Dictionary

강지훈
jhkang@cnu.ac.kr
충남대학교 컴퓨터공학과

Dictionary

Dictionary (Symbol Table)

- A set of Key-object pairs.
  - Name-attribute pairs
- Related operations:
  - Determine if a particular key is in the dictionary.
  - Retrieve the object of that key.
  - Modify the object of that key.
  - Insert a new key and its object.
  - Delete a key and its object.

- Searching times for several implementations
  - Binary Search Trees: O(log n)
  - Hash Tables: O(1)

Public Functions for Class Dic

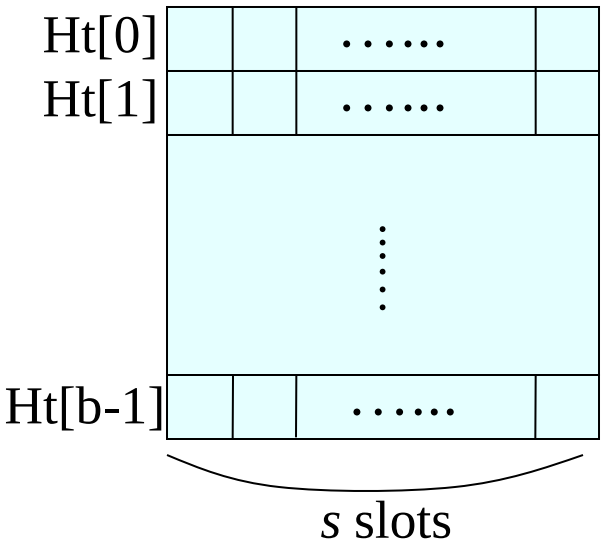
- public Dic()
- public boolean doesKeyExist (Key aKey)
- public Object objectForKey (Key aKey)
- public boolean addObjectForKey (Object anObject, Key aKey)
- public boolean replaceObjectForKey (Object anObject, Key aKey)
- public boolean removeObjectForKey (Key key)
- public void removeAllObjects ()

Hashing을 사용하여 구현된 Dictionary

Hash Table

Hash Table

- We store the identifiers (names) in a fixed size table, called a hash table.
- We use an arithmetic function f that maps a given identifier x into an index $f(x)$ of the hash table.
  - $f(x)$: Hash value
  - Let Ht be a hash table of size b:
    - Then, $0 \le f(x) \le b-1$
    - There are b buckets in Ht.
    - In each bucket, there are s slots.
      - Usually, $s = 1$.



Example

- Hash table: Let $b = 26$, $s = 2$, and $n = 10$.
  - Loading factor: $\alpha = n/sb = 10/(2 \cdot 26) = 10/52 \approx 0.19$
- Hash function $f(X)$:
  - X : an identifier that begins with a letter.
  - $f(X)$: the ordinal number of the first letter of X.
    - $f(\text{ABC}) = 0,\ f(\text{ZXY}) = 25.$
  - Then, $0 \le f(X) \le 25$
- Input Sequence
  - D, A, GA, L, G, A2, A1, A3, A4, E
- Hashing
  - Synonyms: {A, A2, A1, A3, A4}, {GA, G}
  - Overflow
    - When A2 is hashed, collision but no overflow.
    - When A1 is hashed, collision and also overflow.
- Only constant time for each search.
  - It is independent of n.

0	A	A2
1		
2		
3	D	
4	E	
5		
6	GA	G
⋮	⋮	⋮
11	L	
⋮	⋮	⋮
25		

Identifier Density and Loading Factor

- Let T be the number of all possible identifiers in an application and n be the number of identifiers in use.
  - In a language like FORTRAN, the naming rules for the variable are as follows:
    - We use only 26 capital letters and 10 digits.
    - The first characters should be a letter.
    - The maximum length is 6.
  - Then, $T = \sum_{i=0}^{5} 26 \cdot 36^i > 1.6 \cdot 10^9$
    - Any reasonable program will use far less than T identifiers ($n << T$).
    - Ideally, if a hash table can contain T identifiers, it's OK.
      - Usually, the hash table size is relatively very small ($b << T$).
- Definitions
  - n/T : identifier density
  - $\alpha = n/sb$: loading factor (or loading density)

Overflow & Collision

- Two identifiers I_1 and I_2 are synonyms with respect to f iff $f(I_1) = f(I_2)$.

- Overflow : when a new id I is mapped by f into a full bucket.
- Collision : When two non-identical ids are hashed into the same bucket.
  (i.e, $I_1 \neq I_2$ but $f(I_1) = f(I_2)$)
  - When $s = 1$ (that is, there is only one slot in each bucket), overflow and collision occurs simultaneously.

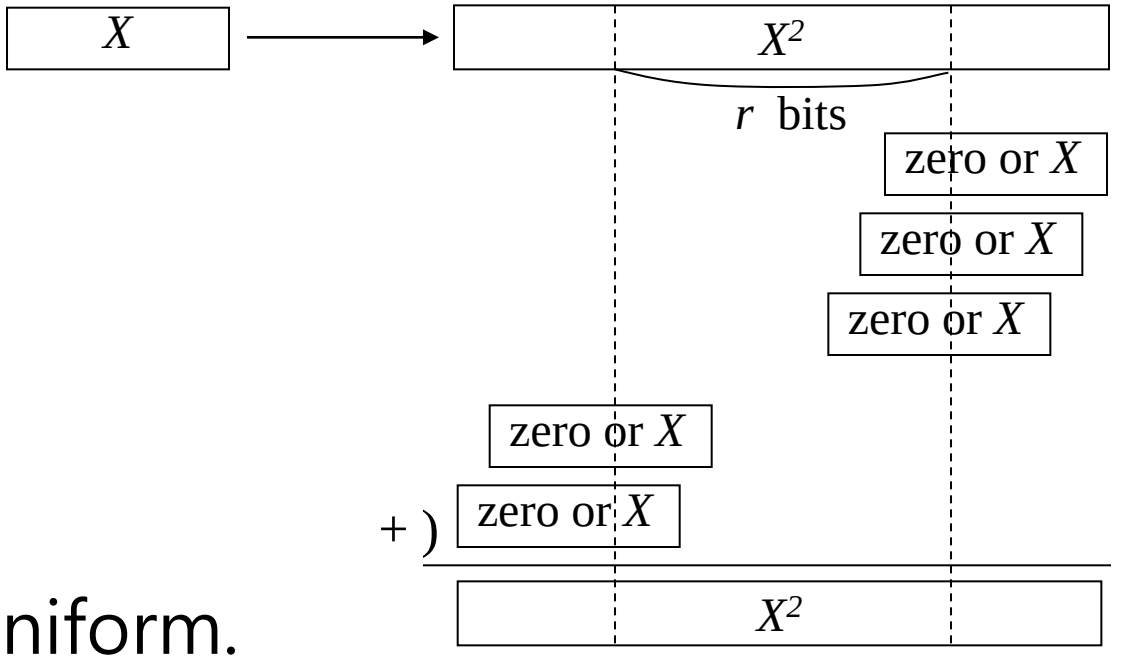
Hash Functions

Hash functions

- Good hash function
  - Collision is not avoidable !
  - Minimum number of collisions.
  - Easy to compute.
- f is a uniform hash function when the following property holds:
  - If x is an id chosen at random, then the probability of $[f(x)=i]$ would be $1/b$ for every bucket i.
  - This means that a random x has an equal chance of hashing into any of the b buckets.
- We will consider four of uniform hash functions.
  - We assume that the ids have been suitably transformed into a numerical equivalent.

Middle of Square

- Let the bucket size $b = 2^r$.
- $f_m(X)$: Take the middle r bits from X^2.
  - $0 \le f_m(X) \le 2^r - 1$



- f_m would be uniform.
  - The middle bits of X^2 will usually depend on all of the characters in X.

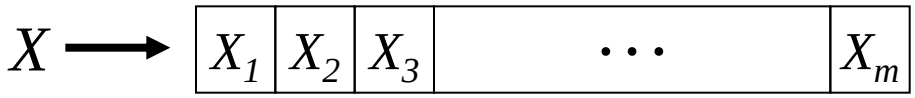
Division

- Let the bucket size $b = M$.
- $f_D(X) = X \mod M$
  - $0 \le f_D(X) \le M - 1$
- The choice of M is critical.
  - If $M = 2^k$, then f_D takes the least significant k bits from X.
    - $\Rightarrow$ f_D does not reflect all characters in X.
    - $\Rightarrow$ f_D would not be a uniform hash function.
  - Choose M as a prime number.
    - In practice, it is sufficient to choose M such that it has no prime divisions less than 20.

Folding

Basic function

- We partition the identifier X into several parts.
  - All parts, except for the last one, have the same length.



- We then add all the parts to obtain the hash address for X.

$$f(X) = X_1 + X_2 + \ldots\ldots + X_m$$

Two ways of addition

- Let X=*123203241112220*.

$X_1 = 123,\ X_2 = 203, X_3 = 241, X_4 = 112,\ X_5 = 20$

1. Shift folding

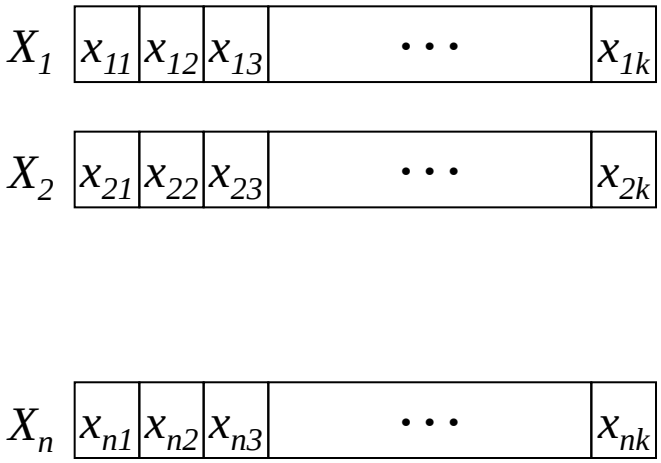
$f(X) = 123 + 203 + 241 + 112 + 20 = 699$

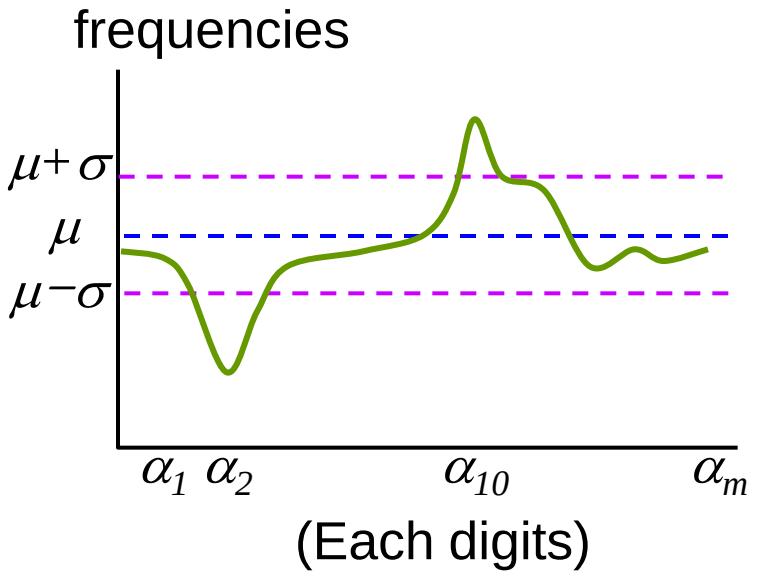
2. Folding at boundary

$f(X) = 123 + 302 + 241 + 211 + 20 = 897$

Digit Analysis

- Particularly useful in the case of a static file, where all the identifiers in the table are known in advance.





- Delete digits such as α_2 or α_{10} from each X_i until the number of digits left is small enough to give an address of the hash table.

Recommendation for Hash function

- For general purpose applications:

  Division method with a divisor M that has no prime factors less than 20.

Overflow Handling

Overflow Handling

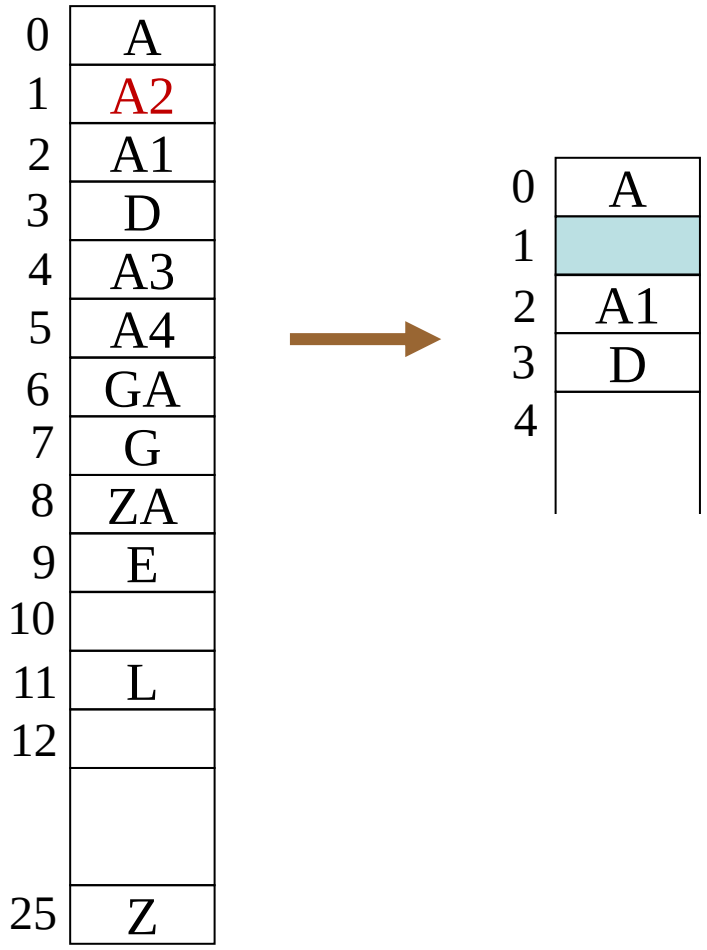
- Open Hashing vs Close Hashing
  - Close Hashing (Internal Hashing)
    - Fixed Space
  - Open Hashing (External Hashing)
    - Unlimited Space

Overflow in Close Hashing

- Linear Probing / Linear Open addressing
  - Find the next empty bucket.
  - Example
    - Let the input sequence be:
    - GA → D → A → G → L → A2 → A1 → A3 → A4 → Z → ZA → E
  - Disadvantages
    - Clustering ⇒ Increase the searching time.
      - In order to find ZA, we should compare 10 times.

0	A
1	A2
2	A1
3	D
4	A3
5	A4
6	GA
7	G
8	ZA
9	E
10	
11	L
12	
25	Z

- Deleted space is not reusable.
  - Assume that A2 was deleted.
  - Now A1 is searched, but A1 cannot be found.



- Expected number of comparisons.
  - $p=(2-\alpha) / (2-2\alpha)$, where α is the loading factor.
    - If $\alpha = 0.5$, then $p = 1.5$.
    - If $\alpha = 0.75$, then $p = 1.25/0.5 = 2.5$.
  - Although the average number of probes is small, the worst case can be large.

- Why does the linear probing perform poorly?
  - Comparison of identifiers with different hash values.

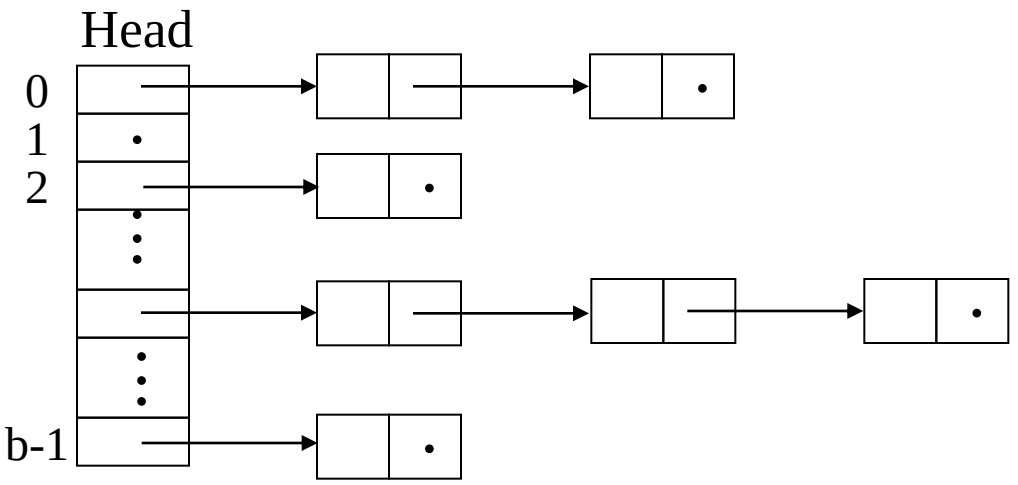
Quadratic Probing

- In linear probing, the clusters tend to merge as we enter more identifiers into the table, thus leading to bigger clusters.
- We can partially curtail the growth of these clusters and hence reduce the average number of probes by using quadratic probing.
- $f(x), (f(x)+i^2) \% b, (f(x)-i^2) \% b,$ for $1 \le i \le (b-1)/2$.
  - $f(x), (f(x)+1) \% b, (f(x)-1) \% b, (f(x)+4) \% b, (f(x)-4) \% b, \ldots$
  - When b is a prime number of the form $4j+3$, the quadratic search examines every bucket in the table.
  - Example
    - Let the bucket size $b = 4 \cdot 2+3=11$ ($j=2$).
    - 0 % 11 = **0**, 1 % 11 = **1**, -1 % 11 = **10**, 4 % 11 = **4**, -4 % 11 = **7**, 9 % 11 = **9**, -9 % 11 = **2**, 16 % 11 = **5**, -16 % 11 = **6**, 25 % 11 = **3**, -25 % 11 = **8**

- Rehashing
  - Apply a series of hash functions $f_1, f_2, \ldots, f_b$.
  - We examine buckets $f_i(x)$, $1 \le i \le b$.
- Random Probing
  - The search for an identifier x in a hash table with b buckets is carried out by examining buckets $f(x)$, $(f(x)+S(i)) \% b$, $1 \le i \le b-1$, where $S(i)$ is a pseudo random number.
  - The random number generator must generate every number from 1 to $b-1$ exactly once.
- Any method for handling overflow in Close Hashing searches for an identifier by comparing identifiers with different hash values.
  - It can be resolved by Chaining method for Open Hashing.

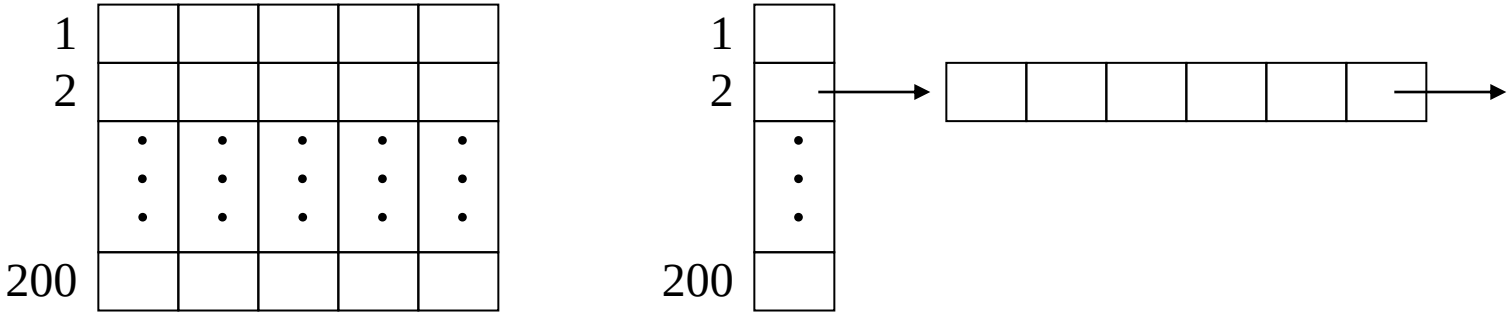
Overflow in Open Hashing

- Chaining
  - Head Node Table
    - Each chain has a head node.
  - Additional space for link fields.
  - Expected number of id comparisons $\cong 1 + \alpha/2$, where α (= n/b) is the loading factor.
  - Overall reduction in space.
    - Initially only head node table is necessary.
    - The other nodes are allocated only as needed.



Space comparison

- Assume that:
  - The length of each symbol is 5 words.
  - $b = 200$, $n = 100$ $(\alpha = 0.5)$



- Close Hashing
  - 200 - 5 = 1000 (words)
- Chaining
  - 200+100-(5+1) = 800 (words)
  - 20% saving
  - If $\alpha \approx 1$, it uses more space.
    But the number of comparisons is smaller than in case of the other methods.

Summary

Conclusion

- Hash function
  - Division is superior.
  - The divisor should be a prime (although it is sufficient to choose a divisor that has no prime factors less than 20)
- Chaining is better than any methods for Close Hashing.
  - Chaining: unlimited space
  - Close hashing: limited space

Summary

- Hashing은 Dictionary를 아주 효율적으로 구현할 수 있는 방법중의 하나이다.
  - 삽입, 삭제, 검색: O(1)
- 모든 이름을 정렬된 순서로 방문해야 한다면?
- Loading factor가 적정 수준 이상이 되면?
  - Hash table이 꽉 차면?

Dynamic Hashing

Motivation

- Hash table에 삽입된 원소가 너무 많아지면?
  - Collision 증가
- 해결책은?
  - Hash Table의 크기를 동적으로 증가시키자!
- 크기를 동적으로 늘릴 때의 문제점은?
  - 모든 원소를 다시 hashing하여 삽입해야 하는가?

h(k,p)

- h(k): hash function
  - The range of h is sufficiently large
- h(k,p): the least significant p bits of h(k)

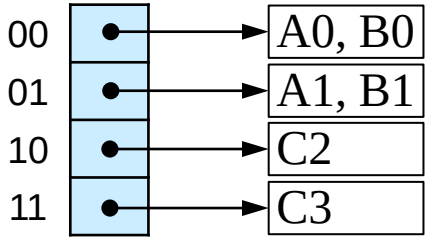
- Example:
  - h(A0,2) = 00 = 0
  - h(A0,3) = 000 = 0
  - h(C5,2) = 01 = 1
  - h(C5,3) = 101 = 5

k	h(k)
A0	100 000
A1	100 001
B0	101 000
B1	101 001
C1	110 001
C2	110 010
C3	110 011
C5	110 101

Dynamic Hashing Using Directories

Directory

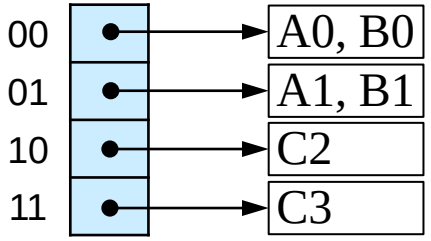
- Pointers to buckets
- Directory Size for h(k,p) : 2^p
  - 2^2 = 4 if h(k,2) is used
  - 2^5 = 32 if h(k,5) is used
- directory depth
  - the number of bits of h(k) used to index the directory
- Example:
  - Directory Depth: 2
    - So, the directory size: 2^4 = 4
  - 2 slots per bucket



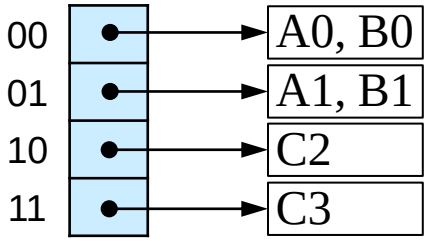
How to search for a key k?

- Merely examine the bucket pointed to by the directory entry d[h(k,p)] when the current directory depth is p.

- Example: Search for a key B1:
  - h(k,B1) = 01
  - Examine the bucket pointed to by d[01] .
  - Found.

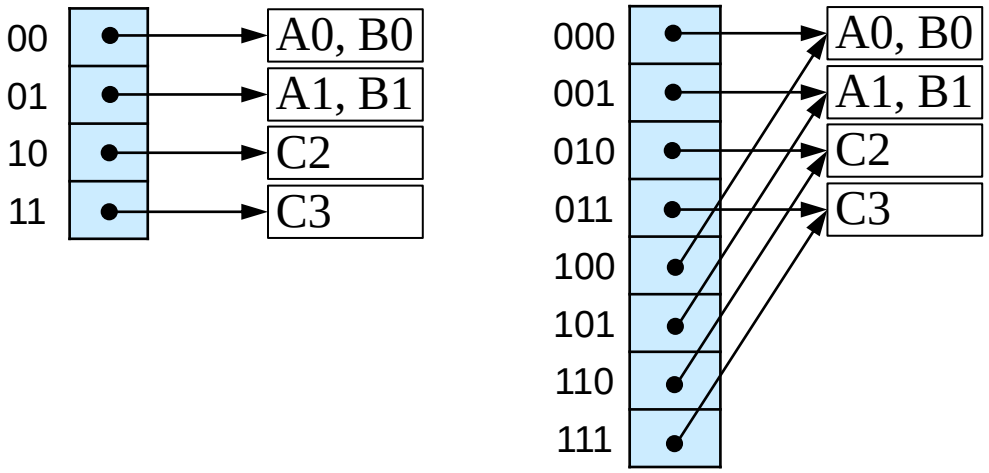


Example 1: Insert C5 [1]



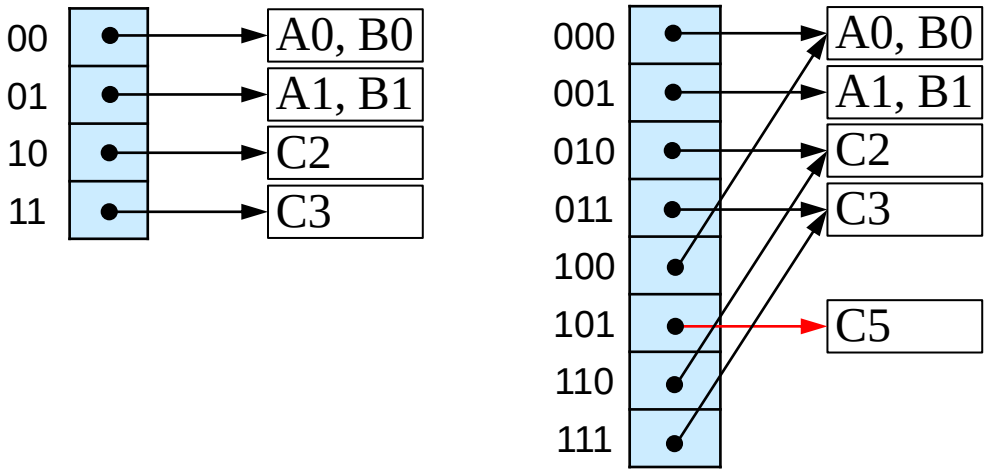
- directory depth: 2
- h(C5,2) = 01
- Examine directory entry d[01] (= d[h(C5,2)])
  - Overflow
- Determine the least u such that h(k,u) is not the same for any keys in the overflow bucket
  - h(A1) = 100001, h(B1) = 101001, & h(C5) = 110101
    So, u = 3

Example 1: Insert C5 [2]



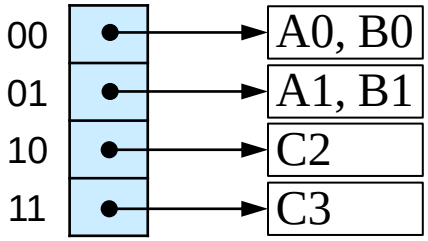
- Double the directory since u is greater than the current directory depth 2.
- Copy the pointers to the buckets so that the pointer in each half of the directory are the same.

Example1: Insert C5 [3]



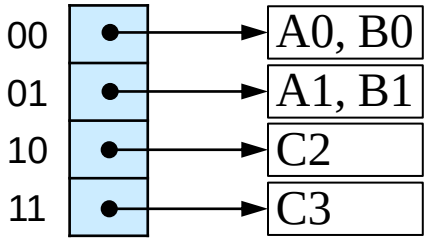
- Split the overflow bucket using h(k,3):
  - h(A1,3) = 001, h(B1,3) = 001, h(C5,3) = 101
  - Create a new bucket with C5 and place a pointer to this bucket in d[101].
- Notice: d[100] points the bucket for A0 and B0 although h(A0,3) = h(B0,3) != 000 ! Why?

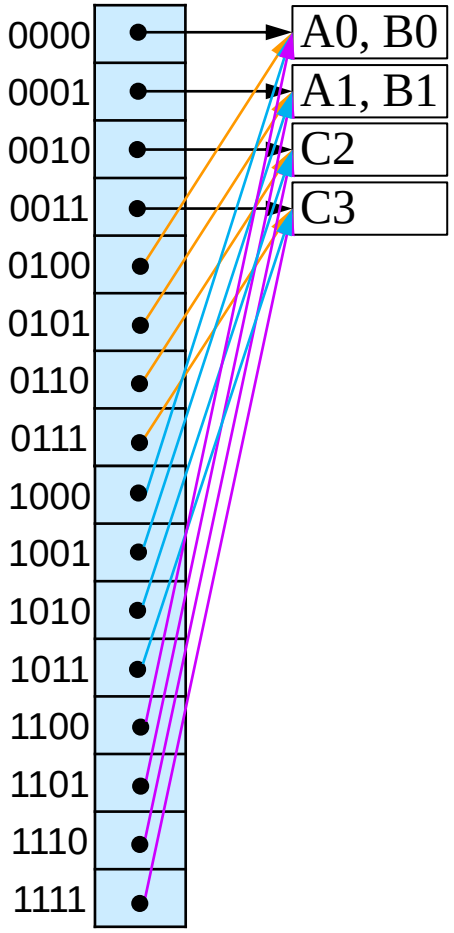
Example 2: Insert C1 [1]



- h(C1,2) = 01
- Examine directory entry d[01] (= d[h(C1,2)])
  - Overflow
- Determine the least u such that h(k,u) is not the same for any keys in the overflow bucket
  - h(A1) = 100001, h(B1) = 101001, & h(C1) = 110001
    So, u = 4

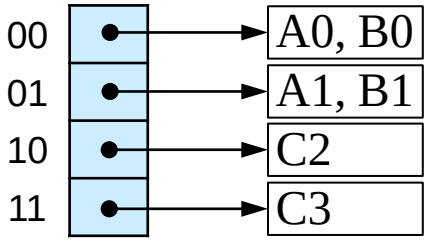
Example 2: Insert C1 [2]

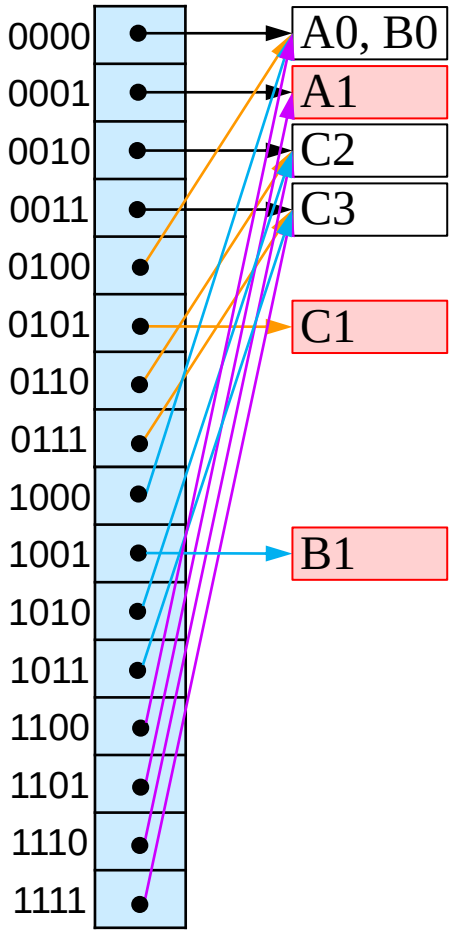




- The new directory size is 2^4 =16.
- Quadruple the directory with size 16
- Copy the bucket pointers 3 times.

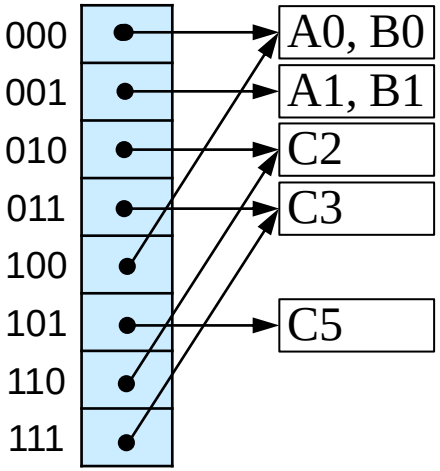
Example 2: Insert C1 [3]





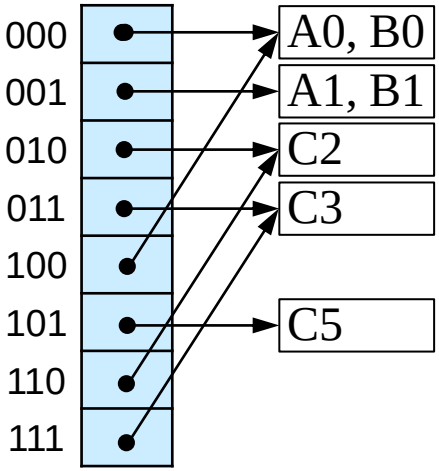
- Split the overflow bucket using h(k,4):
  - h(A1,4) = 0001, h(B1,4) = 1001, h(C5,3) = 0101
  - Remove B1 from the bucket.
  - Create a new bucket with B1 and place a pointer to this bucket in d[1001].
  - Create a new bucket with C5 and place a pointer to this bucket in d[0101].

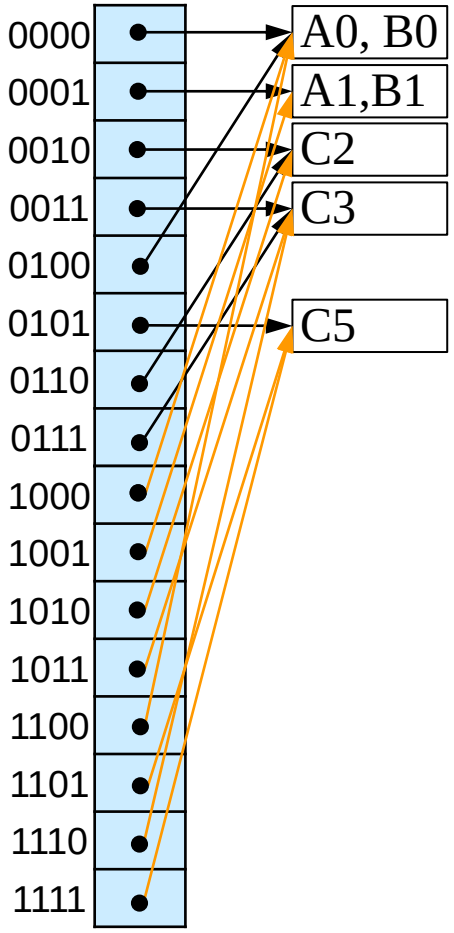
Example 3: Insert C1 [1]



- h(C1,3) = 001
- Examine the bucket pointed to by d[001].
  - Overflow
- Determine the least u:
  - h(A1) = 100001, h(B1) = 101001, h(C1) = 110001
    So, u = 4

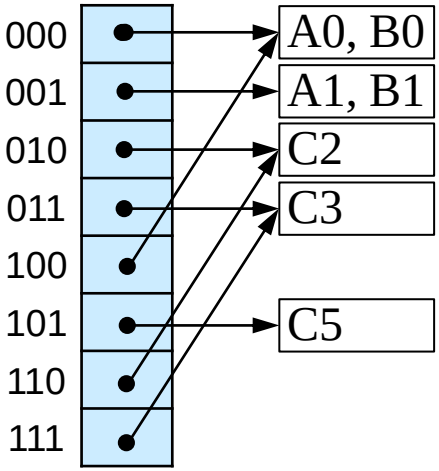
Example 3: Insert C1 [2]

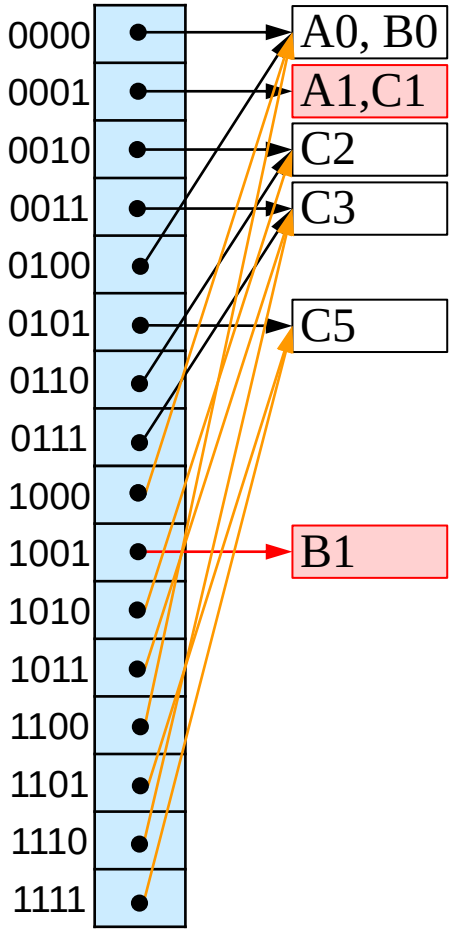




- Double the directory whit the size 16.
- Copy the pointers.

Example 3: Insert C1 [3]





- Split the overflow bucket using h(k,4):
  - h(A1,4) = 0001, h(B1,4) = 1001, & h(C1,3) = 0001
  - Remove B1 from the bucket.
  - Create a new bucket with B1 and place a pointer to this bucket in d[1001].
  - Insert C1 into the bucket pointed to by d[0001].

Summary

- Deletion: similar to insertion

- Array Doubling in dynamic hashing:
  - Need to rehash only the entries in the bucket that overflows rather than in all entries in the table.
    - (cf) static hasing

- Disk IO when buckets are on disk:
  - The directory resides in memory.
  - A search: only 1 disk access.
  - An insert: 1 read and 2 write accesses to the disk.
  - The array doubling: no disk access.

Directoryless Dynamic Hashing

No Directory!

- We do not use directory of bucket pointers.
- Instead, we use a bucket array, ht[]:
  - We assume that this array is as large as possible and so that there is no possibility of increasing its size dynamically.

- No initializing ht[] by using 2 variables q & r :
  - $0 \le q < 2^r$
  - Active buckets: ht[0] ~ ht[2^r + q -1]
    - Each active bucket is the start of a chain of buckets.
  - Overflow buckets: the remaining buckets except the start active bucket on a chain.
    - Each directory pair is either in an active or an overflow bucket.

Indexing

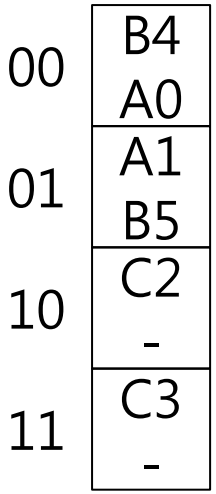
- Indexing:
  - h(k,r+1) for h[0] ~ h[q] and h[2^r] ~h[2^r+q-1]
  - h(k,r) for h[q+1] ~ h[2^r]

- Search algorithm:

```
if (h(k,r) < q) {
    search the chain that begins at bucket ht[h(k,r+1)] ;
}
else {
    search the chain that begins at bucket ht[h(k,r)] ;
}
```

Example

- Hash table ht[] when r=2, q=0 :
  - Indexing: h(k,2)

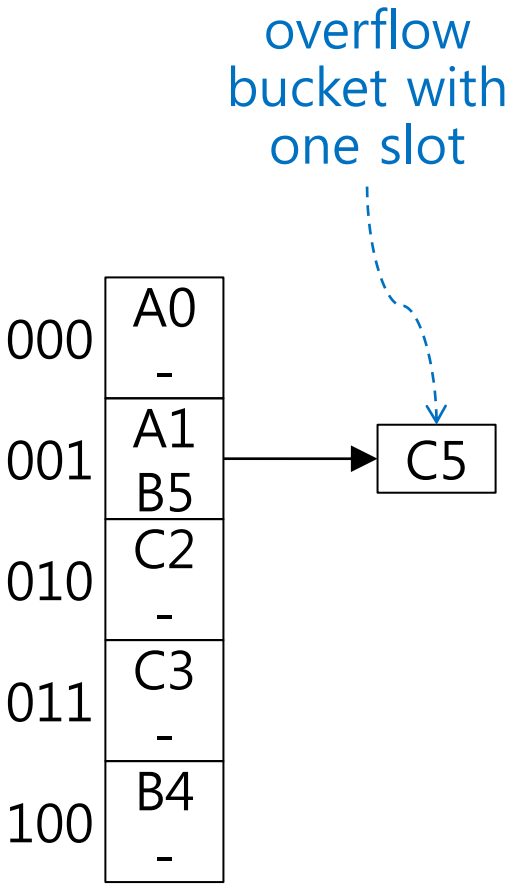


k	h(k)
A0	100 000
A1	100 001
B0	101 000
B1	101 001
B4	101 100
B5	101 101
C1	110 001
C2	110 010
C3	110 011
C5	110 101

  - 4 active buckets
  - 2 slots in each bucket
  - Currently no overflow bucket

Example

- Hash table ht[] when r=2, q=1 :
  - Indexing:
    - h(k,3) for chains ht[000] and ht[100]
    - h(k,2) for chains h[001] ~ h[011]
  - Chain ht[001] has an overflow bucket.
    - The capacity may or may not be the same as that of an active bucket.
  - How to search:
    - h(B0,r) = 00 < q:
      - Examine the chain h(B0,r+1)=100.
    - h(A1,r) = 01 >= q:
      - Examine the chain h(A1,r) =01.



Example: Insert C5 [1]

- r=2, q=0
- h(C5,2) = 01 (≥ q)
- Search the chain ht[01]:
  - Not Found.
- Overflow

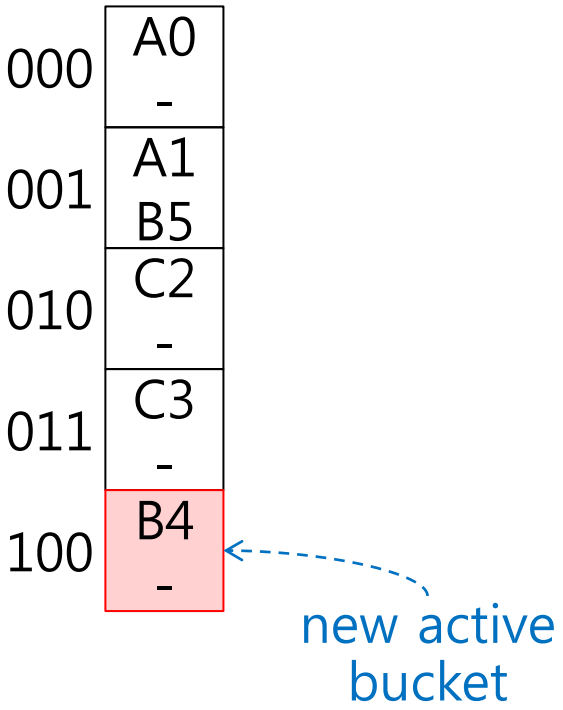
000	B4 A0
001	A1 B5
010	C2 -
011	C3 -

Example: Insert C5 [2]

- Overflow Handling:
  - Activate bucket ht[2^r+q].
  - Reallocate the entries in ht[q] (ie. in ht[0]) using h(k,r+1) either into ht[q] or into ht[2^r+q]:
    - h(B4,3) = 100: reallocate B4 into ht[100].
    - h(A0,3) = 000: no reallocation.
  - Increase q by 1:

```
q = (q+1);
if (q==r) {
    q = 0 ; r++;
}
```

    - new q is 1.

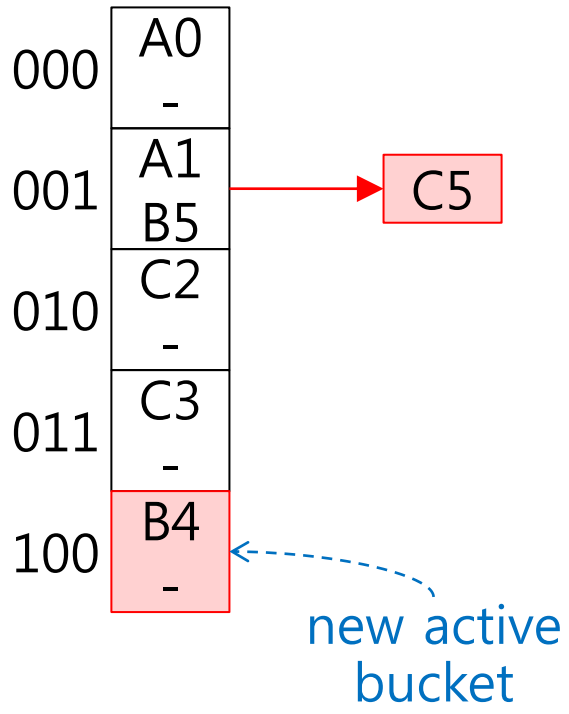


Example: Insert C5 [3]

- Overflow Handling:
  - Activate bucket ht[2^r+q].
  - Reallocate the entries in ht[q] (ie. in ht[0]) using h(k,r+1) either into ht[q] or into ht[2^r+q]:
    - h(B4,3) = 100: reallocate B4 into ht[100].
    - h(A0,3) = 000: no reallocation.
  - Increase q by 1:

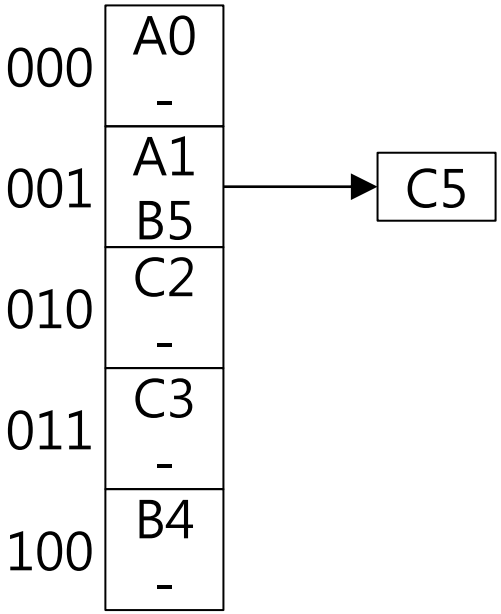
```
q = (q+1);
if (q==r) {
    q = 0 ; r++;
}
```

    - new q is 1.
  - Search C5 using new r and q:
    - h(C5,r) = 01 (>= q)
    - search the chain ht[01]:
    - Overflow: insert C5 into overflow bucket.



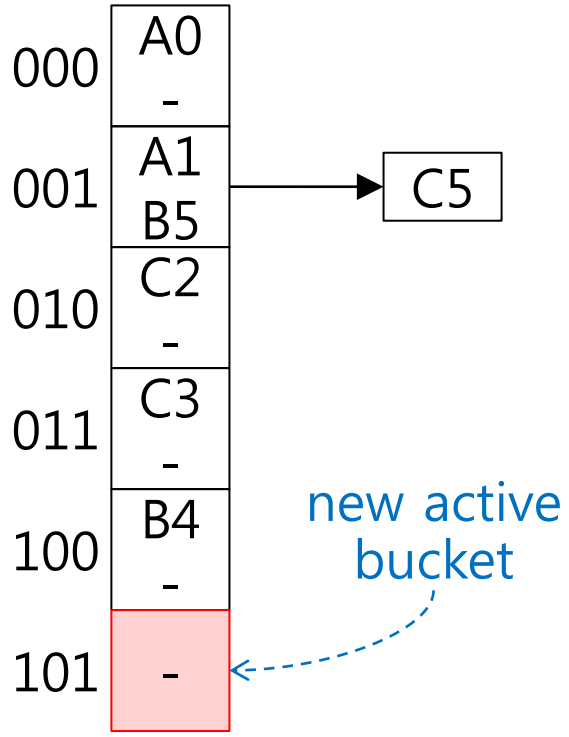
Example: Insert C1 [1]

- r=2, q=0
- h(C5,2) = 01 (≥ q)
- Search the chain ht[01]:
  - Not Found.
- Overflow



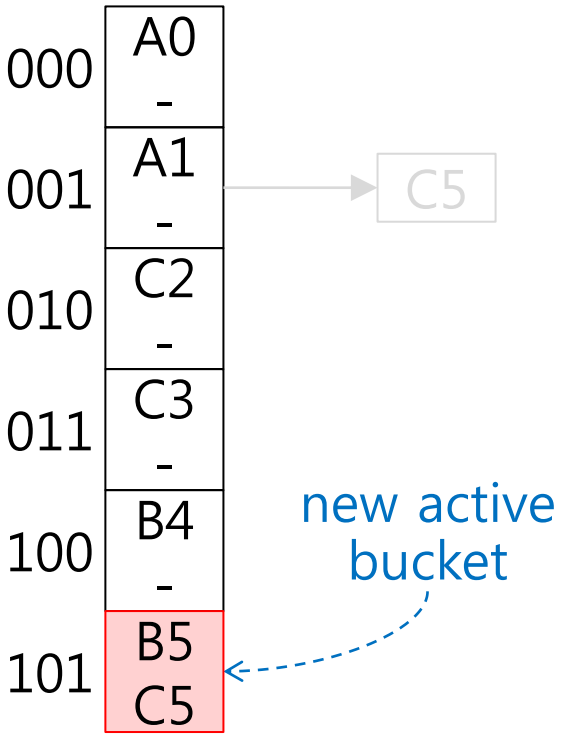
Example: Insert C1 [2]

- Overflow Handling:
  - Activate bucket ht[2^r+q].



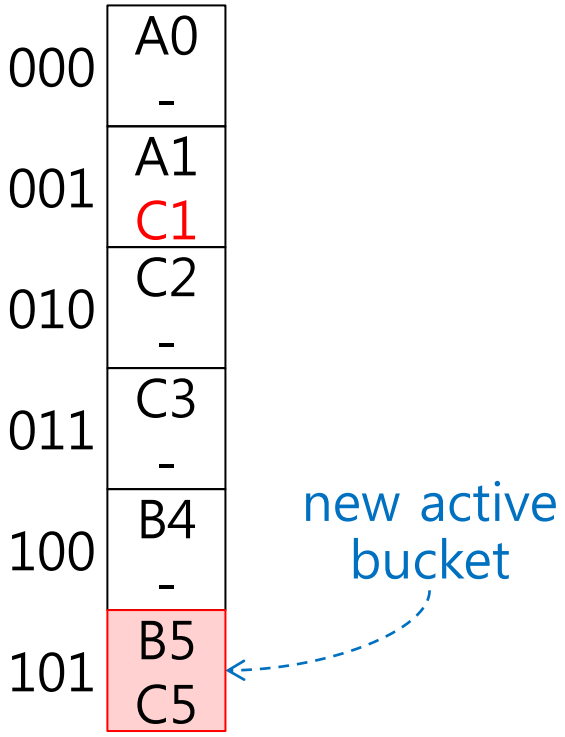
Example: Insert C1 [3]

- Overflow Handling:
  - Activate bucket ht[2^r+q].
  - Reallocate the entries in ht[q] (ie. in ht[0]) using h(k,r+1) either into ht[q] or into ht[2^r+q]:
    - h(A1,3) = 001: no reallocation
    - h(B5,3) = 101: reallocate B5 into ht[100].
    - h(C5,3) = 101
  - Increase q by 1:
    - new q is 2.



Example: Insert C1 [4]

- Overflow Handling:
  - Activate bucket ht[2^r+q].
  - Reallocate the entries in ht[q] (ie. in ht[0]) using h(k,r+1) either into ht[q] or into ht[2^r+q]:
    - h(A1,3) = 001: no reallocation
    - h(B5,3) = 101: reallocate B5 into ht[100].
    - h(C5,3) = 101
  - Increase q by 1:
    - new q is 2.
  - Search C1 using new r and q:
    - h(C1,r) = 01 (< q):
      - h(C1,r+1) = 001
    - Search the chain ht[001]:
    - No Overflow
    - Insert C1 into the active bucket ht[001].



End of Dictionary