Lecture 26 MinMax strategy

Homework 26.1(3) towerhanoi2.f95. Write a program for solving the Tower of Hanoi puzzle where the initial peg has 5 disks instead of 4 as in the classwork problem.

Email: dale@math.hawaii.edu subject line: 190 h26.1(3)

Classwork 26.2(2) max_hailstorm.f95 Find the maximum value of the hailstorm sequence starting with 97.

!c26_2_2max_hailstorm.f95

program hailstorm_max
integer:: M, inf=10**6, n=97
M= _______
do
  !c26 2 2max_hailstorm.f95
  print *, n
  if(n==1)exit
  if(mod(n,2)==0) then
    n=n/2; M= _______ ; cycle
  endif
  if(mod(n,2)/=0) then
    n=3*n+1; M= _______ ; cycle
  endif
enddo;
print*, "Max value = ", M
endprogram

!ans: 9232
**MinMax**

Player *I* and player *II* alternately take turns making moves, player *I* moves first. If *I* wins, he wins $1.00, *II* loses $1.00. If *II* wins, he gets $1.00 and *I* loses $1.00. In case of a draw, both get $0.00. The **payoff** is the amount *I* wins. It is also what *II* loses. Hence *I* wants to maximize the payoff (his winnings) and *II* wants to minimize the payoff (his losses). A negative payoff is a loss for *I* and a gain for *II*. *I* likes positive payoffs, *II* likes negative payoffs.

The moves in a game are pictured as a downward branching tree. The nodes of the tree are the various possible states, configurations or boards of the game. The start node (player *I*'s turn) is at the root at the top. Nodes at odd levels are boards for which it is *I*'s turn to move; nodes at even levels are for *II*. For each move a player can make, there is an edge going down to the next level. The game ends at nodes at the bottom of the tree. We use the minmax strategy to calculate the payoff (*I*'s winnings) for each node of the tree. Nodes at the bottom of the tree (leaves) are configurations where the game ends. They are given payoffs 1, -1, 0 if the configuration they represent are a win for *I*, a win for *II* or a draw.
Given a node, suppose, by recursion, that payoffs have been assigned to lower nodes. At nodes in levels for I, the payoff is the maximum of the payoffs on nodes below it. I will pick a move which leads to a lower node of maximum payoff since he wants to maximize his winnings. If it is II's turn the payoff is the minimum of the payoffs of nodes below it. II will pick a move which leads to a lower node of minimum payoff since he wants to minimize his loss. Calculate the payoffs for the nodes of the tree below and which moves each player should take. Base payoff = final move payoff.

For TicTacToe, the nodes are boards for the game with the empty board at the root. Nodes where I (i.e., player X) wins have
payoffs 1. Nodes where \( O \) wins have payoffs -1. Draws have payoff 0.

**Homework 26.2(3) tictactoeai.f95** Write a program which plays an optimal game for player \( X \) against a human player \( O \). Hint have player \( I \) start with \( X \) at the center. Thus this program is the same as the classwork program but with just one symbol changed.

**Email:** dale@math.hawaii.edu  **Subject line:** 190 h26.2(3)

**Classwork 26.3(6) tictactoe_ai.f95** Write a program which plays an optimal game for player \( O \) against a human player \( X \).

!c26_3_6tictactoeai.f95

```fortran
program tictactoe_ai
  integer :: move, payoff, payoff2
  character(1) :: b(3,3), a(3,3)
  b(1,:)=(/"-","-","-"/)
  b(2,:)=(/"-","-","-"/)
  b(3,:)=(/"-","-","-"/)
  print*, "Enter 0 to quit."
  call print_board(b)
  do
    print*, "Enter numpad position for ", player(b)
    read*, move
    if(move==0) exit
```
If(move<1 .or. move>9) cycle
i=3-(move-1)/3
j=1+mod(move-1,3)
if(b(i,j)/=' -') cycle
b(i,j)=player(b);
call print_board(b); print*
If(base_payoff(b)>-2) exit;

call get_payoff(b,payoff)
inxt=-1;jnxt=-1
do i=1,3; do j=1,3
   if(b(i,j)/=' -') cycle
   a=b; a(i,j)=player(b)
call get_payoff(a,payoff2)
   if(payoff2==payoff)then; inxt=i; jnxt=j;exit;endif
endo; enddo
b(inxt,jnxt)=player(b);
call print_board(b); If(base_payoff(b)>-2) exit
endo
contains
recursive subroutine get_payoff(b,payoff)
character(1)::b(3,3),a(3,3)
integer::i,j,payoff,payoff2,inxt,jnxt,inf=10**6
if(base_payoff(b)>-2)then;
   payoff= ______________ ;return;
endif
if(player(b)=="X")then;payoff= _______
else;payoff= _______
endif

do i=1,3; do j=1,3
  if(b(i,j)/='-',) cycle
  a=b; a(i,j)=player(b)
call get_payoff(a,payoff2)
Select case(player(b))
  case("X"); payoff= __________
  case("O"); payoff= __________
endselect
enddo; enddo
end subroutine

integer function base_payoff(b) result(payoff)
integer::i
character(1)::v(8,3),b(3,3); payoff=-2
v(1,:)=b(1,:);v(2,:)=b(2,:);v(3,:)=b(3,:)
v(4,:)=b(:,1);v(5,:)=b(:,2);v(6,:)=b(:,3)
v(7,:)=(/b(1,1),b(2,2),b(3,3)/)
v(8,:)=(/b(1,3),b(2,2),b(3,1)/)
do i=1,8
  if(all(v(i,:)=="X"))then;payoff=1;return;endif
  if(all(v(i,:)=="O"))then;payoff= _____ ;return;endif
enddo
if(all(b/="-"))then;payoff= _____ ;return;endif
end function
character(1) function player(b)
character(1)::b(3,3),symbol(0:1)
nummove=count(b/='-')+1;symbol=('O','X')
player=symbol(mod(nummove,2));
endfunction
subroutine print_board(b)
integer::k
character(1)::b(3,3)
5 format(3(1x,a1));print 5,(b(k,:),k=1,3)
select case(base_payoff(b))
  case(1);print*," X wins"
  case(-1);print*, ________________
  case(0);print*, ________________
endselect
endsubroutine
endprogram

Classwork 26.1(5) towerhanoi 26.2(2) max_hailstorm
26.3(5) tictactoeai

Email: dale@math.hawaii.edu Subject line: 190 c26(12)
Homework 26.1(3) towerhanoi2.f95 26.2(3)
tictactoeai.f95

Backtracking
Backtracking is also called depth-first search. It can be used as a trial-and-error process to search for a path out of a maze or to search for a coloring of a map. The search is pictured as a tree with a root at the top and branches extending downward. Each node of the tree has an ordered list (possibly empty) of children below it, first child, next child, ... . The node is the parent of its children. We search for a sequence of nodes called a path which starts at the root and meets certain criteria for success, e.g. it gets you out of a maze or the colors along the path color countries on a map. The end (bottom) of the path is the current node. If the current node is fails, we backtrack to the parent and then down to the next child, i.e., a later sibling. If there is no later sibling, we continue backward until one is found. If the current node is does not fail, we try extend the path downward to the first child of the current node. This increases the depth of the current node, hence the term depth-first search as opposed to breadth-first search which searches nodes at a given level before extending the search.
to a lower depth.
The matrix map below indicates countries that share boundaries: 1 if they share a boundary, 0 if not.
We want to find, if possible, a 3-coloring of this map. In the tree for this problem, each of the countries 1, 2, ..., 7 has its own level. For each country, there are three colors to try, \( r, g, b \) for red, green, blue. Each path from the root through the tree determines a coloring of the map. We have to find a path which colors the map so that countries (provinces in this case) which share a boundary get different colors.