Code Thanks Festival 2017

解説

2017/12/02

Hec

Problem A
「Time Penalty」

問題概要

- 問題が８問あり、i 問目の解答状況が t_i で与えられる
  - $t_i > 0$ の場合はコンテスト開始後 t_i 秒で正解
  - $t_i = 0$ の場合はまだ正解していない
- 時間ペナルティは以下のように定義される
  - 最後に正解した問題の t_i と等しい
  - 1問も正解していない場合は 0 とする
- 時間ペナルティを求めよ

解説

- **時間ペナルティは t_i の最大値を求めれば良い**
  - 1問以上正解している場合は、最後に正解した問題の t_i と等しいので
  - そうでない場合、全ての i について $t_i = 0$ であり、定義より 0 なので

- 解法
  - 8つの整数 t_i を標準入力から入力する
  - t_i の最大値を計算する
  - 計算結果を標準出力に出力する

Problem B
「Concatenated Palindrome」

問題概要

- 英小文字から成る文字列 S が与えられる
- S + T が回文になるような文字列 T を考える
  - ここで、S + T は 文字列 S の後に文字列 T を連結した文字列
- 条件を満たす T の中で、T の最小の長さを求めよ

- 制約
  - 1≦|S|≦50 (ここで |S| は文字列S の長さ)

解説

- 条件を満たすＴの中で、Ｔの最小の長さを求めたい
- **Ｔの長さを 0 から小さい順にループで試す**
  - Ｔの長さを決めたときにＳ＋Ｔの回文判定の方法は？
  - 答えの候補となるＴの最大の長さは？

解説

- 文字列 T の長さを決めて、S + T の回文判定
  - 通常の回文判定と同じように判定する
    - S + T の先頭から i 文字目と後ろから i 文字目が同じか比較する
  - S + T の先頭から |S| + 1 文字目以降は任意の文字が使える
  - **(|S| ≥ |T| の場合) T の後ろから i 文字目を S の先頭から i 文字目に一致させれば良い**
    - ここで、|S| は文字列 S の長さ、|T| は文字列 T の長さとする
  - # を任意のアルファベットを表す文字としておき、T = “#...#” として S + T の回文判定するのが簡単

解説

- 答えの候補となる T の最大の長さ
  - Sの先頭 1 文字目から |S|−1 文字目 まで取り出して、逆順にした文字列を T にすれば、S+T は回文
    - ✦ 例: S = “abcde” T = “dcba” S + T = “abcdedcba”
  - **答えの上限は |S|−1**

解説

- 全体の流れ
  - 文字列 T の長さを 0 から |S|−1 まで試す
    - 文字列 T の長さを決めて回文判定する
    - 回文であると判定されたら 文字列 T の長さ が答え
  - |T| = |S|−1 のときに必ずアルゴリズムは終了する

- 時間計算量 $O(|S|^2)$

Problem C
「Factory」

問題概要

- 機械が N 台あり、K 個のプレゼントを製造する
  - i 番目の機械で s_i 回目のプレゼント製造に $a_i + (s_i - 1) \times b_i$ 秒かかる
  - 複数の機械を同時に動かしてはいけない
- プレゼント K 個の製造にかかる最小の合計時間を求めよ

- 制約
  - $1 \leqq N \leqq 10^5$
  - $1 \leqq K \leqq 10^5$
  - $1 \leqq a_i \leqq 10^9$
  - $0 \leqq b_i \leqq 10^9$

解説

- 複数の機械を同時に動かしてはいけない
- 機械を1つ選んで、プレゼントを1個作ることを K 回繰り返す

- **プレゼントを作る時間が最小となる機械を選んで、その機械の状態を更新すれば良い**
  - i 番目の機械でプレゼント1個作るのにかかる時間 t_i とする
  - 初期化: 全ての機械に対して $t_i = a_i$
  - 更新:　j 番目の機械を使用するとき $t_j = t_j + b_j$

解説

- プレゼントを作る時間が最小となる機械を選ぶには?

- プレゼントを作る時間をもとに毎回ソートして機械を選ぶ
  - 時間計算量: O(kn log n) なので TLE となる

- **プレゼントを作る時間をキーとして優先度付きキューで管理**
  - 時間計算量: O((n + k) log n) なので間に合う

Problem D
「Bus Tour」

問題概要

- 1グループ N 人で参加申し込みがあるバスツアーを行う
- 申し込みグループ数に上限はない
- 各バスの定員は M 人
- バスの台数は参加者全員が乗り切れるような最小台数
- 考えられるバスの最大空席数を求めよ

- 制約
  - $1 \leqq N \leqq 10^9$
  - $1 \leqq M \leqq 10^9$

解説

- 申し込みグループ数を a、バスの台数を b とする
- 参加人数は Na 人、バスの定員の合計は Mb 人
- したがって、バスの空席数は Mb−Na 人
- バスの台数が参加者全員が乗り切れる最小台数のとき

バスの空席数が 0 以上 かつ M−1 以下となる

- 空席が M 席以上なら、バスの台数を減らす必要がある
- 空席が 0 席未満なら、バスの台数を増やす必要がある

解説

- Mb−Na が 0 以上 かつ M −1 以下 になるように M を足したり、引いたりということは？

  →バスの空席数は Mb−Na を M で割ったあまりと一致する

  つまり、バスの空席数は −Na mod M

解説

- バスの空席数を最大化するということは?
- 負の整数の mod M を考えるのは大変なので

  −Na mod M → M − (Na mod M) と式変形する
- **求めたいのは Na ≡ k mod M となる最小の正の整数 k**
- k が分かると、答えは M − k

解説

- Na ≡ k mod M は k を固定すると a についての1次合同式
- **a が解を持つ条件は k が gcd(N,M) の倍数のとき**
- kは最小の正の整数なので → k = gcd(N,M)
- **答えは M − gcd(N,M)**
  - gcd(a,b) の時間計算量は O(log max(a,b)) なので間に合う

Problem E
「Coin Authentication」

問題概要

- 重さ w_i g のコインが 10000 枚入った袋が N 個ある
- 本物コインと偽物コインでその重さは異なる
- 袋からコインを選んで質問クエリを投げると、選んだコインの重さの合計を知ることができます。
- 全ての袋に対して、袋の中のコインの真偽を判定せよ

- 制約
  - $1 \leqq N \leqq 50$
  - $w_i \in \{8,9,10,11,12\}$ (入力では与えられない)
  - 質問クエリを投げる回数は10回以下

解説

- 質問クエリを投げることが最大で10回しかできない
- 1つの袋ごとに真偽を調べるとクエリ回数が足りなくなる
- **1回の質問クエリで、複数の袋について袋の中のコインの真偽を判定したい**

解説

- **コインの重さが5通りしかないことに注目する**

  → 袋の中のコインの重さを5進数の各桁に対応させる

- 5つの袋の中のコインの真偽を判定する場合を考える

- 袋の中のコインの重さは w_1,w_2,w_3,w_4,w_5 とする

- 「$w_1w_2w_3w_4w_5$」を5進数とみなして、w_1 のコインを625枚、w_2 のコインを125枚、w_3 のコインを25枚、w_4 のコインを 5 枚、w_5 のコインを1枚と選んで質問クエリを投げると、クエリの答えが「$w_1w_2w_3w_4w_5$」の10進数表記に対応する

解説

- **コインの重さが5通りしかないことに注目する**

  → 袋の中のコインの重さを5進数の各桁に対応させる

- 例: $w_1=3, w_2=1, w_3=0, w_3=2, w_4=4$

- w_1 のコインを625枚、w_2 のコインを125枚、w_3 のコインを25枚、w_4 のコインを 5枚、w_5 のコインを1枚と選んだ場合

  $\rightarrow 3 \times 625 + 1 \times 125 + 0 \times 25 + 2 \times 5 + 4 \times 1 = 2014$

- 2014 を5進数表記すると「31024」となる

解説

- **コインの重さが5通りしかないことに注目する**
- 5進数の各桁は 0 から 4 までの整数に対応する
- **コインの重さを 8g から12g ではなく、0g から 4g とみなす**
- 質問クエリでコインの重さを計測した後に、8× 「選んだコインの枚数」 をコインの重さから引けば良い

解説

- 1回の質問クエリの流れ
  - 真偽が分かっていないコインの袋を最大で5つ選ぶ
  - それぞれの袋の中から、1枚、5枚、25枚、125枚、625枚のコインを選んで、質問クエリを投げる
  - クエリの答えから 8×「選んだコインの枚数」を引いた値を5進数に変換して、コインの真偽を判定する

- 袋の数 N は最大で50なので、質問クエリの回数は高々10回で全ての袋の中のコインの真偽を判定可能

Problem F
「Limited Xor Subset」

問題概要

- N 個の正の整数 a_i が与えられる
- N 個の整数のうち 0 個以上の整数を選び、選んだ全ての整数についてビットごとのXOR が K となるような選び方は何通りあるか mod $10^9 + 7$ で求めよ
  - ただし、0 個選んだときのビットごとのXORは 0 とする
- 制約
  - $1 \leqq N \leqq 10^5$
  - $0 \leqq K \leqq 10^5$
  - **$1 \leqq a_i$, $a_1 + \ldots + a_N \leqq 10^5$**

解説

- 動的計画法 (DP) で数え上げを考える
- 解法0: (TLE 解法)
  - dp[i][j]: a_1, ..., a_i から、XOR が j となる整数の選び方の個数
  - 初期化: dp[0][0] = 1, それ以外 0
  - 遷移:
```
for(i = 0 ; i < n; ++i) {
        for(j = 0; j < 2*max(a); ++j){
            dp[i+1][j] += dp[i−1][j];
            dp[i+1][j^a[i+1]] += dp[i−1][j];
        }
    }
```
  - 答え: dp[n][k]

解説

- 解法 0 の時間計算量は O(n max(a)) で TLE
- **そこで、和の制約 $a_1 + \ldots + a_N \leqq S$ を活用する**
  - この問題では S = 10^5 となる

- 想定解法は 2 通りあります
  - 解法1: a をソートしてから DP
  - 解法2: 重複する a をまとめて計算するDP

解説

- 解法1: a_i をソートしてから DP
  - **a_i を昇順にソートする**
  - dp[i][j]: $a_1, \ldots, a_i$ から、XOR が j となる整数の選び方の個数
  - 初期化: dp[0][0] = 1, それ以外 0, limit = 0
  - 遷移:
```
for(i = 0 ; i < n; ++i){
    for(j = 0; j <= limit; ++j){
        dp[i+1][j] += dp[i−1][j];
        dp[i+1][j^a[i+1]] += dp[i−1][j];
    }
    limit |= a[i+1];
}
```
  - 答え: dp[n][k]

解説

- 解法1: a_i をソートしてから DP
  - 現在 a_i の要素に注目しているとする
  - ビット操作を考えると limit の上限は $2a_i$ で抑えられる
  - インデックス i についての残りのループ回数の上限は $(S - a_i)/a_i$
  - **したがって、dp配列を参照する回数の上限は $2S - 2a_i$**
  - 時間計算量: O(S)
  - 空間計算量: O(S)
    - dp配列は2つの配列を使いまわすことで、O(NS) から減らせる

解説

- 解法2: 重複する a をまとめて計算するDP
  - 正の整数 x が m 個あり、その中から整数を選ぶことを考える
  - **選んだ整数の XOR は 0 または x であり、それぞれ 2^{m-1} 通りの整数の選び方がある**
    - XOR が 0となる選び方は $ {}_mC_0 + {}_mC_2 + \ldots + {}_mC_{2p} + \ldots $ (1)
    - XOR が xとなる選び方は $ {}_mC_1 + {}_mC_3 + \ldots + {}_mC_{2p+1} + \ldots $ (2)
    - 式(1) + 式(2) = $ {}_mC_0 + {}_mC_1 + \ldots + {}_mC_{2p} + {}_mC_{2p+1} + \ldots = (1+1)^m = 2^m $
    - 式(1) − 式(2) = $ {}_mC_0 - {}_mC_1 + \ldots + {}_mC_{2p} - {}_mC_{2p+1} + \ldots = (1-1)^m = 0 $
  - 重複する a_i のXORの計算を1つにまとめることができる

解説

- 解法2: 重複する a をまとめて計算するDP
  - **重複する a を1つにまとめて、 b とします**
  - b の要素数は a の種類数であり、m とおきます
  - c_i をaに含まれる b_i の個数とおきます
    - 例: a = {1,1,2,2,2,3,3,3,4} → b = {1,2,3,4}, c = {2,3,3,1}

解説

- 解法2: 重複する a をまとめて計算するDP
  - dp[i][j]: $b_1, \ldots, b_i$ から、XOR が j となる整数の選び方の個数
  - 初期化: dp[0][0] = 1, それ以外 0
  - 遷移: for(i = 0 ; i < m; ++i) {

```
for(j = 0; j < 2*S; ++j){
    dp[i+1][j] += power(2, c_i −1) × dp[i−1][j];
    dp[i+1][j^a[i+1]] +=  power(2, c_i −1) × dp[i−1][j];
}
}
```

  - power(x, y) は x の y 乗を表す
  - 答え: dp[m][k]

解説

- 解法2: 重複する a をまとめて計算するDP
  - m の個数は最大でいくつか?
  - **b の要素は互いに異なり、1 + 2 + ... + 447 = 100128 > S であるため、mの個数は最大で446個であり、m = O(sqrt(S))**
  - 時間計算量: O(S sqrt(S))
  - 空間計算量: O(S sqrt(S))

Problem G
「Mixture Drug」

問題概要

- N 種類の薬品があります
- 混ぜると毒が発生する薬品の2つの組が M 通りある
- 毒が発生させずに、最大で何種類の薬品を混ぜられるか?

- 制約
  - $1 \leqq N \leqq 40$
  - $0 \leqq M \leqq N(N-1)/2$

解説

- 問題をグラフ理論の用語で言い換えます
  - 薬品は頂点に対応し，毒が発生する2つ組は辺に対応する
  - N 頂点 M 辺の単純無向グラフが与えられます
  - このグラフに対して、以下の条件を満たす頂点集合 S を考えます
    - S に含まれる2頂点を繋ぐ辺は存在しない
  - 条件を満たす頂点集合 S のうち max |S| を求めよ

解説

- 条件を満たす頂点集合 S とは?　→ **独立集合**
- **与えられたグラフの最大独立集合のサイズを求めれば良い**

- 想定 TLE 解法
  - 全列挙 時間計算量 $O(n\ 2^n)$
- 想定解法は 2 通りあります
  - 半分全列挙 + bit DP
  - グラフの次数に基づいた枝刈り探索を行い、極大独立集合を全列挙

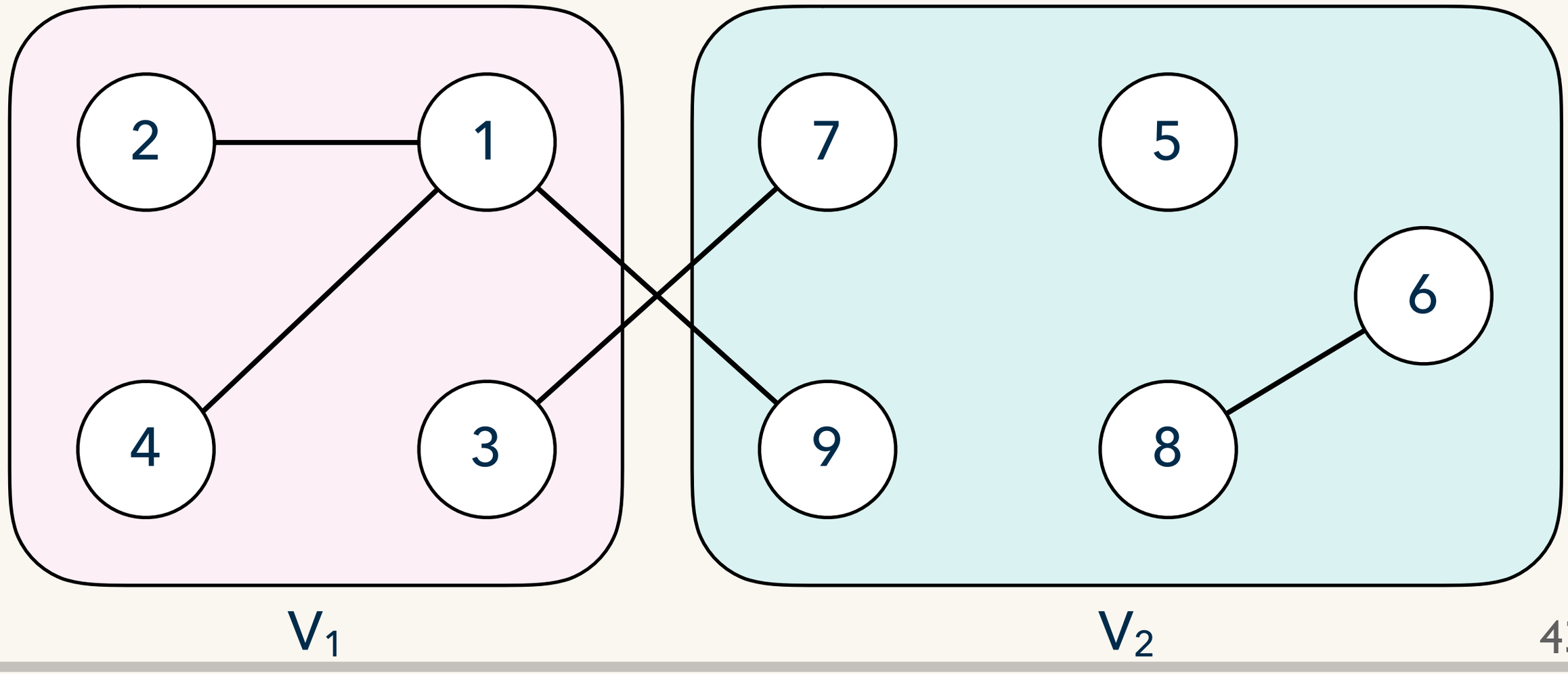
    (詳細は割愛)

解説

- 半分全列挙
  - 頂点集合 V ={1,2,...N} を2つの頂点集合 V_1 = {1,2,...N/2} ,
    V_2 = {N/2+1,N/2+2,...N} に分割する
  - **頂点集合 V_1 の部分集合 S_1 を全列挙して、S_1 に含まれる全ての頂点を使う場合の最大独立集合の計算を考える**

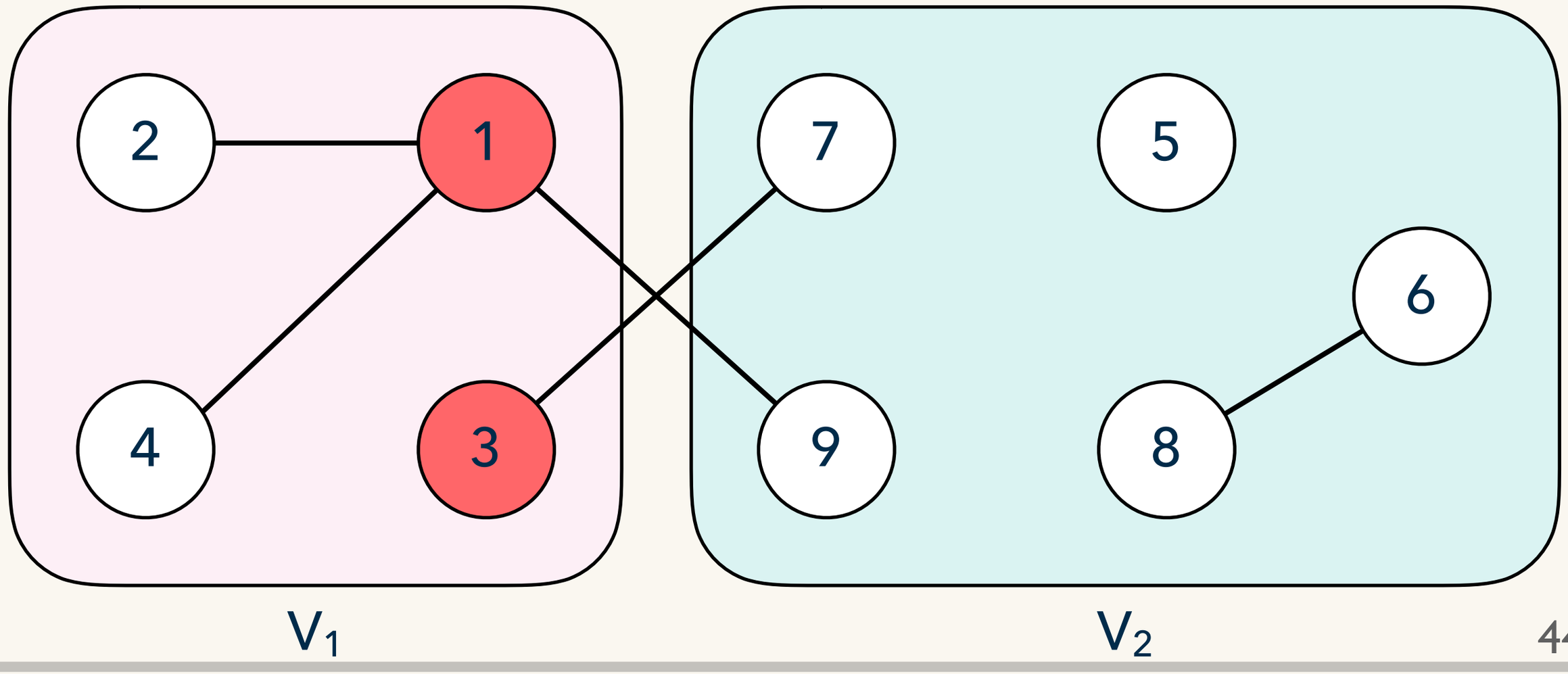
解説

- 以下の図の 9 頂点の無向グラフを例に考える
- 頂点集合 V を V_1= {1,2,3,4} V_2 ={5,6,7,8,9} のように分割する



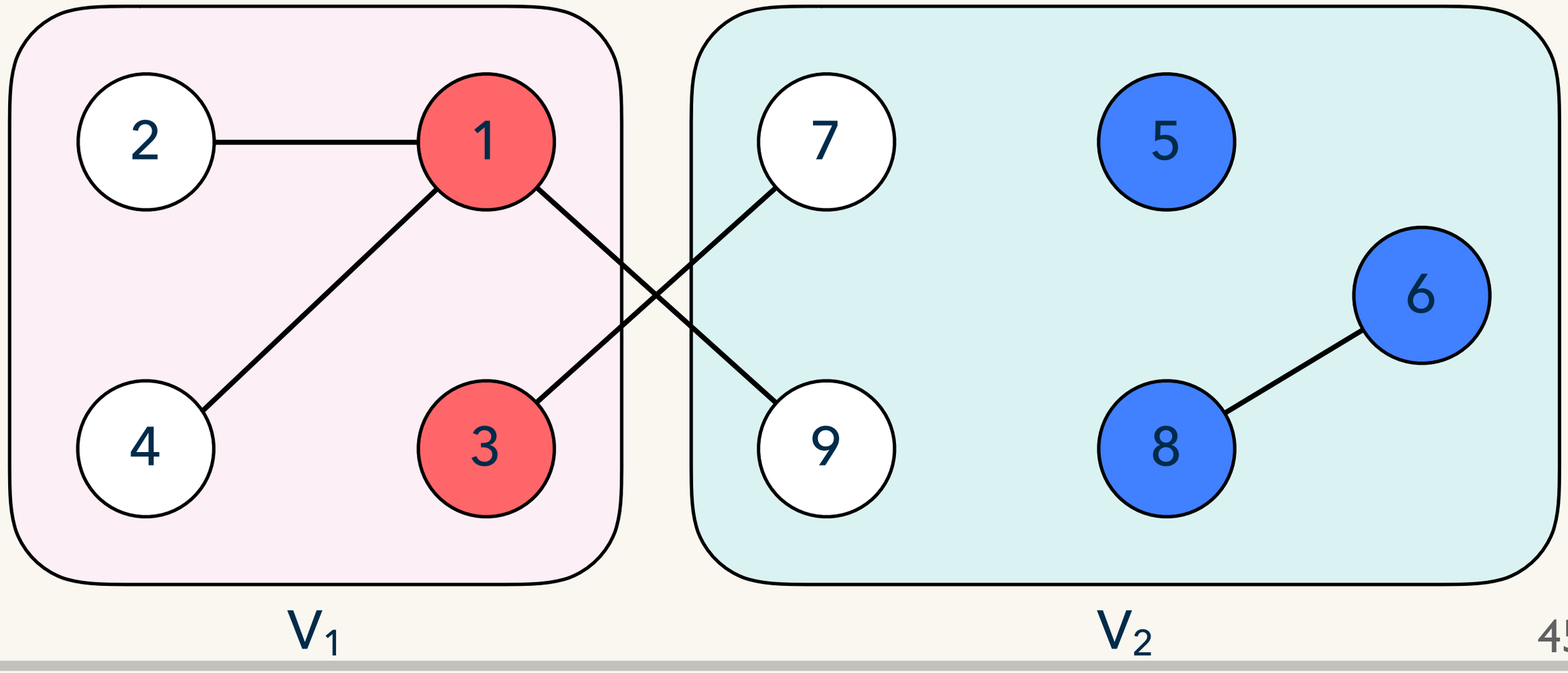
解説

- 例として、V_1 の部分集合 $S_1 = \{1,3\}$ を選ぶ
- V_1 内の辺の関係に注目すると、S_1 は独立集合の条件を満たす



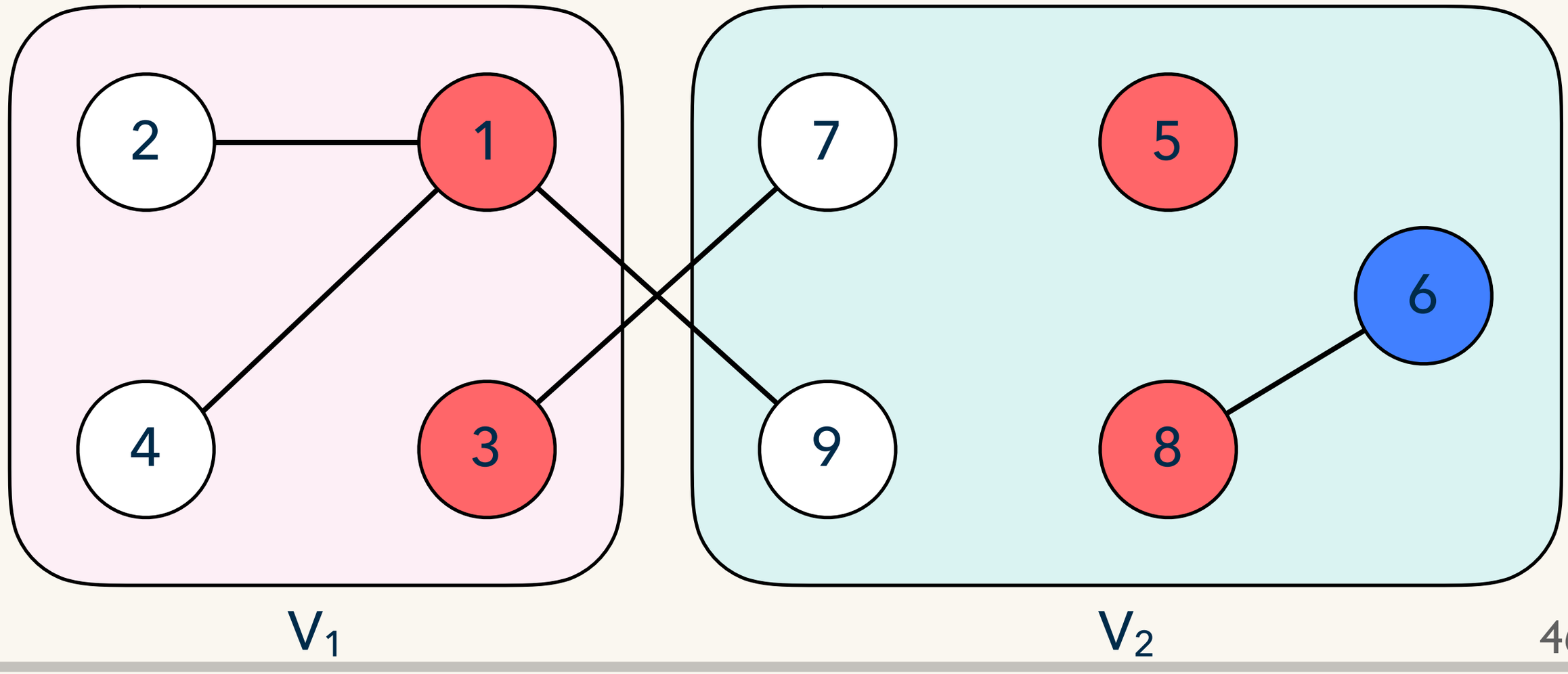
解説

- S_1 を使用する場合、さらに使える V_2 の頂点は?
- V_1 と V_2 間の辺の関係に注目すると、S_1 に含まれる頂点と辺で繋がっていない V_2 の頂点の集合 U = {5,6,8} が候補



解説

- さらに V_2 内の辺に注目して、頂点集合 U = {5,6,8} の最大独立集合 S_2 = {5,8} を求める (S_2 = {5,6} でも良い)
- S_1 を使う場合の V の最大独立集合が $S_1 \cup S_2$ で求まる



解説

- 半分全列挙の流れ
  - 頂点集合 V_1 の部分集合 S_1 を全列挙する
  - S_1 が独立集合であるか判定する
  - S_1 の頂点と辺で結ばれていない V_2 の頂点の集合 U を求める
  - V_2 の部分集合 U の最大独立集合 S_2 を求める
  - **答えは $|S_1| + |S_2|$ の最大値**
- 頂点集合 S_1 の部分集合の全列挙の時間計算量は O(2^{n/2})
- **下線で引かれている部分は bit DP で前計算をしておくことにより時間計算量 O(1) で求められる**

解説

- S_1 が独立集合であるか判定する
  - ok[S] : 頂点集合 $S \subseteq V_1$ が独立集合であるかどうか? (True/False)
  - 初期化:
    - まず、 V_1 の部分集合 S について dp[S] = True とする
    - 次に、$S = \{v, u\}$ $v, u \in V_1$ $(v, u) \notin E$ の場合 dp[S] = False とする
      - E は与えられたグラフの辺集合
  - 遷移:
    - dp[S] = False の場合のみ、$w \notin S$, $w \in V_1$ となる頂点w について

      dp[$S \cup \{w\}$] = False とする

解説

- S_1 の頂点と辺で結ばれていない V_2 の頂点の集合 U を求める
  - set[S] : 頂点集合 $S \subseteq V_1$ について、S と辺で結ばれていない V_2 の頂点集合 (ビットを利用して管理)
  - 初期化:
    - dp[{}] = V_2 (空集合に対する処理)
    - 頂点 $v \in V_1$ について dp[{v}] = { u | 辺 $(v, u) \notin E, u \in V_2$ }
  - 遷移:
    - $w \notin S$, $w \in V_1$ となる頂点w について $dp[S \cup \{w\}] = dp[S] \cap dp[\{w\}]$

解説

- V_2 の部分集合 U の最大独立集合 S_2 を求める
  - dp[S] : 頂点集合 $S \subseteq V_2$ の最大独立集合のサイズ
  - 準備: 頂点集合 $S \subseteq V_2$ について独立集合の判定を行う
    - S_1 が独立集合であるか判定する際に用いた bit DP と同じ
  - 初期化:
    - S が独立集合なら dp[S] = |S| であり、そうでない場合は dp[S] = 0
  - 遷移:
    - $w \notin S$, $w \in V_2$ となる頂点 w について dp[S ∪ {w}] = max(dp[S ∪ {w}] , dp[S]);

解説

- 半分全列挙 + bit DP の時間計算量
  - bit DP の前計算における時間計算量は $O(n\ 2^{n/2})$
  - 頂点集合 V1 の部分集合 S1 を全列挙して、S1を使う最大独立集合を求めるパートの時間計算量 $O(2^{n/2})$
  - **したがって、時間計算量は $O(n\ 2^{n/2})$ となる**

Problem H
「Union Sets」

問題概要

- N 個の集合 {1}, {2}, ..., {N} があります
- 2つの要素を選んで、集合を併合する操作が M 回行われます
- Q 個の質問クエリに答えてください
  - 何回目の操作後に2つの要素が同じ集合に属すのかを出力する
  - 同じ集合に属さない場合は -1 を出力
- 制約
  - $1 \leqq N \leqq 10^5$
  - $1 \leqq M \leqq 10^5$
  - $1 \leqq Q \leqq 10^5$

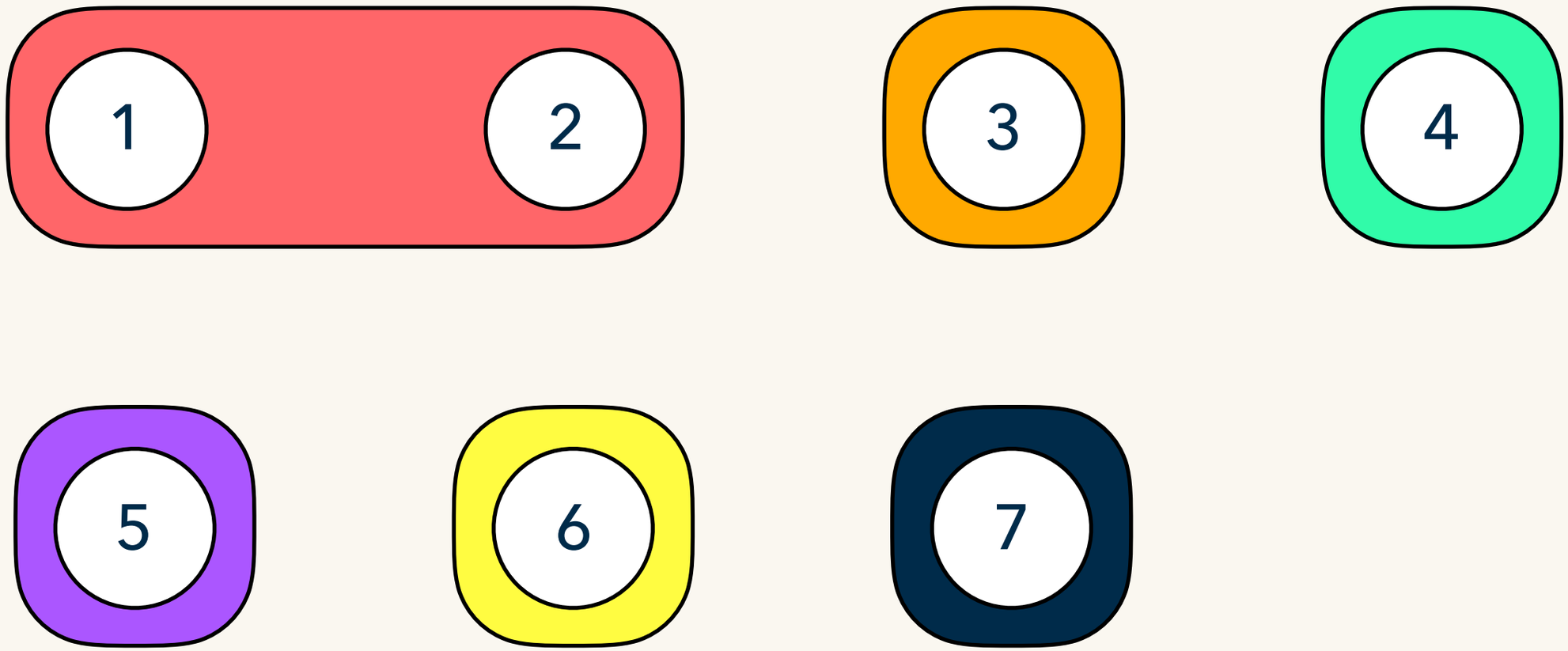
解説

- サンプル1

解説

- 1回目の操作: 要素1を持つ集合と要素2を持つ集合を併合

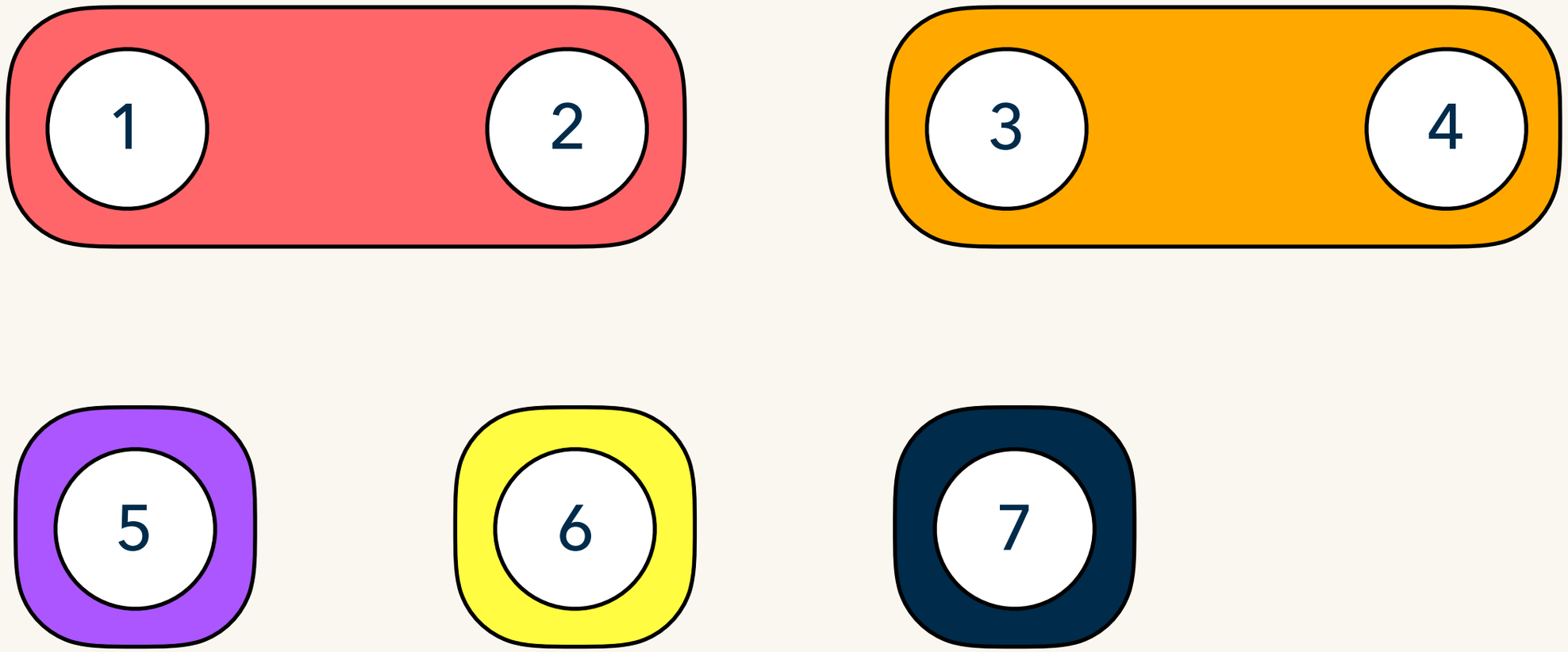
  → 要素1と要素2が同じ集合となる



解説

- 2回目の操作: 要素3を持つ集合と要素4を持つ集合を併合

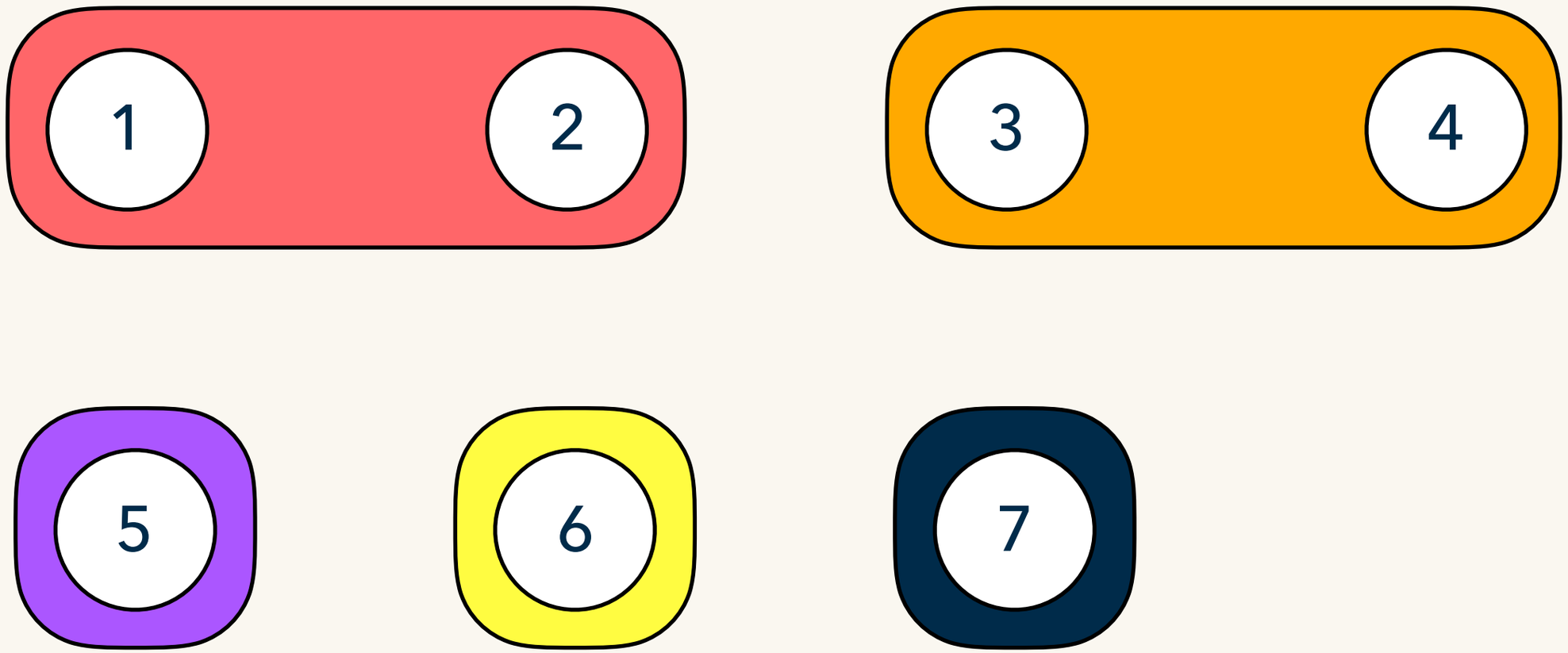
  → 要素3と要素4が同じ集合となる



解説

- 3回目の操作: 要素2を持つ集合と要素1を持つ集合を併合

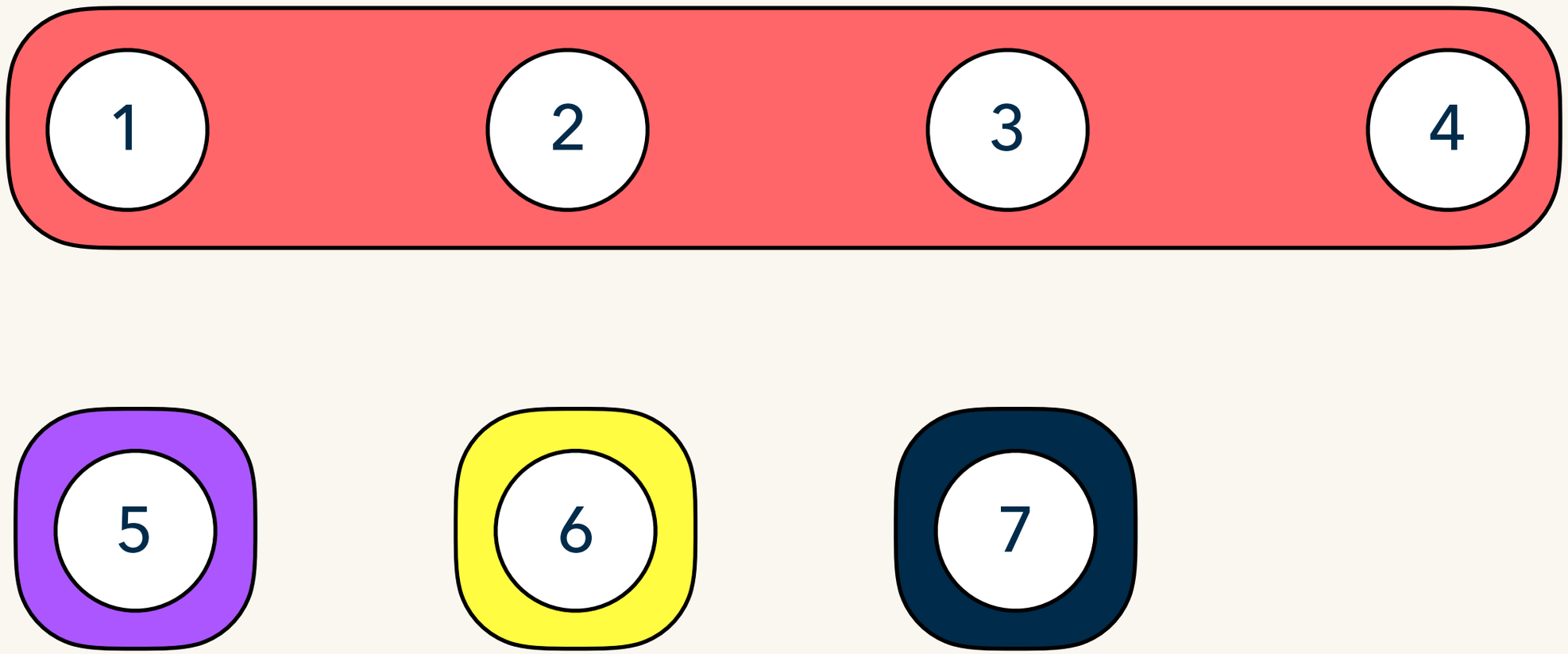
  → 既に同じ集合なので変化なし



解説

- 4回目の操作: 要素2を持つ集合と要素3を持つ集合を併合

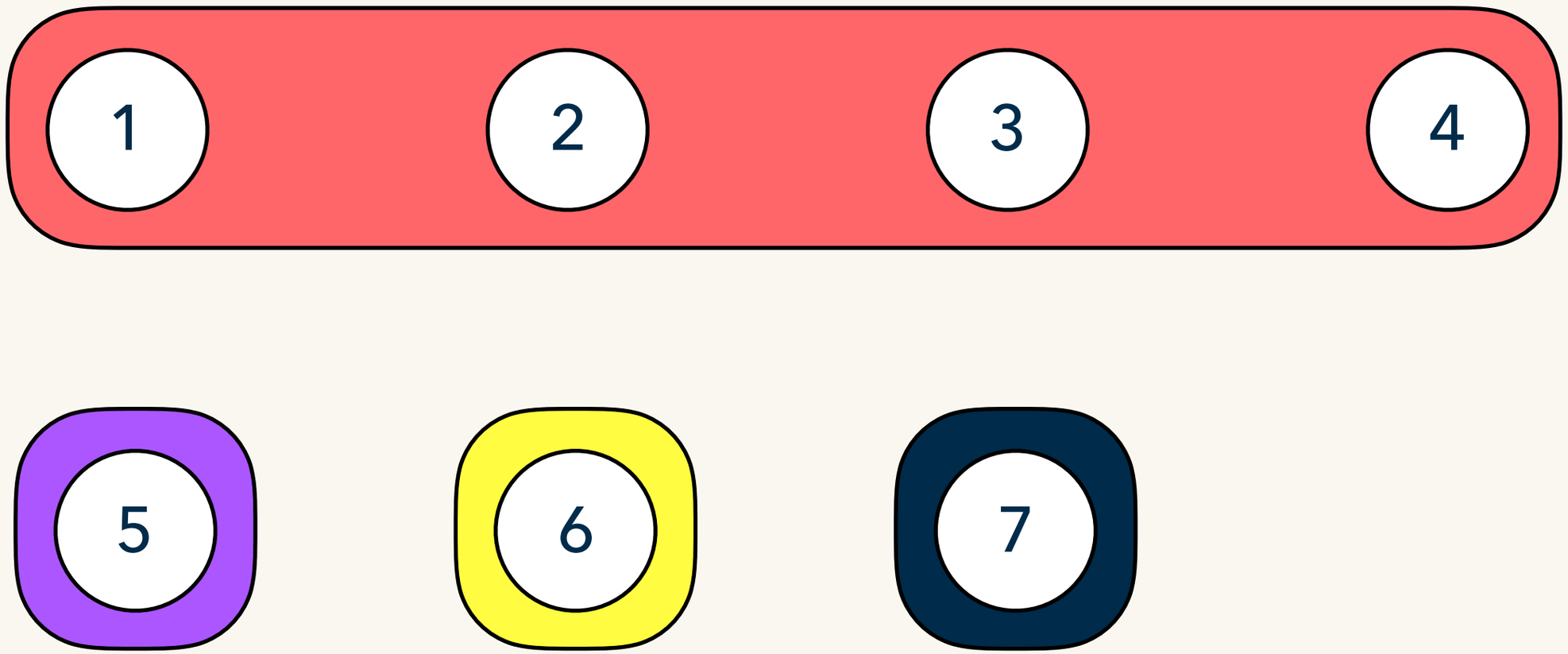
  → 要素1、要素2、要素3と要素4が同じ集合となる



解説

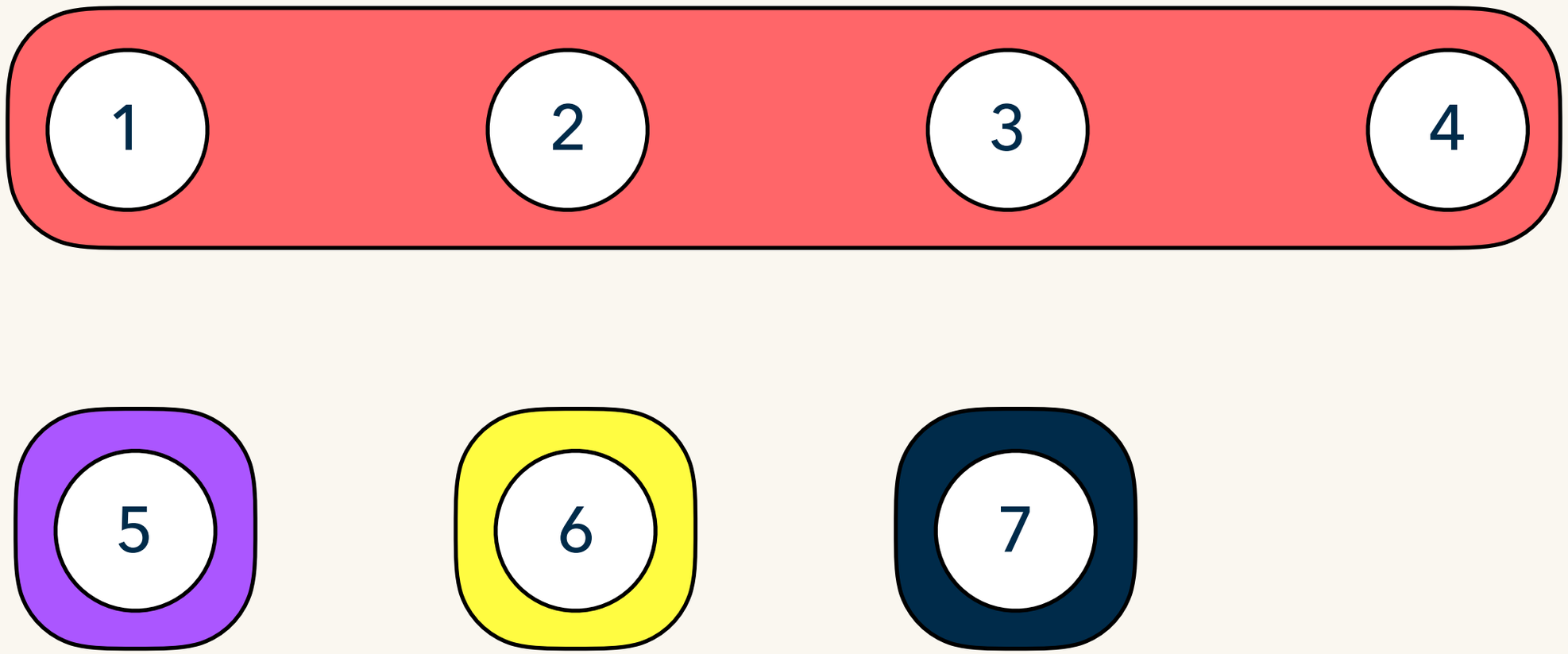
- 5回目の操作: 要素4を持つ集合と要素1を持つ集合を併合

  → 既に同じ集合なので変化なし



解説

- 要素 1 と要素 2 はいつ同じ集合に？ 1回目の操作の後なので 1
- 要素 6 と要素 7 はいつ同じ集合に？ 併合されていないので -1



解説

- 想定 TLE 解法
  - Union-Find で集合の関係を管理する
  - クエリ毎に集合の併合操作を行い答えを求める
  - 時間計算量 O(N + QMa(N)) となるので間に合わない
    - a(N) は アッカーマン関数 A(N,N) の逆関数
- 想定解法は 2 通りあります
  - 解法1: 集合の併合関係を表す森を構築を行い、最小共通祖先(LCA) でクエリを答える
  - 解法2: Union-Find を利用した並列二分探索 (詳細は割愛)

    時間計算量 O((N + Ma(N) + Q) log M)

解説

- 集合の併合関係を表す森を構築
  - **1回の集合の併合操作は、必ず2つの要素を選んで行う**

    → 要素を頂点、併合操作を辺に対応させて、集合の併合関係を無向グラフで表せる
  - この場合，辺で繋がっている要素は同じ集合に属する

    → 各連結成分が1つの集合に対応する

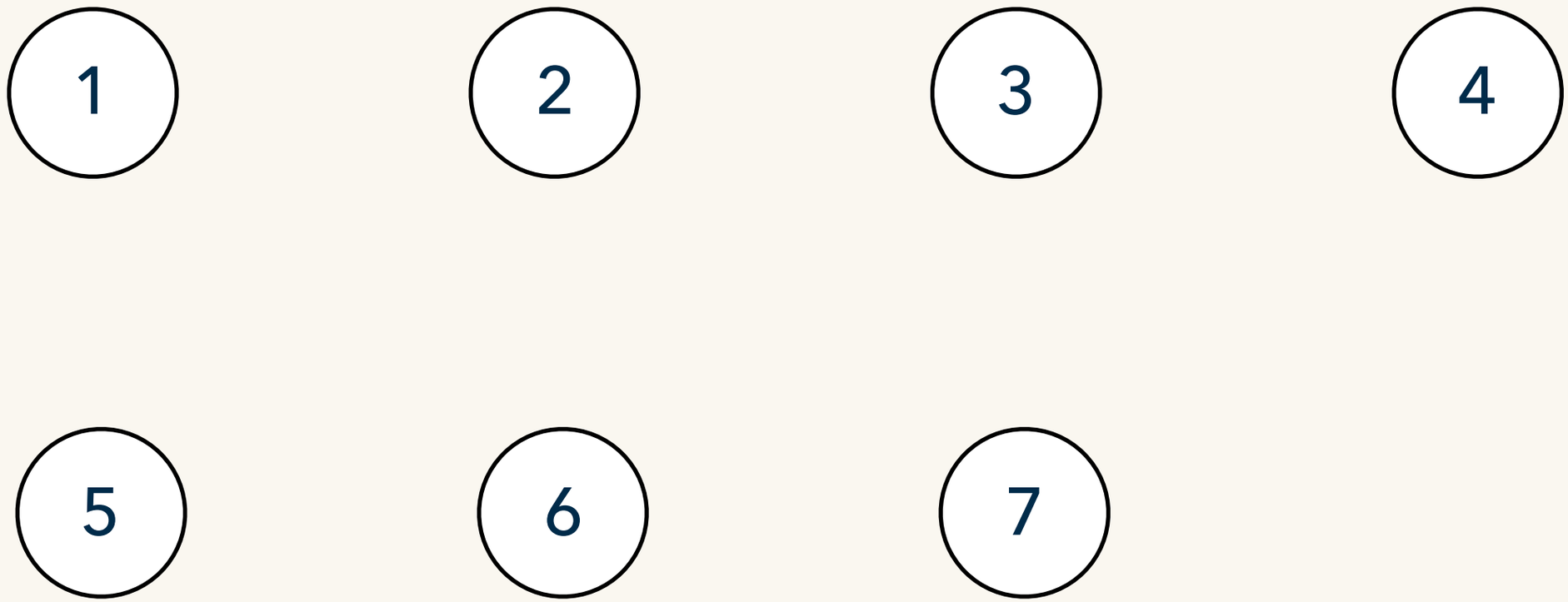
解説

- 集合の併合関係を表す森を構築
  - **集合の併合を繰り返すと、どのようなグラフが得られるか?**
  - 要素 A を持つ集合と要素 B を持つ集合が異なる場合:
    - グラフにおいて、要素 A を含む連結成分と要素 B を持つ連結成分が異なる
    - 要素 A と 要素 B を辺で繋ぐことで，要素 A と要素 Bが同じ連結成分に属するので集合の併合を実現できる
  - 要素 A を持つ集合と要素 B を持つ集合が同じ場合:
    - グラフにおいて、要素 A を含む連結成分と要素 B を持つ連結成分が同じ
    - この場合，要素 A と 要素 B を辺でつながないが、仮に繋ぐと閉路ができる
  - **閉路のないグラフ = 森ができる**

解説

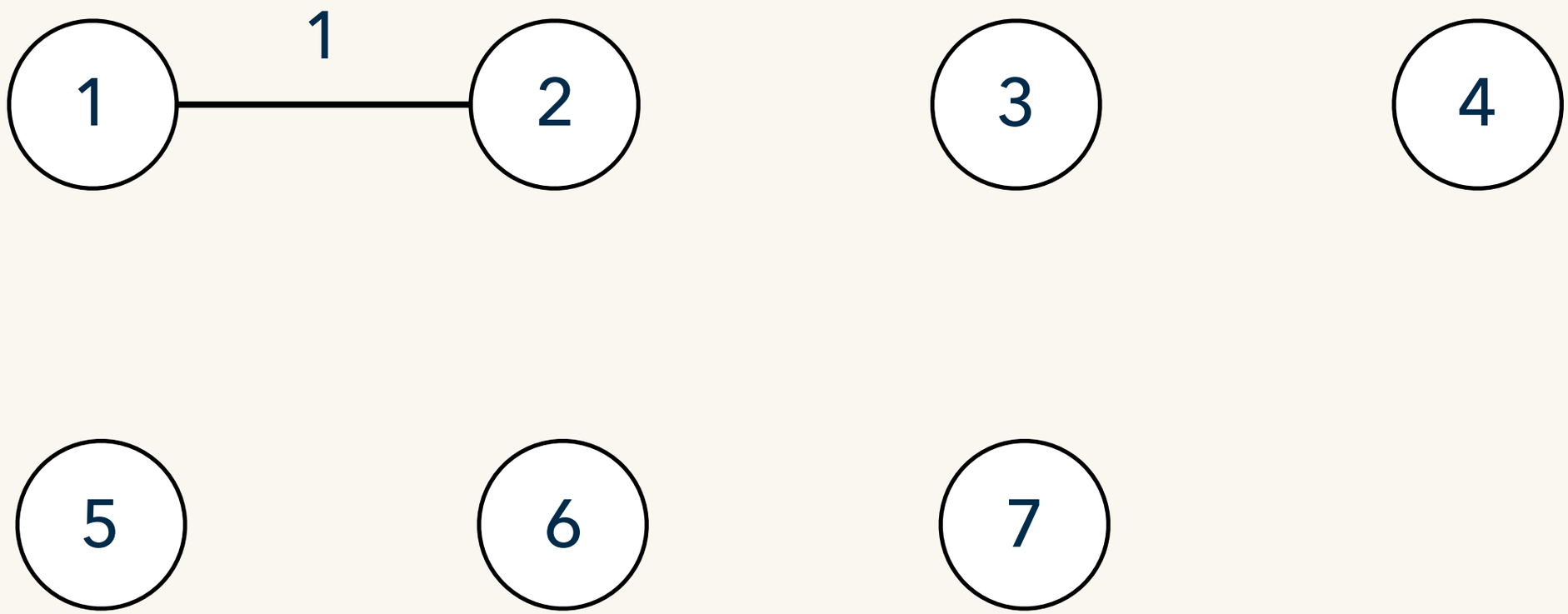
- サンプル1 を元に集合の併合関係を表す森を構築してみる

  さらに、k 回目の操作で要素を繋ぐ辺の重みを k と割り当てる



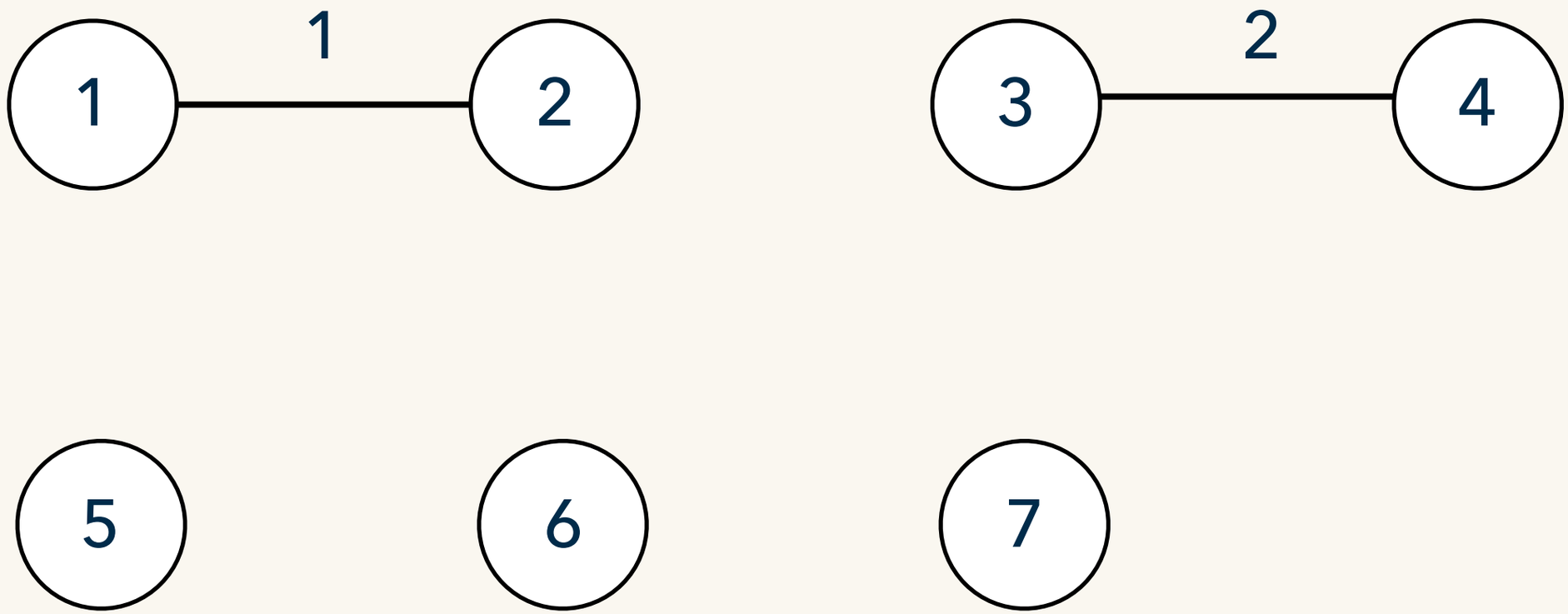
解説

- 1回目の操作: 要素1を持つ集合と要素2を持つ集合を併合



解説

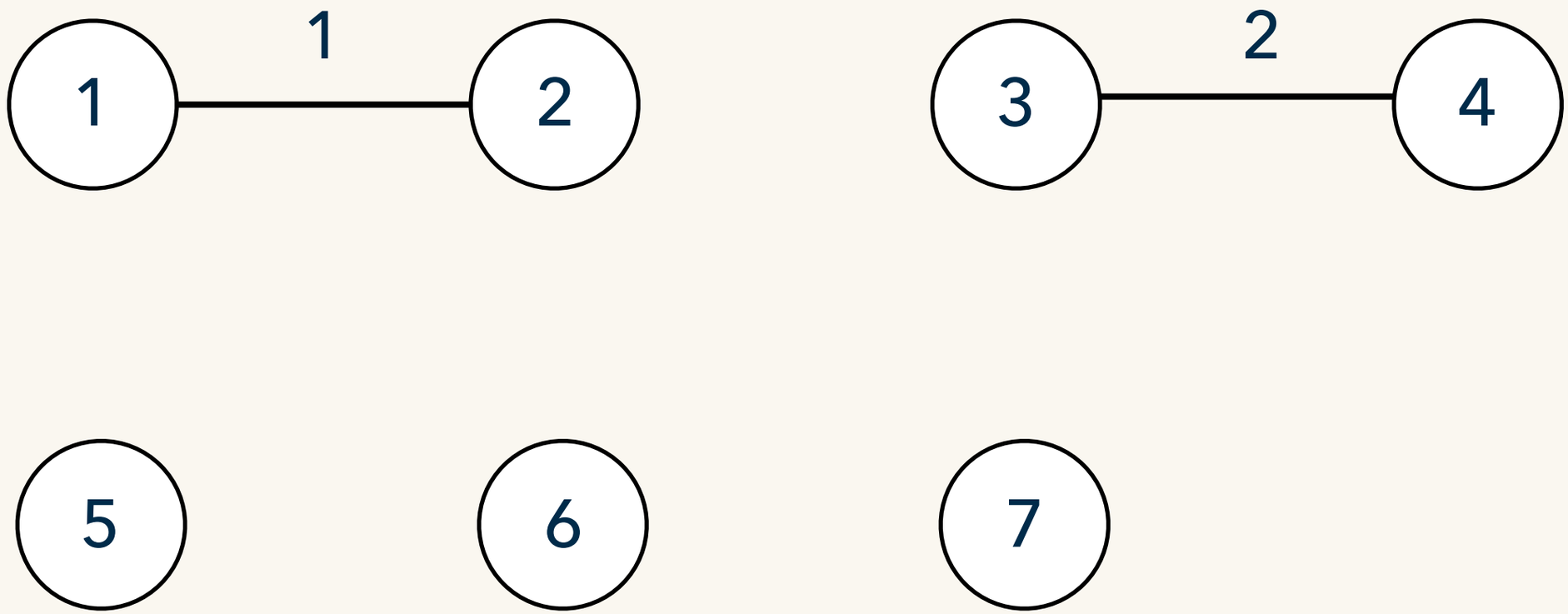
- 2回目の操作: 要素3を持つ集合と要素4を持つ集合を併合



解説

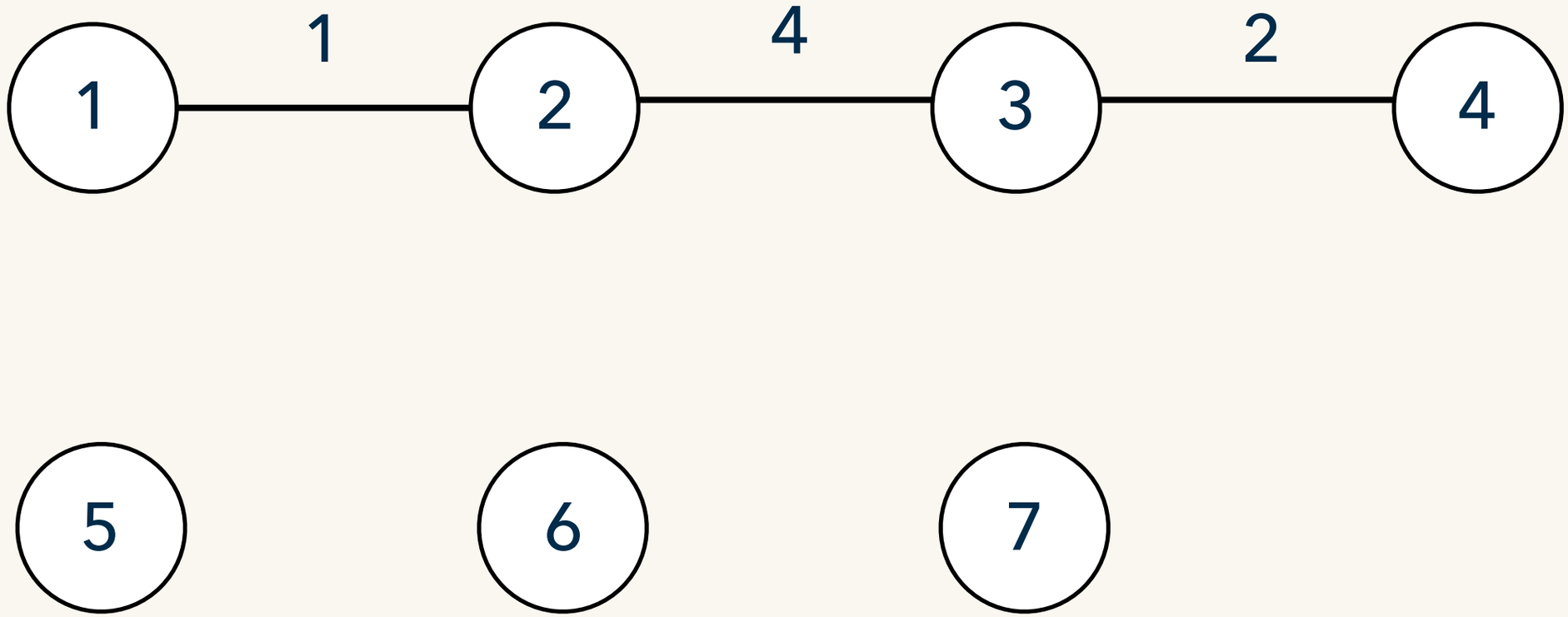
- 3回目の操作: 要素2を持つ集合と要素1を持つ集合を併合

  → 既に同じ集合なので辺で繋がない



解説

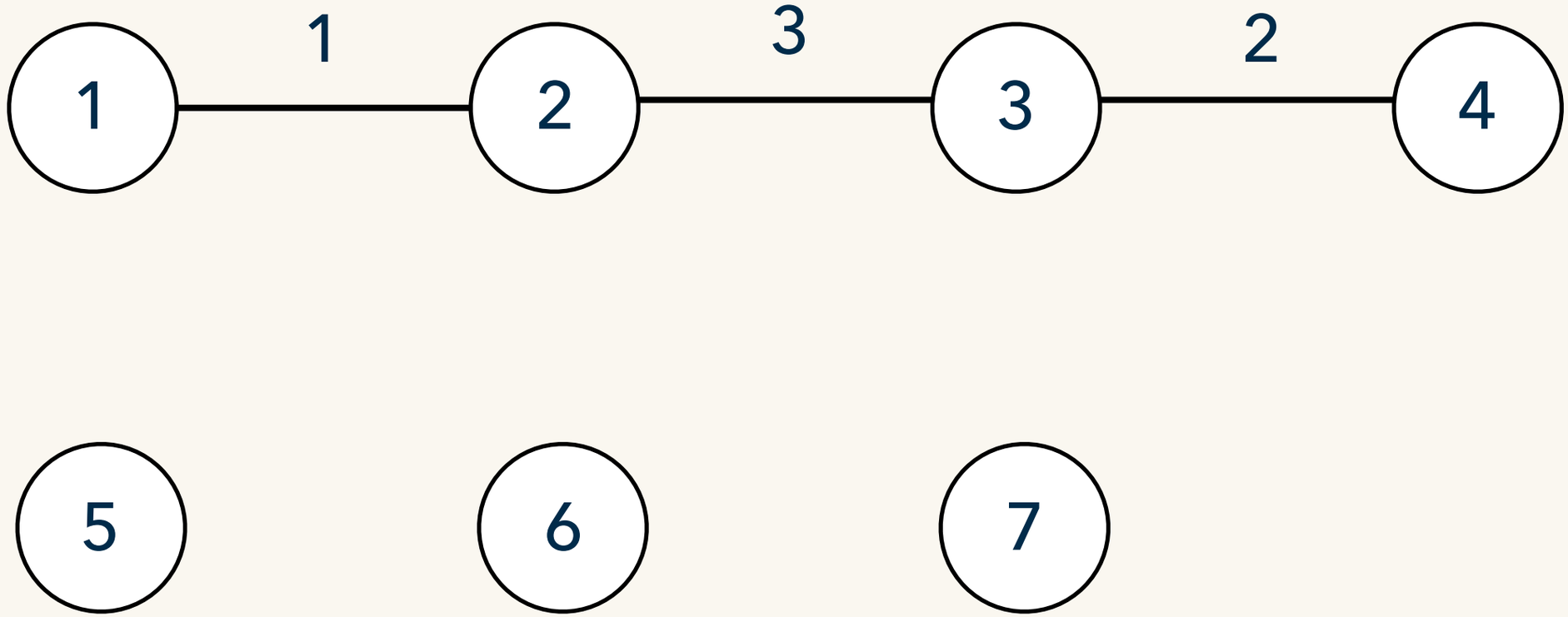
- 4回目の操作: 要素2を持つ集合と要素3を持つ集合を併合



解説

- 5回目の操作: 要素4を持つ集合と要素1を持つ集合を併合

  → 既に同じ集合なので辺で繋がない

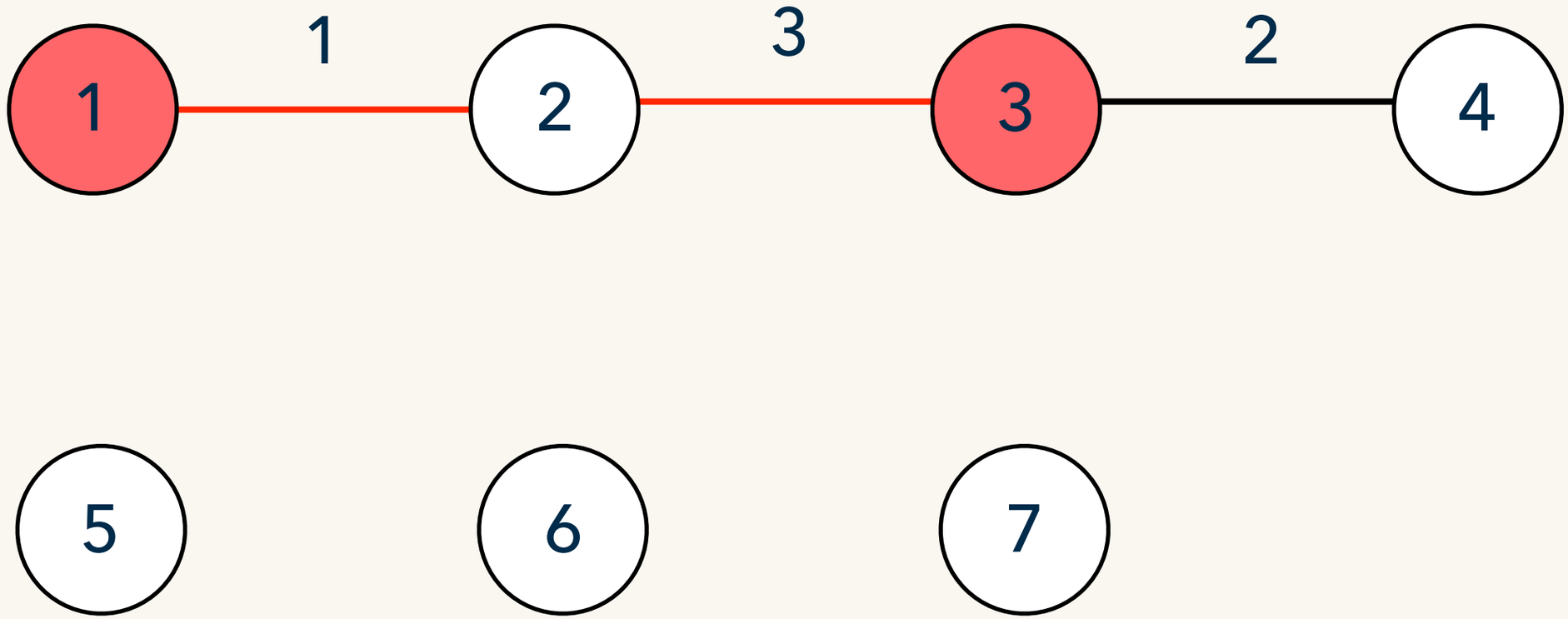


解説

- **2つの要素が同じ集合に属する場合には?**

  → パス上の重みの最大値が答えに対応する

- 例: 要素 1 と要素 3 はいつ同じ集合に?　max(1,3) = 3

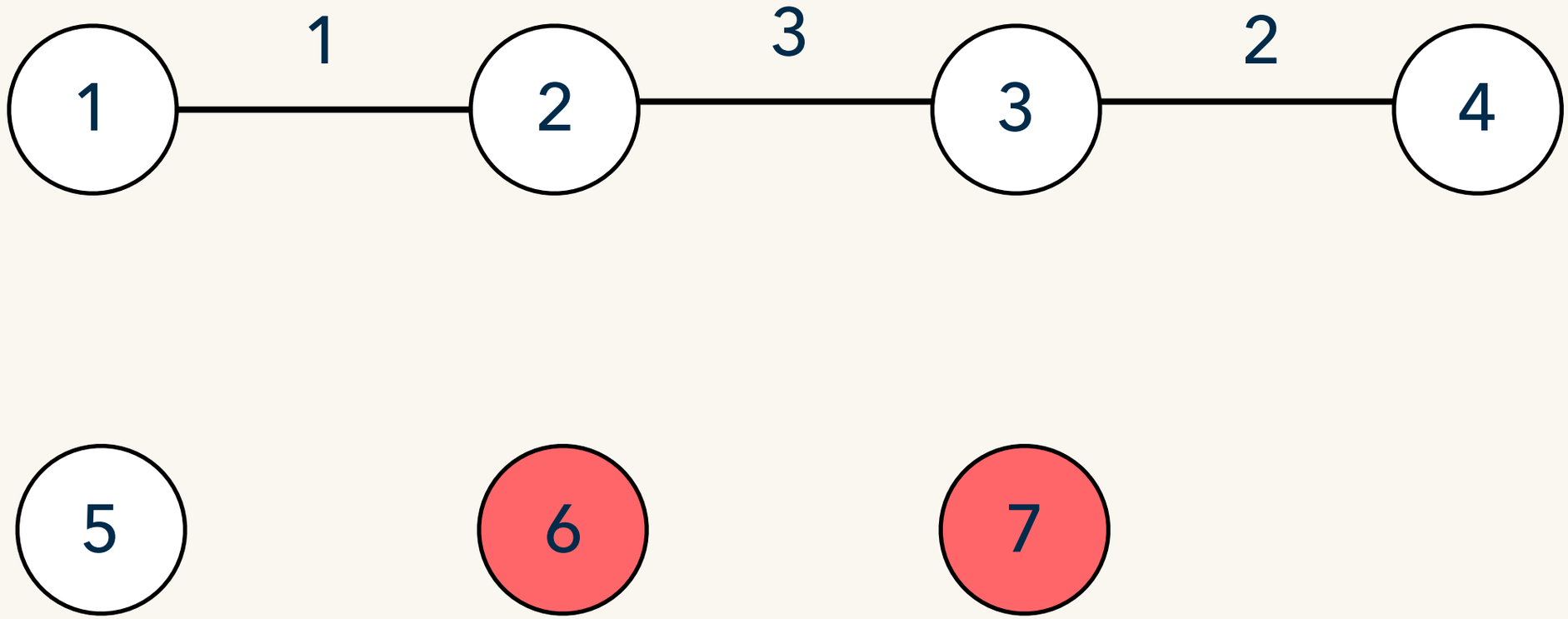


解説

- **2つの要素が同じ集合に属する場合には?**

  → パス上の重みの最大値が答えに対応する

- 例: 要素 6 と要素 7 はいつ同じ集合に? 非連結なので -1



解説

- 解法1の全体の流れ
  - 要素を頂点、併合関係を辺として重み付き無向森を作成
    - ✦ Union-Find を利用したクラスカル法で作成可能
  - クエリで2つの要素が与えられるので、対応する 2 頂点間のパス上の重みの最大値を高速に求める
    - ✦ ダブリングを利用した LCA で 1クエリあたり O(log N) で処理可能
- 時間計算量:
  - 重み付き無向森の作成: O (Ma(N))
  - LCA の初期化: O(N log N) 、1クエリあたり O(log N)
  - **全て合わせて、O(Ma(N) + (N+Q) log N)**

END