CSC263 Tutorial #5 Exercises

Hashmaps

Alexander He Meng

Prepared for February 7th, 2025

Question 1

(a) Checking for duplicate names in an unsorted list

To check if any two students have the same name, we can use a hash table where: - The keys are student names. - The values are counts of occurrences.

Algorithm: 1. Initialize an empty hash table. 2. Iterate through the list: - If the name is already in the hash table, return `True` (duplicate found). - Otherwise, insert the name into the hash table. 3. If no duplicates are found, return `False`.

Complexity: $O(n)$, assuming hash table operations are $O(1)$.

Is a hash table the best choice? Yes, because it provides an optimal $O(n)$ solution compared to $O(n^2)$ for nested loops.

(b) Sorting names using a hash table

A hash table does not inherently support sorting. Instead, a more suitable approach is: 1. Insert names into a balanced BST ($O(n \log n)$). 2. Use quicksort or mergesort ($O(n \log n)$).

A hash table is **not** appropriate here because hashing does not maintain order.

Question 2

(a) Open Addressing Insertions

Linear Probing: $h(k,i) = (h'(k) + i) \mod m$

Quadratic Probing: $h(k,i) = (h'(k) + i^2) \mod m$

Double Hashing: $h(k,i) = (h'(k) + i \cdot h''(k)) \mod m$

(b) INSERT Pseudocode

```
INSERT(T, k):
    i = 0
    repeat:
        j = h(k, i)
        if T[j] is empty:
            T[j] = k
            return j
        i = i + 1
        if i == m:
            return ERROR  // Table full
```

(c) SEARCH Pseudocode

```
SEARCH(T, k):
    i = 0
    repeat:
        j = h(k, i)
        if T[j] == k:
            return j
        if T[j] is empty:
            return NOT FOUND
        i = i + 1
        if i == m:
            return NOT FOUND
```

(d) DELETE Discussion Simply replacing the deleted element with `NIL` disrupts probing sequences. A common solution is to use a special `DELETED` marker, which allows search and insertion to function correctly. However, excessive deletions degrade performance, necessitating periodic rehashing.

Hash Table Insertion Algorithm

The following Python implementation demonstrates an insertion function for open addressing in a hash table:

```python
hashtable = [empty] * 11

def INSERT(k):
    for i in range(0, 12):
        bucket = h(k, i)
        if hashtable[bucket] is empty:
            hashtable[bucket] = k
            return
    # Error case when the table is full
    print("Error: Hash table is full")

# Example usage
h(k, i) = 0 + i
INSERT(1)
INSERT(2)
DELETE(1)
SEARCH(2)
print(hashtable)  # Output: [DELETED, 2, NIL, NIL, NIL, NIL]
```

This code implements linear probing for collision resolution, where the function `h(k, i)` determines the next available slot in case of a collision.