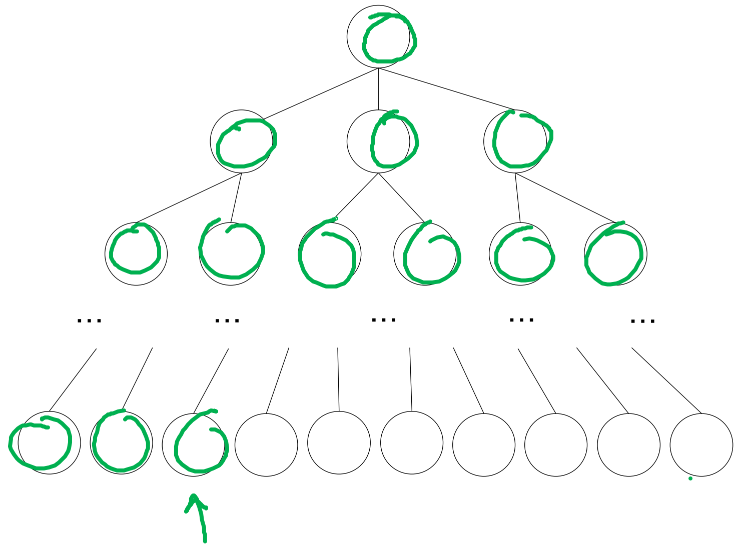
CMSC 170: Problem Solving

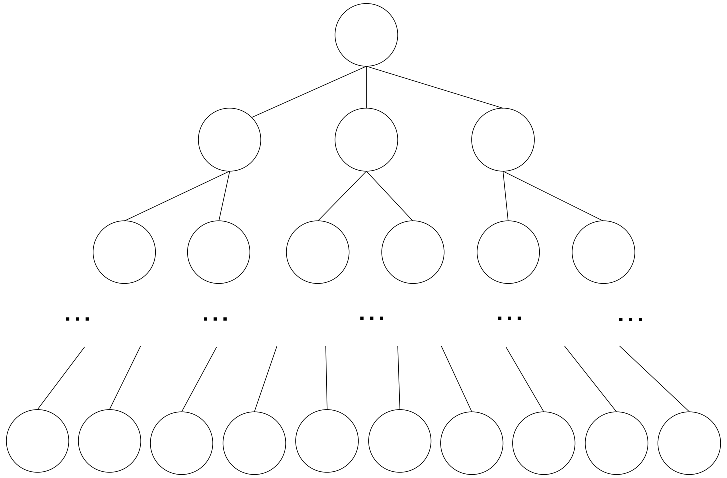
Solving 8-puzzle using A* Search

Miyah Queliste
Institute of Computer Science
University of the Philippines Los Baños

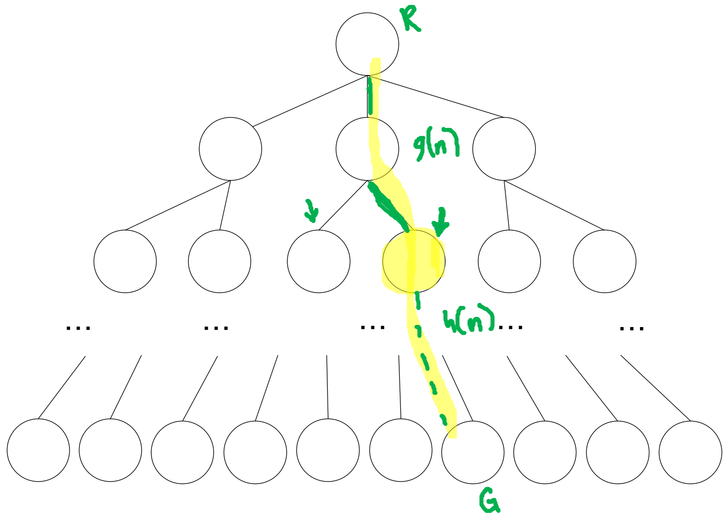
Solving 8-puzzle using A* Search



Solving 8-puzzle using A* Search



Solving 8-puzzle using A* Search



A* score of node n

$$\underline{f(n)} = \underline{g(n)} + \underline{h(n)}$$

$g(n)$: the exact cost of the path from the *starting node* to *node n*

$h(n)$: the heuristic estimated cost from *node n* to the *goal node*

Solving 8-puzzle using A* Search

goal

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	0

current

	0	1	2
0	1	3	6
1	4	2	8
2	0	7	5

	current position	correct position	distance
1 :	(0,0)	(0,0)	0
2 :	(1,1)	(0,1)	1
3 :	(0,1)	(0,2)	1
4 :	(1,0)	(1,0)	0
5 :	(2,2)	(1,1)	2
6 :	(0,2)	(1,2)	1
7 :	(2,1)	(2,0)	1
8 :	(1,2)	(2,1)	2

$h(n) = 8$

Manhattan distance:

$$distance = |x_1 - x_2| + |y_1 - y_2|$$

Solving 8-puzzle using A* Search

```
function AStar {
    openList = {initialState}      #frontier
    closedList = {}                #exploredList
    while(openList is not empty) {
        bestNode = openList.removeMinF();
        closedList.add(bestNode);
        if(GoalTest(bestNode)) return bestNode;
        for(Action a in Actions(currentState)) {
            x = Result(bestNode, a)
            if((x not in openList | closedList) or
                (x in openList and x.g < duplicate.g)):
                x.setParent(bestNode); // can be omitted?
                openList.add(x);
        }
    }
}
```

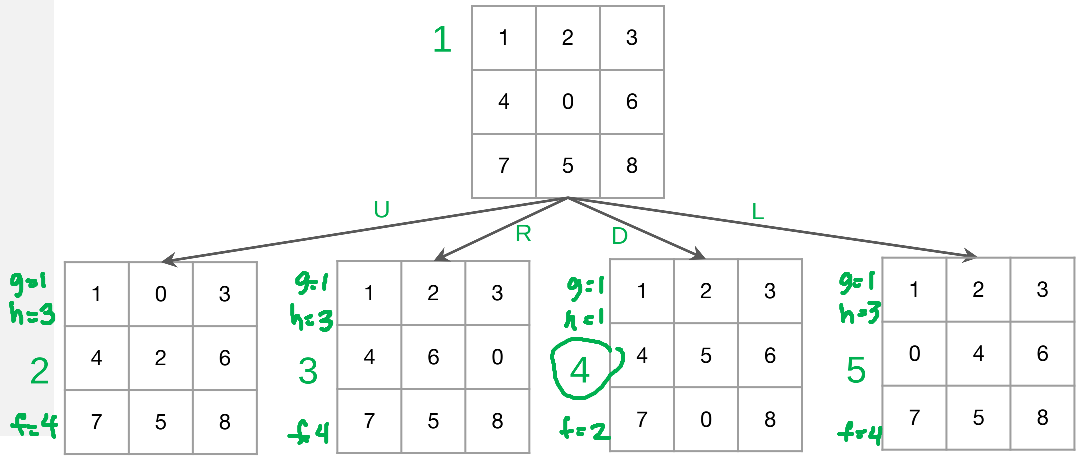
1.

1	2	3
4	0	6
7	5	8

openList	bestNode	closedList
1	1	1

Solving 8-puzzle using A* Search

```
function AStar {
    openList = {initialState}       #frontier
    closedList = {}                 #exploredList
    while(openList is not empty) {
        bestNode = openList.removeMinF();
        closedList.add(bestNode);
        if(GoalTest(bestNode)) return bestNode;
        for(Action a in Actions(currentState)) {
            x = Result(bestNode, a)
            if((x not in openList | closedList) or
                (x in openList and x.g < duplicate.g)):
                x.setParent(bestNode); // can be omitted?
                openList.add(x);
        }
    }
}
```

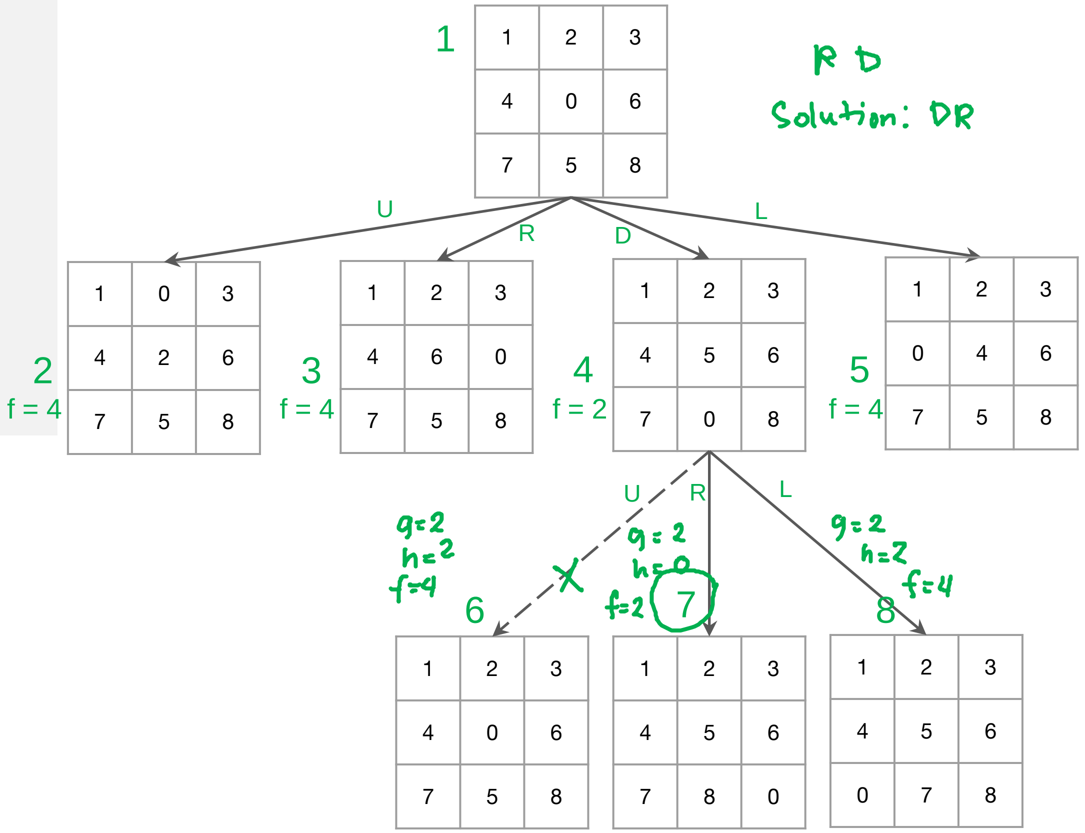


openList	bestNode	closedList
2 3 4 5	~~1~~ 4	1 4

Solving 8-puzzle using A* Search

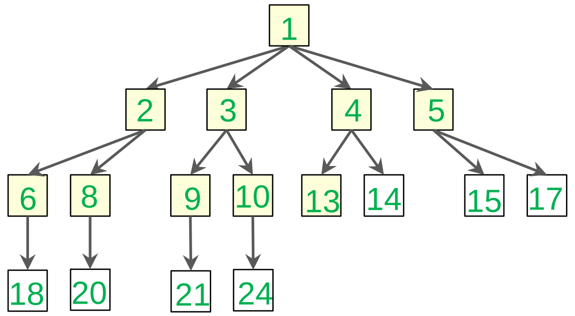
```
function AStar {
     openList = {initialState}      #frontier
     closedList = {}                #exploredList
     while(openList is not empty) {
          bestNode = openList.removeMinF();
          closedList.add(bestNode);
          if(GoalTest(bestNode)) return bestNode;
          for(Action a in Actions(currentState)) {
               x = Result(bestNode, a)
               if((x not in openList | closedList) or
                    (x in openList and x.g < duplicate.g)):
                    x.setParent(bestNode); // can be omitted?
                    openList.add(x);
          }
     }
}
```

openList	bestNode	closedList
2 3 5 7 8	~~4~~ 7	1 4 7

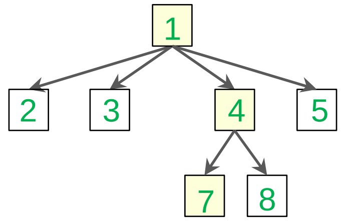


Solving 8-puzzle using A* Search

Explored nodes in BFS (10)

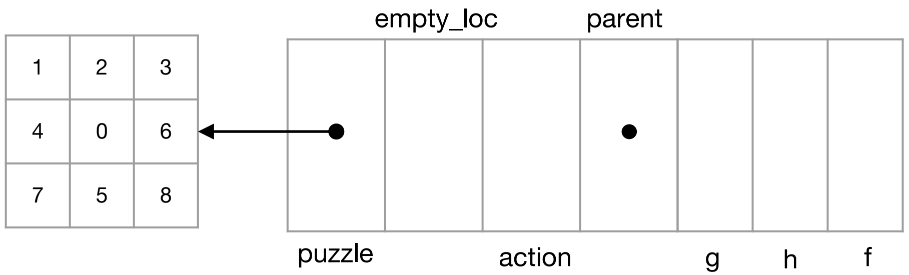


Explored nodes in A* Search (3)



Solving 8-puzzle using Brute Force Search

```
typedef struct state_record {
    int *puzzle;                        // array containing the tile values
    int empty_loc;                      // index of the empty tile
    char action;                        // char action to arrive at this state
    struct state_record *parent;       // a pointer to the parent node
    int g;
    int h;
    int f;
}NODE;
```


1
2
3
4
0
6
7
5
8
empty_loc
parent
puzzle
action
g
h
f