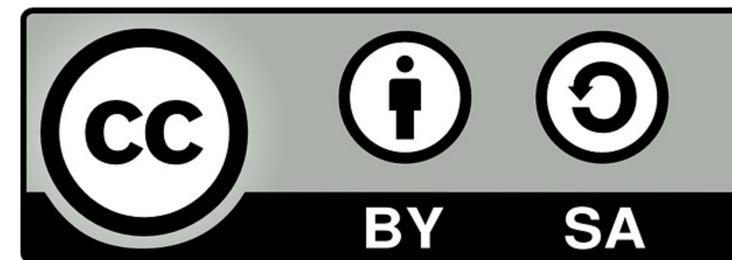
Algorithms (2024 Summer)
#11：グラフアルゴリズム2（最小全域木, トポロジカルソート）

矢谷 浩司
東京大学工学部電子情報工学科



レポート課題

https://yatani.jp/teaching/lib/exe/fetch.php?media=2024algorithms:report.pdf

Turnitin経由で提出をお願いします．それ以外の提出方法は認めません．アカウント情報を含めてTurnitinからECCSアカウント宛にメールをお送りしていますので，確認してください（スパムフォルダも確認してください）．

Turnitinからのメールが届いていない場合は，矢谷までDMください．

期末試験

日時：7/31 12:55**集合，**13:05**開始，**14:40**頃解散予定**

試験時間：80分

会場：工学部2号館4階241，242教室を予定．

後日，座席を指定しますので詳細をお待ち下さい．

持ち込みは認めません．

【重要】期末試験受験者調査

期末試験を受験する予定の人は全員，以下のアンケートで回答をしてください．

https://forms.gle/7XdDFwtKsboLkWS68

回答期限：7/12 24:00

このアンケートに回答していない場合，期末試験の受験を認めないことがあります．

対面試験

期末試験受験は対面です.

ただし，非常に特別な事情がある方に関しては，別途対応することがありますので，その事情を詳細に私に説明していただければと思います.

本郷キャンパス以外にいるため，他学科の期末試験日程と重複するため，などの理由は対象外とします.

試験の公平性を重視するため，対面で試験を受けていただくことを想定していることをご理解ください.

過去問

https://drive.google.com/drive/folders/1Coxpw6JYwA09npRfTwq-Aa_XwlRTviq8?usp=sharing
（ECCSアカウントでのログインが必要）

過去問は，大問・小問の数，解答形式，出題範囲，問題の難易度を規定するものではないことに留意してください．

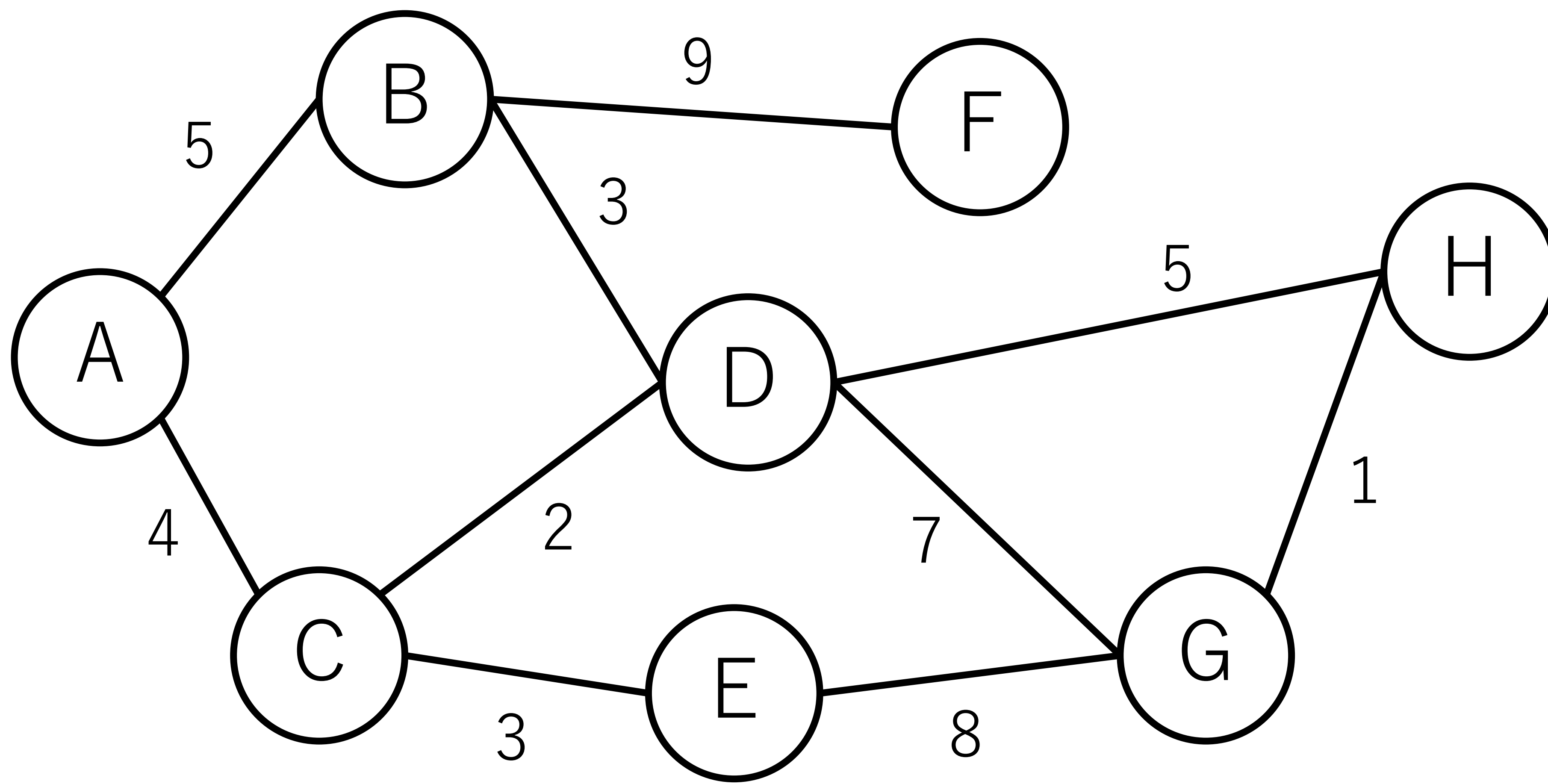
今日のテーマ

最小全域木

トポロジカルソート

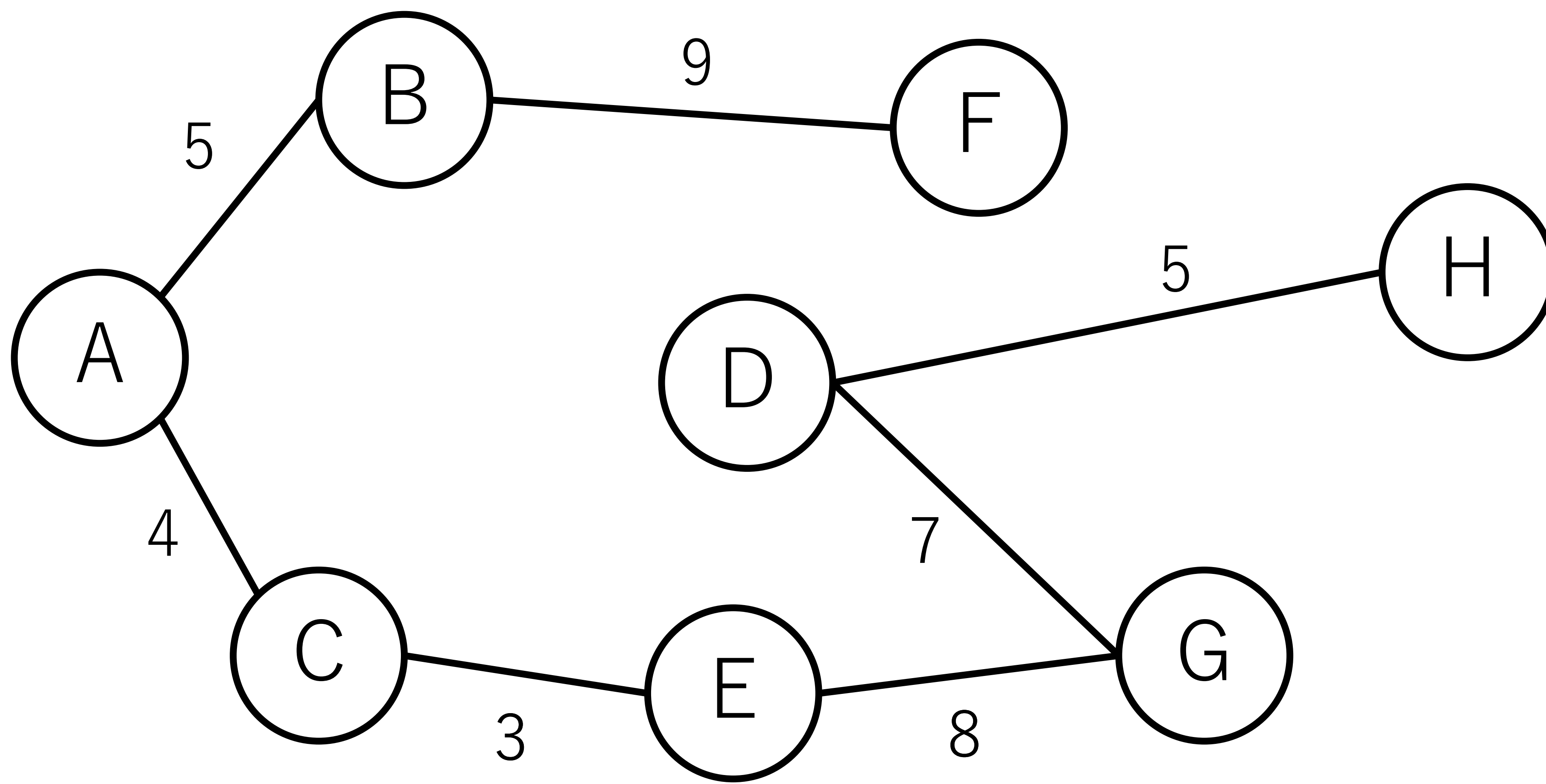
全域木（spanning tree）

グラフにおいて，すべての頂点がつながっている木（閉路を持たない連結グラフ）．



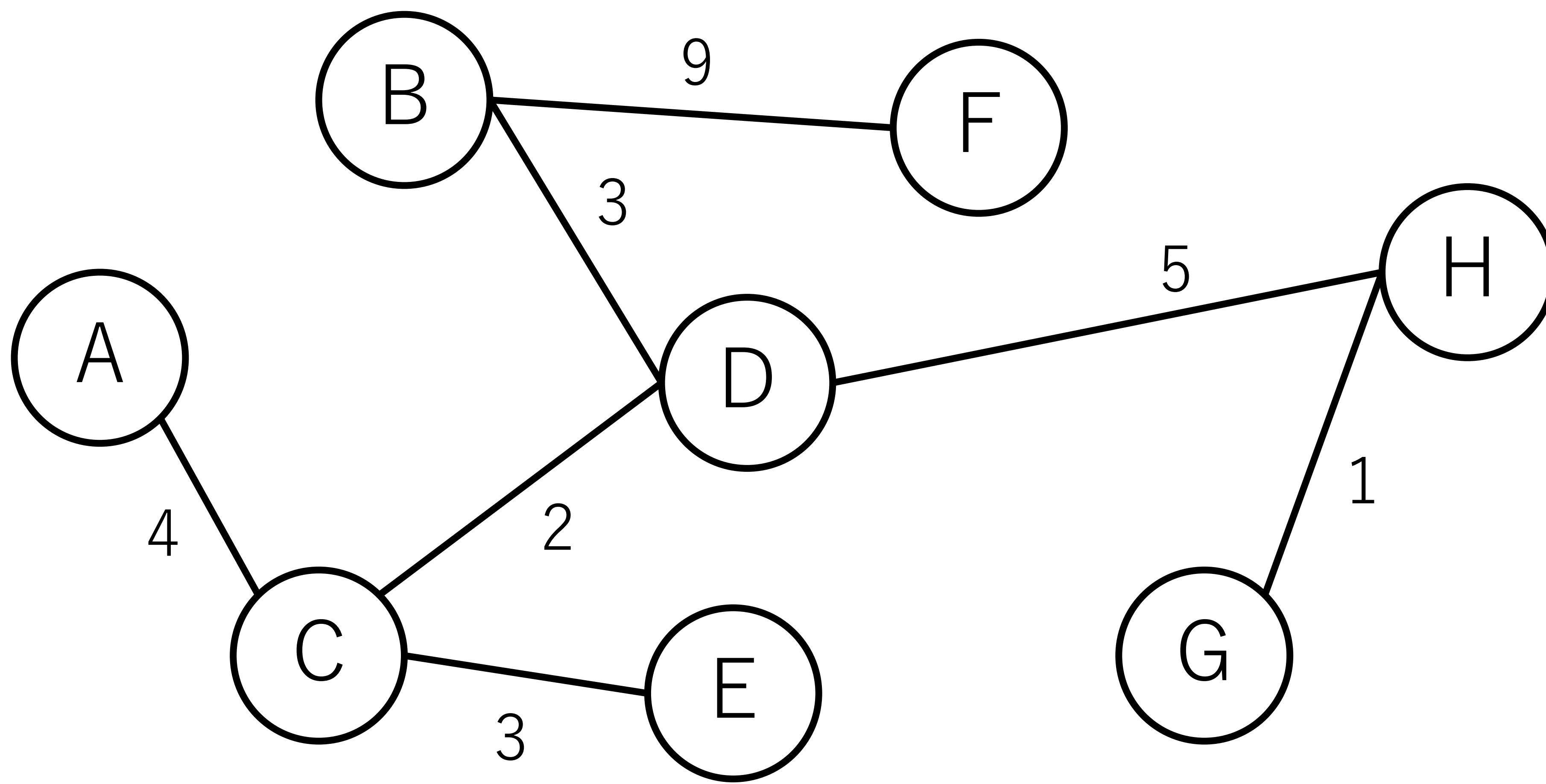
全域木

全域木の1例．木になっていて，すべての頂点がつながっている．



最小全域木（minimum spanning tree）

全域木の中で辺の距離（コスト）の総和が最小になるもの.



最小全域木がわかると何が嬉しい？

「複数の建物を有線のネットワークで接続する時,
コストを最小にするように線を引きたい．」

全体のコストを最小にしつつ，ノード間がつながって
いることが保証されないといけない.

最小全域木の求め方

辺ベースのアプローチ

存在する辺を距離の短い順に並べて順に入れていき,
閉路が出来ないことが確認できた場合は追加し,
全部の辺をチェックしたら終了.

ノードベースのアプローチ

すでに到達した頂点の集合からまだ到達していない
頂点の集合への辺のうち, 距離が最短のものを追加し,
全ノードつながったら終了.

最小全域木のアルゴリズム

辺ベースのアプローチ：クラスカル法

存在する辺を距離の短い順に並べて順に入れていき，
閉路が出来ないことが確認できた場合は追加し，
全部の辺をチェックしたら終了.

ノードベースのアプローチ：プリム法

すでに到達した頂点の集合からまだ到達していない
頂点の集合への辺のうち，距離が最短のものを追加し，
全ノードつながったら終了.

最小全域木のアルゴリズム

辺ベースのアプローチ：クラスカル法

存在する辺を距離の短い順に並べて順に入れていき，
閉路が出来ないことが確認できた場合は追加し，
全部の辺をチェックしたら終了.

ノードベースのアプローチ：プリム法

すでに到達した頂点の集合からまだ到達していない
頂点の集合への辺のうち，距離が最短のものを追加し，
全ノードつながったら終了.

クラスカル法（Kruskal）

#1 全ての辺を距離の短い順にソート.

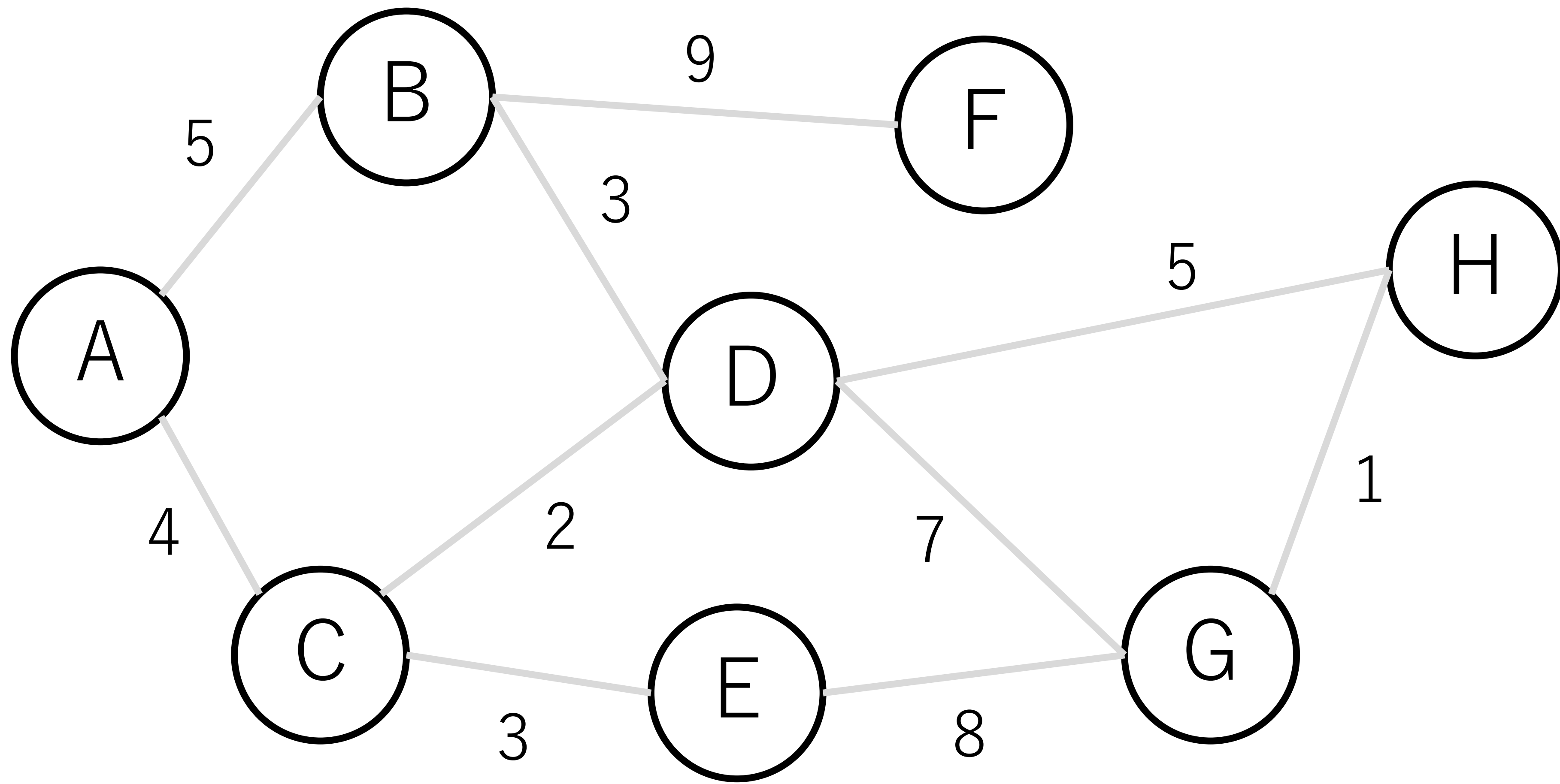
#2 一番距離の短い辺からスタート.

#3 今までに出来た木に辺を追加した時，閉路が新しく出来ないことを確認する．出来ない場合，この辺を最小全域木に追加.

#4 以降，全ての辺をチェックするまで#3を繰り返す.

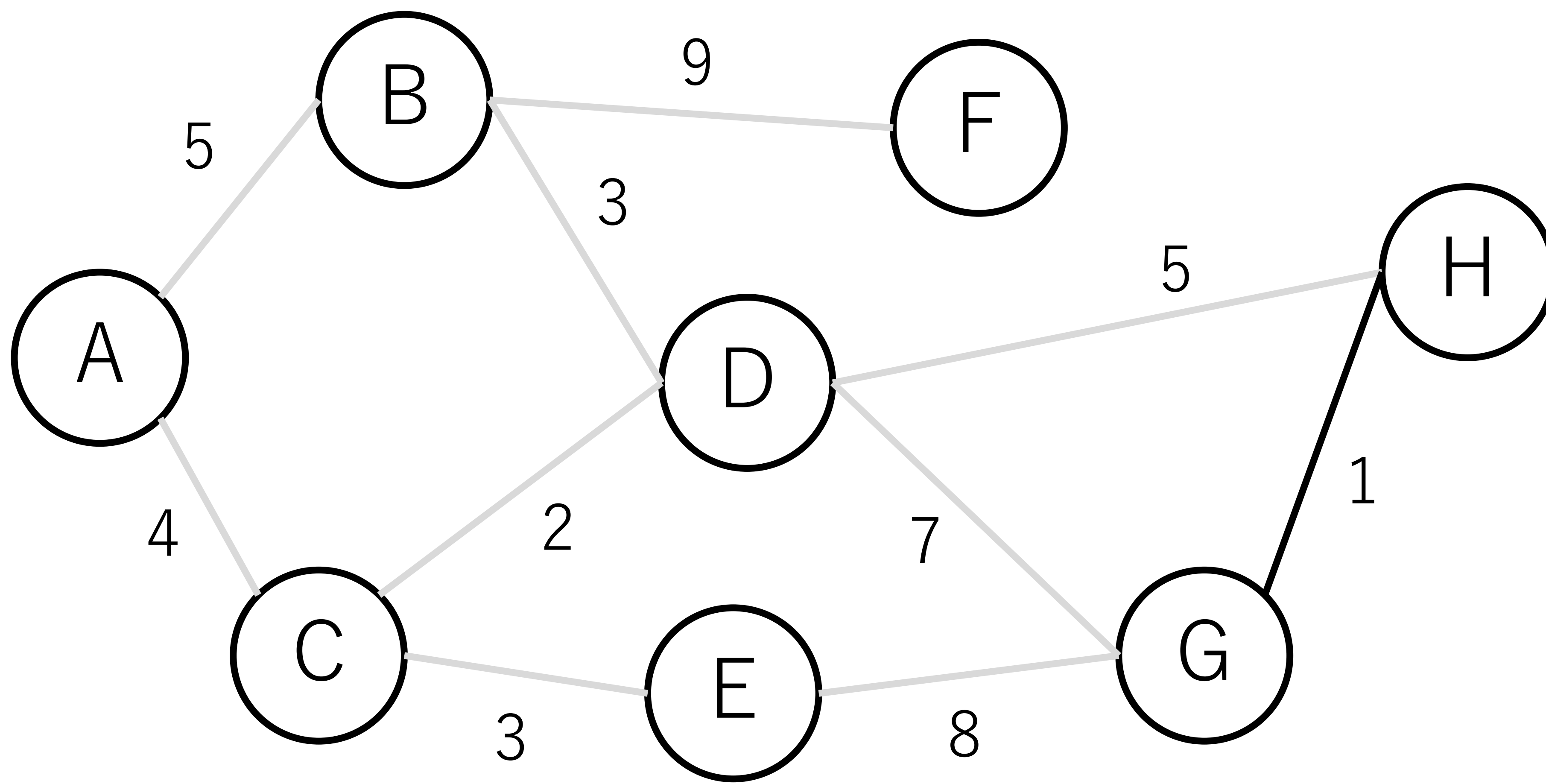
クラスカル法の例

すべての辺を距離の短い順にソート.



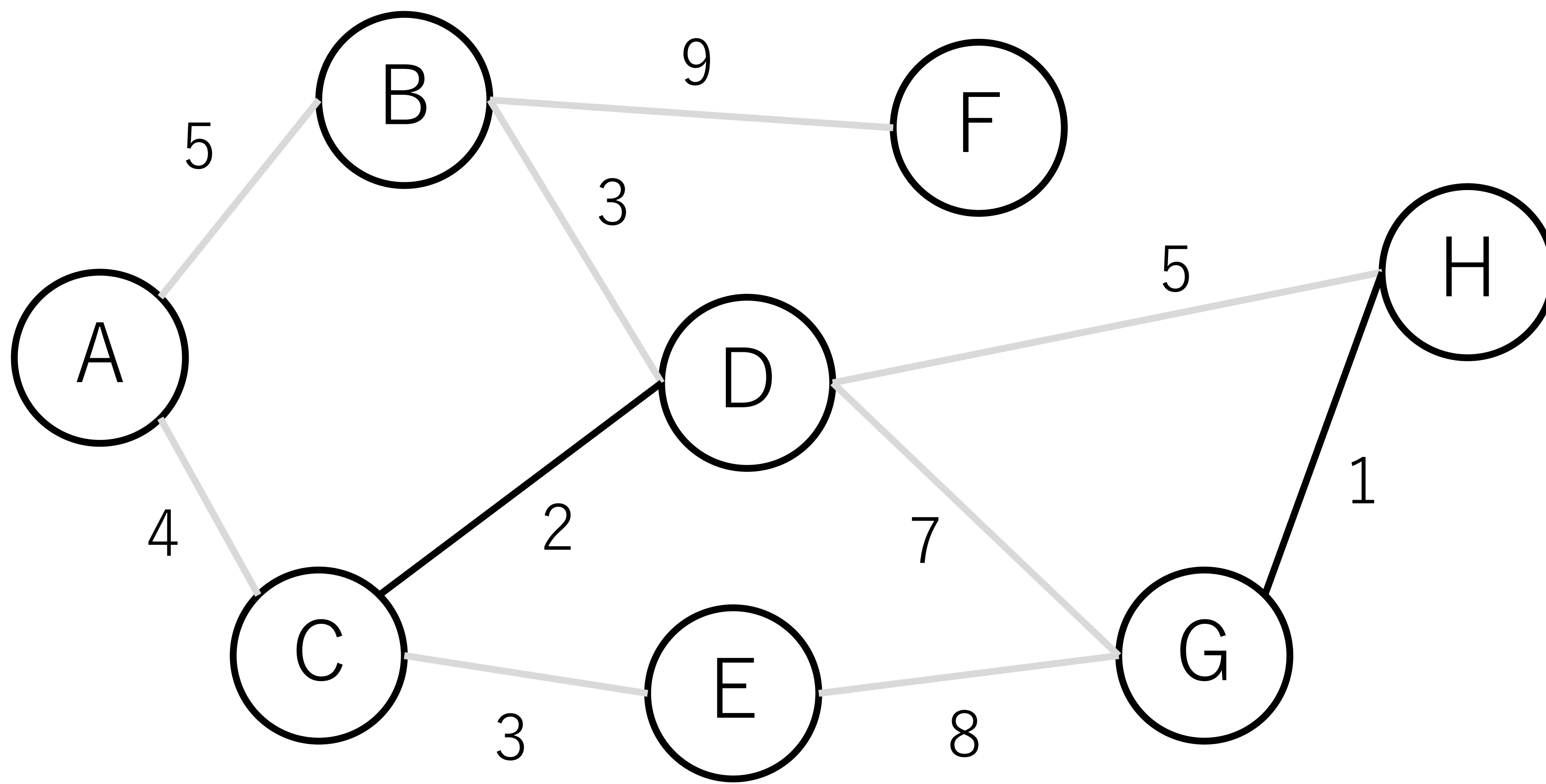
クラスカル法の例

一番距離の短い辺からスタート．閉路にならないので入れる．



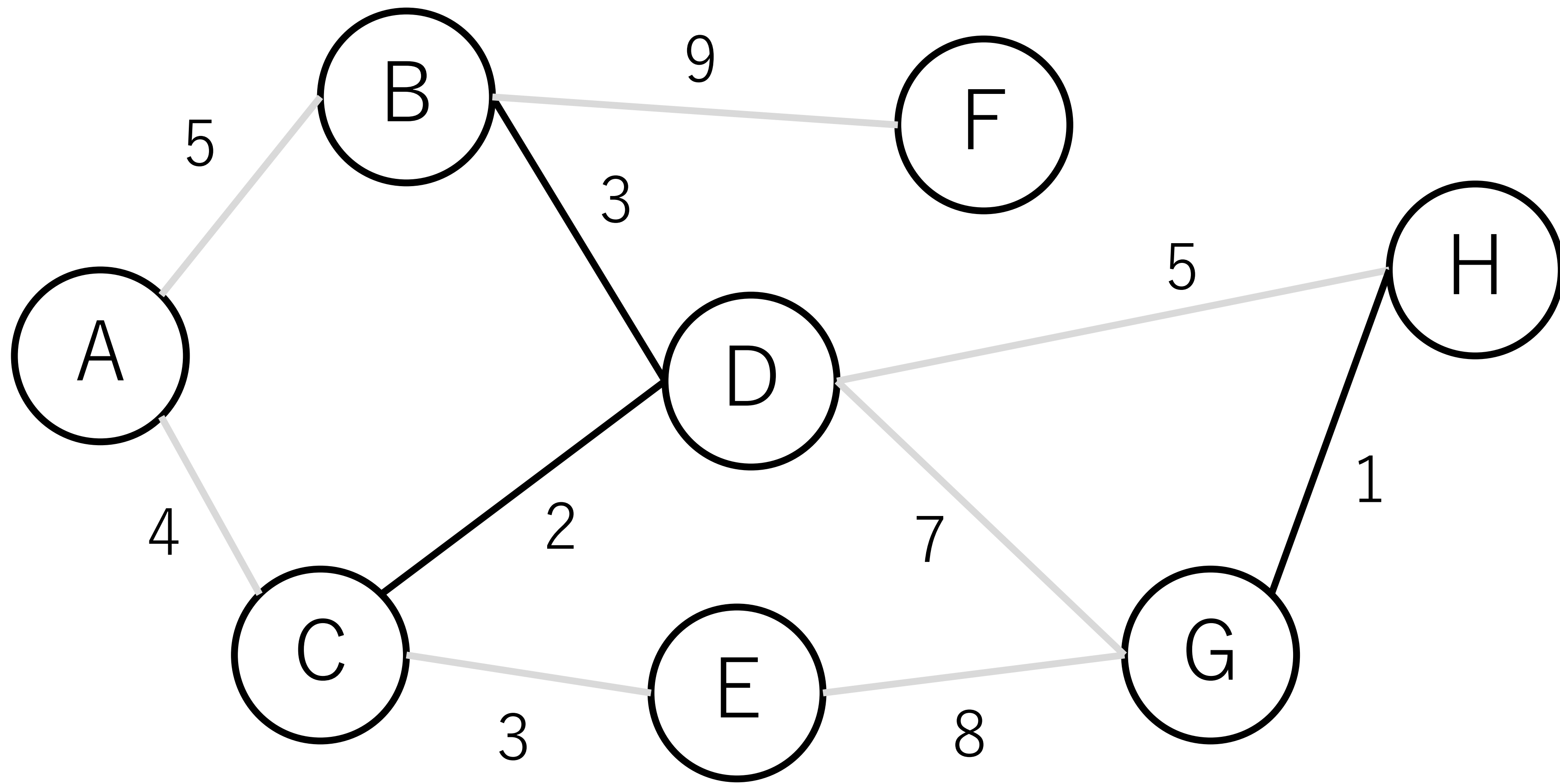
クラスカル法の例

C-Dも同じ.



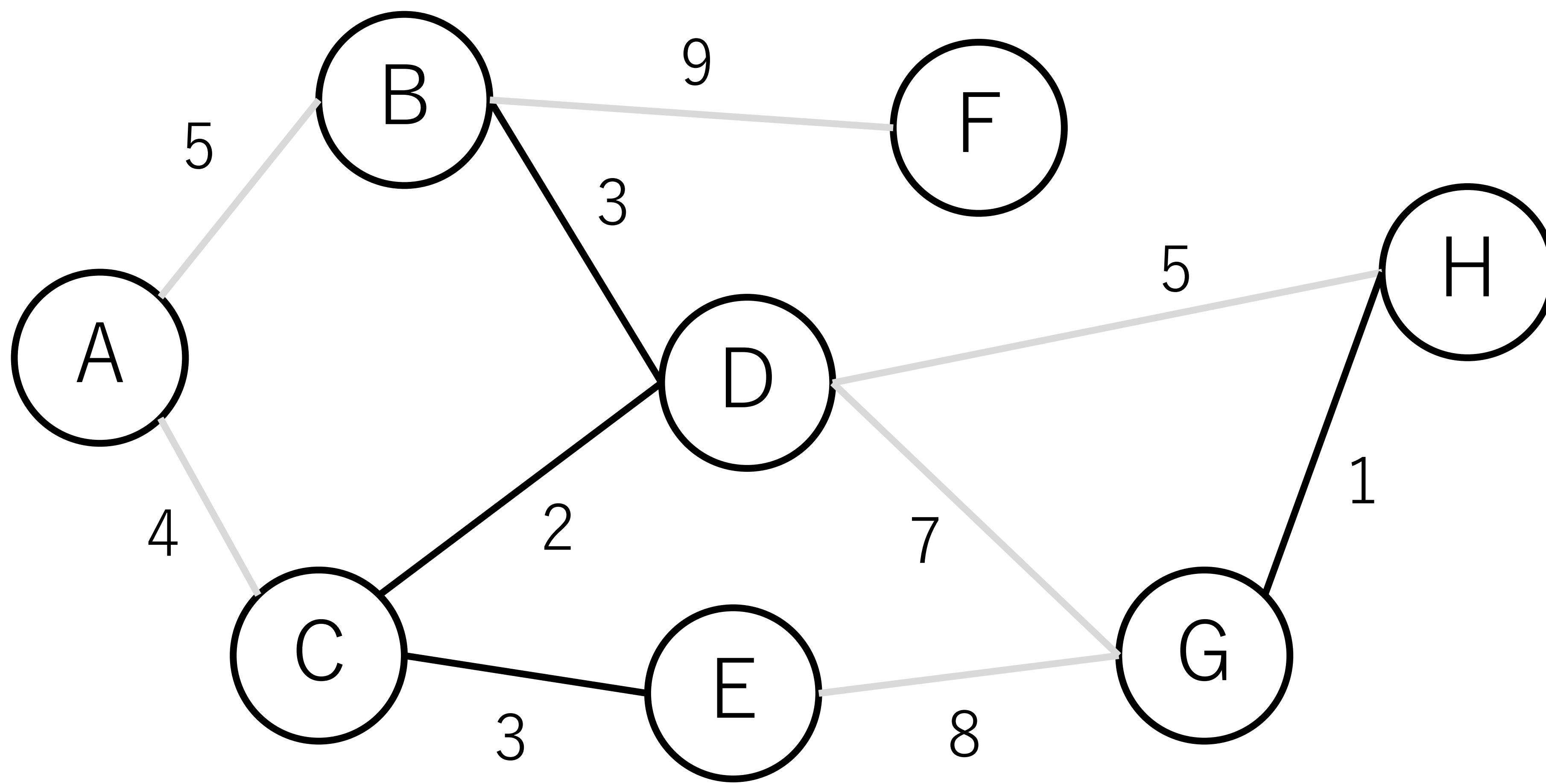
クラスカル法の例

B-Dも同じ.



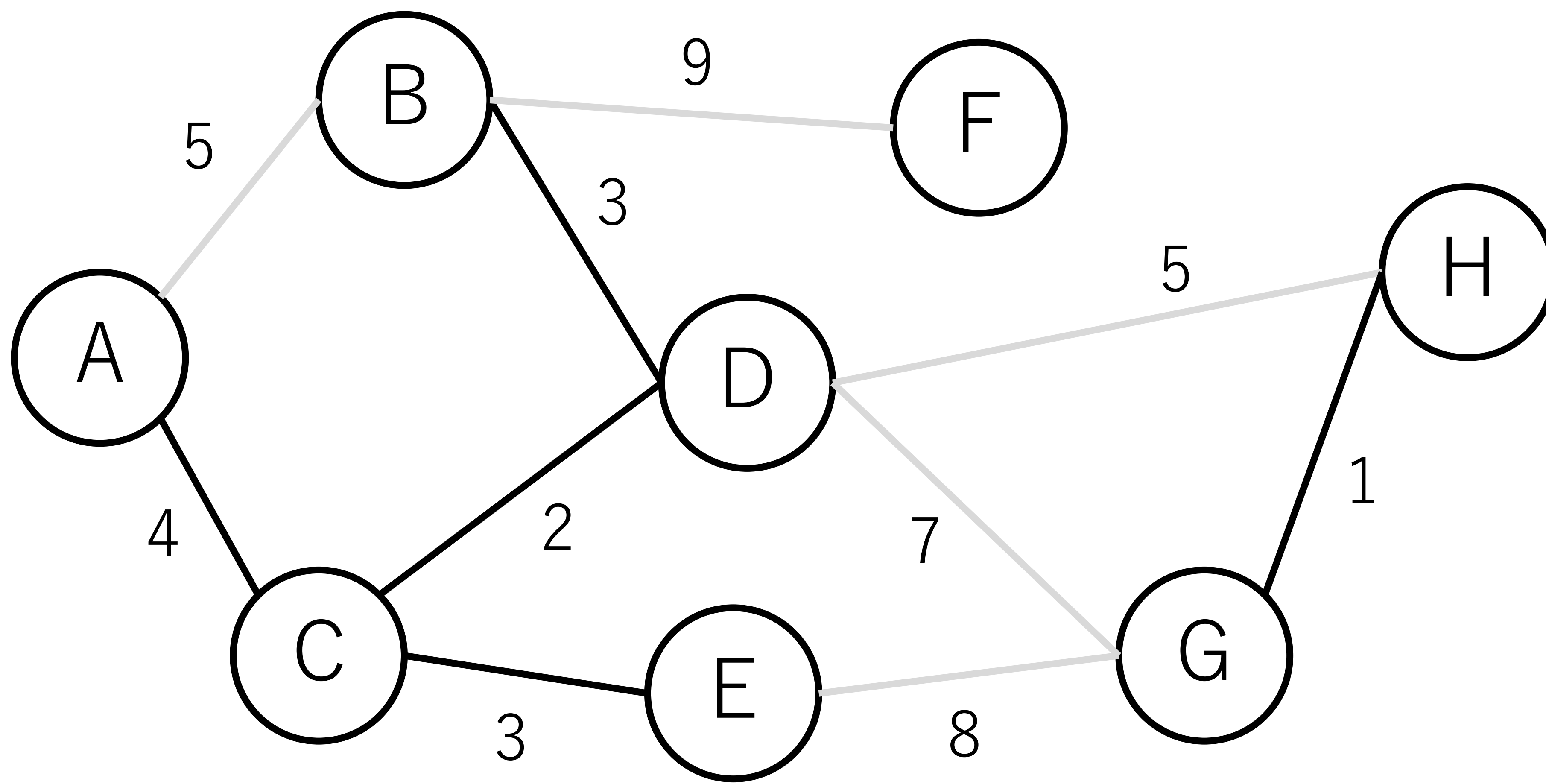
クラスカル法の例

C-Eも同じ.



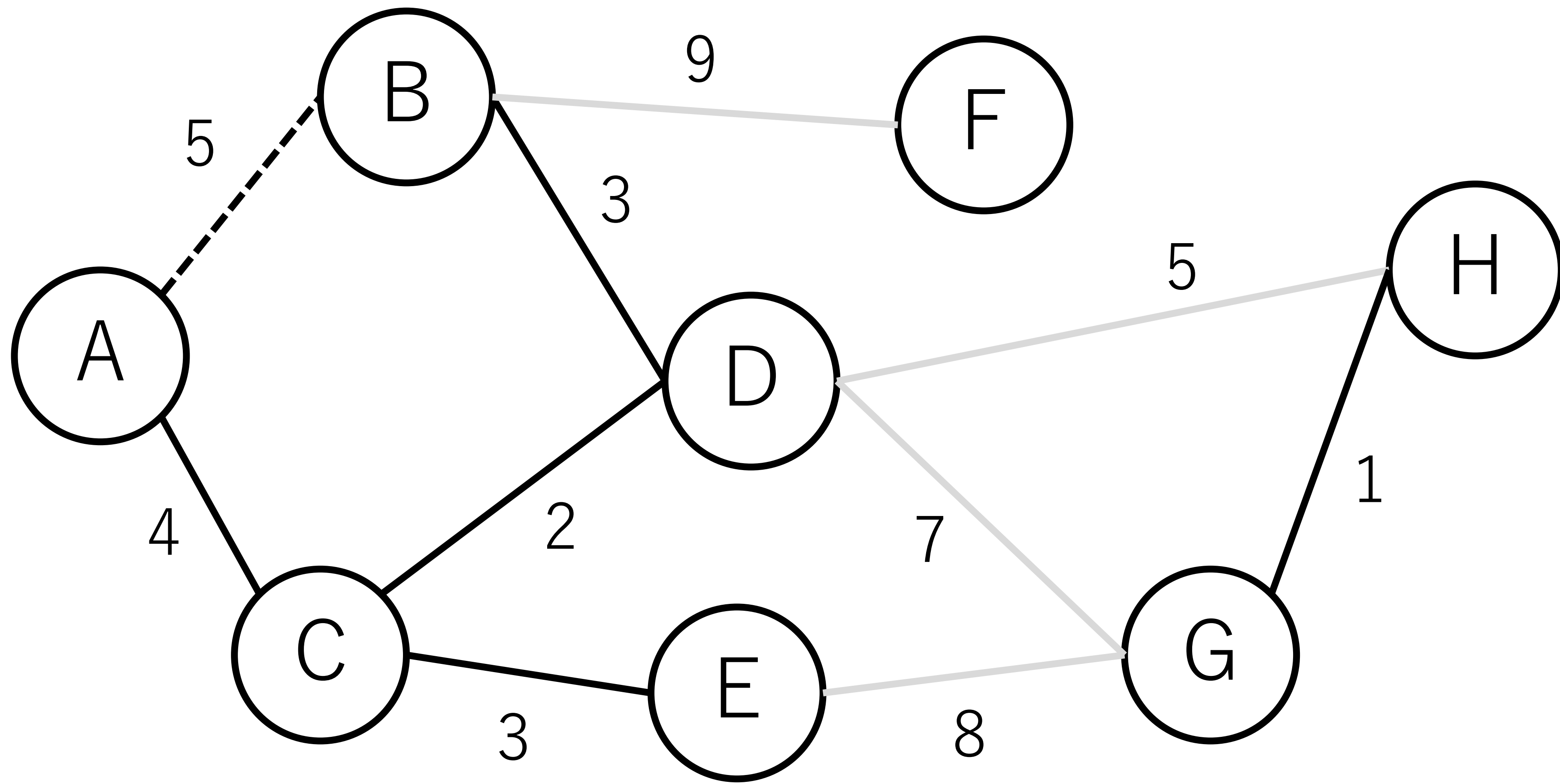
クラスカル法の例

A-Cも同じ.



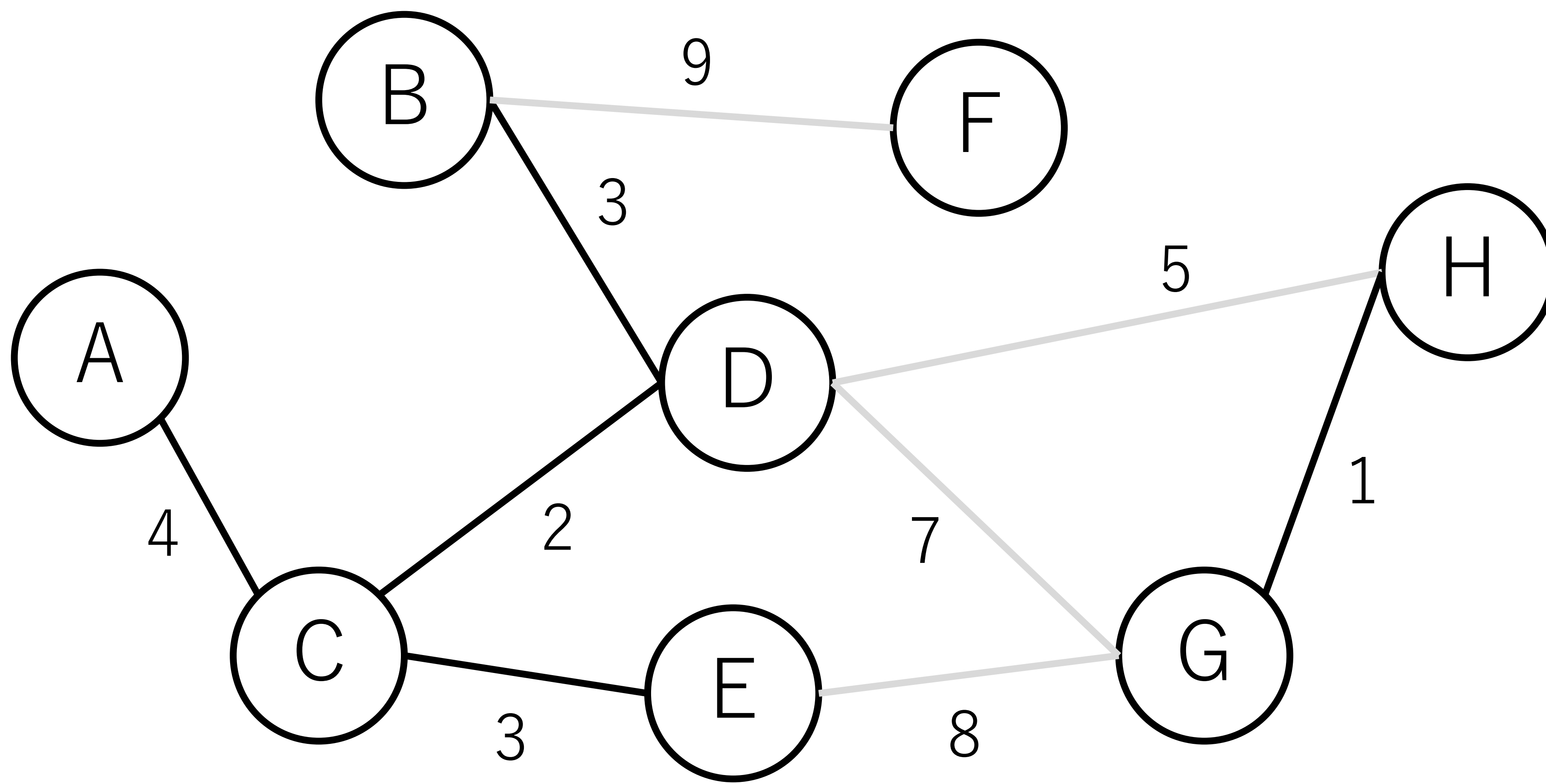
クラスカル法の例

A-Bをつないでしまうと閉路ができてしまう.



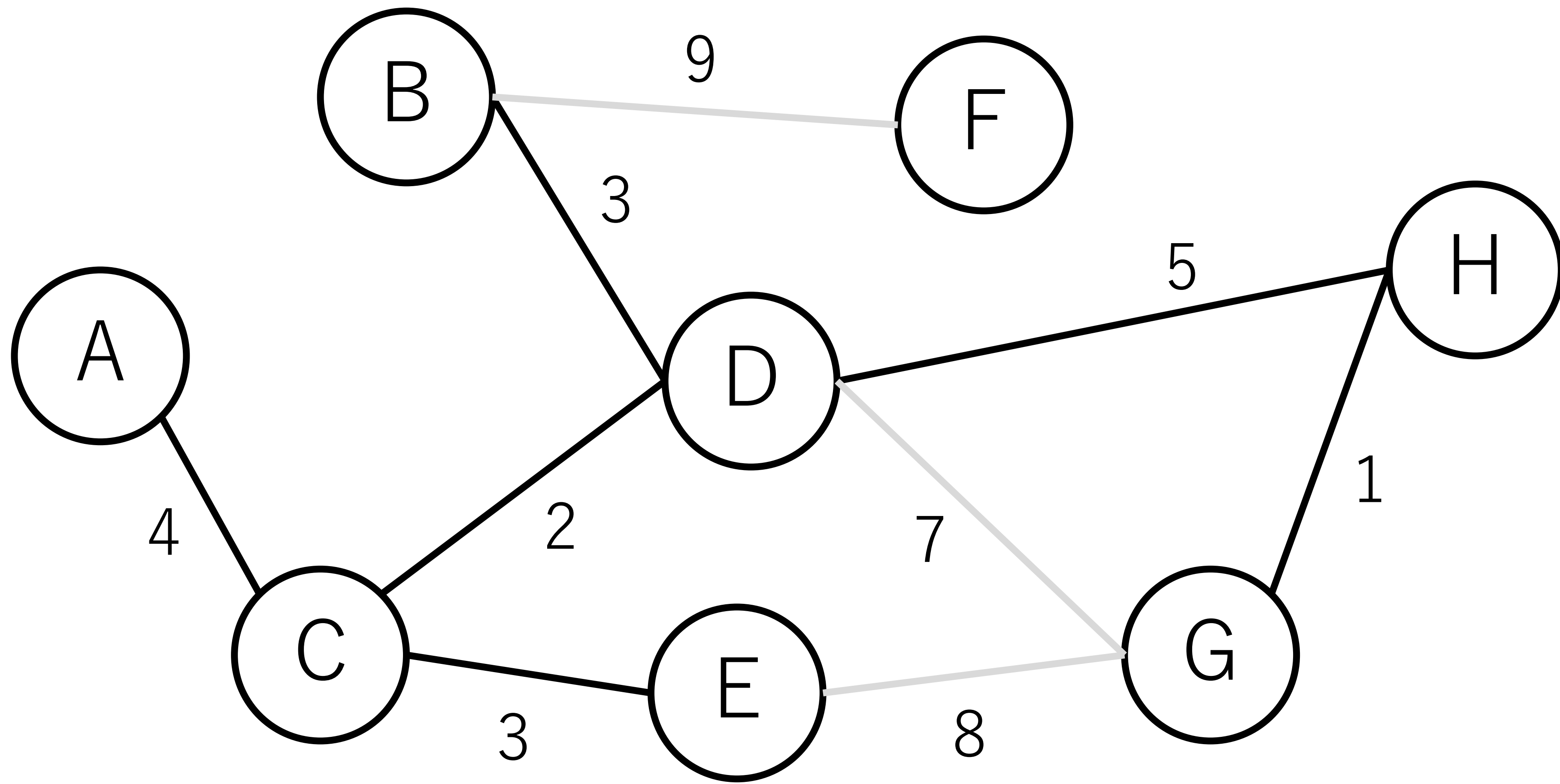
クラスカル法の例

よって，A-Bは入れない．



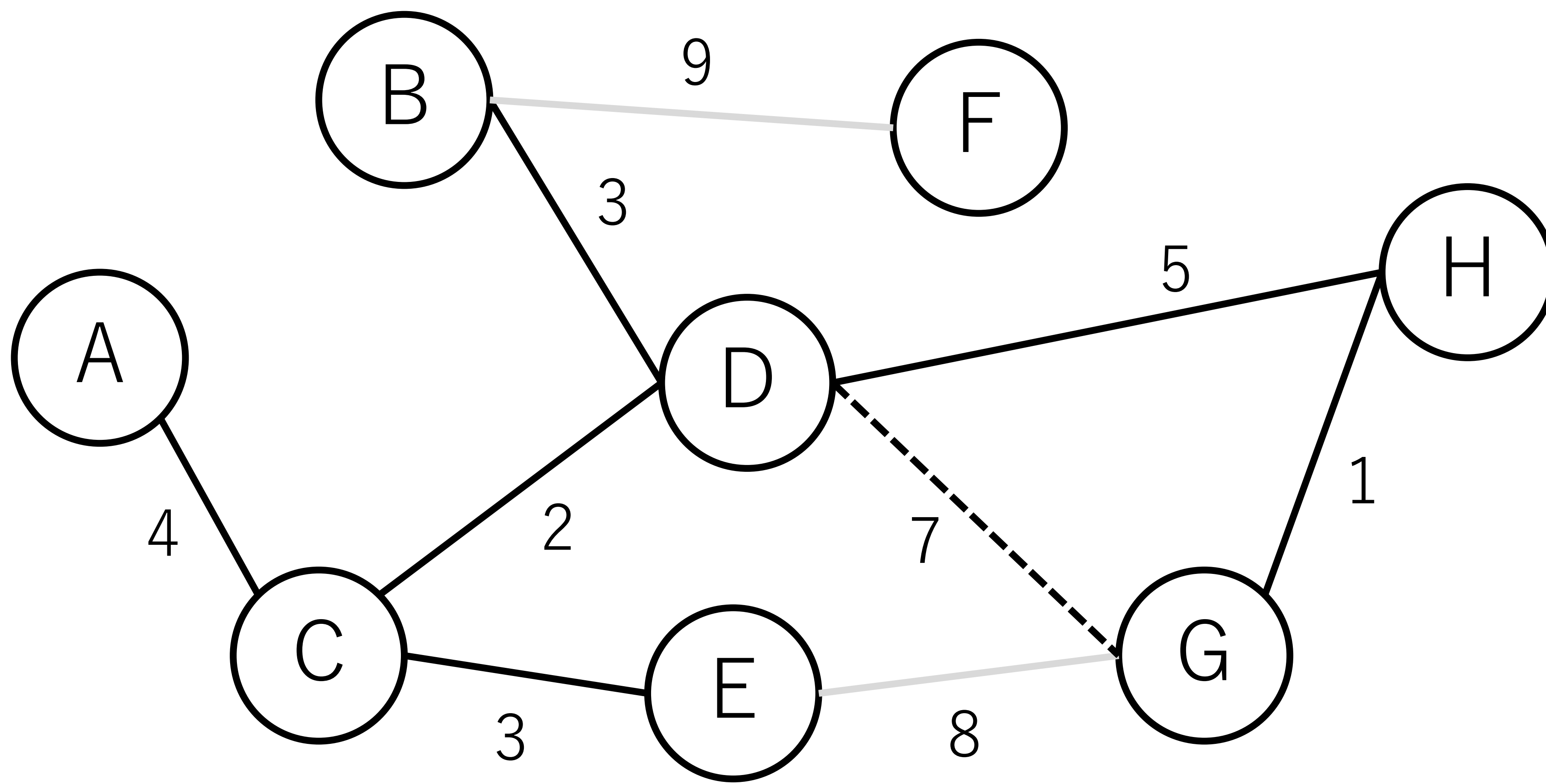
クラスカル法の例

D-Hは入れる.



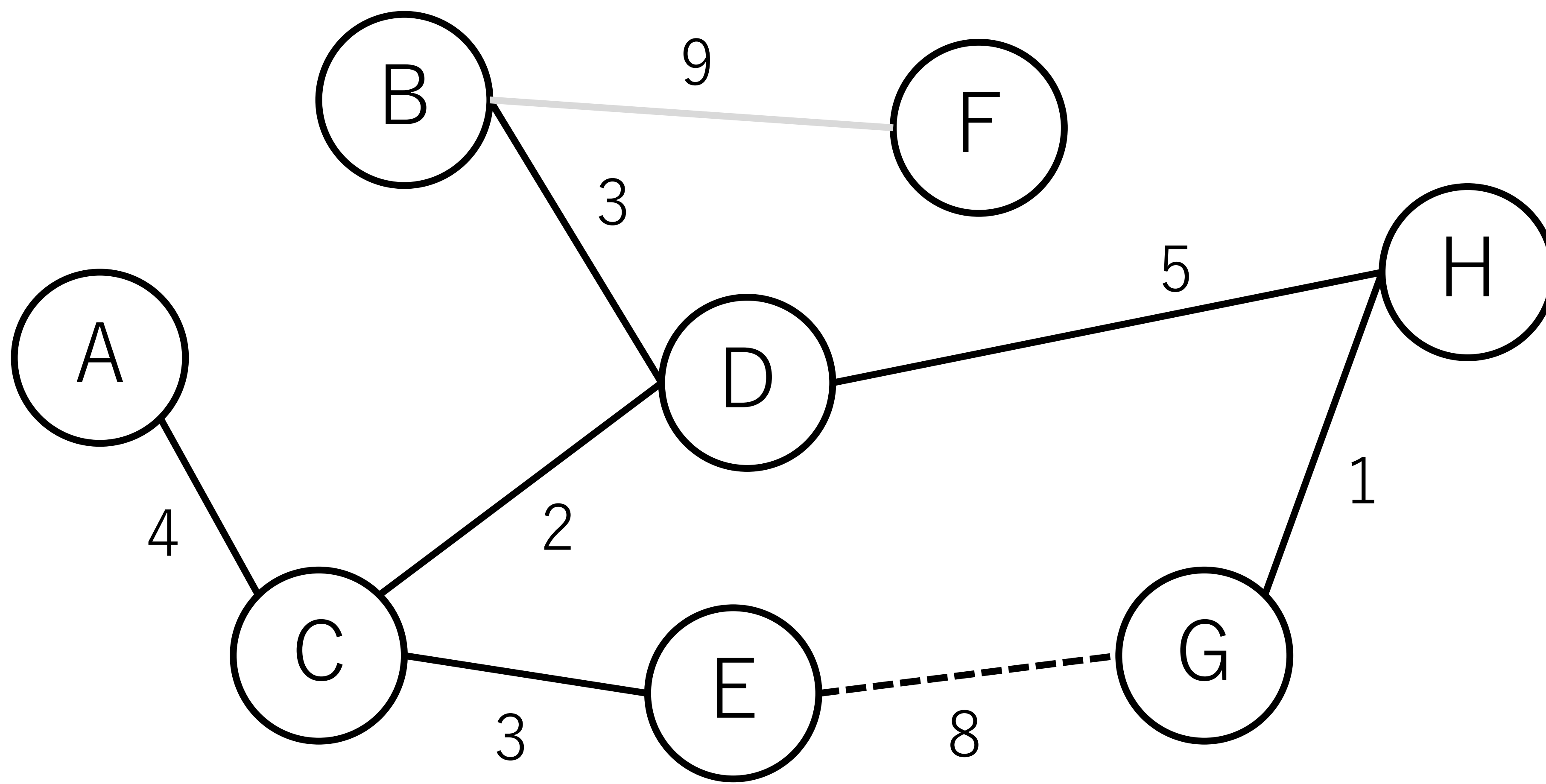
クラスカル法の例

D-Gは入れない.



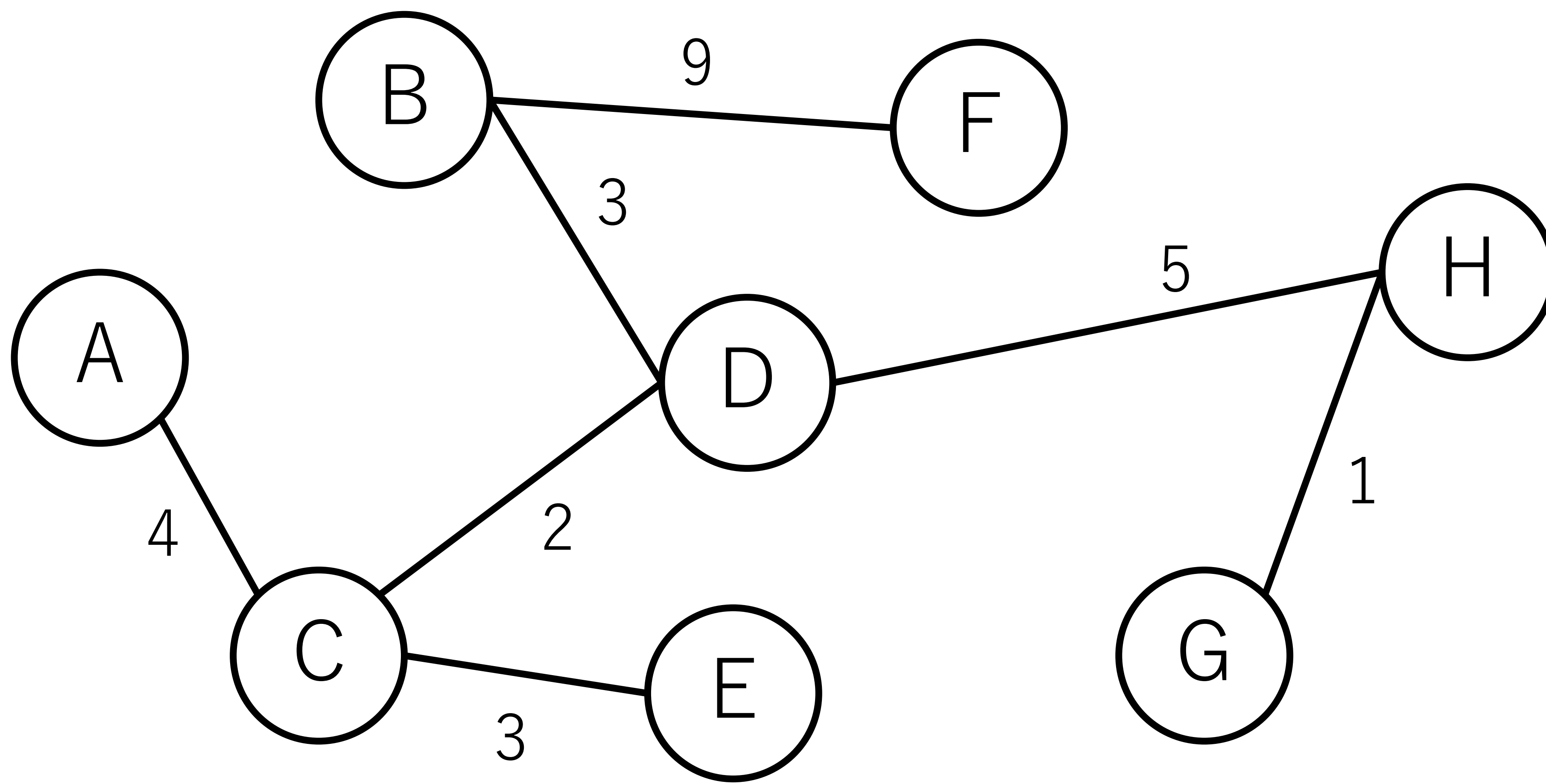
クラスカル法の例

E-Gは入れない.



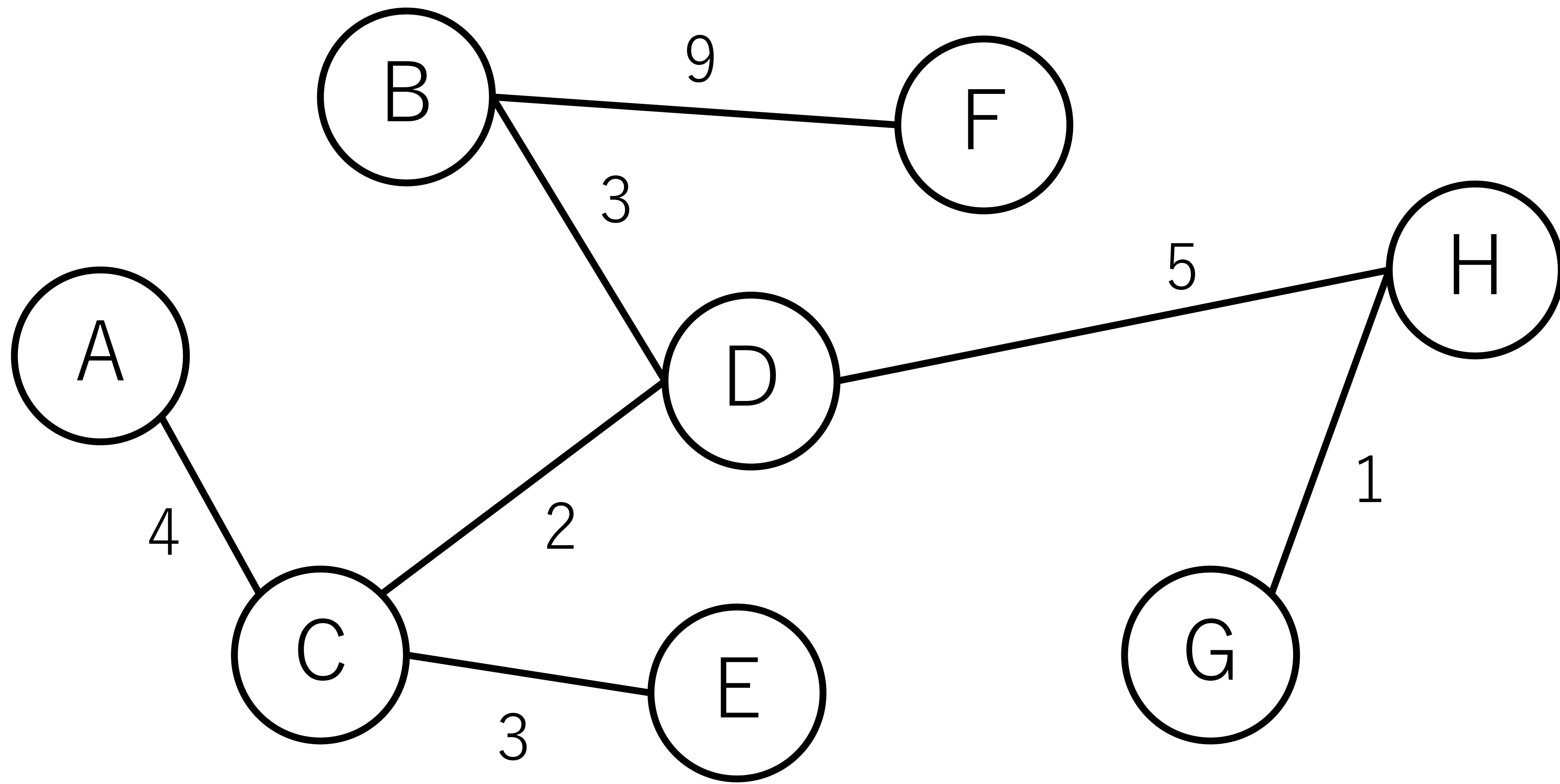
クラスカル法の例

B-Fは入れる.



クラスカル法の例

すべての辺が終わり，終了.



クラスカル法の実装

重要なポイントは2つある.

「存在する辺を距離の短い順に並べて順に入れていき」

これはソートすればよいだけ.

「閉路が出来ないことが確認できた場合は追加し」

これはどうやれば効率的に実現できる？

クラスカル法の実装

「閉路が出来ないことが確認できた場合は追加し」

→辺を足すごとに毎回グラフをたどる？

辺を足すごとにBFSもしくはDFSを行い，閉路がないことを確認する．

辺を1つ足すごとに，ノードが1つ増える．

よってノードの総数を$|V|$とすると，時間計算量は，

$O\left(\sum_{i=2}^{|V|} i\right) \rightarrow O(|V|^2)$であり，あまり効率的でない．

素集合データ構造（Union-Find木）

要素を素集合（互いに重ならない集合）に分割して管理するデータ構造．このデータ構造には2つの操作がある．

Union：2つの集合をマージする．

Find：ある要素がどの集合にいるかを見つける．

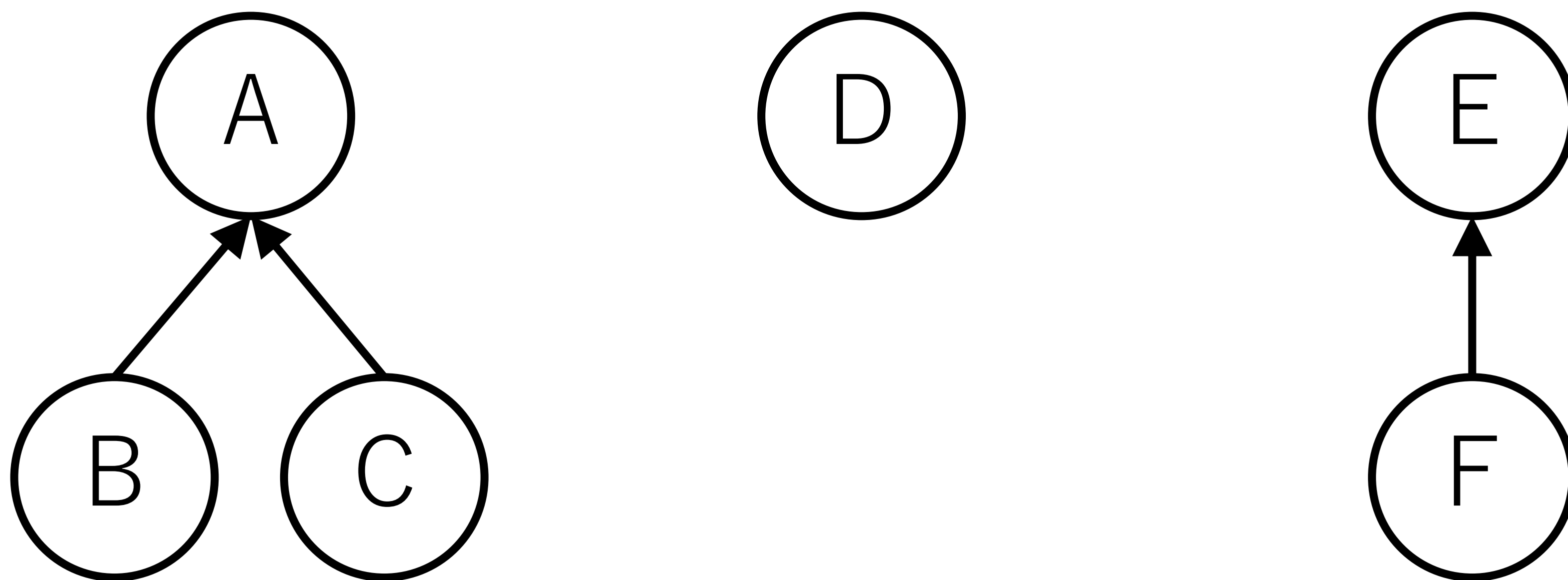
集合を元にroll backする操作はここでは扱わない．

興味のある方は，Undo可能Union-Find，永続Union-Findとか調べてみてください．

Union-Find木の例

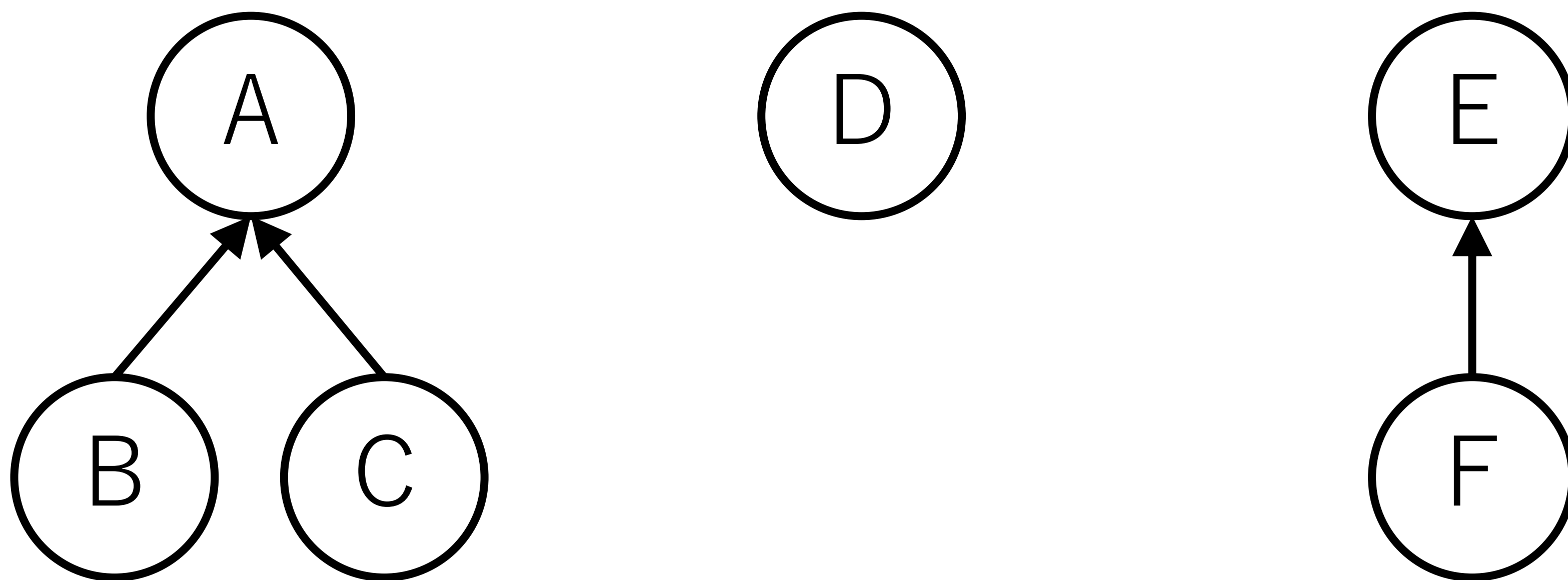
素集合

木が3つある（「森になっている」と表現することもある）．



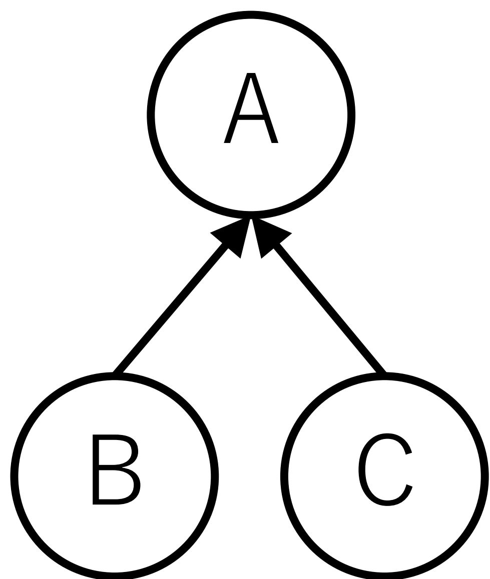
Union-Find木の例

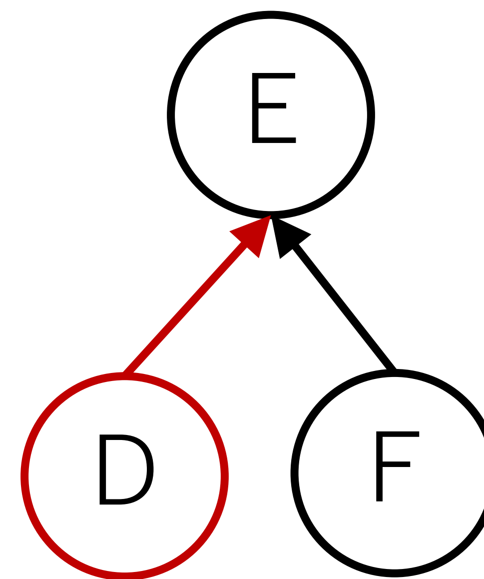
Unite：片方の根からもう片方の根につなぐ．



Union-Find木の例

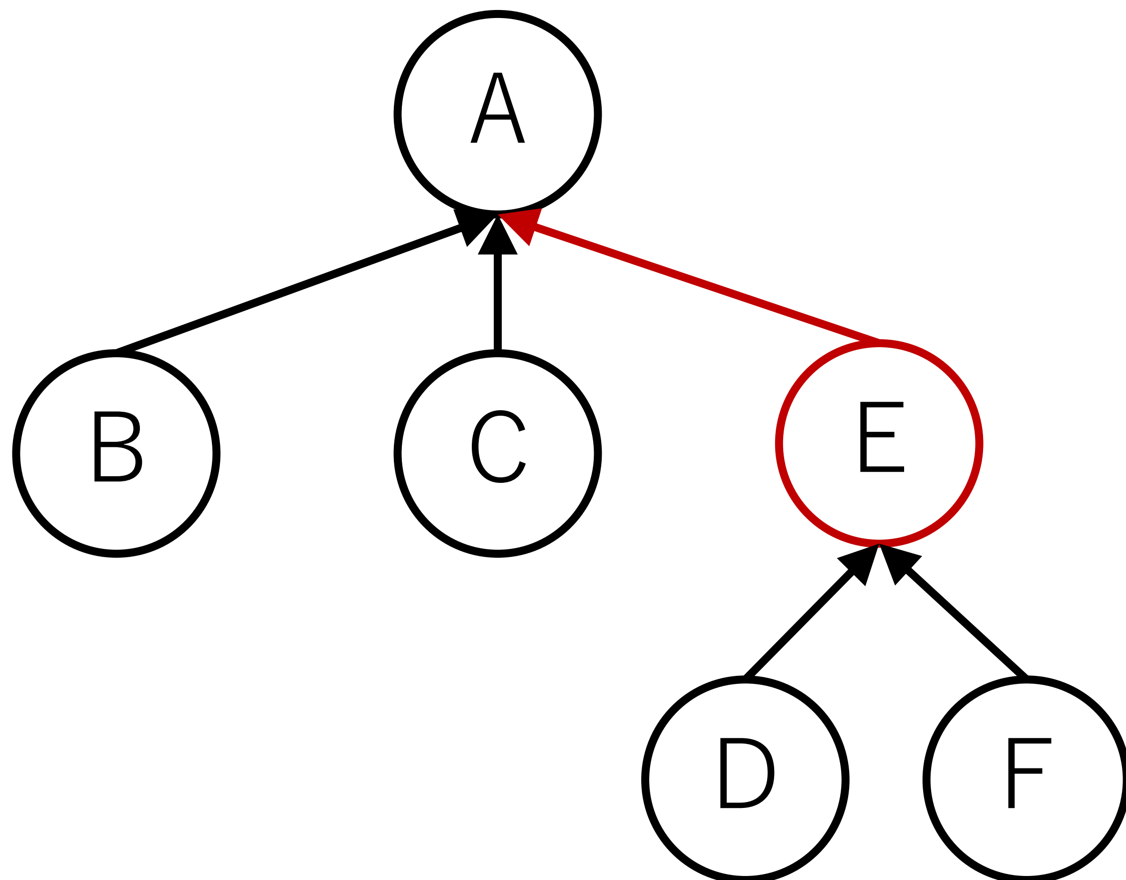
Unite：片方の根からもう片方の根につなぐ.
Dと[E，F]をマージ.





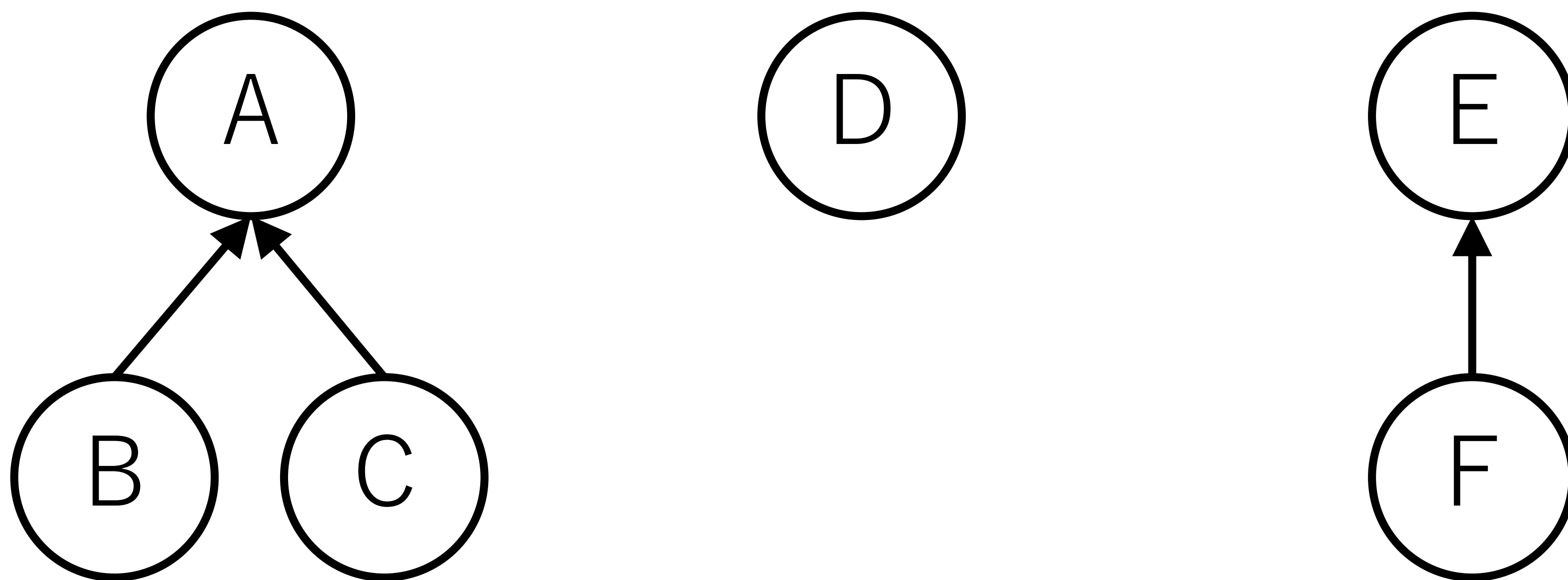
Union-Find木の例

Unite：片方の根からもう片方の根につなぐ．
残り2つの集合をマージ．



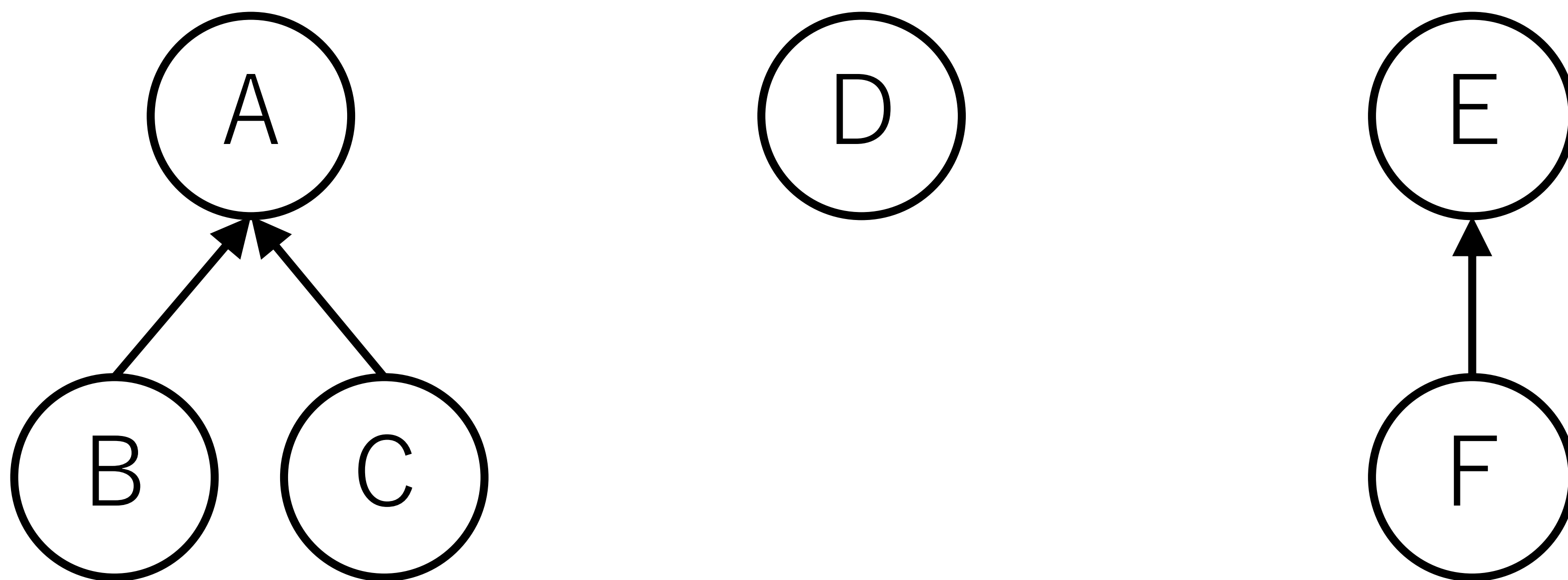
Union-Find木の例

Find：「同じグループである」＝「同じ根である」なので，
根ノードを返す．



Union-Find木の例

2つの要素が同じグループか：根ノードが同じかどうかをチェックする．



Union-Find木の実装

N個の要素がある時，長さNの配列を用意.

この配列には親ノードのindexを入れる.

自分が根ノードの場合は自分自身のindexを入れる.

この値をたどっていけば最終的に根ノードに行き着く.

最初の時点では自分自身しかグループに属していないので,
自分自身が根ノードになる

Union-Find木の効率化

できる限り根ノードに速くたどり着けるように構造を更新する.

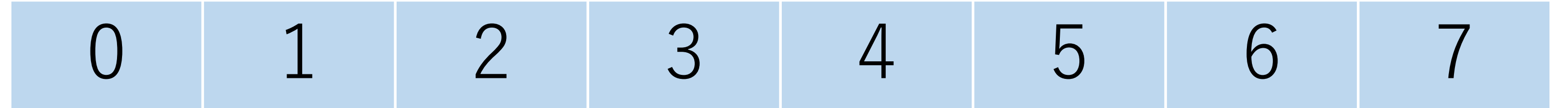
#1 Unite時に木の高さが高い方にマージ.

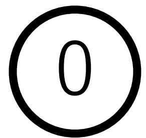
こうすることで，マージのときに出来る限り木を高くしない.

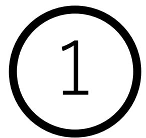
#2 根を調べたときに，直接根につながるようにつなぎ替える. （経路圧縮）

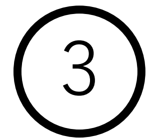
Union-Find木の例

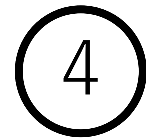
初期化：

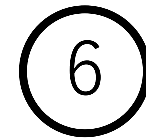


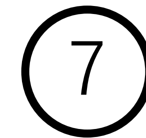






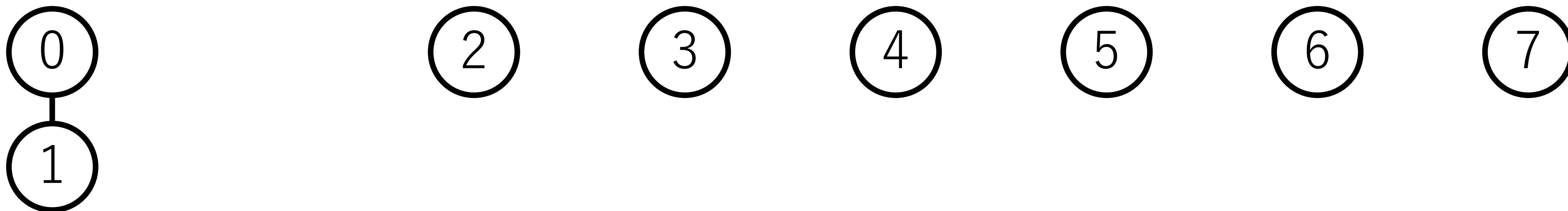






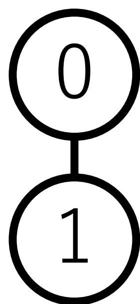
Union-Find木の例

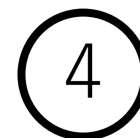
初期化：	0	1	2	3	4	5	6	7
0と1をunite：	0	**0**	2	3	4	5	6	7

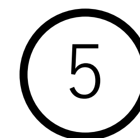


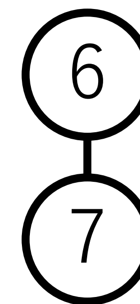
Union-Find木の例

初期化：	0	1	2	3	4	5	6	7
0と1をunite：	0	0	2	3	4	5	6	7
6と7をunite：	0	0	2	3	4	5	6	**6**



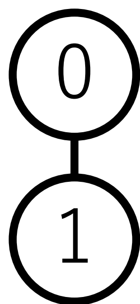


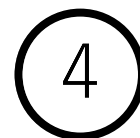


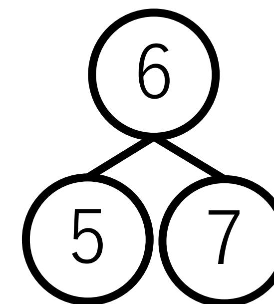


Union-Find木の例

初期化：	0	1	2	3	4	5	6	7
0と1をunite：	0	0	2	3	4	5	6	7
6と7をunite：	0	0	2	3	4	5	6	6
5と7をunite：	0	0	2	3	4	**6**	6	6

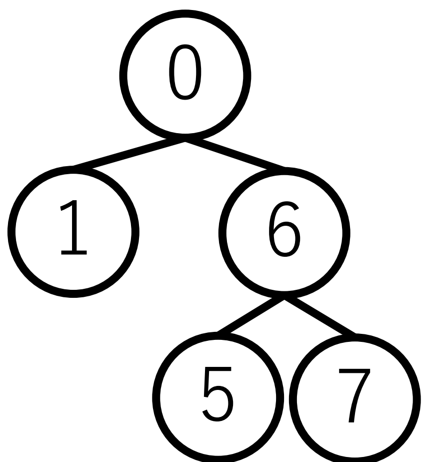


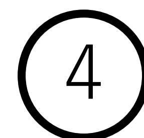




Union-Find木の例

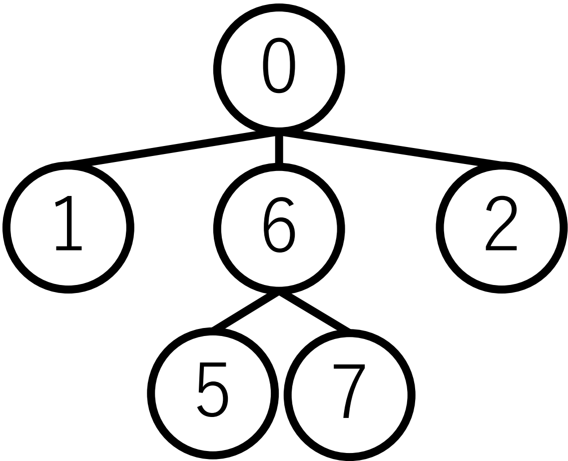
初期化：	0	1	2	3	4	5	6	7
0と1をunite：	0	0	2	3	4	5	6	7
6と7をunite：	0	0	2	3	4	5	6	6
5と7をunite：	0	0	2	3	4	6	6	6
1と7をunite：	0	0	2	3	4	6	**0**	6

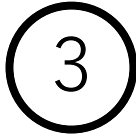


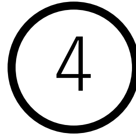


Union-Find木の例

初期化：	0	1	2	3	4	5	6	7
0と1をunite：	0	0	2	3	4	5	6	7
6と7をunite：	0	0	2	3	4	5	6	6
5と7をunite：	0	0	2	3	4	6	6	6
1と7をunite：	0	0	2	3	4	6	0	6
2と5をunite：	0	0	**0**	3	4	6	0	6





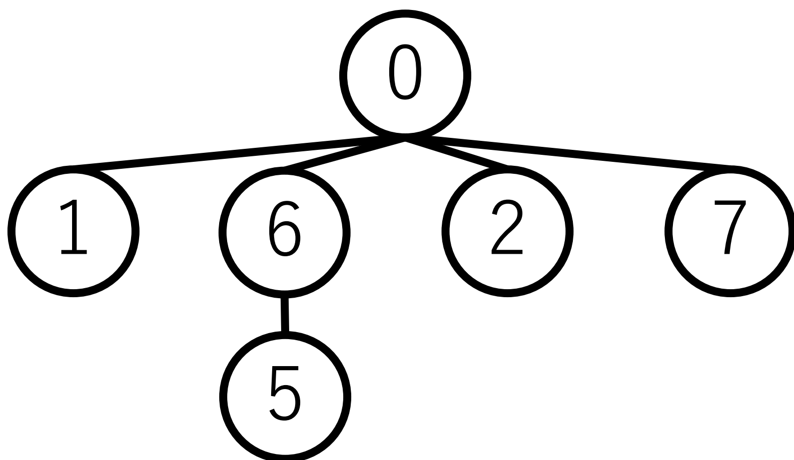


Union-Find木の例

現時点：	0	0	0	3	4	6	0	6

7の根ノード or 属するグループをチェック

チェック完了後：	0	0	0	3	4	6	0	**0**



3 4

Union-Find木の実装例

```
class UnionFind:
        def __init__(self, n):
                self.parent = [i for i in range(n)]
                self.height = [0 for _ in range(n)] # 各木の高さ
```

Union-Find木の実装例

```
def get_root(self, i):
        if self.parent[i] == i: # 自分が根ノードの場合
                return i
        else: # 経路圧縮しながら根ノードを探す
                self.parent[i] = self.get_root(self.parent[i])
                return self.parent[i]
```

Union-Find木の実装例

```
def unite(self, i, j):
        root_i = self.get_root(i)
        root_j = self.get_root(j)
        if root_i != root_j:      # より高い方にマージ
                if self.height[root_i] < self.height[root_j]:
                        self.parent[root_i] = root_j
                else:
                        self.parent[root_j] = root_i
                        if self.height[root_i] == self.height[root_j]:
                                self.height[root_i] += 1
```

Union-Find木の実装例

```
def is_in_group(self, i, j):
        if self.get_root(i) == self.get_root(j):
                return True
        else:
                return False
```

経路圧縮

経路圧縮の際には高さの更新は行わない.

- 経路圧縮前の高さを維持することになる.
- 経路圧縮後の木の高さを調べるのに，根から辿って $O(|E|)$かかるため.

集合のサイズの大きい方にマージするという実装もある.

- 各集合のサイズをすぐに取り出したいときに便利.

Union-Find木の計算量

正確にはアッカーマン関数$A(n,n)$の逆関数$\alpha(n)$になることが知られている.

$$A(0,0)=1, A(1,1)=3, A(2,2)=7, A(3,3)=61,$$
$$A(4,4)=2^{2^{2^{65536}}}-3, \ldots$$

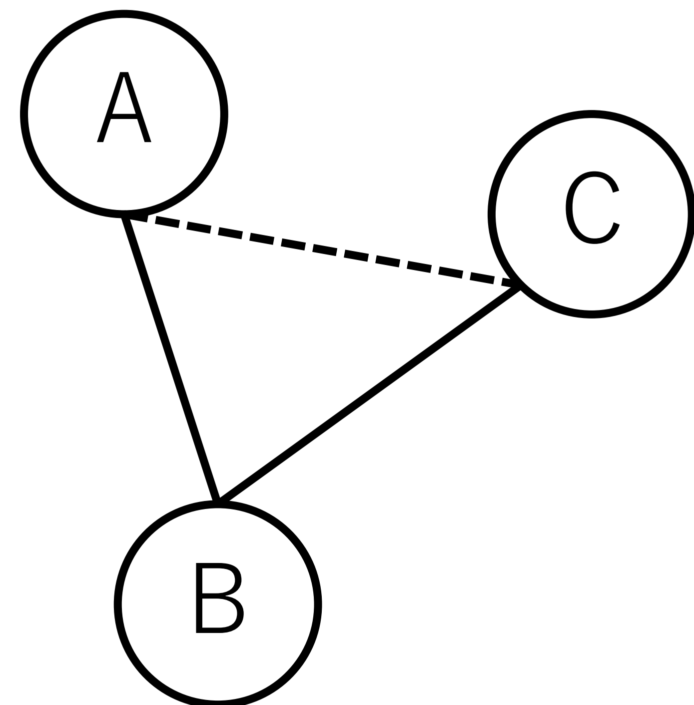
これはlogよりもさらに増加しない関数であり，定数倍とみなして扱われることもある.

Union-Find木による閉路の判定

ある辺が与えられた時，その2つのノードが同じグループに属している場合，その辺を加えると閉路が生まれることになる．

よって，2つのノードが同じグループに属していないことをチェックすれば良い．

→Union-Find木なら速攻できる！



クラスカル法の実装例

```
# 引数：ノードの総数，隣接リスト
def kruskal(V, e_list):
        e_cost_sorted = []    # 距離で整列された辺

        # ソートのために先頭の要素を距離にする
        for e in e_list:
                e_cost_sorted.append([e[2], e[0], e[1]])

        e_cost_sorted.sort()
```

クラスカル法の実装例

```
def kruskal(V, e_list):
    …
    # Union-Find木を使う
    uf_tree = UnionFind(V)
    # 最小全域木の辺を保持するリスト
    mst = []
```

クラスカル法の実装例

```
def kruskal(V, e_list):
    …
    [距離の小さい辺から順に全部見ていく]:
        [e[1], e[2]が同じグループでないならば]:
            [e[1], e[2]を同じグループにする]
            # 最小全域木に追加
            mst.append([e[1], e[2]])
```

クラスカル法の実装例

```
def kruskal(V, e_list):
    …

    # ソートして表示
    mst.sort()
    print(mst)
```

クラスカル法の実行例

```
edges_list = [[0, 1, 5], [0, 2, 4], [1, 0, 5], [1, 3, 3], [1, 5, 9],
[2, 0, 4], [2, 3, 2], [2, 4, 3], [3, 1, 3], [3, 2, 2], [3, 6, 7],
[3, 7, 5], [4, 2, 3], [4, 6, 8], [5, 1, 9], [6, 3, 7], [6, 4, 8],
[6, 7, 1], [7, 3, 5], [7, 6, 1]]

kruskal(8, edges_list)

=== 実行結果 ===

[[0, 2], [1, 3], [1, 5], [2, 3], [2, 4], [3, 7], [6, 7]]
```

クラスカル法の計算量

隣接リストの場合，辺の数を$|E|$として，辺のソートに$O(|E|\log|E|)$かかる．

隣接行列の場合はすべての辺を取り出すところで$O(|V|^2)$かかる．（$|V|$はノードの数）

クラスカル法の計算量

各辺を入れるかどうかの判断はUnion-Find木を使うと，
$O(\alpha(|V|))$となり，これを$O(|E|)$回やるので，$O(|E|\alpha(|V|))$.

よって，アルゴリズム全体では$O(|E|\log|E|)$.

もし，各辺を入れるかどうかの判断を単純に毎回BFSやDFSで辿るとすると$O(|V|^2)$となり，ボトルネックとなりうる．

最小全域木のアルゴリズム

辺ベースのアプローチ：クラスカル法

存在する辺を距離の短い順に並べて順に入れていき，
閉路が出来ないことが確認できた場合は追加し，
全部の辺をチェックしたら終了．

ノードベースのアプローチ：プリム法

すでに到達した頂点の集合からまだ到達していない
頂点の集合への辺のうち，距離が最短のものを追加し，
全ノードつながったら終了．

プリム法（Prim）

#1 最初のノードを1つ選び（どれでも可），訪問済にする．

#2 そのノードに繋がっている全ての辺を取り，最小全域木の候補の辺に入れる．

プリム法（Prim）

#3 最小全域木の候補の辺の中から，接続先のノードが未訪問である最短の距離の辺を選ぶ．（接続先のノードが訪問済の場合は無視して次候補に移る．）

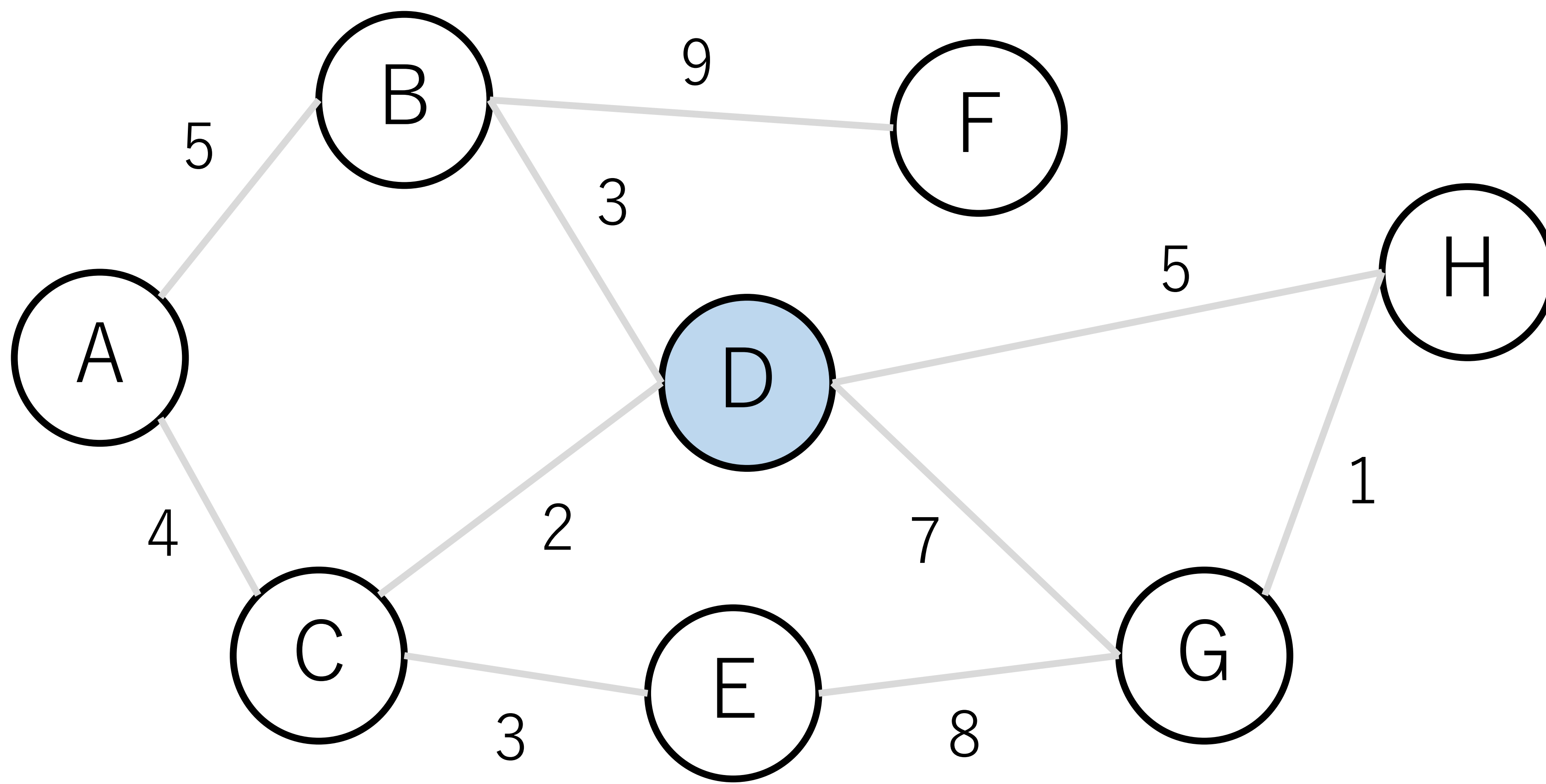
#4 選んだ辺を最小全域木に入れ，その接続先にあるノードを訪問済にする．

#5 #4で新しく訪問したノードから，更にその先につながっている辺のうち，接続先のノードが未訪問の全ての辺を最小全域木の候補に入れる．

#6 以降，全ノードが訪問済になるまで#2～#4を繰り返す．

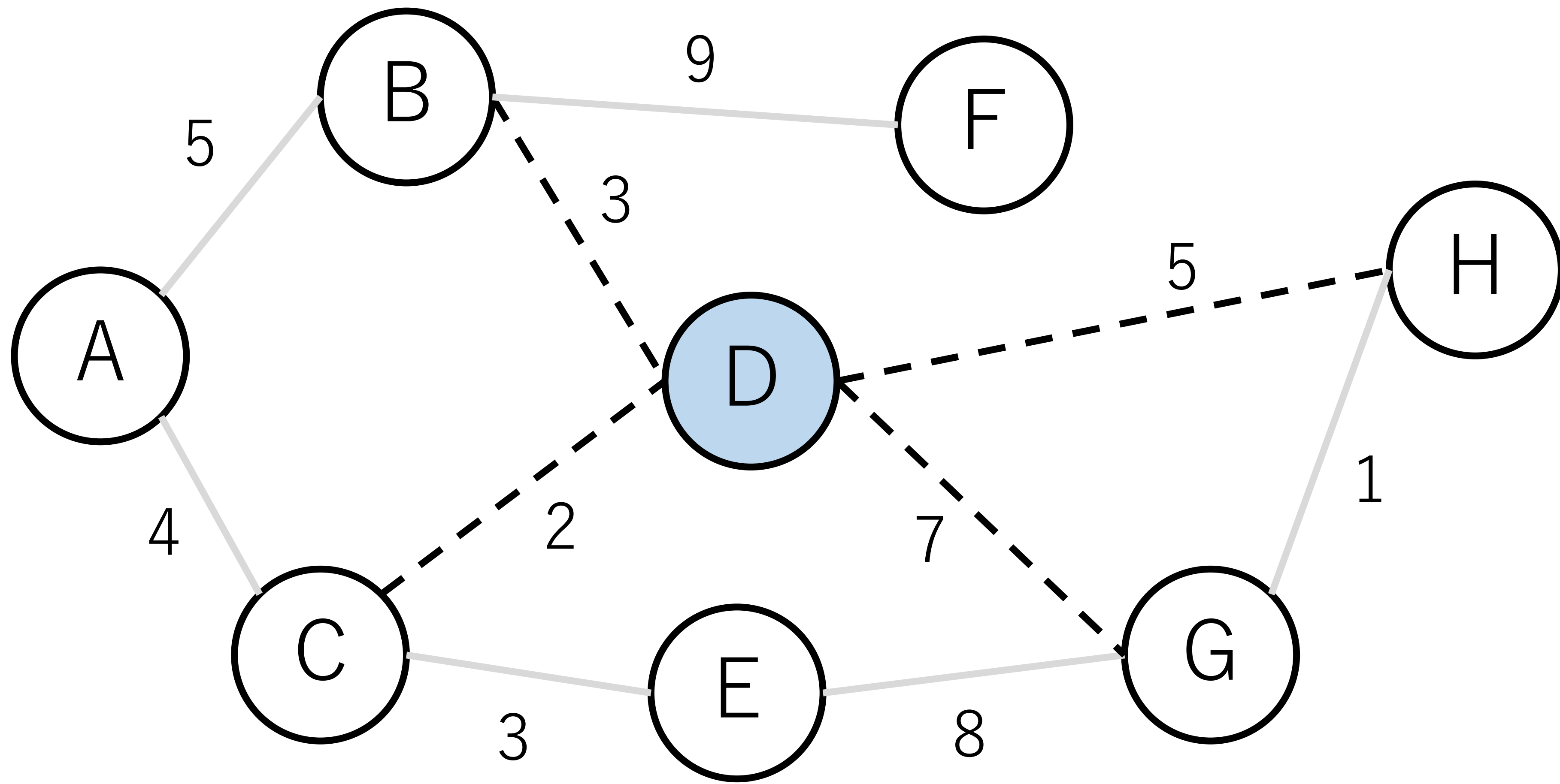
プリム法の例

Dからスタート．（どのノードから始めても良い）



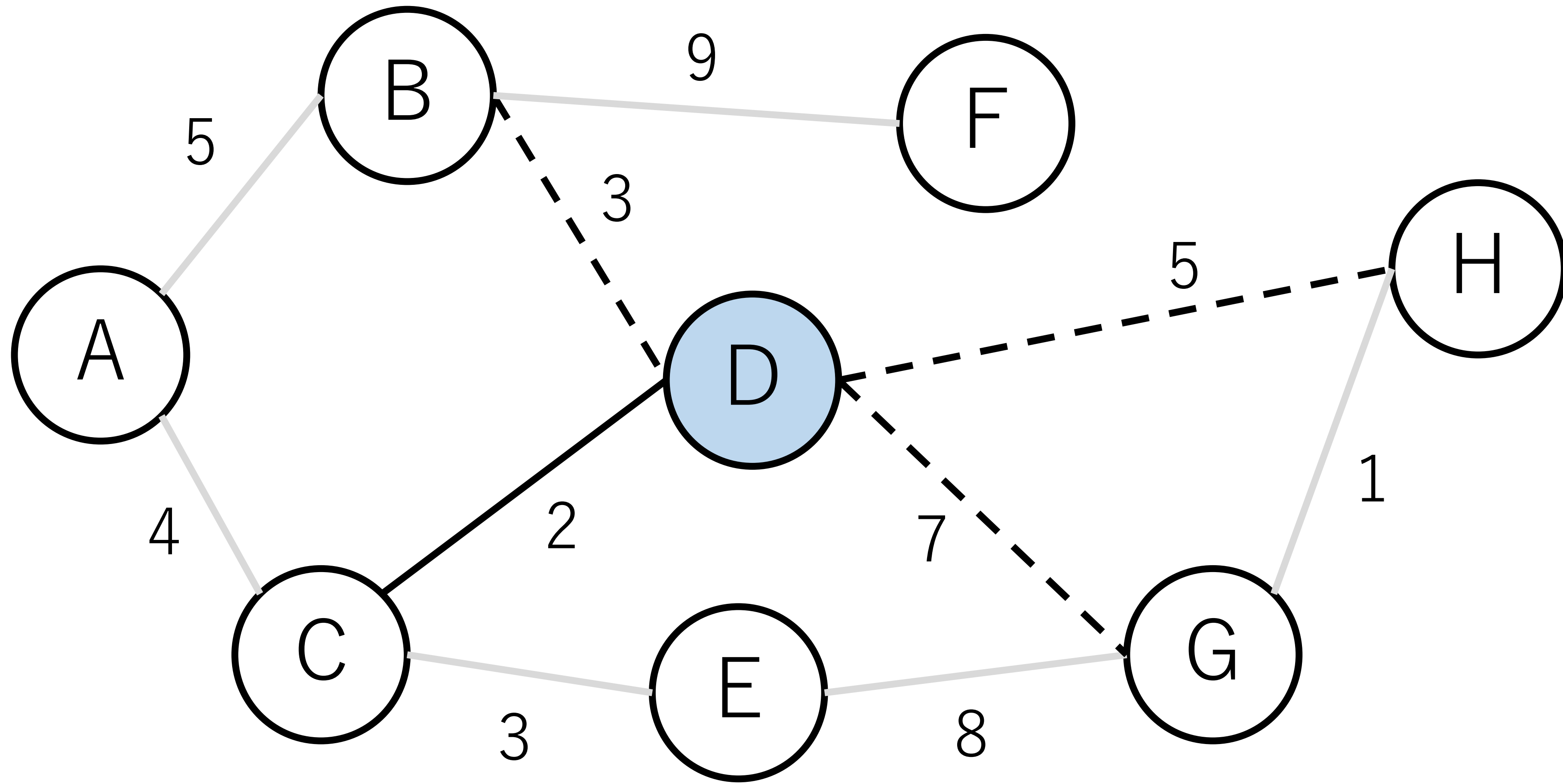
プリム法の例

最小全域木の候補の辺は点線で示すもの.



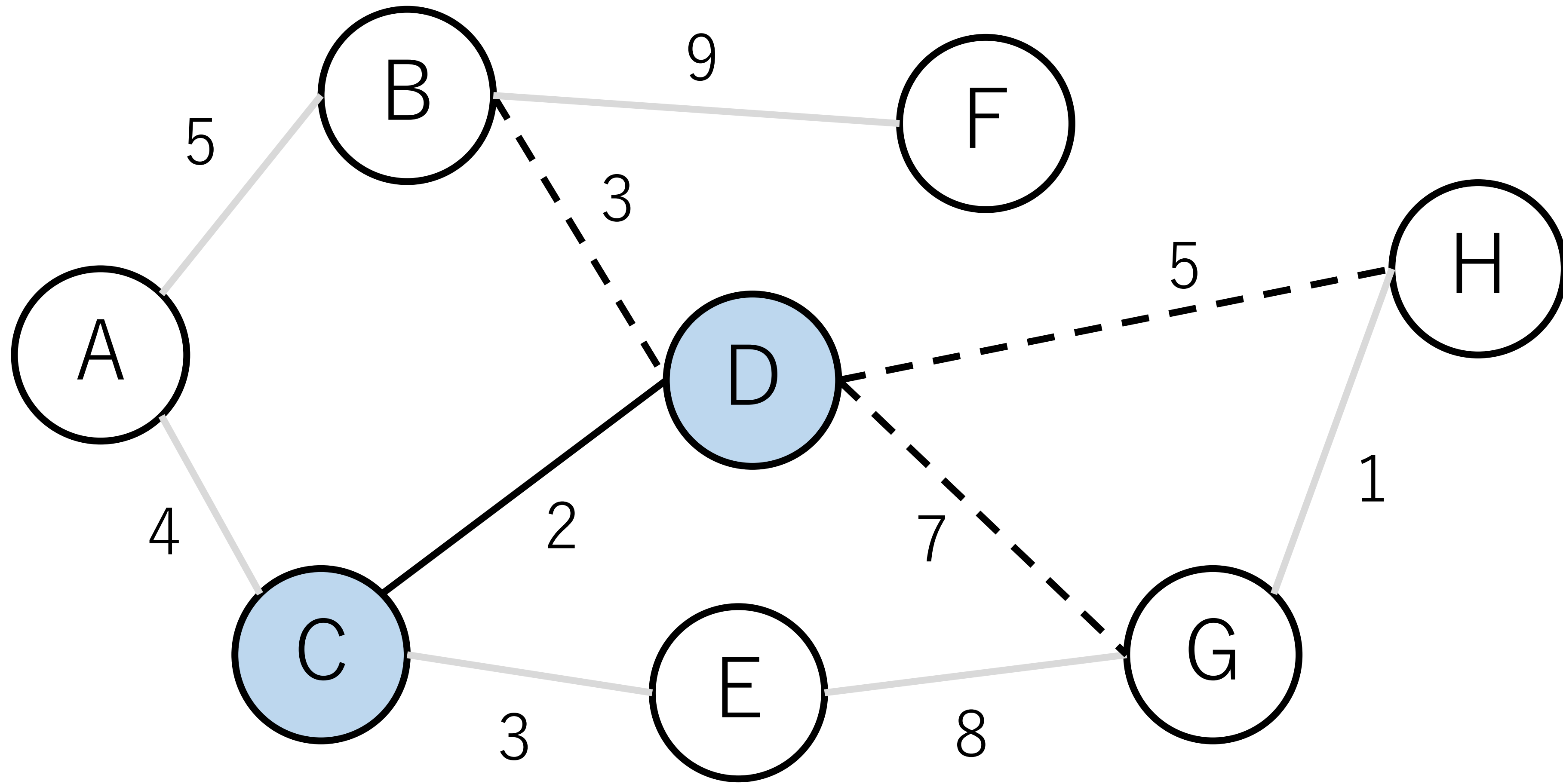
プリム法の例

C-Dの辺が最短なので，この辺を入れてCとつなぐ．



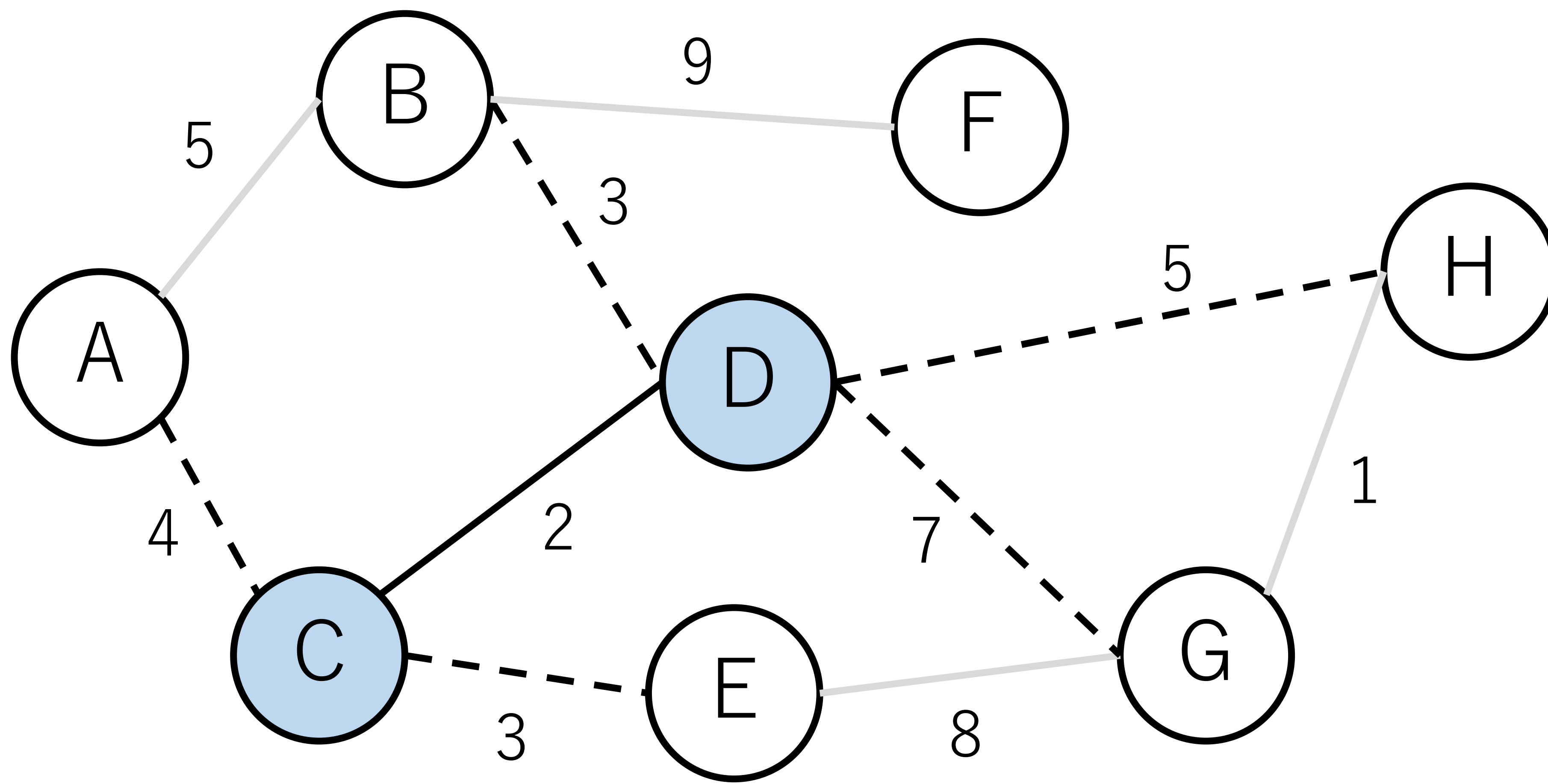
プリム法の例

CとDが「訪問済みグループ」になる．



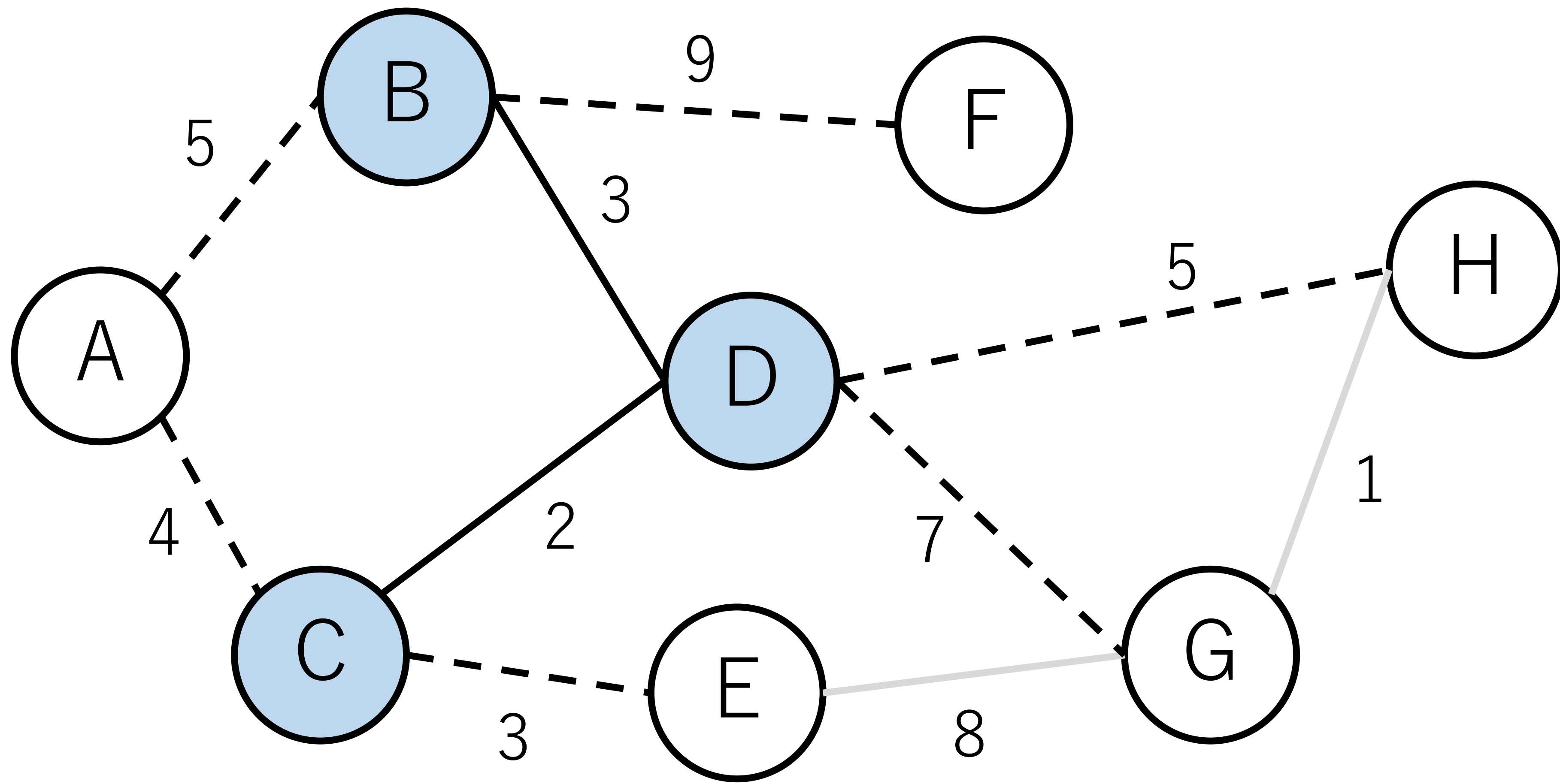
プリム法の例

次に青色ノードから白色ノードにつながっている辺で最短の距離のものを探す.



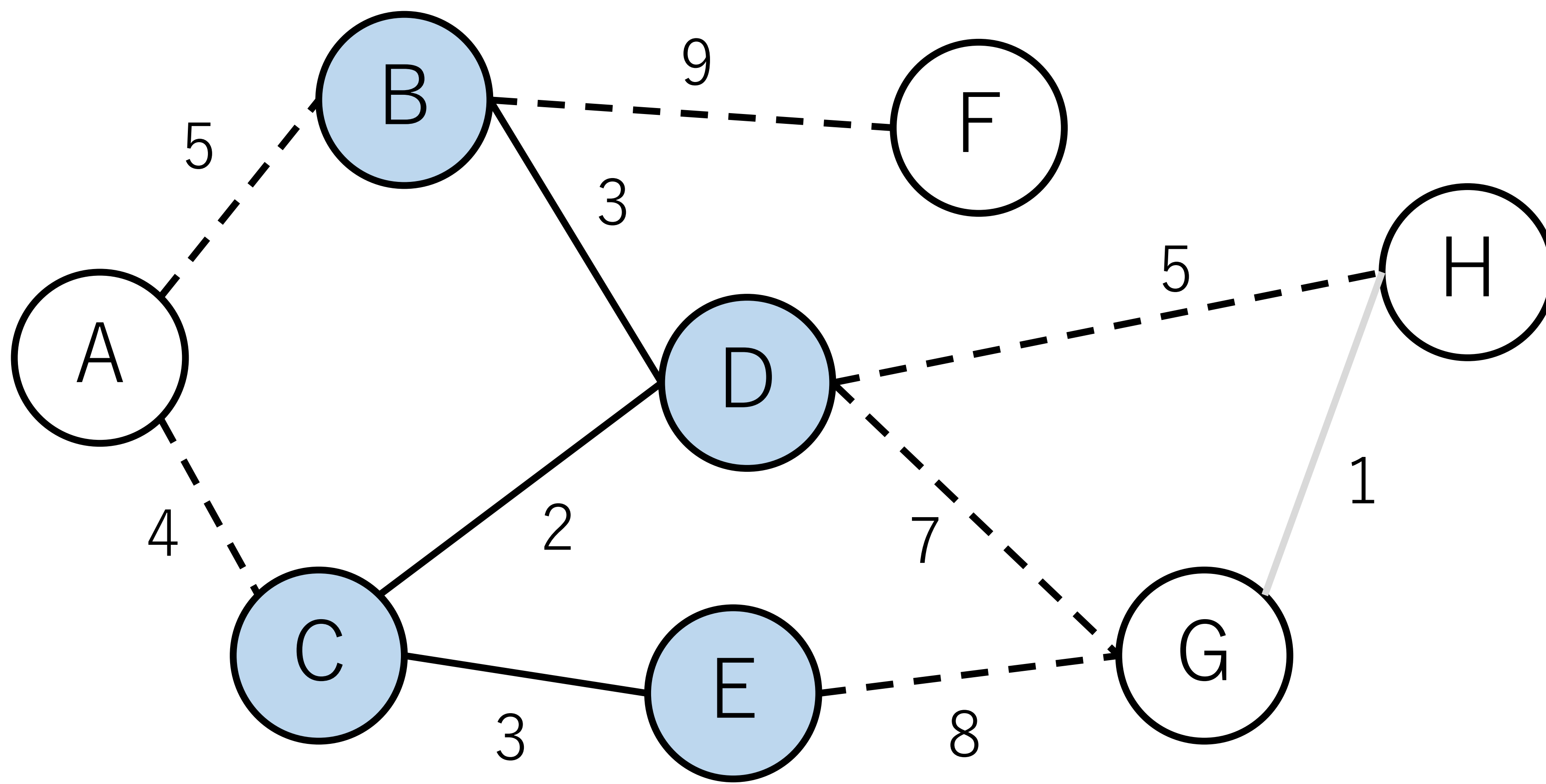
プリム法の例

最短距離は3（B->DとC->E）．ここではB-Dをつなぐ.



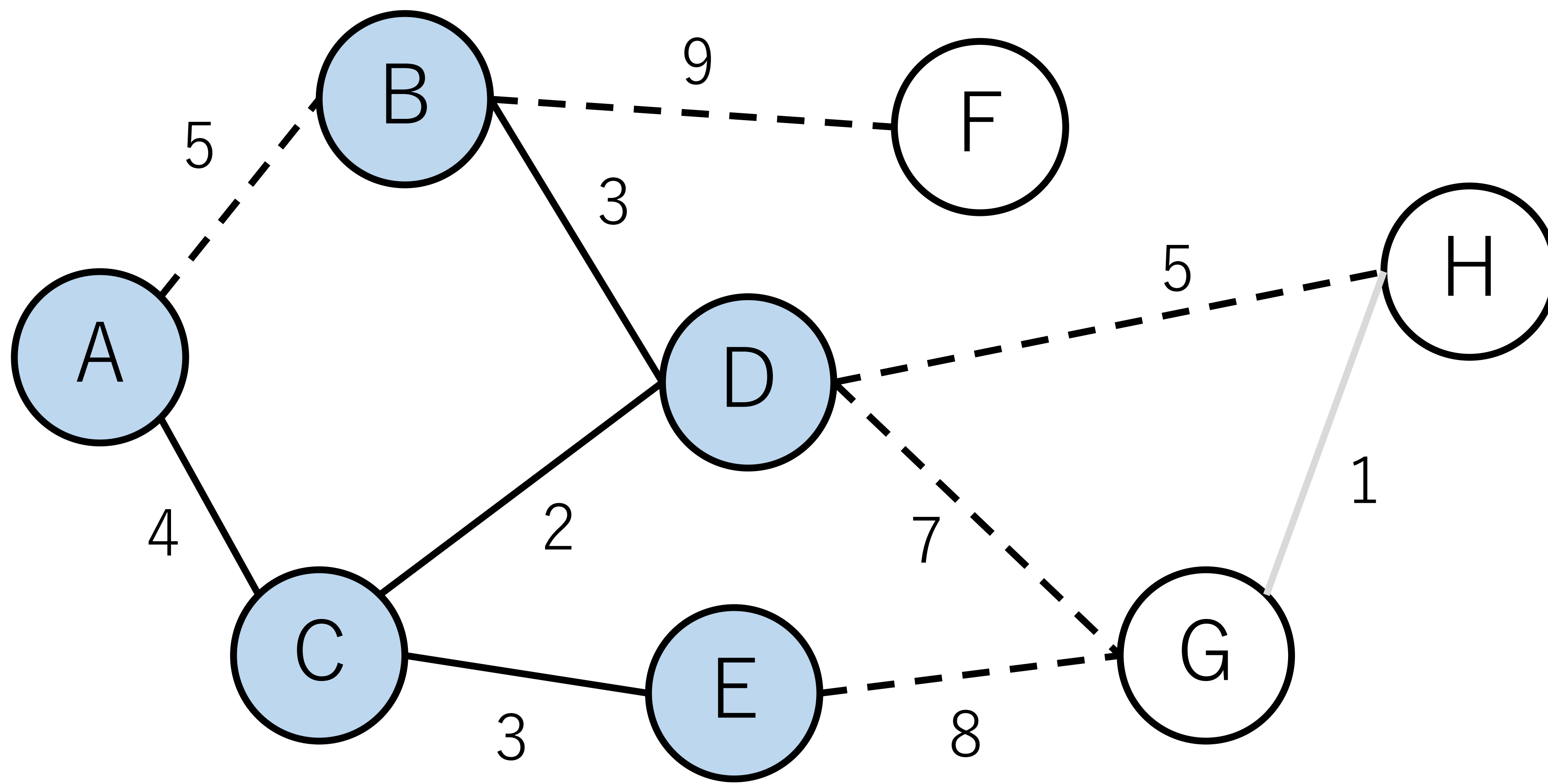
プリム法の例

次に最短距離のものはC-E.



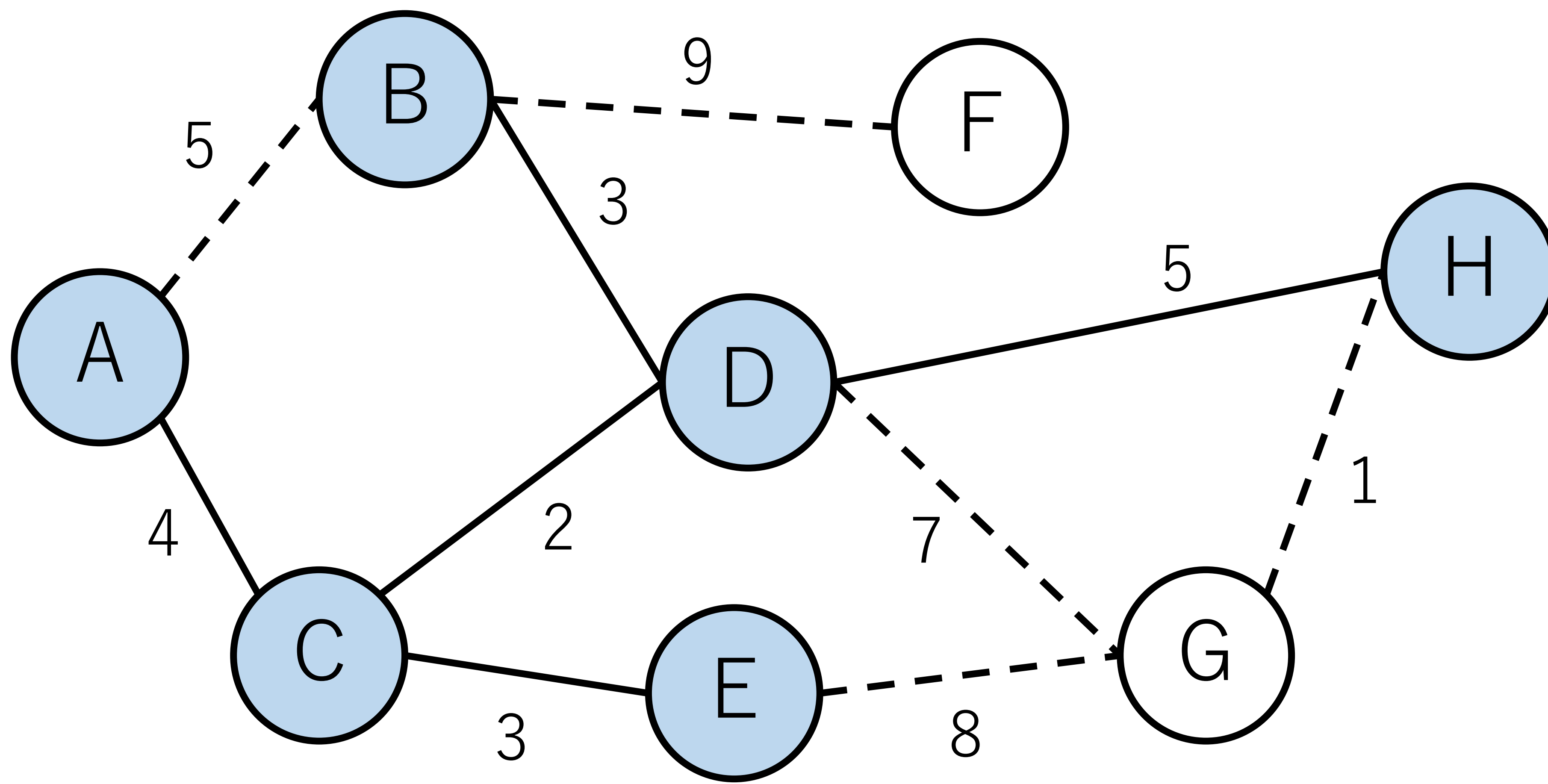
プリム法の例

その次は，A-C.



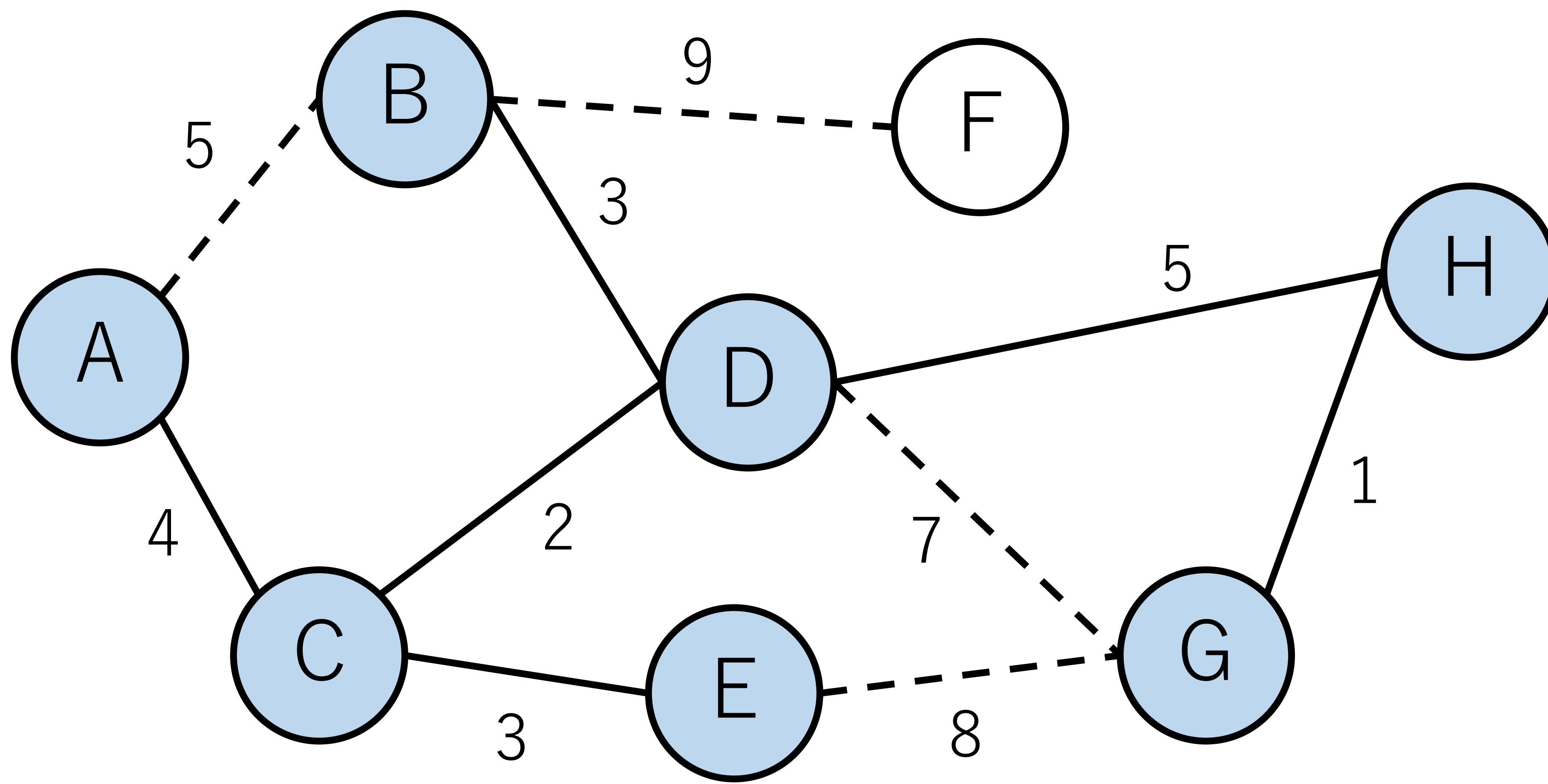
プリム法の例

その次は，D-H.



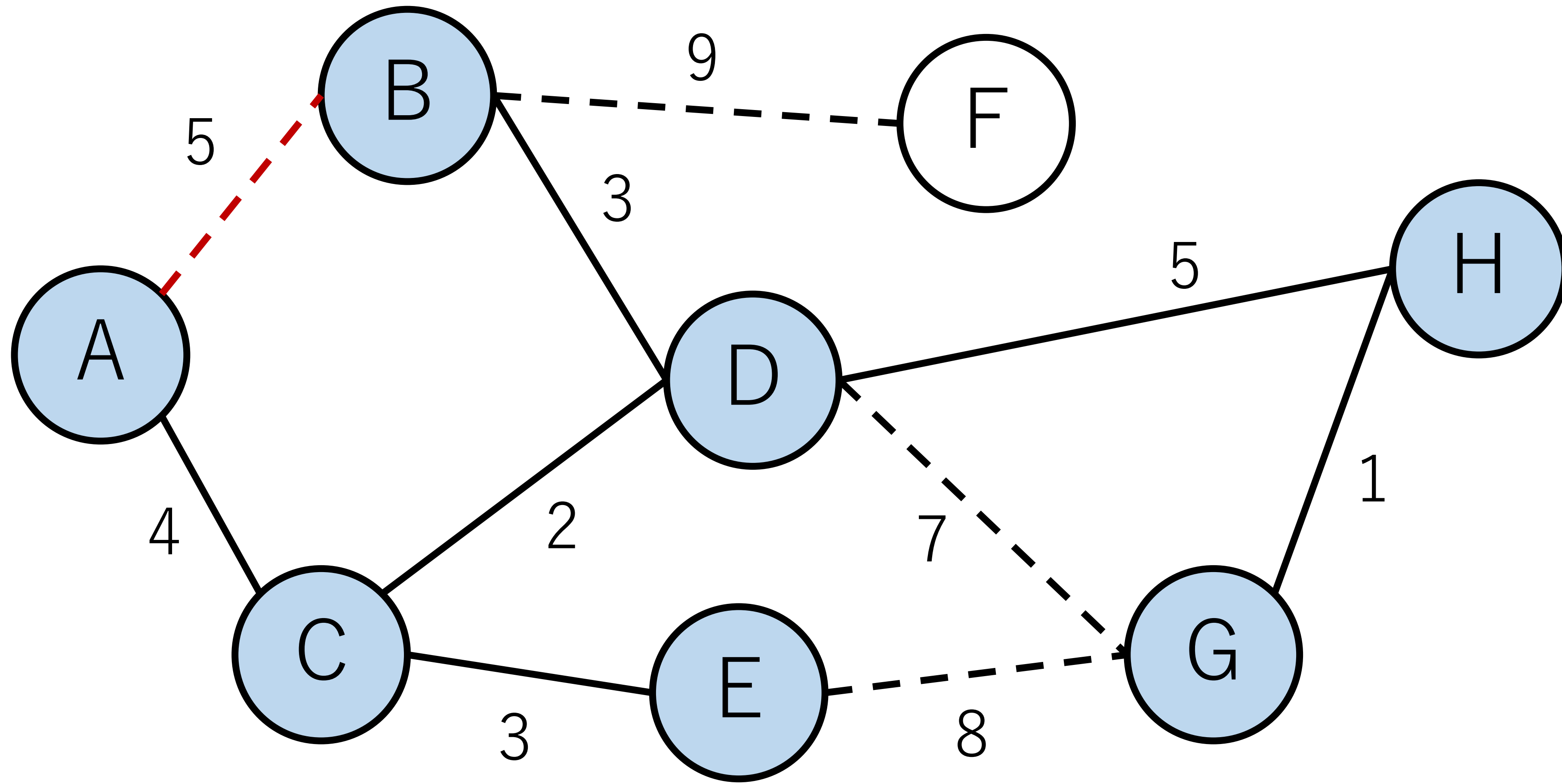
プリム法の例

その次は，G-H.



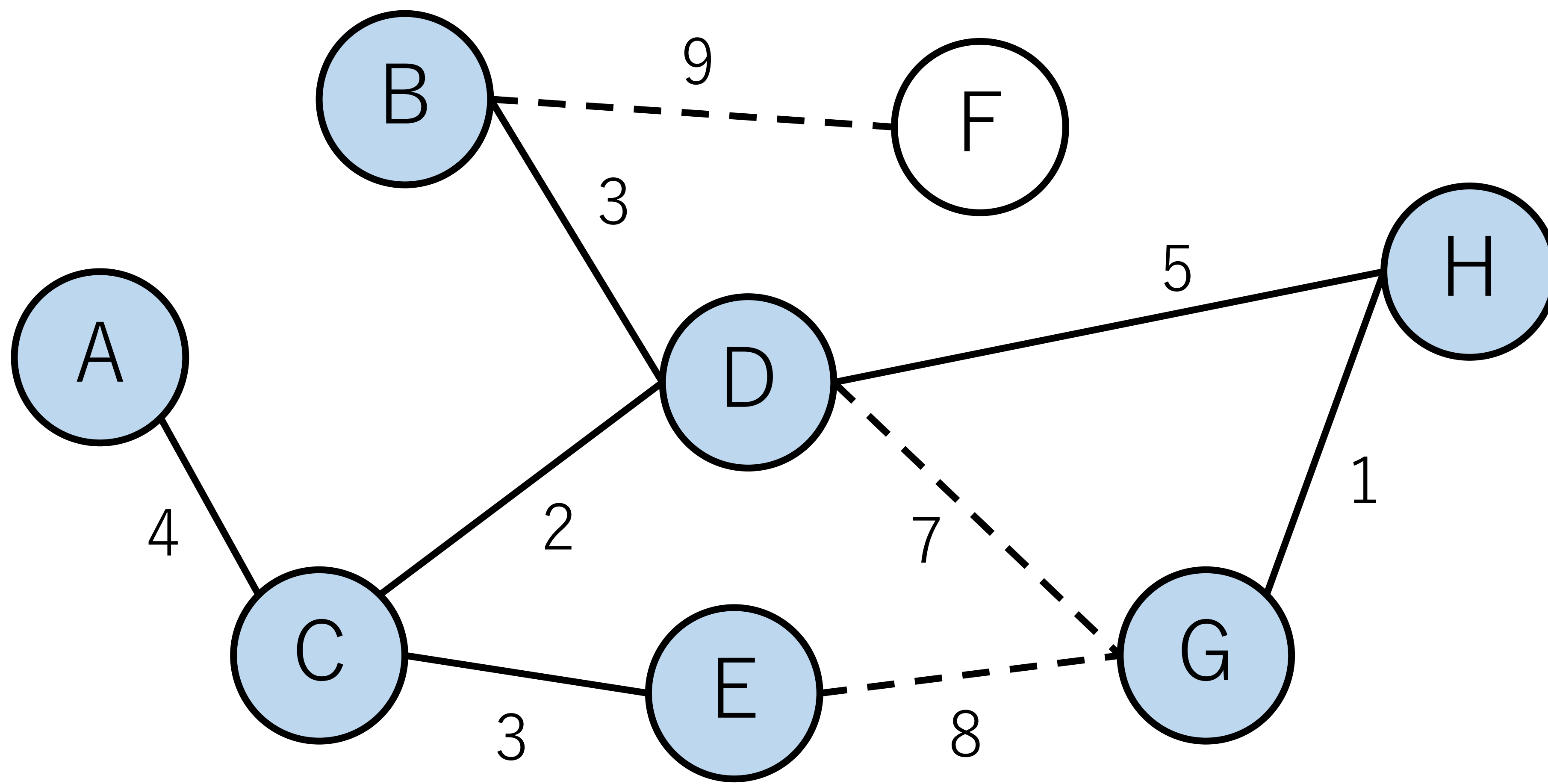
プリム法の例

その次は，A-Bだが，どちらのノードもすでに訪問済なので
この辺はスキップする．



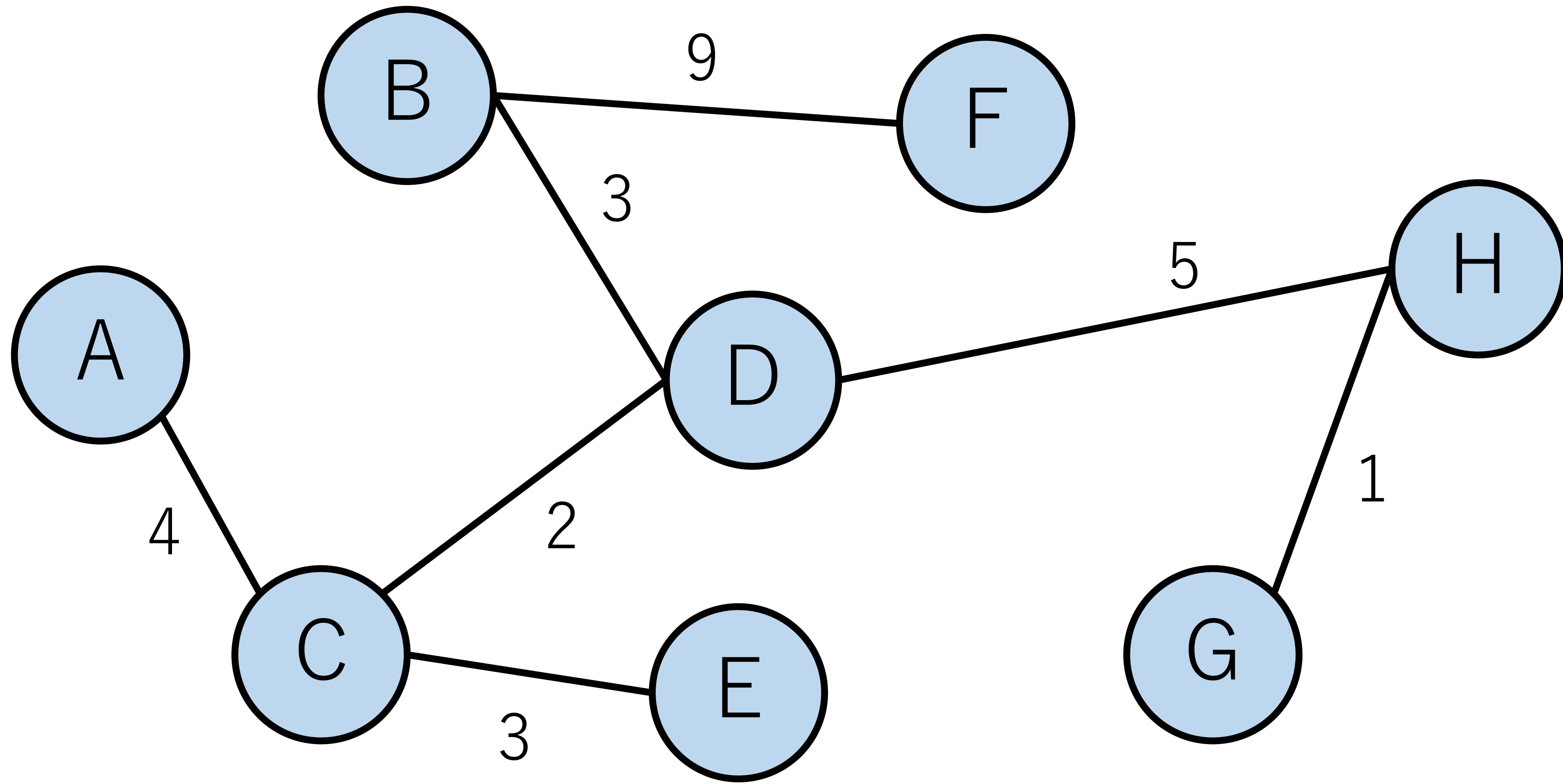
プリム法の例

D-G，E-Gについても同様にスキップ．



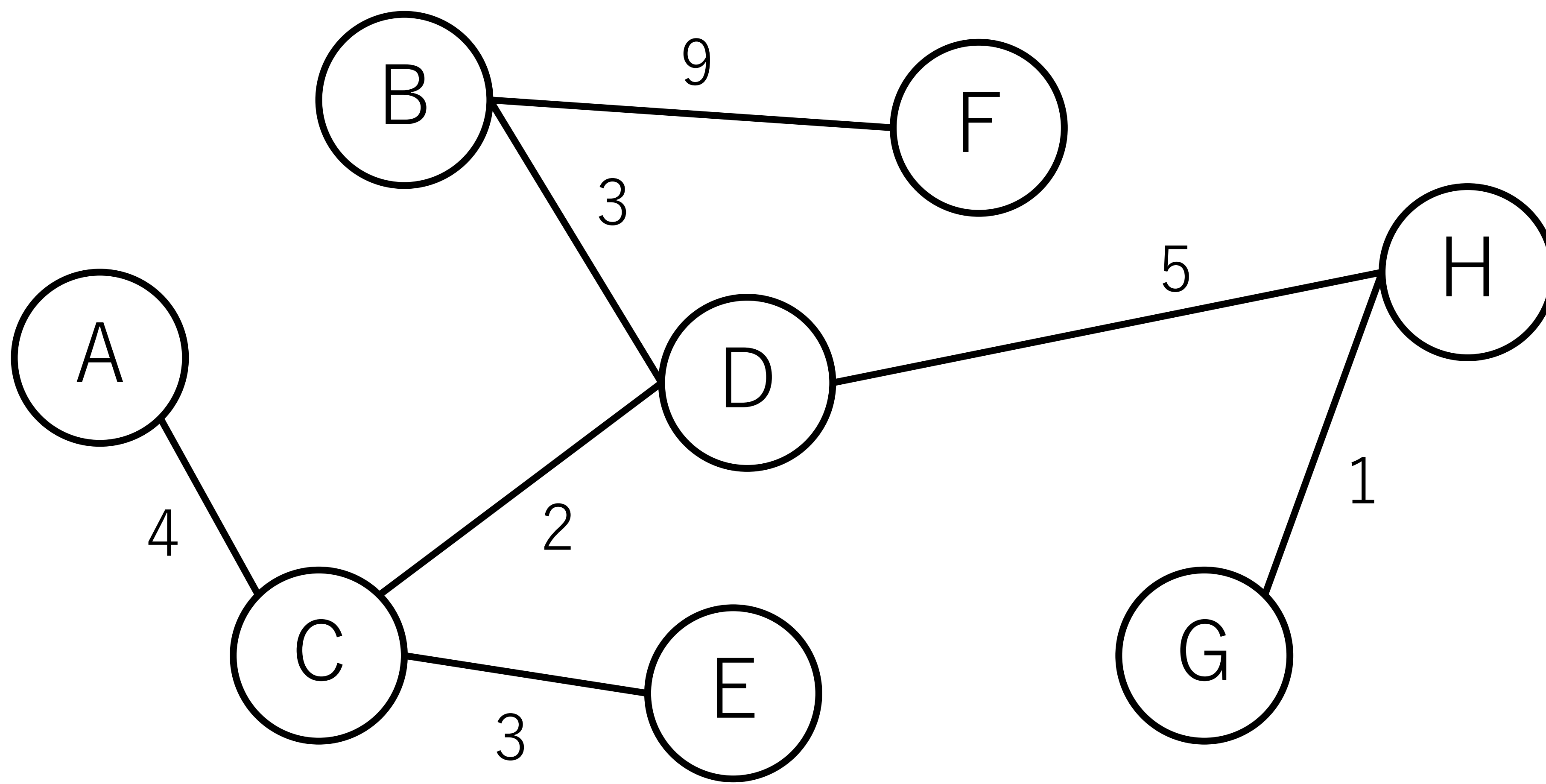
プリム法の例

B-FはFが未訪問だったので，最小全域木に入れる.



プリム法の例

これで終了.



プリム法の計算量

ノードの数を$|V|$として,

隣接行列＋最短の辺の単純な探索：$O(|V|^3)$

最短の辺の探索に$O(|V|^2)$，それが$O(|V|)$回.

隣接リスト＋最短の辺の単純な探索：$O(|E||V|)$

プリム法の計算量

ただし，ダイクストラのときのようにデータ構造を
工夫することで高速化できる．

以下の実装例では，隣接リスト＋ヒープを使っている．

プリム法の実装例

```
import heapq

def prim(V, e_list):
    # edges_from[i]はノードiからのすべての辺を格納
    edges_from = [[] for _ in range(V)]

    # ヒープでソートされるために距離を最初の要素にする
    for e in e_list:
        edges_from[e[0]].append([e[2], e[0], e[1]])
```

プリム法の実装例

```
def prim(V, e_list):
    …
    e_heapq = []            # ヒープ
    mst = []            # 最小全域木
    # ノードが最小全域木に入ったかどうかのフラグ
    included = [False]*V
```

プリム法の実装例

```
def prim(V, e_list):
    …
    # ノードをまず1つ選ぶ．何でも良いがこの実装では
    # ノード0を選ぶことにする．
    included[0] = True
    # ノード0に接続する辺を全てヒープに入れる．
    for e in edges_from[0]:
        heapq.heappush(e_heapq, e)
```

プリム法の実装例

```
def prim(V, e_list):
        …
        while e_heapq:
                [e_heapqの中から最短の辺を取り出す]

                if [その辺の到達先（ノードj）が未訪問なら]:
                        [ノードjを訪問済にする]
                        [mstにその辺を入れる]
                        [ノードjから伸びる全辺をe_heapqに入れる]
```

プリム法の実装例

```
def prim(V, e_list):
    …
    while len(e_heapq):
        min_edge = heapq.heappop(e_heapq)
        #その辺の到達先（ノードj）が未訪問なら追加
        if not included[min_edge[2]]:
            included[min_edge[2]] = True
            mst.append([min_edge[1], min_edge[2]])
```

プリム法の実装例

```
def prim(V, e_list):
    …
    while len(e_heapq):
        …
            #ノードjから伸びる辺をe_heapqに入れる
            for e in edges_from[min_edge[2]]:
                if not included[e[2]]:
                    heapq.heappush(e_heapq, e)
```

プリム法の実装例

```
def prim(V, e_list):
    …

    # ソートして表示
    mst.sort()
    print(mst)
```

プリム法の実行例

```
edges_list = [[0, 1, 5], [0, 2, 4], [1, 0, 5], [1, 3, 3], [1, 5, 9],
[2, 0, 4], [2, 3, 2], [2, 4, 3], [3, 1, 3], [3, 2, 2], [3, 6, 7],
[3, 7, 5], [4, 2, 3], [4, 6, 8], [5, 1, 9], [6, 3, 7], [6, 4, 8],
[6, 7, 1], [7, 3, 5], [7, 6, 1]]
prim(8, edges_list)
=== 実行結果 ===
[[0, 2], [1, 5], [2, 3], [2, 4], [3, 1], [3, 7], [7, 6]]
```

プリム法の計算量（ヒープを使う場合）

上記の実装では，ヒープに入る要素の数は辺の総数になるので，$O(|E|)$.

よって，追加，削除にかかる計算量は$O(\log|E|)$.

ヒープへの追加も取り出しも$O(|E|)$回あるので，全体では$O(|E|\log|E|)$となる.

（ダイクストラ法のときに説明したとおり，$O(\log|E|)$は$O(\log|V|)$と等価であるとも考えられるので，$O(|E|\log|V|)$と説明される場合もある．）

プリム法の計算量

ダイクストラ法と同じように，フィボナッチヒープを使うことにより，$O(|E| + |V| \log|V|)$に落とせることが知られている．

今日のテーマ

最小全域木

トポロジカルソート

トポロジカルソート

閉路の無い有向グラフを「ソート」する.

- 全ての有向辺が1つの向きになるようにノードを並び替える.

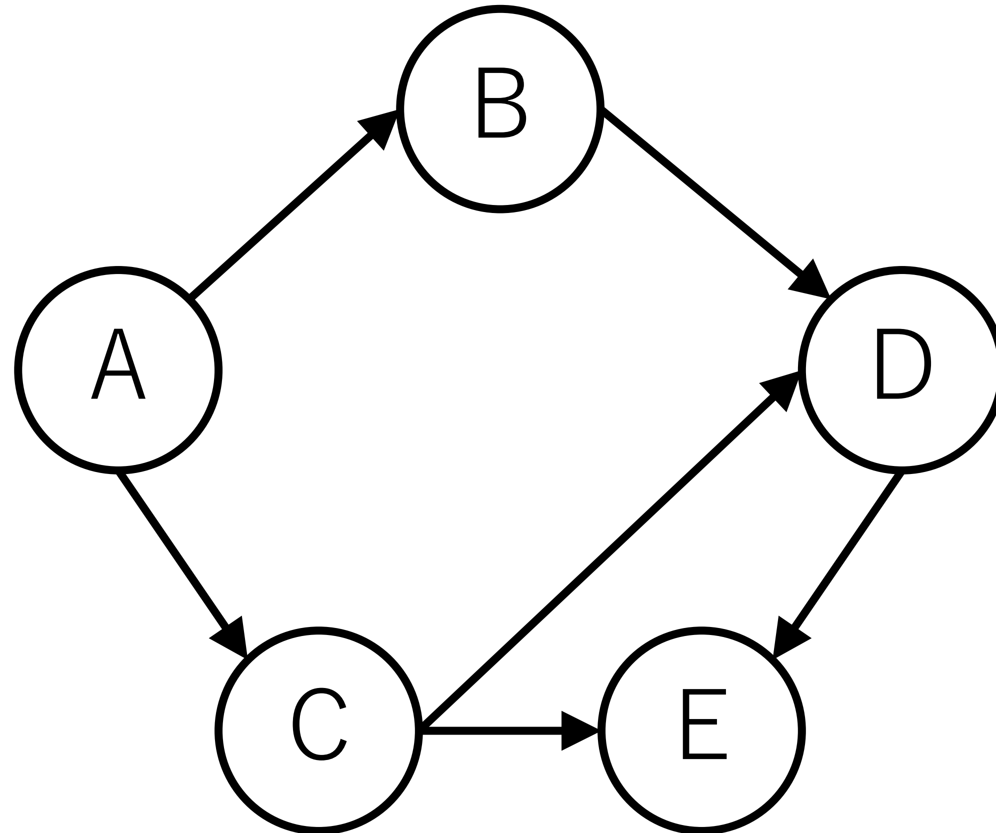
このようなグラフを有向非巡回グラフと呼ぶ.

- 英語ではDirected Acyclic Graph（DAG）.

また，与えられるDAGには多重辺がないとする.

DAG

DAGの例．巡回できる経路は存在しない．



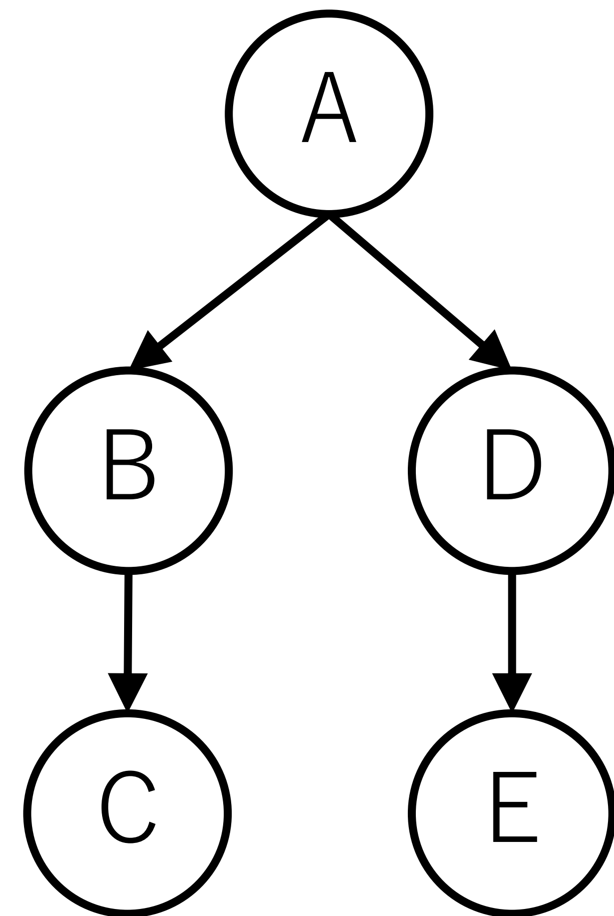
有向木

根である点をただ1つ持ち，辺の向きが根から葉，もしくは葉から根のどちらかに限られる．

辺の向きが全て同じでない場合，「任意の2点を結ぶ経路がただ1つ存在する」という木の制約を満たさない．

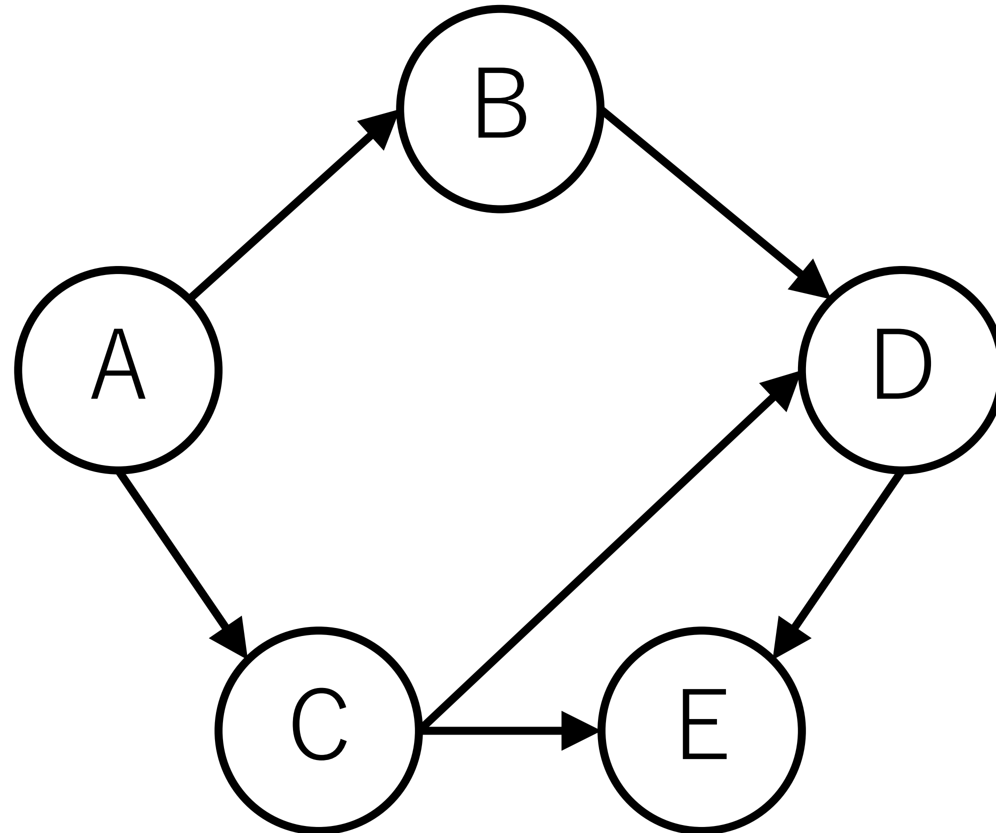
有向木はDAGである．

向きがあって，巡回できる経路がない．



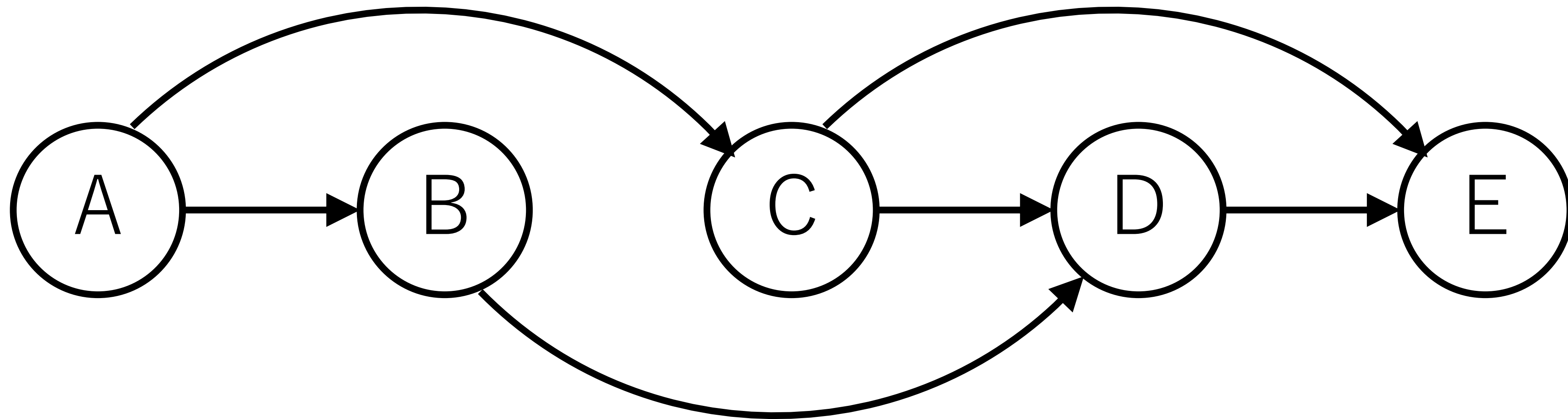
DAG

DAGは必ずしも有向木ではない．この例では，DやEに至る経路が複数存在してしまっている．



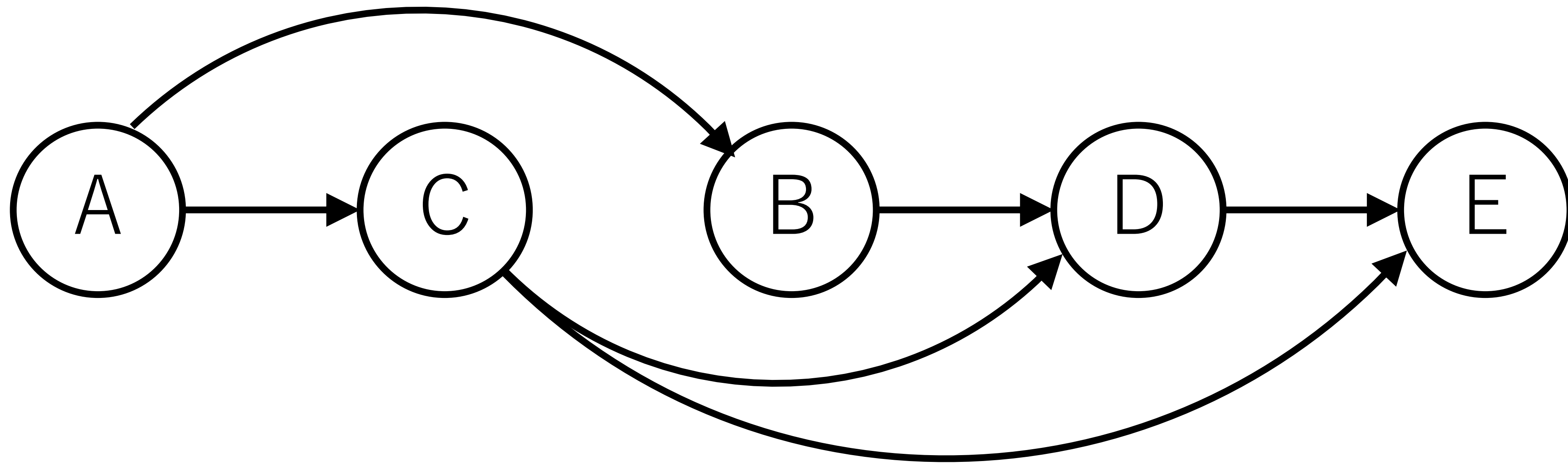
トポロジカルソート例

すべての辺が右方向に向くようにノードを並べ替える
ことができる．



トポロジカルソート例

トポロジカルソートの結果は1つとは限らない.



次数，入次数

次数（degree）：あるノードにつながっている辺の総数．
入次数（indegree）：あるノードに入ってくる辺の総数．

言葉としては，出次数（outdegree）もある．
あるノードから出ていく辺の総数．

入次数，出次数はそれぞれ「いりじすう」，「でじすう」と読むのが正式だそうです．
http://dopal.cs.uec.ac.jp/okamotoy/lect/2014/gn/term01.pdf

次数，入次数

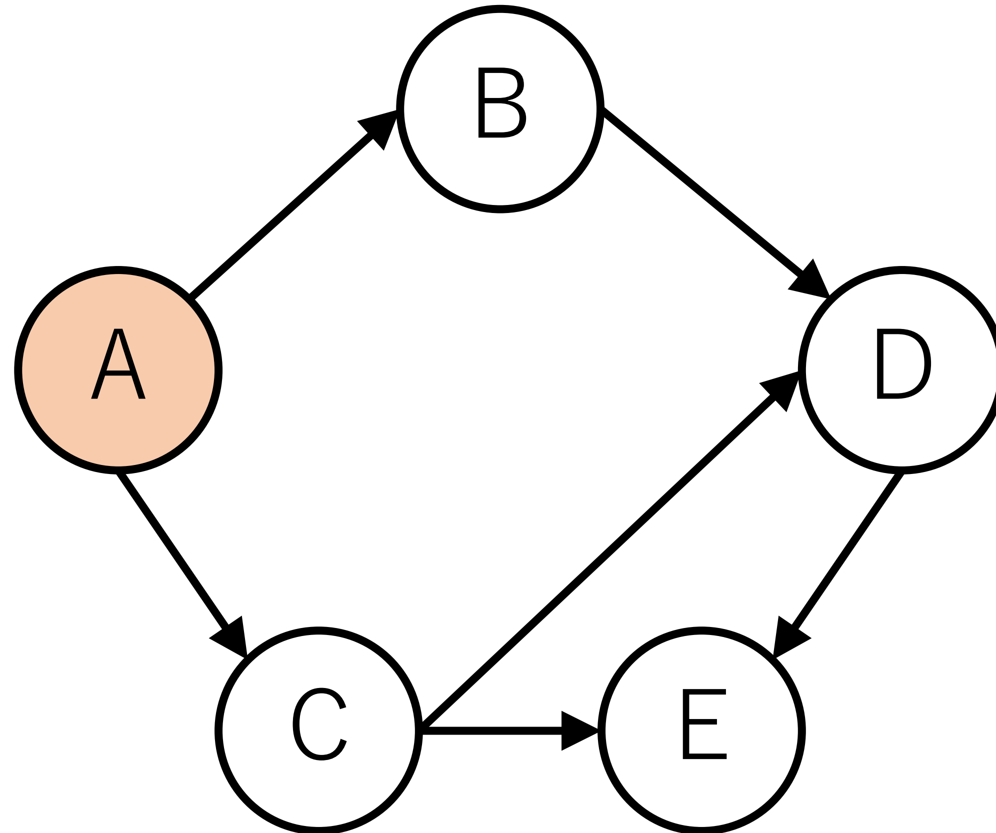
[次数] = [入次数] + [出次数]

自分自身へのループは入次数，出次数ともに1ずつカウントされる．

よって，自分自身へのループ1つに対して，次数は2つ増える．

DAG

DAGには必ず，入次数0のノードが最低1つ存在する．
存在しなければ閉路が存在し，DAGにならない．



トポロジカルソート

代表的なものは2つ.

Kahnさんが提案したもの.

Tarjanさんが提案したもの.

今日はこの2つを順に紹介をしていきます.

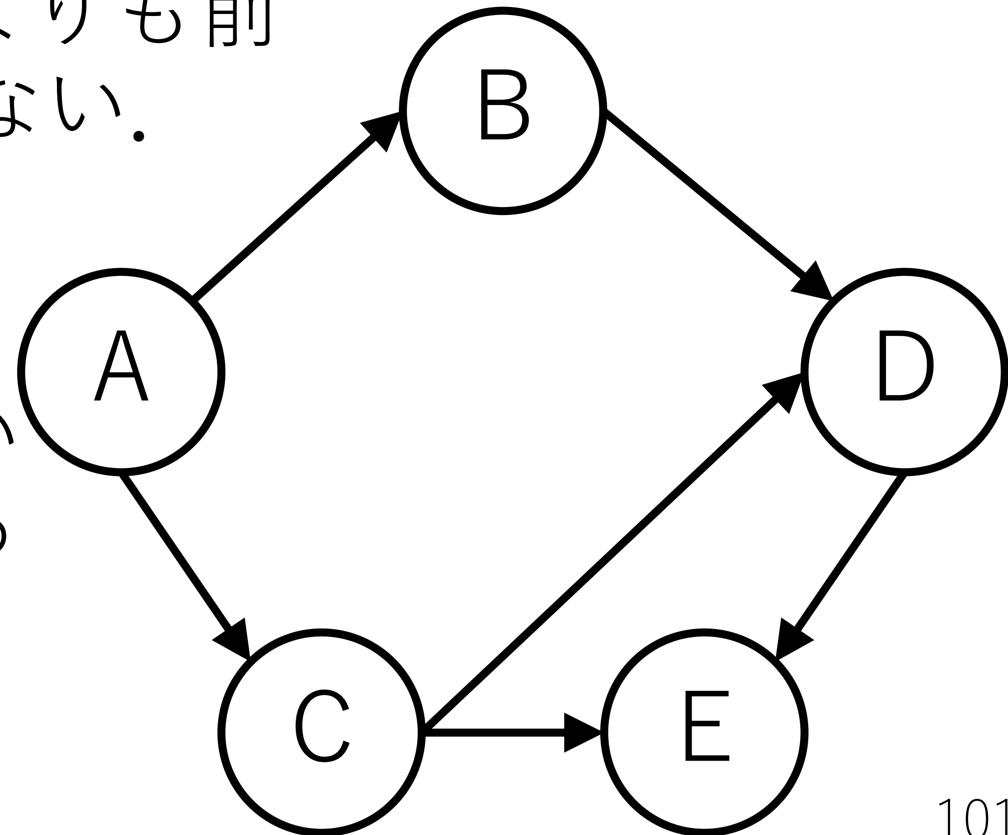
Kahn, A. B. (1962). Topological sorting of large networks. Communications of the ACM, 5(11), 558-562.
Tarjan, R. E. (1976). Edge-disjoint spanning trees and depth-first search. Acta Informatica, 6(2), 171-185.

Kahnのトポロジカルソートの方針

入次数0のノードを見つけ出し，それをグラフから取り除き，ソート済の場所に入れていく．

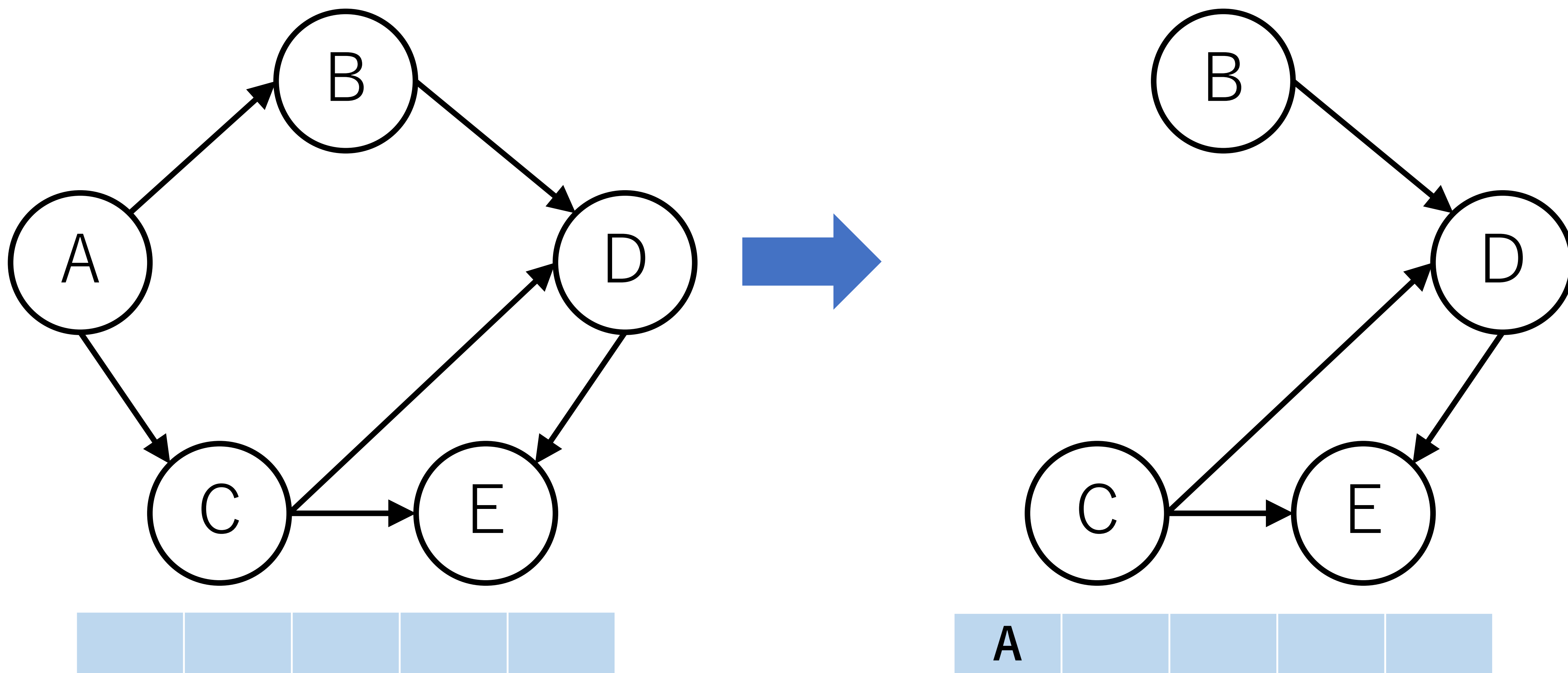
「入次数が0」＝このノードよりも前（左側）に並ぶべきノードはない．

例えば，右のグラフではノードAは，その前段につながるものはないので，ソートした時一番最初に来ることになる．



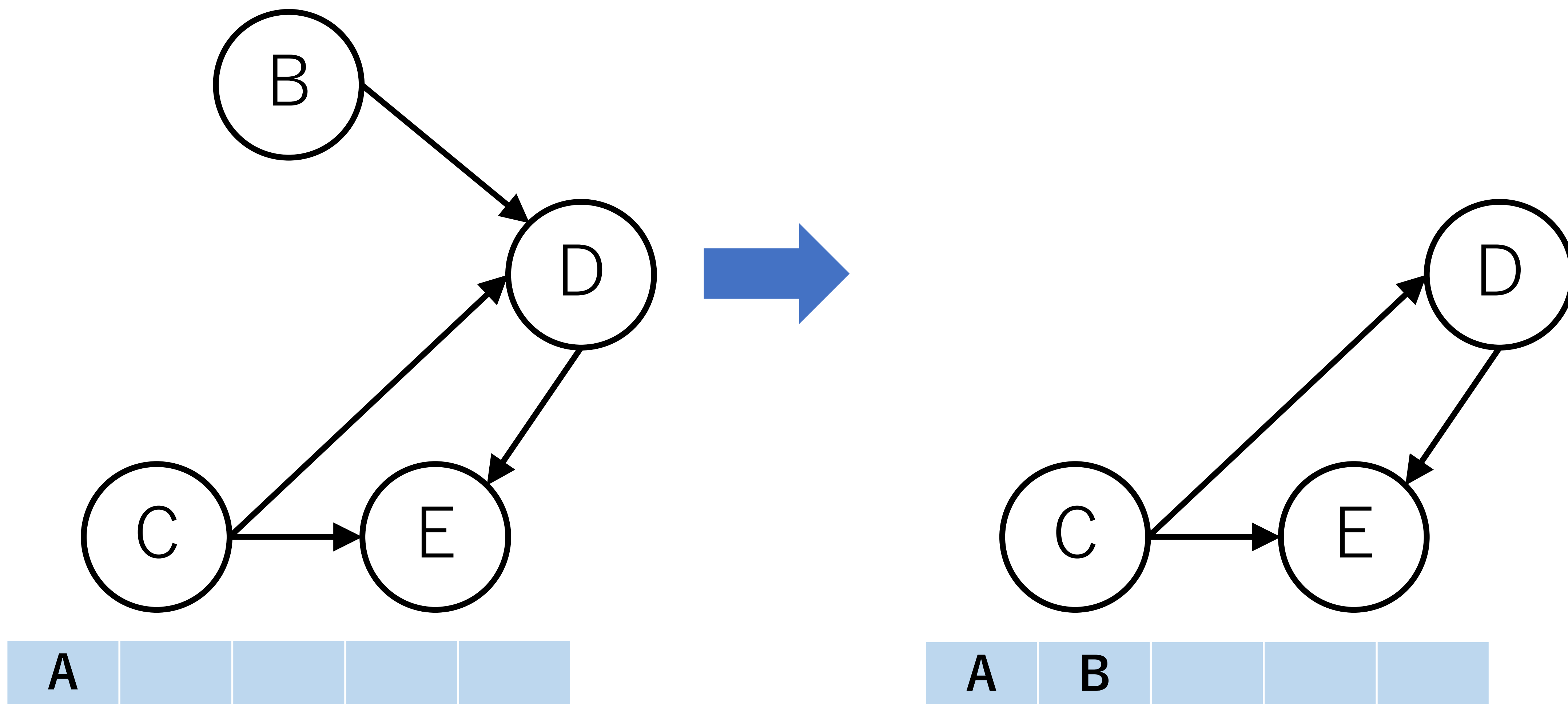
Kahnのトポロジカルソートの実行例

まずノードAを取り出してソート済とする.



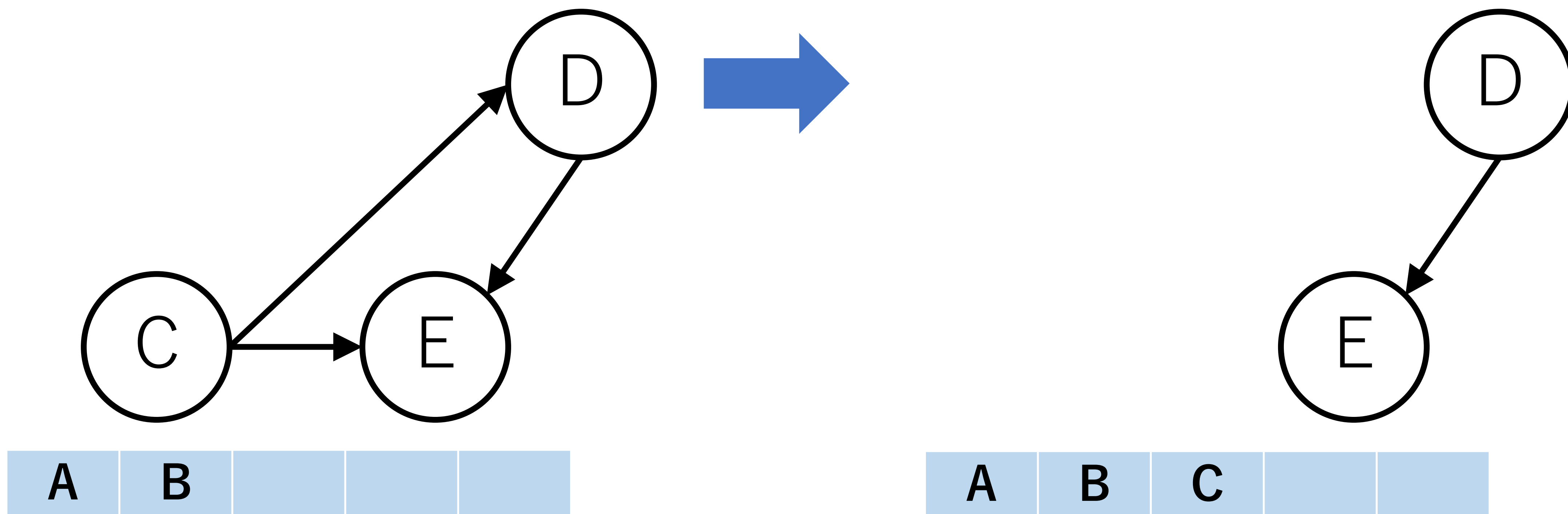
Kahnのトポロジカルソートの実行例

次に入次数が0のノードBを取り出す（ノードCでもよい）．



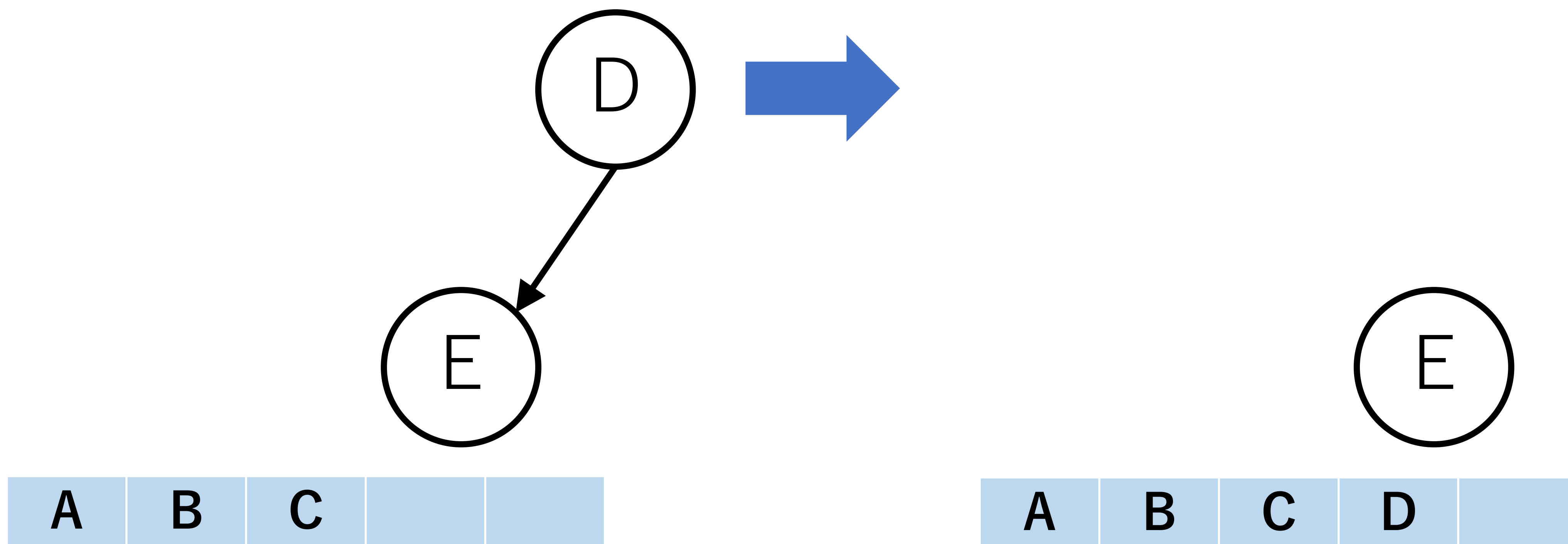
Kahnのトポロジカルソートの実行例

次に入次数が0のノードCを取り出す.



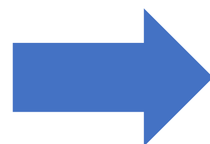
Kahnのトポロジカルソートの実行例

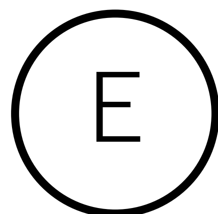
次に入次数が0のノードDを取り出す.

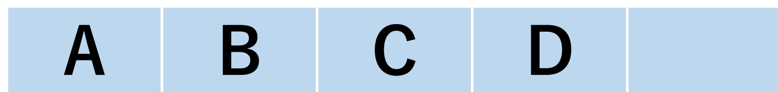


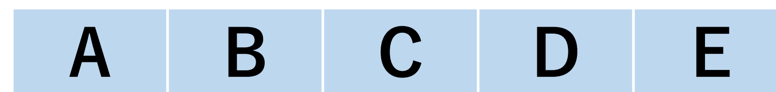
Kahnのトポロジカルソートの実行例

以降，入次数が0となるノードがなくなるまで繰り返す．







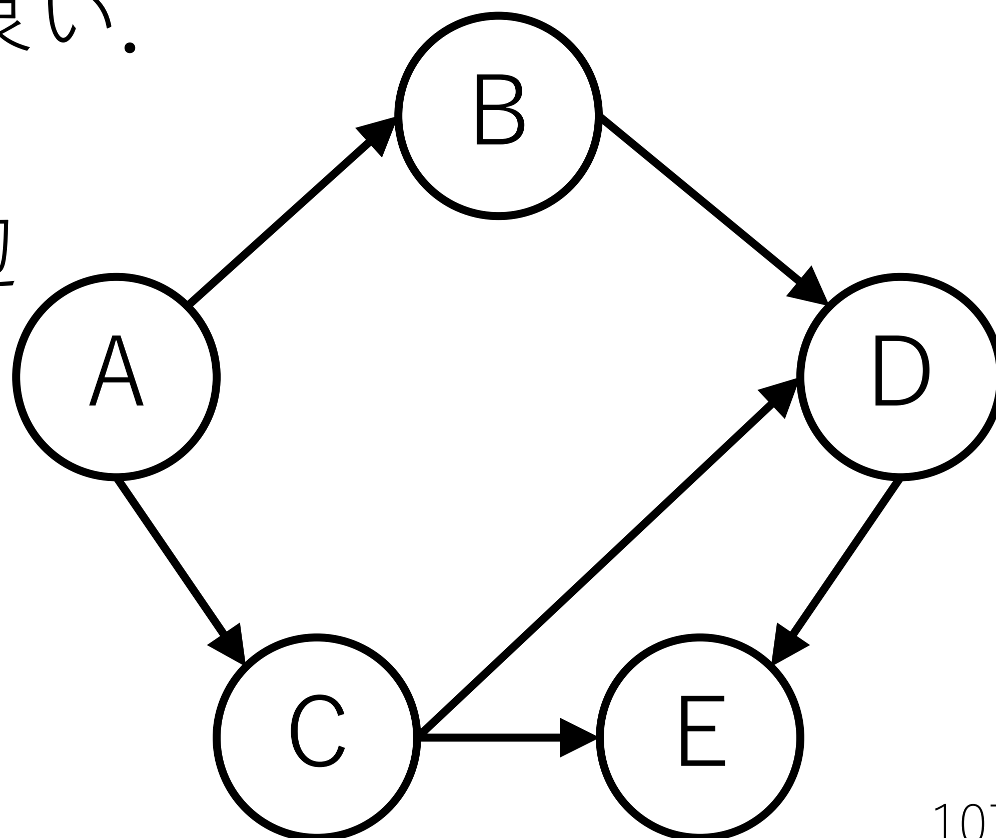


Kahnのトポロジカルソートの実装方針

具体的な実装としては，各ノードの入次数を予め記録しておき，ノードを取り出すたびに入次数の値を更新する．

更新するときには，-1すれば良い．

この時，取り除いたノードの出力辺の接続先ノードの入次数のみ更新すれば良い．



Kahnのトポロジカルソート

whileループを抜けたあと，まだ残っている辺がある場合，
DAGになっていないので，エラーを返す．

Kahnのトポロジカルソートの実装例

```
V = 5       # ノードの総数
E = 6       # 辺の総数
# 有向辺の配列
edges = [[0, 1], [0, 2], [1, 3], [2, 3], [2, 4], [3, 4]]
```

Kahnのトポロジカルソートの実装例

```
from collections import deque

def KahnTopo(V, E, edges):

        indeg = [0]*V   # 入次数を格納する配列
        # 出力辺を保持する配列
        outedge = [[] for _ in range (V)]
```

Kahnのトポロジカルソートの実装例

```
def KahnTopo(V, E, edges):
    …
    # 入次数と出力辺の情報を整理する
    for v_from, v_to in edges:
        indeg[v_to] += 1
        outedge[v_from].append(v_to)
```

Kahnのトポロジカルソートの実装例

```
def KahnTopo(V, E, edges):
    …
    # ソート済のノードを格納する配列
    # 最初に入次数0のものを入れておく
    sorted_g = list(v for v in range(V) if indeg[v]==0)

    # 入次数0のノードを処理するためのdeque
    deq = deque(sorted_g)
```

Kahnのトポロジカルソートの実装例

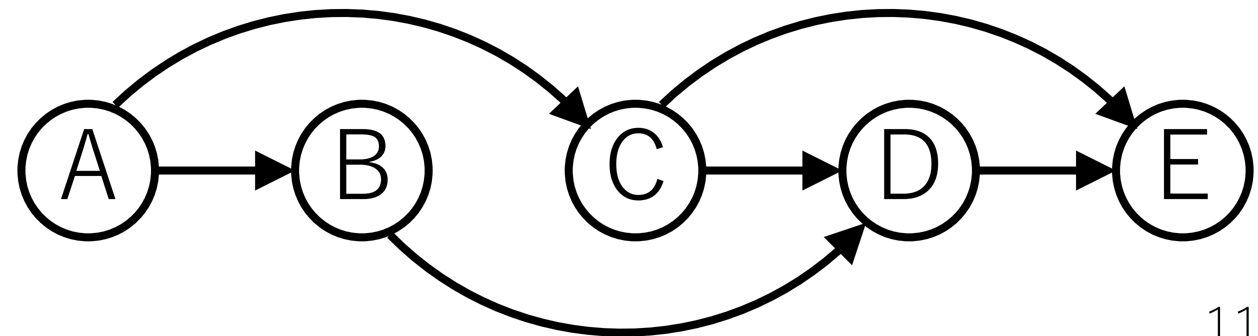
```
def KahnTopo(V, E, edges):
    …
    while deq: #入次数0のノードがある限り繰り返す
        v = deq.popleft()      # deq.pop()でもよい
        [for vからつながるすべてのノードu]:
            [E,  uの入次数を1減らす]
            if [uの入次数が0]:
                [uをdeqとsorted_gに入れる]
```

Kahnのトポロジカルソートの実装例

```
def KahnTopo(V, E, edges):
        …
        if E != 0:
                #DAGになっておらずエラー
                return None

        return sorted_g
```

Kahnのトポロジカルソートの実行例

```
V = 5
E = 6
edges = [[0, 1], [0, 2], [1, 3], [2, 3], [2, 4], [3, 4]]

print(KahnTopo(V, E, edges))

---------
[0, 1, 2, 3, 4]
```

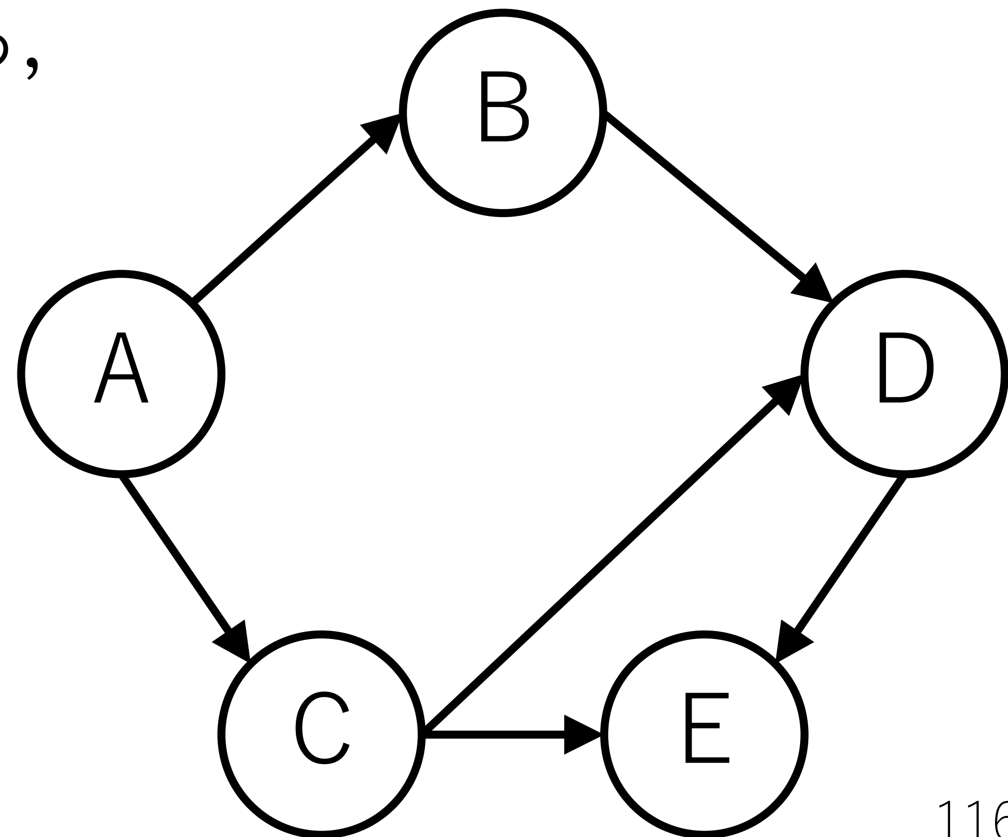


Tarjanのトポロジカルソートの実装方針

ノードを1つ選び，DFSでたどっていく．

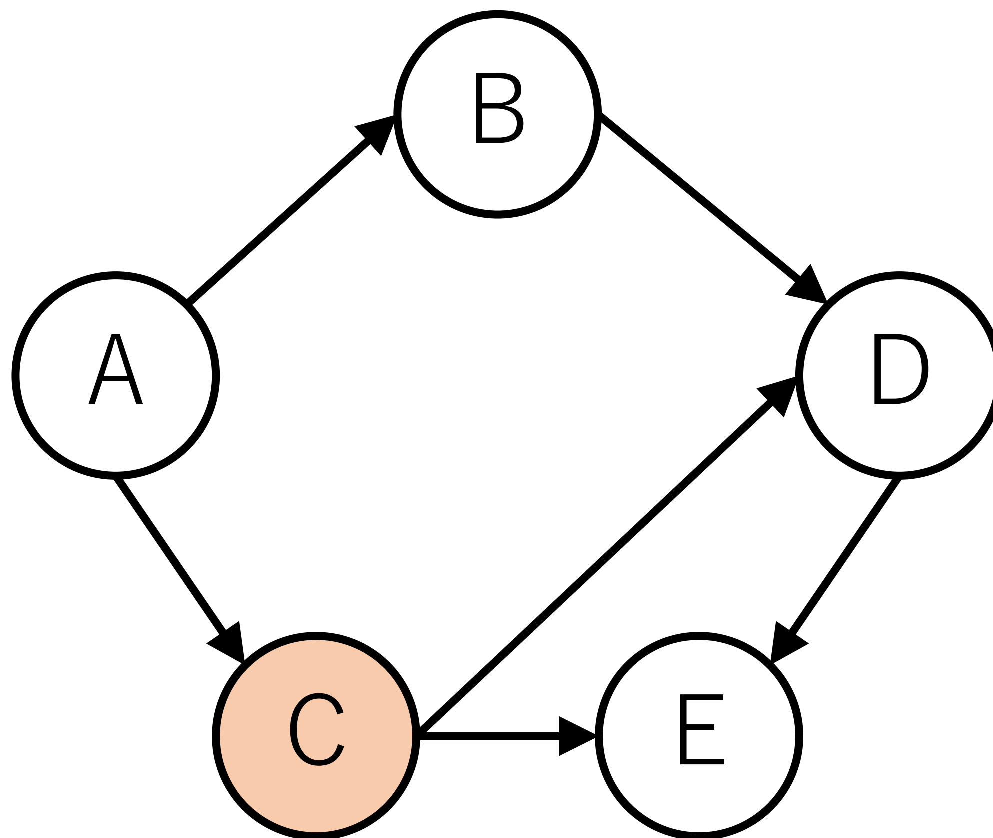
先に進めないところまで到達したら，
後戻りしながらソート済の場所に
先頭から順に入れていく．

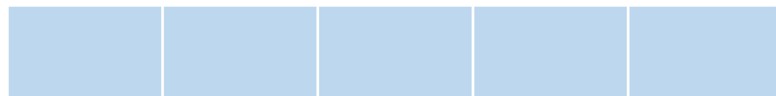
これを全てのノードがチェック
されるまで繰り返す．



Tarjanのトポロジカルソートの実行例

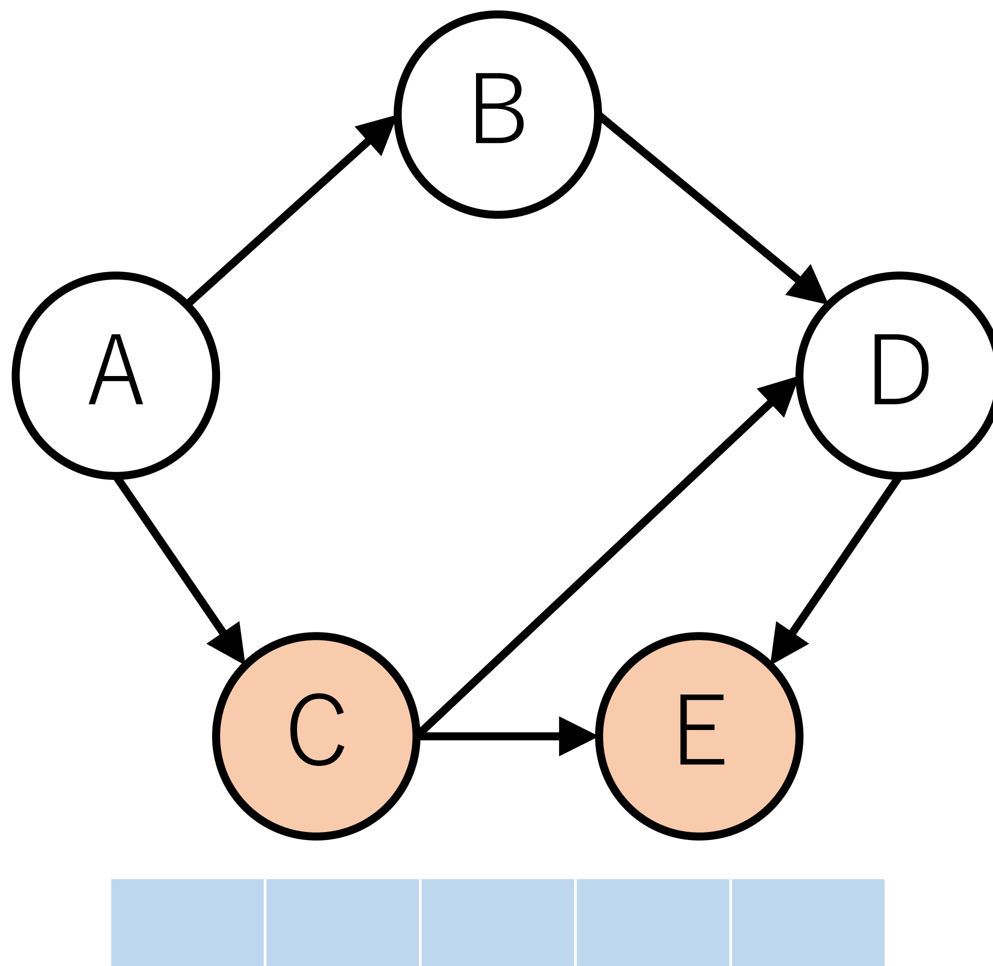
ノードCからスタートする（どこからスタートしても良い）．





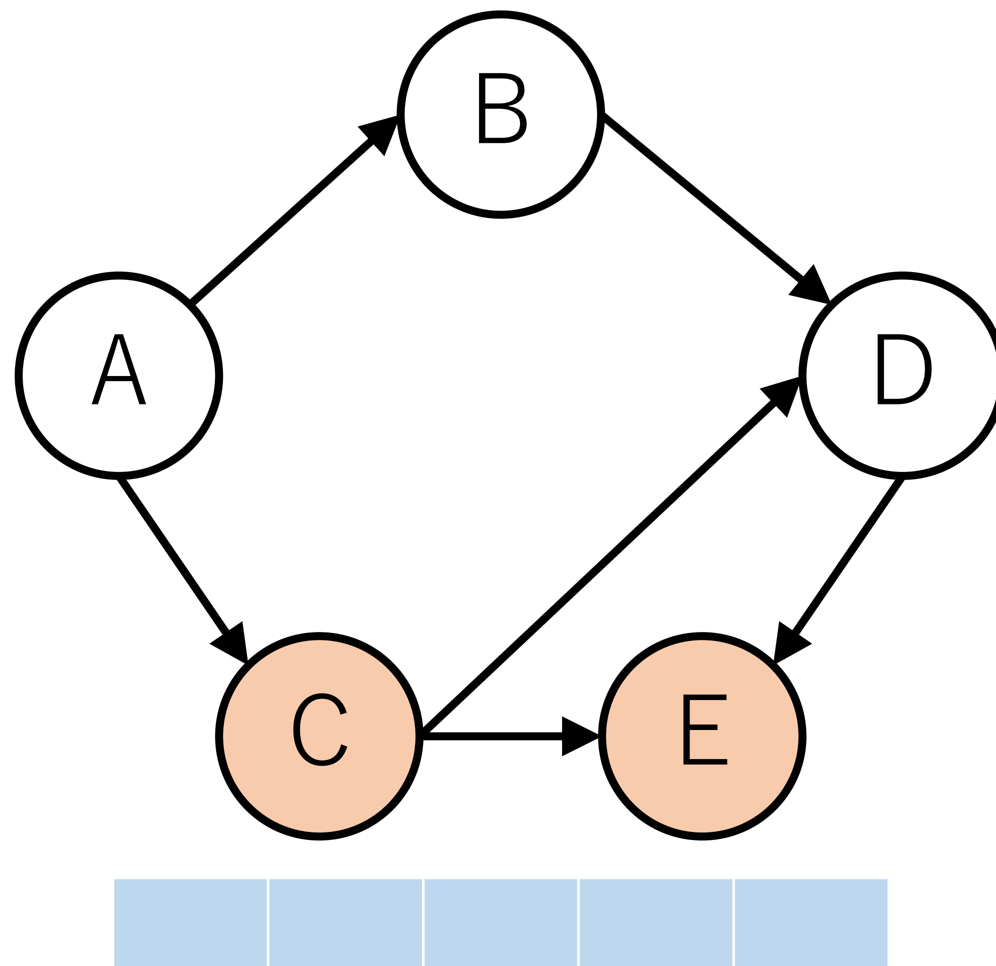
Tarjanのトポロジカルソートの実行例

DFSで行けるところまで行く．今回はC->Eと行ったとする．



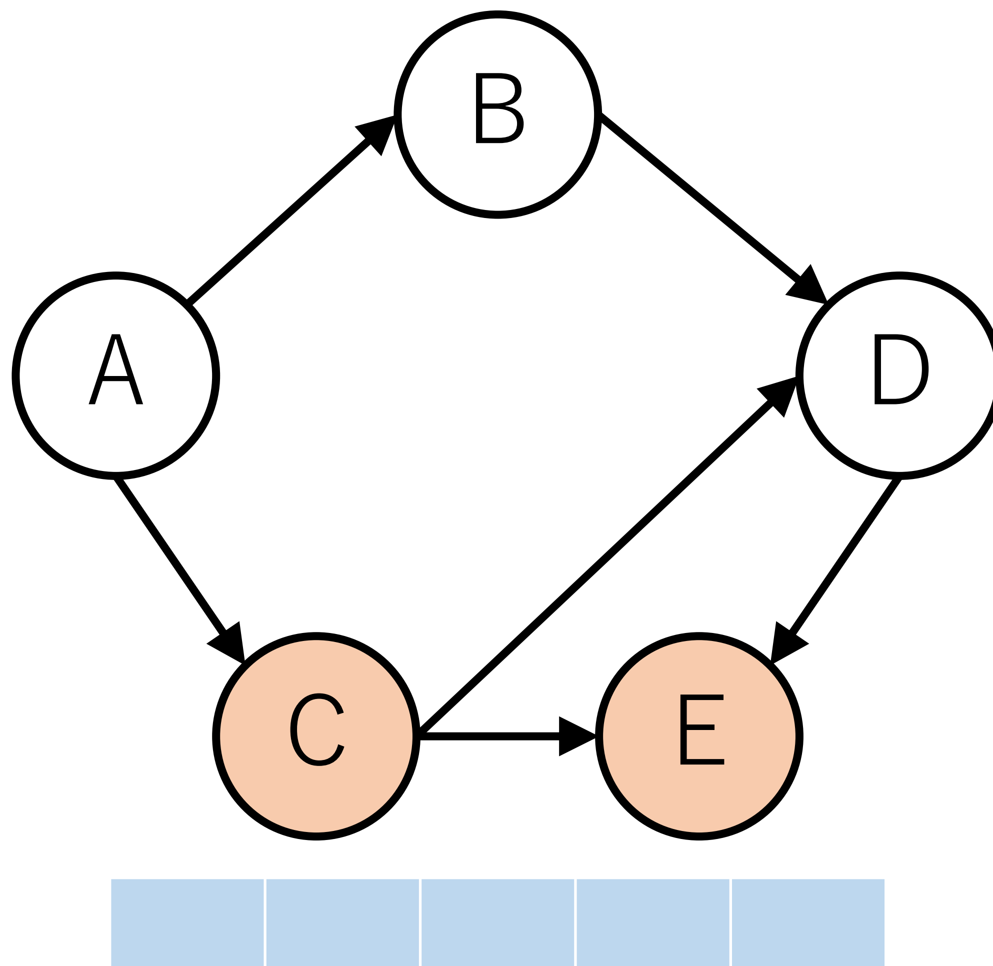
Tarjanのトポロジカルソートの実行例

進めなくなったら，逆戻りし，ソート済に入れる．



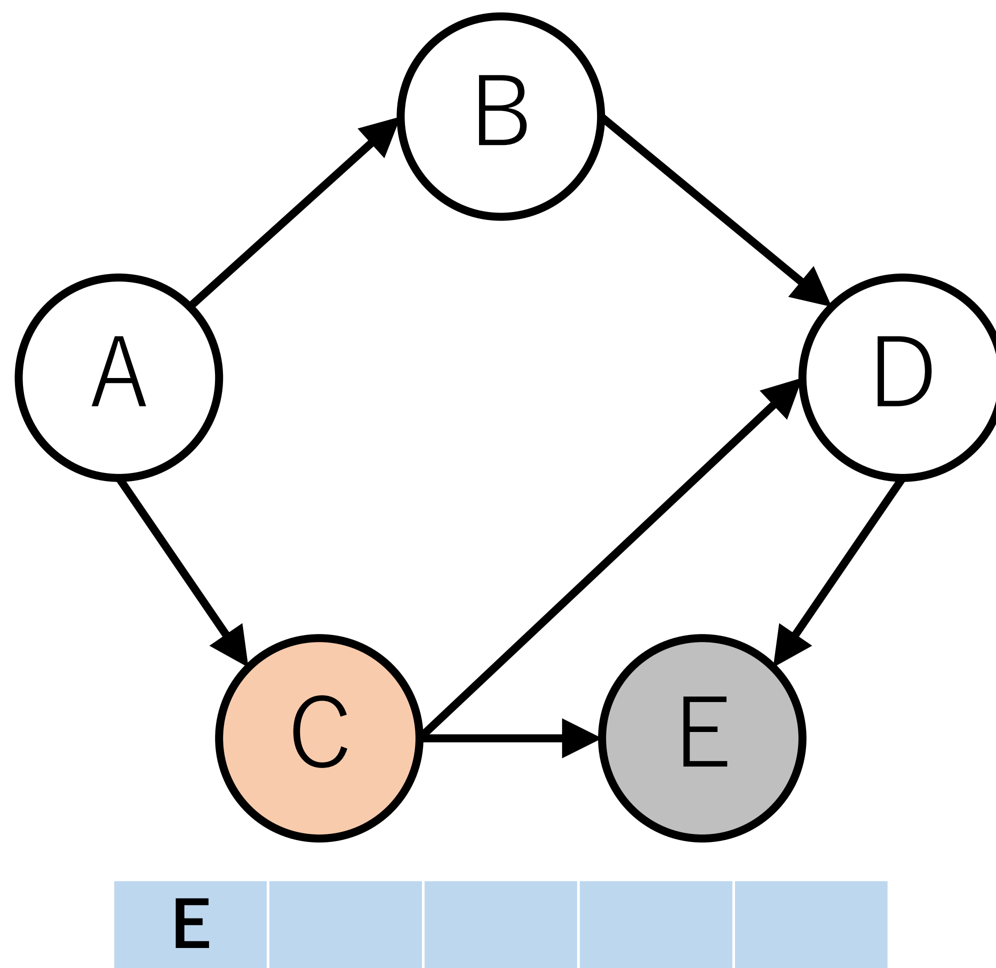
Tarjanのトポロジカルソートの実行例

ただし，**先頭に**追加していく．



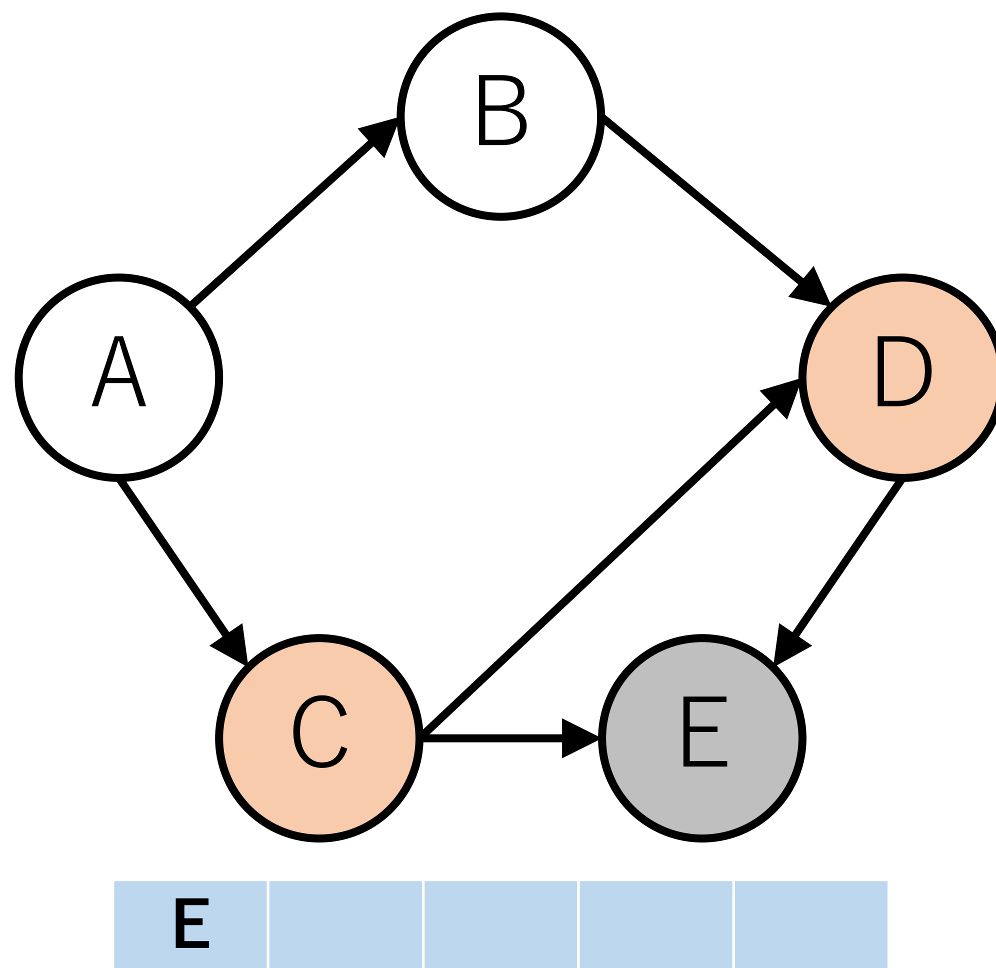
Tarjanのトポロジカルソートの実行例

また，一度でも訪問したノードは訪問済にする．



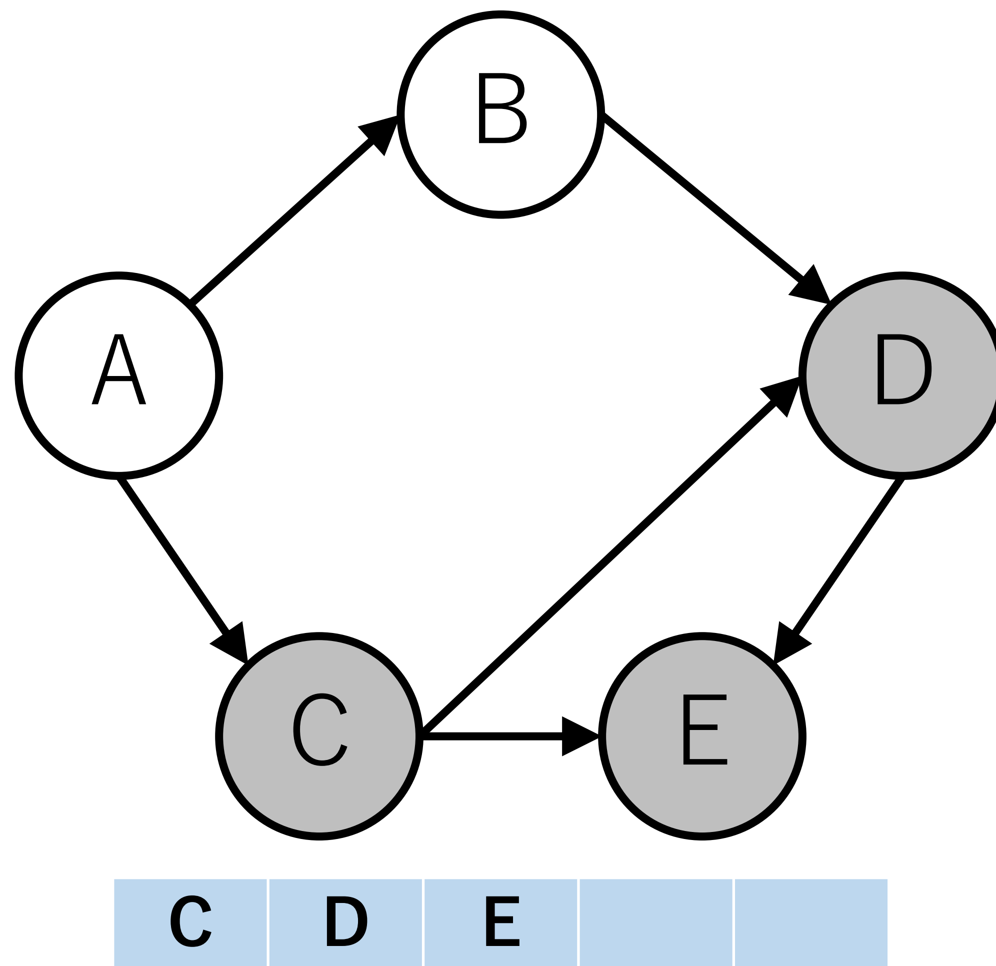
Tarjanのトポロジカルソートの実行例

今回はもう1つDFSで辿ることができるルートがある．



Tarjanのトポロジカルソートの実行例

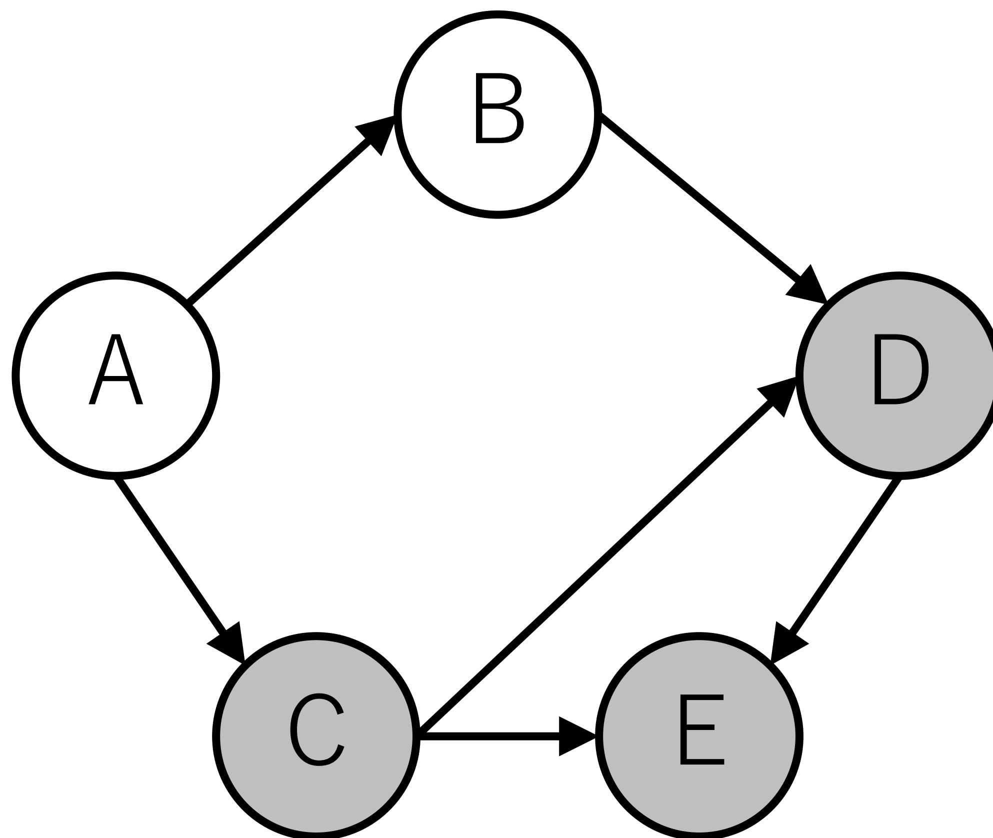
後戻りしながら，**先頭に**追加していく．



C	D	E		

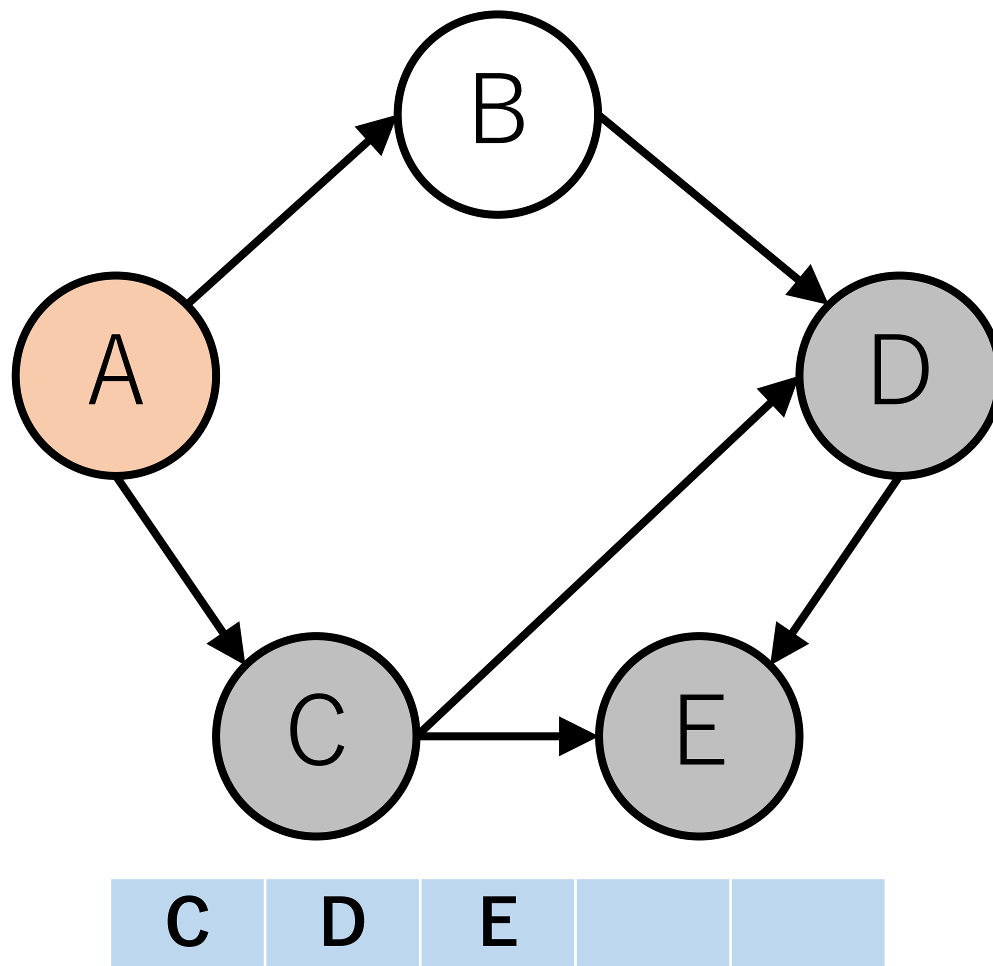
Tarjanのトポロジカルソートの実行例

これ以上戻れないので，未訪問のノードの1つへ移動する．



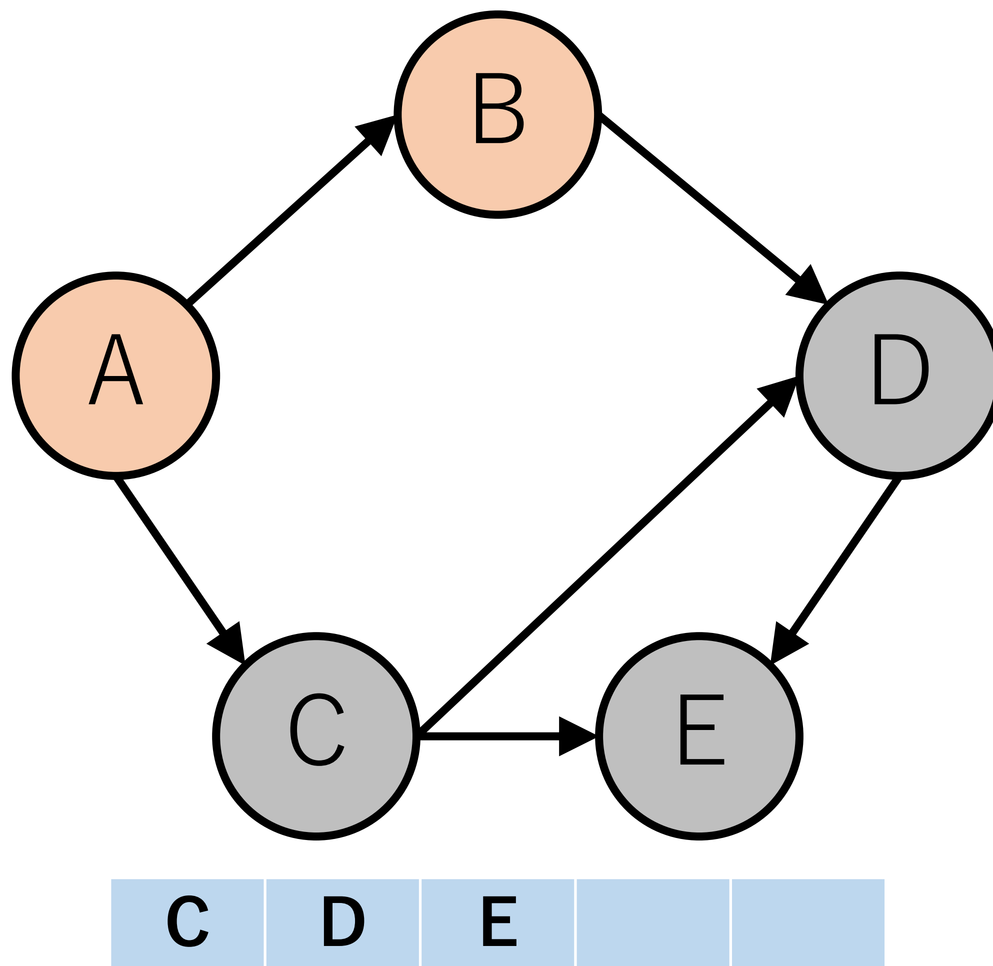
Tarjanのトポロジカルソートの実行例

今回の場合は，ノードAに移動する．



Tarjanのトポロジカルソートの実行例

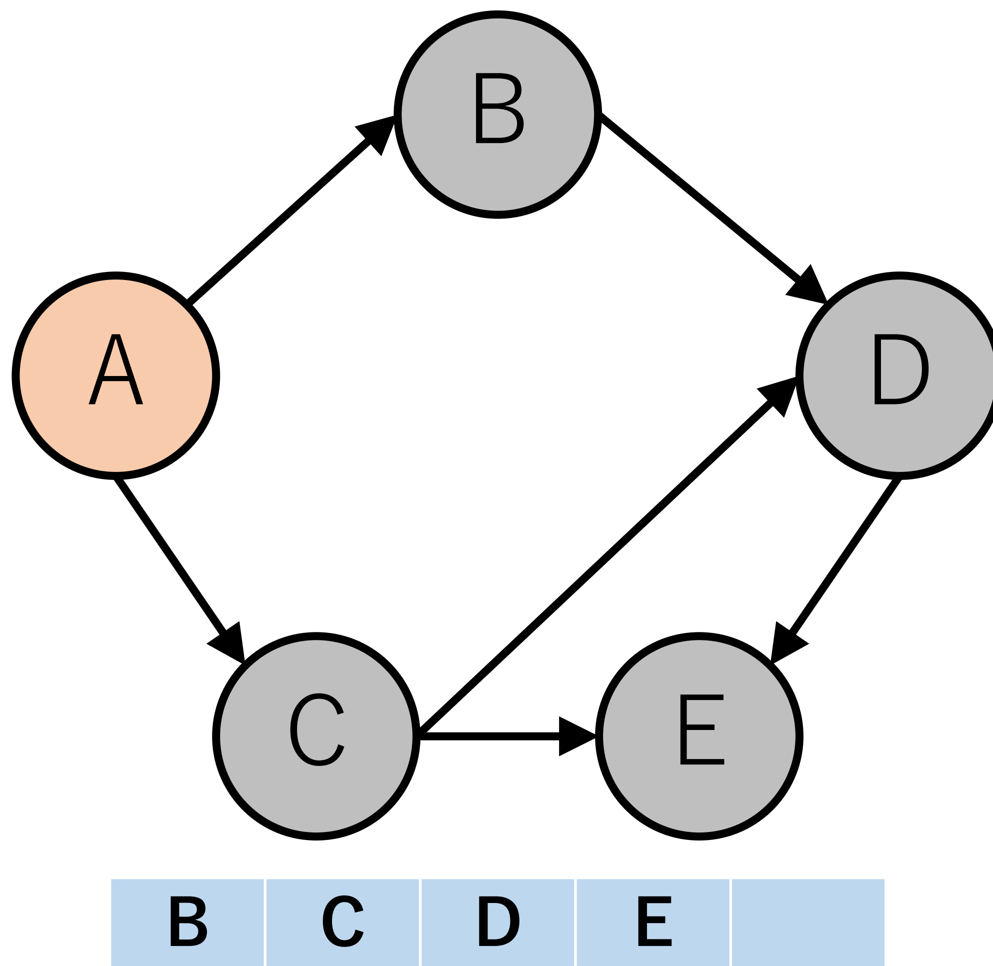
同様にDFSし，ソート済に追加していく．



C	D	E		

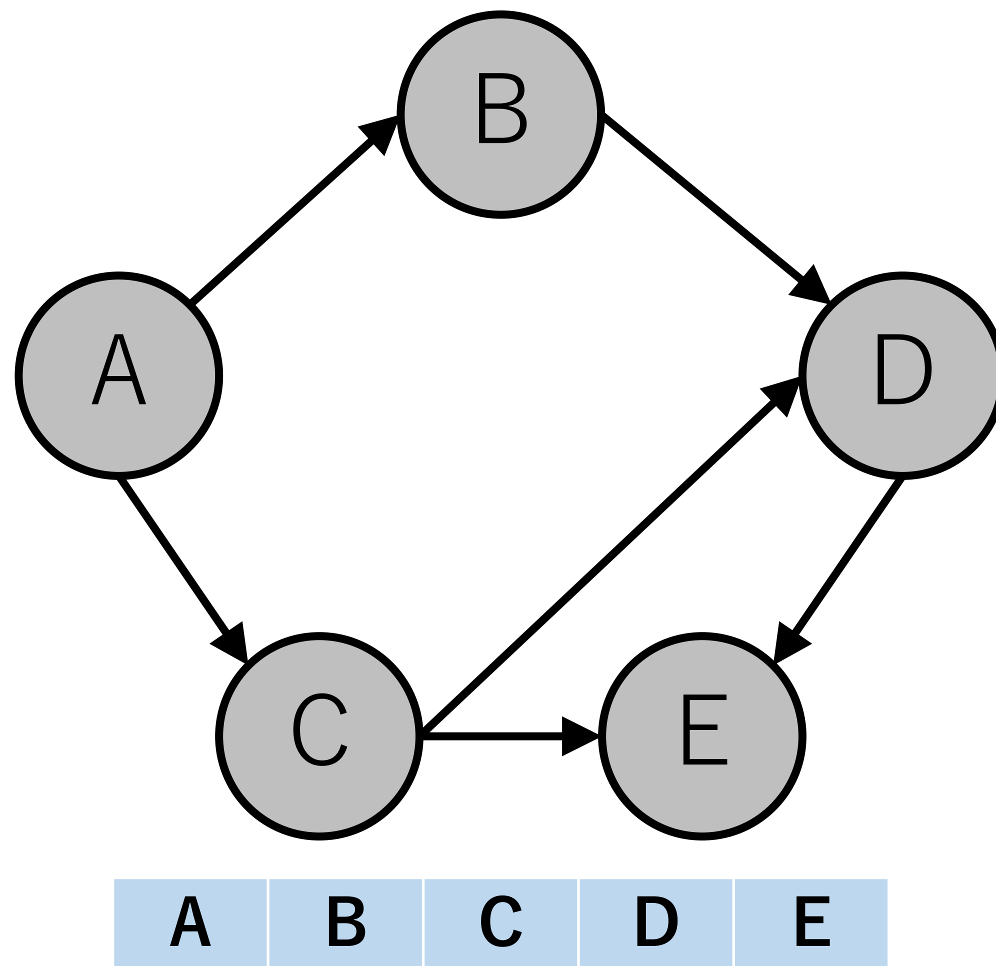
Tarjanのトポロジカルソートの実行例

同様にDFSし，ソート済に追加していく．



Tarjanのトポロジカルソートの実行例

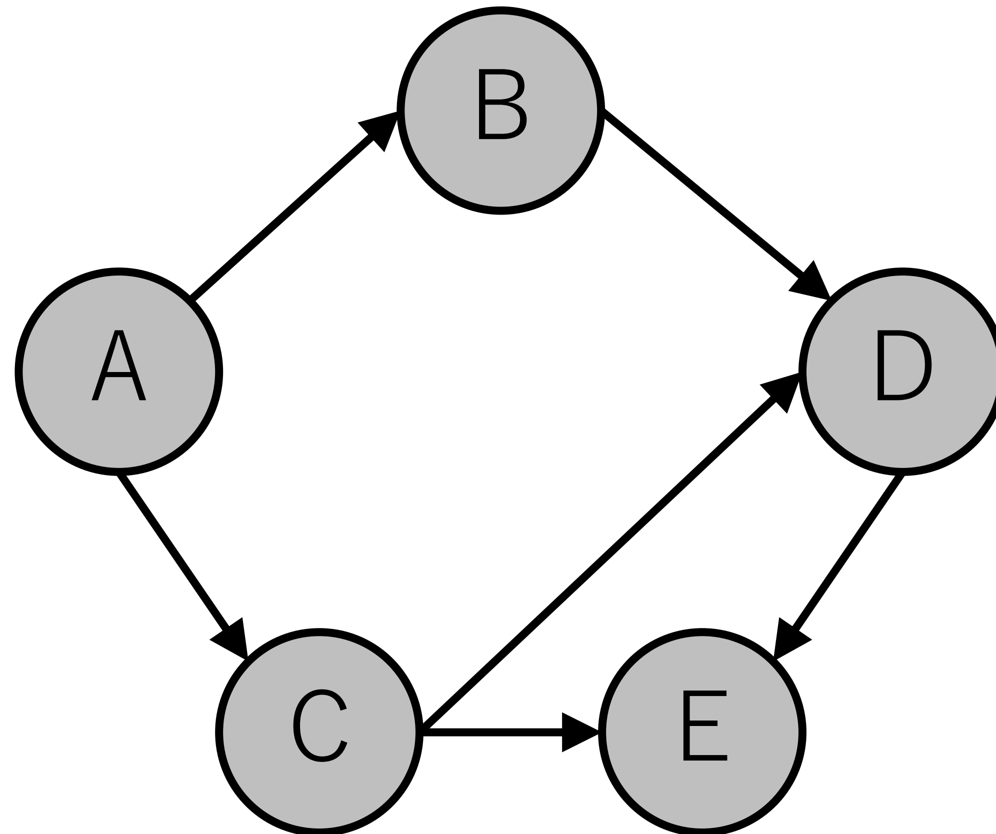
同様にDFSし，ソート済に追加していく．



A	B	C	D	E

Tarjanのトポロジカルソートの実行例

これを繰り返し，全てノードが訪問済になるまでやる．



Tarjanのトポロジカルソート

DFSを行っている途中で「処理中」のノードを再度
訪れることがあった.

現在進行中の探索においてすでに訪れているノードを,
再度訪れていることを意味しており, これは閉路が存在
すること示唆している.

DAGになっていないので, エラーを返す.

Tarjanのトポロジカルソートの実装例

```
def TarjanTopo(V, edges):

    def check(v):
        [再帰で呼び出す関数．後で実装．]

    # ノードをすでに見たかどうかを格納する配列
    # 0：未訪問，1：処理待ち，2：処理済
    visited = [0]*V
    outedge = [[] for _ in range (V)]
```

Tarjanのトポロジカルソートの実装例

```
def TarjanTopo(V, edges):
    …
    for e in edges: outedge[e[0]].append(e[1])
    sorted_g = deque()

    # 全てのノードをチェックする
    for i in range(V): check(i)

    return sorted_g
```

Tarjanのトポロジカルソートの実装例

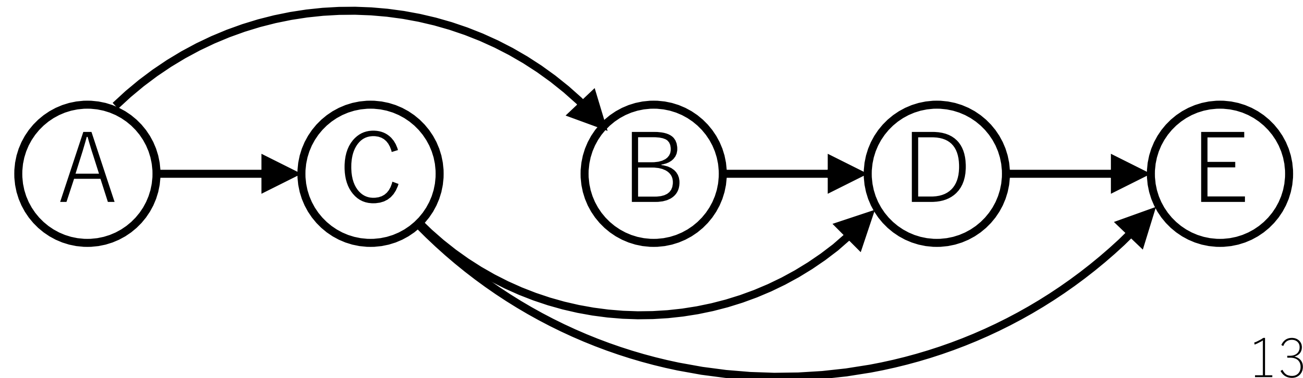
```
def check(v):
	if visited[v] == 1:
		[DAGになっていないので，エラーを返す]
```

Tarjanのトポロジカルソートの実装例

```
def check(v):
    …
    elif visited[v] == 0:
        visited[v] = 1    # 処理待ちにする
        for to_v in outedge[v]: check(to_v)     # 再帰

        visited[v] = 2    # 処理済にする
        # 再帰が終わったら，ソート済の先頭に追加
        sorted_g.appendleft(v)
```

Tarjanのトポロジカルソートの実装例

```
V = 5
edges = [[0, 1], [0, 2], [1, 3], [2, 3], [2, 4], [3, 4]]

print(TarjanTopo(V, edges))


---------
[0, 2, 1, 3, 4]
```



トポロジカルソートの計算量

ソートの本質的な部分はKahnのアルゴリズムの場合while ループ，Tarjanのアルゴリズムの場合再帰部分になる．

入力されたグラフがDAGである場合，全てのノードと辺は高々1回しかチェックされない．

よって，$O(|V|+|E|)$.

トポロジカルソートの応用例

閉路チェック

有向グラフにおいてトポロジカルソートできなかった場合，どこかに閉路が存在する（有向巡回グラフになっている）．

依存関係を調べて整理することに使われる．

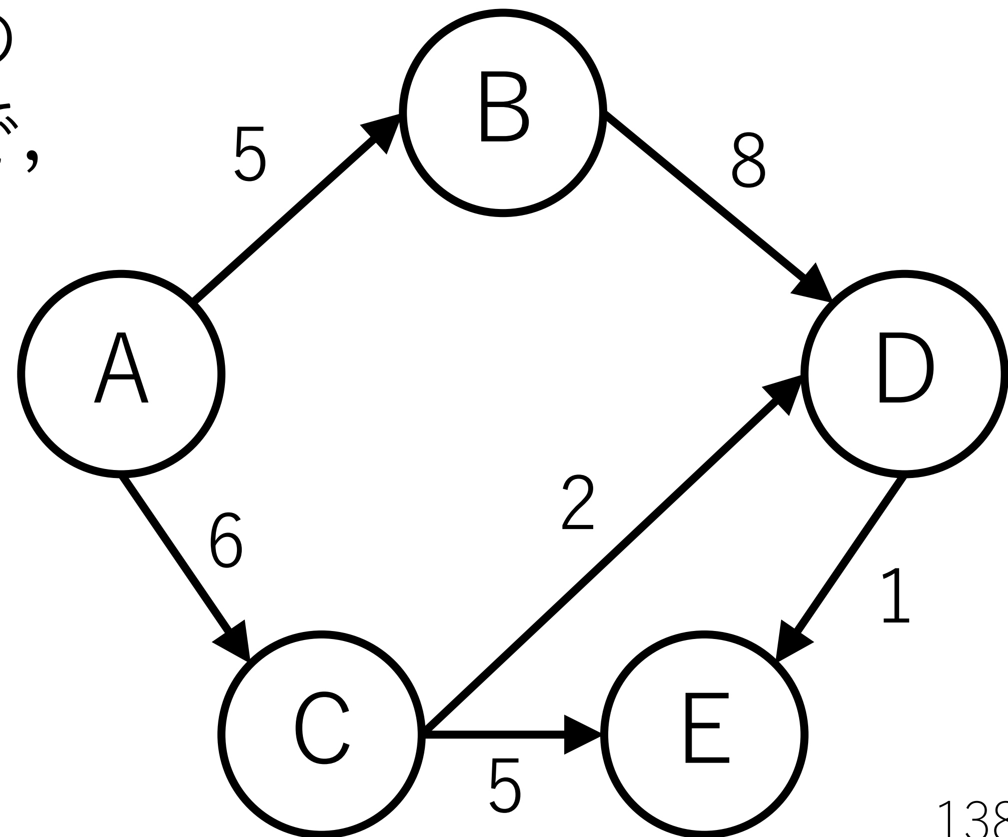
プロジェクト内の実行タスク順序

ビルドにおけるライブラリの依存関係

トポロジカルソートの応用例

DAGにおいての最短経路を求める.

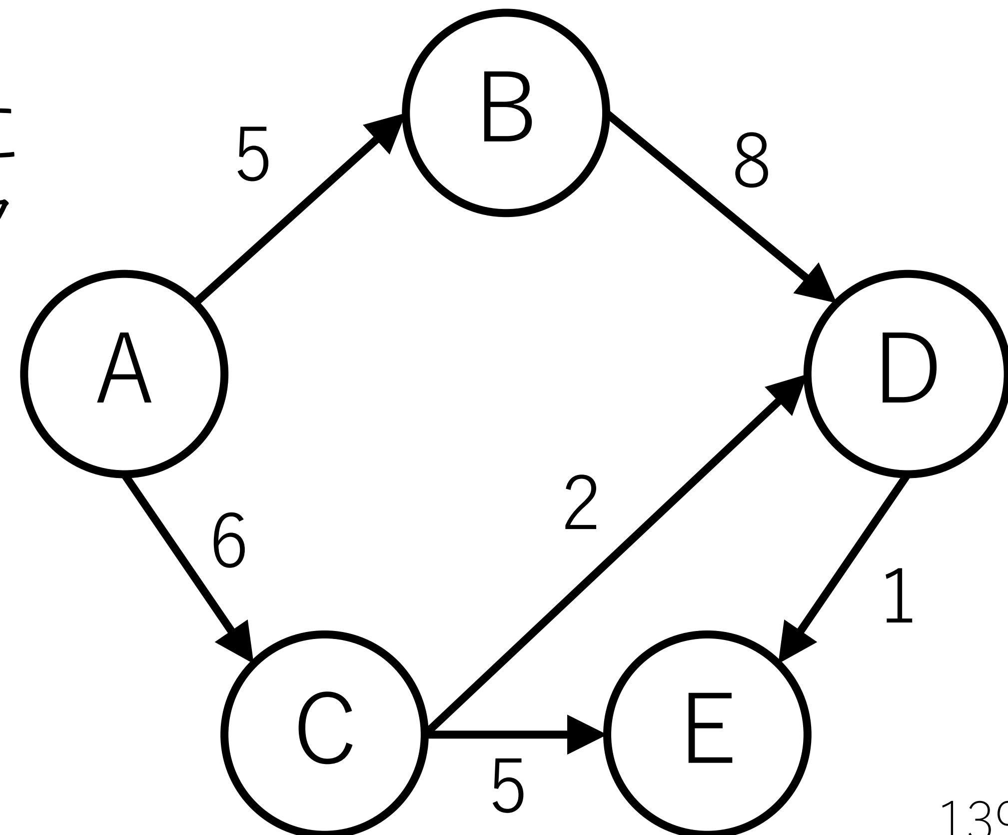
トポロジカルソートを実行し，その
ソート順にノードを見ていくことで,
最短経路を求めることができる.



トポロジカルソートの応用例

あるノードuから出る辺で繋がっているノードvは,
トポロジカルソート結果においてuよりも後に出てくる.

よってトポロジカルソート結果順にノードを調べていれば，今チェックしているノードに入ってくる経路は全て調べ終えた状態になっているはずで，そのノードに至る最短経路が既に計算されていることになる.



DAGでの最短経路を求めるコード例

```
def shortest_path_DAG(V, E, edges, start_vertex):

    # まずトポロジカルソートを実行.
    # コードは前述のスライド参照.
    sorted_g = KahnTopo(V, E, edges)

    if not sorted_g:
        [DAGでないのでエラーを返す]
```

DAGでの最短経路を求めるコード例

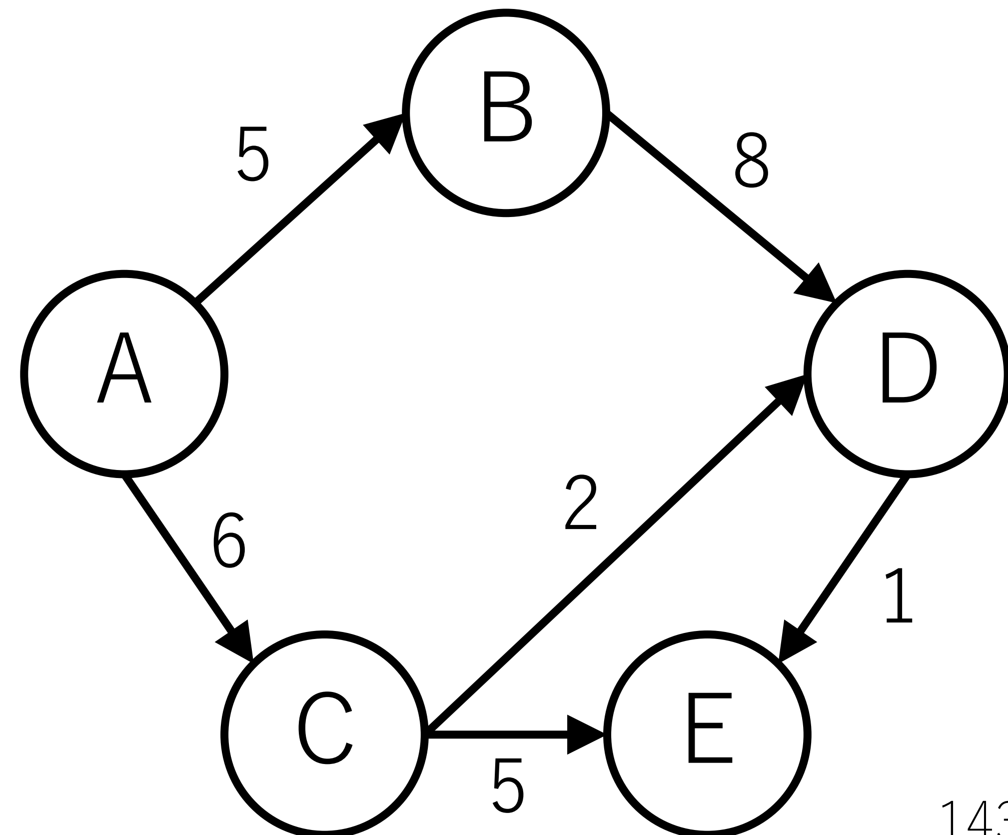
```
def shortest_path_DAG(V, E, edges, start_vertex):
    …
    # 隣接リストを作成
    edge_list = [[] for _ in range(V)]
    for u, v, weight in edges:
        edge_list[u].append([v, weight])

    # 最短距離を保持するリストの初期化
    dist = [float('inf')] * V; dist[start_vertex] = 0
```

DAGでの最短経路を求めるコード例

```
def shortest_path_DAG(V, E, edges, start_vertex):
    …
    # トポロジカルソートでソートされた順にノードを
    # 見ていき，distを更新する．
    for u in sorted_g:
        if dist[u] != float('inf’):
            for v, weight in edge_list[u]:
                if dist[v] > dist[u] + weight:
                    dist[v] = dist[u] + weight
    return dist
```

DAGでの最短経路を求めるコードの実行例

```
V = 5
E = 6
edges = [[0, 1, 5], [0, 2, 6], [1, 3, 8],
[2, 3, 2], [2, 4, 5], [3, 4, 1]]
print(shortest_path(V, E, edges, 0))

--- 実行結果 ---
[0, 5, 6, 8, 9]
```



トポロジカルソートでDAGの最短経路を求める場合の計算量

トポロジカルソート部分は$O(|V| + |E|)$.

最短経路をDPによって求める部分では，各ノードはソートされた順に1回チェックされ，各辺も1回だけ処理される.

よってここも$O(|V| + |E|)$.

2つ合わせても$O(|V| + |E|)$.

ダイクストラ等よりも速いことが期待できるが，DAGでないと使えないことに注意.

今日のまとめ

最小全域木

- クラスカル法，Union Find木
- プリム法

トポロジカルソート

- Kahnのアルゴリズム
- Tarjanのアルゴリズム

コードチャレンジ：基本課題#11-a [1.5点]

スライドで説明したUnion-Find木を参考にして,
クラスカル法を実装してください.

Union-Find木を使っていない実装は認められないので,
注意してください.

ご自身で1から作ったUnion-Find木のコードを使っても
構いません.

コードチャレンジ：基本課題#11-b [1.5点]

Kahnのトポロジカルソートを実装してください．結果は辞書順最小になるようにしてください．

Tarjanのトポロジカルソートは認められません．

deque，heapqを使用しても構いません．

コードチャレンジ：Extra課題#11 [3点]

本日勉強したデータ構造，アルゴリズムに関する問題.

期末試験

日時：7/31 12:55**集合，**13:05**開始，**14:40**頃解散予定**

試験時間：75分

会場：工学部2号館4階241，242教室を予定.

後日，座席を指定しますので詳細をお待ち下さい.

持ち込みは認めません.

【重要】期末試験受験者調査

期末試験を受験する予定の人は全員，以下のアンケートで回答をしてください．

https://forms.gle/7XdDFwtKsboLkWS68

回答期限：7/12 24:00

このアンケートに回答していない場合，期末試験の受験を認めないことがあります．