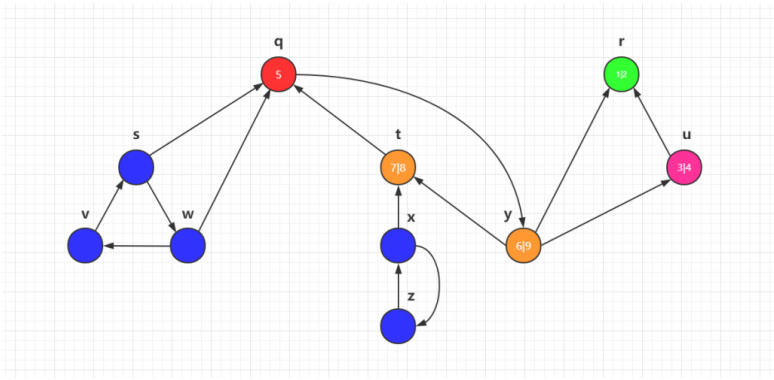

q
r
1|2
s
t
u
3|4
v
w
x
y
z

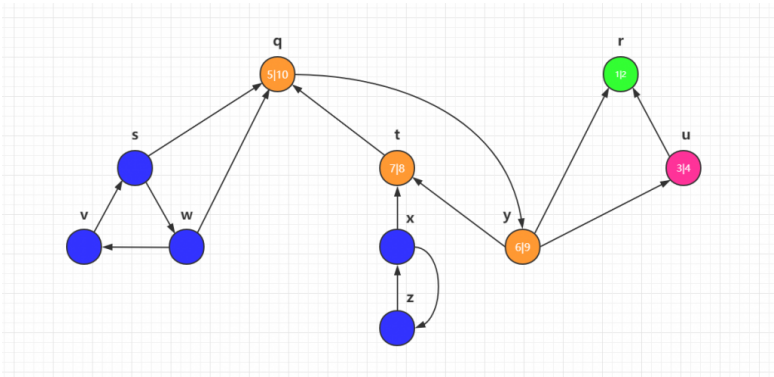

q
5
r
12
s
t
u
3|4
v
w
x
y
z

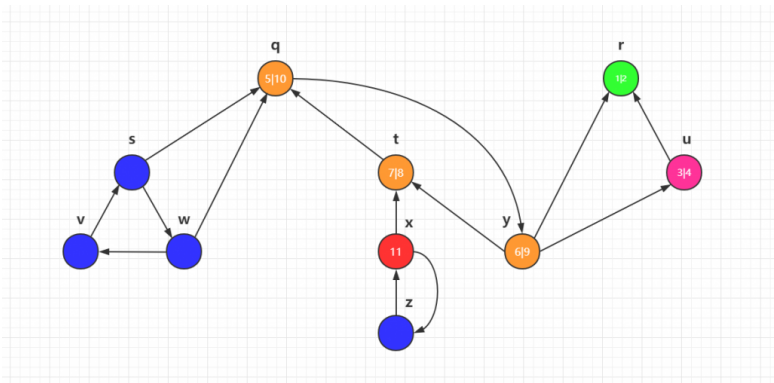

q
5
r
12
s
t
u
3|4
v
w
x
y
6
z

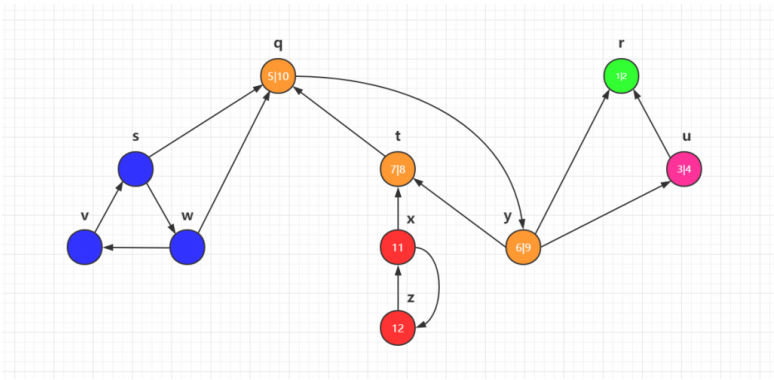

q
5
r
12
s
t
u
3|4
7
v
w
x
y
6
z

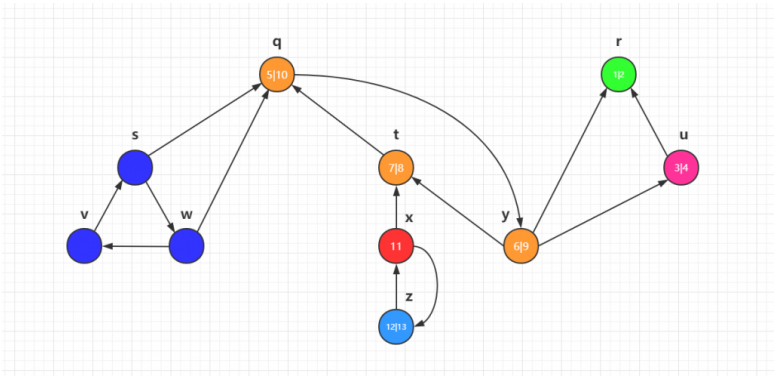

q
5
r
12
s
t
u
3|4
7|8
v
w
x
y
6
z

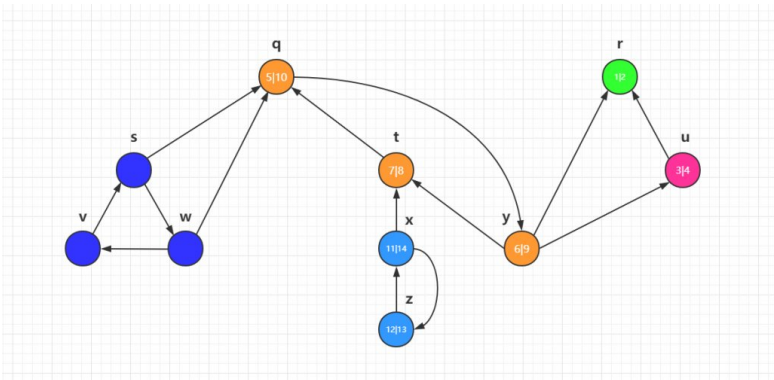

q
5
r
12
s
t
7|8
u
3|4
v
w
x
y
6|9
z

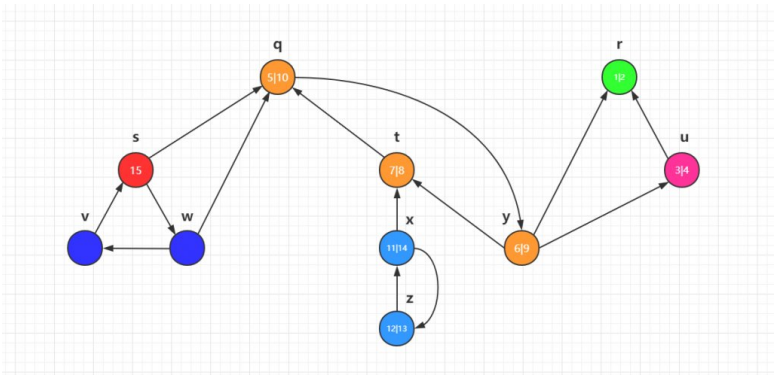

q
5|10
r
12
s
t
7|8
u
3|4
v
w
x
y
6|9
z

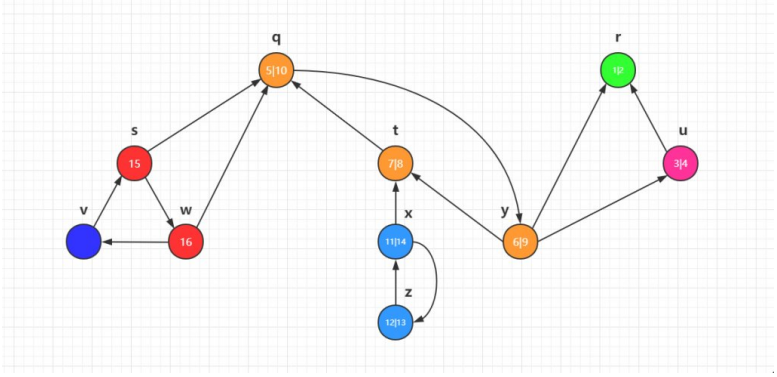

q
5|10
r
12
s
t
7|8
u
3|4
v
w
x
11
y
6|9
z

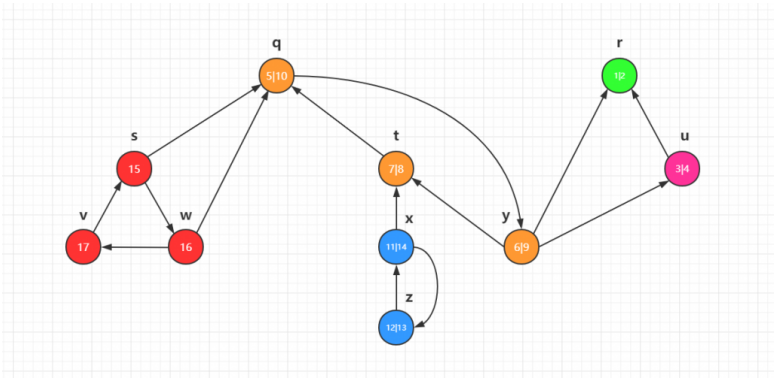

q
5|10
r
12
s
t
7|8
u
3|4
v
w
x
11
y
6|9
z
12

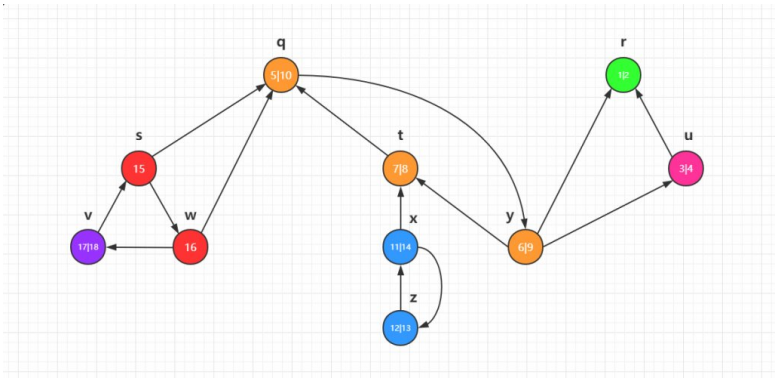

q
5|10
r
1|2
s
t
u
7|8
3|4
v
w
x
y
11
6|9
z
12|13

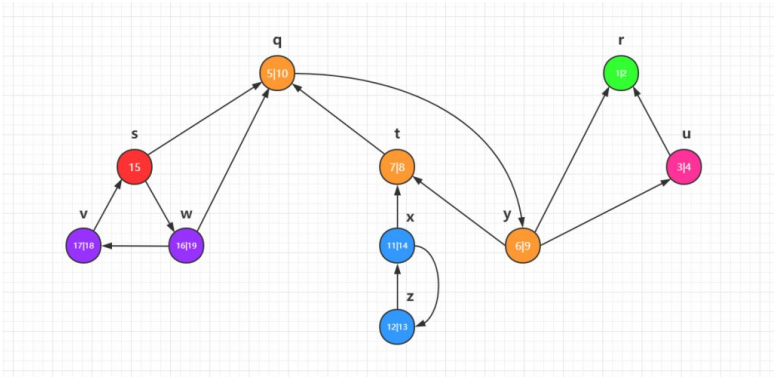

q
5|10
r
1|2
s
t
u
7|8
3|4
v
w
x
y
11|14
6|9
z
12|13

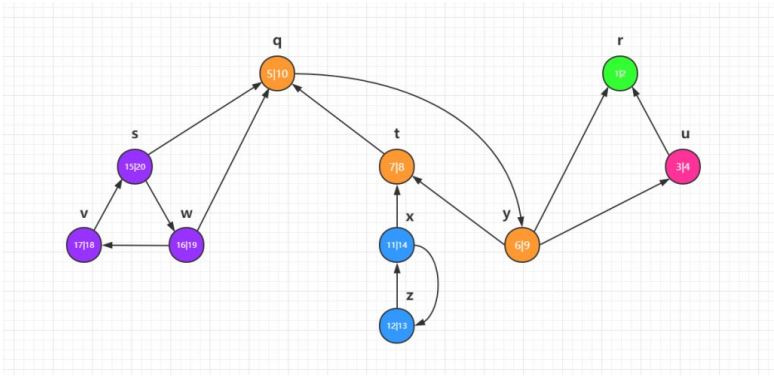

q
5|10
r
1|2
s
15
t
u
7|8
3|4
v
w
x
y
11|14
6|9
z
12|13


q
5|10
r
1|2
s
15
t
u
7|8
3|4
v
w
16
x
y
11|14
6|9
z
12|13


q
5|10
r
1|2
s
15
t
7|8
u
3|4
v
17
w
16
x
11|14
y
6|9
z
12|13


q
5|10
r
1|2
s
15
t
7|8
u
3|4
v
17|18
w
16
x
11|14
y
6|9
z
12|13


q
5|10
r
1|2
s
15
t
7|8
u
3|4
v
17|18
w
16|19
x
11|14
y
6|9
z
12|13


q
5|10
r
1|2
s
15|20
t
7|8
u
3|4
v
17|18
w
16|19
x
11|14
y
6|9
z
12|13