1． 如何判断两个文件是否相同

也叫信息摘要算法,散列算法

1.1 1.逐字节比较

效率低

1.2 2.使用hash算法

对两个文件进行hash运算,比较两个结果.效率高,只机算完两个比较一次

前提,每个文件进行hash运算得到的是独一无二的值.如果两个不同文件进行hash运算得到的结果相同,叫做发生碰撞

2． hash算法能不能用来做加密

不能.因为不可逆

3． 现在有一个key,如何做到通过key实现随机访问数据

对key进行hash运算,将结果放到数组中,用数组的下标索引来实现随机访问数据

3.1 如何解决碰撞问题

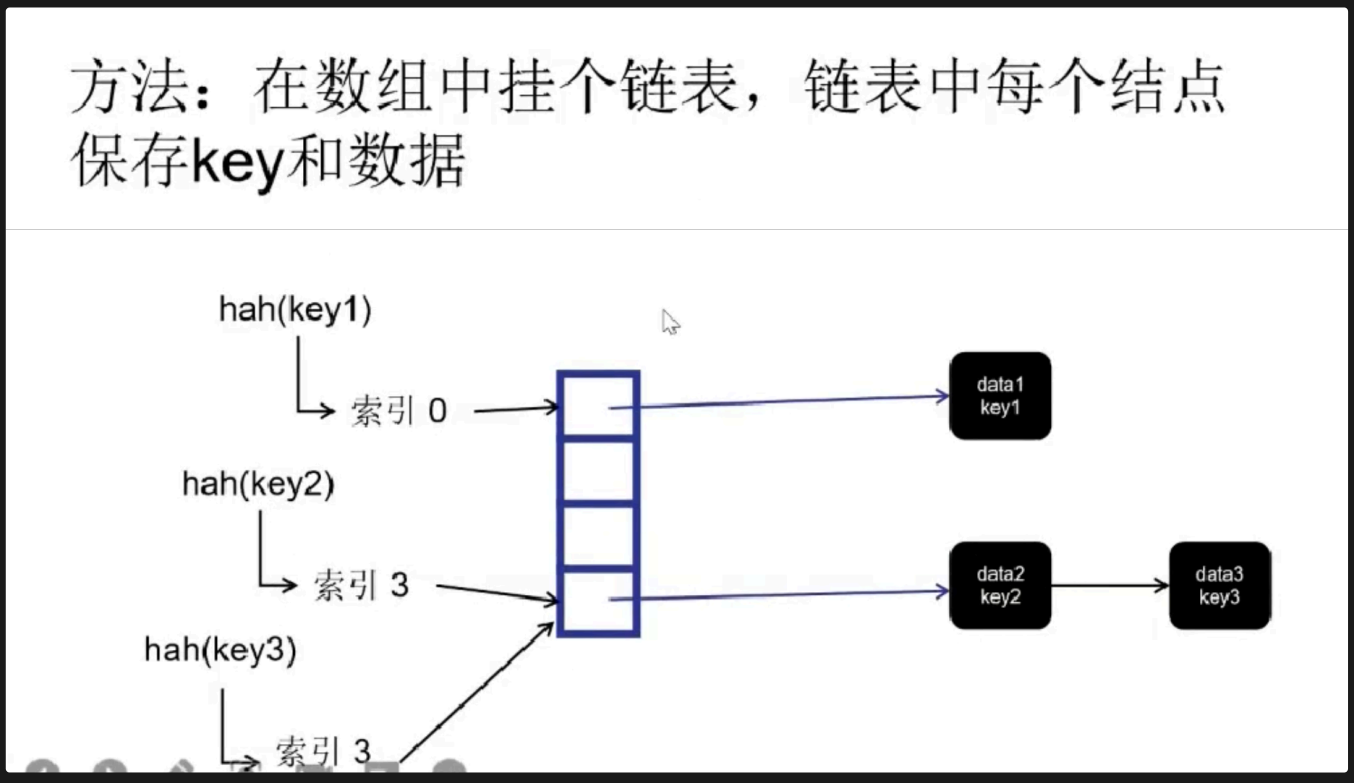

Figure 3-1

如果计算出是链表,就挂到索引数组的后面

4. 官方库的hash表

```cpp
#include <iostream>
#include <unordered_map>
#include "CHash.h"
using namespace std;

int main()
{
#if 0
    unordered_map<string, int> h;
    h.insert(pair<string, int>{"xiaohua", 18});
    h.insert(pair<string, int>{"xiaobai", 19});

    h.erase("xiaohua");

    h["xiaobai"] = 25;

    auto itr = h.find("xiaohua");
    itr = h.find("xiaobai");

    h["张三"] = 98;
#endif // 0
```

Figure 4-1

4.1 CHash.h

```cpp
#pragma once
#include <string>
using namespace std;

#define HT_LEN 4

class CHash
{
private:
    struct NODE
    {
        NODE() :m_pNext(nullptr), m_strKey{}, m_nVal{} {}
```

```
        NODE(const string& strKey, int nVal):m_pNext(nullptr), m_strKey(strKey), m_nVal(nVal) {}
        string m_strKey;
        int m_nVal;
        NODE* m_pNext;
    };
    using PNODE = NODE*;

public:

    CHash() {}
    CHash(const CHash& obj);
    CHash(CHash&& obj);
    CHash& operator=(const CHash& obj) noexcept;
    virtual ~CHash();

    void Insert(const string& strKey, int nVal);
    void Delete(const string& strKey, int nVal);

    int& operator[](const string& strKey);
    bool Find(const string& strKey);

private:
    size_t GetIdx(const string& strKey);
    PNODE FindKey(const string& strKey);

    PNODE m_aryHashTable[HT_LEN] = {};// 结点数组
};
```

Fence 4-1

4.2 CHash.cpp

```
#include "CHash.h"
#include <functional>

CHash::CHash(const CHash& obj)
{
}

CHash::CHash(CHash&& obj)
{
}

CHash& CHash::operator=(const CHash& obj) noexcept
{
    return *this;
}

CHash::~CHash()
{
}

void CHash::Insert(const string& strKey, int nVal)
{
    // 不允许插入重复的Key
    auto pNode = FindKey(strKey);
    if (pNode != nullptr)
    {
```

```
27          pNode->m_nVal = nVal;
28          return;
29      }
30
31      size_t nIdx = GetIdx(strKey);// 获取下标索引
32
33      // 创建新结点
34      auto pNew = new NODE(strKey, nVal);
35      if (pNew == nullptr)
36      {
37          return;
38      }
39      // 新结点插入到链表头
40      pNew->m_pNext = m_aryHashTable[nIdx];
41      m_aryHashTable[nIdx] = pNew;
42  }
43
44  void CHash::Delete(const string& strKey, int nVal)
45  {
46      size_t nIdx = GetIdx(strKey);// 获取下标索引
47      auto pNode = FindKey(strKey);
48      if (pNode == nullptr)    // 没有找到key
49      {
50          pNode->m_nVal = nVal;
51          return;
52      }
53
54      // 删除.与头结点的值交换,删除头结点
55      auto pHead = m_aryHashTable[nIdx];
56      pNode->m_nVal = pHead->m_nVal;
57      pNode->m_strKey = pHead->m_strKey;
58
59      // 删除头结点
60      m_aryHashTable[nIdx] = pHead->m_pNext;
61      delete pHead;
62      return;
63  }
64
65  int& CHash::operator[](const string& strKey)
66  {
67      auto pNode = FindKey(strKey);
68      // 找到了
69      if (pNode != nullptr)
70      {
71          return pNode->m_nVal;
72      }
73
74      // 没有找到
75      auto pNew = new(nothrow) NODE;
76      if (pNew == nullptr)
77      {
78          throw bad_alloc();
79      }
80      pNew->m_strKey = strKey;
81      // 新结点插入到链表头
82      pNew->m_pNext = m_aryHashTable[GetIdx(strKey)];
83      m_aryHashTable[GetIdx(strKey)] = pNew;
84
```

```
    return pNew->m_nVal;
}

bool CHash::Find(const string& strKey)
{
    return FindKey(strKey) != nullptr;
}

size_t CHash::GetIdx(const string& strKey)
{
    size_t nHash = hash<string>{}(strKey);// 临时对象调用仿函数.获取hash值
    size_t nIdx = nHash % HT_LEN;// 获取下标索引

    return nIdx;
}

typename CHash::PNODE CHash::FindKey(const string& strKey)
{
    // 获取索引
    size_t nIdx = GetIdx(strKey);

    // 找到这个Key
    PNODE pNode = m_aryHashTable[nIdx];
    while (pNode != nullptr)
    {
        if (pNode->m_strKey == strKey)
        {
            return pNode;
        }
        pNode = pNode->m_pNext;
    }
    return nullptr;
}
```

Fence 4-2

4.3 哈希表.cpp

```
// 哈希表.cpp

#include <iostream>
#include <unordered_map>
#include "CHash.h"
using namespace std;

int main()
{
#if 0
    unordered_map<string, int> h;
    h.insert(pair<string, int>{"xiaohua", 18});
    h.insert(pair<string, int>{"xiaobai", 19});

    h.erase("xiaohua");

    h["xiaobai"] = 25;

    auto itr = h.find("xiaohua");
    itr = h.find("xiaobai");

```

```
    h["张三"] = 98;
#endif // 0

    CHash h;
    h.Insert("hello", 18);
    h.Insert("world", 17);
    h.Insert("zhangsan", 19);
    h.Insert("李四", 13);
    h.Insert("王五", 14);
    h.Insert("赵六", 12);

    h.Delete("zhangsan", 19);
    h.Delete("王五", 14);
    h.Delete("hello", 18);
    h.Delete("world", 17);

    h["lisi"] = 99;
    h["dabai"] = 87;

    return 0;
}
```

Fence 4-3

5. 哈希算法的时间复杂度是多少

插入:常量阶

访问:常量阶

查询:常量阶