Advanced Competitive Programming

國立成功大學ACM-ICPC程式競賽培訓隊
nckuacm@imslab.org

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Made by 培訓團隊群 CCNS.

Outline

- 最大流
- Articulation point
- Strongly Connected Components

Maximum Flow

最大流

Competitive Programming

最大流

給定帶權重連通圖

找到此圖從 s 點 到 t 點的最大流量

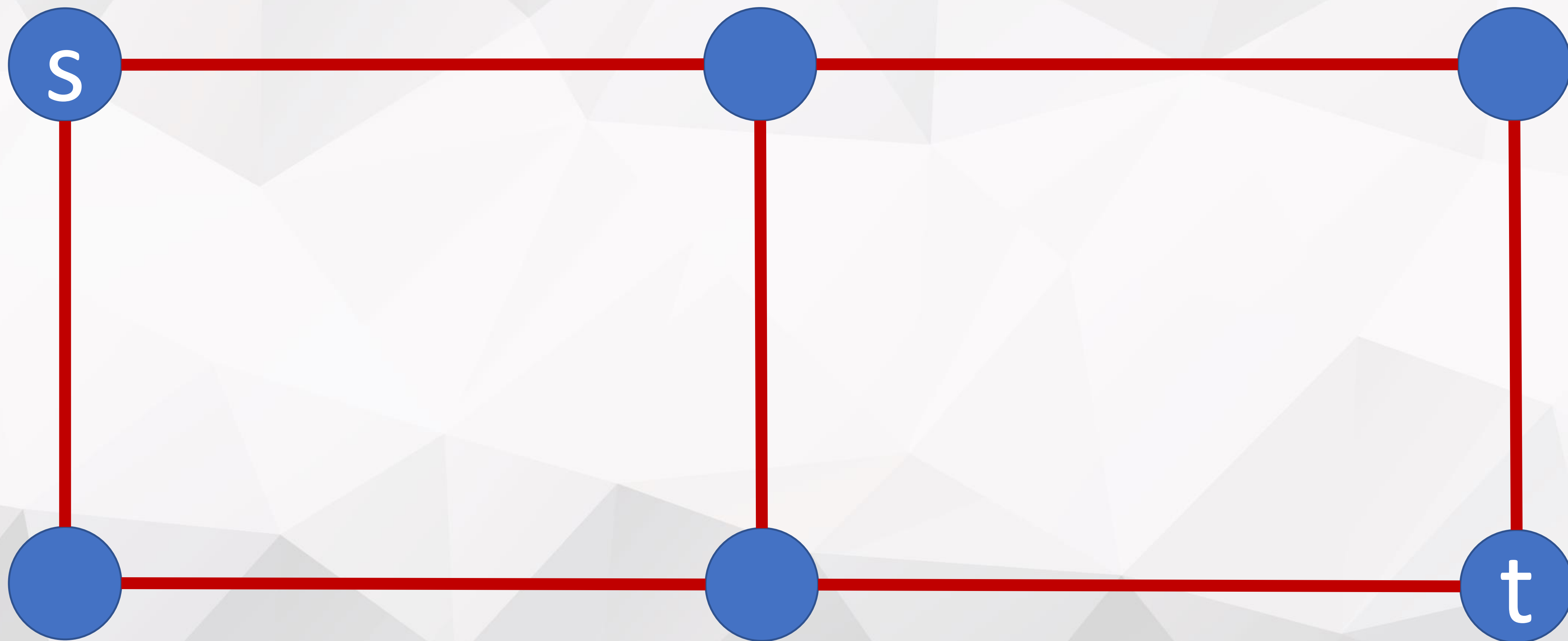
前提

- 流
- 流量
- 容量
- 剩餘容量

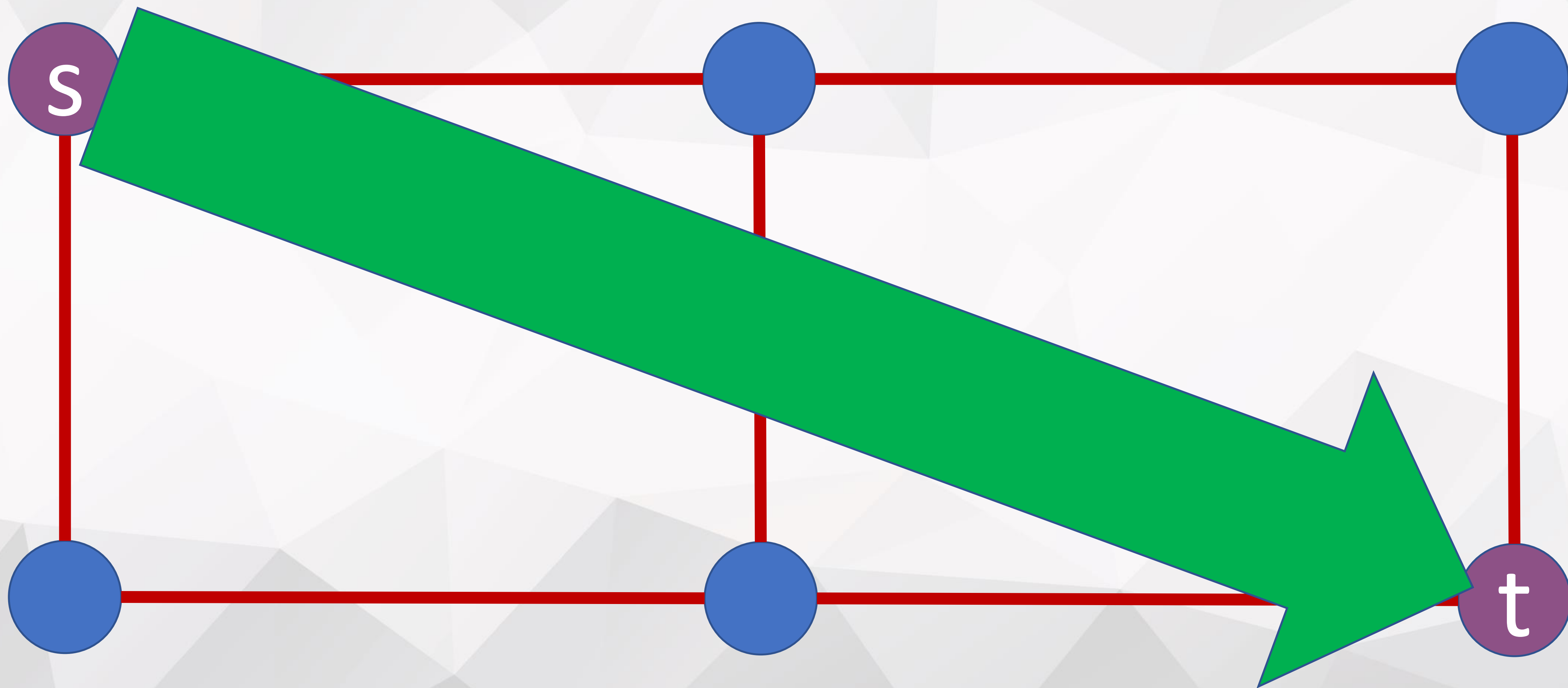
前提

- 流
- 流量
- 容量
- 剩餘容量

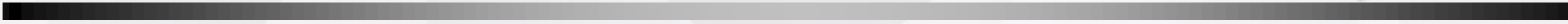
流



流



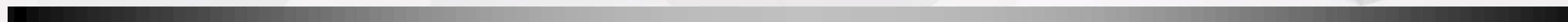
流



s

t

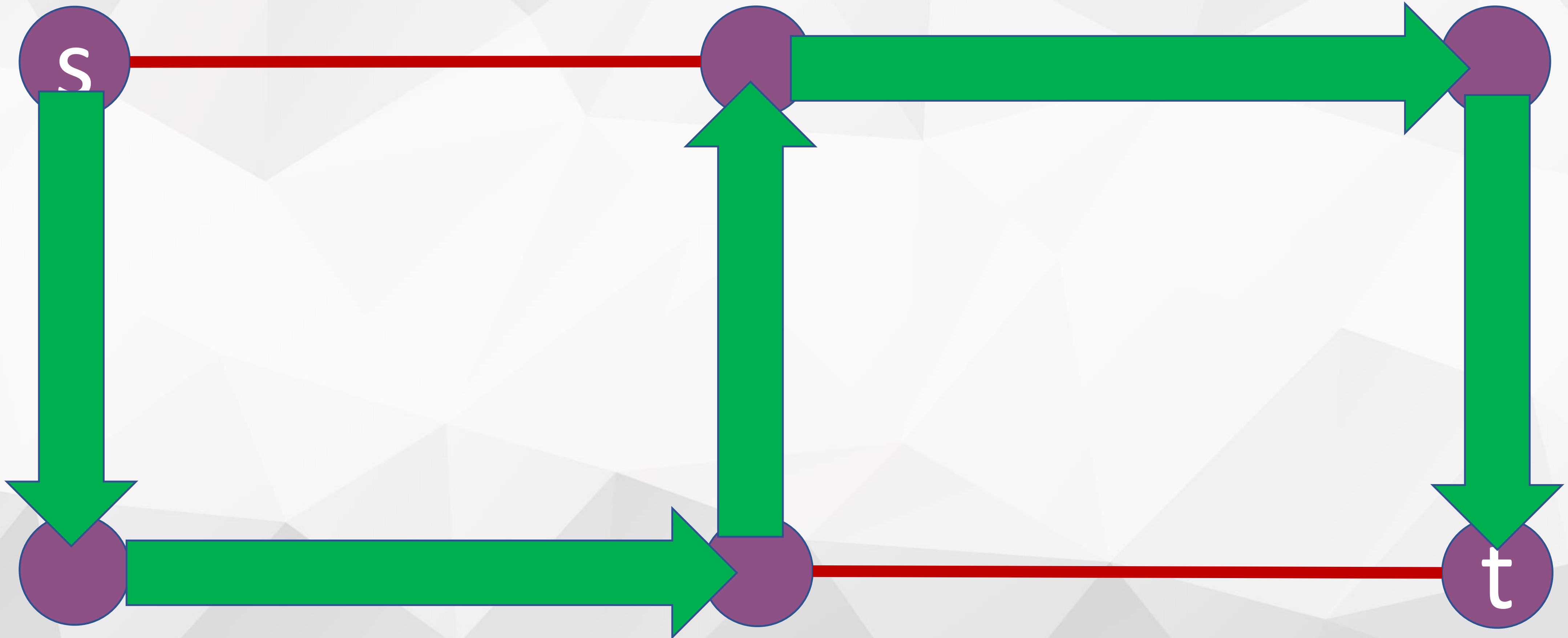
流



s

t

流

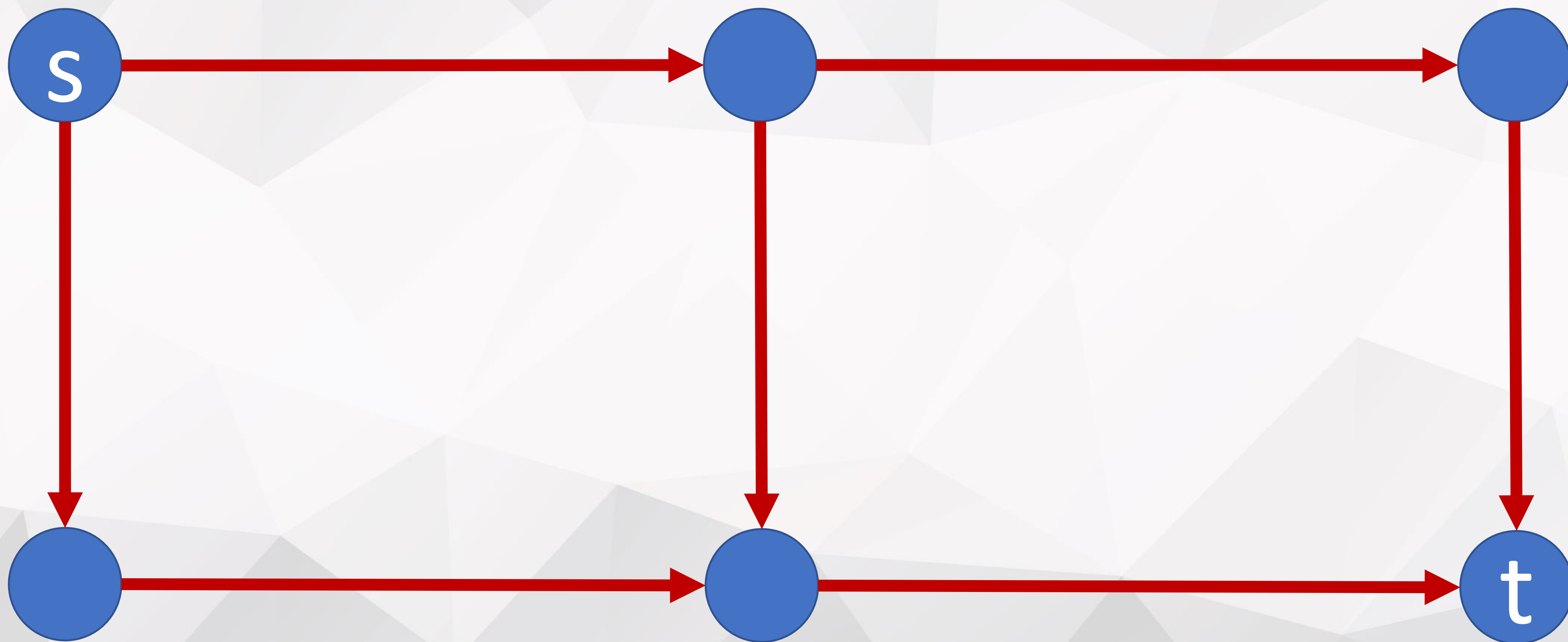


前提

- 流量
- 容量
- 剩餘容量

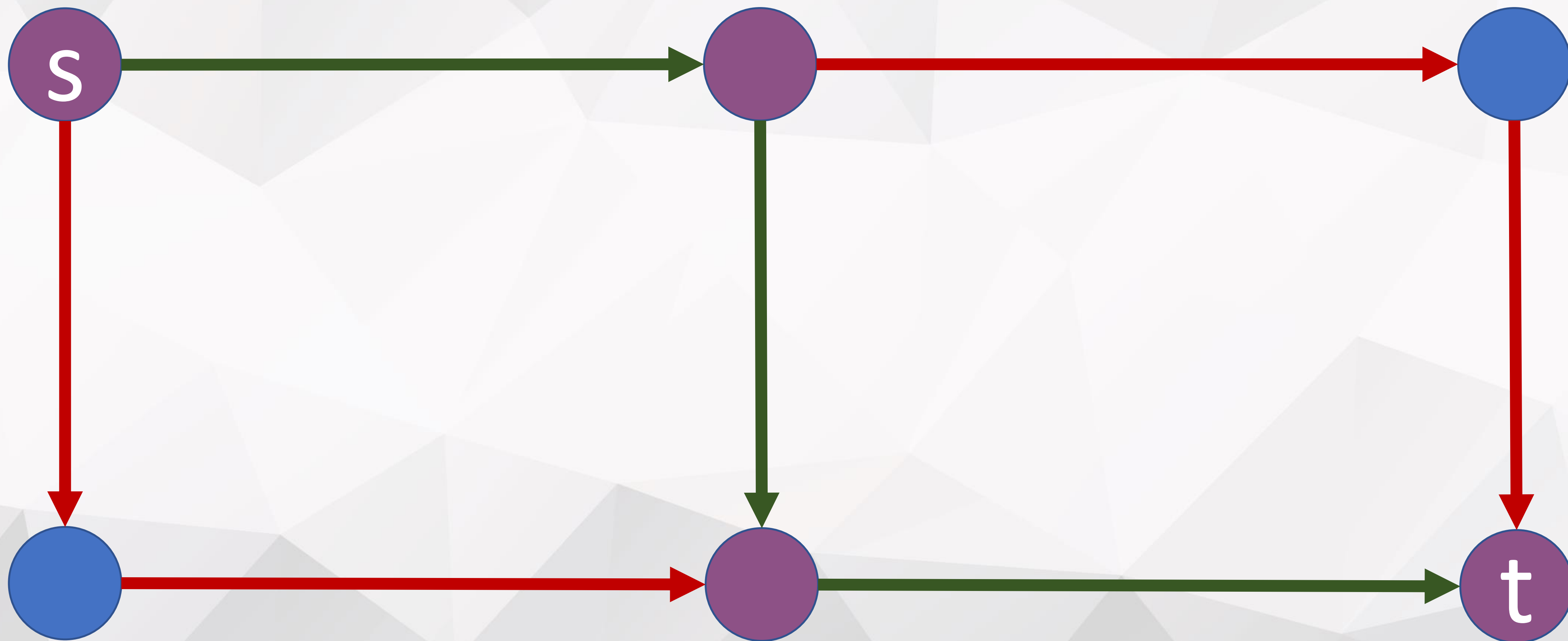
流量

0



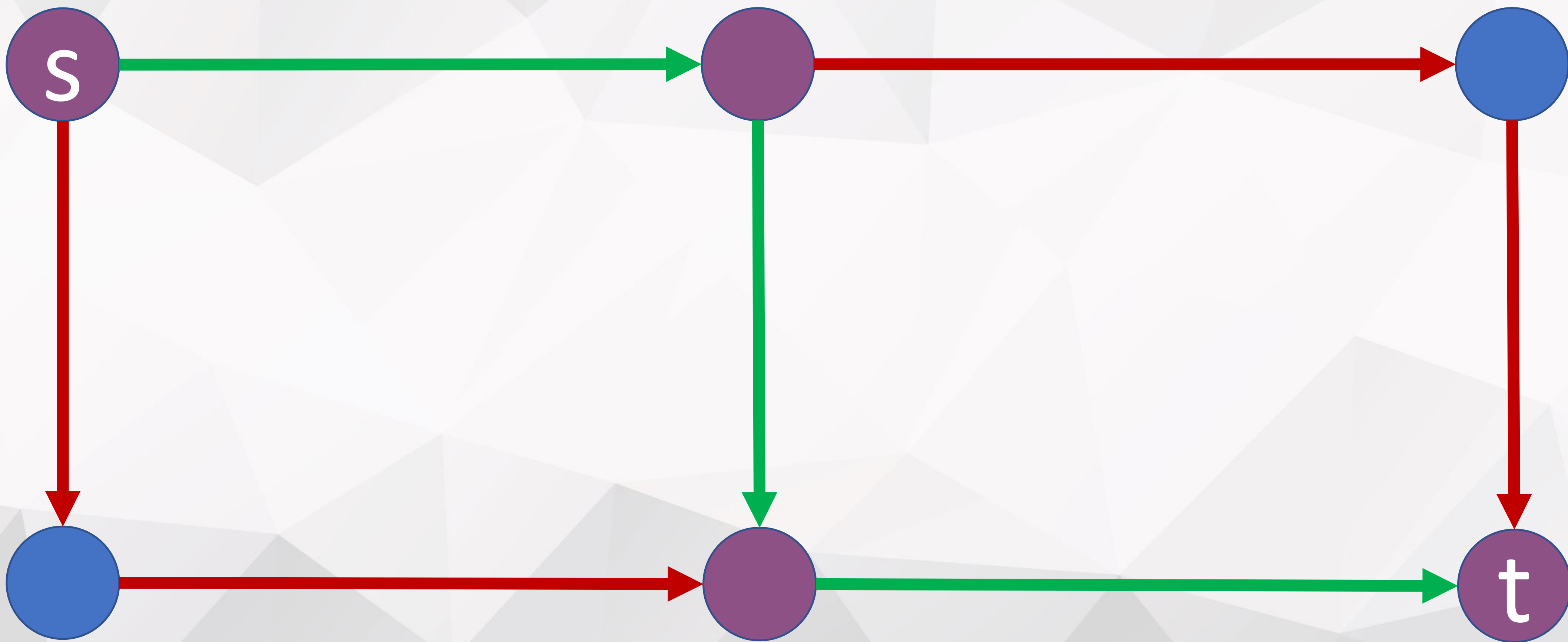
流量

2

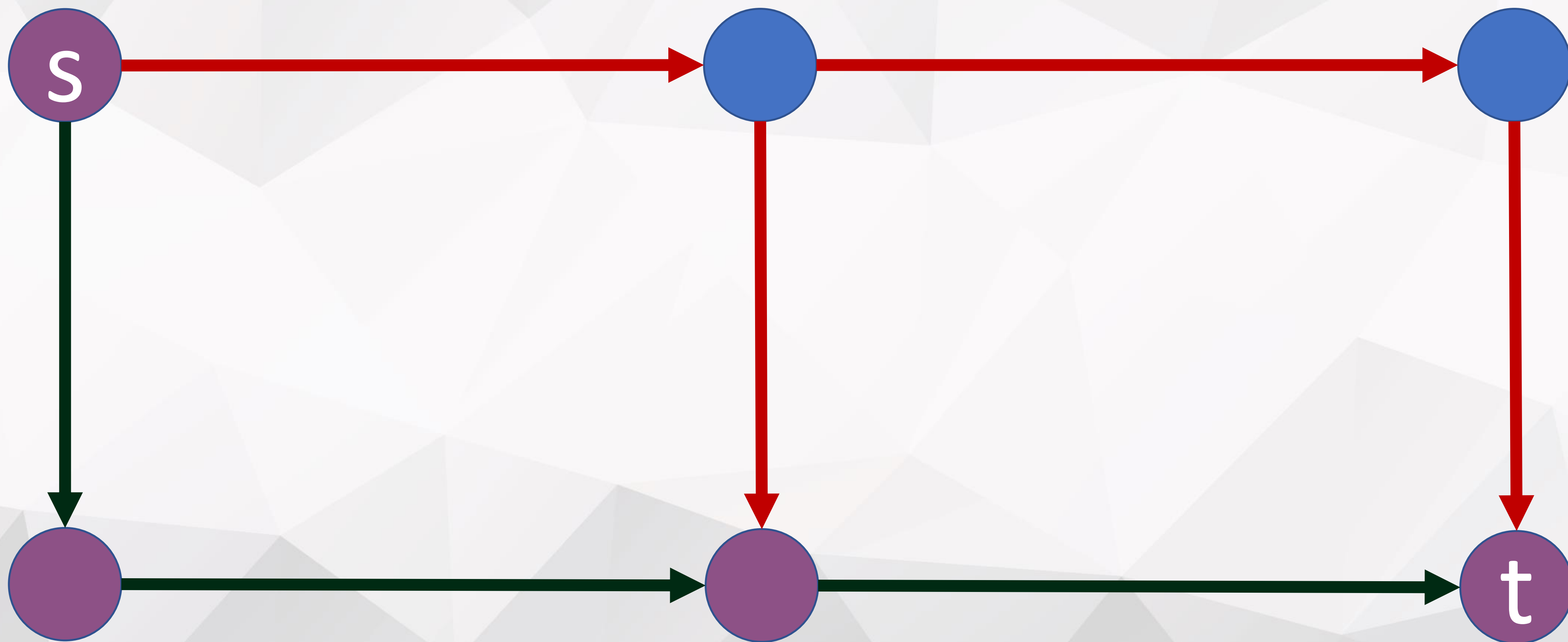


流量 (不必多)

1

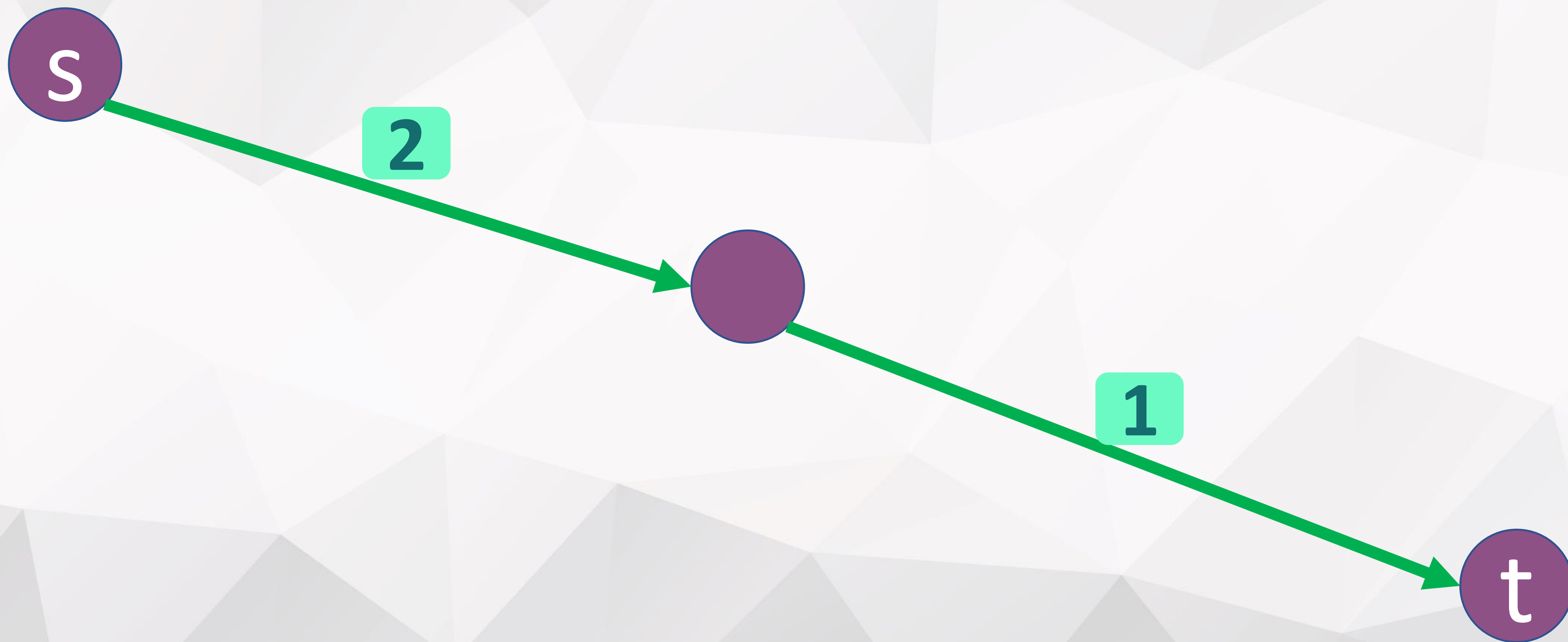


流量



流量

1

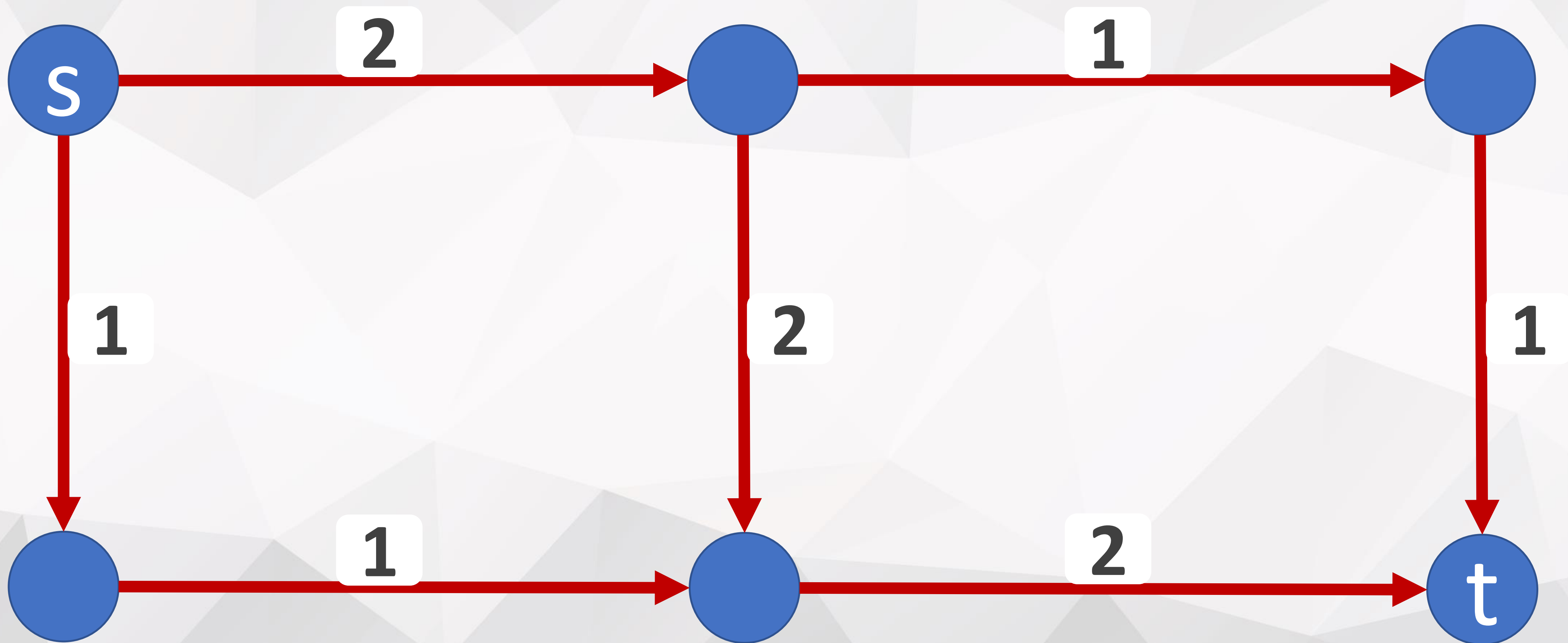


前提

- 流量
- 容量
- 剩餘容量

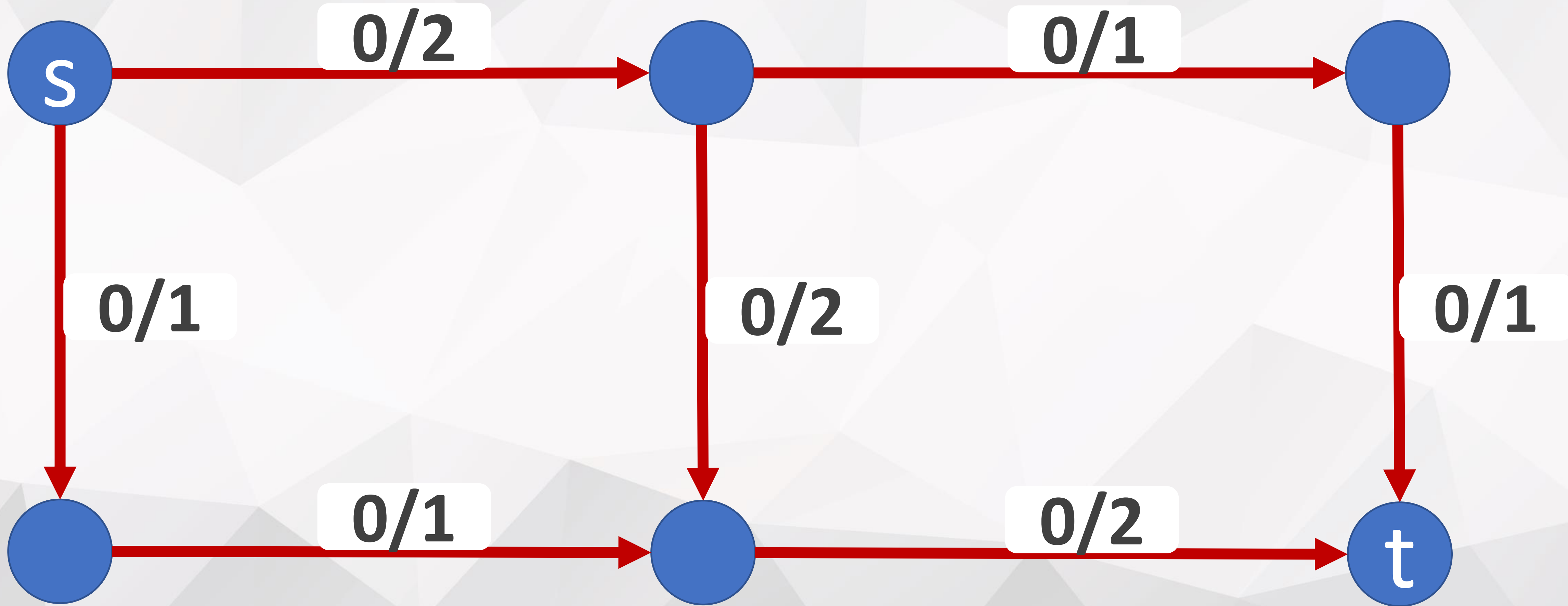
容量

0



流量/容量

0

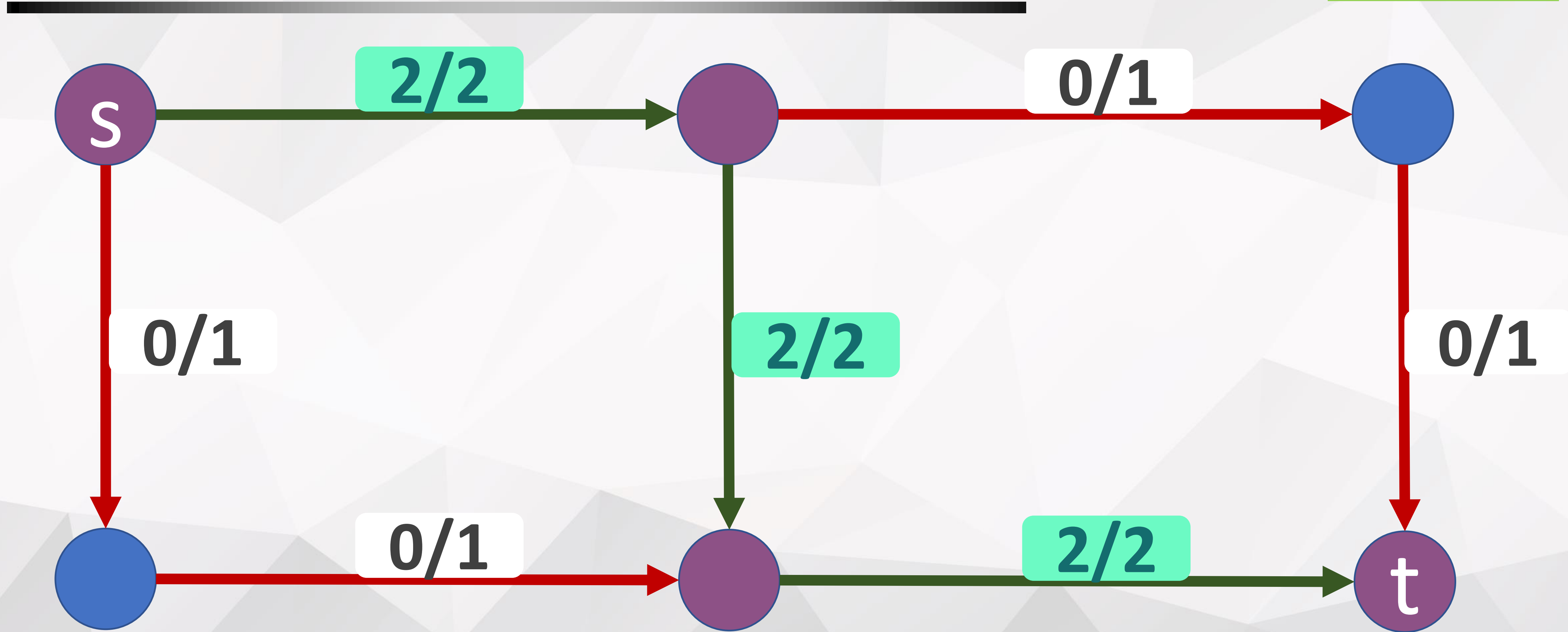


x/y

- x := 流量
- y := 容量

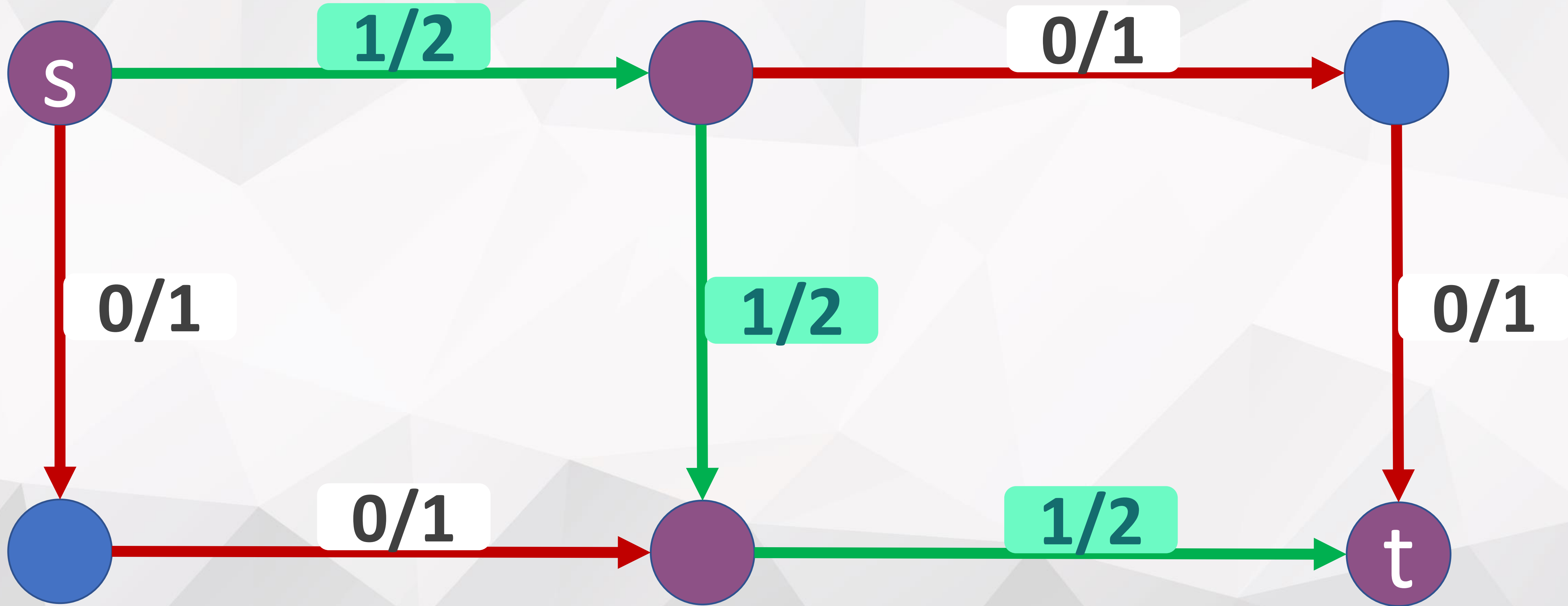
流量/容量

2



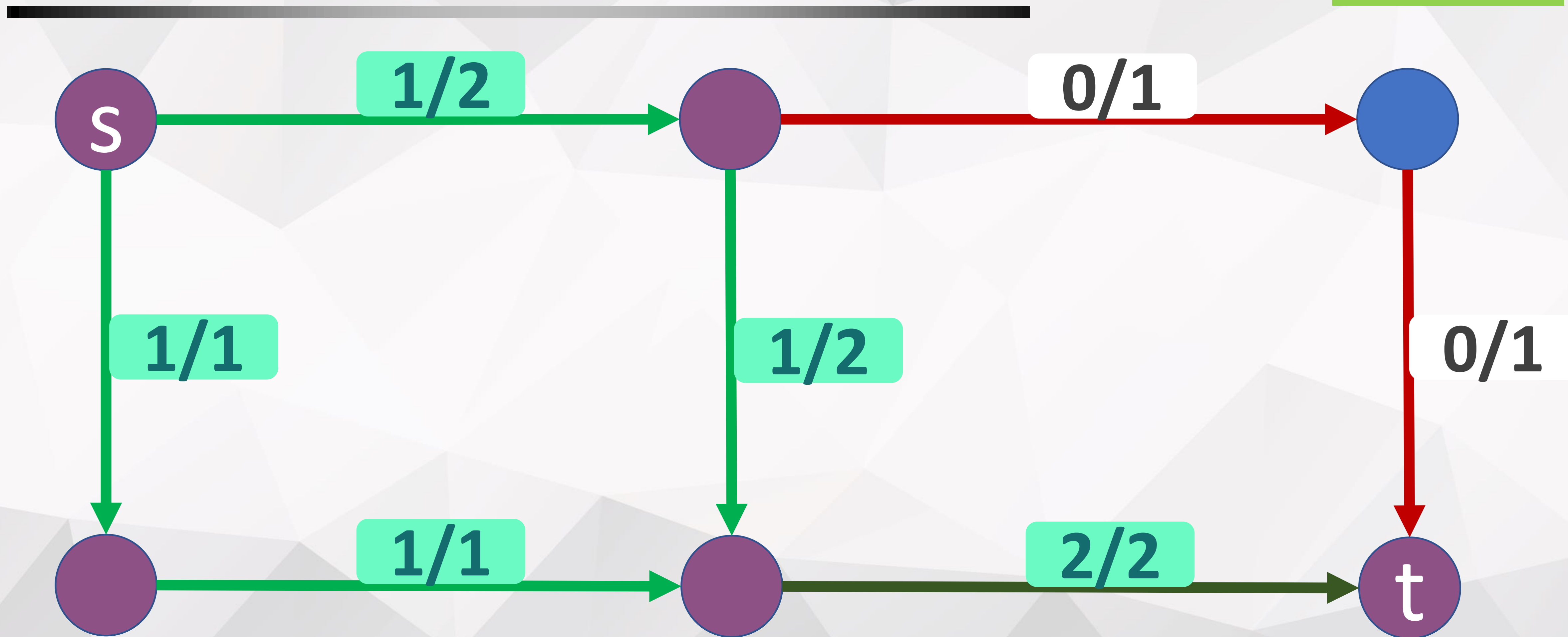
流量/容量

1



流量/容量

2



前提

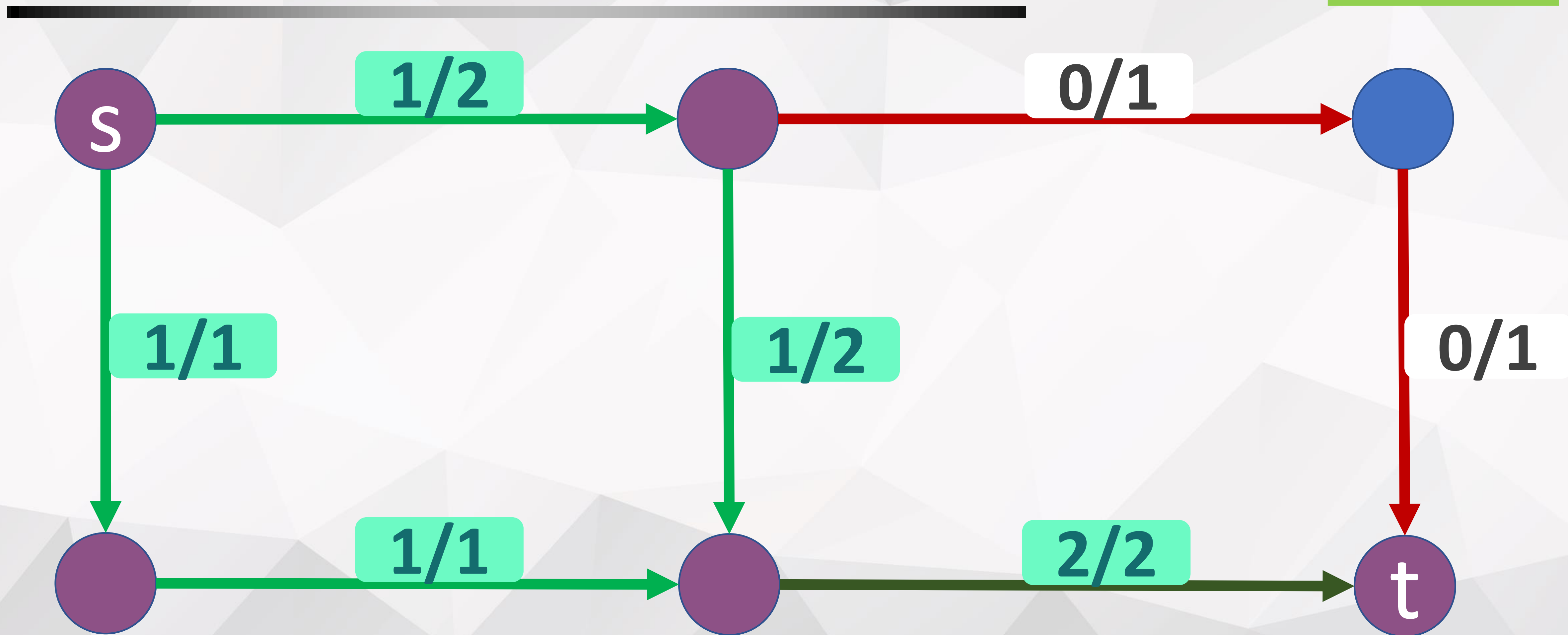
- 流量
- 容量
- 剩餘容量

x/y

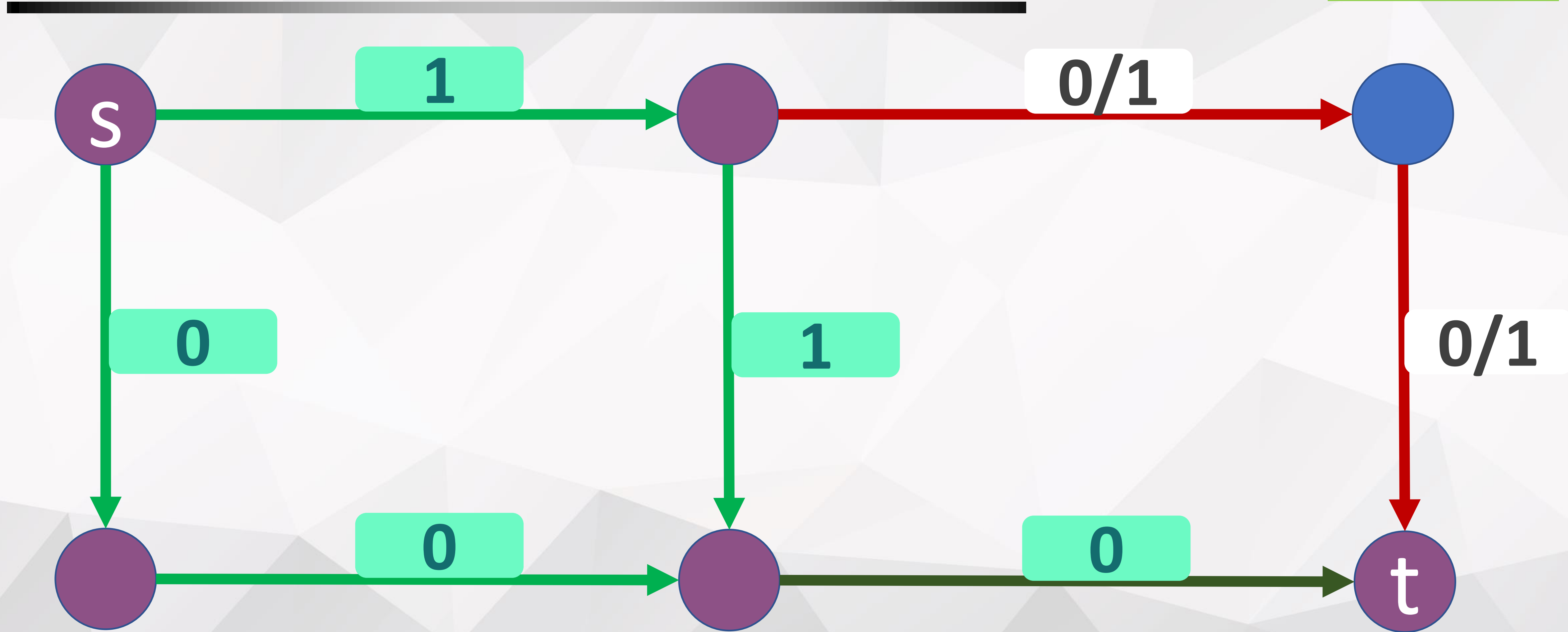
- x := 流量
- y := 容量
- y-x = 剩餘容量

流量/容量

2



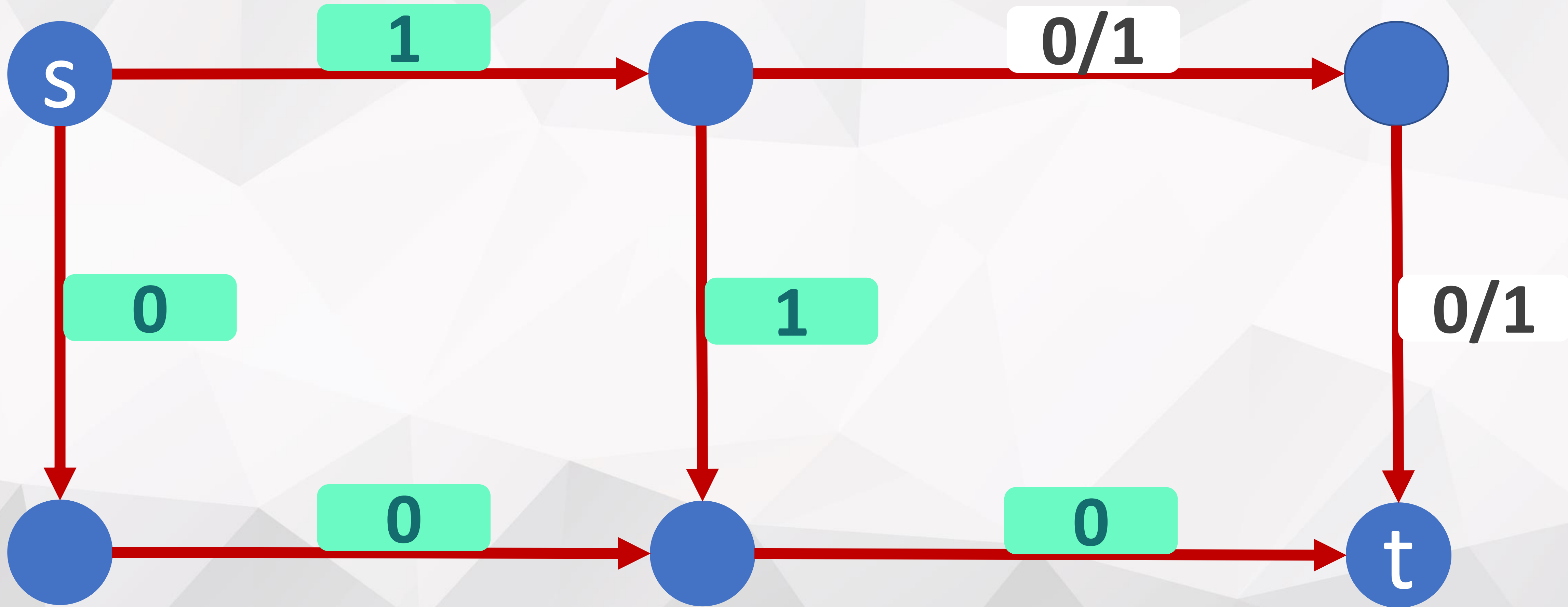
剩餘容量



2

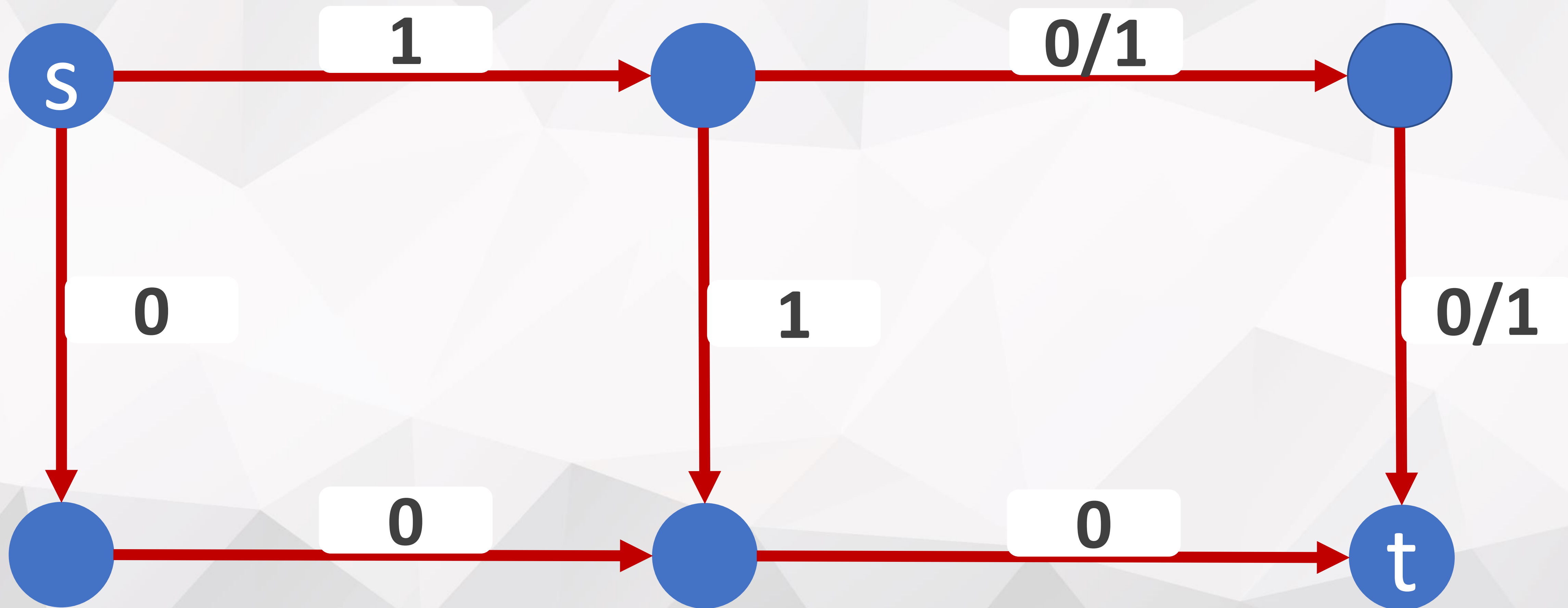
剩餘容量

2

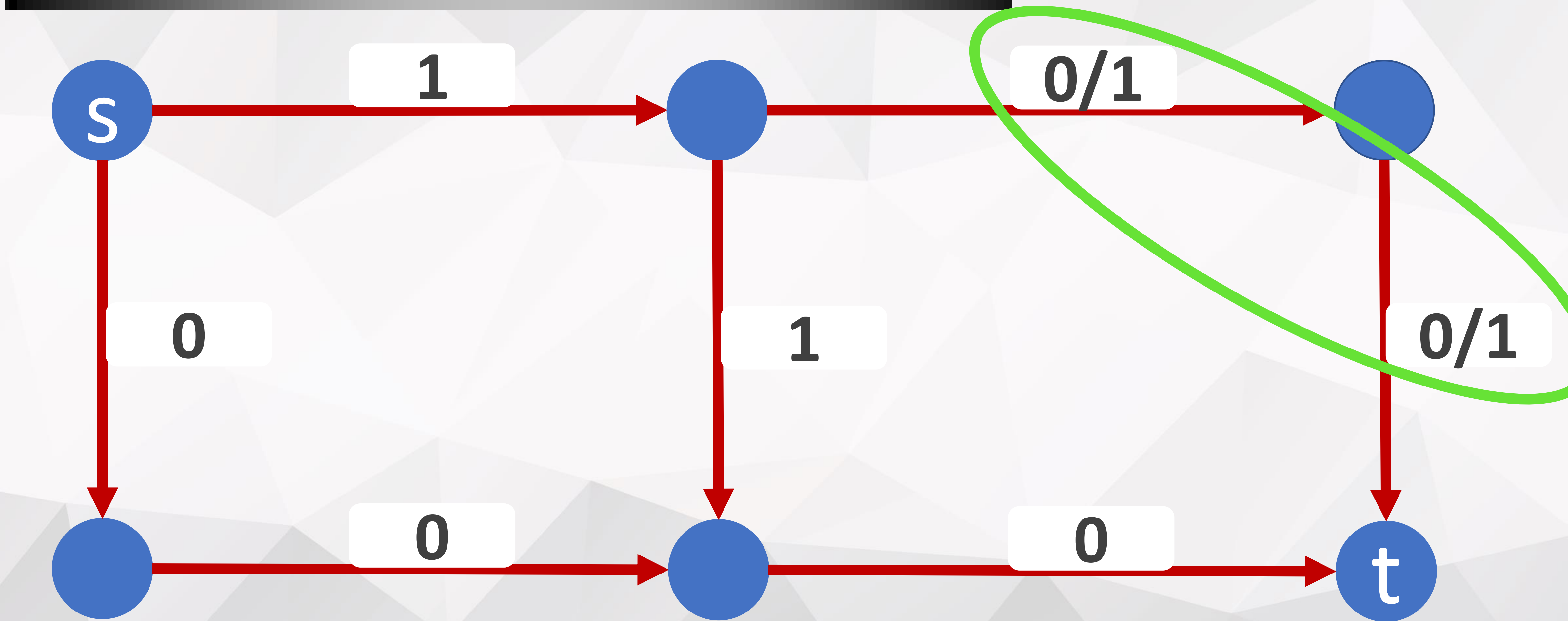


剩餘容量

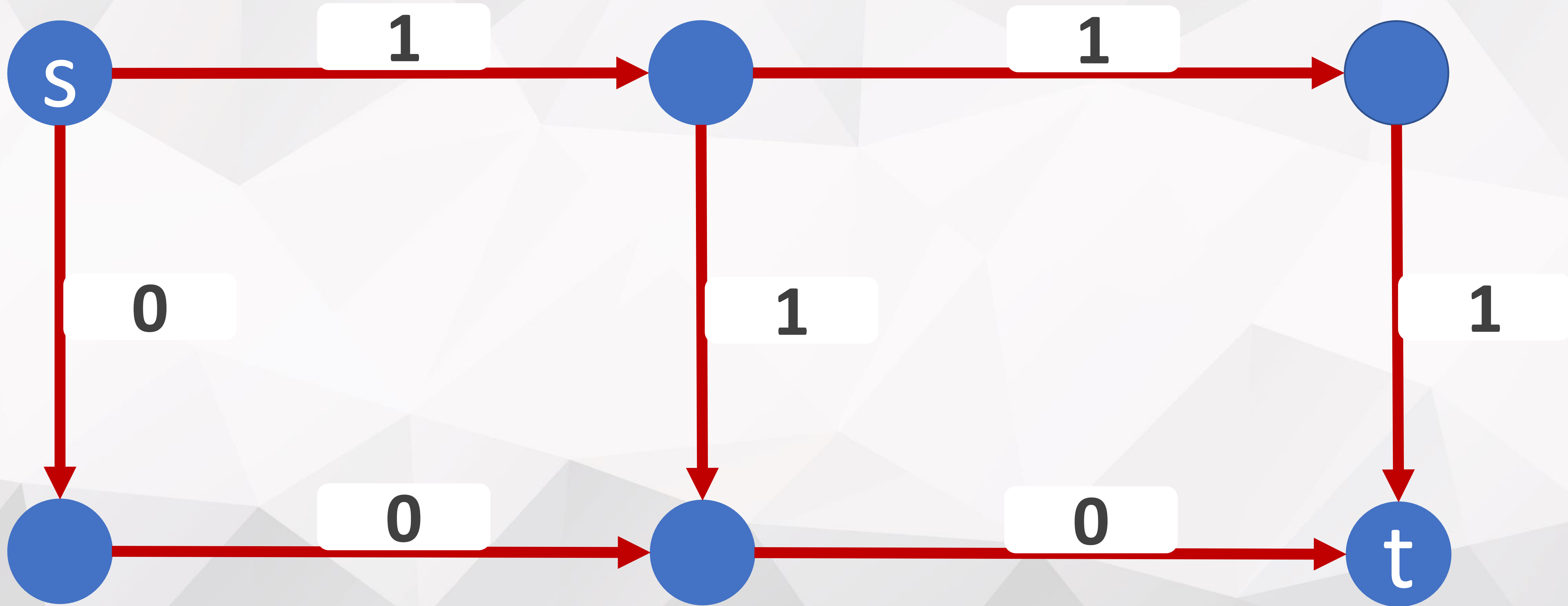
2



剩餘容量

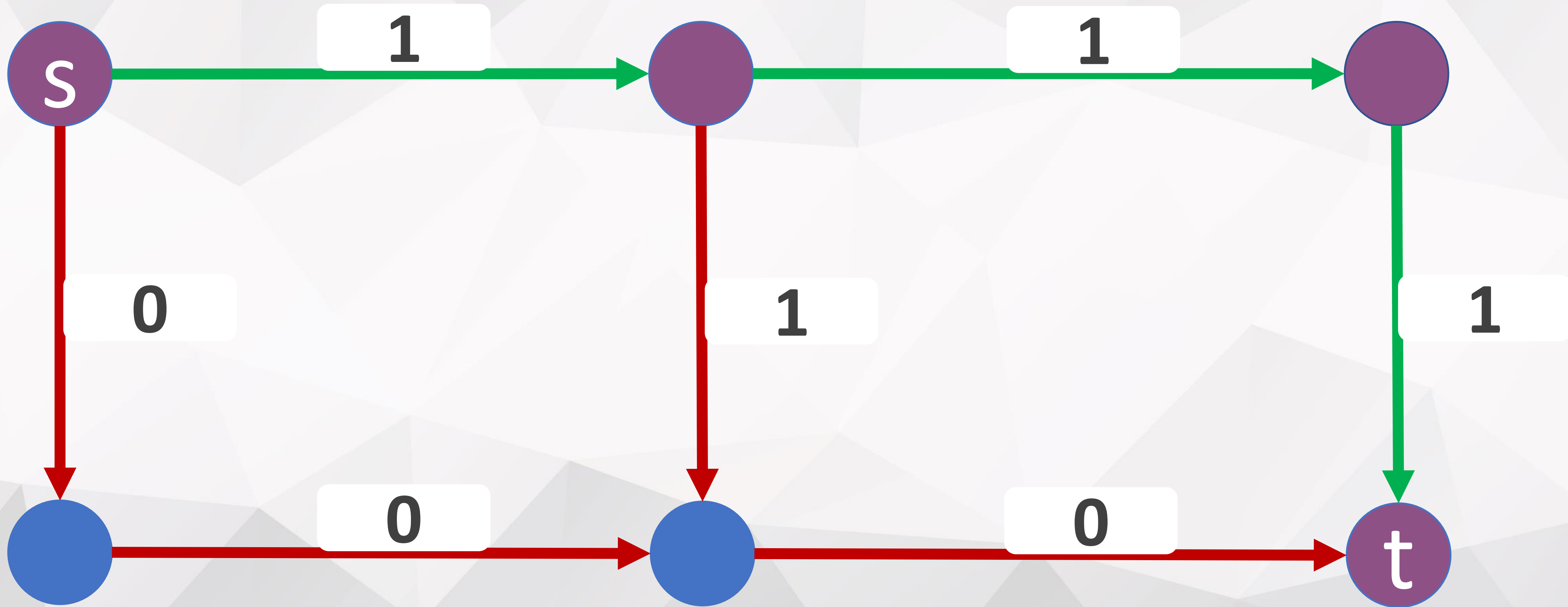


剩餘容量



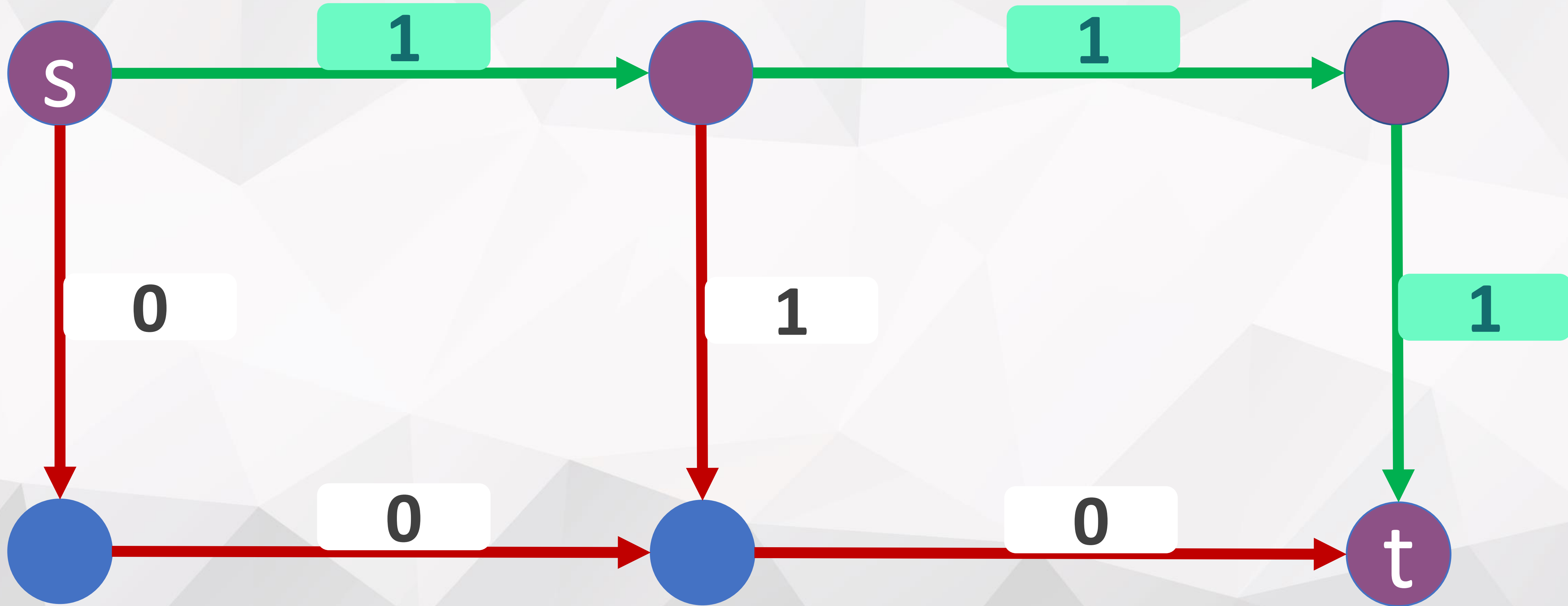
Augmenting path

2



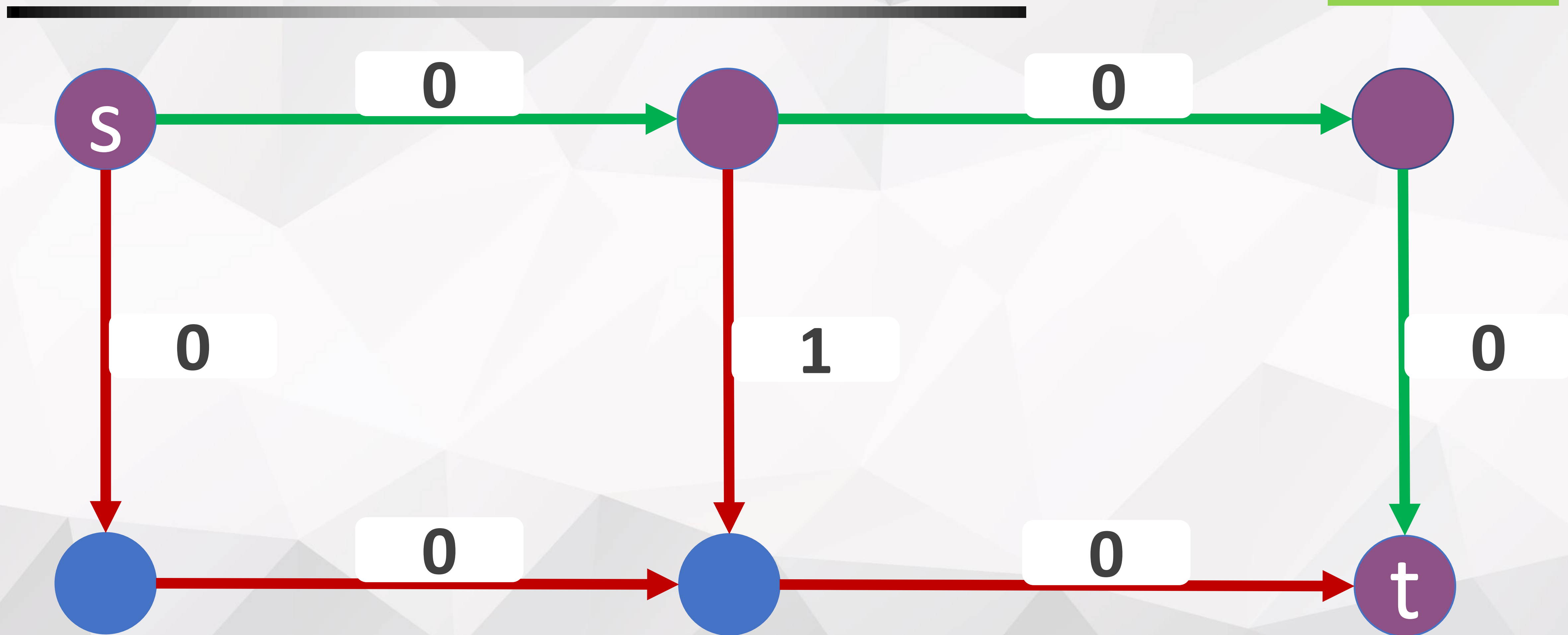
流量: 1

2



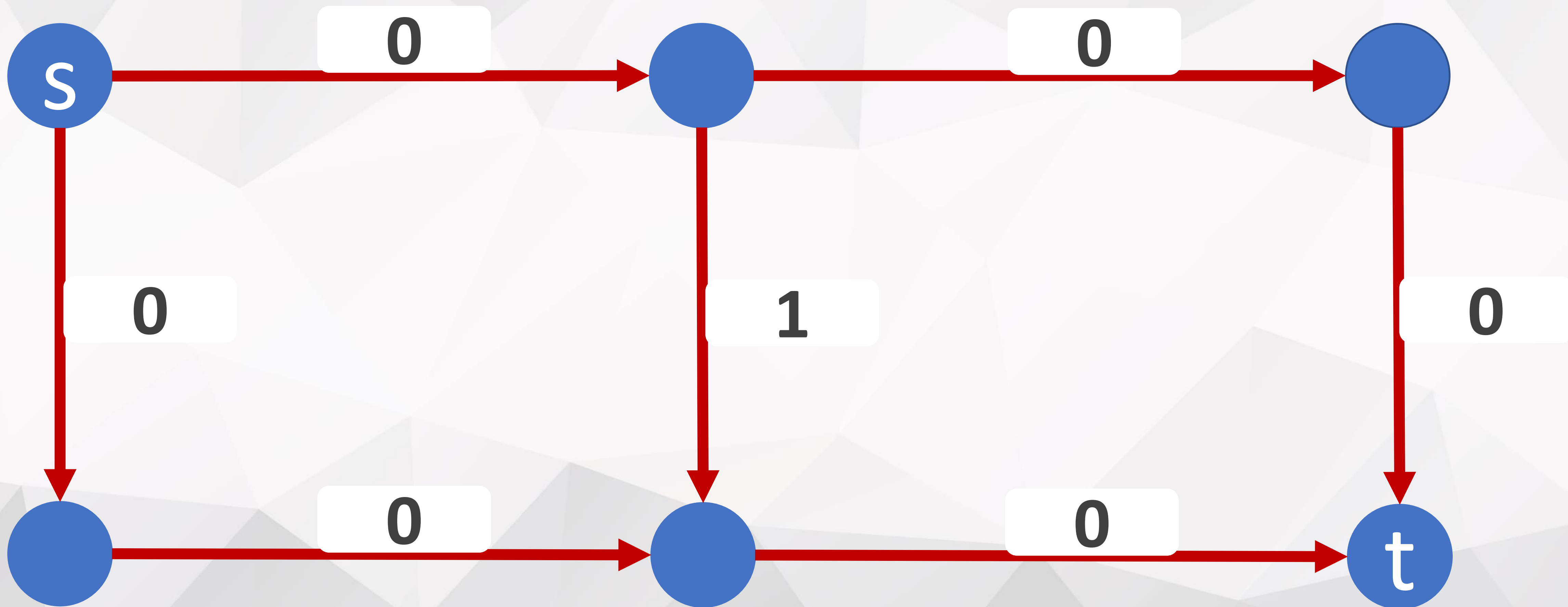
剩餘容量

3



剩餘容量

3



最大流

給定帶權重連通圖

找到此圖從 s 點 到 t 點的最大流量

最大流

給定帶權重連通圖

找到此圖從 s 點 到 t 點的最大流量

剛剛的例子，最大流就是 3

找出最大流

- Augmenting path
- Ford-Fulkerson method
- Edmonds-Karp algorithm

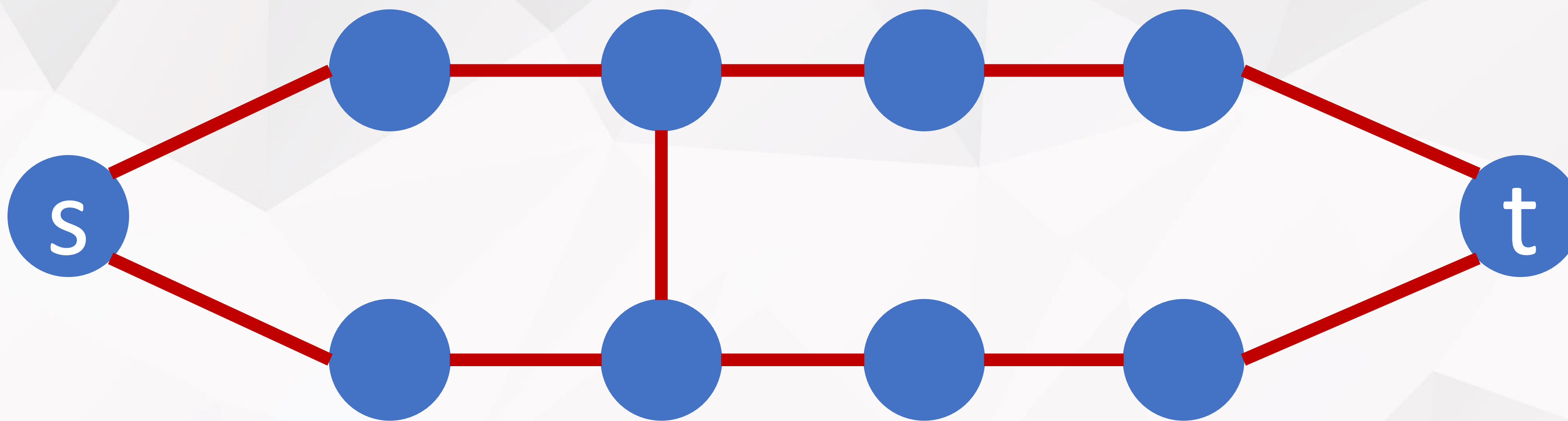
找出最大流

- Augmenting path
- Ford-Fulkerson method
- Edmonds-Karp algorithm

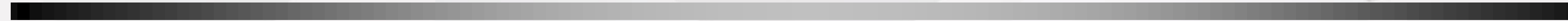
Augmenting path

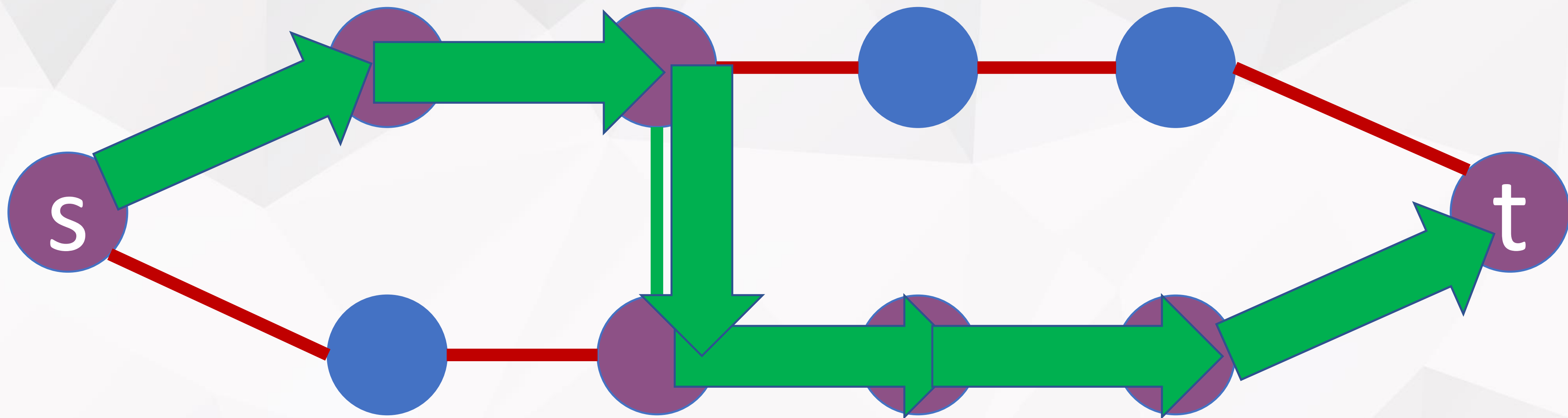
指的是一條路徑

它的剩餘容量
足夠讓大於 0 的流量通過

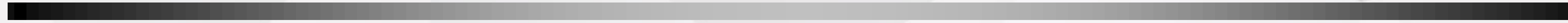

s
t

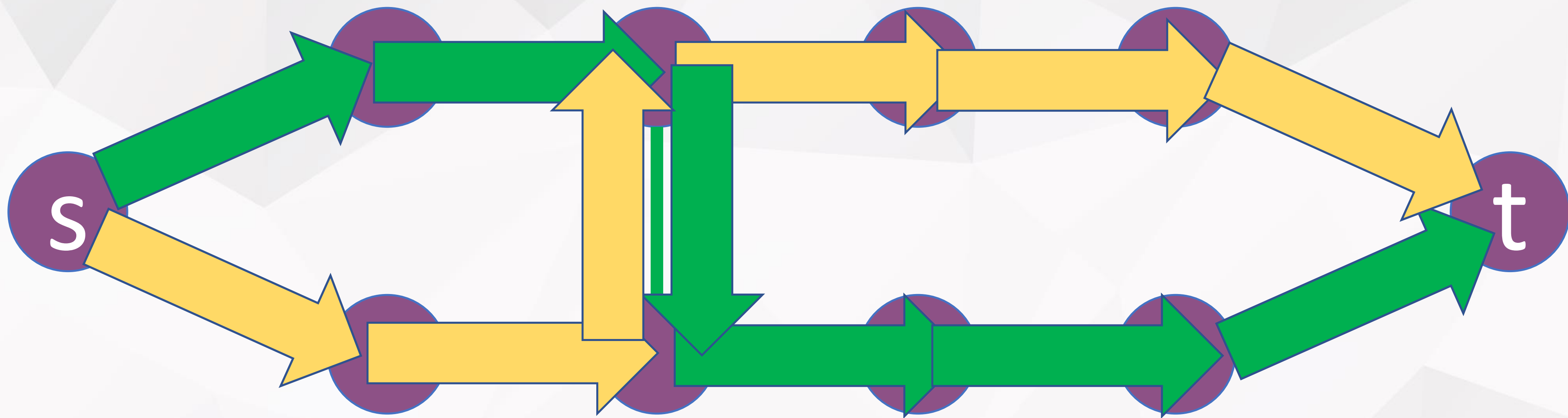
F_1



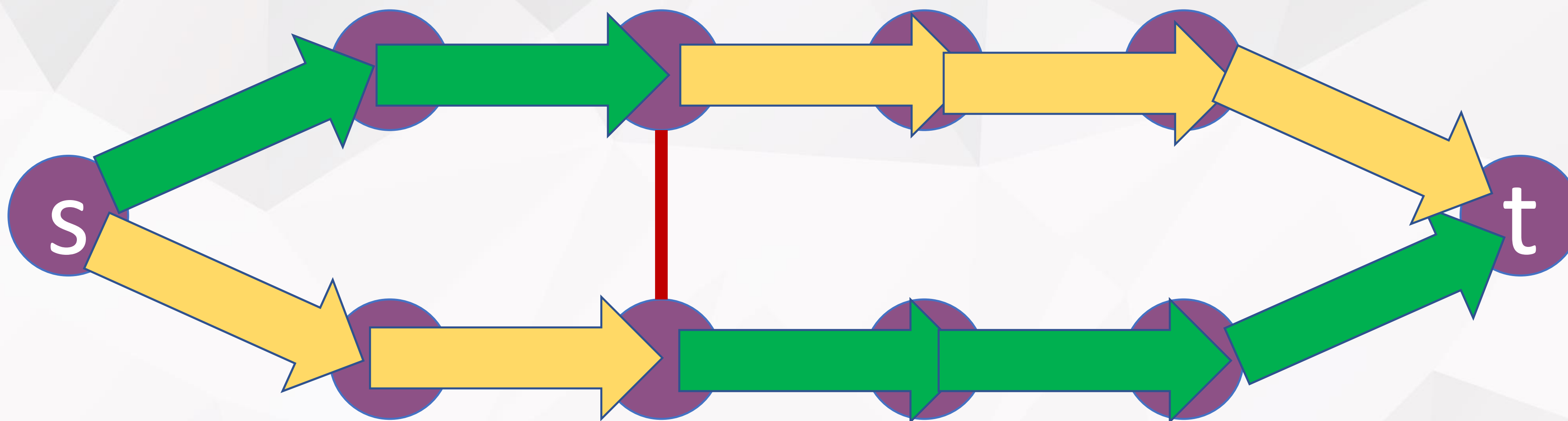


F_2

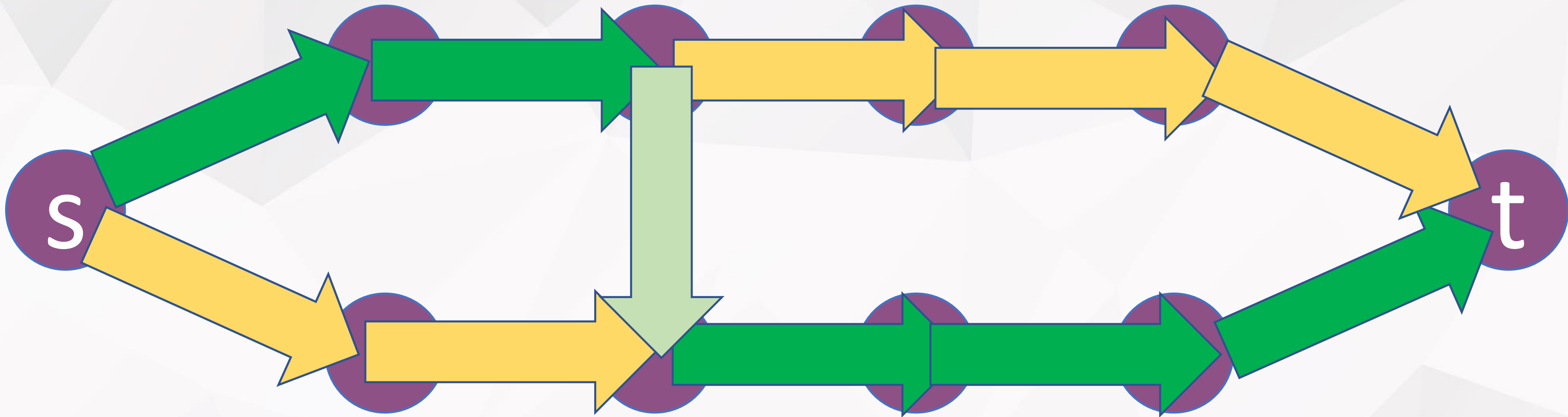




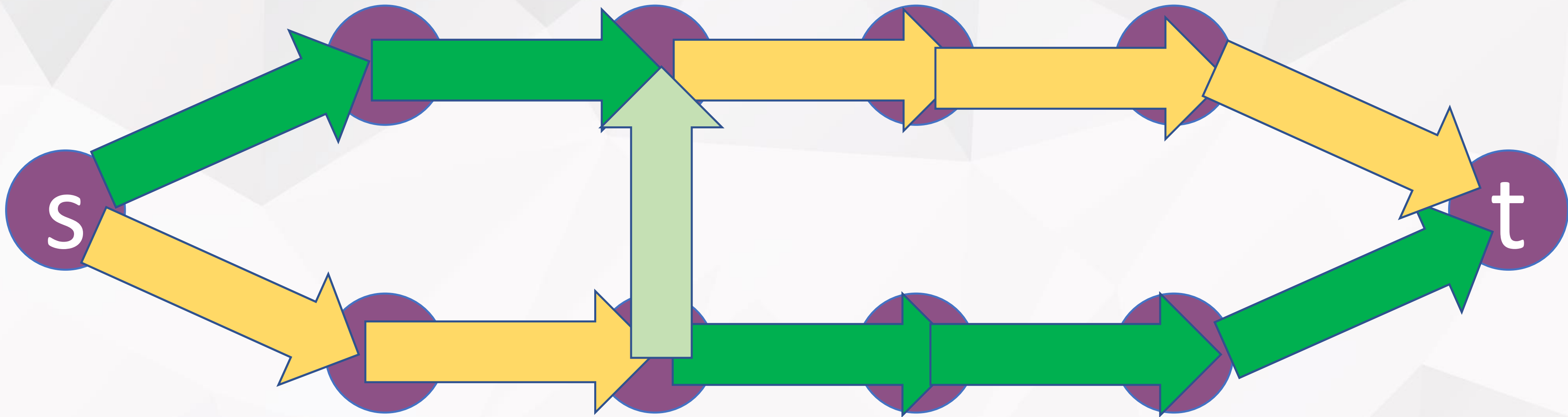
$F_1 = F_2$



$F_1 > F_2$

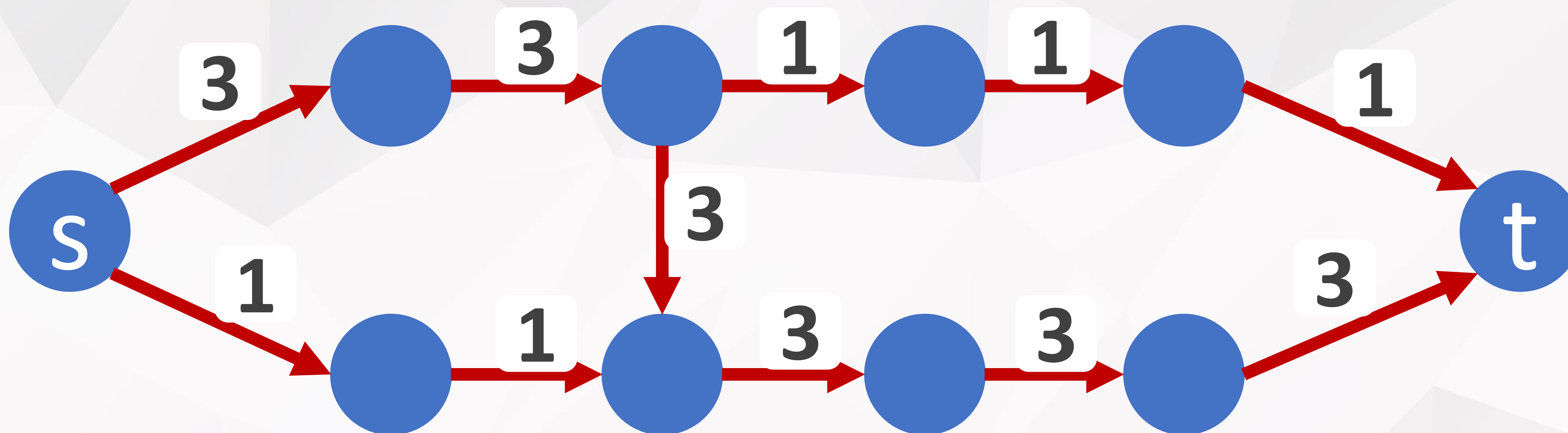


$F_1 < F_2$



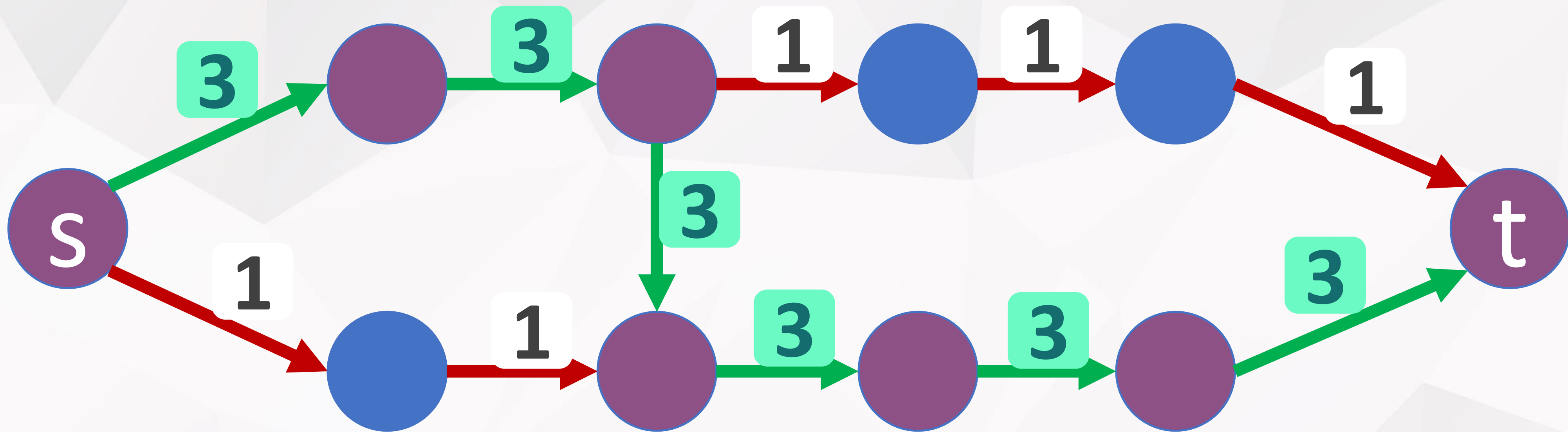
具體來看

0



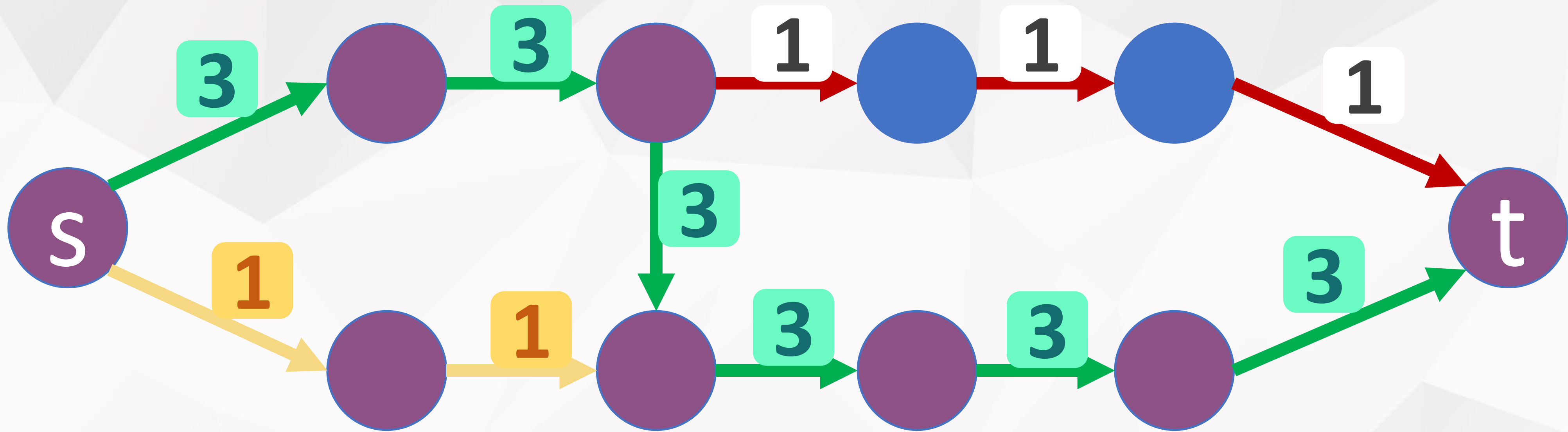
F_1

3



F_2

3



流不過去

因為沒有**剩餘容量**了!!

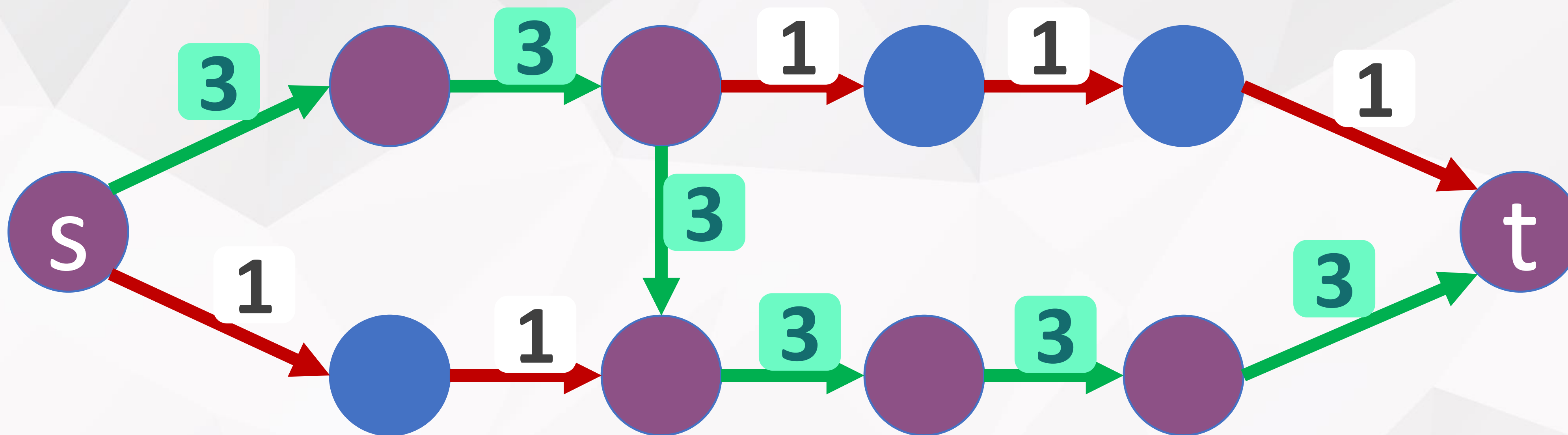
流不過去

因為沒有**剩餘容量**了!!

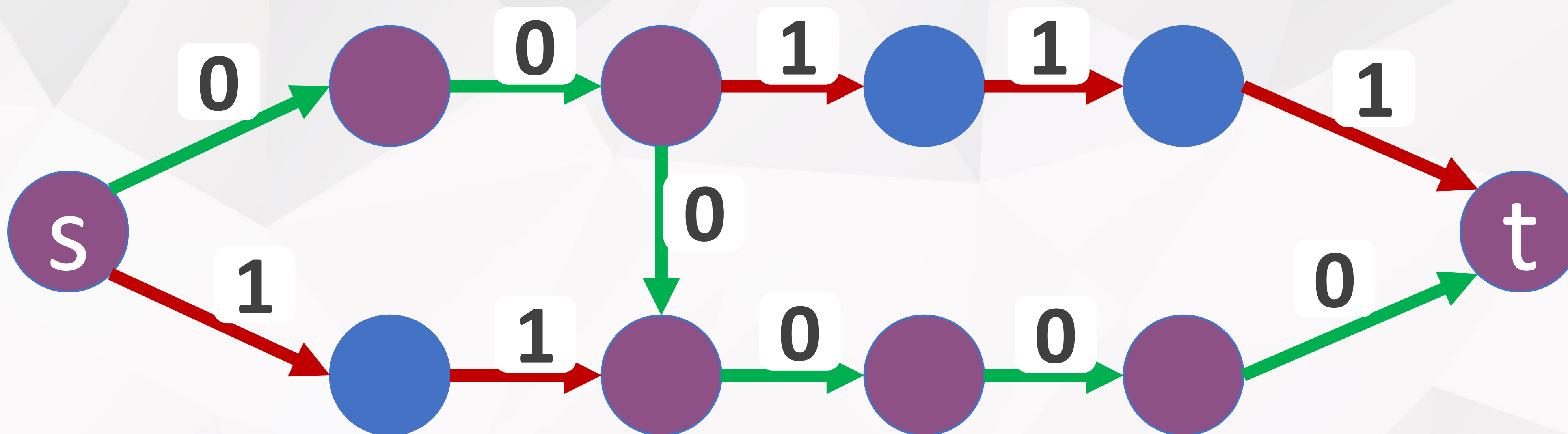
怎麼辦?

F_1 過後，剩餘容量

3

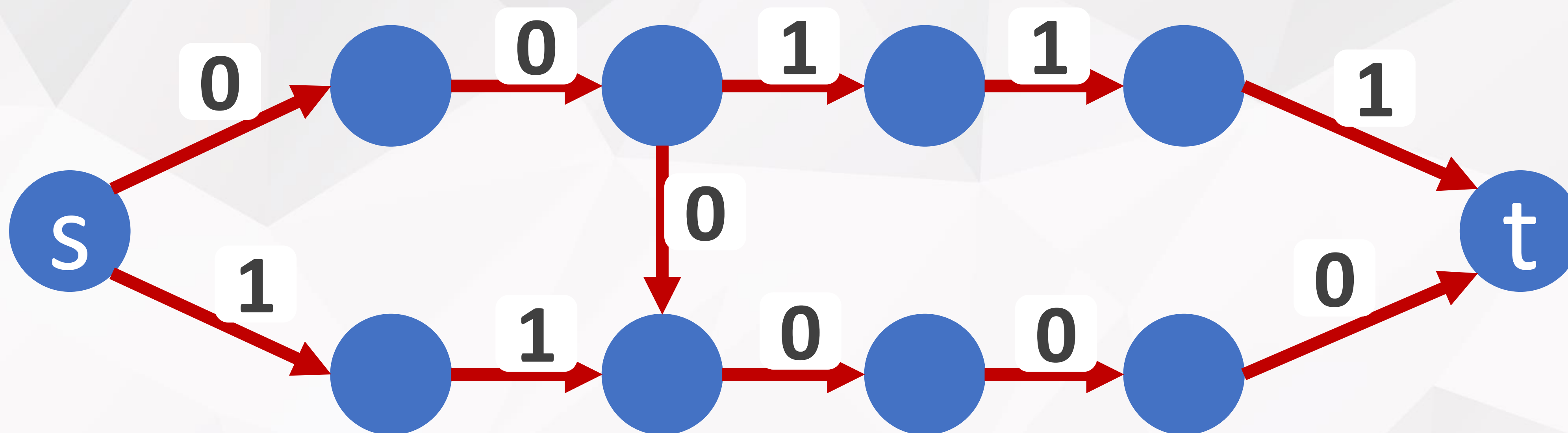


F_1 過後，剩餘容量



F_1 過後，剩餘容量

3



流不過去

因為沒有**剩餘容量**了!!

怎麼辦?

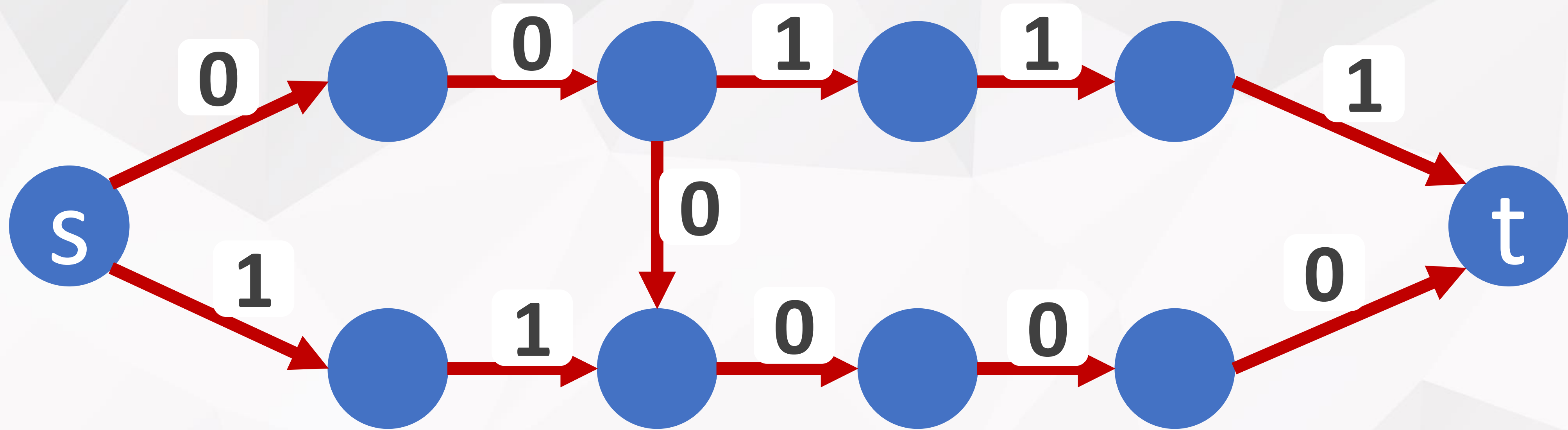
流不過去

因為沒有**剩餘容量**了!!

怎麼辦?

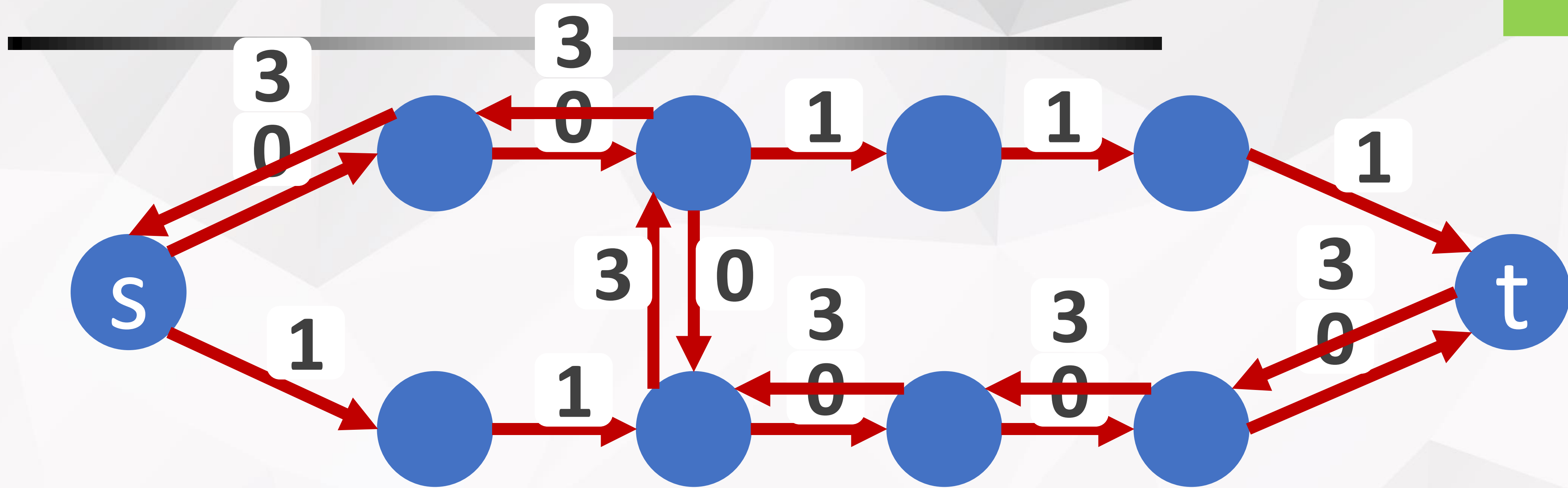
為了 F_2 ，也給他一個**剩餘容量**

剩餘容量



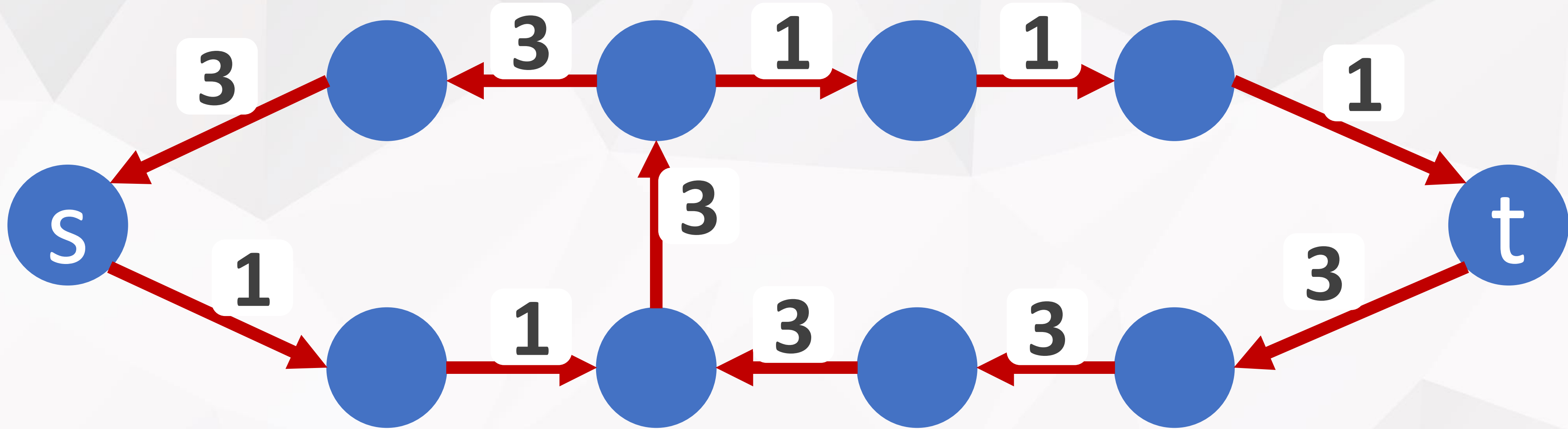
剩餘容量

3



剩餘容量

3



流得過去!!

跟水流一樣

流得過去!!

跟水流一樣

如果第二條水流更強，就有機會壓過第一條水流

流得過去!!

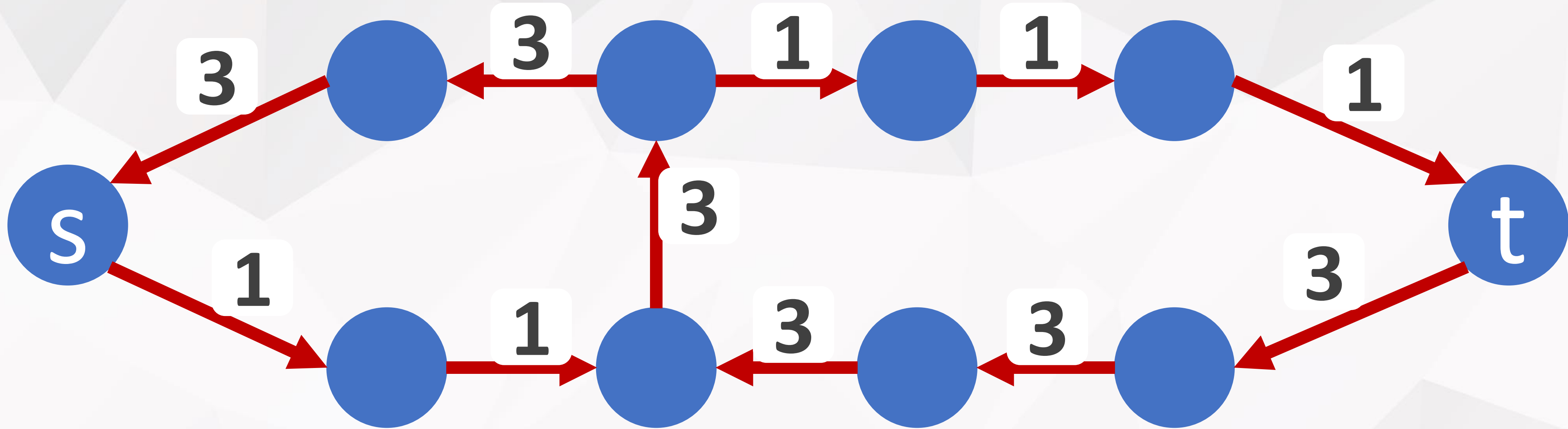
跟水流一樣

如果第二條水流更強，就有機會壓過第一條水流

給他**剩餘容量**，就是給流向逆轉的**機會**
~~跟八卦版很像，偶爾也會出現風向逆轉的傾向~~

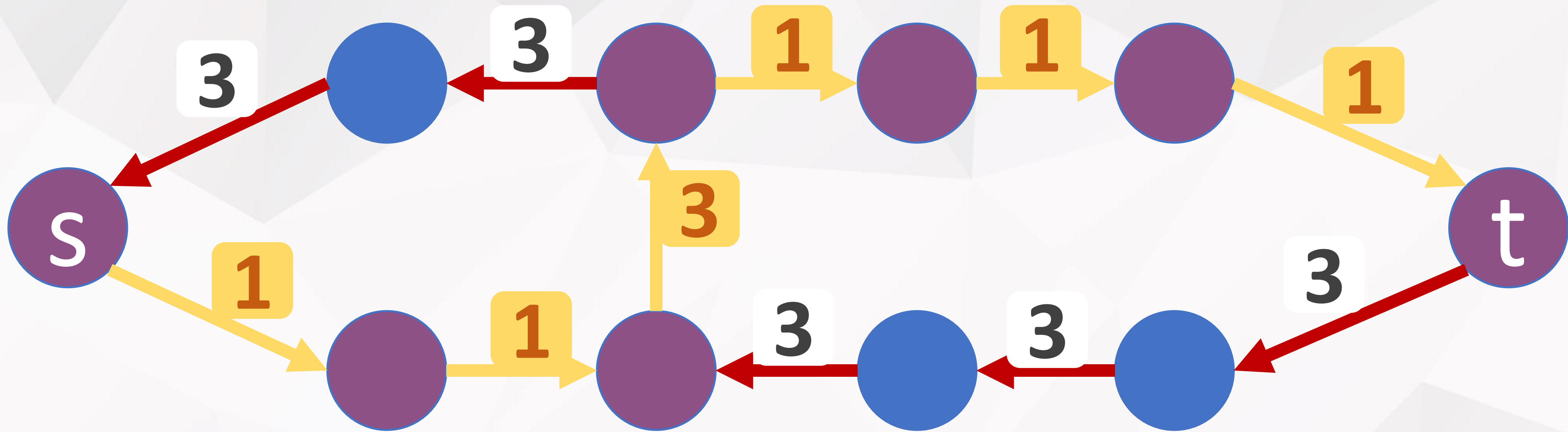
剩餘容量

3



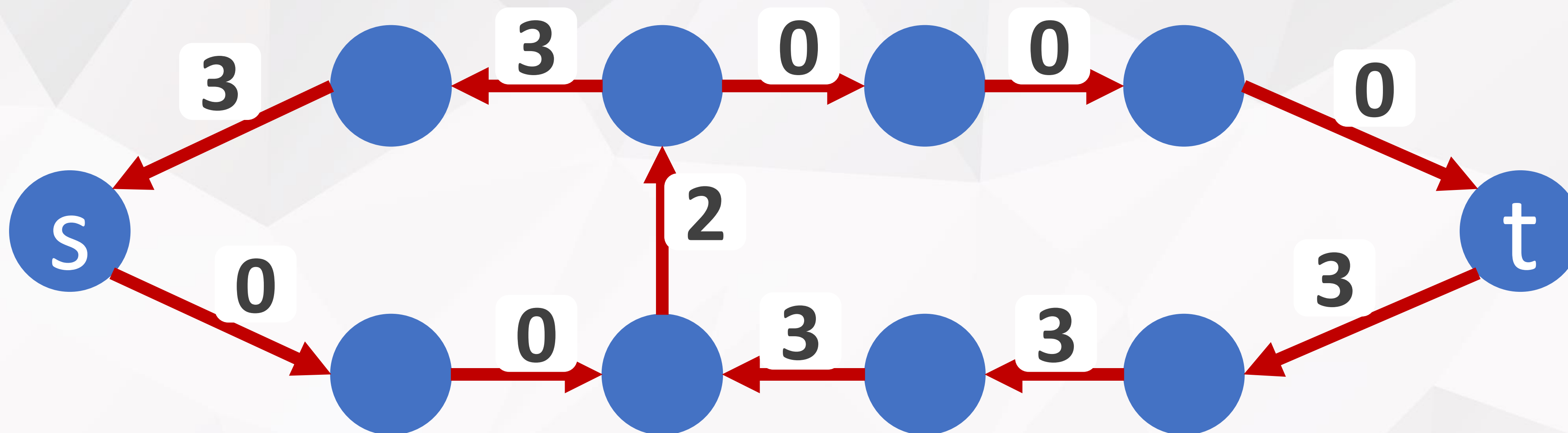
F_2

4



剩餘容量

4



與 F_1 同樣

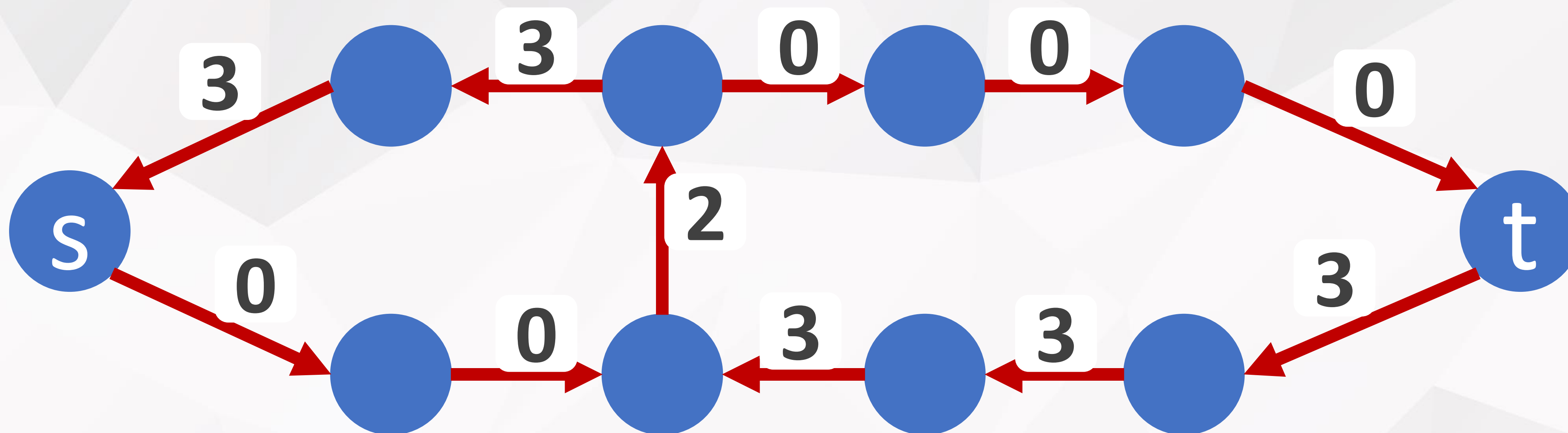
給下一條流**機會**

所以**反向邊**都得給予**同等**的**剩餘容量**

流量多少就給多少

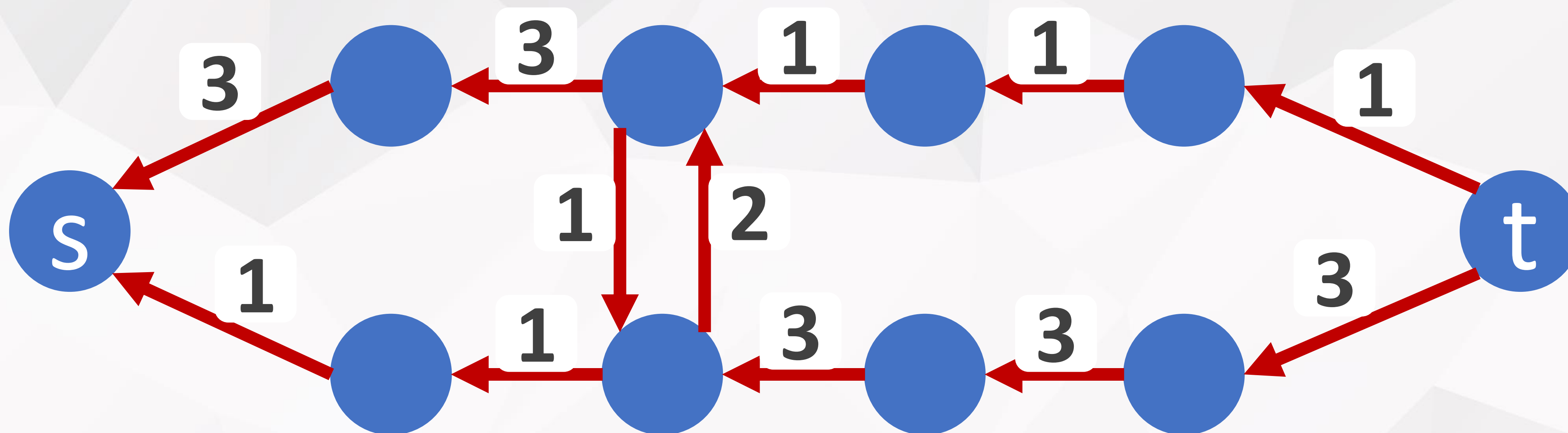
剩餘容量

4



剩餘容量

4



Augmenting path

指的是一條路徑

它的剩餘容量
足夠讓大於 0 的流量通過

顯然已經找不到這條路徑了
故原圖的**最大流**為 4

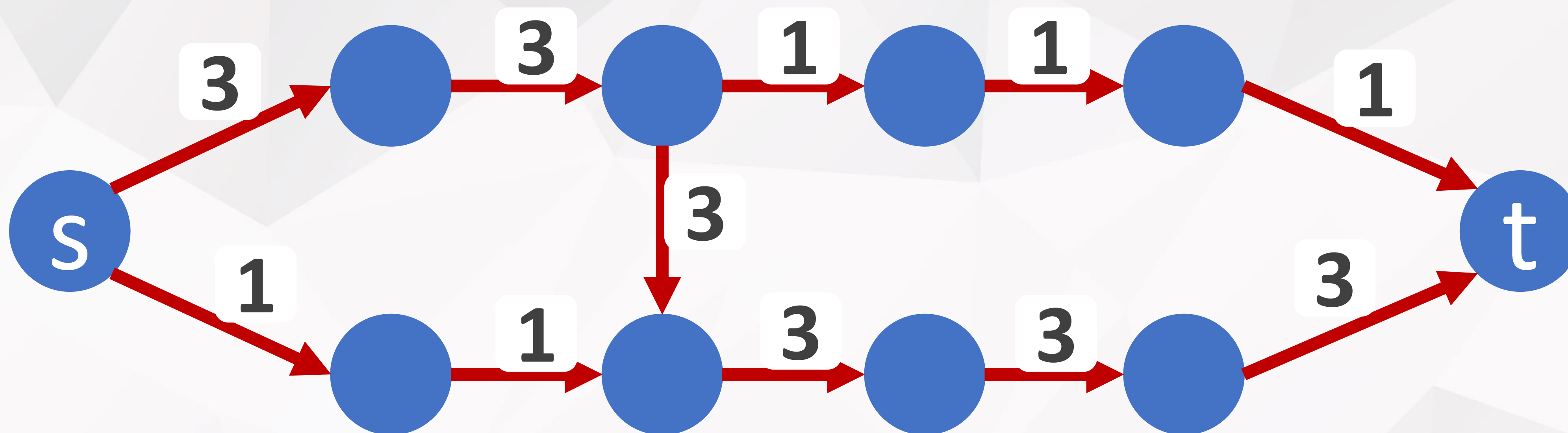
兩個流

實際上相抵之後

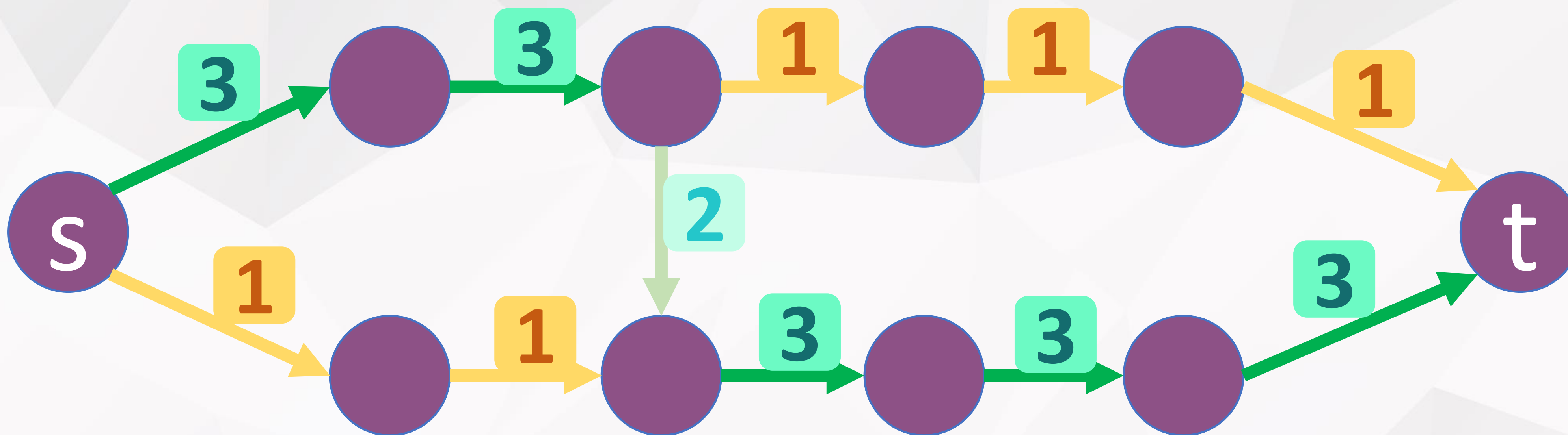
F_1 比 F_2 來得大，
所以在中間相遇部分會有 $F_3 = F_1 - F_2$

剩餘容量

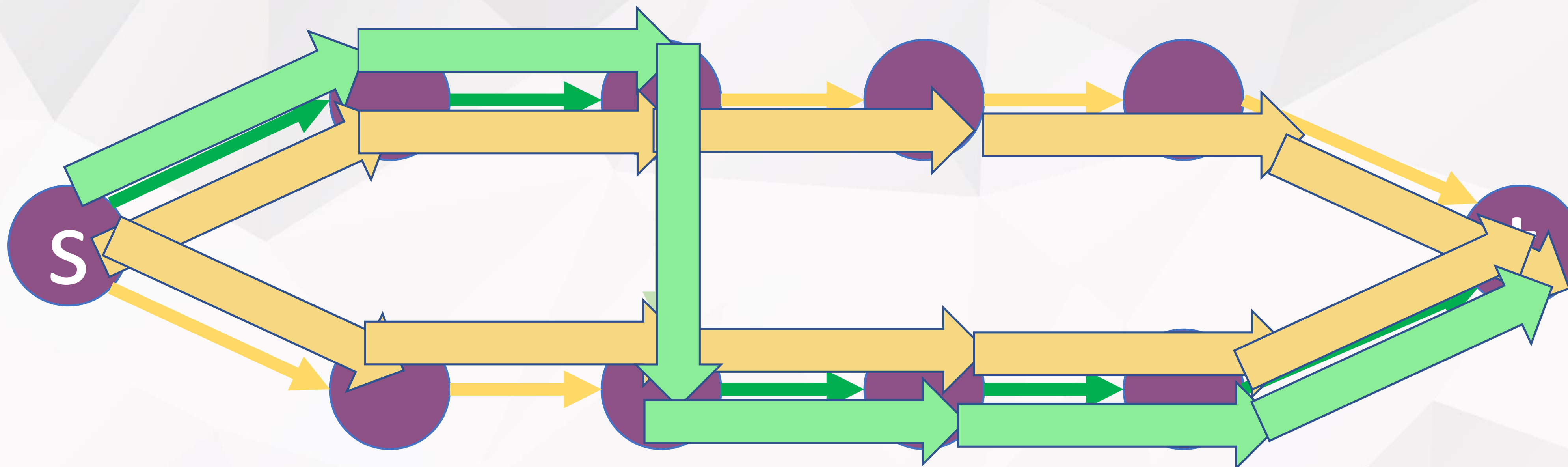
0



流量圖 (權重是流量)



三條流



找出最大流

- Augmenting path
- Ford-Fulkerson method
- Edmonds-Karp algorithm

Ford-Fulkerson method

可以討論如何找到最大流了!

直覺的，對於**存在** augmenting path 的圖

Ford-Fulkerson method

可以討論如何找到最大流了!

直覺的，對於存在 augmenting path 的圖
給他**流過去**就對了!

Ford-Fulkerson method

可以討論如何找到最大流了!

直覺的，對於存在 augmenting path 的圖
給他流過去就對了!

流到**不再能找到 augmenting path**

Ford-Fulkerson method

可以討論如何找到最大流了!

直覺的，對於存在 augmenting path 的圖
給他流過去就對了!

流到不再能找到 augmenting path
也就是流量只能得到 0

找出最大流

- Augmenting path
- Ford-Fulkerson method
- Edmonds-Karp algorithm

Edmonds-Karp algorithm

如何找到 augmenting path?

Edmonds-Karp algorithm

如何找到 augmenting path?

Edmonds-Karp 用 BFS

Edmonds-Karp algorithm

如何找到 augmenting path?

Edmonds-Karp 用 BFS

每當**剩餘容量 >0** 路徑就能延伸

Edmonds-Karp algorithm

如何找到 augmenting path?

Edmonds-Karp 用 BFS
每當**剩餘容量 >0** 路徑就能延伸

原則就是幹你娘流爆 效仿 Ford-Fulkerson method
~~跟產薯條的原則類似~~

實作

```
int max_flow = 0;
```

實作

```
while (1) {
    ⋮
    if (flow[t] == 0) break;
    ⋮
}
```

實作

```
memset(vis, false, sizeof(vis));
vis[s] = true;

memset(flow, 0, sizeof(flow));
flow[s] = INF;

queue<int> Q;
Q.push(s);
```


Made by 培訓團隊群 CCNS

實作

```
while (!Q.empty()) {
  int u = Q.front(); Q.pop();

  for (int v = s; v <= t; v++) {
    if (!R[u][v] || vis[v]) continue;

    vis[v] = true;
    Q.push(v);

    pre[v] = u; // 紀錄 augmenting path
    flow[v] = min(flow[u], R[u][v]); // 流只挑最小的剩餘容量
  }
}
```

實作

```
max_flow += flow[t];

for (int v = t, u; v != s; v = u) {
    u = pre[v];

    R[u][v] -= flow[t];
    R[v][u] += flow[t];
}
```

Questions?

練習

UVa OJ 820 Internet Bandwidth

Articulation Point

關節點

Competitive Programming

Made by 培訓團隊群 CCNS.

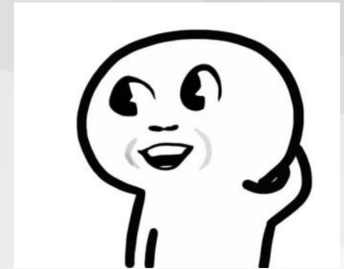
比較有向圖與無向圖「兩點間的關係」

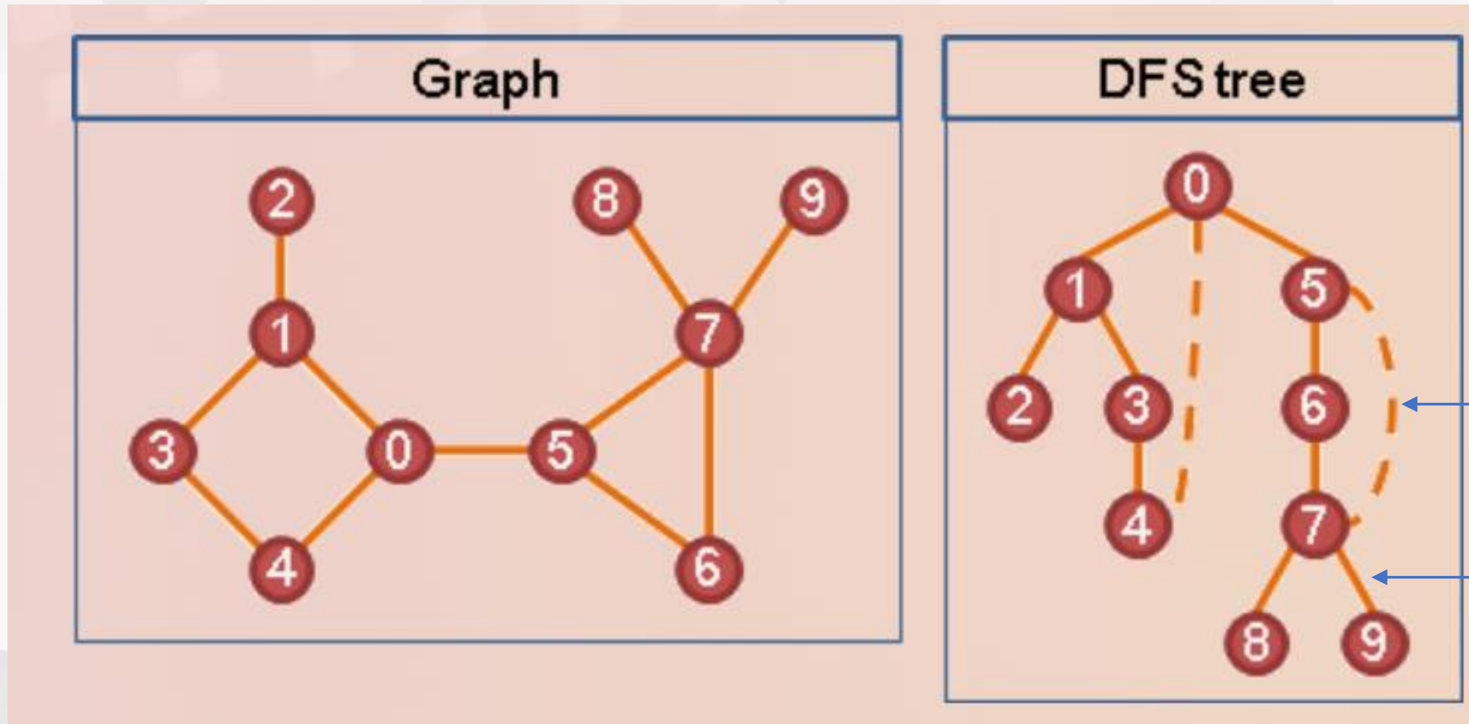
- 無向圖：
  - 連通 Connected：兩點之間邊邊相連
- 有向圖：
  - 強連通 Strongly Connected：兩點之間雙向皆有路可通

使用 Tree 的角度觀察 Graph

- 一張圖裡面可以有數個「連通塊」(Connected Component)
- 連通塊可以使用 DFS、BFS 來「遍歷」
- 連通塊被「遍歷」，形成一棵樹 (遍歷是不重複拜訪點 → 無環連通圖 → 樹)
- 原本圖上的邊就分成
  - 構成樹的邊(tree edges)
  - 未使用的邊(other edges)

無向圖 使用「DFS」遍歷





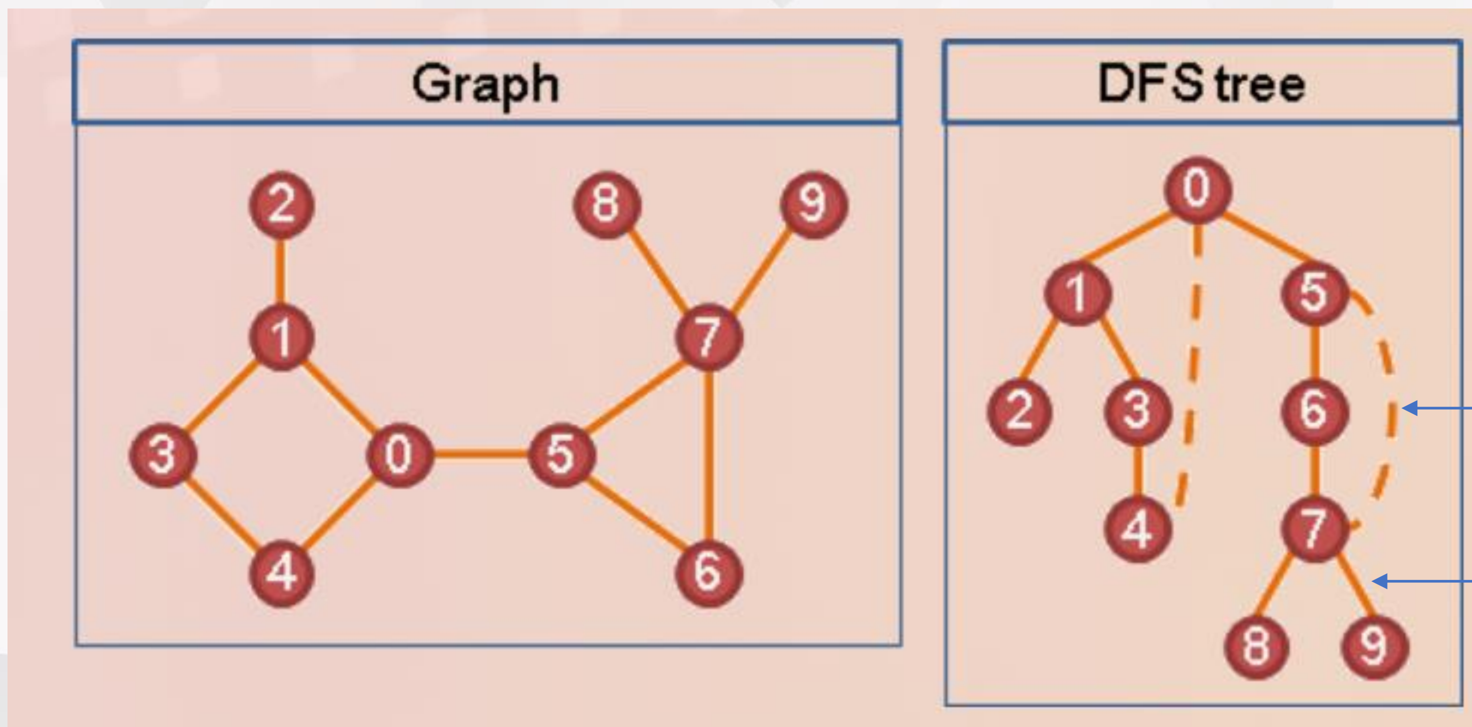
請問使用 DFS 時 other edge 必定只會連到直系祖先？

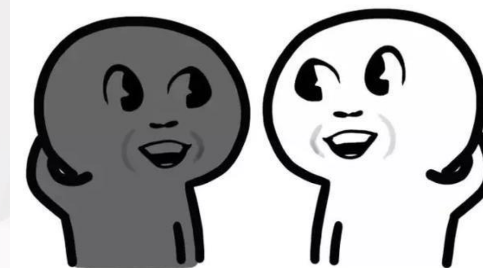
other edge

tree edge

圖片來源：演算法筆記「[Articulation Vertex Bridge]」

無向圖 使用「DFS」遍歷





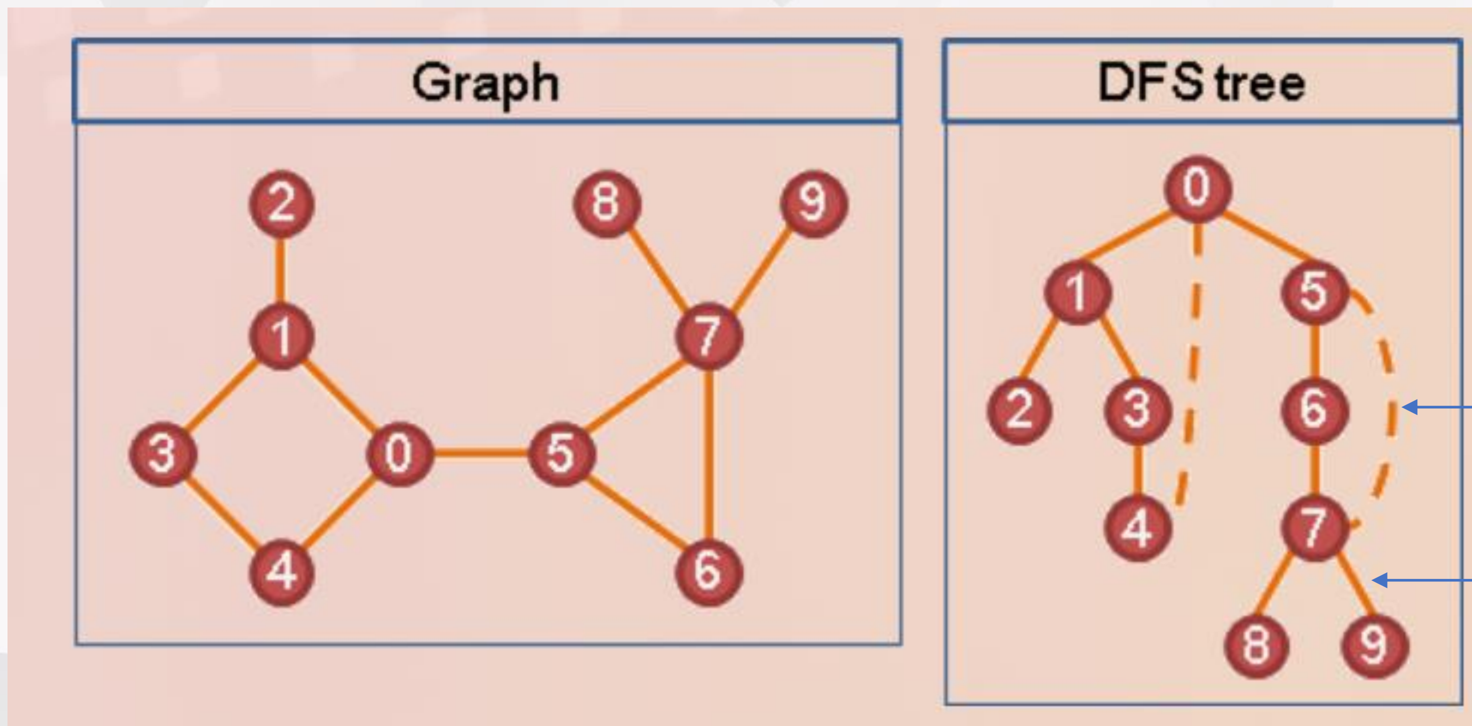
other edge 必定只會連到直系祖先！
因為 DFS 的特性，反證法

重新命名 → back edge

tree edge

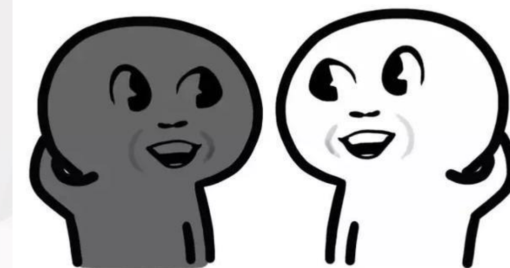
圖片來源：演算法筆記「Articulation Vertex Bridge」

無向圖 使用「DFS」遍歷



back edge

tree edge



如果有不是連接到直系祖先的 edge ，那為什麼 DFS 的時候，這個邊不是 tree edge？

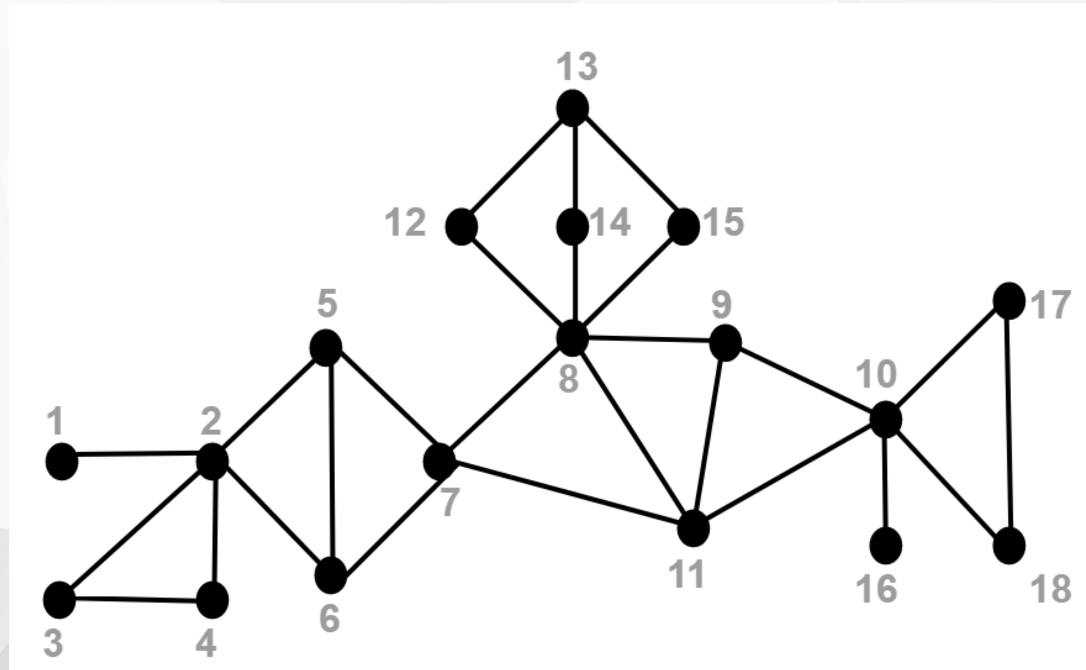
圖片來源：演算法筆記「Articulation Vertex Bridge」

Articulation Point (AP)

- 對於一個連通圖(圖中所有點在同個連通塊中)
- 如果此點被移除後，會導致連通圖不再連通。
- 稱此點為 Articulation Point

觀察 Articulation Point (AP)

- 考慮一個連通圖。

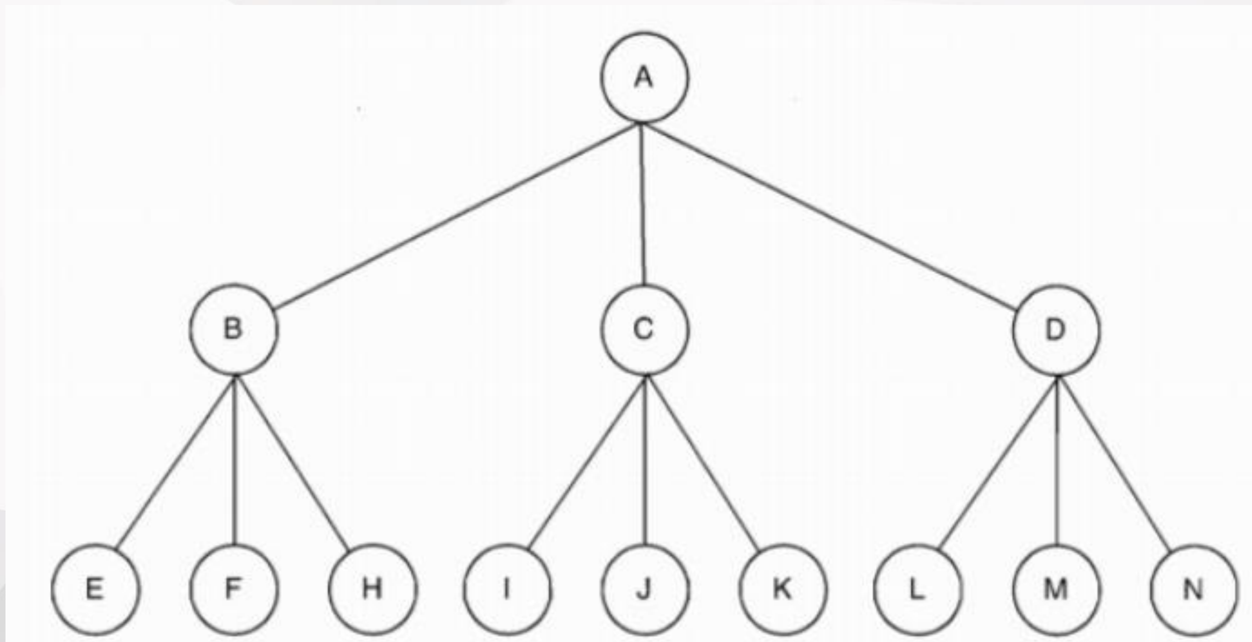


如果移除 point7 ，還是不是連通圖？否
如果移除 point11，還是不是連通圖？是
如果移除 point16，還是不是連通圖？是

圖片來源：Wikipedia「Biconnected component」

觀察 Articulation Point (AP)

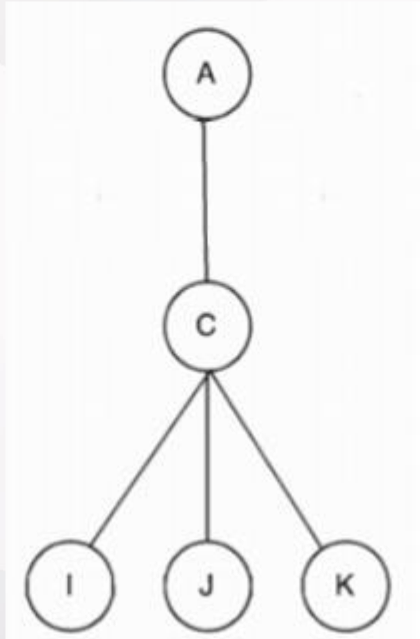
- 考慮一個 tree(無環連通圖)



如果移除 pointA，還是不是連通圖？否
如果移除 pointB，還是不是連通圖？否
如果移除 pointE，還是不是連通圖？是

觀察 Articulation Point (AP)

- 考慮一個 tree (無環連通圖)



如果移除 pointA，還是不是連通圖？是

Tree 上的 AP

- 對於任一 Node：
  - num(parent) + num(child) < 2 → 不是 AP (斷掉只會斷自己)
    - Ex: 葉 → (1+0) → 不是 AP
  - num(parent) + num(child) ≧ 2 → 是 AP (斷掉會斷別人)
    - Ex: 莖 → (1+x) → 是 AP

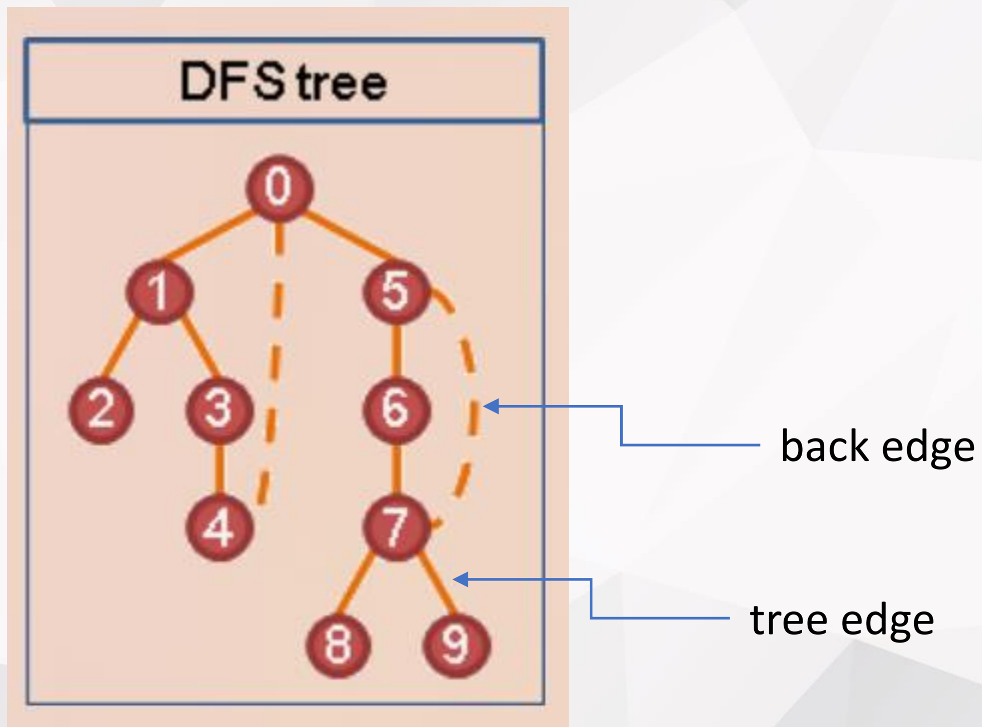
從 tree 到 graph

- 對於一張圖加上邊
  - 有機會讓「非AP」變成「AP」嗎？ 否
  - 有機會讓「AP」變成「非AP」嗎？ 是

使用 DFS tree 的角度找 AP

- 把「圖」經由遍歷得到「帶有 back edge 的 tree」
- 「back edge」有機會讓「AP」變成「非AP」

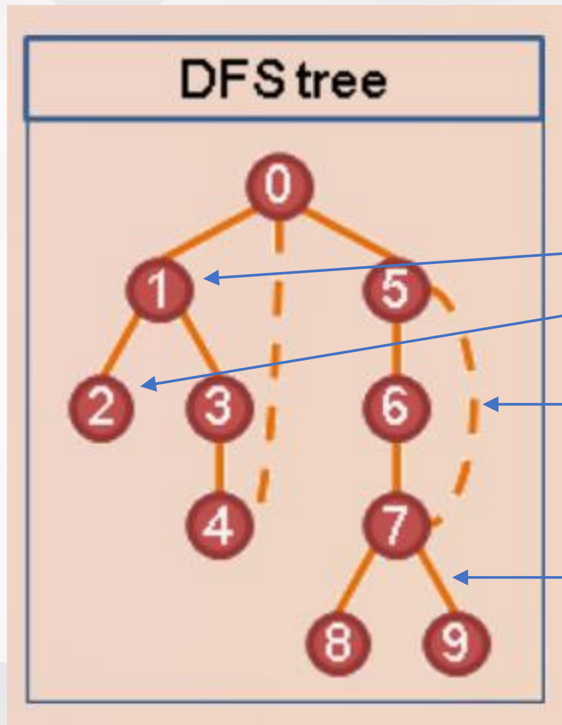
觀察 Back edge 所帶來的影響



如果移除 point0，還是不是連通圖？否
如果移除 point1，還是不是連通圖？否
如果移除 point3，還是不是連通圖？是

圖片來源：演算法筆記「Articulation Vertex Bridge」

觀察 Back edge 所帶來的影響



back edge

tree edge

「此 node 的任意孩子」
→ 無法藉由 back edge 走到更高的位置
→ 那 node 被拔掉會造成連通圖斷掉
→ 此 node 是 AP

這裡的「高」，是取決於 DFS tree 的深度。

圖片來源：演算法筆記「Articulation Vertex Bridge」

題目練習 UVa 315 Network

- 在一個網路建案中，想要請你找出幾個關鍵的點，一旦死掉會導致網路不連通。

解答

討論 low 值 的更新方式

```
void dfs(int p, int u, int d) {//u -> now, p -> parent, d -> depth
        dep[u] = low[u] = d;
        int num = (d == 0)? 0: 1; //num(parent) + num(child) in dfs tree
        bool back_error = 0;

        for(int v: edge[u]) if(v != p){  //v -> target(child or ancestor)
                if(low[v] != -1) low[u] = min(low[u], dep[v]);
                else {
                        ++num;
                        dfs(u, v, d+1);
                        low[u] = min(low[u], low[v]);
                        if(low[v] >= d) back_error = 1;
                }
        }
        if(num >= 2 && back_error == 1) ap++;
}
```

每次的「循邊」都需要維護 low 值

那為什麼這邊不能用 low[v] 取代 dep[v]？

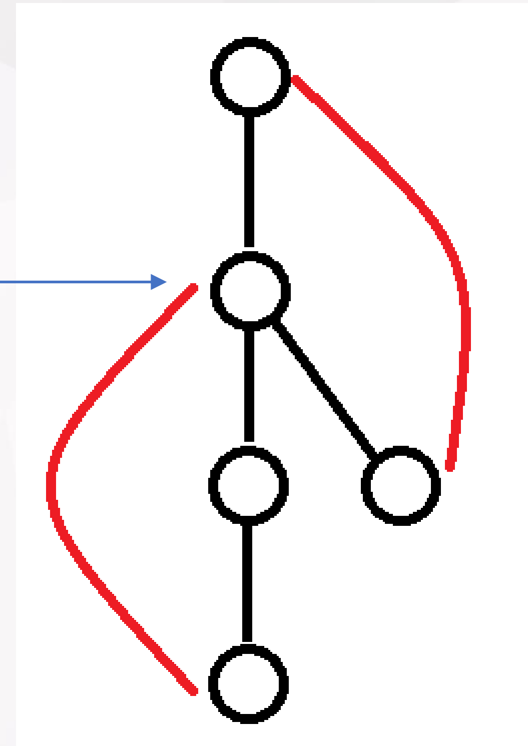
備註：
這是上頁投影片中 uva 315 的解答程式碼。
dep(n) 代表節點 n 的 depth
low(n) 代表節點 n 透過 back edge 能連向最高節點的 depth 值

討論 low 值 的更新方式

考慮以下例子：
請問這個點是不是 AP ？ 是

其正下方的子孫所能觸及的
最高點到達不了 root。

所以 low 不可以上去之後下
來再藉由其他 back edge 上去！



Strongly Connected Component

有向圖的強連通元件

Competitive Programming

Made by 培訓團隊群

CCNS.

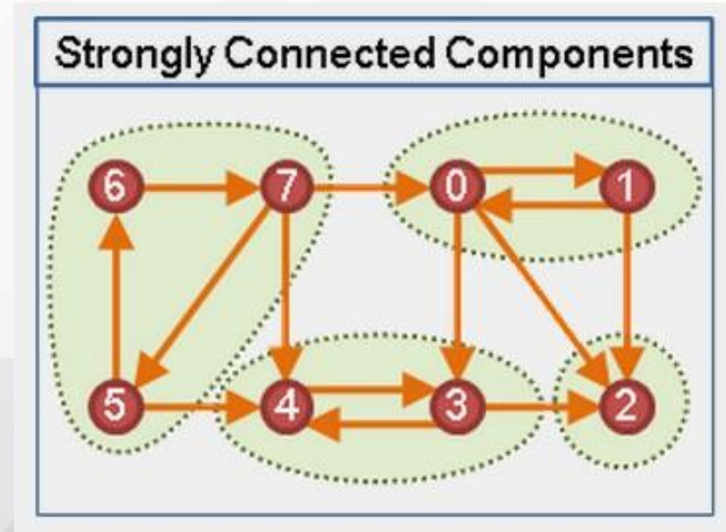
比較有向圖與無向圖「兩點間的關係」

- 無向圖：
  - 連通 Connected：兩點之間邊邊相連
- 有向圖：
  - 強連通 Strongly Connected：兩點之間雙向皆有路可通

Strongly Connected Component

- 考慮一個有向圖中的點集合構成的「子圖」，且集合中任兩點之間須滿足強連通關係，則稱這個極大點集合為一個強連通元件。

- Ex: {5,6,7}、{2} 兩組都是 SCC

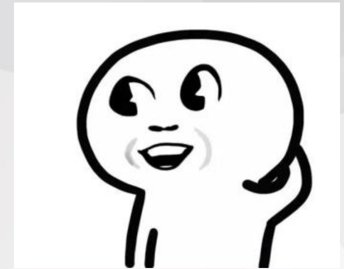


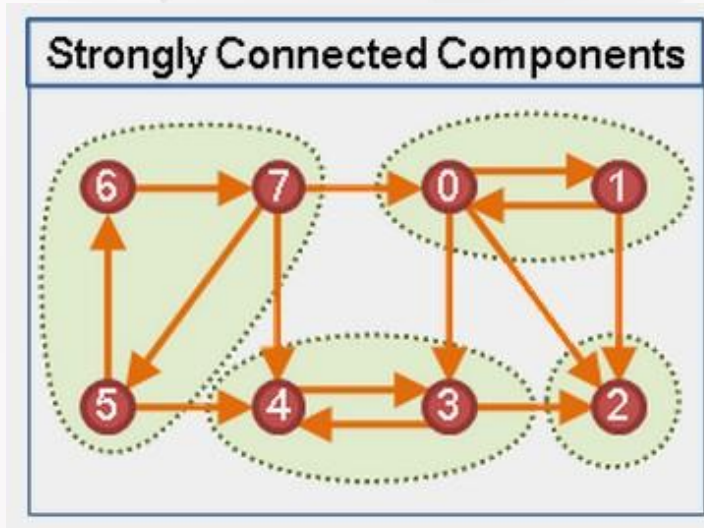
圖片來源：演算法筆記「Articulation Vertex Bridge」

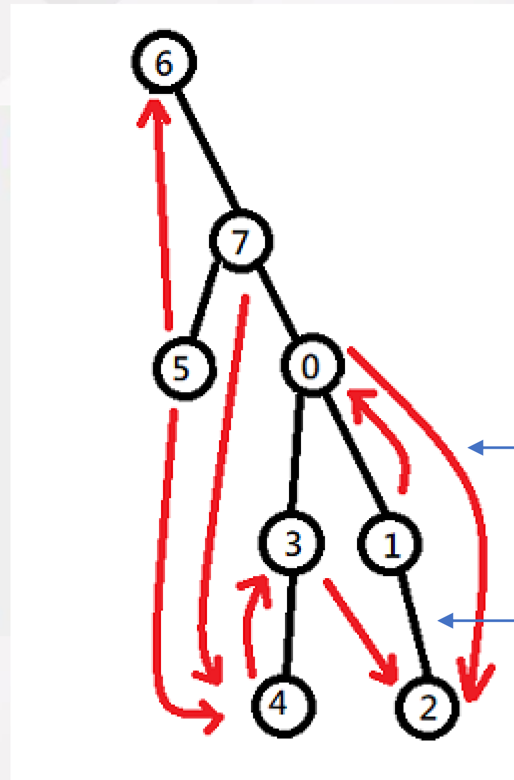
使用 Tree 的角度觀察 Graph

- 一張圖裡面可以有數個「連通塊」(Connected Component)
- 連通塊可以使用 DFS、BFS 來「遍歷」
- 連通塊被「遍歷」，形成一棵樹 (遍歷是不重複拜訪點 → 無環連通圖 → 樹)
- 原本圖上的邊就分成
  - 構成樹的邊(tree edges)
  - 未使用的邊(other edges)

有向圖 使用「DFS」遍歷







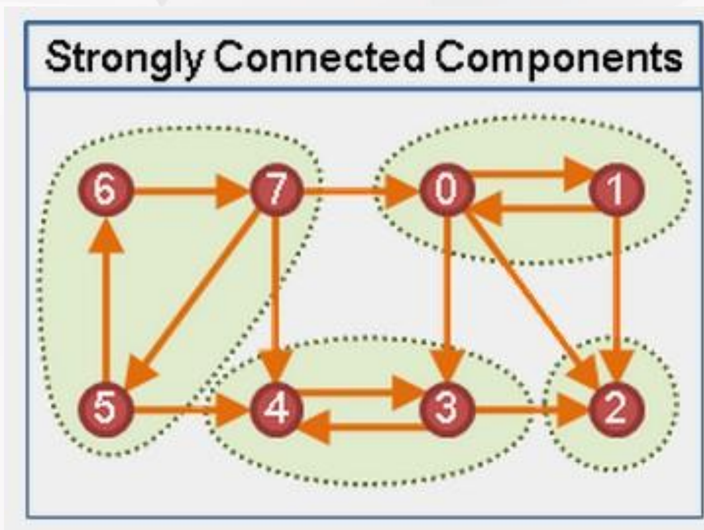
other edge

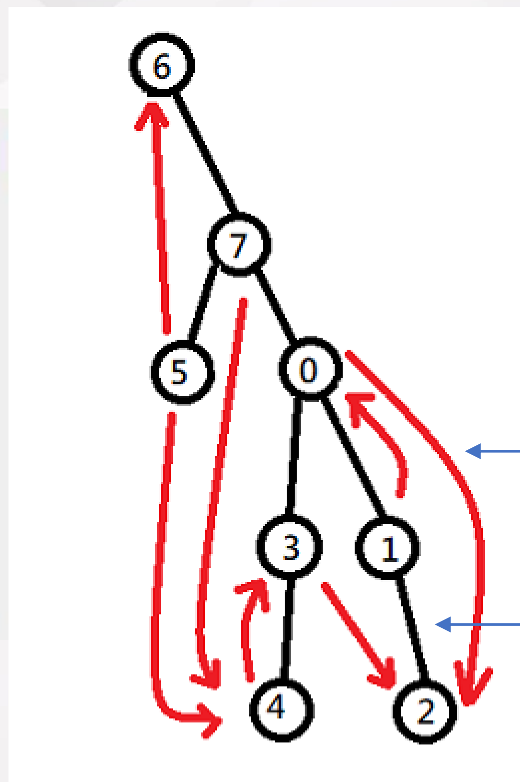
tree edge

請問使用 DFS 時 other edge
必定只會連到直系祖先？

圖片來源：演算法筆記「Articulation Vertex Bridge」

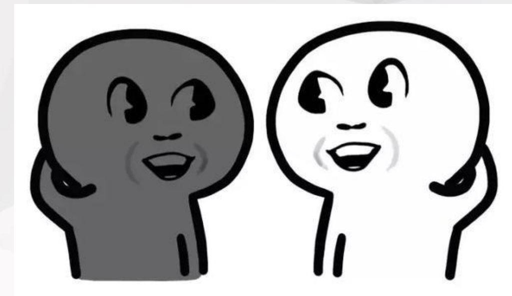
有向圖 使用「DFS」遍歷





other edge

tree edge

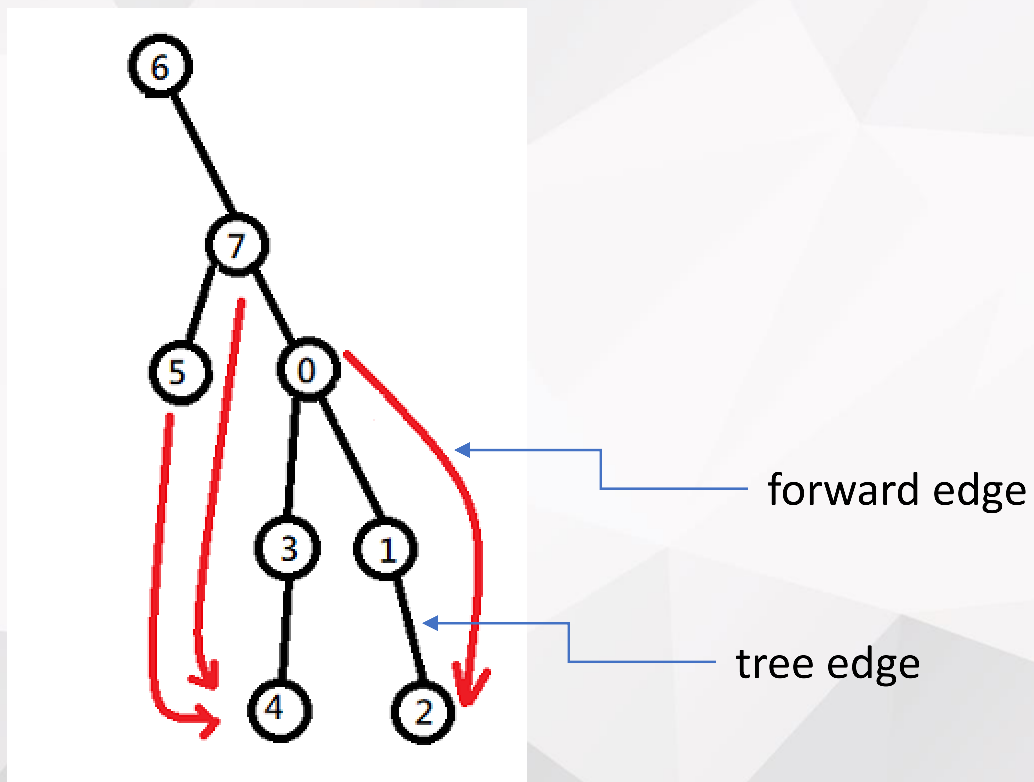


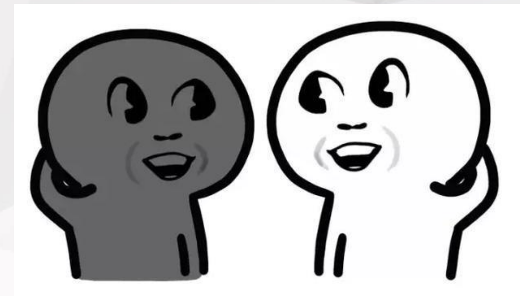
other edge 有可能:

Back edge
Forward edge
Cross edge

圖片來源：演算法筆記「Articulation Vertex Bridge」

有向圖 使用「DFS」遍歷



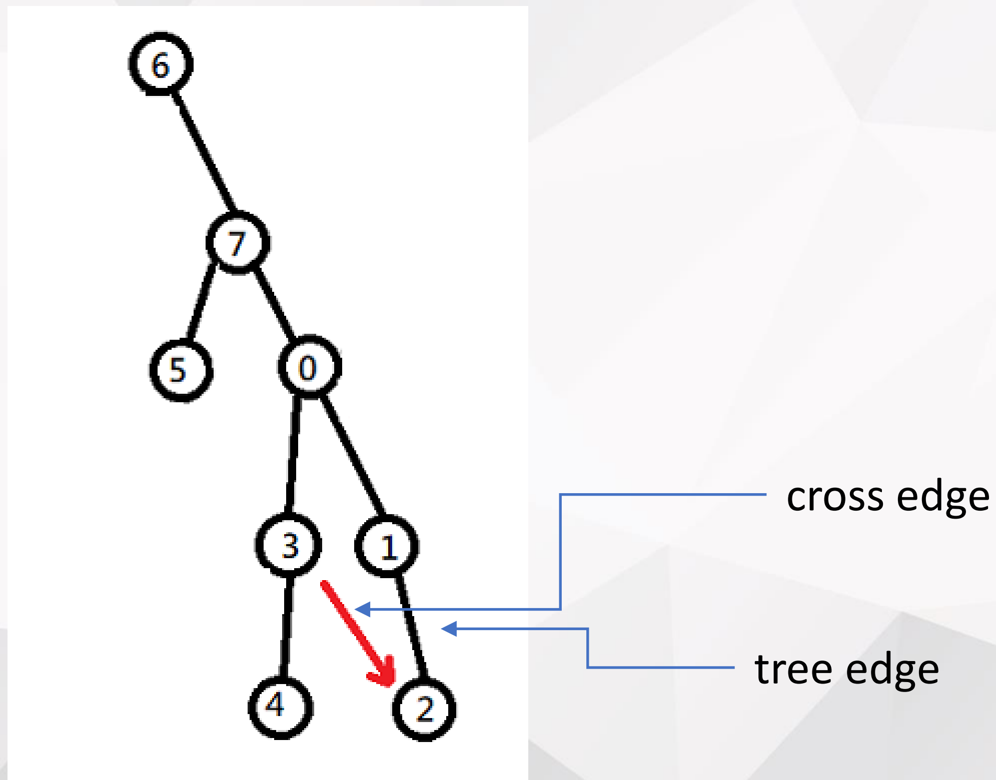


other edge 有可能:

Back edge
Forward edge
Cross edge

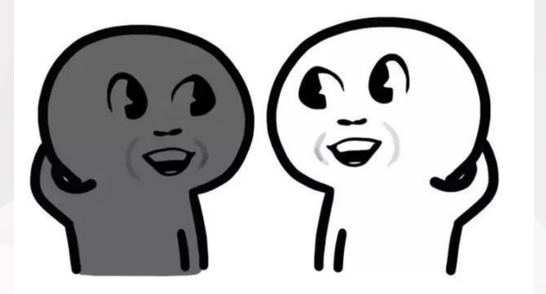
圖片來源：演算法筆記「Articulation Vertex Bridge」

有向圖 使用「DFS」遍歷



cross edge

tree edge

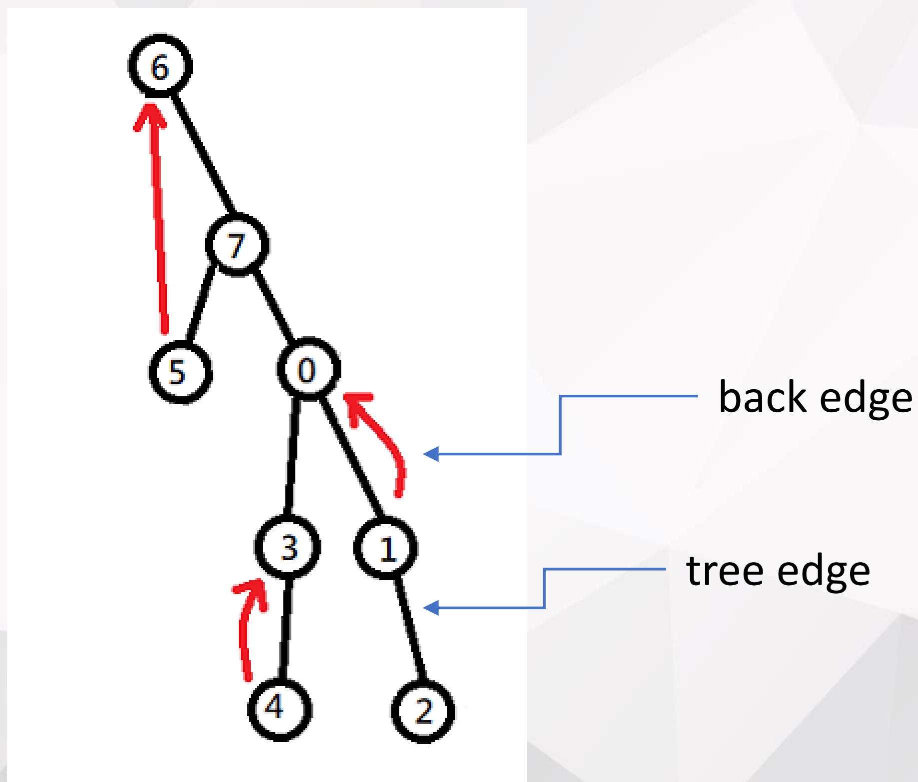


other edge 有可能:

Back edge
Forward edge
Cross edge

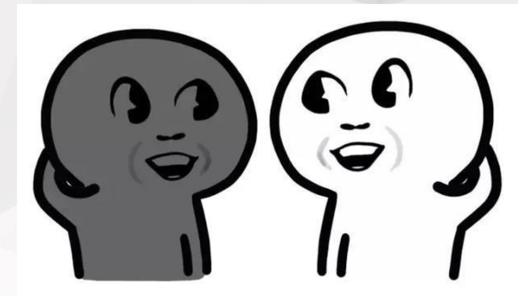
圖片來源：演算法筆記「Articulation Vertex Bridge」

有向圖 使用「DFS」遍歷



back edge

tree edge



other edge 有可能:

Back edge
Forward edge
Cross edge

圖片來源：演算法筆記「Articulation Vertex Bridge」

Tarjan Algorithm

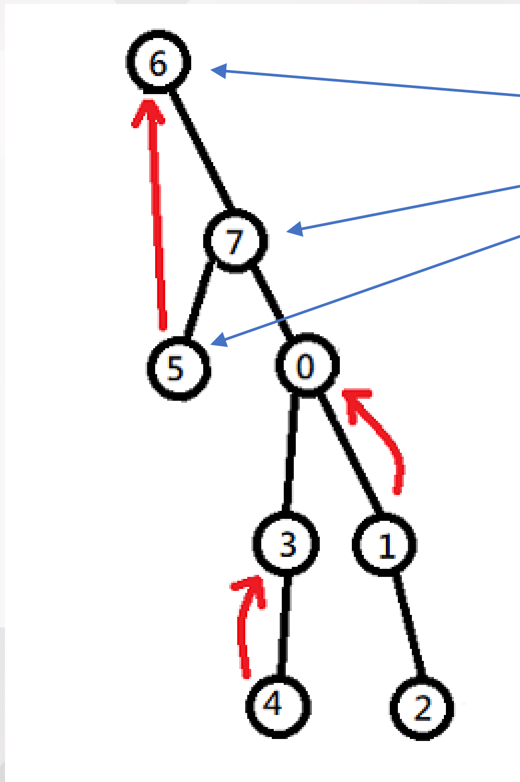
- 形成 SCC 的條件為
- SCC 內任兩點 a → b 且 b → a → 形成有向環

- 一個SCC是由一個或多個環組成的

時間不多直接說結論

- Tree 無環
- Back edge 才可能把 dfs tree 連接成有向環。
- 剩下兩種 edge 不用考慮。

SCC 與 back edge



「同一個 SCC 上的點」
→ 藉由 back edge 走到「同個」高的位置

這裡的「高」，是取決於 DFS tree 的深度。

Ex: Point6, Point7, Point5 的 low值 都會等於 dfn(Point 6)

備註：因為這裡需要確定最高的是哪個點，所以這裡使用 dfn(n) 取代 dep(n)。
dfn(n)為 dfs 的順序，同樣有深度的感覺。

題目練習 ICPC LA 4262 - Road Networks

- 找出 SCC。

解答

Made by 培訓團隊群 CCNS