2023.3.12

不让古人，是谓有志；不让今人，是谓无量。

2023 3月第二周

总体上，依然保持之前的规划方向来学习。
对策略的调整是：每天早上开始就找两道自己力所能及得1500---1600范围内得题。来保持自己得信心以及状态。

算法学习：

1. 单调队列优化.md

刷题

1. atcoder
    1. E RLE.md (dp)
2. luogu(都是水题。)
    1. P1103 书本整理.md
    2. P1564 膜拜.md
    3. P1681 最大正方形II.md
    4. HXY和序列.md

contest

1. 上海理工大学天梯赛vp：
    1. Setsuna的K数列.md
    2. 叠硬币.md
    3. Wiki with Fake AKGPLT.md
2. div2 857
    1. D. Buying gifts.md
    2. C. The Very Beautiful Blanket.md （未补好）

---

atcoder

E.RLE

Editorial - Monoxer Programming Contest 2022（AtCoder Beginner Contest 249）

solve

状态定义：
定义状态$f_{i,j}$表示S串长度为i， T串长度为j方案个数。
状态转移
由小更新大的贡献转移角度：

1. $dp_{i,j}$枚举k。然后贡献后方。
   复杂度分析
   枚举i， j总体的复杂度应该是：$O(N^3)$

考虑优化

关注从大到小直接统计转移：

当我们枚举计算某个状态时，发现贡献来源于几种不同二维数组中某一段之和。因此可以使用前缀和优化。具体的实现方案是：分析枚举在后面拼接不同的字母长度。将其对k的影响分类成三种情况,分别为1 10 100 1000 10000 .假设拼接的的长度为1....9。对于j维度关注关注j - 2段的下标。 假设拼接长度为 10 ....99 考虑 j - 3段的前缀。依此类推。

$$\begin{aligned} dp_{i,j}+=(sum[i-1][j-2]-sum[i-10][j-2])\times 25; \\ dp_{i,j}+=(sum[i-10][j-3]-sum[i-1000][j-3])\times 25 \\ \text{推广到第}10^k\text{次种长度}dp_{i,j}+=(sum[i-10^k]-sum[i-10^{k+1}])\times 25 \end{aligned} \tag{1}$$

初始化

考虑那些全放同一个字母的各种方案作为最小状态：

code

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 3E3 + 10;

ll dp[N][N];
int lg(int x) {
    int res = 1;
    while (x) {
        x /= 10;
        res++;
    }
    return res;//在基础上加1的刚刚好。
}

ll sum[N][N];
int main()
{
```

```
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n , p; cin >> n >> p;
    int pre[6] = {1 , 10 , 100 , 1000 , 10000 , 100000};
    for (int i = 1; i <= n; i++) {
        dp[i][lg(i)] = 26;

    }
    for (int i = 1; i <= n; i++) {
        //小心对无关的状态做贡献。计算错误。还是要详细研究转移过程。
        for (int j = lg(i) + 1; j <= min(n , i * 2); j++) {
            for (int k = 1; k <= 4; k++) {
                dp[i][j] += (((sum[max(0 , i - pre[k - 1])][j - k - 1]
- sum[max(0 , i - pre[k])][j - k - 1]) + p) % p) * 25;
                dp[i][j] %= p;

            }
        }
        for (int j = 1; j <= n; j++) {
            sum[i][j] = (sum[i - 1][j] + dp[i][j]) % p;
        }
    }
    ll ans = 0;
    for (int i = 1; i < n; i++) {
        ans = (ans + dp[n][i]) % p;
    }
    cout << ans << '\n';
}
```

洛谷水题：

P1103 书本整理

P1103 书本整理 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

solve

用一种非常暴力的dp定义方法。把之前的解中可以关注的大多数属性都关注了。对于简单问题这总是可行：

状态定义如下：

$f_{i,j,k}$表示：当前考虑了前i个，书架上的上一本书宽度为j， 已经使用丢掉了k本书的最小不整齐代价。

状态转移方程如下

$$
\begin{gathered}
if(a[i]!=j) \\
f[i][j][k]=min(f[i][j][k],f[i-1][j][k-1]) \\
if(a[i]==j) \\
f[i][a[i]][k]=min(f[i-1][a[i]][k],f[i-1][j][k]+abs(j-w))
\end{gathered} \tag{2}
$$

code

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;


const int N = 2E2 + 10;
const int inf  = 1E9 + 10;
//分别表示当前的尾数，当前去掉的书本的数目。
struct node {
    int h , w;
} a[N];
int f[N][N][N];
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n , kk;
    cin >> n >> kk;

    for (int i = 0; i <= n; i++)
        for (int j = 0 ; j <= 200 ; j++)
            for (int k = 0; k <= kk; k ++) {
                f[i][j][k] = inf;
            }
    //cout << f[0][0][0] << '\n';
    for (int i = 1; i <= n; i++) {
        cin >> a[i].h >> a[i].w;
    }
    sort(a + 1 , a + 1 + n , [&](node & i , node & j) {
        return i.h < j.h;
    });

    for (int i = 1; i <= n; i++) {
        f[i][a[i].w][i - 1] = 0;
    }
    for (int i = 1; i <= n; i++) {
        //考虑状态怎么迁移。
        //当前新的状态可能由什么转移？
        for (int k = 1; k <= kk; k++)
            for (int j = 1; j <= 200; j++) {

```

```
                //f[i][j][k]->k = 0.是什么情况?
                f[i][j][k] = min(f[i][j][k] , f[i - 1][j][k - 1]);
            }
        for (int k = 0; k <= kk; k++ )
            for (int j = 0; j <= 200; j++) {
                f[i][a[i].w][k] = min(f[i][a[i].w][k] , f[i - 1][j][k]
+ abs(a[i].w - j));
            }
    }
    int ans = (1 << 29);
    for (int i = 0; i <= 200; i++) {
        ans = min(ans , f[n][i][kk]);
    }
    cout << ans << '\n';
}
```

P1564 膜拜

将一段01字符串分成若干段。对于每一段满足：一段中字符相同，或两种字符数量的差别不超过m。求出最少可以分成多少段。

solve

线性dp ， 大概是属于被开发烂了的数组模型。但是事实上自己还是做的很慢。

1. 状态定义

   dp_i表示1....i最少可以分成多少段。

2. 状态转移

   $dp_i = min(dp_j + 1)$前提是j ... i合法。

code

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 3000;
int a[N];
int f[N];
int sum[N];
int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n , m;
```

```
    cin >> n >> m;
    fill(f + 1 , f + 1 + n , 100000000);
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        sum[i] = sum[i - 1] + (a[i] == 1);
    }

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            if (sum[i] - sum[j - 1] == i - j + 1 || sum[i] - sum[j - 1] == 0 || abs(i - j + 1 - 2 * (sum[i] - sum[j - 1])) <= m ) {
                f[i] = min(f[i] , f[j - 1] + 1);
            }
        }
    }
    cout << f[n] << '\n';
}
```

P1681 最大正方形II（double experiences）

P1681 最大正方形II - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

这种模型开发有限，但是也不大懂：

类似的问题如下：

P1387 最大正方形 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

最大正方形

solve

状态定义

定义$f_{i,j}$表示以，i , j 为右下角的的满足所有格子都是1的最大正方形边长。

定义$w_{i,j}$表示当前点向左走的最大距离。$h_{i,j}$表示向上走的最大距离。

状态转移方程

$$f_{i,j} = min(f_{i-1,j-1}, w_{i,j}, h_{i,j}) + (s_{i,j==1})$$

code

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 10000 + 10;
```

```cpp
int a[N][N] , f[N][N] , h[N][N] , w[N][N];


int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n , m; cin >> n >> m;
    int ans = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];

            f[i][j] = min({f[i - 1][j - 1] , w[i - 1][j] , h[i][j - 1]}) + a[i][j];
            if (a[i][j])w[i][j] = w[i - 1][j] + 1 , h[i][j] = h[i][j - 1] + 1;
            ans = max(f[i][j] , ans);
        }
    cout << ans << '\n';
}
```

最大正方形二

solve

1. 关注到关注的解是正方形。

状态定义

定义左下角为$f_{i,j,0/1}$表示i , j作为右下角，该处为0/1的最大正方形的面积。

状态转移

如果当前位是0:

$$f_{i,j,0} = min(f_{i-1,j-1,0}, f_{i-1,j,1}, f_{i,j-1,1}) + 1$$

如果当前位是1

$$f_{i,j,1} = min(f_{i-1,j-1,1}, f_{i-1,j,0}, f_{i,j-1,0}) + 1$$

和上述比较，在指标函数中，就已经有了一些与最大h ,最大w相关的长度。

code

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 10000 + 10;

int a[N][N] , f[N][N][2];

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n , m; cin >> n >> m;
    int ans = 0;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];
            if (a[i][j]) {
                f[i][j][1] = min({f[i - 1][j - 1][1] , f[i - 1][j][0] , f[i][j - 1][0]}) + 1;
                ans = max(f[i][j][1] , ans);
            } else {
                f[i][j][0] = min({f[i - 1][j - 1][0] , f[i - 1][j][1] , f[i][j - 1][1]}) + 1;
                ans = max(f[i][j][0] , ans);
            }

        }
    cout << ans << '\n';
}
```

天梯赛训练练习：

Setsuna的K数列

D-Setsuna的K数列_2022年中国高校计算机大赛-团队程序设计天梯赛（GPLT）上海理工大学校内选拔赛 (nowcoder.com)

空间限制：C/C++ 262144K，其他语言524288K

64bit IO Format: %lld

题目描述

Komorebi非常喜欢数列，但他实在太弱了无法想象出一个数列该如何构造，于是他就去FD(Frog Department)找Setsuna求助，Setsuna是一名光荣的FTCer(Frog,Time Controller)，并没有很多空闲，就随口给Komorebi构造了一个神秘的K数列，构建方法如下：

1、创建一个集合A_K，集合内包含K的所有整数次幂，如当$K = 3$时，A_3包含$1, 3, 9, 27 \ldots$，但不会包含$2, 4 \ldots$。

2、取A_K内任意个元素各一个相加，并将他们放进一个新的集合B_K。

3、将B_K内的元素从小到大排序，得到的有序数列即是一个K数列S_K。

例如当$K = 3$时，$S_3 = \{1, 3, 4(3+1), 9, 10(9+1), 12(9+3), 13(9+3+1), \ldots\}$。

Komorebi拿到这个数列后很开心，每天都要盯着这个K数列看很久。久而久之这个数列有些数被Komorebi弄丢了，Komorebi想复原这个数列，又不好意思再去找Setsuna，他就只能来求助你了。

Komorebi想知道K数列S_K的第n项是多少，你可以帮帮他吗？当然这个数有可能会非常大，因此请输出答案对$10^9 + 7$取模后的结果。

输入描述:

solve

观察出一些现象，

1. 假定一个具体的值 k^a,对于$k^{0....a-1}$无论如何组合，都小于当前的数。
2. 类比二进制串：

然后找到一个严格的偏序关系。其实第 n大，各种数字的选择组合情况，就是n所表示的二进制的1上的情况。

code

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 1E6 + 10;
const int mod = 1E9 + 7;

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n , k; cin >> n >> k;
    ll  now = 1;
    ll ans = 0;
    while (n) {
        if (n % 2) {ans = (ans + now) % mod;}
        n /= 2;
```

```
        now = now * k % mod;
    }
    cout << ans << '\n';
}
```

叠硬币

H-叠硬币_2022年中国高校计算机大赛-团队程序设计天梯赛（GPLT）上海理工大学校内选拔赛 (nowcoder.com)

solve

定义dp状态：

$f_{i,j}$表示考虑前i个情况下，当前硬币堆高度之和为j的最小的硬币堆数。

状态转移方程：

$f_{i,j} = min(f_{i-1,j}, f_{i-1,j-w} + 1)$注意判j - w的大小。

但是上面没有解决所有问题：

对于找出转移路径：
定义$pre_{i,j}$表示$f_{i,j}$的解结构的尾部选择。

1. 转移更新时，保持记录即可。

关于还原：

1. 如果已经得到尾部的物品，可以计算上一个子问题状态$f_{i-1,h-a[pre[i][j]]}$

~~**但是还没有解决问题：**~~

还要求求出字典序最小的方案：
经典问题：倒过来考虑即可。进一步证明，在背包问题中的其它问法的笔记中写过：

code

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 3E3 + 10;
const int inf = 1 << 29;
int f[N][N] , pre[N][N];
int a[N];

int main()
```

```
{
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n , H; cin >> n >> H;
    for (int i = 0; i <= n + 1; i++) {
        for (int j = 0; j <= H; j++) {
            f[i][j] = inf;
        }
    }
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }
    sort(a + 1 , a + 1 + n);
    f[n + 1][0] = 0;
    for (int i = n; i >= 1; i--) {
        int x = a[i];
        for (int j = 0; j <= H; j++) {
            f[i][j] = f[i + 1][j];
            pre[i][j] = pre[i + 1][j];
        }
        //动态规划中的复原的基本功没有做好：
        for (int j = H; j >= x; j-- ) {
            if (f[i + 1][j - x] + 1 <= f[i][j]) {
                f[i][j] = f[i + 1][j - x] + 1;
                pre[i][j] = i;
            }
        }
    }
    if (f[1][H] == inf) {cout << -1 << '\n';}
    else {
        cout << f[1][H] << '\n';
        queue<int> que;
        int id = pre[1][H];
        int h = H;
        while (id) {
            que.push(id);
            h -= a[id];
            id = pre[id + 1][h];
        }
        while (que.empty() == false) {
            cout << a[que.front()] << ' ';
            que.pop();
        }
        cout << '\n';
    }
}

/* stuff you should look for
```

```
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/

```

Wiki with Fake AKGPLT

F-Wiki with Fake AKGPLT_“2021年中国高校计算机大赛-团队程序设计天梯赛（GPLT）上海理工大学校内选拔赛”(nowcoder.com)

solve

看代码即可：

code

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 1E6 + 10;

string t = "AKGPLT";

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);

    int n;
    cin >> n;
    while (n --) {
        string s;
        cin >> s;
        if (s > t) {
            cout << 0 << '\n';
            continue;
        }
        bool flag = true;
        for (int i = 0 ; i < (int)s.length(); i++) {
            if (i >= 2 && s[i] > 'K') {
                cout << i - 1 << '\n';
                flag = false;
                break;
```

```
            }
            if (s[i] != 'A') {
                cout << i << '\n';
                flag = false;
                break;
            }
        }
        if (flag) { cout << -1 << '\n';}


    }
}

/* stuff you should look for
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/
```

生长总结

1. 前缀的字符串（不相等）的字典序比其本身小

摘苹果

H-摘苹果_2023年中国高校计算机大赛-团队程序设计天梯赛（GPLT）上海理工大学校内选拔赛（同步赛）(nowcoder.com)

solve

注意到这种操作最多可以进行20次。维护未到尽头的点进行的单点修改即可。

赛时变量写错：惨痛经验。如果确定自己思路是对的。但是一直没有调出来。有可能是低级错误。

1. 变量名写错。

解决方法可以是：

1. 重新写一遍。
2. 再次审查代码中变量名称的意义。

code

```cpp
#include<bits/stdc++.h>
using namespace std;
using ll = long long;

const int N = 1E5 + 10;
set<int> rec;
int n , m;
ll a[N];
class BIT {
    ll c[N];

public:
    ll query(int x) {
        ll res = 0;
        for (; x ; x -= x & (-x))
            res += c[x];
        return res;
    }
    void modify(int x, ll d) {
        for (; x <= n; x += x & (-x)) {
            c[x] += d;
        }
    }
};

BIT d1, d2;

//树状数组求区间和公式：
//cout << (x + 1)*d1.query(x) - d2.query(x) - (x)*d1.query(x - 1) + d2.query(x - 1) << '\n';
//区间修改仔细点，前加后减。小心记错结论。
//求和问题非常容易溢出。

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);

    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        if (a[i] >= 10) {
            rec.insert(i);
        }
        d1.modify(i , a[i]);
        if (a[i] < 100) {
            d2.modify(i , 1);
```

```
        }
    }
    for (int i = 1; i <= m; i++) {
        int op , l , r; cin >> op >> l >> r;
        if (op == 1) {
            auto ptr = rec.lower_bound(l);
            while (ptr != rec.end() && *ptr <= r) {
                ll t = a[*ptr];
                int no = *ptr;
                a[no] -= a[no] / 3 + (a[no] % 3 != 0);
                d1.modify(no , a[no] - t);
                if (a[no] < 100 && t >= 100) {
                    d2.modify(no , 1);
                }
                ptr++;
                if (a[no] < 10) {
                    rec.erase(no);
                }
            }
        } else if (op == 2) {
            cout << d2.query(r) - d2.query(l - 1) << '\n';

        } else if (op == 3) {
            cout << d1.query(r) - d1.query(l - 1) << '\n';
        }
    }
}

/* stuff you should look for
* int overflow, array bounds
* special cases (n=1?)
* do smth instead of nothing and stay organized
* WRITE STUFF DOWN
* DON'T GET STUCK ON ONE APPROACH
*/
```

---

div2 857

D. Buying gifts

solve

考虑对解空间进行枚举优化：对于第一个朋友的礼物选择。枚举一个价值最大值。于是发现可以对解空间进行。只要对每一个最大值d的所有可能枚举完全即可。

1. 关注某个最大值的解空间：

1. 用有序的眼光考察：发现同一种（属于第一个朋友）比当前枚举最大值大的，都要选b_j,而比它小的任意选：考虑最优条件：必选最大值，以及通过可自由选择的一些最大值进行调整。
    1. 如果必选b中最大值大于max.那么最优方案已经确定
    2. 如果必选b中最大值小于。然后从自由选择的值中，找第一个比max大的，以及第一个小于等于max的。

生长

1. contest中，满脑子想着二分。但是二分的途中就已经确定了解。
2. 有一些问题，二分check的过程中，可能就已经重复了所有的解。我相当于使用了一个最蠢的方法。每一次枚举所有的最优子集。然后看是否出过check值。但其实已经得到了所有的最优子集。在一次check的过程中就已经可以算出来。

平常的普通问题：最优子集比较庞大，或者不容易枚举。所有采用一种局部枚举的二分操作。

code

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

const int oo = 0x0fffffff;
const int N = 1E6 + 10;

struct node {
    int x;
    int y;
} a[N];
int n;

// int pre[N];
int sux[N];
int solve() {
    set<int> rec1;
    //然后就是一直选择，一直check
    int ans = 1E9;
    for (int i = 1; i <= n; i++) {
        int mx = 0;
        if (i != n) {
            mx = sux[i + 1];
            ans = min(ans , abs(a[i].x - mx));
            //考虑从前面找一个较大值。
            if (mx < a[i].x ) {
                auto ptr = rec1.lower_bound(a[i].x);
                if (ptr != rec1.end()) ans = min(abs(*ptr - a[i].x) , ans);
                if (ptr != rec1.begin()) -- ptr;
                ans = min(ans , abs(*ptr - a[i].x));
```

```
            }
        } else {
            auto ptr = rec1.lower_bound(a[i].x);
            if (ptr != rec1.end()) ans = min(abs(*ptr - a[i].x) ,
ans);
            if (ptr != rec1.begin()) -- ptr;
            ans = min(ans , abs(*ptr - a[i].x));
        }
        rec1.insert(a[i].y);
    }
    return ans;
}

void work(int testNo)
{
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i].x >> a[i].y;
    }
    sort(a + 1 , a + 1 + n , [&](const node & i , const node & j) {
        return i.x < j.x;
    });
    sux[n + 1] = 0;
    for (int i = n; i >= 1; i--) {
        sux[i] = max(sux[i + 1] , a[i].y);
    }
    cout << solve() << '\n';
}


int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);

    int t; cin >> t;
    for (int i = 1; i <= t; i++)work(i);
}
```

C. The Very Beautiful Blanket

Problem - C - Codeforces

简介

构造一个举证。满足矩阵中的每一个4*4的矩阵中满足如下条件

1. $a_{11} \oplus a_{12} \oplus a_{21} \oplus a_{22} = a_{33} \oplus a_{34} \oplus a_{43} \oplus a_{44}$
2. $a_{13} \oplus a_{14} \oplus a_{23} \oplus a_{24} = a_{31} \oplus a_{32} \oplus a_{41} \oplus a_{42}$

找出一个元素种数最多的矩阵。

solve

特殊的，构造一个矩阵满足以下形式：任何一个四块的异或和都为0.

如果一个数字上的某个位置上有1怎么消除？并且要保证两两不同：

1. 对于最高的位置上： 不同的行打1个1.于是同行的两个必然可以消掉。行与行的数字之间就不会有交叉。
2. 对于较低的位置上，从1开始填比较低的位置。这样任意两行之间，上下两个元素就可以把低位的1给中和掉。于是无论怎么圈，都可以找出一个异或和为0的正方形：

code

```
void work(int testNo)
{
    int n , m;
    cin >> n >> m;
    cout << n*m << '\n';
    for (int i = 0; i < n; i++) {
        for (int j = 0 ; j < m; j++)
            cout << (i << 10) + j << " \n"[j == m - 1];
    }
}
```