CSP/NOIP 入门级模拟赛 Day2 - Solution

第 1 题　寻找朋友

【算法分析】

此题的大意就是让字典序第 1 小的字符串和第 1 小的数配对，字典序第 2 小的字符串和第 2 小的数配对…字典序第 n 小的字符串和第 n 小的数配对。

那么我们可以把字符串按字典序从小到大排序，再将数从小到大排序，依次输出即可。

【核心代码】

```
sort(name+1,name+1+n); //按字典序小到大排序
sort(num+1,num+1+n);   //从小到大排序
for(RI i=1;i<=n;i++)
  cout<<name[i],printf("%d\n",num[i]);
```

【参考程序】

```
#include <bits/stdc++.h>
#define RI register int
using namespace std;

inline int read()
{
    int x=0,f=1;char s=getchar();
    while(s<'0'||s>'9'){if(s=='-')f=-f;s=getchar();}
    while(s>='0'&&s<='9'){x=x*10+s-'0';s=getchar();}
    return x*f;
}

const int N=200100;
int n;
string name[N];
int num[N];

int main()
{
    freopen("find.in","r",stdin);
    freopen("find.out","w",stdout);
    n=read();
    for(int i=1;i<=n;i++)
        cin>>name[i];
    for(int i=1;i<=n;i++)
        num[i]=read();
    sort(name+1,name+1+n); //按字典序小到大排序
    sort(num+1,num+1+n);   //从小到大排序
    for(int i=1;i<=n;i++)
        cout<<name[i],printf("% d\n",num[i]);
```

```
    return 0;
}
```

第 2 题　摆放棋子

【算法分析】

方案数很多，不能一个一个统计方案，所以考虑 dp。

设 f[i][j]表示处理到第 i 行的第 j 列时的方案数，

状态转移方程：

```
f[i][j]=i+j                    if i==1||j==1
f[i][j]=f[i-1][j]+f[i][j-1]    else
```

初态：均为 0

目标：f[n][m]

时间复杂度：O(n^2)

【核心代码】

```
for(int i=1;i<=n;i++)//dp
  for(int j=1;j<=m;j++)
    if(i==1||j==1) f[i][j]=i+j;
    else f[i][j]=f[i-1][j]+f[i][j-1];
```

【参考程序】

```
#include <bits/stdc++.h>
#define RI register int
using namespace std;
const int N=31;
int n,m;
long long f[N][N];

int main()
{
    freopen("chessman.in","r",stdin);
        freopen("chessman.out","w",stdout);
    scanf("%d%d",&n,&m);
    for(RI i=1;i<=n;i++)//dp
        for(RI j=1;j<=m;j++)
            if(i==1||j==1)
                f[i][j]=i+j;
            else
                f[i][j]=f[i-1][j]+f[i][j-1];
    printf("%lld\n",f[n][m]);
    return 0;
}
```

第 3 题　统计分值

【算法分析】

显然，我们可以使用区间前缀和来解决本题。

设 b(i,j)表示以(1,1)为左上角，(i,j)为右下角的区间和，可以得出 b(i,j)=b(i-1,j)+b(i,j-1)-b(i-1,j-1)+a(i,j)，a(i,j)表示(i,j)这个元素的值。

再设 f(i,j)表示以（i-R+1,j-C+1)为左上角，(i,j)为右下角的区间和,可以得出 f(i,j)=b(i,j)-b(i-R,j)-b(i,j-C)+b(i-R,j-C)。

最后 ans = max f(i,j) (1<=i<=n,1<=j<=n)。

【核心代码】

```
for(i=1;i<=n;i++)
    for(j=1;j<=m;j++) //计算以(1,1)为左上角，(i,j)为右下角的区间和
      b[i][j]=b[i-1][j]+b[i][j-1]-b[i-1][j-1]+a[i][j];
for(i=r;i<=n;i++)
  for(j=c;j<=m;j++) { //计算以(i-R+1,j-C+1)为左上角，(i,j)为右下角的区间和
    f[i][j]=b[i][j]-b[i-r][j]-b[i][j-c]+b[i-r][j-c];
    ans=max(ans,f[i][j]); //更新答案
  }
```

【参考程序】

```
#include <bits/stdc++.h>
using namespace std;
int n,m,r,c,i,j,k,a[1001][1001],tot,ans,b[1001][1001],f[1001][1001];
int main()
{
    freopen("aya.in","r",stdin);
    freopen("aya.out","w",stdout);
    cin>>n>>m>>r>>c;
    for(i=1;i<=n;i++)
    for(j=1;j<=m;j++)
    scanf("%d",&a[i][j]);
    for(i=1;i<=n;i++)
    for(j=1;j<=m;j++) //计算以(1,1)为左上角，(i,j)为右下角的区间和
    b[i][j]=b[i-1][j]+b[i][j-1]-b[i-1][j-1]+a[i][j];
    for(i=r;i<=n;i++)
    for(j=c;j<=m;j++) //计算以(i-R+1,j-C+1)为左上角，(i,j)为右下角的区间和
    {
        f[i][j]=b[i][j]-b[i-r][j]-b[i][j-c]+b[i-r][j-c];
        ans=max(ans,f[i][j]); //更新答案
    }
    cout<<ans<<endl;
    fclose(stdin);fclose(stdout);
}
```

第 4 题　食堂打菜

【算法分析】

很显然，可以使用 dp 解决此题。

直接读入的杂乱无章的区间定是不易做 dp 的，我们先把区间按终点快排一遍。

(这里令 l[i]表示第 i 个区间的左端点，r[i]表示第 i 个区间的右端点,sum[i]表示第

i 个区间的食堂个数)

很显然，我们可以以处理的区间数为 dp 的“阶段”。

设 f[i]表示处理完第 i 个区间的右端点以前的所有窗口且选择了吃第 i 个区间的所有窗口的最多食物个数。

状态转移方程：

f[i]=f[j]+sum[i]　　j<i&&r[j]<l[i]

初态：f[i]=sum[i]　　1≤i≤n

目标：max{f[i]}　　1≤i≤n

时间复杂度：O(n^2)

【核心代码】

```
for(int i=1;i<=n;i++) {
       f[i]=t[i].sum;       //至少可以吃掉第 i 个区间的所有窗口
       for(int j=1;j<i;j++)
         if(t[j].r<t[i].l) //保证区间不重叠
           f[i]=max(f[i],f[j]+t[i].sum);//转移
       ans=max(ans,f[i]);  //将最大值保存在 ans 中
}
```

【参考程序】

```
#include <bits/stdc++.h>
#define RI register int
using namespace std;

inline int read()
{
    int x=0,f=1;char s=getchar();
    while(s<'0'||s>'9'){if(s=='-')f=-f;s=getchar();}
    while(s>='0'&&s<='9'){x=x*10+s-'0';s=getchar();}
    return x*f;
}

const int N=1010;
int n;
int f[N];
struct Node{ //结构体记录区间信息
    int l,r;
    int sum;
}t[N];

int cmd(Node a,Node b) //按右端点快排
{
    return a.r<b.r;
}
```

```
int ans;

int main()
{
    freopen("hunger.in","r",stdin);
        freopen("hunger.out","w",stdout);
    n=read();
    for(RI i=1;i<=n;i++)
        t[i].l=read(),t[i].r=read(),t[i].sum=t[i].r-t[i].l+1;
    sort(t+1,t+1+n,cmd);     //快排
    for(RI i=1;i<=n;i++)
    {
        f[i]=t[i].sum;        //至少可以吃掉第 i 个区间的所有窗口
        for(RI j=1;j<i;j++)
            if(t[j].r<t[i].l) //保证区间不重叠
                f[i]=max(f[i],f[j]+t[i].sum);//转移
        ans=max(ans,f[i]);    //将最大值保存在 ans 中
    }
    printf("%d\n",ans);
    return 0;
}
```