pysegmenttree

Viacheslav Greshilov

Nov 28, 2021

CONTENTS

Segment tree is a data structure to perform efficient range queries over an array.

For example, finding the sum/minimum/maximum of the arbitrary continuous interval in *O(Log[N])* time. Logariphmic time complexity is achieved by storing the original input data in a tree like data structure with some additional precalculated data.

This library implementation is primarly inspired by this beatiful article.

CHAPTER

ONE

LIBRARY INSTALLATION

```
$ pip install pysegmenttree
```

CHAPTER

TWO

QUICK START

```
>> from pysegmenttree import stree

# Build the tree
# 'sum' function is used by default
>> tree = stree([5, 1, 9, 4, 5, 11])

# Find sum on the interval [1, 4)
>> tree.query(1, 4)
14

# Set element with index 3 to 6
>> tree.update(3, 6)
>> tree.query(1, 4)
16
```

CHAPTER

THREE

ADVANCED USAGE

There are three predefined query functions available (`QueryFunction`) that can be used with *int* or *float* trees.

```
>> from pysegmenttree import stree, QueryFunction
>> tree = stree([5, 1, 9, 4, 5, 11], func=QueryFunction.MIN)

# Find min on the interval [1, 4)
>> tree.query(1, 4)
1
```

Plain python functions are also suitable, but with them c-extensions will **not** be used.

```
# Warning! A slow version of segment tree will be used.
>> tree = stree([5, 1, 9, 4, 5, 11], func=min)
>> tree.query(1, 4)
1
```

Example with user-defined class `Vec2D`.

```
>> from pysegmenttree import stree
>> from pysegmenttree.test_utils import Vec2D
# List of 2D vectors
>> tree = stree([Vec2D(0, 1), Vec2D(5, -2), Vec2D(-2, 3)], func=max)
# Find the vector of maximum length on the interval [0, 2)
>> tree.query(0, 2)

Vec2D(x=5, y=-2)
```

CHAPTER

FOUR

METHODS COMPLEXITY

Considering that input array has *N* elements.

Method	Time complexity	Space complexity
constructor	O(N)	O(2*N)
query	O(Log[N])	O(1)
update	O(Log[N])	O(1)

CHAPTER

FIVE

SOURCE CODE

The project is hosted on GitHub.

5.1 Reference

5.1.1 stree

pysegmenttree.**stree**(*source: List[T], func: Union[Callable[[T, T], T]], QueryFunction] = QueryFunction.SUM*) → AbstractSegmentTree

Function that returns the best suitable version of the segment tree for the given input.

Note: To use all advantages of c-api extensions, you should use *QueryFunction* enum memeber in *func* argument.

```
>>> st = stree([0, 1, 2, 3], func=QueryFunction.MIN)
>>> type(st)
<class 'pysegmenttree.c_extensions.IntSegmentTree'>
```

But if you pass `min()` the slower version of the tree will be used, so be careful.

```
>>> st = stree([0, 1, 2, 3], func=min)
>>> type(st)
<class 'pysegmenttree._pysegmenttree_py.PySegmentTree'>
```

The same is true for the *float* trees.

```
>>> st = stree([0.0, 1.0, 2.0, 3.0])
>>> type(st)
<class 'pysegmenttree.c_extensions.FloatSegmentTree'>
```

5.1.2 QueryFunction

class pysegmenttree.**QueryFunction**
Enum representing query functions that can be used to build segment trees using c-api extensions.

5.1.3 PySegmentTree

class pysegmenttree.**PySegmentTree**(*source: List[T], func: Union[Callable[[T, T], T]], QueryFunction] = QueryFunction.SUM*)
Creates a pure python segment tree instance.

func is a function that will be used in *query* method. Must be either a function with two arguments *T*, returning *T* or the *QueryFunction* enum member.

```
>>> st = PySegmentTree([1.5, 1, 0, 2], func=min)
```

len(*st*)
Return the number of items in segment tree *st*.

```
>>> len(st)
4
```

query(*start: int*, *end: int*) → Optional[T]
Performs a query operation on the interval [**start**, **end**) with the function chosen during the creation. Note, that **end** is not included. Behaviour is replicated from the python slice operator.

```
>>> st.query(0, 2)
1
>>> st.query(0, 4)
0
```

update(*i: int*, *value: T*)
Set i-th element of the tree to the specified value **value**.

```
>>> st.update(0, -100)
>>> st.query(0, 2)
-100
```

5.1.4 IntSegmentTree

class pysegmenttree.**IntSegmentTree**(*source: List[int]*, *func: Optional[str] = None*)
Typed version of the *PySegmentTree* implemented in C using *long long int* type. The behavior is the same as for *PySegmentTree* except few moments:

- **func** argument in the constructor has *str* type and must be one of the *QueryFunction* enum values ('sum', 'min', ...).
- Raises `OverflowError` if any element exceeds *long long* type range.
- Much faster than *PySegmentTree*.

5.1.5 FloatSegmentTree

class pysegmenttree.**FloatSegmentTree**(*source: List[float]*, *func: Optional[str] = None*)

Same as *IntSegmentTree*, except it uses *double* C-type under the hood.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX