设计哈希集合

```
class Node {
public :
    Node(int data = 0, Node *next = nullptr) : __data(data),
next(next) {}
    int data() { return __data; }

    void insert_after(Node *node) {
        node->next = this->next;
        this->next = node;
        return ;
    }

    Node *find_after(int key) {
        Node *p = this->next;
        while (p && p->data() != key) p = p->next;
        return p;
    }
    void delete_after(int key) {
        Node *p = this, *q;
        while (p->next && p->next->data() != key) p = p->next;
        if (p->next == nullptr) return ;
        q = p->next;
        p->next = p->next->next;
        delete q;
        return ;
    }

private:
    Node *next;
    int __data;
```

开课吧

```
};

class MyHashSet {
public:
    /** Initialize your data structure here. */
    int size, capacity;
    vector<Node> htable;
    MyHashSet(int n = 100) : size(0), capacity(n), htable(capacity)
{}

    void add(int key) {
        int ind = key % capacity;
        if (htable[ind].find_after(key)) return ;
        htable[ind].insert_after(new Node(key));
        size += 1;
        return ;
    }

    void remove(int key) {
        int ind = key % capacity;
        htable[ind].delete_after(key);
        return ;
    }

    /** Returns true if this set contains the specified element */
    bool contains(int key) {
        int ind = key % capacity;
        return htable[ind].find_after(key) != nullptr;
    }
};
```

设计哈希映射

```
typedef pair<int, int> PII;
class Node {
public :
    Node(PII data = PII(0, 0), Node *next = nullptr) : __data(data),
next(next) {}
    PII &data() { return __data; }

    void insert_after(Node *node) {
        node->next = this->next;
        this->next = node;
        return ;
    }

    Node *find_after(int key) {
        Node *p = this->next;
        while (p && p->data().first != key) p = p->next;
        return p;
    }
    void delete_after(int key) {
        Node *p = this, *q;
        while (p->next && p->next->data().first != key) p = p->next;
        if (p->next == nullptr) return ;
        q = p->next;
        p->next = p->next->next;
        delete q;
        return ;
    }

private:
    Node *next;
    PII __data;
};
```

```
class MyHashMap {
public:
    /** Initialize your data structure here. */
    int size, capacity;
    vector<Node> htable;
    MyHashMap(int n = 100) : size(0), capacity(n), htable(capacity)
{}

    /** value will always be non-negative. */
    void put(int key, int value) {
        int ind = key % capacity;
        Node *p = htable[ind].find_after(key);
        if (p) {
            p->data().second = value;
            return ;
        }
        htable[ind].insert_after(new Node(PII(key, value)));
        size += 1;
        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 = key % capacity;
        Node *p = htable[ind].find_after(key);
        if (p == nullptr) return -1;
        return p->data().second;
    }

    /** Removes the mapping of the specified value key if this map
contains a mapping for the key */
    void remove(int key) {
        int ind = key % capacity;
        htable[ind].delete_after(key);
        return ;
    }
};
```

TinyURL 的加密与解密

```
class Solution {
public:
        Solution(){srand(time(0));};
        char rand_ch(){
                // a-z ,A-Z, 0-9
                int x=rand()%62;//0-25,26-51,52-61
                if(x<26)return 'a'+x;
                if(x<52)return 'A'+x-26;
                return '0'+x-52;
        }
        string rand_string(){
                string ret;
                //     生成 10 位随机字符串
                for(int i=0;i<10;i++){
                        ret+=rand_ch();
                }
                return ret;
        }
        //hash_function str1=>str2
        //str1 => str2  且  str2 => str1
        //str1 => str1' str1' =/=>str1

        unordered_map <string, string> h;

        // Encodes a URL to a shortened URL.
```

```
    string encode(string longUrl) {
        // 26+26+10 = 62^8
        // str1 => key1
        //  生成 1 个随机字符串，保证不冲突
        //  如果冲突，再生成 1 个  直到不冲突为止
        string tinyUrl;
        do {
            tinyUrl= rand_string();
            if (h.find(tinyUrl) != h.end()) continue;
            h[tinyUrl]=longUrl;
            break;
        }while(1);
        return tinyUrl;
    }

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

重复的 DNA 序列

```
class Solution {
public:
    // 定义一个哈希表
    // 生成所有子序列，检查出现次数
    vector<string> findRepeatedDnaSequences(string s) {
        vector<string> ans;
        unordered_map <string, int> cnt;
        int len=s.size()-9;
        for(int i=0;i<len;i++){
            cnt[s.substr(i,10)]++;
            if(cnt[s.substr(i,10)]==2){
```

开课吧

```
                ans.push_back(s.substr(i,10));
            }
        }
        //遍历 cnt[]>1
        //任何一个大于 1 的值，它都必须是从 1，2，3，4，。。。
        //for(auto iter=cnt.begin();iter!=cnt.end();iter++){
        //    if(iter->second == 1)continue;
        //    ans.push_back(iter->first);
        //}
        return ans;
    }
};
```

最大单词长度乘积

```
class Solution {
public:
    //bitmap
    //['a'-'z']
    //int m[26];
    //m[0]-'a'
    //m[25]-'z'
/*
    mask1 1110 0000 0000 0000 0000 0010 0000 0000
          abcd efgh ijkl mnop qrst uvwx yz
    mask2 1100 0000 0000 0000 0000 0000 0100 0000
          abcd efgh ijkl mnop qrst uvwx yz

    1&1 = 1
    1&0 = 0
    0&1 = 0
    0&0 = 0
*/
```

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

面试题 16.25. LRU 缓存

```
class Node{
    public:
    Node (int key=0,int val=0,Node *next=nullptr,Node* pre=nullptr)
    :key(key),val(val),next(next),pre(pre) {}

    int key,val;
    Node *next,*pre;

    Node* remove_this(){
        if(next) next->pre = pre;
        if(pre) pre->next = next;
        next = nullptr;
        pre = nullptr;
        return this;
    }
    void delete_next(){
        if(next == nullptr)return;
        delete next->remove_this();
    }
    void insert_pre(Node *node){
        node->next = this;
        node->pre = this->pre;
        if(this->pre) this->pre->next = node;
        this->pre = node;
    }

};

class HashList {
    public:
        HashList() {
            head.next = &tail;
            tail.pre= &head;
        }
        void insert(int key,int val){
            if(h.find(key) != h.end() ){
                h[key]->val = val;
                get(key);
                return; // 注意不要漏掉了这个返回值
            }
            tail.insert_pre(h[key] = new Node(key,val));
        }
        void pop_front(){
            h.erase(h.find(head.next->key));
            head.delete_next();
        }

        int get(int key) {
            if(h.find(key) == h.end()) return -1;
            Node *p=h[key];
            p->remove_this();
            tail.insert_pre(p);
            return p->val;
        }
        int size(){
            return h.size();
        }
    private:
        int cnt;
        Node head,tail;
        unordered_map <int,Node*> h;
};

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

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

    void put(int key, int value) {
        h.insert(key,value);
        if(h.size() > capacity) h.pop_front();
    }
};
```

开课吧

在二叉树中分配硬币

```
class Solution {
public:
    int answer;
    // 返回，汇聚的金币数量
    int moves(TreeNode* root){
        if(root==nullptr)return 0;
        int left_moves=moves(root->left);
        int right_moves=moves(root->right);
        answer += abs(left_moves) + abs(right_moves);
        return left_moves+right_moves+root->val-1;
    }
    int distributeCoins(TreeNode* root) {
        answer=0;
        moves(root);
        return answer;
    }
};
```

二叉树中所有距离为 K 的结点

```
class Solution {
public:
    void dfs(TreeNode* root, int k, vector<int> &ans){
        if (k<0 ||root==nullptr){return;}
        printf("dfs %d %d\n",root->val,k);
        if (k==0){ans.push_back(root->val);return;}
        dfs(root->left,k-1,ans);
        dfs(root->right,k-1,ans);
    }
    TreeNode* getResult(TreeNode* root, TreeNode* target,
     int &k, vector<int> &ans){ // 注意 k 需要传递引用
```

```
        if(root == nullptr) return nullptr;
        printf("get %d %d\n",root->val,k);
        if(root == target){
            dfs(root,k,ans);
            k--;
            return root;
        }else if(getResult(root->left, target, k ,ans)){
            if(k==0) ans.push_back(root->val);
            else dfs(root->right ,k-1,ans);
            k--;
            return target;
        }else if(getResult(root->right, target, k ,ans)){
            if(k==0) ans.push_back(root->val);
            else dfs(root->left, k-1, ans);
            k--;
            return target;
        }
        return nullptr;
    }
    vector<int> distanceK(TreeNode* root, TreeNode* target, int k) {
        vector<int> ans;
        getResult(root, target, k, ans);
        return ans;
    }
};
```

开课吧