09-DaskBag

August 11, 2020

1 Dask bag

Dask proposes ”big data” collections with a small set of high-level primitives like `map`, `filter`, `groupby`, and `join`. With these common patterns we can often handle computations that are more complex than map, but are still structured.

- Dask-bag excels in processing data that can be represented as a sequence of arbitrary inputs (”messy” data)
- When you encounter a set of data with a format that does not enforce strict structure and datatypes.

Related Documentation

- Bag Documenation
- Bag API

[1]:
```
data = list(range(1,9))
data
```

[1]:
```
[1, 2, 3, 4, 5, 6, 7, 8]
```

[2]:
```
import dask.bag as db

b = db.from_sequence(data)
```

[3]:
```
b.compute()  # Gather results back to local process
```

[3]:
```
[1, 2, 3, 4, 5, 6, 7, 8]
```

[4]:
```
b.map(lambda x : x//2).compute() # compute length of each element and collect␣
↪results
```

[4]:
```
[0, 1, 1, 2, 2, 3, 3, 4]
```

[5]:
```
from time import sleep

def slow_half( x):
    sleep(1)
    return x // 2
```

```
res = b.map(slow_half)
res
```

[5]:
```
dask.bag<slow_half, npartitions=8>
```

[6]:
```
%%time
res.compute()
```

```
CPU times: user 7.46 ms, sys: 13.2 ms, total: 20.7 ms
Wall time: 4.38 s
```

[6]:
```
[0, 1, 1, 2, 2, 3, 3, 4]
```

[7]:
```
res.visualize()
```

```
      ␣
↪---------------------------------------------------------------------------

       FileNotFoundError                         Traceback (most recent call␣
↪last)

       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
       165     try:
   --> 166         proc = subprocess.Popen(cmd, startupinfo=get_startupinfo(),␣
↪**kwargs)
       167     except OSError as e:


       /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪__init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn,␣
↪close_fds, shell, cwd, env, universal_newlines, startupinfo, creationflags,␣
↪restore_signals, start_new_session, pass_fds, encoding, errors, text)
       853
   --> 854             self._execute_child(args, executable, preexec_fn,␣
↪close_fds,
       855                                 pass_fds, cwd, env,


       /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪_execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd,␣
↪env, startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite,␣
↪errread, errwrite, restore_signals, start_new_session)
      1701                         err_msg = os.strerror(errno_num)
```

```
   -> 1702                     raise child_exception_type(errno_num, err_msg,␣
↪err_filename)
      1703                 raise child_exception_type(err_msg)


       FileNotFoundError: [Errno 2] No such file or directory: 'dot'


   During handling of the above exception, another exception occurred:


       ExecutableNotFound                          Traceback (most recent call␣
↪last)

       <ipython-input-7-2dde28612c67> in <module>
   ----> 1 res.visualize()


       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(self, filename, format, optimize_graph, **kwargs)
        91         https://docs.dask.org/en/latest/optimize.html
        92         """
   ---> 93         return visualize(
        94             self,
        95             filename=filename,


       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(*args, **kwargs)
       551         raise NotImplementedError("Unknown value color=%s" % color)
       552
   --> 553     return dot_graph(dsk, filename=filename, **kwargs)
       554
       555


       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in dot_graph(dsk, filename, format, **kwargs)
       270     """
       271     g = to_graphviz(dsk, **kwargs)
   --> 272     return graphviz_to_file(g, filename, format)
       273
       274


       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in graphviz_to_file(g, filename, format)
```

```
    282         format = "png"
    283
--> 284     data = g.pipe(format=format)
    285     if not data:
    286         raise RuntimeError(


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪files.py in pipe(self, format, renderer, formatter, quiet)
    134         data = text_type(self.source).encode(self._encoding)
    135
--> 136         out = backend.pipe(self._engine, format, data,
    137                            renderer=renderer, formatter=formatter,
    138                            quiet=quiet)


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in pipe(engine, format, data, renderer, formatter, quiet)
    244     """
    245     cmd, _ = command(engine, format, None, renderer, formatter)
--> 246     out, _ = run(cmd, input=data, capture_output=True, check=True,␣
↪quiet=quiet)
    247     return out
    248


/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
    167     except OSError as e:
    168         if e.errno == errno.ENOENT:
--> 169             raise ExecutableNotFound(cmd)
    170         else:
    171             raise


ExecutableNotFound: failed to execute ['dot', '-Tpng'], make sure the␣
↪Graphviz executables are on your systems' PATH
```

[8]:
```
b.topk
```

[8]:
```
<bound method Bag.topk of dask.bag<from_sequence, npartitions=8>>
```

[9]:
```
b.product(b).compute() # Cartesian product of each pair
# of elements in two sequences (or the same sequence in this case)
```

```
[9]: [(1, 1),
      (1, 2),
      (1, 3),
      (1, 4),
      (1, 5),
      (1, 6),
      (1, 7),
      (1, 8),
      (2, 1),
      (2, 2),
      (2, 3),
      (2, 4),
      (2, 5),
      (2, 6),
      (2, 7),
      (2, 8),
      (3, 1),
      (3, 2),
      (3, 3),
      (3, 4),
      (3, 5),
      (3, 6),
      (3, 7),
      (3, 8),
      (4, 1),
      (4, 2),
      (4, 3),
      (4, 4),
      (4, 5),
      (4, 6),
      (4, 7),
      (4, 8),
      (5, 1),
      (5, 2),
      (5, 3),
      (5, 4),
      (5, 5),
      (5, 6),
      (5, 7),
      (5, 8),
      (6, 1),
      (6, 2),
      (6, 3),
      (6, 4),
      (6, 5),
      (6, 6),
      (6, 7),
```

```
 (6, 8),
 (7, 1),
 (7, 2),
 (7, 3),
 (7, 4),
 (7, 5),
 (7, 6),
 (7, 7),
 (7, 8),
 (8, 1),
 (8, 2),
 (8, 3),
 (8, 4),
 (8, 5),
 (8, 6),
 (8, 7),
 (8, 8)]
```

Chain operations to construct more complex computations

[10]:
```
(b.filter(lambda x: x % 2 > 0)
  .product(b)
  .filter( lambda v : v[0] % v[1] == 0 and v[0] != v[1])
  .compute())
```

[10]:
```
[(3, 1), (5, 1), (7, 1)]
```

[ ]:
```

```

1.1 Daily stock example

Let's use the bag interface to read the json files containing time series.

Each line is a JSON encoded dictionary with the following keys - timestamp: Day. - close: Stock value at the end of the day. - high: Highest value. - low: Lowest value. - open: Opening price.

[11]:
```
# preparing data
import os  # library to get directory and file paths
import tarfile # this module makes possible to read and write tar archives

def extract_data(name, where):
    datadir = os.path.join(where,name)
    if not os.path.exists(datadir):
        print("Extracting data...")
        tar_path = os.path.join(where, name+'.tgz')
        with tarfile.open(tar_path, mode='r:gz') as data:
            data.extractall(where)
```

```
extract_data('daily-stock','data') # this function call will extract json files
```

```
Extracting data…
```

[12]:
```
%ls data/daily-stock/*.json
```

```
data/daily-stock/aet.json   data/daily-stock/hpq.json
data/daily-stock/afl.json   data/daily-stock/ibm.json
data/daily-stock/aig.json   data/daily-stock/jbl.json
data/daily-stock/al.json    data/daily-stock/jpm.json
data/daily-stock/amgn.json  data/daily-stock/luv.json
data/daily-stock/avy.json   data/daily-stock/met.json
data/daily-stock/b.json     data/daily-stock/pcg.json
data/daily-stock/bwa.json   data/daily-stock/tgt.json
data/daily-stock/ge.json    data/daily-stock/usb.json
data/daily-stock/hal.json   data/daily-stock/xom.json
data/daily-stock/hp.json
```

[13]:
```
import dask.bag as db
import json
stocks = db.read_text('data/daily-stock/*.json')
```

[14]:
```
stocks.npartitions
```

[14]:
```
22
```

[15]:
```
stocks.visualize()
```

```
␣
↪---------------------------------------------------------------------------

FileNotFoundError                         Traceback (most recent call␣
↪last)

/usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
    165     try:
--> 166         proc = subprocess.Popen(cmd, startupinfo=get_startupinfo(),␣
↪**kwargs)
    167     except OSError as e:

/usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪__init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn,␣
↪close_fds, shell, cwd, env, universal_newlines, startupinfo, creationflags,␣
↪restore_signals, start_new_session, pass_fds, encoding, errors, text)
```

```
      853
  --> 854             self._execute_child(args, executable, preexec_fn,␣
↪close_fds,
      855                                 pass_fds, cwd, env,


      /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪_execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd,␣
↪env, startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite,␣
↪errread, errwrite, restore_signals, start_new_session)
     1701                         err_msg = os.strerror(errno_num)
  -> 1702                     raise child_exception_type(errno_num, err_msg,␣
↪err_filename)
     1703                 raise child_exception_type(err_msg)


      FileNotFoundError: [Errno 2] No such file or directory: 'dot'


  During handling of the above exception, another exception occurred:


      ExecutableNotFound                          Traceback (most recent call␣
↪last)

      <ipython-input-15-a27501b2e8e0> in <module>
  ----> 1 stocks.visualize()


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(self, filename, format, optimize_graph, **kwargs)
       91         https://docs.dask.org/en/latest/optimize.html
       92         """
  ---> 93         return visualize(
       94             self,
       95             filename=filename,


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(*args, **kwargs)
      551         raise NotImplementedError("Unknown value color=%s" % color)
      552
  --> 553     return dot_graph(dsk, filename=filename, **kwargs)
      554
      555
```

```
      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in dot_graph(dsk, filename, format, **kwargs)
      270     """
      271     g = to_graphviz(dsk, **kwargs)
  --> 272     return graphviz_to_file(g, filename, format)
      273
      274


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in graphviz_to_file(g, filename, format)
      282         format = "png"
      283
  --> 284     data = g.pipe(format=format)
      285     if not data:
      286         raise RuntimeError(


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪files.py in pipe(self, format, renderer, formatter, quiet)
      134         data = text_type(self.source).encode(self._encoding)
      135
  --> 136         out = backend.pipe(self._engine, format, data,
      137                            renderer=renderer, formatter=formatter,
      138                            quiet=quiet)


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in pipe(engine, format, data, renderer, formatter, quiet)
      244     """
      245     cmd, _ = command(engine, format, None, renderer, formatter)
  --> 246     out, _ = run(cmd, input=data, capture_output=True, check=True,␣
↪quiet=quiet)
      247     return out
      248


      /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
      167     except OSError as e:
      168         if e.errno == errno.ENOENT:
  --> 169             raise ExecutableNotFound(cmd)
      170         else:
      171             raise
```

```
        ExecutableNotFound: failed to execute ['dot', '-Tpng'], make sure the␣
↪Graphviz executables are on your systems' PATH
```

[16]:
```
import json
js = stocks.map(json.loads)
```

[17]:
```
import os, sys
from glob import glob
import pandas as pd
import json

here = os.getcwd() # get the current directory
filenames = sorted(glob(os.path.join(here,'data', 'daily-stock', '*.json')))
filenames
```

[17]:
```
['/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb.json',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom.json']
```

[18]:
```
%rm data/daily-stock/*.h5
```

```
rm: cannot remove 'data/daily-stock/*.h5': No such file or directory
```

[19]:
```
from tqdm.notebook import tqdm
for fn in tqdm(filenames):
    with open(fn) as f:
        data = [json.loads(line) for line in f]
```

```
    df = pd.DataFrame(data)

    out_filename = fn[:-5] + '.h5'
    df.to_hdf(out_filename, '/data')
```

```
HBox(children=(FloatProgress(value=0.0, max=21.0), HTML(value='')))
```

[20]:
```
filenames = sorted(glob(os.path.join(here,'data', 'daily-stock', '*.h5')))
filenames
```

[20]:
```
['/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/afl.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aig.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/al.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/amgn.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/avy.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/b.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/bwa.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ge.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hal.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hp.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/hpq.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/ibm.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jbl.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/jpm.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/met.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/pcg.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/tgt.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/usb.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/xom.h5']
```

1.1.1 Serial version

[21]:
```
%%time
series = {}
for fn in filenames:   # Simple map over filenames
    series[fn] = pd.read_hdf(fn)['close']

results = {}

for a in filenames:    # Doubly nested loop over the same collection
    for b in filenames:
        if a != b:     # Filter out bad elements
```

```
            results[a, b] = series[a].corr(series[b])  # Apply function

((a, b), corr) = max(results.items(), key=lambda kv: kv[1])  # Reduction
```

```
CPU times: user 1.04 s, sys: 85.4 ms, total: 1.12 s
Wall time: 1.13 s
```

[22]:
```
a, b, corr
```

[22]:
```
('/home/runner/work/big-data/big-data/notebooks/data/daily-stock/aet.h5',
 '/home/runner/work/big-data/big-data/notebooks/data/daily-stock/luv.h5',
 0.9413176064560879)
```

1.2 Dask.bag methods

We can construct most of the above computation with the following dask.bag methods:

- `collection.map(function)`: apply function to each element in collection
- `collection.product(collection)`: Create new collection with every pair of inputs
- `collection.filter(predicate)`: Keep only elements of colleciton that match the predicate function
- `collection.max()`: Compute maximum element

[23]:
```
%%time

import dask.bag as db

b = db.from_sequence(filenames)
series = b.map(lambda fn: pd.read_hdf(fn)['close'])

corr = (series.product(series)
              .filter(lambda ab: not (ab[0] == ab[1]).all())
              .map(lambda ab: ab[0].corr(ab[1])).max())
```

```
CPU times: user 2.06 ms, sys: 4.52 ms, total: 6.58 ms
Wall time: 6.36 ms
```

[24]:
```
%%time

result = corr.compute()
```

```
CPU times: user 1.86 s, sys: 650 ms, total: 2.51 s
Wall time: 4.47 s
```

[25]:
```
result
```

[25]:
```
0.9413176064560879
```

1.2.1 Wordcount with Dask bag

[26]:
```
import lorem

lorem.text()
```

[26]:
```
'Dolor numquam consectetur dolore labore dolore. Eius est velit labore adipisci
velit non porro. Amet porro dolorem dolor dolor. Amet non dolor modi quaerat
numquam sed adipisci. Quaerat eius aliquam dolorem eius dolore numquam. Non
labore aliquam ipsum. Tempora quaerat eius sit quisquam dolor. Quisquam dolor
aliquam velit dolor non tempora dolore.\n\nUt etincidunt sed numquam. Neque
velit voluptatem quaerat magnam. Neque non dolor est. Porro aliquam quiquia est
amet aliquam quiquia. Tempora ipsum quisquam ipsum numquam non ut. Aliquam eius
adipisci etincidunt eius. Eius est numquam dolorem. Magnam neque eius est
numquam sit.\n\nTempora ipsum velit quiquia. Sit aliquam non porro dolor ipsum
dolor dolore. Quisquam consectetur amet ut. Neque tempora amet est porro est
consectetur. Etincidunt numquam dolore quaerat velit. Sed neque etincidunt
labore quiquia dolore numquam eius. Dolore eius ipsum adipisci. Numquam modi ut
adipisci est non tempora. Consectetur eius sed tempora est quisquam. Adipisci
porro aliquam voluptatem.\n\nQuiquia est dolorem sit eius. Labore sit ipsum
modi. Adipisci est adipisci non sed. Ipsum quiquia sit dolor modi magnam. Dolor
tempora modi modi dolorem porro. Voluptatem quisquam numquam magnam. Eius porro
ut eius sit non. Dolore amet aliquam sit adipisci neque. Eius voluptatem dolorem
voluptatem dolor.\n\nDolor aliquam etincidunt dolore ut. Quaerat quisquam neque
dolore. Ipsum quisquam tempora porro non dolor. Consectetur aliquam modi
consectetur labore dolor eius sit. Dolorem consectetur velit quisquam dolore
adipisci dolorem amet. Magnam etincidunt consectetur modi. Est consectetur
magnam etincidunt. Etincidunt eius magnam magnam est dolor magnam dolor. Dolor
aliquam consectetur porro tempora ipsum dolore. Quiquia est eius ipsum adipisci
aliquam dolore.'
```

[27]:
```
import lorem

for i in range(20):
    with open(f"sample{i:02d}.txt","w") as f:
        f.write(lorem.text())
```

[28]:
```
%ls *.txt
```

```
sample00.txt  sample04.txt  sample08.txt  sample12.txt  sample16.txt
sample01.txt  sample05.txt  sample09.txt  sample13.txt  sample17.txt
sample02.txt  sample06.txt  sample10.txt  sample14.txt  sample18.txt
sample03.txt  sample07.txt  sample11.txt  sample15.txt  sample19.txt
```

[29]:
```
import glob
glob.glob('sample*.txt')
```

[29]:
```
['sample13.txt',
 'sample01.txt',
 'sample14.txt',
 'sample00.txt',
 'sample17.txt',
 'sample19.txt',
 'sample07.txt',
 'sample09.txt',
 'sample08.txt',
 'sample16.txt',
 'sample02.txt',
 'sample04.txt',
 'sample11.txt',
 'sample10.txt',
 'sample05.txt',
 'sample06.txt',
 'sample18.txt',
 'sample03.txt',
 'sample15.txt',
 'sample12.txt']
```

[30]:
```
import dask.bag as db
import glob
b = db.read_text(glob.glob('sample*.txt'))

wordcount = (b.str.replace(".","")  # remove dots
             .str.lower()           # lower text
             .str.strip()           # remove \n and trailing spaces
             .str.split()           # split into words
             .flatten()             # chain all words lists
             .frequencies()         # compute occurences
             .topk(10, lambda x: x[1])) # sort and return top 10 words


wordcount.compute() # Run all tasks and return result
```

[30]:
```
[('modi', 193),
 ('neque', 185),
 ('magnam', 185),
 ('voluptatem', 183),
 ('quaerat', 182),
 ('quiquia', 180),
 ('dolor', 179),
 ('sed', 178),
 ('ipsum', 174),
 ('velit', 174)]
```

1.3 Genome example

We will use a Dask bag to calculate the frequencies of sequences of five bases, and then sort the sequences into descending order ranked by their frequency.

- First we will define some functions to split the bases into sequences of a certain size

1.3.1 Exercise 9.1

- Implement a function `group_characters(line, n=5)` to group `n` characters together and return a iterator. `line` is a text line in genome.txt file.

```
>>> line = "abcdefghijklmno"
>>> for seq in group_character(line, 5):
        print(seq)
```

```
"abcde"
"efghi"
"klmno"
```

- Implement `group_and_split(line)`

  ```
  >>> group_and_split('abcdefghijklmno')
  ['abcde', 'fghij', 'klmno']
  ```

- Use the dask bag to compute the frequencies of sequences of five bases.

[31]:
```
from string import ascii_lowercase as alphabet
alphabet
```

[31]:
```
'abcdefghijklmnopqrstuvwxyz'
```

[32]:
```
def reverse(text):
    k = len(text)
    while k > 0:
        k = k-1
        yield text[k]

reverse_alphabet = reverse(alphabet)
print(*reverse_alphabet)
```

```
z y x w v u t s r q p o n m l k j i h g f e d c b a
```

[33]:
```
class Reverse:

    def __init__(self, data):
        self.data = data
        self.index = len(data)

    def __iter__(self):
```

```
        return self

    def __next__(self):
        self.index = self.index-1
        if self.index < 0:
            raise StopIteration
        else:
            return self.data[self.index]
```

[34]:
```
class Fibonacci:

    def __init__(self, n):
        self.n = n
        self.f0 = 0
        self.f1 = 1

    def __iter__(self):
        return self

    def __next__(self):
        self.n = self.n - 1
        if self.n < 0:
            raise StopIteration
        else:
            self.f0, self.f1 = self.f1, self.f0 + self.f1
        return self.f1

print(*Fibonacci(7))
```

```
1 2 3 5 8 13 21
```

[35]:
```
for c in Reverse(alphabet):
    print(c, end="")
```

```
zyxwvutsrqponmlkjihgfedcba
```

[36]:
```

for c in reverse(alphabet):
    print(c, end="")
```

```
zyxwvutsrqponmlkjihgfedcba
```

[37]:
```
def group_character( line, n=5):
    bases = ''
    for i, b in enumerate(line):
        bases += b
        if (i+1) % n == 0:
```

```
            yield bases
            bases = ''
```

[38]:
```
line = "abcdefghijklmno"
for seq in group_character(line, 5):
    print(seq)
```

```
abcde
fghij
klmno
```

[39]:
```
def group_and_split( line, n):
    return [seq for seq in group_character(line,n)]
```

[40]:
```
%ls data
```

```
daily-stock/     genome04.txt
monthly.land.90S.90N.df_1901-2000mean.dat.txt
daily-stock.tgz  genome05.txt    nucleotide-sample.txt
genome.txt       genome06.txt    nycflights.tar.gz
genome00.txt     genome07.txt    people.json
genome01.txt     irmar.csv       philadelphia-crime-data-2015-ytd.csv
genome02.txt     irmar.json
genome03.txt     latinbooks.tgz
```

[41]:
```
import os
from glob import glob

data_path = os.path.join("data")
with open(os.path.join(data_path,"genome.txt")) as g:
    data = g.read()
    for i in range(8):
        file = os.path.join(data_path,f"genome{i:02d}.txt")
        with open(file,"w") as f:
            f.write(data)

glob("data/genome0*.txt")
```

[41]:
```
['data/genome00.txt',
 'data/genome01.txt',
 'data/genome07.txt',
 'data/genome06.txt',
 'data/genome02.txt',
 'data/genome05.txt',
 'data/genome03.txt',
 'data/genome04.txt']
```

[42]:
```
from operator import itemgetter

import dask.bag as db

b = db.read_text("data/genome0*.txt")

result = (b.filter(lambda line: not line.startswith(">"))
  .map(lambda line: line.strip())
  .map(lambda line : group_and_split(line,5))
  .flatten()
  .frequencies()
  .topk(10,lambda v : v[1]))
```

[43]:
```
result.visualize()
```

```
      ␣
↪---------------------------------------------------------------------------

       FileNotFoundError                         Traceback (most recent call␣
↪last)

       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
       165     try:
   --> 166         proc = subprocess.Popen(cmd, startupinfo=get_startupinfo(),␣
↪**kwargs)
       167     except OSError as e:


       /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪__init__(self, args, bufsize, executable, stdin, stdout, stderr, preexec_fn,␣
↪close_fds, shell, cwd, env, universal_newlines, startupinfo, creationflags,␣
↪restore_signals, start_new_session, pass_fds, encoding, errors, text)
       853
   --> 854             self._execute_child(args, executable, preexec_fn,␣
↪close_fds,
       855                                 pass_fds, cwd, env,


       /usr/share/miniconda3/envs/big-data/lib/python3.8/subprocess.py in␣
↪_execute_child(self, args, executable, preexec_fn, close_fds, pass_fds, cwd,␣
↪env, startupinfo, creationflags, shell, p2cread, p2cwrite, c2pread, c2pwrite,␣
↪errread, errwrite, restore_signals, start_new_session)
      1701                         err_msg = os.strerror(errno_num)
   -> 1702                     raise child_exception_type(errno_num, err_msg,␣
↪err_filename)
```

```
   1703                 raise child_exception_type(err_msg)


    FileNotFoundError: [Errno 2] No such file or directory: 'dot'


During handling of the above exception, another exception occurred:


    ExecutableNotFound                        Traceback (most recent call␣
↪last)

    <ipython-input-43-dc769738af30> in <module>
----> 1 result.visualize()


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(self, filename, format, optimize_graph, **kwargs)
     91         https://docs.dask.org/en/latest/optimize.html
     92         """
---> 93         return visualize(
     94             self,
     95             filename=filename,


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/
↪base.py in visualize(*args, **kwargs)
    551         raise NotImplementedError("Unknown value color=%s" % color)
    552
--> 553     return dot_graph(dsk, filename=filename, **kwargs)
    554
    555


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in dot_graph(dsk, filename, format, **kwargs)
    270     """
    271     g = to_graphviz(dsk, **kwargs)
--> 272     return graphviz_to_file(g, filename, format)
    273
    274


    /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/dask/dot.
↪py in graphviz_to_file(g, filename, format)
    282         format = "png"
    283
```

```
   --> 284     data = g.pipe(format=format)
       285     if not data:
       286         raise RuntimeError(


       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪files.py in pipe(self, format, renderer, formatter, quiet)
       134         data = text_type(self.source).encode(self._encoding)
       135
   --> 136         out = backend.pipe(self._engine, format, data,
       137                            renderer=renderer, formatter=formatter,
       138                            quiet=quiet)


       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in pipe(engine, format, data, renderer, formatter, quiet)
       244     """
       245     cmd, _ = command(engine, format, None, renderer, formatter)
   --> 246     out, _ = run(cmd, input=data, capture_output=True, check=True,␣
↪quiet=quiet)
       247     return out
       248


       /usr/share/miniconda3/envs/big-data/lib/python3.8/site-packages/graphviz/
↪backend.py in run(cmd, input, capture_output, check, encoding, quiet, **kwargs)
       167     except OSError as e:
       168         if e.errno == errno.ENOENT:
   --> 169             raise ExecutableNotFound(cmd)
       170         else:
       171             raise


       ExecutableNotFound: failed to execute ['dot', '-Tpng'], make sure the␣
↪Graphviz executables are on your systems' PATH
```

[44]:
```
result.compute()
```

[44]:
```
[('CTGTG', 472),
 ('CCCAG', 440),
 ('CCTGG', 416),
 ('AAAAA', 392),
 ('TGCTG', 336),
 ('TGTGT', 328),
 ('CCACC', 312),
 ('GGCTG', 304),
```

```
('CACCA', 296),
('GGTGG', 296)]
```

1.3.2 Exercise 9.2

The FASTA file format is used to write several genome sequences.

- Create a function that can read a FASTA file and compute the frequencies for n = 5 of a given sequence.

1.3.3 Exercise 9.3

Write a program that uses the function implemented above to read several FASTA files stored in a Dask bag.

2 Some remarks about bag

- Higher level dask collections include functions for common patterns
- Move data to collection, construct lazy computation, trigger at the end
- Use Dask.bag (`product + map`) to handle nested for loop

Bags have the following known limitations

1. Bag operations tend to be slower than array/dataframe computations in the same way that Python tends to be slower than NumPy/Pandas
2. `Bag.groupby` is slow. You should try to use `Bag.foldby` if possible.
3. Check the API
4. `dask.dataframe` can be faster than `dask.bag`. But sometimes it is easier to load and clean messy data with a bag. We will see later how to transform a bag into a `dask.dataframe` with the to_dataframe method.