BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Balla Mahadev Shrikrishna
UID no.	2023300010
Experiment No.	9

AIM:	To implement a branch and bound algorithm.
Program 1	
PROBLEM STATEMENT:	The 15 puzzle problem consists of 15 numbered tiles on a square frame with a capacity of 16 tiles. An initial arrangement of the tiles is given. The objective is to transform this arrangement into the goal arrangement as shown below through a series of legal moves. We can carry out the search of path initial arrangement to final arrangement by exploring tree organization in branch and bound method. The children of each node x in the tree represents the states reachable from state x by one legal move. **Input –** Initial state of 15 Puzzle problem **Output –** The entire path of tree organization to reach the final/goal state. **Submission –** 1) C/C++ source code of implementation. 2) Verified output for the written source code with multiple inputs. 3) One page report of Exp. 8
ALGORITHM:	**Step 1: Initialization** 1) Input: Initial board configuration. 2) Check Solvability: Calculate inversion count (number of tile pairs out of order). a) Puzzle is solvable iff: For a 4×4 grid - (inversions % 2 == 0) XOR (blank_row_from_bottom % 2 == 1) 3) Initialize Priority Queue with the starting state. **Step 2: Branch and Bound Loop** While the priority queue is not empty: 1) Extract State with the lowest f(n). 2) Check Goal: If the board matches the goal state, return the move sequence. 3) Generate Successors:

	For each valid move (Up, Down, Left, Right): a) Swap the blank tile with the adjacent tile. b) Compute g(n) = g(current) + 1. c) Compute h(n) (Manhattan Distance for the new board). d) Add the new state to the priority queue if not visited. 4) Prune Redundant States: Use a hash table to track visited states and avoid cycles. **Step 3: Termination** If the queue is exhausted, conclude the puzzle is unsolvable.
PROGRAM:	Refer puzzle.c file

RESULT:

```
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp9> gcc puzzle.c
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp9> ./a.exe
Enter the 15-puzzle configuration (use 0 for empty space):
Enter numbers from 0 to 15, row by row (4 numbers per row):
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0

Solving the following 15-puzzle:
 1   2   3   4
 5   6   7   8
 9  10  11  12
13  14  15   _

Solution found in 0 moves:

PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp9> ./a.exe
Enter the 15-puzzle configuration (use 0 for empty space):
Enter numbers from 0 to 15, row by row (4 numbers per row):
1 2 3 4
5 6 7 8
9 10 11 12
13 14 0 15

Solving the following 15-puzzle:
 1   2   3   4
 5   6   7   8
 9  10  11  12
13  14   _  15

Solution found in 1 moves:
R
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp9> ./a.exe
Enter the 15-puzzle configuration (use 0 for empty space):
Enter numbers from 0 to 15, row by row (4 numbers per row):
1 2 3 4
5 6 7 8
9 10 0 12
13 14 11 15

Solving the following 15-puzzle:
 1   2   3   4
 5   6   7   8
 9  10   _  12
13  14  11  15

Solution found in 2 moves:
D R
```

```
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp9> ./a.exe
Enter the 15-puzzle configuration (use 0 for empty space):
Enter numbers from 0 to 15, row by row (4 numbers per row):
1 2 3 4
5 6 7 8
9 10 11 12
13 15 14 0

Solving the following 15-puzzle:
 1   2   3   4
 5   6   7   8
 9  10  11  12
13  15  14   _

This puzzle configuration is not solvable.
PS C:\Mahadev\SE\Sem4\DAA\Lab\Lab Sessions\exp9>
```

CONCLUSION:

Name: Balla Mahadev Shrikrishna
UID: 2023300010
Div.: A
Batch: A

Exp-9

* The 15-Puzzle is a classic ~~slide tile~~ sliding tile puzzle consisting of a 4x4 grid with 15 numbered tiles & one empty space.

* Algo. Used: Branch & Bound with Manhattan Distance Heuristic.
  - Branch & Bound is a state space search algo. that explores the most promising nodes first.
  - Heuristic func^n (Manhattan Distance): The sum of distances of each tile from its goal pos^n (row wise + col wise).
    $f(n) = g(n) + h(n)$ where $g(n)$ = no. of moves taken so far.
    $h(n)$ = Manhattan dist. heuristic

* Implementat^n
  - Priority Queue: Used to explore states with lowest $f(n)$ first.
  - Solvability Check: Ensures the puzzle is solvable before attempting to solve it.
  - Move Generat^n: Generates valid moves while avoiding revisiting states.
  - Terminat^n: Stops when the goal state is reached or the queue is exhausted.

* Time Complexity : $O(b^d)$, where $b \rightarrow$ branching factor (≤ 4, $\because$ max 4 moves per state)
  $d \rightarrow$ depth of the optimal sol^n
* Space Complexity : $O(b^d)$, due to storage of states in the priority queue.
* While it guarantees optimal sol^{n}s, its performance degrades for harder puzzles due to exponential complexity.