B03901078 電機二 蔡承佑

|  |  |  |  |
| --- | --- | --- | --- |
|  | DList | Array | BST |
| data member in a node | \_prev, \_next, \_data | \_data | \_left, \_right, \_data |
| operator++ | Access by \_next  O(1) | Acquired by pointer arithmetic  O(1) | 1.Find the right child. If it exists, return right child.  2.一路往左上走，直到parent有right child為止  About O(log(n)) if balanced |
| operator-- | Access by \_prev  O(1) | Acquired by pointer arithmetic  O(1) | 1.Find the left child. If it exists, return left child.  2.一路往右上走，直到parent有left child為止  About O(log(n)) if balanced |
| begin() | An iterator maintained in the class.  O(1) | An iterator maintained in the class  O(1) | An iterator maintained in the class  O(1) |
| end() | An iterator maintained in the class.  O(1) | Acquired by pointer arithmetic  O(1) | An iterator maintained in the class  O(1) |
| sort() | Insertion sort, O(n2) | STL sort, O(nlog(n)) | N/A |
| size() | Count the path from \_head through \_next to \_head  O(n) | A data member kept in the class  O(1) | A data member kept in the class.  O(1) |
| push\_front() | Modify the first and the second to let them point to the new element  O(1) | 1.Move all the element backward to space out the memory for new element  2.Assign the element to the first position  O(n) | 1.Set\_head->\_leftChild to the newly added element.  2.Update \_head by \_head--  About O(log(n)) if balanced |
| push\_back() | Modify the last and the second last element to let them point the new element  O(1) | 1.Check whether capacity is enough, if not, declare a new memory.  2.Directly assign the element to the last.  O(1) | 1.Set \_tail->rightChild to the newly added element  2.Update \_tail by \_tail++  About O(log(n)) if balanced |
| insert(T&) | N/A | N/A | 1.Check whether the newly added element would change the head and the tail, if so, call push\_front() and push\_back() to handle it.  2.Using binary search to find the appropriate position to insert.  About O(log(n)) if balanced |
| pop\_front() | Connect the second and the last element  O(1) | 1.Delete the first element.  2.Move all the element forward.  O(n) | 1.Delete \_head  2.Assign the new head to \_head  O(1) |
| pop\_back() | Connect the second last element and the first one.  O(1) | Directly delete the last element.  O(n) | 1.Delete \_tail  2.Assign the new tail to \_tail  O(1) |
| erase(T&) | 1.Find the required element from begin()  2.Call the function below  O(n) to find | 1.Find the required element from begin()  2.Call the function below  O(n) to find | 1.Find the required element from begin()  2.Call the function below  O(n) to find |
| erase(iterator) | Connect the neighbor nodes of the node to be deleted  O(1) | 1.Delete the element  2.Move the elements behind forward.  O(n) | 1.Check whether \_head or \_tail, if so, call pop\_back() and push\_back() to handle it, which can also maintain \_head and \_tail  2.For degree-0 node, delete it directly.  3.For degree-1 node, assign its child to its parent.  4.For degree-2 node, replace its value with its successor, then delete its successor.  About O(log(n)) if balanced |
| adta –r 200000 | Prediction:  Fast | Prediction:  Fast | Prediction:  Medium slow |
| 0.02sec | 0.04sec | 0.13sec |
| adtd –f 200000 | Prediction:  Fast | Prediction:  Slow O(n) | Prediction:  Medium slow |
| 0.01sec | 239.4sec | 0.05sec |
| adtd –b 200000 | Prediction:  Fast | Prediction:  Slow O(n) | Prediction:  Medium slow |
| 0.02sec | ~0 | 0.06sec |
| adts | Prediction:  very slow O(n2) | Prediction:  slow O(nlog(n)) | N/A |
| 486.7sec | 0.06sec |
| adtd –r 20000 | Prediction:  Slow | Prediction:  Fast O(1) | Prediction:  Medium slow O(log(n) |
| 1.17sec | 3.57sec | 260.1sec |
| memory use  (200000 nodes) | 10.24MB | 12.1MB | 10.98MB |

p.s. the “\_tail” in bst here I means the last element, not the dummy point.

“adtd –r 20000” is most beyond prediction. Maybe the most time consuming for bst is “getPos”, from \_head to traverse to the given position. BST needs much more time than dlist to access the next node.(Maybe there’s better solution for it)

And the time needed for random delete for array is also non-intuitive. I guess most time is consumed on moving the elements.

Conclusion: dlist and array are easy and intuitive, but both has drawbacks, which is extremely slow and can be fatal disadvantage. Though BST is harder to visualize, but its log(n) performance is obviously better than O(n).