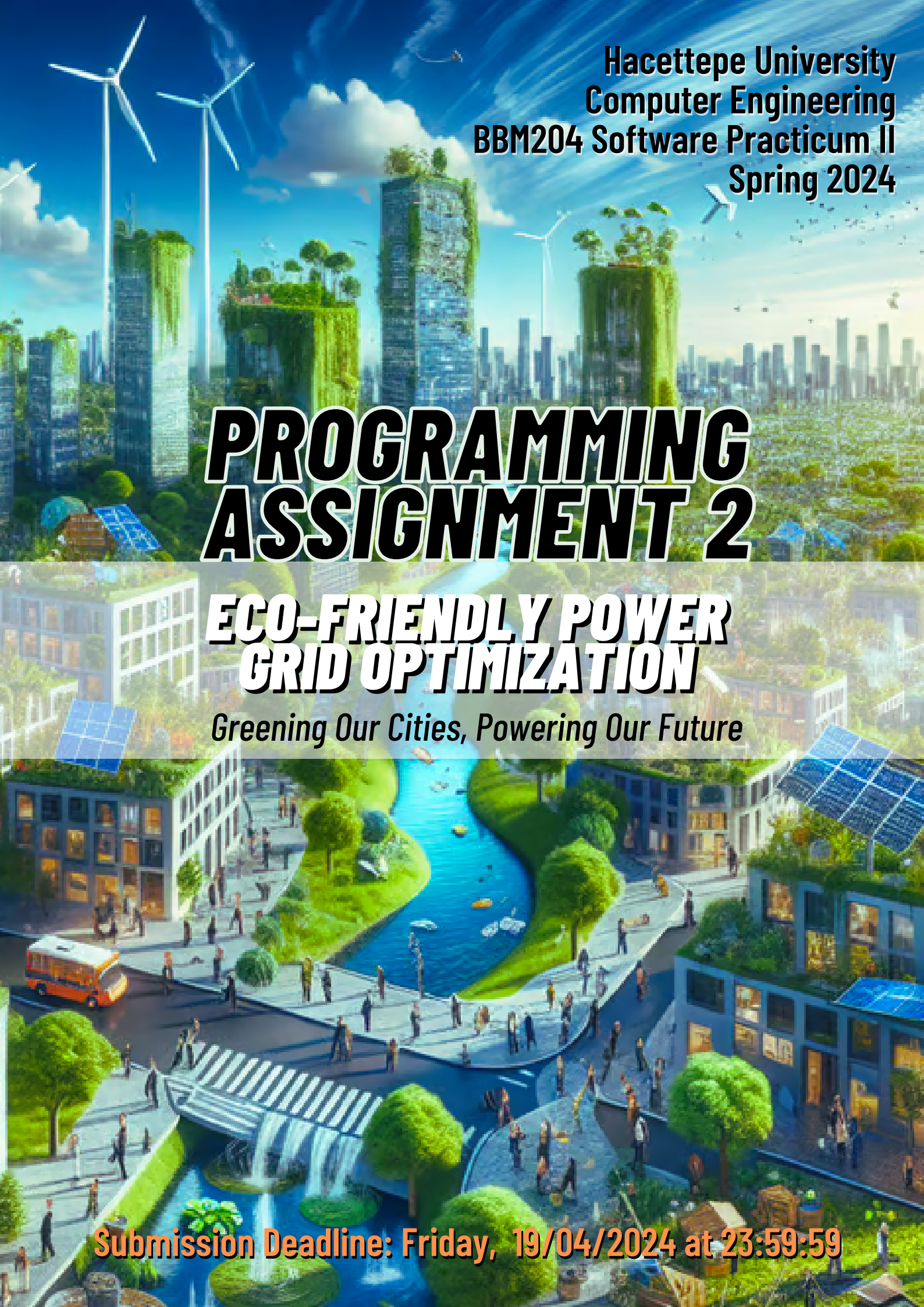

Hacettepe University
Computer Engineering
BBM204 Software Practicum II
Spring 2024

PROGRAMMING ASSIGNMENT 2

ECO-FRIENDLY POWER GRID OPTIMIZATION

Greening Our Cities, Powering Our Future

Submission Deadline: Friday, 19/04/2024 at 23:59:59

Hacettepe Computer Engineering - BBM204 - Spring 2024
Programming Assignment 2 - Deadline: 19/04/2024 at 23:59:59

Topics: Dynamic Programming, Greedy Programming

Course Instructors: Assoc. Prof. Dr. Erkut Erdem, Assoc. Prof. Dr. Hacer Yalım Keleş, Assoc. Prof. Dr. Adnan Özsoy
TAs: Alperen Çakın, Dr. Selma Dilek
Programming Language: OpenJDK 11 - **You MUST USE this starter code**
Due Date: **Friday, 19.04.2024 (23:59:59)**

Eco-Friendly Power Grid Optimization

The Ministry of Environment, Urbanization and Climate Change, in collaboration with the Ministry of Energy and Natural Resources, has embarked on a transformative initiative titled "*Eco-cities: the Future of Urban Türkiye.*" This visionary project aims to revolutionize urban landscapes across Türkiye by transitioning its major cities into eco-cities. These future-forward cities will have sustainable energy management systems at their core, setting a global standard for environmental care and urban living.

Central to the eco-city concept is a reliance on renewable energy sources to power the urban grid. Solar panels produce different amounts of energy depending on the time and weather, wind turbines vary with wind speed, hydroelectric power depends on water flow, etc. Due to the variability in energy production from these sources - affected by time of day, weather, and other environmental factors - the city will rely on an advanced network of battery storage systems (battery banks) to balance energy supply and demand. The battery storage system will be capable of storing energy when production exceeds demand and then supplying energy back into the grid when demand outstrips production. The efficiency of energy release from the batteries increases the longer they are allowed to charge. This efficiency can be thought of as the amount of energy that can be effectively utilized for the city's consumption without loss.

Ankara has been chosen as the pilot city in which the new eco-friendly power grid optimization system will be deployed and tested. In a significant nod to the talent and potential of the nation's youth, the bright students of the Hacettepe University Computer Engineering Department have been given a pivotal role: optimizing the system's efficiency. Your work promises not only to enhance Ankara's transition into an eco-city but also to lay down a blueprint for the sustainable transformation of urban centers across Türkiye and beyond.

Hacettepe Computer Engineering - BBM204 - Spring 2024
Programming Assignment 2 - Deadline: 19/04/2024 at 23:59:59

1 Mission 0: Power Grid Optimization

In this part, you will practice dynamic programming.

"Those who cannot remember the past are condemned to repeat it."

Dynamic Programming - Useful Background

Dynamic Programming is a technique in computer programming that helps to efficiently solve a class of problems that have overlapping subproblems and optimal substructure property. The majority of the Dynamic Programming problems can be categorized into optimization and combinatorial problems. The optimization problems expect you to select a feasible solution, so that the value of the required function is minimized or maximized. In this mission, you will apply a dynamic programming approach to an optimization problem. The two main approaches to dynamic programming are memoization (the top-down approach) and tabulation (the bottom-up approach). You will employ a dynamic programming approach on the problem at hand. To do that, you will be given a pseudocode of a problem solution, in which you will have to make sure you cache the solutions to the sub-problems and use them to find the optimal solution overall.

Using the city's energy demand schedule for a given hour (denoted as $D(t)$, representing the energy demand at hour t) and the battery bank efficiency formula ($E(i)$, indicating the amount of energy that can be effectively released from the battery bank after i hours of charging), you need to implement a dynamic programming solution that optimizes the timing of battery discharges to meet the city's energy demand across all hours, maximizing the effective use of stored energy.

The energy demand schedule stored in `demandSchedule.dat` as an array of integer numbers separated by a space (e.g., `2 6 10 1 3`), will be given as the *first command-line argument*. An example of how to interpret such a schedule is provided below.

Hour at which new demand is sent (t)	1	2	3	4	5
City's energy demand at hour t: D(t)	2	6	10	1	3

The given example schedule can be interpreted as follows: At hour 01:00, the first energy demand of 2 gigawatt-hours (GWh) will be made. The second demand will arrive one hour later and will consist of 6 GWh, etc. Assume that there will be exactly one hour between each demand arrival.

The battery bank charging efficiency formula is given by $E(i) = i^2$, which specifies the amount of energy (in GWs) that can be effectively discharged from the battery bank after i hours of charging.

Charging Time in Hours (i)	1	2	3	4	5
GWs it can discharge E(i)	1	4	9	16	25

More specifically, if the battery bank is discharged in the i^{th} hour and j hours have passed since the previous discharge, then it can satisfy at most $\min[D(i), E(j)]$ GWs or energy, after which the battery bank will be completely drained and will have to recharge for at least one hour before it can be discharged again. **Note that only energy demands arriving at that specific hour can be met; any remaining unsatisfied energy demand from previous hours will have to be met from other sources and does not carry over to the subsequent hour(s)**.

You can assume that the battery bank is completely drained at the very beginning (hour 0). It cannot be used earlier than one hour into the charging (exactly when the first energy demand arrives), and if it is used in the i^{th} hour for the first time, it can satisfy at most up to $E(i)$ GWs of the city's energy demand.

For the given example energy demand schedule, the optimal solution would be to discharge the battery bank at hours 3 and 5, for a total satisfied energy demand count of 12 GWs. If the battery bank is discharged in the first hour it can only produce 1 GW of energy, and if it is discharged in the second hour for the first time it can only produce 4 GWs of energy even though 6 GWs of energy is demanded, because the charging time was only 2 hours ($E(2) = 2^2 = 4$). By discharging the battery bank in hours 3 and 5, we can obtain a total of:

$\min\{D(3) = 10, E(3) = 9\} + \min\{D(5) = 3, E(5-3) = E(2) = 4\} = 9 + 3 = 12$ GWs.

As you can see, this optimization problem is computationally very complex to use a brute force approach to solve it. Any combination of hours at which the battery bank is discharged can be a potential optimal solution, and if we were to check each possibility, there would be $2^N - 1$ possible battery-power usages to inspect for N hours of energy demands. An example for $N = 3$ would lead into inspection of seven sets of hours for discharging the battery bank: $\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}$, which would lead to a terrible exponential computational complexity as N grows. You will implement the dynamic programming approach given in Algorithm 1 to solve the problem with a better time complexity.

Algorithm 1 Optimal Eco-Friendly Power Grid Solution Dynamic Programming

```
procedure GetOptimalPowerGridSolutionDP
    SOL(0) ← 0
    HOURS(0) ← []
    for j← 1, . . . , N do
        SOL(j) ← max_{0<=i<j}[(SOL(i) + min[D(j), E(j−i)]]
        HOURS(j) ← [HOURS(i), j]
    end for
    return SOL, HOURS
end procedure
```

The dynamic programming approach suggested in the given pseudocode works as follows: At each hour j, check already stored optimal solutions for each hour i from 0 to $j-1$ ($0 \leq i < j$). If the demands end at hour j, there is no reason not to discharge the battery bank at hour j as we don't have to save the charge for any future energy demands. Therefore, the optimal solution at any hour j will be the maximum of all solutions over the hour interval ($0 \leq i < j$) given as the recurrence relation $SOL(j) \leftarrow \max_{0<=i<j}[(SOL(i) + \min[D(j), E(j-i)]]$; i.e., the maximum number of GWs

that can be satisfied from the energy demand at hour i plus the number of GWs that can be satisfied at most at hour j with the battery bank charged for $(j-i)$ hours, which is $\min[D(j), E(j-i)]$. A visualization of this reasoning is given below for hour 5.

```
Assume we used the given algorithm and obtained the optimal solutions for hours 0,..,4:
SOL(0) =  0, HOURS(0) = []
SOL(1) =  1, HOURS(1) = [1]
SOL(2) =  4, HOURS(2) = [2]
SOL(3) =  9, HOURS(3) = [3]
SOL(4) = 10, HOURS(4) = [3,4]

Then the optimal solution for hour 5 is the best (max) option among these available ones:
Option 0: Use SOL(0) and discharge at 5 -> SOL(0) + min[D(5)=3, E(5-0)=E(5)=25] =  0 + 3 =  3
Option 1: Use SOL(1) and discharge at 5 -> SOL(1) + min[D(5)=3, E(5-1)=E(4)=16] =  1 + 3 =  4
Option 2: Use SOL(2) and discharge at 5 -> SOL(2) + min[D(5)=3, E(5-2)=E(3)=9]  =  4 + 3 =  7
Option 3: Use SOL(3) and discharge at 5 -> SOL(3) + min[D(5)=3, E(5-3)=E(2)=4]  =  9 + 3 = 12 <- MAX
Option 4: Use SOL(4) and discharge at 5 -> SOL(4) + min[D(5)=3, E(5-4)=E(1)=1]  = 10 + 1 = 11

Since the option 3 results in the maximum value: SOL(5) = 12, HOURS(5) = [HOURS(3), 5] = [3,5]
```

Mission Steps Summarized

- Given the information about city's energy demand for each hour $D(t)$, and the battery bank's power recharge function $E(i)$, find the maximum number of demanded GWs that can be satisfied by discharging the battery bank at certain hours.
- Use the provided Dynamic Programming pseudocode and implement your solution by completing the **`GetOptimalPowerGridSolutionDP`** function in **`PowerGridOptimization.java`**. **A brute force solution will not be accepted and will be graded with 0.**
- The **`GetOptimalPowerGridSolutionDP`** function should return two pieces of information: the maximum number of GWs that can be satisfied, and an array of hours at which the battery bank should be discharged for the calculated maximum energy coverage. This will be achieved by having this method return an instance of **`OptimalPowerGridSolution`** (see **`OptimalPowerGridSolution.java`**).

The expected output for the given sample inputs is given below and should be printed to the STDOUT. **Note that your output format must match the given format to pass the output test**.

```
##MISSION POWER GRID OPTIMIZATION##
The total number of demanded gigawatts: 22
Maximum number of satisfied gigawatts: 12
Hours at which the battery bank should be discharged: 3, 5
The number of unsatisfied gigawatts: 10
##MISSION POWER GRID OPTIMIZATION COMPLETED##
```

2 Mission 1: Eco-Maintenance

In this part, you will practice greedy programming.

Greedy Programming - Useful Background

Greedy Programming is an algorithmic paradigm that constructs a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. This paradigm is ideal for problems where a locally optimal choice also leads to a globally optimal solution. In this mission, you will use a greedy programming approach to solve another optimization problem crucial for sustaining the eco-city.

After successfully deploying our eco-friendly power grid optimization system, we face a new challenge. To ensure the continued efficiency and sustainability of the eco-city's infrastructure, Electric Service Vehicles (ESVs) are tasked with maintenance and emergency response. Each ESV has a specific energy capacity, denoted in kilowatt-hours (kWh), dictating how much energy it can carry for a single dispatch. You can assume all ESVs will have the same energy capacity.

Before each EcoCare mission of dispatching ESVs for various eco-maintenance tasks, the dispatch center receives updated information about the number of available ESVs and the respective energy requirements of the maintenance tasks that need to be carried out across the city. This information will be stored in `ESVMaintenance.dat` given as the *second command-line argument*. The first row will indicate the number of available ESVs and the energy capacity of each ESV in kWh, separated by a space, and the second row will list the energy requirements (in kWh) of the maintenance tasks in a random order, also separated by a space. An example input file that informs about eight available ESVs, each of which has a capacity of 100 kWh, and nine maintenance tasks demands with requirements of 20, 50, 40, 70, 10, 30, 80, 100, and 10 kWh, respectively, will look as follows:

```
8 100
20 50 40 70 10 30 80 100 10
```

Operating an ESV is costly, and we aim to minimize the number of ESVs deployed while ensuring that all maintenance tasks are completed. Your mission is to use a greedy approach based on the Pseudocode 2 to determine the minimum number of ESVs needed to carry out all tasks. **A brute force solution will not be accepted and will be graded 0.**

Algorithm 2 Optimal ESV Deployment Greedy Programming

```
1: procedure getMinNumESVsToDeployGP(maxNumAvailableESVs: int, ESV_CAPACITY: int)
2:     Sort all maintenance tasks by their energy requirements in decreasing order   ▷ So that we can consider the largest demands first.
3:     Store the remaining energy capacities in all available ESVs
4:     for i← 1, . . . , numTasks do
5:         Find the first ESV that can accommodate task i
6:         Get a new ESV only if the task does not fit in any of the already used ESVs
7:     end for
8:     return minNumESVsToDeploy or -1 if all tasks could not be accommodated.
9: end procedure
```

The greedy method tries to handle the most energy-demanding tasks first, taking the locally optimal decision at each step without considering if it might lead to a suboptimal overall solution. This approach offers a fast solution to an otherwise NP-hard problem.

For the provided sample input file, the greedy approach described above will produce the solution illustrated below. This solution will indicate the minimum number of ESVs required to complete all maintenance tasks.

Mission Steps Summarized

- Given the number of available ESVs, their energy capacity, and the energy requirements of maintenance tasks, find the minimum number of ESVs that should be deployed to maintain the eco-city efficiently.
- Implement your solution using the provided Greedy Programming pseudocode in `OptimalESVDeploymentGP.java`. **A brute force solution will not be accepted and will be graded 0.**
- The **`getMinNumESVsToDeploy`** function should return the minimum number of ESVs needed if all tasks can be accommodated, or **`-1`** if the tasks cannot be completed with the available ESVs.
- Instance variable **`maintenanceTasksAssignedToESVs`** should be populated correctly after calling **`getMinNumESVsToDeploy`** function. This variable should store assigned tasks to ESVs.

For the given sample input, the expected output is given below. The output should be printed to the STDOUT immediately after the output from the Mission 0. **Note that your output format must match the given format to pass the output test**.

```
##MISSION ECO-MAINTENANCE##
The minimum number of ESVs to deploy: 5
ESV 1 tasks: [100]
ESV 2 tasks: [80, 20]
ESV 3 tasks: [70, 30]
ESV 4 tasks: [50, 40, 10]
ESV 5 tasks: [10]
##MISSION ECO-MAINTENANCE COMPLETED##
```

In case all tasks cannot be satisfied with the available ESVs (not enough ESVs or the energy demand of at least one task exceeds the ESVs' energy capacity), the output should be as follows:

```
##MISSION ECO-MAINTENANCE##
Warning: Mission Eco-Maintenance Failed.
##MISSION ECO-MAINTENANCE COMPLETED##
```

Must-Use Starter Codes

You MUST use **this starter code**. All classes should be placed directly inside your **zip** archive. Feel free to create other additional classes if necessary, but they should also be directly inside **zip**.

Grading Policy

- Implementation of the algorithms: 90%
  - Dynamic programming tests: 50%
  - Greedy programming tests: 40%
- Output test: 10%

Note that you need to get a NON-ZERO grade from the assignment to get the submission accepted. Submissions graded with 0 will be counted as NO SUBMISSION!

Important Notes

- **Brute force solutions for dynamic and greedy programming parts will not be accepted and will be graded with 0.**
- Do not miss the deadline: **Friday, 19.04.2024 (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/spring2024/bbm204), and you are supposed to be aware of everything discussed on Piazza.
- You must test your code via Tur^{6}Bo Grader (**does not count as submission!**).
- You must submit your work via https://submit.cs.hacettepe.edu.tr/ with the file hierarchy given below:
  - **b<studentID>.zip**
    - Main.java <FILE>
    - PowerGridOptimization.java <FILE>
    - OptimalPowerGridSolution.java <FILE>
    - OptimalESVDeploymentGP.java <FILE>
- The name of the main class that contains the main method should be **Main.java**. **You MUST use this starter code**. The main class and all other classes should be placed directly in your **zip** archive. Feel free to create other additional classes if necessary, but they should also be inside the **zip**.
- This file hierarchy must be zipped before submitted (not .rar, only .zip files are supported).
- **Usage of any external libraries is forbidden.**

Hacettepe Computer Engineering - BBM204 - Spring 2024
Programming Assignment 2 - Deadline: 19/04/2024 at 23:59:59

Run Configuration

Your code will be compiled and run as follows:

```
javac *.java -d .
java Main <demandSchedule> <ESVMaintenance>
```

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.

References

[1] "Dynamic Programming", https://www.programiz.com/dsa/dynamic-programming, Last Accessed: 18/03/2024.

[2] "Introduction to Dynamic Programming 1", https://www.hackerearth.com/practice/algorithms/dynamic-programming/introduction-to-dynamic-programming-1/tutorial/, Last Accessed: 18/03/2024.

[3] Luis Robaina, "A Systematic Approach to Dynamic Programming", https://betterprogramming.pub/a-systematic-approach-to-dynamic-programming-54902b6b0071, Last Accessed: 18/03/2024.

[4] "Greedy Algorithms", https://www.geeksforgeeks.org/greedy-algorithms/, Last Accessed: 18/03/2024.