开散列法

开散列法简介

- 在每个存放数据的地方开一个链表，如果有多个键值索引到同一个位置，只用把它们都一次存放代该位置的链表中即可
- 时间复杂度分析
    - 假设索引的范围是$1,2,\cdots,M$，哈希表的大小为N
    - 每次插入/查询需要进行期望$O(\frac{N}{M})$次比较

开散列法实现

实现模板

```
const int SIZE = 1000000;
const int M = 999997;

struct HashTable
{
    struct Node
    {
        int next, value, key;
    } data[SIZE]; // 定义数据域：包括键、值以及

    int head[M], index; // 链表的头指针

    int f(int key) { return key % M; }   // 该函数的作用在于获取键所对应的值应该在哪条链上

    int get(int key)   // 获取值
    {
        for (int p = head[f(key)]; p; p = data[p].next)   // 本循环只会在找到对应链的末尾仍然没有找到对应值的情况下才会主动终止
            if (data[p].key == key)
                return data[p].value;
        return -1;
    }

    int modify(int key, int value)  // 修改键对应的值，思路类似于 get 函数
    {
        for (int p = head[f(key)]; p; p = data[p].next)
            if (data[p].key == key)
                return data[p].value = value;
    }

    int add(int key, int value)   // 将一个键值对加入到哈希表中
    {
        if (get(key) != -1)    // 如果这个键已经存在了，直接返回-1
```

```
            return -1;
        data[++index] = (Node){head[f(key)], value, key};  // 将结构体数组中进行相应
的更新
        head[f(key)] = index;   // index事实上是一个节点在结构体数组中的编号（索引）
        return value;
    }
};
```

开散列法模板详解

基于开散列法的哈希表的本质

- 基于开散列表的哈希表的本质应该是一个利用另一个数组进行维护的**结构体**数组
- **另一个**数组指的就是上述代码中的head数组
- 可以理解为我们在一个数组中的均匀间隔位置处打上了不同的“标签”，或是在一本页码连续的书中加上均匀书签

查询过程详解

- `for (int p = head[f(key)]; p; p = data[p].next)`
  - `next`事实上是上一个`f(key)`值相同的键的索引（详见添加过程）
  - 如果没有找到，每次都往上次添加的`f(key)`值相同的键查询，直到找不到这样的键，最初添加的一批键由于`head[f(key)]`还未被赋值（默认为0），因此不会再往回查找，而是返回-1
- `if (data[p].key == key) return data[p].value;`
  - 表示已经找到键，返回值

添加过程详解

- `data[++index] = (Node){head[f(key)], value, key};`
  - 新添加的键直接加入`data`数组中，`index`事实上就是这个键的索引
- `head[f(key)] = index;`
  - 将自己的索引直接更新到`head`数组中
  - 下一个`f(key)`值相同的键的`next`将会是自己的索引

程序清单

- `code2.cpp`，在`/coding`路径下
- 其中包含了基于开散列法实现的结构体