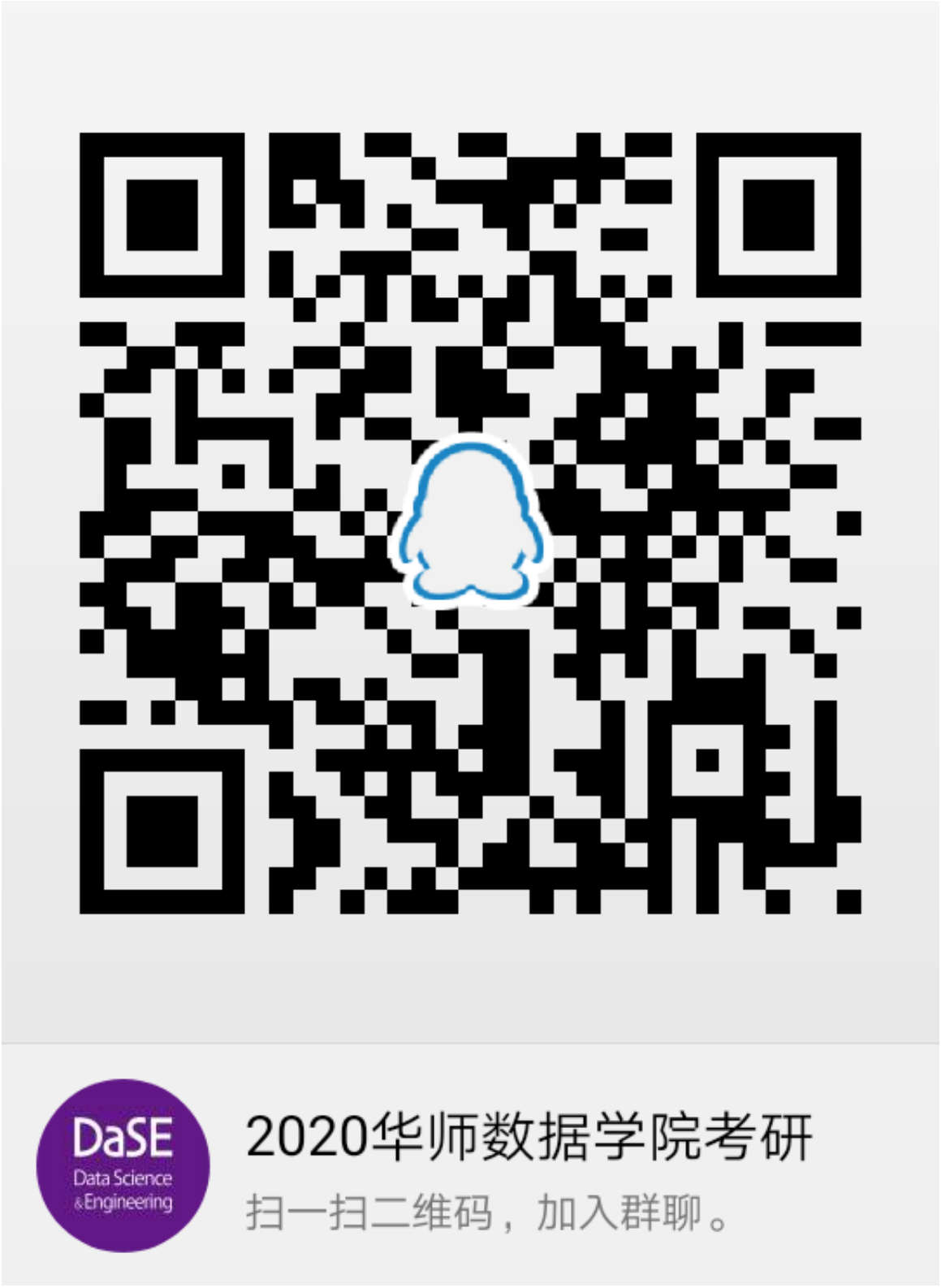
2019 数据学院研究生复试机试题解

前言

此文件包含以下内容

1. 在前言中，我会讲一些关于我对于考研机试的一些经验和想法。(仅代表我一家之言，请根据实际情况制定自己的复习计划)
2. 在第一部分中，我会给出十道机试题的提示，以便给想要自己尝试解题的同学一些帮助。
3. 在之后的一个部分，我会着重讲解每一个题目的细节和技巧，并且给出代码(此代码并不能够保证AC题目，因为没有经过评测机评测，但是能够保证思路是正确的)
4. 本文作者：Tonjar 群里可以联系我，仅供群里同学备考参考，禁止转载或者用于商业用途。(群二维码如下)
5. Leo同学提出了一些修改建议(群里也可以找到)
6. 一些代码为了排版到同一页而可能被压缩。

相关的题目自行在群文件中寻找，在此就不复制了。



我之前有尝试打过ACM（ACM预选赛落榜选手），自学过一些相关内容，所以觉得机试还比较轻松。从机试情况来看前4道比较简单（表一：各题AC人数），而最多的人也AC了4道(表二：AC各个题数的人数)。我对于今年的分数进行了一波分析，得出的结论是，你如果机试综合初试成绩(初试成绩 * 0.6 + 机试 * 0.16)排名排在尾部，那你除非面试分数极高否则没戏。所以如果把前4题作为基本分，学硕至少应该去试着全部拿到。专硕则需要争取更高的分数（表二）。

表一：各题AC人数(总人数：29)

A	B	C	D	E	F	G	H	I	J
29	26	21	17	7	5	2	7	3	1

表二：AC各个题数的人数

AC数	1	2	3	4	5	6	7	8	9	10
学硕	2	4	4	5	1	0	1	0	1	0
专硕	0	0	2	4	2	1	1	0	0	1
总	2	4	6	9	3	1	2	0	1	1

复试语言推荐：c++

个人一直是使用c++来刷算法题的，其他语言对于某些题可能有优势，所以可以作为后备语言。我当时准备的是，如果有大数题就用python，其他一律用c++，今年听说用java的同学普遍出现了一些问题。所以可能的话准备使用c++比较保险。

我当时准备复试的时候就奔着满分去的，所以最后还是达到了自己的目的。然后我讲讲我准备复试的经验。不过我当时复试前有很长一段时间再写毕业论文，所以留给我准备时间比较短，如果你没有相关基础，我认为至少留出至少1个月时间来刷题，并且需要保证每天大部分时间在这个上面。(总之按照自己的基础来，刷的题越多越好)

我上面讲了，我首先有一点基础(这十道题我基本以前都碰到过)，其次，我想要满分。所以我对于机试的准备基本上是练熟练度。(虽然我手速真的不够快，只有最后两道我是第一个AC的)。所以我第一个礼拜拿了一个普通键盘敲简单题（因为我平常使用的是笔记本）。我一开始大约5-8分钟一题，熟练之后大约是2-4分钟一题。这里主要为了敲快+不要拼写错误。然后之后几天主要复习了数据结构里面的一些代码，然后把一些稍微复杂的代码打印出来作为考前的复习。因为我时间不够基本就挑重点复习。

对于没有基础的同学的建议↓

是如果你只想保本(AC基本题)，那你只需要练习简单题+熟悉语言。之所以推荐使用c++，因为里面库函数已经可以支持你完成简单题。

你务必了解的c++库中的工具

1. <algorithm> 中的 sort stable_sort lower_bound min max
2. <cmath>
3. <queue> <stack> <set>
4. <cstring> 中的 memset

此外你需要了解的c++输入的知识

```
while(cin>>xxx);
while(getline(cin,s));//如果getline之前用过cin,需要在当中再插入一个getline  需要库文件<string>  s的类型为string
cin.getline(s,limit)//s的类型为char[]
while(scanf(xxx)!=0);
```

不过在本次复试中，似乎测试都是单个样例，所以并不需要上门的while。(尽管如此我觉得你可能还是需要了解一下)

格式输出请使用printf，一般输出最后不加空格，也就是说

```
1_2_3_4_
1_2_3_4
```

上面_代表空格，第一种可能WA，而第二种则是会AC。

此外你需要了解的知识

1. 在函数内数组一般最大开到1e5，大小为1e6的数组需要在main函数以外开
2. 在程序中，被执行最多次的代码一般最多执行1e6-1e8次。也就是说n为1e10-1e16的需要一个常数时间复杂度的算法；n为1e7-1e8的时间复杂度应为O(n)；n为1e5-1e6的时间复杂度为O(nlogn)；如果n很小才可能用$O(n^2)$和$O(n^3)$的算法。
3. 你需要注意什么时候使用long long

如果你希望能够尽量多地得到一些分数，那你则需要掌握尽可能多的相关技巧。

PS:简单题是指几乎所有可行算法都能AC，且数据量很小。一般都有非常短的解法。

复试机试题的提示

1-4简单题

1. 数字排序：利用库函数/或者手写任意一种常见的排序算法
2. 找零钱：贪心即可
3. 斐波那契数列：暴力枚举
4. 字符串拼接：利用库函数

5-10题

5. 矩阵求和：经典题，可以预先把某些特殊的子矩阵的权值和求出，然后在询问时对于这些特殊的子矩阵的权值和，做简单的运算可以直接得出结果。
6. 阶乘：试试二分查找
7. 分石子：同样试试二分查找
8. 小华的游戏：BFS裸题
9. 消消乐：枚举
10. 素数：注意题中：也就是该数的价值=该数的因子个数 这句话

复试机试的题解

A-数字排序

利用sort函数(无论是c++版的还是c版的注意两个函数的用法细节不同)，并且sort函数默认从小到大排序，需要自定义比较函数。

```
#include<iostream>
#include<algorithm>
using namespace std;
bool cmp(const int&a,const int&b){
    return a>b;
}
int main(){
    int n;
    int a[1111];
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>a[i];
    sort(a,a+n,cmp);
    for(int i=0;i<n;i++)
        cout<<a[i]<<(i==n-1?'\n':' ');
}
```

B-找零钱

要最少，则当然是尽量去用大的。而2元是无限的且1元的有一张，所以不可能不存在方案。然后这里可以使用贪心策略（根据日常经验）。而因为5元的零钱有限，所以对此需要判断一下，是否会用完5元的钱。 PS:不是所有类似题目都是贪心策略。如果面额为1，5，7找25零钱，贪心结果 7*3+1*4 实际上5*5就已经更优了。像这种奇怪面额的就需要考虑动态规划。

```
#include<iostream>
using namespace std;
int main(){
    int x,a;
    cin>>x>>a;
    if(x<=5*a){//5元够
        a=x/5;//需要用的5元的个数
        x%=5;
        cout<<a+(x/2)+(x%2)<<endl;//x/2为2元数量，x%2为1元数量
    }
    else{
        x-=a*5;
        cout<<a+(x/2)+(x%2)<<endl;
    }
}
```

C-斐波那契数列

这里只需要通过递推法强行枚举出所有小于1e9的斐波那契数，然后逐个比对就可以。(如果用递归的话，那你就会超时。除非用记忆化搜索。但是没有必要。)

```
#include<iostream>
using namespace std;
int main(){
    long long a=1,b=1;
    int n;
    cin>>n;
    for(;n>=a;){
        if(n==a){
            cout<<"Yes\n";
            return 0;
        }
        long long c=a+b;//斐波那契数列的递推写法
        a=b;
        b=c;
    }
    cout<<"No\n";
}
```

D-字符串拼接

两种做法，一种是经典的去重排序，另一种是先连接两个字符串，然后用库函数排序，最后在输出时再按题目要求做处理：重复的字符只输出一个。

```
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    char s1[111],s2[55];
    cin>>s1>>s2;
    int n=strlen(s1),m=strlen(s2);
    memcpy(s1+n,s2,sizeof(s2));
    n+=m;
    sort(s1,s1+n);
    cout<<s1[0];
    for(int i=1;i<n;i++)
        if(s1[i]!=s1[i-1])
            cout<<s1[i];
    cout<<endl;
}
```

E-矩阵求和

首先预先计算出所有(1,1,x,y)的答案存储在（x,y）位置上，当计算 (x1,y1,x2,y2)时该值等于 (0,0,x2,y2)+(0,0,x1-1,y1-1)-(0,0,x1-1,y2)-(0,0,x2,y1-1)。关于为什么则可以通过画图知道（类似计算一个矩形的面积）。 然后输入数据是先给查询坐标后给原始矩阵，所以需要先把查询坐标存在一个数组里，再预处理原始矩阵。

这里讨论一下这种方法和暴力方法的时间复杂度度。

此方法$O(nm+q)$ 暴力方法 $O(qnm)$

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int a[1111][1111];//n*m矩阵
int q[100100][4];//查询坐标矩阵
int main(){
    int n,m,Q;
    cin>>n>>m>>Q;
    for(int i=0;i<Q;i++)
        cin>>q[i][0]>>q[i][1]>>q[i][2]>>q[i][3];
    memset(a,0,sizeof(a));
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
            cin>>a[i][j];
    for(int i=1;i<=n;i++)//矩阵预处理开始，从左往右叠加
        for(int j=2;j<=m;j++)
            a[i][j]+=a[i][j-1];
    for(int i=2;i<=n;i++)//从上到下叠加
        for(int j=1;j<=m;j++)
            a[i][j]+=a[i-1][j];
    for(int i=0;i<Q;i++){
        int x1=q[i][0],y1=q[i][1],x2=q[i][2],y2=q[i][3];
        cout<<a[x2][y2]+a[x1-1][y1-1]-a[x2][y1-1]-a[x1-1][y2]<<endl;
    }
}
```

F-阶乘

这题应该有一些很强的方法，但是我不会。我觉得二分查找就可以解决这个问题，注意到计算n!的末尾连续0的个数的时间复杂度为O(logn)，如果使用二分查找记可能答案最大值为M则总体算法时间复杂度为O(logMlogn)所以显然可以接受。

主体部分就是二分搜索的代码。此外需要一些细节上的处理。研究代码请对照数据结构书上的二分搜索代码研究。注意我采用的二分代码是假设搜索区间是左闭右开的。

关于这类题目的一个说明：

1. 这类题目和最初学的二分查找的相似之处在于都需要一个单调数列。而一开始学的单调数列是可以看见的，而这里是隐含的，假设f(x)是单调函数，数列 $a_i = f(i)$ 显然是单调的，而针对这类函数通过a_i 找i 不就显然是可以使用二分查找的。(当然你做题的时候就不怎么显然了，想做出这种题还是需要一点经验。)本题就意图你使用二分查找，还专门给你了f(x)。
2. 对于一些看起来困难而复杂的题目，是可以考虑一下二分这种方法的，因为他至少可以将答案范围取对数放进时间复杂度里。(那这个时候就可以注意到一个这类题的特征，答案范围很可能是能够用int 或者long long 直接表示的，那种需要取模的就无法使用)

```cpp
#include<iostream>
using namespace std;
int cnt(int n){//计算n!末尾连续0的个位
    int m=5;
    int s=0;
    while(n/m!=0){
        s+=n/m;
        m*=5;
    }
    return s;
}
int main(){
    int s=2,e=cnt(2e8)+1;
    int k;
    cin>>k;
    while(s<e){//此部分对照对数组二分查找的代码进行学习
        int mid=(s+e)/2;
        int n=cnt(mid);
        if(n<k)s=mid+1;
        else if(n>k)e=mid;
        else if(cnt(mid-1)!=k){
            cout<<mid<<endl;
            return 0;
        }
        else e=mid;
    }
    e--;while(!(cnt(e-1)<k&&cnt(e)>=k))e++;
    cout<<e<<endl;
}
```

G-分石子

类似上题也是用一下二分，但是这里并不是直接搜索答案，而是搜索每个筐最多放多少重量，然后计算在此前提下需要多少个筐。恰好等于筐数的最小结果即为所求。

这道题显然难度大于上一道题，原因在于你需要自己想出，哪个数列是单调的。如果你能想到这里有一个数列是单调的，你就可以去考虑使用二分查找，从而想出整个题目的解决方法。对比上一道题，这一题没有给出这方面的任何提示，所以要做出这道题也是需要一定的经验的。

代码上，本题与上题结构一致。所以先将上题研究透后再看本题。

```
#include<iostream>
using namespace std;
int a[10010];
int n,k;
int cnt(int m){
    int s=0,cnt=1;
    for(int i=0;i<n;i++){
        if(a[i]>m)return 1e9;//注意这边1e9表示无穷大
        if(a[i]+s>m){
            s=a[i];cnt++;continue;
        }
        s+=a[i];
    }
    return cnt;
}
int main(){
    cin>>n>>k;
    int su=0;
    for(int i=0;i<n;i++){
        cin>>a[i];
        su+=a[i];
    }
    int s=0,e=su+10;
    while(s<e){
        int mid=(s+e)/2;
        int l=cnt(mid);
        if(l>k)s=mid+1;
        else if(l<k)e=mid;
        else if(cnt(mid-1)!=l){
            cout<<mid<<endl;
            return 0;
        }
        else e=mid;
    }
    e--;
    while(cnt(e)>k)e++;
    cout<<e<<endl;
}
```

H-小华的游戏

BFS裸题，然后要加一个来源矩阵（你可能会在最短路算法里面学习过这种东西，参考清华的数据结构书），最后可以通过递归找到那条路径。注意这里对路径有要求，所以这里搜索的时候需要注意顺序。

```
#include<iostream>
#include<queue>
#define x first
#define y second
int dx[]={1,0,0,-1},
    dy[]={0,-1,1,0};//注意顺序和对应
char s[]="DLRU";
using namespace std;
int M[555][555];
int pre[555][555];
void printpath(int x,int y){
    if(pre[x][y]==-1)return;
    int k=pre[x][y];
    printpath(x-dx[k],y-dy[k]);
    cout<<s[k];
}
int main(){
    int n,m;cin>>n>>m;
    memset(M,1,sizeof(M));
    for(int i=1;i<=n;i++)//外面留一圈墙，以简化代码
        for(int j=1;j<=m;j++)
            cin>>M[i][j];
    queue< pair<int,int> >q;
    q.push(make_pair(1,1));
    M[1][1]=1;//利用地图对走过的路进行标记（顺便记录最短路长度
    pre[1][1]=-1;
    while(!q.empty()){
        int x=q.front().x,y=q.front().y;
        q.pop();
        for(int i=0;i<4;i++){
            int nx=x+dx[i],ny=y+dy[i];
            if(M[nx][ny]!=0)continue;
            M[nx][ny]=M[x][y]+1;
            pre[nx][ny]=i;
            if(nx==n&&ny==m){//提前结束，不写一般问题不大
                while(!q.empty())q.pop();
                break;
            }
            q.push(make_pair(nx,ny));
        }
    }
    cout<<M[n][m]-1<<endl;
    printpath(n,m);cout<<endl;
}
```

I-消消乐

我们注意到 n*m<=10 所以我们其实，可以通过枚举所有可能消去所有c的方式然后得到一个最优方案。（最大枚举数量$3^{10} = 59049$）那如何枚举呢，如果你学习过如果枚举全排列的话，这题对你来说就不是很难了。如果你没有学过的话，建议先去理解一下那个。

这里我们从左向右，从上向下搜索，如果遇到c则我们采取某种将它消去的方式，然后递归地解决剩下的。

```
#include<iostream>
#include<cmath>
using namespace std;
char s[10][10];
int n,m,p,q;
int maxscore(int x,int y,int maxsco){
    if(x==n)return maxsco;//达到搜索树的叶子节点
    if(y==m)return maxscore(x+1,0,maxsco);
    if(s[x][y]=='x')return maxscore(x,y+1,maxsco);
    int s1=0,s2=0,s3=0;//分别表示三种消去方式的得分
    s[x][y]='x';
    s1=maxscore(x,y+1,maxsco+p);
    if(y!=m-1&&s[x][y+1]=='c'){
        s[x][y+1]='x';
        s2=maxscore(x,y+1,maxsco+q);
        s[x][y+1]='c';
    }
    if(x!=n-1&&s[x+1][y]=='c'){
        s[x+1][y]='x';
        s3=maxscore(x,y+1,maxsco+q);
        s[x+1][y]='c';
    }
    s[x][y]='c';//注意重置s[x][y]使其重新可用
    return max(s1,max(s2,s3));
}
int main(){
    cin>>n>>m>>p>>q;
    for(int i=0;i<n;i++)
        cin>>s[i];
    cout<<maxscore(0,0,0)<<endl;
}
```

J-素数

该题其实真的很简单，因为他在求1-n一共含多少因数。如果你还没思路的话，看一下下面这张表你可能就知道怎么做了。

n\因子	1	2	3	4	5	6	7	8	9	10
1	√									
2	√	√								
3	√		√							
4	√	√		√						
5	√				√					
6	√	√	√			√				
7	√						√			
8	√	√		√				√		
9	√		√						√	
10	√	√			√					√

他的意思就是求这张表的行到n时里面有多少个格子填上了√。

那他本来给你的是一行一行计算的公式，我们现在只需要一列一列计算即可。

```
#include<iostream>
using namespace std;
int MOD=1e9+7;
int main(){
    int n;
    cin>>n;
    int sum=0;
    for(int i=1;i<=n;i++)
        sum=(sum+n/i)%MOD;
    cout<<sum<<endl;
}
```