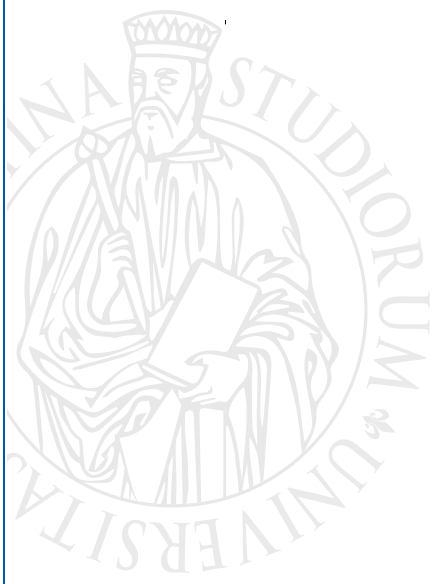
UNIVERSITÀ DEGLI STUDI FIRENZE
DISIA
DIPARTIMENTO DI STATISTICA, INFORMATICA, APPLICAZIONI "GIUSEPPE PARENTI"

Crawling, (pretty) printing and graphing the OEIS

Massimo Nocentini, Donatella Merlini



DISIA WORKING PAPER
2018/06

Crawling, (pretty) printing and graphing the OEIS

Massimo Nocentini and Donatella Merlini

Dipartimento di Statistica, Informatica, Applicazioni
Università di Firenze

October 30, 2018

Abstract

In this paper we present a suite of software tools that allows us to interact with the *Online Encyclopedia of Integer Sequences*; in particular, (i) a *crawler* fetches sequences recursively with respect to their relations, (ii) a *pretty printer* represents the same data stored in the archive using two different formats, namely the old console and modern Jupyter notebooks, (iii) a *grapher* shows connections among sequences by using graph structures.

1 Introduction

The *Online Encyclopedia of Integer Sequences* [Sloane] is an online database of sequences of numbers that collects any kind of data regarding them, available at `https://oeis.org/`. It was founded by N. J. A. Sloane in 1964 and since then has been, and continue to be, updated constantly by contributions of many users. Despite of its powerful searching mechanisms, shown in Figure 1 which reports results about the well known sequence of Fibonacci numbers (see e.g. [Graham et al., 1989]), we design a parallel *suite of software tools* that satisfies the necessities (i) to search the OEIS offline by downloading repeated searches, (ii) to work in the console to use programming facilities to manipulate contents more effectively and (iii) to interface with third-party libraries to visualize networks encoding connections among sequences.

Looking for similar approaches in the recent literature, [Nguyen and Taggart, 2013] mines the OEIS for new mathematical identities, discussing how to store, compare and match integer sequences toward the formalization of some conjectures; on the other hand, searching the word "*oeis*" in GitHub returns one hundred repositories, the majority of them host simple implementations of scripts that download data about a given sequence, targeting all major programming languages. Moreover, [Weidmann] is a project that identifies number sequences given a list of numbers and gives a formula that generates them.

Our approach complements the former ones because it provides a *recursive* and *asynchronous* fetching process, vanilla data storage in JSON files and visualization of sequences' relations; the description of each tool is addressed in the following sections, respectively.

The present suite of tools had been shown at an open school on Combinatorial Method in the analysis of Algorithms and Data Structures in Korea [Nocentini, 2017]; moreover, all the sources that implements the applications can be found online in the repository `https://github.com/massimo-nocentini/oeis-tools`.

2 The Crawler

The script `crawling.py` implements a bot that given a sequence identifier in the form Axxxxxx, where xs are digits, issues an HTTP request to the main OEIS server and waits for a response; once

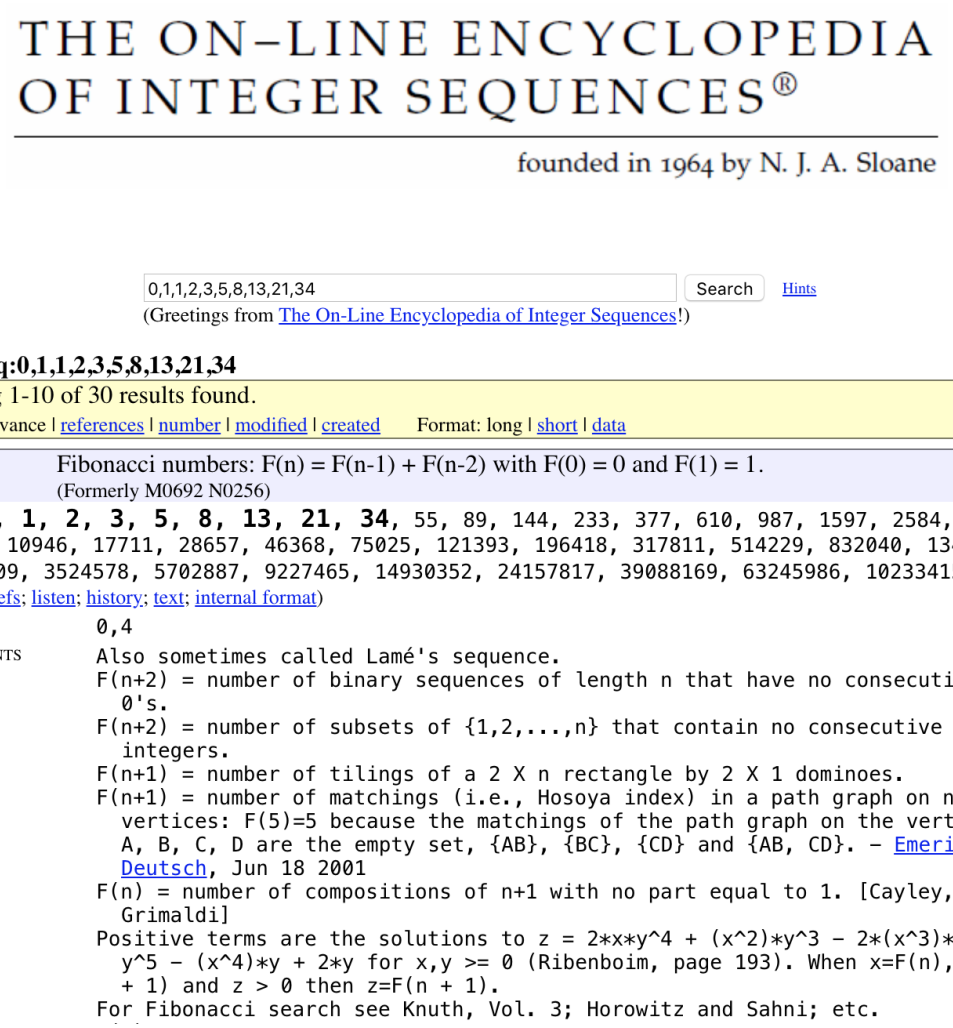


Figure 1: The OEIS search page: the present snapshot shows search results about the sequence of Fibonacci numbers

it is received, the bot stores it locally and, looking into the response's `xref` section that contains a set of other sequences identifiers, repeats its behaviour on each one of them, recursively. Such a bot is commonly known as *crawler*.

Our implementation features neither threads nor race conditions nor data sync; on the contrary, it targets *pure asynchronous computation* by using *async/await* Python primitives only. The approach is educational and we strive to create a simple but elegant codebase which boils down to 300 lines of Python code; eventually, it allows us to cache portions of the OEIS to speed up repeated lookups and to restart the fetching process from the cache already downloaded.

The script presents a help message to explain itself:

```
$ python3.6 crawling.py -h
usage: crawling.py [-h] [--clear-cache] [--restart] [--workers WORKERS]
                   [--log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                   [--cache-dir CACHE_DIR] [--progress-mark PROGRESS_MARK]
                   [S [S ...]]

OEIS Crawler.

positional arguments:
  S                     Sequence to fetch, given in the form Axxxxxx

optional arguments:
  -h, --help            show this help message and exit
  --clear-cache         Clear cache of sequences, according to --cache-dir
  --restart             Build fringe from cached sequences (defaults to False)
  --workers WORKERS     Degree of parallelism (defaults to 10)
  --log-level {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Logger verbosity (defaults to ERROR)
  --cache-dir CACHE_DIR
                        Cache directory (defaults to ./fetched/)
  --progress-mark PROGRESS_MARK
                        Symbol for fetched event (defaults to ●)
```

Example 1. *We illustrate a typical session where we start from scratch. First of all, we want to download the OEIS content about two important and nice sequences, namely those corresponding to the Fibonacci and Catalan numbers (see e.g. [Graham et al., 1989, Stanley, 2015]), known in the OEIS by identifiers* A000045 *and* A000108*, respectively:*

```
$ python3.6 crawling.py A000045 A000108
●●●●●●●●●●●●●●●●●●●●●●●●●●●●●●^C
fetched 30 new sequences:
{'A006480', 'A000045', 'A032443', 'A046161', 'A002390',
 'A137697', 'A000142', 'A000736', 'A094639', 'A176137',
 'A120303', 'A000344', 'A032357', 'A000753', 'A048990',
 'A003519', 'A000108', 'A009766', 'A211611', 'A120274',
 'A008276', 'A129763', 'A094216', 'A106566', 'A003518',
 'A098597', 'A099039', 'A014137', 'A000245', 'A264663'}
```

After stopping the crawler, we check the content of cache with the commands

```
$ python3.6 crawling.py
30 sequences in cache ./fetched/
289 sequences in fringe for restarting
$ ls fetched/
A000045.json    A000245.json    A000753.json    A003519.json
A009766.json    A032443.json    A094216.json    A099039.json
A120303.json    A176137.json    A000108.json    A000344.json
```

```
A002390.json    A006480.json    A014137.json    A046161.json
A094639.json    A106566.json    A129763.json    A211611.json
A000142.json    A000736.json    A003518.json    A008276.json
A032357.json    A048990.json    A098597.json    A120274.json
A137697.json    A264663.json
```

which tell us that 30 *sequences had been fetched and stored in the default directory* `./fetched/`*. Moreover, we can restart the crawler from where it was interrupted with the command*

```
$ python3.6 crawling.py --restart
●●●●●●●●●●●●●●●●●●●●^C
fetched 20 new sequences:
{'A000165', 'A001044', 'A003517', 'A027914', 'A214292', 'A152063',
 'A014138', 'A062103', 'A003422', 'A238717', 'A045520', 'A064062',
 'A144107', 'A045525', 'A007004', 'A002057', 'A244230', 'A099731',
 'A033552', 'A121839'}
```

and we check that new sequences are actually collected,

```
$ python3.6 crawling.py
50 sequences in cache ./fetched/
354 sequences in fringe for restarting
```

as desired.

Having contents stored in JSON files allows us to inspect and manipulate them using every tool available in our working environment, as the next example shows.

Example 2. *Combining the* `cat` *command with the Python module* `json.tool`*, that prints JSON files with respect to indentation, we can visualize data about the sequence of Fibonacci numbers as follows*

```
$ cat fetched/A000045.json | python3.6 -m json.tool
{
    "greeting": "Greetings from The On-Line Encyclopedia of Integer Sequences! http://oeis.org/",
    "query": "id:A000045",
    "count": 1,
    "start": 0,
    "results": [
        {
            "number": 45,
            "id": "M0692 N0256",
            "data": "0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,6765,10946,
                     17711,28657,46368,75025,121393,196418,317811,514229,832040,1346269,2178309,
                     3524578,5702887,9227465,14930352,24157817,39088169,63245986,102334155",
            "name": "Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.",
            "comment": [
                "Also sometimes called Lam\u00e9's sequence.",
                "F(n+2) = number of binary sequences of length n that have no consecutive 0's.",
                "F(n+2) = number of subsets of {1,2,...,n} that contain no consecutive integers.",
                "F(n+1) = number of tilings of a 2 X n rectangle by 2 X 1 dominoes.",
                ... # more comments here
            ]
            ... # more sections here
        }
    ]
}
```

Our implementation takes strong inspiration from [van Rossum and Davis] and provides the following main abstractions:

reader objects, that have the responsibility to be *asynchronous iterators*, having to respond to the message `__anext__` where the computation waits asynchronously for incoming data from the `self.read` coroutine. The code

```
class reader:

    def __init__(self, read):
        self.read = read

    def __aiter__(self):
        return self

    async def __anext__(self):
        chunk = await self.read()
        if chunk:   return chunk
        else:       raise StopAsyncIteration
```

implements the description precisely.

fetcher objects have the responsibilities to (i) create a socket with OEIS server, (ii) establish a working connection, (iii) send an HTTP `GET` request for the desired sequence, (iv) wait for the fetching process completes and (v) close the socket and signal that the work ends successfully.

```
class fetcher:

    def __init__(   self, url,
                    resource_key=lambda resource: resource,
                    done=lambda url, content: print(content)):

        self.url = url
        self.response = b''
        self.sock = None
        self.done = done
        self.resource_key=lambda: resource_key(self.url.resource)

    def encode_request(self, encoding='utf8'):

        request = 'GET {} HTTP/1.0\r\nHost: {}\r\n\r\n'.format(
                self.resource_key(), self.url.host)

        return request.encode(encoding)

    async def fetch(self):

        self.sock = socket.socket()
        self.sock.setblocking(False)

        await loop.sock_connect(self.sock, address=(self.url.host, self.url.port))

        logger.info('Connection established with {} asking resource {}'.format(
                self.url.host, self.url.resource))

        await loop.sock_sendall(self.sock, self.encode_request())

        self.response = await self.read_all()

        self.sock.close()
```

```
            return self.done(self.url, self.response.decode('utf8'))

        async def read(self, nbytes=4096):

            chunk = await loop.sock_recv(self.sock, nbytes)
            return chunk

        async def read_all(self):

            response = [chunk async for chunk in reader(self.read)]
            return b''.join(response)
```

crawler objects have the responsibilities (i) to keep a queue of task, one for each candidate sequence, (ii) to put each ready task into the scheduling process and (iii) to reclaim memory for completed task and, eventually, (iv) to deque them.

```
    class crawler:

        def __init__(self, resources, fetcher_factory, max_tasks):

            self.resources = resources
            self.max_tasks = max_tasks
            self.fetcher_factory = fetcher_factory
            self.q = asyncio.Queue()

        async def crawl(self):

            for res in self.resources: self.q.put_nowait(res)

            tasks = [loop.create_task(coro=self.work()) for _ in range(self.max_tasks)]

            await self.q.join()

            for t in tasks: t.cancel()


        async def work(self):

            while True:

                resource = await self.q.get()

                await self.fetcher_factory(resource, appender=self.q.put_nowait).fetch()

                self.q.task_done()
```

3 The (Pretty) Printer

The script `pprinting.py` provides a proxy for searching into the OEIS, therefore it shows exactly the same contents you see from usual web interface on `http://oeis.org`; additionally, it provides (i) tabular representations of `data` sections in *one and two dimensions* using list and matrix notations, respectively, (ii) filtering capabilities on most response's sections and (iii) interoperability with the crawler tool by taking advantage of cached sequences.

The script presents a help message to explain itself:

```
$ python3.6 pprinting.py -h
usage: pprinting.py [-h]
                    (--id ID | --seq SEQ | --query QUERY | --most-recents M)
                    [--force-fetch] [--cache-dir CACHE_DIR] [--tables-only]
                    [--start-index S] [--max-results R] [--data-only]
                    [--upper-limit U] [--comment-filter C]
                    [--formula-filter F] [--xrefs-filter X] [--link-filter L]
                    [--cite-filter R] [--console-width W]

OEIS Pretty Printer.

optional arguments:
  -h, --help            show this help message and exit
  --id ID               Sequence id, given in the form Axxxxxx
  --seq SEQ             Literal sequence, ordered '[...]' or presence '{...}'
  --query QUERY         Open query for plain search, in the form '...'
  --most-recents M      Print the most recent sequences ranking by M in ACCESS
                        or MODIFY, looking into --cache-dir, at most --max-
                        results (defaults to None)
  --force-fetch         Bypass cache fetching again, according to --cache-dir
                        (defaults to False)
  --cache-dir CACHE_DIR
                        Cache directory (defaults to ./fetched/)
  --tables-only         Print matrix sequences only (defaults to False)
  --start-index S       Start from result at rank position S (defaults to 0)
  --max-results R       Pretty print the former R <= 10 results (defaults to 10)
  --data-only           Show only data repr and preamble (defaults to False)
  --upper-limit U       Upper limit for data repr: U is a dict '{"list":i,
                        "table":(r, c)}' where i, r and c are ints (defaults
                        to i=15, r=10 and c=10), respectively)
  --comment-filter C    Apply filter C to comments, where C is Python `lambda`
                        predicate 'lambda i,c: ...' referring to i-th comment c
  --formula-filter F    Apply filter F to formulae, where F is Python `lambda`
                        predicate 'lambda i,f: ...' referring to i-th formula f
  --xrefs-filter X      Apply filter X to cross refs, where X is Python
                        `lambda` predicate 'lambda i,x: ...' referring to i-th xref x
  --link-filter L       Apply filter L to links, where L is Python `lambda`
                        predicate 'lambda i,l: ...' referring to i-th link l
  --cite-filter R       Apply filter R to citation, where R is Python `lambda`
                        predicate 'lambda i,r: ...' referring to i-th citation r
  --console-width W     Console columns (defaults to 72)
```

In the next examples we show how `pprinting`'s facilities can be used to apply filters, to print data-only visualization and to search by an open query, respectively.

Example 3. *Typing the following command into a shell, it outputs on the* `stdout` *the pretty-printed contents about the sequence of Fibonacci numbers, with two filters applied that show comments made by prof. Barry and the first* 5 *formulae only,*

```
$ python3.6 pprinting.py                          \
    --id A000045                                  \
    --comment-filter 'lambda i,c: "Barry" in c' \
    --formula-filter 'lambda i,f: i < 5'

 A000045 - Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and
  F(1) = 1.
```

```
by _N. J. A. Sloane_, 1964

_Keywords_: `nonn,core,nice,easy,hear,changed`

_Data_:
[0  1  1  2  3  5  8  13  21  34  55  89  144  233  377]

_Comments_:
    ● F(n+2) = Sum_{k=0..n} binomial(floor((n+k)/2),k), row sums of
      A046854. - _Paul Barry_, Mar 11 2003

_Formulae_:
    ● G.f.: x / (1 - x - x^2).
    ● G.f.: Sum_{n>=0} x^n * Product_{k=1..n} (k + x)/(1 + k*x). - _Paul
      D. Hanna_, Oct 26 2013
    ● F(n) = ((1+sqrt(5))^n - (1-sqrt(5))^n)/(2^n*sqrt(5)).
    ● Alternatively, F(n) = ((1/2+sqrt(5)/2)^n -
      (1/2-sqrt(5)/2)^n)/sqrt(5).
    ● F(n) = F(n-1) + F(n-2) = -(-1)^n F(-n).

_Cross references_:
    ● Cf. A039834 (signed Fibonacci numbers), A001690 (complement),
      A000213, A000288, A000322, A000383, A060455, A030186, A020695,
      A020701, A071679, A099731, A100492, A094216, A094638, A000108,
      A101399, A101400, A001611, A000071, A157725, A001911, A157726,
      A006327, A157727, A157728, A157729, A167616, A059929, A144152,
      A152063, A114690, A003893, A000032, A060441, A000930, A003269,
      A000957, A057078, A007317, A091867, A104597, A249548, A262342,
      A001060, A022095, A072649.
    ● First row of arrays A103323, A234357. Second row of arrays
      A099390, A048887, and A092921 (k-generalized Fibonacci numbers).
    ● a(n) = A094718(4, n). a(n) = A101220(0, j, n).
    ● a(n) = A090888(0, n+1) = A118654(0, n+1) = A118654(1, n-1) =
      A109754(0, n) = A109754(1, n-1), for n > 0.
    ● Fibonacci-Pascal triangles: A027926, A036355, A037027, A074829,
      A105809, A109906, A111006, A114197, A162741, A228074.
    ● Boustrophedon transforms: A000738, A000744.
    ● Powers: A103323, A105317, A254719.
    ● Numbers of prime factors: A022307 and A038575.
    ● Cf. A163733.
```

other sections, such as `reference` *and* `link`*, are hidden by default to provide a cleaner output.*

Example 4. *The following command pretty prints (i) the first 3 sequences from our current cache –the result may vary if you try on your own machine–, (ii) ranking them according to the most recent access time, (iii) reporting data only and (iv) limiting up to* 10 *coefficients for linear sequences:*

```
$ python3.6 pprinting.py        \
    --most-recent ACCESS        \
    --data-only                 \
    --max-results 3             \
    --upper-limit '{"list":10}'

 A001044 - a(n) = (n!)^2.
```

```
by _N. J. A. Sloane_, _R. K. Guy_

_Keywords_: `nonn,easy,nice`

_Data_:
[1  1  4  36  576  14400  518400  25401600  1625702400  131681894400]
```

```
 A048990 - Catalan numbers with even index (A000108(2*n), n >= 0): a(n)
  = binomial(4*n, 2*n)/(2*n+1).

by _Wolfdieter Lang_

_Keywords_: `easy,nonn`

_Data_:
[1  2  14  132  1430  16796  208012  2674440  35357670  477638700]
```

```
 A014138 - Partial sums of (Catalan numbers starting 1, 2, 5, ...).

by _N. J. A. Sloane_

_Keywords_: `nonn,nice`

_Data_:
[0  1  3  8  22  64  196  625  2055  6917]
```

Example 5. *The following command pretty prints (i) response about the open query "*`pascal triangle`*", (ii) using* 2*-dimension representation for matrices in* `data` *sections and (iii) reports the first* 2 *sequences in the returned list only,*

```
$ python3.6 pprinting.py
    --query 'pascal triangle'   \
    --tables-only               \
    --data-only                 \
    --max-results 2

 A007318 - Pascal's triangle read by rows: C(n,k) = binomial(n,k) =
  n!/(k!*(n-k)!), 0 <= k <= n.

by _N. J. A. Sloane_ and _Mira Bernstein_, Apr 28 1994

_Keywords_: `nonn,tabl,nice,easy,core,look,hear,changed`

_Data_:
⎡1  0  0   0    0    0   0   0   0  0⎤
⎢1  1  0   0    0    0   0   0   0  0⎥
⎢1  2  1   0    0    0   0   0   0  0⎥
⎢1  3  3   1    0    0   0   0   0  0⎥
⎢1  4  6   4    1    0   0   0   0  0⎥
```

```
⎢1  5  10  10   5    1   0   0   0  0⎥
⎢1  6  15  20  15    6   1   0   0  0⎥
⎢1  7  21  35  35   21   7   1   0  0⎥
⎢1  8  28  56  70   56  28  8   1  0⎥
⎣1  9  36  84  126  126  84  36  9  1⎦
```

```
 A047999 – Sierpiński's [Sierpinski's] triangle (or gasket): triangle,
  read by rows, formed by reading Pascal's triangle mod 2.

by _N. J. A. Sloane_

_Keywords_: `nonn,tabl,easy,nice`

_Data_:
⎡1  0  0  0  0  0  0  0  0  0⎤
⎢1  1  0  0  0  0  0  0  0  0⎥
⎢1  0  1  0  0  0  0  0  0  0⎥
⎢1  1  1  1  0  0  0  0  0  0⎥
⎢1  0  0  0  1  0  0  0  0  0⎥
⎢1  1  0  0  1  1  0  0  0  0⎥
⎢1  0  1  0  1  0  1  0  0  0⎥
⎢1  1  1  1  1  1  1  1  0  0⎥
⎢1  0  0  0  0  0  0  0  1  0⎥
⎣1  1  0  0  0  0  0  0  1  1⎦
```

In parallel of the textual interface, we develop pretty printing functions that integrates in Jupyter notebooks. The aim remains the same, namely to present contents taken from the OEIS targeting a different environment that accepts their representation; this is the time of a dynamic web interface that allows us to evaluate Python code on the fly. Using Jupyter's Markdown language to write textual content, we propose another view of the same data, as shown in Figures 2, 3 and 4; in particular, we take advantage of (i) hyper-references to make sequences labels clickable to quickly visit them in a new tab, (ii) font styles to emphasize words in italics and (iii) bold-face and to render math notations properly such as 2-dimensional array representation for matrices.

```
In [11]: searchable = search(A_id='A000045', cache_info={'cache_dir':'./fetched/', 'cache_first':False})
```

•

```
In [12]: searchable(comment=lambda i,c: "binomial" in c, formula=lambda i,f:False)
```

Out[12]: *Results for query:* *https://oeis.org/search?fmt=json&start=0&q=id%3AA000045*

A000045: Fibonacci numbers: F(n) = F(n-1) + F(n-2) with F(0) = 0 and F(1) = 1.

by *N. J. A. Sloane*, 1964

Keywords: `nonn,core,nice,easy,hear,changed`

Data:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$A000045(n)$	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377

Comments:

- F(n+2) = Sum_{k=0..n} binomial(floor((n+k)/2),k), row sums of A046854. - *Paul Barry*, Mar 11 2003
- The sequence F(n) is the binomial transformation of the alternating sequence (-1)^(n-1)*F(n), whereas the sequence F(n+1) is the binomial transformation of the alternating sequence (-1)^n*F(n-1). Both of these facts follow easily from the equalities a(n;1)=F(n+1) and b(n;1)=F(n) where a(n;d) and b(n;d) are so-called "delta-Fibonacci" numbers as defined in comments to A014445 (see also the papers of Witula et al.). - *Roman Witula*, Jul 24 2012
  - Let P(x) = x/(1+x) with comp. inverse Pinv(x) = x/(1-x) = -P[-x], and C(x)= [1-sqrt(1-4x)]/2, an o.g.f. for the shifted Catalan numbers A000108, with inverse Cinv(x) = x * (1-x).
  - Fin(x) = P[C(x)] = C(x)/[1 + C(x)] is an o.g.f. for the Fine numbers, A000957 with inverse Fin^(-1)(x) = Cinv[Pinv(x)] = Cinv[-P(-x)].
  - Mot(x) = C[P(x)] = C[-Pinv(-x)] gives an o.g.f. for shifted A005043, the Motzkin or Riordan numbers with comp. inverse Mot^(-1)(x) = Pinv[Cinv(x)] = (x - x^2) / (1 - x + x^2) (cf. A057078).
  - BTC(x) = C[Pinv(x)] gives A007317, a binomial transform of the Catalan numbers, with BTC^(-1)(x) = P[Cinv(x)].
  - Fib(x) = -Fin[Cinv(Cinv(-x))] = -P[Cinv(-x)] = x + 2 x^2 + 3 x^3 + 5 x^4 + ... = (x+x^2)/[1-x-x^2] is an o.g.f. for the shifted Fibonacci sequence A000045, so the comp. inverse is Fib^(-1)(x) = -C[Pinv(-x)] = -BTC(-x) and Fib(x) = -BTC^(-1)(-x).
  - Generalizing to P(x,t) = x /(1 + t*x) and Pinv(x,t) = x /(1 - t*x) = -P(-x,t) gives other relations to lattice paths, such as the o.g.f. for A091867, C[P[x,1-t]], and that for A104597, Pinv[Cinv(x),t+1].

Cross references:

- Cf. A039834 (signed Fibonacci numbers), A001690 (complement), A000213, A000288, A000322, A000383, A060455, A030186, A020695, A020701,

Figure 2: This screenshot shows search results about the Fibonacci numbers where (i) the section about comments is filtered such that the word "*binomial*" has to appear in their text and (ii) the section about formulae is hidden.

```
In [20]: searchable = search(seq=[1,1,2,5,14,42, 132, 429], max_results=4)
```

•

```
In [24]: searchable(data_only=True)
```

Out[24]: *Results for query:* https://oeis.org/search?fmt=json&start=0&q=1%2C+1%2C+2%2C+5%2C+14%2C+42%2C+132%2C+429

A000108: Catalan numbers: C(n) = binomial(2n,n)/(n+1) = (2n)!/(n!(n+1)!). Also called Segner numbers.

by *N. J. A. Sloane*

Keywords: `core,nonn,easy,eigen,nice`

Data:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$A000108(n)$	1	1	2	5	14	42	132	429	1430	4862	16796	58786	208012	742900	2674440

A120588: G.f. satisfies: 3*A(x) = 2 + x + A(x)^2, with a(0) = 1.

by *Paul D. Hanna*, Jun 16 2006, Jan 24 2008

Keywords: `nonn,easy`

Data:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$A120588(n)$	1	1	1	2	5	14	42	132	429	1430	4862	16796	58786	208012	742900

Figure 3: This screenshot shows search results of a query using a subsequence, showing *data* sections only.

```
In [28]: searchable = search(query="pascal", table=True)
```

•

```
In [30]: searchable(comment=lambda i,c: i in range(5))
```

Out[30]: *Results for query:* https://oeis.org/search?fmt=json&start=0&q=pascal+keyword%3Atabl

A007318: Pascal's triangle read by rows: C(n,k) = binomial(n,k) = n!/(k!*(n-k)!), 0 <= k <= n.

by *N. J. A. Sloane* and *Mira Bernstein*, Apr 28 1994

Keywords: `nonn,tabl,nice,easy,core,look,hear`

Data:

n,k	0	1	2	3	4	5	6	7	8	9
0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

Comments:

- C(n,k) = number of k-element subsets of an n-element set.
- Row n gives coefficients in expansion of (1+x)^n.
- Binomial(n+k-1,n-1) is the number of ways of placing k indistinguishable balls into n boxes (the "bars and stars" argument - see Feller).
- Binomial(n-1,k-1) is the number of compositions (ordered partitions) of n with k summands.
- Binomial(n+k-1,k-1) is the number of weak compositions (ordered weak partitions) of n into exactly k summands. - *Juergen Will*, Jan 23 2016

Figure 4: This screenshot shows search results of an open query using the "*pascal*" keyword, representing the *data* section as a 2-dimensional array.

4 The Grapher

The script `graphing.py` allows us to represent networks where vertices are sequences and edges are connections among them, according to `xref` sections in their JSON encodings. It integrates with the crawler tool by parsing the fetched files and creates `Graph` objects, defined in the Python module `networkx`, having different layouts according to a set of drawing algorithms.

It presents a help message to explain itself:

```
$ python3.6 graphing.py -h
usage: graphing.py [-h] [--directed] [--cache-dir CACHE_DIR]
                   [--graphs-dir GRAPHS_DIR] [--dpi DPI] [--layout LAYOUT]
                   F

OEIS grapher.

positional arguments:
  F                     Save image in file F.

optional arguments:
  -h, --help            show this help message and exit
  --directed            Draw directed edges
  --cache-dir CACHE_DIR
                        Cache directory (defaults to ./fetched/)
  --graphs-dir GRAPHS_DIR
                        Graphs directory (defaults to ./graphs/)
  --dpi DPI             Resolution in DPI (defaults to 600)
  --layout LAYOUT       Graph layout, choose from: {RANDOM, CIRCULAR, SHELL,
                        FRUCHTERMAN-REINGOLD, SPRING, SPECTRAL} (defaults to
                        SHELL)
```

Example 6. *The following command draws the graph shown in Figure 5, where the width of each vertex grows according to the number of its incoming connections,*

```
$ python3.6 graphing.py --layout FRUCHTERMAN-REINGOLD graph.png
```

in order to emphasize most referenced sequences.

Moreover, it can extract essential data from the whole set of JSON files, such as the list of vertices and edges, to interface with third-party software tools that provide different visualizations; in particular, libraries using the *Javascript* programming language are very powerful and the output they produce are very expressive. For our purposes, we use the `arborjs` library (freely available at `http://arborjs.org/`) to display two additional graphs described in the next two examples, respectively.

Example 7. *Figure 6 reports a new unlabeled graph that shows the underlying structure of sequences connections. Here, the layout spreads vertices such that the ones having many outgoing connections are centered, while those having poor connectivity are left on borders.*

Under the hood, the Fibonacci and Catalan numbers are the two central sequences and both of them have an orbit which contains a set of highly connected sequences.

Example 8. *On the other hand, Figure 7 adds labels and colors to vertices in order to spot their identity and their relevance according to a combination of their properties. In particular, each color is represented by an RGB tuple that gets weigths (i) the number of comments and formulae for red, (ii) the number of references and links for green and (iii) the number of incoming and outgoing connections for blue, respectively. Moreover, we get the complement to* 255 *of each component because many sequences have not so many details and this manipulation allows us to obtain cleaner and more expressive graphs.*

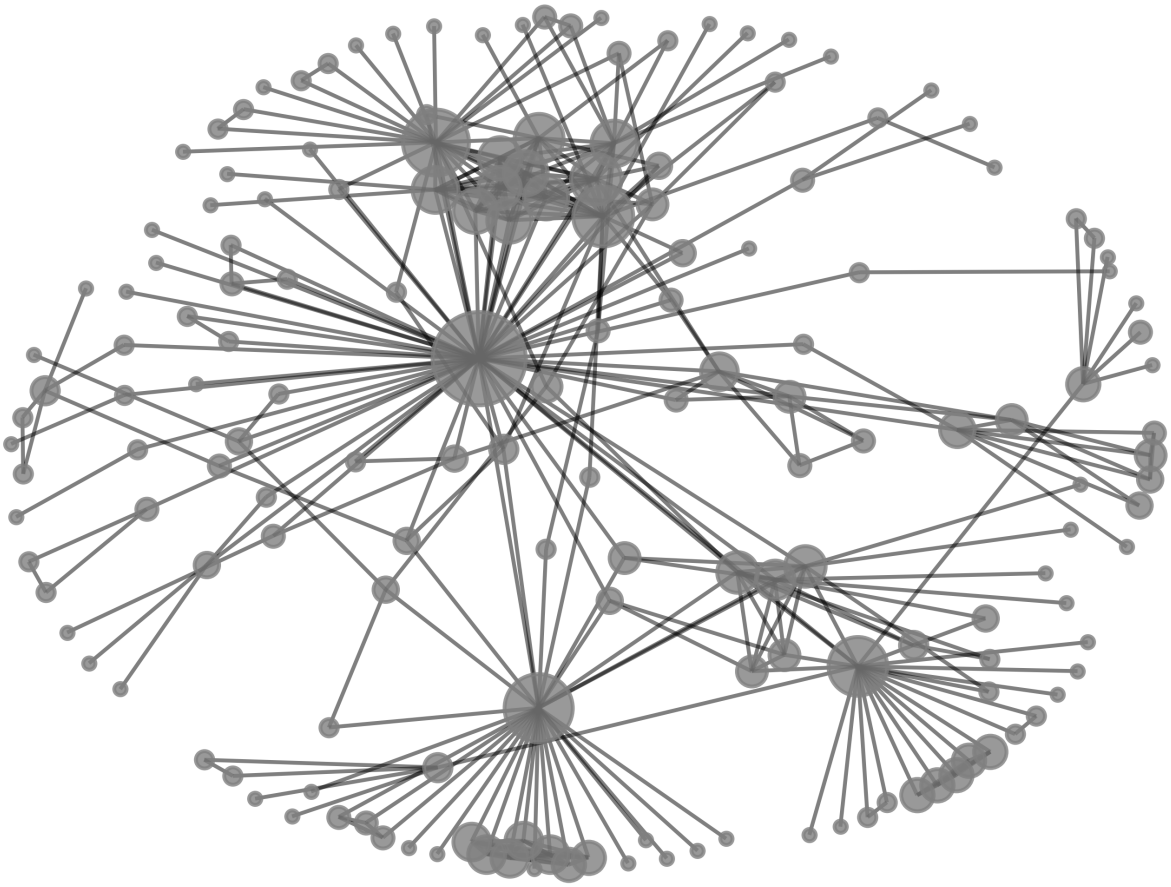


Figure 5: Sequences network where vertices are emphasized according to the number of incoming connections.

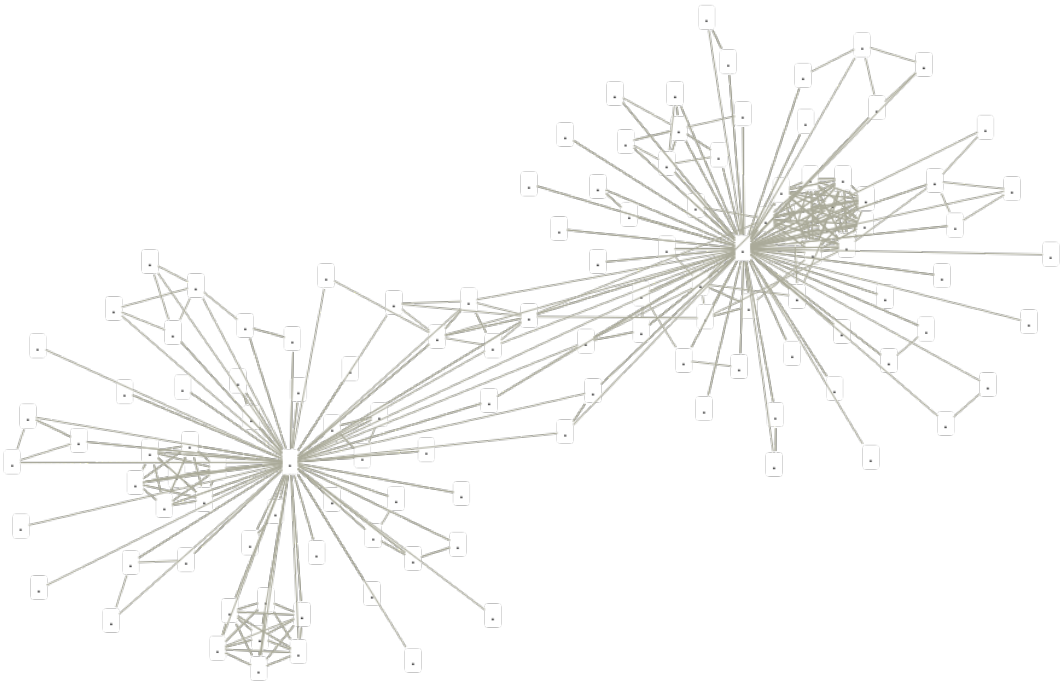


Figure 6: Sequences network abstracting over identifier to spot the underlying structure.

For the sake of clarity, the two sequences in evidence are the Fibonacci and Catalan numbers, the former has the color $(006100)_{16}$ *and its complement* $(FFFFFF)_{16} - (006100)_{16} = (FF9EFF)_{16}$ *means that it has many comments, formulae and connections; the latter has the color* $(7C00E5)_{16}$ *and its complement* $(FFFFFF)_{16} - (7C00E5)_{16} = (83FF1A)_{16}$ *means that it has lots of comments, links and references.*

Remark 9. *Recall that the interpretations given in the previous examples concern a subset of the OEIS only, in particular the one fetched in our session; finally, the more we crawl, the more graphs are effective and accurate.*

Example 10. *Finally, crawling for a while to get more sequences, we represent their connections in Figure 8, arranging them using a circular layout and we emphasize vertices in the dominating set using the red color.*

Conclusions

This paper presents a suite of tools that interacts with the *Online Encyclopedia of Integer Sequences*, whose primary goal is to automate simple and repetitive operations such as (i) crawling sequences to hold a local copy stored in JSON files, (ii) pretty printing data with filtering capabilities, both in the terminal and in Jupyter (`http://jupyter.org/`) notebooks and (iii) to visualize connections among sequences using graphs.

In parallel, this suite has been though to be open to extension and to interface with the hosting environment, UNIX in particular. For instance, the printer can be used in pipe with the `less` command to gain scroll and search features for free or the grapher can be augmented to generate more detailed graph descriptions to be processed by visualization tools.

An additional work direction is to make graphs interactive, namely to tie together the crawler and the grapher in a web-browser interface such that a click on a vertex triggers the execution of the

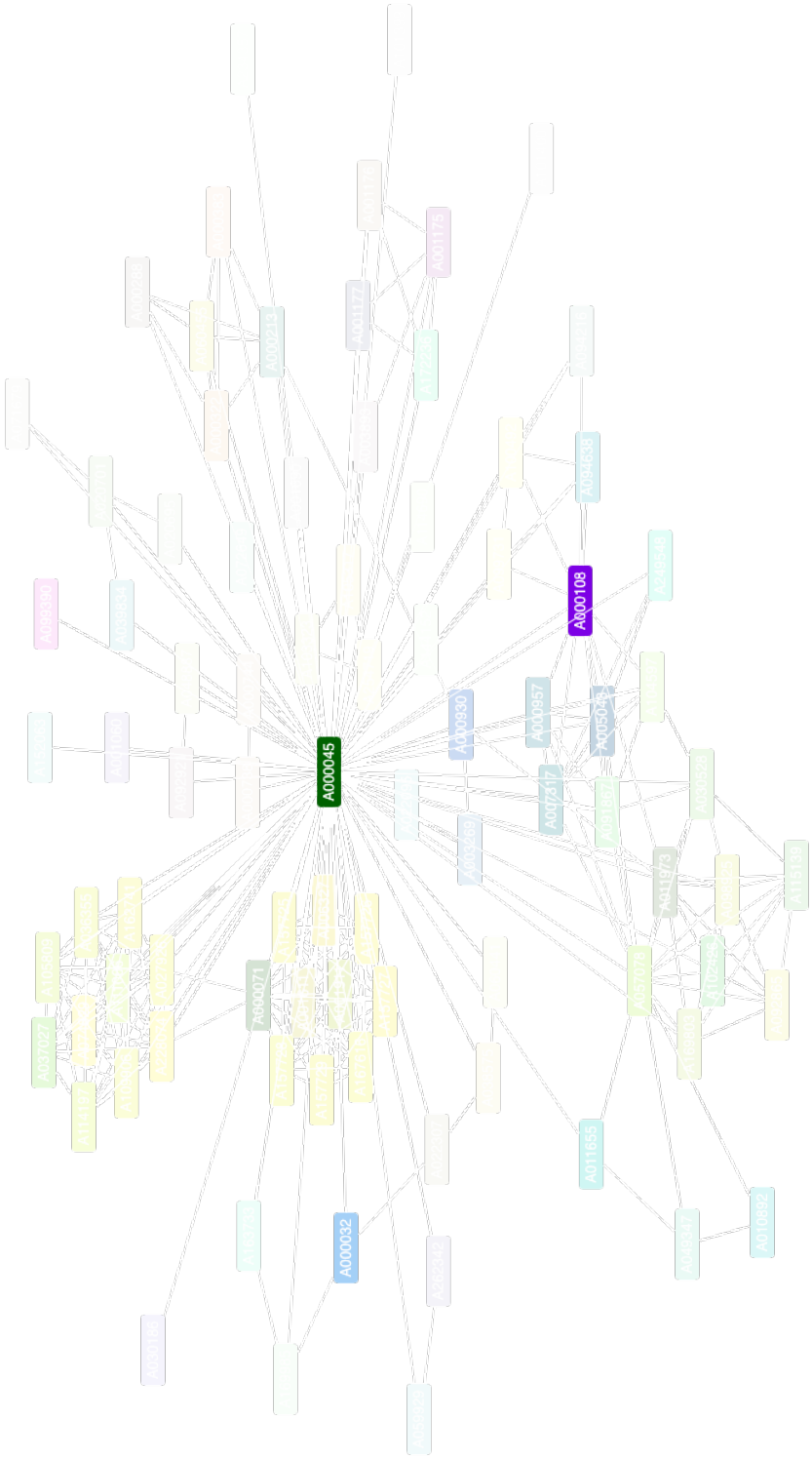


Figure 7: Sequences network with labelel vertices, here we see that the sequence of *Fibonacci numbers* (`https://oeis.org/A000045`) and of *Catalan numbers* (`https://oeis.org/A000108`) are the two central sequences, respectively.

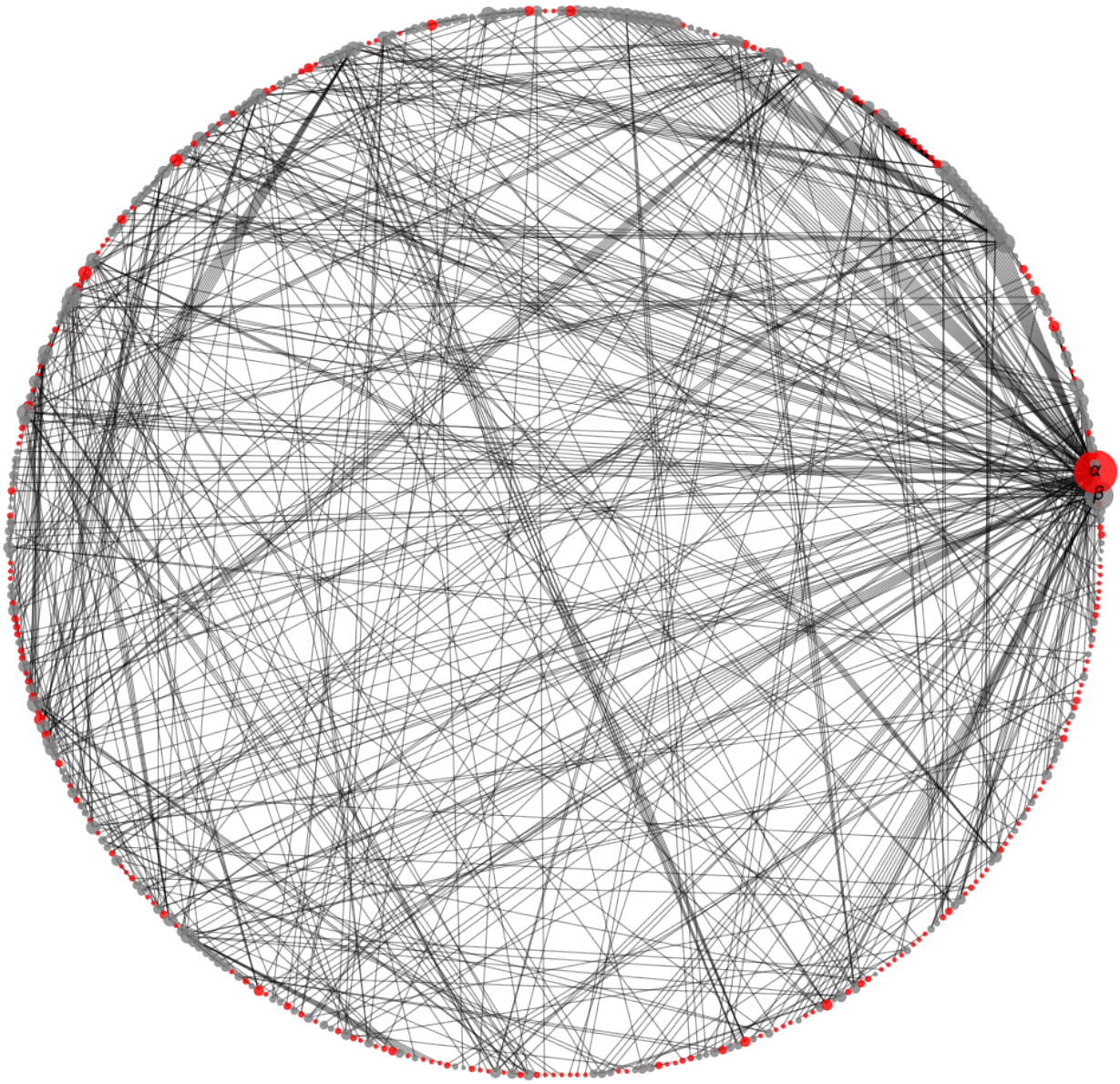


Figure 8: A bigger sequences network composed of 419 sequences; here the sequence of *Fibonacci numbers* is denoted by α and the sequence of *Catalan numbers* is denoted by β, respectively.

fetching process (unless it has been downloaded already) and the new connections are added to the network dynamically.

We wish to point out that the suite of tools presented in this work, the *grapher* in particular, could be used to mine the graph structures for study regularities and patterns among sequences which looks an interesting research activity.

References

R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete mathematics*. Addison-Wesley, Reading, MA, 1989.

H. D. Nguyen and D. Taggart. Mining the Online Encyclopedia of Integer Sequences, 2013. Preprint.

M. Nocentini. *OEIS Tools*. Open School on Combinatorial Method in the analysis of Algorithms and Data Structures, SKKU University, Korea, 2017. URL `http://massimo-nocentini.github.io/PhD/skku-aorc-2017/oeistools.html`.

N. J. A. Sloane. The Encyclopedia of Integer Sequences. URL `http://oeis.org/`.

R. Stanley. *Catalan Numbers*. Cambridge University Press, 1st edition, 2015.

G. van Rossum and J.J. Davis. A Web Crawler With asyncio Coroutines. URL `http://www.aosabook.org/en/500L/a-web-crawler-with-asyncio-coroutines.html`.

P. E. Weidmann. Sequencer. URL `https://github.com/p-e-w/sequencer`.