

#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
1 #include <stdio.h>
2 #include <math.h>
3 #include <algorithm>
4
5 using namespace std;
6
7 int minimum = -1;
8
9
10 int calcDiffSum(int table[3][3]){
11     int sum = 0;
12     for(int row=0;row<3;row++){
13         for(int col=0;col<3;col++){
14             if(table[row][col] != 0){
15                 sum += abs((table[row][col]-1)/3-row) + abs((table[row][col]-1)%3-col);
16             }
17         }
18     }
19     return sum;
20 }
21
22 int zeroLoc(int table[3][3]){
23     int zero_loc;
24     for(int i=0;i<3;i++){
25         for(int k=0;k<3;k++){
26             if(table[i][k]==0){
27                 zero_loc=i*3+k;
28             }
29         }
30     }
31     return zero_loc;
32 }
33
34 void recursive(int table[3][3],int max_depth,int count,int pre_loc,int zero_loc){
35     int diffSum = calcDiffSum(table);
36     if(count == max_depth && diffSum == 0){
37         minimum = max_depth;
38         return;
39     }
40
41     if(minimum != -1)return;
42
43     int next_table[4][3][3],tmp_row,tmp_col,pre_row,pre_col;
44     int dif1,dif2,dif3,dif4;
45
46     switch(zero_loc){
47     case 0:
48         for(int a=0;a<2;a++){
49             for(int b=0;b<3;b++){
50                 for(int c=0;c<3;c++){
51                     next_table[a][b][c] = table[b][c];
52                 }
53             }
54         }
55
56         if(pre_loc != 1){
57             if((table[0][1]-1)%3==0){
58                 dif1 = diffSum - 1;
59             }else{
60                 dif1 = diffSum + 1;
61             }
62             swap(next_table[0][0][0],next_table[0][0][1]);
63             if(count < max_depth && dif1 <= max_depth-count){
64                 recursive(next_table[0],max_depth,count+1,zero_loc,1);
65             }
66         }
67
68         if(pre_loc != 10){
69             if((table[1][0]-1)/3==0){
70                 dif2 = diffSum - 1;
71             }else{
72                 dif2 = diffSum + 1;
73             }
74             swap(next_table[1][0][0],next_table[1][1][0]);
75             if(count < max_depth && dif2 <= max_depth-count){
76                 recursive(next_table[1],max_depth,count+1,zero_loc,10);
77             }
78         }
79         break;
80     case 1:
81         for(int a=0;a<3;a++){
82             for(int b=0;b<3;b++){
83                 for(int c=0;c<3;c++){
84                     next_table[a][b][c] = table[b][c];
85                 }
86             }
87         }
88
89         if(pre_loc != 0){
90             if((table[0][0]-1)%3>=1){
91                 dif1 = diffSum - 1;
92             }else{
93                 dif1 = diffSum + 1;
94             }
95             swap(next_table[0][0][0],next_table[0][0][0]);
96             if(count < max_depth && dif1 <= max_depth-count){
97                 recursive(next_table[0],max_depth,count+1,zero_loc,0);
98             }
99         }
100
101         if(pre_loc != 11){
102             if((table[1][1]-1)/3==0){
103                 dif2 = diffSum - 1;
104             }else{
105                 dif2 = diffSum + 1;
106             }
107             swap(next_table[1][0][1],next_table[1][1][1]);
108             if(count < max_depth && dif2 <= max_depth-count){
109                 recursive(next_table[1],max_depth,count+1,zero_loc,11);
110             }
111         }
112
113         if(pre_loc != 2){
114             if((table[0][2]-1)%3<=1){
115                 dif3 = diffSum - 1;
116             }else{
117                 dif3 = diffSum + 1;
118             }
119             swap(next_table[2][0][1],next_table[2][0][2]);
120             if(count < max_depth && dif3 <= max_depth-count){
121                 recursive(next_table[2],max_depth,count+1,zero_loc,2);
122             }
123         }
124         break;
125     case 2:
126         for(int a=0;a<2;a++){
127             for(int b=0;b<3;b++){
128                 for(int c=0;c<3;c++){
129                     next_table[a][b][c] = table[b][c];
130                 }
131             }
132         }
133
134         if(pre_loc != 1){
135             if((table[0][1]-1)%3==2){
136                 dif1 = diffSum - 1;
137             }else{
138                 dif1 = diffSum + 1;
139             }
140             swap(next_table[0][0][2],next_table[0][0][1]);
141             if(count < max_depth && dif1 <= max_depth-count){
142                 recursive(next_table[0],max_depth,count+1,zero_loc,1);
143             }
144         }
145
146         if(pre_loc != 12){
147             if((table[1][2]-1)/3==0){
148                 dif2 = diffSum - 1;
149             }else{
150                 dif2 = diffSum + 1;
151             }
152             swap(next_table[1][0][2],next_table[1][1][2]);
153             if(count < max_depth && dif2 <= max_depth-count){
154                 recursive(next_table[1],max_depth,count+1,zero_loc,12);
155             }
156         }
157         break;
158     case 10:
159         for(int a=0;a<3;a++){
160             for(int b=0;b<3;b++){
161                 for(int c=0;c<3;c++){
162                     next_table[a][b][c] = table[b][c];
163                 }
164             }
165         }
166
167         if(pre_loc != 0){
168             if((table[0][0]-1)/3>=1){
169                 dif1 = diffSum - 1;
170             }else{
171                 dif1 = diffSum + 1;
172             }
173             swap(next_table[0][1][0],next_table[0][0][0]);
174             if(count < max_depth && dif1 <= max_depth-count){
175                 recursive(next_table[0],max_depth,count+1,zero_loc,0);
176             }
177         }
178
179         if(pre_loc != 11){
180             if((table[1][1]-1)%3==0){
181                 dif2 = diffSum - 1;
182             }else{
183                 dif2 = diffSum + 1;
184             }
185             swap(next_table[1][1][0],next_table[1][1][1]);
186             if(count < max_depth && dif2 <= max_depth-count){
187                 recursive(next_table[1],max_depth,count+1,zero_loc,11);
188             }
189         }
190
191         if(pre_loc != 20){
192             if((table[2][0]-1)/3<=1){
193                 dif3 = diffSum - 1;
194             }else{
195                 dif3 = diffSum + 1;
196             }
197             swap(next_table[2][1][0],next_table[2][2][0]);
198             if(count < max_depth && dif3 <= max_depth-count){
199                 recursive(next_table[2],max_depth,count+1,zero_loc,20);
200             }
201         }
202         break;
203     case 11:
204         for(int a=0;a<4;a++){
205             for(int b=0;b<3;b++){
206                 for(int c=0;c<3;c++){
207                     next_table[a][b][c] = table[b][c];
208                 }
209             }
210         }
211
212         if(pre_loc != 1){
213             if((table[0][1]-1)/3>=1){
214                 dif1 = diffSum - 1;
215             }else{
216                 dif1 = diffSum + 1;
217             }
218             swap(next_table[0][1][1],next_table[0][0][1]);
219             if(count < max_depth && dif1 <= max_depth-count){
220                 recursive(next_table[0],max_depth,count+1,zero_loc,1);
221             }
222         }
223
224         if(pre_loc != 10){
225             if((table[1][0]-1)%3>=1){
226                 dif2 = diffSum - 1;
227             }else{
228                 dif2 = diffSum + 1;
229             }
230             swap(next_table[1][1][1],next_table[1][1][0]);
231             if(count < max_depth && dif2 <= max_depth-count){
232                 recursive(next_table[1],max_depth,count+1,zero_loc,10);
233             }
234         }
235
236         if(pre_loc != 12){
237             if((table[1][2]-1)%3<=1){
238                 dif3 = diffSum - 1;
239             }else{
240                 dif3 = diffSum + 1;
241             }
242             swap(next_table[2][1][1],next_table[2][1][2]);
243             if(count < max_depth && dif3 <= max_depth-count){
244                 recursive(next_table[2],max_depth,count+1,zero_loc,12);
245             }
246         }
247
248         if(pre_loc != 21){
249             if((table[2][1]-1)/3<=1){
250                 dif4 = diffSum - 1;
251             }else{
252                 dif4 = diffSum + 1;
253             }
254             swap(next_table[3][1][1],next_table[3][2][1]);
255             if(count < max_depth && dif4 <= max_depth-count){
256                 recursive(next_table[3],max_depth,count+1,zero_loc,21);
257             }
258         }
259         break;
260     case 12:
261         for(int a=0;a<3;a++){
262             for(int b=0;b<3;b++){
263                 for(int c=0;c<3;c++){
264                     next_table[a][b][c] = table[b][c];
265                 }
266             }
267         }
268
269         if(pre_loc != 2){
270             if((table[0][2]-1)/3>=1){
271                 dif1 = diffSum - 1;
272             }else{
273                 dif1 = diffSum + 1;
274             }
275             swap(next_table[0][1][2],next_table[0][0][2]);
276             if(count < max_depth && dif1 <= max_depth-count){
277                 recursive(next_table[0],max_depth,count+1,zero_loc,2);
278             }
279         }
280
281         if(pre_loc != 11){
282             if((table[1][1]-1)%3==2){
283                 dif2 = diffSum - 1;
284             }else{
285                 dif2 = diffSum + 1;
286             }
287             swap(next_table[1][1][2],next_table[1][1][1]);
288             if(count < max_depth && dif2 <= max_depth-count){
289                 recursive(next_table[1],max_depth,count+1,zero_loc,11);
290             }
291         }
292
293         if(pre_loc != 22){
294             if((table[2][2]-1)/3<=1){
295                 dif3 = diffSum - 1;
296             }else{
297                 dif3 = diffSum + 1;
298             }
299             swap(next_table[2][1][2],next_table[2][2][2]);
300             if(count < max_depth && dif3 <= max_depth-count){
301                 recursive(next_table[2],max_depth,count+1,zero_loc,22);
302             }
303         }
304         break;
305     case 20:
306         for(int a=0;a<2;a++){
307             for(int b=0;b<3;b++){
308                 for(int c=0;c<3;c++){
309                     next_table[a][b][c] = table[b][c];
310                 }
311             }
312         }
313
314         if(pre_loc != 10){
315             if((table[1][0]-1)/3==2){
316                 dif1 = diffSum - 1;
317             }else{
318                 dif1 = diffSum + 1;
319             }
320             swap(next_table[0][2][0],next_table[0][1][0]);
321             if(count < max_depth && dif1 <= max_depth-count){
322                 recursive(next_table[0],max_depth,count+1,zero_loc,10);
323             }
324         }
325
326         if(pre_loc != 21){
327             if((table[2][1]-1)%3==0){
328                 dif2 = diffSum - 1;
329             }else{
330                 dif2 = diffSum + 1;
331             }
332             swap(next_table[1][2][0],next_table[1][2][1]);
333             if(count < max_depth && dif2 <= max_depth-count){
334                 recursive(next_table[1],max_depth,count+1,zero_loc,21);
335             }
336         }
337         break;
338     case 21:
339         for(int a=0;a<3;a++){
340             for(int b=0;b<3;b++){
341                 for(int c=0;c<3;c++){
342                     next_table[a][b][c] = table[b][c];
343                 }
344             }
345         }
346
347         if(pre_loc != 20){
348             if((table[2][0]-1)%3>=1){
349                 dif1 = diffSum - 1;
350             }else{
351                 dif1 = diffSum + 1;
352             }
353             swap(next_table[0][2][1],next_table[0][2][0]);
354             if(count < max_depth && dif1 <= max_depth-count){
355                 recursive(next_table[0],max_depth,count+1,zero_loc,20);
356             }
357         }
358
359         if(pre_loc != 11){
360             if((table[1][1]-1)/3==2){
361                 dif2 = diffSum - 1;
362             }else{
363                 dif2 = diffSum + 1;
364             }
365             swap(next_table[1][2][1],next_table[1][1][1]);
366             if(count < max_depth && dif2 <= max_depth-count){
367                 recursive(next_table[1],max_depth,count+1,zero_loc,11);
368             }
369         }
370
371         if(pre_loc != 22){
372             if((table[2][2]-1)%3<=1){
373                 dif3 = diffSum - 1;
374             }else{
375                 dif3 = diffSum + 1;
376             }
377             swap(next_table[2][2][1],next_table[2][2][2]);
378             if(count < max_depth && dif3 <= max_depth-count){
379                 recursive(next_table[2],max_depth,count+1,zero_loc,22);
380             }
381         }
382         break;
383     case 22:
384         for(int a=0;a<2;a++){
385             for(int b=0;b<3;b++){
386                 for(int c=0;c<3;c++){
387                     next_table[a][b][c] = table[b][c];
388                 }
389             }
390         }
391
392         if(pre_loc != 12){
393             if((table[1][2]-1)/3==2){
394                 dif1 = diffSum - 1;
395             }else{
396                 dif1 = diffSum + 1;
397             }
398             swap(next_table[0][2][2],next_table[0][1][2]);
399             if(count < max_depth && dif1 <= max_depth-count){
400                 recursive(next_table[0],max_depth,count+1,zero_loc,12);
401             }
402         }
403
404         if(pre_loc != 21){
405             if((table[2][1]-1)%3==2){
406                 dif2 = diffSum - 1;
407             }else{
408                 dif2 = diffSum + 1;
409             }
410             swap(next_table[1][2][2],next_table[1][2][1]);
411             if(count < max_depth && dif2 <= max_depth-count){
412                 recursive(next_table[1],max_depth,count+1,zero_loc,21);
413             }
414         }
415         break;
416     }
417 }
418
419 int main(){
420
421     int table[3][3],start_count;
422
423     for(int i=0;i<3;i++){
424         scanf("%d %d %d",&table[i][0],&table[i][1],&table[i][2]);
425     }
426     start_count = calcDiffSum(table);
427     func(table,start_count);
428
429     printf("%d\n",minimum);
430
431     return 0;
432 }
```