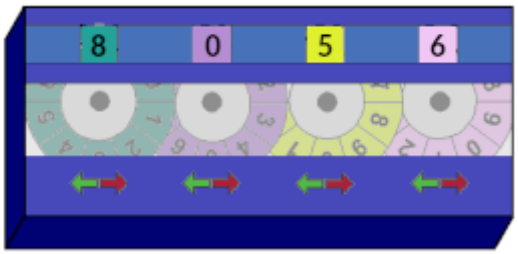
Jogo das Rodas

Codibentinho tem um brinquedo que é uma caixa com quatro rodas. Em cada roda, os dígitos de 0 a 9 estão impressos consecutivamente em sentido horário. As rodas ficam em uma caixa e, na parte superior de cada roda, existe uma janelinha. Os dígitos que aparecem nas janelas formam um número de quatro dígitos. Por exemplo, na figura a seguir, as janelinhas formam o número 8056. Essa é uma *configuração* da caixa.



Cada roda tem dois botões que formam setas. A seta verde gira a roda no sentido anti-horário e a vermelha gira no sentido horário. Com isso o número mostrado na caixa muda. Por exemplo, se o número mostrado for 8056 e apertarmos a primeira seta esquerda, o número será 9056. Se apertarmos a seta esquerda novamente, o número será 0056.

O jogo começa em uma configuração inicial qualquer. Existe uma série de configurações proibidas que jamais devem ser introduzidas na caixa. Neste exercício você deverá escrever um programa que encontra a menor quantidade de passos de uma configuração arbitrária até a configuração 0000 sem jamais passar por qualquer configuração proibida.

Entrada e Saída

A entrada conterá vários casos de teste.

Cada caso de teste começa com um inteiro I que indica a configuração inicial da caixa. Na linha seguida há uma lista que contém todas as configurações proibidas.
O fim da entrada é indicado quando $I=-1$. Este não é um caso de teste.
Como saída imprima, para cada caso de teste, o número mínimo de passos para alcançar a configuração 0000 sem jamais passar por qualquer configuração proibida.

Dicas

Estratégia de Resolução

Use o algoritmo de busca estudado em aula para encontrar o caminho mais curto até a configuração 0000. Para isso, crie uma classe chamada `Estado` que representa os

números mostrados atualmente na caixa. Gere um estado para a configuração inicial e insira esse estado em uma fila de prioridades.

Sua classe deve ter os atributos `numero`, `proibidos` e `custo` (f$\hat{}$). O atributo `numero` representa a configuração atual. O atributo `proibidos` deve ser um conjunto (ou uma lista) que contém todas as configurações proibidas. Já o custo deve ser calculado como a soma f=g+h$\hat{}$=$\hat{}$+h, sendo que g$\hat{}$ é o número de botões que já foram apertados e hh é a heurística, isto é, o palpite de quantos botões ainda precisam ser pressionados para chegar até a configuração 0000.
Para calcular hh, lembre-se que cada roda pode girar no sentido horário ou no sentido anti-horário. Então, se a primeira roda (por exemplo) estiver mostrando o dígito 3, serão necessários pelo menos 3 giros no sentido anti-horário para que essa roda mostre o dígito 0. Por outro lado, se o número mostrado for 6, então serão necessários pelo menos 4 giros no sentido horário. Calcule a melhor estratégia para cada roda e some todos os valores a fim de encontrar o valor de hh.
Sua classe deve também ter um método `transicoes()` que gera as possíveis transições de um estado para os próximos. Uma transição é a configuração resultante de pressionar um botão. Tome cuidado para não gerar nenhuma transição proibida e lembre-se de sempre fazer o cálculo do custo (g+h$\hat{}$+h). O melhor jeito de garantir que esse cálculo será feito é realizá-lo no construtor da classe `Estado` (função `__init__`).
Para ajudar, o código inicial contém as funções `valorBotao(config, botao)` e `giraBotao(config, botao, sentido)`. Em ambos os casos, `config` é um inteiro não negativo menor que 10.000 que representa a configuração. `botao` é um inteiro entre 1 e 4 que indica qual dos botões você quer verificar (da esquerda para a direita) e `sentido` deve ser `'h'` ou `'a'`, isto é, "horário" ou "anti-horário".

Em cada iteração do algoritmo, você deve tirar o primeiro elemento da fila de prioridades, gerar as transições e inseri-las de volta na fila. Se você quiser tornar o algoritmo mais eficiente, você pode criar um conjunto que contém as configurações já visitadas, para que o mesmo estado não seja inserido na fila duas vezes. Para que a fila fique correta, sobrescreva o método `__lt__` da sua classe `Estado`. Ela deve retornar `True` se e somente se o objeto `self` tiver menor custo do que um outro objeto passado como argumento (g+h(+h).
Pare quando o estado obtido indicar que o número mostrado é 0000. Como sempre haverá pelo menos um caminho até o estado final, não é preciso preocupar-se com a situação na qual a fila fica vazia.

Exemplos de Entrada e Saída

Entrada	2000 [1000]

	3681 [] 1111 [111,1011,1101,1110,2011,2101,2110,211, 1201,1210,121,1021,1120,112,1012,1102] -1
Saída	4 10 8
Entrada	4444 [3,1544,10,14,9744,1044,534,1054,543,545,34,1064,554,43,45,1074,53,55,1084,63,65,1094,73,7 45,1844,6454,5434,834,5443,5445,9544,843,845,5454,4434,9044,854,344,4443,4445,9054,8544,44 -1
Saída	24
Entrada	1337 [1,900,9,10,1419,9100,1290,2319,1299,1300,1429,2200,2329,1309,1310,1437,1438,2209,2337,233 8888 [1,3587,3589,7,7688,10,3598,18,19,2578,2587,2589,6688,2598,1588,5688,1088,578,9288,1098,58 6889,2798,6898,5878,1788,5887,5889,1288,5898,4878,9488,278,4887,4889,8988,287,289,4898,387 587,7589,2988,7598,6578,2488,6587,6589,6598,5578,1488,5587,5589,9688,988,5598,478,4578,918 9999 [9888,9889,8898,8899,9988,9989,8998,9898,9899,9998,9999,8888,8889,8988,8989] 5555 [1,2818,2565,8,9,1545,11,1546,1547,1548,2828,1808,17,1554,20,2838,1559,5655,1817,1819,28,1 1655,1656,1657,1659,1918,3455,900,1668,1928,908,910,1938,658,917,919,2455,6555,1948,928,19 -1
Saída	10 26 4 20