

## 8.23 课堂笔记 36

### 本周作业

8.24~9.3题单

黄签:

P1022 (模拟)

P1023 (数学)

P1045 (高精度)

P1037 (floyed / dfs两种解法都写一下)

绿签:

P1043 (dp / dfs+前缀和)

P1069 (数学)

P4822 (最短路)

P1983 (拓扑排序)

### 课堂练习

#### 1. P7075

思路:

一眼大模拟

题目中给了很好的分类条件

公元 1582 年 10 月 15 日 (含) 以后:

公元 1582 年 10 月 5 日 (含) 至 10 月 14 日 (含):

公元 1582 年 10 月 4 日 (含) 以前:

打表肯定是行不通的, 因为数据规模太大了, 思考通过周期性减少数据规模

周期是什么?

公元 1582 年 10 月 4 日 (含) 以前:

每 4 年有一个闰年, 每 4 年的天数是  $365 \times 4 + 1 = 1461$ , 如果我们把天数模 1461, 得到的商就是距离起点公元前 4713 年的年数, 余数就是在这一年中是第几天

首先显然需要实现一个功能, 就是知道某一天是一年的第几天, 求这一天是几月几号 (为

了最后的输出)

```
const int day[12]={31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const int rday[12]={31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int sday(int rr)//求平年中是哪一天
{
    rr++;
    for(int j=0; j<12; j++)
    {
        if(rr>day[j])
        {
            rr=rr-day[j];
        }
        else
        {
            m=j+1;
            d=rr;
            break;
        }
    }
    return d;
}

int srday(int rr)//求闰年中是哪一天
{
    rr++;
    for(int j=0; j<12; j++)
    {
        if(rr>rday[j])
        {
            rr=rr-rday[j];
        }
        else
        {
            m=j+1;
            d=rr;
            break;
        }
    }
    return d;
}
```

其次 这个公元 1582 年 10 月 4 日 (含) 以前, 到底是哪一天的以前?

2299160

$= ((4710-2)/4+1 + (1576-0)/4+1) * 1461 + 366 + 365 + 276$

$= 1573 * 1461 + 1007$

$= 2298153 + 1007$

$= 2299160$

公元 1582 年 10 月 15 日 (含) 以后:

闰年计算规则改变了, 不能再四年一闰当周期来计算

当年份是 400 的倍数, 或日期年份是 4 的倍数但不是 100 的倍数时, 该年为闰年。

考虑到计算规则, 现在可以认为每四百年是一个周期

每四百年的天数就是  $400 * 365 + 97 = 146097$

那么现在也要实现一个功能, 就是知道某一天是一年的第几天, 求这一天是几月几号 (为了最后的输出)

四百年一个周期如何计算?

可以考虑打表

表的下标就是给定一个天数，表中存的数据就是这一天对应的年月日  
所谓洛外日，其定义为从公元 1200 年 1 月 1 日正午 12 点到此后某时刻间所经过的天数，若利用这一天文学法，那么洛外学生会觉得自己很自豪。  
如何把儒略历转化为洛外历？

儒略历的第 2299161 天也就是公元 1582 年 10 月 15 日  
那么公元 1582 年 10 月 15 日是洛外历的第多少天？

$$\begin{aligned} &381*365+92+287 \\ &=139065+92+287 \\ &=139065+379 \\ &=139444 \end{aligned}$$

（少了 1600，1596，1592，1588，1584，5 个闰年）

儒略历的第 2299161 天也就是洛外历的第 139444 天  
儒略历的第 2299162 天也就是洛外历的第 139445 天  
儒略历的第 2299163 天也就是洛外历的第 139446 天

所谓洛外日，其定义为从公元 1200 年 1 月 1 日正午 12 点到此后某时刻间所经过的天数，若利用这一天文学法，那么在 400 年一周期中，就可以很方便地计算出在哪一个周期，知道了在哪一个周期，那么再知道是具体哪一天也并非难事（如果你有一个打好的表）

代码：

```

#include <bits/stdc++.h>
using namespace std;
int y, m, d;
const int day[12]={31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
const int rday[12]={31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
int gy[150000], gm[150000], gd[150000];
int sday(int rr)//求平年中是哪一天
{
    rr++;
    for(int j=0; j<12; j++)
    {
        if(rr>day[j])
        {
            rr=rr-day[j];
        }
        else
        {
            m=j+1;//月
            d=rr;//天
            break;
        }
    }
    return d;
}

int srday(int rr)//求闰年中是哪一天
{
    rr++;
    for(int j=0; j<12; j++)
    {
        if(rr>rday[j])
        {
            rr=rr-rday[j];
        }
        else
        {
            m=j+1;//月
            d=rr;//天
            break;
        }
    }
    return d;
}

```

```

int got(int yy, int mm)
{
    int flag=0;
    if(yy%400==0)
    {
        flag=1;
    }
    else if(yy%4==0)
    {
        flag=1;
        if(yy%100==0)
        {
            flag=0;
        }
    }
    if(flag==0) return day[mm-1];
    else return rday[mm-1];
}

int slove()
{
    gy[0]=1, gm[0]=1, gd[0]=1;
    for(int i=1; i<=146097; i++)
    {
        gy[i]=gy[i-1], gm[i]=gm[i-1], gd[i]=gd[i-1]+1;
        if(gd[i]>got(gy[i], gm[i]))
        {
            gd[i]=1;
            gm[i]++;
            if(gm[i]>12)
            {
                gm[i]=1;
                gy[i]++;
            }
        }
    }
}

int main()
{
    slove();
    int q;
    long long r;
    cin>>q;
    while(q-->0)
    {
        cin>>r;
        if(r<=2299160)
        {
            y=r/1461*4-4713;
            r=r%1461;
            if(r<366)
            {
                sday(r);
            }
            else
            {
                r=r-366;
                y++;
                y+=(r/365);
                r=r%365;
                sday(r);
            }
            if(y<0)
            {
                cout<<d<<" "<<m<<" "<<0-y<<" BC"<<endl;
            }
            else
            {
                cout<<d<<" "<<m<<" "<<y+1<<endl;
            }
        }
        else
        {
            r=r-2159717;
            y=1200;
            y=y+400*(r/146097);
            r=r%146097;
            cout<<gd[r]<<" "<<gm[r]<<" "<<y+gy[r]<<endl;
        }
    }
}

return 0;

```

## 2. P7076

代码:

```
#include <bits/stdc++.h>
using namespace std;
unsigned long long a[1000006];
unsigned long long p[1000006];
unsigned long long q[1000006];
unsigned long long ans;
unsigned long long cnt;
set<unsigned long long> s;
int main()
{
    int n, m, c, k;
    cin >> n >> m >> c >> k;
    for(int i=0; i<n; i++)
    {
        cin >> a[i];
        ans = ans | a[i];
    }
    for(int i=0; i<m; i++)
    {
        cin >> p[i];
        s.insert(p[i]);
        cin >> q[i];
    }
    for(set<unsigned long long>::iterator i=s.begin(); i!=s.end(); i++)
    {
        unsigned long long now=pow(2,*i);
        if((ans&now)!=0)
        {
            cnt++;
        }
    }
    if(k==64&&cnt==0)
    {
        cout<<"18446744073709551616"<<endl;
    }
    else
    {
        cout<<(unsigned long long)pow(2,k+cnt-s.size())<<endl;
    }
    return 0;
}
```

## 3. P1090

思路:

一眼贪心

每次选取最小的两堆果子出来，然后合成一堆果子放回去

快排时间复杂度  $n\log n$ ，如果每放一个果子回去排一次序，时间复杂度将到  $n^2\log n$ ，

显然是不能接受的

所以需要使用能够维持内部有序的数据结构，优先队列，来解决这个问题

//升序队列

priority\_queue <int,vector<int>,greater<int> > q;

//降序队列

priority\_queue <int,vector<int>,less<int> > q;

//greater 和 less 是 std 实现的两个仿函数

代码:

```

#include <bits/stdc++.h>
using namespace std;
priority_queue<int, vector<int>, greater<int>> > q;

int main()
{
    int n, m, a, b, ans, sum;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int m;
        cin >> m;
        q.push(m);
    }
    while (q.size() != 1)
    {
        a = q.top();
        q.pop();
        b = q.top();
        q.pop();
        sum = a + b;
        ans += sum;
        q.push(sum);
    }
    cout << ans << endl;

    return 0;
}

```

#### 4. P6033

除数据范围外和 P1090 合并果子完全一致

也就是说解题的思路是一致的，只是因为这道题的数据范围开到了  $1e7$ ，所以需要一份时间复杂度是  $n$  的代码来解决问题

首先，题目给定的数据是无序的，不管采用什么样的方法去解决问题，首先如果要使用 `sort`，那么时间复杂度就已经来到了  $n\log n$ ，所以这里需要手动实现排序

计数排序是一个非基于比较的排序算法，元素从未排序状态变为已排序状态的过程，是由额外空间的辅助和元素本身的值决定的。

根据待排序集合中最大元素和最小元素的差值范围，申请额外空间；

遍历待排序集合，将每一个元素出现的次数记录到元素值对应的额外空间内；

对额外空间内数据进行计算，得出每一个元素的正确位置；

将待排序集合每一个元素移动到计算得出的正确位置上。

除数据范围外和 P1090 合并果子完全一致

也就是说解题的思路是一致的，只是因为这道题的数据范围开到了  $1e7$ ，所以需要一份时间复杂度是  $n$  的代码来解决问题

在解决了初次排序的问题之后，原来的做法需要使用到优先队列，优先队列本身使用的复杂度也是  $n\log n$  的

优先队列的使用是为了保证在每次的合并果子操作之后，所有的数据依然是有序的而在每次取出的两个果子，经过排序的初始队列肯定还是有序的，如果不再放回合并后的果子，就不会打乱顺序

那么合并之后的果子如果不放回原队列，单独放在一起，那么它们之间是有序的么？

由于上一次取出的果子是上一次最小的两个果子  $a_n, a_{n+1}$ ，这一次取出的果子是倒数三四小的两个果子  $a_{n+2}, a_{n+3}$ ，那么  $a_n < a_{n+1} < a_{n+2} < a_{n+3}$ ，显然  $a_n + a_{n+1} < a_{n+2} + a_{n+3}$

也就是说每次把合并之后的果子都放入一个队列之中，这个队列会是单调的代码：

```
queue<long long>a,b;
int cnt[600000]; //+ 0 1 1 1 e n P
long long abmin()
{
    if(b.empty()||(!a.empty())&&(a.front()<b.front()))
    {
        long long x=a.front();
        a.pop();
        return x;
    }
    else
    {
        long long x=b.front();
        b.pop();
        return x;
    }
}

int read()
{
    int s=0,w=1;char c=getchar();
    while(c>57||c<48) {if(c=='-')w=-1; c=getchar();}
    while(c>47&& c<58) {s*=10;s+=c-48;c=getchar();}
    return w*s;
}

int main()
{
    memset(cnt,0,sizeof(cnt));
    int n;
    long long ans=0;
    cin>>n;
    for(int i=0;i<n;i++)
    {
        int m;
        m=read();
        cnt[m]++;
    }
    for(int i=1;i<=100000;i++)
    {
        for(int j=1;j<=cnt[i];j++)
        {
            a.push(i);
        }
    }
    for(int i=0;i<n-1;i++)
    {
        long long x=abmin();
        long long y=abmin();
        ans+=x+y;
        b.push(x+y);
    }
    cout<<ans<<endl;
}
```

## 5. P2827

同 P6033 思路

开三个队列模拟原来的蚯蚓，切开之后较长的蚯蚓，切开之后较短的蚯蚓

留意的是每次变长的长度  $p$ ，如果其他所有蚯蚓变长  $p$ ，相当于新增加的蚯蚓变短  $p$

代码：



```

#include <bits/stdc++.h>
using namespace std;
queue<long long> a, b, c;
int r[700005];
priority_queue<int> ans;
long long abcmx()
{
    long long x;
    if(b.empty())
    {
        x=a.front();
        a.pop();
        return x;
    }
    if((!a.empty())&&(a.front()>b.front())&&(a.front()>c.front()))
    {
        x=a.front();
        a.pop();
        return x;
    }
    else if(b.front()>c.front())
    {
        x=b.front();
        b.pop();
        return x;
    }
    else
    {
        x=c.front();
        c.pop();
        return x;
    }
}

```

```

int main()
{
    int n, m, q, u, v, t;
    cin>>n>>m>>q>>u>>v>>t;
    double p=(double)u/v;
    for(int i=0; i<n; i++)
    {
        cin>>r[i];
    }
    sort(r, r+n);
    for(int i=n-1; i>=0; i--)
    {
        a.push(r[i]);
    }
    for(int i=1; i<=m; i++)
    {
        long long M=abcmx();
        M+=(i-1)*q;
        int a1=floor(p*(double)M), a2=M-a1;
        a1-=i*q, a2-=i*q;
        b.push(a1); c.push(a2);
        if(i%t==0) cout<<M<<" ";
    }
    while(!a.empty())
    {
        ans.push(a.front());
        a.pop();
    }
    while(!b.empty())
    {
        ans.push(b.front());
        b.pop();
    }
    while(!c.empty())
    {
        ans.push(c.front());
        c.pop();
    }
    cout<<endl;
    for(int i=1; ans.size(); i++)
    {
        if(i%t==0) cout<<ans.top()+m*q<<" ";
        ans.pop();
    }
    return 0;
}

```