5、STL

5.1 优先队列 priority_queue

empty() 如果队列为空返回真

pop() 删除对顶元素

push() 加入一个元素

size() 返回优先队列中拥有的元素个数

top() 返回优先队列队顶元素

在默认的优先队列中，优先级高的先出队。在默认的 int 型中先出队的为较大的数。

```
priority_queueeq1;//大的先出对
priority_queue<int,vector,greater >q2; //小的先出队
```

自定义比较函数：

```
struct cmp
{
bool operator ()(int x, int y)
{
return x > y; // x小的优先级高
//也可以写成其他方式，如： return p[x] > p[y];表示p[i]小的优先级高
}
};
priority_queue<int, vector, cmp>q;//定义方法
//其中，第二个参数为容器类型。第三个参数为比较函数。
```

结构体排序：

```
struct node
{
int x, y;
friend bool operator < (node a, node b)
{
return a.x > b.x; //结构体中，x小的优先级高
}
};
priority_queueeq;//定义方法
//在该结构中，y为值, x为优先级。
//通过自定义operator<操作符来比较元素中的优先级。
//在重载”<”时，最好不要重载”>”，可能会发生编译错误
```

5.2 set 和 multiset

set 和 multiset 用法一样，就是 multiset 允许重复元素。

元素放入容器时，会按照一定的排序法则自动排序，默认是按照 less<>排序规则来排序。不

能修改容器里面的元素值，只能插入和删除。
自定义 int 排序函数：（默认的是从小到大的，下面这个从大到小）

```
struct classcomp {
bool operator() (const int& lhs, const int& rhs) const
{return lhs>rhs;}
};//这里有个逗号的，注意
multiset<int,classcomp> fifth; // class as Compare
```

上面这样就定义成了从大到小排列了。
结构体自定义排序函数：
（定义 set 或者 multiset 的时候定义了排序函数，定义迭代器时一样带上排序函数）

```
struct Node
{
int x,y;
};
struct classcomp//先按照 x 从小到大排序，x相同则按照y从大到小排序
{
bool operator()(const Node &a,const Node &b)const
{
if(a.x!=b.x)return a.x<b.x;
else return a.y>b.y;
}
}; //注意这里有个逗号
multiset<Node,classcomp>mt;
multiset<Node,classcomp>::iterator it;
```

主要函数：
begin() 返回指向第一个元素的迭代器
clear() 清除所有元素
count() 返回某个值元素的个数
empty() 如果集合为空，返回 true
end() 返回指向最后一个元素的迭代器
erase() 删除集合中的元素 (参数是一个元素值，或者迭代器)
find() 返回一个指向被查找到元素的迭代器
insert() 在集合中插入元素
size() 集合中元素的数目
lower_bound() 返回指向大于（或等于）某值的第一个元素的迭代器
upper_bound() 返回大于某个值元素的迭代器
equal_range() 返回集合中与给定值相等的上下限的两个迭代器
(注意对于 multiset 删除操作之间删除值会把所以这个值的都删掉，删除一个要用迭代器)

```
#include<cstdio>
#include<map>
#include<utility>
using namespace std;
map<double,int,greater<double> > q;
map<double,int>::iterator r;
int i;
int n,m;
double d;
double sum;
int t[100010],nm[100010];
int main()
{
        scanf("%d%d",&n,&m);
        for(i=0;i<n;i++){
                scanf("%d%d",&t[i],&nm[i]);
                d=(double)nm[i]/t[i];
                q.insert(make_pair(d,i));
        }
        //printf("%d",m);
        /*for(r=q.begin();r!=q.end();r++){
                printf("%lf ",r->first);
        }*/
        while(m!=0&&!q.empty()){
                r=q.begin();
                if(t[r->second]<=m){
                        sum=sum+nm[r->second];
                        q.erase(r);
                        m=m-t[r->second];
                }
                else{
                        sum=sum+m*r->first;
                        m=0;
                }
        }
        printf("%.2lf",sum);
}
```

```
#include<cstdio>
#include<queue>
using namespace std;
struct node
{
	int x,y;
	bool operator < (const node & a) const
	{
		return x<a.x;
	}
}r;
priority_queue <node> q;
long long int p[500010];
int n,k;
long long i;
long long sum;
int main()
{
	while(~scanf("%d%d",&n,&k)){
		for(i=1;i<=n;i++){
			scanf("%lld",&p[i]);
		}
		sum=0;
		if(n>k){
			for(i=1;i<=k;i++){
				r.x=p[i];
				r.y=i;
				q.push(r);
			}
			for(i=k+1;i<=n;i++){
				r.x=p[i];
				r.y=i;
				q.push(r);
				node m=q.top();
				sum=sum+(m.x)*(i-m.y);
				q.pop();
			}
			for(i=n+1;i<=n+k;i++){
				node m=q.top();
				sum=sum+(m.x)*(i-m.y);
				q.pop();
			}
		}
		else{
			for(i=1;i<=n;i++){
				r.x=p[i];
				r.y=i;
				q.push(r);
			}
			for(i=k+1;i<=n+k;i++){
				node m=q.top();
```

```
                                sum=sum+(m.x)*(i-m.y);
                                q.pop();
                        }
                }
                printf("%lld\n",sum);
        }
}


#include<cstdio>
#include<queue>
using namespace std;
struct node{
        double a;
        int b;
        bool operator <(const node & x) const
        {
                return a<x.a;
        }
}r;
priority_queue<node> q;
int n,i;
long long t[100010],d[100010];
long long sum;
long long s;
node m;
int main()
{
        scanf("%d",&n);
        for(i=0;i<n;i++){
                scanf("%lld%lld",&t[i],&d[i]);
        }
        for(i=0;i<n;i++){
                r.a=(double)d[i]/t[i];
                r.b=i;
                //printf("%lf%d\n",r.a,r.b);
                q.push(r);
        }
        while(!q.empty()){
                m=q.top();
                s=s+sum*d[m.b];
                sum=sum+t[m.b];
                q.pop();
                //printf("%lld%lld\n",s,sum);
        }
        printf("%lld",s);
}
```

9、FFT

//HDU 1402 求高精度乘法

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <algorithm>
#include <math.h>
using namespace std;
const double PI = acos(-1.0);
//复数结构体
struct Complex
{
 double x,y;//实部和虚部 x+yi
 Complex(double _x = 0.0,double _y = 0.0)
 {
 x = _x;
 y = _y;
 }
 Complex operator -(const Complex &b)const
 {
 return Complex(x-b.x,y-b.y);
 }
 Complex operator +(const Complex &b)const
 {
 return Complex(x+b.x,y+b.y);
 }
 Complex operator *(const Complex &b)const
 {
 return Complex(x*b.x-y*b.y,x*b.y+y*b.x);
 }
};
/*
* 进行FFT和IFFT前的反转变换。
* 位置i和 （i二进制反转后位置）互换
* len必须去2的幂
*/
void change(Complex y[],int len)
{
 int i,j,k;
 for(i = 1, j = len/2;i <len-1;i++)
 {
 if(i < j)swap(y[i],y[j]);
 //交换互为小标反转的元素，i<j保证交换一次
 //i做正常的+1，j左反转类型的+1,始终保持i和j是反转的
 k = len/2;
 while(j >= k)
 {
 j -= k;
 k /= 2;
 }
 if(j < k)j += k;
 } }
/*
```

```
* 做FFT
* len必须为2^k形式，
* on==1时是DFT，on==-1时是IDFT
*/
void fft(Complex y[],int len,int on)
{
 change(y,len);
 for(int h = 2; h <= len; h <<= 1)
 {
 Complex wn(cos(-on*2*PI/h),sin(-on*2*PI/h));
 for(int j = 0;j < len;j+=h)
 {
 Complex w(1,0);
 for(int k = j;k < j+h/2;k++)
 {
 Complex u = y[k];
 Complex t = w*y[k+h/2];
 y[k] = u+t;
 y[k+h/2] = u-t;
 w = w*wn;
 }
 }
 }
 if(on == -1)
 for(int i = 0;i < len;i++)
 y[i].x /= len;
}
const int MAXN = 200010;
Complex x1[MAXN],x2[MAXN];
char str1[MAXN/2],str2[MAXN/2];
int sum[MAXN];
int main()
{
 while(scanf("%s%s",str1,str2)==2)
 {
 int len1 = strlen(str1);
 int len2 = strlen(str2);
 int len = 1;
 while(len < len1*2 || len < len2*2)len<<=1;
 for(int i = 0;i < len1;i++)
 x1[i] = Complex(str1[len1-1-i]-'0',0);
 for(int i = len1;i < len;i++)
 x1[i] = Complex(0,0);
 for(int i = 0;i < len2;i++)
 x2[i] = Complex(str2[len2-1-i]-'0',0);
 for(int i = len2;i < len;i++)
 x2[i] = Complex(0,0);
 //求DFT
 fft(x1,len,1);
 fft(x2,len,1);
 for(int i = 0;i < len;i++)
```

```
 x1[i] = x1[i]*x2[i];
 fft(x1,len,-1);
 for(int i = 0;i < len;i++)
 sum[i] = (int)(x1[i].x+0.5);
 for(int i = 0;i < len;i++)
 {
 sum[i+1]+=sum[i]/10;
 sum[i]%=10;
 }
 len = len1+len2-1;
 while(sum[len] <= 0 && len > 0)len--;
 for(int i = len;i >= 0;i--)
 printf("%c",sum[i]+'0');
 printf("\n");
 }
 return 0;
}
```

```
//HDU 4609
//给出 n 条线段长度，问任取 3 根，组成三角形的概率。
//n<=10^5 用 FFT 求可以组成三角形的取法有几种
const int MAXN = 400040;
Complex x1[MAXN];
int a[MAXN/4];
long long num[MAXN];//100000*100000会超int
long long sum[MAXN];
int main()
{
 int T;
 int n;
 scanf("%d",&T);
 while(T--)
 {
 scanf("%d",&n);
 memset(num,0,sizeof(num));
 for(int i = 0;i < n;i++)
 {
 scanf("%d",&a[i]);
 num[a[i]]++;
 }
 sort(a,a+n);
 int len1 = a[n-1]+1;
 int len = 1;
 while( len < 2*len1 )len <<= 1;
 for(int i = 0;i < len1;i++)
 x1[i] = Complex(num[i],0);
 for(int i = len1;i < len;i++)
 x1[i] = Complex(0,0);
 fft(x1,len,1);
 for(int i = 0;i < len;i++)
 x1[i] = x1[i]*x1[i];
 fft(x1,len,-1);
 for(int i = 0;i < len;i++)
 num[i] = (long long)(x1[i].x+0.5);
 len = 2*a[n-1];
 //减掉取两个相同的组合
 for(int i = 0;i < n;i++)
 num[a[i]+a[i]]--;
 for(int i = 1;i <= len;i++)num[i]/=2;
 sum[0] = 0;
 for(int i = 1;i <= len;i++)
 sum[i] = sum[i-1]+num[i];
 long long cnt = 0;
 for(int i = 0;i < n;i++)
 {
 cnt += sum[len]-sum[a[i]];
 //减掉一个取大，一个取小的
 cnt -= (long long)(n-1-i)*i;
 //减掉一个取本身，另外一个取其它
```

```
 cnt -= (n-1);
 cnt -= (long long)(n-1-i)*(n-i-2)/2;
 }
 long long tot = (long long)n*(n-1)*(n-2)/6;
 printf("%.7lf\n",(double)cnt/tot);
 }
 return 0;
}
```