DJILLALI LIABES UNIVERSITY OF SIDI BEL ABBES
FACULTY OF EXACT SCIENCES
DEPARTMENT OF COMPUTER SCIENCES

Module : Algorithmique et Complexité
1ST YEAR OF MASTER'S DEGEREE IN
NETWORKS,INFORMATION SYSTEMS & SECURITY(RSSI)
2021/2022

Le problème de la sous somme avec une approche programmation dynamique

---

Author:
HADJAZI M.Hisham
AMUER Wassim Malik
Group: 01/RSSI

Supervisor:
Dr. MME. BELKHODJA
ZENAIDI Lamia

A paper submitted in fulfilment of the requirements for the
TP-05

December 19, 2021

Contents

1 Solutions of TP-05 **1**

1.1 Q1/ Expliquer brièvement comment ce problème peut-il être résolu avec une approche programmation dynamique. 2

1.1.1 Dynamic Programming with Memoization. 2

1.2 Q2/Donner l'équation de récursive permettant de le résoudre. 2

1.3 Q3/ Ecrire le code java permettant de résoudre le problème avec une approche programmation dynamique (en utilisant l'équation récursive définie au niveau de la question 2). 3

1.4 Q4/Compléter le code pour afficher les sous-ensembles trouvés. 3

1.5 Q5/Pour tester le code, prévoir, comme pour le TP4 : 4

1.5.1 - un choix entre une lecture du tableau à partir du clavier ou une génération automatique et aléatoire du tableau E, ... 4

1.5.2 -une lecture de sa taille n et de la valeur de la somme S à partir du clavier. 4

lecture de sa taille n 4

lecture de la valeur de la somme S 4

1.5.3 -un affichage de la somme, du tableau généré, de la matrice (absente dans le TP4) et des résultats obtenus. 4

la somme 4

tableau généré 4

la matrice 5

résultats obtenus 5

1.6 Q6/Tester le code pour différentes valeurs de S et de n =5, 10, 20, 40, (ce n'est pas nécessaire d'envoyer) 5

1.7 Q7/ Faire des captures d'écran pour n=5, n=10. (À envoyer) 5

1.7.1 n=5 5

1.7.2 n=10 6

1.8 Q8/ Donner des exemples ou des domaines d'application pour ce problème de la sous somme. 6

1.9 Conclusion 7

A Appendix A **8**

A.1 Java Code for SommeDP.java 8

B Appendix B **11**

B.1 Screen Shots of Question 6 11

Chapter 1

Solutions of TP-05

Notes regarding this solution :

This solution and the executions of the code in it was done in the following machine :

- *Machine*: Lenovo Ideapad S210
- *CPU*: Intel Celeron 1037U 1800 MHz
- *RAM*: 8GB DDR3l
- *OS* : Linux Mint 20.2 Cinnamon Kernel v.5.4.0-88
- *IDE* : Eclipse IDE for Java Developers Version: 2019-12 (4.14.0)
- *Java version*: 11.0.11

Définition du problème :

Reprenons le problème traité dans le TP 4 :

Etant donné un ensemble E de n entiers non négatifs, et une valeur S, il s'agit de déterminer s'il existe un sous ensemble de E dont la somme des éléments est égale à S.

Exemple :

E= 3, 6,2, 7, 9 ; S=9 les sous ensemble 3,6, 2,7 et 9 réalisent la solution. Si S=4 aucun sous ensemble ne réalise la solution.

Pour résoudre ce problème, on s'intéresse dans ce TP, à une approche de type « **programmation dynamique** ». On rappelle la fonction récursive déjà utilisée pour le TP4.

```
boolean  Somme(int E[], int n, int S)
{  if (S == 0)  return true;
   if (n == 0 && S > 0) return false;
if (E[n-1] > S)  return  Somme(E, n-1, S);
return  Somme(E, n-1, S) || Somme(E ,  n-1,  S - E [n-1]); }
```

FIGURE 1.1: sous somme Algorithm

1.1 Q1/ Expliquer brièvement comment ce problème peut-il être résolu avec une approche programmation dynamique.

1.1.1 Dynamic Programming with Memoization.

Here we use Memoization technique in Dynamic Programming to save solved sub problems in a 2-dim array rather than compute it again and again. The result is a matrix filled with true and false booleans. the base statements remain the same as before as the recursive solution. This method can save time as the **Time complexity** is approximately $\mathcal{O}(S * n)$ and **Space Complexity** of $S(S * n)$ of course excluding all the extra functions we added to store and print the sub sums and other small functionalities. which makes it way faster than the classic recursive function. The extra variables seen in the full code like **subarray**, **target** and **target2** are just for storing results and presentation, and have nothing to do with the actual solution.

1.2 Q2/Donner l'équation de récursive permettant de le résoudre.

```
if (S = 0)
        return true;

if (n = 0 && S > 0)
        return false;

if (M[S][n] != null)
        return M[S][n];

if (E[n - 1] > S)
        return M[S][n] = F(E, S, n - 1);

         else {
            M[S][n] = F(E, S, n - 1);
            M[S][n] = F(E, S - E[n - 1], n - 1);
        }
```

1.3 Q3/ Ecrire le code java permettant de résoudre le problème avec une approche programmation dynamique (en utilisant l'équation récursive définie au niveau de la question 2).

```java
private static boolean findSubsets(int[] values, Vector<Integer>
    subarray, int target, int n, int target2) {

        if (target == 0) {
            System.out.println("The Sum of subset " + Arrays.
                toString(subarray.toArray()) + " = " + target2);
            return true;
        }
        if (n == 0 && target > 0) {
            return false;
        }

        if (mapSubs[target][n] != null) {
            return mapSubs[target][n];
        }

        if (values[n - 1] > target) {
            mapSubs[target][n] = findSubsets(values, subarray,
                target, n - 1, target2);

        } else {
            mapSubs[target][n] = findSubsets(values, subarray,
                target, n - 1, target2);
            Vector<Integer> subarray2 = new Vector<Integer>(
                subarray);
            subarray2.add(values[n - 1]);
            mapSubs[target][n] = findSubsets(values, subarray2,
                target - values[n - 1], n - 1, target2);
        }
        return mapSubs[target][n];
    }
```

The full code is available at the end of the document in AppendixA A.

1.4 Q4/Compléter le code pour afficher les sous-ensembles trouvés.

```java
        if (target == 0) {
            System.out.println("The Sum of subset " + Arrays.
                toString(subarray.toArray()) + " = " + target2);
            return true;
        }
```

1.5 Q5/Pour tester le code, prévoir, comme pour le TP4 :

1.5.1 - un choix entre une lecture du tableau à partir du clavier ou une génération automatique et aléatoire du tableau E,

```
System.out.println("Select Filling Method:\n(1) : For manual filling\n(2) : For automatic filling\n");
        int a = keyboard.nextInt();

        switch (a) {
        case 1:
            System.out.println("\nManual Filling Activated");
            for (int i = 0; i < n; i++) {
                System.out.println("Value " + (i + 1) + " =");
                values[i] = keyboard.nextInt();
            }
            break;
        case 2:
            System.out.println("\nAutomatic filling Activated");
            Random random = new Random();
            for (int i = 0; i < n; i++) {
                values[i] = random.nextInt(30 + 10);
            }
            break;
        }
```

1.5.2 -une lecture de sa taille n et de la valeur de la somme S à partir du clavier.

lecture de sa taille n

```
        System.out.println("Enter the size of the Array n =\n");
        int n = keyboard.nextInt();
```

lecture de la valeur de la somme S

```
        System.out.println("The target you are looking for =");
        int target = keyboard.nextInt();
```

1.5.3 -un affichage de la somme, du tableau généré, de la matrice (absente dans le TP4) et des résultats obtenus.

la somme

```
System.out.println("The Target Sum is : " + target + "\n");
```

tableau généré

```
System.out.println("The Array is : " + Arrays.toString(values) + "\n");
```

la matrice

```
System.out.println("\nThe Matrix is as follows :\n\n" + Arrays.
    deepToString(mapSubs).replace("], ", "]\n"));
```

résultats obtenus

```
System.out.println("The Sum of subset " + Arrays.toString(subarray.
    toArray()) + " = " + target2);
```

1.6 Q6/Tester le code pour différentes valeurs de S et de n =5, 10, 20, 40, (ce n'est pas nécessaire d'envoyer)

The screen shots of the results of the execution can be found in the AppendixB section B.

1.7 Q7/ Faire des captures d'écran pour n=5, n=10. (À envoyer)

1.7.1 n=5

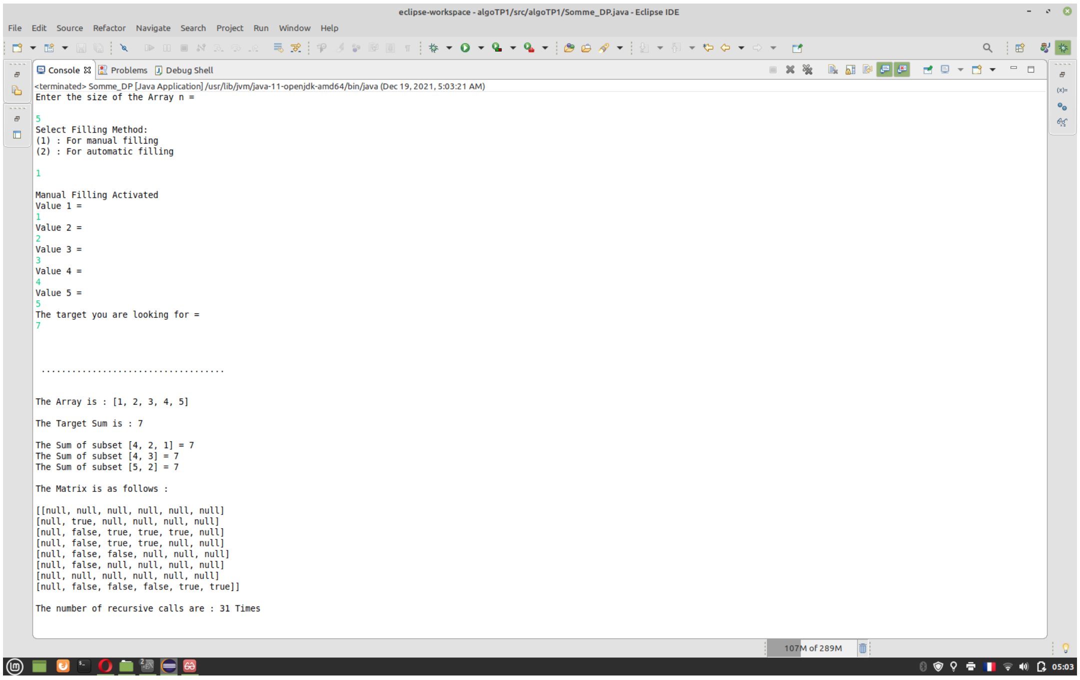

FIGURE 1.2: n=5

1.7.2 n=10

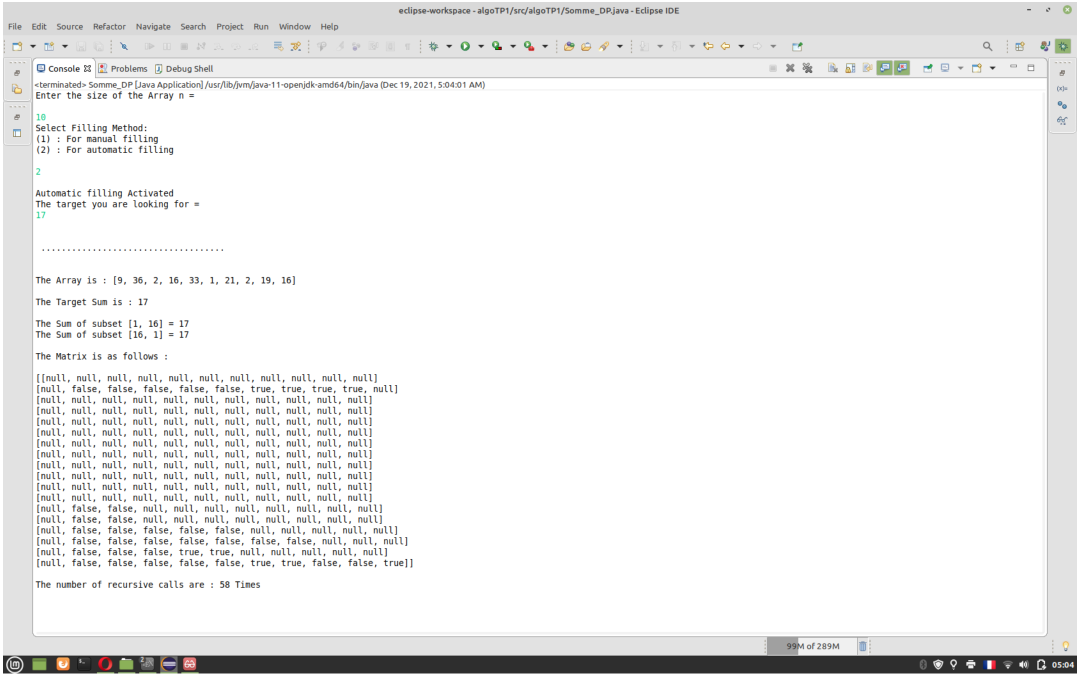

FIGURE 1.3: n=10

1.8 Q8/ Donner des exemples ou des domaines d'application pour ce problème de la sous somme.

- In computer science, it is widely applied to the optimal **memory management in multiple programming**.[8]
- In the field of telecommunication, it is used in **allocating wireless resources to support multiple scalable video sequences**.[2]
- For the application in embedded system, it is used in **generating application specific instructions for DSP applications to reduce the required code size and increasing performance in embedded DSP systems**.[3]
- **Computer passwords**. A computer needs to verify a user's identity before allowing him or her access to an account. The simplest system would have the machine keep a copy of the password in an internal file, and compare it with what the user types. A drawback is that anyone who sees the internal file could later impersonate the user. I believe this alternative is actually implemented on some systems: the computer generates a large number (say 500) of a_i. They are stored in the internal file. A password is a subset of $\{1, \ldots, 500\}$. (in practice, there is a program to convert a normal sequence-of-symbols password to such a subset.) Instead of having the password for the user, the computer keeps the total associated with the appropriate subset. When the user types in the subset, the computer tests whether the total is correct. It does not keep a record of the subset. Thus impersonation is possible only if somebody can reconstruct the subset knowing the a_i and the total.[1]

- **Message verification**. A sender (S) wants to send messages to a receiver (R). Keeping the message secret is not important. However, R wants to be sure that the message he is receiving is not from an imposter and has not been tampered with. S and R agree on a set of a_i (say 500) and a set of totals T_j (say 200). These numbers may be publicly known, but only S knows which subsets of the a_i correspond to which T_j. The message sent by S is a subset of size 100 of $\{1, \ldots, 200\}$. He does this by sending 100 subsets of the a_i corresponding to the message he wants to send.[1]

1.9 Conclusion

Solving This TP was very challenging and engaging, and required some research to solve it recursively as most dynamic programming solutions online are iterative. The most intresting solution I found online and probably the fastest was this from a medium.com blog[7].

```java
class Solution {
    public int findTargetSumWays(int[] nums, int S) {
        int sum = 0;
        for (int num: nums) sum += num;
        if (S > sum || -S < -sum || (S + sum) % 2 == 1) return 0;

        int[] dp = new int[(S + sum) / 2 + 1];
        dp[0] = 1;

        for (int num: nums) {
            for (int i = dp.length - 1; i >= num; i--) {
                dp[i] += dp[i - num]; // Crux
            }
        }

        return dp[dp.length - 1];
    }
}
```

This solution reduces the time complexity and reduces space complexity to a single diminished table only.

What is also interesting is how these topics like finding the subset sum are vital in solving many problems in many areas like **Cryptography**, **Telecommunications**, and **Operating Systems**.

Appendix A

Appendix A

A.1 Java Code for SommeDP.java

```java
//TP5 Algorithmique et Complexite 2021-2022

//Nom:HADJAZI
//Prenom: Mohammed Hisham
//Specialite:   RSSI      Groupe: 01

//Nom:Ameur
//Prenom: Wassim Malik
//Specialite:   RSSI      Groupe: 01

import java.util.Arrays;
import java.util.Random;
import java.util.Scanner;
import java.util.Vector;

public class Somme_DP {
    static long cpt = 0;

    private static Boolean[][] mapSubs;

    // Returns true if there is a subset
    // of set[] with sum equal to given sum
    private static boolean findSubsets(int[] values, Vector<Integer
        > subarray, int target, int n, int target2) {
        cpt++;
        // Base Case 1
        if (target == 0) {
            System.out.println("The Sum of subset " + Arrays.
                toString(subarray.toArray()) + " = " + target2);
            return true;
        }
        // Base Case 2
        if (n == 0 && target > 0) {
            return false;
        }
        // If already calculated just
        // return the result
        if (mapSubs[target][n] != null) {
            return mapSubs[target][n];
        }
        // If last element is greater than
        // sum, then ignore it
```

```
        if (values[n - 1] > target) {
            mapSubs[target][n] = findSubsets(values, subarray,
                target, n - 1, target2);
            /*
             * else, check if sum can be obtained by any of the
                 following (a) including the
             * last element (b) excluding the last element
             */
        } else {
            mapSubs[target][n] = findSubsets(values, subarray,
                target, n - 1, target2);
            Vector<Integer> subarray2 = new Vector<Integer>(
                subarray);
            subarray2.add(values[n - 1]);
            mapSubs[target][n] = findSubsets(values, subarray2,
                target - values[n - 1], n - 1, target2);
        }
        return mapSubs[target][n];
    }

    // Starter function that fills our matrix with null
    private static boolean fillFunc(int[] values, Vector<Integer>
        subarray, int target, int n, int target2) {
        mapSubs = new Boolean[target + 1][n + 1];
        for (int s = 0; s <= target; s++) {
            for (int i = 0; i <= n; i++) {
                mapSubs[s][i] = null;
            }
        }
        return findSubsets(values, subarray, target, n, target2);
    }

    public static void main(String[] args) {

        Scanner keyboard = new Scanner(System.in);

        System.out.println("Enter the size of the Array n =\n");
        int n = keyboard.nextInt();
        int[] values = new int[n];
        Vector<Integer> v = new Vector<Integer>();

        System.out.println("Select Filling Method:\n(1) : For
            manual filling\n(2) : For automatic filling\n");
        int a = keyboard.nextInt();

        switch (a) {
        case 1:
            System.out.println("\nManual Filling Activated");
            for (int i = 0; i < n; i++) {
                System.out.println("Value " + (i + 1) + " =");
                values[i] = keyboard.nextInt();
            }
            break;
        case 2:
            System.out.println("\nAutomatic filling Activated");
            Random random = new Random();
            for (int i = 0; i < n; i++) {
```

```
                values[i] = random.nextInt(30 + 10);
            }
            break;
        }

        System.out.println("The target you are looking for =");
        int target = keyboard.nextInt();
        int target2 = target;
        keyboard.close();

        System.out.println("\n\n
            .................................... \n\n");
        System.out.println("The Array is : " + Arrays.toString(
            values) + "\n");
        System.out.println("The Target Sum is : " + target + "\n");
        fillFunc(values, v, target, n, target2);
        System.out.println("\nThe Matrix is as follows :\n\n" +
            Arrays.deepToString(mapSubs).replace("], ", "]\n"));
        System.out.println("\nThe number of recursive calls are : "
             + cpt + " Times\n");
    }
}
```

Appendix B

Appendix B

B.1 Screen Shots of Question 6

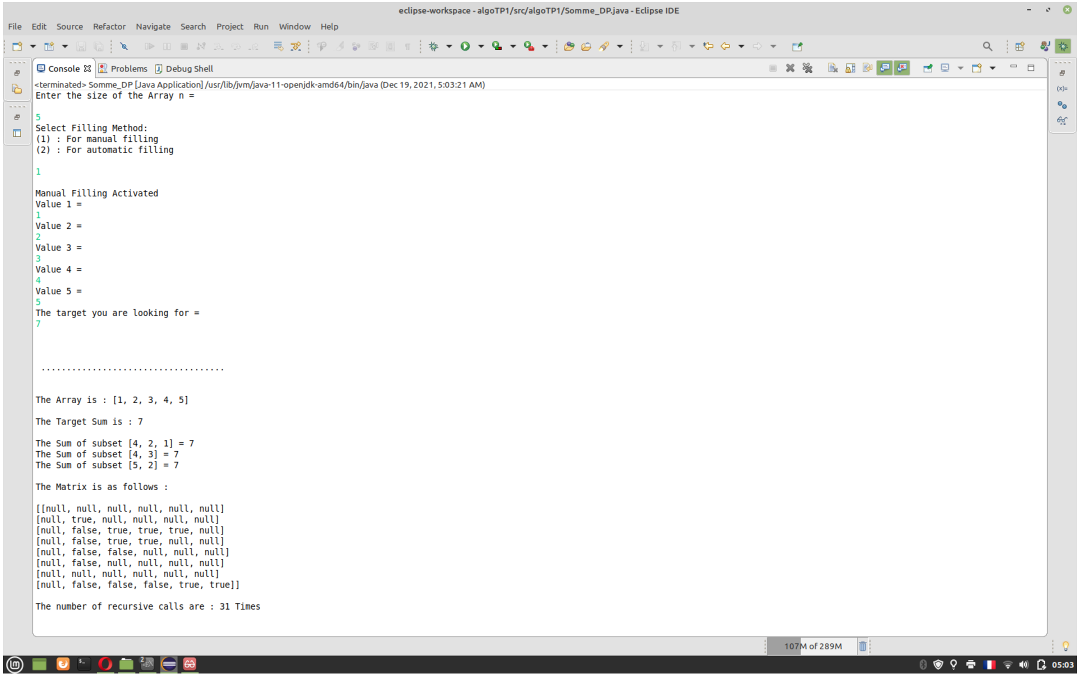

FIGURE B.1: n=5

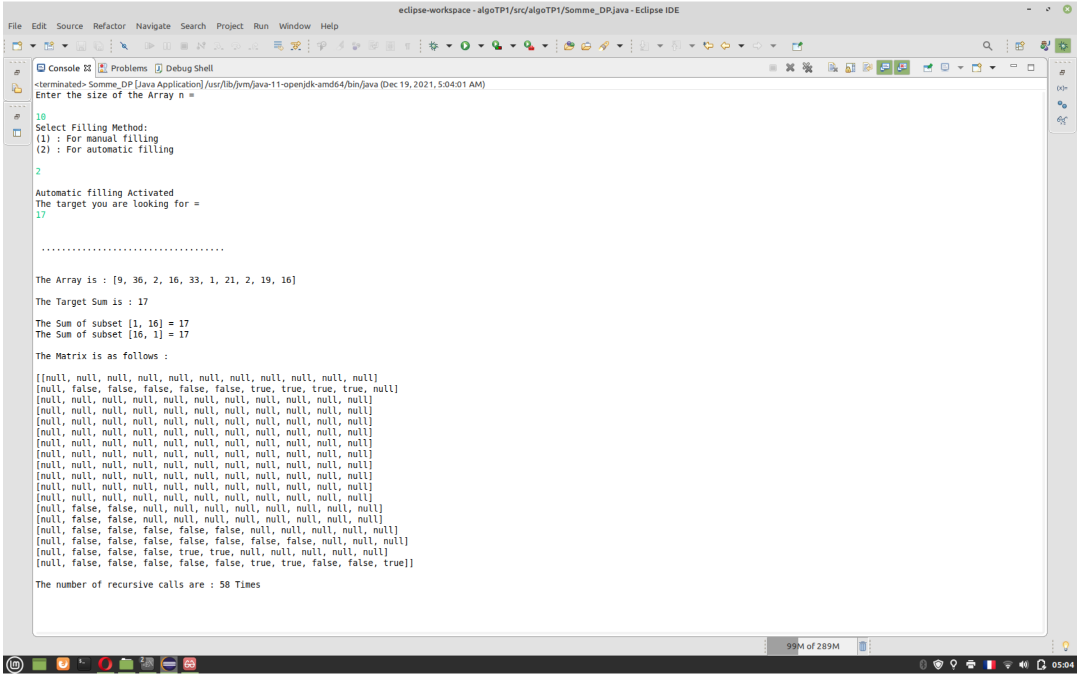

FIGURE B.2: n=10

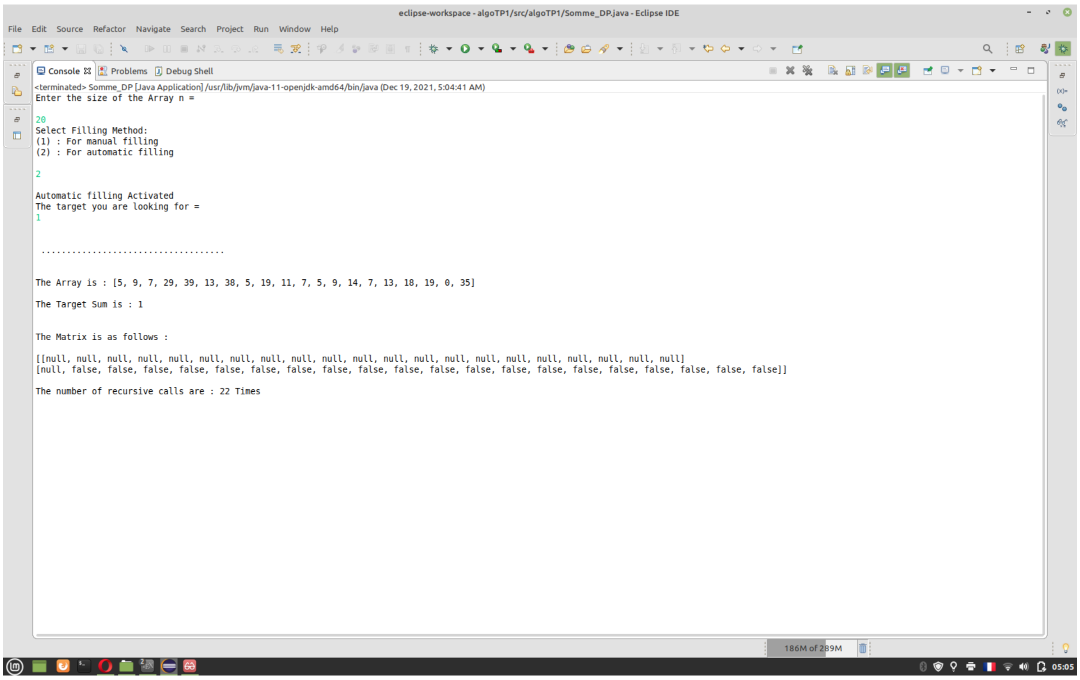

FIGURE B.3: n=20

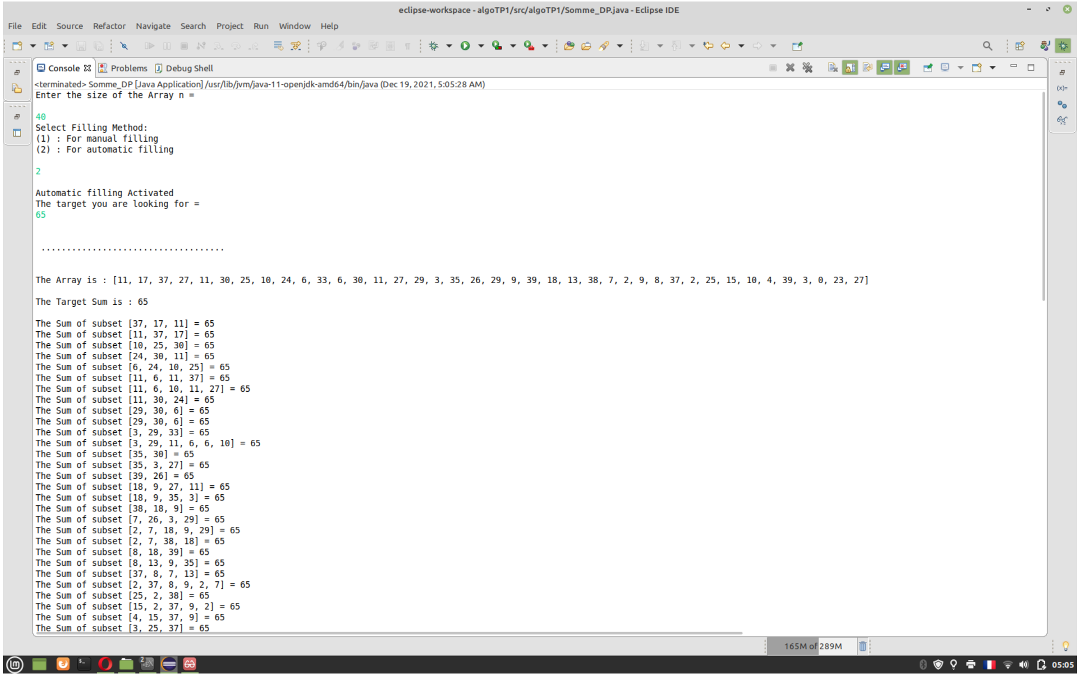

FIGURE B.4: n=40

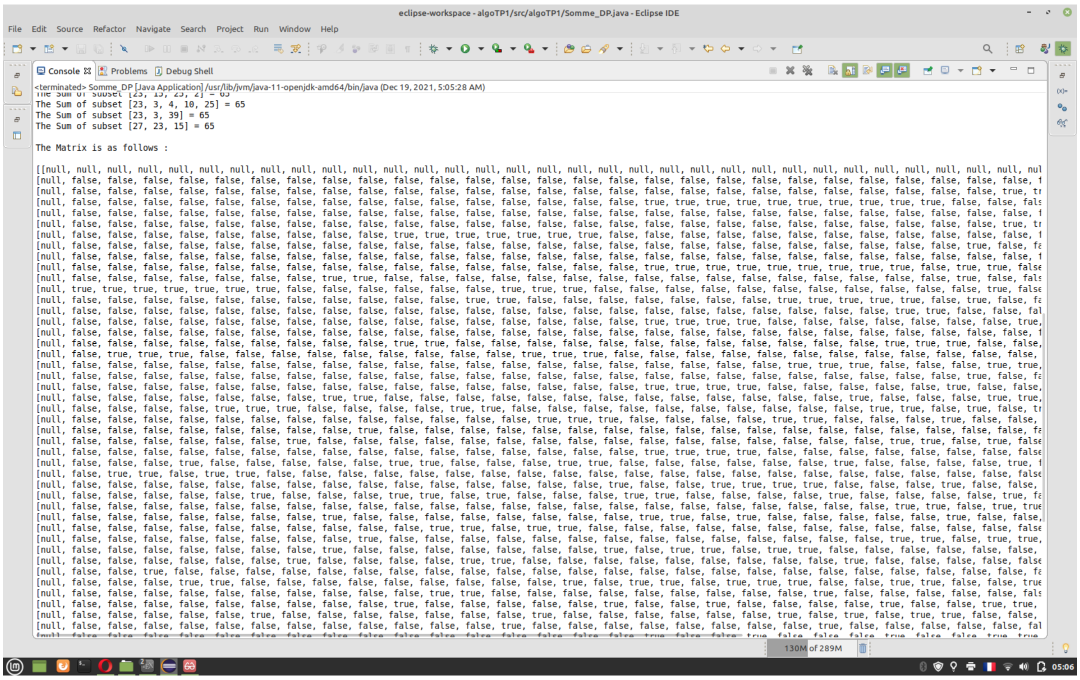

FIGURE B.5: n=40

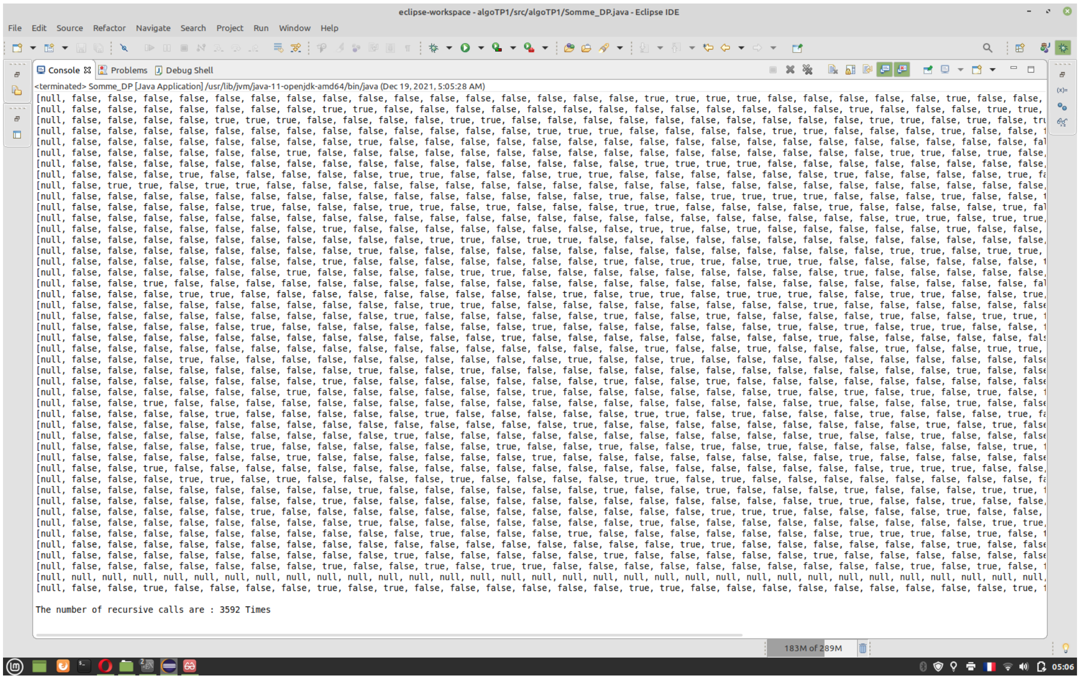

FIGURE B.6: n=40

Bibliography

[1] URL: http://www.math.stonybrook.edu/~scott/blair/Other_uses_subset_sum.html.

[2] Mohammud Z. Bocus et al. ?Per-Subcarrier Antenna Selection for H.264 MGS/CGS Video Transmission Over Cognitive Radio Networks? In: *IEEE Transactions on Vehicular Technology* 61.3 (2012), pp. 1060–1073. DOI: 10.1109/tvt.2011.2181550.

[3] Hoon Choi et al. ?Synthesis of application specific instructions for embedded DSP software? In: *IEEE Transactions on Computers* 48.6 (1999), pp. 603–614. DOI: 10.1109/12.773797.

[4] *Dynamic Programming - Subset Sum Problem*. 2021. URL: https://www.geeksforgeeks.org/subset-sum-problem-dp-25/.

[5] Shenshen Gu. ?A Finite-Time Convergent Recurrent Neural Network Based Algorithm for the L Smallest k-Subsets Sum Problem? In: *Advances in Neural Networks – ISNN 2013 Lecture Notes in Computer Science* (2013), pp. 19–27. DOI: 10.1007/978-3-642-39065-4_3.

[6] Rajeev Singh. *Subset Sum Problem*. 2021. URL: https://www.callicoder.com/subset-sum-problem/.

[7] Fabian Terh. *Solving the Target Sum problem with dynamic programming and more*. 2019. URL: https://medium.com/swlh/solving-the-target-sum-problem-with-dynamic-programming-and-more-b76bd2a661f9.

[8] Hong Wang, Zhiqiang Ma, and I. Nakayama. ?Effectiveness of penalty function in solving the subset sum problem? In: *Proceedings of IEEE International Conference on Evolutionary Computation* (). DOI: 10.1109/icec.1996.542401.