離散数学

# 最小全域木と最大流問題

落合 秀也

# 今日の内容

- 最小全域木
  - プリムのアルゴリズム

- 最大流問題
  - フォード・ファルカーソンのアルゴリズム

# 今日の内容

- 最小全域木
  - プリムのアルゴリズム

- 最大流問題
  - フォード・ファルカーソンのアルゴリズム

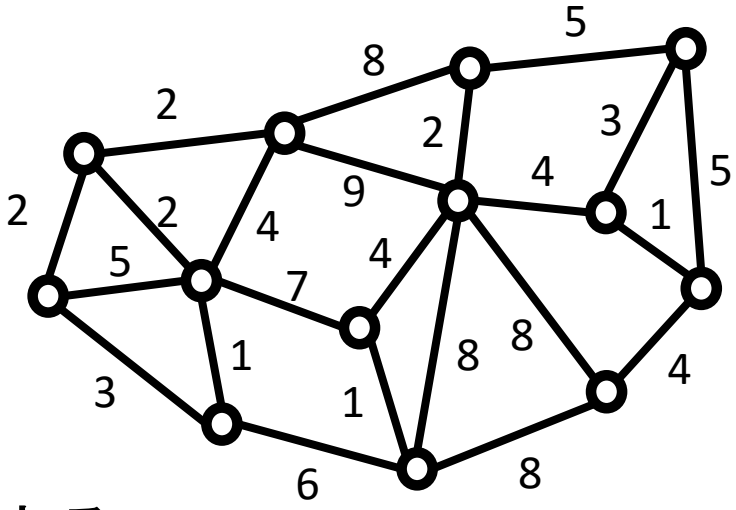
# 最小全域木を考える
# Minimum Spanning Tree Problem

- ラベル付(重み付) グラフ G(V, E)が与えられたとき、ラベルの和が最小となる全域木を作りたい
- 全域木 G(V, T)
  - Vのすべての頂点で構成される木
- 最小全域木

  $\sum_{e \in T} Ce$

  を最小にする T で作られる G(V,T)

  (Ceは辺eのラベル)

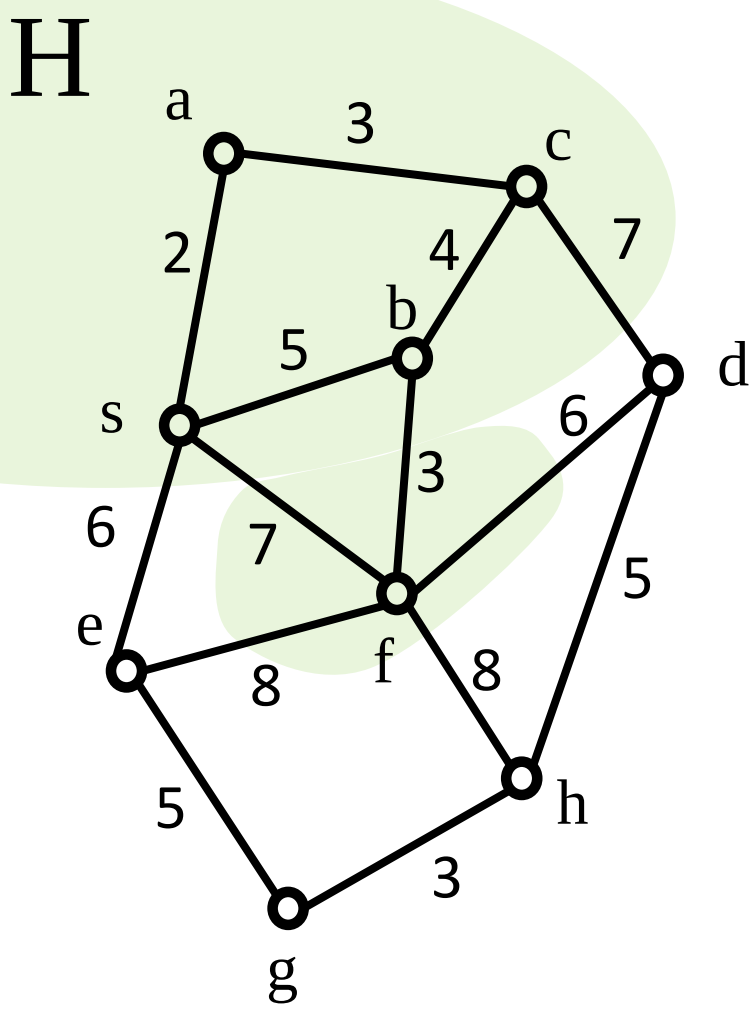



例:ラベルを地点間の配線コストとする

このとき、全地点がつながるネットワークを

最小コストで配線したい

# プリム法による最小全域木の作成

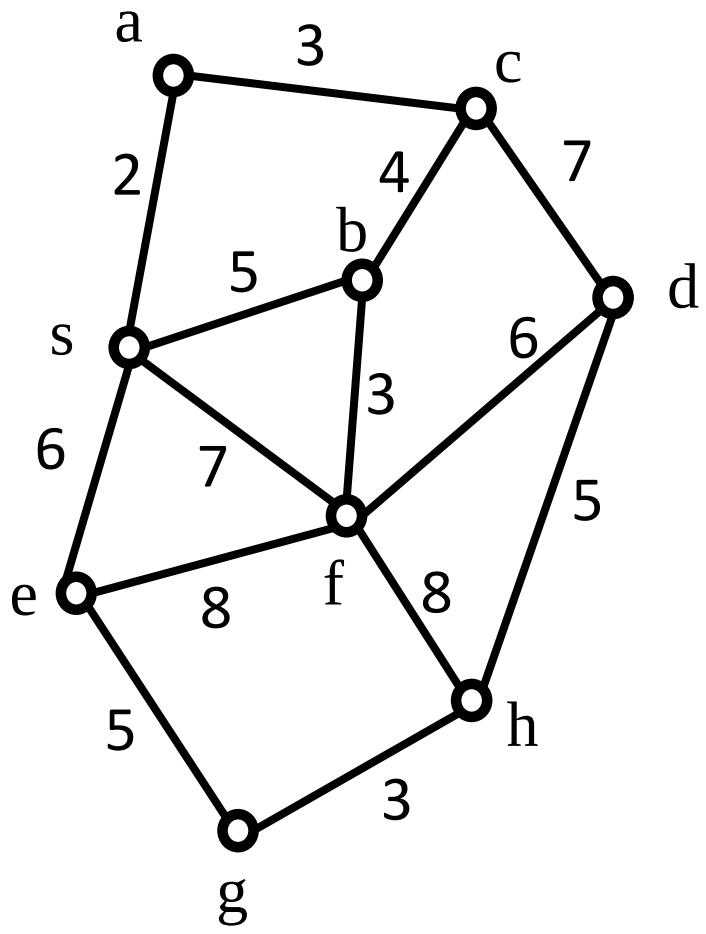



※ 考え方

- 任意の頂点を選び開始点sとする
- Tを辺の集合とする (開始時は空とする)
- Hの領域まで最小全域木の一部を作っているとする。
- Hから出る辺のうち、そのコストが最小の辺で接続する頂点をHに取り込む・・・ (*)
  - その辺をTに追加する
- H=Vとなるまで(*)を繰り返す
- G(V,T) が最小全域木となる

# プリムのアルゴリズム: 素直な実装

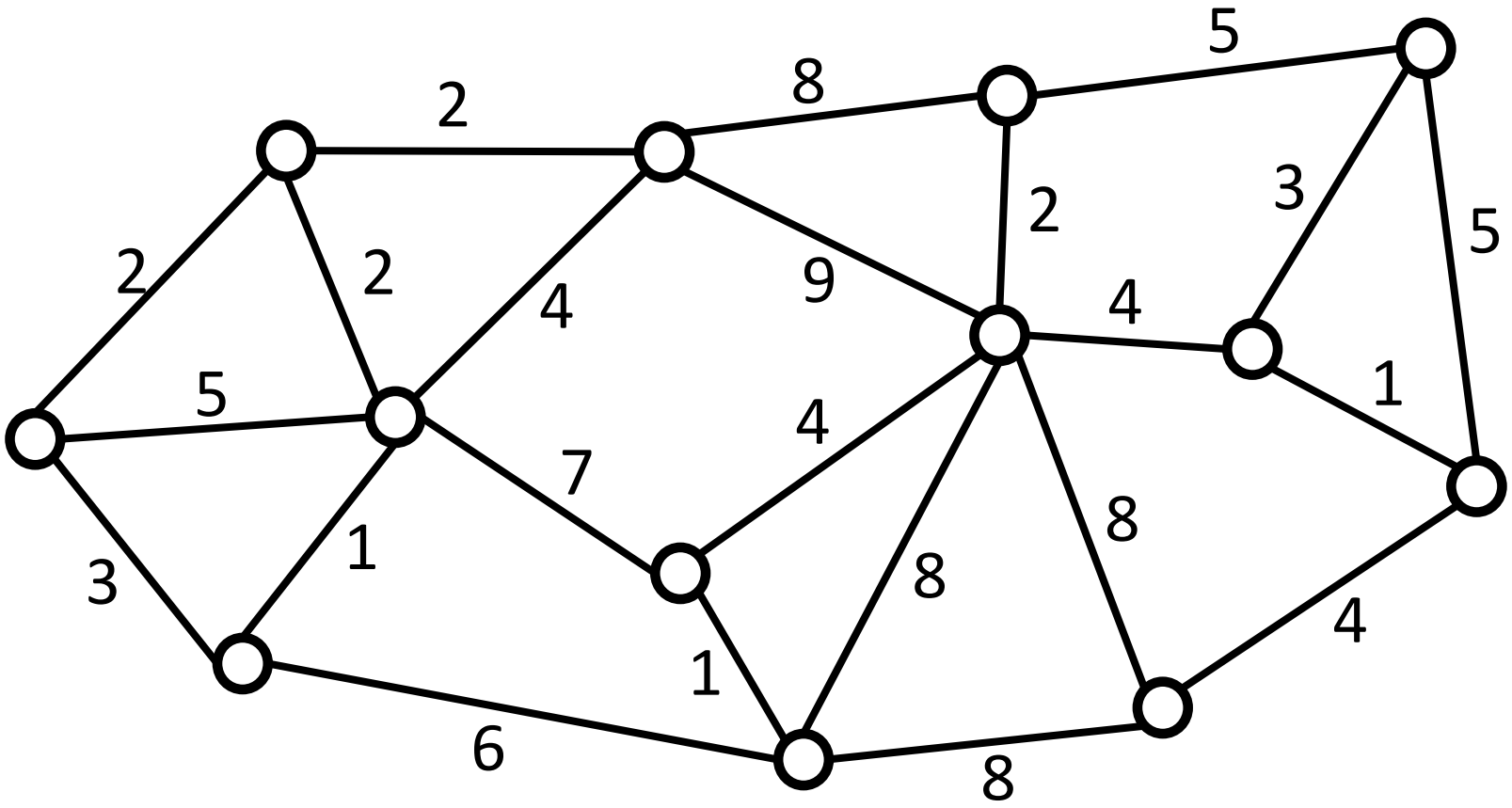
```
H := {s}
T := {}
While H ≠ V
  min_length := ∞, min_edge := null
  Foreach (u,v) : u∈H, v∈H-V
    If min_length > length(u,v) Then,
      min_length := length(u,v)
      min_edge := (u,v)
    EndIf
  EndFor
  T.add ( min_edge )
  H.add( min_edge.right )
EndWhile
```



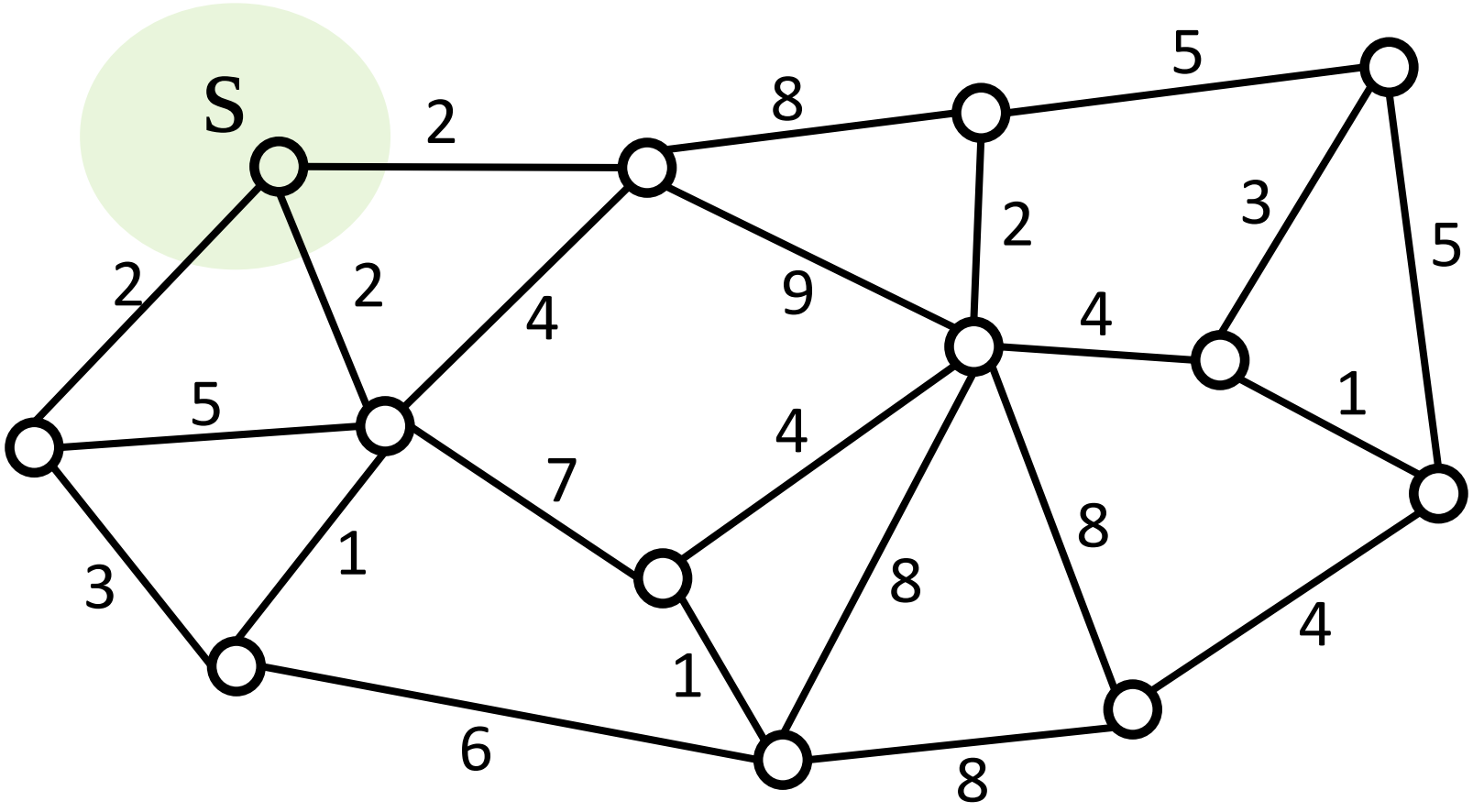

時間計算量 O(nm)
※この疑似コードの場合

# 練習

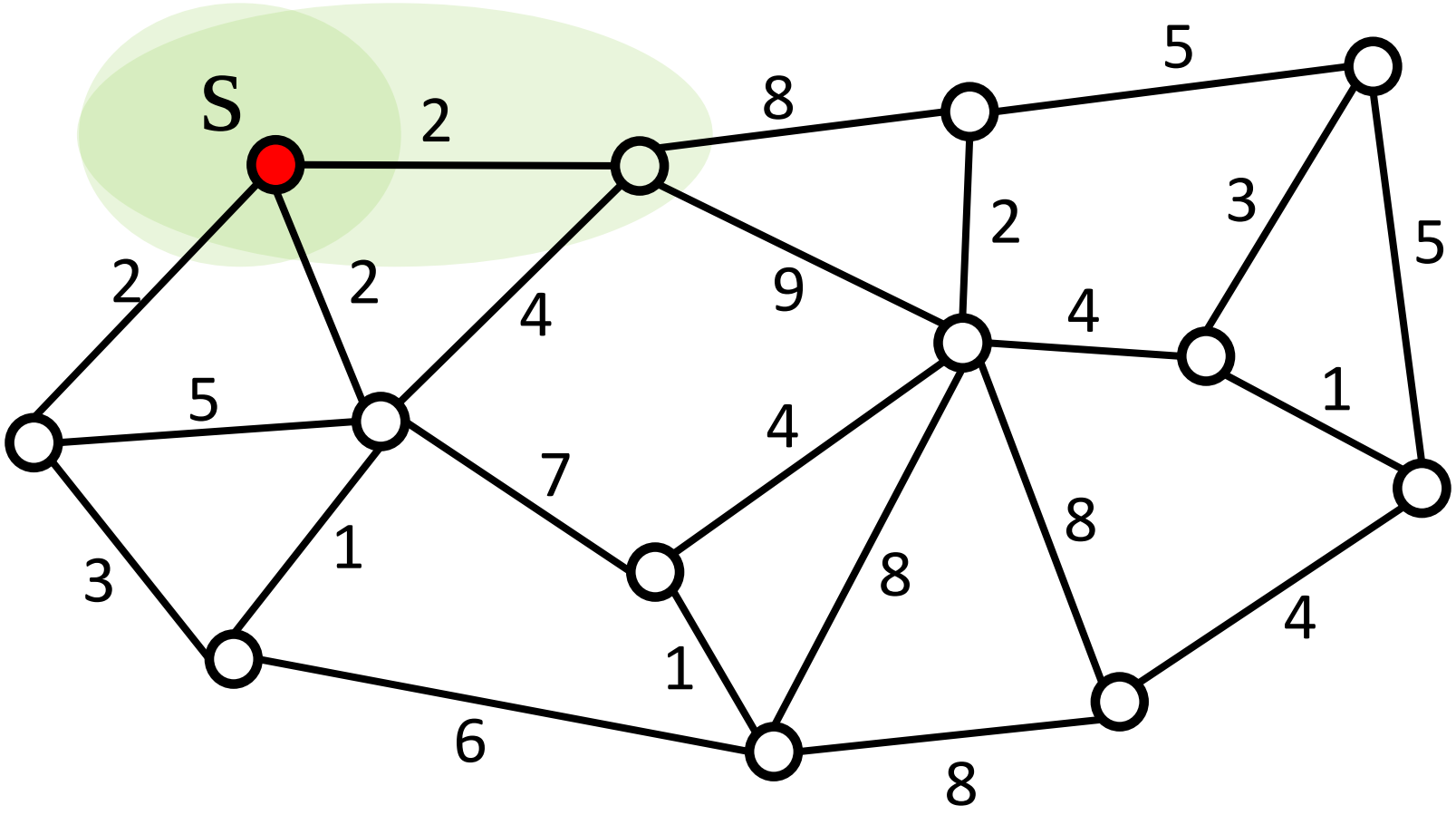
- プリム法により、以下のグラフに対する最小全域木を作成せよ



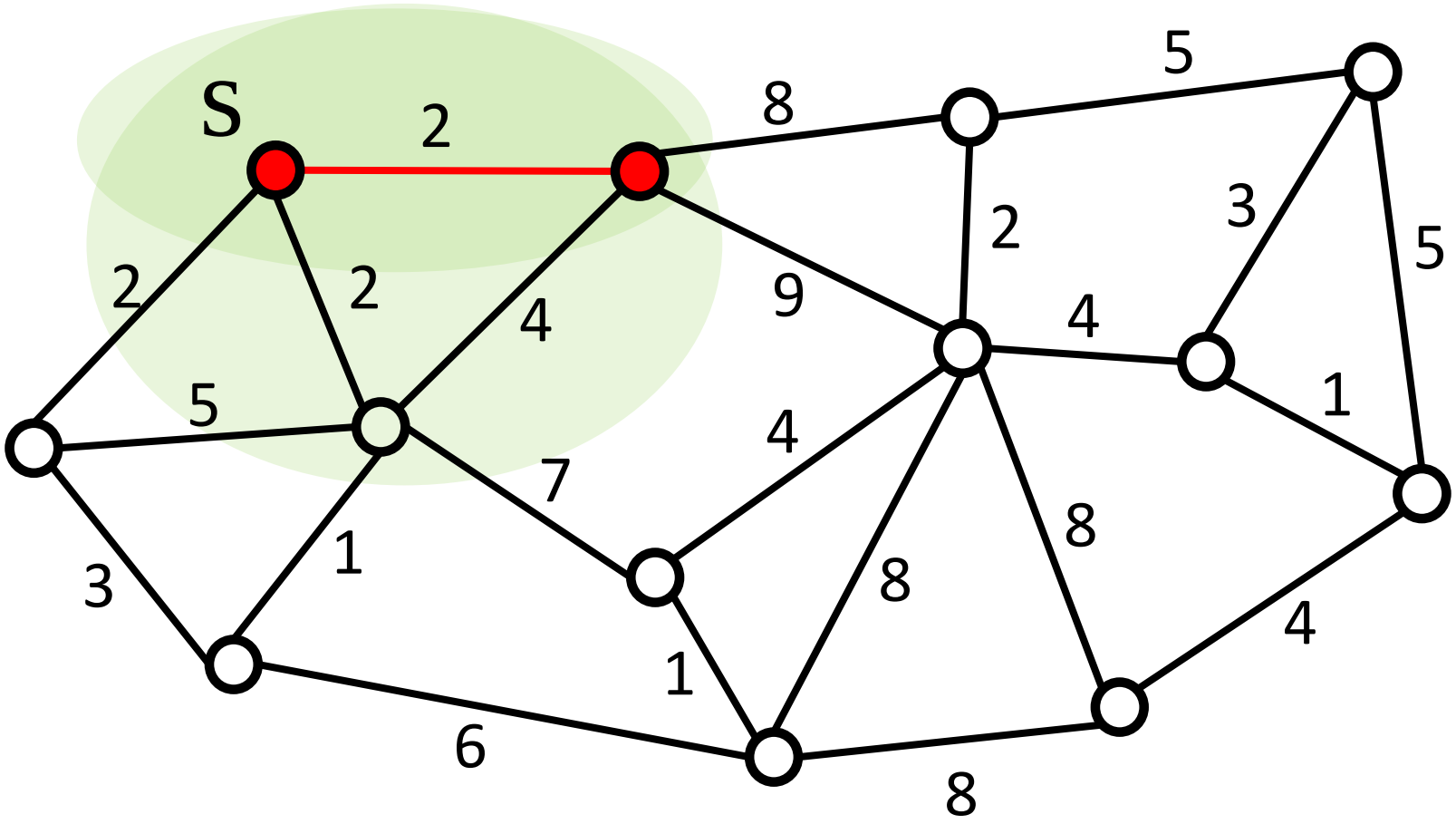
# 1. 開始点を選ぶ



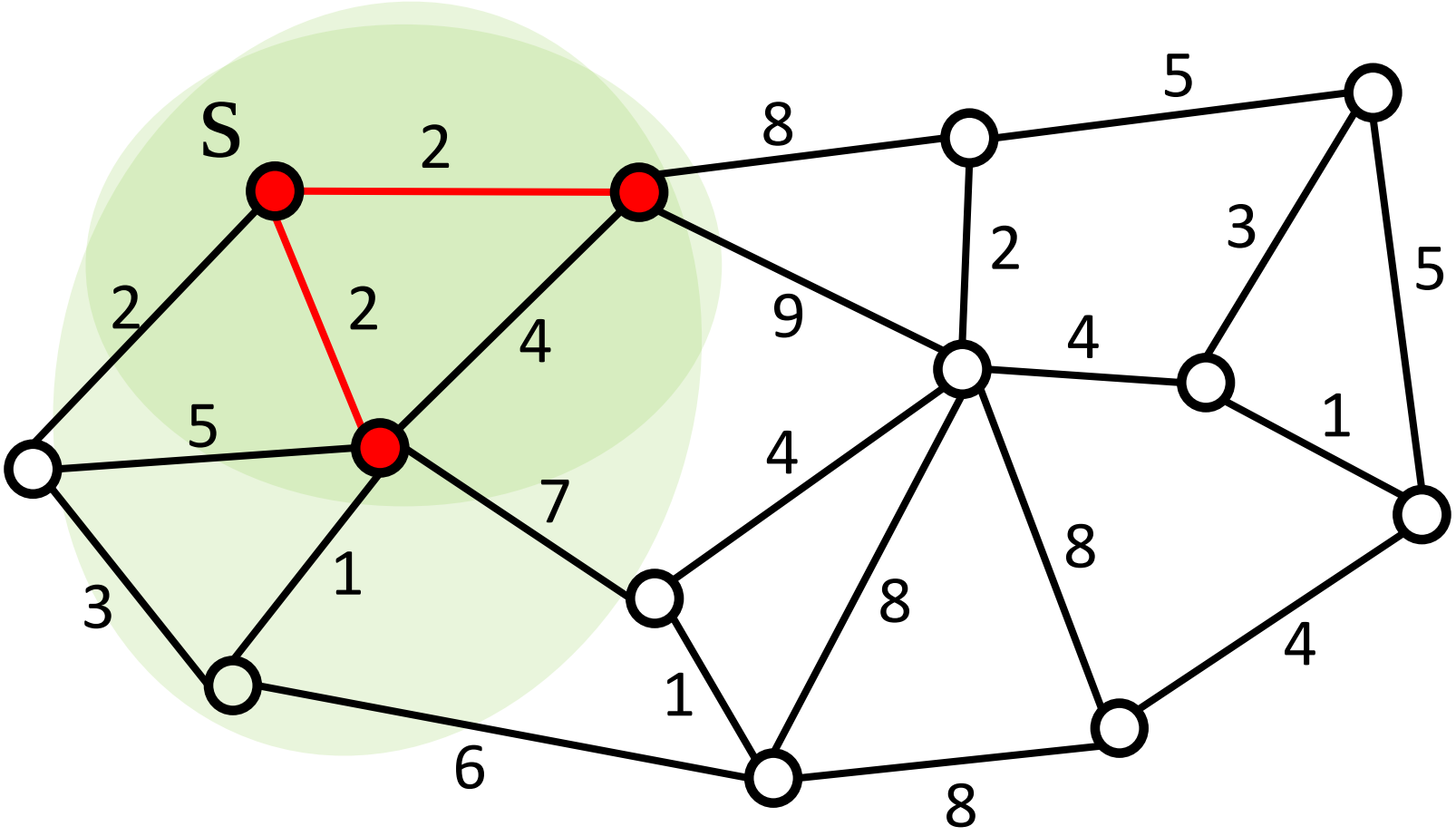
# 2. コストが最小の辺を選び接続する頂点を取り込む (1/12)



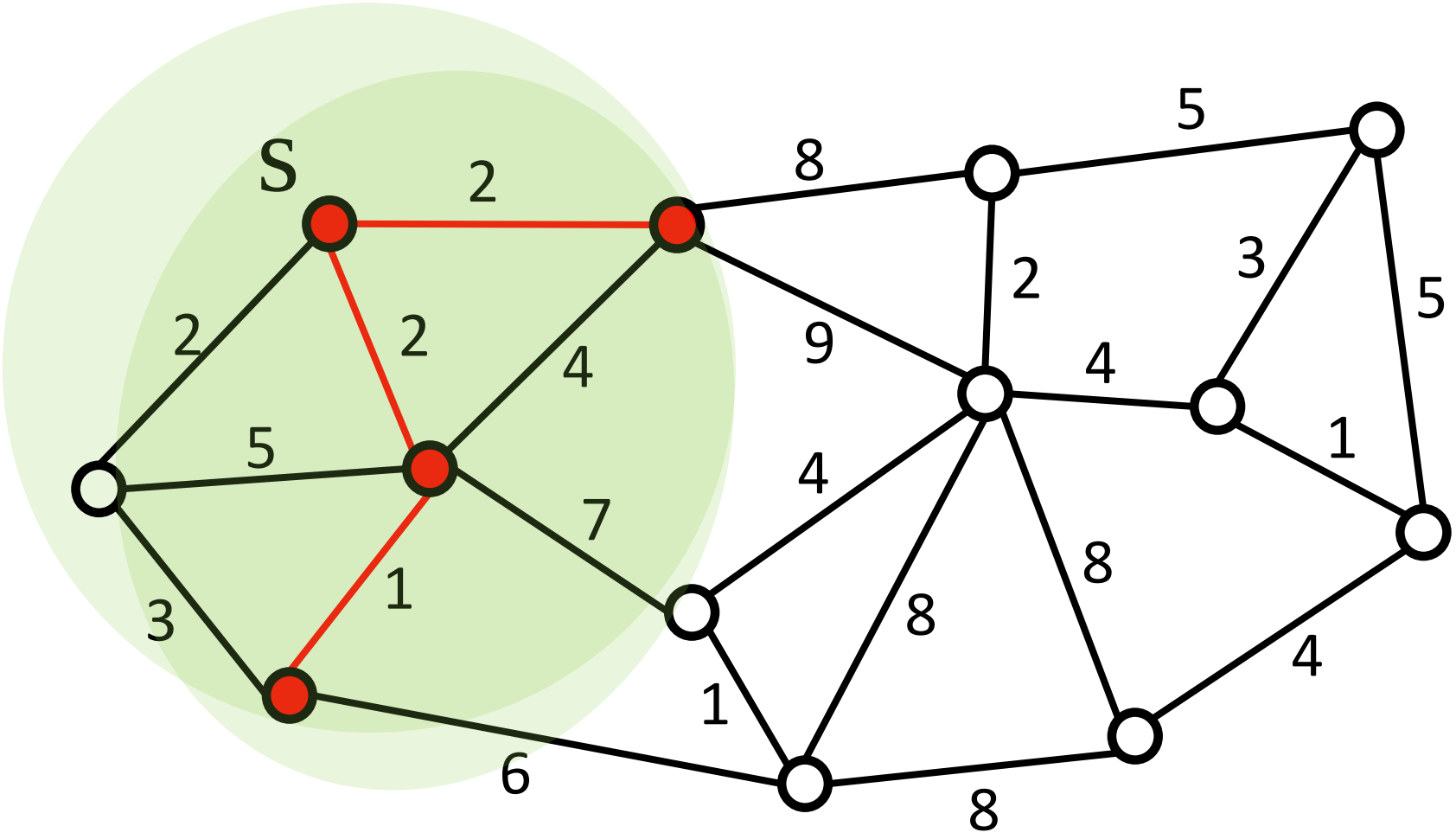
# 2. コストが最小の辺を選び接続する頂点を取り込む (2/12)



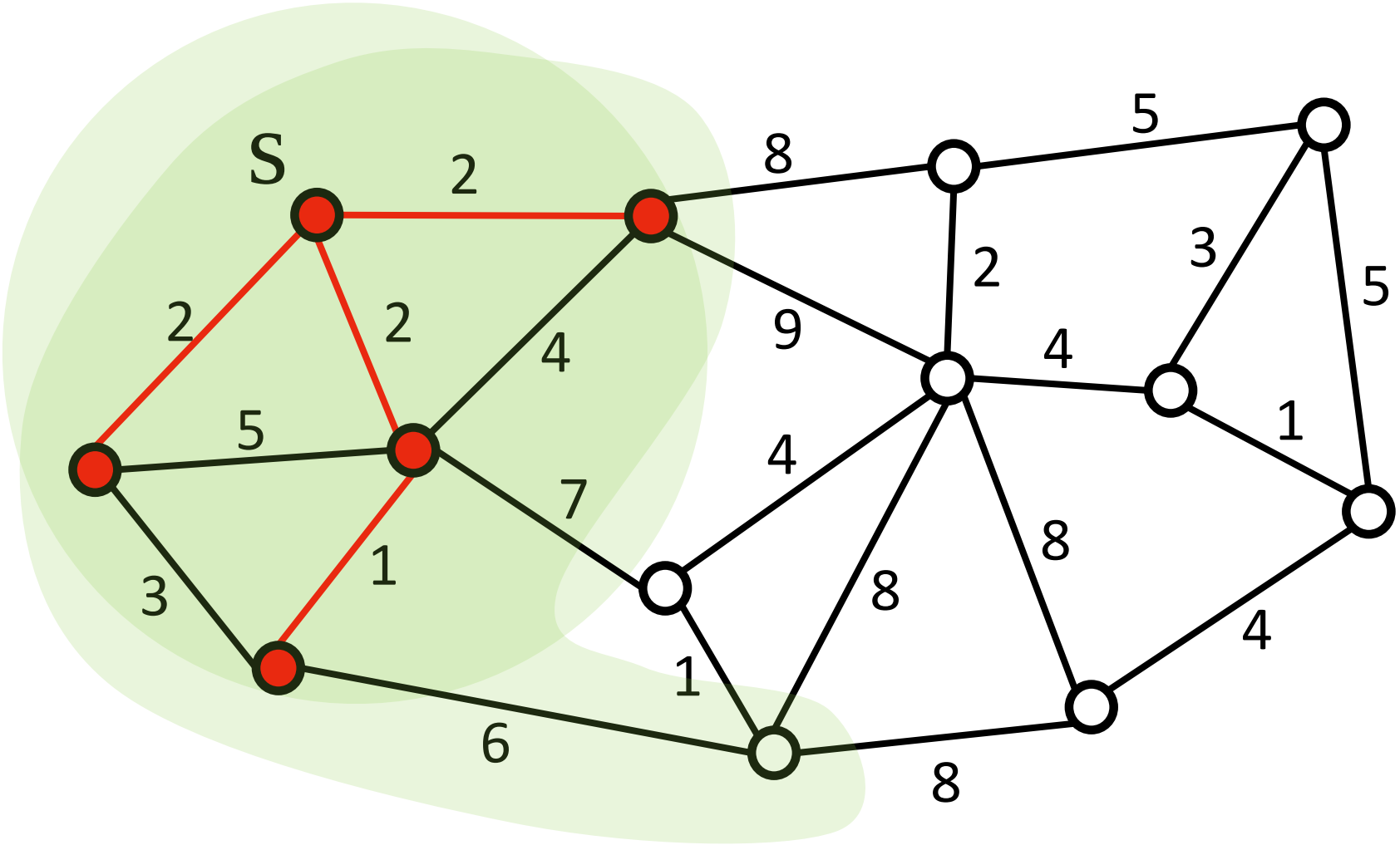
# 2. コストが最小の辺を選び接続する頂点を取り込む (3/12)



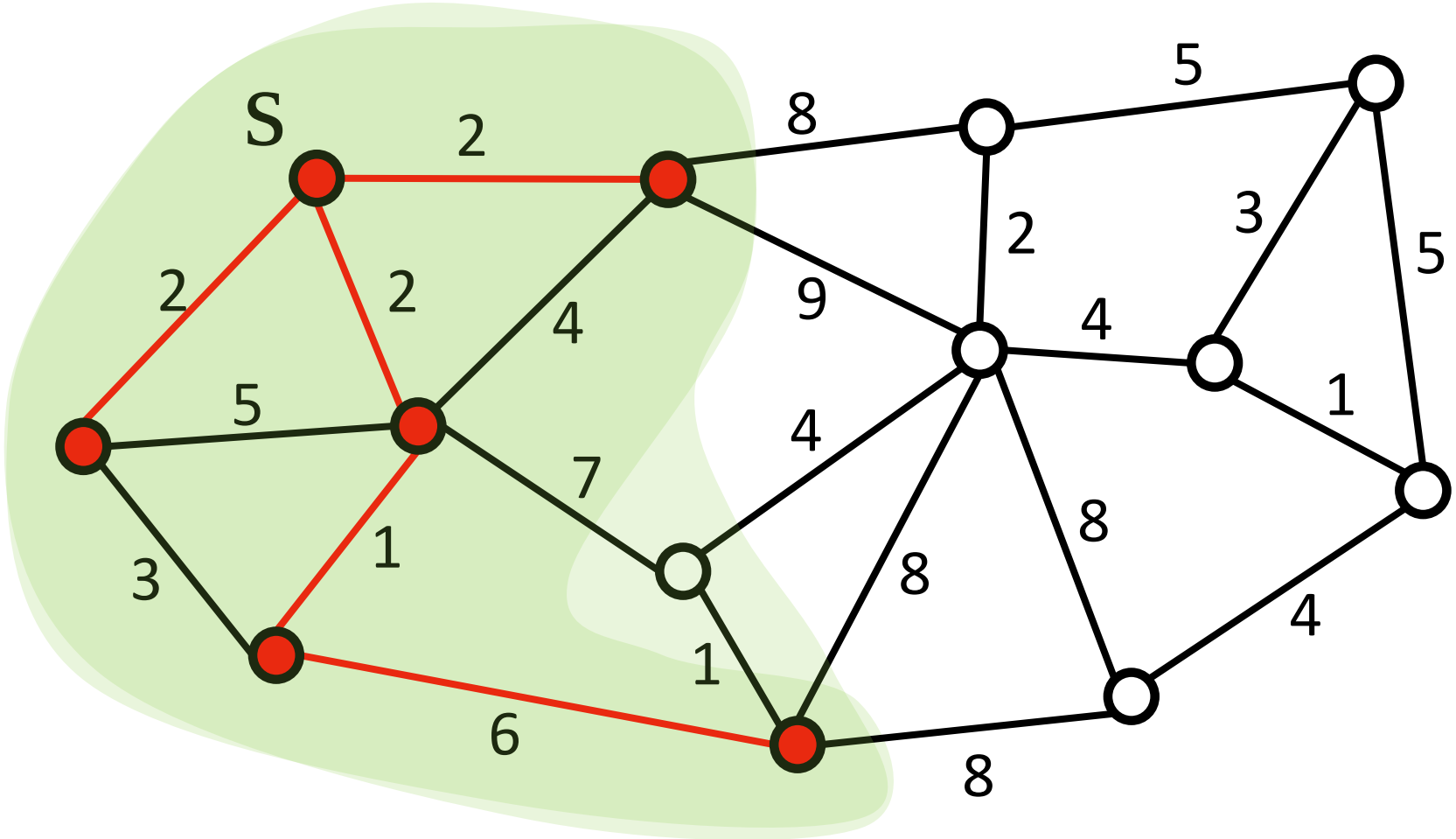
# 2. コストが最小の辺を選び接続する頂点を取り込む (4/12)



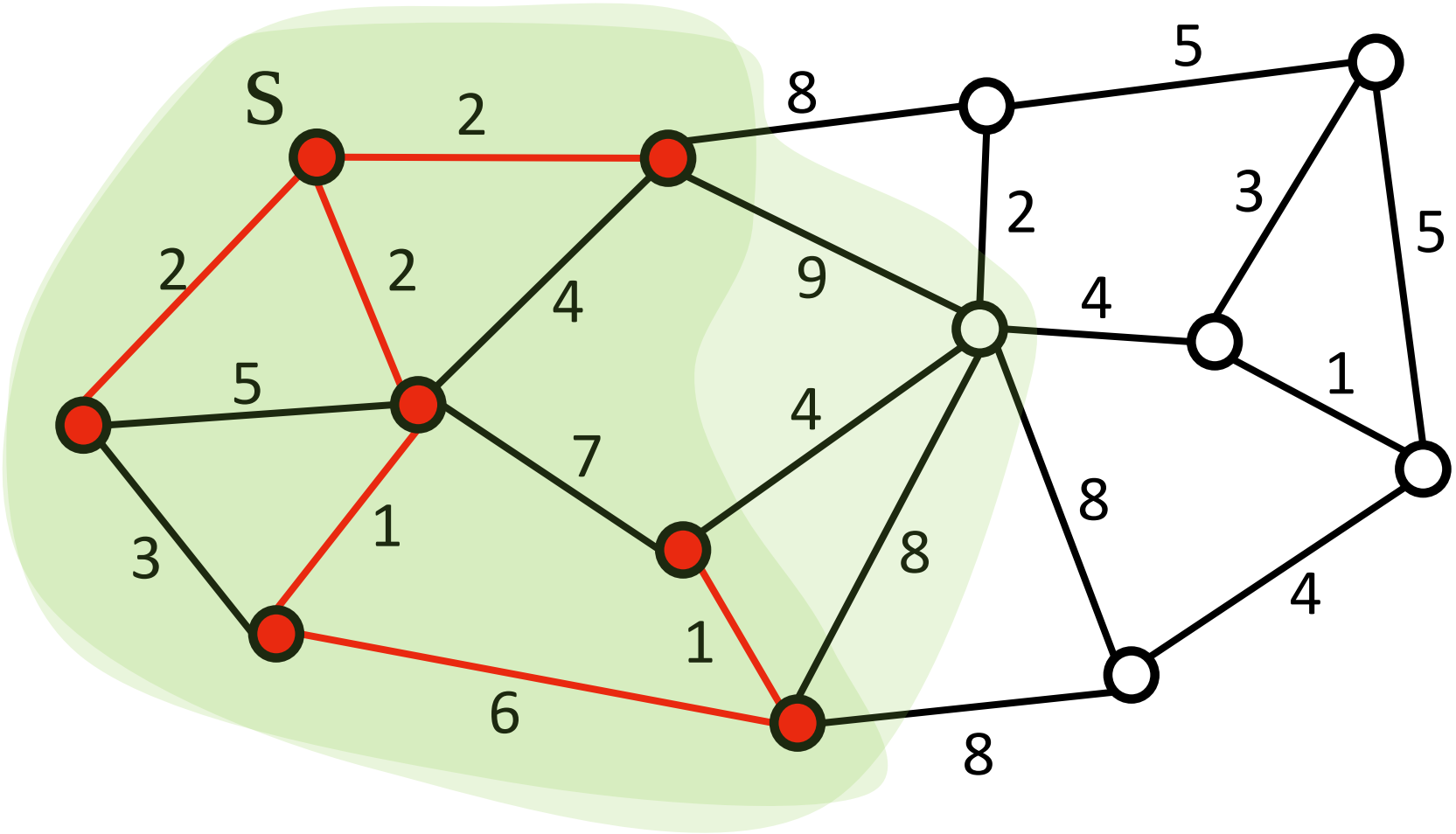
# 2. コストが最小の辺を選び接続する頂点を取り込む (5/12)



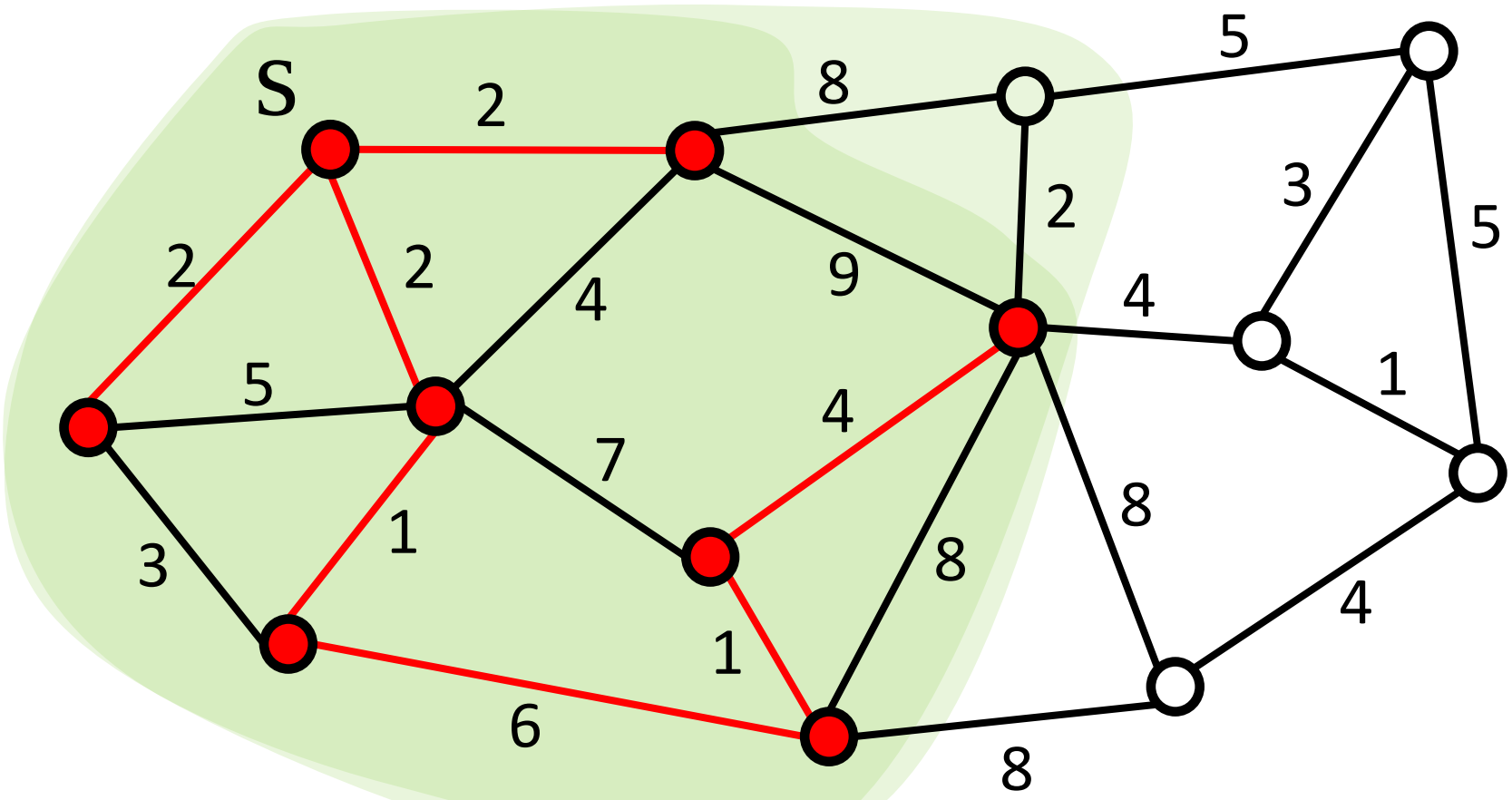
# 2. コストが最小の辺を選び接続する頂点を取り込む (6/12)



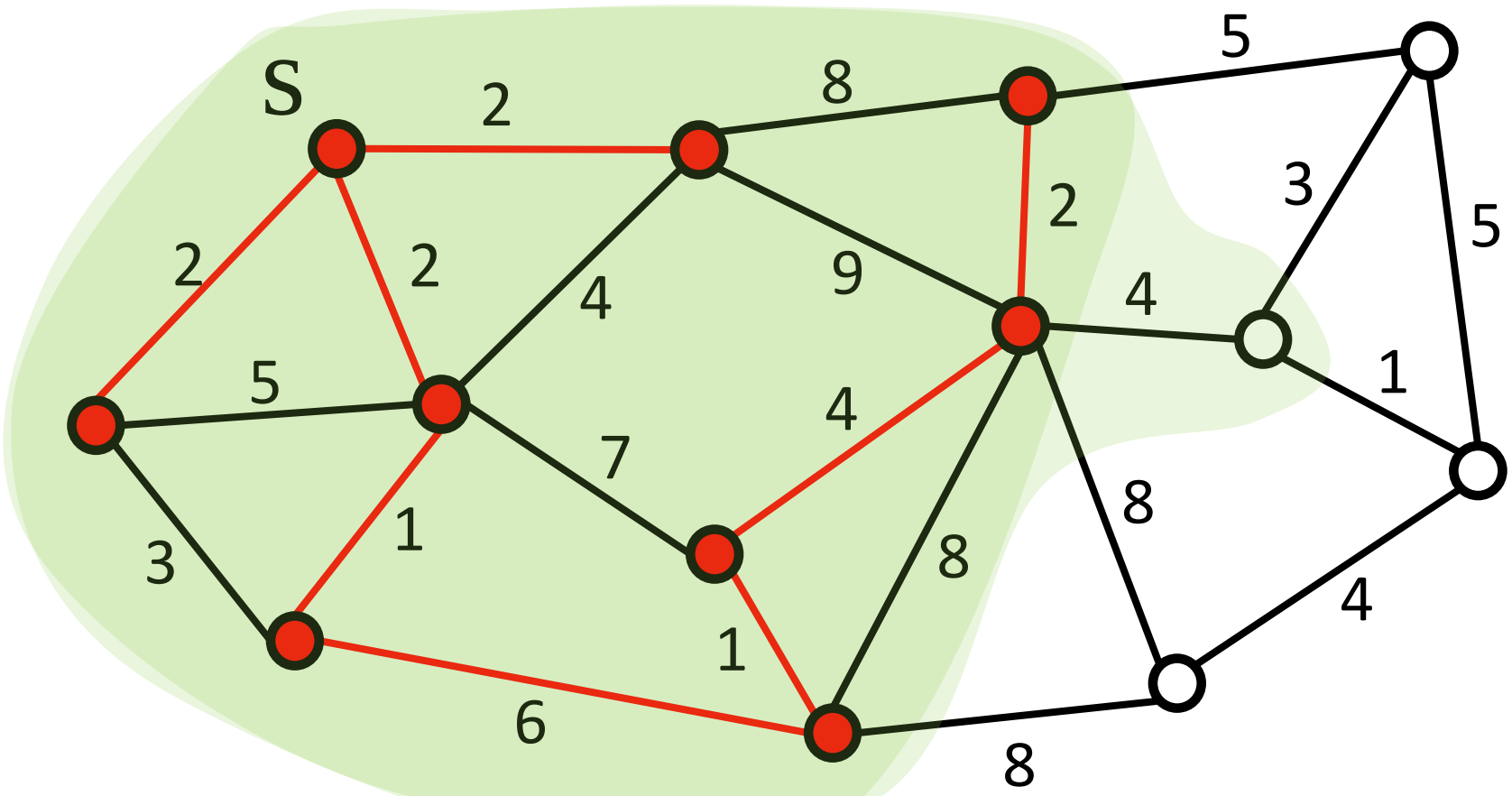
# 2. コストが最小の辺を選び 接続する頂点を取り込む (7/12)



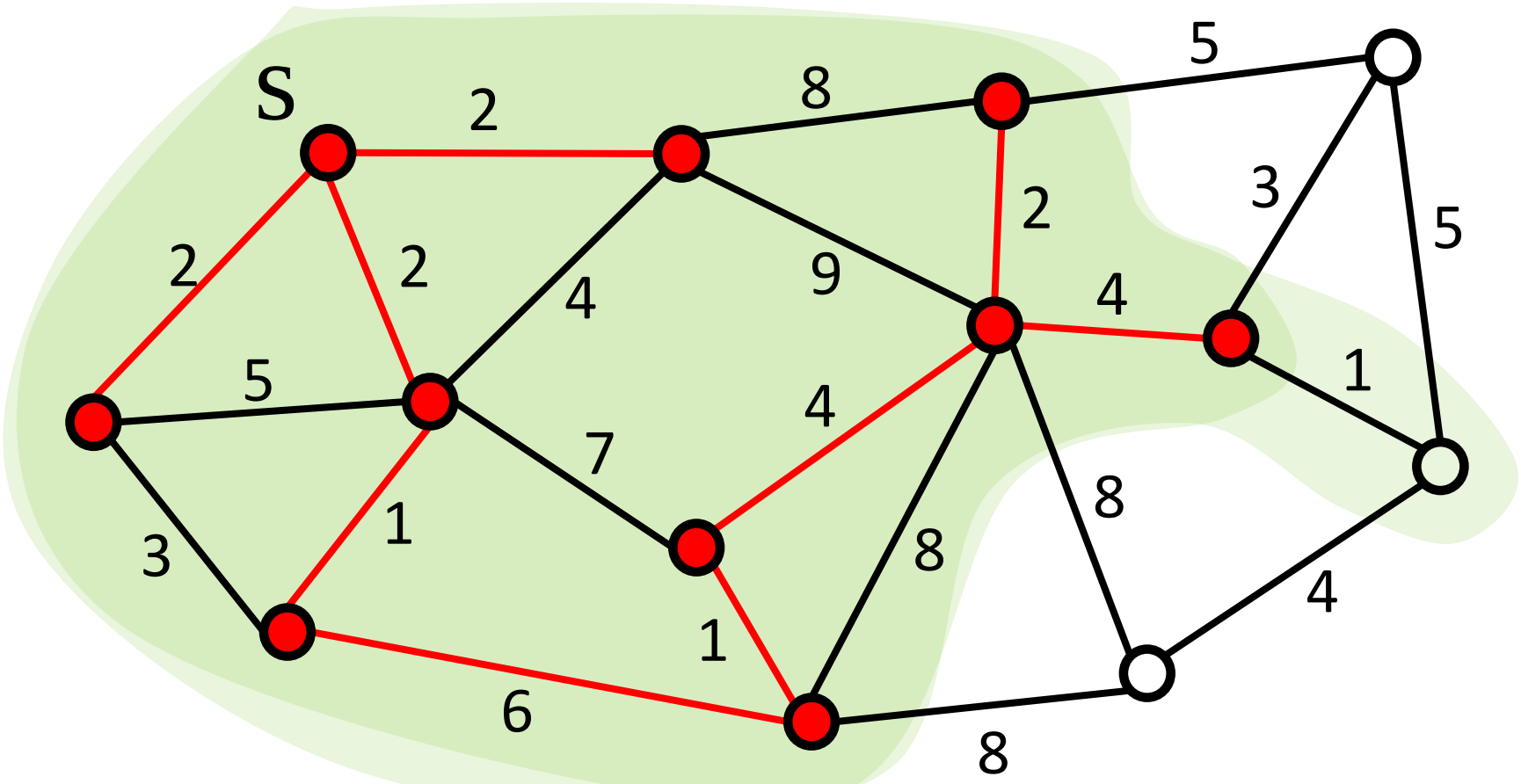
# 2. コストが最小の辺を選び 接続する頂点を取り込む (8/12)



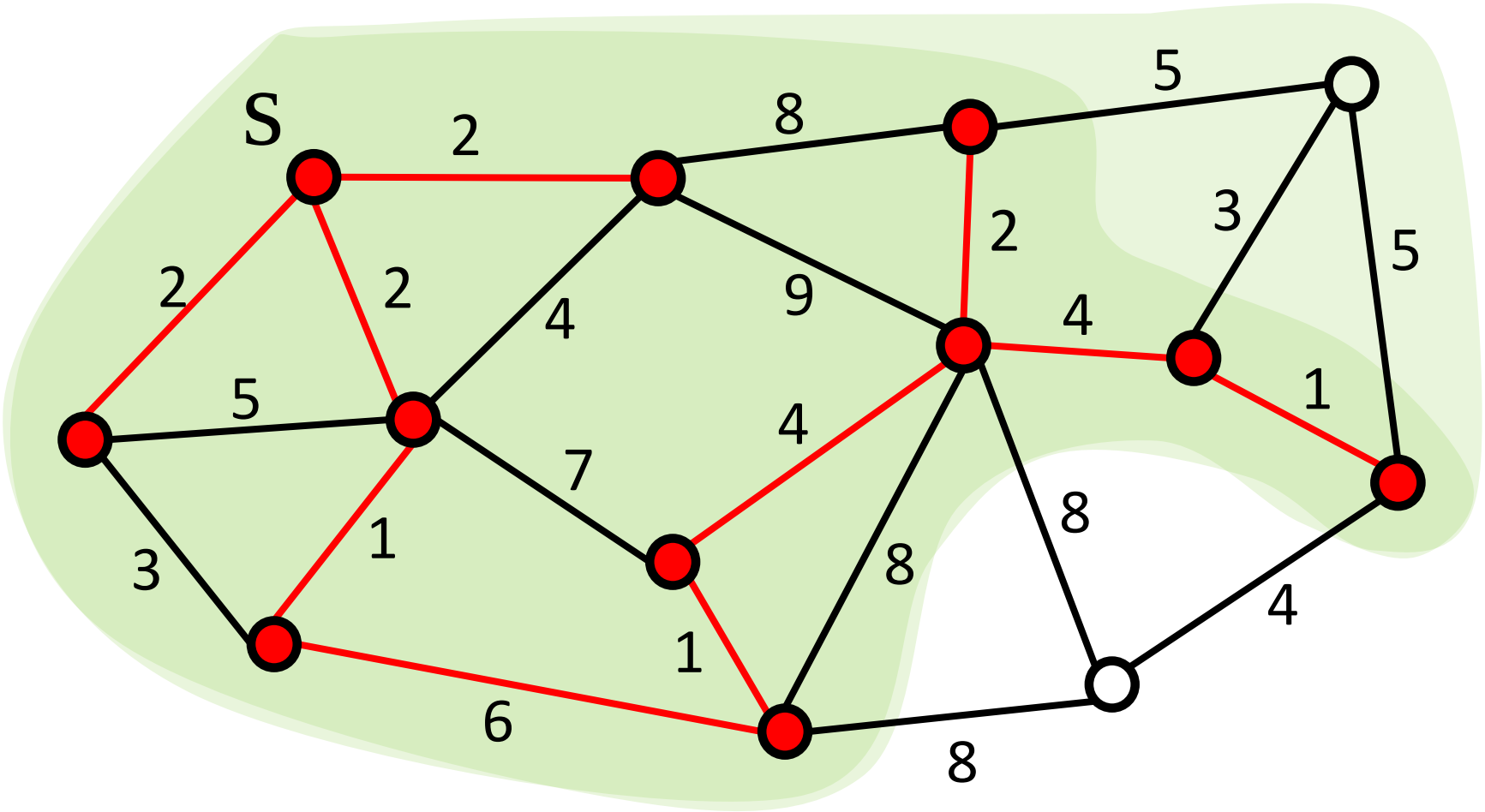
# 2. コストが最小の辺を選び 接続する頂点を取り込む (9/12)



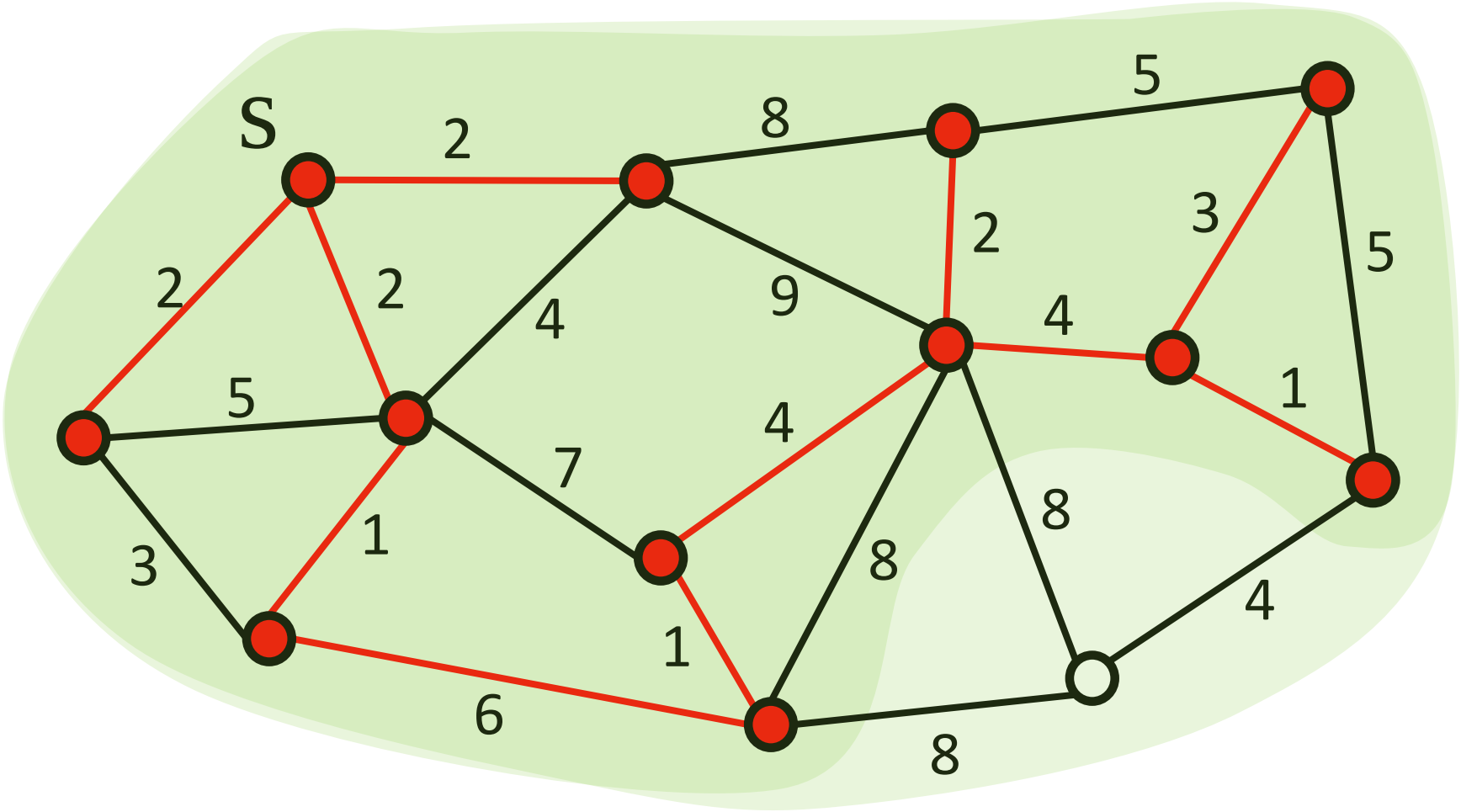
# 2. コストが最小の辺を選び 接続する頂点を取り込む (10/12)



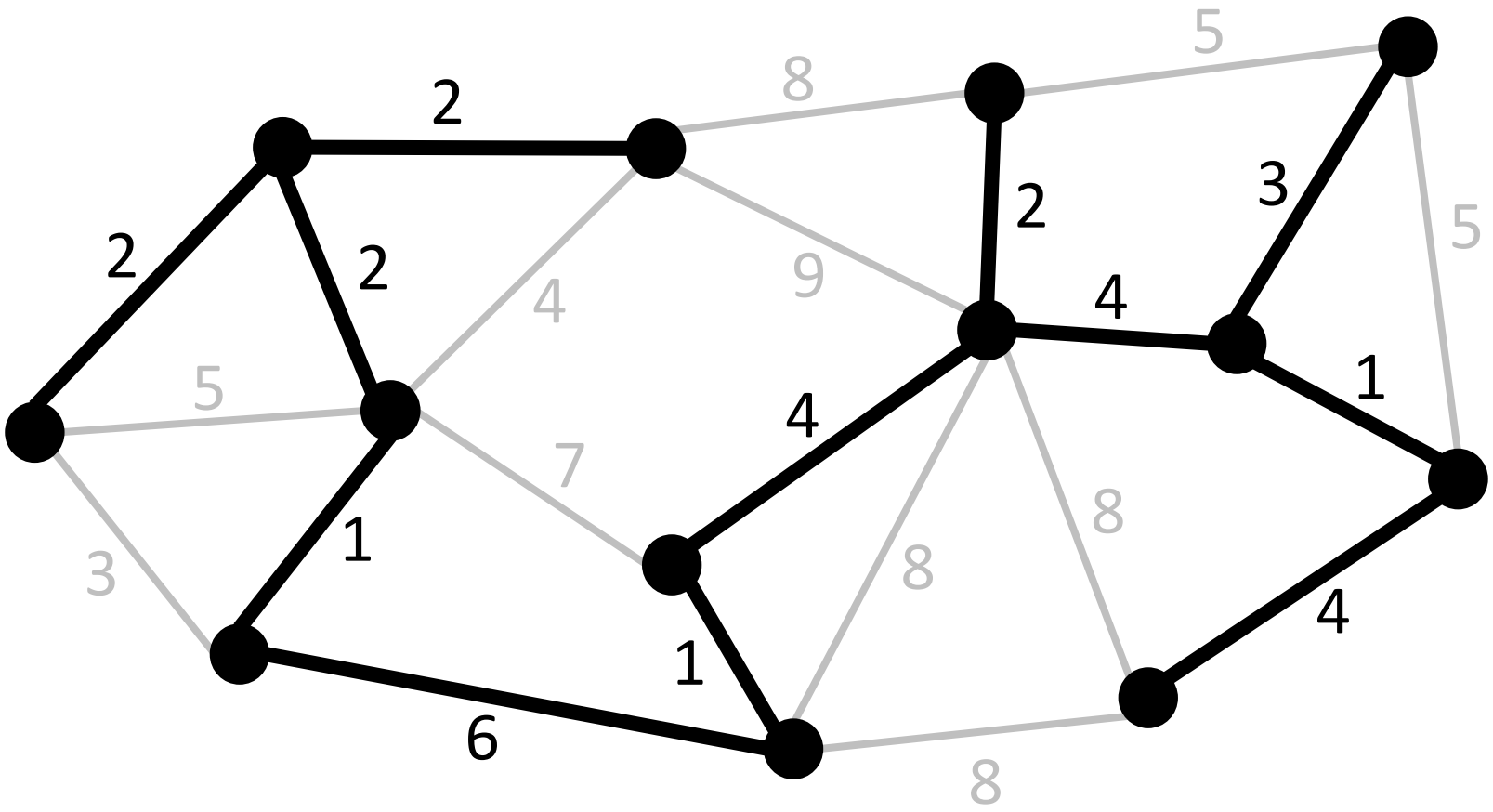
# 2. コストが最小の辺を選び 接続する頂点を取り込む (11/12)



# 2. コストが最小の辺を選び 接続する頂点を取り込む (12/12)



# 3. 最小全域木の完成

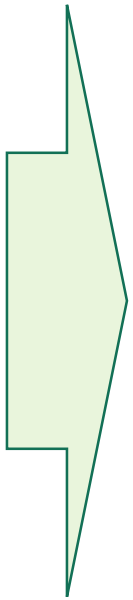


# プリムのアルゴリズム（改良版）

ダイクストラ法で工夫したように、
ダイクストラ法とよく似た形に実装できる

```
H := {s}
T := {}
While H ≠ V
  min_length := ∞, min_edge := null
  Foreach (u,v) : u∈H, v∈H-V
    If min_length > length(u,v) Then,
      min_length := length(u,v)
      min_edge := (u,v)
    EndIf
  EndFor
  T.add ( min_edge )
  H.add( min_edge.right )
EndWhile
```

改良前



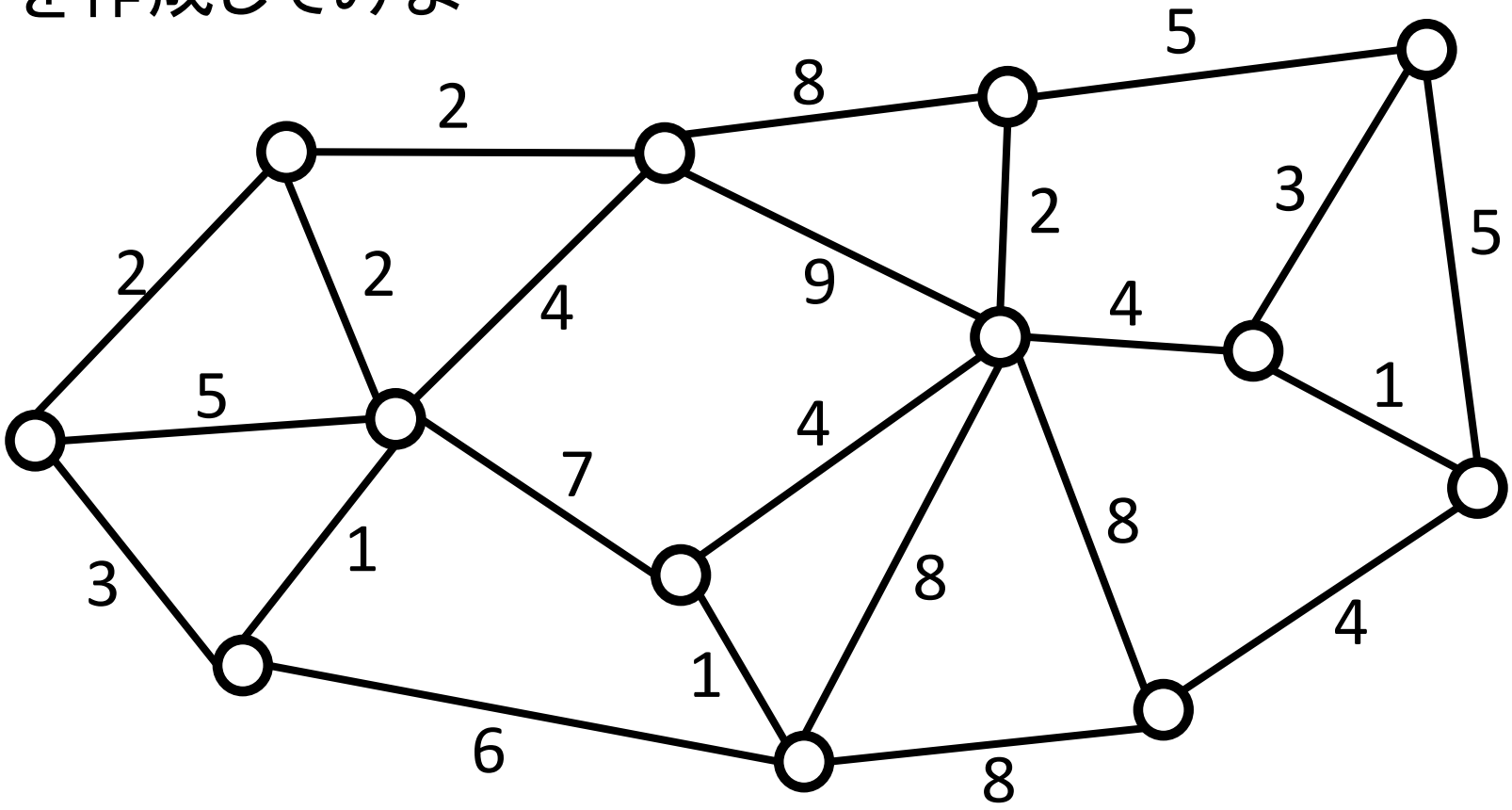
```
Foreach v ∈V  (v≠s)
  c[v] :=∞, prev[v]:=null
  Q.add(v)
EndFor
c[s] :=0

While Q is not empty
  u := Q.extractMin()
  Foreach v in Adj[u]
    If c[v] > length(u,v) Then,
      c[v] := length(u,v)
      prev[v] := u
      (  Q.changeKey()  )
    EndIf
  EndFor
EndWhile
```

改良後

# 確認

- プリム法（改良版）により、以下のグラフに対する最小全域木を作成してみよ

```
While Q is not empty
  u := Q.extractMin()
  Foreach v in Adj[u]
    If c[v] > length(u, v) Then,
      c[v] := length(u, v)
      prev[v] := u
    EndIf
  EndFor
EndWhile
```



# プリム法: 計算量の考察

```
While Q is not empty
  u := Q.extractMin()
  Foreach v in Adj[u]
    If c[v] > length(u, v) Then,
      c[v] := length(u, v)
      prev[v] := u
   (    Q.changeKey()    )
    EndIf
  EndFor
EndWhile
```

- While文の内部は n回実行される。
  - Foreach文の内部は nm回実行される。

  ➔ 実装によっては O(nm) となりえる。

- Q に リストや配列を使う場合
  - Q.extractMin()に O(n) の時間を要する
  - Q.extractMin()は n回 呼び出される

    ➔ $O(n^2)$

- Q に 2分ヒープを使う場合
  - Q.extractMin()に O(log n)の時間を要する
  - Q.extractMin()の呼び出しはn回ある → O(n log n)
  - c[v]の更新に伴うヒープの組換え処理に、O(log n)の時間を要する
  - c[v]の更新は、m回発生する → O(m log n)

    ➔ O( (n+m) log n )

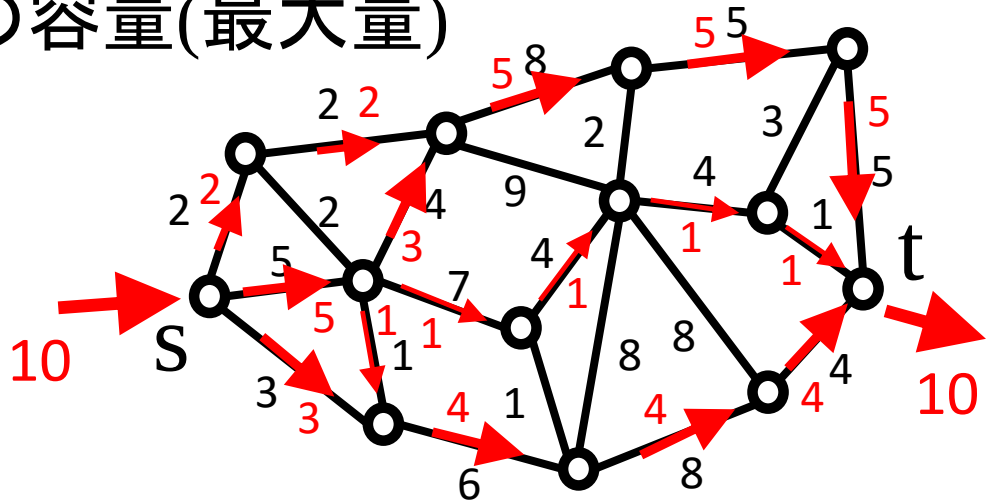
# 今日の内容

- 最小全域木
  - プリムのアルゴリズム

- 最大流問題
  - フォード・ファルカーソンのアルゴリズム

# 最大流問題
# Maximum Flow Problem

- ラベル付(重み付) グラフ G(V, E)が与えられたとき、流入点sから流出点tまでの総流量を最大化したい
- 辺のラベルの意味: 流量の容量(最大量)
  - 運べる荷物の量
  - 流せる水の量
  - 送れるパケット数




- このとき、

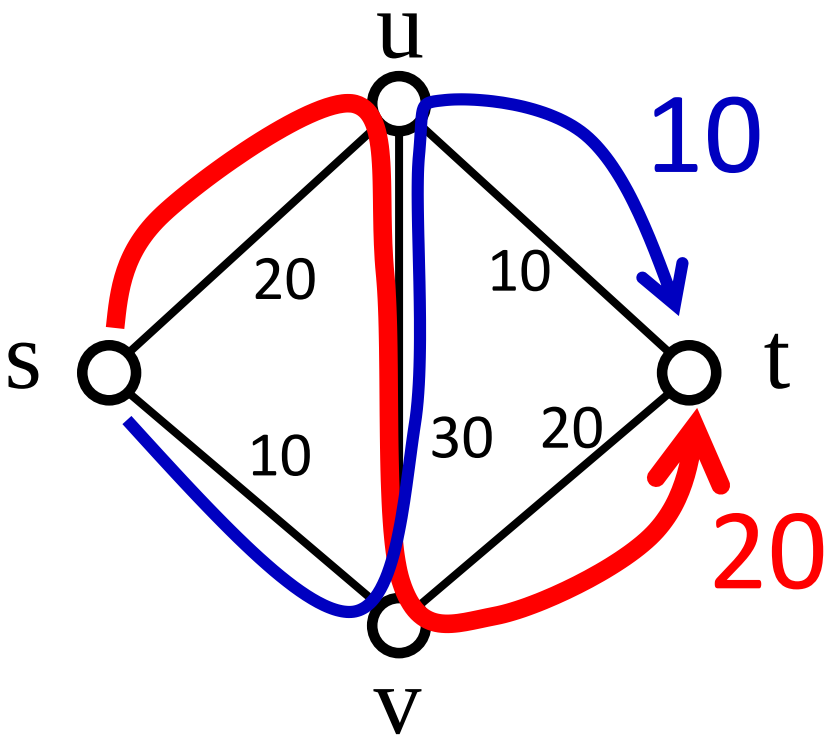
  sに流れ込む流量 = tから流れ出る流量 (グラフの外から見た場合)

  sから流れ出す流量 = tに流れ込む流量 (グラフの中で見た場合)

  である。

  この量の最大値(とそのときのフロー)をどう求めればよいか？

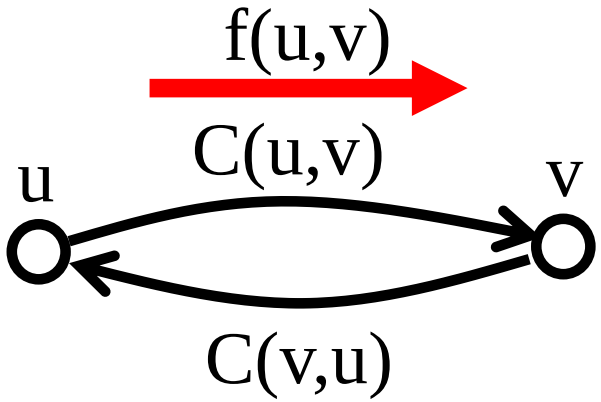
# 最大流問題：解決へのアプローチ




- s-t間の適当な道を考え、その流量上限（最小容量）を算出する･･･赤色のケース
- 赤色のフローを流しても余っている別のs-t間の道を考える(逆向きは押し戻すと考える)。この際の、流量上限を算出する･･･青色のケース
- 上記を繰り返していけば、徐々に総流量が増えてやがてそれ以上流せなくなる。これにより、ネットワーク全体の最大流が求まるのではないか？？

# フローの定式化

- 辺を流れるフロー



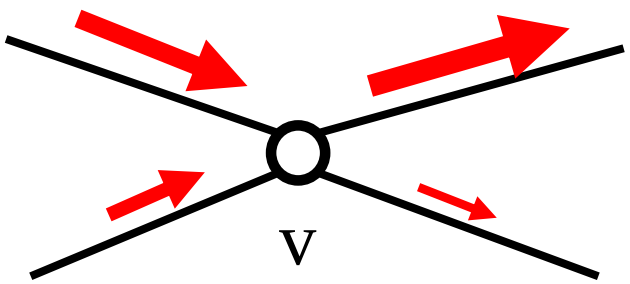

$f(u, v) \leqq C(u, v)$

$f(v, u) = - f(u, v)$

$\therefore \ -C(v, u) \leqq f(u, v) \leqq C(u, v)$

ただし $C(u, v) = C(v, u)$とは限らない

- 頂点への流入量と流出量



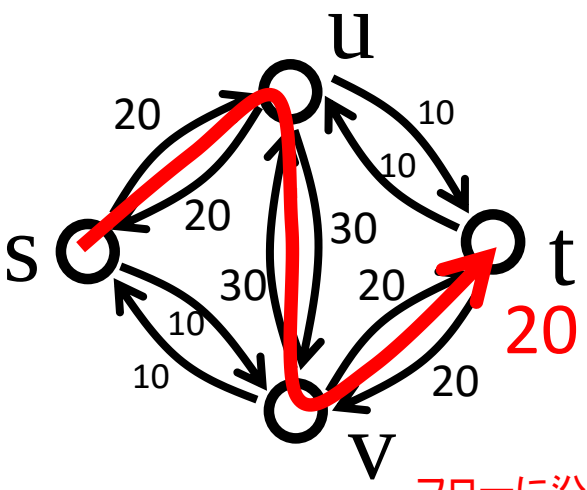

流入量: $f^{in}(v) = \sum_{u \in V,\, f(u,v)>0} f(u,v)$

流出量: $f^{out}(v) = \sum_{w \in V,\, f(v,w)>0} f(v,w)$

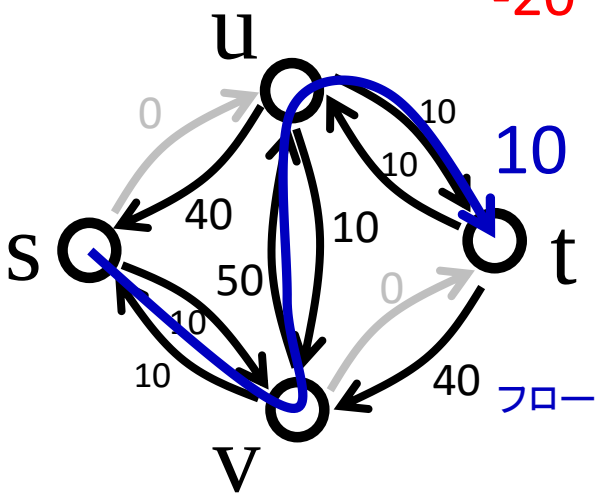
$v \neq s,t$ であれば $f^{in}(v) = f^{out}(v)$

総流量: $total(f) = f^{out}(s) = f^{in}(t)$

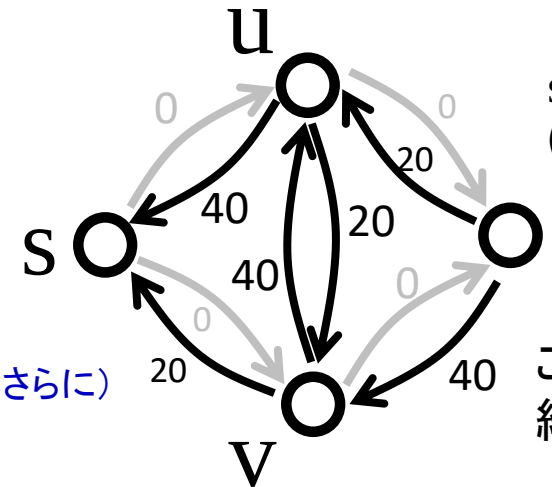
# フォード・ファルカーソン法
# Ford-Fulkerson Algorithm




フローに沿って
-20




フローに沿って（さらに）
-10




※ 考え方

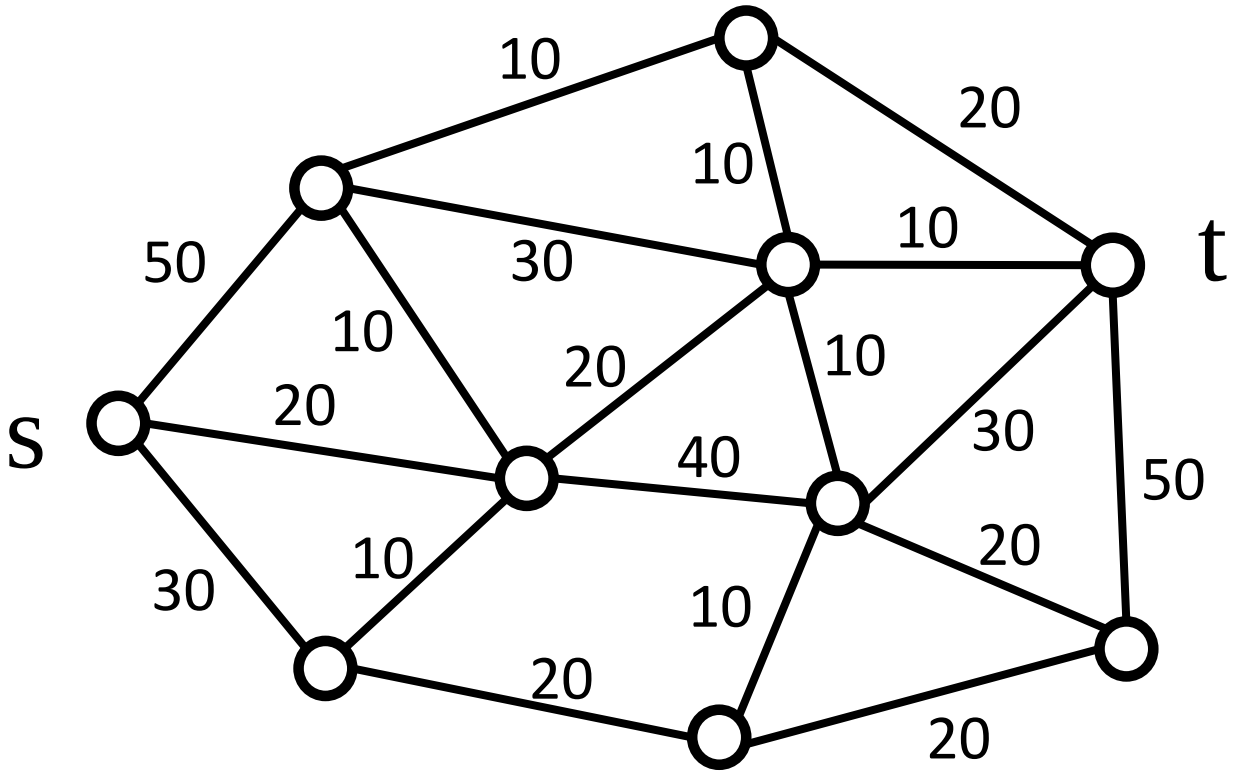
- s-t間の何らかの道を考え、その流量の上限を算出する
- そのフローfを容量Cから引いた値（残余容量）を計算し、グラフから差し引く・・・そのグラフをGfとする
- Gfに対して、同様のことをする（つまり、s-t間の何らかの道を考え、その流量の上限を算出し、総フローを算出し、容量から差し引いて・・・）
- s-t間の道が無くなれば、終了

s-t間の道がなくなった
(容量>0の弧についてのみ考える)

このとき差し引いている
総フローが求める最大流

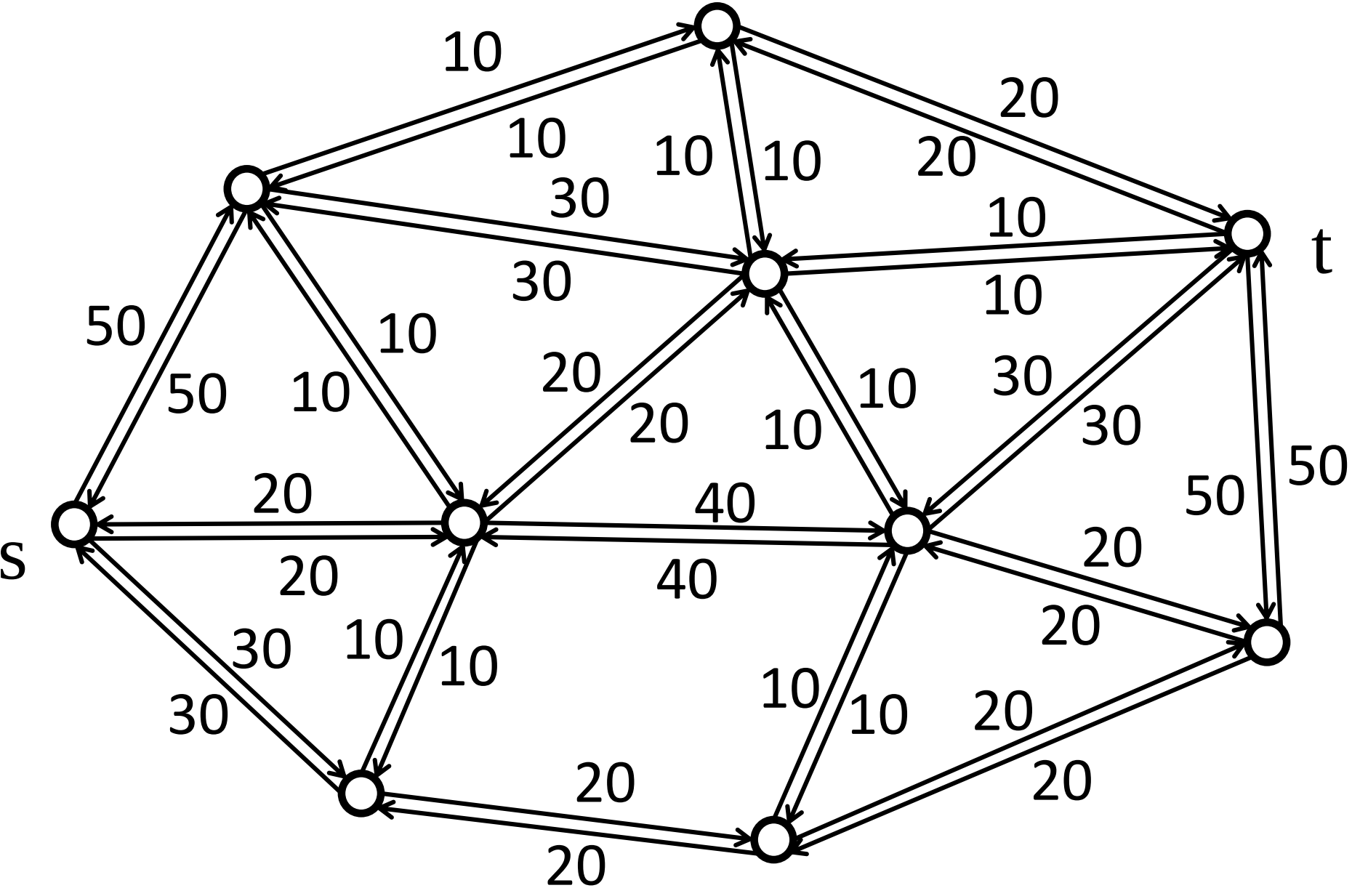
# 練習

- フォード・ファルカーソン法を用いて、以下のフローグラフにおける sから t への最大流を求めよ。

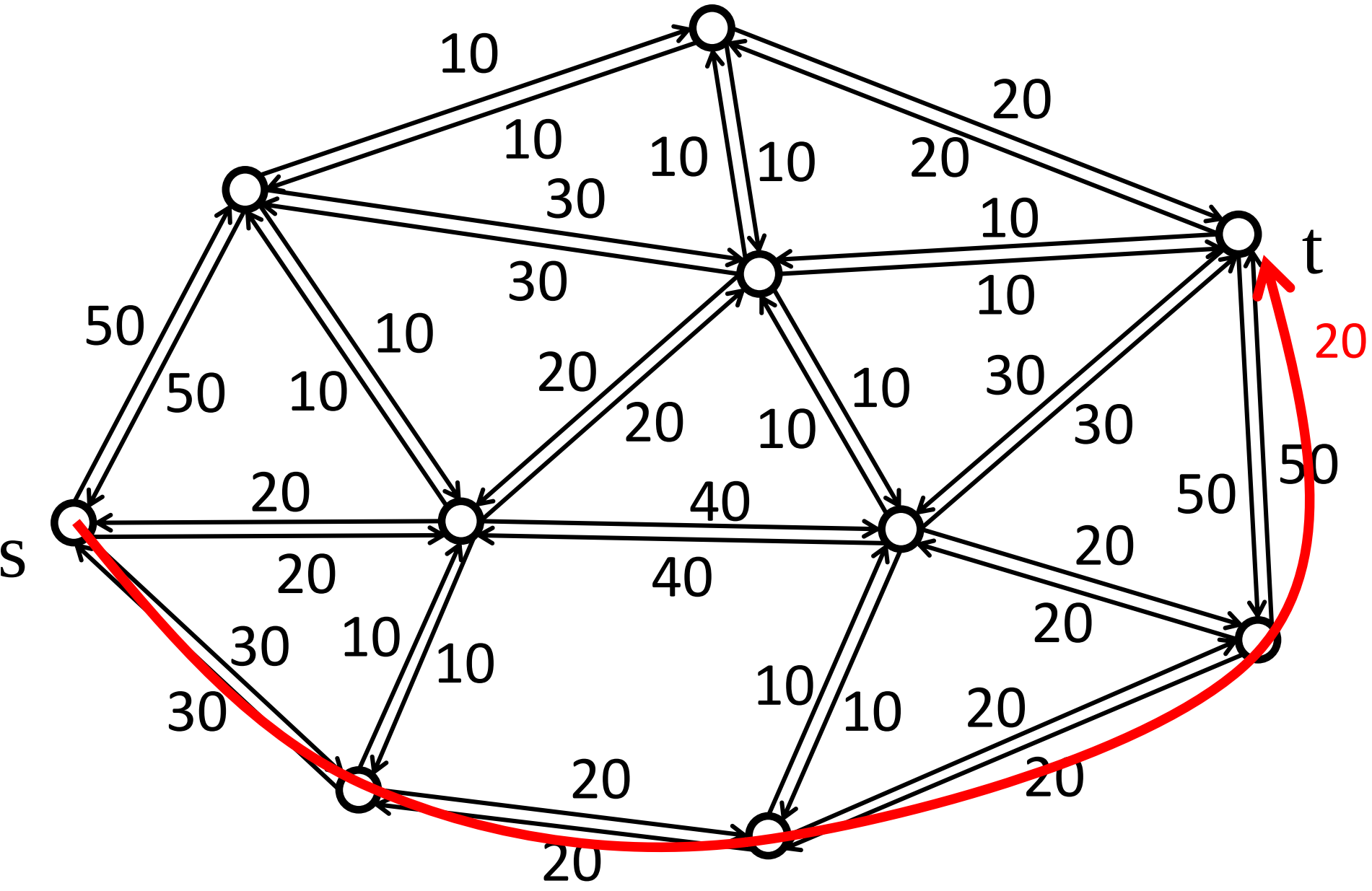



(*) 辺のラベルは、
容量（向きは問わない）を
表している。

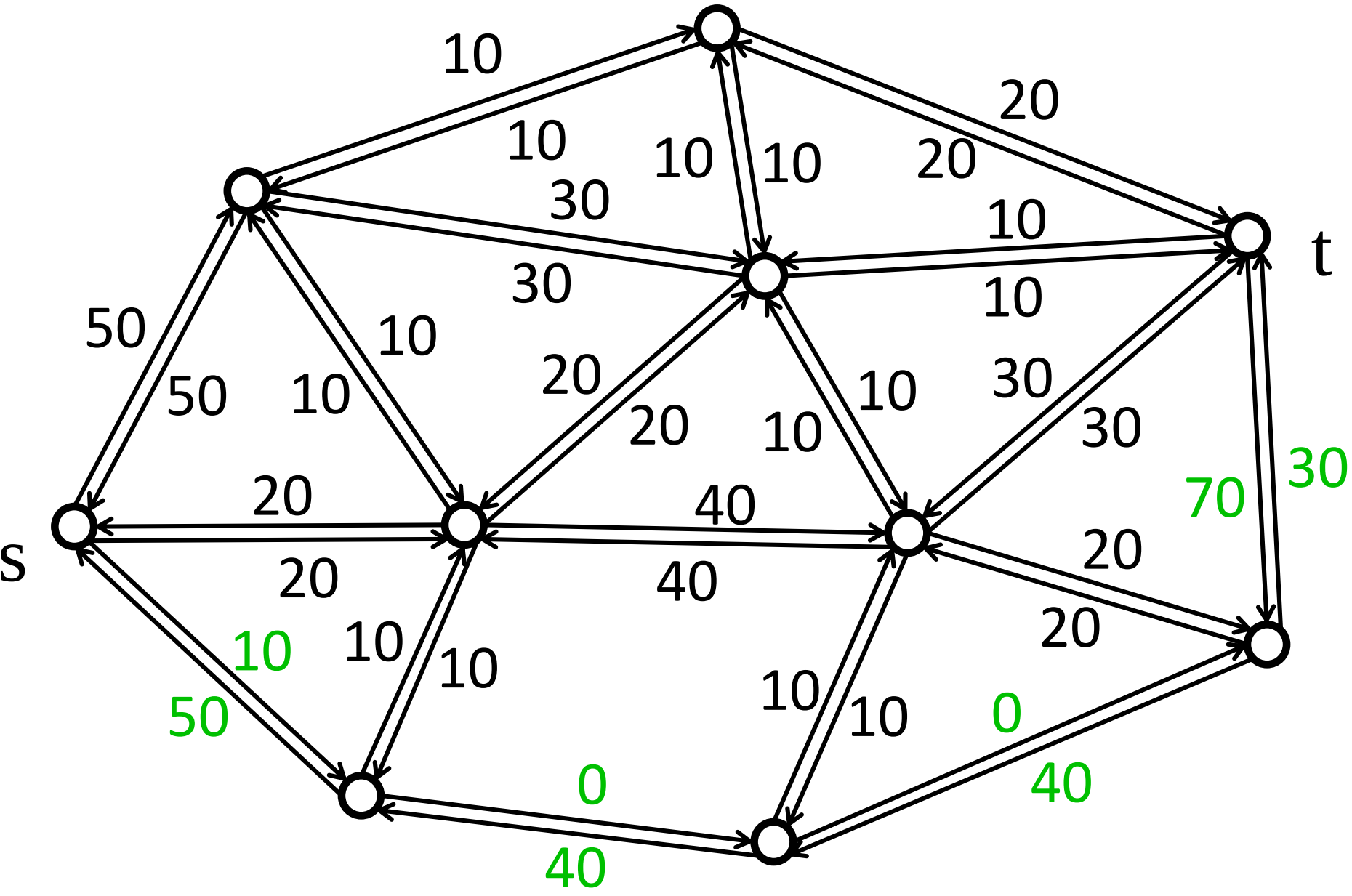
容量を、まず有向グラフに置き換える



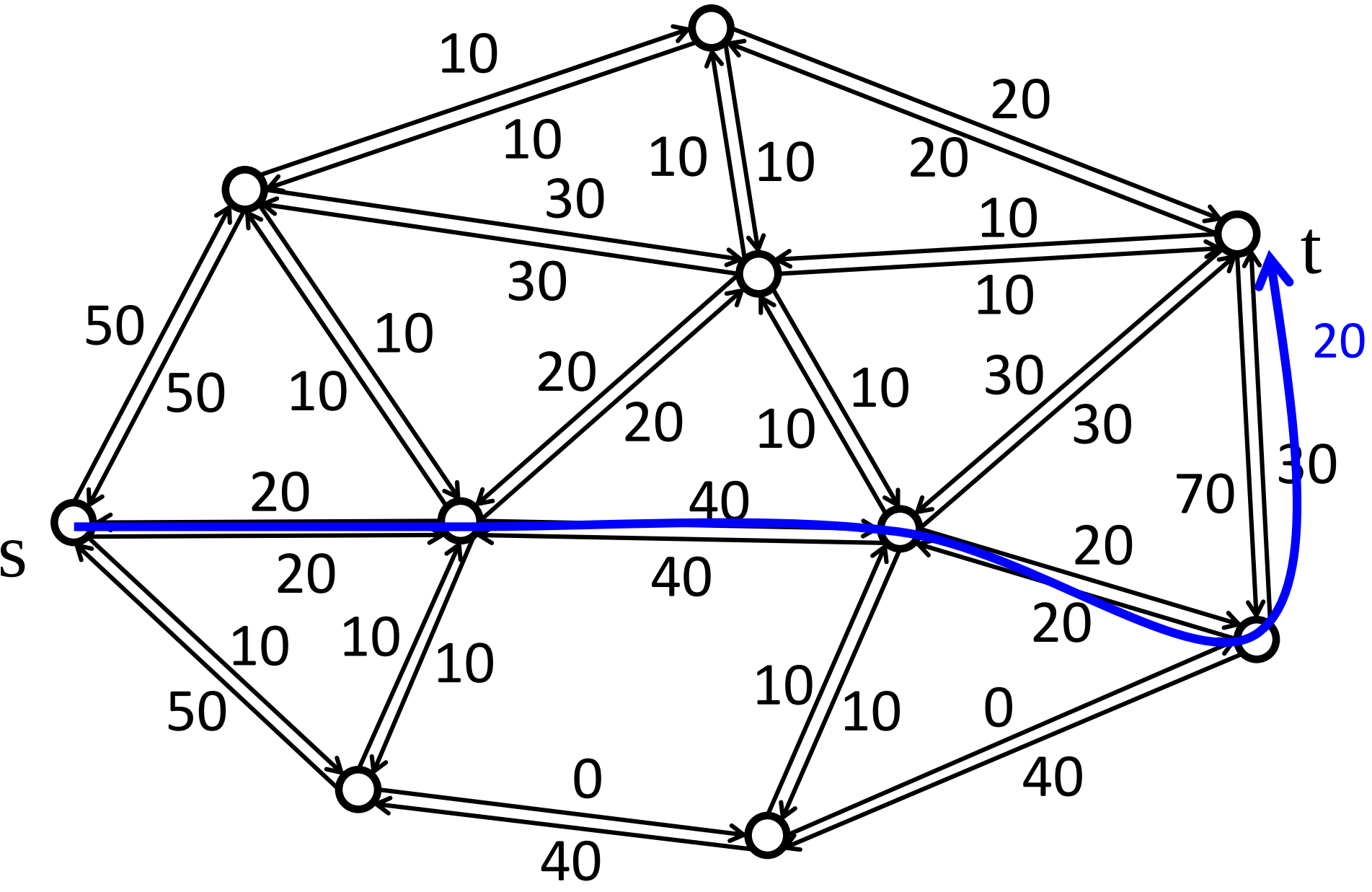
s-t間の適当な道を考え、その流量の上限を算出する。



残余容量 Cf(u,v) = C(u,v) - f(u,v) とし、Gfを作成する

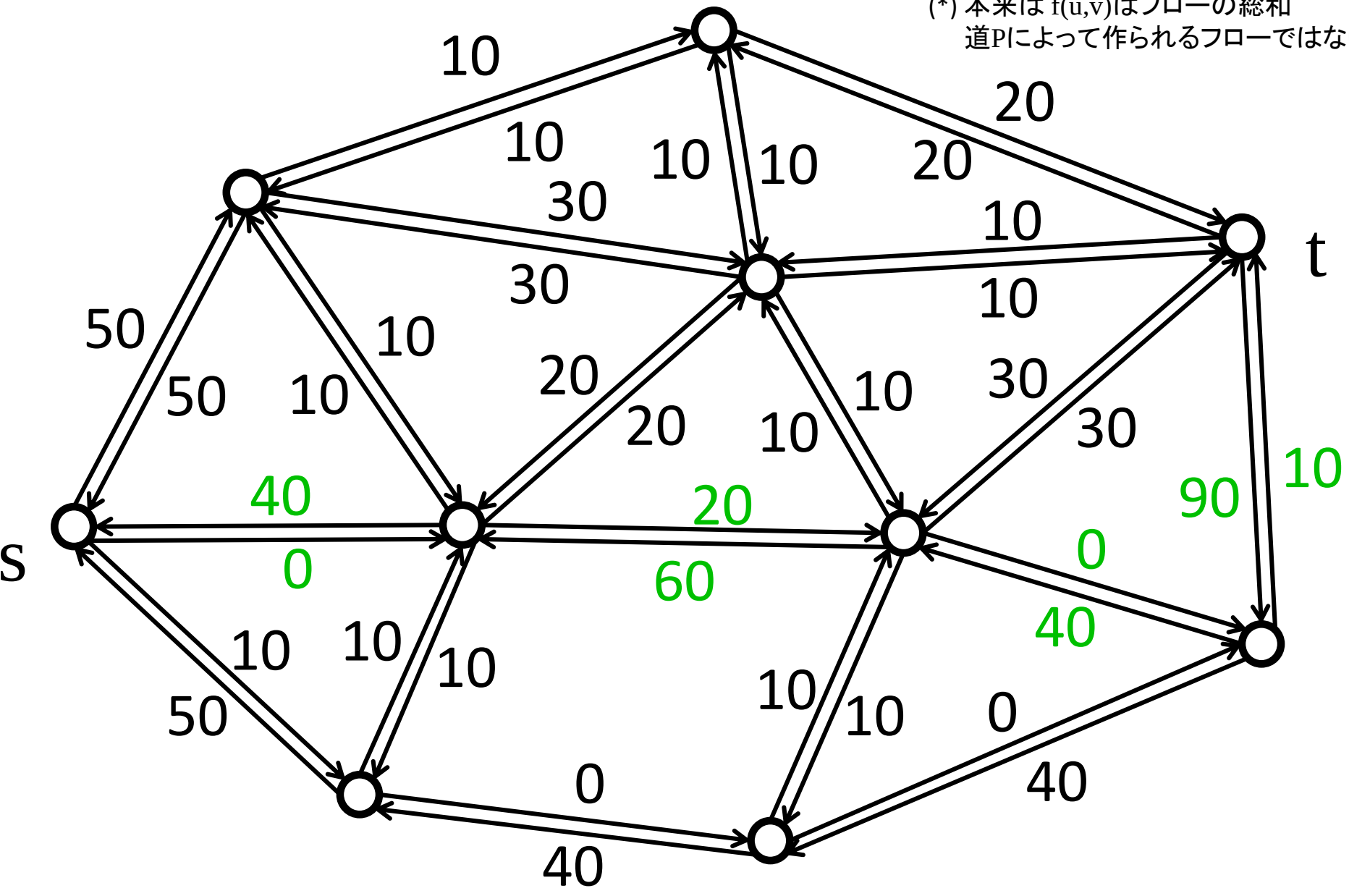


Gfにおいて、s-t間の適当な道を考え、その流量の上限を算出する。

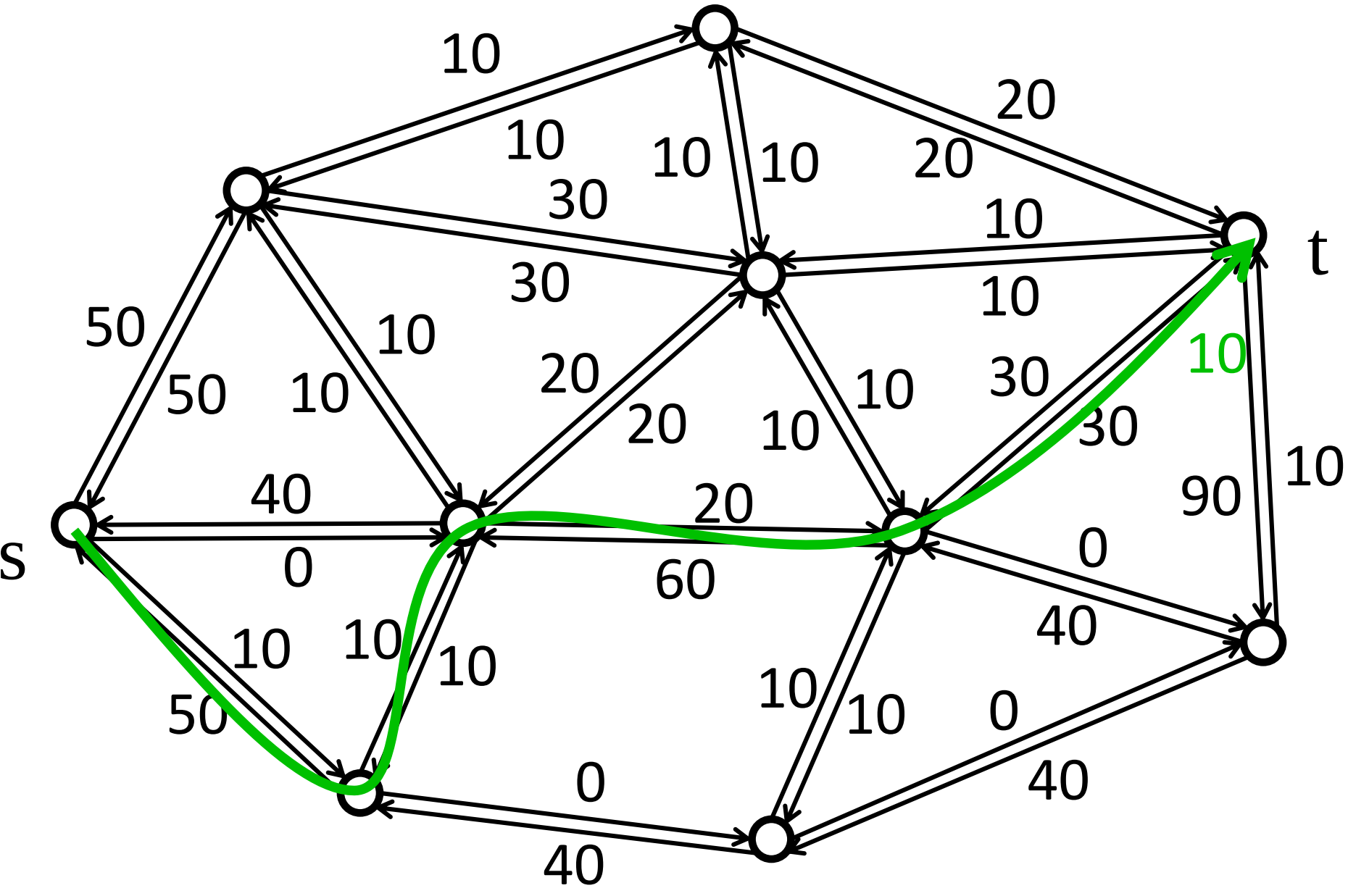


# 残余容量 Cf(u,v) = C(u,v) - f(u,v) とし、Gfを作成する

(*) 本来は f(u,v)はフローの総和
道Pによって作られるフローではない

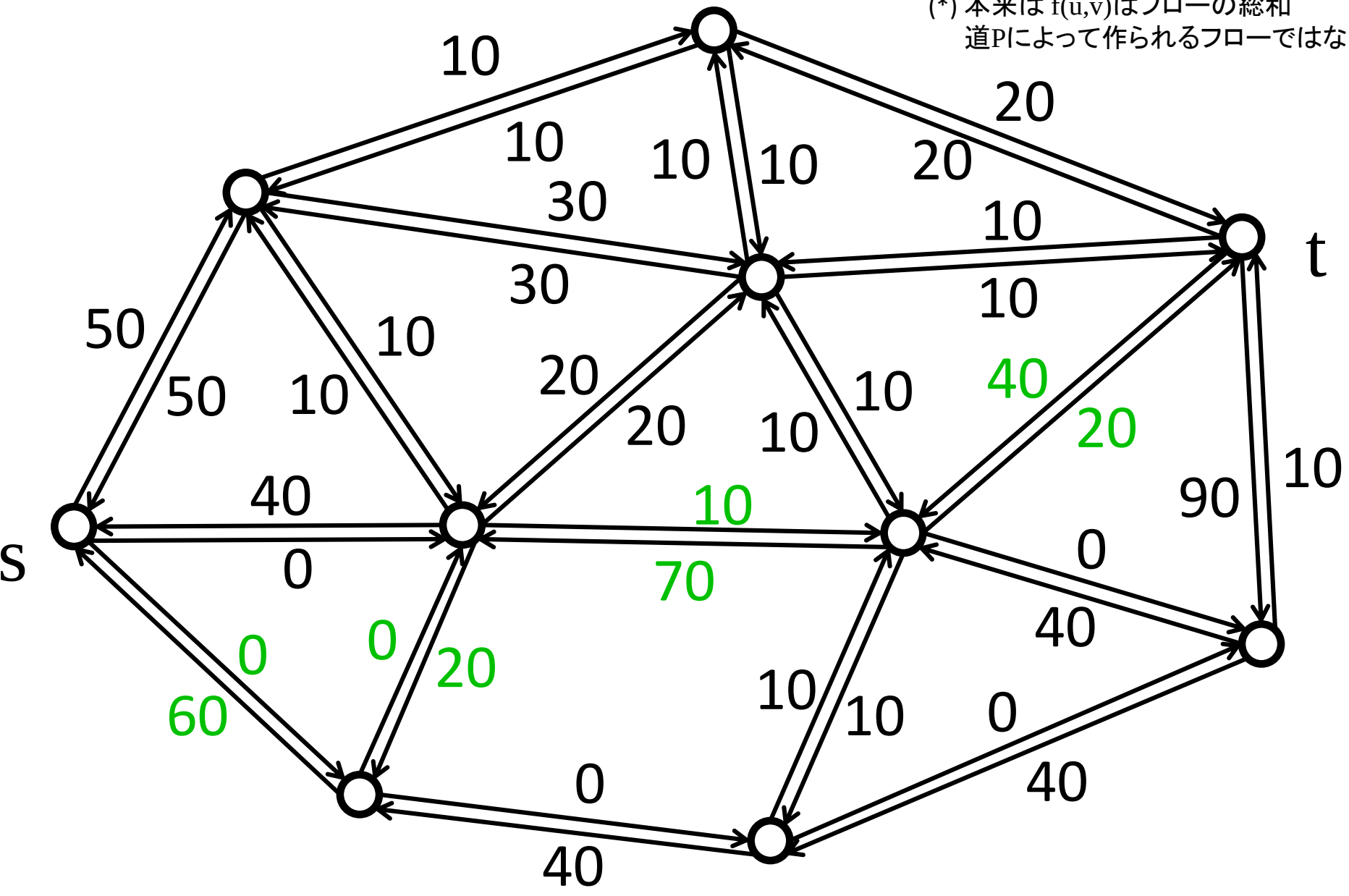


Gfにおいて、s-t間の適当な道を考え、その流量の上限を算出する。



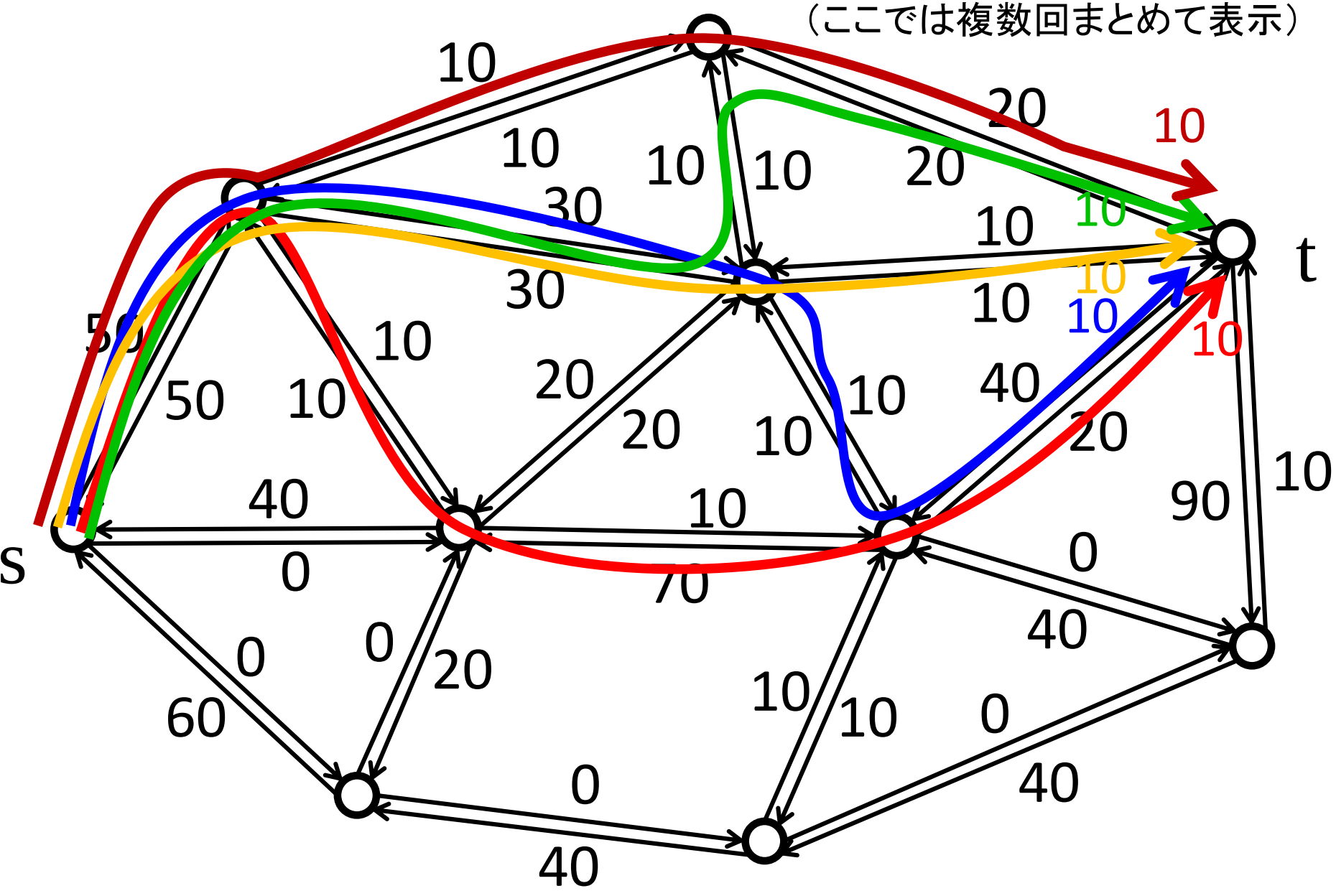
# 残余容量 Cf(u,v) = C(u,v) - f(u,v) とし、Gfを作成する

(*) 本来は f(u,v)はフローの総和
道Pによって作られるフローではない



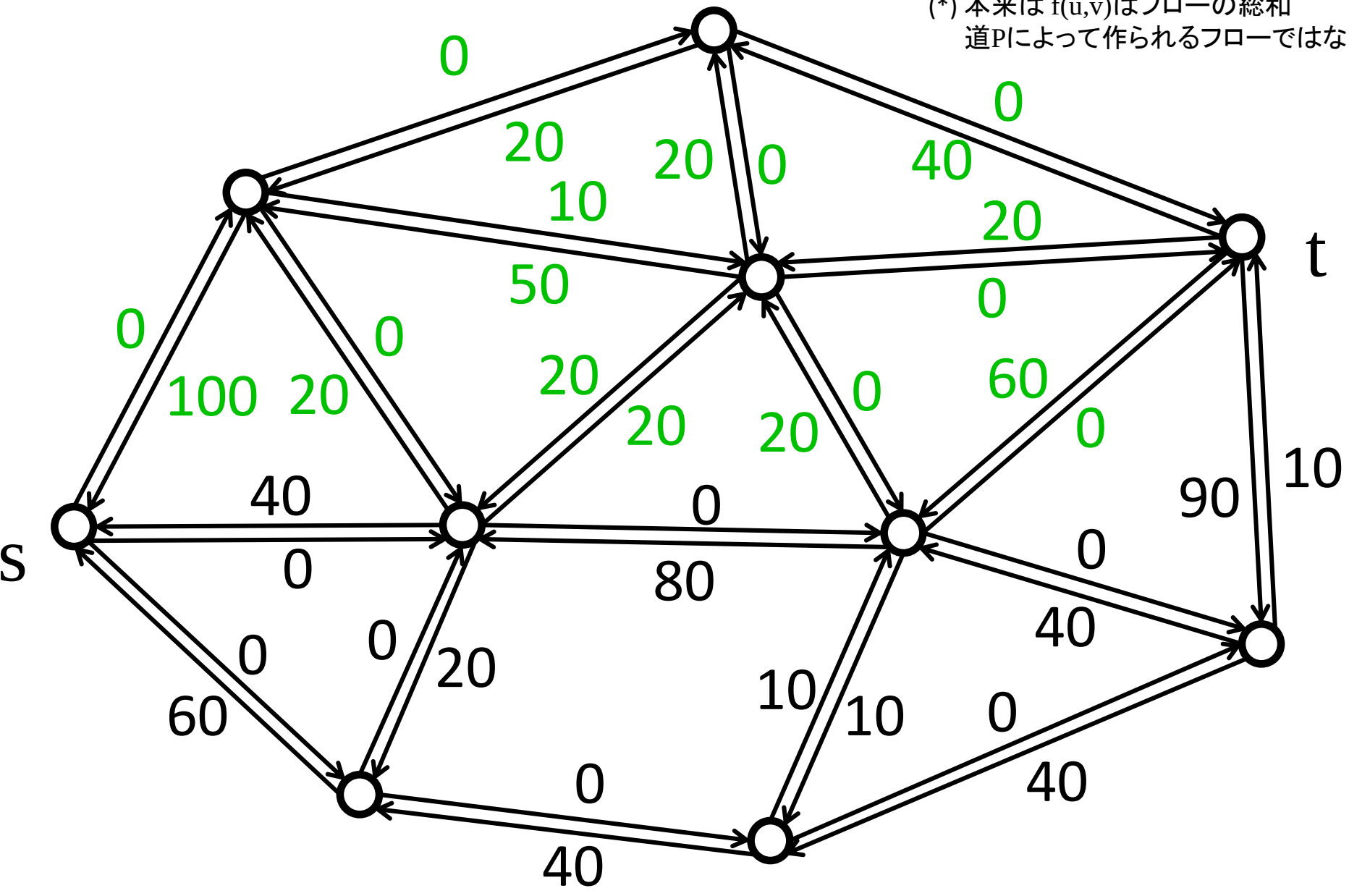
Gfにおいて、s-t間の適当な道を考え、その流量の上限を算出する。

（ここでは複数回まとめて表示）

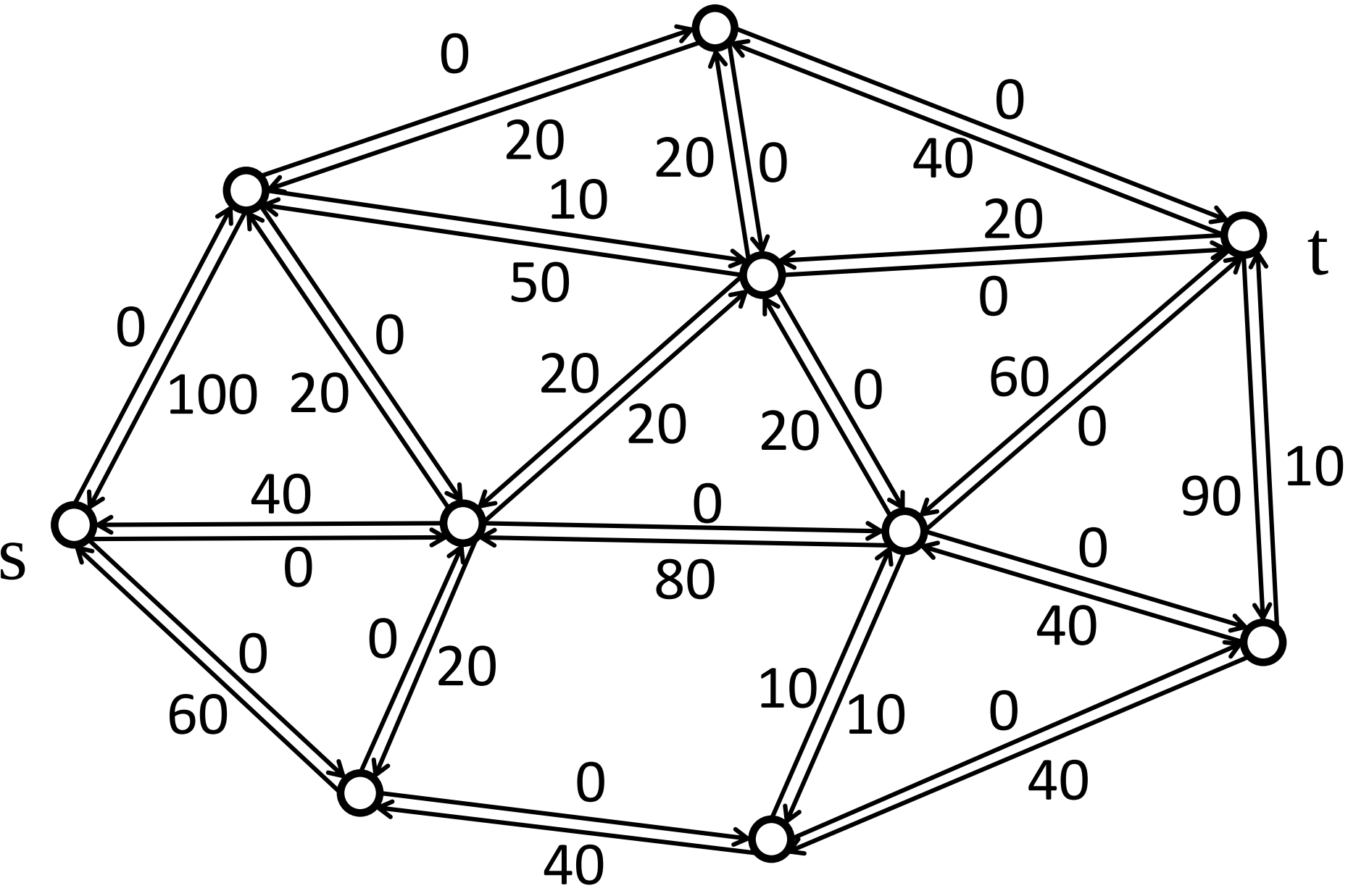


# 残余容量 Cf(u,v) = C(u,v) - f(u,v) とし、Gfを作成する

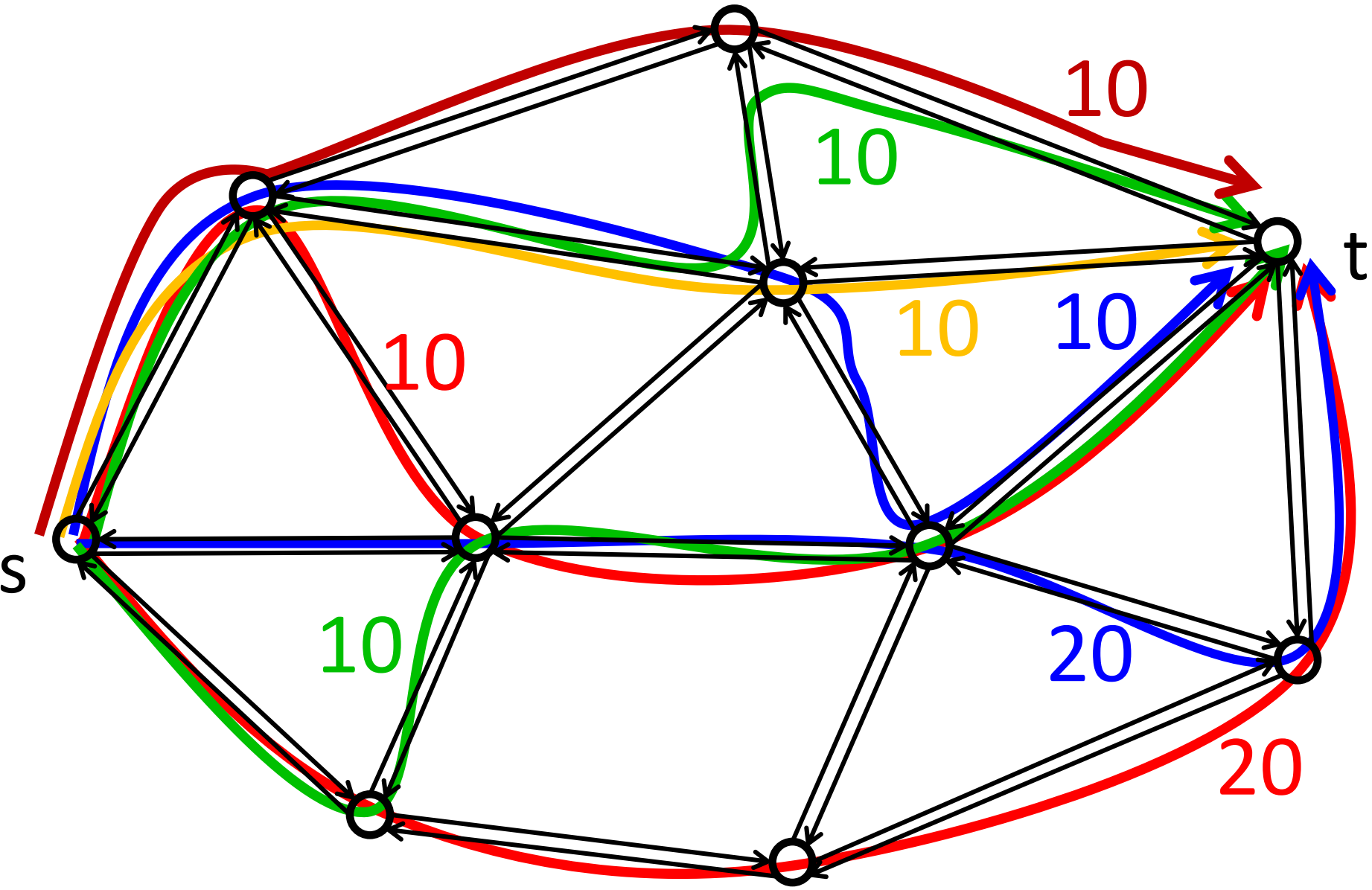
(*) 本来は f(u,v)はフローの総和
道Pによって作られるフローではない



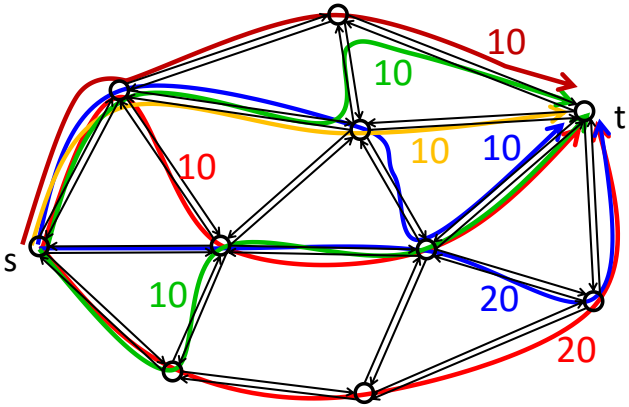
Gfにおいて、s-t間の(Cf>0の辺で作られる)道が存在しないので、終了

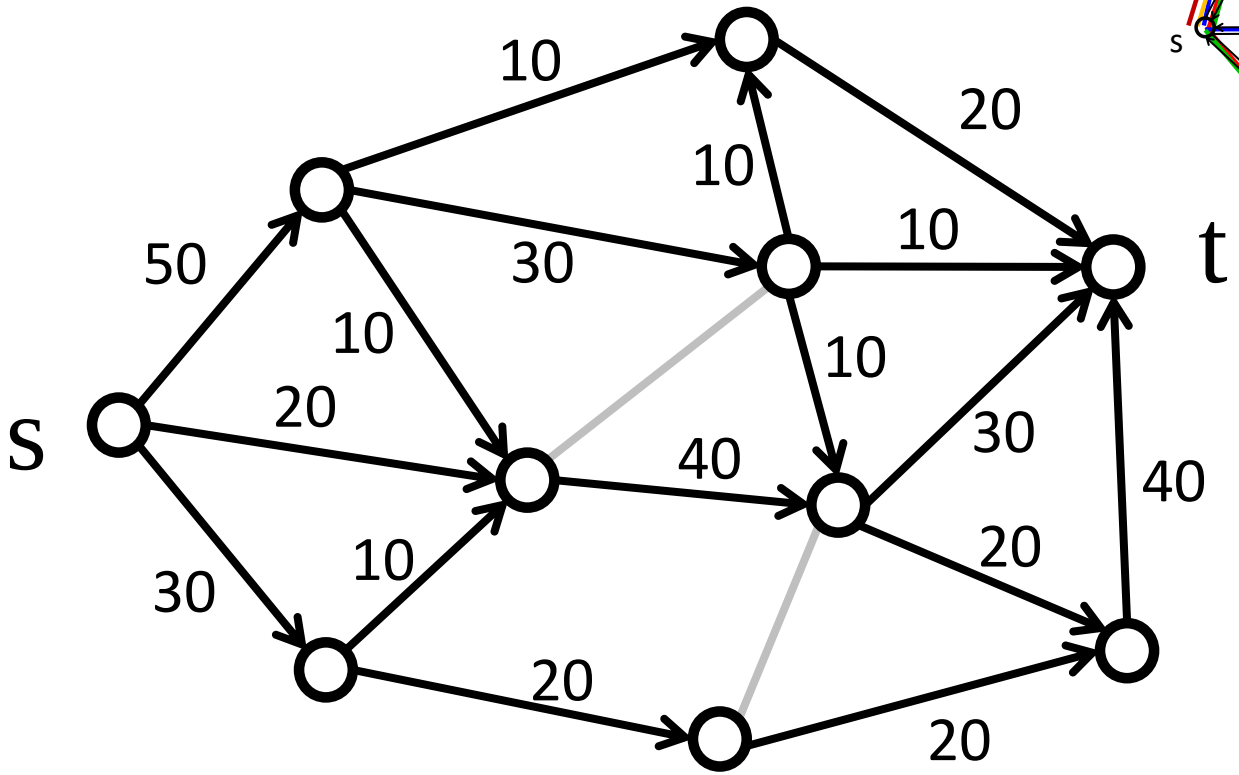


最大流は以下のフローの総和である。



各辺が以下のフローを通すとき、sからtに対して最大流が得られる。その流量は100



# フォード・ファルカーソンのアルゴリズム

```
Max-Flow(G(V,E), s, t)
  ∀e∈E, f(e)=0
  While any paths from s to t exist in residual graph Gf
    P := simple-path(s,t) in Gf
    f' := augment(f, P)
    Update f to be f'
    Update Gf to be Gf'
  EndWhile
Return f
```

道の発見には
「幅優先探索」「深さ優先探索」
などが用いられる

フロー f に道Pを流れるフローの上限分を
追加し、それを f' に代入する処理

※ このアルゴリズムの停止性について考察してみよ

# 今日の内容

- 最小全域木
  - プリムのアルゴリズム

- 最大流問題
  - フォード・ファルカーソンのアルゴリズム

# 今後の予定

- 6月24日： 平面的グラフ、地図
- 7月1日： スキップグラフ
- 7月8日： 内容未定
- 7月22日： 試験 13:00 – 14:30 @241講義室 (未確定)