ENS 492 – Graduation Project (Implementation)

Final Report

Project Title: Efficient integer partition counter and generator

Group Members: Barış Akalın 27043

Supervisor(s): Kağan Kurşungöz

Date: 25/11/2023

- **ENS 491-492 Final Report must contain the following sections (leave section titles unchanged).**
- **You are required to write clearly and concisely: The main body of the report (not including the Title Page, Appendix, and References) must be between 20-25 pages.**
- **Project advisor may have additional requirements for the report.**

1. EXECUTIVE SUMMARY

An integer partition is the representation of a positive integer n as the sum of other positive integers that are sequenced in non-decreasing order. For example, 5 has 7 partitions:

5 = 5

5 = 4 + 1

5 = 3 + 2

5 = 3 + 1 + 1

5 = 2 + 2 + 1

5 = 2 + 1 + 1 + 1

5 = 1 + 1 + 1 + 1 + 1

Different constraints may be applied on partitions. For example, if the constraint is that the difference between the integers forming the partition must be at least 2, then 5 has 2 partitions.

5 = 5

5 = 4 + 1

Difference conditions can be expressed within square brackets. Additionally, difference conditions can be defined not only between consecutive 2 integers but also among more than 2 integers. For example, if the difference condition is [1], the difference between the integers forming the partition cannot be 1. If the difference condition is [0,2], 4 + 4 + 2 cannot be a partition of 10 because the difference between the first integer and the second integer is 0, and the difference between the second integer and the third integer is 2.

The goal of this project is to efficiently count and generate partitions related to Nandi's Identities. The definition of Nandi's Identities is as follows:

Let N be the set of partitions λ that satisfy both the following conditions:

(1) No sub-partition of λ satisfies the difference conditions [1], [0, 0], [0, 2], [2, 0] or [0, 3].

(2) No sub-partition with an odd weight satisfies the difference conditions [3, 0], [0, 4], [4, 0] or [3, $2*$, 3, 0] (where $2*$ indicates zero or more occurrences of 2).

We further define the following sets:

$N1 = \{\lambda \in N \mid m1(\lambda) = 0\}$, (5.1)

$N2 = \{\lambda \in N \mid m1(\lambda), m2(\lambda), m3(\lambda) \leq 1\}$, (5.2)

$N3 = \{\lambda \in N \mid m1(\lambda) = m3(\lambda) = 0, m2(\lambda) \leq 1, \lambda$ has no sub-partition of the form $(2k + 3) + 2k + (2k - 2) + \cdots + 4 + 2$ with $k \geq 1.\}$, (5.3)

Using this notation, we can state the following theorem.

Theorem 7 (Conjectured by Nandi [33], proved by Takigiku and Tsuchioka [44]). For any $n \geq 0$, we have the following three identities.

(1) The number of partitions of n belonging to N1 is the same as the number of partitions of n into parts congruent to ±2, ±3 or ±4 modulo 14.

(2) The number of partitions of n belonging to N2 is the same as the number of partitions of n into parts congruent to ±1, ±4 or ±6 modulo 14.

(3) The number of partitions of n belonging to N3 is the same as the number of partitions of n into parts congruent to ±2, ±5 or ±6 modulo 14. (Baker et al., 2022).

In summary, the number of partitions generated from the above section N1 and bulletpoint 1, section N2 and bulletpoint 2, section N3 and bulletpoint 3 are equal to each other. For example, 12 has 9 partitions according to n1 part and has 9 partitions according to bulletpoint 1. No matter which number is used for testing, equality will be observed in the number of partitions generated from N1 and bulletpoint 1.

N1 part:

12

10 + 2

9 + 3

8 + 4

7 + 5

6 + 6

8 + 2 + 2

6 + 4 + 2

6 + 3 + 3

(1) part:

12

10 + 2

4 + 4 + 4

4 + 4 + 2 + 2

4 + 3 + 3 + 2

3 + 3 + 3 + 3

4 + 2 + 2 + 2 + 2

3 + 3 + 2 + 2 + 2

2 + 2 + 2 + 2 + 2 + 2

2. PROBLEM STATEMENT

2.1. Objectives/Tasks

Six different codes will be written to count and generate the parts of N1, N2, and N3. Subsequently, a web interface will be designed to deliver these codes to users.

2.2. Realistic Constraints

Although there are applications such as Maxima and Maple that serve these purposes, the code used in these applications has high time and space complexity. The goal of the project is to write six different codes while ensuring that these codes are as efficient as possible in terms of both time and space complexity.

3. METHODOLOGY

In the generating algorithms, a nested for loop starts from the lowest integer (which is 2 in N1 and N3, 1 in N2). Integers that satisfy the difference conditions are added, while those that do not are skipped. This process continues until the loops are completed. While doing this, accepted sub-partitions are added to 2D array whose size are n which is the target integer and m which is the maximum number of parts that partition of n can have (n and m will be used in the below parts with the same meaning). Finally, all partitions of n from 1 to m number of parts that have been added to the 2D array are printed. This way, all partitions belonging to a specific integer are obtained.

In the counting algorithm of N1, a recursive function is used. In this recursive function, 5 different cases were used to satisfy all difference conditions. These cases include those starting with 2+2, excluding those starting with 2, those starting with 3+3, excluding those starting with 3, and those starting with 4 and above.

In partitions starting with 2-2, the first two 2's are subtracted, and then 4 is subtracted from the remaining ones. For example, let's consider the partitions of 20 that start with 2-2 and number of parts is 4. These are:

2 + 2 + 6 + 10

2 + 2 + 7 + 9

2 + 2 + 8 + 8

When the described operation is applied, the results are as follows:

(2-2) + (2-2) + (6-4) + (10 - 4) = 2 + 6

(2-2) + (2-2) + (7-4) + (9 - 4) = 3 + 5

(2-2) + (2-2) + (8-4) + (8 - 4) = 4 + 4

As seen, the results obtained are equal to the partitions of 8 with a number of parts equal to 2 (n = 8, m = 2). In summary, the general formula used for those starting with 2-2 is

$C(n - 4 - (m - 2) \times 4, m - 2)$.

In partitions starting with 2, 2 is excluded and 2 is subtracted from the remaining ones. Additionally, to avoid violating the [0,2] and [0,3] difference conditions, the results starting with 2-2 and 3-3 are subtracted from the outcome of this operation. In other words, when calculating partitions starting with 2, cases such as 2 + 4 + 4 and 2 + 5 + 5 are excluded, as they would violate the difference conditions. For example, let's consider the partitions of 20 that start with 2 (2-2's are excluded) and number of parts is 4. There is only one partition that satisfies this condition and it is:

2 + 4 + 6 + 8

When the described operation is applied, the results are as follows:

(2 - 2) + (4 - 2) + (6 - 2) + (8 - 2) = 2 + 4 + 6

It was supposed to be equal to the result for n = 12 and m = 3 but the result for n = 12 and m=3 are as follows:

2 + 2 + 8

2 + 4 + 6

3 + 3 + 6

The necessity of the subtraction operation becomes evident at this point. Although partitions starting with 2-2 and 3-3 are found for n=12 and m=3, these cases lead to unacceptable partitions like 2 + 4 + 4 and 2 + 5 + 5 if they originated from the case starting with 2. Therefore, such cases need to be excluded. In summary, the general formula used for those starting with 2-2 is C(n- 2- (m-1)*2, m-1) and the subtraction of partitions starting with 2-2 and 3-3 from the result obtained from this formula.

In partitions starting with 3-3, the first two 3's are subtracted, and then 4 is subtracted from the remaining ones. Those remaining results that start with 2-2 (such as 3 + 3 + 6 + 6, which violates the [0,3] difference condition) and those starting with 3 (such as 3 + 3 + 7, which violates the [0,4] difference condition) are then excluded. Additionally, the [3, 2*, 3, 0] difference condition needs to be handled in the 3-3 case. Cases like 3 + 3 + 6 + 9, 3 + 3 + 6 + 8 + 11, 3 + 3 + 6 + 8 + 10 + 13, ..., need to be found until they fit within the given n and m, and all such cases should be subtracted too. In summary, the general formula used for those starting with 2-2 is C(n-6-(m-2)*4,m-2) and the subtraction of partitions starting with 2-2, 3 and [3,2*,3,0] from the result obtained from this formula.

In partitions starting with 3, 3 is subtracted and 2 is subtracted from the remaining ones. Those remaining results that start with 2 and 2-2 (such as 3 + 4, which violates the [1] difference condition), those starting with 3-3 (such as 3 + 5 + 5, which violates the [0,2] difference condition), those starting with 4-4 (such as 3 + 6 + 6, which violates the [0,3] difference condition) and those starting with 5-5 (such as 3 + 7 + 7, which violates the [0,4] difference condition) are then excluded. . In summary, the general formula used for those starting with 3 is C(n-3-(m-1)*2,m-1) and the subtraction of partitions starting with 2, 2-2, 3-3, 4-4 and 5-5 from the result obtained from this formula.

For those starting with an integer of 4 and above, 2 is subtracted from all integers. For example, the number of parts for 20 with a count of 4, starting with an integer of 4 and above, is given by n = 20 - 4*2 = 12 and m = 4, which is equal to 0. When we combine these mentioned 5 cases, we obtain the general recursive formula which is C(n,m) = C(n-4-(m-2)*4,m-2) + C(n-2-(m-1)*2,m-1) + C(n-6-(m-2)*4,m-2) + C(n-3-(m-1)*2,m-1) + C(n-m*2,m).

Additionally, for each formula, the subtraction operations mentioned above are applied. In this way, the results obtained from the formulas are added together to reach the total count.

In the counting algorithm of N2, a similar approach to the one in the counting algorithm of N1 is adopted. Unlike N1, in N2, we can use the integers 1, 2, and 3 at most once. Therefore, in addition to the cases starting with 2 in N1, such as those starting with 2 + 2, cases starting with 1 and cases starting with 1 + 3 should also be added. For cases starting with 1, 1 should be excluded, and then 2 should be subtracted from the remaining integers. In other words, the formula for this case is C(n-1-(m-1)*2,m-1). For cases starting with 1 + 3, 1 and 3 should be excluded, and then 2 should be subtracted from the remaining integers. In other words, the formula for this case is C(n-4-(m-2)*2,m-2).

Since 1, 2, and 3 can only be used once each, the cases starting with 1, those starting with 2, those starting with 3, and those starting with 1 + 3 are used in the recursive function only in the first step. For the remaining steps, the recursive function used in N1 can be used. In other words, the general formula for N1 was referred as C(n, m), and the general formula for N2 can be referred as D(n, m), and D(n, m) can be expressed in terms of C(n, m). The general formula for N2 is D(n,m) = C(n-1-(m-1)*2,m-1) + C(n-4-(m-2)*2,m-2) + C(n-2-(m-1)*2,m-1) + C(n-3-(m-1)*2,m-1) + C(n-m*2,m). As can be seen, the formulas for cases starting with 2 + 2 and 3 + 3 in N1 are not present in the general formula of N2, as integers 2 and 3 can be used at most once.

In the counting algorithm of N3, a similar approach to the one in the counting algorithm of N1 and N2 are adopted again. If the general formula for N3 is named as E(n,m), then, similar to N2, N3 can be expressed in terms of N1, that is, E(n,m) can be represented as C(n,m). Since the integers 1 and 3 cannot be used in N3, the case starting with 2 and the case subtracting 2 from each integer forming the partition will be sufficient for the E(n,m) formula in N3. However, in addition to these cases, there is a case similar to the [3, 2*, 3, 0] case, which is no subpartition of the form $(2k + 3) + 2k + (2k - 2) + \cdots + 4 + 2$ with $k \geq 1$. Therefore, similar to the subtraction process in the [3, 2*, 3, 0] case, these cases need to be found until they fit within the given n and m, and all such cases should be subtracted. Therefore, the general formula for N3 is E(n,m) = C(n-m*2,m,0) + C(n-2-(m-1)*2,m-1,2) - mentioned subtraction.

The mentioned 6 algorithms were implemented in Python and the created source codes were transferred to a website built using the Flask which is a Python framework. The website features a select bar to determine which of the 6 algorithms to use, an input field for the user to enter the integer for which they want to find partitions, and two buttons—one to directly print the output on the site and the other to download the output as a .csv file.

4. RESULTS & DISCUSSION

All the objectives in the project, which include writing counting and generating algorithms for Nandi's identities in the N1, N2, and N3 cases, have been achieved, and these algorithms have been made accessible to users on a website. While the identities which are the topic of previous term project for Rogers-Ramanujan, Rogers-Ramanujan-Gordon, and Capparelli's identities had rules regarding the differences between at most two consecutive integers (e.g., requiring a minimum difference of 2 between consecutive integers in Rogers-Ramanujan Identities), however, in Nandi's identities, unlike the other identities, there are difference conditions between 3 integers (e.g., the partition 2 + 5 + 5 is not included due to not satisfying the [0,3] difference condition). The development of algorithms for these conditions as part of this project can be considered a contribution to the previous state-of-the-art.

5. IMPACT

Although it is not within the scope of this project, all developed algorithms have been made available to users through a website. Hence, this service can also be utilized for entrepreneurial purposes. For example, the website could be turned into a paid service, or revenue could be generated through advertisements placed on the website. There is no Freedom-to-Use issue.

6. ETHICAL ISSUES

There are no ethical issues.

7. PROJECT MANAGEMENT

As mentioned in progress report 2, counting and generating algorithms for N1 were developed initially. Subsequently, algorithms for N2 and N3 were developed in a manner similar to those for N1. The website was then created after the development and coding of all these algorithms in Python.

8. CONCLUSION AND FUTURE WORK

It is possible to continue this project. Algorithms for identities with difference conditions between 3 integers, as opposed to 2 in the previous project, have been developed and the previous project have been continued. Similarly, algorithms for identities with difference conditions involving more than 3 integers or different difference conditions from those in Nandi's identities can be developed and this project can be continued.

9. APPENDIX

Website link: https://nandisidentities.pythonanywhere.com/

10. REFERENCES

Baker, K., Kanade, S., Russell, M. C., & Sadowski, C. (2022, August 31). *Principal subspaces of basic modules for twisted affine lie algebras, q-series multisums, and Nandi's identities*. arXiv.org. https://doi.org/10.48550/arXiv.2208.14581