Robot Programming Strategy
ロボットのプログラミング戦略

Problem 問題

After many sleepless nights, you have finally finished teaching a robotic arm to make the hand gestures required for the Rock-Paper-Scissors game.

Now you just need to program it to compete in the upcoming robot tournament!

In this tournament, each robot uses a program that is a series of moves, each of which must be one of the following: R (for “Rock”), P (for “Paper”), or S(for “Scissors”).

Paper beats Rock and loses to Scissors; Rock beats Scissors and loses to Paper; Scissors beats Paper and loses to Rock.

When two robots face off in a match, the first robot to play a winning move wins.

To start, each robot plays the first move of its program.

If the two moves are different, one of the moves beats the other and thus one of the robots wins the match.

If the moves are the same, each robot plays the next move in its program, and so on.

Whenever a robot has reached the end of its program and needs its next move, it returns to the start of its program.

So, for example, the fifth move of a robot with

眠らない多くの夜ののち、あなたはついにロボットの腕に、ジャンケンをするのに必要な手のジェスチャーのしかたを教えることができた。

予定されているロボットトーナメントで競うために、あと必要なのは、プログラムすることだ。

このトーナメントでは、各ロボットは、動きの列であるプログラムを用いる。動きの列の各要素は、（石を表す）R、（紙を表す）P、（ハサミを表す）S のいずれかだ。

紙は石には勝ち、ハサミには負ける。石はハサミには勝ち、紙には負ける。ハサミは紙には勝ち、石には負ける。

あるマッチにおいて 2 つのロボットが向かい合うと、最初に勝つ動きをしたロボットが勝つ。

開始時、それぞれのロボットが各々のプログラムの最初の動きを行う。

もし 2 つの動きが異なるなら、一方の動きがもう他方に勝ち、よって、片方のロボットがマッチに勝利する。

もし動きが同じなら、両方のロボットが自身のプログラムの次の動きをする。以降も同じだ。

あるロボットが自身のプログラムの終わりに達し、次の動きをする必要があるときは、そのプログラムの最初に戻る。

よって例えば、プログラムが RSSP であるロボッ

the program RSSP would be R.

If a match goes on for over a googol (10^{100}) of moves, the judges flip a fair coin to determine the winner.

Once a match is over, the winning robot resets, so it has no memory of the that match.

In its next match, it starts by playing the first move of its program, and so on.

The tournament is played in K rounds and has a single-elimination "bracket" structure.

There are N = 2^K robots in total, numbered 0 through N - 1.

In the first round, robot 0 plays a match against robot 1, robot 2, plays a match against robot 3, and so on, up to robots N - 2 and N - 1.

The losers of those matches are eliminated from the tournament.

In the second round, the winner of the 0-1 match faces off against the winner of the 2-3 match, and so on.

Once we get to the K-th round, there is only one match, and it determines the overall winner of the tournament.

All of the other contestants are so confident that they have already publicly posted their robots' programs online.

However, the robots have not yet been assigned numbers, so nobody knows in advance who their opponents will be.

Knowing all of the other programs, is it possible for you to write a program that is *guaranteed* to win the tournament, no matter how the robot numbers are assigned?

トの 5 番目の動きは R である。

もしもマッチの動きの回数が googol（10^{100}）を越えたなら、審判は公正なコインを投げ、勝者を決める。

マッチが終わると、勝利したロボットはリセットされる。よってそのマッチの記憶は残らない。

その次のマッチでは、自身のプログラムの最初の動きから開始する。以降も同じだ。

トーナメントは K 個のラウンドで行われる。勝ち残り式で行い、負けた者は排除される。

N = 2^K 個のロボットが存在する。それぞれ 0 から N-1 までの番号が振られている。

最初のラウンドで、ロボット 0 はロボット 1 とマッチし、ロボット 2 はロボット 3 とマッチし、以降も同じであって、ロボット N-2 と N-1 まで続く。

それらのマッチにおいて負けた者は、トーナメントから排除される。

第 2 ラウンドでは、その 0-1 マッチでの勝者が、2-3 マッチでの勝者と対戦することになり、以降も同じである。

K 個目のラウンドに到達すると、そこには 1 つのマッチしかない。そこでトーナメント全体の勝者が決定される。

他のすべての参加者は自信があるので、自らのロボットのプログラムをオンラインに公開してすでに投稿してある。

しかしながら、ロボットらはまだ番号が振られてはいないので、自分の対戦相手が誰になるかは誰も前もって知ることができない。

他のすべてのプログラムを知っている状況で、あなたが、ロボットらにどのように番号が振られるにせよ、トーナメントに勝利することが保証されているプログラムを書くことは可能だろうか。

Input 入力

The first line of the input gives the number of test cases **T**; **T** test cases follow.

Each test case begins with one line containing an integer **A**: the number of adversaries (other robots) in the tournament.

Then, there are **A** more lines; the i-th of these contains a string $\mathbf{C_i}$ of uppercase letters that represent the program of the i-th opponent's robot.

入力の最初の行は、テストケースの個数 **T** をもたらす。**T** 個のテストケースらがつづく。

それぞれのテストケースにはまず、整数 **A** を含む行がある。これはトーナメントの対戦相手（他のロボット）の数だ。

その後さらに、**A** 個の行がある。それらの i 番目は、大文字らからなる文字列 $\mathbf{C_i}$ である。これは、i 番目の対戦相手のロボットのプログラムだ。

Output 出力

For each test case, output one line containing `Case #x: y`.

If there is a string of between 1 and 500 characters that is guaranteed to win the tournament, as described above, then y should be the string of uppercase letters representing that program.

Otherwise, y should be `IMPOSSIBLE`, in uppercase letters.

それぞれのテストケースについて、`Case #x: y` を含む 1 行を出力せよ。

1 文字から 500 文字までの長さの文字列で、上記のように、トーナメントに勝利すると保証されているものが存在するなら、y はそのプログラムを表す大文字の文字らであるべきだ。

そうでなければ、y は大文字の文字らで、`IMPOSSIBLE` であるべきである。

Limits 制約

$1 \leq \mathbf{T} \leq 100$.

Time limit: 20 seconds per test set.

Memory limit: 1 GB.

Each character in $\mathbf{C_i}$ is uppercase R, P, or S, for all i.

$\mathbf{A} = 2^K - 1$ for some integer $K \geq 1$.

$1 \leq \mathbf{T} \leq 100$。

時間制限: テストセット 1 つにつき 20 秒。

メモリ制限: 1 GB。

$\mathbf{C_i}$ の各文字は、すべての i について大文字の R ないし P ないし S である。

ある整数 $K \geq 1$ が存在して $\mathbf{A} = 2^K - 1$。

Test set 1 (Visible) テストセット 1（可視）

$1 \leq \mathbf{A} \leq 7$.

$\mathbf{C_i}$ is between 1 and 5 characters long, for all i.

$1 \leq \mathbf{A} \leq 7$.

すべての i について、$\mathbf{C_i}$ の長さは 1 以上 5 以下だ。

Test set 2 (Hidden) テストセット 2（不可視）

$1 \leq \mathbf{A} \leq 255$.

$\mathbf{C_i}$ is between 1 and 500 characters long, for all i.

$1 \leq \mathbf{A} \leq 255$.

すべての i について、$\mathbf{C_i}$ の長さは 1 以上 500 以下だ。

Sample サンプル

Input

```
3
1
RS
3
R
P
S
7
RS
RS
RS
RS
RS
RS
RS
```

Output

```
Case #1: RSRSRSP
Case #2: IMPOSSIBLE
Case #3: P
```

Note: Although all the opponents in each of these sample cases have programs of the same length, this is not necessarily the case.

Opponents within a test case might have programs of different lengths.

次の注意がある。これらのサンプルケースらの全ての対戦相手は同じ長さのプログラムを持っているが、必ずしもそうとは限らない。

あるテストケースの対戦相手らが異なる長さのプログラムらを持っている可能性はある。

In Sample Case #1, there is only one opponent, with the program RS.

Our answer matches the opponent's moves for a while, and the opponent loops through its program several times.

As is starts its fourth pass through its program, we beat it with P.

Other valid solutions exist, like P, RR, and R.

サンプルケース#1 では、対戦相手は 1 つだけで、そのプログラムは RS だ。

私達の答えは、しばらくの間、対戦相手の動きに一致している。対戦相手は自身のプログラムを何度もループする。

それが自身のプログラムの 4 回目の走査を始めたとき、私達は P によってそれに勝つ。

P、RR、R など、他の有効な解らが存在する。

In Sample case #2 there are three opponents, with the programs R, P, and S.

It is up to you to figure out why this case is IMPOSSIBLE!

サンプルケース#2 では、3 個の対戦相手がいる。それぞれのプログラムは R、P、S だ。

このケースが IMPOSSIBLE である理由を考え出すのはあなたの役割だ！

In Sample Case #3, all seven opponents use the same program.

サンプルケース#3 では、7 個の対戦相手すべてが同じプログラムを用いる。

Using the program P, for example, guarantees that you will win.

Remember that each robot begins at the start of its program at the start of each match against a new opponent.

例えば、プログラム P を用いることで、あなたの勝利は保証される。

すでに述べたように、新しい相手とのマッチが始まるたびにプログラムの最初から実行されることに注意せよ。

Analysis 解析

With **A** competitors, there are **A**! possible initial setups for the tournament bracket, and **A** is itself exponential in the number of rounds of the tournament.

A factorial of an exponential is terrifying enough, and we haven't even started the tournament yet!

A 個の対戦相手があるとき、勝ち抜きトーナメントの初期状態の組み方は **A**! 個ある。そして、**A** もまたそれ自身、トーナメントのラウンド数を指数とする指数的な数だ。

指数的な数の階乗とは、それ自体が十分に恐ろしいが、なお私達はトーナメントを開始してすらいない！

Fortunately, we can ignore almost everything about the structure of the tournament.

For any opponent, there is at least one possible initial setup in which we will play our first match against that opponent.

Since we have to guarantee that we will win the tournament regardless of the initial setup, we must be able to defeat *every* opponent.

We cannot tie an opponent, since that coin flip might not come up in our favor!

幸いにも、トーナメントの構造については、私達はほぼすべてを無視できる。

どの対戦相手についても、最初のマッチで私達がその相手と対戦できるようなトーナメントの組み方が 1 つ以上ある。

私達は、トーナメントの初期配置にかかわらず私達が勝つことを保証せねばならないのだから、私達は全ての対戦相手を倒せねばならない。

コイン投げは私達を味方しないかもしれないのだから、ある対戦相手と引き分けるわけにはいかない。

So all we have to do is find a program that beats every opponent's program.

To check a program, we can just check it against each opponent, without worrying about the tournament setup.

よって、すべての対戦相手のプログラムに勝つプログラムを見つけることが、私達がなすべきすべてだ。

プログラムをチェックするには、トーナメントの初期配置を気にすることなく、私達は単に全ての相手に対してチェックすることができる。

Test set 1 テストセット 1

In test set 1, there are at most 7 opponents, and their programs can be at most 5 characters long.

Our program can be longer than that if need be, but how long does it need to be?

We can observe that an optimal winning pro-

テストセット 1 では、最大で 7 個の対戦相手がいる。そのプログラムは、最大で 5 文字以下の長さだ。

私達のプログラムは、必要ならそれより長くできるが、どれほど長くする必要があるだろう？

最適な勝利するプログラムは、他の残りのすべて

gram should never waste a turn by tying with all remaining opponents, since then it could have instead chosen the move that would have beaten all of them; therefore, it should eliminate at least one opponent per move.

So in test set 1, if there is a winning program, it is at most 7 moves long.

We can comfortably check all $3^7 + 3^6 ... + 3$ possible programs of length up to 7.

の対戦相手と引き分けることでターンを無駄に消費すべきではないことに気づける。なぜなら、そのとき、それらの相手すべてを倒す動きを選ぶこともできたからだ。よって、1 つ動くごとに最低 1 つ以上の相手を排除すべきだ。

よってテストセット 1 では、勝つプログラムがあったなら、最大でも 7 個の動きによるものだ。

私達は、7 文字までのありうる $3^7 + 3^6 ... + 3$ 個すべてのプログラムを余裕を持ってチェックできる。

訳注。勝利するプログラムが存在すればそれは最大で 7 文字の長さである、というのはなぜなら、1 つ動くごとに 1 つ以上倒せて、相手の数は最大で 7 個だから、1 つ動くごとに最低の 1 個を倒して、7 個の動きで 7 個とも倒せる、ということである。$3^7 + 3^6 ... + 3$ 個のプログラムとは、7 文字のプログラムの個数は、それぞれの文字が 3 通りであるから 3^7 通りであり、同様に 6 文字のプログラムらや、1 文字のプログラムまでもを考えたということである。なお、$3^7 + 3^6 + 3^5 + 3^4 + 3^3 + 3^2 + 3^1 = 2187 + 729 + 243 + 81 + 27 + 9 + 3 = 3279$ である。3279 個は全探索できる。

When simulating a match, we cannot wait around for a googol of moves; by the same argument above, an optimal winning program should take no more than 7 moves to defeat all opponents, so we only need to simulate at most 7 moves.

If we are still tied with the opponent at that point, we can safely give up on that program.

あるマッチをシミュレートする上では、google 個の動きを待つことはできない。だが上に述べたように、最適な勝利するプログラムはすべての相手を倒すために 7 個以上動くべきではないのだから、私達は最大でも 7 個の動きをシミュレートする必要しかない。

そのときになお相手と引き分けていたならば、そのプログラムを負けだと判定することは正しい。

If we find that no program of length at most 7 beats every opponent's program, then we can declare that the case is `IMPOSSIBLE`.

長さ 7 以下のどのプログラムもすべての対戦相手を倒せないと分かったならば、その場合は `IMPOSSIBLE` なのだと私達は言い切れる。

訳注。引き分けが続く場合の例は次のようなものだろう。これは実際、7 文字を要している。

```
PPPPPPR ←プレイヤー
R
PR
PPR
PPPR
PPPPR
PPPPPR
PPPPPPS
```

Test set 2 テストセット 2

In test set 2, there can be up to 255 opponents, and we cannot generate and check all programs of length up to 255.

We must find a way to construct a winning program if it exists.

テストセット 2 では、最大で 255 個の対戦相手がある。そして私達は、255 文字までの長さのすべてのプログラムを生成してチェックすることはできない。

勝つプログラムが存在するときにはそれを作成する方法を私達は見つけ出さねばならない。

訳注。なお、$3^{255} = 463361507923815775883132622632204343714062836028430459972016081433453575432554786470005897180365365072705551801829664785 07$ である。この桁数は 122 桁だ。対数で桁数を求めるなら $\log_{10} 3^{255} = 255 \log_{10} 3 \approx 255 \times 0.47712125471966244 = 121.66591995351392$ と書ける。10^{122} の広さを持つ解空間は全探索できない。なお、$3^{255} + 3^{254} + ... + 3^2 + 3^1 = 69504226188572366382469893394830651557109425404264568995802412215018036314883217970500884577054804760905832770274449717759$ であり、桁数は 122 桁と変わらない。この値は Python で `sum(3**i for i in range(1, 256))` などとして求められる。

Let's imagine playing against all opponents at once.

How do we choose our program's first move?

We must win or tie against every opponent, so we will consider the set of their first moves.

If it includes all three possible moves, we are doomed; no matter what we pick, at least one opponent will defeat us.

If it includes only one move (e.g. every opponent starts with R), then we can pick the move that defeats that move (in this case, P) and instantly defeat everyone.

Otherwise, the set includes two of the possible moves, and we should pick the move that ties one and beats the other.

For example, if the opponents all lead off with S or P, we should pick S.

すべての対戦相手らにいっぺんに対峙するように考えてみよう。

プログラムの最初の動きをどう選ぼうか？

すべての相手に勝つか引き分けねばならないから、相手の最初の動きらについて考えよう。

そこにもし可能な 3 つすべての動きが含まれていたら絶望だ。なぜならどれを選ぼうとも、1 つ以上の対戦相手が私達を打ち負かす。

（例えばすべての相手が R などと）もし 1 つの動きしか含まれてなかったならば、それに勝つ動き（この場合は P）を選べば、ただちにすべての相手を倒せる。

そうでなければ、そのセットには 2 つの動きが含まれる。私達は、その一方と引き分け、もう一方を破る動きを選ぶべきだ。

例えば、相手らがみな S か P で始めたならば、私達は S を選ぶべきだ。

Eliminating any defeated opponents and proceeding to the next move of this combined "match", we can apply the same strategy, but considering the set of remaining opponents' second moves (looping through their programs as

敗北した相手らを排除し、この組み合わされた「マッチ」の次の動きへと進みながら、私達は同じ戦略を適用できる。しかし、残存している相手らの第 2 の動きらを（必要に応じて彼らのプログラムをループし）考える必要があり、3 個目の動き以降もそうで

needed), and so on.

We will eliminate at least one opponent with each move, so after **A** moves, we will either have our winning program or know that the case is IMPOSSIBLE.

Notice that this limit holds regardless of the lengths of the opponents' programs.

ある。

私達はそれぞれの動きごとに 1 つ以上の敵を排除するから、**A** 個の動きのあとには、私達は勝利するプログラムを手にしているか、さもなくばその場合が IMPOSSIBLE だと分かる。

なお、この制約は、対戦相手らのプログラムらの長さに関係なく言える。

implementation

```
# （期待通り、）test set 1 で AC、test set 2 で WA。
def permutation(l, cs):  # length, characters.
    """文字列 cs の要素からなる l 文字の順列らのリストを返す。"""
    if l == 1:
        return list(cs)
    sol = []             # solution
    for xs in permutation(l-1, cs):
        for c in cs:
            sol.append(xs + c)
    return sol

def simulate(program, C):
    """プログラム program がプログラムら C のすべてに勝つなら program を返し、
    そうでなければ False を返す。"""
    for i in range(9):  # i 文字目を考える
        # 最適なプログラムは 1 文字ごとに 1 個以上のプログラムを撃破できる。
        # よって test set 1 の敵の最大数 7 で勝てていないなら絶望とする。
        if i == 8:
            return False
        i1 = i % len(program)  # 周回した添字を剰余で得る
        move1 = program[i1]    # プレイヤーの i 個目の動き
        C2 = []                # 撃破したもの以外を集める
        for c in C:
            i2 = i % len(c)
            move2 = c[i2]      # 相手の i 番目の動き
            if move1 == "R":
                if move2 == "R":
                    C2.append(c)  # あいこ
                elif move2 == "P":
                    return False  # 負け
```

```
                else:  # move2 == "S":
                    pass          # 勝ち。このプログラムを排除する。
            elif move1 == "P":
                if move2 == "R":
                    pass
                elif move2 == "P":
                    C2.append(c)
                else:  # move2 == "S":
                    return False
            else:  # move1 == "S":
                if move2 == "R":
                    return False
                elif move2 == "P":
                    pass
                else:  # move2 == "S":
                    C2.append(c)
        if len(C2) == 0:  # すべての敵に勝利した
            return program
        C = C2

T = int(input())
for t in range(1, T+1):
    A = int(input())
    C = [input() for _ in range(A)]
    programs = []  # 3^7 + 3^6 + 3^5 + 3^4 + 3^3 + 3^2 + 3^1 = 3279 個のプログラム
    for l in range(1, 8):  # length.
        programs += permutation(l, "RPS")
    result = [simulate(p, C) for p in programs]  # それぞれのプログラムのシミュレート結果
    wins = [x for x in result if x != False]     # すべての敵に勝つプログラムら
    if len(wins) >= 1:
        y = wins[0]
    else:
        y = "IMPOSSIBLE"
    print("Case #{}: {}".format(t, y))
```

implementation

```
# AC.
T = int(input())                    # テストケースの個数
```

```
for t in range(1, T+1):              # 各テストケースについて
    A = int(input())                 # 対戦相手の個数
    C = [input() for _ in range(A)]  # プログラムら
    y = ""                           # 解答
    for i in range(255+1):           # 自分のプログラムの i 文字目を決める
        if i == 255:                 # 倒せるなら 255 文字で倒せる
            y = "IMPOSSIBLE"         # 256 文字目を考える時点で負け
            break
        moves = set(c[i % len(c)] for c in C)  # プログラムらの i 文字目
        if len(moves) == 3:                    # 何を出しても誰かに負ける
            y = "IMPOSSIBLE"
            break
        if len(moves) == 1:                    # 適切に動けばすべて倒せる
            if moves == set(["R"]):
                y += "P"
            if moves == set(["P"]):
                y += "S"
            if moves == set(["S"]):
                y += "R"
            break
        if len(moves) == 2:                    # 一方とあいこ、一方に勝つ
            if moves == set(["R", "P"]):
                y += "P"
                C = [c for c in C if c[i % len(c)] != "R"]  # 倒したのは排除する
            if moves == set(["R", "S"]):
                y += "R"
                C = [c for c in C if c[i % len(c)] != "S"]
            if moves == set(["P", "S"]):
                y += "S"
                C = [c for c in C if c[i % len(c)] != "P"]
    print("Case #{}: {}".format(t, y))
```