[백준] 1920번: 수 찾기

yoool · 5일 전

2023_Autumn_Algorithm_Study

목록 보기 7/7

> 1920번: 수 찾기
> N개의 정수 A[1], A[2], ..., A[N]이 주어져 있을 때, 이 안에 X라는 정수가 존재하는지 알아내는 프로그램을 작성하시오.
>
> - N(1 ≤ N ≤ 100,000)
> - N개의 정수 A[1], A[2], ..., A[N]
> - M(1 ≤ M ≤ 100,000)
> - M개의 수: A에서 각 수의 존재 여부(M개의 줄에 출력: 1, 0)
> - 모든 정수의 범위는 -2^31 보다 크거나 같고 2^31보다 작다.

해시테이블(hash table)

- 키-주소 매핑에 의해 구현된 사전 ADT
- 버켓 배열 + 해시함수
- 탐색, 삽입, 삭제: O(n) 최악시간, O(1) 기대시간

해시함수(hash function) h

- 주어진 형의 키를 고정 범위 [0, M-1]로 매핑 (M은 배열의 용량)
- h(x) = x % M

분리연쇄법(separate chaining)

- 각 버켓 A[i]는 리스트 L_i에 대한 참조를 저장
- L_i는 해시함수가 버켓 A[i]로 매핑한 모든 항목들을 저장, 미니 사전
- 단순하고 빠르지만, 테이블 외부에 추가적 저장공간 요구

해시테이블의 크기 설정

- 소수 선택
- 해시 테이블이 클수록 필요 메모리는 많지만, 충돌을 줄임

제출 번호	아이디	문제	결과	메모리	시간	언어	코드 길이	제출한 시간
69016223	516938	1920	맞았습니다!!	4152 KB	216 ms	C99 / 수정	1466 B	34초 전
69016201	516938	1920	맞았습니다!!	4312 KB	92 ms	C99 / 수정	1467 B	50초 전
69016071	516938	1920	맞았습니다!!	5716 KB	80 ms	C99 / 수정	1468 B	3분 전

- 해시테이블 크기로 1007, 10007, 100007을 넣었을 때의 결과이다
  크기가 커질수록 메모리는 많이 사용하지만, 시간은 빨라진다
  나는 10007을 넣는게 가장 적당하다고 생각했다

```
#pragma warning(disable:4996)
#include<stdio.h>
#include<stdlib.h>
#define HASHTABLE_SIZE 10007

typedef struct Node {
    int data;
    struct Node* next;
} Node;

int hash(int key) {
    return (key >= 0) ? (key % HASHTABLE_SIZE) : ((-key) % HASHTABLE_SIZE);
}

void insertKey(Node* table, int key) {

    int index = hash(key);
    Node* new = (Node*)malloc(sizeof(Node));
    if (new == NULL) exit(1);
    new->data = key;
    new->next = table[index].next;
    table[index].next = new;

}

int searchKey(Node* table, int key) {

    int index = hash(key);
```

```
        Node* cur = table[index].next;

        while (cur != NULL) {
            if (cur->data == key) {
                return 1;
            }
            cur = cur->next;
        }
        return 0;

    }

    int main() {

        int N, M, i, key;

        scanf("%d", &N);

        Node* table = (Node*)malloc(HASHTABLE_SIZE * sizeof(Node));
        if (table == NULL) exit(1);

        for (i = 0; i < HASHTABLE_SIZE; i++) {
            table[i].next = NULL;
            table[i].data = 0;
        }

        for (i = 0; i < N; i++) {
            scanf("%d", &key);
            insertKey(table, key);
        }

        scanf("%d", &M);

        for (i = 0; i < M; i++) {
            scanf("%d", &key);
            printf("%d\n", searchKey(table, key));
        }

        for (i = 0; i < HASHTABLE_SIZE; i++) {
            Node* cur = table[i].next;
            while (cur != NULL) {
                Node* temp = cur;
                cur = cur->next;
                free(temp);
            }
        }
        free(table);

        return 0;
    }
```