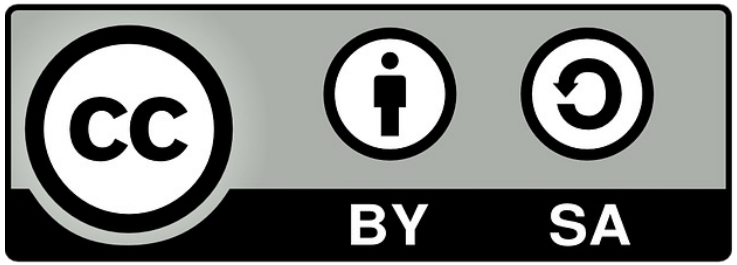
Algorithms (2024 Summer)
#12：グラフアルゴリズム3
（最大流問題）

矢谷 浩司
東京大学工学部電子情報工学科



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

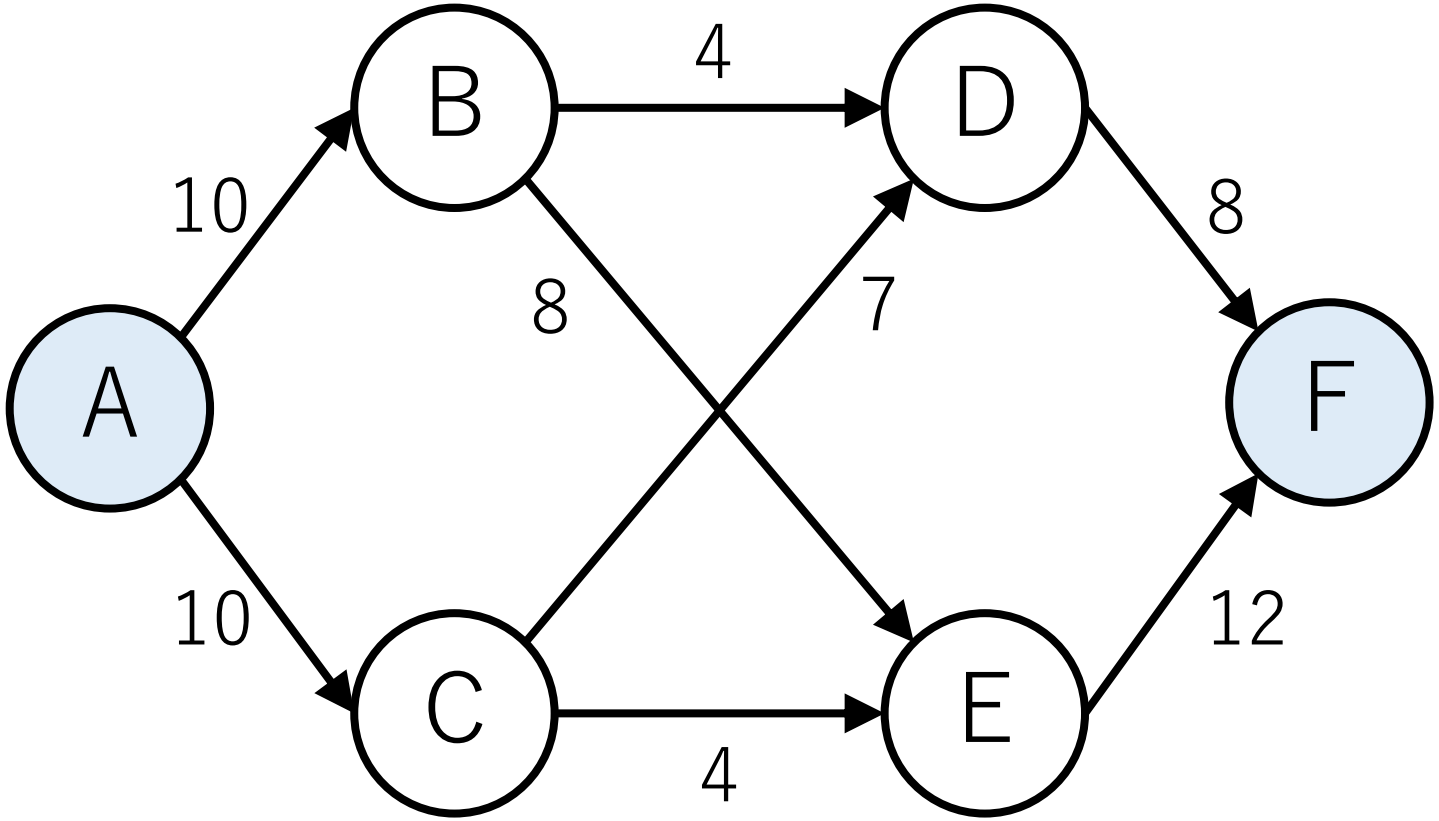
始点からはフローが生まれ（source），終点ではフローがなくなる（sink）.

始点と終点ノード以外ではフローが増減しない.

今日と次回の授業では，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

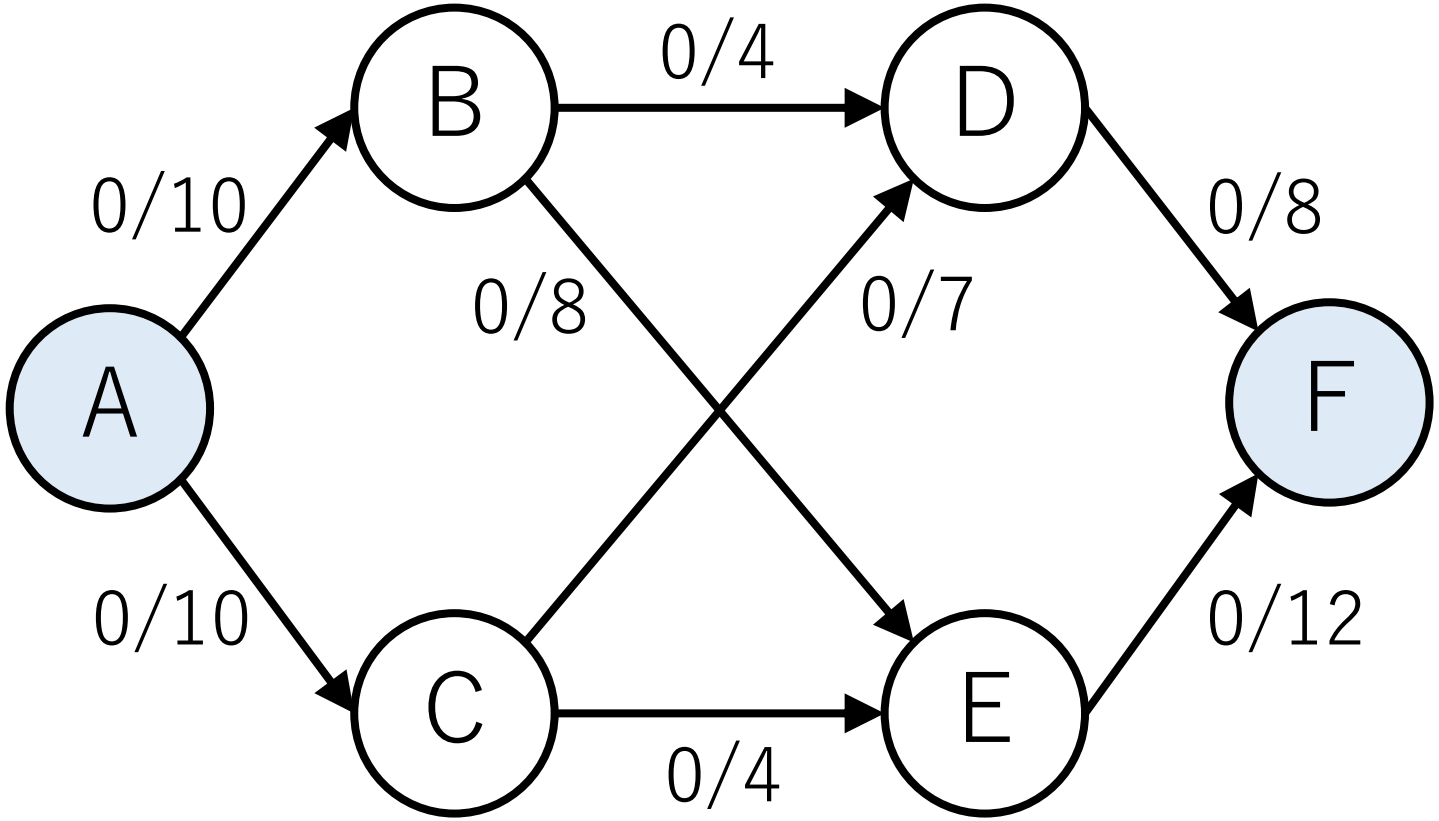
A->Fに至る経路をDFSで見つける.

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.

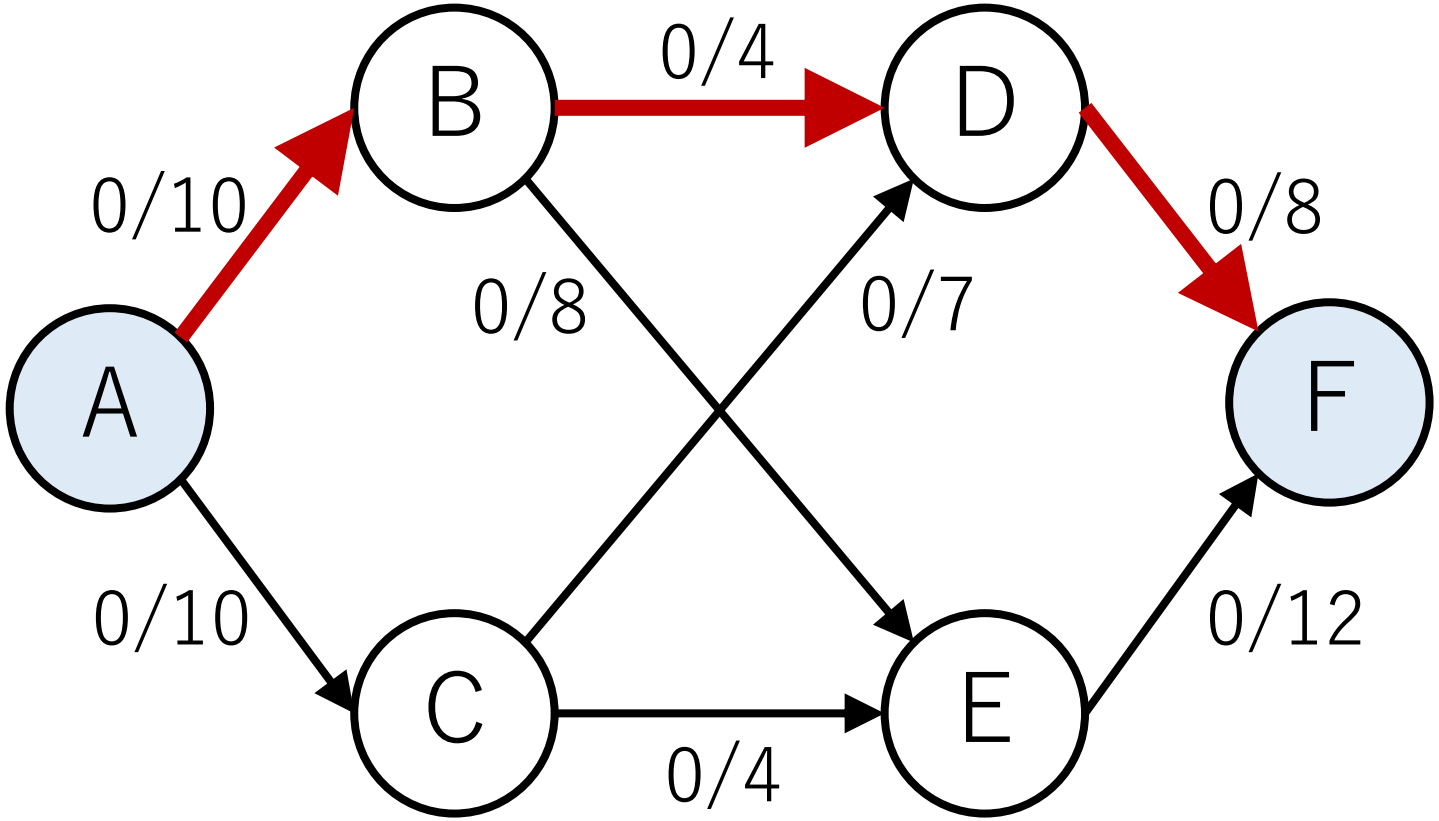
貪欲法による例

AからDFSスタート.



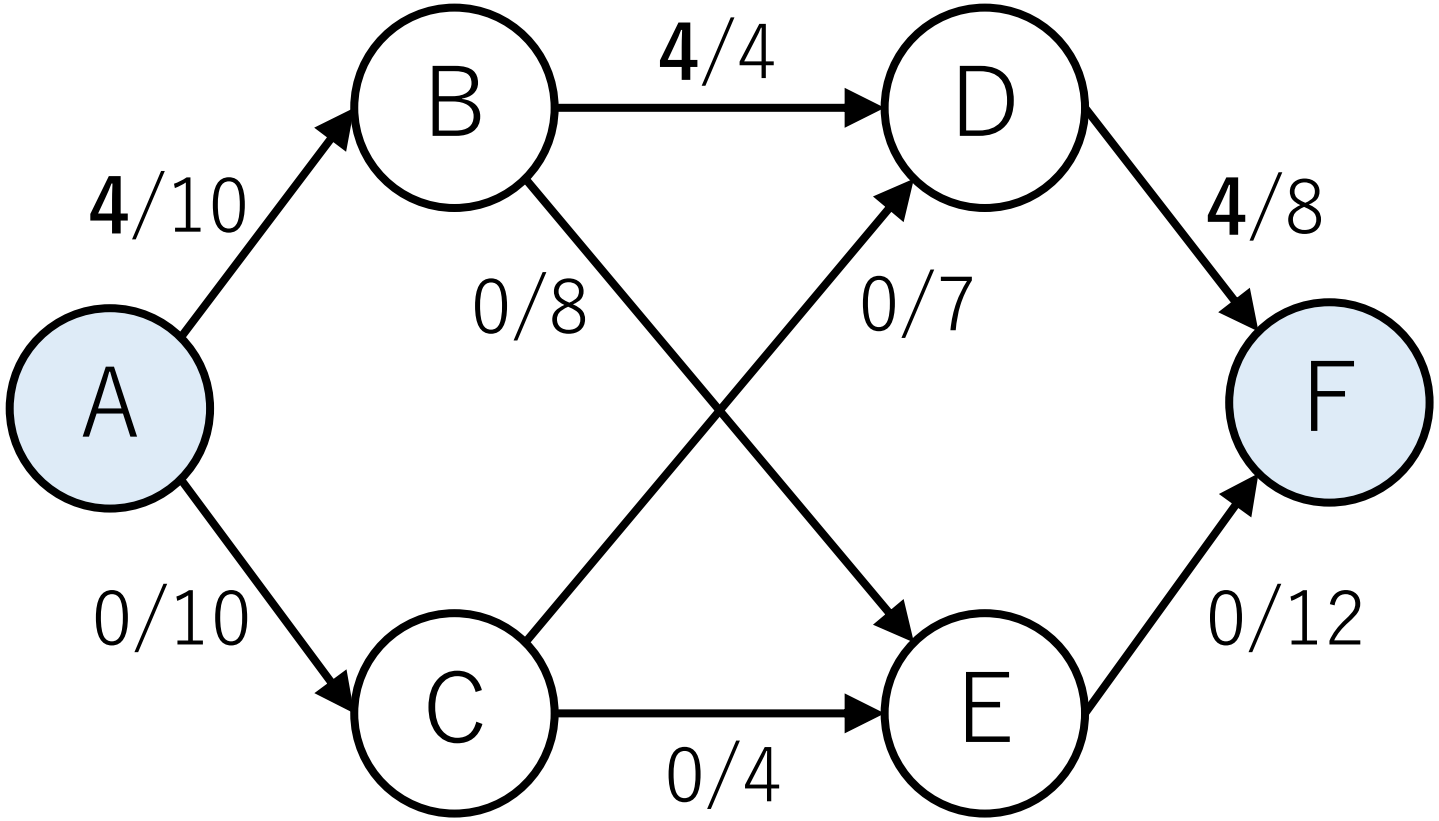
貪欲法による例

A->B->D->Fがみつかる．ここは4流せる．



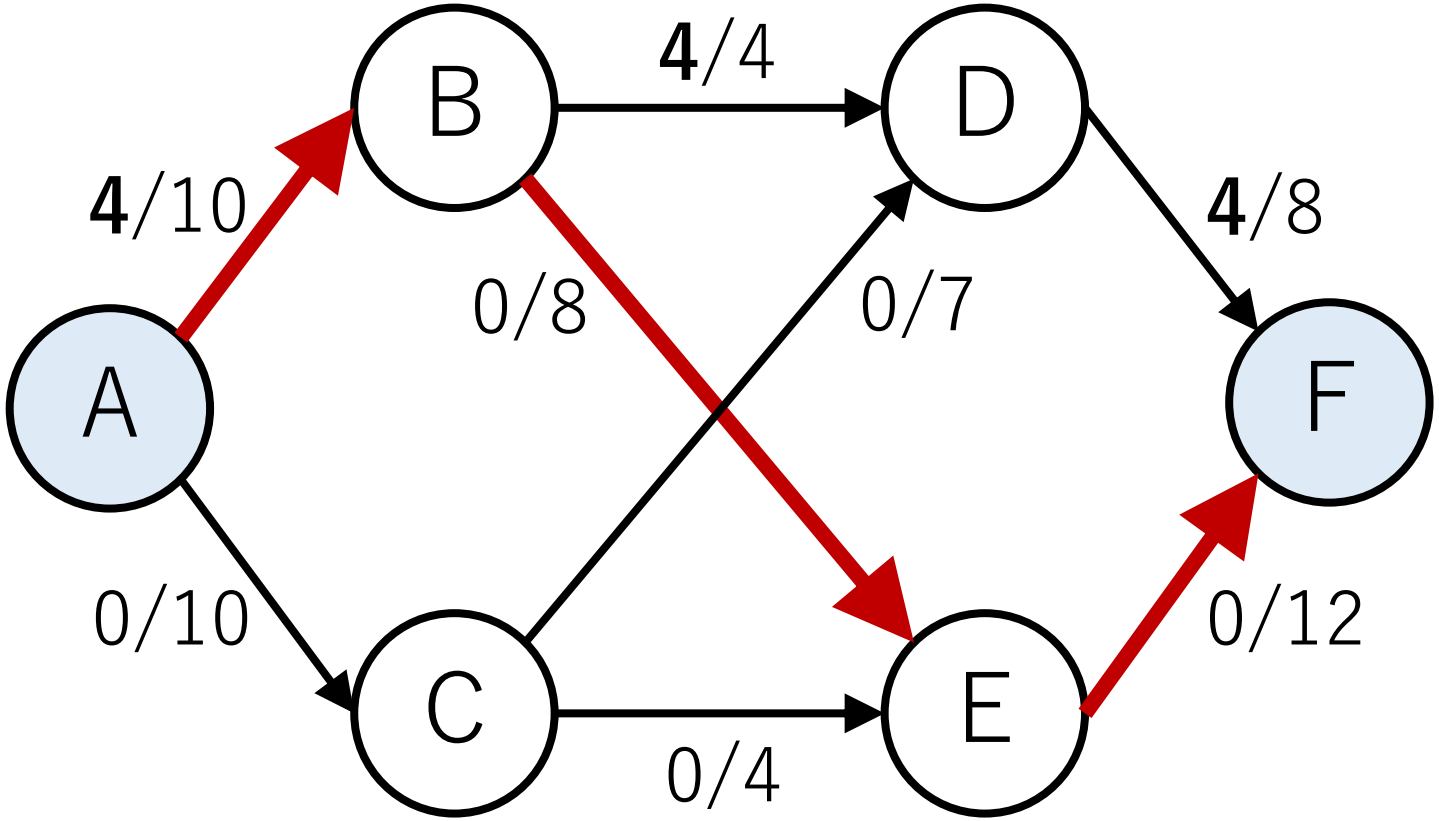
貪欲法による例

A->B->D->Fに4流す.



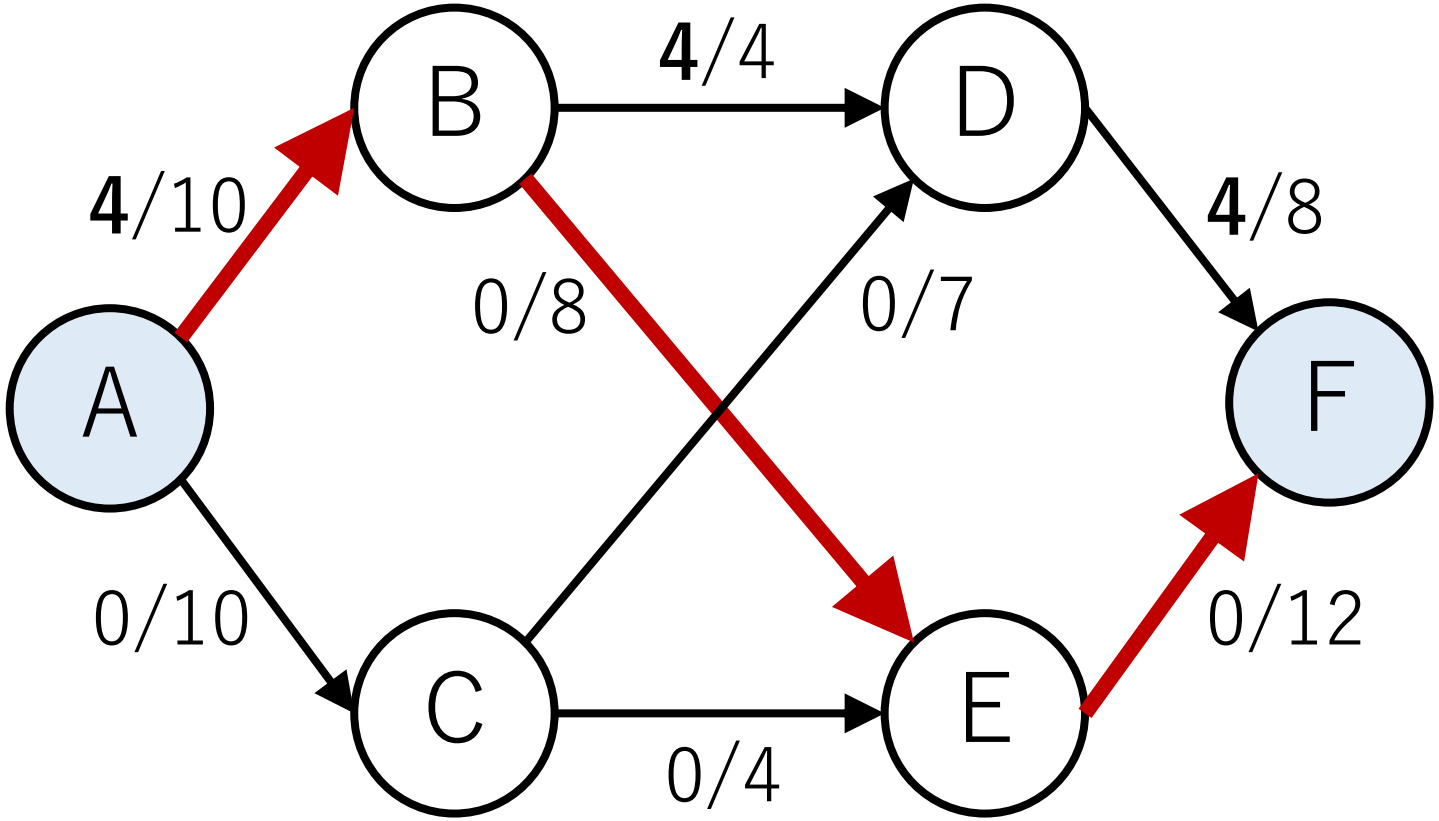
貪欲法による例

次にA->B->E->Fが見つかる.



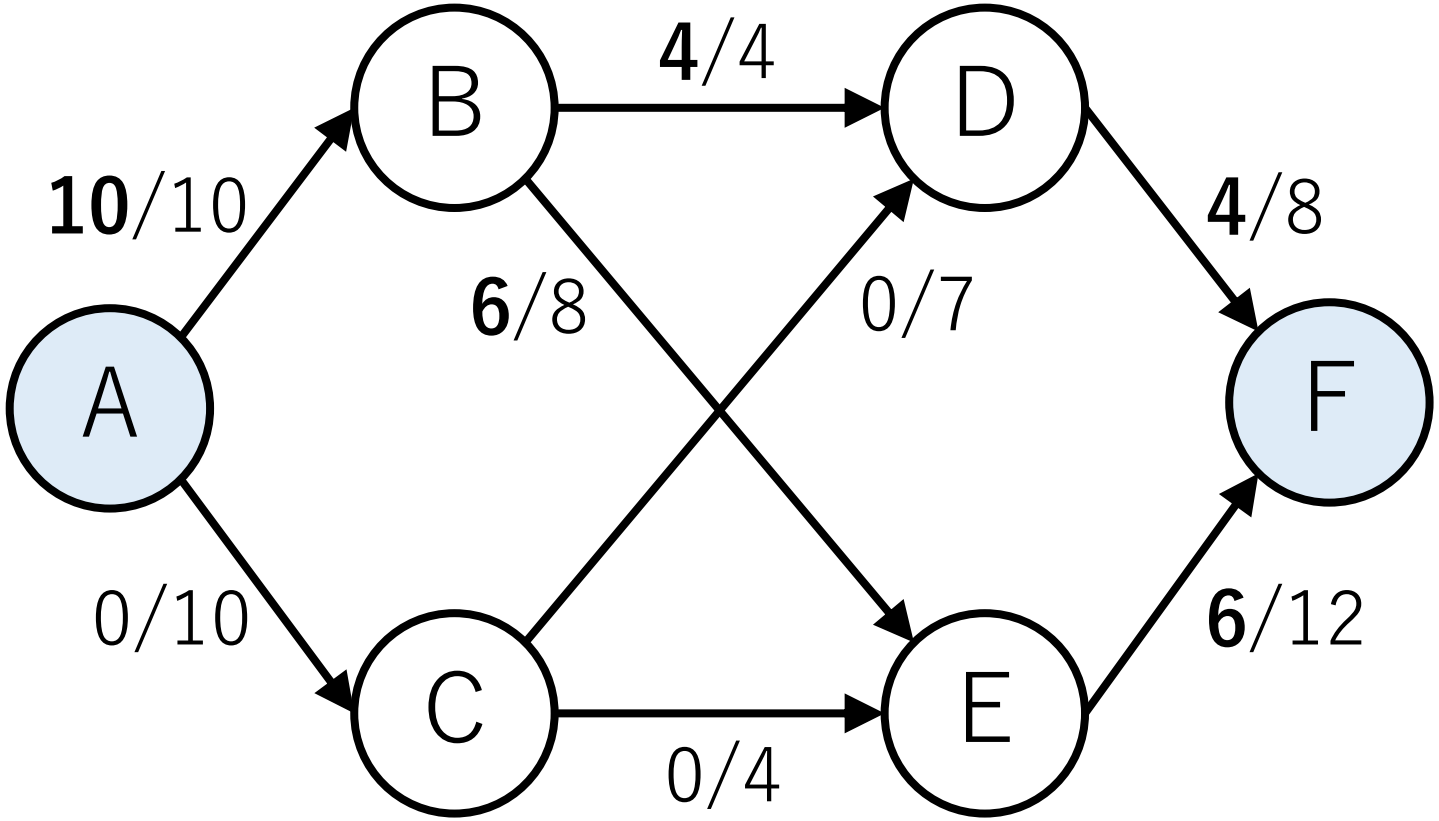
貪欲法による例

B->E->Fは本来8流せるが，A->Bの残りは6.



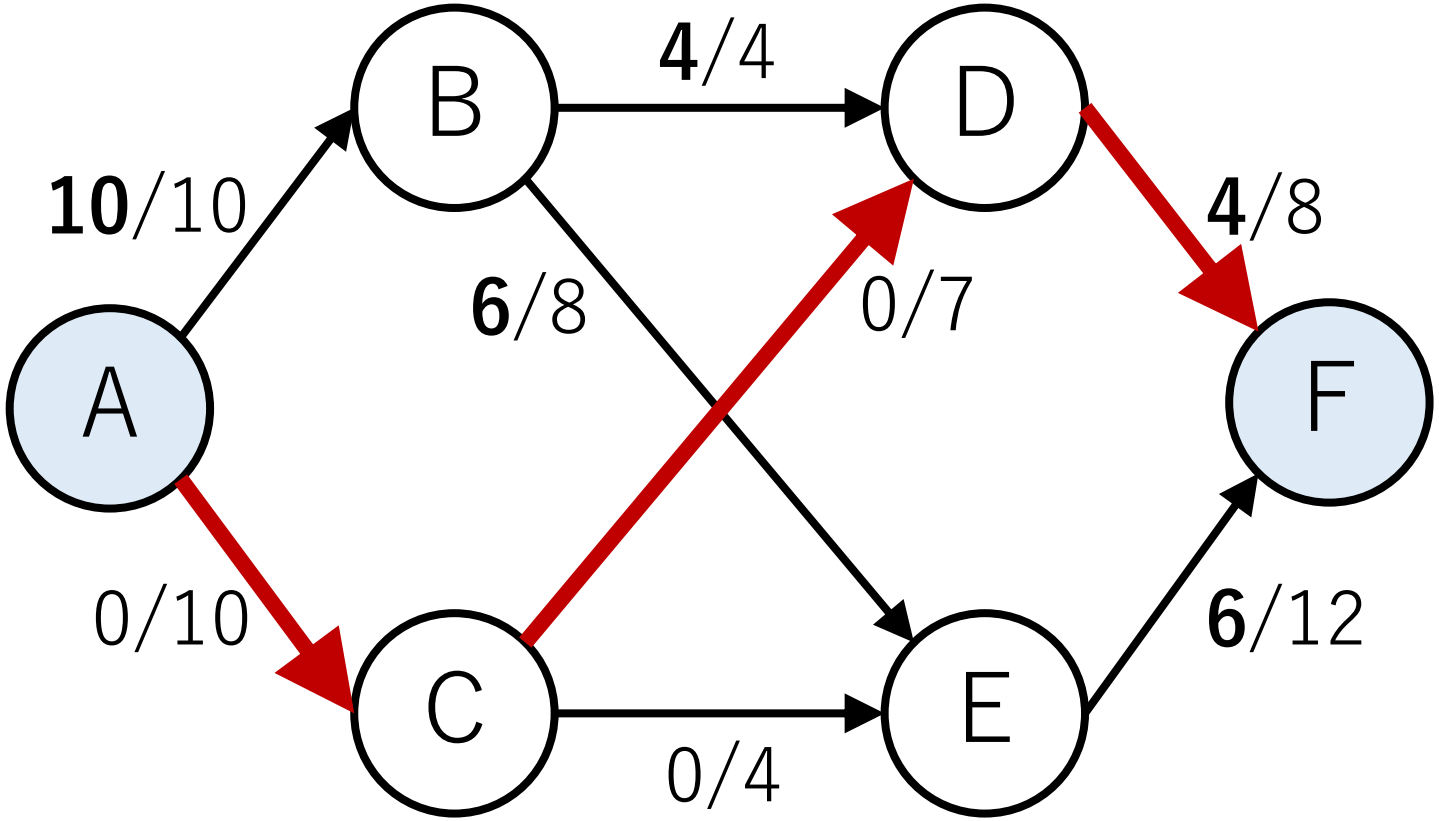
貪欲法による例

A->B->E->Fに6流す.



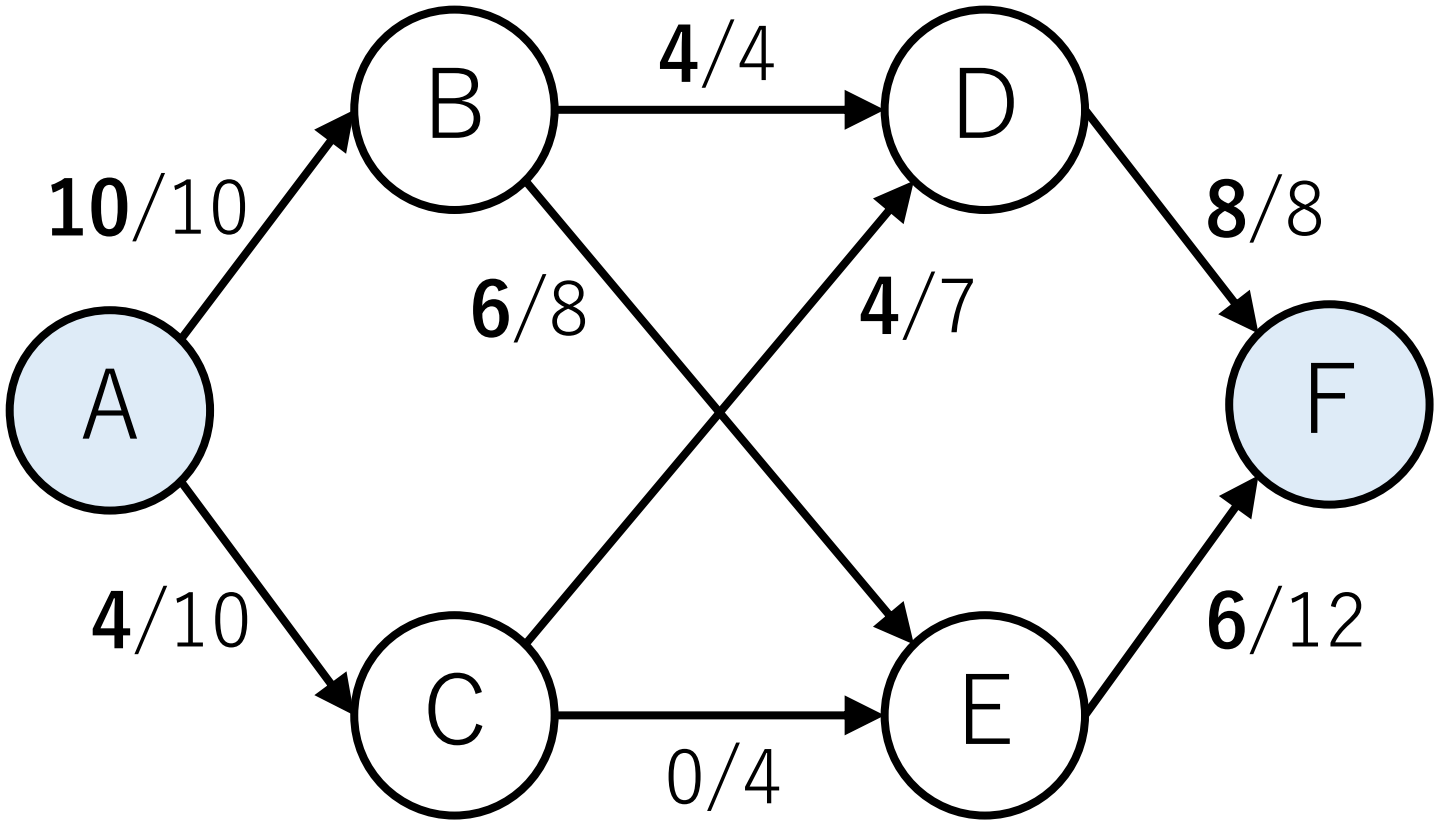
貪欲法による例

次に，A->C->D->Fを見つける．ここに流せるのは4.



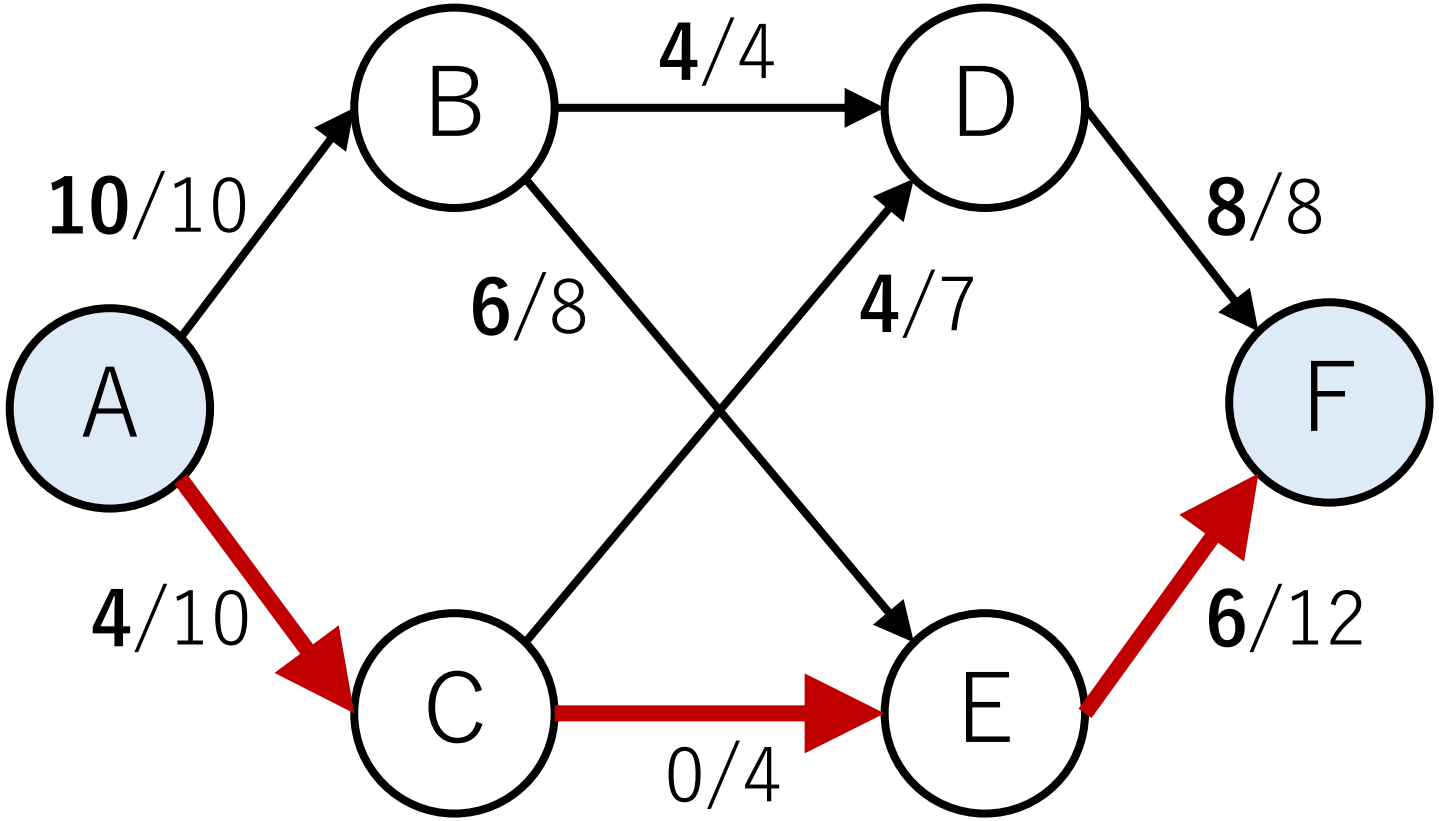
貪欲法による例

A->C->D->Fに4流す.



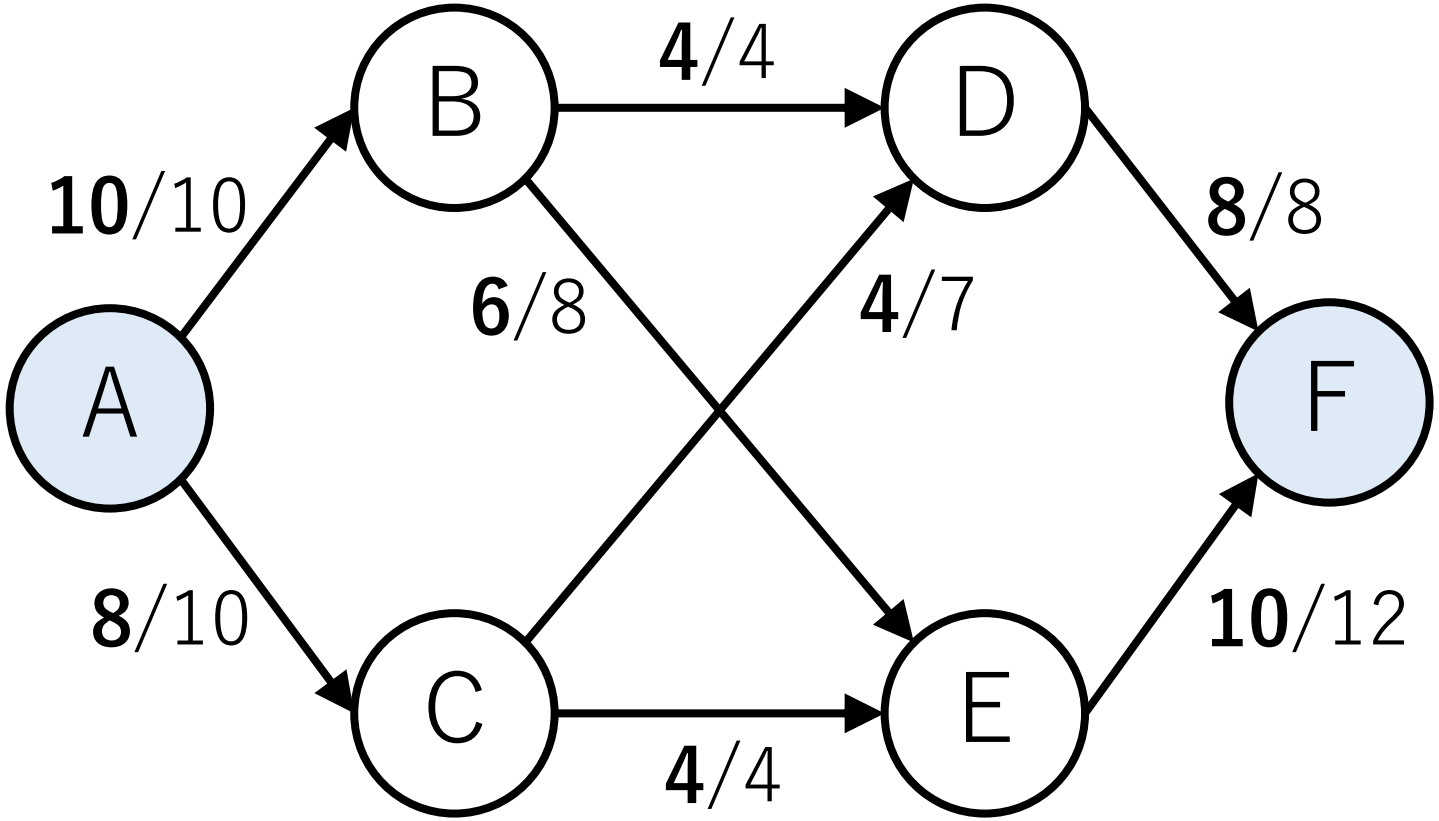
貪欲法による例

最後に，A->C->E->Fを見つける．ここには4流せる．



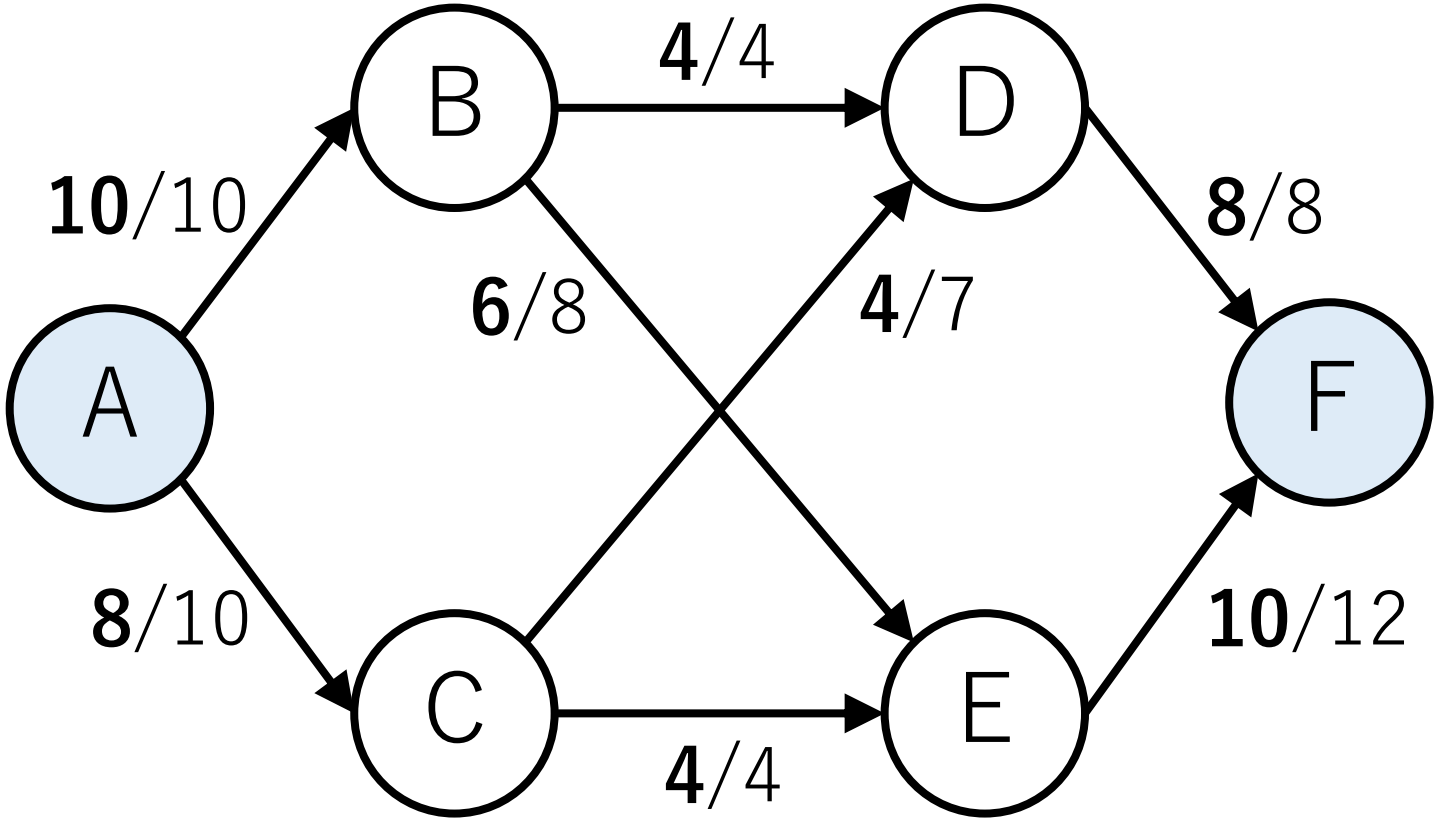
貪欲法による例

A->C->D->Fに4流す.



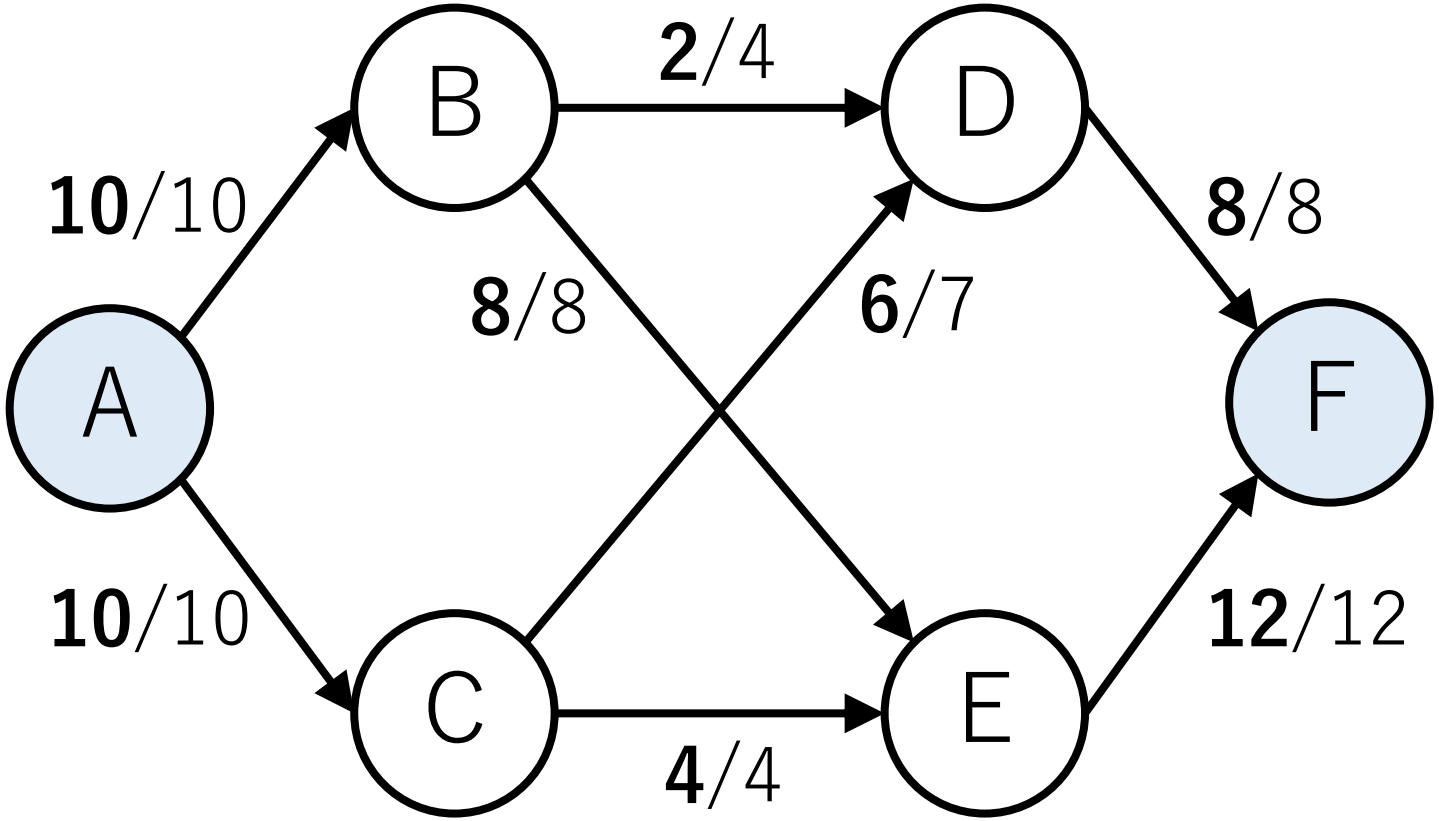
貪欲法による例

よって，最大流は18？



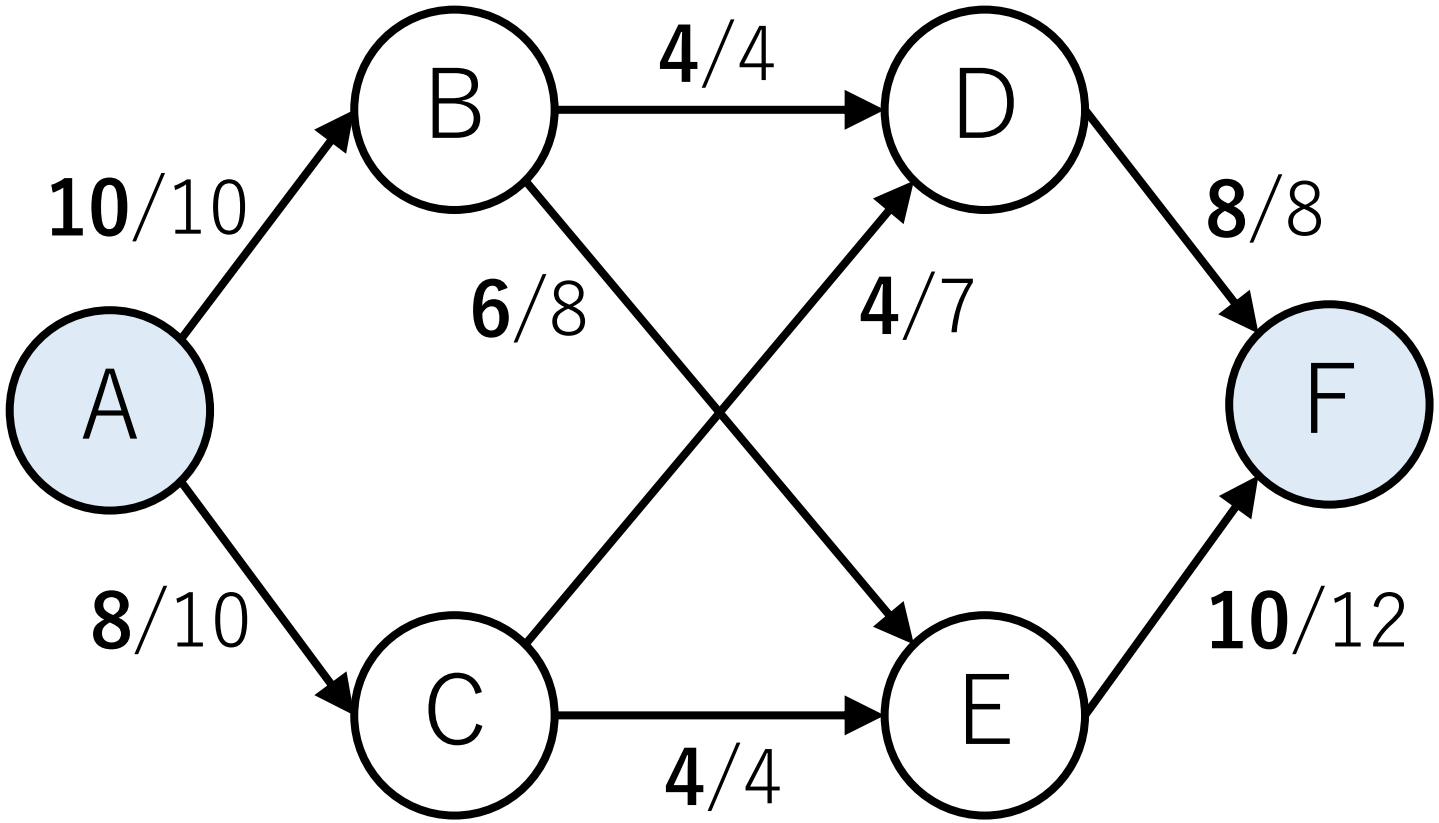
実際には．．．

最適解は20.



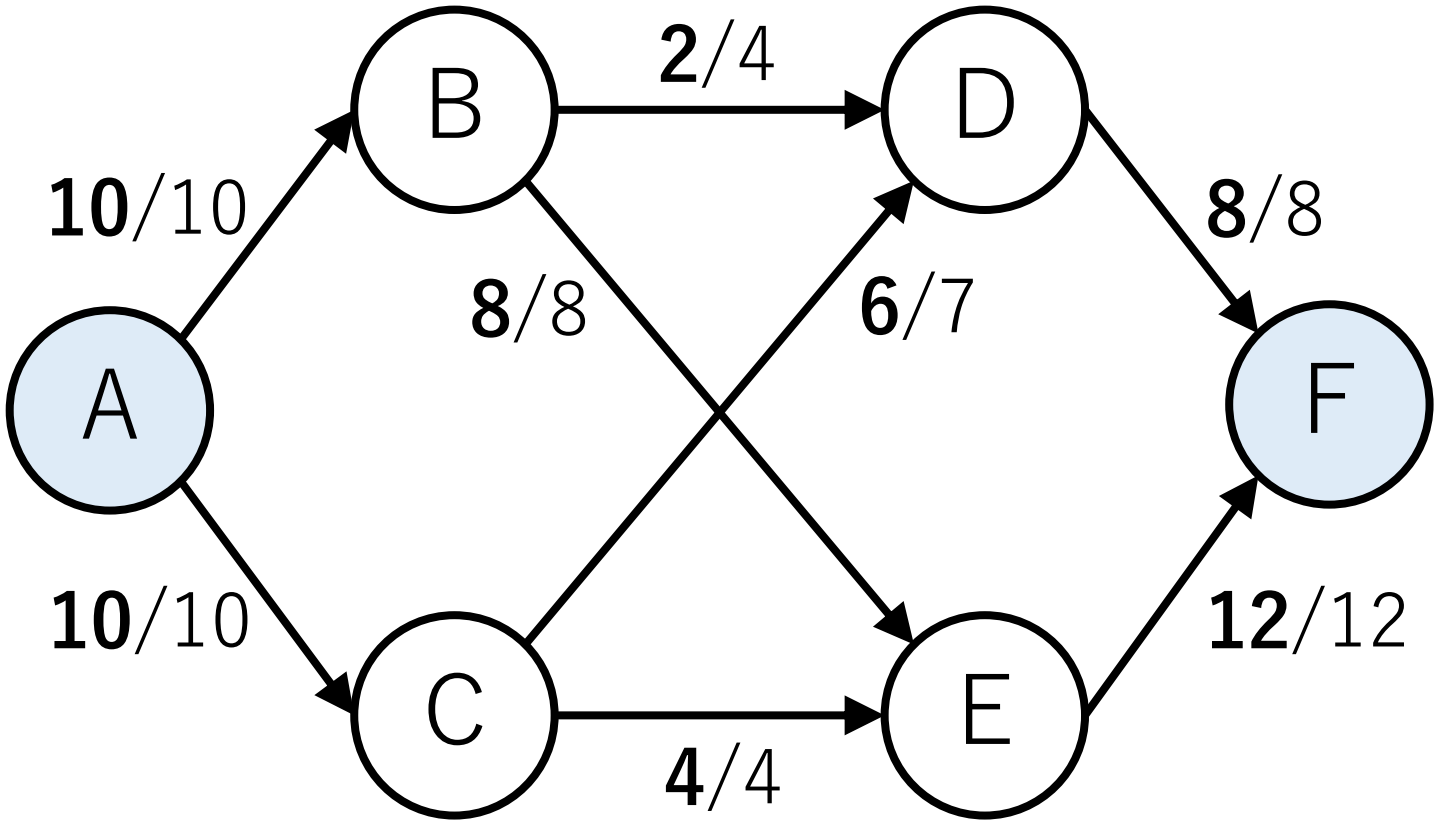
比較してみよう

貪欲法の場合



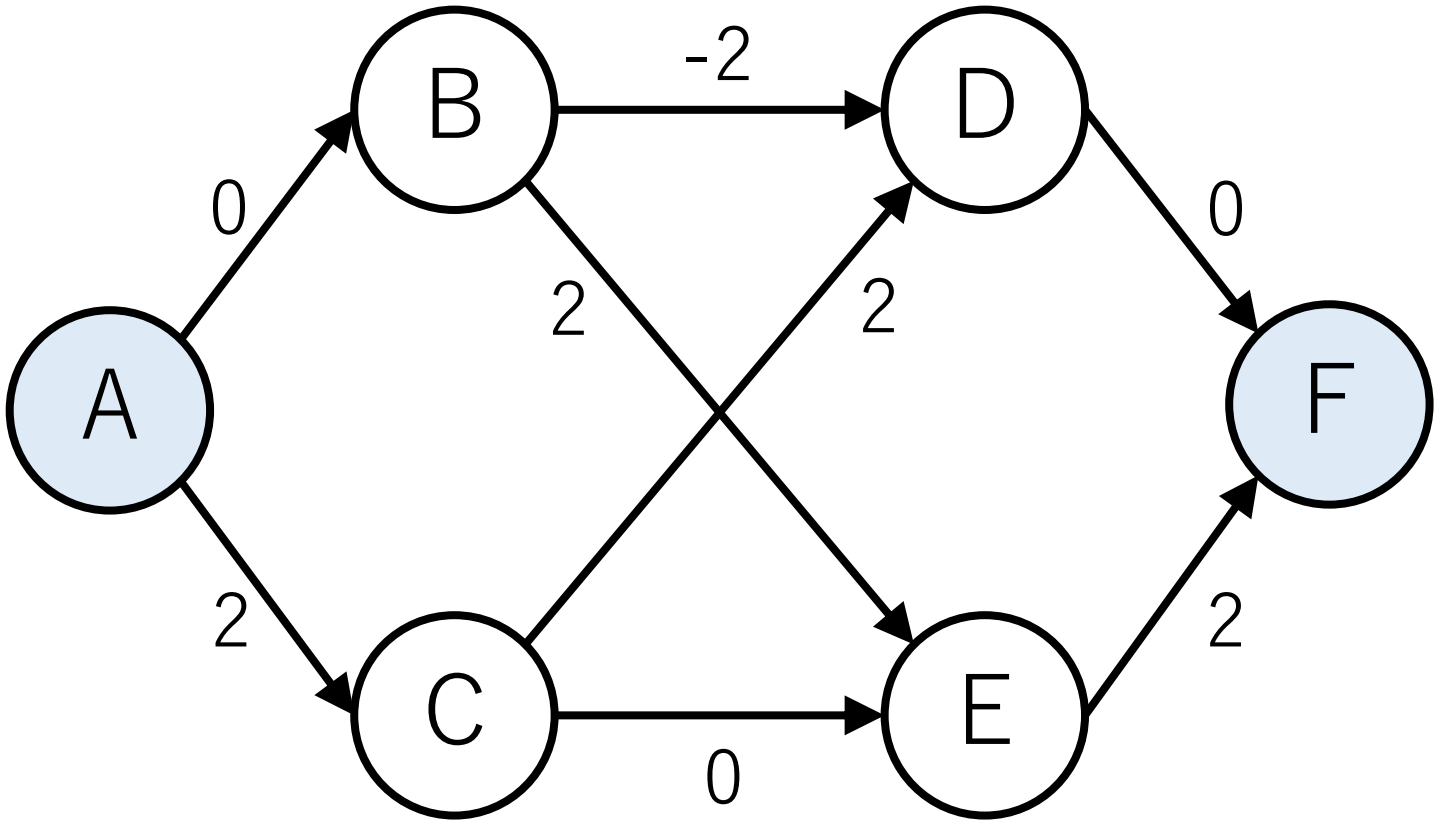
比較してみよう

最適解の場合



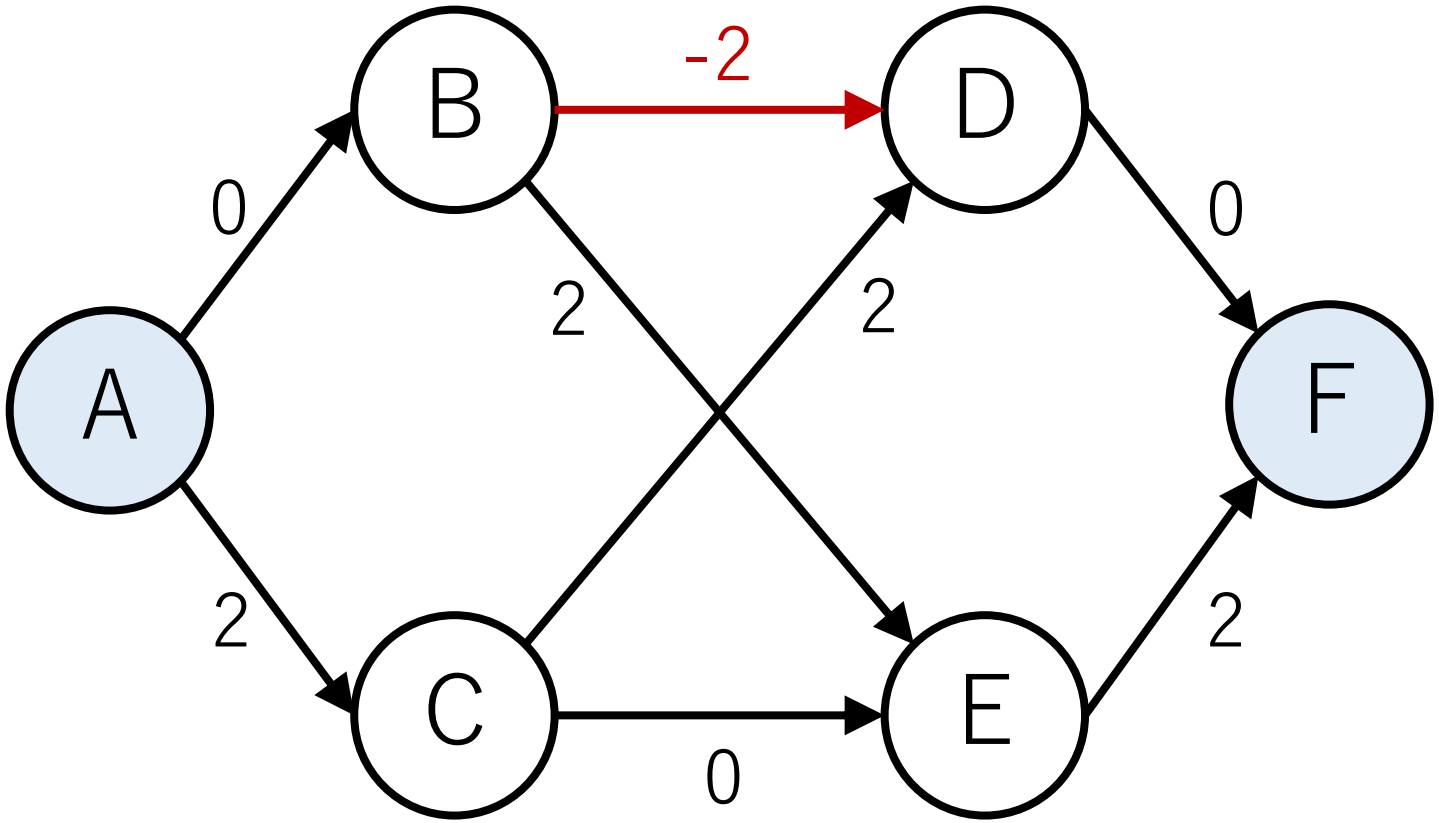
差をとってみよう.

最大流の場合 - 貪欲法の場合



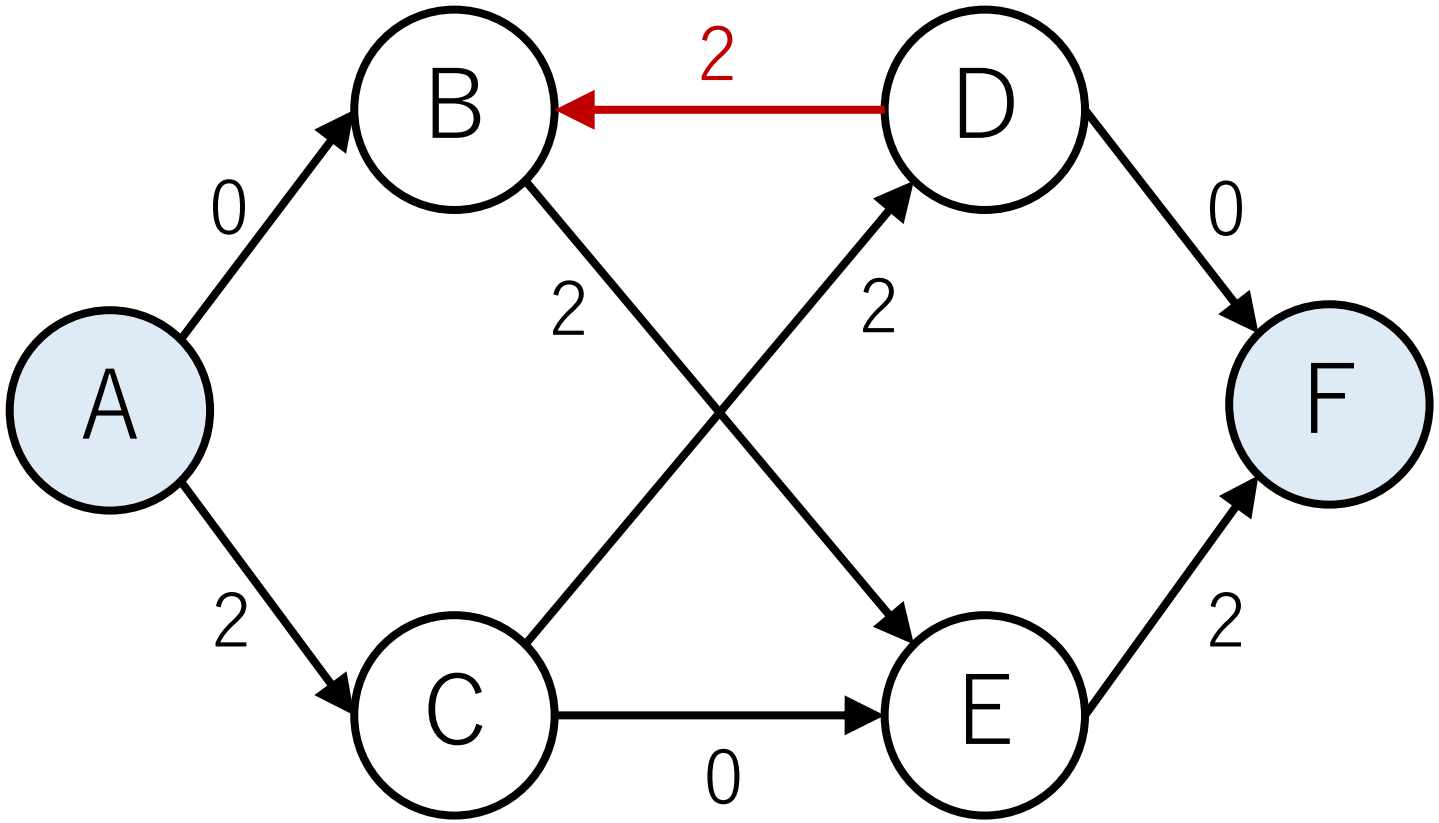
差をとってみよう.

負の流れになる経路がある！



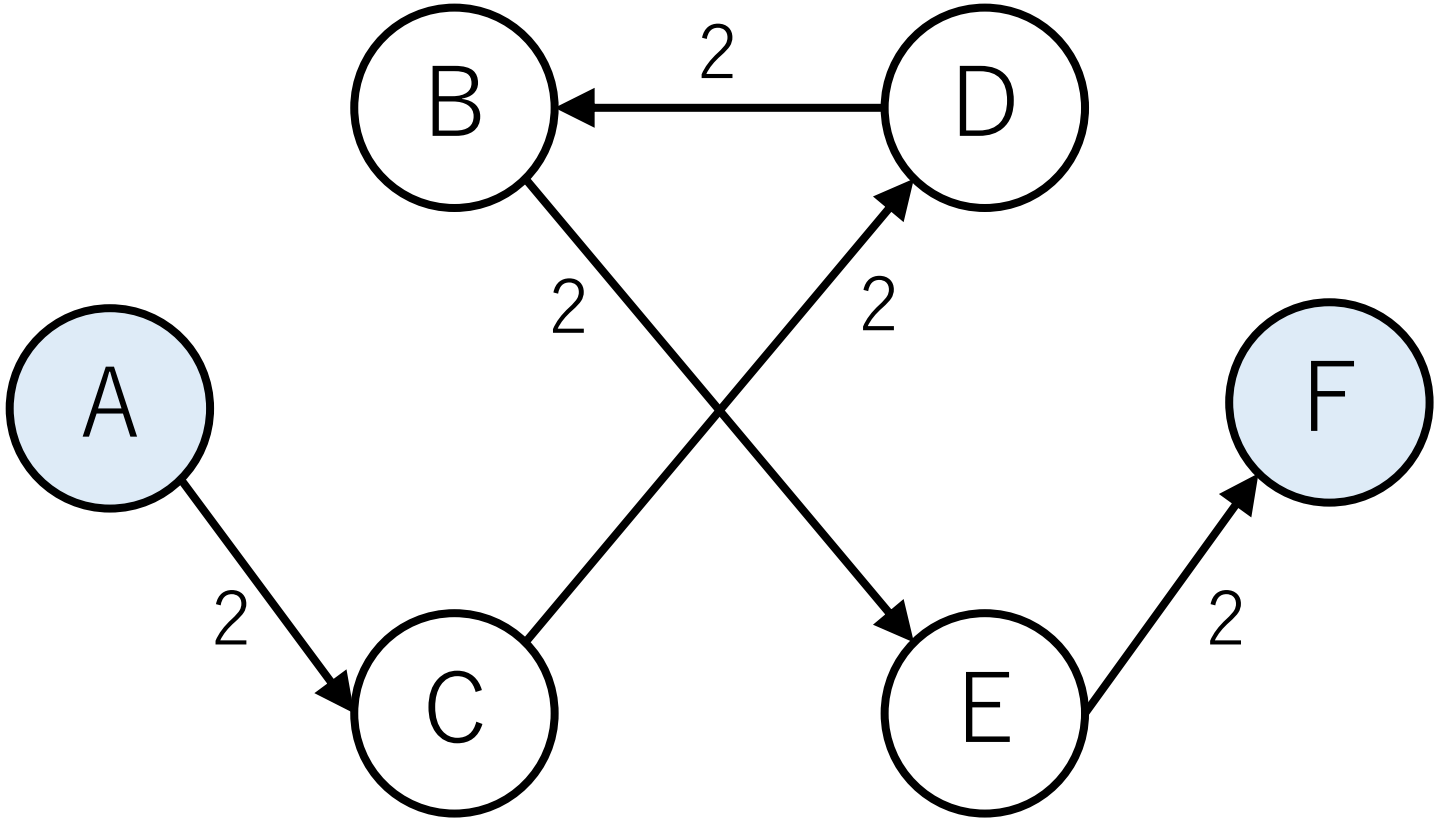
差をとってみよう.

負の流れになる経路がある！



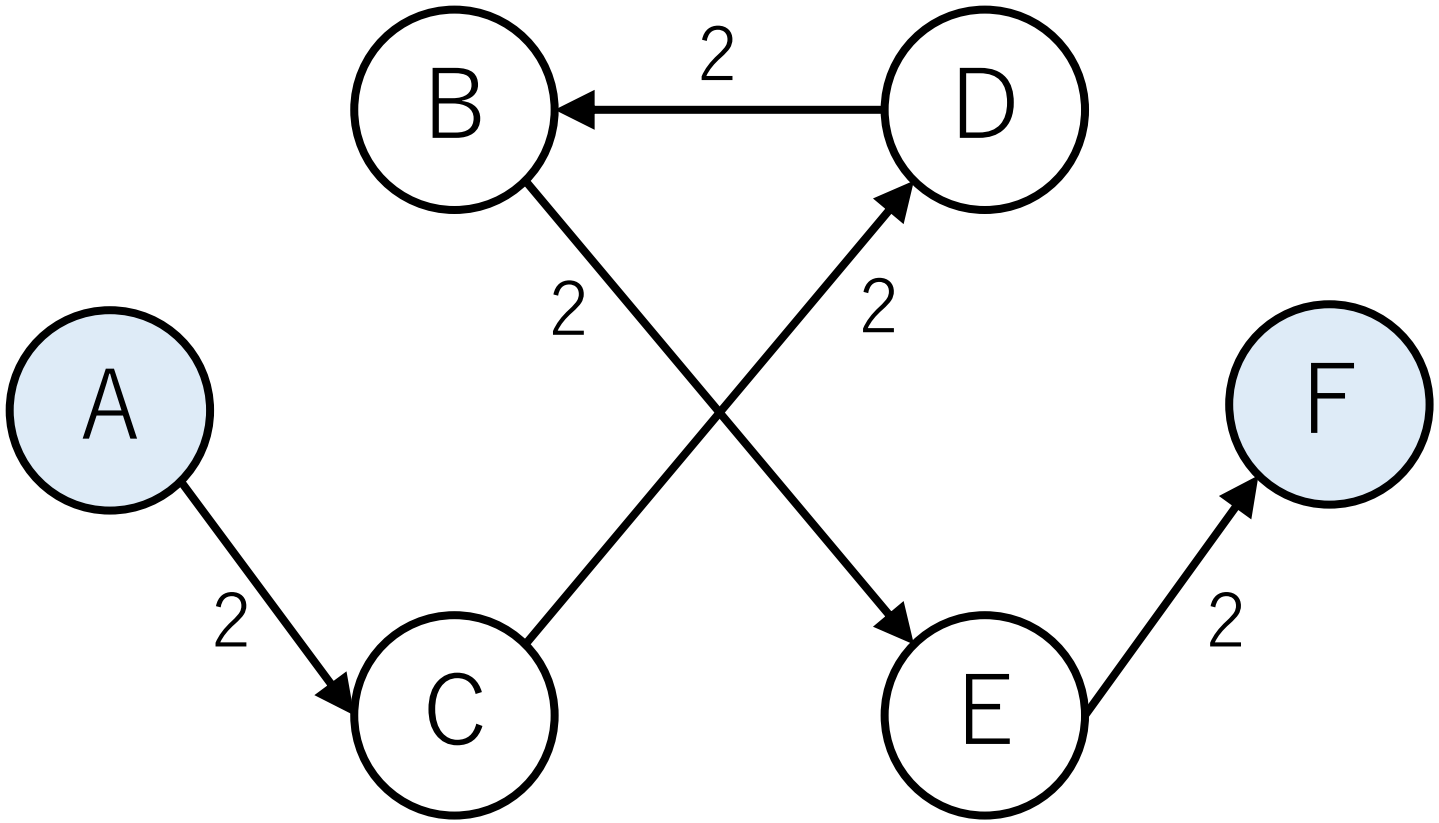
差をとってみよう.

貪欲法では考慮しなかった経路が浮き出る.



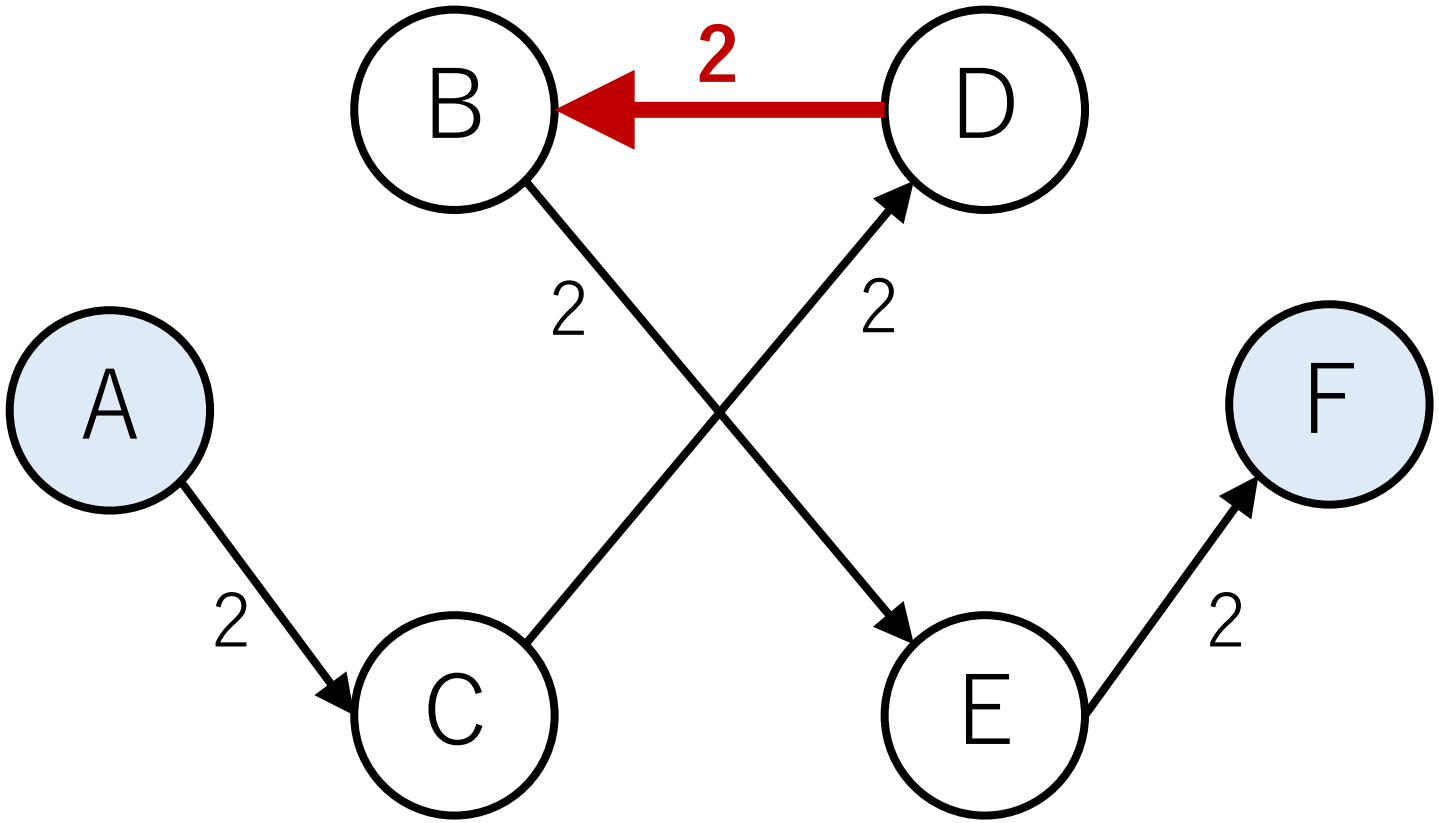
差をとってみよう.

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

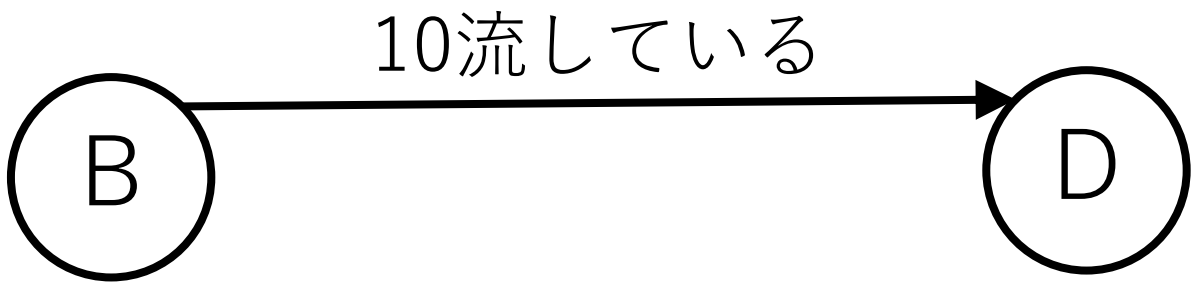
基本は貪欲法と同じく，DFSやBFSで開始ノードから終了ノードまでの経路を探索し，流せるだけ流す．**ただし，経路の探索において，ノード間を逆にたどるような仮想的な経路も考慮する．**

「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．

全ての経路の容量は整数か有理数であるとする．

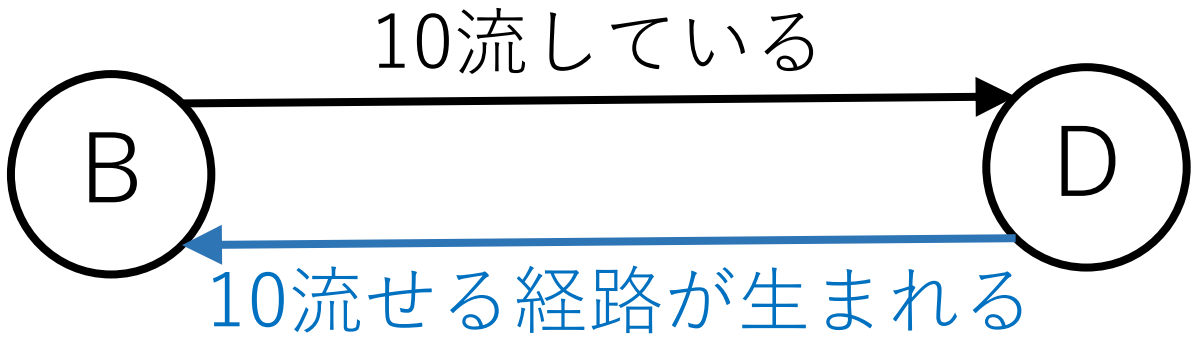
フォード・ファルカーソン法

逆方向の容量＝順方向の今までの流量



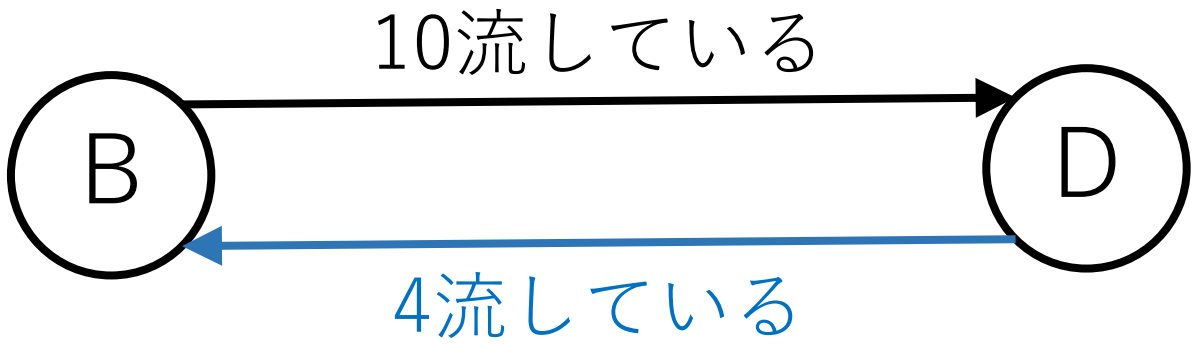
フォード・ファルカーソン法

逆方向の容量＝順方向の今までの流量



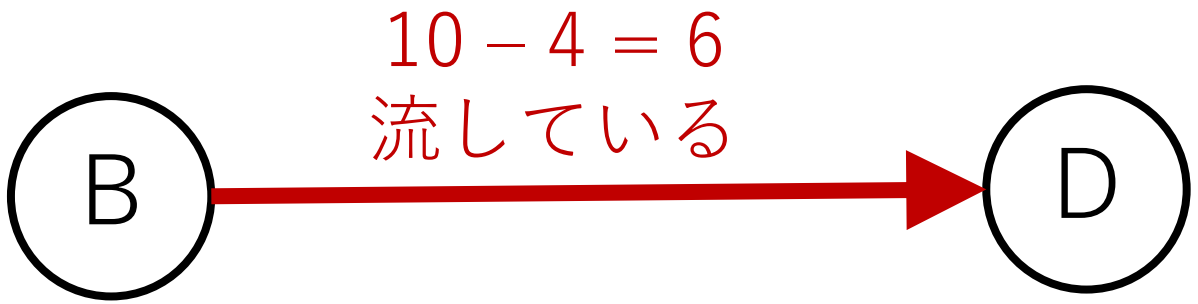
フォード・ファルカーソン法

もし，逆方向に4流すとすると？



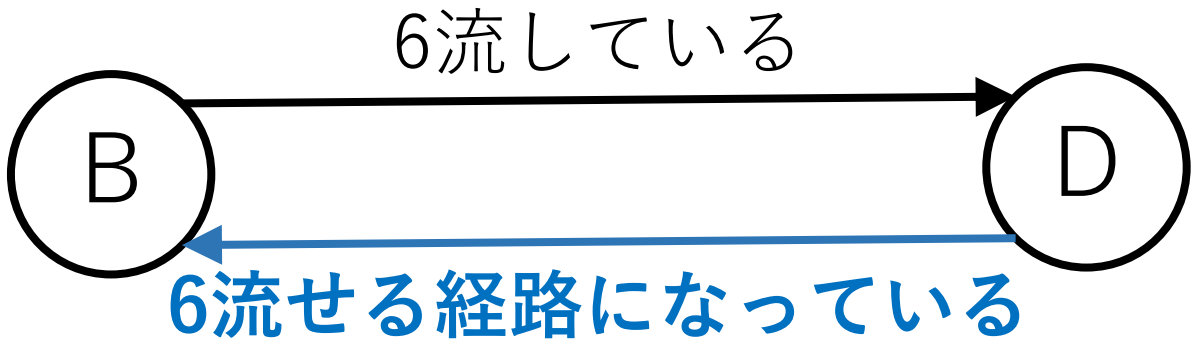
フォード・ファルカーソン法

2つを合わせると，B->Dに6流していることと同値.



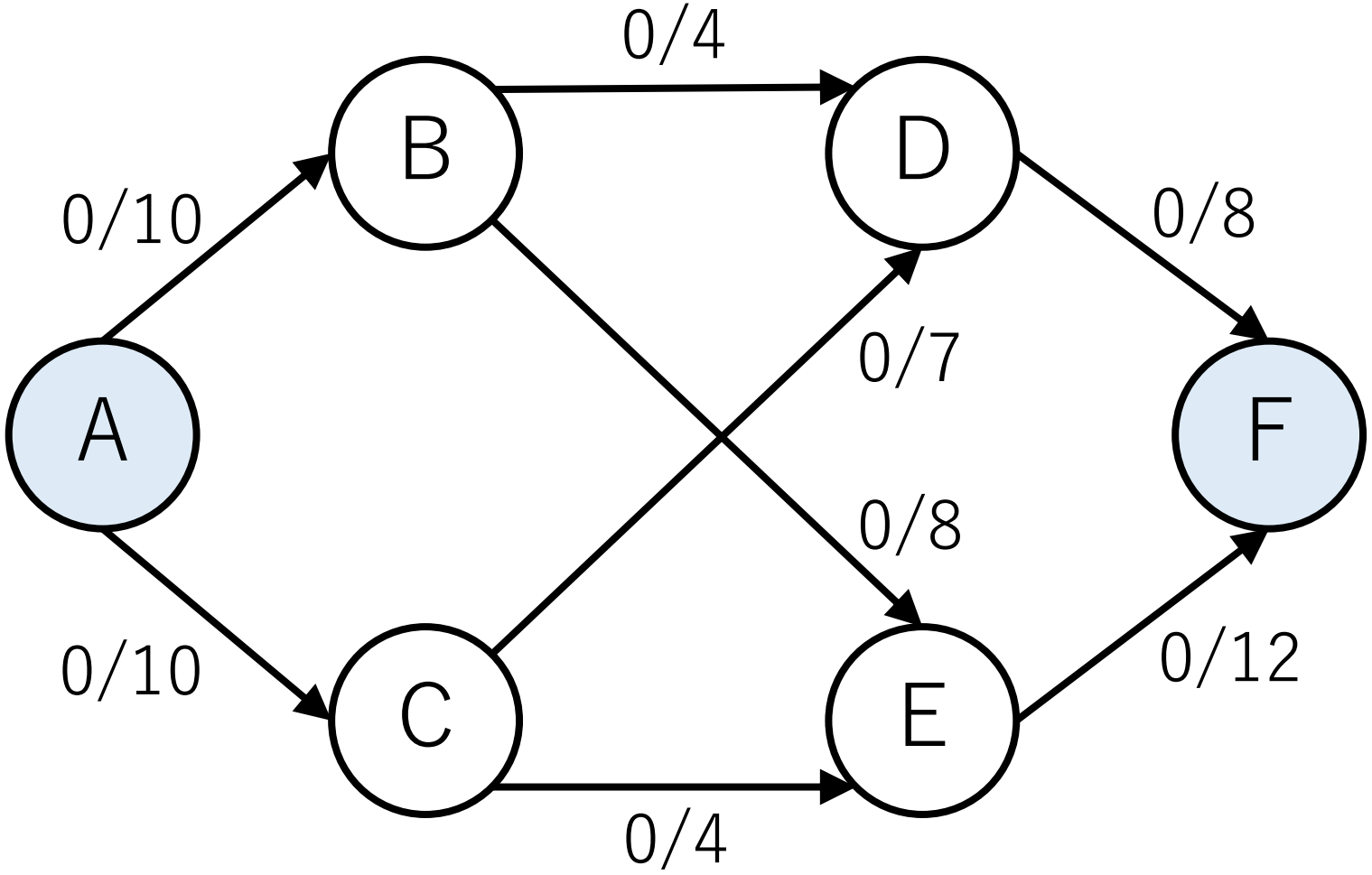
フォード・ファルカーソン法

逆方向の容量も更新する.



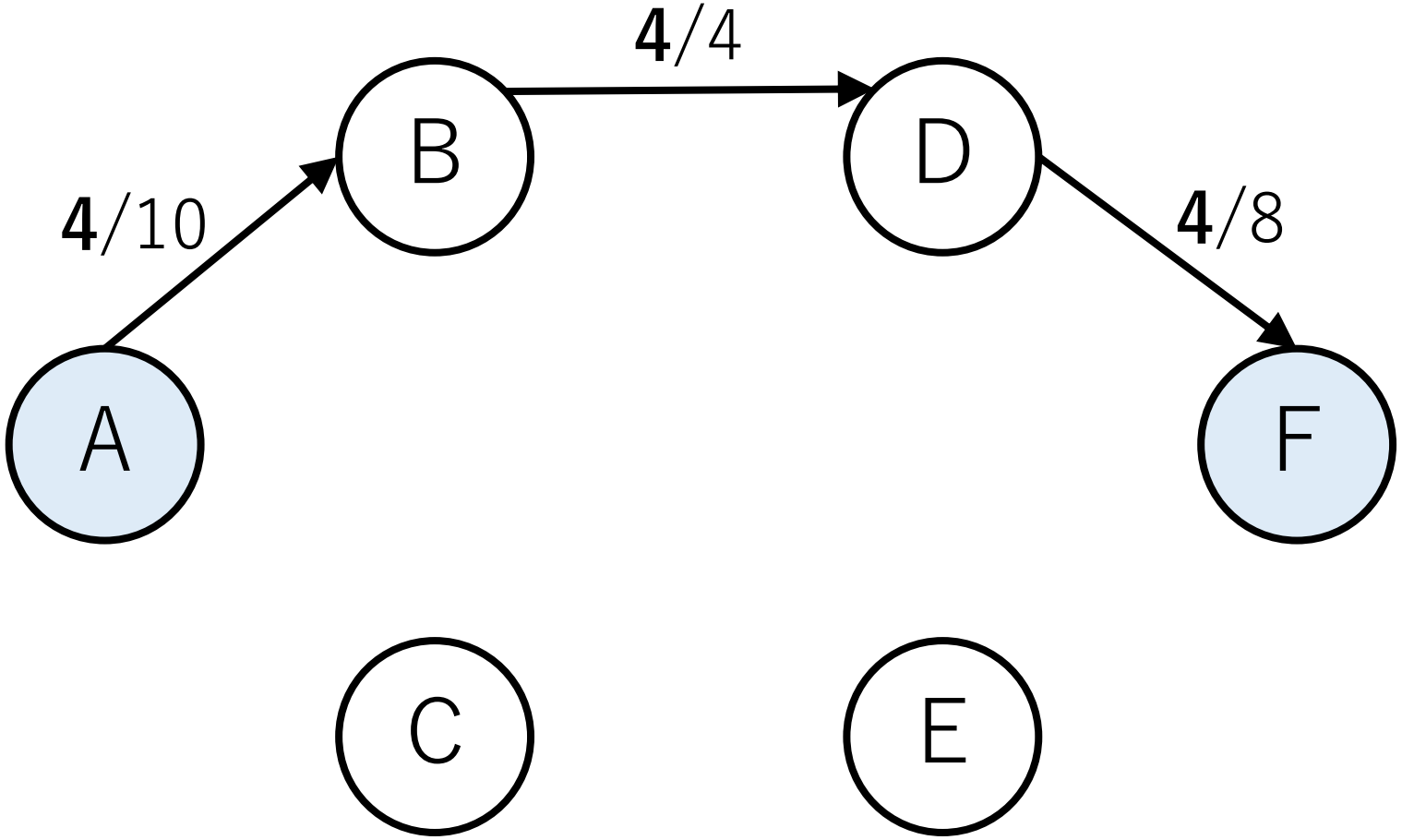
フォード・ファルカーソン法による例

先程の例で見てみよう.



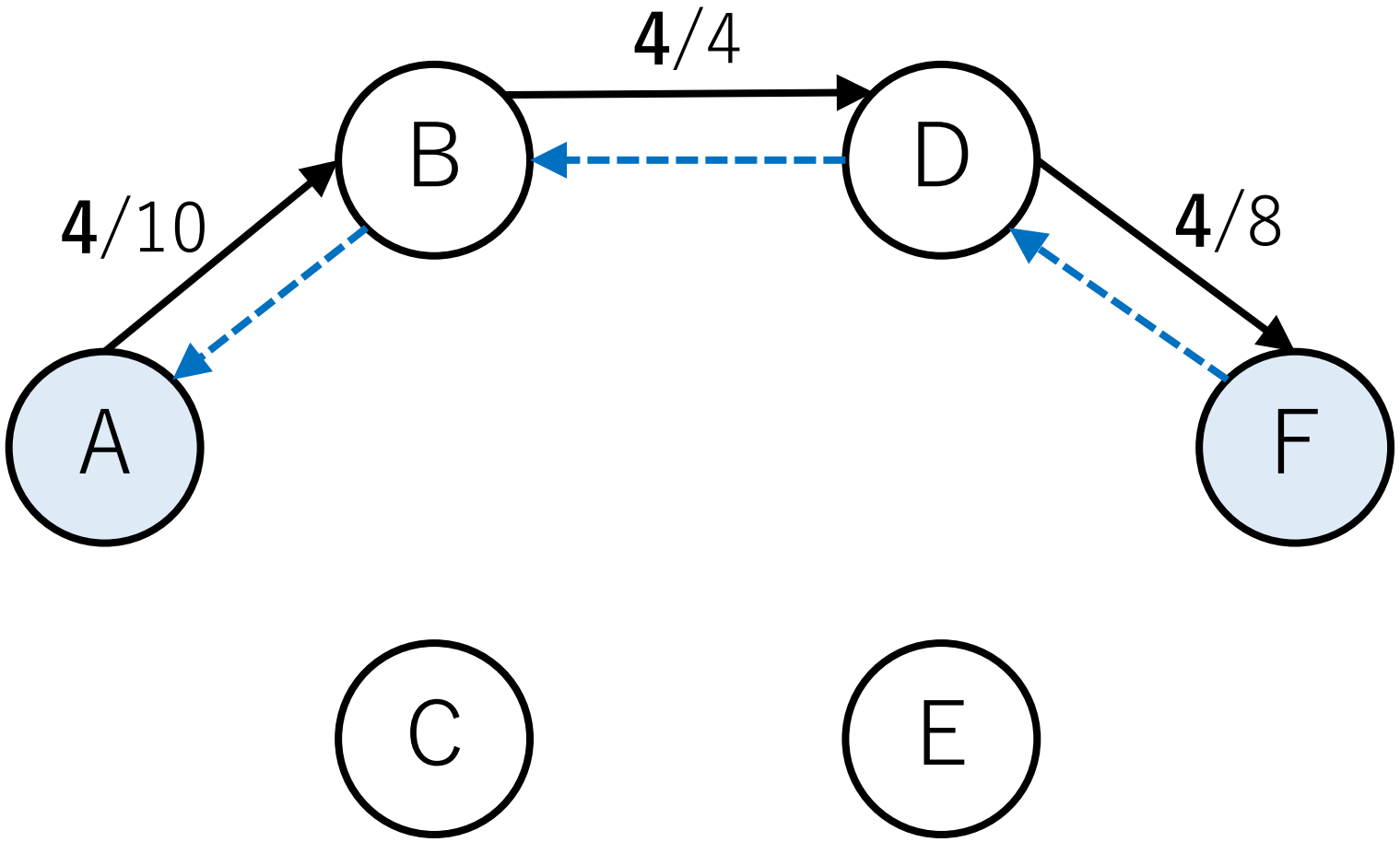
フォード・ファルカーソン法による例

A->B->D->Fを見つけて，4流す．



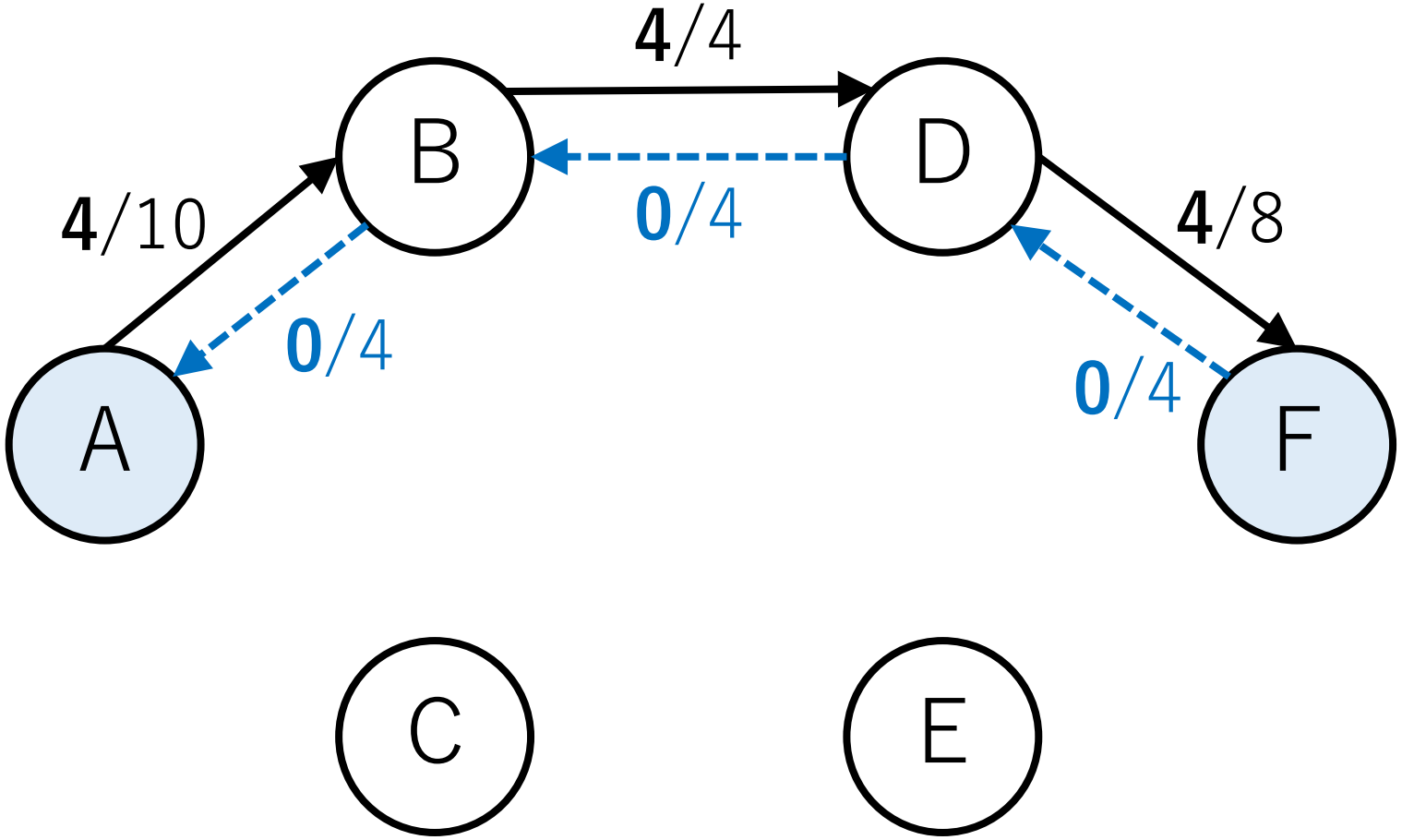
フォード・ファルカーソン法による例

逆方向の経路を設定.



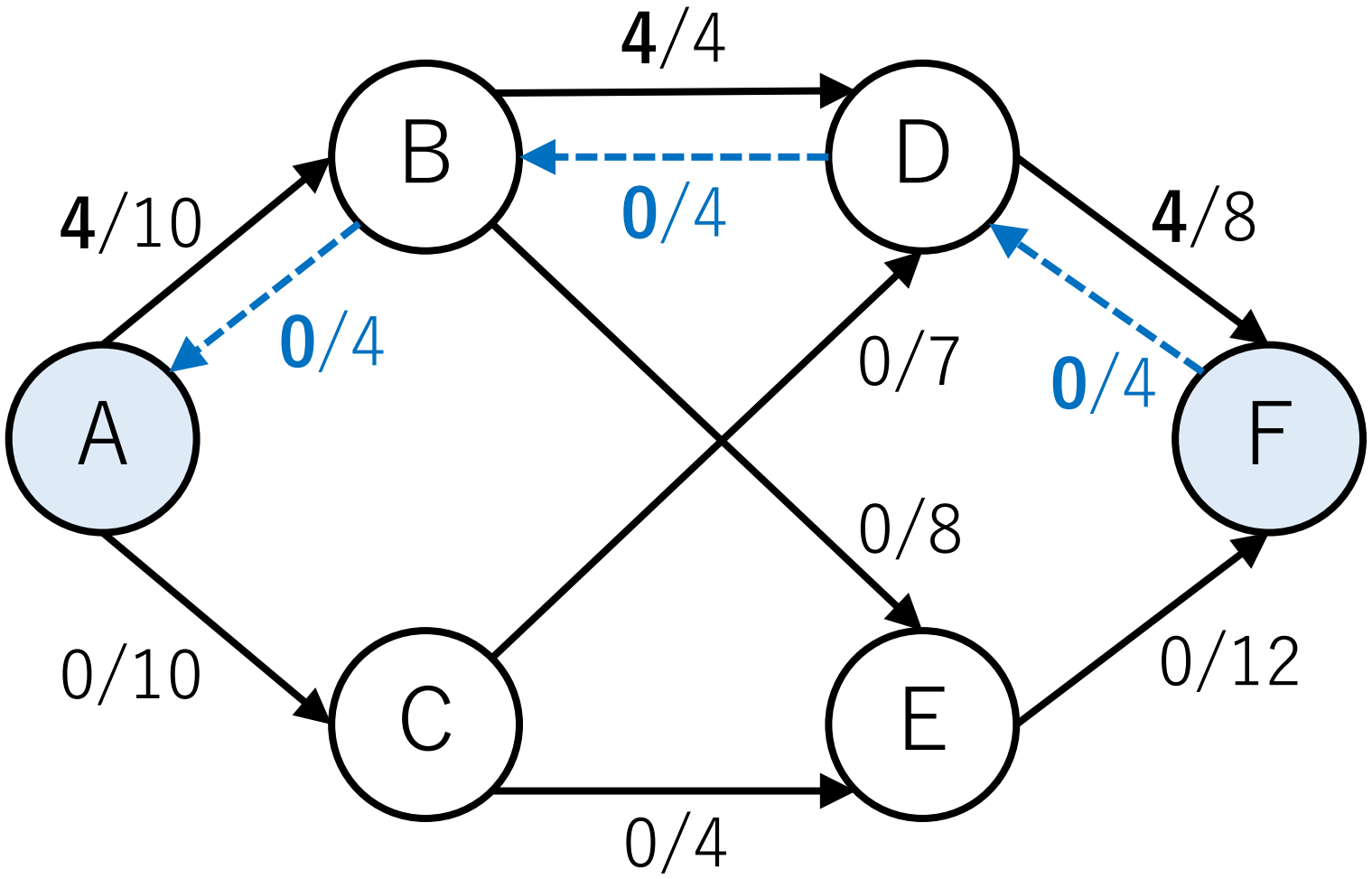
フォード・ファルカーソン法による例

「逆方向の容量＝順方向の今までの流量」で容量を更新.



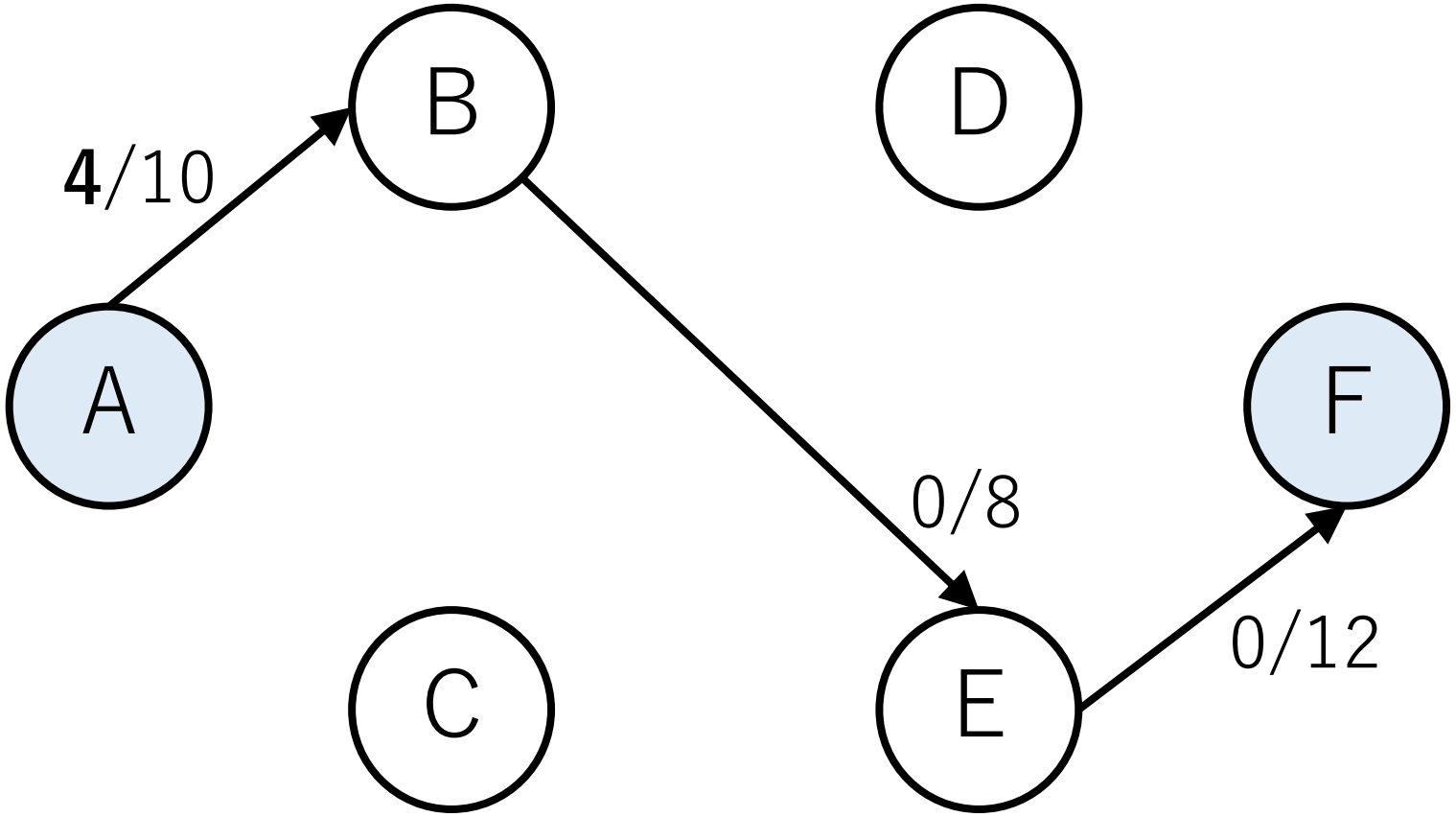
フォード・ファルカーソン法による例

現在の状態.



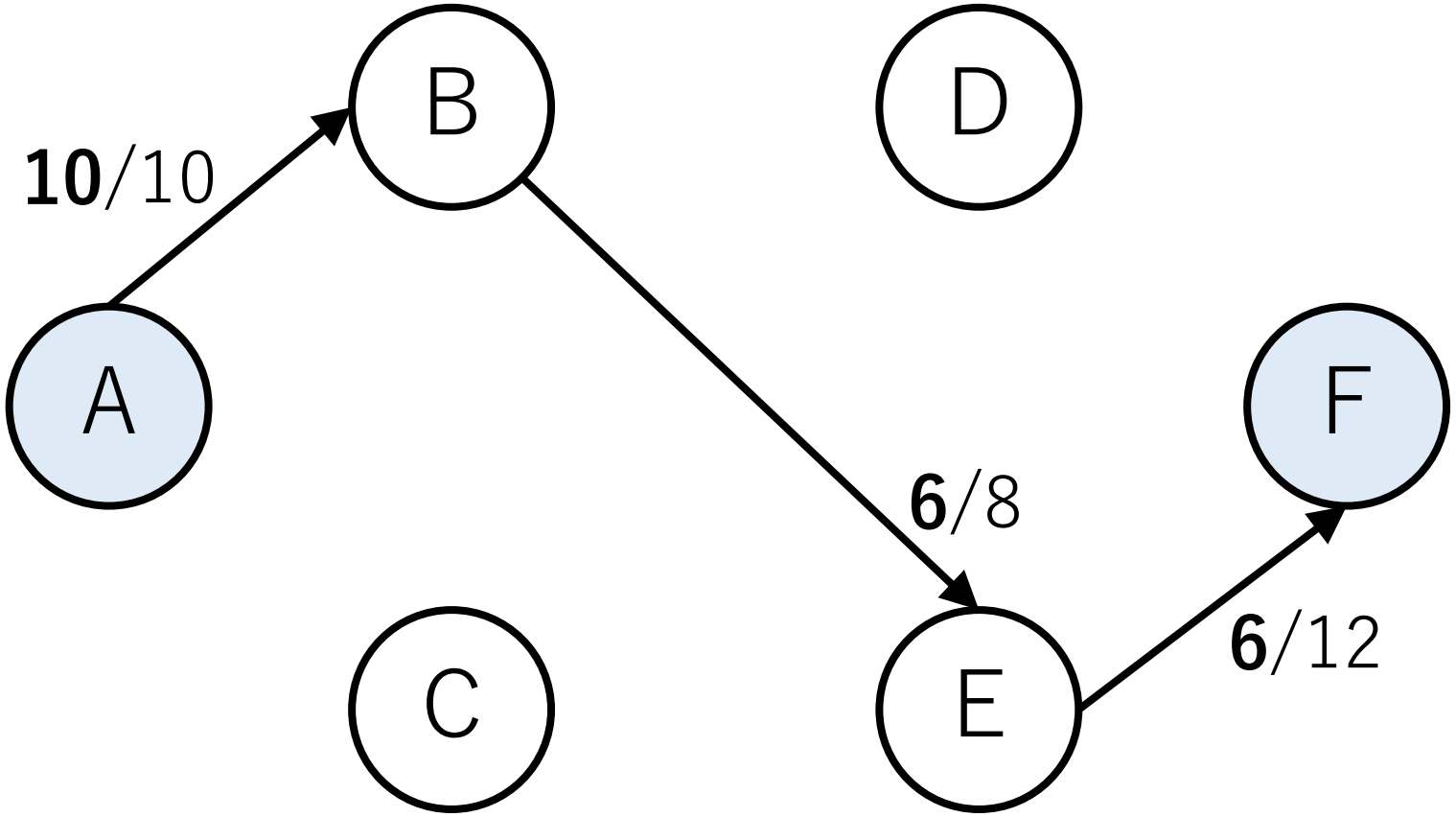
フォード・ファルカーソン法による例

A->B->E->Fを見つける．流せる量は6.



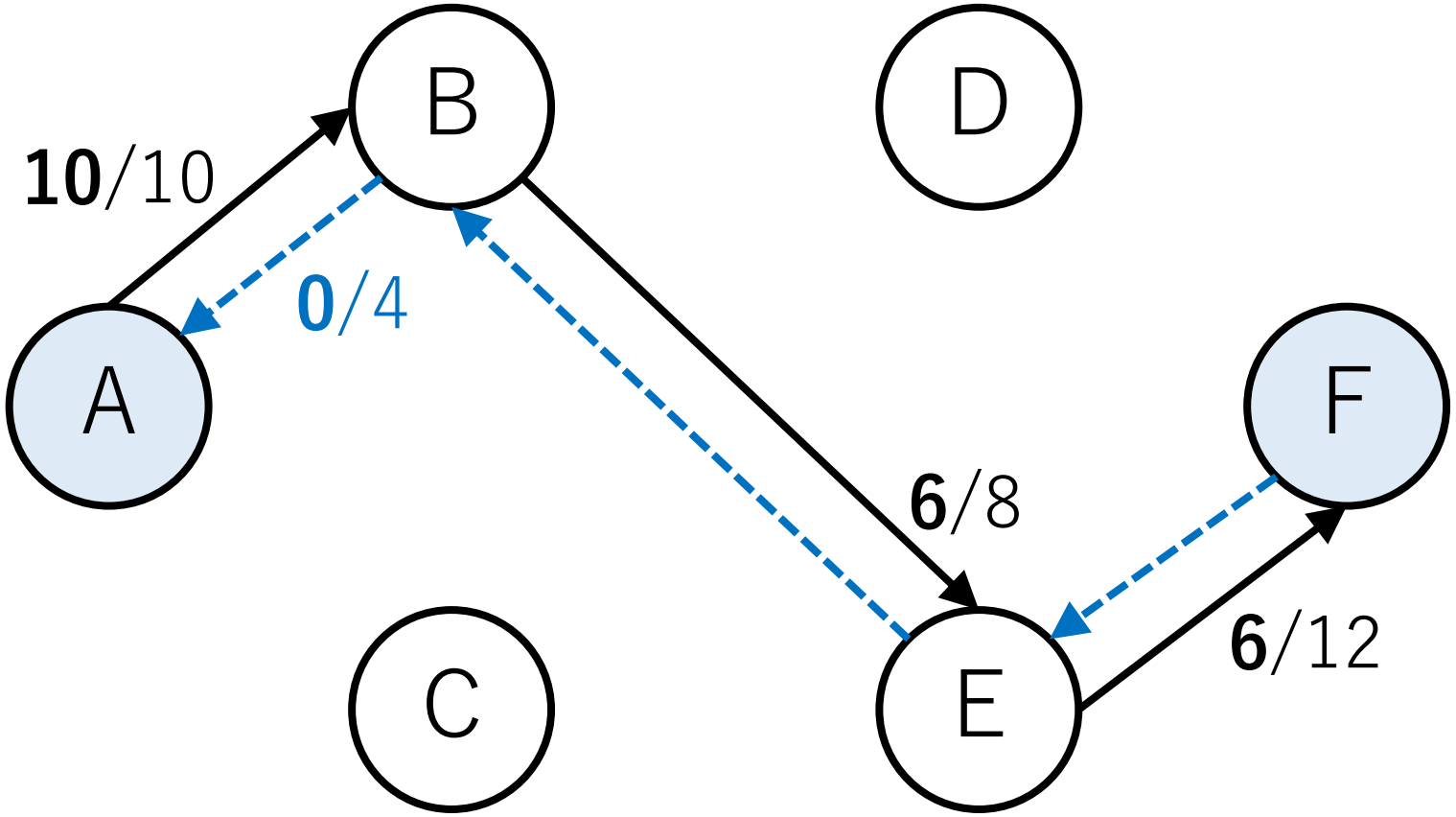
フォード・ファルカーソン法による例

6流す.



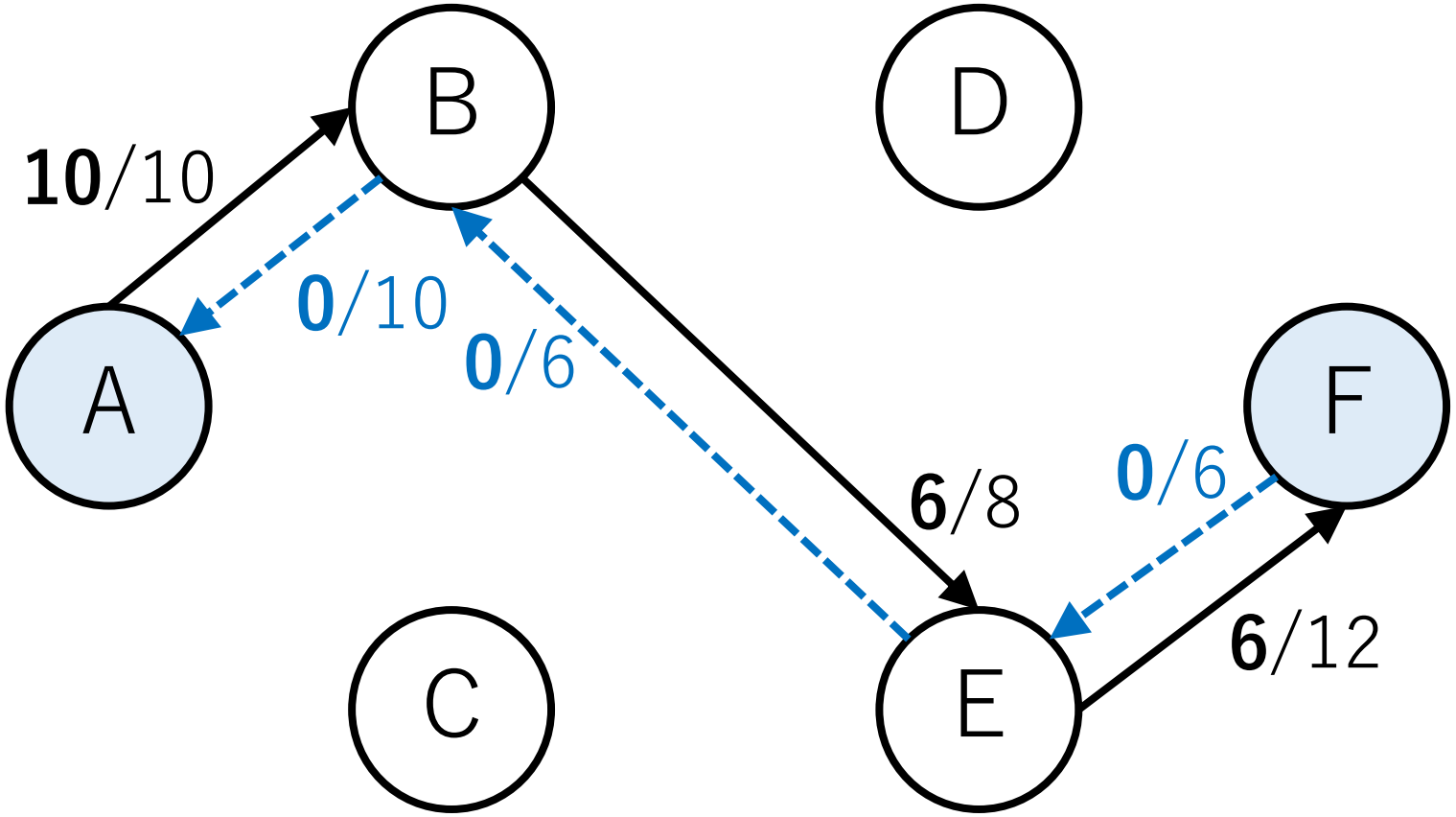
フォード・ファルカーソン法による例

逆方向の経路を設定.



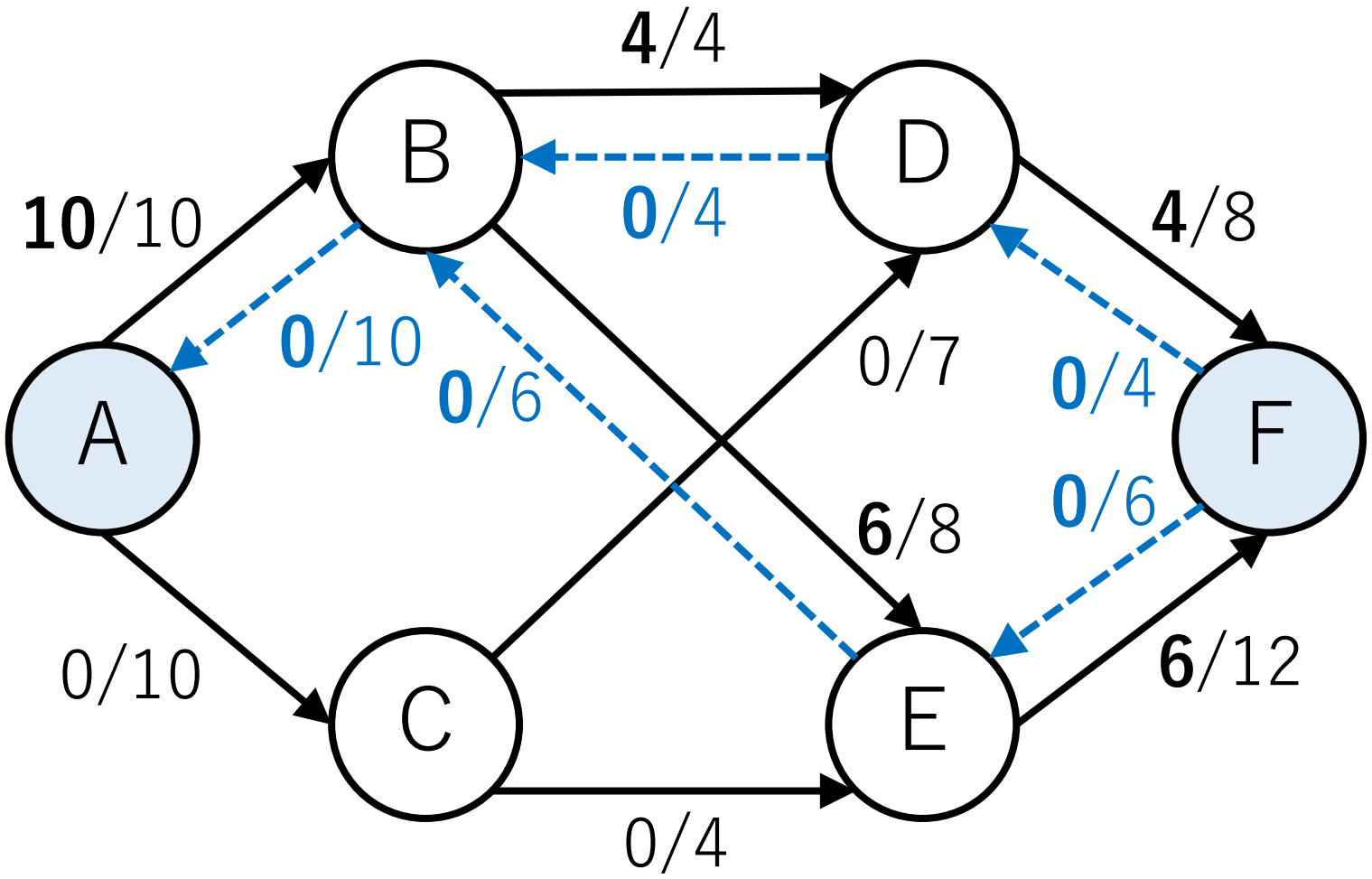
フォード・ファルカーソン法による例

「逆方向の容量＝順方向の今までの流量」で容量を更新.



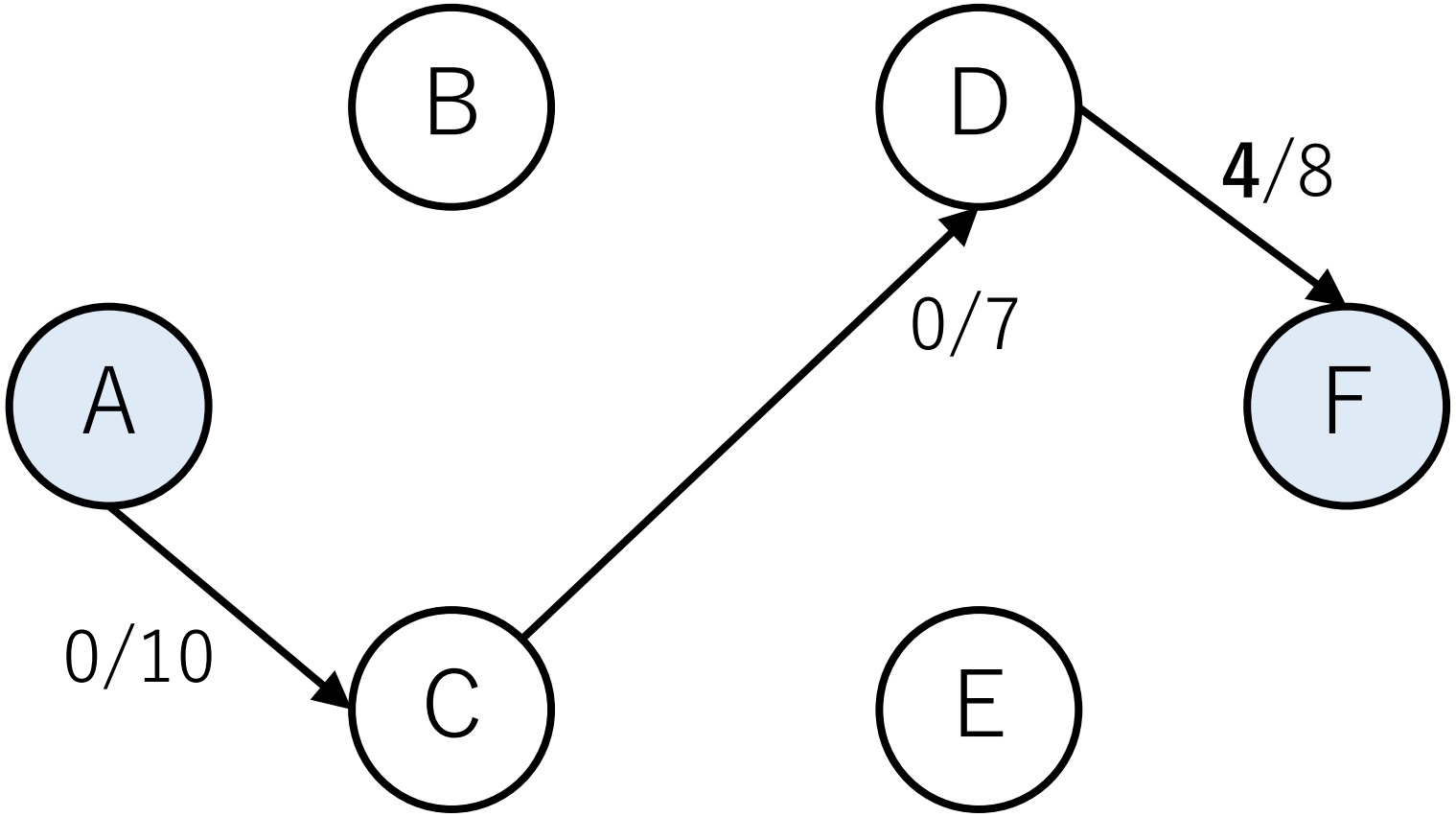
フォード・ファルカーソン法による例

現在の状態.



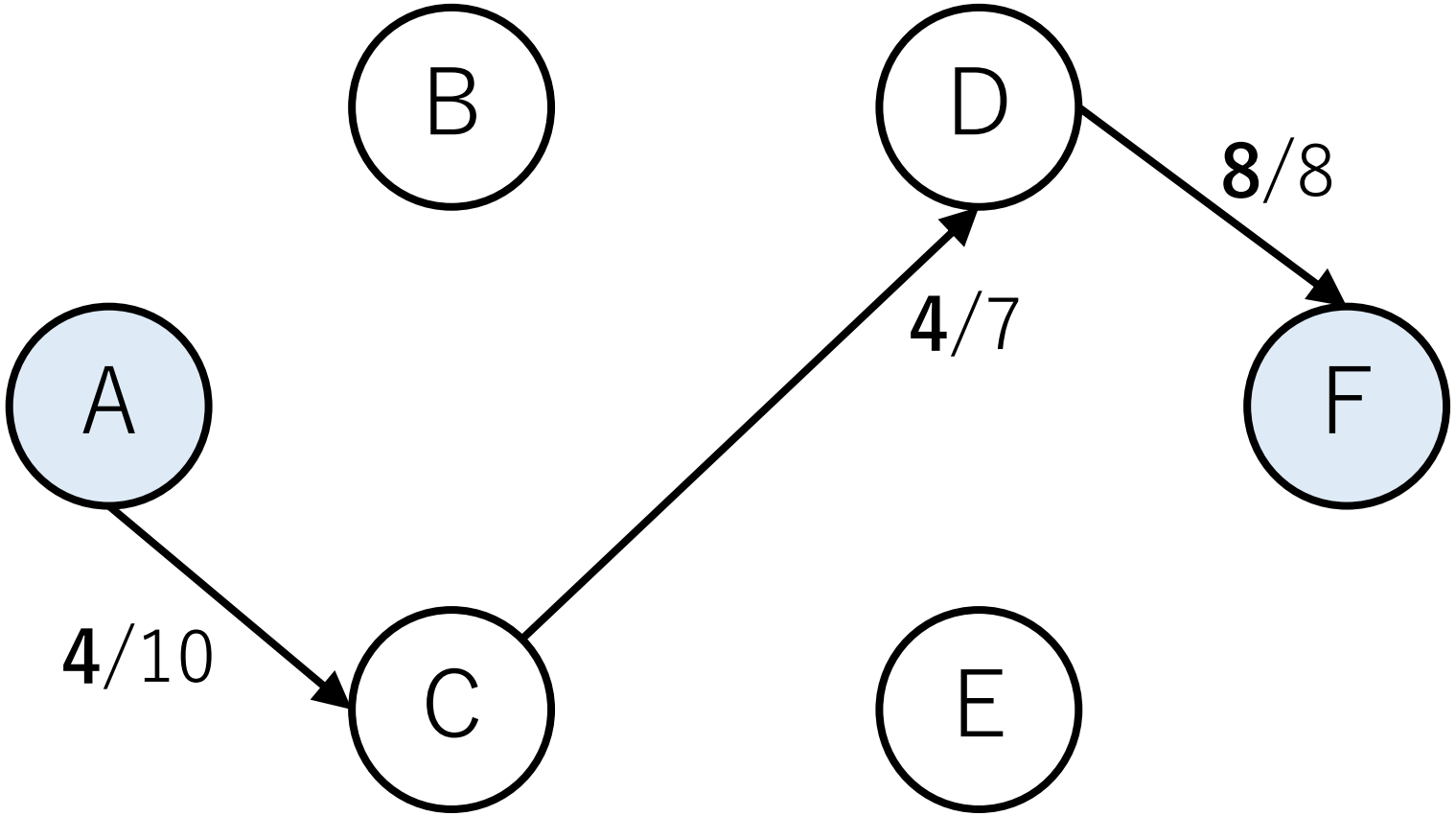
フォード・ファルカーソン法による例

A->C->D->Fを見つける．流せる量は4.



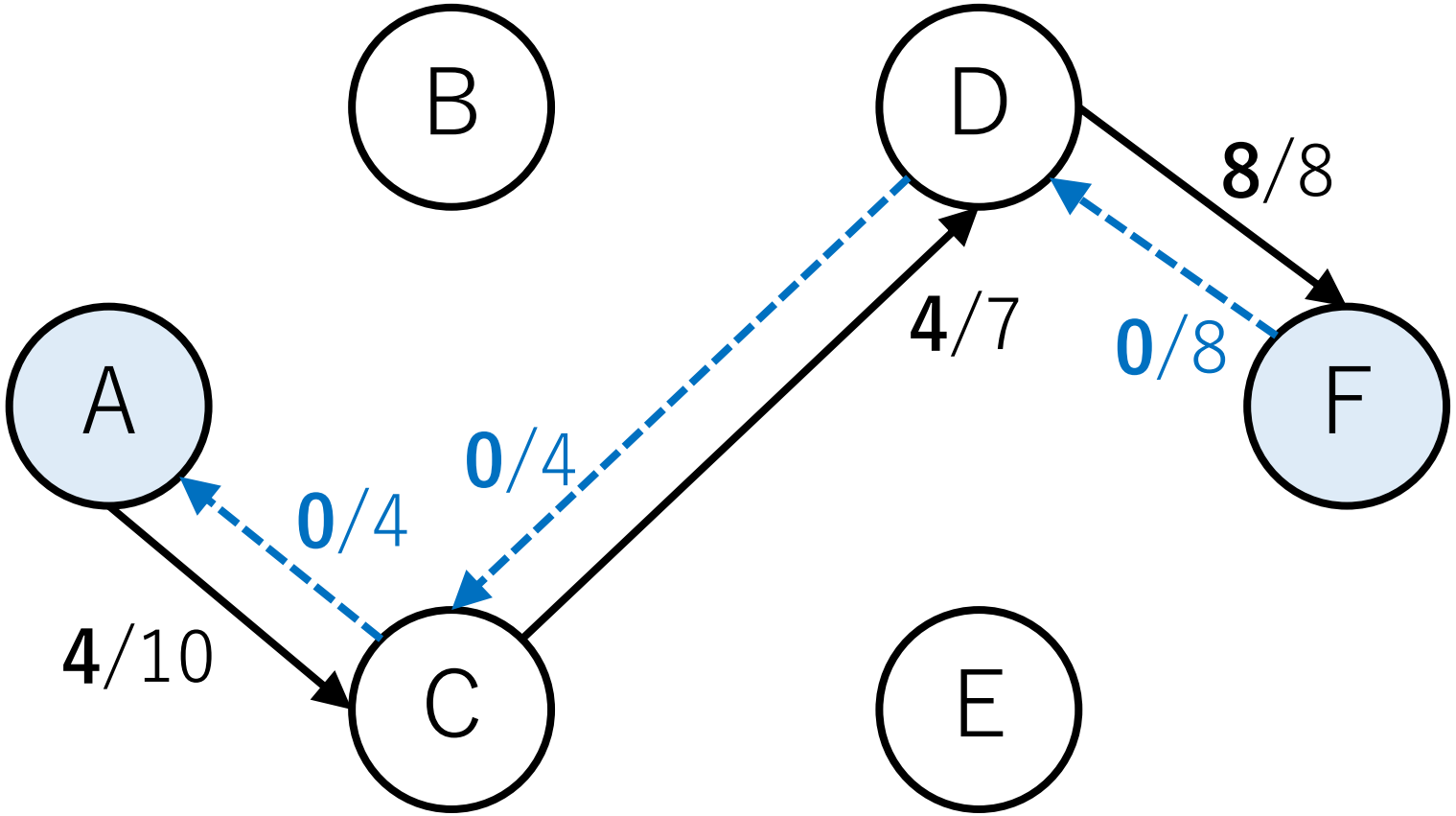
フォード・ファルカーソン法による例

4流す.



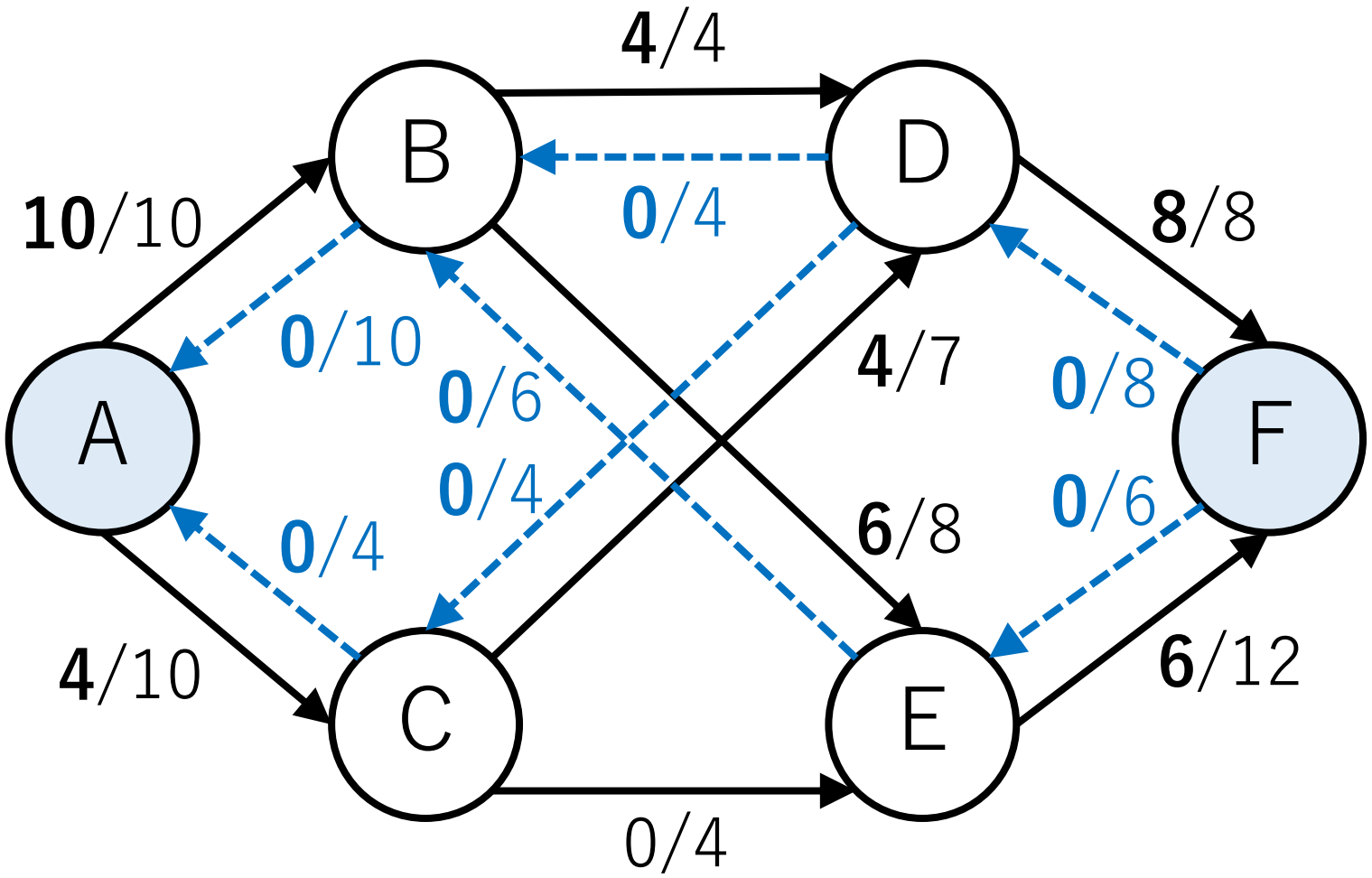
フォード・ファルカーソン法による例

逆方向の経路を設定，容量を更新.



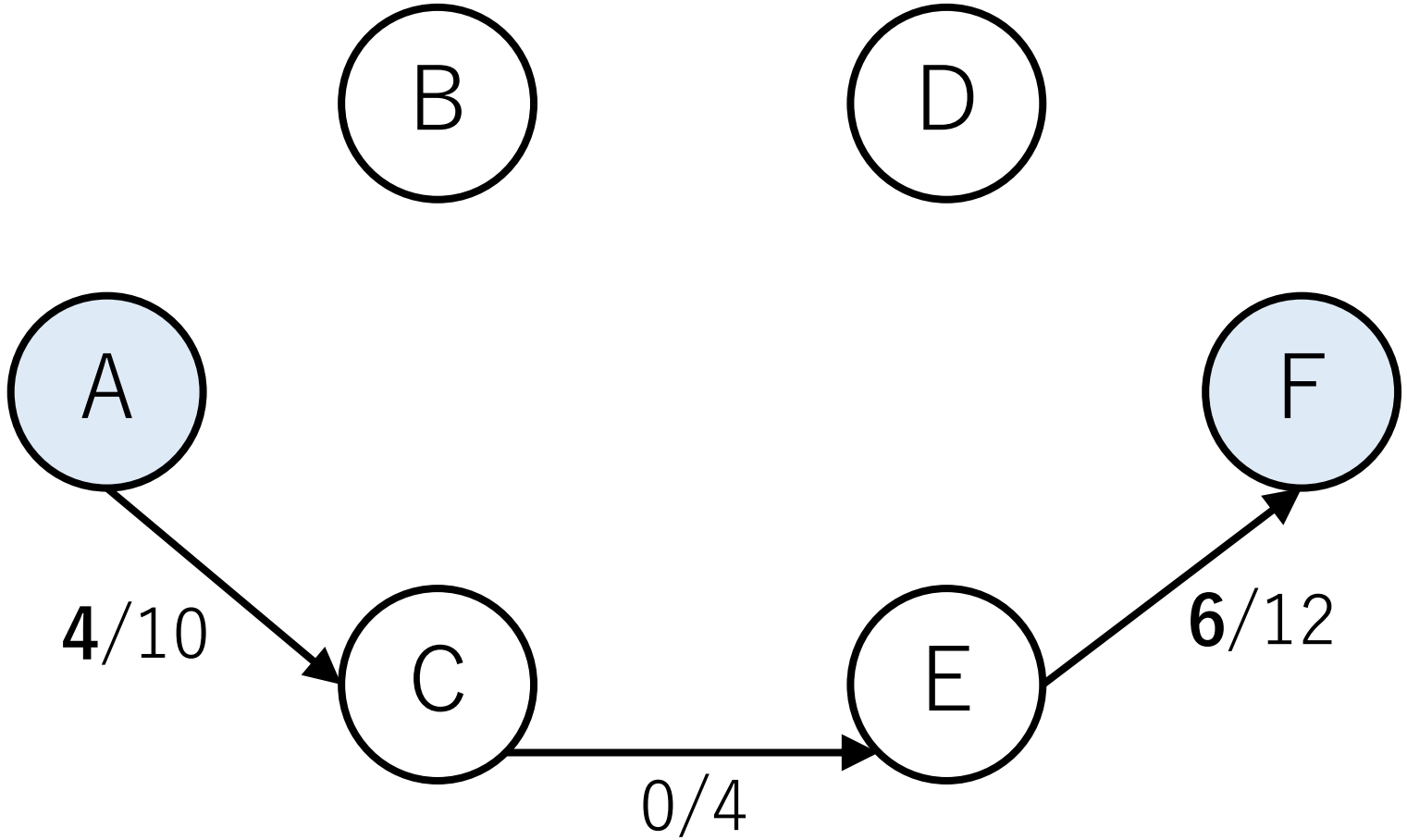
フォード・ファルカーソン法による例

現在までの状態.



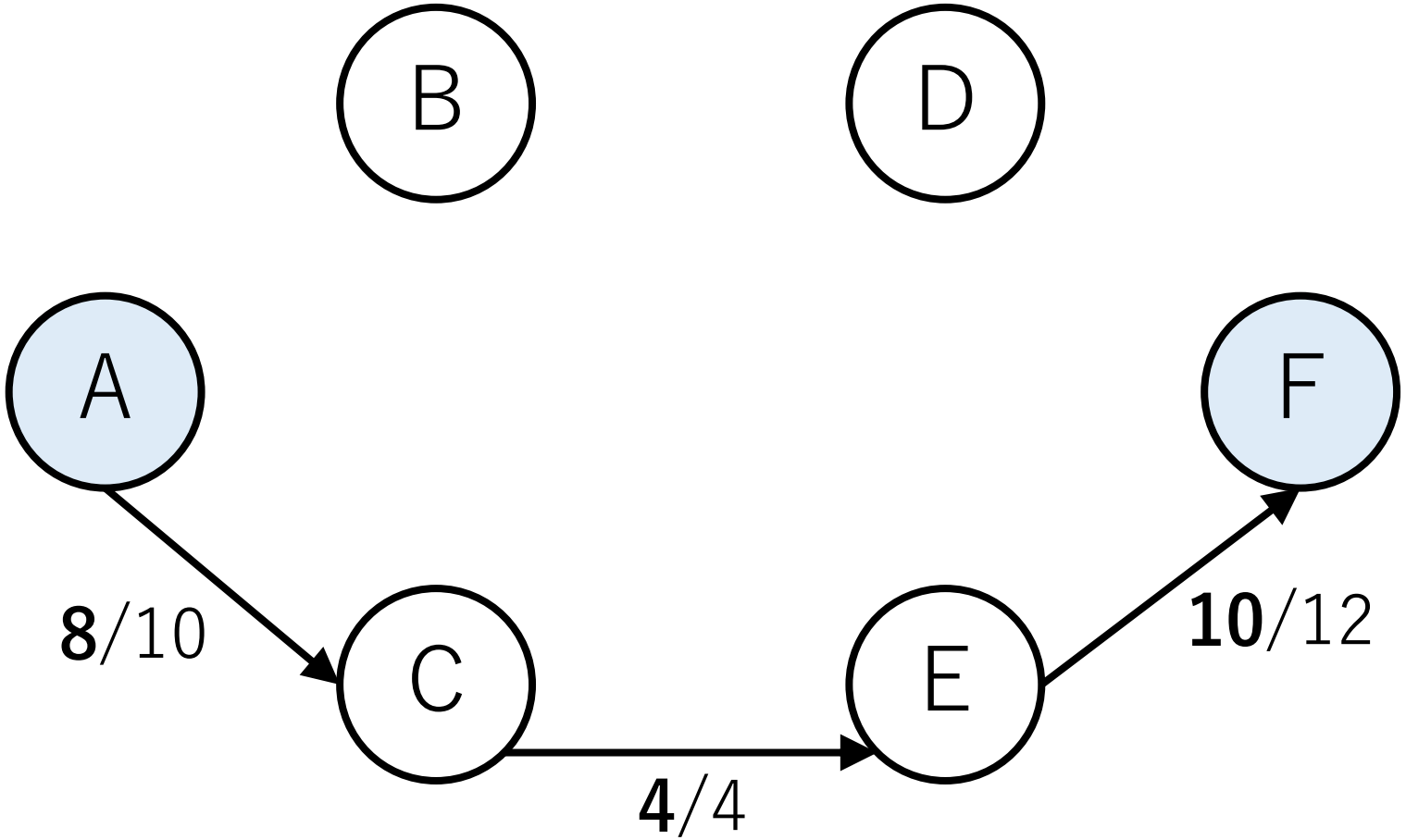
フォード・ファルカーソン法による例

A->C->E->Fを見つける．流せる量は4.



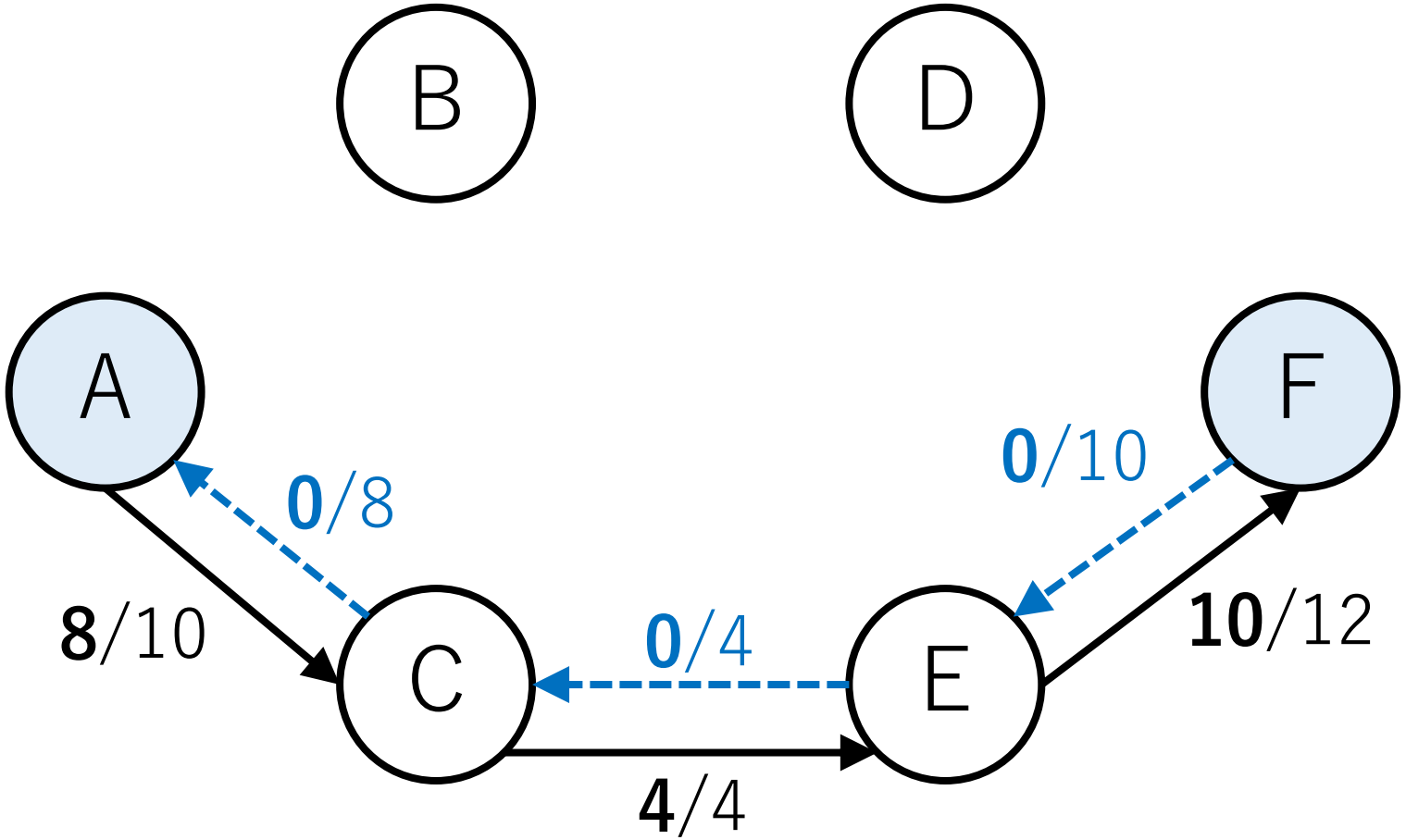
フォード・ファルカーソン法による例

4流す.



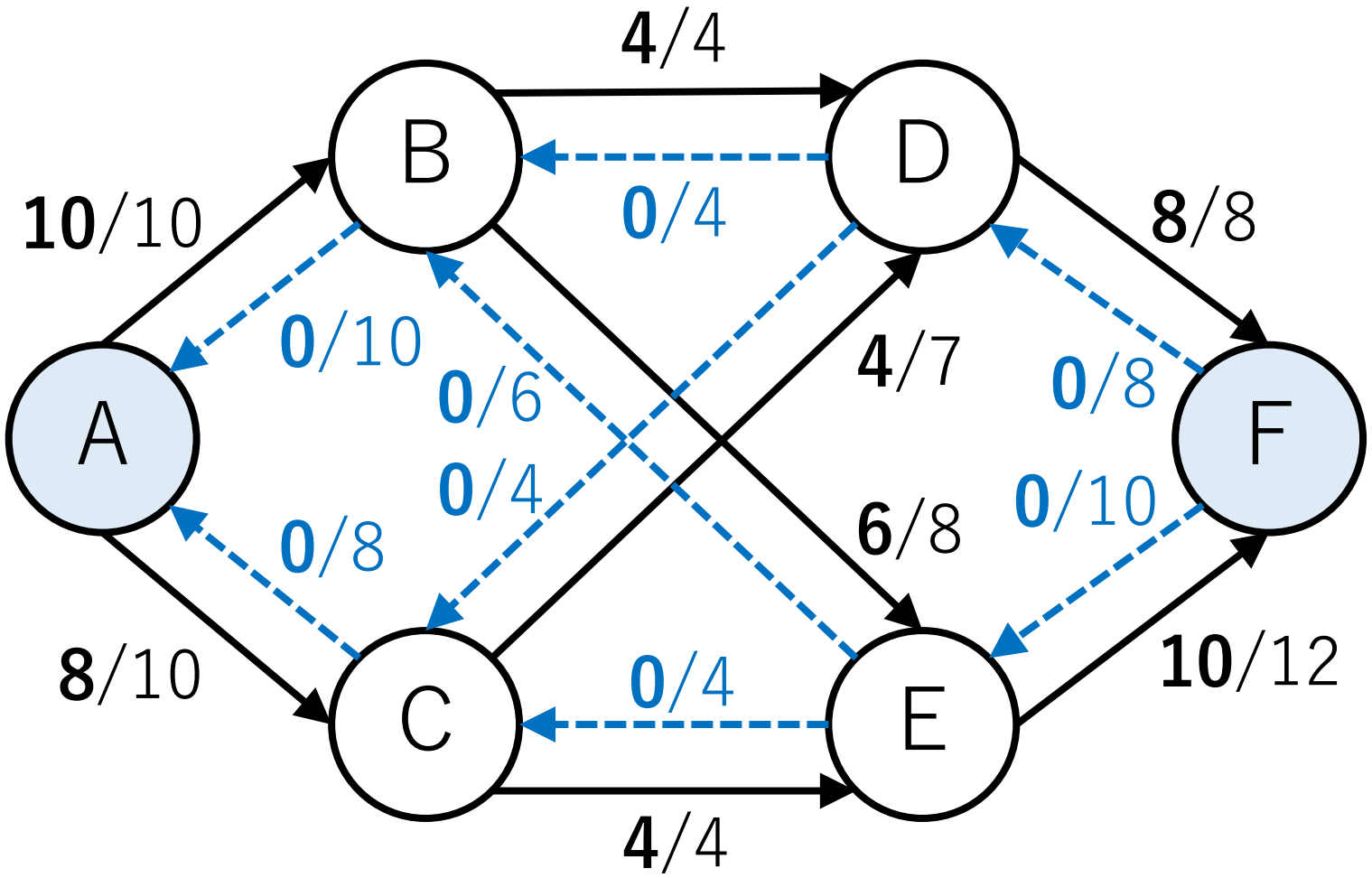
フォード・ファルカーソン法による例

逆方向の経路を設定，容量を更新.



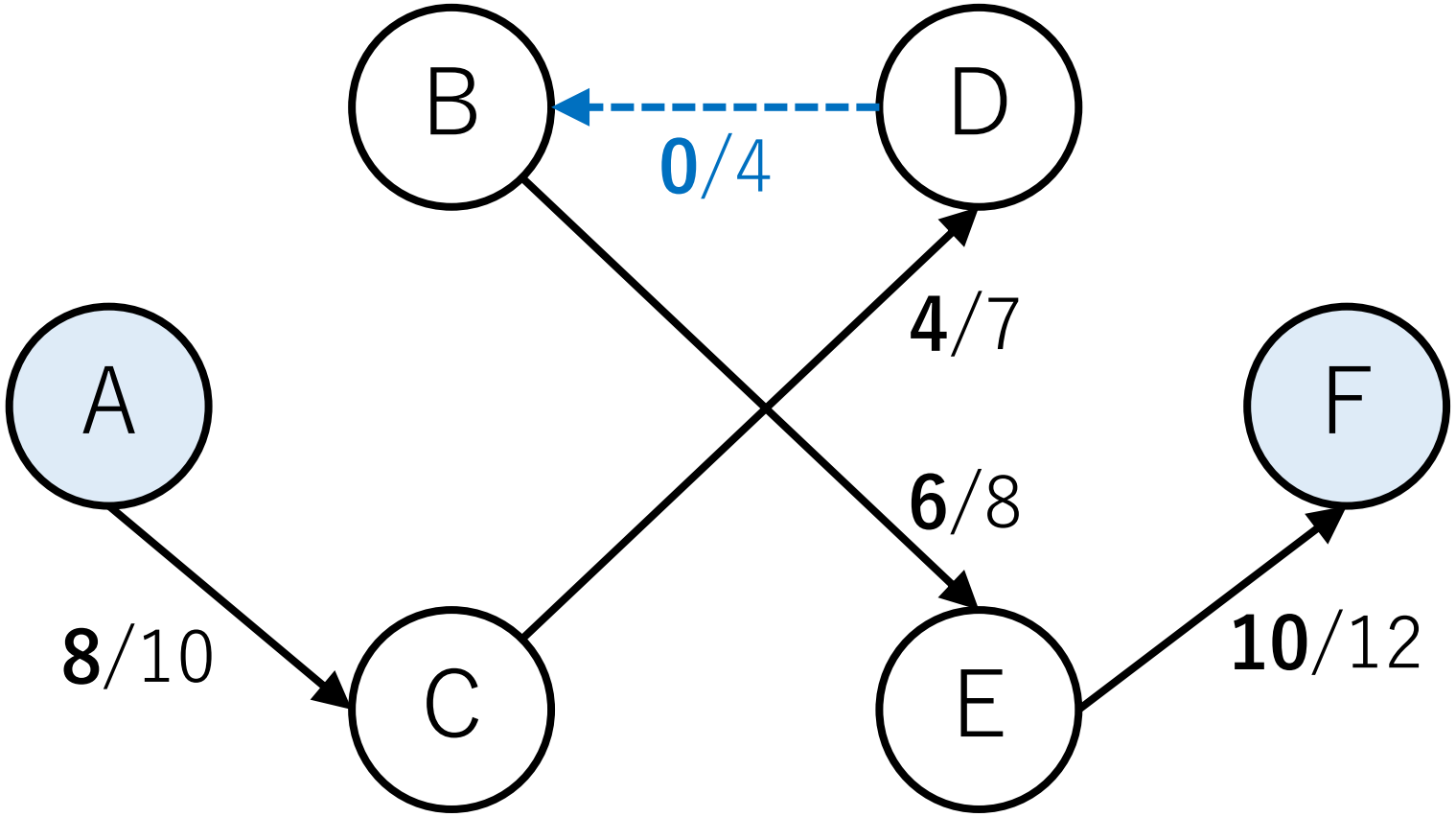
フォード・ファルカーソン法による例

現在までの状態．（ここまでは貪欲法の例と同じ．）



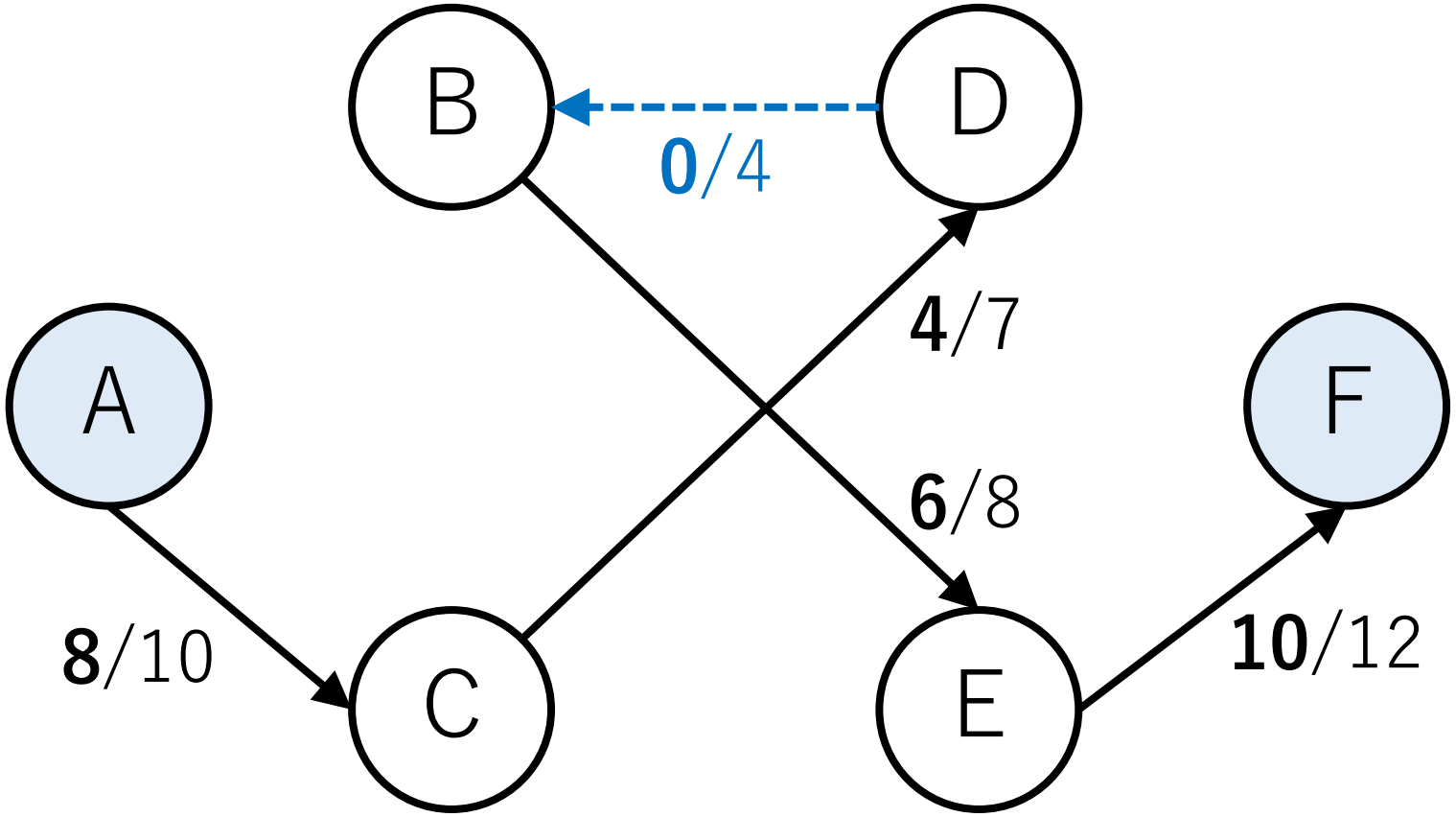
フォード・ファルカーソン法による例

まだ経路が存在する．A->C->D->B->E->F.



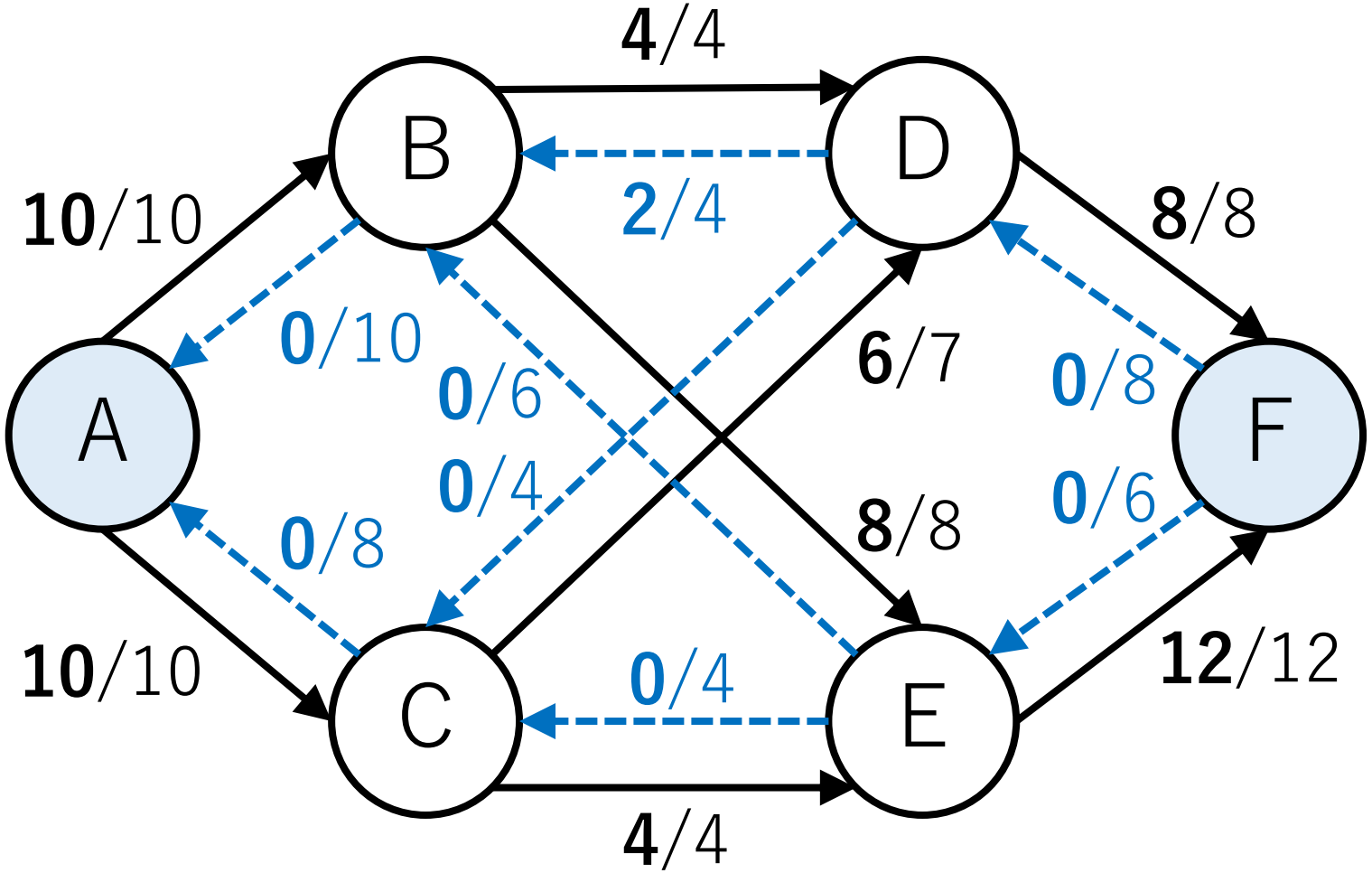
フォード・ファルカーソン法による例

流せる流量は2.



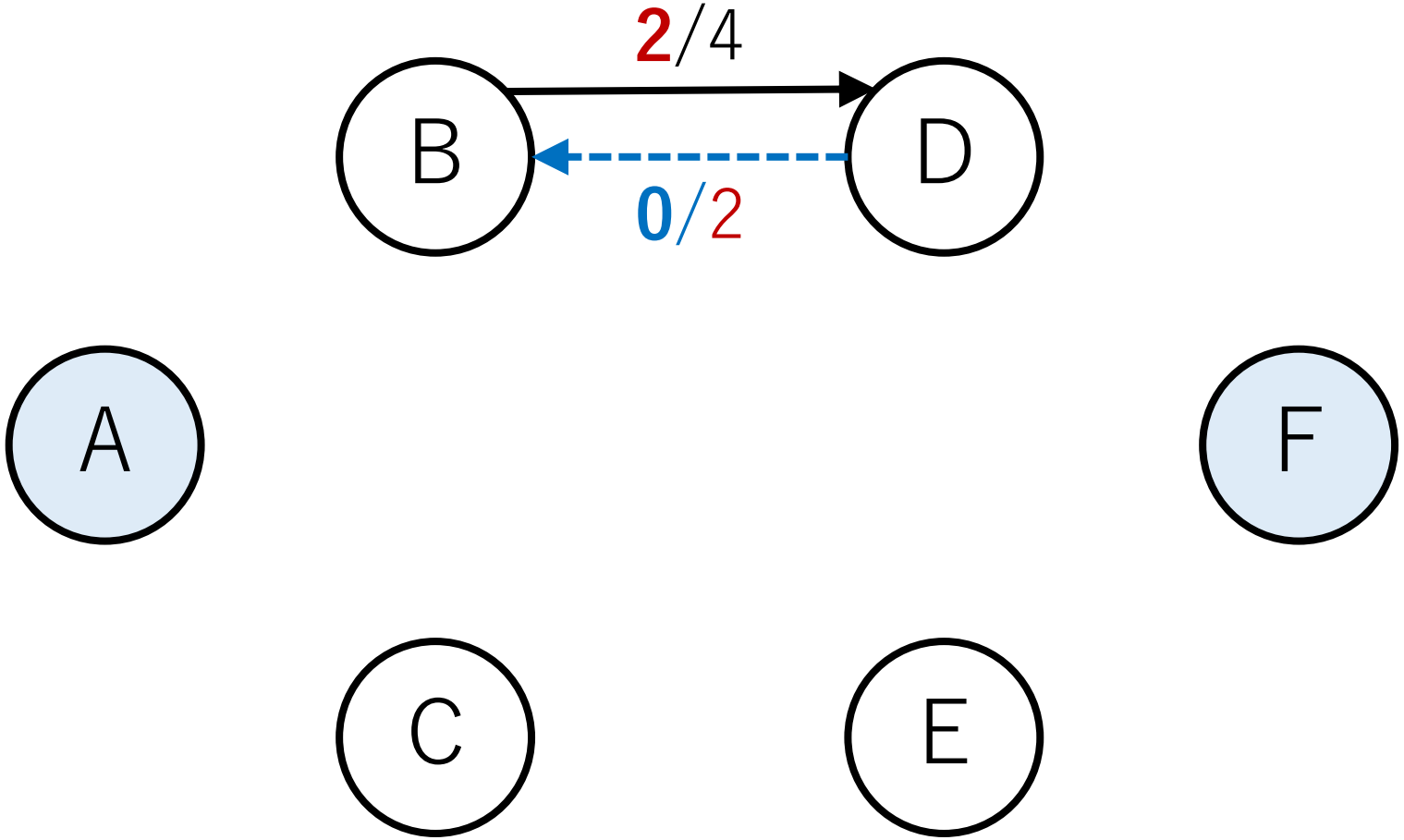
フォード・ファルカーソン法による例

A->C->D->B->E->Fの流量を入れて更新.



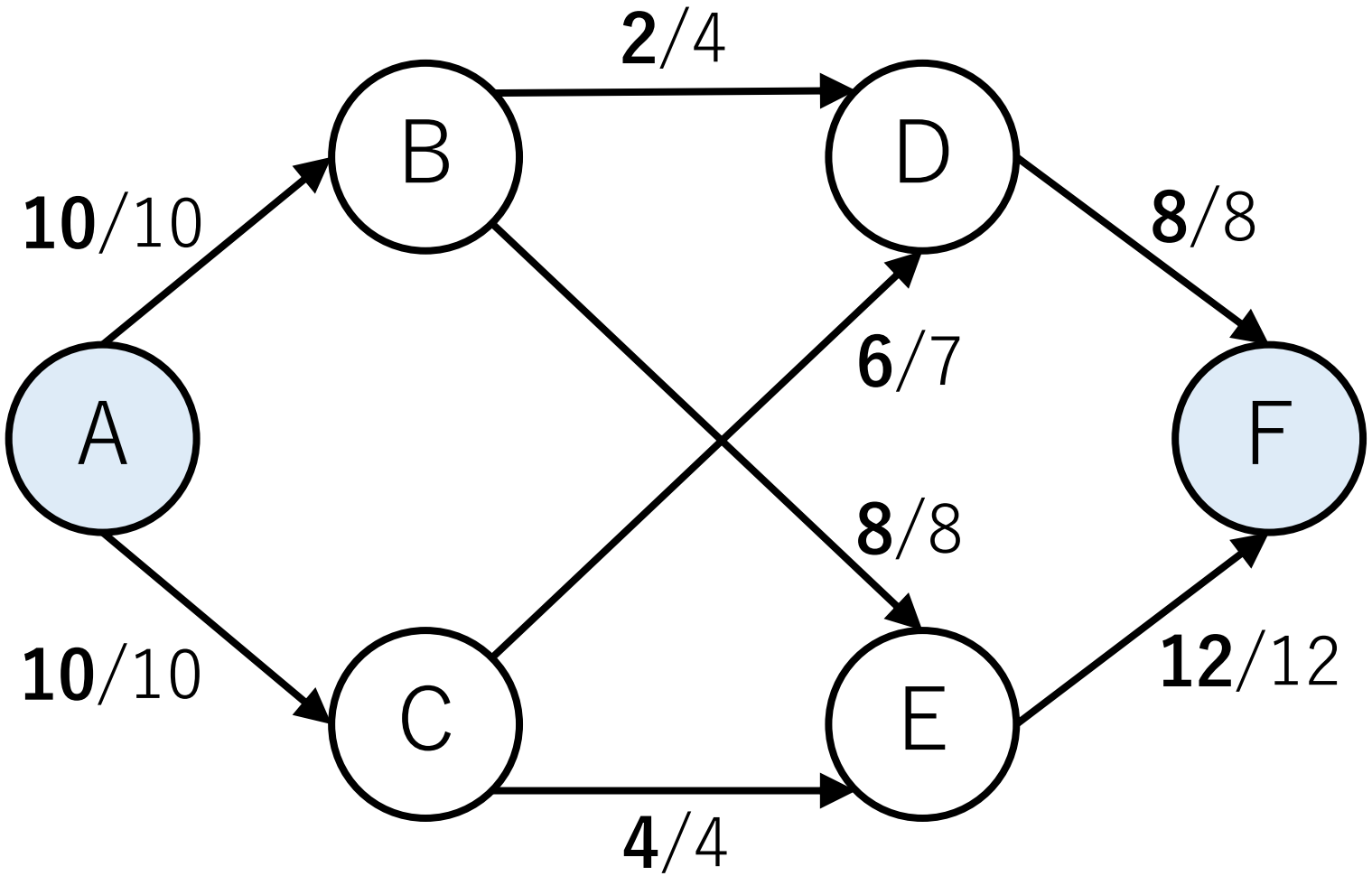
フォード・ファルカーソン法による例

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・ファルカーソン法の美しさ

こんな複雑なこと，考える必要ある？😅

フォード・ファルカーソン法の美しさ

こんな複雑なこと，考える必要ある？😅

「ノード間を逆にたどる」経路を考えることで，流しすぎたフローを逆に戻す処理を実現している．

これによって，流す経路を導出するのをDFS**に一本化できている！**

フローを逆に戻す時も特別な処理をする必要がない．

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関数
# 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

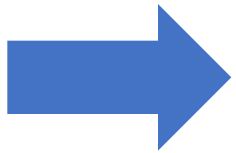
```
def dfs_ff(s, e, flow):
    ...
    for i in range(V):
        if [まだ訪問してないノードで流量に余裕がある]:
            # 今までの最大流量とs – i間の流量のうち,
            # 小さい方を使ってdfs_ffを再帰で呼び出して
            # DFS．返り値は新しく流せる経路の容量.
            f = [dfs_ffを再帰で呼び出す]
```

フォード・ファルカーソン法実装例

```
def dfs_ff(s, e, flow):
        …
                f = [dfs_ffを再帰で呼び出す]
                if f > 0:  # 新しく流せる経路があった場合
                        # 順方向，逆方向の容量を更新
                        capacity[s][i] -= f
                        capacity[i][s] += f
                        return f
```

実行結果（1パス分終了後のcapacity）

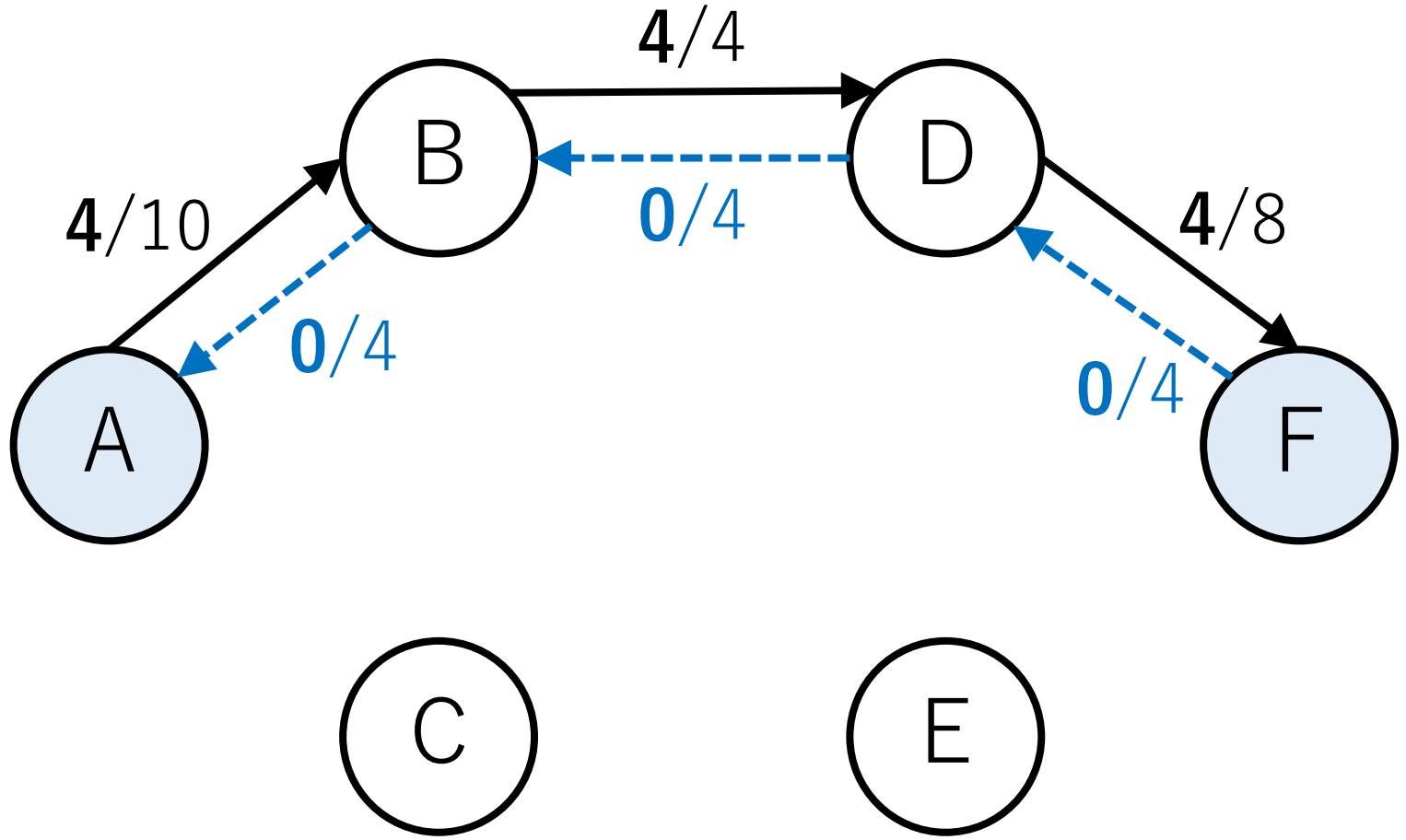
```
[[0, 10, 10, 0, 0, 0],
[0, 0, 0, 4, 8, 0],
[0, 0, 0, 7, 4, 0],
[0, 0, 0, 0, 0, 8],
[0, 0, 0, 0, 0, 12],
[0, 0, 0, 0, 0, 0]]
```



```
[[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.

（授業中に説明した順番通りに経路の容量値が変化するとは限りませんが，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時間計算量

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も最大流が1つしか増やせない．-> 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$O(F|E|)$.

フォード・ファルカーソン法の制限

経路の容量が整数か有理数である必要がある.

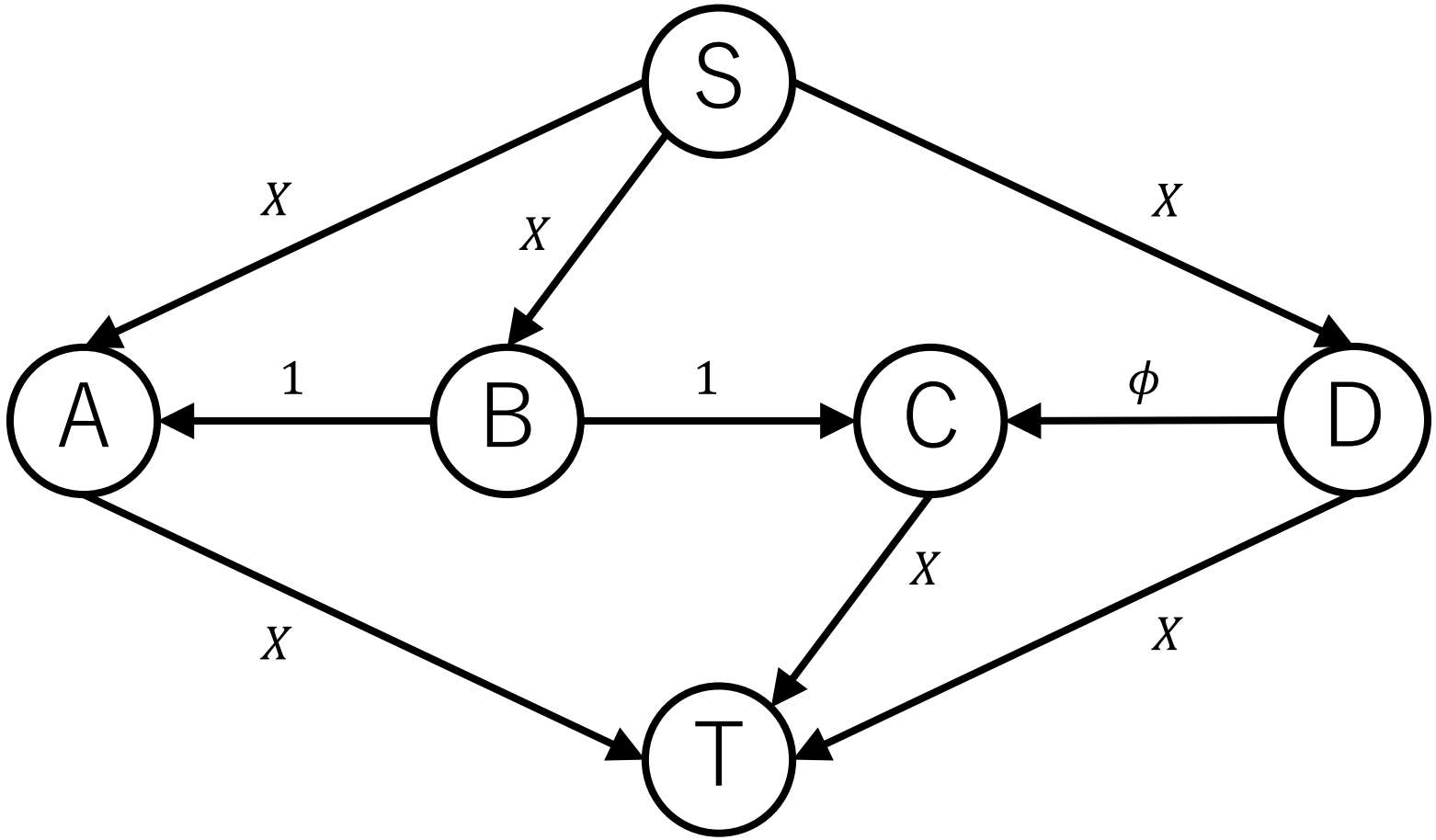
有理数の場合，通分して分母を払うことで，整数の場合と同じように扱うことができる.

無理数の場合，停止せず最適解に収束しないことがある.

https://courses.engr.illinois.edu/cs473/fa2012/notes/22-maxflowalgs.pdf

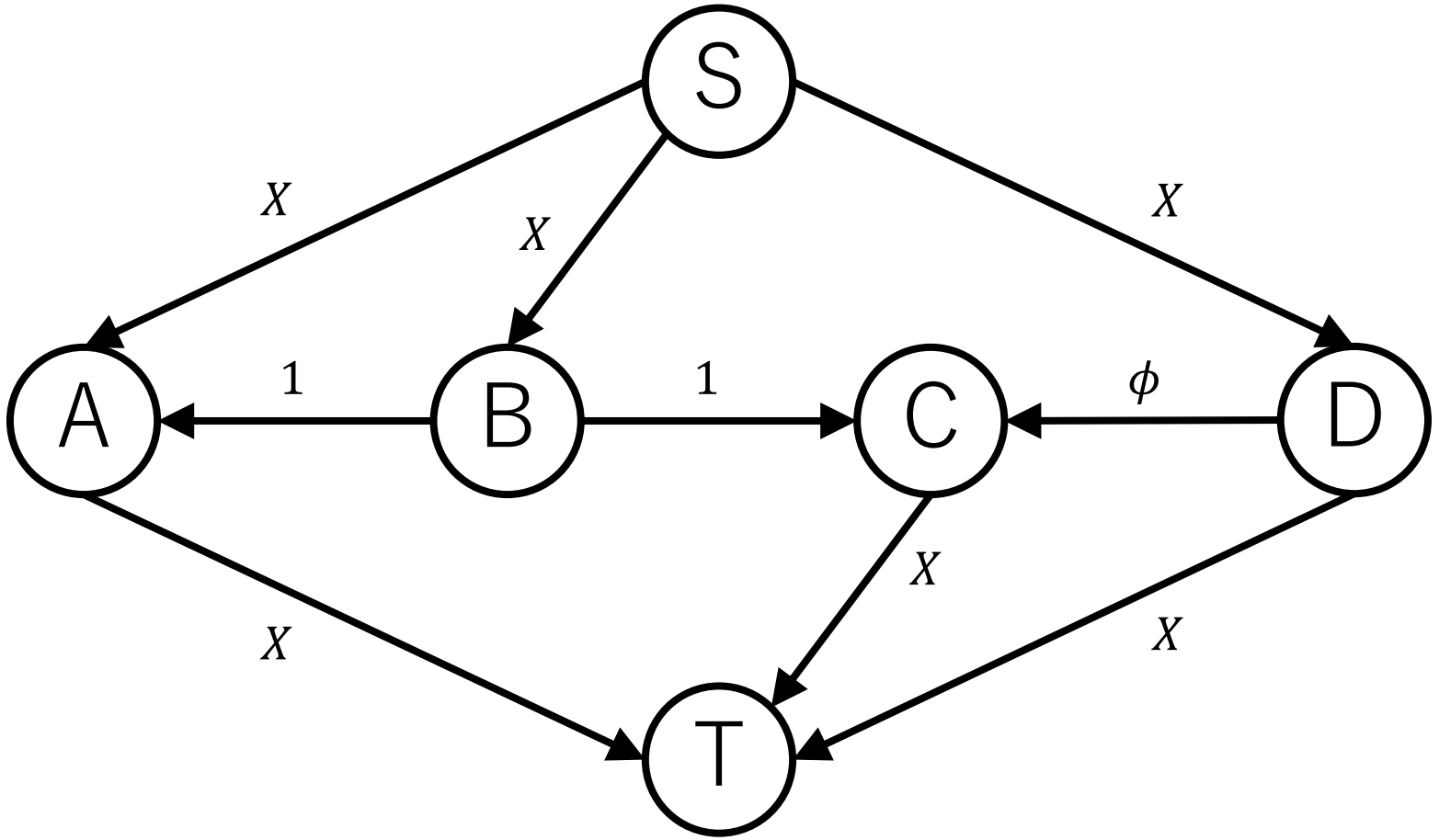
無理数の容量がある場合

以下のようなグラフを考える．Xは十分に大きな値とする．



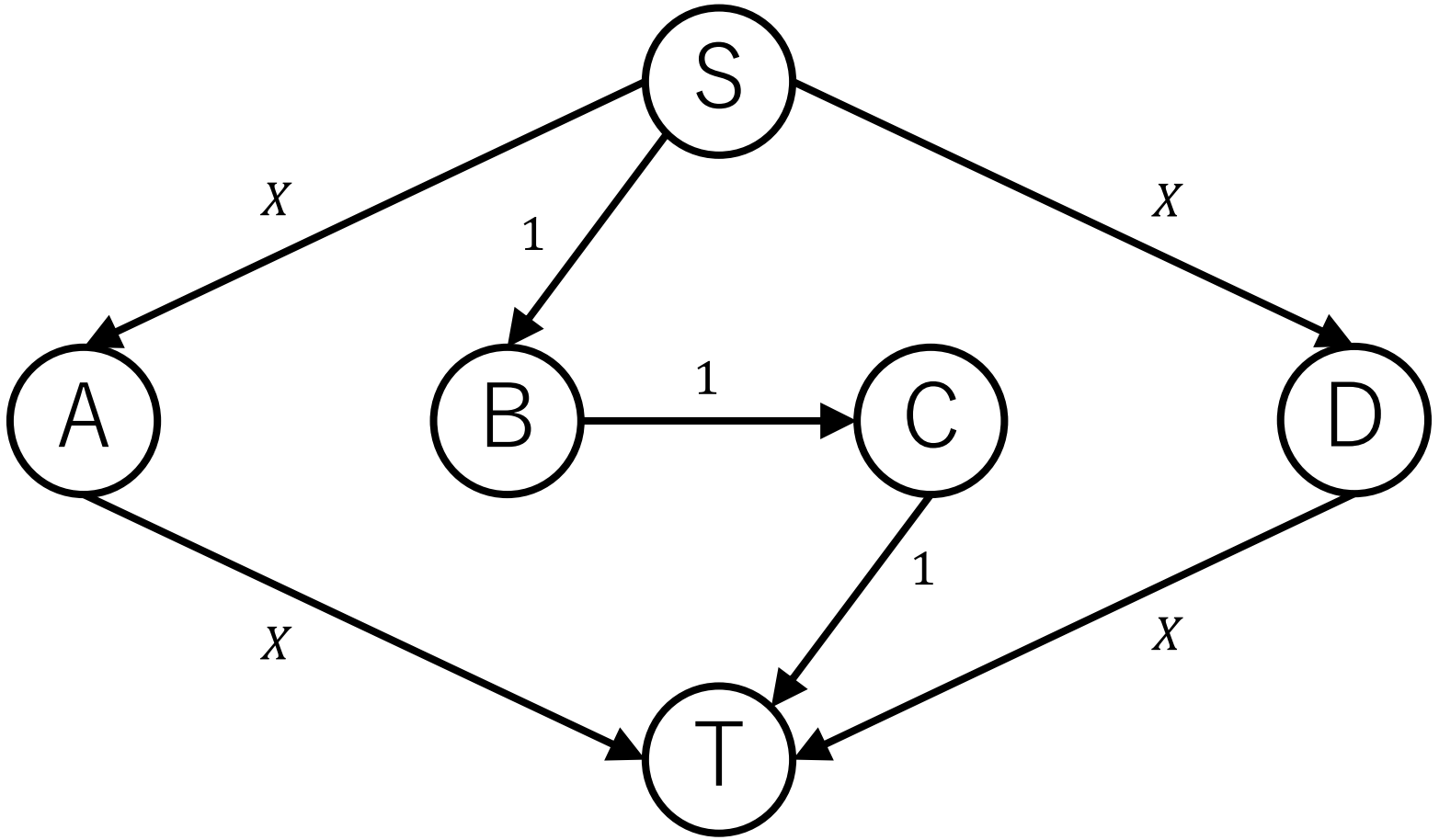
無理数の容量がある場合

また，$\phi = 1 - \phi^2$となるようなϕとする（$\phi = (\sqrt{5} - 1)/2$）．



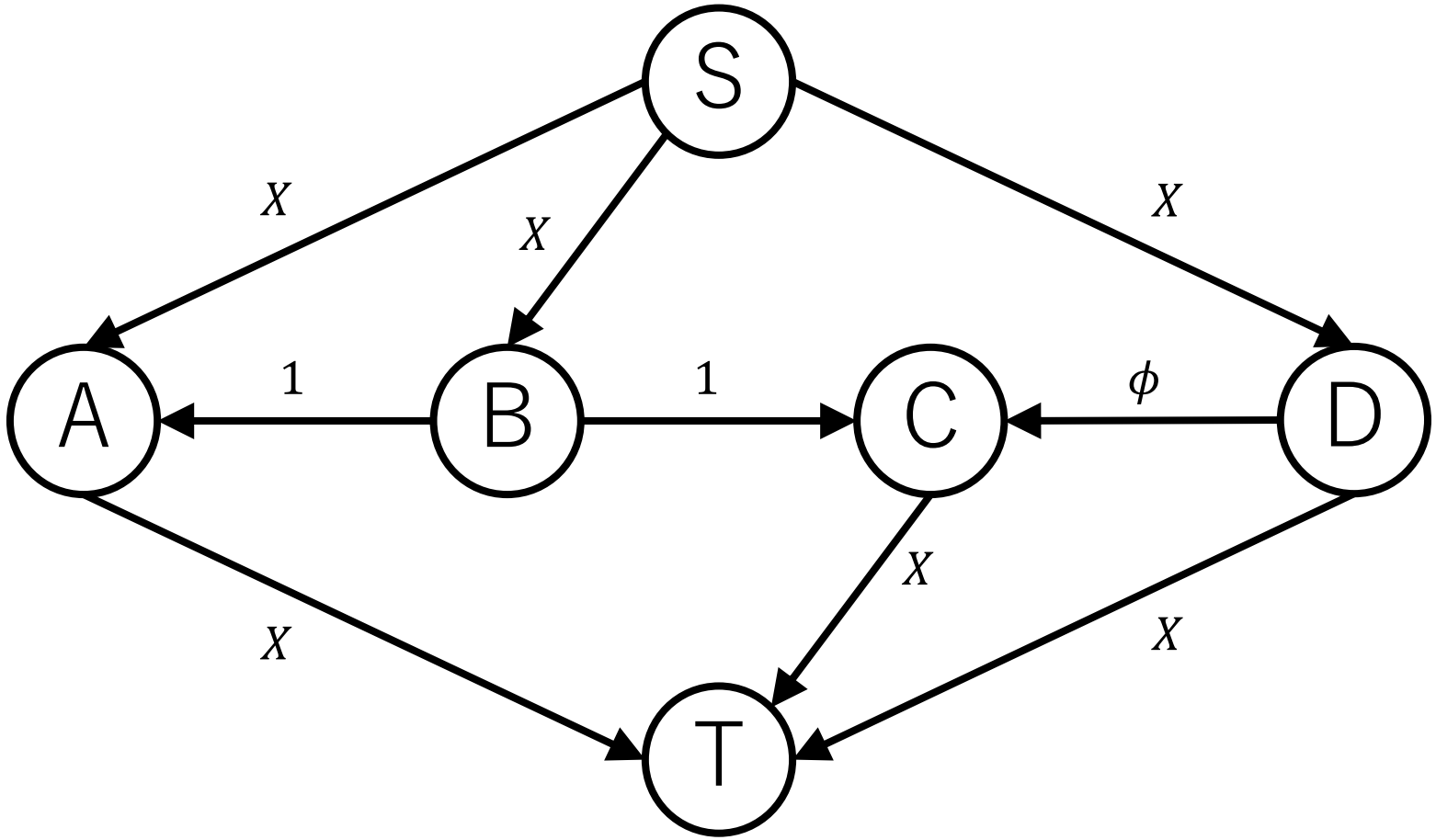
無理数の容量がある場合

このグラフの最大流は$2X + 1$.



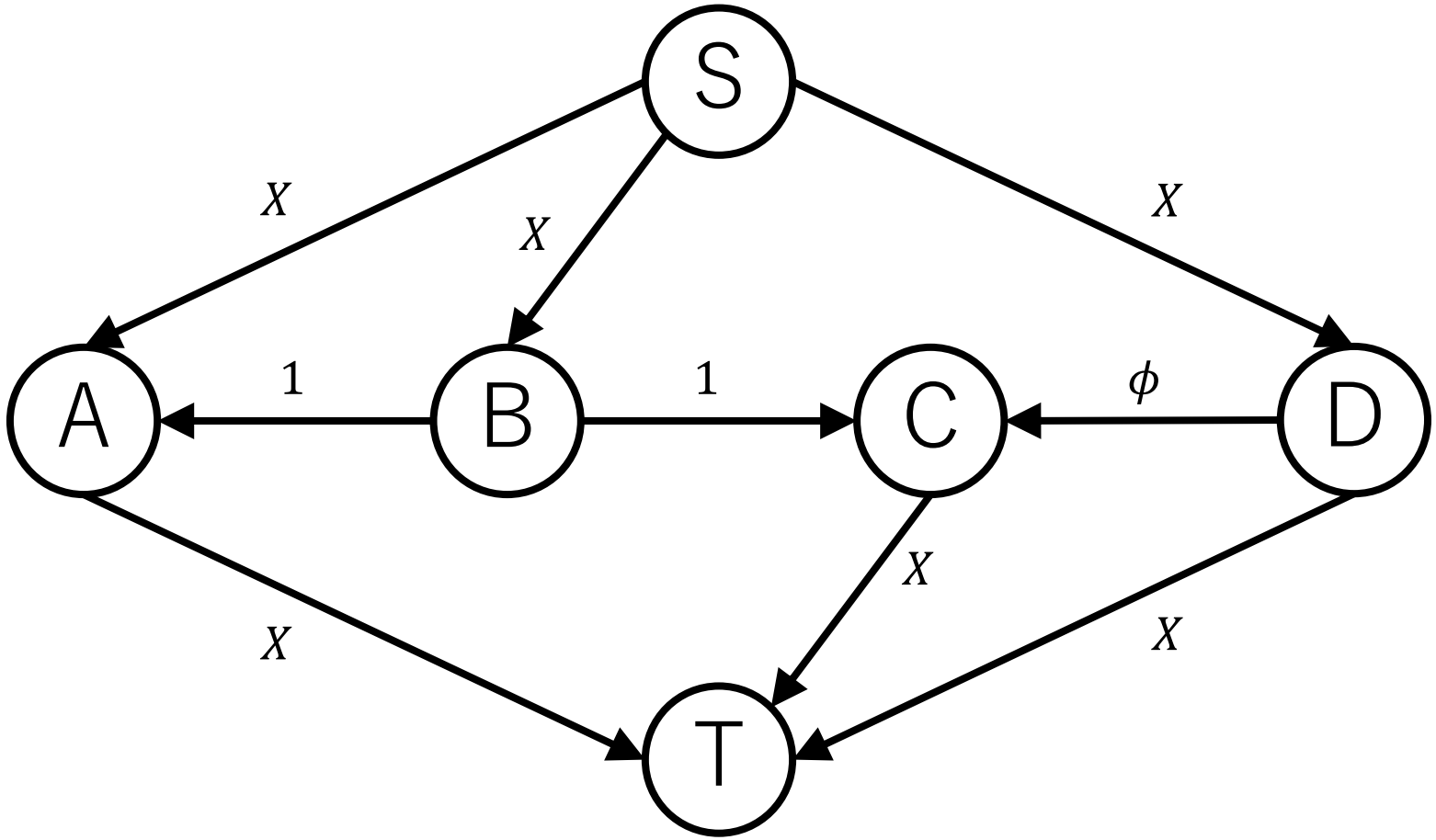
無理数の容量がある場合

うまく行かない場合を考えてみよう.



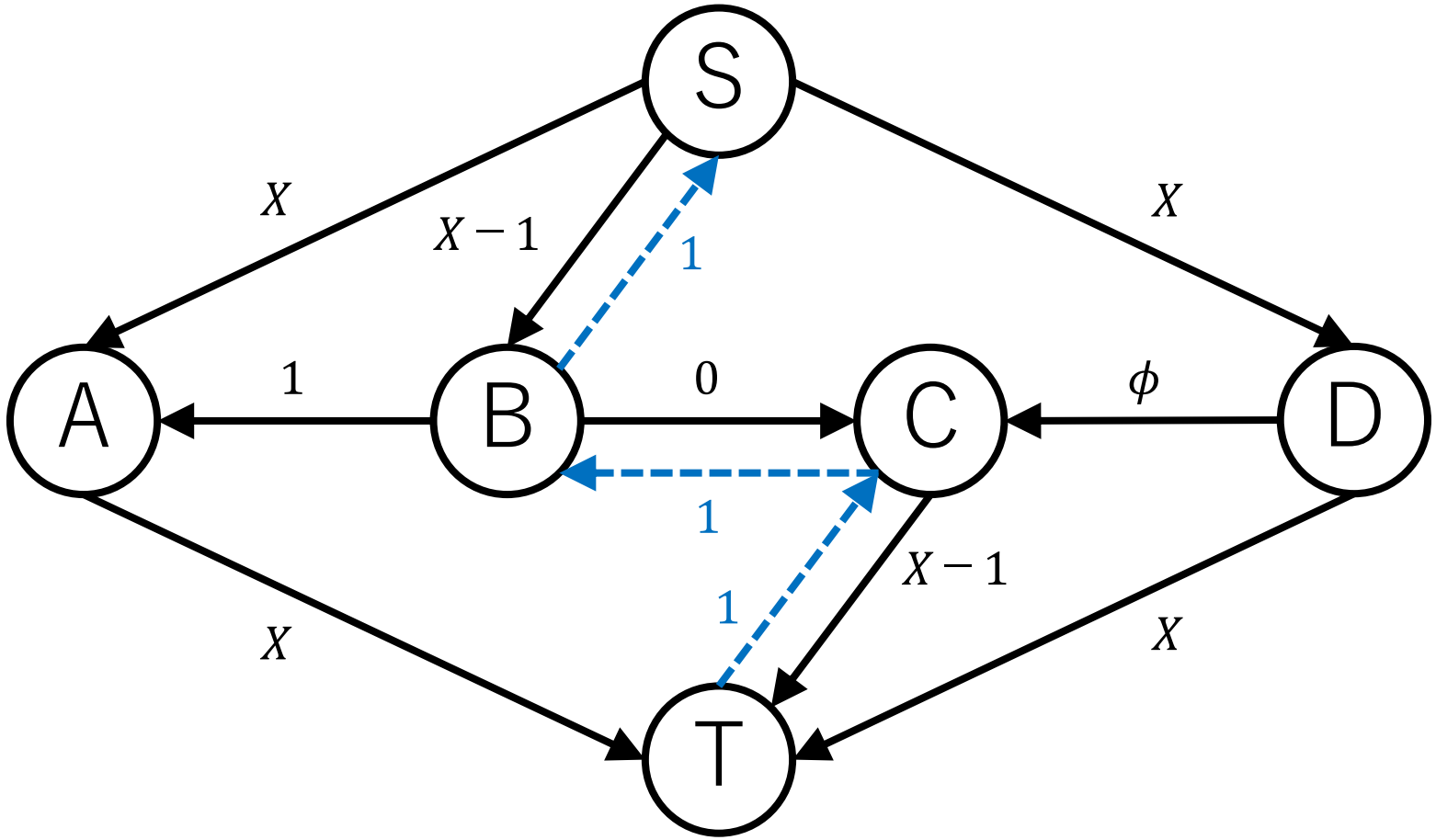
無理数の容量がある場合

(0) S->B->C->Tに1流す.



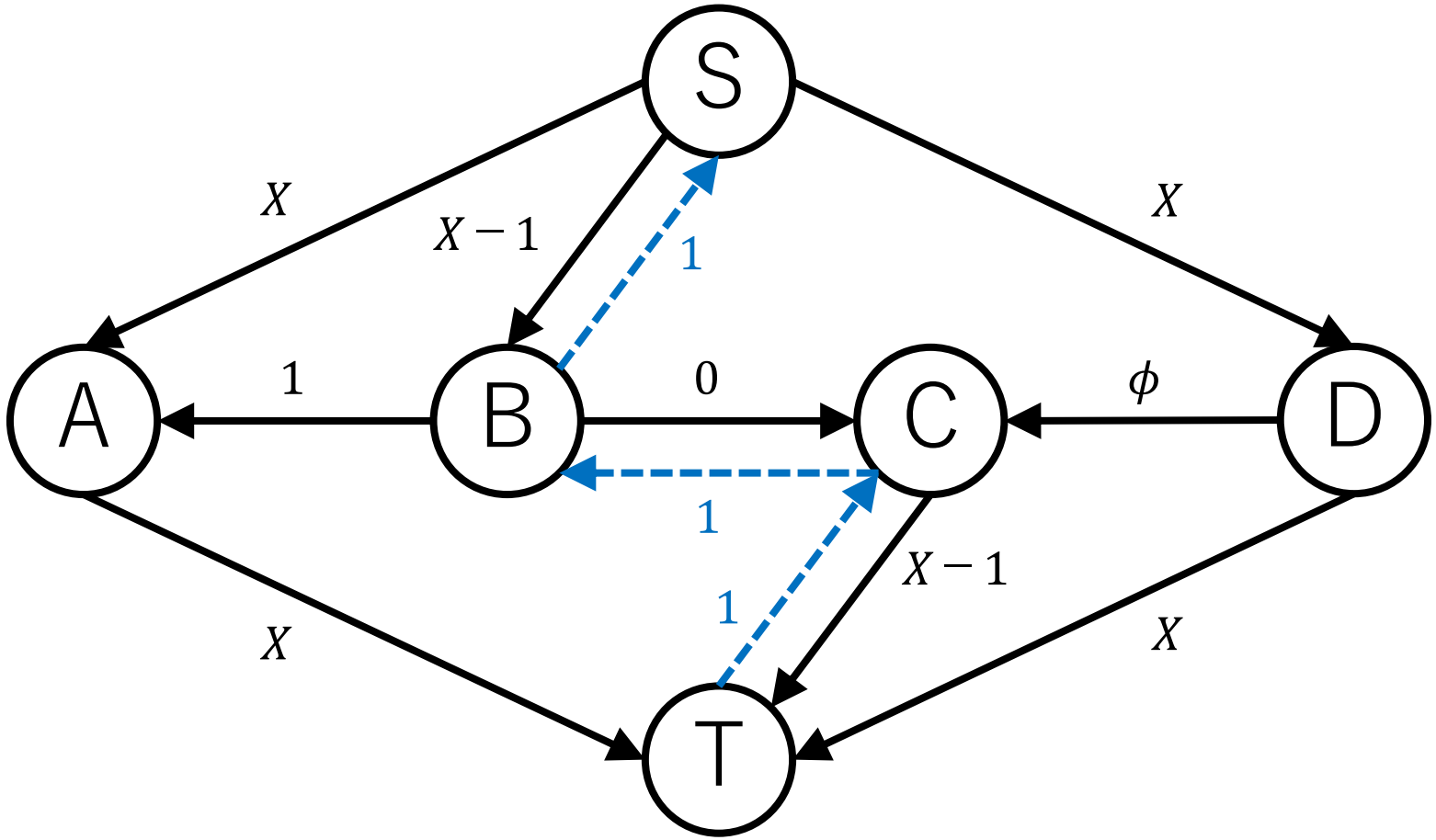
無理数の容量がある場合

S->B->C->Tに1流した後の状態.



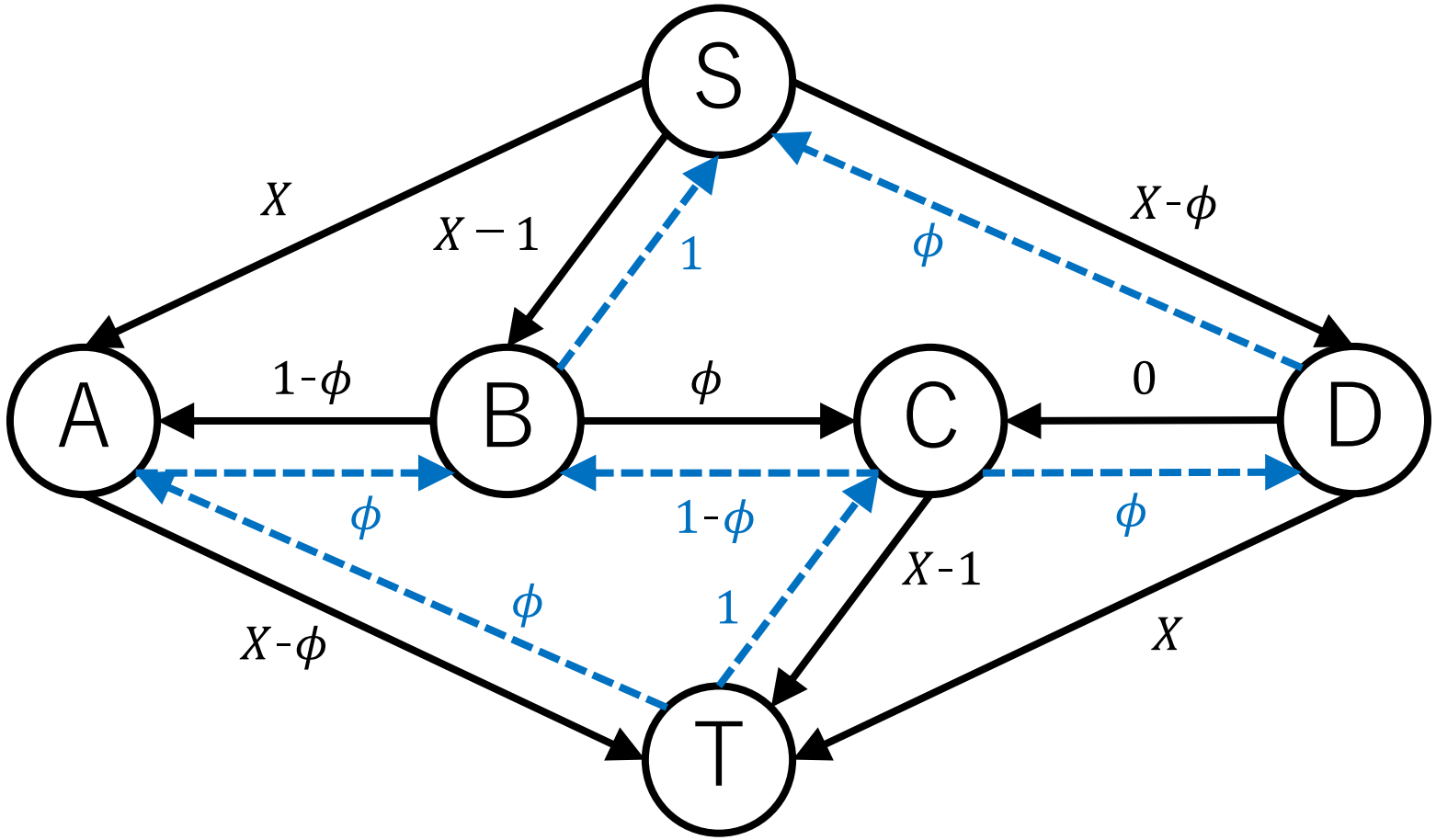
無理数の容量がある場合

(1) S->D->C->B->A->Tにϕ流す.



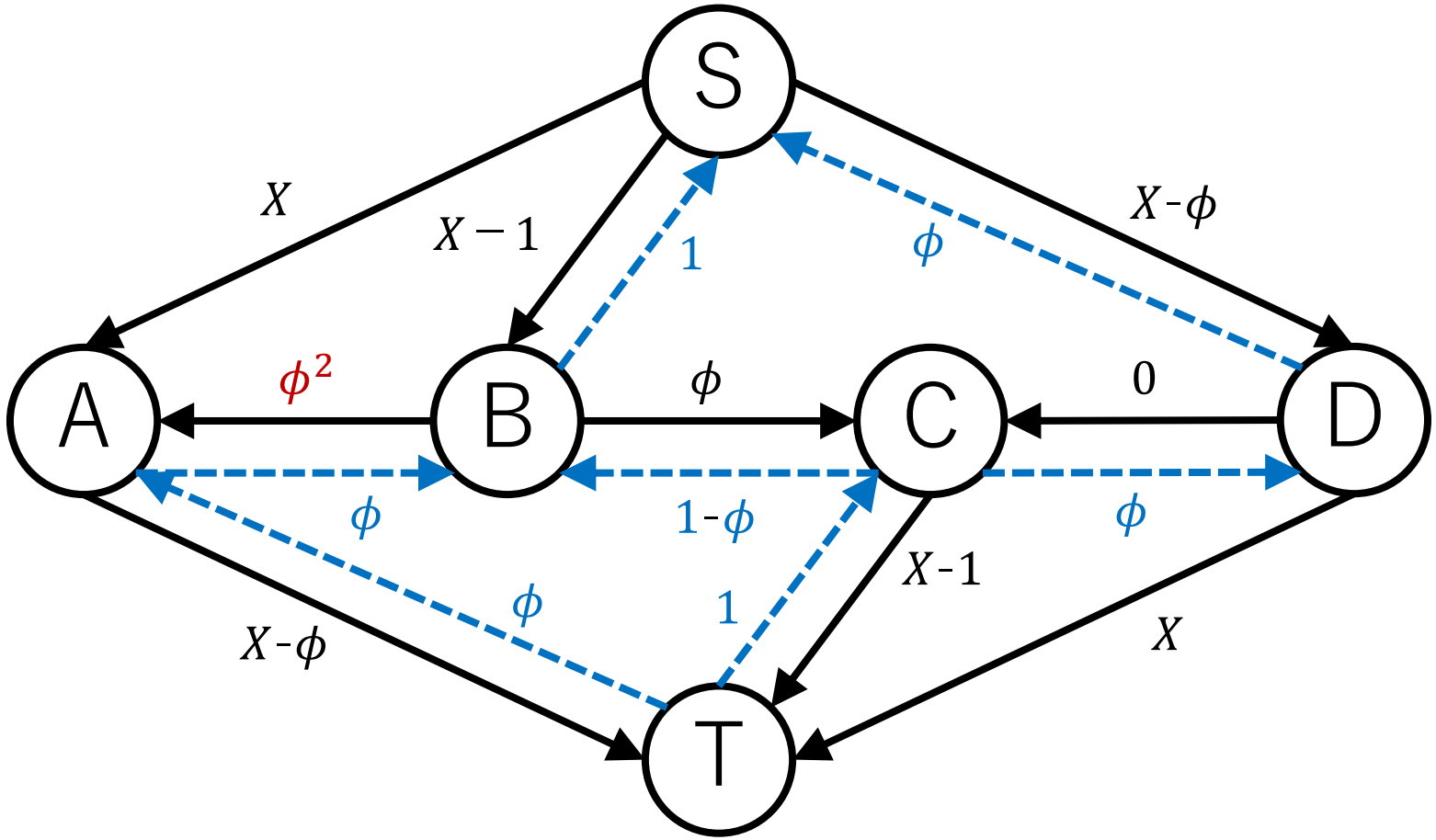
無理数の容量がある場合

S->D->C->B->A->Tにϕ流したあとの状態.



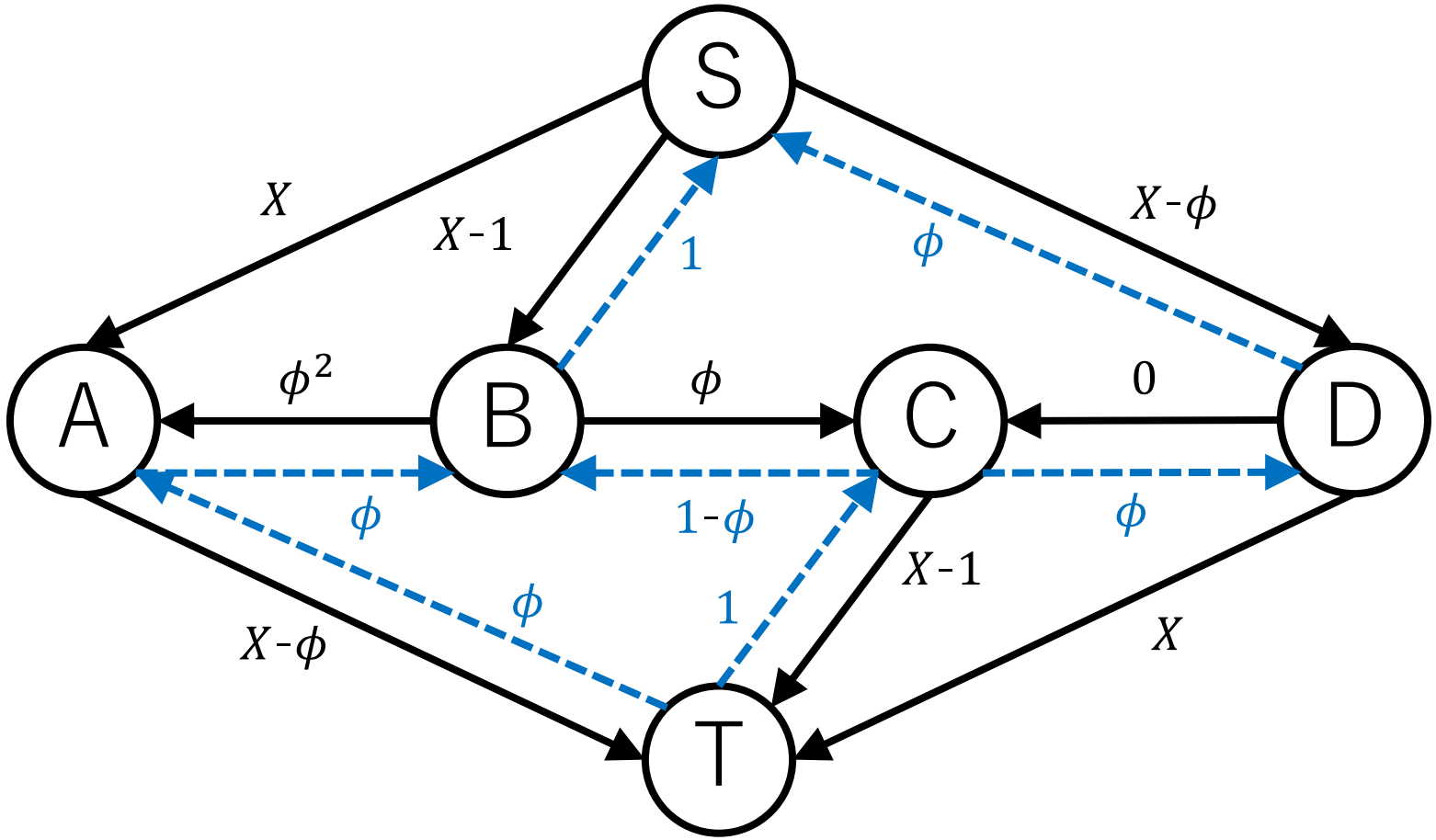
無理数の容量がある場合

$\phi^2 = 1 - \phi$であることを利用する.



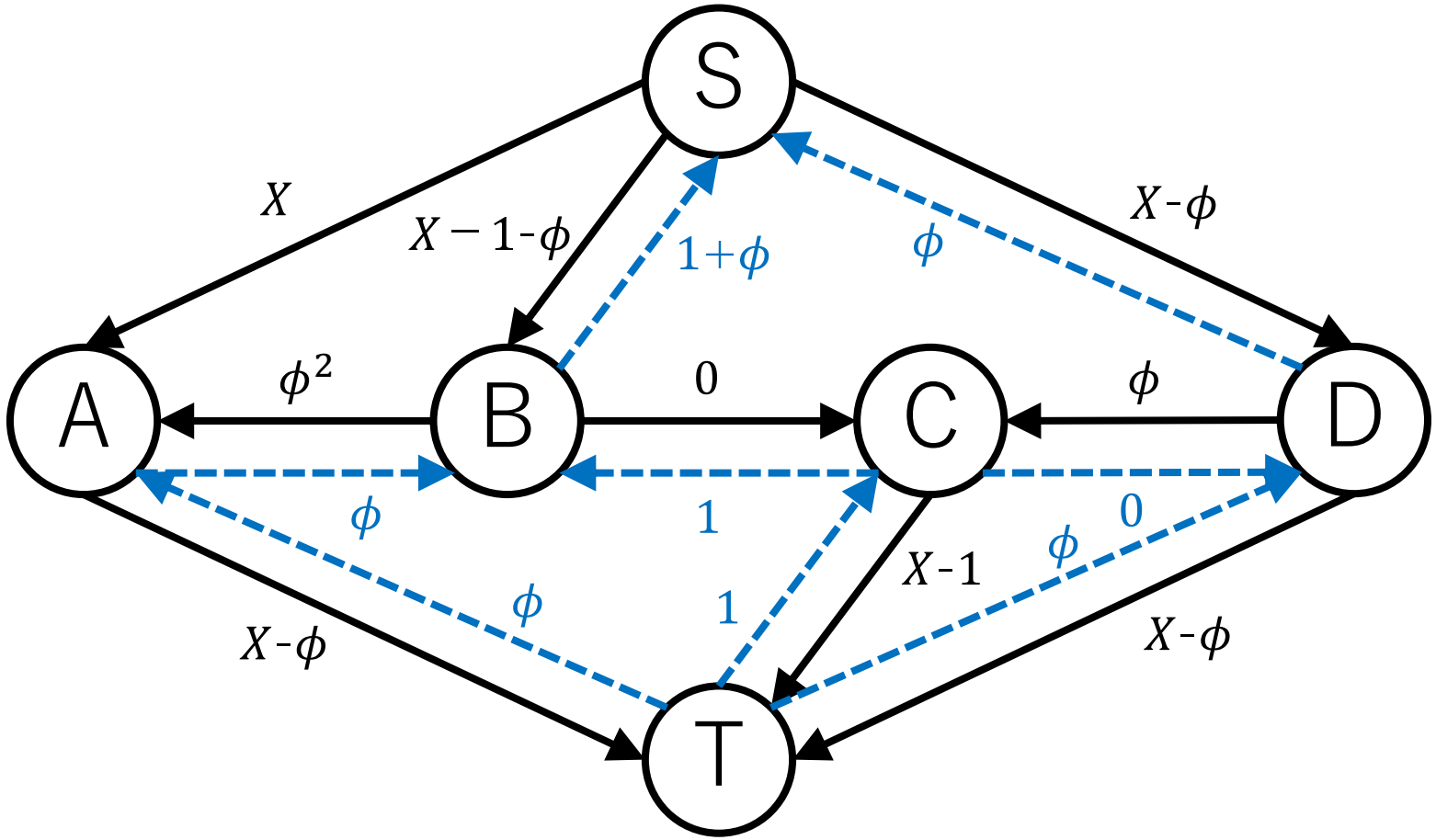
無理数の容量がある場合

(2) S->B->C->D->Tにϕ流す.



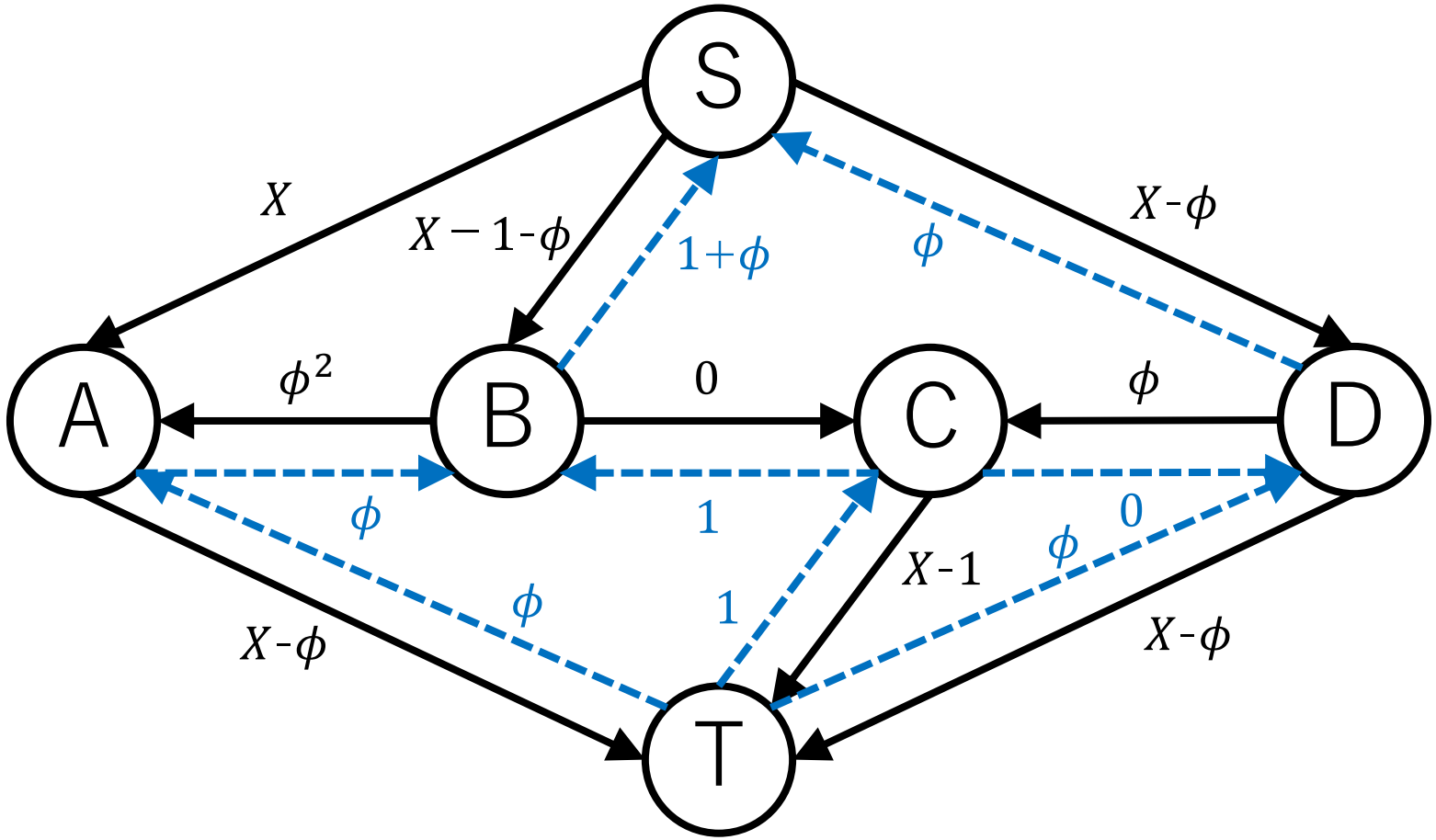
無理数の容量がある場合

S->B->C->D->Tにϕ流したあとの状態.



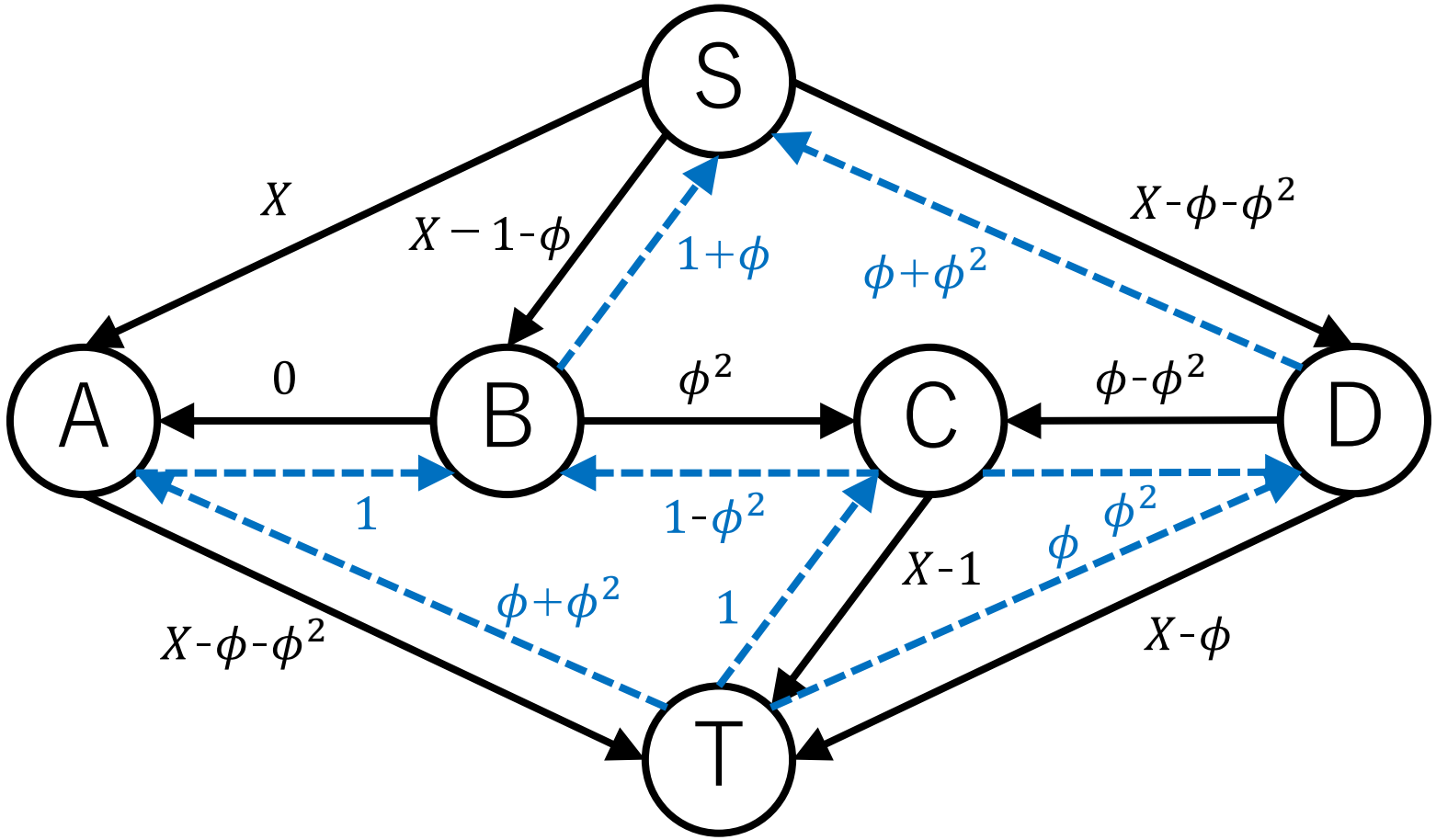
無理数の容量がある場合

(3) S->D->C->B->A->Tにϕ^2流す.



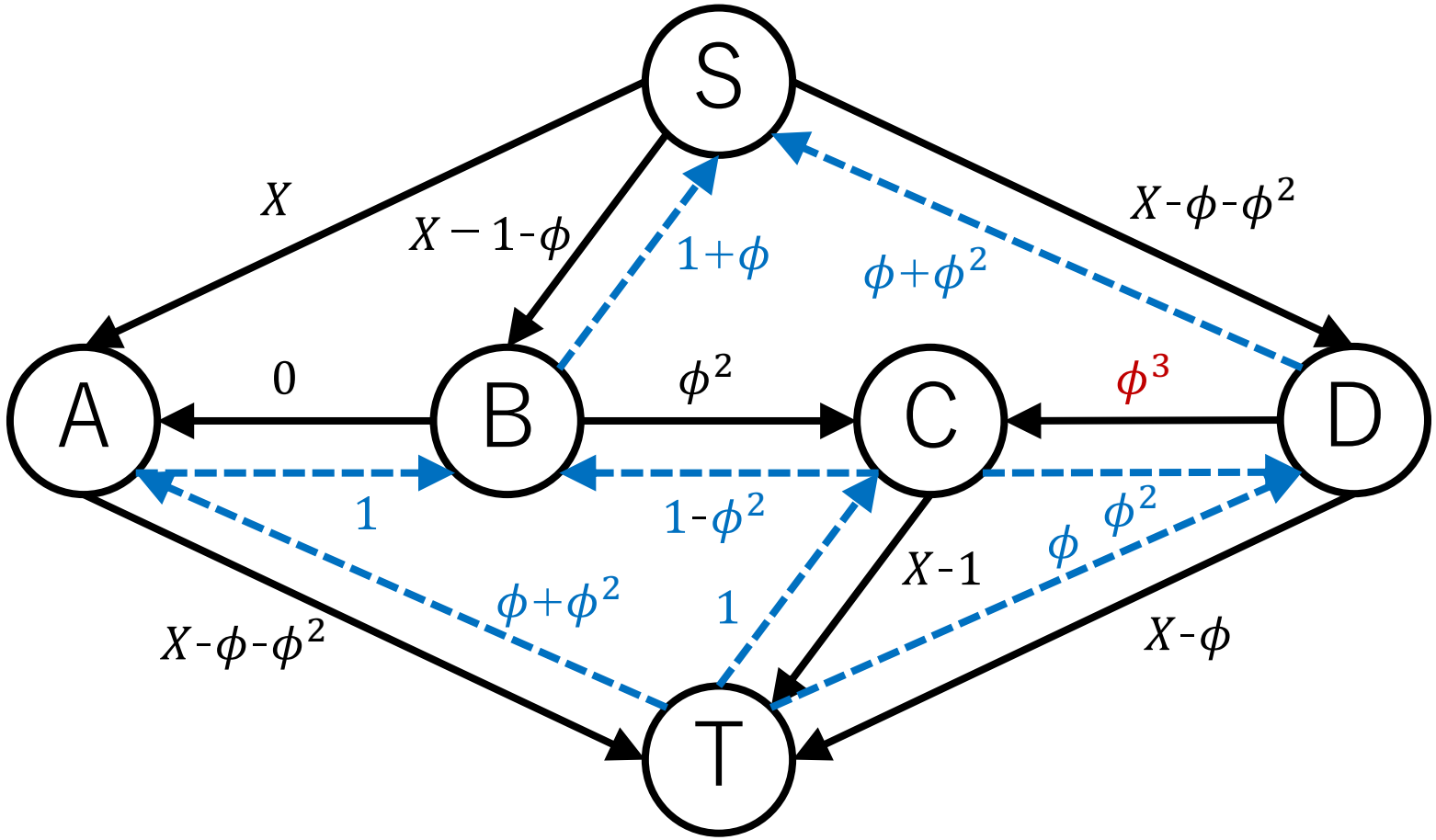
無理数の容量がある場合

S->D->C->B->A->Tにϕ^2流したあとの状態.



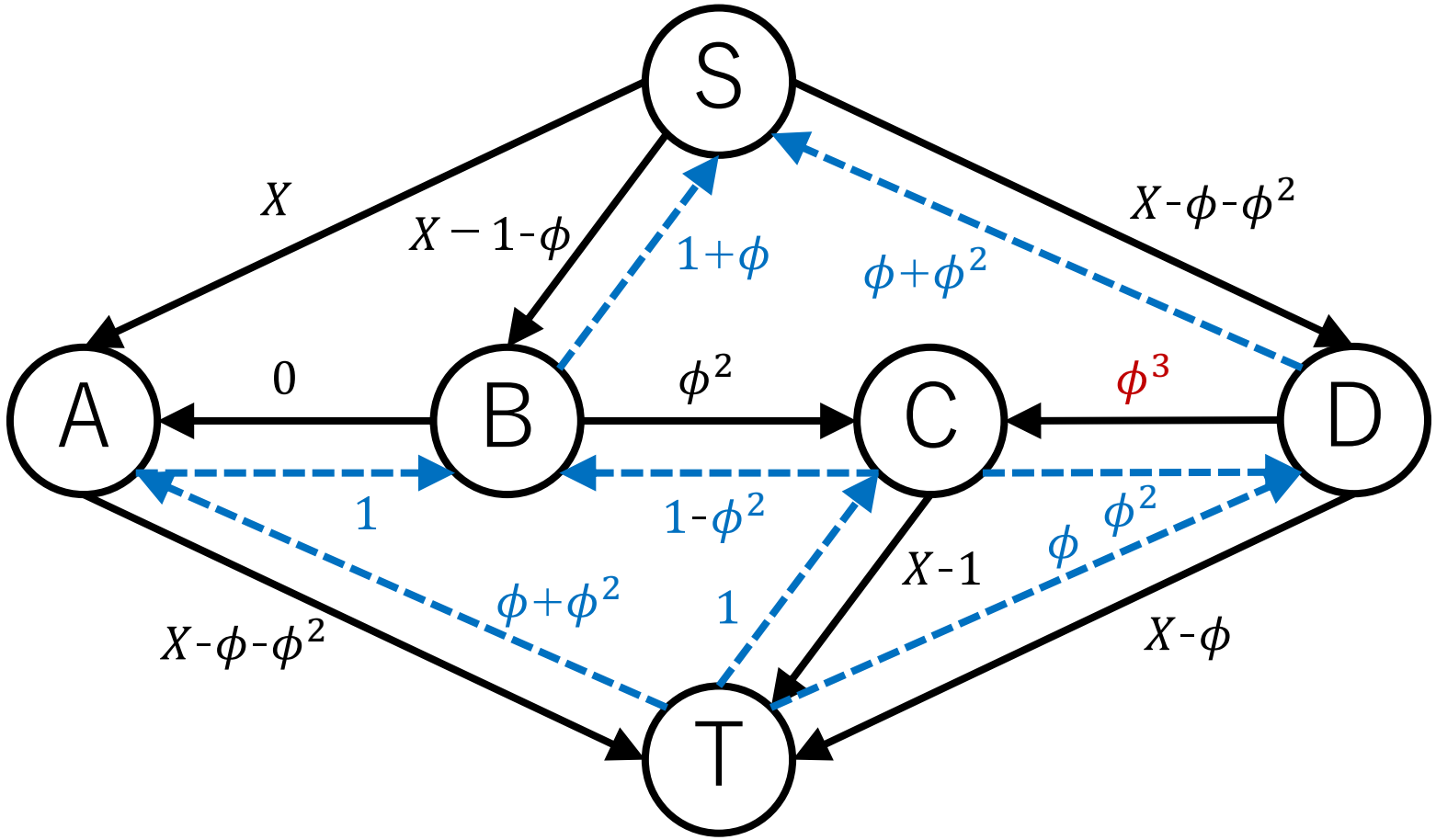
無理数の容量がある場合

$\phi^3 = \phi(1-\phi)$であることを利用する.



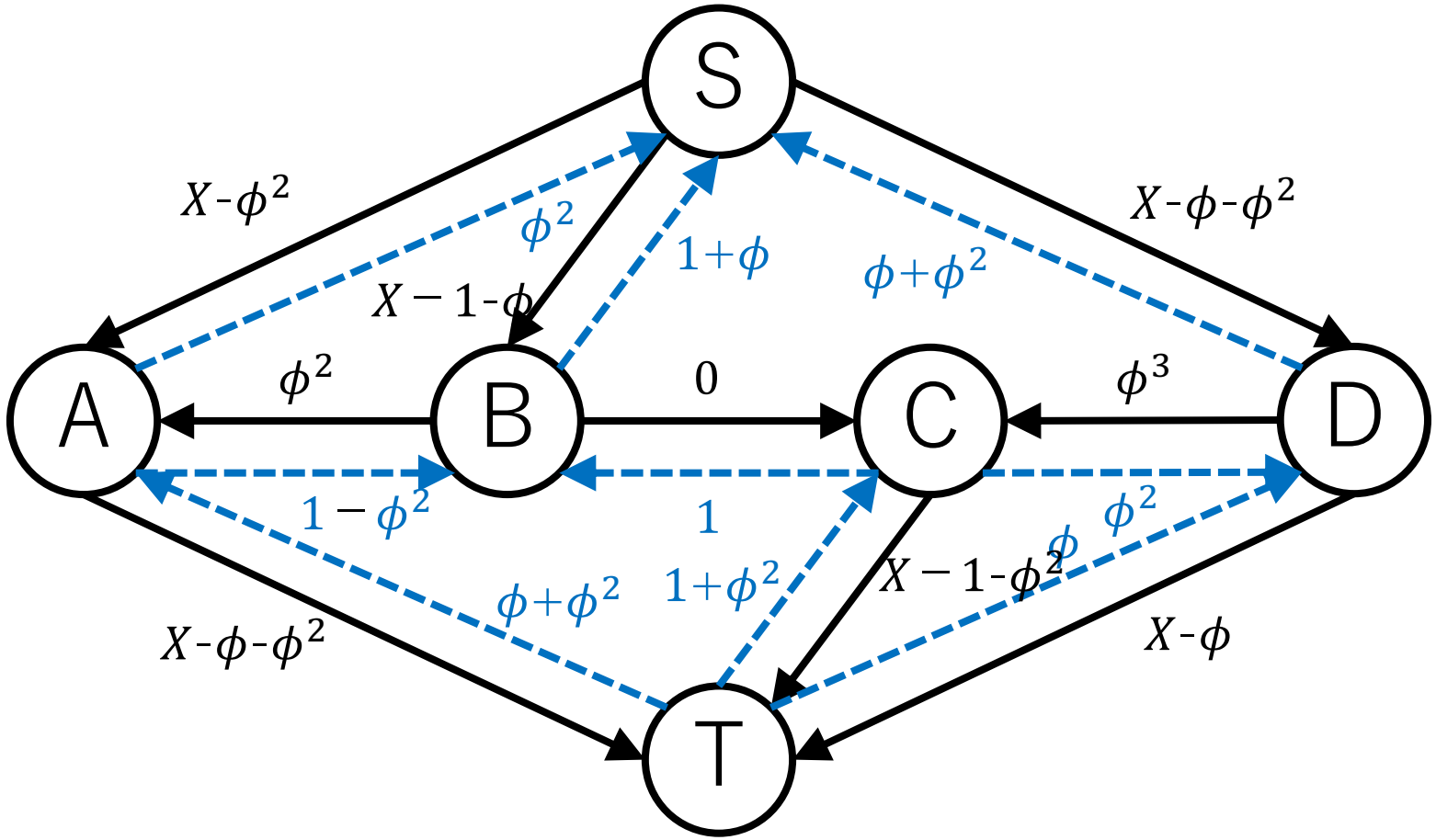
無理数の容量がある場合

(4) S->A->B->C->Tにϕ^2流す.



無理数の容量がある場合

S->A->B->C->Tにϕ^2流したあとの状態.



無理数の容量がある場合

以降，(1)～(4)を繰り返すと，k回目には$2(\phi^{2k-1}+\phi^{2k})$分だけフローを増やすことができ，これを無限回続けることができる．

このときの総フローは，

$$1+2\sum_{i=1}^{\infty}\phi^{i}=1+\frac{2}{1-\phi}=4+\sqrt{5}$$

となり，最適解から大きくずれることになる．

無理数を考えるのは無理矢理？

コンピュータにおいては無理数を完全に表現できないので，気にしなくて良い？

先程の例は，浮動小数点の精度が高くなればなるほど，最悪の場合に処理時間が非常に長くなることを示唆している．

そのような限界があることを知っておくことは重要．🙂

https://courses.engr.illinois.edu/cs473/fa2012/notes/22-maxflowalgs.pdf

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.

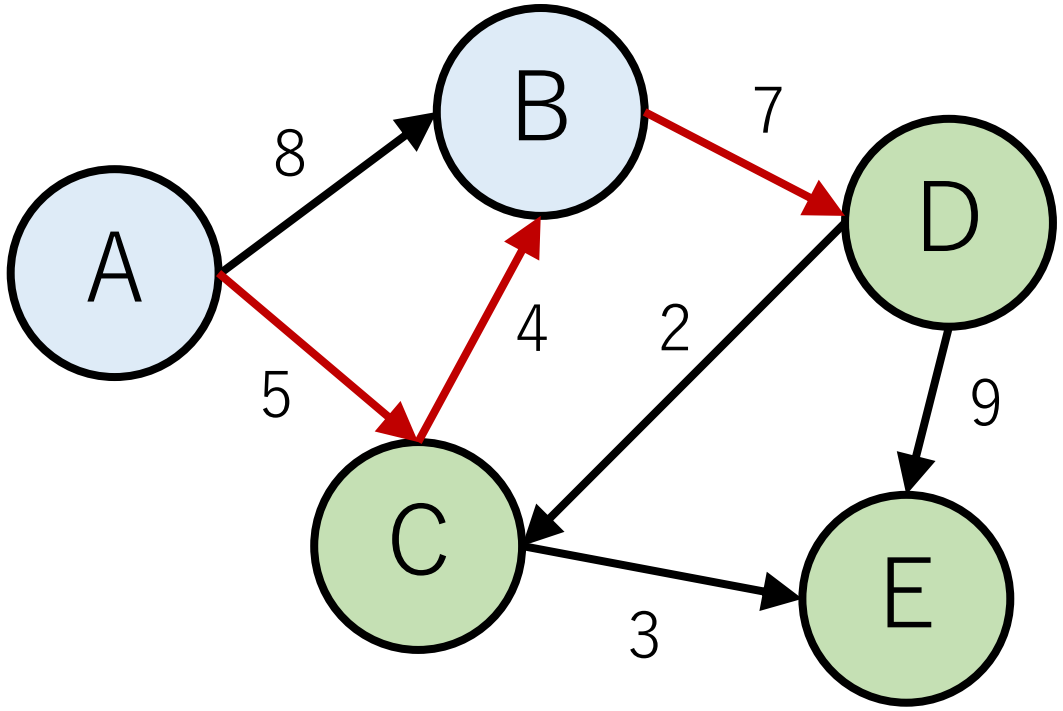
$(S, V \setminus S)$のように表す.

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全部.

グラフのカット

例えば，$S = \{\mathrm{A}, \mathrm{B}\}$ と$V \setminus S = \{\mathrm{C}, \mathrm{D}, \mathrm{E}\}$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．

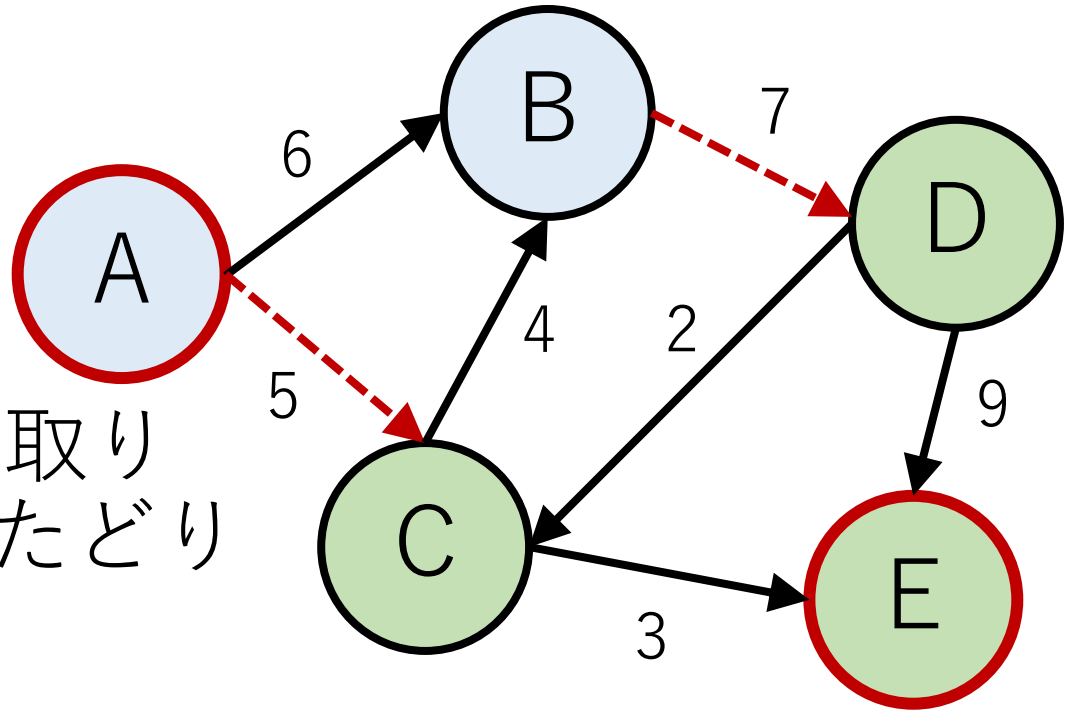


s-tカット

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．



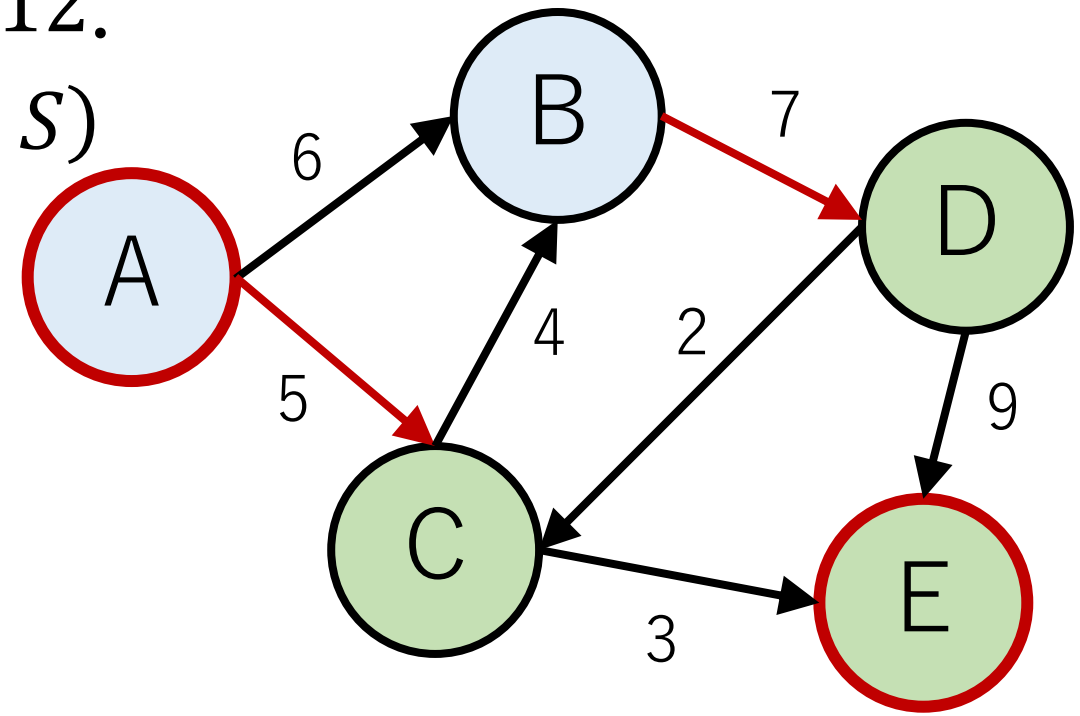
s-tカットのカットエッジを全部取り除くと，開始から終点ノードへたどり着けなくなる．

s-t力ッ ト

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

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

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



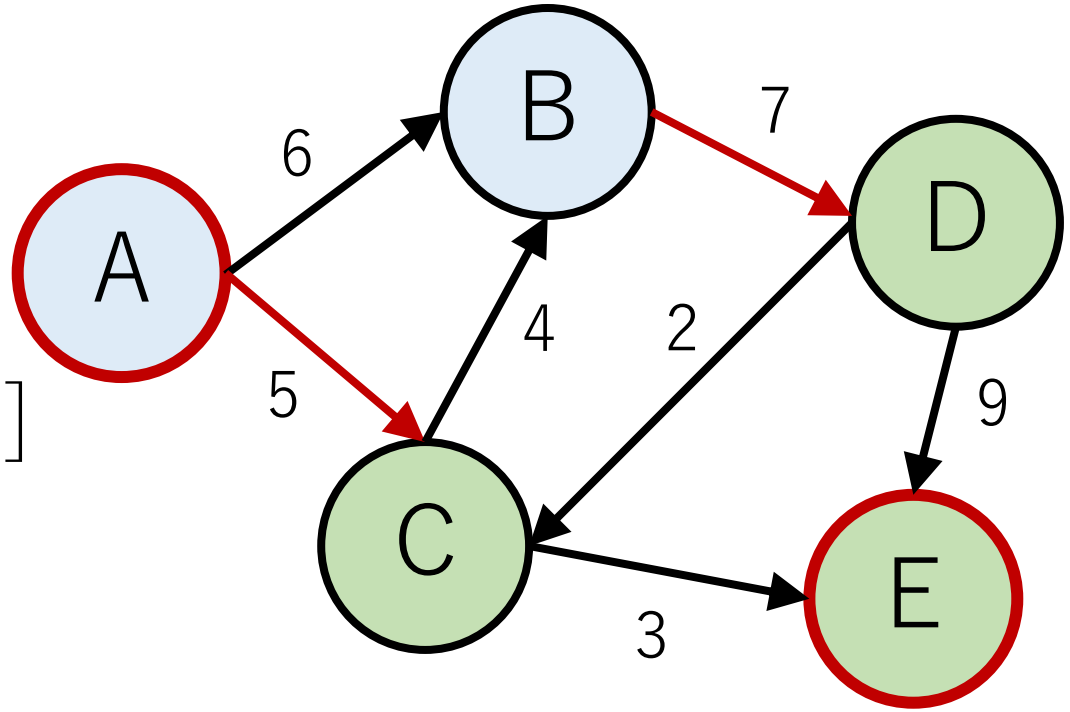
s-tカットの容量とフローの関係

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，

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

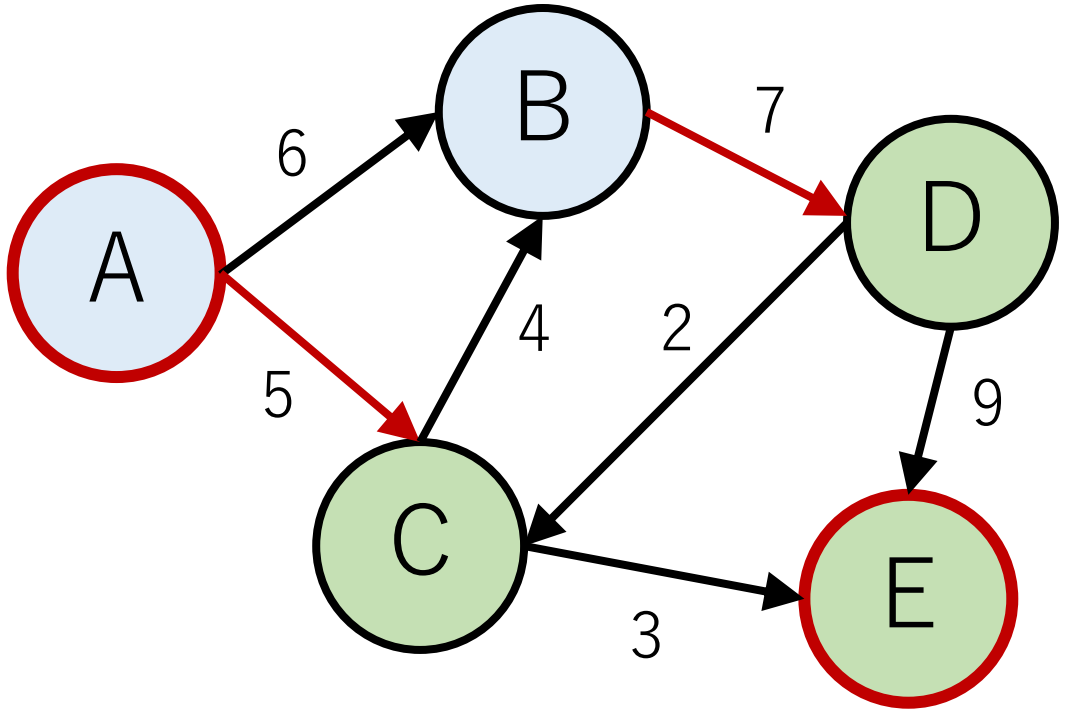


s-tカットの容量とフローの関係

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，

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

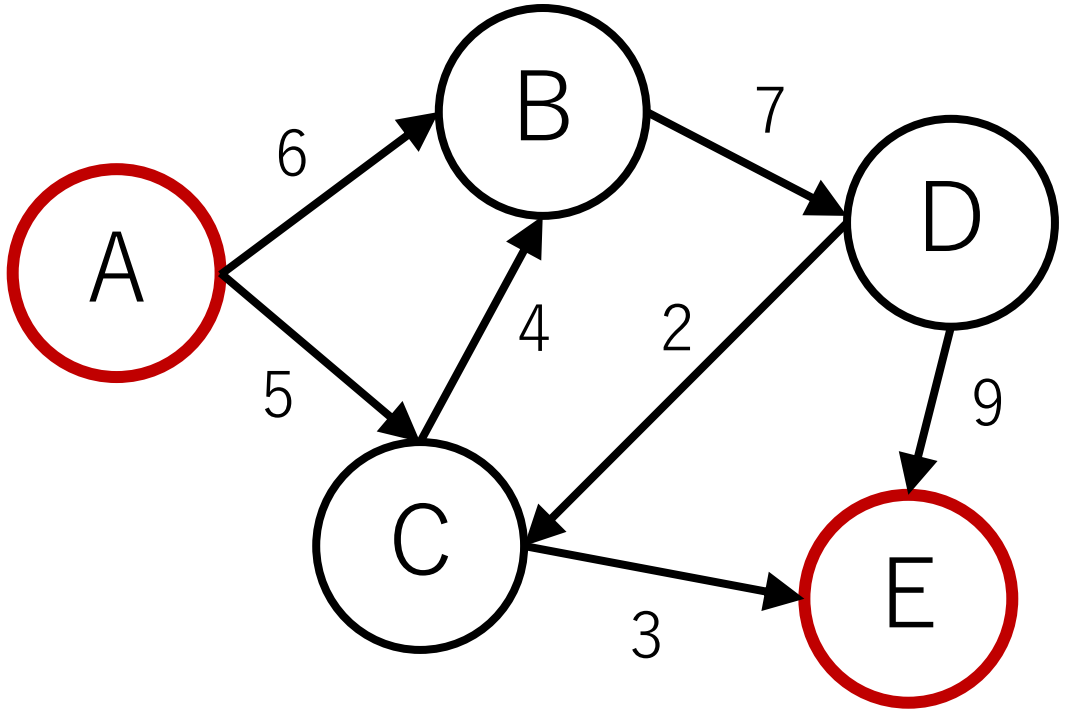
これは任意のs-tカットにおいて成立する不等式となる．



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$U(S \to V \setminus S)$が最小になるようなカット．

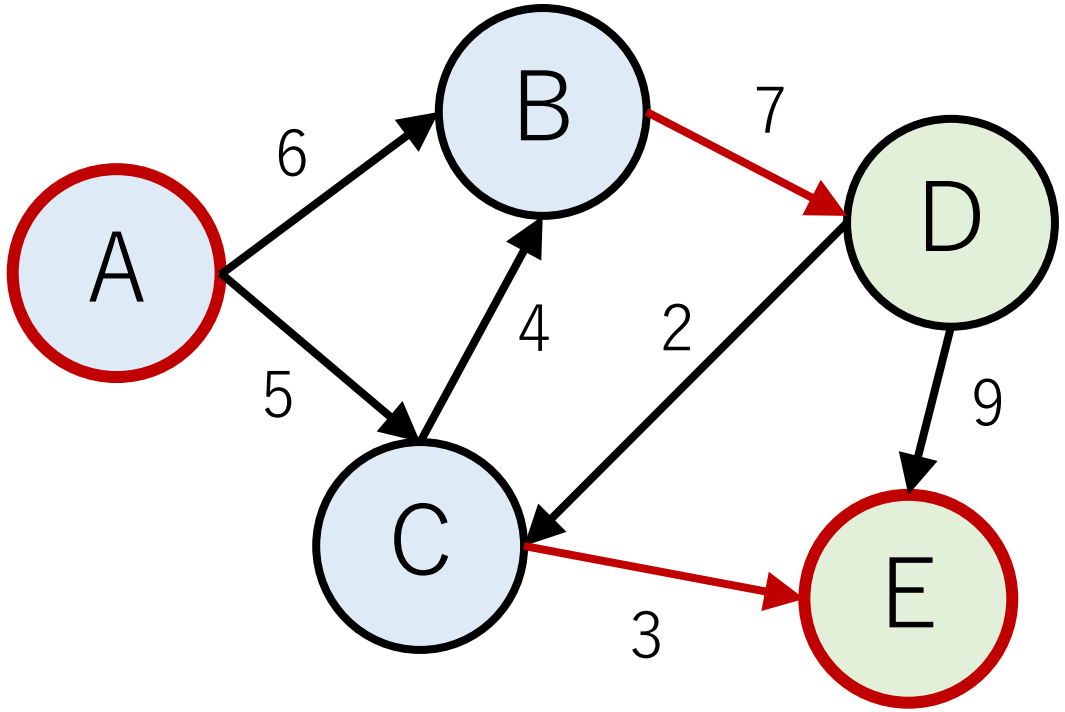
右図の場合はいくら？



最小カット

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

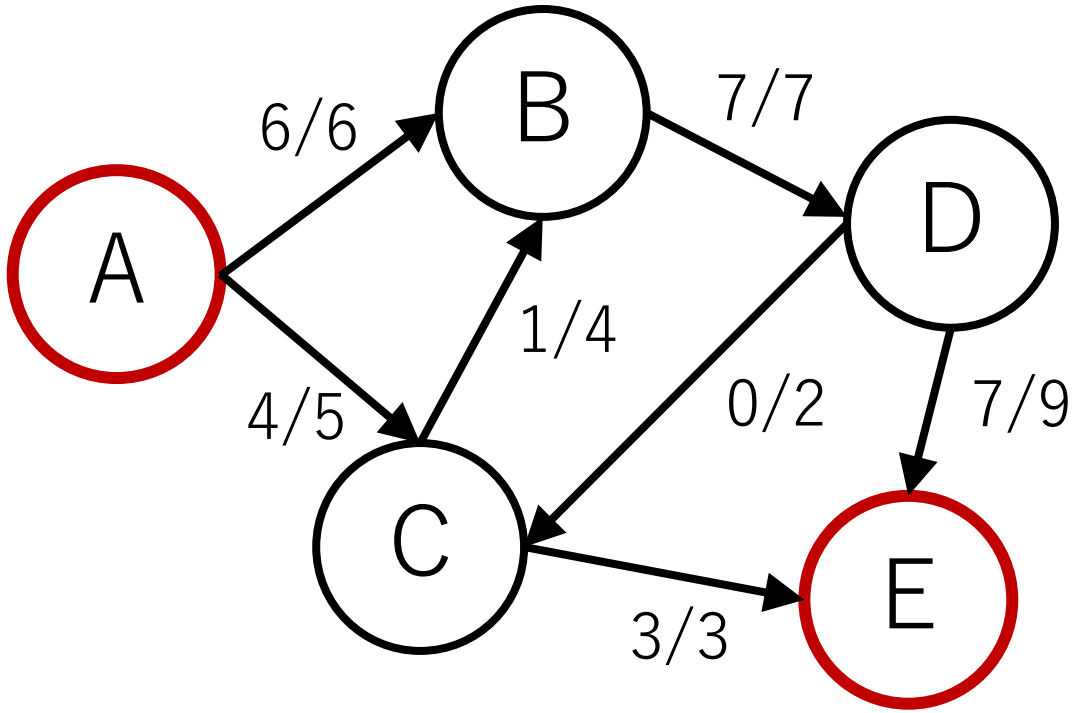
DからCの容量は$U(S \to V \setminus S)$
には入らないので，注意．



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．

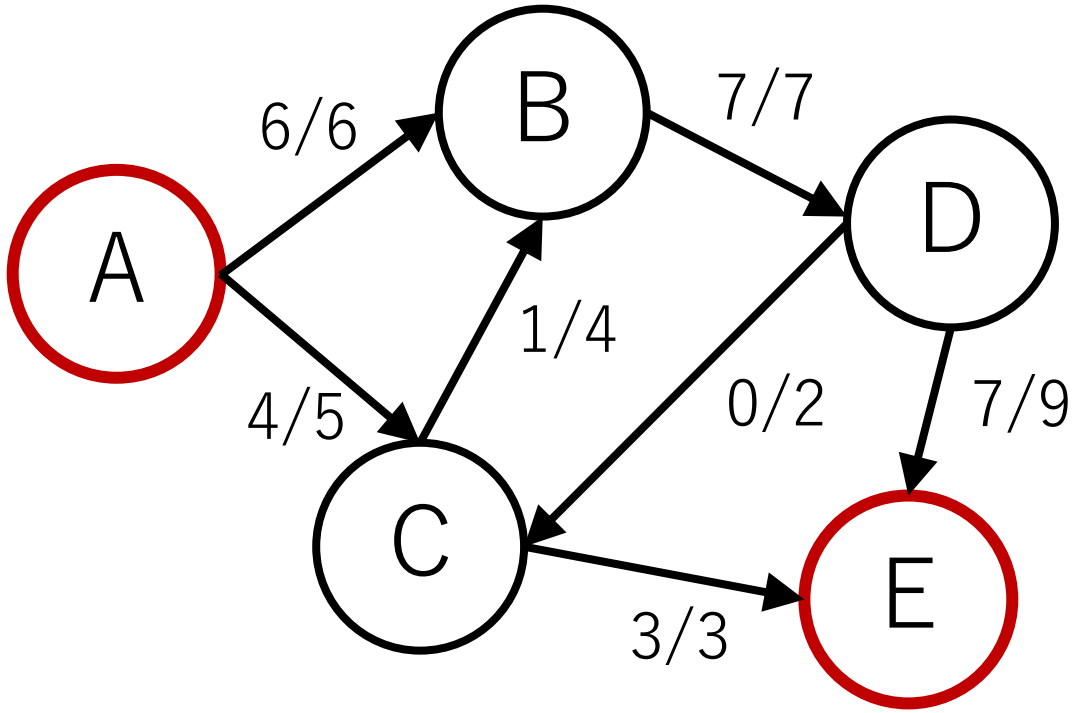
　　残余ネットワークなどとも呼ばれる．

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

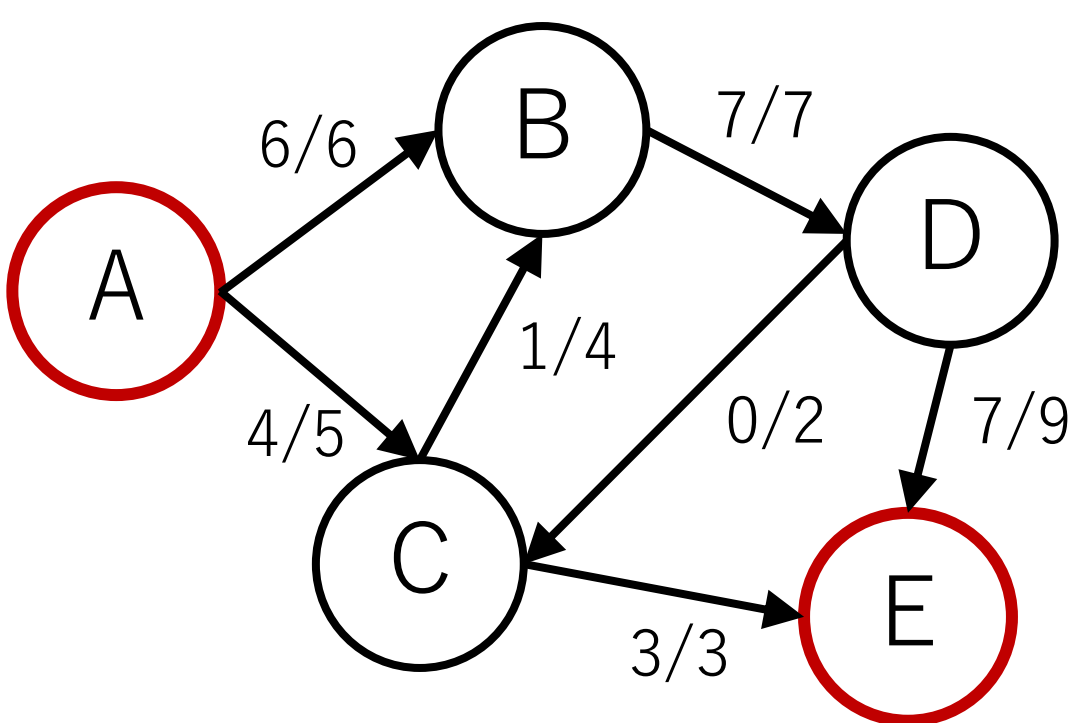
フォード・ファルカーソン法を終了したとき状態は，
以下のようになる（最大流10）．

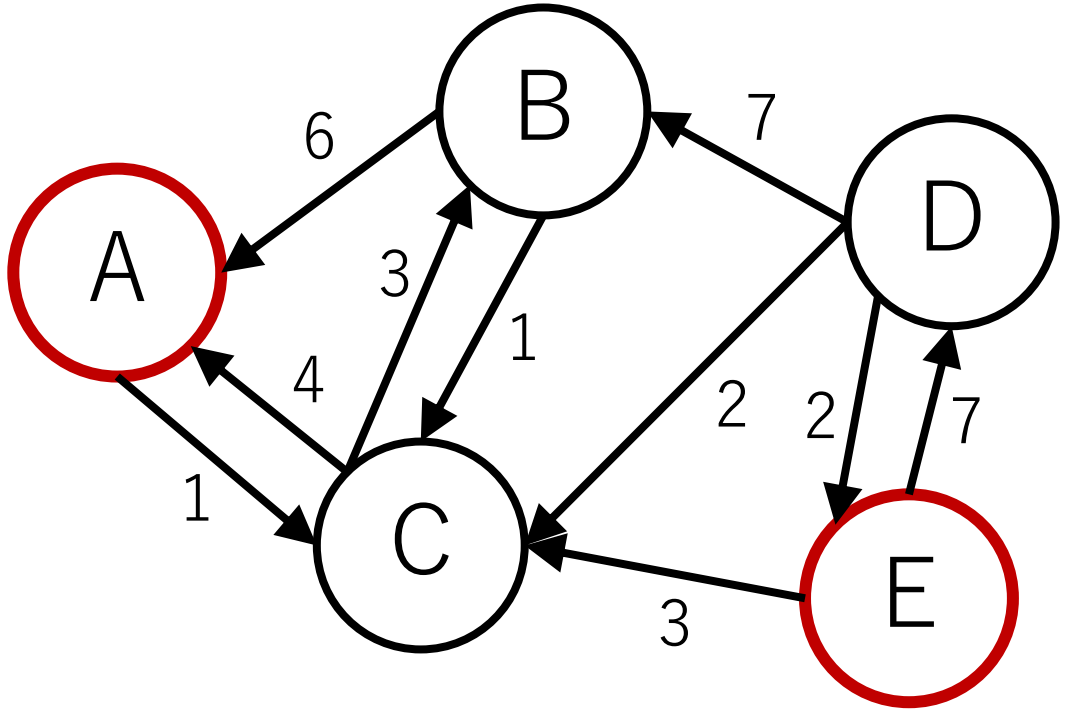


残余ネットワーク

この時の残余ネットワークは以下の右のように表される.

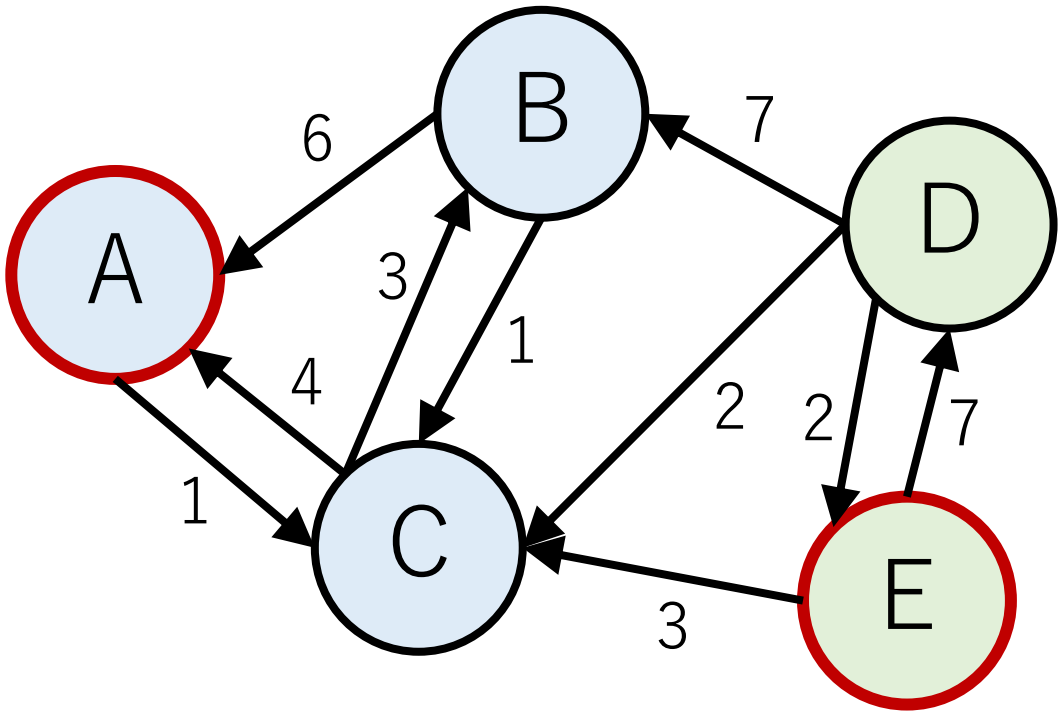
終了状態と残余ネットワークは1対1対応になっている.





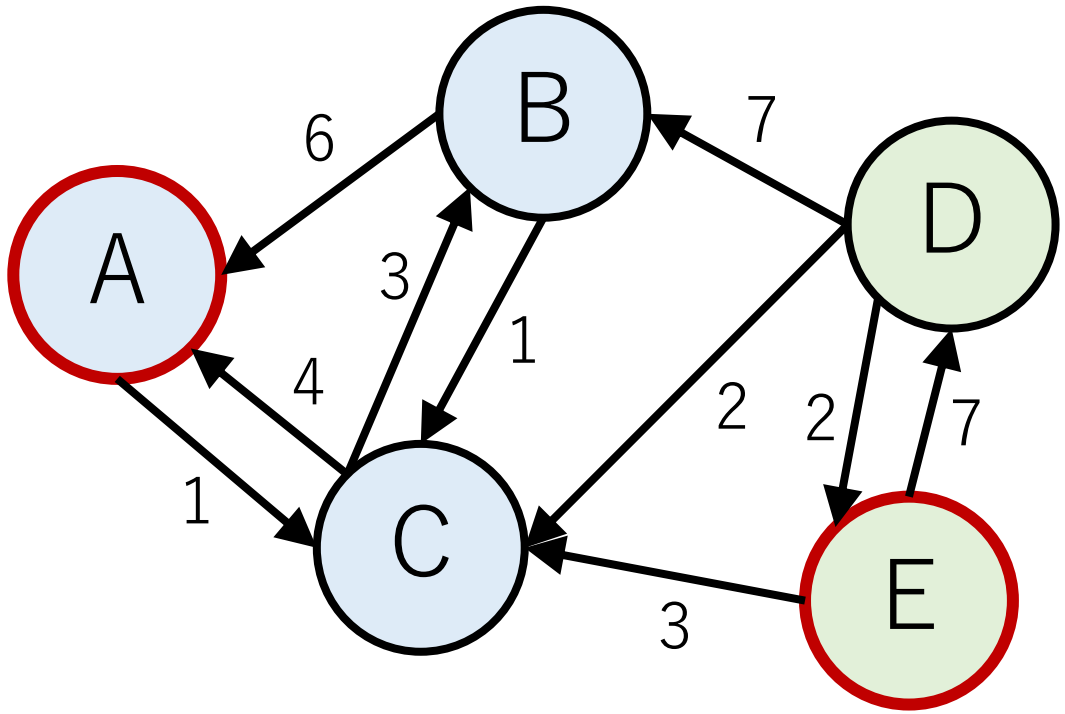
出来上がったs-tカット

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（$U(S \to V \setminus S)$ が最小のもの，最小カット）となっていることを示そう．



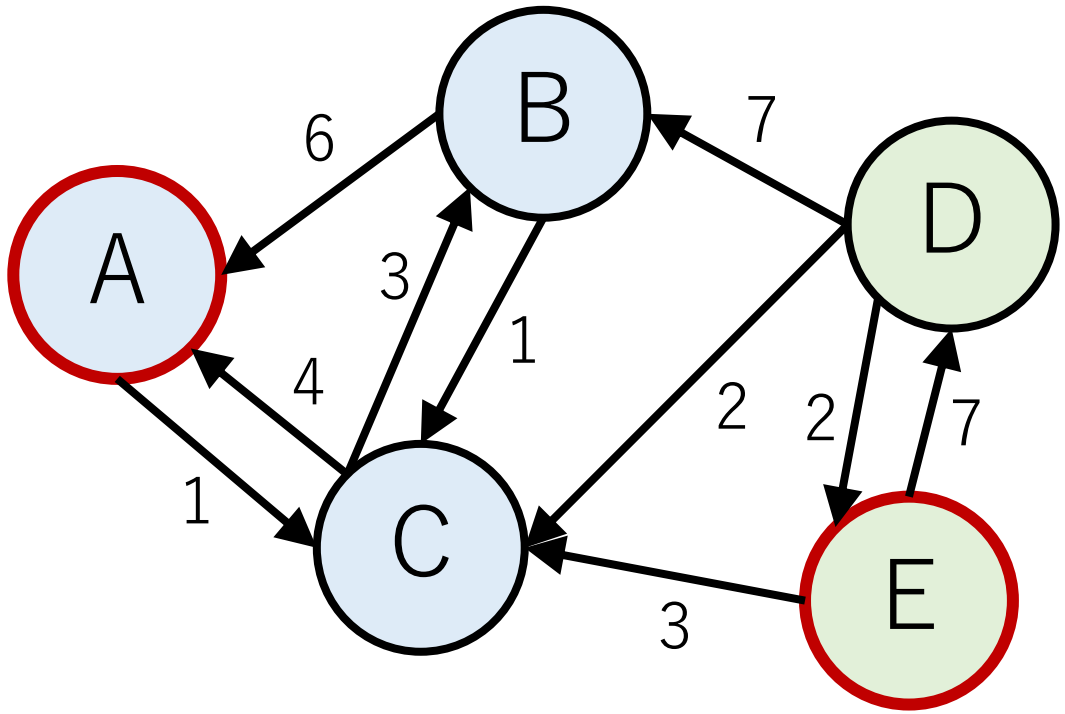
最大流・最小カット定理

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$S \to V \setminus S$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.

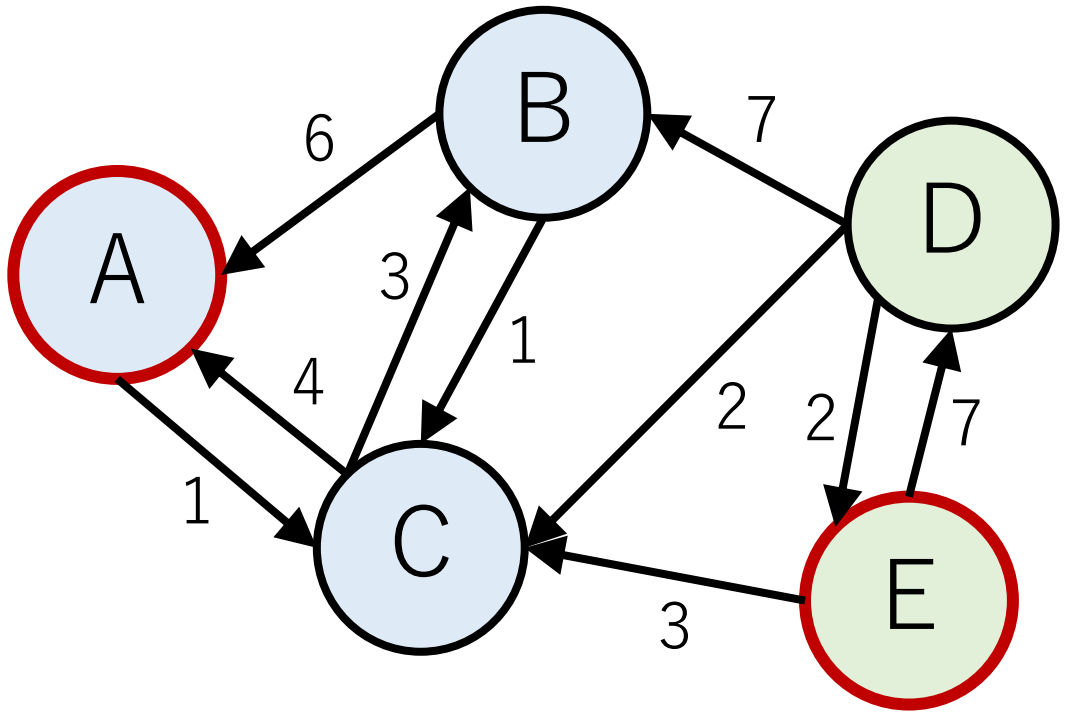


最大流・最小カット定理

この最小カットにおいても，

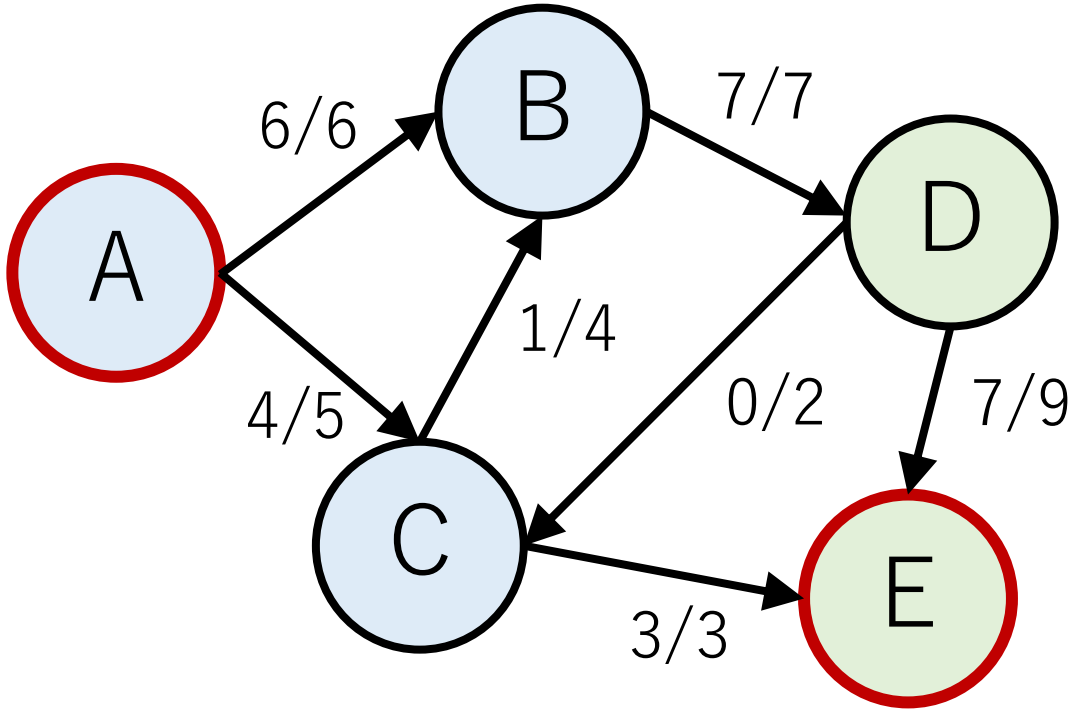
$$f \leq [S\text{から}V \setminus S\text{への流入量}] \leq U_{min}(S \to V \setminus S)$$

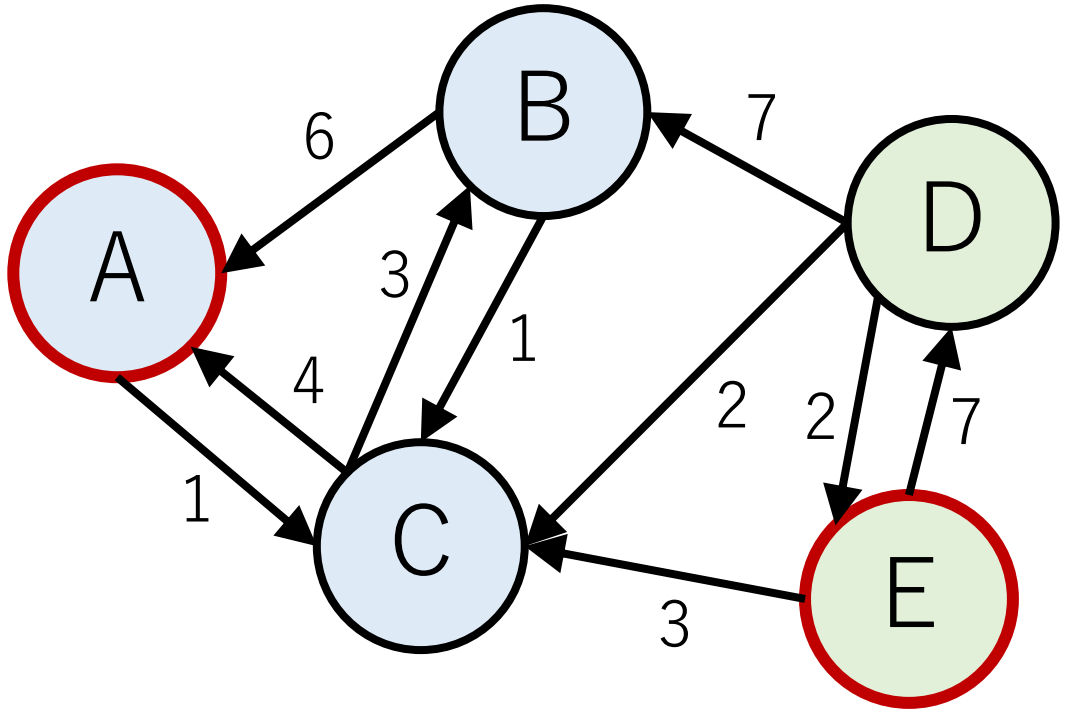
は成立する．



最大流・最小カット定理

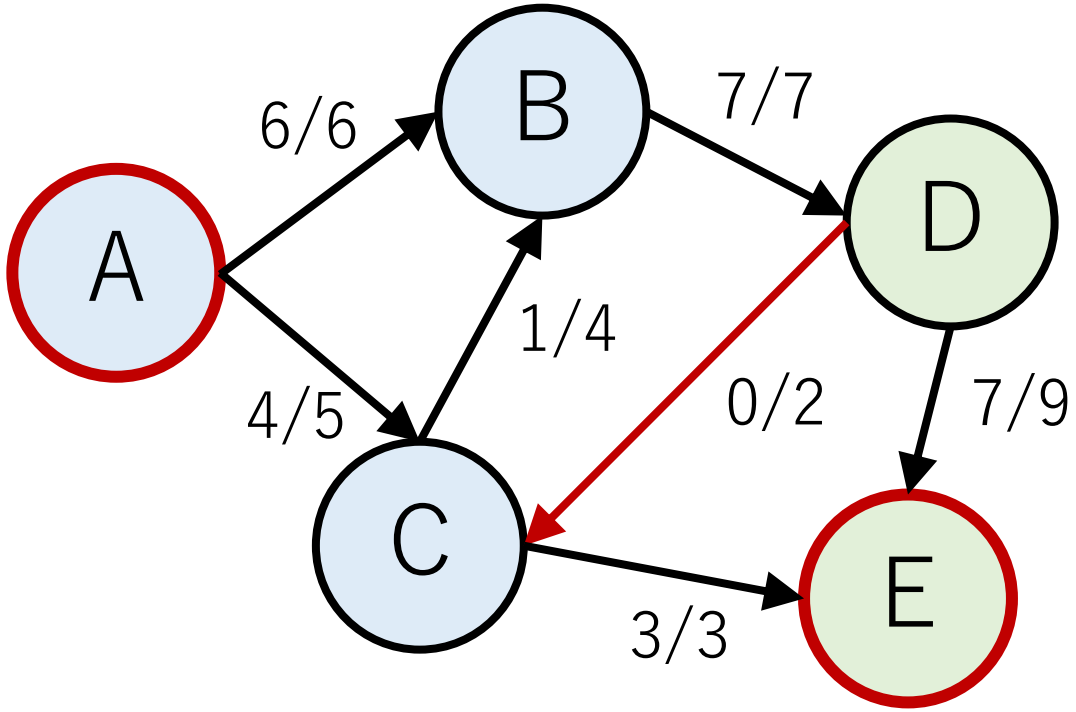
さらに，この不等式を詳しく見ていこう．
$f = [S$から$V \setminus S$への流入量$] - [V \setminus S$からSへの流入量$]$で
あることも思い出しておく．

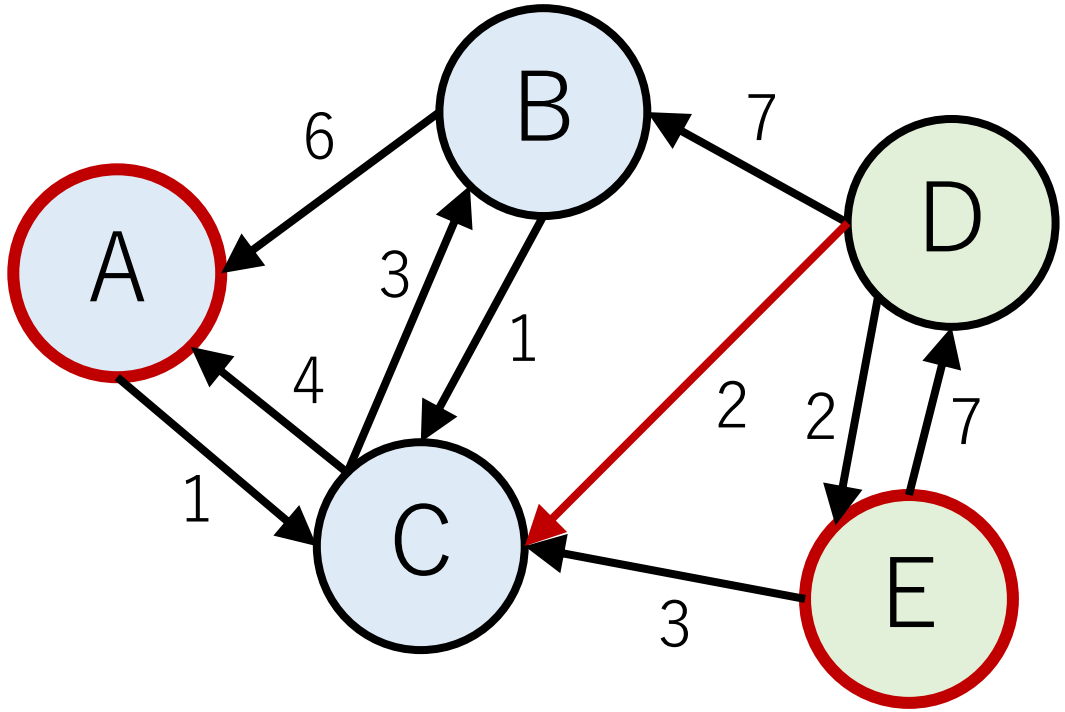




最大流・最小カット定理

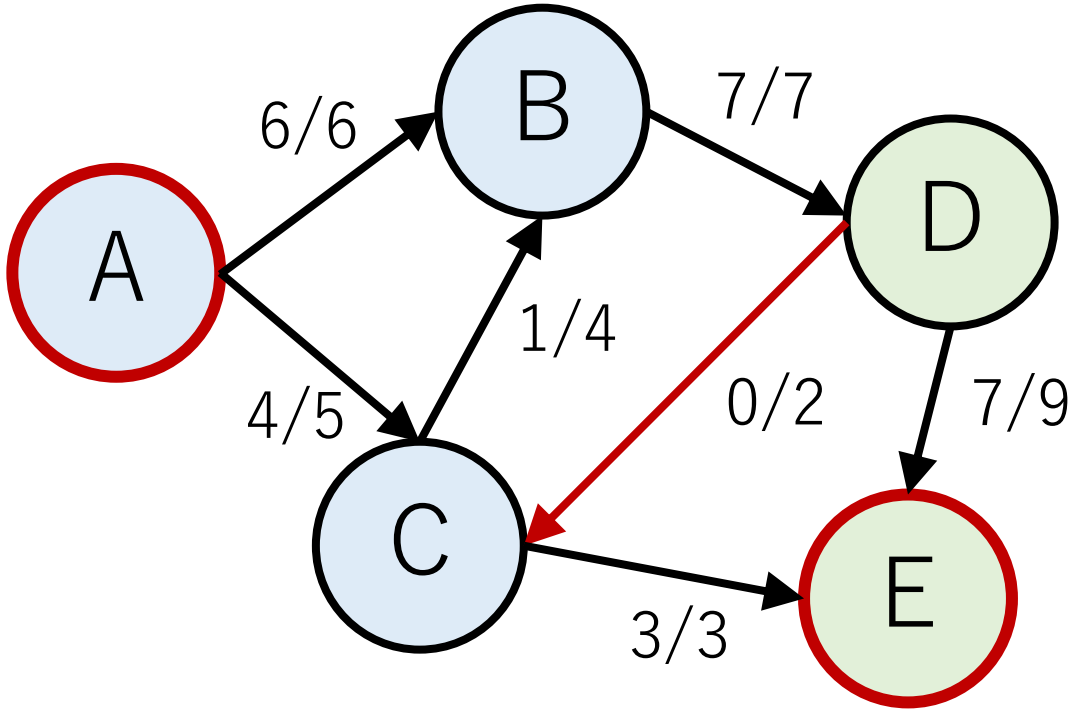
$V \setminus S$ からSに向かう辺のうち，元々与えられたグラフにあったものだけを考える．この場合，DからCの辺のみ．

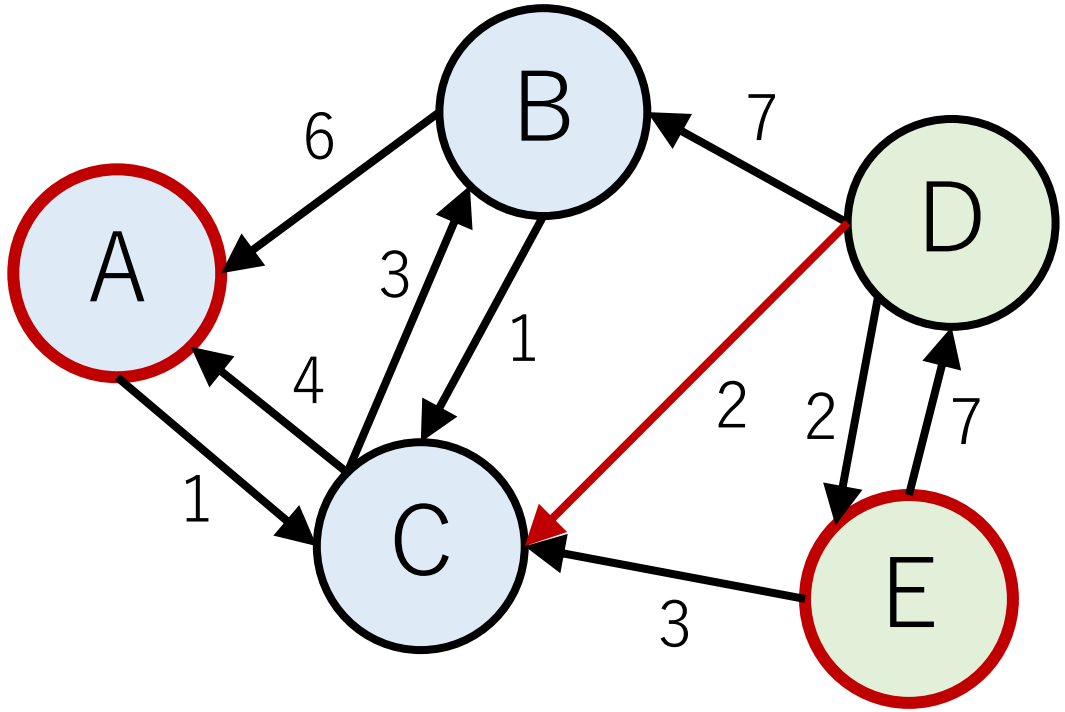




最大流・最小カット定理

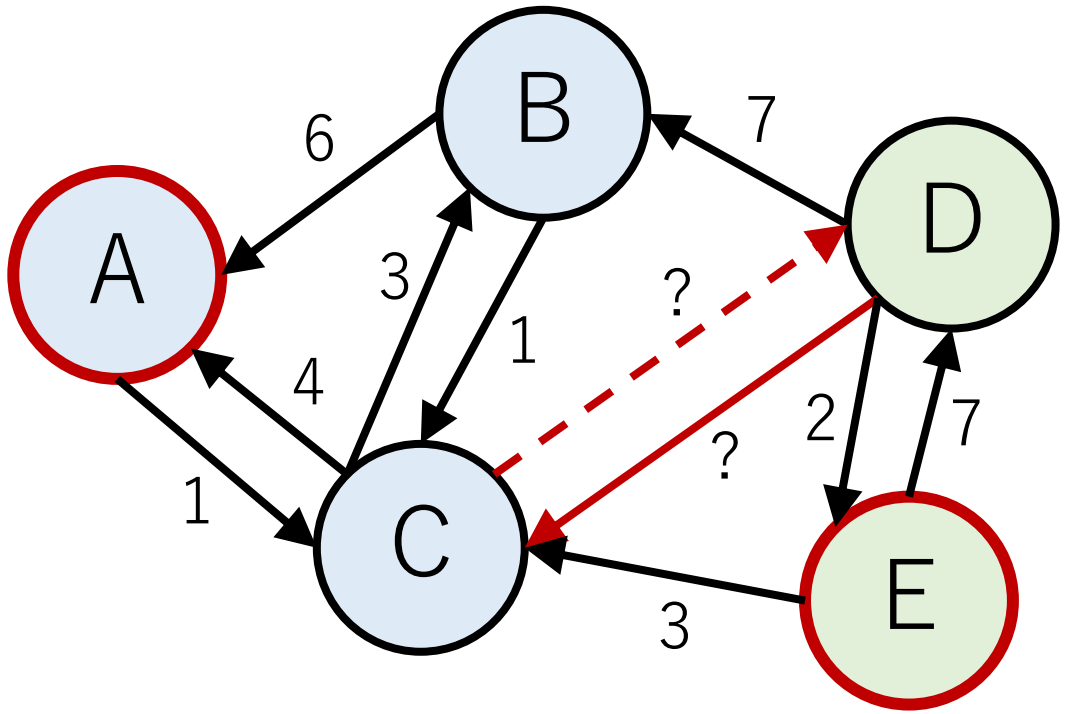
この辺では仮想的な逆辺が出てきていない．つまり
フローを全く流してない，ことになる．





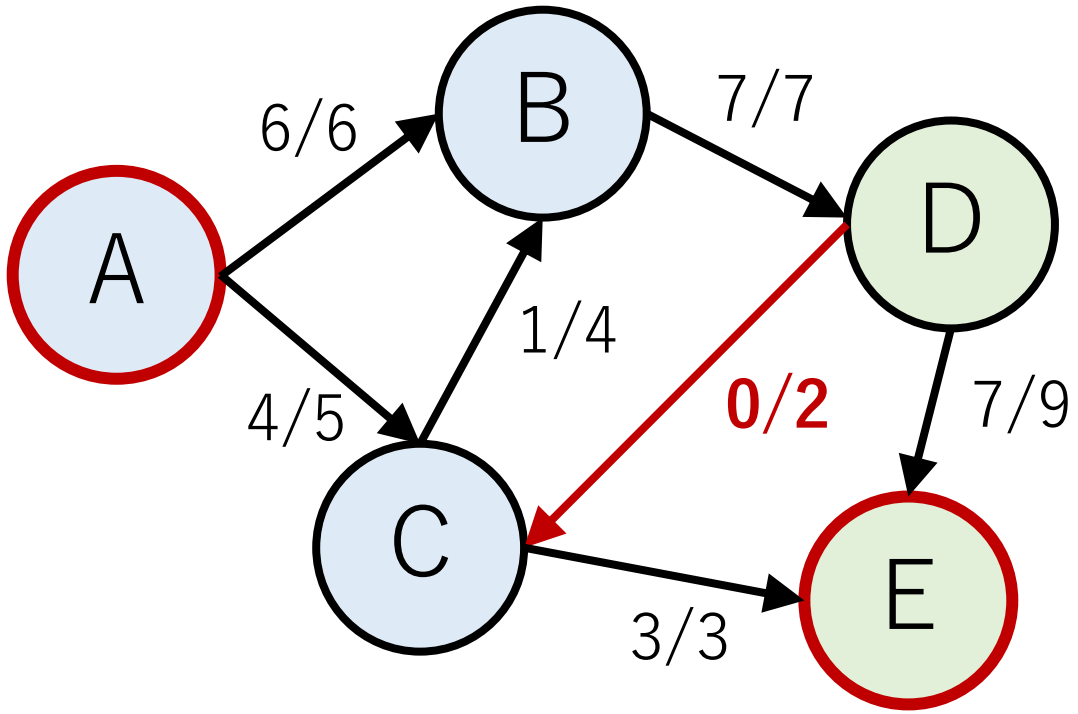
最大流・最小カット定理

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．）



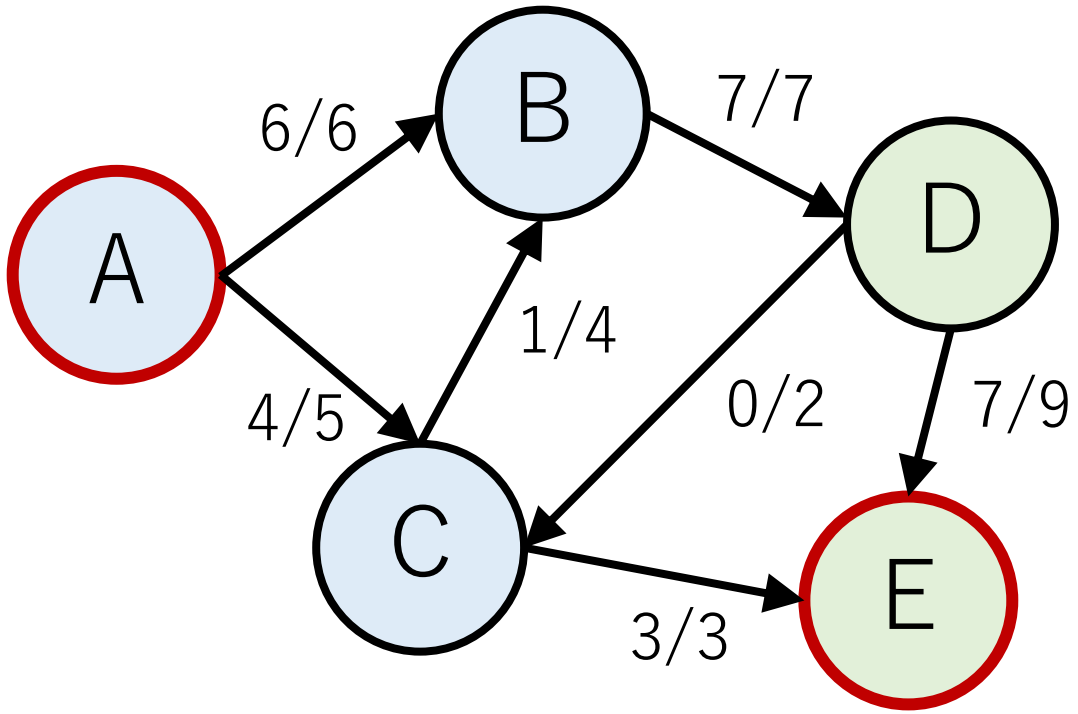
最大流・最小カット定理

つまり，$V \setminus S$からSへの流入量は0となっている．



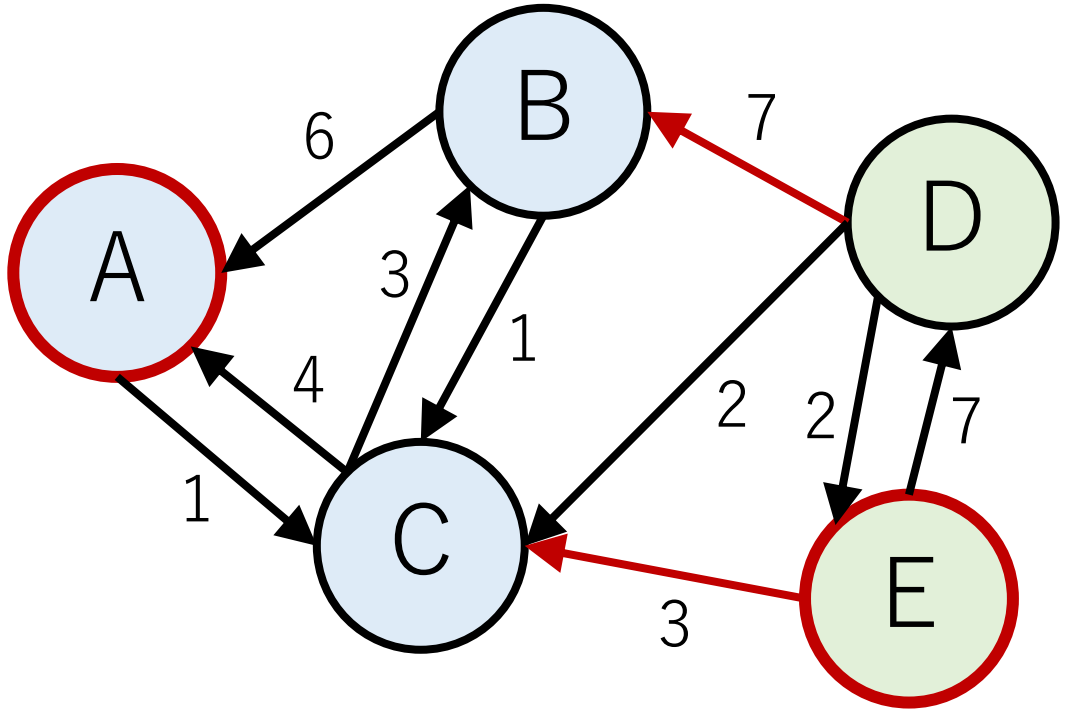
最大流・最小カット定理

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



最大流・最小カット定理

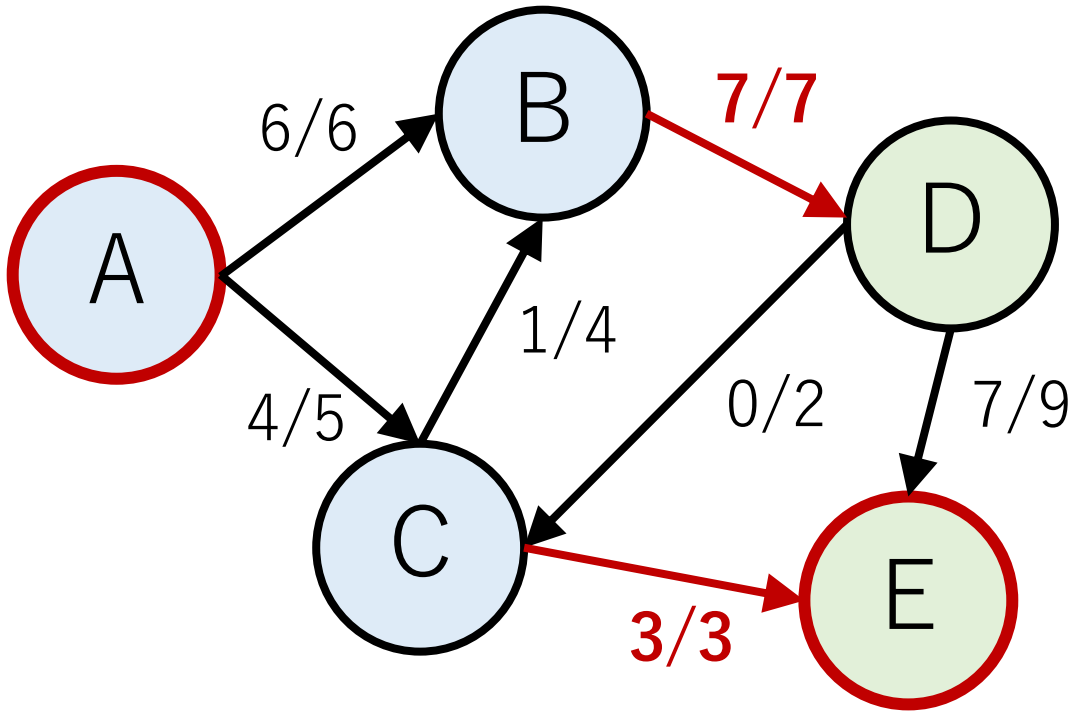
一方，フォード・ファルカーソン法の終了状態ではSから$V \setminus S$に向かう辺は存在していない．（もし存在していれば，さらに流せるはずなので，終了していないはず．）



最大流・最小カット定理

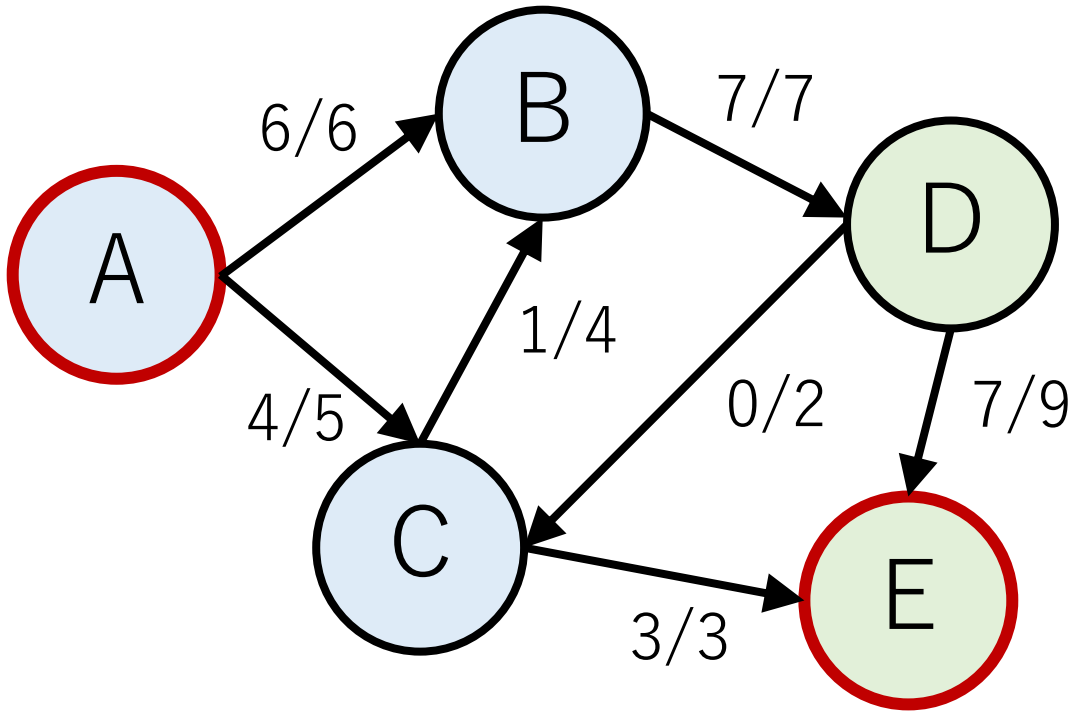
つまり，フローを最大限流したことで，Sから$V \setminus S$にはもう行けない状態となっているからである．すなわち，

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



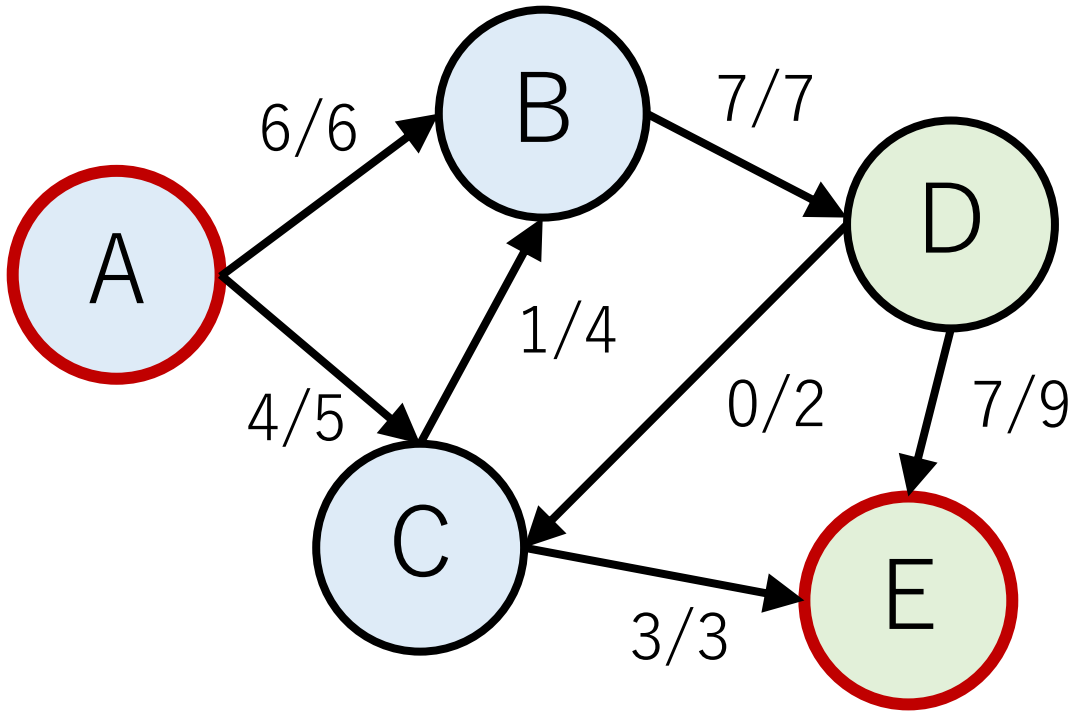
最大流・最小カット定理

すなわち，$f = [S\text{から}V \setminus S\text{への流入量}] = U_{min}(S \to V \setminus S)$
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

$f = U_{min}(S \to V \setminus S)$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．



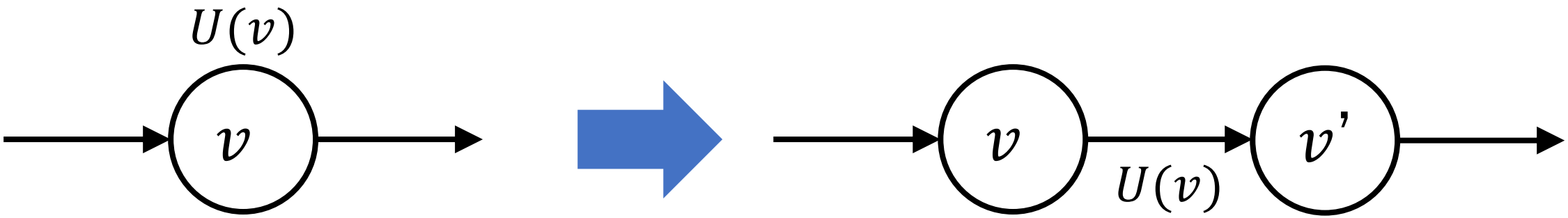
最小カットを愚直に探すと，，，

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

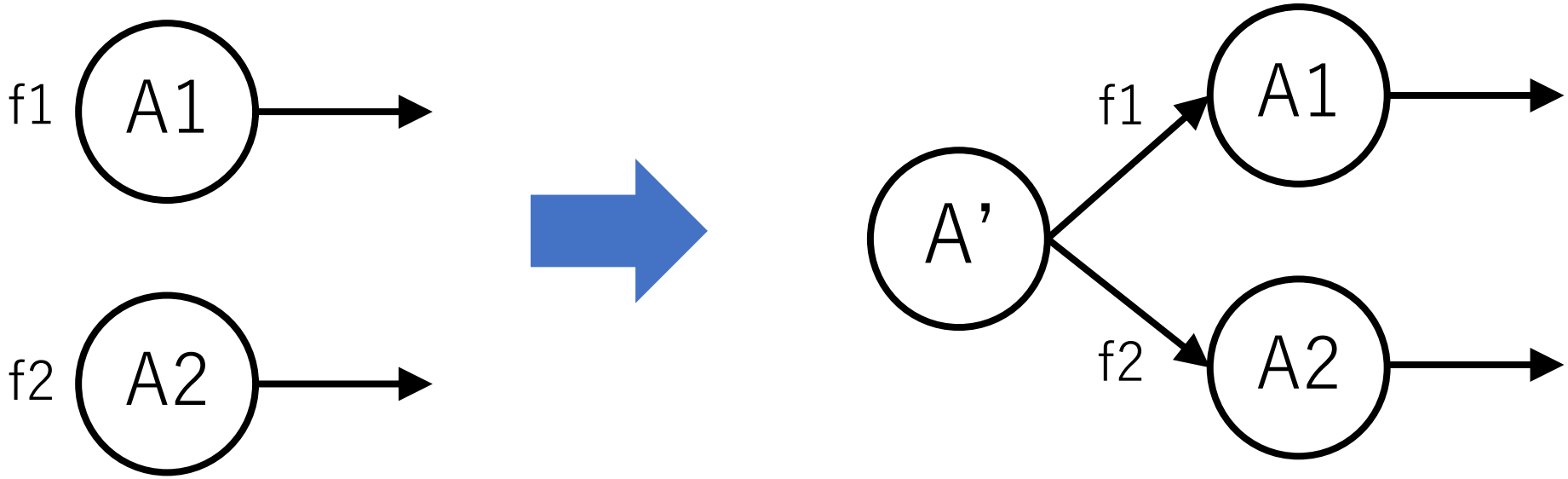
ノードを入ってくる方と出る方の2つに分けて，その間をノードの容量で結ぶ.



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．

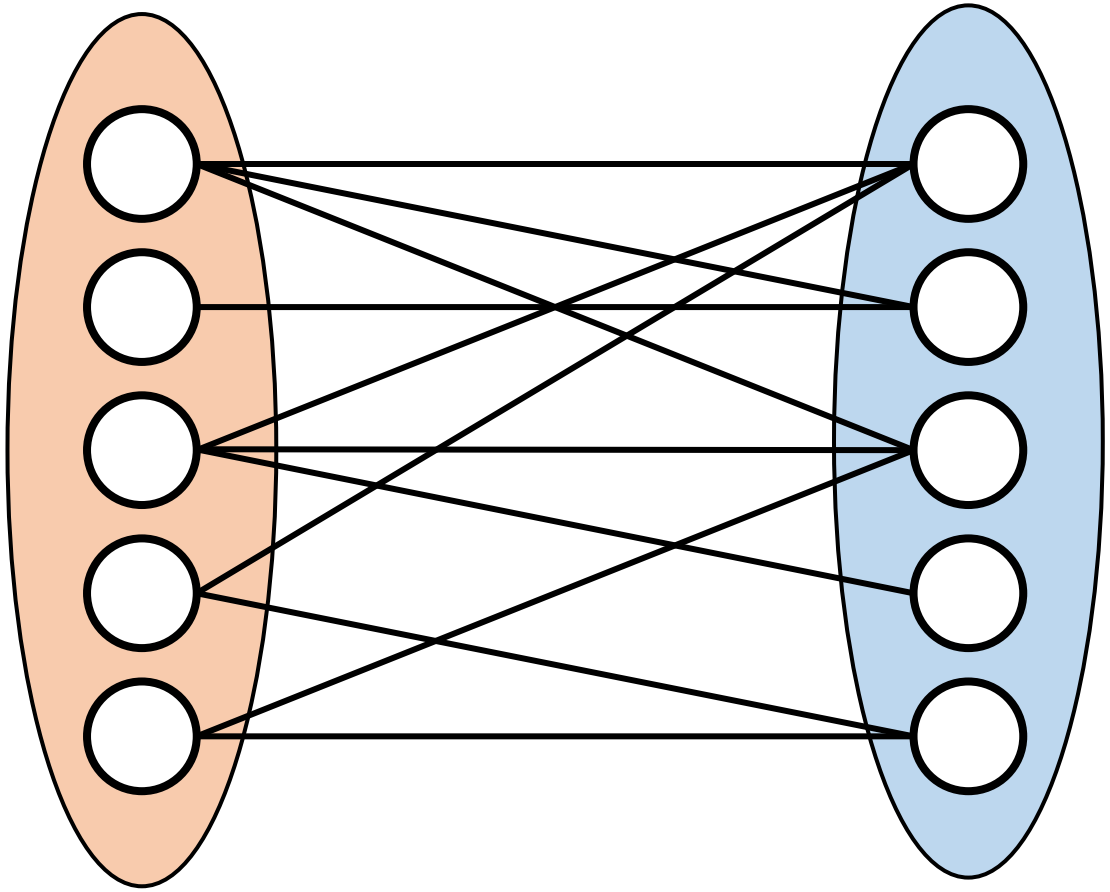


フローアルゴリズムの典型的な応用：

二部グラフのマッチング

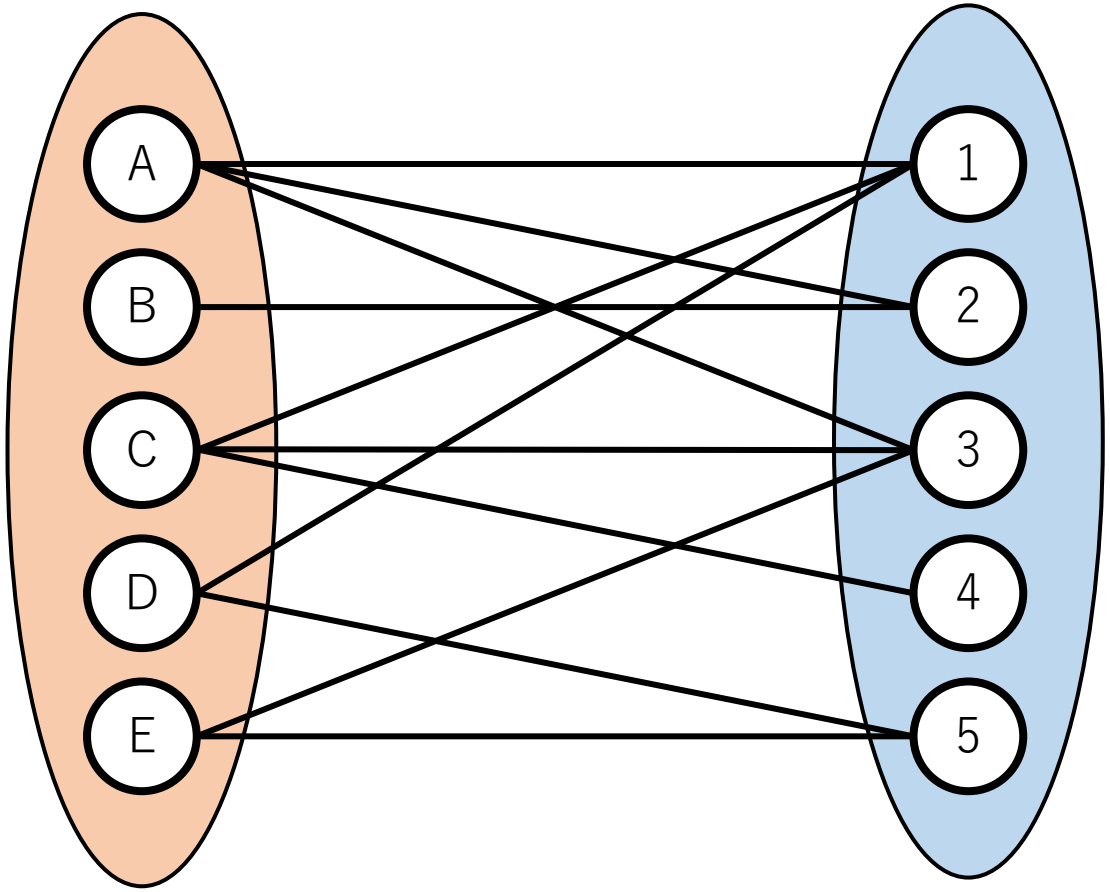
二部グラフ（bipartite graph）

頂点集合を2つの部分集合に分割でき，各集合内の頂点同士の間には辺が無いグラフ.



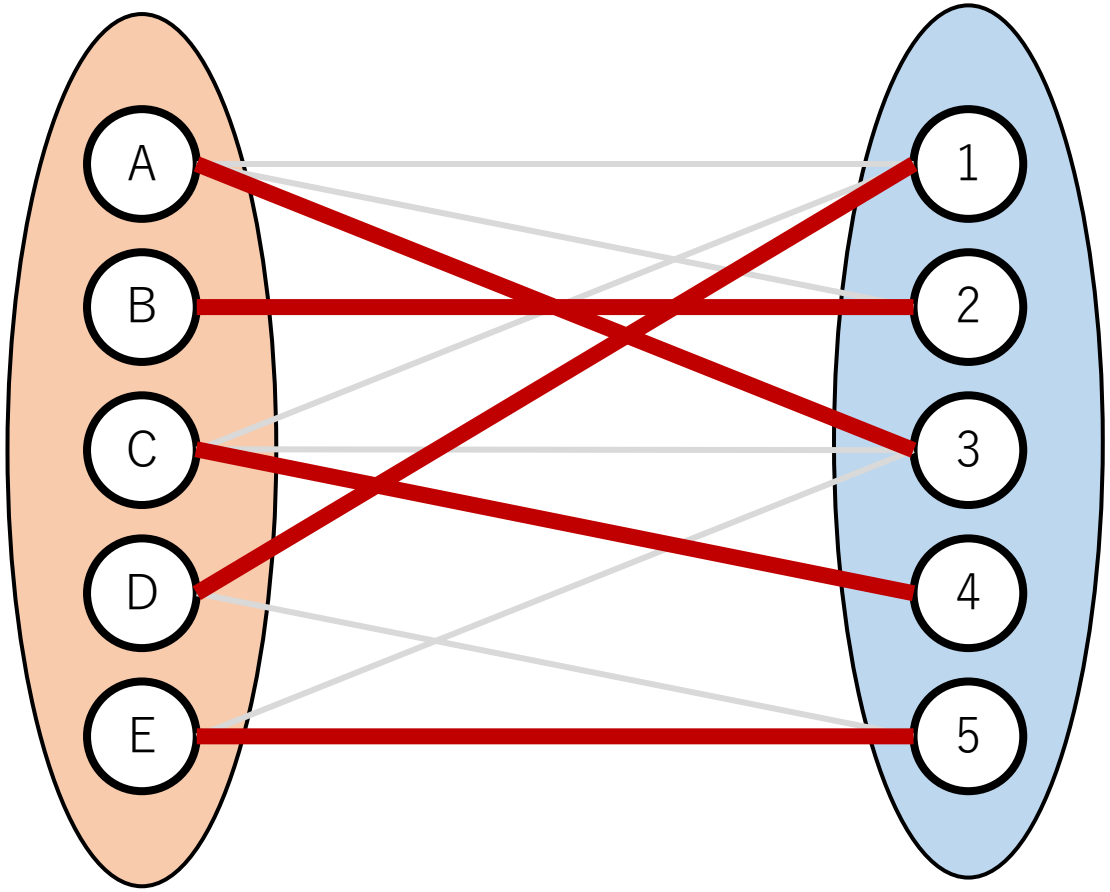
二部グラフの例：ペア組み

左のグループの人と右のグループの人をペアにする.
辺があるのはペアになっても良いことを示す.



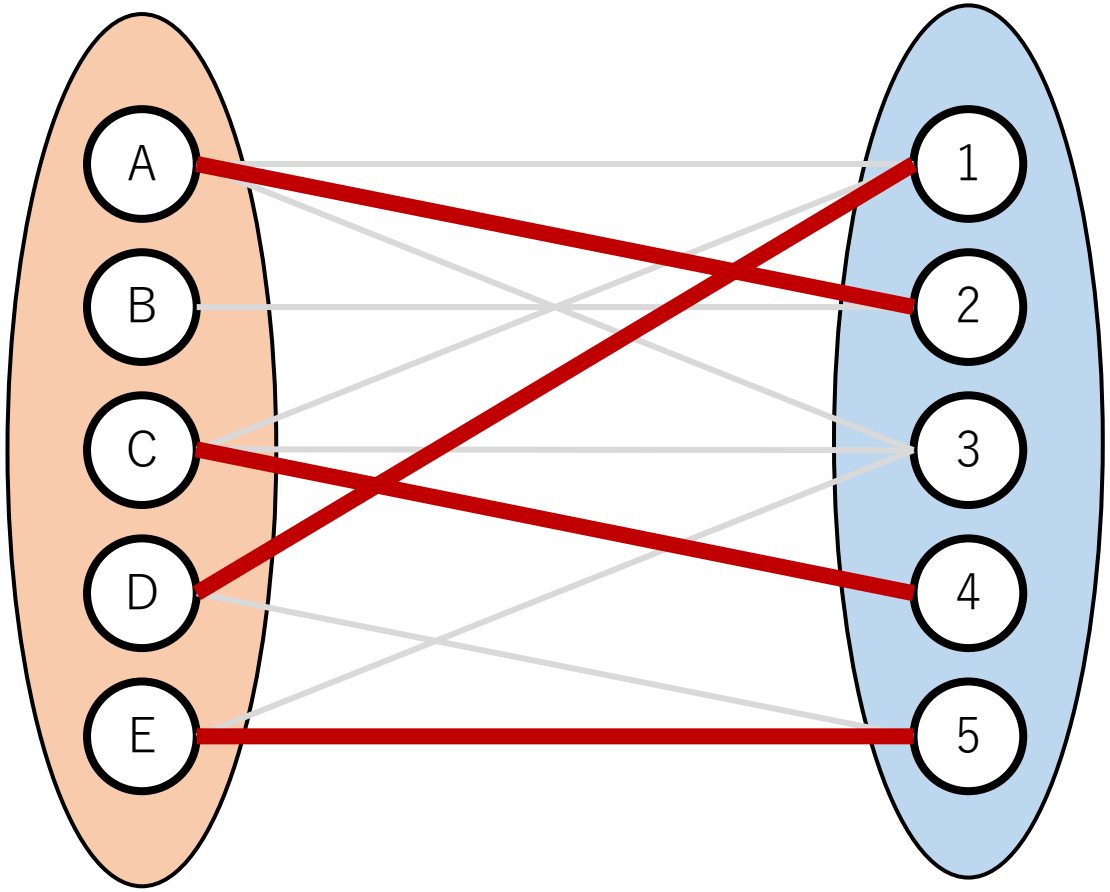
ペア組みがうまくいく例

全員ペアになるケース.



ペア組みが最大でない例

Bと3がペアになれない.



二部グラフのマッチング問題

二部グラフにおいてノード間をつなぐ組み合わせを解く.

答えは必ずしも1つではない（最大マッチングの時など）.

いろんなケースがありうる.

- 複数のノードにつながることを許す（多対多）.
- ある決められたペアの数を達成する.
- などなど. . .

今日のところは

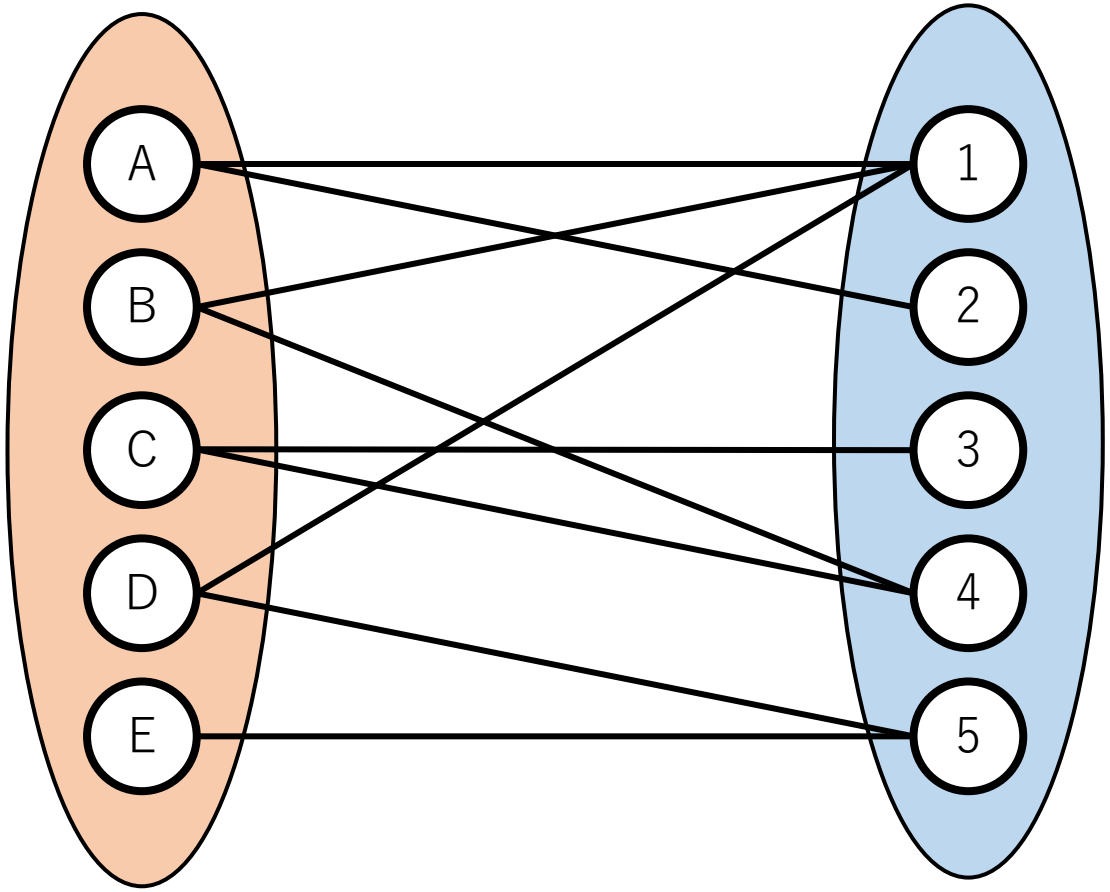
「二部グラフの最大マッチング問題」を紹介.

最大マッチング

ペアの数を最大にする辺の組み合わせ（の1つ）.

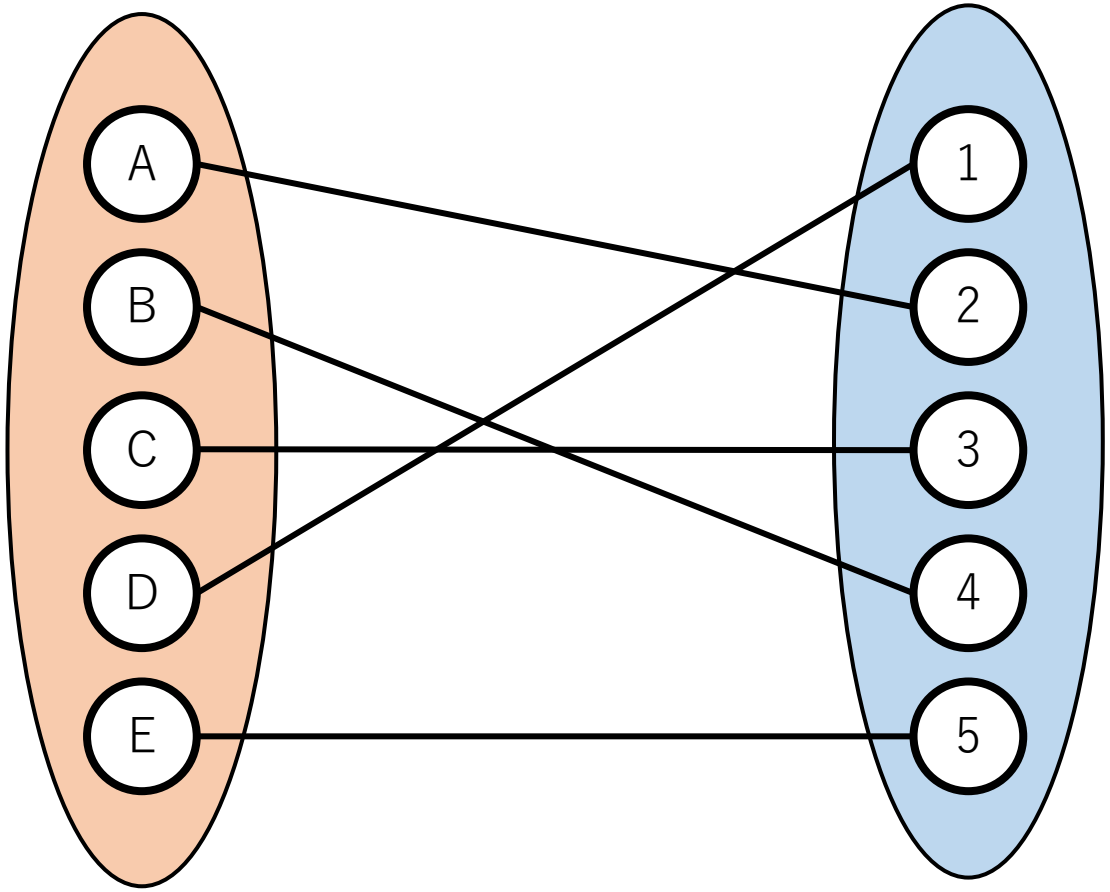
二部グラフの最大マッチング問題

左右のグループをつなぐ辺の数を最大にする．ただし，
どの2辺も共通のノードを持たない．



二部グラフの最大マッチング問題

「どの2辺も共通のノードを持たない」
＝「n股はナシ（n>1）」



二部グラフの最大マッチング問題

最大マッチングになる辺の組み合わせが見つかった時,
残る辺の数は最大.

二部グラフの最大マッチング問題

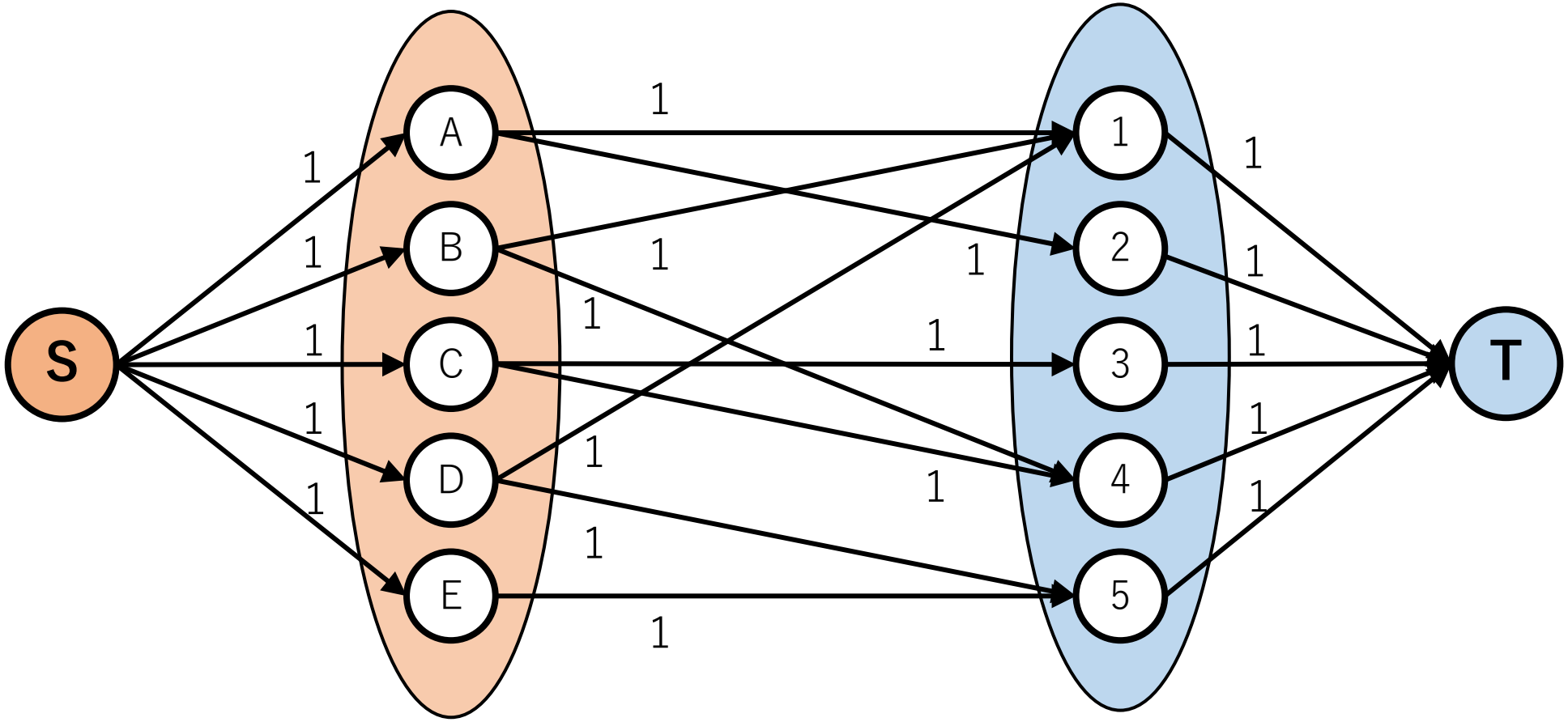
これを最大流問題に置き換えて考える.

辺の容量を1として，左のグループから右のグループ
に流れる流量が最大になる辺の組み合わせを考える.

全ての辺の容量は1なので，流れる流量が最大になれば,
左のグループから右のグループへと流すために使った
辺の数も最大，つまり左右の組み合わせ数も最大と
なっている状態を作り出せる.

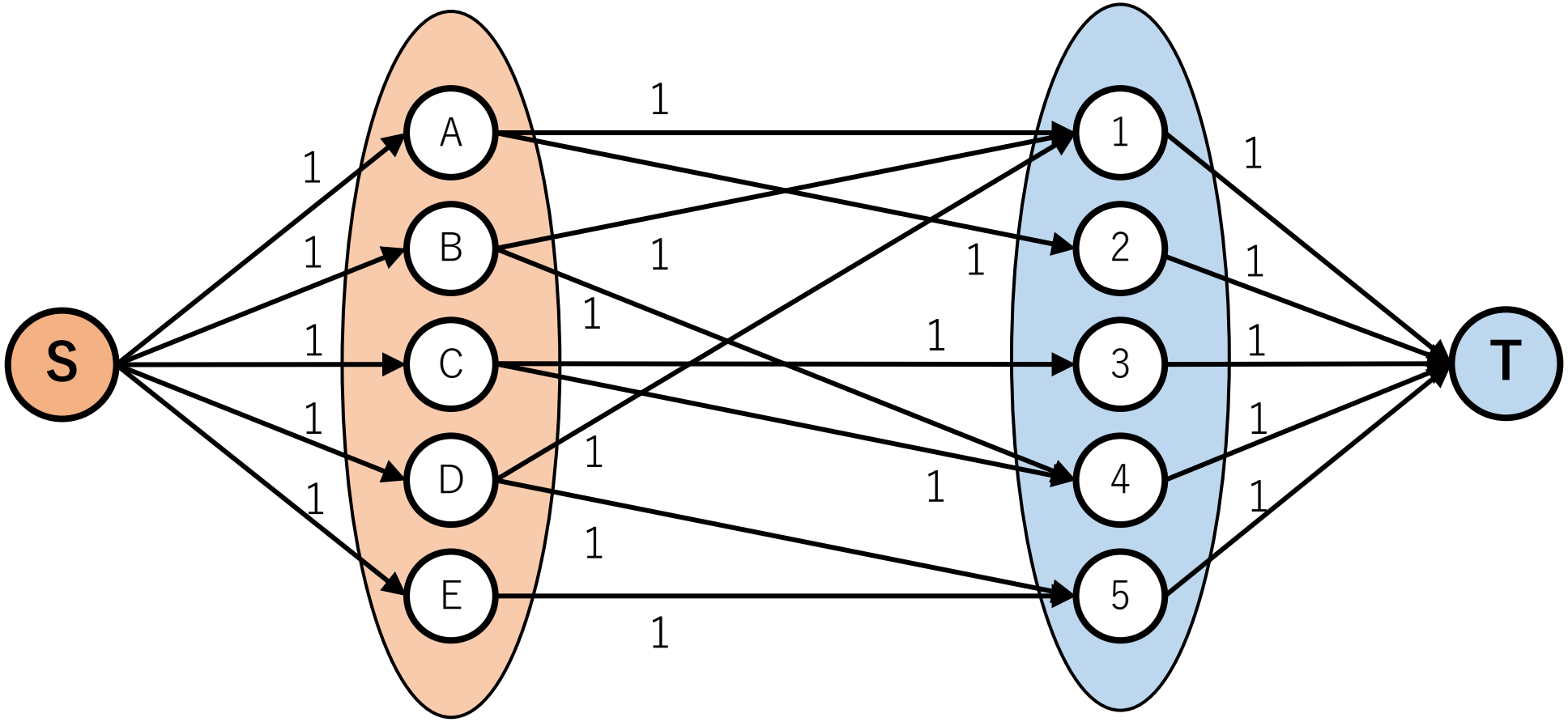
二部グラフの最大マッチング問題

仮想的な開始ノードS，終了ノードTを考えて，SからTへの最大流量を求める．



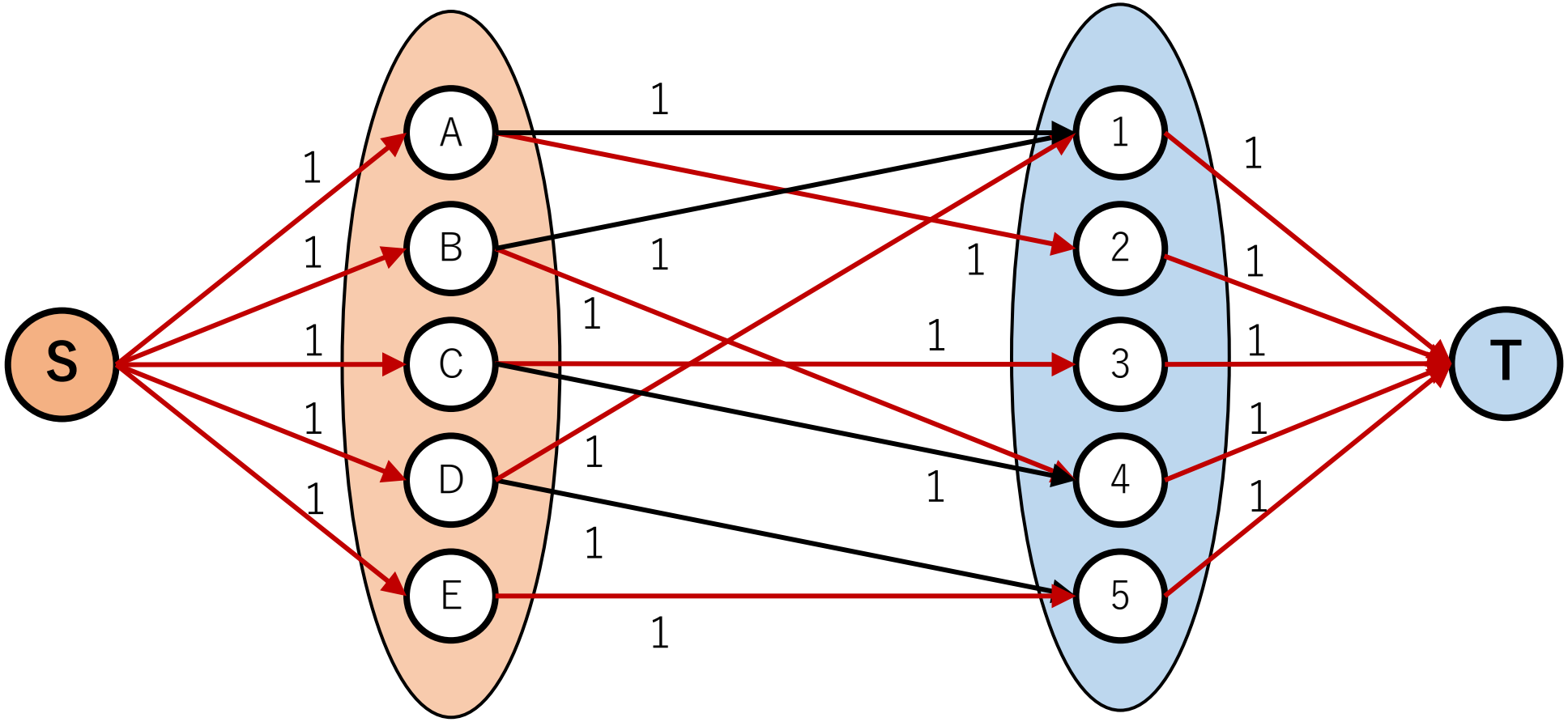
二部グラフの最大マッチング問題

Sから向かう辺，Tに向かう辺も容量を1にすることで，
「ペア」になることを保証する．



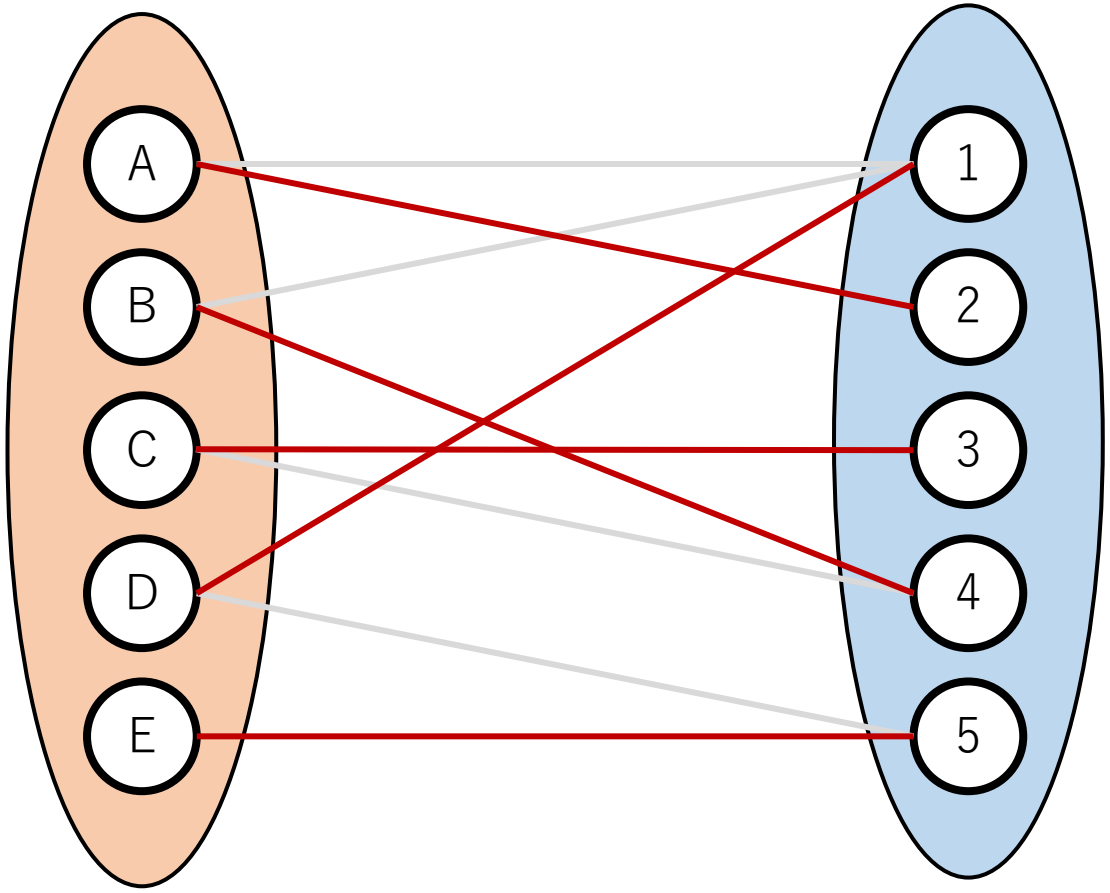
二部グラフの最大マッチング問題

フォード・ファルカーソン法などにより最大流となる経路の組み合わせを求める.



二部グラフの最大マッチング問題

S，Tを取り除くと最大マッチングになっている．



まとめ

最大流問題

フォード・ファルカーソン法

二部グラフの最大マッチング問題

まとめ

フローの問題はいろんな応用があり，自在に使えるととても強い武器になります．

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，その他のアルゴリズムに進むと良いかと思います．

コードチャレンジ：基本課題#12 [2点]

スライドを参考にしてフォード・ファルカーソン法を実装してください.

今回の基本課題はこの1問のみで，これが最後の基本課題です.

コードチャレンジ：Extra課題#12 [3点]

フローアルゴリズムに関する問題.

今回が最後のExtra課題になります. 😊