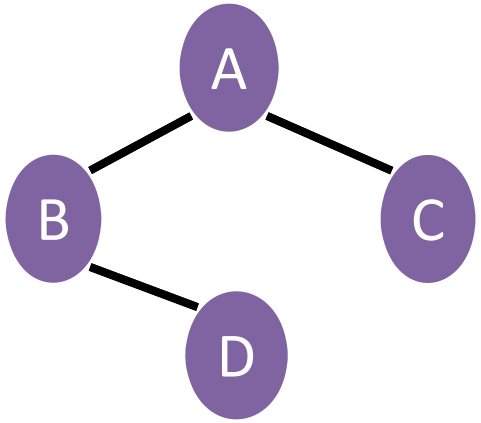
アルゴリズムとデータ構造
第12回: グラフの探索

担当: 上原 隆平(uehara)

2014/05/27

グラフの探索



- グラフの全ての頂点を組織的に訪問して何らかの問題を解くこと
  - e.g., AからDへの路がある？
- 探索方法:
  - 幅優先探索: A -> B -> C -> D
    一つの頂点vから出発，頂点vから近い順にvから到達可能なすべての頂点を訪問する
  - 深さ優先探索: A -> B -> D -> C
    一つの頂点vから出発，頂点vから到達可能な頂点を次々と訪問してその先に初めて訪問する頂点がなくなれば元に戻って別の頂点を訪問する

幅優先探索 (breadth-first search)

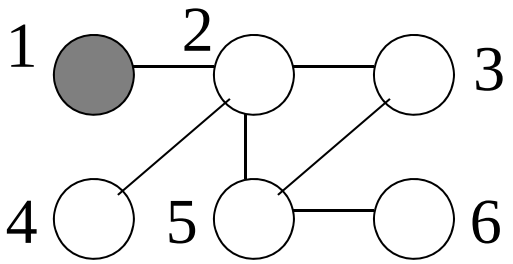
- グラフG=(V,E)、始点s∈Vについて、sから到達可能な頂点をsからの距離(辺数の最小値)の順に全て訪問
- 方法: 全ての頂点を白・灰色・黒で色づけ
  - 白: まだ訪問していない頂点
  - 灰色: 既に訪問したが
    隣接頂点に未訪問あり
  - 黒: 既に訪問済みでかつ
    隣接頂点も訪問済み

  - 全ての頂点が黒になったら探索終了
  - 頂点の色は白→灰色→黒の順に変わる

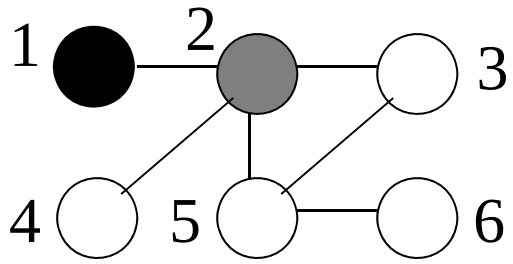
幅優先探索(breadth-first search): アルゴリズム

```
BFS(V,E,s){
  for v∈V do toWhite(v); endfor
  toGray(s);
  Q={s};
  while( Q!={} ){
    u=pop(Q); // Q → Q’ where Q={u}∪Q’
    for v∈{v∈V|(v,u)∈E}
      if isWhite(v) then
        toGray(v); push(Q,v);
      endif
    endfor
    toBlack(u);
  }
}
```

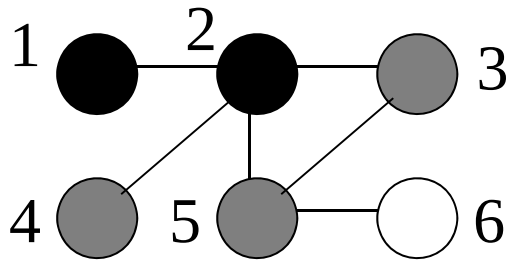
幅優先探索: 例



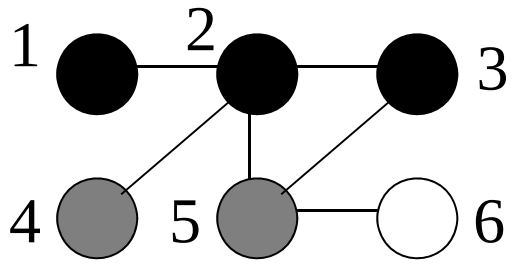
Q={1}



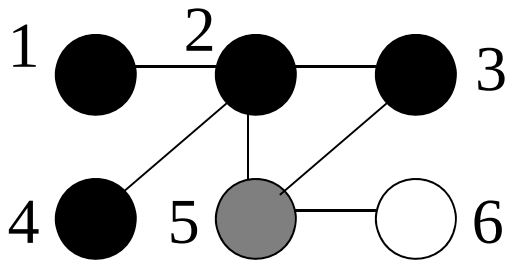
u=1,
visit 2
Q={2}
black 1



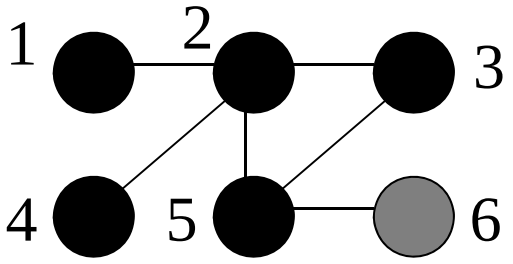
u=2,
visit 3,4,5
Q={3,4,5}
black 2



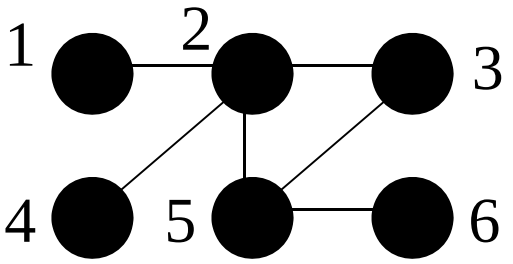
u=3,
visit null
Q={4,5}
black 3



u=4,
visit null
Q={5}
black 4



u=5,
visit 6
Q={6}
black 5



u=6,
visit null
Q={}
black 6

幅優先探索: 計算時間

- 初期化の後では頂点を白に変えることがない
- 各頂点がQに入る/取り出されるのは高々一回
- 各辺を調べるのも高々一回
  - Qから取り出された頂点についてのみ隣接頂点を調べる
- $\therefore O(|V| + |E|)$

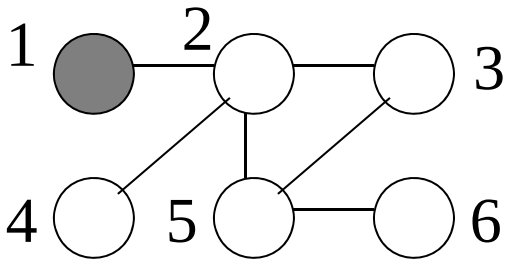
```
BFS(V,E,s){
  for v∈V do
    toWhite(v);
  endfor
  toGray(s);
  Q={s};
  while( Q!={} ){
    u=pop(Q);
    for v∈{v∈V|(v,u)∈E}
      if isWhite(v) then
        toGray(v);
        push(Q,v);
      endif
    endfor
    toBlack(u);
  }}
```

深さ優先探索(depth-first search)

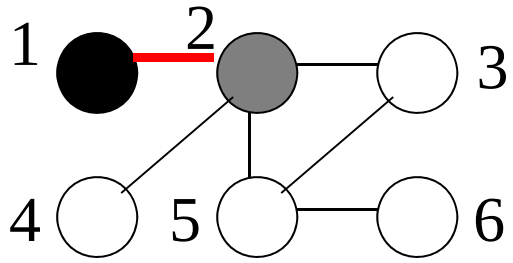
- グラフG=(V,E)、始点s∈Vについて、sから到達可能な頂点を子のない点に行き着くまで子を追いかけ、子がなくなると最も近くの探索未完了な点に戻って探索を続ける

```
dfs(V, E, s) {
    visit(s)
    for  (s, w)∈E  do
        if  notVisited(w)  then
            dfs(V, E, w)
}
```

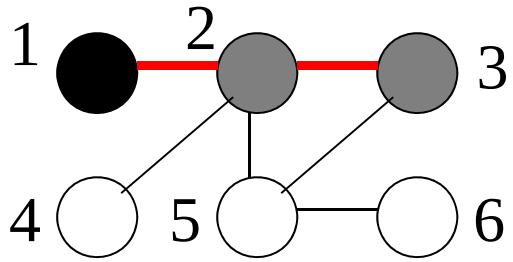
深さ優先探索:例



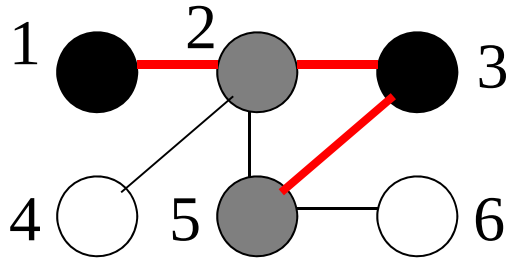
DFS(1)



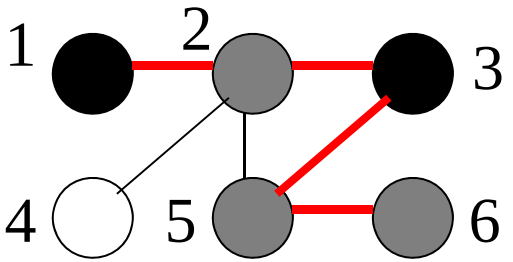
DFS(2)



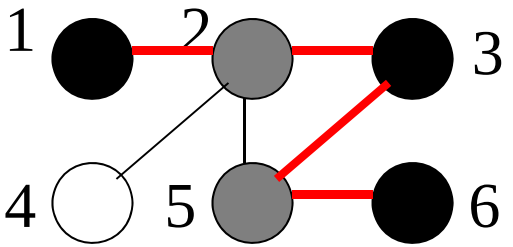
DFS(3)



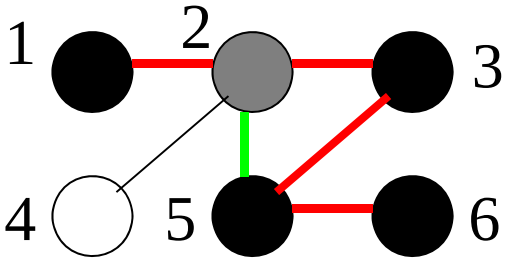
DFS(5)



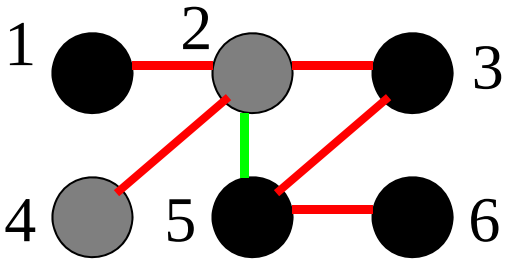
DFS(6)



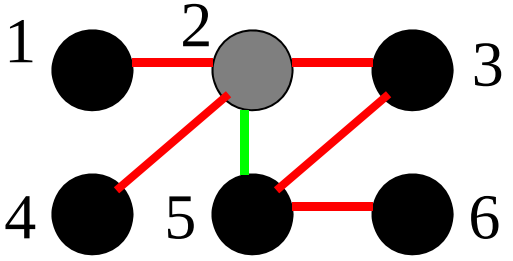
DFS(6)



DFS(5)
DFS(3)
DFS(2)



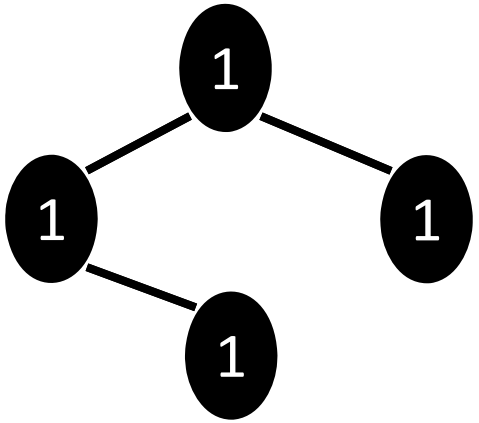
DFS(4)



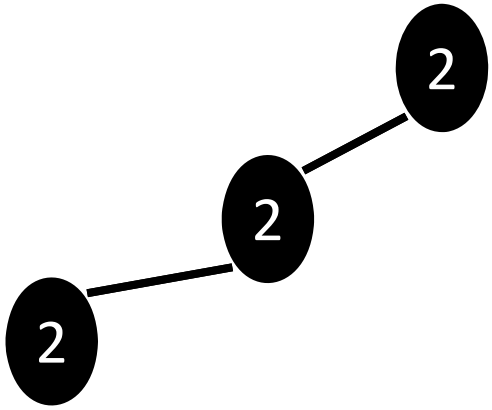
DFS(2)

深さ優先探索の応用: グラフの連結成分分解

- グラフ $G = (V, E)$ を連結グラフ $G_1 = (V_1, E_1), \ldots, G_c = (V_c, E_c)$ に分割
  - $\forall u,v \in V,\ u \in V_i \wedge v \in V_j \wedge i \neq j \Rightarrow cn[u] \neq cn[v]$ となるようなナンバリング cn を与える



G_1



G_2

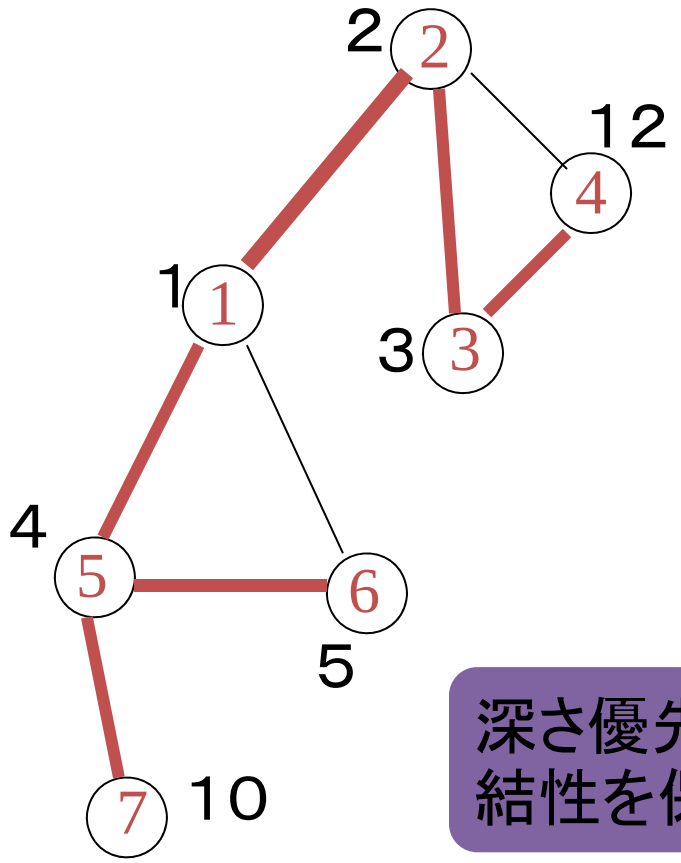
深さ優先探索の応用: グラフの連結成分分解

```
cc(V,E,cn){ //cn[|V|]
  for v∈V do
      cn[v] = 0; /*initialize*/
  endfor
  k = 1;
  for v∈V do
    if cn[v]==0 then
      dfs(V,E,v,k,cn);
      k=k+1;
    endif
  endfor
}
```

```
dfs(V,E,v,k,cn){
  cn[v]=k;
  for u∈{u|(v,u)∈E} do
    if cn[u]==0 then
       dfs(V,E,u,k,cn);
    endif
  endfor
}
```

深さ優先木の構成

連結グラフの各頂点に探索の順に番号をつけ，頂点に印を
つけるときに用いた辺からなるグラフ（閉路は含まない）



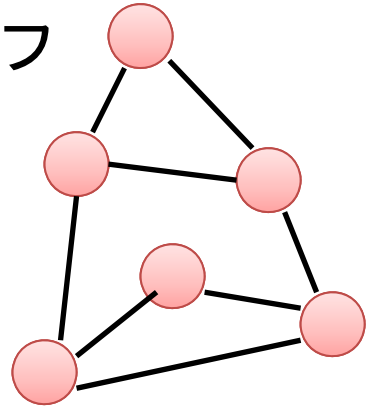
- 木辺:
  深さ優先探索木に使われる辺
- 逆辺：
  それ以外の辺（子孫 → 先祖）

深さ優先探索木は与えられた連結グラフの連結性を保つ最小数の辺からなるグラフを与える

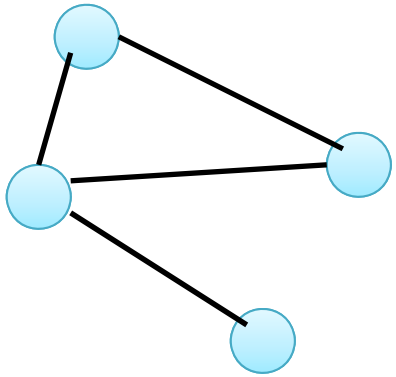
2連結成分分解

- グラフ G が2連結 ⇔ どの1頂点を消しても連結

２連結グラフ



２連結でない例



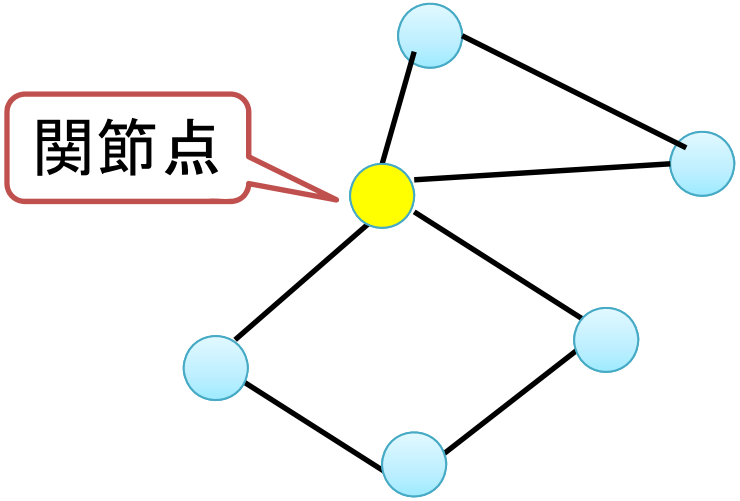
- グラフ G の2連結成分: 極大2連結部分グラフ
  - 部分グラフ: 頂点や辺の削除で得られるグラフ

問題: グラフGを2連結成分に分解せよ

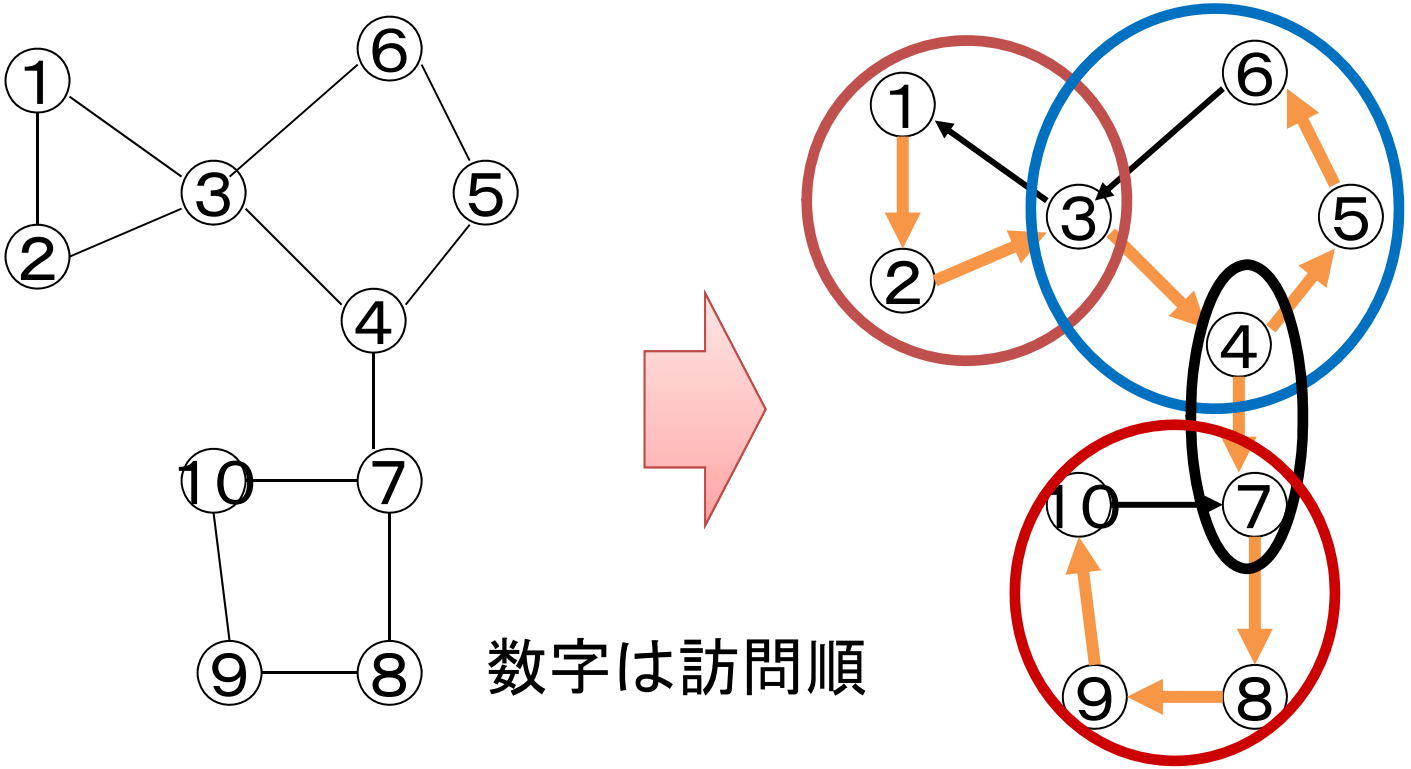
2連結成分分解

- 考え方: 深さ優先探索 (DFS) でグラフをたどる
  - DFS 木の葉に近いものから順に関節点を見付け
  - 毎回その関節点を含む一つの2連結成分を出力

  - 関節点: 複数の2連結成分に共有される頂点



2連結成分分解



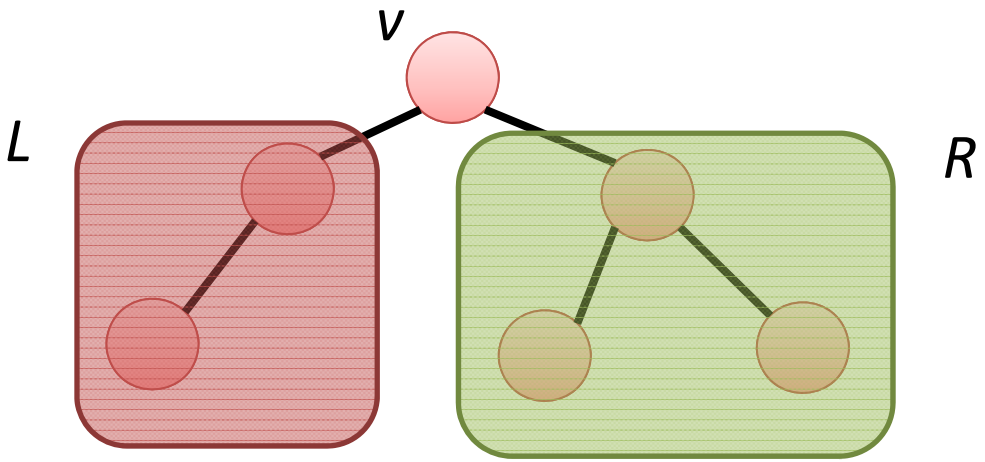
- 2連結成分: {1,2,3}, {3,4,5,6}, {4,7}, {7,8,9,10}
- 関節点: 3, 4, 7

2連結成分分解: 補助定理(1/2)

- G: グラフ，T: G の深さ優先探索木

このとき、Tの根vについて以下が成立

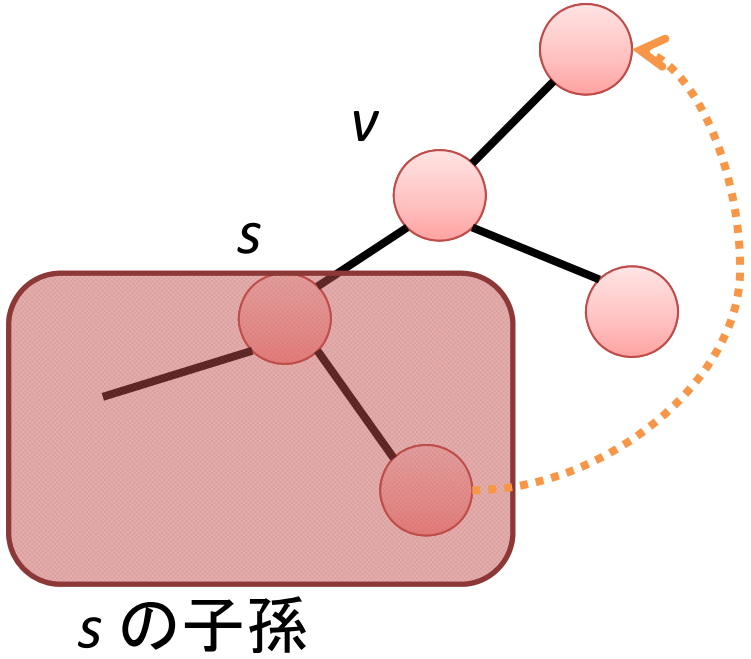
- vが関節点⇔vがTにおいて2個以上の子をもつ



L と R は互いに独立
(L から R への辺がない)

2連結成分分解: 補助定理(2/2)

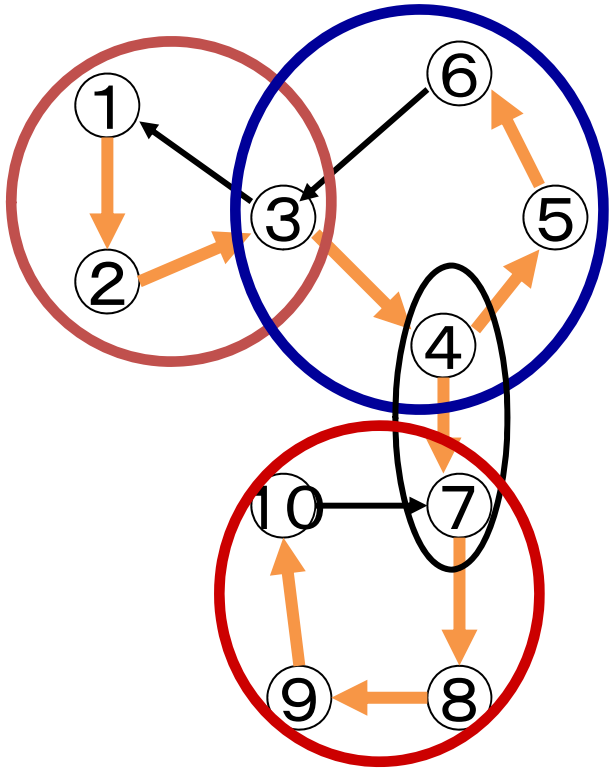
- *T*の根以外の頂点*v*について以下が成立:
  v が関節点 ⇔ *v* の子 *s* が存在して *s* のどの子孫からも *v* 以外の先祖への逆辺がない

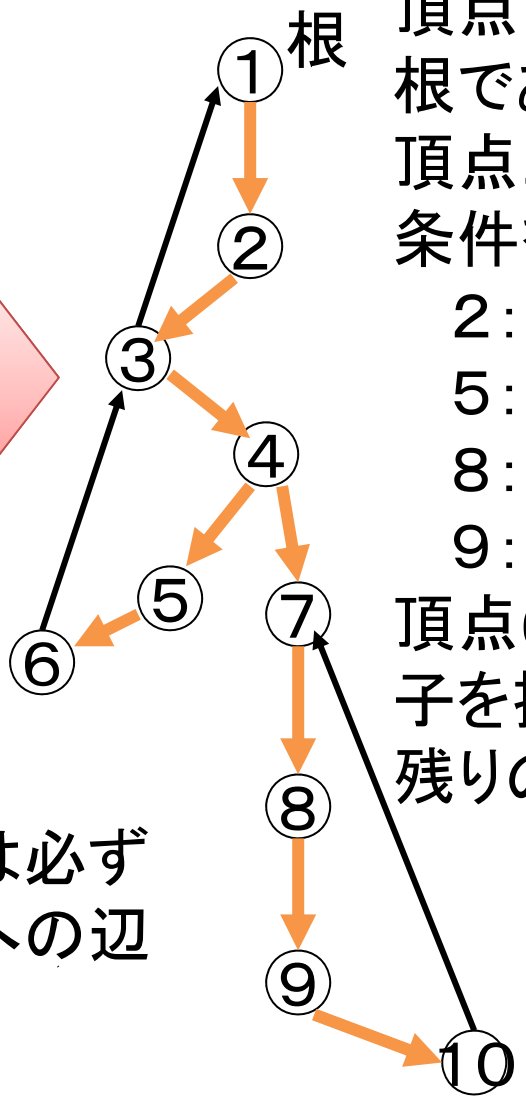


s の子孫から v 以外の先祖への逆辺がない

*先祖以外への逆辺はない
(深さ優先探索の性質)

2連結成分分解: 補助定理の利用



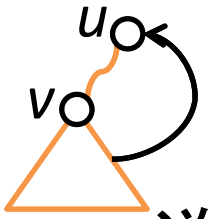


逆辺は必ず
先祖への辺

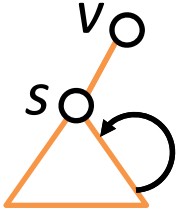
頂点1:
根であるが条件を満たさず
頂点2, 5, 8, 9:
条件を満たす子がない
　2: 子の3から逆辺,
　5:子の6から逆辺
　8: 子孫の10から逆辺,
　9:子の10から逆辺
頂点6,10:
子を持たない➔関節点でない
残りの頂点3.4.7は関節点

2連結成分分解: 補助定理の前提条件が成立することの検査

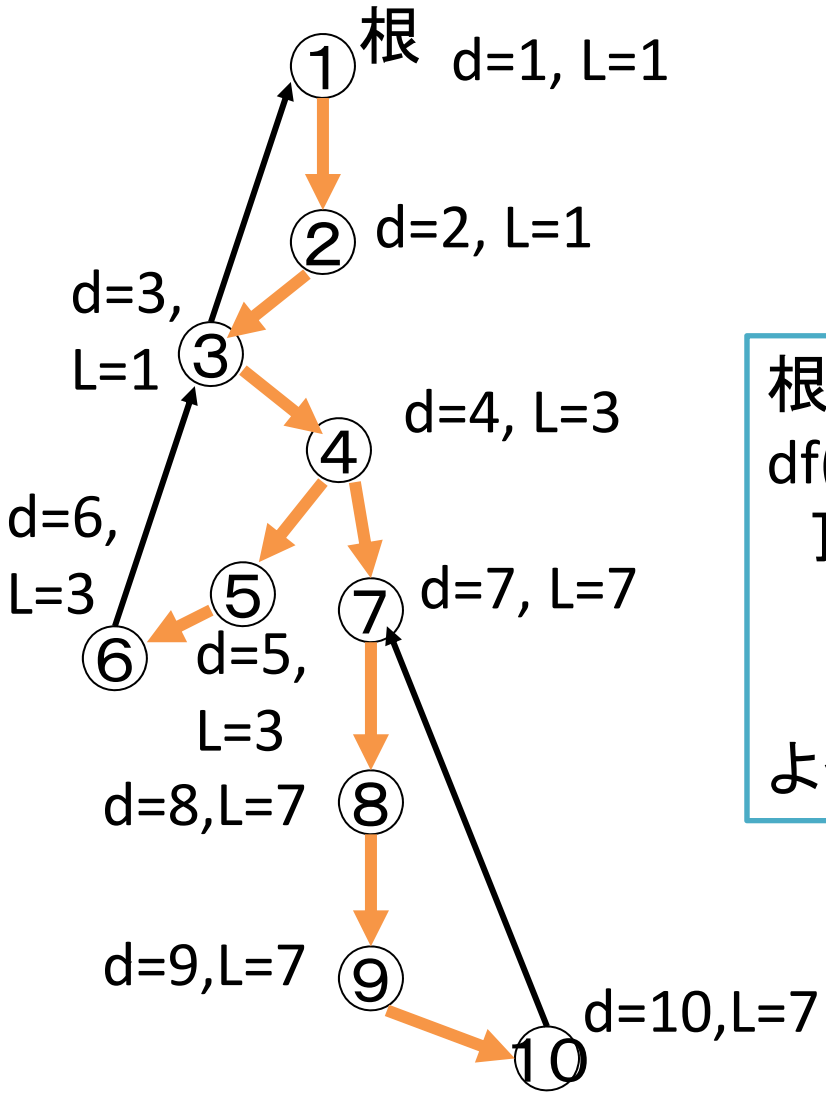
- 各頂点 v について df(v) および L(v) を定義
  - df(v): 深さ優先探索で何番目に v を訪問したか
  - L(v) = min{ df(u) | [$v = u$] or [v か v の子孫から u へ逆辺が存在] }



- vが関節点 ⇔ vは df(v)≦L(s) となる子sを持つ
  - df(先祖)≦df(子孫) より,
    - df(v) > L(s) ⇒ s の子孫から v の先祖への逆辺が存在
    - df(v)≦L(s) ⇒ 逆辺は存在しない



2連結成分分解: 補助定理の前提条件が成立することの検査



根以外の頂点 v について
df(v)≦L(s)となる子 s をもつ頂点は，
　頂点3　df(3) ≦ L(4)
　頂点4　df(4) ≦ L(7)
　頂点7　df(7) ≦ L(8)
よって，これらの頂点が関節点

2連結成分分解: L(v)の求め方

- L(*v*)は以下のうちの最小値
  - df(*v*)
  - {L(*w*) | *w* は *v* の子}
  - {df(*w*) | (*v*, *w*) は逆辺}
- よって深さ優先探索を実行しながら計算可能
  - dfs(*v*) の中で上記の三つの値の最小値を計算

```
dfs(v) {
 df[v] = c; c=c+1;
 L(v) = df[v];                    i
 for(w=0; w<n; ++w) {
   if(df[w]==0 && g[v][w]==1){
     dfs(w);
     if(df[v]≦L[w]) // 関節点 v 発見
     L[v] = min(L[v], L[w]);      ii
   }
   else if (df[w] > 0 && g[v][w] == 1)
     L[v] = min(L[v], df[w]);     iii
 }
}
```

子

先祖

L(v) = min of
(i) df(v),
(ii) L(w) : w∈子(v),
(iii) df(w) : 逆辺(v,w)

```
dfs(v) {
 df[v] = c; c=c+1;
 L(v) = df[v];                    i
 for(w=0; w<n; ++w) {
   if(df[w]==0 && g[v][w]==1){
     push(v,w); // 辺(v,w)をスタックに積む
     dfs(w);
     if(df[v]≦L[w]) // 関節点 v 発見
       辺(v, w)が出てくるまで
         スタックから辺をpopして出力
     L[v] = min(L[v], L[w]);      ii
   }
   else if (df[w] > 0 && g[v][w] == 1)
     L[v] = min(L[v], df[w]);     iii
 }
}
```

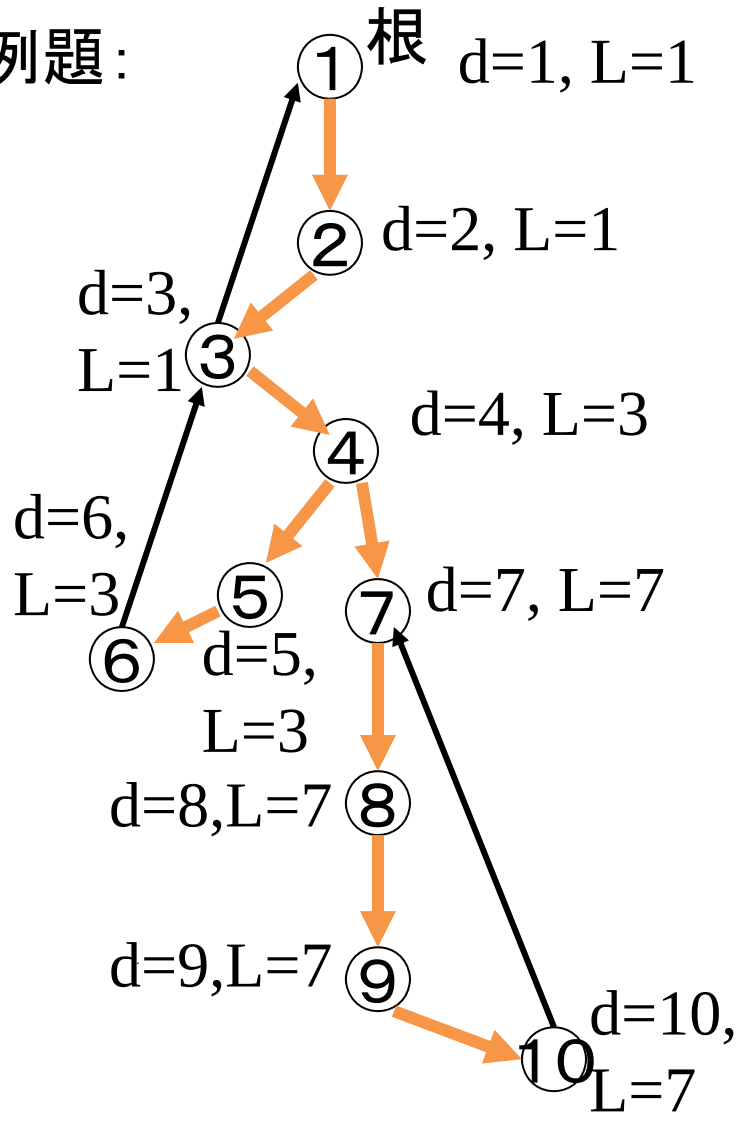
子

先祖

2連結成分出力

L(v) = min of
(i) df(v),
(ii) L(w) : w∈子(v),
(iii) df(w) : 逆辺(v,w)

例題:



push (1,2), (2,3)
L[3]=1
push (3,4), (4,5), (5,6)
L[6]=3, L[5]=3
L[4]=3
push (4,7),(7,8),(8,9),(9,10)
L[10]=7,L[9]=7
L[8]=7
new Comp: (9,10),(8,9),(7,8)
new comp: (4,7)
L[4]=3
new comp: (5,6),(4,5),(3,4)
L[3]=1,L[2]=1
new comp: (2,3),(1,2)

ミニ演習

- どの点から始めても
  幅優先探索木 = 深さ優先探索木
  となる連結グラフを一つ挙げよ
- どの点から始めても
  幅優先探索木 ≠ 深さ優先探索木
  となる連結グラフを一つ挙げよ

- 注意: = と ≠ は辺集合が等しいか否かを表す