초록: 보고서 전체 내용 요약 1문단으로 작성

-This report compares 2 ADT which are Binary Search Tree (BST) and Priority Queue Min Heap in solving the symmetric difference problem between 2 sets. The reference solution uses BST which have average case efficiency but can have possibilities to be skewed and not bushy and have worst case efficiency. The proposed methodology uses Min Heap and Heap Sort to sort both sets and uses 2 pointers to compare the unique elements. This method guarantees a stable time complexity and simpler logic as well as more efficient space memory usage.

1장: 서론

● 선택한 자료구조 설명 요약

- priority queue min heap, has ability to efficiently retrieve the smallest element. A min heap is a complete binary tree where the smallest value is always at the root. This allows for constant-time access to the minimum element (O(1)) and efficient insertion and deletion operations in logarithmic time (O(log n)). It is especially useful in applications like sorting, duplicate removal, and set operations (symmetric difference), offering both time and space efficiency.

● 선택한 자료구조의 중요성 (e.g., 실제 응용 사례)

- A Min Heap is a data structure based on a complete binary tree, where the smallest value is always located at the root node. It guarantees O(log n) time complexity for insertion and deletion operations. This structure is frequently used to implement a priority queue and plays a central role in various algorithms that require repeated selection of minimum values. In practice, min heaps are widely used in operating system task scheduling, network packet processing, and real-time event priority management, making them a fast and efficient tool for handling data in many fields.

● 문제 설명

Given two sets of numbers, A and B, the problem is to find the size of the symmetric difference. The number of elements that are in A but not in B (A - B) and in B but not in A (B - A). The input includes the number of elements and elements of each set, and the output is the total number of elements in the symmetric difference. For example, if A = {1, 2, 4} and B = {2, 3, 4, 5, 6}, then A - B = {1} and B - A = {3, 5, 6}, making the symmetric difference size 4.

● reference solution 요약

The reference solution uses a Binary Search Tree (BST) to compute the size of the symmetric difference between sets A and B. First, it inserts the elements of A into the BST, and then, while reading elements of B, it checks if each element exists in A to count common elements. It then calculates the number of elements unique to each set and adds them.

● 제시하는 방법론의 동기 (reference solution 의 한계점 요약)

The reference solution uses a BST to store elements of A and compare with B, which has an average time complexity of O(log n) for insertion and search, but can degrade to O(n) in the worst case if the tree becomes unbalanced. This performance instability is a limitation. Therefore, this project proposes using a min heap, which guarantees O(log n) time complexity for both insertion and deletion, ensuring more stable and consistent performance when calculating the symmetric difference and minimize the space used. Plus heaps handle duplicate priorities much more naturally than BST.

2장: reference solution 분석

● 선택한 자료구조 상세 설명

-The ADT used on the problem was Binary Search Tree.

- Bushy Binary search tree is a set of nodes and a set of edges that connect those nodes. BST have constraint where there is only exactly one path between any two nodes. The one node is called the root. Every node except the root has exactly one parent. A node that has no child is called a leaf. Every node can either has 0,1 or 2 children only.

-The property of BST is every key in the left subtree is less than root key and every key in the right subtree is greater than root key. And it satisfied the transitive order. Also, no duplicate keys are allowed. Which means no same data in one tree.

- This structure allows basic operations such as search, insert, and delete to be performed with average time complexity of O(log n). However, if the tree becomes skewed (not bushy), the time complexity can degrade to O(n).

● reference solution 상세 설명 (해당 자료구조의 이론적 배경, 시간/공간 복잡도, 연산 방식 등 reference solution을 이해하는 데 필요한 심층적인 설명)

-In the reference solution, elements of set A are first inserted into a BST. Then, for each element in set B, a search is performed in the A tree to check for existence and count common elements. Elements unique to B are stored in a separate tree. The final symmetric difference size is calculated by subtracting the number of common elements from the total number of nodes in A and adding the number of unique nodes in B.

-There are few functions operation that are must to know to understand the reference solution which are:

-CreateNode – to create node in bst, first we need to do memory allocation by using malloc function. Then we must initialize data that are in the node such as key, left and right. After that the node is ready for insertion.

-Insert - to insert key in to the BST, first we must make sure that the root is not NULL and if it is happen to be NULL we can call the function CreateNode to create new node. Then, to satisfy the BST property we need to check either key is greater or smaller than the root. If the key is smaller than the root, key will be inserted at the left subtree but if the key is greater than the root, key will be inserted at the right subtree. Last return the root.

-Search – Now that all key data is inserted at the tree we can search any key by using Search function. The concept of search is basically same like inserted where we must check if the key is smaller or greater then start search based on right or left subtree. We can use recursion calls to keep going down the tree until found the key.

-CountNodes – In this function if root is not NULL it will recursively count the number of node in left and right subtrees and adds 1 for the currect node. Thus, it will return the total number of nodes in the whole BST including the root.

-FreeBST- This function is simply to free the memory after all works are done.

-main – In main function, it read the size of set A and B. Next read and Insert the element of set A in to BST. Then it read element of set B, if the element is already in set A increment will happen in both sets. Else it will be inserted into set B BST which is another BST. After that, it will count the total nodes that are in A but not in B and vice versa. And lastly print symmetric difference result and free all memory allocated.

Based on this reference solution we can see that the the time and space complexity based on each function are:

|  |  |  |
| --- | --- | --- |
| function | Time | space |
| Insert | Size\_a  Average case: O(N log n)  Worst case: O(n)  Size\_b  Average case: O(n log n)  Worst case: O(n^2) | Max memory use:  Nodes A + Nodes B  : O(size\_b + size\_a)  Worst case: O(n)  Skewed tree: O(n) |
| Search | size\_b search in set a  Average case: O(size\_b log size\_a)  Worst case: O(size\_b \* size\_a) |
| CountNodes | Tree traversal: O(n) |

● reference solution 의 한계점 상세 설명

Based on this reference solution there are a few limited we can see here. The first is if the tree is not bushy and it become skewed it will falls into worst case where time complexity can be O(n) and not efficient. Second, to handle the duplicate data it use two BST which is to store nodes A and nodes B this requires more conditional logic and make the code more complex also more memory space is needed.

3장: 제시하는 방법론

● 제시 방법론 요약 설명

The methodology that I would like to purpose is by using the Min Heap. We can sort both sets use Min Heap then applies a two-pointer technique to efficiently calculate the symmetric difference. First sort the set A and set B using heap sort based on Min Heap. Then, compared the element in the sorted array and extract only the different values which are part of the symmetric difference.

● 2장에서 제시한 한계점을 극복할 수 있는 근거 서술

By using this method, it ensures the stability of O(n log n) sorting performance and avoid the skewed case that can happen in BST. It performs a linear comparison of 2 array which give the overall time complexity remains O(n log n). the conditional logic for this method is much more direct, avoid unnecessary nested loops, and less complicated compare to BST method. Lastly since this method use array based heaps which improve the efficiency of space management or take less memory.

● 제시 방법론 상세 서술 (소스코드 별도 제출)

From the main function, first we will read the size of set a and b(n &m) then read each set data and insert it into the array. Here we use the two indices to walk through both sorted array ay the same time. After insertion now we will sort the array use Heap Sort. Sorting is a must for efficient array comparison. Then we can just using 2 pointers to compare both array and count the unique data. If the data are equal then we just going to skip it because they are in both sets. If data in a less than b means its only in set A and counter will increase. If data in a more than b means its only in set B and counter will increase too. And finally add the remaining elements to get final result of symmetric different.

Thus, in this method there are few functions that we must understand to solve the question:

CreateHeap- in this function we will do memory allocation for heap then initialize data in the heap and return it.

Insert- for insertion the value will be insert at the end or last of the leaf which is at last index of the array. Size increase and to maintain the min heap property it will go through heapifyUp.

HeapifyUp- Since data was inserted at the end of the leaf we need to make sure the value satify the min heap property, else if it is not satisfy it will get swap with its above parent. This will loop until the data is bigger than its parent and less than its child.

Swap- swap function is simply to swap the element that needed by using temp variable.

ExtractMin- Next, to extract the min we can just extract the first index in the array and decrease the size. After that data is remove min heap need to go through heapify down.

HeapifyDown- heapify down is also compulsory to maintain the min heap property. After deletion happened the new data that replace the old data need to be check to make sure the new data is the smallest or not. First compare the left side. If the left data is less than root then data will go down to left and left root can be candidate to be new root. Same goes to right side. Until the data reach index it will swap with possible candidate recursively.

HeapSort- Now that all data are correctly inserted into min heap, we need to sort it. A min heap keeps the smallest element at the top but it does not store elements in sorted order. Only the heap[0] is the smallest. So it will call buildHeap to build the array and after that it will called extractMin to extract element in order since extractmin remove the smallest element and returns it. Then the value is stored into arr[i] which is sorted in ascending order.

buildHeap- this function build a valid min heap from unsorted array. It copies the elements from the array into the heap, then start from the last non-leaf and do heapify down to maintain min heap property.

4장: 성능 평가 및 결과 분석

● http://boj.kr/60ca4032b38e4ed8a82906e7b753df8b

● 성능 평가 결과 분석

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Problem | Original ADT | Time / Space | Proposed ADT | Time / Space |
| 1269:  symmetric  intersection | BST | 284ms  13524kb | Min heap | 104ms  3468kb |

Min Heap takes 180ms less time which is 2.73 times faster than BST and save 10074kb memory space which is 3.9 times less memory.

Min Heap method not only improves time efficiency by nearly 3x, but also drastically reduces memory usage by almost 4x compared to the BST implementation. This demonstrates its clear advantage for large-scale or performance-sensitive applications

5장: 결론

● 제시하는 방법론의 의의 및 한계점

In conclusion, this report proposed the methodology od using Min Heap instead of BST to avoids the BST’s worst case time complexity issue O(n) caused by imbalanced BST and provides a stable performance of O( n log n) through heap sort. This makes it highly significant. Moreover, the logic is more straight forward and simple to implement and at the same time offers excellent memory management. However, for very small or extremely large data this method may perform worse.

● 제시하는 방법론의 실제 활용 방안

This Min Heap-based approach is well-suited for sorting large datasets and computing set differences or symmetric differences. It can be effectively applied in various fields such as real-time event processing, log analysis, network packet classification, and priority-based task queues. Especially in systems where stable performance and low complexity are critical, this method can serve as a more suitable alternative to BST.