CMP461: Algorithms

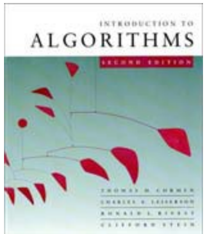
Lecture 05: Hashing I

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Fall 2013

Agenda

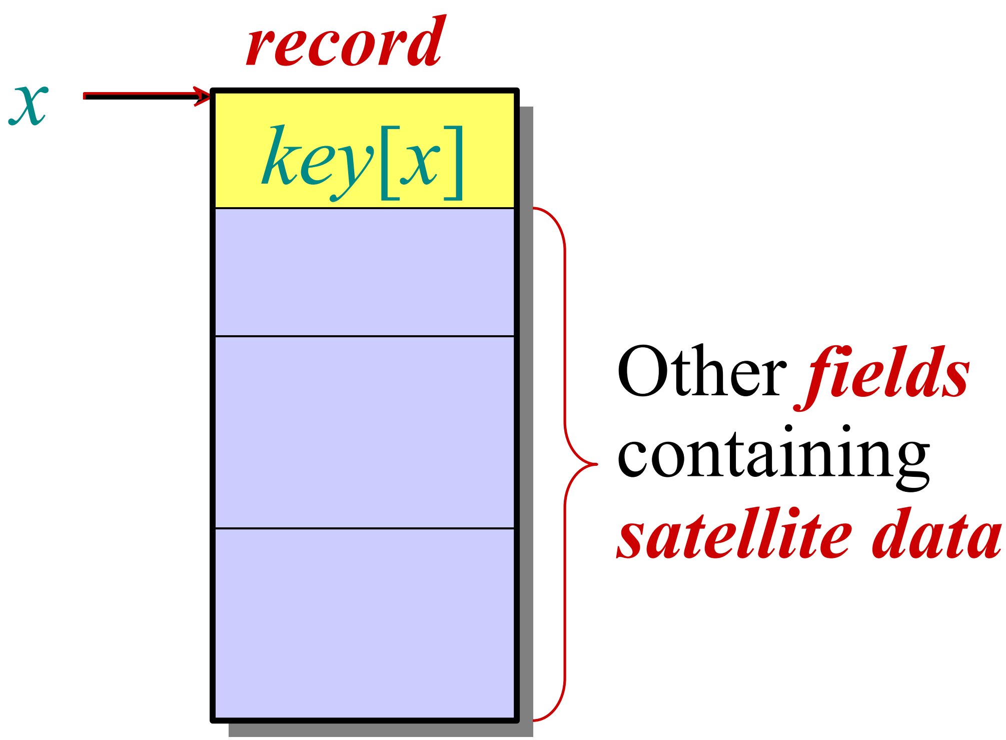
- Direct Access Tables
- Hash Tables
  - Hash Functions
  - Chaining

Acknowledgment
A lot of slides adapted from the slides of Erik Demaine and Charles Leiserson



Symbol-table problem

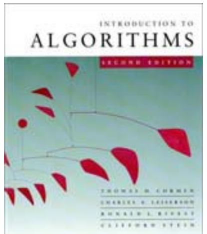
Symbol table S holding n ***records***:



Operations on S:

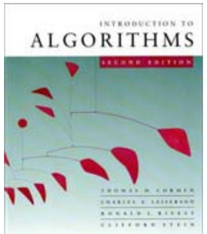
- INSERT(S, x)
- DELETE(S, x)
- SEARCH(S, k)

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.

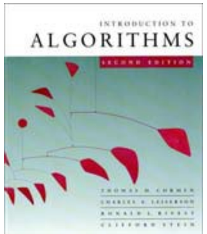


Direct-access table

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 $T[0 \,..\, m-1]$:

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

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



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 $T[0 \,..\, m-1]$:

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

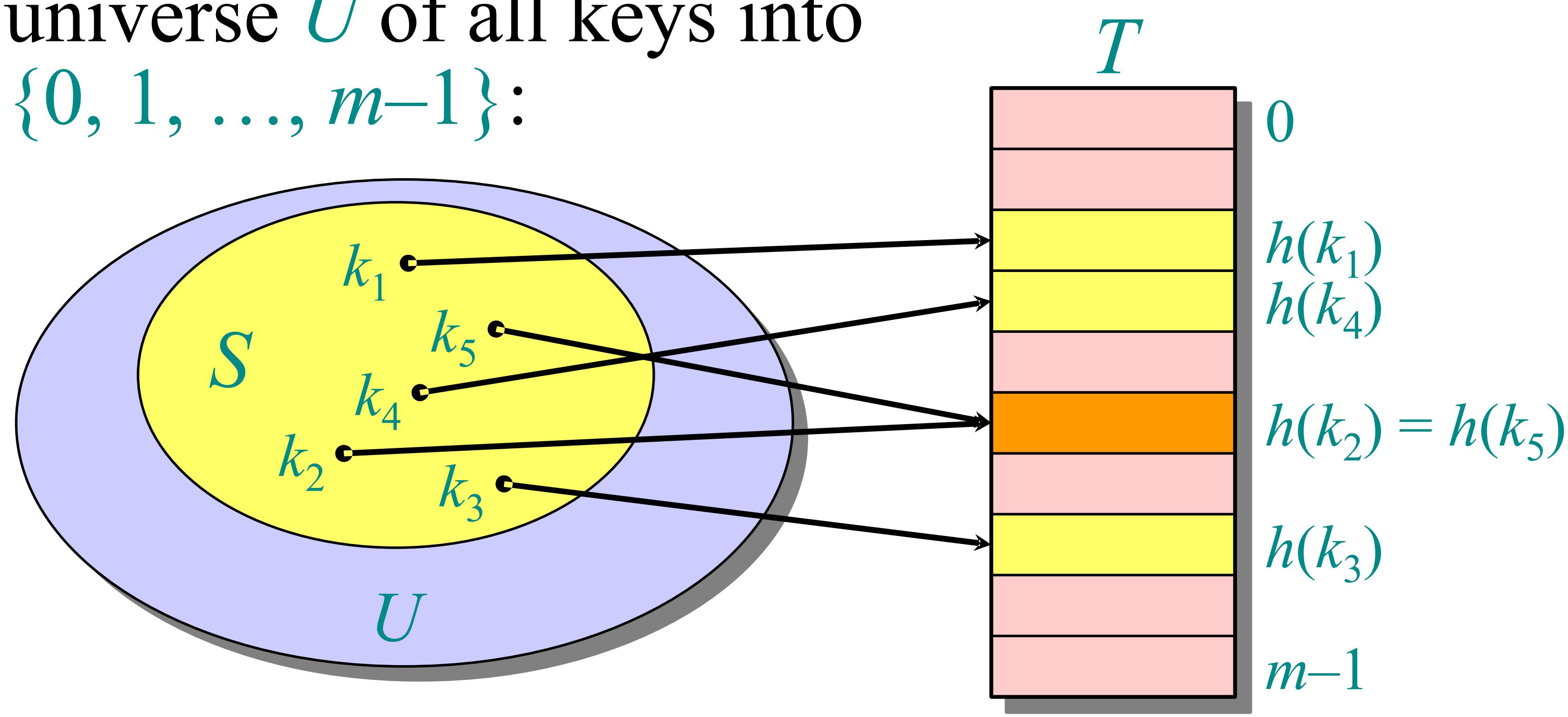
Then, operations take $\Theta(1)$ time.

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

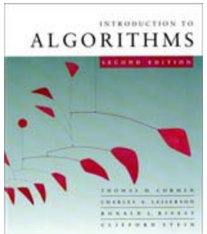
Solution: Use a ***hash function*** h to map the universe U of all keys into $\{0, 1, \ldots, m-1\}$:



When a record to be inserted maps to an already occupied slot in T, a ***collision*** occurs.

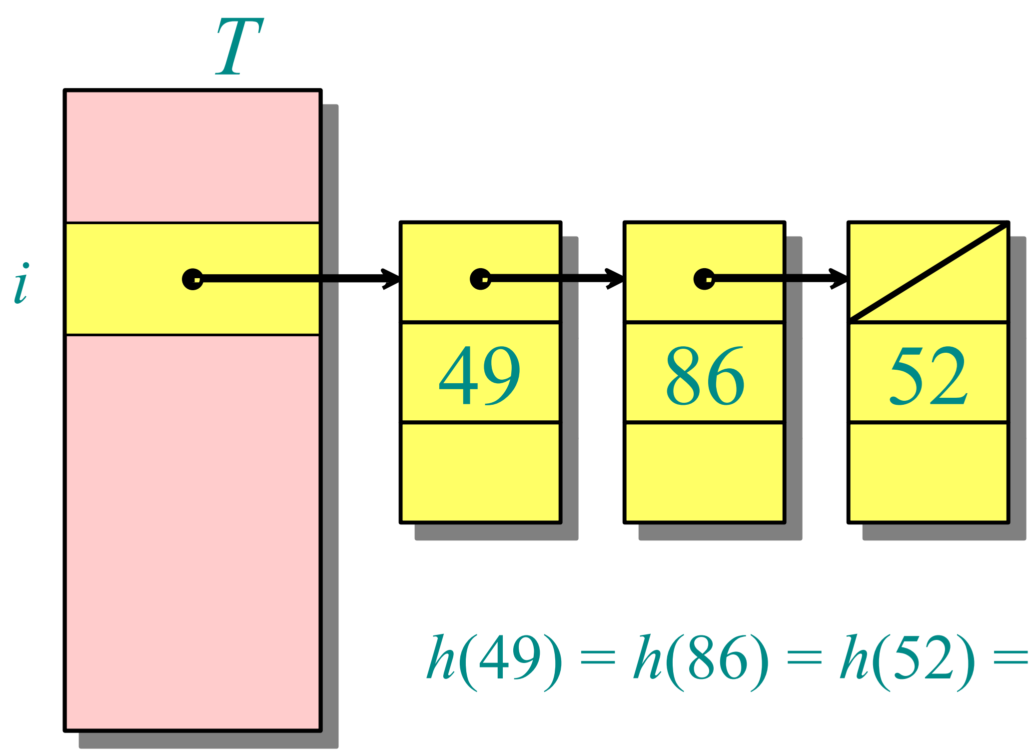
Collision Resolution

- We can resolve the collision by:
  - Chaining
  - Open Addressing

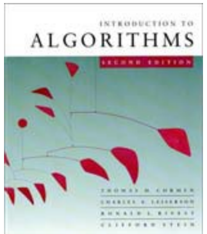


Resolving collisions by chaining

- Link records in the same slot into a list.

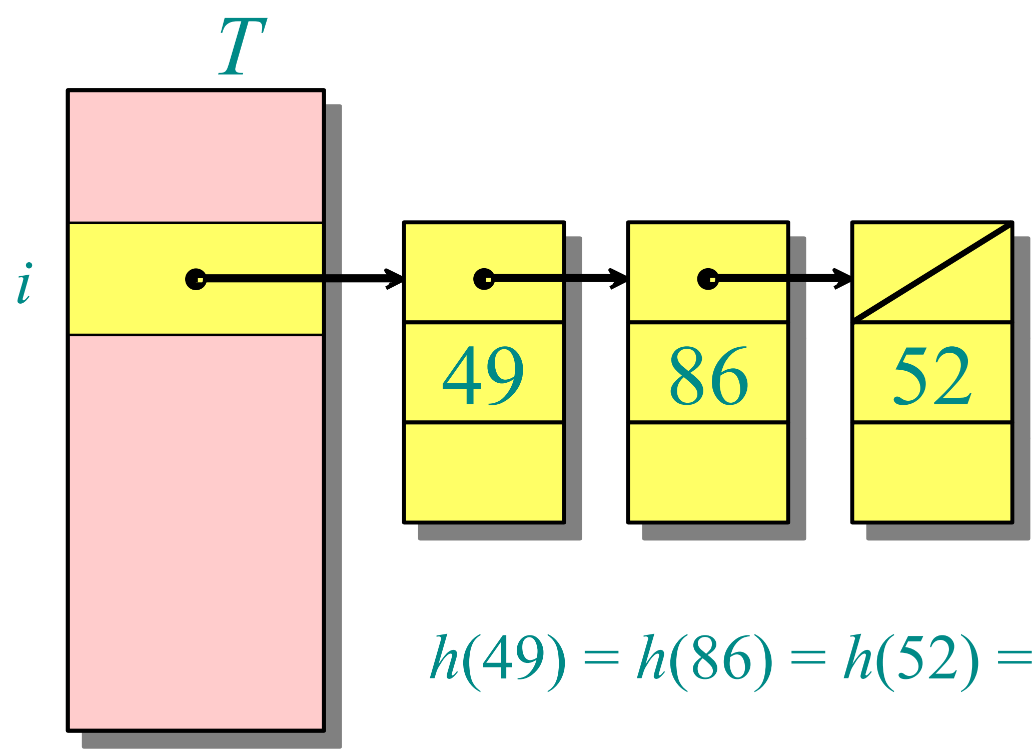


$h(49) = h(86) = h(52) = i$



Resolving collisions by chaining

- Link records in the same slot into a list.



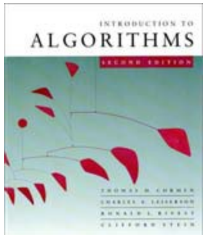
$h(49) = h(86) = h(52) = i$

Insertion: go to $h(key)$ then insert at head of linked list $\Theta(1)$

Search: go to $h(key)$ and find key in linked list

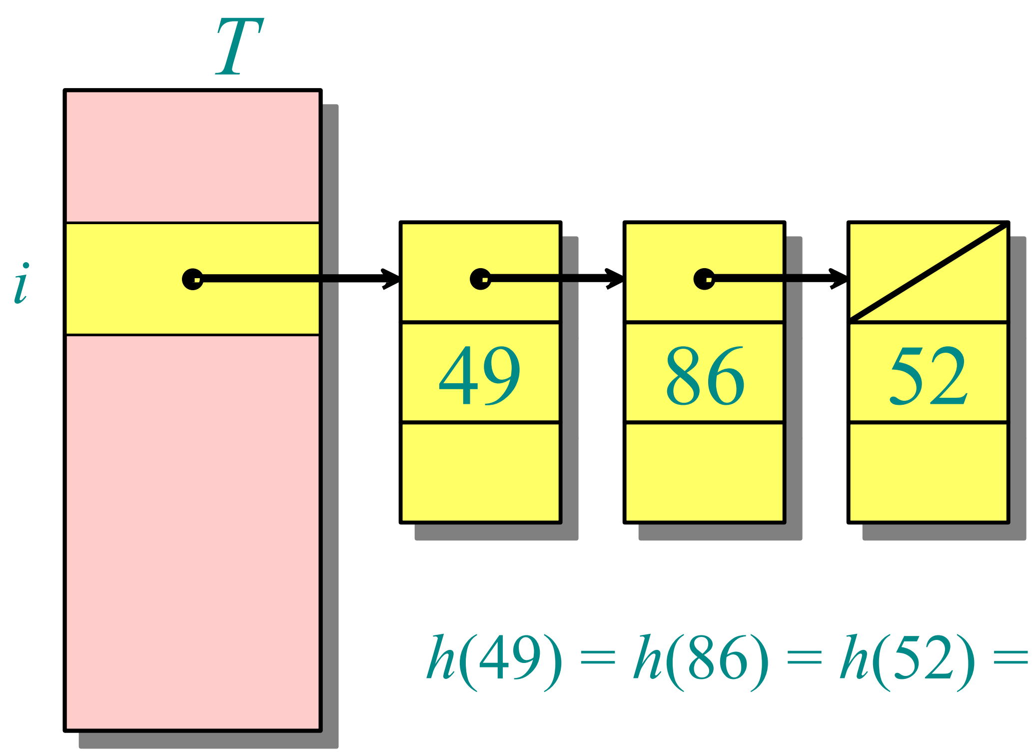
Delete: go to $h(key)$ and find key in linked list then remove

Search and Delete depend on length of linked list!



Resolving collisions by chaining

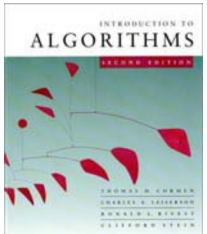
- Link records in the same slot into a list.



$h(49) = h(86) = h(52) = i$

Worst case:

- Every key hashes to the same slot.
- Access time = $\Theta(n)$ if $|S| = n$



Average-case analysis of chaining

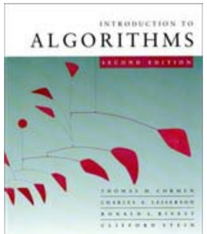
We make the assumption of ***simple uniform hashing:***

- 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.

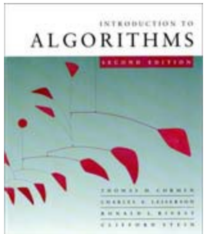
Define the ***load factor*** of T to be

$$\alpha = n/m$$
$$= \text{average number of keys per slot.}$$



Search cost

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

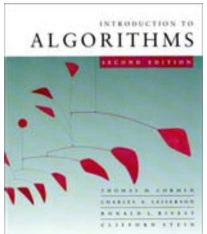


Search cost

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

search the list

apply hash function and access slot



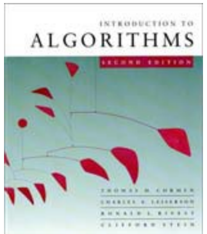
Search cost

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

search the list

apply hash function and access slot

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



Search cost

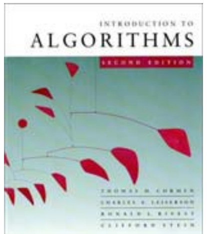
The expected time for an ***unsuccessful*** search for a record with a given key is $= \Theta(1 + \alpha)$.

search the list

apply hash function and access slot

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.)

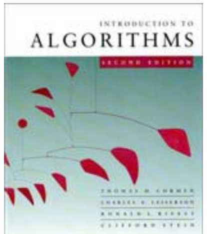


Choosing a hash function

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.

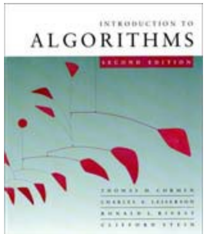


Division method

Assume all keys are integers, and define

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

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.



Division method

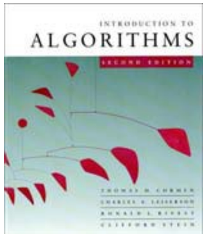
Assume all keys are integers, and define
$$h(k) = k \bmod m.$$

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 $k = 1011000111\underbrace{011010}_{h(k)}{}_2$ and $r = 6$, then $h(k) = 011010_2$.

October 3, 2005 *Copyright © 2001-5 by Erik D. Demaine and Charles E. Leiserson* L7.12



Division method (continued)

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

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.

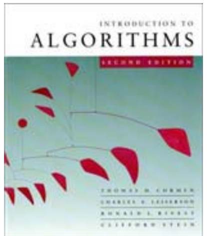
But, this method is popular, although the next method we'll see is usually superior.

Multiplication Method

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

where $0 < A < 1$

- Extracts the fractional part of kA and then multiplies by m
- When m is a power of 2 (e.g. 2^r), it is fast and efficient to implement



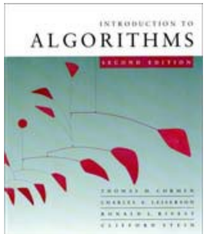
Multiplication method

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

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

where 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 i.e. least significant w bits.
- The rsh operator is fast.

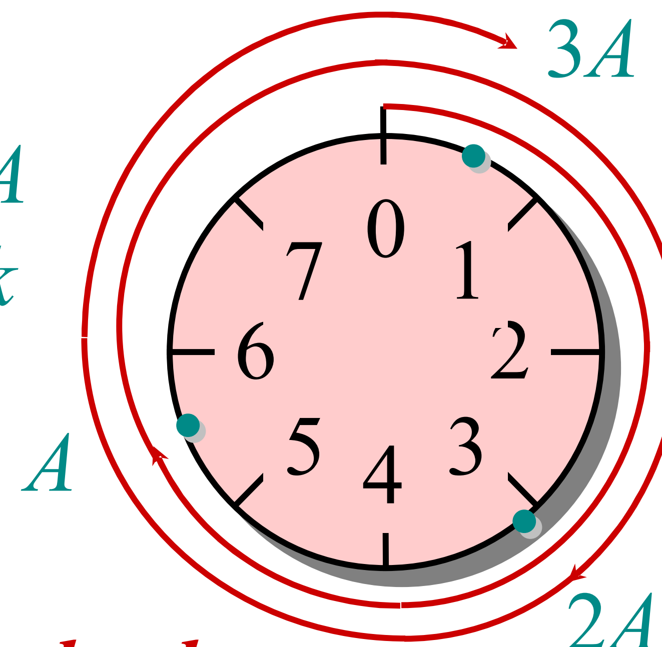


Multiplication method example

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

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

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



Modular wheel

Recap

- Direct Access Tables
- Hash Tables
  - Chaining
  - Hash Functions
- Next:
  - Open Addressing
  - Universal Hashing
  - Perfect Hashing