REPORT

Q2) **DFS vs BFS**

Depth first always expands the most recently expanded node if it has any untried successors. If a dead end is reached the algorithm back tracks to the previous node on the current path (LIKE a STACK-LIFO). This is unlike BFS which iteratively searches the node level by level (Like a FIFO).Though both are complete there are few differences:

**Advantages over BFS**

* Usually efficient for searching in trees have a higher branching factor as all the nodes in the given level don’t have to be reached

**Disadvantages over BFS**

* Does not always find the shortest path and can get stuck (i.e. not always optimal)

Q3)

**KEY**

**\_\_ 🡪removed from stack**

**\_\_ 🡪inserted child nodes**

**\_\_ 🡪 conflict resolution/update of backpointers**

PQ

[ ]S0

[S0]A1 B2 C8

[A1] B2 C8 C4 D5

[B2]C4 D5 C4 E3

[E3]C4 D5 G10

[C4]D5 G10 D6 G14 E6

[D5] G10 G7

[G7]

FINAL PATH🡪S-A-D-G :cost=7

|  |  |  |  |
| --- | --- | --- | --- |
| COST TO NODE | | BACK POINTERS | |
| S | 0 | S | NULL |
| A | 1 | A | S |
| B | 2 | B | S |
| C | 8/4/ | C | S/A |
| D |  | D | A |
| E | 3 | E | B |
| G | 10/7/ | G | E/D |

Q4)

Please run **script.m**

You should see the 3 puzzles being solved in order with a slight delay(last one takes some time as explained below)

Detailed pseudo code and Results are discussed below

**PSEUDOCODE**

* Create a pool with S in it and assign cost 0
* Find the successors suc(S,pool,Don’t)
* OPTIMIZATION and Boundary checking. This reduced the iterations for puzzle 3 from 4100 to 2400 iterations
* Check where 9 is
* If it’s the corner then don’t move it outside the corner(e.g. if 9 is in pos 1 then don’t find suc by moving up)
* Check where the suc comes from in the pool
* If suc was got by moving down then don’t move up and get the parent again
* Do the same for down, left and right conditions also
* Find the cost and rearrange based on the cost
* Here cost is 9-no of misplacements +the level/iteration
* Sort based on ascending order
* Determine if the successor goes in the pool or not inornot(v,pool,PQ,bp,cost2,pcost)
* If suc has not been encountered in pool or PQ then add
* If it is in PQ and pool with lesser cost, then add and update the cost and pointers in pool and PQ
* Else don’t add
* Keep removing the first element from PQ
* If PQ was not seen before in pool,push it in with cost and backpointer
* If goal was reached in pool then make PQ=[] and exit from loop
* Now post processing:
* mov(pool,S,G)
* Trace back g in pool to start
* Look at position of 9s and give the appropriate number for left,right,up,down
* eg:

ch=find(B==9)-find(A==9);

switch(ch)

case 3;

P(i)=1;%moved blank to left

case 1 ;

P(i)=3;%moved down

case -1 ;

P(i)=4;% moved up

case -3 ;

P(i)=2;%moved right

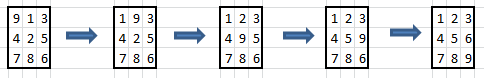
End

* Send the moves to GUI

a) 4 moves

P1 =[ 1 3 1 3]

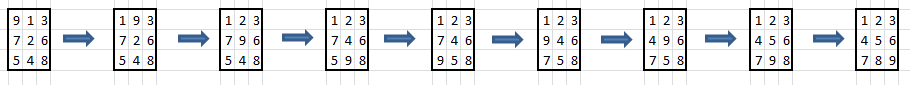
**4 iterations**



b) 8 moves

P2 =[ 1 3 3 2 4 1 3 1]

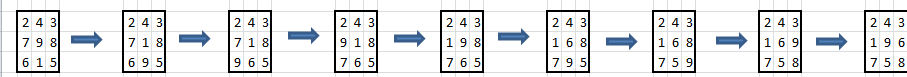
**52 iterations**

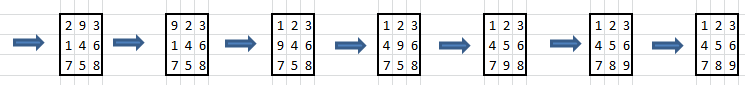


c) 14 moves

P3=[3 2 4 1 3 1 4 2 4 2 3 1 3 1]

**4000+ iterations🡪after optimization takes ~2400 iterations**(please wait for ~45mins)





Q1)