HW05 adtComp

Author

linnil1

Array

Implement

Use a dynamic array to store elements,
If size is over capacity, capacity will become twice large(1 -> 2 -> 4 -> …), and copy all elements to new array.
This will cause additional N copy operations, overall is 2N, so this data structure is O(N), which is linear to data amount.

After sort, `_isSort` set to `true` and
after `push_back` and `erase` , `_isSort` set to `false` .
Otherwise, remain same.

Need to improve

Iterator should overload `-` `<` which is easy for coding.

Sort algorithm

Because `std::sort` has many optimization methods, it's unfair to other sort written by me.
I write a merge sort (with buffer)
But, there are a little difference:

1. When I merge it, I create a array that store the index after sorted. (Linear merge is possible see https://en.wikipedia.org/wiki/Merge_algorithm#Merging_two_lists (https://en.wikipedia.org/wiki/Merge_algorithm#Merging_two_lists))
2. Swap the array until index same as in array.
3. for example
   `c e a b d`
   and my index array is `2 5 0 1 3`
   Then swap `c` to index 2, swap `a` to index 0

   swap `e` to index 4, swap `d` to index 3, swap `b` to index 1
   As the result, index are same as in array.
   This will do minimal steps for swapping data and newing data.
4. Do insertion sort when num < 4. (For compare to std::stable_sort)
5. complexity `O(N log N)`

In the case that the size of testObj is large.
using std::stable_sort

```
adt> adtr 200
adt> adta -r 1000000
adt> adtsort
adt> usage
Period time used : 12.88 seconds
Total time used  : 12.88 seconds
Total memory used: 497.8 M Bytes
```

mySort

```
adt> adtr 200
adt> adta -r 1000000
adt> adtsort
adt> usage
Period time used : 12.89 seconds
Total time used  : 12.89 seconds
Total memory used: 389.1 M Bytes
```

The result shows that my sorting algorithm is as quick as in std and less memory used in my algorithm.

dlist

Implement

Use double linking list to store data.
Add dummy for easier coding.
I implement all function base on `DListNode<T>` , so I overload many `insert` and `erase` function to redirect to `DListNode<T>` type for reusing code.

Need to improve

Iterator need to have a function to get `_node` .

Size should be maintain by variable `_size`.

Sort

I use merge sort.
First, count the size of list, and set iterator go through `size / 2` to middle.
Separate from middle and divide and conquar.
Last, merge is very easy, if the right element is larger than left, just insert it (This will not copy any data). (This is also linear time)

Overall complexity `O(N log N)`

Result

```
adt> adtr 200
adt> adta -r 1000000
adt> adtsort
adt> usage
Period time used : 3.9 seconds
Total time used  : 3.9 seconds
Total memory used: 272.1 M Bytes
```

This is a lot quicker than std::sort with array.

BST

Implement

Binary search tree without balancing.
data structure: parent, left and right tree, data

Need to improve

Balance the tree is very important, which maintain the tree height to `O(log N)`

Iterator

When iterating, if it has right tree, go to right tree and go left until stop and return it, otherwise, go up to parent until it is the left tree of parent.

By this porperity, add dummy for easier coding, dummy is the max one, and its right tree and parent point to itself.
This makes `begin := ++dummy` and `end := dummy`

When go reversed, just switch left and right.

This is different from reference code, I maintain parent to make iterator to travel around, when traveling will cause about 2 times than reference, bus save memory.
My BST

```
dt> adtr 200
adt> adta -r 1000000
adt> usage
Period time used : 5.6 seconds
Total time used  : 5.6 seconds
Total memory used: 272.4 M Bytes

adt> adtdelete -r 100
adt> usAGE
Period time used : 9.99 seconds
Total time used  : 15.37 seconds
Total memory used: 272.1 M Bytes
```

11/16 Updated
I had heard that reference is compiled by O3
My BST with O3
The result is not very far away from reference

```
adt> adtr 200
adt> adta -r 1000000
adt> usage
Period time used : 4.34 seconds
Total time used  : 4.34 seconds
Total memory used: 272 M Bytes

adt> adta -r 100
adt> usAGE
Period time used : 0 seconds
Total time used  : 4.34 seconds
Total memory used: 272.4 M Bytes
```

ref BST

```
adt> adtr 200
adt> adta -r 1000000
adt> usage
Period time used : 3.62 seconds
Total time used  : 3.62 seconds
Total memory used: 272.3 M Bytes

adt> adtdelete -r 100
adt> usAGE
Period time used : 4.67 seconds
Total time used  : 8.29 seconds
Total memory used: 272.3 M Bytes
```

Delete

1. If it is leaf, remove it.
2. If it has only a leaf, remove it, and connnect parent and it's only child.
3. If it is node, ++it to get the next one.
   The next one will have only a leaf or no leaf, remove it same as 1 or 2. and swap this node and next node.
   complexity `O(log N)`

find

Just go down as BST properity, if find the first same thing return iterator.
However, if you want to find where to insert and meet the same one, it should go to left tree (ff is not NULL), and find until left tree is NULL.
complexity `O(log N)`

So insert it N times, complexity `O(Nlog N)`

Summery

I test three data structure with sorting.
The result of sorting time is `dlist < BST < Array`.
`Array` is easiest implement structure, but when sorting, it need lots of coping data.
`dlist` cannot random access, however in merge sort algorithm, it can be very fast.
`BST` is a structure that always maintain sorting order. In my test case, we use random data which make tree not very unbalanced.