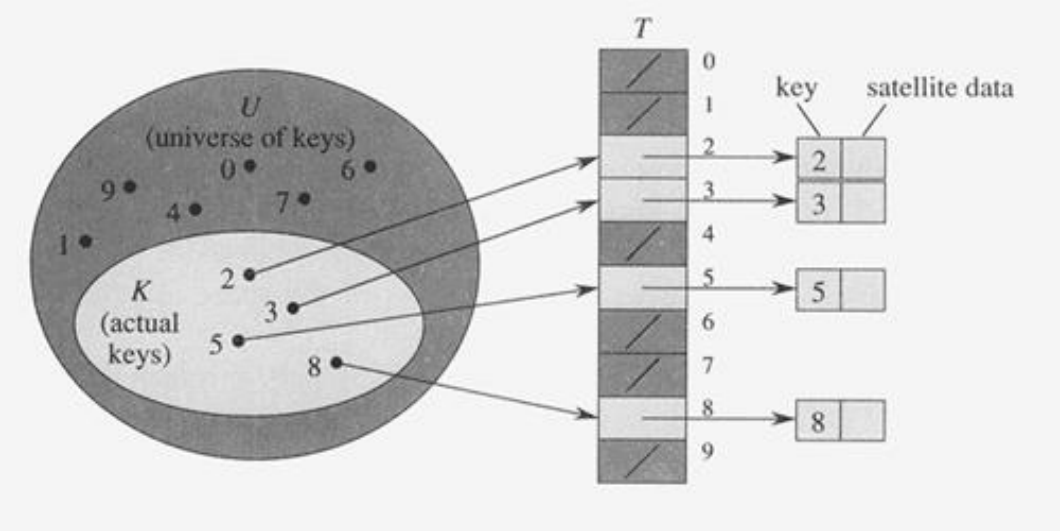
Hash Tables

Direct Address Table

- 정의
  - 모든 가능한 key 값들 (universe of keys) U = {0,1,…,m-1} 과 주소 테이블 (address table) T[0,…,m-1] 을 1:1로 직접 연결

$$T[k] = x \quad (\text{단, } key[x]=k)$$



Direct Address Table

- Operations
  - Dictionary Operation

DIRECT-ADDRESS-SEARCH(T, k)

return T[k]

: key 값이 k 인 데이터의 주소값 리턴

: O(1)

DIRECT-ADDRESS-INSERT(T, x)

T[key[x]] ← x

: 주소가 x 인 데이터를 테이블에 삽입

: O(1)

DIRECT-ADDRESS-DELETE(T, x)

T[key[x]] ← NIL

: 주소가 x 인 데이터를 테이블에서 삭제

: O(1)

Direct Address Table

- 특징
  - 방대한 메모리 필요 => U 원소값이 작은 경우에만 사용가능
  - 중복 key 사용 불가

(Ex) key = 5, 28, 19, 15, 20, 33, 12, 17, 10

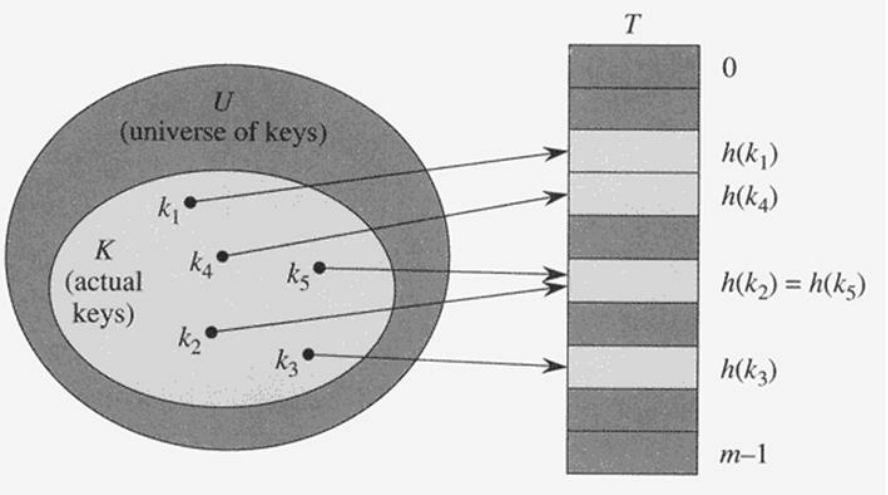
(Ex) key = apple, tomato, banana, ….

Hash Table

- 정의
  - 모든 가능한 key 값들 (universe of keys) U = {0,1,…} 과 주소 테이블 (address table) T[0,…,m-1] 을 hash function 을 사용하여 n:1로 연결

- Hash function h : U → T

  T[h[k]] = x (단, key[x]=k)



Hash Table

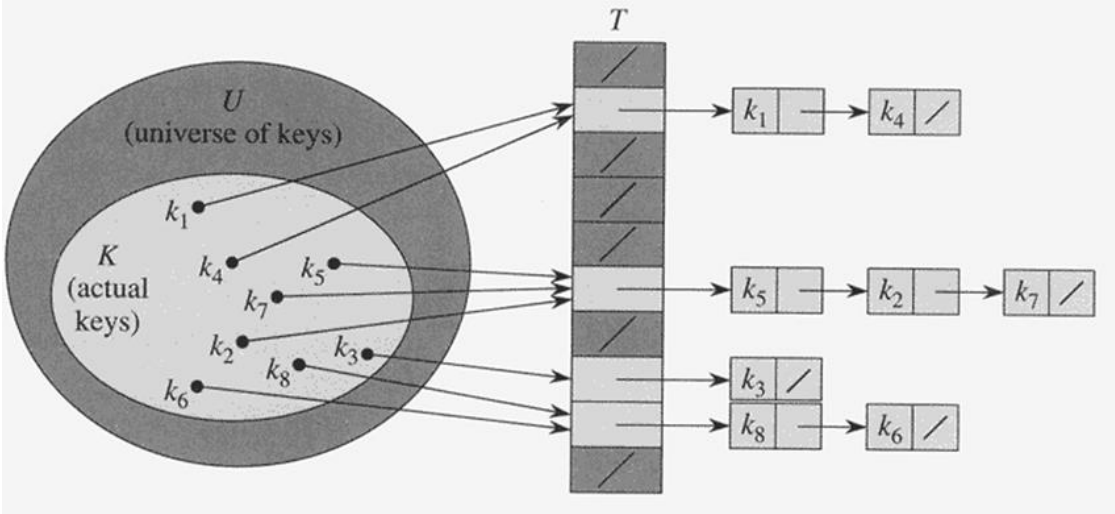
- 특징
  - 실제 사용되는 key 값들 (actual keys) 만 주소 테이블 (address table) 에 등록 => 메모리 절약
  - 중복 key 사용 가능 => Collision 문제 (h(k2) = h(k5))

(ex) key = 5, 28, 19, 15, 20, 33, 12, 17, 10

h(k) = k mod 9

Chained Hash Table

- Chaining
  - 주소테이블의 슬롯 j 에 대하여, h(k) = j 인 key 값 k 가 여러 개 존재하는 경우
    ⇒ 해당 원소들을 linked list 로 구성



T[h[k]] ← head of the linked list

(ex) key = 5, 28, 19, 15, 20, 33, 12, 17, 10
9 slots, h(k) = k mod 9

Chained Hash Table

- Operations

CHAINED_HASH_SEARCH(T,k)

: search for an element with key k in list T[h(k)]

: worst case (n 개의 모든 key 가 하나의 슬롯에 연결) Θ(n)

: average case: Θ(n/m) (m: 슬롯 수) = O(1)

CHAINED_HASH_INSERT(T,x)

: insert x at the head of list T[h(key[x])]

: O(1)

CHAINED_HASH_DELETE(T,x)

: delete x from the list T[h(key[x])]

: O(1) (doubly linked list 경우)

Hash Functions

- Division method: h(k) = k mod m

  (ex) m=12, k=100, h(k)=4

  : m 값은 2^p 값에서 떨어진 값이 바람직

- Multiplication method: h(k) = ⌊m (kA mod 1)⌋

  : 0 < A < 1

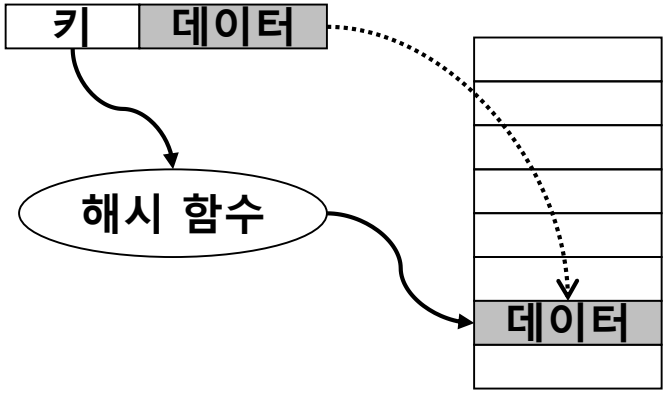
  : kA mod 1 => kA 의 소수부

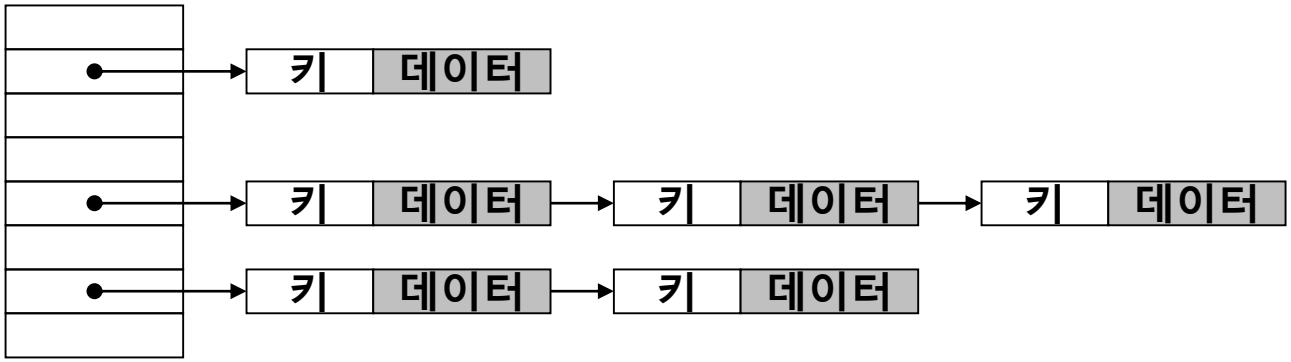
  : 임의의 m 값을 선택하여도 관계없음

- Good hash function : simple & uniform

Map Classes in MFC

- Map 동작 원리





Map 클래스

- 템플릿 클래스
  - afxtempl.h 헤더 파일

클래스 이름	데이터 타입	사용 예
CMap	프로그래머가 결정	CMap<CString, CString&, CPoint, CPoint&> map;

Map 클래스

- 비 템플릿 클래스
  - afxcoll.h 헤더 파일

클래스 이름	키 → 데이터	사용 예
CMapWordToOb	WORD → CObject 포인터	CMapWordToOb map;
CMapWordToPtr	WORD → void 포인터	CMapWordToPtr map;
CMapPtrToWord	void 포인터 → WORD	CMapPtrToWord map;
CMapPtrToPtr	void 포인터 → void 포인터	CMapPtrToPtr map;
CMapStringToOb	문자열 → CObject 포인터	CMapStringToOb map;
CMapStringToPtr	문자열 → void 포인터	CMapStringToPtr map;
CMapStringToString	문자열 → 문자열	CMapStringToString map;

Map 클래스

- 생성, 초기화, 검색

```
CMapStringToString map;
map["사과"] = "Apple";
map["딸기"] = "Strawberry";
map["포도"] = "Grape";
map["우유"] = "Milk";

CString str;
if(map.Lookup("딸기", str))
    cout << "딸기 -> " << (LPCTSTR)str << endl;
```

Map 클래스

- 순환

```
POSITION pos = map.GetStartPosition();
while(pos != NULL){
    CString strKey, strValue;
    map.GetNextAssoc(pos, strKey, strValue);
    cout << (LPCTSTR)strKey << " -> " <<
        (LPCTSTR)strValue << endl;
}
```

Map 클래스

- 삽입과 삭제

```
map.RemoveKey("우유");
map["수박"] = "Watermelon";

// 항목 삽입과 삭제 후 결과를 확인한다.
// POSITION 타입의 변수 pos는 이전의 예제에서 선언한 것이다.
pos = map.GetStartPosition();
while(pos != NULL){
    CString strKey, strValue;
    map.GetNextAssoc(pos, strKey, strValue);
    cout << (LPCTSTR)strKey << " -> " <<
        (LPCTSTR)strValue << endl;
}
```

Map 클래스

- 템플릿 맵 클래스

```
#include "stdafx.h"
#include "Console.h"
#include <afxtempl.h>

CWinApp theApp;

using namespace std;

UINT AFXAPI HashKey(CString& str)
{
    LPCTSTR key = (LPCTSTR) str;
    UINT nHash = 0;
    while(*key)
        nHash = (nHash<<5) + nHash + *key++;
    return nHash;
}
```

Map 클래스

```
int _tmain(int argc, TCHAR* argv[ ], TCHAR* envp[ ])
{
    int nRetCode = 0;

    if (!AfxWinInit(...))
    {
        // 생략 ...
    }
    else
    {
        CMap<CString, CString&, UINT, UINT&> map;
        map[CString ("사과")] = 10;
        map[CString ("딸기")] = 25;
        map[CString ("포도")] = 40;
        map[CString ("수박")] = 15;
```

Map 클래스

```
        UINT nCount;
        if(map.Lookup(CString("수박"), nCount))
            cout << "수박 " << nCount << "상자가 남아있습니다."
            << endl;
    }

    return nRetCode;
}
```