アルゴリズムI

中野

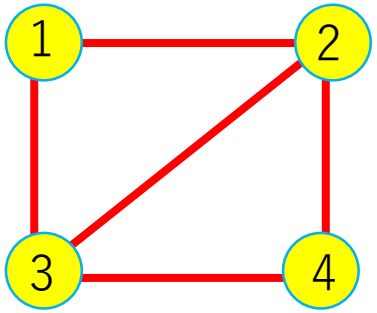
Note 4 基本グラフアルゴリズム

作成　2020.5.7

update2020.5.15 5.28 6.4 6.18 6.25

2021.5.27

グラフの表現

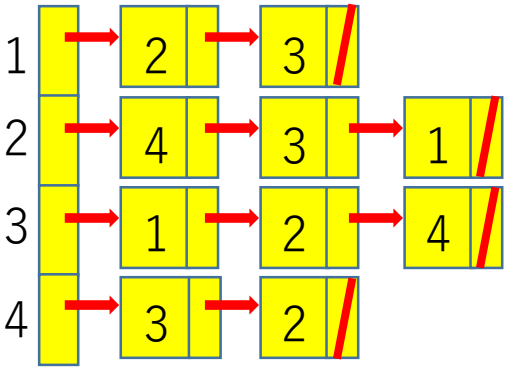


グラフ G=(V,E) の代表的なデータ構造は、2つある。
隣接リスト、と、隣接行列である。
それぞれ、有向グラフ、無向グラフのいずれも、格納できる。

有向グラフは、各辺が方向を持つグラフである。(各辺は一方通行)
無向グラフは、各辺は方向を持たないグラフである。(各辺は、両方向)

比較的、辺の本数の少ないグラフ(sparse graph)を格納するには、
隣接リストが適している。
比較的、辺の本数の多いグラフ(dense graph)を格納するには、
隣接行列が適している。

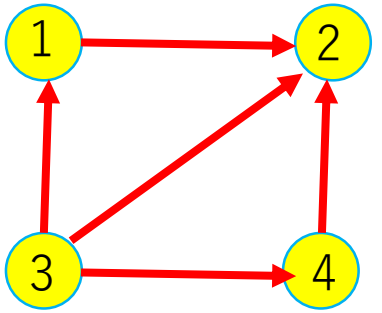
隣接リスト



隣接行列

	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

隣接リスト



隣接リストは、|V|個のリスト Adj[ ]からなる。

各点 $v_i \in V$について, Adj[i]は、点v_iに隣接する点番号のリストである。

リスト中に、点が現れる順番は、任意の順である。

(すなわち気にしない。)

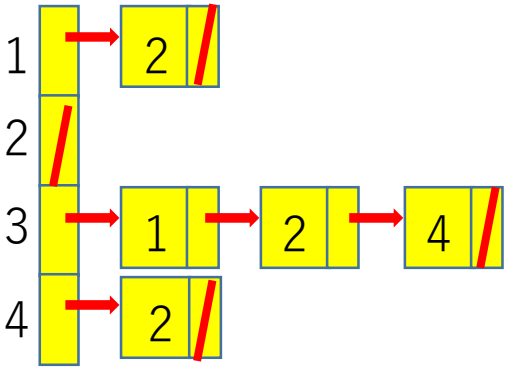
有向グラフを格納する場合には、全ての隣接リストの長さの総和は、|E|である。

(各辺に対応する情報は、ちょうど、1回、現れるので。)

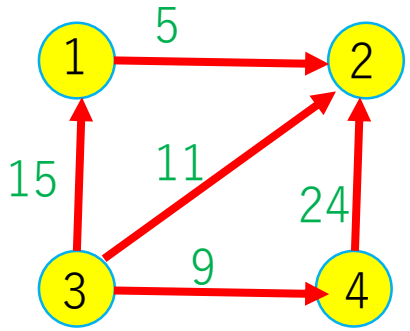
無向グラフを格納する場合には、全ての隣接リストの長さの総和は、2|E|である。

(各辺に対応する情報は、ちょうど、2回、現れるので。)

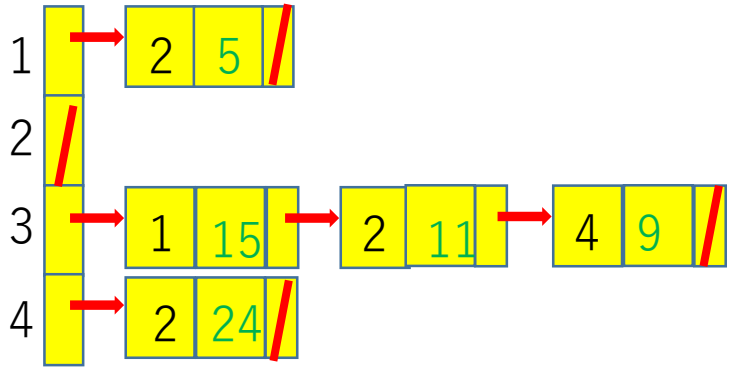
隣接リストは、O(|V| + |E|)のメモリを使用する。



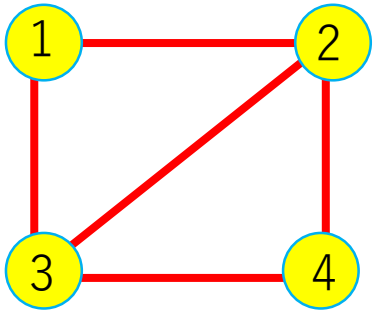
隣接リスト



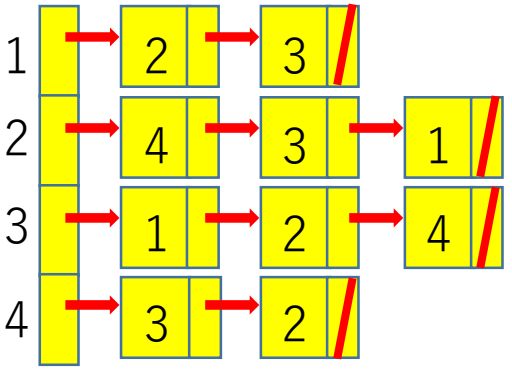
隣接リストは、各辺の重さも格納することができる。
(辺uvの重さは、点uの隣接リスト中の、
vの所に保存すればよい。)



隣接リスト

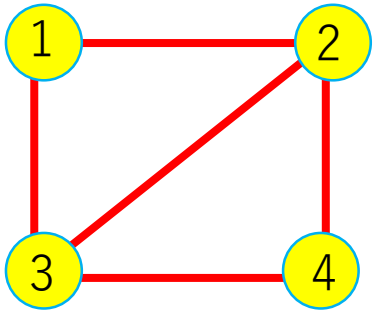


隣接リストの欠点は、2点が与えられたとき、
この間に辺があるかないかを、O(1)時間では、
判定できないことである。
(リストを走査する必要がある。O(|V|)時間かかる。)
(これに対して、隣接行列では、2点が与えられたとき、
この間に辺があるかないかを、O(1)時間で判定できる。)



(隣接リストの利点は、指定した1点に接続する辺のすべてを
高速に列挙できることである。)

隣接行列

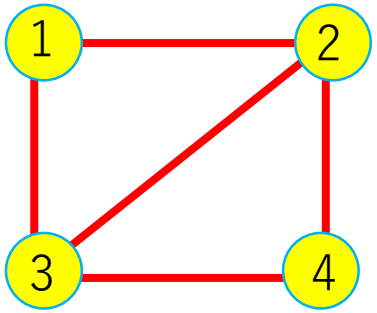


隣接行列は、|V| × |V| の2次元配列A からなる。
点には、番号がついているとする。
例えば、 $V = \{v_1, v_2, \ldots, v_{|V|}\}$とする。
この2次元配列の i行j列の要素 a_{ij}は、
0 もしくは 1 である。
点viと点vjとの間に、
辺がないならば、a_{ij} は 0 であり、
辺があるならば、a_{ij} は 1 である。

	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

隣接行列は、$O(|V|^2)$のメモリを使用する。
(辺の本数に関係ないことに注意。)

隣接行列



無向グラフの隣接行列A について、$A = A^T$ である。
よって、対角線より上の部分のみを格納すれば、
対角線より下の部分は必要ない。
(この工夫によって、使用するメモリの量を半分にすることができる。)

	1	2	3	4
1	0	1	1	0
2	1	0	1	1
3	1	1	0	1
4	0	1	1	0

隣接行列は、各辺の重さも格納することができる。
a_{ij} の値を重さにすればよい。(ただし、0は辺がないことを意味する。)

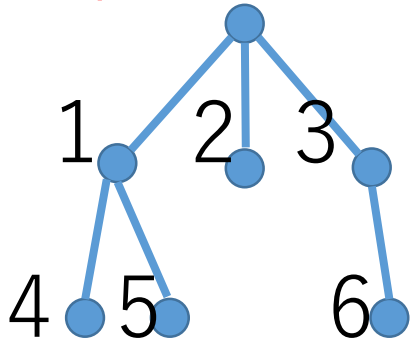
隣接行列は、構造が簡単なので、点の個数が少ないときは、
隣接リストよりも、使用されることが非常に多い。
(ただし、点の個数が大きいときは、検討が必要です。)
また、辺の重さがないときは、(wordの行列ではなくて、) bitの行列にすることで
メモリを節約できる。

グラフの探索

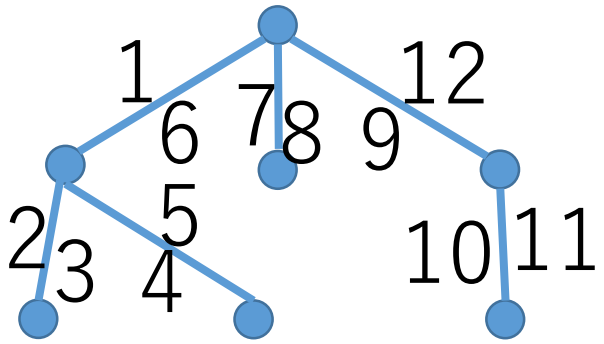
グラフの探索とは、
グラフの全ての点や辺を探すことである。
グラフの探索は、
様々なアルゴリズムの基本となるものである。

基本的なグラフの探索には、
BFS(Breadth-first search)(幅優先探索)と
DFS(Depth-first search)(深さ優先探索)の
2つがある。
BFSは、最短路を求めるダイキストラ法、
最小全域木を求めるプリム法の基本である。

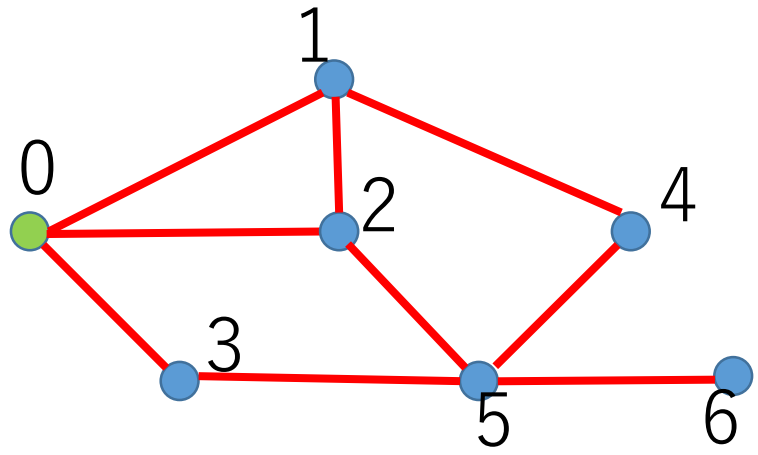
木のBFS



木のDFS



グラフの探索その１　BFS



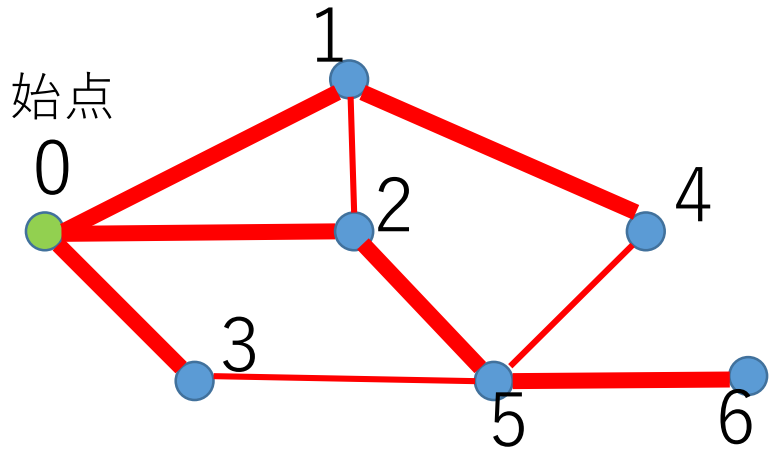
BFSは、グラフ G=(V,E) と Gの1点sが始点として与えられたとき、
Gの辺を調べ、sから到達可能な(辺をたどって到着できる)すべての点を発見する。
また、始点sから、それぞれの到達可能な点への距離(間の辺の最小数)も計算する。
BFSはsから距離1,2,3,...の点を、距離が小さいものから順に発見する。

点u から、辺(u, v) をたどって、点vを発見したとき、点uを点vの親とする。
これを $\pi[v] = u$ と書こう。
辺集合 $\{ (v, \pi[v]) \mid v \in V - \{s\} \}$ は木である。これを、始点sを根とする幅優先木Tと呼ぶ。
(Tの各点vからsへの道が、Gにおけるvからsへの最短路となる。)
(最短路とは、2点を結ぶ、辺の本数が最小の道である。)

BSFは、各点に、次のように色を塗ると考えよう。
各点は、最初は白色であり、途中で灰色になり、最後に黒色となる。
黒色の点の隣接点は、黒色もしくは、灰色のいずれかである。

BFSアルゴリズムの実行中に、点は、ドーナツ状の3グループに分割される。
始点に近い黒の部分、その周りの灰色の部分、その周りの白の部分である。

グラフの探索その１　BFS



BFSは、グラフ G=(V,E) と Gの1点sが始点として与えられたとき、
Gの辺を調べ、sから到達可能な(辺をたどって到着できる)すべての点を発見する。
また、始点sから、それぞれの到達可能な点への距離(間の辺の最小数)も計算する。
BFSはsから距離1,2,3,...の点を、距離が小さいものから順に発見する。

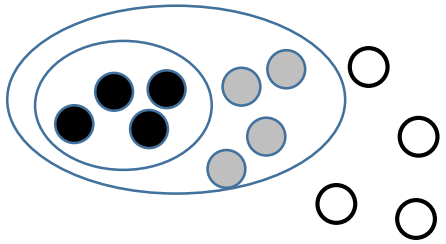
点u から、辺(u, v) をたどって、点vを発見したとき、点uを点vの親とする。
これを $\pi[v] = u$ と書こう。
辺集合 $\{ (v, \pi[v]) \mid v \in V - \{s\} \}$ は木である。これを、始点sを根とする幅優先木Tと呼ぶ。
(Tの各点vからsへの道が、Gにおけるvからsへの最短路となる。)
(最短路とは、2点を結ぶ、辺の本数が最小の道である。)

入口　　キュー　　出口

BSFは、各点に、次のように色を塗ると考えよう。
各点は、最初は白色であり、途中で灰色になり、最後に黒色となる。
黒色の点の隣接点は、黒色もしくは、灰色のいずれかである。

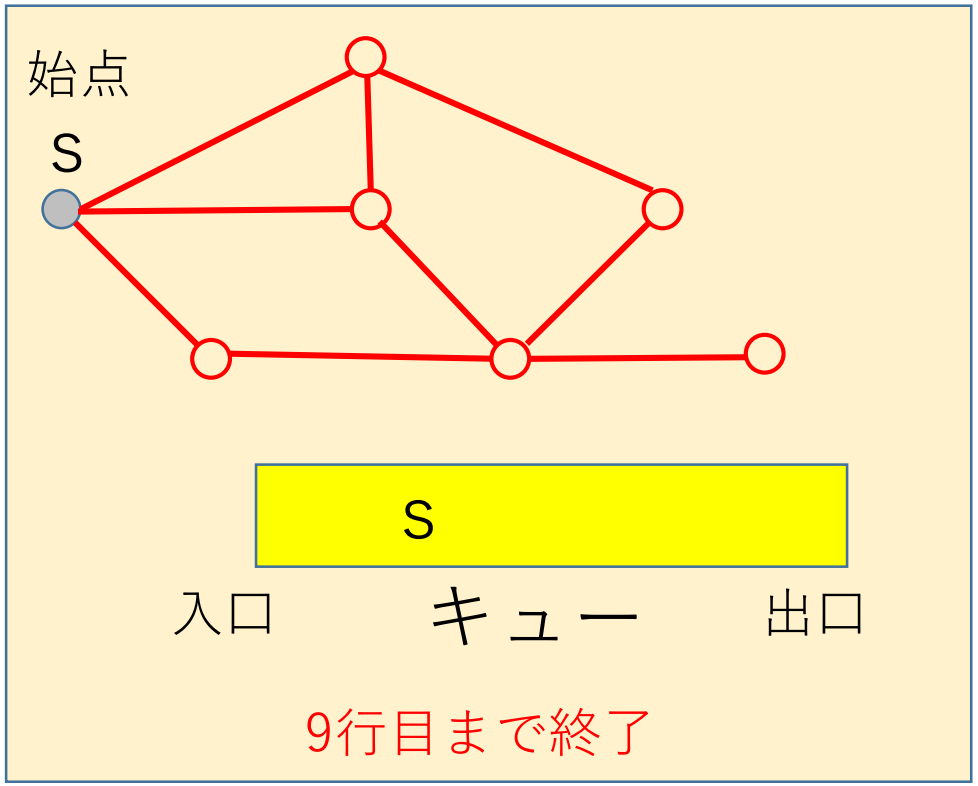
白	未発見
灰色	処理中
黒	処理ずみ

BFSアルゴリズムの実行中に、点は、ドーナツ状の3グループに分割される。
始点に近い黒の部分、その周りの灰色の部分、その周りの白の部分である。



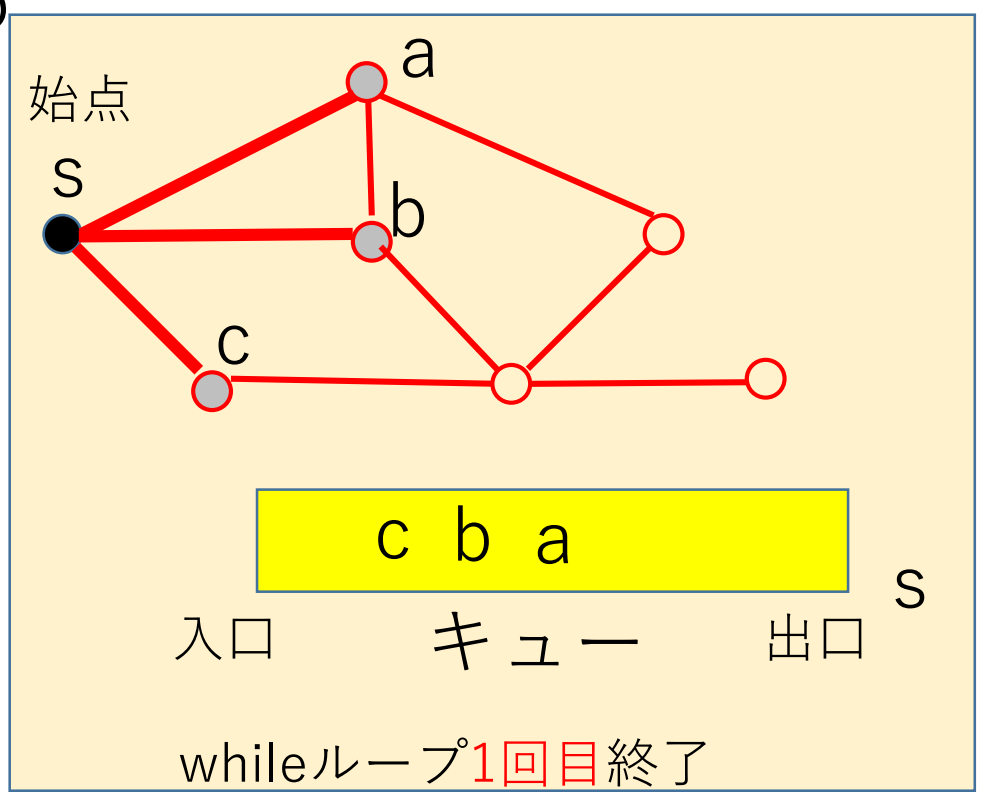
グラフの探索その１　BFS

```
BFS(G, s)
1   for 各点 u ∈ V - { s } do          /* 初期化 */
2       color[u] ← 白色
3       d[u] ← ∞        /* d[u] は始点から点uまでの距離である */
4       π[u] ← NIL      /* π[u]は幅優先木における 点uの親である */
5   color[s] ← 灰色     /*   始点をセット    */
6   d[s] ← 0
7   π[s] ← NIL
8  Q ← φ
9  EnQUEUE(Q.s)         /* キューQ に灰色の点sを格納する。*/

10  while Q ≠ φ do     /* 1点ずつ探索する    */
11      u ← DeQUEUE(Q)        /* これから点u のまわりをチェックする    */
12      for each v ∈ Adj[u] do   /* Adj[u]は点uの隣接リスト */
13          if color[v] = 白 then   /* 点vが新しく発見した点なら    */
14                  color[v] ← 灰色
15                  d[v] ← d[u] +1
16                  π[v] ← u
17                  EnQUEUE(Q.v) /* 点v はあとで処理する    */
18      color[u] ← 黒色
```



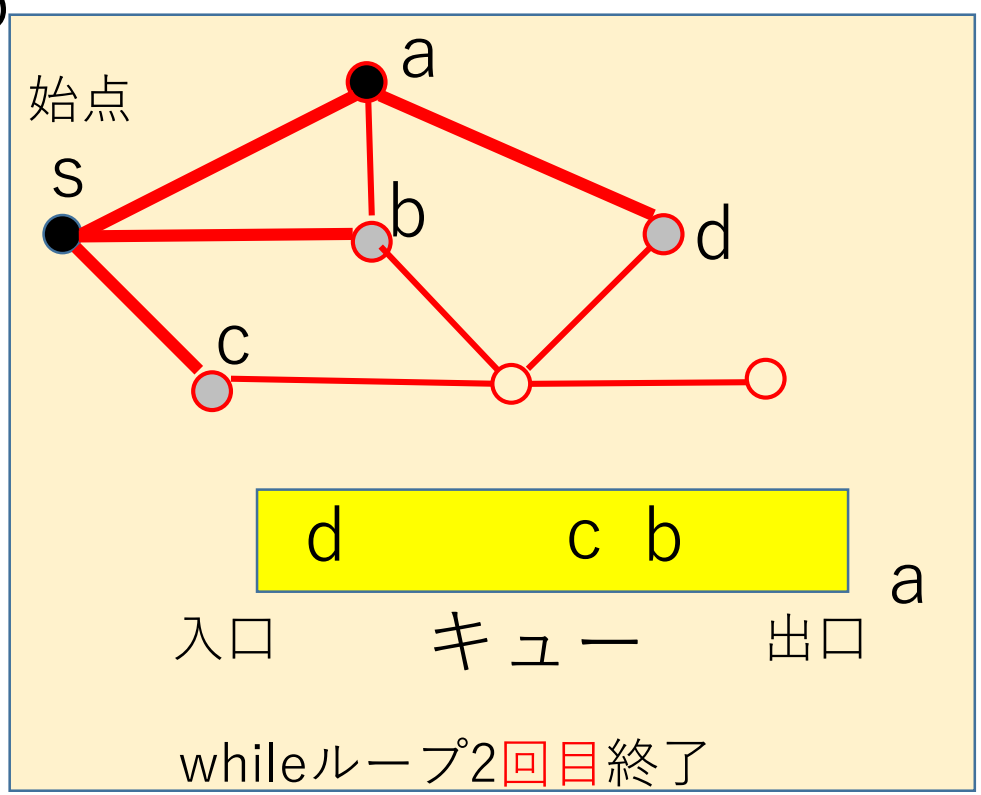
グラフの探索その１　BFS

```
BFS(G, s)
1   For 各点 u ∈ V – { s } do          /* 初期化 */
2       color[u] ← 白色
3       d[u] ← ∞        /* d[u] は始点から点uまでの距離である */
4       π[u] ← NIL      /* π[u]は幅優先木における 点uの親である */
5   color[s] ← 灰色     /*   始点をセット    */
6   d[s] ← 0
7   π[s] ← NIL
8  Q ← φ
9  EnQUEUE(Q.s)        /* キューQ に灰色の点sを格納する。*/

10     while Q ≠ φ do     /* 1点ずつ探索する    */
11      u ← DeQUEUE(Q)       /* これから点u のまわりをチェックする    */
12      for each v ∈ Adj[u] do   /* Adj[u]は点uの隣接リスト */
13          if color[v] = 白 then   /* 点vが新しく発見した点なら    */
14                  color[v] ← 灰色
15                  d[v] ← d[u] +1
16                  π[v] ← u
17                  EnQUEUE(Q.v) /* 点v はあとで処理する    */
18      color[u] ← 黒色
```



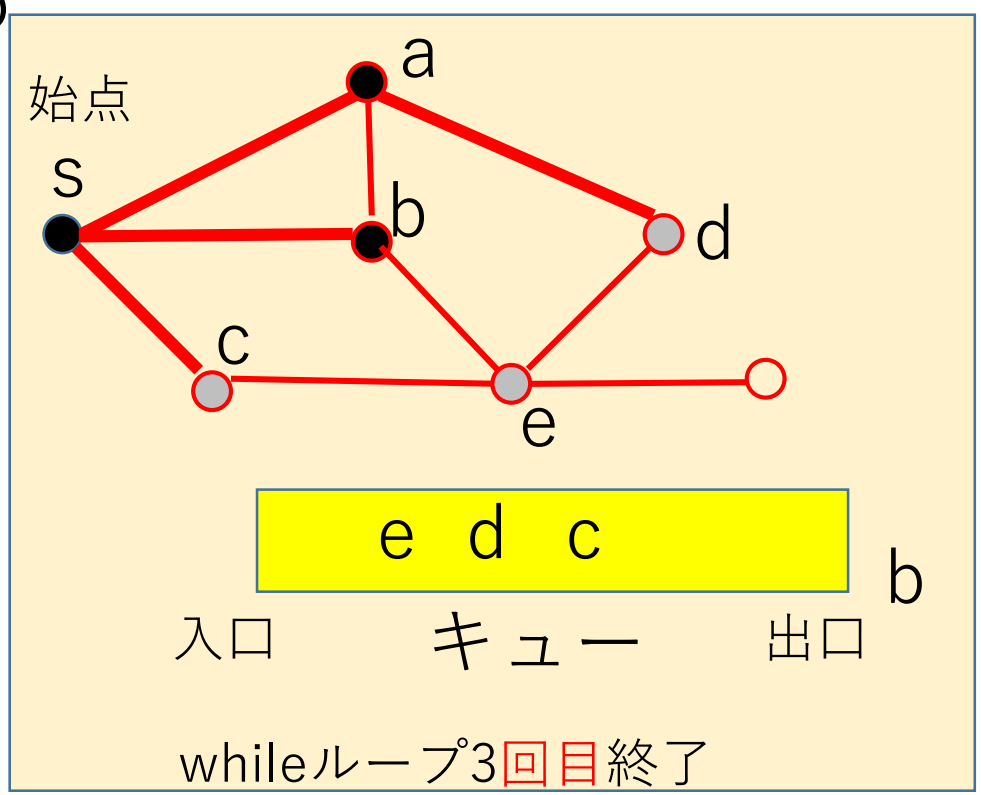
グラフの探索その１　BFS

```
BFS(G, s)
1   For 各点 u ∈ V − { s } do          /* 初期化 */
2       color[u] ← 白色
3       d[u] ← ∞        /* d[u] は始点から点uまでの距離である */
4       π[u] ← NIL      /* π[u]は幅優先木における 点uの親である */
5   color[s] ← 灰色     /*   始点をセット    */
6   d[s] ← 0
7   π[s] ← NIL
8  Q ← φ
9  EnQUEUE(Q.s)        /* キューQ に灰色の点sを格納する。*/

10     while Q ≠ φ do     /* 1点ずつ探索する    */
11      u ← DeQUEUE(Q)       /* これから点u のまわりをチェックする    */
12      for each v ∈ Adj[u] do   /* Adj[u]は点uの隣接リスト */
13          if color[v] = 白 then   /* 点vが新しく発見した点なら    */
14                  color[v] ← 灰色
15                  d[v] ← d[u] +1
16                  π[v] ← u
17                  EnQUEUE(Q.v) /* 点v はあとで処理する    */
18      color[u] ← 黒色
```



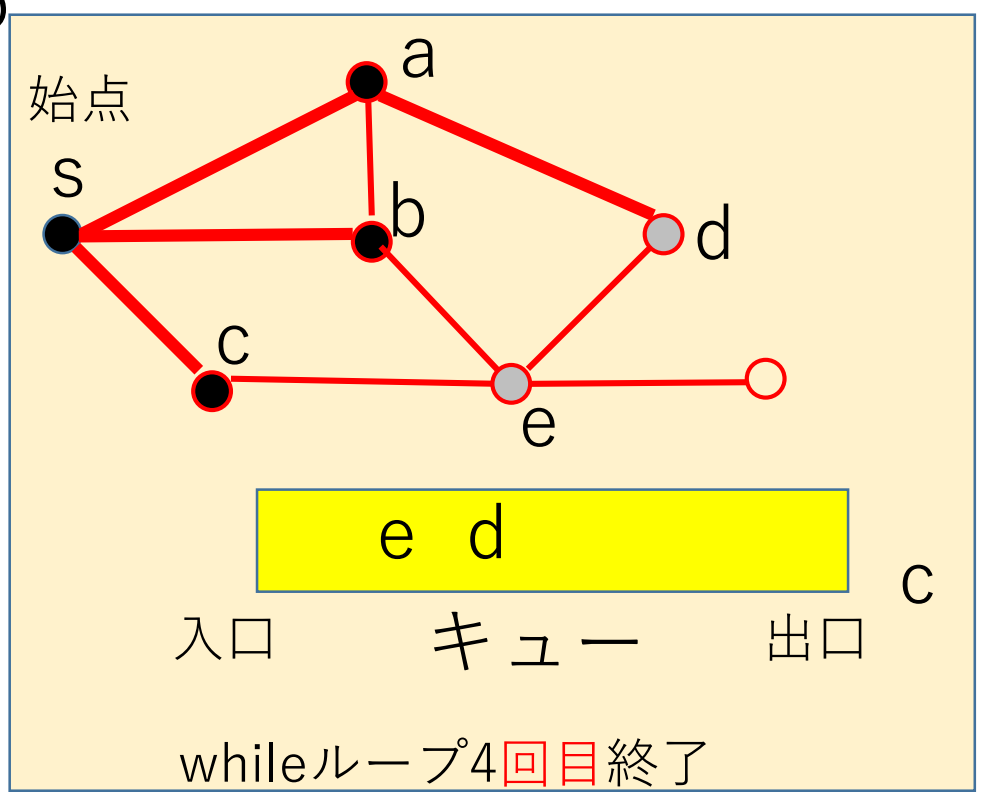
グラフの探索その１　BFS

```
BFS(G, s)
1   For 各点 u ∈ V − { s } do          /* 初期化 */
2       color[u] ← 白色
3       d[u] ← ∞        /* d[u] は始点から点uまでの距離である */
4       π[u] ← NIL      /* π[u]は幅優先木における 点uの親である */
5   color[s] ← 灰色     /*   始点をセット    */
6   d[s] ← 0
7   π[s] ← NIL
8  Q ← φ
9  EnQUEUE(Q.s)        /* キューQ に灰色の点sを格納する。*/

10     while Q ≠ φ do     /* 1点ずつ探索する    */
11      u ← DeQUEUE(Q)       /* これから点u のまわりをチェックする    */
12      for each v ∈ Adj[u] do   /* Adj[u]は点uの隣接リスト */
13          if color[v] = 白 then   /* 点vが新しく発見した点なら    */
14                  color[v] ← 灰色
15                  d[v] ← d[u] +1
16                  π[v] ← u
17                  EnQUEUE(Q.v) /* 点v はあとで処理する    */
18      color[u] ← 黒色
```



グラフの探索その１　BFS

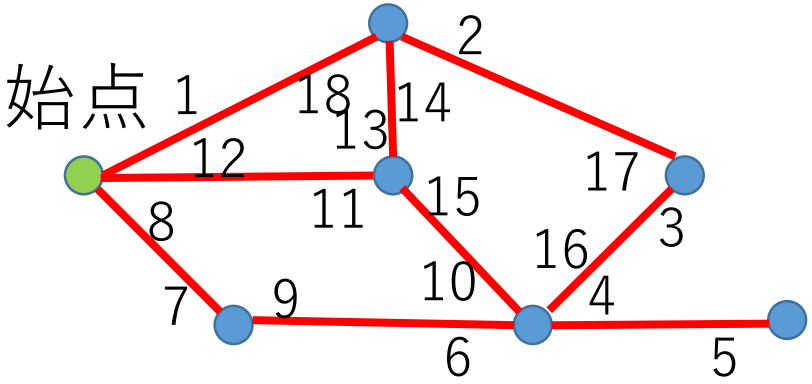
```
BFS(G, s)
1   For 各点 u ∈ V − { s } do          /* 初期化 */
2       color[u] ← 白色
3       d[u] ← ∞        /* d[u] は始点から点uまでの距離である */
4       π[u] ← NIL      /* π[u]は幅優先木における 点uの親である */
5   color[s] ← 灰色     /*   始点をセット    */
6   d[s] ← 0
7   π[s] ← NIL
8  Q ← φ
9  EnQUEUE(Q.s)        /* キューQ に灰色の点sを格納する。*/

10     while Q ≠ φ do     /* 1点ずつ探索する    */
11      u ← DeQUEUE(Q)       /* これから点u のまわりをチェックする    */
12      for each v ∈ Adj[u] do   /* Adj[u]は点uの隣接リスト */
13          if color[v] = 白 then   /* 点vが新しく発見した点なら    */
14                  color[v] ← 灰色
15                  d[v] ← d[u] +1
16                  π[v] ← u
17                  EnQUEUE(Q.v) /* 点v はあとで処理する    */
18      color[u] ← 黒色
```



whileループあと3回で終了

計算時間は O(|V| + |E|) である。

グラフの探索その2　DFS



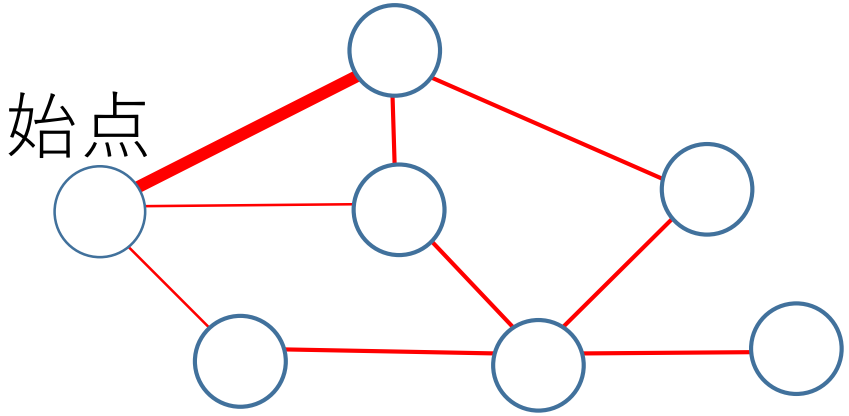
DFSは、グラフの"深い"部分を優先して探索する。すわわち、
もし、現在注目している点に、まだ探索していない辺が接続していれば、
その辺を探索する。ただし、たどりついた点が未探索でないときはすぐに引き返す。
もし、現在注目している点に、探索していない辺が接続していなければ、
(探索終了していれば)、その点を発見したときに探索した辺をたどって、ひき返す。
以上により、始点から到達可能なすべての点を発見し、最後に始点に戻る。

点u から、辺(u, v) をたどって、点vを発見したとき、点uを点vの親とする。これを $\pi[v] = u$ と書こう。
辺集合 $\{ (v, \pi[v]) \mid v \in V - \{s\} \}$ は木である。これを、始点sを根とする深さ優先木Tと呼ぶ。
(Tの各点vからsへの道が、Gにおけるvからsへの最短路とは、必ずしもならないことに注意。)

DFSは、各点に、次のように色を塗ると考えよう。
各点は、最初は白色であり、途中で発見されると灰色になり、
接続する辺の探索が終了すると黒色となる。
黒色の点vの隣接点のうち、$\pi[v]$は灰色or黒色であり、他の点は必ず黒色である。

DFSは、各点vに、2つのタイムスタンプ d[v] と f[v] を割り当てると考えよう。
d[v] は 点vが発見され、vが灰色になった時刻である。
f[v] は 点vに接続する全ての辺の探索が終了し、vが黒色になった時刻である。
任意の点vにおいて、d[v] < f[v] となる。

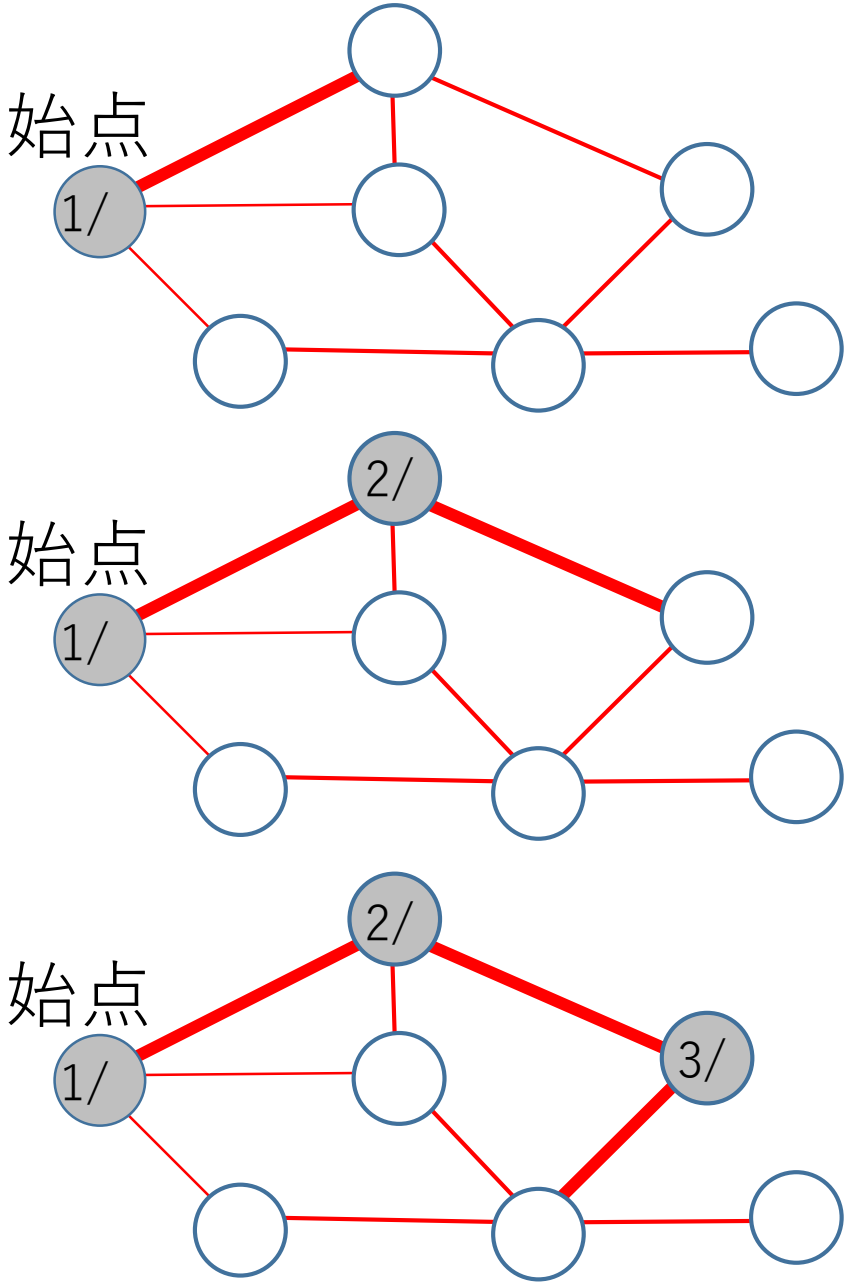
グラフの探索その2　DFS




```
DFS(G)
1   for 各点 u ∈ V[G] do   /*  初期化   */
2       color[u] ← 白色
3       π[u] ← NIL      /*  π[u]は深さ優先木における 点uの親である */
4   time ← 0
5   for 各点 u ∈ V[G] do     /* グラフが非連結の時のためのしかけ  */
6       if   点 u は白色
7       then  DFS-Visit(u)
```

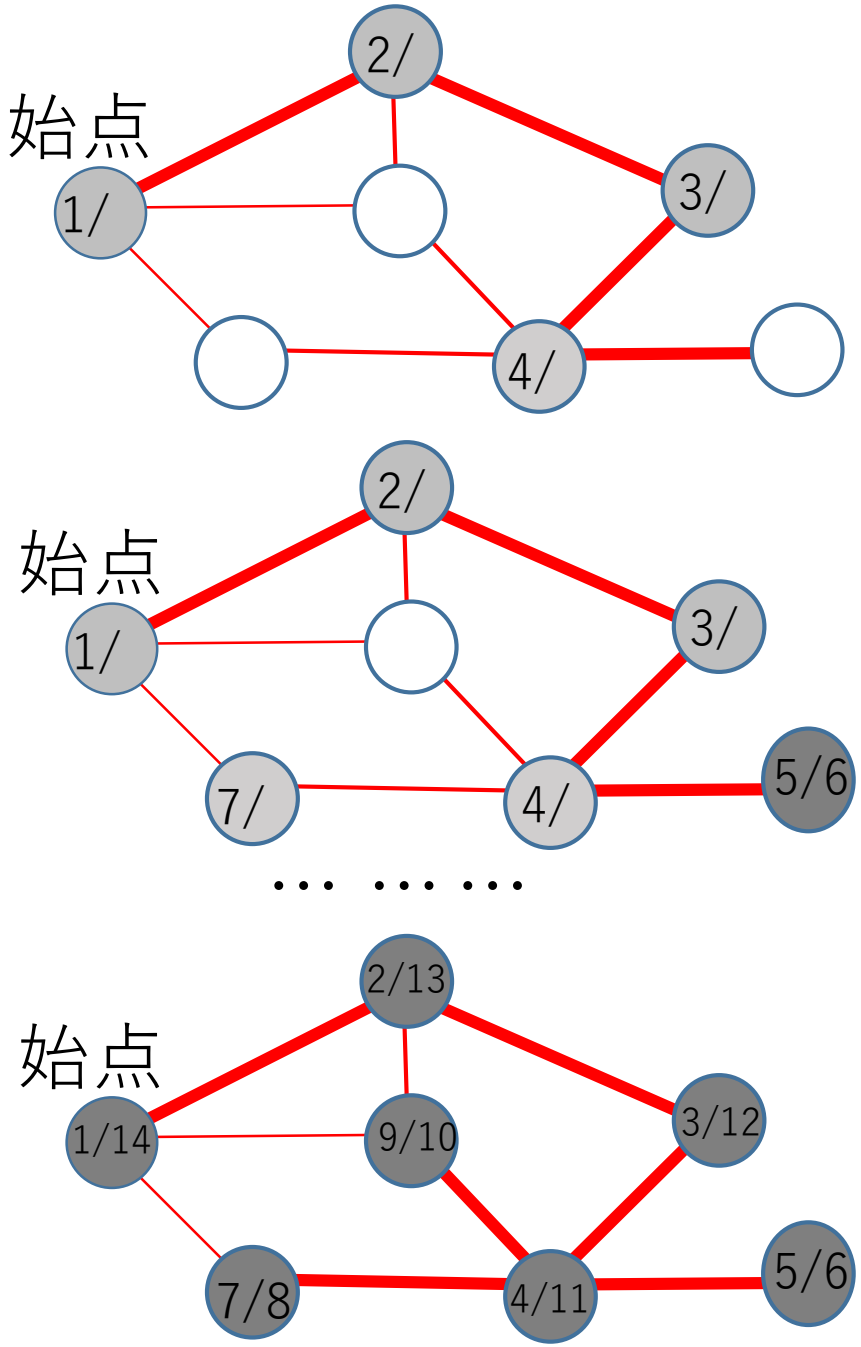
グラフの探索その2　DFS

```
DFS-Visit(u)
{始点をセットする}
1   color[u] ← 灰色        /* 点 u の処理開始  */
2   time ← time + 1
3   d[u] ← time
4   for 各点 v ∈ Adj[u] do     /* 辺 (u,v)を探索する */
6     if color[v] = 白 then     /* 点 v は新発見ならば  */
8         π [v] = u
9          DFS-Visit(v)          /* 先にすすむ   */
12  color[u] ← 黒色                  { 点 u の処理終了 }
13  time ← time + 1
14  f[u] ← time
end
```



グラフの探索その2　DFS

```
DFS-Visit(u)
{始点をセットする}
1   color[u] ← 灰色        /* 点 u の処理開始  */
2   time ← time + 1
3   d[u] ← time
4   for 各点 v ∈ Adj[u] do     /* 辺 (u,v)を探索する */
6     if color[v] = 白 then     /* 点 v は新発見ならば  */
8         π [v] = u
9          DFS-Visit(v)          /* 先にすすむ   */
12  color[u] ← 黒色                  { 点 u の処理終了 }
13  time ← time + 1
14  f[u] ← time
end
```

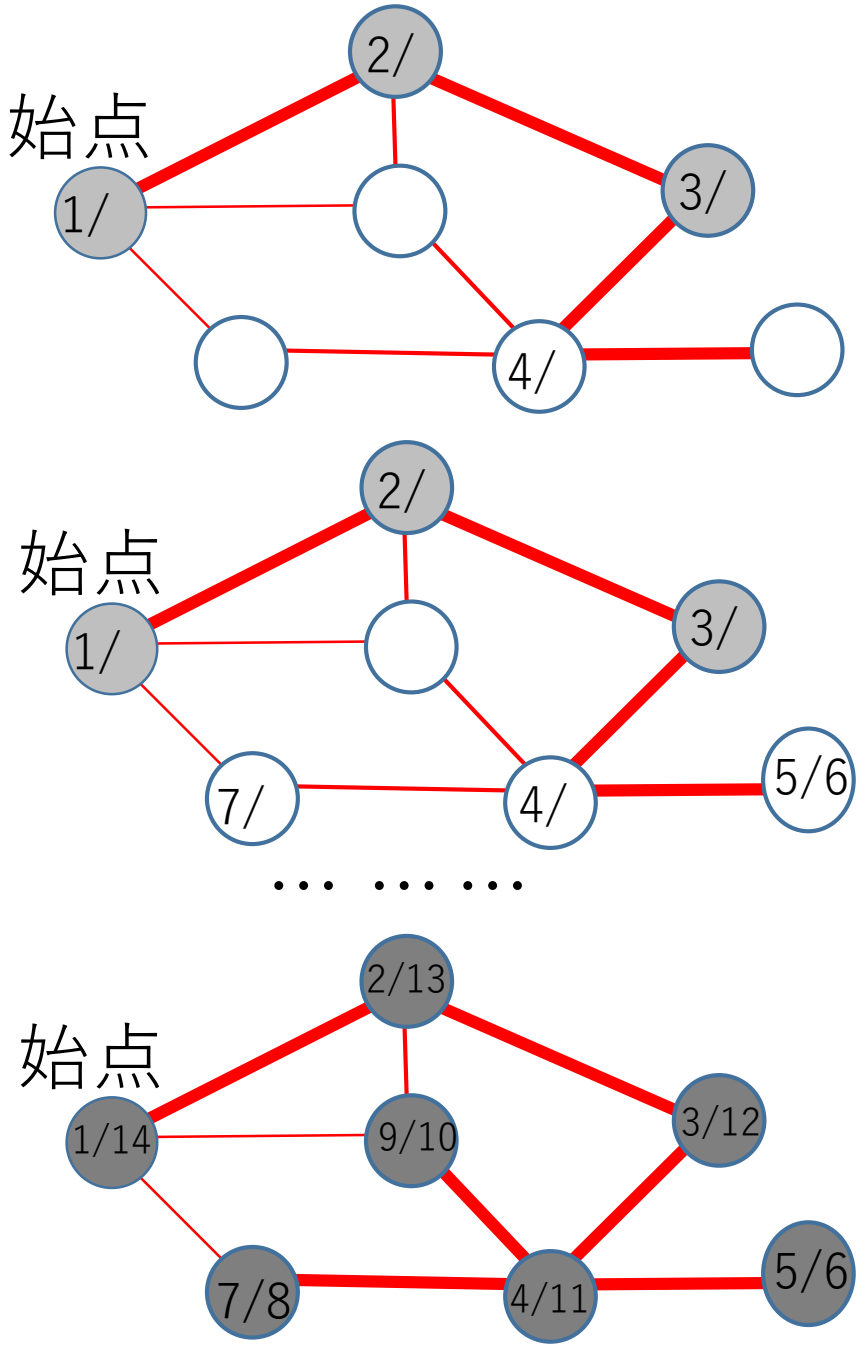


グラフの探索その2　DFS

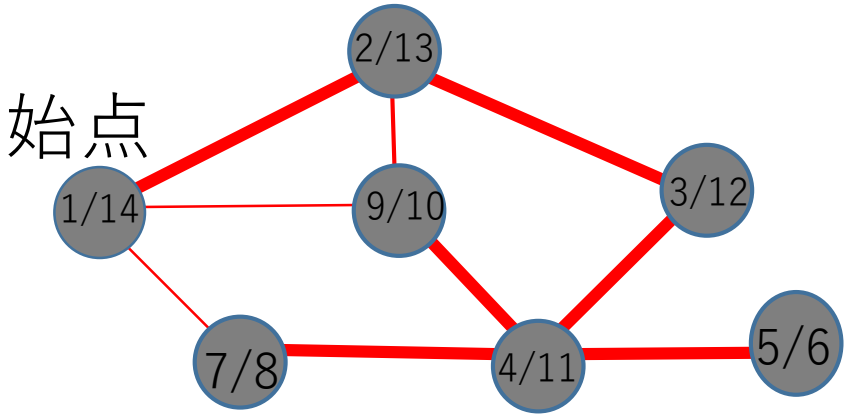
深さ優先木は、再帰呼び出しの構造、そのものに対応する。
すなわち、 u = π[v] ならば、
DFS-Visit(u) 中に、DFS-Visit(v)が呼び出されたことを意味する。
また、2つのタイムスタンプ d[] と f[] は、parenthesis structureとなる。
(d[u] < d[v] < f[v] < f[u]となる。)
すなわち、2組のカッコは、独立 ( ) [ ] , もしくは、
包含([ ]) の関係のみであり、交差 ([ )] はない!

特に無向グラフ G を、深さ優先探索したとき、G の各辺は、
(1) (tree edge)深さ優先木Tに含まれる辺
(2) (back edge)ある点から、その点の(Tにおける)先祖に向かう辺
のいずれかとなる。

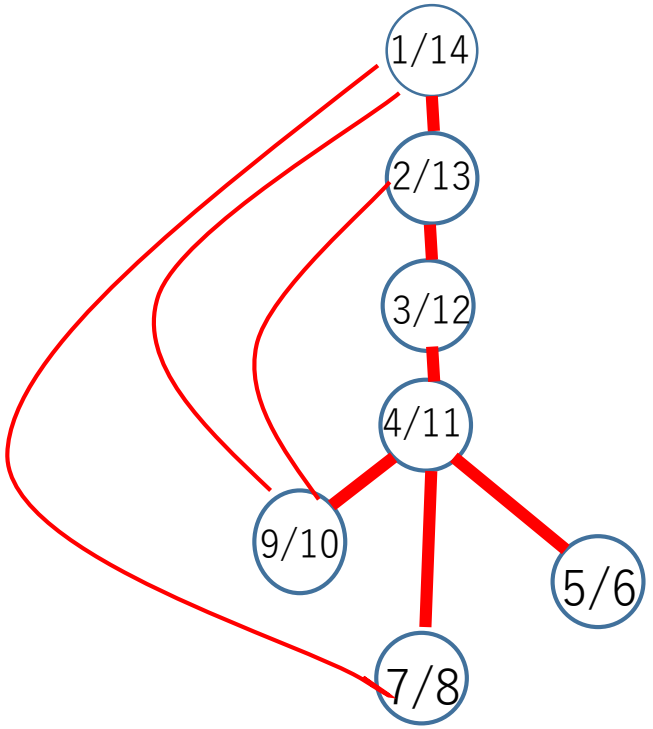
計算時間は O(|V| + |E|) である。



グラフの探索その2　DFS



深さ優先木は、再帰呼び出しの構造、そのものに対応する。
すなわち、 u = π[v] ならば、
DFS-Visit(u) 中に、DFS-Visit(v)が呼び出されたことを意味する。
また、2つのタイムスタンプ d[] と f[] は、parenthesis structureとなる。
(d[u] < d[v] < f[v] < f[u]となる。)
すなわち、2組のカッコは、独立 ( ) [ ] , もしくは、
包含([ ]) の関係のみであり、交差 ([ )] はない!



特に無向グラフ G を、深さ優先探索したとき、G の各辺は、
(1) (tree edge)深さ優先木Tに含まれる辺
(2) (back edge)ある点から、その点の(Tにおける)先祖に向かう辺
のいずれかとなる。

計算時間は O(|V| + |E|) である。

トポロジカルソート

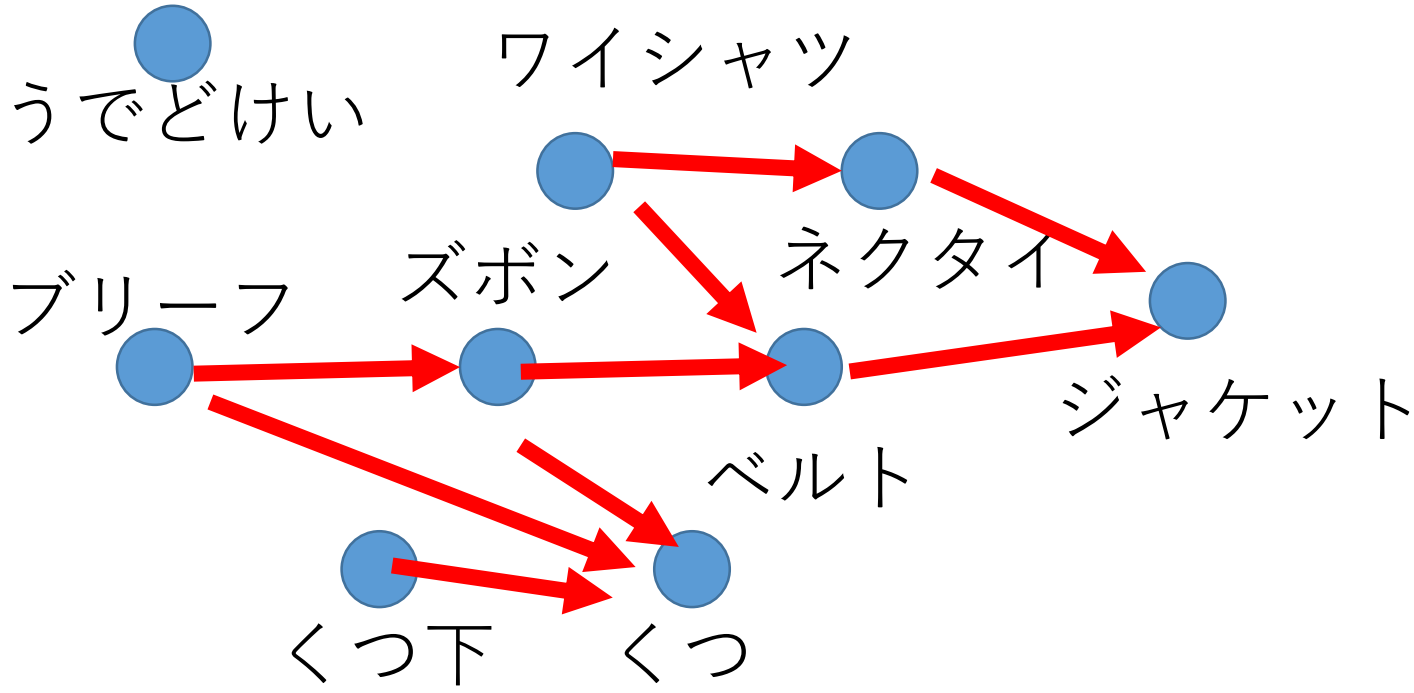
入力:　有向グラフ G=(V,E) ただし、有向サイクルを含まないとする。
（すなわち半順序）

出力:　Vの全順序 "<" で、Gの任意の有向辺 (u,v) について　u < v となるもの。
（すなわち全順序）

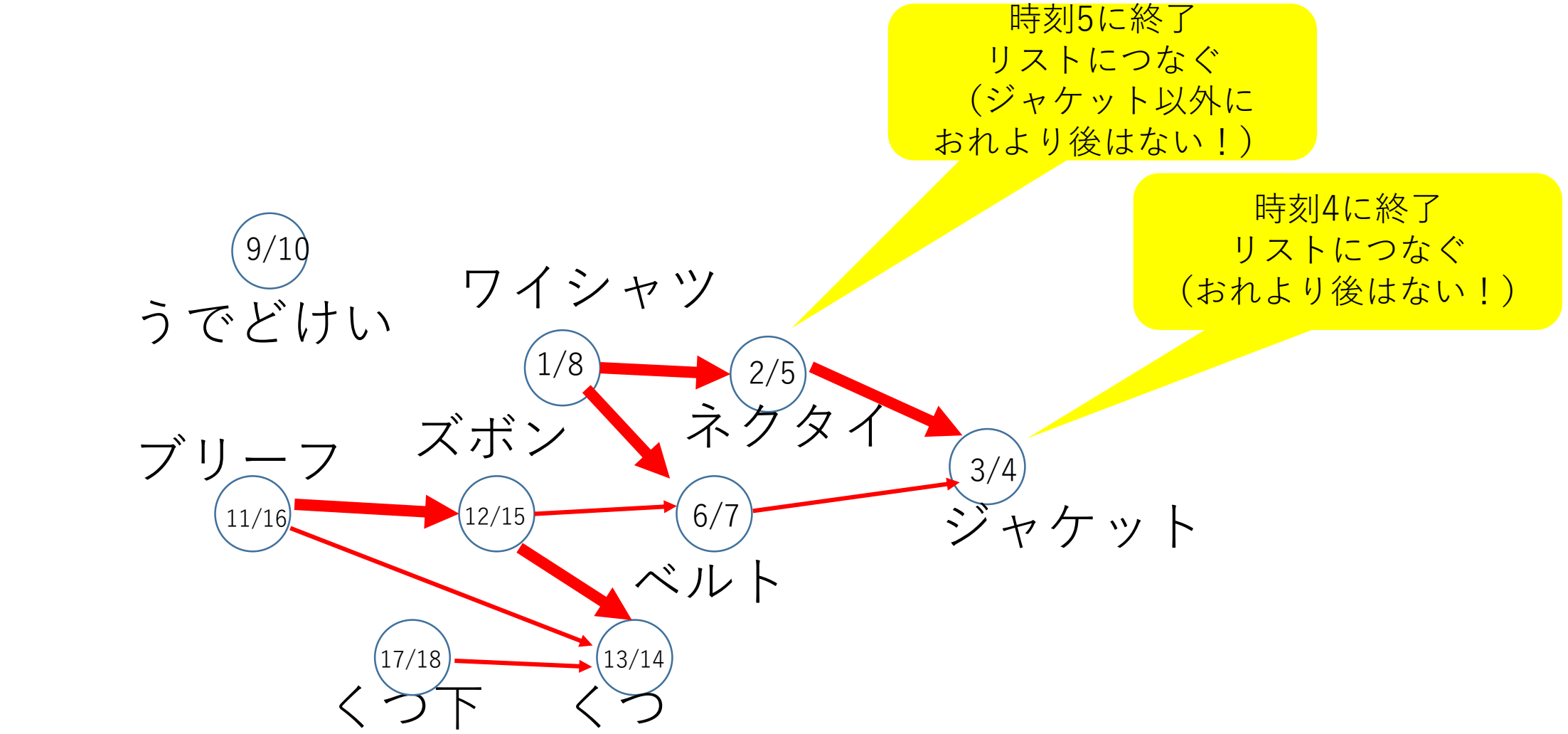
(半順序 = 反射的、反対称的、推移的 な関係)
(全順序 = 任意の2つの要素 a,bの間に、aRb or bRa のいずれかが成り立つ半順序関係)

トポロジカルソートは、
全ての有向辺が右向きになるように、グラフの点のすべてを水平線上に配置することに相当する。

Topological-Sort(G)
begin
　DFS(G)を実行する。
　ただし、各点vの f[v]が決定した時、
　この点を、リストLにつなぐ。
　return リスト L
end

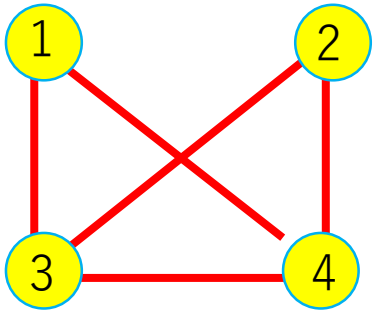


計算時間は O(|V| + |E|) である。



くつ下　ブリーフ　ズボン　くつ　うでどけい　ワイシャツ　ベルト　ネクタイ　ジャケット

理解確認クイズ

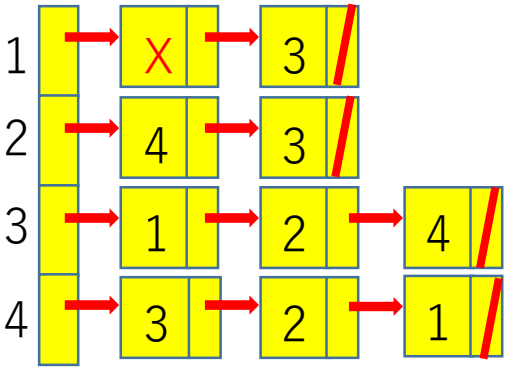


（１）

右上のグラフを、隣接リストと隣接行列で右のように格納した。

Ｘ，Ｙ，Ｚの数字を求めよ。

隣接リスト



隣接行列

	1	2	3	4
1	0	0	1	1
2	Y	0	1	1
3	1	1	0	1
4	Z	1	1	0

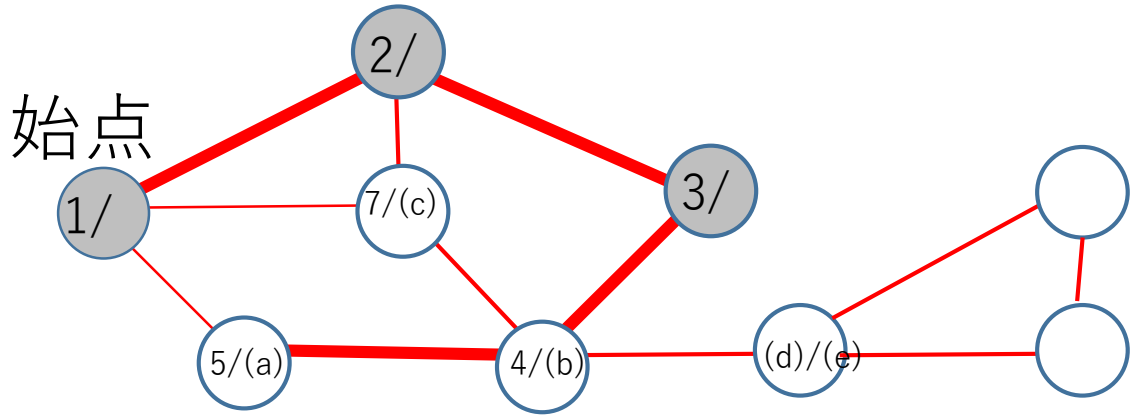
理解確認クイズ

（2）

下のグラフを、DFSする。

この様子を説明せよ。

（各点中の　?/?を埋めよ）



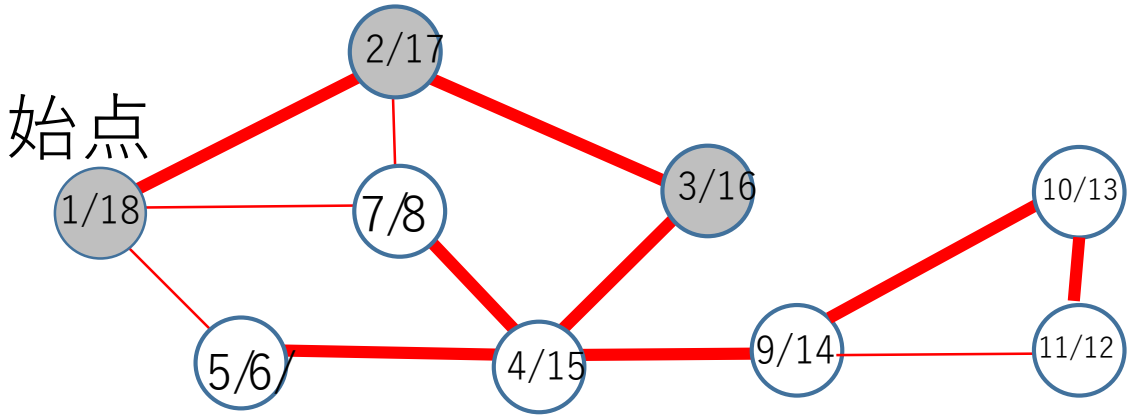
理解確認クイズ

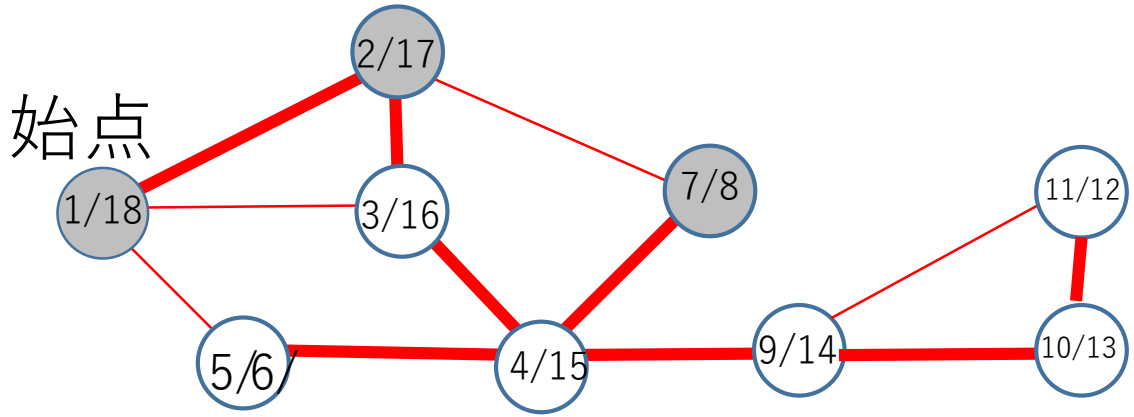
（2）

下のグラフを、DFSする。

この様子を説明せよ。

（各点中の　?/?を埋めよ）

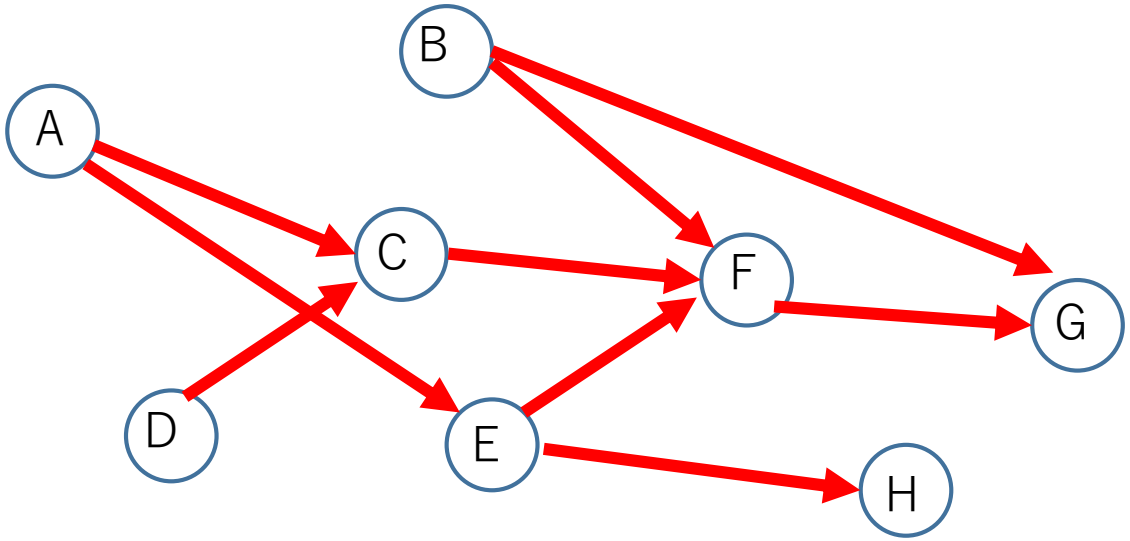




理解確認クイズ

（3）

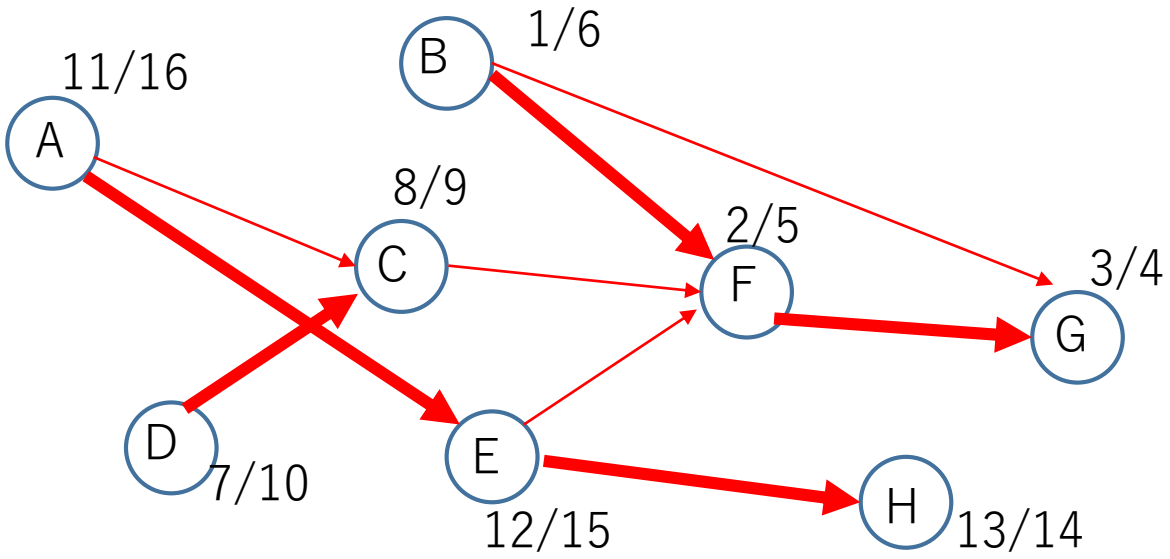
下の有向グラフのトポロジカルソートを求めよ。



理解確認クイズ

（3）

下の有向グラフのトポロジカルソートを求めよ。



A E H D C B F G