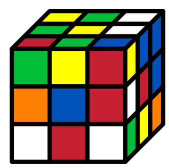
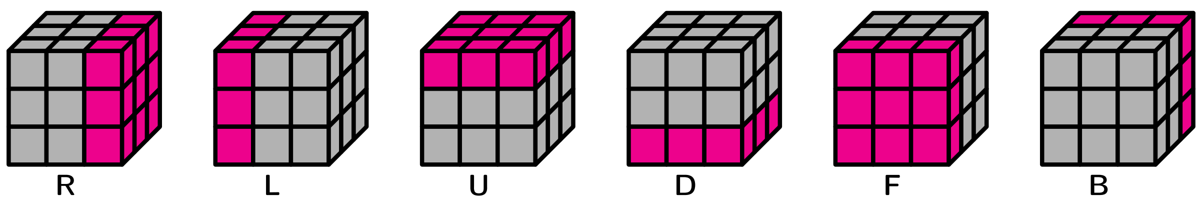
Lesson 7. Big DPs and the Curse of Dimensionality

1 Solving a Rubik's cube

- In a classic Rubik's cube, each of the 6 faces is covered by 9 stickers
- Each sticker can be one of 6 colors: white, red, blue, orange, green and yellow



- Each face of the cube can be turned independently
  - Notation:



  - The letter means turn the face clockwise 90°
    - For example, R means turn the right face clockwise 90°
  - The letter primed means turn the face counter-clockwise 90°
    - For example, R′ means turn the right face counter-clockwise 90°
- The problem: given an initial configuration of the cube, find a *shortest* sequence of turns so that each face has only one color
  - You may assume that you are allowed at most T turns
  - It turns out that any configuration can be solved in 26 turns or less: `http://cube20.org/qtm/`
- How can we formulate this problem as a dynamic program?

Let $1, \dots, N$ be a list of all the possible cube configurations.
initial — solved

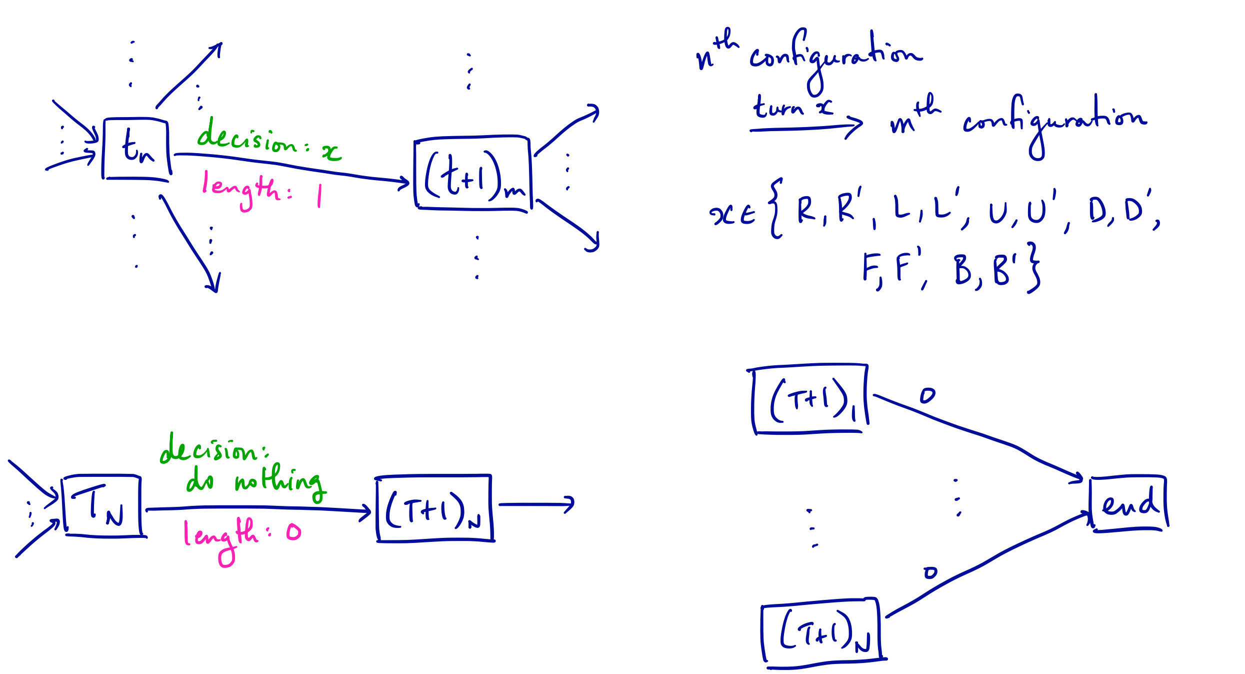
- Stages:

Stage t represents the t^{th} turn of the cube $(t = 1, \dots, T)$ or the end of the decision-making process $(t = T+1)$

- States in stage t (nodes):

Node t_n represents being in the n^{th} configuration with turns $t, t+1, \dots, T$ remaining $(n = 1, \dots, N)$

- Decisions, transitions, and rewards/costs at stage t (edges):



- Source node: 1_1 (initial config @ stage 1)

Sink node: end

- Shortest/longest path? Shortest

- Minimum number of turns required to solve the cube:

= Length of shortest path

- Actual sequence of turns that give the minimum number of turns to solve the cube:

Edges in the shortest path correspond to which turns to make.

2 Tetris

- You've all played Tetris before, right? Just in case...
- Tetris is a video game in which pieces fall down a 2D playing field, like this:

- Each piece is made up of four equally-sized bricks, and the playing field is 10 bricks wide and 20 bricks high
- As the pieces fall, the player can rotate them 90° in either direction, or move them left and right
- When a row is constructed without any holes, the player receives a point and the corresponding row is cleared
- The game is over once the height of bricks exceeds 20
- The problem: given a predetermined sequence of T pieces[1], determine how to place each piece in order to maximize the number of points accumulated over the course of the game
- How can we formulate this problem as a dynamic program?

[1]Normally, the sequence of falling pieces is random and infinitely long. We'll consider this easier version here.

Let 1, …, N be a list of all possible playing fields
(1 = empty, N = full)

- Stages:

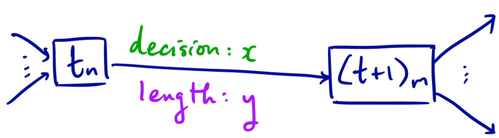
Stage t represents playing the t^{th} piece ($t = 1, \dots, T$), or the end of the decision-making process ($t = T+1$)

- States in stage t (nodes):

Node t_n represents being in the n^{th} playing field with pieces $t, \dots, T$ remaining ($n = 1, \dots, N$)

- Decisions, transitions, and rewards/costs at stage t (edges):

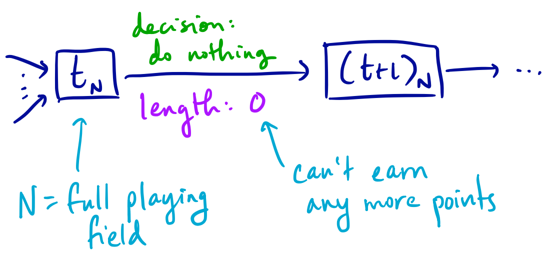
$n \neq N$:

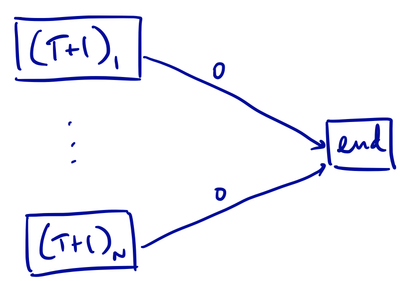


n^{th} playing field $\xrightarrow{\text{placement } x}$ m^{th} playing field

$x \in$ set of all possible piece placements

$$y = \begin{cases} 1 & \text{if line is cleared w/ placement } x \text{ on playing field } n \\ 0 & \text{o/w} \end{cases}$$





- Source node: 1_1 (empty field @ stage 1) Sink node: end

- Shortest/longest path? Longest

- Maximum number of points:

= length of longest path

- Actual placement of pieces that give the maximum number of points:

Edges in the longest path correspond to which placements to make.

3 Big DPs and the curse of dimensionality

- How big are these DPs we just formulated?
- Tetris:
  - Number of states per stage: $N = 2^{200} \approx 1.61 \times 10^{60}$
  - Number of stages T

  $\Rightarrow$ Number of nodes: $N(T+1) + 1 \approx (1.61 \times 10^{60})(T+1) + 1$
- Rubik's cube:
  - Number of states per stage: $N \approx 4.33 \times 10^{19}$
  - Number of stages T

  $\Rightarrow$ Number of nodes: $N(T+1) + 1 \approx (4.33 \times 10^{19})(T+1) + 1$
- The number of states is huge for both these DPs!

$\Rightarrow$ The DPs we formulated (as-is) are not solvable using today's computing power

- This is known as **the curse of dimensionality** in dynamic programming
- **Approximate dynamic programming** is an active area of research that tries to address the curse of dimensionality in various ways
  - For example, for Tetris: https://papers.nips.cc/paper/5190-approximate-dynamic-programming-finally-performs-well-in-the-game-of-tetris.pdf