PROGRAMMING ASSIGNMENT 2

Course Instructors: Assoc. Prof. Dr. Erkut Erdem, Assoc. Prof. Dr. Hacer Yalım Keleş,Asst. Prof. Dr. Adnan Özsoy

TAs: Sümeyye Meryem Taşyürek, Hikmet Can Doğancı

Programming Language: Java (OpenJDK 11)

Due Date: Friday, 18.04.2025 (23:59:59)

1 Finding Diamond in Search of Gold

After Earth became uninhabitable, humanity had no choice but to journey to Kepler-452b. When the expedition team first arrived, they discovered a mysterious cave. As they ventured inside, they stumbled upon the entrance to an ancient dungeon filled with both treasures and traps.

After exploring every room, one of the team members discovered a hidden button. Pressing it revealed a secret chamber within the dungeon. Inside, thousands of scrolls lined the walls. However, when the team attempted to retrieve them, an unknown technology prevented them from doing so.

Amidst their search, they found a single scroll that was unprotected. It revealed the existence of unique artifacts capable of granting humanity powers beyond comprehension. The scroll also contained coordinates to another planet.

With their oxygen supply limited, the team must unlock as many scrolls as possible before time runs out. Each scroll has a complexity value of C, and requires C level of knowledge to bypass its safe-lock. Every safe-lock bypassed gives access to S scrolls.

2 Part 1: Opening the Scrolls

You will use dynamic programming to solve this part.

Since the team has limited time, they must maximize the number of scrolls they can unlock before their oxygen runs out. You have exactly same oxygen amount in minutes just to check every safe-lock in the room. As a team leader, your task is to develop an algorithm that evaluates the complexity of each safe-lock, determining the time required to open it, and the number of scrolls it contains. The algorithm should strategically prioritize safes to maximize the total number of scrolls retrieved in the available time. The algorithm can either work on decoding the alien language, or bypassing safe-lock. While decoding the alien language, computer gains 5 knowledge every minute. These knowledge is spent on bypassing safe-lock. **Note that if a safe-lock can't be opened under a minute, it destroys itself. So you shouldn't check the safe-locks discovered beforehand.**

In the equation 1, you will use the given constraints to find the maximum number of scrolls that can be gathered. In one minute team can either generate knowledge or bypass safe-lock. An example is provided under the equation for a better understanding.

Possibilities:

- Generate 5 units of knowledge, or
- Open the current safe (if sufficient knowledge exists)

Given:

- Each safe i requires ~~C_i~~ ~~knowledge~~ and contains S_i scrolls
- Initial knowledge: 0
- Maximum possible knowledge: $5T$

The optimal strategy follows this Bellman equation:

$$V(i,k) = \max \begin{cases} V(i-1, k-5) & \text{(Generate knowledge)} \\ V(i-1, k+C_i) + S_i & \text{if } k \geq C_i \text{ (Open safe)} \\ V(i-1,k) & \text{(Maintain state)} \end{cases} \quad (1)$$

The complexity table is stored in SafesDiscovered.dat. Team adds new complexity values and scroll outputs every minute. For example at minute 1, team found a safe-lock that has a complexity value of 5. Bypassing this safe-lock will give the team access to 12 scrolls. At minute 2, they found another one with complexity of 5, and scroll amount of 7.

After calculating the number of scrolls acquired, print your answer in the following format:

```
##Initiate Operation Safe-lock##
Maximum scrolls acquired:  32
For the safe set of :[[5,12],[5,7],[10,16],[15,24],[5,16]]
##Operation Safe-lock Completed##
```

	Minute 1	Minute 2	Minute 3	Minute 4	Minute 5
Complexity Level	5	5	10	15	5
Scroll Amount	12	7	16	24	16

Minute 1:

- Generate 5 knowledge: dp[1][5] = 0.
- Cannot open Safe 1 yet (requires 5 knowledge, but you start with 0).

Minute 2:

- Generate another 5 knowledge: dp[2][10] = 0.
- Open Safe 2 Only And Generate Knowledge Afterwards: dp[2][0] = 7.

Minute 3:

- Generate another 5 knowledge: dp[3][15] = 0.
- Open Safe 2 Only And Generate Knowledge Afterwards: dp[3][0] = 7.
- Open Safe 3 Only And Generate Knowledge Afterwards: dp[3][0] = 16.
- Take max of available opens: dp[3][0] = 16.

Minute 4:

- Generate another 5 knowledge: dp[4][20] = 0.
- Open Safe 2 Only And Generate Knowledge Afterwards: dp[4][0] = 7.
- Open Safe 3 Only And Generate Knowledge Afterwards: dp[4][0] = 16.
- Open Safe 4 Only And Generate Knowledge Afterwards: dp[4][0] = 24.
- Take max of available opens: dp[4][0] = 24.

Minute 5:

- Generate another 5 knowledge: dp[5][25] = 0.
- Open Safe 2 Only And Generate Knowledge Afterwards: 7.
- Open Safe 3 Only And Generate Knowledge Afterwards: 16.
- Open Safe 4 Only And Generate Knowledge Afterwards: 24.
- Open Safe 3 Then Open Safe 5: 16 + 16 = 32.
- Take max of available opens: dp[5][0] = 32.

3 Part 2: Gathering the Artifacts

You will use greedy programming to solve this part.

After opening the scrolls, the team discovered data about artifacts located on another planet. The scrolls contained information about the powers, weights, and specific locations of the artifacts. As the team leader, you informed the base about these discoveries. Now, the base will take over from here.

After receiving the messages, the base commander ordered their engineers to write an algorithm that uses the least number of spaceships to retrieve the most artifacts. The algorithm should strategically prioritize the largest artifacts to minimize the total number of spaceships used. **You must use greedy programming for this part. Otherwise, you will receive a score of 0.**

Use equation 2 to implement the optimal solution. To use this equation put the heaviest object to the spaceships first. Go through the each artifact to find the minimum number of spaceships possible. An example is provided below the equation for a better understanding. The weight table is stored in the file ArtifactsFound.dat. Read the weight table and return the minimum number of spaceships required.

Constraints:

- Maximum weight capacity: $C = 100$ units (constant for all ships)
- Set of artifacts with weights $\{w_1, w_2, \ldots, w_n\}$
- Objective: Minimize the total number of spaceships used

The optimal solution can be expressed as:

$$\text{MinShips} = \min \left\{ \underbrace{k}_{\#\text{ of ships}} \in N \;\middle|\; \underbrace{\sum_{i=1}^{k} C_i}_{\text{Total ship capacity}} \geq \underbrace{\sum_{j=1}^{n} w_j}_{\text{Total artifact weight}} \right\} \tag{2}$$

After calculating the number of spaceships required, print your answer in the following format:

```
##Initiate Operation Artifact##
Minimum spaceships required:  3
For the artifact set of :[20,90,80,10,100]
##Operation Artifact Completed##
```

	Artifact 1	**Artifact 2**	**Artifact 3**	**Artifact 4**	**Artifact 5**
Weights	10	80	90	20	100

Step 1:

- Sort the weights in descending order.

Step 2:

- Check the available spaceships, and add spaceship if there is no available space.
- spaceships=(100).

Step 3:

- Check the available spaceships, and add spaceship if there is no available space.
- spaceships=(100,90).

Step 4:

- Check the available spaceships, and add spaceship if there is no available space.
- spaceships=(100,90,80).

Step 5:

- Check the available spaceships, and add spaceship if there is no available space.
- spaceships=(100,90,100).

Step 5:

- Check the available spaceships, and add spaceship if there is no available space.
- spaceships=(100,100,100).
- We put every artifact on the spaceships. We will return the number of spaceships.

4 Starter Code

You must use the starter code provided here.

5 Code Run Configurations

Your code will be compiled and run as follows:

javac *.java -d .
java Main <MaxScrollsDP><MinShipsGP>

6 Grading Policy

- Dynamic Programming Tests - 50%
- Greedy Programming Tests - 40%
- Output Tests - 10%

7 Important Notes

- **Brute force solutions for dynamic and greedy programming parts will not be accepted and will be graded with 0.**
- **Usage of any external libraries is forbidden.**
- Use the provided starter code.
- Do not miss the deadline: Friday, 18.04.2025 (23:59:59).
- Save all your work until the assignment is graded.
- The assignment solution you submit must be your original, individual work. Duplicate or similar assignments are both going to be considered as cheating.
- You can ask your questions via Piazza (https://piazza.com/hacettepe.edu.tr/spring2025/bbm204), and you are supposed to be aware of everything discussed on Piazza.
- You will submit your work via https://submit.cs.hacettepe.edu.tr/ with the file hierarchy given below:

b<studentID>.zip

- Main.java <FILE>
- MaxScrollsDP.java <FILE>
- MinShipsGP.java <FILE>
- OptimalScrollSolution.java <FILE>
- OptimalShipSolution.java <FILE>

- The name of the main class that contains the main method should be **Main.java**. The main class and all other classes should be placed directly (no subfolders) into a zip file named **src.zip**.
- This file hierarchy must be zipped before submitted (not .rar, only .zip files are supported).

8 Academic Integrity Policy

All work on assignments **must be done individually**. You are encouraged to discuss the given assignments with your classmates, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in pseudocode) **will not be tolerated**. In short, turning in someone else's work (including work available on the internet), in whole or in part, as your own will be considered as **a violation of academic integrity**. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

The submissions will be subjected to a similarity check. Any submissions that fail the similarity check will not be graded and will be reported to the ethics committee as a case of academic integrity violation, which may result in suspension of the involved students.