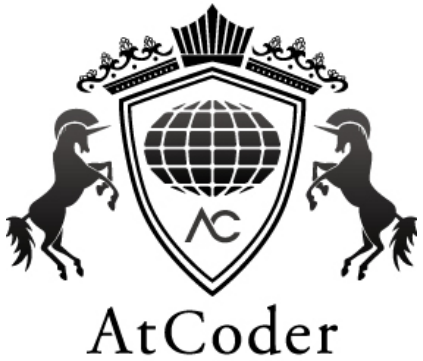
Union-Find
（素集合データ構造）



Union-Find 木の機能

AtCoder

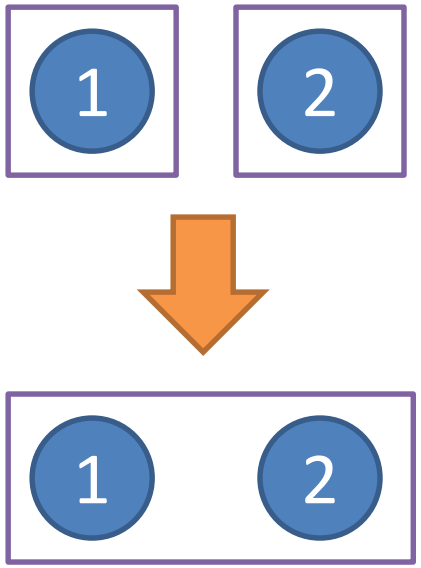
- グループ分けを管理する
- はじめ，n 個の物は全て別々のグループ

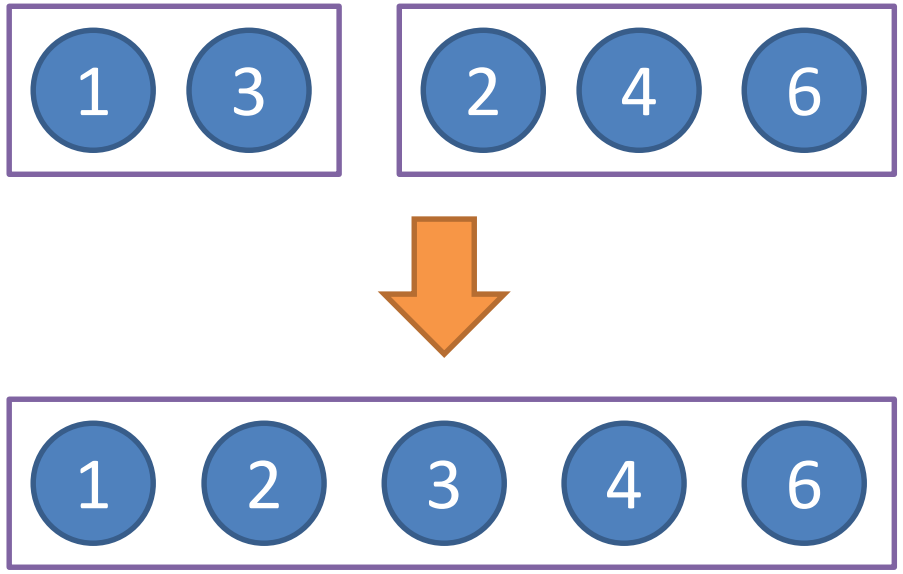
- 次の 2 種類のクエリに対応する
  1. 「まとめる」
  2. 「判定」

クエリ 1：まとめる

AtCoder

- 2 つのグループを 1 つにまとめる

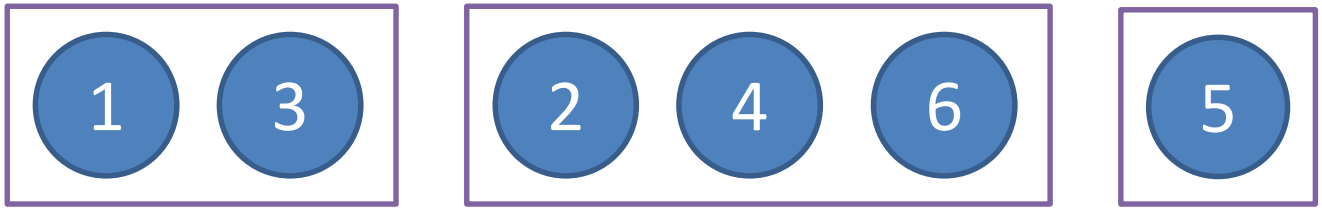




クエリ 2：判定

AtCoder

- 2 つの要素が同じグループに属しているかを判定する

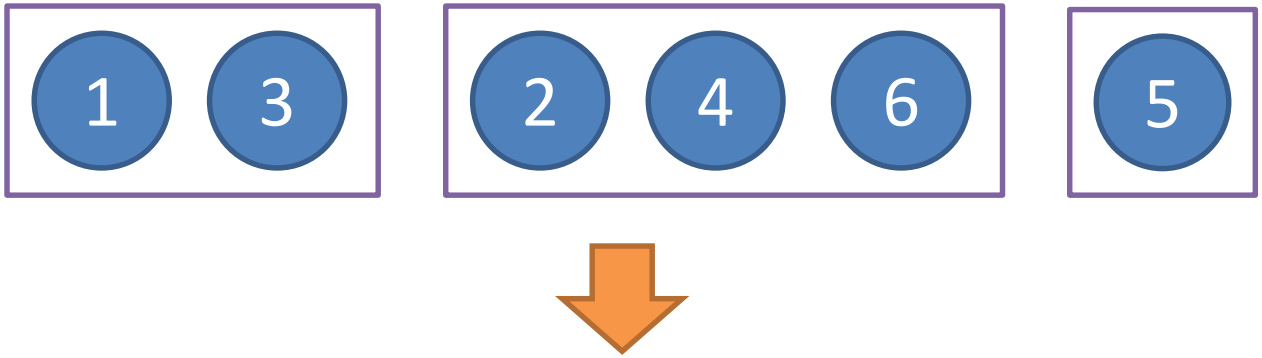


1 と 3 → true
1 と 2 → false

素朴な実現法 (1/2)

AtCoder

- 配列に，同じグループなら同じ数字を入れておく



1	2	3	4	5	6
1	2	1	2	5	2

素朴な実現法 (2/2)

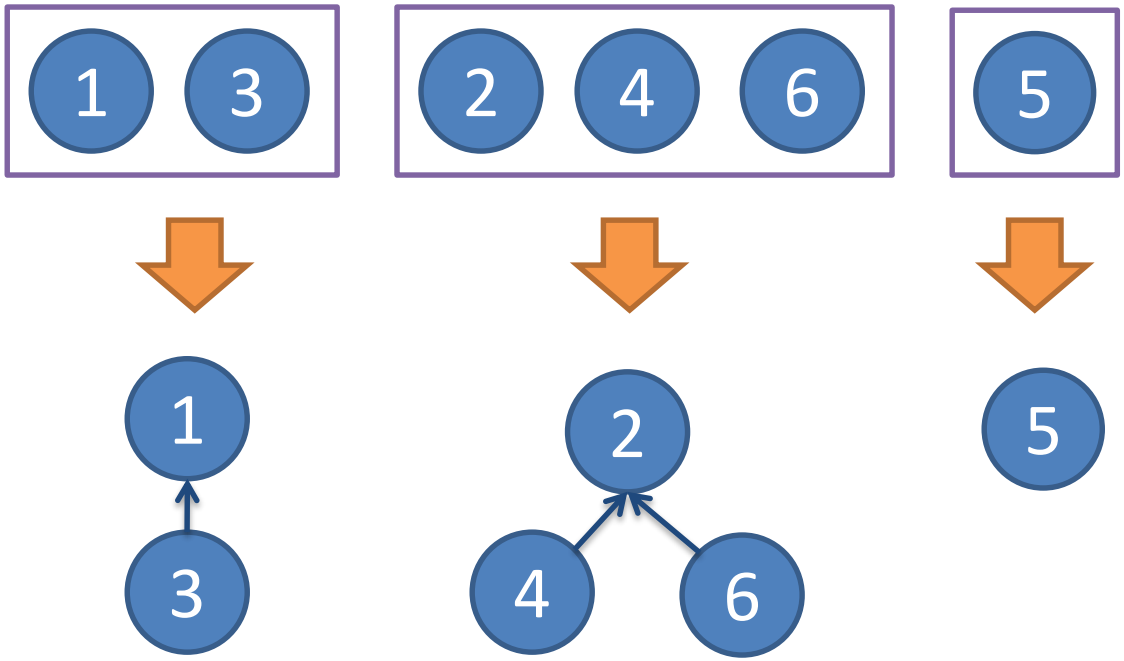
AtCoder

- この方針の問題点
  - グループをまとめる際に，$O(n)$ 時間かかってしまう
  - 実際には，この方針でも少しの工夫でならし $O(\log n)$ 時間にできます
    http://topcoder.g.hatena.ne.jp/iwiwi/20131226/1388062106

- Union-Find 木は，もっと効率的に行う

AtCoder

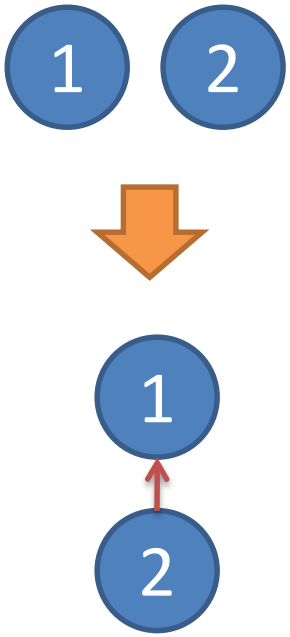
- グループを，1 つの木で表現する
  - したがって，全体では森



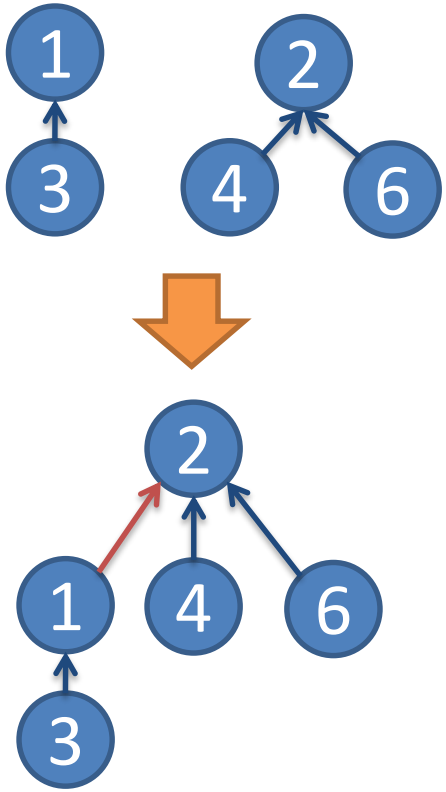
クエリ1：「まとめる」

AtCoder

- 片方の木の根からもう片方の根に辺を張ればよい



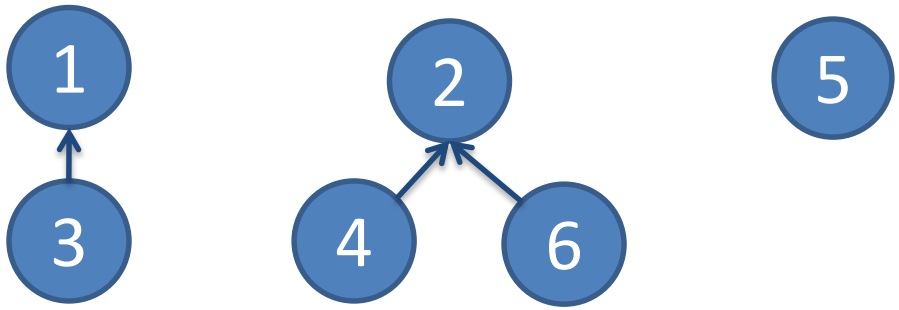
1 と 2 をまとめる
2 から 1 に辺を張る



クエリ2：「判定」

AtCoder

- 2 つの要素を上に辿って，根が同じかどうかを見ればよい



2 と 6 → 根は共に 2 → true
1 と 4 → 根は 1 と 2 → false

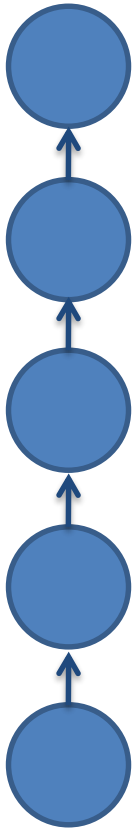
Union-Find 木の効率

AtCoder

- **正当性**：このやり方で，グループの判定はうまくできる

- **効率**：最悪の場合，この方法でもツリーが縦長になると処理が遅くなってしまう

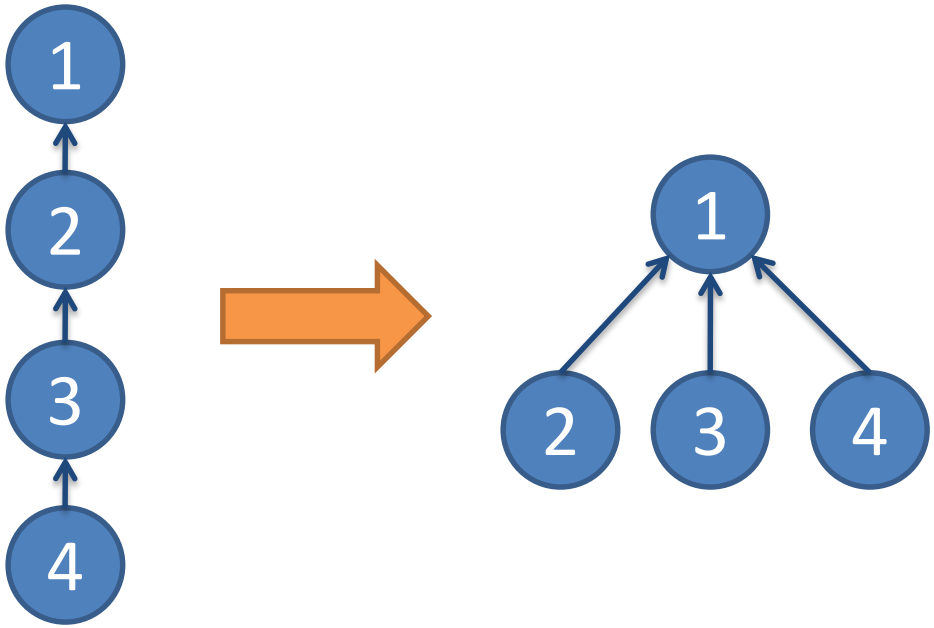
→　**効率化のテクニックを導入**



効率化工夫 1：経路圧縮

AtCoder

- 上向きに辿って再帰的に根を調べる際に，調べたら辺を根に直接つなぎ直す

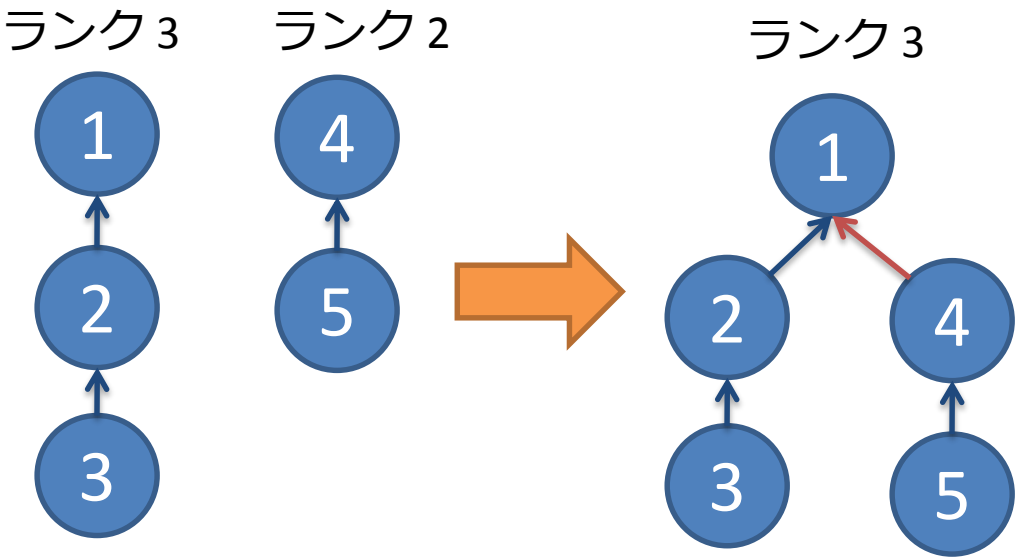


- 4 の根を調べると，2, 3, 4 の根が 1 と分かる

効率化工夫 2：ランク

AtCoder

- 木の高さを持っておき，低い方を高い方に繋げるようにする



- ただし，経路圧縮と組み合わせた際，経路圧縮による高さの減少は無視する
  （つまり，ランクは経路圧縮しなかった場合の高さに相当する）

Union-Find 木の計算量

AtCoder

- 2 つの工夫の両方をして $O(\alpha(n))$
  - $\alpha(n)$ はアッカーマン関数 $A(n,n)$ の逆関数
  - 相当小さい（log より小さい！）
- 片方だけでも，だいたい $O(\log n)$
  - 経路圧縮のみが実装が楽でよい
- これらはならし計算量

Union-Find 木の実装（ランク無し）

AtCoder

- 経路圧縮のみ（ランクは使わない）

```
int par[MAX_N];     // 親の番号

// n 要素で初期化
void init(int n) {
        for (int i = 0; i < n; i++) par[i] = i;
}
```

- par[i] = i ならば根
  - はじめは全部の頂点が根

Union-Find 木の実装（ランク無し）

AtCoder

```
// 木の根を求める
int root(int x) {
    if (par[x] == x) {                    // 根
        return x;
    } else {
        return par[x] = root(par[x]);     // 経路圧縮
    }
}

// x と y が同じ集合に属するか否か
bool same(int x, int y) {
    return root(x) == root(y);
}
```

Union-Find 木の実装（ランク無し）

AtCoder

```
// x と y の属する集合を併合
void unite(int x, int y) {
    x = root(x);
    y = root(y);
    if (x == y) return;

    par[x] = y;
}
```

Union-Find 木の実装（ランク有り）

AtCoder

```
int par[MAX_N], rank[MAX_N];

void init(int n) {
  for (int i = 0; i < n; i++) {
    par[i] = i;
    rank[i] = 0;
  }
}

int root(int x) {
  return par[x] == x ? x : par[x] =
root(par[x]);
}

bool same(int x, int y) {
  return root(x) == root(y);
}
```

```
void unite(int x, int y) {
  x = root(x);
  y = root(y);
  if (x == y) return;

  if (rank[x] < rank[y]) {
    par[x] = y;
  } else {
    par[y] = x;
    if (rank[x] == rank[y]) rank[x]++;
  }
}
```

補足

AtCoder

- Union-Find 木はグループをまとめることはできても，分割することはできない

  重要な制約

- Union-Find 木を改造して機能を付加することが必要になるような問題もあります