Artificial Intelligence Assignment 2 Readme Document

Author:
Joshua Wang
UNI: jw3123

1. The programming language in the project is Python.
2. The version of programming language is Python 2.7.3.
3. Development environment is:
   Application: Sublime text2 with python 2.7.3
   Operating System: Windows 8
   Processor: Intel i7-4702MQ
   Memory: 8GB DDR3
4. How run and compile on CLIC:
   (1) Put all .py files and the file of the puzzle in the same directory
   (2)In the directory, type:

python commandline.py <puzzle's filename>

A command line, indicating how to choose search algorithm, will prompt.
After finding a solution, it will ask whether to show the move step by step, type y to confirm, or type anything else to decline.

5. Explanation of how the program functions:

Commandline.py imports all search modules. All search modules import tree.py as the data structure of tree and warehouse.py for the state, as the configurations of the puzzle. Greedy BFS and A* search also import heuristics.py, which contains the heuristic functions for states.
They are: (Distance means Manhattan distance)
Heuristic_1: the distance between each box not in goal and its nearest goal.
Heuristic_2: the distance between each box and its nearest goal+ the distance between the worker and all box not in the goal.
Both heuristics are admissible because the true cost includes walking of the worker to the correct "pushing position" and push the box, which is clearly greater than heuristic 1 since it does not count worker's move at all, and, heuristic 2 because the distance between the worker and boxes not in the goal is the minimum the worker can move.

How actions cause new states, which is represented by a 2-dimensional list, is include in Warehouse.move(). Given the move represented as a string, it first checks whether it is a legal move. Then it checks whether the move is pushing a box. These cases are handled by replacing strings in the list. After each move, the status of worker's position and whether the goal (all boxes are in the goal) is achieved.

The data structure of tree was implemented by a recursive defaultdict in python. Each value in the dictionary is the same type of defaultdict. This structure corresponds to the tree and the subtree, therefore, we can store and retrieve each node as the root of the subtree. We also record the path from root to a node, so the node can be achieved and if it is the goal state, that path is the solution.The function addandreturnnode() add a note to a parent, resetting the path to it and return the node. This is useful in adding and handling a new node to the search tree.
A node contains the state (a warehouse object), and the cost and expected cost values (Reference: https://gist.github.com/hrldcpr/2012250)

The frontier is implement by deque, which can simulate as a stack or a queue, and the value of element in it can be easily acquired and compared and replaced. In BFS, the deque is simulated as a queue, in DFS, a stack, in UCS, greedy BFS, and A*. We use it as a heap, to pop the element with the smallest key. (The element in it is store as two-tuple (node.state.(expect)cost, node) , and the key is the node.state.(expect) cost)
Also, as illustrated in textbook, in UCS, greedy BFS, and A*, a node is checked whether it is the goal after popping from the frontier, contrary to BFS and DFS, which are checked when expanded.

UCS updates cost (stored in the class node) when expanding (worker moves: cost 1, pushes a box, cost 2), greedy BFS, and A* updates expect cost (stored in the class node) when expanded. For the former, it is simply the value of heuristics. For the latter, it is the sum of heuristics and cost until that point.