Tutorial 10: Hash Table

CAB301 - Algorithms and Complexity

School of Computer Science, Faculty of Science

Agenda

1. **Lecture Recap**: Hash Table
    - Hash Function
    - Collision Resolution
    - Load Factor
2. **Tutorial Questions** + **Q&A**

Data Structures... so far

Data are normally stored in collections, such as arrays, linked lists, trees, etc.

Common operations include **searching**, **inserting**, **deleting**, and **updating**.

Depending on the data structure, complexity varies from $O(\log n)$ (e.g., binary search) to $O(n)$ (e.g., sequential search).

> Can we do better?

Hash Table

> $O(1)$ time complexity for searching, inserting, deleting, and updating.

General idea:

1. **Array** of fixed size n. Call this array $A[0..n-1]$.
2. **Hash function** to map keys (k) to indices. Call this function $h(k)$.

So searching for the index of a key k is as simple as:

$$i \leftarrow h(k)$$

Hash Function

Common strategies for **non-negative numeric** keys:

- **Division method**: $h(k) = k \mod n$
- **Middle-square method**: Square the key, then extract the middle p digits.
- **Folding method**: Divide the key into equal parts of p digits
  - **Shift folding**: Add the parts, and take the last p digits.
  - **Folding at boundaries**: Reverse alternate parts before adding.
- **Selecting digits**: Use specific digits of the key.

Regardless, the hash function should be **efficient**, and **distribute keys uniformly**.

Collision Resolution

Example: $h(k) = k \mod 10$. Here, $h(10) = h(20) = 0$, so we have a **collision** as two keys map to the same index.

Common strategies for resolving collisions:

- **Open addressing**: Find the next available slot, if $x \leftarrow h(k)$ is occupied.
  - **Linear probing**: Check $x + 1, x + 2, x + 3, \ldots$.
  - **Quadratic probing**: Check $x + 1^2, x + 2^2, x + 3^2, \ldots$.
  - **Double probing**: Use a second hash function to determine the next slot.
- **Separate chaining**: Each index is a collection of keys, not just one key.

Load Factor

Load factor $L = \frac{\text{number of keys}}{\text{number of slots}}$ indicates how *full* the hash table is.

For **open addressing**:

- $L < 0.5$: Low chance of collisions, but wasted space.
- $L > 0.8$: High chance of collisions.

For **separate chaining**:

- $L < 1$: waste of space.
- $L > 2$: long chains, slower search.