DSA (CSE102) Practice Questions for lab

26-03-25

Introduction

This document presents four practice problems related to hashing. Only hashing solutions are allowed, and the use of the standard template library (STL) is forbidden. Each problem statement is accompanied by sample inputs/outputs. **Problems 1 and 2** include detailed solutions in both **C** and **C++**. **Problems 3 and 4** include the problem statements and sample I/O, without provided solutions.

1 Problem 1: Find the First Non-Repeated Character

Problem Statement (Hashing Only)

You are given a string S consisting of ASCII characters (assume the set is limited to ASCII range 0–255). Use **only hashing** to determine the first character in S that does not repeat anywhere else in the string. If there is no such character, print `-1`.

Constraints

- Only hashing is allowed.
- No data structure (like arrays for sorting) aside from what is needed to implement your hash approach.
- You may use a fixed-size array of length 256 as a frequency table (this is effectively a hash map keyed by character code).

Sample Input

```
swiss
```

Sample Output

```
w
```

Explanation In `swiss`, the characters appear as follows:

- `s` appears 3 times
- `w` appears 1 time
- `i` appears 1 time

The first character (left-to-right) that appears exactly once is `w`.

C Solution (Without STL)

```
#include <stdio.h>
#include <string.h>

// We assume characters in the ASCII range, so we can use
// an array of size 256 as a "hash" for frequency.
int main() {
    char S[1000];  // Adjust size as needed
    fgets(S, sizeof(S), stdin);

    // Remove trailing newline if any
    size_t len = strlen(S);
    if (len > 0 && S[len - 1] == '\n') {
        S[len - 1] = '\0';
    }

    // Frequency array for ASCII range
    int freq[256];
    for (int i = 0; i < 256; i++) {
        freq[i] = 0;
    }

    // 1) Build frequency map
    for (int i = 0; S[i] != '\0'; i++) {
        unsigned char ch = (unsigned char) S[i];
        freq[ch]++;
    }

    // 2) Identify first non-repeated character
    int found = 0;
    for (int i = 0; S[i] != '\0'; i++) {
        unsigned char ch = (unsigned char) S[i];
        if (freq[ch] == 1) {
            printf("%c\n", ch);
            found = 1;
```

```
            break;
        }
    }

    if (!found) {
        printf("-1\n");
    }

    return 0;
}
```

C++ Solution (Without STL)

```
#include <iostream>
#include <cstring>  // For strlen

using namespace std;

int main() {
    char S[1000];
    cin.getline(S, 1000);

    // Frequency array for ASCII characters
    int freq[256];
    for (int i = 0; i < 256; i++) {
        freq[i] = 0;
    }

    // 1) Build frequency map
    int length = strlen(S);
    for (int i = 0; i < length; i++) {
        unsigned char ch = (unsigned char) S[i];
        freq[ch]++;
    }

    // 2) Find first non-repeated character
    bool found = false;
    for (int i = 0; i < length; i++) {
        unsigned char ch = (unsigned char) S[i];
        if (freq[ch] == 1) {
            cout << S[i] << endl;
            found = true;
            break;
        }
```

```
    }

    if (!found) {
        cout << -1 << endl;
    }

    return 0;
}
```

2 Problem 2: Pair With a Given Sum

Problem Statement (Hashing Only)

You are given an array *arr* (of integers, potentially within some reasonable range) and a target sum K. Use a **hash table** (manually implemented, no STL) to find **any one pair** of elements whose sum is K. If no such pair exists, output an empty indicator, for example `-1 -1`.

Constraints

- Only hashing is allowed (no sorting or other advanced data structures).
- You must implement your own hash table or use a fixed-size array if the integer range is known and limited.
- Collision handling is required if you implement a general-purpose hash function.

Sample Input

```
arr = [2, 7, 11, 15]
K = 9
```

Sample Output

```
2 7
```

Explanation `2 + 7 = 9`, which matches the target sum.

C Solution (Without STL, Direct Addressing Example)

```
#include <stdio.h>

#define MAX_VALUE 10001  // Adjust as needed (based on expected
    data range)
```

```
int main() {
    int n;
    scanf("%d", &n); // number of elements
    int arr[1000];   // array input
    for(int i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }

    int K;
    scanf("%d", &K);

    // "Hash" array for direct addressing
    // hashTable[x] = 1 if we have encountered x, 0 otherwise
    int hashTable[MAX_VALUE];
    for(int i = 0; i < MAX_VALUE; i++){
        hashTable[i] = 0;
    }

    // 1) Traverse arr
    for(int i = 0; i < n; i++){
        int x = arr[i];
        if(x < 0 || x >= MAX_VALUE){
            // out of range for this direct-address approach
            // skip or handle with a more advanced hash
               structure
            continue;
        }

        int complement = K - x;
        if(complement >= 0 && complement < MAX_VALUE &&
           hashTable[complement] == 1){
            // Found the pair (complement, x)
            printf("%d␣%d\n", complement, x);
            return 0;
        }
        // Mark this value as seen
        hashTable[x] = 1;
    }

    // No pair found
    printf("-1␣-1\n");
    return 0;
}
```

C++ Solution (Without STL, Open Addressing Example)

```
#include <iostream>
using namespace std;

struct HashItem {
    int key;
    bool occupied;
};

static const int TABLE_SIZE = 101;

int hashFunction(int key) {
    if (key < 0) key = -key;
    return key % TABLE_SIZE;
}

void insertKey(HashItem table[], int key) {
    int idx = hashFunction(key);
    int startIdx = idx;
    while (table[idx].occupied) {
        idx = (idx + 1) % TABLE_SIZE;
        if (idx == startIdx) {
            // Table is full, or we circled back
            return;
        }
    }
    table[idx].occupied = true;
    table[idx].key = key;
}

bool searchKey(HashItem table[], int key) {
    int idx = hashFunction(key);
    int startIdx = idx;
    while (table[idx].occupied) {
        if (table[idx].key == key) {
            return true;
        }
        idx = (idx + 1) % TABLE_SIZE;
        if (idx == startIdx) {
            return false; // Not found
        }
    }
    return false;
}
```

```
int main() {
    int n;
    cin >> n;
    int arr[1000];
    for(int i = 0; i < n; i++){
        cin >> arr[i];
    }

    int K;
    cin >> K;

    // Initialize hash table
    HashItem table[TABLE_SIZE];
    for(int i = 0; i < TABLE_SIZE; i++){
        table[i].occupied = false;
        table[i].key = 0;
    }

    // Insert and search
    for(int i = 0; i < n; i++){
        int x = arr[i];
        int complement = K - x;

        if (searchKey(table, complement)) {
            cout << complement << "␣" << x << endl;
            return 0;
        }

        // Insert x into the hash table
        insertKey(table, x);
    }

    // No pair found
    cout << "-1␣-1" << endl;

    return 0;
}
```

3 Problem 3: Longest Substring Without Repeating Characters (No Solution Provided)

Problem Statement (Hashing Only)

Given a string S (ASCII-based), determine the **length** of the longest substring containing **no duplicate characters**. You **must** use hashing to keep track of repeated characters. No other data structures (like sorting) may be used.

Sample Input

```
abcabcbb
```

Sample Output

```
3
```

Explanation One of the longest substrings without repeating characters in `abcabcbb` is `abc`, which has length 3.

4 Problem 4: Check for Anagrams (No Solution Provided)

Problem Statement (Hashing Only)

Two strings A and B are said to be anagrams if one can be rearranged to form the other. Using **only hashing**, check if A and B are anagrams. Print `true` if yes, `false` otherwise.

Sample Input

```
A = "listen"
B = "silent"
```

Sample Output

```
true
```

Explanation By rearranging the letters of `"listen"`, you obtain `"silent"`.