计算机视觉总结
SUMMARY OF COMPUTER VISION

(第1版)

LVSHUAILIN

OPEN SOURCE
BEIJING

VERSION 1

一. 数据结构与算法-LeetCode Hot 100

二. PYTHON: 1) NUMPY; 2) PANDAS; 3) PYTHON多进程; 4) PYTHON分布式; 5) PYTHON界面;

三. 深度学习: TensorFlow 2.0; PYTORCH;

四. 图像配准

五. 强化学习

六. OTHERS: 1) Model INFERENCE by EXE; 2) GIT; 3) DOCKER

LVSHUAILIN

2020年2月

目　录

第1章　LeetCode Hot 100

Goals to Achieve

1. unordered_map.

§ 1.1 两数之和unordered_map

HOT100 1.1　问题描述

给定一个整数数组**nums** 和一个目标值**target**, 请你在该数组中找出和为目标值的那两个整数, 并返回他们的数组下标. 你可以假设每种输入只会对应一个答案. 但是, 你不能重复利用这个数组中同样的元素.

示例: 给定nums = [2, 7, 11, 15], target = 9; 因为nums[0] + nums[1] = 2 + 7 = 9;

所以返回[0, 1]

https://leetcode-cn.com/problems/two-sum

1.1.1 解题思路

这里用c++的unordered_map来解决, unordered_map内部是一个关联容器, 采用hash 表结构, 有快速检索的功能.

哈希表是通过key关键字直接访问对应value值的数据结构. 特点是键和值一一对应, 查找时间复杂度**O(1)**.

Example_1: unordered_map插入, 迭代遍历.

unordered_map example_1 code

```
#include <iostream>
#include <unordered_map>
#include <string>
using namespace std;
int main()
```

```
{
    unordered_map<string, double> umap;
    umap["PI"] = 3.14;
    umap.insert(make_pair("a", 2.1));

    // find in umap
    string key = "PI";
    if (umap.find(key) == umap.end())
        cout << "cannot find PI" << endl;
    else
        cout << "find " << umap.find(key)->first << " = " << umap.find(key)->second << endl;

    // iterator of umap
    cout << "entire unorded_map is:"<<endl;
    unordered_map<string, double>::iterator itr;
    for (itr = umap.begin(); itr != umap.end(); ++itr)
        cout << "(" << itr->first << "," << itr->second << ")" << endl;
    system("pause");
    return 0;
}
```

output:

```
find PI = 3
all elements are:
(PI,3.14)
(a,2.1)
```

Example_2: 利用unordered_map输出一段文字中重复单词的个数

unordered_map example_2 code

```
#include <iostream>
#include <unordered_map>
#include <string>
#include <sstream>

using namespace std;

void printWordFreq(const string& str)
{
    unordered_map<string, int> wordFreq;
    string word;
    stringstream ss(str);
    while (ss >> word)
        wordFreq[word]++;

    cout << "all elements are:" << endl;
    for (auto u : wordFreq)
        cout << "(" << u.first << "," << u.second << ")" << endl;
}

int main()
{
    string str = "studies very very hard";
    printWordFreq(str);
    return 0;
}
```

output:

```
all elements are:
(studies, 1)
(very, 2)
(hard, 1)
```

1.1.2 解题代码

```
#include <iostream>
#include <unordered_map>
#include <vector>
using namespace std;

vector<int> twoSum(vector<int>& nums, int target)
{
    unordered_map<int, int> map;
    vector<int> result={};
    int n = (int)nums.size();
    for(int i = 0; i < n; ++i) {
        auto p = map.find(target-nums[i]);
        if(p != map.end()) {
        result.push_back(p->second);
        result.push_back(i);
        }
    map[nums[i]] = i;
    }
    return result;
}

int main()
{
    vector< int > nums = {2,7,11,15};
    vector<int> result;
    result = twoSum(nums,9);
    cout<<"["<<result[0] << "," <<result[1]<<"]"<<endl;
    return 0;
}
```

§ 1.2 两数相加linked list

HOT100 1.2 问题描述

给出两个非空的链表用来表示两个非负的整数. 其中, 它们各自的位数是按照逆序的方式存储的, 并且它们的每个节点只能存储一位数字. 如果, 我们将这两个数相加起来, 则会返回一个新的链表来表示它们的和.

您可以假设除了数字0之外, 这两个数都不会以0开头.

示例: 输入(2 − > 4 − > 3) + (5 − > 6 − > 4), 输出: 7 − > 0 − > 8, 原因: $342 + 465 = 807$

https://leetcode-cn.com/problems/add-two-numbers

1.2.1 解题思路

这里用c++ 链表来解决

Example_1: 创建链表并初始化

linked list example_1 code

```
#include <iostream>

using namespace std;

class Node{
```

```
public:
    int data;
    Node* next;
};

int main()
{
    Node* head = nullptr;
    Node* second = nullptr;
    Node* third = nullptr;

    head = new Node();
    head->data = 1;

    second = new Node();
    second->data = 2;

    third = new Node();
    third->data = 3;

    cout << head->data << " " << second->data << " " << third->data << endl;

    delete head;
    delete second;
    delete third;
    return 0;
}
```

output:

```
1 2 3
```

Example_2: 打印链表中的所有元素

linked list example_2 code

```
#include <iostream>

using namespace std;

class Node{
public:
    int data;
    Node* next;
};

void PrintLinkedList(Node* head)
{
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << "  ";
        temp = temp->next;
    }
    cout << endl;
}

int main()
{
    Node* head = nullptr;
    Node* second = nullptr;
    Node* third = nullptr;

    head = new Node();
    second = new Node();
    third = new Node();

    head->data = 1;
    head->next = second;

    second->data = 2;
    second->next = third;

    third->data = 3;
    third->next = nullptr;


    PrintLinkedList(head);

    delete head;
    delete second;
    delete third;
    return 0;
}
```

output:
1 2 3

Example_3: 链表插入节点

linked list example_3 code

```
#include <iostream>

using namespace std;

class Node{
public:
    int data;
    Node* next;
};

//在链表前面插入节点
void push(Node** head_ref, int newData)
{
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = (*head_ref);
    (*head_ref) = newNode;
}

//在节点后面插入节点
void insertAfter(Node** prev_node, int newData)
{
    if ((*prev_node) == nullptr) {
        cout << "the previous node cannot be nullptr" << endl;
        return;
    }

    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = (*prev_node)->next;
    (*prev_node)->next = newNode;
}

//在尾节点后插入节点
```

```
void append(Node** head_ref, int newData)
{
    Node* newNode = new Node();
    newNode->data = newData;
    newNode->next = nullptr;
    if ((*head_ref) == nullptr) {
        (*head_ref) = newNode;
        return;
    }

    Node* move = (*head_ref);
    while (move->next != nullptr) {
        move = move->next;
    }
    move->next = newNode;
}

// 打印链表
void PrintLinkedList(Node* head)
{
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << "  ";
        temp = temp->next;
    }
    cout << endl;
}

void destroyLinkedList(Node** head_ref) {
    Node* move = (*head_ref);
    Node* next = nullptr;
    while (move != nullptr) {
        next = move->next;
        delete move;
        move = next;
    }
    (*head_ref) = nullptr;
}

int main()
```

```
{
    Node* head = nullptr;

    append(&head, 6);

    push(&head, 7);

    push(&head, 1);

    append(&head, 4);

    insertAfter(&(head->next), 8);

    cout << "linked list is: ";
    PrintLinkedList(head);
    destroyLinkedList(&head);
    return 0;
}
```

output:
linked list is: 1 7 8 6 4

1.2.2 解题代码

```
#include <iostream>
using namespace std;

struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
};

ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {
    int len1 = 1;//记录的长度l1
    int len2 = 1;//记录的长度l2
    ListNode* p = l1;
    ListNode* q = l2;
    while (p->next != NULL)//获取的长度l1
    {
        len1++;
        p = p−>next;
    }
    while (q−>next != NULL)//获取的长度l2
    {
        len2++;
        q = q−>next;
    }
    if (len1 > len2)//较长，在末尾补零l1l2
    {
        for (int i = 1; i <= len1 − len2; i++)
        {
            q−>next = new ListNode(0);
            q = q−>next;
        }
    }
    else//较长，在末尾补零l2l1
    {
        for (int i = 1; i <= len2 − len1; i++)
        {
            p−>next = new ListNode(0);
            p = p−>next;
        }
    }
    p = l1;
    q = l2;
    bool count = false;//记录进位
    ListNode∗ l3 = new ListNode(−1);//存放结果的链表
    ListNode∗ w = l3;//的移动指针l3
    int i = 0;//记录相加结果
    while (p != NULL && q != NULL)
    {
        i = count + p−>val + q−>val;
        w−>next = new ListNode(i % 10);
        count = i >= 10 ? true : false;
        w = w−>next;
        p = p−>next;
        q = q−>next;
    }
    if (count)//若最后还有进位
    {
        w->next = new ListNode(1);
        w = w->next;
    }
    return l3->next;
}

void printLinkedList(ListNode* head)
{
    ListNode* move = head;
    while (move != nullptr) {
        cout << move->val << "  ";
        move = move->next;
    }
}

int main()
{
#if 1
    ListNode* l1 = new ListNode(2);
    ListNode* l1_1 = new ListNode(4);
    ListNode* l1_2 = new ListNode(3);

    l1->next = l1_1;
    l1_1->next = l1_2;


    ListNode* l2 = new ListNode(5);
    ListNode* l2_1 = new ListNode(6);
    ListNode* l2_2 = new ListNode(4);
    l2->next = l2_1;
    l2_1->next = l2_2;
#endif

#if 0
    ListNode* l1 = new ListNode(5);
    ListNode* l2 = new ListNode(5);
#endif

```

```
    ListNode* result = addTwoNumbers(l1, l2);
    printLinkedList(result);
    return 0;
}
```

```
output:
7 0 8
```

§ 1.3 无重复字符的最长子串(double pointer algorithm, set)

HOT100 1.3　问题描述

给定一个字符串, 请你找出其中不含有重复字符的最长子串的长度.

示例1:

输入: “abcabcb”

输出: 3

解释: 因为无重复字符的最长子串是“abc”, 所以其长度为3.

示例2:

输入: “bbbbb”

输出: 1

解释: 因为无重复字符的最长子串是“b”, 所以其长度为1.

示例3:

输入: “pwwkew”

输出: 3

解释: 因为无重复字符的最长子串是“wke”, 所以其长度为3. 请注意, 你的答案必须是子串的长度, “pwke”是一个子序列, 不是子串.

https://leetcode-cn.com/problems/longest-substring-without-repeating-characters

1.3.1 解题思路

https://cloud.tencent.com/developer/article/1377650

这道题主要用到思路是: 滑动窗口

什么是滑动窗口?

其实就是一个队列, 比如例题中的abcabcbb, 进入这个队列(窗口)为abc满足题目要求, 当再进入a, 队列变成了abca, 这时候不满足要求. 所以, 我们要移动这个队列!

如何移动?

我们只要把队列的左边的元素移出就行了, 直到满足题目要求!

一直维持这样的队列, 找出队列出现最长的长度时候, 求出解!

时间复杂度: **O(n)**

1.3.2 解题代码

```
#include <iostream>
#include <string>
#include <unordered_set>
#include <algorithm> // max, min

using namespace std;

int lengthOfLongestSubstring(string s) {
    if (s.size() == 0) return 0;
    unordered_set<char> lookup;
    int maxStr = 0;
    int left = 0;
    for (int i = 0; i < s.size(); i++) {
        while (lookup.find(s[i]) != lookup.end()) {
            lookup.erase(s[left]);
            left++;
        }
        maxStr = max(maxStr, i - left + 1);
        lookup.insert(s[i]);
    }
    return maxStr;
}

int main()
{
    string str = "abcabcb";
    cout << lengthOfLongestSubstring(str) << endl;
    return 0;
}
```

```
output:
3
```

§ 1.4 寻找两个有序数组的中位数(二分查找|折半查找算法)

HOT100 1.4　问题描述

给定两个大小为m和n的有序数组nums1和nums2.

请你找出这两个有序数组的中位数, 并且要求算法的时间复杂度为**O(log(m + n))**.

你可以假设nums1和nums2不会同时为空.

示例1:

nums1 = [1, 3]

nums2 = [2]

则中位数是2.0

示例2:

nums1 = [1, 2]

nums2 = [3, 4]

则中位数是(2 + 3)/2 = 2.5

https://leetcode-cn.com/problems/median-of-two-sorted-arrays

1.4.1 解题思路

用二分查找算法, 也叫做折半查找算法.

Example_1: 二分查找

二分查找算法*example_1 code*

```
// 二分查找折半查找|
int search(int arr[], int key, int left, int right)
{
    while (left <= right)
    {
        int mid = left + (right - left) / 2;
        if (key < arr[mid])
            right = mid - 1;
        else if (key > arr[mid])
            left = mid + 1;
        else
            return mid;
    }
    return −1;
}

int main()
{
    int arr[] = { 0,2 ,3,4};
    int value = 3;

    // left Index of the array
    int left = 0;

    // right Index of the array
    int right = sizeof(arr) / sizeof(arr[0]) − 1;

    cout << "left: " << left << ", right: " << right << endl;

    int ret = search(arr, value, left, right);
    if (ret == −1)
        printf("cannot find the value");
    else
        printf("found the value, the index is: %d\n", ret);
    system("pause");
    return 0;
}
```

output:

```
left: 0, right: 3
found the value, the index is: 2
```

第2章　深度学习

Goals to Achieve

1. pytorch basics
2. pytorch projects

§ 2.1 Pytorch

打印模型结构

pip install torchsummary

summary(model, (3, 32, 32))

print(model)