Algorithms (2020 Summer)
#12：グラフアルゴリズム3

矢谷 浩司

期末レポートに関するQ&A

授業のホームページにてまとめてありますので,
ご確認ください.

https://yatani.jp/teaching/doku.php?id=2020algorithms:start#%E3%82%A2%E3%83%AB%E3%82%B4%E3%83%AA%E3%82%BA%E3%83%A0_2020%E5%B9%B4%E5%BA%A6

今日のお題

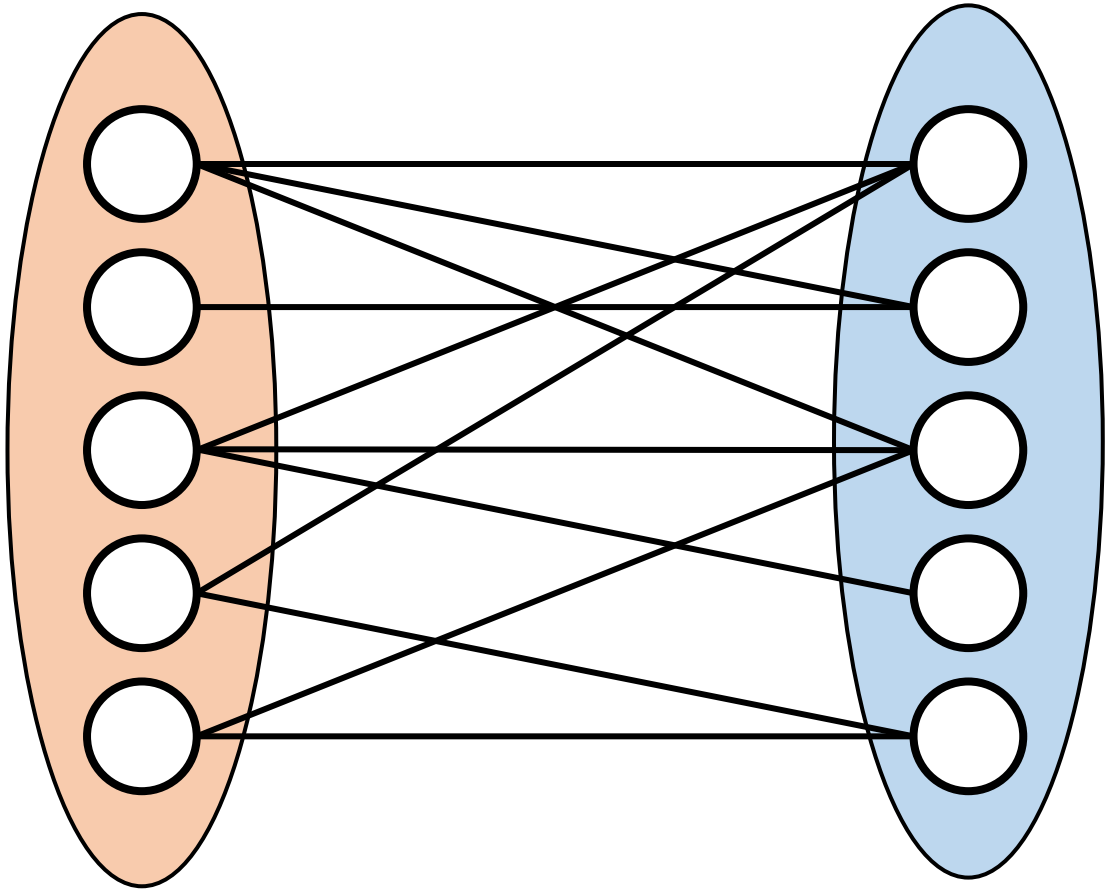
二部グラフのマッチング問題

最小全域木

本講義のまとめ

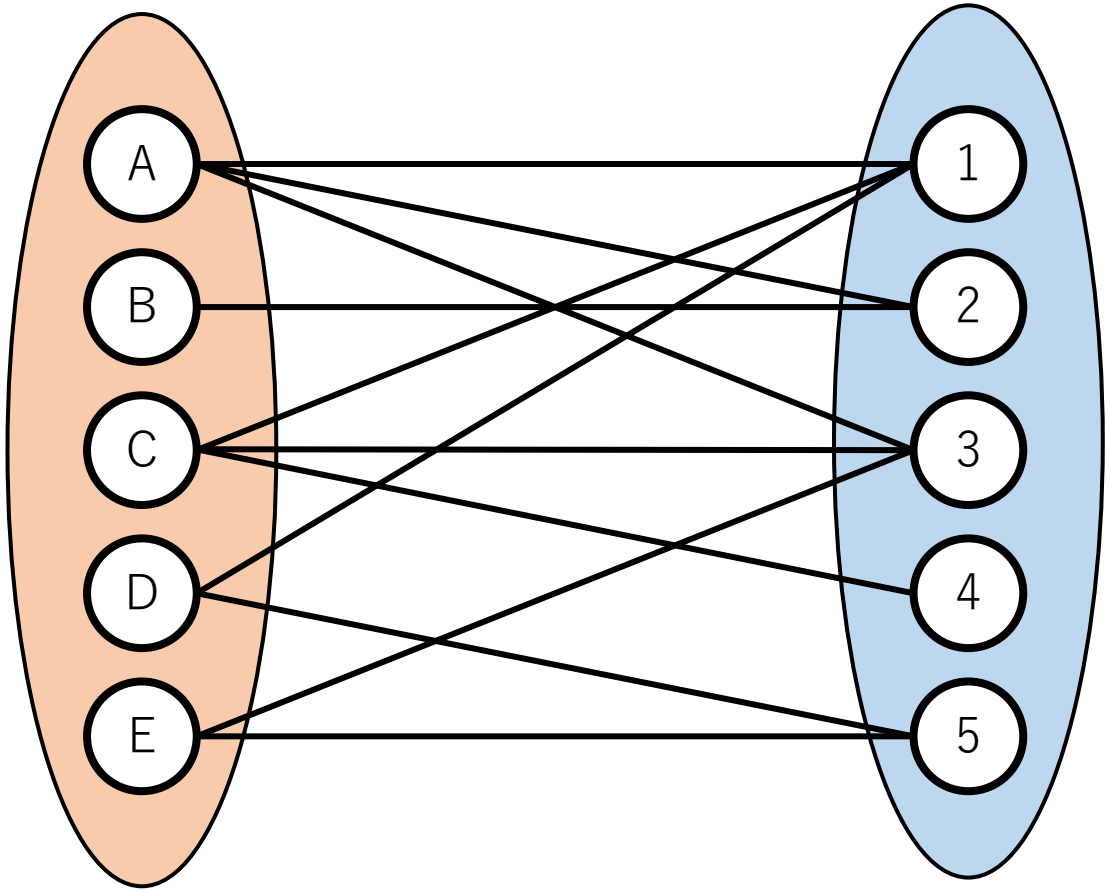
二部グラフ（bipartite graph）

頂点集合を2つの部分集合に分割でき，各集合内の頂点同士の間には辺が無いグラフ.



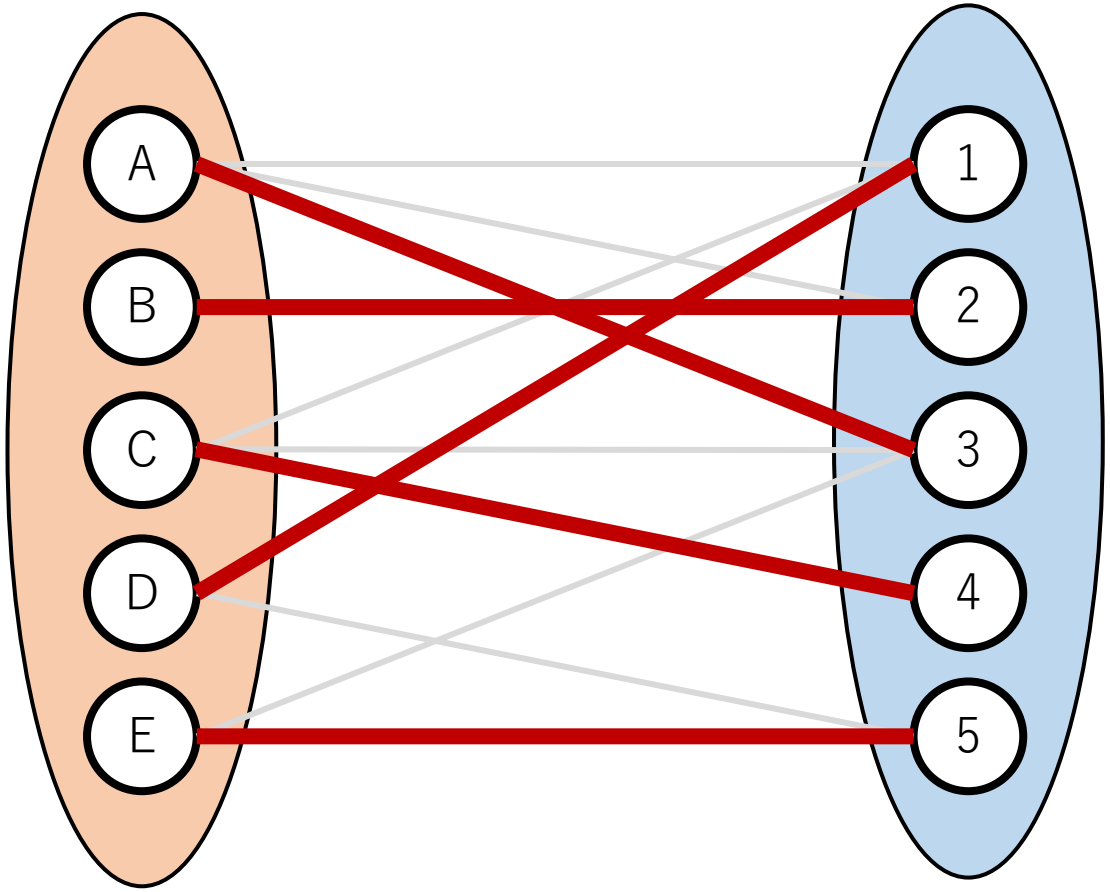
二部グラフの例：ペア組み

左のグループの人と右のグループの人をペアにする.
辺があるのはペアになっても良いことを示す.



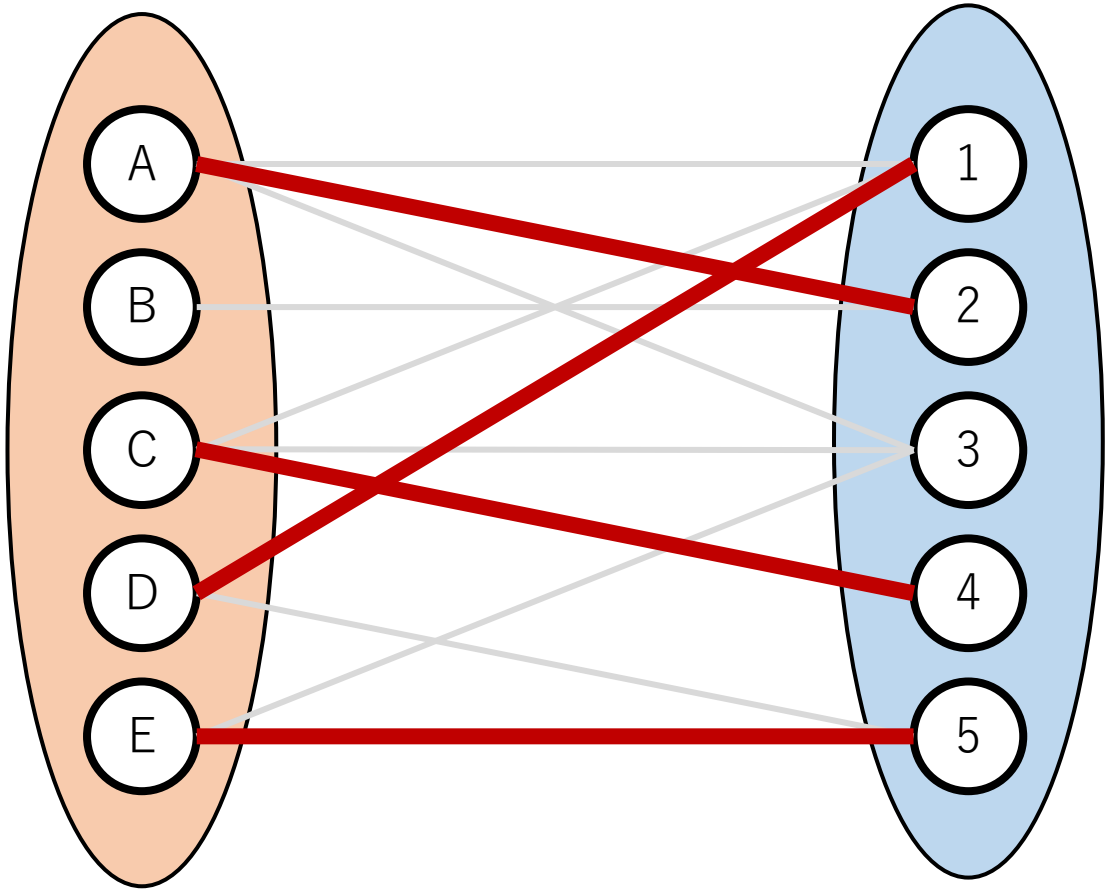
ペア組みがうまくいく例

全員ペアになるケース.



ペア組みが最大でない例

Bと3がペアになれない.



二部グラフのマッチング問題

二部グラフにおいてノード間をつなぐ組み合わせを解く.

答えは必ずしも1つではない（最大マッチングの時など）.

いろんなケースがありうる.

- 複数のノードにつながることを許す（多対多）.
- ある決められたペアの数を達成する.
- などなど. . .

今日のところは

二部グラフのマッチング問題でメジャーな2つを紹介.

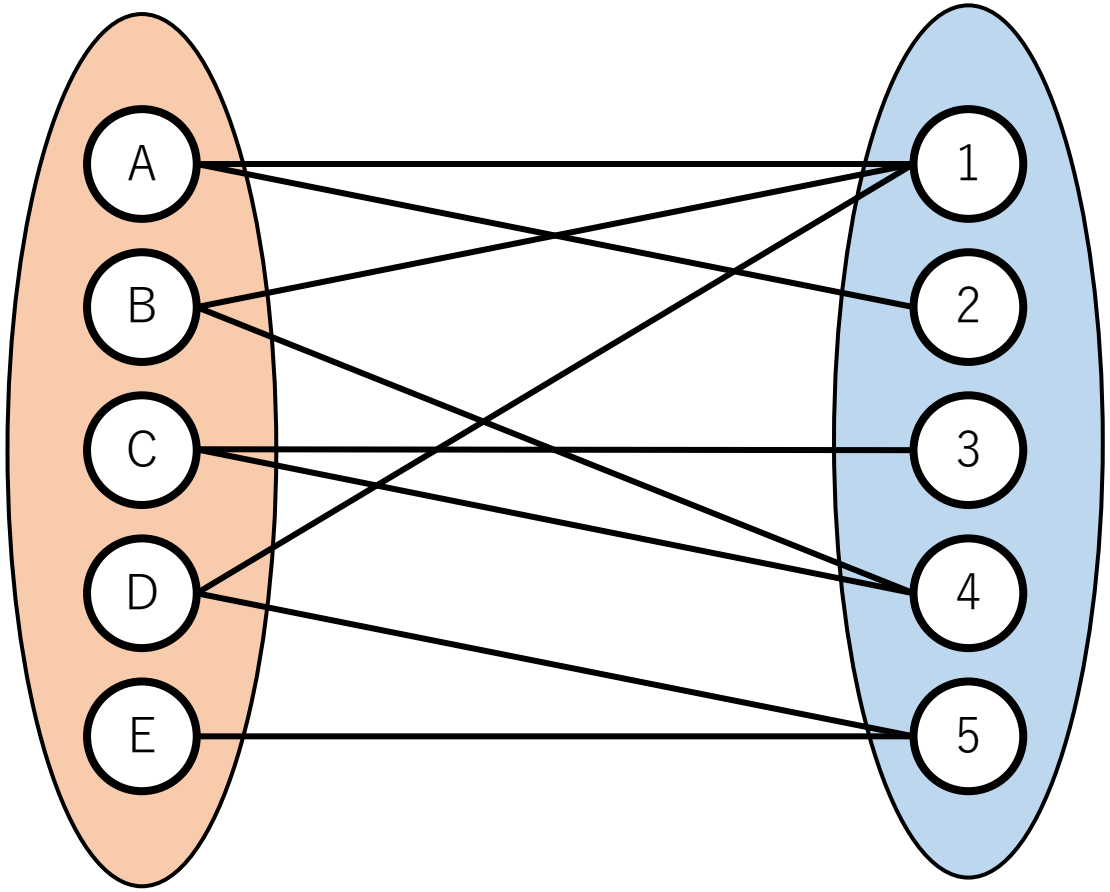
- 二部グラフの最大マッチング問題
- 重みつき二部グラフの最大マッチング問題

最大マッチング

- ペアの数を最大にする辺の組み合わせ（の1つ）.

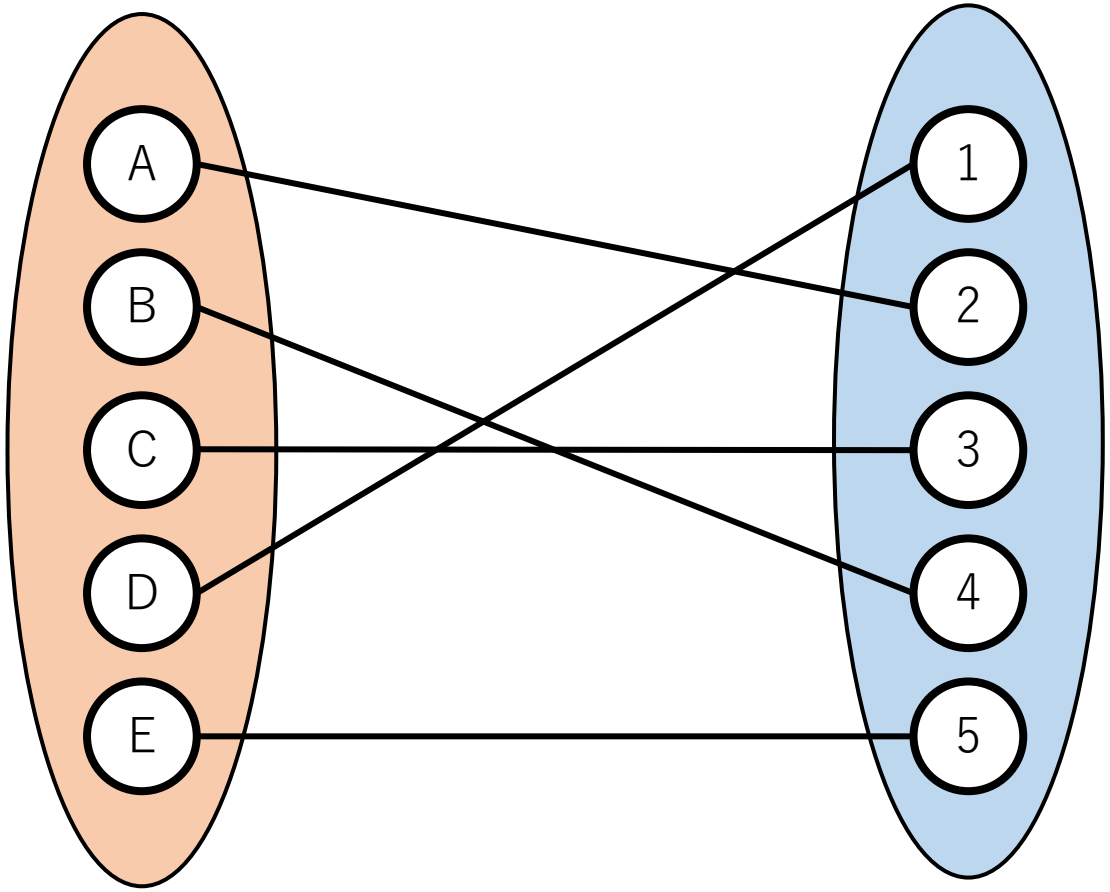
二部グラフの最大マッチング問題

左右のグループをつなぐ辺の数を最大にする．ただし，
どの2辺も共通のノードを持たない．



二部グラフの最大マッチング問題

「どの2辺も共通のノードを持たない」
＝「n股はナシ（n>1）」



二部グラフの最大マッチング問題

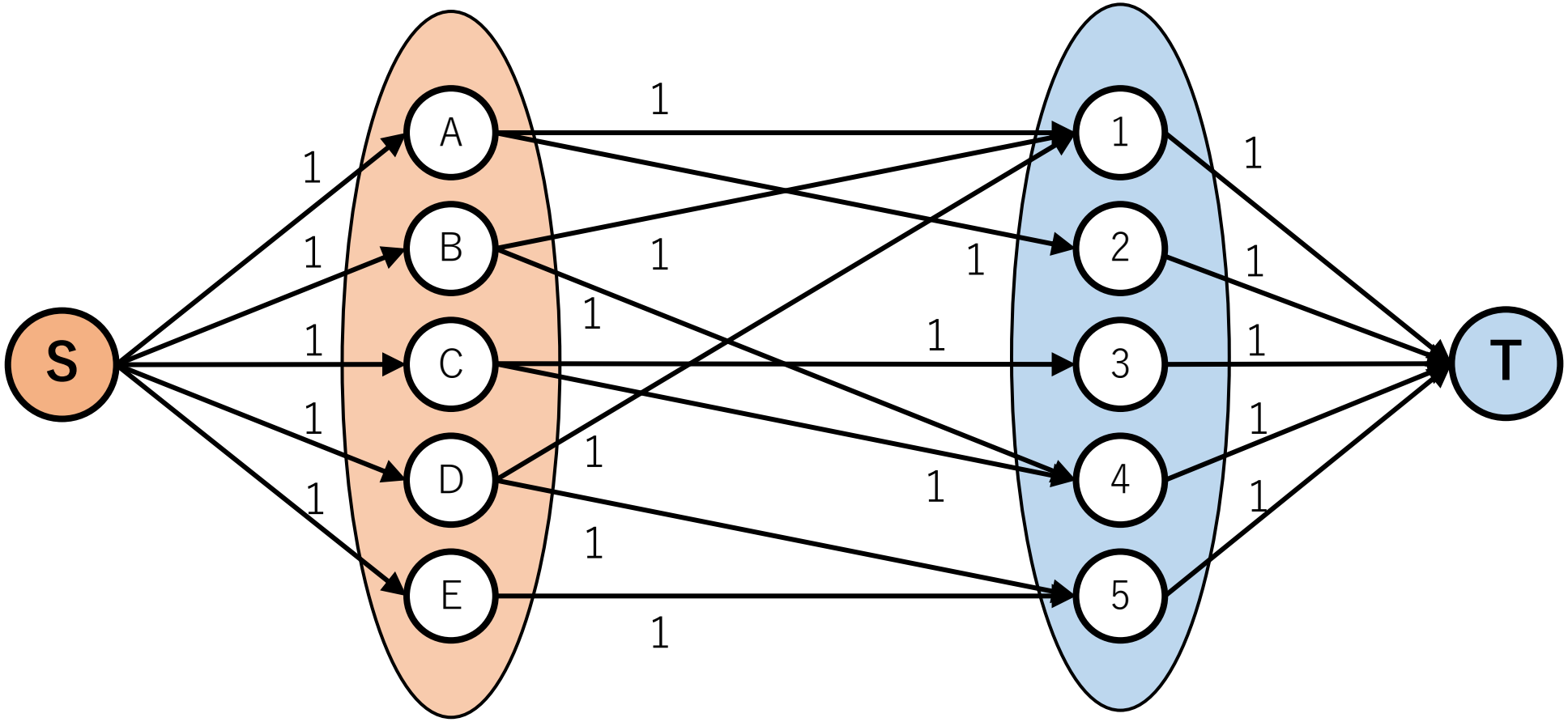
最大マッチングになる辺の組み合わせが見つかった時,
残る辺の数は最大.

これを最大流問題に置き換えて考える.

辺の容量を1として，左のグループから右のグループに流れる流量が最大になる辺の組み合わせを考える.

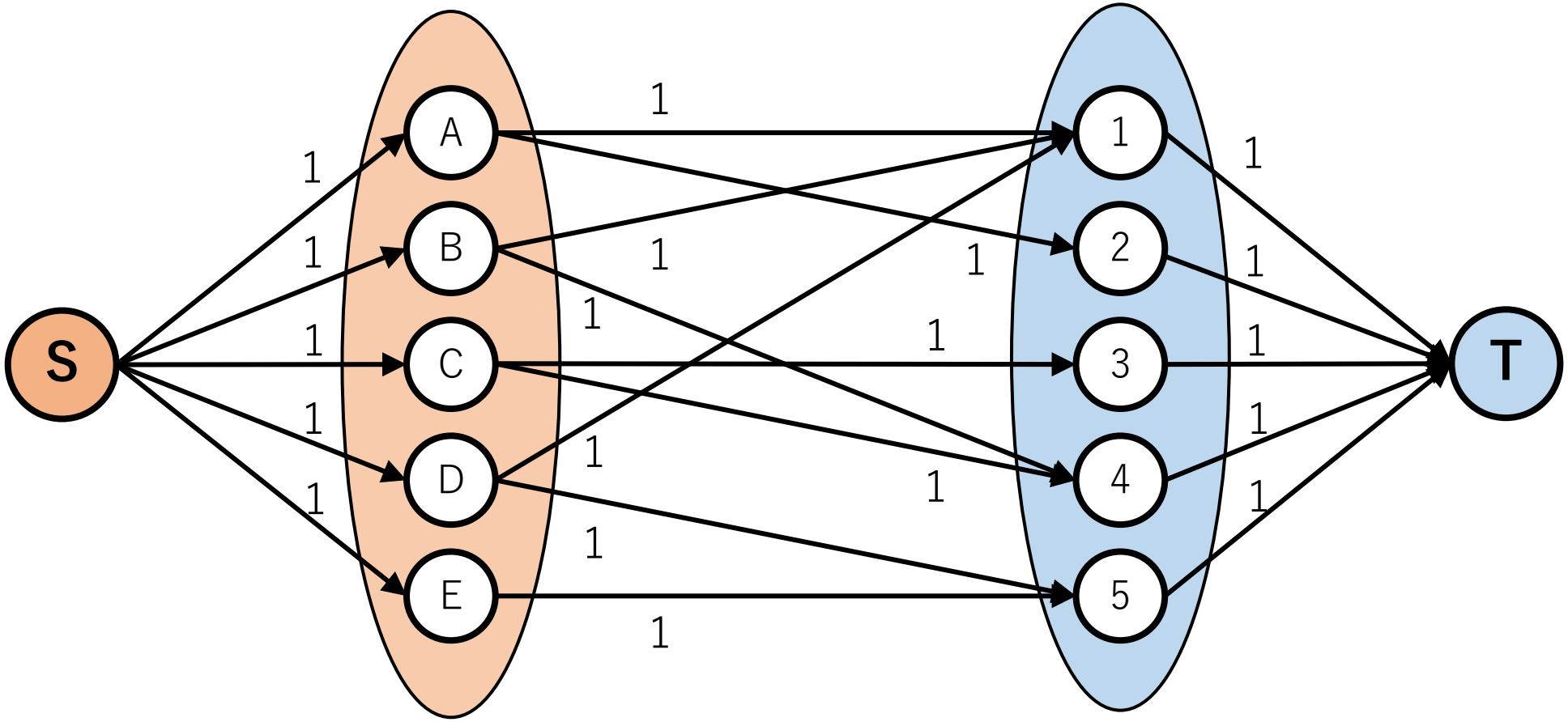
二部グラフの最大マッチング問題

仮想的な開始ノードS，終了ノードTを考えて，SからTへの最大流量を求める．



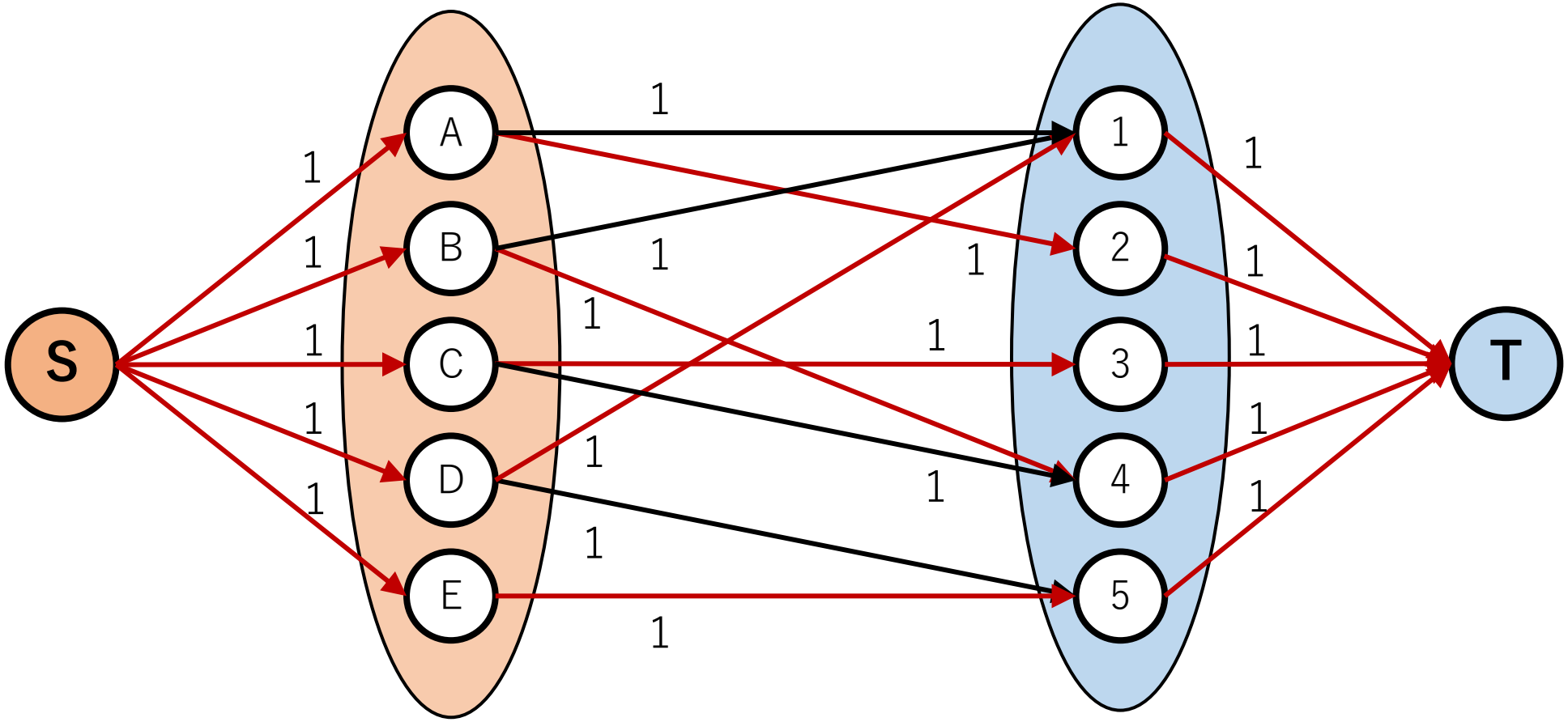
二部グラフの最大マッチング問題

Sから向かう辺，Tに向かう辺も容量を1にすることで，
「ペア」になることを保証する．



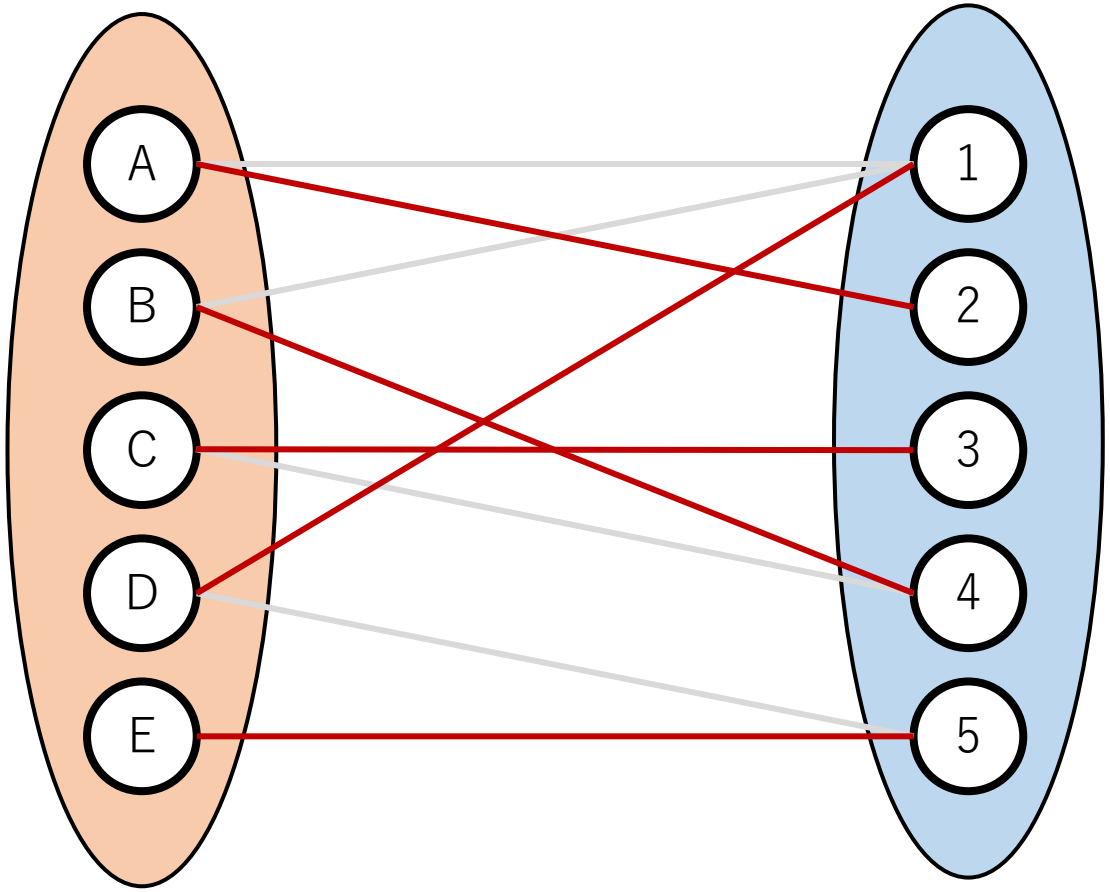
二部グラフの最大マッチング問題

フォード・ファルカーソン法などにより最大流となる経路の組み合わせを求める.



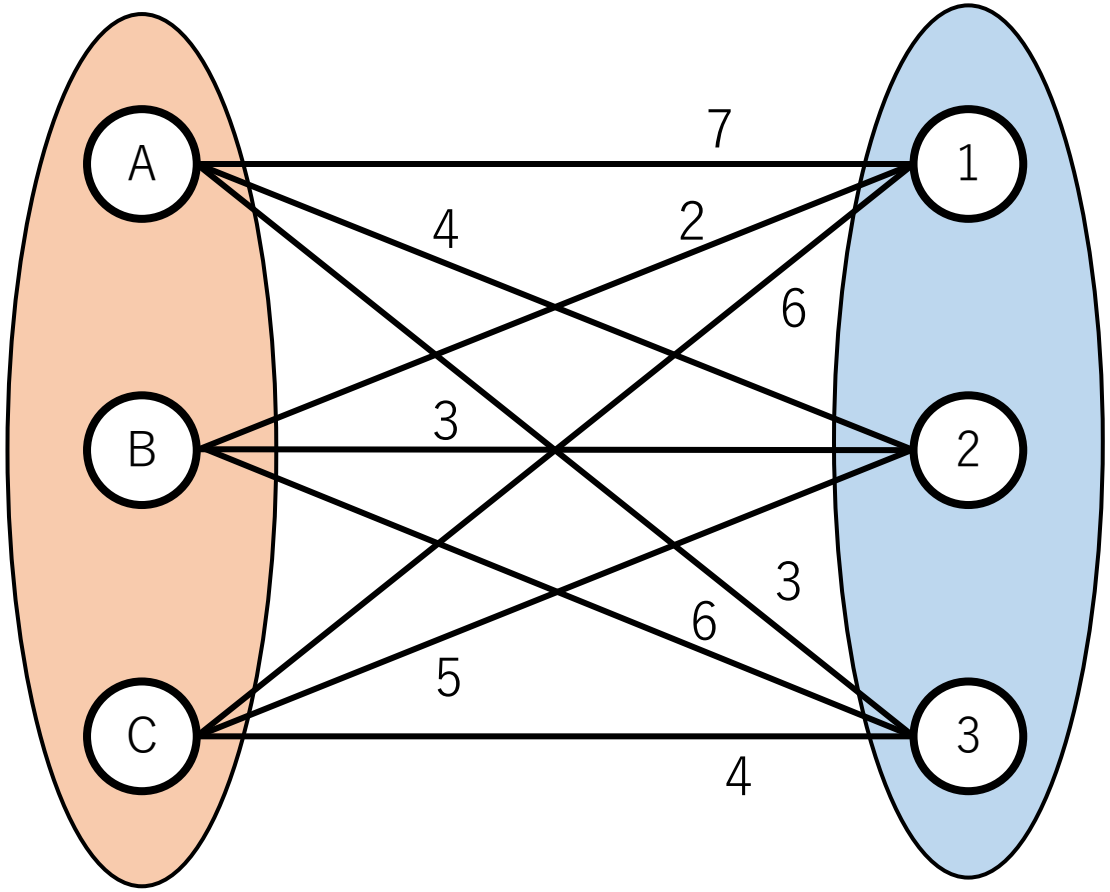
二部グラフの最大マッチング問題

S，Tを取り除くと最大マッチングになっている．



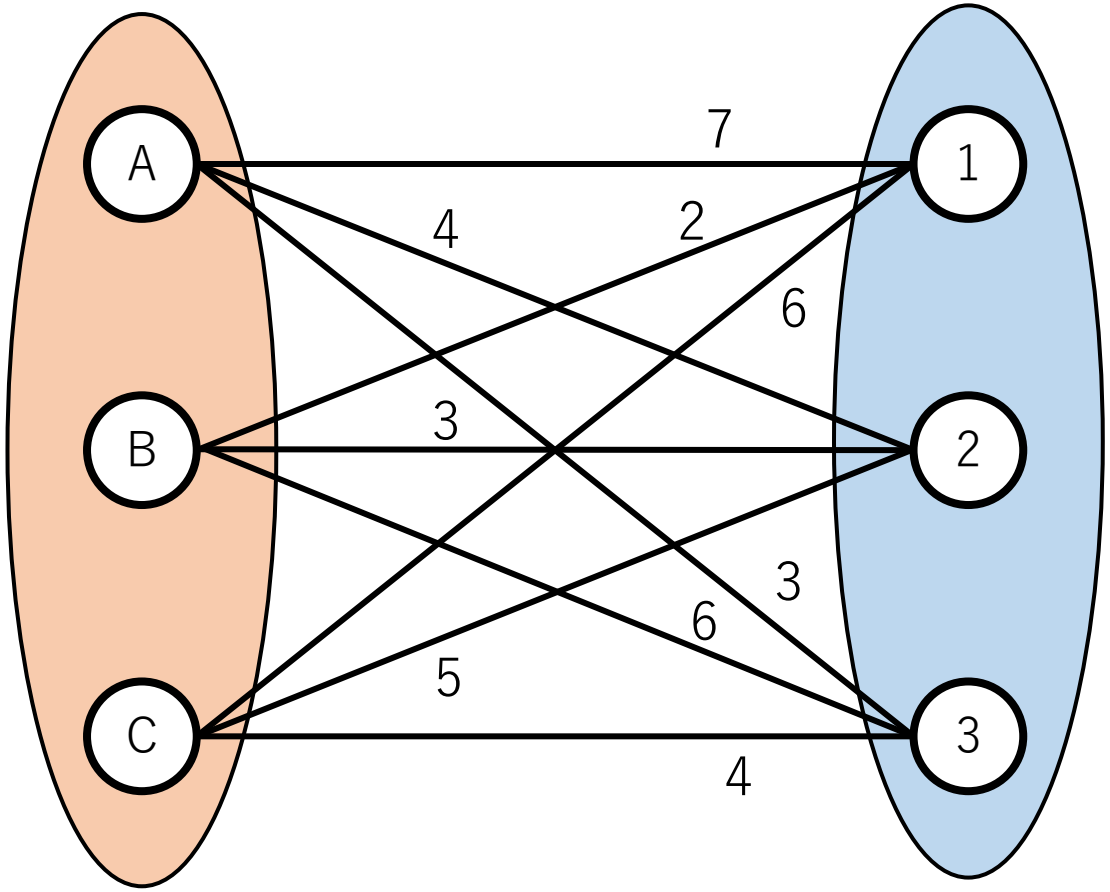
重み付き二部グラフの最大マッチング問題

左右のグループをつなぐ辺の重みの総和を最大にする.
どの2辺も共通のノードを持たない.



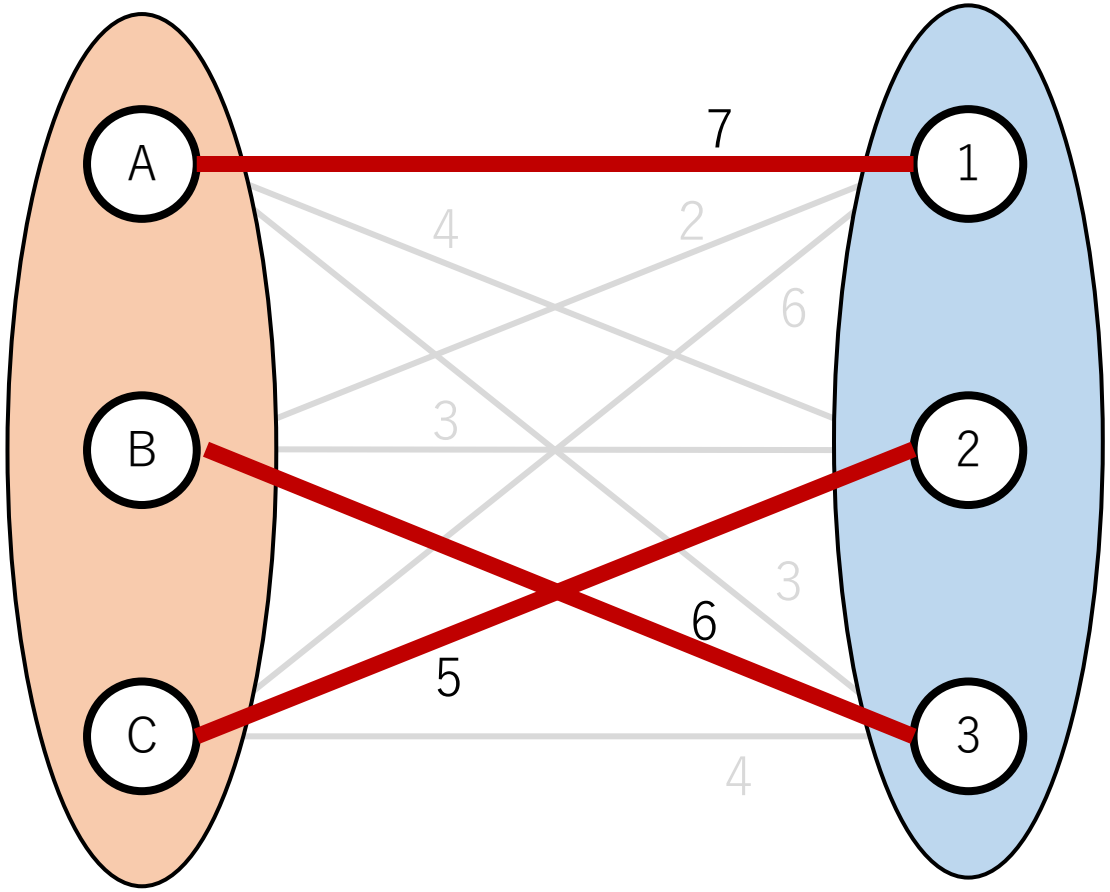
重み付き二部グラフの最大マッチング問題

例）希望度合いがついたペア組み．数字が大きいほどペアになりたい度がより高いことを表す．



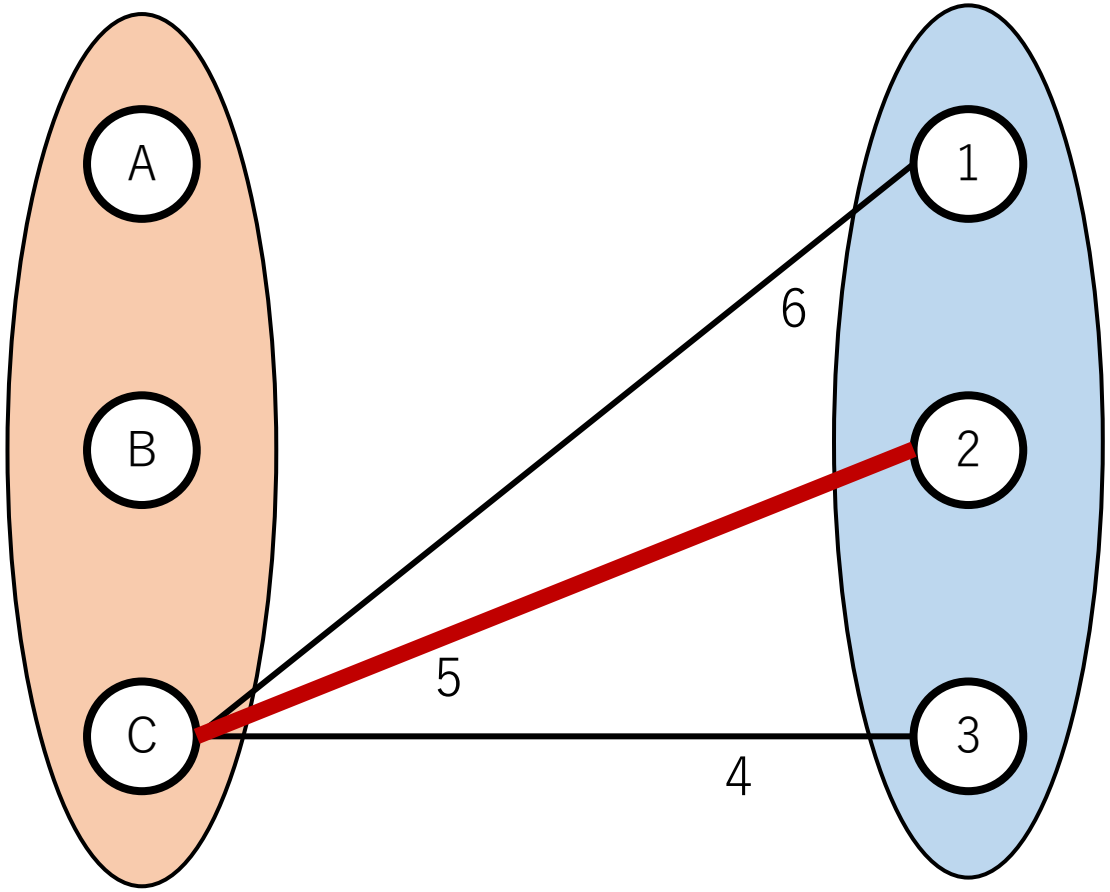
重み付き二部グラフの最大マッチング問題

この場合は以下のようにつないで18.



重み付き二部グラフの最大マッチング問題

個人レベルでは必ずしも最大にならない.



重み付き二部グラフの最大マッチング問題

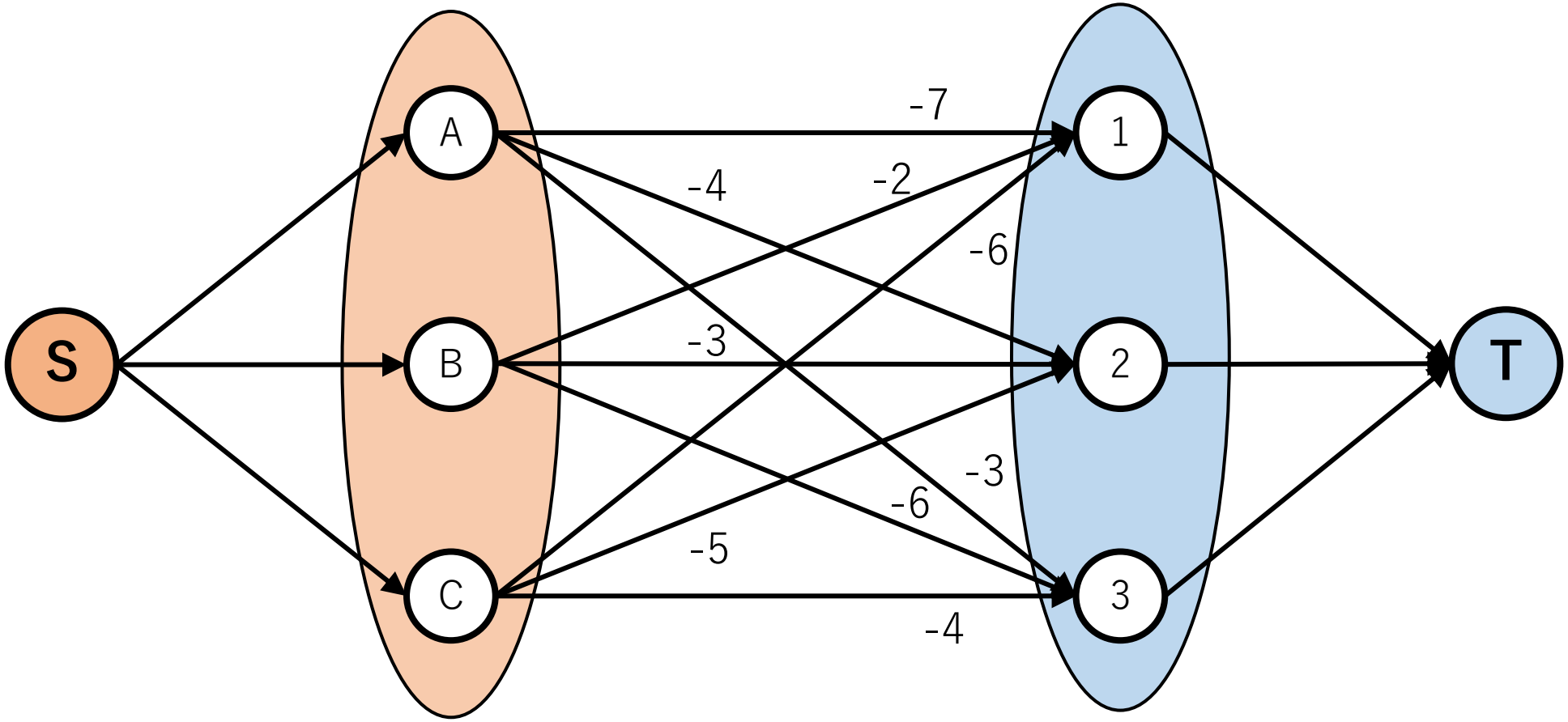
これを最小費用流問題に置き換えて考える.

辺の重みを負にして，辺のコストと見立てる.

元々の重みが大きければ大きいほど，辺のコストは小さくなる.

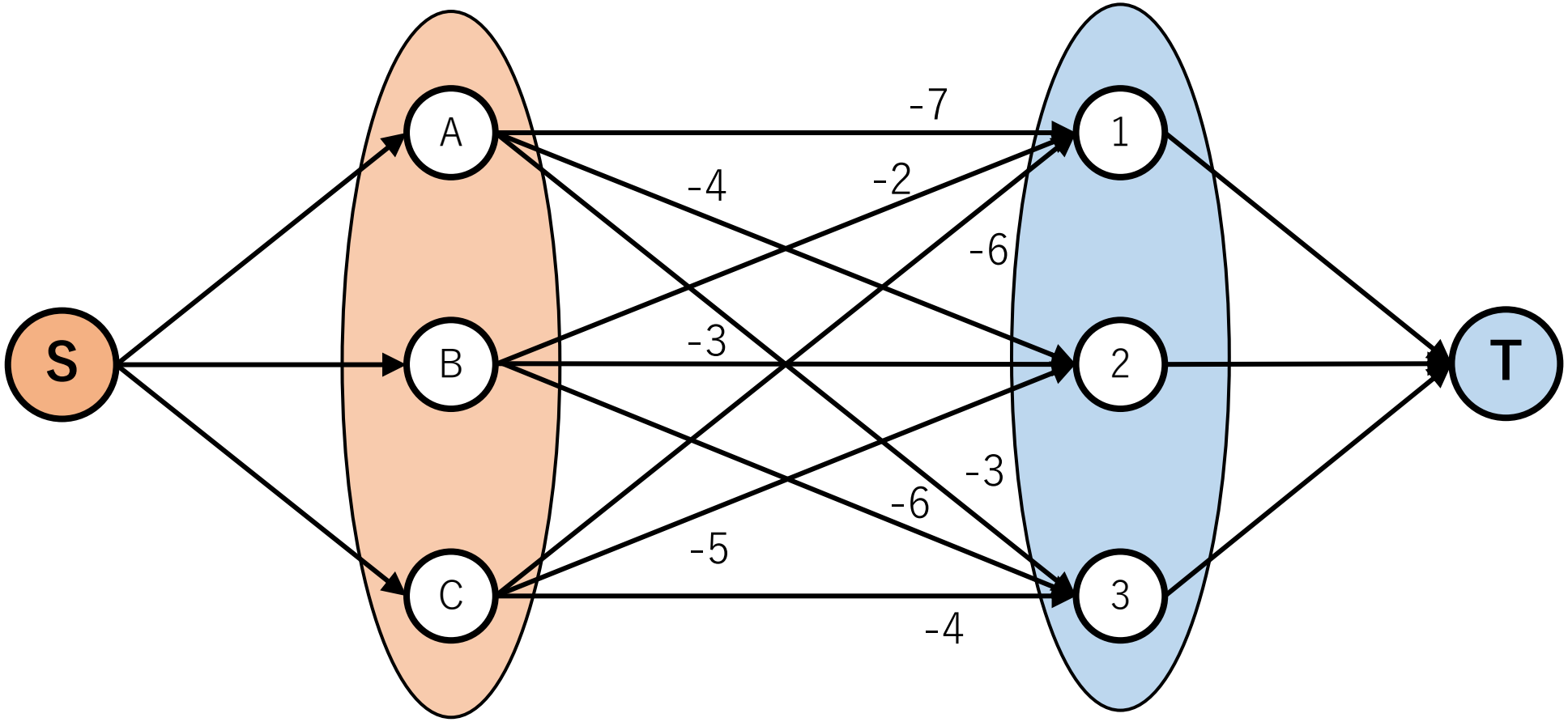
重み付き二部グラフの最大マッチング問題

仮想のSとTを用意し，さらに辺の費用を重みの負の値で設定．全ての辺の容積は1.



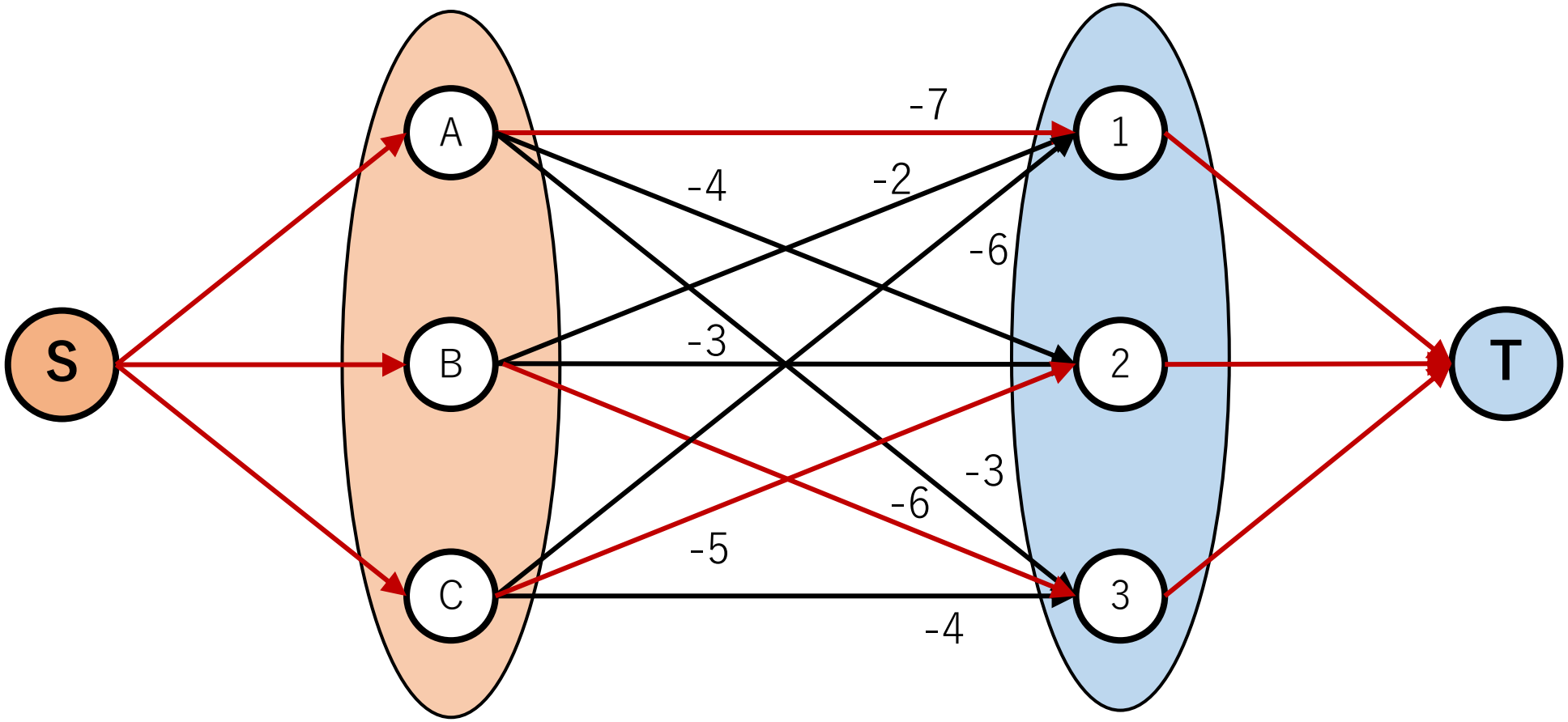
重み付き二部グラフの最大マッチング問題

ノード数（この場合3）分流す時の最小費用流を求める.



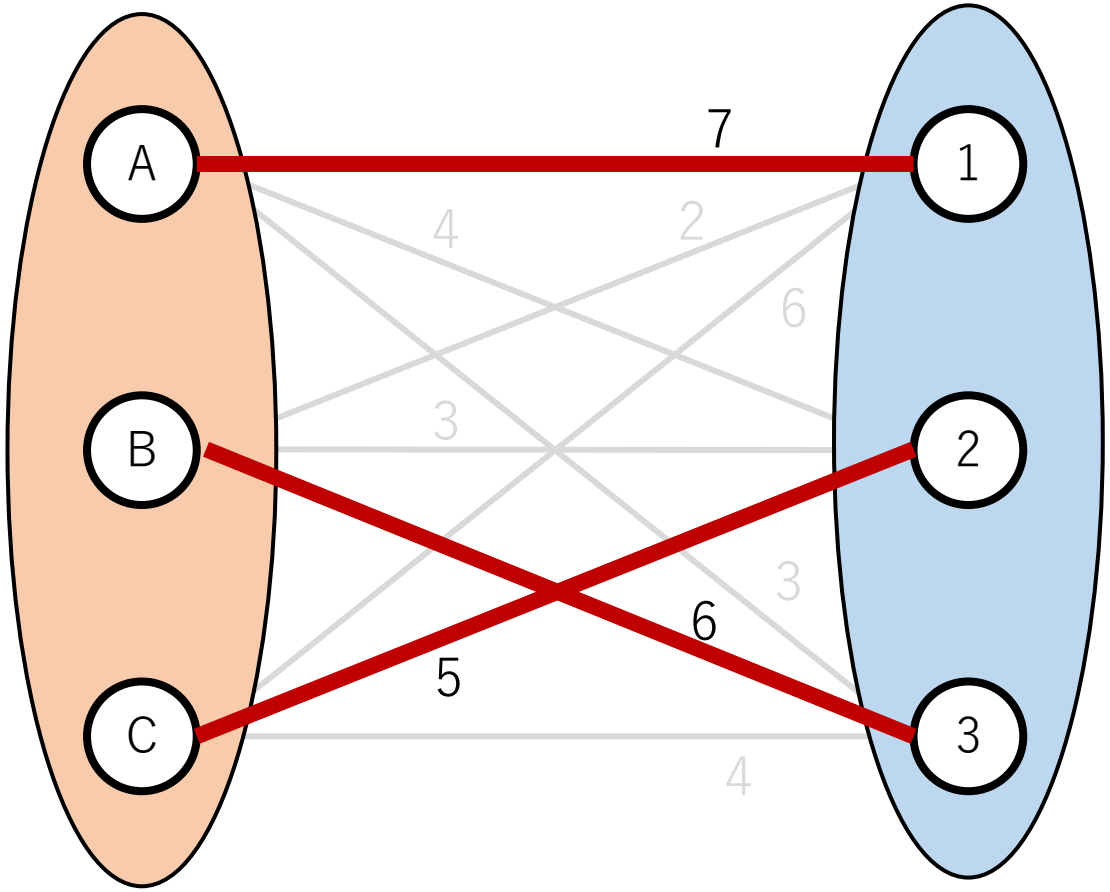
重み付き二部グラフの最大マッチング問題

ノード数（この場合3）分流す時の最小費用流を求める.



重み付き二部グラフの最大マッチング問題

SとTを削除すれば，最大マッチングになっている．



二部グラフのマッチング問題

全体最適を目指すアルゴリズムとなる.

全体最適のために場合によっては大きく不利益を被る個人が出てくることがある.

お互いに現在よりも好ましい組が存在しないようなマッチング方法もある（安定マッチング，ゲール・シャプレイのアルゴリズム）.

駒場での進学選択で使われています. 😊

http://www.c.u-tokyo.ac.jp/zenki/news/kyoumu/2016guidance_algorithm1128.pdf

今日のお題

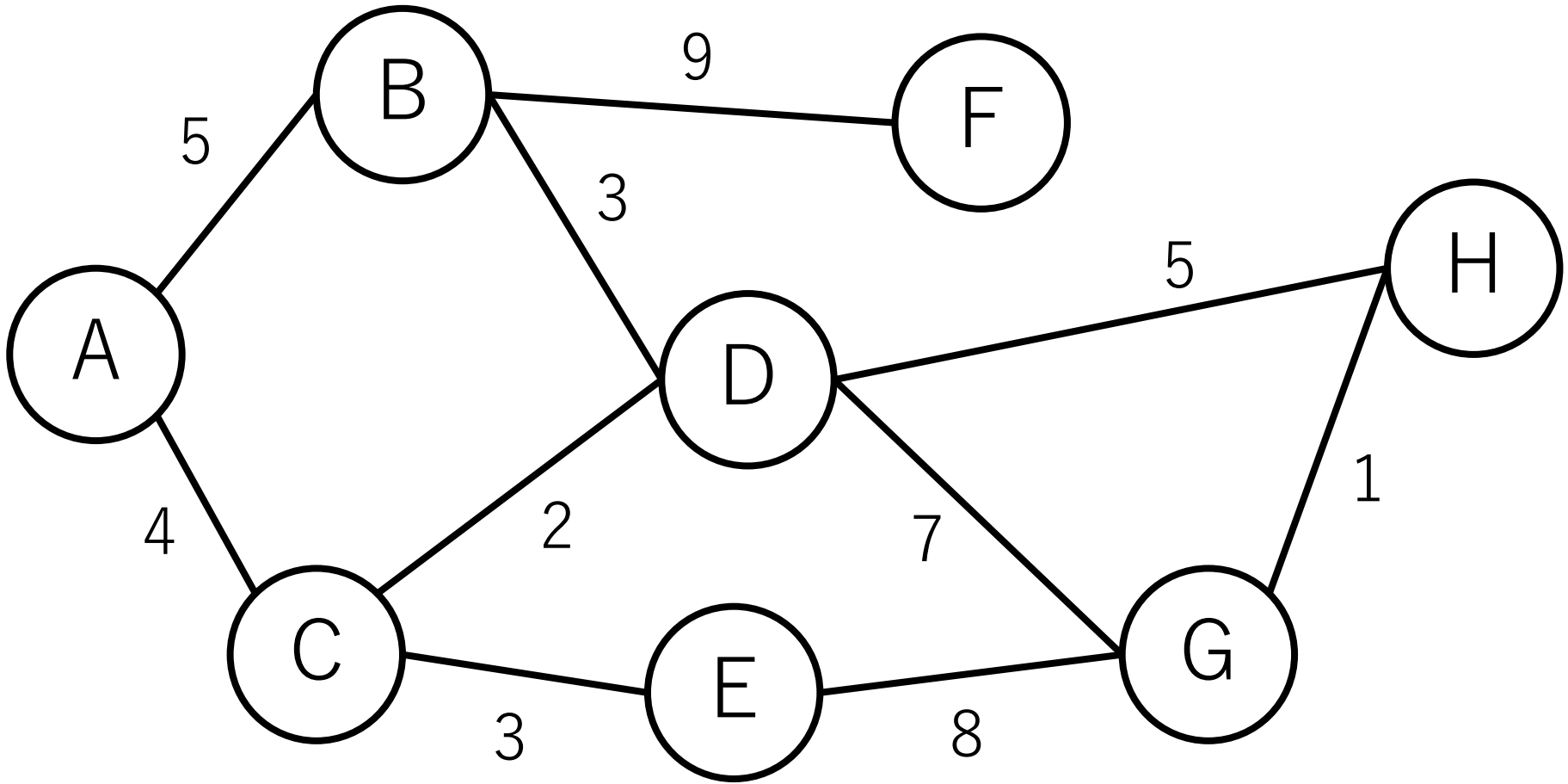
二部グラフのマッチング問題

最小全域木

本講義のまとめ

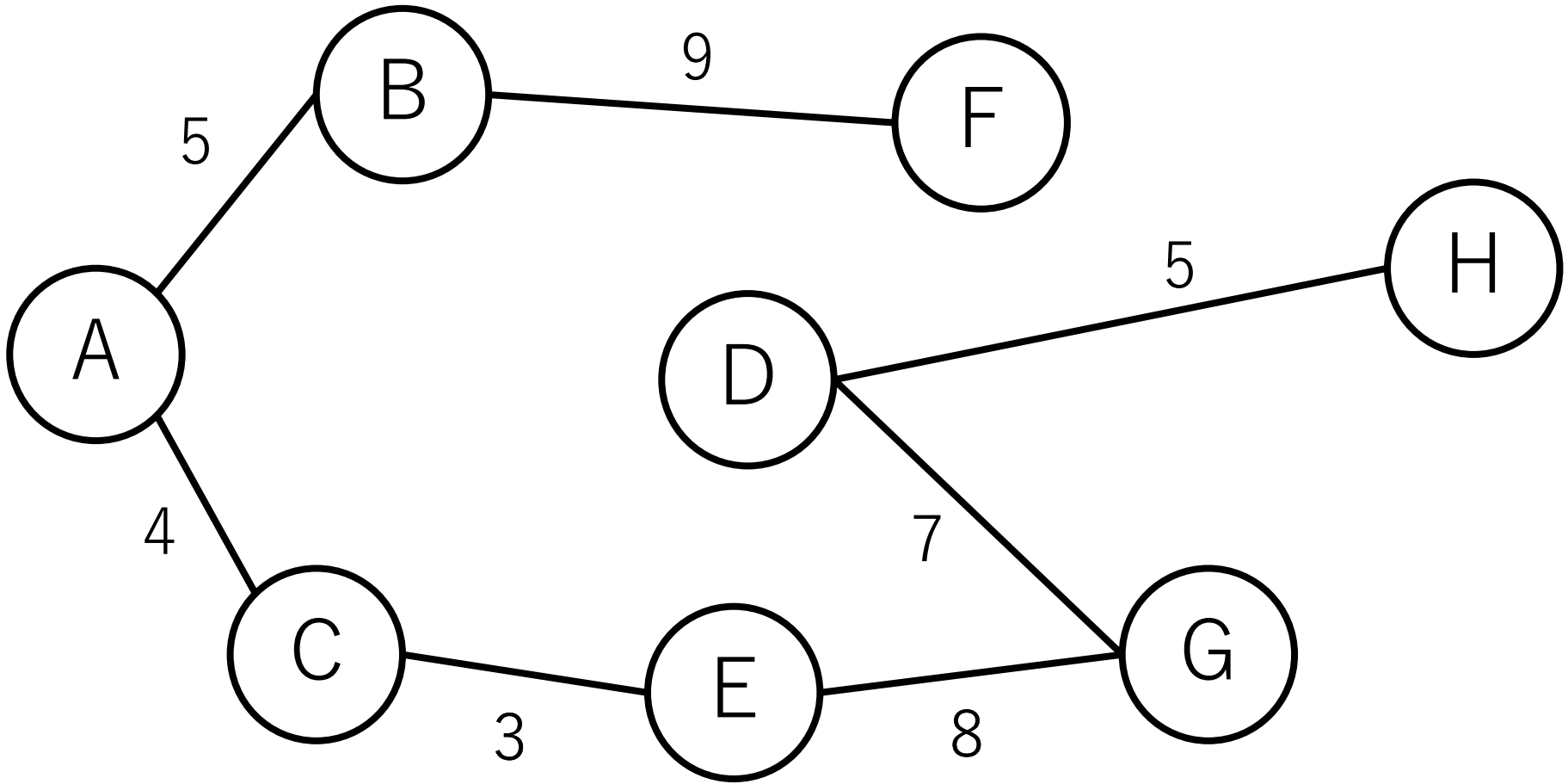
全域木（spanning tree）

グラフにおいて，すべての頂点がつながっている木（閉路を持たないグラフ）．



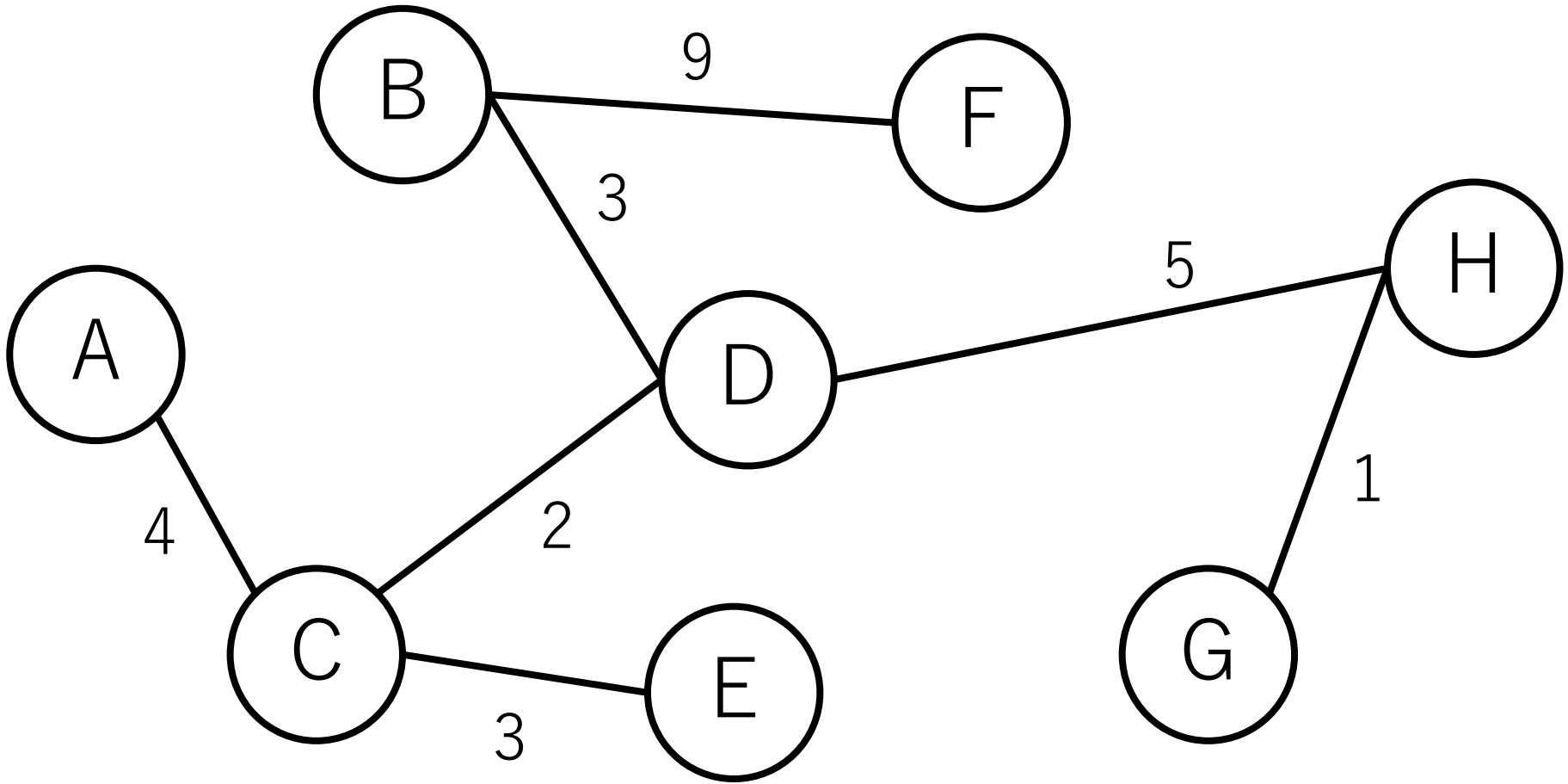
全域木

全域木の1例.



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.

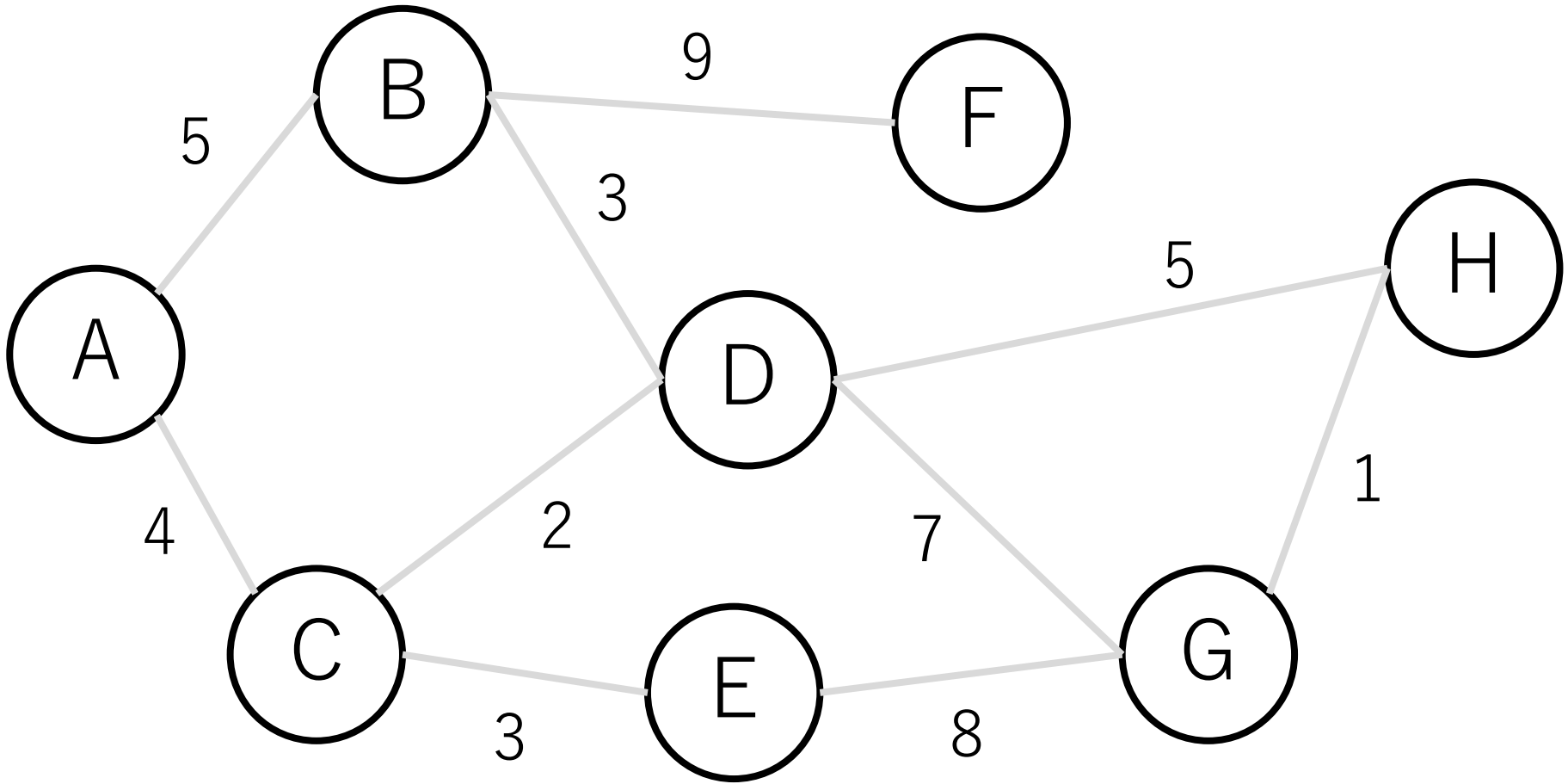
#2 一番距離の短い辺からスタート.

#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.

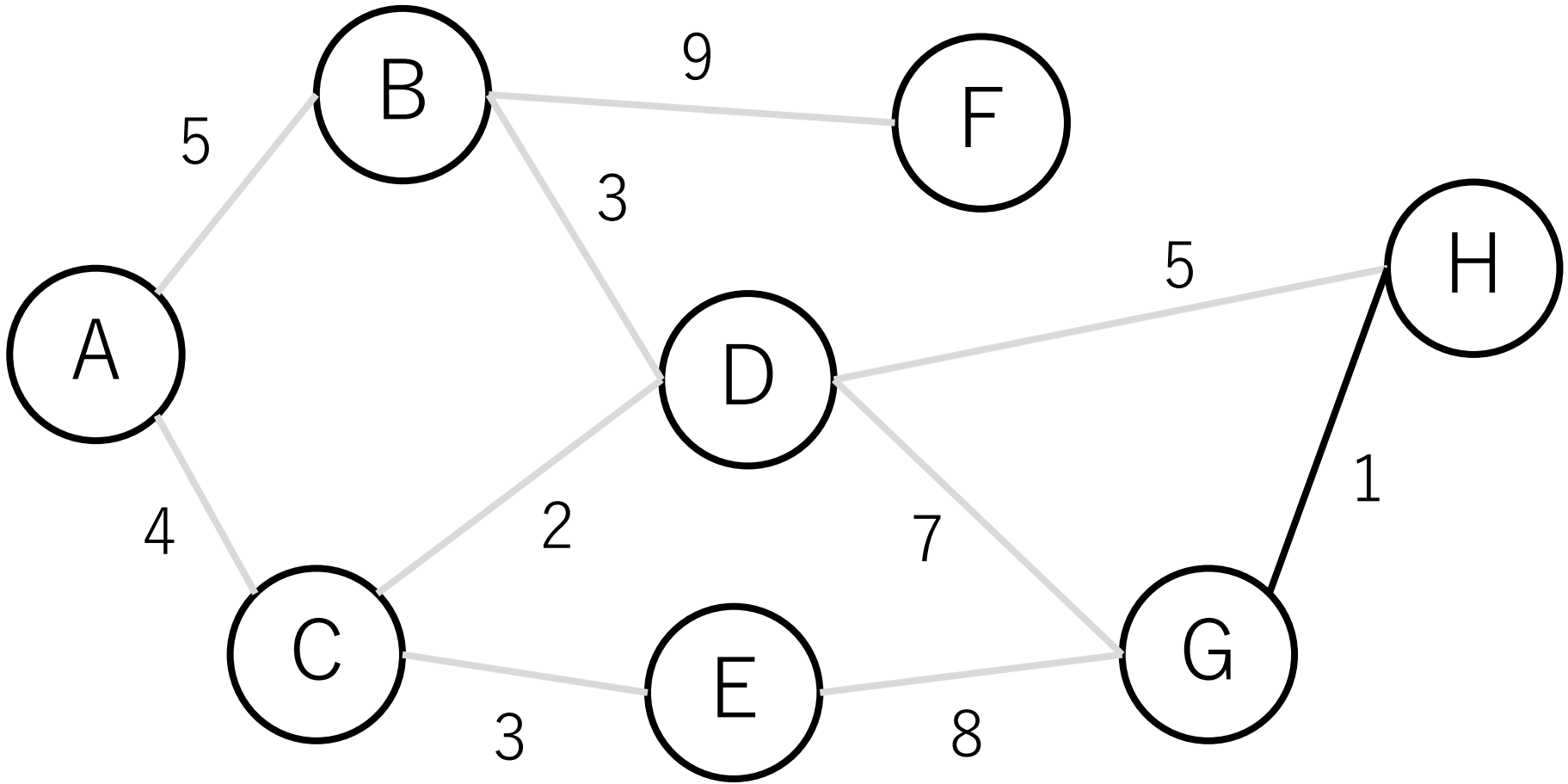
クラスカル法の例

すべての辺を距離の短い順にソート.



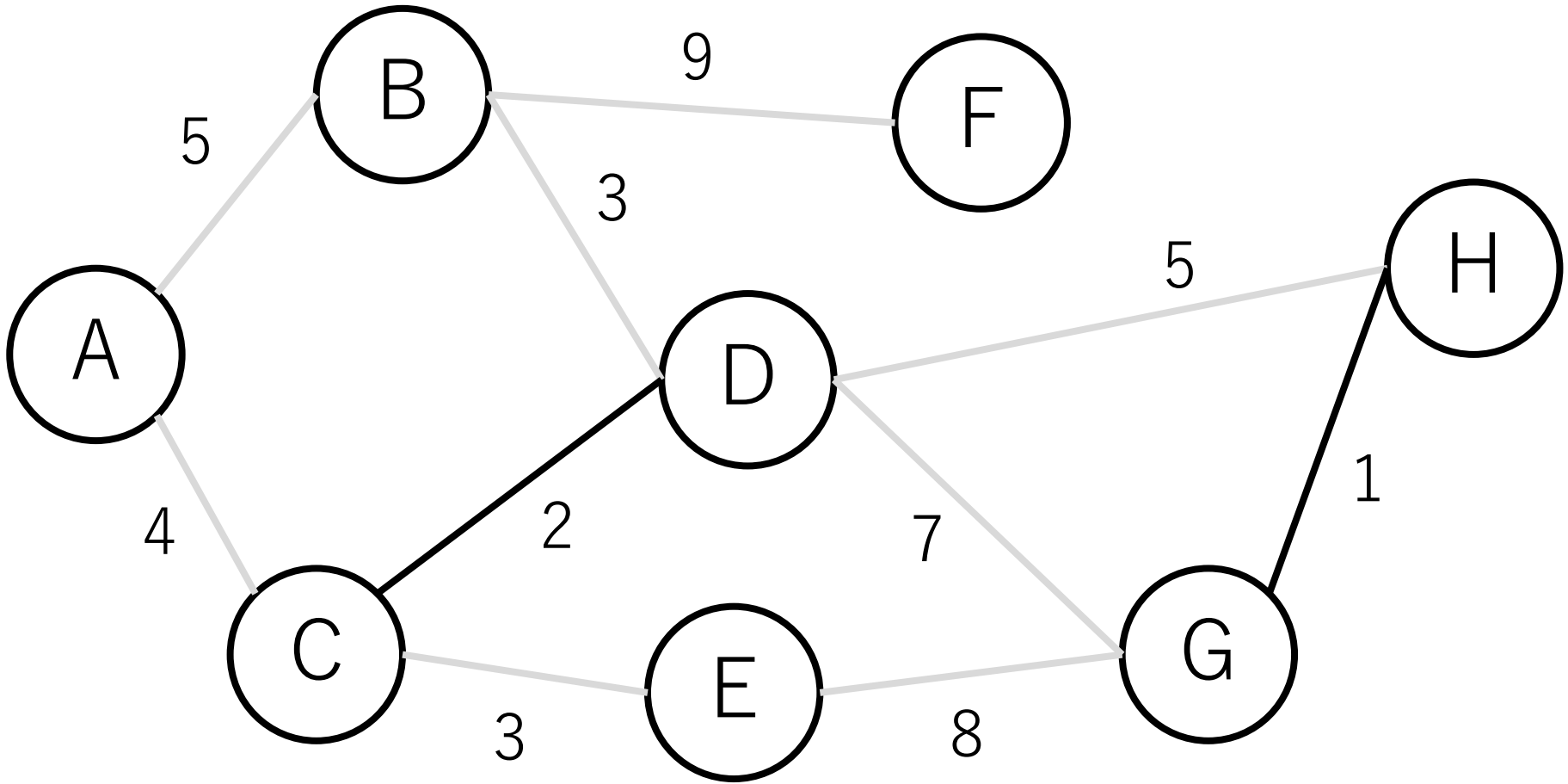
クラスカル法の例

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．



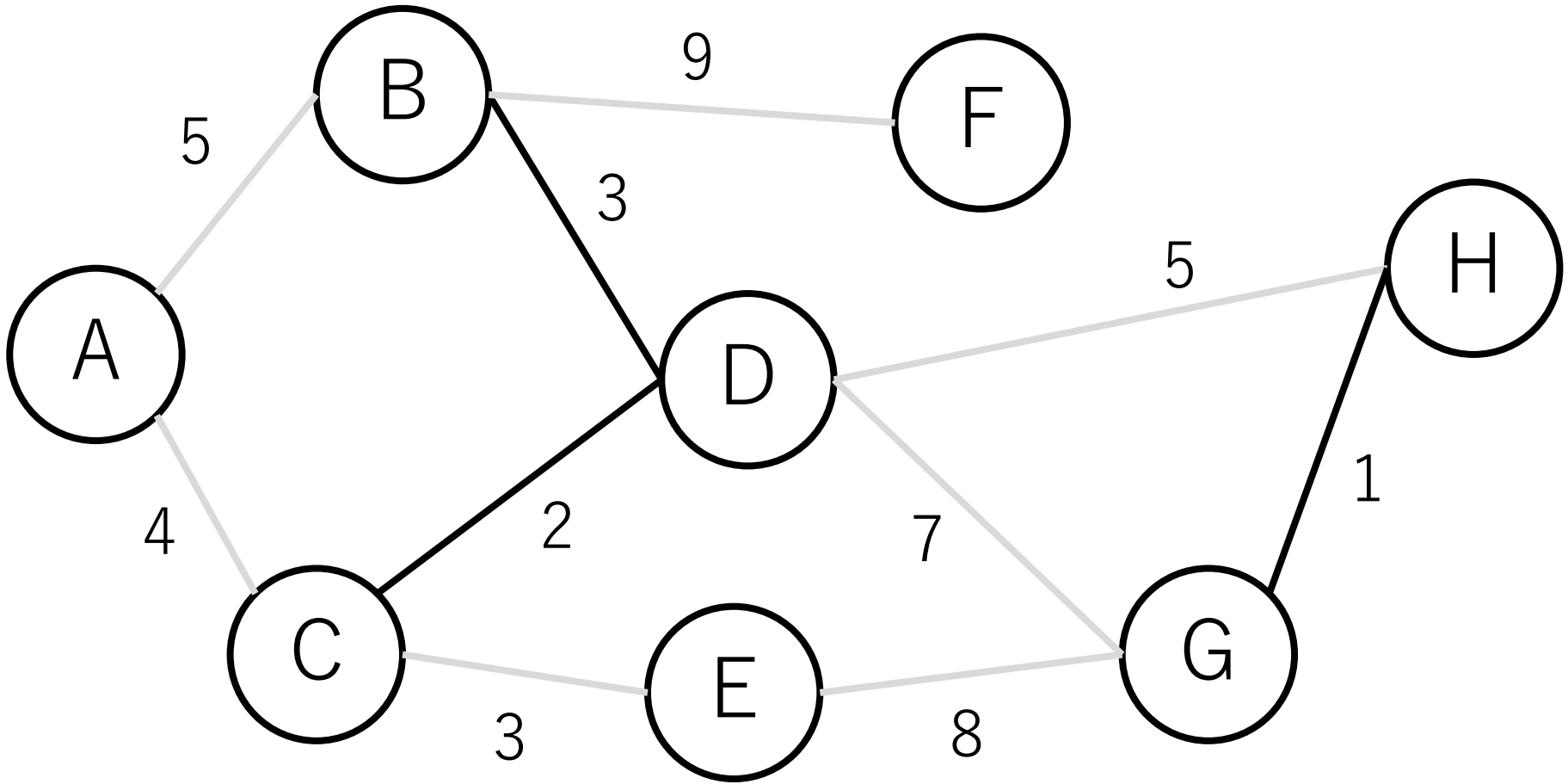
クラスカル法の例

C-Dも同じ.



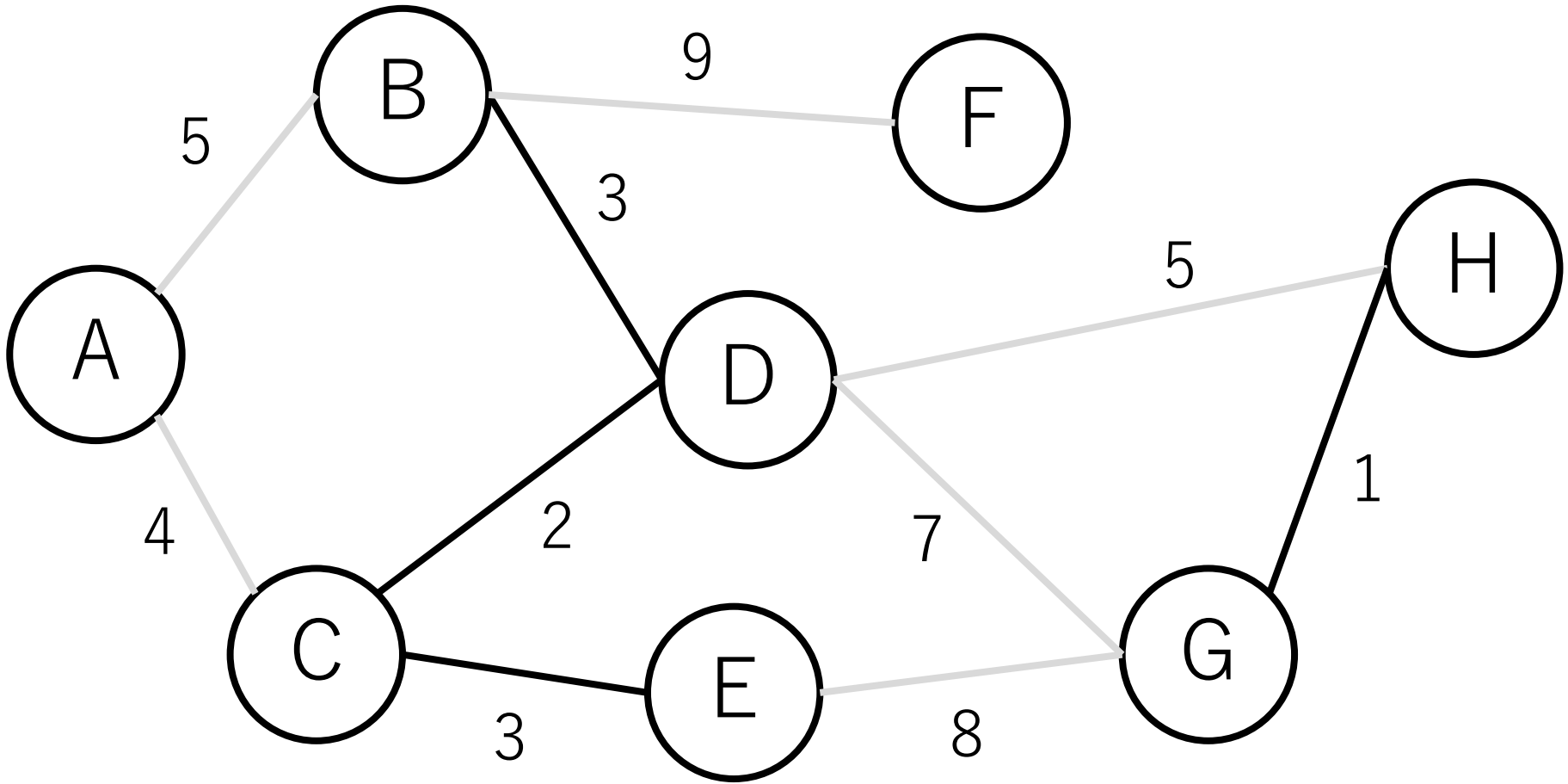
クラスカル法の例

B-Dも同じ.



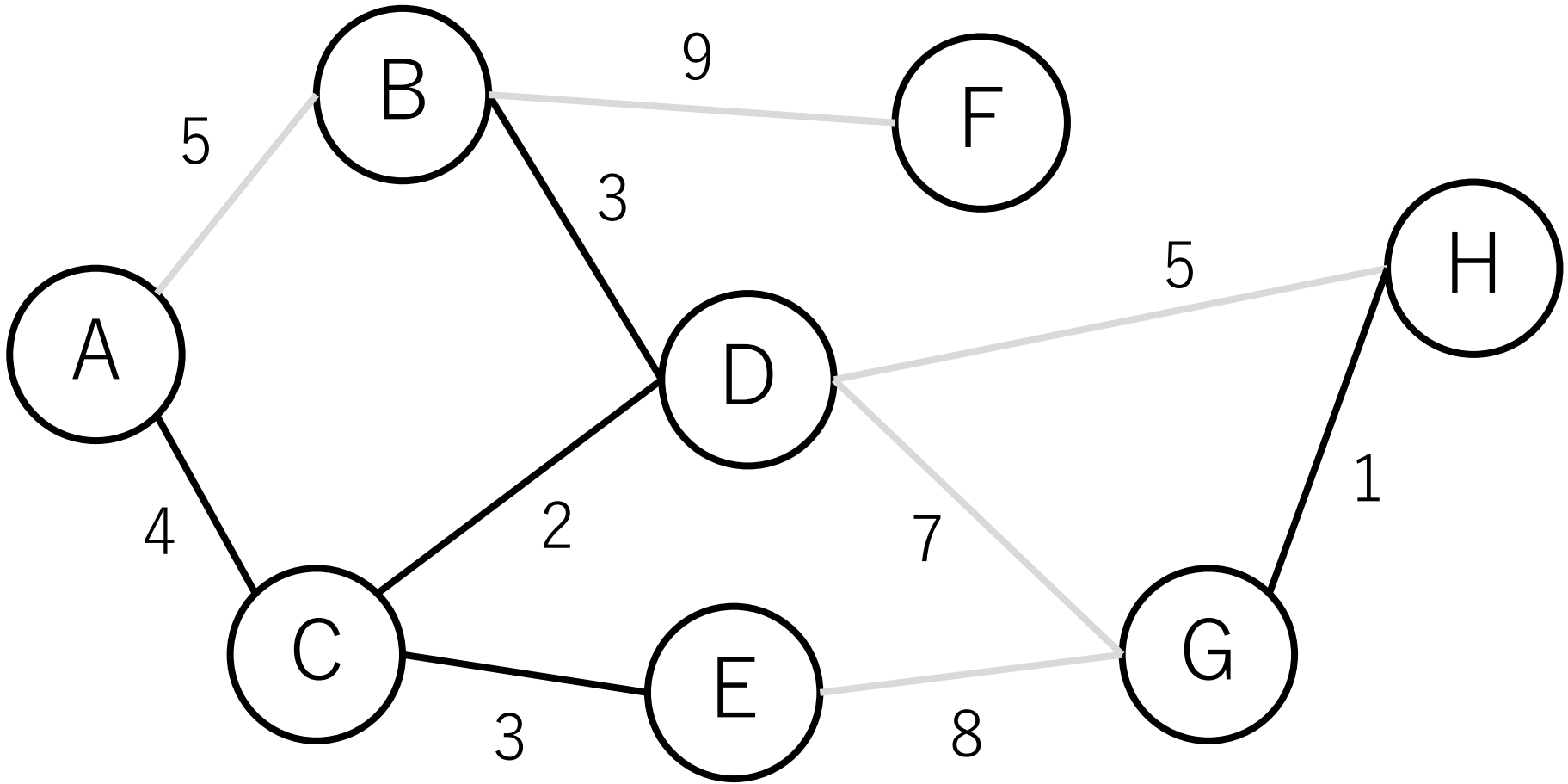
クラスカル法の例

C-Eも同じ.



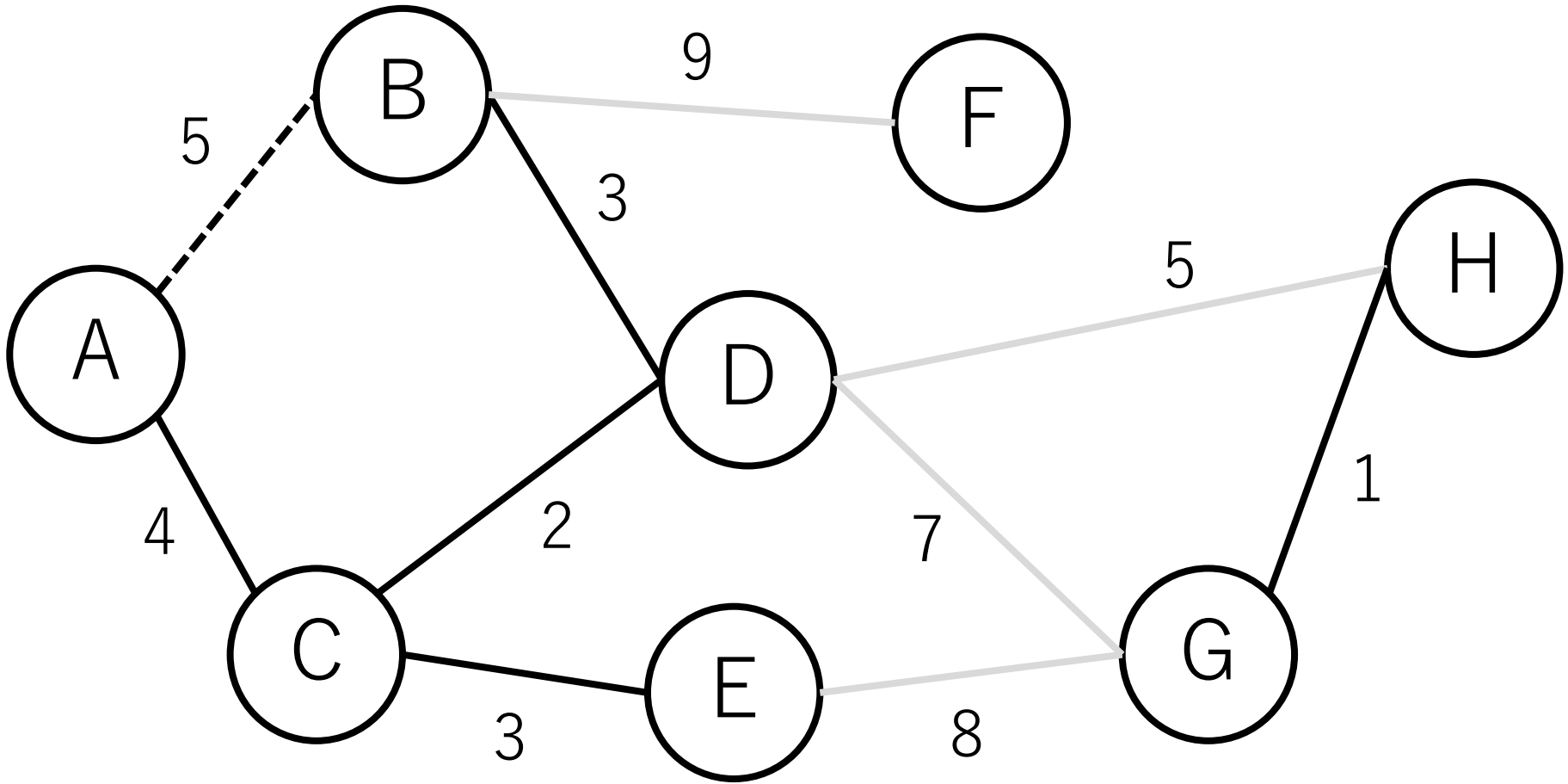
クラスカル法の例

A-Cも同じ.



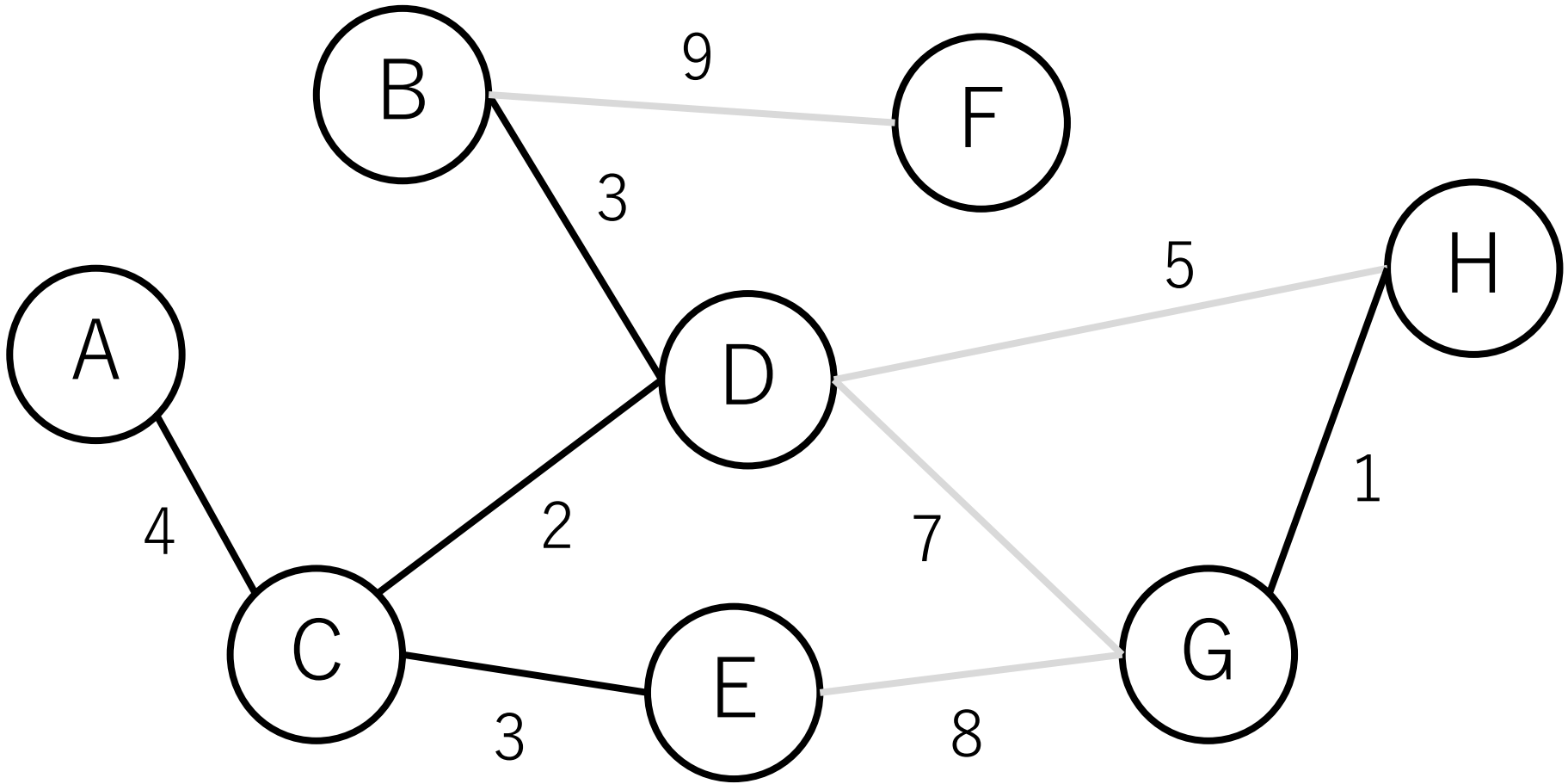
クラスカル法の例

A-Bをつないでしまうと閉路ができてしまう.



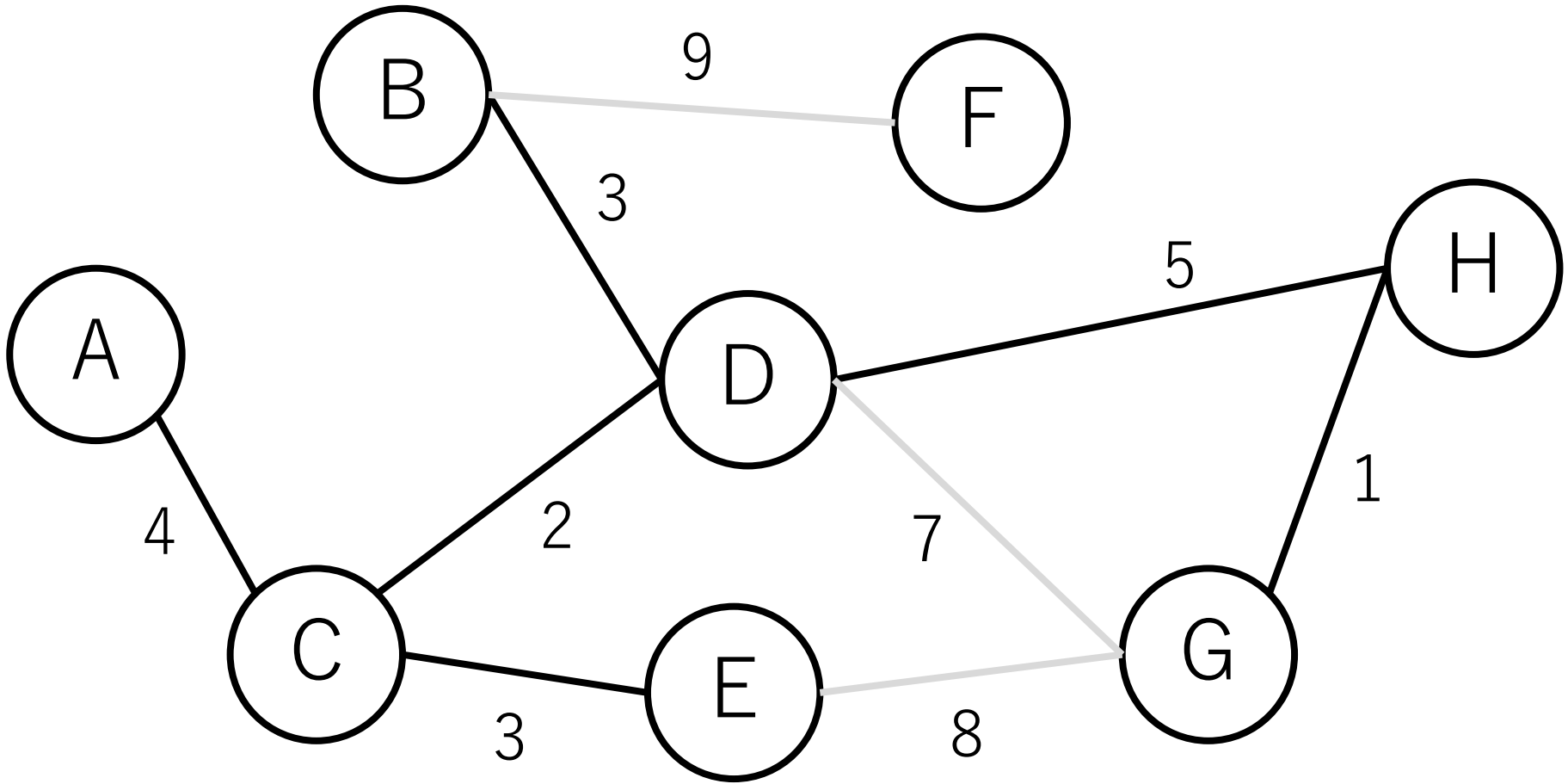
クラスカル法の例

よって，A-Bは入れない．



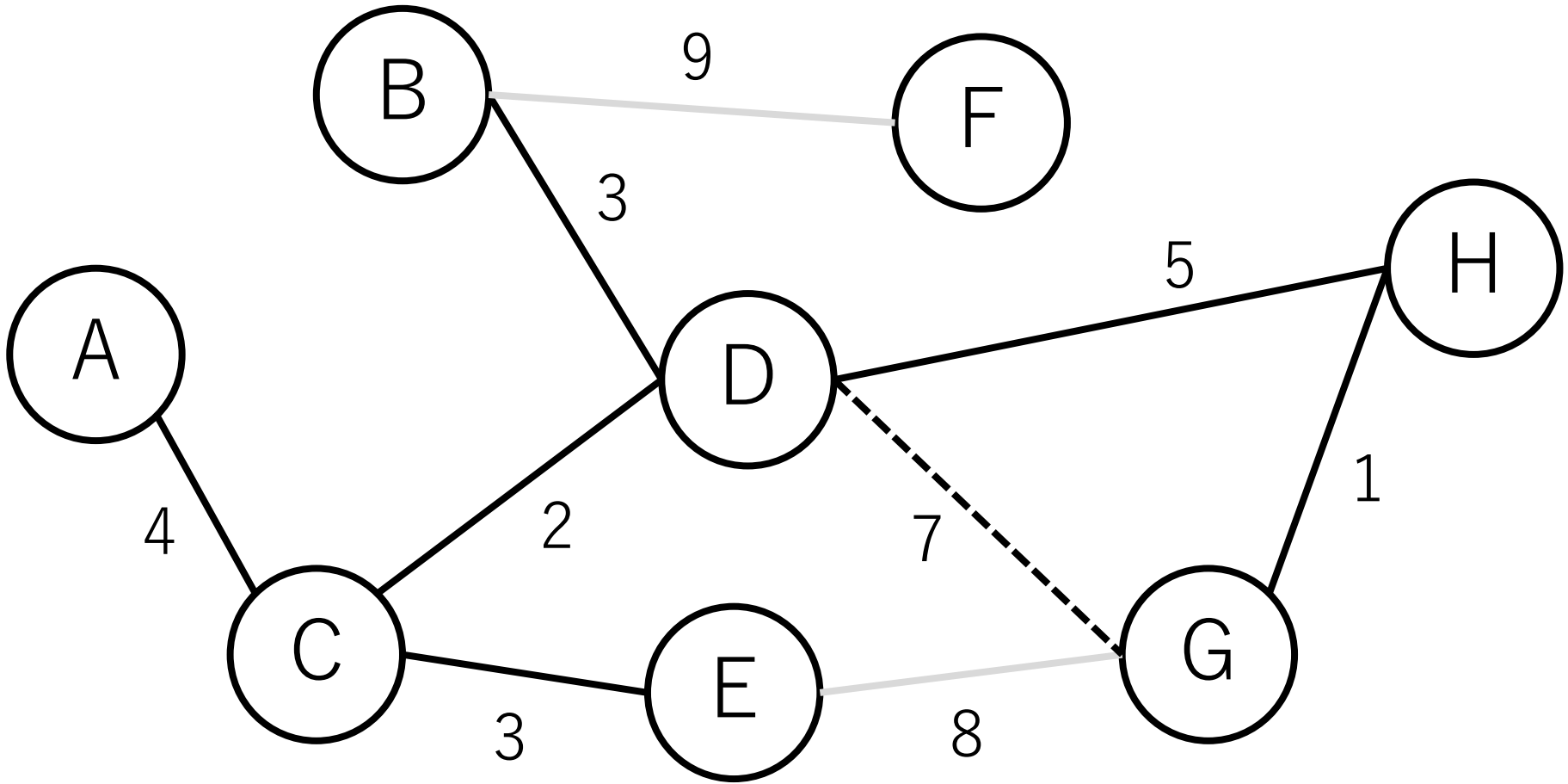
クラスカル法の例

D-Hは入れる.



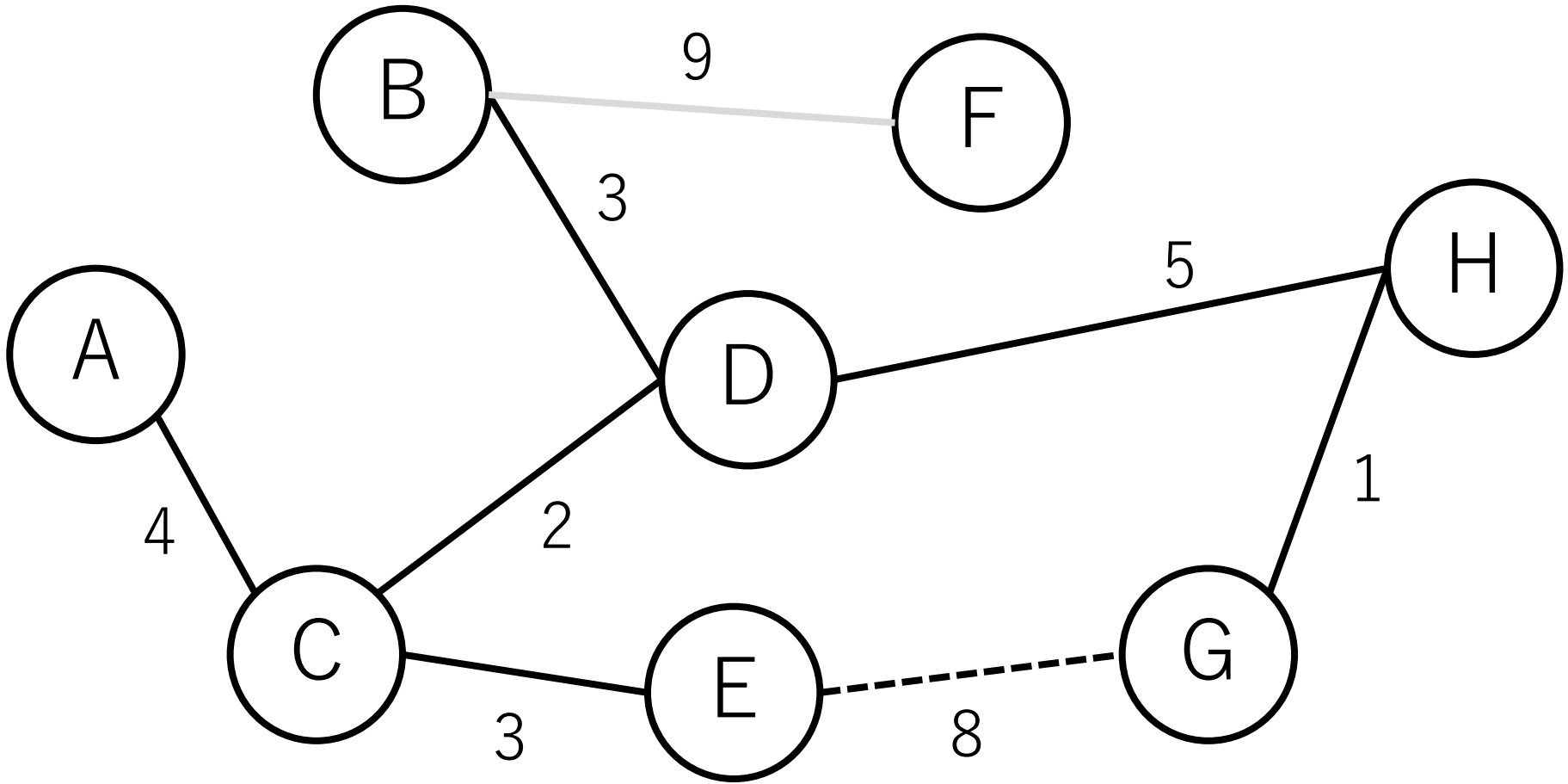
クラスカル法の例

D-Gは入れない.



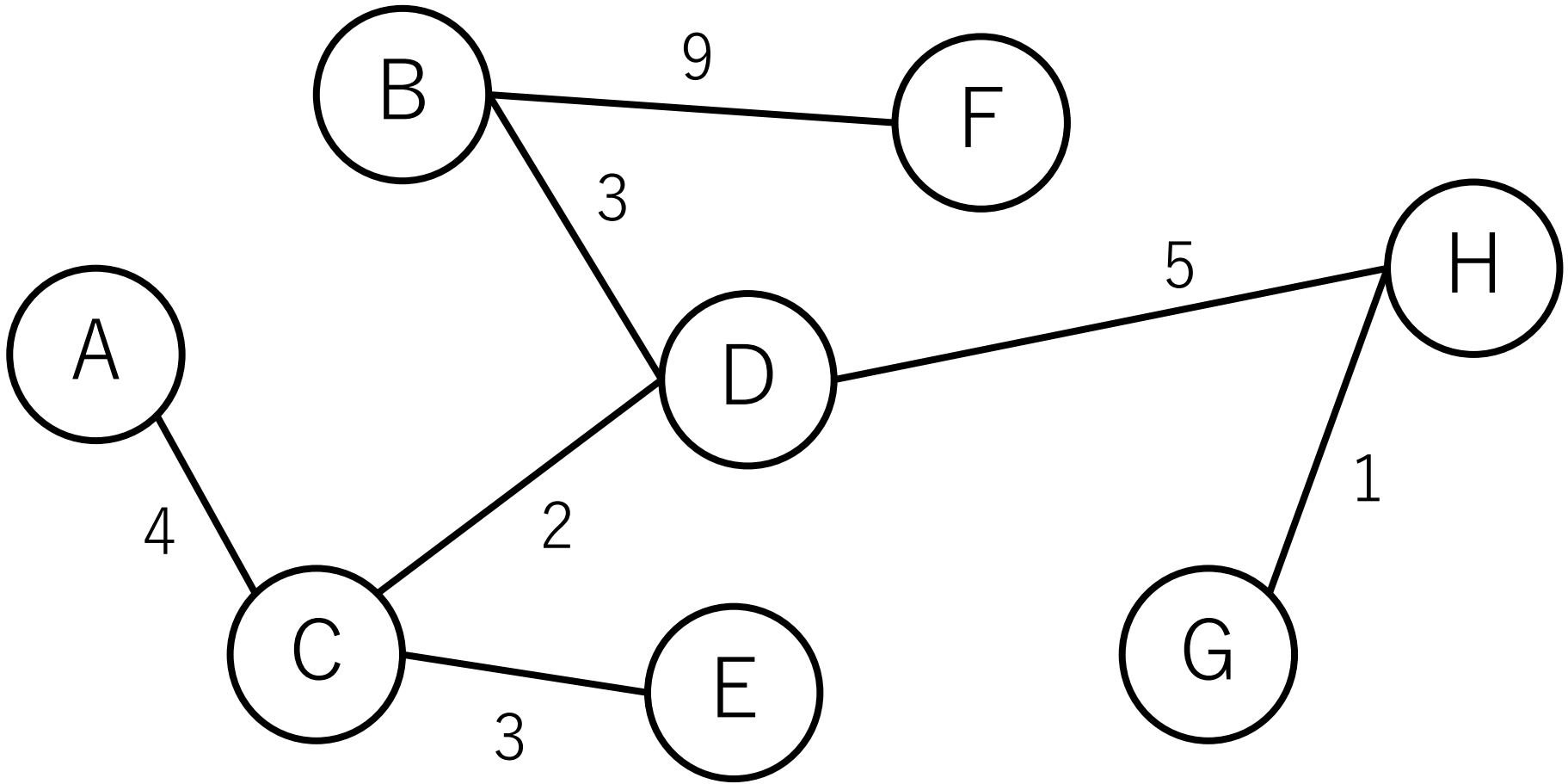
クラスカル法の例

E-Gは入れない.



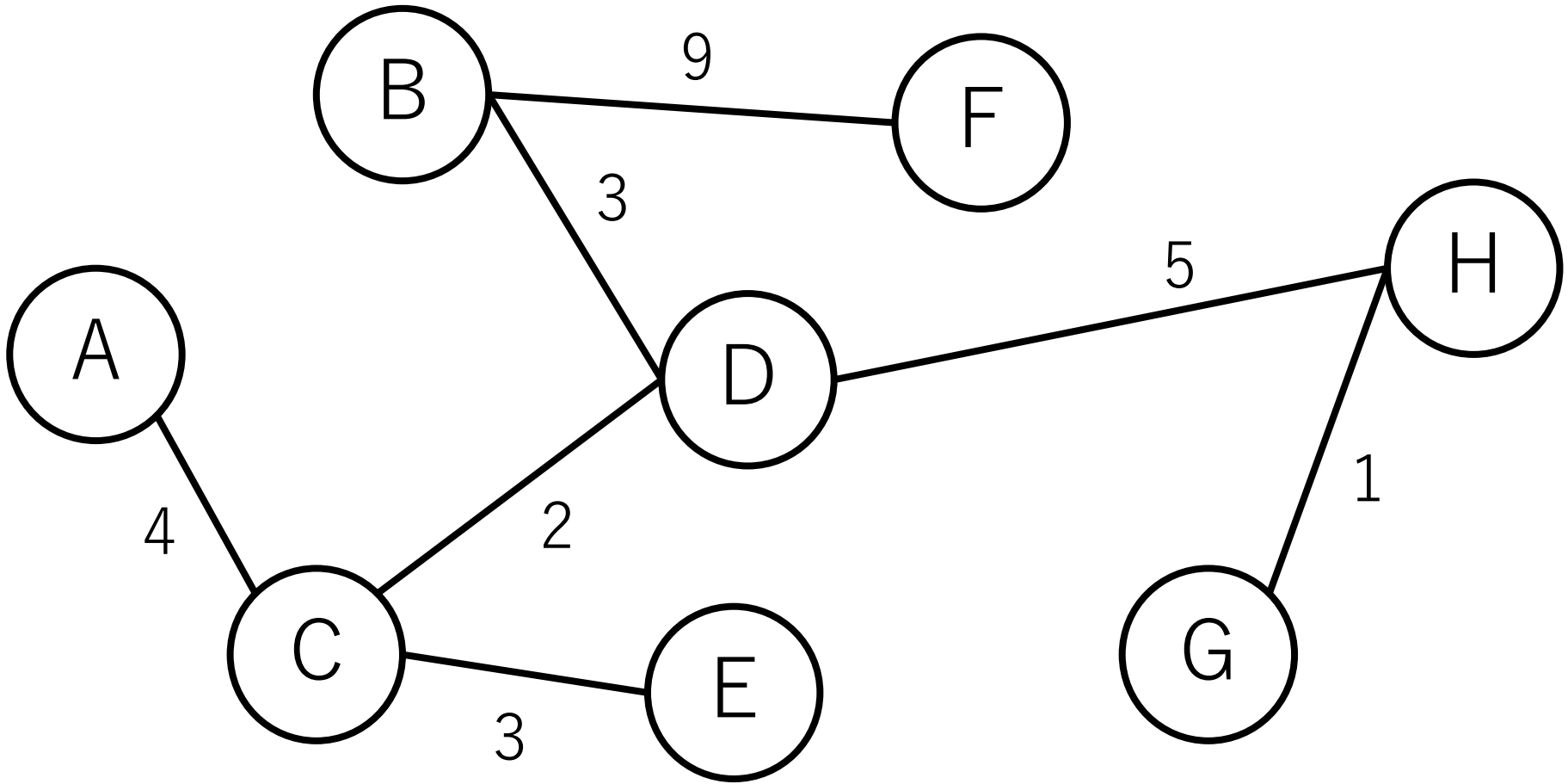
クラスカル法の例

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？

辺を足すごとに毎回グラフをたどるのは非現実的.

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.

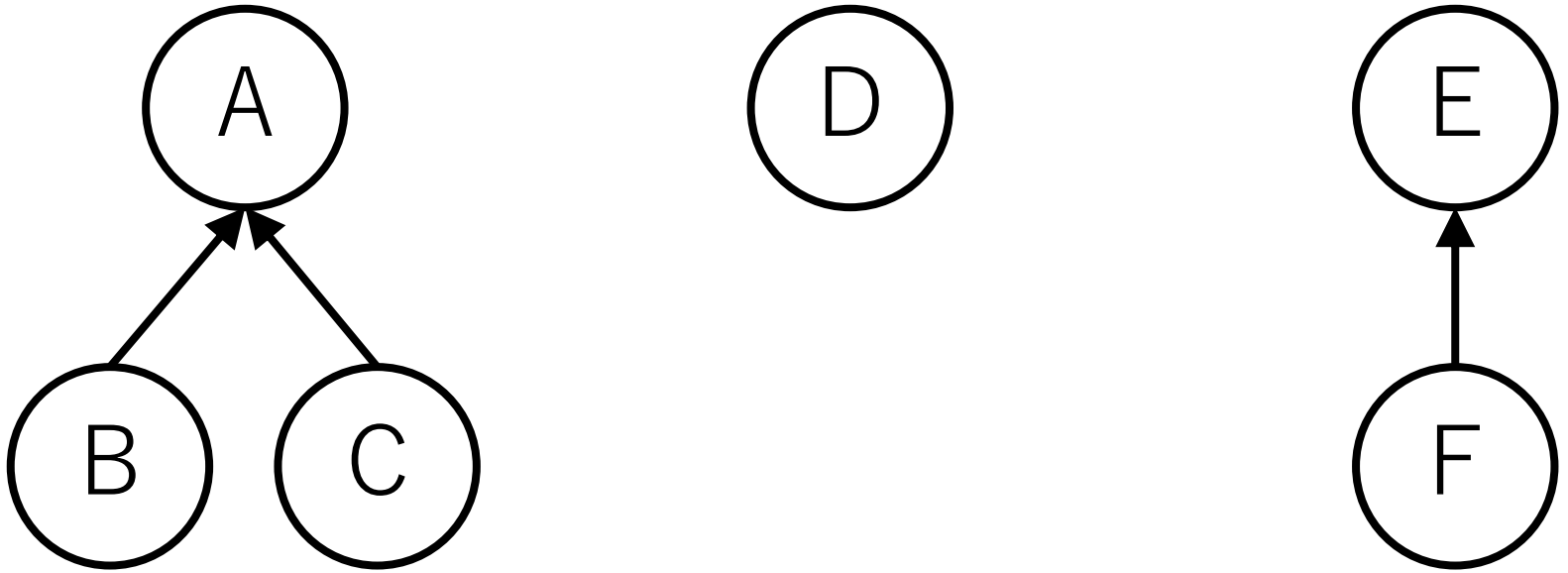
Union：2つの集合をマージする.

Find：ある要素がどの集合にいるかを見つける.

集合を分解する操作はできない.

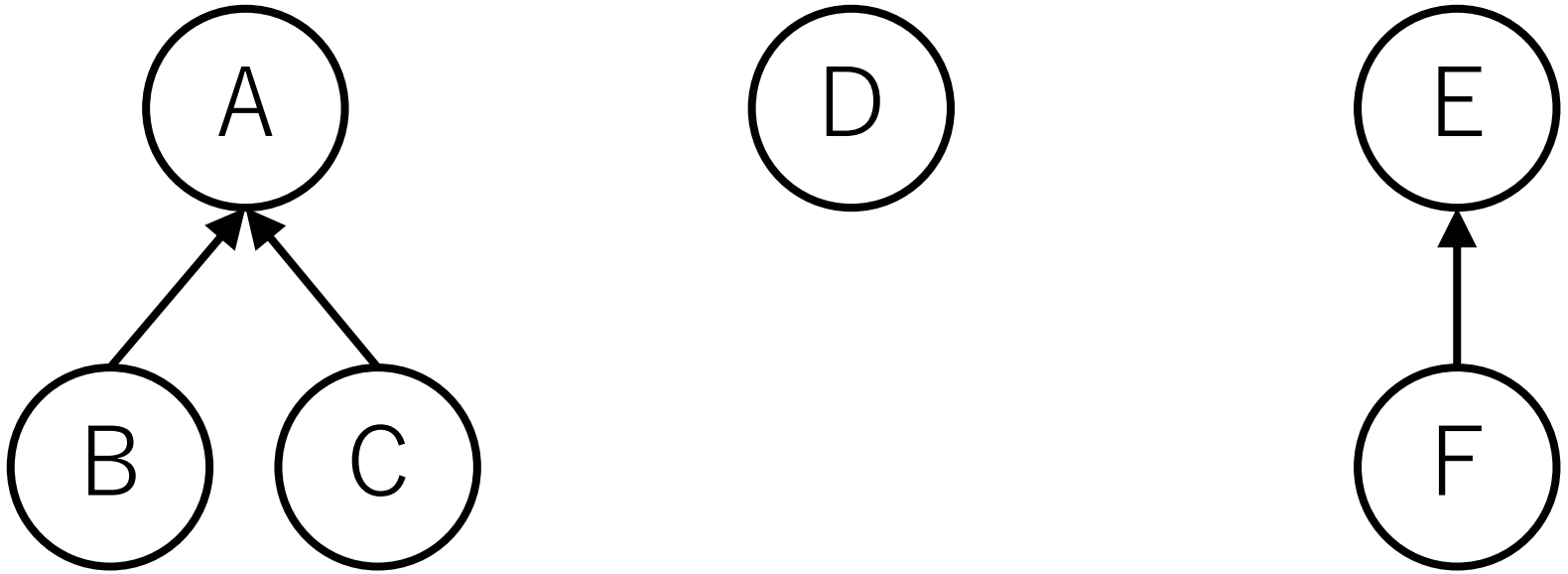
Union-Find木の例

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）.



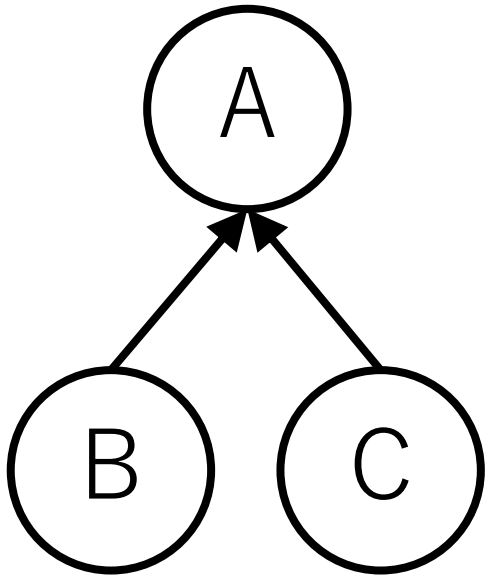
Union-Find木の例

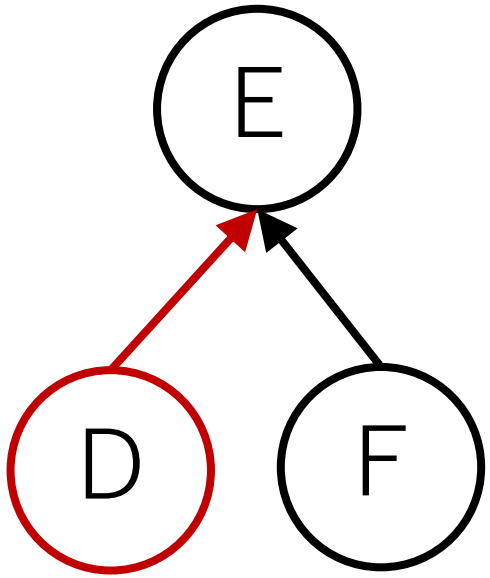
Unite：片方の根からもう片方の根につなぐ.



Union-Find木の例

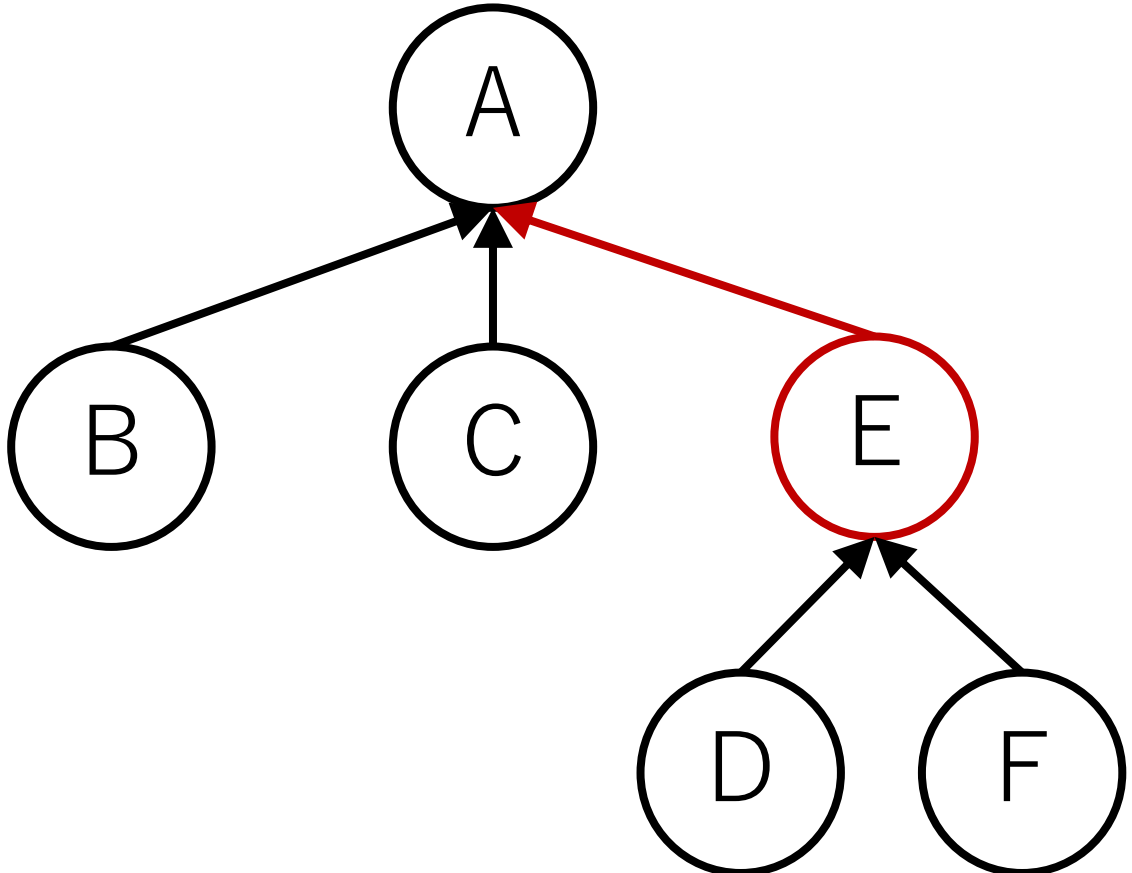
Unite：片方の根からもう片方の根につなぐ.
Dと[E，F]をマージ.





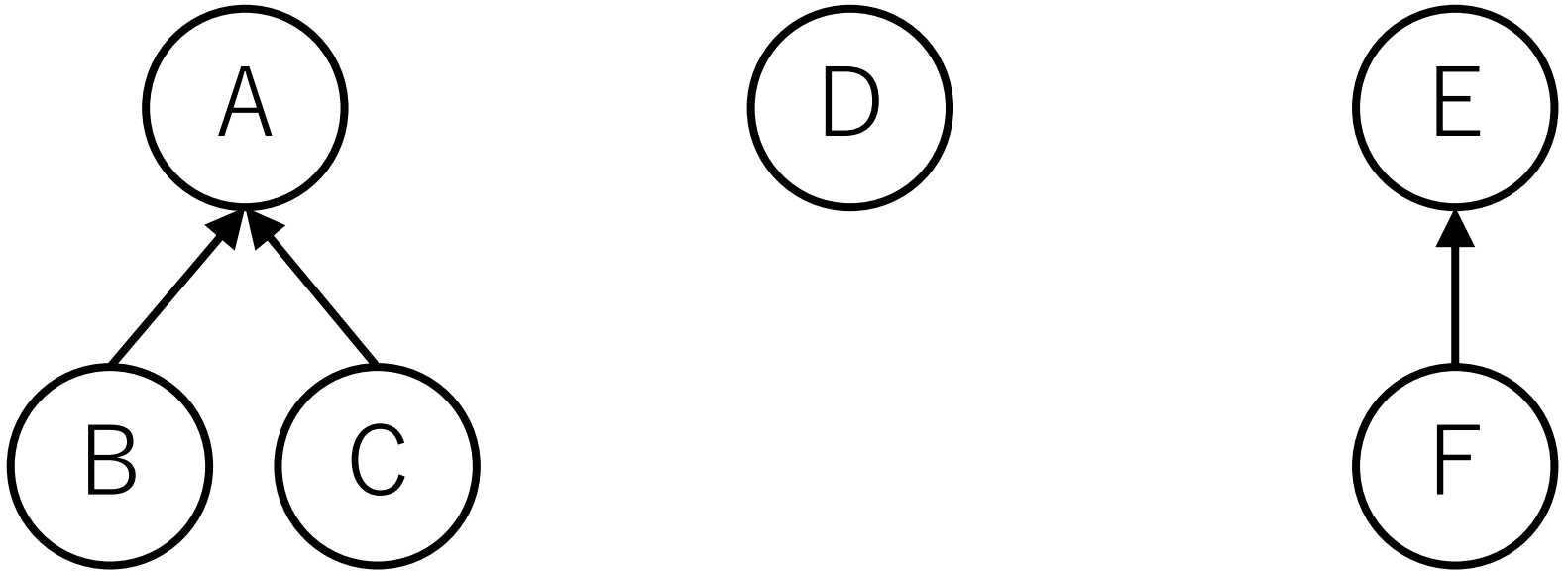
Union-Find木の例

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.



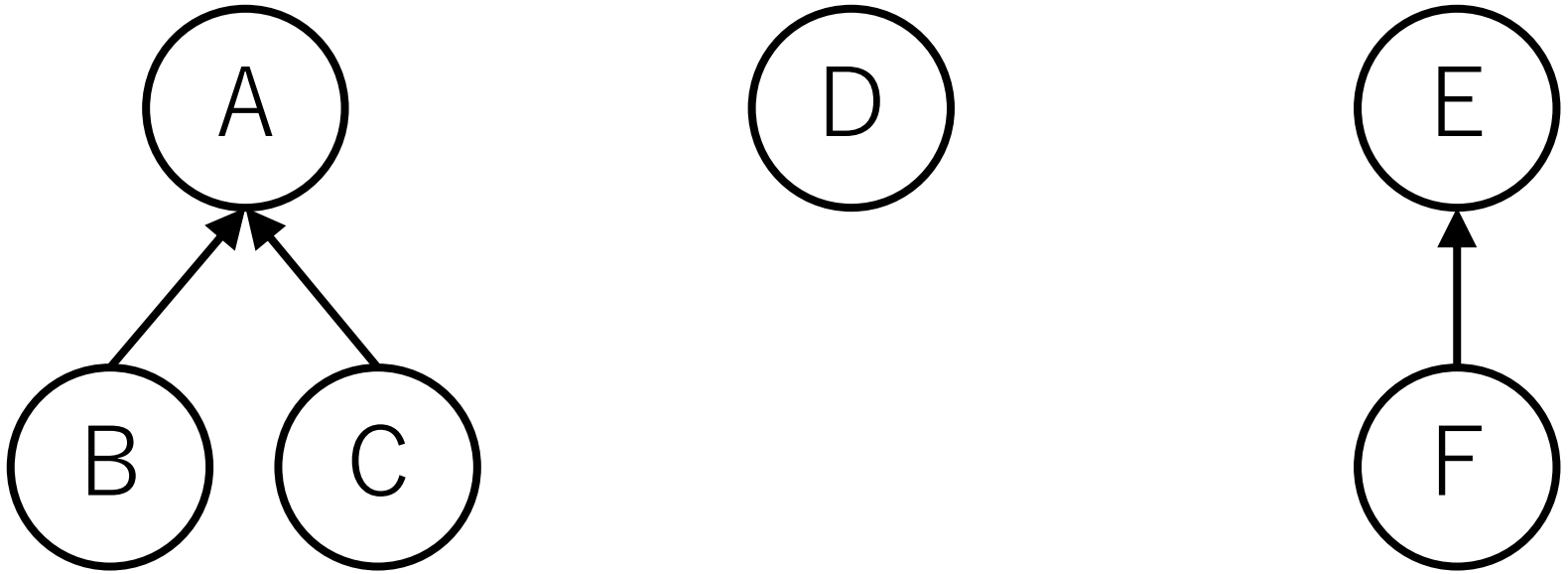
Union-Find木の例

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.

自分が子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）**

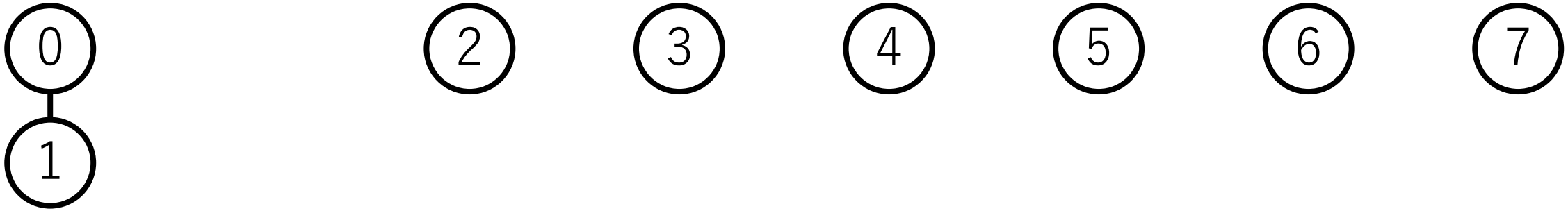
Union-Find木の例

初期化：

0	1	2	3	4	5	6	7

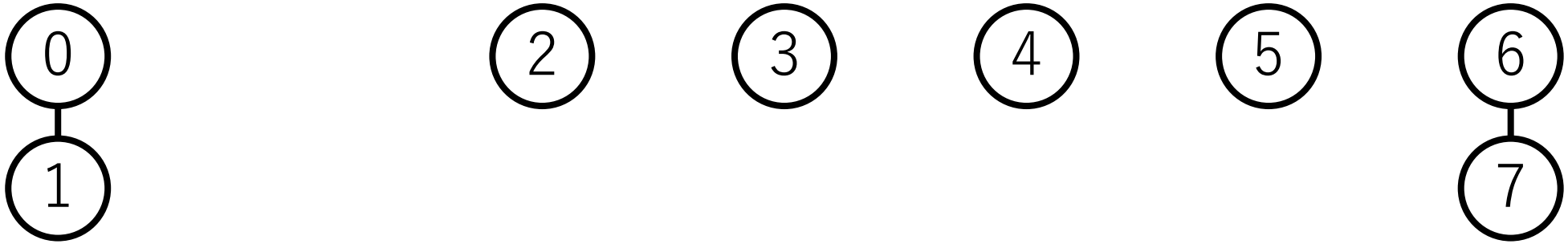
Union-Find木の例

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



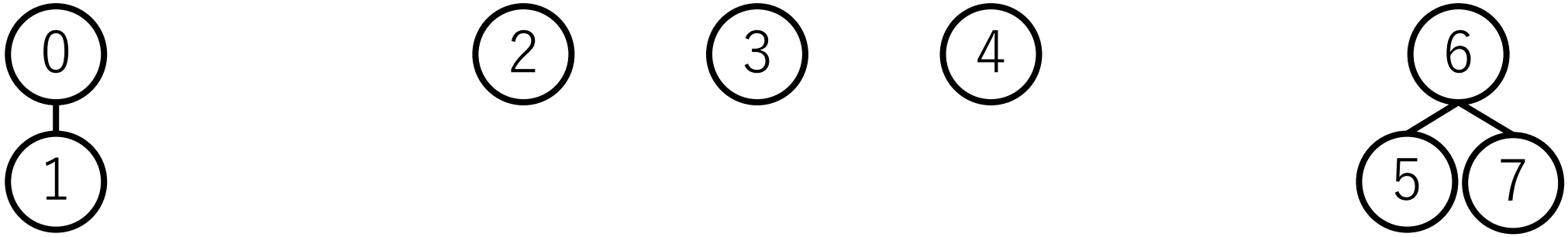
Union-Find木の例

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**



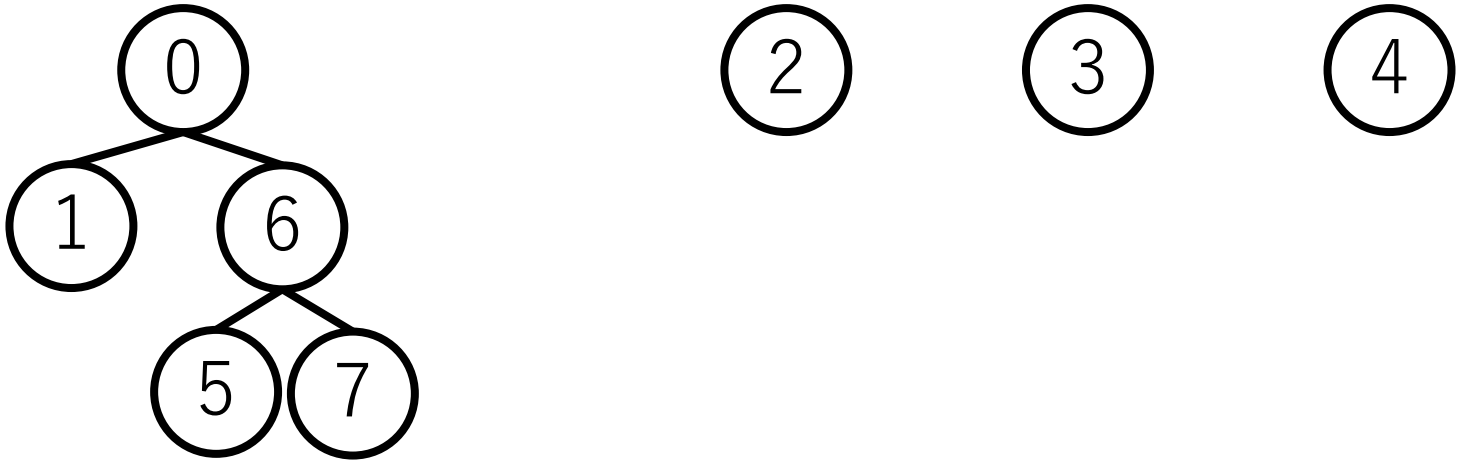
Union-Find木の例

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



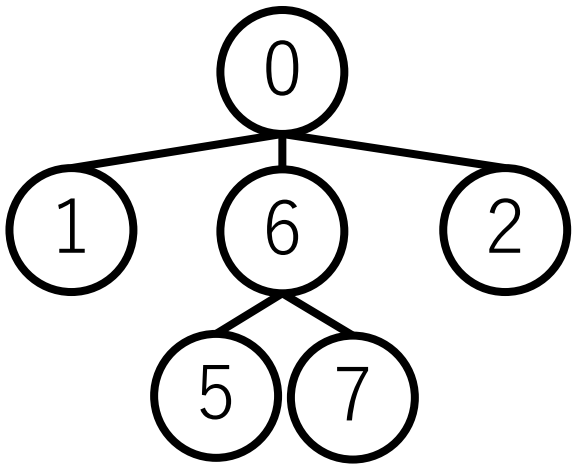
Union-Find木の例

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

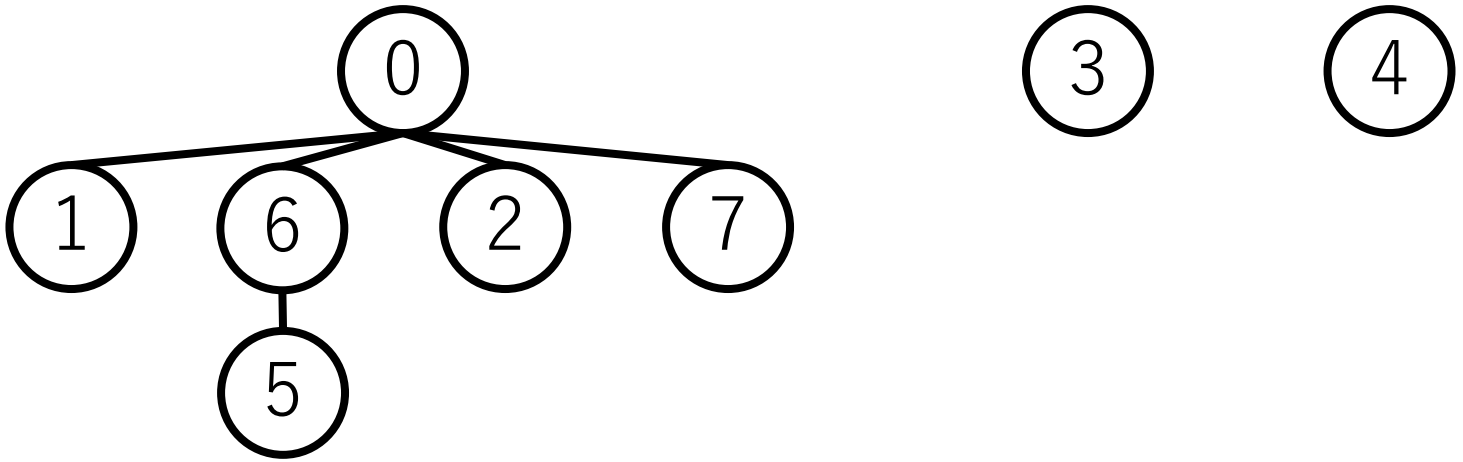


Union-Find木の例

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**



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
```

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.

$$A(0,0)=1, A(1,1)=3, A(2,2)=7, A(3,3)=61,$$

$A(4,4)=2^{2^{2^{65536}}}-3$となる.

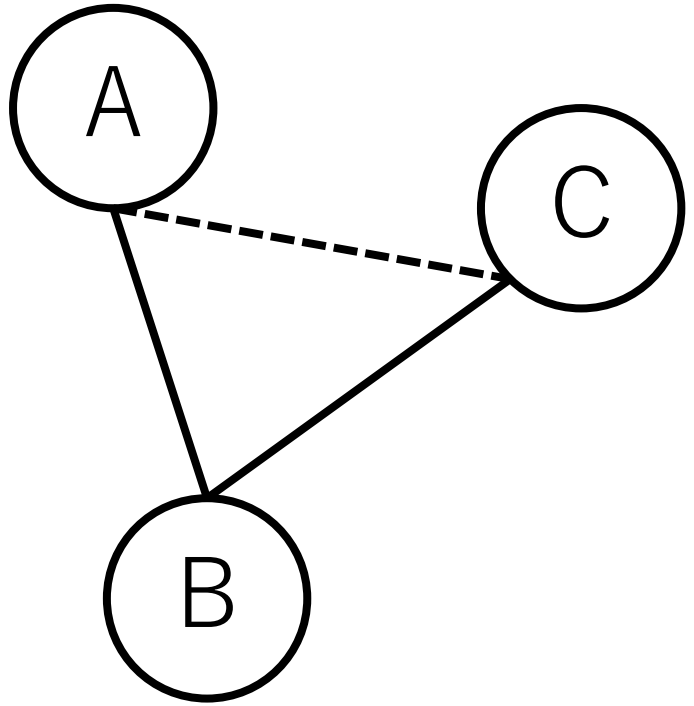
これはlogよりもさらに増加しない関数であり，定数倍とみなして扱われることもある.

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):
    ...
    # Unino-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ために追加で$O(|V|^2)$かかる．（$|V|$はノードの数）

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$O(|E|\log|E|)$.

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．）

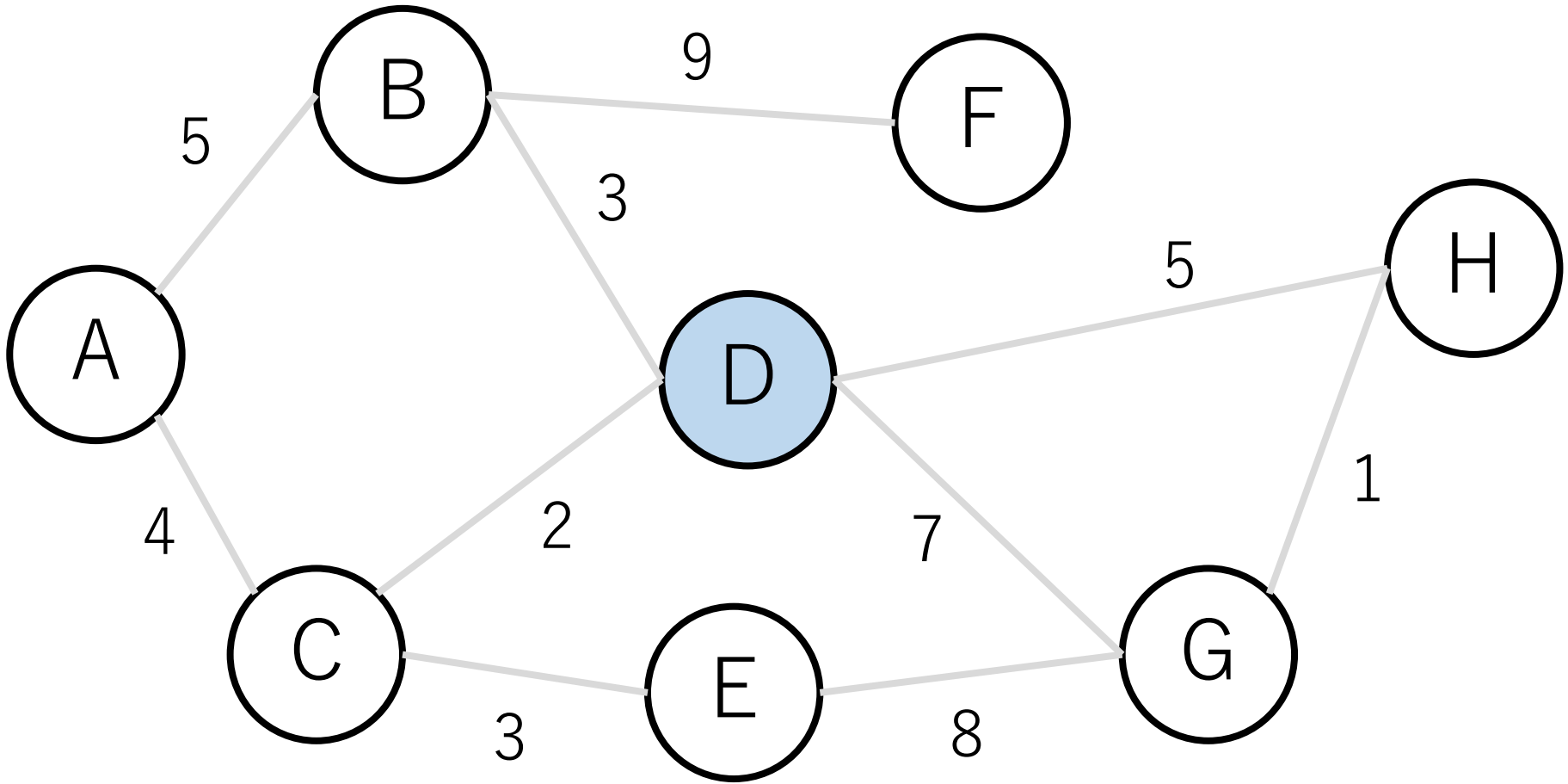
#4 選んだ辺を最小全域木に入れ，その接続先にあるノードを訪問済にする．

#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．

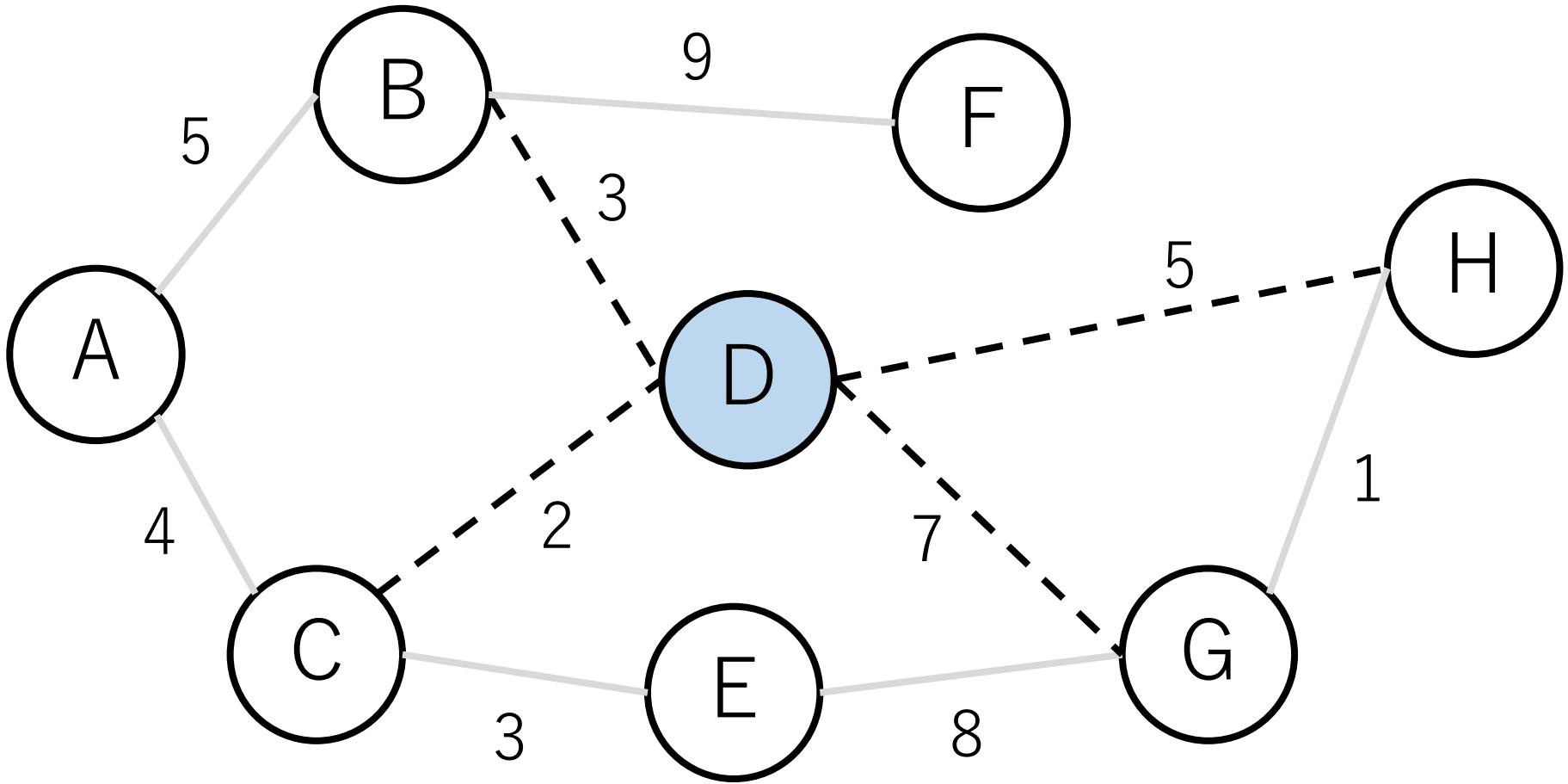
プリム法の例

Dからスタート．（どのノードから始めても良い）



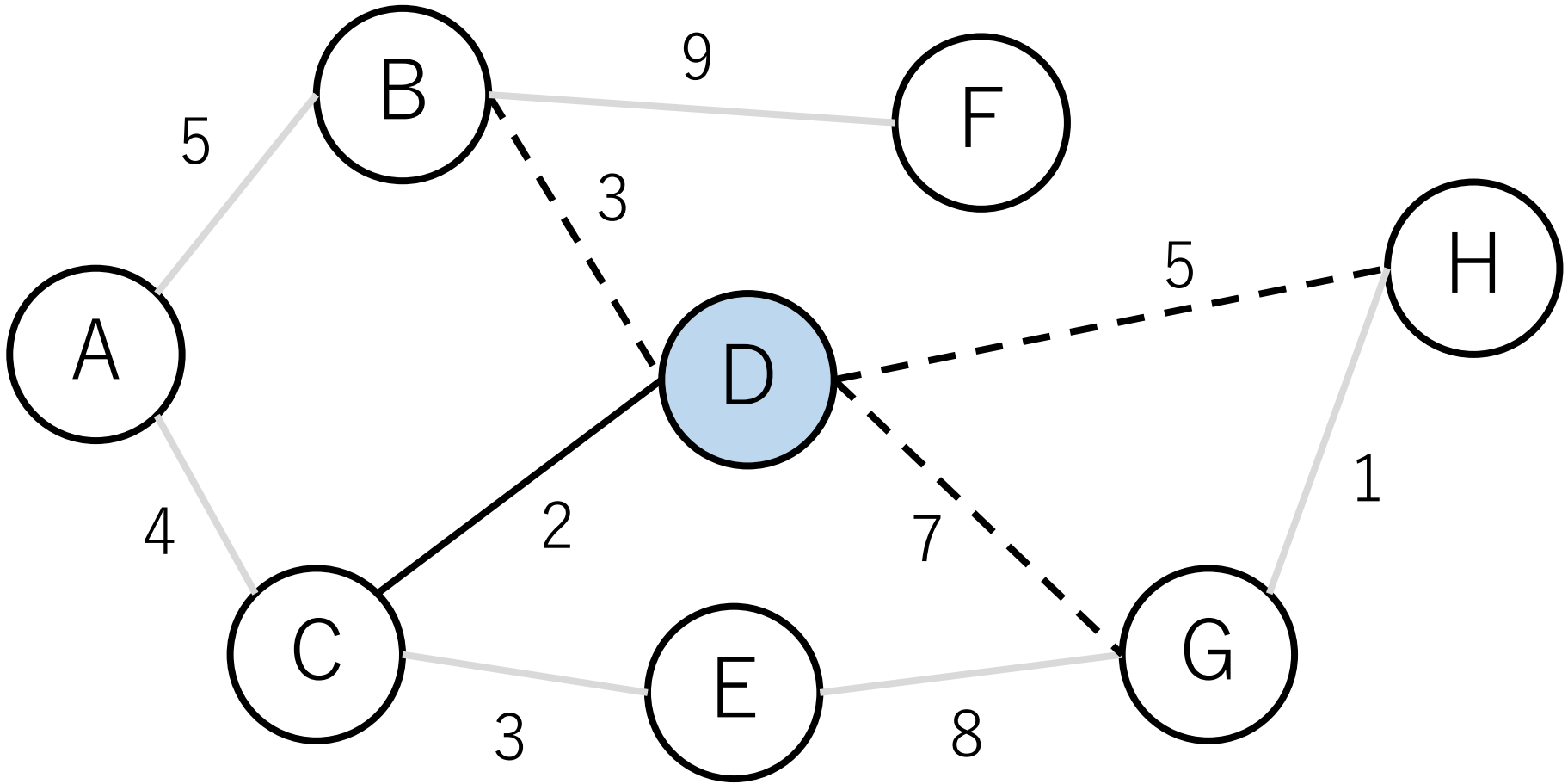
プリム法の例

最小全域木の候補の辺は点線で示すもの.



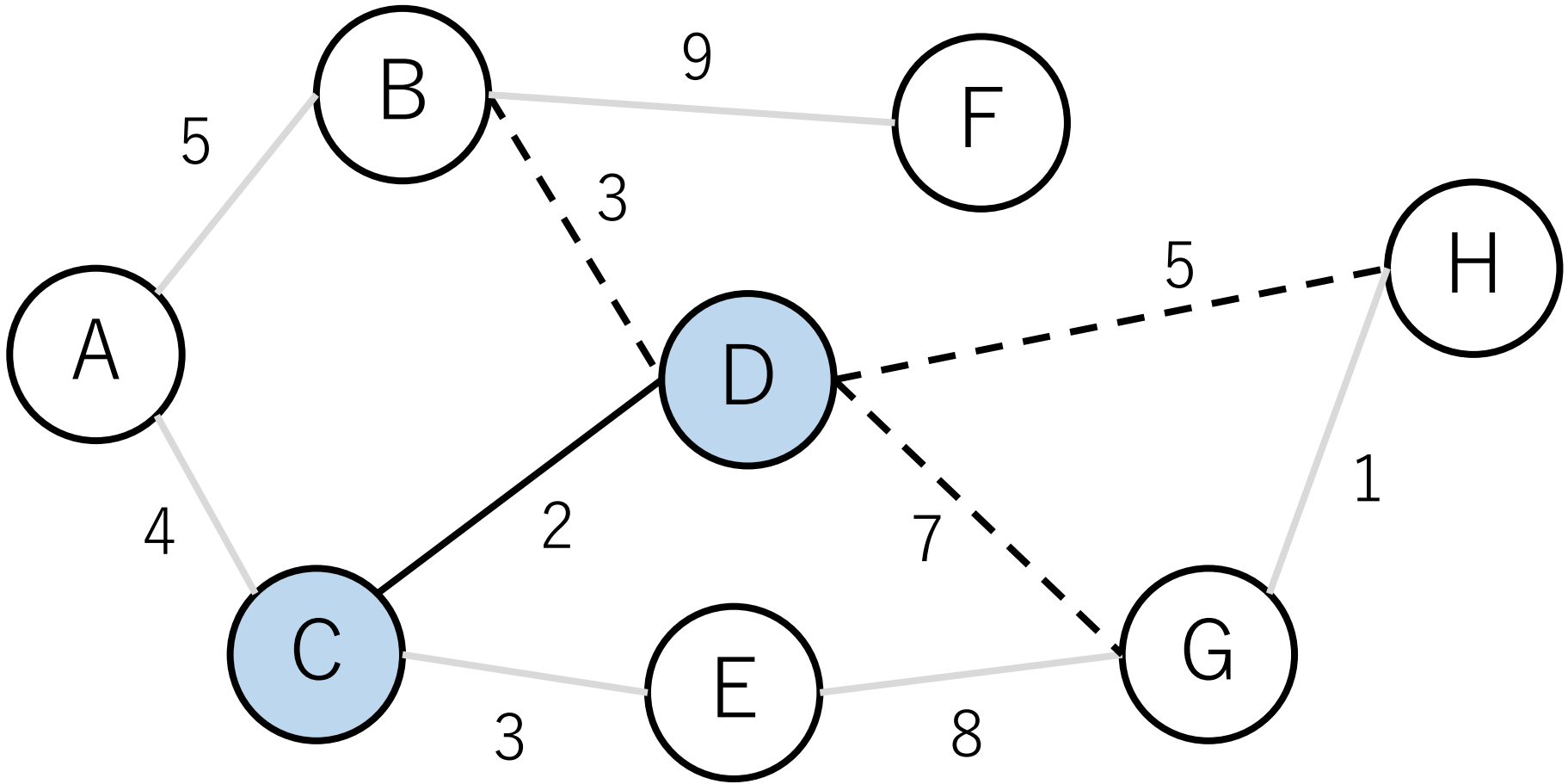
プリム法の例

C-Dの辺が最短なので，この辺を入れてCとつなぐ．



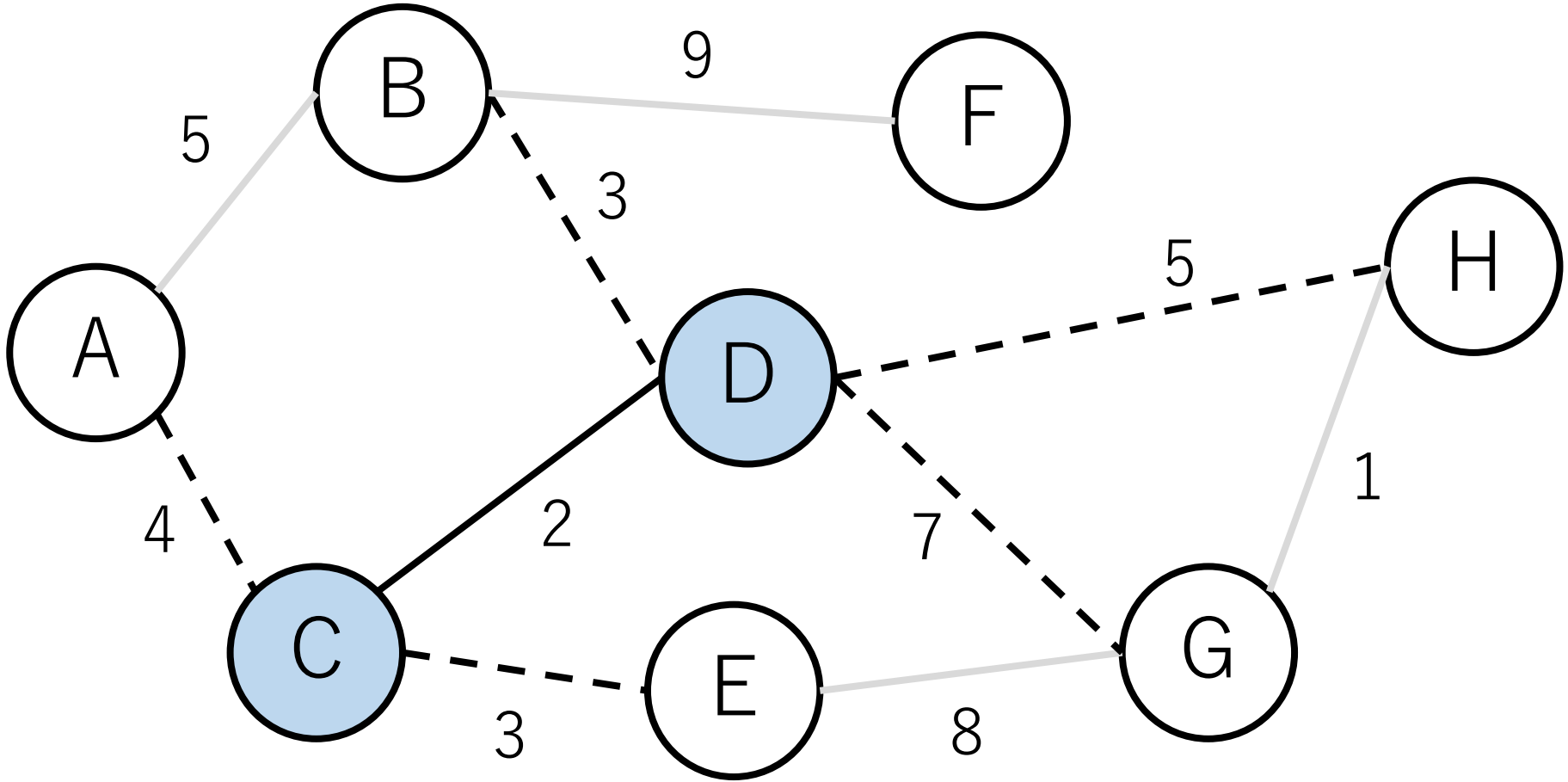
プリム法の例

CとDが「訪問済みグループ」になる.



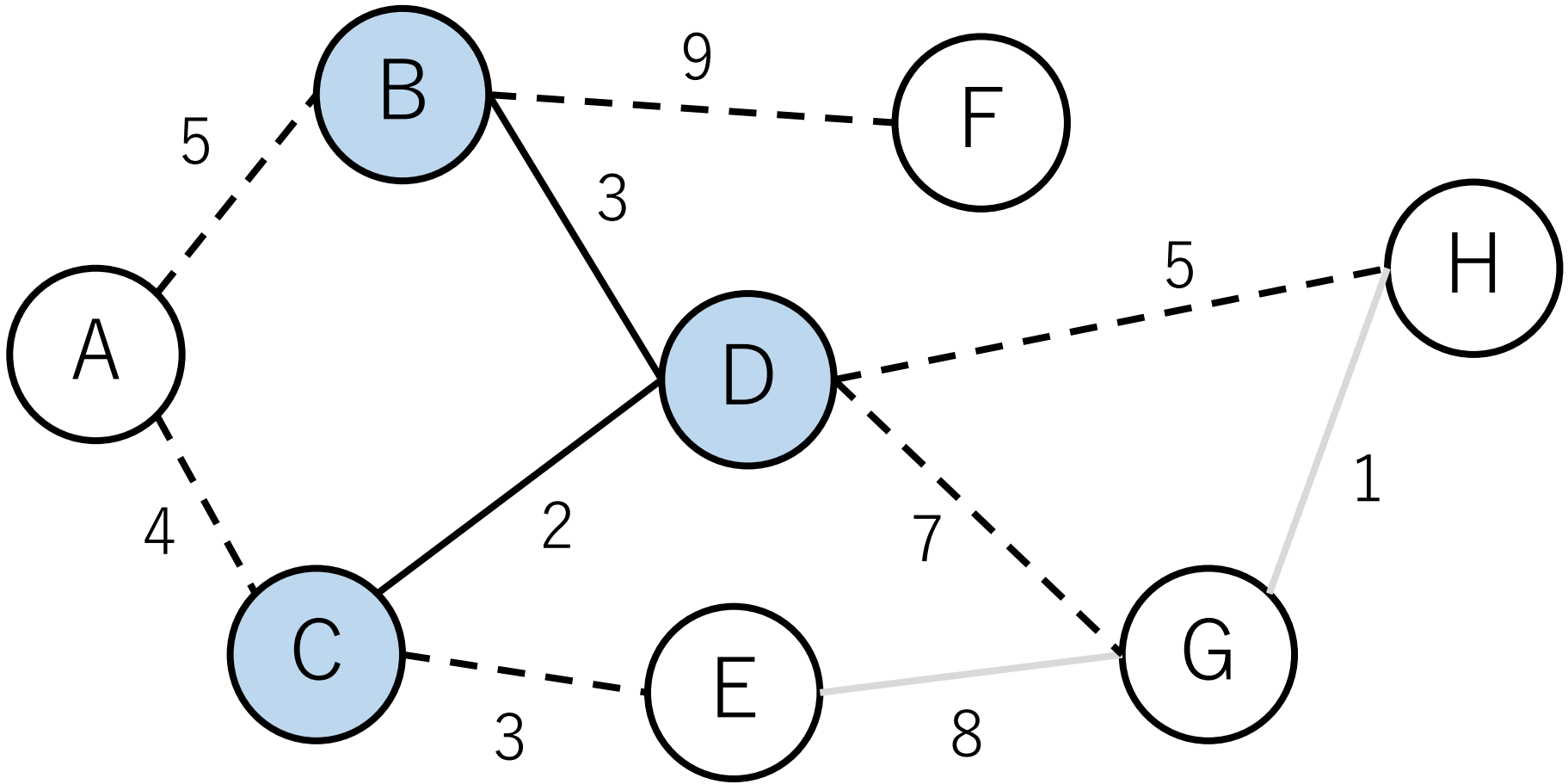
プリム法の例

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.



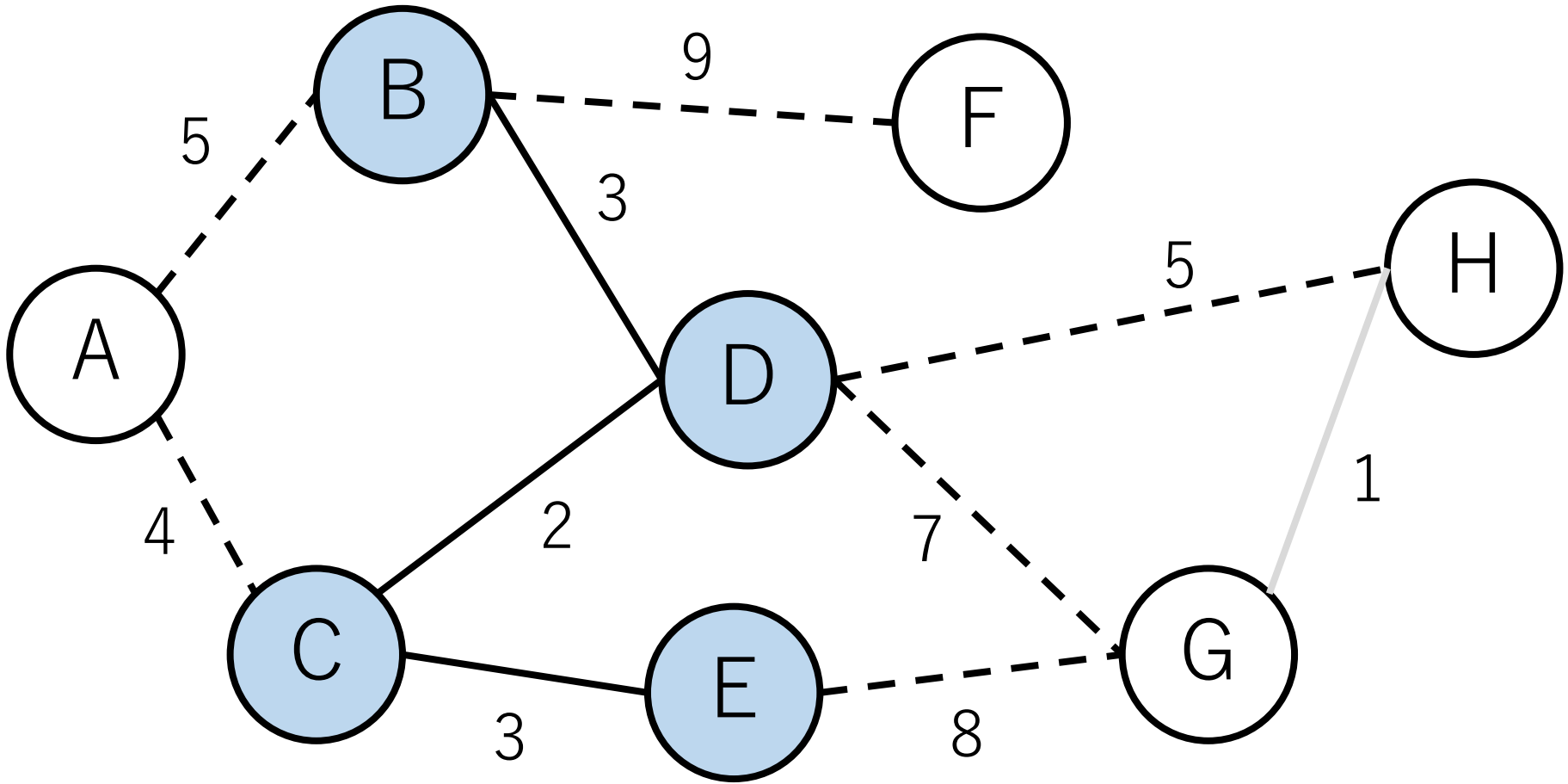
プリム法の例

最短距離は3（B->DとC->E）．ここではB-Dをつなぐ．



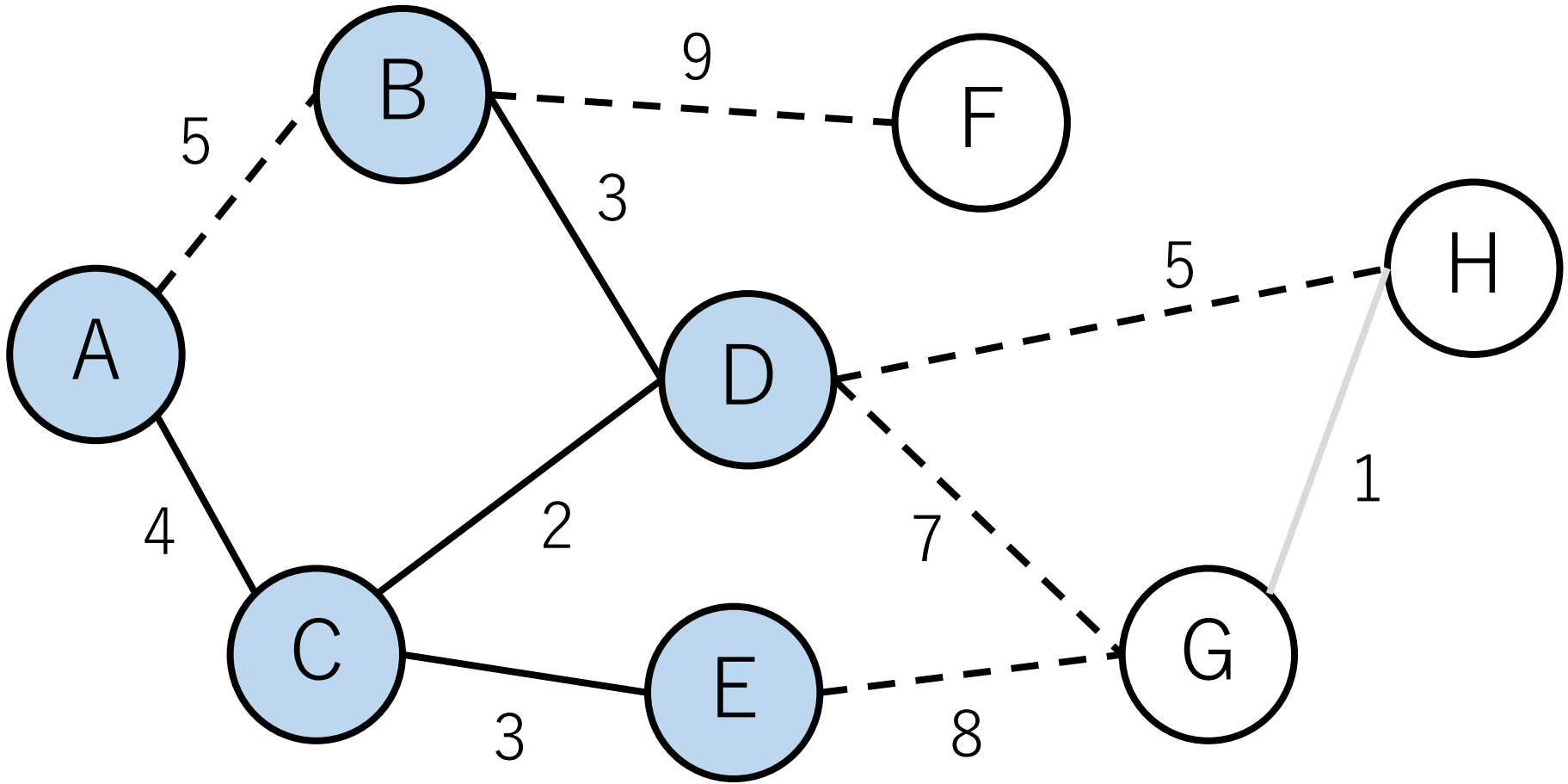
プリム法の例

次に最短距離のものはC-E.



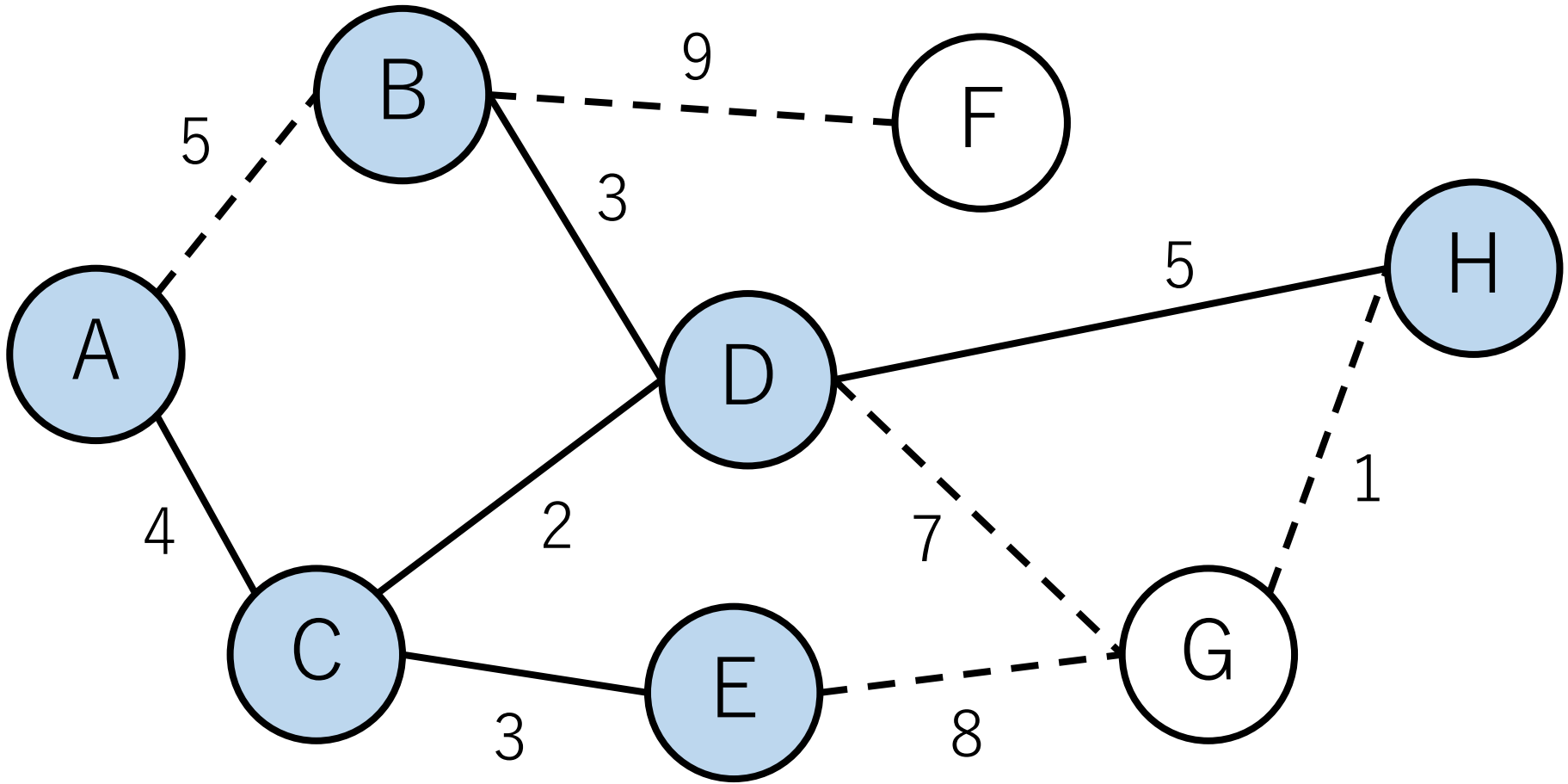
プリム法の例

その次は，A-C.



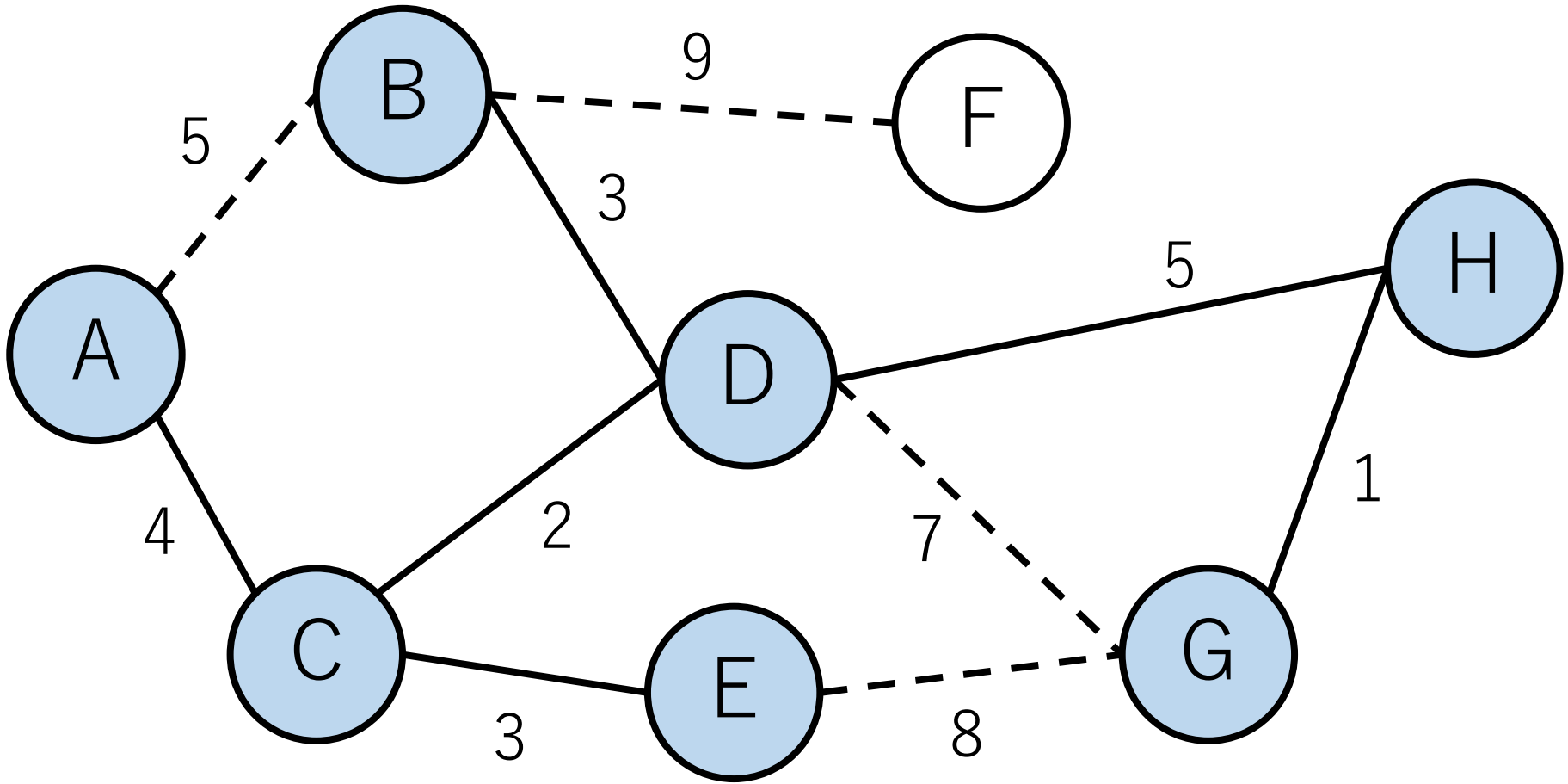
プリム法の例

その次は，D-H.



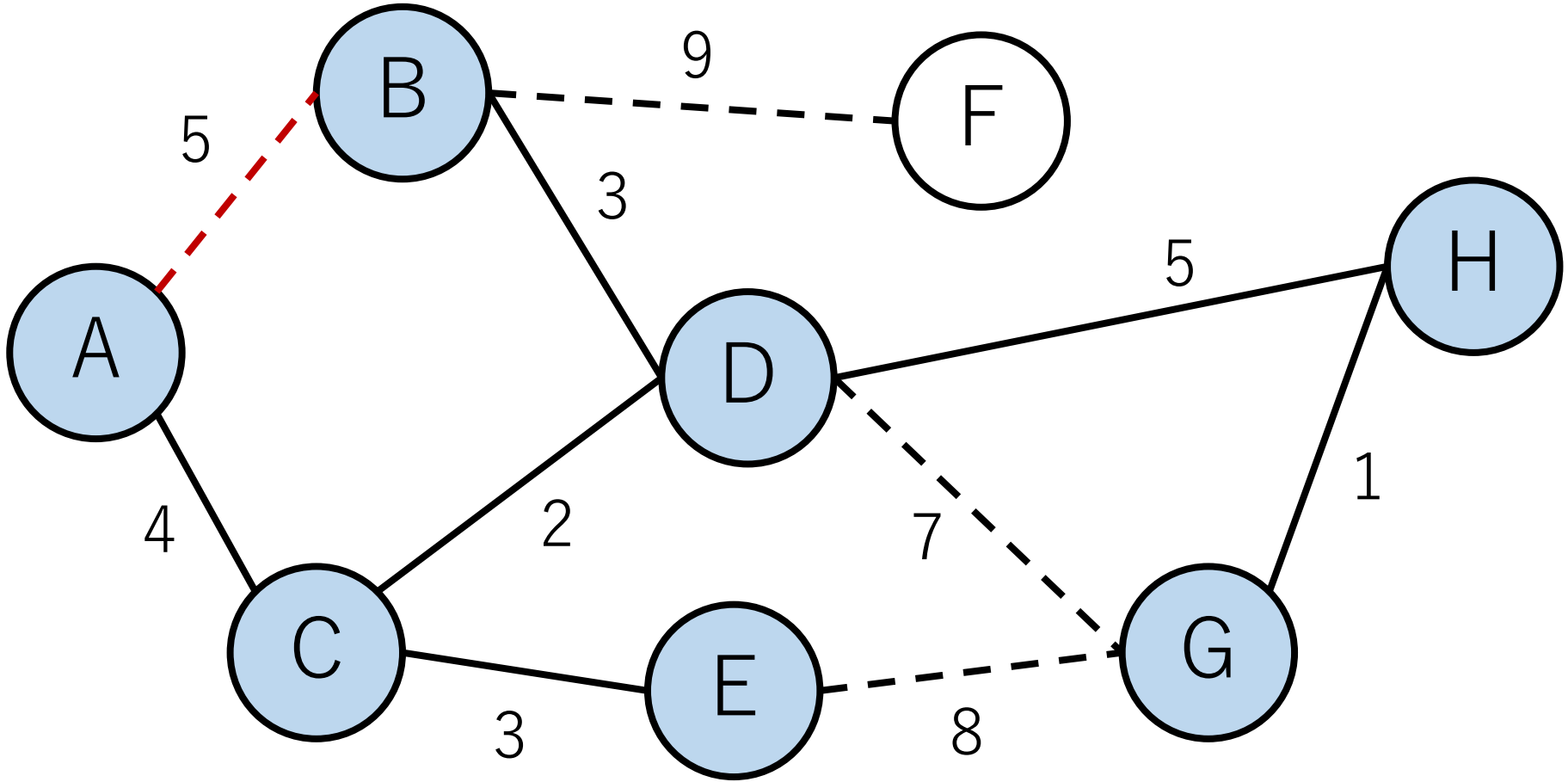
プリム法の例

その次は，G-H.



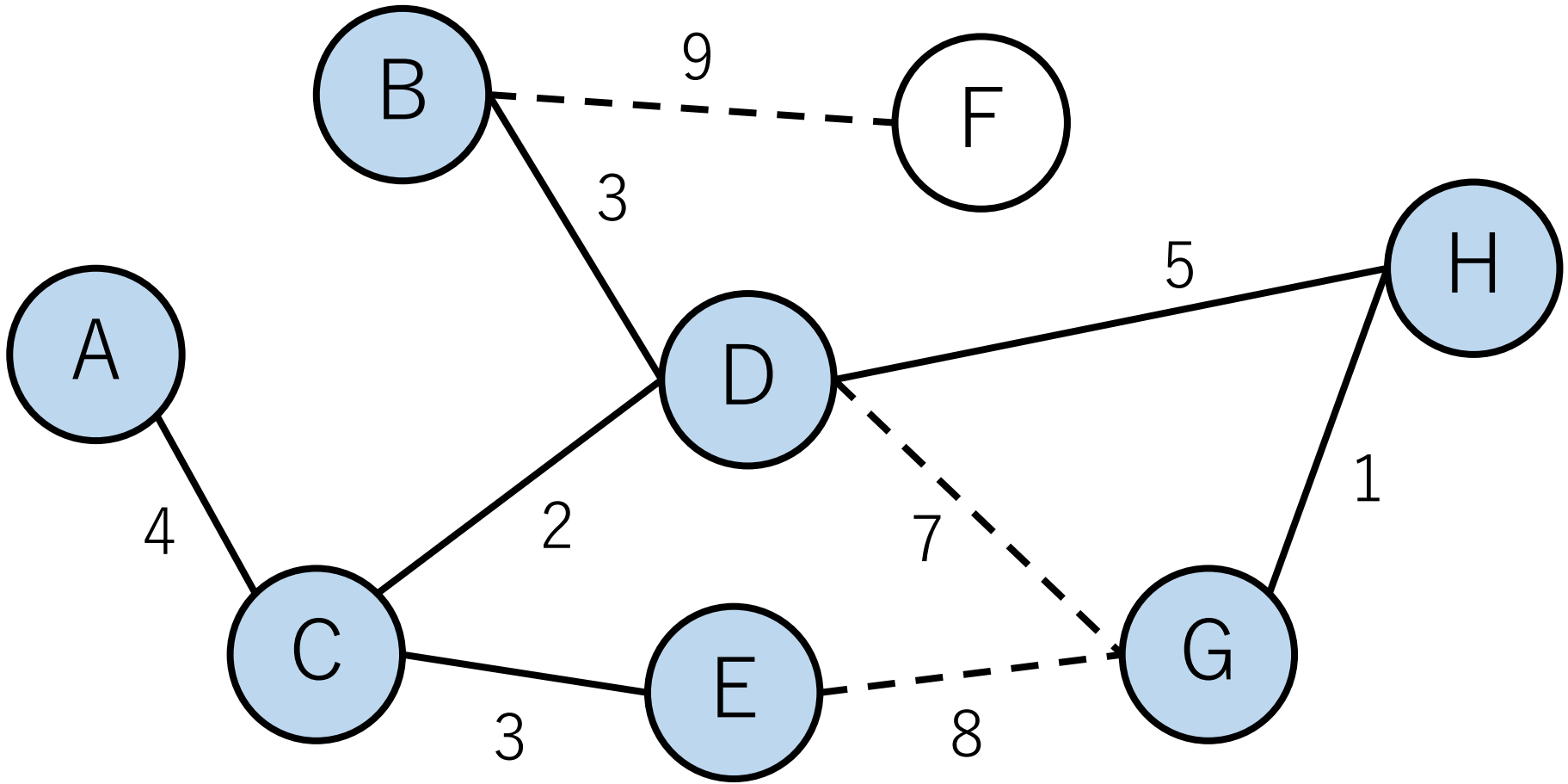
プリム法の例

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．



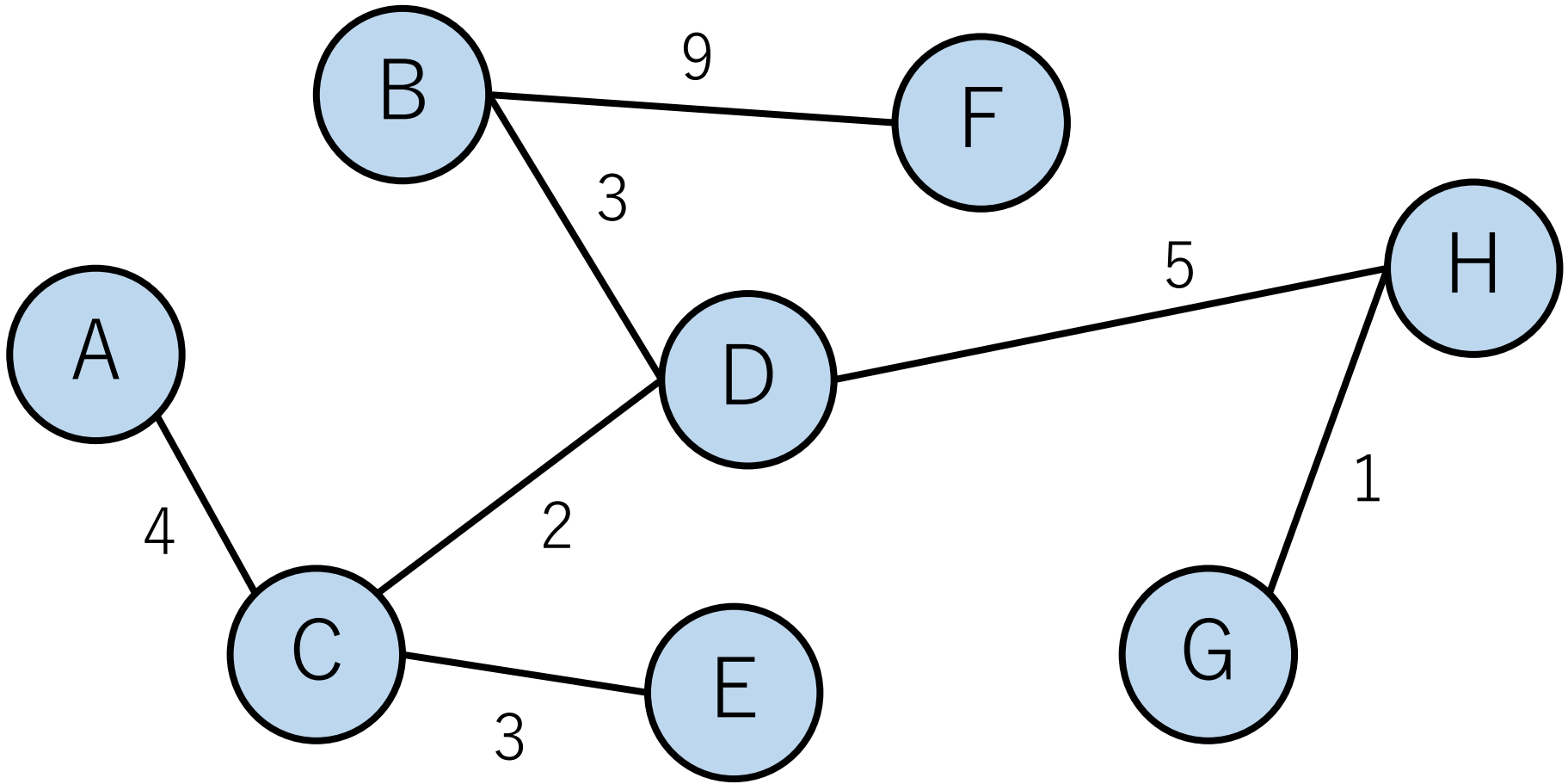
プリム法の例

D-G，E-Gについても同様にスキップ.



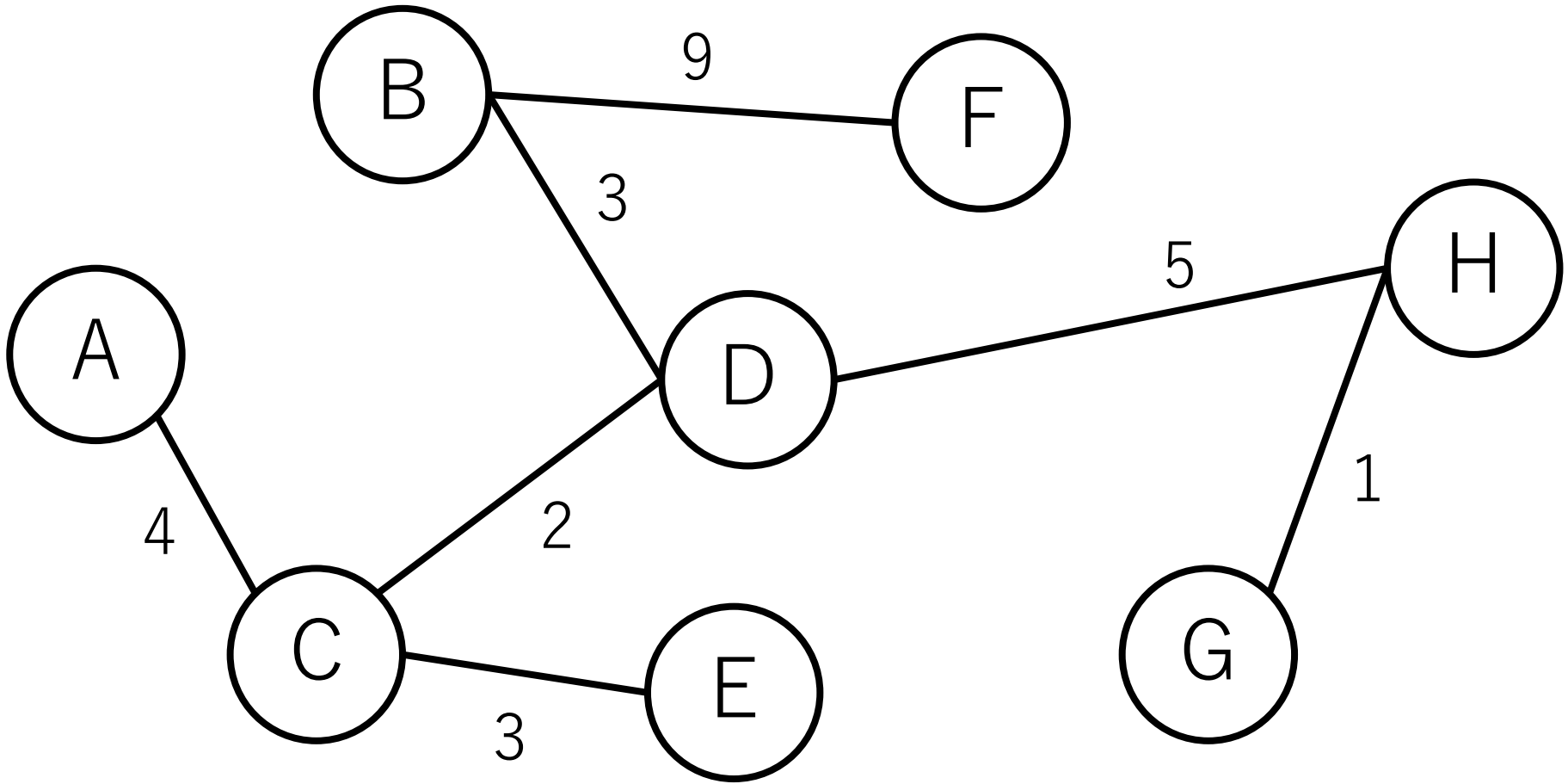
プリム法の例

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＋優先度付きキューを
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]
```

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):
        …


        # 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$O(|E|)$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．

コードチャレンジ：基本課題#12-a [1.5点]

スライドで説明したUnion-Find木を使って，
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ない
ので，注意してください．

コードチャレンジ：基本課題#12-b [1.5点]

スライドを参考にしてプリム法を実装してください.

ヒープを使う実装でないと通らない制約になっていますので，注意してください．この実装ではheapqを使っていただいて構いません.

コードチャレンジ：Extra課題#12

今回はExtra課題はありません.

レポート課題に向けて準備していただければと思います.

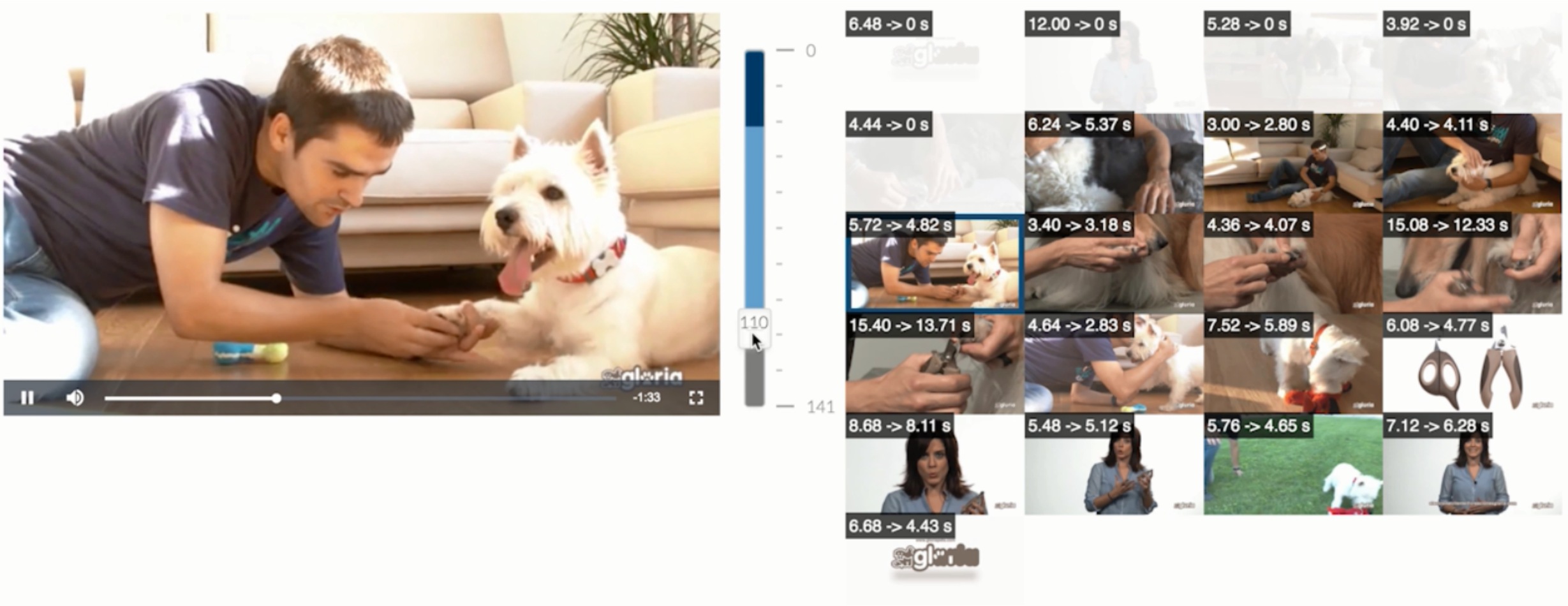
みなさま，最後までお付き合いいただき
誠にありがとうございました！

今までいろんな問題を解いてきました.

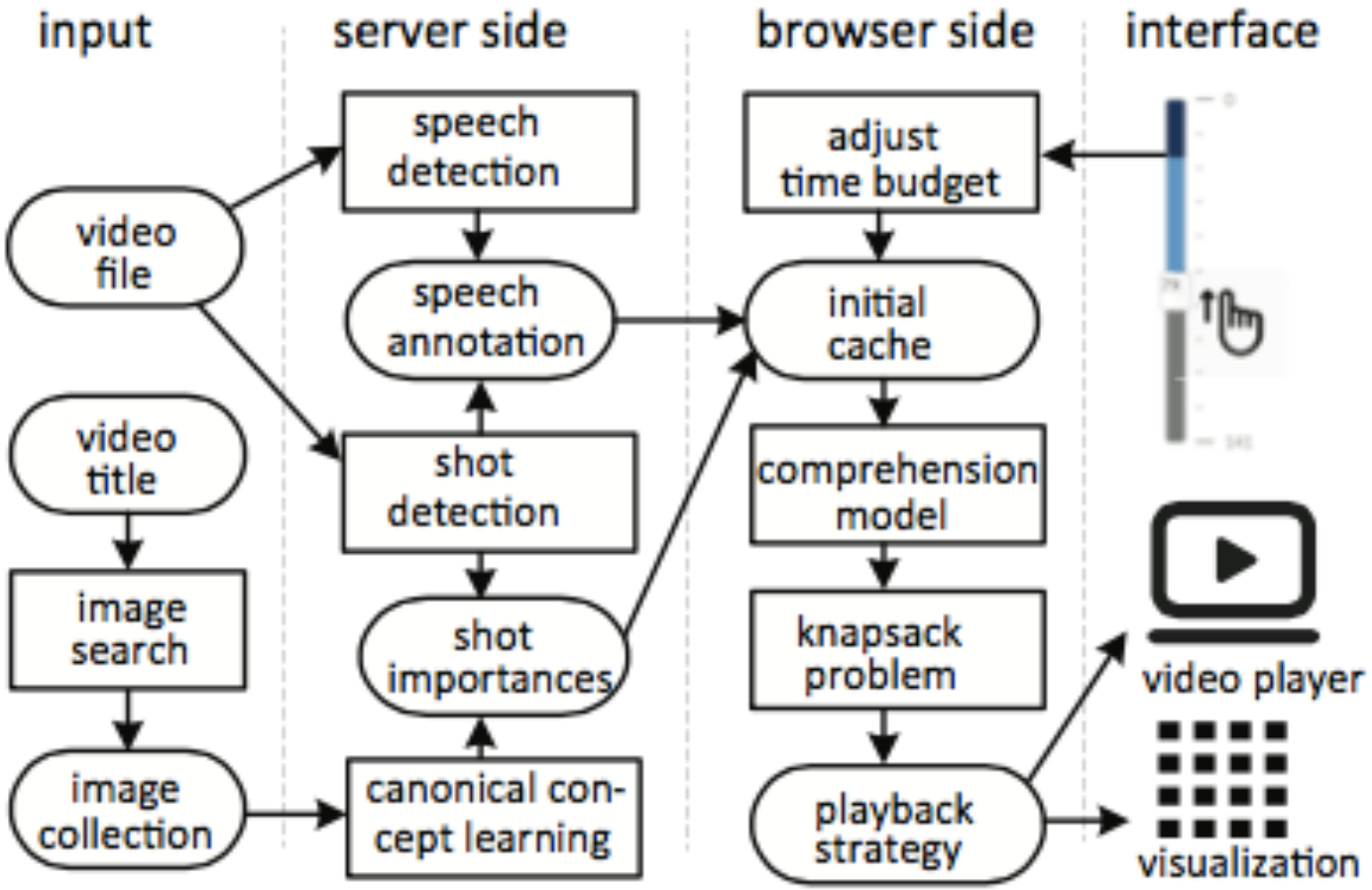
データ構造，累積和
探索
文字列探索
整列
DP
整数
BFS，DFS，橋の検出
グラフ（最短経路，最大流，最小費用流，二部グラフ，最小全域木）

何ができるの？

Haojian Jin, Yale Song, Koji Yatani. ElasticPlay: Interactive Video Summarization with Dynamic Time Budgets.
In Proceedings of ACM MM 2017 (oral presentation).

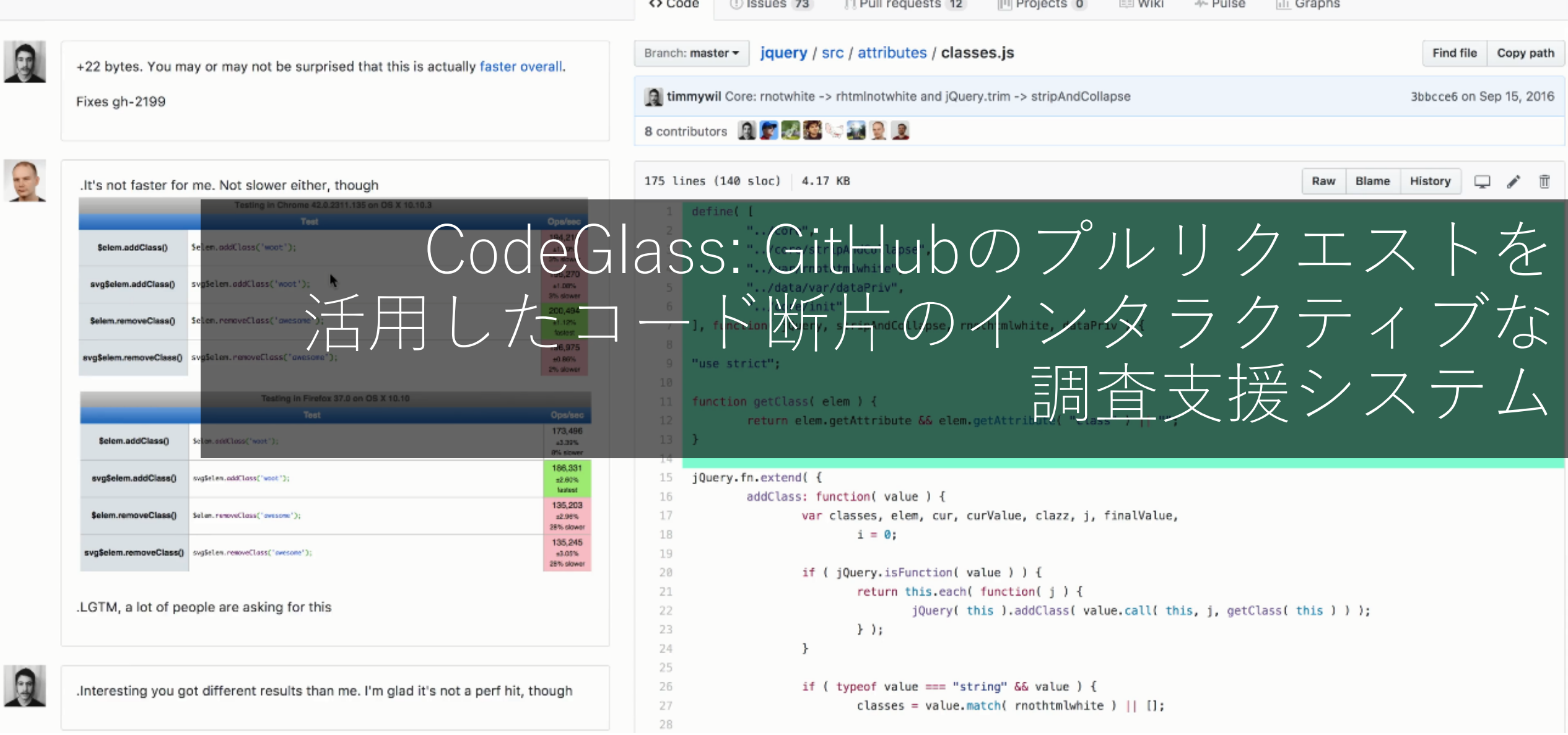


早送りと取捨選択を組み合わせたビデオの加速再生．ユーザは何秒でビデオを見終えたいか，を設定するだけで自動的に再生方針を決定する．



動画を音声がある部分とそうでない部分に分け，さらに重要度を推定し，重要度の低い部分は削除，また音声がない部分はより高速に早送りするようにして，ユーザが指定した再生時間に合わせた再生方法を自動的に生成.

CodeGlass: GitHubのプルリクエストを活用したコード断片のインタラクティブな調査支援システム

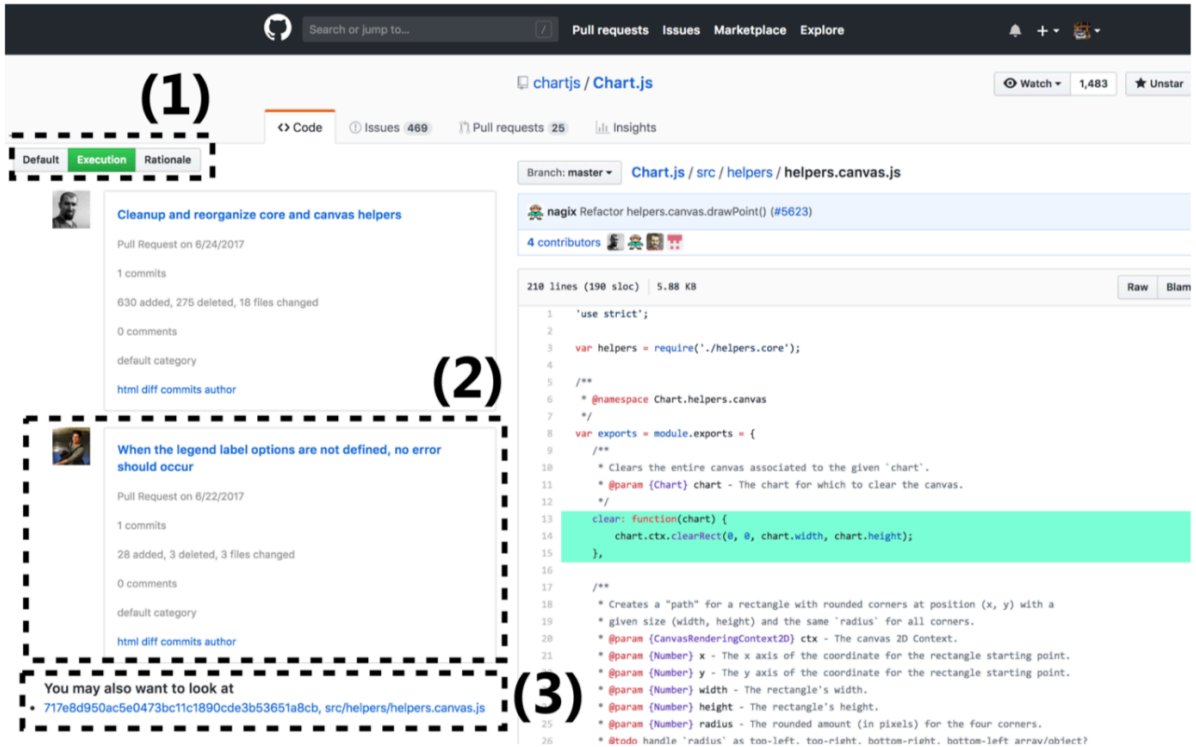


柴藤大介，有薗拓也，宮崎章太，矢谷浩司．CodeGlass: GitHubのプルリクエストを活用したコード断片のインタラクティブな調査支援システム．インタラクション2019（フルペーパー）．

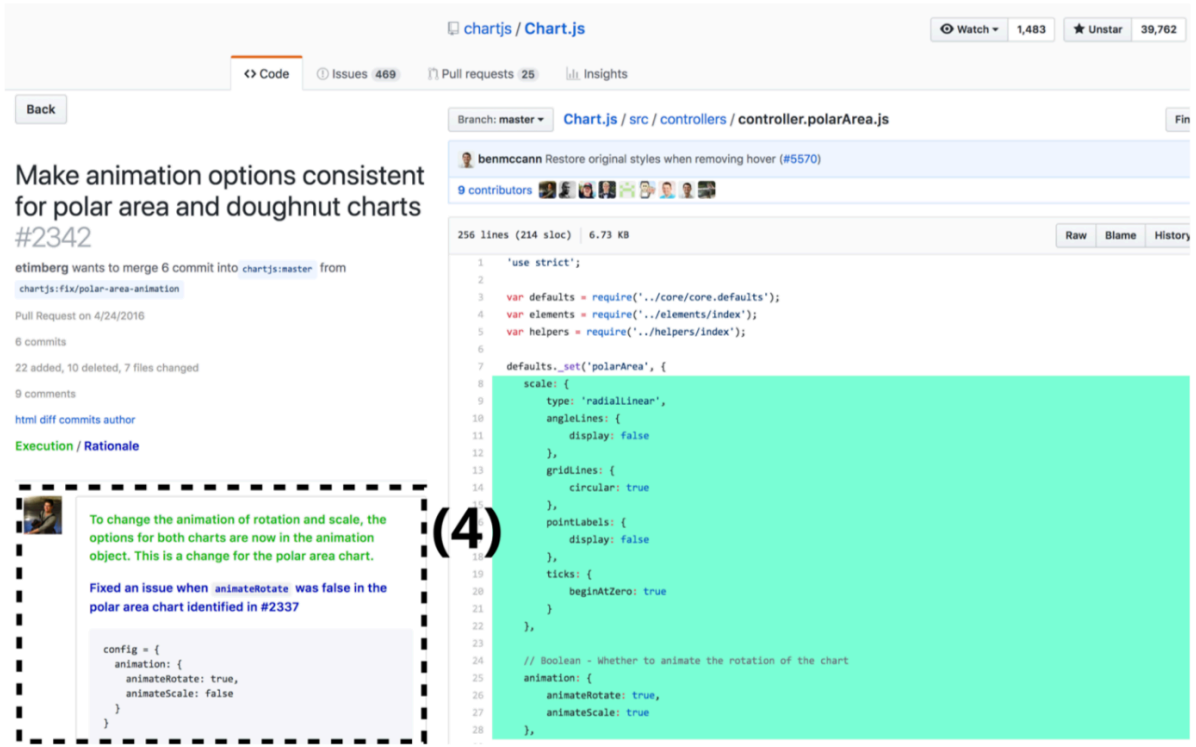
柴藤大介．「Interactive Piece-level Code Examination Using Diffs and Review Comments」東京大学工学部電子情報工学科卒業論文，2016年．優秀卒業論文賞受賞

CodeGlass: GitHubのプルリクエストを活用したコード断片のインタラクティブな調査支援システム

柴藤大介，有薗拓也，宮崎章太，矢谷浩司
IIS-Lab，東京大学



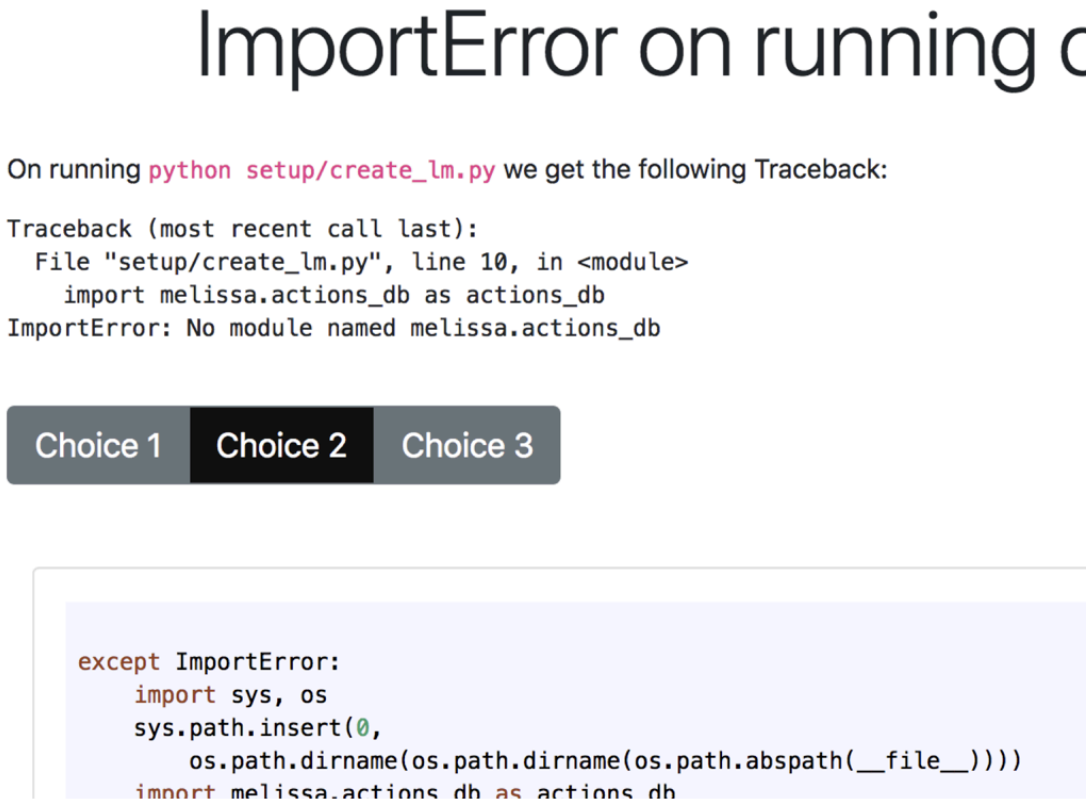
(a) 関連するプルリクエストの一覧画面.



(b) プルリクエストの詳細画面.

(1)時系列順，実装内容に関する情報が多い順，開発背景に関する情報が多い順に，関連するプルリクエストをソート可能.
(2)各プルリクエストの概要情報.
(3)選択されたコード断片と一致する可能性があるコード断片が過去のバージョンで複数ある場合には，そのバージョンにおけるソースコードへのリンクが表示.
(4)プルリクエストの詳細情報．実装内容に分類される文章は緑色で，開発背景に分類される文章は青色で表示される.

RealCode: GitHub上にある実コード変更をプログラミング課題に転用するシステム



柴藤大介，矢谷浩司．「GitHubのプルリクエストを用いたプログラミング課題自動生成システムの実現可能性に関する検討」第80回情報処理学会全国大会，2018年3月．学生奨励賞受賞

問題文: Issueの説明文

コードの解答例: 実際のコード変更

解答の説明文: Pull request内の説明文

存在するPull requestを利用して，実世界のコード変更をプログラミング課題として利用.

3種類のインタフェース（フリップカード，選択式，穴埋め式）で課題を提供．

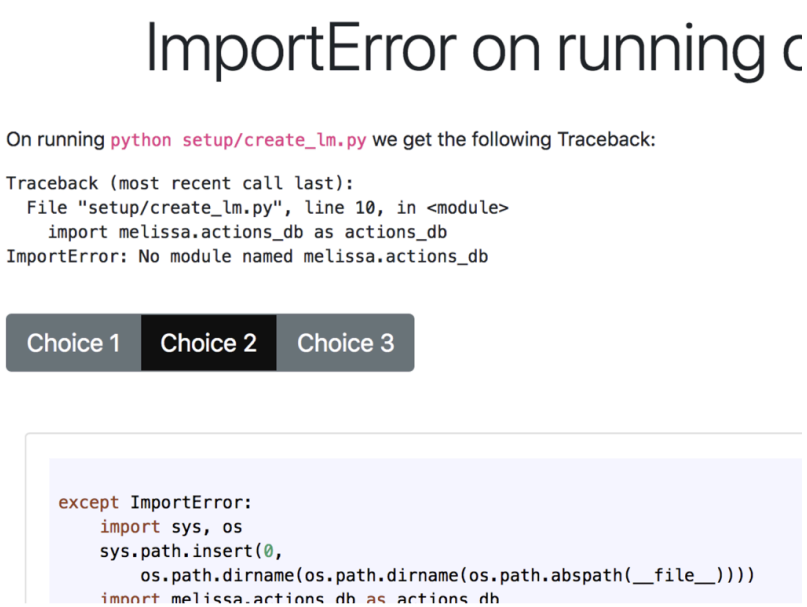
On running `python setup/create_lm.py` we get the following Traceback:

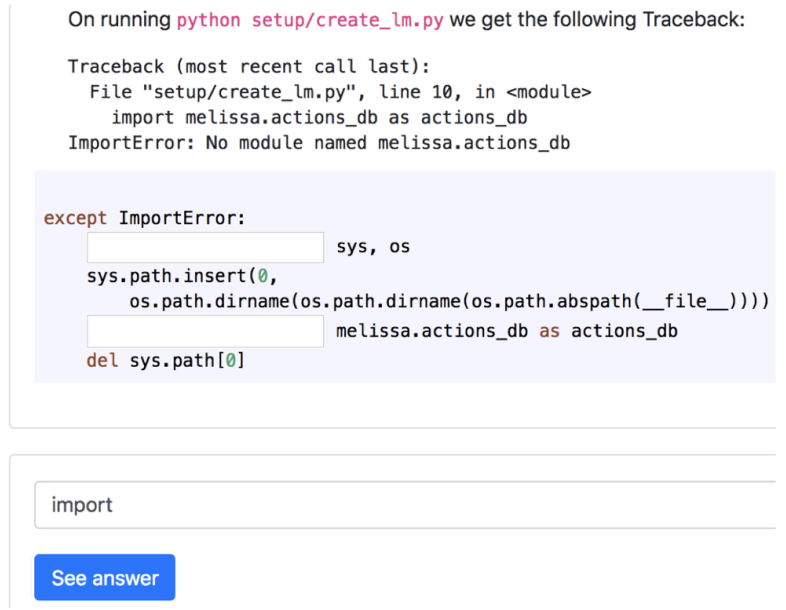
```
Traceback (most recent call last):
  File "setup/create_lm.py", line 10, in <module>
    import melissa.actions_db as actions_db
ImportError: No module named melissa.actions_db
```

Click to flip

```
except ImportError:
    import sys, os
    sys.path.insert(0,
        os.path.dirname(os.path.dirname(os.path.abspath(__file__)
    import melissa.actions_db as actions_db
    del sys.path[0]
```

Click to flip





実開発を体験できるプログラミング課題を提供できる可能性を確認.

「あんまりこういうエラーが発生してそれにどう対処すべきか、みたいな問題を経験したことがないから、なんか新鮮で面白そう。」

「開発の背景というか現状把握に時間がかかるけど、話にストーリーがあって面白い。（~ would be great, というセリフを見て）何かのサービスを改善したい、から問題が始まってるのは、学習のモチベーション的にいい。」

「コード内にコメントがたくさんあって、なんとなくこう可読性というか人に見られることを意識していていいね。」

アルゴリズムの知識は，計算理論に限らず，
信号処理，データ処理，ソフトウェア工学，
HCIなど，色んな場面でとっても役に立つ！

新しい応用先をぜひ一緒に考えましょう！😊

色んな所で腕試ししてみよう！

競技プログラミング

与えられた課題に対して制限時間内に設定された要件を満たすようなコードを書き，提出する．

AtCoder: https://atcoder.jp/

AOJ: http://judge.u-aizu.ac.jp/onlinejudge/

ACM ICPC: https://icpc.iisf.or.jp/

Google Code Jam: https://codingcompetitions.withgoogle.com/codejam

などなど．

競技プログラミング

形式的にはこの講義でやったExtra課題のようなもの.

Pythonに限らずいろんなプログラミング言語で参加可能.

国内のコンテストは日本語でも参加可能.

TAさん曰く,

「この授業を通してアルゴリズムに惹かれてatcoderにも挑戦しているのですが、この授業を一通り受講してある程度練習したら目安としてどのくらいのレベルまで狙えるとかありますかね…？」

→AtCoderであればここでの知識のみであっても水色中盤あたりにはいくのではないかなと思っています

→水色(上位15%)くらい狙えるのかなと思います。Extra課題の難易度が平均的にそれくらい(上位15%が解けるかどうか、くらい)になってると思います。

ぜひチャレンジしてみてください！

この講義で勉強したことを踏まえれば解くことのできる問題はそれなりにあるはず.

ただ，問題文の内容をうまく読み替えてどのアルゴリズムを使うべきかを判定するところは慣れが必要かも.

パズル的要素も多いですが，考えたものをコードに落とし込むエクササイズとしては，とても良いと思います.

国際的なコンテストにもぜひ！

日本からの参加が少ないこともしばしば．．．

英語の壁はあるかもですが，実力でのしあがれる世界でもあるので，ぜひ参加してみてください．

1つお願いしたいことがあります.

授業のslackを開いてください.

授業のアンケートにご協力ください！

今年度初めてやったことがいっぱいありますので，ぜひ皆さんのご意見をお聞かせください．

https://forms.gle/cWaE7m6Gb87sm38R9

来年度のTAさんを募集しますので，我こそはという方はぜひ！😊

slackでDM，もしくはメールを下さい．

来週もあります！ぜひ参加登録を（成績には関係しません）．

https://sites.google.com/g.ecc.u-tokyo.ac.jp/facebook-cyci2020

次に皆さんとお会いするのは

後期実験「情報可視化とデータ解析」になります！
https://iis-lab.org/infovis

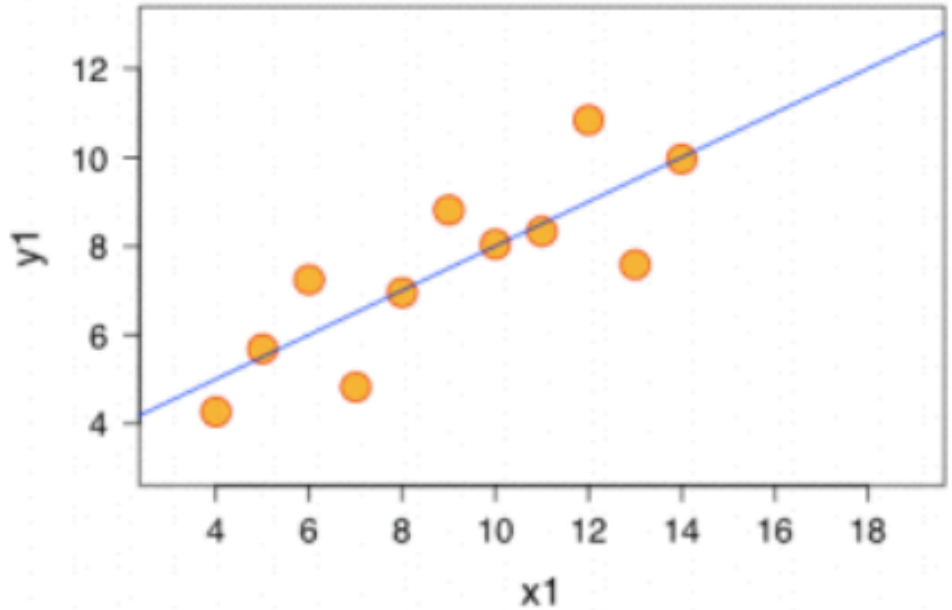
何が違うでしょうか？

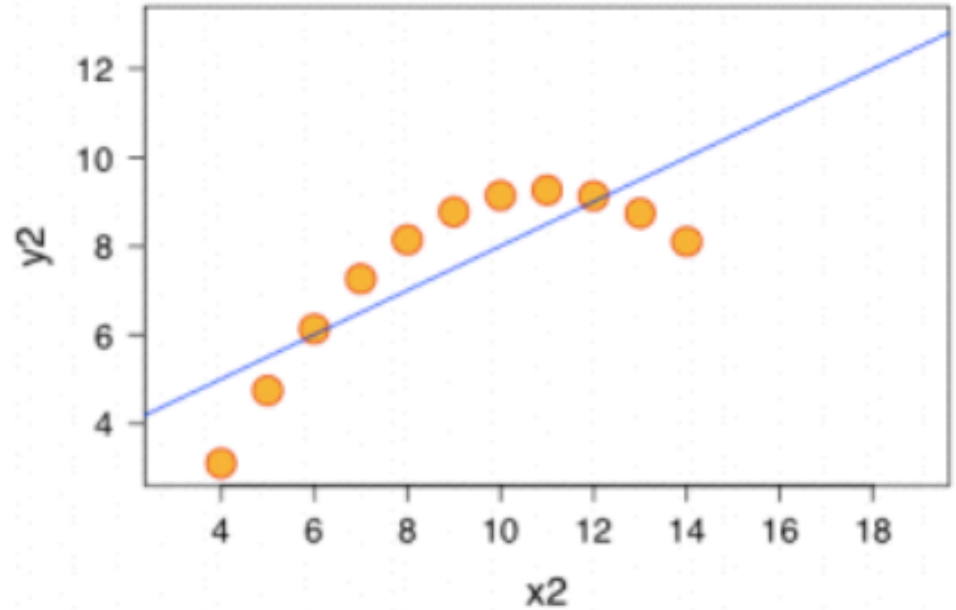
Anscombe's Quartet: Raw Data

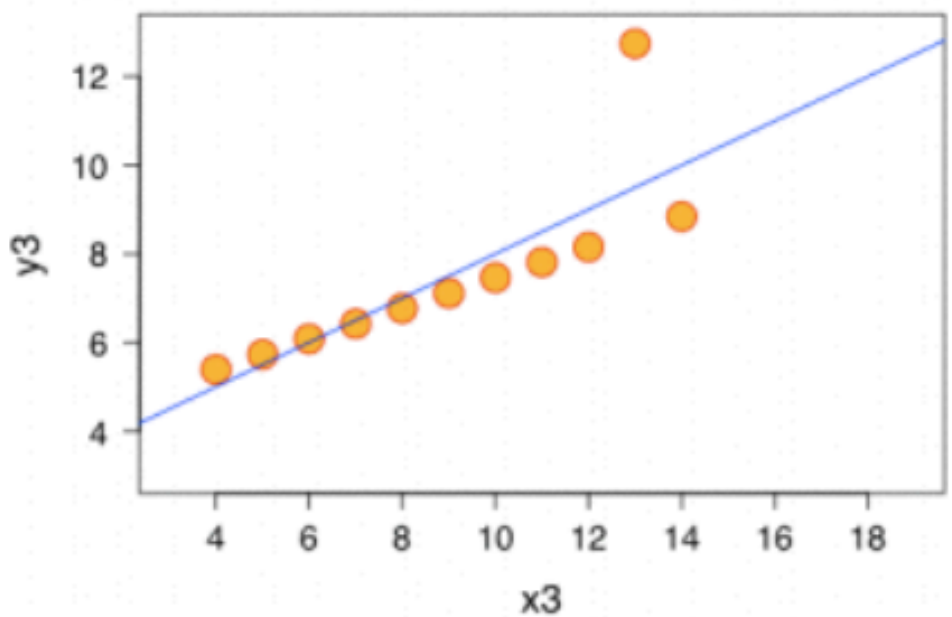
	I		II		III		IV	
	x	y	x	y	x	y	x	y
	10.0	8.04	10.0	9.14	10.0	7.46	8.0	6.58
	8.0	6.95	8.0	8.14	8.0	6.77	8.0	5.76
	13.0	7.58	13.0	8.74	13.0	12.74	8.0	7.71
	9.0	8.81	9.0	8.77	9.0	7.11	8.0	8.84
	11.0	8.33	11.0	9.26	11.0	7.81	8.0	8.47
	14.0	9.96	14.0	8.10	14.0	8.84	8.0	7.04
	6.0	7.24	6.0	6.13	6.0	6.08	8.0	5.25
	4.0	4.26	4.0	3.10	4.0	5.39	19.0	12.50
	12.0	10.84	12.0	9.13	12.0	8.15	8.0	5.56
	7.0	4.82	7.0	7.26	7.0	6.42	8.0	7.91
	5.0	5.68	5.0	4.74	5.0	5.73	8.0	6.89
mean	9.0	7.5	9.0	7.5	9.0	7.5	9.0	7.5
var.	10.0	3.75	10.0	3.75	10.0	3.75	10.0	3.75
corr.	0.816		0.816		0.816		0.816	

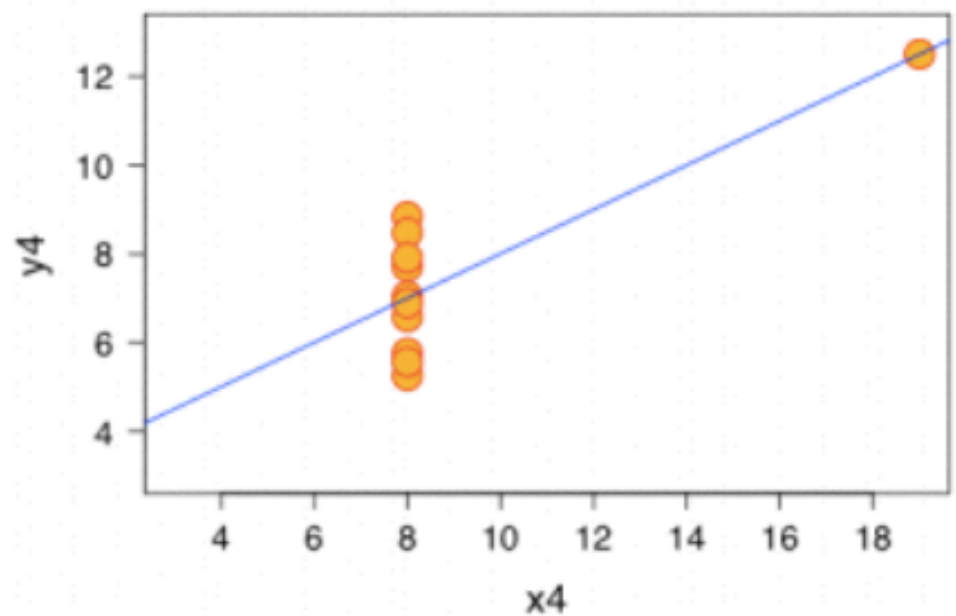
[Anscombe 1973]

😄


y1
x1
4
6
8
10
12
14
16
18


y2
x2
4
6
8
10
12
14
16
18


y3
x3
4
6
8
10
12
14
16
18


y4
x4
4
6
8
10
12
14
16
18

次に皆さんとお会いするのは

後期実験「情報可視化とデータ解析」になります！
https://iis-lab.org/infovis

ユーザ（人間）がデータを見て分析したり，物事の判断をしたりするためにどのように可視化を設計するかを学んでいきたいと思います．

また，現実世界のデータを使って皆さんなりの可視化システムを作っていただきます．😄