競プロ勉強会04

elzup

内容

- UnionFind
- 最小全域木
  - クラスカル法
  - プリム法

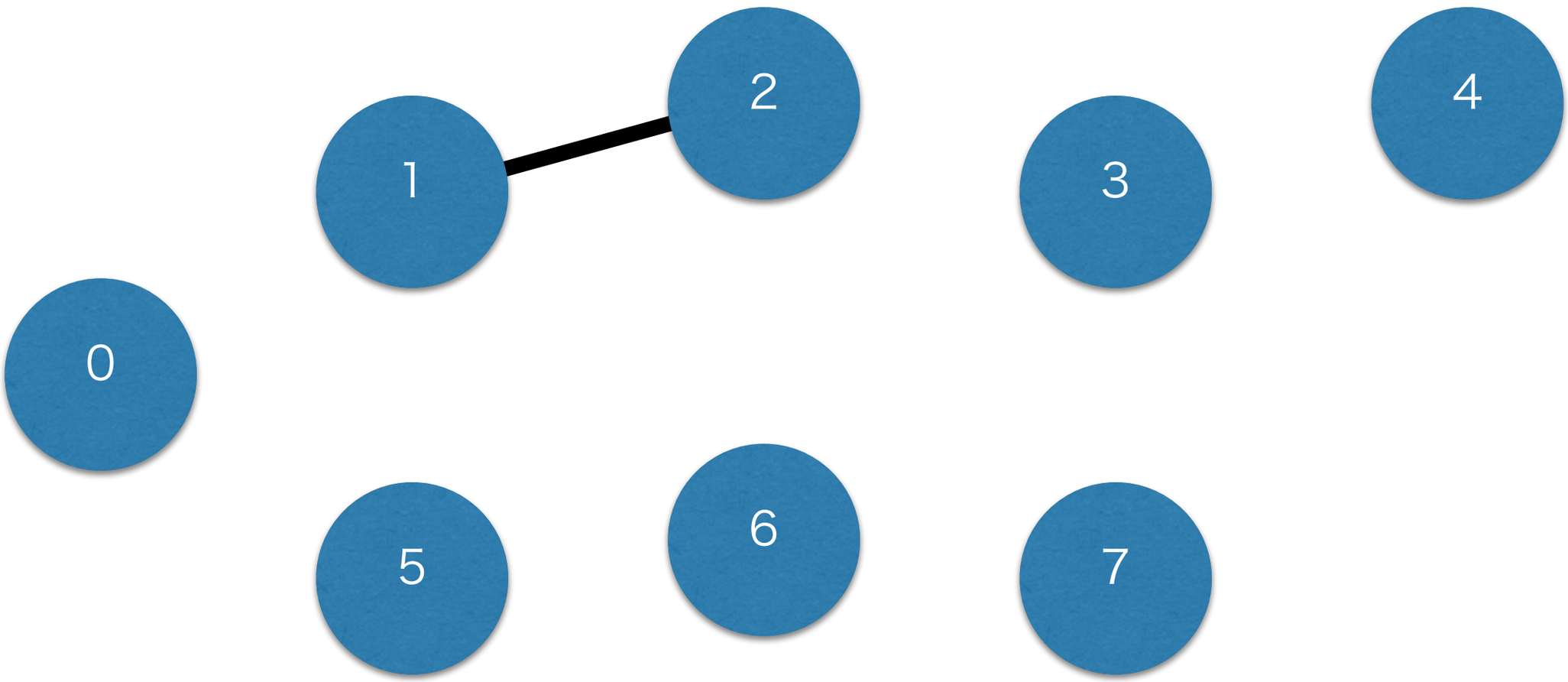
Union Find
(素集合データ構造)

- グループ管理
- AtCoder Typical Contest 001
  http://atc001.contest.atcoder.jp/tasks/unionfind_a

サンプル図解

０１２

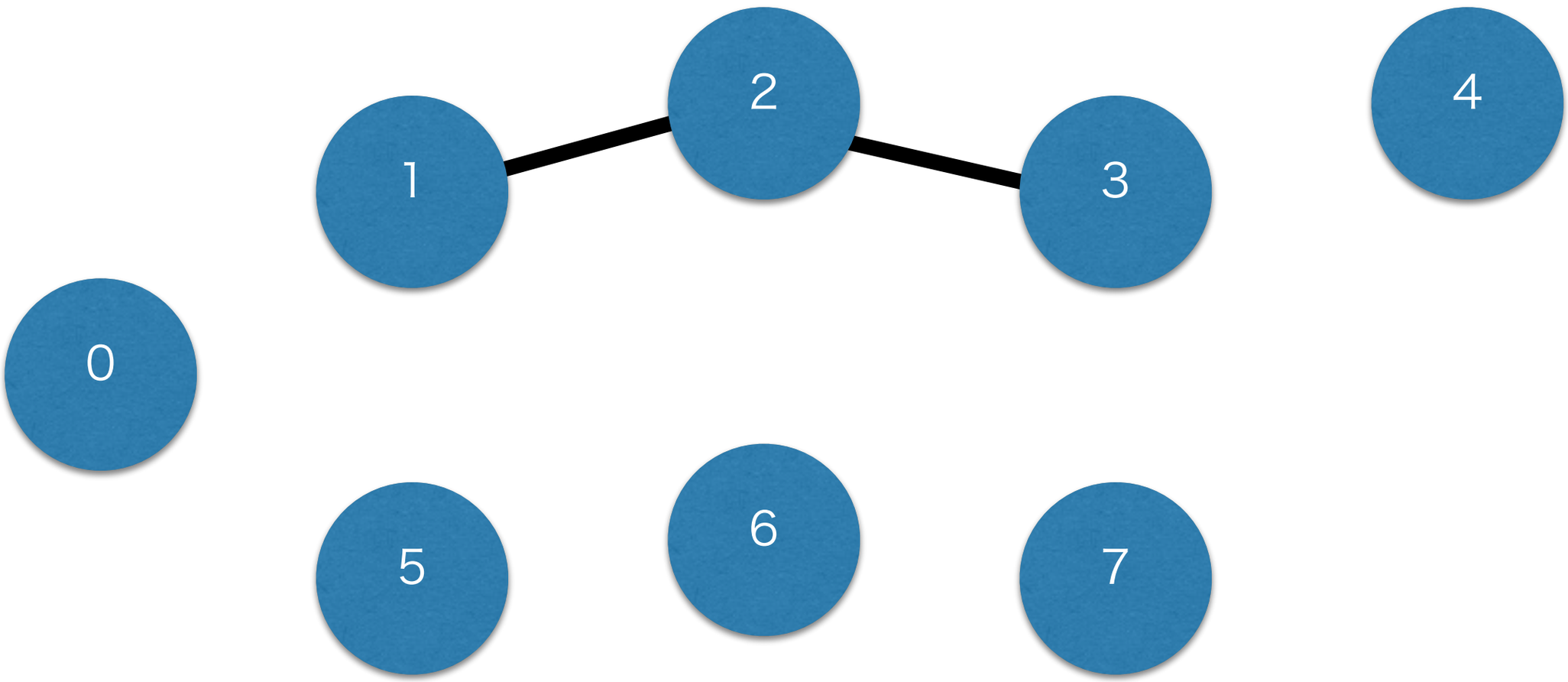
連結



サンプル図解

０３２

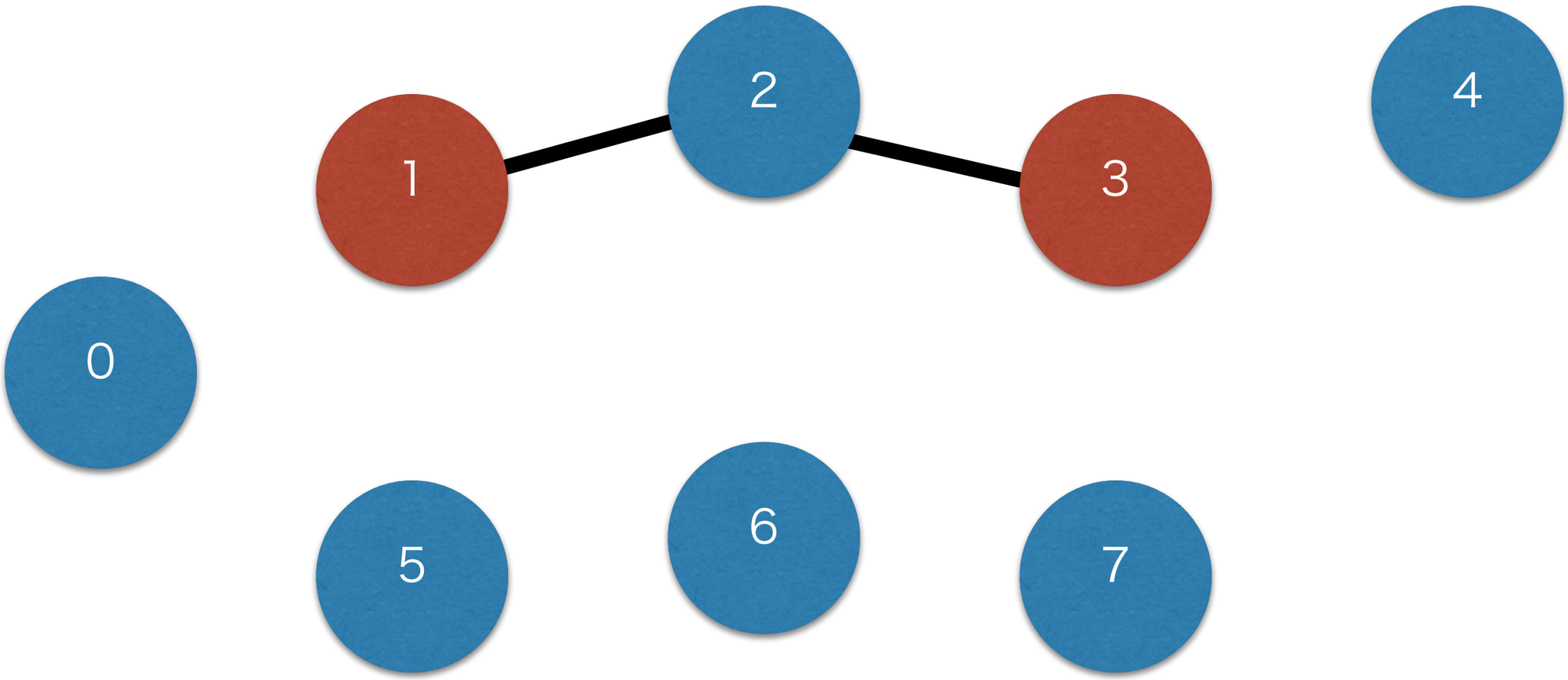
連結



サンプル図解

１１３

確認

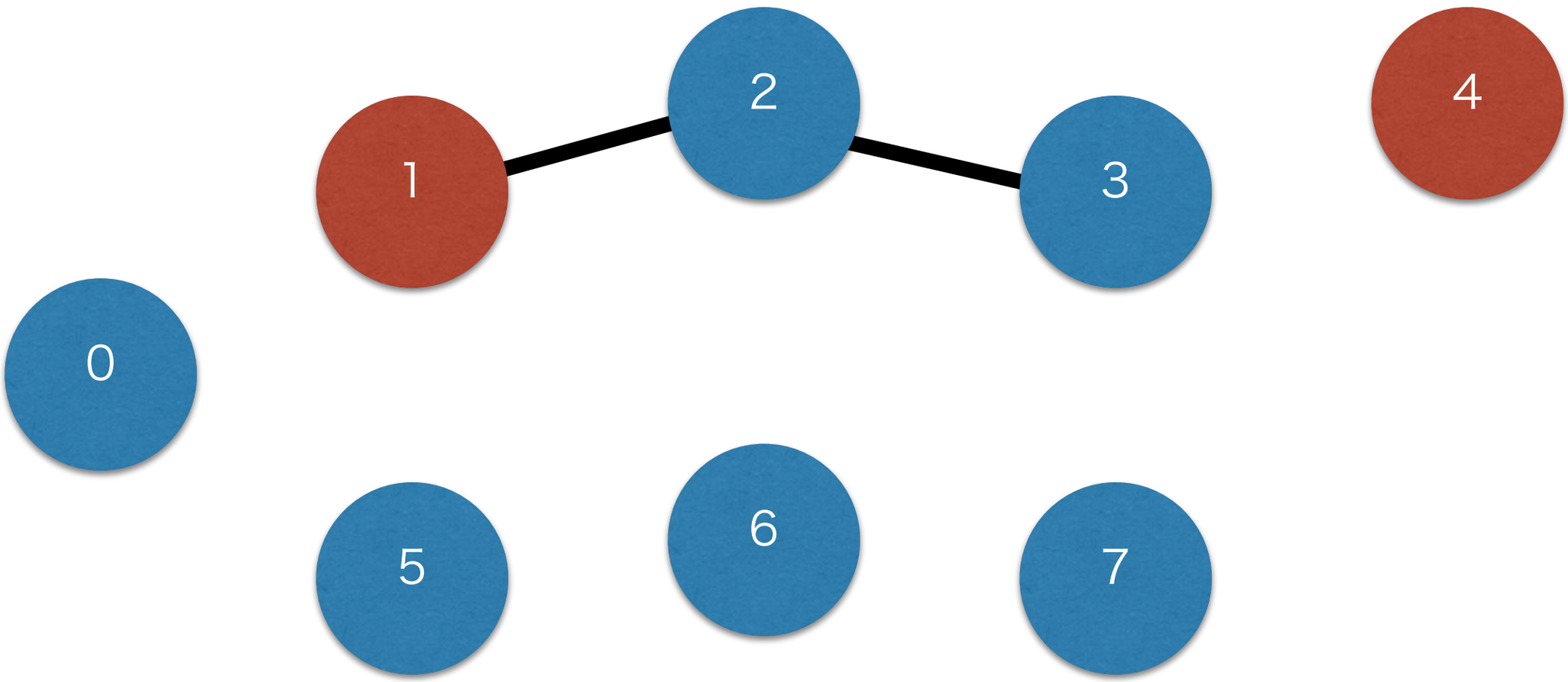


=> Yes

サンプル図解

１１４

確認

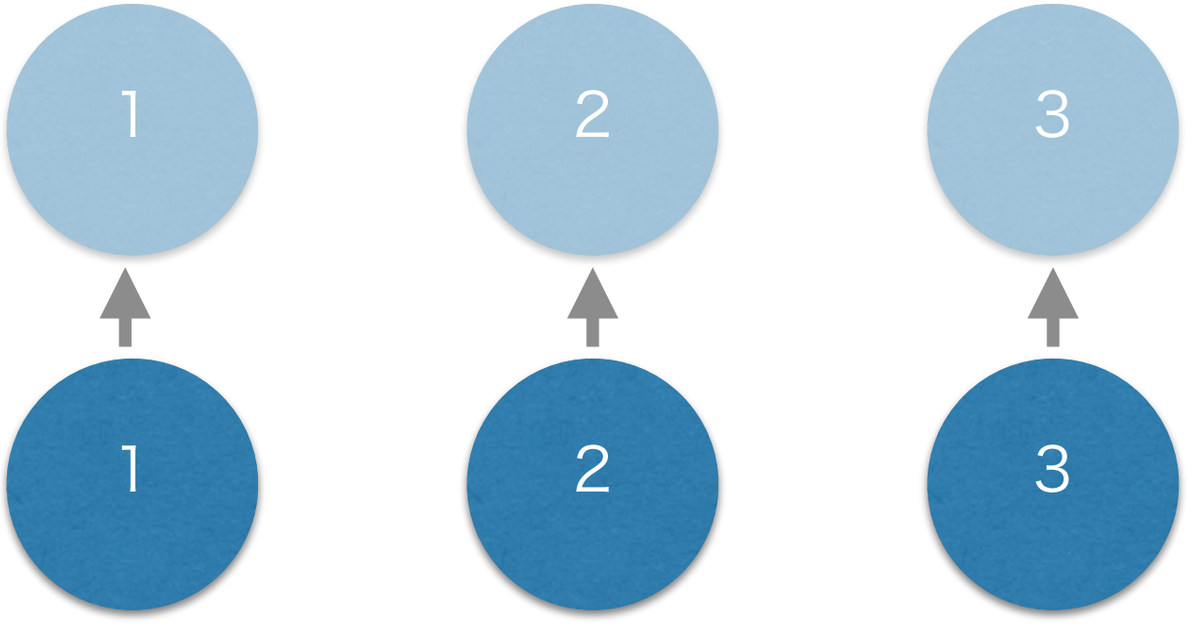


=> No

初期化

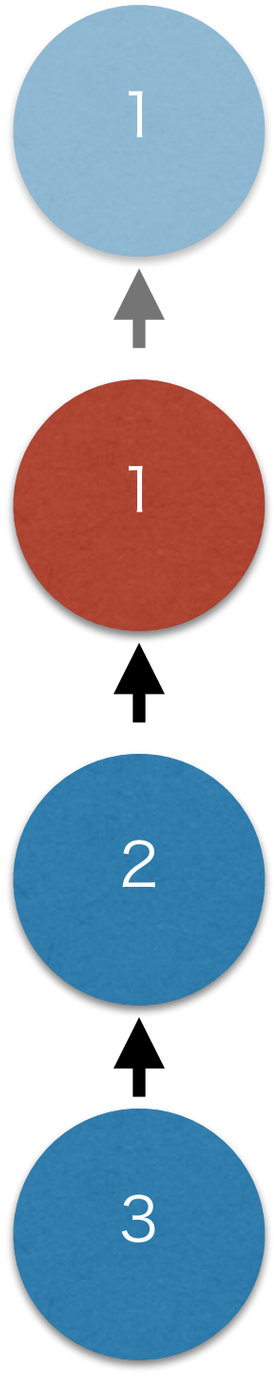
```
・ UnionFind(int n) {
        gid = new int[n];
        for (int i = 0; i < n; i++) {
            gid[i] = i;
        }
}
```

root は自分自身



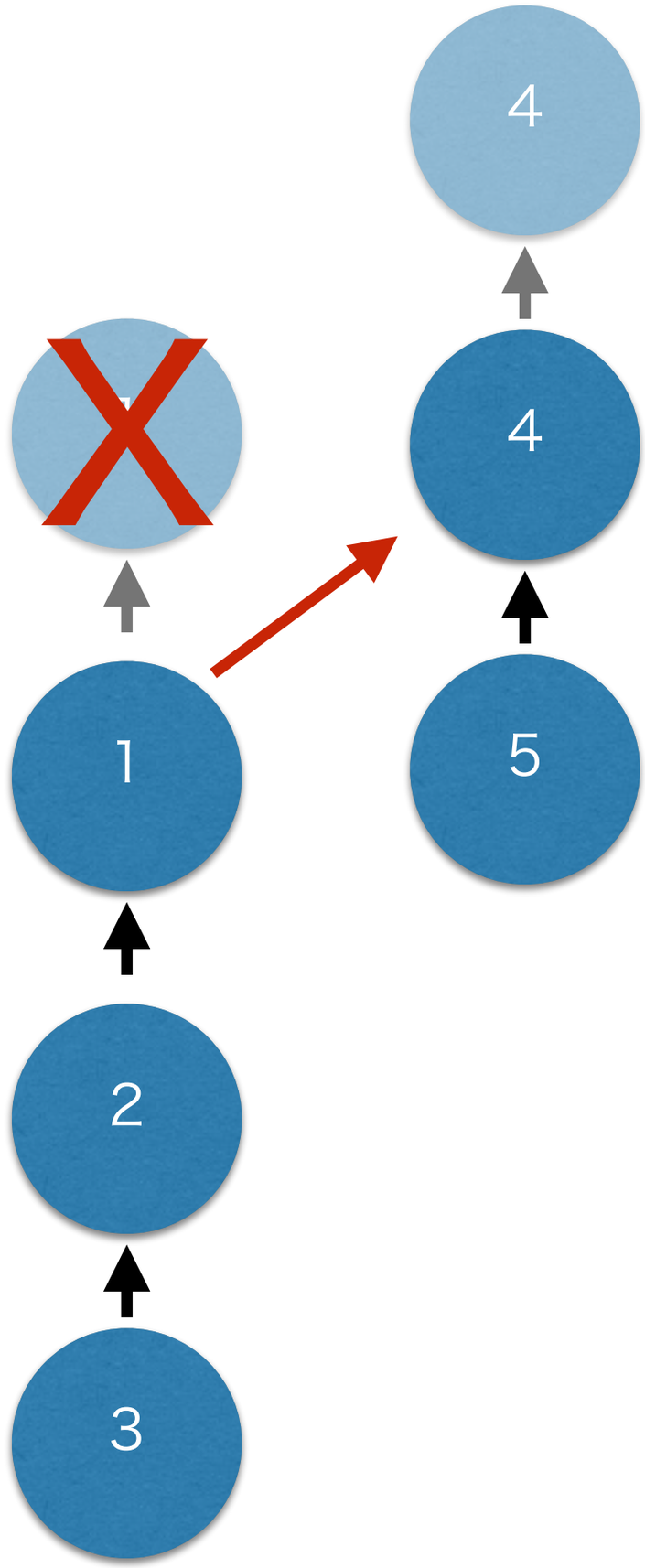
Find():
自分の木のルートを返す

- int find(int a) {
    if (gid[a] == a) {
        return a;
    }
    return find(gid[a]);
}



merge():
木を結合する

- void merge(int a, int b) {
    int ra = find(a);
    int rb = find(b);
    if (ra == rb) {
        return;
    }
    gid[ra] = rb;
}



毎回ルートにつなぎなおす

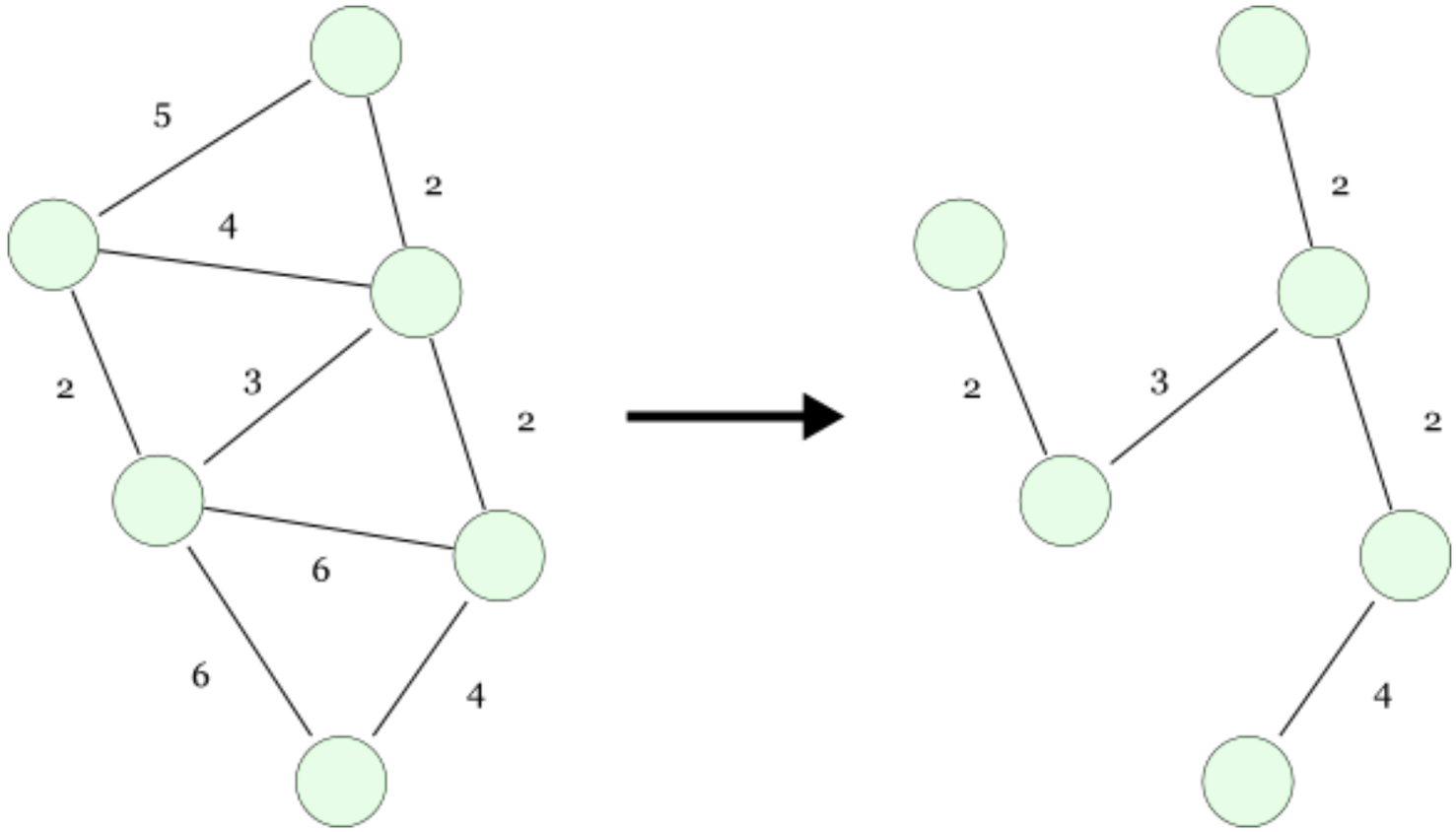
```
int find(int a) {
    if (gid[a] == a) {
        return a;
    }
    return gid[a] = find(gid[a]);
}
```

最小全域木

(Minimum Spanning Tree)

最小全域木

- コストの総和が最小となる全域木



クラスカル法

- エッジ(辺)に注目
- https://ja.wikipedia.org/wiki/クラスカル法
- http://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=GRL_2_A&lang=jp

クラスカル法手順

- エッジをリスト化してソートしておく
- ノードを UnionFind で管理して木の数を記録する
- 木の数が一つにまとまる(全域木になる)まで {
  小さい順にエッジを有効化(両端のノードをmerge)する
  }

ソート

- (implements Comparable<Edge>)

```
...
    @Override
    public int compareTo(Edge o) {
        return this.w - o.w;
    }
...
        Arrays.sort(edges);
```

コストのカウント

- UnionFind uf = new UnionFind(V);
  int weight = 0;
  for (Edge e : edges) {
      if (!uf.equals(e.s, e.t)) {
          weight += e.w;
          uf.merge(e.s, e.t);
          if (uf.groups == 1) {
              break;
          }
      }
  }

プリム法

- 頂点に注目
- https://ja.wikipedia.org/wiki/プリム法
- http://algo-visualizer.jasonpark.me/#path=mst/prim/normal

プリム法手順

- 頂点ごとに接続される辺をリスト化

- 任意の頂点をスタートとする(ダミーを PriorityQueue (pq)に追加)

- 訪問済みの頂点全てから伸びる辺の中で最小コストの辺 (pq .poll) をみる

    - 未訪問の頂点なら訪問済みにして, その頂点に付属する辺をすべて pq に追加

実装

- 「頂点ごとのエッジ」リスト初期化

```
List<Edge>[] edges = new ArrayList[V];
for (int i = 0; i < V; i++) {
    edges[i] = new ArrayList<>();
}
```

実装

- Queue<Edge> q = new PriorityQueue<>();
- q.add(new Edge(0, 0, 0));

実装

```
while (!q.isEmpty()) {
    Edge e = q.poll();
    if (done[e.t]) {    continue;    }
    done[e.t] = true;
    costSum += e.w;
    q.addAll(edges[e.t]);
}
```