Lecture 8: October 4, 2000

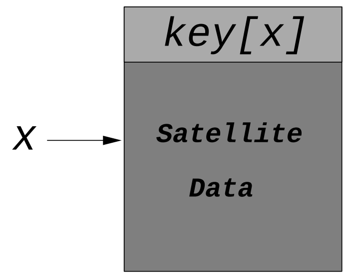
Today:

- Hash Tables
- Collision Resolution
- Choice of Hash Function
- Universal Hashing

Symbol Table Problem

- Maintain a dynamic set, T.
- Support dictionary operations:
  - INSERT(T, x)
  - SEARCH(T, k)
  - DELETE(T, x)

  where x consists of key and satellite data.

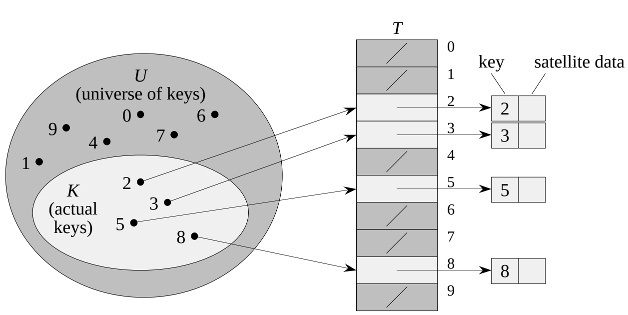


- Examples:
  - Dictionary (word key to definition)
  - Compiler (symbol key to semantic data)

Direct-Address Table

- **Idea:**
  - Universe of keys is $U = \{0, 1, ..., m-1\}$.
  - K = set of keys in use.
  - Define a direct-access table, $T[0..m-1]$, where

$$T[i] = \begin{cases} x & \text{if } i \in K \text{ and } key[x] = i \ , \\ \text{NIL} & \text{otherwise .} \end{cases}$$



Direct-Address Table Dictionary Operations

DIRECT-ADDRESS-SEARCH(T, k)
  return $T[k]$

DIRECT-ADDRESS-INSERT(T, x)
  $T[key[x]] \leftarrow x$

DIRECT-ADDRESS-DELETE(T, x)
  $T[key[x]] \leftarrow$ NIL

- Only $O(1)$ time required for each operation.

Direct-Address Table: Problems

- Range of keys usually large (e.g. ASCII strings).
- Space required for T may be impractical.
- $|K|$ usually much smaller than $|U|$, so...

Wasteful to allocate space for every key in U.

Hash Tables

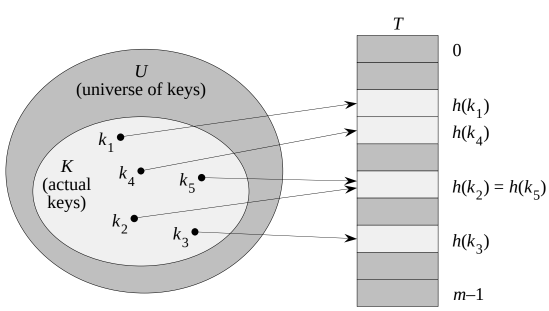
- **Solution:**
  - Use **hash function** h to map U into smaller set, $\{0, 1, \ldots, m-1\}$.

$$h : U \rightarrow \{0, 1, \ldots, m-1\}$$

  - Can create **hash table**, T, where

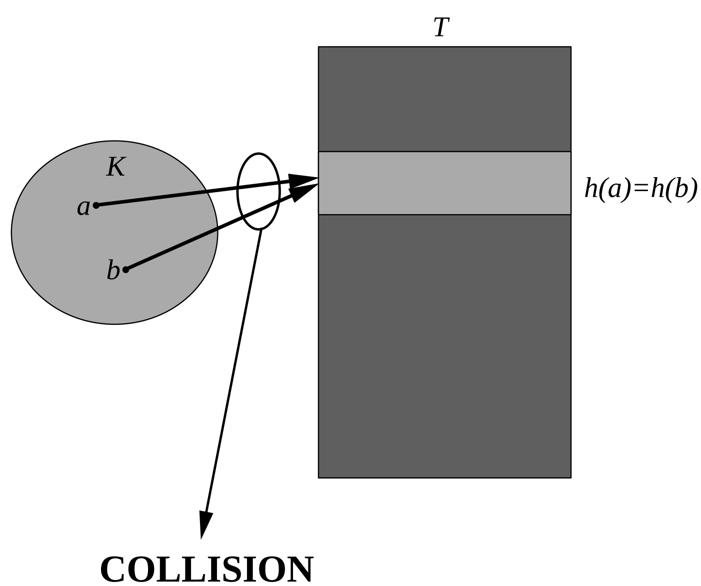
$$T[i] = \begin{cases} x & \text{if } key[x] \in K \text{ and } h(key[x]) = i \text{ ,} \\ \text{NIL} & \text{otherwise .} \end{cases}$$

Hash Tables



Collisions

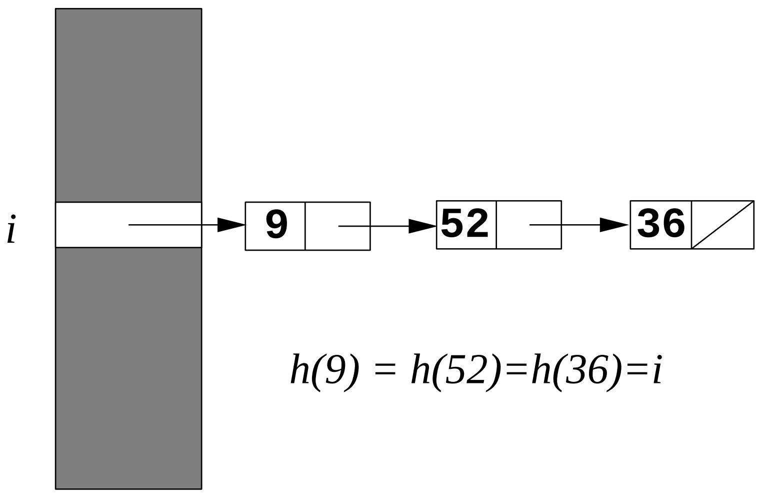
- If some element already occupies slot to which an inserted element is mapped, a **collision** occurs.



- Must detect and resolve collisions! (2 ways)

First method: Chaining

- Each position in hash table is pointer to head of a linked list.
- To insert elements into the table, add to head of list.



Chaining Functions

- **Insertion**

  CHAINED-HASH-INSERT(T, x)
  insert x at the head of list $T[h(key[x])]$

  Worst-case running time $O(1)$.
- **Searching**

  CHAINED-HASH-SEARCH(T, k)
  search for an element with key k in list $T[h(k)]$

  Worst-case running time proportional to length of list $T[h(k)]$ (i.e., $\Theta(n)$).
- **Deletion**

  CHAINED-HASH-DELETE(T, x)
  delete x from the list $T[h(key[x])]$

  Worst-case running time $O(1)$ if doubly-linked lists used.

Analysis of Hashing with Chaining

- Assume each key equally likely to be hashed into any slot (**simple uniform hashing**)
- Given hash table T with m slots holding n elements, define T's **load factor** α as n/m (what is α?)
- Time for computing $h(k)$ is $\Theta(1)$.
- To find an element,
  - Look up its position in the table using h.
  - Search for element in linked list stored at slot.

Analysis Case 1: Unsuccessful Search

- Element for which we are searching is *not* in list.
- Must check each element in the list.
- Uniform hashing $\rightarrow$ average length of lists in $T = \alpha = n/m$.
- Expected number of elements examined $= \alpha$
- Running time: $\Theta(1 + \alpha)$.

Case 2: Successful Search

- Assume CHAINED-HASH-INSERT adds new elements to the end of the list.
- Expected number of elements examined is at most 1 more than number of elements examined when sought-for element was inserted.
- Running time: $\Theta(1 + \alpha)$.

Method 2: Open Addressing

- All elements stored in hash table (i.e., no lists used).
- Each table entry contains either element or NIL.
- When searching for an element, systematically **probe** table slots.
- Hash function, h, determines the sequence of slots examined for a given key.

$$h : U \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}$$

- **Probe sequence** for a given key k given by:

$$\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$$

Open Addressing Insertion

- To insert element with key k into T, check each position in the table in the order specified by h until empty slot is found.

```
HASH-INSERT(T, k)
i ← 0
repeat j ← h(k, i)
        if T[j] = NIL
           then T[j] ← k
                return j
           else i ← i + 1
   until i = m
error "hash table overflow"
```


Open Addressing Searching

- Same as insertion.

```
HASH-SEARCH(T, k)
i ← 0
repeat j ← h(k, i)
        if T[j] = k
           then return j
        i ← i + 1
   until T[j] = NIL or i = m
return NIL
```

- What is drawback of open addressing?

Further Analysis of Open Addressing

- Assume **uniform hashing**.
- Expected number of probes in *unsuccessful* search on an open-address hash table with load factor $\alpha = n/m < 1$ is $\leq 1/(1-\alpha)$.
- Expected number of probes in *successful* search is

$$\leq \frac{1}{\alpha} \ln \frac{1}{1-\alpha} + \frac{1}{\alpha}$$

- Details in book (pp. 237–239).
- Example: hash table $\frac{1}{2}$ full
  Unsuccessful: $\frac{1}{1-\frac{1}{2}} = 2$
  Successful: $\frac{1}{\frac{1}{2}} \ln \frac{1}{1-\frac{1}{2}} + \frac{1}{\frac{1}{2}} \leq 3.387$
- Example: hash table $\frac{9}{10}$ full
  Unsuccessful: $\frac{1}{1-\frac{9}{10}} = 10$
  Successful: $\frac{1}{\frac{9}{10}} \ln \frac{1}{1-\frac{9}{10}} + \frac{1}{\frac{9}{10}} \leq 3.670$

Choice of Hash Function

Ideally:

- Distribute keys uniformly into slots.
- Let $P(k)$ = probability that key k is drawn from U:

$$\sum_{k:h(k)=j} P(k) = \frac{1}{m} \quad \text{for } j = 0, 1, \ldots, m-1$$

  I.e., "sum over all keys k which *hash to* slot j"
- Regularity in key distribution should *not* affect uniformity of hashing!

Division method

- Use hash function

$$h(k) = k \bmod m$$

- Must avoid certain values of m
  - Powers of 2. If $m = 2^p$, $h(k)$ is p lowest order bits of k.
  - Powers of 10. If the keys are decimal numbers, hash function does not depend on all decimal digits of k.
- **Good choices for m are primes not too close to exact powers of 2**

Multiplication method

- Use hash function

$$h(k) = \lfloor m\,(k\,A \bmod 1) \rfloor$$

  where A is a constant, $0 < A < 1$.
- Value of m not critical; typically use $m = 2^p$.
- Optimal choice of A depends on characteristics of data (Knuth says use $A = \frac{\sqrt{5}-1}{2}$)

Multiplication Hashing

- **Example**
  - keys are 7-bit binary, $0 \leq k < 128$
  - $m = 8 = 2^3$
  - $A = .1011001$
  - $k = 1101011$

$$\begin{array}{rr} .\ 1\,0\,1\,1\,0\,0\,1 & A \\ 1\,1\,0\,1\,0\,1\,1 & k \\ \hline 1\,0\,0\,1\,0\,1\,0\,.\,\underbrace{0\,1\,1}_{h(k)}\,0\,0\,1\,1 & kA \end{array}$$

Universal Hashing

- **Problem:** For any choice of hash function, there exists a bad set of identifiers– malicious adversary could force poor performance.
- **Solution:**
  - **RANDOMIZE!**
  - Choose hash function at random, *independent* of keys!
  - To do this, create a *set* of hash functions, $\mathcal{H}$, from which h can be randomly selected!

Universal Hashing

- Let $\mathcal{H}$ be a collection of functions mapping U to $\{0, 1, \ldots, m-1\}$.
- **Definition:** $\mathcal{H}$ is **universal** if for all $x, y \in U$ $(x \neq y)$,

$$|\{h \in \mathcal{H} : h(x) = h(y)\}| = \frac{|\mathcal{H}|}{m}$$

  e.g., the # of functions under which x and y collide
- $\mathcal{H}$ is universal if when h is chosen randomly from $\mathcal{H}$, the chance of collision between x and y is $\frac{1}{m}$

Universal Hashing

Theorem

If h is chosen randomly from $\mathcal{H}$ and used to hash n keys into a table T of size m, the expected number of collisions involving any particular key x is less than $\alpha = n/m$.

Proof

- Let C_x = # of collisions of keys in T with x
- Let

$$c_{yz} = \begin{cases} 1 & \text{if } h(y) = h(z) \\ 0 & \text{otherwise} \end{cases}$$

$$C_x = \sum_{y \in T - \{x\}} c_{xy}$$

Universal Hashing

A single pair collides with probability $\frac{1}{m}$;
That is, $\mathrm{E}[c_{xy}] = 1/m$. Therefore,

$$\begin{aligned} \mathrm{E}[C_x] &= \mathrm{E}\left[\sum_{y \in T - \{x\}} c_{xy}\right] \\ &= \sum_{y \in T - \{x\}} \mathrm{E}[c_{xy}] \\ &= \sum_{y \in T - \{x\}} \frac{1}{m} \\ &= \frac{n-1}{m} \\ &< \alpha \end{aligned}$$

- So, the expected number of collisions with x is $< \alpha$.

Constructing a Universal Hash Function

- Let m be prime
- Decompose key x into $r+1$ digits, each with value $\{0, 1, \ldots, m-1\}$; that is,

$$x = < x_0, x_1, \ldots, x_r >, \text{where } 0 \le x_i < m$$

- Pick $< a_0, a_1, \ldots, a_r >$ from $\{0, 1, \ldots, m-1\}$; set

$$h_a(x) = \sum_{i=0}^{r} a_i x_i \bmod m$$

- How big is $\mathcal{H} = \{h_a\}$?

  One h for each choice of the a_i; so

$$|\mathcal{H}| = m^{r+1}$$

Universal Hashing Cont.

Theorem

$\mathcal{H}$ is universal.

Proof

- Let $x = \langle x_0, x_1, \ldots, x_r \rangle$ and $y = \langle y_0, y_1, \ldots, y_r \rangle$ be *distinct* keys.
- x and y differ in at least one digit position.
- Without loss of generality, assume $x_0 \neq y_0$.
- Must show

$$\begin{aligned} |\{h_a : h_a(x) = h_a(y)\}| &= \frac{|\mathcal{H}|}{m} \\ &= m^r \end{aligned}$$

That is, that the # of functions h_a under which x and y collide is $\frac{|\mathcal{H}|}{m} = m^r$

Universal Hashing Cont.

- **Idea:** Show that for any choice of $a_1, a_2, \ldots, a_r$ there is exactly *one* choice of a_0 such that $h_a(x) = h_a(y)$.
  - m is prime $\rightarrow$ $(x_0 - y_0)$ has multiplicative inverse modulo m.
  - There is a unique solution for a_0 modulo m:

$$a_0(x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i(x_i - y_i) \pmod{m}$$

- m^r possible values for $\langle a_0, a_1, \ldots, a_r \rangle$ $\rightarrow$ each pair of keys x and y collides for exactly m^r values of a.
- m^{r+1} possible values for $a \rightarrow x$ and y collide with probability $m^r/m^{r+1} = 1/m$.
- $\mathcal{H}$ is universal.