作业 2.2

题目描述 1

设计算法, 删除$r[n]$中重复元素

算法描述

hash法判断元素重复, 不重复的元素移动到数组开头,移动次数与不重复元素数量相同

代码

```
func main() {
        var r []int
        var N int

        fmt.Scanln(&N)
        r = make([]int, N)
        for i, _ := range r {
                fmt.Scanf("%d", &r[i])
        }

        var hash map[int]bool = make(map[int]bool)

        count := 0
        for _, x := range r {
                _, ok := hash[x]
                if ok {
                        continue
                }
                hash[x] = true
                r[count] = x
                count++
        }

        var ans []int = r[0:count]
        fmt.Println(ans)
}
```

运行结果

```
6
1 3 3 2 4 3
[1 3 2 4]
```

```
6
1 1 1 1 1 1
[1]
```

```
6
1 2 3 4 5 6
[1 2 3 4 5 6]
```

题目描述 2

设表 $A=a_1,a_2,a_3,\ldots a_n$, 将A拆成B、C两个表, 令B中元素大于等于0, C中元素小于0

算法描述

反复同时从左右两端遍历, 找到左边第一个小于0的数和右边第一个不小于0的数, 交换位置, 反复如此操作直到左右指示变量相遇. 算法遍历了一遍数组, 时间复杂度为$O(n)$

代码

```
var N int
var A []int

func divideArray() ([]int, []int) {
        var l, r int = 0, N - 1
        for l < r {
                for A[l] >= 0 {
                        l++
                }
                for A[r] < 0 {
                        r--
                }
                if l >= r {
                        break
                }
                swap(&A[l], &A[r])
        }
        B := A[0:l]
        C := A[l:N]
        return B, C
}
func main() {
        fmt.Scanln(&N)
        A = make([]int, N)
        for i, _ := range A {
                fmt.Scanf("%d", &A[i])
        }
        B, C := divideArray()
```

```
        fmt.Println("B:", B)
        fmt.Println("C:", C)
}

func swap(l *int, r *int) {
        t := *l
        *l = *r
        *r = t
}
```

运行结果

```
7
0 3 2 4 -7 9 -5
B: [0 3 2 4 9]
C: [-7 -5]
```

```
7
1 2 3 -4 5 6 -7
B: [1 2 3 6 5]
C: [-4 -7]
```

题目描述 3

判断两个单词是否变位词, 如果两个单词字母组成相同但位置不同,称为变位词

算法描述

分别遍历两个单词, 第一个单词的每个字母为相应计数变量 $+1$, 第二个单词为相应计数变量 -1, 如果两个单词相同则直接输出 No, 时间复杂度为$O(n)$

代码

```
var s1, s2 string
var count []int

func main() {

        fmt.Scanln(&s1)
    fmt.Scanln(&s2)
    if s1 == s2 {
        fmt.Println("No")
                        return
    }
```

```
        count := make([]int, 26)
        for _, x := range s1 {
                count[int(x-'a')]++
        }
        for _, x := range s2 {
                count[int(x-'a')]--
        }
        for _, x := range count {
                if x != 0 {
                        fmt.Println("No")
                        return
                }
        }
        fmt.Println("Yes")
}
```

运行结果

```
asdfggfdsa
dsfgaasdfg
Yes
```

```
qwertyuiop
asdfsghjklk
No
```

```
aaa
aaa
No
```

题目描述 4

一堆桃子,每天吃掉一半加上m个,给出n,m,d表示经过n天,每天额外吃m个,最后剩d个,求最初桃子数量

算法描述

递归式为

$$F(n,m,d)=\begin{cases}1 & n=1\\ F(n-1,m,(d+m)*2) & n>1\end{cases}$$

代码

```
func prevDay(n, m, d int) int {
        if n == 1 {
                return d
        }
        return prevDay(n-1, m, (d+m)*2)
}

func main() {
        var n, m, d int
        fmt.Scanln(&n, &m, &d)
        fmt.Println("Num:", prevDay(n, m, d))
}
```

运行结果

```
1 2 3
Num: 3
```

```
4 3 7
Num: 98
```

题目描述 5

计算递推数列

$$\begin{cases} a[i] = 1 \\ a[2i] = a[i] + 1 \\ a[2i+1] = a[i] + a[i+1] \end{cases}$$

算法描述

递归式为

$$\begin{cases} 1 & n = 1 \\ a[n/2] + a[n/2+1] & n \neq 1 \wedge n \equiv 1(mod2) \\ a[n/2] + 1 & n \neq 1 \wedge n \equiv 0(mod2) \end{cases}$$

计算时, 保存了$n = k$时$a[k]$的值, 防止重复计算$a[k]$

代码

```
var dp []int

func calu(N int) int {
        if dp[N] > 0 {
```

```
                return dp[N]
        }
        if N == 1 {
                dp[N] = 1
                return dp[N]
        }
        if N%2 == 1 {
                dp[N] = calu(N/2) + calu(N/2+1)
                return dp[N]
        } else {
                dp[N] = calu(N/2) + 1
                return dp[N]
        }
}

func main() {
        var N int
        fmt.Scanln(&N)
        dp = make([]int, N+1)
        fmt.Println("a[n] =", calu(N))
}
```

运行结果

```
99
a[n] = 37
```

```
2345354
a[n] = 11133
```

题目描述 6

求两个由链表存储的整数之和,不使用其他的数据结构,且指针由高位指向低位

算法描述

先遍历两个链表,对应位相加形成一个新的ANS链表,在ANS链表中递归执行进位操作

代码

```
struct Node {
    int Data;
    Node *Next;
    void Add(Node *p);
    int IncSet();
    Node() = default;
```

```
    Node(int n) {
        Data = n;
        Next = NULL;
    }
};

void Node::Add(Node *p) {
    p->Next = this->Next;
    this->Next = p;
}
int Node::IncSet() {
    if(this->Next == NULL) {
        int c = this->Data / 10;
        this->Data %= 10;
        return c;
    }
    this->Data += this->Next->IncSet();
    int c = this->Data / 10;
    this->Data %= 10;
    return c;
}

int main() {
    Node *A = new Node(), *B = new Node();
    string sa, sb;
    int la, lb;

    cin >> sa >> sb;
    la = sa.size();
    lb = sb.size();

    Node *p = A;
    for(int i=0;i<la;++i) {
        p->Add(new Node(sa[i] - '0'));
        p = p->Next;
    }
    p = B;
    for(int i=0;i<lb;++i) {
        p->Add(new Node(sb[i] - '0'));
        p = p->Next;
    }

    if(la < lb) {
        Node *t = A;
        A = B;
        B = t;
        la = sb.size();
        lb = sa.size();
    }

    Node *Ans = new Node();
    p = Ans;
    Node *q = A;
    for(int i=0;i<la - lb;++i) {
```

```
            p->Add(new Node(q->Next->Data));
            p = p->Next;
            q = q->Next;
        }
        Node *w = B;
        for(int i=0;i<lb;++i) {
            //printf("%d %d\n", q->Next->Data, w->Next->Data);
            p->Add(new Node(q->Next->Data + w->Next->Data));
            p = p->Next;
            q = q->Next;
            w = w->Next;
        }
        int c = Ans->Next->IncSet();
        if(c) {
            Ans->Add(new Node(c));
        }

        p = Ans;
        if(p->Next != NULL) {
            printf("%d", p->Next->Data);
        }
        p = p->Next;
        while(p->Next != NULL) {
            printf(" -> %d", p->Next->Data);
            p = p->Next;
        }
        printf("\n");
        return 0;
    }
```

运行结果

```
1234
567
1 -> 8 -> 0 -> 1
```

```
567
1234
1 -> 8 -> 0 -> 1
```

```
9999
1000
1 -> 0 -> 9 -> 9 -> 9
```

题目描述 7

给定一个链表,将其从后往前每k个元素翻转,如开头不足k个则不执行操作

算法描述

递归遍历,返回过程中找到长为k的部分链表,翻转

代码

```
struct Node{
    int data;
    Node *Next;
    void Ins(int n);
    void Del();
    Node(int n) {
        data = n;
        Next = NULL;
    }
};

void Node::Ins(int n) {
    Node *it = new Node(n);
    it->Next = this->Next;
    this->Next = it;
}
void Node::Del() {
    Node *it = this->Next;
    this->Next = this->Next->Next;
    delete it;
}

int N, K;
Node *head;

pair<Node*, Node*> reserve(Node *oper, int n) {
    if(n == K) {
        return make_pair(oper->Next, oper);
    }
    pair<Node*, Node*> rt = reserve(oper->Next, n + 1);
    oper->Next->Next = oper;
    return rt;
}
int Reserve(Node *oper) {
    if(oper->Next == NULL) {
        return K - 1;
    }
    int t = Reserve(oper->Next);
    if(t) {
        return
            t - 1;
    } else {
        pair<Node*, Node*> rt;
        rt = reserve(oper->Next, 1);
```

```
            oper->Next->Next = rt.first;
            oper->Next = rt.second;
            return K - 1;
        }
        return -1;
    }

    int main() {
        head = new Node(-1);
        scanf("%d %d", &N, &K);

        Node *p = head;
        for(int i=0;i<N;++i) {
            int t;
            scanf("%d", &t);
            p->Ins(t);
            p = p->Next;
        }
        int left = Reserve(head->Next);
        if(!left) {
            pair<Node*, Node*> rt;
            rt = reserve(head->Next, 1);
            head->Next->Next = rt.first;
            head->Next = rt.second;
        }
        p = head->Next;
        for(int i=0;i<N;++i) {
            if(p->Next == NULL)
                break;
            printf("%d -> ", p->data);
            p = p->Next;
        }
        printf("%d\n", p->data);
        return 0;
    }
```

运行结果

```
6 3
1 2 3 4 5 6
3 -> 2 -> 1 -> 6 -> 5 -> 4
```

```
7 3
1 2 3 4 5 6 7
1 -> 4 -> 3 -> 2 -> 7 -> 6 -> 5
```