Fyq训练记录

Chiitoitsu

<a href="I-Chiitoitsu_"蔚来杯"2022牛客暑期多校训练营1 (nowcoder.com)">题目链接

题目大意

初始13张牌，问凑出7个对子的期望步数是多少

题解

期望DP，令$f[i,j]$表示当前i个对子牌库还有j张牌时距离胡牌的期望步数，由于题目规定了始牌中不可能出现三张相同的牌，因此只会出现单张与对子的情况，若有i个对子，就有$13-2*i$张单牌，接下来考虑抽牌，抽牌导致的结果可能是对子增加，也可能对子不变，但牌库总牌数一定减少，那如何计算对子增加的概率呢?

我们发现，若想要对某张已有单牌凑出对子，当前牌库中该单排的数量一定是三张，想想就知道了，四张？那手牌不可能有，两张？那已经抽到了，又不可能弃掉对子，一张？那显然同两张的情况一致。$f[i][j]$转移到$f[i+1][j]$的概率就知道了，就是下一次抽到已有单牌的概率，$f[i][j]$转移到$f[i][j-1]$的概率和凑成对子互补，具体转移方程如下:

- $f[i][j]=\frac{(13-2\times i)\times3}{j}f[i+1][j-1]+(1-\frac{(13-2\times i)\times3}{j})f[i][j-1]$

滚动数组可以优化成一维

代码

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int mod = 1e9 + 7;
const int l = 34 * 4 - 13;
int f[10][150];
int qpow(int a, int n) {
    int res = 1;
    while(n) {
        if(n & 1) res = res * a % mod;
        a = a * a % mod;
        n >>= 1;
    }
    return res;
}

// int dfs(int k, int last) {
//     if(f[k][last] != -1) return f[k][last];
//     if(k == 7) return f[k][last] = 0;
// //     if(last < (13 - 2 * k) * 3) return f[k][last] = 0;
// //     if(!last) return 0;
//     int &v = f[k][last];
```

```
//     v = 0;
//     int p = (13 - 2 * k) * 3 % mod * qpow(last, mod - 2) % mod;
//     v = (p % mod * dfs(k + 1, last - 1) % mod + (1 - p + mod) % mod * dfs(k,
last - 1) % mod) % mod + 1;
//     v = (v % mod + mod) % mod;
//     return v;
// }

void init() {
    memset(f, 0, sizeof(f));
    for(int i = 6; i >= 0; i -- ) {
        for(int j = 0; j <= l; j ++ ) {
            if((13 - 2 * i) > j) continue;
            int p = (13 - 2 * i) * 3 % mod * qpow(j, mod - 2) % mod;
            f[i][j] = p * f[i + 1][j - 1] % mod + (1 - p) * f[i][j - 1] % mod +
1;
            f[i][j] = (f[i][j] % mod + mod) % mod;
        }
    }
}

signed main() {
    int t;
    cin >> t;
    int kk = 0;
    memset(f, -1, sizeof(f));
//     dfs(0, l);
    init();
//     for(int i = 0; i <= 6; i ++ ) {
//         cout << f[i][l] << endl;
//     }
    while(t -- ) {
        getchar();
        kk ++ ;
        map<pair<char, char>, int> cnt;
        int d;
        char c;
        int a = 0, b = 0;
        for(int i = 1; i <= 13; i ++ ) {
            scanf("%c%c", &d, &c);
//             printf("%c%c", d,c);
            cnt[{d, c}] ++ ;
            if(cnt[{d, c}] == 2) {
                a ++ ;
            }
        }
//         printf("%lld\n", a);
        printf("Case #%lld: %lld\n", kk, f[a][l]);
    }
    return 0;
}
```

Link with Game Glitch

<a href="D-Link with Game Glitch_"蔚来杯"2022牛客暑期多校训练营2 (nowcoder.com)">题目链接

题目大意

给出n个食材，m个食谱，为防止无限创造食物，使食谱的产出乘以系数w，问最大的w

题解

最大最小考虑二分，现在目标是求出check函数，想要使食物不被无限制造，问题等价于食谱图中不存在边权乘积大于1的环，但这样写check函数免不了会爆double，我们考虑转换，将边权都取log的话问题似乎更简单了，转化成了食谱图中不存在正环，套用spfa判正环模板即可

代码

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2005;
const double eps = 1e-8;
int head[N], e[N], ne[N], idx = 0;
int f[N];
double dis[N];
double w[N];
bool vis[N];
int n, m;
queue<int> q;
void add(int a, int b, double c) {
    e[idx] = b;
    w[idx] = c;
    ne[idx] = head[a];
    head[a] = idx ++ ;
}
bool check(double x) {
    memset(vis, false, sizeof(vis));
    memset(f, 0, sizeof(f));
    while(q.size()) q.pop();
    for(int i = 1; i <= n ; i ++ ) {
        vis[i] = true;
        dis[i] = 0;
        q.push(i);
    }
    while(q.size()) {
        int pos = q.front();
        q.pop();
        vis[pos] = false;
        for(int i = head[pos]; i != -1; i = ne[i]) {
            int j = e[i];
            if(dis[j] < dis[pos] + w[i] + x) {
                dis[j] = dis[pos] + w[i] + x;
                f[j] = f[pos] + 1;
                if(f[j] >= n) return false;
                if(!vis[j]) {
                    q.push(j);
                    vis[j] = true;
                }
            }
        }
    }
    return true;
```

```
}
int main() {
    cin >> n >> m;
    memset(head, -1, sizeof(head));
    for(int i = 1; i <= m; i ++ ) {
        int a, b;
        double x, y;
        cin >> x >> a >> y >> b;
        add(a, b, log(y * 1.0 / x));
    }

    double l = 0, r = 1.0;
    while(l < r - eps) {
        double mid = (l + r) / 2.0;
        if(check(log(mid)))l = mid;
        else r = mid;
    }
    printf("%.8lf", l);
    return 0;
}
```

Link with Bracket Sequence I

<a href="K-Link with Bracket Sequence I "蔚来杯"2022牛客暑期多校训练营2 (nowcoder.com)">题目链接

题目大意

给出括号序列s，求出长度为m能变成s的合法括号序列p的个数

题解

本题实际是求最长公共子序列是s的方案数，考虑DP，令$f[i,j,k]$表示前i个字符中与原串s的最长公共子序列为j的序列，且左括号数量比右括号多k个的方案数，状态转移如下:

- $s[j+1]=左括号$
  - 放左括号：$f[i,j+1,k+1]=f[i,j+1,k+1]+f[i-1,j,k]$
  - 放右括号：$k \ge 1时,f[i,j,k - 1]=f[i,j,k-1]+f[i-1,j,k]$
- $s[j+1]=右括号$
  - 放左括号：$f[i,j,k+1]=f[i,j,k+1]+f[i-1,j,k]$
  - 放右括号：$k\ge 1时,f[i,j+1,k-1]=f[i,j+1,k-1]+f[i-1,j,k]$
- $s匹配完毕$
  - 放左括号：$f[i,j,k+1]=f[i,j,k+1]+f[i-1,j,k]$
  - 放右括号：$k\ge 1时,f[i,j,k-1]=f[i,j,k-1]+f[i-1,j,k]$

代码

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <queue>
#include <set>
#include <vector>
#include <cmath>
```

```
using namespace std;
typedef pair<int, int> PII;
const int N = 210 , INF = 0x3f3f3f3f, mod = 1e9 + 7;
int f[N][N][N];
int n, m, a[N];
string s;

void add(int& x, int v) {
    x += v;
    if (x >= mod) x -= mod;
}

void solve()
{
    cin >> n >> m >> s;
    s = " " + s;
    memset(f, 0, sizeof f);
    f[0][0][0] = 1;

    for(int i = 1 ; i <= m ; i ++ )
    {
        for(int j = 0 ; j <= n ; j ++ )
        {
            for(int k = 0 ; k <= m ; k ++ )
            {
                if(s[j + 1] == '(') // i位置放(
                {
                    f[i][j + 1][k + 1] = (f[i][j + 1][k + 1] + f[i - 1][j][k]) %
mod ;
                    if(k) f[i][j][k - 1] = (f[i - 1][j][k] + f[i][j][k - 1]) %
mod;
                }
                else if(s[j + 1] == ')') // i位置放)
                {
                    f[i][j][k + 1] = (f[i - 1][j][k] + f[i][j][k + 1]) % mod;
                    if(k) f[i][j + 1][k - 1] = (f[i - 1][j][k] + f[i][j + 1][k -
1]) % mod;
                }
                else // s串匹配完毕 但是长度不够
                {
                    f[i][j][k + 1] = (f[i][j][k + 1] + f[i - 1][j][k]) % mod;
                    if(k) f[i][j][k - 1] = (f[i][j][k - 1] + f[i - 1][j][k]) %
mod;
                }
            }
        }
    }

    cout << f[m][n][0] << endl;
}

signed main()
{
    ios::sync_with_stdio(0),cin.tie(0);
    int T = 1;
    cin >> T;
    while(T -- ) solve();
```

```
    return 0;
}
```

Task Computing

<a href="A-Task Computing_"蔚来杯"2022牛客暑期多校训练营4 (nowcoder.com)">题目链接

题目大意

给出一个长度为n的数组，要求选m个数，并任意排序，最大化$\sum_{i=1}^mw_{ai}*\prod_{j=0}^{i-1}p_{aj}$，每个物品价值为$w_i$

题解

我们将这个贡献的式子展开，$原式=w_{a_1}*p_{a_0}+w_{a_2}*p_{a_0}*p_{a_1}+w_{a_3}*p_{a_0}*p_{a_1}*p_{a_2}...$，从前往后填贡献是这样计算的，那我们换一个方向呢？以前三项为例，计算顺序就变成了$((w_{a_3}*p_{a_2}+w_{a_2})*p_{a_1}+w_{a_1})*p_{a_0}$，可以发现，若当前贡献是$pre$，则每次贡献变化都是先乘上$w$，再加上$p$，于是我们就可以试着写一下cmp函数，对于$x,y$，再从后往前填数的情况向下，想让x优先于y被选择应该满足什么条件呢?

有了排序函数后，就可以通过DP求解了，这里可以将cmp函数反一下，让dp更好写

令$f[i,j]$表示考虑前i个数取j的情况中贡献的最大值

- $f[i,j]=max(f[i-1,j],f[i-1,j-1]*p_i+w_i$

代码

```
#include <iostream>
#include <cstring>
#include <algorithm>

using namespace std;
const int N = 2e5 + 10, INF = 0x3f3f3f3f;
struct Node
{
    double w;
    double p;
    bool operator < (const Node &t) const
    {
        return w + p * t.w < t.w + t.p * w;
    }
}a[N];
int n, m;
double f[N][25];
void solve()
{
    cin >> n >> m;
    for (int i = 1; i <= n; i ++) cin >> a[i].w;
    for (int i = 1; i <= n; i ++)
    {
        cin >> a[i].p;
        a[i].p /= 10000;
    }
    sort(a + 1, a + 1 + n);
```

```
    for (int i = 1; i <= n ; i ++ )
    {
        for (int j = 1; j <= min(i, m); j ++)
        {
            f[i][j] = max(f[i - 1][j], a[i].w + a[i].p * f[i - 1][j - 1]);
        }
    }
    printf("%.12lf\n", f[n][m]);
}

int main()
{
    int T = 1;
    // cin >> T;
    while(T -- ) solve();
    return 0;
}
```

Wall Builder II

<a href="H-Wall Builder II "蔚来杯"2022牛客暑期多校训练营4 (nowcoder.com)">题目链接

题目大意

给出n种砖块，砖块高度为1，宽度为$1-n$，面积为i的砖块数量有$n-i+1$个，问砌成周长最小的矩形的方案并输出

题解

简单贪心，要使砌成的矩形周长最小，不难想到应该越像个正方形越小，接下来考虑怎么填，我们发现长度越小的灵活度越高，因此优先考虑放大的，一块一块填就行

代码

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int N = 105;
int cnt[N];
struct node_ {
    int x1, y1, x2, y2;
};

vector<node_> res[N];
int tmp[N * N];
int find(int l, int r, int x) {
    while(l < r) {
        int mid = (l + r + 1) >> 1;
        if(tmp[mid] > x) r = mid - 1;
        else l = mid;
    }
    return r;
}
signed main() {
    cin.tie(0);
    cout.tie(0);
```

```
    ios::sync_with_stdio(0);
    int t;
    cin >> t;
    while(t -- ) {
        memset(cnt, 0, sizeof(cnt));
        //memset(tmp, 0, sizeof(tmp));
        int n, sumv = 0;
        cin >> n;
        int kk = 0;
        for(int i = 1, j = n; i <= n; i ++, j -- ) {
            sumv += i * j;
            cnt[i] = j;
            res[i].clear();
            for(int k = 1; k <= j; k ++ ) tmp[++ kk] = i;
        }
//         for(int i = 1; i <= kk; i ++ ) cout << tmp[i] << " ";
//         cout << '\n';
        int maxx = 1;
        for(int i = 1; i <= sumv / i; i ++ ) {
            if(sumv % i == 0) {
                maxx = max(maxx, i);
            }
        }
//         sort(tmp + 1, tmp + kk + 1);
        int maxlen = sumv / maxx;
        for(int i = 1; i <= maxx; i ++ ) {
            int nowlen = 0;
            int pos = n;
            while(nowlen != maxlen) {
                for(int j = n; j >= 1; j -- ) {
                    if(cnt[j] && j <= maxlen - nowlen) {
                        pos = j;
                        break;
                    }
                }
                nowlen += pos;
                cnt[pos] -- ;
                res[pos].push_back({nowlen - pos, i - 1, nowlen, i});
                if(nowlen == maxlen) break;
            }
        }
        cout << 2 * (maxlen + maxx) << '\n';
        for(int i = 1; i <= n; i ++ ) {
            for(auto k : res[i]) {
                cout << k.x1 << " " << k.y1 << " " << k.x2 << " " << k.y2 <<
'\n';
            }
        }
    }
    return 0;
}
```

Array

<a href="A-Array_"蔚来杯"2022牛客暑期多校训练营6 (nowcoder.com)">题目链接

题目大意

给出长度为n的数组a，满足$\sum_{i=1}^n\frac{1}{a[i]}\le\frac{1}{2}$，构造数列$c$，满足对于$c$的循环数列$b$，$b[i]=c[i\%m]$，满足$b$中每$a[i]$个数中存在$i$

题解

非正解，我们发现$\sum_{i=1}^n\frac{1}{a[i]}\le\frac{1}{2}$的条件，$a[i]$不会很小，于是可以考虑贪心着填，当遇到已经填到的位置就往前填，为防止首位冲突，我们再最后也尽可能填

代码

```
#include<bits/stdc++.h>
using namespace std;
const int N = 2e6 + 5;
int a[N];
int b[N];
struct node_ {
    int a, pos;
}node[N];

bool cmp(node_ x, node_ y) {
    return x.a < y.a;
}
int main() {
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i ++ ) {
        scanf("%d", &node[i].a);
        node[i].pos = i;
    }

    sort(node + 1, node + n + 1, cmp);
    int s = 1;
    for(int i = 1; i <= n; i ++ ) {
        while(b[s]) s ++ ;
        for(int j = s; ;j += node[i].a) {
            while(b[j] || j > 1000000) j -- ;
            b[j] = node[i].pos;
            if(1000000 - j + s <= node[i].a) break;
        }
        s ++ ;
    }
    printf("1000000\n");
    for(int i = 1; i <= 1000000; i ++ ) {
        if(!b[i])  printf("1 ");
        else printf("%d ", b[i]);
    }
    return 0;
}
```

Longest Common Subsequence

<a href="F-Longest Common Subsequence "蔚来杯"2022牛客暑期多校训练营8 (nowcoder.com)">
题目链接

题目大意

根据二次函数的构造方式，求出s和t的最长公共子序列长度

题解

由于两者的构造方式相同，因此只需找到一个相同的值，后面必然都是相同的，利用map记录值在第一个数组中出现的位置即可

代码

```
#include<bits/stdc++.h>
#define int long long

using namespace std;

const int N = 1995781;

int n;
int m;
int x, a, b, c;
int p;


int lib[N];
int arr[N];
int dp[N];

int s[N];
int t[N];
int len = 0;

int tmparr[N];

bool vis[N];
int h[N];
int pos[N];

int find(int x)
{
    int k = (x % N + N) % N ;

    while(vis[k] && h[k] != x)
    {
        k ++ ;
        if(k == N ) k = 0;
    }

    return k;
}


void solve() {
    memset(vis, false, sizeof(vis));
    int temp;
    int len = 1;
    cin >> n >> m >> p >>  x >> a >> b >> c;
```

```
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        x = (a * x % p * x % p + b * x % p + c) % p;
        int pp = find(x);
//         cout << pp << " ";
        if(!vis[pp]) {
            pos[pp] = i;
            h[pp] = x;
            vis[pp] = true;
        }
    }
//     cout << '\n';
    for (int i = 1; i <= m; i++) {
        x = (a * x % p * x % p + b * x % p + c) % p;
        int pp = find(x);
//         cout << pp << " ";
        if (vis[pp]) {
            ans = max(ans, min(n - pos[pp] + 1, m - i + 1));
        }
    }
    cout << ans << '\n';
}




signed main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int _;
    //_ = 1;
    cin >> _;
    while (_--) {
        solve();
    }
    return 0;
}
```