12 HASH TABLES

DYNAMIC SET IS A GENERAL TERM FOR ANY DATA STRUCTURE REPRESENTING A SET WHICH MAY GROW, SHRINK, OR OTHERWISE CHANGE OVER TIME.

EXAMPLES INCLUDE STACKS, QUEUES, LISTS, HEAPS, PRIORITY QUEUES, TREES, I.E. EVERYTHING WE ARE STUDYING.

$$S = \{ \ldots \underbrace{(\cdot, \cdot, \cdot}_{\text{SATELLITE DATA}}, \text{key}) \ldots \}$$

A DICTIONARY IS A DYNAMIC SET WHICH SUPPORTS THE OPERATIONS

Insert(S, x) : INSERT ELEMENT (POINTED TO BY) x INTO S.

Delete(S, x) : DELETE ELEMENT (POINTED TO BY) x FROM S.

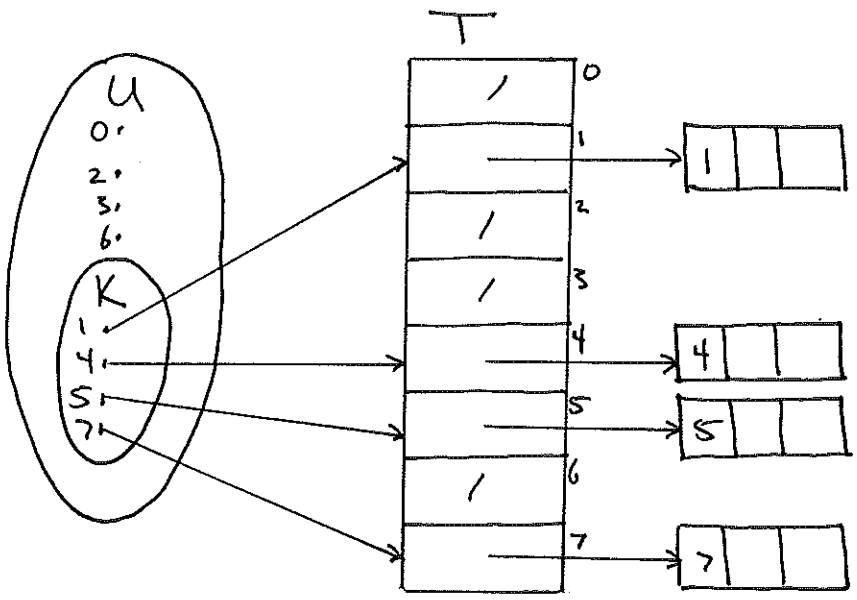
Search(S, k) : RETURN A POINTER x TO AN ELEMENT IN S SUCH THAT $\text{key}[x] = k$, OR NIL IF NO SUCH ELEMENT EXISTS.

SUPPOSE AN APPLICATION NEEDS A DICTIONARY IN WHICH EACH ELEMENT HAS A KEY DRAWN FROM A UNIVERSE

$$U = \{0, 1, \ldots, m-1\}$$

OF POSSIBLE KEYS. WE ASSUME NO TWO ELEMENTS HAVE THE SAME KEY, BUT NOT ALL KEYS ARE NECESSARILY IN USE.

ONE WAY TO PROCEED IS TO HAVE AN ARRAY $T[0 \ldots (m-1)]$ WHOSE INDICES ARE JUST THE KEYS.



AN ARRAY USED IN THIS WAY IS CALLED A <u>DIRECT ADDRESS TABLE</u>.

THE DICTIONARY OPERATIONS ARE TRIVIAL TO IMPLEMENT, AND RUN IN $O(1)$ TIME (WORST CASE). FOR INSTANCE

Search (T, k)
1.) return T[k]

THIS WORKS FINE IF $|U|$ IS NOT TOO LARGE. IF $|U|$ IS LARGE, AND $|K|$ IS SMALL RELATIVE TO $|U|$, THEN THIS ARRANGEMENT WASTES ALOT OF SPACE.

ONE ALTERNATIVE IS TO USE A LINKED LIST INSTEAD OF AN ARRAY. THE Search OPERATION WOULD THEN TAKE $\Theta(n)$ TIME (AVERAGE CASE), WHICH IS CONSIDERED SLOW.

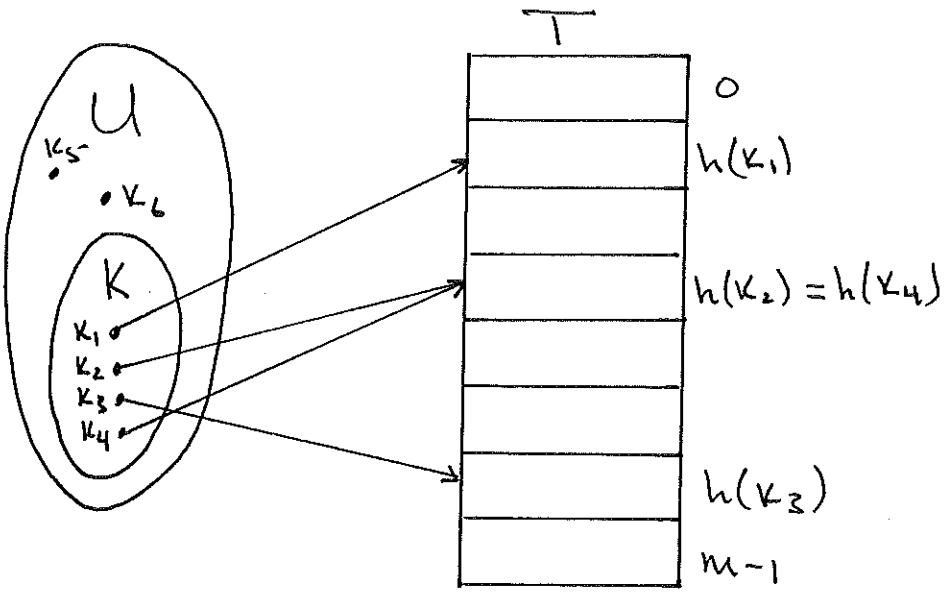
A THIRD ALTERNATIVE IS TO USE A HASH TABLE, WHERE STORAGE REQUIREMENTS ARE $\Theta(|K|)$ AS IN A LINKED LIST, AND Search TAKES $O(1)$ TIME (NOW IN AVERAGE CASE)

A HASH TABLE IS AN ARRAY $T[0 \ldots (m-1)]$ IN WHICH AN ELEMENT WITH Key k IS STORED, NOT IN $T[k]$, BUT IN $T[h(k)]$.

HERE $h : U \to \{0, 1, \ldots, m-1\}$ IS CALLED A HASH FUNCTION. WE SAY KEY k HASHES TO SLOT $h(k)$, AND THAT $h(k)$ IS THE HASH VALUE OF k.

IT MAY BE HOWEVER THAT TWO (OR MORE) KEYS HASH TO THE SAME SLOT: $h(k_1) = h(k_2)$. THIS IS CALLED A *collision*.

IN OTHER WORDS h IS NOT INJECTIVE, SINCE THE WHOLE POINT IS TO SAVE SPACE $m < |U|$.

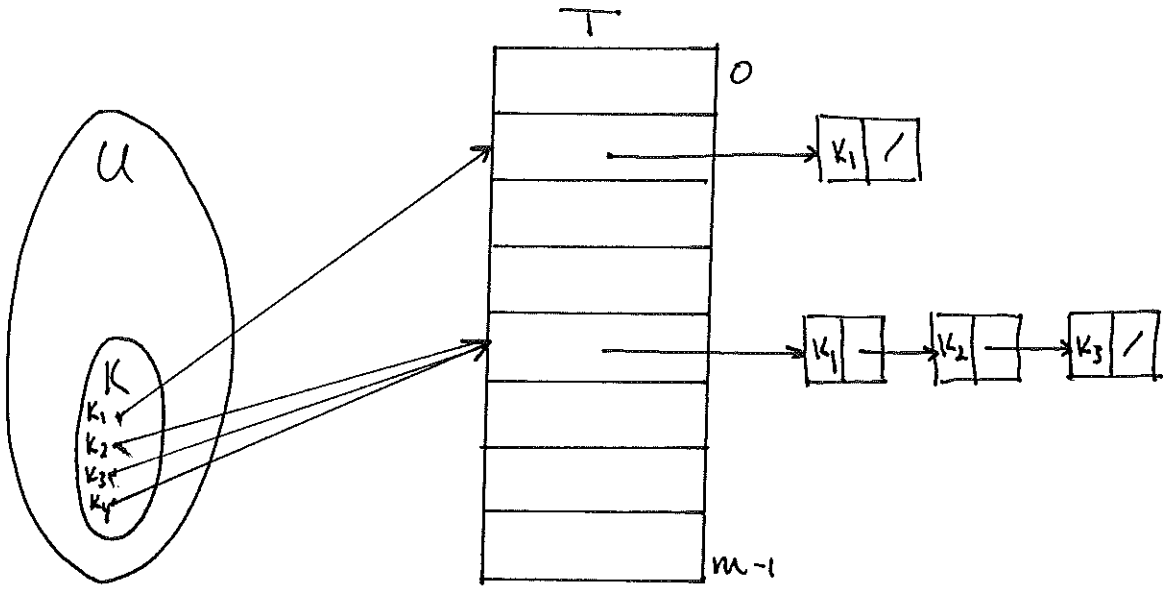


FORTUNATELY THERE ARE EFFECTIVE METHODS FOR RESOLVING *collisions*.

EVEN SO, THE FUNCTION h SHOULD BE CHOSEN IN SUCH A WAY AS TO MINIMIZE THE PROBABILITY THAT TWO RANDOM KEYS COLLIDE. IN OTHER WORDS, h "DISTRIBUTES" THE PROBABILITY OF A COLLISION EVENLY ACROSS THE ARRAY INDICES $\{0, \ldots, m-1\}$.

h IS THEREFORE DESIGNED TO LOOK "RANDOM" IN SOME SENSE, HENCE THE NAME HASH FUNCTION.

<u>CHAINING</u> IS A COLLISION RESOLUTION METHOD IN WHICH ALL ELEMENTS WHOSE KEYS HASH TO THE SAME SLOT ARE PLACED IN A LINKED LIST.



THE DICTIONARY OPERATIONS ARE THEN IMPLEMENTED BY CALLING LIST OPERATIONS.

Insert (T, x)
    insert x into item of list $T[h(key[x])]$

Delete (T, x)
    delete x from list $T[h(key[x])]$

Search (T, k)
    Search for an element x such that $key[x] = k$ in list $T[h(k)]$.

Insert takes $O(1)$ time (worst case). The time to run Search depends on the number of elements in the list $T[h(k)]$, which depends on h. Delete takes $O(1)$ time if the lists are doubly linked.

We will analyze the run time of Search under chaining.

Given a hash table $T[0 \ldots (m-1)]$ with m slots which stores $n = |K|$ elements, define the <u>load factor</u> α to be

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

THUS α IS THE AVERAGE NUMBER OF ELEMENTS STORED IN ANY LIST.

AS NOTED EARLIER, THE PERFORMANCE OF SEARCH DEPENDS DEPENDS ON THE CHOICE OF HASH FUNCTION $h : U \to \{0, .., m-1\}$.

WE MAKE THE FOLLOWING ASSUMPTION ABOUT h.

- ANY RANDOMLY SELECTED KEY IS EQUALLY LIKELY TO HASH INTO ANY OF THE m SLOTS. I.E.

$$\text{PROB}(h(K) = i) = \frac{1}{m} \quad \text{FOR } 1 \le i < m .$$

WE CALL THIS THE ASSUMPTION OF <u>SIMPLE UNIFORM HASHING</u>.

IN THE LANGUAGE OF PROBABILITY THEORY, U IS THE SAMPLE SPACE AND $h : U \to \{0..,m-1\}$ IS A DISCRETE RANDOM VARIABLE WHICH IS UNIFORMLY DISTRIBUTED, I.E. THE PROBABILITY DENSITY FUNCTION OF h IS THE CONSTANT $\frac{1}{m}$.

NOTE WE DO NOT ASSUME THAT ANY TWO KEYS ARE EQUALLY LIKELY TO BE SELECTED. SOME KEYS MAY BE MORE LIKELY TO BE USED THAN OTHERS.

THUS TO CONFORM TO THE ASSUMPTION OF SIMPLE UNIFORM HASHING, h MUST BE CHOSEN WITH THE DISTRIBUTION OF KEYS IN MIND.

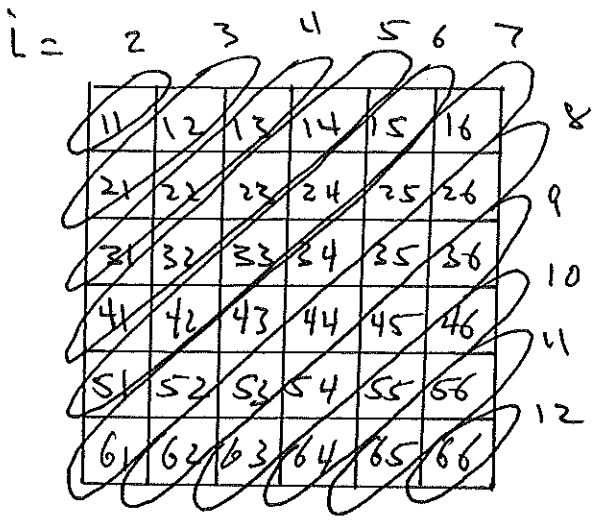
TO ANALYZE Search IN THE AVERAGE CASE WE RECALL

DEFN

IF $f(i)$ IS A FUNCTION DEFINED ON A SEQUENCE OF RANDOM EVENTS i, THE EXPECTED VALUE OR AVERAGE VALUE OF f IS

$$E[f] = \sum_i f(i) \cdot (\text{PROB. OF EVENT } i)$$

EX. TWO FAIR DIE ARE THROWN AND THE SUM i IS RECORDED. LET $f(i) = i$. THEN



$$E[f] = 2 \cdot \frac{1}{36} + 3 \cdot \frac{2}{36} + \cdots + 7 \cdot \frac{6}{36} + \cdots 12 \cdot \frac{1}{36} = 7$$

WE CONSIDER TWO CASES FOR THE RUN TIME OF SEARCH.

CASE 1: THE SEARCH IS UNSUCCESSFUL.

LET $f(i)$ BE THE NUMBER OF KEYS STORED IN SLOT i, $(0 \le i \le m-1)$. THEN

$$\sum_{i=0}^{m-1} f(i) = n,$$

THE NUMBER OF KEYS IN USE. THE AVERAGE LIST LENGTH IS THEN

$$E[f] = \sum_{i=0}^{m-1} f(i) \cdot P(h(k) = i) = \sum_{i=0}^{m-1} \frac{f(i)}{m} = \frac{n}{m} = \alpha.$$

THE RUN TIME OF AN UNSUCCESSFUL SEARCH IS THEREFORE PROPORTIONAL TO α.

CASE 2: THE SEARCH is SUCCESSFUL.

AGAIN LET $f(i)$ BE THE NUMBER OF KEYS STORED IN SLOT i. $(0 \le i \le m-1)$. ON AVERAGE THE KEY SEARCHES FOR WILL BE FOUND HALFWAY DOWN THE LIST.

THE AVERAGE NUMBER OF KEYS EXAMINED IN SLOT i IS THEREFORE $f(i)/2$.

THE NUMBER OF KEYS EXAMINED, AVERAGED OVER ALL SLOTS is

$$E\left[\frac{f}{2}\right] = \sum_{i=0}^{m-1} \frac{f(i)}{2} \cdot \frac{1}{m} = \frac{n}{2m} = \frac{\alpha}{2}.$$

THE RUN TIME OF A SUCCESSFUL SEARCH is ALSO PROPORTIONAL TO α.

IT TAKES TIME $\Theta(1)$ TO COMPUTE $h(k)$. THEREFORE THE AVERAGE TIME TO RUN Search is IN EITHER CASE

$$\Theta(1) + \Theta(\alpha) = \Theta(1+\alpha).$$

IF IT HAPPENS THAT $n = \Theta(m)$, THEN $\alpha = \frac{n}{m} = \Theta(1)$, AND SEARCH RUNS IN CONSTANT TIME, ON AVERAGE.

A GOOD HASH FUNCTION WILL SATISFY, AT LEAST APPROXIMATELY, THE ASSUMPTION OF SIMPLE UNIFORM HASHING. NOTE THAT

$$\text{PROB.}\left(h(K) = i\right) = \sum_{\{K : h(K) = i\}} P(K)$$

WHERE $P(K)$ IS THE PROBABILITY THAT KEY K IS SELECTED. SIMPLE UNIFORM HASHING CAN THEN BE EXPRESSED AS

$$\sum_{\{K : h(K) = i\}} P(K) = \frac{1}{m} \qquad (0 \le i \le m-1).$$

USUALLY THE PROBABILITY FUNCTION P IS UNKNOWN, THOUGH SOME QUALITATIVE INFORMATION MAY BE AVAILABLE.

IF IT HAPPENS THAT THE KEYS ARE UNIFORMLY DISTRIBUTED OVER U, I.E.

$$P(K) = \frac{1}{|U|} \qquad (\forall K \in U)$$

THEN ANY FUNCTION WHICH PLACES APPROXIMATELY $|U|/m$ KEYS IN EACH SLOT WILL SATISFY (APPROXIMATELY) THE SIMPLE UNIFORM HASHING CONDITION.