Playing Games with Algorithms:

- most games are hard to play well:
- Chess is EXPTIME-complete:
  - $n \times n$ board, arbitrary position
  - need exponential ($c^n$) time to find a winning move (if there is one)
  - also: as hard as all games (problems) that need exponential time
- Checkers is EXPTIME-complete:
  $\Rightarrow$ Chess & Checkers are the “same” computationally: solving one solves the 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: in “polynomial time” (mostly 1D)

Kayles:

![Bowling pins](n bowling pins)

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

Let’s play!
First-player win: SYMMETRY STRATEGY

- move to split into two equal halves (1 pin if odd, 2 if even)
- whatever opponent does, do same in other half
  \( K_n + K_n = 0 \ldots \) just like Nim

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

- followers(\( K_n \)) = \( \{ K_i + K_{n-i-1}, K_i + K_{n-i-2} \mid i = 0, 1, \ldots, n - 2 \} \)
  \( \Rightarrow \) \( \text{nimber}(K_n) = \text{mex}\{ \text{nimber}(K_i + K_{n-i-1}), \text{nimber}(K_i + K_{n-i-2}) \mid i = 0, 1, \ldots, n - 2 \} \)

- \( \text{nimber}(x + y) = \text{nimber}(x) \oplus \text{nimber}(y) \)
  \( \Rightarrow \) \( \text{nimber}(K_n) = \text{mex}\{ \text{nimber}(K_i) \oplus \text{nimber}(K_{n-i-1}), \text{nimber}(K_i) \oplus \text{nimber}(K_{n-i-2}) \mid i = 0, 1, \ldots, n - 2 \} \)

RECURRENCE! — write what you want in terms of smaller things

How do we compute it? (BASE CASE)

\[
\begin{align*}
\text{nimber}(K_0) &= 0 \\
\text{nimber}(K_1) &= \text{mex}\{ \text{nimber}(K_0) \oplus \text{nimber}(K_0) \} \\
&= \text{mex}\{ 0 \oplus 0 \} \\
&= 1 \\
\text{nimber}(K_2) &= \text{mex}\{ \text{nimber}(K_0) \oplus \text{nimber}(K_1), \text{nimber}(K_0) \oplus \text{nimber}(K_0) \} \\
&= \text{mex}\{ 0 \oplus 1, 0 \oplus 0 \} \\
&= 2 \\
\text{nimber}(K_3) &= \text{mex}\{ \text{nimber}(K_0) \oplus \text{nimber}(K_2), \text{nimber}(K_0) \oplus \text{nimber}(K_1), \text{nimber}(K_1) \oplus \text{nimber}(K_1) \} \\
&= \text{mex}\{ 0 \oplus 2, 0 \oplus 1, 1 \oplus 1 \} \\
&= 3 
\end{align*}
\]
\[
\text{nimmer}(K_4) = \text{mex}\{\text{nimmer}(K_0) \oplus \text{nimmer}(K_3), \text{nimmer}(K_0) \oplus \text{nimmer}(K_2), \text{nimmer}(K_1) \oplus \text{nimmer}(K_2), \text{nimmer}(K_1) \oplus \text{nimmer}(K_1)\}
\]
\[
= 1
\]

In general: if we compute \(\text{nimmer}(K_0), \text{nimmer}(K_1), \text{nimmer}(K_2), \ldots\) in order, then we always use nimmers that we’ve already computed (because smaller)

– in Python, can do this with for loop:

```python
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]
```

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

DYNAMIC PROGRAMMING

How fast? to compute \(\text{nimmer}(K_n)\):

– look up \(\approx 4n\) previous nimers
– compute \(\approx 2n\) nimsums (XOR)
– compute one mex on \(\approx 2n\) nimers
– call all this \(O(n)\) work “order \(n\)”
– need to do this for \(n = 0, 1, \ldots, m\)

\[
\sum_{n=0}^{m} O(n) = O\left(\sum_{n=0}^{m} n\right) = O\left(\frac{m(m+1)}{2}\right) = O(n^2)
\]

POLYNOMIAL TIME — GOOD
Variations: dynamic programming also works for:

- Kayles on a cycle
  (1 move reduces to regular Kayles ⇒ 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)

Cram: impartial Domineering

- board = $m \times n$ rectangle, possibly with holes
- move = place a domino (make $1 \times 2$ hole)

Symmetry strategies: [Gardner 1986]

- even × even: reflect in both axes
  ⇒ 1st player win
- even × odd: play 2 center □s then reflect in both axes
  ⇒ 1st player win
- odd × odd: OPEN who wins?

Liner Cram = $1 \times n$ cram

- easy with dynamic programming
- also periodic [Guy & Smith 1956]

- $1 \times 3$ blocks still easy with DP
- OPEN: periodic?

Horizontal Cram: $1$ only

⇒ sum of linear crams!

2 × $n$ Cram: Nimbers OPEN Let’s play!
3 × $n$ Cram: winner OPEN

(dynamic programming doesn’t work)