散列表(字典) 散列函数 冲突. 同义词

对关键字进行某种运算得到地址 即 H(key)

①散列函数的构造方法

I、数字分析法 II、平方取中法 III、折叠法

IV. 除留余数法 (P为小于等于表长的最大质数)

②处理冲突的方法

I. 开放地址法 (发生冲突时, 用特定计算方法 $H_0 \to H_1 \to \cdots$ 直到不再冲突)

$H_i = (H(key) + d_i)$

1)线性探测法 $d_i = 1, 2, 3, \cdots m-1$ (循环逐个寻找)

2)二次探测法 $d_i = 1^2, -1^2, 2^2, -2^2, \cdots, +k^2, -k^2$

$(k \le \frac{m}{2})$

对取余法 $H_i = (key \% m + d_i) \% m$

☆☆☆ 注意、有相同数据 直接同位存储, 不需探测!

3)伪随机探测 d_i = 伪随机数序列

II、链地址法 每个键对应位置为链表, 键同的值在链表

III、双重散列 两个简的函数 hash1(key) hash2(key)

hash1()不发生用时, 跳过 hash2() 个槽位

pos = (pos + step) % size_of_table)

二、KMP算法　字符串查找（S中是否含子串w）

从主串中查找模式串是否为其子串，利用next数组使主串指针不回退地查找，线性复杂度

```python
def build_next(patt):
    """
    计算 Next 数组
    """

    next = [0]  # next 数组（初值元素一个 0）
    prefix_len = 0  # 当前共同前后缀的长度
    i = 1
    while i < len(patt):
        if patt[prefix_len] == patt[i]:
            prefix_len += 1
            next.append(prefix_len)
            i += 1
        else:
            if prefix_len == 0:
                next.append(0)
                i += 1
            else:
                prefix_len = next[prefix_len - 1]
    return next
```

```python
def kmp_search(string, patt):
    next = build_next(patt)  # 假设我们已经算出了 next 数组（马上
    i = 0  # 主串中的指针
    j = 0  # 子串中的指针
    while i < len(string):
        if string[i] == patt[j]:  # 字符匹配，指针后移
            i += 1
            j += 1
        elif j > 0:  # 字符失配，根据 next 跳过子串前面的一些字符
            j = next[j - 1]
        else:  # 子串第一个字符就失配
            i += 1

        if j == len(patt):  # 匹配成功
            return i - j
```

$j = LPS[j-1]$

LPS最长公共真前后缀