Log Analyzer

Data Structure Project Report

Joudi Besaiso 2285455
Ryham Houari 2281222
Abrar Al-harazi 2106235

25 December 2023

Introduction:

In this report, we will examine and compare two implementations to monitoring the most visited web pages using a custom hash table and the unordered_map function from the C++ standard library. The purpose is to evaluate each implementation's efficiency and performance in processing a log file and retrieving the top ten most frequented pages.

Hash table Implementation:

a)Data Structure:

The custom hash table is implemented as a vector of pairs, where each pair represents a key-value pair, with the key being the URL and the value being the number of visitors. The size of the hash table is determined during initialization.

b) Collision Resolution:

The collision resolution method employed is linear probing. In case of a collision When we enter a key-value pair into the hash table, we first compute the hash value of the key using the hash function getHash. Then we check to see if the cell at the calculated index in the table is empty. If it is, we insert the key-value pair into that cell.If the cell is not empty, it indicates a collision. In this situation, we begin probing the table by incrementing the index. We continue probing the table until we locate an empty cell or reach the end of the table.When we discover an empty cell, we insert the key-value pair into it.If we reach the end of the table while probing and haven't discovered an empty cell, we may either wrap around to the beginning of the table and continue searching, or we can extend the table and insert the key-value pair into the newly constructed cell.

c)Hash Function:

We develop a custom hash function hash(string key), which iterates through the key characters, updating a hash value with a bitwise left shift and addition. To produce a valid index, the final hash value is modulo-ed with the size of the hash table.

```
unsigned int HashTable::hash(string key) {
    unsigned int hash = 8581;
    int c;

    for (int i = 0; i < key.length(); i++) {
        c = key[i];
        hash = ((hash << 3) + hash) + c;
    }

    return hash % size;
}
```

Hash unction

In this hash function, It sets the hash variable to a fixed value, in this example, 8581. This is the starting point for the hash computation.It goes over each character of the input key one by one. It performs the following procedures on each character:

hash << 3: Left shifts the current value of hash by 3 bits.

+ hash: Adds the original value of hash.

+ c: Adds the ASCII value of the current character.

These operations are a simple and common way to mix the bits of the hash and combine them with the ASCII value of the current character. This process is often used in hash functions to produce a more distributed result.After all of the characters in the key have been processed, the hash value is calculated modulo the size of the hash table (size). This step guarantees that the hash value fits inside the hash table's valid range of indices. In summary, this hash function uses a combination of bitwise operations and addition to provide a hash value for the input key. Specific design decisions in the hash function may influence issues such as distribution and collision resolution inside the hash table..

Un-ordered-map Implementation:

a)Data Structures:
We made use of The C++ standard library's unordered_map is used to construct a hash table with automated scaling and built-in collision resolution.

b)Collision Resolution:
Internal collision resolution is handled by the unordered_map utilizing techniques such as chaining and open addressing. Depending on the C++ implementation, the particular method may differ.

Finding the Top 10 Most Visited Pages:

First, the getTop10 method populates a vector with all key-value pairs from the hash table.The vector is then sorted in descending order based on the visit count using the C++ sort function as the comparator and a custom lambda function. Second, the getTop10 function in the unordered_map method takes the same strategy as the hash table. The top ten most viewed pages are found by iterating over the unordered_map and ranking the elements in descending order depending on the visit count. It populates a vector with key-value pairs from the unordered_map, sorts them, and then returns them. Finally, the sorting and size reducing processes are the same in both methods. When there are more than ten entries, the result is reduced to simply the top ten.

```
vector<pair<string, int>> UnorderedMap::getTop10() {
    vector<pair<string, int>> result;
    for (const auto& pair : const pair<...> & : hashtable) {
        result.push_back(pair);
    }

    sort( first: result.begin(), last: result.end(), comp: []
    (const auto& a : const pair<string, int> & , const auto& b : const pair<string, int> & )
    {
        return a.second > b.second;
    });

    if (result.size() > 10) {
        result.resize( new_size: 10);
    }

    return result;
}
```

getTop10 Function

Performance Comparison:

In the main program, two programs are generated, one for each implementation, and both are checked on the same log file. The elapsed time for each implementation is measured using the C++ chrono library. The findings show how effective each strategy is at analyzing the log file and locating the most visited pages.

```
Top 10 most visited web pages:
index.html # of total visits: 139961
3.gif # of total visits: 24001
2.gif # of total visits: 23601
4.gif # of total visits: 8014
244.gif # of total visits: 5148
5.html # of total visits: 5010
4097.gif # of total visits: 4874
8870.jpg # of total visits: 4493
6733.gif # of total visits: 4278
8472.gif # of total visits: 3843
Total Elapsed Time : 0.279 seconds

Process finished with exit code 0
```

The hash table run output

```
Top 10 most visited web pages:
index.html # of total visits: 139961
3.gif # of total visits: 24001
2.gif # of total visits: 23601
4.gif # of total visits: 8014
244.gif # of total visits: 5148
5.html # of total visits: 5010
4097.gif # of total visits: 4874
8870.jpg # of total visits: 4493
6733.gif # of total visits: 4278
8472.gif # of total visits: 3843
Total Elapsed Time : 0.295 seconds

Process finished with exit code 0
```

The unordered-map run output

Conclusion:

The custom hash table and unordered_map implementations employ two distinct ways to track the most frequently visited web pages. Criteria such as simplicity, control over implementation details, and performance requirements are used to choose amongst them. Both methods show C++'s flexibility in building efficient and effective data structures for real-world applications.