CS 165

Project 2 Report

Wenzhuo Wang

14444481

1) Next Fit

Pseudocode:

For each item:

If item.size > current_bin.remain_capacity:

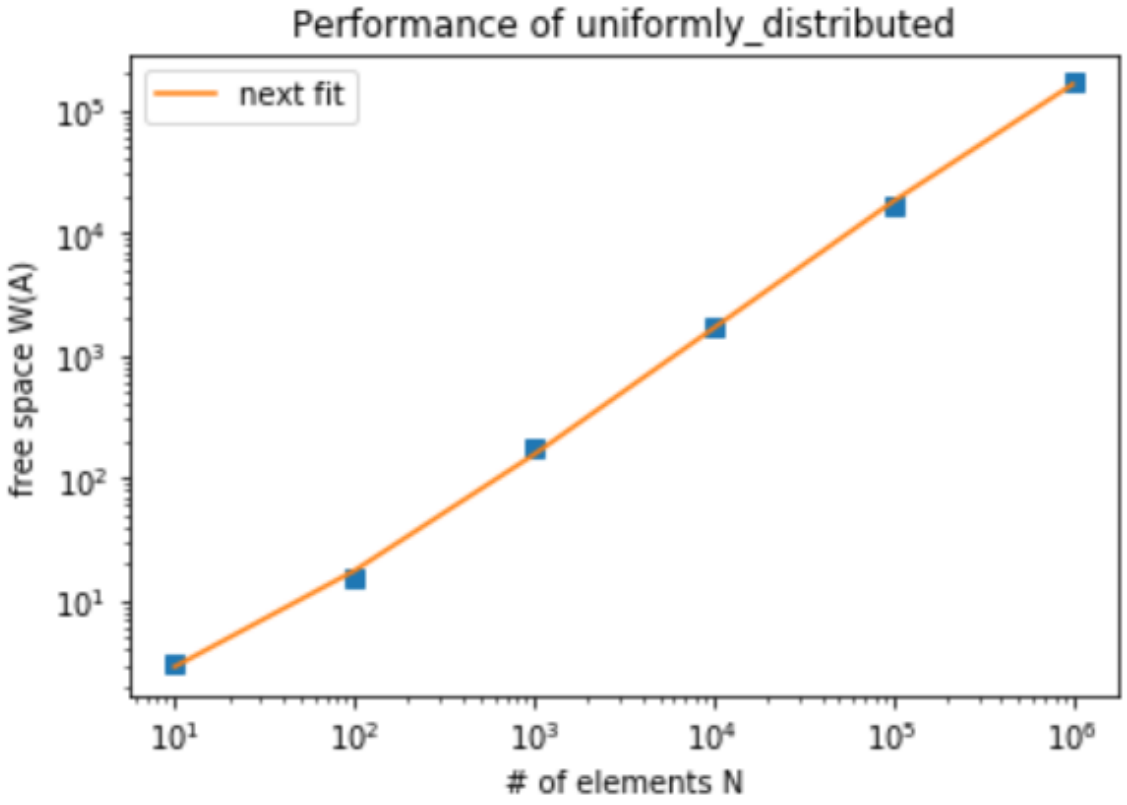
Assign item to current_bin

Else

Create a new bin

Assign item to current_bin

The Next_fit algorithm sets item in current bin by comparing the size and remaining capacity. If the remaining capacity is larger, then the algorithm will assign the item to the current bin and subtract the bin's remaining capacity with the item's size. If the remaining capacity is smaller, then the algorithm will create a new bin. The new bin becomes the current bin, and assigns the item to the current bin and subtract bin's remaining capacity with the item's size. In this algorithm, the assignment of item will only focus on the current bin and will never look back. Hence, there will be a lot of wasted capacity simply because of a large item.



W(A) = 0.16644 N + 1.34208

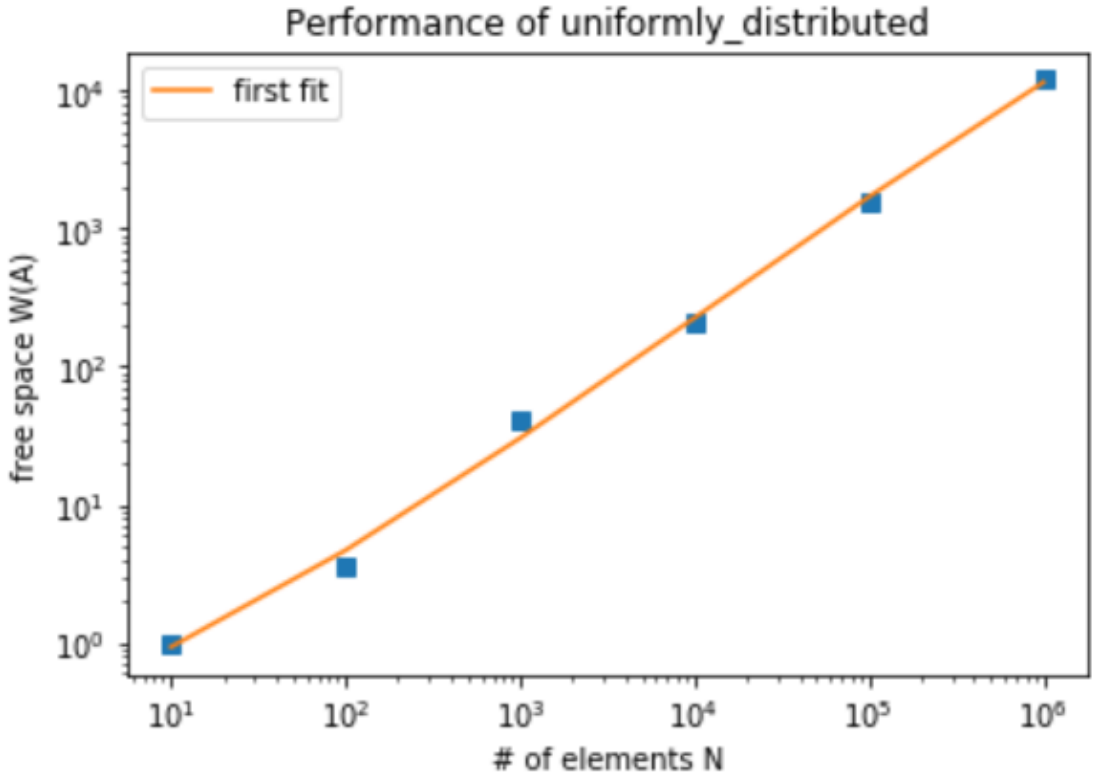
2) First Fit

Pseudocode:

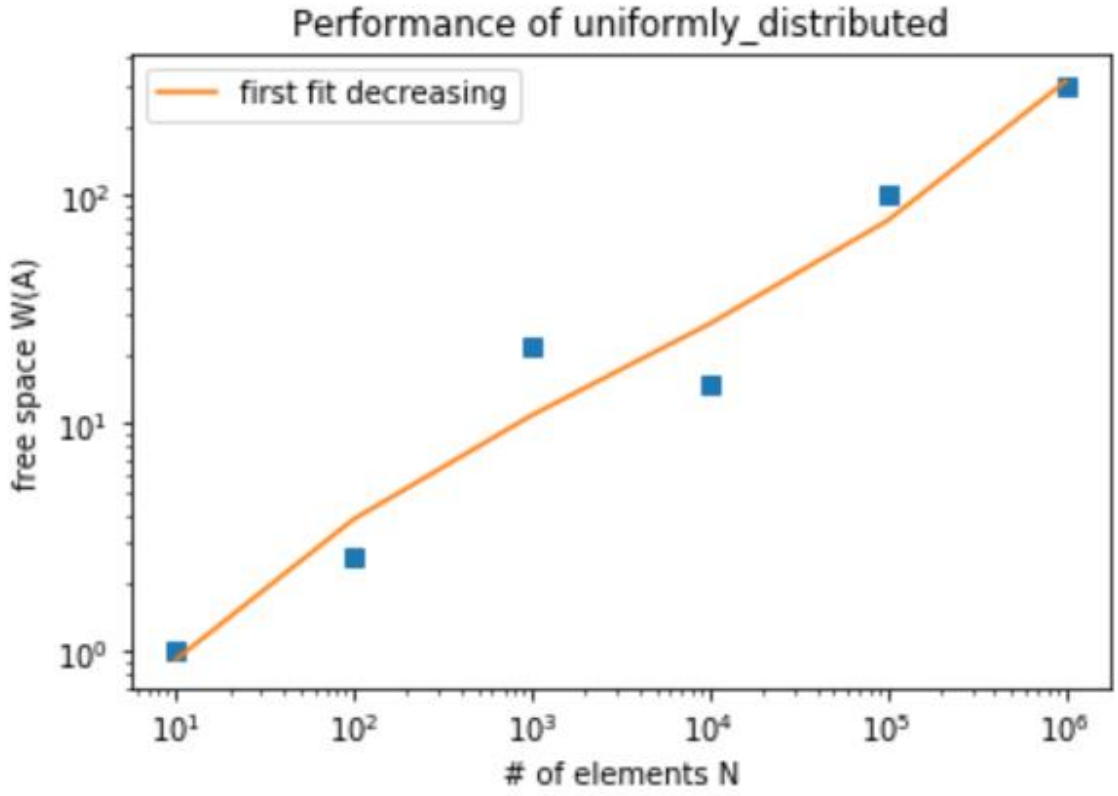
```
For each item:
    Bin = find_first(tree, item)
    If  Bin
        Assign item to Bin
    Else
        Create a new bin
        Assign item to new bin
```

The First_fit algorithm sets the item in the first bin whose remaining capacity is suitable for storing the item. The first argument of find_first function allows two type of parameters of data structures. The first is vector, which takes O(n^2) time. The second is binary search tree, which takes O(n log n) time. The Find_first algorithm will search all of the created bins and return the first bin found which has larger remaining capacity than item. In the fast version, it will compare the item with the root's remaining capacity, if the item is less or equal to the remaining capacity, it will return the root. If not, the pointer to the root will point to the root's right node. Keep this cycle until a node is returned or the current root becomes null. If it does not find any bin, it will return null. After running the find_first function, if a bin is found, the algorithm will assign the item to this bin and subtract the bin's remaining capacity with the item's size. If no bin is available, the algorithm will create a new bin, and assign the item to this bin and subtract bin's remaining capacity with the item's size.
The only difference between the First_fit_decreasing algorithm and the First_fit algorithm is that the former creates a new vector of pair of the item and the item's index and sorts the new vector. Then the First_fit_decreasing algorithm will run the same as how the First_fit algorithm runs but handle the item with its original index instead.

For the binary search tree, it is implemented by a class of Node. The Node has two pointer pointing to the left and right node, and it stores data by remain_cap, index. For every node, the remain_cap of its left node will be smaller, and the remain_cap of its right node will be larger. As a balanced tree, for every node, the difference between the height of its left node and its right node must be smaller than two. This feature could be achieved by the left rotation and right rotation. In this way, the height of whole tree will be smaller than logN so that the complexity of searching a node is O(log n). Thus, the complexity of the whole first_fit algorithm is O(n log n).



W(A) = 0.01199 N + 0.88659



W(A) = 0.000301 N + 1.003519

3) Best Fit

Pseudocode:

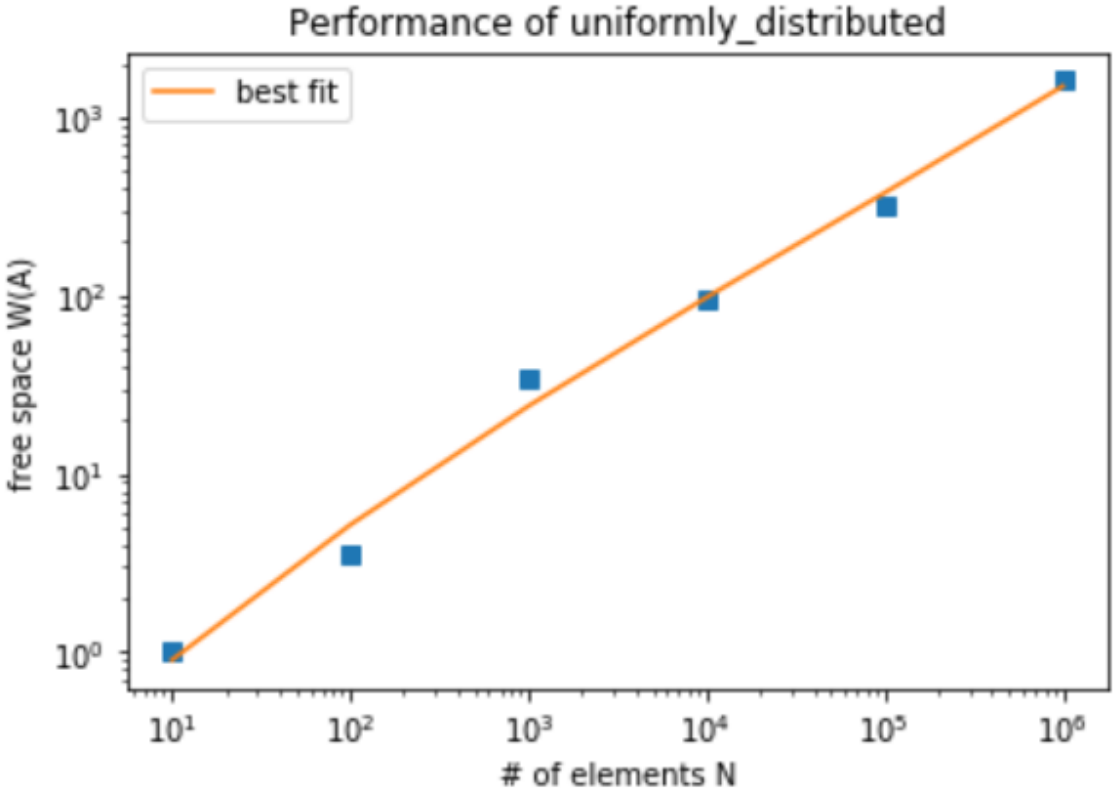
```
For each item:
    Bin = find_best(tree, item)
    If  Bin
        Assign item to Bin
    Else
        Create a new bin
        Assign item to new bin

find_best(tree, item)
    Node *best = NULL;
    int found = 0;
    Node *node = tree;
    while (node != NULL && !found) {
        double result = node.remain_cap - value;
        if (result > 0) {
            if (best == NULL || (node.remain_cap < best.remain_cap) )
            {best = node;}
            node = node.left;
        }
        else if (result < 0) {
            node = node.right;}
        else {
            best = node;
            found = 1;}}
    return best;}
```

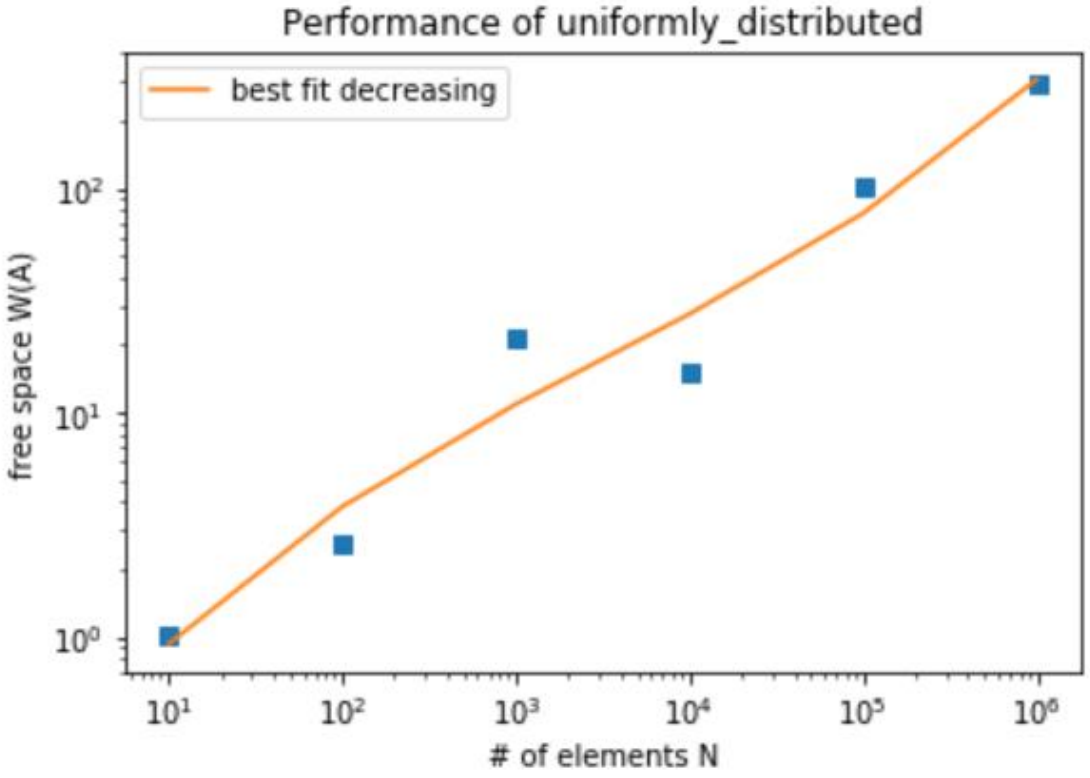
The Best_fit algorithm finds the bin which has the closest remaining capacity to the item. In this way, the item could “best” fit all capacity of the bin. The algorithm will try to find the best bin first. The fast find_best function will take in two arguments, a binary search tree and the item value. The function will see if the root of tree is suitable for the item. If the remaining capacity is exactly the same as the item, it will return the root. If the remaining capacity is smaller than the item, the pointer will then point to the right of the root. If the remaining capacity is larger than the item and the current root is less than the best node, the best node will become the current node. And the pointer of root will point to the left of the root until the tree is null. Then the next step is the same as the find_ best algorithm. If the bin is found, the algorithm will remove that bin and after modified the remaining capacity, insert the bin back with the bin index and the new capacity. If the bin is not found, the algorithm will insert a new index with capacity of (1-item) into the tree.

The only difference between the Best_fit_decreasing algorithm and the Best_fit algorithm is that the former creates a new vector of pair of the item and the item’s index and sorts the new vector. Later the Best_fit_decreasing algorithm will run the same as how the Best_fit algorithm runs but handle the item with its original index instead.

The BST construction mis the same as the first_fit using the remain_cap as the key to decide the node location.



W(A) = 0.001605 N + 0.99048



W(A) = 0.000291 N + 1.003619

Conclusion

The next_fit algorithm will waste a lot of space, so it is the worst fit algorithm. The first_fit algorithm could use the space more efficiently than the next_fit algorithm could, but it still wastes many spaces, while the first_fit_decreasing algorithm and the best_fit_decreasing algorithm waste similar amount of space. For the different algorithm, the next fit performs worst, and the best_fit performs the best. For the same algorithm, the one handling the sorted items in decreasing order could waste as little space as possible.