# Cantor Representation

(See <https://en.wikipedia.org/wiki/Factorial_number_system>)

The normal way of representing numbers is as a sum of multiples of powers of the base:

For instance the number 3147 in base 10 is just 3.103+1.102+4.101+7.100

The Cantor (or *factoradic*) representation of a natural number is one in which the number is represented as a sum of multiples of factorials:

The terms cs can be calculated as

Or, as an Excel formula, INT(MOD(N,FACT(s+1))/FACT(s)).

So the number 3147 in Cantor notation is [4 2 1 0 1 1 0], which is 4.6!+2.5!+1.4!+0.3!+1.2!+1.1!+0.0!

The trailing 0 (which is always 0) may seem surplus to requirements, but it’s convenient to have it there when we’re using the Cantor notation to represent trees or permutations. Likewise there will be value in including leading 0s, dependent on what size the tree being represented is. In either of the two ways of turning a number ionto a tree, [4 2 1 0 1 1 0] has seven edges while [0 4 2 1 0 1 1 0] (which has the same decimal value) has eight.

Notes

* s is numbered from right to left, starting at 0, just like the powers of ten in an ordinary base-ten representation. Later I’ll have reason to refer to the ‘place’ of a term, by which I mean its position counting from left to right and beginning at 1. Place = s+1-m
* cs (the coefficient of s!) can never exceed s, in order to preserve the uniqueness of the Cantor representation

# Permutations

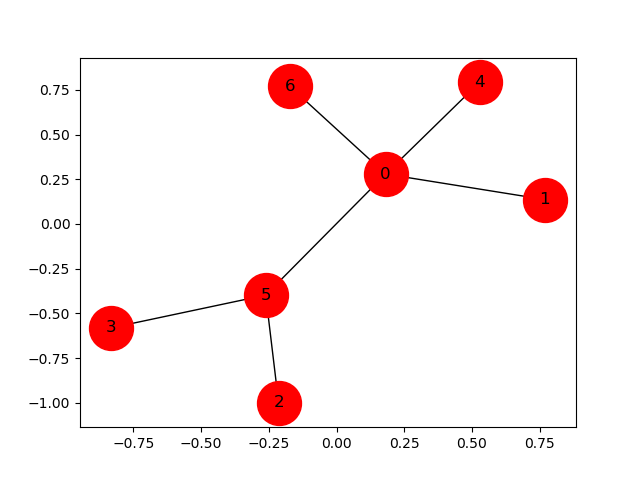
(Note: I haven’t used this idea yet in the Tree problem. It’s how I first arrived at the idea of Cantor representation, back in 1988, so I just want to include it for the sake of completeness).

A permutation of a set of objects is a bijection from the set to itself. If we take a list of objects [a,b,c,d,e,f,g] and a Cantor form [4 2 1 0 1 1 0], then starting with an empty list [] we move along the Cantor form from left to right, popping the (zero-indexed) cth element from the first list to the end of the second:

|  |  |  |
| --- | --- | --- |
| [**4**, 2, 1, 0, 1, 1, 0] | [a, b, c, d, **e**, f, g] | [e] |
| [**2**, 1, 0, 1, 1, 0] | [a, b, **c**, d, f, g] | [e, c] |
| [**1**, 0, 1, 1, 0] | [a, **b**, d, f, g] | [e, c, b] |
| [**0**, 1, 1, 0] | [**a**, d, f, g] | [e, c, b, a] |
| [**1**, 1, 0] | [d, **f**, g] | [e, c, b, a, f] |
| [**1**, 0] | [d, **g**] | [e, c, b, a, f, g] |
| [**0**] | [**d**] | [e, c, b, a, f, g, d] |

So every possible permutation of objects can be indexed with an integer from 0 to

# Graceful Trees



*3*

*1*

*2*

*4*

*54*

*654*

The graceful tree problem is easily stated: if we label the vertices of a tree with numbers from 0 to m (the number of edges), this induces a labelling on the edges – the label on an edge being the absolute difference between the labels of the vertices at either end of the edge. In the example shown, these edge-labels form a complete sequence from 1 to m. This is called a ‘graceful labelling’ and a tree that is *able* to be labelled in this way is called a ‘graceful tree’.

The Graceful Tree Conjecture is that every tree is graceful.

# Representing trees

In working on this problem I’ve come up with a number of representations of trees, to try and pull out one or other characteristic, ands to make it more tractable to mathematical reasoning.

# Adjacency Matrix

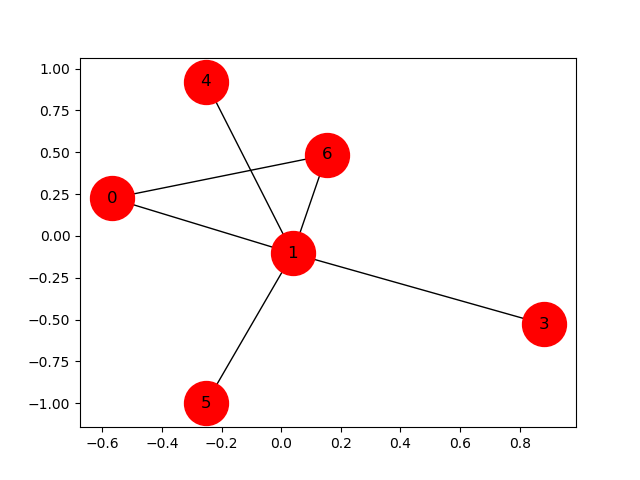
The first is the Adjacency Matrix. This is a matrix filled with ‘0’s and ‘1’s, where a ‘1’ represents an edge. The tree shown above has the following Adjacency Matrix, where the row and column numbers refer to the vertex labels. Zeroes are left out for the sake of avoiding clutter. The principal diagonal is all ‘0’s, as no edge connects a vertex to itself; and the matrix is symmetric, since the edges are undirected.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | ***0*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** |
| ***0*** |  | 1 |  |  | 1 | 1 | 1 |
| ***1*** | 1 |  |  |  |  |  |  |
| ***2*** |  |  |  |  |  | 1 |  |
| ***3*** |  |  |  |  |  | 1 |  |
| ***4*** | 1 |  |  |  |  |  |  |
| ***5*** | 1 |  | 1 | 1 |  |  |  |
| ***6*** | 1 |  |  |  |  |  |  |

The off-diagonals (such as the cells marked in yellow) all represent possible edges with a given label – in the case shown, that label is 2. This is because the edges would join vertices 0‑2, 1‑3, 2‑4, 3‑5, or 4‑6. In a graceful tree (as in this case) each off-diagonal must have one and only one entry (i.e. the number 1). On the yellow diagonal it’s the 3‑5 edge.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***0*** |  |  |  |  |  |  |
|  | ***1*** |  |  |  |  |  |
|  |  | ***2*** |  |  |  |  |
|  |  |  | ***3*** |  |  |  |
|  |  |  |  | ***4*** |  |  |
|  |  |  |  |  | ***5*** |  |
|  |  |  |  |  |  | ***6*** |

Because of its symmetry we can also show this matrix by displaying only the triangle of cells below the principal diagonal, as on the left. I have converted the ‘1’ entries into blacked-out boxes.

Not every such matrix represents a tree. The one on the left represents the graph on the right; notice that the 2-row and 2-column (jointly known as the 2-hook) is empty, meaning that the 2-vertex is not connected. The graph is therefore disjoint. But since it has the right number of edges, this entails that it has a cycle somewhere – in this case, the cycle is

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***0*** |  |  |  |  |  |  |
|  | ***1*** |  |  |  |  |  |
|  |  | ***2*** |  |  |  |  |
|  |  |  | ***3*** |  |  |  |
|  |  |  |  | ***4*** |  |  |
|  |  |  |  |  | ***5*** |  |
|  |  |  |  |  |  | ***6*** |

.

# Edges

A tree can be represented as a collection of edges, given as an unordered pair of vertex labels (but by convention I put the lower number first in the pair, and I order the edges in ascending order of the difference between the two numbers). So the labelled graceful tree shown above has edges (0,1) (3,5) (2,5) (0,4) (0,5) (0,6)

# Adjacency list

This lists, in the 0th position, all the vertices adjacent to ‘0’; in the 1st position all the vertices adjacent to ‘1’; and so on. So the adjacency list for the above graceful tree will be:

0 [1,4,5,6]  
1 [0]  
2 [5]  
3 [5]  
4 [0]  
5 [0,2,3]  
6 [0]

# Cantor representation of trees

There are two ways of using Cantor representation to describe a graph on m edges and m+1 vertices. The first (in my Python code this is called the Tree class) reflects a model of how to build *any* tree. The second (the Stock class) is always graceful (i.e. has a complete sequence of edge-labels) but is not always a *connected* tree.

## Tree Class

To motivate this, imagine building a tree as follows: start with a single vertex (labelled ‘0’) and no edges. Now introduce a new vertex (‘1’) and join it to the first vertex by an edge. Continue to bring in new vertices, consecutively labelled, and join each one to an arbitrary vertex of the existing tree. In this way any tree (ignoring the labels) can be made.

What the labels can do is to allow us to generate an adjacency matrix as we go along. The ‘1’ vertex is joined to the ‘0’ – put an entry in the 0th column of row 1. The ‘2’ vertex is joined to ‘0’ or ‘1’ – put an entry in the 0th or 1st column of row 2. The ‘3’ vertex joins to ‘0’, ‘1’ or ‘2’ – put an entry in the corresponding column of row 3. Continue like this, introducing the kth vertex by putting an entry into one of the cells in row k.

The Cantor representation for this is built up in the same way: moving from right to left we add each term to the list by just noting which vertex the latest addition is joined to. Underneath it you can also note the number of the new vertex, and this gives you a collection of edges. For instance, Tree(3147,7) has a Cantor representation of [4,2,1,0,1,1,0], and its edges are (4, 7), (2,6), (1,5), (0,4), (1,3), (1,2), and (0,1). (The reason we have to specify that this Tree has 7 edges is that we *could* add another by attaching a vertex ‘8’ to vertex ‘0’ and the number of the Tree would still be 3147. In the Python code you can leave out the size value as long as you want the smallest tree with that ident number).

The graph produced by this procedure is guaranteed to be a *tree* – that is, acyclic and connected, planar, bipartite – and every possible tree can be made this way. However, the only way to make the labelling *graceful* is to join every new vertex to the ‘0’ vertex. Given a non-graceful labelling we could then search for graceful labellings by taking every permutation of the labels and applying them to the tree. As there are (m+1)! such permutations (and m! trees to practise them on), a brute-force search would be impractical. There are ways to make this more efficient, but I haven’t spent much time on this approach. However, the Tree class is useful for cataloguing the various topologically distinct trees.

## Stock Class

By contrast with the Tree class, Stocks are often cyclic and unconnected, even though they have the right number of edges. They can be generated from a Cantor representation, but the way of assigning edges is different from the Tree class. The adjacent vertices of a Stock are found by taking the place number of a term – and here I always number places from the left starting at 1 – and adding it to the term. So Stock(3147,7) has the same Cantor representation as Tree(3147,7), but its edges are (4,5), (2,4), (1,4), (0,4), (1,6), (1,7), (0,7).

Every gracefully-labelled tree can be found among the collection of Stocks, so this looks like a promising approach. The chief problem here is that not every such graph is a tree. In fact the proportion of them that *are* trees rapidly falls off as the tree size increases. Starting with a single edge, the number of trees goes thus:

1, 2, 4, 12, 40, 164, 752, 4,020, 23,576, 155,632, 1,112,032

By the time we get to 8 edges, there are 8! Stocks, fewer than 10% of which are connected trees, and the proportion falls by about a third with each new edge. If that pattern continues (and I haven’t proved that it does), the number of topologically distinct unlabelled trees grows much more slowly than the number of gracefully labelled trees. This doesn’t prove that all distinct trees are represented among the (much larger) population of gracefully labelled trees. But it does at least indicate that there is plenty of room for the Conjecture to be true.

So, although we can prove that every graceful tree is a Stock, this doesn’t prove that every tree is graceful. We need to show that every tree (up to isomorphism) is present in the collection of Stocks. So let’s look at how we can identify trees to decide if some arbitrary pair of trees is isomorphic to each other.

# Tree signatures

There is a token, which I am calling a *signature*, that completely specifies a tree, up to isomorphism (i.e. ignoring labelling). Two trees have the same signature iff they are isomorphic.

The algorithm for generating the signature comes from Marthe Bonamy’s paper (see the repository), and she attributes it to Valiente, so I call this the Valiente signature. I’ve modified it slightly – which I explain below – but I’ll start off with Bonamy’s description.

This process is applied to a rooted directed tree, so the first thing to do is to choose a vertex to treat as the root and alter the adjacency list to reflect this change.

Using the Adjacency list from earlier:

0 [1,4,5,6]  
1 [0]  
2 [5]  
3 [5]  
4 [0]  
5 [0,2,3]  
6 [0]

I pick a vertex to be the root, and for now I’ll choose ‘5’. I go through the vertices adjacent to ‘5’ – which are ‘0’, ‘2’ and ‘3’. From *their* lists I remove any ‘5’s:

0 [1,4,6]  
1 [0]  
2 []  
3 []  
4 [0]  
5 [0,2,3]  
6 [0]

Now I go through anything adjacent to *those* vertices, i.e. ‘0’, ‘2’ and ‘3’, and from the lists of vertices in *their* lists I remove any references to (respectively) ‘0’, 2’, and ‘3’. Take ‘0’ first. It’s adjacent to ‘1’, ‘4’ and ‘6’. So from each of their lists I remove any reference to ‘0’:

0 [1,4,6]  
1 []  
2 []  
3 []  
4 []  
5 [0,2,3]  
6 []

This continues until I’ve visited all the vertices that still have non-empty lists, so the job is done, and this is the rooted directed adjacency list.

Now, to generate the signature. Each of the vertices with empty lists I label with the length of its list:

0 3 [1,4,6]   
1 0 []  
2 0 []  
3 0 []  
4 0 []  
5 3 [0,2,3]   
6 0 []

Wherever a labelled vertex is referenced (in column 3) I replace the reference with that vertex’s label, and then append it to the length:

0 3 0,0,0 → 3000  
1 0  
2 0  
3 0  
4 0  
5 3 3(000),0,0 → 3300000  
6 0

Do this repeatedly until the root vertex is labelled:   
5 3,3,0,0,0,0,0

(Actually, I do this recursively. The call is effectively “label the root” which then writes the label “3 + label(0) +label(2)+label(3)” and then recursively fills them in)

At each stage, when I collate the labels of a list of vertices, I arrange them in reverse lexicographic order, so as to guarantee uniqueness.

The main change I’ve made to Bonamy’s procedure is that – again, for the sake of uniqueness - I choose the root to be either the centre vertex or the bicentre of the tree. In the case of a bicentre, I calculate the signature of each candidate root and choose the lexicographically higher of the two.

This algorithm generates a unique signature, and each signature can be unpacked to reconstruct the original unlabelled tree. It gives us a quick way to test whether two graphs (trees, or stocks, or one of each) are isomorphic.

# Mutations

This is now coming to the heart of the approach I’m taking. I want to find a way of mutating a graceful tree so that (a) the result is guaranteed to be also graceful, and a tree, and (b) the signature of a tree will tell me what mutations are needed to produce it. If I can do this I can derive a graceful labelling from the mutations and thereby prove the Conjecture.

The base case I’m using – the universally available starting point, a guaranteed graceful tree of size m – is the star [0,0,0,0,0…0]. I have a way of mutating it using permutations of vertices, but as yet I haven’t worked out how to churn out all possible mutations, and I haven’t even started thinking about how to work out whether there are enough to generate all the trees I need. Reverse engineering a signature to find its mutation is somewhere over the horizon.

What follows is some of my thinking so far.

A mutation takes a tree – usually the star with n edges, Sn – and moves links around. Specifically, it should take a set of edges with their labels and detach them from one vertex to reattach them to another, while preserving the set of edge labels. For example, take S8 and apply the mutation (2,3,4,1,2). This takes the Cantor representation [0,0,0,0,0,0,0,0] and puts a 2 in place 3, 3 in place 4, 4 in place 1 and 1 in place 2 to give [4,1,2,3,0,0,0,0] . This guarantees that the mutated tree is still a tree and still graceful.

A mutation like (2,3,4,1,2) can be built up as follows:

0

1

2

3

4

5

6

7

8

0

1

2

3

4

5

6

7

8

1. Make a list of all the non-zero vertices {1,2,…n}
2. Delete a number q from the list such that 0 < q <n/2 and start the mutation with (q,q).
3. Take two consecutive entries, r and s, in the mutation.
4. Using the larger of the two (say s) pick any number t from the remaining list such that t<n-s
5. Delete t from the list and insert it between r and s.
6. Stop when you like, or when there are no usable numbers left, whichever comes first.

So we might construct the mutation above by going

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **List** | **Mutation** | **choose r,s** | **Max(r,s)** | **n-s** | **choose t** | **Cantor** |
| 1,2,3,4,5,6,7,8 |  |  |  |  |  | [0,0,0,0,0,0,0,0] |
| 1,\_,3,4,5,6,7,8 | 2,2 | 2,2 | 2 | 6 | 4 | [0,2,0,0,0,0,0,0] |
| 1,\_,3,\_,5,6,7,8 | 2,4,2 | 2,4 | 4 | 4 | 3 | [0,4,0,2,0,0,0,0] |
| 1,\_,\_,\_,5,6,7,8 | 2,3,4,2 | 4,2 | 4 | 4 | 1 | [0,4,2,3,0,0,0,0] |
| \_,\_,\_,\_,5,6,7,8 | 2,3,4,1,2 |  |  |  |  | [4,1,2,3,0,0,0,0] |

We stopped here, but we could equally continue with (2,3,4,1,5,2) which gives [4,5,2,3,1,0,0,0] .

0

1

2

3

4

5

6

7

8

But (to revert to a smaller tree) what about [0,1,2,0,0]? Can it be made from [0,0,0,0,0] using a mutation? It looks like that should start with (2,3…) because 2 goes to the 3rd place, but where does the 3 go? In the tree it attaches to 1, but in the Cantor representation that means that 1 is in the 2nd place. That would mean that our mutation must end (….1,2). So what goes in the 1st place in the Cantor rep – what should precede that ‘1’ in the mutation? A zero. So do we write (2,3,0,1,2)? But that would mean 3 going into the 0th place, which doesn’t exist. Perhaps we can write , where the bar means that what goes into place 2 isn’t the 3 but 3-2=1. So, for example, that last S8 mutation, (2,3,4,1,5,2), could yield a different mutation , which means

0

1

2

3

4

5

2→3

3→4

4→1

1→5

(=5-2(=3))→2

Which gives [4,3,2,3,1,0,0,0] . This detaches the 5 from the 7 and attaches it to the 3.

0

1

2

3

4

5

6

7

8

There will be some trees that need more than one mutation; for example, [1,2,3,0,0,0] can be built with (1,1) (2,2) and (3,3) combined, but there isn’t a single permutation set that will do the job. So (for example) (1,3,2,1) + (4,4) is a compound mutation, made from its two constituent parts, and gives us [2,3,1,4,0,0,0,0] (right).

4

7

2

3

0

5

6

1

8

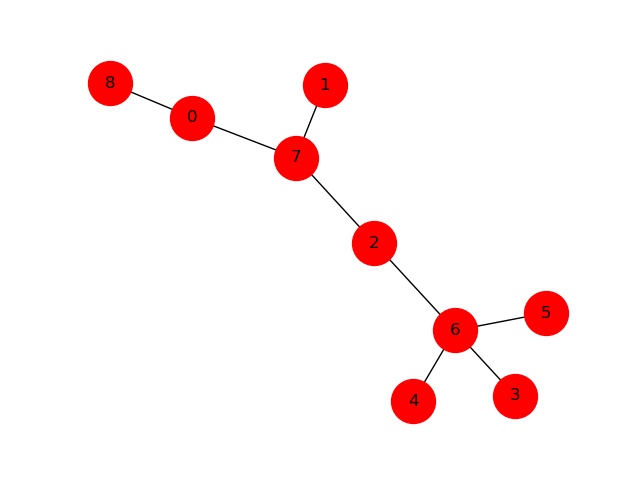
So. Here are my questions:

1. **Can I bar *any* element in the mutation?**I can certainly argue that the element that follows the barred element must be bigger than the following element (i.e in a pair , ). Apart from that, I can’t think of any objection to barring anything.
2. **Can I bar any *number* of elements in the mutation?**Subject to the above rule, yes, it looks like it.
3. **Will this always conserve both gracefulness and treehood?**Again, it seems so – I can’t see why not, and I know of no countrerexamples. Still, I haven’t got a proof.
4. **Can every graceful tree now be generated from a star Sn?**I’m currently looking for counterexamples. No thoughts yet.
5. **And, the 64,000 dollar question – can *every* distinct tree be generated from a star by some mutation?**I have no idea. But, if so – job done! I’m looking for a way of deriving mutations from Valiente signatures.

# Signatures

The signature of a tree is a unique string, that can always be calculated from the tree, and from which the tree can always be reconstructed. I’m using the system described in Marthe Bonamy’s paper, for which she references Valiente, so I’m calling this the Valiente signature. My two small adjustments are:

1. At each stage I take the labels in my list and sort them in descending lexicographic order – so ‘21010, 22000, 31000’ would become ‘31000,22000,21010’
2. I root the tree in its centre vertex. In the case of a bicentre, I calculate both possible signatures and then choose the higher one. So, given a choice between 230002100 and 313000100 (which do, indeed, describe the same tree) I choose the latter.

I’m looking into the possibility that a signature can be used to generate an adjacency matrix. So that signature, 313000100, tells me I have In the matrix) a line of three cells, then one at right-angles to it, then three more in a line: also coming off the original line of three I will have a single cell at right angles. So…

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |  |  |  |
|  | 1 |  |  |  |  |  |  |  |
|  |  | 2 |  |  |  |  |  |  |
|  |  |  | 3 |  |  |  |  |  |
|  |  |  |  | 4 |  |  |  |  |
|  |  |  |  |  | 5 |  |  |  |
|  |  |  |  |  |  | 6 |  |  |
|  |  |  |  |  |  |  | 7 |  |
|  |  |  |  |  |  |  |  | 8 |

[5,4,3,2,2,1,0,0]

The tree is noble (entries in the Cantor representation go up stepwise by either 1 or 0 as you go from right to left), and we already know that, eventually, there can’t be enough noble trees to account for all distinct trees (from 11 edges up there are more distinct trees than noble trees). Can I find an algorithm that will generate an adjacency matrix for any Valienté signature (including trees with no noble labelling) and, if so, can it always be graceful?

Also: Can I generate all and only valid signatures? Part of the problem I’m facing is that a change in a tree – budding a new vertex, or moving vertices around – will commonly change the centre, so the relationship between the two signatures will be obscured.

For a tree with 8 edges the valid signatures are:

The first number is the degree of the central vertex. This is followed by the signatures of the sub-trees that depend from it, in descending order; all numbers from here on in represent the degree of that vertex minus one. The sum of all numbers is the size of the tree (in this case 8). The depths of the two deepest chains must differ by no more than 1. So, for instance, has three dependent trees, with depths of 4,( ), 3,( ), and 1 ( )

The sizes of the dependent trees correspond to some partition of the total number given the central number. For example, the number 8 has 22 partitions, of which 5 have largest part 4:

4+4 4+3+1 4+2+2 4+2+1+1 4+1+1+1+1

After each initial 4, the numbers that follow it can themselves be partitioned in various ways, which correspond to the different subtrees. So

4+4 :

4+3+1 :

:

: 4+2+2 :

:

: 4+2+1+1: : 4+1+1+1+1:

You might be able to think of valid subpartitions that would give signatures that don’t appear in this list; but I suspect they will fail because their two longest subtrees are not the same depth. For example, falls under the 4+4 partition, but it has one subtree of depth 3 and three of depth 1 – which means it’s not rooted at its centre. If you draw it out and re-calculate the signature you get **,** which is already in the list. Or you could think of taking that signature and swapping the root between the 4 and the 3. That moves the 3 to the front and adds 1 (because the root is fiven its full degree), and hangs the 4 off the 3 (and subtracts 1 from it, because it’s not the root any more).