SuperCon2022 本選問題：オートマトンで文字列を区別しよう

概 要

オートマトン理論は、60 年以上の歴史がある計算機科学の一分野であり、プログラミング言語やパターンマッチなどへの応用も数多く知られている。今年の本選問題は、オートマトン理論のある有名な未解決問題[1]（2022 年 8 月現在）にヒントを得て、作成したものである。

1 用語解説

オートマトン 本選問題では、a と b の二種類の文字からなるオートマトン[2]を考える。オートマトン $\mathcal{A}$ は、

- ある n 以下の正の整数の有限集合 $Q = \{1, 2, \ldots, n\}$（**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。これは、状態を**受理** (YES) の状態と**非受理** (NO) の２つに分けるために用いる。F に属する状態を**受理**状態、F に属さない状態を**非受理**状態と呼ぶ）

の４つ組 $\mathcal{A} = \langle Q, T_{\mathsf{a}}, T_{\mathsf{b}}, F \rangle$ で定義される。表 1 はオートマトンの例であり、右の図 1 はこれに対応する図である。正の整数でラベル付けされた各頂点は、各状態に対応する。二重丸で囲まれた頂点は、その状態が受理状態であることを表す。a および b でラベル付けされた矢印は、それぞれ関数 T_{a} と T_{b} を表す。

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

表 1: オートマトンの例

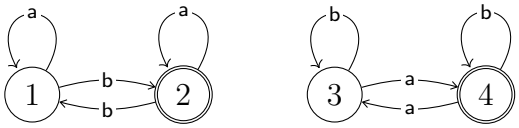

図 1: 表 1 のオートマトンに対応する図

[1]“separating words problem” と呼ばれる問題が背景だが、本選問題を解く上で、この問題について知られる事実は恐らくあまり役に立たないだろう（問題設定がいくつか異なるため）。

[2]本選問題では、決定性有限オートマトンと呼ばれる最も単純なオートマトンのひとつを扱う。オートマトンの定義には、初期の状態を含める場合も多いが、説明の都合上含めていない。

文字列による状態遷移　文字列 $w = c_1 \dots c_n$（ただし、n はある正の整数、各 $c_1, \dots, c_n$ は a または b の文字）について、**文字列 w の遷移関数** T_w を $T_{c_1 \dots c_n}(q) = T_{c_n}(\dots(T_{c_2}(T_{c_1}(q)))\dots)$ で定義する。図的には、$T_{c_1 \dots c_n}(q)$ は状態 q から c_1, c_2, $\dots$, c_n のラベルがついた矢印を順々に辿って到達する状態を指す。たとえば、図 1 のオートマトンにおいて、$T_{\mathsf{bbab}}(1) = T_{\mathsf{b}}(T_{\mathsf{a}}(T_{\mathsf{b}}(T_{\mathsf{b}}(1)))) = T_{\mathsf{b}}(T_{\mathsf{a}}(T_{\mathsf{b}}(2))) = T_{\mathsf{b}}(T_{\mathsf{a}}(1)) = T_{\mathsf{b}}(1) = 2$ である。これは、次のような 4 回の遷移を行うことに対応する（図 1 も参照すること）：$1 \xrightarrow{\mathsf{b}} 2 \xrightarrow{\mathsf{b}} 1 \xrightarrow{\mathsf{a}} 1 \xrightarrow{\mathsf{b}} 2$ 。より多くの例については、表 2 を参照する。

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

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 ($\notin F$)	1 ($\notin F$)	2 ($\in F$)	1 ($\notin F$)
2	2	1	2 ($\in F$)	2 ($\in F$)	1 ($\notin F$)	2 ($\in F$)
3	4	3	3 ($\notin F$)	4 ($\in F$)	4 ($\in F$)	4 ($\in F$)
4	3	4	4 ($\in F$)	3 ($\notin F$)	3 ($\notin F$)	3 ($\notin F$)

表 2: 図 1 のオートマトンにおける文字列の遷移関数の例

オートマトンによる文字列の区別　状態 q と 2 つの文字列 w と w' について、$T_w(q)$ と $T_{w'}(q)$ のいずれか一方が受理状態（$\in F$）でもう一方が非受理状態（$\notin F$）の時、「**文字列 w と w' を状態 q で区別できる**」と呼ぶ。たとえば、図 1 のオートマトンにおいて、文字列 bb と bba を区別できる状態は、3 と 4 の 2 つである（表 2 を参照。たとえば、状態 3 では、$T_{\mathsf{bb}}(3) = 3 \notin F$ と $T_{\mathsf{bba}}(3) = 4 \in F$ のため、bb と bba を区別できる。一方で、状態 1 では、$T_{\mathsf{bb}}(1) = 1 \notin F$ と $T_{\mathsf{bba}}(1) = 1 \notin F$ のため、bb と bba を区別できない）。

表 2 からも分かるように、4 つの文字列 bb、 bba、bbab、bbabb の任意の異なるペアは、bba と bbabb のペアを除いて、1 から 4 のいずれかの状態によって区別できる（表 3）。ここで、このような文字列の区別をする上で、全ての状態を使用する必要はなく、たとえば 1 と 3 の状態のみでも十分である（表 4、この表が表 3 が全く同じであることに着目する）。その一方で、状態 1 のみの場合（表 5）や状態 3 のみの場合（表 6）は不十分である。

本選問題では、入力としてオートマトン $\mathcal{A} = \langle Q, T_{\mathsf{a}}, T_{\mathsf{b}}, F \rangle$ といくつかの文字列が与えられる。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ことを考えよう。

	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

2 問題

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。この時、次の条件（★）を満たす集合 $Q' \subseteq Q$ のうち、できるだけ要素の個数が少ないものを求めよ（個数が同じ場合、Q' の要素の総和が小さいほどよい）。
（★）：Q に属するある状態で区別できるような任意の文字列のペア $\langle w_i, w_j \rangle$ $(1 \leq i, j \leq m)$ について、Q' に属するある状態で区別できる。

入力

入力は以下の形式で与えられる。ただし、入力の読み込みは、**後述のヘッダファイルで定義される関数 sc::initialize を main 関数のはじめに呼び出すことによりおこなうこと**：

n
$T_{\mathsf{a}}(1)\ T_{\mathsf{a}}(2)\ \ldots\ T_{\mathsf{a}}(n)$
$T_{\mathsf{b}}(1)\ T_{\mathsf{b}}(2)\ \ldots\ T_{\mathsf{b}}(n)$
$F_1\ F_2\ \ldots\ F_n$
m
w_1
w_2
$\vdots$
w_m

- 1 行目から 4 行目で、オートマトン $\mathcal{A} = \langle Q, T_{\mathsf{a}}, T_{\mathsf{b}}, F \rangle$ の情報が与えられる。
  - 1 行目には整数 n が与えられる。$Q = \{1, \ldots, n\}$ を表す。
  - 2 行目には整数 $T_{\mathsf{a}}(1), T_{\mathsf{a}}(2), \ldots, T_{\mathsf{a}}(n)$ が空白区切りで与えられる。
  - 3 行目には整数 $T_{\mathsf{b}}(1), T_{\mathsf{b}}(2), \ldots, T_{\mathsf{b}}(n)$ が空白区切りで与えられる。
  - 4 行目には 0 または 1 の整数 $F_1, F_2, \ldots, F_n$ が空白区切りで与えられる。各 i（$1 \leq i \leq n$）について、$F_i = 1$ は i が F の要素であること、$F_i = 0$ は i が F の要素でないこと、を表す。
- 5 行目から $5 + m$ 行目で、m 個の文字列の情報が与えられる。
  - 5 行目には整数 m が与えられる。
  - 続く m 行のうち j 行目には、a または b の文字からなる長さ 1 以上の文字列 w_j が与えられる。

上記の入力は、main 関数のはじめに関数 sc::initialize を呼び出すことで、ヘッダファイルで定義される各変数に値が代入される。入力とヘッダファイルとの間の変数の対応については次の通り。

入力	ヘッダファイル
n	`sc::n`
$T_{\mathsf{a}}(1)\ \ldots\ T_{\mathsf{a}}(n)$	`sc::T[0][0] ... sc::T[0][sc::n - 1]`
$T_{\mathsf{b}}(1)\ \ldots\ T_{\mathsf{b}}(n)$	`sc::T[1][0] ... sc::T[1][sc::n - 1]`
$F_1\ \ldots\ F_n$	`sc::F[0] ... sc::F[sc::n - 1]`
m	`sc::m`
w_1	`sc::ws[0]`
w_2	`sc::ws[1]`
$\vdots$	$\vdots$
w_m	`sc::ws[sc::m - 1]`

制約　入力は以下の制約を満たす：

- $1 \leq n \leq 1{,}000$。
- 各 $1 \leq i \leq n$ について、$1 \leq T_{\mathsf{a}}(i) \leq n$。
- 各 $1 \leq i \leq n$ について、$1 \leq T_{\mathsf{b}}(i) \leq n$。
- 各 $1 \leq i \leq n$ について、$0 \leq F_i \leq 1$。
- $1 \leq m \leq 2{,}000$。
- 各 $1 \leq j \leq m$ について、w_j は a または b の文字からなる文字列で $1 \leq |w_j|$ を満たす。また、$\sum_{j=1}^{m} |w_j| \leq m \times 500$ を満たす。ただし、$|w_j|$ は文字列 w_j の長さを表す。

ただし、最終的な評価では、入力生成器（A を参照）によってランダムに生成された入力例が用いられる。これらの入力例では、とくに以下を満たす。

- $n = 1{,}000$。
- $m = 2{,}000$。
- $\sum_{j=1}^{m} |w_j| = m \times 500$。

出力

解の出力は、**後述のヘッダファイルで定義される関数 sc::output(int k, int $qs[]$) を呼び出すことによりおこなうこと**：

- 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（$qs[0]$ から $qs[k-1]$ の k 個の状態を使用することを表す）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こと。

スコア　$Q' = \{qs[0], \ldots, qs[k-1]\}$ として、問題の条件（★）を満たす場合、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）。**スコアは低いほどよい**。なお、出力の制約を満たさない場合や条件（★）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。解の出力は何回でもおこなっ

てよい（関数 sc::output を複数回使用してよい）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。ただし、制約を満たさない出力が含まれていた場合、出力が正しく読み込まれず、スコアは $2n^3$ として扱われる場合がある（制約を満たさないような sc::output の呼び出しをしないよう注意すること）。

入出力の例

例１　以下の入力を考える。

```
4
1 2 4 3
2 1 3 4
0 1 0 1
4
bb
bba
bbab
bbabb
```

この入力は問題文中の問題例（図 1）に対応する。
この時、以下は $Q' = \{1, 2, 3\}$ の解を出力するプログラムの例である（解の出力に関する部分を切り取ったもの）。この時、スコア $3 \times 4^2 + (1 + 2 + 3) = 54$ である。

```
int qs[3] = {1, 2, 3};
sc::output(3, qs);
```

なお、以下は $Q' = \{1, 3\}$ の解を出力するプログラムの例であり、こちらの解の方がよりよい解である（スコア $2 \times 4^2 + (1 + 3) = 36$）。

```
int qs[2] = {1, 3};
sc::output(2, qs);
```

解の出力は複数回できる（関数 sc::output は複数回呼び出してよい）。以下は 4 回の解の出力をおこなうプログラムの例である。

```
{
  int qs[1] = {4};
  sc::output(1, qs);
}
{
  int qs[4] = {1, 2, 3, 4};
  sc::output(4, qs);
}
{
  int qs[2] = {1, 3};
  sc::output(2, qs);
}
{
  int qs[3] = {1, 2, 3};
  sc::output(3, qs);
}
```

1 番目の解は、条件（★）を満たさず、スコアは $2 \times 4^3 = 128$ である。以降の解は、条件（★）を満たし、スコアは順に $4 \times 4^2 + (1+2+3+4) = 74$、$2 \times 4^2 + (1+3) = 36$、$3 \times 4^2 + (1+2+3) = 54$ である。この時最後の解のみが解答として使用される（つまり、スコア 54 の解として扱われる）。

例 2　以下の入力を考える。

```
10
2 6 8 6 4 10 2 4 10 9
5 4 9 2 9 10 6 6 4 1
0 1 0 1 0 0 1 1 0 0
10
baaaabbb
ababaa
bbaaababaabbabab
bbbabaabb
ababbbbaaa
baaaba
bbabbbaabb
bbabbbbbabaabaaabb
babbbabab
bbbbbbab
```

この時、たとえば以下のプログラムは、スコア $3 \times 10^2 + (1+3+6) = 310$ の解を出力する。

```
int qs[3] = {1, 3, 6};
sc::output(3, qs);
```

3　提出物について

ソースコード（main.cpp）とジョブスクリプト（run_cases.sh）を提出する。

3.1　ソースコード（main.cpp）について

ソースコードは以下のルールに沿うこと（配布するサンプルコードも参考にすること）：

- ヘッダファイル tests/sc_header.h（参照：B）をインクルードすること。MPI を使用する場合、mpi.h を tests/sc_header.h よりも前にインクルードすること（MPI を使用しない場合、mpi.h はインクルードしないこと）。
- main 関数のはじめに sc::initialize(argc, argv) を呼び出し、main 関数のおわりに sc::finalize() を呼び出すこと。
- 解の出力にはヘッダファイルで定義される関数 sc::output を使用すること。またヘッダファイル内の SC22_KEY の値の取得を試みないこと（なお、審査では SC22_KEY の値は「ある値」に変更される）。

コンパイルのコマンド

MPI を使用する場合：

```
mpiFCC -Ofast -fopenmp -Nclang -std=c++17 main.cpp
```

MPI を使用しない場合：

```
FCC -Ofast -fopenmp -Nclang -std=c++17 main.cpp
```

3.2 ジョブスクリプト（run_cases.sh）について

今回のコンテストでは MPI のプロセス数や OpenMP のスレッド数を実行時に指定してもらうため、ジョブスクリプトも提出する。**変更してよいのは、MPI のプロセス数（--mpi proc）の値と OpenMP のスレッドの数（OMP_NUM_THREADS）の値のみで、それ以外の変更を加えてはならない。**tests/jobscript.sh では、main.cpp をコンパイルして、入力生成器が生成する 10 個の入力例を実行する。

MPI を使用する場合のサンプル：

```
#!/bin/bash
#PJM --rsc-list "elapse=0:20:00"
#PJM --out "log/jobscript.sh.%j.out"
#PJM -j
#PJM -L node=1
#PJM --mpi proc=4
export IS_MPI_PROGRAM=1
export OMP_NUM_THREADS=12
export PLE_MPI_STD_EMPTYFILE=off
./tests/jobscript.sh

# MPI を使用する（
    MPI と OpenMP を使用する場合も含む）の場合のジョブスクリプトの例。
# #PJM -L node=1
# #PJM --mpi proc=X
# export IS_MPI_PROGRAM=1
# export OMP_NUM_THREADS=Y
# の X, Y をある整数値に指定する（上は X = 4, Y = 12 の場合）。それ以外の変更は加
    えないこと。
# ・node の値は、ノード数を表す。
# ・proc の値 (X) は、MPI のプロセス数。
# ・IS_MPI_PROGRAM の値は、MPI を使用する場合 1、使用しない場合 0 を指定する
    。
# ・OMP_NUM_THREADS の値 (Y) は、OpenMP のスレッド数を表す。
```

MPI を使用しない場合のサンプル：

```
#!/bin/bash
#PJM --rsc-list "elapse=0:20:00"
#PJM --out "log/jobscript.sh.%j.out"
#PJM -j
#PJM -L node=1

export IS_MPI_PROGRAM=0
export OMP_NUM_THREADS=48
```

```
export PLE_MPI_STD_EMPTYFILE=off
./tests/jobscript.sh

# MPI を使用しない場合のジョブスクリプトの例。
# #PJM −L node=1
#
# export IS_MPI_PROGRAM=0
# export OMP_NUM_THREADS=Y
# の Y をある整数値に指定する(上は Y = 48 の場合)。それ以外の変更は加えないこと
    。
# ・node の値は、ノード数を表す。
#
# ・IS_MPI_PROGRAM の値は、MPI を使用する場合 1、使用しない場合 0 を指定する
    。
# ・OMP_NUM_THREADS の値 (Y) は、OpenMP のスレッド数を表す。
```

4 評価方法

- 提出されたプログラムで、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（参照：A)。各問題例についての実行時間の制限を 60 秒とする。プログラムは制限時間以内に停止しなくてもよい。つまり、時間切れ強制終了となってもよい。

- 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（解が出力されなかった場合には実行時間の制限と同じ 60 秒とする）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。

- 各問題例における点数を合計して、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。

A　入力生成器プログラム

以下の入力生成器プログラムから 4 行目の疑似乱数のシード値（SEED）を「ある値」に変更したプログラムによって生成された 10 個の問題例（tests/random_01.in, ..., tests/random_10.in）を用いて審査する。

入力生成器プログラム（tests/generator.cpp）：

```
#include <cstdio>
#include <cstdlib>
// 審査ではSEED の値を別の「ある値」に変更する。
constexpr unsigned long long SEED = 1ull;

constexpr int CASE_SIZE = 10;
constexpr int A_SIZE    = 2;
constexpr int N         = 1000;
constexpr int M         = 2000;
constexpr int L         = (M * 500);

unsigned long long xor_shift() {
  static unsigned long long x = SEED;
  x                           = x ^ (x << 13);
  x                           = x ^ (x >> 7);
  x                           = x ^ (x << 17);
  return x;
}

// [0, x)
unsigned long long rnd(unsigned long long x) { return xor_shift() % x; }

// [l, r]
unsigned long long range_rnd(unsigned long long l, unsigned long long r)
    { return l + rnd(r - l + 1); }

int main() {
  for (int t = 1; t <= CASE_SIZE; t++) {
    char file_name[64];
    sprintf(file_name, "random_%02d.in", t);

    int **T = (int **)malloc(A_SIZE * sizeof(int *));
    for (int c = 0; c < A_SIZE; c++) {
      T[c] = (int *)malloc(N * sizeof(int));
      for (int i = 0; i < N; i++) { T[c][i] = range_rnd(1, N); }
    }
    int *F = (int *)malloc(N * sizeof(int));
    for (int i = 0; i < N; i++) { F[i] = range_rnd(0, 1); }

    int *ls = (int *)calloc(M, sizeof(int));
    for (int i = 0; i < L; i++) {
      int pos = i < M ? i : range_rnd(0, M - 1);
      ls[pos]++;
    }
    char **ws = (char **)malloc(M * sizeof(char *));
    for (int i = 0; i < M; i++) {
      ws[i] = (char *)malloc((ls[i] + 1) * sizeof(char));
      for (int j = 0; j < ls[i]; j++) { ws[i][j] = 'a' + range_rnd(0,
   A_SIZE - 1); }
      ws[i][ls[i]] = '\0';
```

```
    }

    {
      FILE *fp = fopen(file_name, "w");
      if (fp == NULL) return 1;
      {
        fprintf(fp, "%d\n", N);
        for (int c = 0; c < A_SIZE; c++) {
          for (int i = 0; i < N; i++) {
            if (i > 0) fprintf(fp, " ");
            fprintf(fp, "%d", T[c][i]);
          }
          fprintf(fp, "\n");
        }
        for (int i = 0; i < N; i++) {
          if (i > 0) fprintf(fp, " ");
          fprintf(fp, "%d", F[i]);
        }
        fprintf(fp, "\n");
      }
      {
        fprintf(fp, "%d\n", M);
        for (int i = 0; i < M; i++) { fprintf(fp, "%s\n", ws[i]); }
      }
      fclose(fp);
    }

    free(ls);
    for (int i = 0; i < M; i++) free(ws[i]);
    free(ws);
    free(F);
    for (int c = 0; c < A_SIZE; c++) free(T[c]);
    free(T);
  }
  return 0;
}
```

B　ヘッダファイル

以下のヘッダファイルを必ずインクルードすること。
ヘッダファイル（tests/sc_header.h）：

```
#include <omp.h>

#include <cassert>
#include <cstdio>
#include <cstdlib>
#include <cstring>

namespace sc {
constexpr int CASE_SIZE     = 10;
constexpr double TIME_LIMIT = 60.0;
constexpr int A_SIZE        = 2;

constexpr int N_MIN = 1;
constexpr int N_MAX = 1000;
```

```
constexpr int M_MIN = 1;
constexpr int M_MAX = 2000;
constexpr int L_MAX = (M_MAX * 500);

int n;
int T[A_SIZE][N_MAX];
int F[N_MAX];
int m;
char *w[M_MAX];
int w_len[M_MAX];

// インターフェース

// 出力に用いる関数。出力の際は、この関数を必ず使用しなくてはならない。
// int k: 使用する状態の個数を表す。
// int qs[] : 使用する状態の情報を持った長さ k の配列。
void output(int k, int qs[]);

// 現在の実行時間(単位秒)を返す関数。
double get_elapsed_time();

// 実装
#define SC22_KEY 334  // 審査ではSC22_KEY の値を別の「ある値」に変更する。
template <int key> double &time0() {
  static double t;
  return t;
}
double get_elapsed_time() { return omp_get_wtime() - time0<SC22_KEY>();
    }

// 入力を読み込む関数。
void input() {
  int rank = 0;
#ifdef MPI_VERSION
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
#endif
  if (rank == 0) {
    if (scanf("%d", &n) != 1) assert(false);
    for (int c = 0; c < A_SIZE; c++) {
      for (int i = 0; i < n; i++) {
        if (scanf("%d", &T[c][i]) != 1) assert(false);
      }
    }
    for (int i = 0; i < n; i++) {
      if (scanf("%d", &F[i]) != 1) assert(false);
    }
    if (scanf("%d", &m) != 1) assert(false);
    char s[L_MAX];
    for (int i = 0; i < m; i++) {
      if (scanf("%s", s) != 1) assert(false);
      w_len[i] = strlen(s);
      w[i]     = (char *)malloc((strlen(s) + 1) * sizeof(char));
      strcpy(w[i], s);
    }
  }
#ifdef MPI_VERSION
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
  MPI_Bcast(T, A_SIZE * N_MAX, MPI_INT, 0, MPI_COMM_WORLD);
```

```
  MPI_Bcast(F, N_MAX, MPI_INT, 0, MPI_COMM_WORLD);
  MPI_Bcast(&m, 1, MPI_INT, 0, MPI_COMM_WORLD);
  MPI_Bcast(w_len, M_MAX, MPI_INT, 0, MPI_COMM_WORLD);
  for (int i = 0; i < m; i++) {
    if (rank != 0) MPI_Alloc_mem((w_len[i] + 1) * sizeof(char),
   MPI_INFO_NULL, &w[i]);
    MPI_Bcast(w[i], w_len[i] + 1, MPI_CHAR, 0, MPI_COMM_WORLD);
  }
#endif
}

// 出力に用いる関数。出力の際は、この関数を必ず使用しなくてはならない。
// int k: 使用する状態の個数を表す。
// int qs[] : 使用する状態の情報を持った長さ k の配列。
void output(int k, int qs[]) {
  int rank = 0;
#ifdef MPI_VERSION
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
#endif
  if (rank) return;
  if (omp_get_thread_num()) return;

  printf("%d : begin_output : %f\n", SC22_KEY, get_elapsed_time());
  printf("%d : %d\n", SC22_KEY, k);
  printf("%d : ", SC22_KEY);
  for (int j = 0; j < k; j++) printf("%d ", qs[j]);
  printf("\n");
  printf("%d : end_output : %f\n", SC22_KEY, get_elapsed_time());
  fflush(stdout);
}

void initialize(int &argc, char **&argv) {
#ifdef MPI_VERSION
  MPI_Init(&argc, &argv);
#endif
  input();
  time0<SC22_KEY>() = omp_get_wtime();
}

void finalize() {
  for (int i = 0; i < m; i++) { free(w[i]); }
#ifdef MPI_VERSION
  MPI_Finalize();
#endif
}

#undef SC22_KEY

}  // namespace sc
```