LECTURE 6

Order Statistics
• Randomized divide and conquer
• Analysis of expected time
• Worst-case linear-time order statistics
• Analysis

Hashing I
• Direct-access tables
• Resolving collisions by chaining
• Choosing hash functions
• Open addressing

Hashing II
• Universal hashing
• Universality theorem
• Constructing a set of universal hash functions
Order statistics

Select the \(i\)th smallest of \(n\) elements (the element with rank \(i\)).

- \(i = 1\): minimum;
- \(i = n\): maximum;
- \(i = \lfloor (n+1)/2 \rfloor\) or \(\lceil (n+1)/2 \rceil\): median.

**Naive algorithm**: Sort and index \(i\)th element.
Worst-case running time = \(\Theta(n \lg n) + \Theta(1)\)
\[= \Theta(n \lg n),\]
using merge sort or heapsort (not quicksort).
Randomized divide-and-conquer algorithm

\textbf{Rand-Select}(A, p, q, i) \triangleright \text{ith smallest of } A[p..q]$
\begin{align*}
\text{if } p &= q \text{ then return } A[p] \\
\text{ } r &\leftarrow \text{Rand-Partition}(A, p, q) \\
\text{ } k &\leftarrow r - p + 1 \quad \triangleright k = \text{rank}(A[r]) \\
\text{if } i &= k \text{ then return } A[r] \\
\text{elseif } i &< k \\
\text{then return } \text{Rand-Select}(A, p, r - 1, i) \\
\text{else return } \text{Rand-Select}(A, r + 1, q, i - k)
\end{align*}
Example

Select the $i = 7$th smallest:

$$\begin{array}{cccccccc}
6 & 10 & 13 & 5 & 8 & 3 & 2 & 11 \\
\end{array}$$

Select the $7 - 4 = 3$rd smallest recursively.

Partition:

$$\begin{array}{cccccccc}
2 & 5 & 3 & 6 & 8 & 13 & 10 & 11 \\
\end{array}$$
Intuition for analysis

(All our analyses today assume that all elements are distinct.)

Lucky:

\[ T(n) = T\left(\frac{9n}{10}\right) + \Theta(n) \]
\[ = \Theta(n) \]

Unlucky:

\[ T(n) = T(n - 1) + \Theta(n) \]
\[ = \Theta(n^2) \]

Worse than sorting!

\[ n^{\log_{10}9} = n^1 = 1 \]

Case 3

arithmetic series
Analysis of expected time

The analysis follows that of randomized quicksort, but it’s a little different.

Let \( T(n) \) = the random variable for the running time of \textsc{Rand-Select} on an input of size \( n \), assuming random numbers are independent.

For \( k = 0, 1, \ldots, n-1 \), define the \textit{indicator random variable}

\[
X_k = \begin{cases} 
1 & \text{if PARTITION generates a } k : n-k-1 \text{ split,} \\
0 & \text{otherwise.}
\end{cases}
\]
Analysis (continued)

To obtain an upper bound, assume that the $i$th element always falls in the larger side of the partition:

$$T(n) = \begin{cases} 
  T(\max \{0, n-1\}) + \Theta(n) & \text{if } 0 : n-1 \text{ split,} \\
  T(\max \{1, n-2\}) + \Theta(n) & \text{if } 1 : n-2 \text{ split,} \\
  \vdots \\
  T(\max \{n-1, 0\}) + \Theta(n) & \text{if } n-1 : 0 \text{ split}, 
\end{cases}$$

$$= \sum_{k=0}^{n-1} X_k \left( T(\max \{k, n-k-1\}) + \Theta(n) \right).$$
Calculating expectation

\[
E[T(n)] = E \left[ \sum_{k=0}^{n-1} X_k \left( T(\max\{k, n-k-1\}) + \Theta(n) \right) \right]
\]

Take expectations of both sides.
Calculating expectation

\[ E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n)) \right] \]

\[ = \sum_{k=0}^{n-1} E[X_k (T(\max\{k, n-k-1\}) + \Theta(n))] \]

Linearity of expectation.
Calculating expectation

\[
E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n)) \right]
\]

\[
= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)]
\]

Independence of \(X_k\) from other random choices.
Calculating expectation

\[ E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n)) \right] \]

\[ = \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)] \]

\[ = \frac{1}{n} \sum_{k=0}^{n-1} E[T(\max\{k, n-k-1\})] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n) \]

Linearity of expectation; \( E[X_k] = 1/n \).
Calculating expectation

\[ E[T(n)] = E \left[ \sum_{k=0}^{n-1} X_k \left( T(\max \{k, n-k-1\}) + \Theta(n) \right) \right] \]

= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max \{k, n-k-1\}) + \Theta(n)]

= \frac{1}{n} \sum_{k=0}^{n-1} E[T(\max \{k, n-k-1\})] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n)

\leq 2 \sum_{k=[n/2]}^{n-1} E[T(k)] + \Theta(n) \quad \text{Upper terms appear twice.}
Hairy recurrence

(But not quite as hairy as the quicksort one.)

\[ E[T(n)] = 2 \sum_{n/2}^{n-1} E[T(k)] + \Theta(n) \]

Prove: \( E[T(n)] \leq cn \) for constant \( c > 0 \).

• The constant \( c \) can be chosen large enough so that \( E[T(n)] \leq cn \) for the base cases.

Use fact: \( \sum_{k=[n/2]}^{n-1} k \leq \frac{3}{8}n^2 \) (exercise).
Substitution method

\[ E[T(n)] \leq \frac{2}{n} \sum_{k=\left\lfloor n/2 \right\rfloor}^{n-1} c_k + \Theta(n) \]

Substitute inductive hypothesis.
Substitution method

\[ E[T(n)] \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} c_k + \Theta(n) \]

\[ \leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n) \]

Use fact.
Substitution method

\[
E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} c_k + \Theta(n)
\]
\[
\leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n)
\]
\[
= cn - \left( \frac{cn}{4} - \Theta(n) \right)
\]

Express as \textit{desired} – \textit{residual}.
Substitution method

\[ E[T(n)] \leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} c k + \Theta(n) \]
\[ \leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n) \]
\[ = cn - \left( \frac{cn}{4} - \Theta(n) \right) \]
\[ \leq cn, \]

if \( c \) is chosen large enough so that \( cn/4 \) dominates the \( \Theta(n) \).
Summary of randomized order-statistic selection

- Works fast: linear expected time.
- Excellent algorithm in practice.
- But, the worst case is very bad: $\Theta(n^2)$.

**Q.** Is there an algorithm that runs in linear time in the worst case?

**A.** Yes, due to Blum, Floyd, Pratt, Rivest, and Tarjan [1973].

**Idea:** Generate a good pivot recursively.
Worst-case linear-time order statistics

**SELECT**(*i*, *n*)

1. Divide the *n* elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively **SELECT** the median *x* of the ⌊*n*/5⌋ group medians to be the pivot.
3. Partition around the pivot *x*. Let *k* = rank(*x*).
4. **if** *i* = *k** then return *x**
   **elseif** *i* < *k**
      then recursively **SELECT** the *i*th smallest element in the lower part
   **else** recursively **SELECT** the (*i*−*k*)th smallest element in the upper part

Same as **RAND-SELECT**
Choosing the pivot
Choosing the pivot

1. Divide the $n$ elements into groups of 5.
Choosing the pivot

1. Divide the $n$ elements into groups of 5. Find the median of each 5-element group by rote.
Choosing the pivot

1. Divide the $n$ elements into groups of 5. Find the median of each 5-element group by rote.
2. Recursively SELECT the median $x$ of the $\lfloor n/5 \rfloor$ group medians to be the pivot.
At least half the group medians are $\leq x$, which is at least $\left\lfloor \frac{n}{5} \right\rfloor / 2 = \left\lfloor \frac{n}{10} \right\rfloor$ group medians.
Analysis

( Assume all elements are distinct. )

At least half the group medians are $\leq x$, which is at least $\left\lfloor \frac{n}{5} / 2 \right\rfloor = \left\lfloor \frac{n}{10} \right\rfloor$ group medians.

• Therefore, at least $3\left\lfloor \frac{n}{10} \right\rfloor$ elements are $\leq x$. 

lesser

greater

26
At least half the group medians are $\leq x$, which is at least $\left\lfloor \frac{n}{5} \right\rfloor / 2 = \left\lfloor \frac{n}{10} \right\rfloor$ group medians.

- Therefore, at least $3 \left\lfloor \frac{n}{10} \right\rfloor$ elements are $\leq x$.
- Similarly, at least $3 \left\lceil \frac{n}{10} \right\rceil$ elements are $\geq x$. 

(Assume all elements are distinct.)
Minor simplification

- For $n \geq 50$, we have $3\left\lfloor n/10 \right\rfloor \geq n/4$.
- Therefore, for $n \geq 50$ the recursive call to \textsc{Select} in Step 4 is executed recursively on $\leq 3n/4$ elements.
- Thus, the recurrence for running time can assume that Step 4 takes time $T(3n/4)$ in the worst case.
- For $n < 50$, we know that the worst-case time is $T(n) = \Theta(1)$. 
Developing the recurrence

<table>
<thead>
<tr>
<th>$T(n)$</th>
<th>$\text{SELECT}(i, n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Theta(n)$</td>
<td>1. Divide the $n$ elements into groups of 5. Find the median of each 5-element group by rote.</td>
</tr>
<tr>
<td>$T(n/5)$</td>
<td>2. Recursively $\text{SELECT}$ the median $x$ of the $\left\lfloor n/5 \right\rfloor$ group medians to be the pivot.</td>
</tr>
<tr>
<td>$\Theta(n)$</td>
<td>3. Partition around the pivot $x$. Let $k = \text{rank}(x)$.</td>
</tr>
<tr>
<td>$T(3n/4)$</td>
<td>4. If $i = k$ then return $x$</td>
</tr>
<tr>
<td></td>
<td>elseif $i &lt; k$</td>
</tr>
<tr>
<td></td>
<td>then recursively $\text{SELECT}$ the $i$th smallest element in the lower part</td>
</tr>
<tr>
<td></td>
<td>else recursively $\text{SELECT}$ the $(i-k)$th smallest element in the upper part</td>
</tr>
</tbody>
</table>
Solving the recurrence

\[ T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{3}{4}n\right) + \Theta(n) \]

Substitution: \[ T(n) \leq \frac{1}{5}cn + \frac{3}{4}cn + \Theta(n) \]

\[ = \frac{19}{20}cn + \Theta(n) \]

\[ = cn - \left(\frac{1}{20}cn - \Theta(n)\right) \leq cn \]

if \( c \) is chosen large enough to handle both the \( \Theta(n) \) and the initial conditions.
Conclusions

• Since the work at each level of recursion is a constant fraction (19/20) smaller, the work per level is a geometric series dominated by the linear work at the root.

• In practice, this algorithm runs slowly, because the constant in front of $n$ is large.

• The randomized algorithm is far more practical.

Exercise: Why not divide into groups of 3?
Symbol-table problem

Symbol table $S$ holding $n$ records:

Operations on $S$:
- \text{\texttt{INSERT}}(S, x)
- \text{\texttt{DELETE}}(S, x)
- \text{\texttt{SEARCH}}(S, k)

How should the data structure $S$ be organized?
**Direct-access table**

**Idea:** Suppose that the keys are drawn from the set \( U \subseteq \{0, 1, \ldots, m-1\} \), and keys are distinct. Set up an array \( T[0 \ldots m-1] \):

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

Then, operations take \( \Theta(1) \) time.

**Problem:** The range of keys can be large:
- 64-bit numbers (which represent 18,446,744,073,709,551,616 different keys),
- character strings (even larger!).
Hash functions

Solution: Use a hash function $h$ to map the universe $U$ of all keys into $\{0, 1, \ldots, m-1\}$:

When a record to be inserted maps to an already occupied slot in $T$, a collision occurs.
Resolving collisions by chaining

- Link records in the same slot into a list.

\[ h(49) = h(86) = h(52) = i \]

**Worst case:**
- Every key hashes to the same slot.
- Access time is \( \Theta(n) \) if \( |S| = n \)
Average-case analysis of chaining

We make the assumption of *simple uniform hashing*:

- Each key $k \in S$ is equally likely to be hashed to any slot of table $T$, independent of where other keys are hashed.

Let $n$ be the number of keys in the table, and let $m$ be the number of slots.

Define the *load factor* of $T$ to be

$$\alpha = \frac{n}{m}$$

= average number of keys per slot.
Search cost

The expected time for an unsuccessful search for a record with a given key is $= \Theta(1 + \alpha)$. 
Search cost

The expected time for an unsuccessful search for a record with a given key is

\[ = \Theta(1 + \alpha). \]

apply hash function
and access slot

search the list
Search cost

The expected time for an *unsuccessful* search for a record with a given key is

\[ = \Theta(1 + \alpha). \]

Expected search time = \( \Theta(1) \) if \( \alpha = O(1) \), or equivalently, if \( n = O(m) \).
Search cost

The expected time for an *unsuccessful* search for a record with a given key is

\[ = \Theta(1 + \alpha). \]

Expected search time \[ = \Theta(1) \] if \( \alpha = O(1) \), or equivalently, if \( n = O(m) \).

A *successful* search has same asymptotic bound, but a rigorous argument is a little more complicated. (See textbook.)
Choosing a hash function

The assumption of simple uniform hashing is hard to guarantee, but several common techniques tend to work well in practice as long as their deficiencies can be avoided.

Desirata:

- A good hash function should distribute the keys uniformly into the slots of the table.
- Regularity in the key distribution should not affect this uniformity.
Division method

Assume all keys are integers, and define
\[ h(k) = k \mod m. \]

**Deficiency:** Don’t pick an \( m \) that has a small divisor \( d \). A preponderance of keys that are congruent modulo \( d \) can adversely affect uniformity.

**Extreme deficiency:** If \( m = 2^r \), then the hash doesn’t even depend on all the bits of \( k \):
- If \( k = 1011000111011010_2 \) and \( r = 6 \), then \( h(k) = 011010_2 \).
Division method (continued)

\[ h(k) = k \mod m. \]

Pick \( m \) to be a prime not too close to a power of 2 or 10 and not otherwise used prominently in the computing environment.

**Annoyance:**
- Sometimes, making the table size a prime is inconvenient.

But, this method is popular, although the next method we’ll see is usually superior.
Multiplication method

Assume that all keys are integers, $m = 2^r$, and our computer has $w$-bit words. Define

$$h(k) = (A \cdot k \mod 2^w) \text{ rsh } (w - r),$$

where \text{ rsh} is the “bitwise right-shift” operator and $A$ is an odd integer in the range $2^{w-1} < A < 2^w$.

• Don’t pick $A$ too close to $2^{w-1}$ or $2^w$.
• Multiplication modulo $2^w$ is fast compared to division.
• The \text{ rsh} operator is fast.
Multiplication method example

\[ h(k) = (A \cdot k \mod 2^w) \text{ rsh } (w - r) \]

Suppose that \( m = 8 = 2^3 \) and that our computer has \( w = 7 \)-bit words:

\[
\begin{array}{c}
\times \\
1011001 \\
1101011 \\
\hline
10010100111
\end{array}
\]

\( h(k) \)

\( A \)

\( k \)
Resolving collisions by open addressing

No storage is used outside of the hash table itself.

• Insertion systematically probes the table until an empty slot is found.

• The hash function depends on both the key and probe number:

  \[ h : U \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}. \]

• The probe sequence \( \langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle \) should be a permutation of \( \{0, 1, \ldots, m-1\} \).

• The table may fill up, and deletion is difficult (but not impossible).
Example of open addressing

Insert key $k = 496$:

0. Probe $h(496,0)$
Example of open addressing

Insert key $k = 496$:

0. Probe $h(496,0)$
1. Probe $h(496,1)$

Collision
Example of open addressing

Insert key $k = 496$:

0. Probe $h(496, 0)$
1. Probe $h(496, 1)$
2. Probe $h(496, 2)$
Example of open addressing

Search for key $k = 496$:

0. Probe $h(496,0)$
1. Probe $h(496,1)$
2. Probe $h(496,2)$

Search uses the same probe sequence, terminating successfully if it finds the key and unsuccessfully if it encounters an empty slot.
Probing strategies

Linear probing:

Given an ordinary hash function $h'(k)$, linear probing uses the hash function

$$h(k,i) = (h'(k) + i) \mod m.$$ 

This method, though simple, suffers from primary clustering, where long runs of occupied slots build up, increasing the average search time. Moreover, the long runs of occupied slots tend to get longer.
Double hashing

Given two ordinary hash functions $h_1(k)$ and $h_2(k)$, double hashing uses the hash function

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \mod m.$$ 

This method generally produces excellent results, but $h_2(k)$ must be relatively prime to $m$. One way is to make $m$ a power of 2 and design $h_2(k)$ to produce only odd numbers.
Analysis of open addressing

We make the assumption of *uniform hashing*:
- Each key is equally likely to have any one of the $m!$ permutations as its probe sequence.

**Theorem.** Given an open-addressed hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$. 
Proof of the theorem

Proof.

• At least one probe is always necessary.
• With probability \( n/m \), the first probe hits an occupied slot, and a second probe is necessary.
• With probability \((n–1)/(m–1)\), the second probe hits an occupied slot, and a third probe is necessary.
• With probability \((n–2)/(m–2)\), the third probe hits an occupied slot, etc.

Observe that \( \frac{n–i}{m–i} < \frac{n}{m} = \alpha \) for \( i = 1, 2, \ldots, n \).
Proof (continued)

Therefore, the expected number of probes is

\[
1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( 1 + \frac{n-2}{m-2} \left( \cdots \left( 1 + \frac{1}{m-n+1} \right) \cdots \right) \right) \right)
\]

\[
\leq 1 + \alpha \left( 1 + \alpha \left( 1 + \alpha \left( \cdots (1 + \alpha) \cdots \right) \right) \right)
\]

\[
\leq 1 + \alpha + \alpha^2 + \alpha^3 + \cdots
\]

\[
= \sum_{i=0}^{\infty} \alpha^i
\]

\[
= \frac{1}{1 - \alpha}
\]

The textbook has a more rigorous proof and an analysis of successful searches.
Implications of the theorem

- If $\alpha$ is constant, then accessing an open-addressed hash table takes constant time.
- If the table is half full, then the expected number of probes is $1/(1-0.5) = 2$.
- If the table is 90% full, then the expected number of probes is $1/(1-0.9) = 10$. 
A weakness of hashing

**Problem:** For any hash function $h$, a set of keys exists that can cause the average access time of a hash table to skyrocket.

- An adversary can pick all keys from \( \{k \in U : h(k) = i\} \) for some slot $i$.

**Idea:** Choose the hash function at random, independently of the keys.

- Even if an adversary can see your code, he or she cannot find a bad set of keys, since he or she doesn’t know exactly which hash function will be chosen.
Universal hashing

**Definition.** Let $U$ be a universe of keys, and let $H$ be a finite collection of hash functions, each mapping $U$ to $\{0, 1, \ldots, m-1\}$. We say $H$ is *universal* if for all $x, y \in U$, where $x \neq y$, we have $|\{h \in H : h(x) = h(y)\}| = |H|/m$.

That is, the chance of a collision between $x$ and $y$ is $1/m$ if we choose $h$ randomly from $H$. 
Universality is good

Theorem. Let $h$ be a hash function chosen (uniformly) at random from a universal set $H$ of hash functions. Suppose $h$ is used to hash $n$ arbitrary keys into the $m$ slots of a table $T$. Then, for a given key $x$, we have

$$E[\#\text{collisions with } x] < \frac{n}{m}.$$
Proof of theorem

**Proof.** Let $C_x$ be the random variable denoting the total number of collisions of keys in $T$ with $x$, and let

$$c_{xy} = \begin{cases} 
1 & \text{if } h(x) = h(y), \\
0 & \text{otherwise.}
\end{cases}$$

**Note:** $E[c_{xy}] = 1/m$ and $C_x = \sum_{y \in T \setminus \{x\}} c_{xy}$. 
Proof (continued)

\[ E[C_x] = E \left[ \sum_{y \in T - \{x\}} c_{xy} \right] \]

- Take expectation of both sides.
Proof (continued)

\[ E[C_x] = \mathbb{E} \left[ \sum_{y \in T \setminus \{x\}} c_{xy} \right] \]

\[ = \sum_{y \in T \setminus \{x\}} \mathbb{E}[c_{xy}] \]

- Take expectation of both sides.
- Linearity of expectation.
Proof (continued)

\[
E[C_x] = E\left[ \sum_{y \in T - \{x\}} c_{xy} \right]
= \sum_{y \in T - \{x\}} E[c_{xy}]
= \sum_{y \in T - \{x\}} 1/m
\]

- Take expectation of both sides.
- Linearity of expectation.
- \(E[c_{xy}] = 1/m\).
Proof (continued)

\[ E[C_x] = E \left[ \sum_{y \in T - \{x\}} c_{xy} \right] \]

\[ = \sum_{y \in T - \{x\}} E[c_{xy}] \]

\[ = \sum_{y \in T - \{x\}} 1/m \]

\[ = \frac{n-1}{m} \]

• Take expectation of both sides.
• Linearity of expectation.
• \( E[c_{xy}] = 1/m \).
• Algebra.
Constructing a set of universal hash functions

Let $m$ be prime. Decompose key $k$ into $r + 1$ digits, each with value in the set $\{0, 1, \ldots, m-1\}$. That is, let $k = \langle k_0, k_1, \ldots, k_r \rangle$, where $0 \leq k_i < m$.

Randomized strategy:

Pick $a = \langle a_0, a_1, \ldots, a_r \rangle$ where each $a_i$ is chosen randomly from $\{0, 1, \ldots, m-1\}$.

Define $h_a(k) = \sum_{i=0}^{r} a_i k_i \mod m$. 

How big is $H = \{h_a\}$? $|H| = m^r + 1$. 

REMEMBER THIS!
Universality of dot-product hash functions

**Theorem.** The set $\mathcal{H} = \{h_a\}$ is universal.

**Proof.** Suppose that $x = \langle x_0, x_1, \ldots, x_r \rangle$ and $y = \langle y_0, y_1, \ldots, y_r \rangle$ be distinct keys. Thus, they differ in at least one digit position, wlog position 0.

For how many $h_a \in \mathcal{H}$ do $x$ and $y$ collide?

We must have $h_a(x) = h_a(y)$, which implies that

$$\sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \pmod{m}.$$
Proof (continued)

Equivalently, we have

\[ \sum_{i=0}^{r} a_i (x_i - y_i) \equiv 0 \pmod{m} \]

or

\[ a_0 (x_0 - y_0) + \sum_{i=1}^{r} a_i (x_i - y_i) \equiv 0 \pmod{m}, \]

which implies that

\[ a_0 (x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i (x_i - y_i) \pmod{m}. \]
Fact from number theory

**Theorem.** Let $m$ be prime. For any $z \in \mathbb{Z}_m$ such that $z \neq 0$, there exists a unique $z^{-1} \in \mathbb{Z}_m$ such that

$$z \cdot z^{-1} \equiv 1 \pmod{m}.$$

**Example:** $m = 7$.

<table>
<thead>
<tr>
<th>$z$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$z^{-1}$</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>
Back to the proof

We have

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

and since \( x_0 \neq y_0 \), an inverse \( (x_0 - y_0)^{-1} \) must exist, which implies that

\[ a_0 \equiv \left( - \sum_{i=1}^{r} a_i(x_i - y_i) \right) \cdot (x_0 - y_0)^{-1} \pmod{m}. \]

Thus, for any choices of \( a_1, a_2, \ldots, a_r \), exactly one choice of \( a_0 \) causes \( x \) and \( y \) to collide.
**Q.** How many $h_a$’s cause $x$ and $y$ to collide?

**A.** There are $m$ choices for each of $a_1, a_2, \ldots, a_r$, but once these are chosen, exactly one choice for $a_0$ causes $x$ and $y$ to collide, namely

$$a_0 = \left( \left( - \sum_{i=1}^{r} a_i (x_i - y_i) \right) \cdot (x_0 - y_0)^{-1} \right) \mod m.$$ 

Thus, the number of $h_a$’s that cause $x$ and $y$ to collide is $m^r \cdot 1 = m^r = |H|/m$. 

---
Perfect hashing

Given a set of \( n \) keys, construct a static hash table of size \( m = O(n) \) such that SEARCH takes \( \Theta(1) \) time in the worst case.

**Idea:** Two-level scheme with universal hashing at both levels.

*No collisions at level 2!*