Henwasův problém

Úloha

Po mnoho let pracoval ve starém doku muž jménem Henwas, jehož úkolem bylo přemisťovat bedny. Každá zásilka dorazila v náhodném pořadí a podle jeho starých poznámek měl zboží přeuspořádat do nové konfigurace. Jak Henwas stárnul, začala ho bolet záda a nechtěl dělat zbytečnou práci. Proto se obrátil na vás s prosbou o pomoc. Pomozte chudákovi Henwasovi najít optimální řešení jeho problému!

Nedávno dorazila zásilka tří beden označených písmeny A, B a C. Henwas se podíval do svých poznámek a zjistil, že konečná konfigurace by měla být B-A-C:

```
[[B, A], [C]] -> [[B, A, C]]
```

Aby ušetřil místo, používají poznámky zkrácený kód k popisu konfigurací, což odpovídá předchozímu zápisu. Henwas byl docela sebevědomý, a tak na vás nečekal a začal přesouvat bedny sám. Samozřejmě mohl přesouvat pouze ty bedny, které byly nahoře na libovolné hromadě. Také mohl použít volný prostor na zemi. Aby si pamatoval své kroky, používá číslo 0 k označení země a zapisuje akce následovně:

```
[[B, A], [C]] -> [[B, A, C]]
```

- `(B, 0)` -¿ `[[B], [A], [C]]`
- `(A, C)` -¿ `[[B], [A, C]]`
- `(B, A)` -¿ `[[B, A, C]]`

Poznámka: Pořadí hromad není důležité:

```
[[B, A], [C]] == [[C], [B, A]]
```

Vaše úloha

Zde přicházíte na řadu vy! Vaším úkolem je napsat algoritmus, který nalezne optimální sekvenci akcí k přeuspořádání počáteční konfigurace na cílovou. Naštěstí je většina práce hotová a můžete si stáhnout balíček `task1_blockworld_v3.zip`. Jediné, co musíte udělat, je implementovat A* algoritmus a heuristiku pro tento problém.

Stáhněte si nejnovější verzi v3!

Můžete si vyzkoušet prostředí spuštěním `python blockworld.py`:

```
[[1, 2], [3]] -> [[3, 2, 1]]
```

Při otevření souboru `student.py` uvidíte následující:

```
from blockworld import BlockWorld

class BlockWorldHeuristic(BlockWorld):
    def __init__(self, num_blocks):
        BlockWorld.__init__(self, num_blocks)

    def heuristic(self, goal):
        self_state = self.get_state()
        goal_state = goal.get_state()

        # TODO: Implementujte heuristiku zde.

        return 0.

class AStar():
    def search(self, start, goal):
        # TODO: Vrátí seznam optimálních akcí pro převedení startu na cíl.

        # Přístup k akcím a sousedům:
        # for action, neighbor in state.get_neighbors():
        #     ...

        return None
```

Implementujte heuristiku v `BlockWorldHeuristic::heuristic()` a A* v `AStar::search()`. Po dokončení můžete testovat algoritmus na připravených instancích dostupných ve složkách `problems/<N>/<PID>`. Algoritmus můžete vyhodnotit pomocí:

`python eval.py <N> <PID>`

Příklad:

```
$ python eval.py 7 0
7/0: [[2], [6, 3], [7], [5, 1, 4]] -> [[7, 3, 5, 2, 1, 6], [4]]
Reference | path length: 7, expanded: 1417, time: 1.26s
Yours     | path length: 7, expanded: 1417, time: 1.14s
```

Nápověda

Můžete využít `PriorityQueue` v Pythonu:

```
from queue import PriorityQueue

queue = PriorityQueue()

queue.put( (3, "Nimue") )
queue.put( (1, "Crystin") )
```

```
queue.put( (2, "Rhys") )

while not queue.empty():
    priority, name = queue.get()
    print(f"{priority} - {name}")
```

Výstup bude:

```
1 - Crystin
2 - Rhys
3 - Nimue
```

Vyhodnocení

Nahrajte svůj soubor `student.py` do systému Brute, kde bude automaticky vyhodnocen na sadě problémů s $N = 5, 6, 7, 8, 9$. Za každý N získáte 1 bod, pokud váš kód nalezne platné řešení, a další 1 bod, pokud je řešení optimální. Každý problém má časový limit.