北海道大学

プログラミングコンテスト入門

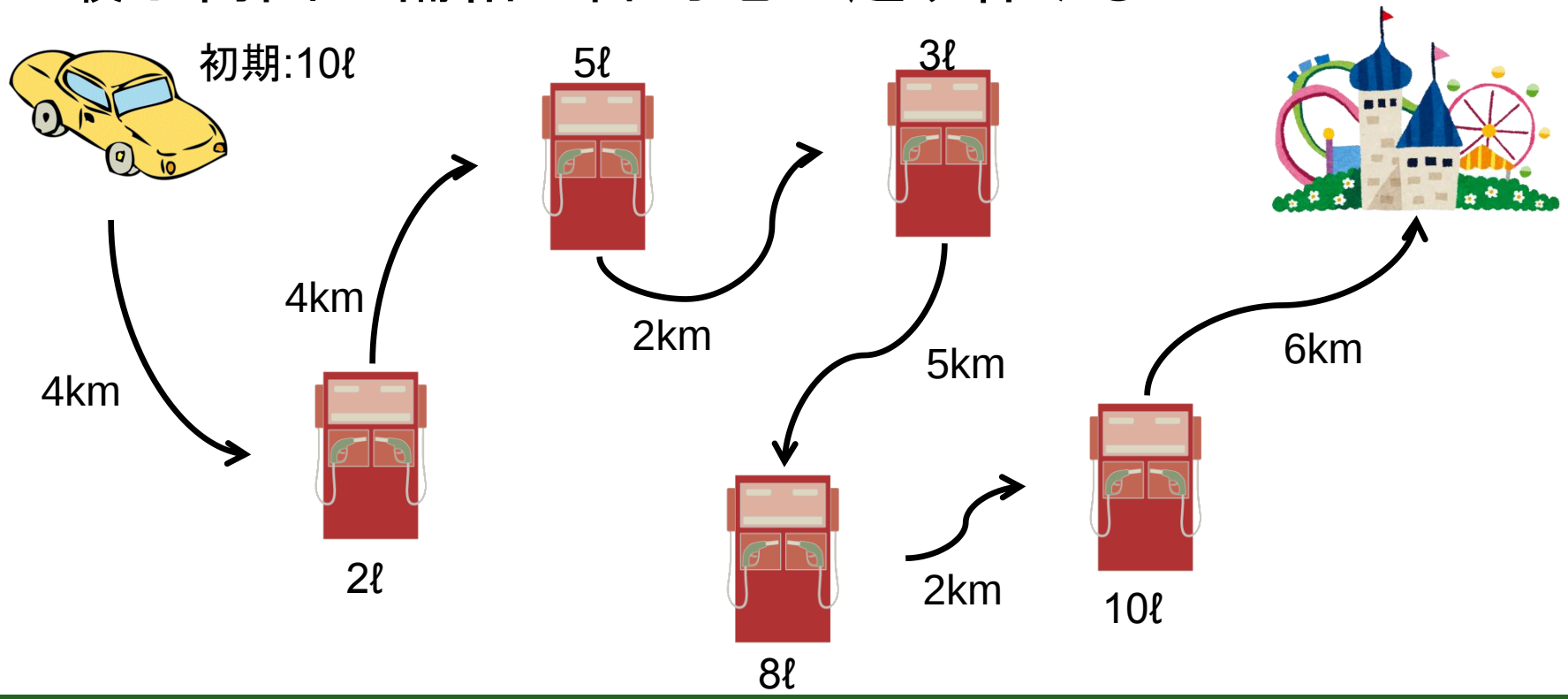
アルゴリズム研究室 井上祐馬

プログラミングコンテストとは？

- 複数の問題が与えられる
- 制限時間内に与えられた問題をできるだけ多く解く
- 「問題を解く」とは、「問題を解くプログラムを作成する」ことである

問題例：ガソリンスタンド

- ガソリンスタンドを経由して車で目的地へ
- 1ℓで1km進める。各地で一定量のガソリンが補給可能
- 最小何回の補給で目的地へ辿り着けるか？



問題文の構成

Expedition

Language: Default

Time Limit: 1000MS **Memory Limit:** 65536K
Total Submissions: 3782 **Accepted:** 1253

Description

A group of cows grabbed a truck and ventured on an expedition deep into the jungle. Being rather poor drivers, the cows unfortunately managed to run over a rock and puncture the truck's fuel tank. The truck now leaks one unit of fuel every unit of distance it travels.

To repair the truck, the cows need to drive to the nearest town (no more than 1,000,000 units distant) down a long, winding road. On this road, between the town and the current location of the truck, there are N (1 <= N <= 10,000) fuel stops where the cows can stop to acquire additional fuel (1..100 units at each stop).

The jungle is a dangerous place for humans and is especially dangerous for cows. Therefore, the cows want to make the minimum possible number of stops for fuel on the way to the town. Fortunately, the capacity of the fuel tank on their truck is so large that there is effectively no limit to the amount of fuel it can hold. The truck is currently L units away from the town and has P units of fuel (1 <= P <= 1,000,000).

Determine the minimum number of stops needed to reach the town, or if the cows cannot reach the town at all.

問題背景

Input

* Line 1: A single integer, N

* Lines 2..N+1: Each line contains two space-separated integers describing a fuel stop: The first integer is the distance from the town to the stop; the second is the amount of fuel available at that stop.

* Line N+2: Two space-separated integers, L and P

入力形式

Output

* Line 1: A single integer giving the minimum number of fuel stops necessary to reach the town. If it is not possible to reach the town, output -1.

出力形式

Sample Input

```
4
4 4
5 2
11 5
15 10
25 10
```

サンプル入力

Sample Output

```
2
```

サンプル出力

例：問題文

- あなたは出発地点からLkm離れた到着地点に車で向かいたい。
- 1km進むためには1ℓのガソリンを消費する。
- 出発時点であなたはPℓのガソリンを持っている。
- 途中でガソリンが0になってしまうと、移動できなくなる。
- ただし、道中N箇所あるガソリンスタンドに立ち寄れる。
- i番目のガソリンスタンドは出発地点から$a_i$kmの地点に存在し、そこではb_iℓのガソリンを補給できる。
- 車は無限にガソリンを積めるとすると、到着地点に到達可能か？可能ならば、最小の補給回数を求めよ。

入力形式

- N
- a_1 b_1
- a_2 b_2
- …
- a_N b_N
- L P

- 1≦N≦10000
- 1≦L≦1000000, 1≦P≦1000000
- 1≦ a_i < L, 1≦b_i≦100

出力形式

- 到着地点に到達できないなら、-1を
- 到達できるなら、最小の補給回数を
- 一行で出力せよ

サンプル

Input:

4

4 4

5 2

5 2

11 5

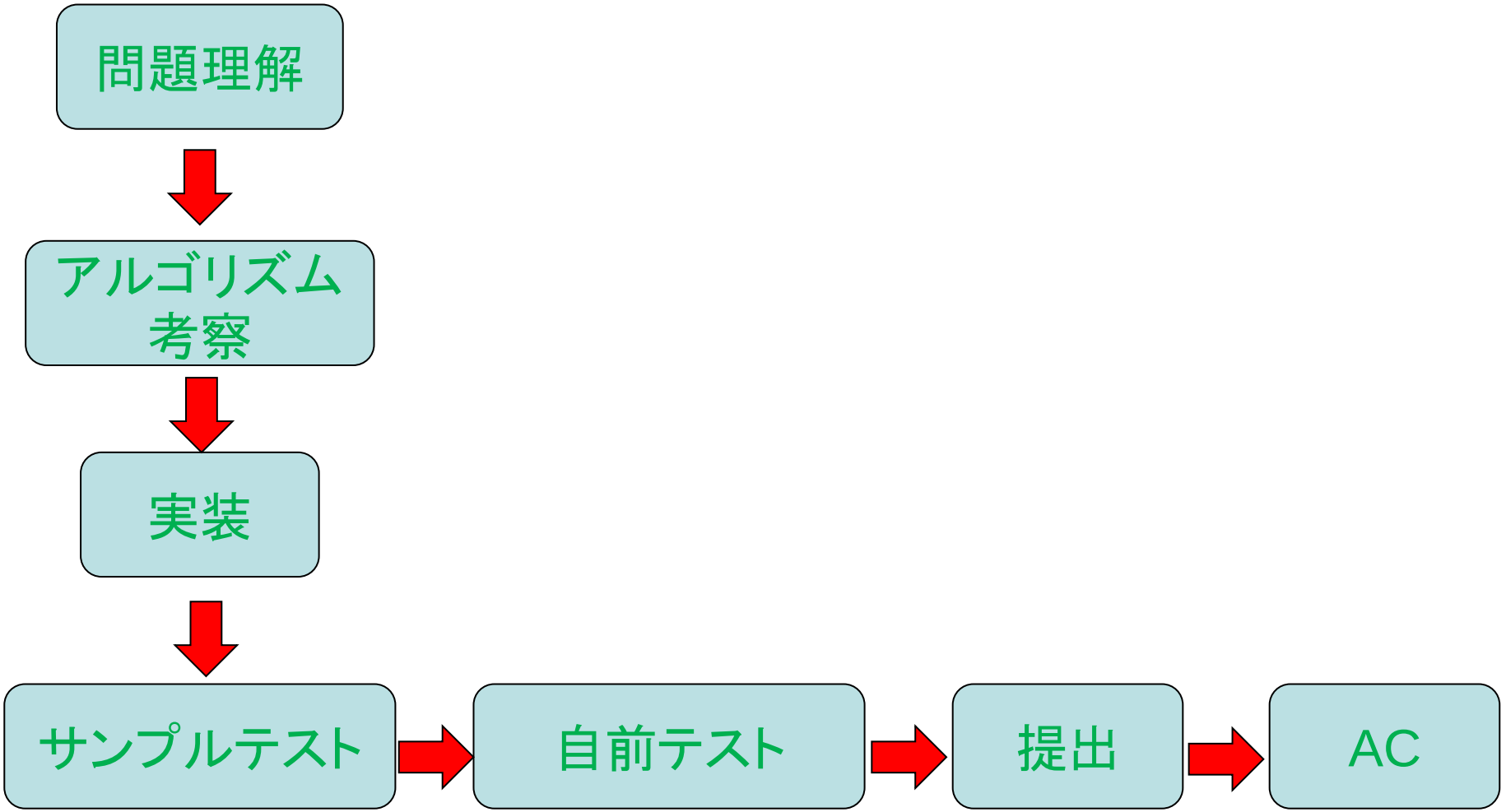
15 10

25 10

Output:

2

コンテストの流れ



問題解釈

- 問題を正しく理解する
  - 記述が複雑だったり、英語だったり、斜め読みしたりすると誤読がしばしば起きる
  - 誤読している問題は(奇跡的偶然がない限り)永遠に解けない

- 問題を正しく理解できているかを確かめるためには
  - 改めて読み直して、読み落とし・勘違いがないか確かめる
  - サンプルを手で解いてみる
  - チーム戦なら、チームメイトにも読んでもらい、内容を確認し合う

アルゴリズム考察

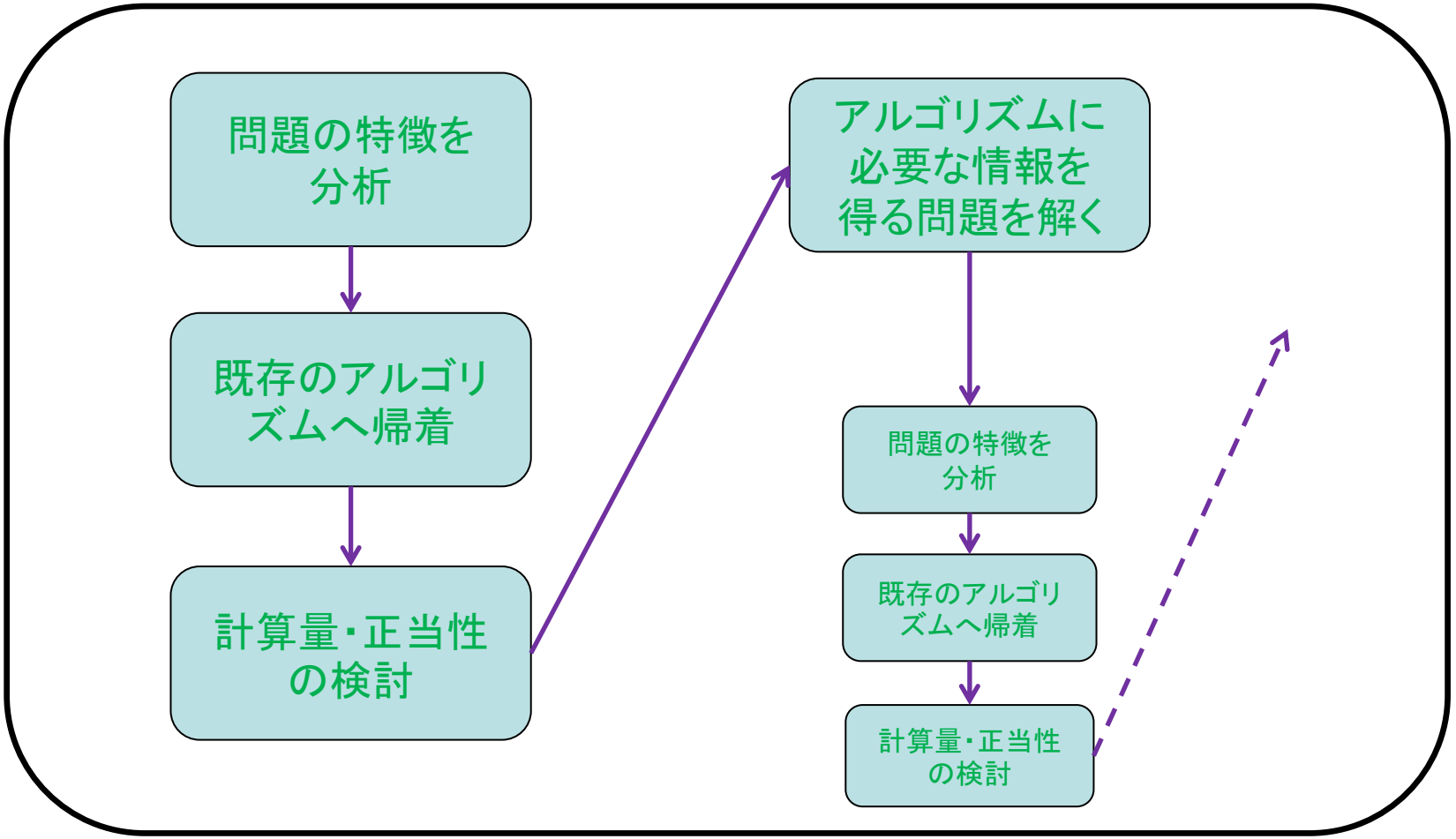
- 問題を解く手順(=アルゴリズム)を考える
  - プログラミングコンテストで最も重要と言ってもよいフェイズ

- 問題から導ける事実をかき集める
  - 「現在のガソリン<次のGSへの距離」なら進めない
  - 使うGSが予め決まっていれば、到達できるか判定できる
  - 「GSを訪れる＝以降そのGSで補給する権利を得る」 etc…
- 事実に基づいてアルゴリズムを設計する
  - 「GSが決まれば到達判定可能⇒GSの決め方を全部試す」など
  - アルゴリズムに必要な情報を求めるためにサブ問題を解くこともしばしばある

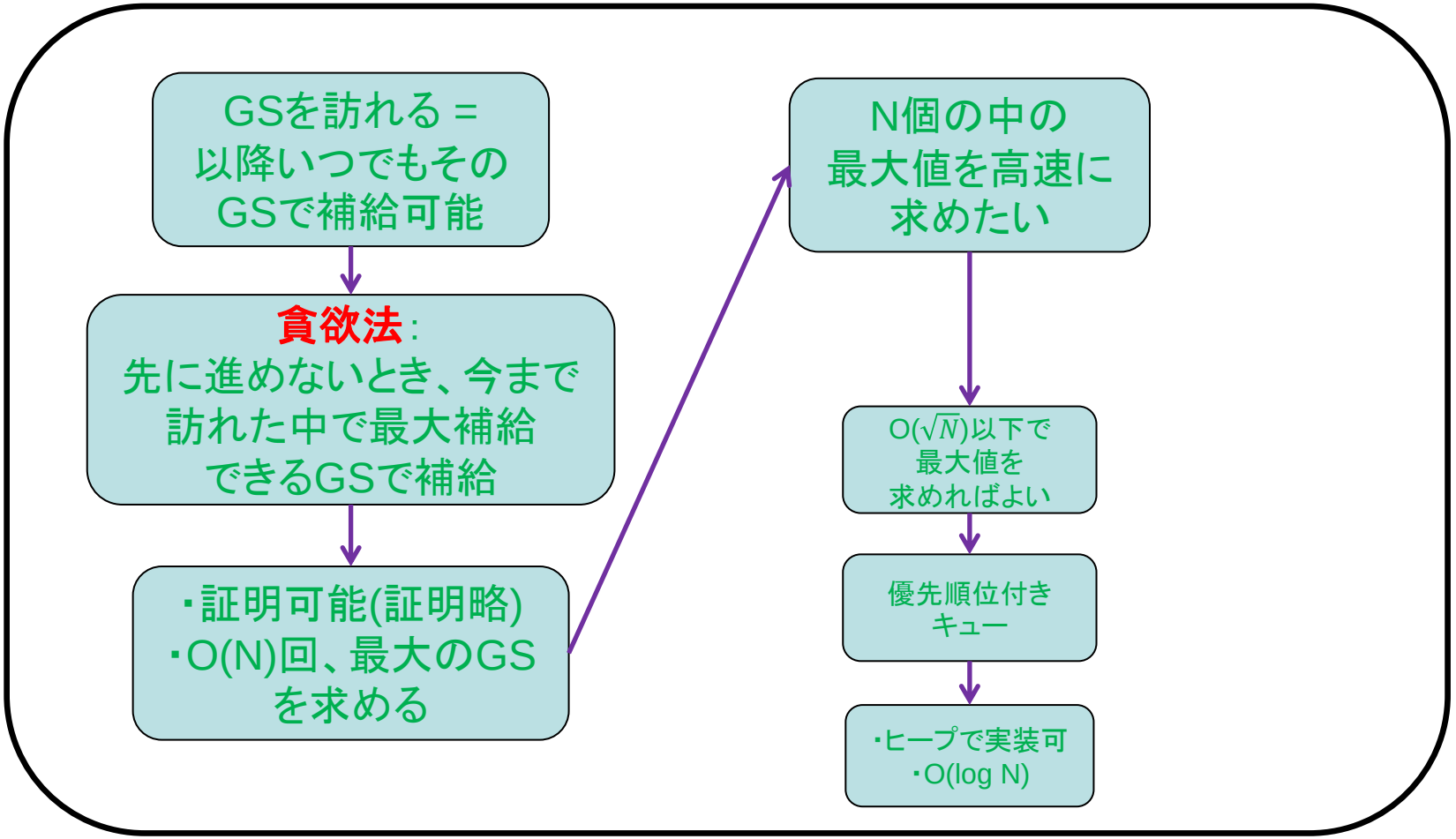
アルゴリズム考察

- 計算量の見積もりは大事
  - ICPCは明確な実行時間制限はないが、さすがに3時間で終わらないプログラムは論外
  - O-記法(オーダー記法)を用いて評価することが多い
    - 計算回数をデータの大きさの関数として概算する
    - 例えば、n個から3つ選んだ和を求めるとき3重ループで計算した場合、$\frac{n(n-1)(n-2)}{6}$回の足し算を行う。これを大まかに見積もってO(n^3)と書く
  - O()の中身が最大になるように制約の数値を入れてみる
    - 例えば、1≦n≦100のとき、O (n^3)はO(1000000) = O(10^6)
  - 大まかに、O(10^7～10^8)程度で1秒ほどかかる
    - これを大きく越えるときは、アルゴリズムを見直す

アルゴリズム考察


問題の特徴を分析
既存のアルゴリズムへ帰着
計算量・正当性の検討
アルゴリズムに必要な情報を得る問題を解く
問題の特徴を分析
既存のアルゴリズムへ帰着
計算量・正当性の検討

アルゴリズム考察


GSを訪れる＝
以降いつでもその
GSで補給可能
貪欲法：
先に進めないとき、今まで
訪れた中で最大補給
できるGSで補給
・証明可能(証明略)
・O(N)回、最大のGS
を求める
N個の中の
最大値を高速に
求めたい
O(√N)以下で
最大値を
求めればよい
優先順位付き
キュー
・ヒープで実装可
・O(log N)

実装

- アルゴリズムをきちんと考察できていれば、あとはそのアルゴリズムを実装するだけ
  - アルゴリズムの設計をきちんとしよう

- とはいえ、複雑な部分も多い
  - 頻出テクニックを身につけておく
  - STLなど、標準で使えるものを最大限活用する
  - 自作のライブラリ等によく使うアルゴリズムをまとめる

  - これらは、バグを減らす意味でも役立つ

実装(入力)

```
int n,L,P;
int a[10000],b[10000];


cin >> n;
for(int i=0;i<n;i++)cin >> a[i] >> b[i];
cin >> L >> P;
```

実装(アルゴリズム)

```
dis[0] = a[0];
for(int i=1;i<n;i++)dis[i] = a[i] – a[i-1];
dis[n] = L – a[n-1];

int curGas = P, res = 0;
priority_queue<int> q;
for(int i=0;i<=n;i++){
  while(curGas < dis[i]){
    if(q.empty())return -1;
    curGas += q.top(); q.pop();
    res++;
  }
  q.push(b[i]);
}
return res;
```

dis[i]にi番目の地点から
次の地点への距離を記録

curGas:現在の残りガソリン, res:答えをメモ

優先順位付きキュー(C++ STLにある)

次の地点への移動に必要なだけ補給

使えるGSがもうなかったら到達不可

最大値が常にtopに来ている

補給したので回数を1増やす

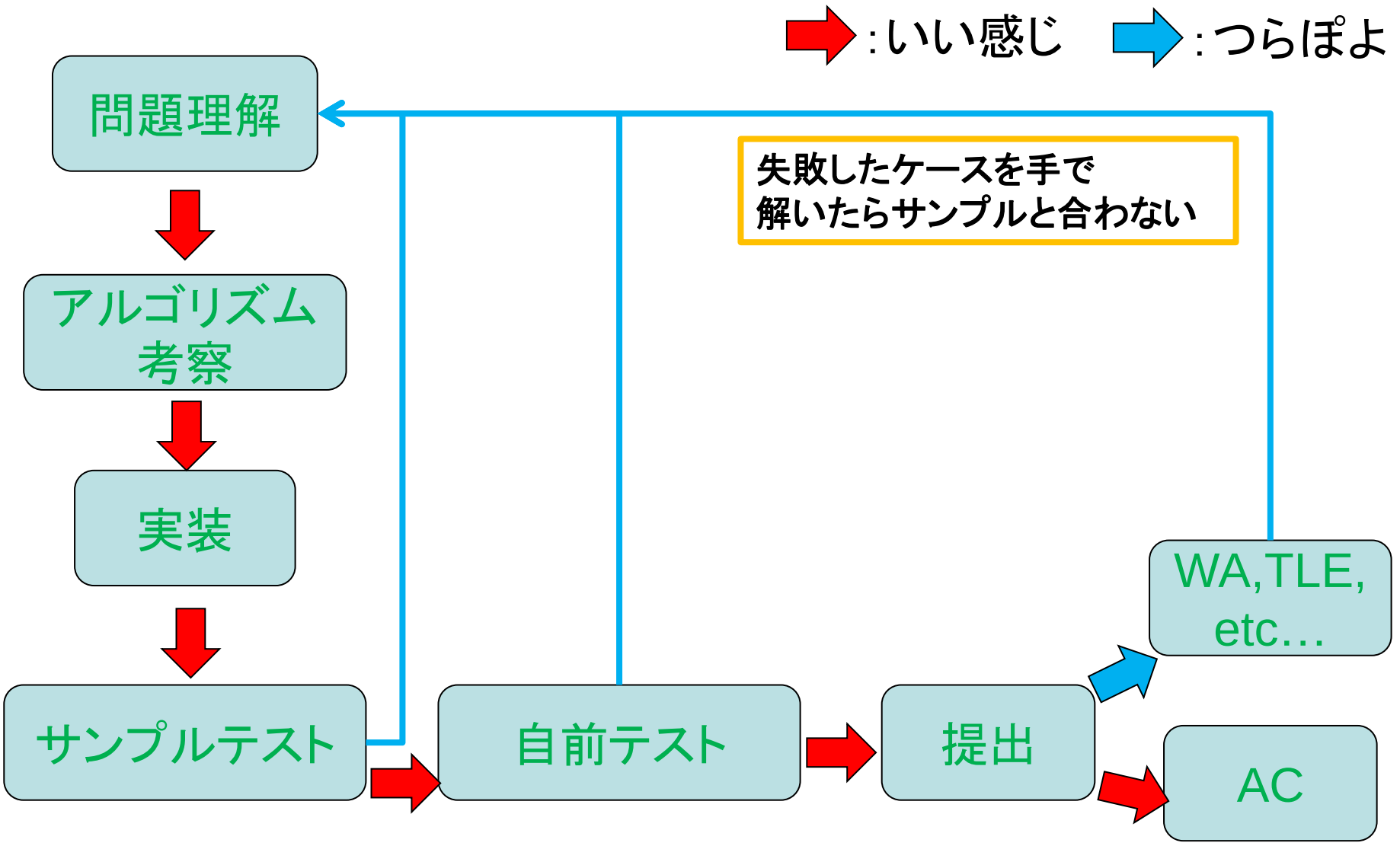
今いるGSをこれ以降使えるようにする

最後まで到達できたら、補給回数を返す

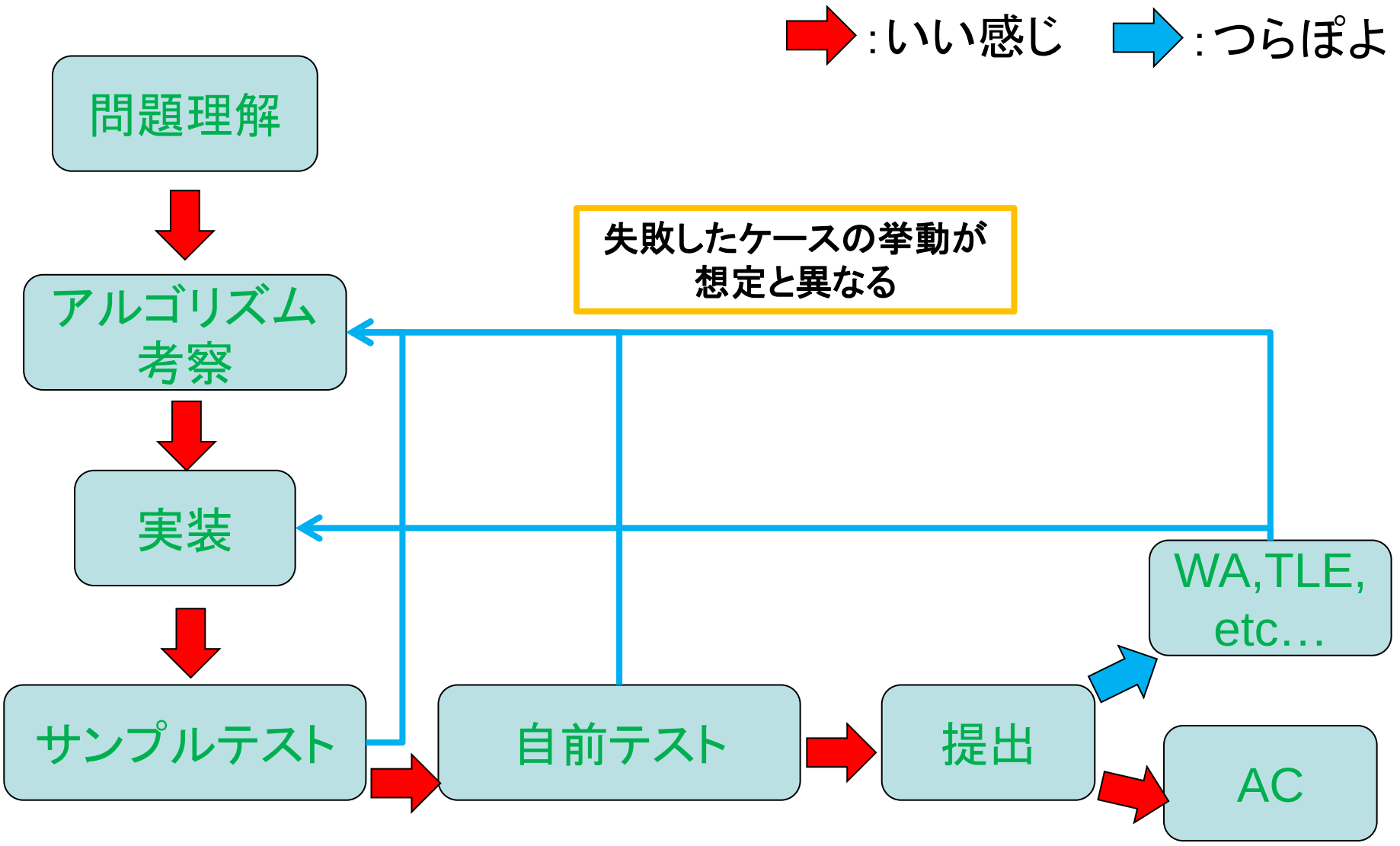
テスト

- どんな優秀なプログラマでも、バグは埋め込んでしまうもの
- テストは必ず行いましょう
  - まずはサンプルが正しく動くか確かめる
  - サンプルだけでは試せないやばそうなケースを自分で作って確かめてみる
    - 最大ケース、最小ケース
    - グラフ：負の辺、ループ、多重辺
    - 計算：0や負の値、オーバーフロー
    - 幾何：図形が接している etc…
  - どんなケースがやばそうか、は経験が大事
    - 一度したミスはチェックシートなどにまとめるとよいかも

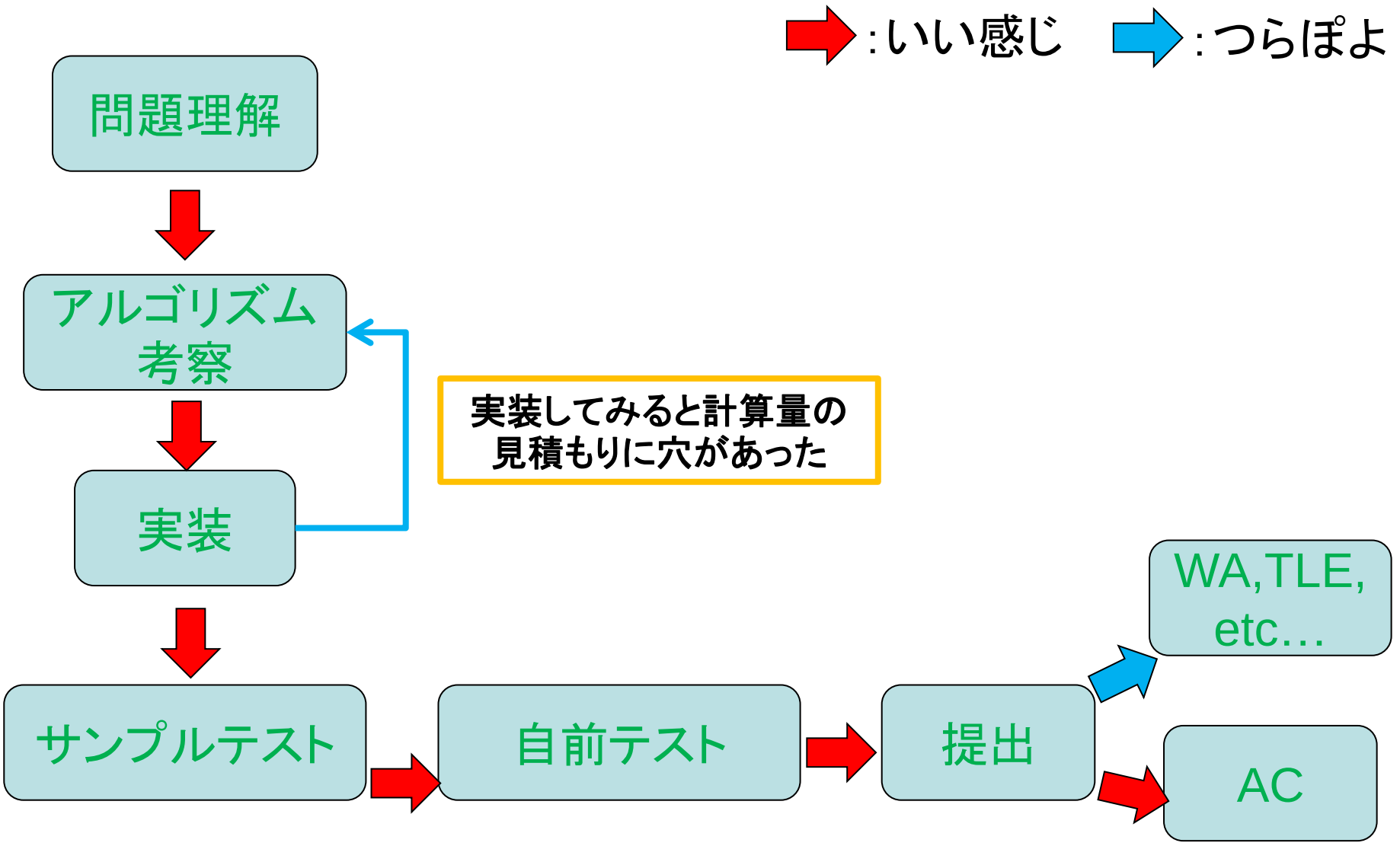
テストで失敗したら


:いい感じ
:つらぽよ
問題理解
アルゴリズム考察
実装
サンプルテスト
自前テスト
提出
WA,TLE, etc…
AC
失敗したケースを手で
解いたらサンプルと合わない

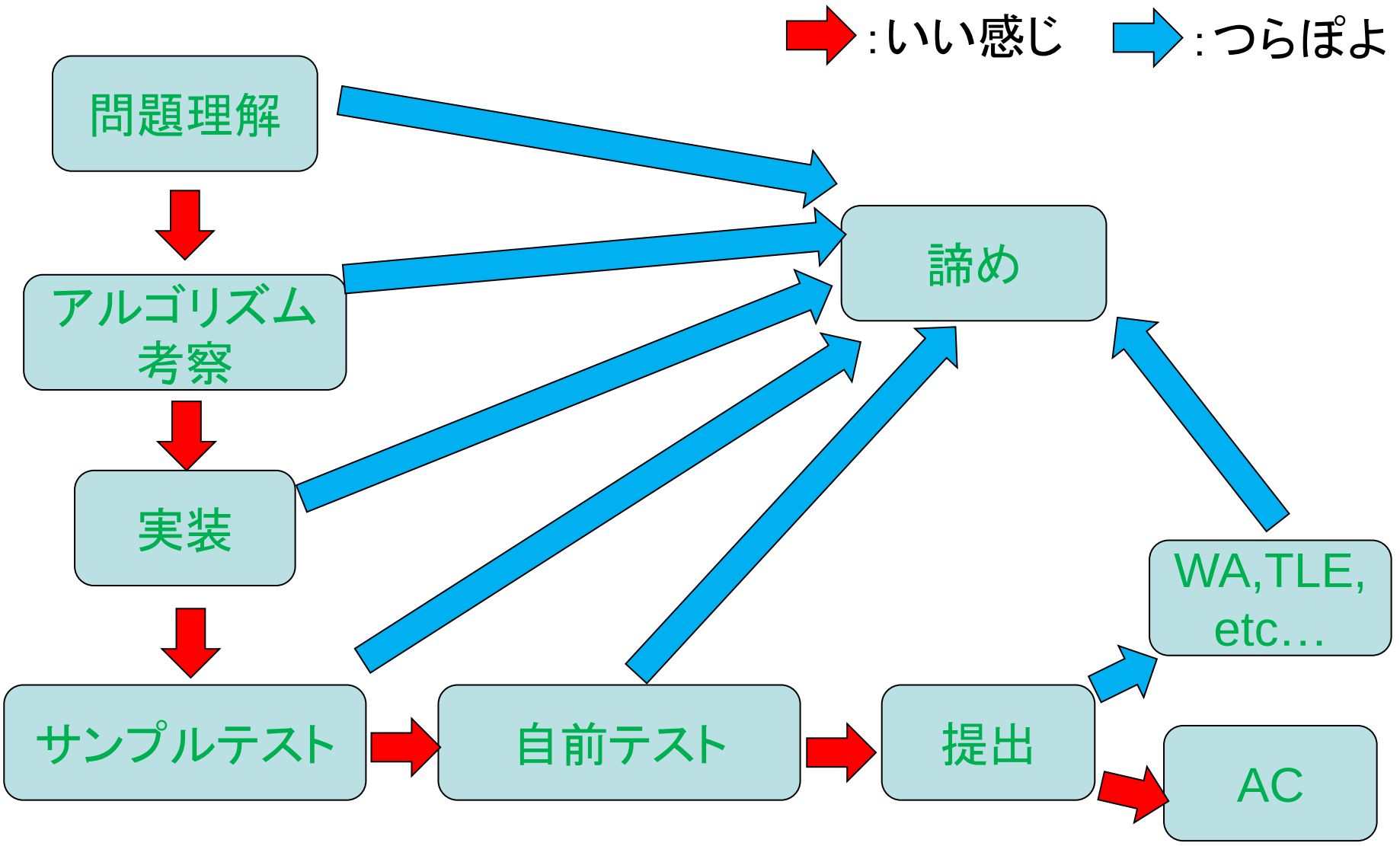
テストで失敗したら


:いい感じ
:つらぽよ
問題理解
アルゴリズム考察
実装
サンプルテスト
自前テスト
提出
WA,TLE, etc…
AC
失敗したケースの挙動が想定と異なる

テストで失敗したら


:いい感じ
:つらぽよ
問題理解
アルゴリズム考察
実装
実装してみると計算量の見積もりに穴があった
サンプルテスト
自前テスト
提出
WA,TLE, etc…
AC

失敗したら



コンテストの流れ

- 「諦め」はネタではない！
  - 解ける問題から解く、解けないときは他の問題も目を通そう
  - 他に解く問題がないときは諦めてはいけない。

- 問題が解けないのはたいていこんなとき
  - 問題解釈がそもそも間違っている
  - アルゴリズムが間違っている
    - 出力が正しくない
    - 計算量の見積もりに誤りがある
  - 実装がバグっている
  - オーバーフローしている
  - 解法(アルゴリズム)が思いつかない

まとめ

- プログラミングコンテストは・・・
  - 問題を読んで特徴を分析し、
  - 十分に効率的なアルゴリズムを考え、
  - それを実装するコンテスト

- よい成績を残すために必要なものは・・・
  - アルゴリズムやデータ構造の知識
  - 必要な情報をピックアップしたり導くための分析、柔軟な思考
  - 頻出な実装ポイントの経験やテクニック
  - デバッグの勘
  - 解ける問題の選定