SUMMER-HOMEWORK

暑假作业

等不是办法，干才有希望

不走心的努力都是敷衍自己

比特就业课假期作业-C和数据结构作业

出题老师：

> C选择题：黄坤（day01-day08）qq：3587670086
>
> C编程题：张文超（day01-day08）qq：3627274478
>
> 数据结构选择题：吴都（day09-day16）qq：1226631755
>
> 数据结构编程题：鲍松山（day09-day16）qq：365690203

作业说明：

> 1、本次作业涵盖内容为C语言和数据结构相关知识点
>
> 2、如果对试卷上的题目，或者答案有问题，可以联系对应老师哦~~
>
> 3、同学们添加老师时备注：姓名+比特班级哦~

day01

一、选择题

1、如下代码输出的是什么（）

```
char a=101;
int sum=200;
a+=27;sum+=a;
printf("%d\n",sum);
```

A: 327 B: 99 C: 328 D: 72

2、对于下面代码执行后输出的是什么（）

```
int value = 1024;
char condition = *((char*)(&value));
if(condition) value += 1; condition = *((char*)(&value));
if(condition) value += 1; condition = *((char*)(&value));
printf("%d %d", value, condition);
```

A: 1026 1 B: 1025 0 C: 1025 1 D: 1024 0

3、假设在32位机器上，读代码选结果（）

```
void func(char para[100])
{
    void *p = malloc(100);
    printf("%d, %d\n", sizeof(para), sizeof(p));
}
```

A: 4,4 B: 100,4 C: 4,100 D: 100,100

4、以下程序执行后的输出结果为（ ）

```
#include <stdio.h>
void func(char *p) { p = p + 1; }
int main()
{
    char s[] = {'1', '2', '3', '4'};
    func(s);
    printf("%c", *s);
    return 0;
}
```

A: 2 B: 编译错误 C: 1 D: 无法确定

5、已知数组D的定义是 int D[4][8]; 现在需要把这个数组作为实参传递给一个函数进行处理。下列可以作为对应的形参变量说明的是【多选】（ ）

A: int D[4][] B: int *s[8] C: int(*s)[8] D: int D[][8]

二、编程题

1、自守数是指一个数的平方的尾数等于该数自身的自然数。请求出n(包括n)以内的自守数的个数

例如：25^2 = 625，76^2 = 5776，9376^2 = 87909376。

输入描述：int型整数

输出描述：n以内自守数的数量。

OJ链接【牛客网题号：HJ99 自守数】【难度：中等】

```
示例：
  输入：5
     2000
  输出：3
     8
  说明：对于样例一，有0，1，5，这三个自守数
```

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

int main(){

    int n = 0, cnt = 0;
    cin >> n;
    for(int i = 0; i <= n; i++)
    {
        int p = 10, j = i;
        while(j / 10) p *= 10, j /= 10;

        if(i * i % p == i) cnt++;
    }
    cout << cnt;

    return 0;
}
```

2、返回小于 N 的质数个数

输入描述：一个整数N

输出描述：小于N的质数数量

备注：0、1 不属于质数。

OJ链接【牛客网题号：OR86 返回小于 N 的质数个数】【难度：简单】

示例：
输入：10
输出：4
说明：N=10，质数有 [2, 3, 5, 7]

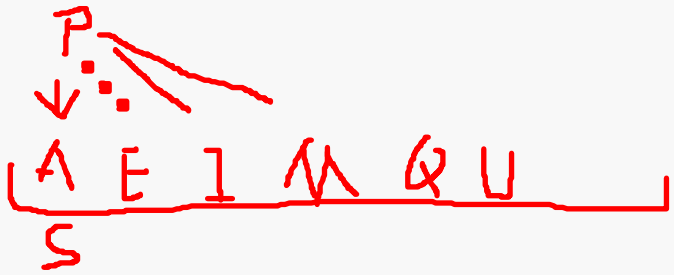
```cpp
#include<iostream>
#include<math.h>
using namespace std;

bool prime(int p)//快速判断素数
{
if (p == 2 || p == 3)
{
return true;
}
if (p % 6 != 1 && p % 6 != 5)
{
return false;
}
for (int i = 5; i <= floor(sqrt(p)); i += 6)
{
if (p % i == 0 || p % (i + 2) == 0)
{
return false;
}
}
return true;
}

int main()
{
    //输入
    int n = 0, cnt = 0;
    cin >> n;

    //素数
    for(int i = 2; i < n; i++)
        if(prime(i)) cnt++;
    //输出
    cout << cnt;

    return 0;
}
```


day02

一、选择题

1、以下程序运行后的输出结果是（ ）

```c
#include <stdio.h>
void fun(char **p)
{
    int i;
    for(i = 0; i < 4; i++)
        printf("%s", p[i]);
}
int main()
{
    char *s[6] = {"ABCD", "EFGH", "IJKL", "MNOP", "QRST", "UVWX"};
    fun(s);
    printf("\n");
    return 0;
}
```



A: ABCDEFGHIJKL B: ABCD C: AEIM D: ABCDEFGHIJKLMNOP

2、数组 a 的定义为： int a[3][4]; 下面哪个不能表示 a[1][1] （）

A: *(&a[0][0]+5) B: *(*(a+1)+1) C: *(&a[1]+1) D: *(a[1]+1)

3、 void (*s[5])(int) 表示意思为（）

A: 函数指针 B: 函数指针数组 C: 数组指针函数 D: 语法错误

4、在64位操作系统上，下面程序返回结果是（）

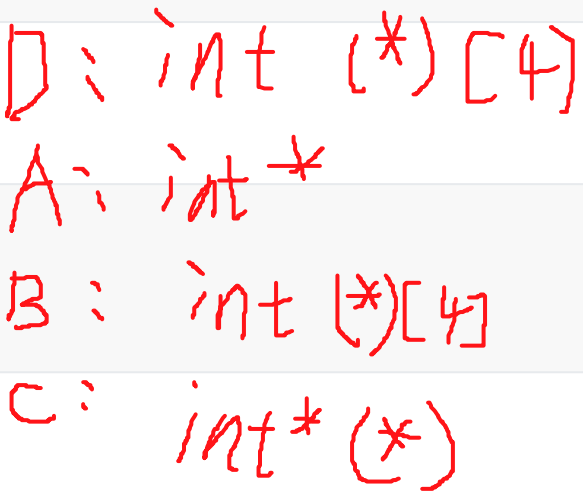
```c
int main()
{
    int *k[10][30];
    printf("%d\n", sizeof(k));
    return 0;
}
```

A: 4 B: 8 C: 1200 D: 2400

5、假设函数原型和变量说明如下，则调用合法的是（）

```c
void f(int **p);
int a[4]={1,2,3,4};
int b[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
int *q[3]={b[0],b[1],b[2]};
```

A: f(a); B: f(b); C: f(q); D: f(&a);



二、编程题

1、在一个字符串中找到第一个只出现一次的字符,并返回它的位置, 如果没有则返回 -1 （需要区分大小写）.（从 0 开始计数）

数据范围： 0≤*n*≤10000 ，且字符串只有字母组成。

要求：空间复杂度 O(n) ，时间复杂度 O(n)

OJ链接【牛客网题号：NC31 第一个只出现一次的字符】【难度：简单】

示例：
输入："google"
返回值：4

输入："aa"
返回值：-1

```
int FirstNotRepeatingChar(char* str )
{

}
```

```
class Solution {
public:
    int FirstNotRepeatingChar(string str) {
        //初始化
        const int N = 10007;
        int low_cnt[26] = {0}, up_cnt[26] = {0};

        //遍历求出现次数或map统计
        for(auto x : str)
            if(x >= 'a') low_cnt[x - 'a']++;
            else up_cnt[x - 'A']++;

        //从前往后找第一个出现的字符
       for(int i = 0; i < str.size(); i++)
           if(str[i] >= 'a')
           {
               if(low_cnt[str[i] - 'a'] == 1)
                   return i;
           }
           else if(up_cnt[str[i] - 'A'] == 1)
               return i;

        return -1;
    }
};
```

2、实现一个算法，确定一个字符串 s 的所有字符是否全都不同。

OJ链接【leetcode 题号：面试题 01.01. 判定字符是否唯一】【难度：简单】

示例：
输入: s = "leetcode"
输出: false

输入: s = "abc"
输出: true

```
bool isUnique(char* astr)
{

}
```

```
class Solution {
public:
    bool isUnique(string astr) {

        int cnt[26] = {0};

        for(auto x : astr)
            cnt[x - 'a']++;

        for(auto x : cnt)
            if(x > 1) return false;

        return true;

    }
};
```

```
}
```

day03

一、选择题

1、设有定义：`char *p;`，以下选项中不能正确将字符串赋值给字符型指针 `p` 的语句是【多选】（ ）

A: `p=getchar();` B: `scanf("%s",p);` C: `char s[]="china"; p=s;` D: `*p="china";`

2、下述程序的输出是（ ）

```
#include<stdio.h>
int main()
{
    static char *s[] = {"black", "white", "pink", "violet"};
    char **ptr[] = {s+3, s+2, s+1, s}, ***p;
    p = ptr;
    ++p;
    printf("%s", **p+1);
    return 0;
}
```

A: `ink` B: `pink` C: `white` D: `hite`

3、若有定义语句：`char s[3][10],(*k)[3],*p;`则以下赋值语句错误的是（ ）

```
1.p = s;
2.p = k;
3.p = s[0];
4.k = s;
```

A: `124` B: `1234` C: `12` D: `234`

4、假设 `sizeof(void *)` 为4，`sizeof(char)` 为1，那么对于 `char str[sizeof("ab")];`，`sizeof(str)` 的值是（ ）

A: `2` B: `3` C: `4` D: `代码无法编译`

5、有如下程序段，则对函数 `fun` 的调用语句正确的是【多选】（ ）

```
char fun(char *);
int main()
{
    char *s = "one", a[5] = {0}, (*f1)(char *) = fun, ch;
    return 0;
}
```

A: `*f1(&a);` B: `f1(*s);` C: `f1(&ch);` D: `ch = *f1(s);要改成(*f1)(s)才正确`

二、编程题

1、给定两个字符串 `s1` 和 `s2`，请编写一个程序，确定其中一个字符串的字符重新排列后，能否变成另一个字符串。

OJ链接【leetcode题号：面试题 01.02. 判定是否互为字符重排】【难度：简单】

示例：
输入: s1 = "abc", s2 = "bca"
输出: true

输入: s1 = "abc", s2 = "bad"
输出: false

输入: s1 = "aa", s2 = "bb"
输出: false

```
bool CheckPermutation(char* s1, char* s2)
{

}
```

```
class Solution {
public:
    bool CheckPermutation(string s1, string s2)
{

        int cnt1[26] = {0}, cnt2[26] = {0};

        for(auto x : s1)
            cnt1[x - 'a']++;
        for(auto x : s2)
            cnt2[x - 'a']++;

        bool flag = true;
        for(int i = 0; i < 26; i++)
            if(cnt1[i] != cnt2[i]) flag =
false;

        return flag;
    }
};
```

2、给定一个字符串，编写一个函数判定其是否为某个回文串的排列之一。回文串是指正反两个方向都一样的单词或短语。排列是指字母的重新排列。回文串不一定是字典当中的单词。

OJ链接【leetcode题号：面试题 01.04. 回文排列】【难度：简单】

示例：
输入："tactcoa"
输出：true（排列有"tacocat"、"atcocta"，等等）

```
bool canPermutePalindrome(char* s)
{

```

```
class Solution {
public:
    bool canPermutePalindrome(string s) {

        int cnt[128] = {0};
        int odd = 0;

        //全字符
        for(auto x : s) cnt[x - '\0']++;

        for(auto x : cnt)
            if(x % 2) odd++;

        if(odd > 1) return false;

        return true;
    }
};
```

}

day04

一、选择题

1、运行以下C语言代码，输出的结果是（ ）

```
#include <stdio.h>
int main()
{
   char *str[3] ={"stra", "strb", "strc"};
   char *p =str[0];
   int i = 0;
   while(i < 3)
   {
      printf("%s ",p++);
      i++;
   }
   return 0;
}
```

A: stra strb strc B: s t r C: stra tra ra D: s s s

2、下列代码输出的结果是什么（ ）

```
#include <stdio.h>
int main() {
   int m[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
   int(*p)[4] = (int(*)[4])m;
   printf("%d", p[1][2]);
   return 0;
}
```

A: 7 B: 3 C: 8 D: 4

3、下列程序的输出结果是（ ）

```
int main()
{
    char p1[15]="abcd", *p2="ABCD", str[50]="xyz";
    strcpy(str + 2, strcat(p1+2, p2+1));
    printf("%s", str);
    return 0;
}
```

A: xyabcAB B: abcABz C: ABabcz D: xycdBCD

4、以下叙述中正确的是（ ）

A: 两个字符串可以用关系运算符进行大小比较

B: 函数调用strlen(s);会返回字符串s实际占用内存的大小（以字节为单位）

C: C语言本身没有提供对字符串进行整体操作的运算符

D: 当拼接两个字符串时，结果字符串占用的内存空间是两个原串占用空间的和

5、程序的结果是什么（ ）

```
#include<stdio.h>
char* f(char* str, char ch)
{
    char* it1 = str;
    char* it2 = str;
    while(*it2 != '\0')
    {
        while (*it2 == ch)
        {
            it2++;
        }
        *it1++ = *it2++;
    }
    return str;
}
int main() {
    char a[10];
    strcpy(a, "abcdcccd");
    printf("%s", f(a, 'c'));
    return 0;
}
```

A: abdcccd B: abdd C: abcc D: abddcccd

二、编程题

1、在日常书面表达中，我们经常会碰到很长的单词，比如"localization"、"internationalization"等。为了书写方便，我们会将太长的单词进行缩写。这里进行如下定义：

如果一个单词包含的字符个数超过`10`则我们认为它是一个长单词。所有的长单词都需要进行缩写，缩写的方法是先写下这个单词的首尾字符，然后再在首尾字符中间写上这个单词去掉首尾字符后所包含的字符个数。

比如`"localization"`缩写后得到的是`"l10n"`，`"internationalization"`缩写后得到的是`"i18n"`。

现给出`n`个单词，将其中的长单词进行缩写，其余的按原样输出。

输入描述：

- 第一行包含要给整数n。1≤n≤100
- 接下来n行每行包含一个由小写英文字符构成的字符串，字符串长度不超过100。

输出描述：按顺序输出处理后的每个单词。

OJ链接【牛客网题号：`OR135`单词缩写】【难度：简单】

示例：

输入：4
word
localization
internationali
pneumonoult

输出：
word
l10n
i18n
p43s

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

string str;
int n;

int main()
{
    cin >> n;

    while(n --)
    {
        cin >> str;
        int l = str.size();

        if(l < 10) cout << str << endl;
        else{
            cout << str[0] << l - 2 << str[l - 1] << endl;
        }
    }

    return 0;
}
```

2、URL化。编写一种方法，将字符串中的空格全部替换为`%20`。假定该字符串尾部有足够的空间存放新增字符，并且知道字符串的“真实”长度。

OJ链接【`leetcode`题号：面试题 01.03. URL化】【难度：简单】

示例：
输入："Mr John Smith   ", 13
输出："Mr%20John%20Smith"

输入："   ", 5
输出："%20%20%20%20%20"

```
char* replaceSpaces(char* S, int ler
{



}
```

```
class Solution {
public:
    string replaceSpaces(string S, int length) {
        int spa_cnt = 0;
        for(int i = 0; i < length; i++)
            if(S[i] == ' ') spa_cnt++;
        for(int i = length + spa_cnt * 2 - 1, j = length - 1; j
>= 0; j--)
        {
            if(S[j] != ' ')
            {
                S[i] = S[j];
                i--;
            }
            else
            {
                S[i] = '0',S[i - 1] = '2', S[i - 2] = '%';
                i -= 3;
            }
        }
    S.resize(length + spa_cnt * 2);//设置大小：原字符串长度+新
增
    return S;
    }
};
```


day05

一、选择题

1、以下程序的输出结果是（ ）

```
int main()
{
   char arr[2][4];
   strcpy (arr[0],"you");
   strcpy (arr[1],"me");
   arr[0][3]='&';
   printf("%s \n",arr);
   return 0;
}
```

A: you&me B: you C: me D: err

2、如下C程序，在64位处理器上运行后 sz 的值是什么（ ）

```
struct st
{
   int *p;
   int i;
   char a;
};
int sz = sizeof(struct st);
```

A: 12 B: 16 C: 24 D: 13

3、执行以下语句后的输出结果是（ ）

```
enum weekday
{
   sun,
   mon = 3,
   tue,
   wed
};
enum weekday workday;
workday = wed;
printf("%d\n",workday);
```

A: 3 B: 4 C: 5 D: 编译错误

4、设有以下定义，则下面叙述中正确的是【多选】（ ）

```
union D
{
   int d1;
   float d2;
}d;
```

A: 变量d与各成员的地址相同 B: d.d1和d.d2具有相同的地址

C: 若给d.d2赋10后,d.d1中的值是10 D: 若给d.d1赋10后,d.d2中的值是10

5、假设C语言程序里使用 malloc 申请了内存，但是没有 free 掉，那么当该进程被kill之后，操作系统会（ ）

A: 内存泄露 B: segmentation fault C: core dump D: 以上都不对

二、编程题

1、字符串压缩。利用字符重复出现的次数，编写一种方法，实现基本的字符串压缩功能。

比如，字符串 aabcccccaaa 会变为 a2b1c5a3 。若“压缩”后的字符串没有变短，则返回原先的字符串。你可以假设字符串中只包含大小写英文字母（a至z）。

OJ链接【leetcode 题号：面试题 01.06. 字符串压缩】【难度：简单】

示例：
输入："aabcccccaaa"
输出："a2b1c5a3"

输入："abbccd"
输出："abbccd"
解释："abbccd"压缩后为"a1b2c2d1"，

```
char* compressString(char* S)
{



}
```

```
class Solution {
public:

    //求数位
    int pnum(int num)
    {
        int x = 10;
        while(num >= x) x *= 10;

        return x / 10;
    }
    string compressString(string S) {

        if(S.size() < 2) return S;// 0 || 1

        string tmp;
        int cnt = 0;
        for(int i = 0, j = 0; j < S.size(); j++)
        {
            if(S[j] == S[i]) cnt++;//记录
            else
            {
                tmp.push_back(S[i]);
                int x = pnum(cnt);
                while(x)//用x判断可避免 x = cnt 时， 尾0未插入
                {
                    tmp.push_back(cnt / x + '0');
                    cnt %= x;
                    x /= 10;
                }
               cnt = 0; i = j--;
            }
        }
        //最后一个字符
        tmp.push_back(S[S.size() - 1]);
        int x = pnum(cnt);
        while(cnt)
        {
            tmp.push_back(cnt / x + '0');
            cnt %= x;
            x /= 10;
        }
        //输出
        if(tmp.size() >= S.size()) return S;
        else return tmp;
    }
};
```

2、配对交换。编写程序，交换某个整数的奇数位和偶数位，尽量使用较少的指令（也就是说，位0与位1交换，位2与位3交换，以此类推）。

OI链接【leetcode 题号：面试题 05.07. 配对交换】【难度：简单】

示例：
输入：num = 2（或者0b10）
输出 1 (或者 0b01)

输入：num = 3
输出：3

```
class Solution {
public:
    int exchangeBits(int num) {
        int o = 0x55555555;
        int t = 0xaaaaaaaa;

        int a = num & o, b = num & t;

        num = (a << 1) + (b >> 1);

        return num;
    }
};
```

```
int exchangeBits(int num)
{




}
```

day06

一、选择题

1、关于内存管理，以下有误的是（ ）

A: malloc在分配内存空间大小的时候是以字节为单位

B: 如果原有空间地址后面还有足够的空闲空间用来分配，则在原有空间后直接增加新的空间，使得增加新空间后的空间总大小是：newSize

C: 如果原有空间地址后面没有足够的空闲空间用来分配，那么从堆中另外找一块newsize大小的内存，并把先前内存空间中的数据复制到新的newSize大小的空间中，然后将之前空间释放

D: free函数的作用是释放内存，内存释放是标记删除，会修改当前空间的所属状态，并且会清除空间内容

2、如下程序输出的结果是什么（ ）

```
#include <stdio.h>
typedef struct List_t
{
   struct List_t* next;
   struct List_t* prev;
   char data[0];
}list_t;
int main()
{
   printf("%d",sizeof(list_t));
   return 0;
}
```

柔性数组不计算大小

A: 4byte B: 8byte C: 5byte D: 9byte

3、以下程序的输出结果是（ ）

```
#include<stdio.h>
void fut(int**s,int p[2][3])
{
   **s=p[1][1];
}
int main()
{
   int a[2][3]={1,3,5,7,9,11},*p;
   p=(int*)malloc(sizeof(int));
   fut(&p,a);
   printf("%d",*p);
   return 0;
}
```

A: 7 B: 9 C: 1 D: 11

4、以下有关C语言的说法中，错误的是（ ）

A: 内存泄露一般是指程序申请了一块内存，使用完后，没有及时将这块内存释放，从而导致程序占用大量内存，但又不使用不释放

B: 可以通过malloc(size_t)函数调用申请超过该机器物理内存大小的内存块

C: 无法通过内存释放函数free(void*)直接将某块已经使用完的内存直接还给操作系统，free函数只是将动态申请内存的使用权释放

D: 可以通过内存分配函数malloc(size_t)直接申请物理内存

5、若要用 fopen 函数打开一个新的二进制文件，该文件既能读也能写，则文件方字符串应是（ ）

A: "ab+" B: "wb+" C: "rb+" D: "ab"

二、编程题

1、递归乘法。 写一个递归函数，不使用 * 运算符， 实现两个正整数的相乘。可以使用加号、减号、位移，但要吝啬一些。

注意：题目要求递归实现。

OJ链接【leetcode 题号：面试题 08.

```
示例:
  输入：A = 1, B = 10
  输出：10

  输入：A = 1, B = 10
  输出：10
```

```
class Solution {
public:
    int multiply(int A, int B) {

        if(A > B) swap(A, B);//减少递归次数

        if(A == 0) return 0;
        if(A == 1) return B;

        int a = A >> 1;
        int sum = multiply(a, B);//记录值，减少递归次数

        if(A & 1) return B + sum + sum;
        else return sum + sum;
    }
};
```

```
int multiply(int A, int B)
{




}
```

2、输出小于等于 n 的与 7 有关数字的个数 …… 71，72， 73...）的个数（一组测试用例里可能有……

数据范围： `1 <= n <= 3000`

输入描述：多组输入每组输入 1 个正整数 n

输出描述：不大于n的与7有关的数字个数，……

OJ链接【牛客网题号： HJ55 挑7】【难度：……

```
示例:
  输入：20
        10

    输出：3
         1
```

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

bool IS7(int num)
{
    while(num)
    {
        if(num % 10 == 7) return true;
        num /= 10;
    }
    return false;
}

int main(){

    int n = 0, cnt = 0;
    cin >> n;
    for(int i = 1; i <= n; i++)
        if(i % 7 == 0 || IS7(i)) cnt++;

    cout << cnt;

    return 0;

}
```

day07

一、选择题

1、函数rewind的作用是（ ）

A: 使位置指针重新返回文件的开头 B: 将位置指针指向文件中所要求的特定位置

C: 使位置指针指向文件的末尾 D: 使位置指针自动移至下一个字符位置

2、以下可作为函数 `fopen` 中第一个参数的正确格式是（ ）

A: `c:user\text.txt` B: `c:\user\text.txt`

C: `"c:\user\text.txt"` D: `"c:\\user\\text.txt"`

3、下面的程序执行后，文件 `test.txt` 中的内容是（ ）

```
#include<stdio.h>
void fun (char *fname,char *st)
```

```
{
    FTLE *myf;
    int i;
    myf = fopen(fname,"w");
    for (i = 0; i < strlen(st); i++) fputc(st[i], myf);
    fclose(myf);
}
int main()
{
    fun("test.txt","new world");
    fun("test.txt","hello,");
    return 0;
}
```

A: hello, B: new worldhello, C: new world D: hello,rld

4、函数（ ）把文件位置重定位到文件中的指定位置

A: fseek B: fread C: fopen D: fgets

5、若调用 fputc 函数输出字符成功，则其返回值是（ ）

A: EOF B: 1 C: 0 D: 输出的字符

二、编程题

1、设计一个算法，算出 n 阶乘有多少个尾随零。

OJ链接【leetcode 题号：面试题 16.05. 阶乘尾数】【难度：简单】

示例:
输入: 3
输出: 0
解释: 3! = 6, 尾数中没有零。

输入: 5
输出: 1
解释: 5! = 120, 尾数中有 1 个零.

```
int trailingZeroes(int n)
{
```

```
class Solution {
public:
    int trailingZeroes(int n) {
        int ans = 0;
        while (n) {
            n /= 5;
            ans += n;
        }
        return ans;
    }
};
```

```
}
```

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

int main(){

    double n = 0; cin >> n;
    double d = n - (int)n;

    cout << (d >= 0.5 ? (int)n + 1 : (int)
n);

    return 0;
}
```

2、写出一个程序，接受一个正浮点数值，输出i ,向上取整；小于 0.5 ，则向下取整。

数据范围：保证输入的数字在 32 位浮点数范围

输入描述：输入一个正浮点数值

输出描述：输出该数值的近似整数值

OJ链接【牛客网题号： HJ7 取近似值】【难度:

```
示例:
  输入：5.5
    输出：6
    说明：0.5>=0.5，所以5.5需要向上取整为6
```

day08

一、选择题

1、下列关于 const 和 #define 定义常量的区别，说法不正确的有（ ）

A: define宏是在预处理阶段展开。const常量是编译运行阶段使用

B: 宏没有类型，不做任何类型检查，仅仅是展开。const常量有具体的类型，在编译阶段会执行类型检查

C: define宏仅仅是展开，有多少地方使用，就展开多少次，不会分配内存。const常量会在内存中分配（可以是堆中也可以是栈中）

D: const定义和#define定义的常量在程序运行过程中只有一份拷贝

2、程序最后输出什么（ ）

```
#include<stdio.h>
#define Mul(x,y) ++x*++y
int main()
{
   int a = 1;
   int b = 2;
   int c = 3;
   printf("%d",Mul(a+b,b+c));
   return 0;
}
```

A: 14 B: 18 C: 24 D: 问题表达式，结果不确定

3、下述有关预编译和编译，说法错误的是（ ）

A: C语言由源代码生成的各阶段如下，C源程序 - 编译预处理 - 编译 - 优化程序 - 汇编程序 - 链接程序 - 可执行文件

B: 常见的预编译指令有#include，#define，#if、#else和#endif

C: 编译程序可以识别一些特殊的符号，比如__LINE__，表示当前行号的整数，这些是在编译阶段处理的

D: #define定义宏，可以多次使用

4、下面选项中关于编译预处理的叙述正确的是（ ）

A: 预处理命令行必须使用分号结尾 B: 凡是以#号开头的行，都被称为编译预处理命令行

C: 预处理命令行不能出现在程序的最后一行 D: 预处理命令行的作用域是到最近的函数结束处

整个文件

5、以下代码的输出结果是（ ）

```
#include <stdio.h>
#define a 10
void foo();
int main()
{
   printf("%d..", a);
   foo();
   printf("%d", a);
   return 0;
}
void foo(){
   #undef a
   #define a 50
}
```

A: 10..10 B: 10..50 C: Error D: 0

二、编程题

1、某种特殊的数列 a1, a2, a3, ... 的定义如下：$a_1 = 1, a_2 = 2, \ldots, a_n = 2 * a_{n-1} + a_{n-2}$其中(n > 2)。给出任意一个正整数k，求该数列的第k项模以32767的结果是多

输入描述：

- 第1行是测试数据的组数 n，后面跟着 n 行输入。
- 每组测试数据占1行，包括一个正整数 k (1 ≤ k < 1

输出描述：n 行，每行输出对应一个输入。输出应是一

OJ链接【牛客网题号：OR140 数列】【难度：简单】

```
#include<iostream>
#include<stdlib.h>

using namespace std;
//打表--减少重复操作
int a[1000001] = {0};

int main()
{
    //建表
    for(int i = 1; i <= 1000000; i++)
    {
        if(i <= 2) a[i] = i;
        else a[i] = (2 * a[i - 1] + a[i - 2]) %
32767;
    }
    int T = 0; cin >> T;
    while(T--)
    {
        int n = 0; cin >> n;
        cout << a[n] << endl;
    }
    return 0;
}
```

```
#include <iostream>
#include <string.h>
#include <string>
using namespace std;

int main(){

    int num = 0;
    int eng = 0;
    int spa = 0;
    int oth = 0;

    string str;
    getline(cin, str);//读取一行。可读取空格
    for(auto x : str)
        if(x == ' ') spa++;
        else if((x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z')) eng++;
        else if(x >= '0' && x <= '9') num++;
        else oth++;

    cout << eng << endl << spa << endl << num << endl << oth;
    return 0;
}
```

示例：
输入：2
1
8
输出：
1
408

2、输入一行字符

数据范围：输入的字符串长度满足 1 <= n <= 1000

输入描述：输入一行字符串，可以有空格

输出描述：统计其中英文字符，空格字符，数字字符，其他字符的个数

OJ链接【牛客网题号：HJ40 统计字符】【难度：中等】

示例：
输入：1qazxsw23 edcvfr45tgbn hy67uj m,ki89ol.\\/;p0-=\\][
输出：26
3
10
12

day09

一、选择题

1、下面算法的时间复杂度是( )

```
void Func(int N) {
    int count = 0;
    for (int k = 0; k < 2 * N * N ; ++ k){
        ++count;
    }
    int M = 10;
    while (M--){
        ++count;
    }
    printf("%d\n", count);
}
```

A. O(M) B. O(N) C. O(N^2) D. O(2N^2)

2、下面算法的时间复杂度是( )

```
int func(unsigned int n) {
  if (n == 0 || n==1)
      return 1;
  else
      return n * func(n-1);
}
```

A. O(n^2) B. O(nlogn) C. O(n) D. O(logn)

3、以下算法的时间复杂度为( )

```
void fun(int n) {
   int i=l;
   while(i<=n){
      i=i*2;
   }
}
```

A. O(n) B. O(n2) C. O(nlog2n) D. O(log2n)

4、动态顺序表中，( )操作需要检查是否需要扩容

A.删除 B.插入 C.初始化 D.清空

5、在长度为 X 的顺序表下标为 i 的位置前插入一个元素（ 1 ≤ i ≤ X+1 ），元素的移动次数为( )

A. X - i + 1 B. X - i C. X D. X-1

二、编程题

1、一维数组的动态和

给你一个数组 nums 。数组「动态和」的计算公式为： runningSum[i] = sum(nums[0]...nums[i]) 。请返回 nums 的动态和。OJ链接【 LeetCode 题号: 1480.一维数组的动态和】【简单】

```
示例 1：
输入：nums = [1,2,3,4]
输出：[1,3,6,10]
解释：动态和计算过程为 [1, 1+2, 1+2+3, 1+2+3+4] 。

示例 2：
输入：nums = [1,1,1,1,1]
输出：[1,2,3,4,5]
解释：动态和计算过程为 [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1] 。

示例 3：
输入：nums = [3,1,2,10,1]
输出：[3,4,6,16,17]
```

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* runningSum(int *nums, int numsSize, int *returnSize)
{
```

```
class Solution {
public:
    vector<int> runningSum(vector<int>& nums) {
        vector<int> res = nums;

        for(int i = 1; i < res.size(); i++)
            res[i] += res[i - 1];


        return res;

    }
};
```

```
}
```

2、搜索插入位置

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。请必须使用时间复杂度为 `O(log n)` 的算法。

OJ链接【LeetCode 题号: 35. 搜索插入位置】【简单】

示例 1:
输入: nums = [1,3,5,6], target = 5
输出: 2

示例 2:
输入: nums = [1,3,5,6], target = 2
输出: 1

示例 3:
输入: nums = [1,3,5,6], target = 7
输出: 4

示例 4:
输入: nums = [1,3,5,6], target = 0
输出: 0

示例 5:
输入: nums = [1], target = 0
输出: 0

```
int searchInsert(int *nums, int numsSize, int target)
{
```

```
class Solution {
public:
    int searchInsert(vector<int>& nums, int tar) {

        int l = 0, r = nums.size() - 1;

        while(l < r)
        {
            int mid = l + r >> 1;

            if(nums[mid] >= tar) r = mid;
            else l = mid + 1;
        }

        if(nums[l] < tar) l++;//插入到最后
        return l;
    }
};
```

```
}
```

day10

一、选择题

1、下列数据结构中，不属于线性表的是( )

A.队列 B.顺序表 C.二叉树 D.链表

2、对于一个头指针为 head 的带头结点的单链表，判断该表为空的条件是（ ）

A. head=NULL B. head→next==NULL C. head→next=head D. head!=NULL

3、当线性表的元素总数基本稳定，且很少进行插入和删除操作，但要求以最快的速度存取线性表中的元素时，应采用什么存储结构？（ ）

A. 顺序表 B. 单链表 C. 循环链表 D. 双链表

4、带头指针 L 的双向循环链表中，指针 p 指向双向循环链表的尾结点的条件是:( )

A. p->prior == L&&L->next==p

B. p->next == L&&L->prior==p

C. L->prior==p

D. p->next==NULL

5、已知 pPre 为指向链表中某结点的指针， pNew 是指向新结点的指针，以下哪段伪码算法是将一个新结点插入到链表中 pPre 所指向结点的后面？（ ）

A. pPre->link = pNew; pNew = null;

B. pPre->link = pNew->link; pNew->link = null;

C. `pNew->link = pPre->link;` `pPre->link = pNew;`

D. `pNew->link = pPre->link;` `pPre->link = null;`

二、编程题

1、搜索旋转排序数组

整数数组 `nums` 按升序排列，数组中的值**互不相同**。在传递给函数之前，`nums` 在预先未知的某个下标 `k`（`0 <= k < nums.length`）上进行了 **旋转**，使数组变为 `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]`（下标 **从 0 开始**计数）。例如，`[0,1,2,4,5,6,7]` 在下标 3 处经旋转后可能变为 `[4,5,6,7,0,1,2]` 。给你 **旋转后**的数组 `nums` 和一个整数 `target` ，如果 `nums` 中存在这个目标值 `target` ，则返回它的下标，否则返回 -1 OJ链接【`LeetCode` 题号: 33. 搜索旋转排序数组】【中等】

```
示例 1：
输入：nums = [4,5,6,7,0,1,2], target = 0
输出：4

示例 2：
输入：nums = [4,5,6,7,0,1,2], target = 3
输出：-1

示例 3：
输入：nums = [1], target = 0
输出：-1
```

```
int search(int* nums, int numsSize,
{



}
```

```
class Solution {
public:
    int search(vector<int>& nums, int target) {

        int l = 0, r = nums.size() - 1;

        while(l < r)
        {
            int mid = l + r >> 1;
            if(nums[mid] == target) return mid;

            if(nums[mid] >= nums[l])//左半部
            {
                if(nums[mid] >= target && nums[l] <= target) r =
mid - 1;//在[l, mid)
                else l = mid + 1;//(mid, r]
            }
            else//右半部
            {
                if(nums[mid] <= target && nums[r] >= target) l =
mid + 1;//在(l, mid]
                else r = mid - 1; //[l, mid)
            }
        }

        if(nums[l] == target) return l;
        else return -1;
    }
};
```

2、二进制链表转整数

给你一个单链表的引用结点 head。链表中每个结点的值不是 0 就是 1。已知此链表是一个整数数字的二进制表示形式。请你返回该链表所表示数字的**十进制值**。OJ链接【`LeetCode` 题号: 1290. 二进制链表转整数】【简单】

示例 1：
输入：head = [1,0,1]
输出：5
解释：二进制数 (101) 转化为十进制数 (5)

示例 2：
输入：head = [0]
输出：0

示例 3：
输入：head = [1]
输出：1

示例 4：
输入：head = [1,0,0,1,0,0,1,1,1,0,0,0,0,0,0]
输出：18880

示例 5：
输入：head = [0,0]
输出：0

提示：
链表不为空。链表的结点总数不超过

```
int getDecimalValue(struct ListNo
{




}
```

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    int getDecimalValue(ListNode* head) {
        int res = 0;
        while(head)
        {
            res <<= 1;
            res += head->val;
            head = head->next;
        }

    return res;
    }
};
```

day11

一、选择题

1、若栈采用链式存储结构,则下列说法中正确的是（ ）

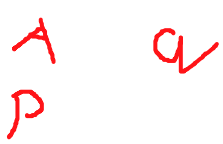
A.需要判断栈满但不需要判断栈空

B.不需要判断栈满也不需要判断栈空

C.需要判断栈满且需要判断栈空

D.不需要判断栈满但需要判断栈空

2、设指针变量 p 指向单链表中结点 A，若删除单链表中结点 A，则需要修改指针的操作序列为（）。



A. `q=p->next; p->data=q->data; p->next=q->next; free(q);`

B. `q=p->next; q->data=p->data; p->next=q->next; free(q);`

C. `q=p->next; p->next=q->next; free(q);`

D. `q=p->next; p->data=q->data; free(q);`

3、用链表表示线性表的优点是 （ ）。

A.便于随机存取 B.花费的存储空间比顺序表少

C.便于插入与删除 D.数据元素的物理顺序与逻辑顺序相同

4、队列是一种（ ）的线性表。

A. 先进先出 B. 先进后出 C. 只能插入 D. 只能删除

5、和顺序栈相比，链栈有一个比较明显的优势是（ ）

A. 通常不会出现栈满的情况 B. 通常不会出现栈空的情况

C. 插入操作更容易实现 D. 删除操作更容易实现

二、编程题

1、表内指定区间反转

将一个节点数为 `size` 链表 m 位置到 n 位置之间的区间反转，要求时间复杂度 $O(n)$，空间复杂度 $O(1)$。 例如：给出的链表为 `1 -> 2 -> 3 -> 4 -> 5 -> NULL`，`m=2,n=4` 返回 `1 -> 4 -> 3 -> 2 -> 5 -> NULL`

OJ链接【NC题号: 21. 链表内指定区间反转】【简单】

示例1
输入：{1,2,3,4,5},2,4
返回值：{1,4,3,2,5}

示例2
输入：{5},1,1
返回值：{5}

```
struct ListNode* revers
{
```

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int m, int n) {
        // write code here
        if((n == m && m == 1) || !head->next) return head;

        //虚拟头指针，避免头指针逆转
        ListNode *hhead = new ListNode(0);
        hhead->next = head;
        //找逆转部分的前一个节点
        ListNode *left = hhead;
        int cnt = 0;
        while(cnt < m - 1)
        {
            cnt ++;
            left = left->next;
        }
        //逆转
        ListNode *cur = left->next; cnt++;
        ListNode *pre, *ne;
        while(cnt <= n)
        {
            cnt++;
            ne = cur->next;
            cur->next = pre;
            pre = cur;
            cur = ne;
        }
        //链接
        left->next->next = cur;
        left->next = pre;
        return hhead->next;
    }
};
```

```
}
```

2、从链表中删去总和值为零的连续节点

给你一个链表的头节点 `head`，请你编写代码，反复删去链表中由 **总和**值为 0 的连续节点组成的序列，直到不存在这样的序列为止。删除完毕后，请你返回最终结果链表的头节点。你可以返回任何满足题目要求的答案。

OJ链接【LeetCode 题号: 1171. 从链表中删去总和值为零的连续节点】【中等】

```
示例 1：
输入：head = [1,2,-3,3,1]
输出：[3,1]
提示：答案 [1,2,1] 也是正确的。

示例 2：
输入：head = [1,2,3,-3,4]
输出：[1,2,4]

示例 3：
输入：head = [1,2,3,-3,-2]
输出：[1]
```

```
struct ListNode* removeZeroSumSu
{



}
```

```
class Solution {
public:
    map<int, ListNode*> msp;
    ListNode* removeZeroSumSublists(ListNode* head) {
        ListNode *hhead = (ListNode*)malloc(sizeof
(ListNode));
        hhead->val = 0, hhead->next = head;
        msp.insert({0, hhead});
        int sum = 0;

        while(head)
        {
            sum += head->val;
            if(msp.count(sum))
            {
                ListNode * cur = msp[sum]->next;
                ListNode * ne = cur->next;
                while(cur != head)
                {
                    sum += cur->val;
                    msp.erase(sum);
                    cur = cur->next;
                }

                sum += head->val;
                msp[sum]->next = head->next;
                head = head->next;
            }
            else
            {
                msp[sum] = head;
                head = head->next;
            }
        }
        return hhead->next;
    }


};
```

day12

一、选择题

1、下列关于队列的叙述中正确的是（ ）

A. 在队列中只能插入数据 B. 在队列中只能删除数据

C. 队列是先进先出的线性表 D. 队列是先进后出的线性表

2、设有四个元素 A、B、C、D 顺序进栈，在进栈过程中可以出栈，出栈次序错误的排列是（ ）

A. ABCD B. DCBA C. ACBD D. DCAB

3、字符 A 、 B 、 C 依次进入一个栈，按出栈的先后顺序组成不同的字符串，至多可以组成 （ ）个不同的字符串?

A. 14 B. 5 C. 6 D. 8

4、已知栈 s 允许在两端出栈，但只允许在一端入栈;队列 Q 只允许在一 端入队列，在另一端出队列。设栈 s 和队列 Q 的初始状态为空，e1,e2,e3, e4,e5,e6 依次通过栈 S ,一个元素出栈后即进队列 Q ，则不可能得到的出队列的顺序是（ ）

A. e2,e4,e3,e5,e1,e6 B. e2,e5,e1,e3,e4,e6

C. e5,e1,e6,e3,e2,e4 D. e4,e1,e3,e5,e2,e6

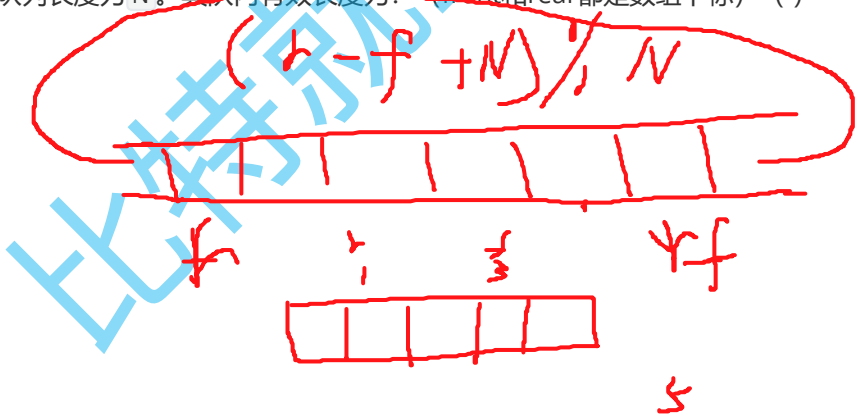
5、现有使用数组实现的一个循环队列， front 指向队列的首元素， rear 指向队列末尾元素的下一个位置， rear指向的位置不保存元素；循环队列长度为 N 。其队内有效长度为? （front和rear都是数组下标） （ ）

A. (rear - front + N) % N + 1

B. (rear - front + N) % N

C. (rear - front) % (N + 1)

D. (rear - front + N) % (N - 1)



二、编程题

1、链表求和

给定两个用链表表示的整数，每个节点包含一个数位。这些数位是反向存放的，也就是个位排在链表首部。编写函数对这两个整数求和，并用链表形式返回结果。OJ链接【 LeetCode 题号: 02.05. 链表求和】【中等】

```
示例:
输入：(7 -> 1 -> 6) + (5 -> 9 -> 2)，即617 + 295
输出：2 -> 1 -> 9，即912
```

```
struct ListNode* addTwoNumbers(struct ListNode* l1, struct ListNode* l2)
{
```

```
class Solution {
public:
    ListNode* addTwoNumbers(ListNode* l1, ListNode* l2) {

        int ne = 0;
        ListNode* phead = new ListNode(0);
        ListNode* pre = phead;

        while(l1 || l2 || ne)
        {
            if(l1)
            {
                ne += l1->val;
                l1 = l1->next;
            }
            if(l2)
            {
                ne += l2->val;
                l2 = l2->next;
            }

           pre->next = new ListNode(ne % 10);
           ne /= 10;
           pre = pre->next;

        }
        pre = nullptr;

        return phead->next;

    }
};
```

2、括号的最大嵌套深度

如果字符串满足以下条件之一，则可以称之为 **有效括号字符串（valid parentheses string，可以简写为** `VPS`）：

- 字符串是一个空字符串 ""，或者是一个不为 "(" 或 ")" 的单字符。
- 字符串可以写为 AB（A 与 B 字符串连接），其中 A 和 B 都是 **有效括号字符串** 。
- 字符串可以写为 (A)，其中 A 是一个 **有效括号字符串** 。

类似地，可以定义任何有效括号字符串 S 的 **嵌套深度** `depth(S)`：

- depth("") = 0
- depth(C) = 0，其中 C 是单个字符的字符串，且该字符不是 "(" 或者 ")"
- depth(A + B) = max(depth(A), depth(B))，其中 A 和 B 都是 **有效括号字符串**
- depth("(" + A + ")") = 1 + depth(A)，其中 A 是一个 **有效括号字符串**

例如：""、"()()"、"()(()())" 都是 **有效括号字符串**（嵌套深度分别为 0、1、2），而 ")(" 、"(()" 都不是 **有效括号字符串** 。给你一个 **有效括号字符串** s，返回该字符串的 s **嵌套深度** 。

OJ链接【`LeetCode` 题号: 1614. 括号的最大嵌套深度】【简单】

示例 1：
输入：s = "(1+(2*3)+((8)/4))+1"
输出：3
解释：数字 8 在嵌套的 3 层括号中。

示例 2：
输入：s = "(1)+((2))+(((3)))"
输出：3

示例 3：

输入：s = "1+(2*3)/(2-1)"
输出：1

示例 4：
输入：s = "1"
输出：0

```
int maxDepth(char *s)
{

}
```

```
class Solution {
public:
    int maxDepth(string s) {
        int res = 0, Max = 0;

        for(auto x : s)
        {
            if(x == '(')
                res ++;
            else if(x == ')')
                res --;
            Max = max(Max, res);
        }

        return Max;
    }
};
```

day13

一、选择题

1、设栈 S 和队列 Q 的初始状态均为空，元素 a，b，c，d，e，f，g 依次进入栈 S。若每个元素出栈后立即进入队列 Q，且7个元素出队的顺序是 b，d，c，f，e，a，g，则栈 S 的容量至少是（）。

A. 1 B. 2 C. 3 D. 4

2、一个栈的入栈序列为 1,2,3,...,n，其出栈序列是 p 1 ,p 2 ,p 3 ,...p n 。若 p 2 = 3，则 p 3 可能取值的个数是（）

A. n-3 B. n-2 C. n-1 D.无法确定

3、已知栈 S 的初始状态为空，元素 a,b,c,d,e,f 依次入栈 S，出栈的序列为 b,d,c,f,e,a 则栈S的容量至少为（）

A. 6 B. 5 C. 4 D. 3

4、假设以数组A[60]存放循环队列的元素,其头指针是front=47,当前队列有50个元素,则队列的尾指针值为( )

A.3 B.37 C.97 D.50

5、若用一个大小为 6 的数组来实现循环队列，且当 rear 和 front 的值分别为 0 和 3 。当从队列中删除一个元素，再加入两个元素后，rear 和 front 的值分别为( )

A.1和5 B.2和4 C.4和2 D.5和1

二、编程题

1、整理字符串

给你一个由大小写英文字母组成的字符串 `s` 。

一个整理好的字符串中，两个相邻字符 `s[i]` 和 `s[i+1]`，其中 `0<= i <= s.length-2`，要满足如下条件:

- 若 `s[i]` 是小写字符，则 `s[i+1]` 不可以是相同的大写字符。
- 若 `s[i]` 是大写字符，则 `s[i+1]` 不可以是相同的小写字符。

请你将字符串整理好，每次你都可以从字符串中选出满足上述条件的 **两个相邻**字符并删除，直到字符串整理好为止。请返回整理好的**字符串**。题目保证在给出的约束条件下，测试样例对应的答案是唯一的。**注意：**空字符串也属于整理好的字符串，尽管其中没有任何字符。OJ链接【LeetCode题号: 1544.整理字符串】【简单】

```
示例 1：
输入：s = "leEeetcode"
输出："leetcode"
解释：无论你第一次选的是 i = 1 还是 i = 2，都会使 "leEeetcode" 缩减为 "leetcode" 。

示例 2：
输入：s = "abBAcC"
输出：""
解释：存在多种不同情况，但所有的情况都会导致相同的结果。例如：
"abBAcC" --> "aAcC" --> "cC" --> ""
"abBAcC" --> "abBA" --> "aA" --> ""

示例 3：
输入：s = "s"
输出："s"
```

```
char * makeGood(char *s)
{



}
```

```
class Solution {
public:
    string makeGood(string s) {

        stack<char> sta;
        sta.push('0');

        for(auto x : s)
        {
            if(islower(x))//小写
            {
                if(x - 32 == sta.top()) sta.pop();
                else sta.push(x);
            }
            else//大写
            {
                if(x + 32 == sta.top()) sta.pop();
                else sta.push(x);
            }
        }
        string res;
        while(sta.size() > 1)
        {
            res += sta.top();
            sta.pop();
        }
        reverse(res.begin(), res.end());

        return res;

    }
};
```

2、开幕式焰火

开幕式开始了，空中绽放了一颗二叉树形的巨型焰火。给定一棵二叉树 root 代表焰火，节点值表示巨型焰火这一位置的颜色种类。请帮小扣计算巨型焰火有多少种不同的颜色。

OJ链接【LeetCode 题号: LCP 44. 开幕式焰火】【简单】

```
示例 1：
输入：root = [1,3,2,1,null,2]
输出：3
解释：焰火中有 3 个不同的颜色，值分别为 1、2、3

示例 2：
输入：root = [3,3,3]
输出：1
解释：焰火中仅出现 1 个颜色，值为 3
```

```cpp
class Solution {
public:

    unordered_set<int> s;

    void dfs(TreeNode* root)
    {
        if(root == nullptr) return ;

        s.insert(root->val);
        dfs(root->left);
        dfs(root->right);
    }

    int numColor(TreeNode* root) {

        dfs(root);

        return s.size();

    }
};
```

day14

一、选择题

1、设非空二叉树的所有子树中，其左子树上的结点值均小于根结点值，而右子树上的结点值均不小于根结点值，则称该二叉树为排序二叉树。对排序二叉树的遍历结果为有序序列的是（ ）

A.中序序列 B.前序序列 C.后序序列 D.前序序列或后序序列

2、一棵有15个节点的完全二叉树和一棵同样有15个节点的普通二叉树，叶子节点的个数最多会差多少个?( )

A. 3 B. 5 C. 7 D. 9

3、一棵有 n 个结点的二叉树，按层次从上到下、同一层从左到右顺序存储在一维数组 A[1..n] 中，则二叉树中第 i 个结点（i从1开始用上述方法编号）的右孩子在数组 A 中的位置是（ ）

A. A[2*i](2*i <= n)

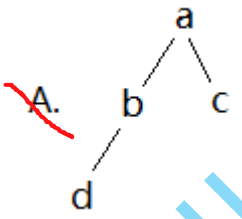
B. A[2*i + 1](2*i + 1 <= n)

C. A[i - 2]

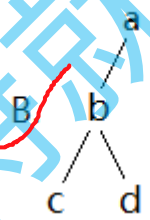
D. 条件不充分,无法确定

4.中序遍历为 abcd 的二叉树可能是下面的哪棵？（）【不定项选择】

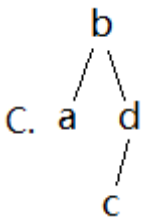
A.



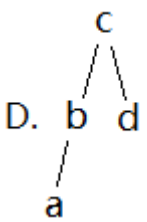
B.



C.



D.



5、已知-算术表达式的中缀表达式为 a-(b+c/d)*e，其后缀形式为（ ）

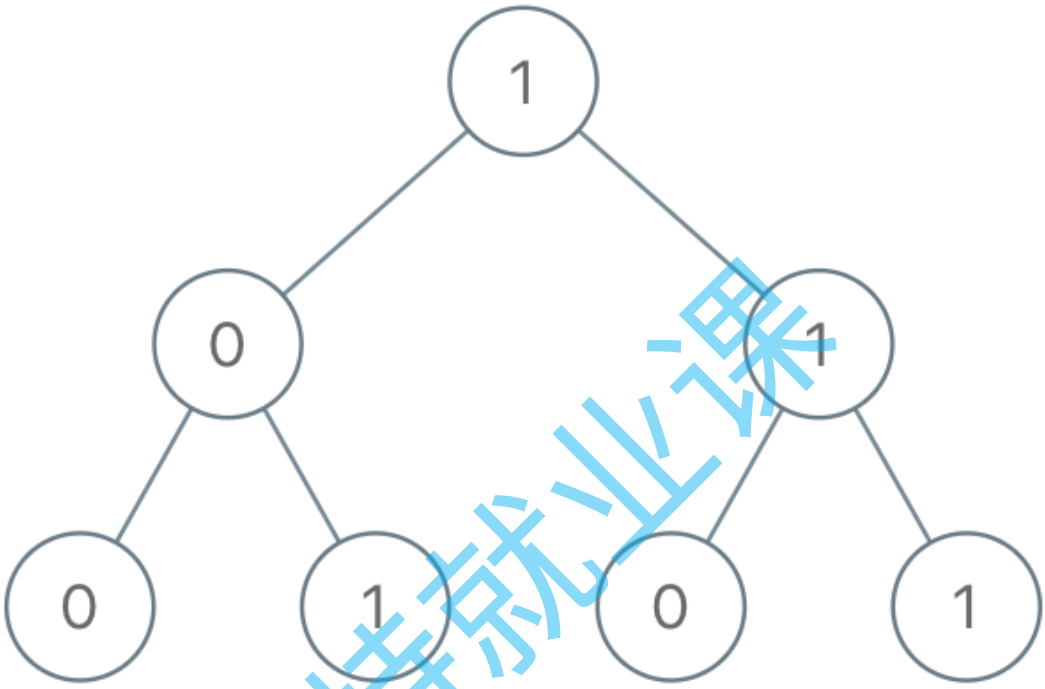
A. -a+b*c/d B. -a+b*cd/e C. -+*abc/de D. abcd/+e*-

二、编程题

1、从根到叶的二进制数之和

给出一棵二叉树，其上每个结点的值都是 `0` 或 `1`。每一条从根到叶的路径都代表一个从最高有效位开始的二进制数。例如，如果路径为 `0 -> 1 -> 1 -> 0 -> 1`，那么它表示二进制数 `01101`，也就是 `13`。对树上的每一片叶子，我们都要找出从根到该叶子的路径所表示的数字。返回这些数字之和。题目数据保证答案是一个**32 位**整数。

OJ链接【`LeetCode` 题号: 1022.从根到叶的二进制数之和】【简单】




```
示例 1：[见上图]
输入：root = [1,0,1,0,1,0,1]
输出：22
解释：(100) + (101) + (110) + (111) = 4 + 5 + 6 + 7 = 22

示例 2：
输入：root = [0]
输出：0

示例 3：
输入：root = [1]
输出：1

示例 4：
输入：root = [1,1]
输出：3
```

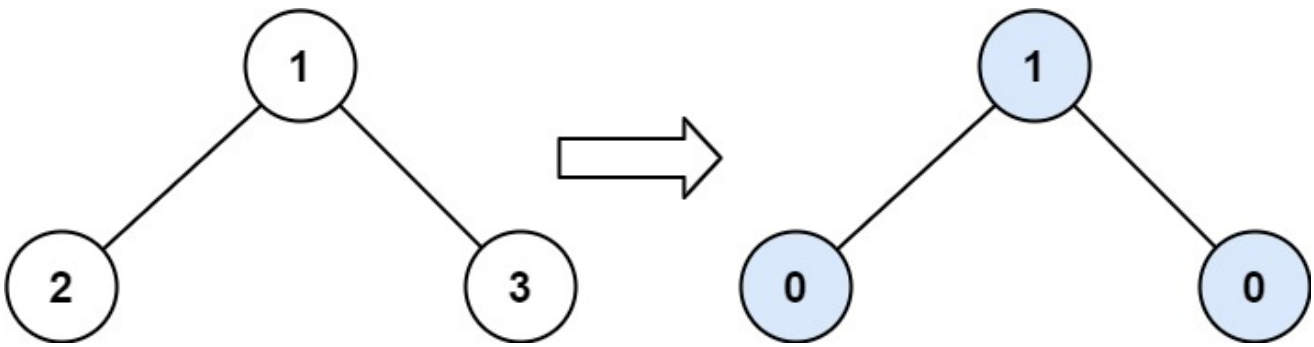
```
int sumRootToLeaf(
{
}
```

```
class Solution {
public:

    int res = 0;

    void dfs(TreeNode* root, int val)
    {

        if(root == nullptr) return ;//空节点

        val = (val << 1) + root->val;//二进制值

        if(root->left == nullptr && root->left == root->right)//叶子节点
        {
            res += val;
            return ;
        }

        dfs(root->left, val);
        dfs(root->right, val);
    }

    int sumRootToLeaf(TreeNode* root) {

        dfs(root, 0);

        return res;
    }
};
```

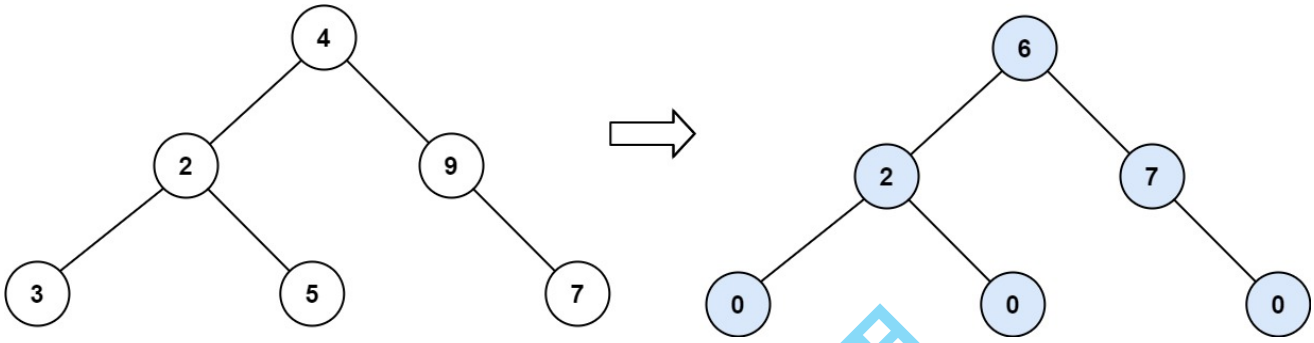
2、二叉树的坡度

给你一个二叉树的根节点 `root` ，计算并返回 **整个树** 的坡度 。一个树的**节点的坡度**定义即为，该节点左子树的节点之和和右子树节点之和的**差的绝对值**。如果没有左子树的话，左子树的节点之和为 0 ；没有右子树的话也是一样。空结点的坡度是 0 。**整个树**的坡度就是其所有节点的坡度之和。

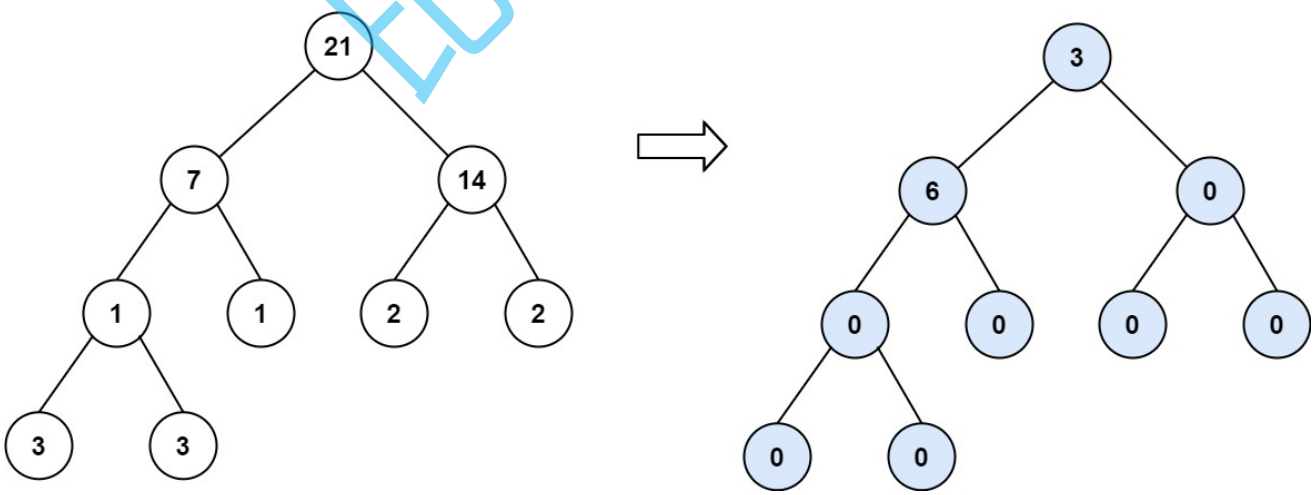
OJ链接【`LeetCode` 题号: 563.二叉树的坡度】【简单】



示例 1:[见上图]
输入：root = [1,2,3]
输出：1
解释：
节点 2 的坡度：|0-0| = 0（没有子节点）
节点 3 的坡度：|0-0| = 0（没有子节点）
节点 1 的坡度：|2-3| = 1（左子树就是左子节点，所以和是2，右子树就是右子节点，所以和是3 ）
坡度总和：0 + 0 + 1 = 1



示例 2:[见上图]
输入：root = [4,2,9,3,5,null,7]
输出：15
解释：
节点 3 的坡度：|0-0| = 0（没有子节点）
节点 5 的坡度：|0-0| = 0（没有子节点）
节点 7 的坡度：|0-0| = 0（没有子节点）
节点 2 的坡度：|3-5| = 2（左子树就是左子节点,所以和是3,右子树就是右子节点,所以和是 5 ）
节点 9 的坡度：|0-7| = 7（没有左子树，所以和是 0 ；右子树正好是右子节点，所以和是 7 ）
节点 4 的坡度：|(3+5+2)-(9+7)| = |10-16| = 6（左子树值为 3、5 和 2 ，和是 10 ；
右子树值为 9 和 7 ，和是 16 ）
坡度总和：0 + 0 + 0 + 2 + 7 + 6 = 15



示例 3:[见上图]
输入：root = [21,7,14,1,1,2,2,3,3]
输出：9

```
int findTilt(struc
{
}
```

```
class Solution {
public:
    int res = 0;
    int dfs(TreeNode* root)
    {
        if(root == nullptr) return 0;
        int leftsum = dfs(root->left);//左子树和
        int rightsum = dfs(root->right);//右子树和
        res += abs(leftsum - rightsum);//坡度
        return leftsum + rightsum + root->val;//当前子树和
    }
    int findTilt(TreeNode* root) {
        dfs(root);
        return res;
    }
};
```

day15

一、选择题

1、在按层序遍历二叉树的算法中，需要借助的数据结构是（）

A. 队列 B. 栈 C. 线性表 D. 有序表

2、一棵二叉树的前序（先序）遍历序列为 ABCDEFG，它的中序遍历序列可能是（）

A. CABDEFG B. ABCDEFG C. DACEFBG D. ADCFEG

3、某二叉树结点的中序序列为 A、B、C、D、E、F、G、H，后序序列为 B、D、C、A、F、G、H、E。该二叉树的层次次序序列为?（）

A. E、G、H、F、A、C、D、B B. E、A、H、C、G、B、D、F

C. E、A、G、H、C、F、B、D D. E、G、A、C、H、D、F、B

4、将一颗有 100 个结点的完全二叉树从根这一层开始，每一层从左到右依次对结点进行编号，根节点编号为 1，则编号为 98 的节点的父节点编号为（）

A. 47 B. 48 C. 49 D. 50

5、设二叉树的先序遍历序列和后序遍历序列正好相反，则该二叉树满足的条件是（）。

A. 空或只有一个结点 B. 高度等于其结点数

C. 任一结点无左孩子 D. 任一结点无右孩子

二、编程题

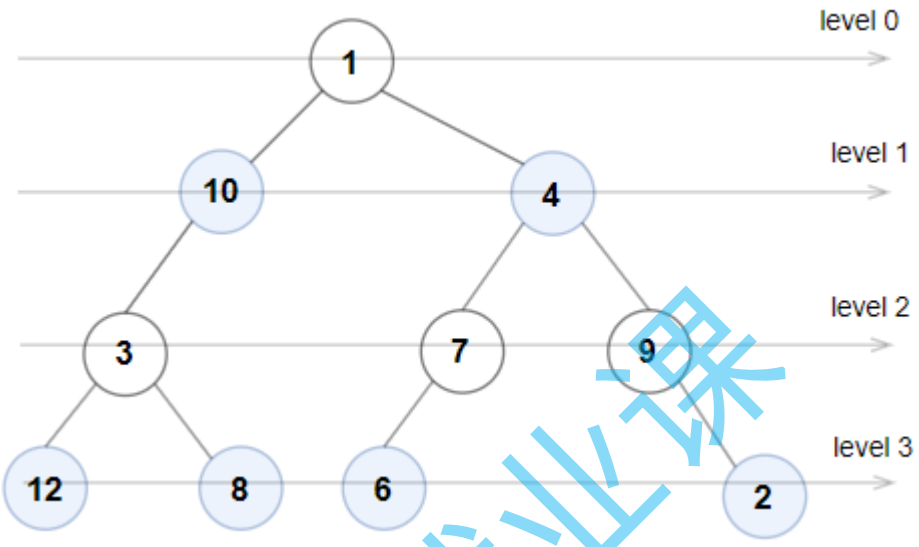
1、奇偶树

如果一棵二叉树满足下述几个条件，则可以称为**奇偶树**：

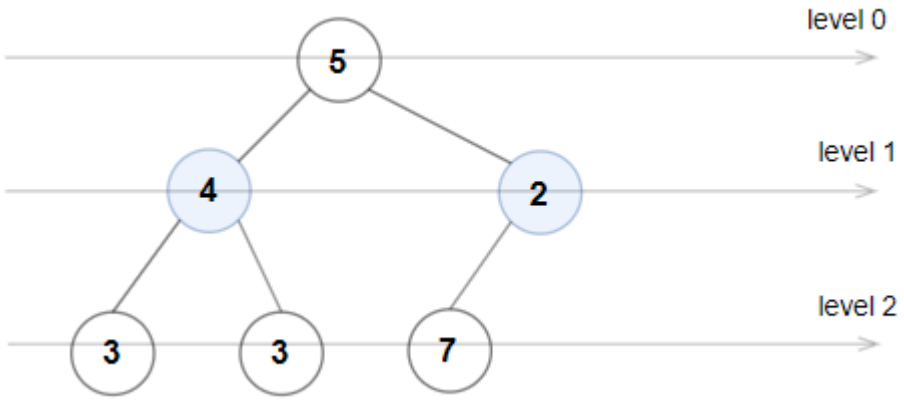
- 二叉树根节点所在层下标为 0 ，根的子节点所在层下标为 1 ，根的孙节点所在层下标为 2 ，依此类推。
- **偶数下标**层上的所有节点的值都是**奇**整数，从左到右按顺序**严格递增**
- **奇数下标**层上的所有节点的值都是**偶**整数，从左到右按顺序**严格递减**

给你二叉树的根节点，如果二叉树为**奇偶树**，则返回`true`，否则返回`false`。

OJ链接【`LeetCode`题号: 1609. 奇偶树】【中等】



示例 1：[见上图]
输入：root = [1,10,4,3,null,7,9,12,8,6,null,null,2]
输出：true
解释：每一层的节点值分别是：
0 层：[1]
1 层：[10,4]
2 层：[3,7,9]
3 层：[12,8,6,2]
由于 0 层和 2 层上的节点值都是奇数且严格递增，而 1 层和 3 层上的节点值都是偶数且严格递减，因此这是一棵奇偶树。



```
class Solution {
public:

    queue <TreeNode*> que;

    bool bfs(int lev)
    {
        if(que.empty()) return true;

        int sz = que.size();
        int Max = 1e6 + 11, Min = 0;
        if(lev % 2 == 1)
            for(int i = 0; i < sz; i++)
            {
                auto node = que.front();
                que.pop();
                if(node->val % 2 == 1 || node->val >= Max) return false;
                Max = node->val;

                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
            }
        else
            for(int i = 0; i < sz; i++)
            {
                auto node = que.front();
                que.pop();
                if(node->val % 2 == 0 || node->val <= Min) return false;
                Min = node->val;

                if(node->left) que.push(node->left);
                if(node->right) que.push(node->right);
            }

        return bfs(lev + 1);

    }

    bool isEvenOddTree(TreeNode* root) {

        que.push(root);

        return bfs(0);

    }
};
```

2、数组的相对排序

给你两个数组 arr1 和 arr2，

- arr2 中的元素各不相同
- arr2 中的每个元素都出现在 arr1 中

对 arr1 中的元素进行排序，使 arr1 中项的相对顺序和 arr2 中的相对顺序相同。未在 arr2 中出现过的元素需要按照升序放在 arr1 的末尾。OJ链接【LeetCode 题号: 1122. 数组的相对排序】【简单】

示例：
输入：arr1 = [2,3,1,3,2,4,6,7,9,2,19], arr2 = [2,1,4,3,9,6]
输出：[2,2,2,1,4,3,3,9,6,7,19]

```cpp
class Solution {
public:
    vector<int> relativeSortArray(vector<int>& arr1, vector<int>& arr2) {
        //桶排序
        int arr[1011] = {0};
        vector<int> res;
        for(auto x : arr1)//统计元素
            arr[x]++;
        for(auto x : arr2)//提取优先元素
            while(arr[x])
            {
                res.push_back(x);
                arr[x]--;
            }
        for(int i = 0; i < 1011; i++)//桶排序
        {
            while(arr[i])
            {
                res.push_back(i);
                arr[i]--;
            }
        }
        return res;

        /* 键值对 + 自写cmp
         * 可以给每个元素增加一个关键值，优先按照关键值排序
         */

    }
};
```


}

day16

一、选择题

1、具有1000个节点的二叉树的最小深度为（）(第一层深度为1)

A. 11 B. 12 C. 9 D. 10

2、已知一棵完全二叉树中共有626个结点，叶结点的个数应为（）

A. 311 B. 312 C. 313 D. 314

3、在一棵二叉树中有30个叶子结点，仅有一个孩子的结点有20个，则该二叉树共有（）个结点

A. 79 B. 76 C. 56 D. 81

4、一棵完全二叉树具有1000个结点，则此完全二叉树有 （）个度为2的结点。

A. 497 B. 498 C. 499 D. 500

5、表达式"X=A+B*(C-D)/E"的后缀表示形式可以是（）

A. XAB+CDE/-*= B. XA+BC-DE/*= C. XABCD-*E/+= D. XABCDE+*/=

二、编程题

1、将句子排序

一个**句子**指的是一个序列的单词用单个空格连接起来，且开头和结尾没有任何空格。每个单词都只包含小写或大写英文字母。我们可以给一个句子添加**从 1 开始的单词位置索引**，并且将句子中所有单词**打乱顺序**。比方说，句子"This is a sentence"可以被打乱顺序得到"sentence4 a3 is2 This1"或者"is2 sentence4 This1 a3"。给你一个**打乱顺序**的句子s，它包含的单词不超过9个，请你重新构造并得到原本顺序的句子。

OJ链接【LeetCode题号: 1859. 将句子排序】【简单】

示例 1：
输入：s = "is2 sentence4 This1 a3"
输出："This is a sentence"
解释：将 s 中的单词按照初始位置排序，得到 "This1 is2 a3 sentence4"，然后删除数字。

示例 2：
输入：s = "
输出："Me
解释：将 s

```
char * so
{

}
```

```
class Solution {
public:
    string sortSentence(string s) {

        string arr[10], res;

        for(int i = 0, j = 0; j < s.size(); j++)
            if(isdigit(s[j]))
            {
                arr[s[j] - '0'] = s.substr(i, j - i);
                i = ++j + 1;
            }
        for(auto str : arr)
            if(!str.empty())
                res += (str + ' ');
        //res.resize(res.size() - 1);
        res.pop_back();
        return res;
    }
};
```

2、最长和谐子序列

和谐数组是指一个数组里元素的最大值和最小值之间的差别**正好是**`1`。现在，给你一个整数数组`nums`，请你在所有可能的子序列中找到最长的和谐子序列的长度。数组的子序列是一个由数组派生出来的序列，它可以通过删除一些元素或不删除元素、且不改变其余元素的顺序而得到。

OJ链接【`LeetCode`题号: 594. 最长和谐子序列】【中等】

```
示例 1：
输入：nums = [1,3,2,2,5,2,3,7]
输出：5
解释：最长的和谐子序列是 [3,2,2,2,3]

示例 2：
输入：nums = [1,2,3,4]
输出：2

示例 3：
输入：nums = [1,1,1,1]
输出：0
```

```
class Solution {
public:

    unordered_map<int, int> mvc;
    int findLHS(vector<int>& nums) {

        for(auto x : nums)
            mvc[x]++;
        int res = 0;

        for(auto [x, y] : mvc)
        {
            if(mvc.count(x + 1) == 0) continue;
            else res = max(res, mvc[x] + mvc[x + 1]);
        }

        return res;
    }
};
```