Space and Time Tradeoffs

Hashing

- A very efficient method for implementing a *dictionary*
- A set with the operations:
  - find (lookup)
  - insert
  - delete

- Important applications:
  - symbol tables
  - databases (*extendible hashing*)

- Files comprise of records and records comprise of field
  - *key* → at least one field used to identify entities (records)

Hash tables and hash functions

- *hashing* → map n keys of a given file into a table of size m
- *hash table* → the table of size m
- *hash function* → a predefined function used to do the mapping
  $h(K)$ → location (cell) in the hash table for record with key k

Example: student records
Key: SSN
Hash function: $h(K) = K \bmod m$
where m is some integer (typically, prime)
If m = 1000, where is record with SSN= 314159265 stored?

Collisions

collision→ two or more keys assigned to the same cell of the hash table

$$h(K_1) = h(K_2)$$

- There are collisions if hash table size m is smaller than number of records n
- Worst case → all records being assigned to the same cell
- Every hash schema must have a collision resolution mechanism

Collisions

- Good hash functions result in fewer collisions but some collisions should be expected

- Hash function should:
  - Be easy to compute
  - Distribute keys among hash table as evenly as possible

- Hash table should be:
  - Not excessively large compared to the number of keys
  - Large enough not to result in a lot of conflicts

- Select appropriate hash table size and a good hash function

Collisions

Two principal hashing schemes handle collisions differently:

- *Open hashing*
  - each cell is a header of linked list of all keys hashed to it
- *Closed hashing*
  - one key per cell
  - in case of collision, finds another cell by
    - *linear probing:* use next free bucket
    - *double hashing:* use second hash function to compute increment

Open hashing (Separate chaining)

- Keys are stored in linked lists attached to cells of a hash table

Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

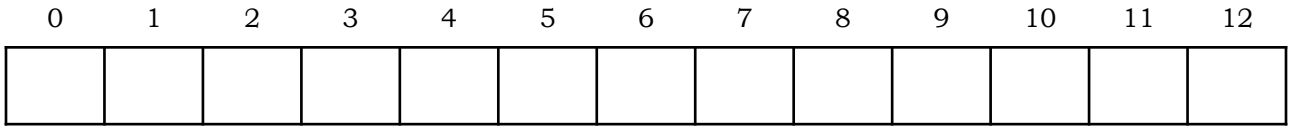
Open hashing (Separate chaining)

- Keys are stored in linked lists attached to cells of a hash table

Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13



Open hashing (Separate chaining)

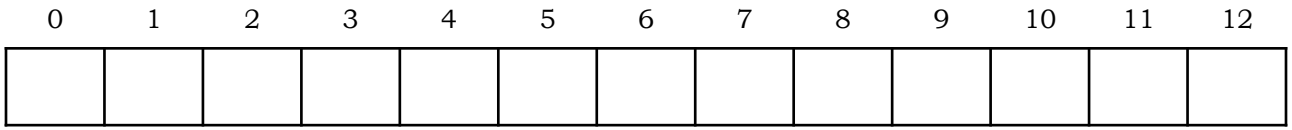
- Keys are stored in linked lists attached to cells of a hash table

Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26



Open hashing (Separate chaining)

- Keys are stored in linked lists attached to cells of a hash table

Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses								

0	1	2	3	4	5	6	7	8	9	10	11	12

Open hashing (Separate chaining)

- Keys are stored in linked lists attached to cells of a hash table

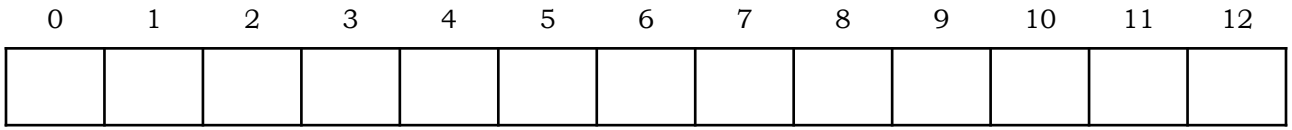
Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12



Open hashing (Separate chaining)

- Keys are stored in linked lists attached to cells of a hash table

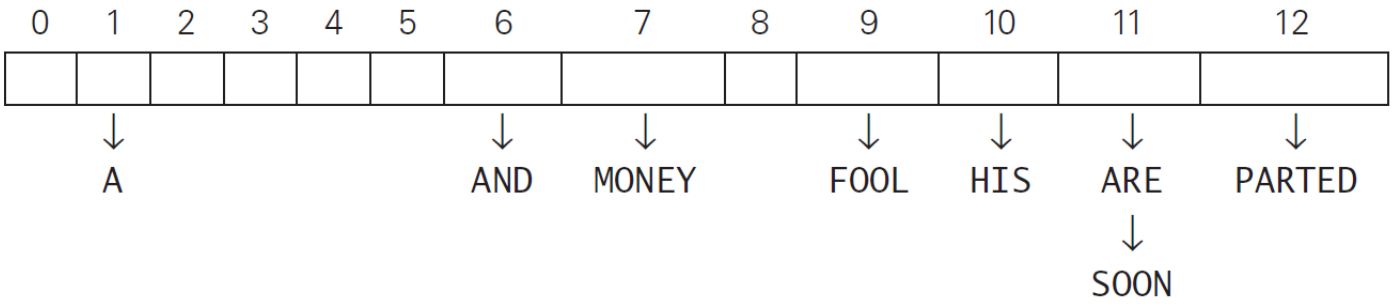
Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

$h(A) = 1\ mod\ 13 = 1$
$h(FOOL) = (6 + 15 + 15 + 12) mod\ 13 = 9$
$h(ARE) = (1 + 18 + 5) mod\ 13 = 11$
$h(SOON) = (19 + 15 + 15 + 14) mod\ 13 = 11$

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12



Open hashing (Separate chaining)

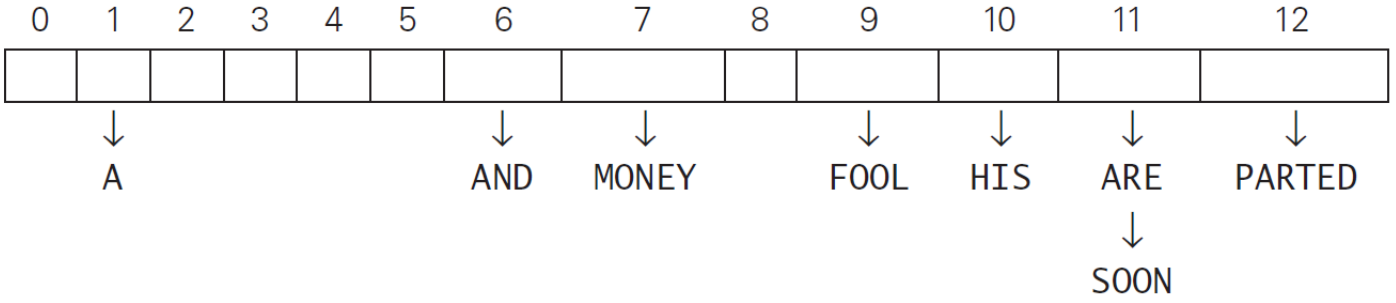
- How to search?
- Calculate hash function for key
- Lookup in the linked list assigned to that hash table cell

Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup SOON:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26



Open hashing (Separate chaining)

- How to search?
- Calculate hash function for key
- Lookup in the linked list assigned to that hash table cell

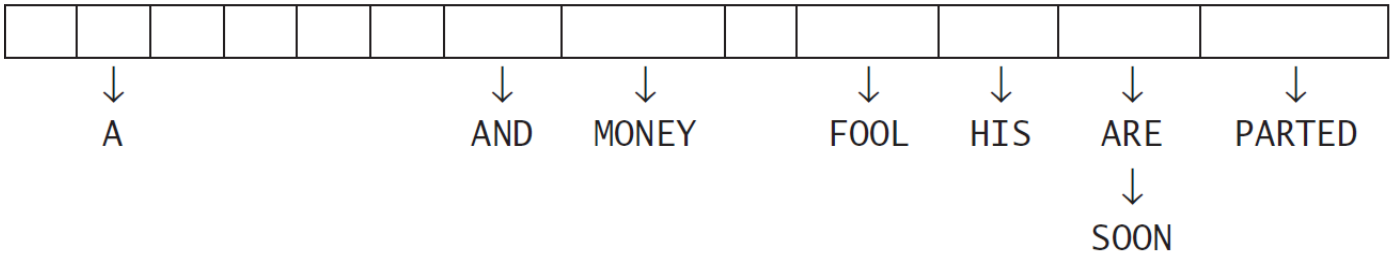
Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup SOON:

$h(SOON) = (19 + 15 + 15 + 14) mod\ 13 = 11$

found in linked list assigned to 11 → successfull search



Open hashing (Separate chaining)

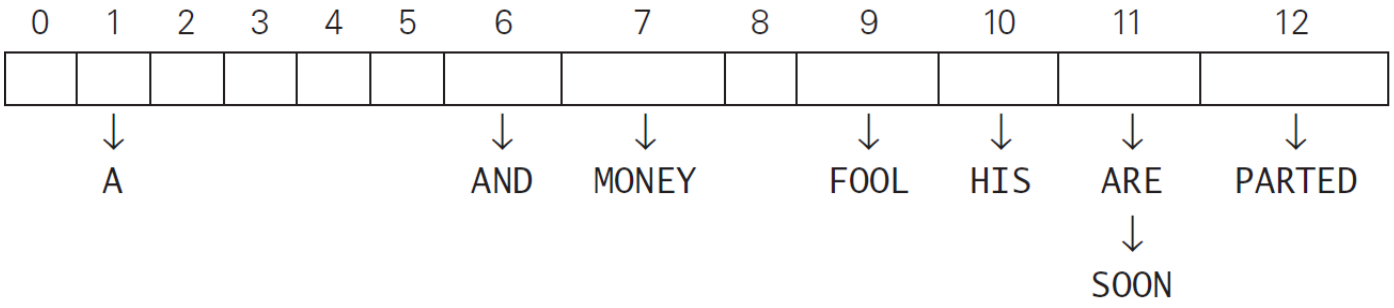
- How to search?
- Calculate hash function for key
- Lookup in the linked list assigned to that hash table cell

Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup KID:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26



Open hashing (Separate chaining)

- How to search?
- Calculate hash function for key
- Lookup in the linked list assigned to that hash table cell

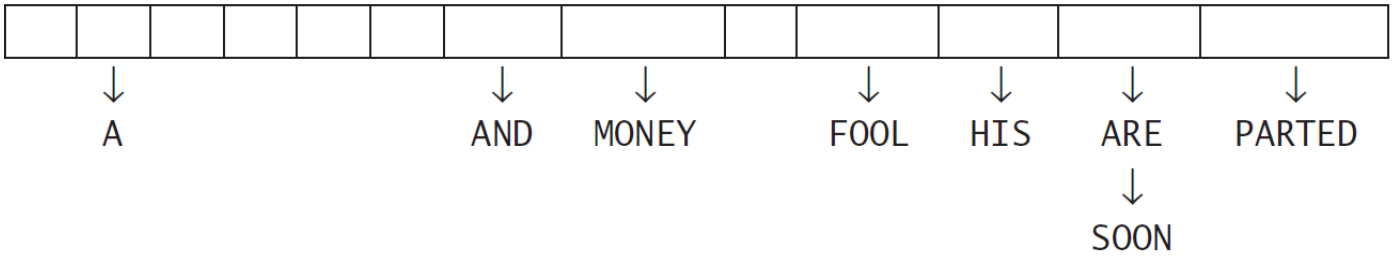
Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup KID:

$h(KID) = (11 + 9 + 4) mod\ 13 = 11$

not found in linked list assigned to 11 → un-successfull search



Open hashing

- If hash function distributes keys uniformly, average length of linked list will be $\alpha = n/m$. This ratio is called *load factor*.

- Average number of probes (linked searches) in successful, S, and unsuccessful searches, U *are*:

$$S \approx 1 + \frac{\alpha}{2} \qquad U = \alpha$$

- α too small → lots of empty lists, inefficient use of space
- α too large → longer linked lists, longer search time
- $\alpha \approx 1$ → efficient schema, search in time of calculating hash function

Open hashing

- Insertion → calculate hash, add at the end of responsible linked list
- Search → calculate hash, search the responsible linked list
- Deletion → calculate hash, remove from the responsible linked list

Closed hashing – linear probing

- Keys stored in hash table without use of linked list
- We should have $m \geq n$
- Calculate hash function for key
  - Empty → put the new key there
  - Not empty → check the next cell
  - End of hash table → wrap to the beginning

Closed hashing – linear probing

Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses								

0	1	2	3	4	5	6	7	8	9	10	11	12

Closed hashing – linear probing

Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12

Closed hashing – linear probing

Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

$h(A) = 1 \; mod \; 13 = 1$

$h(FOOL) = (6 + 15 + 15 + 12) mod \; 13 = 9$

$h(ARE) = (1 + 18 + 5) mod \; 13 = 11$

$h(SOON) = (19 + 15 + 15 + 14) mod \; 13 = 11$

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12

0	1	2	3	4	5	6	7	8	9	10	11	12
	A											
	A								FOOL			
	A					AND			FOOL			
	A					AND			FOOL	HIS		
	A					AND	MONEY		FOOL	HIS		
	A					AND	MONEY		FOOL	HIS	ARE	
	A					AND	MONEY		FOOL	HIS	ARE	SOON
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

Closed hashing – linear probing

- How to search?
- Calculate the hash function
- Lookup hash table
  - Equal → successful
  - Empty → un-successful
  - O.W. → check next until found, reached empty or made a whole round

Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup KID:

0	1	2	3	4	5	6	7	8	9	10	11	12
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

Closed hashing – linear probing

- How to search?
- Calculate the hash function
- Lookup hash table
  - Equal → successful
  - Empty → un-successful
  - O.W. → check next until found, reached empty or made a whole round

Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup KID: $h(soon) = (19 + 15 + 15 + 14) mod\ 13 = 11$

0	1	2	3	4	5	6	7	8	9	10	11	12
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

Closed hashing – linear probing

- How to search?
- Calculate the hash function
- Lookup hash table
  - Equal → successful
  - Empty → un-successful
  - O.W. → check next until found, reached empty or made a whole round

Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup KID: $h(soon) = (19 + 15 + 15 + 14) mod\ 13 = 11$

0	1	2	3	4	5	6	7	8	9	10	11	12
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

- Cell 11→ not equal
- Cell 12→ equal

Closed hashing – linear probing

- How to search?
- Calculate the hash function
- Lookup hash table
  - Equal → successful
  - Empty → un-successful
  - O.W. → check next until found, reached empty or made a whole round

Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup KID:

0	1	2	3	4	5	6	7	8	9	10	11	12
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

Closed hashing – linear probing

- How to search?
- Calculate the hash function
- Lookup hash table
  - Equal → successful
  - Empty → un-successful
  - O.W. → check next until found, reached empty or made a whole round

Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup KID: $h(KID) = (11 + 9 + 4) mod\ 13 = 11$

0	1	2	3	4	5	6	7	8	9	10	11	12
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

Closed hashing – linear probing

- How to search?
- Calculate the hash function
- Lookup hash table
  - Equal → successful
  - Empty → un-successful
  - O.W. → check next until found, reached empty or made a whole round

Example:

$h(K)$: sum of K 's letters' positions in the alphabet MOD 13

Lookup KID: $h(KID) = (11 + 9 + 4) mod\ 13 = 11$

0	1	2	3	4	5	6	7	8	9	10	11	12
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

- Cell 11→ not equal
- Cell 12→ not equal
- Cell 0→ not equal
- Cell 1→ not equal
- Cell 2→ empty → un-successful

Closed hashing – linear probing

- Insertion → calculate hash, add at the location or the next free cell
- Search → calculate hash, search the responsible cell and its next cell until the item is found or empty cell reached
- Deletion → not straight forward

- Bad deletion example:

0	1	2	3	4	5	6	7	8	9	10	11	12
PARTED	A					AND	MONEY		FOOL	HIS	ARE	SOON

- $DELETE(ARE) \rightarrow h(ARE) = (1 + 18 + 5)\ mod\ 13 = 11$

0	1	2	3	4	5	6	7	8	9	10	11	12
PARTED	A					AND	MONEY		FOOL	HIS		SOON

- $SEARCH(SOON) \rightarrow h(SOON) = (19 + 15 + 15 + 14)\ mod\ 13 = 11$
- Position 11 in hash table is empty → not found!
- NOT CORRECT!

Closed hashing – linear probing

- Solution for deleting:
- Use lazy deletion
- Mark cells as deleted, but not empty
- After some certain number of deletions, create the hash table again
- Excluding the ones that have been marked as deleted

Closed hashing – linear probing

- If hash function distributes keys uniformly, average probability of a cell being full will be $\alpha = n/m$. This ratio is called *load factor*.

- Average number of probes in successful, S, and unsuccessful searches, U *are*:

$$S \approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right), U \approx \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

α	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$
50%	1.5	2.5
75%	2.5	8.5
90%	5.5	50.5

Closed hashing

- When hash table gets fuller, performance deteriorates and clustering happens
- Cluster → a sequence of contiguously occupied cells
- Clusters are bad!
- Bigger cluster → higher probability of new key being added to that cluster
- During search all cells in a cluster may need to be checked to figure out a key is there or not

Closed hashing – double hashing

- Use another hash function $s(k)$ when collision happens
- $s(k)$ determines the number of cell jumps

- In case of collision:
- $l = h(k)$
- $(l + s(k)) mod\ m, (l + 2s(k)) mod\ m, \ldots$

- For best performance $s(k)$ and m must be relatively prime

- Small table → $\begin{cases} s(k) = m - 2 - k\ mod\ (m-2) \\ 8 - k\ mod\ 8 \end{cases}$
- Large table → $s(k) mod\ 97 + 1$

Closed hashing – double hashing

Example:

Keys: A, FOOL, AND, HIS, MONEY, ARE, SOON, PARTED

h(*K*): sum of *K* 's letters' positions in the alphabet MOD 13

S(K): sum of *K* 's letters' positions in the alphabet MOD 5

keys	A	FOOL	AND	HIS	MONEY	ARE	SOON	PARTED
hash addresses	1	9	6	10	7	11	11	12

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

0	1	2	3	4	5	6	7	8	9	10	11	12

Hashing vs. Balanced Trees

- Asymptotic time efficiency for search, insert, delete

	Average case	Worst case
Hashing	$\theta(1)$	$\theta(n)$
Balanced Trees	$\theta(\log n)$	$\theta(\log n)$

- Ordering preservation
  - Hashing → not stable
  - Balanced trees → stable

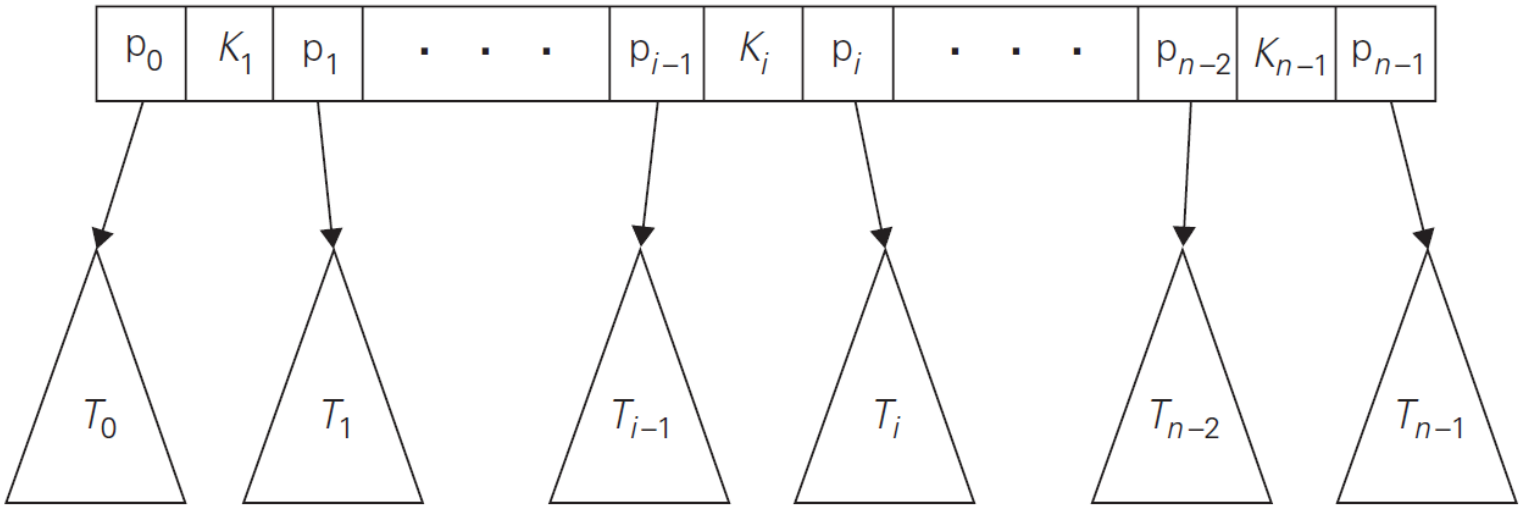
Indexing: B-Tree

- Index → a principal in organizing a very large number of records on a disk. Provides some information about the location of records

- B-Tree → most important index organization for data sets of structured records

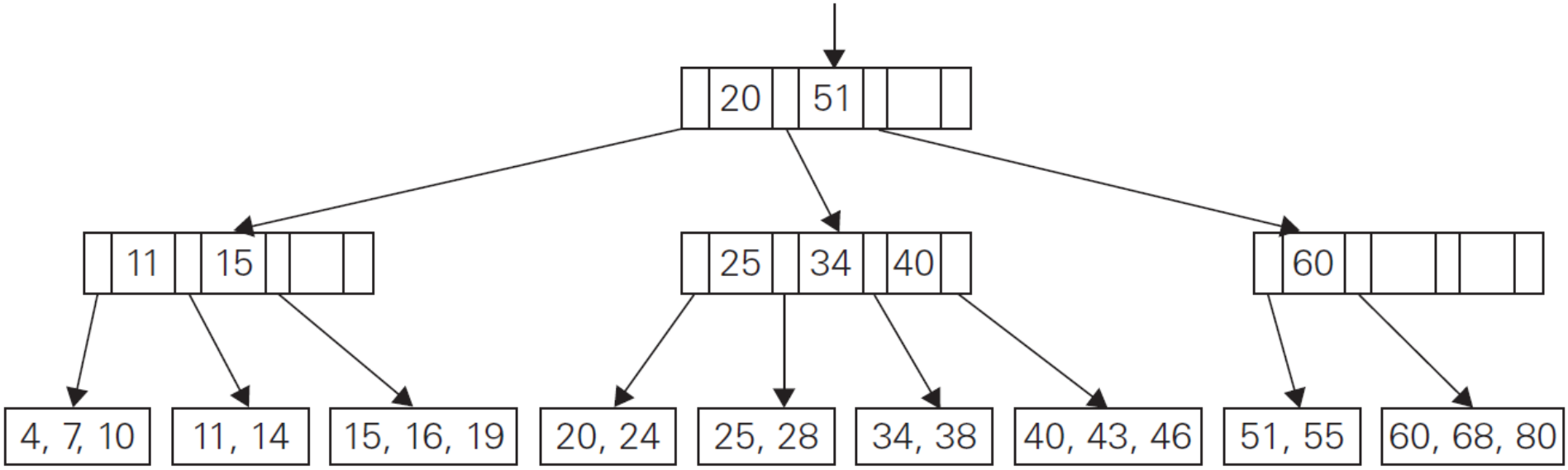
Indexing: B-Tree

- All data records are sorted at the leaves
- In increasing order of the keys
- Parental nodes are used for indexing
- Each parental node contains up to $n - 1$ ordered keys
- $(k_1 < k_2 < \cdots < k_{n-1})$



Indexing: B-Tree

- A B-Tree of order $m \geq 2$ must satisfy:
- Root is either a leaf or has between 2 and m children
- Each node (except root and leaves) has between $\left\lceil \frac{m}{2} \right\rceil$ and m children
- Tree is perfectly balanced. All leaves are at the same level



Indexing: B-Tree

- Searching:
- Start with root, follow a chain of pointers to the leaf considering the range of each child

- Number of nodes of a B-Tree to be accessed during a search is equal to the height of the tree plus one

order m	50	100	250
h's upper bound	6	5	4

- In actual applications this number rarely exceeds 3

- Insertion and deletion are less straightforward but can be done in $O(\log n)$

Indexing: B-Tree

- Insertion:
- Apply search to the new record's key k to find the appropriate leaf for the new record

1. Free space in leaf available → add it in appropriate order in the leaf

2. No room for the new record → split leaf in half, add the middle key to the parent with each half as the children, if parent is also full, keep splitting…

Discussion

1. For the input 30, 20, 56, 75, 31, 19 and hash function $h(K) = K \bmod 11$

   a. construct the open hash table.

   b. find the largest number of key comparisons in a successful search in this table.

   c. find the average number of key comparisons in a successful search in this table.

Discussion

1. a.

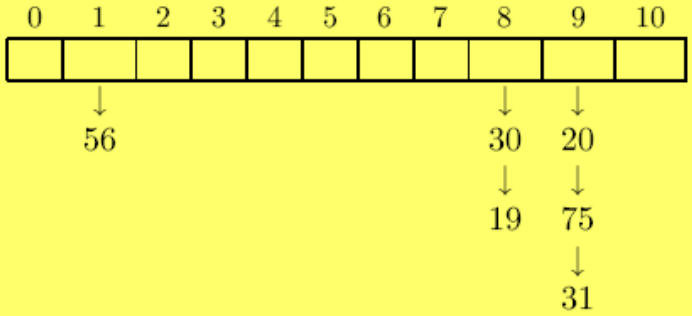
The list of keys: 30, 20, 56, 75, 31, 19

The hash function: $h(K) = K \bmod 11$

The hash addresses:

K	30	20	56	75	31	19
$h(K)$	8	9	1	9	9	8

The open hash table:



b. The largest number of key comparisons in a successful search in this table is 3 (in searching for $K = 31$).

c. The average number of key comparisons in a successful search in this table, assuming that a search for each of the six keys is equally likely, is

$$\frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 2 = \frac{10}{6} \approx 1.7.$$

Discussion

2. For the input 30, 20, 56, 75, 31, 19 and hash function $h(K) = K \bmod 11$

   a. construct the closed hash table.

   b. find the largest number of key comparisons in a successful search in this table.

   c. find the average number of key comparisons in a successful search in this table.

Discussion

2. a.

The list of keys: 30, 20, 56, 75, 31, 19

The hash function: $h(K) = K \bmod 11$

The hash addresses:

K	30	20	56	75	31	19
$h(K)$	8	9	1	9	9	8

0	1	2	3	4	5	6	7	8	9	10
								30		
								30	20	
	56							30	20	
	56							30	20	75
31	56							30	20	75
31	56	19						30	20	75

b. The largest number of key comparisons in a successful search is 6 (when searching for $K = 19$).

c. The average number of key comparisons in a successful search in this table, assuming that a search for each of the six keys is equally likely, is

$$\frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 6 = \frac{14}{6} \approx 2.3.$$

Discussion

5. ▶The ***birthday paradox*** asks how many people should be in a room so that the chances are better than even that two of them will have the same birthday (month and day). Find the quite unexpected answer to this problem. What implication for hashing does this result have?

Discussion

5. The probability of n people having different birthdays is $\frac{364}{365}\frac{363}{365}...\frac{365-(n-1)}{365}$. The smallest value of n for which this expression becomes less than 0.5 is 23. Sedgewick and Flajolet [SF96] give the following analytical solution to the problem:

$$(1-\frac{1}{M})(1-\frac{2}{M})...(1-\frac{n-1}{M}) \approx \frac{1}{2} \quad \text{where } M = 365.$$

Taking the natural logarithms of both sides yields

$$\ln(1-\frac{1}{M})(1-\frac{2}{M})...(1-\frac{n-1}{M}) \approx -\ln 2 \quad \text{or} \quad \sum_{k=1}^{n-1} \ln(1-\frac{k}{M}) \approx -\ln 2.$$

Using $\ln(1-x) \approx -x$, we obtain

$$\sum_{k=1}^{n-1} \frac{k}{M} \approx \ln 2 \quad \text{or} \quad \frac{(n-1)n}{2M} \approx \ln 2. \quad \text{Hence,} \quad n \approx \sqrt{2M \ln 2} \approx 22.5.$$

The implication for hashing is that we should expect collisions even if the size of a hash table is much larger (by more than a factor of 10) than the number of keys.