C/C++ 進階班
演算法

網路流
(Flow Networks)

李耕銘

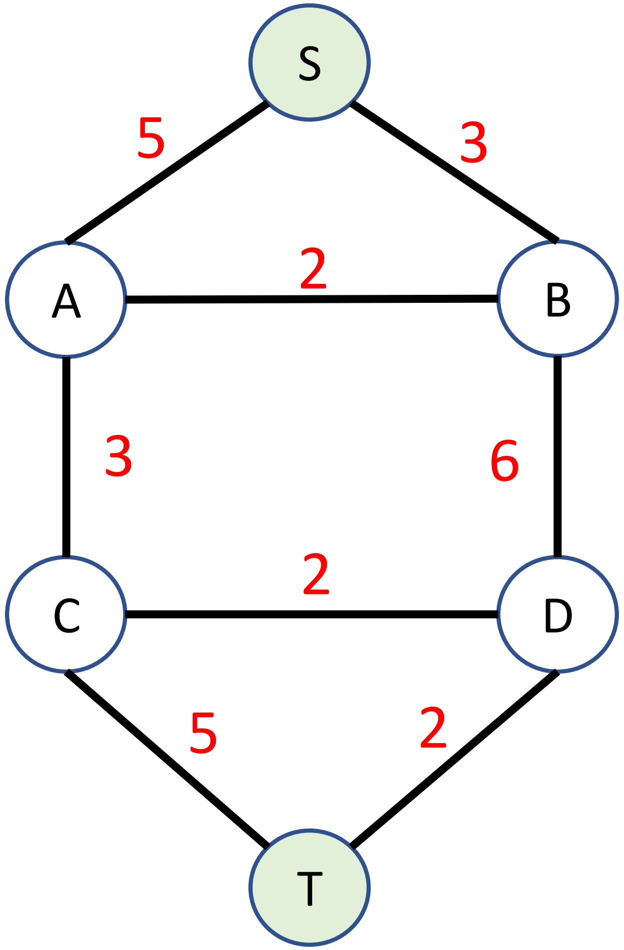
課程大綱

- 網路流簡介
- 最大流最小割定理
- Ford-Fulkerson Algorithm
- Edmonds-Karp Algorithm

網路流簡介

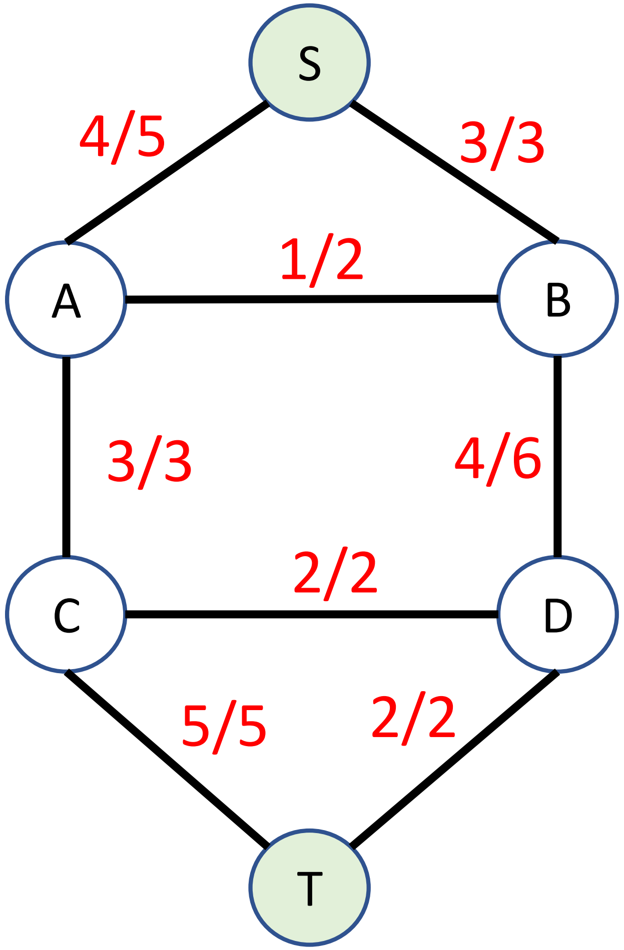
網路流簡介

- 網路流 NetWork Flow
  - 把圖想成是水管分布圖
    1. 邊想成是水管
    2. 邊上的權重想成是水管的容量上限 (正數)
    3. 點的權重是兩水管接合處的容量上限
       - ✓ 一般不考慮點的權重
  - 頂點中有兩個特殊點
    1. 源點 (Sources)
    2. 匯點 (Sinks)
  - 找出從源點至匯點能夠支撐的最大流量



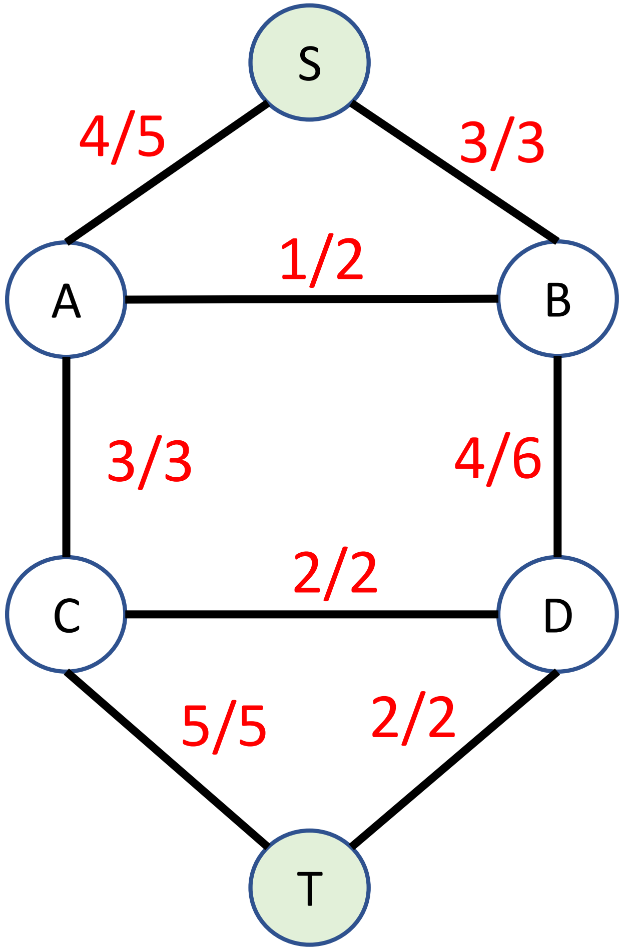
網路流簡介

- 範例，S→T 的最大流量：
  - S→A→C→T : 3
  - S→A→B→D→C→T : 1
  - S→B→D→T : 2
  - S→B→D→C→T : 1
  - 總共 : 3 + 1 + 2 + 1 = 7



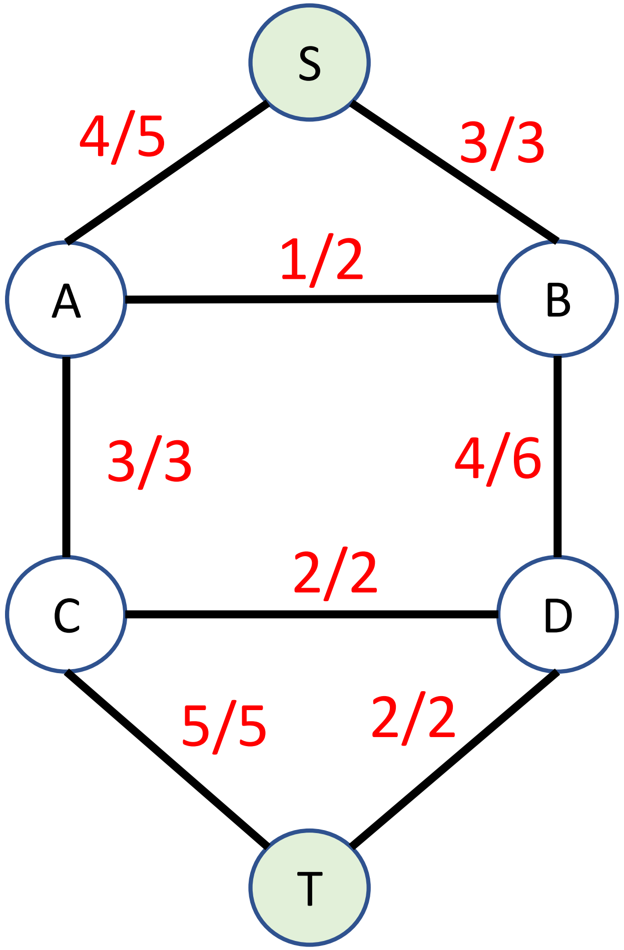
網路流簡介

- 網路流問題名詞定義
  1. 網路 (Network)：圖 G = (V,A) 為有向圖，稱網路
  2. 源點與匯點 (Source and Sink)：源點 S 為網路流的起點、匯點 T 為網路流的終點，其餘點為中間點
  3. 流量 (Flow)：每條邊/弧上的數值表目前經過該邊/弧的流量 $\boldsymbol{F(u,v)}$，所有流量的集合為網路的一個流
  4. 容量 (Capacity)：每條邊/弧上的數值表 $\boldsymbol{C(u,v)}$ 該邊/弧的流量上限
  5. 剩餘容量 (Residual Capacity)：每條邊/弧上的容量減去流量，稱為該邊/弧的剩餘容量
     - $\mathbf{Cf}(\boldsymbol{u},\boldsymbol{v}) = \boldsymbol{C}(\boldsymbol{u},\boldsymbol{v}) - \boldsymbol{F}(\boldsymbol{u},\boldsymbol{v})$
  6. 剩餘網路 (Residual Network)：剩餘容量的集合
  7. 網路流量 (Flow of Network)：由源點出發至匯點匯集的總流量，若其又為該網路所能達到的最大流量，稱為最大流 (Maximum Flow)。
     - $|\mathbf{f}| = \sum f(s,v) = \sum f(v,t)$



網路流簡介

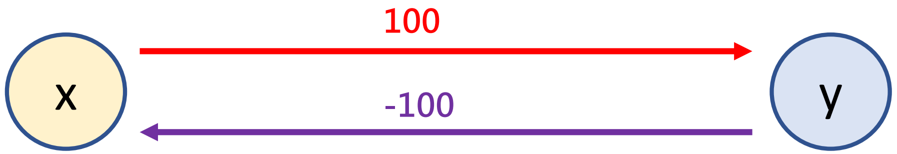
- 網路流上的弧 (Arcs of Network Flow)
  1. 不飽和弧：若一條邊/弧上的流量小於其容量，稱為不飽和弧
     - ✓ $F(u,v) < C(u,v)$
  2. 飽和弧：若一條邊/弧上的流量恰好等同於其容量，稱為飽和弧
     - ✓ $F(u,v) = C(u,v)$
  3. 零流弧：若一條邊/弧上的流量為零，稱為零流弧
     - ✓ $F(u,v) = 0$
  4. 非零流弧：若一條邊/弧上的流量不為零
     - ✓ $F(u,v) \neq 0$
  5. 前向弧與後向弧：若 P 為源點到匯點的路徑，並定義該路徑方向便是由源點到匯點，若該路徑中邊/弧的流量方向與路徑方向相同，稱為前向弧，若不同則為後向弧



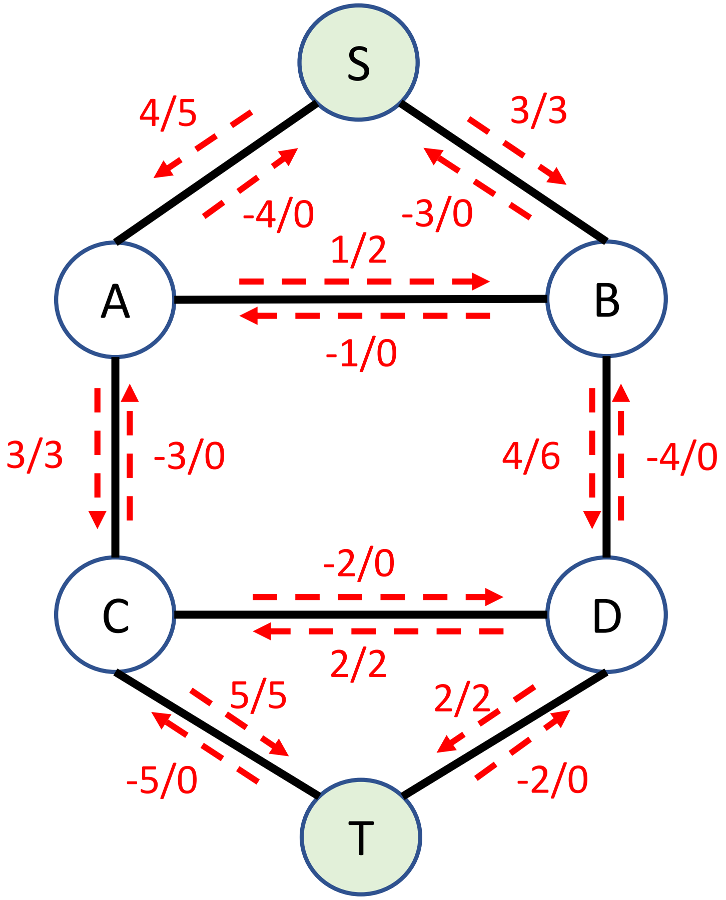
網路流問題原則

網路流簡介

- 網路流問題的三大限制
  1. 容量限制 (Capacity Constraints)
     - 流量不大於容量
     - $F(u, v) \leq C(u, v)$
  2. 流量守恆 (Flow Conservation)
     - 任意節點的流入量必等於流出量
     - 源點流出總流量等於流入匯點的總流量
     - $\sum F(u, i) = \sum F(j, u)$
  3. 斜對稱性 (Skew Symmetry)
     - u 到 v 的淨流量加上 v 到 u 的淨流量必為零
     - $F(u, v) + F(v,u) = 0$
       - ✓ x 向 y 流了 100，等同 y 向 x 流了 -100的流量



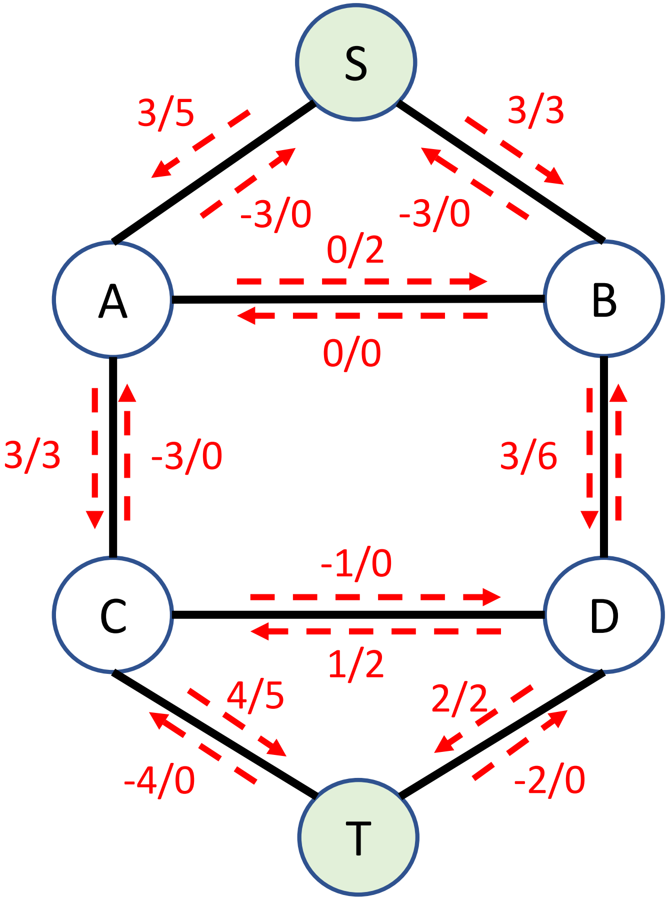
- 可行流 (Positive Flow)：若一個流符合以上三限制，稱為可行流

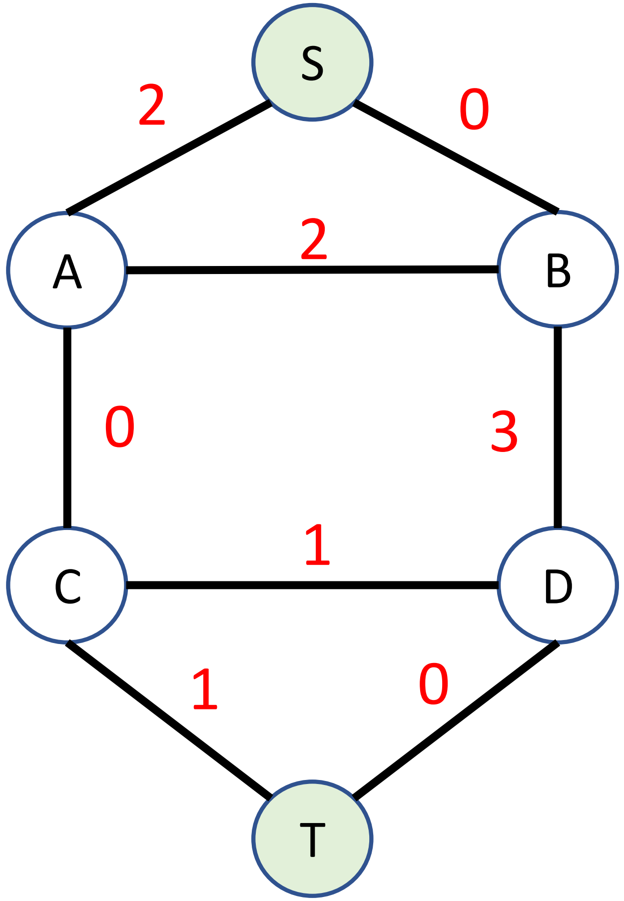


網路流簡介

剩餘容量 (Residual Capacity)

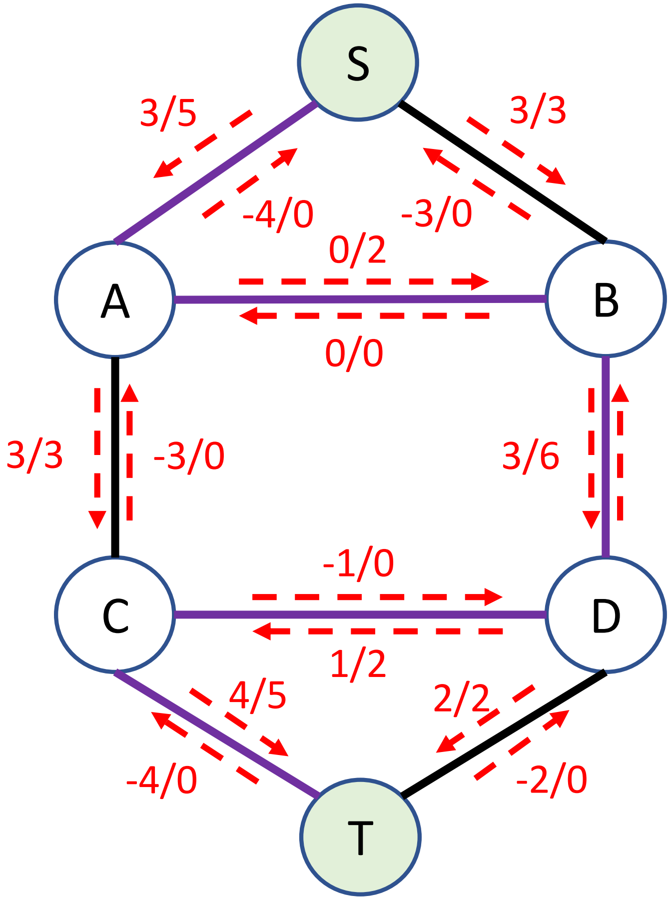
- ✓ 每條邊/弧上的容量減去流量，$\mathbf{Cf}(\boldsymbol{u},\boldsymbol{v}) = \boldsymbol{C}(\boldsymbol{u},\boldsymbol{v}) - \boldsymbol{F}(\boldsymbol{u},\boldsymbol{v})$，稱為該邊/弧的**剩餘容量**
- ✓ 重新配置水流時可以直接在**殘餘網路** (Residual Network) 上實作





網路流簡介

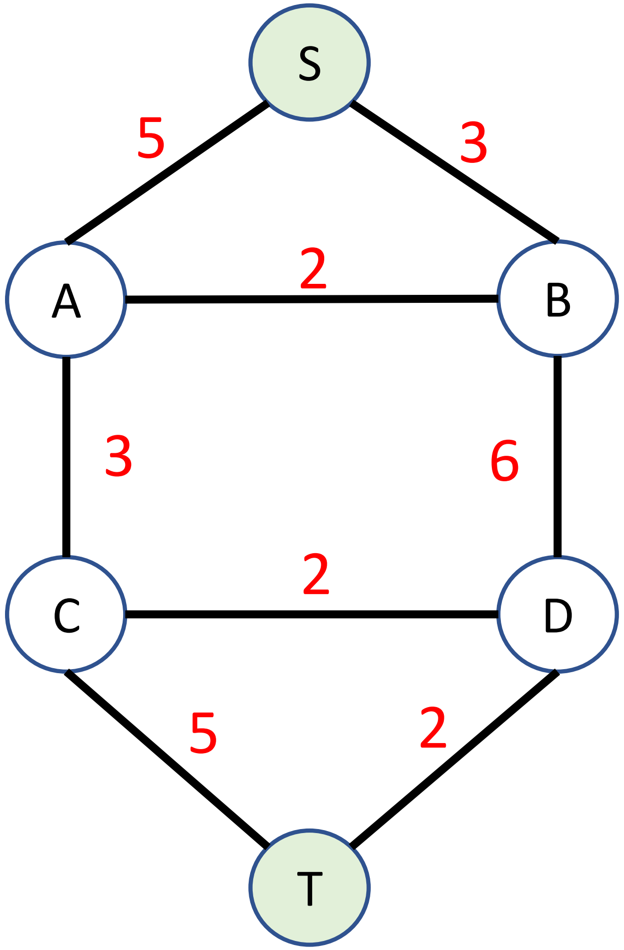
- 增廣路徑 (Augmenting Path)
  1. 若 f 是一個可行流、P 是源點到匯點的一條路徑，且 P 滿足以下條件，稱 P 為可行流 f 的一條增廣路徑：
     ✓每條前向弧為非飽和弧→可以在該弧上增加流量
  2. 增廣路徑上的 $Cf(u_k, u_{k+1})$ 大於零
     ✓ 可以在該增廣路徑上加流量
     ✓ 加上流量後整個流仍是可行流，且網路流的總流量增加
  3. Example：
     ✓ 增廣路徑：S→A→B→D→C→T
     ✓ 可增加流量 1
  4. Key Points：增廣路徑適合在殘餘網路上尋找



網路流簡介

網路最大流問題的兩大演算法

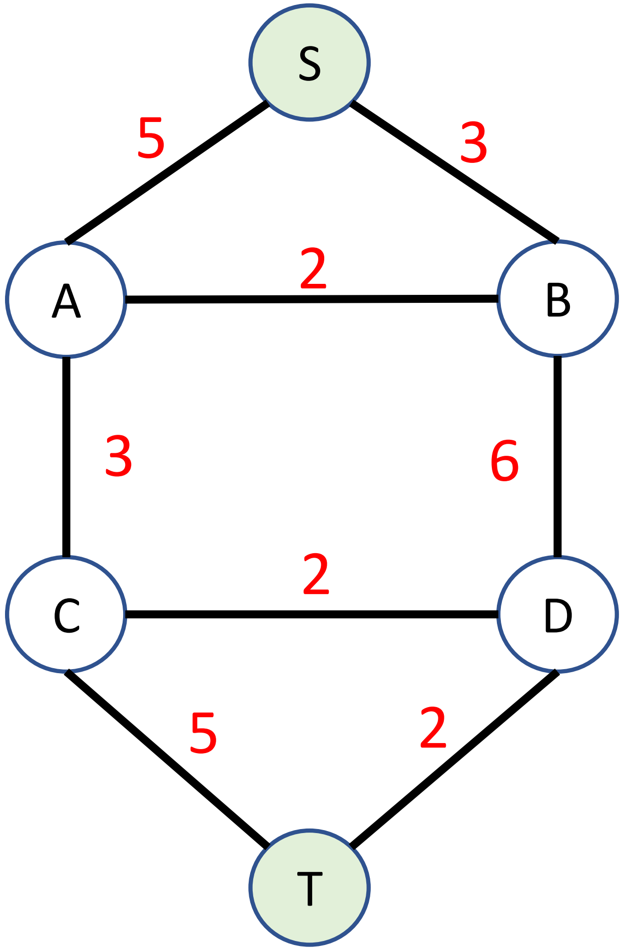
1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流
      ✓ O((V+E) f) = O(Ef)



網路流簡介

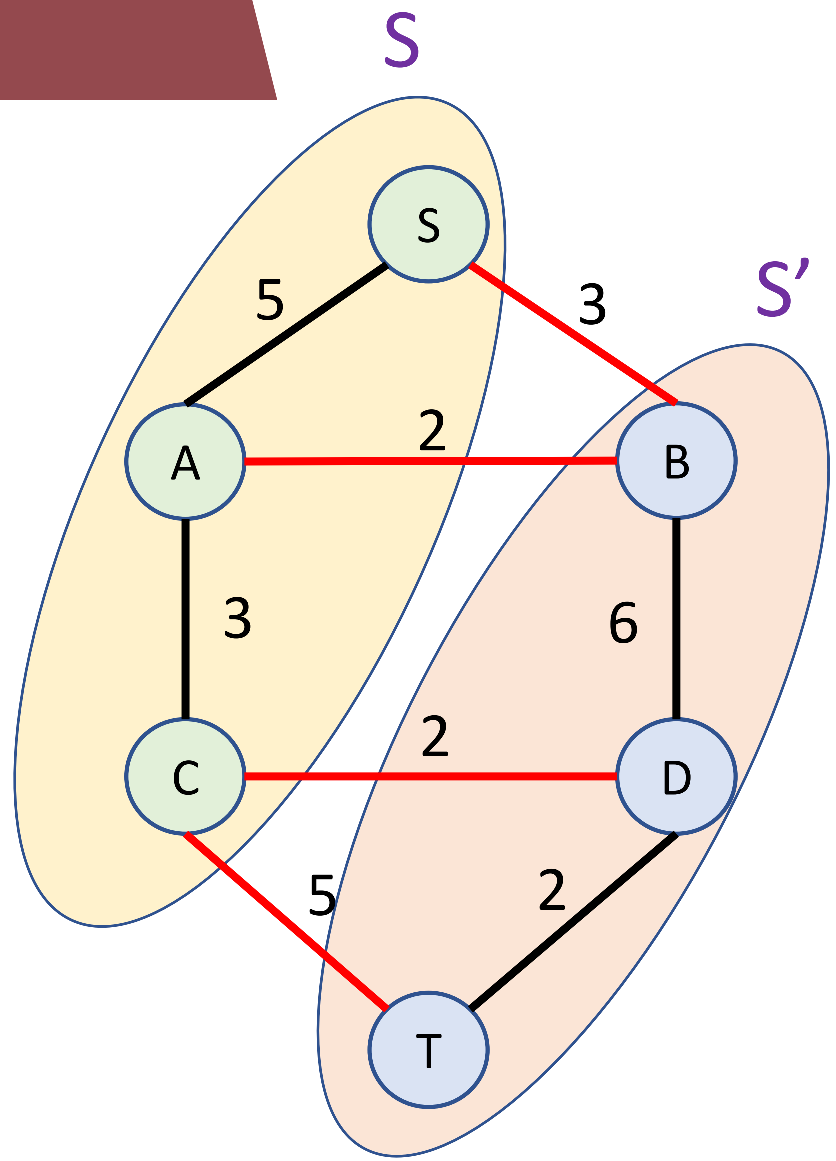
網路最大流問題的兩大演算法

2. Edmonds-Karps algorithm
   a. 與 Ford-Fulkerson method 雷同
      - ✓ 找增廣路徑時以廣度優先搜尋(BFS)尋找
      - ✓ 每次找到的增廣路徑必經過最少的邊/弧
   b. 從頂點的觀點：
      - ✓ 每找出增廣路徑並修正殘餘網路後，相當於消除一條殘餘網路中的最短路徑 (假設弧的距離相等)
      - ✓ 修正後的後向弧也不會縮短最短路徑的長度
   c. 從邊/弧的觀點：
      - ✓ 網路中最多只有 O(VE)條增廣路徑
      - ✓ BFS找增廣路徑的複雜度為O(V+E)，且 E ≥ V
      - ✓ 總複雜度為 $\boldsymbol{O}\big((\boldsymbol{V}+\boldsymbol{E})(\boldsymbol{VE})\big) = \boldsymbol{O}(\boldsymbol{VE}^2)$



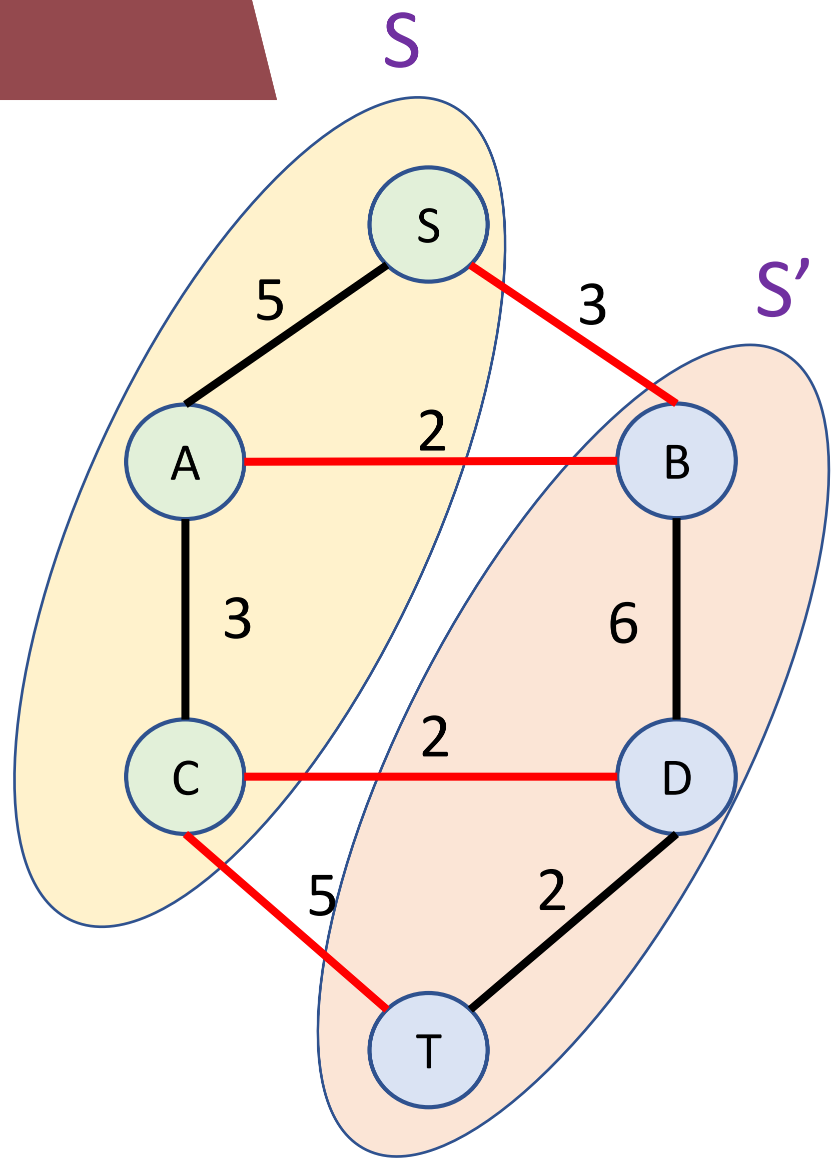
網路流簡介

- 割 Cut
  1. 在一個流量網路 G = (V,A) 中，把所有頂點 V 分成不相交的兩集合 S 與 S′
     - ✓ 源點在 S
     - ✓ 匯點在 S′
  2. 若 A′ 是 A 的最小子集，使得 G 中去除 A′ 後可以使 G 成為兩不相交的子圖 $G_1(S, A_1)$ 與 $G_2(S', A_2)$，則稱 A′ 是 S 與 S′ 的割集
  3. A′ 裡弧的容量總和則稱為割的容量
  4. 若 A′ 為 G 所能產生的割集中容量和最小的，則 A′ 稱為該圖的最小割 (Minimum Cut)
- Example：
  1. 若把所有頂點分成 S = {S,A,C}、S′ = {B,D,T}，則 A′ = $\{\overline{SB}, \overline{AB}, \overline{CD}, \overline{CT}\}$
  2. A′ 的容量為：3+2+2+5 = 12



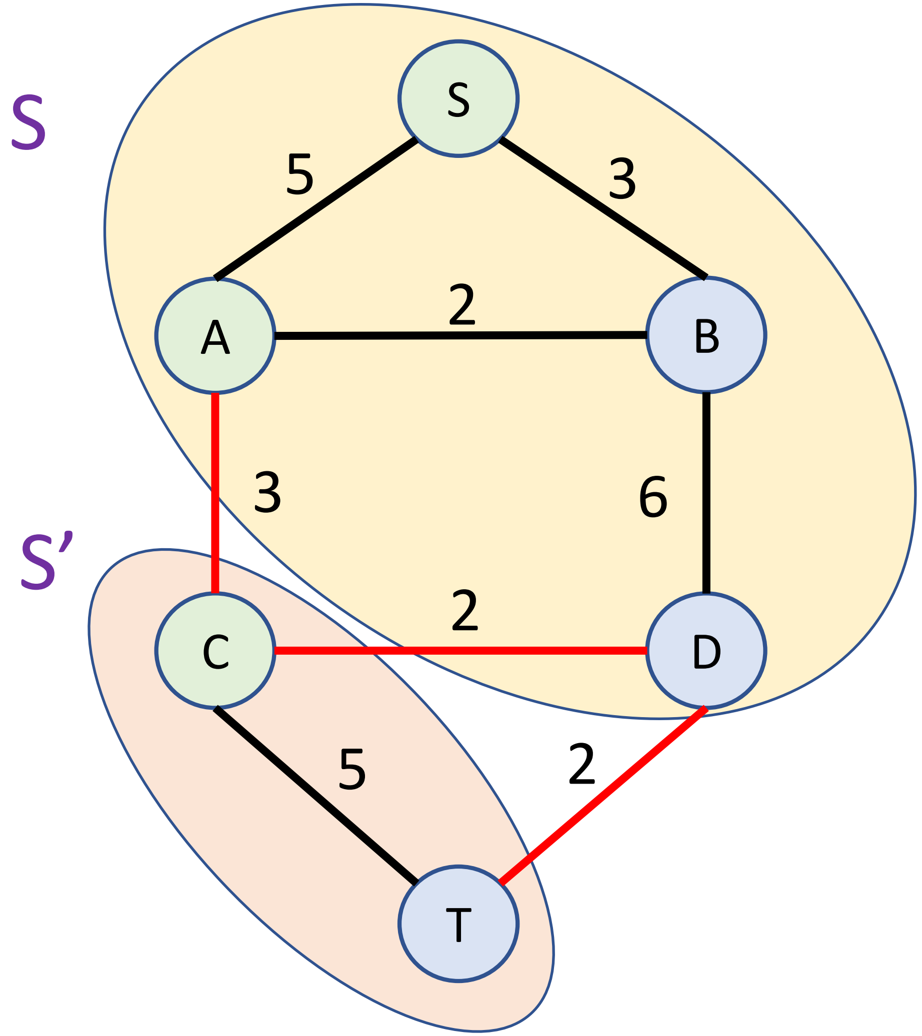
網路流簡介

- 最大流最小割定理 (Maximum Flow Minimum Cut Theorem)
  - ➢ 流量網路 G = (V,A) 中，以下三個條件等價
    1. 有一流 f 為 G 的最大流
    2. G 的殘餘網路中沒有增廣路徑
    3. 存在一割 C，割的容量為流量 f
  - ➢ 假設割集中 S, T 分屬兩不同的點集合
    1. Cf(u, v) = 0
       - ✓ 否則便可產生一條增廣路徑到 v 在所屬的集合中
    2. Cf(u, v) = C(u, v) - F (u, v)
       - ✓ C(u, v) = F (u, v)
    3. C 為最小割
  - ➢ 找尋網路流中的最大流 = 找尋網路流中的最小割
    - ✓ $|f| \leq \boldsymbol{Capcity\ of\ cut}$



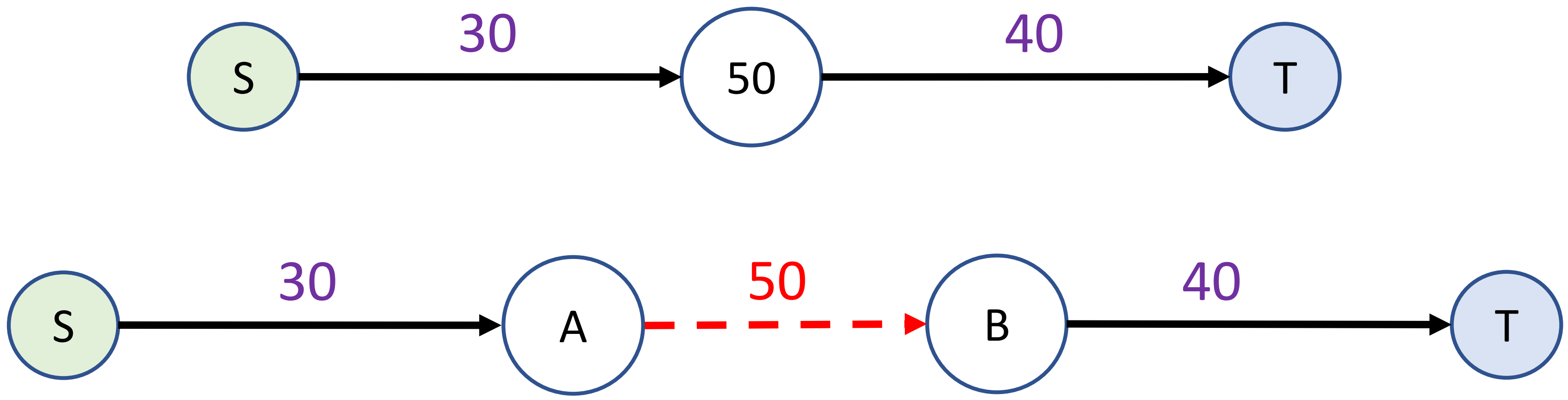
網路流簡介

- 最小割集 (Minimum Cut)
  - ➢ 找尋網路流中的最大流 = 找尋網路流中的最小割
    - ✓ 最小割的弧一定是飽和弧
    - ✓ 殘餘網路中的最小割容量為 0
  - ➢ 從源點開始沿殘餘網路的前向弧搜索
    - ✓ 直至找到每條路徑中第一條容量為 0 的弧
    - ✓ 那些弧的集合就是最小割



網路流簡介

- 網路流問題的變化：點容量 (Capacity of Vertices)
  1. 一般網路流問題只會限制邊上的容量
     - 頂點只要符合流量守恆便可
  2. 如果給定頂點的流量限制？
     - 把頂點拆成兩個點與一條邊
     - 該頂點的流量限制即為邊的流量限制
  3. 變回一般的網路流問題了！



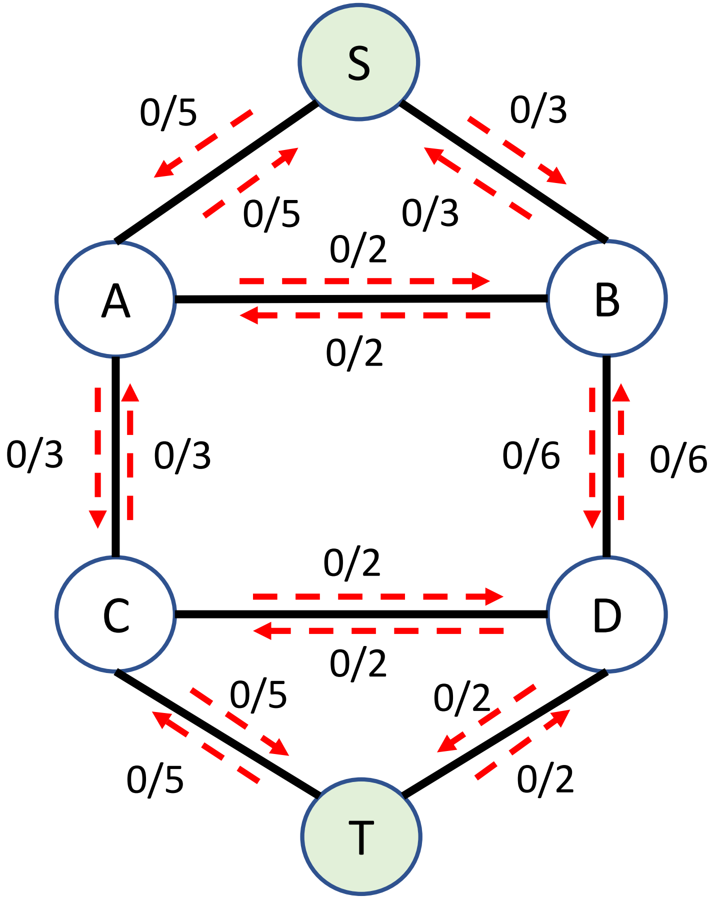
網路流簡介

- 網路流問題的變化：多個源點與匯點 (Multiple Source/Sink Vertices)
  1. 一般網路流問題只會有一個源點與匯點
  2. 如果同時給定多個源點與匯點？
     - ➢ 把所有源點與匯點各自連接成一個點
     - ➢ 新增的邊容量為無限大
  3. 變回一般的網路流問題了！

Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

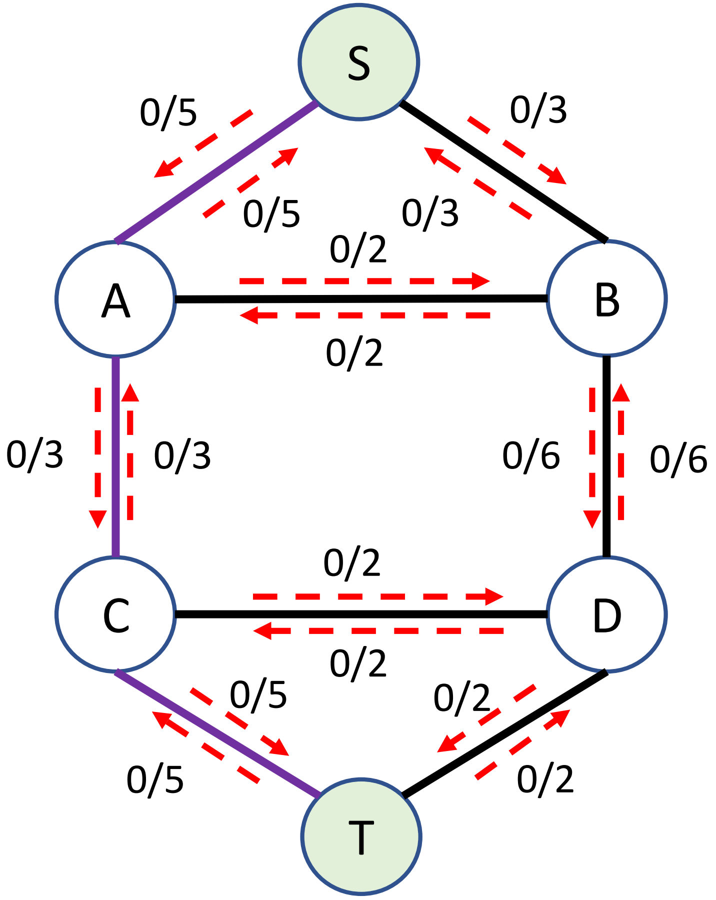
1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流



最大流是由許多小細流匯聚而成
用一條條的小細流累積出最大流

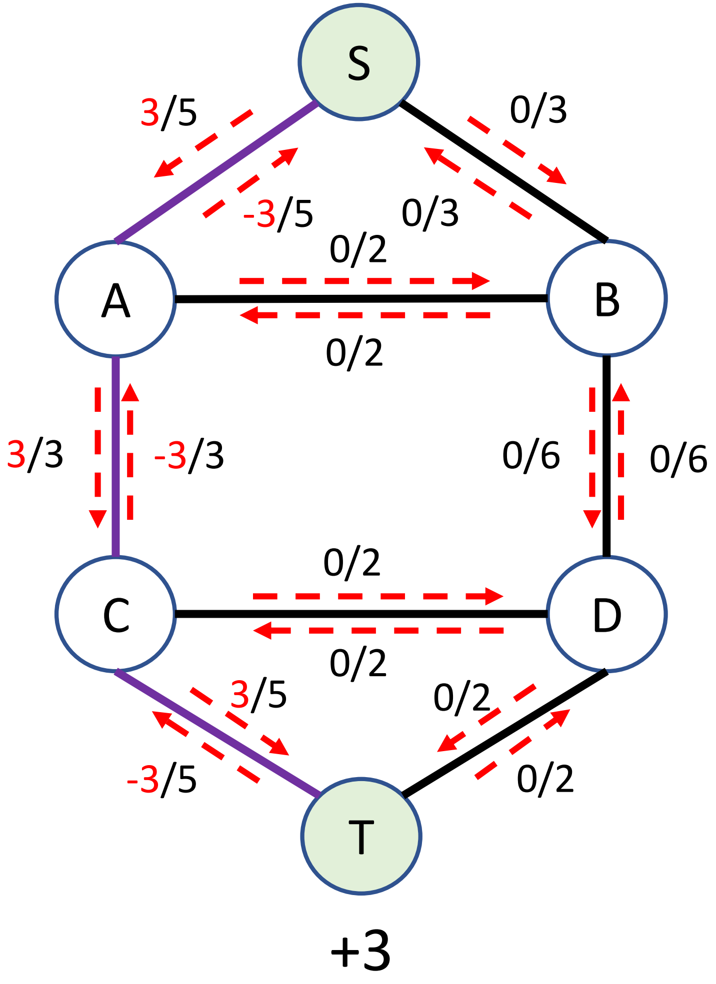
Ford-Fulkerson Algorithm

1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流



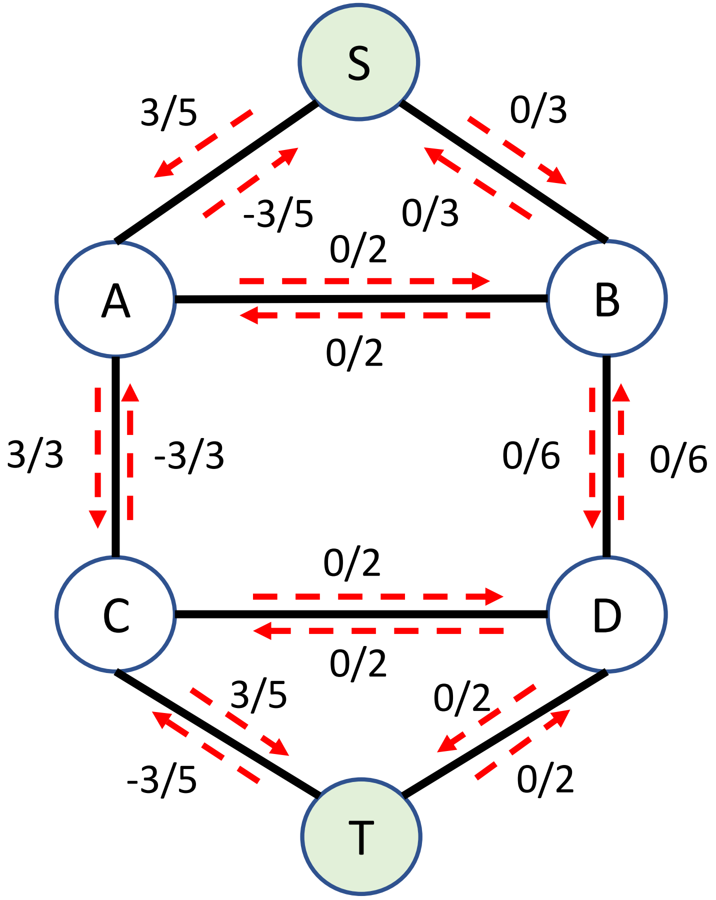
Ford-Fulkerson Algorithm

1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流



Ford-Fulkerson Algorithm

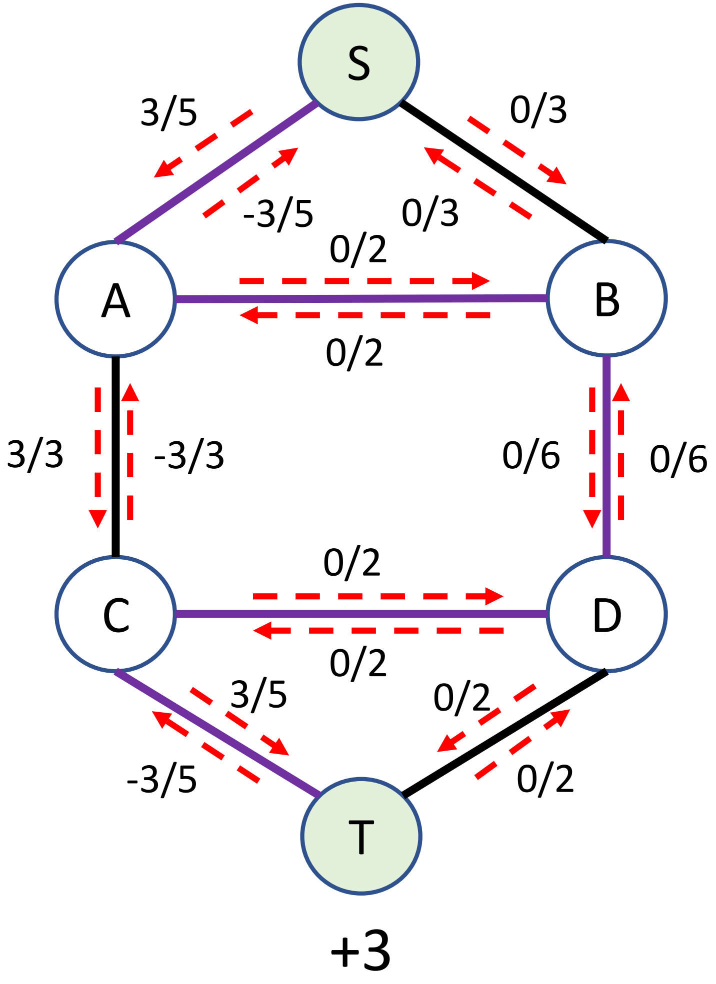
1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流



+3

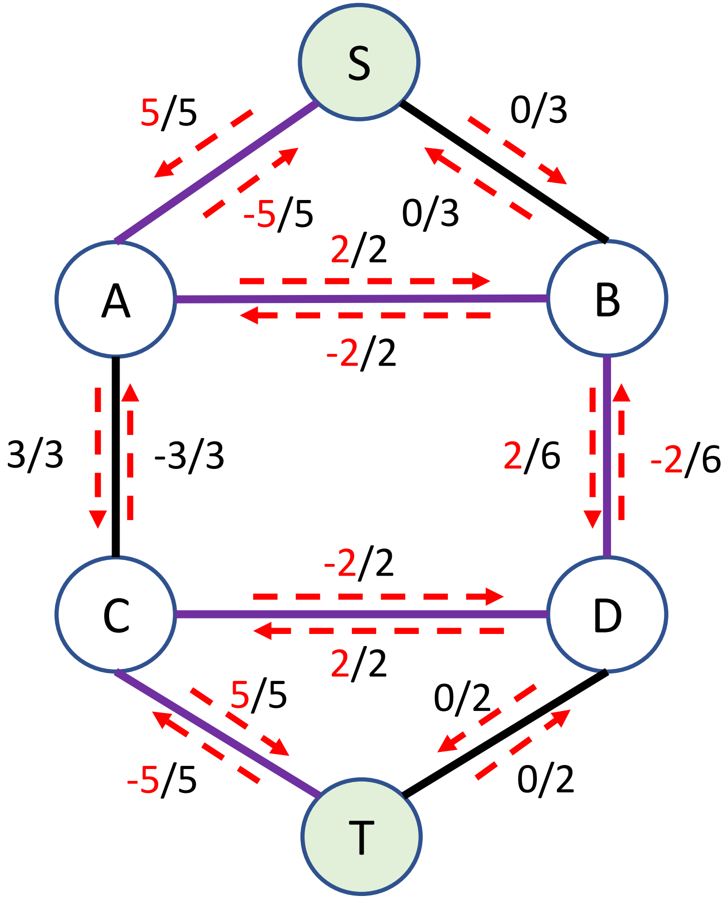
Ford-Fulkerson Algorithm

1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流



Ford-Fulkerson Algorithm

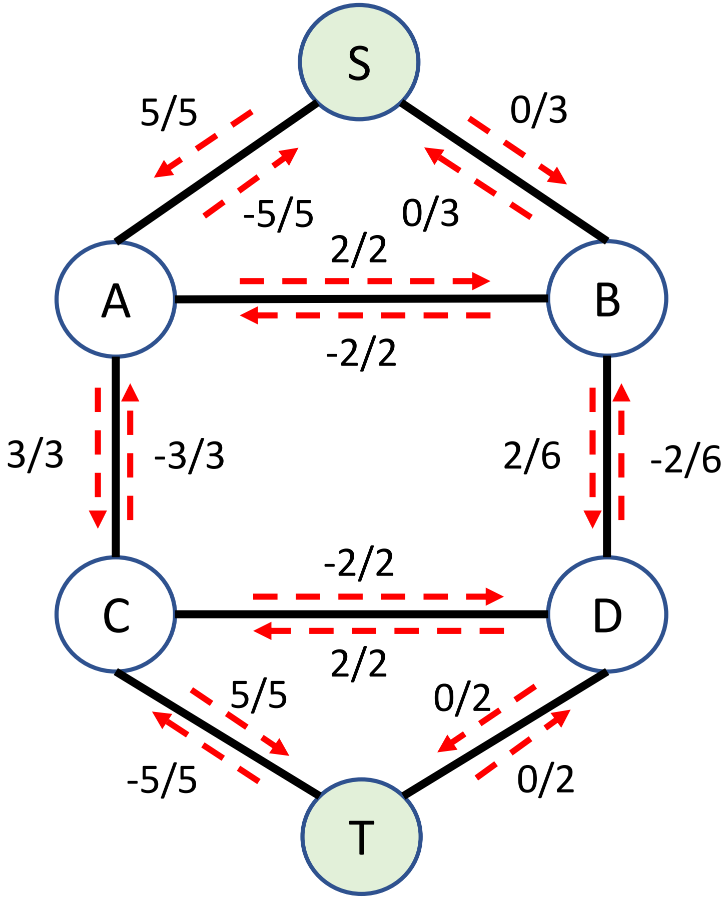
1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流



+3+2

Ford-Fulkerson Algorithm

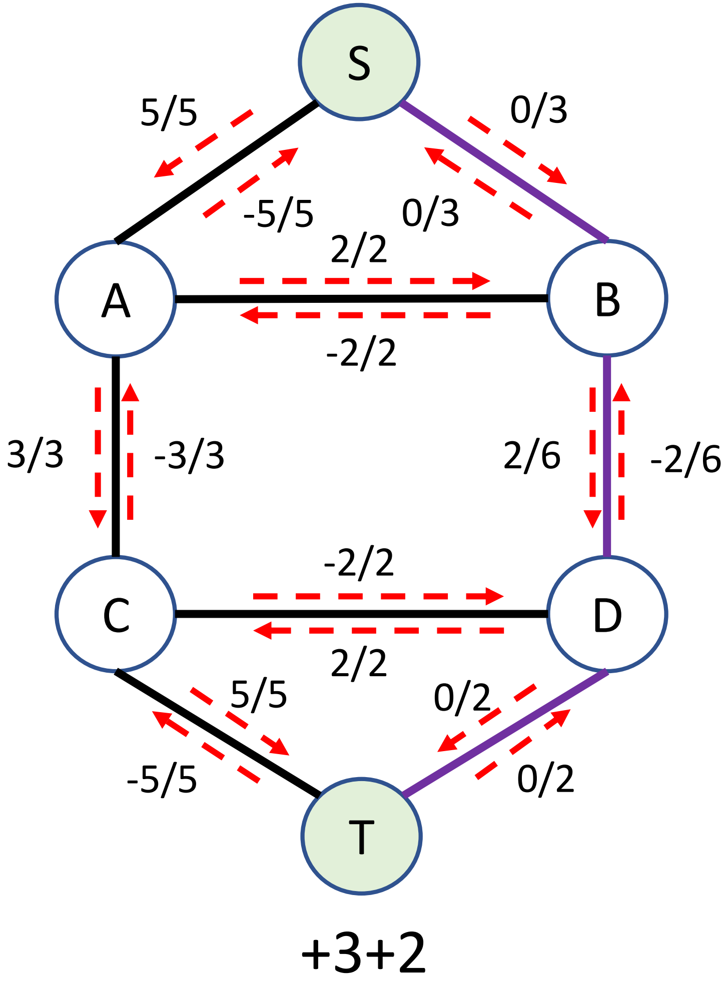
1. Ford-Fulkerson method
    a. 在殘餘網路中隨意找一條增廣路徑
    b. 利用該增廣路徑增加流量後修正殘餘網路
        ✓ 路徑中最小容量的弧作為增加流量
    c. 重複步驟 a、b，直至找不到增廣路徑為止
    d. 累積流量則為最大流量，時間複雜度為 O(Ef)
        ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
        ✓ f 為該網路的最大流



+3+2

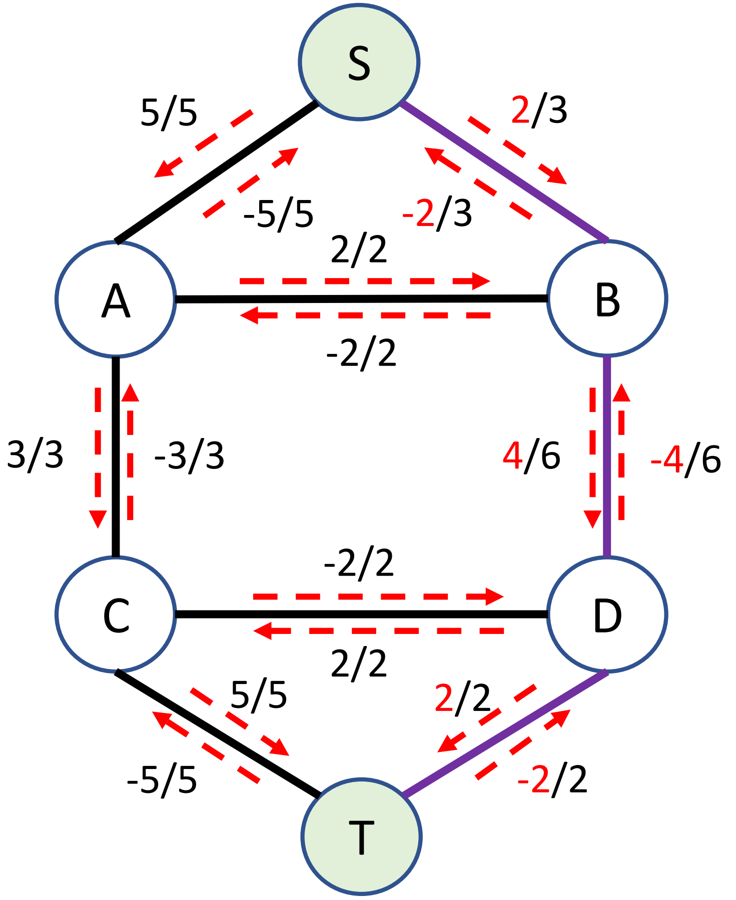
Ford-Fulkerson Algorithm

1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流



Ford-Fulkerson Algorithm

1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流

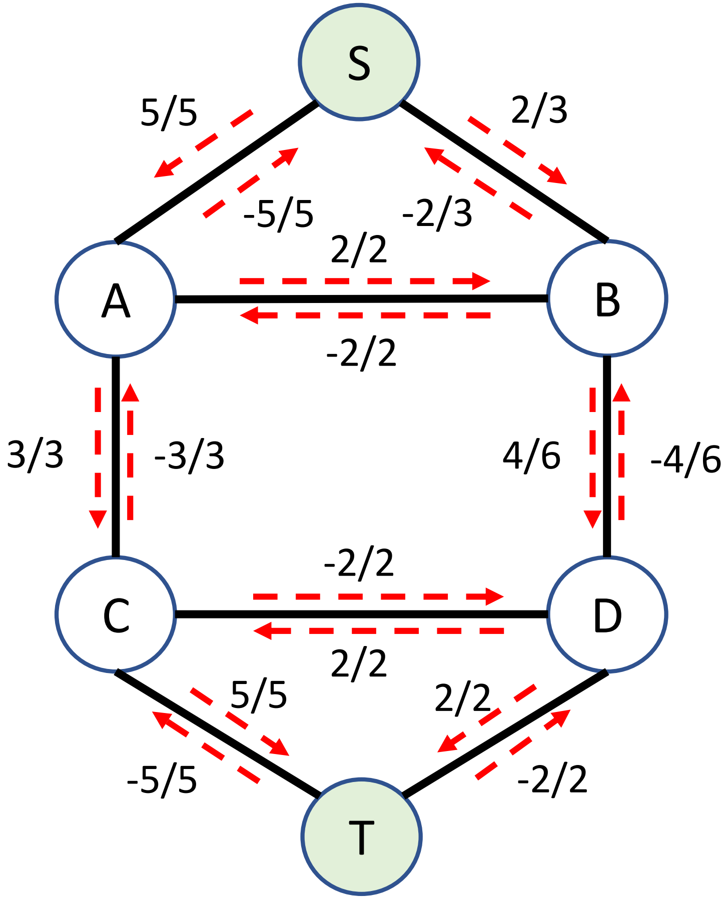


+3+2+2

李耕銘

Ford-Fulkerson Algorithm

1. Ford-Fulkerson method
   a. 在殘餘網路中隨意找一條增廣路徑
   b. 利用該增廣路徑增加流量後修正殘餘網路
      ✓ 路徑中最小容量的弧作為增加流量
   c. 重複步驟 a、b，直至找不到增廣路徑為止
   d. 累積流量則為最大流量，時間複雜度為 O(Ef)
      ✓ V+E 為每次搜尋增廣路徑的時間，且 E ≥ V
      ✓ f 為該網路的最大流



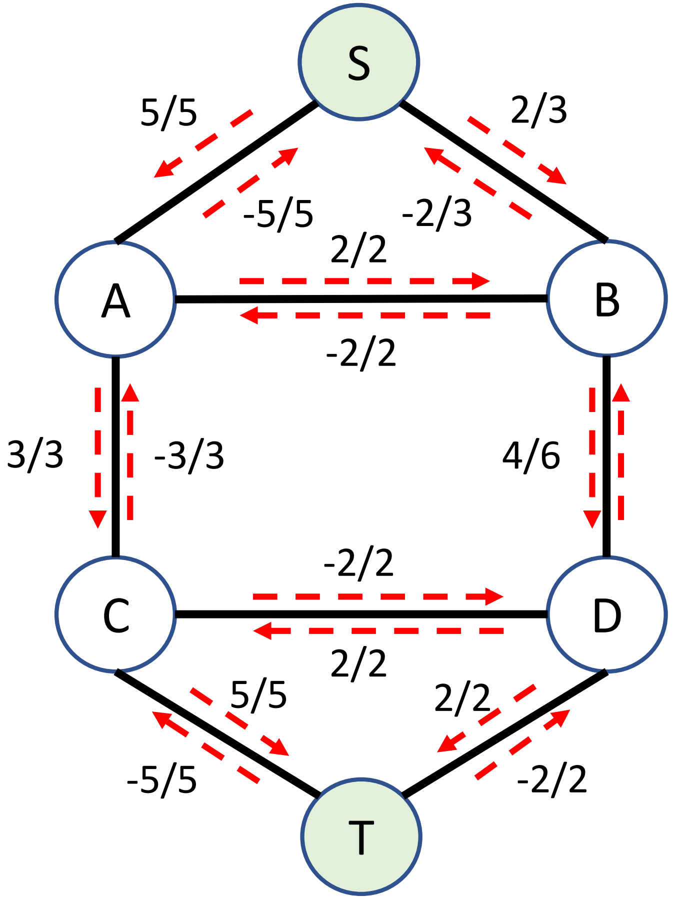
+3+2+2

Ford-Fulkerson Algorithm

Ford-Fulkerson 演算法步驟

1. 初始化所有 flow→ $f(u,v) = 0$
2. 找出一個從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v) > 0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p) = min\{C_f(u,v): (u,v) \in p\}$
   3. 對路徑上的所有弧 $e(u,v) \in p$
      - ✓ $f(u,v) = f(u,v) + C_f(p)$
      - ✓ $f(v,u) = f(v,u) - C_f(p$

- 累積流量則為最大流量，時間複雜度為 O(Ef)
  - ✓ V+E 為每次搜尋增廣路徑的時間
  - ✓ f 為該網路的最大流，假設為整數



+3+2+2

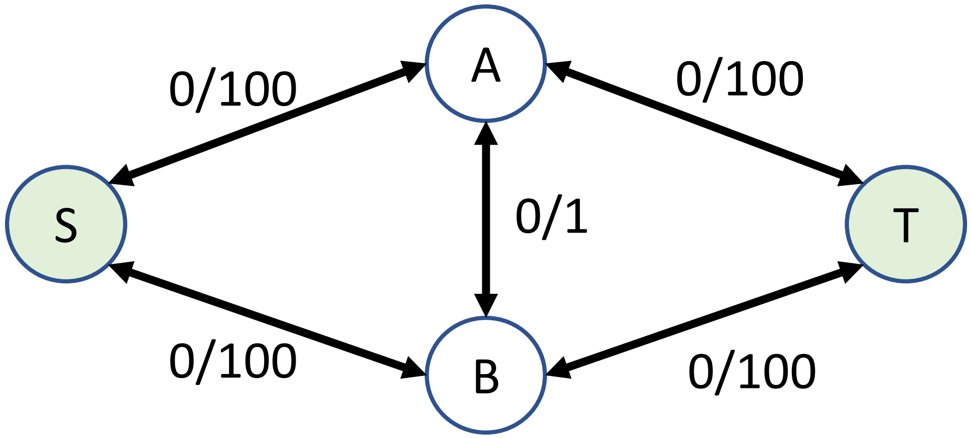
Edmonds-Karp Algorithm

Ford-Fulkerson Algorithm

Ford-Fulkerson method

a. 極端的狀況下會需要重複許多次運算
   - ✓ 每次增廣路徑都只能新增一個流量
   - ✓ 時間複雜度為 O(Ef)，E 為邊/弧的數目、f 為該網路的最大流

b. 利用廣度優先搜尋 (BFS) 可改善此狀況

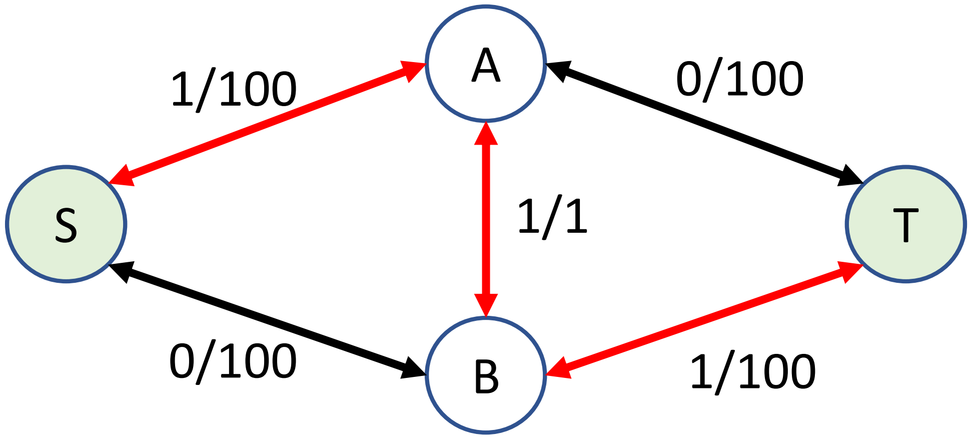


Ford-Fulkerson Algorithm

Ford-Fulkerson method

a. 極端的狀況下會需要重複許多次運算
   - ✓ 每次增廣路徑都只能新增一個流量
   - ✓ 時間複雜度為 O(Ef)，E 為邊/弧的數目、f 為該網路的最大流

b. 利用廣度優先搜尋 (BFS) 可改善此狀況

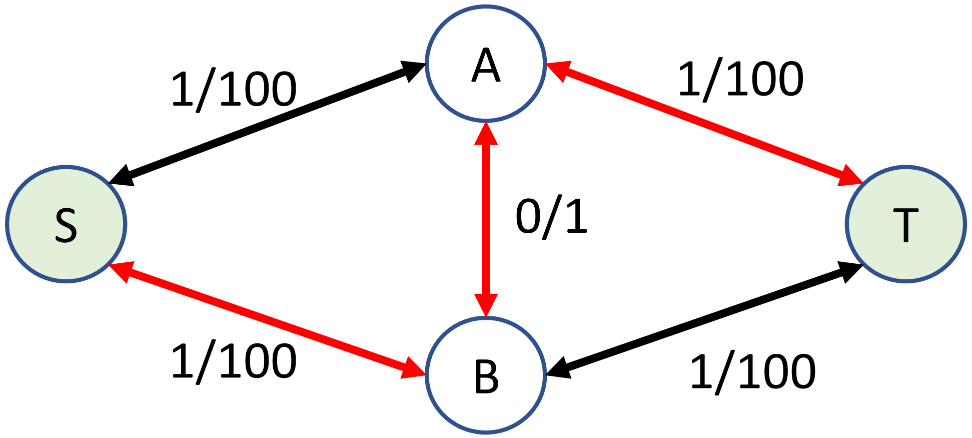


Ford-Fulkerson Algorithm

Ford-Fulkerson method

a. 極端的狀況下會需要重複許多次運算
   - ✓ 每次增廣路徑都只能新增一個流量
   - ✓ 時間複雜度為 O(Ef)，E 為邊/弧的數目、f 為該網路的最大流

b. 利用廣度優先搜尋 (BFS) 可改善此狀況

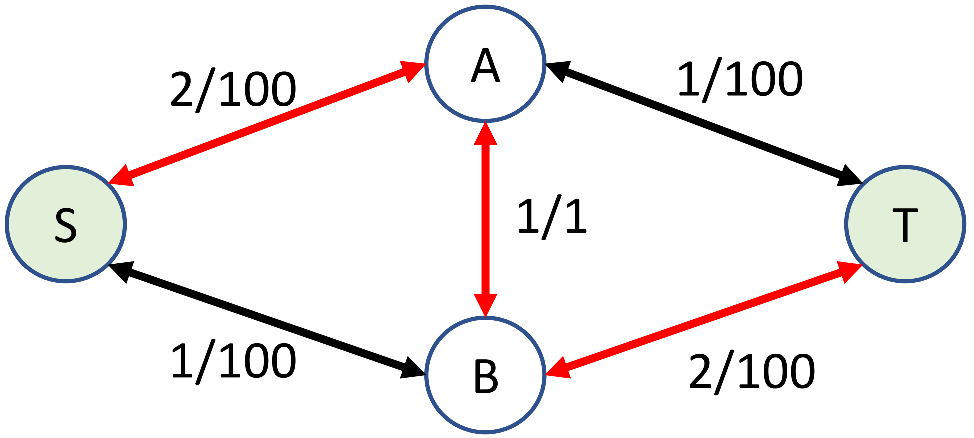


Ford-Fulkerson Algorithm

Ford-Fulkerson method

a. 極端的狀況下會需要重複許多次運算
   - ✓ 每次增廣路徑都只能新增一個流量
   - ✓ 時間複雜度為 O(Ef)，E 為邊/弧的數目、f 為該網路的最大流

b. 利用廣度優先搜尋 (BFS) 可改善此狀況

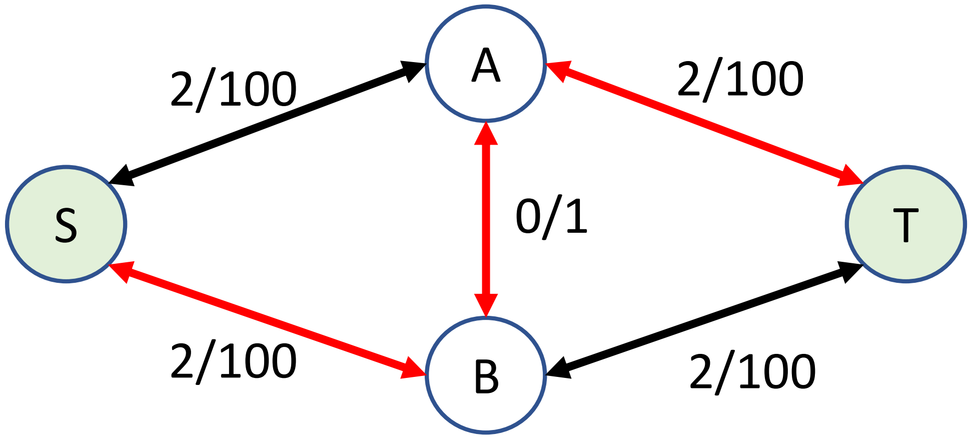


Ford-Fulkerson Algorithm

Ford-Fulkerson method

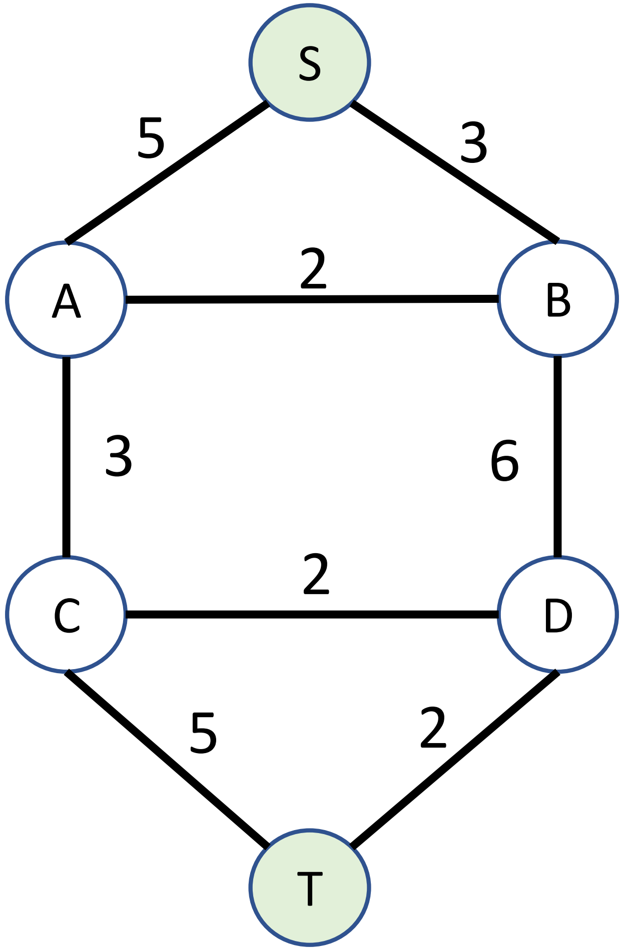
a. 極端的狀況下會需要重複許多次運算
   - ✓ 每次增廣路徑都只能新增一個流量
   - ✓ 時間複雜度為 O(Ef)，E 為邊/弧的數目、f 為該網路的最大流

b. 利用廣度優先搜尋 (BFS) 可改善此狀況



Edmonds-Karp Algorithm

2. Edmonds-Karps algorithm
   - ✓ 與 Ford-Fulkerson method 雷同
     - ◻ 找增廣路徑時以廣度優先搜尋(BFS)尋找
     - ◻ 每次找到的增廣路徑必經過最少的邊/弧

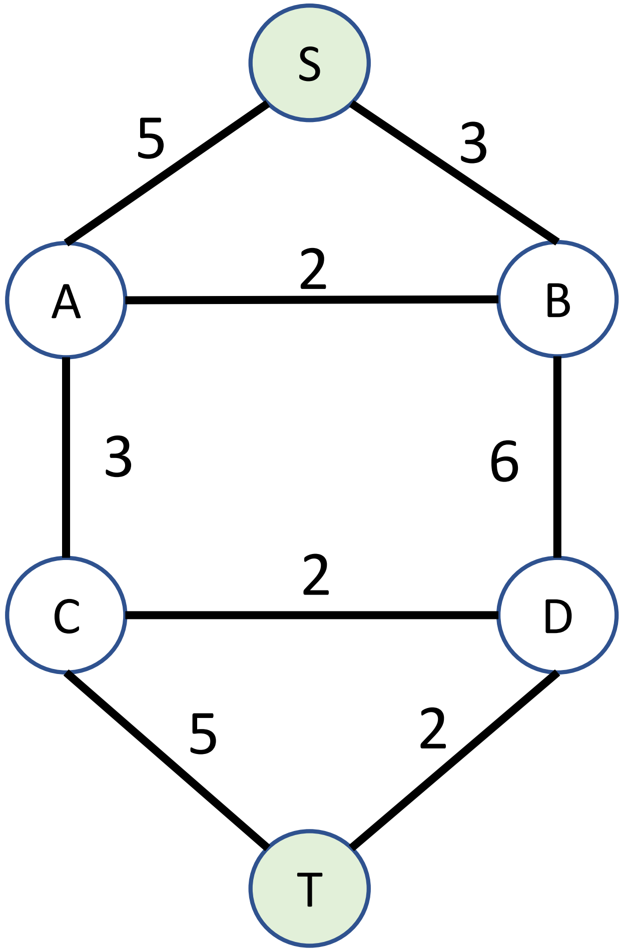


Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v)=0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v)>0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p)=min\{C_f(u,v):(u,v)\in p\}$
   3. 對路徑上的所有弧 $e(u,v)\in p$
      - ✓ $f(u,v)=f(u,v)+C_f(p)$
      - ✓ $f(v,u)=f(v,u)-C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big)=O(VE^2)$
  - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
  - ✓ $O(VE)$ 為增廣路徑最多的數目

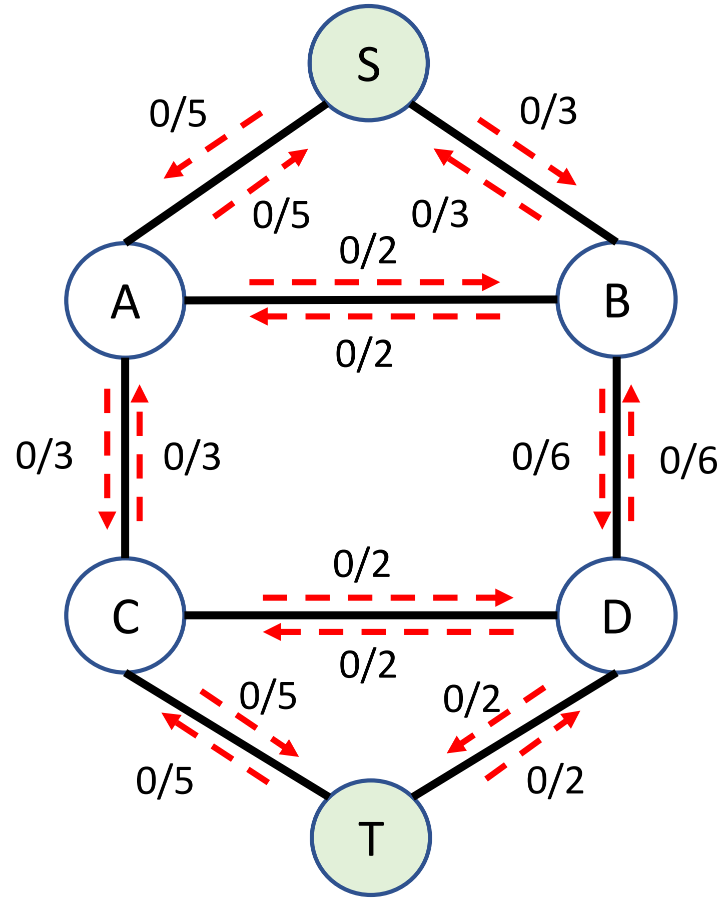


Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v)=0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v)>0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p)=min\{C_f(u,v):(u,v)\in p\}$
   3. 對路徑上的所有弧 $e(u,v)\in p$
      - ✓ $f(u,v)=f(u,v)+C_f(p)$
      - ✓ $f(v,u)=f(v,u)-C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big)=O(VE^2)$
  - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
  - ✓ $O(VE)$ 為增廣路徑最多的數目

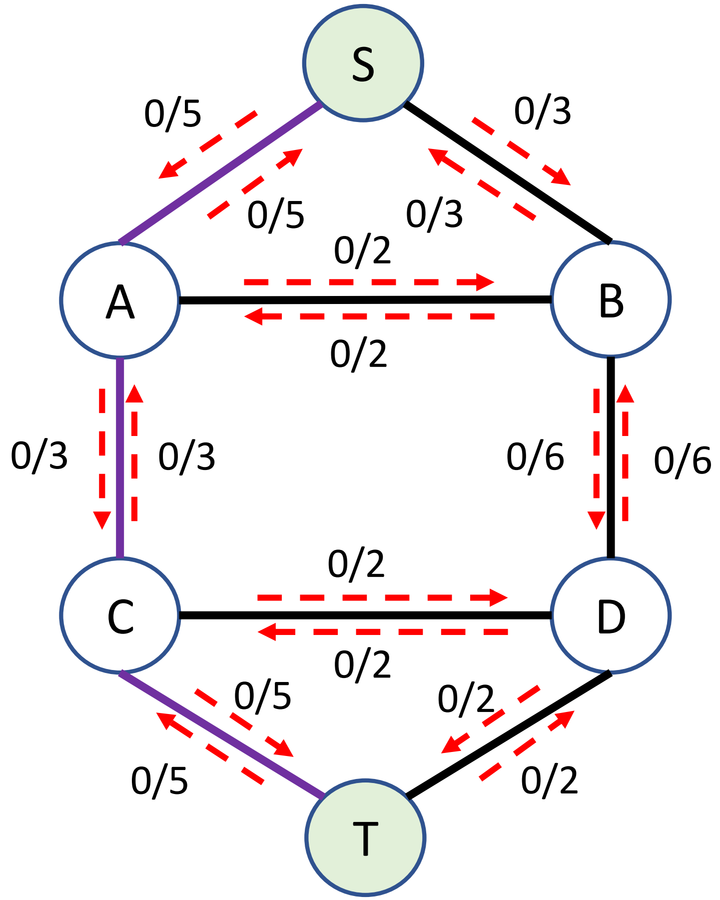


Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v)=0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v)>0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p)=min\{C_f(u,v):(u,v)\in p\}$
   3. 對路徑上的所有弧 $e(u,v)\in p$
      - ✓ $f(u,v)=f(u,v)+C_f(p)$
      - ✓ $f(v,u)=f(v,u)-C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big)=O(VE^2)$
  - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
  - ✓ $O(VE)$ 為增廣路徑最多的數目

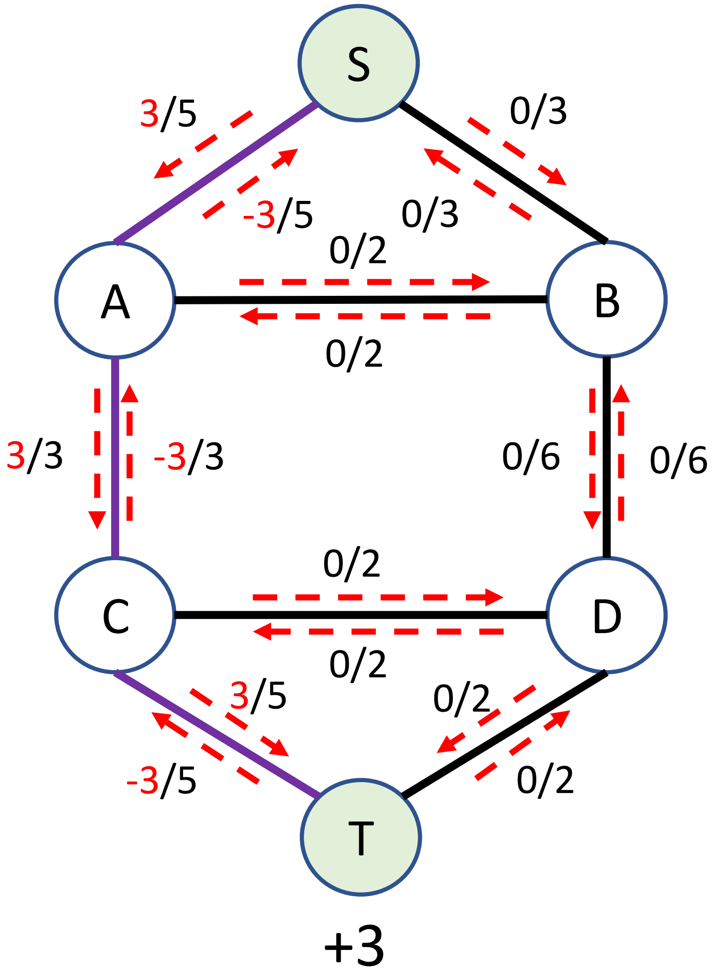


Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v)=0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v)>0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p)=min\{C_f(u,v):(u,v)\in p\}$
   3. 對路徑上的所有弧 $e(u,v)\in p$
      - ✓ $f(u,v)=f(u,v)+C_f(p)$
      - ✓ $f(v,u)=f(v,u)-C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big)=O(VE^2)$
  - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
  - ✓ $O(VE)$ 為增廣路徑最多的數目

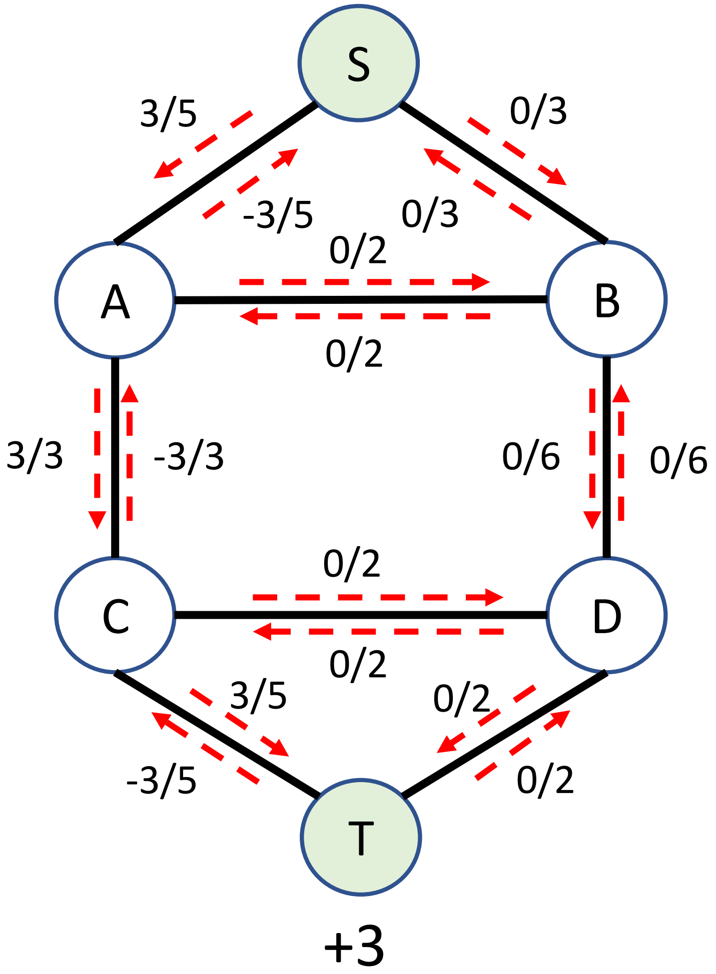


Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v) = 0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v) > 0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p) = min\{C_f(u,v): (u,v) \in p\}$
   3. 對路徑上的所有弧 $e(u,v) \in p$
      - ✓ $f(u,v) = f(u,v) + C_f(p)$
      - ✓ $f(v,u) = f(v,u) - C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big) = O(VE^2)$
  - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
  - ✓ $O(VE)$ 為增廣路徑最多的數目

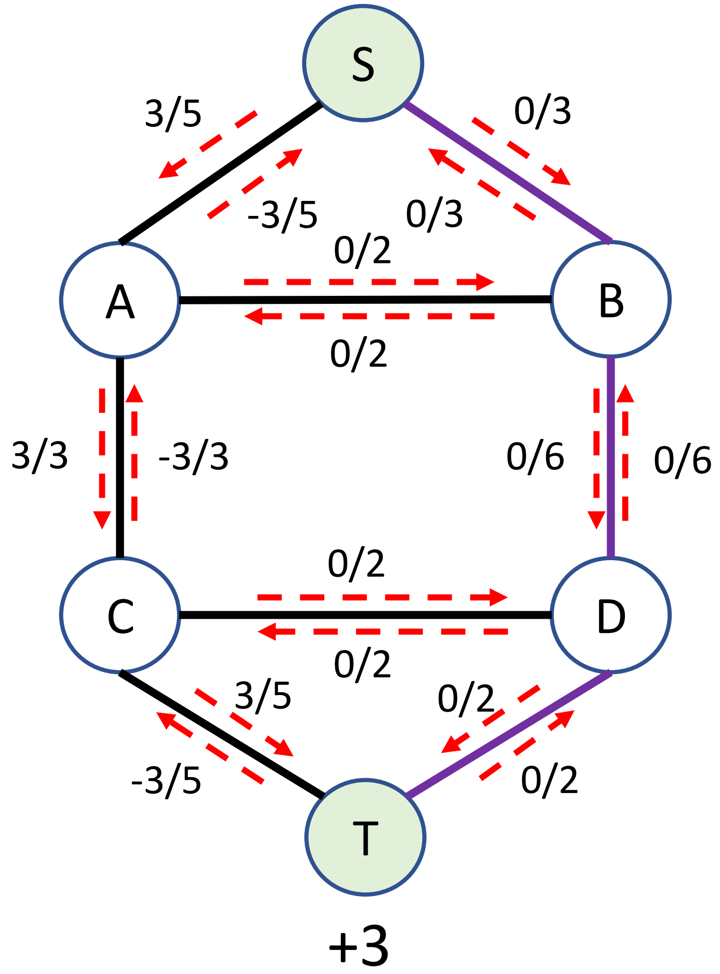


Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v)=0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
    1. 該路徑上的所有弧都滿足 $Cf(u,v)>0$
    2. 找出路徑上最小殘餘容量的弧
        - ✓ $C_f(p)=min\{C_f(u,v):(u,v)\in p\}$
    3. 對路徑上的所有弧 $e(u,v)\in p$
        - ✓ $f(u,v)=f(u,v)+C_f(p)$
        - ✓ $f(v,u)=f(v,u)-C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big)=O(VE^2)$
    - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
    - ✓ $O(VE)$ 為增廣路徑最多的數目

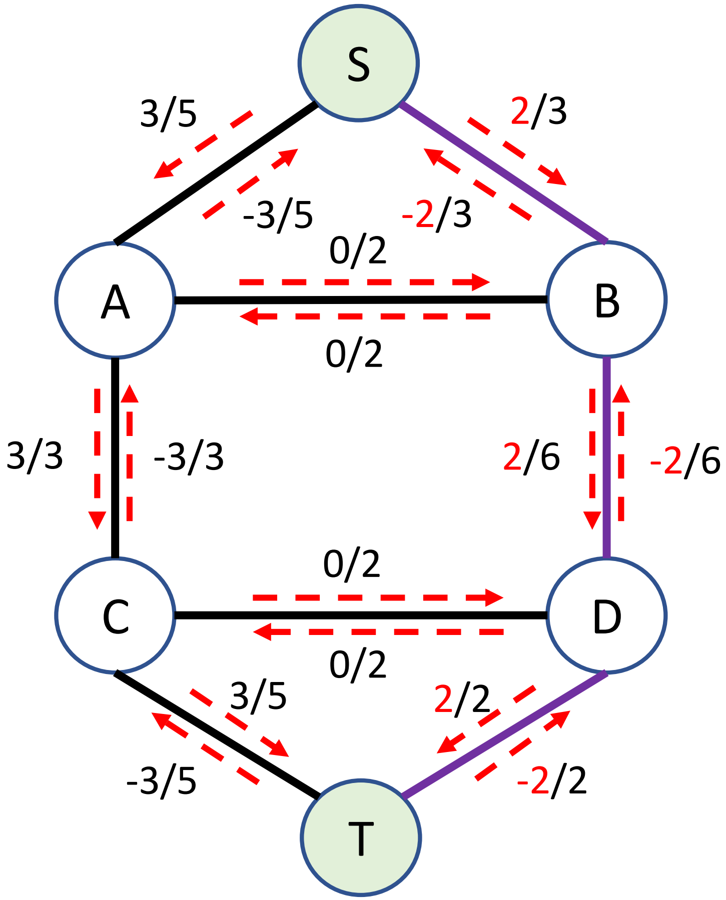


Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v) = 0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v) > 0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p) = min\{C_f(u,v): (u,v) \in p\}$
   3. 對路徑上的所有弧 $e(u,v) \in p$
      - ✓ $f(u,v) = f(u,v) + C_f(p)$
      - ✓ $f(v,u) = f(v,u) - C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big) = O(VE^2)$
  - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
  - ✓ $O(VE)$ 為增廣路徑最多的數目



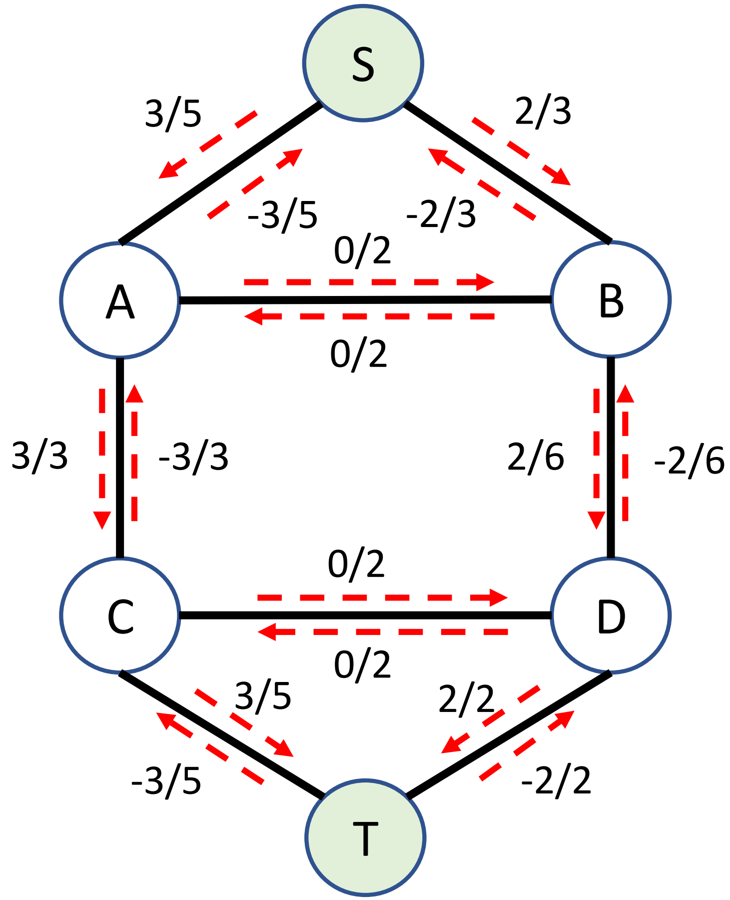
+3+2

Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v)=0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v)>0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p)=min\{C_f(u,v):(u,v)\in p\}$
   3. 對路徑上的所有弧 $e(u,v)\in p$
      - ✓ $f(u,v)=f(u,v)+C_f(p)$
      - ✓ $f(v,u)=f(v,u)-C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big)=O(VE^2)$
  - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
  - ✓ $O(VE)$ 為增廣路徑最多的數目



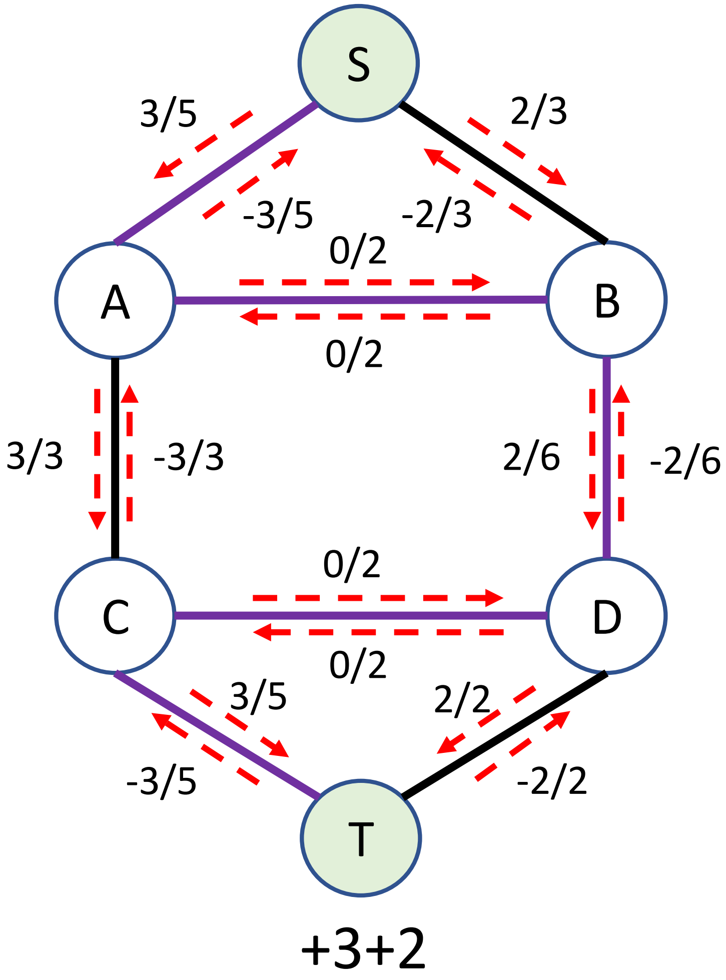
+3+2

Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v) = 0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
    1. 該路徑上的所有弧都滿足 $Cf(u,v) > 0$
    2. 找出路徑上最小殘餘容量的弧
        - ✓ $C_f(p) = min\{C_f(u,v): (u,v) \in p\}$
    3. 對路徑上的所有弧 $e(u,v) \in p$
        - ✓ $f(u,v) = f(u,v) + C_f(p)$
        - ✓ $f(v,u) = f(v,u) - C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big) = O(VE^2)$
    - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
    - ✓ $O(VE)$ 為增廣路徑最多的數目

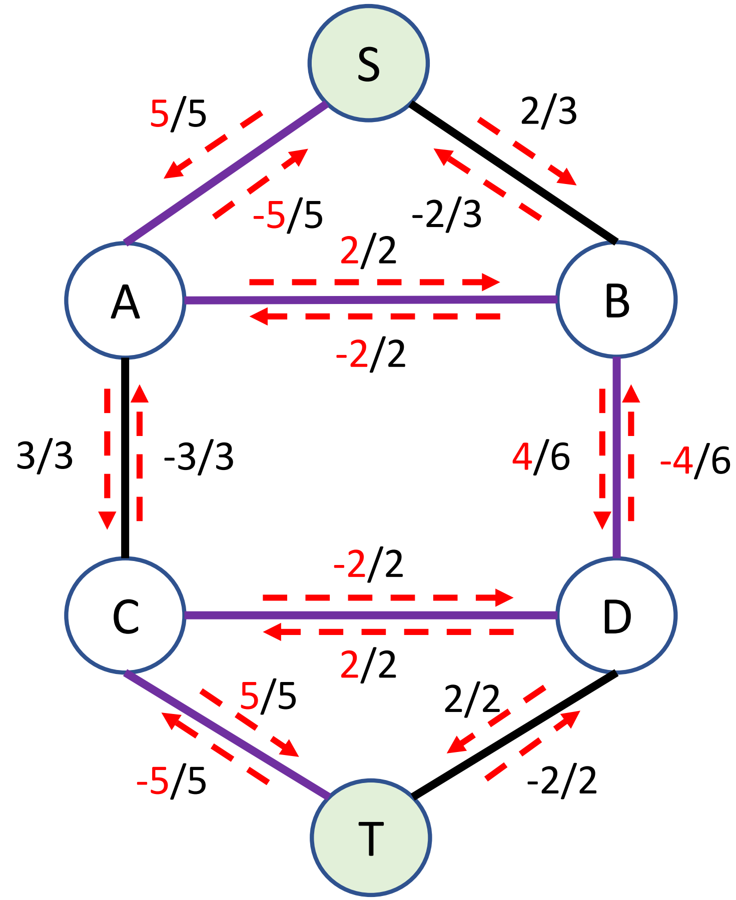


Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v)=0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
    1. 該路徑上的所有弧都滿足 $Cf(u,v)>0$
    2. 找出路徑上最小殘餘容量的弧
        - ✓ $C_f(p)=min\{C_f(u,v):(u,v)\in p\}$
    3. 對路徑上的所有弧 $e(u,v)\in p$
        - ✓ $f(u,v)=f(u,v)+C_f(p)$
        - ✓ $f(v,u)=f(v,u)-C_f(p$

- 總複雜度為 $O\big((V+E)(VE)\big)=O(VE^2)$
    - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
    - ✓ $O(VE)$ 為增廣路徑最多的數目



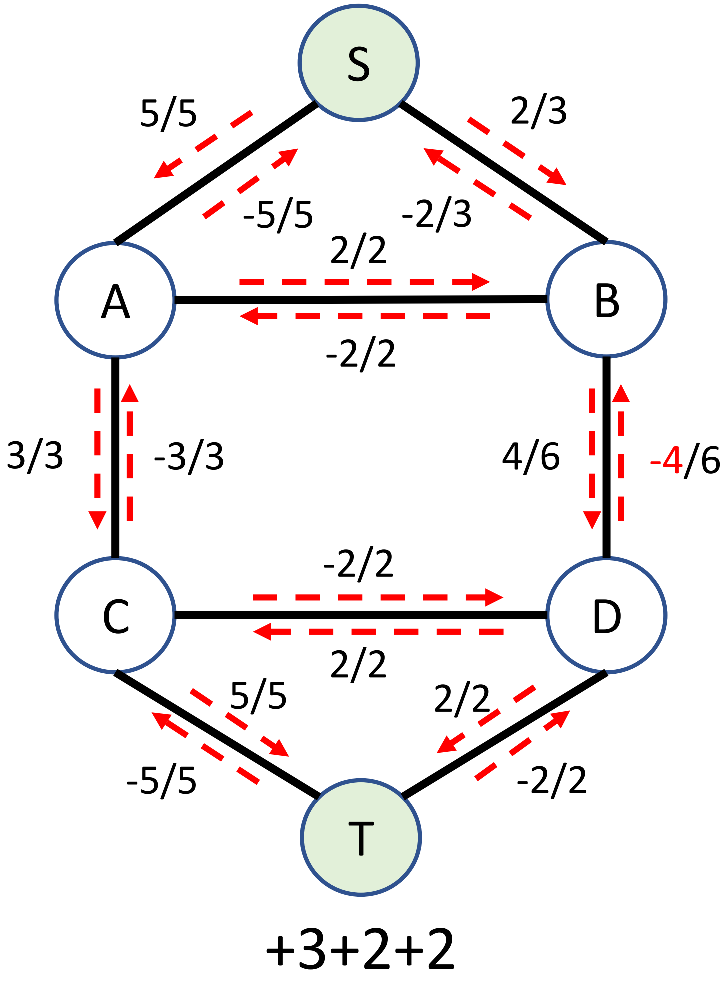
+3+2+2

Edmonds-Karps Algorithm

Edmonds-Karps 演算法步驟

1. 初始化所有 flow→ $f(u,v)=0$
2. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
   1. 該路徑上的所有弧都滿足 $Cf(u,v)>0$
   2. 找出路徑上最小殘餘容量的弧
      - ✓ $C_f(p)=min\{C_f(u,v):(u,v)\in p\}$
   3. 對路徑上的所有弧 $e(u,v)\in p$
      - ✓ $f(u,v)=f(u,v)+C_f(p)$
      - ✓ $f(v,u)=f(v,u)-C_f(p$

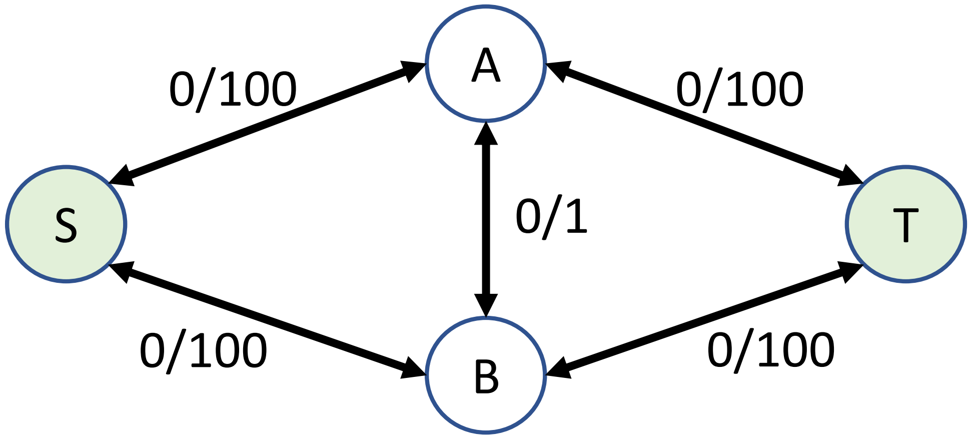
- 總複雜度為 $O\big((V+E)(VE)\big)=O(VE^2)$
  - ✓ $O(V+E)$ 為每次搜尋增廣路徑的時間
  - ✓ $O(VE)$ 為增廣路徑最多的數目



Edmonds-Karps Algorithm

Edmonds-Karps algorithm

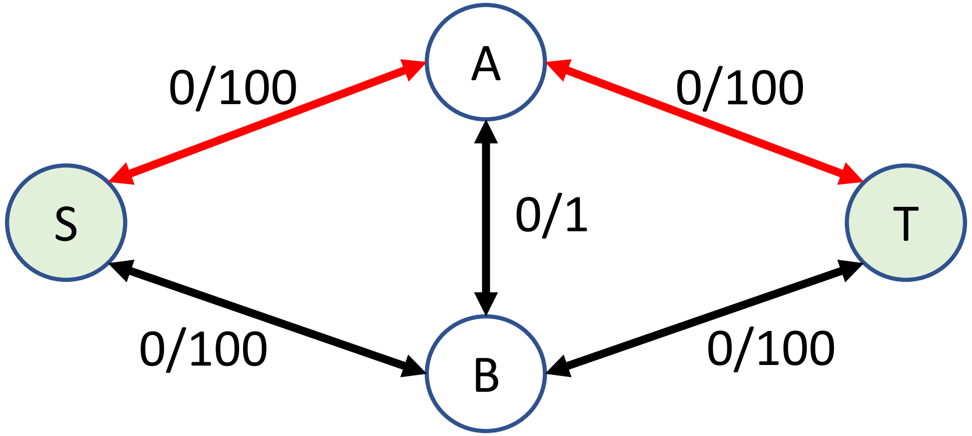
a. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
b. 最多只有 O(VE)條增廣路徑
c. BFS找增廣路徑的複雜度為O(V+E)，且 E ≥ V



Edmonds-Karps Algorithm

Edmonds-Karps algorithm

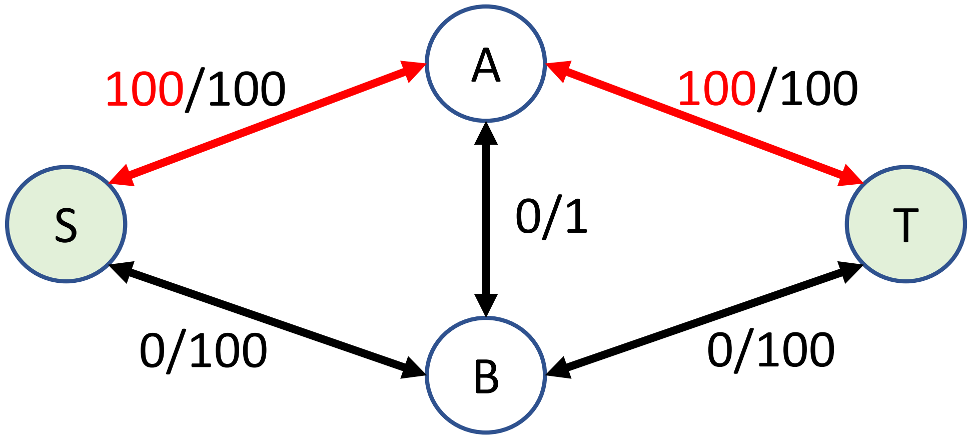
a. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
b. 最多只有 O(VE)條增廣路徑
c. BFS找增廣路徑的複雜度為O(V+E)，且 E ≥ V



Edmonds-Karps Algorithm

Edmonds-Karps algorithm

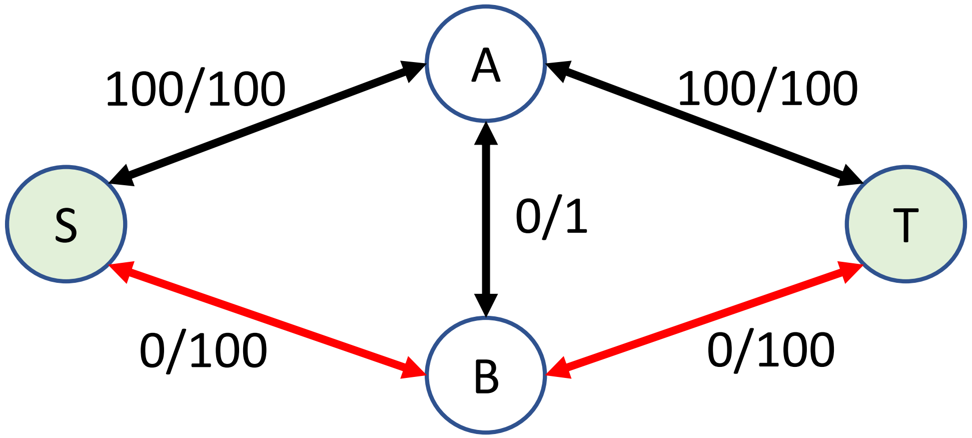
a. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
b. 最多只有 O(VE)條增廣路徑
c. BFS找增廣路徑的複雜度為O(V+E)，且 E ≥ V



Edmonds-Karps Algorithm

Edmonds-Karps algorithm

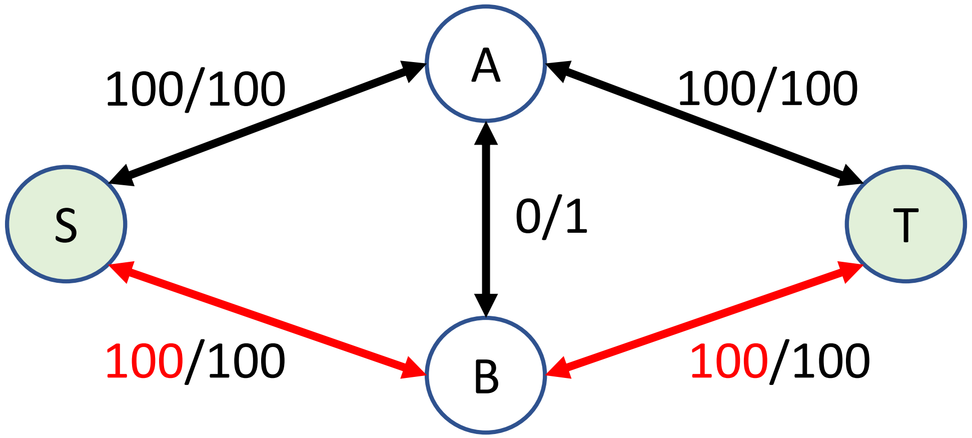
a. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
b. 最多只有 O(VE)條增廣路徑
c. BFS找增廣路徑的複雜度為O(V+E)，且 E ≥ V



Edmonds-Karps Algorithm

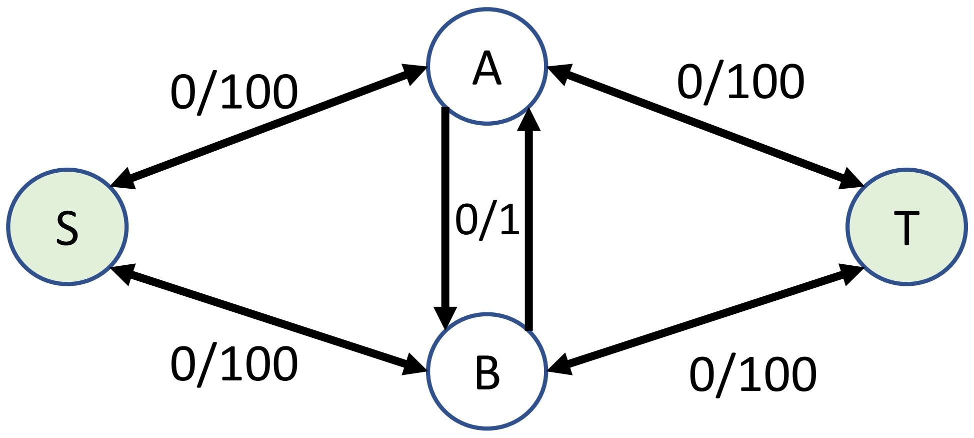
Edmonds-Karps algorithm

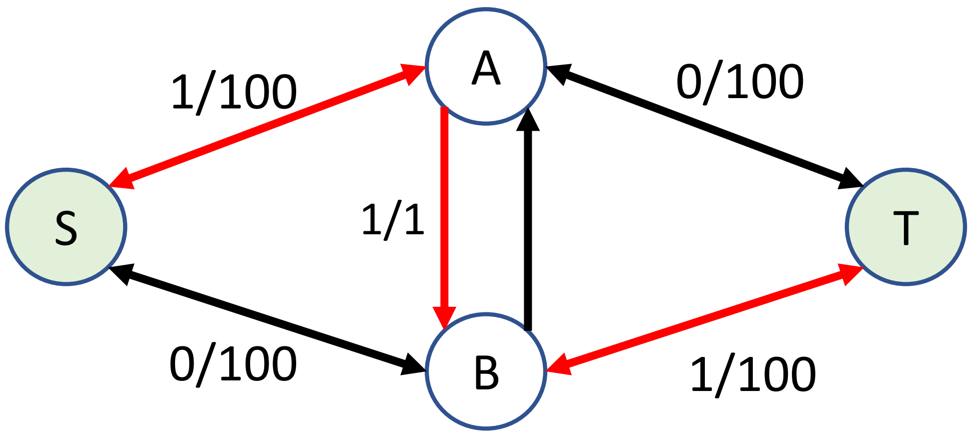
a. 以廣度優先搜尋(BFS)找出從 S → T 的增廣路徑
b. 最多只有 O(VE)條增廣路徑
c. BFS找增廣路徑的複雜度為O(V+E)，且 E ≥ V



Edmonds-Karps Algorithm

- 為何最多只有 O(VE)條增廣路徑？
  1. 若 $e(u,v)$ 為某次增廣路徑中容量最小的弧，則經過該次增廣後
     - ✓ $e(u,v)$ 為飽和弧，即 $e(u,v)$ 不能再被採納入後續的增廣路徑中
     - ✓ 若 $e(u,v)$ 再次出現在增廣路徑中，則必沿 $e(v,u)$ 方向增廣
     - ✓ EX：$\overline{AB}$ 不能再出現在後續的增廣路徑中，$\overline{BA}$ 才可以

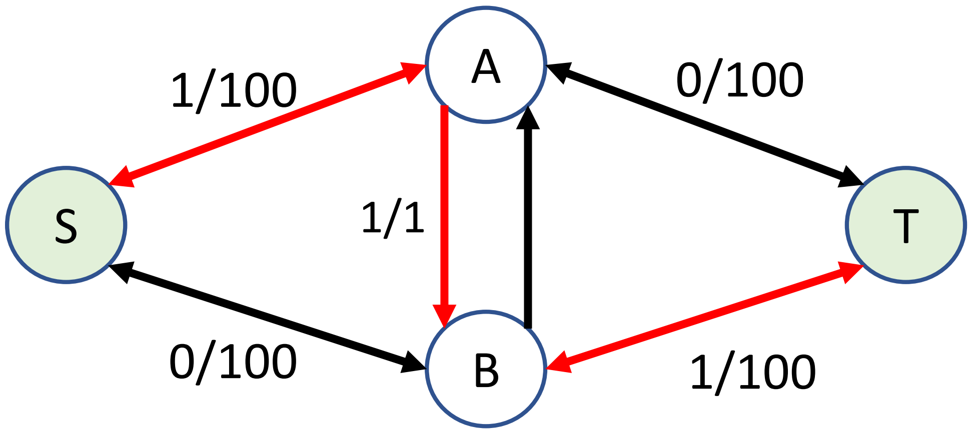


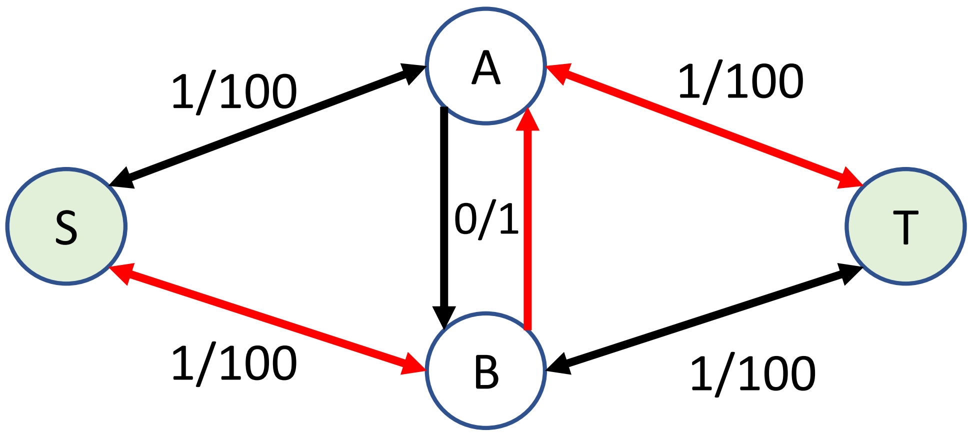


Edmonds-Karps Algorithm

- 為何最多只有 O(VE)條增廣路徑？

2. 讓 $e(u,v)$ 消失的該增廣路徑中，$\delta f(s,v) = \delta f(s,u) + 1$
   - ✓ δ 表最短路徑的長度
   - ✓ $\delta f(s,B) = \delta f(s,A) + 1$
3. 讓 $e(u,v)$ 再次出現的增廣路徑中，$\delta f'(s,v) = \delta f'(s,u) - 1$
   - ✓ $\delta f'(s,B) = \delta f'(s,A) - 1$



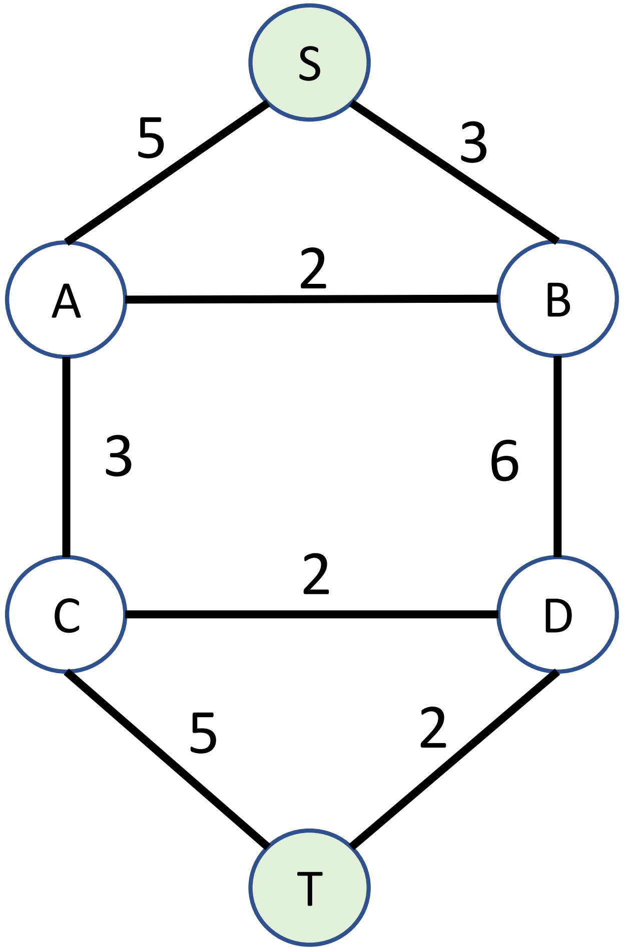


Edmonds-Karps Algorithm

- 為何最多只有 O(VE)條增廣路徑？
  4. BFS 中最短增廣路徑的長度遞增或持平，故 $\delta f(s,u)+1 \le \delta f'(s,u)-1$
     - ✓ 已知
       - $\delta f(s,v)=\delta f(s,u)+1$
       - $\delta f'(s,v)=\delta f'(s,u)-1$
     - ✓ $\delta f(s,u)+1 \le \delta f'(s,u)-1$
     - ✓ $\delta f(s,u)+2 \le \delta f'(s,u)$
  5. 最短增廣路徑的長度最長為 |V| - 1
     - ✓ 弧成為廣路徑中容量最小的弧 (critical edge) 的次數不超過 (|V|-1)/2
     - ✓ 每次增廣至少有一條 critical edge，且 $|E_f| \le 2|E|$，總增廣次數為 O(VE)

Edmonds-Karp Algorithm

2. Edmonds-Karps algorithm
   a. 與 Ford-Fulkerson method 雷同
      - ✓ 找增廣路徑時以廣度優先搜尋(BFS)尋找
      - ✓ 每次找到的增廣路徑必經過最少的邊/弧
   b. 從頂點的觀點：
      - ✓ 每找出增廣路徑並修正殘餘網路後，相當於消除一條殘餘網路中的最短路徑 (假設弧的距離相等)
      - ✓ 修正後的後向弧也不會縮短最短路徑的長度
   c. 從邊/弧的觀點：
      - ✓ 網路中最多只有 O(VE)條增廣路徑
      - ✓ BFS找增廣路徑的複雜度為O(V+E)，且 E ≥ V
      - ✓ 總複雜度為 $\boldsymbol{O}\big((\boldsymbol{V}+\boldsymbol{E})(\boldsymbol{VE})\big)=\boldsymbol{O}(\boldsymbol{VE}^{2})$



Example Code

Mission

寫出 Edmonds-Karps Algorithm！