算法数理工学
第8回

定兼　邦彦

万能ハッシュ法

- 運が悪いと，n 個のキーが同じスロットにハッシュされ，平均検索時間が $\Theta(n)$ になってしまう
- 万能ハッシュ法 (universal hashing) では，ハッシュ関数をランダムに選択する
- どのように意地悪くキーを選択しても，平均として良い性能を示す．

- H: キーの普遍集合 U から値域 $\{0,1,\ldots,m-1\}$へのハッシュ関数の有限集合
- H が万能 (universal) ⇔ 全ての異なるキーの組 $x, y \in U$ に対し, $h(x) = h(y)$ となるハッシュ関数 $h \in H$ の個数が $|H|/m$ 以下
- ハッシュ関数を万能な H の中からランダムに選んだときに, x と y が衝突する確率が $1/m$ 以下
- これは $h(x)$ と $h(y)$ が値域 $\{0,1,\ldots,m-1\}$ からランダムに選択されたときの衝突確率に等しい

定理 h を万能な集合から選択されたハッシュ関数とする. h を用いて n 個のキーをサイズが m のハッシュ表にハッシュする. 衝突はチェイン法で解消する. このとき, キー k のハッシュ先のリストの長さの期待値 $\mathrm{E}[n_{h(k)}]$ は

高々α(キーが表に存在しないとき)

高々1+α(キーが表に存在するとき)

期待値の計算は全てハッシュ関数に関して行い,

キーの分布については何も仮定しないことに注意

証明:異なるキーのペア k, l に対して，指標確率変数

$$X_{kl} = \begin{cases} 1 & (h(k) = h(l)) \\ 0 & (h(k) \neq h(l)) \end{cases}$$ を定義する.

ハッシュ関数の定義より，1つのキーのペアが衝突を起こす確率は高々 $1/m$. つまり $\Pr\{h(k)=h(l)\} \leq 1/m$

よって $\mathrm{E}[X_{kl}] \leq 1/m$.

キー k に対し，k 以外で k と同じスロットにハッシュされるキーの個数を確率変数 Y_k で表す.

$$Y_k = \sum_{\substack{l \in T \\ l \neq k}} X_{kl}$$

$$\mathrm{E}[Y_k] = \mathrm{E}\left[\sum_{\substack{l \in T \\ l \neq k}} X_{kl}\right] = \sum_{\substack{l \in T \\ l \neq k}} \mathrm{E}[X_{kl}] \leq \sum_{\substack{l \in T \\ l \neq k}} \frac{1}{m}$$

- $k \notin T$ のとき $n_{h(k)} = Y_k$ かつ $|\{l : l \in T \text{ and } l \neq k\}| = n$
従って $\mathrm{E}\left[n_{h(k)}\right] = \mathrm{E}[Y_k] \leq \dfrac{n}{m} = \alpha$

- $k \in T$ のとき, キー k はリスト $T[h(k)]$ に存在し, カウント Y_k には k は含まれていないので

$$n_{h(k)} = Y_k + 1 \quad \text{かつ} \quad |\{l : l \in T \text{ and } l \neq k\}| = n - 1$$

従って $\mathrm{E}\left[n_{h(k)}\right] = \mathrm{E}[Y_k + 1] \leq \dfrac{n-1}{m} + 1 < 1 + \alpha$

万能ハッシュ関数族の設計

- どんなキー k も 0 から p-1 までの範囲に入るような十分大きな素数 p を選ぶ. $p > m$ を仮定.
- $h_{a,b}(k) = ((ak + b) \bmod p) \bmod m$ と定義
  ただし $a \in \{1,2,\ldots,p\text{-}1\}$, $b \in \{0,1,,\ldots,p\text{-}1\}$.
  m は素数でなくてもいい.
- 定理: ハッシュ関数のクラス
  $H_{p,m} = \left\{h_{a,b}: a \in Z_p^*, b \in Z_p\right\}$ は万能である.
- 証明: 相異なるキー $k, l \in Z_p$ を考える. ハッシュ関数 $h_{a,b}$ に対し
  $r = (ak+b) \bmod p$
  $s = (al+b) \bmod p$ と置く.

- 命題：$r \neq s$

  $r - s \equiv a(k - l) \pmod{p}$ である．

  a も $k - l$ も法 p の下で 0 ではない．

  p は素数だから右辺の積も 0 ではない．

- (k,l) を固定する． $p(p-1)$ 個存在するハッシュ関数のパラメタのペア (a,b) は，(k,l) を異なるペア (r,s) に写像する．

- $r \neq s$ であるペアは $p(p-1)$ 個存在するので，(a,b) と (r,s) には1対1対応がある．

$$a = \left((r-s)\left((k-l)^{-1} \bmod p\right)\right) \bmod p$$

$$b = (r - ak) \bmod p$$

- (a,b) を一様ランダムに選べば，(r,s) も一様ランダム．

- 従って，相異なるキー k と l が衝突する確率は，法 p の下で相異なる r と s をランダムに選択したときに $r \equiv s \pmod{m}$ となる確率に等しい．
- r を固定すると，r 以外の $p-1$ 個の値の中で $r \equiv s \pmod{m}$ となる s の個数は高々

$$\left\lceil \frac{p}{m} \right\rceil - 1 \le \frac{p+m-1}{m} - 1 = \frac{p-1}{m}$$

- よって，s と r が衝突する確率は高々 $1/m$
- 従って，任意の異なる値 $k,l \in Z_p$ のペアに対し

$$\Pr\{h_{a,b}(k) = h_{a,b}(l)\} \le \frac{1}{m}$$

つまり $H_{p,m} = \{h_{a,b} : a \in Z_p^*, b \in Z_p\}$ は万能

線形時間のソーティング

- 今までのソートアルゴリズム
  - 入力要素の比較のみに基づく
  - 比較ソート (comparison sort) と呼ぶ
  - 例：マージソート，ヒープソート，クイックソート
  - 計算量：$\Omega(n \lg n)$
- この節でのソートアルゴリズム
  - 比較以外の演算を用いる
  - 例：計数ソート，基数ソート
  - 計算量：$O(n)$ (線形時間)

計数ソート (counting sort)

- n 個の各入力要素が 1 から k の範囲の整数であると仮定 (k: ある整数)
- $k = O(n)$ のとき，計数ソートは $O(n)$ 時間
- 基本的なアイデア
  – 各入力要素 x に対し，x より小さい要素の数を決定
  – その要素数から x の出力配列での位置が決まる
  – 例: x より小さい数が17個 ⇒ x の出力位置は18
  – 複数の要素が同じ値を持つ場合に対応する必要あり

計数ソートのプログラム

- $A[1..n]$, $B[1..n]$: 入出力配列
- $C[1..k]$: 補助的な配列

```
void COUNTING_SORT(int *A, int *B, int *C, int n, int k)
{
  int i,j;
  for (i=1; i<=k; i++) C[i] = 0;
  for (j=1; j<=n; j++) C[A[j]] = C[A[j]] + 1;
  //  C[i] は i に等しい要素の個数を含む

  for (i=2; i<=k; i++) C[i] = C[i] + C[i-1];
  //  C[i] は i 以下の要素の個数を含む

  for (j=n; j>=1; j--) {
    B[C[A[j]]] = A[j];
    C[A[j]] = C[A[j]] - 1;
  }
}
```

	1	2	3	4	5	6	7	8
A	3	6	4	1	3	4	1	4

	1	2	3	4	5	6
C	2	0	2	3	0	1

```
for (j=1; j<=n; j++) C[A[j]] = C[A[j]] + 1;
 //  C[i] は i に等しい要素の個数を含む
```

	1	2	3	4	5	6
C	0	2	2	4	7	7

```
for (i=2; i<=k; i++) C[i] = C[i] + C[i-1];
 //  C[i] は i 以下の要素の個数を含む
```

	1	2	3	4	5	6	7	8
B	1	1	3	3	4	4	4	6

```
for (j=n; j>=1; j--) {
 B[C[A[j]]] = A[j]; // A[j]以下がC[A[j]]個
 C[A[j]] = C[A[j]] - 1;
}
```

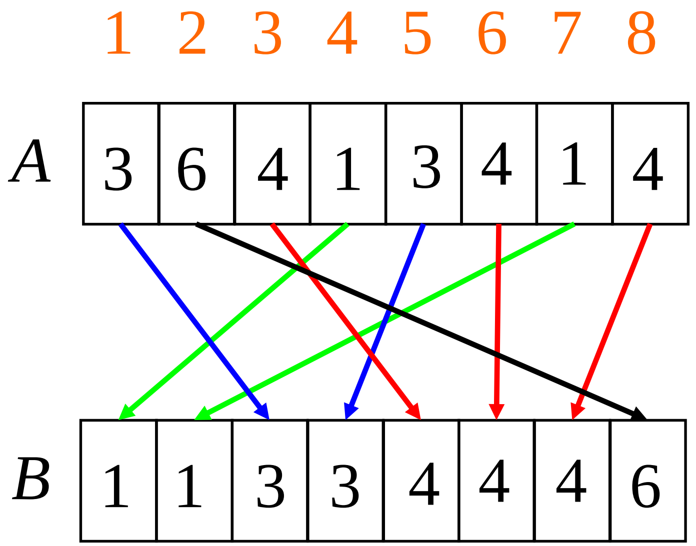
計数ソートの計算時間

- C の初期化: O(k) 時間
- 頻度の計算: O(n) 時間
- 累積頻度の計算: O(k) 時間
- 出力配列の計算: O(n) 時間

- 全体で O($k + n$) 時間
- k = O(n) のとき，全体で O(n) 時間
- 比較ソートの下限 Ω(n lg n) を下回る

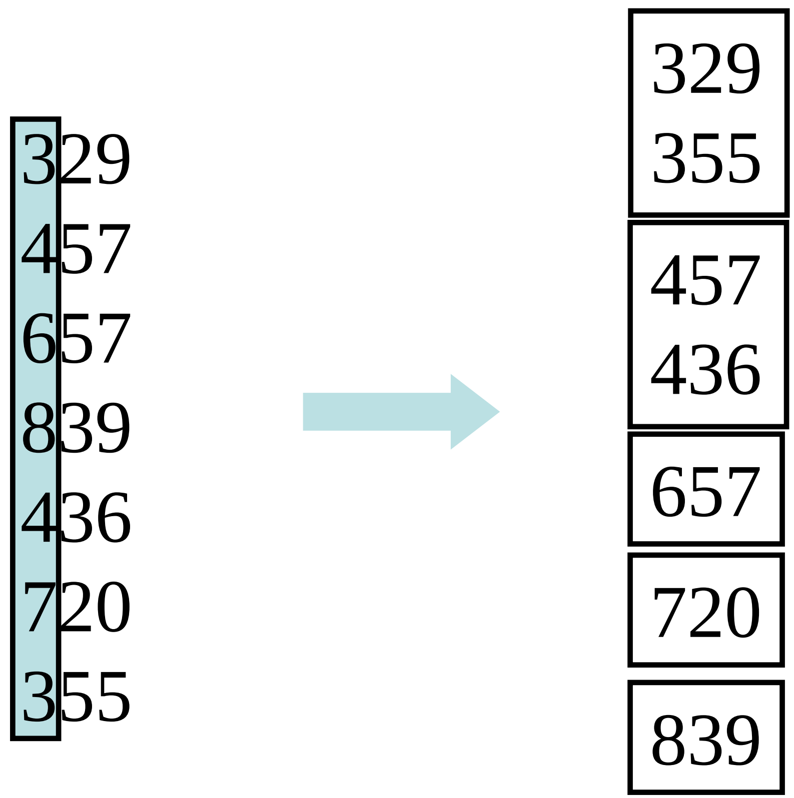
安定なソート

- 同一の値は入力と出力で同じ順序になる
- 付属データをキーに従ってソートする場合に重要



基数ソート (radix sort)

- 数の集合をある桁に従って分割する機械がある
- この機械を用いて d 桁の数 n 個をソートしたい



- まず最上位桁でソート (分割) し，それぞれを再帰的にソートすることを考える
- 大量の部分問題が生成される

- 解 : 最下位桁からソートする⇒基数ソート
  - まず最下位桁に従ってソート
  - 次に下から2桁目に従ってソート
  - d 桁目まで繰り返す

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

基数ソートでは各桁のソートは安定である必要がある

基数ソートの擬似コード

RADIX_SORT(A,d)

1. for (i=1; i<=d; i++)
2. 　　安定ソートを用いて i 桁目に関して配列をソート

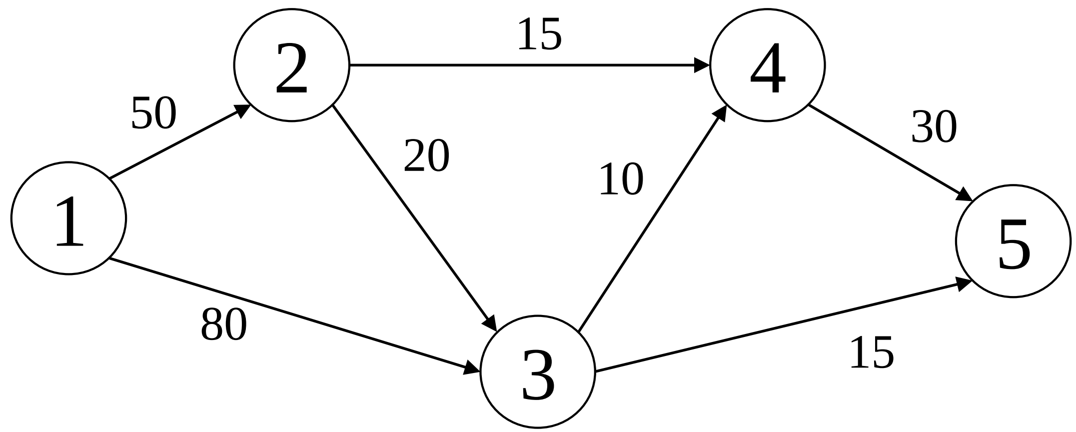
実行時間の解析

- 各桁が 1 から k の範囲として計数ソートを使用
- d パスあるので $\Theta(d(n+k))$ 時間
- d が定数で $k = O(n)$ ならば，全体で $O(n)$ 時間

- n = 100万個の64ビットの数をソートするとする
- 比較ソートでは1つの数あたりほぼ $\lg n = 20$ 回の操作が必要となる
- 基数ソートでは，これらの数を4桁の2^{16}進数と思うと，4回のパスでソートできる
- 計数ソートを用いる基数ソートはin-placeではない
  – メモリが高価な場合はクイックソートなどの方が良い

グラフ

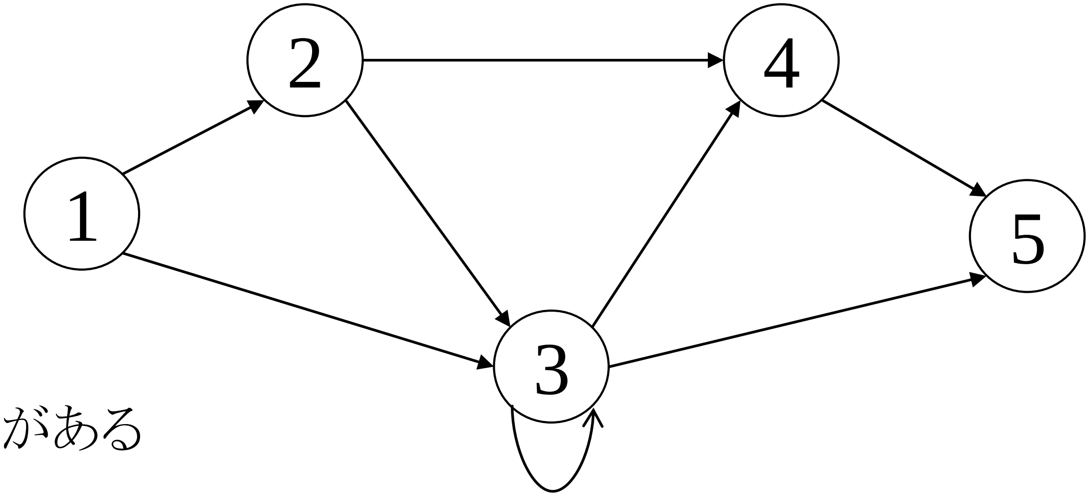
- グラフ $G = (V, E)$
  - V: 頂点 (節点) 集合 $\{1,2,\ldots,n\}$
  - E: 枝集合, $E \subseteq V \times V = \{(u,v) \mid u, v \in V\}$



- 無向グラフ
  - 枝は両方向にたどれる
- 有向グラフ
  - 枝 (u,v) は u から v の方向のみたどれる
  - 注: 無向グラフは有向グラフで表現できる (枝数を倍にする)
- 枝には重み(長さ)がついていることもある
  - なければ全て長さ 1 とみなす

グラフの表現

- 節点数 n, 枝数 m とする



- 隣接行列表現
  - n 行 n 列の行列 A で表現

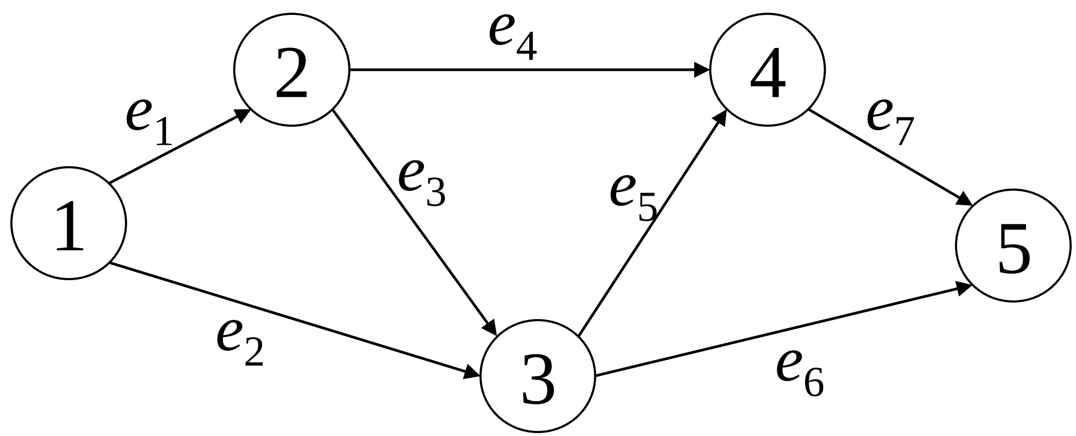
$$A_{ij} = \begin{cases} 1 & i \text{ から } j \text{ への枝がある} \\ 0 & \text{無い} \end{cases}$$

  - 領域計算量 $O(n^2)$
  - i 行目は節点 i から出ている枝を表す
  - 無向グラフなら対称行列
  - 対角成分が 1 ⇔ セルフループ

$$\begin{array}{c} ① \\ ② \\ ③ \\ ④ \\ ⑤ \end{array} \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

接続行列表現

- n 行 m 列の行列 A で表現
- 行は節点に，列は枝に対応
- 領域計算量 O(mn)
- 各枝に対し，
  - 始点の行に 1
  - 終点の行に –1
- 無向グラフなら両方 1
- セルフループは表せない

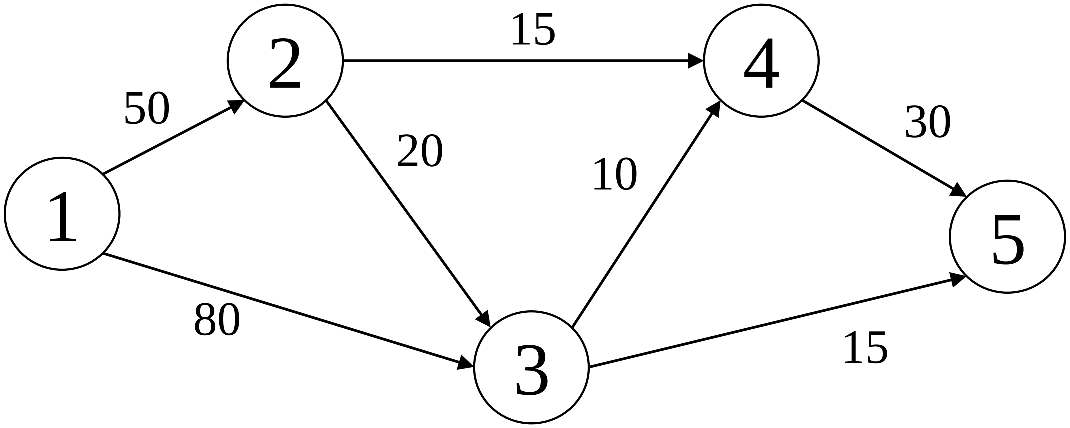


$$
\begin{array}{c c}
 & \begin{array}{ccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \end{array} \\
\begin{array}{c} ① \\ ② \\ ③ \\ ④ \\ ⑤ \end{array} &
\begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & -1 & -1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & -1 & -1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & -1 & -1
\end{pmatrix}
\end{array}
$$

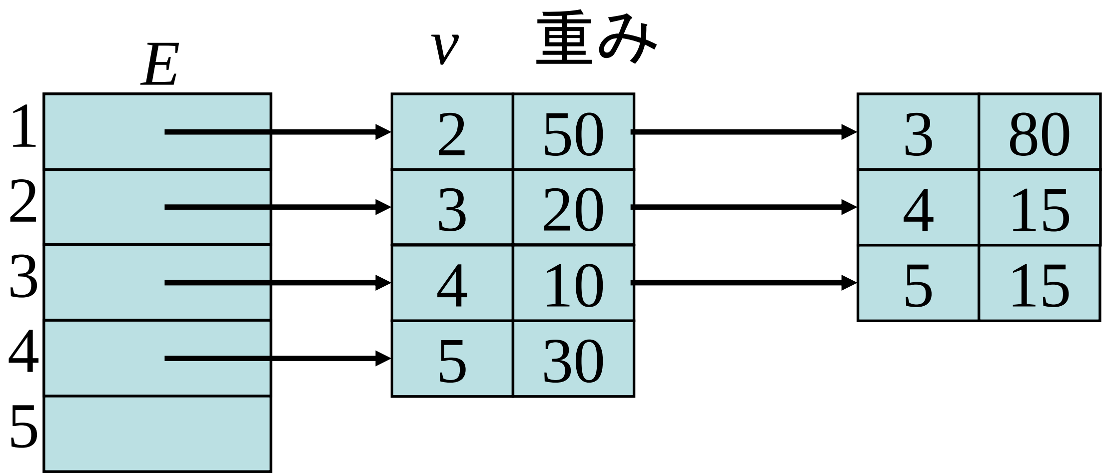
疎な有向グラフのデータ構造

- グラフが疎 $\Leftrightarrow m = \mathrm{o}(n^2)$ とする
- グラフが疎の場合，行列による表現は冗長

- 各節点 u ごとに，そこから出ている枝 (u,v) をリストに格納する
- u は共通なので v だけリストに入れる
- 枝に重み w がついているときは (v, w) をリストに入れる
- 隣接行列を圧縮したようなもの
- グラフの隣接リスト表現と呼ぶことにする



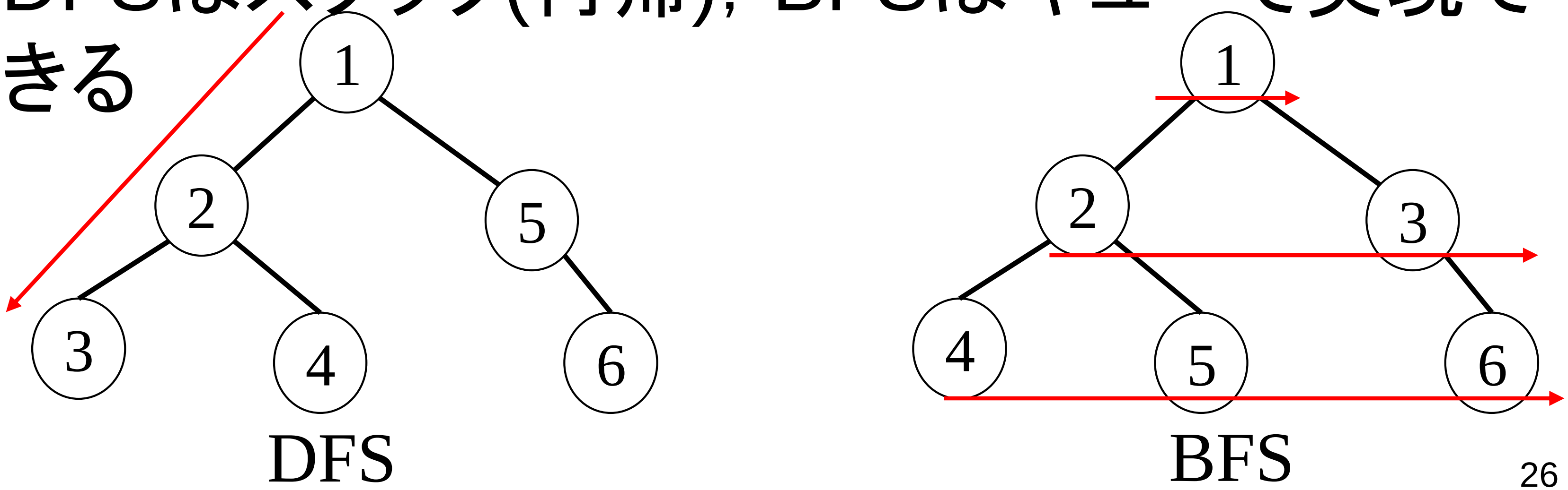
$$\begin{array}{c} ① \\ ② \\ ③ \\ ④ \\ ⑤ \end{array} \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



領域計算量 O(n+m)

深さ優先探索，幅優先探索

- 木やグラフの探索法
- 深さ優先探索 (depth-first search, DFS)
  - 行き止まりになるまで先に進む
- 幅優先探索 (breadth-first search, BFS)
  - 全体を同時に探索する
- DFSはスタック(再帰)，BFSはキューで実現できる



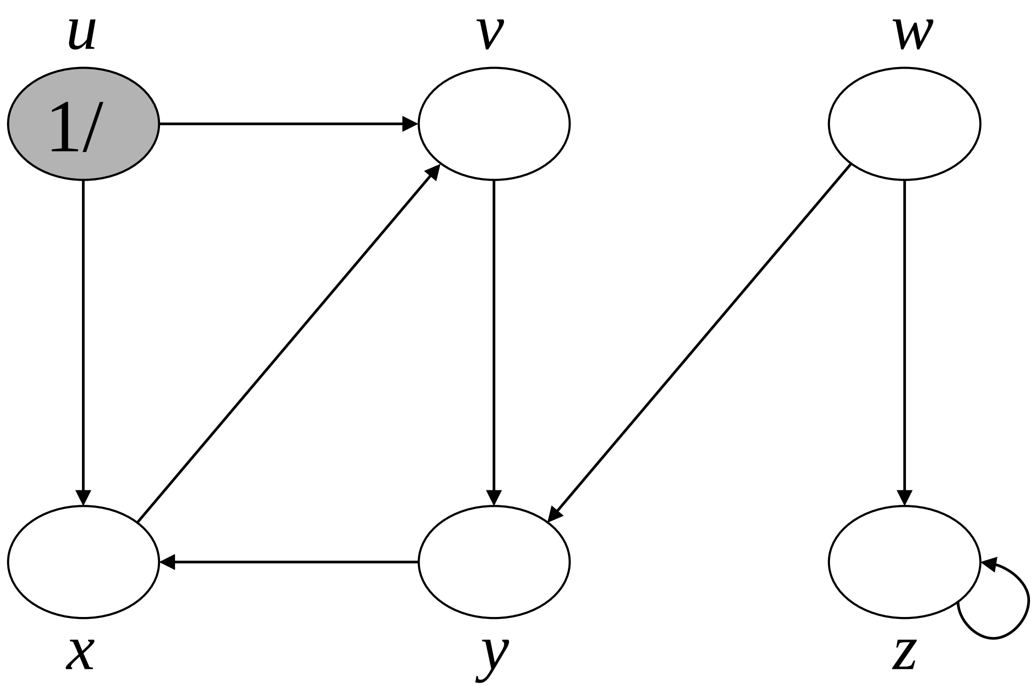
DFS(G)

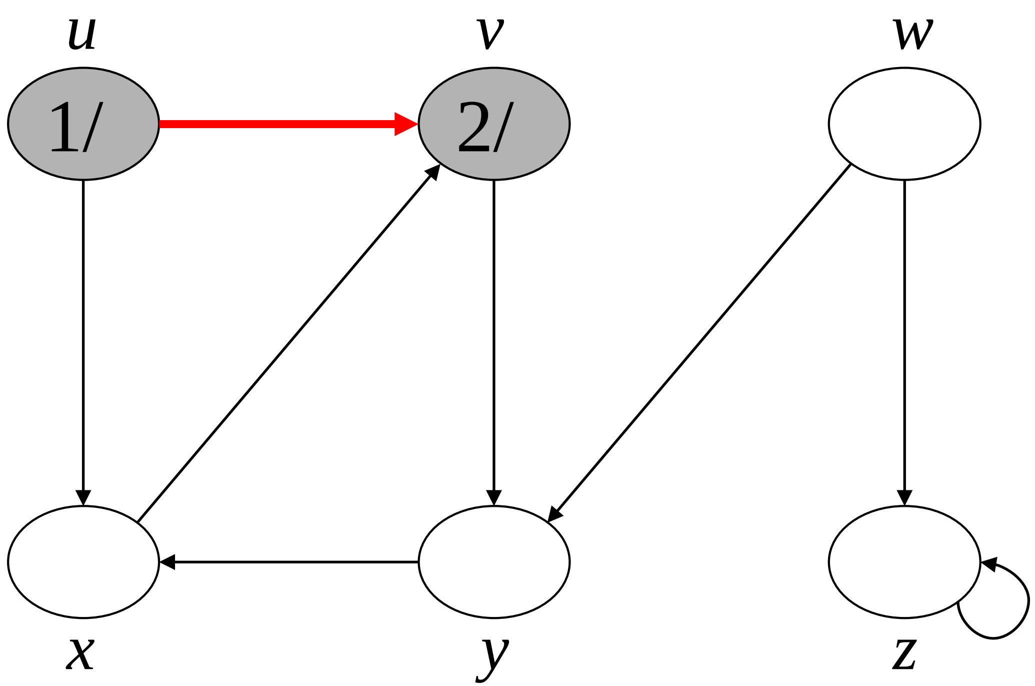
1. for each $u \in V[G]$
2. $\quad$ $color[u] \leftarrow$ 白
3. $\quad$ $\pi[u] \leftarrow$ NIL
4. $time \leftarrow 0$
5. for each $u \in V[G]$
6. $\quad$ if $color[u] =$ 白
7. $\quad\quad$ DFS-Visit(u)

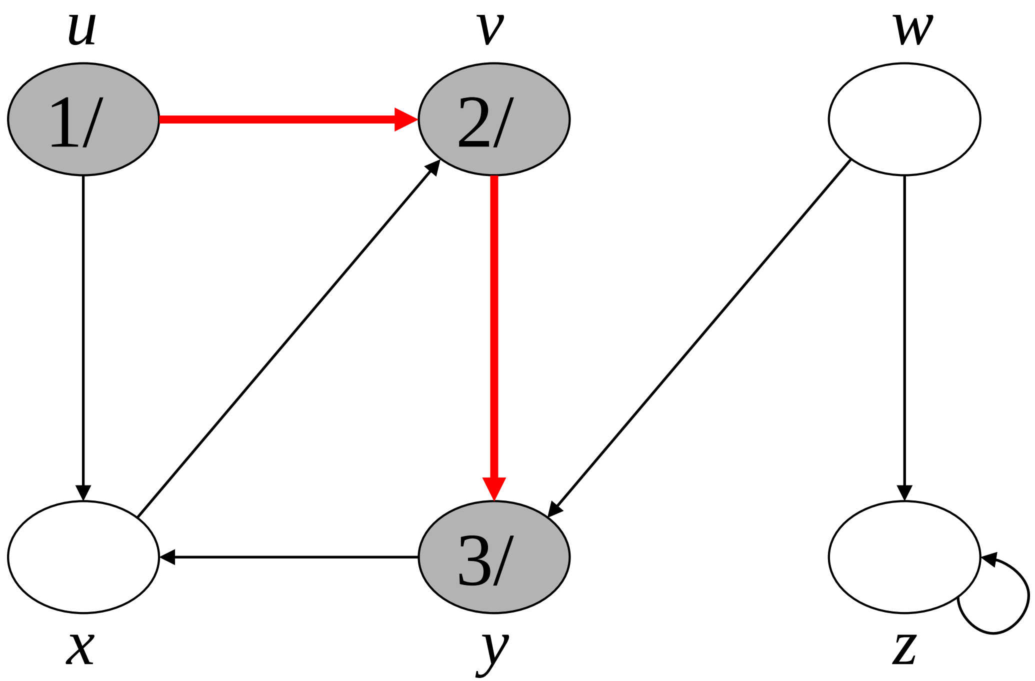
DFS-Visit(u)

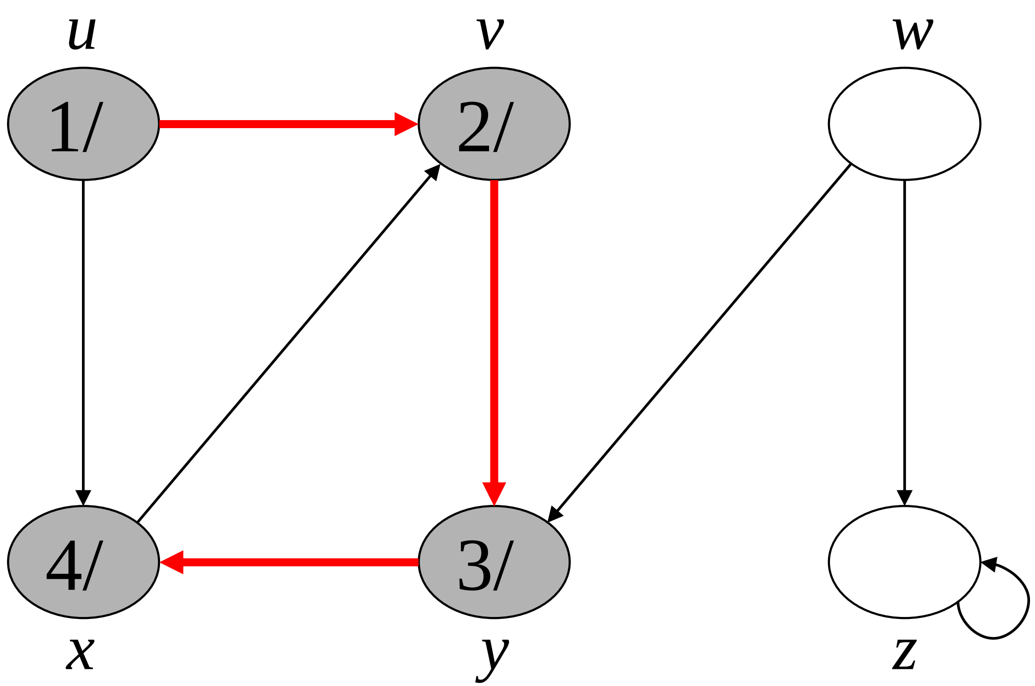
1. $color[u] \leftarrow$ 灰
2. $time \leftarrow time + 1$
3. $d[u] \leftarrow time$
4. for each $v \in Adj[u]$
5. $\quad$ if $color[v] =$ 白
6. $\quad\quad$ $\pi[v] \leftarrow u$
7. $\quad\quad$ DFS-Visit(v)
8. $color[u] \leftarrow$ 黑
9. $time \leftarrow time + 1$
10. $f[u] \leftarrow time$

白：未訪問
灰：探索中
黒：探索終了

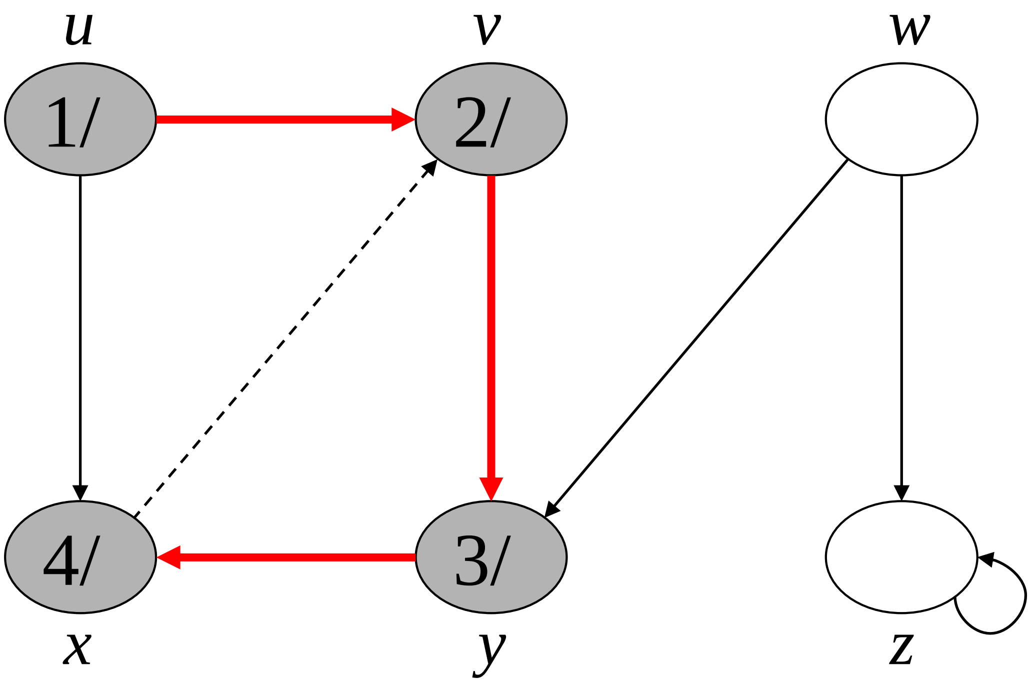



u
v
w
1/
2/
x
y
z

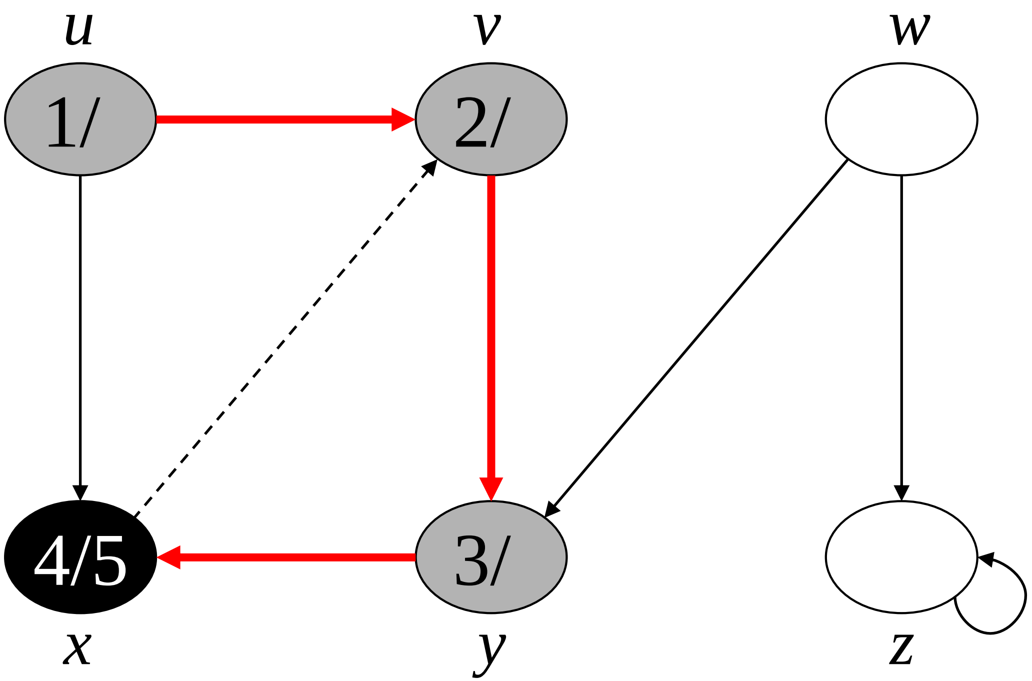

u
v
w
1/
2/
3/
x
y
z

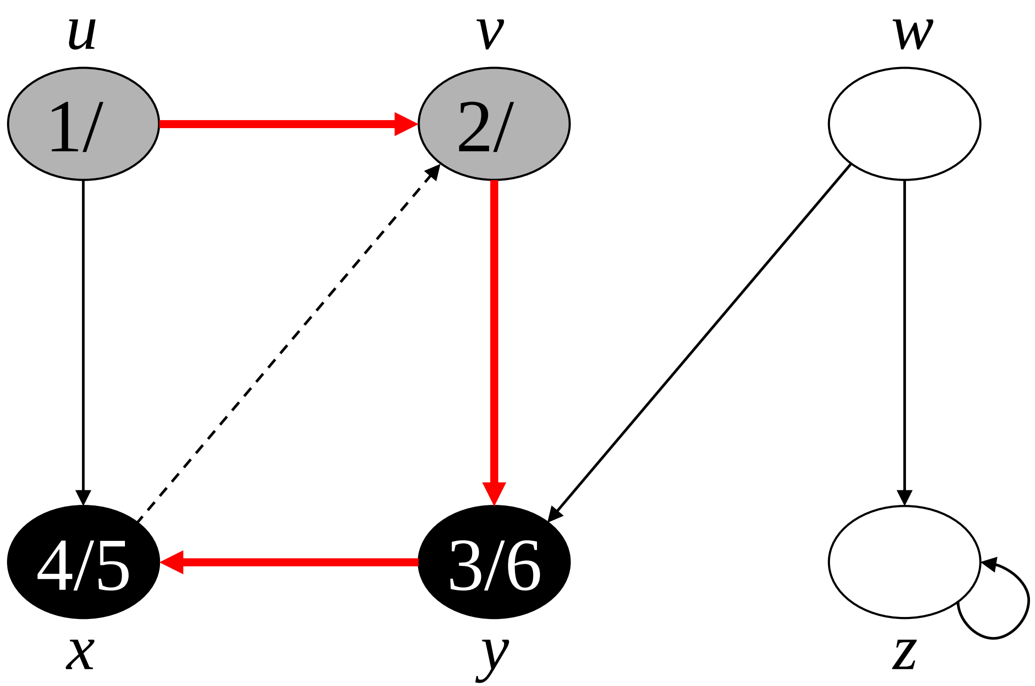

u
v
w
1/
2/
4/
3/
x
y
z

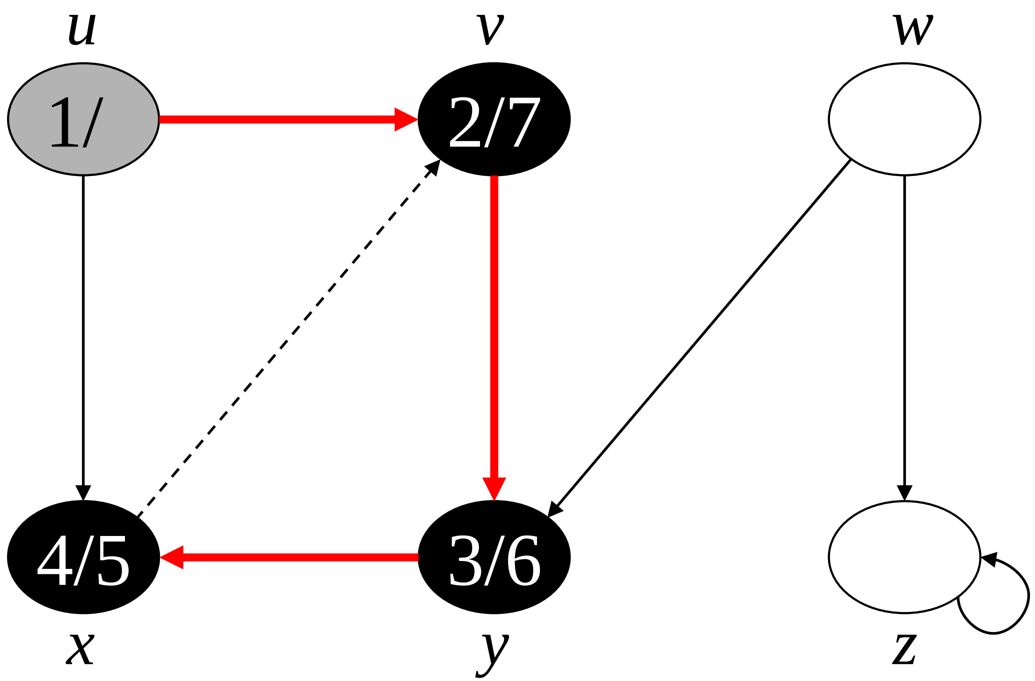
辺 (x, v) をたどる
v は既に訪問しているので行かない



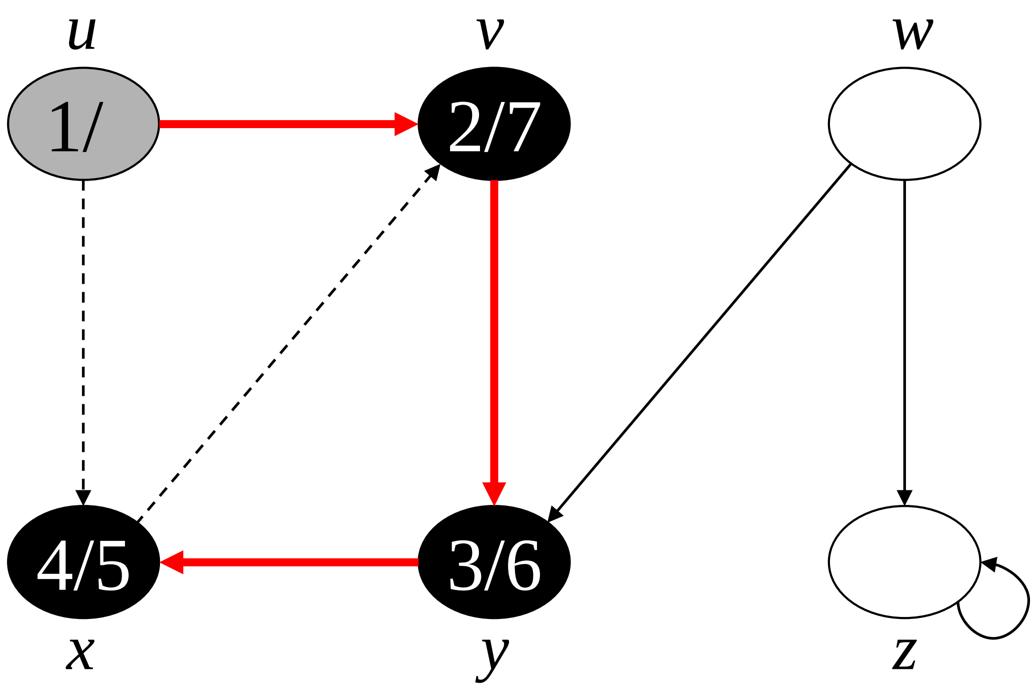
白:未訪問
灰:探索中
黒:探索終了

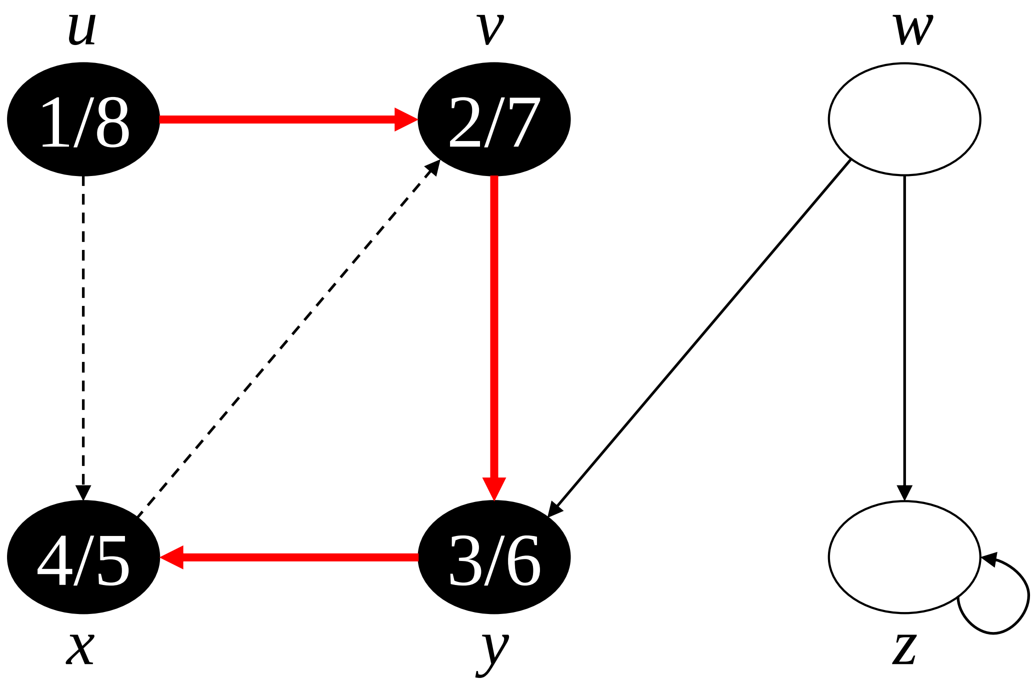


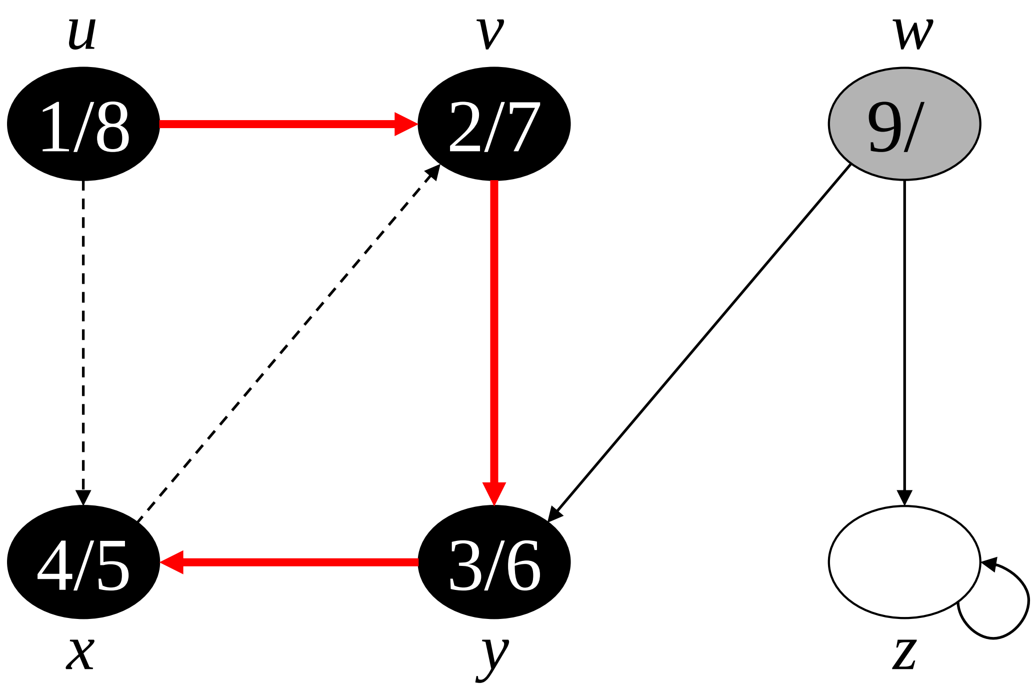

u
v
w
1/
2/
4/5
3/6
x
y
z

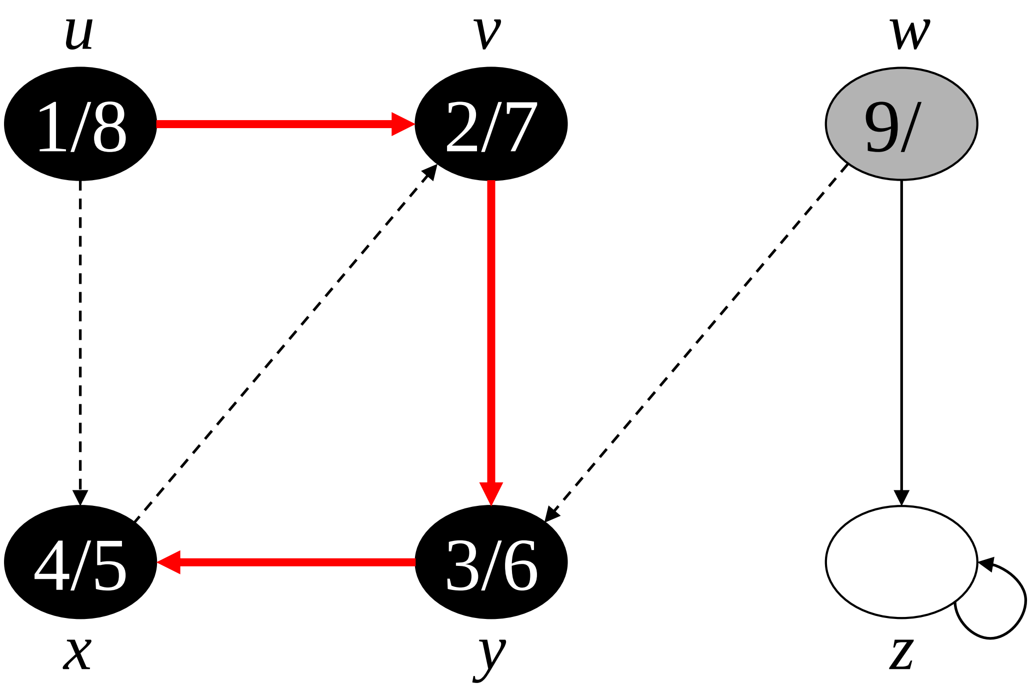

u
v
w
1/
2/7
4/5
3/6
x
y
z

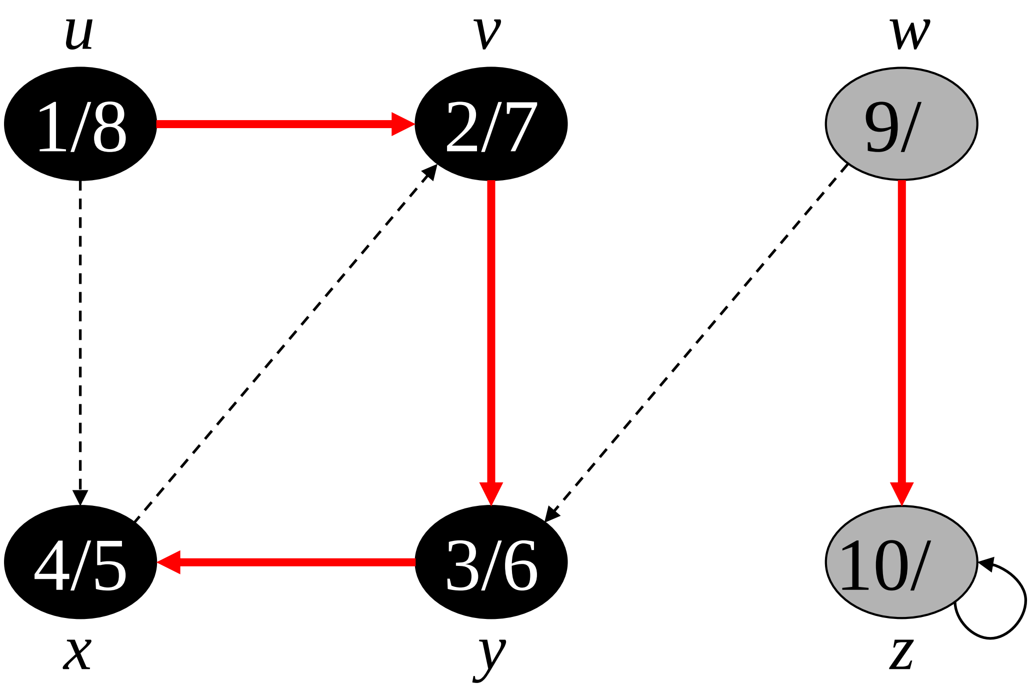
辺 (u, x) をたどる
x は既に訪問しているので行かない

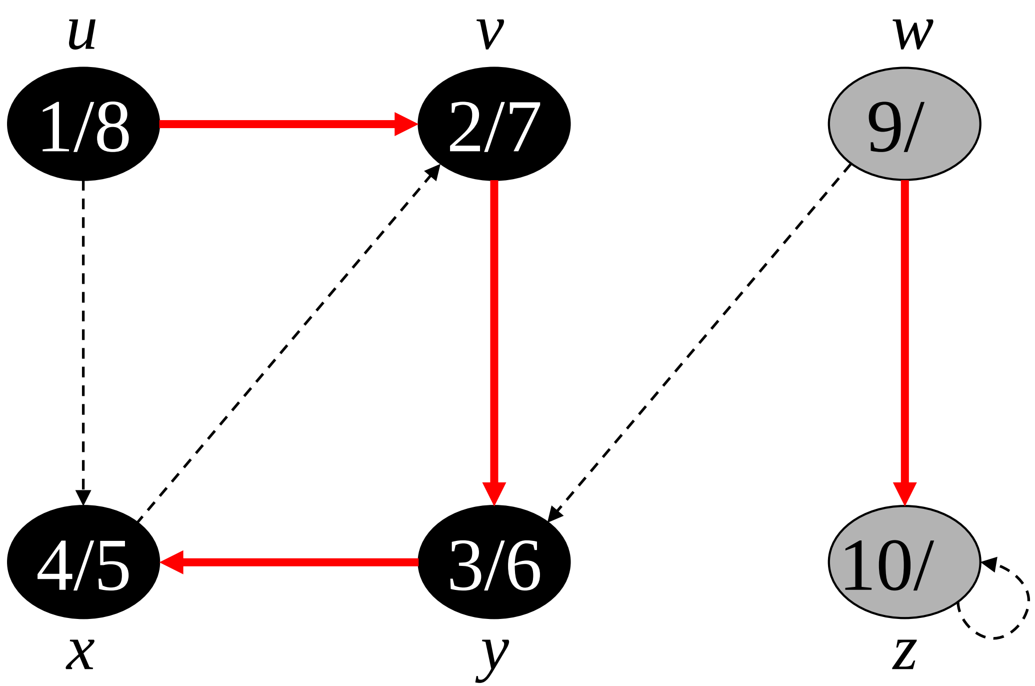


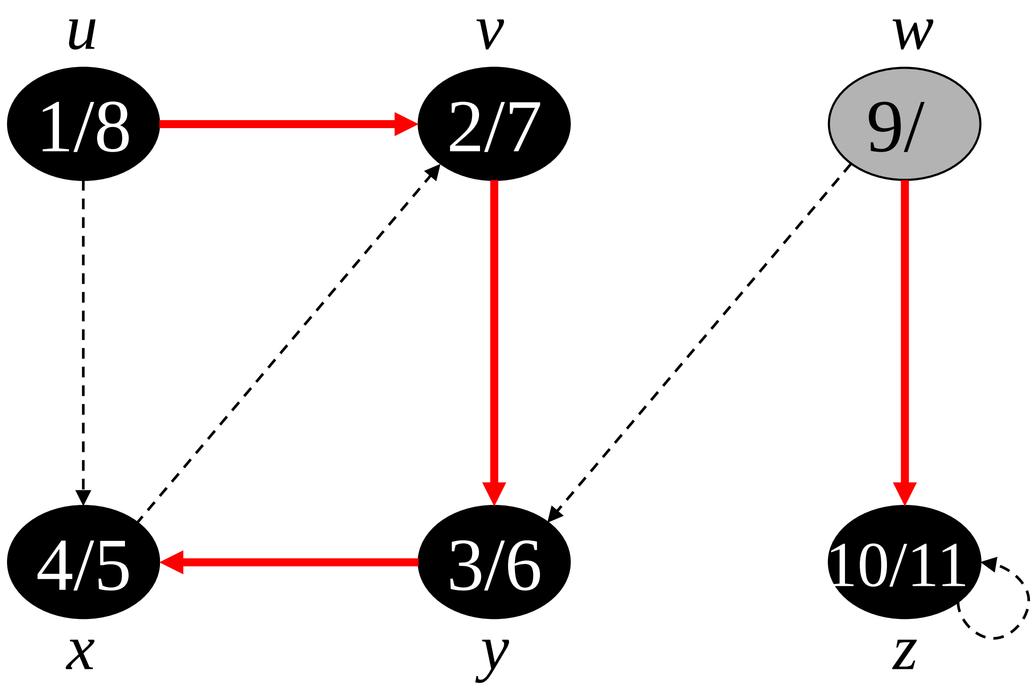

u
v
w
1/8
2/7
4/5
3/6
x
y
z

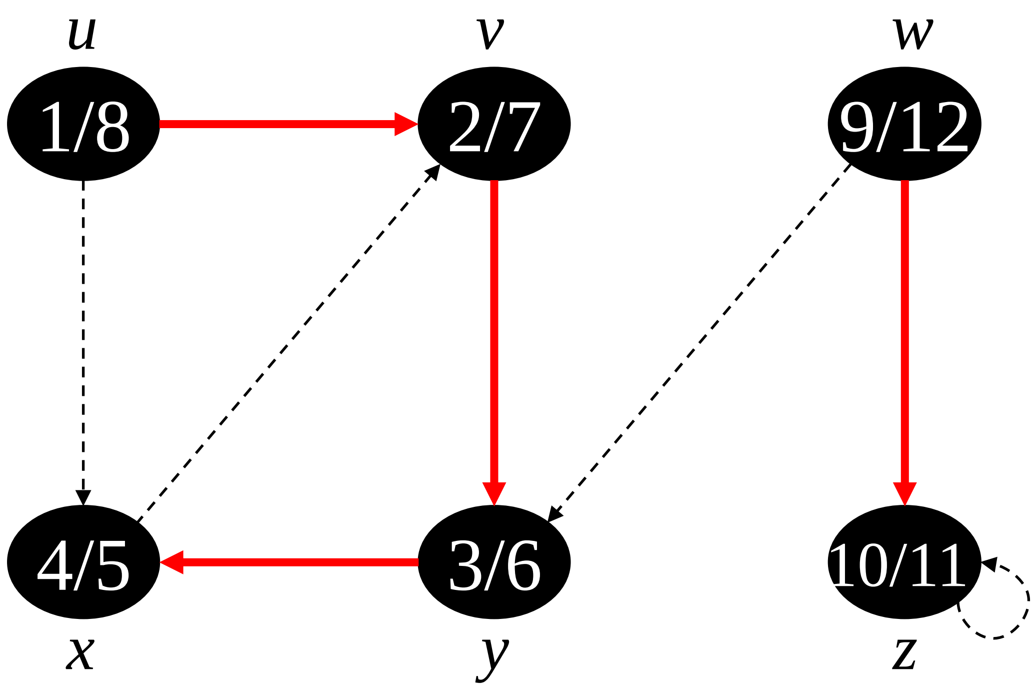

u
v
w
1/8
2/7
9/
4/5
3/6
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
10/
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
10/
x
y
z


u
v
w
1/8
2/7
9/
4/5
3/6
10/11
x
y
z


u
v
w
1/8
2/7
9/12
4/5
3/6
10/11
x
y
z

- 辺 (u, v) をたどったら，v の親を u にする $(\pi[v] = u)$
- グラフ $G_\pi = (V, E_\pi)$ を次のように定義する
  - $E_\pi = \{(\pi[v], v) \in E \mid v \in V$ かつ $\pi[v] \neq \mathrm{NIL}\}$
- G_π は深さ優先探索森 (depth-first forest)
  - 連結ならば深さ優先探索木 (depth-first tree)
- 各点は探索前は白,探索中は灰,探索終了後は黒
- 各点 v は2つのタイムスタンプを持つ
  - $d[v]$ は v を最初に発見した (灰色にした) 時刻
  - $f[v]$ は v の隣接リストを調べ終わった (黒にした) 時刻
- タイムスタンプは 1 から $2n$ の整数
- 全ての u に対し $d[u] < f[u]$
- DFSの実行時間は $\Theta(n+m)$

- 定理 (括弧付け定理):
グラフ G に対する任意の深さ優先探索を考える.
任意の2つの頂点 u, v に対し, 以下の3つの条件の中の1つだけが成立する.
  - 区間 $[d[u], f[u]]$ と区間 $[d[v], f[v]]$ には共通部分が無く, 深さ優先森において u と v はどちらも他方の子孫ではない.
  - 区間 $[d[u], f[u]]$ は区間 $[d[v], f[v]]$ に完全に含まれ, u が v の子孫となる深さ優先探索木が存在する
  - 区間 $[d[v], f[v]]$ は区間 $[d[u], f[u]]$ に完全に含まれ, v が u の子孫となる深さ優先探索木が存在する
- 注:2つの区間が部分的に重なることは無い

証明: まず $d[u] < d[v]$ の場合を考える

- $d[v] < f[u]$ のとき
  - u がまだ灰色のときに v が発見されている. つまり v は u の子孫.
  - v は u よりも後に発見されたので, u の探索が終わる前に v の探索は終わる. つまり $f[v] < f[u]$.
  - このとき区間 $[d[v], f[v]]$ は区間 $[d[u], f[u]]$に含まれる.
- $f[u] < d[v]$ のとき
  - $d[u] < f[u] < d[v] < f[v]$ なので区間に共通部分は無い.
  - つまり一方が探索中に他方が発見されない.
  - よってどちらも他方の子孫では無い
- $d[v] < d[u]$ の場合も同様に証明できる.

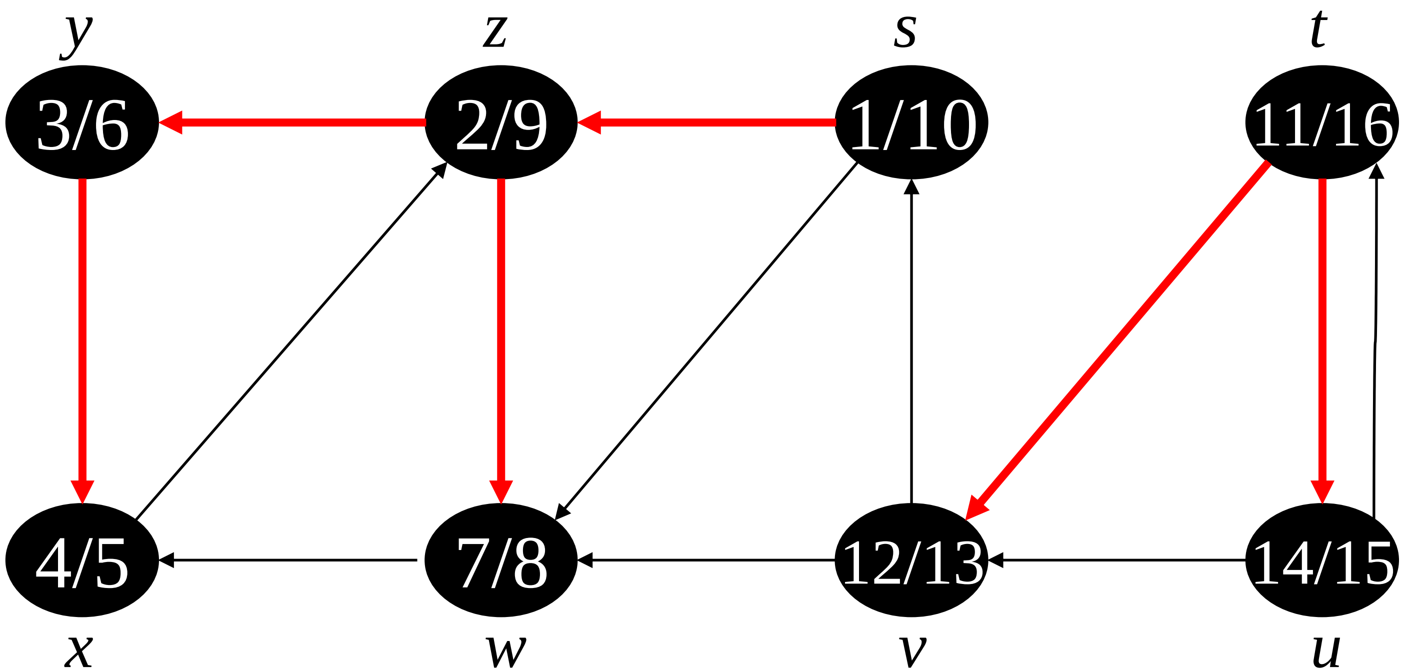
- 系 : 有向または無向グラフに対する深さ優先探索森において頂点 v が 頂点 $u\ (\neq v)$ の子孫であるための必要十分条件は

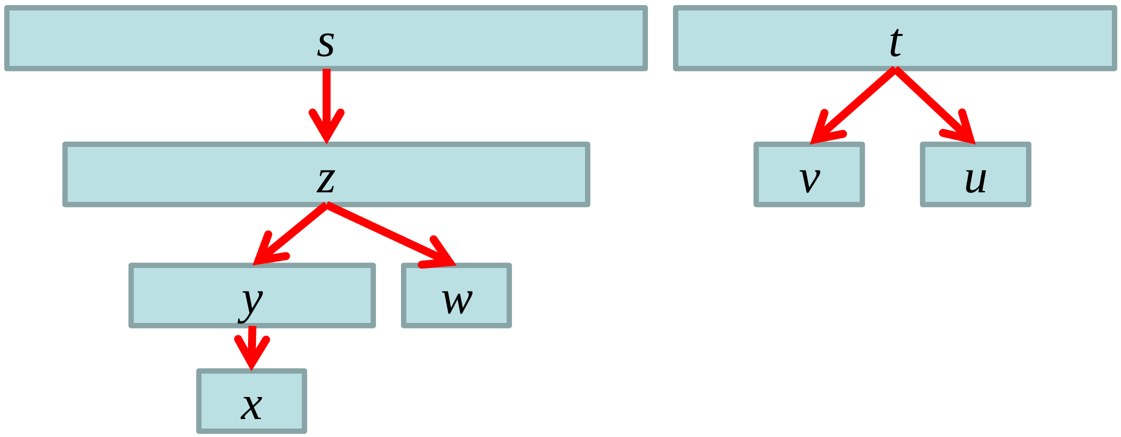
  $d[u] < d[v] < f[v] < f[u]$

- 定理 (白頂点経路定理): グラフ G の深さ優先探索森において, 頂点 v が 頂点 u の子孫であるための必要十分条件は, 探索が u を発見する時刻 $d[u]$ に u から v に至る白頂点だけからなるパスが存在することである.

- 証明：⇒) v が u の子孫であると仮定する．uv間のパス上の任意の頂点を w とする．w は u の子孫なので，$d[u] < d[w]$．つまり時刻 $d[u]$ では w は白．
- ←) 時刻 $d[u]$ に u から v に至る白頂点だけからなるパスが存在するが，深さ優先探索木において v が u の子孫にならないと仮定する．一般性を失うことなく，このパス上の v 以外の全ての頂点は u の子孫になると仮定できる (そのような点で u に一番近いものを考える)．

- このパス上で v の直前の点を w とする.
- w は u の子孫 (uを含む) なので $f[w] \le f[u]$.
- v の発見は u の後で, w の終了の前なので, $d[u] < d[v] < f[w] \le f[u]$.
- 区間は部分的には重ならないので $f[v] < f[u]$. つまり v は u の子孫.





1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

(*s* (*z* (*y* (*x* *x*) *y*) (*w* *w*) *z*) *s*) (*t* (*v* *v*) (*u* *u*) *t*)

辺の分類

- グラフ G の探索森 G_π を1つ固定する.
  G の辺を4種類に分類する

1. tree edge (木辺)
   – G_πの辺
2. back edge (後退辺)
   – 辺 (u, v) で, u と v がある探索木の頂点とその先祖の場合. セルフループも後退辺とみなす.
3. forward edge (前進辺)
   – 辺 (u, v) で, 木辺でなく, u と v がある探索木の頂点とその子孫の場合.
4. cross edge (横断辺)
   – 上の3つ以外

定理: 無向グラフ G を深さ優先探索するとき，G の任意の辺は木辺または後退辺である.

証明: (u, v) を G の任意の辺とし，一般性を失うことなく $d[u] < d[v]$ と仮定する.

v は u の隣接リストに含まれるので，u に対する処理が終わる前に v に対する処理が終わる.

辺 (u, v) が最初に u から v に向かって探索されたなら，v はその時点では白である. すると (u, v) は木辺.

辺 (u, v) が最初に v から u に向かって探索されたなら，u はその時点では灰である. すると (u, v) は後退辺.

定理:

・辺 (u,v) が木辺または前進辺

$\Leftrightarrow d[u] < d[v] < f[v] < f[u]$

・辺 (u,v) が後退辺

$\Leftrightarrow d[v] < d[u] < f[u] < f[v]$

・辺 (u,v) が横断辺

$\Leftrightarrow d[u] < f[u] < d[v] < f[v]$