2018夏令营机试

前言

此文件包含以下内容

1. 在第一部分中，我会列出8道机试的提示，以便给想要自己尝试解题的同学一些帮助。
2. 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(我并没有参加过夏令营的机试，仅仅在练习的时候打过代码，确实可能会因为某些数据而WA)
3. 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)
4. Leo同学提出了一些修改建议(群里也可以找到)
5. 一些代码为了排版到同一页而可能被压缩。
6. 注明一点：阅读代码时应该着重于理解整个算法流程，而非死扣所有细节。
7. **非常需要注意的一点**：下列代码有几个并非是最好的答案，可以写得更加简单(取决于题目的数据)，我当时这样写有想要练习的意味在，故仅供参考。

相关的题目自行在群文件中寻找，在此就不复制了。

DaSE
Data Science
&Engineering

2020华师数据学院考研

扫一扫二维码，加入群聊。

2018夏令营机试提示

1. 归并排序
2. 处理文档
3. 关键数值
4. 多少个重复串 暴力猜即可
5. hash table 模拟+链表/数组
6. 三角阵 简单的动态规划+递归求路径
7. 最短步骤 BFS
8. 表达式求值 经典题，栈的应用，逆波兰表达式

2018夏令营机试题解

A-归并排序

你可以使用在归并排序中的归并过程，但利用sort函数则似乎更加方便。

```
#include<iostream>
#include<algorithm>
using namespace std;
int main(){
    int n,m;
    int a[200];
    cin>>n;
    for(int i=0;i<n;i++)
        cin>>a[i];
    cin>>m;
    m+=n;
    for(int i=n;i<m;i++)
        cin>>a[i];
    sort(a,a+m);
    cout<<m;
    for(int i=0;i<m;i++)
        cout<<" "<<a[i];
    cout<<endl;
}
```

B-处理文档

这里是单纯的字符串处理，感觉没啥能说的。注意细节。

```cpp
#include<iostream>
#include<string>
using namespace std;
int main(){
    string s;
    getline(cin,s);
    int cnt=0;
    for(int i=0,state=0;i<s.length();i++){
        if(state==0&&s[i]!=' '){
            state=1;
            cnt++;
        }
        if(state==1&&s[i]==' ')cout<<" ";
        else if(state==1){
            if(s[i]=='D'&&i+3<s.length()&&
                s[i+1]=='a'&&s[i+2]=='S'&&s[i+3]=='E'){
                i+=3;
                cout<<"DaSE";
            }
            else {
                cout<<char(s[i]<='z'&&s[i]>='a'?s[i]:s[i]-'A'+'a');
            }
        }
        if(s[i]==' ')state=0;
    }
    cout<<endl<<cnt<<endl;
}
```

C-关键数值

最大最小值没啥能说的，中位数不过就稍微分类处理一下。 我下面写的稍烦，还是sort一下，找中间的比较好。

```
#include<iostream>
#include<cstring>
using namespace std;
int main(){
    int n;
    int min=200,max=-100;//注释一下，我为什么在很多代码中一些值会比题目中的值大，比如说数组会多声明几个，这里的min取了200而非100，主要原因在于一是你这样做首先不会错，二可以避免一些边界问题，三是某些值打起来手顺。
    int a;
    int c[300];
    memset(c,0,300*sizeof(int));
    cin>>n;
    for(int i=0;i<n;i++){
        cin>>a;
        c[a+100]++;
        if(a<min)min=a;
        if(a>max)max=a;
    }
    cout<<max<<" "<<min<<" ";
    int cnt=0;
    if(n&1==1){//奇数
        for(int i=0;i<300;i++){
            cnt+=c[i];
            if(cnt>=n/2+1){
                cout<<i-100<<".00\n";
                break;
            }
        }
    }
    else{//偶数
        for(int i=0;i<300;i++){
            cnt+=c[i];
            if(cnt>n/2){//中间两个数相等的情况
                cout<<i-100<<".00\n";
                break;
            }
            if(cnt==n/2){//中间两个数不等的情况
                for(int j=i+1;j<300;j++){
                    if(c[j]!=0){
                        printf("%.2f\n",(double)(i+j)/2-100);
                        break;
                }    }break;
}    }    }    }
```

D-多少个重复串

枚举所有可能的长度呗。然后一个个测试。从长度小的开始测试，第一个通过测试的即为所求。

```
#include<iostream>
using namespace std;
int main(){
    char a[1010];
    cin>>a;
    int loop=-1,i,j,n=strlen(a);
    for(loop=n;loop>1;loop--){//loop为循环节个数
        if(n%loop!=0)continue;
        int flag=0;
        for(i=0,j=n/loop;a[j];i++,j++){//测试
            if(a[i]!=a[j]){
                flag=1;
                break;
            }
        }
        if(flag==0){
            a[n/loop]=0;//结尾处断掉，以方便输出
            cout<<loop<<endl<<a<<endl;
            return 0;
        }
    }
    cout<<1<<endl<<a<<endl;

}
```

E-hash table

完全按照题面做就可以了。我这里写了一个链表。(因为当时看错题，所以多了一些不必要的代码)

可以理解一下我这种不用指针的写法，可能对于某些同学来说这种做法比指针更加容易理解(我个人是觉得用这个比较方便)。

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

struct block{
    int v;
    int next;
}b[200];
int used=0;
int h[10]={-1,-1,-1,-1,-1,-1,-1,-1,-1,-1};
void add(int i,int x){
    b[used].v=x;
```

```
    b[used].next=-1;
    if(h[i]==-1){h[i]=used++;return;}
    for(i=h[i];i!=-1;i=b[i].next){
        if(b[i].next==-1){
            b[i].next=used;
            break;
        }
    }
    used++;
}

void show(int R){
    for(int i=0;i<R;i++){
        cout<<i;
        for(int j=h[i];j!=-1;j=b[j].next)
            cout<<" "<<b[j].v;
        cout<<endl;
    }
    for(int i=0;i<20;i++)cout<<"-";
    cout<<endl;
}

void del(int i,int x){
    if(h[i]==-1)return ;
    if(b[h[i]].v==x)h[i]=b[h[i]].next;
    for(i=h[i];b[i].next!=-1;i=b[i].next){
        if(b[b[i].next].v==x){
            b[i].next=b[b[i].next].next;
            return;
}   }   }

int main(){
    int n,R,a;
    cin>>n>>R;
    while(n--){
        cin>>a;add(a%R,a);
    }
    cin>>n;
    while(n--){
        string s;
        cin>>s;
        if(s=="Show")show(R);
        else{
            cin>>a;
            if(s=="Delete")del(a%R,a);
            else add(a%R,a);
        }
    }
}
```

F-三角阵

动态规划：状态转移方程 value(i,j)=max(value(i-1,j),value(i-1,j-1))+h(i,j)

同时记录下，从哪里转移过来的即可。最终根据你最后取的那个位置进行回溯便能找到走的那条路。

```
#include<iostream>
#include<cmath>
using namespace std;
int a[111][111],dp[111][111],from[111][111];
void print(int i,int j){
    if(from[i][j]==-1){
        cout<<"(0,0)";
        return;
    }
    print(i-1,j-from[i][j]);
    cout<<"=>("<<i<<","<<j<<")";
}
int main(){
    int n;
    cin>>n;
    for(int i=0;i<n;i++)
        for(int j=0;j<=i;j++)
            cin>>a[i][j];
    dp[0][0]=a[0][0];
    from[0][0]=-1;
    for(int i=1;i<n;i++){
        dp[i][0]=dp[i-1][0]+a[i][0];
        dp[i][i]=dp[i-1][i-1]+a[i][i];//边界
        from[i][0]=0;from[i][i]=1;
        for(int j=1;j<i;j++){//非边界用上述状态转移方程进行转移
            dp[i][j]=a[i][j]+max(dp[i-1][j],dp[i-1][j-1]);
            from[i][j]=dp[i-1][j]>dp[i-1][j-1]?0:1;
        }
    }
    int max=dp[n-1][0];
    int maxi=0;
    for(int i=0;i<n;i++){
        if(max<dp[n-1][i]){
            max=dp[n-1][i];
            maxi=i;
        }
    }
    cout<<max<<endl;
    print(n-1,maxi);cout<<endl;
}
```

G-最短步骤

BFS，基本就是模板。这种应该属于几乎必考的东西，还是应该非常熟练地掌握的。

```cpp
#include<iostream>
#include<queue>
#define x first
#define y second
using namespace std;
int dx[]={0,0,1,-1},
    dy[]={1,-1,0,0};
int main(){
    int n,m;
    int a[111][111];
    queue<pair<int,int> >q;
    cin>>n>>m;
    for(int i=0;i<n;i++)
        for(int j=0;j<m;j++)
            cin>>a[i][j];
    int Xa,Ya,Xb,Yb;
    char c;
    cin>>Xa>>c>>Ya>>Xb>>c>>Yb;
    if(a[Xa][Ya]==1){
        cout<<"NO\n";
        return 0;
    }
    q.push(make_pair(Xa,Ya));
    a[Xa][Ya]=1;
    while(!q.empty()){
        int x=q.front().x,y=q.front().y;
        q.pop();
        if(x==Xb&&y==Yb){
            cout<<a[x][y]-1<<endl;
            return 0;
        }
        for(int i=0;i<4;i++){
            int nx=x+dx[i],ny=y+dy[i];
            if(nx<0||ny<0||nx>=n||ny>=m||a[nx][ny]!=0)continue;
            a[nx][ny]=a[x][y]+1;
            q.push(make_pair(nx,ny));
        }
    }
    cout<<"NO\n";

}
```

H-表达式求值

我这里写得比较暴力，可以通过定义各个运算符的优先级（需要区分栈内栈外）来简化代码。

具体相关原理，搜索表达式求值或者逆波兰表达式。

```
#include<iostream>
#include<stack>
#include<string>
using namespace std;
int main(){
    string s;
    cin>>s;
    stack<char>s1;
    int c[111]={0};
    int n=0;
    for(int i=0;i<s.length();i++){
        switch(s[i]){
            case '=':
                while(!s1.empty()){
                    if(s1.top()=='+')c[++n]=-1;
                    if(s1.top()=='-')c[++n]=-2;
                    if(s1.top()=='*')c[++n]=-3;
                    if(s1.top()=='/')c[++n]=-4;
                    s1.pop();
                }
                ++n;
                break;
            case '0':case '2':case '4':case '6':case '8':
            case '1':case '3':case '5':case '7':case '9':
                c[n]*=10;//数字，处理数字的情况，存入一个数组待用。
                c[n]+=s[i]-'0';
                break;
            case '+':case'-'://优先级较低，需要把栈内优先级更高的pop掉
                while(!s1.empty()&&(s1.top()=='*'||s1.top()=='/')){
                    c[++n]=(s1.top()=='*'?-3:-4);
                    s1.pop();
                }
                s1.push(s[i]);
                c[++n]=0;
                break;
            case '*':case '/':case '('://优先级最高，直接放入
                s1.push(s[i]);
                if(s[i]!='(')
                c[++n]=0;
                break;
            case ')'://优先级最低，但需要把栈内到第一个左括号之前的全部pop
                while(s1.top()!='('){//把符号记成不同的负值存入数组，该数组即代表逆波兰表达式
```

```
                    if(s1.top()=='+')c[++n]=-1;
                    if(s1.top()=='-')c[++n]=-2;
                    if(s1.top()=='*')c[++n]=-3;
                    if(s1.top()=='/')c[++n]=-4;
                    s1.pop();
                }
                s1.pop();
                break;
        }
    }

    stack<double>s2;
    for(int i=0;i<n;i++){//遇到数字就放入栈，遇到符号就从栈中取两个，然后运算后放入栈。
        if(c[i]>=0)s2.push(c[i]);
        else{
            double b=s2.top();s2.pop();
            double a=s2.top();s2.pop();
            switch(c[i]){
                case -1:s2.push(a+b);break;
                case -2:s2.push(a-b);break;
                case -3:s2.push(a*b);break;
                case -4:s2.push(a/b);break;
            }
        }
    }
    printf("%.2f\n",s2.top());

}
```