Chip-firing game

(a.k.a. Abelian sandpile model)

Abelian sandpile model is a simple mathematical model for complicated natural processes such as avalanches.

It was introduced by physicists:

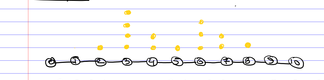
[Per Bak, Chao Tang, Kurt Wiesenfeld, 1987]

Mathematicians studied a closely related model, called the chip-firing game in

[Anders Björner, Laszlo Lovász Peter Shor, 1991]

- A pile of sand (or snow) can be modeled by a configuration (c_i) of non-negative integers c_i assigned to vertices i of some graph G (e.g. G = a large grid). Here c_i is the number of grains of sand (or chips) at vertex i.

Example



configuration $= (0,1,4,2,1,3,2,1,0,0)$

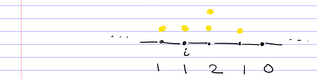
- For each vertex i, we fix some critical value d_i. (Typically, d_i is the degree of i. But it does not really matter.)

- A configuration (c_i) is stable if $c_i < d_i \quad \forall i$

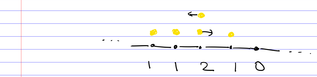
- If configuration is unstable and there exists a vertex i s.t. $c_i \geq d_i$, then vertex i can topple (or fire) by sending one chip to each neighbor of i.

Example

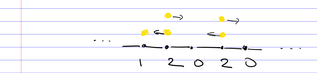
vertex i topples



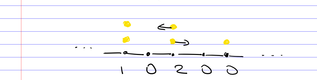
Then another vertex can topple



Then two more vertices can topple (in any order)



Another toppling



etc.

One unstable vertex can generate an "avalanche" of topplings.

This continues until we obtain a stable configuration.

This is called a stabilization.

Let's describe this in more details...

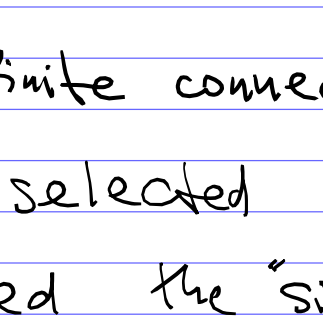
$G = (V, E)$ a finite connected graph with one selected vertex $q \in V$, called the "sink".

Remark. We'll assume that all chips that go into the "sink" disappear in order to make sure that any configuration stabilizes.

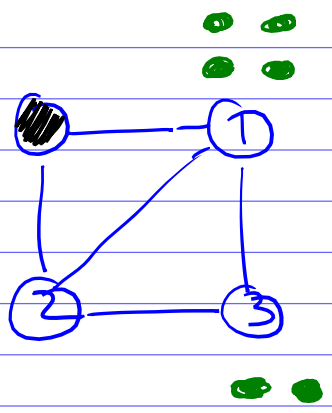
We'll assume $V = \{0, 1, \ldots, n\}$ and $q = 0$.

A configuration $C = (c_1, \ldots, c_n)$ is any non-negative integer vector. We think of c_i as the number of chips at vertex i.

Example $G =$



A configuration $C = (4, 0, 2)$



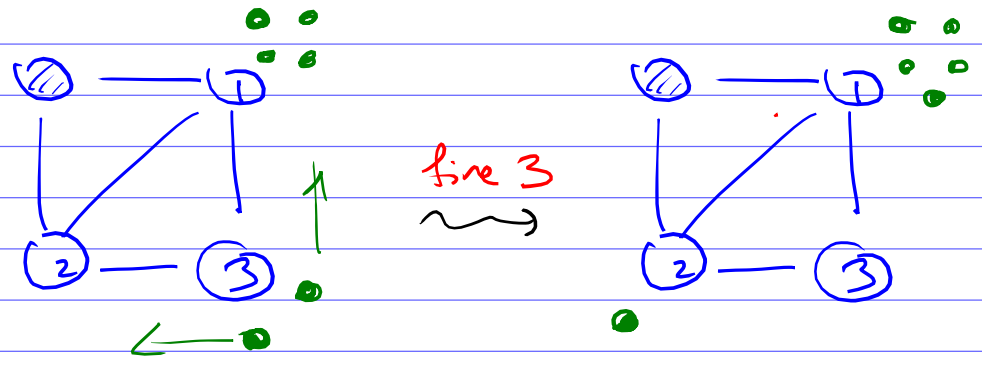
The are no chips at the sink. q.

The sink is a "black hole": any chip that goes into it "vanishes"

Let $d_i = \deg_G(i)$ degree of vertex i.

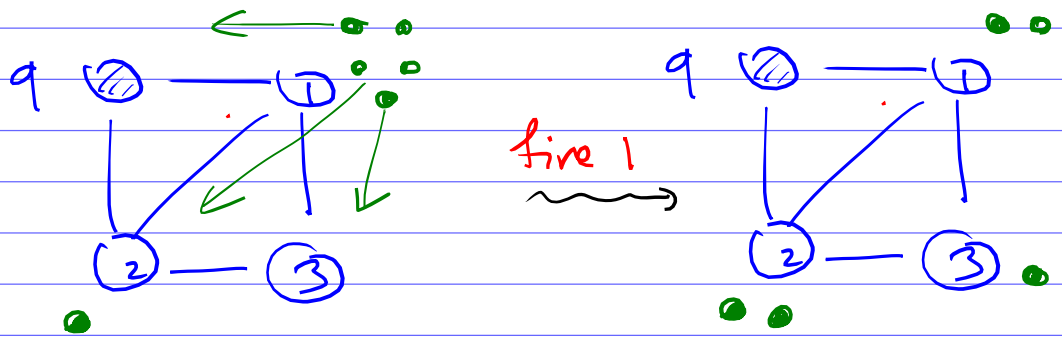
If $c_i \geq d_i$ then i is called a critical site. We can fire (or topple) such vertex i by sending one chip to each neighbor of i.

Example. We can fire vertex 3.



$$C = (4, 0, 2) \rightsquigarrow (5, 1, 0)$$

Then we can fire vertex 1



$$(5, 1, 0) \rightsquigarrow (2, 2, 1)$$

Notice that one chip goes into the sink and disappears. So the total number of chips decreases.

Now we cannot fire any vertex, because # chips at any vertex $<$ the degree of the vertex.

Def. A chip configuration $(c_1, \ldots, c_n)$ is called stable if $c_i < d_i \quad \forall i$.

(i.e. we cannot fire any vertex)

Lemma. For any initial chip configuration $C_{init} = (c_1, \ldots, c_n)$,

- we will always obtain a stable chip configuration

$$C_{stab} = (c_1', \ldots, c_n')$$

after a finite number of firings.

- The resulting stable configuration C_{stab} is unique, i.e., it depends only on C_{init} but does not depend on a choice of firings.

C_{stab} is called the stabilization of the initial conf. C_{init}.

Example (the previous example)

For $C_{init.} = (4, 0, 2)$ the stabilization is

$$C_{stab} = (2, 2, 1).$$

We will arrive to the same stable configuration if we first fire vertex 1 and then fire vertex 3.

Proof of Lemma

Finiteness: Let's assume (just for this proof) that the chips that go into the sink $q=0$ don't disappear, but stay there. So we also have the number c_0 of chips at the sink.

Let $dist(i,j)$ be the distance between vertices i & j in G, e.g. the length of a shortest path between i & j.

Define the "value" of a configuration c as

$$val(c) := \sum_{i=0}^{n} c_i \cdot N^{D-dist(0,i)}$$

where $D := \max_{i,j} dist(i,j)$ is the diameter of G and N is a VERY LARGE number.

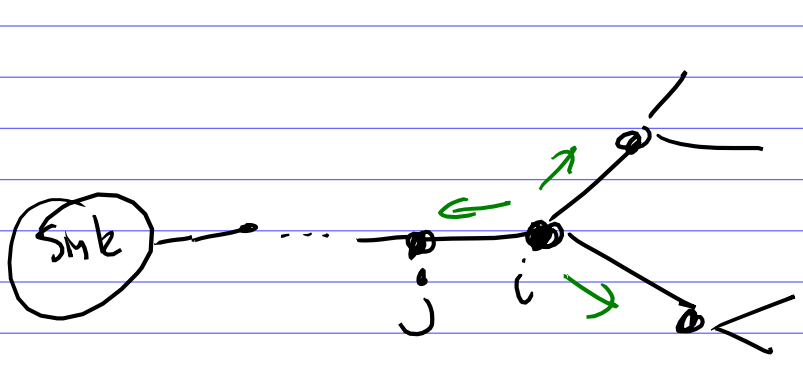
In other words, the c_0 chips at the sink 0 are the most expensive. They cost much more than any other chips. Then the chips at immediate neighbors of the sink are the next most expensive chips, etc.

We claim that, if we fire (topple) a vertex i

$$c \rightsquigarrow c',$$

then $val(c') \geq val(c)$.

Indeed, at least one neighbor j of the vertex i is closer to the sink than i,



So the chip that goes to j becomes much more expensive.

If we fix the total number of chips, then there will be a finite number of possible configurations c.

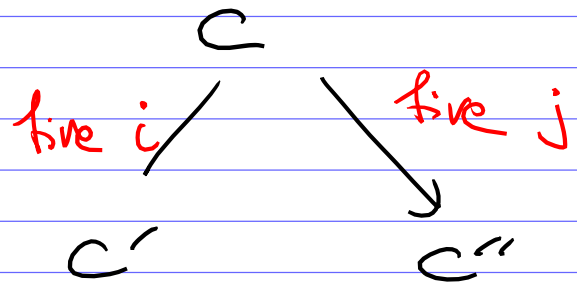
Since for each firing $val(c)$ strictly increases (by at least 1), we cannot keep firing vertices forever.

This means that for any initial configuration c_{init}, after a finite number of firings we get a stable configuration.

Uniqueness. (All stabilizations of c_{init} are the same.)

"Diamond Lemma" argument.

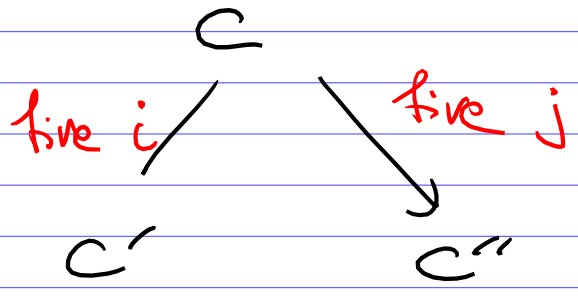
Suppose that for a configuration c we can fire two vertices i & j



Then for configuration c' we can still fire vertex j (because # chips at j can only increase, so it remains unstable).

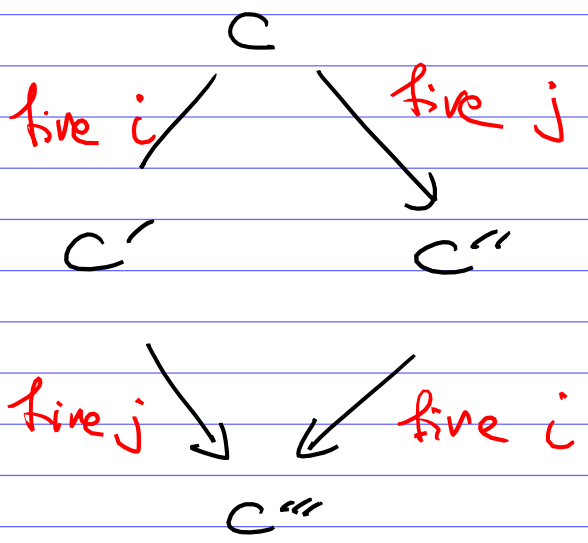
Similarly, for c'' we can still fire vertex i.

So whenever we have



we should have

this is a "diamond" →

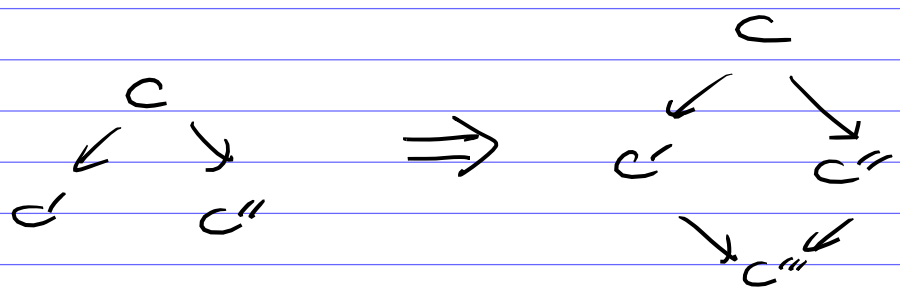


we get the same configuration c''' if we first fire i and then j or the other way around

Remark:

Firing at i commutes with firing at j. This explains the word "Abelian" in "Abelian sandpile model".

This fact

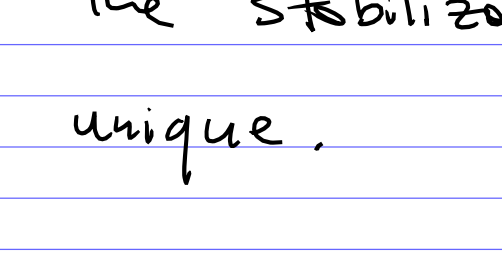


implies the uniqueness of the stabilization.

Induction on val(C)

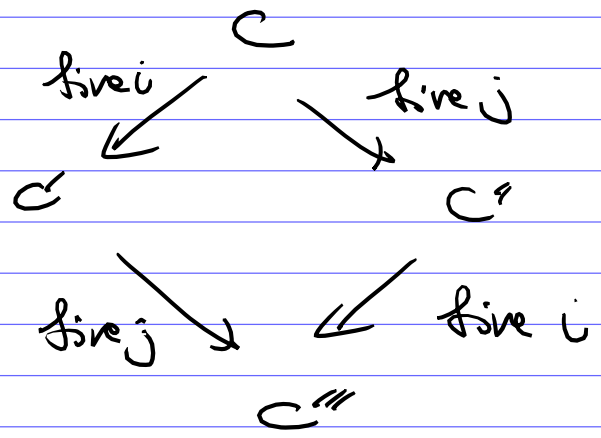
Assume that, for any configuration $\tilde{C}$ with $val(\tilde{C}) > val(C)$, any choice of firings leads to the same stable config.

Now, if there are 2 different ways to start firing for configuration C



Since $val(C') > val(C)$ and $val(C'') > val(C)$, the stabilizations of C' & C'' are unique

But since we have



The stabilization of C' should be the same as the stabilization of C'' and the same as the stabilization of C''.

$\Rightarrow$ The stabilization of C is unique.

□

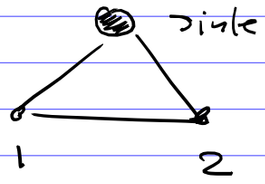
Clearly G has $d_1 \cdot d_2 \cdot \ldots \cdot d_n$ stable configurations $(c_1, \ldots, c_n)$ $\quad 0 \le c_i < d_i \ \forall i$

Consider the following random model (Markov chain) on stable configurations C.

(1) Randomly select a vertex $i \in \{1, \ldots, n\}$ (with uniform distribution), and add 1 chip to vertex i, i.e. increase c_i by 1.

(2) Stabilize the configuration.

Basically, we want to keep repeating steps (1) & (2) (randomly drop a chip, stabilize, drop a chip, stabilize, etc.)

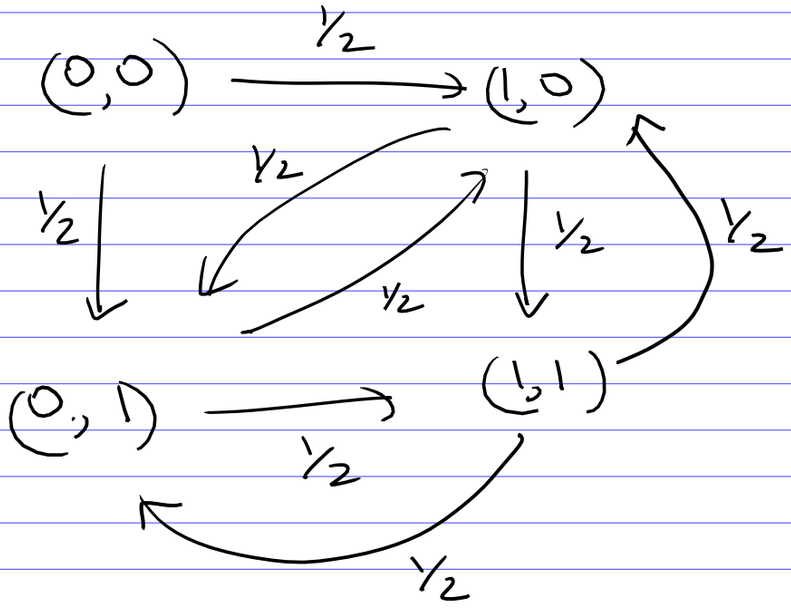
Example $G = K_3 =$ 

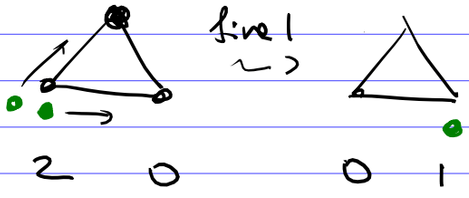
$d_1 = d_2 = 2$

4 stable configurations:

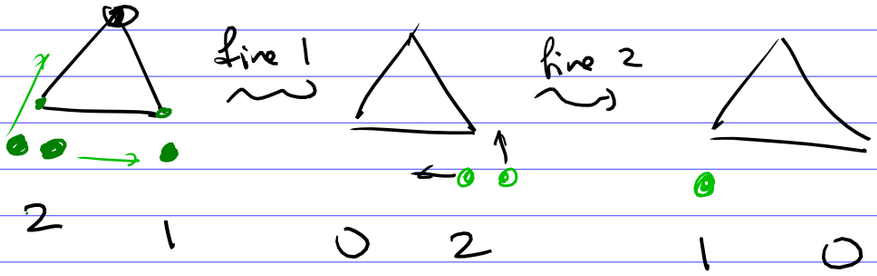
$(0,0)$ $(0,1)$, $(1,0)$, $(1,1)$

Markov chain:





the stabilization of $(2,0)$



the stabilization of $(2,1)$

Observations:

- the configuration (0,0) can only appear in the beginning of this random process. But it can never happen again.
- 3 other configurations (1,0), (0,1) (1,1) can "recur". After we run this random process for a while. All 3 of these configurations can happen with the same probability.

So the steady state distribution of this Markov chain is

$$\text{Prob}(00) = 0$$

$$\text{Prob}(01) = \text{Prob}(10) = \text{Prob}(11) = \frac{1}{3}.$$

For any graph G, we have this random process (Markov chain) on $d_1 \cdot d_2 \cdot \cdots \cdot d_n$ stable configurations.

Definition.

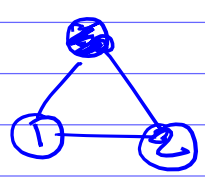
A stable configuration is called recurrent if it keeps occuring in this random process.

Equiv., C is recurrent if there is a non-zero probability to go

$C \to \cdots \to C$ in this Markov chain.

Theorem The number of recurrent configurations equals the number of spanning trees of G.

Moreover, the steady state distribution of this random process is the uniform distribution on the recurrent configurations.

Example. $G = K_3 =$ 

The recurrent configurations are $(1,0)$ $(0,1)$ $(1,1)$.

(The configuration $(0,0)$ is stable but not recurrent.)

K_3 has 3 spanning trees.

Q: How to describe recurrent configurations?

Parking functions again

Theorem For $G = K_{n+1}$, a configuration $(c_1, \ldots c_n)$ is recurrent iff $(n - c_1, \ldots, n - c_n)$ is a parking function.

Example $G = K_3$

recurrent confs. $(1\,0)$ $(0\,1)$ $(1\,1)$

parking functions: $(1\,2)$ $(2\,1)$ $(1\,1)$

In general, for any G, recurrent configurations correspond to certain G-parking functions.