NOIP2008 提高组解题报告

angwuy

1 word

这道题完全是送分题，只需要直接统计，再判断素数。
参考程序：

```
var
  st:string;
  max,min,i:longint;
  a:array['a'..'z']of longint;
  ch:char;
function fun(n:longint):boolean;
var i:longint;
begin
    if n<2 then begin fun:=false;exit;end;
    for i:=2 to n-1 do
       if n mod i=0 then begin fun:=false;exit;end;
    fun:=true;
end;
begin
    assign(input,'word.in');
    reset(input);
    assign(output,'word.out');
    rewrite(output);
    readln(st);
    fillchar(a,sizeof(a),0);
    for i:=1 to length(st) do
       inc(a[st[i]]);
    max:=0;
    min:=101;
    for ch:='a' to 'z' do
    if a[ch]>0 then
    begin
       if a[ch]>max then max:=a[ch];
       if a[ch]<min then min:=a[ch];
    end;
    if fun(max-min) then
    begin
        writeln('Lucky Word');
```

```
        writeln(max-min);
    end
    else
    begin
        writeln('No Answer');
        writeln(0);
    end;
    close(input);
    close(Output);
end.
```

2 matches

搜索题，由于输入的情况只有 25 种，所以打表也是一种可行的方法。在数据最大时，经过人工和电脑证明是不会到达四位数的，所以可以直接用 O(1000*1000)的搜索算法
参考程序:

```
const
    mat:array[0..9]of longint=(6,2,5,5,4,5,6,3,7,6);
function fun(m:longint):longint;
var t:longint;
begin
    t:=0;
    while m>0 do
    begin
        inc(t,mat[m mod 10]);
        m:=m div 10;
   end;
    fun:=t;
end;
var a:array[0..1000] of longint;
   n,i,j,ans:longint;
begin
    assign(input,'matches.in');
    reset(input);
    assign(output,'matches.out');
    rewrite(output);
    readln(n);
    if n<10 then begin writeln(0);close(output);exit;end;
    a[0]:=6;
    for i:=1 to 1000 do
       a[i]:=fun(i);
    dec(n,4);
    for i:=0 to 1000 do
```

```
    if a[i]<n then
    begin
        for j:=0 to 1000-i do
            if a[i]+a[j]+a[i+j]=n then inc(ans);
    end;
    writeln(ans);
    close(input);
    close(output);
end.
```

3 message

DP 题，两条路线必定一上一下，而且，当到达某一列后，前面对后面的不会有影响，符合动态规划的无后效性，方程如下：
用 dp[I,j,k]表示当到达 I 列时，上路线在 j 行到，下路线在 k 行到的最大值。
另外加一个预处理，sum[I,j1,j2]表示在第 I 列 j1 到 j2 行的数加起来的和。
边界条件：
dp[2,1,k]:=sum[1,1,k];
递推方程:
dp[I,j,k]:=max(dp[I-1,j2,k2]+sum[I-1,j,j2]+sum[I-1,k,k2]) {j<=j2<k<=k2}
答案：
max(dp[m,j,n]+sum[m,j,n])
参考程序：

```
const maxn=10;
var
  a:array[1..maxn,1..maxn]of longint;
  dp,sum:array[1..maxn,1..maxn,1..maxn]of longint;
  n,m,i,j,k,i1,i2,j1,j2,k1,k2:longint;
function max(a,b:longint):longint;
begin
    if a>b then max:=a else max:=b;
end;
begin
    assign(input,'message.in');
    reset(input);
    assign(output,'message.out');
    rewrite(output);
    readln(m,n);
    for i:=1 to m do
    begin
        for j:=1 to n do
          read(a[i,j]);
        readln;
```

```
    end;
    for i:=1 to m do
    begin
        for i1:=1 to n do
        begin
            sum[i,i1,i1]:=a[i,i1];
            for i2:=i1+1 to n do
                sum[i,i1,i2]:=sum[i,i1,i2-1]+a[i,i2];
        end;
    end;
    fillchar(dp,sizeof(dp),255);
    for i:=2 to n do
       dp[2,1,i]:=sum[1,1,i];
    for i:=2 to m-1 do
    for j:=1 to n-1 do
    for k:=j+1 to n do
    if dp[i,j,k]>-1 then
    begin
        for j2:=j to k-1 do
        for k2:=k to n do
           dp[i+1,j2,k2]:=max(dp[i+1,j2,k2],dp[i,j,k]+sum[i,j,j2]+sum[i,k,k2]);
    end;
    k:=0;
    for i:=1 to n-1 do
     k:=max(k,dp[m,i,n]+sum[m,i,n]);
    writeln(k);
    close(input);
    close(output);
end.
```

4.twostack

这道题大概可以归结为如下题意:

有两个队列和两个栈,分别命名为队列 1(q1),队列 2(q2),栈 1(s1)和栈 2(s2).最初的时候,q2,s1 和 s2 都为空,而 q1 中有 n 个数(n<=1000),为 1~n 的某个排列.

现在支持如下四种操作:

a 操作,将 q1 的首元素提取出并加入 s1 的栈顶.

b 操作,将 s1 的栈顶元素弹出并加入 q1 的队列尾.

c 操作,将 q1 的首元素提取出并加入 s2 的栈顶.

d 操作,将 s2 的栈顶元素弹出并加入 q1 的队列尾.

请判断,是否可以经过一系列操作之后,使得 q2 中依次存储着 1,2,3,…,n.如果可以,求出字典序最小的一个操作序列.

这道题的错误做法很多,错误做法却能得满分的也很多,这里就不多说了.直接切入正题,就是即将介绍的这个基于二分图的算法.

注意到并没有说基于二分图匹配,因为这个算法和二分图匹配无关.这个算法只是用到了给一个图着色成二分图.

第一步需要解决的问题是,判断是否有解.

考虑对于任意两个数 q1[i]和 q1[j]来说,它们不能压入同一个栈中的充要条件是什么(注意没有必要使它们同时存在于同一个栈中,只是压入了同一个栈).实际上,这个条件 p 是:存在一个 k,使得 i<j<k 且 q1[k]<q1[i]<q1[j].

首先证明充分性,即如果满足条件 p,那么这两个数一定不能压入同一个栈.这个结论很显然,使用反证法可证.
假设这两个数压入了同一个栈,那么在压入 q1[k]的时候栈内情况如下:
…q1[i]…q1[j]…
因为 q1[k]比 q1[i]和 q1[j]都小,所以很显然,当 q1[k]没有被弹出的时候,另外两个数也都不能被弹出(否则 q2 中的数字顺序就不是 1,2,3,…,n 了).
而之后,无论其它的数字在什么时候被弹出,q1[j]总是会在 q1[i]之前弹出.而 q1[j]>q1[i],这显然是不正确的.

接下来证明必要性.也就是,如果两个数不可以压入同一个栈,那么它们一定满足条件 p.这里我们来证明它的逆否命题,也就是"如果不满足条件 p,那么这两个数一定可以压入同一个栈."
不满足条件 p 有两种情况:一种是对于任意 i<j<k 且 q1[i]<q1[j],q1[k]>q1[i];另一种是对于任意 i<j,q1[i]>q1[j].
第一种情况下,很显然,在 q1[k]被压入栈的时候,q1[i]已经被弹出栈.那么,q1[k]不会对 q1[j]产生任何影响(这里可能有点乱,因为看起来,当 q1[j]<q1[k]的时候,是会有影响的,但实际上,这还需要另一个数 r,满足 j<k<r 且 q1[r]<q1[j]<q1[k],也就是证明充分性的时候所说的情况…而事实上我们现在并不考虑这个 r,所以说 q1[k]对 q1[j]没有影响).
第二种情况下,我们可以发现这其实就是一个降序序列,所以所有数字都可以压入同一个栈.
这样,原命题的逆否命题得证,所以原命题得证.

此时,条件 p 为 q1[i]和 q1[j]不能压入同一个栈的充要条件也得证.

这样,我们对所有的数对(i,j)满足 1<=i<j<=n,检查是否存在 i<j<k 满足 p1[k]<p1[i]<p1[j].如果存在,那么在点 i 和点 j 之间连一条无向边,表示 p1[i]和 p1[j]不能压入同一个栈.此时想到了什么?那就是二分图~
二分图的两部分看作两个栈,因为二分图的同一部分内不会出现任何连边,也就相当于不能压入同一个栈的所有结点都分到了两个栈中.
此时我们只考虑检查是否有解,所以只要 o(n)检查出这个图是不是二分图,就可以得知是否有解.

此时,检查有解的问题已经解决.接下来的问题是,如何找到字典序最小的解.
实际上,可以发现,如果把二分图染成 1 和 2 两种颜色,那么结点染色为 1 对应当前结点被压入 s1,为 2 对应被压入 s2.为了字典序尽量小,我们希望让编号小的结点优先压入 s1.
又发现二分图的不同连通分量之间的染色是互不影响的,所以可以每次选取一个未染色的编号最小的结点,将它染色为 1 并从它开始 dfs 染色,直到所有结点都被染色为止.这样,我们就得到了每个结点应该压入哪个栈中.接下来要做的,只不过是模拟之后输出序列啦~
还有一点小问题,就是如果对于数对(i,j),都去枚举检查是否存在 k 使得 p1[k]<p1[i]<p1[j]的话,那么复杂度就升到了 o(n^3).
解决方法就是,首先预处理出数组 b,b[i]表示从 p1[i]到 p1[n]中的最小值.接下来,只需要枚举所有数对(i,j),检查 b[j+1]是否小于 p1[i]且 p1[i]是否小于 p1[j]就可以了.

附代码(除去注释不到 100 行),带注释.代码中的 a 数组对应文中的队列 p1.
已经过掉所有标准数据,以及 5 7 2 4 1 6 3 这组让很多贪心程序挂掉的数据~

```
#include <iostream>

using namespace std;

const int nn = 1002, mm = nn * 2, inf = 1000000000;
int n, tot, now;
int a[nn], b[nn], head[nn], color[nn];
int adj[mm], next[mm];
int stack[3][nn];
bool result;

void addedge(int x, int y) //加边
{
    ++ tot;
    adj[tot] = y;
    next[tot] = head[x];
    head[x] = tot;
}

bool dfs(int i) //dfs 染色,检查图是否是二分图的经典算法
{
    int temp = head[i];
    while (temp) //邻接表,检查每一条边
    {
        if (! color[adj[temp]]) //如果与当前结点的结点还未染色
        {
            color[adj[temp]] = 3 - color[i]; //进行染色
            dfs(adj[temp]); //dfs
        }
        if (color[adj[temp]] == color[i]) return false;
            //如果两个相邻结点染色相同,说明此图不是二分图,返回无解
        temp = next[temp];
    }
    return true;
}

int main()
{
    freopen("twostack.in", "r", stdin);
    freopen("twostack.out", "w", stdout);

    //输入
```

```
scanf("%d", n);
for (int i = 1; i <= n; ++ i) scanf("%d", a[i]);

//预处理 b 数组
b[n + 1] = inf;
for (int i = n; i >= 1; -- i) b[i] = min(b[i + 1], a[i]); //"min" in stl

//枚举数对(i,j)并加边
tot = 0;
for (int i = 1; i <= n; ++ i)
    for (int j = i + 1; j <= n; ++ j)
        if (b[j + 1] < a[i] & a[i] < a[j])
        {
            addedge(i, j);
            addedge(j, i);
        }

//dfs 染色
memset(color, 0, sizeof(color));
result = true;
for (int i = 1; i <= n; ++ i) //每次找当前未染色的编号最小的结点,并染颜色 1
    if (! color[i]) //当前位置尚未被染色
    {
        color[i] = 1;
        if (! dfs(i)) //染色时出现矛盾,此时图不是一个二分图,即无法分配到两个栈中
        {
            result = false; //记录无解
            break;
        }
    }

if (! result) //无解
    printf("0");
else //有解
{
    //模拟求解
    now = 1;
    for (int i = 1; i <= n; ++ i)
    {
        //将当前数字压入对应的栈
        if (color[i] == 1)
            printf("a ");
        else
            printf("c ");
```

```
            stack[color[i]][0] ++;
            stack[color[i]][stack[color[i]][0]] = a[i]; //this will work even if stack[1][0] = 0

            //循环检查,如果可以的话就从栈顶弹出元素
            while (stack[1][stack[1][0]] == now || stack[2][stack[2][0]] == now)
            {
                if (stack[1][stack[1][0]] == now)
                {
                    printf("b ");
                    stack[1][0] --;
                }
                else if (stack[2][stack[2][0]] == now)
                {
                    printf("d ");
                    stack[2][0] --;
                }
                now ++;
            }
        }
    }
}
```