FSE598 前沿计算技术

模块 3　算法设计与分析
单元 3　高级数据结构
第 1 讲　哈希(散列)表

本课程的部分内容是基于 Thomas H.Cormen、Charles E.Leiserson 等人的“算法简介”教材

课程概要

学习内容

- 字典操作
- 直接寻址表
- 通过链寻址法解决哈希（散列）表冲突
- 哈希(散列)函数
  - 除余法
  - 乘积法
  - 通用哈希(散列)法

哈希表

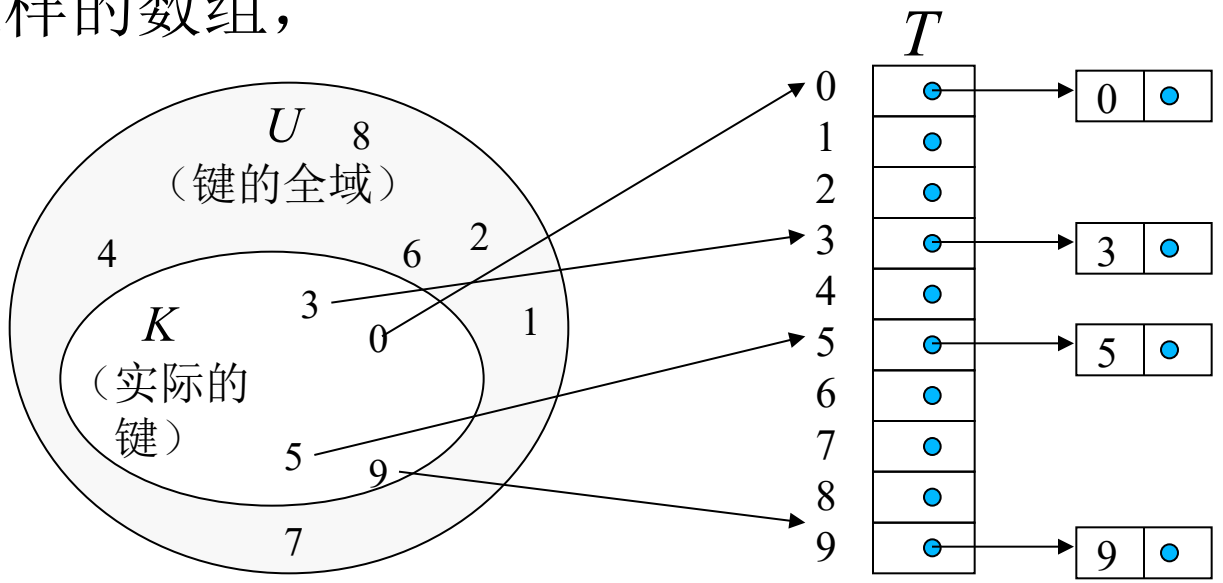
- 很多应用程序只需要常见操作的一个子集，最常见的是：
  - 搜索
  - 插入
  - 删除

  该子集被称为“**字典操作**”。Python 字典数据结构来自这里。
- 哈希表
  - 一种实现字典的高效数据结构
  - 一般数组的推广
  - 通过**键**（存储在数组元素中的有唯一性的数值）来计算索引。
  - 搜索时间：最坏情况（类似链表）运行时间为 $O(n)$，但对于很多其他情况，在 $O(1)$ 时间内即可完成

直接寻址表

直接寻址表是指这样的数组，

- 数组大小 = $|U|$
- 索引 = 键
- 操作
  时间 = O(1)



- 字典操作
  很容易

```
Search(T, k)
    return(T(k))
Insert(T, k)
    T(k) := k;
delete(T, k)
    T(k) := nil;
```

缺点：

- 当 U 很大且 K 较小时，它会浪费大量的空间。
- 示例：为每个学生预留一个停车位

示例：创建学生数据库

一个数据项包括：

- 姓名：40 个字符的字符串
- ASU ID：9 位整数
- 电子邮件：40 个字符的字符串
- 电话：10 位整数

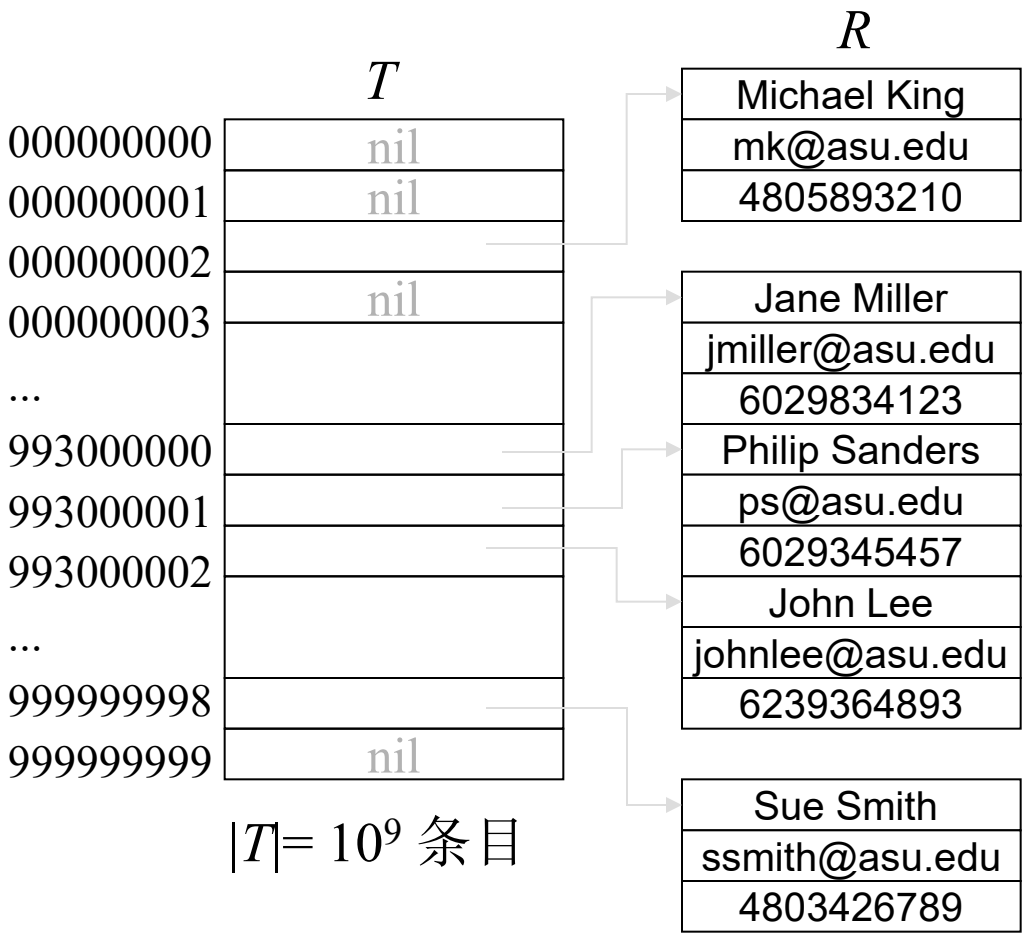
如果我们用直接寻址表

- 哪个数据字段应该是“键”？
- 该表应该包含多少个元素？

示例：采用直接寻址表的学生数据库

字典操作

```
Search(T, id)
        return(T(id).name)
Insert(T, n, e, p)
        id := getID
        T[id] := new(R)
        T[id].name := n
        T[id].email := e
        T[id].phone := p
delete(T, id)
        // free(T[id]).R
        T[id]) := nil;
```



哈希表与直接寻址表

直接寻址表

- 数组大小 = $|U|$
- 索引 = 键
- 操作时间 = O(1)

哈希表

- 大小 = $m < |U|$，如 $m = |K|$
- 索引 = h(key)，h 是一个确定性的多对一函数。典型的 h 函数是 $index = key \bmod (m)$
- 搜索时间：最坏情况的运行时间为 O(n)，但如果 h 选择得当，在很多其他情况下 O(1) 时间内即可完成。

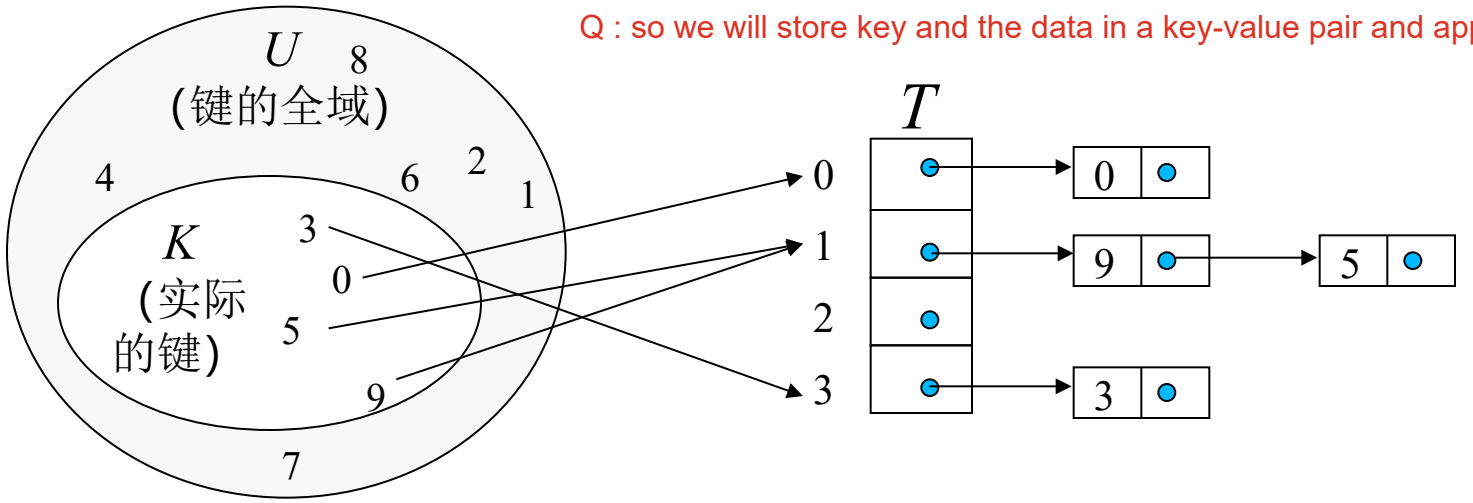
通过链寻址法解决哈希表冲突

什么是良好的哈希函数？

- 在哈希表插槽中“随机”或均匀分布键
- 易于计算

如果两个键映射到同一插槽（地址），则会发生冲突

用链（链表）来储存所有元素



Q : so we will store key and the data in a key-value pair and append to the link list?

我们如何用哈希表来创建学生数据库？

哈希表的字典操作

```
Chained-Hash-Search(T, k)
    p := T[h(k)]
    While p <> nil Do
        If p.key == k
            Then return (p)
            Else  p := p.next
    return (nil)
```

```
Chained-Hash-Insert(T, k)
    p := new(R)
    p.key := k
    p.next := T(h(k))
    T[h(k)] := p
```

插入链表首位

```
Chained-Hash-Delete(T, k)
    p := T[h(k)]
    If p <> nil Then
        If p.key == k
            Then T[h(k)] = p.next
    Else
        q := p.next
        While q <> nil Do
            If q.key == k
                Then p.next := q.next
                     return
                Else p := p.next
                     q := q.next
```

在首位找到

沿着链表搜索

最坏情况（所有键均进入一个插槽）的运行时间：

- Chained-Hash-Search(T, k) → O(n)
- Chained-Hash-Insert(T, k) → O(1)
- Chained-Hash-Delete(T, k) → O(n)

字典操作的平均性能

平均运行时间取决于

- 载荷因子 $\alpha = n/m$，即一条链中存储的平均元素数量。
- 哈希函数对键的分布情况。

- 如果哈希是一个**简单均匀**函数：所有键在 m 个插槽中的分布概率均相同，则平均搜索时间（比较次数）为 α
- 如果插槽数 m 与键数 n 成比例，即 $n = O(m)$，则
  $\alpha = n/m = O(m)/m = O(1)$
  搜索操作需要常数时间！

哈希函数

什么是良好的哈希函数？

- **简单均匀哈希**：在哈希表的插槽中以相同概率分布键。从形式上，我们可将其定义为：

$$\sum_{\forall k:h(k)=j} P(k) = \frac{1}{m} \qquad for\ j = 0,1,\cdots m-1$$

我们可以这样描述，即所有键映射到同一插槽 j 的键的概率之和是相等的 = 1/m，例如，如果有 2 个插槽，则偶数映射到 0，奇数映射到 1。概率各为 ½.

- **易于计算**：将“键”解释为自然数，例如：键“p r t”，可以用 128 个字符的 ASCII 码（base-128），如（ 112, 114, 116 ）或 $112*128^2 + 114*128 + 116 = 1,849,716$ 映射至整数。

注意：转换时间是字符串长度的函数。

哈希函数：除余法

- **除余法哈希函数：** 通过取 k 除以 m 的余数，将“键”映射到 m 个插槽中的一个：

  $$h(k) = k \bmod m$$

- 规避冲突：避免有规律的分布（创建随机效果），如避免使用特定的 m 值：
  - m 不应为 2 的幂，因为如果 $m = 2^p$，则 $h(k)$ 只取决于 k 的最低阶位，而不取决于“键”的全部位。
  - 如果该应用将十进制数字作为“键”，则 m 不应为 10 的幂。
  - $m = 2^p - 1$ 不适合于以字符作为 base-2^p 数字的字符串，例如，对于 ASCII，$p = 7$。

  m 的适当值应为不太接近 2 的幂的素数。

- 另一方面，为了便于计算，通常 $m = 2^r$ or $m = 10^r$

除余法示例

假设 ASU 有 50000 名学生，我们想用哈希表来构建学生数据库。ASU ID 为“键”。我们不介意在不成功的搜索中平均检查 3 个键。

对于 $h(k) = k \bmod m$，什么是良好的哈希函数？

- $m = 10\ 000\ (10^4)$
- $m = 9\ 999\ (10^4 - 1)$
- $m = 50\ 000\ /\ 3 = 16666.67 = 16667$
- 两个最接近的素数是：16661 和 16673
- 最终选择是
  $m = 16673$

哈希函数：乘积法

- **乘积法哈希函数**的定义是：
  - 将键 k 乘以常数 A（$0 < A < 1$），即 kA
  - 选取 kA 的分数部分，即

    $kA \bmod 1 = kA - \lfloor kA \rfloor$
  - 将分数部分乘以 m，并取下限，即

    $h(k) = \lfloor m\ (kA \bmod 1) \rfloor$
- **优点**：冲突不取决于 m 的值，我们可以选择 $m = 2^r$，计算机中的内存块通常是 2 的幂。
- **A 的择选**：A 可以取任何值，但某些 A 的值要比其他值有更好的表现。据 Knuth 建议：$A \approx (\sqrt{5}-1)/2$

  例如：$m = 10000$ 且 $k = 123456$，则

  $h(k) = 41$

哈希函数：通用哈希法

- 一个哈希函数最适合某一组“键”，但可能并不适合所有“键”。
- 恶意的对手总能找到这样一组“键”，使得所有键都映射到同一个插槽，从而造成最坏情况的发生。
- 这个想法类似于随机化快速排序：不依赖于输入模式。
- 解决方案：定义**一组**哈希函数 {h1，h2，…，hk}，每个传入的“键”会随机选择一个哈希函数对键进行哈希运算。
- 插入：从一组哈希函数中随机选择一个哈希函数 h_i
- 搜索和删除：

Q : What if when searching the slot, the same key through two different hash functions and came out to be two slot(bucket)?

须尝试所有哈希函数 for $h = h_i$ to h_k do

...

哈希表总结

本节课，我们学习了

- $|T| = |U|$ 的直接寻址表
  - 插入、搜索和删除的时间为 $O(1)$
  - 浪费空间
- 通过链寻址法解决哈希表冲突
  - 搜索时间：最坏情况的运行时间为 $O(n)$，但在很多其他情况下，也可以在 $O(1)$ 时间内完成。
  - 动态数据分配
  - 指针的空间浪费
- 哈希函数
  - 除余法、乘积法和通用哈希法