

Efficient Generation of Set Partitions

Michael Orlov
orlovm@cs.bgu.ac.il

March 26, 2002

Abstract

This paper shows how to implement a set partitions generator, where each iteration has constant space and amortized time complexity. This is achieved by defining a notation for set partitions, from which following and preceding partitions are easily derivable. The method is not novel — the purpose of this paper is to aid programmers in implementing this task. Example implementation in C++ is available.

1 Theoretical foundation

In order to justify using special notation for a partition of a set, and explain the notation at the same time, we prove a simple lemma.

Lemma 1.1. *There is a bijection between all partitions of the set $\{x_1, \dots, x_n\}$ and the set¹*

$$\{\langle k_1, \dots, k_n \rangle : \forall 2 \leq i \leq n : k_1 \leq k_i \leq \max_{1 \leq j < i} k_j + 1, k_i \in \mathbb{Z}\} \quad (1.1)$$

for all $n \in \mathbb{N}$ and $k_1 = \text{const} \in \mathbb{Z}$.

Proof. Below, we pick $k_1 = 1$.

For $n = 0$ both sets are empty, and the lemma holds; so we continue with positive n 's.

Assume there are two different partitions \mathcal{P}_1 and \mathcal{P}_2 of $A = \{x_1, \dots, x_n\}$. We now sort \mathcal{P}_1 's and \mathcal{P}_2 's subsets on their elements lowest index, and enumerate these subsets starting with k_1 . Then, we define k_i to be the number assigned to the subset where x_i resides. For example,

$$\begin{aligned} \mathcal{P}_1 &= \{\{x_5, x_3\}, \{x_1, x_2\}, \{x_7, x_6, x_4\}\} \\ &= \langle \overbrace{\{x_1, x_2\}}^1, \overbrace{\{x_3, x_5\}}^2, \overbrace{\{x_4, x_6, x_7\}}^3 \rangle && (1 < 3 < 4) \\ \kappa_1 &= \langle 1, 1, 2, 3, 2, 3, 3 \rangle && k_i \text{ for each } x_i \end{aligned}$$

The sequence κ conforms to conditions stated in the lemma, since if it wouldn't, there would be a k_i such that $k_i > \max_{1 \leq j < i} k_j + 1$, implying a subset with

¹Elements of this set are known as *Restricted Growth strings*.

index lower than k_i not containing any of $\langle x_1, \dots, x_i \rangle$. This contradicts sorting subsets on their elements lowest index.

Thus, it remains to show that applying this algorithm to \mathcal{P}_1 and \mathcal{P}_2 will result in different sequences $\kappa_1 \neq \kappa_2$. This follows from each partition \mathcal{P} being reconstructible from its corresponding κ .

To complete the proof, we need to show that each sequence $\{k_1, \dots, k_n\}$ in (1.1) corresponds to some partition of $\{x_1, \dots, x_n\}$. This is easy to see — all we do is start with subset indexed k_1 , and then move on the sequence, when upon encountering k_i we either adjoin x_i to existing subset, or create a new one (if k_i is “new”). For example:

$$\begin{aligned} \kappa &= \langle 1, 2, 3, 2, 4, 4, 2, 5, 3, 3 \rangle \\ &= \langle \{x_1\}, \{x_2, x_4, x_7\}, \{x_3, x_9, x_{10}\}, \{x_5, x_6\}, \{x_8\} \rangle \\ \mathcal{P} &= \{ \{x_1\}, \{x_2, x_4, x_7\}, \{x_3, x_9, x_{10}\}, \{x_5, x_6\}, \{x_8\} \} \end{aligned}$$

□

2 Algorithms

We now present two algorithms for sequence initialization, and another two that consecutively iterate over all sequence values allowed by Lemma 1.1, for incrementing and decrementing κ . In the algorithms, the following invariant is preserved:

$$m_i = \max\{k_0, \dots, k_i\} \quad (2.1)$$

Alg. 1 and Alg. 2 initialize first and last partition sequences in the enumeration. These algorithms pick $k_0 = 0$ (which is k_1 in Lemma 1.1), so that κ is usable as an array of indexes into a set.

Algorithm 1 INITIALIZE-FIRST(κ, M)

Require: $\kappa = \langle k_0, \dots, k_{n-1} \rangle, M = \langle m_0, \dots, m_{n-1} \rangle$

Ensure: κ and M correspond to the first partition sequence for any set of size n .

- 1: **for** $i = 0$ to $n - 1$ **do**
 - 2: $k_i \leftarrow m_i \leftarrow 0$
-

Algorithm 2 INITIALIZE-LAST(κ, M)

Require: $\kappa = \langle k_0, \dots, k_{n-1} \rangle, M = \langle m_0, \dots, m_{n-1} \rangle$

Ensure: κ and M correspond to the last partition sequence for any set of size n .

- 1: **for** $i = 0$ to $n - 1$ **do**
 - 2: $k_i \leftarrow m_i \leftarrow i$
-

Calls to Alg. 3 and Alg. 4, which iterate over partition sequences, run in constant memory and amortized constant time (if iteration is done in single direction). They do not assume that $k_0 = 0$.

Finally, Alg. 5 returns size of the partition represented by some κ and M .

Algorithm 3 NEXT-PARTITION(κ, M)

Require: $\kappa = \langle k_0, \dots, k_{n-1} \rangle$ is a valid partition sequence for some set $\{x_0, \dots, x_{n-1}\}$, and $M = \langle m_0, \dots, m_{n-1} \rangle$ is the corresponding maximum sequence, according to (2.1).

Ensure: κ and M are advanced to represent the following partition in some constant enumeration of partitions of the set, unless κ is the last partition in the enumeration.

```
1: ▷ We never test  $k_0$  against  $m_0$ 
2: for  $i = n - 1$  to 1 do
3:   if  $k_i \leq m_{i-1}$  then
4:      $k_i \leftarrow k_i + 1$ 
5:      $m_i \leftarrow \max(m_i, k_i)$ 
6:     for  $j = i + 1$  to  $n - 1$  do
7:        $k_j \leftarrow k_0$ 
8:        $m_j \leftarrow m_i$ 
9:   return
10: fail
```

Algorithm 4 PREVIOUS-PARTITION(κ, M)

Require: $\kappa = \langle k_0, \dots, k_{n-1} \rangle$ is a valid partition sequence for some set $\{x_0, \dots, x_{n-1}\}$, and $M = \langle m_0, \dots, m_{n-1} \rangle$ is the corresponding maximum sequence, according to (2.1).

Ensure: κ and M are advanced to represent the preceding partition in some constant enumeration of partitions of the set, unless κ is the first partition in the enumeration.

```
1: ▷ We never test  $k_0$  against  $k_0$ 
2: for  $i = n - 1$  to 1 do
3:   if  $k_i > k_0$  then
4:      $k_i \leftarrow k_i - 1$ 
5:      $m_i \leftarrow m_{i-1}$ 
6:     for  $j = i + 1$  to  $n - 1$  do
7:        $k_j \leftarrow m_j \leftarrow m_i + j - i$ 
8:   return
9: fail
```

Algorithm 5 PARTITION-SIZE(M)

Require: $M = \langle m_0, \dots, m_{n-1} \rangle$ is the maximum sequence that corresponds to some κ .

Ensure: Partition size is returned.

```
1: return  $m_{n-1} - m_0 + 1$ 
```

3 Examples

Initializing κ and M with INITIALIZE-FIRST for $n = 4$, and then executing NEXT-PARTITION until it fails, would produce the partitions shown in Tab. 1.

Iteration	κ	Partition size	Partition
1	$\langle 0, 0, 0, 0 \rangle$	1	$\{\{a, b, c, d\}\}$
2	$\langle 0, 0, 0, 1 \rangle$	2	$\{\{a, b, c\}, \{d\}\}$
3	$\langle 0, 0, 1, 0 \rangle$	2	$\{\{a, b, d\}, \{c\}\}$
4	$\langle 0, 0, 1, 1 \rangle$	2	$\{\{a, b\}, \{c, d\}\}$
5	$\langle 0, 0, 1, 2 \rangle$	3	$\{\{a, b\}, \{c\}, \{d\}\}$
6	$\langle 0, 1, 0, 0 \rangle$	2	$\{\{a, c, d\}, \{b\}\}$
7	$\langle 0, 1, 0, 1 \rangle$	2	$\{\{a, c\}, \{b, d\}\}$
8	$\langle 0, 1, 0, 2 \rangle$	3	$\{\{a, c\}, \{b\}, \{d\}\}$
9	$\langle 0, 1, 1, 0 \rangle$	2	$\{\{a, d\}, \{b, c\}\}$
10	$\langle 0, 1, 1, 1 \rangle$	2	$\{\{a\}, \{b, c, d\}\}$
11	$\langle 0, 1, 1, 2 \rangle$	3	$\{\{a\}, \{b, c\}, \{d\}\}$
12	$\langle 0, 1, 2, 0 \rangle$	3	$\{\{a, d\}, \{b\}, \{c\}\}$
13	$\langle 0, 1, 2, 1 \rangle$	3	$\{\{a\}, \{b, d\}, \{c\}\}$
14	$\langle 0, 1, 2, 2 \rangle$	3	$\{\{a\}, \{b\}, \{c, d\}\}$
15	$\langle 0, 1, 2, 3 \rangle$	4	$\{\{a\}, \{b\}, \{c\}, \{d\}\}$

Table 1: Partitions for the set $\{a, b, c, d\}$, in the order produced by NEXT-PARTITION

One can check that number of partitions of size k equals $S(4, k)$, where $S(n, k)$ are Stirling numbers of the Second kind, shown in Tab. 2.

k	0	1	2	3	4	5	6	7	8	9	10
p											
0	1										
1	0	1									
2	0	1	1								
3	0	1	3	1							
4	0	1	7	6	1						
5	0	1	15	25	10	1					
6	0	1	31	90	65	15	1				
7	0	1	63	301	350	140	21	1			
8	0	1	127	966	1701	1050	266	28	1		
9	0	1	255	3025	7770	6951	2646	462	36	1	
10	0	1	511	9330	34105	42525	22827	5880	750	45	1

Table 2: Stirling numbers $S(n, k)$ of the Second kind, up to $n = 10$. Note that $S(n, k) = S(n-1, k-1) + kS(n-1, k)$.

4 Algorithms for partitions of size p

What if we only want to iterate over partitions with given number of subsets? It turns out that it is not too hard to modify the algorithms in Sec. 2 to uphold

this restriction.

Alg. 6 and Alg. 7 initialize first and last partition sequences in the p -restricted enumeration (again, as in Alg. 1 and Alg. 2, we pick $k_0 = 0$).

Algorithm 6 p -INITIALIZE-FIRST(κ, M, p)

Require: $\kappa = \langle k_0, \dots, k_{n-1} \rangle, M = \langle m_0, \dots, m_{n-1} \rangle, 1 \leq p \leq n$ **Ensure:** κ and M correspond to the first partition sequence for any set of size n , with partitions containing p subsets.

- 1: **for** $i = 0$ to $n - p$ **do**
 - 2: $k_i \leftarrow m_i \leftarrow 0$
 - 3: **for** $i = n - p + 1$ to $n - 1$ **do**
 - 4: $k_i \leftarrow m_i \leftarrow i - (n - p)$
-

Algorithm 7 p -INITIALIZE-LAST(κ, M, p)

Require: $\kappa = \langle k_0, \dots, k_{n-1} \rangle, M = \langle m_0, \dots, m_{n-1} \rangle, 1 \leq p \leq n$ **Ensure:** κ and M correspond to the last partition sequence for any set of size n , with partitions containing p subsets.

- 1: **for** $i = 0$ to $p - 1$ **do**
 - 2: $k_i \leftarrow m_i \leftarrow i$
 - 3: **for** $i = p$ to $n - 1$ **do**
 - 4: $k_i \leftarrow m_i \leftarrow p - 1$
-

Alg. 8 and Alg. 9 are the p -restricted counterparts of Alg. 3 and Alg. 4. They still run in constant memory, but the amortized time complexity is not as clear.² These algorithms also assume that $k_0 = 0$, for clarity.

Thus, invoking p -INITIALIZE-FIRST($\kappa, M, 3$) for $n = 4$, followed by iterations of p -NEXT-PARTITION until it fails, will produce results equivalent to lines 5, 8, 11, 12, 13 and 14 of Tab. 1, in the same order.

A C++ implementation

An example source code for partitions iteration accompanies this paper, and should be available at the same location as the article. `partition.[hC]` implement the iterators, and `user.C` is the testing user front-end. As well, a program that generates Stirling numbers of the Second kind, `stirling.C`, is provided. All the sources are in public domain.



²Measuring `for` loops suggests that amortized time complexity is constant if $\exists x : p/n \leq x < 1$, otherwise it is at most linear. We do not formally analyze time complexities in this section, since the difference between amortized constant and linear times does not usually matter in practical applications (e.g., if a partition is constructed according to each κ , it already takes linear time).

Algorithm 8 p -NEXT-PARTITION(κ, M)

Require: $\kappa = \langle k_0, \dots, k_{n-1} \rangle$ is a valid partition sequence for some set $\{x_0, \dots, x_{n-1}\}$, and $M = \langle m_0, \dots, m_{n-1} \rangle$ is the corresponding maximum sequence, according to (2.1).

Ensure: κ and M are advanced to represent the following partition of *the same size* in some constant enumeration of such partitions of the set, unless κ is the last partition in the enumeration.

```

1:  $p \leftarrow \text{PARTITION-SIZE}(M)$ 
2: for  $i = n - 1$  to 1 do
3:   if  $k_i < p - 1 \wedge k_i \leq m_{i-1}$  then
4:      $k_i \leftarrow k_i + 1$ 
5:      $m_i \leftarrow \max(m_i, k_i)$ 
6:     for  $j = i + 1$  to  $n - (p - m_i)$  do
7:        $k_j \leftarrow 0$ 
8:        $m_j \leftarrow m_i$ 
9:     for  $j = n - (p - m_i) + 1$  to  $n - 1$  do
10:       $k_j \leftarrow m_j \leftarrow p - (n - j)$ 
11:   return
12: fail

```

Algorithm 9 p -PREVIOUS-PARTITION(κ, M, p)

Require: $\kappa = \langle k_0, \dots, k_{n-1} \rangle$ is a valid partition sequence for some set $\{x_0, \dots, x_{n-1}\}$, and $M = \langle m_0, \dots, m_{n-1} \rangle$ is the corresponding maximum sequence, according to (2.1).

Ensure: κ and M are advanced to represent the preceding partition of *the same size* in some constant enumeration of such partitions of the set, unless κ is the first partition in the enumeration.

```

1:  $p \leftarrow \text{PARTITION-SIZE}(M)$ 
2: for  $i = n - 1$  to 1 do
3:   if  $k_i > 0 \wedge p - m_{i-1} \leq n - i$  then
4:      $k_i \leftarrow k_i - 1$ 
5:      $m_i \leftarrow m_{i-1}$ 
6:     for  $j = i + 1$  to  $i + (p - m_i) - 1$  do
7:        $k_j \leftarrow m_j \leftarrow m_i + j - i$ 
8:     for  $j = i + (p - m_i)$  to  $n - 1$  do
9:        $k_j \leftarrow m_j \leftarrow p - 1$ 
10:   return
11: fail

```