2015 级算法期末上机解题报告

马国瑞

一、引言

本次上机题目共 9 道，满分 100 分，其中 4 道简单题（A~D）设置一题 20 分，60 分封顶，多过一题不再得分；3 道中等题（E~G）设置一题 15 分，30 分封顶，多过一题不再得分；2 道难题（H&I）设置一题 10 分，10 分封顶，多做不得分。另外，F 题有两个测试点，分别为 6 分（小数据）和 9 分（大数据）。

期末上机成绩按上述规则折算后平均得分 63.0952 分，90 分及以上人数 31 人（其中 100 分 5 人，90 分 26 人），70~89 分人数 40 人（其中 70 分 1 人，75 分 29 人，81 分 10 人），60~69 分人数 49 人（其中 60 分 45 人，66 分 4 人），60 分以下 48 人（其中 55 分 1 人，46 分 1 人，40 分 36 人，20 分 10 人）。

本次期末上机暴露出许多同学考试心态的问题，由于过于紧张导致在上机全过程中由于代码细节问题被简单题卡住（比如 B 题第二个多项式计算时候的循环终点问题、最终结果忘记对 10007 取模、x 与 y 其中一个为 0 时答案自认为是 0 等等），并且全程没有检查到这一错误的存在。在此建议大家今后任何考试请保持良好的心态，戒骄戒躁，夯实基础，定能取得满意成绩。

另外，在此感谢大家一学期以来对我们工作的大力支持！

若本解题报告有任何错误或疑问可通过邮箱联系：mgr9514@buaa.edu.cn

二、解题报告

A 简单·最大公约数

思路分析

本题考查数论算法，主要运用的是求解最大公约数的“辗转相除法”，送分题，全部同学通过此题。

具体思想由于在中学阶段已经学过相关知识，在此不再赘述。

参考代码

```
/*
 Author: 马国瑞
 Result: AC   Submission_id: 236062
 Created at: Sun Jan 01 2017 01:41:24 GMT+0800 (CST)
 Problem_id: 672 Time: 37  Memory: 2632
*/
```

```
#include <iostream>
using namespace std;
int main()
{
    int a,b;
    while(cin>>a>>b){
        //辗转相除法
        if(a<b) swap(a,b);
        while(a%b>0){
            int c=b;
            int d=a%b;
            a=c;
            b=d;
            if(a<b) swap(a,b);
        }
        cout<<b<<endl;
    }
}
```

B 简单·多项式计算器 II

思路分析

本题考查秦九韶算法，与第一次上机 A 题相似度很高。期末上机共 153 人提交该题代码，共 137 人通过此题。

对于单个一元 n 次多项式的值的求解方法如下：

方法一 直接计算法

此方法的思路是这样的：存储 x 值和一元 n 次多项式的系数数组，计算的时候针对每一项分别计算，相加，最后得到答案。实现代码如下：

```
int solve(int a[], int x, int n){
    /*此处 a 数组为一元 n 次多项式系数，
    n 为一元 n 次多项式的次数*/
    int sum = 0;
    int buf = 1;
    for(int i=0;i<=n;i++){
        buf=1;
        for(int j=0;j<i;j++)
            buf = (buf * x) % 10007;    //此处计算各项 x 的 i 次方
        sum+=(buf*a[i])%10007;
        sum%=10007;
    }
    return sum;
}
```

不难发现，所需的乘法次数为 $T(n)=n*(n-1)/2=O(n^2)$，大大浪费了时间，也是本题卡时间主要卡的代码。

方法二

此方法是对方法一的一种优化，即计算 x 的 k 次项的之前，通过一个变量存储前一项时候计算得到的 x 的 k-1 次幂。实现代码如下：

```
int solve(int a[], int x, int n){
    /*此处 a 数组为一元 n 次多项式系数，
    n 为一元 n 次多项式的次数*/
    int sum = 0;
    int buf = 1;      //通过 buf 的值存储 x 的幂
    for(int i=0;i<=n;i++){
        if(i>0)
            buf = (buf * x) % 10007;    //计算 x 的 i 次幂
```

```
        sum+=(buf*a[i])%10007;
        sum%=10007;
    }
    return sum;
}
```

和方法一相比，方法二的乘法次数大大降低。针对每次查询，所需要的乘法次数为 2n 次；而针对每组数据的 k 次查询，所需要的乘法次数为 2kn 次。

方法三 秦九韶算法（霍纳法则）

伪代码片段如下：

```
y = 0;
for i = n downto 0
    y = aᵢ + x * y
```

思路是：把一个 n 次多项式

$$f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$$

改写成如下形式：

$$f(x)=((\ldots(a_nx+a_{n-1})x+a_{n-2})x+\cdots+a_1)x+a_0$$

求多项式的值时，首先计算最内层括号内一次多项式的值，然后由内向外逐层计算一次多项式的值。这样，求 n 次多项式 f(x)的值就转化为求 n 个一次多项式的值。

针对每次查询，所需的乘法次数为 n；针对每组数据的 k 次查询，所需的乘法次数为 kn。

对于本题，分别对个多项式的值求解完之后再将两个多项式的值相乘即可。注意此过程也需要对 10007 取模。

方法二代码详见参考代码二，方法三代码详见参考代码一。

参考代码一

```
/*
 Author: 马国瑞
 Result: AC   Submission_id: 234139
 Created at: Wed Dec 28 2016 13:29:51 GMT+0800 (CST)
 Problem_id: 675 Time: 2804Memory: 2088
*/

#include <cstdio>
```

```
int a[100007],b[100007];
int main()
{
    int T;
    scanf("%d",&T);
    while(T--){
        int n;
        scanf("%d",&n);
        for(int i=0;i<=n;i++)
            scanf("%d",&a[i]);
        int m;
        scanf("%d",&m);
        for(int i=0;i<=m;i++)
            scanf("%d",&b[i]);
        int c,x,y;
        scanf("%d",&c);
        while(c--){
            scanf("%d%d",&x,&y);
            int bufa=0,bufb=0;
            //分别求解两个多项式的值然后相乘求解
            //秦九韶算法
            for(int i=n;i>=0;i--){
                bufa*=x;
                bufa%=10007;
                bufa+=a[i];
                bufa%=10007;
            }
            for(int i=m;i>=0;i--){
                bufb*=y;
                bufb%=10007;
                bufb+=b[i];
                bufb%=10007;
            }
            int ans=(bufa*bufb)%10007;
            printf("%d\n",ans);
        }
    }
}
```

参考代码二

```
/*
 Author: 马国瑞
```

```
 Result: AC   Submission_id: 234142
 Created at: Wed Dec 28 2016 13:32:54 GMT+0800 (CST)
 Problem_id: 675 Time: 1962Memory: 2064
*/

#include <cstdio>
int a[100007],b[100007];
int main()
{
    int T;
    scanf("%d",&T);
    while(T--){
        int n;
        scanf("%d",&n);
        for(int i=0;i<=n;i++)
            scanf("%d",&a[i]);
        int m;
        scanf("%d",&m);
        for(int i=0;i<=m;i++)
            scanf("%d",&b[i]);
        int c,x,y;
        scanf("%d",&c);
        while(c--){
            scanf("%d%d",&x,&y);
            int bufa=0,bufb=0;
            //对两个多项式分别计算然后相乘求解
            int px=1,py=1;
            for(int i=0;i<=n;i++){
                if(i>0) px*=x;
                px%=10007;
                bufa+=a[i]*px;
                bufa%=10007;
            }
            for(int i=0;i<=m;i++){
                if(i>0) py*=y;
                py%=10007;
                bufb+=b[i]*py;
                bufb%=10007;
            }
            int ans=(bufa*bufb)%10007;
            printf("%d\n",ans);
```

```
        }
    }
}
```

C 简单·简单的贪心

思路分析

本题使用的贪心策略如下：按照题目中给的方法最后求出的最小值的计算是通过每次优先取 2 个最大值计算，最后求出的最大值的计算是通过每次优先取 2 个最小值计算。最终分别得出最大值和最小值，作差即可。注意数据范围为 long long.

排序过程可使用优先队列或区间 sort，由于排序次数为$n-1$，对于第 i 次排序（1≤i≤n-1），需要对$n-i+1$个数进行排序，故时间复杂度为：

$$T(n)=\sum_{i=1}^{n-1}\Theta[(n-i+1)\lg(n-i+1)]=\Theta(n^2\lg n)$$

优先队列解法详见参考代码一，区间 sort 解法详见参考代码二。

期末上机共 155 人提交该题代码，125 人通过本题。

参考代码一

```
/*
 Author: 刘柘林
 Result: AC   Submission_id: 233677
 Created at: Tue Dec 27 2016 20:26:47 GMT+0800 (CST)
 Problem: 671 Time: 18  Memory: 2644
*/

#include <iostream>
#include <queue>
using namespace std;

#define LL long long

int main()
{
    int n;
    while(cin >> n)
    {
        priority_queue<int> pq1;
        priority_queue<int, vector<int>, greater<int> >pq2;
        int a,b,c;
        for(int i = 1; i <= n; i++)
        {
```

```
            cin >> a;
            pq1.push(a);
            pq2.push(a);
        }
        while(pq1.size()>1)
        {
            a = pq1.top();  pq1.pop();
            b = pq1.top();  pq1.pop();
            c = a*b+1;
            pq1.push(c);
        }
        int N = pq1.top();
        while(pq2.size()>1)
        {
            a = pq2.top();  pq2.pop();
            b = pq2.top();  pq2.pop();
            c = a*b+1;
            pq2.push(c);
        }
        int M = pq2.top();
        cout << M-N << endl;
    }
}
```

参考代码二

```
/*
 Author: 马国瑞
 Result: AC   Submission_id: 233765
 Created at: Tue Dec 27 2016 21:33:10 GMT+0800 (CST)
 Problem_id: 671 Time: 1   Memory: 2632
*/

#include <iostream>
#include <algorithm>
using namespace std;
bool cmp(long long a, long long b){
    if(a>b) return 1;
    else return 0;
}
int main() {
    int n;
    long long p[100],pp[100];
```

```
    long long ansmax=0,ansmin=0;
    while(cin>>n){
        for(int i=0;i<n;i++) {
            cin >> p[i];
            pp[i]=p[i];   //注意复制数组
        }
        sort(p,p+n);
        int lpos=0,rpos=n;
        //求解最大值
        while(rpos-lpos>1){
            p[rpos]=p[lpos]*p[lpos+1]+1;
            rpos++;
            lpos+=2;
            sort(p+lpos,p+rpos);
        }
        ansmax=p[lpos];
        //求解最小值
        lpos=0;
        rpos=n;
        sort(pp,pp+n,cmp);
        while(rpos-lpos>1){
            pp[rpos]=pp[lpos]*pp[lpos+1]+1;
            rpos++;
            lpos+=2;
            sort(pp+lpos,pp+rpos,cmp);
        }
        ansmin=pp[lpos];
        cout<<ansmax-ansmin<<endl;
    }
}
```

D 简单·梦想始发车

思路分析

期末上机本题共有 43 位同学提交代码，22 位同学通过本题。

本题为区间求和问题与第 k 顺序量求解问题结合起来的题。

首先通过区间求和问题的解题思想（第一次上机 C 题）枚举得到所有可能的路程，然后再在这一所有可能的路程数组中通过类快速排序寻找第 k 顺序量。

快速排序过程在此不做过多叙述。我们假设求的是第 k 小的数，数组 A[p…r]被划分成 A[p…q]和 A[q+1…r]两部分，则可以根据左边的元素个数 q-p+1 和 k 的大小关系只在左边和右边递归求解（这里需要注意递归右边的时候找的是第 k-(q-p+1)小的元素）。因此，快速排序算法可以得到优化。

算法分析

通过区间求和的思想枚举得到所有可能路程的时间为$O(n^2)$，在期望意义下类快速排序求第 k 顺序量的时间复杂度为$O(n^2)$（数组规模为$O(n^2)$，主方法计算过程参见第二次上机 E 题助教版解题报告），因此总的时间复杂度为$O(n^2)$.

参考代码

```
/*
 Author: 马国瑞
 Result: AC   Submission_id: 233992
 Created at: Wed Dec 28 2016 04:31:03 GMT+0800 (CST)
 Problem_id: 669 Time: 705 Memory: 65344
*/

#include <iostream>
#include <algorithm>
#include <cstring>
#include <iomanip>
#include <cstdio>
using namespace std;

int a[4017];
int b[16000007];

int divide(int l, int r)
{
    int x=b[r];
```

```
    int i=l-1;
    for(int j=l;j<r;j++){
        if(b[j]<=x){
            i++;
            //exchange a[i]&a[j]
            swap(b[i],b[j]);
        }
    }
    swap(b[i+1],b[r]);
    return i+1;
}
//快速排序
long long Quick_Sort(int head, int tail, int i )
{
    if(head==tail) return b[head];
    else if(head<tail){
        int q = divide(head, tail);
        int k = q-head+1;
        if(i==k) return b[q];
        //看 i 与 k 的大小关系只取一个分支
        else if(i<k)
            return Quick_Sort(head,q-1,i);
        else
            return Quick_Sort(q+1,tail,i-k);
    }
}

int main(){
    a[1]=0;
    int n,k;
    while(~scanf("%d%d",&n,&k)){
        memset(a,0,sizeof(a));
        memset(b,0,sizeof(b));
        for(int i=2;i<=n;i++){
            scanf("%d",&a[i]);
            a[i]+=a[i-1]; //区间求和
        }
        //区间求和枚举数组
        int cnt=0;
        for(int i=1;i<=n;i++){
            for(int j=1;j<=n;j++){
```

```
            if(j!=i){
                if(i<j) b[cnt]=a[j]-a[i];
                else b[cnt]=a[i]-a[j];
                cnt++;
            }
        }
    }

    int num=cnt/2;
    int buf=Quick_Sort(0,cnt-1,k);
    printf("%d\n",buf);
  }
}
```

F 中等・Magry 恼人的词典编辑

思路分析

期末上机共计 86 位同学提交本题代码，39 位同学通过本题得到满分 15 分，另有 15 位同学获得 6 分。

本题求得的最小代价为冒泡排序运行过程中需要对相邻两个数交换的次数。然而冒泡排序时间复杂度为$O(n^2)$，不符合本题对时间复杂度的要求。因此，我们必须要将问题转化才能求解本题。

可以通过举例得到，所求的最小代价也是整个数组的逆序数（严格的证明过程不在此赘述）。因此，我们可将问题转化为求逆序数的问题。求解逆序数的问题，归并排序是一种上佳的选择。

简单来说，在归并排序操作中，当把 rightSubArray（右半部分有序数组）中的元素复制到原 Array 的时候，**统计这个数字一下子跨过了 leftSubArray（左半部分有序数组）中的多少个数字，这个数字就是这次排序中的逆序数。**

逆序数的求解涉及分治算法，需要考虑以下三种情形：

① 完全在左子数组中；

② 完全在右子数组中；

③ 合并数组时跨越中间的数的情况。

最终结果即为上述三种情况所得逆序数之和。

具体计算上，两个子区间已经完成升序排序，我们不断从两个子区间的左端取出最小的元素，从左到右放置在一个临时数组上。如果我们发现当前最小元素（记为 A）在右子区间上，说明左子区间中剩下的所有元素和 A，都是逆序对（①A<左子区间最小元素<左子区间剩下的其他元素 ②左子区间所有元素和 A 的位置关系保持不变）。

本题只是将比较的元素换成字符串，其他未变。对于 string 类型，由于>、<已被重载为字典序大小的比较，故可直接使用。当然，也可以使用 char 数组、strcmp(char* a, char* b)函数比较两个字符串的大小（strcmp 函数也可自己实现）。

string 实现参见参考代码一，char 数组实现参见参考代码二。

算法分析

和归并排序相同，将数组递归二分子问题分别求解，将两个长度之和为 n 的有序子序列合并成一个有序序列过程至多进行 n-1 次比较，时间复杂度为$\Theta(n)$. 对于两路归并排序算法，递归式如下：

$$T(n)=\begin{cases}\Theta(1) & (n=1)\\ 2T\left(\frac{n}{2}\right)+\Theta(n) & (n>1)\end{cases}$$

由主定理可知，a=2，b=2，$log_b a=1$，$f(n)=\Theta(n)=\Theta(n^{log_b a})$

因此，$T(n)=\Theta(n^{log_b a}lgn)=\Theta(nlgn)$

故算法的时间复杂度为$\Theta(nlgn)$

参考代码一

```
/*
 Author: 马国瑞
 Result: AC   Submission_id: 234190
 Created at: Wed Dec 28 2016 14:44:55 GMT+0800 (CST)
 Problem_id: 674 Time: 538 Memory: 30284
*/

#include<cstring>
#include<iostream>
#include<string>
#include<cstdio>
using namespace std;
string a[100010],b[100010];
//a 将原数组复制之后存储排序结果，b 为辅助数组

//归并排序算法
long long slot(int head, int tail)
{
    long long ans=0;
    if(head>=tail)   //需要处理这样的情形
        return 0;
    int mid=(head+tail)/2;
    int i=head,j=mid+1,k=i;
    //统计左右两个子数组的逆序数并进行子数组的排序
    ans=slot(head,mid)+slot(mid+1,tail);

//合并子数组过程
//操作过程中累加逆序对的数量
    while((i<=mid) && (j<=tail))
    {
        if(a[i]>a[j])
        {
```

```
            b[k++]=a[j++];
            ans+=(mid-i+1);
        }
        else
            b[k++]=a[i++];
    }

    //必要的补充
    while(i<=mid)
    {
        b[k++]=a[i++];
        //ans+=(mid-i+1);
    }
    while(j<=tail)
        b[k++]=a[j++];
    for(i=head; i<=tail; ++i)
        a[i]=b[i];
    return ans;
}
int main()
{
    int n;
    int T;
    cin>>T;
    while(T--){
        cin>>n;
        for(int i=0;i<n;i++)
            cin>>a[i];
        long long ans=slot(0,n-1);
        cout<<ans<<endl;
    }
}
```

参考代码二

```
/*
 Author: 马国瑞
 Result: AC   Submission_id: 234235
 Created at: Wed Dec 28 2016 15:24:34 GMT+0800 (CST)
 Problem_id: 674 Time: 231 Memory: 10824
*/

#include<cstring>
```

```
#include<iostream>
#include<string>
#include<cstdio>
using namespace std;
typedef struct{
    char str[30];
}S;
//自己实现的 strcmp
bool cmp(S a, S b){
    int lena=strlen(a.str);
    int lenb=strlen(b.str);
    int minlen=min(lena,lenb);
    bool flag=false;
    for(int i=0;i<minlen;i++){
        if(a.str[i]>b.str[i]){
            flag=true;
            return 1;
        }
        else if(a.str[i]<b.str[i]){
            flag=true;
            return 0;
        }
    }
    if(flag==false){
        if(lena>lenb) return 1;
        else return 0;
    }
}
S a[100010],b[100010];
//a 将原数组复制之后存储排序结果，b 为辅助数组

//归并排序算法
long long slot(int head, int tail)
{
    long long ans=0;
    if(head>=tail)   //需要处理这样的情形
        return 0;
    int mid=(head+tail)/2;
    int i=head,j=mid+1,k=i;
    //统计左右两个子数组的逆序数并进行子数组的排序
    ans=slot(head,mid)+slot(mid+1,tail);
```

```
//合并子数组过程
//操作过程中累加逆序对的数量
    while((i<=mid) && (j<=tail))
    {
        if(strcmp(a[i].str,a[j].str)>0)
        {
            //b[k++]=a[j++];
            strcpy(b[k].str,a[j].str);
            k++;
            j++;
            ans+=(mid-i+1);
        }
        else
            strcpy(b[k++].str,a[i++].str);
    }

    //必要的补充
    while(i<=mid)
    {
        strcpy(b[k++].str,a[i++].str);
        //ans+=(mid-i+1);
    }
    while(j<=tail)
        strcpy(b[k++].str,a[j++].str);
    for(i=head; i<=tail; ++i)
        strcpy(a[i].str,b[i].str);
    return ans;
}

int main()
{
    int n,t,x,y;
    int T;
    scanf("%d",&T);
    while(T--){
        scanf("%d",&n);
        //初始化字符串数组
        for(int i=0;i<n;i++){
            memset(a[i].str,0,sizeof(a[i].str));
            memset(b[i].str,0,sizeof(b[i].str));
```

```
        }
        for(int i=0;i<n;i++)
            scanf(" %s",a[i].str);
        long long ans=slot(0,n-1);
        printf("%lld\n",ans);
    }
}
```

G 中等・Magry 摆什锦糖

思路分析

期末上机共计 92 位同学提交本题代码，64 位同学通过本题。

本题为 01 背包问题，以美味程度为质量，以 k 为背包容量，价值可以视作为 1，只不过问题的角度变换为能不能有一个方案使得质量正好为背包容量 k.

对于本题，可直接设一个 bool 类型的 dp 数组，然后类似 01 背包问题那样对问题求解即可，最后判断 dp[k]的值即可。

算法分析

时间复杂度为$\Theta(n^2)$

参考代码

```
/*
 Author: 朱辉
 Result: AC   Submission_id: 235960
 Created at: Fri Dec 30 2016 14:58:43 GMT+0800 (CST)
 Problem: 681 Time: 6   Memory: 2932
*/

#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;

int Weight[1005];
bool ans[200005];

int n,k;

int main()
{
    while(~scanf("%d%d",&n,&k))
    {
        for(int i=1;i<=n;i++)
            scanf("%d",&Weight[i]);

        memset(ans,0,sizeof(ans));
```

```
        ans[0]=1;
        for(int i=1;i<=n;i++)
        {
            ans[Weight[i]]=1;
            for(int j=k;j>=0;j--)
            {
                if(ans[j])
                {
                    if(j>Weight[i])
                        ans[j-Weight[i]]=1;
                    ans[j+Weight[i]]=1;
                }
            }
        }
        if(ans[k]) printf("Yes\n");
        else printf("No\n");
    }
}
```

H 难题 · 大独裁者的以后

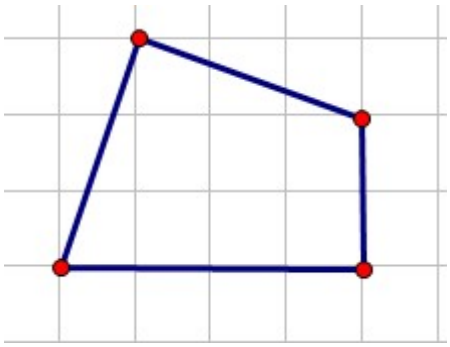
思路分析

期末上机共计 15 位同学提交本题代码，5 位同学通过本题。

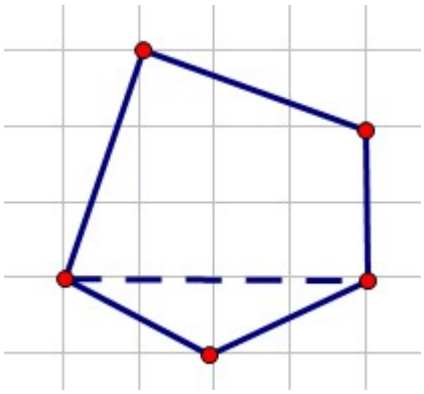
本题为“稳定凸包”问题，在题目及样例的理解上有一定难度。可分析得出：当凸包上每一条边都存在 3 条及以上给定的点的时候可唯一确定“大独裁者的地盘”。理由如下：

首先“凸包”一定是凸多边形，满足条件：所有的内角均在$(0, 180°)$范围内。

给定某条边少于 3 个点的情况：

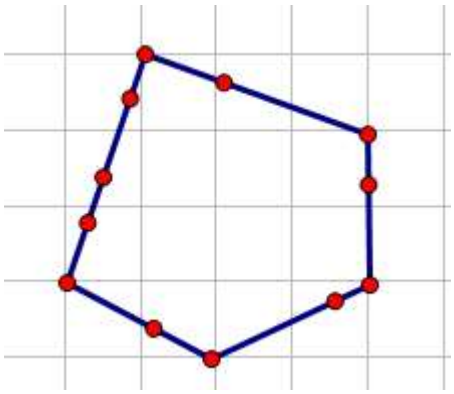


这 4 个点确实能构成一个凸包，但它们也有可能是凸包的部分点，因为这 4 个点也能构成下图所示凸包的一部分点，因此不符合题意：



如果所有的边上都有 3 个及以上给定的点，那么不可能再找到一个点使它扩展成一个新的凸包，否则构成的新多边形将是凹的。

下图就是符合题意的“稳定凸包”：



那么，问题就转化为凸包上是否任意一条边都存在三点共线的问题。做法是：根据给定的点，取最下、最左方的点为第一个点顺时针或逆时针排序，构建凸包，然后求是否任意一条边均三点共线即可。

参考代码使用 Graham 算法构建图包，时间复杂度为$\Theta(nlgn)$

部分参考自：http://www.cnblogs.com/xdruid/archive/2012/06/20/2555536.html

参考代码

```
/*
 代码转载自：http://blog.csdn.net/non_cease/article/details/7782468
 Result: AC   Submission_id: 235140
 Created at: Thu Dec 29 2016 01:46:48 GMT+0800 (CST)
 Problem_id: 680 Time: 11  Memory: 2748
*/

#include <iostream>
#include <algorithm>
#include <stdio.h>
#include <math.h>

using namespace std;

const int N = 40005;

typedef double DIY;

struct Point
{
    DIY x,y;
};

Point p[N];
```

```
Point stack[N];
Point MinA;

int top;

DIY dist(Point A,Point B)
{
    return sqrt((A.x-B.x)*(A.x-B.x)+(A.y-B.y)*(A.y-B.y));
}

DIY cross(Point A,Point B,Point C)
{
    return (B.x-A.x)*(C.y-A.y)-(B.y-A.y)*(C.x-A.x);
}

bool cmp(Point a,Point b)
{
    DIY k=cross(MinA,a,b);
    if(k>0) return 1;
    if(k<0) return 0;
    return dist(MinA,a)<dist(MinA,b);  //这里共线的点按距离从小到大排序
}

void Graham(int n)
{
    int i;
    for(i=1; i<n; i++)
        if(p[i].y<p[0].y||(p[i].y==p[0].y&&p[i].x<p[0].x))
            swap(p[i],p[0]);
    MinA=p[0];
    sort(p+1,p+n,cmp);
    stack[0]=p[0];
    stack[1]=p[1];
    top=1;
    for(i=2; i<n; i++)
    {
        //注意这里我们把共线的点也压入凸包里
        while(cross(stack[top-1],stack[top],p[i])<0&&top>=1) --top;
        stack[++top]=p[i];
    }
}
```

```
bool Judge()
{
    for(int i=1;i<top;i++)
    {
        //判断凸包的一条边上是否至少有 3 点
        if((cross(stack[i-
1],stack[i+1],stack[i]))!=0&&(cross(stack[i],stack[i+2],stack[i+1]))!=0)
            return false;
    }
    return true;
}

int main()
{
    int t,n,i;
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d",&n);
        for(i=0;i<n;i++)
            scanf("%lf%lf",&p[i].x,&p[i].y);
        if(n<6)
        {
            puts("No");
            continue;
        }
        Graham(n);
        if(Judge()) puts("Yes");
        else       puts("No");
    }
    return 0;
}
```