8-puzzle See website link http://mypuzzle.org/sliding

Classwork 24.1(6) 8puzzle.f95. Write a program for playing the 8-puzzle.

We represent the 8-puzzle board with a matrix $b$ with 0 being the “hole”. An initial matrix might be

\[
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 7 \\
8 & 6 & 0
\end{pmatrix}
\quad \text{and we slide blocks around to get the goal} \quad
\begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 0
\end{pmatrix}.
\]

For any given board, each numbered block $n$ on the board will have a position. Let $[i_0,j_0]$ be the position of the hole, i.e., 0. For this initial board $[i_0,j_0] = [3,3]$.

Moving a block into the hole is equivalent to swapping the positions of the block and the hole and this is equivalent to moving the hole into the block’s position. The position $[i_{nxt},j_{nxt}]$ of the block will be the next position of the hole. The user uses arrow keys to move the hole around.
!c23_1_6_8puzzle.f95

program eight_puzzle
! implicit none

integer :: b(3,3), goal(3,3)
integer :: m=0, n, i0=3, j0=3, inxt, jnxt

b(1,:)= (/1,2,3/); goal(1,:)= (/1,2,3/)
b(2,:)= (/4,5,7/); goal(2,:)= ______
b(3,:)= (/8,6,0/); goal(3,:)= ______
5 format (3(i2))
print 5, (b(i,:), i=1,3)
print*, "Enter numpad arrow key or 5. 0 quits."

do
  read *, n
  if (n==0) _____
  select case (n)
    case (5, 2)
      inxt=min(i0+1,3); jnxt=j0; m=m+1
    case (___)
      ______
    case (___)
      ______
    case (___)
      ______
    case (___)
      ______
  end select
  i0=inxt; j0=jnxt
  print 5, (b(i,:), i=1,3)
do
case default  !case for wrong key inputs

_______
endselect
   call swap( ______ , ______ )
i0= ______ ; j0= ______
print 5, (b(i,:), i=1,3)
if(all(b==goal)) then;
   print*,'Solved in',m,' moves';exit;
endif
enddo
endprogram

subroutine swap(i,j);
k=i; i=j; j=k;
endsubroutine

The `min_path` algorithm can solve the 2×3 board puzzle. For each board, endcode its 2×3 matrix `a` as the number of a vertex of a graph `g`. To each 2×3 board `a`, encoder will assign the number of the graph’s matching vertex. Going in the reverse direction, for the number of each vertex of the graph, decoder gives the matching board matrix `a`. Each move (one block slide) between two boards is mapped to an
edge between the corresponding vertices with weight 1. The \texttt{start} and \texttt{goal} boards will map to a start and goal vertex of the graph, \texttt{g\_start} and \texttt{g\_goal}. The \texttt{minpath} algorithm then finds a path from \texttt{g\_start} to \texttt{g\_goal}. The \texttt{decoder} then maps this path back to a sequence of moves from \texttt{start} to \texttt{goal} which solves the problem.

For example
\begin{align*}
1 & \quad 2 & \quad 3 \\
5 & \quad 6
\end{align*}
and
\begin{align*}
1 & \quad 2 & \quad 3 \\
5 & \quad 6
\end{align*}
are two boards separated by just one move. These two boards are encoded by two vertices of the graph \texttt{g} which has an edge of weight 1 between them. Since there are $6! = 6 \cdot 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 720$ different boards, the graph \texttt{g} has 720 vertices, one for each board. We consider only this small $2 \times 3$ board since a $4 \times 4$ puzzle has $16! \approx 2 \times 10^{13}$ possible boards which is too big even for a supercomputer.

\textbf{CLASSWORK 24.2(6) 8puzzle\_ai.f95. Write an AI program for solving the $2 \times 3$ 8-puzzle.} Copy from website.
!c24_2_6_8puzzle_ai.f95

subroutine min_length(g,min_g,n)
   integer::g(720,720),min_g(720,720),n,steps
   steps=1; min_g=g
   do
      do i=1,n-1; do j=i+1,n
         min_g(i,j)=minval(min_g(i,1:n)+min_g(1:n,j))
         min_g(j,i)=min_g(i,j)
      enddo;enddo
      steps=2*steps; if(steps>=n-1) return
   enddo
endsubroutine

logical function is_allowed(b) !only allow permutations
   integer::b(2,3)
   is_allowed = ______
   do i=1,2; do j=1,3; do i1=1,2; do j1=1,3
      if(i==i1 .and. j==j1) cycle
      if(_______) then
         is_allowed = ______; return
      endif
   enddo;enddo;enddo;enddo
endfunction

subroutine coder(encoder,decoder,numvertices)
   integer::encoder(0:5,0:5,0:5,0:5,0:5,0:5)
   integer::decoder(720,2,3)
   integer::t1,t2,t3,b1,b2,b3
   integer::numvertices,a(2,3),vertex
   logical::is_allowed

encoder=-1; decoder=-1
vertex=0
do t1=0,5; do t2=0,5; do t3=0,5
do b1=0,5; do b2=0,5; do b3=0,5
  a(1,:)=(/t1,t2,t3/); a(2,:)=(/b1,b2,b3/)
  if(.not. is_allowed(a)) cycle
  vertex = ______
  encoder(t1,t2,t3,b1,b2,b3)= ______
  decoder(vertex,1,:)=(/ ______ /)
  decoder(vertex,2,:)=(/ ______ /)
enddo; enddo; enddo; enddo; enddo; enddo
numvertices = vertex
endsubroutine

subroutine make_graph(g,encoder,decoder,numvertices)
  integer:: g(720,720), u, v, direction
  integer:: encoder(0:5,0:5,0:5,0:5,0:5,0:5)
  integer:: decoder(720,2,3)
  integer:: a(2,3), b(2,3), numvertices
  logical:: is_allowed
  g=10**6; do i=1,720; g(i,i)=0; enddo
do u=1, numvertices
  a=decoder(u,:,:)  
do direction = 1,4
    call move(a,direction,b)
    if(all(a==b)) cycle
    v=encoder( ______ )
    g(u,v)= ______
endo
 subroutine move(a,direction,b)
  integer::a(2,3),b(2,3),c(2,3),direction,inxt,jnxt,i,j
  integer::mloc(2)
  logical::is_allowed
  mloc=minloc(a); i=mloc(1); j=mloc(2)
  inxt=i; jnxt=j; b=a
  select case(direction)
    case(1); inxt=min(i+1,2)
    case(2); inxt=max(i-1,1)
    case(3); jnxt=min(j+1,3)
    case(4); jnxt=max(j-1,1)
  endselect
  call swap(b(i,j),b(inxt,jnxt))
 end subroutine

 program test_5_puzzle
  integer::start(2,3),a(2,3),b(2,3),goal(2,3),direction
  integer::m=0, i=3, j=3,inxt,jnxt,num_moves=0
  integer::g(720,720),min_g(720,720),path(720)
  integer::encoder(0:5,0:5,0:5,0:5,0:5,0:5)
  integer::decoder(720,2,3)
  integer::g_start,g_goal,numvertices=720
  logical::is_allowed
  start(1,:)=(/4,2,0/); goal(1,:)=(/1,2,3/)
  start(2,:)=(/5,1,3/); goal(2,:)=(/4,5,0/)
  a=start
  call print_board(a)
  call coder(encoder,decoder,numvertices)
g_start=encoder(start(1,1), start(1,2), &
& start(1,3), start(2,1), start(2,2), start(2,3))
g_goal=encoder(goal(1,1), goal(1,2), goal(1,3), &
& goal(2,1), goal(2,2), goal(2,3))
print*, g_start, g_goal
call make_graph(g, encoder, decoder, numvertices)
!~ call print_graph(g, numvertices)
call print_encoding(decoder, numvertices)
call min_length(g, min_g, numvertices)
!~call print_graph(min_g, numvertices)
call find_path(g, min_g, numvertices, g_start, g_goal, path, m)
do i=1, m;
   a=decoder(path(i), :, :, :)
   call print_board(a)
enddo
print*, m-1, " steps"
endprogram

subroutine find_path(g, min_g, n, i, j, path, m)
integer:: g(720, 720), min_g(720, 720)
integer:: path(720), i, j, mm(1)
m=1; path(1)=i;
do
   if(path(m)==j) exit;
   mm=minloc(max(g(path(m), 1:n), 1) + min_g(1:n, j))
   m=m+1; path(m)=mm(1);
enddo
end subroutine

subroutine print_encoding(decoder, numvertices)
integer:: a(2, 3), decoder(720, 2, 3), numvertices, u
9  format(a,i3,a)
do u=1,numvertices
    write(*,9) '(',u,')='
a=decoder(u,:,:)
call print_board(a)
enddo; print*
endsubroutine

subroutine print_graph(g,n)
integer::g(720,720)
11  format(5x,200(1x,a,i2,a))
print 11,('(',i,')',i=1,50)
12  format(3x,'(',i2,')',720(i2,3x))
do i=1,50; print 12,i,g(i,1:50); enddo; print*
endsubroutine

subroutine swap(i,j); k=i; i=j; j=k;
endsubroutine
subroutine print_board(b)
integer::b(2,3)
5  format(3(i2))
print 5,(b(k,:),k=1,2)
print*
endsubroutine

CLASSWORK 24.1(6)8puzzle 24.2(6)8puzzle_ai
email: dale@math.hawaii.edu  subject line: 190 c24(12)
Quizzed on the 15puzzle program. Not quizzed on majic_squares.
Homework 24.1(7) 15_puzzle.f95. Write a program for playing the 15-puzzle (4×4 instead of 3×3).

email: dale@math.hawaii.edu subject line: 190 h24.1(7)

You have a 4×4 matrix of 15 sliding blocks with the goal
\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15
\end{array}
\]

You need to change the sizes of the boards \( b, \text{goal} \). For your initial board \( b \), pick any board other than the goal board. Also change 5 format(3(i2)) to 5 format(4(i3)) You also need to change the initial position \([i0,j0]\) of the hole in your initial board. The classwork line

```fortran
integer::m=0,n,i0=3,j0=3,inxt,jnxt
```

initialized the position of the hole to \([3,3]\) which was the hole position of its initial board.

! 15_puzzle.f95 subject line: 190 h24.1(7)
A magic square is an \( n \times n \) array of containing the integers 1, 2, 3, ..., \( n^2 \). Each of these integers occurs exactly once in the matrix and all rows, columns and the two diagonals all have the same sum. In the 3\( \times \)3 magic square below, all rows, columns and diagonals total to 15, \( 2+7+6=15 \), \( 2+9+4=15 \), \( 2+5+8=15 \).

<table>
<thead>
<tr>
<th>2</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Here’s an algorithm for constructing a magic square for odd \( n \). We will place the integers one at a time in increasing order \( k = 1, 2, 3, ..., n^2 \). Put 1 in the center of the top row. After filling a square with an integer \( k \), the next integer, \( k+1 \) will go in the square which is diagonally up and to the right of \( k \)’s square. If there is no higher row, wrap to the bottom row. If there is no column on the right, wrap to the first column. If the next square is already filled, then move down from \( k \)’s square. Here’s the 3\( \times \)3 square.
We represent a magic square with an $n \times n$ matrix of integers where 0 indicates an empty square.

The center of the top row has position $(1, (1 + n)/2)$. 

Write a program `magic_square` which fills a 3×3 magic square using the algorithm above with a subroutine `next_square` which calculates the next position.

```fortran
program magic_square
integer::board(3,3)=0
integer::n=3, k=1, i, j, next_i, next_j

! the cell numbers are k=1,2,3,...
! (i,j), (next_i,next_j) are current and next board positions.
! start at top middle (i, j) = (1, (1+n)/2) and cell number k=1.
i= ____ ; j=(1+n)/2; board(i,j)= ____
call print_board
do k=2,n**2  ! k=the number to put in position board(i,j)
    ! rule for the next square's row index next_i.
    if(i>1)then;  next_i=i-1
    else;  next_i=n
    endif
    ! rule for the next square's column index next_j.
    if(____)then;  next_j= ____
enddo
```
else; next_j = ____
endif
if(board(next_i, next_j) > 0) then
    i = ____; j = ____ ! case for occupied position
else
    i = ____; j = ____ ! case for unoccupied position
endif
    board(i, j) = ____
call print_board
doddo
contains

subroutine print_board
integer::i
    print*, (board(i,:), char(10), i=1,3)
endsubroutine
endprogram