

ACM 算法与微应用实验室 2021 年 11 月月赛题解

2021 年 12 月 1 日

题目概览

题目编号	题目名称	命题人	做法
A	克隆干员	AgOH	模拟
B	中转站	AgOH&Tifa	前缀和
C	三斜求积术	AgOH	模拟
D	子树大小	AgOH	dfs
E	pro5	AgOH	模拟
F	Go	AgOH	并查集

鸣谢

感谢Tifa大佬参与本次比赛的出题工作。

A. 克隆干员

做法

模拟过程。比较简便的做法如下：

首先我们先将干员的四种不同朝向时的状况分别保存于数组中（也就是先把干员旋转好），然后开出一个初始值为 0 的数组用来表示战场，最后每输入一个点后就向这个战场数组中涂色（填 1）即可。

标程

```
#include <iostream>
#include <string>
using namespace std;
string s[10];
int fw[5][10][10],a[15][15]; // fw[i] i从1到4分别保存干员朝向上下左右时的攻击范围；a为战场
int main()
{
    for(int i=0;i<7;i++) cin>>s[i]; // 用string简单读入
    for(int i=1;i<=7;i++) for(int j=1;j<=7;j++) fw[1][i][j]=s[i-1][j-1]-'0'; // 朝上
    for(int i=1;i<=7;i++) for(int j=1;j<=7;j++) fw[2][i][j]=fw[1][8-i][j]; // 朝下
    for(int i=1;i<=7;i++) for(int j=1;j<=7;j++) fw[3][i][j]=fw[1][j][i]; // 朝左
    for(int i=1;i<=7;i++) for(int j=1;j<=7;j++) fw[4][i][j]=fw[1][8-j][i]; // 朝右
    int n;
    cin>>n;
    for(int t=0;t<n;t++)
    {
        int x,y,f;
        cin>>x>>y>>f;
        for(int i=x-3;i<=x+3;i++) // 因为刚才已经旋转好了所以直接涂色即可
            for(int j=y-3;j<=y+3;j++)
                if(!(i<1||i>10||j<1||j>10)) // 如果没涂到战场范围之外的话
                    a[i][j]=fw[f][i+4-x][j+4-y]; // 就涂色即可（若之前是0我们要把它变成1，若之前是1则还是1，用或即可）
    }
    for(int i=1;i<=10;i++)
    {
        for(int j=1;j<=10;j++)
            cout<<a[i][j];
        cout<<endl;
    }
    return 0;
}
```

B. 中转站

做法

本题改编自 [CSP-S2019 江西] 和积和。

本题有多种解法，这里提供一种效率不高但比较好想的前缀和做法。

设区间限制为 $[l, r]$ 时，玩家玩游戏的会获得的分数为 $S(l, r)$ ，即：

$$S(l, r) = \sum_{i=l}^r \sum_{j=l}^r a_i b_j$$

设我们要求解的答案——所有可能的区间约束下每次你能获得的分数的总和——为 ans ，有：

$$ans = \sum_{l=1}^n \sum_{r=l}^n S(l, r)$$

让我们来尝试着化简一下这个式子。首先，根据分配律，有：

$$S(l, r) = \sum_{i=l}^r a_i \times \sum_{j=l}^r b_j$$

设数列 $\{a_n\}$ 的前缀和数列为 $\{pre_n^a\}$ ，数列 $\{b_n\}$ 的前缀和数列为 $\{pre_n^b\}$ ，有：

$$\begin{aligned} S(l, r) &= (pre_r^a - pre_{l-1}^a) \times (pre_r^b - pre_{l-1}^b) \\ &= pre_r^a pre_r^b - pre_r^a pre_{l-1}^b - pre_{l-1}^a pre_r^b + pre_{l-1}^a pre_{l-1}^b \end{aligned}$$

设 $pre_i^{ab} = pre_i^a pre_i^b$ ，有：

$$S(l, r) = pre_r^{ab} + pre_{l-1}^{ab} - pre_r^a pre_{l-1}^b - pre_{l-1}^a pre_r^b$$

把 $\sum_{r=l}^n$ 拆进去，有：

$$\begin{aligned} ans &= \sum_{l=1}^n \left(\sum_{r=l}^n pre_r^{ab} + (n-l+1)pre_{l-1}^{ab} - \sum_{r=l}^n pre_r^a pre_{l-1}^b - \sum_{r=l}^n pre_{l-1}^a pre_r^b \right) \\ &= \sum_{l=1}^n \left(\sum_{r=l}^n pre_r^{ab} + (n-l+1)pre_{l-1}^{ab} - pre_{l-1}^b \sum_{r=l}^n pre_r^a - pre_{l-1}^a \sum_{r=l}^n pre_r^b \right) \end{aligned}$$

设数列 $\{pre_n^a\}$ 的前缀和数列为 $\{pre_n^{pre^a}\}$ ，数列 $\{pre_n^b\}$ 的前缀和数列为 $\{pre_n^{pre^b}\}$ ，数列 $\{pre_n^{ab}\}$ 的前缀和数列为 $\{pre_n^{pre^{ab}}\}$ ，上式即化为：

$$\sum_{l=1}^n \left((pre_n^{pre^{ab}} - pre_{l-1}^{pre^{ab}}) + (n-l+1)pre_{l-1}^{ab} - (pre_n^{pre^a} - pre_{l-1}^{pre^a}) pre_{l-1}^b - (pre_n^{pre^b} - pre_{l-1}^{pre^b}) pre_{l-1}^a \right)$$

数列 $a\{n\}$ 和数列 $b\{n\}$ 是已知的，我们可以 $O(n)$ 预处理出 $\{pre_n^a\}$ 与 $\{pre_n^b\}$ ，然后我们又可以 $O(n)$ 预处理出 $\{pre_n^{pre^a}\}$ 、 $\{pre_n^{pre^b}\}$ 、 $\{pre_n^{ab}\}$ 。这样上式后半部分的一大坨就可以 $O(1)$ 得出结果了，总时间复杂度 $O(n)$ 。

在实现的过程中需要注意本来原式只有加法，无论怎样式子中都不会出现负数。但转化为前缀和后式子中出现了减法，在值取模后相减有可能出现负数，需要把负数转回整数再继续运算。

标程

```
#include <iostream>
using namespace std;
const int MAXN = 5e5+5;
const int MOD = 1e9+7;
typedef long long ll;
ll a[MAXN], b[MAXN];
ll prea[MAXN], preb[MAXN], preab[MAXN];
ll preprea[MAXN], prepreb[MAXN], prepreab[MAXN];
int n;
inline void calcPre(ll a[], ll pre[]) // 计算a数组的前缀和存于pre数组中
{
    for(int i=1; i<=n; i++)
        pre[i] = (a[i] + pre[i-1]) % MOD;
}
ll solve()
{
    ll sum = 0;
    for(int l=1; l<=n; l++)
    {
        sum = (sum + prepreab[l] - prepreab[l-1] + MOD) % MOD; // 第一项
        sum = (sum + (n-l+1)*preab[l-1] % MOD) % MOD; // 第二项
        sum -= (preprea[l] - preprea[l-1]) * preb[l-1] % MOD; // 第三项
        sum = (sum + MOD) % MOD; // 有可能出现负数，转为正数
        sum -= (prepreb[l] - prepreb[l-1]) * prea[l-1] % MOD; // 第四项
        sum = (sum + MOD) % MOD; // 有可能出现负数，转为正数
    }
    return sum;
}
int main()
{
    cin >> n;
    for(int i=1; i<=n; i++) cin >> a[i];
    for(int i=1; i<=n; i++) cin >> b[i];
    calcPre(a, prea);
    calcPre(b, preb);
    for(int i=1; i<=n; i++) preab[i] = prea[i] * preb[i] % MOD;
    calcPre(prea, preprea);
    calcPre(preb, prepreb);
    calcPre(preab, prepreab);
    cout << solve() << endl;
    return 0;
}
```

C. 三斜求积术

做法

签到题，按照题目说明中给出的海伦公式进行模拟即可。

标程

```
#include <cstdio>
#include <cmath>
int main()
{
    int t;
    scanf("%d", &t);
    while(t--)
    {
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        double p = (a+b+c)/2.0;
        double s = sqrt(p*(p-a)*(p-b)*(p-c));
        printf("%.2f\n", s);
    }
    return 0;
}
```

D. 子树大小

做法

求各子树大小是树上的经典基操，一遍 dfs 即可解决。

标程

```
#include <iostream>
using namespace std;
const int maxn = 1e4+5;
struct E { int to,next; } Edge[maxn<<1]; // 链式前向星，因为是树所以开2倍maxn大小即可
int tot,Head[maxn];
inline void AddEdge(int u,int v) { Edge[tot]={v,Head[u]}; Head[u]=tot++; }
int siz[maxn];
void dfs(int u,int f)
{
    siz[u]=1; // 每个结点的大小等于自己的所有子树的大小之和加1（因为还包括结点自己）
    for(int i=Head[u];~i;i=Edge[i].next)
    {
        int v = Edge[i].to;
        if(v==f) continue; // 防止走向父亲
        dfs(v,u); // 计算v子树大小
        siz[u]+=siz[v]; // 在u子树大小中加上v子树大小
    }
}
#include <cstring>
int main()
{
    memset(Head,-1,sizeof(Head)); // 链式前向星-1写法，将Head数组初始化为-1
    int n; cin>>n;
    for(int i=1,u,v;i<n;i++) // 读入数据，注意正反加两条边
        cin>>u>>v, AddEdge(u,v), AddEdge(v,u);
    for(int rt=1;rt<=n;rt++)
    {
        dfs(rt, 0);
        for(int i=1;i<=n;i++)
            cout<<siz[i]<<' ' ;
        cout<<endl;
    }
    return 0;
}
```

E. pro5

做法

标程

F. Go

做法

标程