

华中科技大学

2019

算法设计与分析实验报告

专 业： 计算机科学与技术

班 级： CS1703

学 号： U201714670

姓 名： 范唯

完成日期： 2019 年 12 月 12 日



华中科技大学课程设计报告

目 录

1	完成情况汇报.....	1
2	POJ 解题报告.....	2
2.1	通关截图.....	2
2.2	通关题目详解(节选).....	2
3	LEETCODE 解题报告.....	11
3.1	相关定义和概念.....	11
3.2	基本理论和方法.....	12
4	心得体会.....	13
4.1	关于算法设计.....	13
4.2	关于编程技术.....	14

1 完成情况汇报

因为之前主要用的语言是Python3.并且CSP选的考试语言也是python,所以一直在刷LeetCode.以下是LeetCode的完成进度.

#	题名	题解	通过率	难度	出现频率
✓ 1	Two Sum	660	47.1%	简单	
✓ 2	Add Two Numbers	610	36.2%	中等	
✓ 3	Longest Substring Without Repeating Characters	532	32.2%	中等	
✓ 4	Median of Two Sorted Arrays	291	36.4%	困难	
✓ 5	Longest Palindromic Substring	264	27.8%	中等	
✓ 6	ZigZag Conversion	203	45.8%	中等	
✓ 7	Reverse Integer	439	33.3%	简单	
✓ 8	String to Integer (atoi)	297	18.6%	中等	
✓ 9	Palindrome Number	362	56.7%	简单	
10	Regular Expression Matching	121	25.4%	困难	
✓ 11	Container With Most Water	205	59.8%	中等	
✓ 12	Integer to Roman	164	61.9%	中等	
✓ 13	Roman to Integer	405	59.8%	简单	
✓ 14	Longest Common Prefix	289	35.5%	简单	
✓ 15	3Sum	214	24.7%	中等	
✓ 16	3Sum Closest	125	42.2%	中等	
✓ 17	Letter Combinations of a Phone Number	225	51.6%	中等	

以下是POJ平台的完成进度.

Problem Status List

Problem ID: User ID: alex8802732 Result: Accepted Language: All Go

Run ID	User	Problem	Result	Memory	Time	Language	Code Length	Submit Time
21149704	alex8802732	1050	Accepted	212K	32MS	C++	1205B	2019-12-13 19:30:24
21149367	alex8802732	3579	Accepted	316K	375MS	C++	1035B	2019-12-13 18:36:14
21149330	alex8802732	3269	Accepted	740K	360MS	C++	1954B	2019-12-13 18:26:56
21148983	alex8802732	1723	Accepted	264K	32MS	C++	445B	2019-12-13 17:15:52
21112634	alex8802732	3233	Accepted	932K	1500MS	C++	1623B	2019-12-02 11:14:31
21112625	alex8802732	3233	Accepted	932K	1485MS	C++	1831B	2019-12-02 11:11:40
21112592	alex8802732	3233	Accepted	932K	1454MS	C++	1847B	2019-12-02 10:55:23
21109700	alex8802732	3714	Accepted	4960K	2672MS	C++	1141B	2019-11-30 23:19:47
21109368	alex8802732	2503	Accepted	10504K	1047MS	C++	601B	2019-11-30 21:44:47
21109333	alex8802732	2503	Accepted	10504K	1047MS	C++	659B	2019-11-30 21:38:16
21109293	alex8802732	2366	Accepted	476K	516MS	C++	691B	2019-11-30 21:27:12
21108962	alex8802732	3295	Accepted	152K	16MS	C++	1723B	2019-11-30 19:29:30
21106469	alex8802732	1753	Accepted	164K	47MS	C++	1601B	2019-11-29 21:25:04
21105906	alex8802732	1753	Accepted	164K	16MS	C++	1540B	2019-11-29 20:12:42
20990273	alex8802732	1005	Accepted	140K	0MS	C++	516B	2019-10-25 21:27:14
20989647	alex8802732	1000	Accepted	124K	0MS	C	107B	2019-10-25 19:27:39

2. POJ

2.1 通关截图

通关AC题号如下图↓

Problem Status List

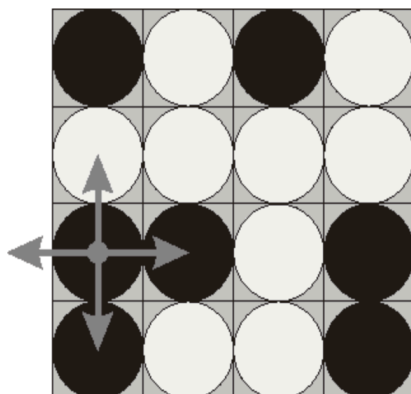
Problem ID: User ID: alex8802732 Result: Accepted Language: All Go

Run ID	User	Problem	Result	Memory	Time	Language	Code Length	Submit Time
21149704	alex8802732	1050	Accepted	212K	32MS	C++	1205B	2019-12-13 19:30:24
21149367	alex8802732	3579	Accepted	316K	375MS	C++	1035B	2019-12-13 18:36:14
21149330	alex8802732	3269	Accepted	740K	360MS	C++	1954B	2019-12-13 18:26:56
21148983	alex8802732	1723	Accepted	264K	32MS	C++	445B	2019-12-13 17:15:52
21112634	alex8802732	3233	Accepted	932K	1500MS	C++	1623B	2019-12-02 11:14:31
21112625	alex8802732	3233	Accepted	932K	1485MS	C++	1831B	2019-12-02 11:11:40
21112592	alex8802732	3233	Accepted	932K	1454MS	C++	1847B	2019-12-02 10:55:23
21109700	alex8802732	3714	Accepted	4960K	2672MS	C++	1141B	2019-11-30 23:19:47
21109368	alex8802732	2503	Accepted	10504K	1047MS	C++	601B	2019-11-30 21:44:47
21109333	alex8802732	2503	Accepted	10504K	1047MS	C++	659B	2019-11-30 21:38:16
21109293	alex8802732	2366	Accepted	476K	516MS	C++	691B	2019-11-30 21:27:12
21108962	alex8802732	3295	Accepted	152K	16MS	C++	1723B	2019-11-30 19:29:30
21106469	alex8802732	1753	Accepted	164K	47MS	C++	1601B	2019-11-29 21:25:04
21105906	alex8802732	1753	Accepted	164K	16MS	C++	1540B	2019-11-29 20:12:42
20990273	alex8802732	1005	Accepted	140K	0MS	C++	516B	2019-10-25 21:27:14
20989647	alex8802732	1000	Accepted	124K	0MS	C	107B	2019-10-25 19:27:39

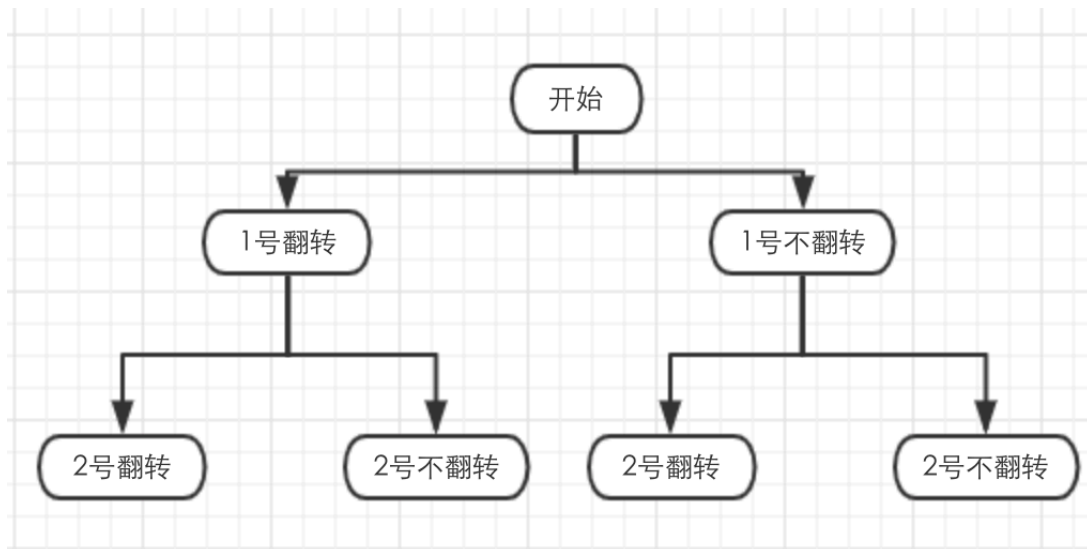
2.2 通关题目详解(节选)

2.2.1 POJ 1753 Flip Game

题目大意: 有4×4的正方形, 每个格子要么是黑色, 要么是白色, 当把一个格子的颜色改变(黑->白或者白->黑)时, 其周围上下左右(如果存在的话)的格子的颜色也被反转, 问至少反转几个格子可以使4×4的正方形变为纯白或者纯黑?



解题思路: 本题主要采用递归、二叉树方法来做.也是一种暴力枚举的方法,但是可以将黑白状态用2进制表示,这样0000 0000 0000 0000就代表都为白色.可以将运算转换为位运算.而二叉树的形式大致为下图.(以此类推此第16层)



从最后一层开始搜索,从下往上.以上述二叉树举例:

如果2号翻转后,棋盘为纯色,则结束搜索.如果不为纯白,则将2变为初态,查看棋盘是否为纯色,如果仍然不是,则返回至上一层,将1号翻转.再往下层搜索.

如果每一种情况都试过了, 还是没变成纯色, 则输出“Impossible”。

AC代码如下:

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

bool samecolor();
void flip(int i);
bool b_tree(int i);
int getTimes();
void search();

int tf[16] ;
int sq[4][4];
int times = -1;

int main(){
    char temp;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            cin >> temp;
            if (temp == 'b')
                sq[i][j] = 1;
            else
                sq[i][j] = 0;
        }
    }
}
```

```

        search();
        if (times == -1)
        {
            cout << "Impossible" << endl;
        }
        else
        {
            cout << times << endl;
        }
    }

void search()
{
    tf[0] = 0;
    flip(0);
    b_tree(1);

    tf[0] = 1;
    flip(0);
    b_tree(1);
}

int getTimes(){
    int curtime = 0;
    for(int i=0;i<16;i++){
        if(tf[i] == 0){
            curtime +=1;
        }
    }
    return curtime;
}

bool b_tree(int i){//i为搜索层数
    if (i == 16)
    {
        if (samecolor())
        {
            if (times == -1)
                times = getTimes();
            else
            {
                int tempTime = getTimes();
                times = tempTime < times ? tempTime:times;
            }
            return true;
        }
        else
            return true;
    }

```

```

}

tf[i] = 0;
flip(i);
b_tree(i+1);
tf[i] = 1;
flip(i);
b_tree(i+1);
return 1;
}

void flip(int i){
    int n = i/4;
    int m = i%4;
    sq[n][m] = (sq[n][m] == 1)? 0:1;
    if (m != 0){
        sq[n][m-1] = (sq[n][m-1] == 1)? 0:1;
    }
    if (n != 0){
        sq[n-1][m] = (sq[n-1][m] == 1)? 0:1;
    }
    if (m != 3){
        sq[n][m+1] = (sq[n][m+1] == 1)? 0:1;
    }
    if (n != 3){
        sq[n+1][m] = (sq[n+1][m] == 1)? 0:1;
    }
}

bool samecolor(){
    for(int i = 0; i <= 3; i++){
        for(int j = 0; j <= 3; j++){
            if(sq[i][j] != sq[0][0]) return false;
        }
    }
    return true;
}

```

2.2.2 POJ 3295 Tautology

题目大意: 输入长度 ≤ 100 的由K,A,N,C,E,p,q,r,s,t组成的逻辑表达式，其中K,A,N,C,E分别代表与，或，非，条件，等价，小写字母为命题变量。若该式为永真式，则输出tautology，否则输出not.输入数据有多组，遇0停止输入。

解题思路: 考虑表达式最多只有5个变量，真值表最多只有 $2^5=32$ 种排列，所以选择枚举所有排列判断表达式是否恒真。所以可以通过五位二进制数来表示每一种情况.从00000~11111.

而类比前缀表达式，用一个栈维护就行。这里用了系统递归的栈。taut函数不断压栈,最后将长表达式分解为最短子表达式。

AC代码如下:

```
#include <iostream>
#include <string>
using namespace std;
string wwf;
int current;
int flag;
int taut(int i) {
    char ch = wwf[current++];
    if(ch=='p' || ch=='q' || ch=='r' || ch=='s' || ch=='t'){
        return (i>>int (ch-'p'))&1;
    }
    else if(ch=='K') return taut(i) & taut(i);
    else if(ch=='A') return taut(i) | taut(i);
    else if(ch=='N') return !taut(i);
    else if(ch=='C') return (!taut(i)) | taut(i);
    else if(ch=='E') return taut(i) == taut(i);
    else return 0;
}

int main() {
    while(cin >> wwf && wwf[0]!='0') {
        flag = 1;
        for(int i=0; i<32; i++) {
            current = 0;
            if(!taut(i)) {
                flag = 0;
                break;
            }
        }
        if(flag) cout << "tautology" << endl;
        else cout << "not" << endl;
    }

    return 0;
}
```

2.2.3 POJ 3233 Matrix Power Series

题目大意:

Description

Given a $n \times n$ matrix A and a positive integer k , find the sum $S = A + A^2 + A^3 + \dots + A^k$.

解题思路:借鉴网上的巧妙算法,不难发现可以对一个矩阵进行迭代就可以算出最后的结果.

$$\begin{vmatrix} A & 0 \\ 1 & 1 \end{vmatrix}^2 = \begin{vmatrix} A^2 & 0 \\ A+1 & 1 \end{vmatrix}$$

$$\begin{vmatrix} A & 0 \\ 1 & 1 \end{vmatrix}^3 = \begin{vmatrix} A^3 & 0 \\ A^2+A+1 & 1 \end{vmatrix}$$

$$\begin{vmatrix} A & 0 \\ 1 & 1 \end{vmatrix}^4 = \begin{vmatrix} A^4 & 0 \\ A^3+A^2+A+1 & 1 \end{vmatrix}$$

AC代码如下:

```
#include <iostream>
#include <algorithm>

using namespace std;
const int max_number = 110;
const int mod = 2;
typedef long long ll;

struct matrix {
    ll a[max_number][max_number];
};

matrix que, ans;
```

```

long long n, k, m;

matrix multip( matrix x, matrix y ) {
    matrix tmp;
    for( ll i = 0; i < 2*n; i ++ ) {
        for( ll j = 0; j < 2*n; j ++ ) {
            tmp.a[i][j] = 0;
            for( ll k = 0; k < 2*n; k ++ ) {
                tmp.a[i][j] = ( tmp.a[i][j] + x.a[i][k] * y.a[k][j] ) % m;
            }
        }
    }
    return tmp;
}

void func( ll x ) {
    while(x) {
        if( x&1 ) {
            ans = multip( ans, que );
        }
        que = multip( que, que );
        x /= 2;
    }
}

int main() {
    while( cin >> n >> k >> m ) {
        memset( ans.a, 0, sizeof(ans.a) );
        memset( que.a, 0, sizeof(que.a) );
        for( ll i = 0; i < n; i ++ ) {
            for( ll j = 0; j < n; j ++ ) {
                cin >> que.a[i][j];
            }
        }
        for( ll i = n; i < 2*n; i ++ ) {
            que.a[i][i-n] = que.a[i][i] = 1;
        }
        for( ll i = 0; i < 2*n; i ++ ) {
            ans.a[i][i] = 1;
        }
        func(k+1);
        for( ll i = n; i < 2*n; i ++ ) {
            for( ll j = 0; j < n; j ++ ) {
                if( i == j+n ) {
                    ans.a[i][j] --;
                }
                if( j != n-1 ) {
                    cout << ( ans.a[i][j] + m ) % m << " ";
                } else {

```

```

        cout << ( ans.a[i][j] + m ) % m << endl;
    }
}
}
return 0;
}

```

2.2.4 POJ 1050 To the Max

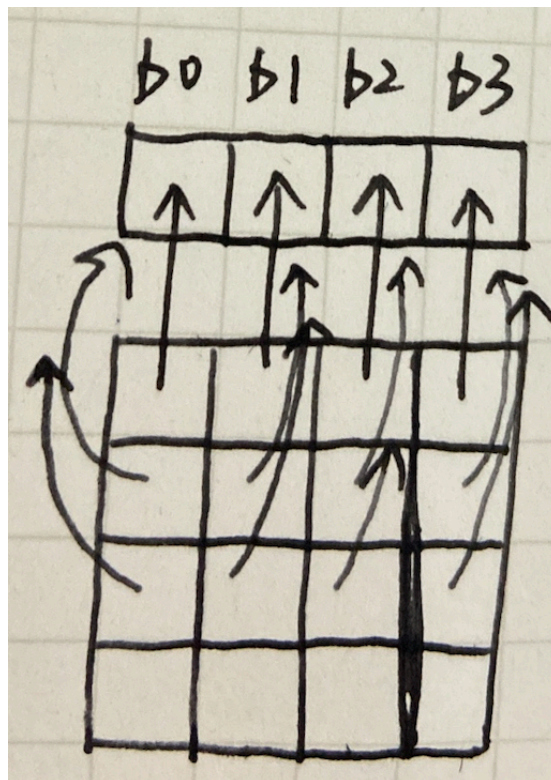
题目大意: 给出 $n \times n$ 的矩阵, 求和最大的子矩阵。

解题思路: 可以利用矩阵压缩把二维的问题转化为一维的最大子段和问题。因为是矩阵和, 所以我们可以把这个矩形的高压缩成1, 用加法就行了。

而一维的最大子段和问题在动态规划的课上已经讲过, 这里不再赘述。

如下图所示(手画的比较丑). 用这种方法将矩阵从二维压缩到一维, b 数组在更新起始行时重置为0。

建议配合代码一起理解。



AC代码如下

```

#include <iostream>
#define SIZE 101

using namespace std;

int a[SIZE][SIZE];

```

```

int MaxSubArray(int n, int* a)
{
    int max = 0;
    int b = 0;
    for (int i = 0; i < n; i++)
    {
        if (b > 0) b += a[i];
        else b = a[i];
        if (b > max) max = b;
    }
    return max;
}

int MaxSubMatrix(int m, int n)
{
    int i, j, k, sum;
    int max = 0;
    int b[SIZE];
    for (i = 0; i < m; i++)
    {
        for (k = 0; k < n; k++)
        {
            b[k] = 0;
        }
        for (j = i; j < m; j++)
        {
            for (k = 0; k < n; k++)
            {
                b[k] += a[j][k];
            }
            sum = MaxSubArray(k, b);
            if (sum > max)
            {
                max = sum;
            }
        }
    }
    return max;
}

int main()
{
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cin >> a[i][j];
        }
    }
}

```

```

    }
}
cout << MaxSubMatrix(n, n) << endl;
return 0;
}

```

3. LeetCode

因为备战CSP,所以刷POJ的同时也做了一会儿LeetCode.所以顺带也写两题心得体会.

3.1 通关截图

通关AC题号如下图↓

#	题名	题解	通过率	难度	出现频率 
✓ 1	Two Sum	660	47.1%	简单	
✓ 2	Add Two Numbers	610	36.2%	中等	
✓ 3	Longest Substring Without Repeating Characters	532	32.2%	中等	
✓ 4	Median of Two Sorted Arrays	291	36.4%	困难	
✓ 5	Longest Palindromic Substring	264	27.8%	中等	
✓ 6	ZigZag Conversion	203	45.8%	中等	
✓ 7	Reverse Integer	439	33.3%	简单	
✓ 8	String to Integer (atoi)	297	18.6%	中等	
✓ 9	Palindrome Number	362	56.7%	简单	
10	Regular Expression Matching	121	25.4%	困难	
✓ 11	Container With Most Water	205	59.8%	中等	
✓ 12	Integer to Roman	164	61.9%	中等	
✓ 13	Roman to Integer	405	59.8%	简单	
✓ 14	Longest Common Prefix	289	35.5%	简单	
✓ 15	3Sum	214	24.7%	中等	
✓ 16	3Sum Closest	125	42.2%	中等	
✓ 17	Letter Combinations of a Phone Number	225	51.6%	中等	

3.2 通关题目详解(节选)

3.2.1 LeetCode 4 Median of Two Sorted Arrays

题目大意: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。

解题思路:因为在插入一个新的数到一个数组时,我们经常采用二分法,这样可以快速找到该数的定位.但是对于两个有序数组,如何进行二分法呢?

因为需要找到中位数,所以按照定义,这个数两边的数字数量是相同的,从nums1中切割一刀,那么nums2中的另外一刀也是唯一确定好的.所以我们只需要找到切割第一个数组的位置,即找到了中位数所在的地方.那么二分法既可以从一个数组拓展到两个数组了。

AC代码如下

```
class Solution:
    def findMedianSortedArrays(self, nums1: [int], nums2: [int]) -> float:
        if len(nums1) > len(nums2): nums1, nums2 = nums2, nums1
        l, r = 0, 2 * len(nums1)
        while l <= r:
            m1 = (l + r) // 2
            m2 = len(nums1) + len(nums2) - m1
            left1 = nums1[(m1 - 1) // 2] if m1 != 0 else float("-inf")
            right1 = nums1[m1 // 2] if m1 != 2 * len(nums1) else float("+inf")
            left2 = nums2[(m2 - 1) // 2] if m2 != 0 else float("-inf")
            right2 = nums2[m2 // 2] if m2 != 2 * len(nums2) else float("+inf")
            if left1 > right2: r = m1 - 1
            elif left2 > right1: l = m1 + 1
            else: return (max(left1, left2) + min(right1, right2)) / 2
```

3.2.2 LeetCode 5 Longest Palindromic Substring

题目大意:找出一个数组的最长回文子串。

Example 1:

Input: "babad"
Output: "bab"
Note: "aba" is also a valid answer.

Example 2:

Input: "cbbd"
Output: "bb"

解题思路: 虽然用暴力法:从每一个点向外扩张.看起来很简单,但是时间复杂度极高.

Manacher (俗称马拉车)算法可以降低时间成本.因为篇幅问题.就不做赘述.

AC代码如下:

```
class Solution:
    def longestPalindrome(self, s: str):
        s_ext = '#' + '#' + s + '#'
        s_ext_len = len(s_ext)
        RL = [0] * s_ext_len
        mid = 0
        maxright = 0
        maxlen = 0
        for i in range(s_ext_len):
            #如果i还处于最右边界线内,找对称点的RL[i]
            if i < maxright:
                RL[i] = min(RL[2*mid-i], maxright-i)
            else:
                RL[i] = 1
            #如果两边都没有超出边界且两边元素相同,则扩张到不能扩张为止
            while i >= RL[i] and i + RL[i] < s_ext_len and s_ext[i-RL[i]] == s_ext[i+RL[i]]:
                RL[i] += 1
            #如果最右边界扩张出去了,更新中心点
            if RL[i] + i - 1 > maxright:
                maxright = RL[i] + i - 1
                mid = i
            maxlen = max(maxlen, RL[i])
        final_mid = (RL.index(max(RL)) - 1) / 2
        final_long = max(RL) / 2 - 1
        return s[int(final_mid - final_long):int(final_mid + final_long + 1)]
```

4. 心得体会

- 关于算法设计

在算法设计层面,因为此前都是按照刷题-看知识点的方法来学习算法知识,所以导致了知识的不完善、不系统.这次在学习了王老师的算法课后,我受益良多.

首先是关于递归:

递归指的是调用自己的函数。

每个递归函数都有两个条件：基线条件和递归条件。

栈有两种操作：压入和弹出。

所有函数调用都进入调用栈。

调用栈可能很长，这将占用大量的内存。

然后是关于动态规划:

若要解一个给定问题，我们需要解其不同部分（即子问题），再合并子问题的解以得出原问题的解。通常许多子问题非常相似，为此动态规划法试图仅仅解决每个子问题一次，从而减少计算量：一旦某个给定子问题的解已经算出，则将其记忆化存储，以便下次需要同一个子问题解之时直接查表。

最后是关于贪心算法:

贪心选择性质：所求解的问题的整体最优解可以通过一系列局部最优的选择来，即贪心选择达到。贪心选择所依赖的是以前所做过的选择，而对以后所做的选择没有关系。

最优子结构性质：一个问题的最优解包含其子问题的最优解。

- 关于编程技术

通过本次算法实验,我系统学习了C++编程技术,发现了流式输出的优越性.而且发现了一些很方便调试的小技巧.

比如,用以下方法来选择执行的代码块.

```
#if 1
    cout << "算法" << endl;
#else
    cout << "编程" << endl;
#endif
```

还有关于类的使用,之前一直主要使用python,在使用了C++后发现类和继承的优势.继承允许用一个类来定义另一个类,这使得维护和修改程序非常的方便.

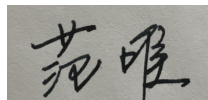
华中科技大学课程设计报告

一、原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

特此声明!

作者签字:



二、对课程设计的学术评语（教师填写）

三、对课程设计的评分（教师填写）

评分项目 (分值)	报告撰写 (50 分)	课设过程 (50 分)	最终评定 (100 分)
得分			

指导教师签字: _____