Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação – ICMC
Departamento de Ciências de Computação – SCC

SCC-218 – Projeto de Algoritmos

Professor Gustavo Batista
gbatista@icmc.usp.br

Projeto – *Backtracking*
Data limite para entrega: 14/09 – 23:59
Endereço para entrega: https://www.hackerrank.com/scc218-projeto-1

Descrição

Jan-Ken-Puzzle é um jogo criado em 2017 para explorar quebra-cabeças procedurais. Há diferentes algoritmos eficientes para a geração de configurações iniciais com solução garantida e conhecida (de n^2 a $n \log n$). Porém, dada uma configuração inicial, descobrir uma solução para o jogo é uma tarefa um pouco mais complicada.

Neste trabalho, você deverá desenvolver um algoritmo de ***backtracking*** para resolver diferentes configurações iniciais para Jan-Ken-Puzzle. **Obs.:** é possível solucionar o jogo com programação dinâmica, mas as entradas de teste, neste projeto, estão dentro de limites suficientemente pequenos para serem solucionáveis por *backtracking*.

Jan-Ken-Puzzle (JKP) é inspirado no famoso jogo de tabuleiro **Resta 1**, e incorpora o ciclo existente em Jan-Ken-Pon (ou Pedra-Papel-Tesoura). Um tabuleiro de JKP consiste de cédulas que podem ser vazias, ou conter uma dentre três possíveis peças: pedra, papel ou tesoura.

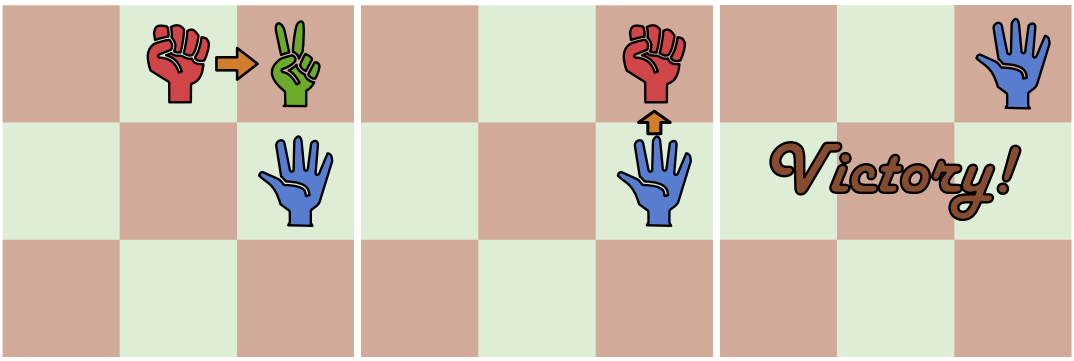
O objetivo do jogo é efetuar movimentos a fim de levar o tabuleiro a conter apenas uma única peça.

Há três movimentos possíveis:

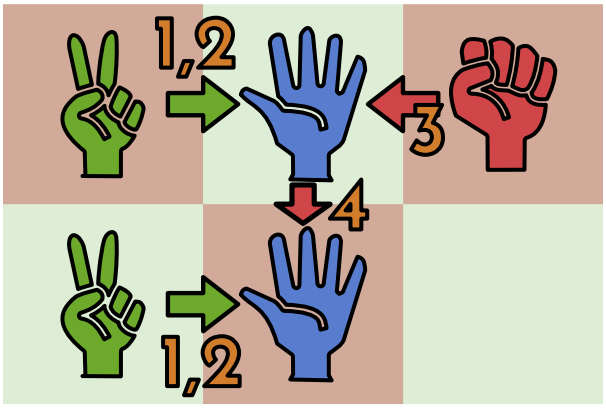
- Mover uma **pedra** para uma cédula adjacente, se a casa adjacente contiver uma **tesoura.** A **tesoura** é descartada no processo;
- Mover uma **tesoura** para uma cédula adjacente, se a casa adjacente contiver um **papel.** O **papel** é descartado no processo; e
- Mover um **papel** para uma cédula adjacente, se a casa adjacente contiver uma **pedra.** A **pedra** é descartada no processo.

As cédulas são consideradas adjacentes apenas nos eixos cartesianos (*i.e.*, não conta a diagonal).

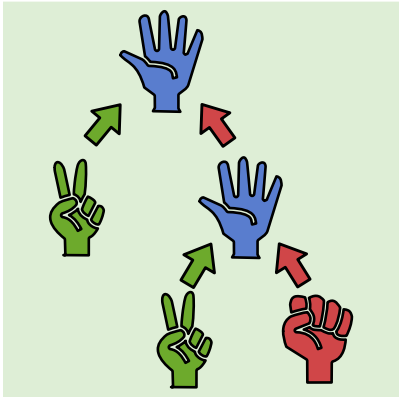
As figuras abaixo ilustram este processo.



Uma característica interessante do Jan-Ken-Puzzle é a de que toda a solução pode ser representada por uma árvore, na qual o pai de um nó representa a posição destino de uma peça. É possível observar que todas as sub-árvores filhas de um nó podem ser resolvidas independentemente e em qualquer ordem, mas a ordem que os filhos sobreescrevem o pai é relevante. A independência das sub-árvores faz com que o número de maneiras distintas de se efetuar uma **mesma solução** cresca exponencialmente. A figura a seguir ilustra um tabuleiro com uma solução demarcada que possui ramificações que podem ser resolvidas em qualquer ordem.



A árvore da solução apresentada é a que segue:



Por fim, um tabuleiro pode conter mais de uma solução que leva a um mesmo estado final. O tabuleiro mostrado anteriormente contém mais uma solução. Você consegue encontrá-la? Quantos são os estados finais possíveis para esse tabuleiro?

Entrada

Os casos de teste serão fornecidos pela entrada padrão (`stdin`). A primeira linha do arquivo de entrada contém dois números inteiros, *R* e *C*, que indicam o número de linhas e de colunas do tabuleiro, respectivamente. O tabuleiro terá área máxima de **16 cédulas** ($R \times C \leq 16$) e possuirá no máximo **11 peças**.

As próximas *R* linhas contém *C* números inteiros separados por um character de espaço. O *j*-ésimo número da *i*-ésima linha corresponde ao estado da cédula na linha *i*, coluna *j* do tabuleiro. Cada cédula é representada por um valor entre 0 e 4, conforme descrito a seguir:

0. Cédula vazia;
1. Cédula contém uma pedra;
2. Cédula contém uma tesoura; e
3. Cédula contém um papel.

Há solução para todas as entradas providas.

Saída

A saída deverá ser fornecida para a saída padrão (`stdout`). A saída de erro (`stderr`) poderá ser utilizada para ajudar na depuração do código, mas será desconsiderada na avaliação. Observe, no entanto, que impressões desnecessárias dentro do *backtracking* podem prejudicar a performance de execução do programa.

A primeira linha da saída deve conter a quantidade de maneiras diferentes de se chegar a um estado final qualquer (quando o tabuleiro tem apenas uma peça).

A segunda linha da saída deve conter o número inteiro *A* que representa quantos estados finais **distintos** existem, sendo que um estado final é descrito pela posição do tabuleiro onde está a última peça e qual o seu tipo.

As *A* linhas seguintes devem apresentar os estados finais distintos, ordenados por linha, desempatando por coluna e por tipo de peça, nessa ordem. As linhas e colunas são representadas por números de 1 a *R* e de 1 a *C*, respectivamente. A *i*-ésima linha deve apresentar linha, coluna e peça, separados por um espaço em branco, nessa ordem, correspondente ao *i*-ésimo estado final.

Exemplos de entrada e saída

Caso 1

Entrada	Saída
2 2 1 2 0 3	1 1 1 2 3

Caso 2

Entrada	Saída
2 3 2 3 1 2 3 0	4 2 1 1 1 2 2 1

Dicas

- A representação em árvore da solução que foi apresentada não é necessária para solucionar o problema por *backtracking*. É possível simplificar o problema pensando apenas em todos os movimentos individuais possíveis para um estado.
- É possível podar o a árvore de busca do *backtracking* quando as peças formam “ilhas” disjuntas no tabuleiro. Por que?