CSCE 625

Assignment -1

Fall 2017

Pentyala, Shiva Kumar – UIN: 127003995

10-5-2017

Objective:

To implement A* search and develop a heuristic that will solve the Blocksworld problem more efficiently.

Implementation:

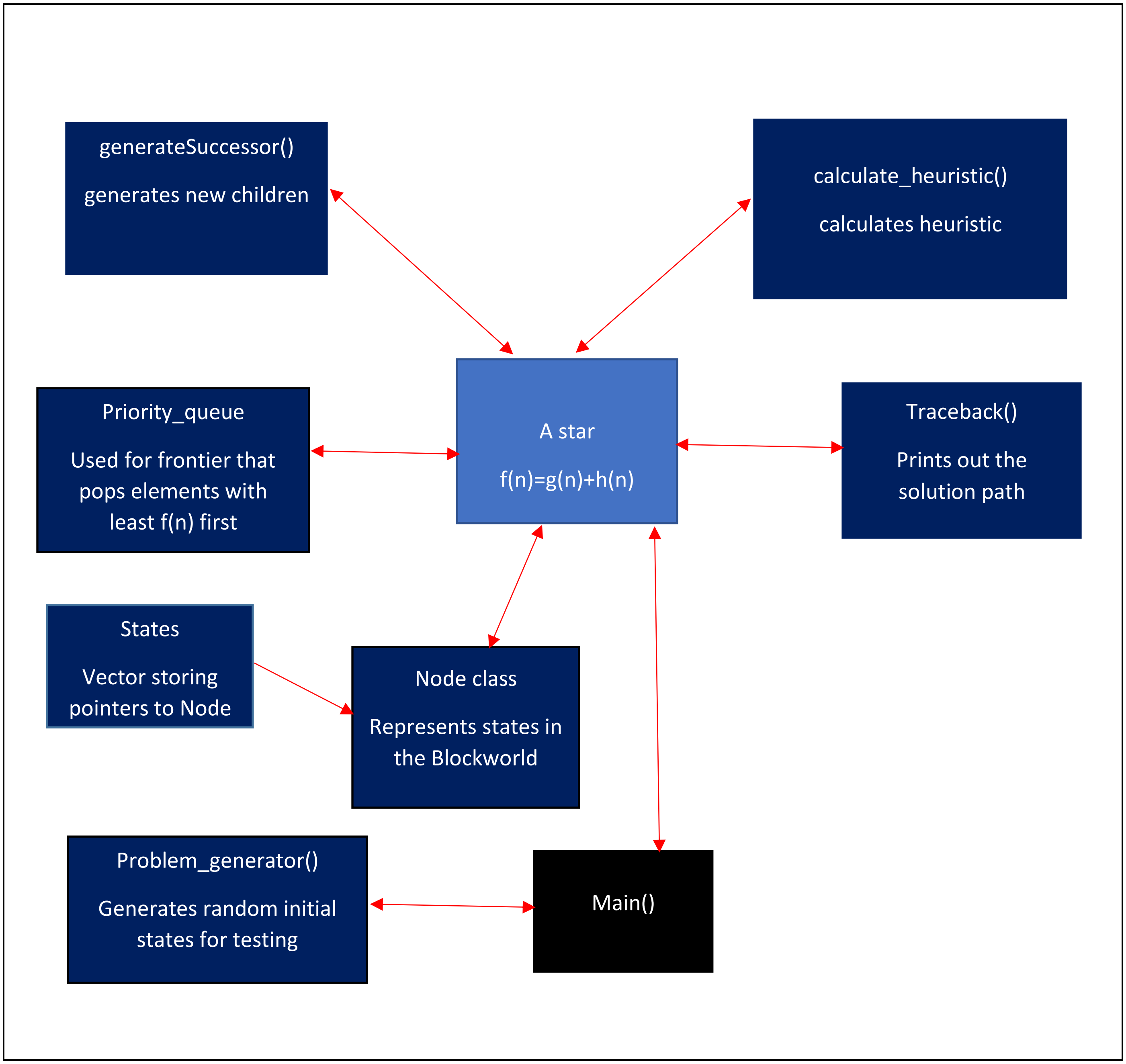
generateSuccessor()
generates new children
calculate_heuristic()
calculates heuristic
Priority_queue
Used for frontier that pops elements with least f(n) first
A star
f(n)=g(n)+h(n)
Traceback()
Prints out the solution path
States
Vector storing pointers to Node
Node class
Represents states in the Blockworld
Problem_generator()
Generates random initial states for testing
Main()

Table of Performance Metrics:

Blocks	Stacks	Total Goal Tests/ Number of Iterations	Max Queue Size	Depth	Time (Seconds)
5	3	**9**	18	**9**	0.0014 **Success**
5	10	**7**	67	**7**	0.0634 **Success**
8	4	**13**	78	**13**	0.015 **Success**
10	5	**15**	157	**15**	0.059 **Success**
10	10	**14**	481	**14**	0.64 **Success**
15	10	**24**	966	**24**	3.60 **Success**
15	15	**21**	1531	**21**	9.24 **Success**
18	15	**25**	1684	**25**	11.44 **Success**
20	15	**29**	2616	**29**	20.64 **Success**
23	17	**39**	3687	**39**	52.36 **Success**
25	18	**38**	4126	**38**	82.24 **Success**
26	20	**41**	5930	**41**	174.18 **Success**
26	26	**41**	7208	**41**	378.02 **Success**

Heuristic Description:

The below heuristic is used to solve the problem efficiently.

h = 22.5 * **p_0** - 12.5 * **p_2** - 1.5***p_1** + 5.5***sum_1** + 10.5***sum_2**

p_0 -> Total Number of blocks out of place on all the stacks.

p_1 -> Number of blocks out of place on the first stack compared to Goal state.

p_2 -> Total Number of blocks on all the stacks except the 1st stack.

sum_1 -> Each block 'b1' on the stacks other than 1st stack is compared with its below blocks on the same stack and if ASCII (b1) > below block, then 1 point is incremented for b1. This check for b1 will be done with all of its below blocks one by one and then moving to next stack and repeating this process for all the elements on that stack.

sum_2 - > This is only for the blocks on 1st stack and its opposite to that of the above sum_1 calculation procedure. If for each block on stack 1 let's say 'b1' if ASCII(b1) < below block then 1 point is incremented. This check for b1 will be done with all the blocks which are below it.

- Coefficients are added by analyzing the weightage ratio that needs to be given for each and their values are assigned by trial and error after checking with multiple test cases.

A* Algorithm used:

(f,g are initially set to infinite, h=0 for every node.)

```
function AStar returns a solution or failure
States <- explored set
frontier <-  priority queue ordered by PATH-COST (f(n)), with only one initial element.
loop do
    IF frontier empty then return failure
    node <-POP from frontier /* Chooses the lowest cost node in the frontier based on f(n)*/
    IF GOAL-TEST(node.STATE) then return Solution(node)
    IF iterations > 1000 return failure
    child_node <- Generate children
    For each generated children:
        g'= (node->g) + 1;
        IF child_node is not in explored or frontier then
                Child_node - >g =g'
                Calculate heuristic and f=g+h
                Add child_node.state to explored (States)
                Frontier <- Insert(child_node,frontier)
        ELSE IF child_node is in explored or frontier with higher Path Cost then
                  Replace that with new child_node
```

Discussion:

- For most of my test cases, I observed that **Total Goal tests = Depth**. That means I'm directly moving towards the goal without visiting unnecessary nodes. Hence I strongly think that my Heuristic is **admissible** as its not overestimating and also in many cases its prediction is same as the true path length to the goal.

- I was able to solve 26 blocks and 25 stacks with an average depth of 41, which seems pretty good for me. Although there are few cases where Total Goal tests is little more than the depth but still final path is optimal.

- During implementation I felt that estimating sum_1 and sum_2 are crucial for the heuristic to work effectively. Initially when I was using a simple heuristic with number of blocks out of place it was doing >1000 Goal tests for 7 blocks 5 stacks. So adding sum_1 and sum_2 to heuristic have improved the search by a great factor.

- Using the C++ STL library's Priority Queue, I was unable to iterate through its elements so I have extended its functionality in order to iterate through its elements.

- Number of Iterations/ Goal Tests is NOT exceeding 41 for any number of blocks and stacks input.

Ideas to Improve:

- Queue Size is >2000, if more than 20 blocks is given as input. Although Heuristic is working perfectly because of the excess queue size program run time is increasing as it needs to iterate through all the elements of Priority queue to check if the node already exists. This checking will be done for every node. Which would be of the order O(n^2).

- Additionally, Priority queue is built using heap and delete operation in heap causes a lot of trouble as heap doesn't support search by value. This would be O(n*log(n)). Hence I would suggest using a Hash Map/Unordered set/ 2-3 tree which does search, delete, insert operations in O(1), O(log n) time respectively. This way I think we can reduce the Running time.

Thank You so much, Patrick