#2057672 Solution for ALDS1_13_B: 8 Puzzle by naoto172

Source Code Status Test Cases

Policy: public Reviewed: 15

00.00 sec 2760 KB 434 lines 9528 bytes 2016-11-03 20:38

```
#include <stdio.h>
#include <math.h>
#include <algorithm>

using namespace std;

int minimum = -1;


int calcDiffSum(int table[3][3]){
    int sum = 0;
    for(int row=0;row<3;row++){
        for(int col=0;col<3;col++){
            if(table[row][col] != 0){
                sum += abs((table[row][col]-1)/3-row) + abs((table[row][col]-1)%3-col);
            }
        }
    }
    return sum;
}

int zeroLoc(int table[3][3]){
    int zero_loc;
    for(int i=0;i<3;i++){
        for(int k=0;k<3;k++){
            if(table[i][k]==0){
                zero_loc=3*i+k;
            }
        }
    }
    return zero_loc;
}

void recursive(int table[3][3],int max_depth,int count,int pre_loc,int zero_loc){
    int diffSum = calcDiffSum(table);
    if(count >= max_depth && diffSum == 0){
        minimum = max_depth;
        return;
    }

    if(minimum != -1)return;

    int next_table[4][3][3],tmp_row,tmp_col,pre_row,pre_col;
    int dif1,dif2,dif3,dif4;

    switch(zero_loc){
    case 0:
        for(int a=0;a<2;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 1){
            if(table[0][1]-1)%3==0){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[0][0][0],next_table[0][0][1]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[0],max_depth,count+1,zero_loc,1);
            }
        }
        if(pre_loc != 10){
            if(table[1][0]-1)/3==0){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[1][0][0],next_table[1][1][0]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,10);
            }
        }
        break;
    case 1:

        for(int a=0;a<3;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 0){
            if(table[0][0]-1)%3==1){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[0][0][0],next_table[0][0][0]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[0],max_depth,count+1,zero_loc,0);
            }
        }
        if(pre_loc != 11){
            if(table[1][1]-1)/3==0){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[1][0][1],next_table[1][1][1]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,11);
            }
        }
        if(pre_loc != 2){
            if(table[0][2]-1)%3==1){
                dif3 = diffSum - 1;
            }else{
                dif3 = diffSum + 1;
            }
            swap(next_table[2][0][1],next_table[2][0][2]);
            if(count < max_depth && dif3 <= max_depth-count){
                recursive(next_table[2],max_depth,count+1,zero_loc,2);
            }
        }
        break;
    case 2:
        for(int a=0;a<2;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 1){
            if(table[0][1]-1)%3==2){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[0][0][2],next_table[0][0][1]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[0],max_depth,count+1,zero_loc,1);
            }
        }
        if(pre_loc != 12){
            if(table[1][2]-1)/3==0){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[1][0][2],next_table[1][1][2]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,12);
            }
        }
        break;
    case 10:
        for(int a=0;a<3;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 0){
            if(table[0][0]-1)%3==1){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[0][1][0],next_table[0][0][0]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[0],max_depth,count+1,zero_loc,0);
            }
        }
        if(pre_loc != 11){
            if(table[1][1]-1)/3==0){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[1][1][0],next_table[1][1][1]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,11);
            }
        }
        if(pre_loc != 20){
            if(table[2][1]-1)%3==1){
                dif3 = diffSum - 1;
            }else{
                dif3 = diffSum + 1;
            }
            swap(next_table[2][1][0],next_table[2][2][0]);
            if(count < max_depth && dif3 <= max_depth-count){
                recursive(next_table[2],max_depth,count+1,zero_loc,20);
            }
        }
        break;
    case 11:
        for(int a=0;a<3;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 1){
            if(table[0][1]-1)%3==1){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[0][1][1],next_table[0][0][1]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[0],max_depth,count+1,zero_loc,1);
            }
        }
        if(pre_loc != 10){
            if(table[1][0]-1)%3==2){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[1][1][0],next_table[1][1][1]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,10);
            }
        }
        if(pre_loc != 12){
            if(table[1][2]-1)%3==2){
                dif3 = diffSum - 1;
            }else{
                dif3 = diffSum + 1;
            }
            swap(next_table[1][1][2],next_table[1][1][1]);
            if(count < max_depth && dif3 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,12);
            }
        }
        if(pre_loc != 21){
            if(table[2][1]-1)%3==1){
                dif4 = diffSum - 1;
            }else{
                dif4 = diffSum + 1;
            }
            swap(next_table[2][1][1],next_table[2][2][1]);
            if(count < max_depth && dif4 <= max_depth-count){
                recursive(next_table[2],max_depth,count+1,zero_loc,21);
            }
        }
        break;
    case 12:
        for(int a=0;a<3;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 2){
            if(table[0][2]-1)%3==1){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[0][1][2],next_table[0][0][2]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[0],max_depth,count+1,zero_loc,2);
            }
        }
        if(pre_loc != 11){
            if(table[1][2]-1)%3==2){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[1][1][2],next_table[1][1][1]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,11);
            }
        }
        if(pre_loc != 22){
            if(table[2][2]-1)%3==1){
                dif3 = diffSum - 1;
            }else{
                dif3 = diffSum + 1;
            }
            swap(next_table[2][1][2],next_table[2][2][2]);
            if(count < max_depth && dif3 <= max_depth-count){
                recursive(next_table[2],max_depth,count+1,zero_loc,22);
            }
        }
        break;
    case 20:

        for(int a=0;a<2;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 10){
            if(table[1][0]-1)%3==2){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[1][0][0],next_table[1][1][0]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,10);
            }
        }
        if(pre_loc != 21){
            if(table[2][0]-1)%3==1){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[2][0][0],next_table[2][1][0]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[2],max_depth,count+1,zero_loc,21);
            }
        }
        break;
    case 21:
        for(int a=0;a<3;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 20){
            if(table[2][0]-1)%3==1){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[0][2][0],next_table[0][1][0]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[0],max_depth,count+1,zero_loc,20);
            }
        }
        if(pre_loc != 11){
            if(table[1][1]-1)%3==2){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[1][2][1],next_table[1][1][1]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,11);
            }
        }
        if(pre_loc != 22){
            if(table[2][2]-1)%3==1){
                dif3 = diffSum - 1;
            }else{
                dif3 = diffSum + 1;
            }
            swap(next_table[2][2][1],next_table[2][1][1]);
            if(count < max_depth && dif3 <= max_depth-count){
                recursive(next_table[2],max_depth,count+1,zero_loc,22);
            }
        }
        break;
    case 22:
        for(int a=0;a<2;a++){
            for(int b=0;b<3;b++){
                for(int c=0;c<3;c++){
                    next_table[a][b][c] = table[b][c];
                }
            }
        }

        if(pre_loc != 21){
            if(table[2][1]-1)%3==2){
                dif1 = diffSum - 1;
            }else{
                dif1 = diffSum + 1;
            }
            swap(next_table[0][2][1],next_table[0][2][2]);
            if(count < max_depth && dif1 <= max_depth-count){
                recursive(next_table[0],max_depth,count+1,zero_loc,21);
            }
        }
        if(pre_loc != 12){
            if(table[1][2]-1)%3==2){
                dif2 = diffSum - 1;
            }else{
                dif2 = diffSum + 1;
            }
            swap(next_table[1][2][1],next_table[1][2][2]);
            if(count < max_depth && dif2 <= max_depth-count){
                recursive(next_table[1],max_depth,count+1,zero_loc,12);
            }
        }
        break;
    }
}

void func(int table[3][3],int max_depth){
    while(minimum == -1){
        recursive(table,max_depth,0,-1,zeroLoc(table));
        max_depth += 2;
    }
}


int main(){

    int table[3][3],start_count;

    for(int i=0;i<3;i++){
        scanf("%d %d %d",&table[i][0],&table[i][1],&table[i][2]);
    }
    start_count = calcDiffSum(table);

    func(table,start_count);

    printf("%d\n",minimum);

    return 0;
}
```