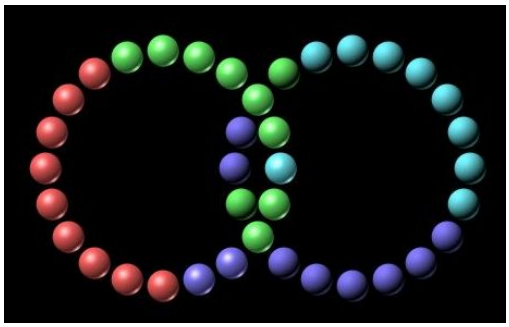
Puzzle dos Colares

Projeto nº 1

Introdução à Inteligência Artificial - edição 2020/21

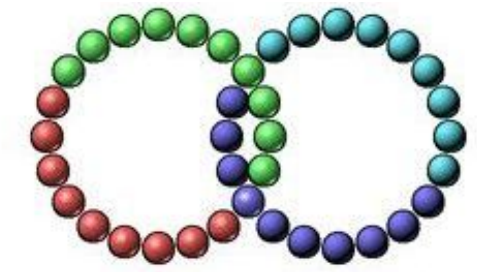
Entrega: 7 de Outubro (1m antes da meia-noite)



O Puzzle dos 2 Colares

Temos dois colares, ambos com 20 bolas cada e em que duas das bolas são partilhadas, i.e. existem 38 bolas no total (ver figura em cima). Existem duas cores com 10 bolas, (na figura, o verde e o roxo) e outras duas com 9 bolas (o azul e encarnado). Na figura em cima, as bolas partilhadas pelos dois colares são de cor verde. Dada uma configuração do puzzle, podemos rodar quaisquer dos dois colares uma bola no sentido horário ou no sentido anti-horário. Por exemplo, se aplicássemos ao colar da direita a rotação horária de uma bola, seriam de cor roxa as duas bolas partilhadas.

Atinge-se o objectivo quando cada um dos colares tem concentradas e sem interrupções duas das cores, uma de 9 e outra de 10 bolas. A figura a seguir é um exemplo de uma solução final. O anel esquerdo concentra as bolas verdes e encarnadas em duas sequências sem interrupções e o anel direito concentra as duas restantes cores, o azul e roxo, também em sequências sem interrupções. Notem que em qualquer solução final haverá sempre em cada anel uma bola de uma terceira cor, que aparece isolada separando as duas sequências ininterruptas.



Objetivos

Modelem este puzzle como um problema de procura num grafo, de acordo com o Paradigma do Espaço de Estados, usando a implementação disponibilizada pelo módulo ***searchPlus.py***.

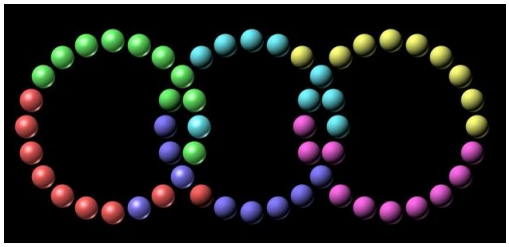
Importante

1. Para facilitar a correcção, a classe do Problema que criarem tem de se chamar **PuzzleColares**, mas a construção da instância do problema é livre quanto ao tipo e número de argumentos usados.
2. Ao construirem o estado inicial terão de verificar se é válido. Cada um dos colares tem de ter 20 bolas, há duas bolas que fazem parte de ambos os colares e há duas cores com 10 bolas cada e outras duas com 9 cada. Podem escolher as cores da figura e do enunciado ou outras;
3. Notem que terão que implementar o método ***display()*** que permite visualizar de modo *pretty* (em modo de texto) o estado do puzzle, para evitar mostrar os estados através da visualização bruta das estruturas de dados.
4. Tenham atenção que se um estado s for input do método ***result()*** ele deve permanecer exactamente igual, não sendo modificado pela execução do método. Deve ser gerado um estado completamente novo s' que é devolvido e não alterar o estado s de input do método!
5. É importante que dois estados exactamente com os mesmos valores nos seus atributos sejam considerados iguais mesmo que sejam objectos distintos!

Extras

O puzzle dos colares pode ter mais do que 2 colares. Por exemplo, na figura seguinte temos um puzzle com 3 colares em que há 6 cores: duas com 10 bolas e as restantes quatro cores com 9, e em que há 4 bolas partilhadas. Os colares das pontas partilham ambos 2 bolas com o colar do meio, que assim partilha as 4 bolas. Na solução final, cada um dos colares das pontas concentra duas das cores em sequências ininterruptas de 10 e 9 bolas e o colar do meio concentra duas das cores em sequências ininterruptas de 9 bolas.

Daremos um bónus de 0.25 valores aos alunos que modelarem o problema de modo a podermos ter puzzles com quaisquer número de colares.



Teste dos programas

Fazendo uso das funcionalidades do notebook que permite a coexistência de texto e código, convém demonstrarem que a construção das instâncias avisa sobre os vários casos de estados iniciais inválidos; que os métodos ***actions()***, ***result()***, ***path_cost()*** e ***goal_test()*** estão a funcionar bem, ilustrando com exemplos. Não se esqueçam de explicar o 'pretty display' se não for muito intuitivo, e demonstrem também que não modificam os estados que sofrem as acções e que a igualdade dos estados não depende destes serem objectos diferentes.

O método ***exec()***, definido a seguir, permite executar uma sequência de acções a partir de um estado qualquer, devolvendo o estado resultante e o custo acumulado num par *(estadoRes,custoTotal)*. Podem usá-lo para testarem a execução de uma sequência de diversas acções partindo de um estado particular.

```
    def exec(p,estado,accoes):
        custo = 0
        for a in accoes:
            seg = p.result(estado,a)
            custo = p.path_cost(custo,estado,a,seg)
            estado = seg
        return (estado,custo)
```

Para fazerem um teste mais forte, podem começar por gerar um puzzle solução e depois executam uma sequência de poucas acções, 5 ou 6. A seguir, podem gerar um problema com a configuração que resultou dessa sequência como estado inicial. Finalmente, podem executar, por exemplo, o algoritmo de procura em largura em árvore, para verificar se encontra a solução, que é exactamente a sequência de acções espelho das usadas para gerar o estado inicial a partir do final. Se o algoritmo de procura encontrar a solução é sinal que a modelação está correcta!

Entrega

Código e Relatório

O relatório é **obrigatório** e também o formato do ficheiro de submissão: o código e o relatório têm de ser entregues conjuntamente num único ficheiro Jupyter Notebook. Qualquer trabalho que não tenha relatório (só o código) ou que não cumpra esse formato não é avaliado e tem 0 de nota.

No notebook podem explicar a modelação em python para os estados e acções, ilustrar e correr o código e apresentar os testes que fizeram.

Nós fornecemos um ficheiro esqueleto, ***IIA2021-proj1-XX.ipynb***, (substituam XX pelo número do grupo). Não se esqueçam de preencher os nomes e números dos elementos do vosso grupo.

Código a não ser alterado

Não alterem nem o ***utils.py*** nem o ***searchPlus.py*** e **não os devem submeter!** Iremos correr os mesmos 2 ficheiros para todos.

Prazo

Um único ficheiro ***IIA2021-proj1-XX.ipynb*** até ao dia **7 de Outubro** às 23:59