MASTERMIND: LAST DETAILS

David Kauchak
CS52 – Spring 2017

Admin

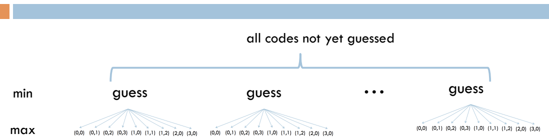
Assignment 7

Assignment 8

Midterm

Course registration

Key heuristic



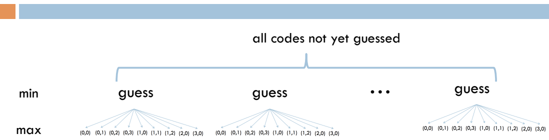
The one that minimizes the maximum remaining candidates

Max (codemaker response): assume we get the response with the largest remaining candidate set

Min (our guess): pick the one that, worst case, results in the smallest candidate set

How do we calculate this?

Key heuristic



The one that minimizes the maximum remaining candidates

min
For all codes not yet guessed:
max
Consider all possible responses:
Calculate the size of the remaining candidates if we guessed that code and got that response
select response with largest remaining for that code
select code with smallest max

Game tree

We can precompute the entire tree of possibilities

Expensive upfront to compute

Playing becomes fast

Game tree

[Red, Red, Green] ("best" first guess)

codemaker response	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	1	4	3	0	6	4	2	6	1

Game tree

[Red, Red, Green] ("best" first guess)

codemaker response	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	1	4	3	0	6	4	2	6	1

Recurse!

Game tree

Parent: [Red, Red, Green]

(0,0)

(26 guesses
1 candidate answer)

[Blue, Blue, Blue]

Game tree

Parent: [Red, Red, Green]

(0,0) (26 guesses 1 candidate answer)

[Blue, Blue, Blue]

codemaker response	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	0	0	0	0	0	0	0	0	1

What now?

Game tree

Parent: [Red, Red, Green]

(0,0) (26 guesses 1 candidate answer)

[Blue, Blue, Blue]

codemaker response	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	0	0	0	0	0	0	0	0	1
	Lose	Lose	Lose	Lose	Lose	Lose	Lose	Lose	Win

Use lose to indicate we don't have any options left (this shouldn't happen if we use a reasonable strategy)

Game tree

[Red, Red, Green] ("best" first guess)

codemaker response	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	1	4	3	0	6	4	2	6	1

Game tree

Parent: [Red, Red, Green]

(0,1) (26 guesses 4 candidate answers)

[Green, Blue, Blue]

Game tree

Parent: [Red, Red, Green]

(0,1) (26 guesses 4 candidate answers)

[Green, Blue, Blue]

	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	0	0	0	0	0	1	1	1	1

What now?

Game tree

Parent: [Red, Red, Green]

(0,1) (26 guesses 4 candidate answers)

[Green, Blue, Blue]

	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	0	0	0	0	0	1	1	1	1
	Lose	Lose	Lose	Lose	Lose		?		Win

Game tree

Parent: [Red, Red, Green]

(0,1) (26 guesses 4 candidate answers)

[Green, Blue, Blue]

	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	0	0	0	0	0	1	1	1	1
	Lose	Lose	Lose	Lose	Lose		Recurse		Win

Building the game tree

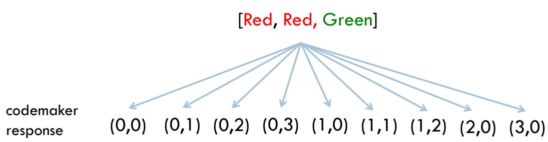
If 0 options then Lose

If 1 option and the response ***was*** (num_pegs, 0) then Win

Otherwise, build another Tree:

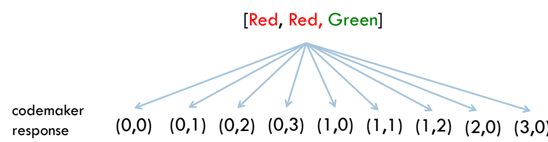
- Guess = one that minimizes the maximum remaining candidates over all responses
  - Break ties by 1) those that are still valid codes and 2) found first in candidate (valid) list
- Recurse on responses

Representing the game tree



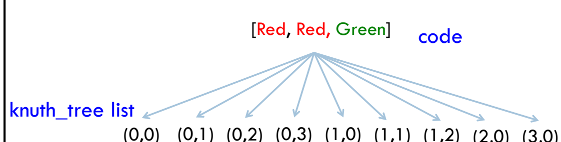
How do we store this tree?

Representing the game tree



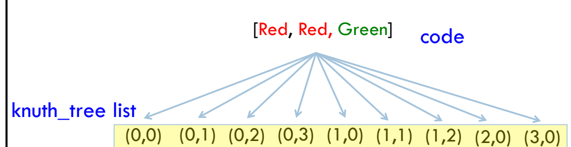

```
datatype knuth_tree = Lose
                    | Step of code * knuth_tree list
                    | Win;
```

Representing the game tree




```
datatype knuth_tree = Lose
                    | Step of code * knuth_tree list
                    | Win;
```

Representing the game tree



The responses aren't explicitly stored in the tree

There is an *implicit* ordering to the subtrees that correspond to these

```
datatype knuth_tree = Lose
                    | Step of code * knuth_tree list
                    | Win;
```

Representing the game tree

[Blue, Blue, Blue]

codemaker response	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	0	0	0	0	0	0	0	0	1
	Lose	Lose	Lose	Lose	Lose	Lose	Lose	Lose	Win

Write some SML to create this tree.

Representing the game tree

[Blue, Blue, Blue]

codemaker response	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	0	0	0	0	0	0	0	0	1
	Lose	Lose	Lose	Lose	Lose	Lose	Lose	Lose	Win

```
datatype knuth_tree = Lose
                    | Step of code * knuth_tree list
                    | Win;
```

Write some SML to create this tree.

Representing the game tree

[Blue, Blue, Blue]

codemaker response	(0,0)	(0,1)	(0,2)	(0,3)	(1,0)	(1,1)	(1,2)	(2,0)	(3,0)
candidates remaining	0	0	0	0	0	0	0	0	1
	Lose	Lose	Lose	Lose	Lose	Lose	Lose	Lose	Win

```
datatype knuth_tree = Lose
                    | Step of code * knuth_tree list
                    | Win;
```

```
Step ([Blue, Blue, Blue], [Lose, Lose, Lose, Lose, Lose, Lose, Lose, Lose, Win]);
```

A simple example

```
local
    fun first []       = raise InternalInconsistency
      | first (x::xs) = x;
in
fun badNextMove (Step (code, tree)) = (code, first tree)
  | badNextMove _                   = raise InternalInconsistency;
end
```

What is the type signature of this function?

What does it do?

A simple example

```
local
    fun first []       = raise InternalInconsistency
      | first (x::xs) = x;
in
fun badNextMove (Step (code, tree)) = (code, first tree)
  | badNextMove _                    = raise InternalInconsistency;
end
```

knuth_tree -> (code * knuth_tree)

Returns the next code and then always chooses the first element in the knuth tree (i.e. associated with response (0,0))

Midterm

SML
- datatypes (with non-zero constructors, recursive datatypes)
- mutual recursion
- handling exceptions

Binary numbers
- signed representation
- adding
- shifting

Parsing: EBNF grammars

Circuits
- general ideas (building circuits, truth tables, etc.)
- minterm expansion
- specific circuits (decoders, multiplexers)

Midterm

Encryption
- encryption/decryption
- modular arithmetic

Resources:
- We will provide you with the graphical pictures for the gates.
- Like the previous midterms, you may bring one single-sided, 8.5" x 11" piece of paper with notes.

Course registration