图算法介绍

1. 基础概念

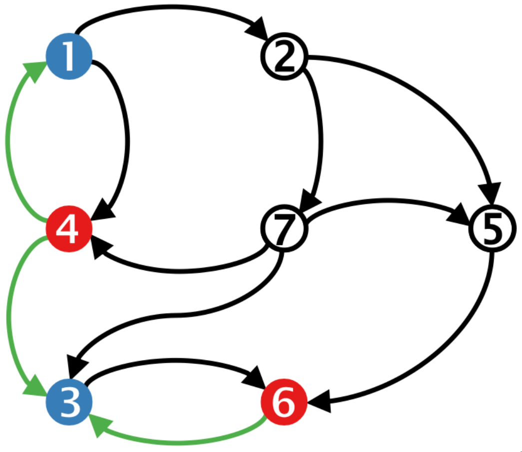
https://github.com/LongLee220/Graph-algorithm

基础知识

复杂度

空间复杂度：算法执行完所需要的存储空间

例如图G=(V,E)，|V|=10000,|E|=100000



存储矩阵A:

$$
\begin{vmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0
\end{vmatrix}
$$

边列表:

$$
\begin{array}{c}
\begin{array}{cccccccccccc}
e_{12} & e_{14} & e_{25} & e_{27} & e_{36} & e_{41} & e_{43} & e_{56} & e_{63} & e_{73} & e_{74} & e_{75}
\end{array} \\
\begin{vmatrix}
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{vmatrix}
\end{array}
$$

时间复杂度：$\lim \frac{T(n)}{O(n)} = C$

例 sum$[a_0, a_1, \dots, a_{n-1}]$与var$[a_0, a_1, \dots, a_{n-1}]$

$$s = \mathrm{a}_0 + a_1 + \cdots + a_{n-1} \qquad m = \frac{s}{n}$$

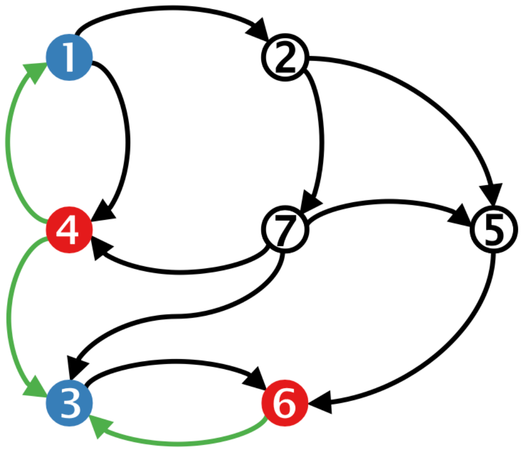
$$v = \frac{(a_0-m)^2+(a_1-m)^2+\cdots+(a_{n-1}-m)^2)}{n}$$

目标

存储与效率

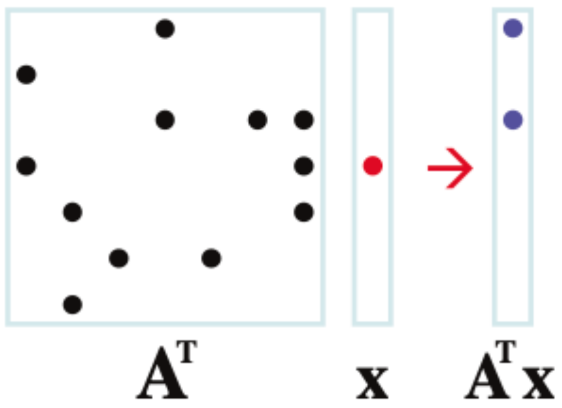
基础知识

广度优先搜索（Breadth-First-Search，BFS）

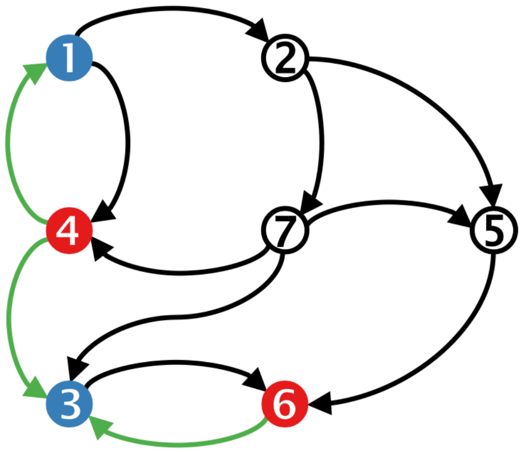


l=0	4
l=1	1,3
l=2	2,6
l=3	5,7

$$y = A^T * x$$



深度优先搜索（Deep-First-Search，DFS）



l=0	4
l=1	1
l=2	2
l=3	5
l=4	6
l=5	3
l=6	7

Algebra Breadth-First-Search

```
Input: Digraph G
Output: Node set of v_i
A = Adjacent(G)  #构造初始矩阵
BFS_set = [] #存储所有BFS节点
v_i = [0, … , 0,1,0, … 0]_{n×1}
A^T[i] = 0
while v != 0 do:    # 此判断条件针对非联通,
    v = A^T * v   #矩阵与向量相乘
    for i in range(n) and i ∉ BFS_set do:
        if v[i] != 0 do:#判断哪些位置有新的邻居
            BFS_set.append(i)      #添加新邻居
            A[i] = 0
            v[i] = 1
```

Step =1

$$\begin{bmatrix}1\\0\\1\\0\\0\\0\\0\end{bmatrix} \quad \begin{bmatrix}0&0&0&0&0&0&0\\1&0&0&0&0&0&0\\0&0&0&0&0&0&0\\1&0&0&0&0&0&1\\0&1&0&0&0&0&1\\0&0&1&0&1&0&0\\0&1&0&0&0&0&0\end{bmatrix}$$

Step =2

$$\begin{bmatrix}0\\1\\0\\0\\0\\1\\0\end{bmatrix} \quad \begin{bmatrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&1&0&0&0&0&1\\0&0&0&0&0&0&0\\0&1&0&0&0&0&0\end{bmatrix}$$

Step =3

…

Algebra Breadth-First-Search

```
Input: Digraph G
Output: Node set of v_i
A = Adjacent(G)  #构造初始矩阵
BFS_set = [] #存储所有BFS节点
v_i = [0, … , 0,1,0, … 0]_{n×1}
A^T[i] = 0
while v != 0 do:    # 此判断条件针对非联通,
    v = A^T * v   #矩阵与向量相乘
    for i in range(n) and i ∉ BFS_set do:
        if v[i] != 0 do:#判断哪些位置有新的邻居
            BFS_set.append(i)      #添加新邻居
            A[i] = 0
            v[i] = 1
```


$$A^T = \begin{bmatrix}0&0&0&1&0&0&0\\1&0&0&0&0&0&0\\0&0&0&1&0&1&1\\0&0&0&0&0&0&0\\0&1&0&0&0&0&1\\0&0&1&0&1&0&0\\0&1&0&0&0&0&0\end{bmatrix} \quad v_4 = \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\end{bmatrix}$$

图算法介绍

1．基础概念

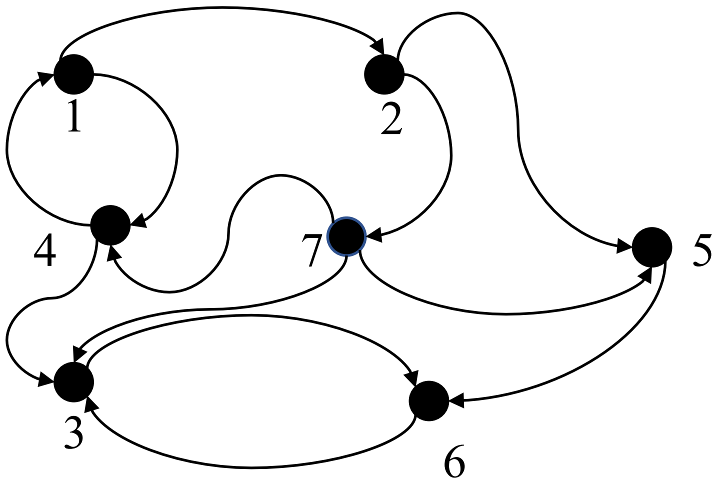
2．强连通分量

强连通分量

Kosaraju （$O(m+n)$）

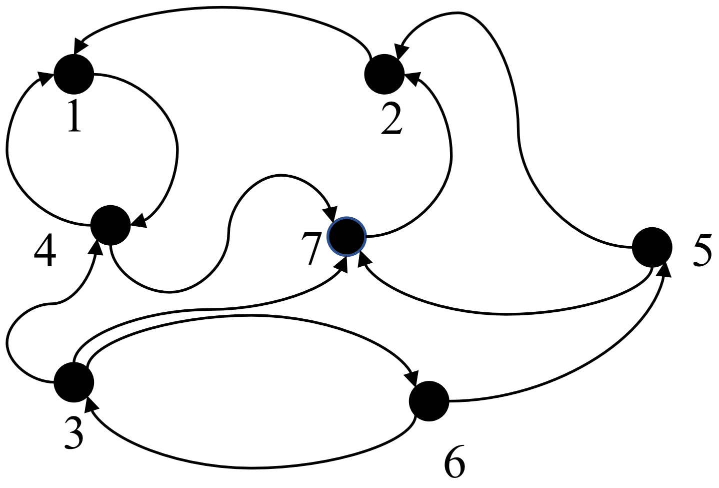
Kosaraju Algorithm (G, s, t)

```
1.  Input: G
2.  Output: Component(G)
3.  Call DFS(G) to compute finishing times f[u] for v ∈ V
4.  Compute G^T
5.  Call DFS(G^T).
6.  but in the main loop of DFS, consider the decreasing
7.  Output the vertices of
```



G:

$1 \to 2 \to 5 \to 6 \to 3$
$\to 7 \to 4$



G^T:

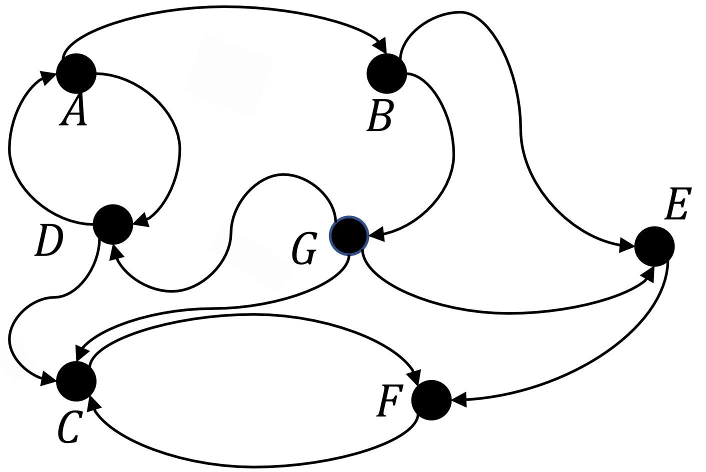
$4 \to 1$
$\to 7 \to 2$

Connected _Component_1 = [1,2,4,7]

强连通分量

tarjan （$O(m+n)$）

```
tarjan Algorithm (G, s, t)
Input: G
Output: Component(G)
DFN[u] = Low[u] = ++Index
    Stack.push(u)
    for each (u, v) in E do:
        if (v is not visted) do:
            tarjan(v)
            Low[u] = min(Low[u], Low[v])
        else if (v in S) do:
            Low[u] = min(Low[u], DFN[v])
    if (DFN[u] == Low[u]) do:
        repeat
            v = S.pop
            print v
        until (u== v)
```



1.

+ node	dfn	low	Stack
A	1	1	A
B	2	2	A,B
E	3	3	A,B,E
F	4	4	A,B,E,F
C	5	5	A,B,E,F,C

2.

- node	dfn	low	Stack
C	4	4	A,B,E,F
F	3	3	A,B,E
E	2	2	A,B

3.

+ node	dfn	low	Stack
G	6	6	A,B,G
D	7	1	A,B,G,D

4.

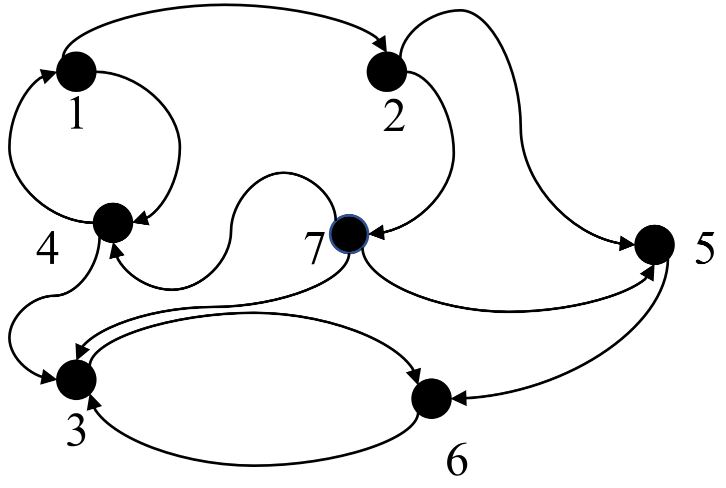
M_node	dfn	low	Stack
G	6	1	A,B,G,D
B	2	1	A,B,G,D

强连通分量

Algebra （$O(m+n)$）

```
Algebra Algorithm (G, s, t)

Input: G
Output: Component(G)
V_list = [1, 2, …, n]
Cmp_list = [ ]
while V_list != Ø do:
        node = V_list[0]
        cmp_node = [ ]
        C = Algebra_BFS(G, node)
        D = Algebra_BFS(G^T, node)
        cmp_node.append(C∩D)
        Cmp_node.append(cmp_node)
        V_list.remove(cmp)
```



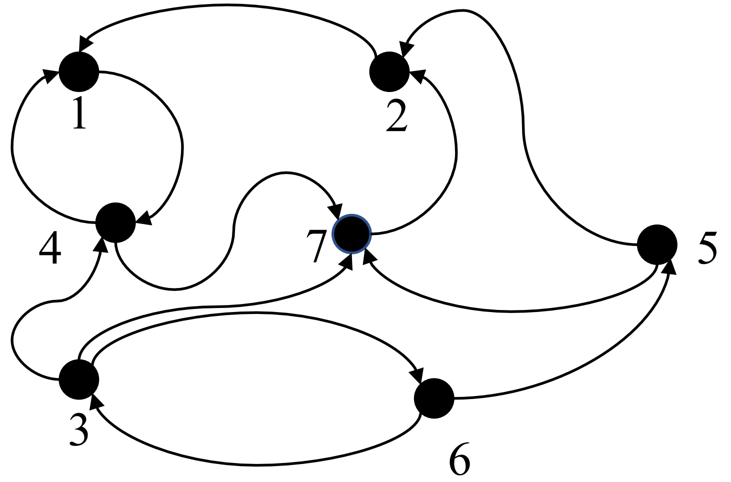
G:

1 →2,4

→5,7,3

→6

C = [1, 2, 4, 5, 7, 3, 6]



G^T:

1 →4

→7

→2

D = [1, 4, 7, 2]

Connected _Component_1 = [1,2,4,7]

图算法介绍

1. 基础概念

2. 强连通分量

3. **最短路算法**

最短路

Dijkstra Algorithm （有向+非负权） 1957 $O(n^2)$

Dijkstra Algorithm (G, w, s)

```
Initialize-Single-Source
S = Ø
Q = V
while Q != Ø do:
    U = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u] do:
        RELAX(u, v, w)
```

Initialize-Single-Source

for each $v \in V$ **do:**

$d(v) = \infty$

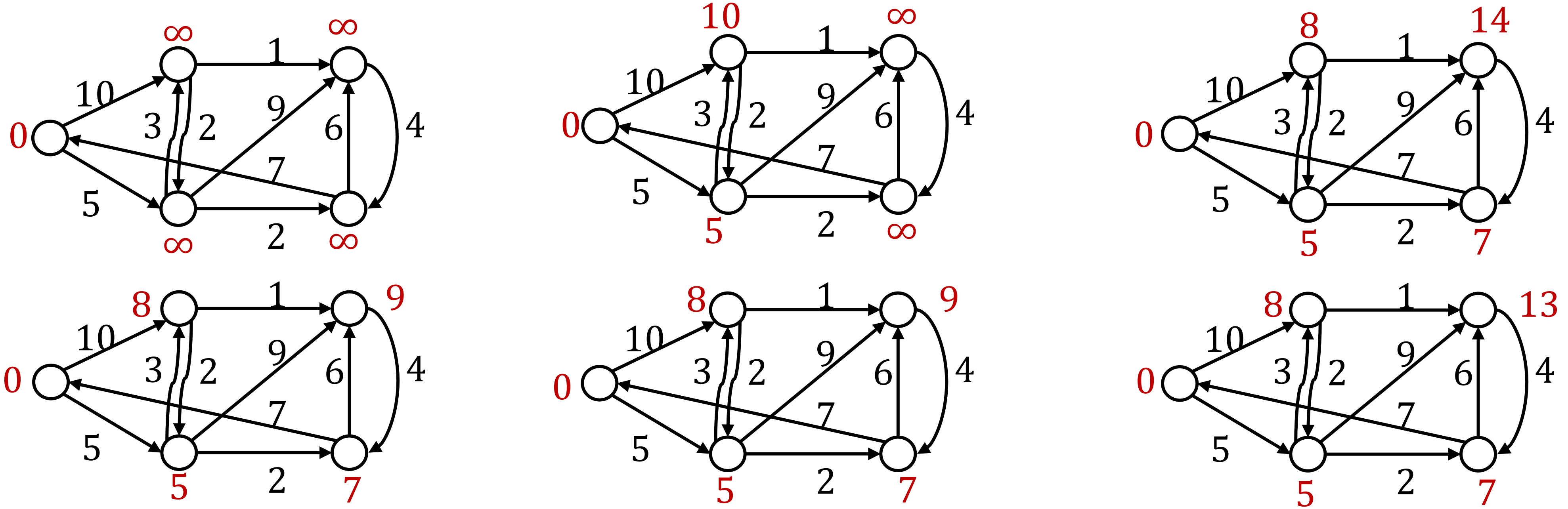
$d(s) = 0$

RELAX(u, v, w)

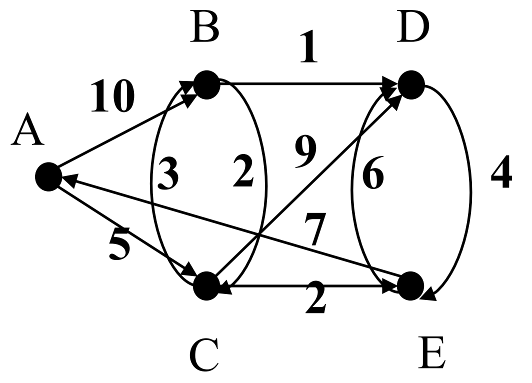
if $p(s,v) > p(s,u) + w(u,v)$ **then:**

$p(s,v) = p(s,u) + w(u,v)$

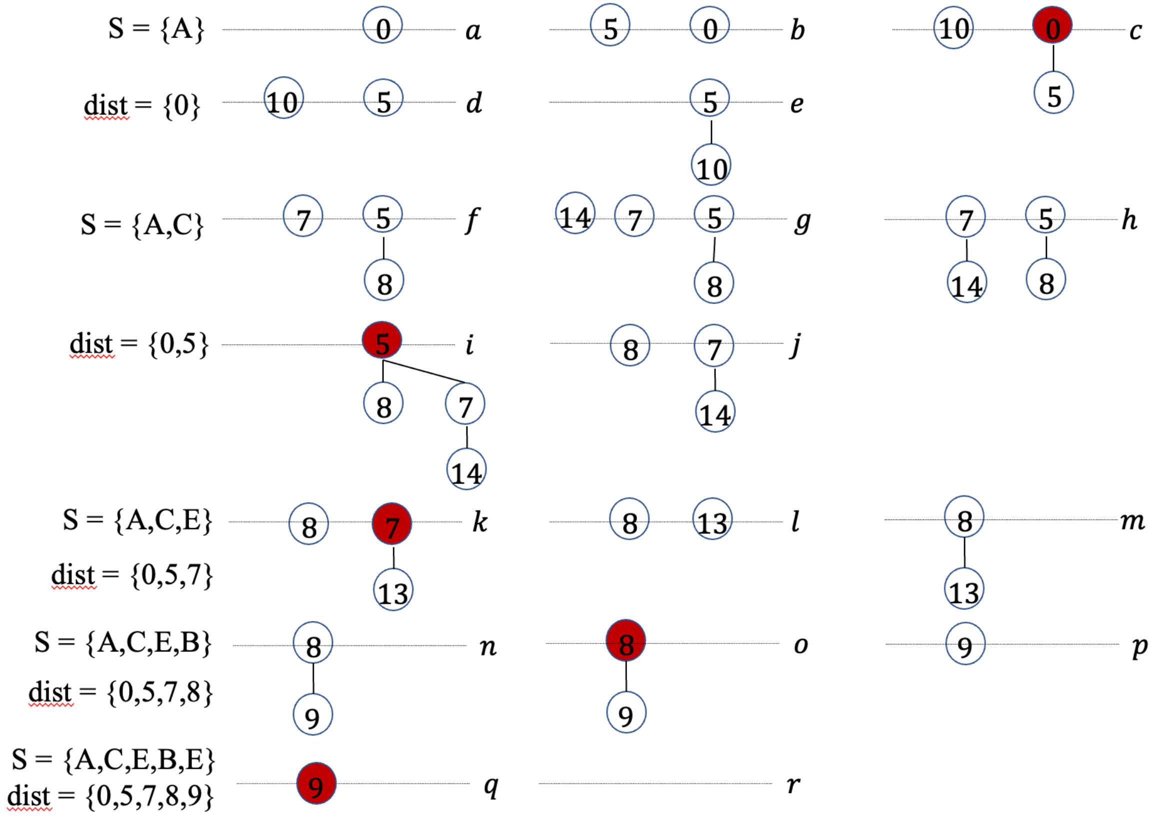
$v.\pi = u$



Dijkstra Algorithm （有向+非负权）1987 $O(n + m\log m)$




```
Fibonacci Heaps Dijkstra Algorithm (G, w, s)
Initialize-Single-Source
Make Heap
S = ∅
Q = V
while Heap != ∅ do:
    U = root
    S = S ∪ {u}
    for each vertex v ∈ Q − S, Adj[u] do:
        if p(s,v) > p(s,u) + w(u,v) then:
            decrease node v or insert v
    delete u in Heap
```



步骤：从a-b的堆示意图

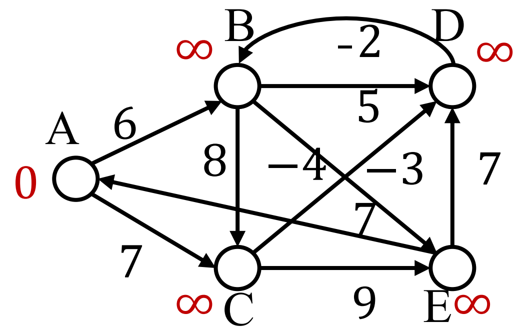
Bellman-Ford Algorithm （有向+正负权）1962 $O(nm)$

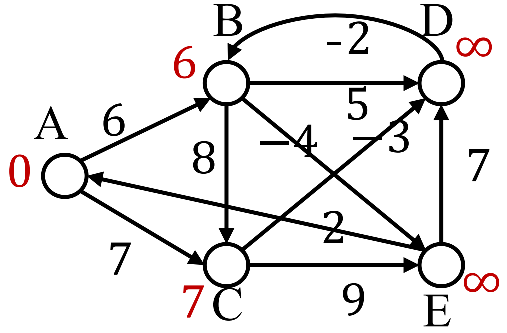
Bellman-Ford(G, w, s)

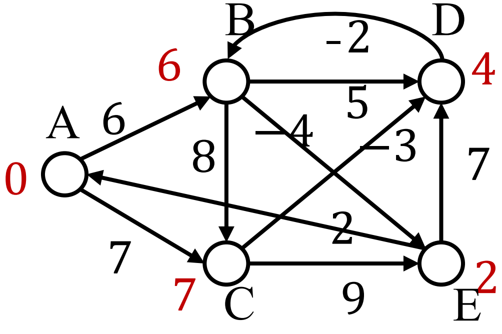
Initialize-Single-Source
for i from 1 to |V|-1 **do:**
    for each edge(u, v) in |E| **do:**
        RELAX(u, v, w)
for each edge(u, v) in **|E|** **do:**
    if $p(s,v) > p(s,u) + w(u,v)$ **do:**
        return False

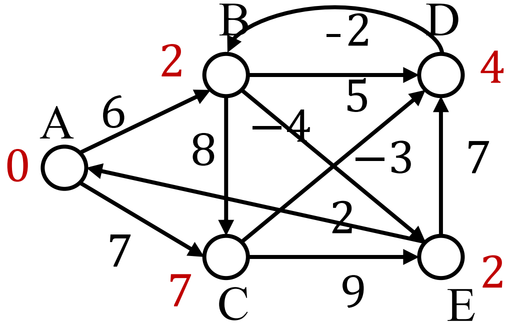
RELAX(u, v, w)

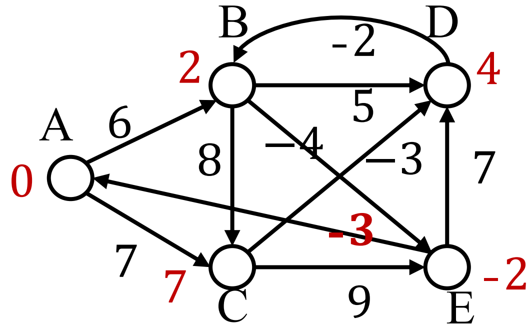
if $p(s,v) > p(s,u) + w(u,v)$ **do:**
    $p(s,v) = p(s,u) + w(u,v)$
    $v.\pi = u$











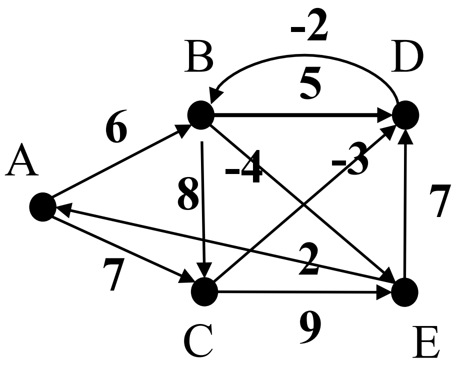
Bellman-Ford Algorithm （有向+正负权）2014

```
Bellman-Ford(G, w, s)
```

```
Initialize-Single-Source
for i from 1 to |V|-1 do:
    d = d min.+ A
if d != d min.+ A do:
    return “A negative-weight cycle exists.”
```

$$d_k(v) = \min_{\forall u \in N}(d_{k-1}(u) + A(u,v))$$

0	7	6	--	--
--	0	--	-3	9
--	8	0	5	-4
--	--	-2	0	--
2	--	--	7	0



0
∞
∞
∞
∞

d_0

0	7	6	--	--
--	0	--	-3	9
--	8	0	5	-4
--	--	-2	0	--
2	--	--	7	0

Initial	Step_1	Step_2	Step_3	Step_4
0	0	0	0	0
∞	7	7	7	7
∞	6	6	2	2
∞	∞	4	4	4
∞	∞	2	2	-2
d_0	d_1	d_2	d_3	d_4

最短路

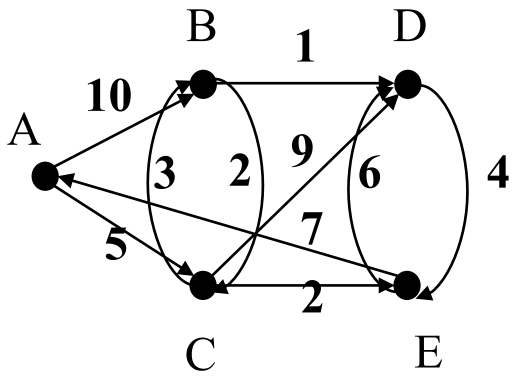
Floyd-Warshall（有向+正权） 1962 $\boldsymbol{O(n^3)}$

Floyd-Warshall Algorithm (G, w, s)

```
Input: A matrix
Output: D matrix
for all i != j do:
        d_ij = A_ij
for i from 1 to n do:
        d_ii = 0
for k from 1 to n do:
        for i from 1 to n do:
                for j from 1 to n do:
                        d_ij = min{d_ij, d_ik + d_kj}
```

$$A \begin{vmatrix} 0 & 10 & 5 & \infty & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & 3 & 0 & 9 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & 17 & 12 & 6 & 0 \end{vmatrix} \quad B \begin{vmatrix} 0 & 10 & 5 & 11 & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & 3 & 0 & 4 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & 17 & 12 & 6 & 0 \end{vmatrix}$$

$$C \begin{vmatrix} 0 & 8 & 5 & 9 & 7 \\ \infty & 0 & 2 & 1 & 4 \\ \infty & 3 & 0 & 4 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & 15 & 12 & 6 & 0 \end{vmatrix} \quad D \begin{vmatrix} 0 & 8 & 5 & 9 & 7 \\ \infty & 0 & 2 & 1 & 4 \\ \infty & 3 & 0 & 4 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & 15 & 12 & 6 & 0 \end{vmatrix}$$



$$\begin{vmatrix} 0 & 10 & 5 & \infty & \infty \\ \infty & 0 & 2 & 1 & \infty \\ \infty & 3 & 0 & 9 & 2 \\ \infty & \infty & \infty & 0 & 4 \\ 7 & \infty & \infty & 6 & 0 \end{vmatrix}$$

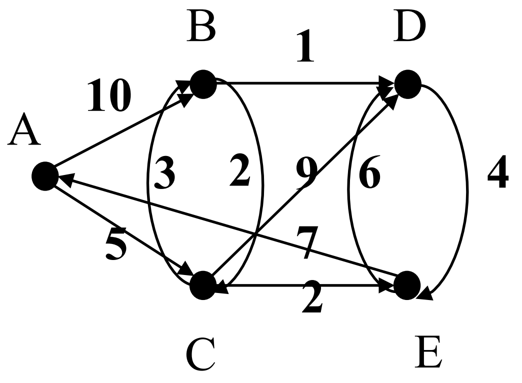
$$E \begin{vmatrix} 0 & 8 & 5 & 9 & 7 \\ 11 & 0 & 2 & 1 & 4 \\ 9 & 3 & 0 & 4 & 2 \\ 11 & 19 & 16 & 0 & 4 \\ 7 & 15 & 12 & 6 & 0 \end{vmatrix}$$

Floyd-Warshall（有向+正权）

```
Floyd-Warshall Algorithm (G, w, s)
Input: A matrix
Output: D matrix
for all i != j do:
      d_ij = A_ij
for i from 1 to n do:
      d_ii = 0
while True do:
      for i from 1 to n do:
            for j from 1 to n do:
                  D = D.min[D(i,:) min.+ D(:,j)]
```

$$
\begin{vmatrix}
0 & 8 & 5 & 11 & 7 \\
\infty & 0 & 2 & 1 & 4 \\
9 & 3 & 0 & 4 & 2 \\
11 & \infty & \infty & 0 & 4 \\
7 & 17 & 12 & 6 & 0
\end{vmatrix}
$$

$$
\begin{vmatrix}
0 & 8 & 5 & 9 & 7 \\
11 & 0 & 2 & 1 & 4 \\
9 & 3 & 0 & 4 & 2 \\
11 & 19 & 16 & 0 & 4 \\
7 & 15 & 12 & 6 & 0
\end{vmatrix}
$$



$$
\begin{vmatrix}
0 & 10 & 5 & \infty & \infty \\
\infty & 0 & 2 & 1 & \infty \\
\infty & 3 & 0 & 9 & 2 \\
\infty & \infty & \infty & 0 & 4 \\
7 & \infty & \infty & 6 & 0
\end{vmatrix}
$$

图算法介绍

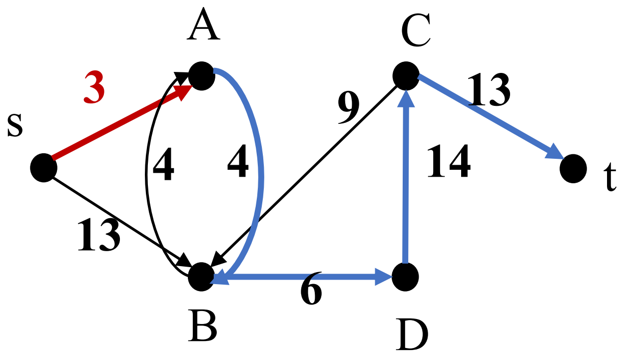
1. 基础概念

2. 强连通分量

3. 最短路算法

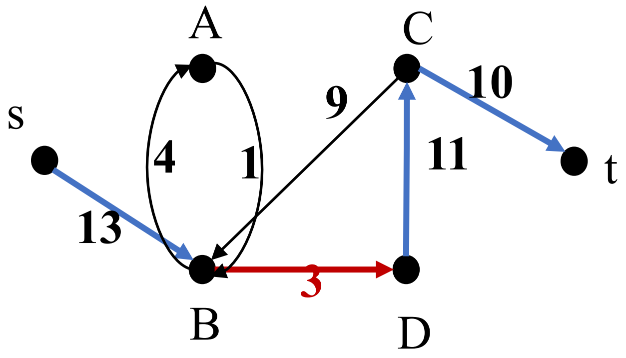
4. **最大流算法**

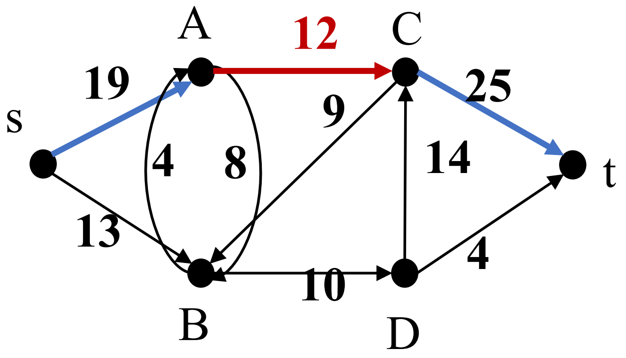
Ford-Fulkerson（ $O(mn)$ ）

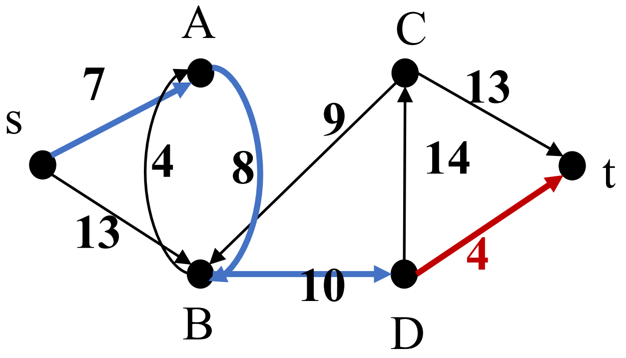


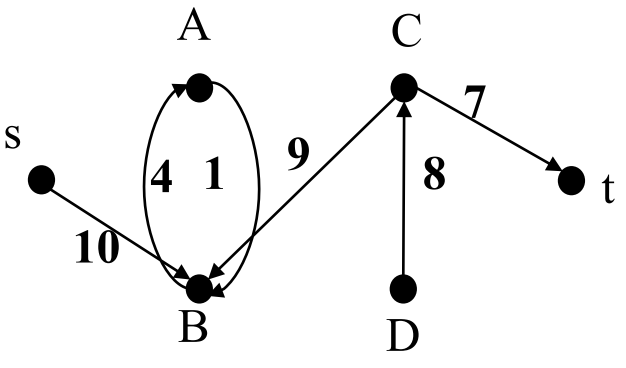
Ford-Fulkerson Algorithm (G, s, t) （DFS）

```
Input: (G, s, t)
Output: maximum_flow(s,t)
for e(u, v) ∈ E do:
    e(u, v).f = 0
while find a route from s to t in the residual network
    m = min{e(u, v).f, e(u, v) ∈ route}
    for e(u, v) ∈ route do:
        if e(u, v) ∈ f do:
            e(u, v).f  = e(u, v).f  + m
        else do:
            e(u, v).f  = e(u, v).f  - m
```



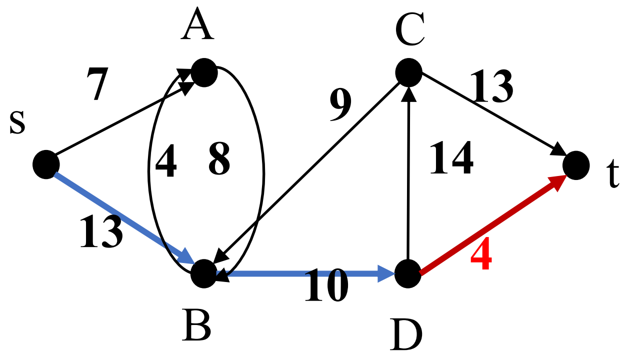




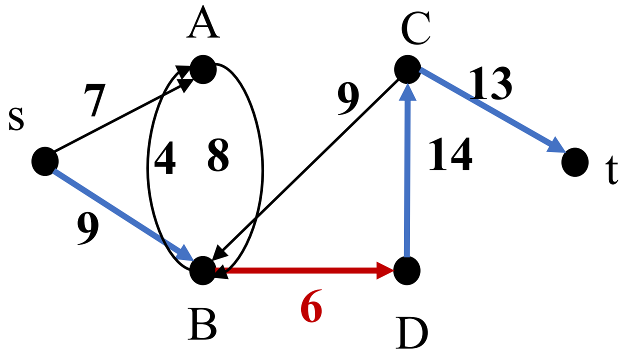


Edmonds-karp（$O(m^2n)$）

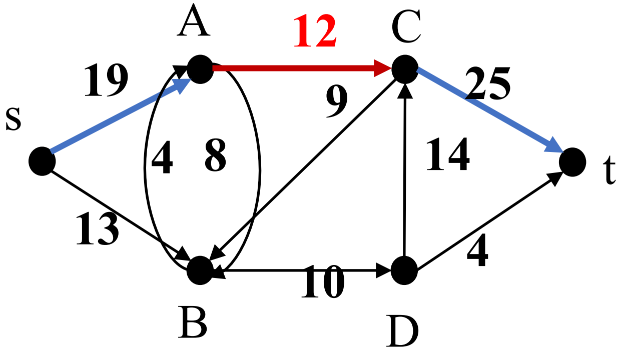
```
Edmonds-karp Algorithm (G, s, t) （BFS）
Input: (G, s, t)
Output: maximum_flow(s,t)
for e(u, v) ∈ E do:
    e(u, v).f = 0
while find a shortest route from s to t in E
    m = min{e(u, v).f, e(u, v) ∈ route}
    for e(u, v) ∈ route do:
        if e(u, v) ∈ f do:
            e(u, v).f  = e(u, v).f  + m
        else do:
            e(u, v).f  = e(u, v).f  - m
```



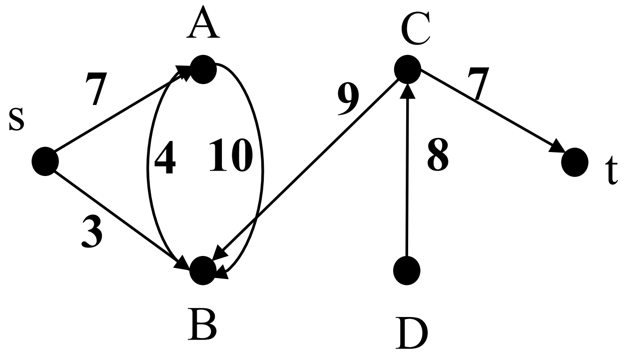
l=0	s
l=1	A,B
l=2	D
l=3	C,t



l=0	s
l=1	A,B
l=2	D
l=3	C
l=4	t



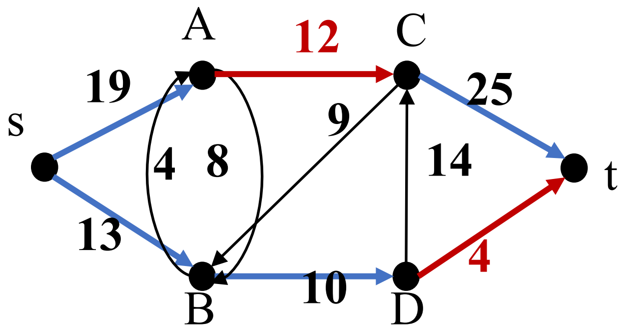
l=0	s
l=1	A,B
l=2	C,D
l=3	t



Dinic（$O(n^2m)$）

```
Dinic Algorithm (G, s, t)

Input: (G, s, t)
Output: maximum_flow(s,t)
while find a shortest route from s to t in E do:
    BFS()
    while find a do:
        ans += a
return ans
```

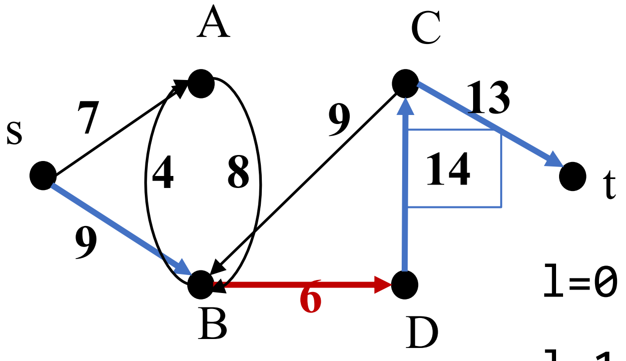


l=0 s

l=1 A,B

l=2 C,D

l=3 t



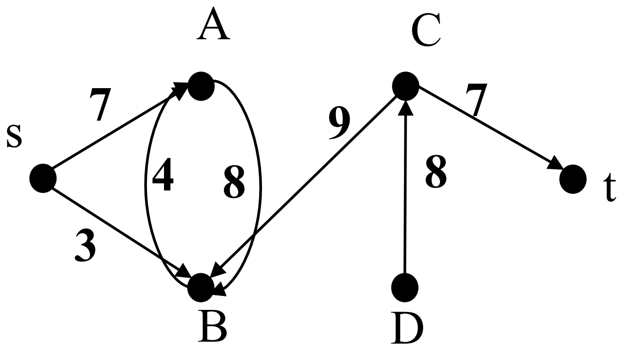
l=0 s

l=1 A,B

l=2 D

l=3 C

l=4 t



图算法介绍

1. 基础概念

2. 强连通分量

3. 最短路算法

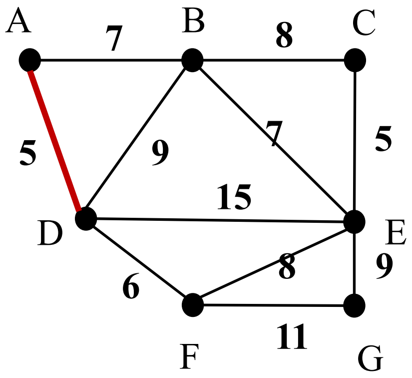
4. 最大流算法

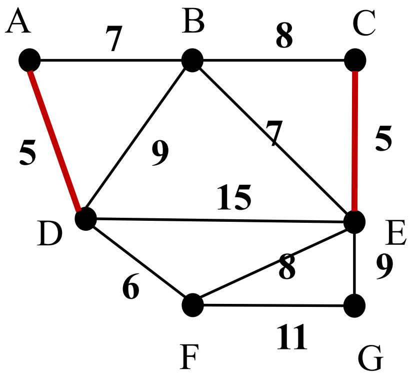
5. **支撑树算法**

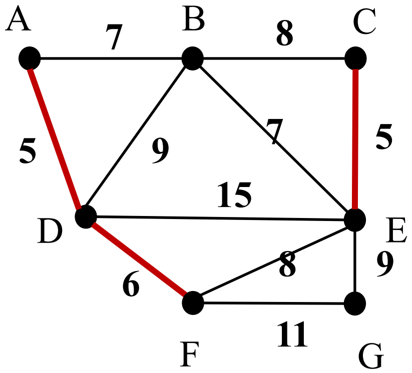
Kruskal（$O(n^2 logm)$）

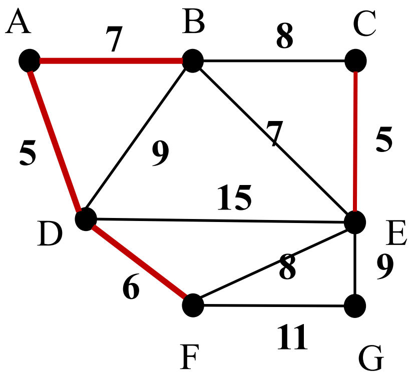
Kruskal Algorithm (G, s, t)

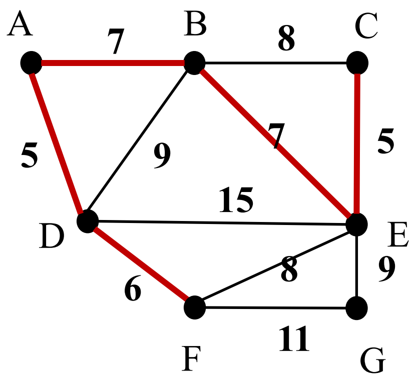
```
Input: G
Output: Tree(G)
Tree(G) = ∅
for each vertex v ∈ V do:
    MAKE-SET(v)
sort the edges of E into nondecreasing order by weight
for each edge(u, v) ∈ E do:
    taken in nondecreasing order by weight
    if FIND-SET(u) != MAKE-SET(v) then:
        Tree(G) = Tree(G) ∪{(u, v)}
        UNION(u, v)
return Tree(G)
```

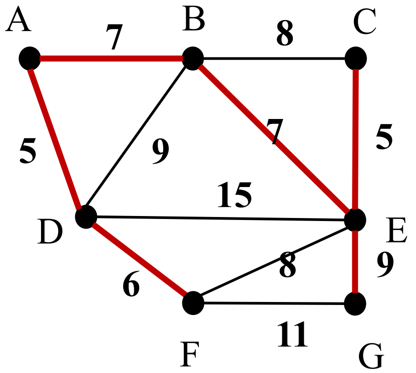


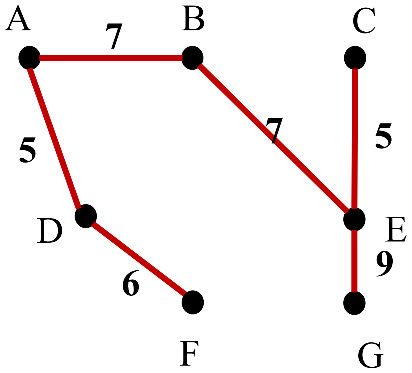






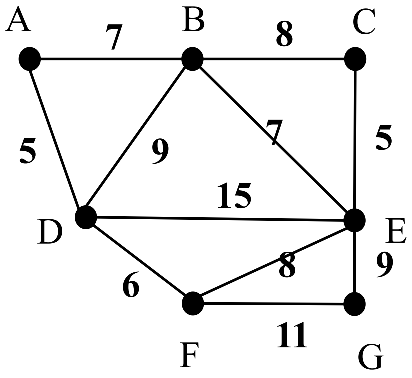


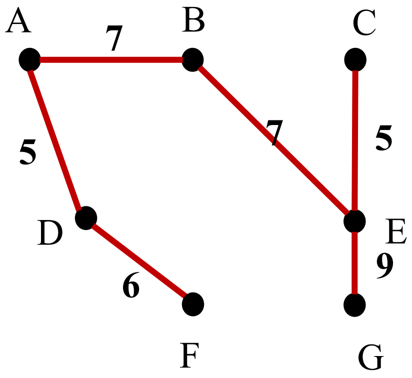


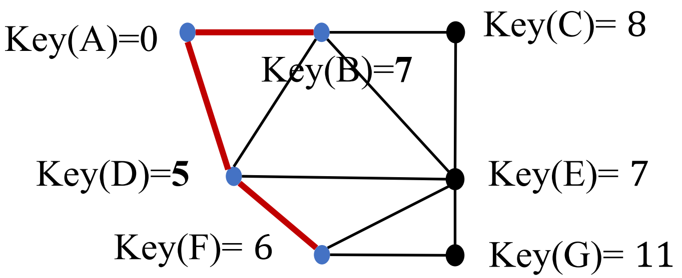


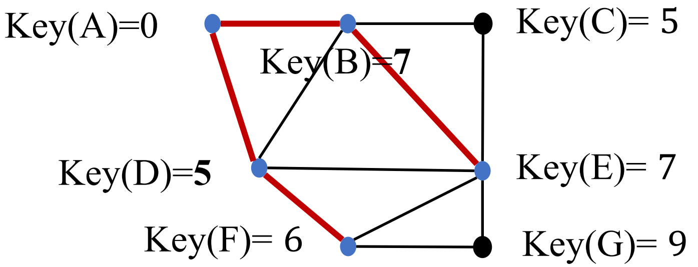
Prim（$O(m+nlogm)$）

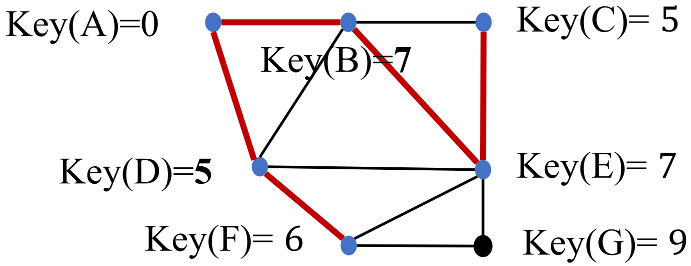
```
Prim Algorithm (G, s, t)
for each vertex u ∈ V do:
    key[u] = ∞
    π[u] = NIL
Key[r]=0
Q = V
while Q != ∅ do:
    u = EXTRACT-MIN(Q)
    for each v ∈ Adj[u] do:
        if v ∈ Q & w(u, v) < key[v] then:
            π[v] = u
            key[v] = w(u, v)
```

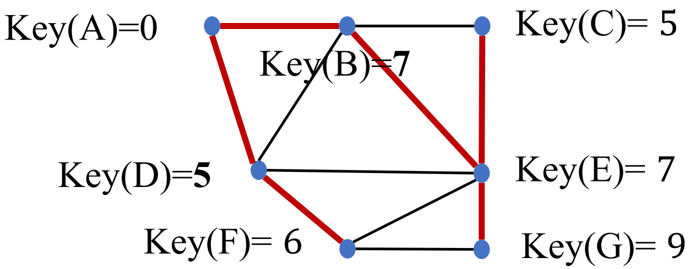


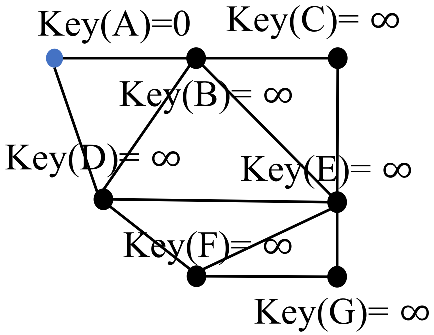


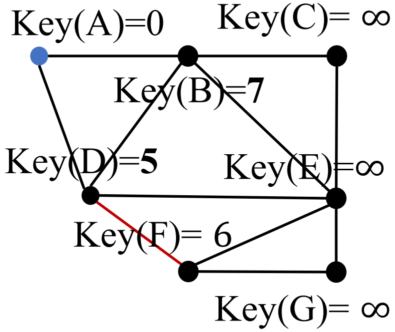


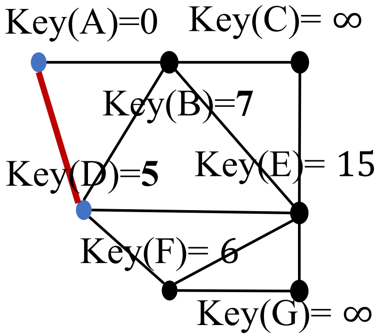


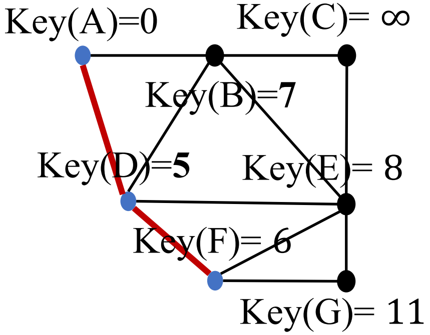




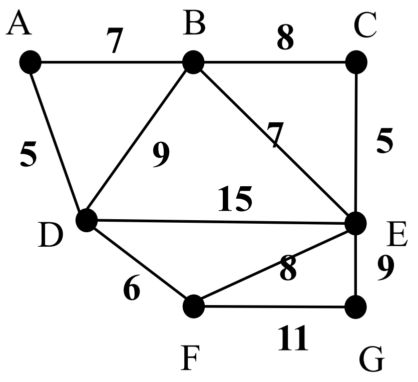








```
Prim Algorithm (G, s, t)
```
```
Initial u
for each vertex v ∈ V\u do:
    d[v] = ∞
    d[u] = 0
S = V\u
Q = set.add(u)
while S != Ø do:
    u = EXTRACT-MIN(Q)
    d = d.min(A[u])
    select ’argmin(d(u’)) u’∈ S
    S.del(u’)
    Q.add(u’)
```



$$\begin{vmatrix} \infty & 7 & \infty & 5 & \infty & \infty & \infty \\ 7 & \infty & 8 & 9 & 7 & \infty & \infty \\ \infty & 8 & \infty & \infty & 5 & \infty & \infty \\ 5 & 9 & \infty & \infty & 15 & 6 & \infty \\ \infty & 7 & 5 & 15 & \infty & 8 & 9 \\ \infty & \infty & \infty & 6 & 8 & \infty & 11 \\ \infty & \infty & \infty & \infty & 9 & 11 & \infty \end{vmatrix}$$

d =[0, ∞, ∞, ∞, ∞, ∞, ∞]

A, d =[0, 7, ∞, 5, ∞ , ∞, ∞]

D, d =[0, 7, ∞, 5, 15, 6, ∞]

F, d =[0, 7, ∞, 5, 8, 6, ∞]

B, d =[0, 7, 8, 5, 7, 6, ∞]

E, d =[0, 7, 5, 5, 7, 6, 9]

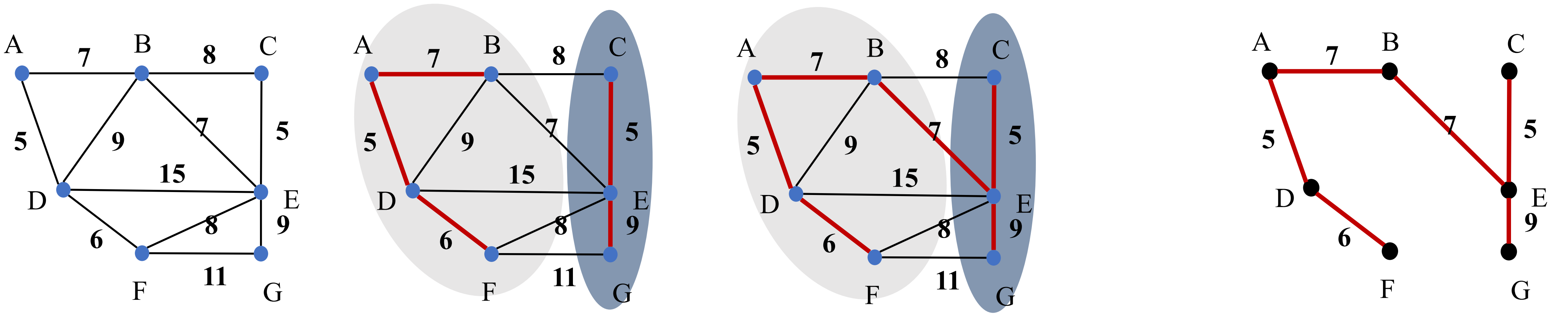
C, d =[0, 7, 5, 5, 7, 6, 9]

F, d =[0, 7, 5, 5, 7, 6, 9]

Boruvka（$O(nlogm)$）

```
Boruvka Algorithm (G, s, t)

Input: G
Output: Tree(G)
Tree(G) = Ø
while Tree(G) does not from a spanning tree do:
    find an edge(u, v) that is safe for Tree(G)
        Tree(G) = Tree(G) ∪{(u, v)}
return Tree(G)
```



{A-D, B-A, C-E, F-D, G-E}　　{B-C, B-E, D-E, F-E, F-E, F-G}

Homework

Homework：①破圈法求最小树，试着分析算法复杂性

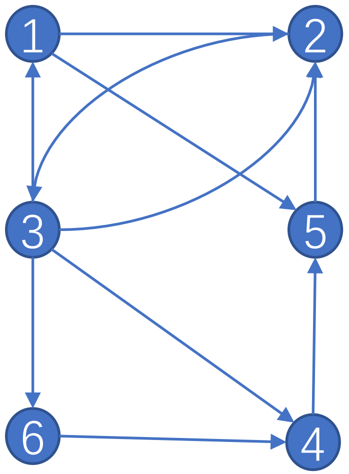
②改进kruskal算法（增加判断不连通的情况，处理G为不连通的情形）

5.如何求任意两点的最短路?

6.有负权的网络如何设计最短路?

7.最大流的快速求解算法

Johnson Algorithm



Stack
1
2
3

Blocked
1
2
3

B

Stack
1
2
3
4
5

Blocked
1
2
3
4
5

B
2:5
4:5

Stack
1
2
3
6

Blocked
1
2
3
4
5
6

B
2:5
4:5
4:6

Stack
1

Blocked
1

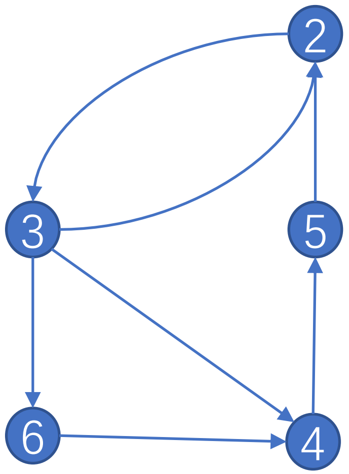
B

Stack
1
5
2
3

Blocked
1
5
2
3

B

Johnson Algorithm （n + m）*C



Homework

Homework：①破圈法求最小树，试着分析算法复杂性

②改进kruskal算法（增加判断不连通的情况，处理G为不连通的情形）

5.如何求任意两点的最短路?

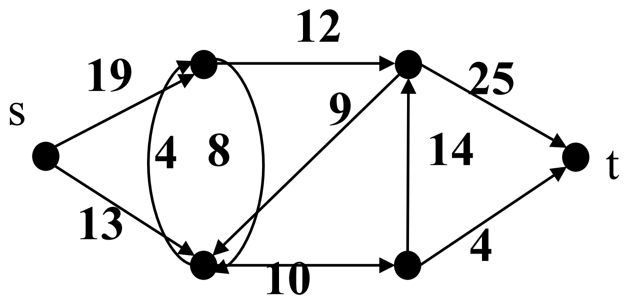
6.有负权的网络如何设计最短路?

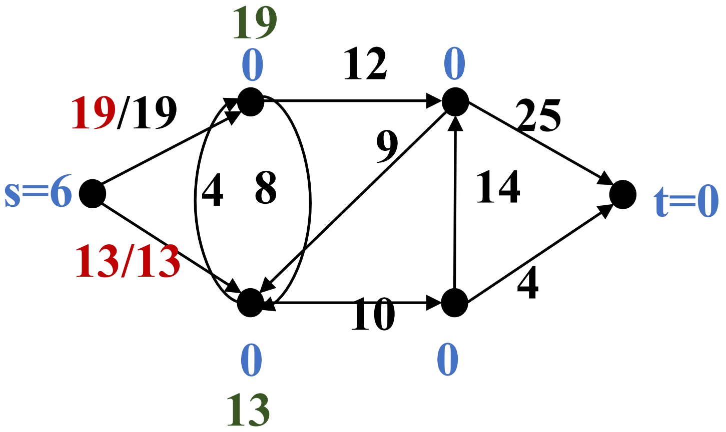
7.最大流的快速求解算法:

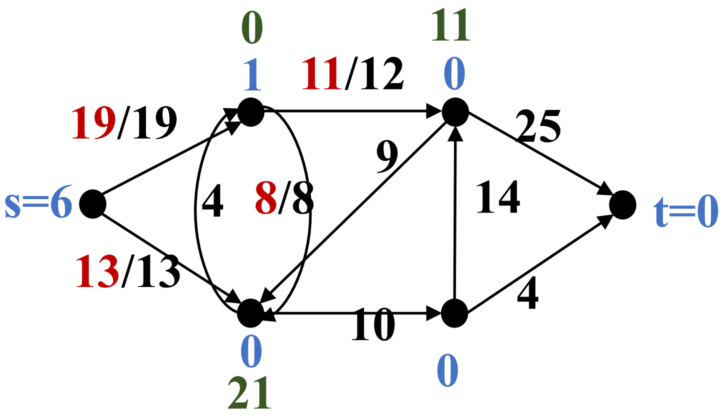
https://arxiv.org/pdf/2203.00671.pdf

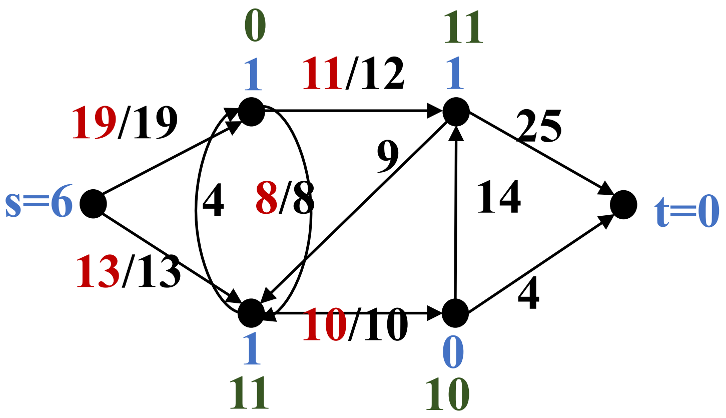
Push_Relabel （$O(n^2m)$）

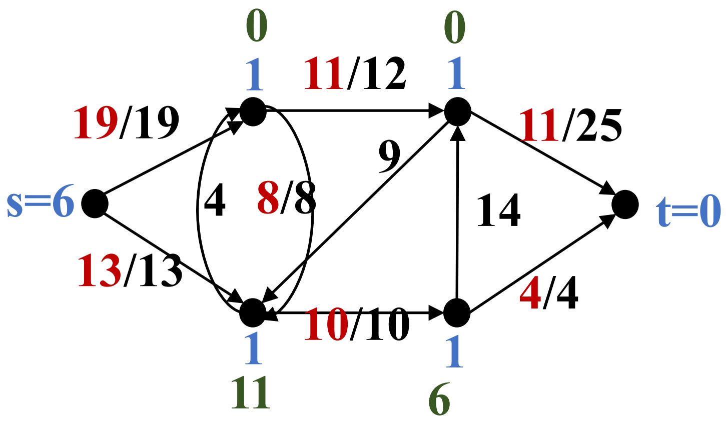
```
Push_Relabel Algorithm (G, s, t)
Initialize PreFlow : Initialize Flows and Heights While it
is possible to perform a Push() or Relabel() on a vertex
// Or while there is a vertex that has excess flow Do Push()
or Relabel()
// At this point all vertices have Excess Flow as 0 (Except
source // and sink)
Return flow.
```

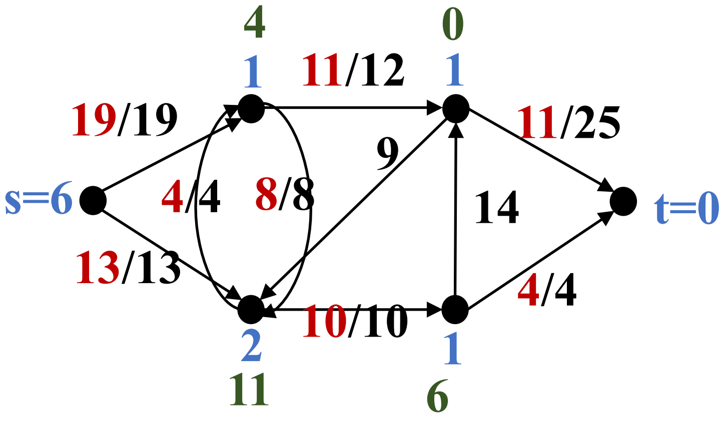


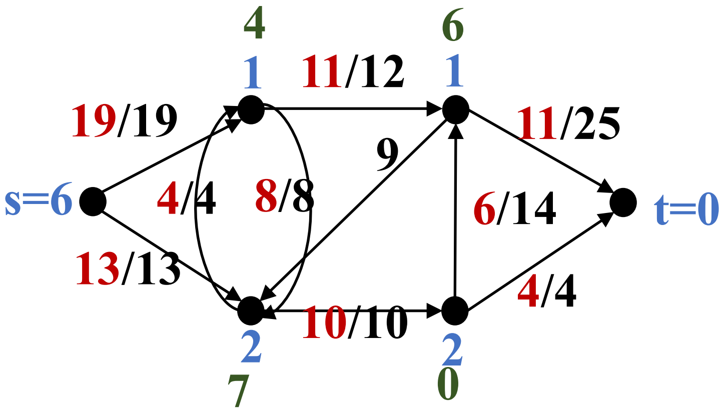


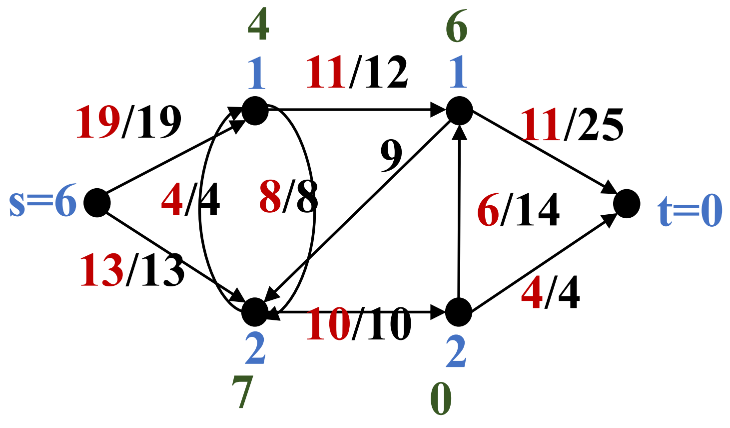


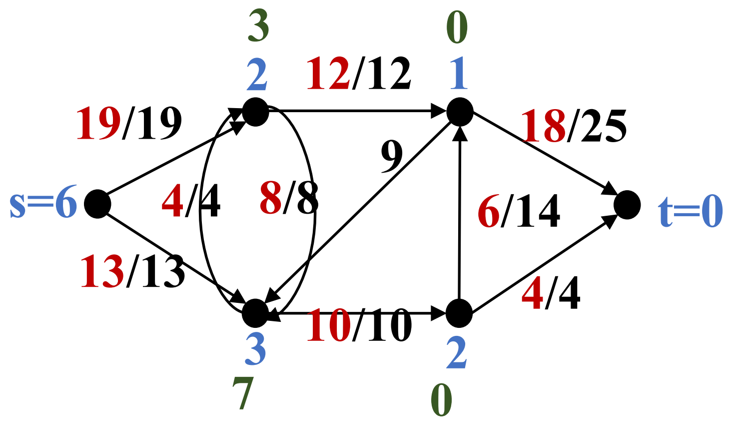


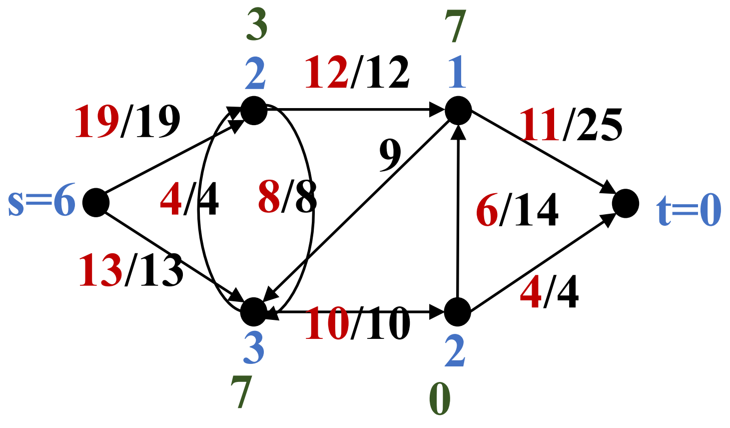


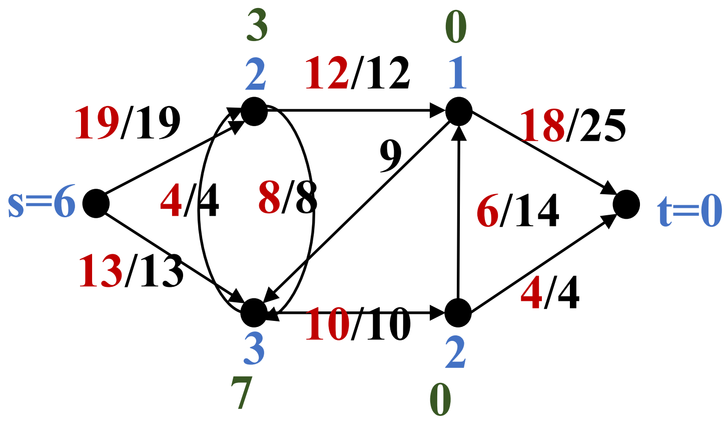





4
1
11/12
6
1
19/19
11/25
9
s=6
4/4
8/8
6/14
t=0
13/13
4/4
10/10
2
7
2
0


3
2
12/12
0
1
19/19
18/25
9
s=6
4/4
8/8
6/14
t=0
13/13
4/4
10/10
3
7
2
0


3
2
12/12
7
1
19/19
11/25
9
s=6
4/4
8/8
6/14
t=0
13/13
4/4
10/10
3
7
2
0


3
2
12/12
0
1
19/19
18/25
9
s=6
4/4
8/8
6/14
t=0
13/13
4/4
10/10
3
7
2
0