【第十四课】哈希表与布隆过滤器

```
/*************************************************************************
  > File Name: 1.hash_table.cpp
  > Author: huguang
  > Mail: hug@haizeix.com
  > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

class HashTable {
public:
    HashTable(int n = 100) : flag(n), data(n), cnt(0) {}
    void insert(string s) {
        int ind = hash_func(s) % data.size(); // 计算哈希值
        recalc_ind(ind, s); // 冲突处理
        if (flag[ind] == false) {
            data[ind] = s;
            flag[ind] = true;
            cnt += 1;
            if (cnt * 100 > data.size() * 75) {
                expand();
            }
        }
        return ;
    }
    bool find(string s) {
        int ind = hash_func(s) % data.size(); // 计算哈希值
        recalc_ind(ind, s); // 冲突处理
        return flag[ind];
    }

private:
    int cnt;
    vector<string> data;
    vector<bool> flag;

    void expand() {
        int n = data.size() * 2;
        HashTable h(n);
        for (int i = 0; i < data.size(); i++) {
            if (flag[i] == false) continue;
            h.insert(data[i]);
        }
        *this = h;
        return ;
    }
    int hash_func(string &s) {
        int seed = 131, hash = 0;
        for (int i = 0; s[i]; i++) {
            hash = hash * seed + s[i];
        }
        return hash & 0x7fffffff;
    }
    void recalc_ind(int &ind, string &s) {
        int t = 1;
        while (flag[ind] && data[ind] != s) {
            ind += t * t;
            t += 1;
            ind %= data.size();
```

kaikeba
开课吧

```
        }
        return ;
    }
};

int main() {
    int op;
    string s;
    HashTable h;
    while (cin >> op >> s) {
        switch (op) {
            case 1: h.insert(s); break;
            case 2: cout << "find " << s << " : " << h.find(s) << endl; break;
        }
    }
    return 0;
}
```

```
/*************************************************************************
  > File Name: 1.hash_table.cpp
  > Author: huguang
  > Mail: hug@haizeix.com
  > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

class HashTable {
public:
    HashTable(int n = 100) : flag(n), data(n), cnt(0) {}
    void insert(string s) {
        int ind = hash_func(s) % data.size(); // 计算哈希值
        recalc_ind(ind, s); // 冲突处理
        if (flag[ind] == false) {
            data[ind] = s;
            flag[ind] = true;
            cnt += 1;
            if (cnt * 100 > data.size() * 75) {
                expand();
            }
        } else if (data[ind] != s) {
            buff.insert(s);
        }
        return ;
    }
    bool find(string s) {
        int ind = hash_func(s) % data.size(); // 计算哈希值
        recalc_ind(ind, s); // 冲突处理
        if (flag[ind] == false) return false;
        if (data[ind] == s) return true;
        return buff.find(s) != buff.end();
    }

private:
    int cnt;
    vector<string> data;
    vector<bool> flag;
    set<string> buff;

    void expand() {
        int n = data.size() * 2;
        HashTable h(n);
        for (int i = 0; i < data.size(); i++) {
            if (flag[i] == false) continue;
            h.insert(data[i]);
        }
        for (auto x : buff) {
            h.insert(x);
        }
        *this = h;
        return ;
    }
    int hash_func(string &s) {
        int seed = 131, hash = 0;
        for (int i = 0; s[i]; i++) {
            hash = hash * seed + s[i];
        }
        return hash & 0x7fffffff;
    }
    void recalc_ind(int &ind, string &s) {
        return ;
    }
};

int main() {
    int op;
    string s;
    HashTable h;
    while (cin >> op >> s) {
        switch (op) {
            case 1: h.insert(s); break;
            case 2: cout << "find " << s << " : " << h.find(s) << endl; break;
        }
    }
    return 0;
}
```

```
/*************************************************************************
  > File Name: 1.hash_table.cpp
  > Author: huguang
  > Mail: hug@haizeix.com
  > Created Time:
 ************************************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <queue>
#include <stack>
#include <algorithm>
#include <string>
#include <map>
#include <set>
#include <vector>
using namespace std;

class Node {
public :
    Node(string data = "", Node *next = nullptr) : data(), next(nullptr) {}
    string data;
    Node *next;
    void insert(Node *node) {
        node->next = this->next;
        this->next = node;
        return ;
    }
};

class HashTable {
public:
    HashTable(int n = 100) : data(n), cnt(0) {}
    void insert(string s) {
        int ind = hash_func(s) % data.size(); // 计算哈希值
        recalc_ind(ind, s); // 冲突处理
        Node *p = &data[ind];
        while (p->next && p->next->data != s) p = p->next;
        if (p->next == nullptr) {
            p->insert(new Node(s));
            cnt += 1;
            if (cnt > data.size() * 3) expand();
        }
        return ;
    }
    bool find(string s) {
        int ind = hash_func(s) % data.size(); // 计算哈希值
        recalc_ind(ind, s); // 冲突处理
        Node *p = data[ind].next;
        while (p && p->data != s) p = p->next;
        return p != nullptr;
    }

private:
    int cnt;
    vector<Node> data;

    void expand() {
        int n = data.size() * 2;
        HashTable h(n);
        for (int i = 0; i < data.size(); i++) {
            Node *p = data[i].next;
            while (p) {
                h.insert(p->data);
                p = p->next;
            }
        }
        *this = h;
        return ;
    }
    int hash_func(string &s) {
        int seed = 131, hash = 0;
        for (int i = 0; s[i]; i++) {
            hash = hash * seed + s[i];
        }
        return hash & 0x7fffffff;
    }
    void recalc_ind(int &ind, string &s) {
        return ;
    }
};

int main() {
    int op;
    string s;
    HashTable h;
    while (cin >> op >> s) {
        switch (op) {
            case 1: h.insert(s); break;
            case 2: cout << "find " << s << " : " << h.find(s) << endl; break;
        }
    }
    return 0;
}
```

705. 设计哈希集合

```
class Node {
public:
    Node(int key = 0, Node *next = nullptr) : key(key), next(next) {}
    int key;
    Node *next;
    void insert_after(Node *node) {
        node->next = this->next;
        this->next = node;
        return ;
    }
    void remove_after() {
        if (this->next == nullptr) return ;
        Node *p = this->next;
        this->next = this->next->next;
        delete p;
        return ;
    }
};

class MyHashSet {
public:
    /** Initialize your data structure here. */
    vector<Node> data;
    MyHashSet() : data(100) {}

    int hash_func(int key) { return key & 0x7fffffff; }

    void add(int key) {
        if (contains(key)) return ;
        int ind = hash_func(key) % data.size();
        data[ind].insert_after(new Node(key));
        return ;
    }

    void remove(int key) {
        int ind = hash_func(key) % data.size();
        Node *p = &data[ind];
        while (p->next && p->next->key != key) p = p->next;
        p->remove_after();
        return ;
    }

    /** Returns true if this set contains the specified element */
    bool contains(int key) {
        int ind = hash_func(key) % data.size();
        Node *p = data[ind].next;
        while (p && p->key != key) p = p->next;
        return p != nullptr;
    }
};

/**
 * Your MyHashSet object will be instantiated and called as such:
 * MyHashSet* obj = new MyHashSet();
 * obj->add(key);
 * obj->remove(key);
 * bool param_3 = obj->contains(key);
 */
```

706. 设计哈希映射

```
class Node {
public:
    Node(int key = 0, int value = 0, Node *next = nullptr)
    : key(key), value(value), next(next) {}
    int key, value;
    Node *next;
    void insert_after(Node *node) {
        node->next = this->next;
        this->next = node;
        return ;
    }
    void remove_after() {
        if (this->next == nullptr) return ;
        Node *p = this->next;
        this->next = this->next->next;
        delete p;
        return ;
    }
};

class MyHashMap {
public:
```

kaikeba 开课吧

```
    /** Initialize your data structure here. */
    vector<Node> data;
    MyHashMap() : data(100) {}
    int hash_func(int key) { return key & 0x7fffffff; }
    /** value will always be non-negative. */
    void put(int key, int value) {
        int ind = hash_func(key) % data.size();
        Node *p = &data[ind];
        while (p->next && p->next->key != key) p = p->next;
        if (p->next) {
            p->next->value = value;
        } else {
            p->insert_after(new Node(key, value));
        }
        return ;
    }

    /** Returns the value to which the specified key is mapped, or -1 if this map contains no mapping for the key */
    int get(int key) {
        int ind = hash_func(key) % data.size();
        Node *p = data[ind].next;
        while (p && p->key != key) p = p->next;
        if (p == nullptr) return -1;
        return p->value;
    }

    /** Removes the mapping of the specified value key if this map contains a mapping for the key */
    void remove(int key) {
        int ind = hash_func(key) % data.size();
        Node *p = &data[ind];
        while (p->next && p->next->key != key) p = p->next;
        p->remove_after();
        return ;
    }
};

/**
 * Your MyHashMap object will be instantiated and called as such:
 * MyHashMap* obj = new MyHashMap();
 * obj->put(key,value);
 * int param_2 = obj->get(key);
 * obj->remove(key);
 */
```

kaikeba 开课吧

面试题 16.25. LRU 缓存

```
class Node {
public :
    Node(int key = 0, int value = 0, Node *prev = nullptr, Node *next = nullptr)
    : key(key), value(value), prev(prev), next(next) {}
    int key, value;
    Node *next, *prev;
    Node *remove_this() {
        if (this->prev) this->prev->next = this->next;
        if (this->next) this->next->prev = this->prev;
        this->next = this->prev = nullptr;
        return this;
    }
    void insert_prev(Node *node) {
        node->next = this;
        node->prev = this->prev;
        if (this->prev) this->prev->next = node;
        this->prev = node;
        return ;
    }
};

class HashList {
public :
    int capacity;
    Node head, tail;
    unordered_map<int, Node *> data;
    HashList(int capacity) : capacity(capacity) {
        head.next = &tail;
        tail.prev = &head;
    }
    void put(int key, int value) {
        if (data.find(key) != data.end()) {
            data[key]->value = value;
            data[key]->remove_this();
        } else {
            data[key] = new Node(key, value);
        }
        tail.insert_prev(data[key]);
        if (data.size() > capacity) {
            data.erase(data.find(head.next->key));
            delete head.next->remove_this();
        }
        return ;
    }
    int get(int key) {
        if (data.find(key) == data.end()) return -1;
        data[key]->remove_this();
        tail.insert_prev(data[key]);
        return data[key]->value;
    }
};


class LRUCache {
public:
    HashList h;
    LRUCache(int capacity) : h(capacity) {}

    int get(int key) {
        return h.get(key);
    }

    void put(int key, int value) {
        h.put(key, value);
        return ;
    }
};

/**
 * Your LRUCache object will be instantiated and called as such:
 * LRUCache* obj = new LRUCache(capacity);
 * int param_1 = obj->get(key);
 * obj->put(key,value);
 */
```

kaikeba 开课吧

535. TinyURL 的加密与解密

```
class Solution {
public:
    Solution() { srand(time(0)); }
    char ch(int x) {
        x %= 62;
        if (x < 26) return x + 'a';
        if (x < 52) return x - 26 + 'A';
        return x - 52 + '0';
    }
    string rand_string(int n) {
        string s = "";
        for (int i = 0; i < n; i++) {
            s += ch(rand());
        }
        return s;
    }
    unordered_map<string, string> h;
    // Encodes a URL to a shortened URL.
    string encode(string longUrl) {
        string s;
        do {
            s = rand_string(5);
        } while (h.find(s) != h.end());
        h[s] = longUrl;
        return s;
    }

    // Decodes a shortened URL to its original URL.
    string decode(string shortUrl) {
        return h[shortUrl];
    }
};

// Your Solution object will be instantiated and called as such:
// Solution solution;
// solution.decode(solution.encode(url));
```

187. 重复的DNA序列

```
class Solution {
public:
    vector<string> findRepeatedDnaSequences(string s) {
        unordered_map<string, int> h;
        for (int i = 0, I = s.size() - 9; i < I; i++) {
            h[s.substr(i, 10)] += 1;
        }
        vector<string> ret;
        for (auto x : h) {
            if (x.second == 1) continue;
            ret.push_back(x.first);
        }
        return ret;
    }
};
```

kaikeba 开课吧

318. 最大单词长度乘积

```
class Solution {
public:
    int maxProduct(vector<string>& words) {
        vector<int> mark(words.size());
        for (int i = 0; i < words.size(); i++) {
            for (auto c : words[i]) {
                mark[i] |= (1 << (c - 'a'));
            }
        }
        int ans = 0;
        for (int i = 0; i < words.size(); i++) {
            for (int j = i + 1; j < words.size(); j++) {
                if (mark[i] & mark[j]) continue;
                ans = max(ans, int(words[i].size() * words[j].size()));
            }
        }
        return ans;
    }
};
```

240. 搜索二维矩阵 II

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int i = 0, j = matrix[0].size() - 1;
        while (i < matrix.size() && j >= 0) {
            if (matrix[i][j] == target) return true;
            if (matrix[i][j] < target) i += 1;
            else j -= 1;
        }
        return false;
    }
};
```

979. 在二叉树中分配硬币

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int getResult(TreeNode *root, int &n, int &m) {
        n = m = 0;
        if (root == nullptr) return 0;
        n = 1, m = root->val;
        int ans = 0, n1, m1;
        ans += getResult(root->left, n1, m1);
        ans += abs(n1 - m1);
        n += n1, m += m1;
        ans += getResult(root->right, n1, m1);
        ans += abs(n1 - m1);
        n += n1, m += m1;
        return ans;
    }
    int distributeCoins(TreeNode* root) {
        int n, m;
        return getResult(root, n, m);
    }
};
```

430. 扁平化多级双向链表

```
/*
// Definition for a Node.
class Node {
public:
    int val;
    Node* prev;
    Node* next;
    Node* child;
};
*/

class Solution {
public:
    Node* flatten(Node* head) {
        Node *p = head, *q, *k;
        while (p) {
            if (p->child) {
                q = p->next;
                k = flatten(p->child);
                p->child = nullptr;
                p->next = k;
                k->prev = p;
                while (p->next) p = p->next;
                p->next = q;
                if (q) q->prev = p;
            }
            p = p->next;
        }
        return head;
    }
};
```

863. 二叉树中所有距离为 K 的结点

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void dfs(TreeNode *root, int c, int k, vector<int> &ret) {
        if (k < 0) return ;
        if (root == nullptr) return ;
        if (c == k) {
            ret.push_back(root->val);
            return ;
        }
        dfs(root->left,  c + 1, k, ret);
        dfs(root->right, c + 1, k, ret);
        return ;
    }
    TreeNode *getResult(TreeNode *root, TreeNode *target, int &k, vector<int> &ret) {
        if (root == nullptr) return nullptr;
        if (root == target) {
            dfs(root, 0, k, ret);
            return root;
        }
        if (getResult(root->left, target, k, ret)) {
            k -= 1;
            if (k == 0) ret.push_back(root->val);
            dfs(root->right, 0, k - 1, ret);
            return target;
        } else if (getResult(root->right, target, k, ret)) {
            k -= 1;
            if (k == 0) ret.push_back(root->val);
            dfs(root->left, 0, k - 1, ret);
            return target;
        }
        return nullptr;
    }
    vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {
        vector<int> ret;
        getResult(root, target, k, ret);
        return ret;
    };
};
```

kaikeba 开课吧