

AtCoder Beginner Contest 009

解説

はじめに

- ・ 競技プログラミングが初めての人
- ・ まだまだ競技プログラミングに慣れていない人

最も基礎的なことは過去の解説に載っています！
特に ABC004 の解説が詳しいです。

<http://www.slideshare.net/chokudai/abc004>

AtCoder の練習用コンテストも活用しましょう！

<http://practice.contest.atcoder.jp/>

A問題

引越し作業

A - 問題概要

問題

N 個のダンボールを運びたい。

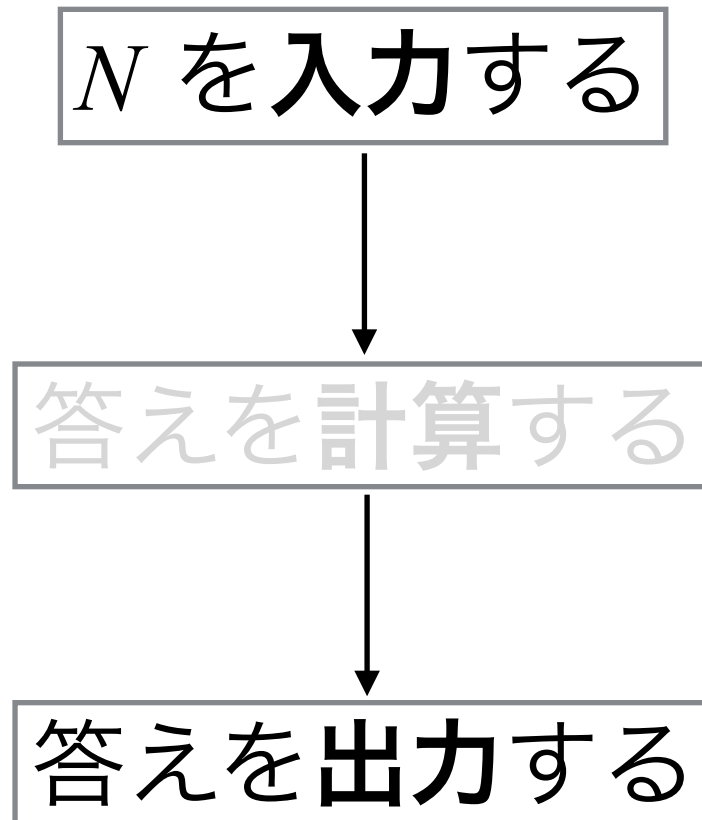
1 往復で最大 2 個のダンボールを運ぶことができる。

最低でも何往復する必要があるか？

制限

$$1 \leq N \leq 1,000$$

A - プログラムの流れ

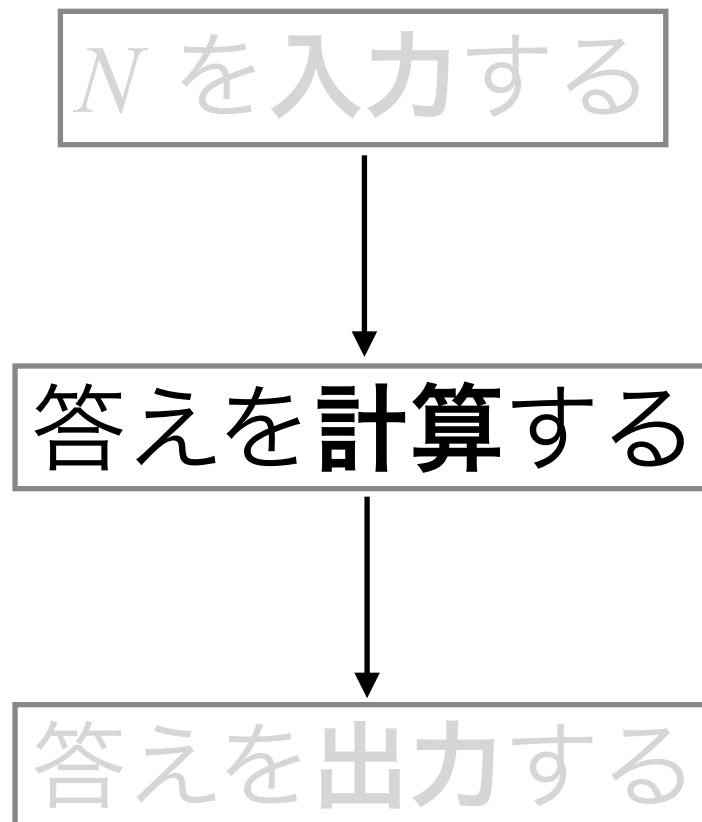


入力、出力でつまづく場合
→入出力について復習！

<http://www.slideshare.net/chokudai/abc004>

<http://practice.contest.atcoder.jp/>

A - プログラムの流れ



入力、出力でつまづく場合
→入出力について復習！

<http://www.slideshare.net/chokudai/abc004>

<http://practice.contest.atcoder.jp/>

残りは答えの計算



答えを計算する方法 (=アルゴリズム) を考えよう！

A - 解き方

重要な事実

ダンボールが 2 個以上残っているときは
1 回の往復で 2 個運んでしまうほうがよい。

当たり前のようなことですが、
プログラムを書いて計算させるにあたっては
こういった事実をきちんと確認するのが大事です！

A - 解き方

この事実を使えば、次のように運ぶのが最適:

- ・ダンボールが 1 個しかないなら、それを運ぶ。
- ・ダンボールが 2 個以上あれば、2 個運ぶ。

これをダンボールがなくなるまで繰り返して、そのときの回数が答えになる。

A - プログラム

擬似コードで表現すると、こんな感じ:

```
count ← 0           # 往復回数を初期化
while N > 0:         # ダンボールがまだある間
    if N == 1        # 残りが 1 個のとき
        N ← N - 1    # その 1 個を運ぶ
    else              # 2 個以上残っているとき
        N ← N - 2    # 2 個まとめて運ぶ
    count ← count + 1 # 往復回数を 1 増やす
```

A - さらに考えると

ダンボールの個数 N が……

偶数なら

毎回 2 個運べば、 $N / 2$ 回の往復で運びきれる。

奇数なら

はじめの $(N - 1) / 2$ 回では 2 個運び、

残った 1 個を最後の 1 回で運べばいい。

A - さらに考えると

ダンボールの個数 N が……

偶数なら

毎回 2 個運べば、 $N / 2$ 回の往復で運びきれ。

奇数なら

はじめの $(N - 1) / 2$ 回では 2 個運び、
残った 1 個を最後の 1 回で運べばいい。

A - さらに考えると

ダンボールの個数 N が……

偶数なら

毎回 2 個運べば、 $N/2$ 回の往復で運びきれる。

奇数なら

はじめの $(N - 1) / 2$ 回では 2 個運び、
残った 1 個を最後の 1 回で運べばいい。

つまり、往復回数は $N/2$ の切り上げ！

A - 計算量について

- ・はじめに考えたアルゴリズムは $O(N)$ 時間
- ・ $N/2$ の切り上げを計算するだけなら $O(1)$ 時間

今回は N が小さいのでどちらを用いても解けます。

より難しい問題に挑戦するときには、自分の考えたアルゴリズムの計算量を見積もれるようになっていないと厳しいです。

少しずつ慣れていき、計算量を常に考える癖をつけていきましょう。

B問題

心配性な富豪、
ファミリーレストランに行く。

B - 問題概要

問題

ファミレスのメニューに N 種類の料理があり、それぞれの料理の値段がわかっている。

2 番目に高い料理の値段はいくらか？

制限

$$1 \leq N \leq 100$$

すべての料理の値段が同じであることはない。

B - 解く前に確認

この問題で必要なこと

- ・ 配列、リスト等の基本的な取り扱い
- ・ ソート（整列）の方法

配列の基礎については、

またもや過去スライドをチェック！

<http://www.slideshare.net/chokudai/abc004>

B - ソート (整列)

ソートとは……

配列などの要素を、ある順番に従って並び替える

たとえば [51, 39, 66, 42, 10] という配列を

昇順 (小さい方から順番) にソート

→ [10, 39, 42, 51, 66]

降順 (大きい方から順番) にソート

→ [66, 51, 42, 39, 10]

B - ソートアルゴリズム

アルゴリズムやプログラミングの本では
たくさんのソートアルゴリズムが紹介されている

挿入ソート、選択ソート、バブルソート、クイックソート、マージソート、...

B - ソートアルゴリズム

アルゴリズムやプログラミングの本では
たくさんのソートアルゴリズムが紹介されている

挿入ソート、選択ソート、バブルソート、クイックソート、マージソート、…

しかし実際にソートをしたいときにこれらを
自分で実装する必要は（ふつうは）ない。なぜか？
→ **言語の標準ライブラリ**に入っていることが多い！

もちろん、ソートアルゴリズムを学んだり、同じ目的のアルゴリズムでも手法によって計算量などの特徴が違ふことを理解したりするために、自分で色んなソートアルゴリズムを調べて実装してみるのはよいこと。

B - ソートアルゴリズム

各言語のライブラリまで全部説明はできないので……



Python ソート



Google 検索

I'm Feeling Lucky

各自で調べましょう！

B - 本題、解き方

値段が高い順にソートして、
その 2 番目を見るだけでは？

B - 注意

値段が高い順にソートして、
その 2 番目を見るだけでは？

ダメです

値段が 100 円、200 円、300 円、300 円するとき
2 番目に高いのは 200 円（300 円ではない）

問題文にもちゃんと書かれていました

B - 解き方

値段の配列を

10	50	90	50	80	90	30
----	----	----	----	----	----	----

B - 解き方

値段の配列を

10	50	90	50	80	90	30
----	----	----	----	----	----	----

とりあえず降順にソートして

90	90	80	50	50	30	10
----	----	----	----	----	----	----

B - 解き方

値段の配列を

10	50	90	50	80	90	30
----	----	----	----	----	----	----

とりあえず降順にソートして

90	90	80	50	50	30	10
----	----	----	----	----	----	----

1番高い料理たち

2番目に高い料理
これが知りたい

B - 解き方

値段の配列を

10	50	90	50	80	90	30
----	----	----	----	----	----	----

1番高い料理たちの次に出てくるものを求めればいい！

とりあえず降順にソートして

90	90	80	50	50	30	10
----	----	----	----	----	----	----

1番高い料理たち

2番目に高い料理
これが知りたい

B - プログラム

今回も擬似コードにしてみると、こんな感じ:

```
# N は料理の種類数
# A は値段の配列

X ← sort(A)           # A を降順にソートする
for i = 0 to N - 1    # 先頭から順に見ていく
    if X[i] != X[0]    # 1 番高い料理でないなら
        answer ← X[i] # それを答えにして
        break         # ループを抜け出す
```

B - 解き方その2

値段が高い順にソートして、
その 2 番目を見るだけでは？

この考えはそのままではダメだった。

B - 解き方その2

値段が高い順にソートして、
その 2 番目を見るだけでは？

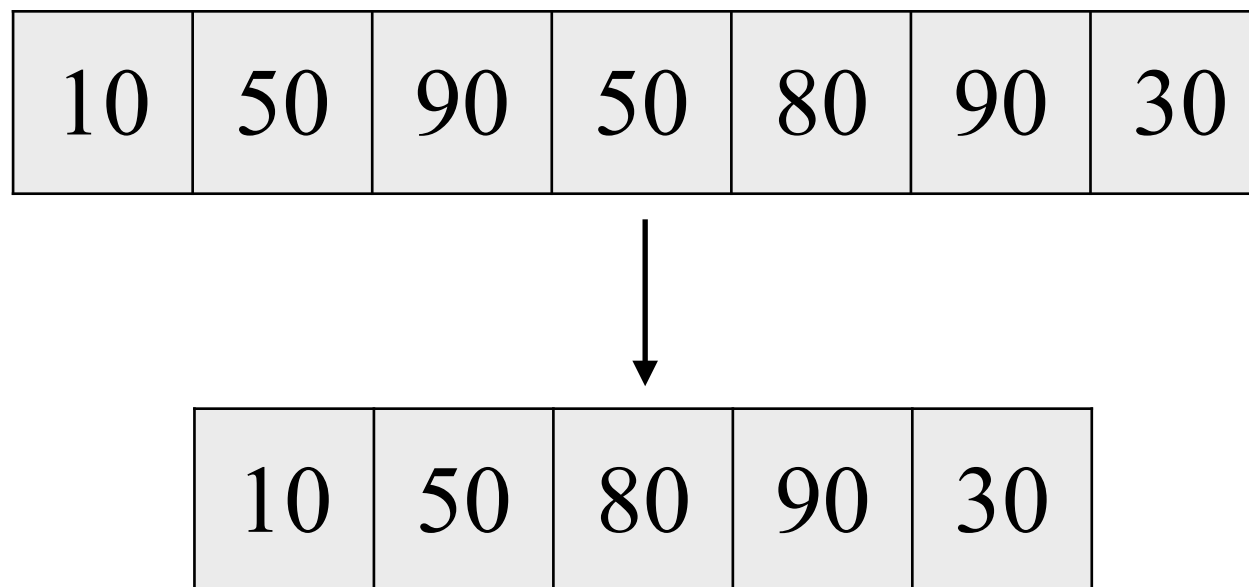
この考えはそのままではダメだった。

しかし、

同じ値段の料理がない時にはこれでもうまくいく！

B - 解き方その2

なら、**同じ値段の料理をなくしてしまえばいい！**



先にこういった処理を行っておけば、
降順ソートして 2 番目を見るだけで OK

B - 重複削除アルゴリズム

標準ライブラリにある言語も多いはず



Ruby 重複 削除



Google 検索

I'm Feeling Lucky

C問題

辞書式順序ふたたび

C - 問題概要

問題

文字列 S を並び替えて文字列 T を作る。
ただし動いた文字の個数が K 以下でなければダメ。
そのような制限のもとで、辞書順最小の T は？

制限

$$1 \leq N \text{ (} S \text{ の文字数)} \leq 100$$

$$1 \leq K \leq N$$

C - 辞書順最小のアイデア

基本は、**問題文にあったヒント**のように実装

読んでない場合は http://abc009.contest.atcoder.jp/tasks/abc009_3

ヒントの繰り返しですが、重要なのは

- ・ **最初のほうに小さいアルファベットを持ってくる**
- ・ あるアルファベットを持ってきたときに、
制限を満たせるかどうかを判定する

の 2 点です。

C - プログラムのイメージ

今回も擬似コードでイメージを掴みましょう:

```
T ← “”  
for i = 0 to N - 1  
  for c in (まだ使える文字を小さい順に)  
    if (T + c にして大丈夫なら)  
      T ← T + c  
    break
```

C - プログラムのイメージ

今回も擬似コードでイメージを掴みましょう:

```
T ← “”  
for i = 0 to N - 1  
    for c in (まだ使える文字を小さい順に)  
        if (T + c にして大丈夫なら)  
            T ← T + c  
            break
```

括弧書きでごまかした部分をちゃんと考える！

C - まだ使える文字を小さい順に

T を先頭から 1 文字ずつ決めていくが、
その途中で「**まだ使える文字**」とは何か？

$S = \text{“program”}$ で

$T = \text{“aro”}$ と最初の 3 文字だけ決まっているとき

C - まだ使える文字を小さい順に

T を先頭から 1 文字ずつ決めていくが、
その途中で「まだ使える文字」とは何か？

$S = \text{“program”}$ で

$T = \text{“aro”}$ と最初の 3 文字だけ決まっているとき

p, g, m は T にまだ入っていないので使える

C - まだ使える文字を小さい順に

T を先頭から 1 文字ずつ決めていくが、
その途中で「まだ使える文字」とは何か？

$S = \text{“program”}$ で

$T = \text{“aro”}$ と最初の 3 文字だけ決まっているとき

p, g, m は T にまだ入っていないので使える

a, o は T に入っているのもう使えない

C - まだ使える文字を小さい順に

T を先頭から 1 文字ずつ決めていくが、
その途中で「まだ使える文字」とは何か？

$S = \text{“p} \color{red}{r} \text{og} \color{red}{r} \text{am”}$ で

$T = \text{“a} \color{red}{r} \text{o”}$ と最初の 3 文字だけ決まっているとき

p, g, m は T にまだ入っていないので使える

a, o は T に入っているのもう使えない

$\color{red}{r}$ は S に 2 回出てきており、
 T には 1 回しか出てきていないのでまだ使える

C - まだ使える文字を小さい順に

T を先頭から 1 文字ずつ決めていくが、

その途中ですべての文字が使われてしまったら、

4 文字目の候補は p, g, m, r の 4 通り！

$S = \text{“program”}$ で

$T = \text{“aro”}$ と最初の 3 文字だけ決まっているとき

p, g, m は T にまだ入っていないので使える

a, o は T に入っているのもう使えない

r は S に 2 回出てきており、
 T には 1 回しか出てきていないのでまだ使える

$C - T + c$ にして大丈夫？

つまり、 T の次の文字を c にしてしまった結果、
このあとどう頑張っても動いた文字の個数が K を
超えてしまう……、ということにならないだろうか？

$C - T + c$ にして大丈夫？

つまり、 T の次の文字を c にしてしまった結果、
このあとどう頑張っても動いた文字の個数が K を
超えてしまう……、ということにならないだろうか？



**残りの部分でできるだけ頑張って、
動いた文字が少なくなるようにしてみる！**
その結果動いた文字の個数が K 以下にできれば OK

$C - T + c$ にして大丈夫？

$S = \text{“program”}$ 、 $K = 3$ で、

$T = \text{“aro”}$ と最初の 3 文字だけ決まっているとき

$T = \text{“arog”}$ にしても大丈夫か？

C - T + c にして大丈夫？

$S = \text{“program”}$ 、 $K = 3$ で、

$T = \text{“aro”}$ と最初の 3 文字だけ決まっているとき

$T = \text{“arog”}$ にしても大丈夫か？

S	p	r	o	g	r	a	m
T	a	r	o	g	*	*	*

すでに決まっている部分
(不一致 1 文字)

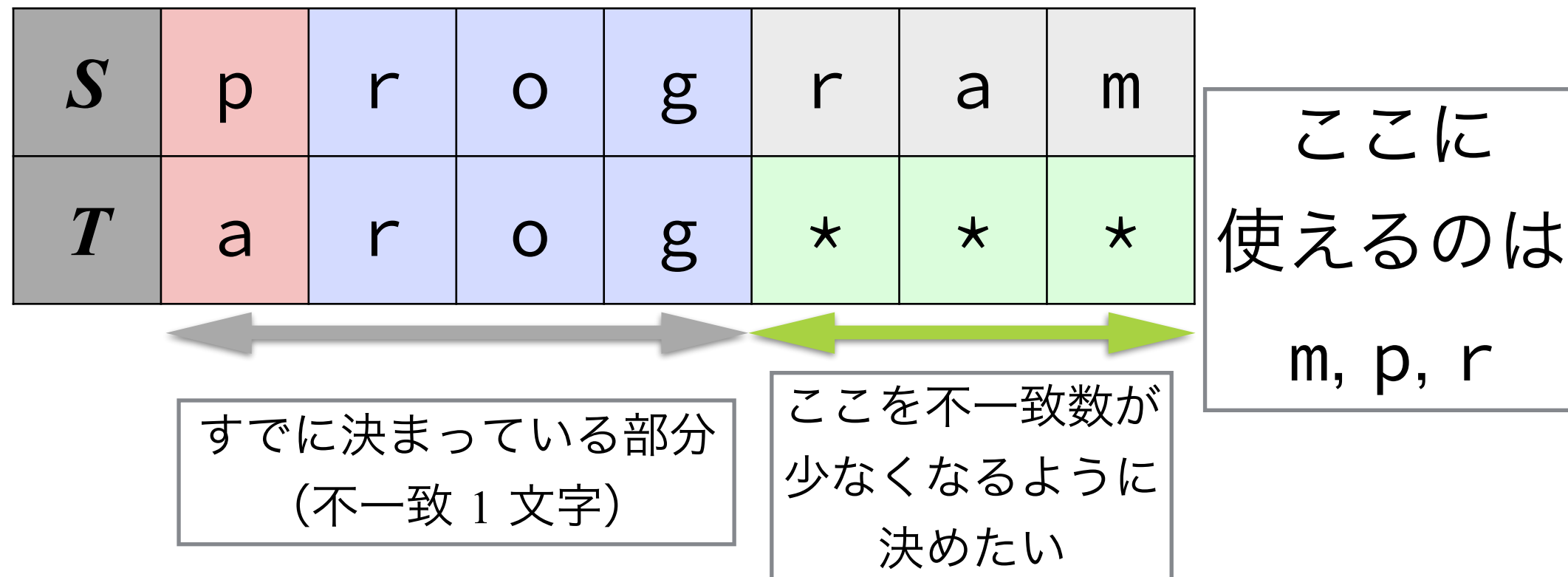
ここを不一致数が
少なくなるように
決めたい

C - T + c にして大丈夫？

$S = \text{“program”}$ 、 $K = 3$ で、

$T = \text{“aro”}$ と最初の 3 文字だけ決まっているとき

$T = \text{“arog”}$ にしても大丈夫か？



C - T + c にして大丈夫？

$S = \text{"program"}$ 、 $K = 3$ で、

$T = \text{"aro"}$ と最初の 3 文字だけ決まっているとき

$T = \text{"arog"}$ にしても大丈夫か？

<i>S</i>	p	r	o	g	r	a	m
<i>T</i>	a	r	o	g	*	*	*

すでに決まっている部分
(不一致 1 文字)

ここを不一致数が
少なくなるように
決めたい

m, p, r をうまく
並び替えて
“ram” との
不一致数を
最小にすればよい！

C - 結局のところ

問題

同じ長さの文字列 s, t が与えられる。

t を並び替えて、 s との不一致の数をどれだけ少なくできるか？

これが解ければいい。

前ページまでの例だと $s = \text{“ram”}$, $t = \text{“mpr”}$

C - 具体例で

$s = \text{“pipeline”}$, $t = \text{“eppstein”}$ の場合を考える

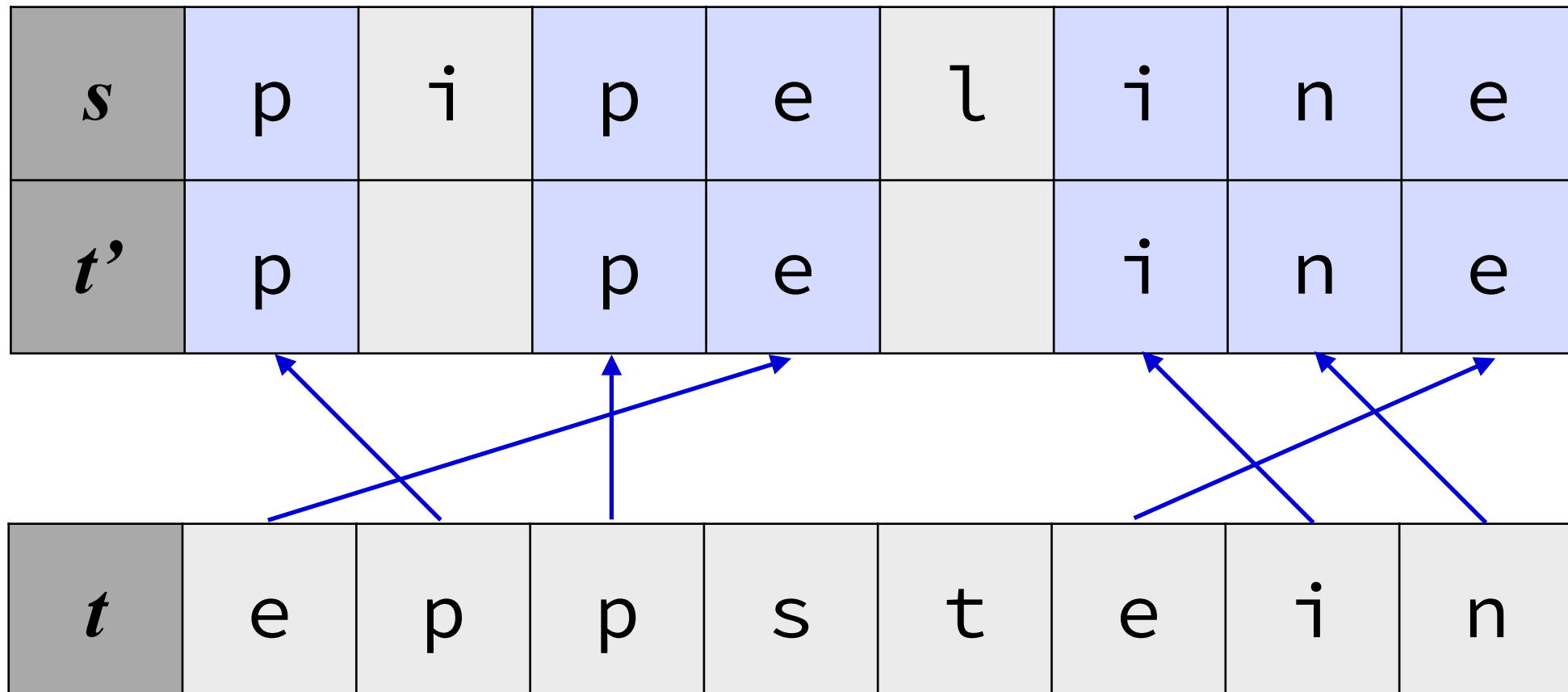
s	p	i	p	e	l	i	n	e
t'								

t' は並び替え後の t を表すとする

t	e	p	p	s	t	e	i	n
-----	---	---	---	---	---	---	---	---

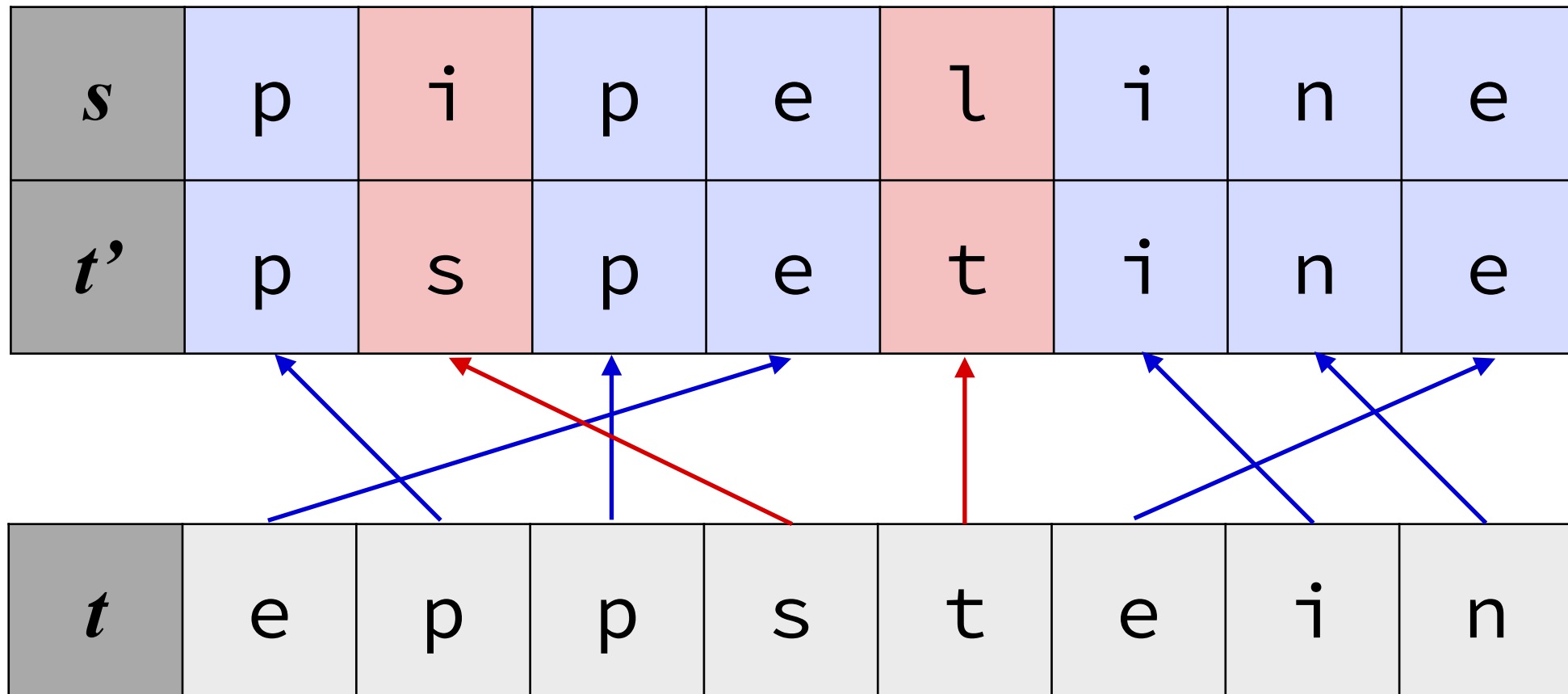
C - 直感的には

同じ文字のところに持ってくれば不一致は減りそう



C - 直感的には

残りの文字はどう並び替えても不一致になる



C - 実は

今の直感的な方法が**最適**になる

- ・ t の文字を順に見ていき、
 - ・ s に**同じ文字がありそこが空いてれば**持っていく
 - ・ **なければ**後回し
- ・ 残った t の文字を適当な空いた場所に持っていく

で OK

C - なぜか？

s に ‘a’ が n 個、 t に ‘a’ が m 個あるとする。

このとき、 $\min(n, m)$ 個の ‘a’ は
並び替えによって一致させられるが、
残りの ‘a’ はどうしても不一致になる。

さっきの直感的な方法では、
 $\min(n, m)$ 個ぴったり一致させることができる！

C - ということは

この問題には意外と簡単に答えられる

同じ長さの文字列 s, t が与えられる。
 t を並び替えて、 s との不一致の数をどれだけ少なくできるか？

→ $\min(s \text{ にある } a \text{ の数}, t \text{ にある } a \text{ の数})$
+ $\min(s \text{ にある } b \text{ の数}, t \text{ にある } b \text{ の数})$
+ ...
+ $\min(s \text{ にある } z \text{ の数}, t \text{ にある } z \text{ の数})$

が答え！

C - お疲れ様でした

以上で、はじめの擬似コードでごまかした部分をきちんとプログラムで計算できるようになった。

これで C 問題が解ける。

N の値を小さめにしているため、多少の実装方針の差異で計算量が違っていても時間には余裕があると思います。

D問題

漸化式

D - 問題概要

問題

整数列 A は最初の K 項が与えられ、
それ以降の項は与えられた漸化式で決まる。
 A の M 項目の値を求めよ。

制限

$$1 \leq K \leq 100$$

$$1 \leq M \leq 1,000,000,000 (=10^9)$$

D - 漸化式

そもそも漸化式とは？

数列の各項の値が、**それ以前の項の値**によって
決められるとき、その決め方を表す等式

例: フィボナッチ数列

$$F_{n+2} = F_{n+1} + F_n \text{ — 漸化式}$$

$$F_0 = 0$$

$$F_1 = 1$$

—— 初期値

D - 漸化式の計算

漸化式はそのまま簡単にプログラムにできる
(フィボナッチ数列の例)

```
# 初期値を入れる
```

```
F[0] ← 0
```

```
F[1] ← 1
```

```
# 漸化式に従って前から順に計算
```

```
for k = 2 to N
```

```
    F[k] ← F[k-1] + F[k-2]
```

D - 漸化式の計算

漸化式はそのまま簡単にプログラムにできる
(フィボナッチ数列の例)

N 項目を計算するには
**漸化式に従った計算を
 N 回ぐらいしないといけない**

```
# 初期値を  
F[0] ← 0  
F[1] ← 1
```

漸化式に従って前から順に計算

```
for k = 2 to N  
    F[k] ← F[k-1] + F[k-2]
```

D - 今回の場合

漸化式

$$A_{N+K} = (C_1 \text{ AND } A_{N+K-1}) \text{ XOR } \cdots \text{ XOR } (C_K \text{ AND } A_N)$$

D - 今回の場合

漸化式

$$A_{N+K} = (C_1 \text{ AND } A_{N+K-1}) \text{ XOR } \cdots \text{ XOR } (C_K \text{ AND } A_N)$$

長いので、AND と XOR をこう書き換えます↓

$$A_{N+K} = (C_1 \cdot A_{N+K-1}) \oplus \cdots \oplus (C_K \cdot A_N)$$

D - 今回の場合

漸化式

$$A_{N+K} = (C_1 \cdot A_{N+K-1}) \oplus \cdots \oplus (C_K \cdot A_N)$$

漸化式に従って次の項を計算するときに

AND や XOR を K 回ぐらい計算する必要がある

D - 今回の場合

漸化式

$$A_{N+K} = (C_1 \cdot A_{N+K-1}) \oplus \cdots \oplus (C_K \cdot A_N)$$

漸化式に従って次の項を計算するときに

AND や XOR を K 回ぐらい計算する必要がある

つまり、 M 項目を計算するためには

$M \times K (\leq 10^{11})$ 回ぐらいの計算が必要になる！

とてもじゃないけど 2 秒では……

D - 高速な漸化式の計算

高速に M 項目を求めるために……

D - 高速な漸化式の計算

高速に M 項目を求めるために……

行列

が使えます！

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 4 & 1 & 5 \\ 2 & 2 & 1 \end{pmatrix}$$

← こういうやつ

D - 行列について

今回使うのは

- ・ 行列とベクトルの積
 - ・ 行列と行列の積
- です。

ここで頑張って説明するよりも、
ネットにたくさんある分かりやすい解説を
見たほうがよいと思うので……



行列 積



Google 検索

I'm Feeling Lucky

D - 高速な漸化式の計算

フィボナッチ数列を例にして考えます

$$F_{n+2} = F_{n+1} + F_n$$

$$F_0 = 0$$

$$F_1 = 1$$

フィボナッチ数列は F_n と F_{n+1} の
2 個だけわかっているならば、 F_{n+2} の値もわかる！

D - 高速な漸化式の計算

はじめは F_0, F_1 の値を知っている

D - 高速な漸化式の計算

はじめは F_0, F_1 の値を知っている

→ F_2 が計算できるので、 F_1, F_2 の値がわかる

D - 高速な漸化式の計算

はじめは F_0, F_1 の値を知っている

→ F_2 が計算できるので、 F_1, F_2 の値がわかる

→ F_3 が計算できるので、 F_2, F_3 の値がわかる

D - 高速な漸化式の計算

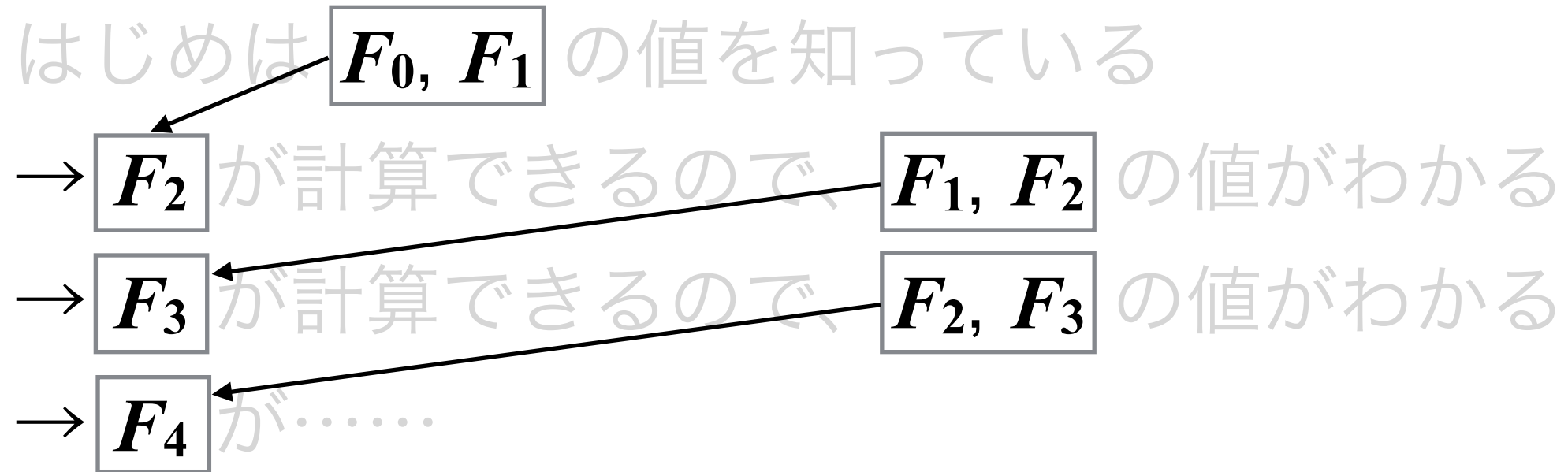
はじめは F_0, F_1 の値を知っている

→ F_2 が計算できるので、 F_1, F_2 の値がわかる

→ F_3 が計算できるので、 F_2, F_3 の値がわかる

→ F_4 が.....

D - 高速な漸化式の計算



漸化式の計算を 2 個まとまりで考えよう

$$\begin{pmatrix} F_1 \\ F_0 \end{pmatrix} \longrightarrow \begin{pmatrix} F_2 \\ F_1 \end{pmatrix} \longrightarrow \begin{pmatrix} F_3 \\ F_2 \end{pmatrix} \longrightarrow \begin{pmatrix} F_4 \\ F_3 \end{pmatrix} \longrightarrow \dots\dots$$

D - 高速な漸化式の計算

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} \rightarrow \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix}$$

この矢印は結局何をしているのか？

D - 高速な漸化式の計算

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} \rightarrow \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix}$$

この矢印は結局何をしているのか？

実は、**行列を掛けている！**

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} F_{n+1} + F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix}$$

D - 高速な漸化式の計算

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

この行列を 1 回かけると、添字が 1 つ進む


D - 高速な漸化式の計算

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

この行列を 1 回かけると、添字が 1 つ進む

なら、 **n 回かけると、添字が n 進む！！**

$$\begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \cdots \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix}$$


 n 個

D - 高速な漸化式の計算

F_n を求めるためには

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

を計算すればいい (**行列の累乗**) ことが分かった

普通だと**行列の掛け算を n 回**しないといけない
=全然早くなってない！

D - 高速な累乗の計算

ある行列 A の n 乗を求めたいとする
(例として $n = 100$ の場合を考える)

$$A^{100}$$

D - 高速な累乗の計算

100 を 2 進数で表すと 1100100 になる

$$A^{100} =$$

$$100 = 64 \quad 32 \quad 16 \quad 8 \quad 4 \quad 2 \quad 1$$

つまり $100 = 64 + 32 + 4$ と書ける

D - 高速な累乗の計算

$A^{100} = A^{64} A^{32} A^4$ と同じように分解できる！

$$A^{100} = A^{64} A^{32} A^{16} A^8 A^4 A^2 A^1$$

$$100 = 64 \quad 32 \quad 16 \quad 8 \quad 4 \quad 2 \quad 1$$

A の (2 の累乗) 乗が求められればよさそう

D - 高速な累乗の計算

これは A を次々に 2 乗していけば計算できる

$$A^{100} = \overset{\boxed{2\text{乗}}}{A^{64} \leftarrow A^{32}} \overset{\boxed{2\text{乗}}}{\leftarrow A^{16}} \overset{\boxed{2\text{乗}}}{\leftarrow A^8} \overset{\boxed{2\text{乗}}}{\leftarrow A^4} \overset{\boxed{2\text{乗}}}{\leftarrow A^2} \overset{\boxed{2\text{乗}}}{\leftarrow A^1}$$

$$100 = 64 \quad 32 \quad 16 \quad 8 \quad 4 \quad 2 \quad 1$$

A を 2 乗していきながら、 n のビットが立っている場所のものだけを掛け算していけばいい！

D - 高速な累乗の計算

$$A^{100} = \overset{\boxed{\text{2乗}}}{A^{64} \leftarrow A^{32}} \overset{\boxed{\text{2乗}}}{\leftarrow A^{16}} \overset{\boxed{\text{2乗}}}{\leftarrow A^8} \overset{\boxed{\text{2乗}}}{\leftarrow A^4} \overset{\boxed{\text{2乗}}}{\leftarrow A^2} \overset{\boxed{\text{2乗}}}{\leftarrow A^1}$$

$$100 = 64 \quad 32 \quad 16 \quad 8 \quad 4 \quad 2 \quad 1$$

必要な行列の掛け算の回数は、

n を 2 進数で表したときの桁数 $= O(\log n)$

D - ようやく解き方

$O(\log n)$ 回の行列の掛け算で漸化式の
 n 項目が計算できるようになった！

これを使えば D 問題が解けるのでは？

D - ようやく解き方

$O(\log n)$ 回の行列の掛け算で漸化式の
 n 項目が計算できるようになった！

これを使えば D 問題が解けるのでは？

ん？ちょっと待てよ……

D - ようやく解き方

今回の漸化式

$$A_{N+K} = (C_1 \cdot A_{N+K-1}) \oplus \cdots \oplus (C_K \cdot A_N)$$

行列の累乗では、

掛け算や足し算で表される漸化式が対象だった

→ AND や XOR はどうすればいいのか？？？

D - ようやく解き方

今回の漸化式

$$A_{N+K} = (C_1 \cdot A_{N+K-1}) \oplus \cdots \oplus (C_K \cdot A_N)$$

行列の累乗では、

掛け算や足し算で表される漸化式が対象だった

→ AND や XOR はどうすればいいのか???

実は大丈夫！行列累乗が使える！！

D - ようやく解き方

- ・ AND を掛け算のようなもの
- ・ XOR を足し算のようなもの

と思って行列の累乗を計算すればいい！

(つまり、普通の行列の計算をするときの $+$ を XOR に置き換えて、 \times を AND に置き換える)

D - ようやく解き方

- ・ AND を掛け算のようなもの
- ・ XOR を足し算のようなもの

と思って行列の累乗を計算すればいい！

(つまり、普通の行列の計算をするときの $+$ を AND に置き換えて、 \times を XOR に置き換える)

なんでそんなことをしても大丈夫なの？

「掛け算のようなもの」ってどういうこと？

D - 行列積・累乗に必要なこと

行列の要素は必ずしも

- ・ **普通の数**
- ・ **普通の足し算**
- ・ **普通の掛け算**

である必要はない！

とはいえ、足し算や掛け算のかわりに
何でも好きな演算をいれていいわけでもない。
足し算と掛け算っぽいものじゃないとダメ

D - 行列積・累乗に必要なこと

足し算と掛け算っぽさとは……？

- ・ 足し算ならこれが成り立ってほしいよね
- ・ 掛け算ならこれが成り立ってほしいよね

という性質。

今からその性質を挙げますが、結構数が多いので覚えるというよりは「こういう感じなのかー」と理解してもらえればいいと思います。

D - 行列積・累乗に必要なこと

以下の性質を満たしていれば行列積ができる！

$$a + (b + c) = (a + b) + c$$

$$a + 0 = 0 + a = a$$

$$a + b = b + a$$

$$a \times (b \times c) = (a \times b) \times c$$

$$a \times 1 = 1 \times a = a$$

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(a + b) \times c = (a \times c) + (b \times c)$$

$$a \times 0 = 0 \times a = 0$$

D - 行列積・累乗に必要なこと

以下の性質を満たしていれば行列積ができる！

$$a + (b + c) = (a + b) + c$$

$$a + 0 = 0 + a = a$$

$$a + b = b + a$$

足し算は好きな順に
計算してもいい

$$a \times (b \times c) = (a \times b) \times c$$

$$a \times 1 = 1 \times a = a$$

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(a + b) \times c = (a \times c) + (b \times c)$$

$$a \times 0 = 0 \times a = 0$$

D - 行列積・累乗に必要なこと

以下の性質を満たしていれば行列積ができる！

$$a + (b + c) = (a + b) + c$$

$$a + 0 = 0 + a = a$$

$$a + b = b + a$$

足し算しても変わらない
ような値(0)が存在する

$$a \times (b \times c) = (a \times b) \times c$$

$$a \times 1 = 1 \times a = a$$

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(a + b) \times c = (a \times c) + (b \times c)$$

$$a \times 0 = 0 \times a = 0$$

D - 行列積・累乗に必要なこと

以下の性質を満たしていれば行列積ができる！

$$a + (b + c) = (a + b) + c$$

$$a + 0 = 0 + a = a$$

$$a + b = b + a$$

足し算は左右を
入れ替えても結果が同じ

$$a \times (b \times c) = (a \times b) \times c$$

$$a \times 1 = 1 \times a = a$$

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(a + b) \times c = (a \times c) + (b \times c)$$

$$a \times 0 = 0 \times a = 0$$

D - 行列積・累乗に必要なこと

以下の性質を満たしていれば行列積ができる！

$$a + (b + c) = (a + b) + c$$

$$a + 0 = 0 + a = a$$

$$a + b = b + a$$

$$a \times (b \times c) = (a \times b) \times c$$

$$a \times 1 = 1 \times a = a$$

掛け算は好きな順に
計算してもいい

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(a + b) \times c = (a \times c) + (b \times c)$$

$$a \times 0 = 0 \times a = 0$$

D - 行列積・累乗に必要なこと

以下の性質を満たしていれば行列積ができる！

$$a + (b + c) = (a + b) + c$$

$$a + 0 = 0 + a = a$$

$$a + b = b + a$$

$$a \times (b \times c) = (a \times b) \times c$$

$$a \times 1 = 1 \times a = a$$

掛け算しても変わらない
ような値(1)がある

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(a + b) \times c = (a \times c) + (b \times c)$$

$$a \times 0 = 0 \times a = 0$$

D - 行列積・累乗に必要なこと

以下の性質を満たしていれば行列積ができる！

$$a + (b + c) = (a + b) + c$$

$$a + 0 = 0 + a = a$$

$$a + b = b + a$$

$$a \times (b \times c) = (a \times b) \times c$$

$$a \times 1 = 1 \times a = a$$

$$\begin{aligned} a \times (b + c) &= (a \times b) + (a \times c) \\ (a + b) \times c &= (a \times c) + (b \times c) \end{aligned}$$

足し算と掛け算の間に
分配法則がなりたつ

$$a \times 0 = 0 \times a = 0$$

D - 行列積・累乗に必要なこと

以下の性質を満たしていれば行列積ができる！

$$a + (b + c) = (a + b) + c$$

$$a + 0 = 0 + a = a$$

$$a + b = b + a$$

$$a \times (b \times c) = (a \times b) \times c$$

$$a \times 1 = 1 \times a = a$$

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(a + b) \times c = (a \times c) + (b \times c)$$

$$a \times 0 = 0 \times a = 0$$

足し算しても変わらない
ような値(0)と掛け算すると
0になる

D - 半環

いま挙げた性質を満たすものを**半環**といいます。

足し算を XOR、掛け算を AND だと思っても
いま挙げた性質が全部成り立つことが確かめられる！
→**行列の累乗で漸化式の計算ができる！！**

実際に性質を満たすことを確かめるのは省略しますが、余裕があればトライしてみてください。

また、今回のように非負整数に対して「足し算を XOR」「掛け算を AND」とすれば性質を満たすことを「非負整数は XOR と AND に関して半環をなす」と言ったりします。

D - 解き方に戻りまして

XOR, AND からなる漸化式でも行列累乗が
使えることが無事に確認できた！

あとは、今回の漸化式を進めるような行列を
実際に作って累乗すればいい！

D - 行列の構成

こういう行列になります

$$\begin{pmatrix} C_1 & C_2 & \cdots & C_{K-1} & C_K \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix}$$

D - 行列のイメージ

1 行目は、掛け算したときに漸化式が出てくるように

$$\begin{pmatrix} C_1 & C_2 & \cdots & C_{K-1} & C_K \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} A_{N+K-1} \\ A_{N+K-2} \\ A_{N+K-3} \\ \vdots \\ A_N \end{pmatrix}$$

→ 掛け算の結果、漸化式が出てくる

→ 結果の 1 個目が A_{N+K} になる

D - 行列のイメージ

2 行目以降は、項をずらしていくイメージ

$$\begin{pmatrix} C_1 & C_2 & \cdots & C_{K-1} & C_K \\ \boxed{1} & \boxed{0} & \cdots & \boxed{0} & \boxed{0} \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} A_{N+K-1} \\ A_{N+K-2} \\ A_{N+K-3} \\ \vdots \\ A_N \end{pmatrix}$$

たとえば、2 行目に注目すると

掛け算の結果 2 個目には A_{N+K-1} が出てくる

D - まとめ

- ・ 漸化式は行列の累乗を使って早く進められる
- ・ 足し算、掛け算のかわりに XOR, AND を使う

先ほどの行列を $M-1$ 乗して A_1 から A_K までが入った初期値のベクトルに掛け算すると A_M が求まる！

行列の掛け算は定義通りに計算して $O(K^3)$ なので、全体での計算量は $O(K^3 \log M)$ になる。

D - 注意

どんな漸化式でも行列累乗が使えるわけではない！

$$X_{n+2} = X_{n+1} \times X_n^2$$

以前の項どうしの間で掛け算があったりするとダメ

$$X_{n+2} = aX_{n+1} + bX_n$$

こういう (定数)×(以前の項) の足し算の形ならOK

ただし、**足し算や掛け算が普通のものじゃなくても半環なら OK** というのが今回のポイントだった