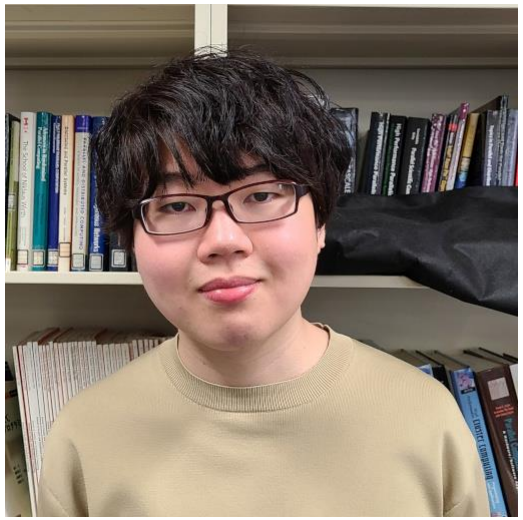
四條畷プログラミング学習会2024 実践編

大阪大学
青山 昂生

自己紹介



- 名前: 青山 昂生
  - X: @small_onions
  - AtCoderなど: kotamanegi
- 所属
  - 学生: 大阪大学 伊野研究室 博士1年
  - アルバイト: 株式会社ALGO ARTIS

- 趣味
  - 競技プログラミング

出題された課題に対し、
より早く正確に課題を解く
プログラムを書く競技

目次

- 中級編
  - 配列とfor文
  - 計算量の話
- 上級編
  - グラフ
  - 動的計画法
- 裏技編
  - 提出デバッグ

一次予選の問4について

- 多くの問題で**集計**を実装する必要
  - 集計が実装できれば大丈夫

- 過去問を全部解ければ怖いものなし？
  - AOJ/AtCoder-JOI https://joi.goodbaton.com/
  - 難易度3まで解ければOK

問題: 希少な数

問題文

長さ N の整数列 $A = (A_1, A_2, \ldots, A_N)$ が与えられる.

A に出現する整数のうち，出現回数が最小である整数を出力せよ．ただし，そのような整数が複数考えられる場合は，考えられる整数のうち最も小さい整数を出力せよ.

D - 希少な数, JOI 2021/2022 一次予選 (第2回), https://atcoder.jp/contests/joi2022yo1b/tasks/joi2022_yo1b_d

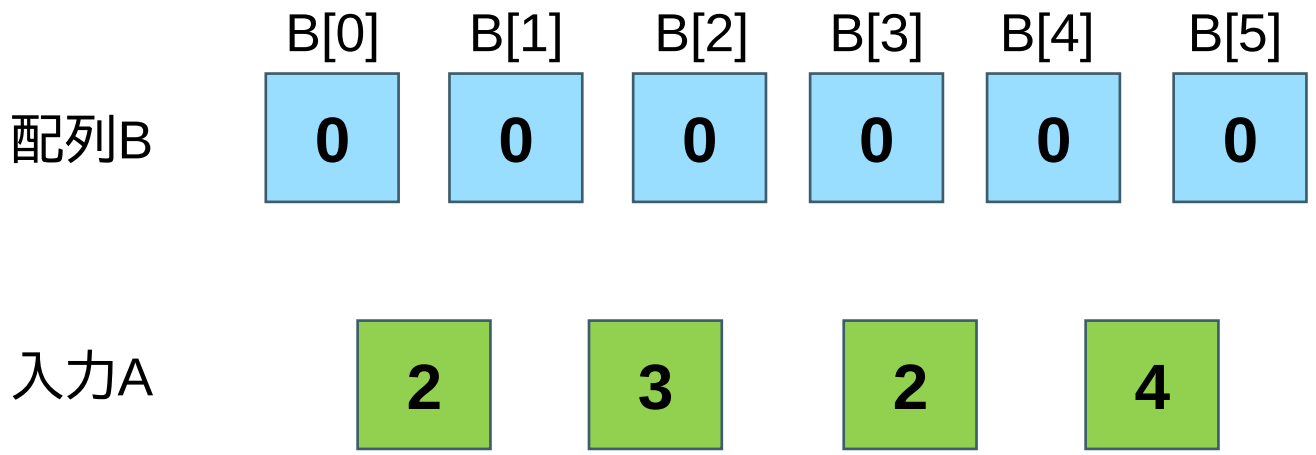
制約

- $1 \leqq N \leqq 100$.
- $1 \leqq A_i \leqq 2\,000$ $(1 \leqq i \leqq N)$.

出現回数の集計

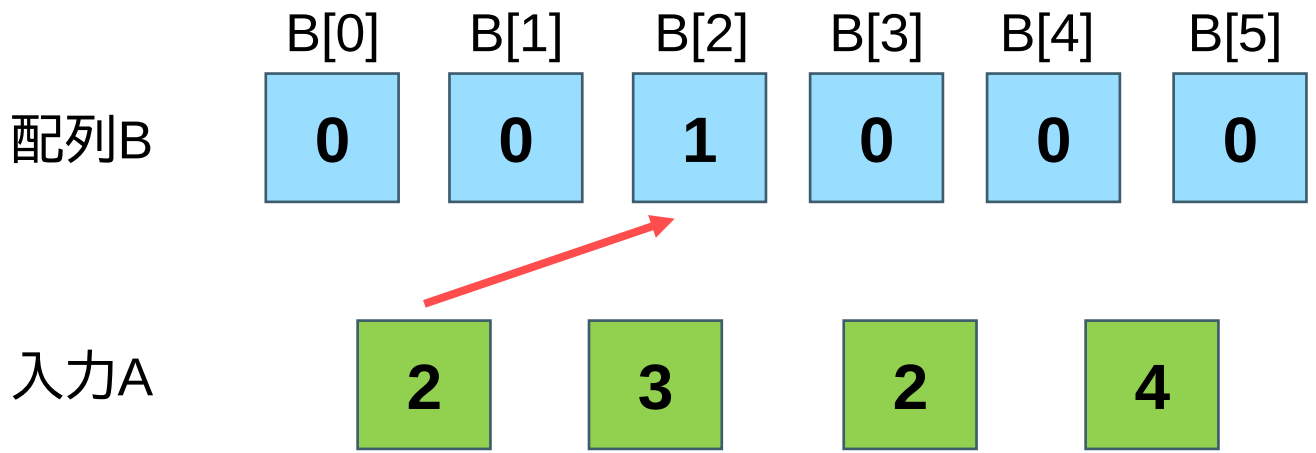
集計の方法

- 集計用の配列Bを用意する
  - Python: B = [0] * 2001
  - C++: int B[2001] = {};
- 集計していく



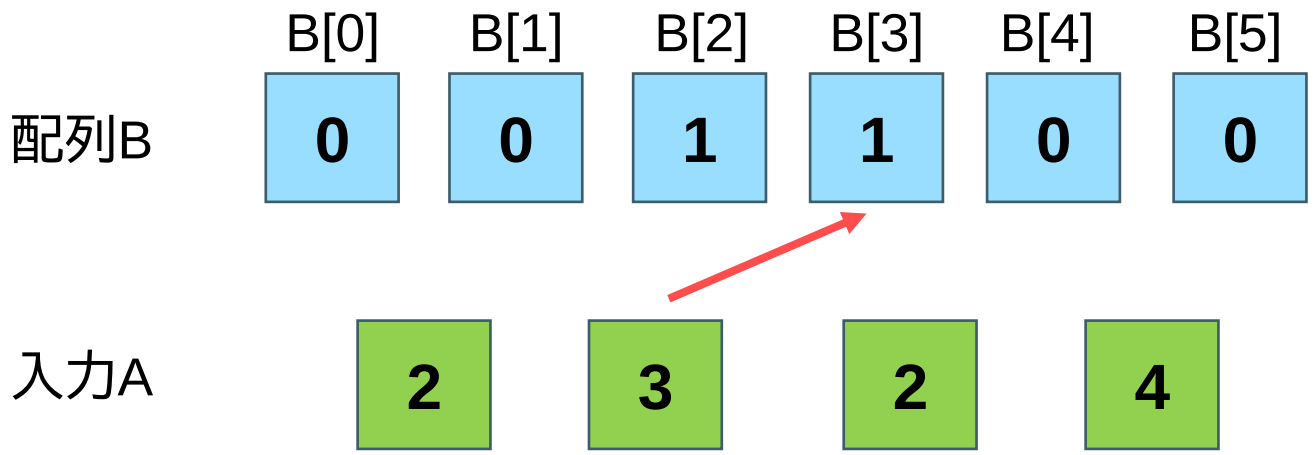
集計の方法

- 集計用の配列Bを用意する
  - Python: B = [0] * 2001
  - C++: int B[2001] = {};
- 集計していく



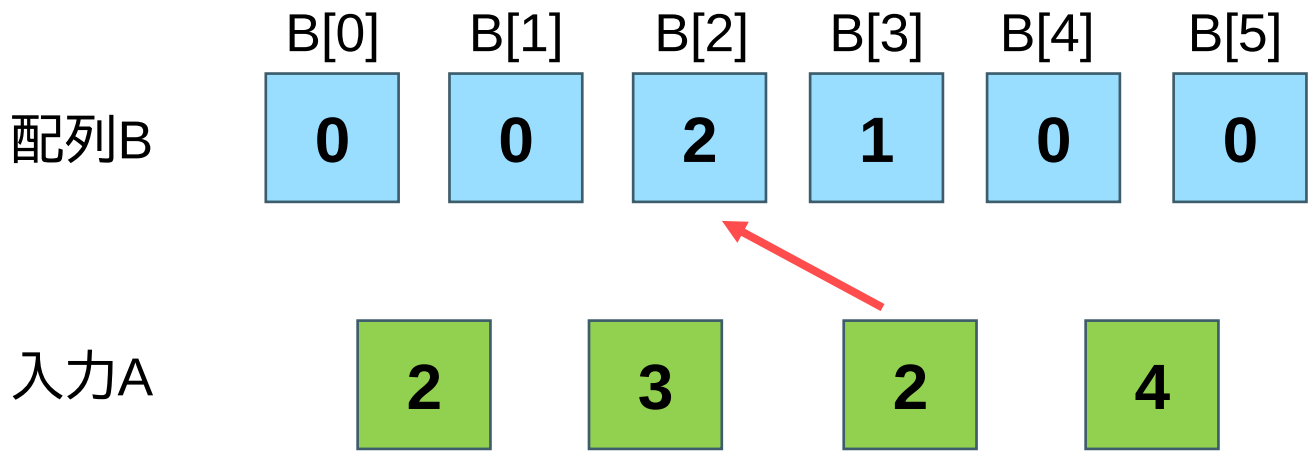
集計の方法

- 集計用の配列Bを用意する
  - Python: B = [0] * 2001
  - C++: int B[2001] = {};
- 集計していく



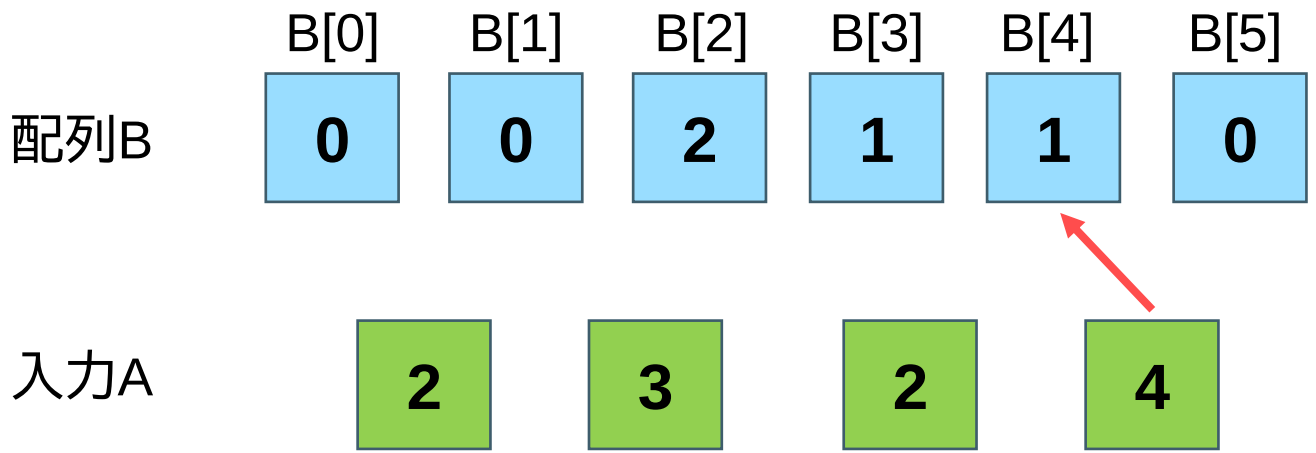
集計の方法

- 集計用の配列Bを用意する
  - Python: B = [0] * 2001
  - C++: int B[2001] = {};
- 集計していく



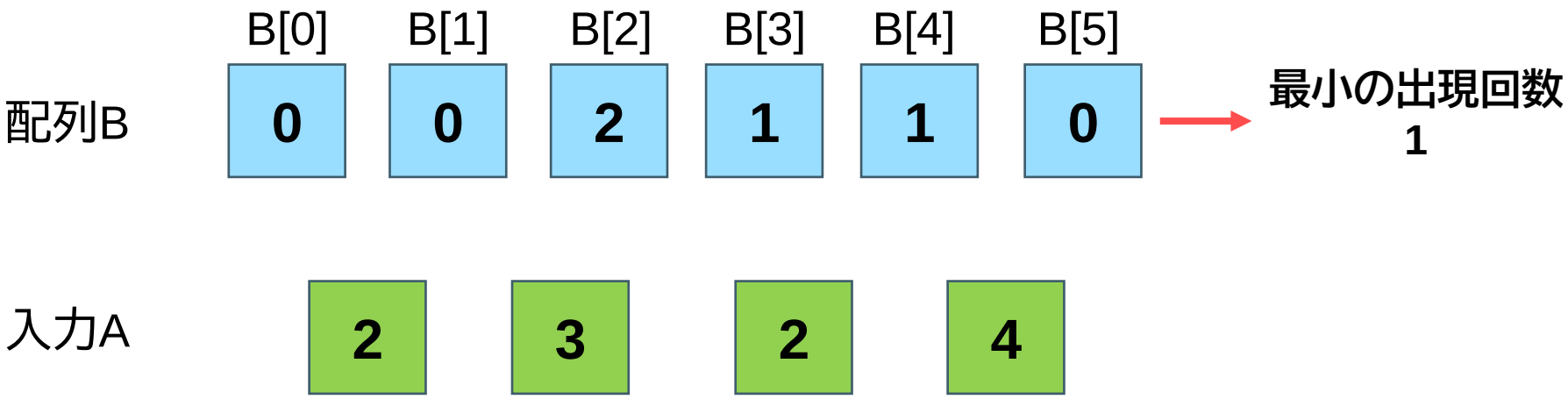
集計の方法

- 集計用の配列Bを用意する
  - Python: B = [0] * 2001
  - C++: int B[2001] = {};
- 集計していく



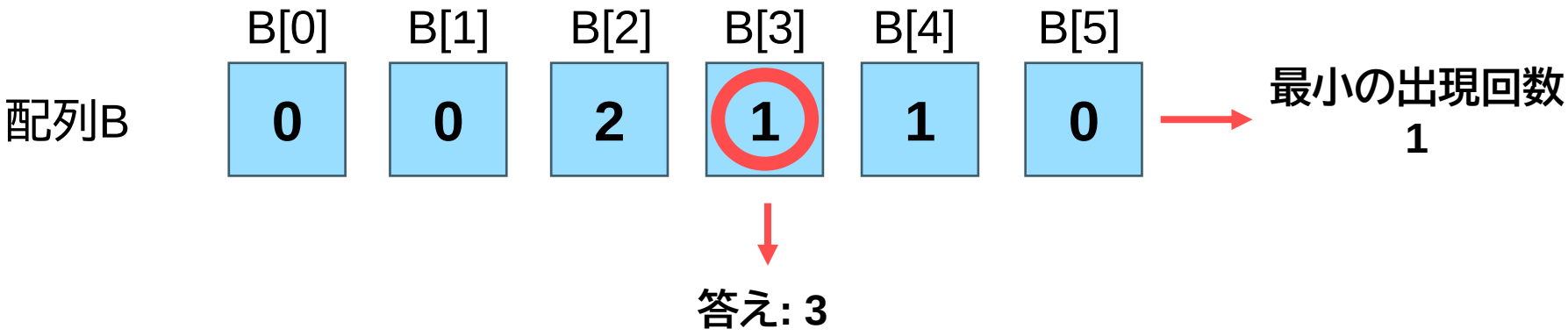
集計の方法

- 集計用の配列Bを用意する
  - Python: B = [0] * 2001
  - C++: int B[2001] = {};
- 集計していく
- 1回以上出現している数字のうち、最小の出現回数を求める



集計の方法

- 集計用の配列Bを用意する
  - Python: B = [0] * 2001
  - C++: int B[2001] = {};
- 集計していく
- 1回以上出現している数字のうち、最小の出現回数を求める
- 最小の出現回数の数字のうち最小の数字を求める



ほかの問4について

- （基本的に）書いてあることを実装すればよい
  - プログラミング経験を積むしかない
  - 頑張ってください

- 一次予選では基本的に計算量を問う問題は出題されない
  - 実行時間が長いなら繰り返し文の条件間違いなどを疑う

プログラミングに慣れよう

演習問題（すべて問4）

- 希少な数 https://atcoder.jp/contests/joi2022yo1b/tasks/joi2022_yo1b_d
- 二人三脚 https://atcoder.jp/contests/joi2023yo1a/tasks/joi2023_yo1a_d
- 現れている数字 https://atcoder.jp/contests/joi2024yo1a/tasks/joi2024_yo1a_d
  - 少し言い換えが必要
- 箱と鍵 https://atcoder.jp/contests/joi2022yo1a/tasks/joi2022_yo1a_d
  - 条件付き集計
- マラソン大会 https://atcoder.jp/contests/joi2023yo1c/tasks/joi2023_yo1c_d
  - 集計後の作業が若干面倒

目次

- 中級編
  - 配列とfor文
  - 計算量の話
- 上級編
  - グラフ
  - 動的計画法
- 裏技編
  - 提出デバッグ

計算量の概念

- 計算量: アルゴリズムの計算効率を評価するための指標
- 2つの計算量
  - ☆ 時間計算量: 実行時間の評価
  - 空間計算量: 使用メモリ量の評価

- 1回の計算として数えるもの
  - 四則演算（+, -, *, /）
  - 比較演算子（<, >, == など）
  - 変数の読み出し/書き込み

```
print(a + b)
```

```
if a == b:
```

```
c = b
```

オーダ記法

- n が十分大きいときの挙動を比較するための記法
  - $\mathcal{O}(n^2)$：n^2 に比例して増える関数
    - 例: $f(n) = 2n^2 + n + 1, 3n^2 - 10$
  - $\mathcal{O}(n)$：n に比例して増える関数
    - 例: $f(n) = n + 1, 3n - 5$

- 定数倍は区別しない（ことが多い）
  - 表現揺れはよくあること
  - 詳しくは大学数学で

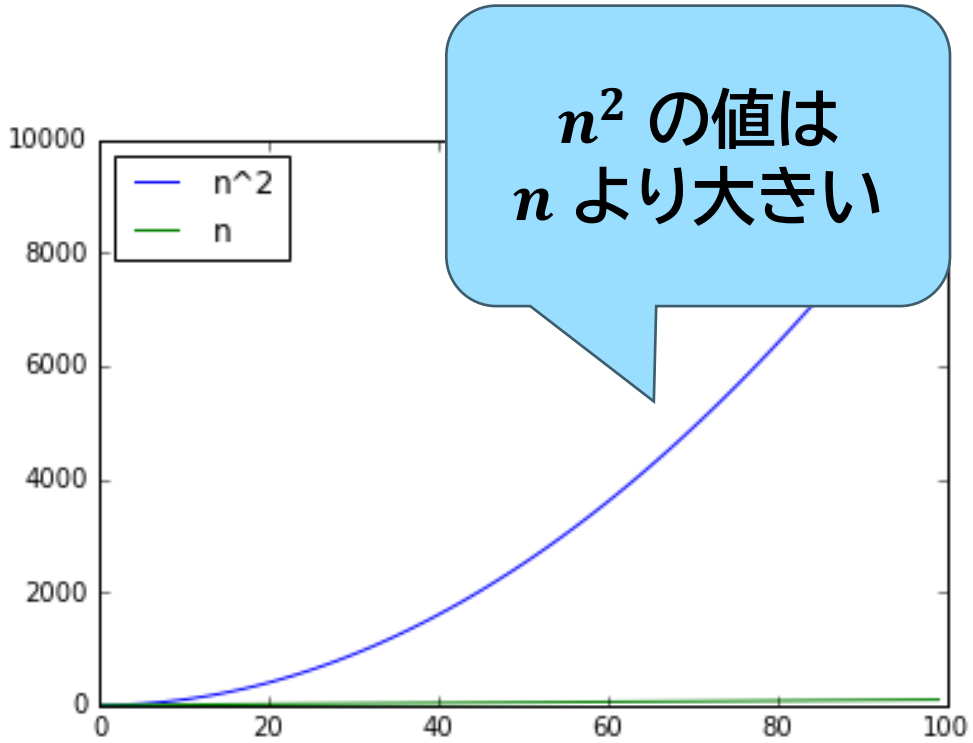


図: 計算量オーダーの求め方を総整理！ ～ どこから log が出て来るか ～, @drken, https://qiita.com/drken/items/872ebc3a2b5caaa4a0d0#1-2-%E3%83%A9%E3%83%B3%E3%83%80%E3%82%A6%E3%81%AE-o-%E8%A8%98%E6%B3%95

計算量を求めてみる

$\mathcal{O}(nm)$

n回 { m回 {

```
n = int(input())
m = int(input())

answer = 0
for i in range(n):
    for j in range(m):
        answer += i*j

print(answer)
```

$\mathcal{O}(n^2)$

n回 { i回 {

$i \leq n$

```
n = int(input())

answer = 0
for i in range(n):
    for j in range(i):
        answer += i*j

print(answer)
```

計算量を削減するためには

- for文を外に出す、消す

- 競プロで時間制限に間に合う計算回数は10^8回前後
  - 実際には様々な要因が働く
  - 怪しいときは実装してみるのも手

```python
n = int(input())
m = int(input())
t = int(input())

answer = 0
for i in range(n):
    for j in range(m):
        temp = 0
        for k in range(t):
            temp += k
        answer += i * j * temp

print(answer)
```

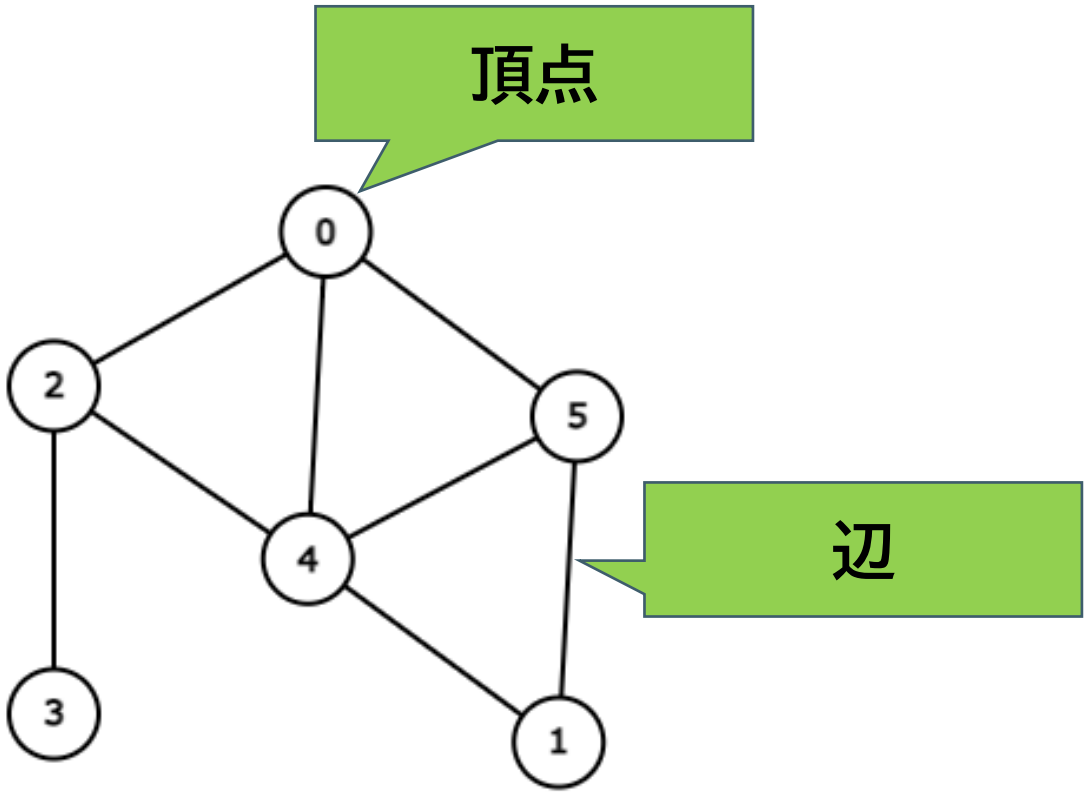
```python
n = int(input())
m = int(input())
t = int(input())

answer = 0

temp = 0
for k in range(t):
    temp += k

for i in range(n):
    for j in range(m):
        answer += i * j * temp

print(answer)
```

for文を分離

目次

グラフ（グラフ理論）

- 関係性を表現するための道具
  - もの、こと: 頂点
  - 関係性: 辺



解釈①: 路線図

- 頂点: 駅
- 辺: 直接移動できる駅

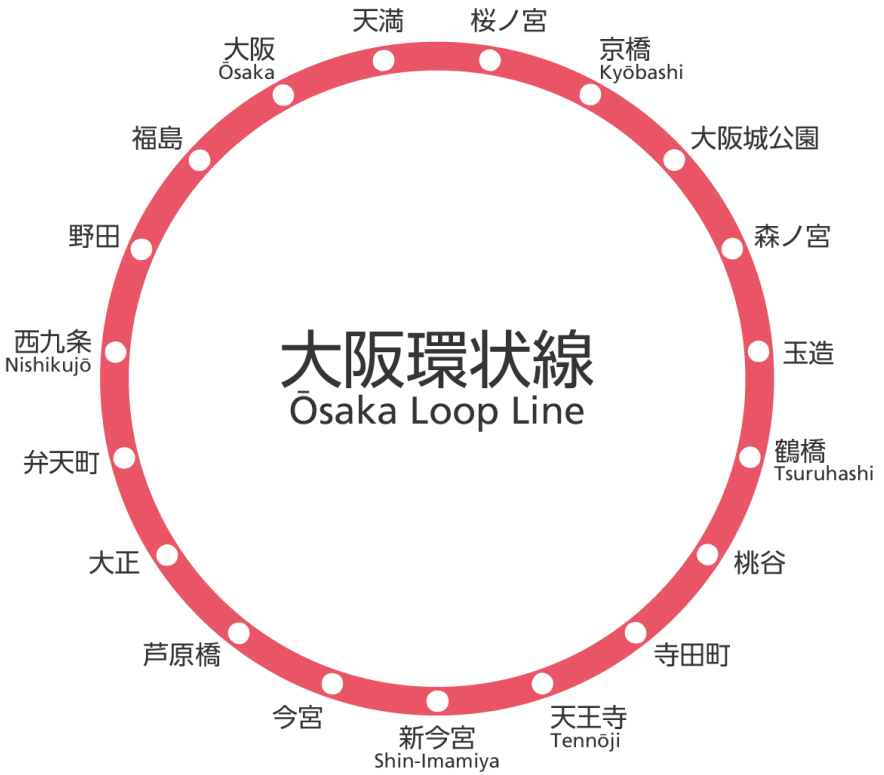


図: https://rp-tj.blogspot.com/2021/09/osakaloopline-map.html

解釈②: 人間関係

- 頂点 = 人間、辺 = 友達関係の**ソーシャルネットワーク**

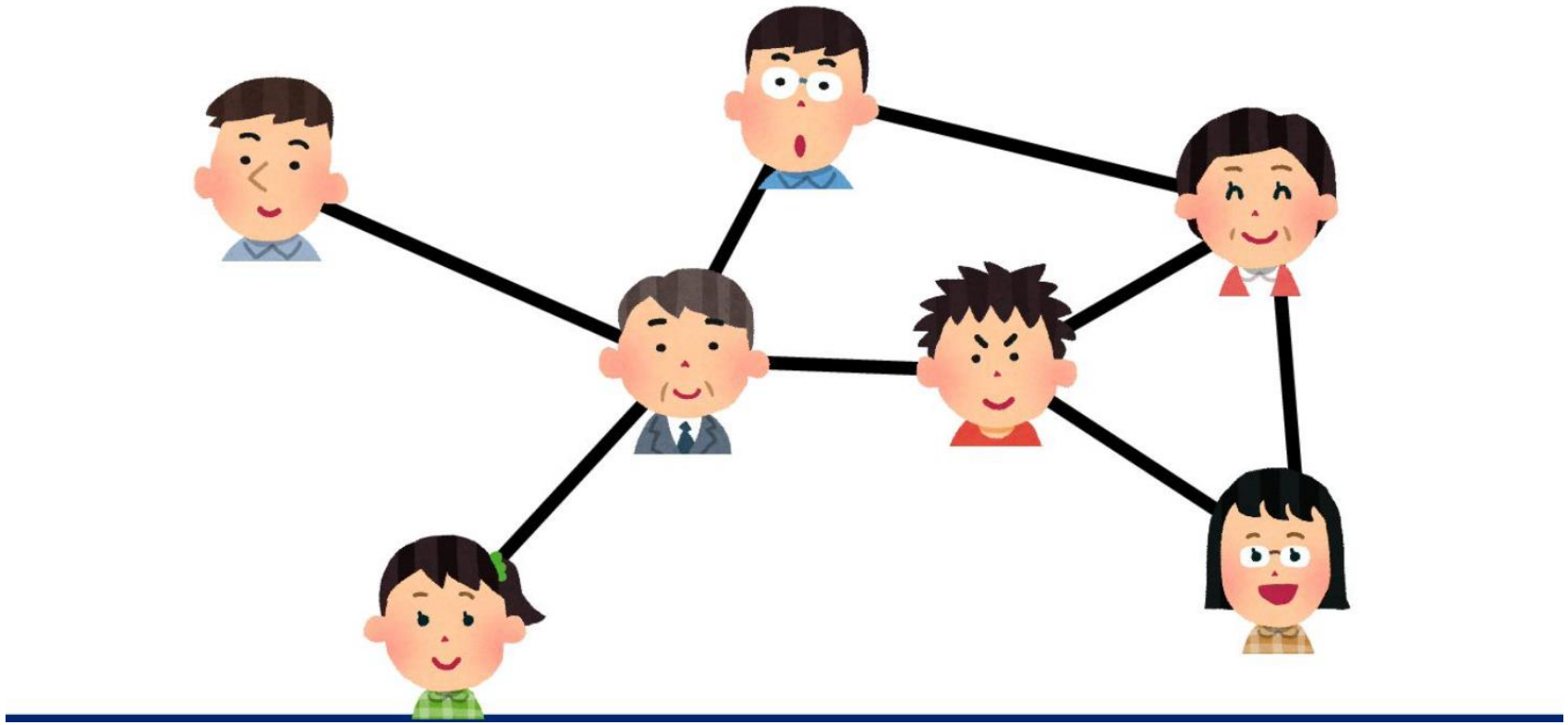
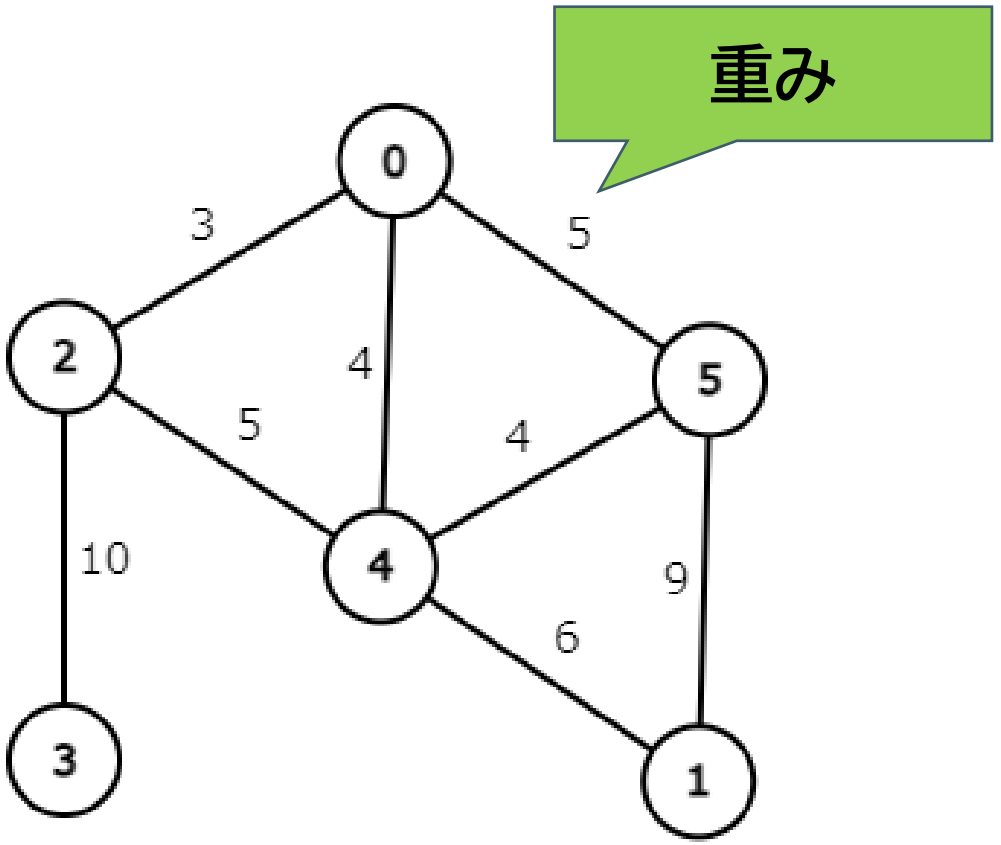


図: グラフアルゴリズム実践活用術, 佐藤竜馬
https://speakerdeck.com/joisino/gurahuarugorizumushi-jian-huo-yong-shu?slide=9

重み

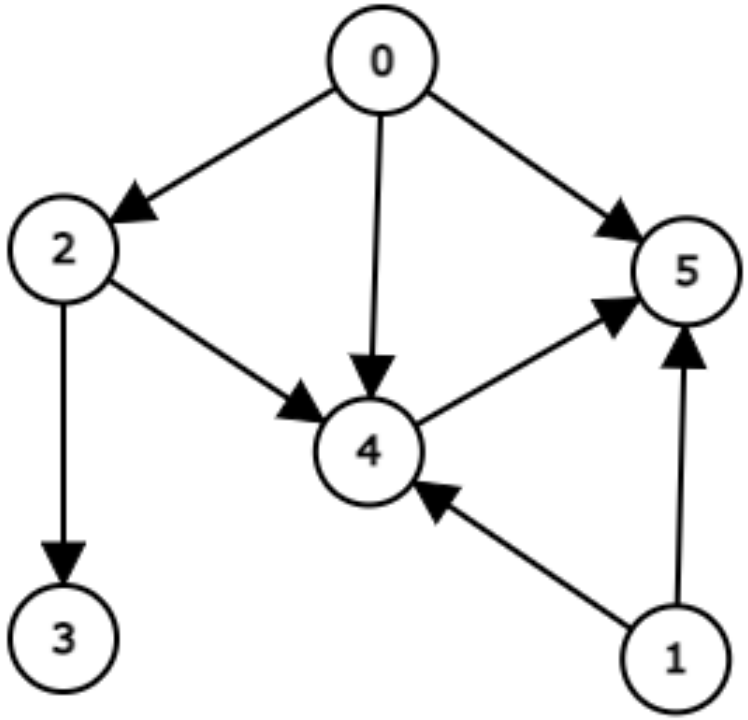
- 辺に重みという情報を載せる
  - 重みは数字で表現
  - コストと呼ばれることもある

- 意味を持たせる
  - 路線図: 移動時間
  - 人間関係: 関係の深さ



有向グラフと無向グラフ

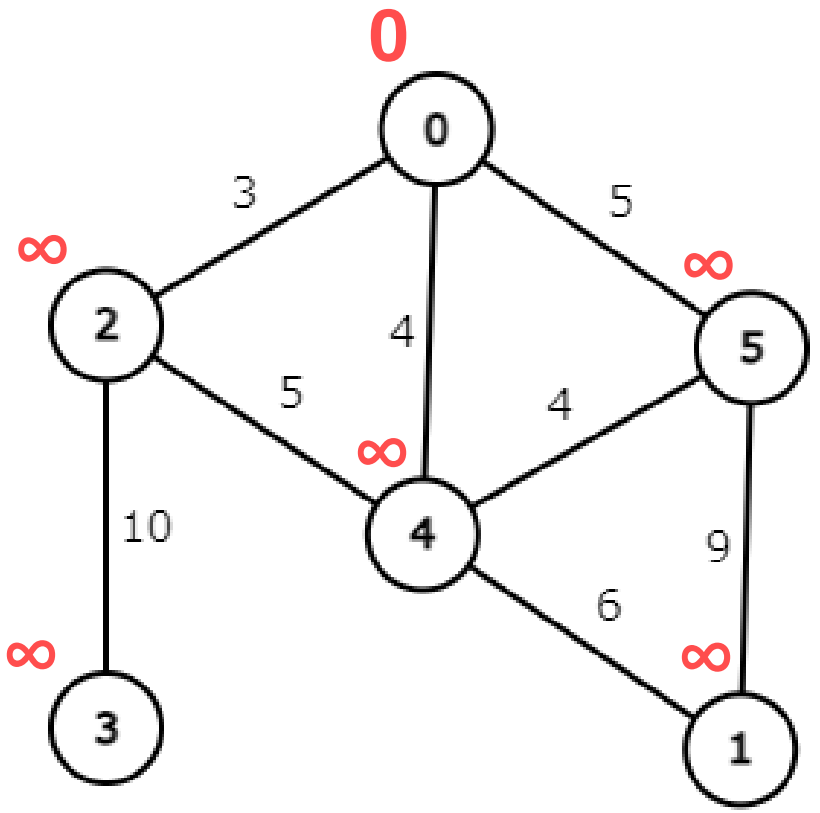
- 向きを付ける
  - 有向グラフ: 辺に向きがある
  - 無向グラフ: 辺に向きがない
- 表現能力が上がる
  - 路線図: 一方通行など
  - 人間関係: 片思い



最短経路を求めるアルゴリズム

- 頂点 A → 頂点 B へ最短で辿りつくためには？
  - 例: 頂点 0 → 頂点 1 への最短経路

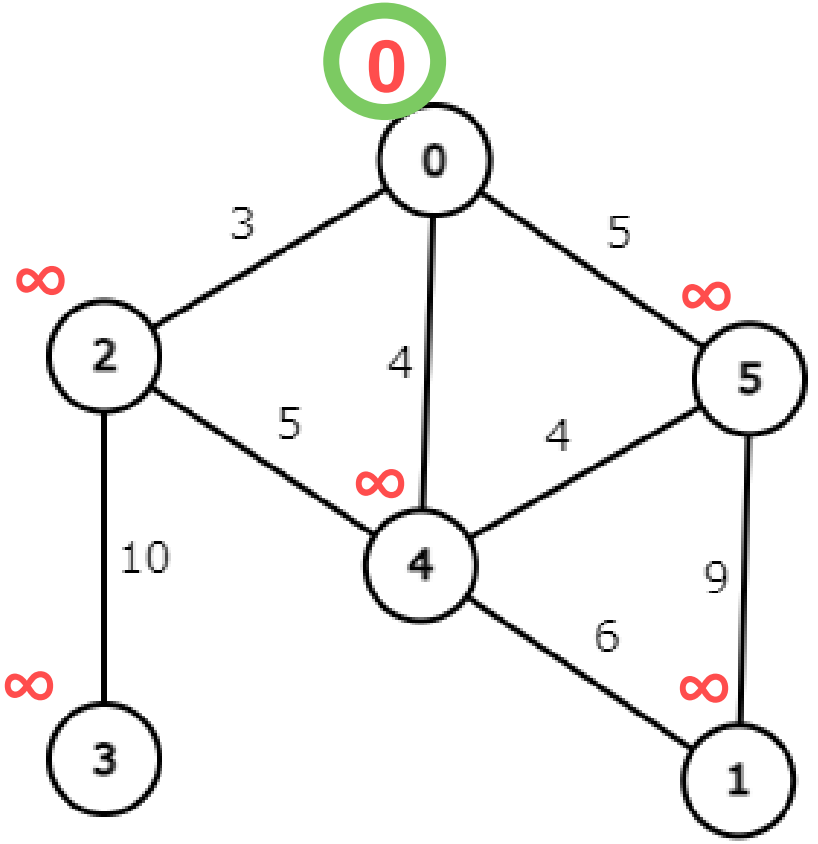
- Dijkstra法
  - 距離を初期化
    - スタート地点: 距離0
    - それ以外: 距離∞



最短経路を求めるアルゴリズム

- 頂点 A → 頂点 B へ最短で辿りつくためには？
  - 例: 頂点 0 → 頂点 1 への最短経路

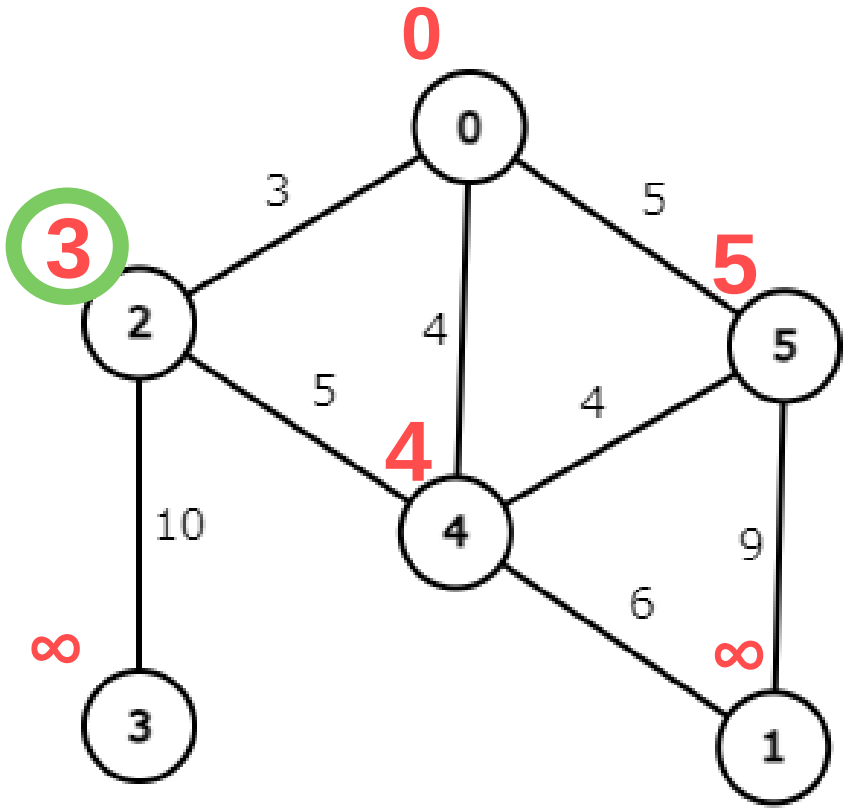
- Dijkstra法
  - 距離を初期化
    - スタート地点: 距離0
    - それ以外: 距離∞
  - 全ての頂点が確定するまで繰り返す
    - 未確定の頂点で距離が一番小さい頂点 x を選ぶ
    - 頂点 x の距離を確定する
    - 隣接点の距離を更新する



最短経路を求めるアルゴリズム

- 頂点 A → 頂点 B へ最短で辿りつくためには？
  - 例: 頂点 0 → 頂点 1 への最短経路

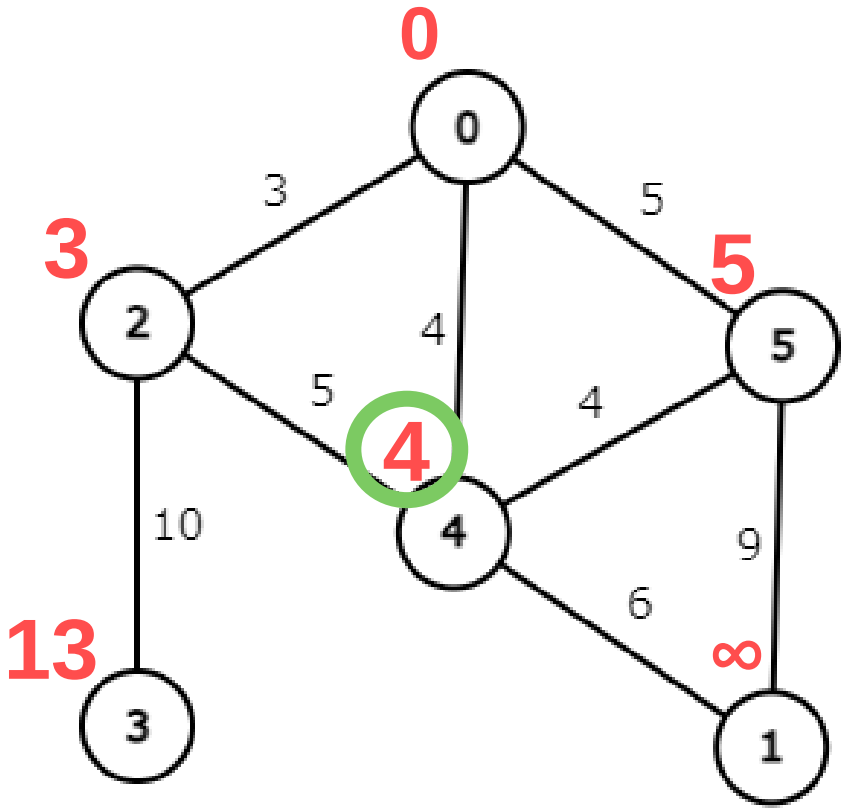
- Dijkstra法
  - 距離を初期化
    - スタート地点: 距離0
    - それ以外: 距離∞
  - 全ての頂点が確定するまで繰り返す
    - 未確定の頂点で距離が一番小さい頂点 x を選ぶ
    - 頂点 x の距離を確定する
    - 隣接点の距離を更新する



最短経路を求めるアルゴリズム

- 頂点 A → 頂点 B へ最短で辿りつくためには？
  - 例: 頂点 0 → 頂点 1 への最短経路

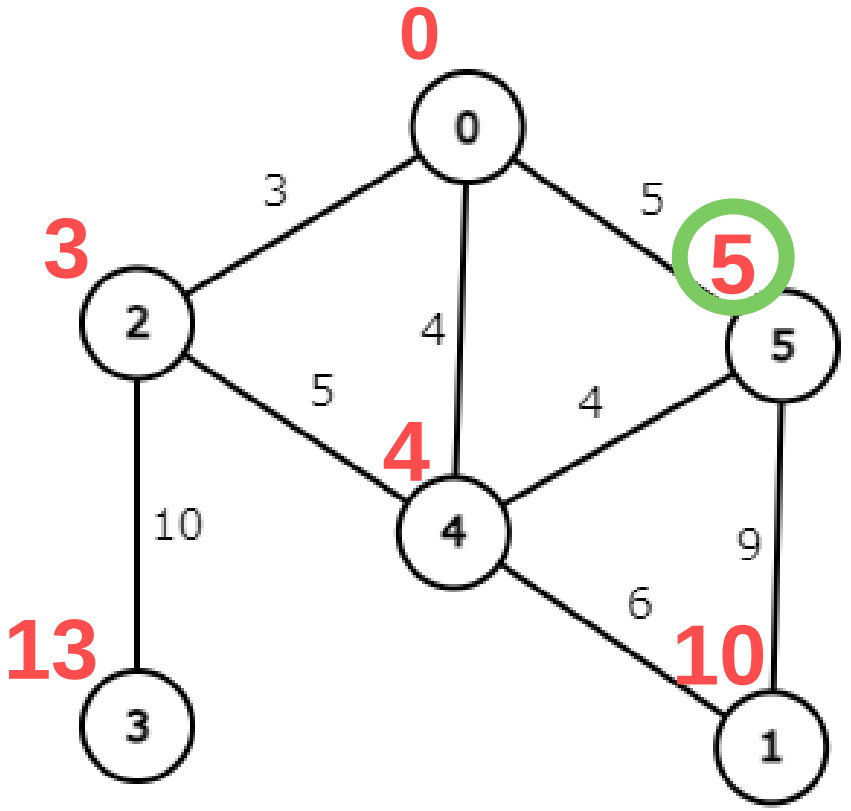
- Dijkstra法
  - 距離を初期化
    - スタート地点: 距離0
    - それ以外: 距離∞
  - 全ての頂点が確定するまで繰り返す
    - 未確定の頂点で距離が一番小さい頂点 x を選ぶ
    - 頂点 x の距離を確定する
    - 隣接点の距離を更新する



最短経路を求めるアルゴリズム

- 頂点 A → 頂点 B へ最短で辿りつくためには？
  - 例: 頂点 0 → 頂点 1 への最短経路

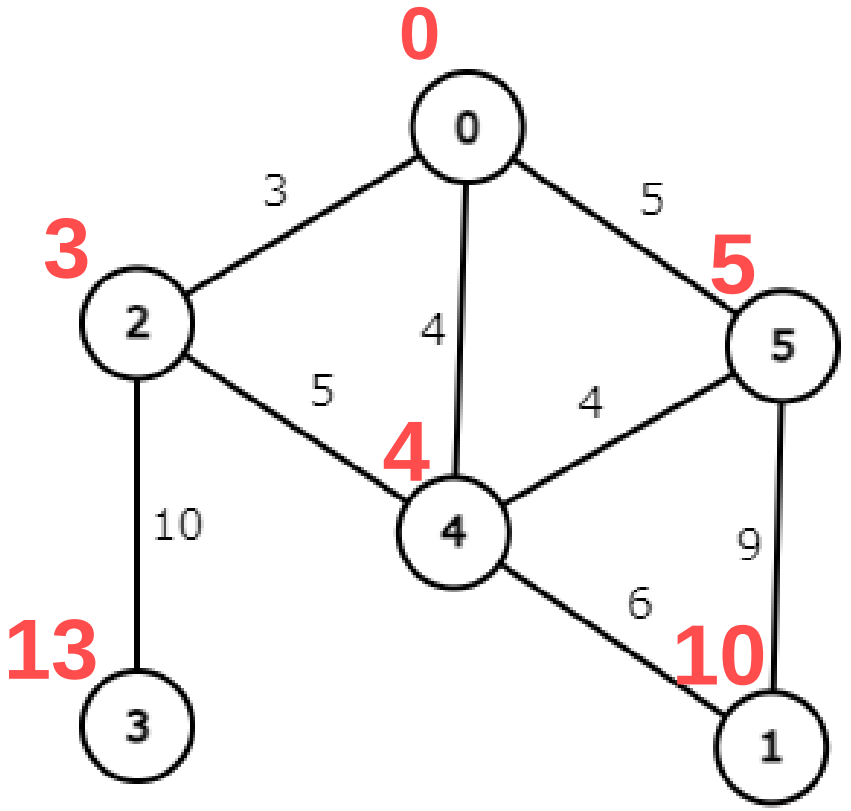
- Dijkstra法
  - 距離を初期化
    - スタート地点: 距離0
    - それ以外: 距離∞
  - 全ての頂点が確定するまで繰り返す
    - 未確定の頂点で距離が一番小さい頂点 x を選ぶ
    - 頂点 x の距離を確定する
    - 隣接点の距離を更新する



最短経路を求めるアルゴリズム

- 頂点 A → 頂点 B へ最短で辿りつくためには？
  - 例: 頂点 0 → 頂点 1 への最短経路

- Dijkstra法
  - 距離を初期化
    - スタート地点: 距離0
    - それ以外: 距離∞
  - 全ての頂点が確定するまで繰り返す
    - 未確定の頂点で距離が一番小さい頂点 x を選ぶ
    - 頂点 x の距離を確定する
    - 隣接点の距離を更新する



演習問題

- Adjacency List https://atcoder.jp/contests/abc276/tasks/abc276_b
- Easy Graph Problem https://atcoder.jp/contests/math-and-algorithm/tasks/typical90_bz
- Ladder Takahashi https://atcoder.jp/contests/abc277/tasks/abc277_c
- 幅優先探索 https://atcoder.jp/contests/abc007/tasks/abc007_3

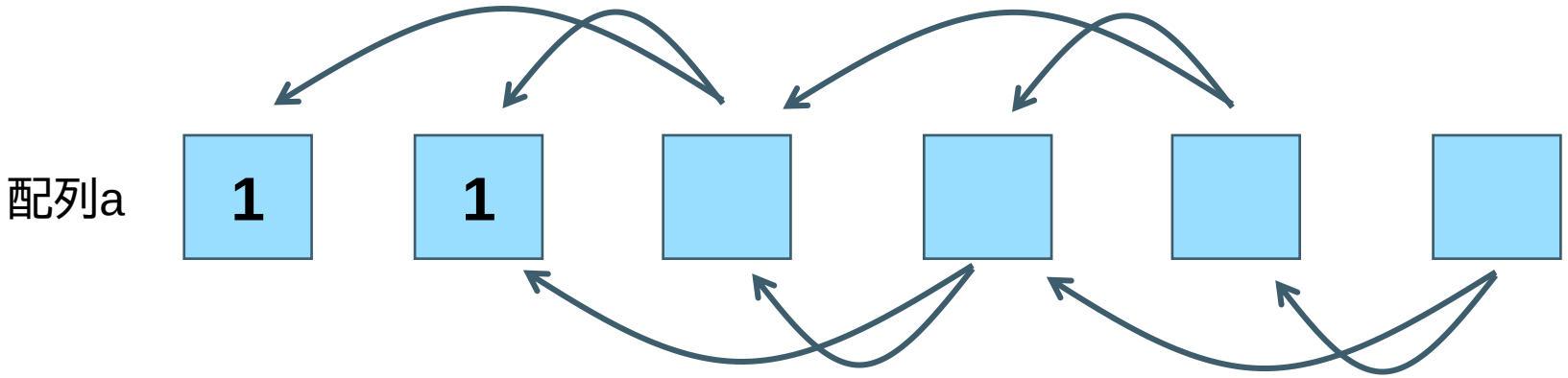
目次

- 中級編
  - 配列とfor文
  - 計算量の話
- 上級編
  - グラフ
  - 動的計画法
- 裏技編
  - 提出デバッグ

動的計画法

- 計算できるところから計算していく

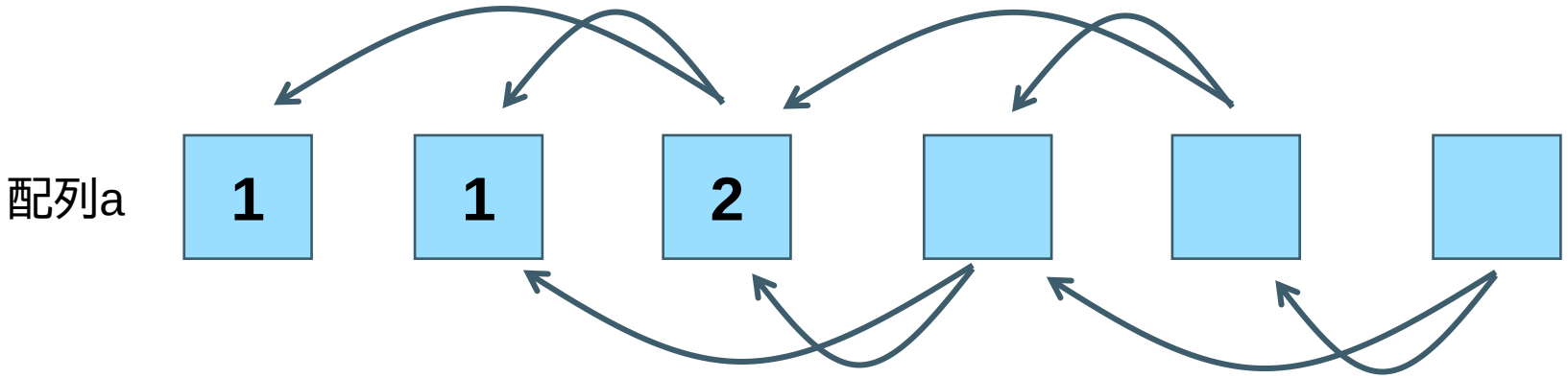
- 例: フィボナッチ数列
  - $a_i = a_{i-1} + a_{i-2}$



動的計画法

- 計算できるところから計算していく

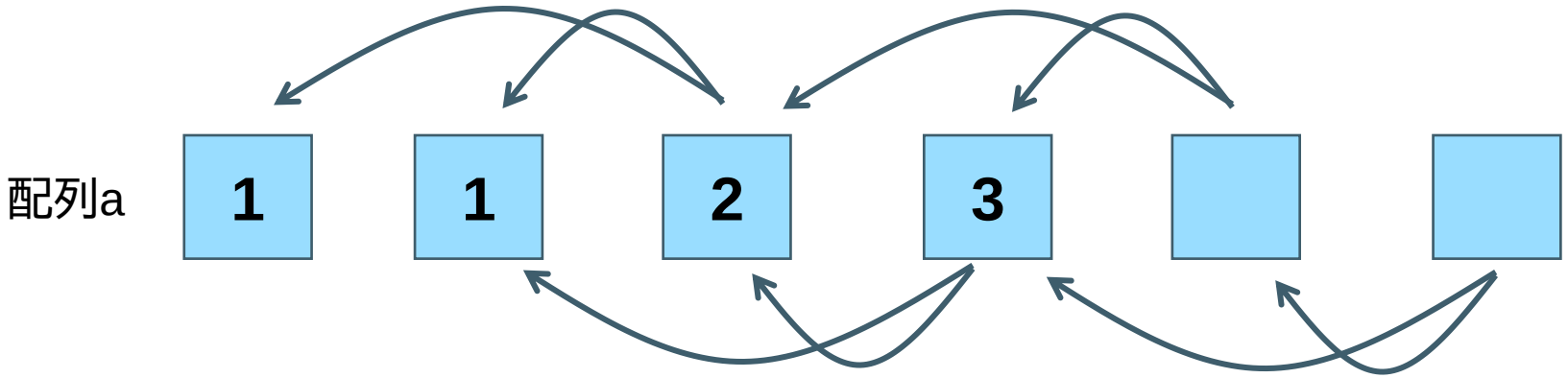
- 例: フィボナッチ数列
  - $a_i = a_{i-1} + a_{i-2}$



動的計画法

- 計算できるところから計算していく

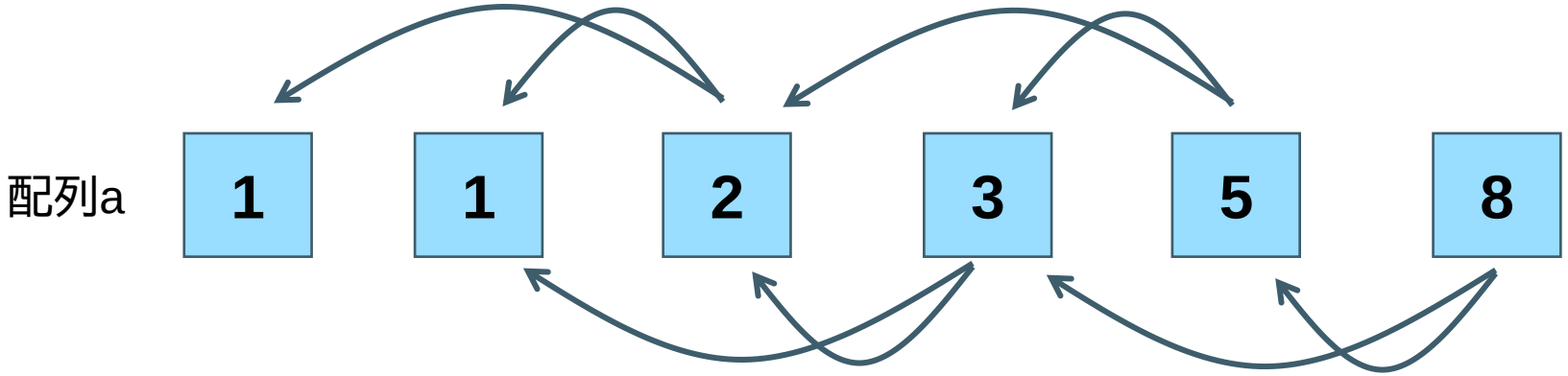
- 例: フィボナッチ数列
  - $a_i = a_{i-1} + a_{i-2}$



動的計画法

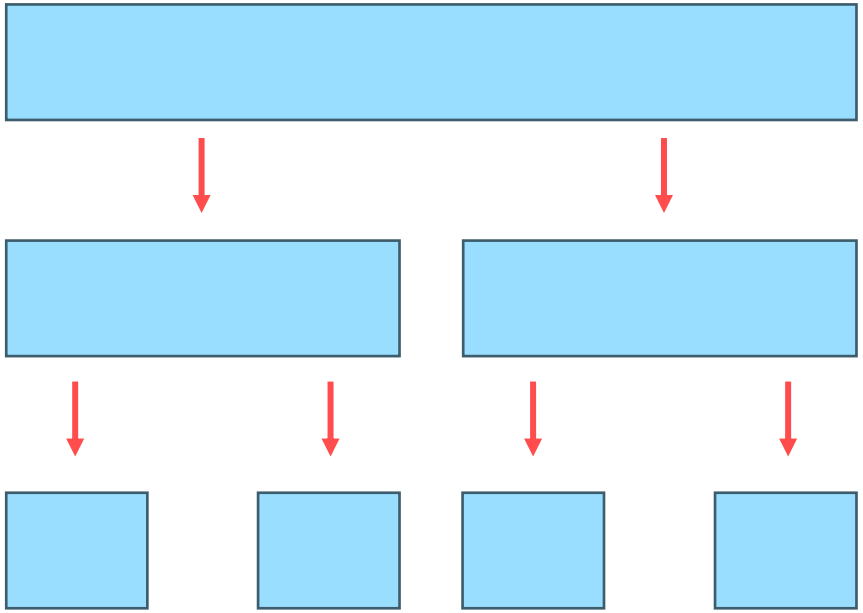
- 計算できるところから計算していく

- 例: フィボナッチ数列
  - $a_i = a_{i-1} + a_{i-2}$



動的計画法のアドバイス

- 大学学部3年以降の内容の先取りです
  - 難しくて当たり前
- コツ: 総当たりを書いてみる
  - 処理の**依存関係**がわかるはず
  - 処理を分割しよう
- 進んだコツ: 依存関係の向きを意識する
  - 漸化式: 明らか
  - 区間DP: 区間長を使う
  - ゲーム理論: ターン数、石の数など



演習問題

- DP まとめコンテスト
  - https://atcoder.jp/contests/dp?lang=ja

- 問題を解いていくのが近道

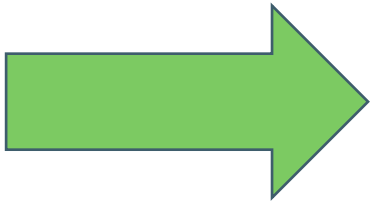
目次

- 中級編
  - 配列とfor文
  - 計算量の話
- 上級編
  - グラフ
  - 動的計画法
- 裏技編
  - 提出デバッグ

裏技: 採点システムを利用する

- 提出された解答ソースコードは競技システム上で自動採点され，提出結果のフィードバックが与えられる．
- 提出は何度でも行うことができる．フィードバックを参考として解答ソースコードを修正しても構わない．各課題について，競技時間内に提出した解答ソースコードの得点の最大値がその課題の得点となる．

https://www.ioi-jp.org/joi/2023/2024-yo1-yo2-joigho-rule



提出ペナルティなし

普段とは違う
戦略を取れる

ハック: 故意に間違える

WAとREは故意に作り出すことができる

- WA: 間違った答えを出力
  - 間違った答えを見つけることは、正しい答えを見つけるより遥かに簡単
- RE: 実行時エラーを故意に引き起こす

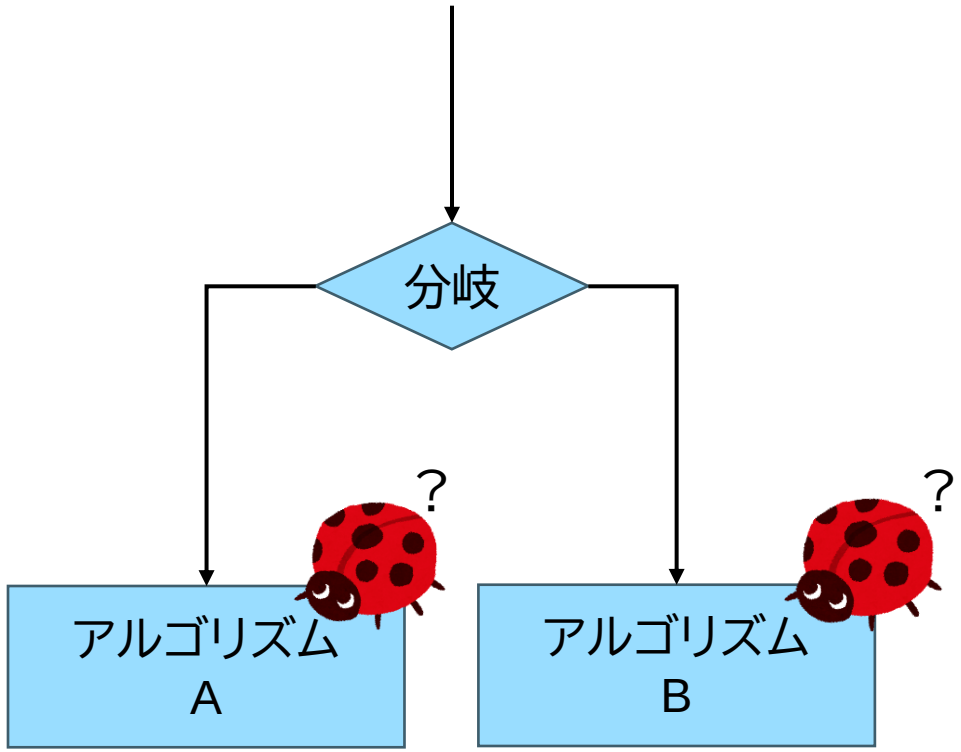
source.py

```
1    raise RuntimeError()
```

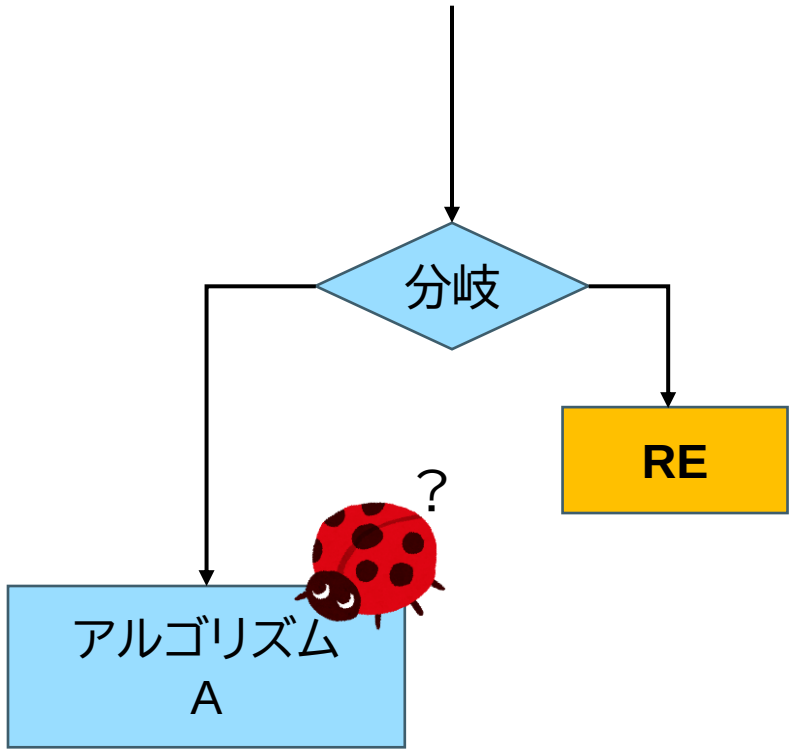
```
3    int main(){
4      assert(0 == 1);
5    }
```

応用例: 提出デバッグ

- 動作が怪しい2つのアルゴリズム
  - 片方は正しいアルゴリズム
  - 片方はWAを出すアルゴリズム

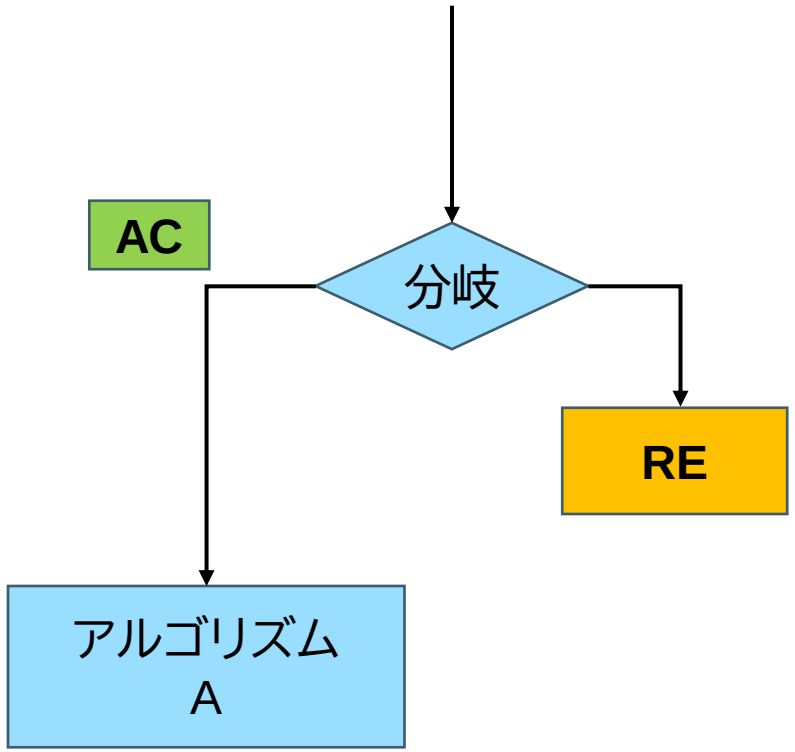


- 片方をREに置き換える
  - このプログラムを提出

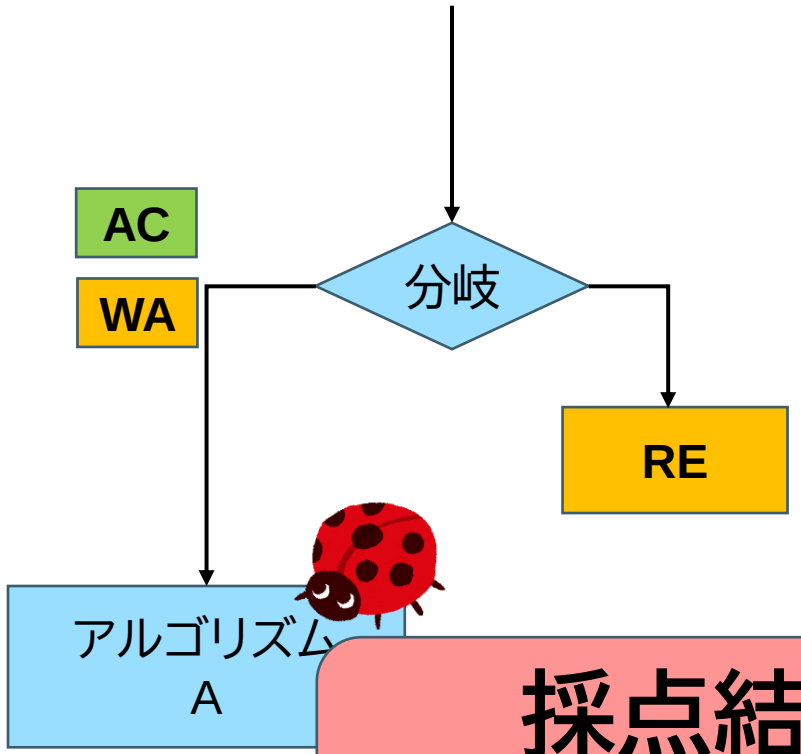


応用例: 提出デバッグ

- アルゴリズムAが正しい場合
  - ACとREが返ってくる



- アルゴリズムAが間違っている場合
  - ACとWAとREが返ってくる



採点結果で
デバッグができる

注意

AtCoderのシステムに攻撃を加えることは禁止しております。

https://atcoder.jp/contests/abc358/rules

- 競技システムが混雑する可能性があるため，無用な提出を行ってはならない．

https://www2.ioi-jp.org/joi/2022/2023-yo1-yo2-joigho-rule.html

やり過ぎは
ご法度

演習問題

- 三方比較 https://atcoder.jp/contests/joi2023yo1b/tasks/joi2023_yo1b_b
  - 整数 A, B が入力で与えられるので、大小を比較する問題
- 以下のプログラムを作って提出してみましょう
  - 正しいプログラム
  - A ＜ B のとき間違った答えを出力するプログラム
  - A ＜ B のときREで実行時エラーを引き起こすプログラム