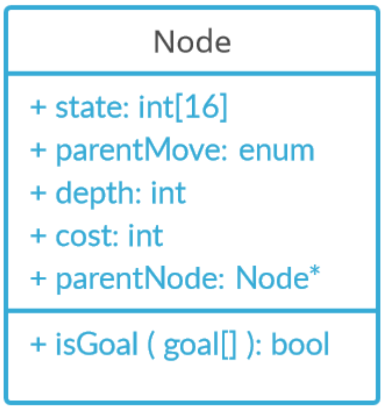
CSE 4082 – Project 1 Design Document

Büşra Gökmen - 150116027
Ömer Faruk Çakı - 150117821

1.1 Node Class



Class diagram of Node Class

Name	**Type**	**Description**
Node	Class	Node class is the data structure we used for the node representations. It contains the state of the board
Directions	Enum	Enumeration of available valid moves.
state	Array<int>	1D array representation of the board state.
depth	int	Depth of the current node
cost	int	Cost of the node so far
parentMove	Directions	The move that caused that state (from parent)
parentNode	Node *	Pointer that points the parent node which produced this node
isGoal()	function	Compares the state of the current node with the goal state to find whether the node is the goal state or not.

1.2 Variables/Data Structures

Variable	Description
`int expanded`	Number of expanded nodes.
`int maxNodes`	Number of nodes stored in memory(added in frontier list)
`std::vector<Node *> frontier`	Frontier list, contains Nodes.
`std::unordered_map<std::string, bool> expandedList`	Explored set. Holds the state information of previously expanded nodes to not to expand another node with the same resulting state.

1.3 Functions

Function	Description
`main(int argc, char **argv)`	Main function of the program. It takes command line arguments(input file to be read, algorithm of choice) from user and executes corresponding algorithm.
`addExpanded(&expandedList,Node.state)`	The function which used to save the state of the currently expanded node to the explored set.
`expand_node(&expandedList, &frontier, Node *node)`	The function that expands the provided node by adding it's children moves(steps) to the frontier list if the resulting state is not previously visited. Returns number of nodes which newly added to frontier list.
`expand_node_ils(&expandedList, &frontier, Node *node, int limit)`	Same with the `expand_node` function but only used by Iterative Lengthening Search. Gets an additional limit parameter and doesn't expand new nodes if their costs exceed the limit.
`addIfNotExpanded(&expandedList, &frontier, Node *node, int &count)`	The function receives a Node and checks it's state if it's a previously discovered state or not. If it's a new state add the state to explored set by calling `addExpanded` function and adding the Node to

	frontier list.
`addExpanded(&expandedList, state[])`	Received a state[16] and insert into explored set.
`isExpanded(&expandedList,state[])`	Receives a state[16] to check. Return true if the state is in the explored set, false otherwise.
`convertStateString(int state[])`	This is the core function of the explored set operations. It’s some kind of basic hashing function. In order to store explored states more efficiently this function takes an integer array and converts into string. Memory usage is reduced to 16 bytes from 64 bytes for each state saved on the explored set.
`Manhattan(x1,y1,x2,y2)`	Calculates a Manhattan distance between two points given as x,y coordinates
`totalManhattan(arr[])`	Receives a state of a board and returns total Manhattan distance of misplaced tiles. Calls `Manhattan()` for each misplaced tiles
`h3_dist(x1,y1,x2,y2)*`	Calculates a distance between two points considering diagonal moves as well. It finds a distance with higher cost compared to Manhattan distance and provides a better heuristic than h2. First calculate the maximum number of diagonal movements possible to reach the goal square and normal movements later to find a distance.
`totalH3_dist(arr[])*`	Receives a state of a board and returns total h3 distance of misplaced tiles. Calls `h3_dist()` for each misplaced tile.
`UCS(startingState[],goalState[])`	Receives starting state and goal state and solves the problem using Uniform Cost Search algorithm. Prints all related information after goal state is found.
`select_UCS(&frontier, Node *current)`	A function to be used to determine which node in the frontier list to be expanded next. Returns the index of the smallest cost node in the frontier list.
`ILS(startingState[],goalState[], limit)`	Receives starting state and goal state and solves the problem using Iterative Lengthening Search algorithm. Prints all related information after goal state is found.
`ILS_master(startingState`	Higher order function for the `ILS()`.Starting from

`[], goalState[])`	limit 1, it calls the `ILS()` each time by incrementing the limit value by one on each iteration until goal state is reached.
`select_ILS(&frontier, Node *current)`	A function to be used to determine which node in the frontier list to be expanded next. Returns the index of the smallest cost node in the frontier list.
`getMisplacedCount(state[],goalState[])`	Returns number of misplaced tiles.
`h1(&frontier, goalState[], Node *current)`	Return the index of the node to be expanded next on the frontier list, based on the h1 heuristic.
`h2(&frontier, goalState[], Node *current)`	Return the index of the node to be expanded next on the frontier list, based on the h2 heuristic (Manhattan distance).
`h3(&frontier, goalState[], Node *current)*`	Return the index of the node to be expanded next on the frontier list, based on the h3(Bonus part, using `totalH3_dist()` distance function).
`A_star_search(startingState[], goalState[], heuristic)`	Provided starting state, goal state and heuristic of choice. It runs the A* algorithm using the selected heuristic function.

*Bonus part related functions.

h3 Heuristic

We proposed a new h3 heuristic. Unlike Manhattan distance heuristic, h3 first considers all possible diagonal movements and remaining vertical/horizontal movements later. This function provides a higher value estimated cost which still does not exceed the real cost. While this heuristic is still not better than h2 in few random generated inputs in different depths, we observed it is better than the h1 and h2 overall. Especially in larger depth solutions.

2. Outputs

The total number of expanded nodes

d: depth of the solution	UCS	ILS	A*-H1	A*-H2	A*-H3
2	41	48	4	4	4
4	288	296	11	11	10
6	157989	163858	276	52	57
8	47102	49303	360	105	33
10	49349	51992	4584	187	179
12	-	-	1125	261	224
16	-	-	7886	684	628
20	-	-	-	1784	1416
24	-	-	-	13436	5791
28	-	-	-	57609	32429

The maximum number of nodes stored in memory

d: depth of the solution	UCS	ILS	A*-H1	A*-H2	A*-H3
2	207	101	16	16	16
4	1212	616	55	55	44
6	573752	355345	1234	274	300
8	178905	59800	1677	548	427
10	187426	59800	11874	849	597
12	-	-	38961	2854	832
16	-	-	132929	1874	1667
20	-	-	-	8295	6912

24	-	-	-	60661	27681
28	-	-	-	239123	146198

a-)

UCS:
i. The cost of the solution found: 7
ii. The solution path itself : RIGHT -> DOWN_RIGHT -> DOWN_LEFT
iii. The number of expanded nodes: 350

ILS:
i. The cost of the solution found: 7
ii. The solution path itself : RIGHT -> DOWN_RIGHT -> DOWN_LEFT
iii. The number of expanded nodes:354

A* h1:
i. The cost of the solution found:7
ii. The solution path itself : RIGHT -> DOWN_RIGHT -> DOWN_LEFT
iii. The number of expanded nodes: 11

A* h2:
i. The cost of the solution found: 7
ii. The solution path itself : RIGHT -> DOWN_RIGHT -> DOWN_LEFT
iii. The number of expanded nodes: 7

Bonus Part
A* h3:
i. The cost of the solution found: 7
ii. The solution path itself : RIGHT -> DOWN_RIGHT -> DOWN_LEFT
iii. The number of expanded nodes: 7

b-) UCS:
i. The cost of the solution found: - (Execution takes too long)
ii. The solution path itself : - (Execution takes too long)
iii. The number of expanded nodes: - (Execution takes too long)

ILS:
i. The cost of the solution found: - (Execution takes too long)
ii. The solution path itself : - (Execution takes too long)

iii. The number of expanded nodes:- (Execution takes too long)

A* h1:
i. The cost of the solution found:17
ii. The solution path itself : RIGHT -> UP_RIGHT -> UP_RIGHT -> UP_LEFT -> LEFT -> DOWN_LEFT -> DOWN_RIGHT
iii. The number of expanded nodes: 1305

A* h2:
i. The cost of the solution found: 17
ii. The solution path itself : RIGHT -> UP_RIGHT -> UP_RIGHT -> UP_LEFT -> LEFT -> DOWN_LEFT -> DOWN_RIGHT
iii. The number of expanded nodes: 73

Bonus Part
A* h3:
i. The cost of the solution found: 17
ii. The solution path itself : RIGHT -> UP_RIGHT -> UP_RIGHT -> UP_LEFT -> LEFT -> DOWN_LEFT -> DOWN_RIGHT
iii. The number of expanded nodes: 24

c-) UCS:
i. The cost of the solution found: -(Execution takes too long)
ii. The solution path itself : -(Execution takes too long)
iii. The number of expanded nodes: -(Execution takes too long)

ILS:
i. The cost of the solution found: -(Execution takes too long)
ii. The solution path itself : -(Execution takes too long)
iii. The number of expanded nodes:-(Execution takes too long)

A* h1:
i. The cost of the solution found:-(Execution takes too long)
ii. The solution path itself : -(Execution takes too long)
iii. The number of expanded nodes:-(Execution takes too long)

A* h2:
i. The cost of the solution found: 20
ii. The solution path itself : UP -> LEFT -> DOWN -> LEFT -> UP -> UP -> UP -> DOWN_RIGHT -> DOWN -> RIGHT -> DOWN_LEFT -> LEFT -> UP_LEFT -> RIGHT
iii. The number of expanded nodes: 275

Bonus Part
A* h3:
i. The cost of the solution found: 20

ii. The solution path itself : UP -> LEFT -> DOWN -> LEFT -> UP -> UP -> UP -> DOWN_RIGHT -> DOWN -> RIGHT -> DOWN_LEFT -> LEFT -> UP_LEFT -> RIGHT

iii. The number of expanded nodes: 226