Mergeable Heap (Doubly Linked List)

This code implements problem 10-2 from the book "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein.

The implementation is with doubly linked list.

🔨 How It Works

main.py

`input_` (class instance) gets from the user the input type, list type and lines from file (if reading from file).

then, calling `read_instructions(input_)` to read the instructions according to `input_`.

instructions.py

read_instructions(input_) use `eval` to treat module variable as a class:

```
module = eval(input_.list_types[input_.list_type])
```

then, decide if to execute instructions from file or manually by the user:

Manually: calling `read_manually(module)` and wait for valid instruction according to the manu.

From file: we use the attribute `lines` of `input_` to iterate each line.

then, `run_instruction(module, line)` which execute current instruction.

`getattr()` is used to call the correct instruction from the correct class, dynamically.

For example, `getattr(x, 'foobar')` is equivalent to `x.foobar`.

That why

```
getattr(module, instructions[inst])(lst1, val)
```

is equivalent to `UnsortedList.insert(lst1, val)` (for example).

linked_list.py, sorted_list.py, unsorted_list.py, disjoint_list.py called from instruction.py, according to `module` variable.

🗀 General Modules Description

- main.py calls settings.py (Settings class) and instructions.py.
- settings.py handles most of the input and output interactions.
- instructions.py execute the instructions from file or from typing.
- linked_list.py has 2 classes: LinkedList and Node.
- SortedList class (sorted_list.py) and UnsortedList class (unsorted_list.py) inherited from LinkedList class.
- DisjointLists (disjoint_list.py) class inherited from UnsortedList class.

🗄 Inside the modules

- `class Settings`

Function	Description
__ init __(self)	create Settings obj with `list_type`, `input_type` and `lines` attrributes
get_lines(self)	get `filename` and return `lines`
print_inst_update(line, lst1, lst2, union_heap, mergeable_heap)	print lists after each instruction

- `instructions.py`

Function	Description
def read_instructions(input_)	calls `read_manually()` or execute line by line
def read_manually(module)	get line from user and calls `run_instruction(module, line)`
def run_instruction(module, line)	get module and line to execute instruction

- `LinkedList`

Function	Description	⏱ Time Complexity
__ init __(self, head=None, tail=None)	initialize empty linked list with: head=tail=null, heap_size=0 and empty dictionary (hash)	O(1)
print_list(self)	prints current heap, head, tail, heap_size and dictionary of self (list). Iterating over the list	O(n)
insert(self, value)	insert value to end of list if not already exists (using a dictionary)	O(1)
value_in_dict(lst, value)	return true if value already in dictionary, else return false (explained in **Important Notes** section)	O(1)
empty_list(self)	return true if list is empty, else return false	O(1)

- `Node`

Function	Description	⏱ Time Complexity
__ init __(self, value=0, next_node=None, prev_node=None)	creates a node in doubly linked list	O(1)
get_next(self)	get next node of current's node	O(1)
set_next(self, new_next)	set next node of current's node	O(1)
def get_prev(self)	get previous node of current's node	O(1)
set_prev(self, new_prev)	set previous node of current's node	O(1)

- `class SortedList(LinkedList)`

Function	Description	⏱ Time Complexity
__ init __(self)	calls LinkedList __ init __ and create empty list	O(1)
make_heap()	call the SortedList __ init __ constructor	O(1)
insert(self, value)	if value not exists in dict, creates node and search the right position.	O(n)
minimum(self)	print and return the minimum of the list (its the head, that why its O(1))	O(1)
extract_min(self)	print, remove and return the minimum of the list (its the head, that why its O(1))	O(1)
union(self, l1, l2)	union l1 and l2 and creates new list which is self (explained below)	O(n) (n is the longest list)
union_insert(self, current)	get current node, insert it to end of self (list) in O(1) and return next node	O(1)

```
    # a snippet of union(self, l1, l2):
    # starts from the the head of each list, find the minimum between the two,
    # and get the next node after the minimum between the two.
    ...
    l1_current = l1.head
    l2_current = l2.head

    while l1_current and l2_current:
        if l1_current.value < l2_current.value:
            l1_current = self.union_insert(l1_current)
        else:
            l2_current = self.union_insert(l2_current)

    ...
```

- `class UnsortedList(LinkedList)`

Function	Description	⏱ Time Complexity
__ init __(self)	calls LinkedList __ init __ and create empty list	O(1)
make_heap()	call the UnSortedList init constructor	O(1)
minimum(self)	print and return the minimum of the list in O(n). The list isn't sorted so we need to find the minimum in a linked list.	O(n)
extract_min(self)	print, remove and return the minimum of the list in O(n). uses minimum() to find the minimum node.	O(n)
union(self, l1, l2)	union l1 and l2 and creates new list which is self (just connect the tail of l1 to head of l2)	O(1)
super().insert(self,value)	insert value to end of list if not already exists (using a dictionary)	O(1)

- `class DisjointLists(UnsortedList)`

Function	Description	⏱ Time Complexity
__ init __(self)	calls UnsortedList __ init __ and create empty list	O(1)
make_heap()	call the DisjointLists __ init __ constructor	O(1)
insert(l1, l2, first_heap, value)	checks that the lists are disjoints. if the user try to insert to list2 value that exists in list1 it doesn't insert, else it's insert. it uses a dictionary (hash table) to check it.	O(1)
super().minimum(self)	print and return the minimum of the list in O(n). The list isn't sorted so we need to find the minimum in a linked list.	O(n)
super().extract_min(self)	print, remove and return the minimum of the list in O(n). uses minimum() to find the minimum node.	O(n)
super().union(self, l1, l2)	union l1 and l2 and creates new list which is self (just connect the tail of l1 to head of l2)	O(1)

Summery

Function	Sorted	Unsorted	Disjoint Unsorted
Make Heap	O(1)	O(1)	O(1)
Insert	O(n)	O(1)	O(1)
Minimum	O(1)	O(n)	O(n)
Extract Minimum	O(1)	O(n)	O(n)
Union	O(n)	O(1)	O(1)

💡 Important Notes

- To see full details of the linked list structure - uncomment lines 46-50 in linked_list.py
- There are 3 example folders for each type of list, presenting the output of 1 input example.
- To insert more lists to union: after 'union', type 'MakeHeap' to create another heap. insert values to the list, then type 'union' and then 'MakeHeap' again. etc...
- When list union with empty list, it creates mergeable heap.
- In python, 'in' operation on a dictionary has a time complexity of O(1)
- test files are in the top level folder.

✨ Execute program

- create .txt file in the top level directory
- open cmd in the same directory or open a project on your preferred IDE.
- run main.py
- type the file name when the program ask to.

Examples:

1) Detailed Output - Sorted List (portion of output)

```
insert filepath: reg_t.txt
----------------------

instruction: Insert 98

~~~ HEAP 1 ~~~
current list: [5, 10, 20, 100]
head: 5
tail: 100
heap_size: 4
dict: {5: 5, 10: 10, 100: 100, 20: 20}

~~~ HEAP 2 ~~~
current list: [97, 98]
head: 97
```

```
tail: 98
heap_size: 2
dict: {97: 97, 98: 98}

----------------------

instruction: Union

~~~ Mergeable Heap ~~~
current list: [5, 10, 20, 97, 98, 100]
head: 5
tail: 100
heap_size: 6
dict: {5: 5, 10: 10, 20: 20, 97: 97, 98: 98, 100: 100}

----------------------
```

2) 3 heaps Sorted List

```
insert filepath: 3_heaps.txt
----------------------
instruction: MakeHeap

~~~ HEAP 1 ~~~
current list: []
----------------------

instruction: Insert 4

~~~ HEAP 1 ~~~
current list: [4]
----------------------

instruction: MakeHeap

~~~ HEAP 1 ~~~
current list: [4]
~~~ HEAP 2 ~~~
current list: []
----------------------

instruction: Insert 3

~~~ HEAP 1 ~~~
current list: [4]
~~~ HEAP 2 ~~~
current list: [3]
----------------------

instruction: Union

~~~ Mergeable Heap ~~~
current list: [3, 4]
----------------------

instruction: Insert 2
```

```
~~~ Mergeable Heap ~~~
current list: [2, 3, 4]
----------------------

instruction: MakeHeap

~~~ HEAP 1 ~~~
current list: [2, 3, 4]
~~~ HEAP 2 ~~~
current list: []
----------------------

instruction: Insert 1

~~~ HEAP 1 ~~~
current list: [2, 3, 4]
~~~ HEAP 2 ~~~
current list: [1]
----------------------

instruction: Union

~~~ Mergeable Heap ~~~
current list: [1, 2, 3, 4]
----------------------
```

3) Doubles test on Disjoint Unsorted List

```
insert filepath: doubles_test.txt
----------------------
instruction: MakeHeap

~~~ HEAP 1 ~~~
current list: []
----------------------

instruction: Insert 1

~~~ HEAP 1 ~~~
current list: [1]
----------------------

instruction: Insert 1

~~~ HEAP 1 ~~~
current list: [1]
----------------------

instruction: Insert 17

~~~ HEAP 1 ~~~
current list: [1, 17]
----------------------

instruction: Insert 17
```

```
~~~ HEAP 1 ~~~
current list: [1, 17]
----------------------

instruction: Insert 4

~~~ HEAP 1 ~~~
current list: [1, 17, 4]
----------------------

instruction: Insert 4

~~~ HEAP 1 ~~~
current list: [1, 17, 4]
----------------------

instruction: Insert 1

~~~ HEAP 1 ~~~
current list: [1, 17, 4]
----------------------

instruction: Insert 17

~~~ HEAP 1 ~~~
current list: [1, 17, 4]
----------------------

The minimum is: 1
Extract the minimum
instruction: ExtractMin

~~~ HEAP 1 ~~~
current list: [17, 4]
----------------------

instruction: MakeHeap

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: []
----------------------

instruction: Insert 1

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: [1]
----------------------

instruction: Insert 1

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: [1]
```

```
----------------------

17 exists in first heap
instruction: Insert 17

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: [1]
----------------------

17 exists in first heap
instruction: Insert 17

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: [1]
----------------------

4 exists in first heap
instruction: Insert 4

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: [1]
----------------------

4 exists in first heap
instruction: Insert 4

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: [1]
----------------------

instruction: Insert 1

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: [1]
----------------------

17 exists in first heap
instruction: Insert 17

~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: [1]
----------------------

The minimum is: 1
Extract the minimum
instruction: ExtractMin
```

```
~~~ HEAP 1 ~~~
current list: [17, 4]
~~~ HEAP 2 ~~~
current list: []
----------------------

instruction: Union

~~~ Mergeable Heap ~~~
current list: [17, 4]
----------------------
```

Author

Idan Kogan

316375591

2022