Data Structures

November 30

Objectives

Discuss the following topics:

- **Data Compression and Huffman Codes**
- **Hashing**
- **Hash Functions**
- **Collision Resolution**
- **Deletion**
- **Perfect Hash Functions**

Hashing

Objectives

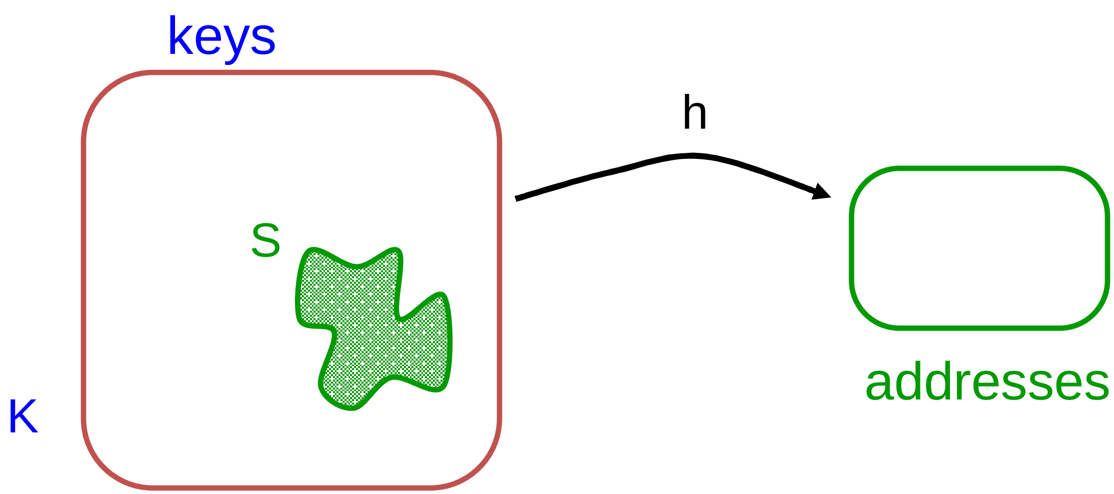
Discuss the following topics:

- **Data Compression and Huffman Codes**
- **Hashing**
- **Hash Functions**
- **Collision Resolution**
- **Deletion**
- **Perfect Hash Functions**

Hashing

- To find a function (*h*) that can transform a particular key (*K*) (a string, number or record) into an index in the table used for storing items of the same type as *K,* the function *h* is called a **hash function**
- If *h* transforms different keys into different numbers, it is called a **perfect hash function**
- To create a perfect hash function, the table has to contain at least the same number of positions as the number of elements being hashed

Hashing



synonyms ⇒ collisions (botsingen)

choice of address function (hash function)

easy to compute

good spread (little clustering)

clustering

primary, secondary

search in O(1) time

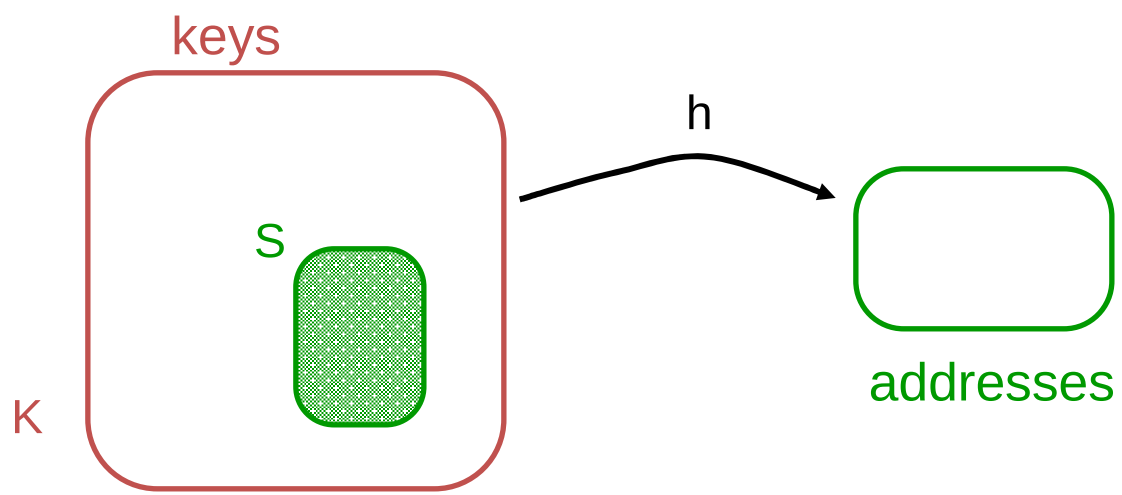
provided few collisions and little clustering

- table size
- address function (hash function)
- given keys

A linear example

0	1	2	3	4	5	6	7	8	9	10	MOD 11
			3	8	60		29	74			60,29 ,74,3 8
	23		3	8	60	19	29	74			19,23
	23	40	3	8	60	19	29	74			40

Perfect Hashing



Static Hashing:

S known

h to be determined such that *no* collisions

$x \in S$?

compute h(x)

look at address h(x):

x present $\Rightarrow x \in S$

other key $\Rightarrow x \notin S$

Cichelli's Perfect Hash

2 do	20 record
3 end	21 packed
4 else	22 not
5 case	23 then
6 downto	24 procedure
7 goto	25 with
8 to	26 repeat
9 otherwise	27 var
10 type	28 in
11 while	29 array
12 const	30 if
13 div	31 nil
14 and	32 for
15 set	33 begin
16 or	34 until
17 of	35 label
18 mod	36 function
19 file	37 program

g:

a	11	l	15	u	14
b	15	m	15	v	10
c	1	n	13	w	6
f	15	p	15	y	13
g	3	r	14		
h	15	s	6		
i	13	t	6		

(remaining such as z 0)

$h(key) = L + g(key[1]) + g(key[L])$

L → length

Cichelli

perfect hash table example (an aside)

- This example is actually also an example of a ***minimal*** perfect hash (minimality means that the number of keys, n, is equal to the table size, m; recall for perfect hash need $n \leq m$).
- Note in general you can have m^n hash functions (the number of functions from a finite set A to a finite set B is $|B|^{|A|}$). -- |.| is num of elements.
- The number of perfect hash functions is m*(m-1)*…*(m-n) = m!/(m-n)! (that is the number of injections from a finite set A to a finite set B is: $|B|\cdot(|B|-1)\cdot(|B|-2)\cdot\ldots\cdot(|B|-|A|)$)

Perfect Hash Functions

- If a function requires only as many cells in the table as the number of data so that no empty cell remains after hashing is completed, it is called a **minimal perfect hash function**
- **Cichelli's method** is an algorithm to construct a minimal perfect hash function
- It is used to hash a relatively small number of reserved words

Cichelli's perfect hash by example

Compute for each character *ch* occurring in first or last position of the keywords how times this happens for such an *ch*;

For the example:

2	A
1	B
2	C
8	D
9	E
5	F
1	G
1	H
2	I
	J
	K
4	L

2	M
6	N
7	O
	Q
3	P
5	R
1	S
7	T
1	U
2	V
2	W
	X
1	Y
	Z

Cichelli's perfect hash by example

Sort the keys on the sum of frequencies of first and last:

ELSE	18
END	17
OTHERWISE	16
TYPE	16
DO	15
DOWNTO	15
TO	14
FILE	14
RECORD	13
NOT	13
THEN	13
OR	12
OF	12
PROCEDURE	12
REPEAT	12
CASE	11
WHILE	11
PACKED	11
FUNCTION	11
DIV	10
AND	10
MOD	10
NIL	10
FOR	10
CONST	9
GOTO	8
SET	8
IN	8
LABEL	8
VAR	7
IF	7
BEGIN	7
UNTIL	5
PROGRAM	5
WITH	3
ARRAY	3

Cichelli's perfect hash by example

Modify the previous list once more according to the following:
Make sure that any word whose hash value is determined by assigning the associated character values already determined by previous words is placed next:

ELSE	18
END	17
OTHERWISE	16
D0	15
DOWNTO	15
TYPE	16
TO	14
…… etc	………. etc

Determine hash value conflicts as early as possible!

Cichelli’s perfect hash by example

ELSE	18
END	17
OTHERWISE	16
D0	15
DOWNTO	15
TYPE	16
TO	14
FILE	14
OF	12
THEN	13
NOT	13
FUNCTION	11
RECORD	13
REPEAT	12
OR	12
FOR	10
PROCEDURE	12
PACKED	11
WHILE	11
CASE	11

CONST	9
DIV	10
VAR	7
AND	10
MOD	10
PROGRAM	5
NIL	10
LABEL	8
SET	8
IN	8
IF	7
GOTO	8
BEGIN	7
UNTIL	5
ARAY	3
WITH	3

Cichelli's perfect hash by example

Backtracking search procedure: attempts to find a set of associated values which will permit the unique referencing of all members of the key word list.

It does this by trying the words one by one in order.

Cichelli's perfect hash by example

If both the first and last letter of the keyword have an associated value, try the word.

If either the first or last letter has an associated value, vary the associated value of the unassigned character from zero to the *maximum* allowed associated value, trying each occurrence.

If both letters are as yet unassigned, vary the first and then the second, trying each possible combination.

Cichelli's perfect hash by example

Each "try" tests whether the given hash value is already assigned and, if not, reserves the value and assigns the letters.

If all keywords have been processed, stop;
Otherwise invoke the search procedure recursively to place the next word.

If the "try" fails (i.e., is assigned), remove the word by backtracking.

See also pages 538-542 in Drozdek

Cichelli's Method summary

- Where *g* is the function to be constructed

h(*word*) = (*length*(*word*) + *g*(*firstletter*(*word*)) + *g*(*lastletter*(*word*))) mod *TSize*

- The algorithm has three parts:
  - Computation of the letter occurrences
  - Ordering the words
  - Searching

Hash Functions

- The **division** method is the preferred choice for the hash function if very little is known about the keys

  TSize =*sizeof*(*table*), as in $h(K) = K \bmod TSize$

- In the **folding** method, the key is divided into several parts which are combined or folded together and are often transformed in a certain way to create the target address

Hash Functions (continued)

- In the **mid-square** method, the key is *squared* and the middle or *mid* part of the result is used as the address
- In the **extraction** method, only a part of the key is used to compute the address
- Using the **radix transformation**, the key K is transformed into another number base; K is expressed in a numerical system using a different radix

Hash Functions (continued)

- Truncation
- Multiplication: $h(k) = \lfloor \text{fract}(\phi K) \cdot M \rfloor$, where ϕ is irrational; Knuth suggests $\phi = (\sqrt{5}-1)/2 \approx 0.6180339887$

Collision Resolution; open hashing

- In open addressing, all elements are stored in the hash table itself
- That is each entry is either a key or nil.
- Avoids pointers altogether: compute the sequence of slots to be examined
- Larger number of slots for the same memory: potentially fewer collisions and faster retrieval
- Insertion: successively examine, or ***probe*** the hash table until you find an empty slot in which you put the key – same sequence is also followed by the search

Collision Resolution

- The sequence of addresses to be probed may depend on the key
- When the probing sequence is specified for each key, we are dealing with ***open addressing.*** The probing sequence can be specified by a probing function.
- h(K), h(K)+p(1), h(K)+p(2), …, h(K)+p(M-1), where M is the Tsize (table size)

Collision Resolution

- h(K), h(K)+p(1), h(K)+p(2), …, h(K)+p(M-1), where M is the Tsize (table size)
- The numbers h(K) through h(K)+p(M-1) are also normalized, usually by mod M:
- h(K) mod M, h(K)+p(1) mod M , … , h(K)+p(M-1) mod M; require that it generates a permutation of {0,…,M-1}

Collision Resolution

- The simplest method is ***linear probing,*** for which $p(i) = i$, and for the ith probe, the position to be tried is $(h(K) + i) \bmod Tsize$
- Recall that (a,b) denotes the gcd of a and b
- Or slightly more general: $h(K) - i \cdot c \pmod{Tsize}$, with (c,Tsize)=1 – guarantees permutation, such linear probings are called ***permissible***
- In the sequel we denote Tsize also by M or m

0	1	2	3	4	5	6	7	8	9
	x	x				x	x	x	x

Clustering

Check what happens with inserting a random key

Collision Resolution

- ***pseudo-random***: $h(K) + r_0 \bmod M$, $h(K)+r_1 \bmod M$, ..., $h(K)+r_{M-1}$, where$r_0 = 0, r_1, \ldots, r_{M-1}$ is pseudo random permutation
- ***Quadratic***: $h(K) \pm i^2 \pmod{M}$ provided M is prime, $M \equiv 3 \pmod{4}$; again to guarantee permutation
- ***Double Hashing:*** $h(K) - i \cdot p(K) \pmod{M}$
  - p probe function
  - h hash function
  - independent : p is not derived from h
  - Permutation iff $(M, p(K)) = 1$ (no common factors, such probe is called *permissible*

Collision Resolution

- *Claim:* h(K)- i p(K) mod M, all different for $0 \leq i \leq M-1$ iff (p(K),M)=1
- Recast: a-ib mod M, $0 \leq i < M$ are mutually different numbers iff (b,M)=1
- Proof: a) assume (b,M)=1 and $a-i_1b=a-i_2b$ mod M, $0 \leq i_1, i_2 < M$. We see: $(i_2-i_1)b=s*M$ for some integer s. M divides evenly into (i_2-i_1), as (M,b)=1 . Or $(i_2-i_1)=0$ mod M. I.e., $i_2-i_1 = s_2*M$, for integer s_2. Hence,

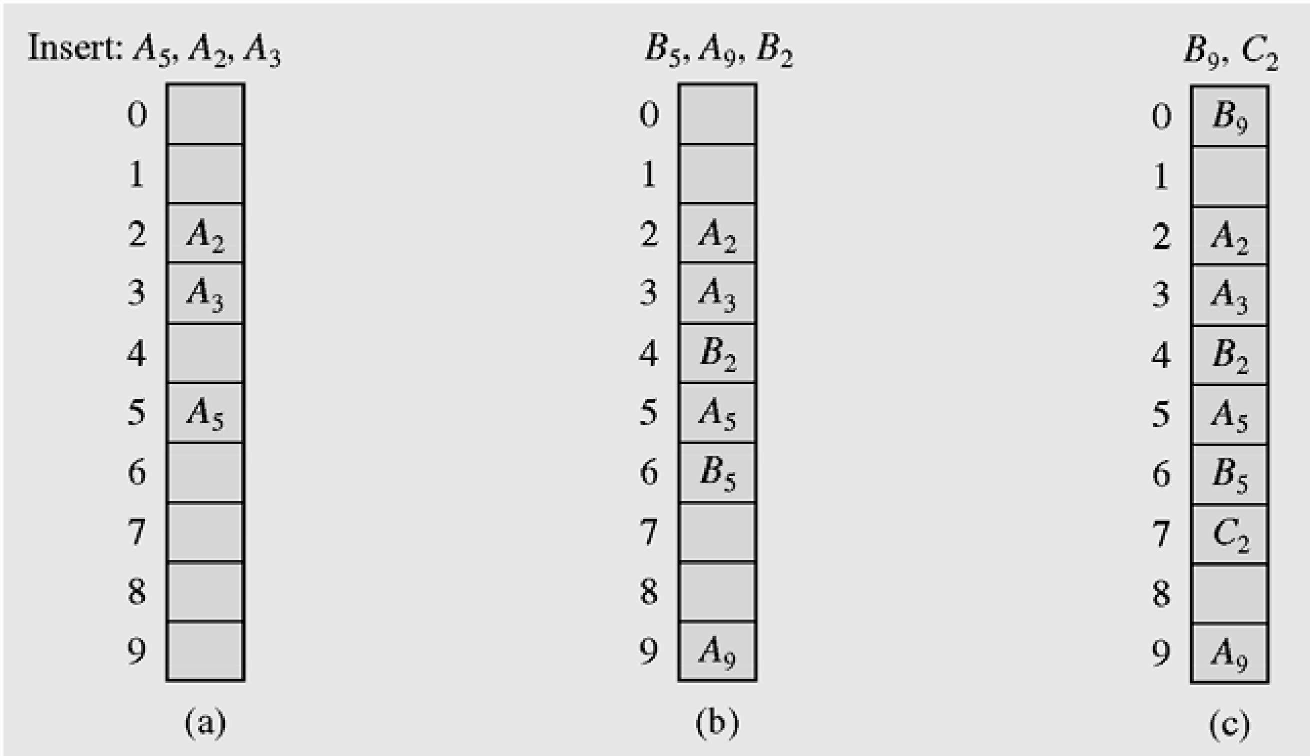
$0 \leq |s_2*M| = |i_2-i_1| \leq M-1$ or

$0 \leq |s_2|*|M| = |i_2-i_1| < M$, thus $s_2=0$

Collision Resolution

- Proof: b) we will show $(b,M)=1 \Rightarrow$ different or equivalently $(b,M) \neq 1 \Rightarrow$ not all different.
- Let d= (b,M) (we know d>1). Consider i_1:=0, and i_2:=M/d; clearly: $0 \leq i_2, i_1 < M$ and (i_2-i_1) = M/d; hence, $b^* (i_2-i_1) = b^* M/d = b' M$ or $b^* (i_2-i_1) = 0 \bmod M$, or $a-b^* i_1 = a - b^* i_2 \bmod M$, for some $0 \leq i_2, i_1 < M$. Hence, not all different.

Collision Resolution (continued) Linear



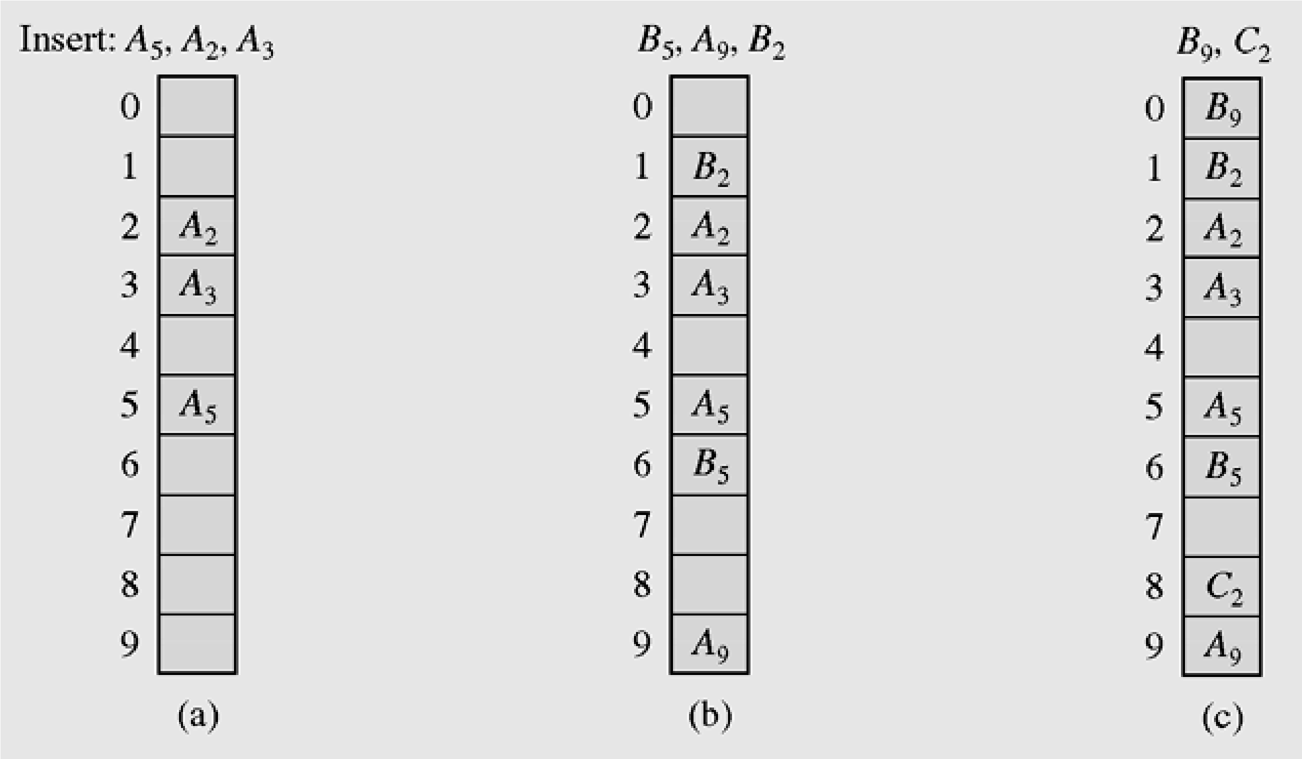
Resolving collisions with the linear probing method.

Subscripts indicate the home positions of the keys being hashed.

Collision Resolution (continued)
Linear example

0	1	2	3	4	5	6	7	8	9	10	MOD 11
				38	60		29	74			60,29,74,38
	23			38	60	19	29	74			19,23
	23	40		38	60	19	29	74			40

Collision Resolution (continued)



Using quadratic probing for collision resolution

Collision Resolution (continued)
Double Hashing example

Insert 60, 29,74, 38, 19, 23, 40;

key	60	29	74	38	19	23	40	**K**
hash address	5	7	8	5	8	1	7	**h(K) = K mod 11**
probe	1	2	3	3	4	4	1	**p(K) = (K mod 4) + 1**

0	1	2	3	4	5	6	7	8	9	10	
		38			60		29	74			60,29,74,38
	23	38		19	60		29	74			19, 23
	23		40	38	60	40	29	74			40

Collision Resolution (continued)

	Linear Probing	Quadratic Probing[a]	Double Hashing
successful search	$\frac{1}{2}\left(1+\frac{1}{1-LF}\right)$	$1-\ln(1-LF)-\frac{LF}{2}$	$\frac{1}{LF}\ln\frac{1}{1-LF}$
unsuccessful search	$\frac{1}{2}\left(1+\frac{1}{(1-LF)^2}\right)$	$\frac{1}{1-LF}-LF-\ln(1-LF)$	$\frac{1}{1-LF}$

Load Factor $LF = \frac{\text{number of elements in the table}}{\text{table size}}$

[a] The formulas given in this column approximate any open addressing method that causes secondary clusters to arise, and quadratic probing is only one of them.

Formulas approximating, for different hashing methods, the average numbers of trials for successful and unsuccessful searches (Knuth, 1998)

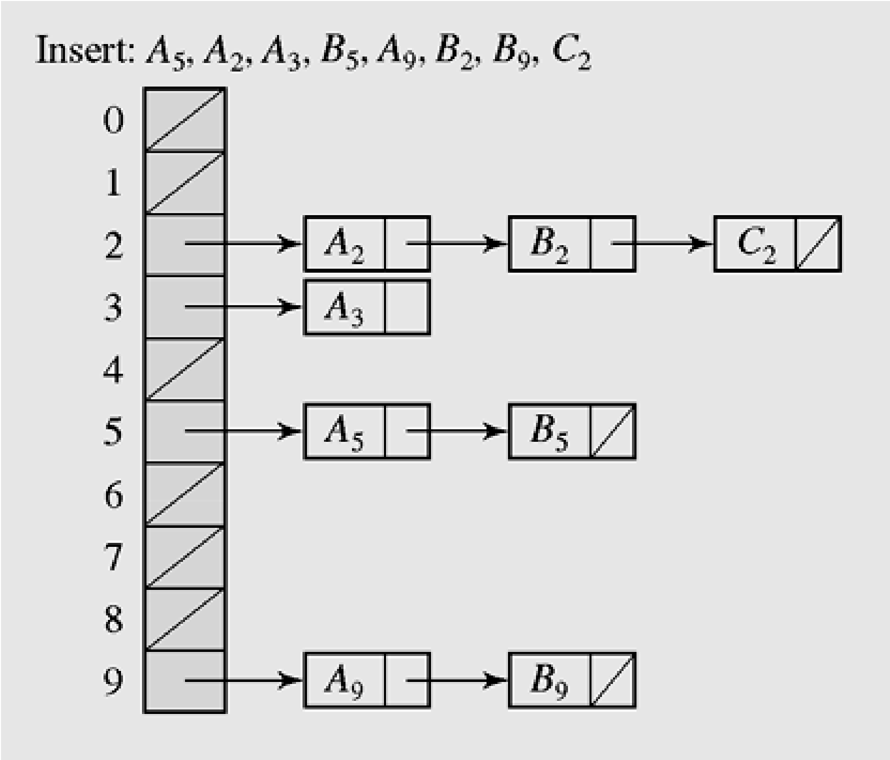
Clustering in Open Hashing

- **linear:** primary clustering, neighbors in each other’s paths
- **secondary clustering:** synonyms follow the same path (pseudorandom, quadratic)
- **double:**
  - ‘ independent’
  - step size (p(K))

Chaining

- In **chaining**, each position of the table is associated with a linked list or **chain** of structures whose `info` fields store keys or references to keys
- This method is called **separate chaining**, and a table of references (pointers) is called a **scatter table**

Chaining (continued)

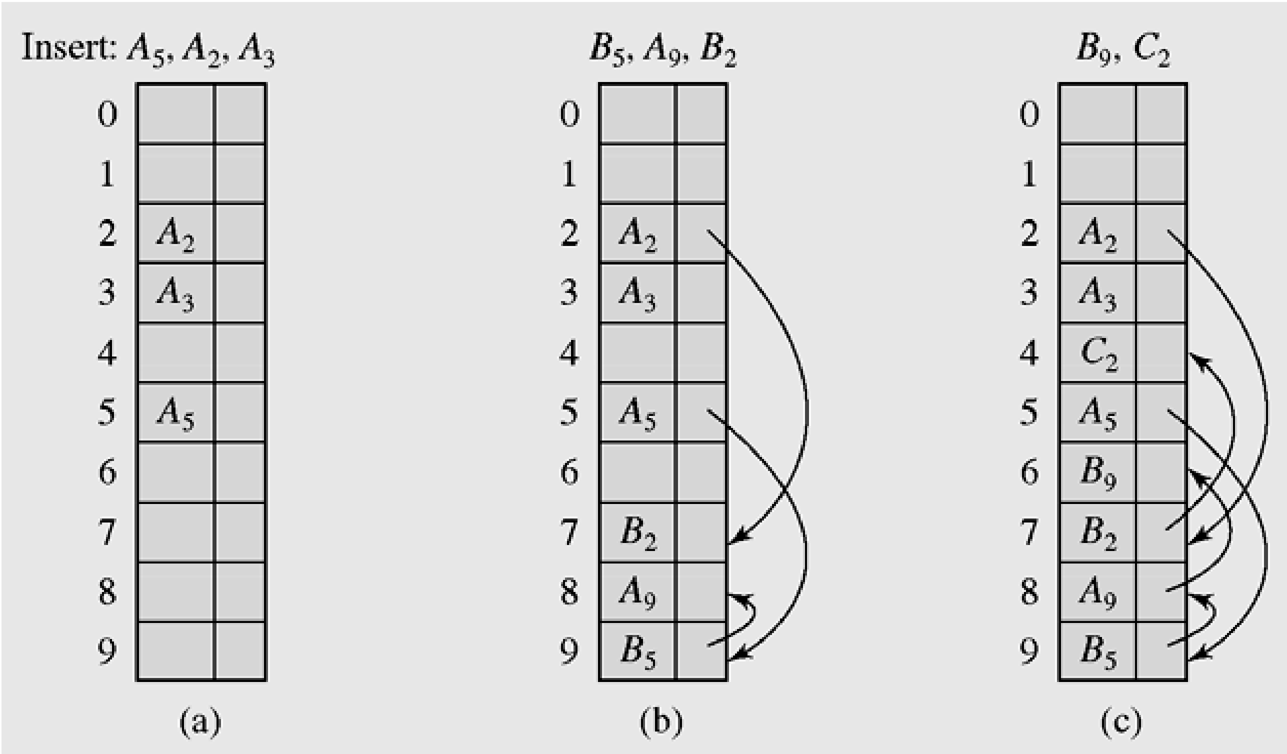


In chaining, colliding keys are put on the same linked list

Chaining (continued)

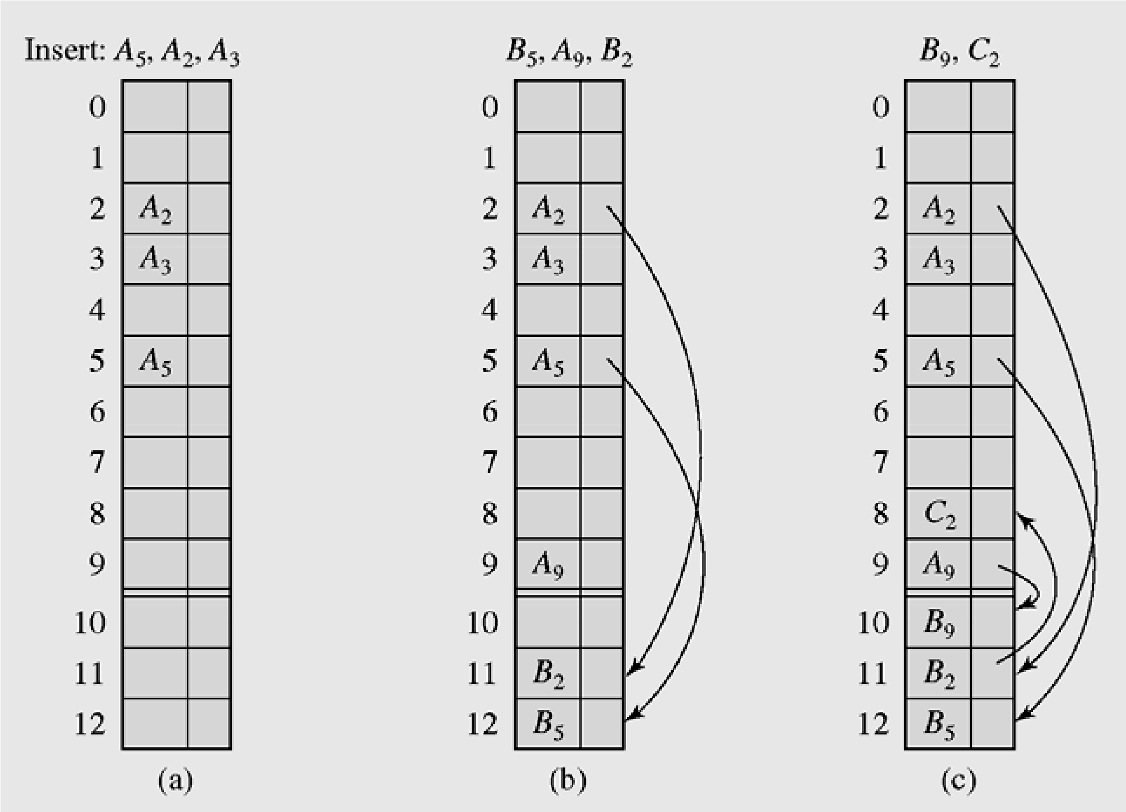
- A version of chaining called **coalesced hashing** (or **coalesced chaining**) combines linear probing with chaining
- An overflow area known as a **cellar** can be allocated to store keys for which there is no room in the table

Chaining (continued)



Coalesced hashing puts a colliding key in the last available position of the table

Chaining (continued)

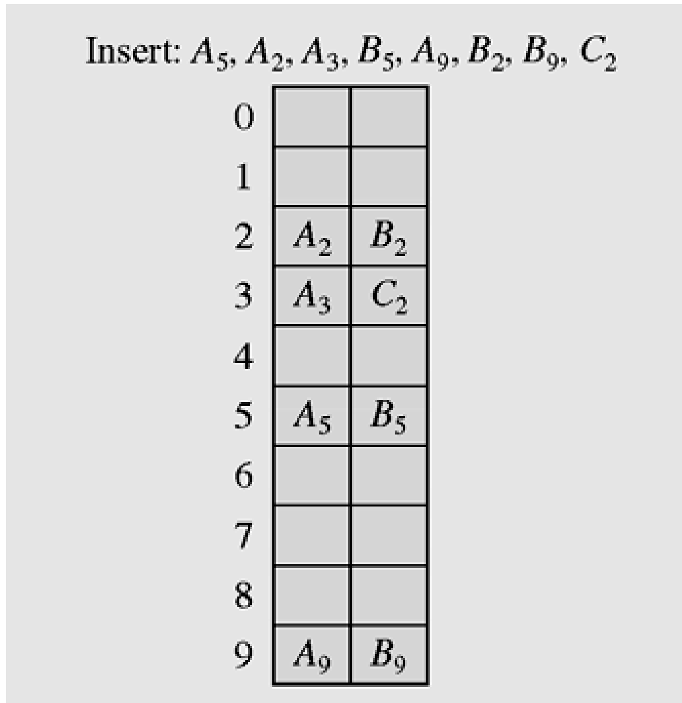


Coalesced hashing that uses a cellar

Bucket Addressing

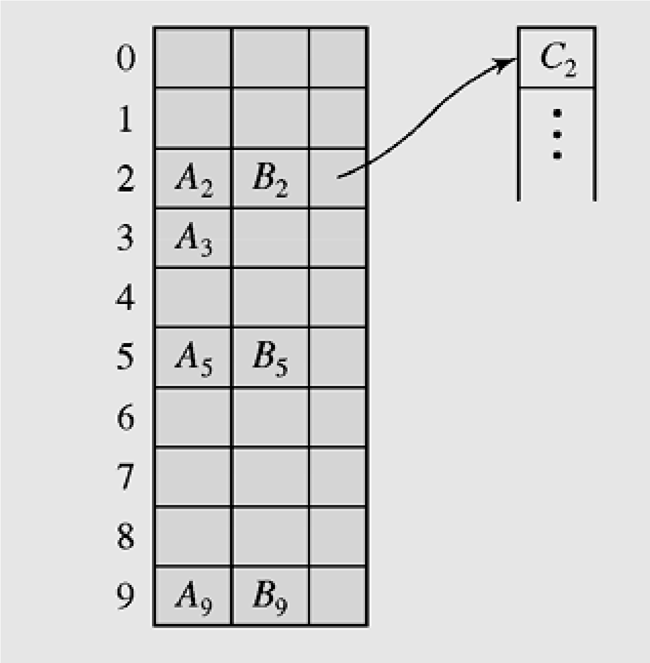
- To store colliding elements in the same position in the table can be achieved by associating a bucket with each address
- A **bucket** is a block of space large enough to store multiple items

Bucket Addressing (continued)



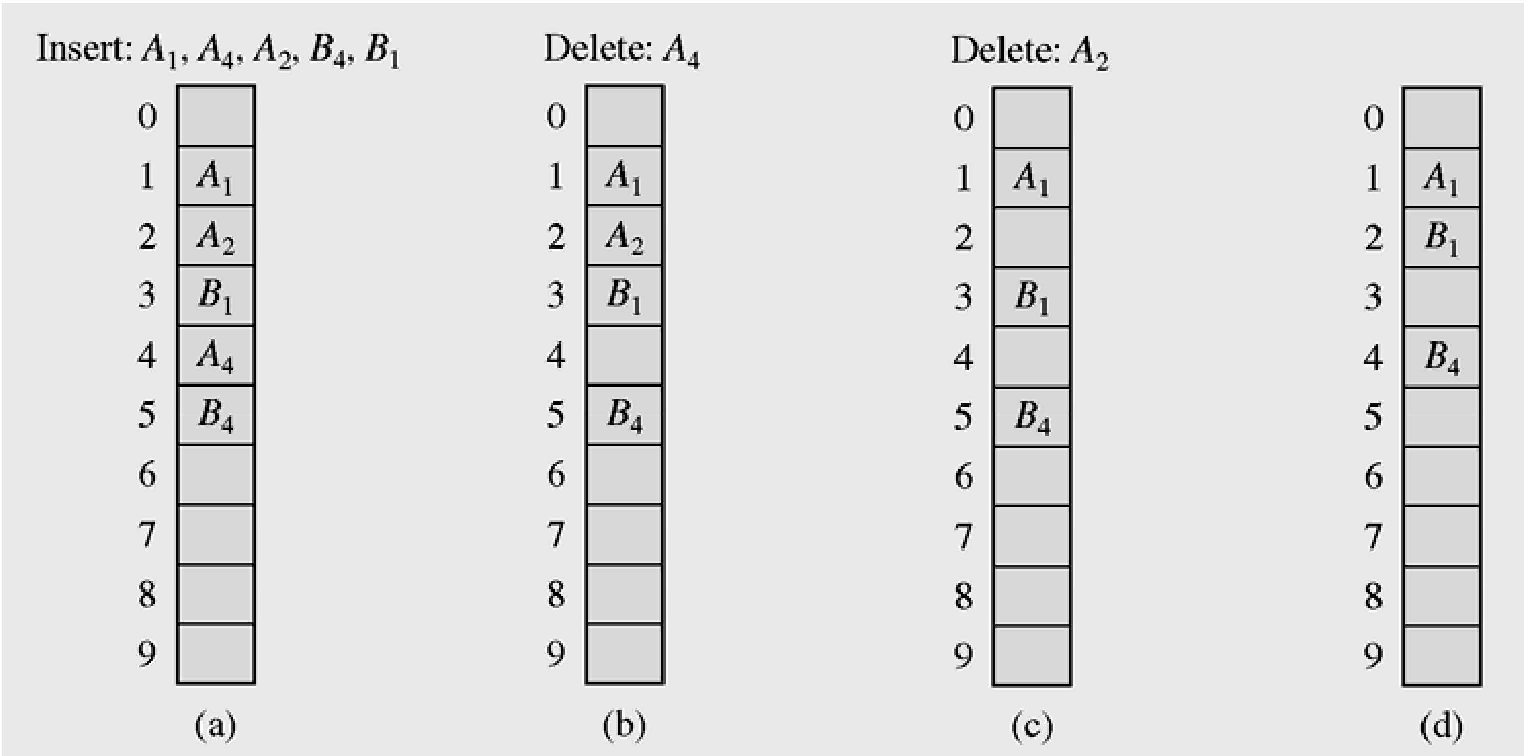
Collision resolution with buckets and linear probing method

Bucket Addressing (continued)



Collision resolution with buckets and overflow area

Deletion



Linear search in the situation where both insertion and deletion of keys are permitted

Summary

- Hash functions include the division, folding, mid-square, extraction and radix transformation methods
- Collision resolution includes the open addressing, chaining, and bucket addressing methods
- Cichelli's method is an algorithm to construct a minimal perfect hash function