SuperCon2022 本選問題：
オートマトンで文字列を区別しよう

2022.08.22

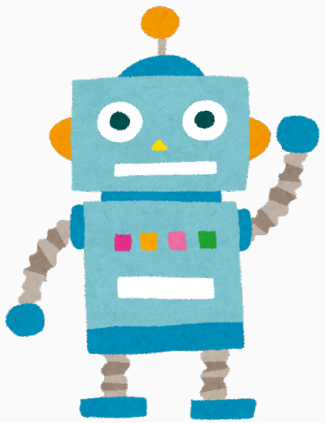
問題背景：オートマトン

オートマトンとは・・・

「（初期の状態を定めた上で）

文字列を入力すると、

受理（YES）または非受理（NO）を出力する機械」

入力		**オートマトン**		出力
bb	⟶		⟶	非受理
bba	⟶		⟶	非受理
bbab	⟶		⟶	受理
bbabb	⟶		⟶	非受理

問題背景：オートマトン

オートマトンとは・・・

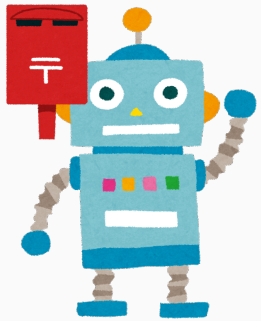
「（初期の状態を定めた上で）

文字列を入力すると、

受理（YES）または非受理（NO）を出力する機械」

オートマトンでできることの具体例：

「郵便番号の形式（[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9] の**正規表現**）にマッチする文字列か？」の判定

入力		**オートマトン**		出力
152-8550	⟶		⟶	受理
152	⟶		⟶	非受理
152-0033	⟶		⟶	受理
1520033	⟶		⟶	非受理

問題背景：オートマトン

オートマトンとは・・・

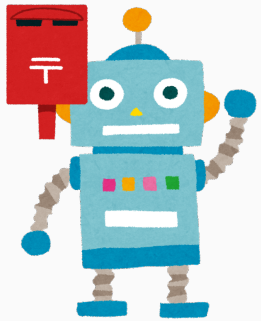
「（初期の状態を定めた上で）

文字列を入力すると、

受理（YES）または非受理（NO）を出力する機械」

オートマトンでできることの具体例：

「郵便番号の形式（[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9] の**正規表現**）にマッチする文字列か？」の判定

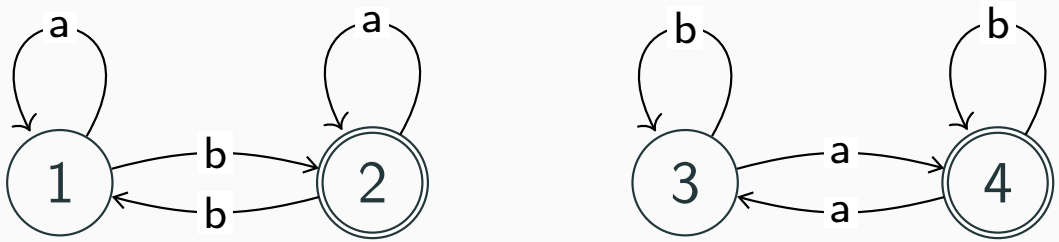
入力		**オートマトン**		出力
152-8550	⟶		⟶	受理
152	⟶		⟶	非受理
152-0033	⟶		⟶	受理
1520033	⟶		⟶	非受理

☞ このように、文字列を「受理」と「非受理」の２つに分類する

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
-



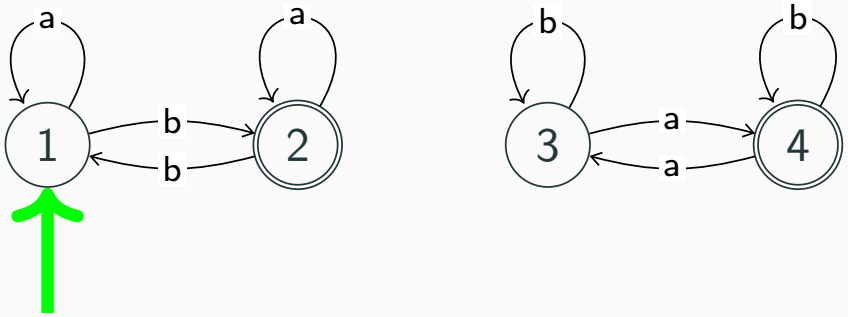
問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

入力	初期	オートマトン	終了	出力
bb	1			
bba	1			
bbab	1			
bbabb	1			

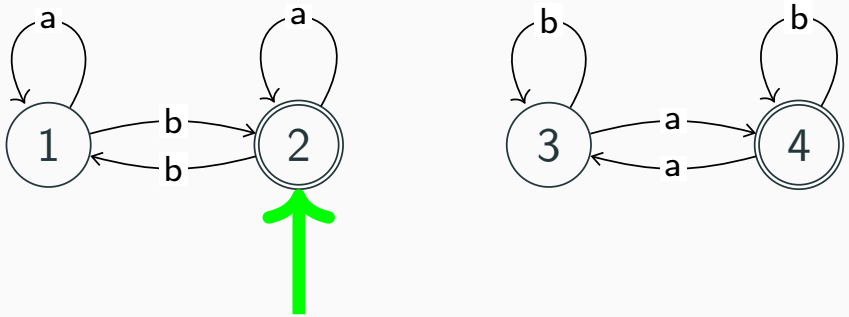


問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（○）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

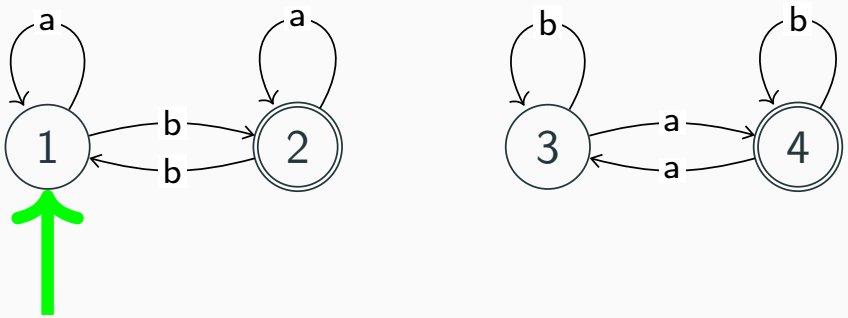
入力	初期	オートマトン	終了	出力
bbb	1			
bba	1			
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

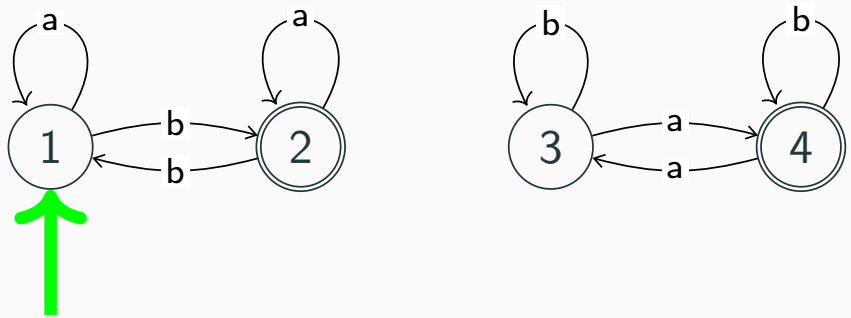
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1			
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

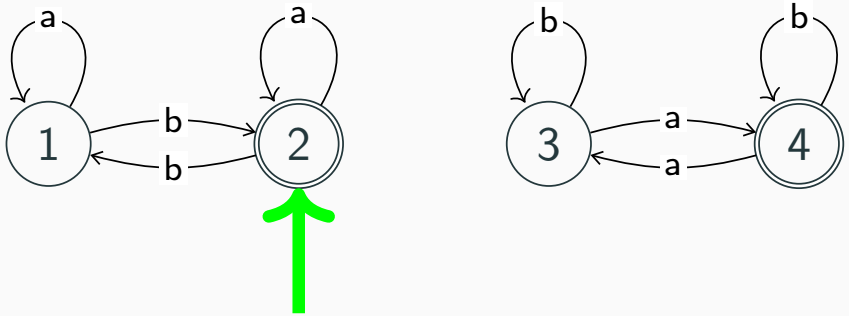
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1			
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（○）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

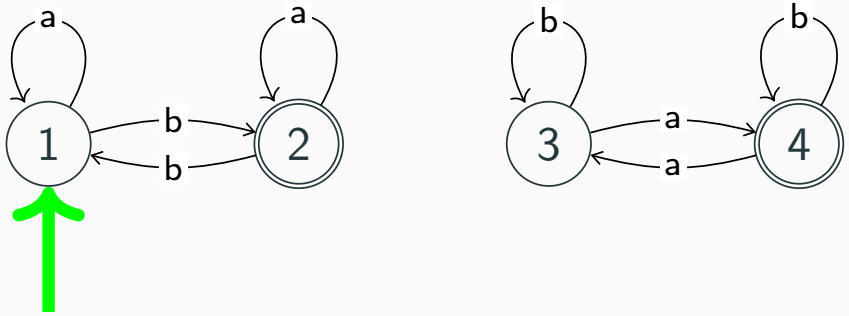
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1			
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

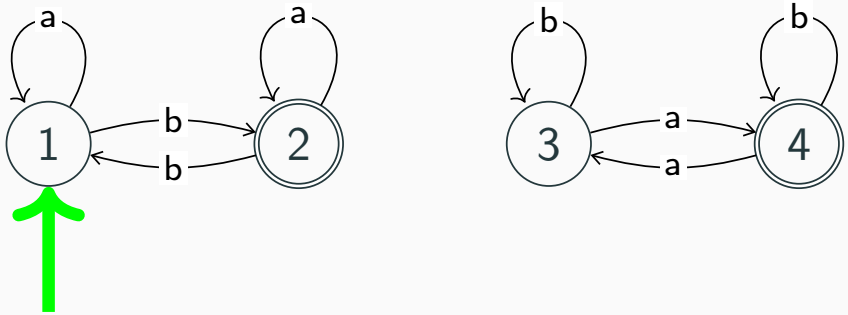
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1			
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（○）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

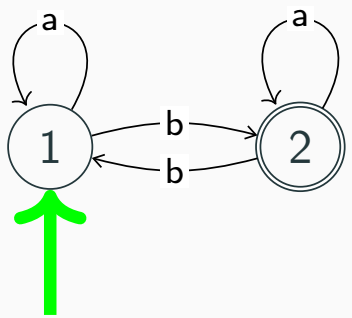
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

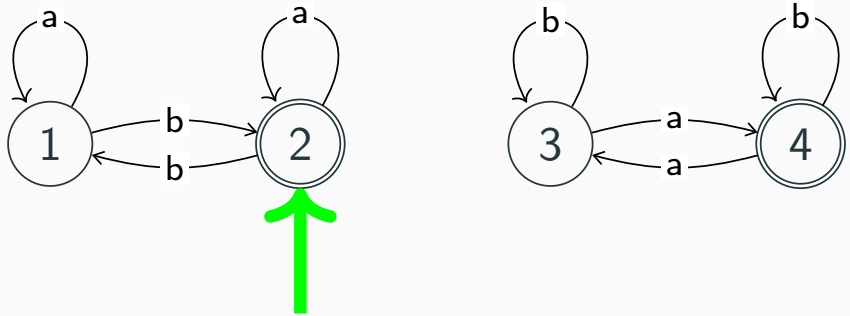
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

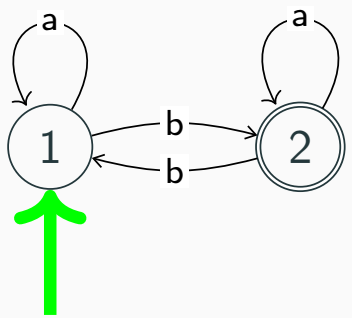
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（○）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

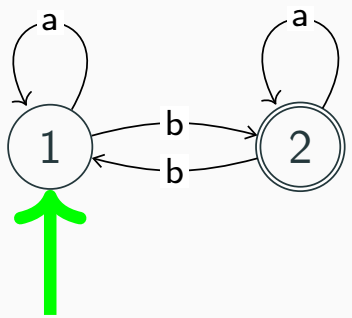
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（○）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

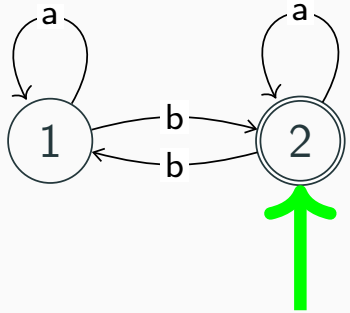
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bba{b	1			
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

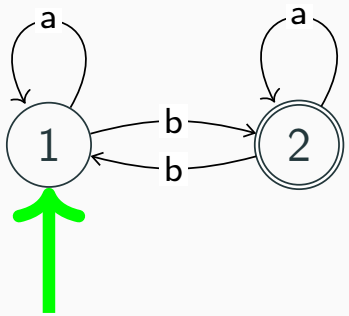
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

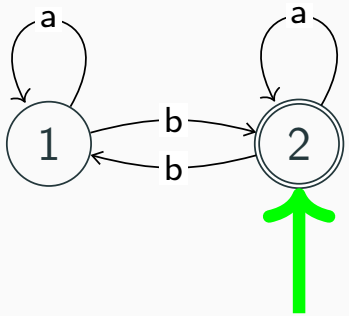
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
bbabb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（○）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

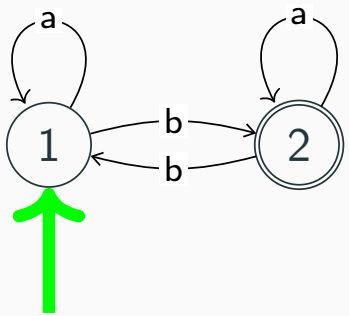
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
b{babb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

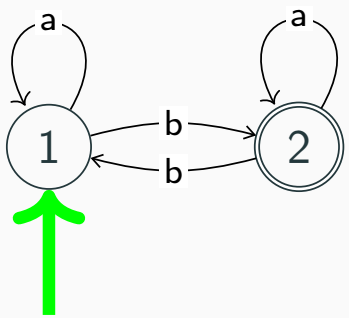
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
bb|abb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（○）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

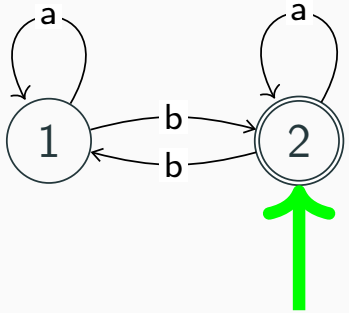
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
bba‖bb	1			

問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
bbab[b	1			

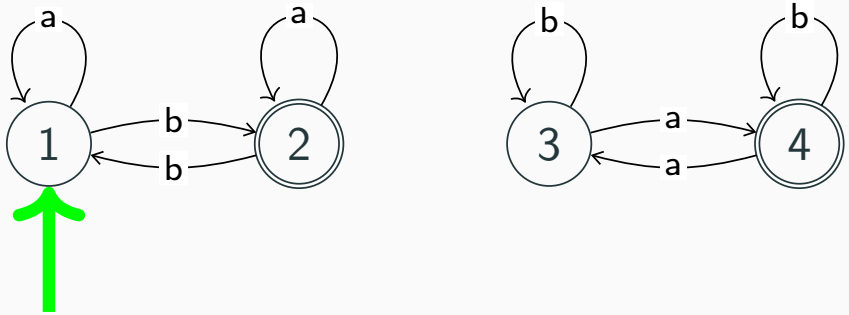
問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

（初期の状態が 1 の時、「　　」かつそのときにのみ「受理」）

入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
bbabb	1		1	非受理

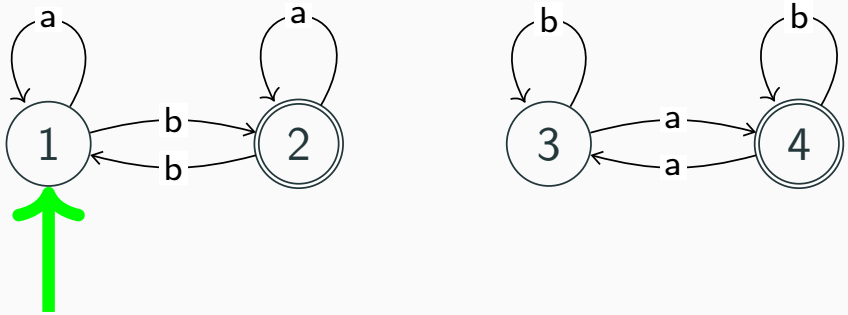
問題背景：オートマトン（遷移の方法、受理/非受理の決め方）

オートマトンは、以下のような図で表される。

- （正の整数でラベル付けされた）各頂点は、ある**状態**を表す。
- 二重丸 ◎ は、その状態が**受理状態**であることを表す。
- ラベル付けされた矢印は、それぞれの文字の**遷移関数**を表す。
- 出力（受理/非受理）は「初期の状態から**入力の文字列に従って遷移**した先が受理（◎）か非受理（〇）か？」で決まる。

文字列による遷移の例 （先頭から１文字ずつ読み、対応する矢印を順に辿っている）

（初期の状態が 1 の時、**「b が奇数回現れる」**かつそのときにのみ「受理」）

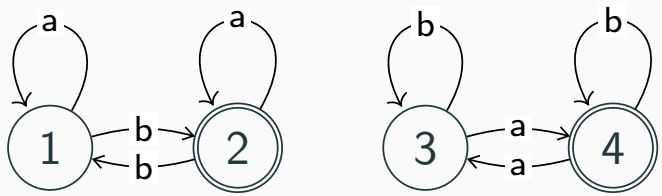
入力	初期	オートマトン	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
bbabb	1		1	非受理

問題背景：オートマトン（C 言語による動作の理解）

```c
#include <stdio.h>
int main(void) {
  const char* s = "bb";   // 入力の文字列
  goto q1;                // 初期状態
q1:
  if (*s == '\0') {
    printf("非受理\n");
    return 0;
  } else if (*s == 'a') {
    s++;
    goto q1;
  } else if (*s == 'b') {
    s++;
    goto q2;
  }
q2:
  if (*s == '\0') {
    printf("受理\n");
    return 0;
  } else if (*s == 'a') {
    s++;
    goto q2;
  } else if (*s == 'b') {
    s++;
    goto q1;
  }
q3:
  // ...(略)
q4:
  // ...(略)

  return 1;
}
```

← 下のオートマトンに対応する C 言語のプログラム（初期状態 1 入力の文字列 bb の場合）



問題背景：オートマトン（正確な定義）

オートマトンは、以下の 4 つ組 $\mathcal{A} = \langle Q, T_{\mathsf{a}}, T_{\mathsf{b}}, F \rangle$ である：

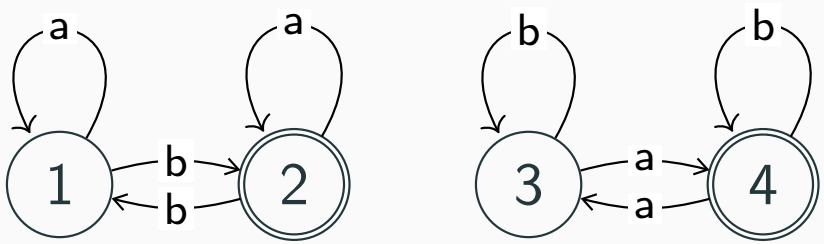
- ある n 以下の正整数の集合 $Q = \{1, 2, \ldots, n\}$（**状態**の集合）
- 関数 $T_{\mathsf{a}} \colon Q \to Q$（**文字 a の遷移関数**）
- 関数 $T_{\mathsf{b}} \colon Q \to Q$（**文字 b の遷移関数**）
- 集合 $F \subseteq Q$（**受理状態**の集合）

$Q = \{1, 2, 3, 4\}$ $(n = 4)$

$F = \{2, 4\}$

遷移関数：

q	$T_{\mathsf{a}}(q)$	$T_{\mathsf{b}}(q)$
1	1	2
2	2	1
3	4	3
4	3	4



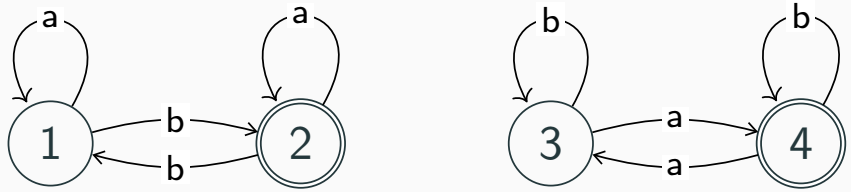
オートマトンで文字列を区別する

オートマトンで文字列を区別する

初期の状態を q として文字列 w と w' の出力が異なるとき（一方が受理を出力してもう一方が非受理を出力するとき）、
「**文字列 w と w' を状態 q で区別できる**」とよぶ。

オートマトンで文字列を区別する

初期の状態を q として文字列 w と w' の出力が異なるとき（一方が受理を出力してもう一方が非受理を出力するとき）、
「**文字列 w と w' を状態 q で区別できる**」とよぶ。

入力	初期	**オートマトン**	終了	出力
bb	1		1	非受理
bba	1		1	非受理
bbab	1		2	受理
bbabb	1		1	非受理

状態 1 で区別できる文字列のペア（✓ のペアを区別できる）：

	bb	bba	bbab	bbabb
bb			✓	
bba			✓	
bbab	✓	✓		✓
bbabb			✓	

オートマトンで文字列を区別する（複数の状態を使うとき）

状態の集合： $Q = \{1, 2, 3, 4\}$, 受理状態の集合： $F = \{2, 4\}$

遷移関数：($T_w(q)$ は状態 q から文字列 w で遷移した先の状態)

q	$T_{\mathsf{a}}(q)$	$T_{\mathsf{b}}(q)$	$T_{\mathsf{bb}}(q)$	$T_{\mathsf{bba}}(q)$	$T_{\mathsf{bbab}}(q)$	$T_{\mathsf{bbabb}}(q)$
1	1	2	1 (非受理)	1 (非受理)	2 (受理)	1 (非受理)
2	2	1				
3	4	3				
4	3	4				

オートマトンで文字列を区別する（複数の状態を使うとき）

状態の集合： $Q = \{1, 2, 3, 4\}$, 受理状態の集合： $F = \{2, 4\}$

遷移関数：($T_w(q)$ は状態 q から文字列 w で遷移した先の状態)

q	$T_{\mathsf{a}}(q)$	$T_{\mathsf{b}}(q)$	$T_{\mathsf{bb}}(q)$	$T_{\mathsf{bba}}(q)$	$T_{\mathsf{bbab}}(q)$	$T_{\mathsf{bbabb}}(q)$
1	1	2	1 (非受理)	1 (非受理)	2 (受理)	1 (非受理)
2	2	1	2 (受理)	2 (受理)	1 (非受理)	2 (受理)
3	4	3	3 (非受理)	4 (受理)	4 (受理)	4 (受理)
4	3	4	4 (受理)	3 (非受理)	3 (非受理)	3 (非受理)

オートマトンで文字列を区別する（複数の状態を使うとき）

状態の集合： $Q = \{1, 2, 3, 4\}$, 受理状態の集合： $F = \{2, 4\}$
遷移関数：($T_w(q)$ は状態 q から文字列 w で遷移した先の状態)

q	$T_{\mathsf{a}}(q)$	$T_{\mathsf{b}}(q)$	$T_{\mathsf{bb}}(q)$	$T_{\mathsf{bba}}(q)$	$T_{\mathsf{bbab}}(q)$	$T_{\mathsf{bbabb}}(q)$
1	1	2	1 (非受理)	1 (非受理)	2 (受理)	1 (非受理)
2	2	1	2 (受理)	2 (受理)	1 (非受理)	2 (受理)
3	4	3	3 (非受理)	4 (受理)	4 (受理)	4 (受理)
4	3	4	4 (受理)	3 (非受理)	3 (非受理)	3 (非受理)

	bb	bba	bbab	bbabb
bb		✓	✓	✓
bba	✓		✓	
bbab	✓	✓		✓
bbabb	✓		✓	

表 3: 集合 $\{1, 2, 3, 4\}$ に属するいずれかの状態で区別できる文字列のペア

	bb	bba	bbab	bbabb
bb		✓	✓	✓
bba	✓		✓	
bbab	✓	✓		✓
bbabb	✓		✓	

表 4: 集合 $\{1, 3\}$ に属するいずれかの状態で区別できる文字列のペア

	bb	bba	bbab	bbabb
bb			✓	
bba			✓	
bbab	✓	✓		✓
bbabb			✓	

表 5: 集合 $\{1\}$ に属するいずれかの状態で区別できる文字列のペア

	bb	bba	bbab	bbabb
bb		✓	✓	✓
bba	✓			
bbab	✓			
bbabb	✓			

表 6: 集合 $\{3\}$ に属するいずれかの状態で区別できる文字列のペア

☞ **状態を多く使えば区別できる文字列のペアも多くなるが、**
使う状態をできるだけ少なくしたい（上の表 3 と表 4 は全く同じ表）。

問題

作成するプログラム

- 入力としてオートマトン $\mathcal{A} = \langle Q, T_{\mathsf{a}}, T_{\mathsf{b}}, F \rangle$ と m 個の文字列 $w_1, \ldots, w_m$ が与えられる。ただし、Q はある n について $Q = \{1, 2, \ldots, n\}$ となるような集合である。
- これらの文字列に関して、状態の集合 Q で区別できる文字列のペアをすべて区別できるような集合 Q' がありうるだろう。
- Q' は Q の部分集合で、なるべく要素の個数が少ないものを探すプログラムを作成する。

入力の生成方法

入力生成器プログラム（tests/generator.cpp） に従って生成された 10 個の問題例を解く：

- オートマトンの状態数は $n = 1000$ で固定。
- 文字列の個数は $m = 2000$ で固定。
- 文字列の長さの総和は $\sum_{j=1}^{m} |w_j| = m \times 500$ で固定。
- 入力のオートマトンの遷移関数の値 $T_{\mathsf{a}}(1), \ldots, T_{\mathsf{a}}(n)$, $T_{\mathsf{b}}(1), \ldots, T_{\mathsf{b}}(n)$ と受理状態の集合 F は一様ランダムに与えられる。
- 審査では、SEED の値を「ある値」に変更して、問題例を生成する（つまり、初期の入力生成器プログラムが生成する問題例とは異なる問題例で審査する）。

入出力の方法

入力　ヘッダファイル（tests/sc_header.h）で定義される関数 sc::initialize を main 関数のはじめに呼び出すことで、ヘッダファイルで定義される変数に情報が与えられる。

出力　ヘッダファイル（tests/sc_header.h）で定義される関数 void sc::output(int k, int qs[]) を呼び出すことによりおこなう：

- k には、使用する状態の個数（$1 \leq k \leq n$）を与える。
- qs には、使用する状態の情報を持った長さ k の配列を与える。各 i（$0 \leq i \leq k-1$）について $1 \leq qs[i] \leq n$ を満たすようにする。

解の出力は何回でもおこなってよいが、（時間内に最後まで出力された解のうち）最後に出力された解のみを解答として使用する。

スコア

スコアの定義
$Q' = \{qs[0], \ldots, qs[k-1]\} \subseteq Q$ が、問題の条件を満たす（すなわち集合 Q' で、状態の集合 Q で区別できる文字列のペアをすべて区別できる）場合の解のスコアを

$$k \times n^2 + (\sum_{i=0}^{k-1} qs[i])$$

とする（つまり、k が小さいほどスコアが低くなり、k が同じ場合には、$(\sum_{i=0}^{k-1} qs[i])$ が小さいほどスコアが低くなる）。**このスコアをできるだけ低くする**。

出力の制約や問題の条件を満たさない場合のスコアは、$2n^3$（正しい出力した場合のどのスコアよりも高い、最悪のスコア）とする。

評価方法（勝利条件）

- 提出されたプログラムで、入力生成器プログラムに従って生成された 10 個の問題例を解く。各問題例についての実行時間の制限を 60 秒とする。
- 各問題例について、解のスコアが低い順に順位を付ける。スコアが同じだった場合には、解が出力されるまでのプログラムの実行時間が短い順に順位を付ける。1 位には 20 点、2 位には 19 点、. . .、20 位には 1 点を与える。
- 10 問の点数を合計して、総合順位を付ける。点数の合計が同じ場合には、スコアの合計が小さい順に総合順位を付ける。スコアの合計が同じ場合には、解が出力されるまでのプログラムの実行時間の合計が短い順に総合順位を付ける。

（・・・昨年は、「**交換モンテカルロ法**（と擬似焼きなまし法を合わせたような手法）」のチームが優勝したが、今年は？）

考えられる手法

本問は「**遷移関数のシミュレーション**」と「**使う状態の個数の最小化**」の２つのパートに分けられるが、重要なのは後者。

これは**組合せ最適化の問題**[¶1]だが、解空間がとても広い（単純には $n = 1000$ 個の各状態を「使う」か「使わない」か $2^{1000} \simeq 1.07 \times 10^{301}$ 通り）。最適解を求めることは現実的でなく、何らかの探索によって「出来るだけ最適解に近い解」を求めるようになるだろう。探索の手法としては、たとえば以下（そして、富岳をいかに活かすか！）：

1. 一様ランダム移動による局所探索
2. 山登り法（貪欲法）
3. ビームサーチ（beam search）
4. 模擬焼きなまし法
5. 交換モンテカルロ法

[¶1]キーワード：“Minimum Test Collection Problem”, “Set Cover Problem”

本選問題の原形となった問題

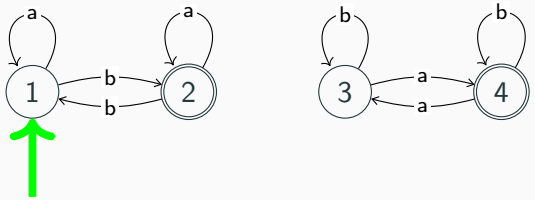
オートマトンに関する未解決問題（2022/08 現在）

（separating words problem）

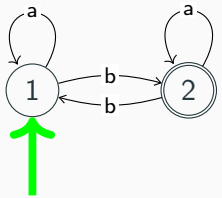
入力として２つの異なる文字列 w, w' が与えられる。この時、「w と w' をある状態で区別できる**（状態数が）最小のオートマトン**」の状態数は、入力長 $N = |w| + |w'|$ に対してどの位の大きさか？

例：bb と bbab を区別できるオートマトン

（状態数 4：冗長）



（状態数 2：**最小**）



（今回の本選問題は、上の問題を背景に、いくつかアレンジを加えたもの）

本選問題の原形となった問題

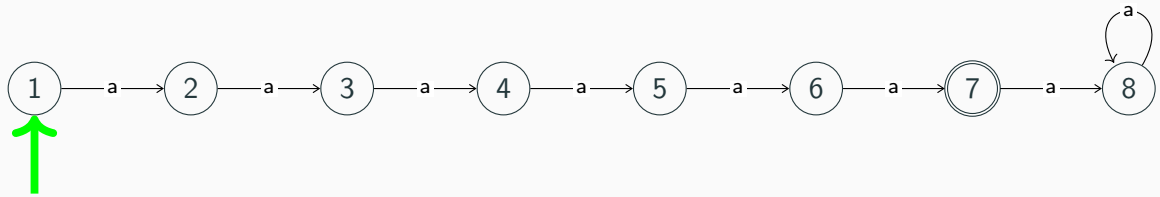
オートマトンに関する未解決問題（2022/08 現在）

（separating words problem）

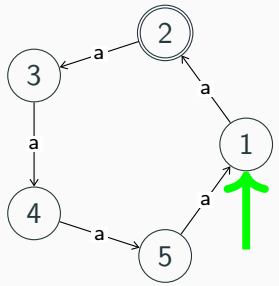
入力として２つの異なる文字列 w, w' が与えられる。この時、「w と w' をある状態で区別できる**（状態数が）最小のオートマトン**」の状態数は、入力長 $N = |w| + |w'|$ に対してどの位の大きさか？

例：aaaaaa と aaaaaaaaaaaa を区別できるオートマトン（長さ 6 と 12）

（状態数 8：冗長）



（状態数 5：（もう少しだけ小さくできます））



文字列の長さが異なる場合、（上の例のように円環状のもので）状態数 $\mathcal{O}(\log N)$ のオートマトンがあることが知られているが . . . 長さが同じ場合では？