目录

O(1)!=1,可能是3

1. 组织程序结构

自顶向下

思先路，什么问题，如何解决

想好框架分几个函数实现

想好用几个变量表示（写函数的时候也会增加变量）

uva489 uva12412

这种题一般代码量是需要不少的，常常会有很多冗余的部分，所以在写的过程中就要怎样让接近的功能用一个函数去实现，尽力让代码充分利用，这样去想也许会多一点做它的动力了。 uva220

重复性强的代码，相似的功能，尽量用一个函数实现

if else 尽量用循环

1. c++结构体

struct Point{

int x,y;

Point (int x=0,int y=0):x(x),y(y){}

};

Point operator + (const Point&A,const Point&B) { //构造函数

retuen Point(A.X+B.x,A.y+B.y);

}

ostream& operator <<( //不搞了

Point a,b(1,2);

a.x=3;

struct Point{

T x,y;

Point (T x=0,T y=0):x(x),y(y){}

};

Point<T> operator + (const Point<T>&A,const Point<T>&B) { //构造函数

retuen Point<T>(A.X+B.x,A.y+B.y);

}

Point<int>a;

重载运算符：

struct ant

{

int id;

int p;

int d;

bool operator <(const ant &a) const {return p<a.p;}

};

struct Person p1 = {"zhangsan",20,170};

struct Stu

{

int nNum;

bool bSex;

char szName[20];

char szEmail[100];

//构造函数初始化

Stu()

{

nNum = 0;

bSex = false;

memset(szName,0,sizeof(szName));

memset(szEmail,0,sizeof(szEmail));

}

};

1.结构体指针，malloc分配空间，不会自动赋值 new会。 这里包括指针赋NULL

(测试发现如果不构造函数初始化为零，那么new，malloc都是随机的；但是初始化后new不随机，malloc仍随机）

所以要注意初始化啊

这里的初始化可以是

int a=0;

int b=1;

struct ss \* next,\*be=NULL;

ss(int a=0,struct ss \* next=NULL):a(a),next(next){} 这两种。不管哪一种都符合上述测试

2. 注意何时需要分配内存。 写入数据才需要。而只是指向地址则不需要。故假设p1 写入了数据，p1要new，而p2=p1即可，无需new。

3.大写NULL 注意free

6.int height(node\* root)

{

if(root==NULL)

return 0;

int lh=height(root->next);

int rh=height(root->be);

return (lh>rh)?lh+1:rh+1;

}

struct ListNode {

int val;

ListNode\* next;

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

};

ListNode\* h = new ListNode(0);

或 ListNode h(0);

struct node{

int a;

struct node \*next,\* be;

node(int a=0,struct node \* next=NULL,struct node \* be=NULL):a(a),next(next),be(be){}

};

Node a (1，null,null);

node \* root=new node;

node \* root1=(node\*)malloc(sizeof(node));

int h=height(root); //h=1,因为是new出来的，root不认为是NULL

int h2=height(root1); //出错，因为root是随机的

root1=NULL; int h2=height(root1); h2=0;

7. 由上，再创立节点以及给其next赋NULL，赋值时要小心注意。

注意顺序。

root new了。然后root=NULL。就不要;再root->next=new了。

还有一个错误。比如 node \*p=root；

p=NULL；这样p就和root又无关了。不要用p赋root了

1. 数学

Catalan：

h(n)=h(n-1)\*(4\*n-2)/(n+1); h(n)=C(2n,n)/(n+1) (n=0,1,2,...) h(n)=c(2n,n)-c(2n,n-1)(n=0,1,2,...)

h(0)=1,h(1)=1 h(n)= h(0)\*h(n-1)+h(1)\*h(n-2) + ... + h(n-1)\*h(0) (n>=2)

f(2)=f(3)=1, f(n+1)=f(n)\* (4\*n-6)/n (purple book) 但是这里4\*n-6/n会有除不尽。

可以catalan[i]\*(4\*j-6)/j

组合数学：

C(n,m)=n!/m!\*(n-m)!

A(n,m)=n!/(n-m)!

C(n, k) =C(n-1, k) +C(n-1, k-1)

1. Gdb

文本界面的调试器

在gcc编译时加上-g选项

生成test.exe文件后，执行gdb test.exe

l 显示源代码（10行） l 15（显示15的前面5行+后四行） l -显示当前最先显示行的前十行

help显示帮助，也可以如help list help stack

r运行程序

b设置断点 b main在main函数开始处加断点，用c继续（而不是r） b可以指定行号和函数开头

n执行当前行（s也是） 区别n将函数视为整体，s进入函数内部

u执行至指定行 如u 9，或u main

在提示符下直接按ENTER，会执行上一条命令（如s）

p显示变量，表达式值

i lo（info local）显示所有局部变量，还有 i b显示断点，i disp显示display

disp程序停止时显示变量和表达式

cl 类似b，清除断点，cl清除所有断点

delete disp（d disp）删除disp，（dis disp） en disp暂时禁止或回恢复 b也可以类似操作

q退出

其他cond，ig。。。另行学习

bt 查看调用栈

#0为当前栈帧#1为“上一个”

使用up 选择上一个栈帧

gdb显示SIGSEGV 说明段错误

1. 堆栈

编译后产生的可执行文件包含内容，与操作系统有关。UNIX\Linux ELF，DOS COFF，Win PE。

段（segmentation）指二进制文件内的区域，size（命令行）程序可以得到可执行文件中各个段的大小。

text正文段储存指令，data数据段储存已经初始化的全局变量，bss段储存未赋值的全局变量 。dec总大小。hex以16进制表示。

可执行文件运行时调用栈，所在段才被创建。所在堆栈段也有大小，不能越界访问。若总是增加栈帧，就会越界。称为栈溢出。也有可能与局部变量太大有关。

栈空间大小与系统有关。

//由于栈上的空间是自动分配自动回收的，所以栈上的数据的生存周期只是在函数的运行过程中，运行后就释放掉，不可//以再访问。而堆上的数据只要程序员不释放空间，就一直可以访问到，不过缺点是一旦忘记释放会造成内存泄露。

1.申请后系统的响应

栈 ： 只要栈的剩余空间大于所申请的空间，系统将为程序提供内存，否则将报异常提示栈溢出。

堆：首先应该知道操作系统有一个记录空闲内存地址的链表，当系统受到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该结点从空闲结点链表中删除，并将该结点的空间分配给程序，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样，代码中的delete语句才能正确的释放本内存空间。另外，由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。也就是说堆会在申请后还要做一些后续的工作这就会引出申请效率的问题

2.申请大小的限制

栈： 在Windows下，栈是向低地址扩展的数据结构，是一块连续的内存的区域。这句话的意思是栈顶的地址和栈的最大容量是系统预先规定好的，在Windows下，栈的大小是2M（也有的说是1M，总之是一个编译时就确定的常数），如果申请的空间超过栈的剩余空间时，将提示overflow。因此，能从栈获得的空间较小。

堆：堆是向高地址扩展的数据结构，是不连续的内存区域。这是由于系统是用链表来存储空闲内存地址的，自然是不连续的，而链表的遍历方向是由低地址向高地址。堆的大小受限于计算机系统中有效的虚拟内存。由此可见，堆获得的空间比较灵活，也比较大。

3.申请方式

stack:

由系统自动分配。 例如，声明在函数中一个局部变量 int b; 系统自动在栈中为b开辟空间

heap:

需要程序员自己申请，并指明大小，在c中malloc函数

如p1 = (char \*)malloc(10);

在C++中用new运算符

如p2 = (char \*)malloc(10);

但是注意p1、p2本身是在栈中的。

于是：new malloc都在堆上；递归时注意不要溢出

栈溢出 1.递归调用太多 2.局部变量太大

不要在函数内开很大的静态数组 （栈空间有限）

1. 内存

首先，1个内存单元的长度是8bits，占一个字节；

32位是指地址长度，四个字节；

地址计算机是按照字节存储，一个字节一个地址编号；

32位，int 4个字节，x[0][0] x[0][1]地址编号差4

内存地址指向内存单元（可能1个或更多），32位系统每次读取的内存单元都是4个字节的偶数倍，可能会根据需要忽略多余的字节。

地址编号是一个一个字节编的；

mips一个字指4个字节

一个字节两个16进制数

1. 数据大小

int 32 范围[-2^31 , 2^31 -1] 即 [-2147483648，2147483647], 1e9

%d %u

unsigned int 0～4294967295

int -2147483648～2147483647 2e9

unsigned long 0～4294967295

long -2147483648～2147483647

long long的最大值：9223372036854775807 1e18 I64d (is i not L)

long long的最小值：-9223372036854775808

unsigned long long的最大值：1844674407370955161 输入输出%llu %lld大概均可 不确定时用cin，cout

\_\_int64的最大值：9223372036854775807

\_\_int64的最小值：-9223372036854775808

unsigned \_\_int64的最大值：18446744073709551615

1、编译器不同导致使用64位的申明方式不同; 最好%I64d

2、long long / unsigned long long 一般是Linux下申明方式、如：G++

3、\_\_int64 /unsigned \_\_int64一般是Windows下使用64位的申明方式，如:VS

4、在赋值时需要注意加上ll进行显式赋值;

5、当进行64位与 32位的混合运算时，32位整数会被隐式转换成64位整数。

6、输出printf("");,long long使用%lld输出，\_\_int64使用%I64d，无符号使用u替代d即可。

1. 链表

struct node\* creat()

{

struct node \*head,\*New,\*tail;

head=NULL;

int n=0;

printf("输入整数，-1为结束标志\n");

scanf("%d",&n);

while(n!=-1)

{

New=(struct node\*)malloc(sizeof(struct node));

New->number=n;

New->next=NULL;

if(head==NULL)

head=New;

else

tail->next=New;

tail=New;

scanf("%d",&n);

}

fflush(stdin);

return head;

}

1. 输入输出
2. 注意加括号 cout<<(c>d?c+1:d+1)
3. while(scanf(%d,&n)==1&&n)

while(scanf("%d%s%s",d,s,s)==3&&d!=-1)

scanf(" (%d,%d)", &u, &v);

Scanf中添加

空白字符: 空白字符会使scanf()函数在读操作中略去输入中的一个或多个空白字符。

非空白字符: 一个非空白字符会使scanf()函数在读入时剔除掉与这个非空白字符相同的字符。

学到了

scanf("%c%d%d", &s, &x, &y);

getchar(); //如果用scanf，注意处理\n uva1589

for(int i = 1; i <= 8; i++)

for(int j = 1; j <= 8; j++)

{

char ch='\n';

while(ch=='\r'||ch=='\n')

{

ch=getchar();

}

{

board[i][j] =ch;

if(board[i][j] == 'W') num\_w++;

else if(board[i][j] == 'B') num\_b++;

}

}

1. Cin 遇空格停

long double 用 Lf

Double lf

Float f

1. 浮点数一种精度输出方法

//cout << "Case #" << ++kase << ": "<<fixed<<setprecision(5)<< sqrt(ans) << endl;

1. Return 时注意转换类型
2. 1<<2可能更快

&x【0】【0】+1和&x【0】+1完全不同

char x【4】【5】

scanf("%s",x[0]+1); 表示从x【0】【1】开始读x【0】，即，若原为abc，读kkkk，得x【0】akkkk 注意，若没有初始化，x【0】【0】还是’\0’

scanf("%s",&x[0]+1); 表示读x【1】

1. %p16进制输出，不是地址专用
2. 输入多次换行的用cin

四舍五入保留小数

三位

double ans = 0.0005;

int d=(int)(ans\*1000+0.5);

ans=d/1000.0;//必须是.0

1. 一个个读入字符时

for(int i=0;i<n;i++)

cin>>str[i];

1. 存字母

储存字母

for(int i = 0; i < len1; i++)

a[s1[i]-'A']++; 0的ASCII码是48

1. 注意转义

输出\ 用\\

1. 操作符

X^0=x; x^x=0 ^=1 k是偶数，则使k=k+1；若k是奇数，则使k=[k-1](https://www.baidu.com/s?wd=k-1&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)

· 与运算&： 对应位均为1时为1，其它为0。//n&1判断n是不是奇数，是返回1

· 或运算|： 对应位均为0时为0，其它为1。

· 异或运算^： 对应位不相同时为1，相同时为0.

· 按位取反~： 每一位取反

· 右移>>: 将二进制进行右移，低位丢掉，高位补零。

· 左移<<： 将二进制进行左移，低位补零，高位丢掉

1. 一些输入函数

printf（“%c”，‘\0’）输出空格

scanf读字符串（%d也一样）时xxx‘\n’，\n不读，仍在缓存区，gets会读\n并转化为‘\0’存入数组

char \*fgets(char \*buf, int bufsize, FILE \*stream);，读size-1，最后一个赋‘\0’；会读‘\n’并存入数组，但是存入后会结束输入，后一个赋‘\0’

getchar同scanf

如果开头就是一个‘\n’，scanf%d%s会忽略（删掉），%c会读；getchar会读；gets会读，getline也会读；getline类似于fgets

cin会忽略第一个\n，但是会把结束的\n存在缓冲区

gcc编译器扩展定义了getline()函数.

用于读取一行[字符](https://baike.baidu.com/item/%E5%AD%97%E7%AC%A6" \t "https://baike.baidu.com/item/getline%E5%87%BD%E6%95%B0/_blank)直到换行符,包括换行符, 适合1 2 3 这样带空格的

一般用法

while(getline(cin, s)) {

stringstream ss(s);

while(ss >> a[n]) ;

输入1 2 3 4 5\n，写入的会是12345，空格被略去

1. 二进制转十进制

二进制转化为10进制

int v=0;

while(len--)

v=v\*2+readchar()-'0';

return v;

1. c++大整数

<https://www.jianshu.com/p/80b2c16ac4e6>

最好学一下java或者py的大整数

1. c++特性
2. 引用

void swap2（int &a，int&b）会按引用传递，改变实参值

String长度 .size .length (c中strlen不包括’\0’，而且不用于string)

getline(cin,xx,xx) 对于string类 string line; getline（cin，line）;

<sstream> 字符串流（可以直接读取） stringstream ss(line); while(ss>>x)

1. 模板

注意也可以用于函数

template<typename T>

struct Point{

T x，y;

Point(T x=0,T y=0):x(x),y(y);{}

}

template<typename T>

Point<T> operator + （const Point<T> A,const Point，<T> B）{

return Point<T>(A.x+B.x,A.y+B.y);

}

template<typename T>

ostream& operator << (ostream &out,const Point<T> &p)

{ out <<"("<<p.x<<","<<p.y<<")";

return out;

}

就可以用各种类型的Point Point<int> a(1,2),b(3,4)

1. 常用函数

1.<cctype>

2.memset 在<cstring>里

3.fabs 在<cmath>

4.string和char

输出string用cout或.c\_str（）；

赋值string s时，char c[20],strcpy（c，s.str()）

1.<ctype.h> isalpha() isdigit(是否为数字）（也可以用-‘0’来实现） isprint .tolower toupper

2.#include <algorithm>

abs

sort(start,end,排序方法)

bool compare(int a,int b)

{

return a>b; 从大到小排序

}

struct node

{

int a;

int b;

double c;

}

bool cmp(node x,node y)

{

if(x.a!=y.a) return x.a<y.a;

if(x.b!=y.b) return x.b>y.b;

return x.c>y.c;

}

3.memset <string.h>

4.sprintf(),<cstdio> int sprintf(char \*buffer, const char \*format, [argument]...)

（1）buffer：是char类型的指针，指向写入的字符串指针；

（2）format：格式化字符串，即在程序中想要的格式；

（3）argument：可选参数，可以为任意类型的数据；

char\* who = "I";

char\* whom = "CSDN";

sprintf(s, "%s love %s.", who, whom); //产生："I love CSDN. " 这字符串写到s中

sprintf(s, "%10.3f", 3.1415626); //产生：" 3.142"

连接字符串与,将数据化为字符串 返回写入的字符数；出错-1

5.strcpy(s1,s2)将s2复制到s1

6.strcmp,相等返回0,第一个参数可以是其一个指针，指向起始位置

7.如果一个函数要求char\*参数，可以使用c\_str()方法：

string s = "Hello World!";

printf("%s", s.c\_str()); //输出 "Hello World!"

7.puts输出字符串并换行

8.next\_permutation按字典序生成下一个排列 1.所有排列都生成后返回false；2.注意是下一个，所以最好先sort，再输出，再n\_p;

while(next\_permutation(a,a+n)) 生成1-n

do{}while（n\_p）；（先输出开始的）

9.用pair时typedef方便。

typedef pair<int,int>p; queue<p>que;

10.sprintf(buf, "%05d%05d", abcde, fghij);

if(strlen(buf) > 10) break;

判断位数

11.%05d，位数大于5位则输出实际位数

12.判断类型的函数命名 is\_pirme

13.sqrt 返回double， （math.h）

floor（）向下取整，

floor（sqrt（n）+0.5），利用四舍五入避免浮点误差（如本是整数值变成了xxx.99999）

14.position=max\_element(a,a+n)-a;

printf("%d\n",\*max\_element(a,a+n));

15.fill

int a[1000];

fill(a,a+1000,1);

16.memset 对字节（byte）而不是位（bit）

17.inline 内联

18substr s.substr(i,len) 从s【i】开始，长度为len //不加就到end（包含'\0'）

19.string 直接==比较

20.upper\_bound(a,a+n,k) 返回指向满足ai>k的ai的最小指针；lower ai>=k的最小指针

先排序

21.sscanf

char buf[512] = ;

　　sscanf("123456 ", "%s", buf);

　　printf("%s/n", buf);

　　结果为：123456

　　2. 取指定长度的字符串。如在下例中，取最大长度为4字节的字符串。

　　sscanf("123456 ", "%4s", buf);

　　printf("%s/n", buf);

　　结果为：1234

22.strchr

原型： char \*strchr(const char \*s,char c);

#include<string.h>

查找字符串s中首次出现字符c的位置,返回首次出现c的位置的指针，如果s中不存在c则返回NULL。

23.const double PI = acos(-1.0);

24.assert(pres) pres错误时终止程序；

25.关于引用 如double& a=x；则改变a实际就是改变x；x必须也为double

26.log(x) e log10(x) logn^m(x)=1/mlogn(x)

27.exp(x)=e^x

28.%%输出%号

29.int a=atoi(s) 转化字符串为整型

30.int N=ceil((double)(3\*b-a)/(3\*a-b)); ceil 向上取整

31.fabs针对浮点数，abs针对整数

32.reverse(a,a+n) 翻转 algorithm

33.bool xxx(const A& a, const A& b) {return a.xxx < b.xxx;} 升序

34.q[i] = question{a, b, i};

35.new calloc 构造？

36.map无法sort，可以转移到vector

for(map<int,double>::iterator curr=P.begin();curr!=P.end();++curr)

vec.push\_back(make\_pair(curr->first,curr->second));

sort(vec.begin(), vec.end(), cmp);

37set自动排序升序 map按键自动升序

1. assert（exp） exp==false return
2. Stl

[https://www.jianshu.com/writer#/notebooks/35440518/notes/45263079](https://www.jianshu.com/writer" \l "/notebooks/35440518/notes/45263079)

1. 优化

1.uva 12099

j+k<=sum[i] 只考虑ww2<ww1+30,ww3<ww2+30 （每本书宽度不超过30，则若大于可以放到下一层）

先排序，这样放书高度不变

2.求位数可以m/10，cnt++

也可以 cnt=log10（n）

3.求分数相加，可以先算出来lcm分母，再加分子\*lcm/分母； 最后可以再上下除以gcd

4.有的求f（a-b）可以用f（a）-f（b） uva1640

5.划分区间，设模板 uva1640

6.写出公式后尝试化简 uva10213 https://blog.csdn.net/QWsin/article/details/53635397

1. 公式先化简； 如标准差公式
2. 英文

<https://www.jianshu.com/p/0aa490c7d4f3>

1. 时间复杂度

Oorder，表示数量级

无循环或循环和n无关，O（1）；输入输出，定义赋值，O（1）

并列循环 O（max（f1，f2。。。））

多层循环 \*

多项式时间（p）：n2，n3，logn，nlogn

np：2^n，n！

平均，最好，最坏时间复杂度；求和P（i）\*T（i）

空间复杂度只计算临时变量所占空间，以避免重复；

T2n2 >> On2

递归时间复杂度分析：写出递推式 如T（n）=2T（n/2）+O（n）=....（mergesort）

1.计算机一秒10^9次运算

假设时间要求一秒，10^6充足，1e7勉强，1e8很悬（限于循环很简单）

2.sort O（nlogn） ，二分查找O（logn）

3.O(M\*N\*8)=O(M\*N)

4.dp T为状态总数\*每个状态的决策

1. 常见错误

1.rte : 数组越界，小了

2.cpe 命名有歧义

3.有时int太小了用long long

4.看上去很简单的题，很有可能数据范围会比较大int不够

5.string 用cin

6.pe 用空格隔开时

cout << ans[0];

for(int i = 1; i < ans.size(); i++) cout << " " << ans[i];

7.sizeof的用法 返回的是数组大小 s1【100】 返回100；而不是内容大小;用strlen

string 用str.size（）

8.for循环先判断

9.dp时注意边界条件或者转移方程有的达不到要求，先判断

如： if(k>1)

d【i】【k】=d【i-1】【k-1】

10。推出公式后注意验证边界和大值；如0,1,100000；

11.讨论时可能会被题目迷惑 不要漏掉不常见的情况 如uva11346 s>ab；s\_>0 fabs(x)<1e-6

12.\\勿忘转义 if s[i]='\\' || '/'

13.LL b=1<<N 会有问题 （1默认int） 改为LL b=1LL<<n

14.测试一下 1 和 大数 ； uva1213 n=k=1

观察输出 再做修改 有的时候公式会需要稍作修改

uva11440

15.long long tot=( long long)(2\*a+1)\*(2\*b+1)-1; //正确

cout<<tot<<endl;

long long tott=( long long)((2\*a+1)\*(2\*b+1)-1);//这样加了括号先溢出了

cout<<tott<<endl;

16.(ans-1+mod)%mod 防止ans-1=-1，输出-1的情况

17.数据类型不匹配 尤其是LL 和int； 函数的返回输入尤其要注意

18.超出n的范围 根据公式求的cnt ，要min（n-i，cnt） uva1363

19设了maxn，但是数组忘了maxn+5

20.scanf==4 坑了

20.

有的编译器，计算5%3=2；有的算出来5%3=-1

为了避免两种的差别

我们统一用：ans=（ans+mod）%mod

就可以把这个都化成正的最小的那个啦

21.max 中类型应该一致，否则先转换

22.vector函数调用要加&

23.一般定义了NULL 但没得nul！！

24

结构体链表注意细节！！

头指针和新的节点都是malloc的，不是直接struct\* （或者先struct\*之后再=malloc也可以）

h1=h2=malloc (错误！

25.任何累加累减累乘操作都很危险

26.局部全局变量位置

27.scanf，cin是遇到 空格 回车 和 tab键 结束输入，而gets只能遇到回车结束输入 (字符串和string

1. 字符数组如果不初始化，最后一位不是'\0'
2. 小知识
3. Acsii

0--48 A --65 a--97

1. 常用思路
2. 最小值最大 二分