哈希表

题目描述

刷新

map 类是stl库中经常用到的一类结构，它实现了构建两个任意类型变量的映射关系，但它的实现是基于比较的，每次查询可能需要多达十几次比较。
HashTable 也实现了同样的功能——将任意类型变量 A 映射到任意类型变量 B ，但效率更高。
具体而言，它的实现原理，是将 A 通过一个固定的算法函数——称为**哈希算法**（ Hash ），得到一个数值，称为**哈希值**，通常将这个值作为数组下标，将 B 当作数组中对应的元素。 A 被称为**关键字**（ key ）， B 被称为**关键字值**（ value ），构造的映射关系称为**键值对**。
举个例子：
假设固定对 string 类型的哈希算法为：从下标 0 到 length-1 ，将每个字符的 ASCII 码相加，最后对数组长度 n 取模，得到数组下标。
当 n =10时，字符串 ABC 的哈希值就是
$(int('A') + int('B') + int('C')) \mod 10 = (65 + 66 + 67) \mod 10 = 8$
因此，构建 ABC 映射到 XYZ 的关系时， XYZ 就被放在数组下标为8的位置。
从这个例子上我们可以发现两点：

- 长度为3的字符串的个数就已经远超过10个，而上述算法是对于任意长度的字符串都能使用，得到的结果均为 0-9 。因此，哈希算法是一种压缩算法，也就是，哈希值的空间通常远小于输入的空间。
- 不难看出，上述算法中 ABC 和 CBA 对应的哈希值相等，因此查询 ABC 和 CBA 会出现冲突，这就是**哈希冲突**，这是压缩算法无法避免的问题，但在本题中不需要考虑哈希冲突的问题。

如果想要深入了解 map 或 hash table 的原理，可以参考以下网址：
how does a hash table work (https://stackoverflow.com/questions/730620/how-does-a-hash-table-work)
哈希表（散列表）原理详解 (https://blog.csdn.net/duan19920101/article/details/51579136)
map和HashMap原理详解 (https://blog.csdn.net/city_to_sky/article/details/80042586)

需要再给出的模板函数类 Hash 的基础上，完善并实现模板类 HashTable 。

HashClass.h

```
template <typename T>
class Hash
{
private:
    int n;

public:
    Hash(int _n) : n(_n) {}
    int operator()(const T &x);
};

template <typename T1, typename T2>
class HashTable
{
public:
    void addItem(const T1 &key, const T2 &value);
    void removeItem(const T1 &key);
    T2 *findByKey(const T1 &key);
};
```

HashFunc.h

```
template<>
int Hash<int>::operator()(const int &x)
{
    return x % n;
}

template<>
int Hash<std::string>::operator()(const std::string &x)
{
    int ret = 0;
    for (char c : x)
    {
        ret += static_cast<int>(c);
        ret %= n;
    }
    return ret;
}
```

文件下载地址：下载链接 (/staticdata/1987.FWysYitdNe6pzk5X.pub/WykEGVajOesJVWCV.download.zip/download.zip)

输入格式

第一行包含用空格分隔的四个正整数 `n1` ， `n2` ， `n3` ， `n4` ，表示

- `int` 映射到 `int` 的哈希表元素总个数不超过 `n1` 。
- `int` 映射到 `string` 的哈希表元素总个数不超过 `n2` 。
- `string` 映射到 `int` 的哈希表元素总个数不超过 `n3` 。
- `string` 映射到 `string` 的哈希表元素总个数不超过 `n4` 。

接下来一行包含一个正整数 `p` ，表示之后有 `p` 个操作。
对每个操作给出一行，包含一个正整数 `opr` ，和若干个参数，用空格分隔， `opr` 表示操作类型，如下：

- `opr` 为1时，参数为两个整数 `a` 和 `b` ，表示建立 `a` 到 `b` 的映射关系；
- `opr` 为2时，参数为一个整数 `a` 和一个字符串 `b` ，表示建立 `a` 到 `b` 的映射关系；
- `opr` 为3时，参数为一个字符串 `a` 和一个整数 `b` ，表示建立 `a` 到 `b` 的映射关系；
- `opr` 为4时，参数为两个字符串 `a` 和 `b` ，表示建立 `a` 到 `b` 的映射关系；
- `opr` 为-1时，参数为一个整数 `a` ，表示删除 `a` 到整数的映射关系；
- `opr` 为-2时，参数为一个整数 `a` ，表示删除 `a` 到字符串的映射关系；
- `opr` 为-3时，参数为一个字符串 `a` ，表示删除 `a` 到整数的映射关系；
- `opr` 为-4时，参数为一个字符串 `a` ，表示删除 `a` 到字符串的映射关系；

接下来一行包含一个正整数 `q` ，表示之后有 `q` 个查询。
对每个操作给出一行，包含一个正整数 `opr` ，和另一个参数，用空格分隔， `opr` 表示操作类型，如下：

- `opr` 为1时，参数为整数 `a` ，表示输出 `a` 映射到的整数；
- `opr` 为2时，参数为整数 `a` ，表示输出 `a` 映射到的字符串；
- `opr` 为3时，参数为字符串 `a` ，表示输出 `a` 映射到的整数；
- `opr` 为4时，参数为字符串 `a` ，表示输出 `a` 映射到的字符串；

输出格式

对每个查询输出一行，包含查询到的整数或字符串，若查询的映射关系不存在，则输出 `NULL` 。

数据规模和约定

$1 \leq n1, n2, n3, n4 \leq 5000 \, , 1 \leq p, q \leq 10000$
所有输入的字符串长度不大于100
不需考虑哈希冲突相关问题，如果出现关键字哈希值相同，则之后出现的映射关系覆盖之前的映射。

时间限制：1s

空间限制：256MB

输入样例

```
10 1 1 10
2
1 8 10
4 aut bst
5
1 9
1 8
4 aut
3 tjh
4 gc
```

输出样例

```
NULL
10
bst
NULL
NULL
```

提交格式

根据提供的 `main.cpp` 、 `HashClass.h` 和 `HashFunc.h` 的内容，编写 `HashTable` 类的代码并完善 `HashClass.h` 与 `HashFunc.h` 。
应该将文件打包成一个zip压缩包并上传。评测时，OJ会将提供的 `main.cpp` 和 `Makefile` 贴入目录下进行编译并执行。

语言和编译选项

#	名称	编译器	额外参数	代码长度限制
0	custom	make		1048576 B

递交历史

#	状态	时间

1

递交答案

语言和编译选项 custom

```
1
```

提交

文件请拖入编辑器中，或 上传文件