# Beginner

**Target Audience:** students new to competitive programming

* Focus on basic control structures, loops, conditionals, and simple data structures
* Topics: implementation, basic math, string manipulation, easy simulations

## Problems:

1. “Hello World!” [Link](https://open.kattis.com/problems/hello)
2. “R2” [Link](https://open.kattis.com/problems/r2)
   1. Hints: solve the equation for desired variable
3. “Digits” [Link](https://open.kattis.com/problems/digits)
   1. Consider a base case (1)
4. “Cold-puter Science” [Link](https://open.kattis.com/problems/cold)
   1. Remember that there are at most 100 temps given
5. “Oddities” [Link](https://open.kattis.com/problems/oddities)
   1. Modulus operation
6. “Provinces and Gold” [Link](https://open.kattis.com/problems/provincesandgold)
   1. Calculate the total buying power
   2. Being able to purchase one thing doesn’t change the total buying power at all

# Intermediate

**Target Audience:** students with some experience in competitive programming or basic algorithms

* Topics include sorting, binary search, greedy algorithms, and dynamic programming

## Problems:

1. “Speed Limit” [Link](https://open.kattis.com/problems/speedlimit)
   1. The total time recorded is cumulative
   2. Keep track of a total and increase it for each segment
2. “Statistics” [Link](https://open.kattis.com/problems/statistics)
   1. Use builtin functions to calculate the min and max
3. “Apaxiaaans!” [Link](https://open.kattis.com/problems/apaxiaaans)
   1. Only a single pass is required
   2. Keep track of another string for the new name
4. “Jumbled Compass” [Link](https://open.kattis.com/problems/compass)
   1. Make sure to normalize the angle (keep it between -180 and 180)
   2. Make sure to handle the edge case of a diametrically opposed angle
5. “Autori” [Link](https://open.kattis.com/problems/autori)
   1. Remember that the first letter of the answer will always be the first letter of the input
   2. Check for the delimiting character (hyphen)
6. “Birds on a Wire” [Link](https://open.kattis.com/problems/birds)
   1. Make sure to handle the case of an empty wire properly
   2. Formula for calculating how many birds can fit in a gap: (max-min)//d

# Intermediate Advanced

**Target Audience:** students who have experience in algorithms and have experience with atypical implementations.

## Problems:

1. “Two Charts Become One” [Link](https://open.kattis.com/problems/twochartsbecomeone)
   1. Use a stack or recursion as you parse the chart
   2. Use a dictionary (map) to represent the hierarchy, where each key is a department and it’s value is a set of its sub-departments
   3. Check if the root department numbers are the same for both charts and compare the sub-departments (may require sorting)
2. “Travel the Skies” [Link](https://open.kattis.com/problems/traveltheskies)
   1. Use a graph to represent flights and a dictionary to store the number of customers at each airport on each day.
   2. Simulate the flight days (iterate through and update values as needed)
   3. Consider edge cases (flights overfilled or customers have no way to get to their destination)

# Advanced

**Target Audience:** students who are proficient in data structures and algorithms

* Topics include more complex dynamic programming, graph theory, combinatorics and number theory

## Problems:

1. “Abridged Reading” [Link](https://open.kattis.com/problems/abridgedreading)
   1. Solution in C++
   2. Maintain an array to store the number of pages for each chapter
   3. Recognize that the hierarchy of chapters for a directed acyclic graph with prerequisites defining the edges
   4. Track all chapters with no incoming edges (in-degree of 0) as culminating chapters
   5. Use DFS to propagate the costs down the hierarchy from each chapter to its prerequisites
   6. Use Lowest Common Ancestor when checking pairs of culminating concept chapters, calculate the pages required using the LCA to avoid double counting shared prerequisites
2. “Knapsack” [Link](https://open.kattis.com/problems/knapsack)
   1. Solution in C++
   2. Create a 2D DP array where rows represent items and columns represent capacities
   3. For each item, check if it can fit in the current capacity. If it can, decide whether to include it based on the maximum value achievable
   4. Remember to backtrack after filling the table
3. “Single Source Shortest Path, Non-Negative Weights” [Link](https://open.kattis.com/problems/shortestpath1)
   1. Simple implementation of known algorithm (Dijkstra’s)
4. “Unique Snowflakes” [Link](https://open.kattis.com/problems/snowflakes)
   1. Sliding window approach
   2. Use a set to keep track of the current unique snowflakes in the window and use a double ended queue to store the order of the snowflakes as they are processed
   3. When a duplicate is encountered, shift the start of the window until the duplicate is removed
   4. Every time you add a unique snowflake to the window, check if the current length of the window is greater than the previously recorded maximum length
5. “A Classy Problem” [Link](https://open.kattis.com/problems/classy)
   1. Map each class to a value
   2. Prioritize the classes that come last in the description as more important
   3. Make sure to pad with neutral values to make each of the class distinctions the same length
   4. Make sure to sort first by the numerical value, then alphabetically if there is a tie

# Expert

**Target Audience:** students with a deep knowledge of algorithms who are frequent competitors of programming competitions

## Problems:

1. “Frogger” [Link](https://open.kattis.com/problems/frogger)
   1. Create a class to represent a state of the board and the number of turns taken
   2. Use BFS to explore all possible states (frog positions) and keep track of the distances
   3. Maintain a list to track visited states (make sure this is easily indexed, otherwise you will TLE later on)