power of 2

开个数组模拟乘法运算

```
# include <stdio.h>
void mul(int*a ,int b,int *len){
    //a*b a为大数 b为【0，9】len为a的有效数字部分（即a是几位数）
    int temp;
    int carry=0;
    int i=0;
    //a逆序表示，a[0]个位 a[1]十位...
    //比如12345在a数组里表示为5 3 2 1 0 0 0 0....
    for(i=0;i<*len;++i){
        temp=a[i]*b+carry;
        a[i]=temp%10;
        carry=temp/10;
    }
    while(carry!=0){
        a[i]=carry%10;
        carry=carry/10;
        i++;
    }
//    printf("%d",i);
    *len=i;
}

int main(){
    int n;
    scanf("%d",&n);
    int arr[100002]={1};
    for(int i=1;i<100002;++i) {
        arr[i]=0;
    }
    int len=1;
    int *p=&len;

    while (n){
        mul(arr,2,p);
        n--;
    }

    //逆序输出
    for(int i=len-1;i>=0;--i){
        printf("%d",arr[i]);
    }

};
```

人工智能翻译

1. 每3位进行一次翻译
   比如3位数字写为xyz
   3位的第一位，单独拿出来，翻译成x 百，后两位一起翻译成几十几，十位数是0，翻译成几。

2. 上述翻译完成后，在后面加上million 或者thousand等位数后缀

```c
#include <stdio.h>
#include <malloc.h>
char *my_strcat(char *a, char *b) {
    //拼接字符串。返回新的地址c
    //不用自带的strcat是因为，strcat会直接在第一个字符串后面添加b的内容。返回地址还是a的首地址
    char *c = (char *) malloc(20*sizeof(char)); //局部变量，用malloc申请内存
    char *tempc = c; //把首地址存下来
    while (*a != '\0') {
        *c++ = *a++;
    }
    while ((*c++ = *b++) != '\0') {
        ;
    }
    //注意，此时指针c已经指向拼接之后的字符串的结尾'\0' !
    return tempc;//返回值是局部malloc申请的指针变量，需在函数调用结束后free之
}


char* onesdigit2en(int num){
    //个位数翻译
    switch (num) {
        case 0:return my_strcat("","");
        case 1:return my_strcat("One","");
        case 2:return my_strcat("Two","");
        case 3:return my_strcat("Three","");
        case 4:return my_strcat("Four","");
        case 5:return my_strcat("Five","");
        case 6:return my_strcat("Six","");
        case 7:return my_strcat("Seven","");
        case 8:return my_strcat("Eight","");
        case 9:return my_strcat("Nine","");
        default:return my_strcat("","");//不写defalut直接RE
    }
}


char* tensdigit2en(int num1,int num2){
    //2位数翻译,[10,19]需要特别翻译
    char *temp="";
    switch (num1) {
        case 1:
            switch (num2) {
                case 0:return my_strcat("Ten","");
                case 1:return my_strcat("Eleven","");
                case 2:return my_strcat("Twelve","");
                case 3:return my_strcat("Thirteen","");
                case 4:return my_strcat("Fourteen","");
                case 5:return my_strcat("Fifteen","");
                case 6:return my_strcat("Sixteen","");
                case 7:return my_strcat("Seventeen","");
                case 8:return my_strcat("Eighteen","");
                case 9:return my_strcat("Nineteen","");
                default:return my_strcat("","");
            }
```

```
            break;
            //Twenty 在常量区，指针永远返回同一个地址，不要对常量区字符串修改，最好开个新地址进行深拷贝
        case 2:temp="Twenty ";break;
        case 3:temp="Thirty ";break;
        case 4:temp="Forty ";break;
        case 5:temp="Fifty ";break;
        case 6:temp="Sixty ";break;
        case 7:temp="Seventy ";break;
        case 8:temp="Eighty ";break;
        case 9:temp="Ninty ";break;
        case 0:temp="";break;
        default:temp="";break;
    }
    char* ans= my_strcat(temp, onesdigit2en(num2));//用自定义strcat，每次都开辟新地址
    return ans;
}


void  right_just(char *s,char filler,int strlen,char* out){
    //strlen 数字的位数
    for(int i=0;i<10-strlen;++i){
        out[i]=filler;
    }
    int index=0;
    for(int i=10-strlen;i<10;++i){
        out[i]=s[index++];
    }
    out[10]='\0';
}
int main(){

    char num[11]={'\0'};//剩余 自动'\0'
    //多开一位存'\0'
    //最大数字2147483647 10位数
    int len=0;
    scanf("%s",&num);
    while (num[len++]!='\0'){;}
    len--;

    char just_num[11]="";//用*just_num会RE

    right_just(num,'0',len,just_num);//右对齐，如12345会补充成0000012345 由于英语是从高位翻译，顺序读数组。就会从高位开始翻译

    int b1=just_num[0]-'0';//十亿位的数字
    if(b1!=0){
        printf("%s Billion ",onesdigit2en(b1));
    }
    //每次处理3位
    for(int i=1;i<10;i=i+3){
        int n1=just_num[i]-'0';
        int n2=just_num[i+1]-'0';
        int n3=just_num[i+2]-'0';
        if(n1|n2|n3!=0){
            if(n1!=0){
                printf("%s Hundred ",onesdigit2en(n1));
```

```
            }
            char* temp=tensdigit2en(n2,n3);

            printf("%s ",temp);
            switch (i) {
                case 1:printf("%s ","Million");
                    break;//不加break,case匹配成功后，继续向下执行
                case 4:printf("%s ","Thousand");
                    break;
                default:;
            }
        }

    }

}

```

蚂蚁大战

直接模拟战争过程

定义结构体 战场warplace：每个单元可以存2个蚂蚁的战斗力，分别存在up和down里。如果只有一个蚂蚁，则活的蚂蚁移动到up上。

在负向蚂蚁看来，正向蚂蚁是一直过来的。所以只要移动正向蚂蚁就行。

一个时间单位move一个距离。

move之后开始打仗。判断序列的每个站位，是否有打仗。有就更新站位和存货蚂蚁战力。

move n次之后，就算是最左边蚂蚁，也肯定走到最右边去了。循环结束。

```
#include <stdio.h>
struct warplace{
    //战场，可以容纳2个蚂蚁的战力
    int up;
    int down;
};
int can_move(struct warplace* s, int pos, int n){
    //判断是否当前蚂蚁是否能继续移动
    int flag=0;
    if(pos<n-1){
        for(int i=pos+1;i<n;++i){
            if(s[i].up <= 0){
                flag=1;
                break;
            }
        }
    }
    return flag;
}
```

```c
void move(struct warplace* s,int n ){
    //更新蚂蚁位置，单个蚂蚁移动一步,可以认为负数蚂蚁不动，只有正蚂蚁移动。都动会交叉错位
    //n是蚂蚁数量
    for(int i=0;i<n;++i){
        if(can_move(s, i,n)){
            if(s[i].up > 0){
                s[i+1].down=s[i].up;//移动到新位置
                s[i].up=0;//原位置清零
            }
        }
    }
}
void war(struct warplace*s,int n){
    //开始交战，死亡的蚂蚁战力记录为0,并更新 up down数据和次序
    for(int i=0;i<n;++i){
        int abs_a= s[i].up > 0 ? s[i].up : -1 * s[i].up;
        int abs_b= s[i].down > 0 ? s[i].down : -1 * s[i].down;
        if(abs_a>abs_b){
            s[i].down=0;
        }
        if(abs_a==abs_b){
            s[i].up=0;
            s[i].down=0;
        }
        if(abs_a<abs_b){
            s[i].up=s[i].down;
            s[i].down=0;
        }
    }
}

int main(){
    //input
    int n;//蚂蚁数量
    scanf("%d",&n);
    int count=0;
    struct warplace antsplace[10000];
    while (count<n){
        antsplace[count].down=0;
        scanf("%d",&antsplace[count++].up);
    }
    //交战
    int time=0;
    while(time++<n){
        move(antsplace,n);
        war(antsplace,n);
    }
    //output
    for(int i=0;i<n;i++){
        if(antsplace[i].up != 0){
            printf("%d ",antsplace[i].up);
        }
    }
}
```

diverse

hash思想

每个字符hash到一张表，并检测碰撞。hash必须满足象不会跳跃，且单调。这里选择f(x)=x;

得到hash表之后，看看表的非空值是否连续。

复杂度O(N)

```
#include <stdio.h>

int judge_diverse(char *s,int len){

    //hash,mod26，看有没有 连续的，没有碰撞的哈希

    //先产生哈希表
    int hashmap[26]={0};
    int flag=1;
    for(int i=0;i<len;++i){
        int index=s[i]-'a';
        if(hashmap[index]==0){
            hashmap[index]=1;
        } else{
            flag=0;
        }
    }
    //判断是否连续
    int i=0;
    int j=25;
    while (hashmap[i++]!=1){;}
    while (hashmap[j--]!=1){;}
    i--;j++;
    for(int k=i;k<=j;++k){
        if(hashmap[k]==0){
            flag=0;
        }
    }
    return flag;
}


int main(){
    int n;
    scanf("%d\n",&n);
    int count=0;
    int ans[200];
    int index=0;

    while (count<n){
        char s[51]={'\0'};
        int len=0;
        scanf("%s",s);
        while (s[len++]!='\0'){;}
        len--;
        ans[index++]=judge_diverse(s,len);
        count++;

    }

```

```
    //output
//    freopen("test1.txt", "w", stdout);
    for(int i=0;i<index-1;++i){
        if(ans[i]){
            printf("%s\n","Yes");
        } else{
            printf("%s\n","No");
        }
    }
    //最后一个输出，不要换行
    if(ans[index-1]){
        printf("%s","Yes");
    } else{
        printf("%s","No");
    }
}
```

排序后检查连续

看是否是连续递增 不重复数组：

对字符串进行排序和查重，排序算法最低复杂度O(nlgn)

然后检测是否连续，用法1代码检测连续的算法，复杂度O（n）；

递归

递归，如果长度为i的数组 diverse, 长度为i+1的数组 diverse就是，比Min(原数组)少1，或者max（原数组）大1. 递归复杂度目测$O(N^2)$