

2023/3/5

青春须早为，岂能长少年

1. 总括

承上启下 [2023.2月下旬.md](#)

上一阶段的情况：

1. 算法学习进度，
2. 相对较慢。其中数位dp中的几道很难的问题，梦幻珠宝岛卡了两三天。
3. dls的课程只是刷到区间dp（dp进阶第五节。）
4. 比赛强度：
 1. 有比赛基本都打了。包括一场牛客小白月赛。两场师兄拉的练习赛。div2 div1 + div2 , edu div2 atcoder abc 两场。
5. 练习强度：
 1. 虽然一天有三道题目情况。但是题目难度参差不齐。只要是比赛，课程内容的题。
6. 复习强度
 1. 对于上一阶段的总结复习。现在有了新的计划。在一个阶段结束之后。打印笔记。复习，具体的策略还要进一步研究。但是给予一定强度的复习，再次生长思考以及记忆是必要的。

本阶段目标：

1. 尝试板刷：

1. 板刷图论基础专题。
2. 板刷各种动态规划专题。

2. 继续学习dls的课程内容。

1. 不会的问题就跳，掌握课程对于一个类型的dp的关键点。
2. 总结区域赛dp问题的难度水平。

3. 比赛上

1. atcoder
2. codeforces

4. 为校赛做哪些准备？

1. 学学基础数学问题。
2. 常用的数据结构可以再熟练一点。
3. 各种类型的dp问题再做一遍。

笔记记录

语法等的学习

1. 关于位运算中的一些函数使用：**未补**

算法学习

动态规划：

1. 数位dp进阶
 1. [数位dp进阶.md](#)

图论

1. 树哈希：[树哈希.md](#)

补题

二分

1. [D. Maximum Subarray.md](#)
2. [D. Renting Bikes.md](#)

提高string简单题速度。

- 1.

div3 (2023/3.4)

1. [2023.3上旬.md](#)
2. [树哈希.md](#)

刷题

动态规划：

1. 牛客
 1. [删括号.md](#)
 2. [美丽序列.md](#)
 3. [codeforces.md](#)
 4. [和与或.md](#)
 5. [牛牛与数组.md](#)
 6. [牛牛的回文串.md](#)
 7. [牛牛去买球.md](#)
 8. [牛牛的计算机内存.md](#)

atcoder

1. [YYYYX.md](#)

数位dp进阶

数数3

[数数3 - 题目 - Daimayuan Online Judge](#)

求区间中有多少个数字a满足存在连续三个数位 a_i, a_{i+1}, a_{i+1} 使得 $a_i < a_{i+1} < a_{i+2}$

[basic.md](#) 参照题解:

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6
7  ll dp[20][2][20][5];
8
9
10 //用单词前三个的习惯。
11 ll dfs(int rem , int exit , int pre , int inc) {
12     if (rem == 0) return exit;
13     if (dp[rem][exit][pre][inc] != -1)
14         return dp[rem][exit][pre][inc];
15
16     ll &res = dp[rem][exit][pre][inc];
17     res = 0;
18
19     for (int i = 0; i <= 9; i++) {
20         int inc_ = (i > pre) ? min(inc + 1 , 3) : 1;
21         res += dfs(rem - 1 , exit || inc_ == 3 , i , inc_);
22     }
23     return res;
24 }
25
26 ll solve(ll x) {
27     x++; //细节1
28     vector<int> d;
29     while (x) {d.push_back(x % 10); x /= 10;}
30     //处理前导0的情况。
31     ll ans = 0;
32     int m = d.size();
33     reverse(d.begin(), d.end());
34     for (int i = 1; i < m; i++) {
35         for (int j = 1; j <= 9; j++) {
36             ans += dfs(i - 1, 0 , j , 1);
37         }
38     }
39     //然后处理贴着上界走的情况。
40     int exit = 0 , pre = 0, inc = 0;
41     for (int i = 0; i < m; i++) {
42         for (int j = (i == 0); j < d[i] ; j++) {
43             //同时要记录前缀的一些信息。
44             int inc_ = (j > pre) ? min(inc + 1 , 3) : 1;
45             ans += dfs(m - i - 1 , exit || inc_ == 3, j , inc_);
46         }
47         inc = (d[i] > pre) ? min(inc + 1, 3) : 1;
48         pre = d[i];
49     }
```

```

49     exit |= (inc == 3);
50 }
51 return ans;
52 }
53
54 int main()
55 {
56     ios::sync_with_stdio(false);
57     cin.tie(0);
58
59     memset(dp , -1 , sizeof dp);
60     ll l , r;
61     cin >> l >> r;
62     cout << solve(r) - solve(l - 1) << '\n';
63 }
64

```

CF Round #739 (Div 3) F, Nearest Beautiful Number

[Problem - F2 - Codeforces](#)

找到最小的，满足大于等于n的，美丽数位k的数字。

solve.

暴力搜索加剪枝：并不是数位dp的解法。

1. 可以估算复杂度非常小，为n的字符值之和。
1. dfs算法从可能的最小数字解开始进行枚举

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4
5  const int oo = 0xffffffff;
6  const int N = 1E6 + 10;
7
8  void work(int testNo)
9  {
10     int n , k; cin >> n >> k;
11     vector<int> d;
12     //尽量开大一点，拓展当前值域，增多可表达内容。
13     int vis[10] {};
14     while (n) {d.push_back(n % 10); n /= 10;}
15     reverse(d.begin() , d.end());
16     // x: 表示当前遍历的位置。 large, 前缀是否大于规定的前缀。nums前面的k的前缀中数字的
17     // 种数。
18     function<bool (int , int , int , int)> dfs = [&](int x , int large , int
19     nums , int cunt) {
20         //能走到一步必然有解了。
21         if (x == (int)d.size()) {
22             cout << nums << '\n';
23             return true;
24         }
25     };
26 }
27

```

```

22         } else {
23             //然后从哪里开始枚举呢？分情况。
24             //如果已经large。那么就从0开始。否则从 d[x]开始
25             for (int i = (large ? 0 : d[x]); i <= 9; i++) {
26                 //然后开始各种枚举枚举构造大法。
27                 vis[i] += 1;
28                 int ncunt = cunt;
29                 if (vis[i] == 1)ncunt += 1;
30                 if (ncunt <= k && dfs(x + 1 , large | (i > d[x]) , nums * 10
+ i , ncunt)) {
31                     return true;
32                 }
33                 vis[i] -= 1;
34             }
35             return false;
36         }
37     };
38     dfs(0 , 0 , 0 , 0);
39 }
40
41 int main()
42 {
43     ios::sync_with_stdio(false);
44     cin.tie(0);
45
46     int t; cin >> t;
47     for (int i = 1; i <= t; i++)work(i);
48 }
49
50 /* stuff you should look for
51 * int overflow, array bounds
52 * special cases (n=1?)
53 * do smth instead of nothing and stay organized
54 * WRITE STUFF DOWN
55 * DON'T GET STUCK ON ONE APPROACH
56 */

```

乘法

[乘法 - 题目 - Daimayuan Online Judge](#)

简介:

将乘法转换成加减法组合的最小花费。

solve

1. 从高位到低位考虑:

定义 $S_i = a_1 \dots a_i 000000$ ($i - 1$ 个0)二进制串的值。T为前期操作地结果。

1. 假设现在枚举到了第i位。要有解，必须满足:

1. $S - T = 0$ 或者 $T - S = (1 \ll (i))$

2. 否则，在低位进行任意加减操作。都不会把差异消除。

2. 状态设计

从高位到低位对应 $i = 1 \dots i = n$

f_i 表示从高位开始考虑到了第 i 个位置。

g_i 表示从高位开始考虑到了第 i 个位置。 $T_i - S_i = (2^{n-i})$

3. 初始化:

1. $f_0 = 0, g_i = 1$

4. 状态转移方程

1. 如果 $s_i = '0'$

1. 对于 f_i 有如下转移:

1. f_{i-1} 啥都不变。

2. 对于 g_i 有如下方案

1. f_{i-1} , 对应方案加当前位权。

2. g_{i-1} , 对应方案减去当前位权。

2. 如果 $s_i = '1'$

1. 对于 f_i

1. g_{i-1} 对应的方案减去当前位权。

2. f_{i-1} 对应的方案加上当前为位权。

2. 对于 g_i

1. g_{i-1} 对应的方案减去当前位权。

code

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  int f[N] , g[N];
7
8  int main()
9  {
10     ios::sync_with_stdio(false);
11     cin.tie(0);
12     string s; cin >> s;
13     int n = s.size();
14     s = ' ' + s;
15     fill(f , f + 1 + n , N);
16     fill(g , g + 1 + n , N);
17     f[0] = 0; g[0] = 1;
18     for (int i = 1 ; i <= n; i++) {
19         if (s[i] == '1' ) {
20             f[i] = min(f[i - 1] + 1 , g[i - 1] + 1);
21             g[i] = g[i - 1];
22
23         } else {
24             f[i] = f[i - 1];
25             g[i] = min(f[i - 1] + 1 , g[i - 1] + 1);
26         }
27         //cout << "now is  " << i << "  " << f[i] << "  " << g[i] << '\n';
28     }
```

```

29     cout << f[n] * 2 - 1 << '\n';
30 }

```

P3188 [HNOI2007]梦幻岛宝珠

特殊的01背包问题。

特殊点：

1. 物品的重量大小比较特殊： $w = a2^b$
2. 背包总承受重量非常大。

10mins

关注数字的特殊性：

1. 数字都是一个二进制数的倍数。
2. a, b 都非常小。

可以优化值域，枚举一些特殊的二进制数字。

solve

考虑将背包按照 b 分组。

关注几个现象

1. 同一组中的背包组合的体积大小都集中在部分区域：

对于 b, a .

①. $\underbrace{\hspace{1cm}}_{0 \cdot 0 \cdots \cdots a} \quad b-1 \uparrow 0 \quad \text{每一个二进制表示, 占着。}$

②. $b \in [1, 20], n \leq 100, \quad \text{max}(\text{sum } b) = 2000$

所以物品集中在 ~~7000~~ $[1 \times 2^b, 2000 \times 2^b]$ 的区域。

面对这些现象可以采取什么样的策略？

CCPC Changchun 2020 D, Meaningless Sequence

[CCPC Changchun 2020 D, Meaningless Sequence - 题目 - Daimayuan Online Judge](#)

注意仔细地读标号。防止读错。

第一次读这道题地时候，没有发现 & 是位于下标中的。

solve

~~反正只知道，是我一些不懂的规律。~~

打表可以发现， $a_n = c^{\text{popcount}(n)}$

特别的有， $a_0 = 1$

自然而然地问题转换成了一道简单地数位dp。

solve1

dfs过程中枚举前缀。然后直接利用组合数计算贡献即可。

solve2

枚举任意前缀时，发现后缀的贡献总是：

$$c^{\text{pre}(1)} \times (c + 1)^{\text{suffix_len}}$$

code of solve1

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 3E3 + 10;
6  const ll mod = 1E9 + 7;
7
8  string s;
9  ll c;
10 ll d[N][N][2] , C[N][N] , p[N];
11 int n;
12
13 ll dfs(int rem , int sum , bool larger) {
14
15     if (rem == 0) return p[sum];
16     if (larger == false) {
17         ll res = 0;
18         for (int i = 0; i <= rem ; i ++ )
19             res += (p[sum] * C[rem][i] % mod) * p[i] % mod;
20         return res;
21     }
22     ll res = 0;
23     for (int i = 0; i <= s[n - rem] - '0'; i++) {
24         res = (res + dfs(rem - 1, sum + (i == 1) , i == (s[n - rem] - '0'))
25 % mod;
26         }
27     return res;
28 }
```



```

29 void init() {
30     cin >> s >> c;
31     n = s.size();
32     p[0] = 1;
33     for (int i = 1; i < N; i++) p[i] = p[i - 1] * c % mod;
34     for (int i = 0; i < N; i++) {
35         for (int j = 0; j <= i; j++) {
36             if (j == 0) c[i][j] = 1;
37             else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
38         }
39     }
40 }
41
42 int main()
43 {
44     ios::sync_with_stdio(false);
45     cin.tie(0);
46     init();
47     cout << dfs(n , 0 , true) << '\n';
48
49 }
50
51 /* stuff you should look for
52 * int overflow, array bounds
53 * special cases (n=1?)
54 * do smth instead of nothing and stay organized
55 * WRITE STUFF DOWN
56 * DON'T GET STUCK ON ONE APPROACH
57 */

```

code of solve2

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  const int mod = 1E9 + 7;
7  ll p[N];
8
9  int main()
10 {
11     ios::sync_with_stdio(false);
12     cin.tie(0);
13
14     string s; int c;
15     cin >> s >> c;
16     int n = s.size();
17
18     p[0] = 1;
19     for (int i = 1 ; i <= n; i++) {
20         p[i] = p[i - 1] * (c + 1) % mod;
21     }
22
23     ll pre = 1 , ans = 0;

```

```

24     for (int i = 0; i < n; i++) {
25         if (s[i] == '1') {
26             ans = (ans + pre * p[n - i - 1]) % mod;
27             pre = pre * c % mod;
28         }
29     }
30     cout << (pre + ans) % mod << '\n';
31 }
32
33 /* stuff you should look for
34 * int overflow, array bounds
35 * special cases (n=1?)
36 * do smth instead of nothing and stay organized
37 * WRITE STUFF DOWN
38 * DON'T GET STUCK ON ONE APPROACH
39 */

```

CCPC Jinan 2020 L, Bit Sequence

[CCPC Jinan 2020 L, Bit Sequence - 题目 - Daimayuan Online Judge](#)

[L-Bit Sequence 第45届国际大学生程序设计竞赛 \(ICPC\) 亚洲区域赛 \(济南\) \(nowcoder.com\)](#)

简介

定义 f_i 为 i 的二进制串中的 1 的个数。给定一个 01 序列 $a_{0\dots m-1}$ 。求取 $[0\dots x-1]$ 中，满足对于任意 $0 \leq i < m, f(x+i) \% 2 = a_i$

solve

观察一些现象：

1. 由于 m 的范围比较小。因此两个数字相加，结果相比于 x ， m 只能影响低 7 位。而前面高位的奇偶性，不会受到影响。

树哈希

解决问题：

1. 快速判断树是否同构的问题：
 1. 同构的概念：对一棵树，进行对同父亲的子树进行互换。进行若干次操作后，两颗树可以相等。那么称这棵树是同构的。

参考：[一种好写且卡不掉的树哈希 - 博客 - peehs moorhsum 的博客 \(uoj.ac\)](#)

算法简介

关注树结构的属性：定义一个哈希函数。

科学的定义一个哈希函数。

1. 根哈希函数，代表了当前子树的结构情况。
2. 哈希函数，和子树的哈希函数联系。
最终就是要降不同的子树结构不会落入同一个哈希值的概率。
这里直接找一些大佬的哈希函数设计。

入门问题

1. (<https://codeforces.com/contest/1800/problem/G>)

板子

邓老师

```
1 using ll = long long;
2 using ull = unsigned long long;
3 mt19937_64 rnd(chrono::steady_clock::now().time_since_epoch().count());
4 ull bas = rnd();
5 //ull bas = (ull)1E18 + 7;
6 void work(int testNo)
7 {
8     int n; cin >> n;
9     vector<vector<int>> e(n);
10    for (int i = 1; i < n; i++) {
11        int u, v; cin >> u >> v;
12        u--; v--;
13        e[u].push_back(v);
14        e[v].push_back(u);
15    }
16
17    vector<ull> h(n), f(n);
18    function<ull(ull)> H = [&](ull x) {
19        return x * x * x * 19890535 + 19260817;
20    };
21
22    function<ull(ull)> F = [&](ull x) {
23        return H(x & (((1ll << 32) - 1))) + H(x >> 32);
24    };
25
26    function<void(int, int)> dfs = [&](int u, int par) {
27        h[u] = bas;
28        //
29        for (auto v : e[u]) if (v != par) {
30            dfs(v, u);
31            h[u] += F(h[v]);
32            // rec[h[v]].push_back(v);
33        }
34        //cout << "no is " << u << " " << rec.size() << '\n';
35        //通过节点的hash值情况进行一些哈希。
36    };
37 }
```

补题

二分

D. Maximum Subarray

<https://codeforces.com/contest/1796/problem/D>

简介

给定n组数组。每一个数组长度为m。

定义一种运算。选定数组 a, b 进行与运算，运算结果为 $c_i = \max(a_i, b_i)$ 。选定任意两个数组，运算得到值域 $\{c\}$ 。找出， $\min(\max(b_i))$ 。

solve

1. 第一个点是敏锐的感受到是二分查找问题。

1. 设计check函数上：枚举 x 。对于任意一个数组，都映射到一个长度为n数字d上。如果 $b_i \geq x, d_i = 1$ else $d_i = 0$ 。如果存在两个数组之间进行max运算之后的最小值大于等于 x 。那么必然运算的结果是所有二进制位上都为1的数字。

2. 投射到小值域。鸽笼定理：只需要关注解的存在性。高达 2^5 的数组数量。对于d的集合。必然都映射到d的集合上。这样就将数字分成了若干类。只需要枚举 $0 \leq d \leq 1^m - 1$ 这几类的数字d即可。

code

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  int a[N][11];
7  int n, m;
8  int a1, a2;
9
10 bool check(int x)
11 {
12     int t = 1 << (m);
13     vector<int> rec(t, -1);
14     //然后
15     for (int i = 0; i < n; i++) {
16         int now = 0;
17         for (int j = 0; j < m; j++) {
18             if (a[i][j] >= x) now |= (1 << j);
19         }
20         rec[now] = i;
21     }
22     t--;
23     if (rec[t] != -1) {
24         a1 = a2 = rec[t];
25         return true;
26     }
27     for (int i = 0; i < t; i++)
28         for (int j = i + 1; j <= t; j++) {
29             if (rec[i] != -1 && rec[j] != -1 && (i | j) == t) {
30                 a1 = rec[i]; a2 = rec[j];
31                 return true;
32             }
33         }
34     return false;
35 }
36
```

```

37 int main()
38 {
39     ios::sync_with_stdio(false);
40     cin.tie(0);
41     cin >> n >> m;
42     for (int i = 0; i < n; i++)
43         for (int j = 0; j < m; j++)
44             cin >> a[i][j];
45     int low = 0 , high = 1E9 + 10;
46     while (low < high) {
47         int mid = (low + high + 1) / 2;
48         if (check(mid))
49             low = mid;
50         else high = mid - 1;
51     }
52     cout << a1 + 1 << ' ' << a2 + 1 << '\n';
53 }
54
55 /* stuff you should look for
56 * int overflow, array bounds
57 * special cases (n=1?)
58 * do smth instead of nothing and stay organized
59 * WRITE STUFF DOWN
60 * DON'T GET STUCK ON ONE APPROACH
61 */

```

二分

D. Renting Bikes

<https://codeforces.com/problemset/problem/363/D>

题目简介

有 n 个学生。他们之中有公共的可使用的钱 记为 a 。每一个学生有自己的钱。它们去租自行车。但是他们自己的钱买自己的自行车。公共的钱可以任意的分配。问最多可以有多少人拥有自行车。同时使得他们自己付出的钱最少。

solve

1. 尝试二分。枚举一个数字 x ， check： 购买自行车的数量是否可以达到这个数字。
 1. 最保守的策略是， 钱最多的 x 个人， ——匹配买最便宜的 x 辆自行车。如果不够就去补
 1. 如果其它结构可以买到 x 辆。那么上述的购买策略也一定合理。
 2. 所以上述购买的策略是边界策略。作为check的标准。
2. 花钱最小
 1. 有多少补多少。显然就是 $\min(\text{sum} - a, 0)$.sum指的是所有单车的价格。

code

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 using ll = long long;
4
5 const int N = 1E6 + 10;

```

```

6
7  ll n , m , a;
8  ll p[N] , b[N];
9
10 bool check(int x) {
11     ll sum = 0;
12     for (int i = 1; i <= x; i++) {
13         sum += max(0LL , b[i] - p[n - x + i]);
14     }
15     return sum <= a;
16 }
17
18 int main()
19 {
20     ios::sync_with_stdio(false);
21     cin.tie(0);
22     cin >> n >> m >> a;
23     for (int i = 1; i <= n; i++) cin >> p[i];
24     for (int i = 1; i <= m; i++) cin >> b[i];
25
26     int low = 0 , high = min(n , m);
27     sort(p + 1 , p + n + 1);
28     sort(b + 1 , b + m + 1);
29     while (low < high) {
30         int mid = (low + high + 1) / 2;
31         if (check(mid)) low = mid;
32         else high = mid - 1;
33     }
34     ll sum = 0;
35     for (int i = 1; i <= low ; i++) {
36         sum += b[i];
37     }
38     cout << low << ' ' << max(0LL , sum - a) << '\n';
39 }
40 /* stuff you should look for
41 * int overflow, array bounds
42 * special cases (n=1?)
43 * do smth instead of nothing and stay organized
44 * WRITE STUFF DOWN
45 * DON'T GET STUCK ON ONE APPROACH
46 */

```

div3 位压

F. Dasha and Nightmares

<https://codeforces.com/contest/1800/problem/F>

简介

1. 给定若干个字符串
2. 找出若干对字符串满足：

solve

管理字符串的方法：

1. 压位，表示当前字符串中的各种字符出现了多少次。
 1. 如果简单的压位记录。会丢失掉某个字母是否存在的信息。

计算答案：

solve1

1. 枚举一个字母，表示在拼接串中该字母不存在。
2. 然后扫一遍。
 1. 但是初始化的花费将会非常的大。
2. 算法复杂度会达到 $\sum |s| + 2^6 * 26$

solve2

1. 在扫描的过程中。开一个巨大的map 或者 unordered_map
2. 直接记录，统计即可。

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6
7
8  int main()
9  {
10     ios::sync_with_stdio(false);
11     cin.tie(0);
12
13     int n ;
14     cin >> n;
15     ll ans = 0;
16     int mask = (1 << 26) - 1;
17     unordered_map <int, int > rec[26];
18     for (int i = 1; i <= n; i++) {
19         string s; cin >> s;
20         int ch = 0 , bit = 0;
21         for (auto c : s) {
22             ch |= 1 << (c - 'a');
23             bit ^= 1 << (c - 'a');
24         }
25         for (int i = 0; i < 26; i++) {
26             if (!(ch & (1 << i))) {
27                 ans += rec[i][bit ^ mask ^ (1 << i)];
28                 rec[i][bit]++;
```

```

29     }
30 }
31 }
32 cout << ans << '\n';
33 }
34
35 /* stuff you should look for
36 * int overflow, array bounds
37 * special cases (n=1?)
38 * do smth instead of nothing and stay organized
39 * WRITE STUFF DOWN
40 * DON'T GET STUCK ON ONE APPROACH
41 */

```

solve3

认识到抽象出来的26位是比字符串数量更大的。考虑在1的基础上做出改进。

1. 初始化数组的时候，只要枚举所有字符串的种类即可。贡献是 $O(n) * 26$
2. 统计上的复杂度是 $O(n)$
3. 预处理字符串的花费是 $O(\sum |s|)$

```

1 //这回只花了114514min就打完了。
2 //真好。记得多手造几组。ACM拍什么拍。
3 #include "bits/stdc++.h"
4 using namespace std;
5 template<typename typC,typename typD> istream &operator>>(istream
&cin,pair<typC,typD> &a) { return cin>>a.first>>a.second; }
6 template<typename typC> istream &operator>>(istream &cin,vector<typC> &a) {
for (auto &x:a) cin>>x; return cin; }
7 template<typename typC,typename typD> ostream &operator<<(ostream
&cout,const pair<typC,typD> &a) { return cout<<a.first<<' '<<a.second; }
8 template<typename typC,typename typD> ostream &operator<<(ostream
&cout,const vector<pair<typC,typD>> &a) { for (auto &x:a) cout<<x<<'\n';
return cout; }
9 template<typename typC> ostream &operator<<(ostream &cout,const vector<typC>
&a) { int n=a.size(); if (!n) return cout; cout<<a[0]; for (int i=1; i<n;
i++) cout<<' '<<a[i]; return cout; }
10 template<typename typC,typename typD> bool cmin(typC &x,const typD &y) { if
(x>y) { x=y; return 1; } return 0; }
11 template<typename typC,typename typD> bool cmax(typC &x,const typD &y) { if
(x<y) { x=y; return 1; } return 0; }
12 template<typename typC> vector<typC> range(typC l,typC r,typC step=1) {
assert(step>0); int n=(r-l+step-1)/step,i; vector<typC> res(n); for (i=0;
i<n; i++) res[i]=l+step*i; return res; }
13 #if !defined(ONLINE_JUDGE)&&defined(LOCAL)
14 #include "my_header\debug.h"
15 #else
16 #define dbg(...) ;
17 #define dbgn(...) ;
18 #endif
19 typedef unsigned int ui;

```



```

20 typedef long long ll;
21 #define all(x) (x).begin(),(x).end()
22 // template<typename T1,typename T2> void inc(T1 &x,const T2 &y) { if
    ((x+=y)>=p) x-=p; }
23 // template<typename T1,typename T2> void dec(T1 &x,const T2 &y) { if
    ((x+=p-y)>=p) x-=p; }
24 const int N=1<<26;
25 int cnt[N];
26 int main()
27 {
28     ios::sync_with_stdio(0); cin.tie(0);
29     cout<<fixed<<setprecision(15);
30     int n,i,j,k;
31     cin>>n;
32     vector<int> a(n),b(n);
33     for (i=0; i<n; i++)
34     {
35         string s;
36         cin>>s;
37         for (auto c:s) a[i]|=1<<c-'a',b[i]^=1<<c-'a';
38     }
39     ll r=0;
40     for (k=0; k<26; k++)
41     {
42         int B=(1<<26)-1^(1<<k);
43         for (i=0; i<n; i++) if (1^a[i]>>k&1)
44         {
45             ++cnt[b[i]];
46             r+=cnt[b[i]^B];
47         }
48         for (i=0; i<n; i++) if (1^a[i]>>k&1) --cnt[b[i]];
49     }
50     // for (i=0; i<n; i++) for (j=i; j<n; j++) if
    (__builtin_popcount(a[i]|a[j])==25&&__builtin_popcount(b[i]^b[j])==25) ++r;
51     cout<<r<<endl;
52 }
53

```

动态规划 刷题

删括号

[删括号 \(nowcoder.com\)](https://www.nowcoder.com/)

solve

关注一些解结构：

1. 发现最终删除的括号必然是连续排的。

定义状态： $d_{i,j,k}$ 表示对于 $s_{0..i}$ 删去 k 个删去若干个括号，此时左括号数目减去右括号数目为 k 情况下 $s_{0..i}$ 和 $t_{0..j}$ 是否匹配。

2. 考虑各种迁移情况：

1. 假设当前 $dp_{i,j,k}$ 为 false。无论怎么对当前位怎么操作最后都是 false.
2. 如果当前 $dp_{i,j,k}$ 为 true，且 k 为 0. 当前遇到了 ')' 显然错误因为追求的是连续的删除。

3. 如果当前 $dp_{i,j,k}$ 为true, 遇到'(', 那么连续删除的括号加1.
 4. 如果当前 $dp_{i,j,k}$ 为true, 遇到')', 那么连续的'('被抵消掉一个.
3. 体会到迁移总是正确的。(复习的话体会不到就继续体会。qaq)

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 100 + 10;
6  bool dp[N][N][N];
7
8  int main()
9  {
10     ios::sync_with_stdio(false);
11     cin.tie(0);
12     string s , t;
13     cin >> s >> t;
14     s += ' ';
15     t += ' ';
16     s = ' ' + s;
17     t = ' ' + t;
18     int n = s.length() - 1, m = t.length() - 1;
19     dp[0][0][0] = true;
20     for (int i = 0; i <= n; i++)
21         for (int j = 0; j <= i; j++)
22             for (int k = 0; k < n / 2; k++) {
23                 if (dp[i][j][k]) {
24                     if (k == 0 && s[i + 1] == t[j + 1])
25                         dp[i + 1][j + 1][k] = true;
26                     if (s[i + 1] == '(')
27                         dp[i + 1][j][k + 1] = true;
28                     else if (k)
29                         dp[i + 1][j][k - 1] = true;
30                 }
31             }
32     string ans[2] = {"Impossible\n" , "Possible\n"};
33     cout << ans[dp[n][m][0]];
34 }

```

美丽序列

<https://ac.nowcoder.com/acm/problem/21313>

solve

这种问题的解空间非常清晰明了。

关注几种属性进行分类

1. i , 表示考虑了前 i 个元素。
2. j , 表示结构的尾数位
3. k , 表示连续下降的位数是 k .
4. sum 结构的和位 sum .

通过枚举上述情况，就可以一条不漏的得到了所有的可能。并且完成迁移。这种模型的解是非常容易统计的。

code

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 60;
6  const int mod = 1E9 + 7;
7
8  int a[N];
9  ll dp[N][N][3][1700];
10
11 int main()
12 {
13     ios::sync_with_stdio(false);
14     cin.tie(0);
15     int n; cin >> n;
16     for (int i = 1; i <= n; i++)
17         cin >> a[i];
18     dp[0][0][0][0] = 1;
19     //
20     for (int t = 1; t <= n; t++) {
21         //枚举当前位置上放的东西
22         for (int now = 0; now <= 40; now++)
23             //枚举尾数
24             for (int i = 0; i <= 40; i++)
25                 //枚举连续长度
26                 for (int j = 0; j < 3; j++)
27                     //枚举前面的平均数
28                     for (int k = 0; k <= 1600; k++) {
29                         if (a[t] != -1 && now != a[t]) continue;
30                         if ((now < i && j == 2) || (now * (t - 1) >
k)) continue;
31                         if (now < i)
32                             dp[t][now][j + 1][k + now] = (dp[t][now][j + 1]
[k + now] + dp[t - 1][i][j][k]) % mod;
33                         else dp[t][now][1][k + now] = (dp[t - 1][i][j][k] +
dp[t][now][1][k + now]) % mod;
34                     }
35     }
36 }
37 ll ans = 0;
38 for (int i = 0; i <= 40; i++)
39     for (int j = 0; j < 3; j++)
40         for (int k = 0; k <= 1600; k++)
41             { ans += dp[n][i][j][k];
42             ans %= mod;
43         }
44 cout << ans << '\n';
45 }
```

简介

看题就行。

solve

1. 观察所有解空间，枚举任何解。对于一组具体的方案。可以先考察问题的解决顺序，优秀子集。各种问题的解决顺序，必然满足一些上界。

1. 优秀子集这里自定义为：所有枚举的解都可以在这个子集中找到更优的等效解。

顺序的探究：

对于一组解中，任意相邻的任务(假设存在两个以上的任务。不妨标记其为1，2。其中分别完成时间为 t_1, t_2 ，价值损失速度为 p_1, p_2 。开始做第一个任务时候。其分数和为sum。

关注两个量： $s_{1,2}$ 表示先1后2。 $s_{2,1}$ 反之。

$$\begin{aligned} s_{12} &= sum - t_1 p_1 - (t_1 + t_2) p_2 \\ s_{21} &= sum - t_2 p_2 - (t_1 + t_2) p_1 \\ s_{12} - s_{21} &= t_2 p_1 - t_1 p_2. \\ s_{12} - s_{21} &\geq 0 \\ \frac{p_1}{t_1} &\geq \frac{p_2}{t_2} \end{aligned} \tag{1}$$

利用这个递推序列排一个序。满足任何相邻的元素都满足上述关系。

2. 利用上述规则

1. 考察任何一个方案 a 。并且从优秀子集中的方案 b 映射到该方案。
2. 通过对 b 不断地调整。发现一种总分数一直减少的方案可以构造到该方案中。而一个方案只有一个结构。上述构造逐步计算的过程是等价过程。
3. 综上，在上述顺序的数组上做01背包即可。

$f_{i,j}$ 表示 j 状态下，完成 i 道题的最大方案。

code

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  const ll inf = 1E18;
7
8  ll t[N], p[N], f[N], v[N];
9  int id[N];
10
11 int main()
12 {
13     ios::sync_with_stdio(false);
14     cin.tie(0);
15     int n, T; cin >> n >> T;
16     for (int i = 1; i <= n; i++)
17         cin >> v[i];
```

```

18     for (int i = 1; i <= n; i++)
19         cin >> p[i];
20     for (int i = 1; i <= n; i++)
21         cin >> t[i];
22     iota(id, id + n + 1, 0);
23     sort(id + 1, id + 1 + n, [&](int i, int j) {
24         return 1.0 * p[i] / t[i] > 1.0 * p[j] / t[j];
25     });
26     // cerr << id[1] << '\n';
27     //状态设计表示刚好完成i任务。j时间下的最大成就。
28     ll ans = 0;
29     for (int x = 1; x <= n; x++) {
30         int i = id[x];
31         for (int j = T; j >= t[i]; j--) {
32             f[j] = max(f[j], f[j - t[i]] + v[i] - j * p[i]);
33         }
34     }
35     for (int i = 0; i <= T; i++) {
36         ans = max(ans, f[i]);
37     }
38     cout << ans << '\n';
39 }
40
41 /* stuff you should look for
42 * int overflow, array bounds
43 * special cases (n=1?)
44 * do smth instead of nothing and stay organized
45 * WRITE STUFF DOWN
46 * DON'T GET STUCK ON ONE APPROACH
47 */

```

生长思考：

1. 非常精彩的解空间压缩。

这里的解空间优化。转换成了一个01背包问题。关注了一个优秀的解集。而这个解集就是做01背包的解集。

和与或

[和与或\(nowcoder.com\)](https://nowcoder.com)

solve

观察出一个性质：从二进制的角度看，若干个数字相加，二进制串的某一个位上不能够有进位。

不断地枚举最终和的数字前缀：

对于任意二进制串前缀：

1. 当前位置为1时，那么要有一个A提供1。其它的数在该位上提供0。
2. 按照这样的枚举方法。由乘法计数原理，所有情况都考虑齐全。

观察枚举过程中，一些可以重复利用的信息。在枚举的过程中，对于 a_i 有两种属性——是否被限制（前缀是否贴着上界前缀走。）这决定了，当前情形下，这些数字是当前位置上可以取的数字。那么这时可以感受到，后面的选择和当前的0，1相关。（形象地看，就是后续发展的子树相同。可以记录这一类子树的叶子数）

3. 状态设计

$dp_{i,j}$ 表示当前在枚举第*i*位情况下，*j*表示限制情况（状态压缩的方法，如果*i*位上为0表示第*i*个数字的枚举被限制。为1反之）

4. 始化化：

初始化为-1。从dfs(62, 0)\$开始搜索。

5. 状态转移

如果sum当前的位置上选择0，其余所有项选0。此时那些原本被限制的，并且当前位上为1的下沉。

如果sum当前位置上选择1。有下面两种情况：

1. 非限制数字选择1.

1. 此时原本被限制的且上界的当前位上为1的 数字不再被限制。

2. 限制位数字且上界该位置上本来就有1。

1. 同上。

生长思考

1. 那么神奇，怎么思考？

2. 体会到了数位dp中贴上界问题在数位dp中存在的现象。多个数枚举制约下的限制，以及上界数的下沉。

code

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  const int mod = 1E9 + 9;
7
8  ll a[N], dp[100][1050];
9  int n;
10
11 ll dfs(int pos, int limit) {
12     if (pos == -1) return 1;
13     if (dp[pos][limit] != -1) return dp[pos][limit];
14
15     ll& res = dp[pos][limit];
16     res = 0;
17     //记录拿一些a在当前的pos上为1
18     int rec = 0;
19     for (int i = 0; i < n; i++)
20         if (a[i] & (1LL << pos)) {
21             rec |= 1LL << i;
22         }
23     res += dfs(pos - 1, limit | rec);
24     //考虑的当前pos上取1的情况。
25     //枚举哪一个1在这个位置上可以做出贡献
26     for (int i = 0; i < n; i++)
27         if (limit & (1LL << i))
28             res = (res + dfs(pos - 1, limit | rec)) % mod;
29         else if (rec & (1LL << i))
30             res = (res + dfs(pos - 1, (limit | rec) ^ (1LL << i))) % mod;
```

```

31     return res;
32 }
33 int main()
34 {
35     ios::sync_with_stdio(false);
36     cin.tie(0);
37
38     cin >> n;
39     for (int i = 0; i < n; i++) {
40         cin >> a[i];
41     }
42     memset(dp, -1, sizeof dp);
43     cout << dfs(61, 0) << '\n';
44 }
45
46 /* stuff you should look for
47 * int overflow, array bounds
48 * special cases (n=1?)
49 * do smth instead of nothing and stay organized
50 * WRITE STUFF DOWN
51 * DON'T GET STUCK ON ONE APPROACH
52 */

```

牛牛与数组

<https://ac.nowcoder.com/acm/problem/21738>

状态设计

1. 定义 $f_{i,j}$ 表示当前数组， i 位置上放置的是 j 元素。

状态转移

前缀和优化，再加上一个类似线性筛选的东西。类比埃氏筛法，复杂度为 $k \log k$

总复杂度为 $n * k * \log(k)$

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  const int mod = 1e9 + 7;
7  ll f[20][N];
8  int main()
9  {
10     ios::sync_with_stdio(false);
11     cin.tie(0);
12     int n, k; cin >> n >> k;
13     f[0][1] = 1;
14     for (int i = 1; i <= n; i++) {
15         //首先对这个数组的前缀和来一个小的处理：

```

```

16         ll sum = 0;
17         for (int j = 1; j <= k; j++)
18             sum = (sum + f[i - 1][j]) % mod;
19         for (int j = 1; j <= k; j++) {
20             f[i][j] = sum;
21             for (int t = j * 2; t <= k; t += j)
22                 f[i][j] = (f[i][j] - f[i - 1][t] + mod) % mod;
23         }
24     }
25     ll ans = 0;
26     for (int i = 1; i <= k; i++) {
27         ans = (ans + f[n][i]) % mod;
28     }
29     cout << ans << '\n';
30
31 }
32
33 /* stuff you should look for
34 * int overflow, array bounds
35 * special cases (n=1?)
36 * do smth instead of nothing and stay organized
37 * WRITE STUFF DOWN
38 * DON'T GET STUCK ON ONE APPROACH
39 */

```

牛牛的回文串

[牛牛的回文串 \(nowcoder.com\)](https://www.nowcoder.com/)

给定各种操作的代价。包括增，删，改。
求将该字符串变成回文串的最小代价。

10mins

这种解空间探究，无从下手：

1. 小规模问题是什么？体会不了。
2. 解空间中。各种操作应用都非常灵活，看上去毫无联系。

solve

1. 第一个问题，对于任何一个操作。就结果而言（比方说，换，删）并不只有一个方案。可以先改成某一些值再删除，所以得先把这些最优操作的代价求出来。（解结构的优化之一）。

1. 这一个操作可以通过floyd求取。

2. 处理掉一个字符

1. 直接的删掉。
2. 先转换再删掉。
3. 先在对称的位置增加一个字符，两个匹配掉（对其他字符不产生影响）。
4. 在对称位置增加一个字符，两个字符同时成k字符。

3. 字符转换：

1. 转换中间字符最小代价。

2. 关于状态设计：

$dp_{i,j}$ 表示将 $i...j$ 的子段变成回文串的最小花费。

3. 状态转移

对于计算 $dp_{i,j}$ ，发现几种策略。考虑各种方案。

1. s_i 和 s_j 进行匹配。
 1. 统一换成同一个字母。
2. 删掉 s_i
3. 删掉 s_j

code

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  const int N = 1E6 + 10;
5  const int oo = 1E9;
6  // #define int ll
7  string s;
8  //表示转换
9  ll cost[26][26], add[26], erase[26];
10 ll deal[26];
11 //表示删除。或者增加成一个删掉。
12 ll dp[100][100];
13 void floyd() {
14     for (int k = 0; k < 26; k++)
15         for (int i = 0; i < 26; i++)
16             for (int j = 0; j < 26; j++) {
17                 cost[i][j] = min(cost[i][k] + cost[k][j], cost[i][j]);
18             }
19     //计算删除的最小信息。
20     //删除的最小代价怎么统计?
21     for (int i = 0; i < 26; i++) {
22         deal[i] = erase[i];
23         for (int j = 0; j < 26; j++) {
24             //不借助中间字符。直接删增，匹配掉。
25             //改删
26             deal[i] = min(deal[i], cost[i][j] + erase[j]);
27             // 改改。
28             //增改改。
29             for (int k = 0; k < 26; k++)
30                 deal[i] = min(deal[i], add[j] + cost[i][k] + cost[j][k]);
31         }
32     }
33 }
34 void init() {
35     for (int i = 0; i < 26; i++) {
36         for (int j = 0; j < 26; j++) {
37             cost[i][j] = oo;
38         }
39         cost[i][i] = 0; add[i] = erase[i] = deal[i] = oo;
40     }
41 }
42 int main()
43 {
44     ios::sync_with_stdio(false);
45     cin.tie(0);
```

```

46     init();
47     int n; cin >> s >> n;
48     for (int i = 1; i <= n; i++) {
49         string ch; cin >> ch;
50         char a, b; ll c;
51         if (ch == "change") {
52             cin >> a >> b >> c;
53             cost[a - 'a'][b - 'a'] = min(cost[a - 'a'][b - 'a'], c);
54         }
55         else if (ch == "add") {
56             cin >> a >> c;
57             add[a - 'a'] = min(c, add[a - 'a']);
58         }
59         else {
60             cin >> a >> c;
61             erase[a - 'a'] = min(erase[a - 'a'], c);
62         }
63     }
64     floyed();
65     int sz = s.size();
66     for (int i = sz - 1; i >= 0; i--) {
67         for (int j = i + 1; j < sz; j++) {
68             dp[i][j] = oo;
69             if (s[i] == s[j]) dp[i][j] = dp[i + 1][j - 1];
70             dp[i][j] = min(dp[i][j], deal[s[i] - 'a'] + dp[i + 1][j]);
71             dp[i][j] = min(dp[i][j], deal[s[j] - 'a'] + dp[i][j - 1]);
72             for (int k = 0; k < 26; k++)
73                 dp[i][j] = min(dp[i][j], dp[i + 1][j - 1] + cost[s[i] - 'a']
74 [k] + cost[s[j] - 'a'][k]);
75         }
76         if (dp[0][sz - 1] == oo)
77             cout << -1 << '\n';
78         else cout << dp[0][sz - 1] << '\n';
79         //return OLL;
80     }
81
82     /* stuff you should look for
83     * int overflow, array bounds
84     * special cases (n=1?)
85     * do smth instead of nothing and stay organized
86     * WRITE STUFF DOWN
87     * DON'T GET STUCK ON ONE APPROACH
88     */

```

牛牛去买球

[牛牛去买球\(nowcoder.com\)](https://www.nowcoder.com/)

solve

简化问题， 找一个解：

1. 每一个包里面的红球、蓝色球的数量变化1。无论如何变化， 同色的球的数量至少为k。
2. 寻找满足上条件， 花费最低的解。

关注几种解结构：

1. 所有商品中， 红球的数量都减1。
2. 所有商品中， 蓝球的数量都减1。

只考虑这两种情况是不全面的。如下：

但是还是遗漏了一些解：反例如下：

```
1 | 2 10
2 | 6 5
3 | 4 4
4 | 1 1
```

该情况下， 无法做到两个减到最小值。但是两个都选了， 由于一个包里面球数量守恒， 无论怎么变化依然满足条件。其关键是两种球数目之和大于等于 $2 * k - 1$ 。很显然， 平均使得最大值最小， 依然大于等于k。

反之， 如果总的球数小于 $2 * k - 1$ 。解满足题意得充要条件是， 两种球中的一种， 减到极限了也依然满足数量大于等于k。前述两种情况下， 01背包得到的就是这种解的最优值。

1. 不妨设当前的子问题为蓝色球（最劣情况）。显然 f_i 通过背包求解后， 其解就是满足蓝色球减到极限了， 也满足题意的最小价值的解。那么我们枚举任意一种总球数小于 $2 * k - 1$ 的合法的（蓝色球减到极限， 也满足大于等于k条件）解。显然可以属于上述子问题解集（并不一定是最优解， 其实问题的解就是满足基本条件的解集中取最优值。）那么 f_i 更优。
2. 反之红色球亦然。

```
1 | #include<bits/stdc++.h>
2 | using namespace std;
3 | using ll = long long;
4 |
5 | const int N = 1E6 + 10;
6 | const ll inf = 1E15;
7 | int a[N] , b[N] , v[N];
8 | //当前的最小值是什么?
9 | //考虑了前i个商品。
10 | //选择了多少个商品。
11 | //是哪一类球更多
12 | ll f[N];
13 |
14 | int main()
15 | {
16 |     ios::sync_with_stdio(false);
17 |     cin.tie(0);
18 |
19 |     int n , k; cin >> n >> k;
20 |     for (int i = 1; i <= n; i++) cin >> a[i];
21 |     for (int i = 1; i <= n; i++) cin >> b[i];
22 |     for (int i = 1; i <= n; i++) cin >> v[i];
```

```

23     ll ans = inf;
24     memset(f , 0x3 , sizeof f);
25     f[0] = 0;
26     for (int i = 1; i <= n; i++) {
27         int w = a[i] - 1;
28         for (int j = 50000; j >= w ; j --)
29             f[j] = min(f[j] , f[ j - w] + v[i]);
30     }
31     for (int i = k; i <= 500000; i++) {
32         ans = min(ans , f[i]);
33     }
34     memset(f , 0x3 , sizeof f);
35     f[0] = 0;
36     for (int i = 1; i <= n; i++) {
37         int w = b[i] - 1;
38         for (int j = 50000; j >= w ; j --)
39             f[j] = min(f[j] , f[j - w] + v[i]);
40     }
41     for (int i = k; i <= 500000; i++) {
42         ans = min(ans , f[i]);
43     }
44     memset(f , 0x3 , sizeof f);
45
46     f[0] = 0;
47     for (int i = 1; i <= n; i++) {
48         int w = b[i] + a[i];
49         for (int j = 50000; j >= w ; j --)
50             f[j] = min(f[j] , f[j - w] + v[i]);
51     }
52     for (int i = k * 2 - 1; i <= 500000; i++) {
53         ans = min(ans , f[i]);
54     }
55     if (ans == inf) ans = -1;
56     cout << ans << '\n';
57 }
58
59 /* stuff you should look for
60 * int overflow, array bounds
61 * special cases (n=1?)
62 * do smth instead of nothing and stay organized
63 * WRITE STUFF DOWN
64 * DON'T GET STUCK ON ONE APPROACH
65 */

```

牛牛的计算机内存

[牛牛的计算机内存\(nowcoder.com\)](http://nowcoder.com)

简介

找出一种指令顺序，使得访存代价花费的时间最小：时间代价的计算方式是新的访问代价为：

solve

状压dp.

将选择状态压缩为 s ，第 $i-1$ 位置上为1，意味着这个任务已经选了。

定义 dp_s ：当选择情况为 s 。前面的最小代价。（选择的先后，不影响最终内存的使用情况。）

状态转移方程：

1. dp_s 枚举，上一个 s 的状态是什么。然后迁移即可。

完成的一些任务：

1. 字符串到二进制之间的转换。
2. 统计某些任务使用之后的，内存的剩余使用情况。

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  #define end en
5
6  const int N = 1E6 + 10;
7  int bit[100];
8  int end[1 << 20];
9  ll dp[1 << 20];
10
11 int main()
12 {
13     ios::sync_with_stdio(false);
14     cin.tie(0);
15
16     int n , m;
17     cin >> n >> m;
18     for (int i = 0; i < n; i++) {
19         string s; cin >> s;
20         int now = 0;
21         for (int j = 0; j < m; j++) {
22             if (s[j] == '0')continue;
23             else now |= (1 << j);
24         }
25         bit[i] = now;
26     }
27     auto f = [&](int x, int y) -> int{
28         //表示x。当前是选择哪一个任务？
29         //y表示前面的额基础情况。
30         //这个函数的目的就是计算出有多少的位置是不一样的。
31         int k = 0;
32         for (int i = 0; i < m; i++) {
33             if (((bit[x] >> i) & 1) && (end[y] >> i & 1) == 0)
34                 k++;
35         }
36         return k;
37     };
38     memset(dp , 0x3f , sizeof dp);
39     dp[0] = 0;
40     for (int s = 1; s < (1 << n); s++) {
41         for (int j = 0; j < n; j++) {
42             //说明当前有
```

```

43         if (s & (1 << j)) {
44             int k = f(j, s ^ (1 << j));
45             dp[s] = min(dp[s], dp[s ^ (1 << j)] + k * k);
46             end[s] = end[s ^ (1 << j)] | bit[j];
47         }
48     }
49 }
50 cout << dp[(1 << n) - 1] << '\n';
51
52 }
53
54 /* stuff you should look for
55 * int overflow, array bounds
56 * special cases (n=1?)
57 * do smth instead of nothing and stay organized
58 * WRITE STUFF DOWN
59 * DON'T GET STUCK ON ONE APPROACH
60 */

```

这是见过的一道最简单的状压dp了。

ATcoder

XYXX

[My Submissions - AtCoder Regular Contest 157](#)

solve

定义一些概念：

x 表示X的个数。 n 表示字符串长度。 pos_i 表示字符串中第 $(i + 1)$ 个Y的位置：

先分类讨论几种情况：

1. $x \geq k$ ，正常贪心。尽量使得YXXX之间的X被填满。设这种子串处理了， a 个。那么相比于其它一些非同类操作，其贡献至少增加 a 个。
2. $x < k$ ，该情况下处理完X字符后，还有一些剩余的修改机会。
 1. **证明**关于先处理完X字符，显然，如果没有处理完，就说明有两次得到Y的机会白白浪费了。可以拿出处理Y的操作来处理x。无论何种情形，处理完X总是最优的。
 2. 处理完x之后，那么剩下的操作次数在处理Y中如何分配？
 1. 使用等效转换的逆向思维。将问题转换成：让Y全部变成了X，然后一定量操作机会下，它它们变成Y。求最优解。于是转换成了第一种情况下同样的问题。

实现方式，先分类讨论，进行预处理，然后使用优先队列进行维护。

code

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4
5  const int N = 1E6 + 10;
6  int sum[N], pre[N];

```

```

7  int n , k , ans;;
8  string s;
9
10 void solve() {
11     vector<int>pos;
12     for (int i = 1; i <= n; i++)
13         if (s[i] == 'Y')pos.push_back(i);
14     if (pos.size() == 0) {
15         cout << max( k - 1 , 0) << '\n';
16         return;
17     }
18     priority_queue<int , vector<int> , greater<int>> que;
19     //对于连续段应该怎么吃处理?
20     for (int i = 1; i < (int)pos.size(); i++) {
21         if (pos[i] == pos[i - 1] + 1)ans++;
22         else que.push(pos[i] - pos[i - 1] - 1);
23     }
24     while (que.empty() == false && k) {
25         int top = que.top(); que.pop();
26         if (top <= k) {
27             ans += top + 1;
28             k -= top;
29         } else {
30             ans += k;
31             k = 0;
32         }
33     }
34     /*处理两端的情况*/
35     ans += k;
36     cout << ans << '\n';
37 }
38
39 int main()
40 {
41     ios::sync_with_stdio(false);
42     cin.tie(0);
43     cin >> n >> k;
44     cin >> s;
45     s = ' ' + s;
46     for (int i = 1; i <= n; i++) {
47         sum[i] = sum[i - 1] + (s[i] == 'X');
48     }
49     if (k > sum[n]) {
50         for (int i = 1; i <= n; i++) {
51             if (s[i] == 'X')s[i] = 'Y';
52             else s[i] = 'X';
53         }
54         k = n - k;
55     }
56     solve();
57
58 }
59 /* stuff you should look for
60 * int overflow, array bounds
61 * special cases (n=1?)

```

```
62 * do smth instead of nothing and stay organized
63 * WRITE STUFF DOWN
64 * DON'T GET STUCK ON ONE APPROACH
65 *2023/3/5 陈九日
66 */
```

区间动态规划进阶

ICPC Beijing 2017 J, Pangu and Stones

[ICPC Beijing 2017 J, Pangu and Stones - 题目 - Daimayuan Online Judge](#)

题目简介

1. 石子合并问题:

对一堆石子, 合并连续的长度为 $[L, R]$ 的石子。

和最简单的区间动态规划问题不同。这里对连续合并的石堆个数有限制。

solve

1. 显然不能简单的枚举每一堆的分界点。

定义一个状态:

$f_{i,j,k}$ 表示 i, j , 为和并为 k 堆的最小代价。

辅助状态的转移方程。

1. 枚举 i, j 。
2. 枚举 k 。
3. 枚举第一堆的尾部。

$$\begin{aligned} k \geq 2; f_{i,j,k} &= \min(f_{i,mid,1} + f_{mid+1,r,k-1}) \\ k = 1; f_{i,j,1} &= \min(f_{i,j,2\dots k} + sum_{l\dots r}) \end{aligned} \quad (2)$$

code

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using ll = long long;
4  #define f dp
5
6  const int N = 110;
7  const int inf = 1 << 29;
8  int n , L , R;
9  int f[N][N][N];
10 int sum[N];
11
12 void solve() {
13     for (int i = 1; i <= n ; i++) {
14         int x; cin >> x;
```



```

15         sum[i] = sum[i - 1] + x;
16     }
17     for (int i = 1; i <= n; i++)
18         for (int j = 1; j <= n; j++)
19             for (int k = 1; k <= n; k++)
20                 f[i][j][k] = inf;
21     for (int d = 0; d <= n; d++) {
22         for (int l = 1; l + d <= n; l++) {
23             int r = l + d;
24             if (d == 0) {
25                 f[l][r][1] = 0;
26             } else {
27                 for (int k = 2; k <= n; k++) {
28                     for (int mid = l; mid < r; mid++) {
29                         f[l][r][k] = min(f[l][r][k], f[l][mid][1] + f[mid +
30 1][r][k - 1]);
31                     }
32                     if (k >= L && k <= R) f[l][r][1] = min(f[l][r][k], f[l]
33 [r][1]);
34                 }
35                 f[l][r][1] += sum[r] - sum[l - 1];
36             }
37         }
38     }
39     if (f[1][n][1] >= inf) cout << 0 << '\n';
40     else cout << f[1][n][1] << '\n';
41 }
42
43 int main()
44 {
45     ios::sync_with_stdio(false);
46     cin.tie(0);
47     while (cin >> n >> L >> R) solve();
48 }

```

2023/3/5
