SOLUTION TO THE 15 -PUZZLE PROBLEM

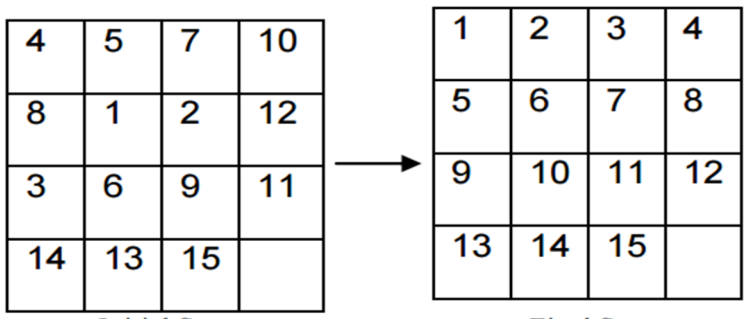
A general paper on the 15-puzzle problem with possible solutions

Presented by Poojita Suri and Neelanja Chaturvedi

All data presented has been borrowed from various sources on the internet, and the references have been mentioned accordingly.

1.0 INTRODUCTION TO THE PROBLEM

The 15 puzzle problem consists of 15 squares numbered from 1 to 15 that are placed in a box leaving one position out of the 16 empty. The goal is to reposition the squares from a given arbitrary starting arrangement by sliding them one at a time into the final configuration. One of the instances of the initial and the final state are depicted in the figure below:



Initial State Final State

Sam Loyd was the man who invented the 14-15 or Boss puzzle. This was a version where the starting position was very similar to the home position except that the 14 and 15 were inverted. In the early 1870's he offered a $1000 reward to anyone who could solve it, which set off "fifteen fever". However no-one claimed the prize for the simple reason that it is not solvable! The reason shall dealt with later.

1.1 SOLVABILITY

Johnson & Story (1879) used a parity argument to show that half of the starting positions for the *15*-puzzle are impossible to resolve, no matter how many moves are made. This is done by considering a function of the tile configuration that is invariant under any valid move, and then using this to partition the space of all possible labeled states into twoequivalence classes of reachable and unreachable states.

In 15 – Puzzle problem, there are total of 16 tiles which contains distinct numbers and a blank space. These 16 tiles can result in 16! Initial configurations. Out of these much configurations only half of the configurations are solvable and others are not. Thus only 16! / 2 initial configurations can lead to goal configuration using limited number of moves. Given an initial configuration, it can be checked whether the configuration is solvable or not.

The steps for determining solvability are as follows:

Step 1: Shift the blank tile at the bottom right corner of the grid. This can be easily done.

Step 2: Calculate Permutation Inversion for each tile. An inversion is when a tile precedes another tile with a lower number on it [1].

For example consider a configuration shown in Figure 6. Consider tile 12, there are 11 tiles numbered 1 – 11, with smaller number than 12, that appear after 12. Thus inversion number for tile 12 is 11.

Consider tile 14, there are 6 tiles numbered 5, 9, 3, 8, 13 and 6, with smaller number than 14, that appear after 14. Thus, inversion number for tile 14 is 6.

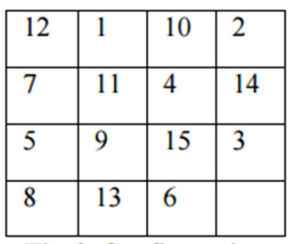


Fig 6: Configuration

Inversion number of the tiles is as follows:

Tile 12 → 11 inversions
Tile 1 → 0 inversions
Tile 10 → 8 inversions
Tile 2 → 0 inversions
Tile 7 → 4 inversions
Tile 11 → 6 inversions
Tile 4 → 1 inversions
Tile 14 → 6 inversions
Tile 5 → 1 inversions
Tile 9 → 3 inversions
Tile 15 → 4 inversions
Tile 3 → 0 inversions
Tile 8 → 1 inversions
Tile 13 → 1 inversions
Tile 6 → 0 inversions

Step 3: Calculate the sum of inversions for all the tiles.

In the above example, sum = 46.

Rule: Odd permutation inversions of the puzzle are impossible to solve [2], all even permutations are solvable [3]. Archer also presented a simple proof of above rules [4].

Thus, the above configuration is solvable as the permutation inversion is even.

Consider another configuration as shown in Figure 7.

Tile 13 → 12 inversions
Tile 10 → 9 inversions
Tile 11 → 9 inversions
Tile 6 → 5 inversions
Tile 5 → 4 inversions
Tile 7 → 4 inversions
Tile 4 → 3 inversions
Tile 8 → 3 inversions
Tile 1 → 0 inversions
Tile 12 → 3 inversions
Tile 14 → 3 inversions
Tile 9 → 3 inversions
Tile 3 → 1 inversions
Tile 15 → 1 inversions
Tile 2 → 0 inversions

Sum = 59.

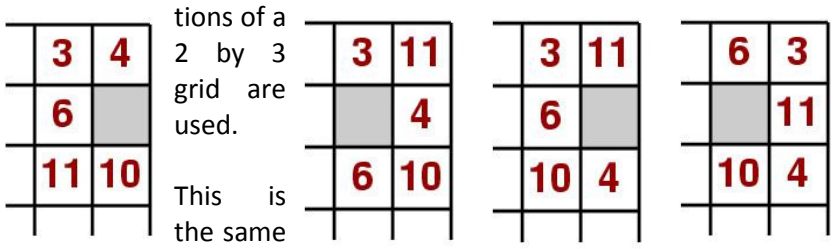
Since, the number is odd, the above arrangement of the puzzle cannot be solved.

1.3 FINDING SOLUTIONS

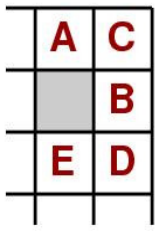
After having worked out whether a given puzzle is solvable or not, we try to find the possible solutions for this problem. There are two ways this problem can be solved [4]:

1.3.1 SIMPLE SOLUTION

One logical process to follow is to get the tiles in place in order, i.e. by getting the 1 into place, then the 2,3,4 etc. However one may quickly find that by placing the 3 in place, the 4 cannot fit without disturbing the 3 again. From the work before, it is known that a permutation of (11 4) on the top right-hand 2 by 2 grid is not legal by sliding so instead permutations of a 2 by 3 grid are used.



This is the same as a permutation of (4 10 6 *)(6 3 11 *) (11 * 4) (3 6 11 10 * 4)= (11 6 * 4). In a similar way, the 7 and 8 can be placed in the correct positions. The bottom two lines will have similar problems; the 9 and 13 need to be in the right order then cycled round to their positions, and finally one is left with another 3 by 2 grid with the 10 and 14 to be placed first like the 3 and 4, then the last three rotated to get home! Of course it makes sense to think about the other numbers and where they should eventually end up, and so move them in those particular directions.

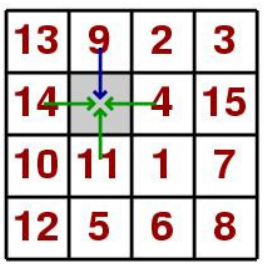


For the general case if a tile, B needs to get into a upper right corner (either of the grid, or previously positioned tiles) in the order A B at the top, the cycle of moves from this position is thus: (A C B D E *)(D B C E *)(B A E C*) in terms of the legal slides.

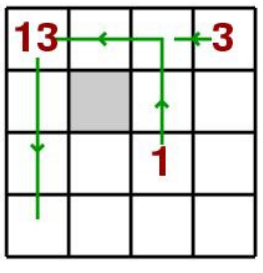
These motions can then be used for any part of the problem in a corner, just mirror the permutation to fit the corner.

1.3.2 FASTEST SOLUTION

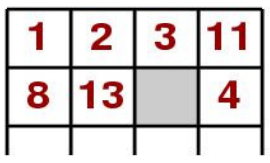
Obviously trying to solve the problem like this is not the fastest (i.e. the one with the least number of moves) as one tends to move some tiles in the wrong direction unnecessarily as the first ones are concentrated on individually. As one can see in arranging the corner piece, it is sometimes necessary to go backwards to be able to go forwards, but these motions ought to be limited where possible

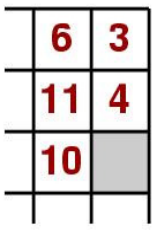


A computer could be used to evaluate all possible solutions and then show the quickest one. It would look at all possible moves at each node- where there is a branching off of possible solutions. For example in the previous problem (shown again to the left), the starting position has a node of different 4 paths or branches; the 4, 9, 11 or 14 could be moved. If the 9 is chosen to be moved, the new node would have 3 branches, but the 9 would be discounted as it leads back to the original position. One way of getting the computer to perform the search is to make it evaluate possibilities at each node, to pick one path and follow it until either the home state is reached or a dead end. If a dead end is reached, the computer goes back to the last node and takes a different path. However, one cannot be sure that this is the fastest, as it discards all other possibilities once it has found a solution. Instead the computer could examine all nodes that are one permutation from the starting position. If none of these are the solution they are expanded to nodes 2 single permutations away from the starting state, etc. until the home state is found, so this will be the shortest. This however is very complicated and certainly not something a human could do easily, so instead the problem of whether a person can solve the fifteen puzzle in the shortest solution will be examined.



One thing a human can do is to work out a heuristic or best fit- an estimate of the number of moves a tile would take to get from a particular position on the grid to the home state. The heuristic really wants to be either optimistic or correct as it will show the minimum number of moves expected to get home. It is worked out by looking at the position of each tile in the starting state and calculating the number of moves it would take to get to its home place if the space were adjacent to it, in the direction of home. In the example, the 1 would need 4 moves to get home (so the heuristic is 4), the 13 would need 3, the 3, 1 move. It is known that the heuristic calculated in this way is optimistic or accurate, because if the empty space were to the left of its position now, the 13 could get to its home place in the heuristic's estimated number of moves. However as it is not, the heuristic would be larger as the tile cannot be simply lifted up. The heuristic of a particular individual problem is just the sum of all the heuristics.





It would seem sensible to work out the total heuristic for a particular problem and then follow the path of best fit where the value of the heuristic decreases. For the example taken, the heuristic at the start would be 4 + 1 + 1 +...+ 3 + 3 + 3 = 36 for the tiles in chronological order. To decrease the heuristic, the 9 or 14 could be moved, as they are the only two moves that move the tile closer to its home position. Moving one tile in its home direction decreases the value of the heuristic by one. A person can keep track of the moves made and where the corresponding best fit nodes are. When a dead end is reached, the person should go back to the last node and examine the other branches, until a solution is found. However in practice this does not work out. It was seen from the process of moving corner pieces into position that it is necessary to move tiles in what seems the wrong direction in order to solve the problem.

The most common heuristics used to solve the 15-puzzle problem are
1. Branch and bound algorithm
2. A* algorithm

The general algorithm followed is [6]:

1. Get the first state, which is your start state.
2. Get all the possible states in which puzzle can be.
3. Add these new states in open state list
4. Add processed sate in the closed list
5. Pick the next state which has the lowest cost in the open list
6. Start repeating the steps from 1 to 6 unless you are into the final state, or no more states to examine (in this case, no solution exists).
7. Once you have the final state, backtrack to the start node and that would be the optimal path to find the solution.

1.4 CONCLUSION

In conclusion, a human can combine both ways of solving the problem; through simple logical solving and by trying to move the tiles in the direction of best fit. In this way a solution will be found that will be reasonably short, and probably shorter than any solution the person would have found by trial and error. Obviously it does not take into account the time needed to work out the heuristic at the start. Hopefully, understanding the principles used in the fifteen puzzle would help a person understand the problem and to find the best way to go about achieving a solution, whether they want a solution that is fast in time or number of moves, or just to solve it for fun.

1.5 IMPLEMENTATION

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int Game[4][4];
void Reset();
void Left();
void Right();
void Up();
void Down();
void Print();
int  Check();
void Logo();
void main()

{
int counter=0;
 char character='a';
```

```
clrscr();

Reset();
Logo();
Print();
while(character!='q')
{
 character=getch();
 switch(character)
 {
 case 80:
 Down();
 Print();
 Check();
 break;

 case 77:
 Right();
 Print();
 Check();
 break;

 case 75:
 Left();
 Print();
 Check();
 break;

 case 72:
 Up();
 Print();
 Check();
 break;

 case'r':
 Reset();
 Print();
 Check();
 break;
 }
 }
}
void Logo()
{
 clrscr();


printf("\n==================================================
=========================");
 printf("\n              15 PUZZLE GAME");

printf("\n==================================================
=========================");

 printf("\n\n\n\n\n\n\t\t\t Keys: \n\n\n\t\t\t\t Up");
 printf("\n\n\t\t\t\t Down");
 printf("\n\n\t\t\t\t Left");
 printf("\n\n\t\t\t\t Right\n\n\n\n");


 printf("\n\n\n\n\t\t              'q' to Quit.");
 printf("\n\n\n\n\n\n\n\t\t\tPress Any Key To Continue");

getch();
}
void Print()
{      clrscr();

 printf("\n\n\n\n\n\n\n\n\n\n\n\n\n");
 printf("\n\t\t\t_____________________");
 printf("\n\t\t\t|    |    |    |");
 printf("\n\t\t\t|    |    |    |");
 gotoxy(27,16);
 {
  printf("%d",Game[0][0]);
 }
 gotoxy(32,16);
{
 printf("%d",Game[0][1]);
}
gotoxy(37,16);
{
 printf("%d",Game[0][2]);
}
gotoxy(42,16);
{
 printf("%d",Game[0][3]);
}
printf("\n\t\t\t|    |    |    |");
printf("\n\t\t\t---------------------");
printf("\n\t\t\t|    |    |    |");
printf("\n\t\t\t|    |    |    |");
gotoxy(27,20);
{
 printf("%d",Game[1][0]);
}
gotoxy(32,20);
{
 printf("%d",Game[1][1]);
}
gotoxy(37,20);
{
 printf("%d",Game[1][2]);
}
gotoxy(42,20);
{
 printf("%d",Game[1][3]);
}
printf("\n\t\t\t|    |    |    |");
printf("\n\t\t\t---------------------");
printf("\n\t\t\t|    |    |    |");
printf("\n\t\t\t|    |    |    |");
gotoxy(27,24);
{
 printf("%d",Game[2][0]);
} gotoxy(32,24);
{
 printf("%d",Game[2][1]);
}
gotoxy(37,24);
{
 printf("%d",Game[2][2]);
}
gotoxy(42,24);
{
 printf("%d",Game[2][3]);
}
printf("\n\t\t\t|    |    |    |");
printf("\n\t\t\t---------------------");
printf("\n\t\t\t|    |    |    |");
printf("\n\t\t\t|    |    |    |");
gotoxy(27,28);
{
 printf("%d",Game[3][0]);
}
gotoxy(32,28);
{
 printf("%d",Game[3][1]);
}
gotoxy(37,28);
{
 printf("%d",Game[3][2]);
}
gotoxy(42,28);
{
 printf("%d",Game[3][3]);
}
printf("\n\t\t\t|    |    |    |");
printf("\n\t\t\t~~~~~~~~~~~~~~~~~~~~~");

}
void Reset()
{
 int reset=15,i,j;
```

```
 for ( i=0;i<4;i++)
 {
  for ( j=0;j<4;j++)
  {
   Game[i][j]=reset--;
  }
 }
 Game[3][3]=0;
}
int Check()
{

    int count=0,i,j,counter;
    int check=0;
    if(check!=0)
    {
    counter++;
    }

 for ( i=0;i<4;i++)
 {
  for ( j=0;j<4;j++)
  {
   count++;
   if(i==3 && j==3)
   {
    break;
   }
   if( Game[i][j]==count)
   {
    check++;
   }

  }
 }
 if(check==15)
 {
  gotoxy(32,36);
  printf("Congratulations");
  return 1;

 }
 else return 0;
}

void Left()
{
 int x,y,i,j;

 for ( i=0;i<4;i++)
 {
  for ( j=0;j<4;j++)
  {
   if (Game[i][j]==0)
   {
    x=i;
    y=j;
   }
  }
 }
 if(y!=3)
 {
  Game[x][y]=Game[x][y+1];
  Game[x][y+1]=0;
 }
}

void Right()
{
 int x,y,i,j;

 for ( i=0;i<4;i++)
 {
  for ( j=0;j<4;j++)
  {
   if (Game[i][j]==0)
```

```
   {
    x=i;
    y=j;
   }
  }
 }
 if(y!=0)
 {
  Game[x][y]=Game[x][y-1];
  Game[x][y-1]=0;
 }
}
void Up()
{
 int x,y,i,j;

 for ( i=0;i<4;i++)
 {
  for ( j=0;j<4;j++)
  {
   if (Game[i][j]==0)
   {
    x=i;
    y=j;
   }
  }
 }
 if(x!=3)
 {
  Game[x][y]=Game[x+1][y];
  Game[x+1][y]=0;
 }


}
```

REFERENCES

[1]. http://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html

[2]. Johnson, W. W. (1879). Notes on the”15” Puzzle. American Journal of Mathematics 2(4):397–404.

[3]. Story, W. E. (1879) "Notes on the '15 Puzzle. II.' “, American Journal of Mathematics 2, 399-404. [17]

[4]. Archer, Aaron F. (1999), "A modern treatment of the 15 puzzle", The American Mathematical Monthly 106 (9): 793–799.

[5]. http://www.math.ubc.ca/~cass/courses/m308-02b/projects/grant/fifteen.html

[6]. http://www.codeproject.com/Articles/616874/Puzzle-using-A-A-Star-Algorithm-Csharp