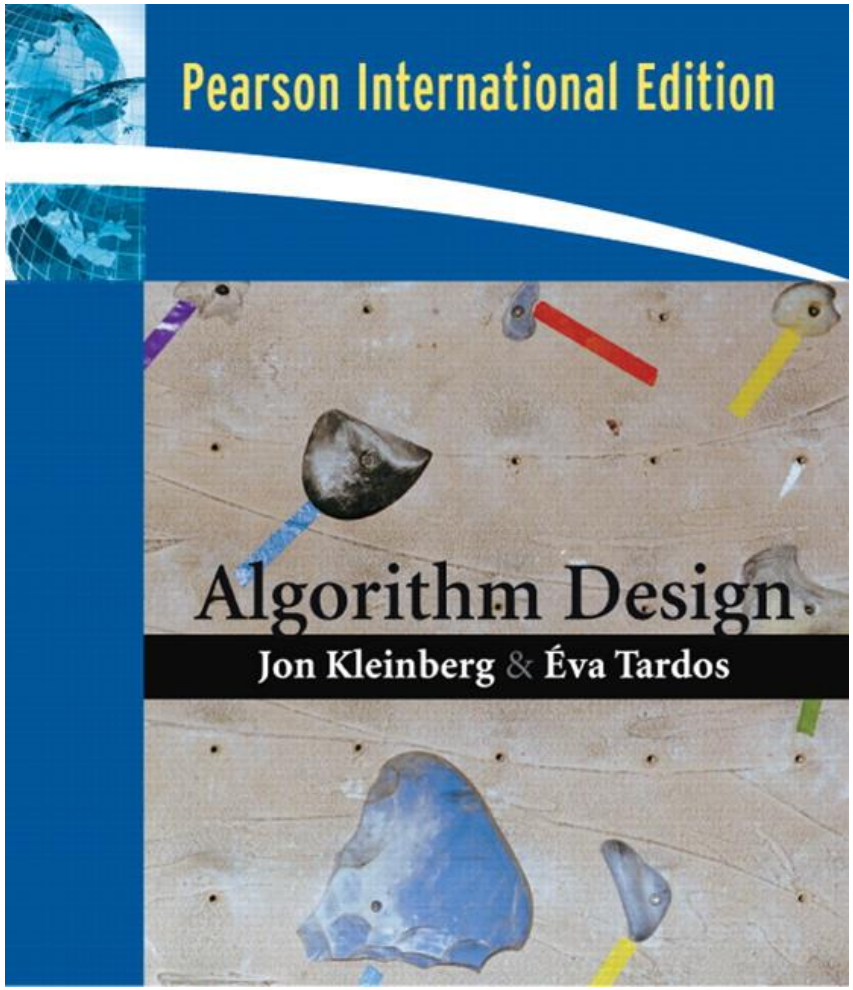
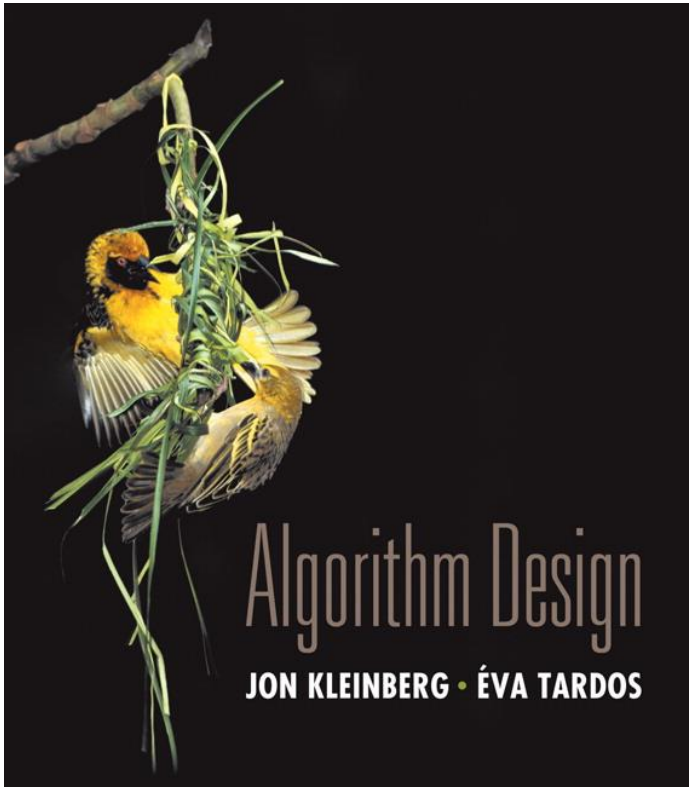
離散数学

アルゴリズム設計
Algorithm Design

落合 秀也

著書 Algorithm Design を読み解く





章構成 (1/2)

1. Introduction: Some Representative Problems
2. Basics of Algorithm Analysis
3. Graphs
4. Greedy Algorithm
5. Divide and Conquer
6. Dynamic Programming
7. Network Flow
8. NP and Computational Intractability

章構成 (2/2)

9. PSPACE: A Class of Problems beyond NP
10. Extending the Limits of Tractability
11. Approximation Algorithm
12. Local Search
13. Randomized Algorithms

Epilogue: Algorithms That Run Forever

本日の進め方

1. これまでの講義で扱ってきた部分をピックアップ
  そして、
    どのような位置づけになっているか
    どのような記述になっているか
  を確かめる

2. 関連する重要キーワードをピックアップ
  そして、その意味について、考える

講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

講義で扱った内容はどこで触れられているか

3 Graph

3.1 Basic Definitions and Applications

3.2 Graph Connectivity and Graph Traversal

3.3 Implementing Graph Traversal Using Queues and Stacks

3.6 Directed Acyclic Graphs and Topological Ordering

4 Greedy Algorithm

4.4 Shortest Paths in a Graph

4.5 The Minimum Spanning Tree Problem

6 Dynamic Programming

6.8 Shortest Paths in a Graph

6.9 Shortest Paths and Distance Vector Protocols

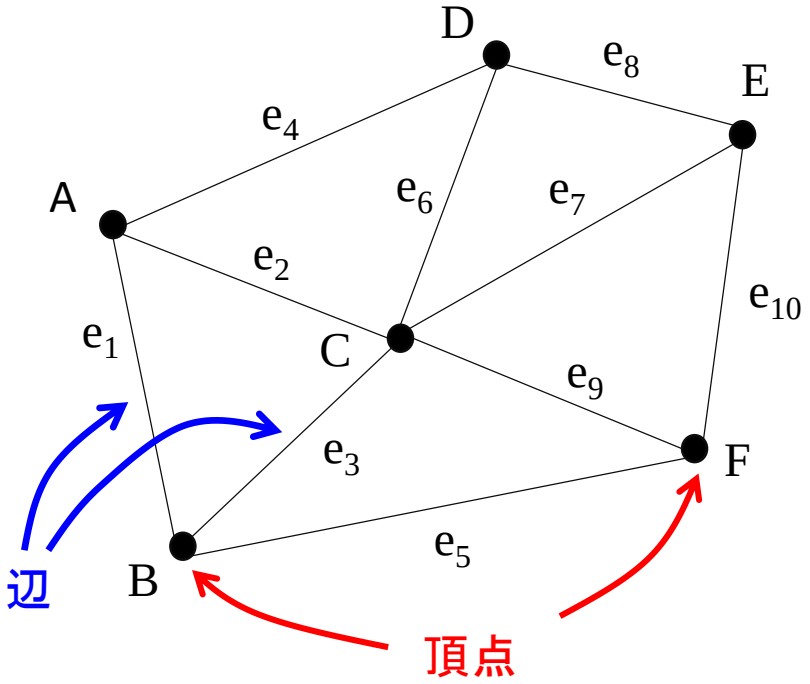
7 Network Flow

7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

グラフ(Graph)を数学的に捉える

- グラフGは、二つの集合で構成されている
  - 頂点(vertex)の集合V
    - 点(point)、節点(node)とも言う
  - 辺(edge)の集合E

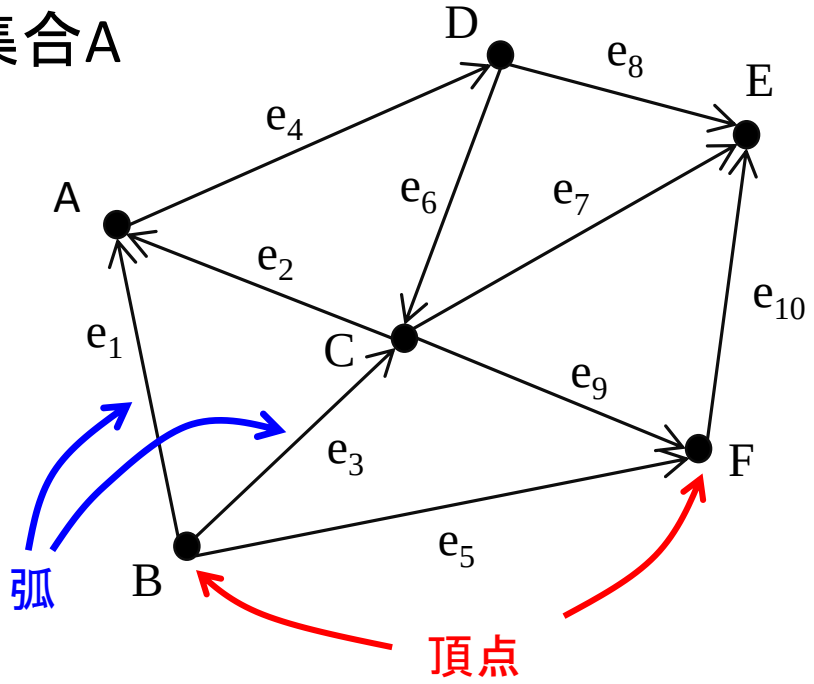
- G (V, E) と書く
  - V={A, B, C, D, E, F}
  - E={ e1, e2, …, e10 }
  - e1={A,B}, e2={A,C}, …



有向グラフ (directed graph, digraph)

- 有向グラフDは、二つの集合で構成されている
  - 頂点(vertex)の集合V
    - 点(point)、節点(node)とも言う
  - 弧(arc): 頂点の順序対の集合A

- D (V, A) と書く
  - V={A, B, C, D, E, F}
  - A={ e1, e2, …, e10 }
  - e1=<B,A>, e2=<C,A>, …



向きの無いグラフを、無向グラフと呼ぶこともある

グラフによる実世界のモデル化

- 交通網
  - 頂点：都市や駅
  - 辺　：道路や路線
- コンピュータ・ネットワーク
  - 頂点：端末やスイッチ
  - 辺　：ケーブル
- WWW
  - 頂点：ページ
  - 辺　：リンク
- ディジタル回路
  - 頂点：論理素子
  - 辺　：配線
- 分子構造
  - 頂点：原子
  - 辺　：結合
- 依存関係
  - 頂点：事象
  - 辺　：原因、結果の関係
- Social Network
  - 頂点：人間，
  - 辺　：関係（知人，etc.）

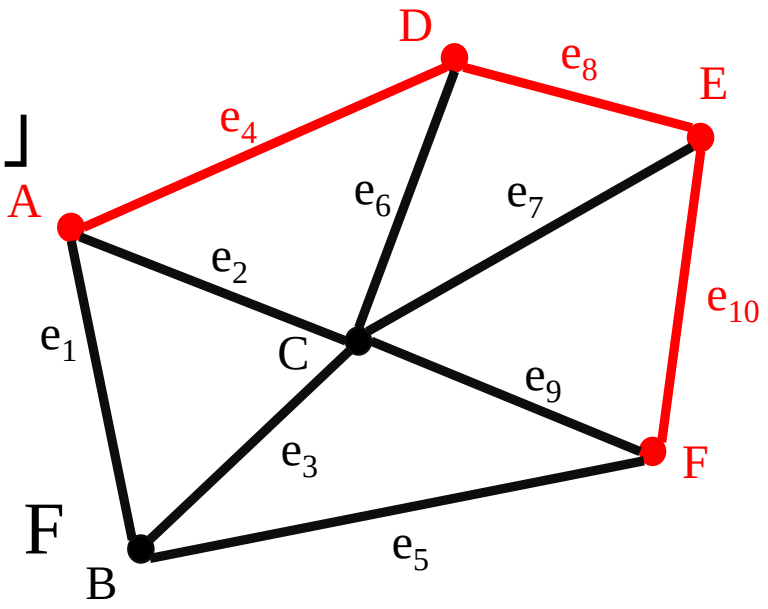
道 (path)

- 頂点-辺-頂点-辺-頂点-…-頂点 (ただし、同じ頂点を通らないもの) を、道(パス: path)と呼ぶ
- モデル上の道を考察する上で重要な概念

- 道は
  「辺の列」または「頂点の列」
  で表現可能

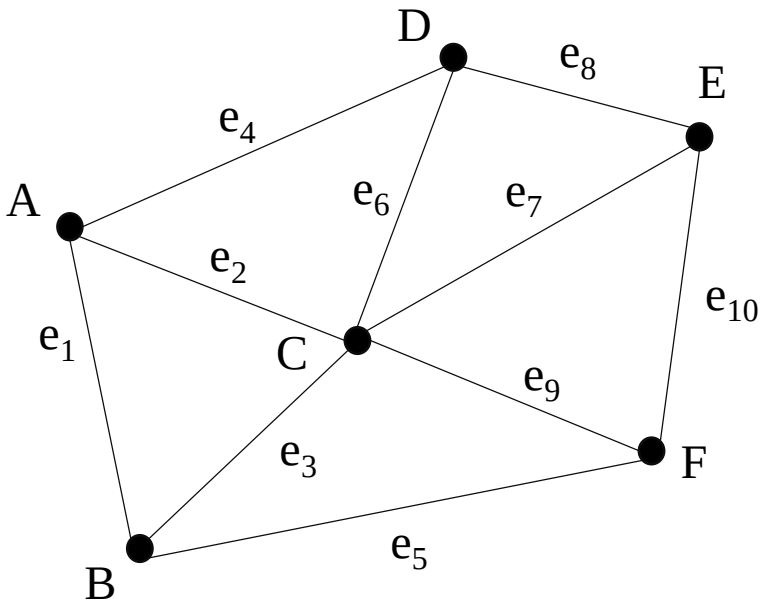
(例) 右図の場合 :

e4, e8, e10 または A, D, E, F

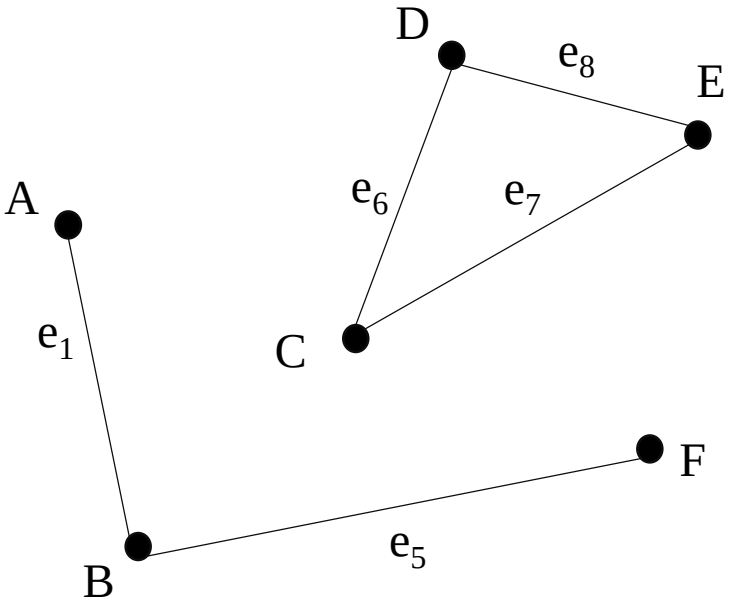


連結 (connectivity)

- グラフGにおいて、任意の2頂点間に道が存在すれば、グラフGは連結である、という。



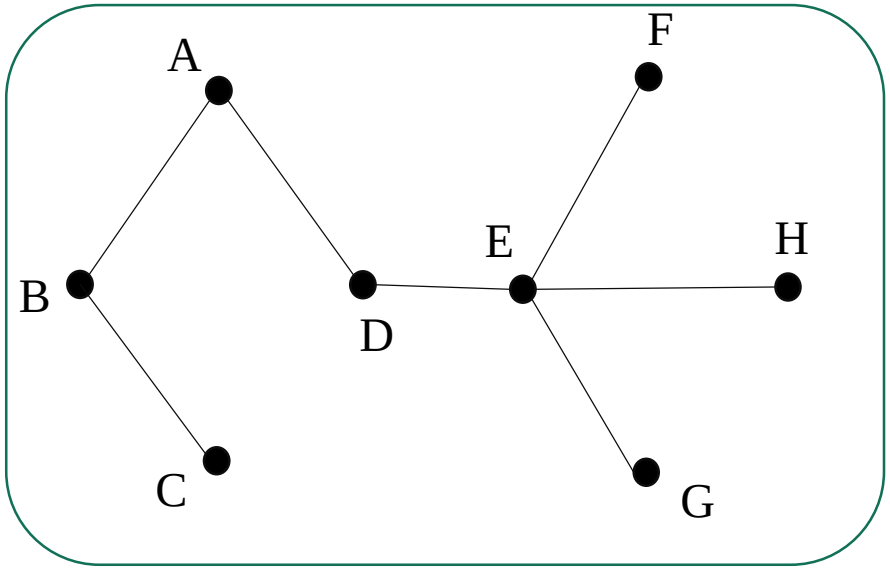
連結なグラフ



連結でないグラフ

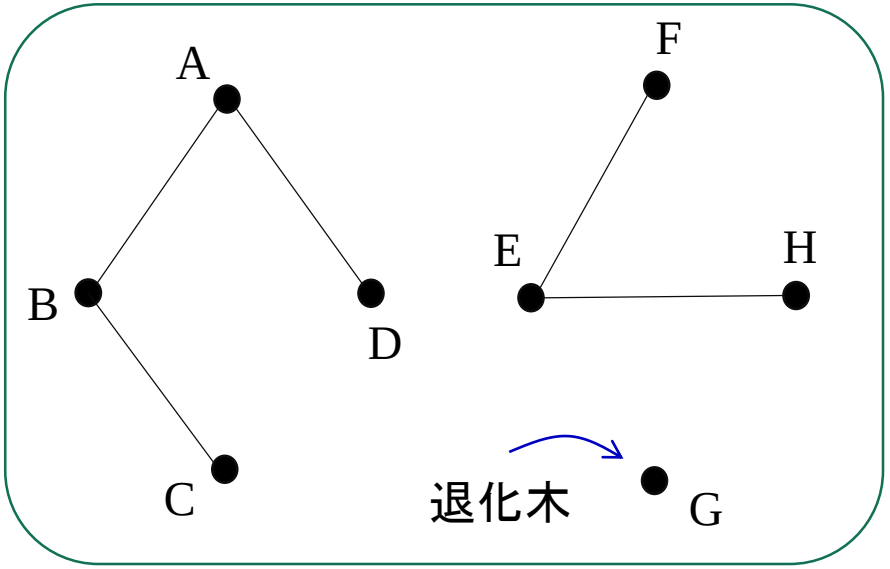
無閉路(acyclic, cycle-free)グラフ

- 閉路のないグラフ



連結グラフ

木 (tree)



連結でないグラフ

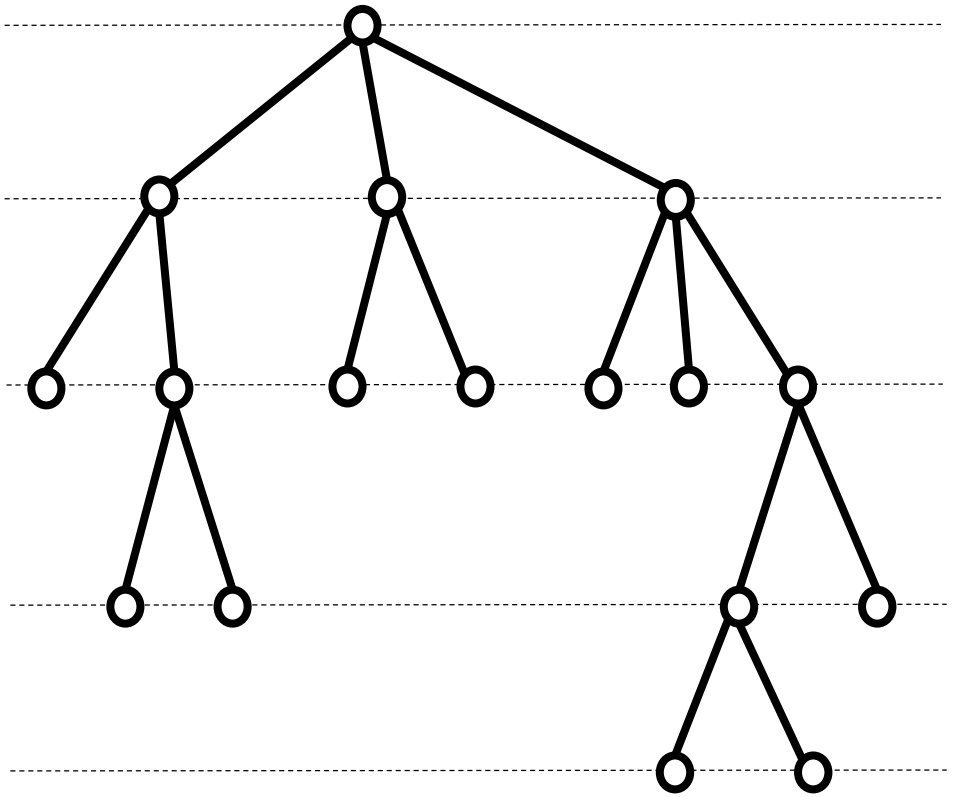
森 (forest)

木の構造

- 根(root)
- 枝(branch)
- 葉(leaf)

- 親(parent)
- 子(child)
- 先祖(ancestor)
- 子孫(descendant)

- 深さ(depth, level)



講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

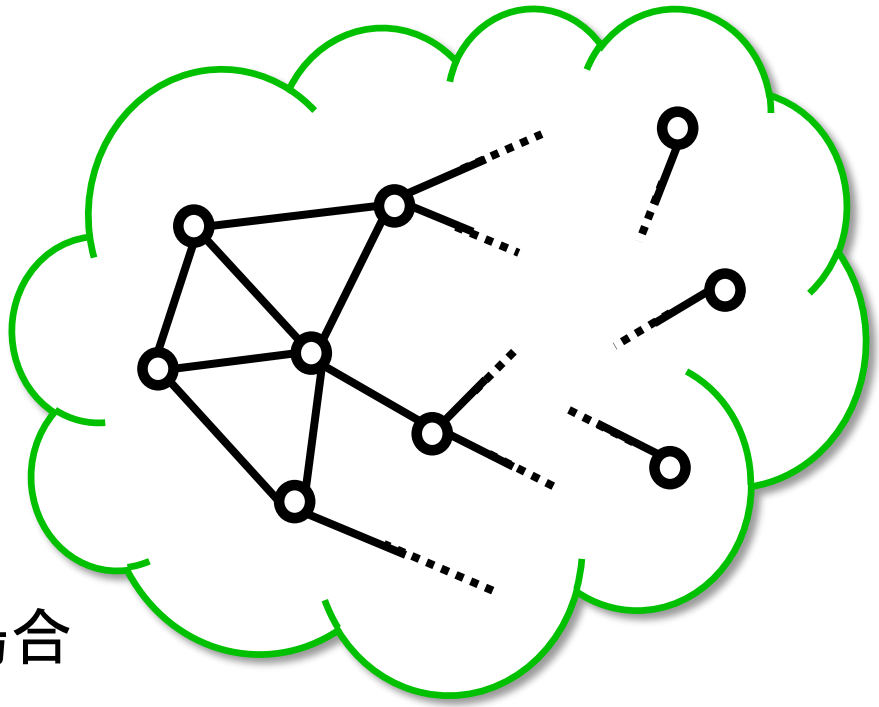
7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

連結性の判定問題を考える

- グラフG(V,E)が与えられたとき、
  Gが連結かどうか、を判定したい。

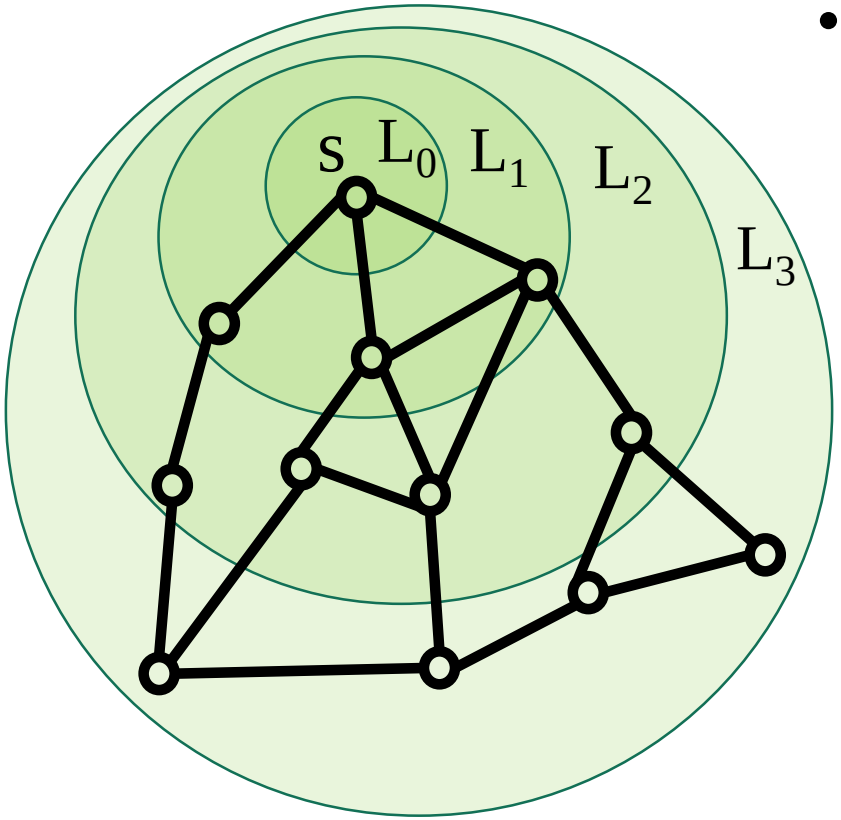
- 小さいグラフなら、
  紙に書いてみればよい

- 一般には簡単ではない
  - 大きいグラフの場合
  - コンピュータに判断させる場合



グラフG

幅優先探索(Breadth-First Search)

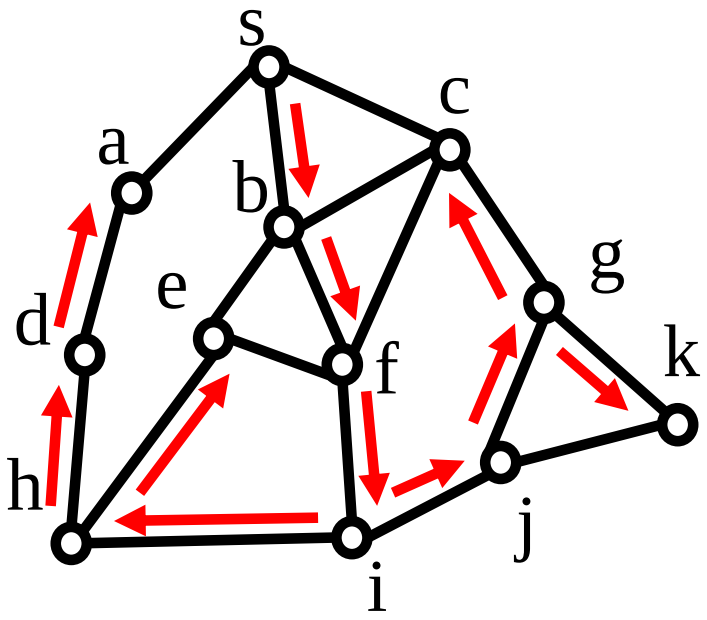


- 基本的な考え方
  - 頂点sから開始・・・L_0とする
  - L_0から、外向きに辺をつたい、接続する頂点を取り込む ・・・L_1とする
  - L_1から、外向きに辺をつたい、接続する頂点を取り込む・・・L_2とする
  - L_2から、外向きに辺をつたい、接続する頂点を取り込む・・・L_3とする

    ・・・

頂点sから湧き出す水によって引き起こされる洪水（Flood）が到達可能な頂点すべてに伝搬するイメージ

深さ優先探索 (Depth-First Search)

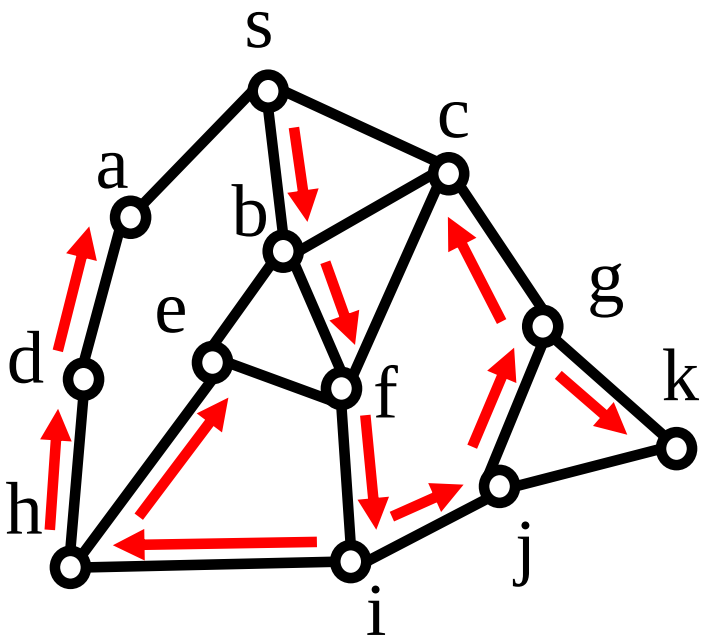


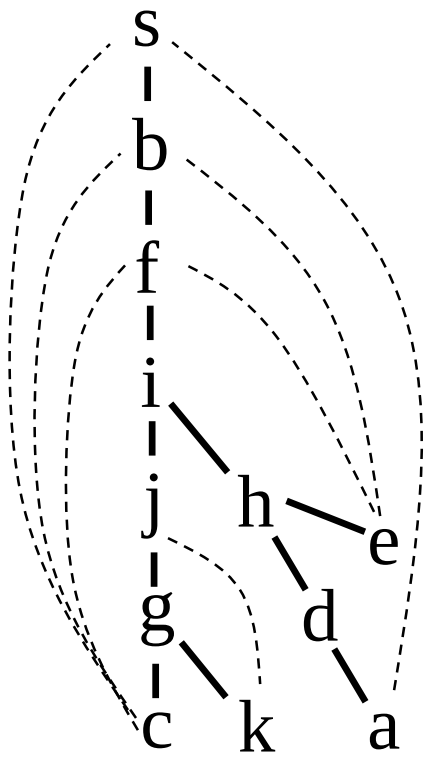
- 基本的な考え方
  - 開始点sから、辺をたどって行きつくところ（=dead end）まで行ってみる（ただし同じ頂点は取らない）
  - 行き止まり(=dead end)に当たれば、1つ戻って、別の辺をたどってみる
  - 1つ戻ってもダメなら、2つ戻ってみる
  - 2つ戻ってもダメなら、3つ戻ってみる
  - …

頂点sから歩き始めた人が、すべての行き止まりにあたるまで、隈なく歩いていくイメージ

深さ優先探索によって作られる木

- 出来るだけ深いところまで一気に作る
- 戻りながら、別に行ける頂点があれば、そちらの枝を作る





深さ優先探索木と呼ばれる

講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

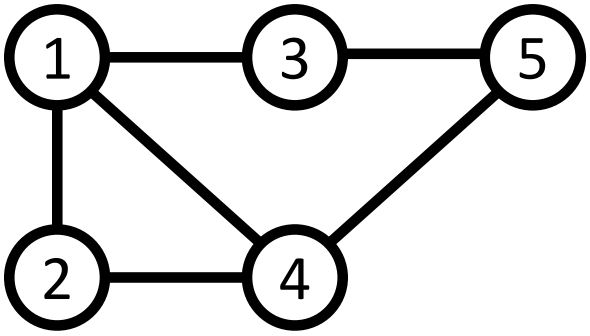
4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

グラフを隣接リストで表現する

- 隣接リスト(adjacency list)
  1. ある頂点vの、近隣の頂点をリストで表現したもの: Adj[v]
  2. 全頂点に対して、同様のリストを作る: Adj – 配列

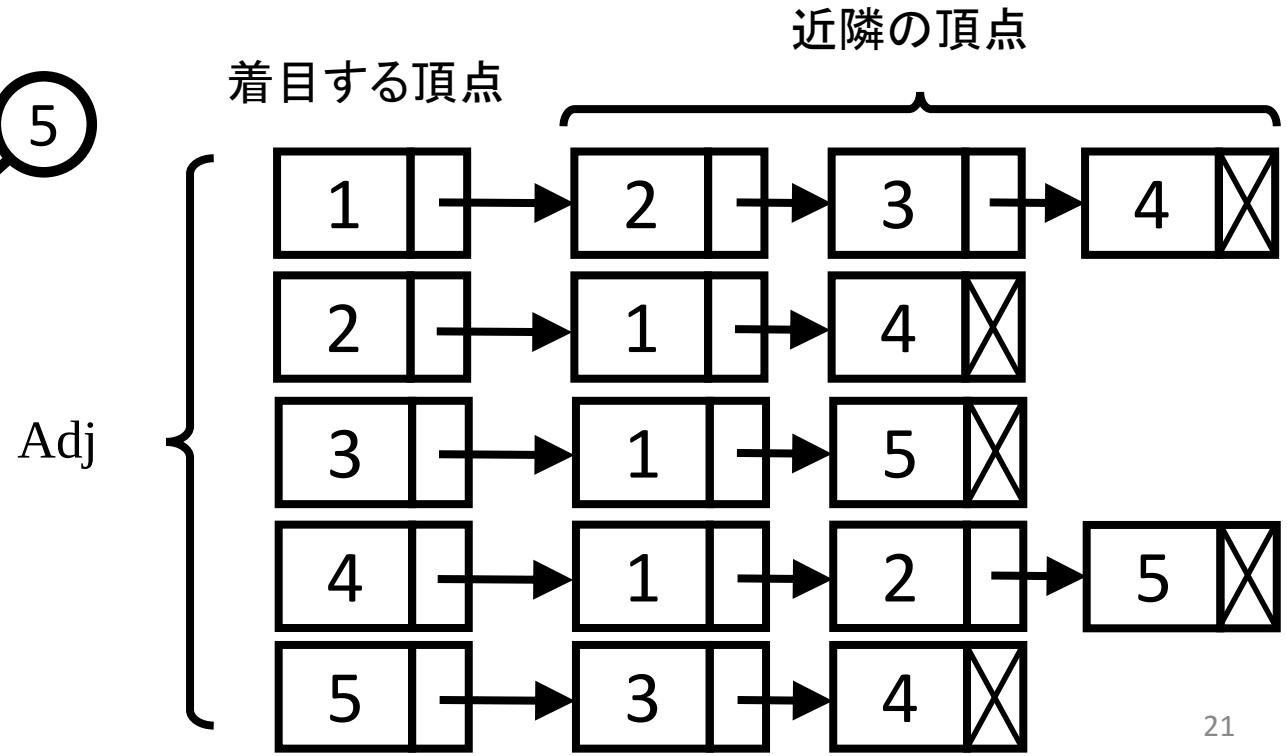


Adjの大きさは

$O(n+m)$

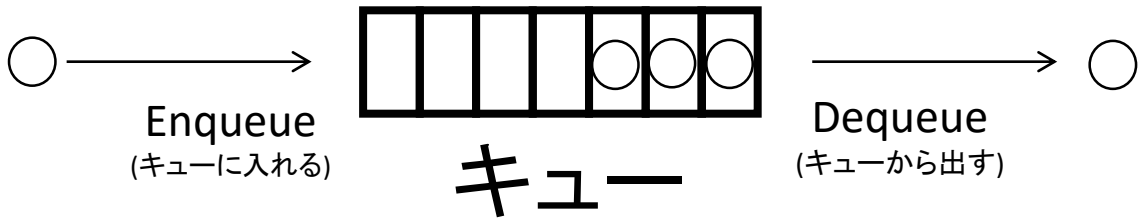
n: 頂点の数

m: 辺の数



キュー(Queue)の働きと使い方

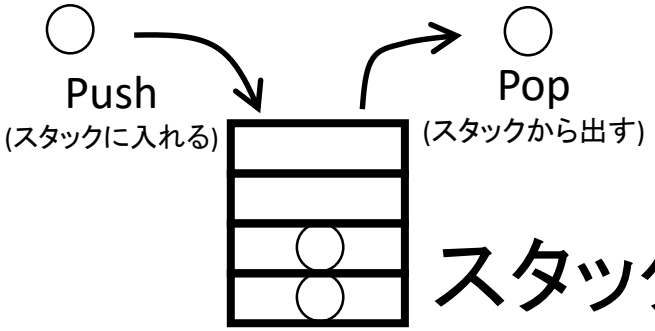
- 先入れ先出し装置: Fast In Fast Out (FIFO)



- 入れた順番に取り出せるメモリ:
  - 3, 1, 2, 8, 6, 5, 4 の順に投入すれば、
    3, 1, 2, 8, 6, 5, 4 の順に出てくる

深さ優先探索を実現する スタック(Stack)を利用する

- 後入れ先出し装置: Last In Fast Out (LIFO)



- 入れた順番の逆に取り出せるメモリ:
  (1) 3, 1, 2を投入
  (2) 2個取り出す
  (3) 8, 6 を投入
  (4) 3個取り出す
  (5) 5, 4 を投入
  (6) 2個取り出す

取り出される列は
2, 1, 6, 8, 3, 4, 5
となる

講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

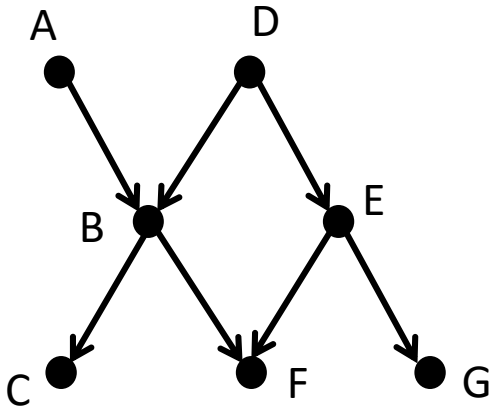
6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

3.6 Directed Acyclic Graph and Topological Ordering

有向非巡回グラフ
Directed Acyclic Graph (DAG)

- 閉路の無い有向グラフ
- 半順序性を持つ



- トポロジカルソートにより、頂点を一列に並べることができる(全順序に対応付けできる)
  - 上図の場合: A, D, B, E, C, F, G など
  - ジョブのスケジューリング等で使われる

講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

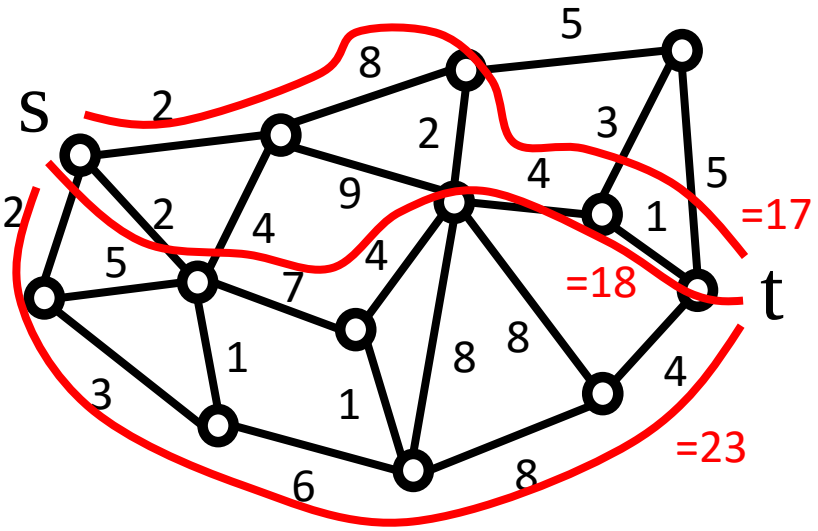
4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

最短経路問題を考える

- ラベル付(重み付)グラフ G=(V,E)が与えられたとき、頂点sから頂点tへの最短経路 s-t を求めたい
- 辺のラベルの意味:
  - 地点間の道のり
  - 地点間の移動時間
  - 地点間の移動に要する燃料の量
  - 状態の遷移で失う資金



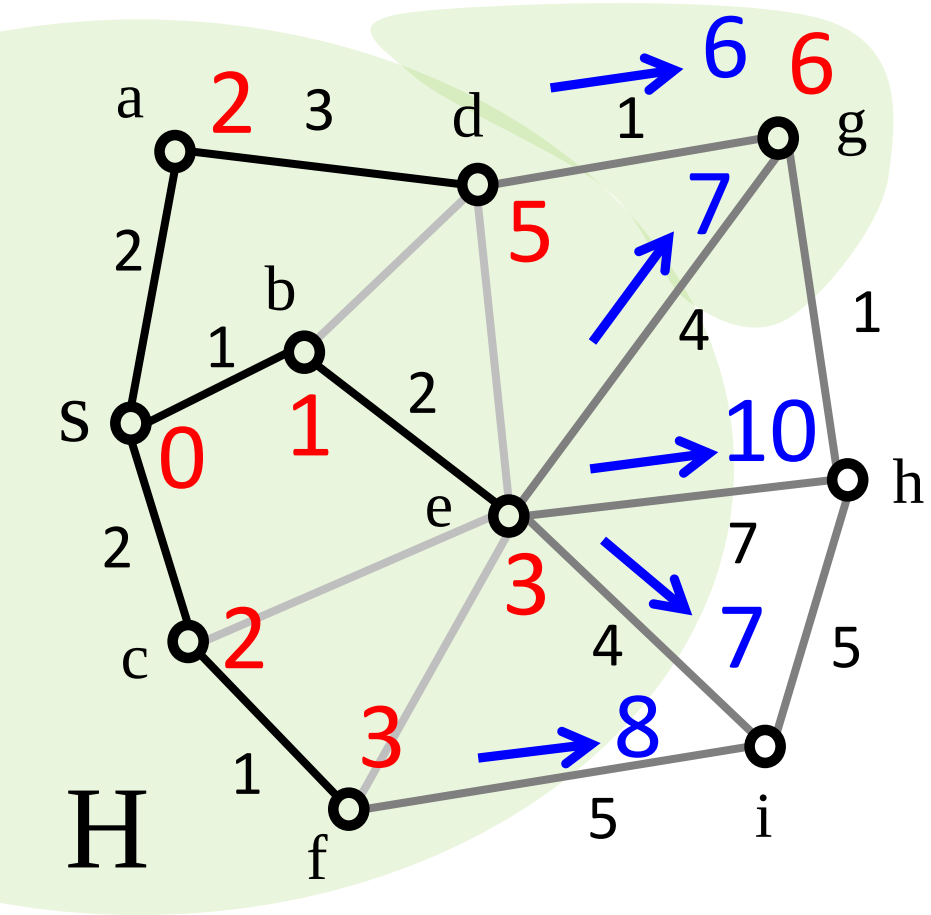
(*) 実際はコスト16の道が存在する

- 最小コストでの移動経路を発見したい

4.4 Shortest Paths in a Graph

ダイクストラ法(Dijkstra's Algorithm)

1959年：Edsger Dijkstraによって考案された



- 考え方 -- G(V,E)に対して
  - 開始点 s∈Vから各頂点への最短経路を求める問題
  - いま、すでに部分Hに関して、最短経路が導出されているとする
    - sからの距離が判明している
  - Hと接する v ∈ V-H に対し、sからの距離を考える
  - その距離の最小値を与えるvを、Hに追加する

赤字: 判明しているsからの最短距離

4.4 Shortest Paths in a Graph

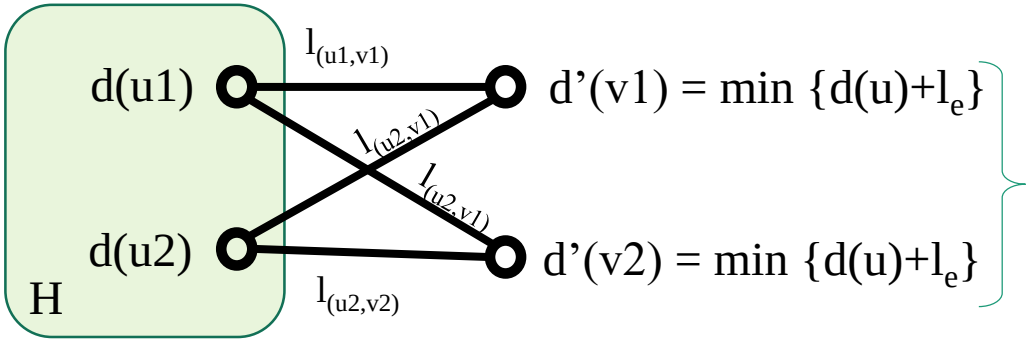
ダイクストラ法(もう少し正確な定義)

- 用語の定義:
  - グラフG(V,E)において、$u, v \in V$, $e=(u,v) \in E$とする。
  - 辺e=(u,v)の長さをl_eあるいは$l_{(u,v)}$で表すとする。
  - 開始点をsとする。d(u)で、sからuまでの最短距離を表すとする。
  - prev(v)で、sからvへの最短経路におけるvの直前の頂点を表す。
- 挙動の定義: Vの部分集合 H を以下のように作成する。
  - 初期状態: $H = \{s\}$, $d(s)=0$, $d(v)=\infty$ $(v \neq s)$, $prev(v)=null$ $(v \in V)$ とする。
  - $u \in H$ で辺e=(u, v)でつながっている $v (\in V-H)$ を考え、

$$d'(v) = \min_{e=(u,v):u \in H} \{ d(u) + l_e \}$$

を計算する。

  - $v \in V-H$において、上記をさらに最小化するvを選定し、そのvをHに追加し、その距離をd(v)とおき、その時のuを用いて、prev(v)=uとする。



4.4 Shortest Paths in a Graph

ダイクストラ法: 計算量の考察

```
While Q is not empty
  u := Q.extractMin()
  Foreach v in Adj[u]
    If d[v] > d[u] + length(u, v) Then,
      d[v] := d[u] + 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)
  - d[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
  - d[v]の更新は、m回発生する → O(m log n)

    ➔ O((n+m) log n)

講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

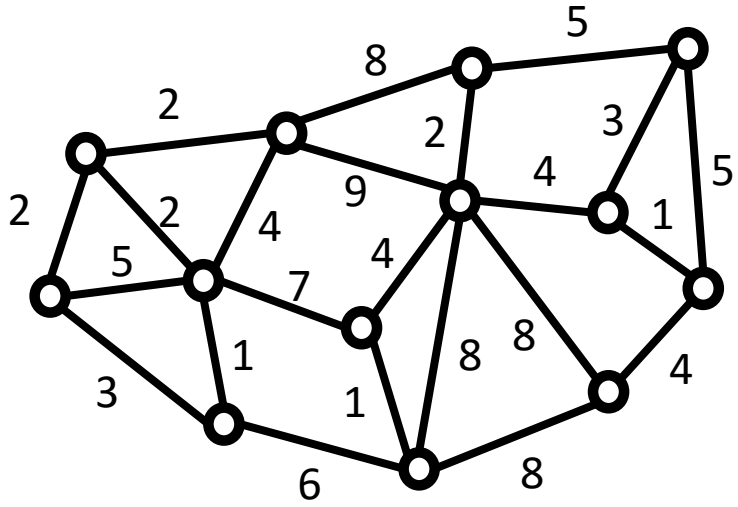
最小全域木を考える
Minimum Spanning Tree Problem

- 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

4.5 The Minimum Spanning Tree Problem

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)

講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

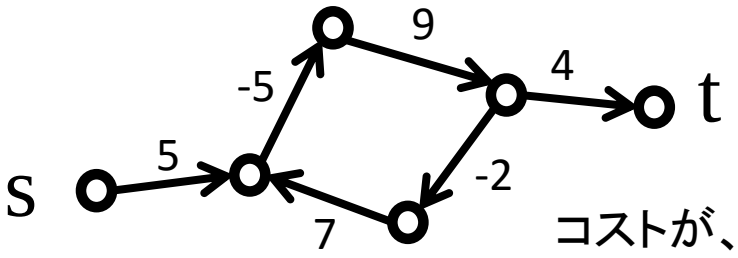
6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

最短経路問題の解法

- ダイクストラ法
  - すべての辺の重みが非負の場合に適用可能
  - 貪欲法(Greedy Algorithm)の一種

- ベルマン・フォード法
  - 辺の重みに負があっても良いケースがある（有向グラフで、有向閉路の和が負でない場合）
  - 動的計画法(Dynamic Programming)の一種



コストが、資金の支出の場合、
マイナスは、収入を意味する

ベルマン・フォード法

- i回目から、i+1回目への計算(漸化式)を考える
  - ここで、iは、拡散量(sから伝搬した辺の数)に相当する
- i回目(i個の辺の伝搬を考えたとき)における、頂点sから頂点vまでの最短距離をOPT(i,v)とする。このとき、以下が言える。

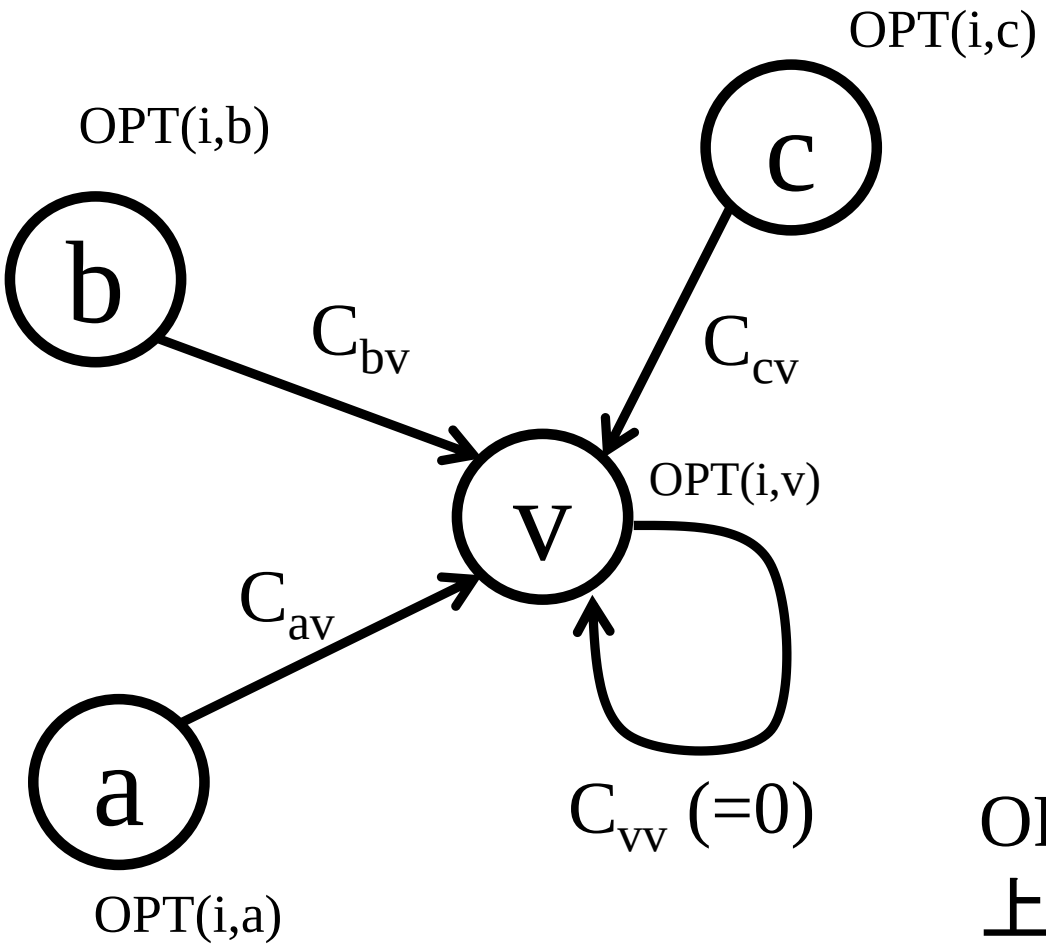
$$OPT(i+1, v) = \min_{u \in nbr(v)} \{OPT(i,u)+C_{uv}\}$$

(*) ここでnbr(v)は、vの近隣頂点(自分自身も含む)の集合とする
またC$_{uv}$は辺(u,v)のコストとし、便宜上C$_{vv}$=0 とする。

- 初期条件 (i=0のとき)
  - OPT(0,s)=0, OPT(0,v)=∞ (v∈V, v≠s)

6.8 Shortest Paths in a Graph

$OPT(i+1, v) = \min_{u \in nbr(v)} \{OPT(i,u)+C_{uv}\}$ の意味



$OPT(i,v)$
$OPT(i,a)+C_{av}$
$OPT(i,b)+C_{bv}$
$OPT(i,c)+C_{cv}$

OPT(i+1,v)は、
上記で最小のものとする

講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

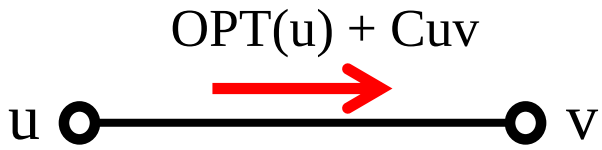
4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

ベルマンフォード法の分散化

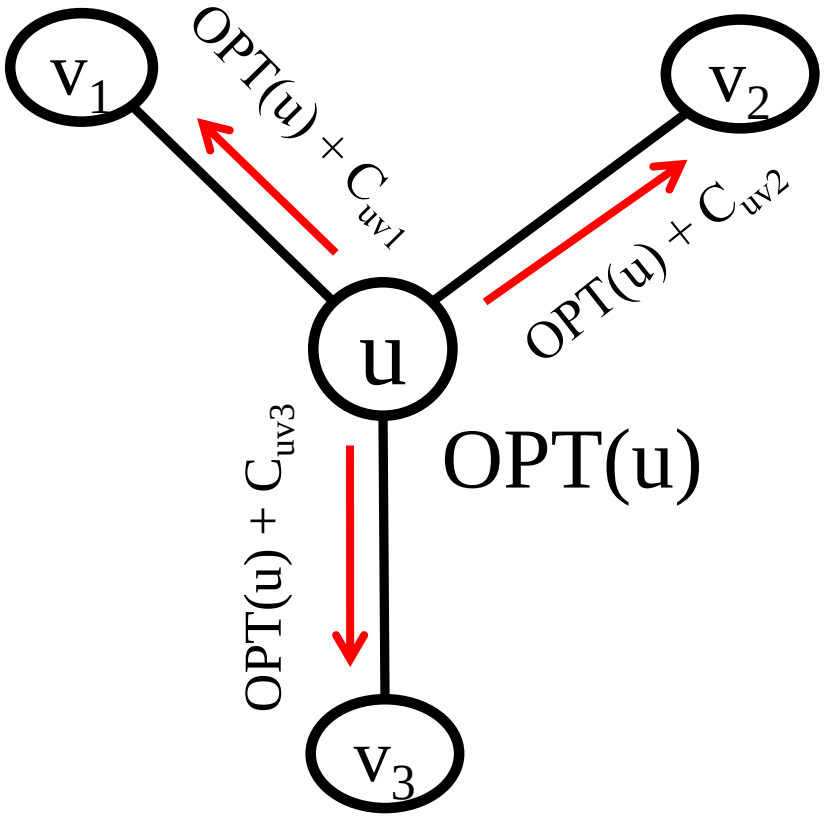
- グラフ G(V,E)において、各頂点は、辺で接続するもう片方の頂点と通信を行うものとする。



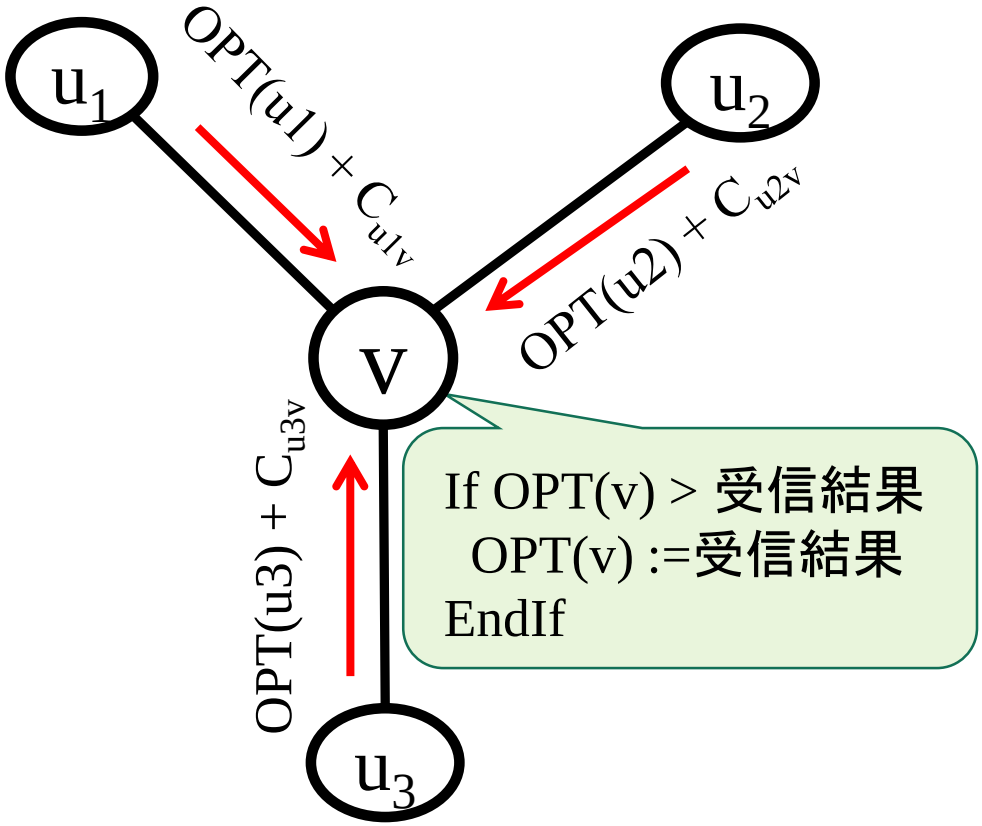
- このとき、
  - 各頂点uから、頂点v (∈nbr(u))に向けて、OPT(u)+Cuvを発信する
  - 受信側の頂点vでは、受信した値の中(他者からのものも含む)で、最小のものをOPT(v)として選ぶ
  - 開始点sのOPT(s)は 0 に固定する

という処理を行うと、ベルマンフォード法を分散的に実装できる

ベルマンフォード法の分散化：各頂点の動作



頂点uからの送信
(定期的に行う)



頂点vでの受信と処理

講義で扱った内容はどこで触れられているか

3 Graph
- 3.1 Basic Definitions and Applications
- 3.2 Graph Connectivity and Graph Traversal
- 3.3 Implementing Graph Traversal Using Queues and Stacks
- 3.6 Directed Acyclic Graphs and Topological Ordering

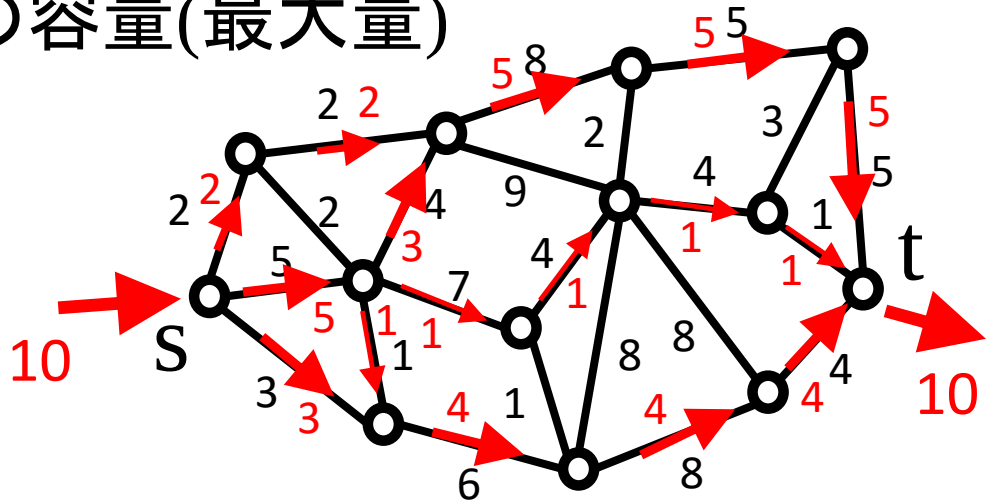
4 Greedy Algorithm
- 4.4 Shortest Paths in a Graph
- 4.5 The Minimum Spanning Tree Problem

6 Dynamic Programming
- 6.8 Shortest Paths in a Graph
- 6.9 Shortest Paths and Distance Vector Protocols

7 Network Flow
- 7.1 The Maximum-Flow Problem and the Ford-Fulkerson Algorithm

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)

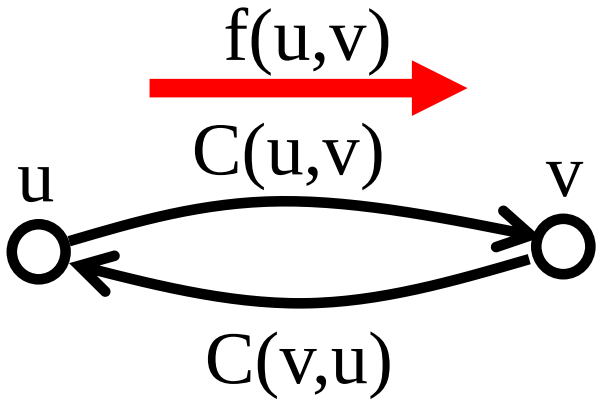
  sから流れ出す流量 = tに流れ込む流量 (グラフの中で見た場合)

  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？

フローの定式化

- 辺を流れるフロー



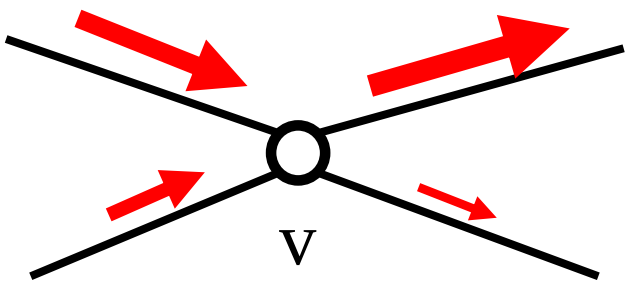
$f(u, v) \leqq C(u, v)$

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

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

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



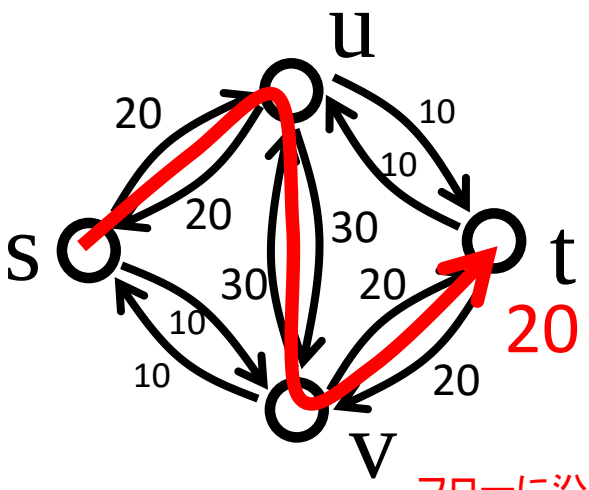
流入量: $f^{in}(v) = \sum_{u \in V,\ f(u,v)>0} f(u,v)$

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

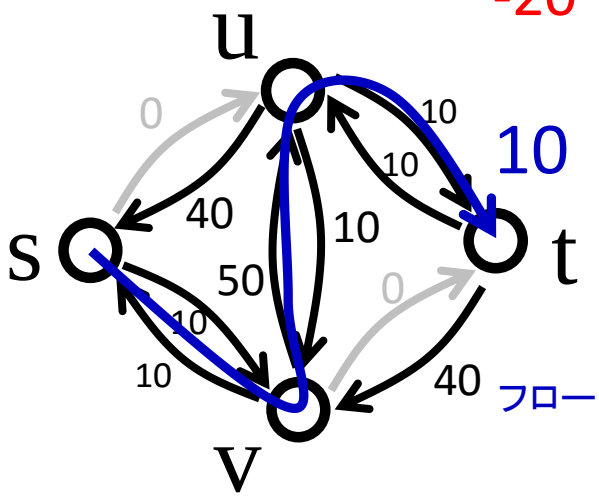
$v \neq s,t$ であれば $f^{in}(v) = f^{out}(v)$

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



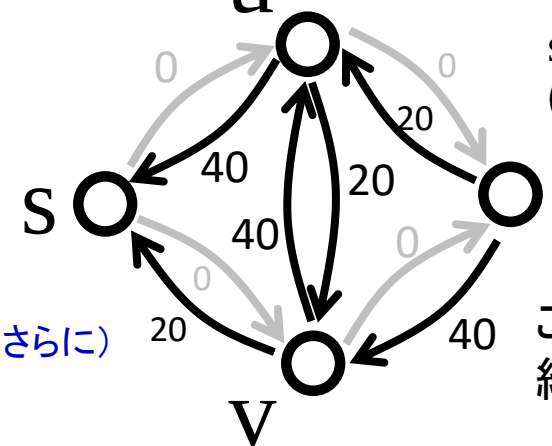
フローに沿って
-20



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

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

本日の進め方

1. これまでの講義で扱ってきた部分をピックアップ
そして、
どのような位置づけになっているか
どのような記述になっているか
を確かめる

2. 関連する重要キーワードをピックアップ
そして、その意味について、考える

キーワード

- Greedy Algorithm (貪欲アルゴリズム)
- Divide and Conquer (分割統治法)
- Dynamic Programming (動的計画法)

- NP and Computational Intractability
- Approximation Algorithms
- Local Search
- Randomized Algorithms

Greedy Algorithm (貪欲アルゴリズム)

- 「小さなステップで、より良い方向に進んでいくと、最終的に、最適解に到達できる」アルゴリズムのファミリー
- 「ローカルな問題を解決し、積み上げていくと(それ自体は正しいのかどうか自明ではないが)、最終的に全体の問題を解決できる」アルゴリズムのファミリー

- 例)
  - ダイクストラ法
  - プリム法
  - Interval Scheduling 問題
  - Optimal Caching 問題
  - Huffman Codes and Data Compression

Divide and Conquer (分割統治法)

- 「一つの問題を、複数の部分問題に分割し、それぞれの部分問題について、同様の方法で解き、それらの解を処理することで、問題の解に到達する」アルゴリズムのファミリー
- 「問題の分割・統合を再帰的に行うことによって問題を解く」アルゴリズムのファミリー

- 例）
- Mergesort
- Finding the Closest Pair of Points
- Convolutions and the Fast Fourier Transform

Dynamic Programming (動的計画法)

- 「貪欲アルゴリズム や 分割統治法よりも強力に問題を解決する」アルゴリズム
- 「貪欲アルゴリズム や 分割統治法よりも適用範囲の広い」アルゴリズム
- 「広い解の空間を作り出し、個別に最適解を計算し、注意深くそれを伝搬させることで全体的な最適解に到達する」アルゴリズム
- 「漸化式を走らせることで、力学的にそこにたどり着く（収束する）」アルゴリズム
- 「分割統治法的な考えの潜んでいる」アルゴリズム
- 「貪欲アルゴリズムの反対の方向から攻める」アルゴリズム
- 例） ベルマンフォードのアルゴリズム

計算不可能な問題への実用的アルゴリズム

- NP and Computational Intractability
  - 「計算複雑性」が理由で解けない問題が存在する。
- Approximation Algorithms
  - 最適解が得られなくても近似解を得られるものがある
  - （実用上十分であれば、良い、という考え方）
- Local Search
  - いま考えられるよりも、より良い解にたどり着く（ただし、それが全体の最適解となっているかは不明）
  - （実用上十分であれば、良い、という考え方）
- Randomized Algorithms
  - 対象の期待値を問題にすることもある
    （入力データを確率的に与えられている等）
  - 動作を乱数にゆだねるものもある

「離散数学」講義の内容（まとめ）

4月8日　集合
4月15日 関係と関数
4月22日 順序集合と束
5月6日　命題計算
5月16日 ブール代数
5月20日 グラフの構造と種類
5月27日 グラフ探索アルゴリズム
6月3日　最短経路問題
6月10日 演習（レポート提出）
6月17日 最小全域木、最大流問題
6月24日 平面グラフ
7月1日　スキップグラフ
7月8日 アルゴリズム設計