hLabs: Project HYPAR

Last modified: 13 July 2019

1 Hierarchy

```
Portfolio (object)
├── start: datetime.datetime (earliest of self.stocks' starts)
├── end: datetime.datetime (latest of self.stocks' ends)
├── partition_dates: List[List[start, end]]
├── stock_ids: set(ticker)
└── stocks: Dict[ticker, Stock]
    └── Stock (StockSegment)
        ├── ticker: str
        ├── pseudonym: str (format: S-123 or S-123x for multiple segments)
        ├── start: datetime.datetime (earliest of self.segments' starts)
        ├── end: datetime.datetime (latest of self.segments' ends)
        ├── data: StockData (object)
        │   ├── ticker: str
        │   └── several Dicts and Lists containing API data
        ├── segment_ids: Set[segment_id]
        └── segments: Dict[segment_id, StockSegment]
            └── StockSegment (object)
                ├── ticker: str
                ├── pseudonym: str
                ├── segment_id: str (format: ticker-123)
                ├── start: datetime.datetime
                ├── end: datetime.datetime
                └── num_shares: int
```

2 segment_id and pseudonym Generation

A `StockSegment` is guaranteed a unique `segment_id`. A `set` of previously used `segment_ids` ensures that the numerical component of the id does not increase unnecessarily. Only when the `set` is empty is a new id generated using `itertools.count()` generator. A `StockSegment`'s `segment_id` is generated at the time of `Stock` instantiation to ensure uniqueness. `StockSegment.pseudonym` is generated at the time of `Portfolio` instantiation for the same reason. The numerical component of `pseudonyms` of both `StockSegment` and `Stock` is controlled by the same mechanism as `segment_id` using a separate `set` and `itertools.count()` instance.

3 Lookup Efficiency

A Portfolio contains a dict of Stock objects that are key-referenced by the Stock ticker. In this way, a Stock lookup is $O(1)$ on average. At the Portfolio level, a StockSegment can be searched with efficiency $O(1)$ by first searching the Portfolio.stock_ids and then the Stock.segment_ids. While having a set for each Stock is somewhat redundant, its maintenance is minimal because it is a set and it avoids $O(n)$ when adding/removing a StockSegment. At the Stock level, of course, StockSegment lookup is $O(1)$ by referencing the StockSegment by its segment_id in the Stock's segments dict.

4 Add/Remove Efficiency

4.1 Adding Stock and StockSegment

Adding a Stock and/or StockSegment at the Portfolio level has $O(n)$[1] efficiency. Adding a Stock requires first to check if the ticker exists in the Portfolio.stock_ids set. While this set may seem redundant, iterating through a dict's keys, values, and items has $O(n)$ efficiency, which is no better than a list. Thus, like the Stock.segment_ids set, this reduces lookup to $O(1)$.

If a Stock is being added, its ticker is checked for existence in Portfolio.stock_ids. If it exists, then a comparison of Stock.segment_ids is performed between the existing Stock and that which is being added. If the difference between sets is not null, the difference of the sets are added to the existing Stock. Stock's and Portfolio's start, end, and pseudonym (Stock only) are updated accordingly[2]. If the Stock.ticker does not exist, the Portfolio's start, end, and stock_ids are updated accordingly.

If a StockSegment is being added to the Portfolio, its ticker is searched for in the Portfolio.stock_ids, which is $O(1)$. If it does not exist, a Stock is created and added to the Portfolio. The Portfolio's start, end, and stock_ids are updated accordingly.

4.2 Removing Stock and StockSegment

To remove a Stock, its existence is first checked. If it exists, it is removed from the Portfolio's stock_ids set and stocks dict. This has $O(1)$ because of set and dict lookups by key are both $O(1)$.

To remove a StockSegment, its existence is checked first by its ticker in Portfolio.stock_ids. If it exists, its segment_id existence is checked in Stock.segment_ids.If it exists, it is removed. This has efficiency $O(1)$.

In both cases, the Portfolio's startand end are updated accordingly.

[1]This actually is just an estimate. Best, average, and worst-case scenario need to be computed.

[2]The start and end dates are compared to the current Portfolio.start and Portfolio.end. If the newly added element's start is earlier than the existing Portfolio.start, it replaces it. Likewise, if the newly added element's end is after the existing Portfolio.end, it replaces it.