ICPC練習会
〜全域木いろいろ〜

北海道大学大学院　情報科学研究科
博士1年　井上 祐馬

内容

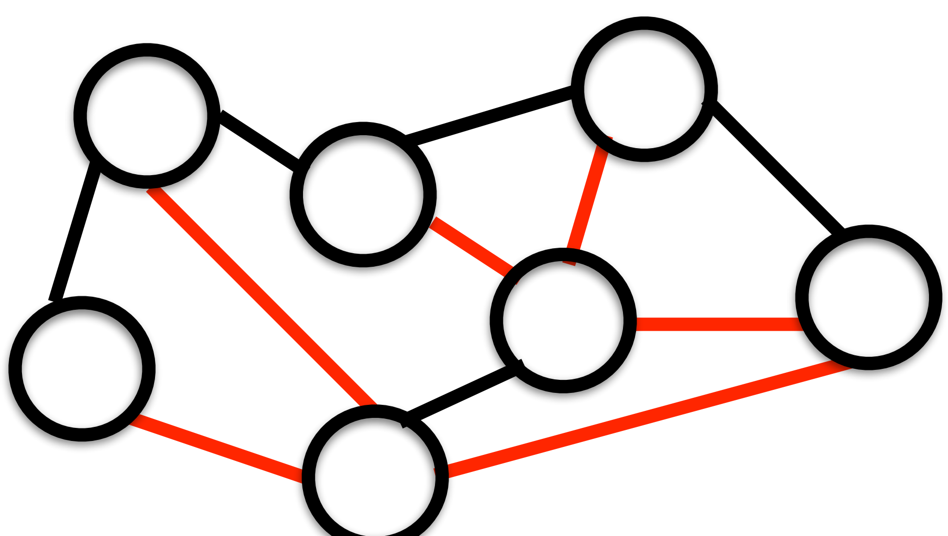
- 全域木の定義
- 最小全域木を求めるアルゴリズム
  - prim法
  - kruskal法
- 最小全域木アルゴリズムの応用
- 行列木定理

蟻本を読もう！

～ 完 ～

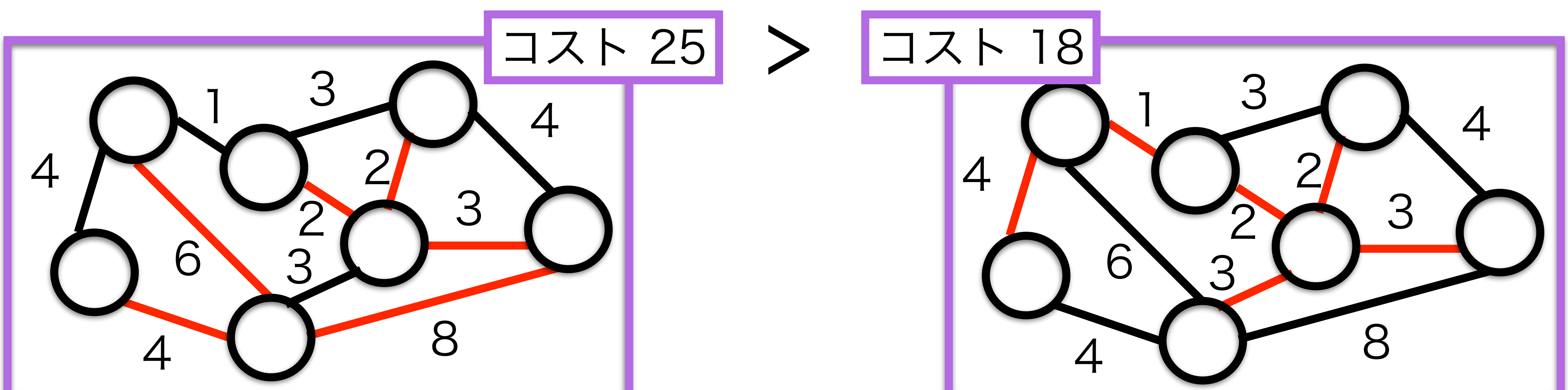
全域木 (Spanning Tree) とは

- (連結な) 無向グラフ G = (V, E) において，以下の条件を満たす部分グラフ T = (V', E') を全域木と呼ぶ
  1. V = V'
  2. T は木 (連結 かつ サイクルを含まない)
- 非連結を許容する場合は全域森という



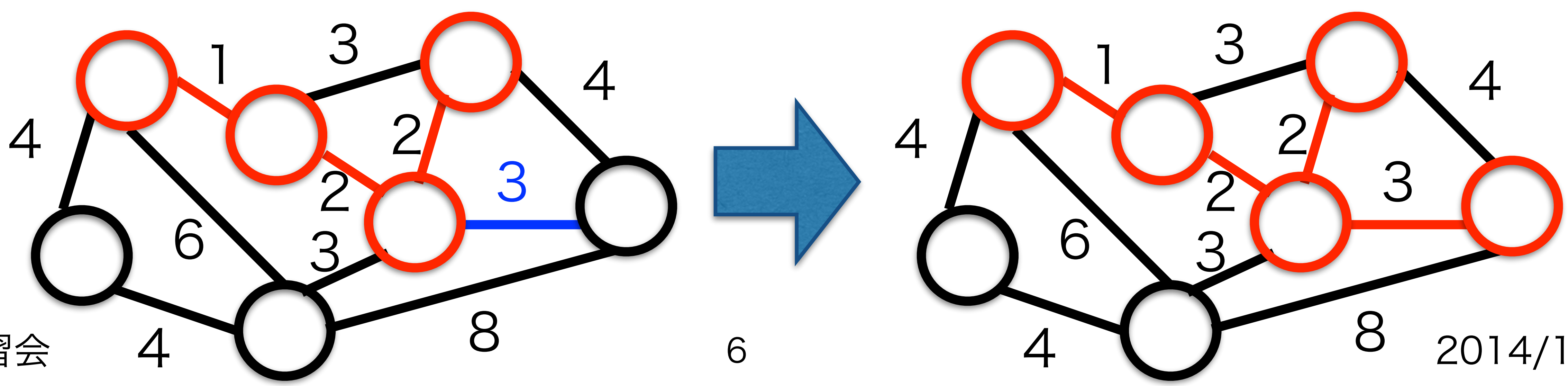
最小全域木とは

- G が重み付きグラフである場合，全ての全域木の中で辺の重みの総和が最小のものを最小全域木と呼ぶ
- 1つの最小全域木は O(|E| log|V|) で求められる
  - プリム法
  - クラスカル法



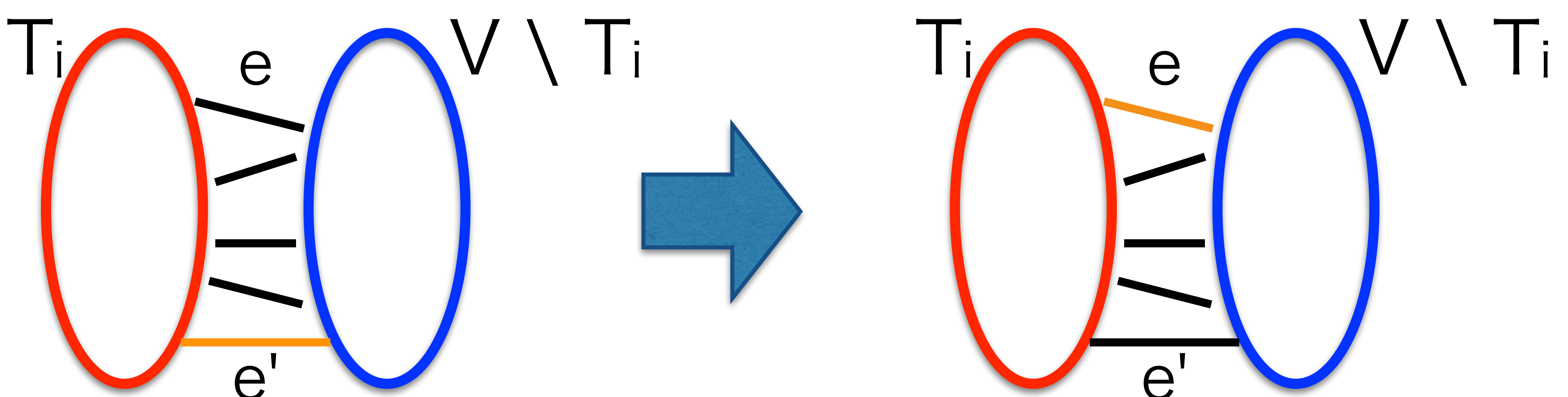
プリム (Prim) 法

- i 個の頂点を持つ部分全域木 T_i から i+1 個の頂点を持つ部分全域木 T_{i+1} を構築
- i+1 個目の頂点を選ぶ戦略は貪欲法
  - $T_i = (V_i, E_i)$ とする
  - T_i と $V \setminus T_i$ を結ぶ辺の中で最小コストの辺を追加し、$V \setminus T_i$ 側の端点を追加



プリム法の正当性

- T_i と $V \setminus T_i$ を結ぶ最小コストの辺をeとする
- eを含まない最小全域木 T' が存在すると仮定する
  - T' には T_i と $V \setminus T_i$ を結ぶ e 以外の辺 e' が必ず存在
  - e' を e に置き換えても全域木
  - e のコストは e' 以下 → e に置き換えても最小全域木



プリム法の実装

- ダイクストラ法と同様に実装可能
  - 「まだ訪れてない頂点への最短距離」を「まだ加えてない頂点への最小辺」に変えるだけ
  - 開始地点は任意
  - 毎回全頂点への最小辺を調べると$O(|V|^2)$
  - 最小辺を二分ヒープで管理することで $O(|E| \log|V|)$

実装例 (O(V²))

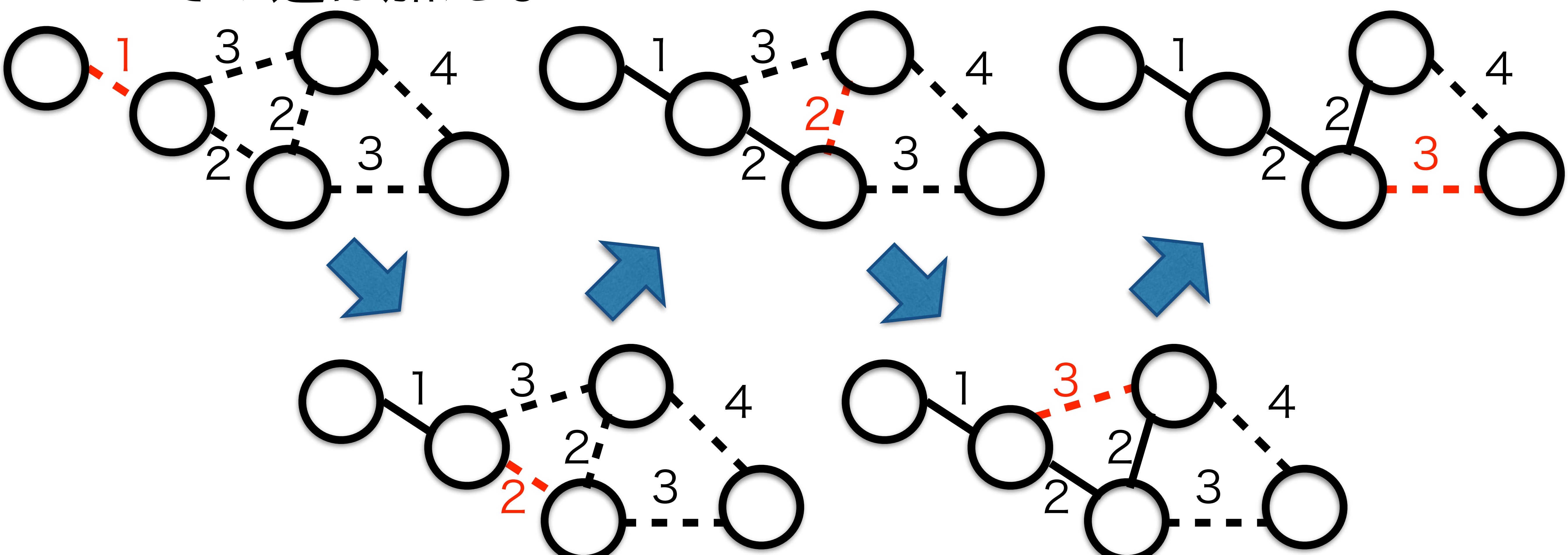
```
int MinSpanningTree(){
   int res = 0;
   vector<int> d(n,INF); d[0] = 0;
   vector<bool> use(n,0);

   for(;;){
     int v = -1;
     for(int u=0;u<n;u++){
       if(!use[u] && (v<0 || d[v] > d[u]))v = u;
     }
     if(v<0)break;
     use[v] = 1; res += d[v];

     for(int i=0;i<(int)g[v].size();i++){
       d[g[v][i].to] = min( d[g[v][i].to], g[v][i].cost);
     }
   }
   return res;
 }
```

実装例 (O(E logV))

```
typedef pair<int,int> pii;

int MinSpanningTree(){
  int res = 0;
  vector<int> d(n,INF); d[0] = 0;
  priority_queue<pii, vector<pii>, greater<pii> > q;
  q.push(pii(0,0));

  while(q.size()){
    int cost = q.top().first, v = q.top().second; q.pop();
    if(d[v] < cost)continue;
    res += cost; d[v] = -INF;

    for(int i=0;i<(int)g[v].size();i++){
      int to = g[v][i].to, nxtcost = g[v][i].cost;
      if(d[to] > nxtcost){
        d[to] = nxtcost;
        q.push( pii(nxtcost,to) );
      }
    }
  }
  return res;
}
```

クラスカル (Kruskal) 法

- コストの小さい辺から順に追加していく貪欲法
- ただし，その辺を加えるとサイクルができる場合，その辺は加えない



クラスカル法の正当性

- プリム法と基本的に同様
- 辺 e = (u, v) を追加する前の状態が最小全域木の部分グラフであるとする
  - e を加えてサイクルができない場合:
    - eを加えずに u の連結成分 と v の連結成分を結ぶ辺 e' が以降に存在すれば全域木となるが，e' のコストは e 以上
  - e を加えてサイクルができる場合:
    - すでに u, v は連結であり，できるサイクル中のどの辺もコストが e 以下 → サイクルを e で切断するのがコスト最小

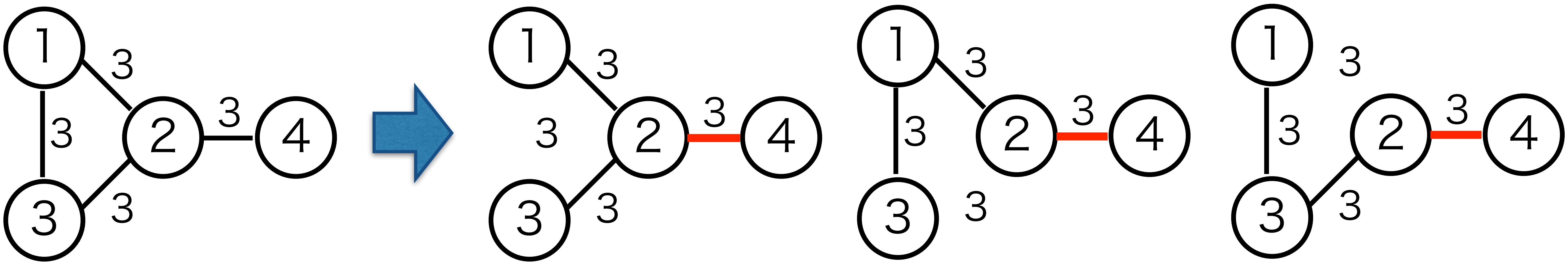
クラスカル法の実装

- まず辺をソート: O(|E| log|V|)
- 辺追加を |E| 回行うたびに、サイクルになるか判定する
  → 連結な頂点集合を Union-Find 木で管理: O(α(|V|))
  - 頂点を結んだら集合を併合
  - すでに連結な頂点同士を結ぶとサイクルができる
  - 全体で O(α(|V|) |E|) (ほぼ O(|E|))
- ソートは最初の1回のみなので、辺の削除クエリ付きの場合でも1回あたりはO(|E|)で済む

実装例

```
struct edge{
  int from, to, cost;
  edge(int x,int y, int z):from(x),to(y),cost(z){}
  bool operator<(const edge &x)const{ return cost>x.cost; }
  //逆！！！ (for priority_queue)
};

int MinSpanningTree(){
  UnionFind connect(n);
  priority_queue<edge> q;
  for(int v=0;v<n;v++){
    for(int i=0;i<(int)g[v].size();i++)q.push(g[v][i]);
  }

  int res = 0;
  while(q.size()){
    edge e = q.top(); q.pop();
    if(!connect.same(e.from,e.to)){
      res += e.cost;
      connect.unite(e.from,e.to);
    }
  }
  return res;
}
```

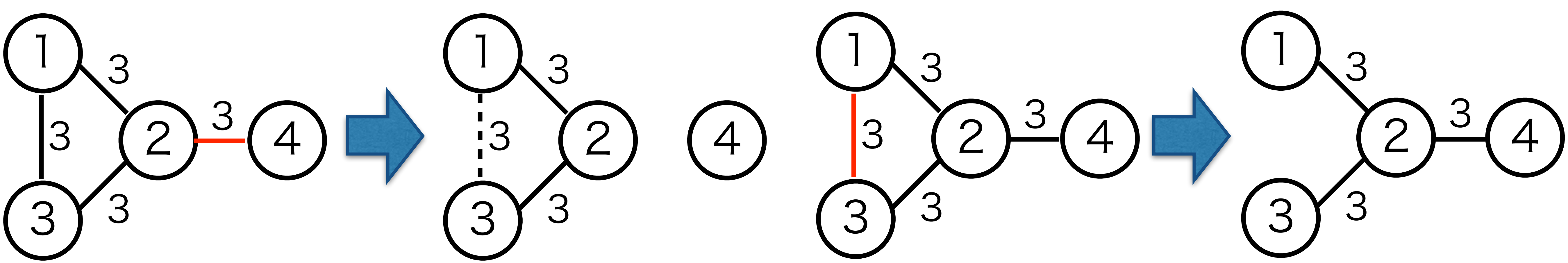
問題例: ICPC tokyo 2014 F "There is No Alternative"



- 重み付き無向グラフ G = (V, E) が与えられる
- G の最小全域木に必ず含まれる辺は何本あるか？重みの総和とともに答えよ
- 制約: $3 \le |V| \le 500$, $1 \le |E| \le 50{,}000$

解法

- まず最小全域木を1つ求め，そのコスト C を記録
- すべての辺 e について，e を取り除いたグラフの最小全域木を求め，そのコスト C' と C を比較
  - 全域木が作れなければ e は必ず最小全域木に必要
  - C < C' ならば，e は必ず最小全域木に必要



解法

- まず最小全域木を1つ求め，そのコスト C を記録
- すべての辺 e について，e を取り除いたグラフの最

較

木に必要

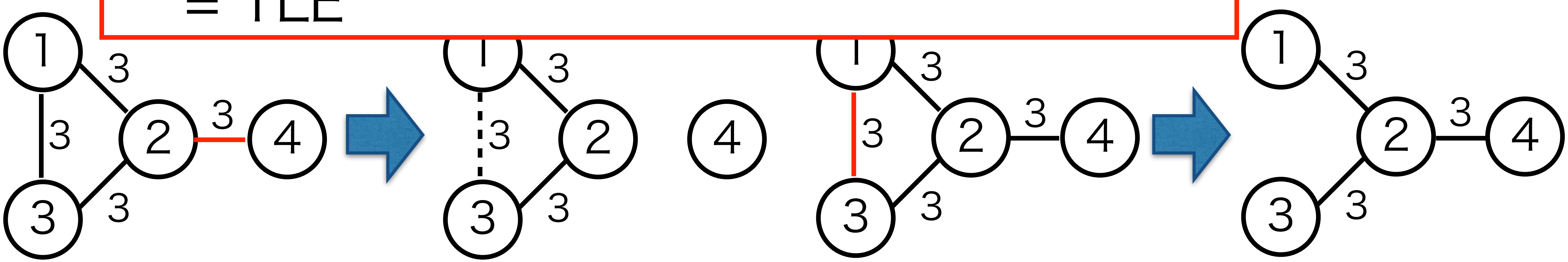
要

クラスカル法で

・前処理ソート O(|E| log|V|)

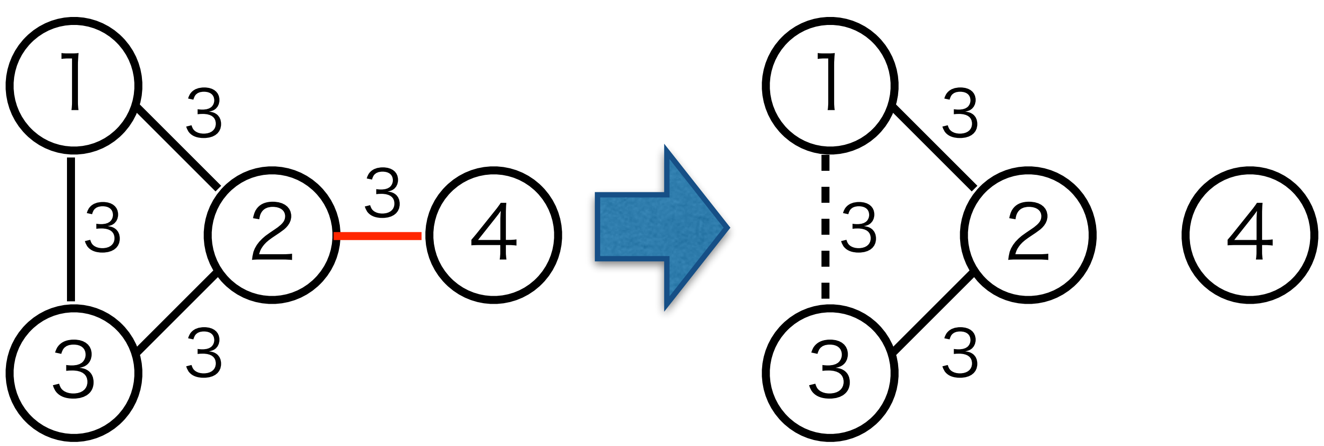
・各辺を無視した最小全域木を計算 $O(|E|^2)$

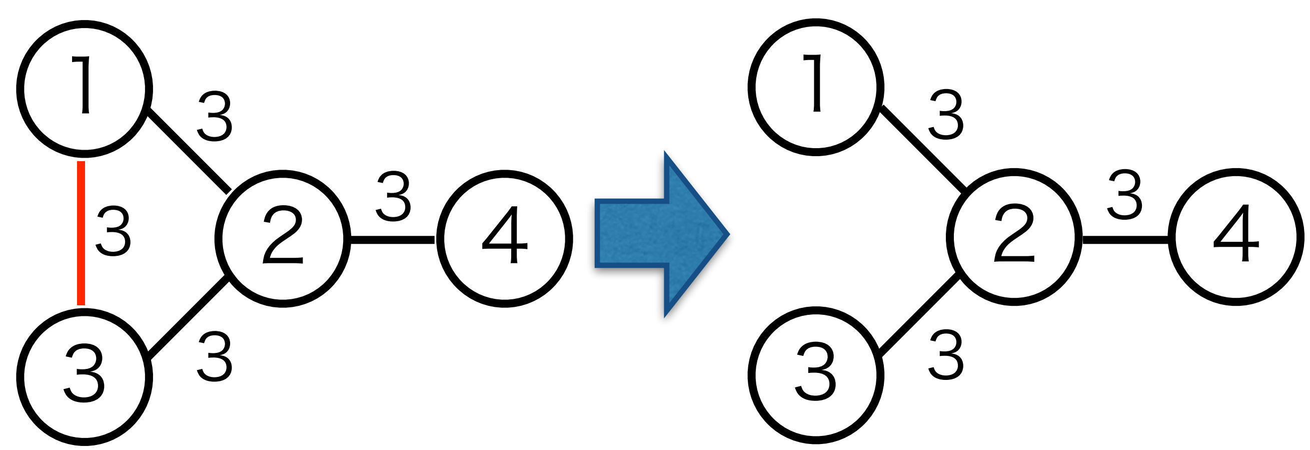
= TLE



解法

- まず最小全域木を1つ求め，そのコスト C を記録
- 求めた最小全域木に含まれる辺 e について，e を取り除いたグラフの最小全域木を求め，そのコスト C' と C を比較
  - 全域木が作れなければ e は必ず最小全域木に必要
  - C < C' ならば，e は必ず最小全域木に必要





解法

- まず最小全域木を1つ求め，そのコスト C を記録
- 求めた最小全域木に含まれる辺 e について，e を取り除い

を比較

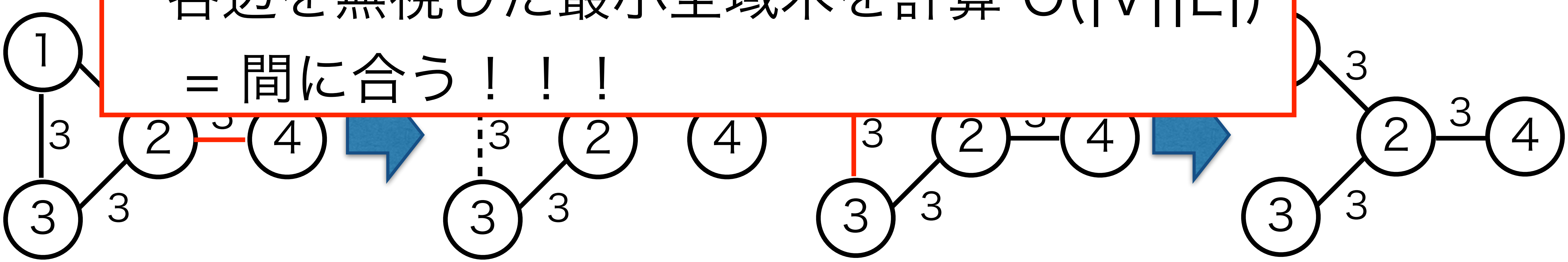
最小全域木に含まれる辺の数 = |V|-1

クラスカル法で

・前処理ソート O(|E| log|V|)

・各辺を無視した最小全域木を計算 O(|V||E|)

＝ 間に合う！！！

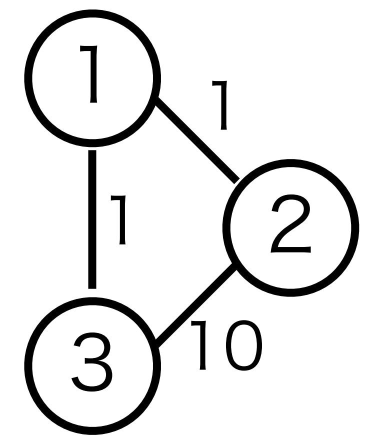


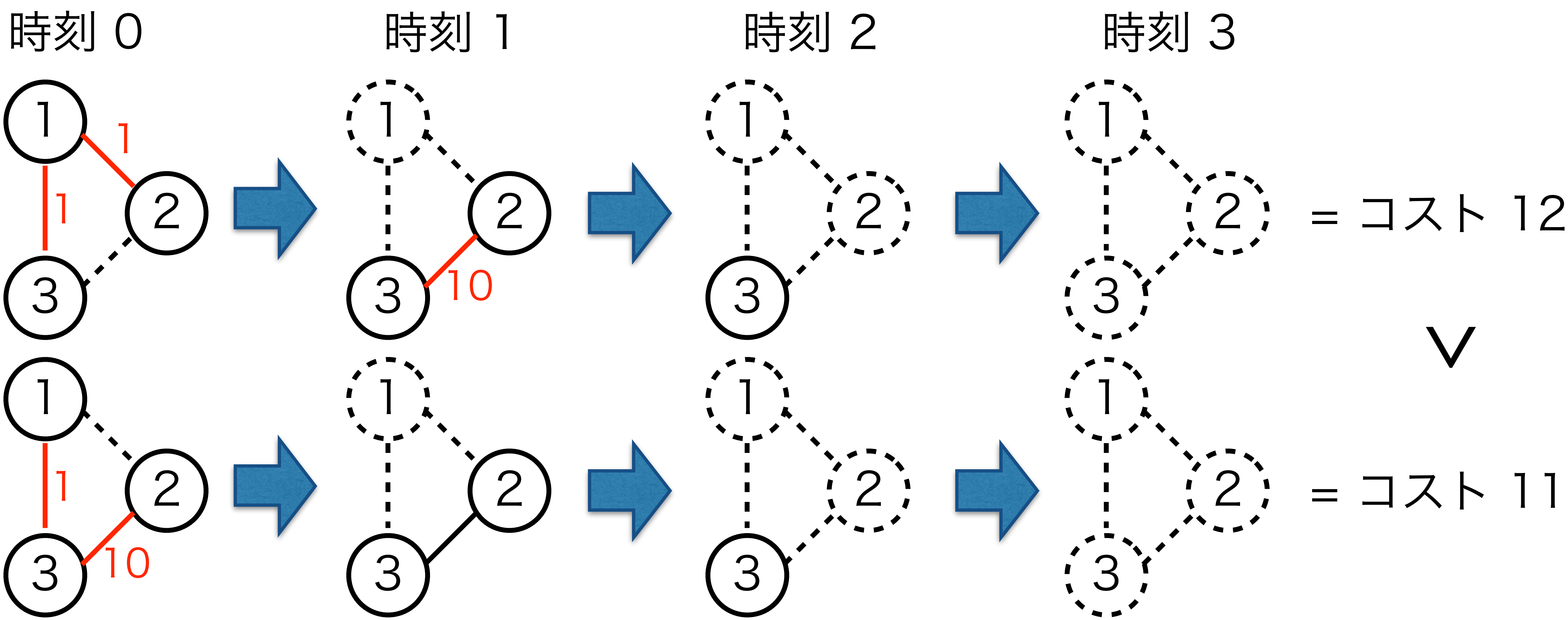
問題例: 模擬国内予選 2013 D "Sinking Islands"

- n 個の島があり，i 番目の島はそれぞれ時間 h_i で沈む
- 橋を作る候補が m 通りあり，j 番目の橋は a_j と b_jを結び、建設に c_j かかる
- 橋は任意のタイミングで建造できる
- 沈んでいない島同士を常に行き来できるようにしたい (どうしてもできなくなったらそれ以上橋は作らない)
- コストを最小化せよ
- 制約: $2 \leq n \leq 200$, 同じ島のペアを結ぶ橋はない

問題例: 模擬国内予選 2013 D "Sinking Islands"

例: $h_1 = 1$, $h_2 = 2$, $h_3 = 3$, 橋の候補:





解法

- 時刻の逆順にシミュレーション
  - すべての島が沈んだ状態から浮いてくると考える
  - 浮いてきた島同士は連結にしなければならない
  - できなくなったらそれまでのつなぎ方はリセット
- 浮いてきた島同士を連結にするとき，全域木にするのがベスト
  - 各時点での最小全域木を求める
- 時刻は $O(n)$，それぞれの最小全域木計算は $O(n^2)$
  - 全体で $O(n^3)$

おまけ：行列木定理

- 無向グラフの全域木の個数がわかる

定義

無向グラフ G = (V, E) のラプラシアン行列 L とは,

- $L_{i,j} = -1$ (i ≠ j かつ e = (i, j) ∈ E)
- $L_{i,j} = 0$ (i ≠ j かつ e = (i, j) ∈ E)
- $L_{i,j} = d_i$ (i = j, ただし di は i の次数)

定理

無向グラフ G の全域木の個数は，G のラプラシアン行列 L の任意の余因子に等しい (余因子はすべて等しい)

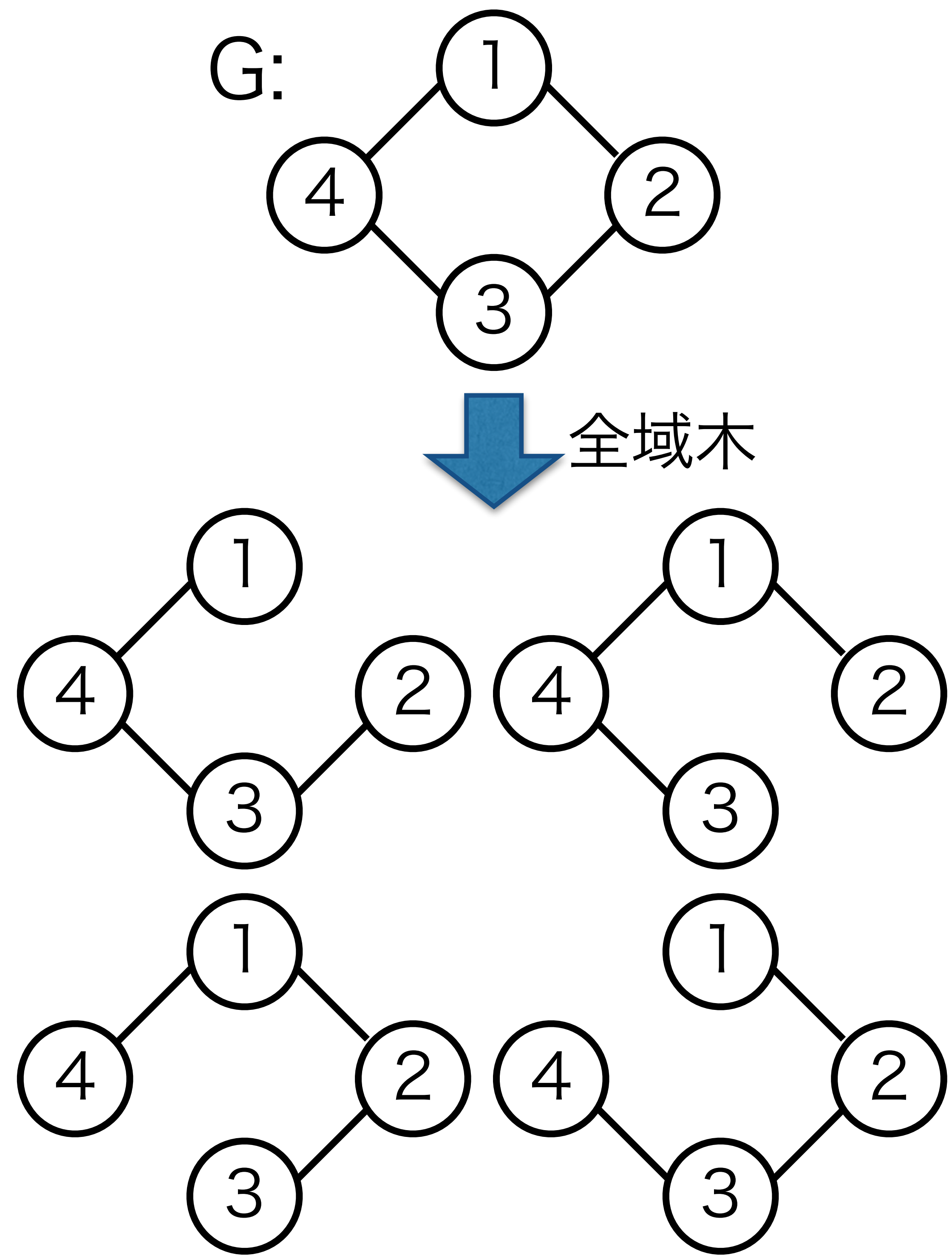
行列木定理: 例

- G の ラプラシアン行列 L は,

$$L = \begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 2 & 0 & -1 \\ -1 & 0 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

- L の (1,2)-余因子は

$$\widetilde{L}(1,2) = -\begin{vmatrix} -1 & 0 & -1 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{vmatrix} = 4$$



全域木関連・その他話題

- 有向全域木
  - 有向グラフ G において，根となる頂点 r が与えられたとき，r から全ての頂点に到達可能な G の非巡回部分グラフを有向全域木と呼ぶ
- シュタイナー木
  - 無向グラフ G = (V, E) と V の部分集合 S について，S をすべて含む連結で非巡回な部分グラフをシュタイナー木と呼ぶ