

Algorithms (2020 Summer)

#11 : グラフアルゴリズム2

矢谷 浩司

今日の問題：グラフ & フロー

グラフの上に流れ（フロー）が存在する。

例)

A町からB町までの物資の最大流通可能量を求める

ネットワーク上でのある端末から別の端末間で

最大スループットを求める

一番通信費の安いルーティングを求める

SourceとSink

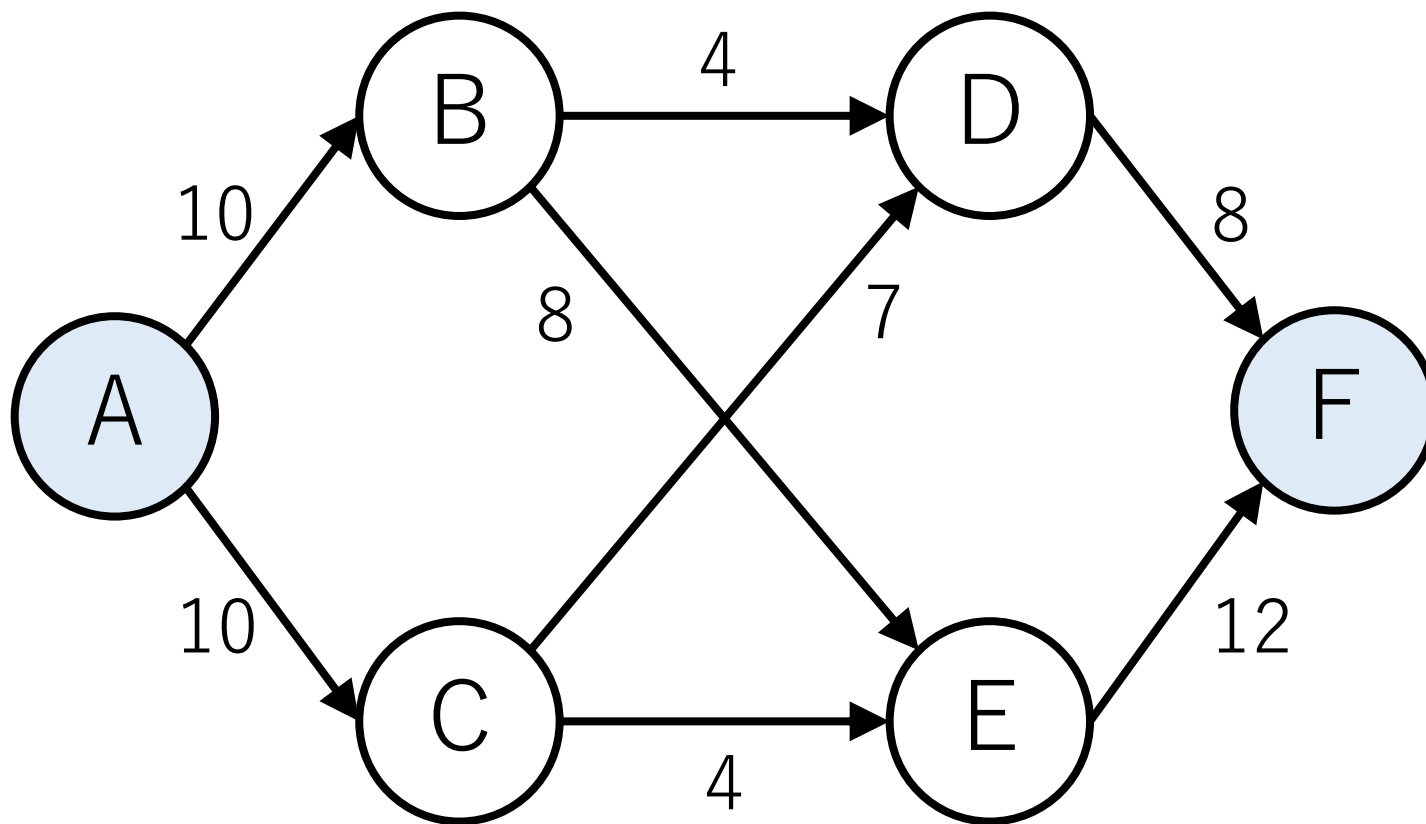
始点からはフローが生まれ (source) , 終点ではフローがなくなる (sink) .

始点と終点ノード以外ではフローが増減しない.

今日の授業では, sourceとsinkは1つしかないものとして説明します.

最大流問題

あるノードから別のノードへの流量の最大値を求める。



貪欲法 (greedy algorithm)

今ある選択肢のうち，なにか1つを選んで次に進む。
(何かしらの順序や評価基準で一番のものを選ぶ
こともある.)

そして，それ以降この状況のことは振り返らない。

ものすごく簡単に実装できる！

貪欲法

必ずしも最適解を導出できるとは限らない. 例えば, 25円を1円, 8円, 13円という3種類の硬貨を使って最小枚数で支払う場合など.

貪欲法 : 13円, 8円, 1円, 1円, 1円, 1円で6枚.

最適解 : 8円, 8円, 8円, 1円で4枚.

「13円を出さない」というケースを考慮できないため.

貪欲法で解く例

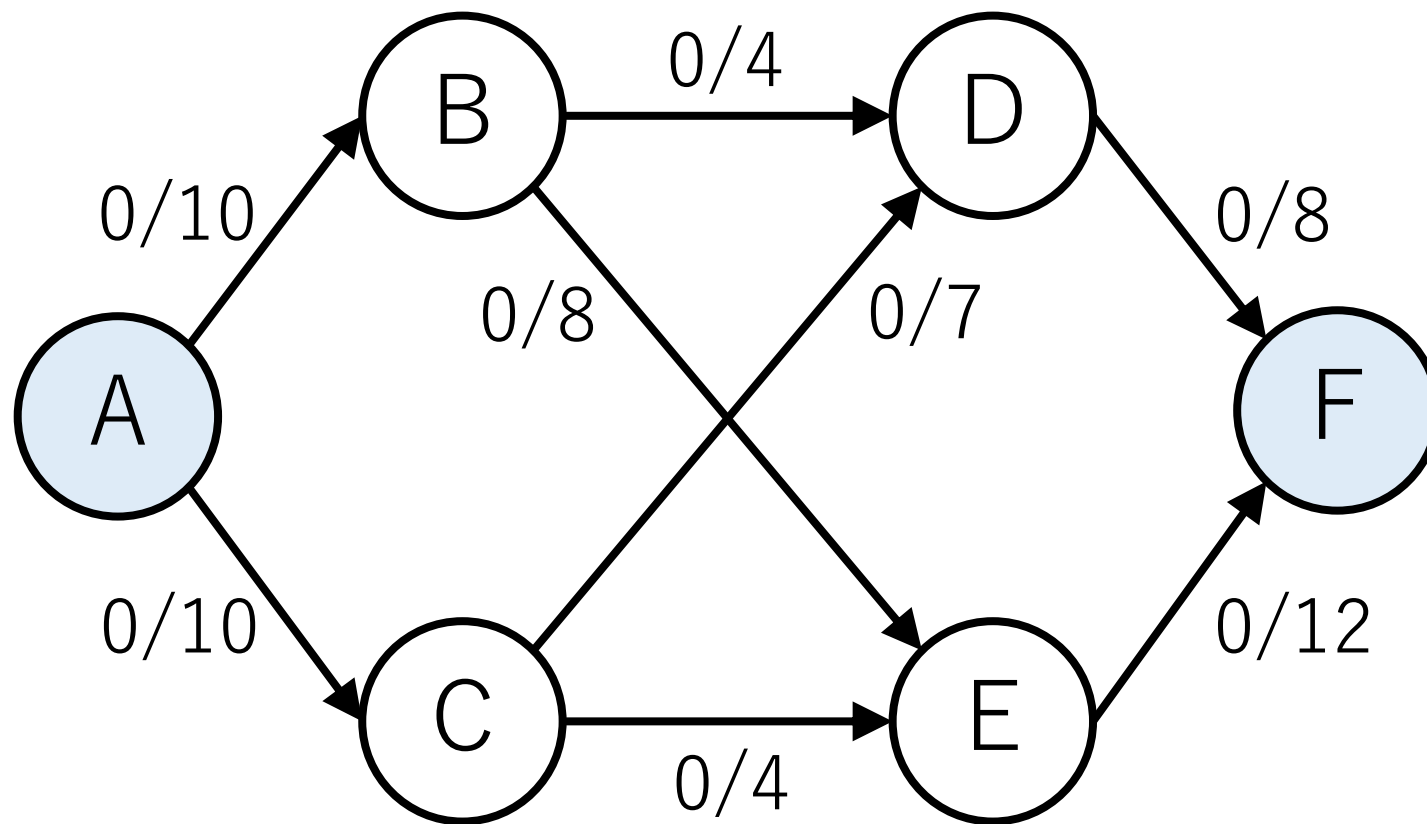
A->Fに至る経路をDFSで見つける.

見つけた経路において流せるだけ流す.

それを全ての経路でやる.

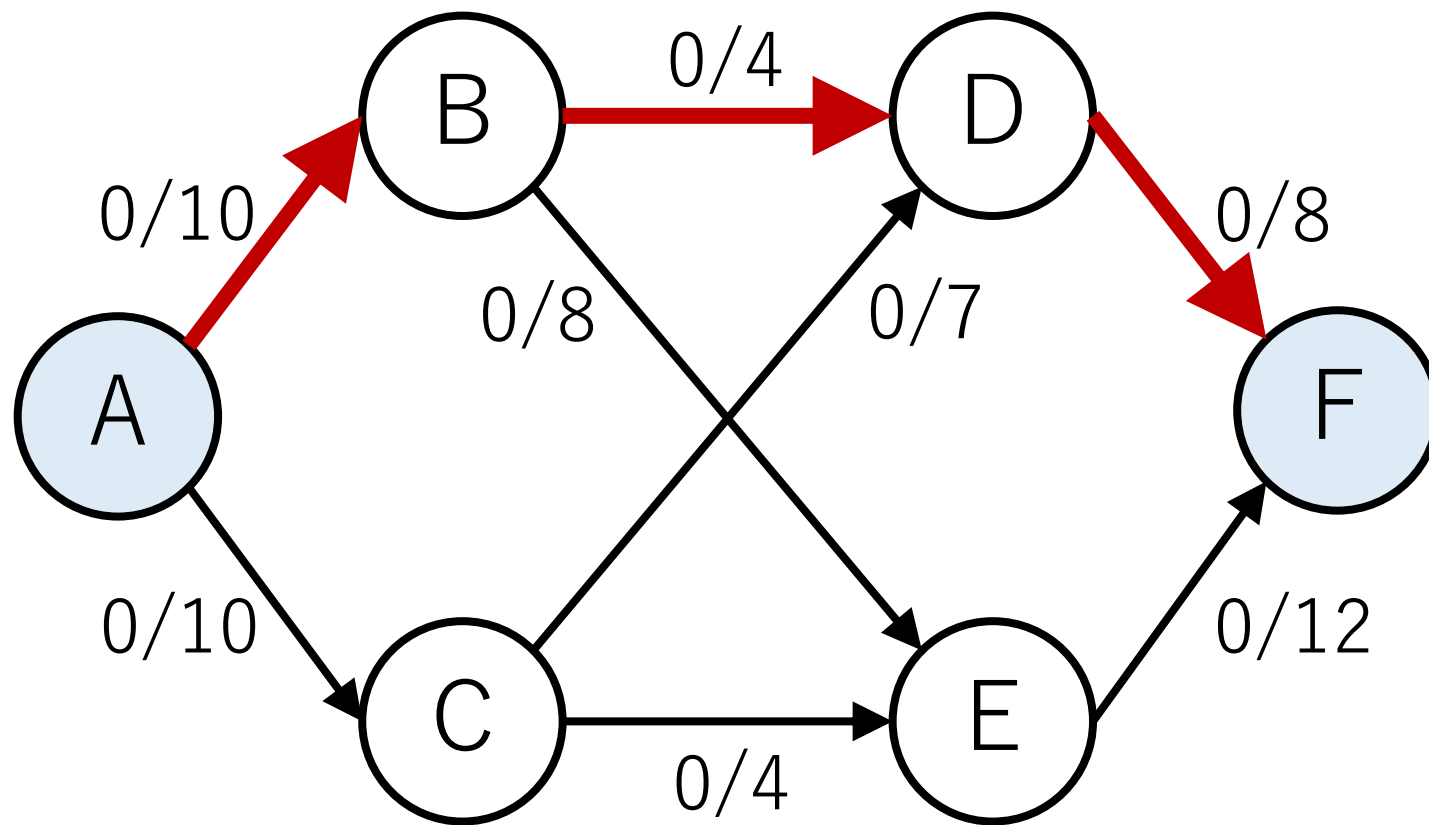
貪欲法による例

AからDFSスタート.



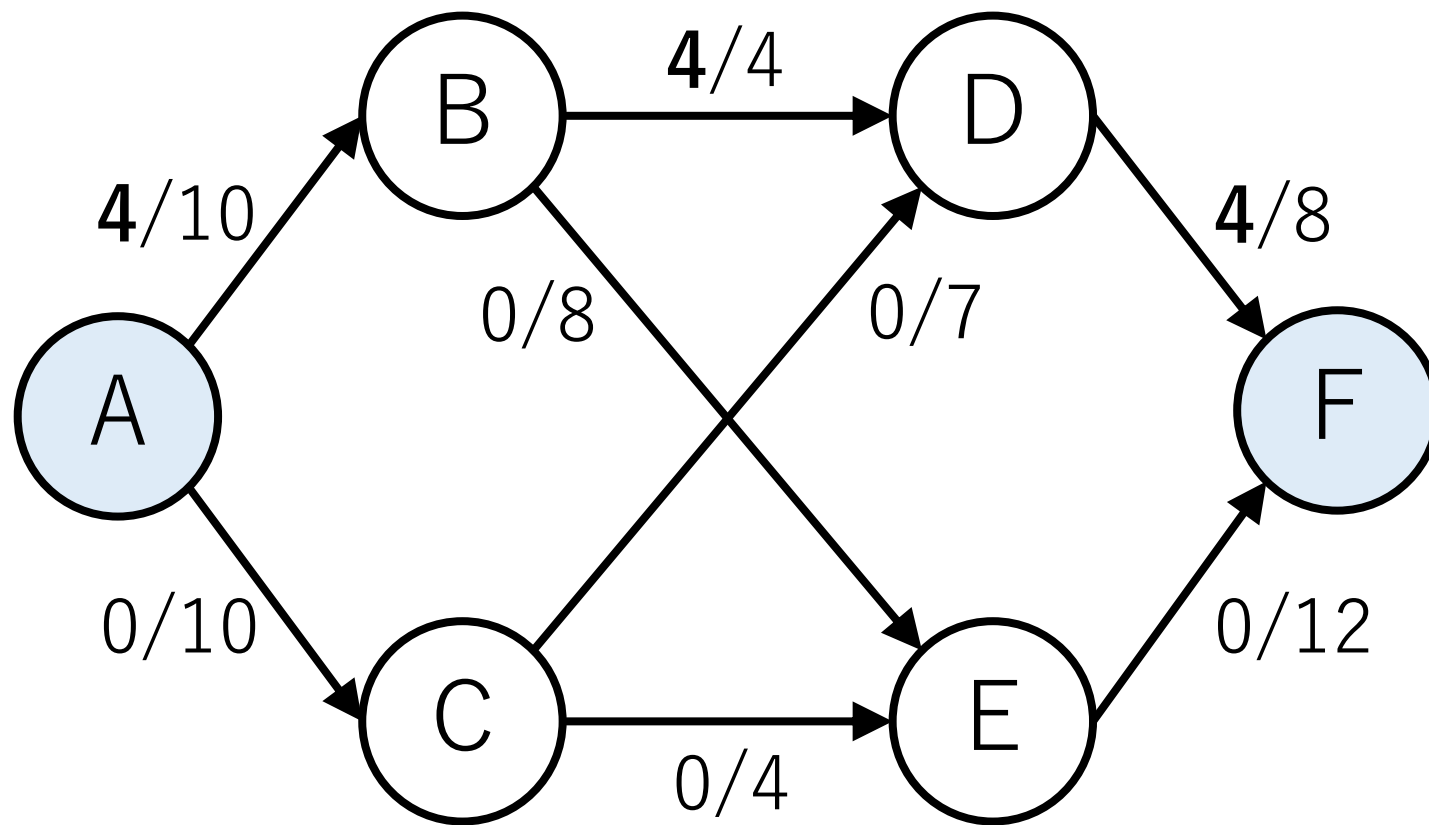
貪欲法による例

A->B->D->Fが見つかる。ここは4流せる。



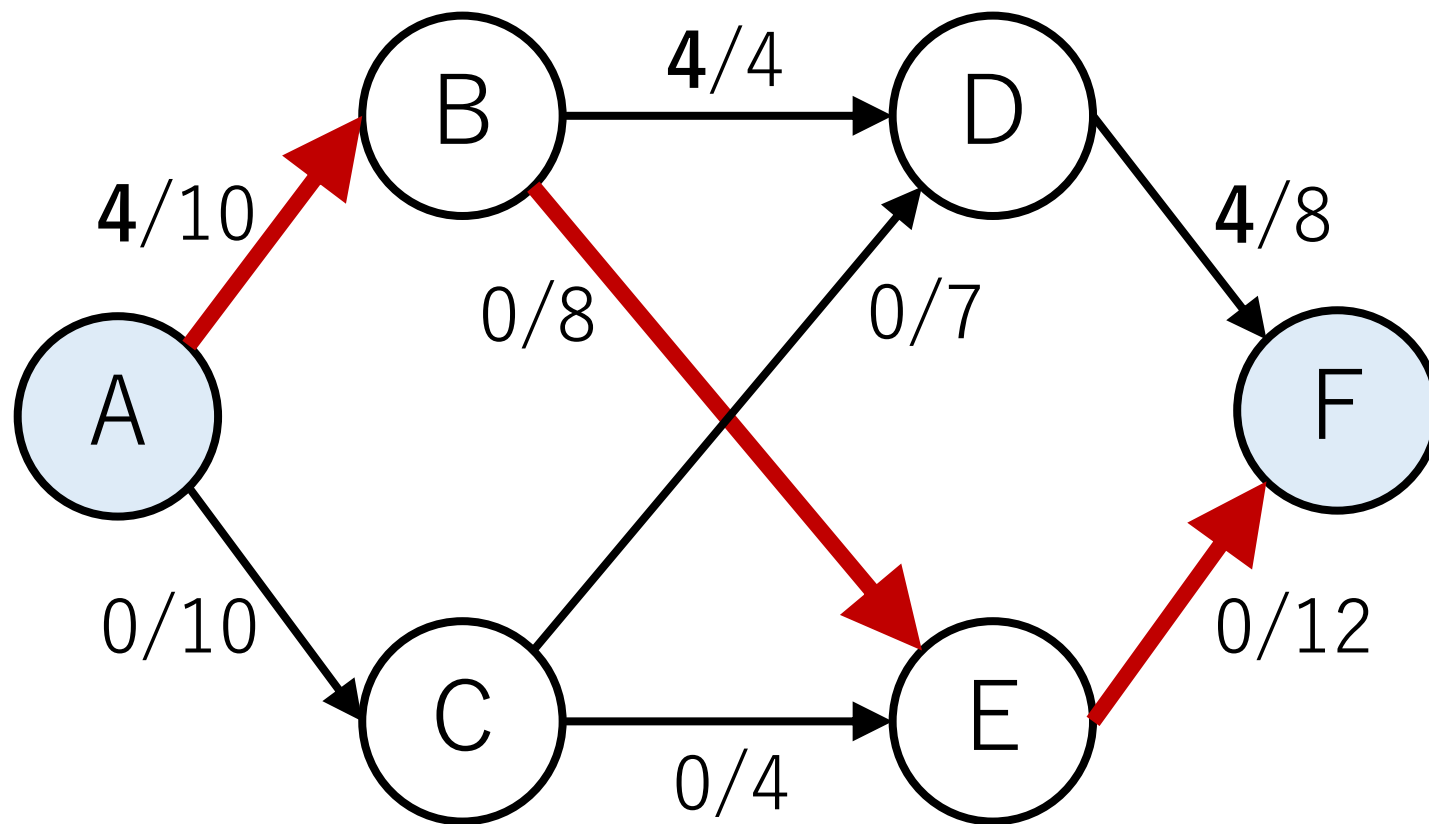
貪欲法による例

A->B->D->Fに4流す。



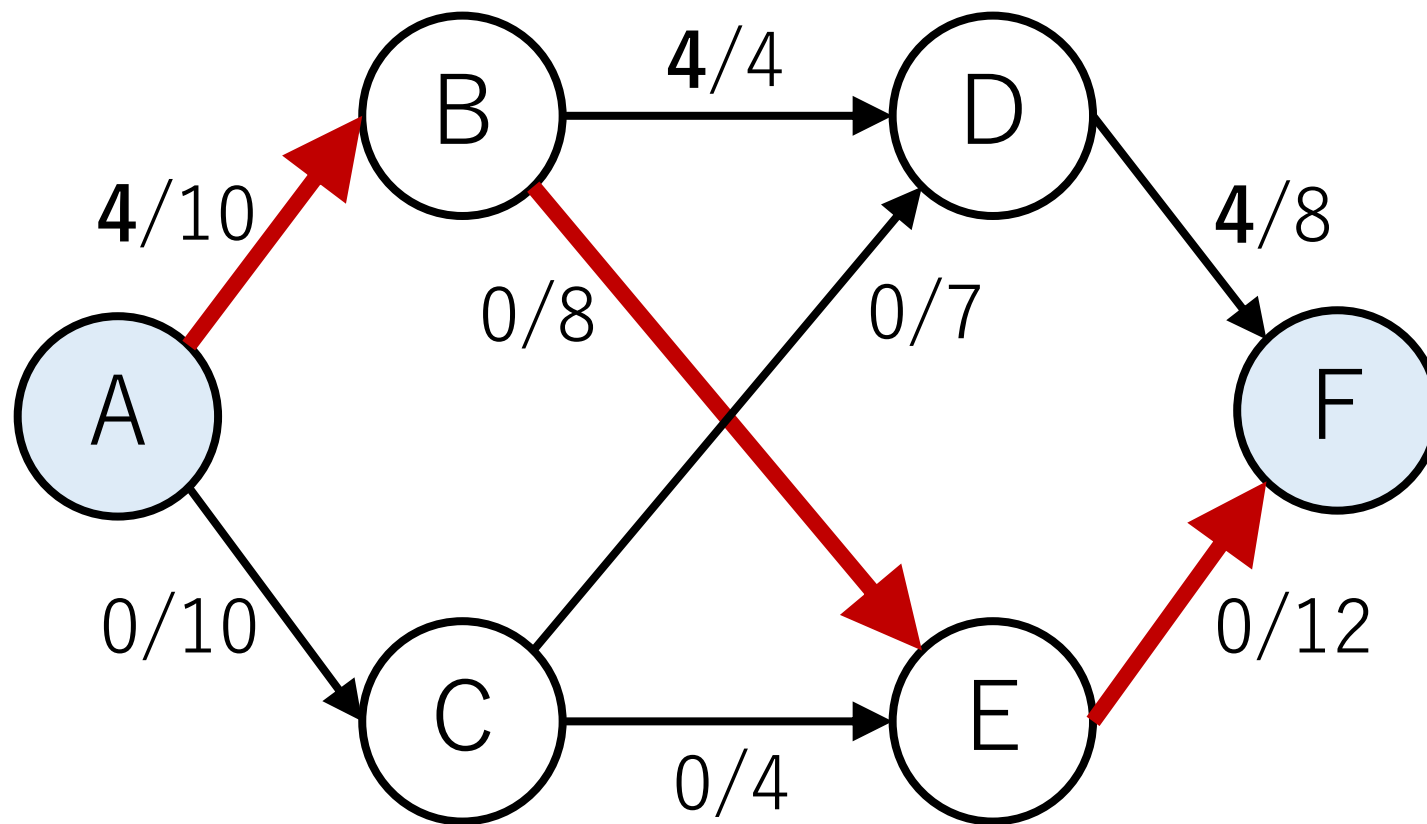
貪欲法による例

次にA->B->E->Fが見つかる。



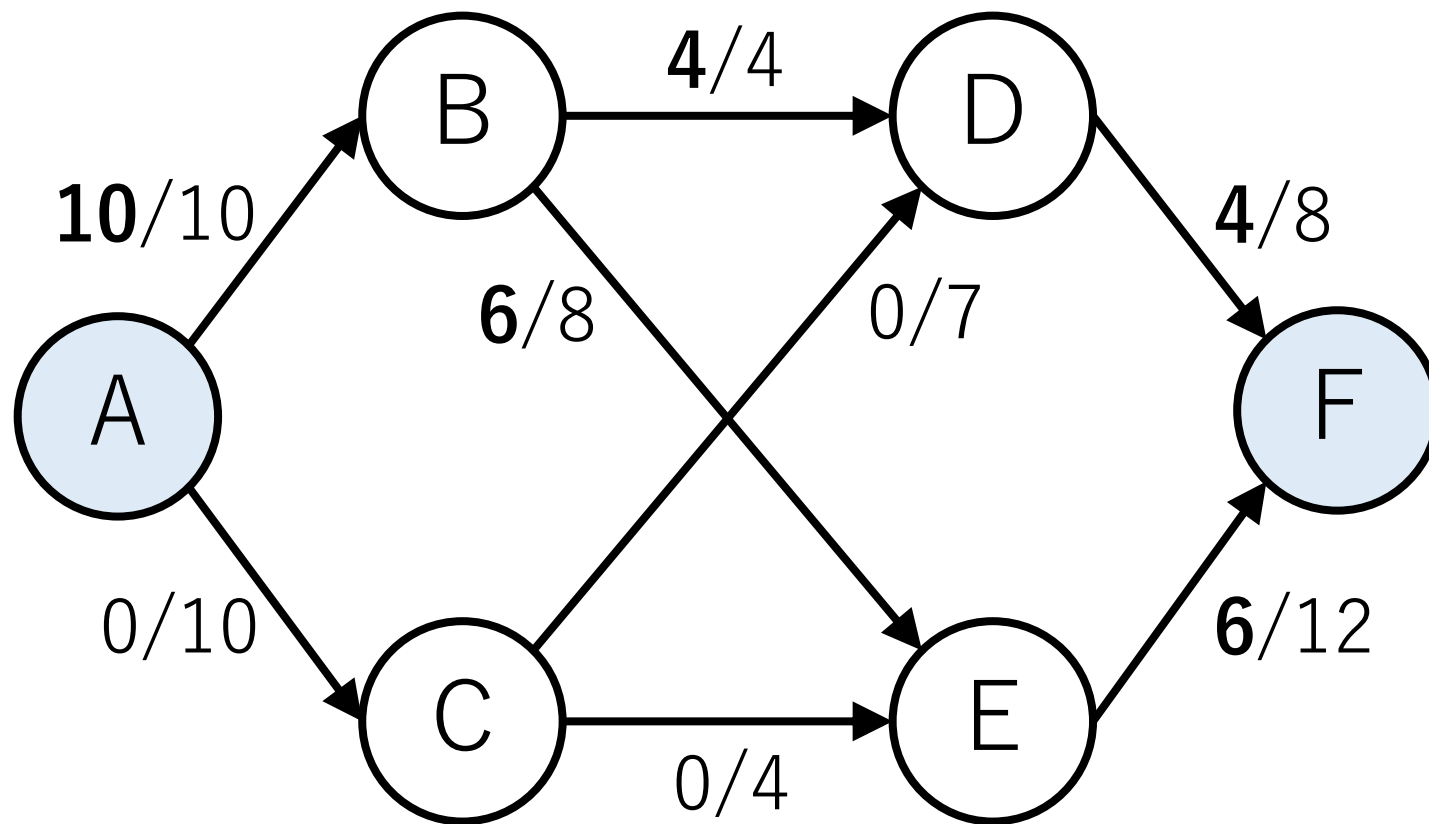
貪欲法による例

B->E->Fは本来8流せるが、A->Bの残りは6.



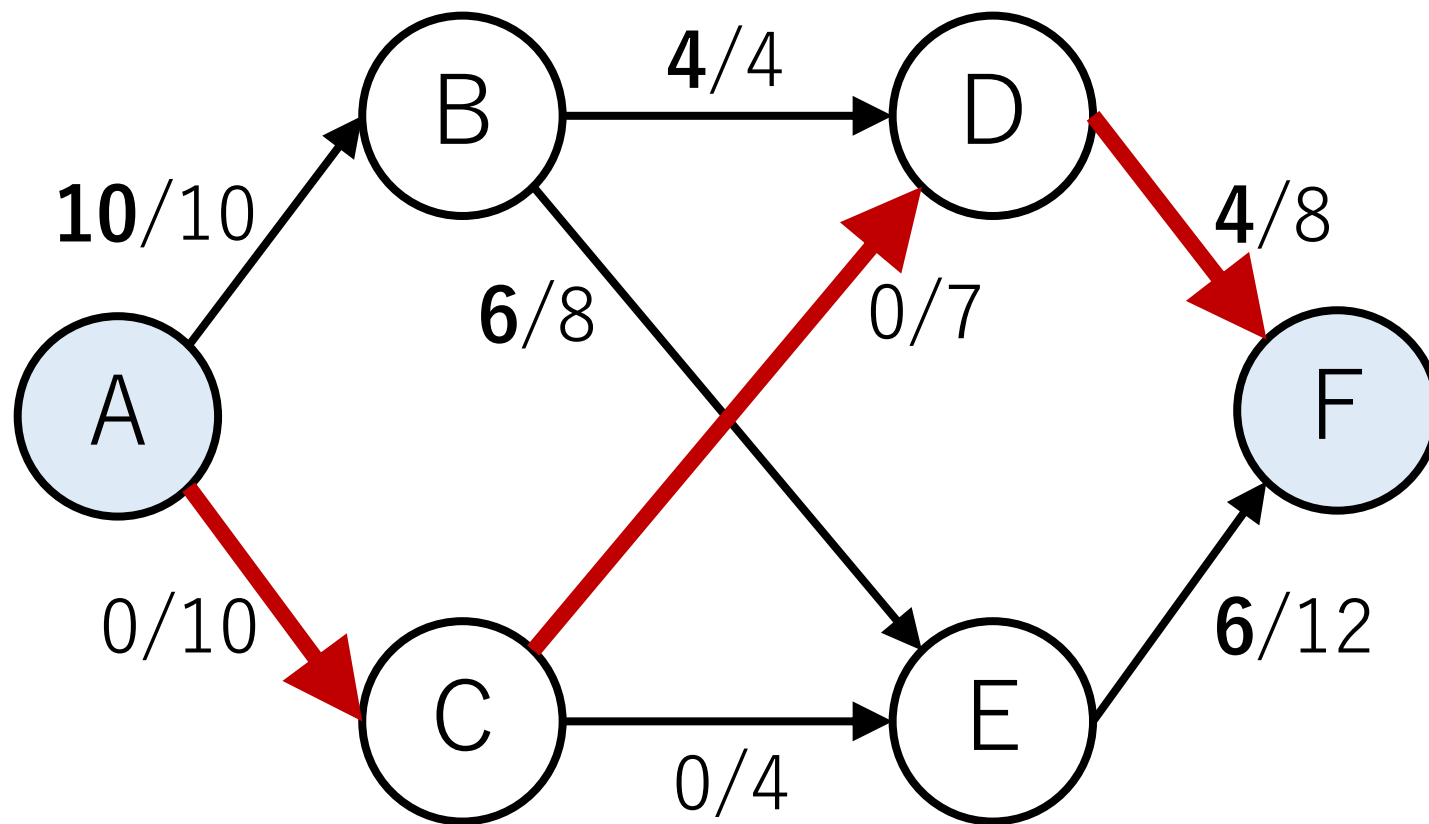
貪欲法による例

A->B->E->Fに6流す。



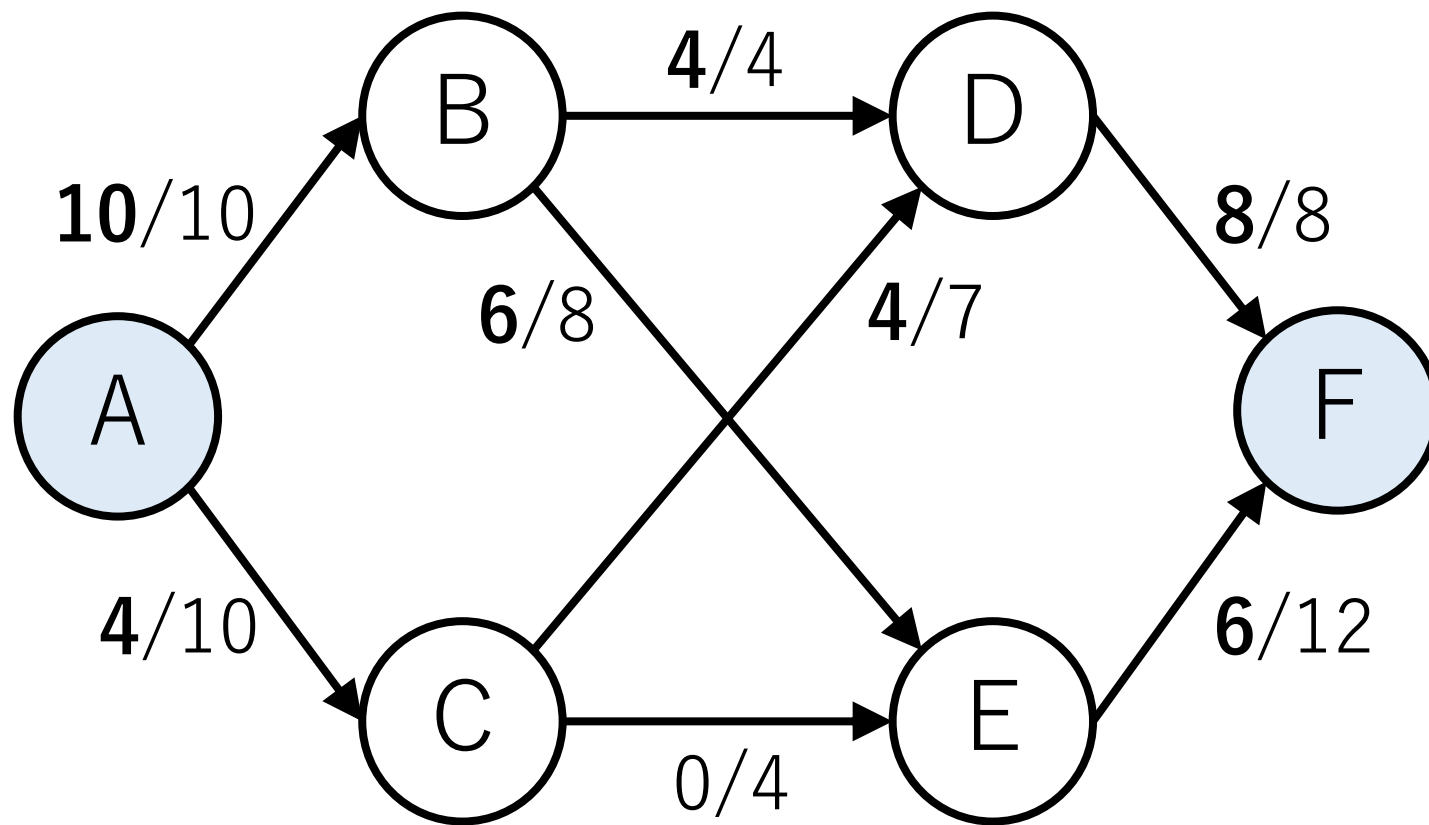
貪欲法による例

次に, $A \rightarrow C \rightarrow D \rightarrow F$ を見つける. ここに流せるのは4.



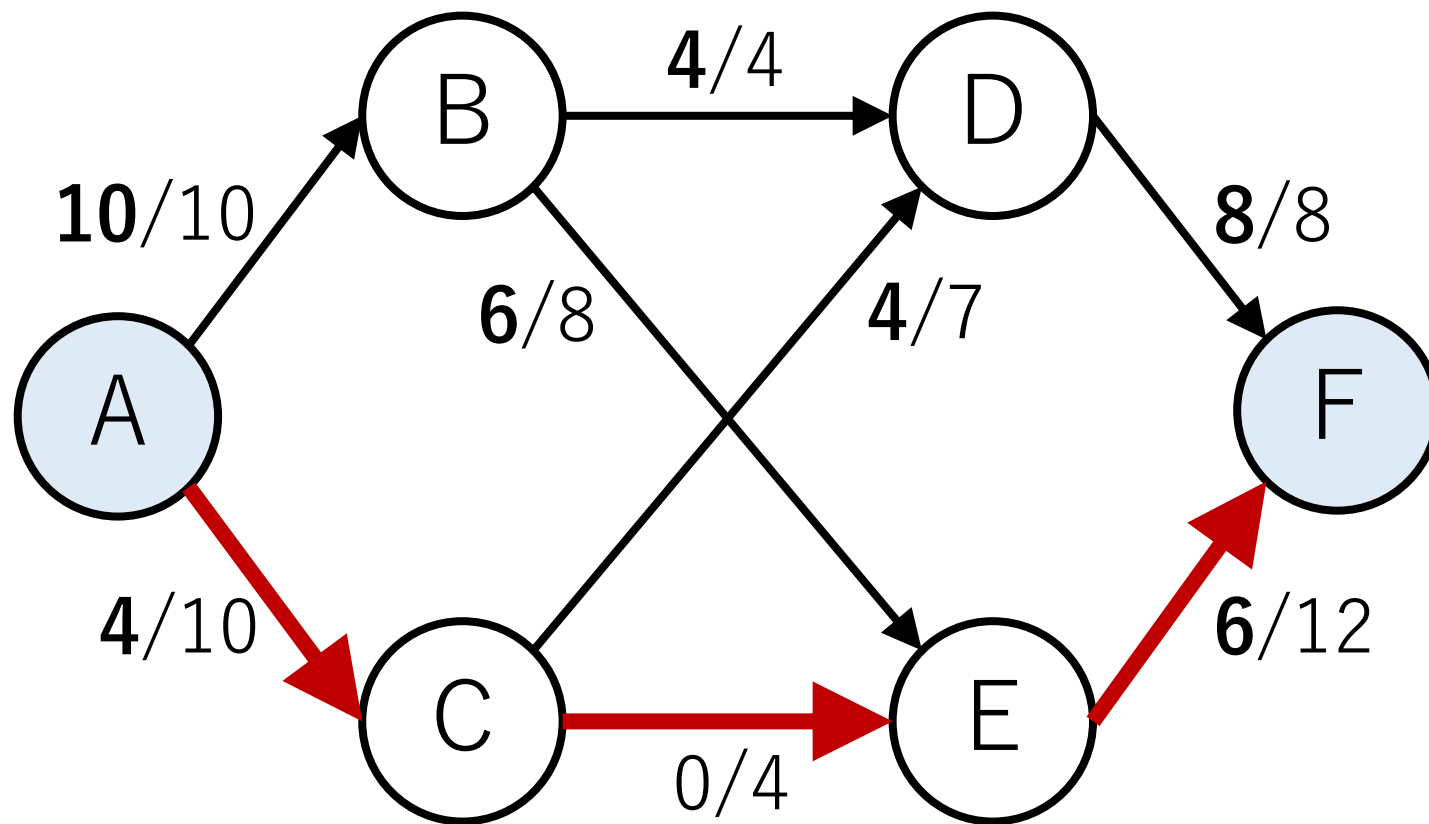
貪欲法による例

A->C->D->Fに4流す。



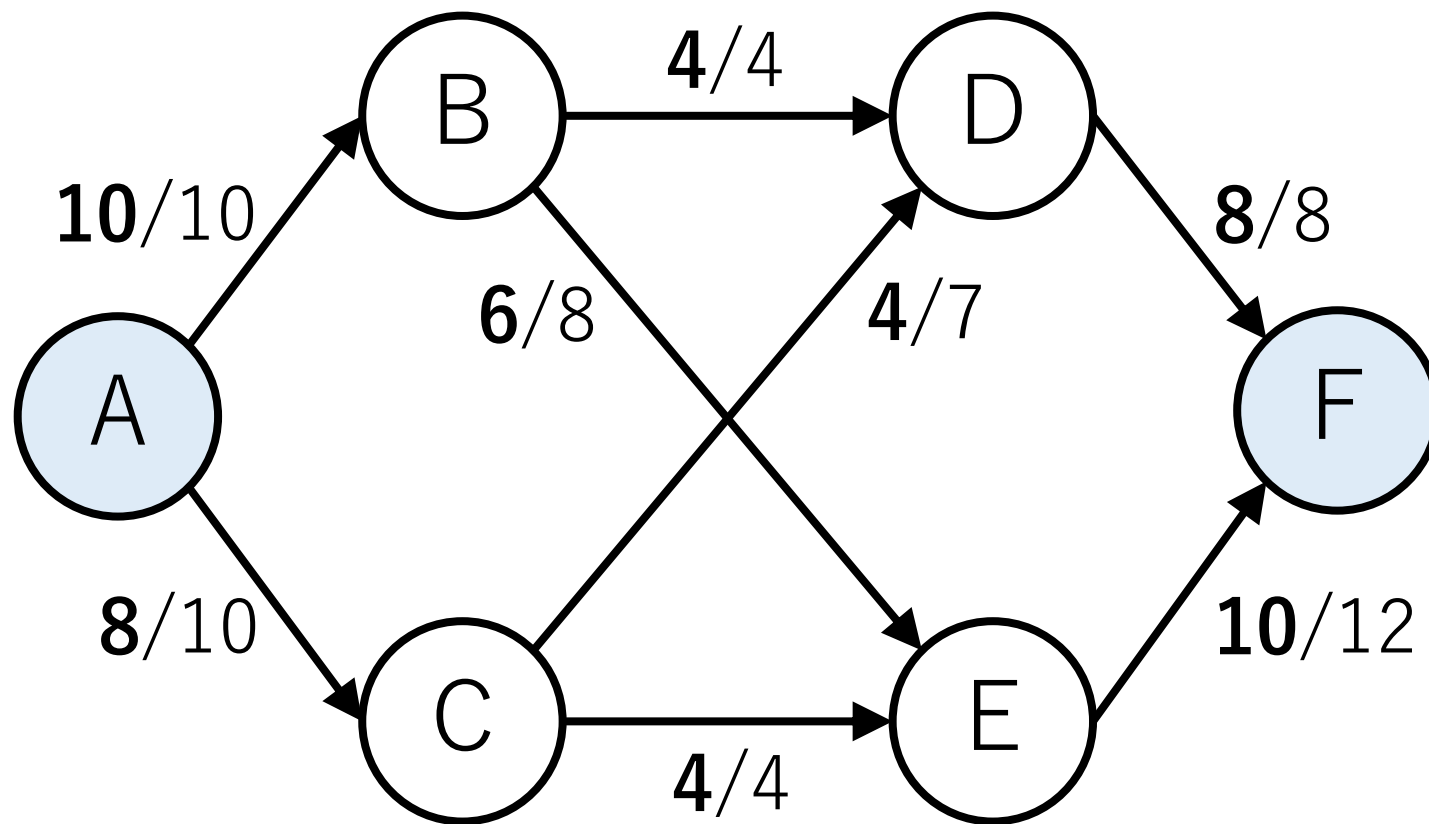
貪欲法による例

最後に， $A \rightarrow C \rightarrow E \rightarrow F$ を見つける．ここには4流せる．



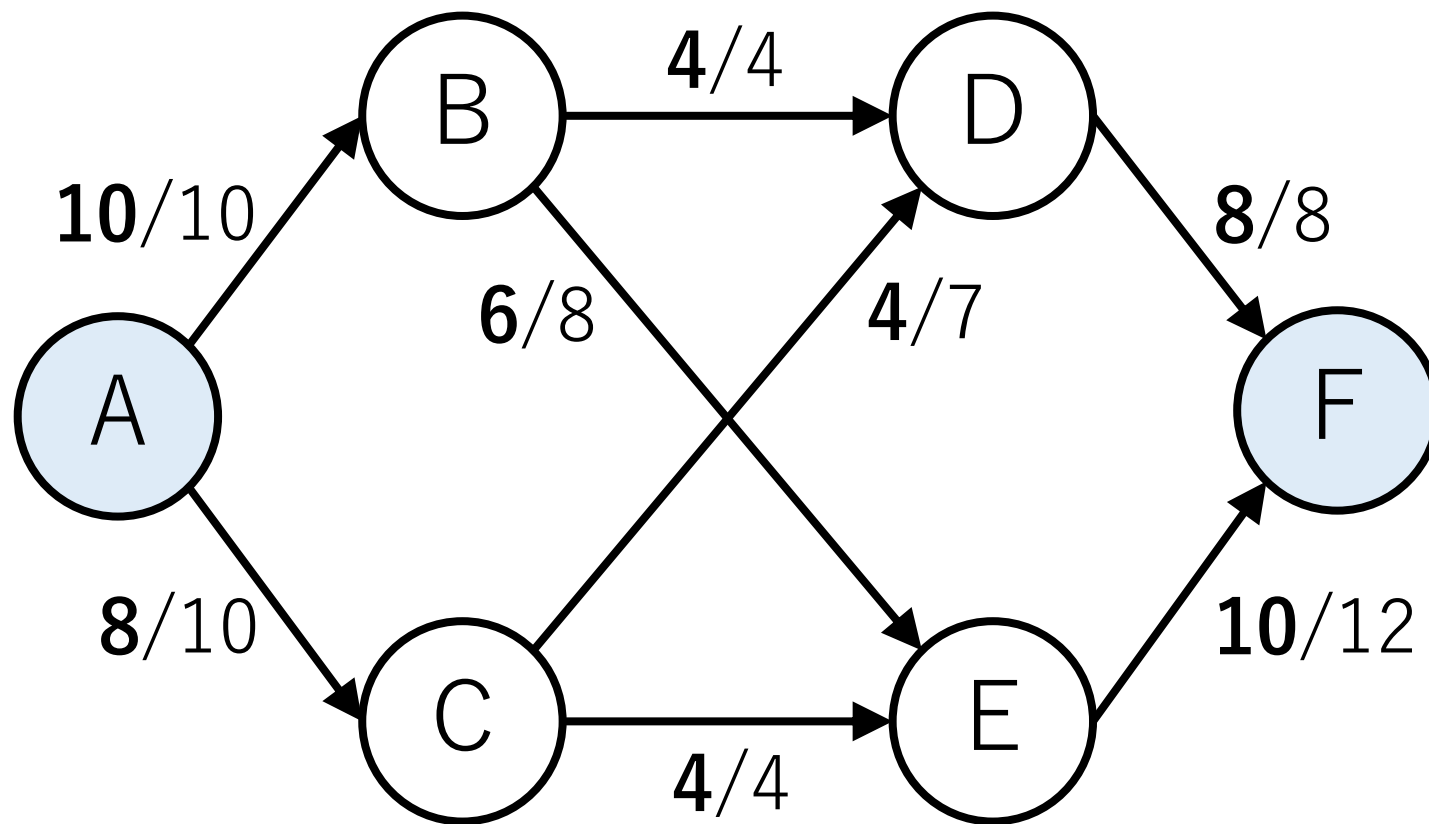
貪欲法による例

A->C->D->Fに4流す。



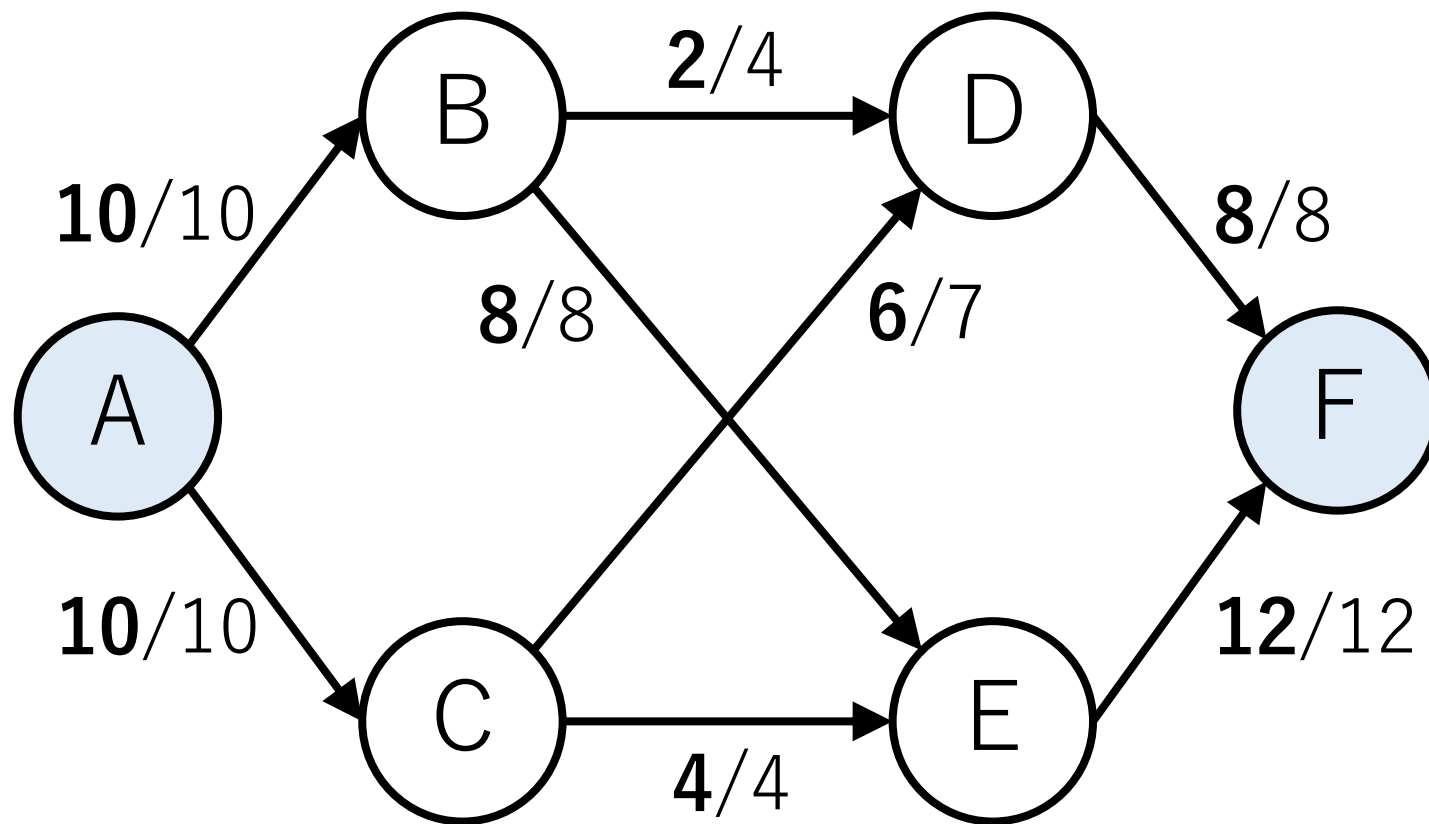
貪欲法による例

よって、最大流は18？



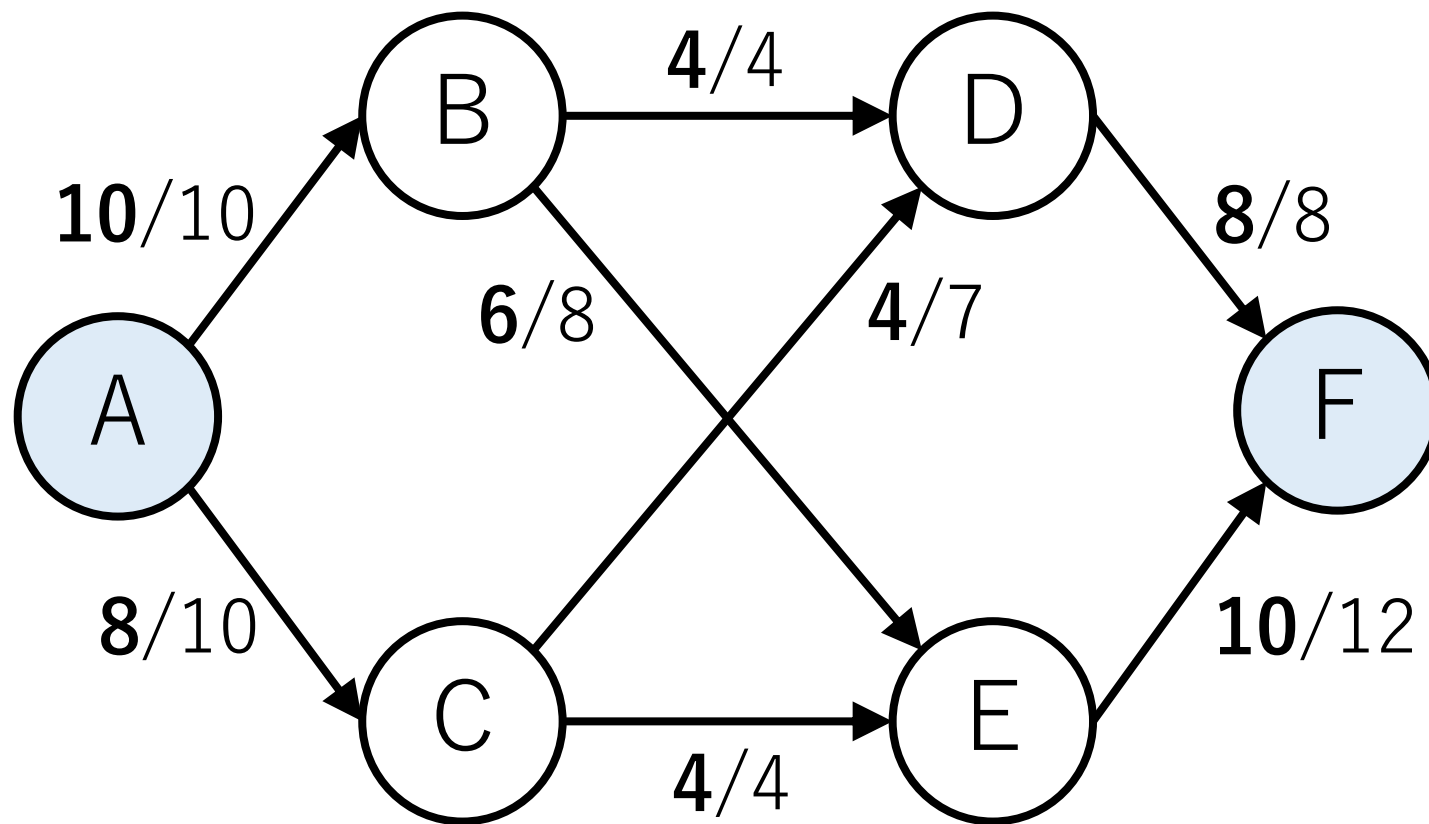
実際には. . .

最適解は20.



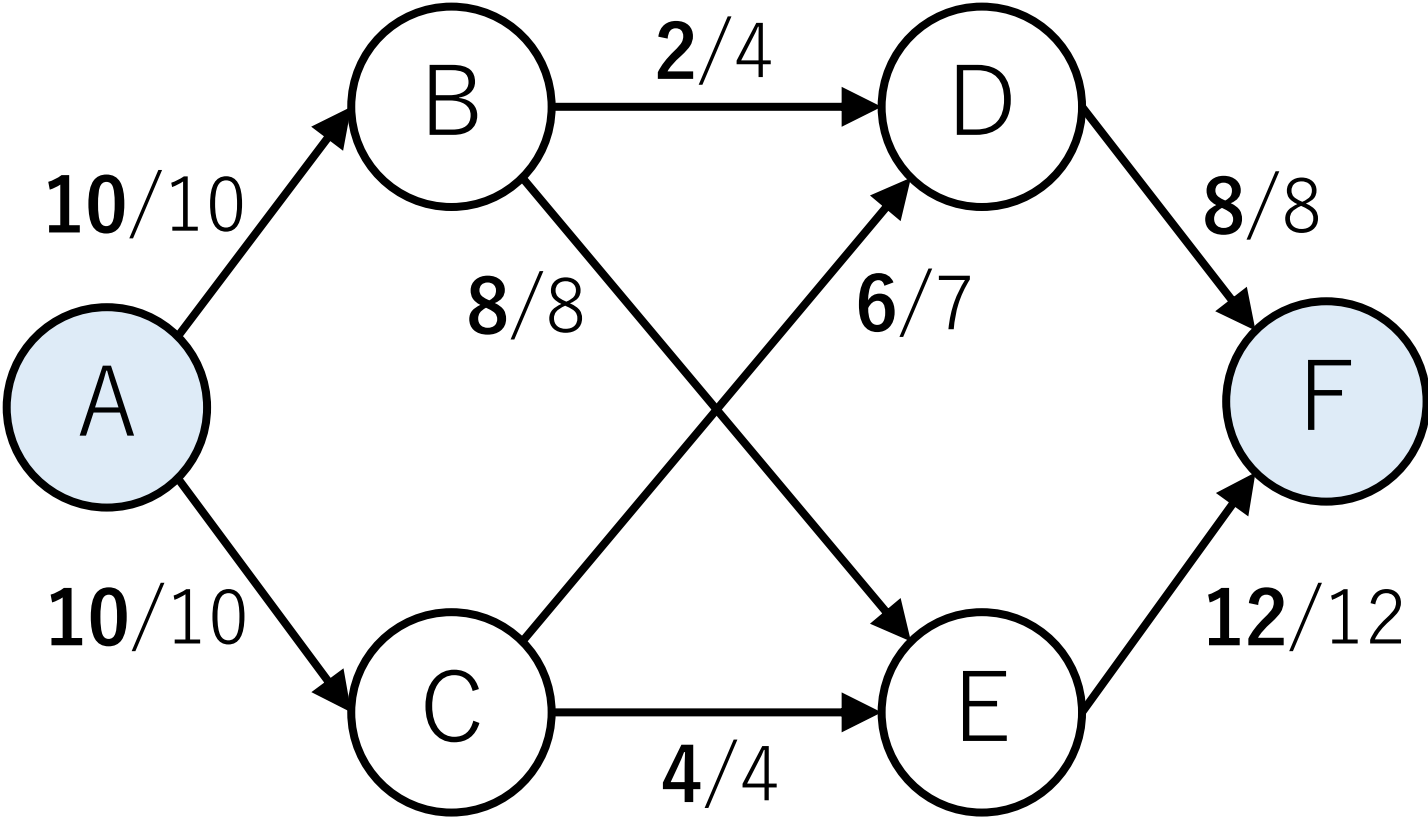
比較してみよう

貪欲法の場合



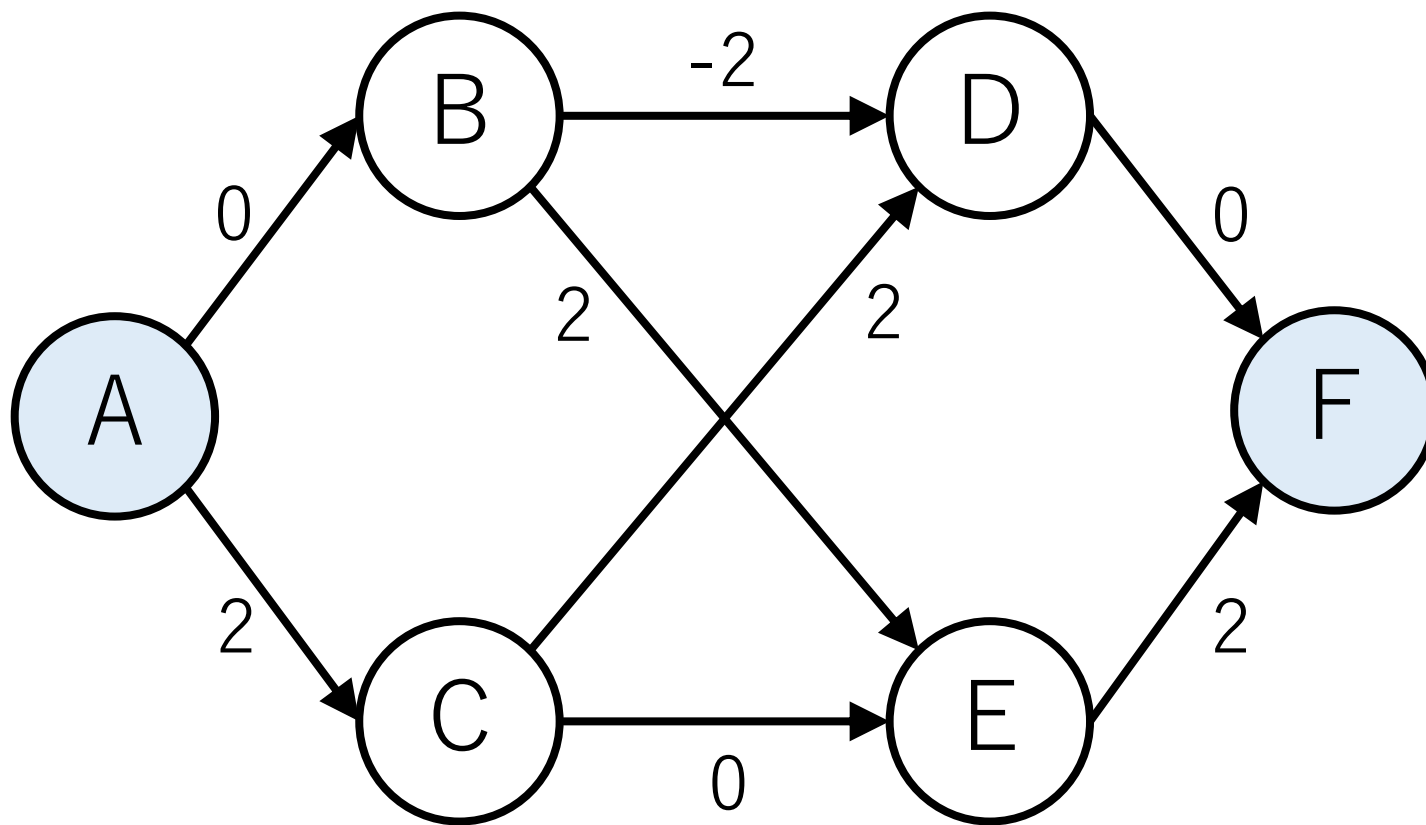
比較してみよう

最適解の場合



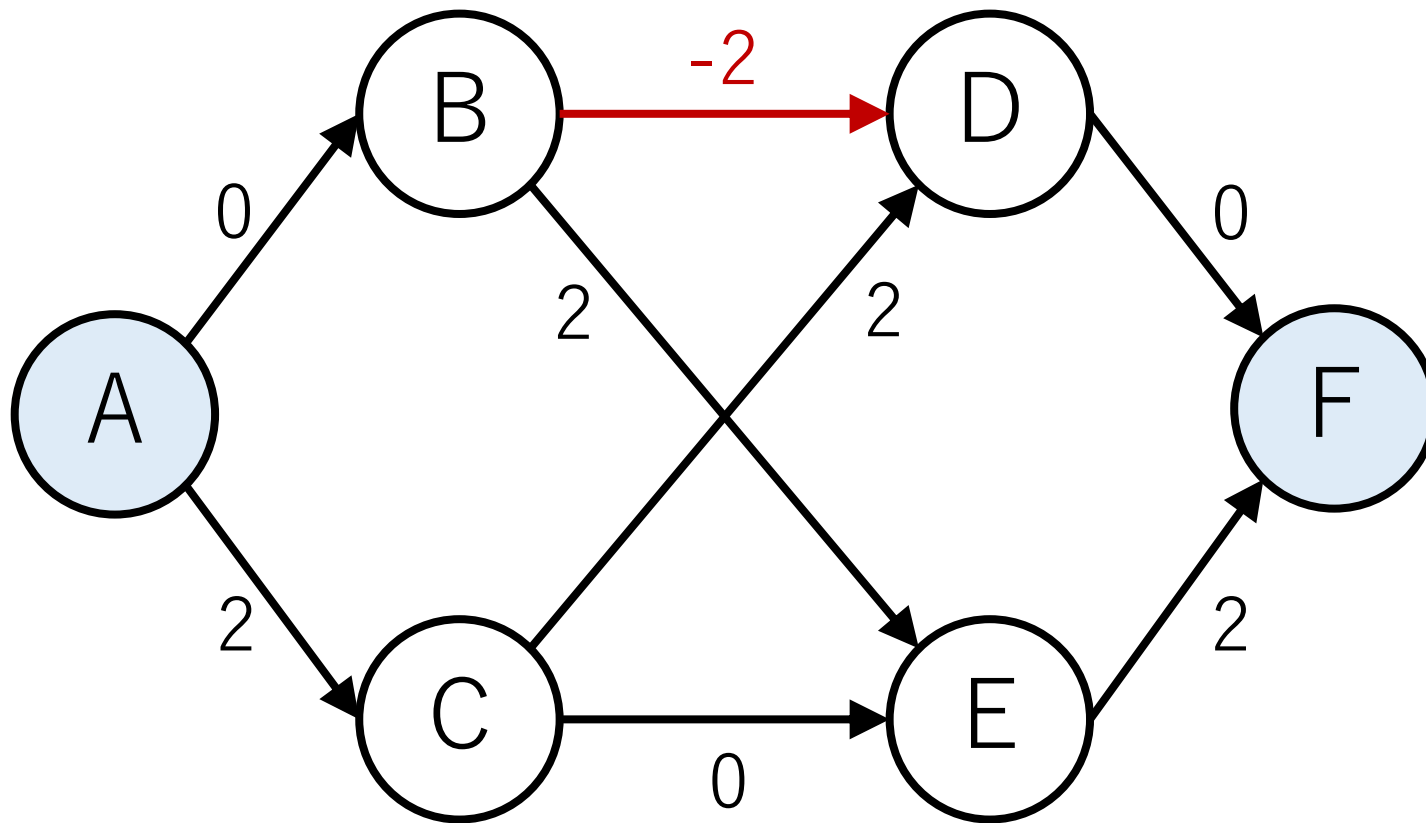
差をとってみよう。

最大流の場合 - 貪欲法の場合



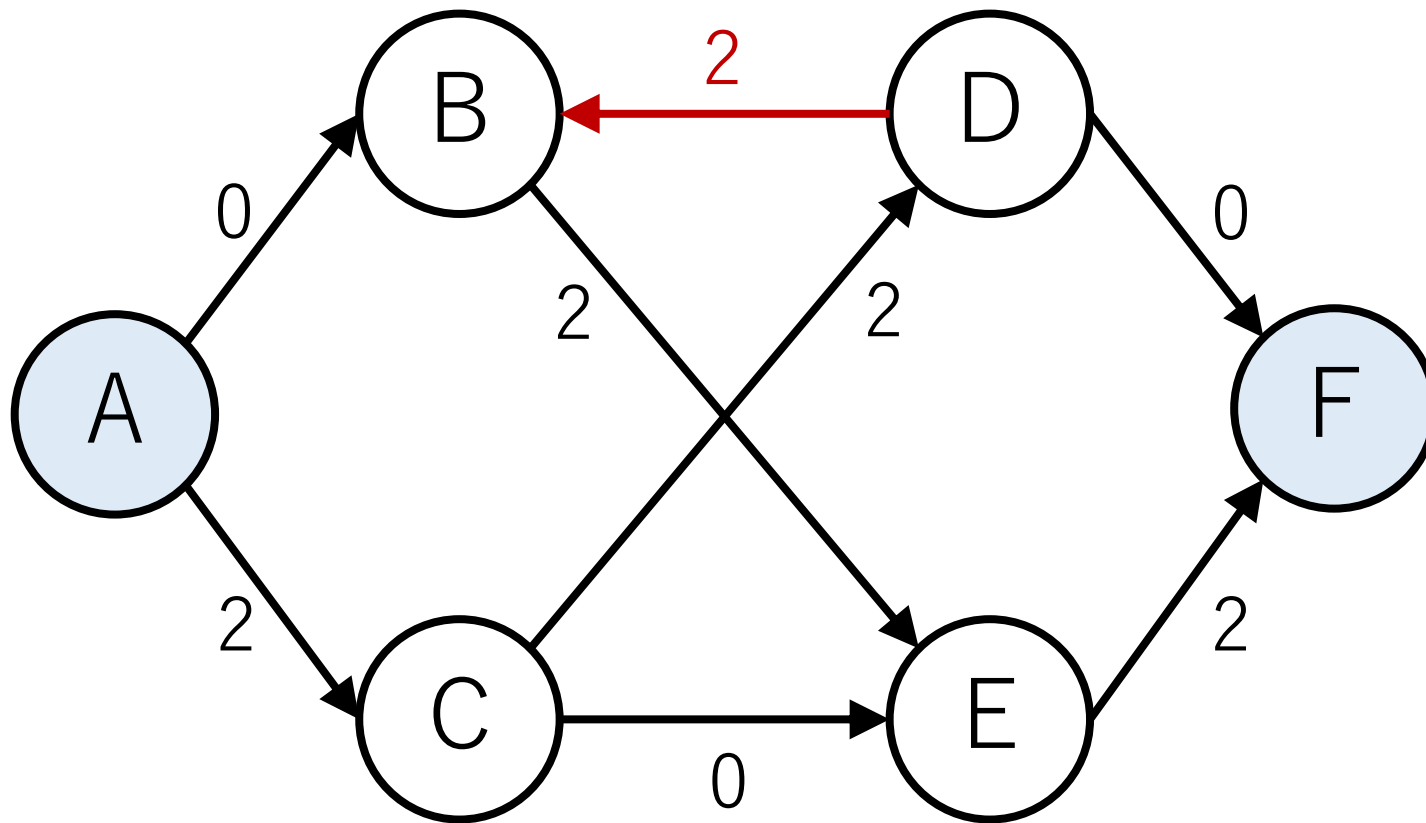
差をとってみよう。

負の流れになる経路がある！



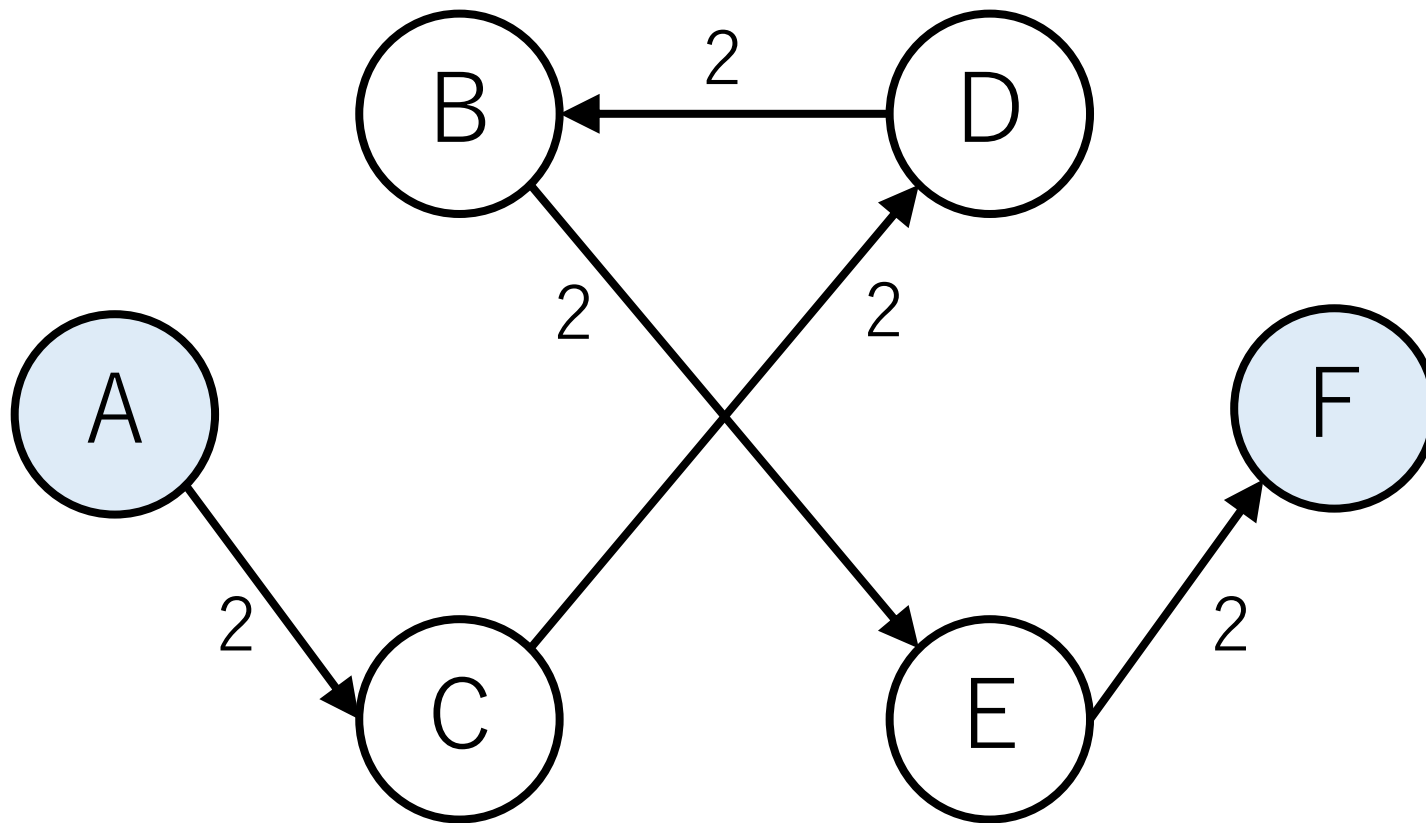
差をとってみよう。

負の流れになる経路がある！



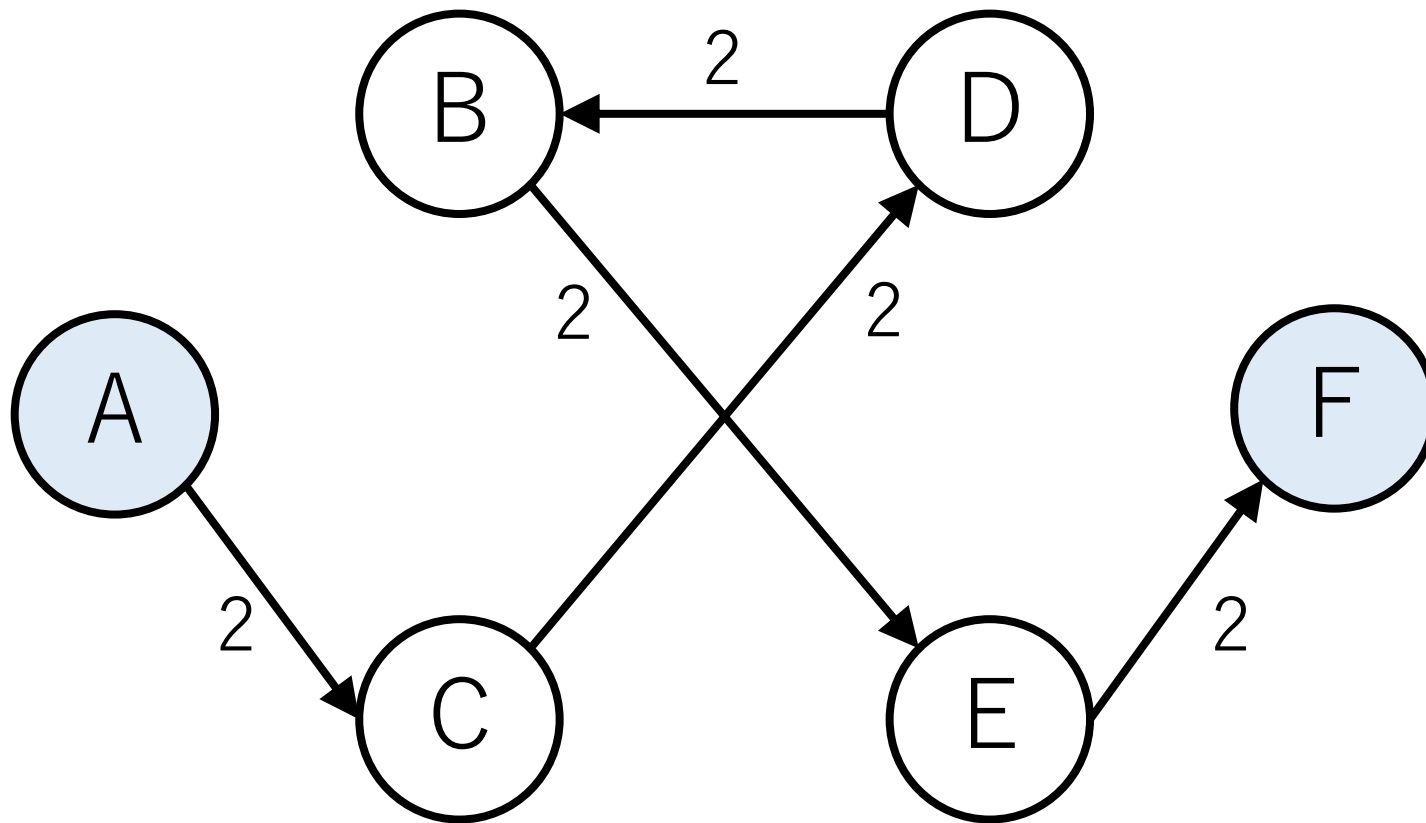
差をとってみよう。

貪欲法では考慮しなかった経路が浮き出る。



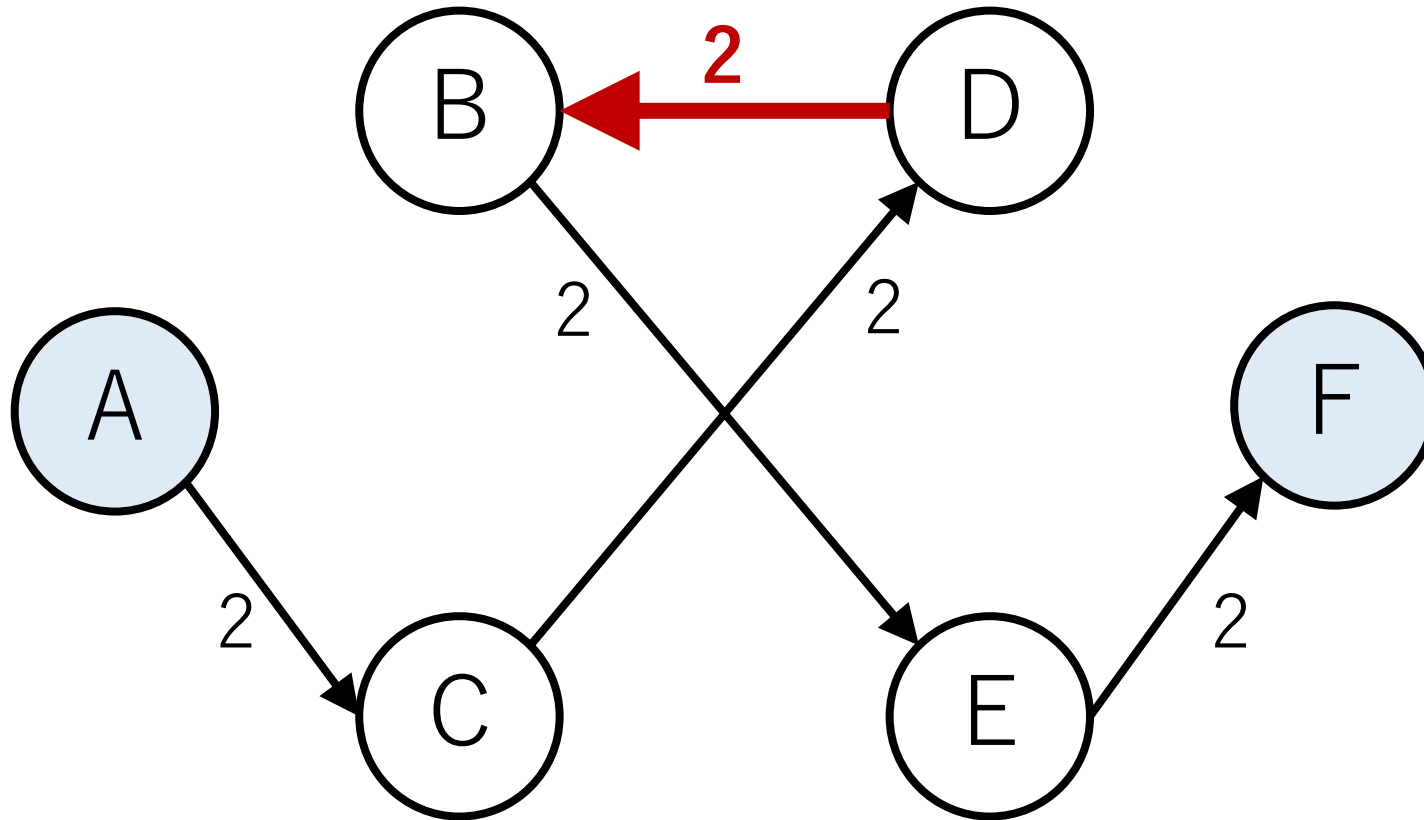
差をとってみよう。

貪欲法 + A->C->D->B->E->Fを考えられれば, 20になる!



アルゴリズムの道筋

逆方向に流れる経路も考えることはできないか？



フォード・ファルカーソン法

Ford-Fulkerson

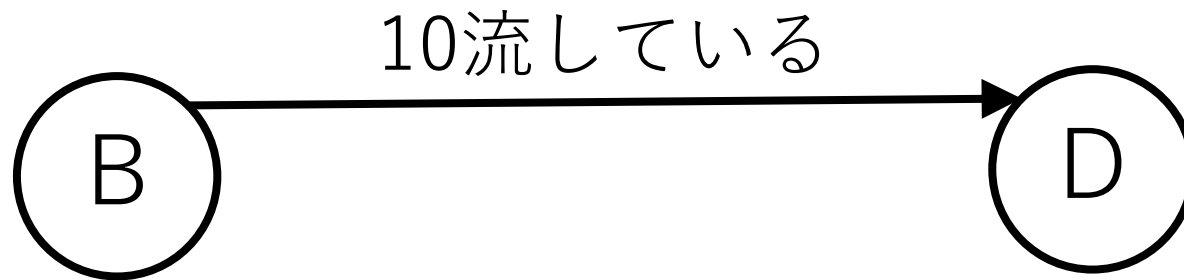
基本は貪欲法と同じく，DFSやBFSで開始ノードから終了ノードまでの経路を探索し，流せるだけ流す。

ただし，経路の探索において，ノード間を逆にたどるような仮想的な経路も考慮する。

「ノード間を逆にたどる」経路の容量は，順方向の今までの流量と同じとする。

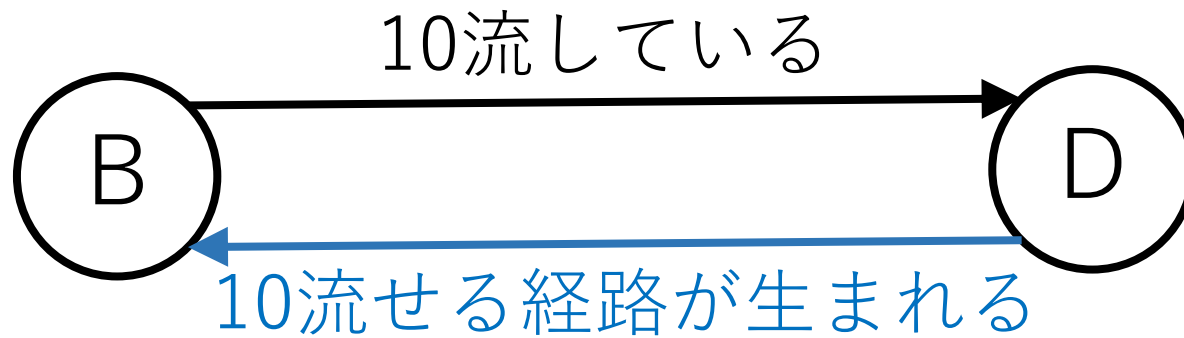
フォード・ファルカーソン法

逆方向の容量 = 順方向の今までの流量



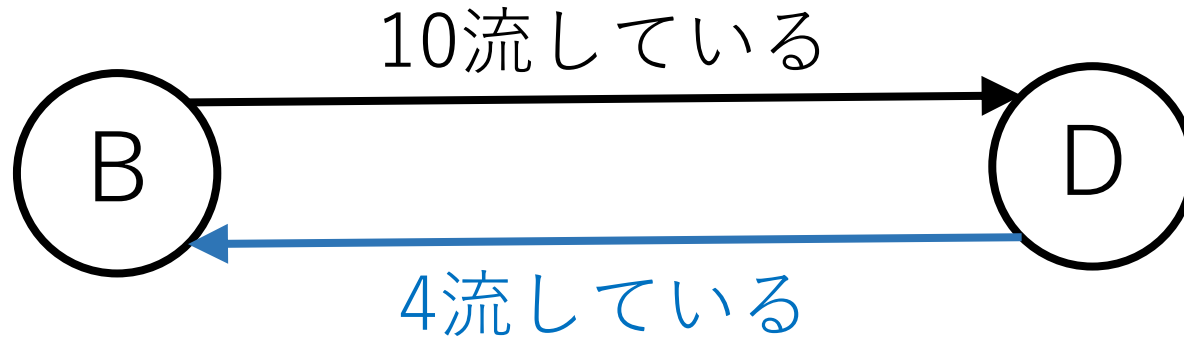
フォード・ファルカーソン法

逆方向の容量 = 順方向の今までの流量



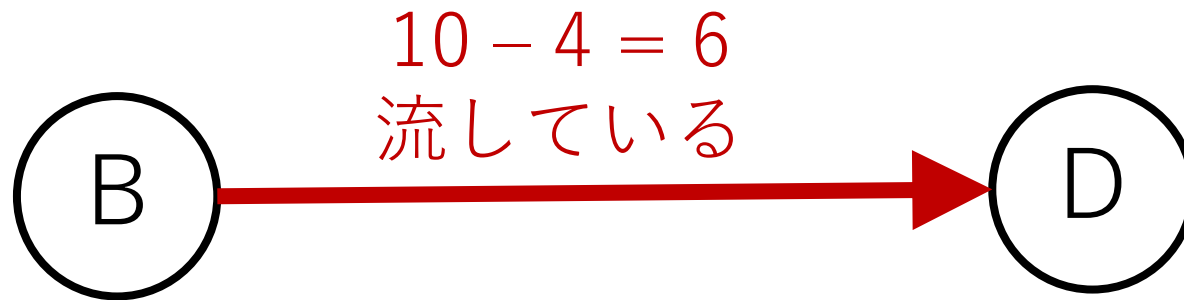
フォード・ファルカーソン法

もし、逆方向に4流すとすると？



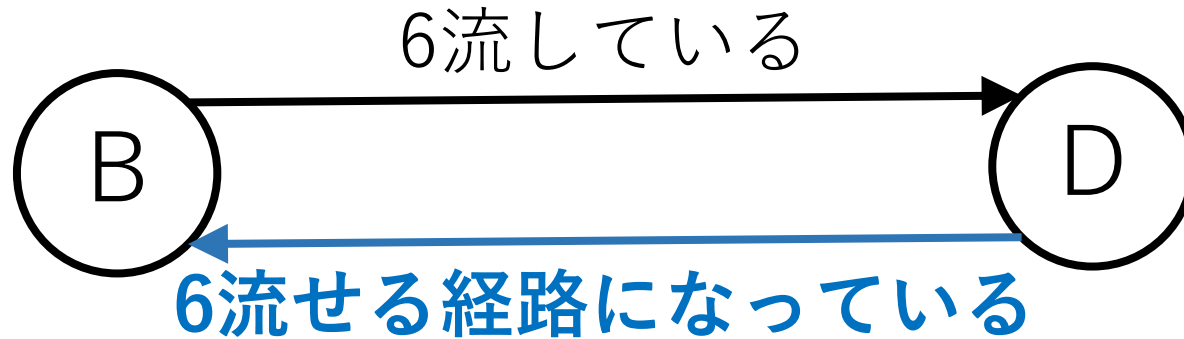
フォード・ファルカーソン法

2つを合わせると, B->Dに6流していることと同値.



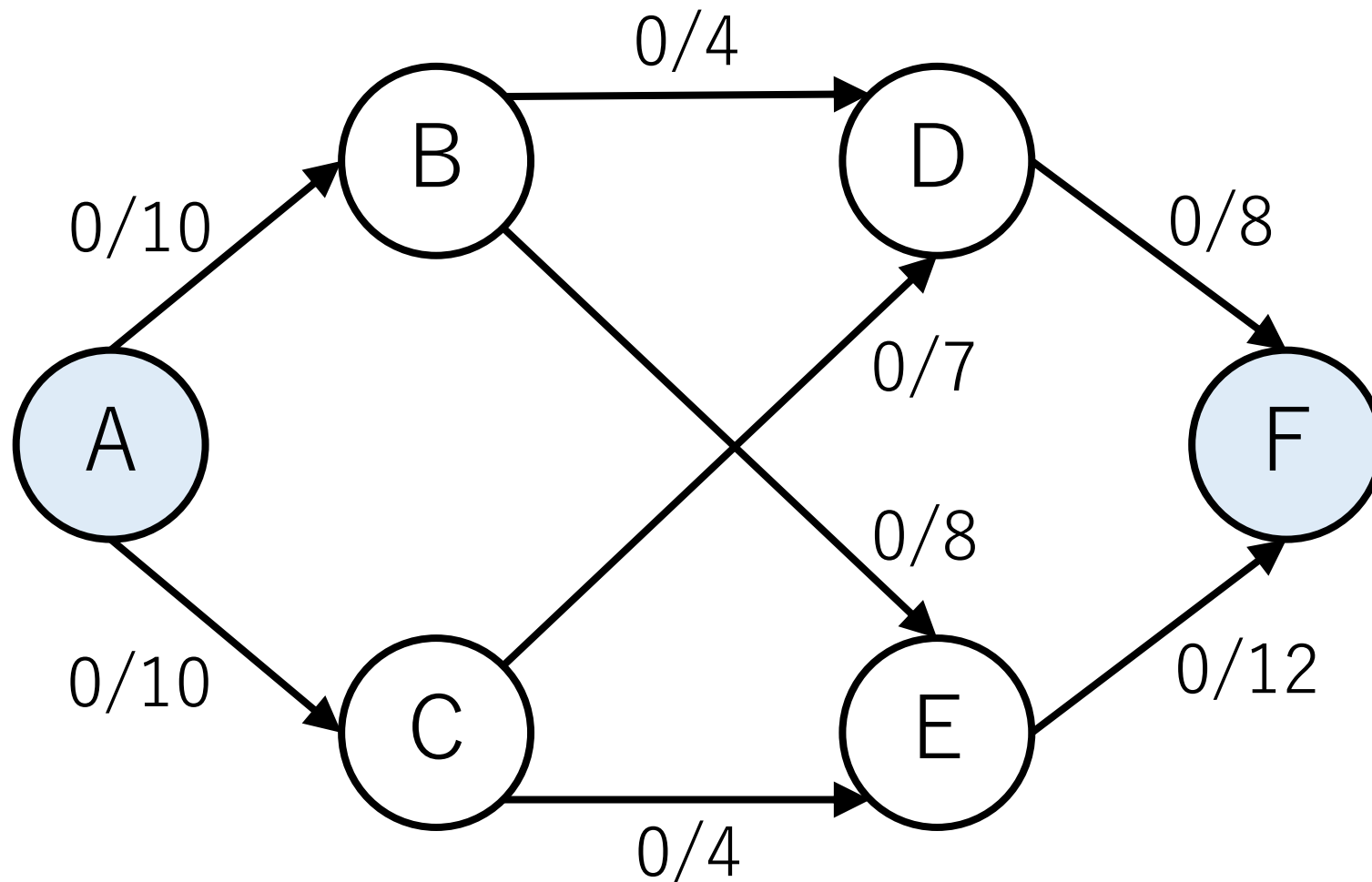
フォード・ファルカーソン法

逆方向の容量も更新する。



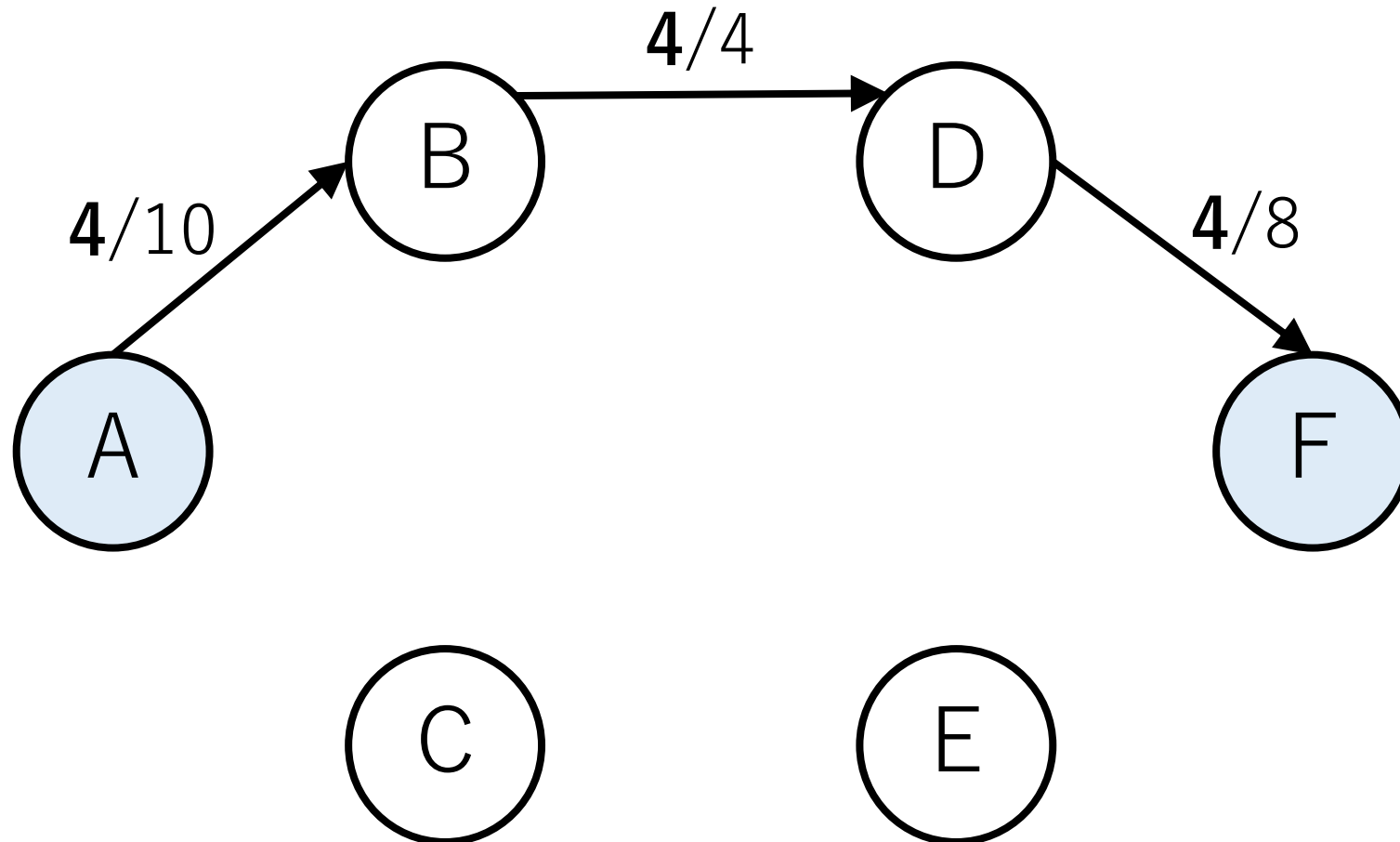
フォード・ファルカーソン法による例

先程の例で見てみよう。



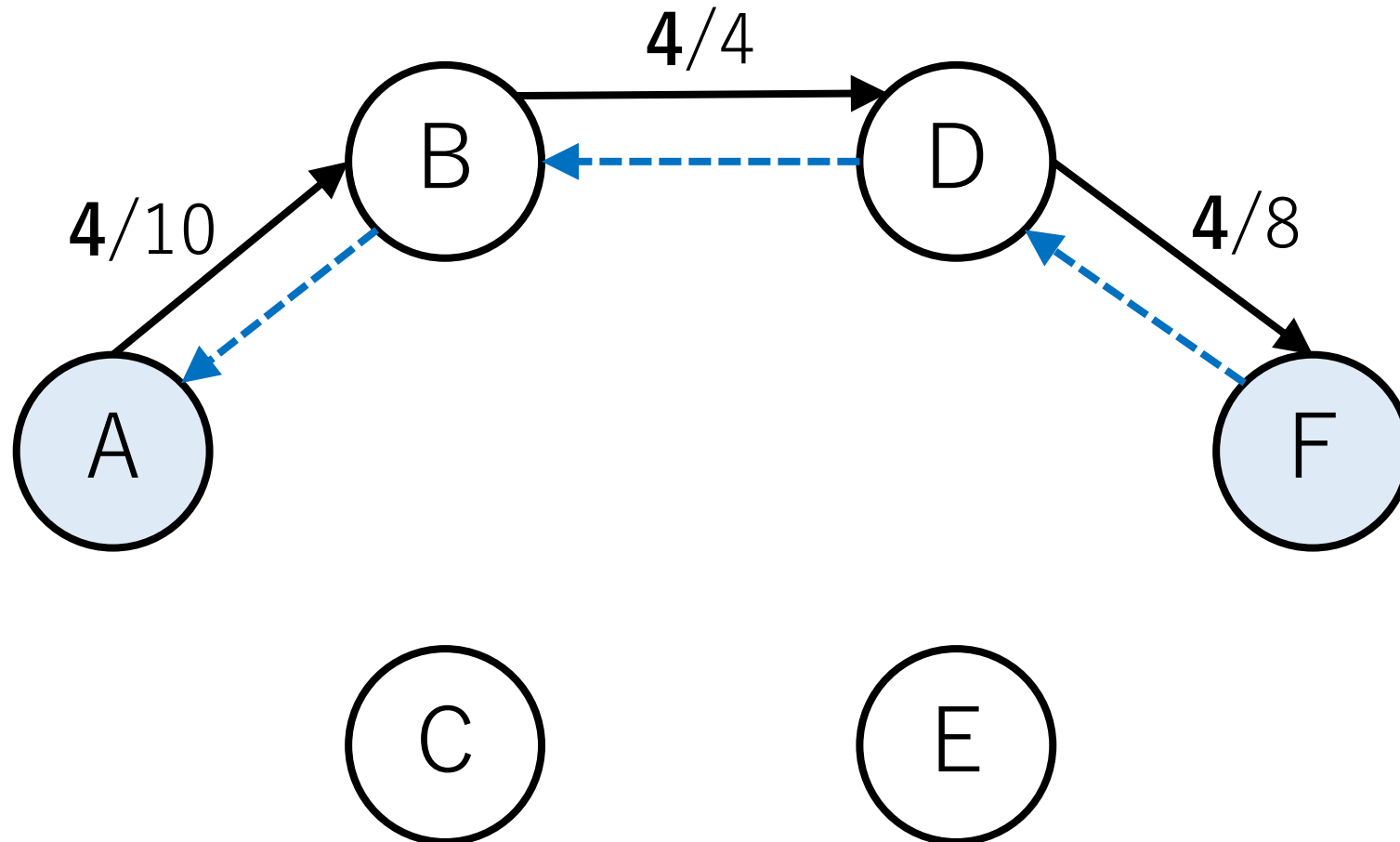
フォード・ファルカーソン法による例

A->B->D->Fを見つけて、4流す。



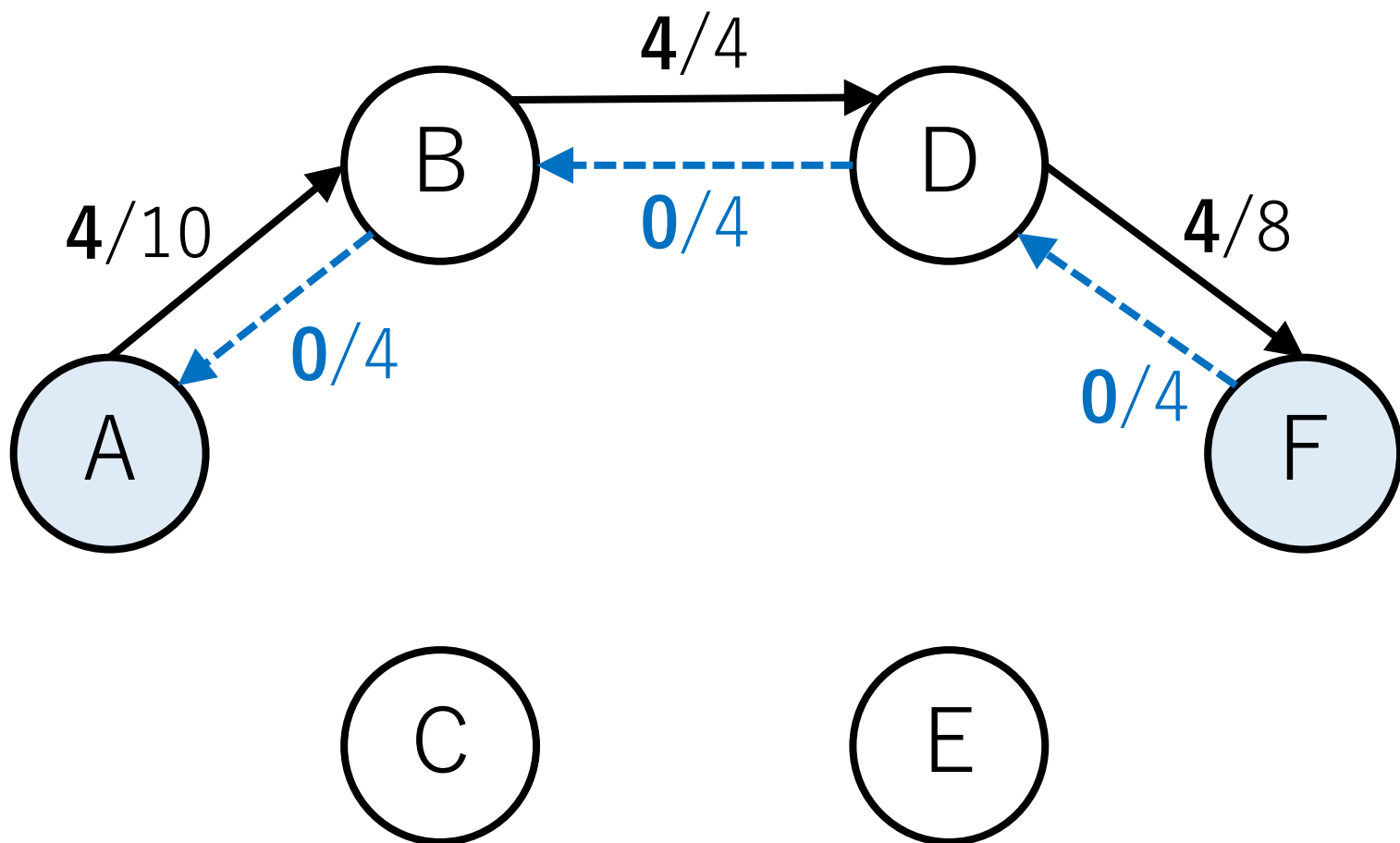
フォード・ファルカーソン法による例

逆方向の経路を設定.



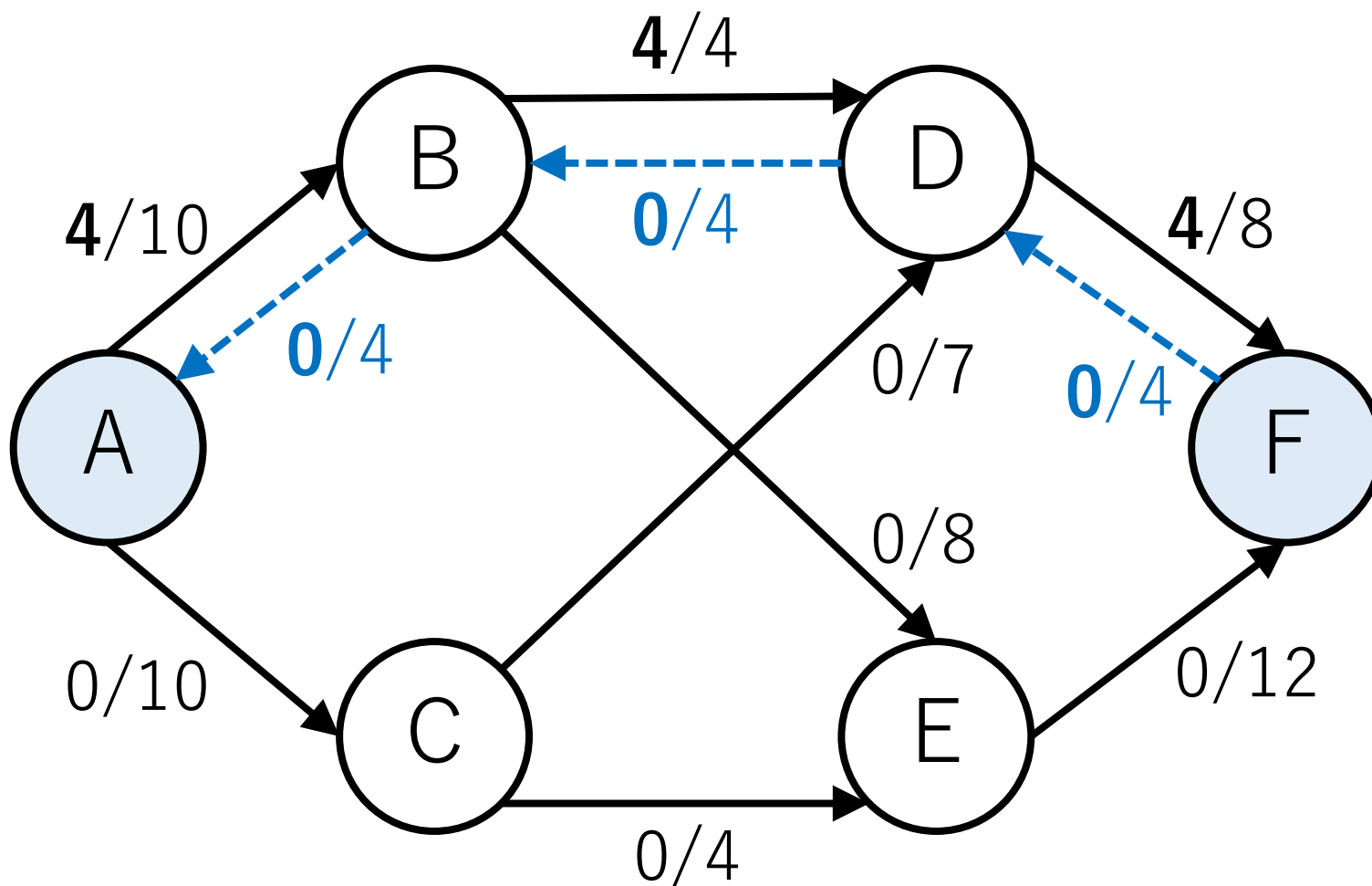
フォード・ファルカーソン法による例

「逆方向の容量 = 順方向の今までの流量」で容量を更新.



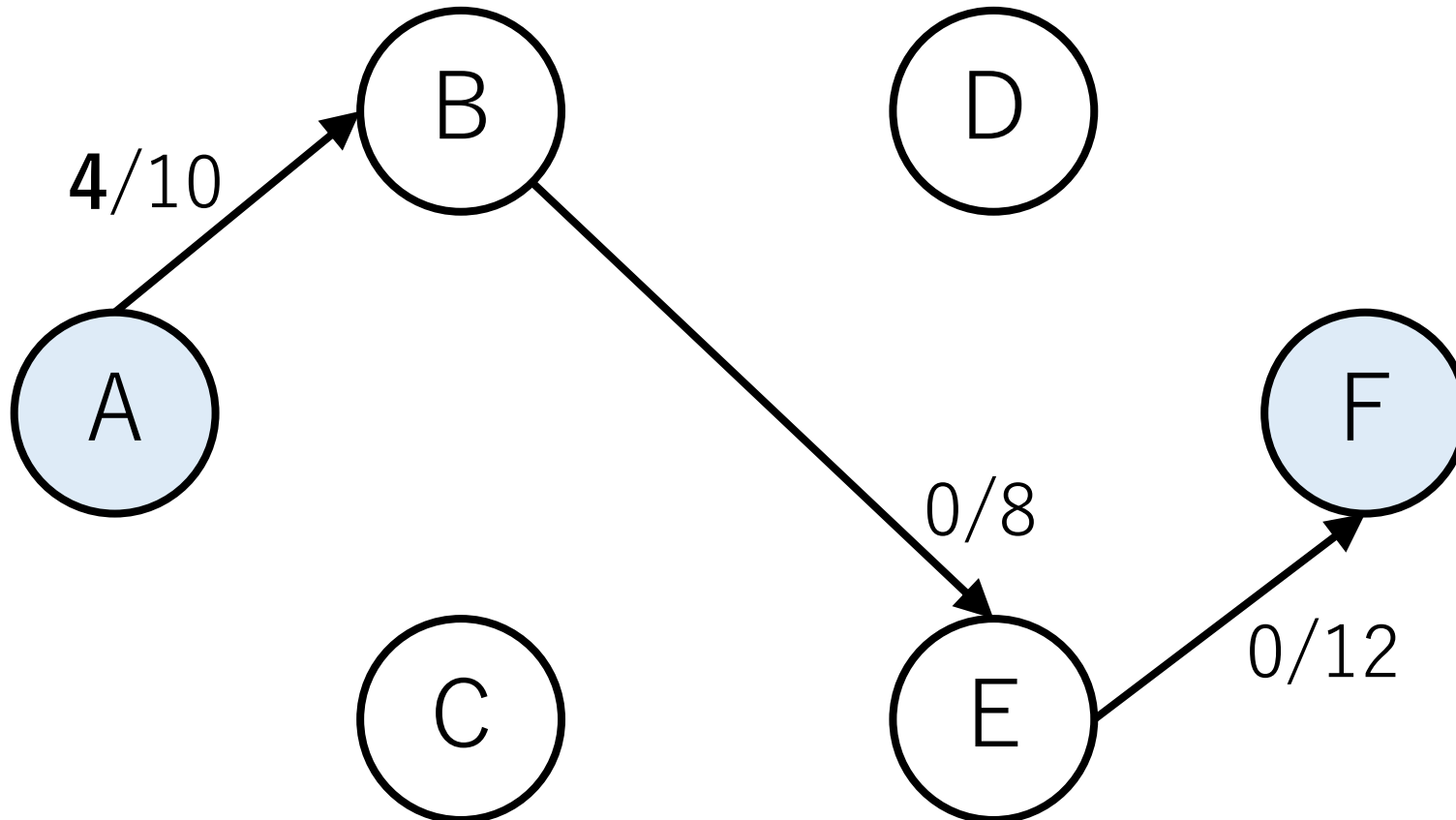
フォード・ファルカーソン法による例

現在の状態.



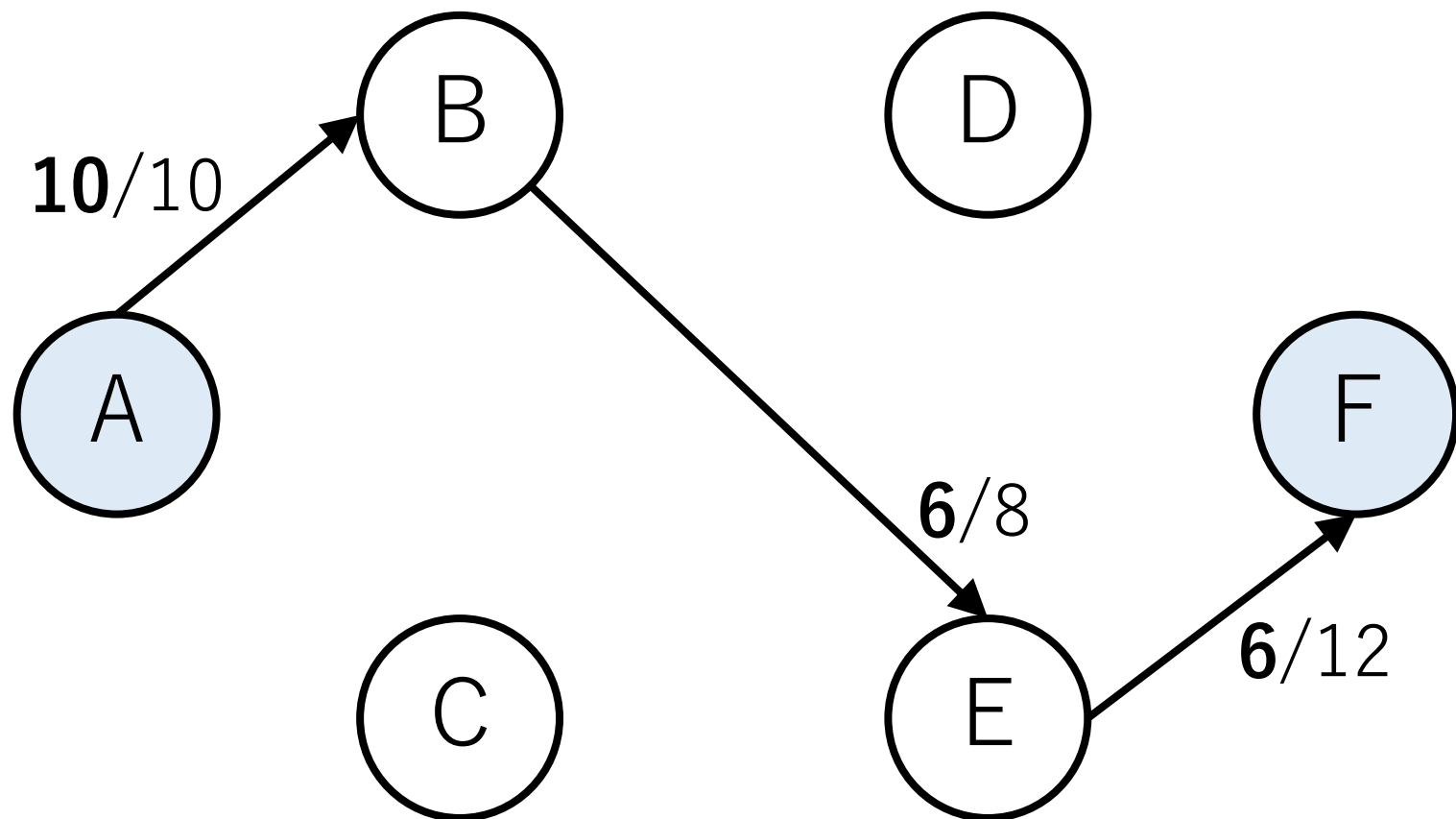
フォード・ファルカーソン法による例

A->B->E->Fを見つける。流せる量は6。



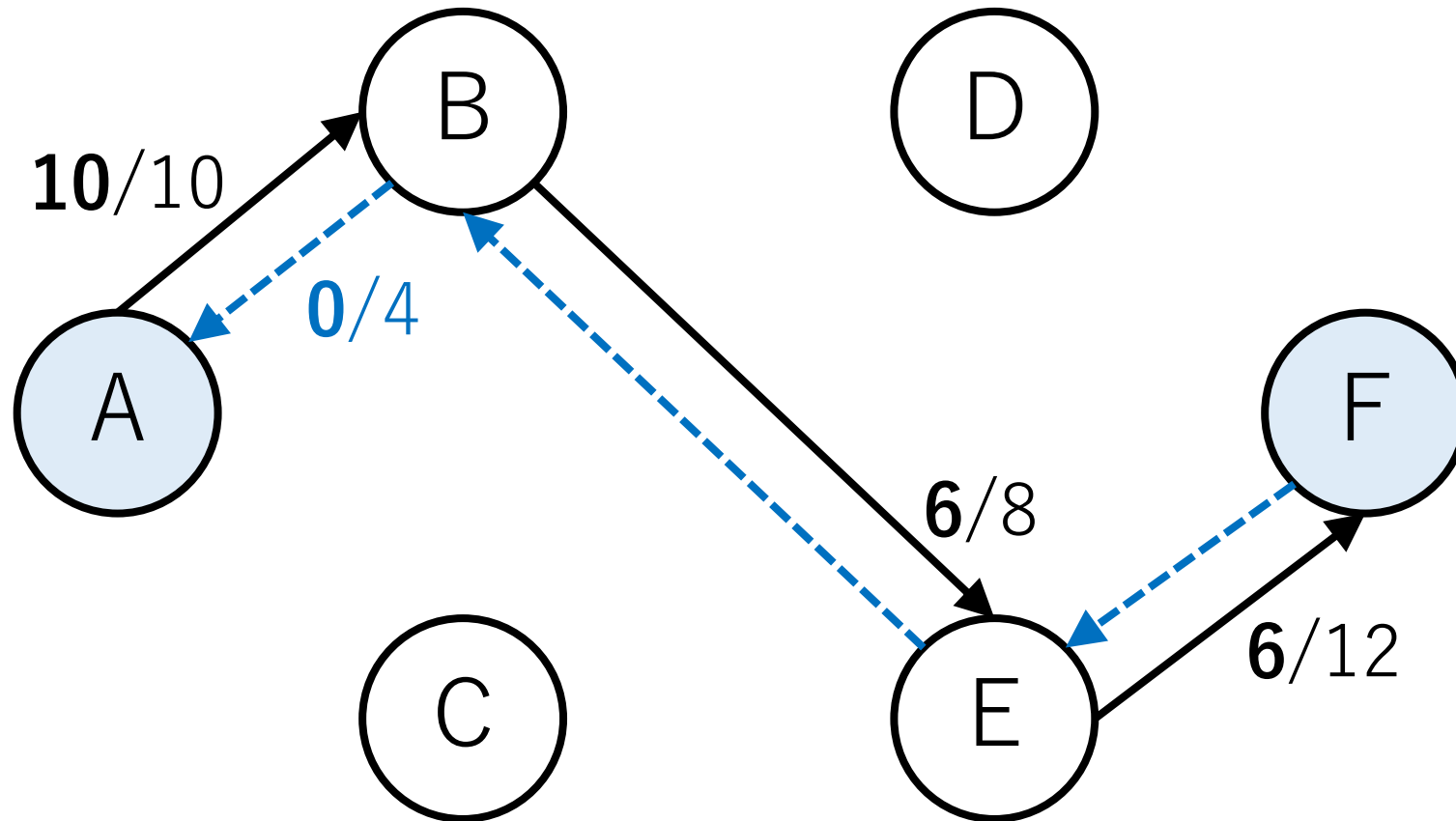
フォード・ファルカーソン法による例

6流す。



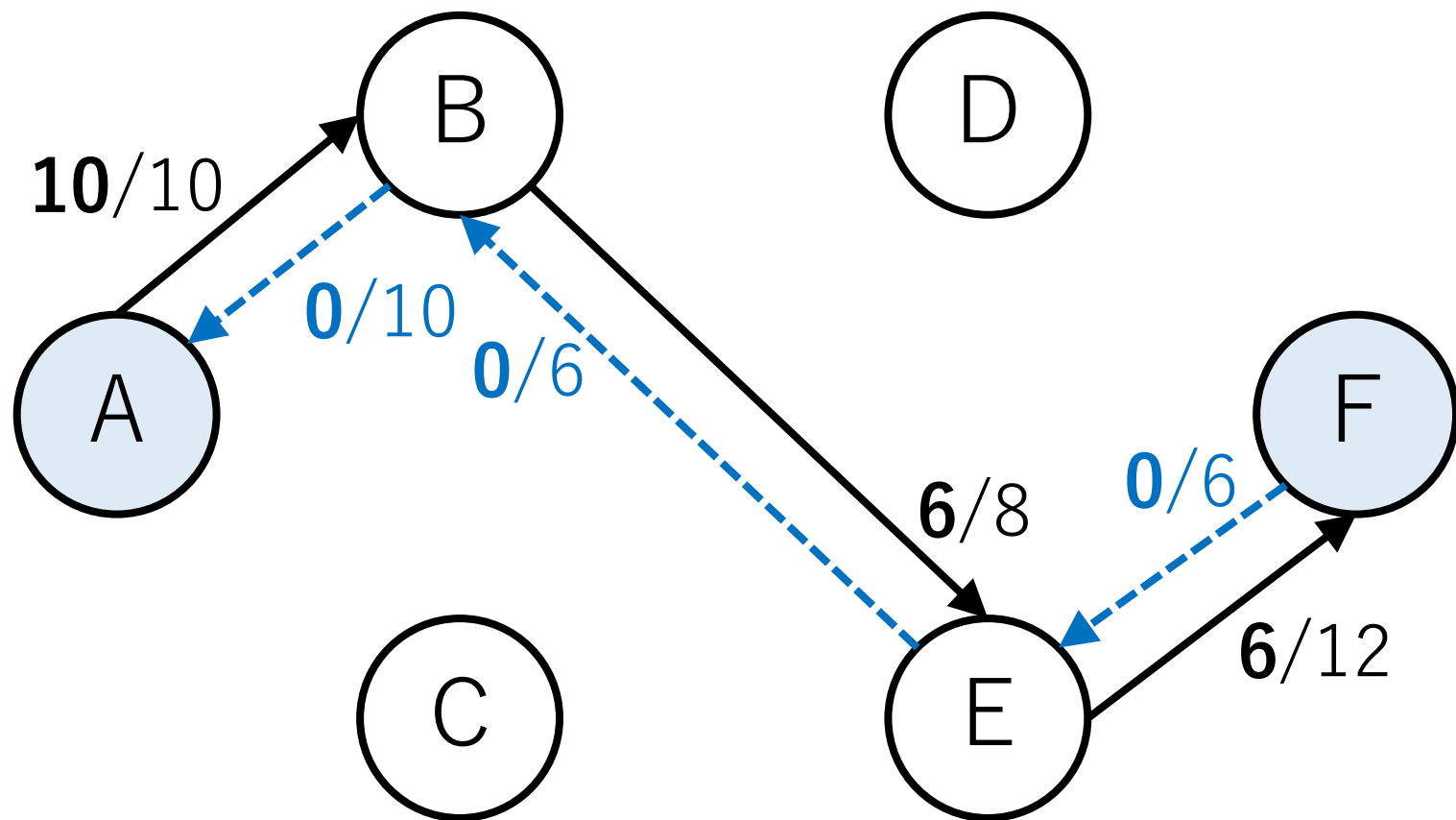
フォード・ファルカーソン法による例

逆方向の経路を設定.



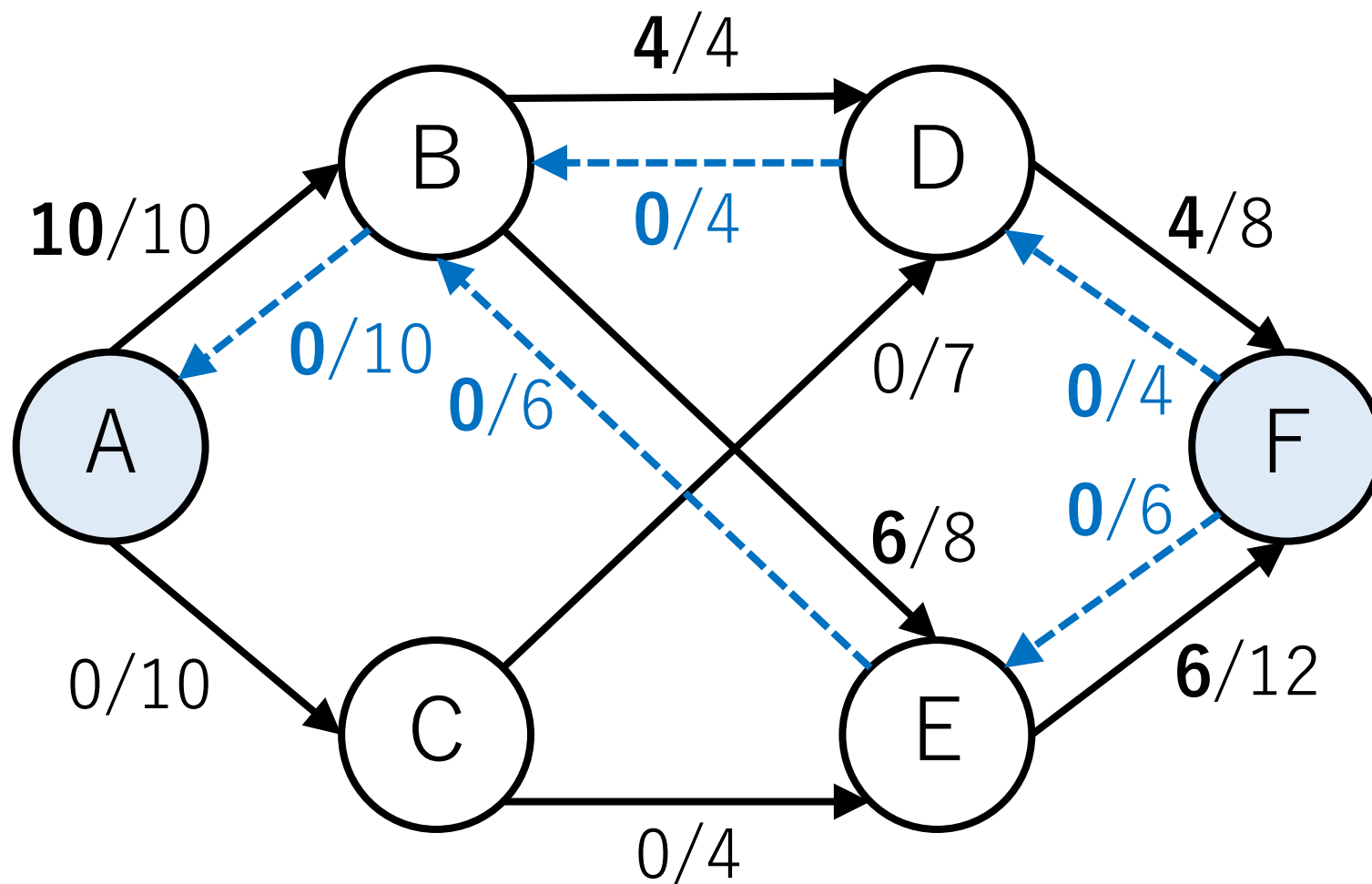
フォード・ファルカーソン法による例

「逆方向の容量 = 順方向の今までの流量」で容量を更新.



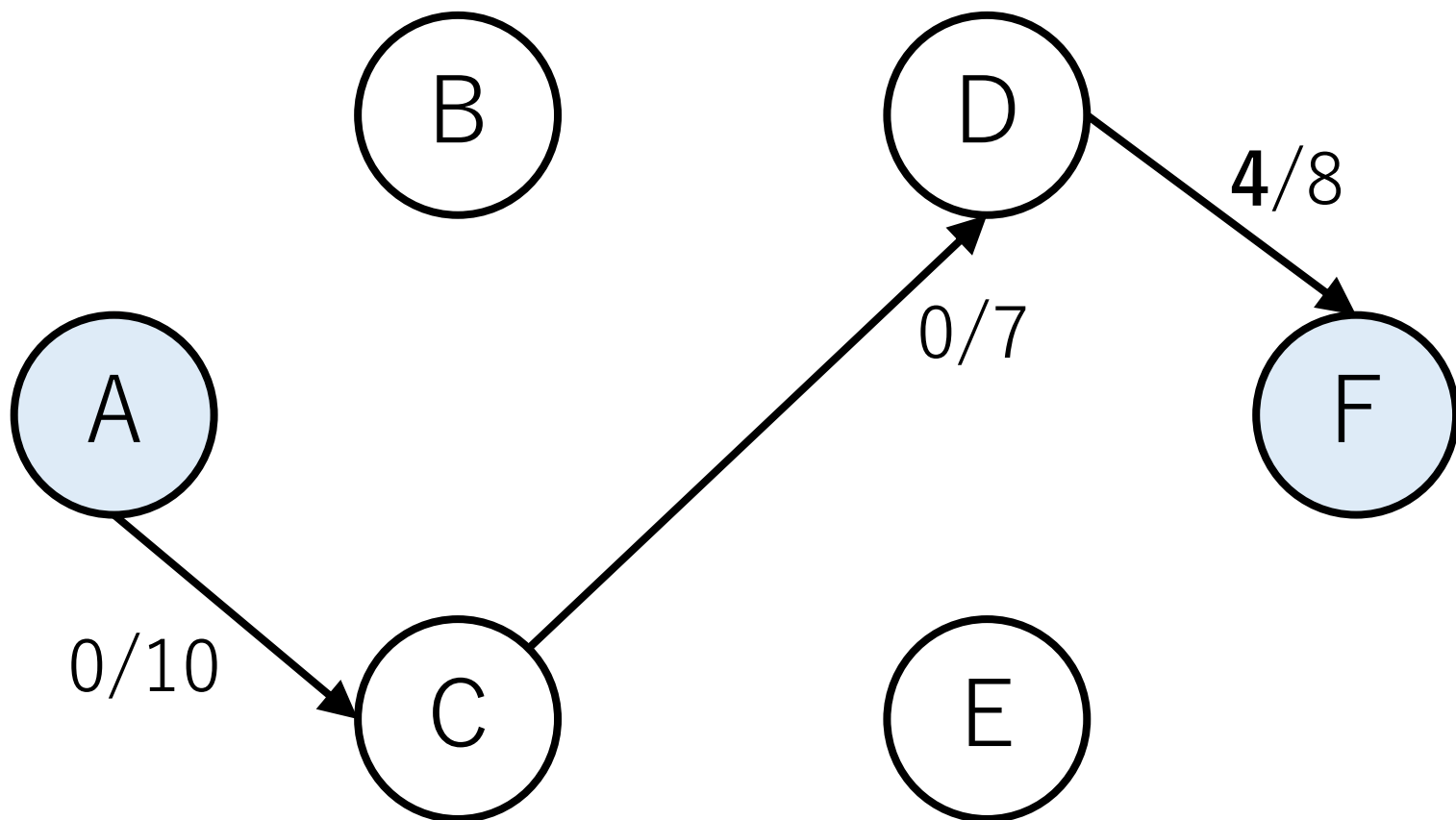
フォード・ファルカーソン法による例

現在の状態.



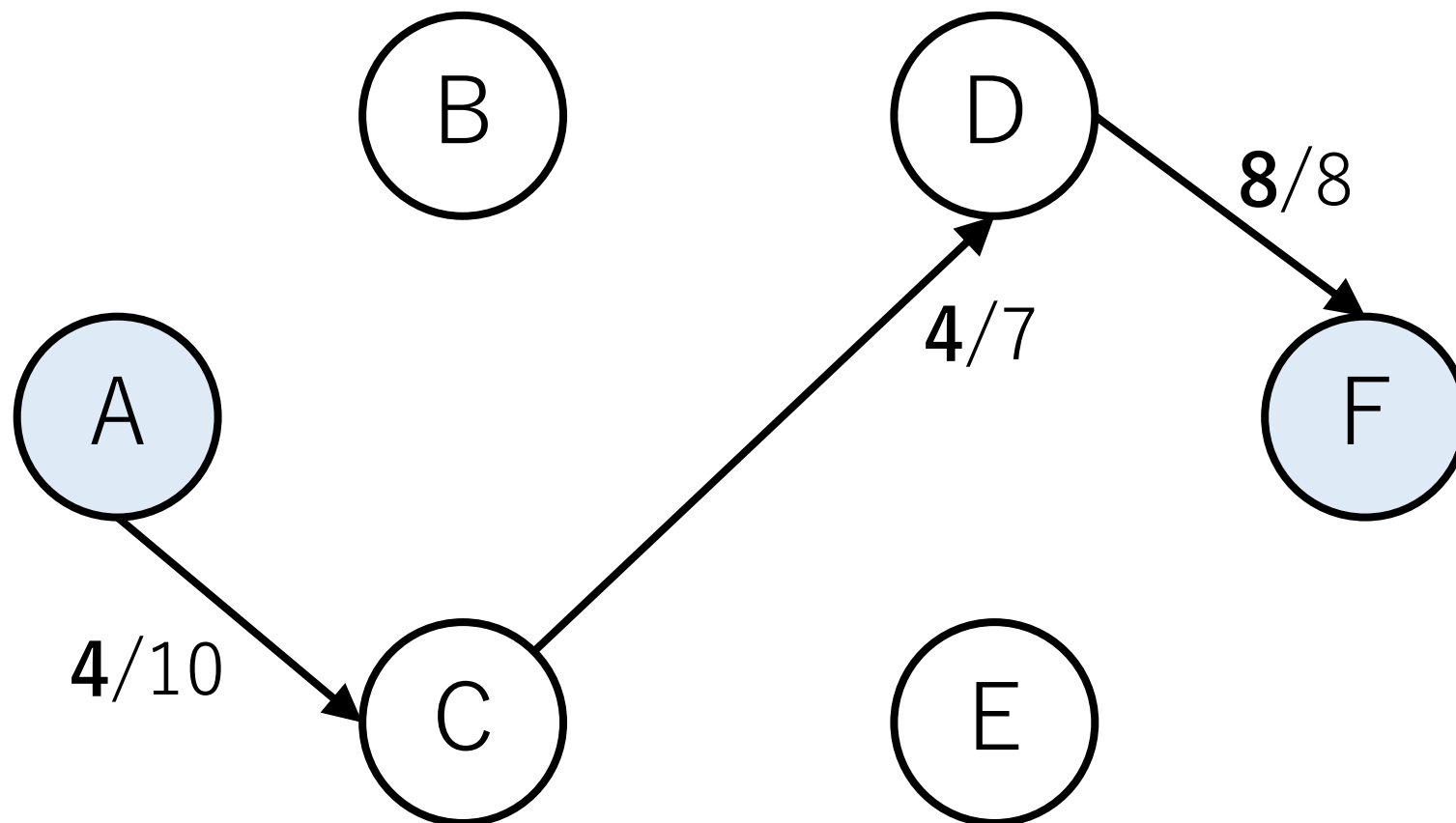
フォード・ファルカーソン法による例

A->C->D->Fを見つける。流せる量は4。



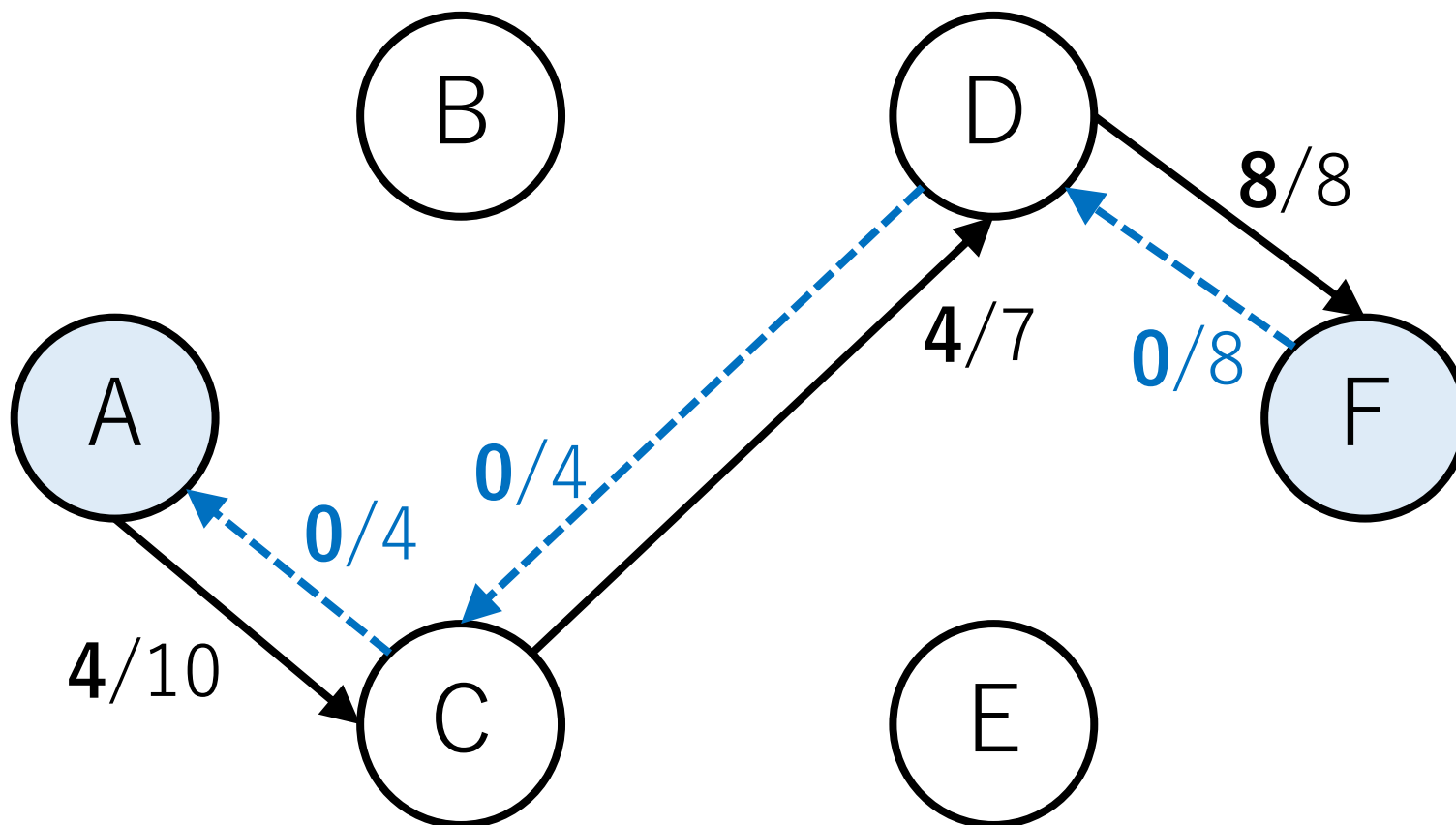
フォード・ファルカーソン法による例

4流す。



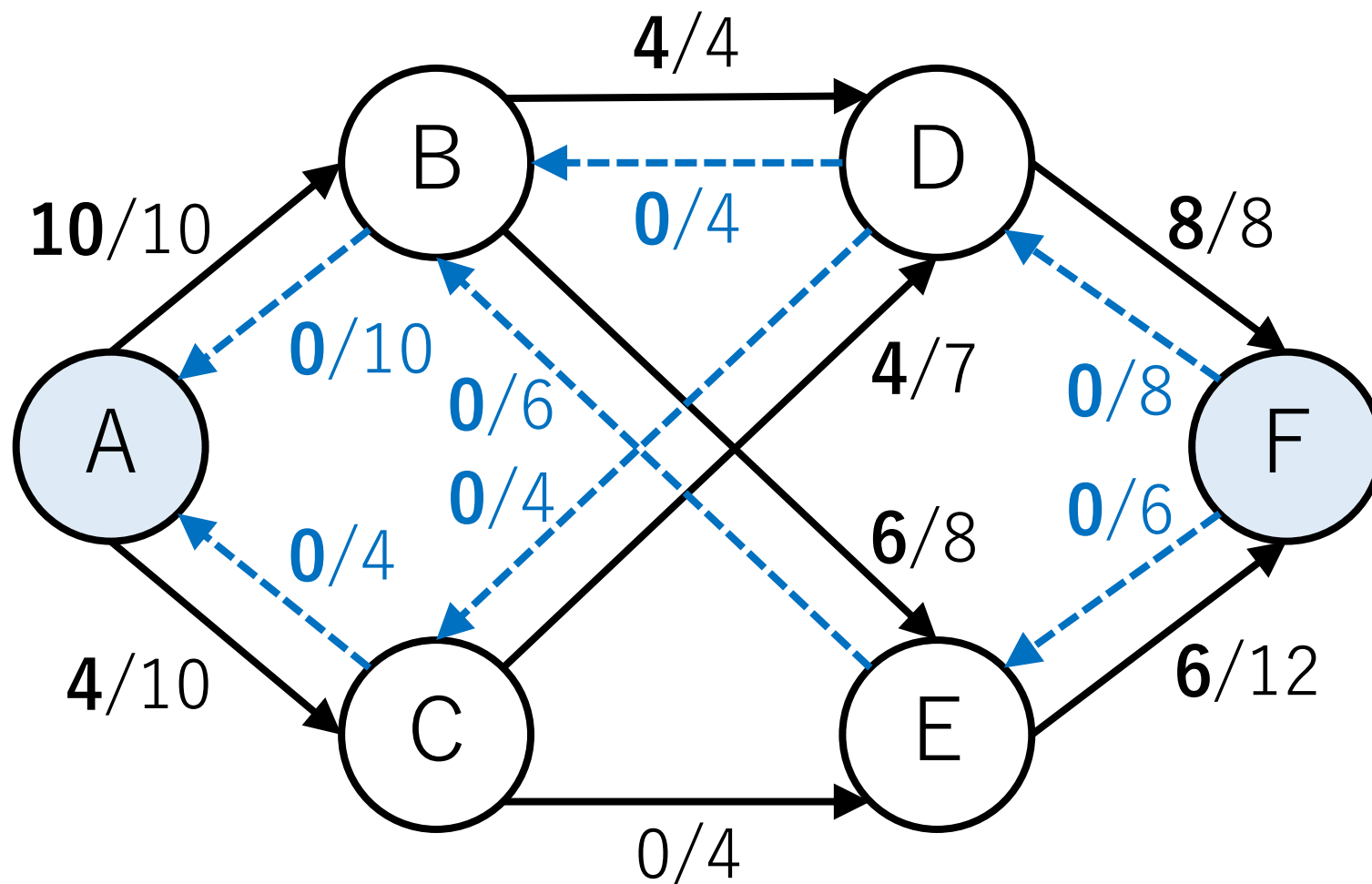
フォード・ファルカーソン法による例

逆方向の経路を設定，容量を更新．



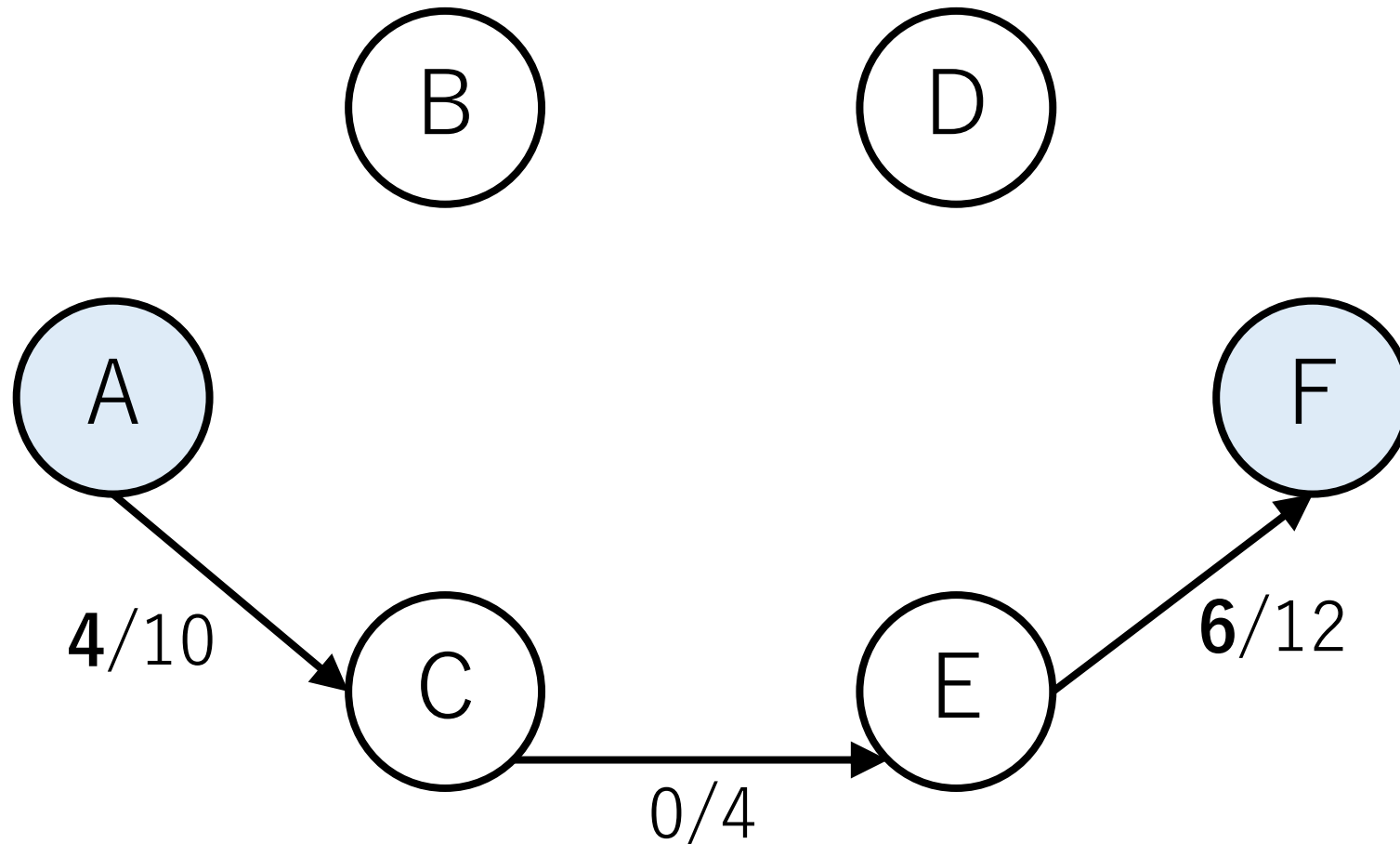
フォード・ファルカーソン法による例

現在までの状態.



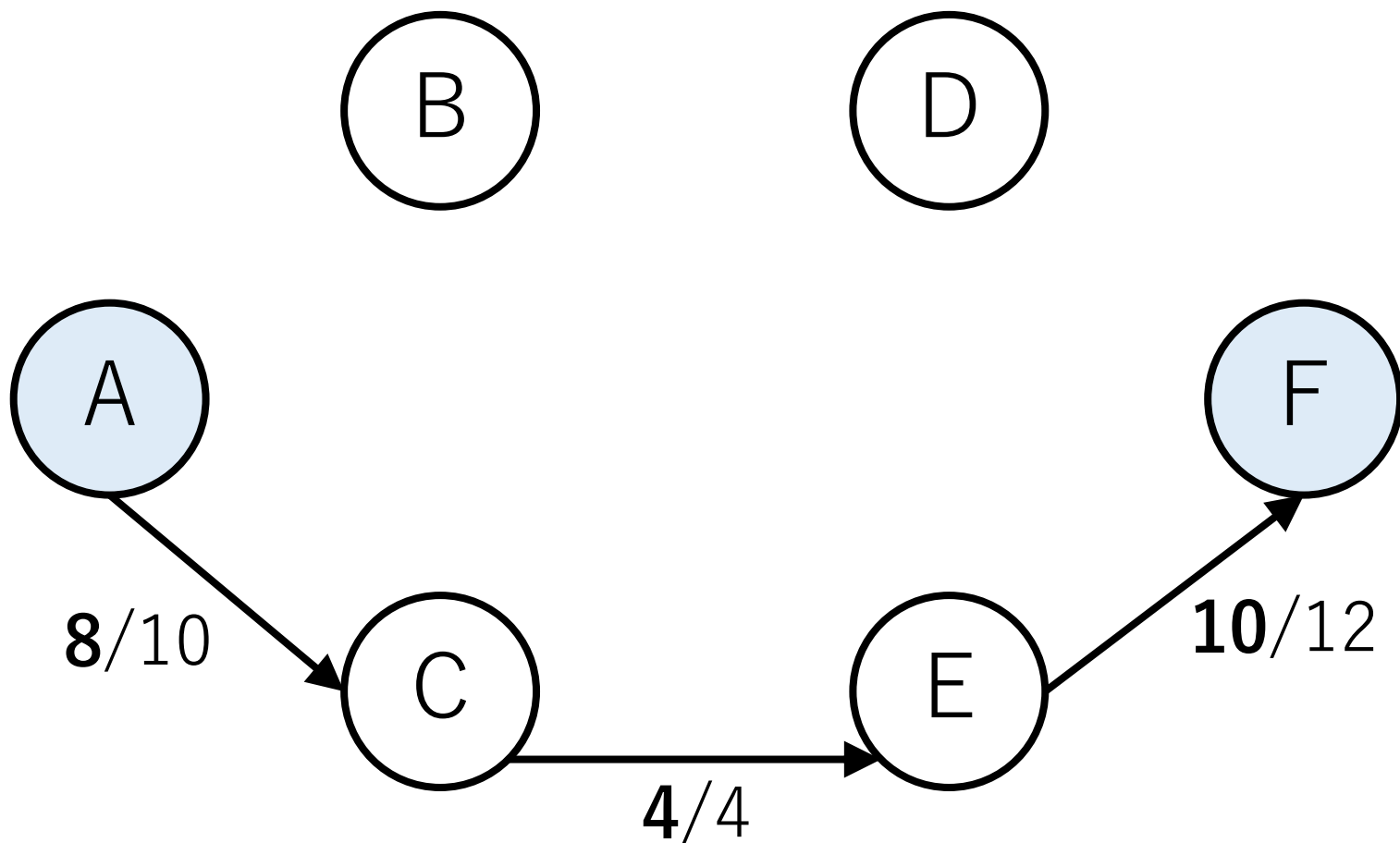
フォード・ファルカーソン法による例

A->C->E->Fを見つける。流せる量は4。



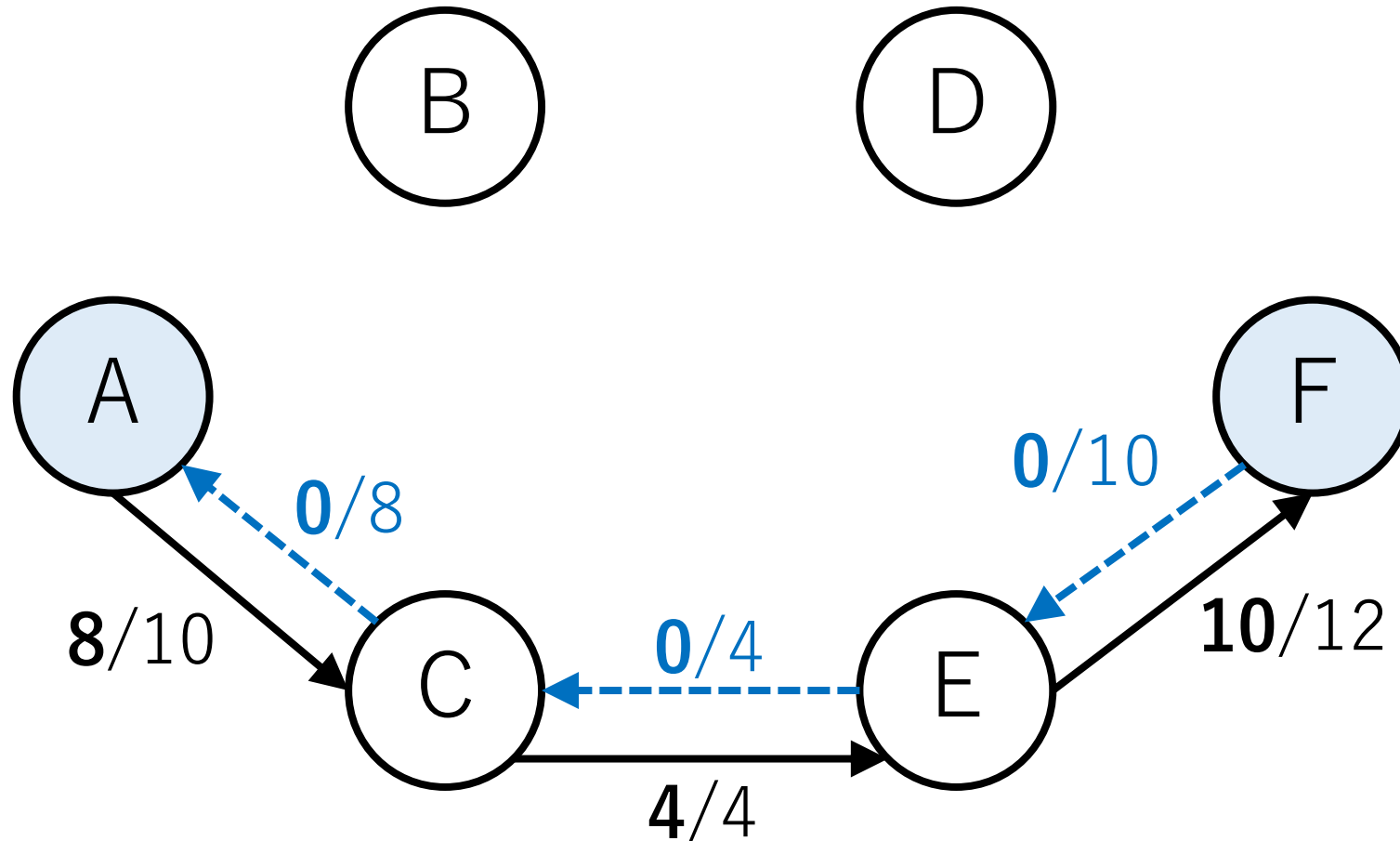
フォード・ファルカーソン法による例

4流す。



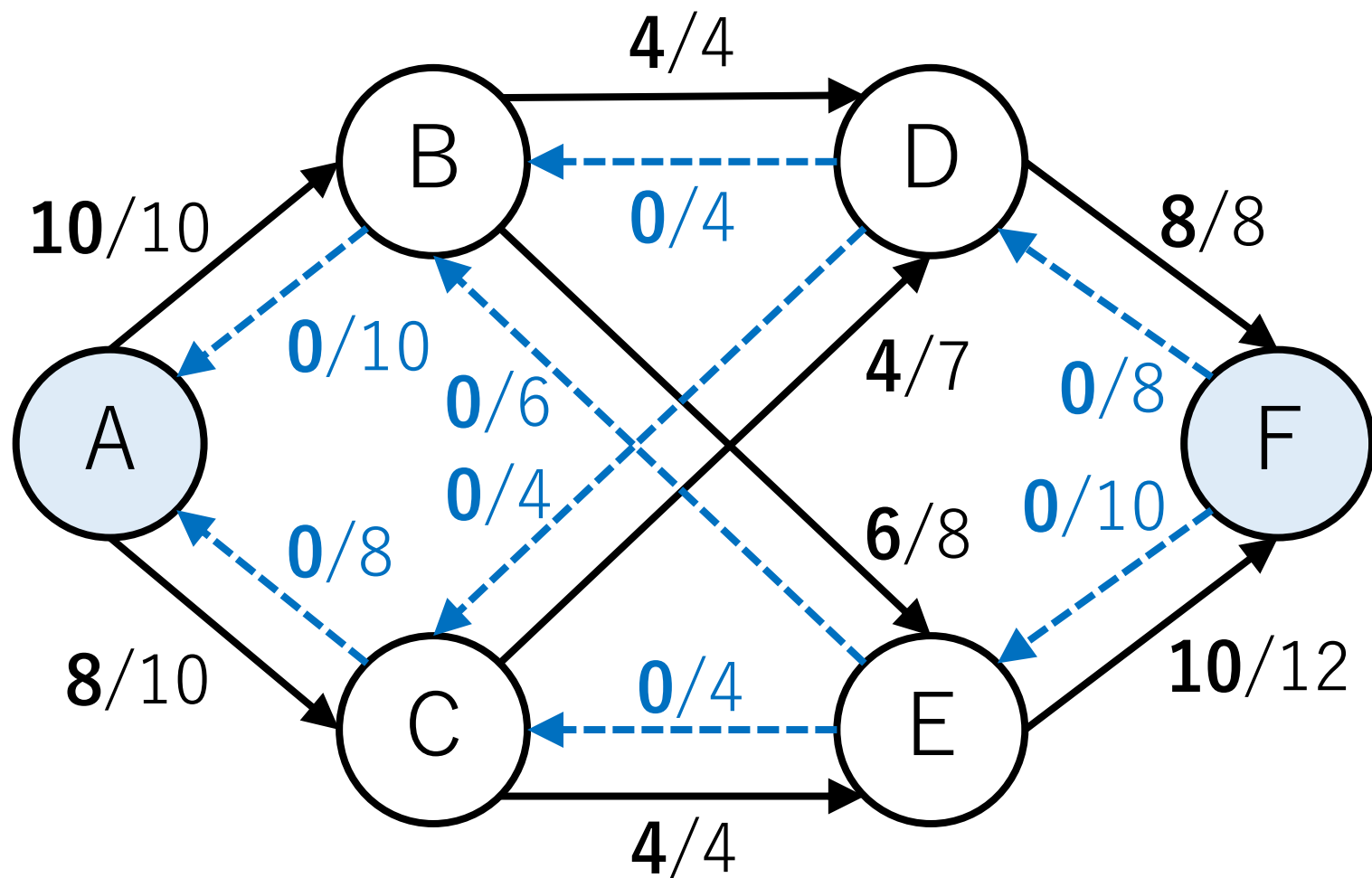
フォード・ファルカーソン法による例

逆方向の経路を設定，容量を更新.



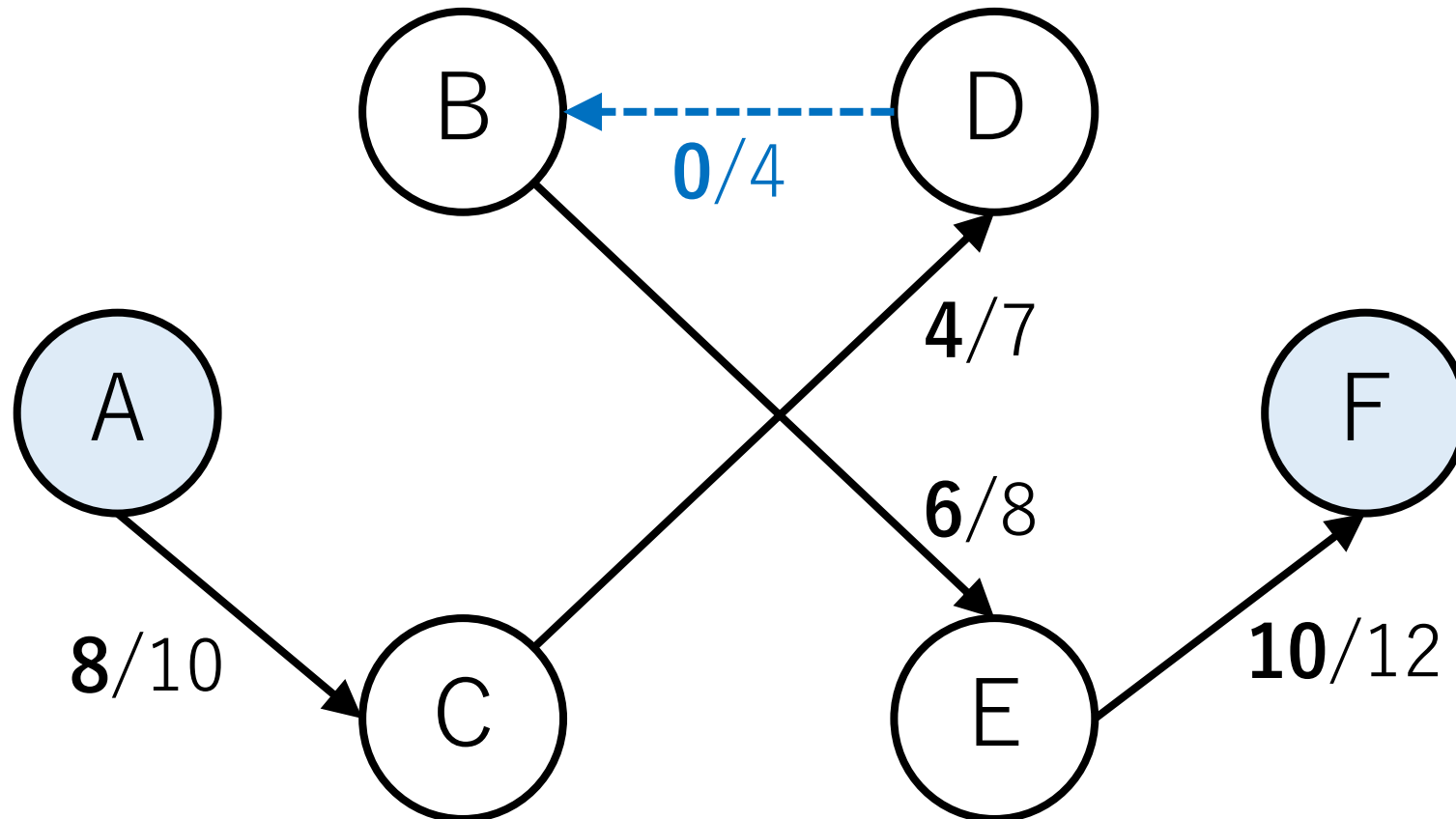
フォード・ファルカーソン法による例

現在までの状態. (ここまでは貪欲法の例と同じ.)



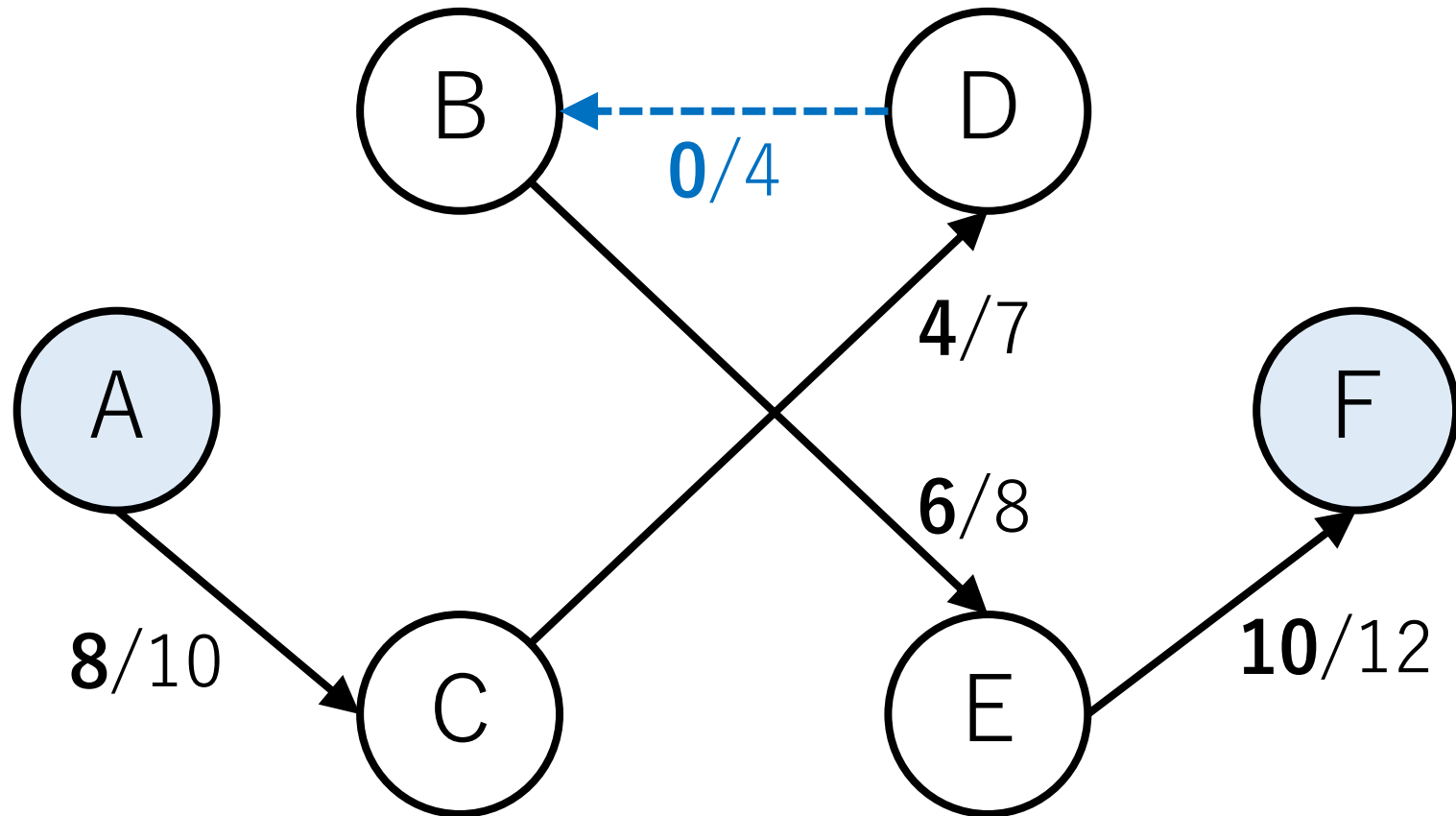
フォード・ファルカーソン法による例

まだ経路が存在する. $A \rightarrow C \rightarrow D \rightarrow B \rightarrow E \rightarrow F$.



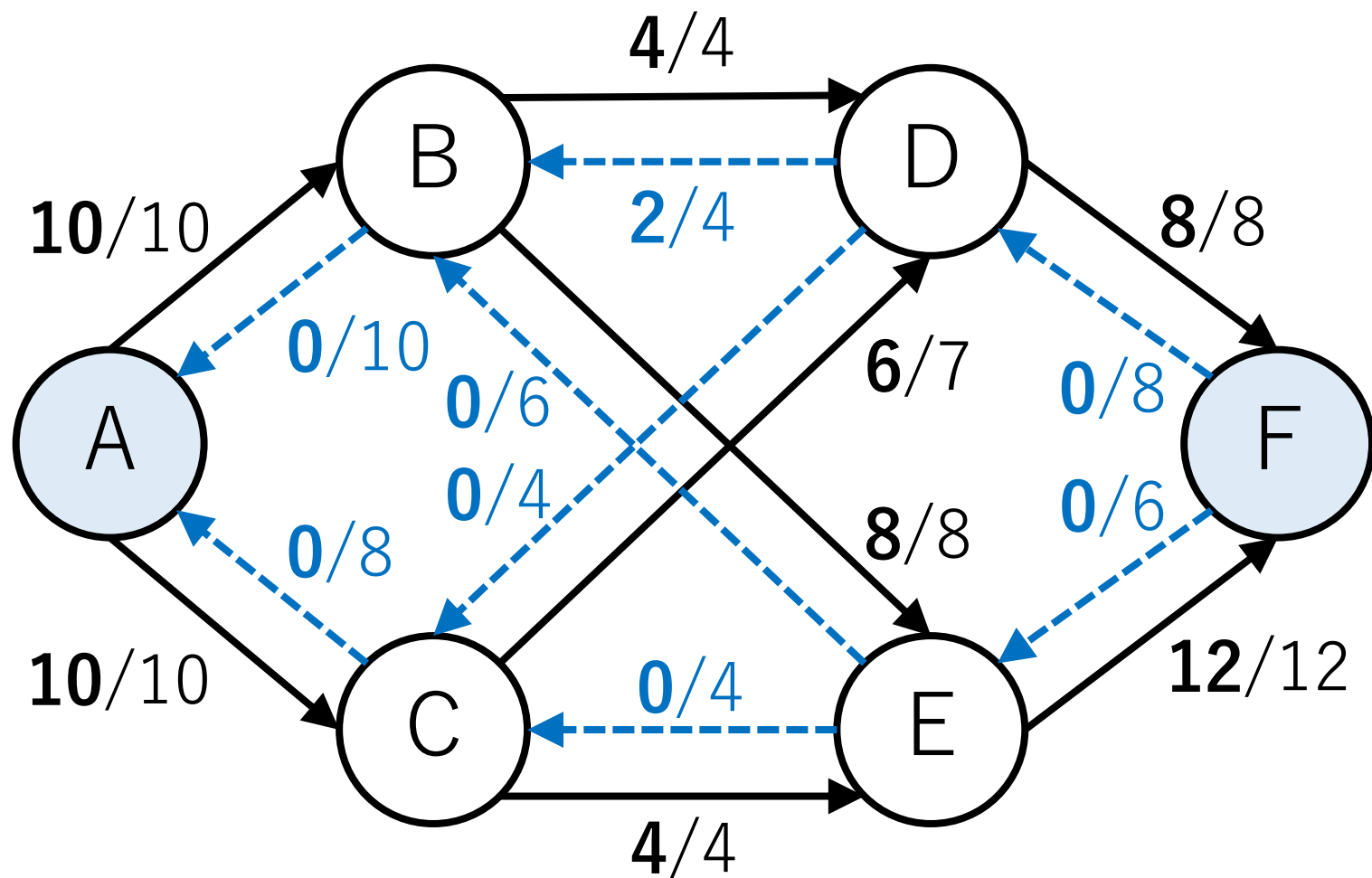
フォード・ファルカーソン法による例

流せる流量は2.



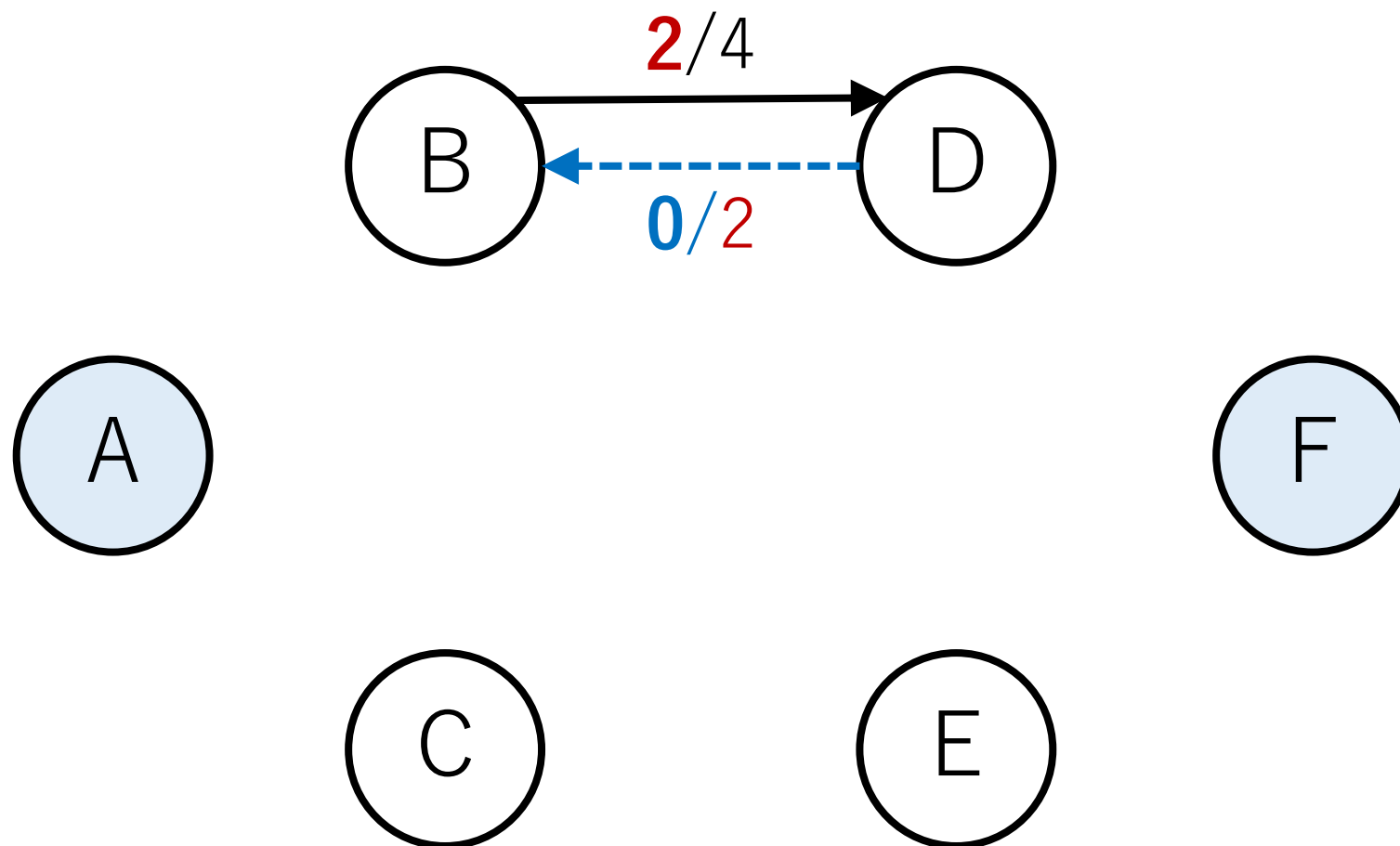
フォード・ファルカーソン法による例

A->C->D->B->E->Fの流量を入れて更新.



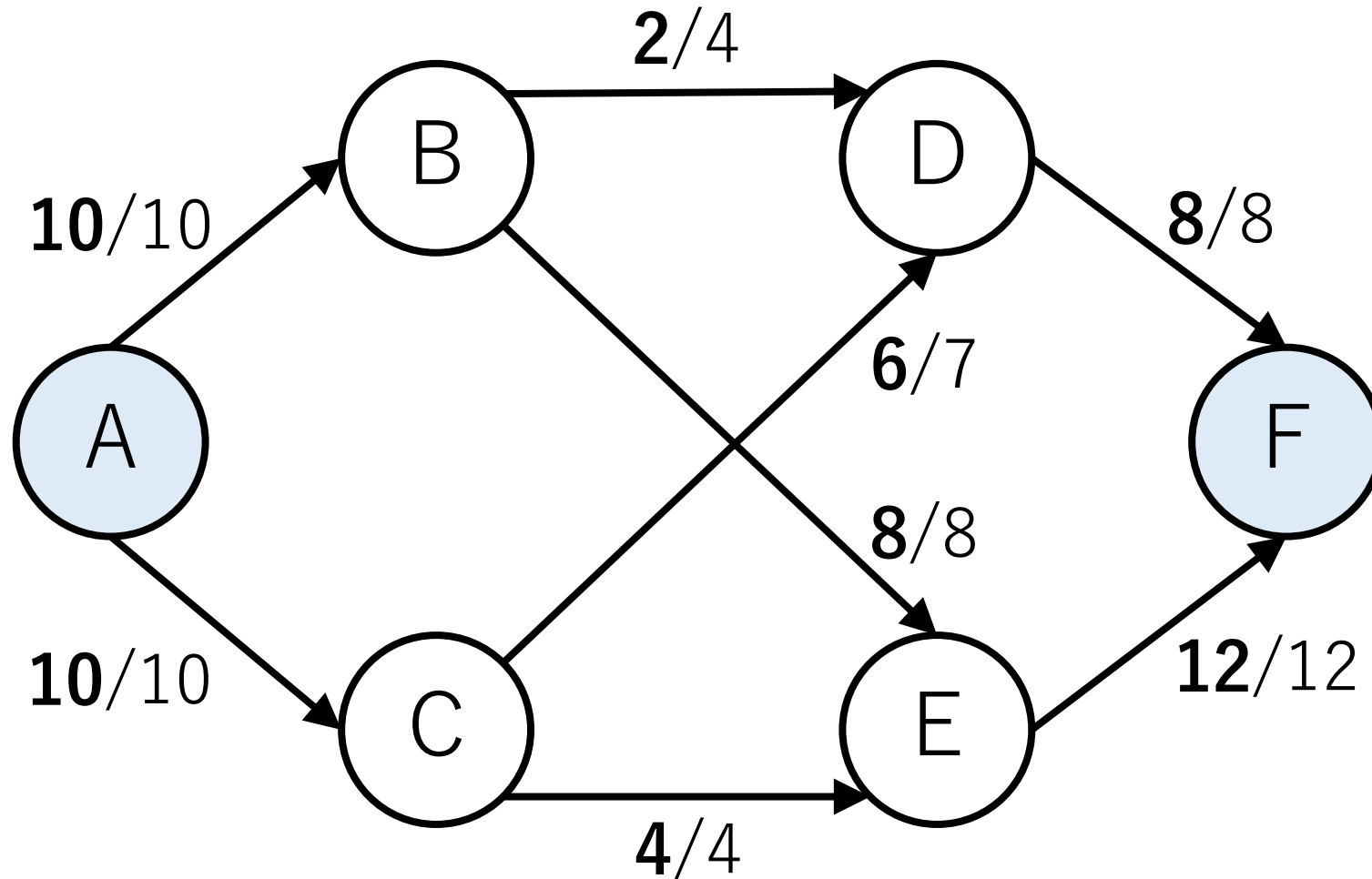
フォード・ファルカーソン法による例

B->Dの流量, D->Bの容量を2に変更.



フォード・ファルカーソン法による例

以上で終了.



フォード・ファルカーソン法実装例

以下，隣接行列を使う実装で説明。

```
capacity = [  
[0, 10, 10, 0, 0, 0],    # A->B: 10, A->C: 10  
[0, 0, 0, 4, 8, 0],     # B->D: 4, B->E: 8  
[0, 0, 0, 7, 4, 0],     # C->D: 7, C->E: 4  
[0, 0, 0, 0, 0, 8],     # D->F: 8  
[0, 0, 0, 0, 0, 12],    # E->F: 12  
[0, 0, 0, 0, 0, 0]     # Fから出ていくものなし。  
]
```

フォード・ファルカーソン法実装例

`max_flow = 0`

`V = 6`

[以下を流せるパスが見つかるまでループ]:

[DFSを使って, start – end間をつなぐパスを1つ
見つけ, そのパスに流せる流量を返す]

[その流量分をmax_flowに足す]

`print(max_flow)` # 最大フロー

フォード・ファルカーソン法実装例

ループ部分の実装

while True:

 # 毎回のDFSで使う訪問フラグ

 visited = [False for _ in range(V)]

 f = dfs_ff(0, 5, 10**9) # DFS実行

 if not f: break # もう流せないなら終了

 max_flow += f

フォード・ファルカーソン法実装例

$s - e$ 間においてある1つのパスで流せる流量を返す

引数: DFSの開始ノード, DFSの終了ノード, DFSの開始ノードの前のノードまでに流せる最大流量

```
def dfs_ff(s, e, flow):
```

フォード・ファルカーソン法実装例

```
def dfs_ff(s, e, flow):  
    # sとeが一緒のノードなら今までの最大流量  
    # をそのまま返す  
    if (s == e): return flow  
  
    # ノードsを訪問したと記録.  
    visited[s] = True
```

フォード・ファルカーソン法実装例

```
def dfs_ff(s, e, flow):
```

```
    ...
```

```
    # 流せる容量がある辺をすべてチェック
```

```
    # まだ見ていない辺があればDFS
```

```
    for i in range(V):
```

```
        [次のスライド参照]
```

```
    return 0 # 辺を全部見ていれば終了
```

フォード・ファルカーソン法実装例

```
def dfs_ff(s, e, flow):
```

```
    ...
```

```
    for i in range(len(capacity[start])):
```

```
        #まだ訪問していないノードで流量に余裕がある場合
```

```
        if not visited[i] and capacity[s][i]:
```

```
            # 今までの最大流量と、ノードs - ノードi間  
            # の残りの容量のうち、小さい方を次へ送る.
```

```
            f = dfs_ff(i, end, min(flow, capacity[s][i]))
```

フォード・ファルカーソン法実装例

```
def dfs_ff(s, e, flow):
```

```
    ...
```

```
        f = dfs_ff(i, e, min(flow, capacity[s][i]))
```

```
        if f > 0: # 新しく流せる経路があった
```

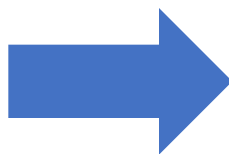
```
            capacity[s][i] -= f          # 順方向
```

```
            capacity[i][s] += f          # 逆方向
```

```
        return f
```


実行結果 (1パス分終了後のcapacity)

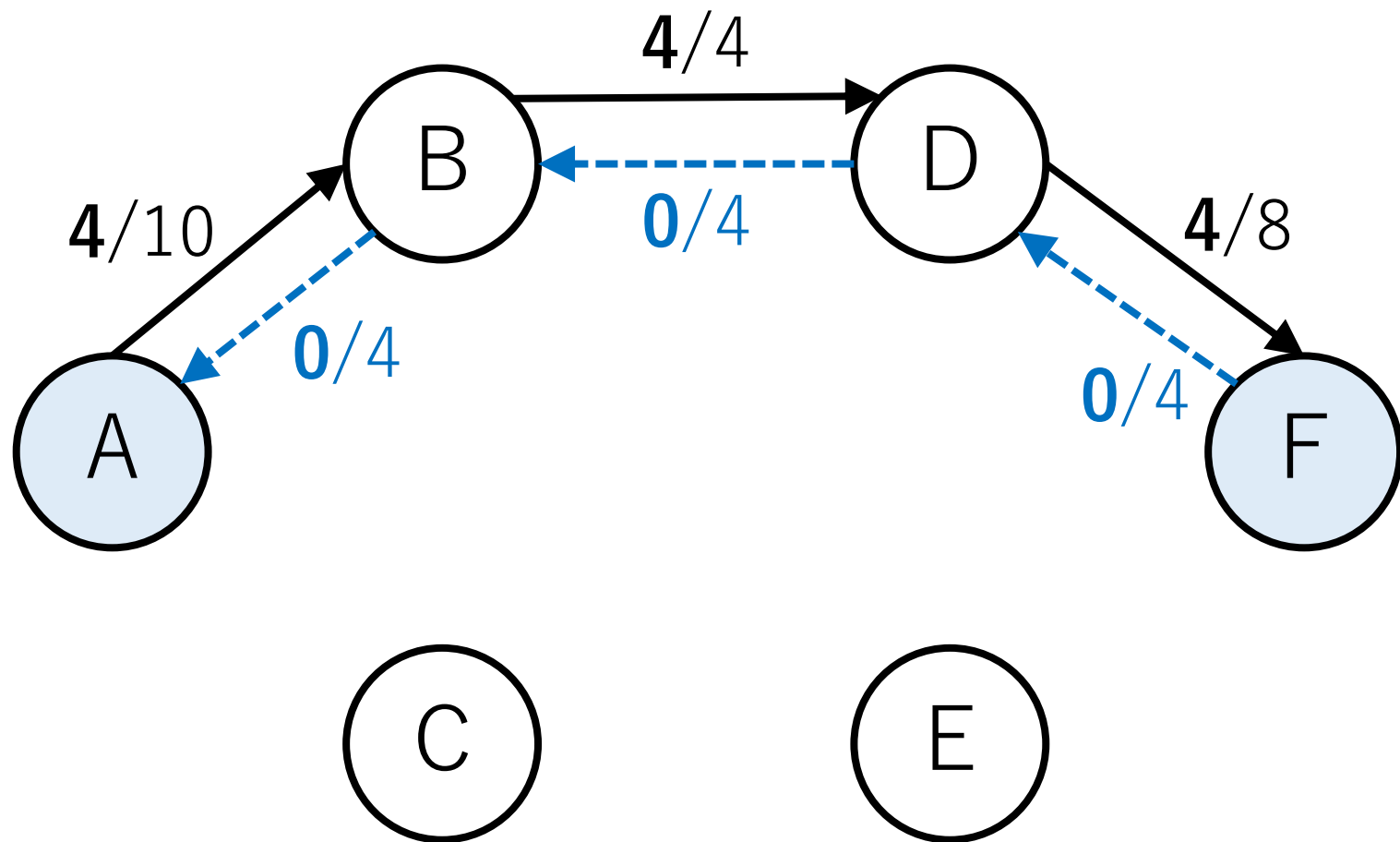
[[0, 10, 10, 0, 0, 0],
[0, 0, 0, 4, 8, 0],
[0, 0, 0, 7, 4, 0],
[0, 0, 0, 0, 0, 8],
[0, 0, 0, 0, 0, 12],
[0, 0, 0, 0, 0, 0]]



[[0, 6, 10, 0, 0, 0],
[4, 0, 0, 0, 8, 0],
[0, 0, 0, 7, 4, 0],
[0, 4, 0, 0, 0, 4],
[0, 0, 0, 0, 0, 12],
[0, 0, 0, 4, 0, 0]]

実行結果 (1パス分終了後のcapacity)

```
[[0, 6, 10, 0, 0, 0],  
[4, 0, 0, 0, 8, 0],  
[0, 0, 0, 7, 4, 0],  
[0, 4, 0, 0, 0, 4],  
[0, 0, 0, 0, 0, 12],  
[0, 0, 0, 4, 0, 0]]
```



実行結果

残りの実行結果もぜひ追ってみてください。

(授業中に説明したとおりに経路の容量値が変化することは限りませんが、最終的には最適解になります。)

フォード・ファルカーソン法の計算量

ノード数 $|V|$, 最大流の最適値を F とする.

最悪の場合, 新しい経路が見つかってても最大流が1つしか増やせない. $\rightarrow F$ 回の最大流値の更新が発生.

経路探索のためには高々 $|V|$ のノードを訪れる. 隣接行列の場合, 辺のチェックのために, 毎回 $O(|V|)$ 個の操作が必要.

よって, 上記実装では $O(F|V|^2)$. 隣接リストなら $O(F|E|)$.

グラフのカット

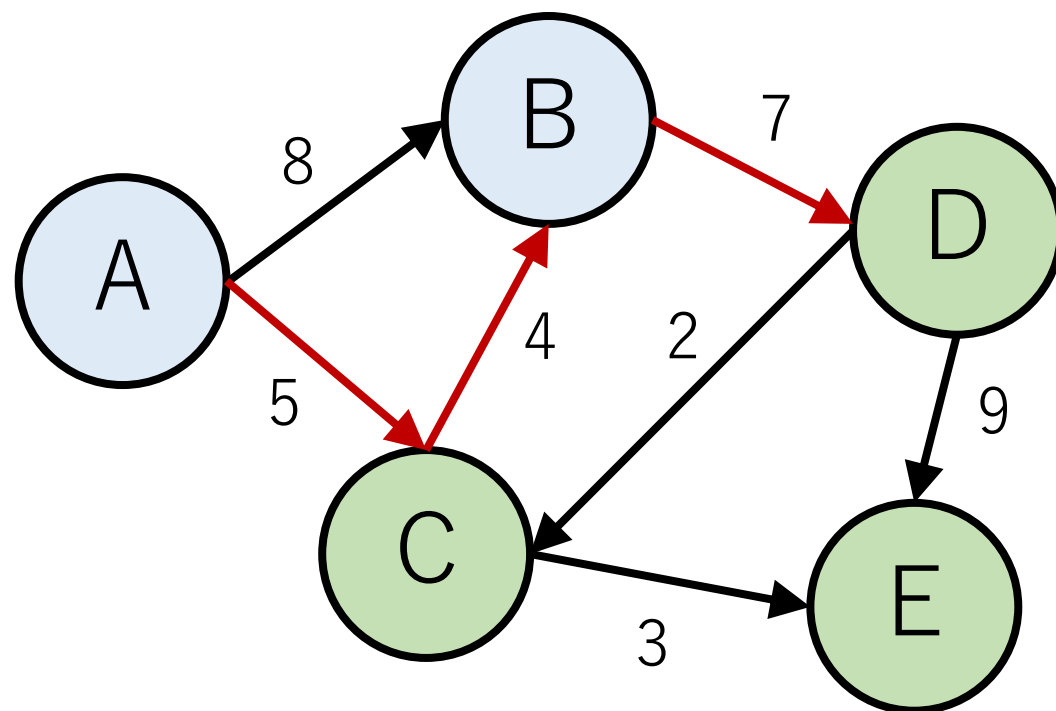
カット = あるグラフを2つのグループ（集合）に分ける。
 $(S, V \setminus S)$ のように表す。

カットエッジ = 辺のうち、ノードが別々のグループに属しているものの全部。

グラフのカット

例えば, $S = \{A, B\}$ と $V \setminus S = \{C, D, E\}$ の2つのグループに分ける (カットを作る).

このときのカットエッジは, 赤色の辺となる.



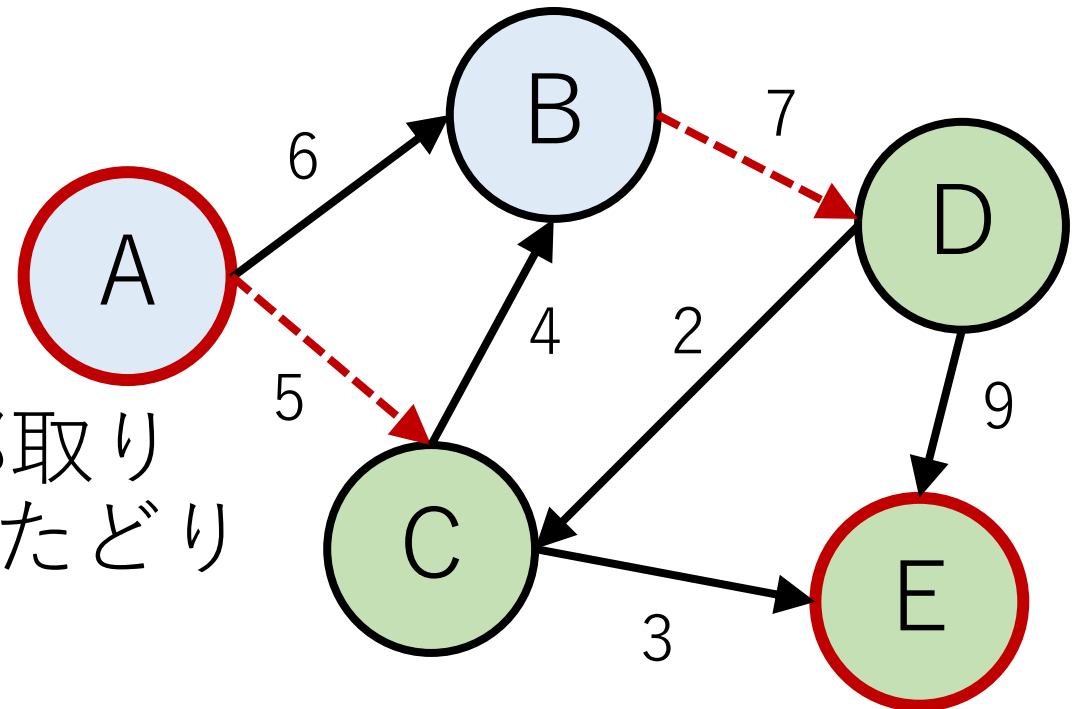
s-tカット

S に開始ノードが、 $V \setminus S$ に終点ノードが含まれるカット。

右図の場合、開始ノードがA、終点ノードがEであれば、s-tカットとなる。

s-tカットはこれ以外にもいっぱいある。

s-tカットのカットエッジを全部取り除くと、開始から終点ノードへたどり着けなくなる。

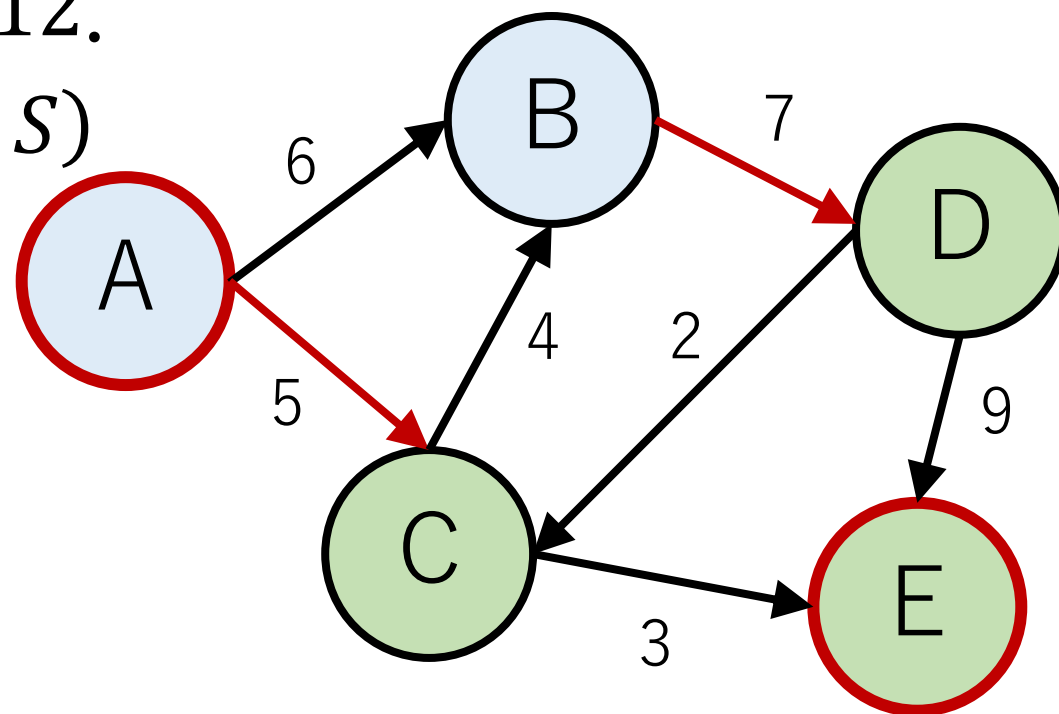


s-tカット

S から $V \setminus S$ に向かう辺の容量の総和を $U(S \rightarrow V \setminus S)$ と表す.

右の図の場合, $U(S \rightarrow V \setminus S) = 12$.

CからBの容量は $U(S \rightarrow V \setminus S)$ には入らない.



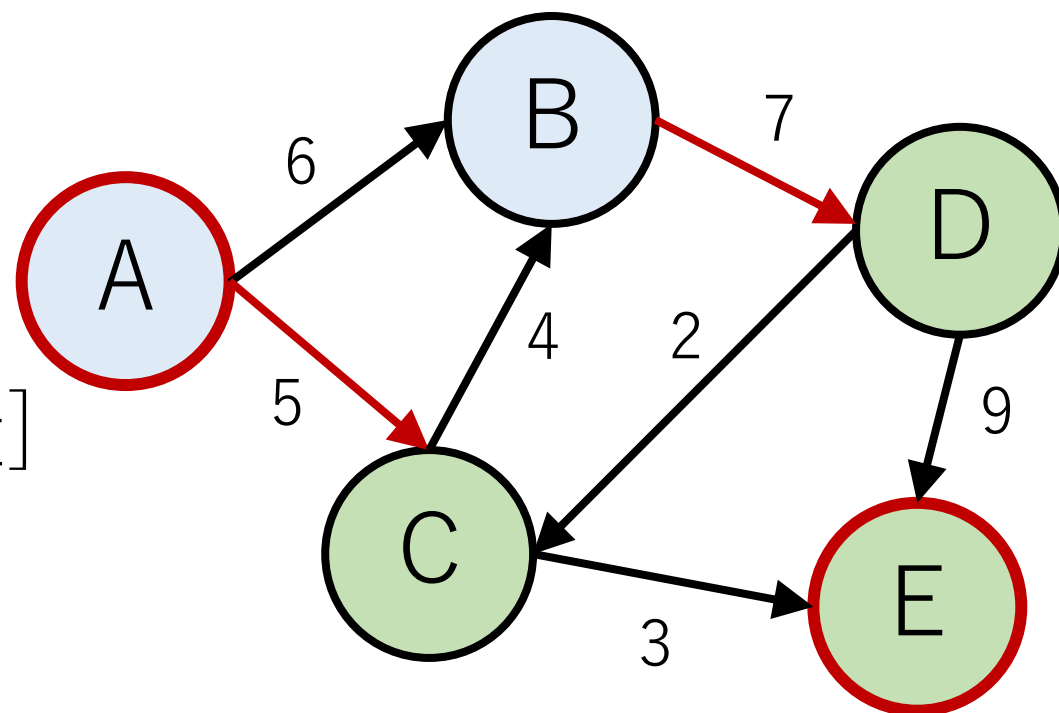
s-tカットの容量とフローの関係

フロー f は以下のように計算される。

$$f = [S \text{ から } V \setminus S \text{ への流入量}] - [V \setminus S \text{ から } S \text{ への流入量}]$$

$[V \setminus S \text{ から } S \text{ への流入量}]$ は0かそれ以上なので、

$$f \leq [S \text{ から } V \setminus S \text{ への流入量}]$$

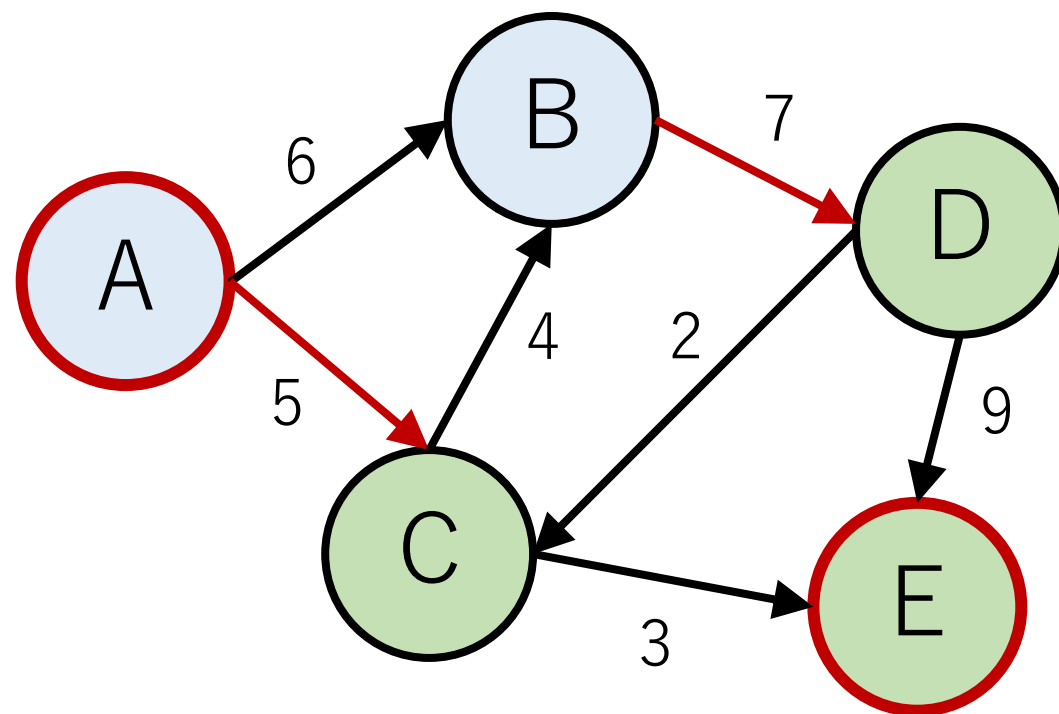


s-tカットの容量とフローの関係

さらに, $[S$ から $V \setminus S$ への流入量]はそのパスの容量を超えることはない. よって,

$$f \leq [S \text{から} V \setminus S \text{への流入量}] \leq U(S \rightarrow V \setminus S)$$

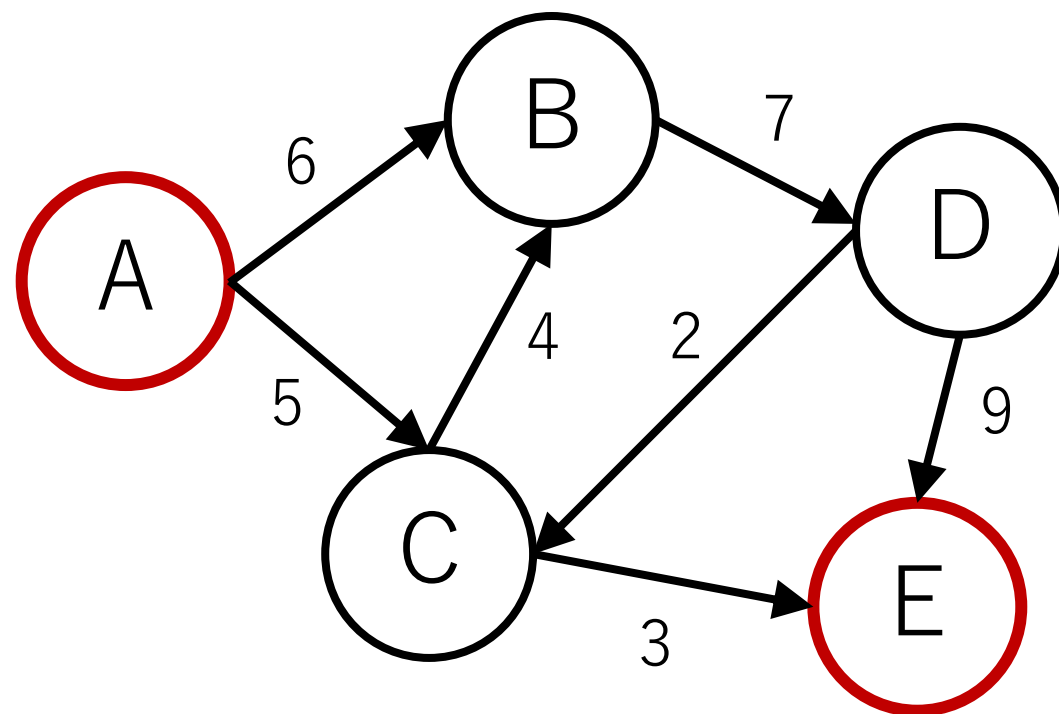
これは任意のs-tカットにおいて成立する不等式となる.



最小カット

s-tカットにおいて, S から $V \setminus S$ へ向かう辺の容量の和 $U(S \rightarrow V \setminus S)$ が最小になるようなカット.

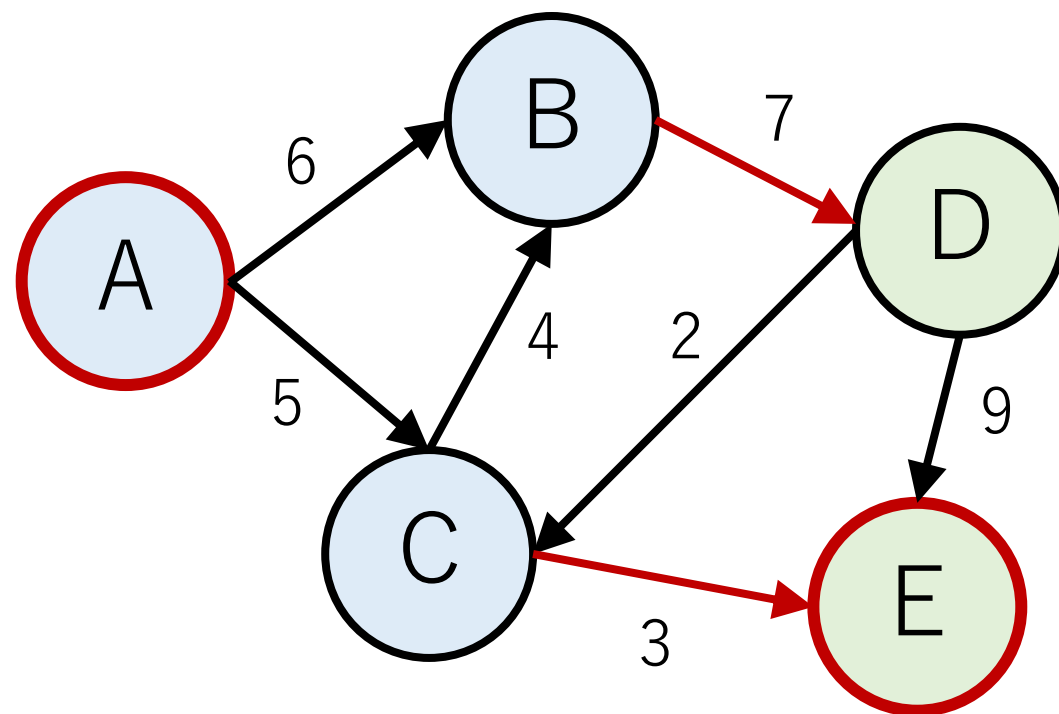
右図の場合はいくら？



最小カット

右図の場合, $U_{min}(S \rightarrow V \setminus S) = 10$ となる.

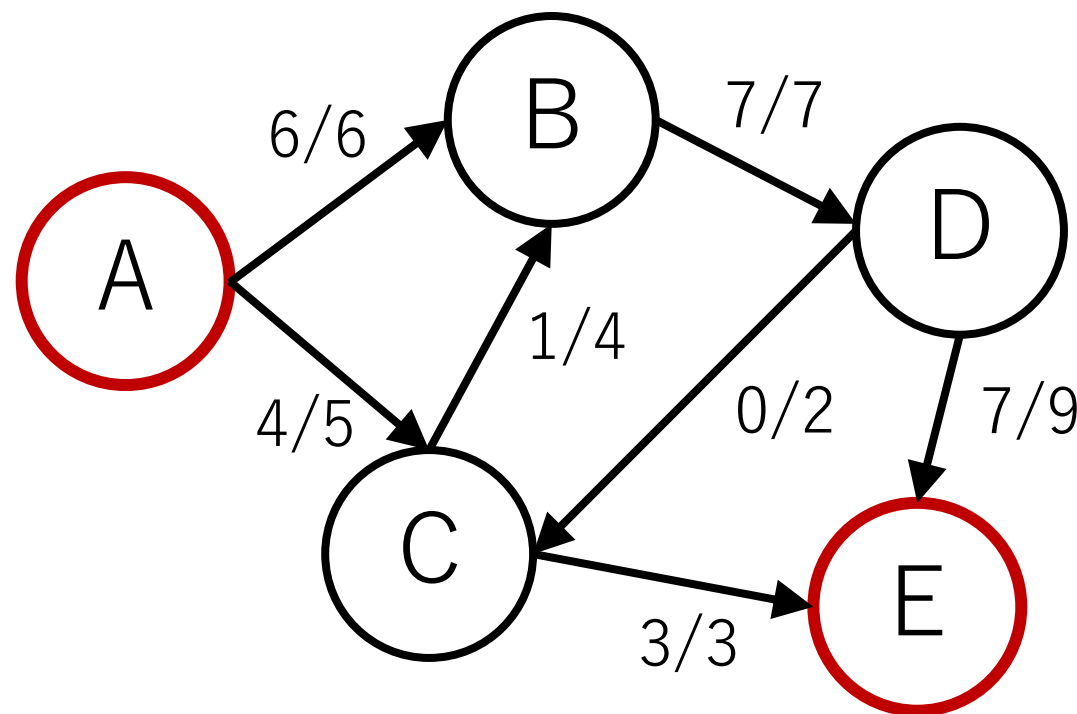
DからCの容量は $U(S \rightarrow V \setminus S)$
には入らないので, 注意.



ちなみに、

このグラフで最大流を求めると、10.

なんとなく、最大流と最小カット
関係ありそう... 😊



最大流・最小カット定理

「与えられたグラフにおいて、最大流の流量はs-tカットにおける最小カットの容量に等しい」

フォード・ファルカーソン法で最大流を求めることができることの基盤となっているもの。

以下ではこの定理に関する説明をしますが、フォード・ファルカーソン法が未消化な人は軽く聞いておいてもらえれば。😅

フォード・ファルカーソン法の終了状態

フォード・ファルカーソン法により最大流を求めた後、
仮想的に考えた逆辺を含めた残り容量のグラフを考える。

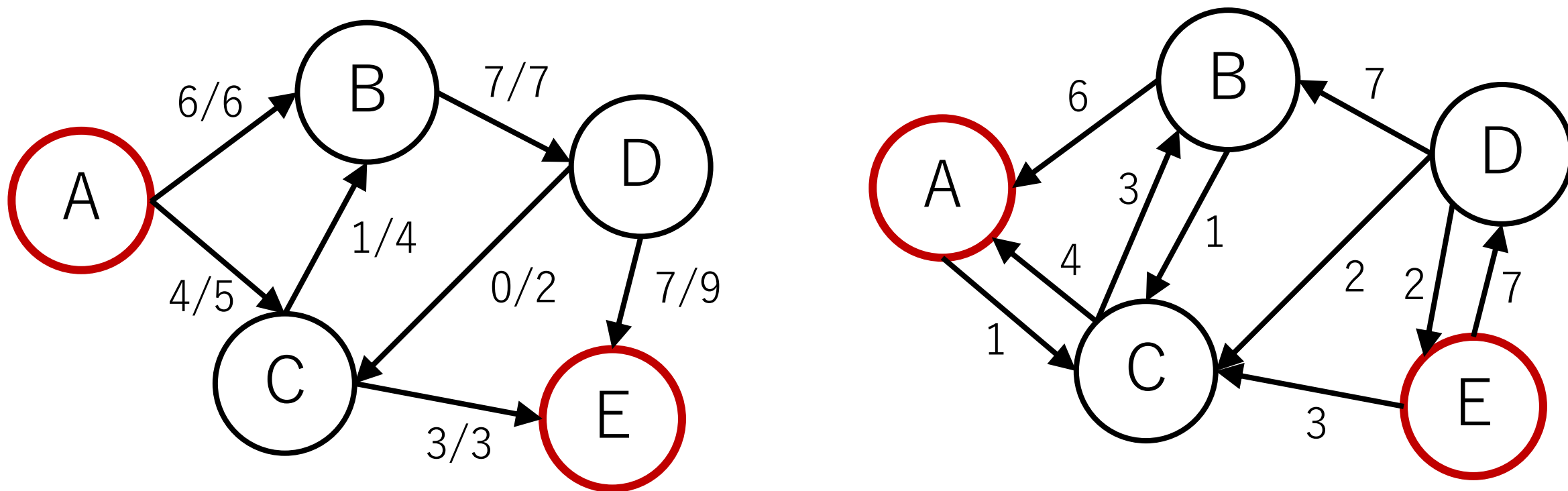
残余ネットワークなどとも呼ばれる。

このグラフにおいては、開始ノードから到達できる
ノードのグループと到達できないノードのグループに
自動的に分かれる。

つまりs-tカットが自動的に出来上がる。

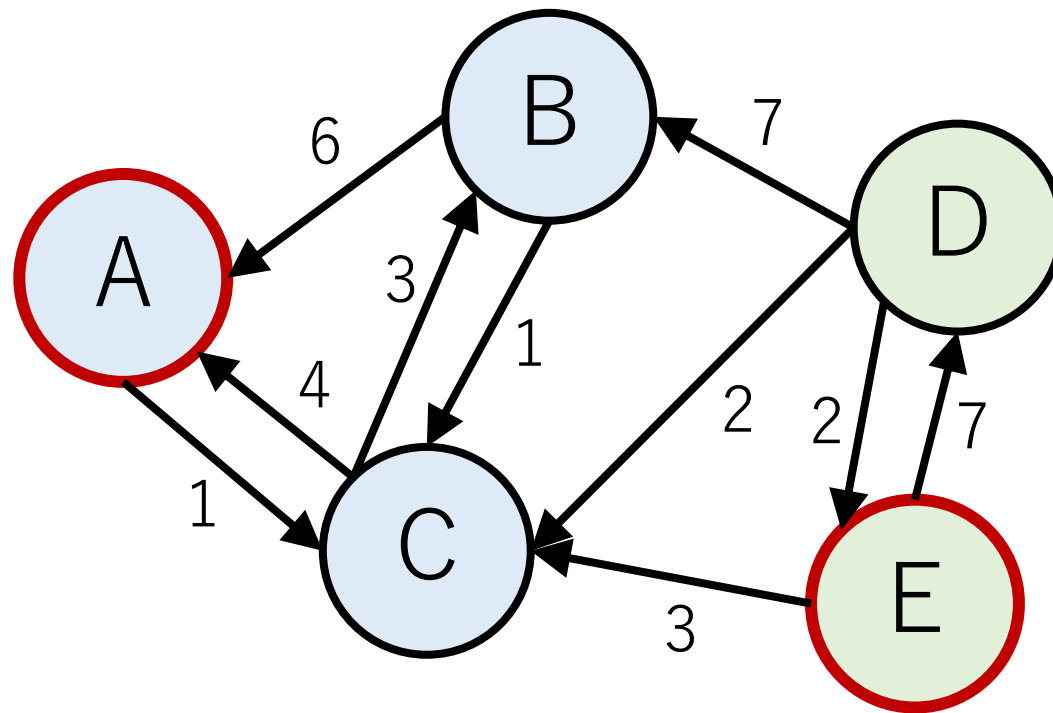
フォード・ファルカーソン法の終了状態

フォード・ファルカーソン法を終了したときの、各辺の容量（仮想的に考えた逆辺も含む）を考えよう。



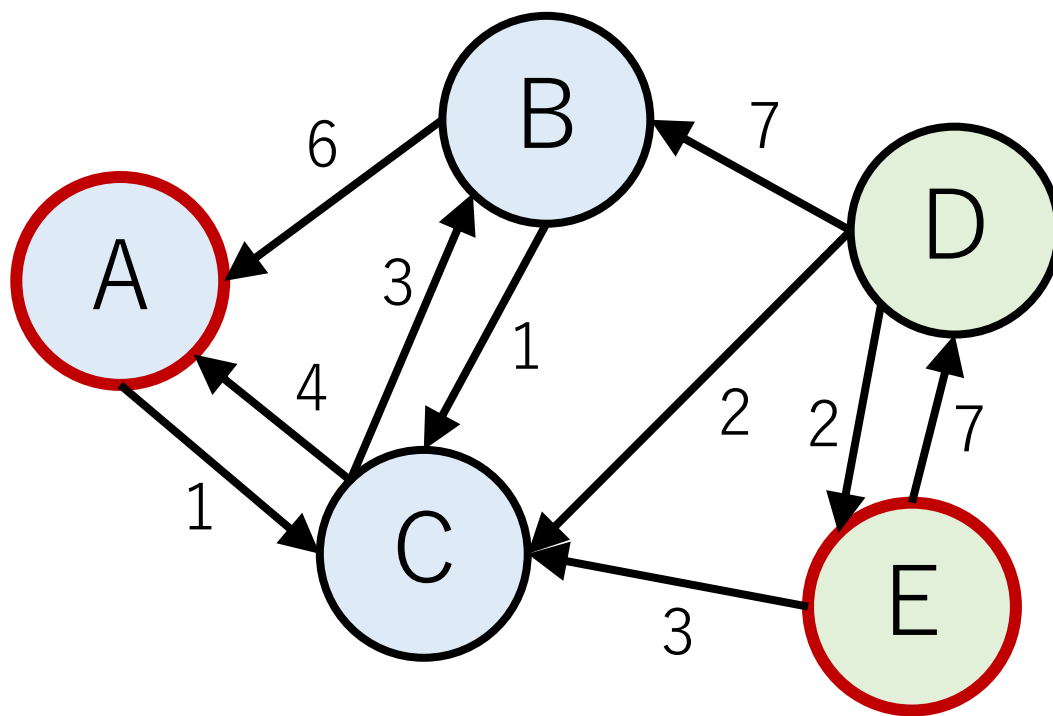
出来上がったs-tカット

以下のグラフにおいて、開始ノードAから辿ることのできるのは、BとC。これがSになる（青のグループ）。



最大流・最小カット定理

このs-tカットは最小のもの ($U(S \rightarrow V \setminus S)$ が最小のもの, 最小カット) となっていることを示そう。



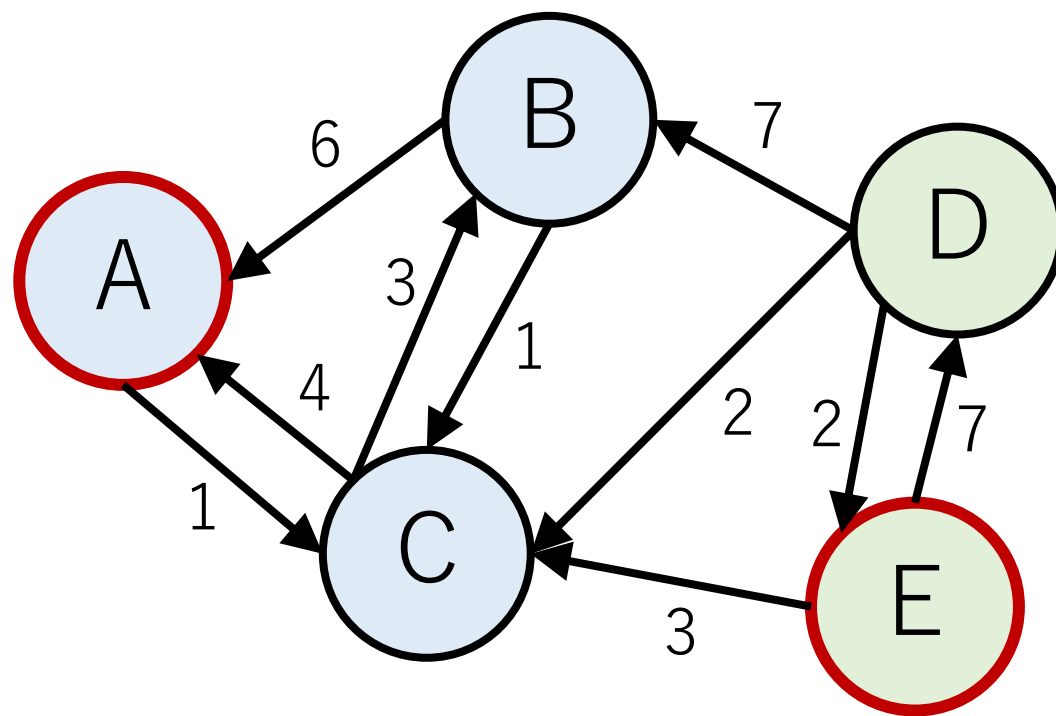
最大流・最小カット定理

もしこれが最小カットでないとすると，このs-tカットに至る前に別のところで $S \rightarrow V \setminus S$ にすでに行くパスがなくなっているはず。

それでもこのs-tカットに至っているとすれば，それはどこかで容量以上のフローを流してしまっていることになり，矛盾する。

最大流・最小カット定理

よってフォード・ファルカーソン法で得られたs-tカットは最小カットである。

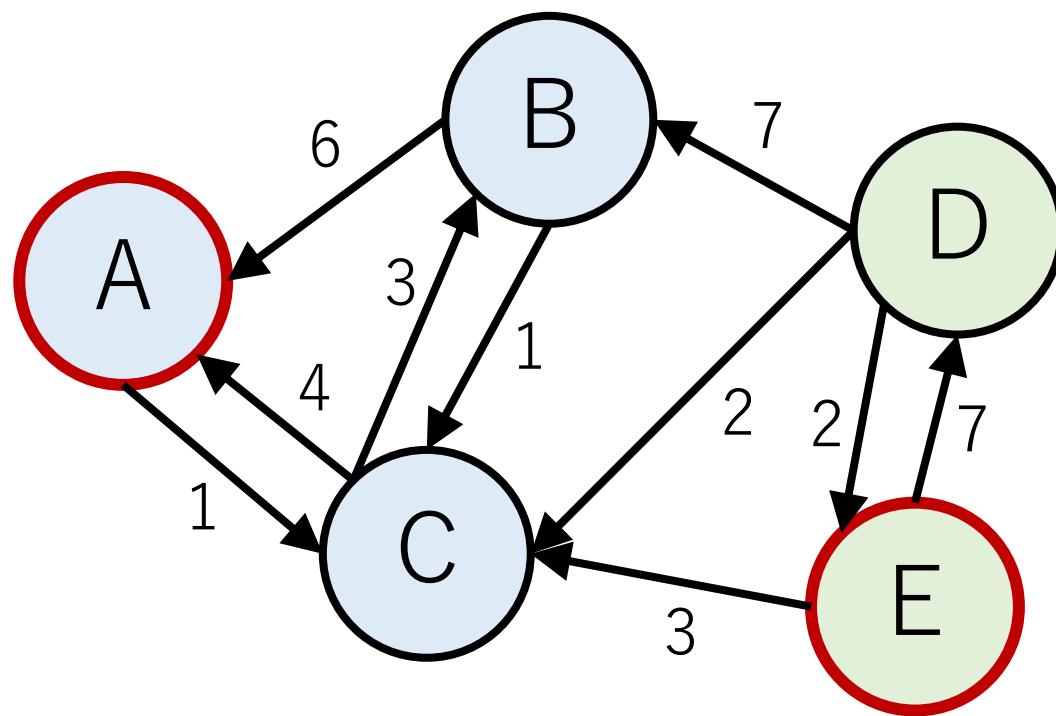


最大流・最小カット定理

この最小カットにおいても、

$$f \leq [S \text{ から } V \setminus S \text{ への流入量}] \leq U_{\min}(S \rightarrow V \setminus S)$$

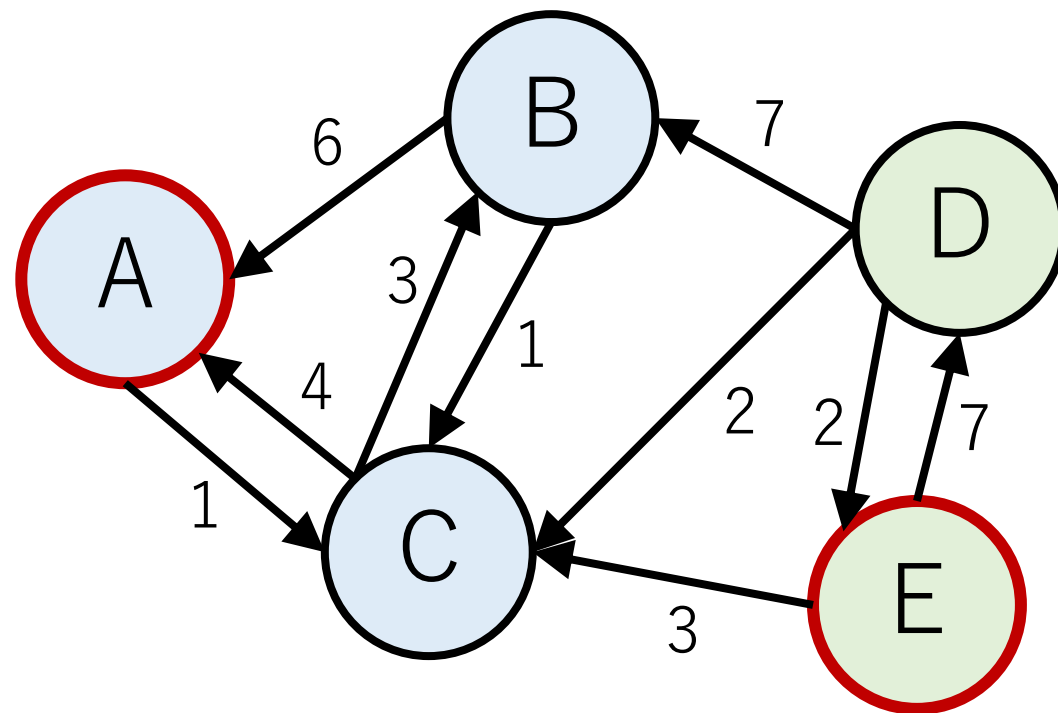
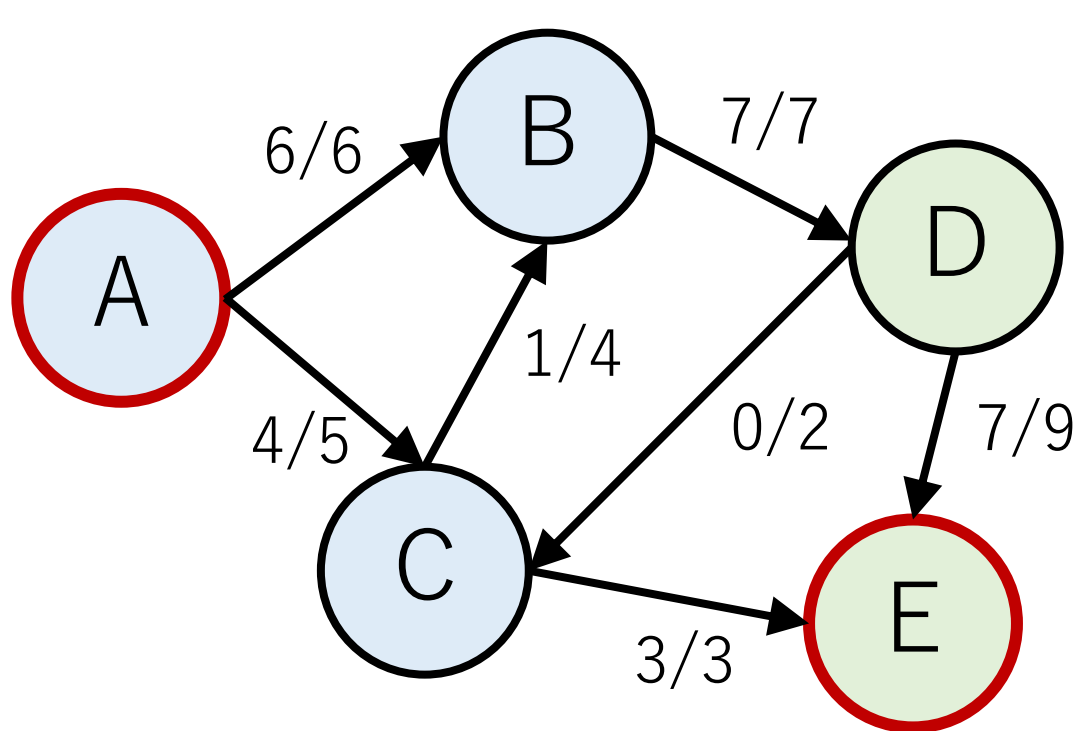
は成立する。



最大流・最小カット定理

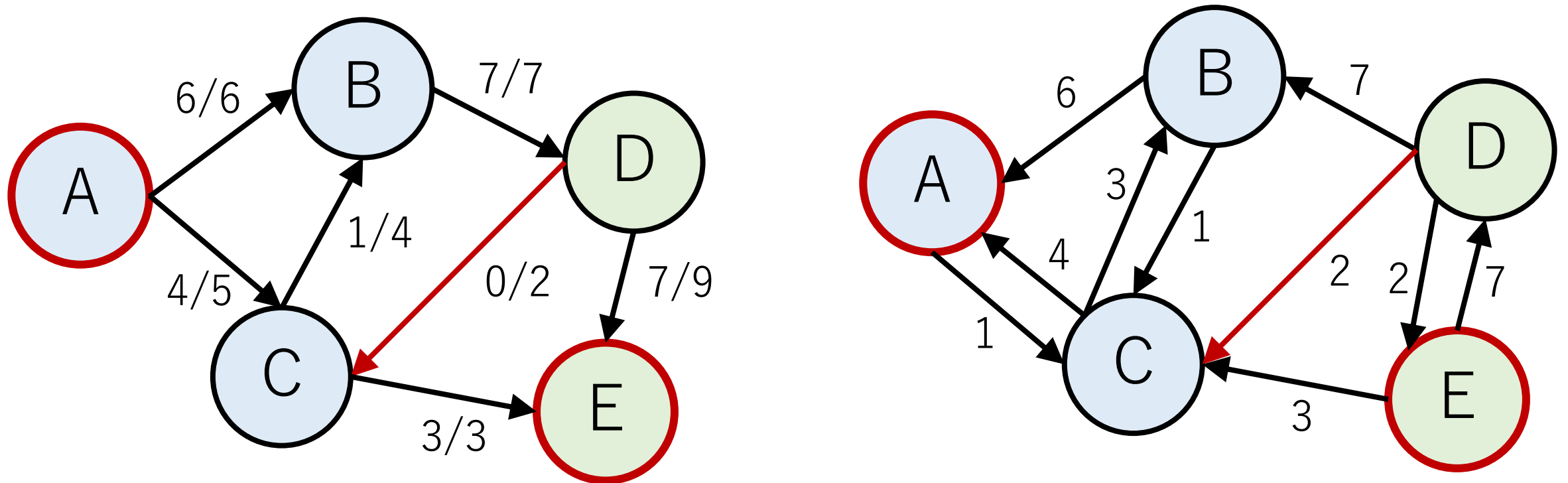
さらに、この不等式を詳しく見ていこう。

$f = [SからV \setminus Sへの流入量] - [V \setminus SからSへの流入量]$ であることも思い出しておく。



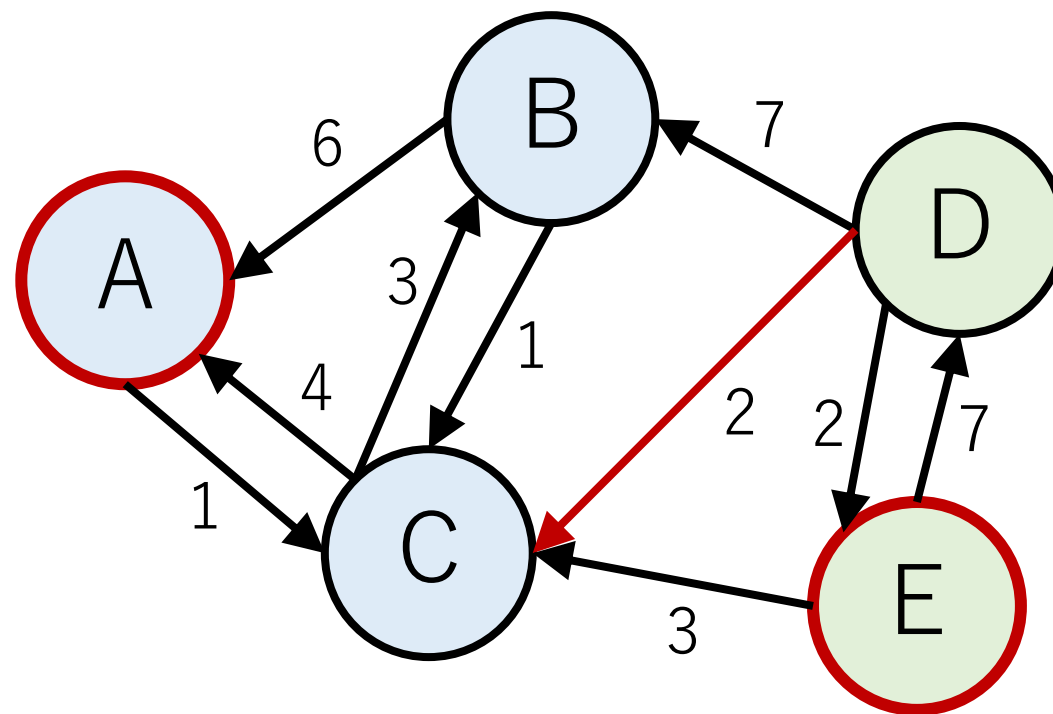
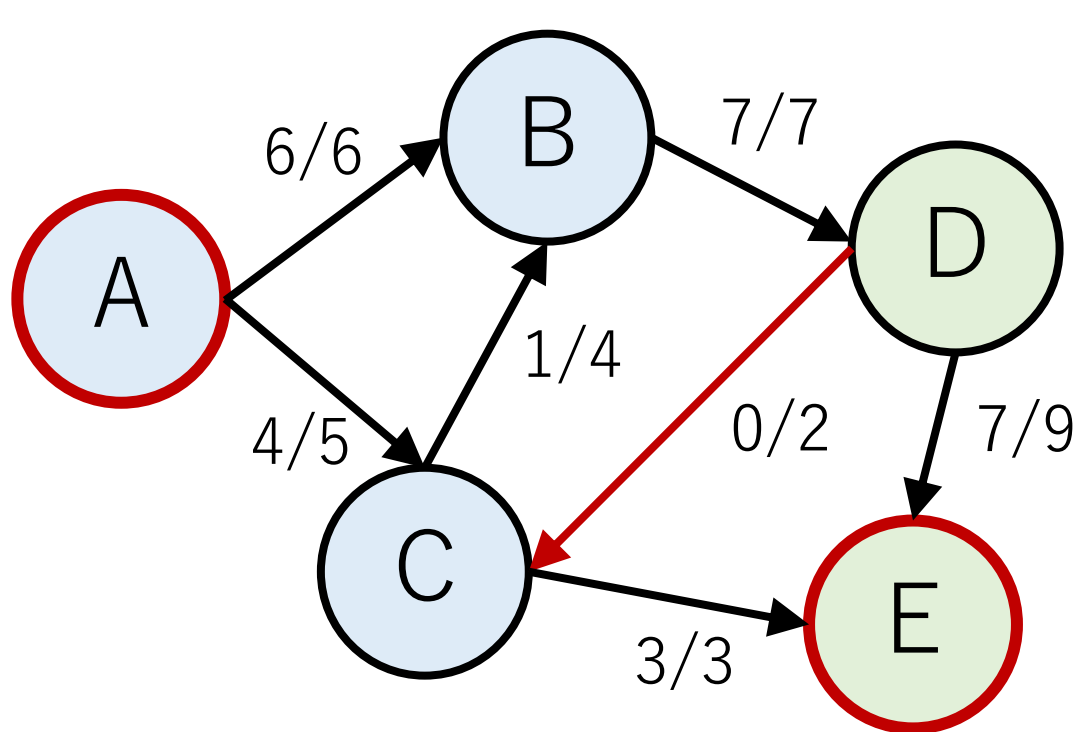
最大流・最小カット定理

$V \setminus S$ から S に向かう辺のうち、元々与えられたグラフにあったものだけを考える。この場合、DからCの辺のみ。



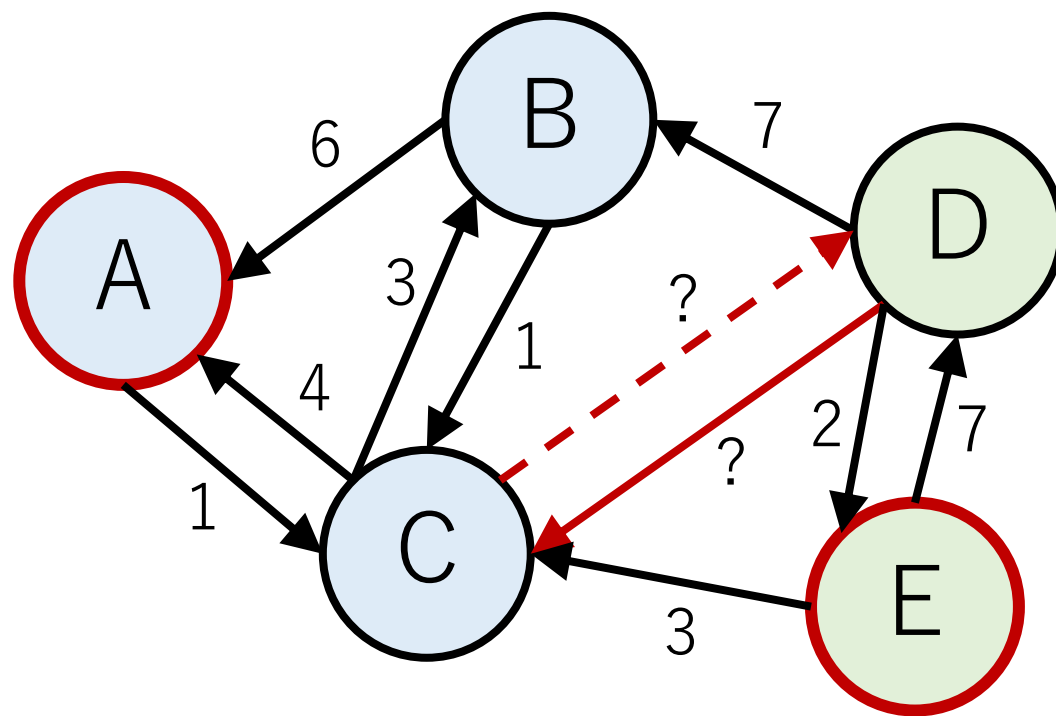
最大流・最小カット定理

この辺では仮想的な逆辺が出てきていない。つまりフローを全く流してない，ことになる。



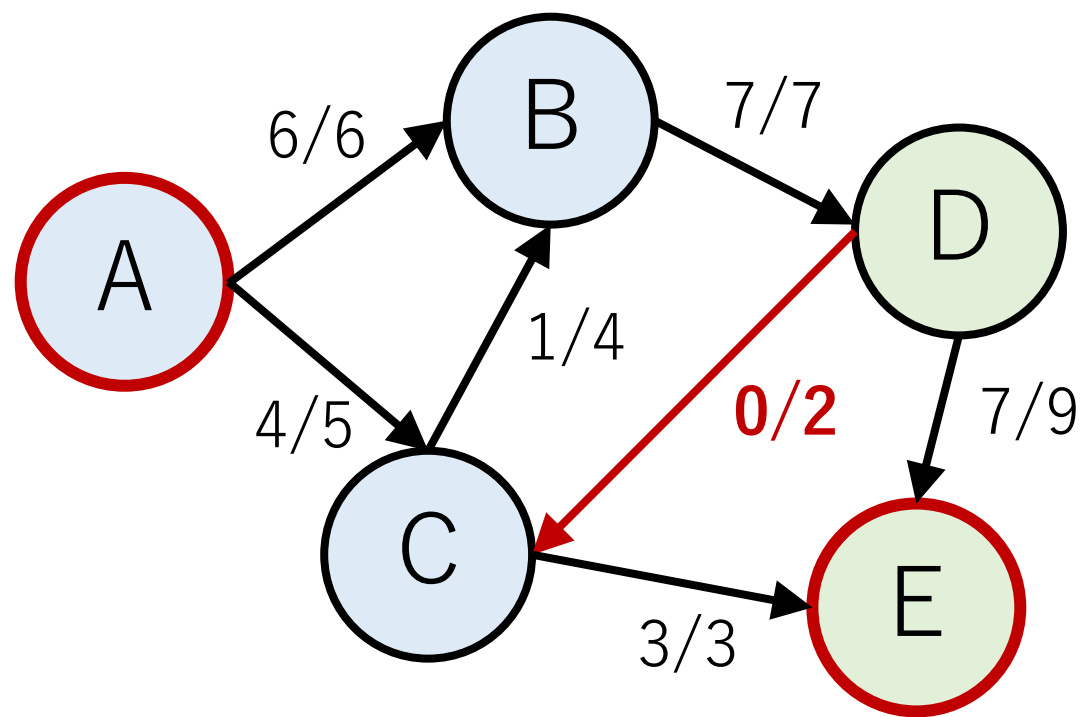
最大流・最小カット定理

もしこの逆辺が存在するならば，開始ノードから終点ノードに向けてさらにフローを流せるはず。（ここがカットにはなっていないはず。）



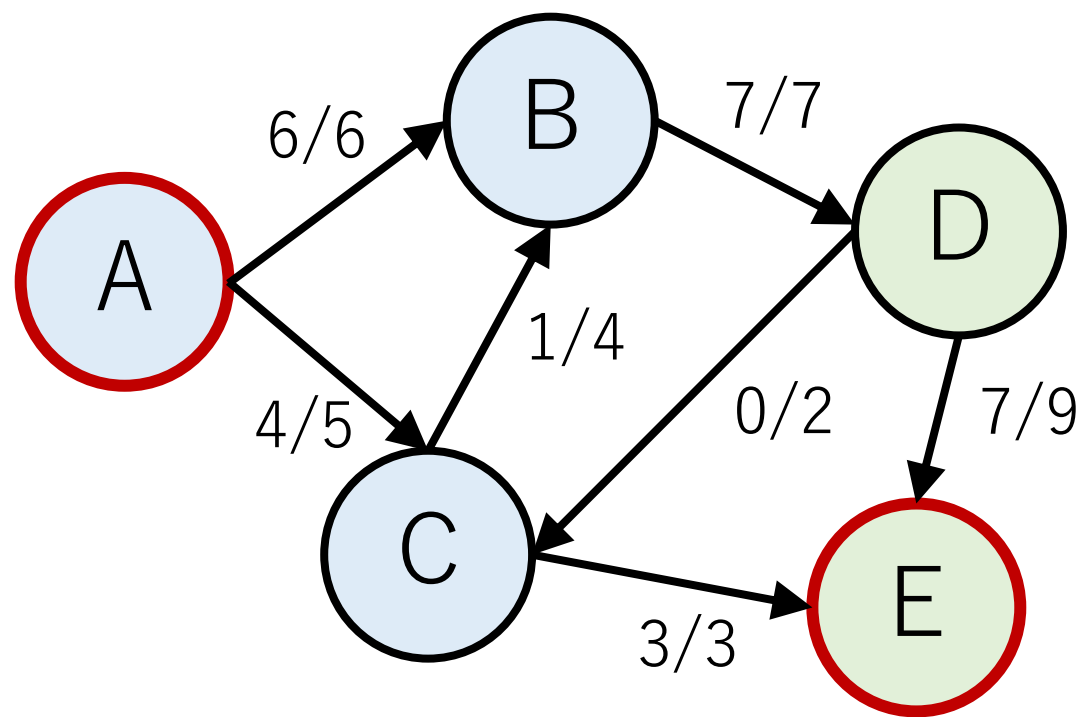
最大流・最小カット定理

つまり, $V \setminus S$ から S への流入量は0となっている.



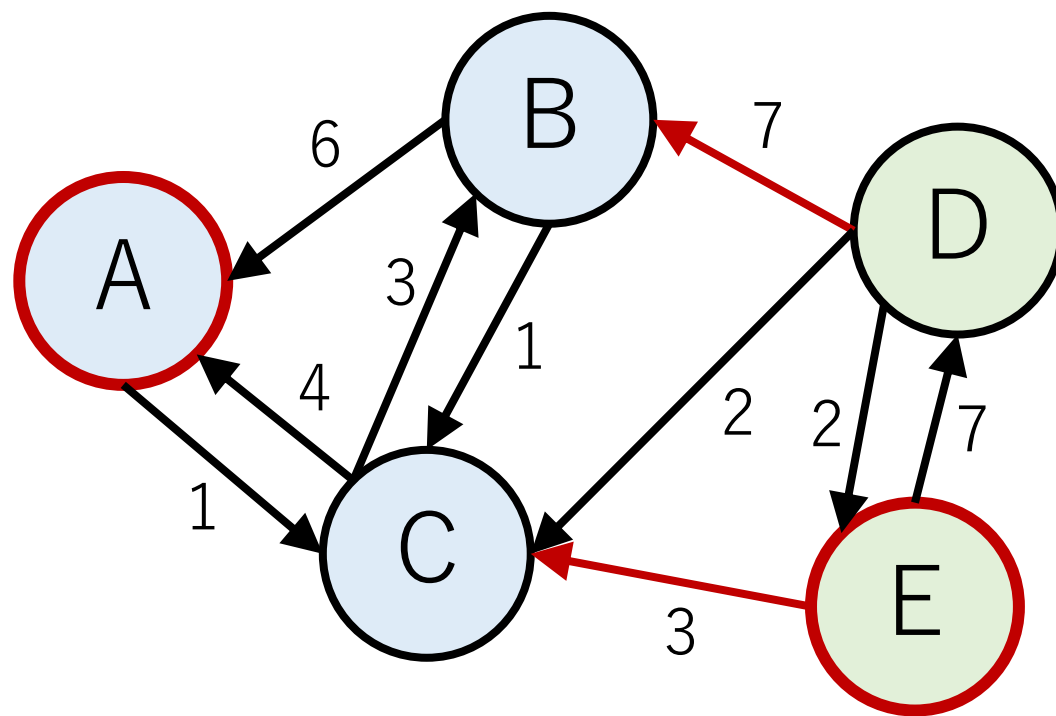
最大流・最小カット定理

よってこの時, $f = [SからV \setminus Sへの流入量]$ となっている.



最大流・最小カット定理

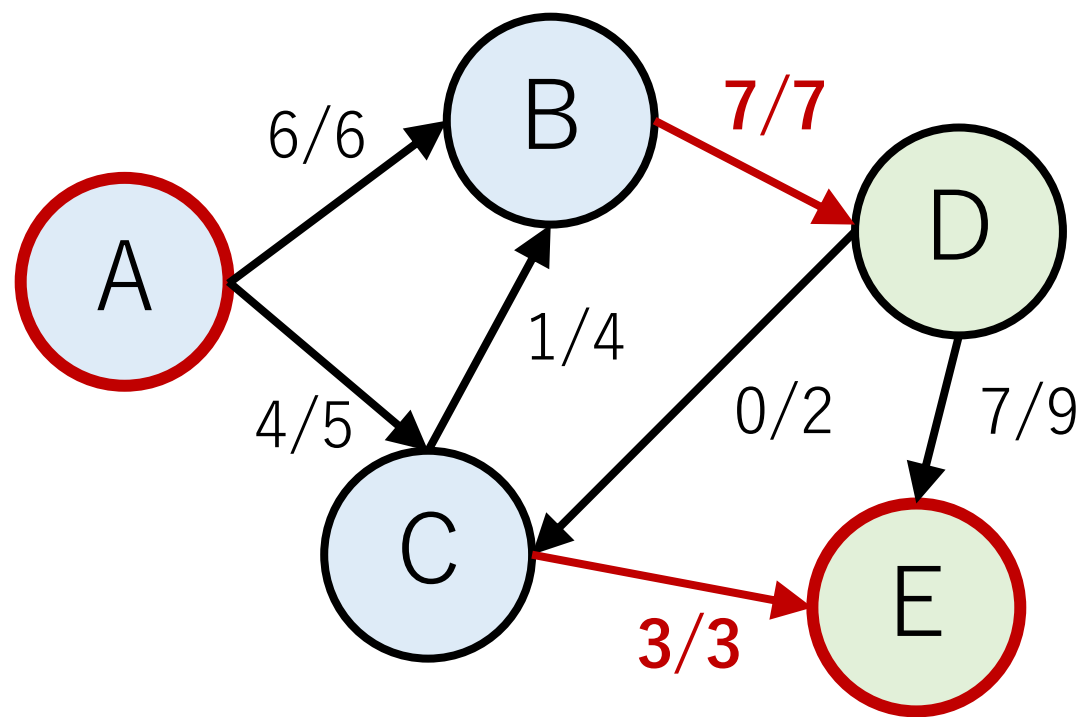
一方、フォード・ファルカーソン法の終了状態では S から $V \setminus S$ に向かう辺は存在していない。(もし存在していれば、さらに流せるはずなので、終了していないはず。)



最大流・最小カット定理

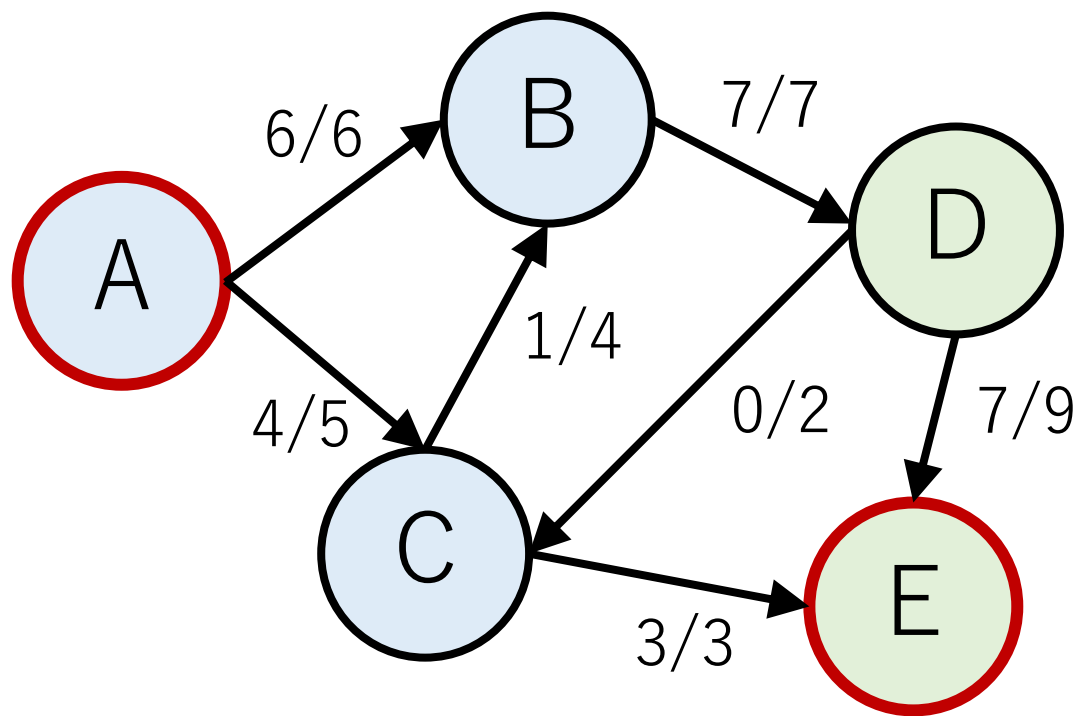
つまり、フローを最大限流したことで、 S から $V \setminus S$ にはもう行けない状態となっているからである。すなわち、

$$[S \text{ から } V \setminus S \text{ への流入量}] = U_{\min}(S \rightarrow V \setminus S)$$



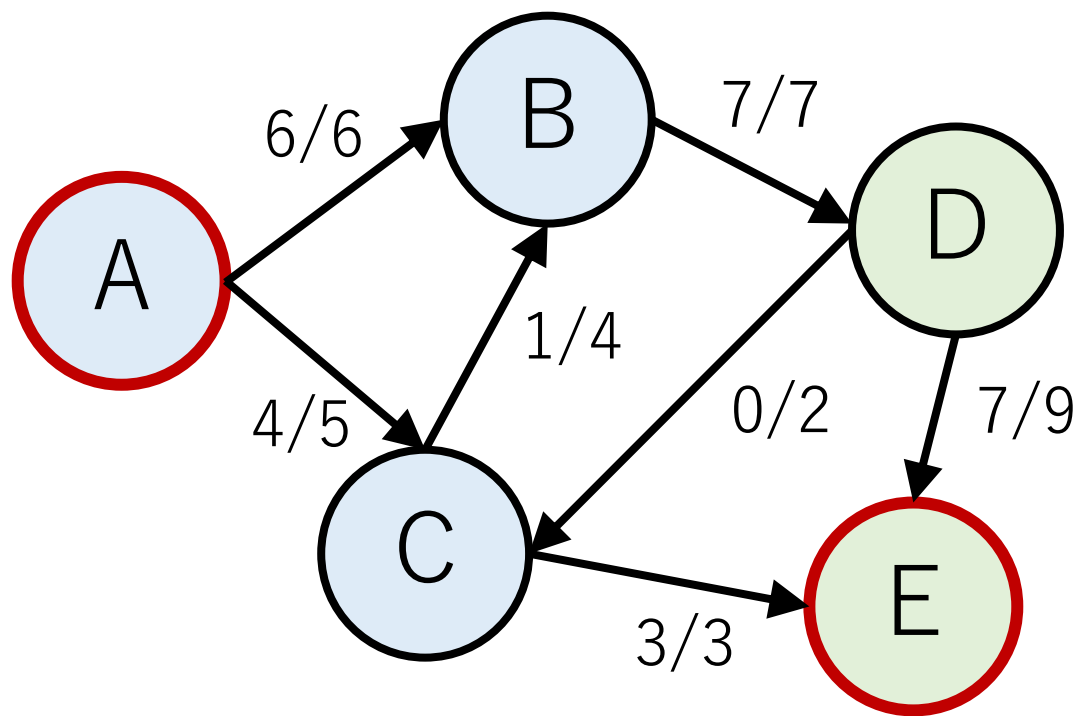
最大流・最小カット定理

すわなち, $f = [SからV \setminus Sへの流入量] = U_{min}(S \rightarrow V \setminus S)$
となる! つまり, フローは容量目一杯となっている.



最大流・最小カット定理

$f = U_{min}(S \rightarrow V \setminus S)$ となり，これは「与えられたグラフにおいて，最大流の流量はs-tカットにおける最小カットの容量に等しい」ことを表している。



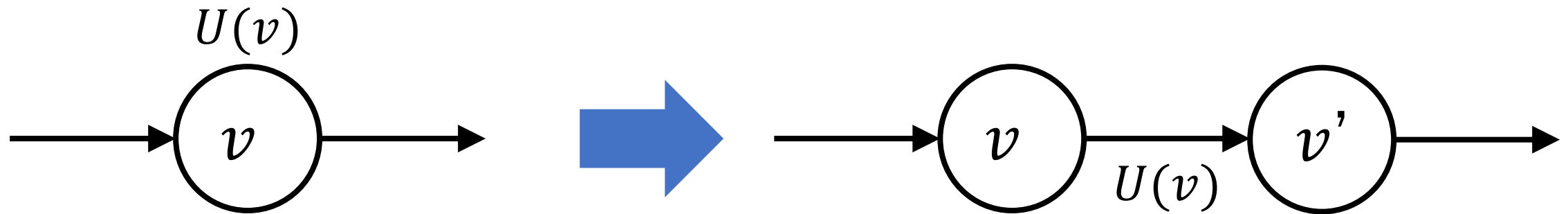
最小カットを愚直に探すと, , ,

s-tカットの分け方は $2^{|V|-2}$ あるので, $O(2^{|V|})$. . .

フォード・ファルカーソン法では $O(F|V|^2)$ や $O(F|E|)$ で
求められるため, かなり現実的になる.

ノードに容量がある場合

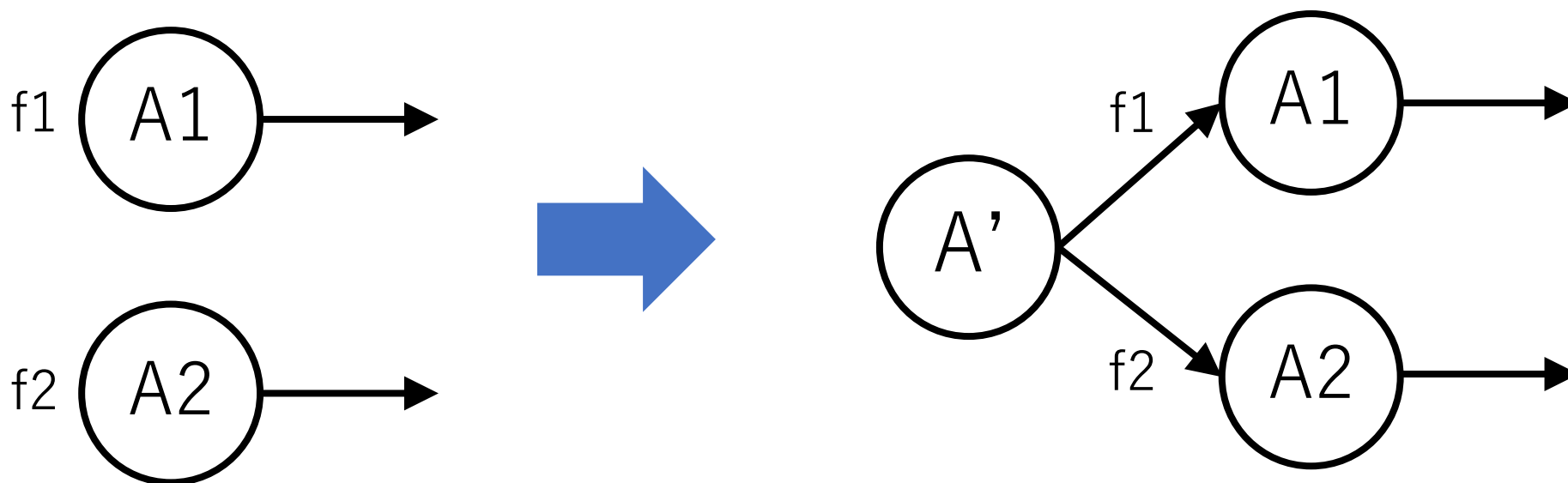
ノードを入ってくる方と出る方の2つに分けて, その間をノードの容量で結ぶ.



複数のsourceやsinkがある場合

各sourceに向かう辺を持つ新しい始点ノードを作り、その各辺にそれぞれの最大流出量を付与する。

sinkの場合も同様に新しい終点ノードを作れば良い。



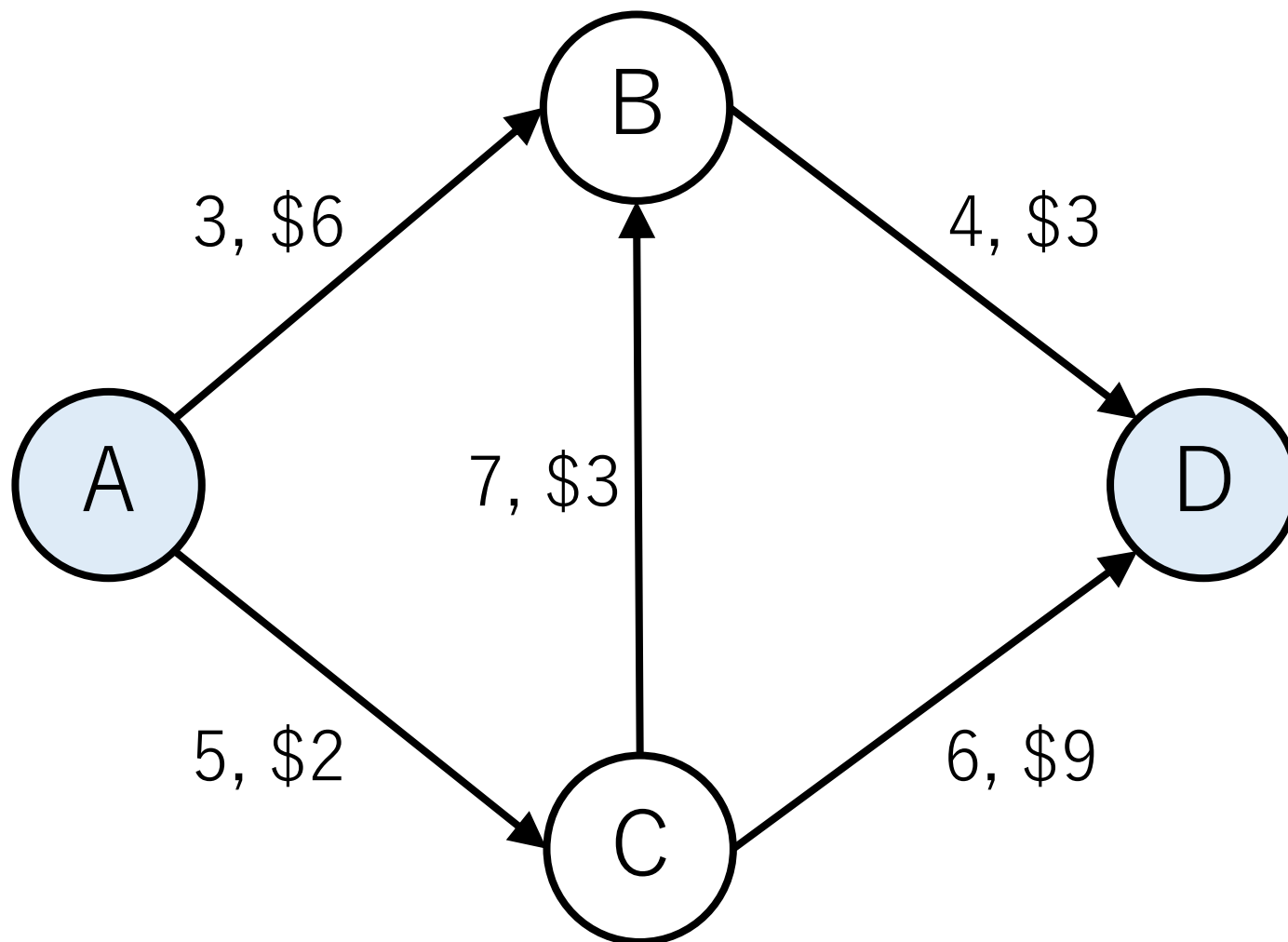
最小費用流

「ある流量 f を開始ノードから終了ノードに流す時、最小のコストはいくらか。」

経路に容量だけでなく、コスト（流量1に対する費用）が定義されている。

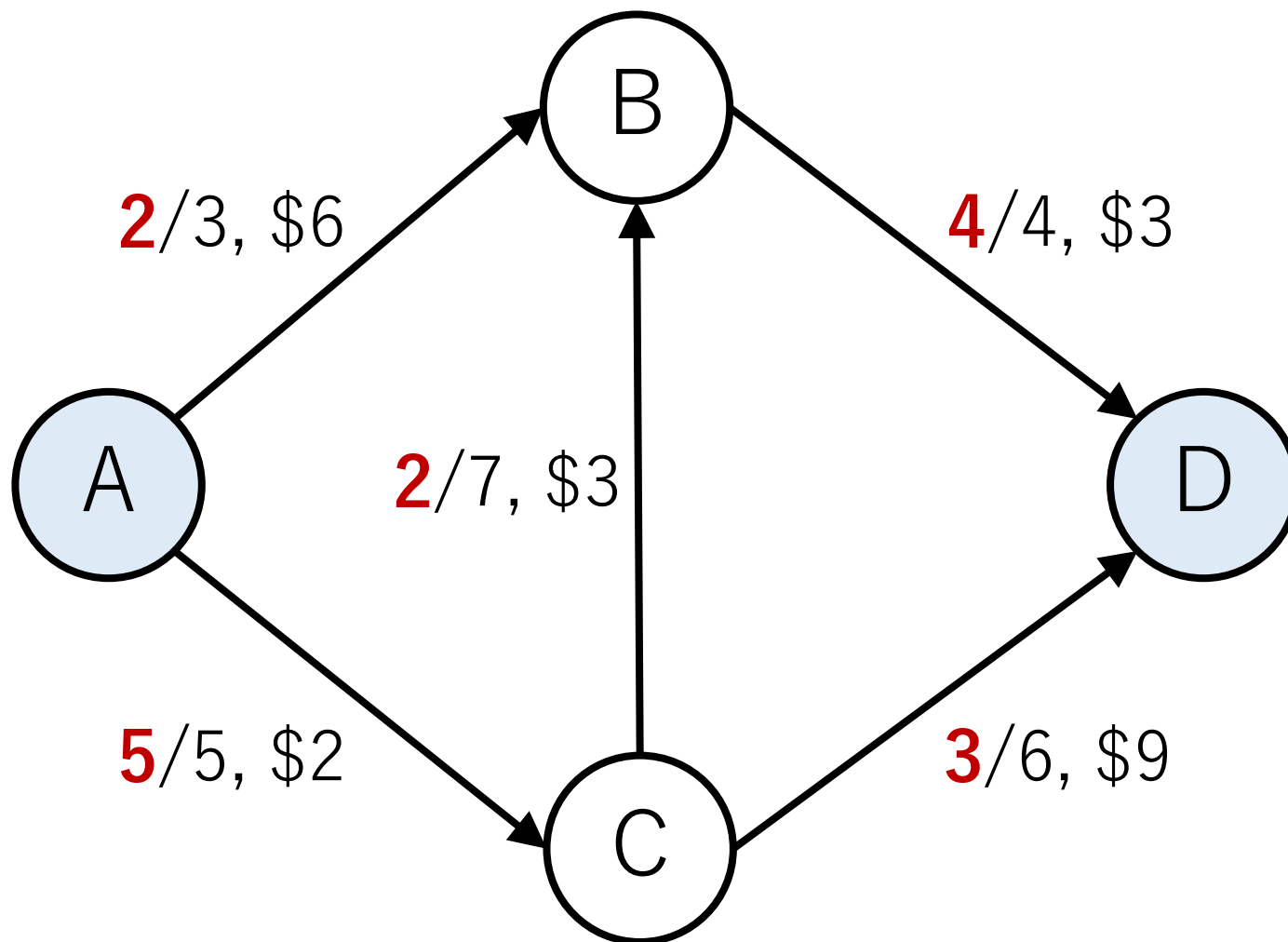
最小費用流

7流す時の最小費用は？



最小費用流

7流す時の最小費用は？ -> 67



プライマル・デュアル法

Primal-Dual

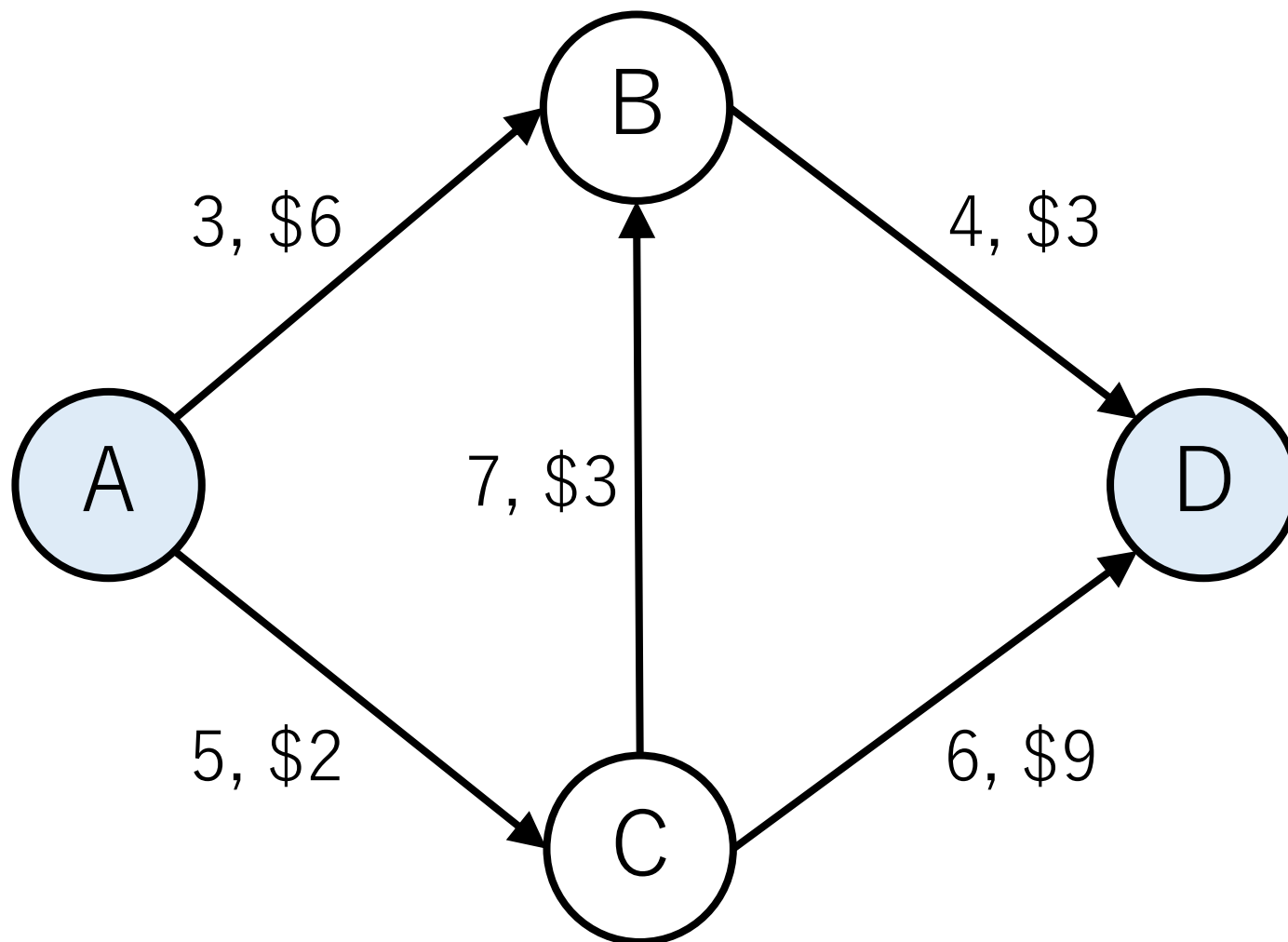
基本はフォード・ファルカーソン法と同じ。逆方向の経路も考える。

経路の探索において、その時点での最小コストの経路を求めて、そこに流せるだけ流す。

最短経路はベルマン・フォード法などを使う。
(負の経路が存在するため)

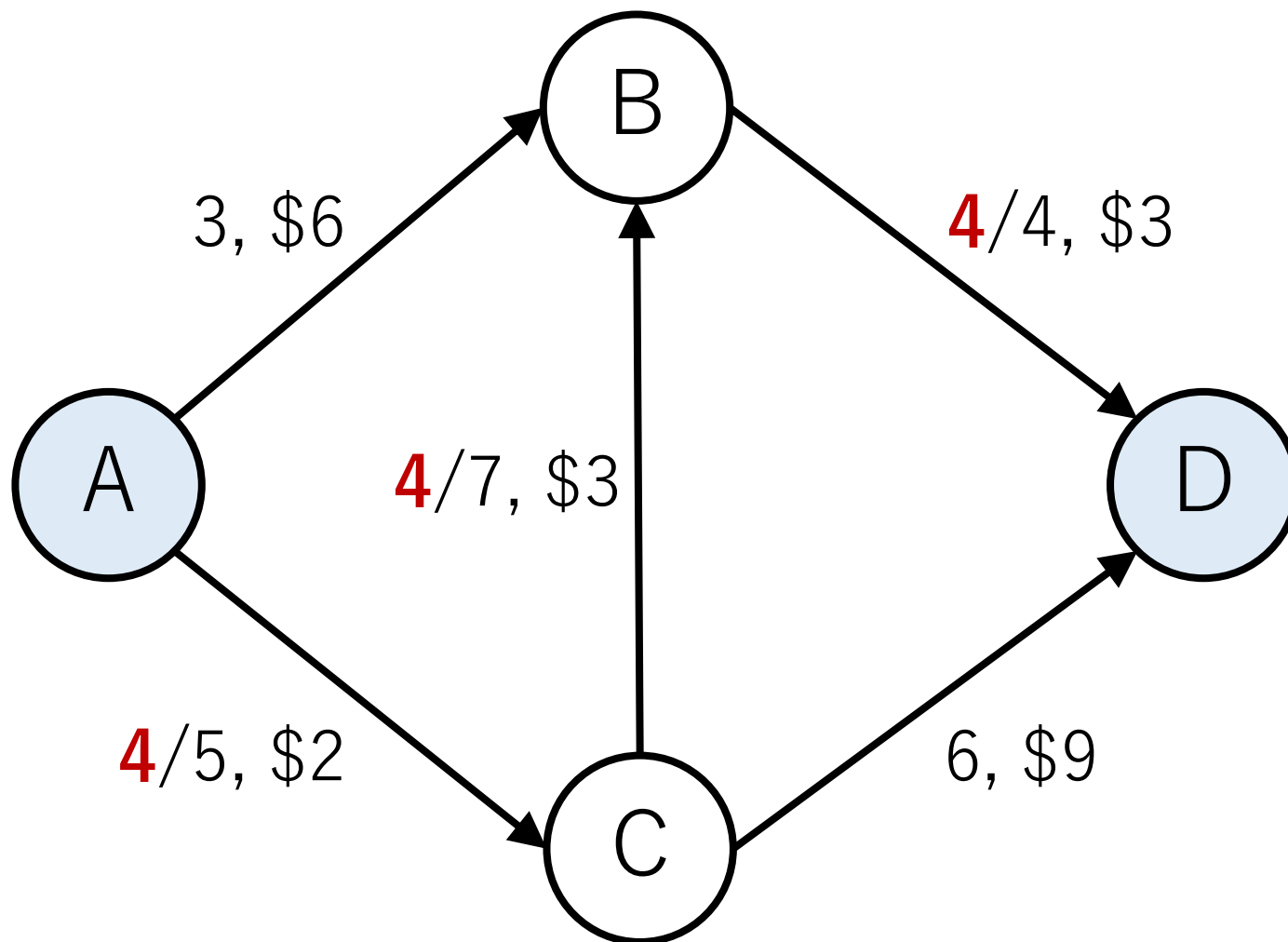
最小費用流の求め方例

7流す時の最小費用は？



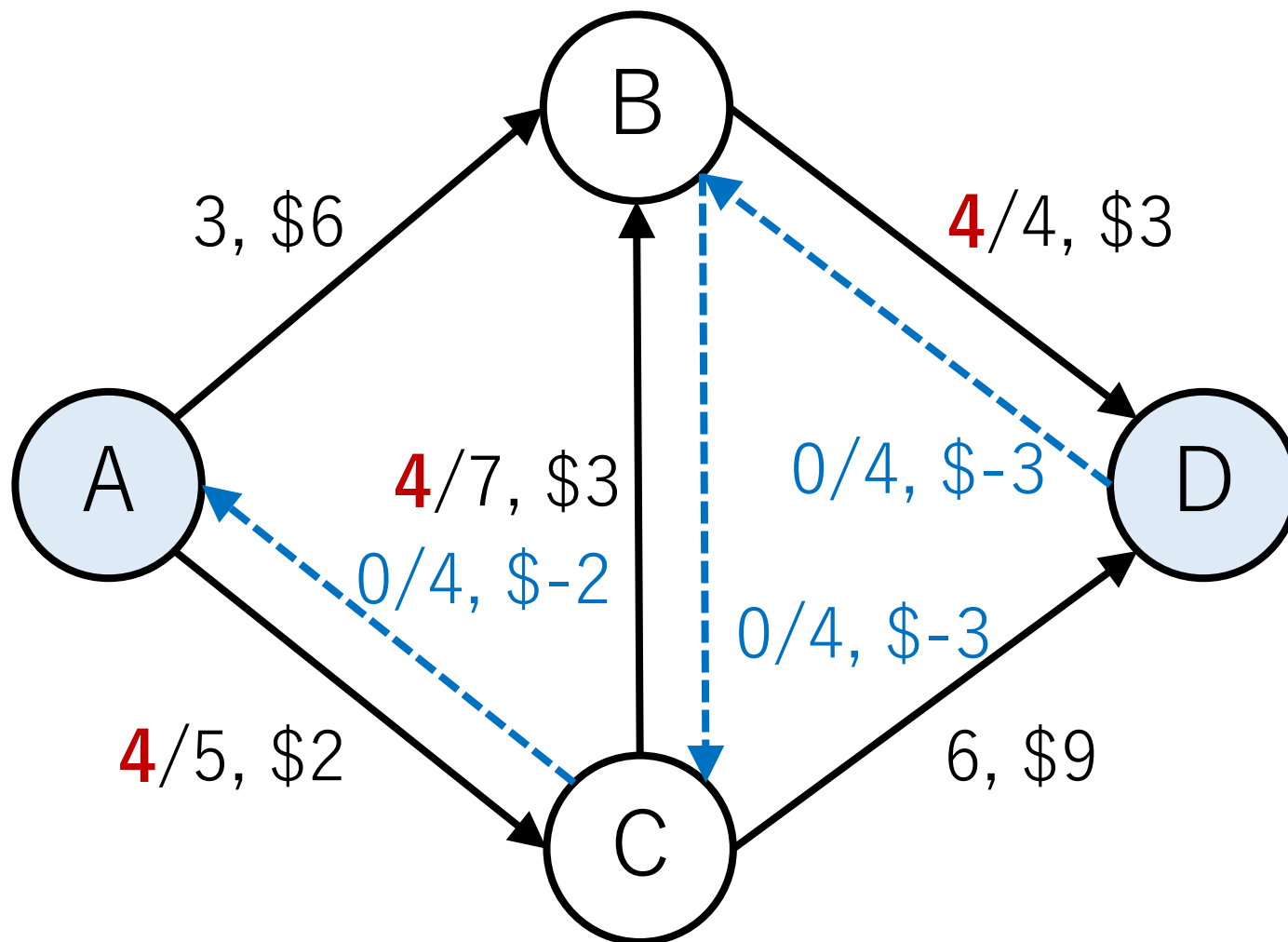
最小費用流

A->C->B->Dが最小費用経路\$8/流量で4流せる。



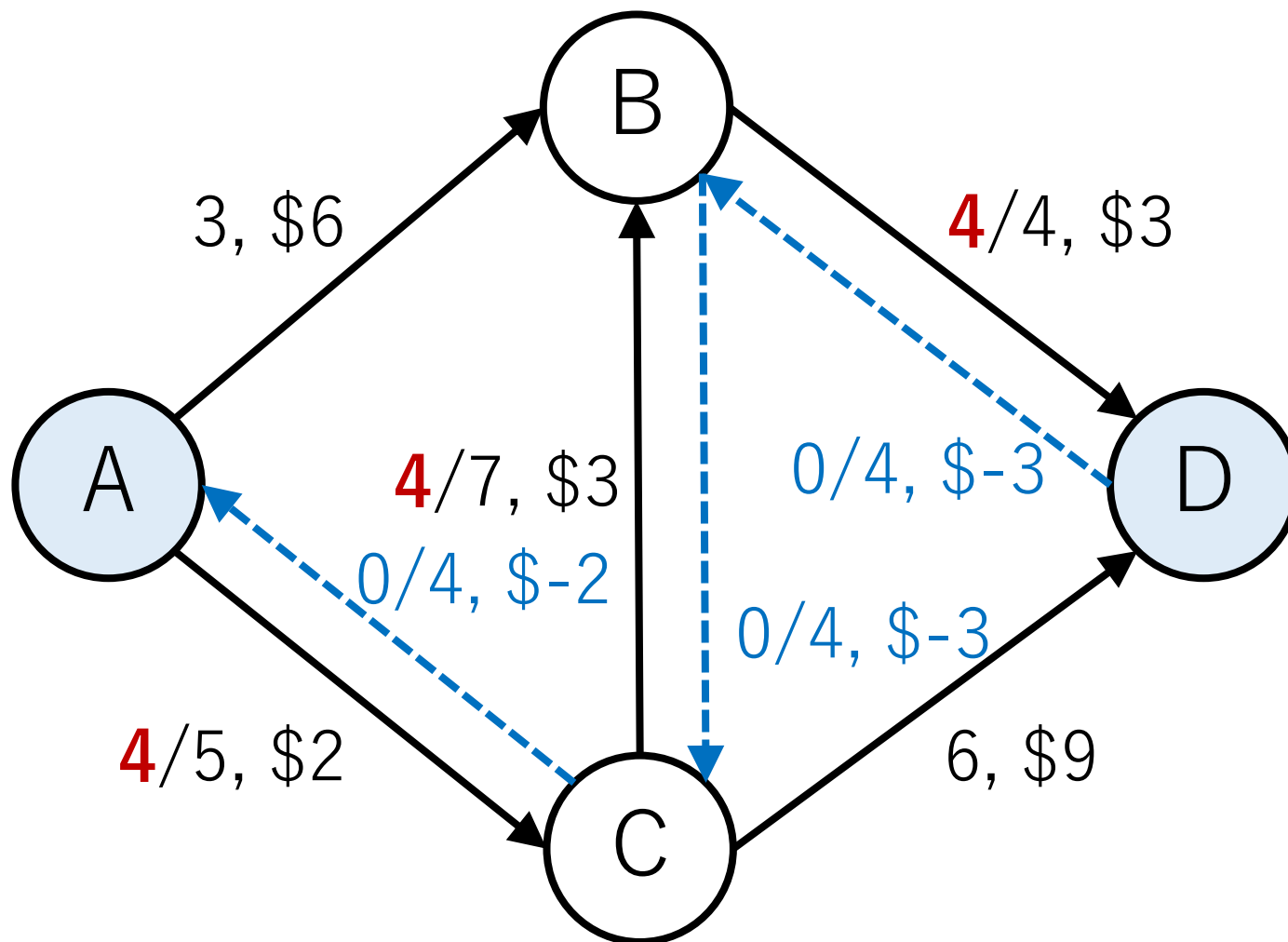
最小費用流

逆の経路を設定する。



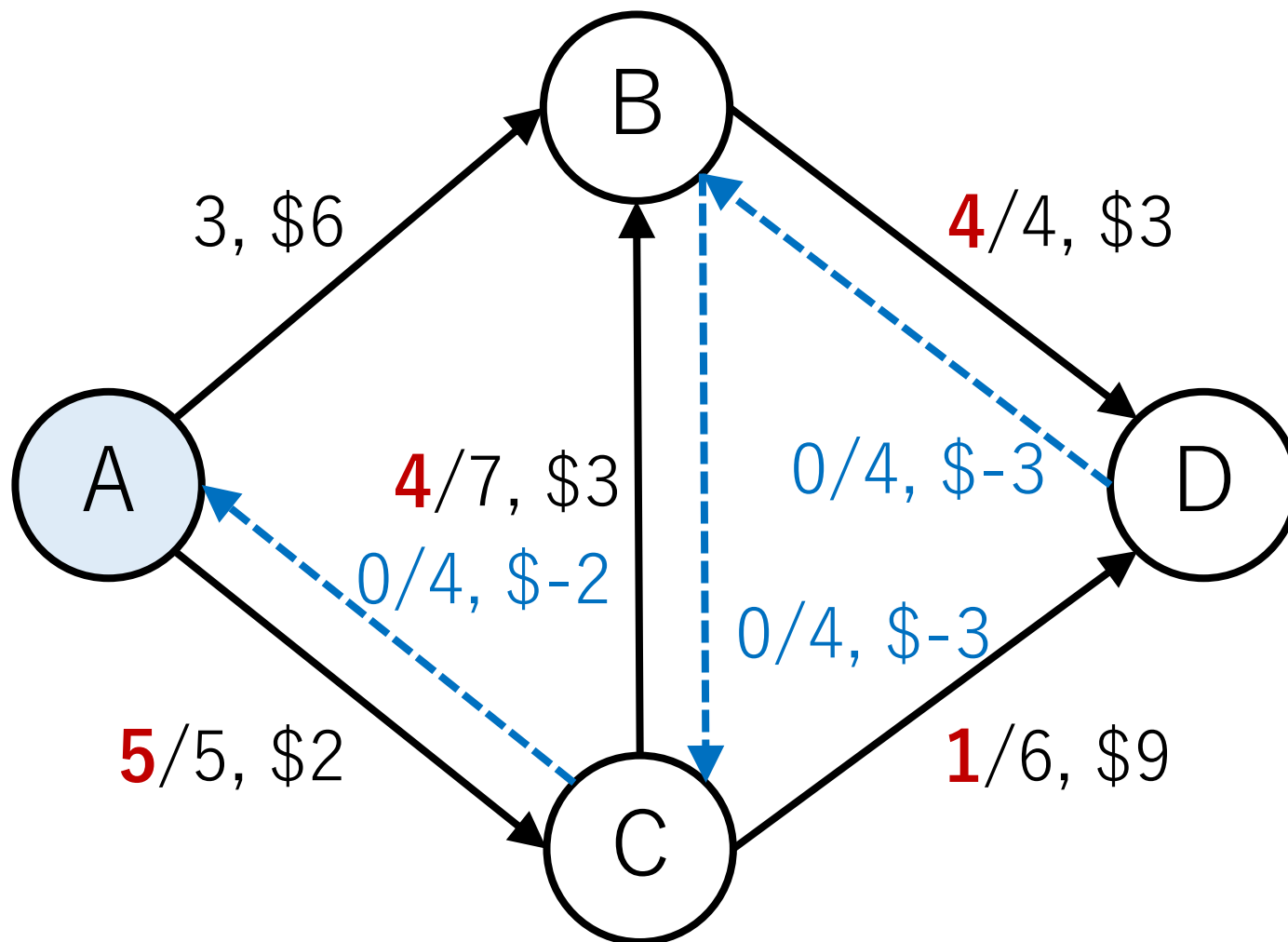
最小費用流

A->C->Dは1流せて\$11/流量. A->B->C->Dは3流せて\$12/流量.



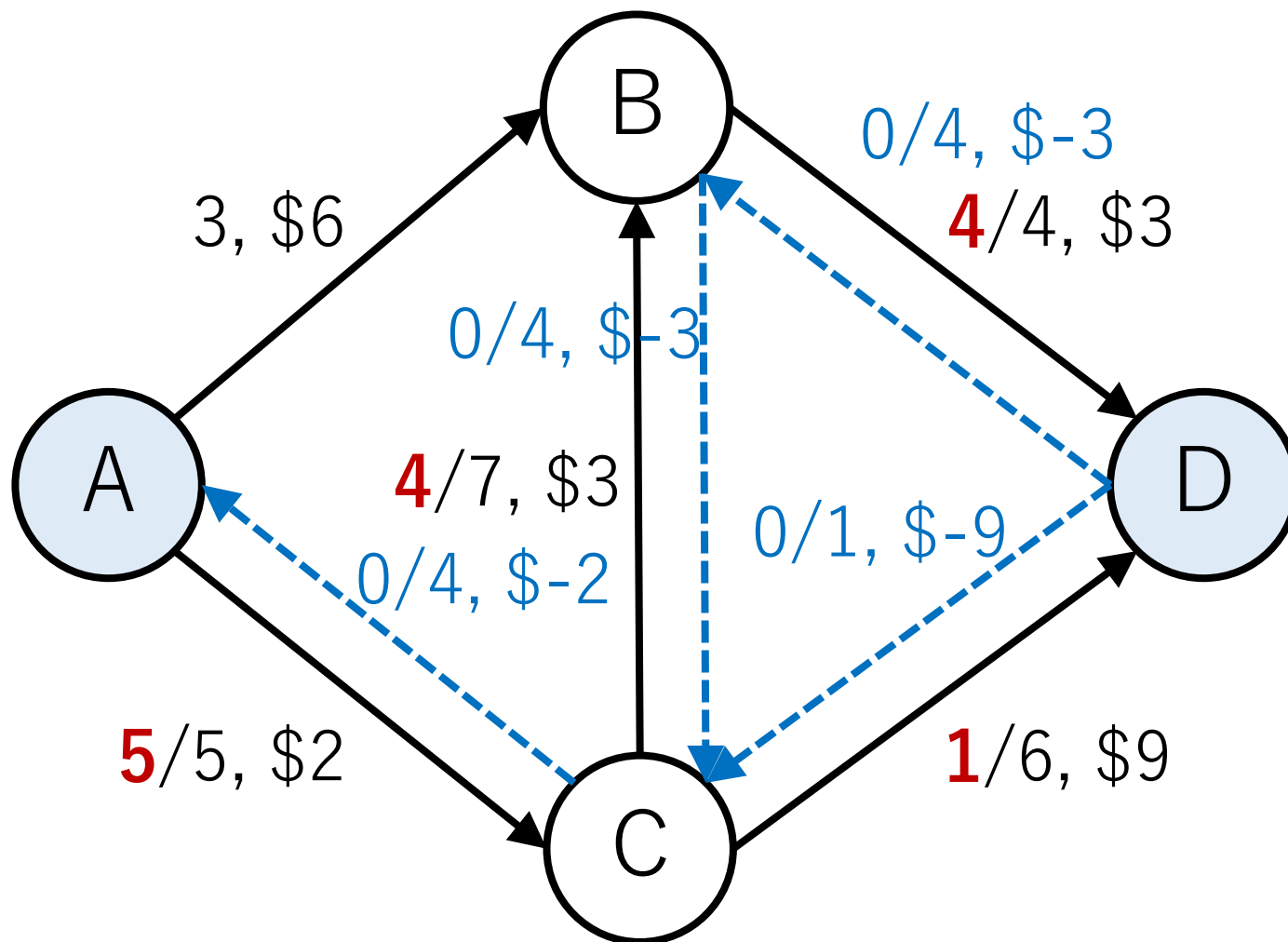
最小費用流

よって, A->C->Dに1流す.



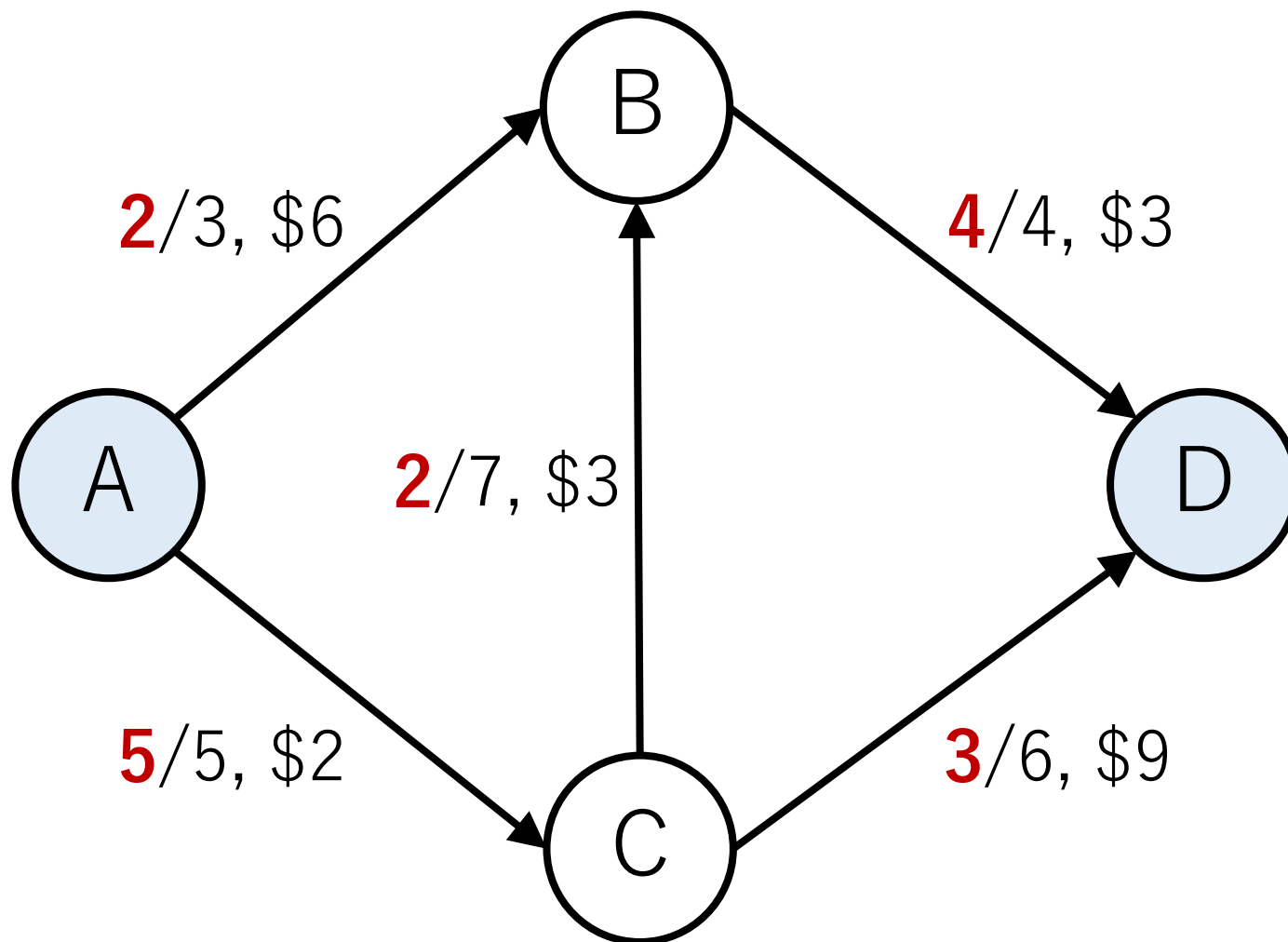
最小費用流

次にA->B->C->Dに2を流す。これは $6-3+9=\$12/\text{流量}$ 。



最小費用流

逆経路の流れが全て相殺されることを確認し，終了．



プライマル・デュアル法の計算量

ノード数 $|V|$ ，エッジ数 $|E|$ ，設定流量を F とする．

最小コスト経路探索のためにはベルマン・フォード法を使うと $O(|V||E|)$ 必要．（隣接リストを使う場合）

さらに，流量を1ずつしか増やせないのが最悪のケース．

よって， $O(F|V||E|)$ ．

プライマル・デュアル法の計算量

ポテンシャルと呼ばれる工夫をするとダイクストラ法を使うことができる。

この場合、 $O(F|V|^2)$ 。ヒープを使うダイクストラ法を使えば、さらに計算量を減らせる。

まとめ

最大流問題

フォード・ファルカーソン法

最小費用流問題

プライマル・デュアル法

今回紹介したものの以外のアルゴリズムもありますが、この2つが最もよく紹介されるものかと思います。

まとめ

フローの問題はいろいろな応用があり，自在に使えるところとても強い武器になります（来週その1つをご紹介します．お楽しみに😊）．

まずは，フォード・ファルカーソン法の考え方，実装をしっかりと消化してください．その上で，他のフローの問題や，最大流・最小カット定理の理解に進むと良いかと思います．

コードチャレンジ：基本課題#11 [2点]

スライドを参考にして，隣接リスト形式の入力を直接扱うフォード・ファルカーソン法を実装してください。

隣接リストを隣接行列に変換して，スライドの実装例を使うのはなしとします。

今回の基本課題はこの1問のみです。

コードチャレンジ：Extra課題#11 [3点]

最小費用流問題に関する問題.

今回が最後のExtra課題になります. 😊