A CAT Algorithm for Sand Piles[‖]

P. Massazza

Università degli Studi dell'Insubria, Dipartimento di Informatica e Comunicazione,
Via Mazzini 5, 21100 Varese, Italy
paolo.massazza@uninsubria.it

Abstract

We present a CAT (Constant Amortized Time) algorithm for the exhaustive generation of the sand piles in the lattice SPM(n). More precisely, given an integer n, we show that the sequence of the sand piles in SPM(n) (ordered with respect to the negative lexicographic ordering) can be generated in $O(1)$ amortized time by using $O(n)$ space.

1 Introduction

In this paper, we consider the problem of generating particular integer partitions that are called *sand piles*. Sand piles are related to a special case of the *chip firing game*, a well-known model in combinatorics and game theory introduced by Björner et al. in [2].

Here the game consists of simulating the fall of sand grains organized into adjacent columns of decreasing heights. The game starts with n grains stacked in column 1 and has a simple evolution rule: if there are two adjacent columns, say i and $i+1$, with heights differing by at least 2, a grain falls down from column i to column $i+1$, leading to a new state. Thus, each state can be represented by a linear partition of n, and the aim is that of generating all the linear partitions of n which represent states of the game, that is, configurations which are reachable from the initial state, i.e. the partition (n).

More formally, we are interested in the standard (sequential) Sand Pile Model, SPM, introduced by Goles and Kiwi [6] as a restriction of the discrete dynamical model proposed in 1973 by Brylawski [3] in order to study the set of the linear partitions of an integer n. SPM has been widely studied in physics and in the theory of cellular automata to represent granular objects, and also analysed from a combinatorial point of view, see [1, 6, 7, 8].

The set of linear partitions obtained from (n) is denoted by SPM(n) and turns out to be a lattice with respect to a suitable ordering (the dominance ordering) induced by the evolution rule of the game [6]. Many combinatorial properties of SPM(n) are known. In particular, useful bounds for |SPM(n)| have been obtained in [4].

Here we study the problem of the exhaustive generation of the elements of SPM(n) by means of a CAT (Constant Amortized Time) algorithm. We show that by considering the negative lexicographic ordering on SPM(n), one can define an iterative algorithm that uses $O(n)$ space to sequentially generate all the elements of SPM(n) in $O(1)$ amortized time.

2 Preliminaries

A linear partition of n is a non-increasing sequence of positive integers $s = (s_1, \ldots, s_k)$ such that $\sum_{i=1}^{k} s_i = n$; its *height* is s_1 and its *length* is $l(s) = k$. The *height difference* of s at i is defined as $\delta_i(s) = s_i - s_{i+1}$ (assume $s_i = 0$ for $i > l(s)$).

[‖]Partially supported by Project M.I.U.R. PRIN 2007–2009: *Mathematical aspects and forthcoming applications of automata and formal languages*

Given a linear partition s and an integer i, we define an operation $\mathrm{Fall}(s,i)$ which acts on s by subtracting 1 to the i-th component while adding 1 to the $(i+1)$-th one, provided that $\delta_i(s) \geq 2$. More formally, we have:

$$\mathrm{Fall}((s_1,\ldots,s_k),i) = \begin{cases} (s_1,\ldots,s_{i-1},s_i-1,s_{i+1}+1,\ldots,s_k) & \text{if} \quad 1 \leq i \leq k, \\ & \delta_i(s) \geq 2 \\ \perp \text{ (undefined)} & \text{otherwise} \end{cases}$$

The set of *sand piles* with n grains, $\mathrm{SPM}(n)$, is defined as the closure of $\{(n)\}$ under Fall. This set contains all the linear partitions of n representing reachable states of the so-called *sand pile game*, a simple game which starts with n grains stacked in one column and evolves according to this dynamical rule: you can move a grain from column i to column $i+1$ if and only if the height difference at i is at least 2.

Goles and Kiwi [6] showed that $\mathrm{SPM}(n)$ has a unique *fixed point* (i.e. a configuration where no grain can fall) and that it turns out to be a lattice with respect to the *dominance ordering*, $\geq$: let $s=(s_1,\ldots,s_h)$ and $t=(t_1,\ldots,t_k)$ belong to $\mathrm{SPM}(n)$, then

$$s \geq t \quad \Longleftrightarrow \quad \sum_{i=1}^{j} s_i \geq \sum_{i=1}^{j} t_i, \quad j=1,\ldots,\max(h,k).$$

Moreover, we recall that a characterization of sand piles was given in [6]. In the sequel, we are interested in a particular total ordering on $\mathrm{SPM}(n)$ given by the so-called negative lexicographic or *neglex* ordering, $<_{\mathrm{neglex}}$, which is defined as follows:

$$(s_1,\ldots,s_h) <_{\mathrm{neglex}} (t_1,\ldots,t_k)$$

if and only if

$$\exists i,\ 1 \leq i \leq \min(h,k) : s_j = t_j, 1 \leq j < i, s_i > t_i.$$

It is immediate to note that if $s,t \in \mathrm{SPM}(n)$ and $s > t$ (with respect to the dominance ordering) then $s <_{\mathrm{neglex}} t$. An important property of the sand piles is stated in the following (see [6]):

Lemma 2.1. *For any $s \in SPM(n)$ we have $l(s) = O(\sqrt{n})$.*

Moreover, the following Lemma can be immediately proved.

Lemma 2.2.

1. $(a_1,\ldots,a_p,b,b,c_1,\ldots,c_q) \in SPM(n)$ *if and only if*

$$(a_1,\ldots,a_p,b+1,b-1,c_1,\ldots,c_q) \in SPM(n);$$

2. $(a_1,\ldots,a_p,a_p-2,a_p-3,\ldots,2,1) \in SPM(n)$ *if and only if*

$$(a_1,\ldots,a_p,a_p-1,a_p-3,a_p-4\ldots,3,2) \in SPM(n);$$

3. $(\ldots,c,c,c,\ldots) \notin SPM(n)$.

Given $s=(s_1,\ldots,s_k) \in \mathrm{SPM}(n)$ and two integers i,j, $1 \leq i \leq j \leq k$, we denote by $s[i,j]$ the subsequence $(s_i,s_{i+1},\ldots,s_j)$, which can be easily shown to be a sand pile in $\mathrm{SPM}(\sum_{h=i}^{j} s_h)$. We also indicate by $\cdot$ the *concatenation* product,

$$(s_1,\ldots,s_h)\cdot(t_1,\ldots,t_k) = (s_1,\ldots,s_h,t_1,\ldots,t_k).$$

Moreover, we indicate by $\mathrm{Rmost}(s)$ ($\mathrm{Lmost}(s)$) the index i of the rightmost (leftmost) column of s such that $\delta_i(s) > 1$. The *set of moves* of a sand pile s is defined as

$$\mathrm{M}(s) = \{e | 1 \leq e \leq l(s), \delta_e(s) > 1\}.$$

A sand pile s with $|\mathrm{M}(s)| > 1$ is called *branching*.

Definition 2.1. *Let $s \in SPM(n)$ and $i < Rmost(s)$. Then s is called minimal at i if and only if $i \in M(s)$ and for all $t \in SPM(n)$ with $t[1,i] = s[1,i]$ and $i \in M(t)$ we have $s <_{neglex} t$.*

By definition, minimal sand piles are branching and, if $s, t \in \mathrm{SPM}(n)$ are minimal at i and have the same prefix $s[1,i] = t[1,i]$, then $s = t$. Moreover, note that if s is minimal at i and j with $i < j$, then $j = l(s) - 1$. For example, $(6,4,2) \in \mathrm{SPM}(12)$ and $(6,4,4,2) \in \mathrm{SPM}(16)$ are minimal, the former at 1 and 2, the latter at 1 and 3. The set of minimal sand piles with n grains is $\mathrm{MSP}(n) \subset \mathrm{SPM}(n)$ and admits the partition $\mathrm{MSP}(n) = \mathrm{MSP}_1(n) \cup \mathrm{MSP}_2(n)$ where $\mathrm{MSP}_1(n) = \{s \in \mathrm{MSP}(n) \mid \exists! i \text{ s.t. } s \text{ is minimal at } i\}$ and $\mathrm{MSP}_2(n) = \{s \in \mathrm{MSP}(n) \mid \exists i, j, i \neq j, \text{ s.t. } s \text{ is minimal at } i, j\}$.

Example 1. Let us consider the sand piles in SPM(10) and list them according to the neglex ordering (we underline the columns where sand piles turn out to be minimal):

$$(10), (9,1), (\underline{8},2), (8,1,1), (\underline{7},3), (7,2,1), (\underline{6},4), (6,3,1), (6,2,2), (6,2,1,1),$$
$$(5,5), (5,4,1), (\underline{5},3,2), (5,3,1,1), (5,2,2,1), (4,\underline{4},2), (4,4,1,1), (4,3,3),$$
$$(4,3,2,1).$$

For minimal sand piles it is immediate to prove the following: Fact Let $s \in \mathrm{MSP}(n)$ be minimal at i and let $k = l(s)$. Then we have:

- $\mathrm{M}(s[i,k]) = \{i\} \cup T$ with $T \subseteq \{k-1, k\}$;
- if $i < k-1$ then $\mathrm{M}(\mathrm{Fall}(s[i,k], i)) \subseteq \{k-1, k\}$.

For any $k \geq 0$ we denote by $s^{(k)}$ the $(k+1)$-th sand pile in the ordered sequence of elements in $\mathrm{SPM}(n)$.

In the sequel, we are interested in a function that associates with a sand pile s minimal at i a suitable set of at most $l(s) - i + 1$ sand piles. More formally, we define a function $\mathrm{DASP} : \mathrm{MSP}(n) \mapsto 2^{\mathrm{SPM}(n)}$ such that $\mathrm{DASP}(s)$ denotes the set of *directly associated sand piles* of a minimal sand pile s, univocally identified by the following definition.

Definition 2.2. *Let $t \in SPM(n)$ and $s \in MSP(n)$. We say that t is directly associated with s if and only if*

- *$s^{(e)} \leq_{neglex} t <_{neglex} s^{(d)}$, where $s^{(d)} = Fall(s,i)$ and either $s^{(e)} = Fall(s, Rmost(s))$ (if $s \in MSP_1(n)$ is minimal at i) or $s^{(e)} = Fall(s,j)$ (if $s \in MSP_2(n)$ is minimal at $i, j = l(s) - 1$);*
- *either $M(t[i+1, l(t)]) = \emptyset$ or there is $l, i < l \leq l(s)$, such that*
  - *$l = Lmost(t[i+1, l(t)])$;*
  - *for all v with $s^{(e)} \leq_{neglex} v <_{neglex} s^{(d)}$ and $l = Lmost(v[i+1, l(v)])$ we have $t <_{neglex} v$.*

By definition we have $|\mathrm{DASP}(s)| \leq l(s) - i + 1$. In particular, if s is minimal at i then $\mathrm{DASP}(s)$ univocally identifies a set $T(s)$ of consecutive integers in the range $[i+1, l(s)]$ such that $|\mathrm{DASP}(s)| = |T(s)| + 1$: $k \in T(s)$ if and only if there is $s^{(f)}$ in the range $[s^{(e)}, s^{(d-1)}]$ with $k = \mathrm{Lmost}(s^{(f)}[i+1, l(s)])$.

Example 2. Figure 1 illustrates the lattice SPM(14) where minimal sand piles are circled. With respect to this picture we have

$$\begin{array}{lcl}
\mathrm{DASP}((12,2)) & = & \{(12,1,1)\}, \\
\mathrm{DASP}((5,5,3,1)) & = & \{(5,5,2,2),(5,5,2,1,1)\}, \\
\mathrm{DASP}((6,4,4)) & = & \{(6,4,3,1),(6,4,2,2),(6,3,2,2,1)\}, \\
\mathrm{DASP}((6,4,2,2)) & = & \{(6,4,2,1,1)\}, \\
\mathrm{DASP}((7,5,2)) & = & \{((7,4,3),(7,4,2,1),(7,3,2,1,1)\},
\end{array}$$

and then

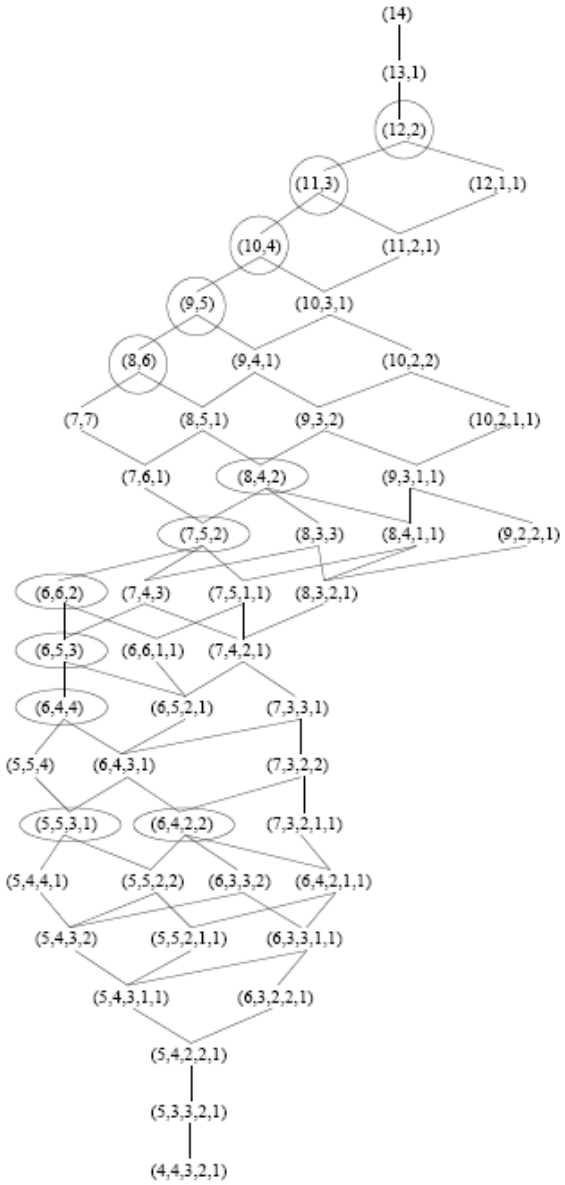

Figure 1: Minimal sand piles in SPM(14).

$$
\begin{array}{lcl}
\mathrm{T}((12,2)) & = & \{\}, \\
\mathrm{T}((5,5,3,1)) & = & \{4\}, \\
\mathrm{T}((6,4,4)) & = & \{2,3\}, \\
\mathrm{T}((6,4,2,2)) & = & \{\}, \\
\mathrm{T}(7,5,2) & = & \{2,3\}.
\end{array}
$$

The following Lemma can be easily proved.

Lemma 2.3. *Let $s, t \in MSP(n)$. Then we have $DASP(s) \cap DASP(t) = \emptyset$ and*

$$\bigcup_{s \in MSP(n)} |DASP(s)| \leq |SPM(n)|.$$

3 The Problem

Here we are interested in the following problem:

Problem: Sand Piles Generation (SPG)

Input: an integer n

Output: the sequence of all and only the sand piles in SPM(n).

Note that because of the exponential growth of |SPM(n)| (see [4, Th. 1]) a standard approach based on dynamic programming is not feasible due to the huge space requirement. Moreover, we recall that in [8] it has been shown how to construct the lattice SPM($n+1$) from SPM(n) by an algorithm which is linear (in time and in space) with respect to |SPM(n)|. This construction points out a self-similarity in each lattice SPM(n). In particular (see [8, Proposition 4]), by adding a grain to the first column of each sand pile in SPM(n) we obtain a sublattice of SPM($n+1$), denoted by SPM$(n)^{\downarrow_1}$.

Thus, our approach is that of producing the ordered sequence (with respet to $<_{\text{neglex}}$) of sand piles in SPM(n),

$$s^{(0)} = (n), s^{(2)} = (n-1,1), \ldots, s^{(f)},$$

where $s^{(f)}$ is the unique fixed point, by an iterative process that computes $s^{(k+1)}$ by moving a grain in column Rmost$(s^{(k)})$ either in $s^{(k)}$ or in a previously generated sand pile $s^{(j)}$ that is minimal at Rmost$(s^{(k)})$ and such that $s^{(j)}[1, \text{Rmost}(s^{(k)})] = s^{(k)}[1, \text{Rmost}(s^{(k)})]$.

More formally, the process exploits an array Min of suffixes of minimal sand piles and maintains it in a way that the following condition holds: if $s^{(k)}$ is the latest generated sand pile and $i = \text{Rmost}(s^{(k)})$ then, for $j \leq i$, Min$[j]$ provides the suffix $s^{(c_j)}[j+1, l(s^{(c_j)})]$ of the sand pile $s^{(c_j)}$ ($c_j < k$) that is minimal at j and shares a prefix of length j with $s^{(k)}$, i.e. $s^{(k)}[1,j] = s^{(c_j)}[1,j]$. Thus, the process iterates the following steps:

1. Let $s^{(k)} = (a_1, \ldots, a_i, \ldots, a_l)$ be the latest generated sand pile and let $i = \text{Rmost}(s^{(k)})$. If there is $j < k$ such that $s^{(j)} = (a_1, \ldots, a_i, b_{i+1}, \ldots, b_m)$ is minimal at i (with $s^{(j)}[i+1, m]$ stored in Min$[i]$)

   then use it to compute

   $$s^{(k+1)} := \text{Fall}(s^{(j)}, i) = \text{Fall}(s^{(k)}[1,i] \cdot \text{Min}[i], i)$$

   and free Min$[i]$ (i.e. set Min$[i] := ()$);

   else set $s^{(k+1)} := \text{Fall}(s^{(k)}, i)$;

2. consider M$(s^{(k+1)})$ and for $e \in \text{M}(s^{(k+1)}) \cap \{i-1, i, i+1, \text{Rmost}(s^{(k+1)}) - 1\}$ if Min$[e] = ()$ set Min$[e] := s^{(k+1)}[e+1, l(s^{(k+1)})]$.

Example 3. Starting with $s^{(0)} = (8)$ and Min $= [(), (), ()]$ we proceed to $s^{(1)} = (7,1)$ and then to the sand pile $s^{(2)} = (6,2)$ minimal at 1. Now we have M$(s^{(2)}) = \{1,2\}$ and we store $s^{(2)}[2,2]$ in Min[1], (Min $= [(2), (), ()]$). By moving a grain in the first column of $s^{(2)}$ you obtain the smallest sand pile with the first component equal to 5. Once we have computed $s^{(3)} = \text{Fall}(s^{(2)}, \text{Rmost}(s^{(2)})) = (6,1,1)$, the next sand pile would require a move in the first column. Actually, since Min$[1] \neq ()$, we proceed as follows: first, get the latest generated sand pile that is minimal at 1 by constructing $s^{(3)}[1,1] \cdot \text{Min}[1] = (6,2)$, then set Min$[1] := ()$ and compute $s^{(4)} = \text{Fall}((6,2),1) = (5,3)$. Again, we have M$(s^{(4)}) = \{1,2\}$ and we update Min, Min $= [(3), (), ()]$. The process continues by computing

$$\begin{array}{lcll}
s^{(5)} & = & \text{Fall}(s^{(4)}, \text{Rmost}(s^{(4)})) = (5,2,1), & \text{Min} = [(3), (), ()], \\
s^{(6)} & \overset{\text{Min}[1]}{=} & (4,4), & \text{Min} = [(), (), ()], \\
s^{(7)} & = & \text{Fall}(s^{(6)}, \text{Rmost}(s^{(6)})) = (4,3,1), & \text{Min} = [(), (), ()], \\
s^{(8)} & = & \text{Fall}(s^{(7)}, \text{Rmost}(s^{(7)})) = (4,2,2), & \text{Min} = [(2,2), (), ()], \\
s^{(9)} & = & \text{Fall}(s^{(8)}, \text{Rmost}(s^{(8)})) = (4,2,1,1), & \text{Min} = [(2,2), (), ()], \\
s^{(10)} & \overset{\text{Min}[1]}{=} & (3,3,2), & \text{Min} = [(), (), ()], \\
s^{(11)} & = & \text{Fall}(s^{(10)}, \text{Rmost}(s^{(10)})) = (3,3,1,1), & \text{Min} = [(), (), ()], \\
s^{(12)} & = & \text{Fall}(s^{(11)}, \text{Rmost}(s^{(11)})) = (3,2,2,1), & \text{Min} = [(), (), ()].
\end{array}$$

4 The Algorithm

The idea presented in the previous section leads immediately to Algorithm 3 shown below. At each iteration we have a sand pile (`partition`) and the associated set of moves, represented by an increasing sequence of indices (the content of `St`, a stack with the top indicating the rightmost column where a move can occur). Moreover, a variable `latestmove` provides the index of the column where a move occurred in the previous generated sand pile, while an array `Min` is used to store suffixes of minimal sand piles.

Given an integer n, we start by setting `partition` to (n), the first sand pile in SPM(n), and by pushing 1 onto `St` (lines 2-3). Then, as long as the stack is not empty (i.e. at least one move is possible), the index `i` of the righmost column where a move can occur (in the latest generated sand pile, i.e. the current value of `partition`) is popped from `St`. In order to generate the next sand pile by a move in position `i`, we first check whether a sand pile minimal at `i` (with prefix equal to `partition`$[1,i]$) is available (line 6): in this case RESTORE returns such a sand pile (line 7) and the move will act on it. Now, FALL moves the grain so that `partition` gets a new value (the next sand pile, line 9). Then, we update the set of moves. In line 10 MOVES returns the ordered list of new indices of columns where a move is possible: note that each entry belongs to $\{i-1, i, i+1, r-1, r\}$ where r is the length of the sand pile. If the new generated sand pile is minimal, then suitable suffixes are saved in `Min` (STORE), while the associated positions are pushed onto the stack in order (lines 11–16). Finally the index denoting the rightmost move is pushed onto the stack (line 17). Observe that the algorithm computes a spanning tree associated with the lattice SPM(n).

Algorithm 3: Generation of SPM(n).

```
1:  PROCEDURE SANDPILEGENERATION(n)
2:  partition ← (n); Min ← [(), ..., ()];
3:  PUSH(St,1); latestmove ← 0;
4:  while ISNOTEMPTY(St) do
5:    i ← POP(St);
6:    if Min[i] ≠ () then
7:      partition ← RESTORE(partition,Min,i,latestmove);
8:    end if
9:    FALL(partition,i);
10:   (l_1, ..., l_p) ← MOVES(partition,i);
11:   for i = 1 to p − 1 do
12:     if Min[l_i] = () then
13:       STORE(partition,Min,l_i,latestmove);
14:       PUSH(St,l_i);
15:     end if
16:   end for
17:   PUSH(St,l_p); latestmove ← i;
18: end while
```

Example 4. Figure 2 illustrates the lattice SPM(14) and the associated spanning tree (thick edges) which shows how the algorithm works. Every sand pile is directly computed from its parent. The tree is ternary and nodes with at least 2 children correspond to minimal sand piles which have a suffix saved in `Min`. More precisely, unary nodes are not minimal, binary nodes are in $\mathrm{MSP}_1(n)$, while ternary nodes are in $\mathrm{MSP}_2(n)$. The sequence of sand piles produced by the algorithm can be seen as a variant of the depth-first traversal.

The correctness of the algorithm is proved in the following theorem.

Theorem 4.1. SANDPILEGENERATION*(n) generates all and only the sand piles in SPM(n).*

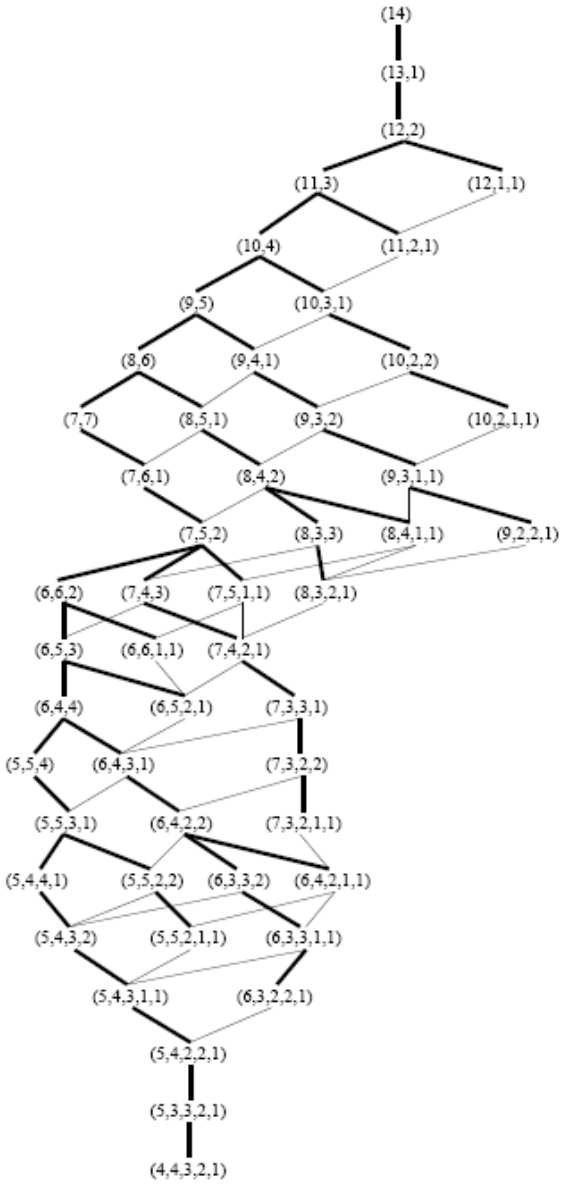

Figure 2: The spanning tree of SPM(14).

Proof: We prove that:

1. each generated partition is a sand pile;
2. the sequence of generated sand piles is strictly increasing (w.r.t. $<_{\text{neglex}}$);
3. every sand pile of SPM(n) appears in the sequence.

(1) An easy induction on k shows that $s^{(k)}$ is a sand pile. The basis is $s^{(0)} = (n)$. By induction, $s^{(k-1)} = (a_1, \ldots, a_i, \ldots, a_r)$ is a sand pile, with $i = \text{Rmost}(s^{(k-1)})$. Then, the next generated partition is a sand pile since the algorithm computes either $s^{(k)} = \text{Fall}(s^{(k-1)}, i)$ or $s^{(k)} = \text{Fall}(s^{(e)}, i)$ with $e < k-1$.
(2) If $s^{(k-1)} = (a_1, \ldots, a_i, \ldots, a_r)$ and $i = \text{Rmost}(s^{(k-1)})$, then we have $s^{(k)} = (a_1, \ldots, a_{i-1}, a_i - 1, \ldots)$ and so $s^{(k-1)} <_{\text{neglex}} s^{(k)}$.

(3) We argue it by contradiction. Let $t = (t_1, \ldots, t_q)$ be the smallest missing sand pile in the (ordered) sequence. This means that there are two sand piles

$$s^{(k-1)} = (a_1, \ldots, a_i \ldots, a_r), \quad s^{(k)} = (a_1, \ldots, , a_{i-1}, a_i - 1, b_1, \ldots, b_p)$$

(with $i = \text{Rmost}(s^{(k-1)})$) such that

$$s^{(k-1)} <_{\text{neglex}} t <_{\text{neglex}} s^{(k)}$$

and thus $t = (a_1, \ldots, a_{i-1}, t_i, \ldots, t_q)$. Now, let $d = \sum_{k=i+1}^{r} a_k$ and distinguish two cases. If $t_i = a_i$, then we have

$$(a_{i+1}, \ldots, a_r) <_{\text{neglex}} (t_{i+1}, \ldots, t_q)$$

which is obviously false since $(a_{i+1}, \ldots, a_r), (t_{i+1}, \ldots, t_q) \in \text{SPM}(d)$ and the unique fixed point is $(a_{i+1}, \ldots, a_r)$ (note that $\text{M}((a_{i+1}, \ldots, a_r)) = \emptyset$ since $i = \text{Rmost}(s^{(k-1)})$).

Thus, we have $t_i = a_i - 1$. Suppose first that $s^{(k)} = \text{Fall}(s^{(k-1)}, i)$ (i.e. there is not a sand pile minimal at i with prefix equal to $s^{(k-1)}[1, i]$). If $i = r - 1$ then $a_r = 1$. As a consequence we have

$$(a_i, 1) <_{\text{neglex}} (t_i, \ldots, t_q) <_{\text{neglex}} (a_i - 1, 2)$$

which is a contradiction since $t_i = a_i - 1$ implies $t_{i+1} = 2$ and $t = s^{(k)}$. So, $i < r - 1$ and $\delta_i(s^{(k-1)}) = 2$ (if $\delta_i(s^{(k-1)}) > 2$ a sand pile minimal at i with prefix $s^{(k-1)}[1, i]$ would exist). Now, observe that we necessarily have

$$s^{(k-1)} = (a_1, \ldots, a_i, a_i - 2, a_i - 2, a_i - 3, a_i - 4, \ldots, 2, 1)$$

otherwise, by Lemma 2.2, we could find a sand pile minimal at i with prefix $s^{(k-1)}[1, i]$. So, we obtain

$$t <_{\text{neglex}} s^{(k)} = (a_1, \ldots, a_i - 1, a_i - 1, a_i - 2, a_i - 3, a_i - 4, \ldots, 2, 1)$$

that is a contradiction since it implies that a sand pile of type

$$(a_1, \ldots, a_{i-1}, \ldots, c, c, c, \ldots)$$

would belong to $\text{SPM}(n)$.

Therefore, we have $s^{(k)} = \text{Fall}(s^{(e)}, i)$ for a previous $(e < k - 1)$ sand pile

$$s^{(e)} = (a_1, \ldots, a_{i-1}, a_i, a_i - 2, b_2 \ldots, b_p)$$

minimal at i. So, we have

$$(a_i \ldots, a_r) <_{\text{neglex}} (a_i - 1, t_{i+1}, \ldots, t_q) <_{\text{neglex}} (a_i - 1, a_i - 1, b_2 \ldots, b_p),$$

that is,

$$t = (a_1, \ldots, a_{i-1}, a_i - 1, a_i - 1, t_{i+2}, \ldots, t_q).$$

In this case, from $t <_{\text{neglex}} s^{(k)}$ we obtain that $s^{(e)}$ is not minimal at i. In fact, by Lemma 2.2, we have $\hat{t} = (a_1, \ldots, a_{i-1}, a_i, a_i - 2, t_{i+2}, \ldots, t_q) \in \text{SPM}(n)$ and $\hat{t} <_{\text{neglex}} s^{(e)}$.

J

5 Complexity analysis

With respect to the complexity, observe that if STORE had running time $O(1)$ then SANDPILEGENERATION would be a CAT algorithm (RESTORE can be easily implemented so that it requires $O(1)$ list operations). Unfortunately, the lengths of suffixes in `Min` are not $O(1)$. Nevertheless, an amortized analysis shows that the cost of all the calls to STORE is $O(|\text{SPM}(n)|)$.

In order to do that, we recall that $\text{DASP}(s)$ associates with a sand pile s minimal at i a set of at most $l(s) - i + 1$ sand piles in the subtree rooted at s. Thus, we get the result by proving that each minimal sand pile s can be processed in time $\Theta(|\text{DASP}(s)|)$.

The next Lemma illustrates an important relation between RESTORE and STORE: informally, every call to STORE that deals with a suffix longer than 2 is preceded in the same iteration by a call to RESTORE.

Lemma 5.1. *Let $s^{(k)}$ be the sand pile minimal at j, with $l(s^{(k)}) - j > 2$, generated at the k-th iteration of* SandPileGeneration *and let i be the integer popped from* `St`*. Then,*

- *at the beginning of the iteration we have* `Min`$[i] \neq ()$*, that is, a sand pile $s^{(e)} = s^{(k-1)}[1,i] \cdot$* `Min`$[i]$ *minimal at i is restored, $e < k$;*
- *either $j = i-1$ or $j = i+1$.*

Proof: First, note that a move has a local effect, that is, if $i \in \mathrm{M}(s)$ then $\mathrm{M}(\mathrm{Fall}(s,i)) \subseteq \mathrm{M}(s) \cup \{i-1, i+1\}$. Moreover, recall that when a sand pile t minimal at j is generated we have $j < \mathrm{Rmost}(t)$, $|\mathrm{M}(t[j, l(t)])| > 1$ and the suffix $t[j+1, l(t)]$ is saved in `Min`$[j]$.

Thus, let $s = s^{(k-1)}$, $t = s^{(k)}$ and $i = \mathrm{Rmost}(s)$. Suppose that at the beginning of the k-th iteration we have `Min`$[i] = ()$, that is, $t = \mathrm{Fall}(s,i)$, and distinguish two cases. If $l(s) \leq i+1$ it is immediate to see that t can not be minimal at j with $l(t) - j > 2$. So, let $l(s) > i+1$. Suppose first that $j = i$ ($j < i+1$ since $i = \mathrm{Rmost}(s)$). This implies that $\delta_i(s) \geq 4$ and $\mathrm{Rmost}(t) = i+1$ since $\delta_{i+1}(t) = 2$. Then, by applying Lemma 2.2 to t it is easy to find a sand pile $v \in \mathrm{SPM}(n)$ such that $\delta_i(v) = 2$, $v[1,i] = t[1,i]$ and $v <_{\mathrm{neglex}} t$. This means that t is not minimal at i (contradiction).

Thus, at the beginning of the k-th iteration we have `Min`$[i] \neq ()$ and then $t = \mathrm{Fall}(s[1,i] \cdot$ `Min`$[i], i)$ and $l(t) = i + r$, where $r = l($`Min`$[i])$. By recalling Fact 2, we easily get $j \in \{i-1, i+1\}$. This means that either $t[i, i+r]$ (of length $r+1$) is saved in `Min`$[i-1]$ or $t[i+2, i+r]$ (of length $r-1$) is saved in `Min`$[i+1]$.

$\mathcal{J}$

An immediate consequence of Lemma 5.1 is:

Corollary 5.1. *Let d, e be two integers, $d < e$, such that*

- *$s^{(d)} = (a_1, \ldots, a_i, a_{i+1}, \ldots, a_{i+r})$ is minimal at i, with $r \geq 3$ and $r - |DASP(s^{(d)})| \geq 3$;*
- *$s^{(e)} = Fall(s^{(d)}, i)$ is minimal at $i-1$.*

Then, $|DASP(s^{(d)})| = |DASP(s^{(e)})|$.

Proof: Observe that $s^{(d)} \in \mathrm{MSP}_j(n)$ if and only if $s^{(e)} \in \mathrm{MSP}_j(n)$, $j \in \{1,2\}$. So, we first consider the case $s^{(d)} \in \mathrm{MSP}_1(n)$.

Since $|\mathrm{DASP}(s^{(d)})| \leq r-3$ we have $\min(T(s^{(d)})) > i+1$. So, it is immediate to show that there are $|\mathrm{DASP}(s^{(d)})|$ sand piles $\alpha_1, \ldots, \alpha_{|\mathrm{DASP}(s^{(d)})|}$ (depending only on $s^{(d)}[i+2, i+r]$) such that

$$\mathrm{DASP}(s^{(d)}) = \{\alpha \cdot \alpha_1, \ldots, \alpha \cdot \alpha_{|\mathrm{DASP}(s^{(d)})|}\}$$

where $\alpha = s^{(d)}[1, i+1]$.

Then, by noting that $s^{(e)} = (a_1, \ldots, a_i - 1, a_{i+1} + 1, a_{i+2}, \ldots, a_{i+r})$, that is, $s^{(d)}[i+2, i+r] = s^{(e)}[i+2, i+r]$, we get

$$\mathrm{DASP}(s^{(e)}) = \{\beta \cdot \alpha_1, \ldots, \beta \cdot \alpha_{|\mathrm{DASP}(s^{(d)})|}\}$$

with $\beta = s^{(e)}[1, i+1]$. A similar reasoning holds if $s^{(d)} \in \mathrm{MSP}_2(n)$.

$\mathcal{J}$

We can now prove that the cost of all the calls to Store during the execution of SandPileGeneration is bounded by the number of sand piles. More precisely, we have:

Lemma 5.2. *Let $M(n)$ be the sum of the lengths of all the suffixes that are saved in* `Min` *during the execution of* SandPileGeneration*(n). Then,*

$$M(n) = O(|SPM(n)|).$$

Proof: For each sand pile s minimal at i let $l'(s) = l(s) - i$ and define $L_{\geq 3}(n) = \{s \in \mathrm{MSP}(n) \mid l'(s) \geq 3\}$ and $S(n) = \sum_{s \in L_{\geq 3}} |\mathrm{DASP}(s)|$. By Lemma 2.3, we have $S(n) \leq |\mathrm{SPM}(n)|$ and then it is sufficient to prove that $M(n) = \sum_{s \in \mathrm{MSP}(n)} l'(s) = O(S(n))$. So, consider the partition $L_{\geq 3}(n) = \hat{S}_n \cup \tilde{S}_n$ given by

$$\hat{S}_n = \{s \in L_{\geq 3}(n) \mid l'(s) - |\mathrm{DASP}(s)| < 3\},$$

$$\tilde{S}_n = \{s \in L_{\geq 3}(n) \mid l'(s) - |\mathrm{DASP}(s)| \geq 3\}.$$

Thus, by setting $G_n = \mathrm{MSP}(n) \setminus L_{\geq 3}(n)$, we have

$$\begin{aligned} M(n) &\leq 2|G_n| + \sum_{s \in \hat{S}_n} l'(s) + \sum_{s \in \tilde{S}_n} l'(s) \\ &\leq 2|G_n| + \sum_{s \in \hat{S}_n} |\mathrm{DASP}(s)| + 3|\hat{S}_n| + \sum_{s \in \tilde{S}_n} l(s) \\ &\leq 6|\mathrm{SPM}(n)| + \sum_{s \in \tilde{S}_n} l(s). \end{aligned}$$

Hence, it is sufficient to prove that $\sum_{s \in \tilde{S}_n} l(s) = O(|\mathrm{SPM}(n)|)$. This is done by showing that each $s \in \tilde{S}_n$ can be processed in time $\Theta(|\mathrm{DASP}(s)|)$ (instead of $\Theta(l(s))$).

In order to do that, STORE is implemented in a way that the amount of information that is saved for $s \in \tilde{S}_n$ is $\Theta(|\mathrm{DASP}(s)|)$. The idea is based on Corollary 5.1 and we give here an outline.

Let us consider the d-th iteration of SANDPILEGENERATION(n) when a sand pile $s^{(d)} \in \tilde{S}_n$ minimal at i is generated. Observe that $h_d = \min(T(s^{(d)}))$ is known at the e-th iteration, $e > d$, when RESTORE is called and $s^{(e)} = \mathrm{Fall}(s^{(d)}, i)$ is produced: actually, this corresponds to the value of the variable `lastmove` at the beginning of the e-th iteration. Thus, if $s^{(e)}$ turns out to be minimal at $i-1$ then, by Corollary 5.1, we know in advance that $h_e = \min(T(s^{(e)})) = h_d$ and we can save space (and time) by copying in `Min`$[i-1]$ only the last $l(s^{(e)}) - h_e \leq |\mathrm{DASP}(s^{(e)})|$ columns of $s^{(e)}$ (those that are sufficient to restore $s^{(e)}$ in a next iteration).

$\mathcal{J}$

We can now state the following:

Theorem 5.1. *Procedure* SANDPILEGENERATION*(n) runs in $O(1)$ amortized time using $O(n)$ space.*

Proof: Procedure SANDPILEGENERATION can be implemented by representing sand piles as double linked lists (each node is a column), that is, by letting `partition` be a double linked list. Then, the space complexity is easily determined by recalling Lemma 2.1. In fact, `Min` requires $O(n)$ space since it can be implemented as an array of length $O(\sqrt{n})$ containing links to sand piles of length $O(\sqrt{n})$. Moreover, `St` is a stack of height $O(\sqrt{n})$ since its content is an increasing sequence of indices of columns (the set of moves of the current sand pile).

With respect to the time complexity, let $T_{\mathrm{SPG}}(n)$ be the running time of SANDPILEGENERATION(n) and observe that:

1. the number of iterations is $|\mathrm{SPM}(n)| - 1$;
2. the stack operations (POP, PUSH, ISNOTEMPTY) have cost $O(1)$;
3. FALL and MOVES (by Fact 2) runs in time $O(1)$;
4. RESTORE can be easily implemented so that it requires $O(1)$ operations on linked lists;
5. the cost of STORE is $\Theta(l)$ where l is the length of the suffix of the sand pile that is saved.

So, it is immediate to see that $T_{\mathrm{SPG}}(n) = O(M(n))$, where $M(n)$ is the sum of the lengths of the sand piles that are saved in `Min` during the execution of SANDPILEGENERATION(n). Hence, by Lemma 5.2, it follows that $T_{\mathrm{SPG}}(n) = O(|\mathrm{SPM}(n)|)$, that is, SANDPILEGENERATION runs in amortized time $O(1)$.

$\mathcal{J}$

6 Conclusions

A prototype of the algorithm has been developed in C. The following table shows the values of $M(n)$ and $|\mathrm{SPM}(n)|$ for small n. An optimized version of the algorithm will be developed in the next future.

n	10	20	30	40	50	100	130
$M(n)$	16	147	901	4227	16641	4058831	58992577
$\lvert\mathrm{SPM}(n)\rvert$	19	175	1016	4672	18172	4295757	61829854

To conclude, let us consider the exhaustive generation problem for different discrete dynamical models that are generalizations of SPM. First, in [7] the Ice Pile Model IPM(k) was introduced for any fixed integer k as a natural extension of SPM, obtained by replacing operation Fall with Slide_k, defined as

$$\mathrm{Slide}_k(s,i) = (s_1, \ldots, s_{i-1}, \underbrace{p, p, \ldots, p}_{k'+2}, s_{i+k'+1}, \ldots, s_l)$$

if there is $k' < k$ such that

$$s = (s_1, \ldots, s_{i-1}, p+1, \underbrace{p, p, \ldots, p}_{k'}, p-1, s_{i+k'+1}, \ldots, s_l)$$

otherwise $\mathrm{Slide}_k(s,i) = \perp$. Goles, Morvan and Phan also gave a characterization of partitions reachable in IPM(k), while showing a formula for the (unique) fixed point. By a first investigation, it seems that the neglex ordering can be successfully applied also to produce in amortized time $O(1)$ the sequence of elements in IPM(k).

With respect to other generalizations, the discrete models BSPM (Bidimensional Sand Pile Model) and BIPM (Bidimensional Ice Pile Model) have been introduced in [5] by adding a further dimension to SPM and IPM(k), respectively. These bidimensional models exhibit some differences with respect to the unidimensional case. In particular, no characterization is known for reachable states and several fixed points may exist. Nevertheless, it would be interesting to investigate whether CAT algorithms for both cases can be obtained by considering a similar approach.

Acknowledgements. The author would like to thank Roberto Mantaci of LIAFA for stimulating discussions.

References

[1] P. Bak, C. Tang, K. Wiesenfeld, Self-organized criticality, *P*hys. Rev. A 38 (1988), 364–374.

[2] A. Björner, L. Lovsz, P.W. Shor, Chip-firing games on graphs, *E*uropean J. Combin. 12 (1991), 283–291.

[3] T. Brylawski, The lattice of integer partitions, *D*iscrete Math. 6 (1973), 201–219.

[4] S.Corteel, D. Gouyou-Beauchamps, Enumeration of sand piles, *D*iscrete Math. 256 n.3 (2002), 625–643.

[5] E. Duchi, R. Mantaci, H.D. Phan, D. Rossin Bidimensional sand pile and ice pile models, *P*U.M.A. vol. 17 (2007) n.1-2, 71–96.

[6] E. Goles, M.A. Kiwi, Games on line graphs and sand piles, *T*heoret. Comput. Sci. 115 (1993), 321–349.

[7] E. Goles, M. Morvan, H.D. Phan, Sandpiles and order structure of integer partitions, *D*iscrete Appl. Math. 117 (2002), 51–64.

[8] M. Latapy, R. Mantaci, M. Morvan, H.D. Phan Structure of same sand piles model, *Theoret. Comput. Sci.* 262 (2001), 525–556.