# 上机6解析

## A丢失的学号

| 难度 | 考点 |
| --- | --- |
| 1 | 字符串 |

### 题目分析

本题利用二维字符数组（可以看做字符串数组）与 `strcmp()` 函数即可完成编写，同时需要注意不存在 `78` 的书院代码。

这里顺便复习一下 `strcmp()` 函数的用法：对于字符串 $a, b$，自左向右逐个字符相比（按 ASCII 值大小相比较），直到出现不同的字符时，若 $a$ 的该字符比 $b$ 的该字符大，则称 $a$ 的字典序大于 $b$。

若 $a < b$，则 `strcmp(a, b) < 0`；若 $a > b$，则 `strcmp(a, b) > 0`；若 $a < b$，则 `strcmp(a, b) = 0`。因此，可以利用这个函数来判断两个字符串是否相等。

### 示例代码

```
#include <stdio.h>
#include <string.h>

char s[][10] = {"ShiE", "FengRu", "ShiJia", "ShouE", "ZhiZhen", "",
"ZhiXing"};
char t[10];

int main(){
    int i;

    scanf("%s", t);
    printf("193");
    for(i = 0; i < 7; i++)
        if(strcmp(s[i], t) == 0)
            printf("%d", i + 73);   // 将 [0, 7) 映射到代码上
    printf("299\n");

    return 0;
}
```

## B排排队

| 难度 | 考点 |
| --- | --- |
| 2 | 指针 |

## 题目解析

简单的二维数组+数据交换

## 示例代码

```
#include <stdio.h>

int main(){
    int m,n,q[35][35]={0},i,j,x,y,k,l,p;
    scanf("%d %d",&m,&n);
    for (i=1;i<=m;i++)
        for (j=1;j<=n;j++)
            scanf("%d",&q[i][j]);
    while (~scanf("%d %d %d %d",&x,&y,&k,&l)){
        p=q[x][y];
        q[x][y]=q[x+k][y+l];
        q[x+k][y+l]=p;
    }
    for (i=1;i<=m;i++){
        for (j=1;j<=n;j++)
            printf("%d ",q[i][j]);
        printf("\n");
        }
    return 0;
}
```

---

# c准备生日礼物

| 难度 | 考点 |
| --- | --- |
| 3 | 二维数组，循环 |

## 题目分析

从当前的日期往后循环，计数，直到月数和日数与出生的相同；

注意闰年的区别，可以用一个二维整型数组储存非闰年和闰年每一个月的天数；

这个题的一个坑就是对2月29日的处理，因为这一天的生日并不是每年都有的。

## 示例代码1

这个代码巧妙地避开了2月29日的特殊情况，依靠一天天去+1来求解

```
#include <stdio.h>
#include <stdlib.h>
int mon_day[2][13] = {{0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
                      {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}}; //
非闰年和闰年每一个月的天数
int isLeap(int);                                                            //
判断是不是闰年
int main()
{
    int year, month, day;
    int cu_year, cu_month, cu_day;
    int count = 0, leap;

    while (scanf("%d%d%d%d%d%d", &year, &month, &day, &cu_year, &cu_month,
&cu_day) != EOF)
    {
        count = 0;                                          //计数
        while (!(cu_month == month && cu_day == day)) //直到生日那天
        {
            count++;
            cu_day++;
            leap = isLeap(cu_year);                 //注意不能放在while循环外面，因为
年份可能会改变
            if (cu_day > mon_day[leap][cu_month]) //日期超过本月最大天数重置
            {
                cu_month++;
                cu_day = 1;
            }
            if (cu_month > 12) //月份超过12重置
            {
                cu_year++;
                cu_month = 1;
            }
        }

        printf("%d\n", count);
    }
    return 0;
}
int isLeap(int year)
{
    return ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0);
}
```

## 示例代码2

这个代码封装了更多的函数来处理下一个2月29日、每月的天数。通过以年、月为单位求和取代一天天去加来求日期，更加高效。

```
#include <stdio.h>
int main()
{
    int year, month, day;
    int cu_year, cu_month, cu_day, birth_year;
    int i, j, count, c1, c2, leapdate;
    while (scanf("%d%d%d", &year, &month, &day) != EOF)
    {
        if (month == 2 && day == 29)
            leapdate = 1;
        else
            leapdate = 0;
        scanf("%d%d%d", &cu_year, &cu_month, &cu_day);
        if (leapdate)
            birth_year = getNextLeapYear(cu_year);
        else
            birth_year = cu_year;
        c1 = getDays(birth_year, month, day);    //今年生日的天数
        c2 = getDays(cu_year, cu_month, cu_day); //今日的天数
        if (c2 > c1)
        {
            birth_year++;
            if (leapdate)
                birth_year = getNextLeapYear(birth_year);
            c1 = getDays(birth_year, month, day); //今年生日的天数
        }
        printf("%d\n", c1 - c2);
    }
    return 0;
}
int isLeapYear(int y)
{
    return ((y % 4 == 0) && (y % 100 != 0)) || (y % 400 == 0);
}
int getMonthDays(int y, int m)
{
    switch (m)
    {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        return 31;
```

```
    case 4:
    case 6:
    case 9:
    case 11:
        return 30;
    case 2:
        if (isLeapYear(y))
            return 29;
        else
            return 28;
    default:
        return 0;
    }
}

int getNextLeapYear(int y)
{
    while (!isLeapYear(y))
    {
        y++;
    }
    return y;
}
//计算1999年1月1日到y、m、d多少天
int getDays(int y, int m, int d)
{
    int i, count = 0;
    for (i = 1999; i < y; i++)
    {
        count += 365;
        if (isLeapYear(i))
            count++;
    }
    for (i = 1; i < m; i++)
    {
        count += getMonthDays(y, i);
    }
    count += d;
    return count;
}
```

## 示例代码3

这段代码封装了两个函数分别处理，整月整年地统计今天到生日的间隔。效率更高。

```
#include <stdio.h>
#include <math.h>
#include <string.h>
int year1, mon1, day1, year2, mon2, day2, n1, n2, ans;
```

```c
int month[13] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

int judge(int y)
{
    if (y % 400 == 0)
        return 1;
    if (y % 4 == 0 && y % 100 != 0)
        return 1;
    return 0;
}

void calc1() //生日不是2月29
{
    int i, j, ans = 0;
    if (mon2 * 100 + day2 > mon1 * 100 + day1) //今年生日过了
    {
        ans = month[mon2] - day2;
        for (i = mon2 + 1; i <= 12; i++)
            ans += month[i];
        if (mon2 <= 2 && judge(year2) == 1)
            ans++;

        for (i = 1; i < mon1; i++)
            ans += month[i];
        if (mon1 > 2 && judge(year2 + 1) == 1)
            ans++;
        ans += day1;
        printf("%d\n", ans);
    }
    else
    {
        if (mon1 == mon2)
            ans = day1 - day2;
        else
        {
            ans = month[mon2] - day2;
            for (i = mon2 + 1; i < mon1; i++)
                ans += month[i];
            if (mon2 <= 2 && mon1 > 2)
                ans += judge(year2);
            ans += day1;
        }
        printf("%d\n", ans);
    }
}

void calc2() //2/29生日
{
    int i, j, ans = 0;
```

```
    if (judge(year2) == 1) //如果当年是闰年
    {
        if (mon2 == 2)
            ans = 29 - day2;
        else if (mon2 == 1)
            ans = 60 - day2; //生日没过
        else
        {
            ans = month[mon2] - day2;
            for (i = mon2 + 1; i <= 12; i++)
                ans += month[i];
            year2++;
            while (judge(year2) != 1)
            {
                ans += 365;
                year2++;
            }
            ans += 60;
        }
    }
    else
    {
        ans = month[mon2] - day2;
        for (i = mon2 + 1; i <= 12; i++)
            ans += month[i];
        year2++;
        while (judge(year2) != 1)
        {
            ans += 365;
            year2++;
        }
        ans += 60;
    }
    printf("%d\n", ans);
}

int main()
{
    int n, m;

    while (scanf("%d%d%d%d%d%d", &year1, &mon1, &day1, &year2, &mon2, &day2)
!= EOF)
    {
        if (mon2 == mon1 && day2 == day1)
        {
            printf("0\n");
            continue;
        }
        if (mon1 == 2 && day1 == 29)
```

```
            calc2();
        else
            calc1();
    }
    return 0;
}
```

# D查找子串

| 难度 | 考点 |
| --- | --- |
| 3 | 字符串 |

## 题目分析

如参考代码1所示，从字符串a的首字母开始，从a中取出子串b长度sub_len的字符串存入新的数组temp中，可以利用strcmp函数比较temp和b，若相同，则指针向后移动sub_len位，继续进行取出子串的操作；若不相同，指针只能向后移动一位，再进行取出子串的操作；直到当前指针指向的字符串长度小于b的长度时，程序结束。

若temp和b相同时，输出当前指针str的值（即地址）与a的值之差，即可得到子串首字母在a中的相对位置。

参考代码2中使用了strstr，其实string.h中的strstr能直接完成查找子串的功能（不过能自己写出程序对学习指针更有帮助哦）。

## 参考代码1

```
#include <stdio.h>
#include <string.h>

void pos_strstr(char *str, char *substr);

char a[20],b[20];

int main()
{
  gets(a);
  gets(b);
  pos_strstr(a,b);
  return 0;
}

void pos_strstr(char *str, char *substr)
{
    char temp[20];
    int str_len=strlen(str),sub_len=strlen(substr),comp;
```

```
    if(str_len>=sub_len)
    {
      strncpy(temp,str,sub_len);  //将str指向的sub_len个字符储存到temp中
      temp[sub_len]='\0';
      comp=strcmp(temp,substr);
      if(comp==0)
      {
          printf("%d-%d\n",str-a,str-a+sub_len-1);
          str+=sub_len;
          pos_strstr(str,substr);
      }
      else
      {
          str++;
          pos_strstr(str,substr);
      }
    }
}
```

## 参考代码2

```
#include <stdio.h>
int main()
{
    char a[20],b[20];
    char *p;
    int len;
    gets(a);
    gets(b);
    len = strlen(b);
    p = a;
    while((p = strstr(p, b))!=NULL){
        printf("%d-%d\n",p-a, p-a+len-1);
        p += len;
    }
    return 0;
}
```

---

# E Palindrome

| 难度 | 考点 |
| --- | --- |
| 3 | 字符串 |

## 题目分析

PPT 71 页提供了一个回文串判断的写法供参考，这里不再赘述。

显然，使用该方法判断一个串 $s$ 是否为回文串所需时间开销，和串长度近似成正比。如果此时再枚举子串的起止位置，则额外需要两重循环，因此整个算法的时间开销可以近似认为与串长 $|s|$ 的三次方成正比，记为 $\mathcal{O}(|s|^3)$。请注意，这里对大 O 记号的表述与真正的定义有所出入，仅是为了方便说明而引入该记号。

在本题中，采用 $\mathcal{O}(|s|^3)$ 的写法即可顺利通过。但如果 $|s|$ 大一些，变成了 $1 \leq |s| \leq 1,000$ 时，这种写法有可能会超时。在此我们介绍一种 $\mathcal{O}(|s|^2)$ 的写法：我们不妨钦定原串中的一个对称轴，并考虑从该位置开始向两边逐渐扩展，判断扩展之后的新串是否为回文串。

显然，若 $s$ 是回文串，那么在 $s$ 两端加入一个字符 $c$ 而成的串 $csc$ 也是一个回文串；若 $s$ 不是回文串，那么在 $s$ 两端加入一个字符 $c$ 而成的串 $csc$ 也不可能是一个回文串。利用这个性质，我们只需要检查每次扩展时两端的字符是否相等，即可判断该扩展出来的串是不是回文串。这样，我们只需要枚举原串中的每个对称轴，并对对称轴扩展找到该对称轴上所有回文串即可。不难证明，这样的做法时间复杂度是 $\mathcal{O}(|s|^2)$ 的。

如果 $|s|$ 再大一些，变成了 $1 \leq |s| \leq 1,000,000$ 时，还有办法可以解决吗？事实上，求最长回文子串有一种 $\mathcal{O}(|s|)$ 的做法，可以用与读入字符串差不多的时间找到最长的回文子串。当然，这种做法属于课外内容，感兴趣的同学可以自行搜索 Manacher 算法。

## 示例代码

```c
#include <stdio.h>
#include <string.h>

#define N (100 + 5)
#define max(a, b) ((a) > (b) ? (a) : (b))

// 返回 s[] 的下标范围 [st, ed) 是否为回文串
int IsPalindrome(char s[], int st, int ed){
    while(st < ed){
        if(s[st] != s[ed - 1]) return 0;
        st++; ed--;
    }
    return 1;
}

char s[N];

int main(){
    int q, n, ans;
    int i, j;

    scanf("%d", &q);
    while(q--){
        scanf("%s", s);
        n = strlen(s);
```

```
        ans = 0;
        for(i = 0; i <= n; i++)
            for(j = i; j <= n; j++){
                if(IsPalindrome(s, i, j) == 1)
                    ans = max(ans, j - i);
            }
        printf("%d\n", ans);
    }

    return 0;
}
```

## F 水水の超越方程

| 难度 | 考点 |
| --- | --- |
| 3 | 数据类型、位运算 |

### 题目解析

通过分析我们不难发现$f(x)$的单调区间是$[\frac{1}{e},+\infty]$，也就是在题目所给的区间内单调。那么对于方程$f(x)=0$我们可以使用二分法来找它的根。众所周知，单调函数$f(x)$在区间$[l,r]$上有根的充要条件是$f(l)\cdot f(r)\leq 0$，而在本题数据不会使得根在区间端点上，所以条件可以简化成$f(l)\cdot f(r)<0$. 通过判断$f(\frac{1}{e})\cdot f(10^6)$是否小于零便可知方程是否有解。

有了这个性质，我们可以每一次取区间中点$\frac{l+r}{2}$作为试探点，根据$f(\frac{l+r}{2})$的正负，来判断根是在区间$[l,\frac{l+r}{2}]$还是$[\frac{l+r}{2},r]$中，然后更新根所在的区间。这样每次可以将根所在的区间缩小二分之一的长度，直到区间长度小于$1e6$的时候，我们认为此时区间中所有点都是方程**精确到小数点后六位**的解。

### 示例代码

```
#include <stdio.h>
#include <math.h>
#define eps 1e-6
#define f(x) (a * x * log(x) - b * exp(-pow(x - 1/e, 4)) + c)
double a, b, c, e;
int main()
{
    double l,r,x;
    e = 2.71828182845;
    while (~scanf("%lf %lf %lf", &a, &b, &c))
    {
        l = 1/e, r = 1e6, x;
        if (f(l) * f(r) > 0)
        {
            printf("No solution.\n");
```

```
            continue;
        }
        while (r - l > eps)
        {
            x = (l + r) / 2;
            if (f(l) * f(x) > 0)
                l = x;
            else
                r = x;
        }
        printf("%.7f\n", l);
    }
    return 0;
}
```

---

# G轴对称图形

| 难度 | 考点 |
| --- | --- |
| 5 | 二维数组快排 |

## 题目分析

第一步，将所有读入的点按y升（或降）序进行排序，当y相同的时候按x的升（或降）序进行排序，这里需要注意快排比较函数的写法，因为是对二维数组进行排序，输入的参数本质是整型指针。

第二步，依次比较每一处y相同点的第一个和最后一个，第二个和倒数第二个点……的x坐标平均值，一旦出现不同的平均值，说明不是轴对称图形。

第三步，每组对应的点纵坐标应该相同，如果输入的点是（0，0）（0，1）（1，1）（1，0）时，排序后的顺序应为（0，0）（1，0）（0，1）（1，1），可以看到，点1和点2/点3和点4的在同一水平线上，点1纵坐标+点2纵坐标=点3纵坐标+点4纵坐标，所以是轴对称图形。具体写法详见标程。

## 示例程序

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

int m[500005][2] = {0};
int n;

int cmp(const void *a, const void *b)
{
    int *m = (int *)a;
    int *n = (int *)b;
    if (*(m + 1) > *(n + 1))
    {
```

```
        return 1;
    }
    else if (*(m + 1) < *(n + 1))
    {
        return -1;
    }
    else
    {
        return *m - *n;
    }
}

int main()
{
    int i, j, k, haveANs = 0;
    double mid = -10086.0, midTmp;
    scanf("%d", &n);
    for (i = 0; i < n; ++i)
    {
        scanf("%d %d", &m[i][0], &m[i][1]);
    }

    qsort(m, n, sizeof(m[0]), cmp);

    for (i = 0; i < n;)
    {
        // printf("aa%d\n", i);
        j = i;
        k = i;
        while (m[j][1] == m[i][1] && j <= n)
        {
            j++;
        }
        i = j;
        j--;
        if (haveANs == 0)
        {
            mid = (double)m[k][0] + (double)m[j][0];
            mid /= 2;
            haveANs=1;
        }
        while (k <= j)
        {
            // printf("aa%f %f\n",mid,);
            midTmp = ((double)m[k][0] + (double)m[j][0]) / 2;
            if (midTmp != mid)
            {
                printf("Not a symmetric figure.");
                return 0;
```

```
            }
            k++;
            j--;
        }
    }

    printf("x = %0.1f", mid);
    return 0;
}
```

---

## H 膜膜膜

| 难度 | 考点 |
| --- | --- |
| 2 | 质数 |

## 题目解析

一句话总结题意就是：输出大于素数$n$的第$m$个素数。改编自课件c8-指针2中的“例7-14”。

数据范围较大，同时查询次数较多，因此需要存素数表来进行查询。

## 示例代码

```
#include<stdio.h>
#include<stdlib.h>
#define MaxN 90007
int prime[10005]={2,3,5,7,11}; //素数表初始化
int c=5;

int isPrime(int n)//判断是否为素数
{
    int i;
    for (i=0;i<c;i++)
        if (n%prime[i]==0) return 0;
    return 1;
}

void buildPrime()//建立素数表
{
    int n=11,step=2;
    while (n<MaxN)
    {
        n+=step; //每6个连续的数中只需判断6*i+1和6*i+5这两个数即可，对于本题，直接n++也不会超时
        if (isPrime(n)) prime[c++]=n;
        step=6-step;
```

```
    }
}

int comp_int(const int *p1,const int *p2)
{
    return (*p1-*p2);
}

int main()
{
    int n,m;
    buildPrime();
    while (~scanf("%d%d",&n,&m))
    {
        int *p=bsearch(&n,prime,c,sizeof(int),comp_int);//bsearch函数见课件c8的57页
        printf("%d\n",*(p+m));
    }
    return 0;
}
```

---

# I Serein的两数之和

| 难度 | 考点 |
| --- | --- |
| 3 | 数组 |

## 题目分析

首先使用`qsort`函数将无序数组变为有序，然后问题变成在有序数组中找出两个数，使它们的和为target。

使用双指针，一个指针指向值较小的元素，一个指针指向值较大的元素。指向较小元素的指针从头向尾遍历，指向较大元素的指针从尾向头遍历。

- 如果两个指针指向元素的和 sum == target，那么得到要求的结果；
- 如果 sum > target，移动较大的元素，使 sum 变小一些；
- 如果 sum < target，移动较小的元素，使 sum 变大一些。

## 示例代码

```
#include<stdio.h>
#include<stdlib.h>
int nums[100005];
int cmp(const void *a,const void *b)
{
    return *(int *)a-*(int *)b;
```

```
}
void f(int i,int j,int tag)
{
    int sum=nums[i]+nums[j];
    if(sum==tag)
    printf("%d %d\n",nums[i],nums[j]);
    else if(sum>tag)
    {
        j--;
        f(i,j,tag);
    }
    else
    {
        i++;
        f(i,j,tag);
    }
}
int main()
{
    int i,n,tag;
    while(scanf("%d %d",&n,&tag)>0)
    {
        for(i=0;i<n;i++)
            scanf("%d",&nums[i]);
        qsort(nums,n,sizeof(int),cmp);
        f(0,n-1,tag);
    }
    return 0;
}
```

# J“压缩的矩阵”乘法

## 题目分析

本题涉及的知识点主要包含进制转换、矩阵乘法（二维数组的定义和使用）。

这里稍微说明一下，如果要使用指针指向二维数组，需定义为(*A)[“行”的长度]。这是因为[]的优先级比*高，若不加括号，则*A[len]等同于*(A[len])，它是一个大小为len的数组，只不过里面存放的数据类型是某种指针；而(*A)[len]才是一个指针，它指向了一个大小为len的数组。

建议大家掌握这种写法，以便将二维数组的相关操作封装成函数。不过：①实在难以理解的话也可以不用指针，将所有操作装入main函数即可。②除了示例代码演示的方法，二维数组的解引用还有很多方式，感兴趣的同学可以再做了解。

## 示例代码

```
#include <stdio.h>
```

```c
#define LEN 7

void parser(int (*M)[LEN], char *str) { //将输入的字符串转换为矩阵
    int M_index, str_index = 0;
    while (str[str_index]) {
        for (M_index = 0; M_index < LEN; M_index++) {
            M[str_index][LEN - 1 - M_index] = str[str_index] % 2;
            str[str_index] /= 2;
        }
        str_index++;
    }
}

void Mat_Mul(int (*A)[LEN], int (*B)[LEN], int (*result)[LEN]) { //矩阵乘法
    int i, j, k;
    for (i = 0; i < LEN; i++) {
        for (j = 0; j < LEN; j++) {
            result[i][j] = 0;
            for (k = 0; k < LEN; k++) {
                //result[i][j] = (result[i][j] % 1000003 + A[i][k] * B[k][j] %
1000003) % 1000003;
                result[i][j] += A[i][k] * B[k][j];
                result[i][j] %= 1000003;
            }
        }
    }
}

int main() {
    int A1[LEN][LEN], A2[LEN][LEN], B[LEN][LEN];
    int (*A)[LEN] = A1, (*result)[LEN] = A2, (*tmp)[LEN];
    char str[10];
    int i, j;
    gets(str);
    str[LEN] = '\0';
    parser(A, str);
    while (gets(str) != NULL) {
        str[LEN] = '\0';
        parser(B, str);
        Mat_Mul(A, B, result);
        tmp = A;
        A = result;
        result = tmp;
    }
    for (i = 0; i < LEN; i++) {
        for (j = 0; j < LEN - 1; j++) {
            printf("%d ", A[i][j]);
        }
```

```
        printf("%d\n", A[i][LEN-1]);
    }
    return 0;
}
```

---

## κbuaaycm

| 难度 | 考点 |
| --- | --- |
| 4 | 模拟 |

## 题目分析

没有思维难度的模拟题。所谓模拟，就是题目说什么，你就做什么。字符串处理是大家最容易遇到也是最容易翻车的模拟题，还请保证一定的熟练度与思维缜密度，注意输入输出函数的理解，测试数据的构造以及库函数的正确使用。

本题的坑点包括但不限于：

1. 换行符，这个已在HINT中给出，以前上机也有遇见，注意`gets()`会将`\r`留在字符串尾。同学们本地测试的时候是无法直接测试`\r`的，这时考虑用条件语句判断，从而获得稳定性。（OJ上的测试数据不保证是以`\r\n`结尾还是以`\n`结尾）
2. 是否正确提取出5、6位数字，注意过短、过长、以及所谓的前导0；
3. 是否满足`3min`内的`10`条这一条件，并注意输出顺序；
4. 对`buaaycm`的判断（前后带有除换行符外的空字符，大小写不匹配等均不可）；
5. 发言为仅有换行的字符串。

## 参考代码

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define isnum(x) ((x) <= '9' && (x) >= '0' ? 1 : 0) // 判断字符是否为数字
char str_queue[105][105]; // 使用了队列来保存字符串，当然也可不使用
char id_queue[105][10];
int head = -1, tail = -1;
char str[105];
int time_queue[105];
char laji[100];
char id[10];
char banlist[5][10] = {"23333", "233333", "66666", "666666", "114514"};
//判断是否是禁词之一
int judge_ban(char x[])
{
    int i;
    for(i = 0; i <= 4; i++)
```

```
    {
        if(strcmp(x, banlist[i]) == 0)
            return 0;
    }
    return 1;
}
//判断是否含合法车牌
int judge(char s[])
{
    int i, len = strlen(s), cnt = 0;
    memset(id, 0, sizeof(id));
    for(i = 0; i < len; i++)
    {
        if(isnum(s[i])) // 在数字串中判断
        {
            if(cnt >= 6) // 此时该数字串已经不合法了
            {
                cnt = 0;
                memset(id, 0, sizeof(id));
                while(i < len - 1 && isnum(s[i+1])) i++; // 进入下一个数字串
            }
            else
                id[cnt++] = s[i];
        }
        else
        {
            if((cnt == 5 || cnt == 6) && judge_ban(id))
                return 1;
            cnt = 0;
            memset(id, 0, sizeof(id));
        }
    }
    return ((cnt == 5 || cnt == 6) && judge_ban(id));
}
void deal(int t)
{
    int i;
    printf("yaiyai!\n");
    if(tail == head || (tail >= 0 && t - time_queue[tail] > 180))
        printf("buaamyc\n");
    else
    {
        for(i = tail; i > head; i--)
        {
            if(t - time_queue[i] <= 180)
                printf("%ds:[%s] %s\n", t - time_queue[i], id_queue[i],
str_queue[i]);
        }
    }
```

```
}
int main()
{
    int h, m, s;
    while(~scanf("%d:%d:%d", &h, &m, &s)) // 可用此方法直接按整型读入时间，便于计算
    {
        int i = 0;
        char c;
        while(getchar()!='\n'); // 或使用gets(laji); 去除scanf留在缓冲区的换行符。
        while((c=getchar())!='\n' && c != '\r' && c != EOF)
        {
            str[i++] = c;
        }
        str[i] = '\0';
        /*或改为
        gets(str);
        int len = strlen(str);
        if(str[len-1] == '\r') str[len-1] = 0, len--;
        这两种方法均同时适用于\r\n 和 \n 的情况
        */
        int t = (h * 60 + m) * 60 + s; // 直接转化成秒进行计算。
        if(strcmp(str, "buaaycm") == 0)
            deal(t);
        else
        {
            if(judge(str) == 1)
            {
                time_queue[++tail] = t;
                strcpy(str_queue[tail], str);
                strcpy(id_queue[tail], id);
                if(tail - head > 10)
                    head++; // 只保留最新的10条
            }
        }
    }
    return 0;
}
```

PS. 本题目是小亚美机器人的一个子功能，且各种实际情况已经经过了简化（比如时间可能隔天，重复的车牌应该去重等等。。）（没办法，只能用C语言就是这么麻烦）。欢迎大家在QQ找小亚美玩。

---

# Lshorn1的ddl

| 难度 | 考点 |
| --- | --- |
| 4 | 队列 |

# 题目分析

为了让总时间最小，应使用 $k$ 次"自动AC机"，并且每次都对花费时间最大的任务使用。但如果每次都遍历 $w$ 找最大值并修改会超时。 我们希望高效找到最大值。解题的关键在于发现本题隐含的单调性。先被操作的任务一定不会比后被操作的任务花费小，假设两个任务的花费分别为 $a, b$ 且 $a \geq b$，则分别对其进行一次操作后，花费变为 $\lfloor \frac{a}{2} \rfloor, \lfloor \frac{b}{2} \rfloor$ ($\lfloor x \rfloor$表示对 $x$ 向下取整)，则 $\lfloor \frac{a}{2} \rfloor \geq \lfloor \frac{b}{2} \rfloor$。我们可以使用一个队列 $q$ 来存储每次新产生的任务，并将原序列 $w$ 也看作一个队列。最大值一定是两个队列的首元素之一。每次操作将最大值 $max$ 出队，将 $\lfloor \frac{max}{2} \rfloor$ 入队 $q$，这两个队列均单调不增。$k$ 次操作后 $q$ 和 $w$ 的所有剩余元素的总和就是答案，注意答案需要使用 `long long` 存储。

# 示例代码

```
#include<stdio.h>

#define M 500055

int n,k;
int w[M],que[M],fr,bk = -1,p; // fr,bk分别表示队首和队尾的位置,且均指向实际存在的元素,因此bk需初始化为-1
long long res;

int main()
{
    int i;
    scanf("%d%d",&n,&k);
    for(i = 0;i < n;i++)
        scanf("%d",&w[i]);
    for(i = 0;i < k;i++)
    {
        int tmp;
        if(w[p] >= que[fr])
        {
            tmp = w[p];
            ++p;
        }
        else
        {
            tmp = que[fr];//获取队首元素
            ++fr;//出队
            //本题中需要找最大值,当队列为空时que[fr]为0,不会被选择
        }
        ++bk;
        que[bk] = tmp / 2;//入队
    }
    for(i = p;i < n;i++)
        res += w[i];
    for(i = fr;i <= bk;i++)
```

```
        res += que[i];
    printf("%lld",res);
    return 0;
}
```