**MIT ES.S20 Lecture 4: Dynamic Programming and Algorithms for Gameplay**

## Lecture Outline

*based on Erik Demaine's 2011 Notes*

1. Hard Games and Easy Games
2. Kayles
3. Dynamic Programming Strategy for Kayles
4. Subproblem Structure
5. Cram

## Hard Games and Easy Games

- Heuristic Structure and Conceptualization

  Why is the Rubik's Cube difficult enough that we basically have to limit ourselves to algebraic manipulations?

  In most games, you can construct heuristics that make you perform *better*, but to actually be able to predict what's going to happen is much more complicated (we need the large tree structure). The Rubik's Cube is hard for us to play because we can't really come up with good heuristics.

- Algorithmic Complexity

  Games whose structures *look like* trees will (unless they obey some higher-level structure) take an amount of time exponential in the number of moves (the complexity class EXPTIME) to predict.

  Some games, however, or subproblems of games can be solved in an amount of time that's a normal polynomial of the number of moves (PSPACE), and that's the point of *dynamic programming*: we want to make fast solutions for games with simple structure.

EXPTIME and PSPACE are some of the essential *complexity classes* for you non-CS people here, which characterize how hard problems are. If we don't lecture on it, you can take 6.045/18.404 to learn more (!)

- Algorithms

  We're going to be in computer science-land for a little while here because it's very easy to think of these simple games in terms of code. The important thing to understand here is that this is *not* an approach just for programming; what we're really looking at here is actually the *structure of the problem.*

## Kayles

- Game Setup

  Say that we have n bowling pins lined up in a row. A move hits one or two adjacent pins. The last player to move wins.

  The *symmetry* strategy (which gives a win to the first player) says that whatever our opponent does, we reflect it about the x-axis and do that too. Our first move removes 1-2 pins to split it into even halves.

- Impartial Game and Relation to Nim

  Sprague-Grundy implies that we can represent Kayles as an impartial game. If we want to define our "nimber" strategy-mapping, we can define it in terms of building up on the pins:

```
g :: last location of pin -> Sprague-Grundy Label

g(n) = mex{ g(i) + g(n-i-1) | i=0,1...n-1,
            g(i) + g(n-i-2) | i=0,1...n-2 }

    because our opponent can remove one or two pins at
    the ith spot, our success is equivalent to the success
    of the combination of those games.

g(0) = 0
    # Base Case
```

```
g(1) = mex{(g(0) + g(0))}
     = 1
    # We can have only go to 0

g(2) = mex{g(1) + g(0), g(0) + g(0)}
     = 2
    # We can go to 10 or 00

g(3) = mex{g(0) + g(2), g(0) + g(1), g(1) + g(1)}
     = 3
    # We can go to 001, 010, or 101

g(4) = mex{g(0) + g(3), g(0) + g(2), g(1) + g(2),
           g(1) + g(1), g(2) + g(1), g(2) + g(0)}
     = 1
    # We can go to 0001, 0010, 1011, or 1001
```

- Computing our Kaylumbers

  Because g(n) only depends on some set of g(i) for i < n, we get to build up our solution from what we've already seen.

```
k = dict()

def mex(nbr):
    s = set(nbr)
    for n in range(MAX_N):
        if(not (n in s)): return n

for n in range(0, MAX_N):
    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)])
```

## Dynamic Programming

- Optimal Substructure

  Bottom-Up Dynamic programming works in situations where our problems depend only on a well-defined subset of problems we've seen before. If we can recognize and interact with them efficiently, then we can make a DP solution.

- Time Complexity

  Note that for each time we want to compute a new n, we have to do at most 2n steps. This gives O(n^2) complexity and is very workable for large n.

- Variations That Work

  Kayles on a ring of pins: after the first move, it's just the same and the second player wins.

  Kayles on a tree: if we can remove 1-2 vertices. Note that this looks a lot like Green Hackenbush

  Kayles with more ball sizes: we can generalize this to different ways of taking away pins

  Class Exercise: How do we solve these and who wins?

## Cram

- Game Rules

  We have a `m x n` rectangle which can have holes (!) on which we can place `1 x 2` dominoes. The last player to move wins.

- Strategies

  On `even x even` boards, we can reflect in both axes and the second player will win

  On `odd x even` boards, we can play two center dominoes and then reflect: the first player will win.

  `odd x odd` boards are an open problem.

- Variants

  In Linear Cram, the board is 1xn (Dynamic Programming!) and periodic. 1x3 blocks can still work with DP, but whether or not it's periodic is open.

  `2 x n` and `3 x n` Cram are both open problems (without optimal substructure).