2016 级算法第一次上机解题报告

15081070 张雨任

一、总结

本次上机主要考查知识点为数论、递推、分治等算法知识。

关于题目，ABCD 考察了数论和递推的应用，有的是数论和递推相结合，E 考察了快速排序分治法的应用，F 题考察遍历的剪枝，时间复杂度为 O(n^2)，也可以从高度入手时间复杂度为 O(n)，G 题考察分治和归并排序的应用；FG 上机的时候没有做出来。递推要找到前一项或前几项和当前项的逻辑关系，然后再转化到数学关系上；数论有套路，有些常用的数论做法要了解，数论和递推如果实在没有头绪还可以多列几项或者找规律；使用分治法的时候，需要理解这个问题的本质，分割成合理可求的子问题，然后再合并。

二、解题报告

A. The stupid owls

思路分析

本题考察错排和阶乘的递推公式，对于 n 个元素，有递推公式：

$$ans[n]=0 , n=1$$

$$ans[n]=1 , n=2$$

$$ans[n]=(n-1)(ans[n-1]+ans[n-2]) , n>=3$$

此公式算出来是 n 个元素错排的种数，想求概率，还要除以总的情况数 n!，最后答案乘以 100，结果保留 2 位小数，再输出百分号。

算法分析

这里讲一下错排递推公式：对于 n 个元素进行错排，进行如下动作。首先，先错排第一个元素，那么它可以选择除了自己以外的其他所有位置，那么就是有(n-1)种选择。其次，错排剩余的(n-1)个元素，假设第一个元素错排之后排在了第 j 个元素的位置上，那么接下来错排该第 j 个元素，此时可再分两种情况：情况一，j 元素选择错排到刚刚“侵占自己位置”的

第一个元素的位置，那么相当于互换了这两个元素的位置，剩余的(n-2)个元素没受到影响，因此相当于错排剩余的(n-2)个元素，即有 ans[n-2]中错排方法；情况二，j 元素没有选择错排到刚刚“侵占自己位置”的第一个元素的位置，选择了其他位置，那么就相当于除去第一个元素，包括 j 的剩余(n-1)个元素进行错排，由于 j 不能排在第一个元素的位置上，否则会和情况一有重叠，因此相当于 j 也参与到错排之中，因此是(n-1)个元素进行错排，因此是 ans[n-1]。情况一和情况二是独立的两种情况，因此应用加法原理得出 ans[n-1]+ans[n-2]，然后再应用乘法原理乘上第一个元素的选择种数，因此是(n-1)(ans[n-1+ans[n-2])。算法的时间复杂度为 O(n)。

参考代码

```
#include <cstdio>
using namespace std;

int n;
double ans[30];
double jiecheng[30];
double res;

int main() {
    while(~scanf("%d",&n)) {
        ans[1]=0;
        ans[2]=1;
        jiecheng[1]=1;
        jiecheng[2]=2;
        for(int i=3;i<=n;i++) {
            ans[i]=(i-1)*(ans[i-1]+ans[i-2]);
            jiecheng[i]=jiecheng[i-1]*i;
        }
        res=ans[n]/jiecheng[n]*100;
        printf("%.2f\%\n",res);
    }
}
```

B. ModricWang 和数论

思路分析

上机的时候是通过找规律做出来的。

ans[1]=1，可取余数为 0

ans[2]=2，可取余数为 0, 2

ans[3]=3，可取余数为 0, 1, 3

ans[4]=3，可取余数为 0, 1, 4

ans[5]=4，可取余数为 0, 1, 2, 5

ans[6]=4, ans[7]=5, ans[8]=5, ans[9]=6, ans[10]=6 …

对于第 n 项，就有(n+1)/2 种（不算自身），再额外加上自身（除以任意大于自身的数），因此是(n+1)/2+1 种选择。

算法分析

正规思路：首先，某数 k 除以任意比自己大的数均可以取得余数为 k 自身，这是 1 种；其次，除了余数为 k 的情况外，k 的余数必可取得小于 k 的一半的数，如：对于整数 6，除以任意数可取得的余数有 0, 1, 2，因此是(n+1)/2 种选择。两种情况和在一起，就是(n+1)/2+1 种选择。时间复杂度 O(1)。

参考代码

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

long long a;

int main() {
    while(cin>>a) {
        cout<<(a+1)/2+1<<"\n";
    }
}
```

C. AlvinZH 去图书馆

思路分析

本题可以运用递推的方法解决。

因为只能进行 1, 2, 3 步的跨越，所以某步的种数等于前三步种数之和。但是由于不能连续两次直接跨三步的操作，因此对递推还要有所改良。

算法分析

先看总体思路，可以选择 1, 2, 3 步进行跨越，因此显然有 ans[n]=ans[n-1]+ans[n-2]+ans[n-3]，即可以分别从前 1 个，前 2 个，前 3 个砖进行跨越。但还要考虑不能连续进行两次“3 格”跨越，简称“跨 3”，因此原式要进行修改，即为 ans[n]=ans[n-1]+ans[n-2]+res[n-3]，达到 n 之前的最后一步跨 1 和跨 2 都没有影响，不存在“连续跨 3”的非法情况，因此 ans[n-1]和 ans[n-2]可以直接加到 ans[n]上。问题在于跨 3 之前的最后一步不能“跨 3”，否则就连续两次“跨 3”操作了，但 ans[n-3]是有可能最后一步跨 3 的，因此要改用 res 来表示最后一步“跨 3”的情况。现在来看如何定义 res 数组以及 res 的递推公式，res[i] 代表第 i 步之前的最后一步（即第 i-1 步）进行了跨 3，因此 res[i] 不能够从 res[i-3] 通过一次跨 3 操作到达，所以显然有 res[n]=ans[n-1]+ans[n-2]，这样就排除了连续两次跨 3 的问题。

最终递推式：

$$ans[i]=ans[i-1]+ans[i-2]+res[i-3]\ ,\ res[i]=ans[i-1]+ans[i-2]$$

递推的时间复杂度为 O(n)。

参考代码

```
#include <cstdio>
using namespace std;

int n;
long long ans[52];
long long res[52];

int main() {
    res[0]=1;
```

```
    res[1]=2;
    res[2]=3;
    ans[0]=1;
    ans[1]=2;
    ans[2]=4;
    for(int i=3;i<52;i++){
        ans[i]=ans[i-1]+ans[i-2]+res[i-3];
        res[i]=ans[i-1]+ans[i-2];
    }
    while(~scanf("%d",&n)) printf("%lld\n",ans[n-1]);
}
```

D. 水水的 Horner Rule

思路分析

本题考查霍纳法则的应用和进制转换。

善良的助教已经给出了霍纳法则的公式，多项式的形式又恰好和 h 进制转十进制的按位展开类似，即可把多项式的未知数 x 看做 h（进制），a0 到 an 分别为被转换的数从低位到高位的每一位。

算法分析

把进制转换转化为多项式求值，从而可以使用霍纳法则进行求解。要注意在给定公式 A(x)=a0+x*(a1+x*(a2+...+x*(an-1+x*an) ··))中，因为 an 乘了 n-1 次 x（也就是 h），所以从 an 到 a0 对应 h 进制数从高位到低位，因此最高位对应着数字 an。数据读入用 string 比较方便，可以直接用下标获取每一位，再把 char 转为 int 进行运算。注意数据范围，x 的十进制在 int 范围内，其 h (2<=h<=10) 进制可在 int 外，因此使用 long long 比较保险。霍纳法则的时间复杂度 O(n)。

参考代码

```
#include <cstdio>
#include <iostream>
#include <string>
using namespace std;

long long h1,h2,n,y1,y2;
string x1,x2;

int main() {
    std::ios::sync_with_stdio(false);
    while(cin>>n) {
        for(int j=0;j<n;j++) {
            cin>>h1>>x1>>h2>>x2;
            y1=0;
            y2=0;
            for(int i=0;i<=x1.length()-1;i++) {
                y1=(y1*h1)+(x1[i]-'0');
            }
```

```
          for(int i=0;i<=x2.length()-1;i++) {
              y2=(y2*h2)+(x2[i]-'0');
          }
          cout<<y1+y2<<"\n";
      }


  }

}
```

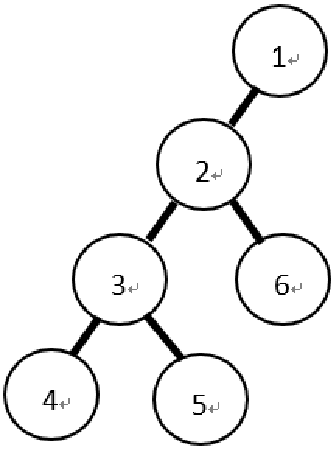
E. ModricWang's QuickSort

思路分析

本题考查快排和分治，需要模拟快排的划分，按照题意模拟一遍即可。重点在于要数清楚递归的趟数，标记好当今操作的是第几趟第几部分元素。然后分治的时候要严格根据题中的操作步骤进行模拟，令变量 pivot 为分隔元素，保证 pivot 左边的数都比 pivot 小，pivot 右边的数都比 pivot 大。

算法分析

首先看一下输出，要输出第二趟的第二部分，这个很容易出错，容易输出其他趟数其他部分的元素，或者上下界多一少一，可以使用计数器来进行数组部分的标记，每次递归前进行累加，代表现在在操作第几部分的元素，递归树（部分）如图（结点的数字为标记）：



只要大于 3 就判断：如果到了 4 和 5，就返回，这样才能够保证计数器为 6 的时候输出的是第二趟第二部分的元素。因此输出第二趟第二部分即等价于输出计数器为 6 时的 i 和 j 之间的数。

剩下的就考细心了，按照题中划分的操作步骤严格模拟整个划分的过程（否则虽然排序没问题，但是具体每次递归的元素可能与题意不同，导致输出不同），循环中要维护“pivot 左边的数都比 pivot 小，pivot 右边的数都比 pivot 大”，因此每次都要把 pivot 左边比 pivot 大的数和 pivot 右边比 pivot 小的数交换，具体步骤分析见代码注释。除了要注意的数组下标上下界的多一少一，还要额外要注意 mid（分隔元素 pivot 的下标）的取值，应为 (low+high)/2+1，因为数列元素个数为偶数个时选择后一个作为分隔元素。刚开始写的时候就没有+1，所以样

例总是没有输出，这道题要注意细节。快排的时间复杂度为 O(nlgn)。

参考代码

```
#include <cstdio>

#include <algorithm>
using namespace std;

int n,cnt;
int data[1000007];

void partition(int val[],int low,int high,int &cnt) {
    if(cnt>3) {
        if(cnt==6) {
            for(int i=low;i<=high;i++) printf("%d ",val[i]);
            printf("\n");
            return;
        }
        else return;
    }
    else {
        if(low<high) {
            int i=low;//对应步骤 1
            int j=high;//对应步骤 1
            int pivot=val[(i+j)/2+1];//对应步骤 1
            while(i<j) {//对应步骤 3，每次比较 i 和 j 的大小
                while(i<j && val[i]<=pivot) i++;//对应步骤 4
                while(i<j && val[j]>pivot) j--;//对应步骤 5
                if(i<j) swap(val[i],val[j]);
                //对应步骤 6，交换两指针指向的值，并循环
            }//对应步骤 7，退出
            partition(val,low,i-1,++cnt);//分治
            partition(val,i,high,++cnt);//分治
        }
    }
}

int main() {
    while(~scanf("%d",&n)) {
        for(int i=0;i<n;i++) scanf("%d",&data[i]);
        cnt=1;
```

```
        partition(data,0,n-1,cnt);
    }
}
```

F. AlvinZH 的儿时梦想——木匠篇

思路分析

本题考查遍历的剪枝，但是暴力不剪枝也可以过，也可以从高度入手。

第一种思路，题中要求结果为桶的最大盛水量，即体积最大，直径为选取的两边的横坐标距离，高为两条边较小的，两边要分别在原点两端，所以从两边遍历即可。

第二种思路是从高度入手，总共只有 0~100 总共 101 中高度，从高度从大到小进行遍历，每个高度都选择半径最大的情况，这样时间复杂度为 O(n)。

算法分析

思路一：

首先用结构体存储位置和高度，然后按照位置坐标从小到大排序，存在结构体数组中。使用嵌套循环，外层从左到右遍历，遍历原点左边的边，内层从右到左遍历，遍历原点右边的点，每两个边用圆柱体积公式计算盛水量，记录遍历过的盛水量的最大值。坑点：两个边分别在原点的两侧，一开始没注意到；减少用 double 进行运算，一开始输入用的 double，结果一直超时，int 的运算要比 double 快很多，这个很重要；hint 中提到了 PI 取 acos(-1)，遇到浮点数的问题一定要考虑精度；数学库函数比四则运算慢很多。

剪枝：以原点左侧为例，右侧类似。对于原点左侧的所有边，找到高度最高的边，有多个最高的边就选取最左边的边。对于符合条件的最高边，其右侧的边绝对无法和原点右侧的某边组成更大容积的圆柱体，原因是这个最高边，又高、半径又长，而该最高边右侧的边，又短、半径又小，因此左侧的边只需要遍历到这个边即可，其这个最高边的右侧的均可忽略。原点右侧同理。时间复杂度 O(n^2)。

思路二：

首先，在输入数据的时候，用两个数组来存储每个高度对应的横坐标值，输入的 x 小于 0 记在 leftx 数组中，x 大于 0 记在 rightx 数组中，如果有遇到高度相同的时候，则记录距离原点更远的横坐标，目的是使得半径拉得更长。这样用 leftx[h]和 rightx[h]即可获取到高度为 h 的横坐标绝对值最大的两个点，分别在原点两侧。

其次，由于高度都为整数，所以在遍历的时候从大到小遍历每个高度，选择距离原点最远的横坐标作为 lx 和 rx 值，比如当前高度为 i，则能够保证在大于等于 i 的高度中获取到距离原点最远的横坐标，即可算出当前高度所能达到的最大直径，从而算出体积，然后实时更新

体积的最大值即可。时间复杂度为 O(n)。

参考代码一（暴力）

```
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

int n;
const double PI=acos(-1);
double maxi=0.0;

struct Stick {
    int x;
    int h;
}stick[105];

bool cmp(Stick a,Stick b) {
    return a.x<b.x;
}

int main() {
    while(~scanf("%d",&n)) {
        for(int i=0;i<n;i++) {
            scanf("%d%d",&stick[i].x,&stick[i].h);
        }
        sort(stick,stick+n,cmp);
        maxi=0.0;
        for(int i=0;i<n && stick[i].x<=0;i++) {
            for(int j=n-1;j>=0 && stick[j].x>=0;j--) {
                maxi=max(maxi,PI*(stick[j].x-stick[i].x)*(stick[j].x-
                     stick[i].x)*min(stick[i].h,stick[j].h)/4.0);
            }
        }
        printf("%.3f\n",maxi);
    }
}
```

参考代码二（暴力+剪枝）

```
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

int n,lh,lx,rh,rx;
const double PI=acos(-1);
double maxi=0.0;

struct Stick {
    int x;
    int h;
}stick[105];

bool cmp(Stick a,Stick b) {
    return a.x<b.x;
}

int main() {
    while(~scanf("%d",&n)) {
        lh=lx=rh=rx=0;
        for(int i=0;i<n;i++) {
            scanf("%d%d",&stick[i].x,&stick[i].h);
            if(stick[i].x<0) {
                if(stick[i].h>lh) {
                    lh=stick[i].h;
                    lx=stick[i].x;
                }
                else if(stick[i].h==lh) {
                    lx=(lx<stick[i].x?lx:stick[i].x);
                }
            }
            else if(stick[i].x>0) {
                if(stick[i].h>rh) {
                    rh=stick[i].h;
                    rx=stick[i].x;
                }
                else if(stick[i].h==rh) {
```

```
                rx=(rx>stick[i].x?rx:stick[i].x);
            }
        }
        else {
            if(stick[i].h>lh) {
                lh=stick[i].h;
                lx=stick[i].x;
            }
            else if(stick[i].h==lh) {
                lx=(lx<stick[i].x?lx:stick[i].x);
            }
            if(stick[i].h>rh) {
                rh=stick[i].h;
                rx=stick[i].x;
            }
            else if(stick[i].h==rh) {
                rx=(rx>stick[i].x?rx:stick[i].x);
            }
        }
    }
    sort(stick,stick+n,cmp);
    maxi=0.0;
    for(int i=0;i<n && stick[i].x<=lx;i++) {
        for(int j=n-1;j>=0 && stick[j].x>=rx;j--) {
            maxi=max(maxi,PI*(stick[j].x-stick[i].x)*(stick[j].x-
                stick[i].x)*min(stick[i].h,stick[j].h)/4.0);
        }
    }
    printf("%.3f\n",maxi);
  }
}
```

参考代码三（从高度入手）

```
#include <cstdio>
#include <cmath>
#include <algorithm>
using namespace std;

const double PI=acos(-1);
```

```
int n,x,h,lx,rx;
int leftx[105],rightx[105];
double ans;

int main() {
    while(~scanf("%d",&n)) {
        for(int i=0;i<=100;i++) {
            leftx[i]=105;
            rightx[i]=-105;
        }
        for(int i=0;i<n;i++) {
            scanf("%d%d",&x,&h);
            if(x<=0) leftx[h]=min(leftx[h],x);
            if(x>=0) rightx[h]=max(rightx[h],x);
        }
        lx=105;
        rx=-105;
        ans=0.0;
        for(int i=100;i>=0;i--) {
            lx=min(lx,leftx[i]);
            rx=max(rx,rightx[i]);
            if(lx<=0 && rx>=0) {
                ans=max(ans,i*PI*(rx-lx)*(rx-lx)/4.0);
            }
        }
        printf("%.3f\n",ans);
    }
}
```

G. D&C--玲珑数

思路分析

本题考查分治思想和归并排序的应用。玲珑对定义为一个数列两两元素之间的关系，即 i<j 且 a[i]>2*a[j]，玲珑数为一个数列玲珑对的个数。很容易想到暴力 O(n^2)遍历的方法，但是会超时，这里使用归并排序的改进版，使用的性质是归并时参与合并的两个子序列是单调的，以及分治的思想：一串数的玲珑数等于前半部分子序列的玲珑数、后半部分子序列的玲珑数以及 a[i]在前半部分 a[j]在后半部分所组成的玲珑数的和。

算法分析

首先，使用分治法是符合玲珑数的定义的，一串数的玲珑数等于前半部分子序列的玲珑数、后半部分子序列的玲珑数以及 a[i]在前半部分 a[j]在后半部分所组成的玲珑数的和。所以通过分治法，结果可以写成这三个部分的加和，前两部分可以通过分治来解决，第三部分在归并排序进行归并的时候解决。

其次，来讲一下为什么使用归并排序来解决这个问题。玲珑对是一个数列前后两个元素的关系，与元素在序列中的顺序和大小有关。通俗来讲就是前面的数大于后面的数的两倍。在归并时，合并的两个子序列是具有单调性，假设子序列 1 的长度为 m，子序列 2 的长度为 n，在计算玲珑关系的时候，子序列 1 的每个值都要遍历，而子序列 2 的值在每一次循环中只需要遍历一段即可，比如上一次循环遍历到子序列 1 的第 i 个值，子序列 2 遍历到 j+1 时不能满足玲珑关系，即子序列 1 的第 i 个值可与子序列 2 的前 j 个值满足玲珑关系，那么下一次循环，子序列 1 取第 i+1 个值，子序列 2 的前 j 个值就不用再看了，因为比子序列 1 第 i+1 项小的第 i 项都和子序列 2 的前 j 项满足玲珑关系，那就更不必说第 i+1 项了，因此只需要从子序列 2 的第 j+1 项开始检查玲珑关系即可，因此两个子序列的检查虽然看上去是嵌套的，但其实两个子序列都是线性遍历，因此时间复杂度有 O(m+n)，而暴力算法是 O(m*n)。

最后讲一下为什么归并排序能够保证不重复计算玲珑对。举个例子，对于数列 9, 3, 5, 1，显然玲珑数为 3，即(9, 3), (9, 1), (5, 1)。在归并的时候，第一次归并的两个子序列为 9 和 3，根据上述算法检查玲珑关系得到计数器值为 1，之后对 9, 3 进行排序，排序的同时能够消除这两个数的玲珑关系，得到 3, 9, 5, 1，此时 3, 9 不存在玲珑关系，并且组内的排序不影响组外数与组内数的玲珑关系，如：(9, 1), (5,1)的玲珑关系没有被打破，因此排序能够在计数的同时消除被计数的子序列的玲珑关系，同时还可以维护还没有被计数的数之间的玲珑关系，从而做到不重不漏。归并排序的时间复杂度 O(nlgn)。

除此之外，还要额外注意元素都在 int 范围内，但是乘二就超过了，所以要用 long long。p、q 大小不定，要判断是否需要交换。

参考代码

```
#include <cstdio>
#include <algorithm>
#define ll long long
using namespace std;

int n,t,p,q;
ll origin[10005];
ll data[10005];

ll merge_process(int l,int mid,int r) {
    ll cnt=0;
    int i=l;
    int j=mid+1;
    int s=r-l+1;
    while(i<=mid) {//计算两个单调子序列各取一个数组成的玲珑对的个数，O(m+n)
        while(j<=r && data[i]>2*data[j]) j++;
        cnt+=(j-mid-1);
        i++;
    }
    //以下为归并过程
    ll* tmpArr;
    tmpArr=new ll[s];
    int k=0;
    i=l;
    j=mid+1;
    while(i<=mid && j<=r) {
        if(data[i]<=data[j]) tmpArr[k++]=data[i++];
        else tmpArr[k++]=data[j++];
    }
    while(i<=mid) tmpArr[k++]=data[i++];
    while(j<=r) tmpArr[k++]=data[j++];
    for(int a=0;a<k;a++) data[a+l]=tmpArr[a];
    delete []tmpArr;
    return cnt;
}
```

```
ll solve(int l,int r) {
    ll ans=0;
    if(l<r) {
        int mid=(l+r)/2;
        //分治，3 个子问题
        ans+=solve(l,mid);
        ans+=solve(mid+1,r);
        ans+=merge_process(l,mid,r);
    }
    return ans;
}

int main() {
    while(~scanf("%d",&n)) {
        for(int i=0;i<n;i++) {
            scanf("%lld",&origin[i]);
        }
        scanf("%d",&t);
        for(int i=0;i<t;i++) {
            scanf("%d%d",&p,&q);
            if(p>q) swap(p,q);//判断 p、q 交换
            for(int j=p;j<=q;j++) {
                data[j]=origin[j];
            }
            printf("%lld\n",solve(p,q));
        }

    }

}
```