Integer Partitions

&

Bounded Integer Partition

Tomer Alon

Pace University

Abstract

This paper explores the notion of Integer Partition, or more specifically, partitioning a positive integer *n* into *k* parts. I will illustrate an example of Integer Partition by distributing *n* unlabeled balls into *k* unlabeled urns. The topic of the Bounded Integer Partition problem is also introduced in this paper, but is not as heavily focused on as is the general idea of Integer Partition. In addition, this paper will also cover a Java program designed to calculate the partitioning of *n* into *k* parts in two (2) different methods, while also calculating the overall number of partitions of *n*, and printing out those partitions as different sums.

*Keywords*: Integer Partition, Bounded Integer Partition, Recursion & Dynamic Programming.

Integer Partition

Integer Partition, also denoted as *P(n)*, or *P(n,k)*, is another way of representing a particular positive integer as addition of other positive integers. The notation *P(n,k)* denotes the number of ways of writing *n* as a sum of exactly *k* terms, or equivalently, the number of partitions into parts of which the largest is exactly *k*. For instance, *P(*5,3*)* = 2, since the partitions of 5 of length 3 are (3,1,1) and (2,2,1), which can be represented as 3 + 1 + 1, and 2 + 2 + 1.

Dickson (2005) notes that another way of defining the partition of a positive number *n* is “as a solution of the Diophantine equation:

1 \* *x*1 + 2 \* *x*2 + 3 \* *x*3 + … + *n \* xn* = *n*

where the partitions of *n* include the sums corresponding to the solutions.” It should be mentioned that every permutation of *k* is the same; (2 + 2 +1) = (1 + 2 + 2) = (2 +1 + 2), and thus only the sorted lists are included in the output set of *P(n,k)* (S. Cha, 2011).

The recursive formula for integer partitioning is *P(n,k)* = *P(n-1, k-1)* + *P(n-k, k)*, with *P(n,k)* = 0 for *k* > *n*, and *P(n,k)* = 1 for *k* = *n* or *k =* 1.



By convention, partitions are usually ordered from largest to smallest (Skiena, 1990, p. 51). The integer four (4) can be written as:

4 = 4

= 3 + 1

= 2 + 2

= 2 + 1 + 1 and

= 1 + 1 + 1 + 1.

It thus follows that *P(*4*)*  = 5 (N. Calkin, J. Davis, K. James, E. Perez, & C. Swannack, 2007). *P(n)* is usually referred to as the number of unrestricted partitions. I will cover some of what is called a “Bounded Integer Partition,” a little later. For this project in particular, I only handled the unrestricted Integer Partition in Java. More on my Java program later as well.

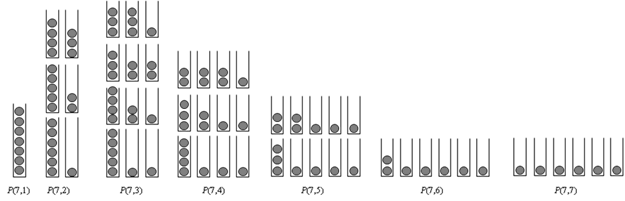
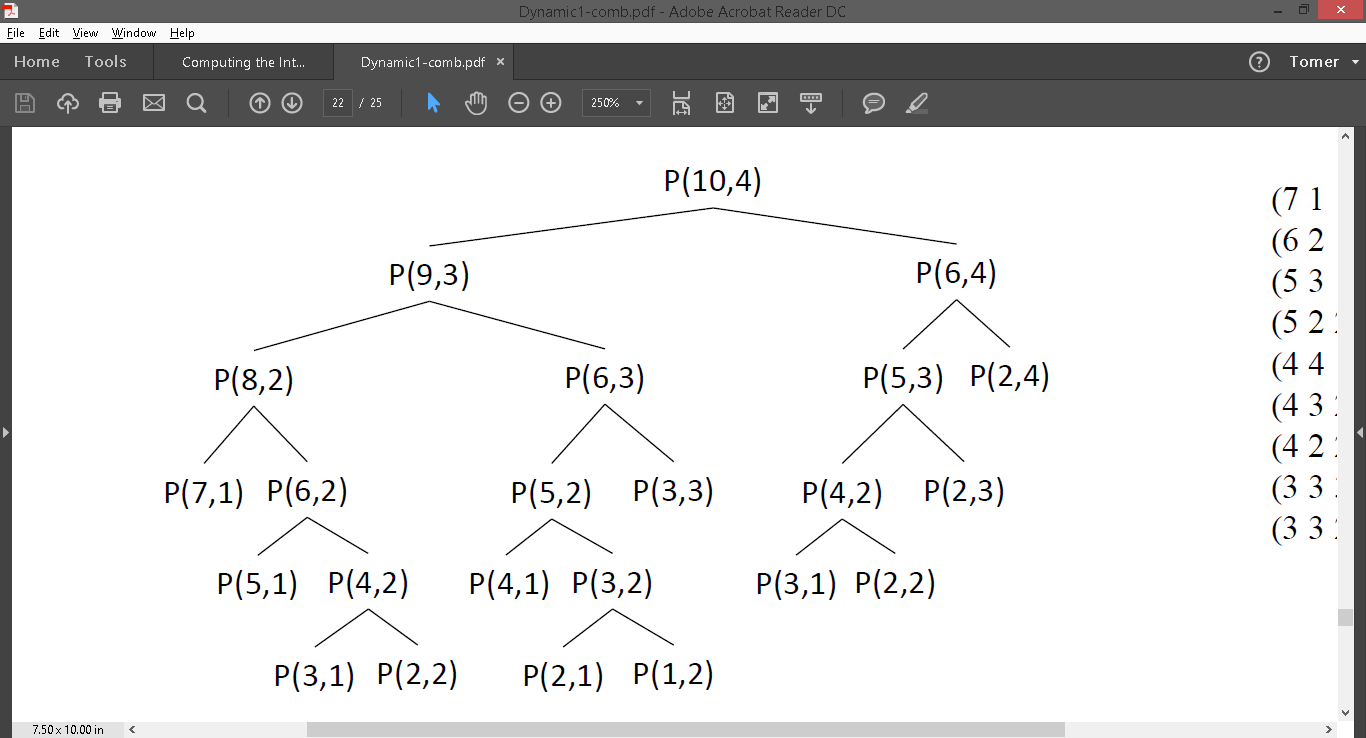
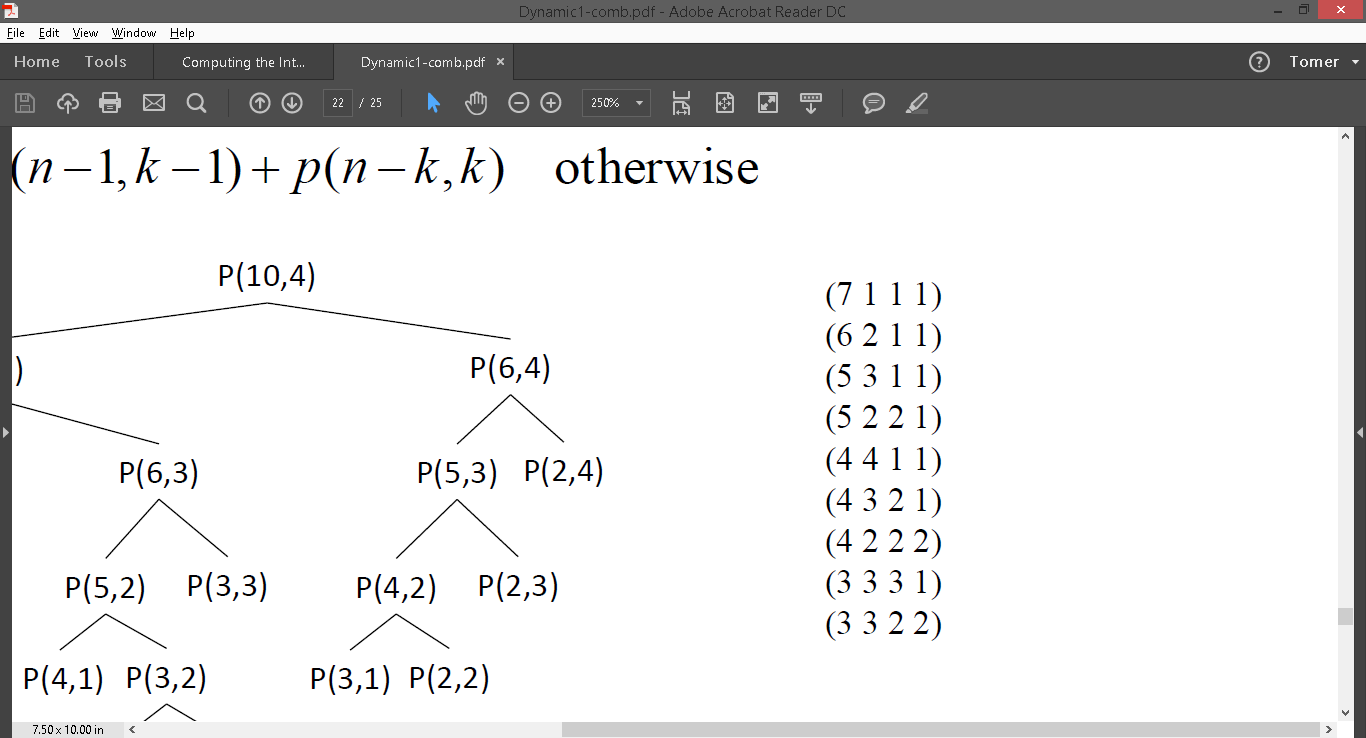
Quite a well-known problem in the field of mathematics, specifically in combinatorics, Integer Partitioning– part of the overall partitioning problems (int. Partitions, Set Partitions, etc.) can be visualized when trying to distribute *n* unlabeled balls into *k* unlabeled urns (Knuth, D. E., 2005).

Figure 1(a) Integer Partition of 7 *P(*7*)*.

|  |  |
| --- | --- |
|  |  |
| Figure 1(b) *p*(*n*,*k*) triangle. | Figure 1(c) *p*(7,3) = *p*(6,2) + *p*(4,3). |

The first recursive equation given in this paper is represented visually in figure 1. It is easier to understand the problems and equations when seen in a particular instance, rather than in their original form. In figure 1(b), each slot in the triangle is representative of the number of partitions of integer *n*. The last row shows the number of partitions for *n* = 10, or *k* partitions of that integer. *P(*10*)* = 42, and *P(*10,4*)* = 9. Likewise, Figure 1(c) is an example of representing *k* for a certain integer *n*, by adding two different *k*’s for two different *n*’s; *P(*6,2*)* + *P(*4,3*)* = 3 + 1 = *P(*7,3*)* = 4. Another way to illustrate Integer Partition can be done using a tree and extended branches. We will use the example of *P(*10,4*)*:

If we go back to the example of using unlabeled balls in unlabeled urns to illustrate Integer Partition, we can utilize the same example to also illustrate a Bounded Integer Partition. Suppose that each urn’s capacity is *b*, meaning that no urn can have more than *b* number of balls in it; this condition represents an upper Bounded Integer Partition. The recursive formula for an upper Bounded Integer Partition coefficient is:



Figure 3 is a triangle visualization of upper Bounded Integer Partition coefficients for *b* = 4 and 5 respectively.

|  |  |
| --- | --- |
| I4 | I5 |
| (a) *p*4(*n*,*k*) triangle | (b) *p*5(*n*,*k*) distribution |

Figure 3 Upper Bounded surjective Integer Partition coefficient triangles.

The following equation for an Integer Partition takes into account both the lower bound, *a*, and the upper bound, *b*, of an urn’s capacity:



For any upper bound Integer Partition where *b* ≥ *k,* the result would be the same as for an unbounded Integer Partition. For example, in the case of *P*[1, ∞]*(n,k)*, where the upper bound is ∞, the partition calculated equals to that of *P(n,k)*. The following table provides evidence for this claim:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *P*[1,∞](*n*,*k*) coefficient table. | | | | | | | | | | | |
| *n \ k* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3 | **1** | **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | **1** | **2** | **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 5 | **1** | **2** | **2** | **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 6 | **1** | **3** | **3** | **2** | **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 | |
| 7 | **1** | **3** | **4** | **3** | **2** | **1** | **1** | 0 | 0 | 0 | 0 | 0 | |
| 8 | **1** | **4** | **5** | **5** | **3** | **2** | **1** | **1** | 0 | 0 | 0 | 0 | |
| 9 | **1** | **4** | **7** | **6** | **5** | **3** | **2** | **1** | **1** | 0 | 0 | 0 | |
| 10 | **1** | **5** | **8** | **9** | **7** | **5** | **3** | **2** | **1** | **1** | 0 | 0 | |

The table’s values represent the same coefficients from the triangle in Figure 1(b).

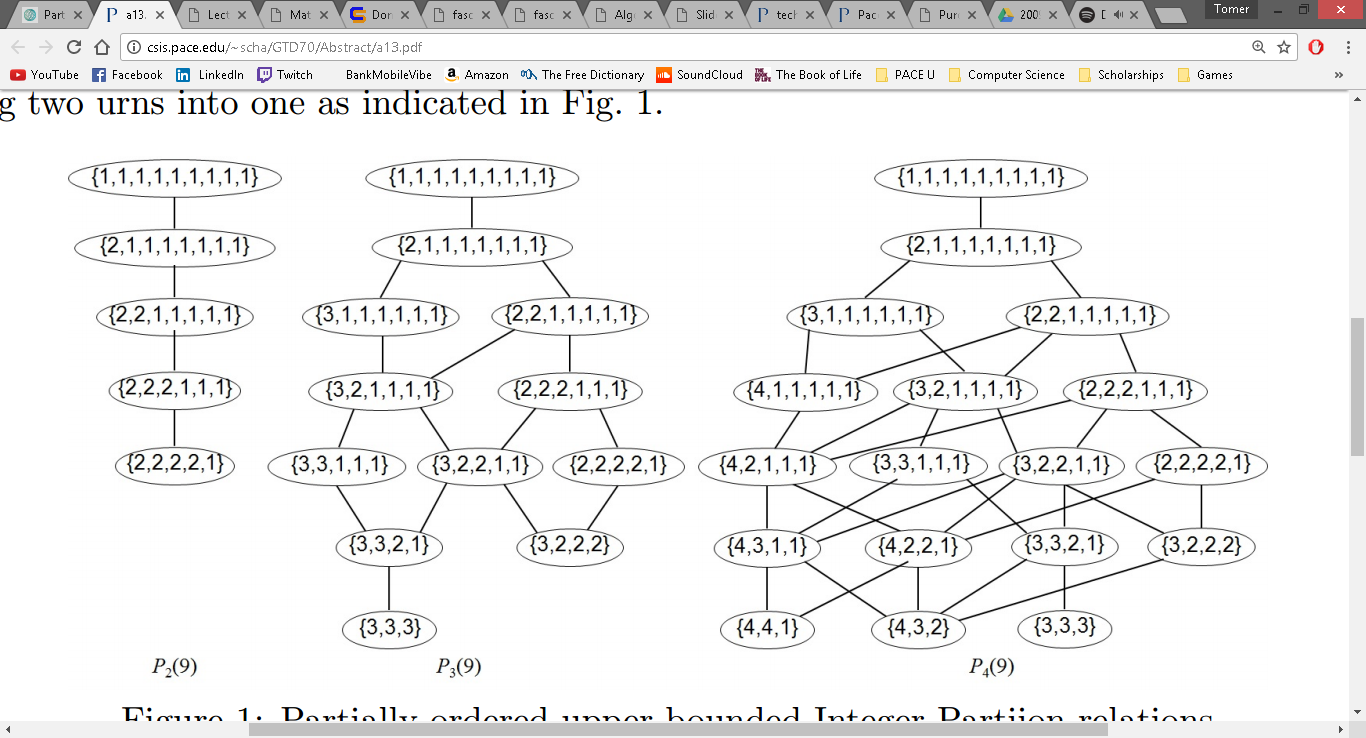
When talking about *partially ordered relation* on Integer Partition, we deal with a level graph, in which the *k*th level has *Pb(n,k)* number of nodes. The edges from *k* to *k* + 1 level indicate splitting an urn to two and those from *k* + 1 to *k* level mean merging the two urns into one (Martini, A. C., Lee, J. I., & Simnica, B., 2015). Visually, the tree would look like this:

Figure 4 Partially ordered upper Bounded Integer Partition relations (Martini, A. C., Lee, J. I., & Simnica, B., 2015).

Martini, Lee, and Simnica have provided another recursive equation for an Integer Partition with lower and upper bounds:



**Discussion**

Learning about Integer Partitions has been a great opportunity to dive deeper into the complex, yet interesting set of mathematical and computer science-oriented problems. Nevertheless, my research about Integer Partition and Bounded Integer Partition is far from over– there are so many more aspects of these combinatorics problems to be analyzed.

**Limitations of These Studies**

While I would have very much enjoyed exploring the specifics of Bounded Integer Partitions, and even attempt making a Java program for computing the partitions’ outputs, there are a few challenges for this kind of task. Firstly, there is never ‘enough’ research to be done when it comes to this topic, as the idea of Partitions is still being studied to this day, where mathematicians and computer scientists alike are still looking for ways to improve the already-discovered equations and algorithms. There are many terminologies and concepts behind Partitions that I am not yet familiar with, and so attempting to explain the topic is hard for me. My current study is *enough* for the basic understanding of Integer Partitions, but still lacks a solid foundation for the Bounded Integer Partition aspect. Of course, I was able to come up with a Java program that would calculate Integer Partitions, but trying to do the same with a certain boundary has been proven quite difficult. Regardless, I am confident enough to continue this study on my own to expand my knowledge on the topic.

**Partitionk.java**

Problem: Integer Partition *P(n,k)*

Input: Any positive integer for *n* & *k*

Output: *P(n,k)*

For this Java program, I implemented two of the functions previously used in class as an assignment. The functions were a recursive and a dynamic approach to calculating Stirling Numbers of the Second Kind. Using the recursive definition of Integer Partition, I simply changed the equations. While scouting the internet for some help with the Integer Partition, I stumbled upon two functions that helped me build up the program to do more than just find *P(n,k)*. Both functions obtained from the internet were commented within the code to discern between the two. The first one, from *Stackexchange.com* utilizes a dynamic approach to calculate the total number of partitions for a certain *n*. The second function, found on Princeton’s notes for Java class, prints out all the partitions for *n*, going from one (1) integer to *n* integers (1 + 1 + 1 + 1 + 1 + … + *n*). Upon launching the program, the user needs to input two integer values: *n* and *k*. Within seconds the program outputs *P(n,k)* for both methods (recursive and dynamic), while also printing the number of total partitions, as well as the integers of which the sum is equal to *n*. In the case that a large number for *n* is inserted, the list of the different sums of integers will flow the console; therefore, a bypass I had thought of would be to print out the output into a txt file– an idea worth exploring, but have not had time to do so.

**Conclusion**

Going forth with this study, I have learned, and expanded my knowledge, on Integer Partition. Although I feel I have learned enough to explain the fundamentals of Integer Partition, there is more research to be done on the subject, and even on the algorithms derived after it. The topic is a wonderful candidate for future research; I would want to learn more about topic, but even more about the programming that is inspired from it. It is within my goals to be able to clearly understand every line of code I encounter, and be able to utilize it and implement it more efficiently.

References

Calkin, N., Davis, J., James, K., Perez, E., & Swannack, C. 2007. Computing the Integer Partition Function. *Mathematics of Computation,* *76*(259), 1619-1638.

Cha, S.-H. 2011. Recursive Algorithm for Generating Partitions of an Integer. *Technical Report*. New York.

Dickson, L. E., 2005. Partitions. *History of the Theory of Numbers, Vol. 2. Ch. 3: Diophantine Analysis.* New York: Dover, p. 101-164.

Knuth, D. E. 2005. The Art of Computer Programming *– vol 4, Fascicle 3: Generating All Combinations and Partitions*.

Knuth, D. E. 2006. The Art of Computer Programming *– vol 4, Fascicle 4: History and Further References.*

Martini, A. C., Lee, J. I., & Simnica, B., 2015. Partially Ordered Bounded Integer Partition and Computing Its Properties with Memoization Technique.

Sedgewick, R., & Wayne, K. 2016. Partition.java. Retrieved from Princeton.edu

Skiena, S. 1990. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica.* Reading, MA: Addison-Wesley.

IntegerPartition.java. Retrieved from:

https://codegolf.stackexchange.com/questions/812/partition-of-a-positive-integer-n