• Homework 8 due tomorrow **at 5pm**
• Homework 9 released at around the same time
• Final Exam:
  • Thursday, May 14 from 4pm to 5:50pm in WWH 101
  • Must bring a Laptop! (Email me *asap* if you do not have one!)
• **Expectation maximization**: determine the expected value of the optimal move

• **Minimax**: assume two players taking turns
  • Compute using recursion/memoization on game state assuming each player makes a move which maximizes their score / chances of winning

• **Working backwards**: solve a game for a simple end case and work backwards using induction

• **Nim games**: class of games with two players taking turns removing objects from distinct heaps
  • Player who can no longer take anything loses
Exercise: Ten pirates (numbered 1-10) have just found 100 gold pieces. The lowest number pirate suggests an allocation of the gold pieces. If more than half of the pirates disagree, they throw the lowest numbered pirate overboard and repeat the process. Assuming you are pirate 1, what is the maximum amount of gold you can receive without swimming? All pirates are rational and want to maximize their gold. They will disagree unless it strictly benefits them.
1. If there is 1 pirate, he gets all 100 pieces.
2. If there are 2 pirates, number 1 gets all 100, as even if pirate 2 disagrees, he will never be more than half the pirates.
3. If there are 3 pirates, number 1 gets 99 and gives 1 gold piece to pirate 3.
4. If there are 4 pirates, number 1 gets 99 and gives 1 gold to pirate 3.
5. If there are 5 pirates, numbers 1 gets 98 and gives 1 gold to pirates 3 and 5.
6. In general, if there are n pirates, 3, 5, 7, etc. all get 1 gold, and pirate 1 takes the rest. So for 10 pirates, pirate 1 gets 96 gold.
**Exercise:** You are playing a game in which four fair coins are flipped and the amount of money you receive in dollars is equal to the number of heads that appear in total. If you do not like the outcome of the first four flips, you have the option to re-flip the four coins, but you are obligated to take the second outcome. Determine the expected value of this game.
Exercise: You are playing a game where the player gets to draw the number 1-100 out of hat, replace and redraw as many times as they want, with their final number being how many dollars they win from the game. Each redraw costs one dollar. How much would you charge someone to play this game?
• Let $X$ be the threshold below which you will redraw, and at or above which you will stop
• Expected value $E$:

$$E = -1 + \frac{X-1}{100} E + \sum_{t=X}^{100} t$$

• Maximize $E$
  • $X = 87$, $E \approx 86$
  • Charge more than $E$
Exercise: You have 10 piles of stones, the $i^{th}$ pile holds $s_i$ stones ($1 \leq s_i \leq 10^9$). Each turn you or your opponent (turns alternate) can remove any number of stones from one of the piles. The first player that cannot move loses. Give an algorithm to determine if you can win with optimal play.
• Convert the size of each pile into its binary representation
• XOR the sizes (“Nim-sum”)  
  • Winning position: XOR = 0, Losing position: XOR > 0
  • You can always move from losing to winning  
    • Zero any leading 1’s, and add 1 to any less 1’s  
    • But you can never move from winning to winning  
      • All columns that are all 0 will not benefit by adding a 1  
      • All columns that have even 1’s will become 1 by removing a 1
• Strategy: you want to start in a losing position and move into a winning position
• Example: try 3 piles of sizes 4, 5, 7
• How would you search for a string P inside a string T?
• Naïve approach: two for-loops
  • $O(|P| \cdot |T|)$ time
  • Note: searching natural language text, this isn’t too bad because usually the inner loop breaks early, but not good for generic strings (e.g., genetic code)
• Better approach: instead of comparing each substring of T against P, compare a hash of the substring against a hash of P
  • Efficiency gained through fewer equality comparisons
  • You can also make it faster through using a rolling hash function
  • Risk of hash collisions
  • Worst case running time is still $O(|P| \cdot |T|)$
• Can we do better?
A deterministic finite automaton is a directed graph where each node has edges coming out of it labeled with the letters from a fixed finite alphabet $\Sigma$

- One node is designated as the start node, and a set of nodes are designated as final nodes
- If $|\Sigma| = k$, then each node has $k$ distinctly labeled outgoing edges, one for each letter in $\Sigma$
- The strings accepted by a DFA are the ones corresponding to walks from the start state to a final state following the correctly labeled edges.
Exercise: Give a DFA for $\Sigma = \{0, 1\}$ and strings that have an odd number of 1’s and any number of 0’s
Exercise: Give a DFA for $\Sigma = \{0, 1\}$ and strings that have an odd number of 1’s and any number of 0’s
**Exercise:** Give a DFA for $\Sigma = \{a, b, c\}$ that accepts any string with aab as a substring.
Exercise: Give a DFA for $\Sigma = \{a, b, c\}$ that accepts any string with $aab$ as a substring.
Exercise: Give a DFA for $\Sigma = \{a, b\}$ that accepts any string with aababb as a substring
Exercise: Give a DFA for $\Sigma = \{a, b\}$ that accepts any string with $aababb$ as a substring
Exercise: Suppose you are given many texts (strings) $T_1, \ldots, T_n$ and one pattern string $P$. You want to determine which texts have the pattern $P$ as a substring. What is the total runtime of doing this using DFAs?
• String matching with a DFA is very fast
• How do you build the DFA?
• Knuth-Morris-Pratt
  • Quickly builds the DFA
  • Compactly stores the DFA
  • Simulates the DFA

• Given a pattern P and a string T
  • The array can be built in $O(|P|)$ time
  • The string can be matched in $O(|T|)$ time
• General idea: build an array where each entry corresponds to a DFA state
• Recall from previous exercises that each state of the DFA corresponds to the “progress” we’ve made in pattern matching
  • Suppose each entry in the array corresponds to a prefix z of the pattern P
• When we get a letter that doesn’t make progress, which state do we return to?
• The array entry will refer to the index of the longest pattern prefix which is a proper suffix of z
**Example: For pattern ABABAC**

<table>
<thead>
<tr>
<th>Index</th>
<th>Prefix z</th>
<th>Longest proper suffix that is also a prefix of z</th>
<th>Refers to…</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td>∅</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>∅</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>AB</td>
<td>∅</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>ABA</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>ABAB</td>
<td>AB</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>ABABA</td>
<td>ABA</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>ABABAC</td>
<td>∅</td>
<td>0</td>
</tr>
</tbody>
</table>
Competitive Programming 5.8, 6.1—6.4