图算法介绍

1. 基础概念

复杂度

时间复杂度：$\lim \frac{T(n)}{O(n)} = C$

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

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

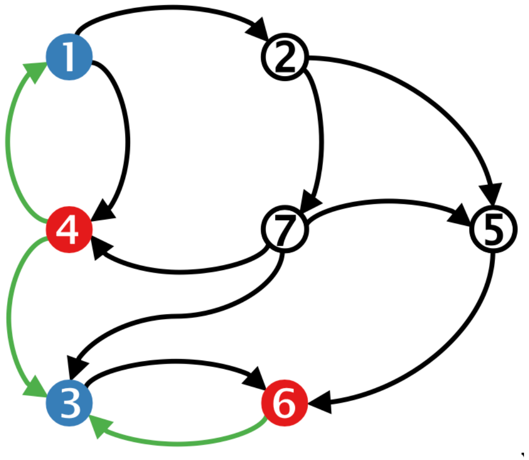
边列表

目标

存储与效率

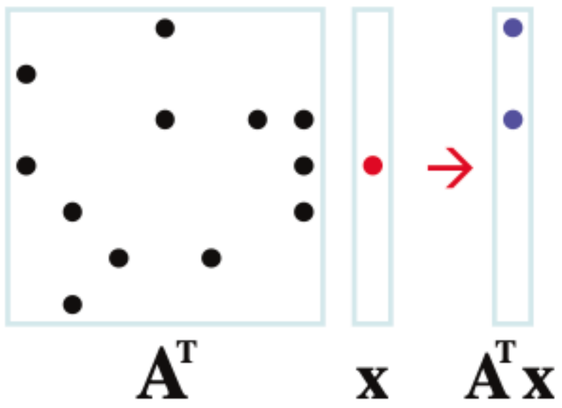
基础知识

广度优先搜索（Breadth-First-Search，BFS）

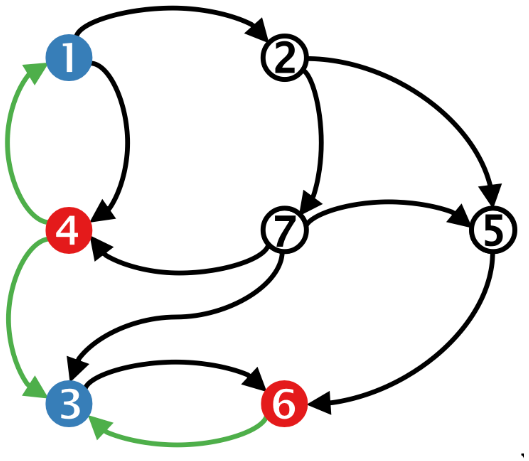


l=0	4
l=1	1,3
l=2	2,6
l=3	5,7

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

图算法介绍

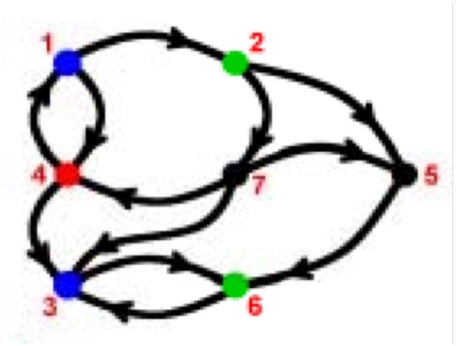
1. 基础概念

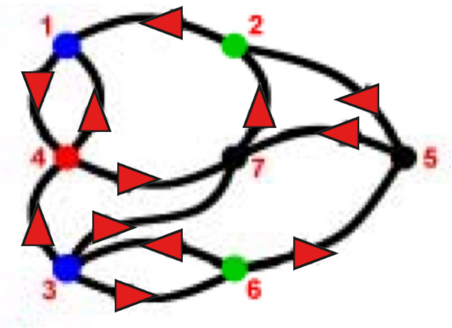
2. **强连通分量**

强连通分量

Kosaraju （$O(m+n)$）

```
Kosaraju Algorithm (G, s, t)
Input: G
Output: Component(G)
Call DFS(G) to compute finishing times f[u] for v ∈ V
Compute G^T
Call DFS(G^T).
but in the main loop of DFS, consider the decreasing
Output the vertices of
```





G:
1→2 →5→6 →3
→7 →4

G^T:
4→1
→7 →2

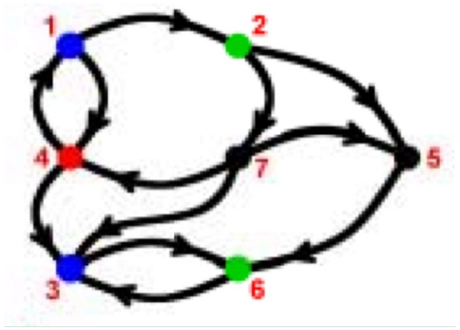
Connected _Component_1 = [1,2,4,7]

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

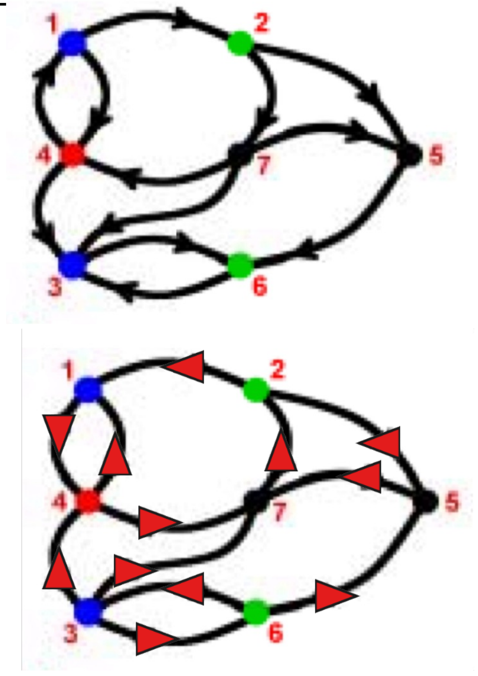
4.

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**

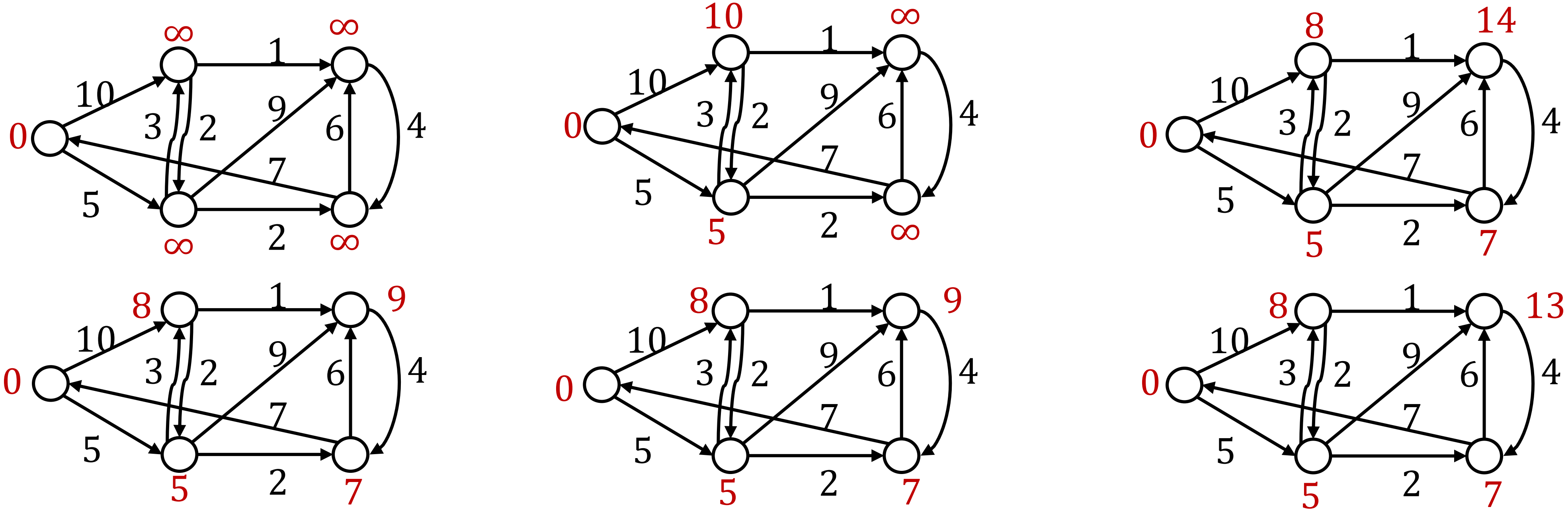
最短路

Dijkstra Algorithm （有向+非负权）

```
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 $v \in G, Adj[u]$ do:
        RELAX(u, v, w)
```

```
Initialize-Single-Source
for each $v \in V$ do:
    $d(v) = \infty$
    $d(s) = 0$
```

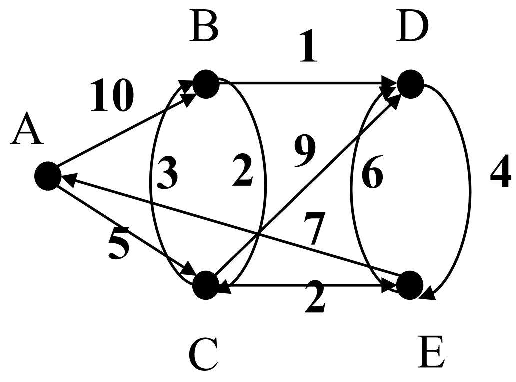
```
RELAX(u, v, w)
if $p(s,v) > p(s,u) + w(u,v)$ then:
    $p(s,v) = p(s,u) + w(u,v)$
    $v.\pi = u$
```

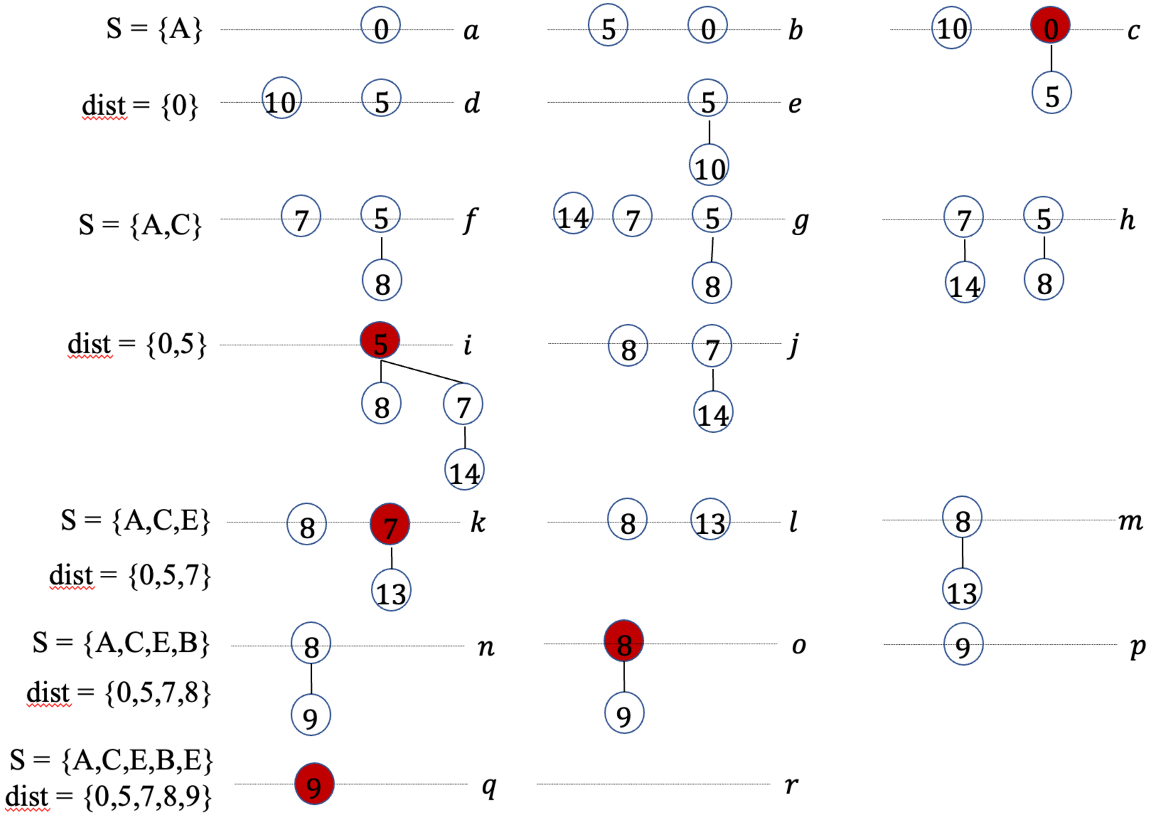


最短路

Dijkstra Algorithm （有向+非负权）

```
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 $v \in Q - S, Adj[u]$ do:
        if $p(s,v) > p(s,u) + w(u,v)$ then:
            decrease node v or insert v
    delete u in Heap
```





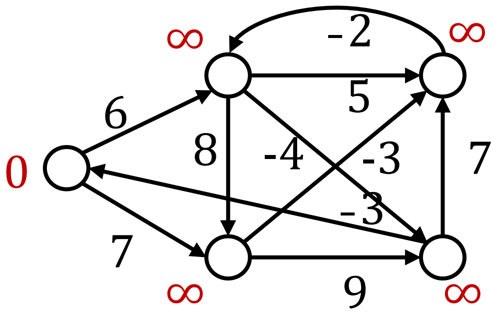
步骤：从a-b的堆示意图

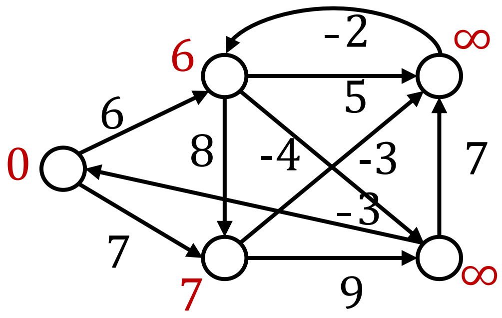
最短路

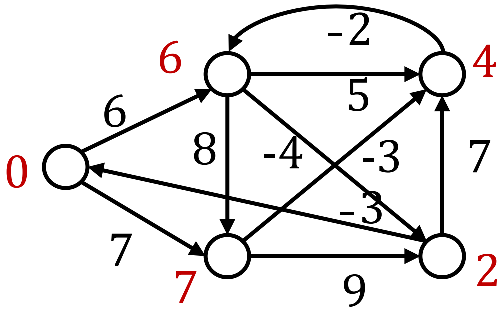
Bellman-Ford Algorithm （有向+正负权）

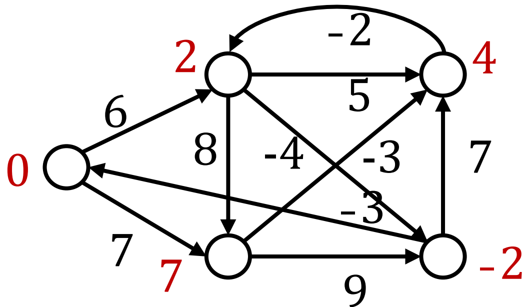
```
Bellman-Ford(G, w, s)
Initialize-Single-Source
for i from 1 to |V|-1 do:
    for each edge(u, v) in |E| do:
        RELAX(u, v, w)
for each edge(u, v) in |E| do:
    if $p(s,v) > p(s,u) + w(u,v)$ do:
        return False
```

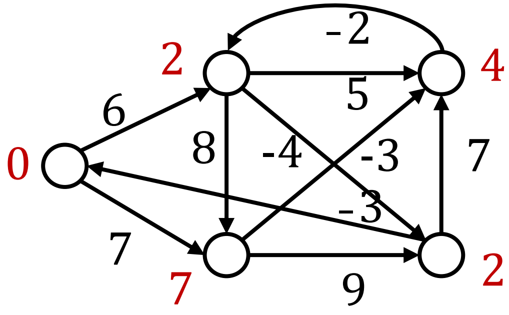
```
RELAX(u, v, w)
if $p(s,v) > p(s,u) + w(u,v)$ do:
    $p(s,v) = p(s,u) + w(u,v)$
    $v.\pi = u$
```











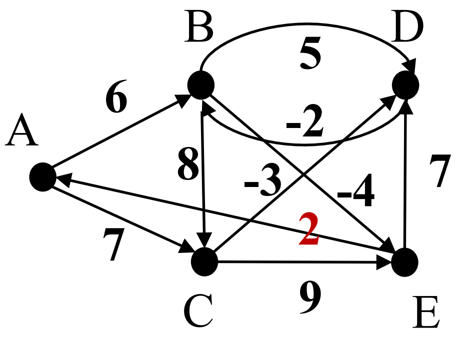
最短路

Bellman-Ford Algorithm （有向+正负权）

```
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

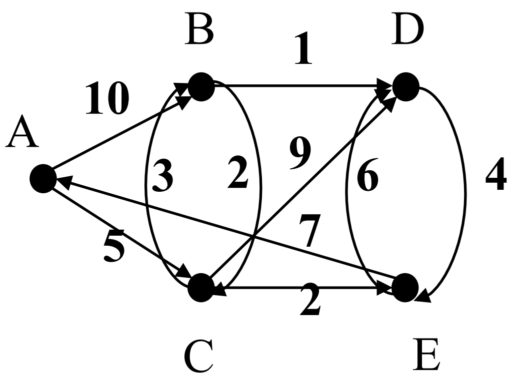
0	7	6	--	--
--	0	--	-3	9
--	8	0	5	-4
--	--	-2	0	--
2	--	--	7	0

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

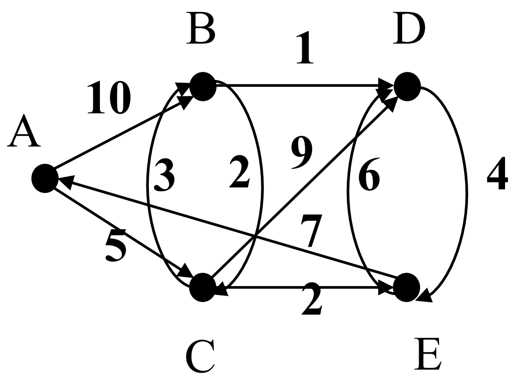
最短路

Floyd-Warshall（有向+正负权）

```
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:
    $d_{ii} = 0$
for k from 1 to n do:
    for i from 1 to n do:
        for j from 1 to n do:
            $d_{ij} = \min\{d_{ij}, d_{ik} + d_{kj}\}$
```



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

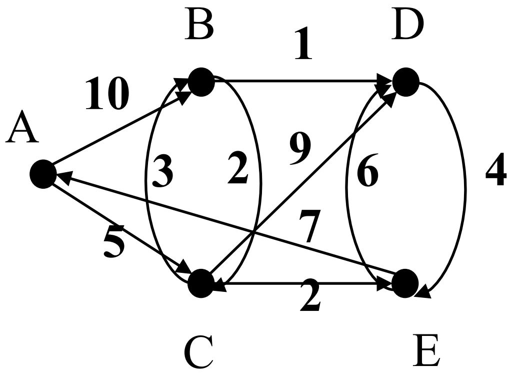


$$\begin{vmatrix} 0 & 8 & 5 & 9 & 7 \\ 11 & 0 & 2 & 1 & 4 \\ 9 & 3 & 0 & 4 & 2 \\ 11 & 19 & 16 & 0 & 4 \\ 7 & 15 & 12 & 16 & 0 \end{vmatrix}$$

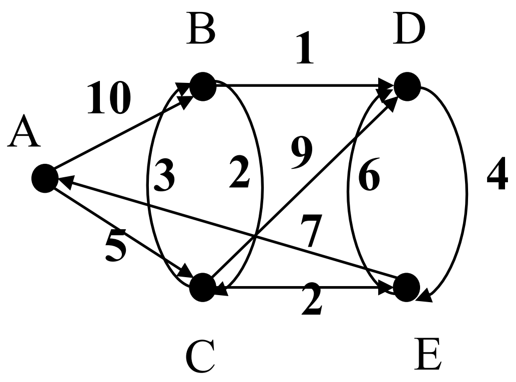
最短路

Floyd-Warshall（有向+正负权）

```
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:
    $d_{ii} = 0$
for k from 1 to n do:
    for i from 1 to n do:
        for j from 1 to n do:
            $d_{ij} = \min\{d_{ij}, d_{ik} + d_{kj}\}$
```



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



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

图算法介绍

1. 基础概念

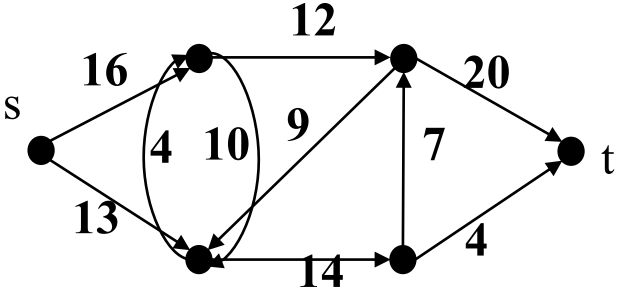
2. 强连通分量

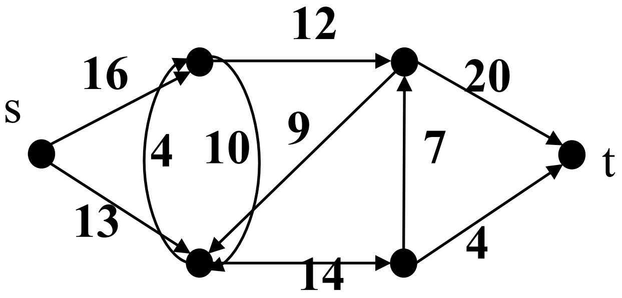
3. 最短路算法

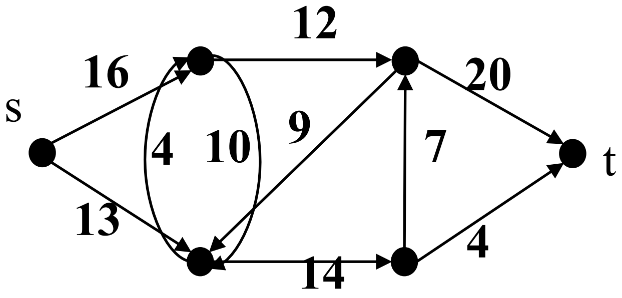
4. **最大流算法**

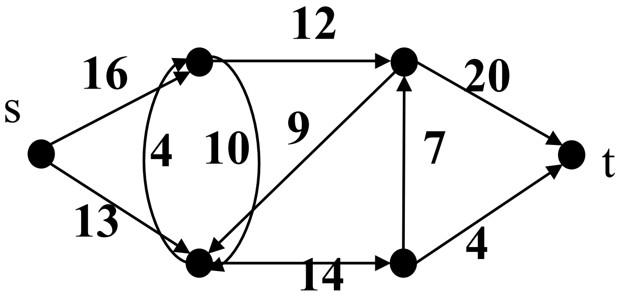
Ford-Fulkerson（ $O(mn)$ ）

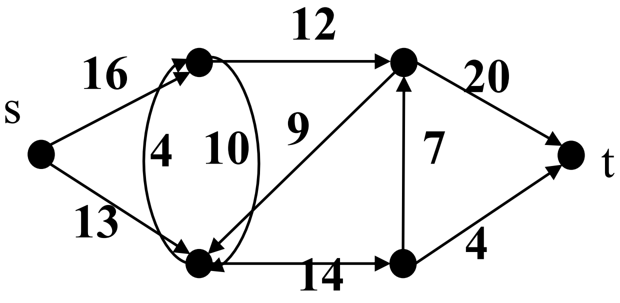
```
Ford-Fulkerson Algorithm (G, s, t)
Input: (G, s, t)
Output: maximum_flow(s,t)
for e(u, v) ∈ E do:
    e(u, v).f = 0
while find a route from s to t in E
    m = min{e(u, v).f, e(u, v) ∈ route}
    for e(u, v) ∈ route do:
        if e(u, v) ∈ f do:
            e(u, v).f  = e(u, v).f  + m
        else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

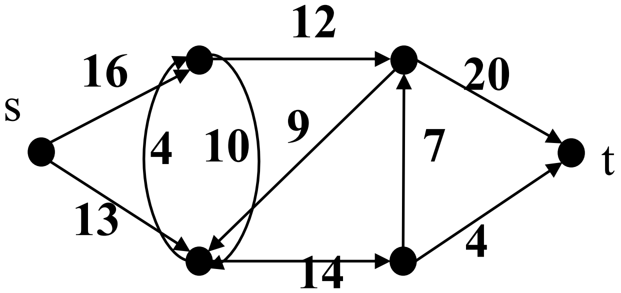










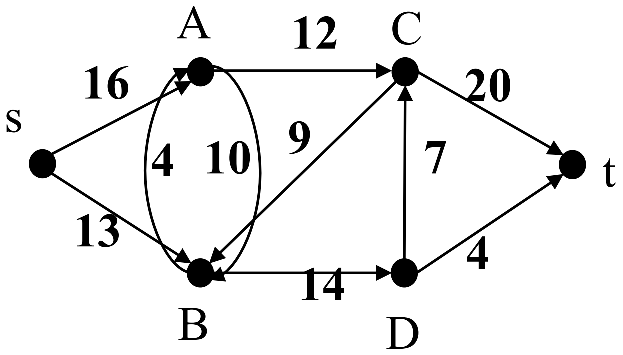


最大流

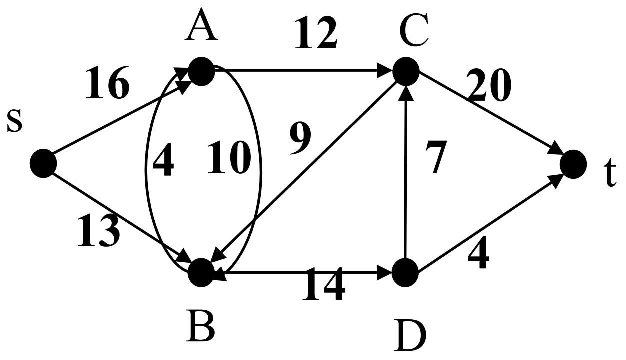
Edmonds-karp（$O(m^2n)$）

Edmonds-karp Algorithm (G, s, t)

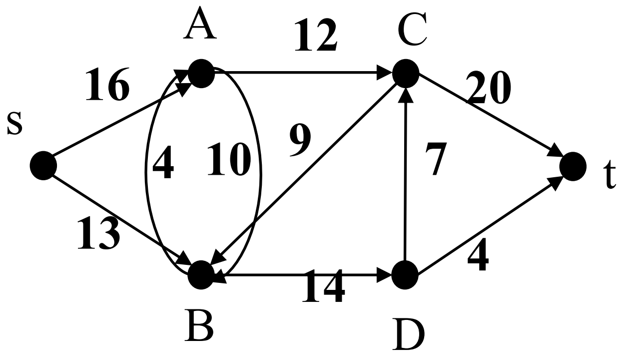
```
Input: (G, s, t)
Output: maximum_flow(s,t)
for e(u, v) ∈ E do:
    e(u, v).f = 0
while find a shortest route from s to t in E
    m = min{e(u, v).f, e(u, v) ∈ route}
    for e(u, v) ∈ route do:
        if e(u, v) ∈ f do:
            e(u, v).f  = e(u, v).f  + m
        else do:
            e(u, v).f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



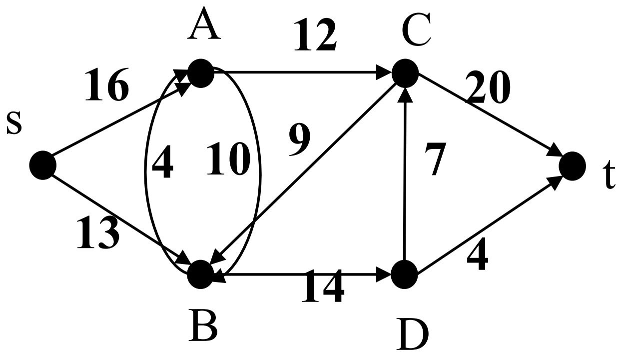
l=0	s
l=1	A,B
l=2	C,D
l=3	t



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

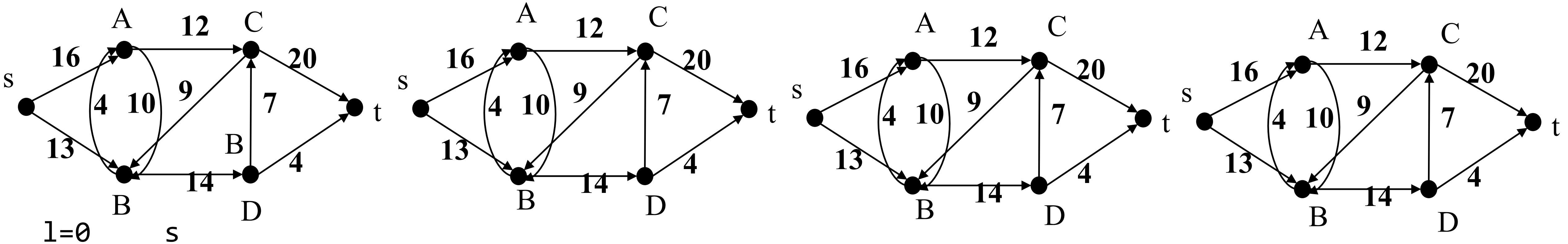


l=0	s
l=1	A,B
l=2	D
l=3	C
l=4	t



Dinic（$O(n^2m)$）

```
Dinic Algorithm (G, s, t)
Input: (G, s, t)
Output: maximum_flow(s,t)
while there exits a path p from s to t in the residual network do:
    BFS()
    while find a do:
        ans += a
return ans
```



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

图算法介绍

1. 基础概念

2. 强连通分量

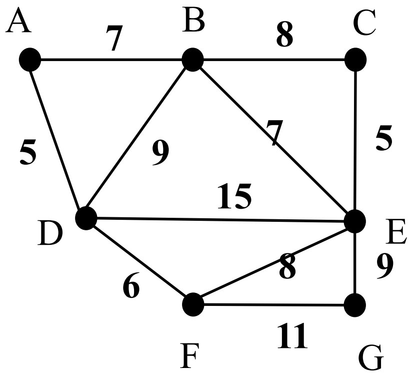
3. 最短路算法

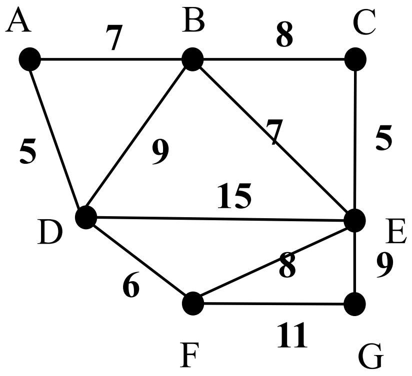
4. 最大流算法

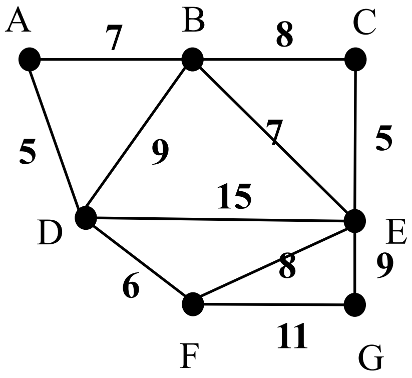
5. **支撑树算法**

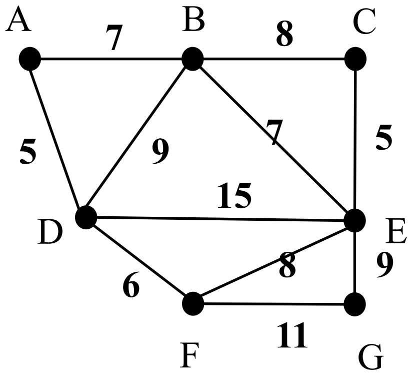
Boruvka（$O(nlogm)$）

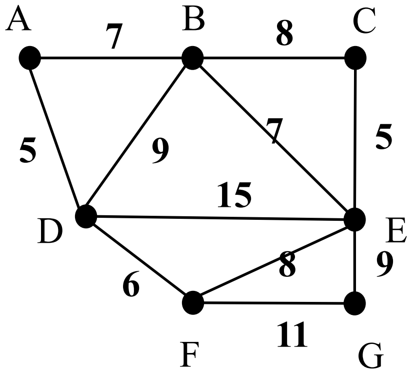
```
Boruvka Algorithm (G, s, t)
Input: G
Output: Tree(G)
Tree(G) = Ø
while A does not from a apanning tree do:
    find an edge(u, v) that is safe for Tree(G)
        Tree(G) = Tree(G) ∪{(u, v)}
return Tree(G)
```







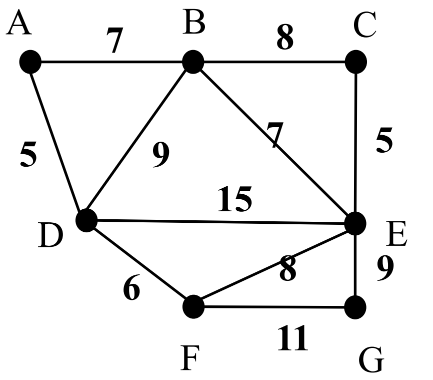


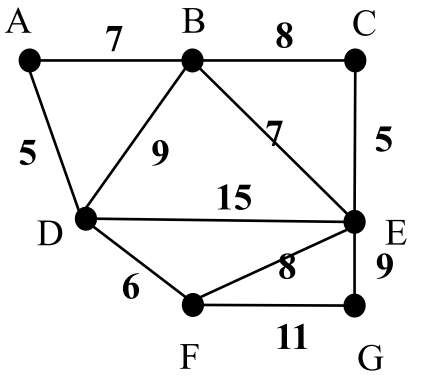


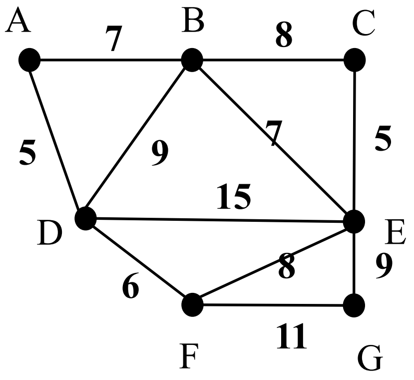
Kruskal（$O(n^2logm)$）

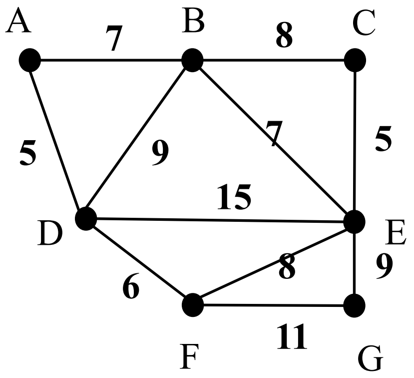
Boruvka Algorithm (G, s, t)

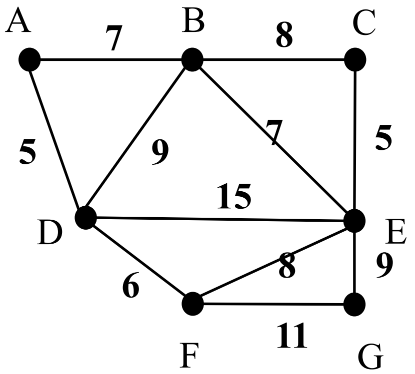
```
Input: G
Output: Tree(G)
Tree(G) = ∅
for each vertex v ∈ V do:
    MAKE-SET(v)
sort the edges of E into nondecreasing order by weight
for each edge(u, v) ∈ E do:
    taken in nondecreasing order by weight
    if FIND-SET(u) != MAKE-SET(v) then:
        Tree(G) = Tree(G) ∪{(u, v)}
        UNION(u,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

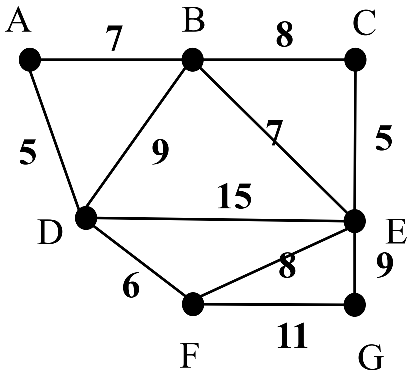












Prim（$O(m+nlogm)$）

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