River crossing problems: fox, geese, missionaries

A farmer has a fox, goose and bag or corn on a river’s left bank. He must transport them to the right bank with a boat which can hold him and one other occupant. He can’t leave the fox and goose alone on the same side (fox eats goose) and can’t leave the goose and corn alone on the same side (goose eats corn). Goal: move everything to the right bank.

By a state or configuration or board we mean the arrangement of the items on the respective sides of the river. We will encode a board (state, configuration) with a matrix $\mathbf{b}$. For each board, there are moves (rowing a boat load across the river) which can produce a new board. There is an initial board start and a goal board goal. Solving the problem means finding a sequence of moves leading from start to goal.

Some boards are allowed, some not; some moves are allowed, some not. Write functions is_allowed_board and is_allowed_move which return .true. if allowed and .false. if not. Write a subroutine move which given a board and a move (a boatload), calculates the next board. If the next
board is not allowed, we discard it and stay with the original board. Finally write a program which allows a user to solve the problem.

**Fox goose corn problem**

Encode a configuration with a $5 \times 2$ matrix \( a \). Rows 1 - 5 encode the respective positions of boat=1, man=2, fox=3, goose=4, corn=5. The first column of \( a \) gives the number of items on the left bank; the second column gives the number on the right bank. If

\[
\begin{array}{c|cc}
 & \text{left bank} & \text{right bank} \\
\hline
\text{boat} & 1 & 0 \\
\text{man} & 1 & 0 \\
\text{fox} & 0 & 1 \\
\text{goose} & 1 & 0 \\
\text{corn} & 0 & 1 \\
\end{array}
\]

then \( a \) represents the configuration

\[
[ \text{man goose} ] \text{boat} \quad \rightarrow \quad [ \text{fox corn} ]
\]

Left bank: boat, man and goose.

Right bank: fox and corn.
The start and goal configurations are:

\[
\text{start} = \begin{bmatrix}
1 & 0 \\
1 & 0 \\
1 & 0 \\
1 & 0 \\
1 & 0 \\
\end{bmatrix} \quad \text{goal} = \begin{bmatrix}
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
0 & 1 \\
\end{bmatrix}
\]

A move is determined by a boat’s load which we represent with a 5-place vector \( \text{load} = [\text{load}(1), \ldots, \text{load}(5)] = \text{load}(\text{boat}), \text{load}(\text{man}), \text{load}(\text{fox}), \text{load}(\text{goose}), \text{load}(\text{corn}) \). \text{load} = [1,1,0,0,1] means the boat load is the boat (position 1), man (position man=2) and corn (position corn=5). No fox, no goose (0s in positions 3, 4).
program fox_goose
integer::a(5,2),next_a(5,2),start(5,2),goal(5,2)
integer,parameter::boat=1,man=2,fox=3,goose=4,corn=5
integer::num_moves=0,load(5)
character(2)::choice
logical::allowed_move,allowed_board
start(boat,:)=(/1,0/); goal(boat,:)=(/0,1/)
start(man,:)=(/1,0/); goal(man,:)=(/0,1/)
start(fox,:)=(/1,0/); goal(fox,:)=(/0,1/)
a=start; 9 format(/,a,/)print 9," ENTER: m,f,g,c,q=quit "
do
call print_board(a)
if(all(a==goal))then;
  10 format(a,i2,a,/)print 10,"Solved in ",num_moves," steps."
exit
endif
read*,choice
if(choice=="q") then; exit; endif
SELECT CASE (choice)
   CASE ("m");  load=(/1,1,0,0,0/)
   CASE ("f");  load=(/1,1,1,0,0/)
   CASE ("g");  load=(/1,1,0,1,0/)
   CASE ("c");
   CASE default;load=(/0,0,0,0,0/)
ENDSELECT
call move(a,load,next_a)
if(all(a==next_a))then
   print*,"Try again."; cycle
endif
   a=next_a; num_moves=num_moves+1
enddo
dendprogram
subroutine move(a,load,next_a)
integer::load(5),a(5,2),next_a(5,2),boat=1
integer::boatside,otherside
logical::allowed_board,allowed_move
if(.not. allowed_move(a,load))then
   next_a=a; return ! no change if not allowed
endif
boatside=dot_product((/1,2/),a(boat,:))
otherside = dot_product((/2,1/), a(boat,:))
next_a(:,boatside) = a(:,boatside) - load
next_a(:,otherside) = ________
if (.not. allowed_board(next_a)) then
    ________  ! no change if not allowed
endif
endsubroutine

logical function allowed_board(a)
integer:: a(5,2)
integer:: boat=1, man=2, fox=3, goose=4, corn=5
logical:: rule(4)
rule(1) = .not. (a(fox,1) == a(goose,1) .and. a(man,1) /= a(goose,1))
rule(2) = .not. (a(fox,2) == a(goose,2) .and. a(man,2) /= a(goose,2))
rule(3) = ________
rule(4) = ________
allowed_board = all(rule)
endfunction

logical function allowed_move(a, load)
integer:: a(5,2), load(5), boatside
integer:: boat=1, man=2, fox=3, goose=4, corn=5
logical:: rule(3)
boatside = dot_product((/1,2/), a(boat,:))
rule(1) = load(man) == 1
rule(2) = sum(load(2:5)) <= 2
As with the 8-puzzle, the min-path algorithm can solve puzzles of this kind. First encode the configurations (matrices $a$) as vertices (numbers) of a graph $g$. When a new allowed matrix configuration $a$ is found, encoder matches it with the next vertex number $vertex = vertex + 1$. Going in the
reverse direction, for each graph vertex, decoder gives the matching configuration matrix \( a \). Each move (boat load) between two configurations \( a, b \) is mapped to an edge between the corresponding vertices \( u, v \). The graph matrix \( g \) labels these edges with weight 1, \( g(u, v) = 1 \). The start and goal configurations will map to vertices \( g_{\text{start}} \) and \( g_{\text{goal}} \) of the graph. The minpath algorithm then finds a path from \( g_{\text{start}} \) to \( g_{\text{goal}} \). decoder then maps this path back to a sequence of moves from start to goal which solves the problem.

**CLASSWORK 25.2(6)** fox_goose_ai.f95 On your own.

!c25_2_6fox_goose_ai.f95

```fortran
program fox_goose_corn
integer::a(5,2),next_a(5,2),start(5,2),goal(5,2)
integer::boat=1,man=2,fox=3,goose=4,corn=5
integer::g(100,100),min_g(100,100),path(100)
integer::load(5),num_moves=0,decoder(100,5,2)
integer::encoder(0:1,0:1,0:1,0:1,0:1)
integer::g_start,g_goal; character(1)::choice
logical::allowed_move,allowed_load
start(boat,:)= (/1,0/); goal(boat,:)= (/0,1/)```

start(man,:) = (/1, 0/); goal(man,:) = (/0, 1/)
start(fox,:) = (/1, 0/); goal(fox,:) = (/0, 1/)

________________ ; ________________
________________ ; ________________

a = start

call coder(encoder, decoder, numvertices)
g_start = encoder(start(1,1), start(2,1), &
& start(3,1), start(4,1), start(5,1))
g_goal = encoder(goal(1,1), goal(2,1), &
& goal(3,1), goal(4,1), goal(5,1))
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;
    call print_board(decoder(path(i),:,:))
endo
dono
print*, m-1, " steps"
endprogram

!********************
subroutine move(a, load, next_a)
    !... copy from fox_goose.f95
endsubroutine
logical function allowed_board(a)
  !... copy from fox_goose.f95
endfunction
logical function allowed_move(a,load)
  !... copy from fox_goose.f95
endfunction
!

subroutine coder(encoder,decoder,numvertices)
integer::encoder(0:1,0:1,0:1,0:1,0:1),vertex
ingTEGER::decoder(100,5,2),a(5,2),numvertices
integer::boat=1,man=2,fox=3,goose=4,corn=5
logical::allowed_board
vertex=0
DO nm=1,0,-1;DO nf=1,0,-1;DO ng=1,0,-1;DO nc=1,0,-1
   a(:,1)=(/nm,nm,nf,ng,nc/)!
   a(:,2)=(/1,1,1,1,1/)-a(:,1)
   IF(.NOT. allowed_board(a))cycle
vertex = ________
   encoder(nm,nm,nf,ng,nc)= vertex
   decoder(vertex,:,:)= a
endo;endo;endo;endo;endo
numvertices=vertex
endsubroutine
subroutine make_graph (g, encoder, decoder, numvertices)
integer:: boat=1, man=2, fox=3, goose=4, corn=5
integer:: numvertices, g(100,100), u, v
integer:: encoder(0:1,0:1,0:1,0:1,0:1)
integer:: a(5,2), b(5,2), load(5), decoder(100,5,2)
g=10**6; do i=1,100; g(i,i)=0; enddo
do u=1,numvertices
  a=decoder(u,:,:);
  do nf=0,1; do ng=0,1; do nc=0,1
    load=(/1,1,nf,ng,nc/)
    call move(a,load,b)
    if(.not. all(a==b)) then !a=b means invalid move
      v=encoder(b(1,1),b(2,1),b(3,1),b(4,1),b(5,1))
      g(u,v)=
    endif
  enddo; enddo; enddo
endo
end subroutine

subroutine min_length (g, min_g, n)
integer:: g(100,100), min_g(100,100), 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
endo
end subroutine
steps=2*steps; if(steps>=n-1) return
enddo
endsubroutine

subroutine find_path(g,min_g,n,i,j,path,m)
  integer::g(100,100),min_g(100,100)
  integer::path(100),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
  10 format(i2,a,i2,a,i3,a,200(i3))
endsubroutine

subroutine print_board(a); integer::a(5,2)
  integer::boat=1,man=2,fox=3,goose=4,corn=5
  character(30)::right,left;character(8)::middle
  right=repeat(" man",a(man,1))//repeat(" fox",a(fox,1))
  right=trim(right)//repeat(" goose",a(goose,1))
  right=trim(right)//repeat(" corn",a(corn,1))
  left=repeat(" man",a(man,2))//repeat(" fox",a(fox,2))
  left=trim(left)//repeat(" goose",a(goose,2))
  left=trim(left)//repeat(" corn",a(corn,2))
  if(a(boat,1)==1)then
    print*,"[",trim(right)," ]boat--[",trim(left)," ]"
  else
    print*,"[",trim(right)," ]--boat[",trim(left)," ]"
  endif
endsubroutine
subroutine print_graph(g,n)
integer::g(100,100)
11 format(5x,200(1x,a,i2,a))
print 11,('(','i,')',i=1,n)
12 format(3x,'(','i2,')',200(i2,3x))
do i=1,n; print 12,i,g(i,1:n); enddo; print*
endsubroutine
subroutine print_encoding(decoder,numvertices)
integer::c(5),a(5,2),decoder(100,5,2),numvertices,u
9 format(a,i2,a)
do u=1,numvertices
   write(*,9,advance='no') '(',u,')='
   call print_board(decoder(u,:,:))
endo; print*
endsubroutine

Classwork 25.1(6) fox_goose 25.2(6) fox_goose_ai
Email: dale@math.hawaii.edu  Subject line: 190  c25(12)

For the quiz, one of the miss_cann subroutines
allowed_board, allowed_move, move.

Missionaries/Cannibals Problem

There are three missionaries, three cannibals and one boat on a river’s left bank. The cannibals can’t outnumber the missionaries on any side (cannibals eat missionaries). The boat
always needs at least one passenger (doesn’t row itself) and can have at most two. Goal: move everyone to the right bank.

Encode a configuration with a $3 \times 2$ matrix $a$. Rows 1-3 encode the respective positions of $\text{boat}=1, \text{miss}=2, \text{cann}=3$. The first column of $a$ gives the number of items on the left bank; the second column gives the number on the right bank. If

$$a = \begin{bmatrix}
\text{boat} & 1 \\
\text{miss} & 3 \\
\text{cann} & 1 \\
\end{bmatrix}
$$

then $a$ represents the configuration $[ \text{mmm c } \text{boat} -- [ \text{cc } ] ]$

Left bank: boat, 3 missionaries, 1 cannibal
Right bank: 2 cannibals.

The start and goal configurations are:

$$\begin{bmatrix}
1 & 0 \\
3 & 0 \\
3 & 0 \\
\end{bmatrix} \quad \begin{bmatrix}
0 & 1 \\
0 & 3 \\
0 & 3 \\
\end{bmatrix}$$
A move is determined by a boat’s load which we represent with a 3-place vector \( \text{load} = [\text{load}(1), \text{load}(2), \text{load}(3)] = \text{load(boat), load(miss), load(cann)} \). \( \text{load} = [1, 2, 1] \) means the boat load is the boat (position boat=1), 2 missionaries (position miss=2) and 1 cannibal (position cann=3).

**Homework 25.1(6)** miss_cann.f95

email: dale@math.hawaii.edu subject: 190 h25.1(6)

!miss_cann.f95 subject line: 190 h25.1(6)

!3 missionaries, 3 cannibals
!cannibals may not outnumber the missionaries on either bank.
!a for positions, a(:,1)) = # on left side, a(:,2) = # on right side
!rows 1,2,3 for boat, missionaries, cannibals, boat holds 2

program miss_cann
integer::a(3,2),next_a(3,2)
integer::start(3,2),goal(3,2),load(3)
integer:: boat=1,miss=2,cann=3,num_moves=0
character(2)::choice
logical::allowed_move,allowed_board
start(boat,:)=(/1,0/); goal(boat,:)=(/0,1/)
start(miss,:)=(/3,0/); goal(miss,:)=(/0,3/)
__________________; ____________________
a=start
10 format(/,a,/
print 10, " ENTER: m,c,mm,mc,cc,q=quit "
do
  call print_board(a)
  if(all(a==goal))then;
    11 format(a,i2,a,/
    print 11,"Solved in ",num_moves," steps."
    exit
  endif
read*,choice
if(choice=="q") then; exit; endif
SELECT CASE (choice)
  CASE ("m"); load=(/1,1,0/)
  CASE ("c"); load=(/1,0,1/)
  CASE ("mm"); load=(/1,2,0/)
  CASE ("mc"); load=(/1,1,1/)
  CASE ("cm"); load=________
  CASE ("cc"); load=________
  case default;load=(/0,0,0/)
ENDSELECT
  call move(a,load,next_a)
  if(all(a==next_a))then
    print*,"Try again."
    cycle
  endif
  a=next_a; num_moves=num_moves+1;
enddo  
endprogram  

subroutine move(a,load,next_a) 
integer::load(3),a(3,2),next_a(3,2),boat=1 
integer::boatside,otherside  
logical::allowed_board,allowed_move 
if(.not. allowed_move(a,load))then  
    next_a=a; return 
endif 
boatside=dot_product((/1,2/),a(boat,:))  
otherside=dot_product((/2,1/),a(boat,:))  
next_a(:,boatside)=a(:,boatside)-load  
next_a(:,otherside)= __________  
if(.not. allowed_board(next_a))then  
    __________ ! no change if not allowed.  
endif  
endsubroutine  

logical function allowed_board(a) 
integer::a(3,2),boat=1,miss=2,cann=3  
logical::rule(2)  
rule(1)= a(miss,1)==0 .or. a(miss,1)>=a(cann,1)  
rule(2)= __________ ! case for side 2  
allowed_board = all(rule)
endfunction

logical function allowed_move(a, load)
integer:: a(3,2), load(3)
integer:: boat=1, miss=2, cann=3, boatside
logical:: rule(3)
boatside=dot_product((/1, 2/), a(boat,:))

rule(1)= ________ <= sum(load(2:3)) ! at least on
rule(2)= sum(load(2:3)) <= ________ ! at most two
rule(3)= all(load <= a(:,boatside))

allowed_move = all(rule)
endfunction

subroutine print_board(a)
integer:: a(3,2), boat=1, miss=2, cann=3
character(30):: right, left
right= repeat("m", a(miss, 1))//repeat("c", a(cann, 1))
left= repeat("m", a(miss, 2))//repeat("c", a(cann, 2))
if(a(boat,1)==1) then
    print*, "[", trim(right), "]boat--[", trim(left), "]"
else
    print*, "[", trim(right), "]--boat[", trim(left), "]"
endif
endsubroutine
program miss_cann
integer::a(3,2),next_a(3,2),start(3,2),goal(3,2)
integer::boat=1,miss=2,cann=3,num_moves=0
integer::g(100,100),min_g(100,100),path(100)
integer::encoder(0:1,0:3,0:3),decoder(100,3)
integer::g_start,g_goal,load(3)
character(2)::choice
logical::allowed_move,allowed_load
load(boat)=1; load(miss)=1; load(cann)=1
load(boat)=1; load(miss)=1; load(cann)=1
start(boat,:)=(/1,0/); goal(boat,:)=(/0,1/)
start(miss,:)=(/3,0/); goal(miss,:)=(/0,3/)
_____________ ; ________________
a=start
call coder(encoder,decoder,numvertices)
g_start=encoder(start(1,1),start(2,1),start(3,1))
g_goal=encoder(goal(1,1),goal(2,1),goal(3,1))
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(:,1)=decoder(path(i),:);
a(:,2)=(/1,3,3\()\)-a(:,1)
call print_board(a)
enddo
print*,m-1," steps"
endprogram

!************************************************

subroutine move(a,load,next_a)
!... copy from miss_cann.f95
endsubroutine

logical function allowed_board(a)
!... copy from miss_cann.f95
endfunction

logical function allowed_move(a,load)
!... copy from miss_cann.f95
endfunction

!************************************************

subroutine coder(encoder,decoder,numvertices)
integer:: encoder(0:1,0:3,0:3),decoder(100,3)
integer::numvertices,a(3,2),vertex
logical::allowed_board
vertex = 0
do k=1,0,-1; do i=3,0,-1; do j=3,0,-1
  a(:,1)=(/k,i,j/); a(:,2)=(/1,3,3/)-a(:,1)
  if(allowed_board(a)) then
    vertex = __________
    encoder(k,i,j)=vertex
    decoder(vertex,:)=(/k,i,j/)
  endif
enddo; enddo; enddo
numvertices = vertex
endsubroutine

subroutine make_graph(g,encoder,decoder,numvertices)
inginteger::numvertices,boat=1,miss=2,cann=3
integer::g(100,100),u,v
integer::encoder(0:1,0:3,0:3),decoder(100,3)
inginteger::a(3,2),next_a(3,2),load(3)
g=10**6; do i=1,100; g(i,i)=0; enddo
  do u=1,numvertices
    a(:,1)=decoder(u,:); a(:,2)=(/1,3,3/)-a(:,1)
    do nm=0,2; do nc=0,2
      load(boat)=1; load(miss)=nm; load(cann)=nc
      call move(a,load,next_a)
      if(.not. all(a==next_a)) then
        v=encoder(next_a(1,1),next_a(2,1),next_a(3,1))
        g(u,v)= ___
      endif
    enddo; enddo
endo do enddo
subroutine min_length(g, min_g, n)
integer:: g(100,100), min_g(100,100), 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
endo
endsubroutine

subroutine find_path(g, min_g, n, i, j, path, m)
integer:: g(100,100), min_g(100,100)
integer:: path(100), 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);
endo
endsubroutine

subroutine print_board(a)
integer:: a(3,2), boat=1, miss=2, cann=3
character(30):: right, left
right= repeat("m", a(miss,1))//repeat("c", a(cann,1))
left= repeat("m", a(miss,2))//repeat("c", a(cann,2))
if(a(boat,1)==1)then
  print*," [","trim(right),","]boat--[","trim(left),","]"
else
  print*," [","trim(right),","]--boat[","trim(left),","]"
endif
endsubroutine

subroutine print_graph(g,n)
  integer::g(100,100)
  11 format(5x,200(1x,a,i2,a))
  print 11,('[',i,']',i=1,n)
  12 format(3x,('[',i2,']'),200(i2,3x))
  do i=1,n; print 12,i,g(i,1:n); enddo; print*
endsubroutine

subroutine print_encoding(decoder,numvertices)
  integer::a(3,2),decoder(100,3),numvertices,u
  9 format(a,i2,a)
  do u=1,numvertices
    write(*,9,advance='no') '(',u,')='
    a(:,1)=decoder(u,:); a(:,2)=(/1,3,3/)-a(:,1)
    call print_board(a)
  enddo; print*
endsubroutine