Direct Access Tables

- Given a set of items have integer keys in the range 1 .. \( m \)
  - Use the value of the key itself to select a slot in a table to directly store the item
  - To search for an item with key \( k \), just look in slot \( k \)
    - If there’s an item there, you’ve found it
    - If the tag is 0, it’s missing.
  - Runs in Constant time, \( O(1) \)
Limitations of Direct Access Tables

- Constraints
  - Keys must be unique
  - Keys must lie in a small range
  - For storage efficiency, keys must be dense in the range
  - If they’re sparse (lots of gaps between values), a lot of space (memory) is needed to store the table

Effect of Non-unique Keys

- If a search can be satisfied by any item with key, \( k \), performance is still \( O(1) \) but
- If multiple items share the same key, then worst-case time to find any specific key is: \( O(n_{max}) \)
  - Where \( n_{max} \) is the largest number of duplicates.
  - A Linked list of duplicates are “attached” to each slot
Hash Function

\[ H( \text{key} ) \rightarrow \text{integer} \]

- To map a key to an integer in the range \( 0 \ldots m-1 \)
- Applying this function to the key produces an address
- If \( h \) maps each key to a unique integer then search runs in \( O(1) \) time

Examples of Hash Functions

- Given a string, \( s \), with length of \( n \) characters, return an integer in value of \( 0 \ldots 255 \)
  - Example 1: Sum the characters
    ```c
    int hash( char *s, int n ) {
        int sum = 0;
        while( n-- ) sum = sum + *s++;
        return sum % 256;
    }
    ```
  - Example 2: XOR the characters
    ```c
    sum = sum ^ *s++;
    ```
  - In general, any function that generates integers in \( 0..m-1 \) for some suitable \( m \) will do
Examples of Hash Function Collisions

- With this hash function
  ```c
  int hash( char *s, int n ) {
    int sum = 0;
    while( n-- ) sum = sum + *s++;
    return sum % 256;
  }
  ```
- hash( “AB”, 2 ) and hash( “BA”, 2 )
  - return the same value!
  - This is called a collision
  - A variety of techniques can resolve collisions

Hash Tables - Collisions

- Collision occurs when the hash function maps two different keys to the same address

- The table must be able to
  - Recognize that >1 item may be at same location
    - Store the actual key with the item in the hash table
    - Check for a hit
      - if ( table[ H(key) ].key == key ) then hit
      - else try another entry
  - Resolve
    - Variety of techniques
Resolving Collisions with Linked Lists

- Collisions - Resolution
  - Linked list attached to each primary table slot
  - Searching for \( i_1 \)
    - Calculate \( H(i_1) \)
    - Item in table, \( i \), doesn’t match
    - Follow linked list to \( H(i_1).\text{next} \)
  - If NULL found, key isn’t in table

Hash Table with Overflow Area

- Overflow area
  - Linked list constructed in special area of table called overflow area
- Adding \( k \) and \( j \) when \( H(k) == H(j) \)
  - Add \( k \)
    - Since \( H[k] \) empty, add \( k \) at \( H[k] \)
  - Add \( j \)
    - Get \( F=\)first free slot in overflow area
    - Put \( j \) into \( F \)
    - Set \( F.\text{next} = H[k].\text{next} \) (might be null)
    - Set \( H[k].\text{next} \) pointer to \( F \)
- Searching: Same as Linked List
Handling Overflow with Re-Hashing

- When a collision occurs, try using a second hash function
  - Many variations
  - General term: re-hashing

- Adding k and j when \( H(k) == H(j) \)
  - Add k
    - Since was \( H[k] \) empty,
    - add k at \( H[k] \)
  - Add j
    - Repeat until \( (H'[j]) \) has an empty slot
      » put j into it
      » Update pointers

- Searching - Use \( H(x) \), then \( H'(x) \)

Handling Overflow with Linear Probing

- Linear probing
  - Go to the next slot until one found empty
    - \( H'(x) + 1 \)

- Can lead to bad clustering
- Re-hash keys fill in gaps between other keys and exacerbate the collision problem
## Summary of Hash Tables

- **Potential $O(1)$ search time**
  - If a suitable function $h(key) \rightarrow integer$ can be found

- **Space for speed trade-off**
  - “Full” hash tables don’t work

- **Collisions**
  - Inevitable
    - Hash function reduces amount of information in key
  - Various resolution strategies
    - Linked lists
    - Overflow areas
    - Linear probing

---

## Hash Table Demonstration