MultiDirMap

Release 0.2.0

Oct 02, 2019

Contents

CHAPTER 1

multidirmap

Multidirectional map where an arbitrary number of columns can be used as keys.

1.1 Status

1.2 Installation

```
$ pip install multidirmap
```

1.3 Documentation

https://multidirmap.readthedocs.io/en/latest/

1.4 Quick Start

```
>>> from multidirmap import MultiDirMap
>>> crew = MultiDirMap(
        ["character", "portrayed_by", "role", "nicknames"],
        key_columns=2,
        data=[["Malcolm Reynolds", "Nathan Fillion", "Captain", ["Mal", "Captain␣
↪Tight Pants"]],
              ["Zoë Washburne", "Gina Torres", "First Mate"],
              ["Hoban Washburne", "Alan Tudyk", "Pilot", "Wash"]])
>>> crew["Malcolm Reynolds"].role
Captain
>>> crew.portrayed_by["Nathan Fillion"].nicknames
["Mal", "Captain Tight Pants"]
```

1.5 Features

- As many columns as desired can be used as key columns for the mapping
- O(1) retrieval from any key column
- Internal consistency is maintained through any modifications to the contents
- Insertion order is maintained in the primary key column
- Built-in pretty printing of the mapping

1.6 Use Cases

Dictionaries are ubiquitous in Pyton and provide an extremely useful and fast mapping from keys to values. Sometimes, a single, uni-directional mapping is not enough, though, and while bidict extends this functionality to a bidirectional mapping, *multidirmap* provides an array-like datastructure where any number of columns can be used for O(1) retrieval. In its simplest implementation (2 columns, one of which is a key column), it essentially provides the same functionality as a dict, albeit with additional overhead (don't do that...). 2 columns that are both key columns will behave somewhat like a bidict, albeit with slightly different syntax. But *multidirmap* is significantly more flexible in that any number of key and non-key columns can be used. A somewhat similar effect could be achieved with pandas DataFrames, though these (1) will not ensure uniqueness of keys, hence a retrieval may return any number of rows, (2) use an array structure, hence retrieval is O(n) which for large arrays can get *very* slow, and (3) require the installation of pandas, which is a rather large library to include just for this feature.

Say we want to work with information from the Periodic Table of Elements, like

```
[["H", "Hydrogen", 1, [1, 2, 3]],
 ["He", "Helium", 2, [4, 3]],
 ["Li", "Lithium", 3, [7, 6]],
 ...
 ["Og", "Oganesson", 118, [295, 294]]]
```

where the columns indicate symbol, name, atomic number, and nucleon numbers of isotopes respectively. The first three columns are obvious candidates for key columns as they are by definition unique. *multidirmap* allows placing this information in a unified datastructure where questions like "What are the isotope nucleon numbers of Lithium?", "What is the chemical element symbol of Potassium?", or "What is the name of the element with atomic number 46?"

can be asked with a simple syntax and O(1) retrieval. Any number of additional key and non-key columns could be added.

The use case that prompted the development on this package involved the *struct* module: For a binary interface I needed to convert back and forth between (1) a string representation of the variable type, (2) an integer representation of the variable type, (3) the struct format char, and (4) the size in bytes of the variable. Again, 1-3 are obvious candidates for key columns, with 4 being a non-key column. Without *multidirmap*, several separate dicts have to be used to provide each needed mapping from one column to another and there is easy way to ensure that these dicts remain consistent with each other through possible changes.

CHAPTER 2

Usage

2.1 Creating / Updating a MultiDirMap

```
>>> from multidirmap import MultiDirMap
>>> crew = MultiDirMap(
        ["character", "portrayed_by", "role", "nicknames"],
        key_columns=2,
        data=[["Malcolm Reynolds", "Nathan Fillion", "Captain", ["Mal", "Captain␣
↪Tight Pants"]],
              ["Zoë Washburne", "Gina Torres", "First Mate"],
              ["Hoban Washburne", "Alan Tudyk", "Pilot", "Wash"]])
```

MultiDirMap takes three arguments:

- "columns" is required and is a list of names for the columns in the map. The first column must be a key column and is below referred to as the "primary key column"
- "key_columns" gives the number of columns (in the order in which they are given in "columns" that should be key columns (and will only accept unique and hashable values). It is an optional argument and defaults to len(columns)
- "data" gives the data with which to initialize the MultiDirMap and is optional

Note that

```
>>> my_map = MultiDirMap(columns, data=my_data)
```

is exactly equivalent to

```
>>> my_map = MultiDirMap(columns)
>>> my_map.update(my_data)
```

Data insertion order is maintained in the primary key column.

2.1.1 Accepted Data Formats

Data for the map can be provided in three different formats:

```
>>> crew.update([["Inara Serra", "Morena Baccarin", "Companion", "Ambassador"],
                 ["Jayne Cobb", "Adam Baldwin", "Mercenary", "Hero of Canton"]])
>>> crew.update([{"character": "Kaywinnet Lee Frye", "portrayed_by": "Jewel Staite",
                  "role": "Mechanic", "nicknames": "Kaylee"},
                 {"character": "Simon Tam", "portrayed_by": "Sean Maher",
                  "role": "Medic"}])
>>> crew.update({"River Tam": ["Summer Glau", None, "Méi-mei"],
                 "Derrial Book": ["Ron Glass", "Shepherd", "Preacher"]})
```

Values for non-key columns are optional.

All values in key columns must be hashable, so

```
>>> crew.update([["Yolanda", "Saffron", "Bridget"], "Christina Hendricks", "Grifter
↪"]])
Traceback (most recent call last):
...
TypeError: unhashable type: 'list'
```

2.1.2 Key Conflicts

In a normal Python dict, inserting an entry with a key that already exists overwrites the existing entry. In a multidirectional mapping, things are a little more complicated, so `MultiDirMap.update()` takes two additional keyword arguments, `"overwrite"` and `"skip_duplicates"`:

- "overwrite" (default: "primary" can take the values "none", "primary", "secondary", or "all" It indicates which key columns may be overwritten (with "secondary" meaning all key columns other than the primary one). An entry that has a value that is overwritten by an update will be completely removed from the MultiDirMap
- "skip_duplicates" (default False) describes the behaviour when an entry is encountered that may not be overwritten. If False, the update operation is aborted and a `DuplicateKeyError` is raised. **An aborted update will never leave the map in a modified state, this includes the order of the primary key column.** So if the 10th entry in an update operation encounters a conflict, the first 9 will not end up in the map either. If True, conflicting entries will simply be skipped and all non-conflicting entries are inserted.

```
>>> crew.update([["Yolanda", "Christina Hendricks", "Grifter"]])
>>> crew.update([["Bridget", "Christina Hendricks", "Grifter"]], overwrite="none")
Traceback (most recent call last):
...
DuplicateKeyError: One or more keys in ["Bridget", "Christina Hendricks", "Grifter"]␣
↪were duplicates
>>> crew.update([["Bridget", "Christina Hendricks", "Grifter"]], overwrite="primary")
>>> crew["Bridget"].portrayed_by
Christina Hendricks
>>> crew["Yolanda"]
Traceback (most recent call last):
...
KeyError: "Yolanda"
```

Note that an entry that overwrites another one, can "free up" keys in other columns for subsequent updates. This is not currently checked for within an update operation, so it is possible that two consecutive updates with `overwrite="primary"` or `overwrite="secondary` will succeed where a combined operation would raise a DuplicateKeyError.

2.2 Key Column Methods

Under the hood, all key columns are stored as dicts and support dict methods with one important caveat: **Key Columns in a MultiDirMap are read-only.** This means that any of the following will raise a `TypeError`:

```
>>> crew.portrayed_by["Nathan Fillion"] = [...]
>>> del crew.portrayed_by["Nathan Fillion"]
>>> crew.portrayed_by.clear()
>>> crew.portrayed_by.pop("Nathan Fillion")
>>> crew.portrayed_by.popitem()
>>> crew.portrayed_by.setdefault("Nathan Fillion", default=None)
>>> crew.portrayed_by.update(...)
```

On the other hand, all of the following methods will work as expected:

```
>>> crew.portrayed_by["Nathan Fillion"]
>>> for name in crew.portrayed_by: ...
>>> crew.portrayed_by.get("Nathan Fillion")
>>> crew.portrayed_by.keys()
>>> crew.portrayed_by.values()
>>> crew.portrayed_by.items()
```

Operating directly on the MultiDirMap is equivalent to operate on its primary key column, with the exception that writing access is permitted, so

```
>>> crew["Malcolm Reynolds"]
>>> crew.character["Malcolm Reynolds"]
```

are equivalent, but in the case of

```
>>> del crew["Malcolm Reynolds"]
>>> del crew.character["Malcolm Reynolds"]
```

the first one will work, while the second one will raise a `TypeError`.

Note that for modifying methods other than `update()`, behaviour will always correspond to overwriting of primary key columns being permitted and overwriting of secondary key columns being forbidden.

2.3 Row Elements

Accessing an entry in a key column returns a custom object called a `MultiDirMapRow`. This object contains all data of the row (including the key that was used to retrieve this element). So it is entirely possible (though of questionable utility) to write

```
>>> crew["Malcolm Reynolds"].character
Malcolm Reynolds
```

All attributes can be accessed with dot notation. Furthermore, a `MultiDirMapRow` has the methods `aslist()` and `asdict()`:

```
>>> crew["Malcolm Reynolds"].aslist()
["Malcolm Reynolds", "Nathan Fillion", "Captain", ["Mal", "Captain Tight Pants"]]
>>> crew["Malcolm Reynolds"].asdict()
{"character": "Malcolm Reynolds", "portrayed_by": "Nathan Fillion",
 "role": "Captain", "nicknames": ["Mal", "Captain Tight Pants"]}
```

Attributes can be modified and changes are propagated to the rest of the map (subject to not conflicting with existing secondary keys):

```
>>> mal = crew["Malcolm Reynolds"]
>>> mal.portrayed_by = "Alan Tudyk"
Traceback (most recent call last):
...
DuplicateKeyError: ...
>>> mal.nicknames = None
>>> crew.portrayed_by["Nathan Fillion"].aslist()
["Malcolm Reynolds", "Nathan Fillion", "Captain", None]
```

2.4 Equality Testing

Two MultiDirMaps will compare equal if their column names, number of key columns, and entries are identical. Order - while preserved in the primary key column regardless of Python version - does not affect equality testing.

2.5 Ordering

2.5.1 Reordering the Secondary Key Columns

While the primary key column always maintains insertion order, the order of the secondary key columns can be scrambled by insertions that remove existing elements by overwriting some of their keys. Consistent ordering between primary and secondary key columns can be restored by calling `reorder_secondary_keys()` on a map. Note that this can be slow on large maps as it will recreate all secondary dictionaries.

2.5.2 Sorting a MultiDirMap

An existing map can be sorted with an arbitrary comparison function:

```
>>> crew.sort(key=lambda entry: entry.portrayed_by, reverse=True)
>>> print(crew)
character*          portrayed_by*       role                nicknames
=============================================================================
River Tam           Summer Glau         None                Méi-mei
...
Hoban Washburne     Alan Tudyk          Pilot               Wash
```

- "key" is the function that serves as the key for the comparison function. If no key is given, sorting is done by the entries in the primary key column
- "reverse" (default `False`) reverses the sorting.

Note that sorting can be slow on large maps as it will recreate all key dictionaries.

2.6 Printing

Printing a MultiDirMap will output it as a table with key columns marked by an asterisk. Formatting parameters can be set by

```
MultiDirMap.print_settings(max_width=80, max_cols=4, max_col_width=20)
```

- "max_width" sets the maximum total width of the table in characters
- "max_cols" set the maximum number of columns that will be displayed. Supernumerary columns will be replaced by "…"
- "max_col_width" sets the maximum width of each column in characters. Entries that are too long will be cropped.

```
>>> print(crew)
character*          portrayed_by*       role                nicknames
================================================================================
Malcolm Reynolds    Nathan Fillion      Captain             ['Mal', 'Captain Ti
Zoë Washburne       Gina Torres         First Mate          None
Hoban Washburne     Alan Tudyk          Pilot               Wash
Inara Serra         Morena Baccarin     Companion           Ambassador
Jayne Cobb          Adam Baldwin        Mercenary           Hero of Canton
Kaywinnet Lee Frye  Jewel Staite        Mechanic            Kaylee
Simon Tam           Sean Maher          Medic               None
River Tam           Summer Glau         None                Méi-mei
Derrial Book        Ron Glass           Shepherd            Preacher
>>> crew.print_settings(max_cols=3, max_col_width=15)
>>> print(crew)
character*      portrayed_by*   ... nicknames
================================================
Malcolm Reynold Nathan Fillion  ... ['Mal', 'Captai
Zoë Washburne   Gina Torres     ... None
Hoban Washburne Alan Tudyk      ... Wash
Inara Serra     Morena Baccarin ... Ambassador
Jayne Cobb      Adam Baldwin    ... Hero of Canton
Kaywinnet Lee F Jewel Staite    ... Kaylee
Simon Tam       Sean Maher      ... None
River Tam       Summer Glau     ... Méi-mei
Derrial Book    Ron Glass       ... Preacher
```

CHAPTER 3

Performance

Information below is provided only to give an idea of what kind of performance can be expected from a MultiDirMap. It is not meant as a replacement for any of the data structures mentioned below, but its flexibility affords it some overlap with them.

3.1 Compared to dict

A MultiDirMap with two columns, one of which is a key column, exhibits slightly slower retrieval than Python's built-in dict with significantly slower creation. **Do not use a MultiDirMap when a dict would suffice** :-)

3.2 Compared to bidict

A MultiDirMap with two columns that are both key columns is *slightly* slower in both retrieval and creation than a bidict.

3.3 Compared to pandas DataFrame

A pandas DataFrame (which does not ensure uniqueness of keys) has significantly faster creation than a MultiDirMap (since it is an array under the hood) but in retrieval, a MultiDirMap vastly outperforms it, since the DataFrame retrieves on O(n) as opposed to O(1).

CHAPTER 4

Reference

4.1 multidirmap

A multidirectional mapping with an arbitrary number of key columns.

class multidirmap.**MultiDirMap**(*columns*, *key_columns=None*, *data=None*)
A multidirectional mapping with an arbitrary number of key columns.

clear()
Clear all key dicts, thereby deleting all stored data.

get(*key*, *default=None*)
Redirects to get() of the primary key dict.

items()
Redirects to items() of the primary key dict.

keys()
Redirects to keys() of the primary key dict.

pop(*key*, *default=<object object>*)
Pop an entry from the mapping.

Entry is returned from the primary key dict and it as well as all consequently orphaned entries are removed from the secondary key dicts.

popitem()
Pop from the end of the primary key dict.

All consequently orphaned entries are removed from the secondary key dicts.

print_settings(***kwargs*)
Change the print settings for __str__().

max_width gives the maximum width of the entire table. max_cols gives the maximum number of columns. max_col width gives the maximum width of each column.

reorder_secondary_keys()
Refresh the order of the secondary key dicts.

This will recreate all secondary key dicts such that the keys will be in the same order as the primary keys, as the secondary key order can get scrambled when secondary entries are overwritten.

sort(*key=<function <lambda>>*, *reverse=False*)
Sort the map by the given key.

If no key is given, sorting is done by the entries in the primary key column.

update(*data*, *overwrite='primary'*, *skip_duplicates=False*)
Update the map with the provided data.

overwrite can be "none", "primary", "secondary", or "all" and determines whether an entry can still be added when it conflicts with an existing entry. skip_duplicates determines whether a conflicting entry that will not overwrite should be skipped. If False, an exception will be raised in that situation and a rollback performed, so that the update() operation does not change the state of the map.

values()
Redirects to values() of the primary key dict.

exception multidirmap.**DuplicateKeyError**
Raised when secondary key is not unique.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

MultiDirMap could always use more documentation, whether as part of the official MultiDirMap docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/janrg/multidirmap/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

To set up *multidirmap* for local development:

1. Fork multidirmap (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/multidirmap.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- pytest -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

CHAPTER 6

Authors

- Jan Greis - https://github.com/janrg

CHAPTER 7

Changelog

7.1 0.2.0 (2019-07-12)

- Custom sorting
- Reordering of secondary keys

7.2 0.1.0 (2018-07-28)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index