HANDOUT 2

FERNANDO RODRIGUEZ-VILLEGAS

Hanoi Towers

The towers of Hanoi is a puzzle invented by the French mathematician Lucas in 1883. It consists of three pegs and n disks of decreasing size. At the start all the disks are in one peg stacked from largest at the bottom to smallest at the top. The goal of the puzzle is to transfer all the disks to another peg one at a time never having a larger disk on top of a smaller one.

Let us label the disks $d_1, \cdots, d_n$ in increasing order of size and p_0, p_1, p_2 the pegs, with p_0 the initial peg where the disks are and p_1 the final peg where the disks must be moved to. It is clear that in order to move d_n to p_1 all other disks $d_1, \cdots, d_{n-1}$ must be at p_2. Hence the following recursive procedure solves the puzzle.

```
hanoi(n,p_0,p_1,p_2)=
              hanoi(n-1,p_0,p_2,p_1)
              <d_n,p_0,p_1>
              hanoi(n-1,p_2,p_1,p_0)
```

where `hanoi(n,p,q,r)` takes the disks $d_1, \cdots d_n$ from peg p to peg q (using the auxiliary peg r) and `<d,p,q>` means move disk d from p to q.

If h_n is the number of disk moves in `hanoi(n,p,q,r)` then the recursion implies that

$$h_{n+1} = 2h_n + 1.$$

Since we also know that $h_1 = 1$ by induction on n we can prove that

$$h_n = 2^n - 1.$$

Graph of a puzzle

To a puzzle like the Hanoi towers we can associate an oriented graph Γ. The vertices of Γ are the legal positions or states of the puzzle. Two states s and t are joined by an edge $s \to t$ if one can pass from s to t by a legal move, i.e. by following the rules of the game. (If the puzzle is completely reversible as in the case of the Hanoi towers we can just put an edge without any orientation between pairs of states that can differ in a legal move.)

Solving the puzzle corresponds to navigating from the original state to the goal state (or *some* goal state if there is more than one) by moving through the edges of Γ. A bird's eye view of the graph would allow one to easily solve the puzzle.

Hanoing

In this puzzle we have a row of n squares colored randomly white, red or blue and the goal is to change them all to the same color (any one of three possible

would do). The rules are that a square can only change color if it is the first square of that color (reading form left to right) and it can only change to another color if this new color is also the first. For example, if the original state is $RRWB$ then we can only change it to $WRWB, BRWB$ or $RRBB$.

The following Tcl code is a version of the puzzle. Clicking on a square will cyclically change its color to the allowable ones.

```
#--------------------------------------------------------------------
# Hanoing puzzle.
# Fernando Rodriguez Villegas.
# Math, puzzles and computers, Spring 2003.
# University of Texas at Austin.
#
# In a unix shell this program runs with the command
# wish hanoing.tcl
#--------------------------------------------------------------------

# n is the number of buttons on the puzzle, set to 5 by default.

set n 5

# colors is the list of colors the buttons can have (in an arbitrary but fixed
# cyclic ordering)

set colors {"white" "red" "blue"}


# This bit of code creates the frame on which the buttons are going to
# be placed.

frame .f
grid .f -row 1 -column 0 -columnspan 8

#--------------------------------------------------------------------
# First we put in 3 buttons called e, r and d; e, contains the value
# of n, which can be changed by the user  (though at this point
# decreasing n doesn't work very well); r, is bound to the command
# Reset (defined below), which creates the colored buttons of the
# puzzle; and d, simply quits the whole thing.

entry .f.e -width 3 -textvariable n
button .f.r -text "reset" -command Reset
button .f.d -text "dismiss" -command "destroy ."

# We now place these buttons on the frame.

set i 0
```

```
foreach p {e r d} {
grid .f.$p -row 1 -column [expr 2*$i] -columnspan 2
incr i
}

#-------------------------------------------------------------------------
# The process Reset (re-)initializes the puzzle.

proc Reset {} {
global n colors

for {set i 1} {$i <= $n} {incr i} {

# We choose a random color from the list of colors.  The whole
# expression [expr {round(3*rand()-.5)}] simply returns a random
# choice of number 0, 1 or 2

    set color [lindex $colors [expr {round(3*rand()-.5)}]]

# This kills the $i-th button if it exists; we need it to redefine the
# button.

    if {[winfo exists .b$i]} {
        destroy .b$i
    }

# We create the $i-th button of the color we picked above (both when
# by itself and when the cursor is over it) and is bound to the
# command Do $i; i.e. when the user clicks this button the process Do
# runs with this value of $i.

button .b$i -height 2 -width 2  -background $color -activebackground $color -command "Do $i"
grid .b$i -row 0 -column [expr {$i - 1}]
}
}

#-------------------------------------------------------------------------
# Process Do that is bound to the colored buttons.

proc Do {i} {
global n colors

# The tree questions are encoded in the 0/1 variables test, test1 and
# test2 (in this order).

# First answer is yes unless we need to change it later.

set test 1
```

```
# ccolor is the current color of the $i-th button.

set ccolor  [.b$i cget -background]

# Check whether $j-th button with j < i is of same color as current
# one; if any is, change first answer test to 0. (Yes, we should
# use a while loop...)

for {set j 1} {$j < $i} {incr j} {
    if {[.b$j cget -background] == $ccolor} {
set test 0
}   }


# If first answer test is yes proceed further otherwise do nothing.

if {$test} {

# Set second answer test1 to 1.

    set test1 1

# Pick next color in the list and call it ncolor.

    set cindex [lsearch $colors $ccolor]
    set ncolor [lindex $colors [expr {($cindex+1)%3}]]

# Check whether $j-th button with j < i is of the same color as new
# color; if any is, change second answer test1 to 0. (Yes again, we
# should use a while loop...)

    for {set j 1} {$j < $i} {incr j} {
    if {[.b$j cget -background] == $ncolor} {
    set test1 0
}   }

# If second answer test1 is yes change color of button to new one
# otherwise proceed to third question.

if {$test1} {

# Change color to $i-th button.

    .b$i configure -background $ncolor -activebackground $ncolor

# Try third color.
```

```
} else {

# Third answer test2 is yes unless we change it later.

        set test2 1
# New color (again) called ncolor is 2 further down the list from current.

        set ncolor [lindex $colors [expr {($cindex+2)%3}]]

# Check whether $j-th button with j < i is of the same color as new
# color; if any is, change third answer test2 to 0. (Indeed, we
# should...)

    for {set j 1} {$j < $i} {incr j} {
    if {[.b$j cget -background] == $ncolor} {
        set test2 0
    }
}

# If third answer test2 is yes change color of button to new one
# otherwise we are done.

if {$test2} {
    .b$i configure -background $ncolor -activebackground $ncolor
}   }   }   }

#--------------------------------------------------------------------------
# Initialize puzzle.

Reset
```

We determine what colors a given square can change to as follows.

Q1: Is this square's color the first?

If NO then end, square cannot change color. If YES continue.

Q2: Is the next color in the list the first?

If YES change square to this color and end. If NO continue.

Q3: Is the color after the next in the list the first?

If YES change square to this color and end. If NO then end, square cannot change color.

Homework (due Tuesday Feb 11)

(1) (Noticed by two students in the class) Consider the sequence $1, 2, 1, \cdots$ given by the number of bits that change in the binary expansion of one term to the next in $0, 1, 2, \cdots$. What sequence is it? Can you prove it?

(2) Is there another way to solve the Hanoi towers puzzle with n disks in $2^n - 1$ moves other than the one above?
(3) Explain why there are 3^n possible states for the Hanoi towers puzzle with n disks.
(4) Let Γ_n be the graph of the Hanoi towers puzzle with n disks. Draw the graphs $\Gamma_1, \cdots, \Gamma_4$. Explain how to pass from Γ_n to Γ_{n+1}; why is Γ_n connected? (i.e. why is any possible configuration of disks stacked in the pegs in increasing order reachable from the original position by a sequence of legal moves?).
(5) Find a Hamiltonian circuit on $\Gamma_1, \cdots, \Gamma_4$. Do you see a way of describing a Hamiltonian circuit in any Γ_n?
(6) Explain in detail why the Hanoi towers and the Hanoing puzzle are isomorphic (i.e. have the same underlying structure).
(7) Solving the Hanoing puzzle for $n = 4$ optimally, what is the largest posible number of necessary moves that one might encounter? (N.B. this number depend of whether we solve the abstract puzzle or the one coded above since in the coded version we always move through colors in the same direction). Can you give the answer to this question for arbitrary n?
(8) Modify the Tcl code for the Hanoing puzzle above to model the Chinese rings puzzle. Suggestions: use squares to represent the rings with colors, say, black for on, white for off. Change the line

```
set color [lindex $colors [expr {round(3*rand()-.5)}]]
```

to

```
set color black
```

to initialize the puzzle with all rings on. Change the `Do i` procedure to one describing the rules of the Chinese rings puzzle.

Department of Mathematics University of Texas at Austin, TX 78712