Basic Dynamic and Recursive Programming Problems:

1) Check out: Optimal Strategy for a Game - GeeksforGeeks.
2) In general UIL seems to recently take problems from this site: https://www.geeksforgeeks.org/explore?page=3&category=Java&sortBy=submissions&itm_source=geeksforgeeks&itm_medium=main_header&itm_campaign=practice_header
3)

Problem #1

1. **Optimal Strategy for a Game**
2. Given an array **arr[]** of size **n** which represents a **row of n coins** of values **V1 . . . Vn**, where **n is even**. We play a game against an opponent by alternating turns. In each turn, a player selects either the **first or last coin** from the row, removes it from the row permanently, and receives the value of the coin. Determine the **maximum possible** amount of money we can **definitely win** if we move first.
3. **Note:** The opponent is as clever as the user.
4. **Examples:**
5. ***Input:*** *arr[] = [5, 3, 7, 10]*
   Output: *15 -> (10 + 5)*
   Explanation: *The user collects the maximum value as 15(10 + 5). It is guaranteed that we cannot get more than 15 by any possible moves.*
6. ***Input:*** *arr[] = [8, 15, 3, 7]*
   Output: *22 -> (7 + 15)*
   Explanation: *The user collects the maximum value as 22(7 + 15). It is guaranteed that we cannot get more than 22 by any possible moves.*

   WHS Notes:
   a) **This was in a recent UIL Competition**
   b) **The natural greedy algorithm does not work in all cases**
   c) **Hints, probably solve this problem recursively. If the code times out, look at some kind of dp or memorization.**

Problem #2:

Count All Palindromic Subsequence in a given String

Given a string **s** of length **n**, the task is to count number of palindromic subsequence (need not necessarily be distinct) present in the string **s**.

Example:

Input: *s = "abcd"*
Output: *4*
Explanation: *Palindromic subsequence are : "a" ,"b", "c" ,"d"*
Input: *s = "aab"*
Output: *4*
Explanation: *palindromic subsequence are :"a", "a", "b", "aa"*
Input: *s = "geeksforgeeks"*
Output: *81*

Notes: This was a District 2025 problem which we timed out on. I am providing a naïve recursive solution which would time out with the judge's data, and one which would work with memorization.

Problem #3 Knap Sack Problem

Heidi

Program Name: Heidi.java Input File: heidi.dat

Heidi has begun thieving. She needs your help determining what she can fit in her bag when she robs a place. Her bag will have a maximum capacity, and each object will have a weight and a value on the black market. You need to determine what the maximum value she can obtain is, and which objects can be used to get that value.

Input: Input will begin with an integer, num (0 < num <= 100), denoting the number of test cases to follow. Each test case will begin with two space separated integers, n (0 < n <= 100) and c (0 < c <= 1000), denoting the number of items, and maximum weight Heidi can carry in her bag. The following line will contain n integers, denoting the values of each item Heidi can steal. The line after will contain n integers denoting the weights of each item. It can be assumed that the ith value corresponds with the ith weight. All weights and values will be strictly positive.

Output: Output the maximum value Heidi can obtain, followed by a colon and a space. Followed by a list of items Heidi chose in numerical order (they will be 1-indexed, so the first item in the list is 1, and the last is n). If there are no items small enough to fit in the pack, output “0: None”. If there are multiple combinations with the same weight and value, select the option that contains individual objects with the largest values, even if there are less of them.

Sample input:

```
3
3 50
60 100 120
10 20 30
4 60
40 100 50 60
20 10 40 30
5 100
10 25 30 40 50
30 40 50 60 70
```

Sample output:

```
220: 2 3
200: 1 2 4
65: 2 4
```

This is a classic CS problem. If you have not done this before, try solving first just for the largest values first, i.e. 220,200, 65 above.

Problem #4:

Coin Change - Count Ways to Make Sum

Given an integer array of **coins[]** of **size n** representing different types of **denominations** and an integer **sum**, the task is to count all combinations of coins to make a given value **sum**.

Note: Assume that you have an **infinite** supply of each type of coin.

Examples:

Input: *sum = 4, coins[] = [1, 2, 3]*
Output: *4*
Explanation: *There are four solutions: [1, 1, 1, 1], [1, 1, 2], [2, 2] and [1, 3]*

Input: *sum = 10, coins[] = [2, 5, 3, 6]*
Output: *5*
Explanation: *There are five solutions:*
[2, 2, 2, 2, 2], [2, 2, 3, 3], [2, 2, 6], [2, 3, 5] and [5, 5]

Input: *sum = 10, coins[] = [10]*
Output: *1*
Explanation: *The only is to pick 1 coin of value 10.*

Input: *sum = 5, coins[] = [4]*
Output: *0*
Explanation: *We cannot make sum 5 with the given coins*

Problem #5 Guillermo

Program Name: Guillermo.java Input File: guillermo.dat

You just started your new internship, and your boss Guillermo has asked you to schedule several 1 hour meetings with the team on different days. The problem is that this is a remote team and so your coworkers work all over the globe. And, being an intern and all, you feel as though if you can't schedule these meetings your return offer is as good as gone and you'll have to work at McDonalds. The time zones of each of your coworkers vary from UTC −12:00 to UTC +14:00. A person's availability may span multiple time windows throughout the day and may even wrap past midnight.

Input: The first line contains an integer N (1 ≤ N ≤ 50) denoting the number of days your boss wants a meeting scheduled. Each day starts with a single integer P (1 ≤ P ≤ 50) - the number of people. Each person's data will be given as a JSON object (see example input) describing their timezone offset in the format ±HH:MM and their availability throughout the day.

Output: For each day, output the lexicographically smallest 1 hour meeting time in the format HH:MM − HH:MM in UTC time zone. If no such slot exists, output "The fries are in the bag." to prepare for your new job since you won't be getting a return offer.

Sample input:

```
1
3
{
"timezone": "+02:00",
"availability": [("09:00", "12:00"), ("14:00", "18:00")]
},
{
"timezone": "-04:00",
"availability": [("08:00", "11:00"), ("13:00", "16:00")]
},
{
"timezone": "+00:00",
"availability": [("13:00", "17:00")]
}
```

Sample output:

```
13:00-14:00
```

Explanation:

Coworker1 (+02:00)

09:00-12:00 local -> 07:00-10:00 UTC
14:00-18:00 local -> 12:00-16:00 UTC

Coworker2 (-04:00)
08:00–11:00 local → 12:00–15:00 UTC
13:00–16:00 local → 17:00–20:00 UTC

Coworker3 (+00:00, already UTC)
13:00-17:00

The earliest 1 hour overlap of these 5 time windows is 13:00-14:00.

Problem #6 Amelia

Program Name: Amelia.java Input File: amelia.dat

Amelia is now a TA for ECS 1100 at her college program. She wants your assistance creating a sorting program to determine who the best students are, so that we can find out who deserves extra credit. The sorting algorithm is outlined as follows:

- First, sort the students based on attendance percentage (classes attended/total classes), as the students who go to all the classes deserve the best treatment (the professor's words, not ours), with the higher attendance percentage coming first.
- Next, if attendance is the same between two students, sort by improvement average between exam 1 to exam 2, and exam 2 to exam 3 (((exam2 – exam1) + (exam3 – exam2))/2), with the higher improvement coming first.
- Next, if the improvement scores are the same between two students, sort by number of daily quizzes completed, with the most quizzes coming first.
- If all other requirements are the same, sort alphabetically by last name then first name.

Input: The input will begin with an integer, num (0 < num <= 1000), denoting the number of test cases to follow. Each test case will begin with one integer, n (0 < n < 1000), denoting the number of students in the class to be sorted. Each line will begin with two strings, separated by spaces, denoting the first and last names of the student in question, respectively. These strings will be followed by 6 integers, ca, tc, e1, e2, e3, and dq, denoting the classes attended, total classes, exam 1 score, exam 2 score, exam 3 score, and daily quizzes completed, respectively. Each of the student records will appear on its own line, and all values (string and integer both) will be separated by spaces.

Output: Output the names of the students, first name then last name separated by spaces, followed by a space, followed by the average of the 3 exam grades rounded to 2 decimal places, each on its own line, in sorted order. Every test case should be followed by a line of 10 asterisks.

Sample input:

```
2
3
Benjamin Armstrong 10 11 82 89 95 9
Derrick Martin 8 11 70 80 90 11
```

Matthew Sheldon 10 11 92 93 94 11
2
William Armstrong 8 9 80 90 100 7
AppleBottom Jeans 6 9 70 85 100 8

Sample output:

Benjamin Armstrong 88.67
Matthew Sheldon 93.00
Derrick Martin 80.00
****
William Armstrong 90.00
AppleBottom Jeans 85.00
******

Problem # 7 Easy DFS—Finding components of a graph

StatementYou are given a 2D grid representing a map where:

- '1' = land
- '0' = water

An island is a group of '1's that are connected horizontally or vertically (4 directions: up, down, left, right). Cells connected only diagonally do not count as the same island.Your task is to count the total number of islands in the grid. The grid is surrounded by water on all edges.Constraints

- 1 ≤ rows, cols ≤ 200
- Grid contains only '0' and '1'
- Time limit: 1 second (plenty for DFS)
- Memory limit: 256 MB

Input Format

- First line: two integers R C (number of rows and columns)
- Next R lines: each contains C characters ('0' or '1') without spaces

Output Format

- Single integer: the number of islands

Sample Input 1

4 5
11110
11010
11000
00000

Sample Output 1

1

Sample Input 2

4 5
11000
11000
00100
00011

Sample Input 2

4 5

11000
11000
00100
00011
Sample Output 2
3