SP.268	Lecture 5	Mar. 8, 2011

Playing Games with Algorithms:

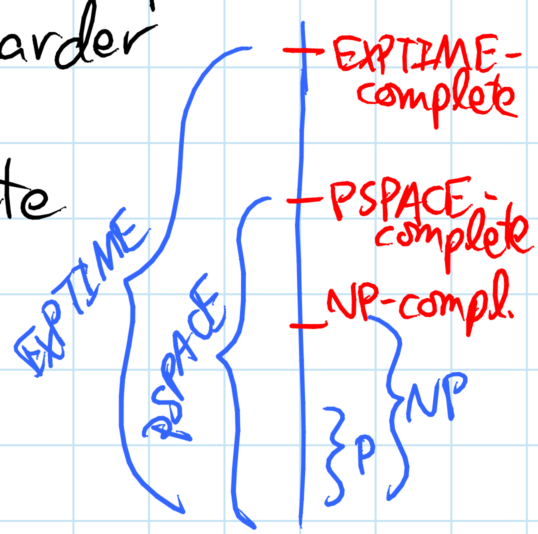
http://erikdemaine.org/papers/AlgGameTheory_GONC3/

- most games are hard to play well:
- Chess is EXP.TIME-complete:
  - $n \times n$ board, arbitrary position
  - <u>need</u> exponential (c^n) time to find a winning move (if there is one)
  - also: as hard as <u>all</u> games (problems) that need exponential time

- Checkers is EXPTIME-complete
  - ⇒ Chess & Checkers are the "same" computationally: solving one solves other
  (PSPACE-complete if draw after poly. moves)

- Shogi (Japanese Chess) is EXPTIME-complete
- Japanese Go is EXPTIME-complete
  - U.S. Go might be harder

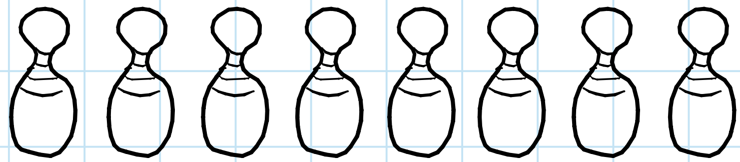
- Othello is PSPACE-complete
  - conjecture requires exponential time, but not sure (implied by $P \neq NP$)



- can solve some games fast: (mostly 1D)
  in "polynomial time" $\sim n^c$

<u>TODAY</u>: 1D impartial games via Dynamic Programming

<u>Kayles</u>: [Dudeney 1908]



$\longleftarrow n \longrightarrow$ bowling pins

- move = hit one or two adjacent pins
- last player to move wins (normal play)

$\boxed{\text{Let's play!}}$

<u>First-player win</u>: <u>SYMMETRY STRATEGY</u>
- move to split into two equal halves (1 pin if odd, 2 if even)
- whatever opponent does, do same in other half

$(K_n + K_n = 0$ ... just like Nim)

But what if we played the <u>sum</u> of 2 Kayles games?
or Kayles + Nim?

↳ can make a move in <u>either</u> game
~ still last player to move wins

Impartial game, so Sprague-Grundy Theory says Kayles $\equiv$ Nim somehow

one pin

- $\text{followers}(K_n) = \{K_i + K_{n-i-1} \mid i = 0, 1, \dots, n-1\} \cup \{K_i + K_{n-i-2} \mid i = 0, 1, \dots, n-2\}$

two pins

$$\Rightarrow g(K_n) = \text{mex}(g(\text{followers}(K_n)))$$
$$= \text{mex}\{g(K_i + K_{n-i-1}) \mid i = 0, 1, \dots, n-1\} \cup \{g(K_i + K_{n-i-2}) \mid i = 0, 1, \dots, n-2\}$$

Sprague-Grundy value / nimber

nim sum / binary XOR

- $g(X+Y) = g(x) \oplus g(y)$

$$\Rightarrow g(K_n) = \text{mex}\{g(K_i) \oplus g(K_{n-i-1}) \mid i = 0, 1, \dots, n-1\} \cup \{g(K_i) \oplus g(K_{n-i-2}) \mid i = 0, 1, \dots, n-2\}$$

RECURRENCE – write what you want in terms of smaller things

How do we compute it?

$g(K_0) = 0$ (BASE CASE)

$$g(K_1) = \text{mex}\{\underbrace{g(K_0) \oplus g(K_0)}_{0 \oplus 0 = 0}\}$$
$$= 1$$

$$g(K_2) = \text{mex}\{g(K_0) \oplus g(K_1),$$
$$0 \oplus 1 = 1$$
$$g(K_0) \oplus g(K_0)\}$$
$$0 \oplus 0 = 0$$
$$= 2$$

$\boxed{\text{so e.g.}}$ $K_2 + *2 = 0 \Rightarrow 2^{nd}$ player win

$$g(K_3) = \text{mex}\{g(K_0) \oplus g(K_2),$$
$$0 \oplus 2 = 2$$
$$g(K_0) \oplus g(K_1),$$
$$0 \oplus 1 = 1$$
$$g(K_1) \oplus g(K_1)\}$$
$$1 \oplus 1 = 0$$
$$= 3$$

$$g(K_4) = \text{mex}\{g(K_0) \oplus g(K_3),$$
$$0 \oplus 3 = 3$$
$$g(K_0) \oplus g(K_2),$$
$$0 \oplus 2 = 2$$
$$g(K_1) \oplus g(K_2),$$
$$1 \oplus 2 = 3$$
$$g(K_1) \oplus g(K_1)\}$$
$$1 \oplus 1 = 0$$
$$= 1$$

In general: if we compute
$g(K_0), g(K_1), g(K_2), \ldots$ in order,
then we always use values
that we've already computed
(because smaller)

- in Python, can do this with for loop:

```
k = {}
for n in range(0, 1000):
    k[n] = mex ([k[i] ^ k[n-i-1] for i in range(n)] +
                    [k[i] ^ k[n-i-2] for i in range(n-1)])
    print n, "-", k[n]

def mex(nimbers):
    nimbers = set(nimbers)
    n = 0
    while n in nimbers:
        n = n + 1
    return n
```

960 - 4	972 - 4	984 - 4
961 - 1	973 - 1	985 - 1
962 - 2	974 - 2	986 - 2
963 - 8	975 - 8	987 - 8
964 - 1	976 - 1	988 - 1
965 - 4	977 - 4	989 - 4
966 - 7	978 - 7	990 - 7
967 - 2	979 - 2	991 - 2
968 - 1	980 - 1	992 - 1
969 - 8	981 - 8	993 - 8
970 - 2	982 - 2	994 - 2
971 - 7	983 - 7	995 - 7

periodic mod 12!
(starting at 72)
[Guy & Smith 1972]

DYNAMIC PROGRAMMING

How fast? to compute $g(K_n)$:

- look up $\approx 4n$ previous nimbers
- compute $\approx 2n$ nimsums (XOR)
- compute one mex on $\approx 2n$ nimbers

- call all this $\boxed{O(n)}$ work "order n"

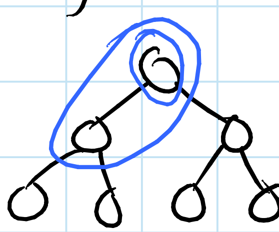
- need to do this for $n = 0, 1, \ldots, m$

$$\Rightarrow \sum_{n=0}^{m} O(n) = O\left(\sum_{n=0}^{m} n\right) = O\left(\frac{m(m+1)}{2}\right) = \underline{\underline{O(m^2)}}$$

POLYNOMIAL TIME — GOOD

Variations: dynamic programming also works for:

- Kayles on a cycle
  (1 move reduces to regular Kayles
  $\Rightarrow$ 2nd player win)
- Kayles on a tree:



target vertex or 2 adj. vertices

- Kayles with various ball sizes:
  hit 1 or 2 or 3 pins
  (still 1st player win)

also green Hackenbush trees

Cram: impartial Domineering
- board = $m \times n$ rectangle
  possibly with holes
- move = place a domino (make 1×2 hole)

Symmetry strategies: [Gardner 1986]
- even × even: reflect in both axes
  ⇒ 2^{ND} player win
- even × odd: play center 2 □s
  then reflect in both axes
  ⇒ 1^{ST} player win
- odd × odd: $\boxed{\text{OPEN}}$ who wins?

Linear Cram = $1 \times n$ cram
- easy with dynamic programming
- also periodic [Guy & Smith 1956]

- 1×3 blocks still easy with DP
- $\boxed{\text{OPEN}}$: periodic?

Horizontal Cram: $\boxed{\ \mid\ }$ only
⇒ sum of linear crams!

$2\times n$ Cram: Nimbers $\boxed{\text{OPEN}}$
$3\times n$ Cram: winner $\boxed{\text{OPEN}}$
(dynamic programming doesn't work)

$\boxed{\text{Let's play!}}$