PROGETTO CARBUROBOT

Consegna

CarbuRobot. Un robot si può muovere (move) in una sequenza di stanze, in numero di stanze qualsiasi. Nelle stanze possono trovarsi oggetti che possono essere afferrati (get) o rilasciati a terra (put) dal robot. Il robot puo' afferrare e tenere un solo un oggetto alla volta. Gli oggetti hanno un peso espresso in unità intere nonnegative. Quando il robot si muove senza oggetti consuma per ogni move 1 litro di carburante, quando si muove con un oggetto consuma per la move tanti litri di carburante quanto il peso dell'oggetto più 1. Quando afferra o rilascia l'oggetto get/put consuma tanti litri quanto il peso dell'oggetto manipolato. Es. se afferra un oggetto di 5kg consuma 5 litri. Nelle stanze possono trovarsi dei distributori di carburante, se il robot si trova in una stanza con ristributore può esreguire l'azione rifornisci (refill) che ricarica, incrementando di una quantità fissa es.+10, il carburante contenuto nel serbatotio del robot. Naturalmente se non ha sufficiente carburante non può eseguire una certa azione. Esempio Problema : dato uno stato iniziale di carburante nel robot e oggetti di pesi diversi sparsi in giro e distributori si richiede uno stato finale con posizioni finali di oggetti.

Funzioni utilizzate in A*

h1 (Euristica del numero di oggetti fuori posto):
Descrizione: Questa euristica valuta quanti oggetti sono posizionati in modo errato rispetto allo stato obiettivo. Conta il numero di oggetti presenti nelle stanze nello stato corrente che non coincidono con gli oggetti nelle stanze dello stato obiettivo.
Questa euristica, quindi, permette di stimare quanti oggetti devono essere spostati o scambiati di posizione per far coincidere il tuo stato corrente con lo stato obiettivo.
h2 (Euristica del numero di stanze fuori posto):
Descrizione: Questa euristica valuta quanti spazi o stanze sono posizionati in modo errato rispetto allo stato obiettivo. Conta il numero di stanze che contengono oggetti diversi rispetto alle stanze dello stato obiettivo.
L'utilità è quella di riuscire a stimare quanti spazi o stanze devono essere "riordinate" o "ripiene" per raggiungere lo stato obiettivo.

Algoritmi utilizzati

Iterative Deepening Search (IDS):

è un algoritmo di ricerca iterativa che combina le caratteristiche di una ricerca in profondità con la capacità di trovare soluzioni ottimali.
IDS è chiamato con l'oggetto problem come argomento, che rappresenta il problema specifico da risolvere.

display=True indica che l'algoritmo dovrebbe mostrare l'output durante l'esecuzione.
s3 conterrà le informazioni sulla soluzione trovata.
s3.solution() restituirà la sequenza di azioni che costituiscono la soluzione.
s3.state conterrà lo stato finale raggiunto dalla ricerca.

Breadth-First Tree Search:

è una strategia di ricerca in ampiezza che esplora tutti i successori di un nodo prima di passare ai successori dei suoi successori.
BFS è chiamato con l'oggetto problem come argomento, che rappresenta il problema specifico da risolvere.
display=True indica che l'algoritmo dovrebbe mostrare l'output durante l'esecuzione.
s3 conterrà le informazioni sulla soluzione trovata.
s3.solution() restituirà la sequenza di azioni che costituiscono la soluzione.
s3.state conterrà lo stato finale raggiunto dalla ricerca.

Uniform Cost Search:

è un algoritmo basato su costo che esplora lo spazio degli stati minimizzando il costo totale.
Uniform_cost_search è chiamato con l'oggetto problem come argomento, che rappresenta il problema specifico da risolvere.
display=True indica che l'algoritmo dovrebbe mostrare l'output durante l'esecuzione.
sol2 conterrà le informazioni sulla soluzione trovata.
sol2.solution() restituirà la sequenza di azioni che costituiscono la soluzione.
sol2.state conterrà lo stato finale raggiunto dalla ricerca.

A Search con Euristiche:

è un algoritmo basato su euristiche che cerca di minimizzare il costo totale stimato.
Astar_search è chiamato con l'oggetto problem come argomento, che rappresenta il problema specifico da risolvere, e una specifica euristica (problem.h1 o problem.h2) per guidare la ricerca.
display=True indica che l'algoritmo dovrebbe mostrare l'output durante l'esecuzione.
s3 conterrà le informazioni sulla soluzione trovata.
s3.solution() restituirà la sequenza di azioni che costituiscono la soluzione.
s3.state conterrà lo stato finale raggiunto dalla ricerca.
Inoltre, il codice utilizza il modulo time per calcolare il tempo di esecuzione di ciascun algoritmo di ricerca.
La logica del tempo di esecuzione è semplice: prima dell'avvio della ricerca viene dato lo start() al contatore del tempo, poi viene eseguito l'algoritmo di ricerca e all' uscita da questo (soluzione trovata) viene stoppato il contatore (stop()).

	PROBLEMA 1	PROBLEMA 2	PROBLEMA 3
Iterative Deepening Search	Examined 64 Nodes. Iterative Deepening Search Solution: ['right', 'get_candle', 'right', 'put_candle']. Iterative Deepening Search Time: 0.002848386764526367 seconds	Examined 97306 Nodes. Iterative Deepening Search Solution: ['refill', 'right', 'right', 'get_bag', 'right', 'put_bag', 'left', 'left', 'get_candle', 'right', 'put_candle', 'right']. Iterative Deepening Search Time: 2.552614212036133 seconds	Examined 120228 Nodes. Iterative Deepening Search Solution: ['put_bag', 'right', 'refill', 'refill', 'right', 'right', 'get_candle', 'right', 'put_candle', 'left', 'left', 'get_desk', 'right', 'right']. Iterative Deepening Search Time: 4.5215277671813965 seconds
Breadth-First Tree Search Solution	Examined 72 nodes. Breadth-First Tree. Search Solution: ['right', 'get_candle', 'right', 'put_candle']. Breadth-First Tree Search Time: 0.006258726119995117 seconds.	Examined 118239 nodes. Breadth-First Tree Search Solution: ['refill', 'right', 'right', 'get_bag', 'right', 'put_bag', 'left', 'left', 'get_candle', 'right', 'put_candle', 'right']. Breadth-First Tree Search Time: 5.324701547622681 seconds.	Examined 160326 nodes. Breadth-First Tree Search Solution: ['put_bag', 'right', 'refill', 'refill', 'right', 'right', 'get_candle', 'right', 'put_candle', 'left', 'left', 'get_desk', 'right', 'right']. Breadth-First Tree Search Time: 7.659287929534912 seconds.
Uniform Cost Search Solution	Examined 109 nodes 43 paths have been expanded and 40 paths remain in the frontier. Uniform Cost Search Solution: ['right', 'get_candle', 'right', 'put_candle']. Uniform Cost Search Time: 0.009505748748779297 seconds	Examined 3307 nodes 1462 paths have been expanded and 498 paths remain in the frontier. Uniform Cost Search Solution: ['refill', 'right', 'right', 'get_bag', 'right', 'put_bag', 'left', 'left', 'get_candle', 'right', 'put_candle', 'right']. Uniform Cost Search Time: 0.5799834728240967 seconds.	Examined 2350 nodes 1069 paths have been expanded and 313 paths remain in the frontier. Uniform Cost Search Solution: ['put_bag', 'right', 'refill', 'refill', 'right', 'right', 'get_candle', 'right', 'put_candle', 'left', 'left', 'get_desk', 'right', 'right']. Uniform Cost Search Time: 0.28685855865478516 seconds.
Heuristic Search with h1 Solution	Examined 74 nodes 29 paths have been expanded and 31 paths remain in the frontier. Heuristic Search with h1 Solution: ['right', 'get_candle', 'right', 'put_candle']. Heuristic Search with h1 Time: 0.00725555419921875 seconds.	Examined 2033 nodes 899 paths have been expanded and 363 paths remain in the frontier Heuristic Search with h1. Solution: ['refill', 'right', 'right', 'get_bag', 'right', 'put_bag', 'left', 'left', 'get_candle', 'right', 'put_candle', 'right']. Heuristic Search with h1 Time: 0.2741544246673584 seconds.	Examined 1547 nodes 715 paths have been expanded and 225 paths remain in the frontier Heuristic Search with h1. Solution: ['put_bag', 'right', 'refill', 'refill', 'right', 'right', 'get_candle', 'right', 'put_candle', 'left', 'left', 'get_desk', 'right', 'right']. Heuristic Search with h1 Time: 0.15833115577697754 seconds.
Heuristic Search with h2 Solution	Examined 32 nodes 12 paths have been expanded and 15 paths remain in the frontier. Heuristic Search with h2 Solution: ['right', 'get_candle', 'right', 'put_candle']. Heuristic Search with h2 Time: 0.001172780990600586 seconds.	Examined 1395 nodes 612 paths have been expanded and 270 paths remain in the frontier Heuristic Search with h2 Solution: ['refill', 'right', 'right', 'get_bag', 'right', 'put_bag', 'left', 'left', 'get_candle', 'right', 'put_candle', 'right']. Heuristic Search with h2 Time: 0.1483011245727539 seconds.	Examined 1146 nodes 518 paths have been expanded and 213 paths remain in the frontier Heuristic Search with h2 Solution: ['put_bag', 'right', 'refill', 'refill', 'right', 'right', 'get_candle', 'right', 'put_candle', 'left', 'left', 'get_desk', 'right', 'right']. Heuristic Search with h2 Time: 0.10745000839233398 seconds.

PROBLEMA 1

initial_state = { "robot": 0, "hand": None, "fuel": 15, "rooms": ['station', 'candle', 'null', 'null','null'] }

goal_state = { "robot": 2, "hand": None, "fuel": 5, "rooms": ['station', 'null', 'candle', 'null', 'null'] }

Il problema 1, ha le seguenti specifiche:

robot -> passa da room n.0 a room n.2;

hand -> vuota sia alla partenza che all'arrivo;

fuel -> nello stato obiettivo deve essere > di 5 unità (definito nel goal_test());

rooms -> spostamento dell'oggetto "candle" dalla room n.1 a quella n.2.

PROBLEMA 2

initial_state1 = { "robot": 0, "hand": None, "fuel": 15, "rooms": ['station', 'candle', 'bag', 'null','null'] }

goal_state1 = { "robot": 3, "hand": None, "fuel": 5, "rooms": ['station', 'null', 'candle', 'bag', 'null'] }

Il problema 2, ha le seguenti specifiche:

robot -> passa da room n.0 a room n.3;

hand -> vuota sia alla partenza che all'arrivo;

fuel -> nello stato obiettivo deve essere > di 5 unità (definito nel goal_test());

rooms -> spostamento dell'oggetto "candle" dalla room n.1 a quella n.2 e "bag" dalla room n.2 alla room n.3; inoltre è necessaria una refill (carburante non sufficiente per tutte le azioni).

PROBLEMA 3

Esempio di utilizzo 3 initial_state2 = { "robot": 0, "hand": "bag", "fuel": 5, "rooms": ['null', 'station', 'desk', 'candle','null'] }

goal_state2 = { "robot": 4, "hand": "desk", "fuel": 5, "rooms": ['bag', 'station', 'null', 'null', 'candle'] }

Il problema 3, ha le seguenti specifiche:

robot -> passa da room n.0 a room n.4;

hand -> contiene l'oggetto "bag" alla partenza, sostituito da "candle" all' arrivo;

fuel -> nello stato obiettivo deve essere > di 5 unità (definito nel goal_test());

rooms -> spostamento dell'oggetto "candle" dalla room n.3 a quella n.4 e "bag" dalla room n.2 alla hand del robot; inoltre sono necessarie più refill (carburante non sufficiente per tutte le azioni).

Marco Amici