자료구조 프로젝트

문제 선정

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Num | ori | Ori time memory | New 1 | New time memory |  |
| 1920  \*contains | Hashtable | 72ms  5716kb | Merge sort+ binary search(array) | 72ms  1896kb |  |
|  |  |  | Min heap |  |  |
| 2751  \*sort up | Merge sort | 404ms  8932kb | minheap | 448ms  5024kb |  |
| 1269  \*find same intersection | bst | 284ms  13524kbb | Min heap | 104ms  3468kb |  |

1. 서론

* 선택한 자료구조 설명 요약
* Min heap pq
* 선택한 자료구조의 중요성 (e.g., 실제 응용 사례)

- Min heap은 완전 이진 트리 기반의 자료구조로, 항상 루트 노드에 가장 작은 값이 위치하도록 구성되어 있으며, 삽입과 삭제 연산이 O(log n)의 시간 복잡도를 보장하는 것이 특징이다. 이 구조는 우선순위 큐(Priority Queue)의 구현에 자주 사용되며, Dijkstra 알고리즘과 같이 최소값을 반복적으로 선택해야 하는 다양한 알고리즘에서 핵심적으로 활용된다. 실제로 운영체제의 작업 스케줄링, 네트워크 패킷 처리, 실시간 이벤트 우선순위 관리 등 많은 분야에서 min heap은 빠르고 효율적인 데이터 처리 도구로 널리 쓰이고 있다.

* 문제 설명

-이 문제는 공집합이 아닌 두 자연수 집합 A와 B가 주어졌을 때, A에는 있지만 B에는 없는 원소(A - B)와 B에는 있지만 A에는 없는 원소(B - A)를 합친 대칭 차집합의 원소 개수를 구하는 것이다. 입력으로는 집합 A와 B의 원소 개수, 그리고 각 집합의 원소들이 주어지며, 출력은 대칭 차집합에 포함되는 원소들의 총 개수이다. 예를 들어 A = {1, 2, 4}, B = {2, 3, 4, 5, 6}일 때, A - B = {1}, B - A = {3, 5, 6}이므로 대칭 차집합의 크기는 4가 된다.

* reference solution 요약

- reference solution는 이진 탐색 트리(BST)를 이용하여 두 집합 A와 B의 대칭 차집합 크기를 구하는 프로그램으로, 먼저 A의 원소들을 BST에 삽입하고, B의 원소들을 입력받으면서 A에 존재하는지 탐색하여 공통 원소 개수를 센 뒤, A에는 있지만 B에는 없는 원소의 개수와 B에는 있지만 A에는 없는 원소의 개수를 각각 계산하여 합한 값을 출력하며, 마지막에 동적 할당된 메모리를 해제한다.

* 제시하는 방법론의 동기 (reference solution 의 한계점 요약)
* 기존 reference solution에서는 이진 탐색 트리(BST)를 사용하여 집합 A의 원소를 저장하고 B의 원소와 비교함으로써 대칭 차집합의 원소 개수를 계산하지만, BST는 삽입과 탐색 과정에서 평균적으로는 O(log n)의 성능을 보이지만 최악의 경우 O(n)까지 떨어질 수 있으며, 균형을 유지하지 않으면 성능 저하가 발생한다는 단점이 있다. 따라서 본 프로젝트에서는 항상 O(log n)의 시간 복잡도로 삽입 및 삭제가 가능한 min heap을 활용하여 보다 안정적이고 일관된 성능으로 대칭 차집합 계산을 수행하고자 한다.

1. reference solution 분석

* 선택한 자료구조 상세 설명 (bst)

-이진 탐색 트리(BST, Binary Search Tree)는 각 노드가 최대 두 개의 자식 노드를 가지며, 왼쪽 서브트리에는 현재 노드보다 작은 값들이, 오른쪽 서브트리에는 큰 값들이 위치하도록 구성된 자료구조이다. 이러한 구조 덕분에 탐색, 삽입, 삭제와 같은 기본 연산들이 평균적으로 O(log n)의 시간 복잡도를 가지며, 정렬된 데이터를 트리 형태로 표현하고 빠르게 접근할 수 있는 장점이 있다. 하지만 트리가 한쪽으로 치우치게 되면(not bushy) 최악의 경우 O(n)의 시간 복잡도를 가질 수 있는 단점도 존재한다.

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

- Reference solution에서는 BST를 사용하여 먼저 집합 A의 원소들을 이진 탐색 트리에 삽입하고, 집합 B의 각 원소에 대해 A 트리에서 존재 여부를 탐색하여 공통 원소 수를 파악한 후, B에만 있는 원소들은 별도의 트리에 저장한다. 이후 A의 전체 노드 수에서 공통 원소 수를 뺀 값과 B 전용 트리의 노드 수를 더해 대칭 차집합의 원소 개수를 계산한다. 이때 삽입 및 탐색은 평균적으로 O(log n)의 성능을 보이며, 공간 복잡도는 각 트리에 포함된 노드 수만큼 O(n)을 차지한다. 해당 방식은 메모리를 적절히 사용하면서 두 집합 간의 중복 여부를 빠르게 판별할 수 있도록 설계되어 있다.

* reference solution 의 한계점 상세 설명

-이 BST 기반 reference solution은 평균적인 상황에서는 효율적일 수 있지만, 입력된 원소들이 이미 정렬된 상태이거나 특정 방향으로만 치우친 경우 트리가 편향되어 탐색 및 삽입 시간이 O(n)으로 증가할 수 있다는 구조적 한계를 가진다. 또한 중복 처리를 위해 별도의 조건문과 두 개의 트리를 사용해야 하므로 코드가 복잡해지고 메모리 관리가 다소 까다로울 수 있다. 특히 대용량 입력이 들어올 경우 균형을 유지하지 않는 일반 BST는 성능이 급격히 저하될 수 있어, 이러한 경우에는 보다 안정적인 시간 복잡도를 보장하는 자료구조가 필요하다.

1. 제시하는 방법론

* 제시 방법론 요약 설명

-제안하는 방법론은 최소 힙(Min Heap)을 활용하여 두 집합을 정렬한 후, 투 포인터 기법을 통해 대칭 차집합을 효율적으로 계산하는 방식이다. 입력된 두 집합 A와 B를 각각 최소 힙 기반의 힙 정렬로 정렬한 후, 정렬된 상태에서 각 원소를 비교하면서 서로 다른 값들만 추출함으로써 대칭 차집합의 개수를 계산한다. 이 방식은 힙을 사용하여 안정적인 O(n log n) 정렬 성능을 확보하며, BST 기반 방식에서 발생할 수 있는 성능 불안정을 방지하는 장점을 가진다.

* 2장에서 제시한 한계점을 극복할 수 있는 근거 서술

- BST는 삽입과 탐색의 평균 시간 복잡도가 O(log n)이지만, 트리가 한쪽으로 치우칠 경우 O(n)까지 성능이 저하될 수 있다는 구조적 한계가 있다. 반면, 최소 힙을 활용한 제안 방식은 입력 데이터를 먼저 O(n log n) 시간에 정렬한 후, 선형 시간에 두 배열을 비교하므로 전체 알고리즘의 시간 복잡도는 항상 O(n log n)으로 유지된다. 따라서 입력 순서나 데이터 분포에 따라 성능이 불안정해지는 BST의 약점을 보완할 수 있고, 코드 구조도 더 단순하며 메모리 해제와 중복 체크 로직이 명확하다는 점에서 유지보수성과 효율성 측면에서 우수하다.

* 제시 방법론 상세 서술 (소스코드 별도 제출)

-본 방법론에서는 먼저 각각의 집합 A와 B를 배열로 입력받고, 최소 힙 기반의 힙 정렬 알고리즘을 사용하여 두 배열을 정렬한다. 이후 투 포인터 기법을 적용하여 두 정렬된 배열을 동시에 순회하면서 값이 동일한 경우 건너뛰고, 값이 다를 경우 대칭 차집합에 해당하는 원소로 간주하여 개수를 누적한다. 이 과정을 모든 원소에 대해 수행하고, 한쪽 배열의 순회가 끝난 이후에는 남은 원소 수를 추가하여 최종 대칭 차집합의 크기를 출력한다. 이 방식은 정렬된 상태를 이용하여 불필요한 탐색을 제거하고, 일정한 시간 복잡도로 결과를 도출할 수 있어 효율적이다.

1. 성능 평가 및 결과 분석

* acmicpc.net 에서의 성능 평가 결과 첨부 (acmicpc.net 에서 ‘내 제출’ - 소스코드 - ‘공유’ 버튼으로 boj.kr 링크 생성 후 보고서에 첨부)
* 성능 평가 결과 분석

1. 결론

* 제시하는 방법론의 의의 및 한계점
* 제시하는 방법론의 실제 활용 방안