历年 CSP 题目解析

仅为参考练习所用

作者：lonlyn
组织：Shanxi University Algorithm Group
时间：December 30, 2021
版本：1.0

特别声明

该书仅供内部学习使用，如果有侵权请联系作者。

信息学竞赛的发展，吸引越来越多的人加入了“卷”的行列。CCF CSP 的历年题解在网上也是随处可见，但题解质量参差不齐。很多题解只有标准答案，缺少题目分析；更有甚者无法通过答案，充满了分号大小写问题等错误。

本书的目的是为了实现以下几点：

- 提供规范的代码程序。这里的规范，既要具有程序的可读性，也要具备考场的简易性。
- 提供多样的解题思路。有些时候，网上的大佬往往一语道破问题求解的思路，但怎么想到的却往往不提。这里力求从部分分开始，逐渐深入，汇集众人智慧，逐步解决难题。
- 提供筛选的额外补充。做一道题的目的不是只做一道题，而是可以做到举一反三，但我们常常忽略这一点。

感谢 ElegantLaTeX 提供如此精美的模板，希望这本书能够给大家带来帮助。

lonlyn

December 30, 2021

目录

第 1 章 CCF CSP 认证总览

待补充。

第 2 章 第 23 次认证（2021 年 09 月）

2.1 题目及设计知识点

题目编号	题目名称	知识点
1	数组推导	模拟
2	非零段划分	模拟，数学
3	脉冲神经网络	模拟
4	收集卡牌	状压 dp
5	箱根山岳险天下	树链剖分，动态树

2.2 202109-1 数组推导

2.3 202109-2 非零段划分

2.4 202109-3 脉冲神经网络

2.5 202109-4 收集卡牌

2.6 202109-5 箱根山岳险天下

第 3 章 第 24 次认证（2021 年 12 月）

3.1 题目及涉及知识点

题目编号	题目名称	知识点
1	序列查询	数学
2	序列查询新解	数学
3	登机牌条码	模拟，多项式除法
4	磁盘文件操作	线段树
5	极差路径	树分治

3.2 202112-1 序列查询

3.2.1 50% 数据——模拟

3.2.1.1 思路

模拟一下这个过程，计算出每一个 $f(i)$ 后加起来即可。

考虑针对确定的 x，如何求解 $f(x)$。我们可以从小到大枚举 A 中的数，枚举到第一个大于等于 x 的数即可。注意末尾的判断。

枚举 x 时间复杂度 $\mathbf{O}(N)$，计算 $f(x)$ 时间复杂度 $\mathbf{O}(n)$，整体时间复杂度 $\mathbf{O}(nN)$。

3.2.1.2 C++ 实现

待补充。

3.2.2 100% 数据——利用 $f(x)$ 单调性

3.2.2.1 思路

为了方便，设 $f(n+1)=\infty$。

通过模拟，可以得到一个显然的结论：

定理 3.1 ($f(x)$ 的单调性)

对于 $x,y\in[0,N)$，若 $x\le y$，则 $f(x)\le f(y)$。

那么，我们可以从小到大枚举 x，同时记录目前 $f(x)$ 的值，设为 y，那么 A_{y+1} 是第一个大于 x 的数。当需要计算 $f(x+1)$ 的时候，我们从小到大依次判断 $A_{y+1},A_{y+2},\cdots$ 是否满足条件，直到遇到第一个大于 $f(x+1)$ 的数 A_z，那么 $f(x+1)=z-1$。之后，在 $f(x+1)$ 的基础上以同样的步骤求 $f(x+2)$，直到求完所有的值。

考虑该算法的时间复杂度，枚举 x 的复杂度是 $\mathbf{O}(N)$，而 A 数组中每个数对多被枚举一次，枚举所有 x 的整体复杂度 $\mathbf{O}(n)$，可以得到整体复杂度 $\mathbf{O}(N+n)$。

3.2.2.2 C++ 实现

```cpp
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
#define ll long long
#define il inline
const int maxn = 210;
int n, N;
int a[maxn];
ll ans = 0;
int main() {
    scanf("%d%d", &n, &N);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
```

```
    }
    int cur = 0;
    for (int i = 0; i < N; ++i) {
        while (cur < n && a[cur + 1] <= i)
            ++cur;
        ans += cur;
    }
    printf("%lld\n", ans);
    return 0;
}
```

3.2.3 100% 数据——阶段求和

3.2.3.1 思路

在提示中，指出了可以将 $f(x)$ 相同的值一起计算。现在需要解决的问题是如何快速确定 $f(x)$ 值相等的区间。

通过观察和模拟可以发现，随着 x 增大，$f(x)$ 只会在等于某个 A 数组的值时发生变化。更具体的说，对于某个属于 A 数组的值 A_i 来说，$[A_i, A_{i+1} - 1]$ 间的 $f(x)$ 值是相同的，这样的数共有 $A_{i+1} - A_i$ 个。

也可以以另一种方式理解：对于一个值 y，考虑有多少 x 满足 $f(x) = y$。当 $x < A_y$ 时，$f(x) < y$，当 $x \geq A_{y+1}$ 时，$f(x) > y$。只有 $x \in [A_y, A_{y+1}]$ 时才能得到 $f(x) = y$。

得到范围后，我们就可以根据 A 数组来进行求和计算。

考虑 $f(x) = n$ 的处理：我们可以得知满足 $f(x) = n$ 的 x 共有 $N - A_n$ 个，根据上文推算，我们可以将 A_{n+1} 设置为 $A_n + (N - A_n) = N$ 即可等效替代。

时间复杂度 $\mathbf{O}(n)$。

3.2.3.2 C++ 实现

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
using namespace std;
#define ll long long
#define il inline
const int maxn = 210;
int n, N;
int a[maxn];
ll ans = 0;
int main() {
    scanf("%d%d", &n, &N);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }
    a[n + 1] = N;
    for (int i = 1; i <= n + 1; ++i) {
        // 处理区间 [A(i-1),A(i)] 的 f(x) 值的和
```

```
        ans += 1ll * (a[i] - a[i - 1]) * (i - 1);
    }
    printf("%lld\n", ans);
    return 0;
}
```

3.3 202112-2 序列查询新解

3.3.1 与上一题的比较

1. 上一题是求和，而本题要求求绝对值的和，无法转化为两者求差的形式。
2. $f(x), g(x)$ 的变化是各自独立的，当 $f(x)$ 改变时，$g(x)$ 可能不变，也可能改变；$g(x)$ 对 $f(x)$ 也是如此。
3. 对于所有数据点，n 和 N 都增大了许多。如果复杂度涉及到 n，则最多预计为 $\mathbf{O}(n\log n)$ 级别；如果涉及到 N，则必须是亚线性级别。

3.3.2 70% 数据——计算出每个 $f(x), g(x)$ 的值

3.3.2.1 思路

由于 1,2 条限制，我们无法直接对 $f(x), g(x)$ 分别进行处理。但我们可以求出每个 $f(x), g(x)$ 的值，再计算求和即可。

$f(x)$ 的计算同第一问，任意方法皆可。单个 $g(x)$ 的值可以直接 $\mathbf{O}(1)$ 求得。

3.3.2.2 C++ 实现

待补充

3.3.3 100% 数据——对 $f(x), g(x)$ 都相同的区间进行求和处理

3.3.3.1 思路

注：为了防止混淆，将题目中的 r 改为 $ratio$。

假设 $f(x)$ 一共有 x 种取值，$g(x)$ 一共有 y 种取值。直接来看 $f(x), g(x)$ 的组合一共有 xy 种，但注意到 $f(x), g(x)$ 都是单调不递减函数，所以真正的组合只有 $x+y$ 种。

在第一题中已经说明 $f(x)$ 的取值范围为 $[0, n]$，在 $\mathbf{O}(n)$ 级别。考虑 $g(x)$ 的取值情况，将 $ratio$ 的公式带入可以得到 $g(x) = \lfloor \frac{x}{ratio} \rfloor = \lfloor \frac{x}{\lfloor \frac{N}{n+1} \rfloor} \rfloor$。由于 x 取值有 N 种，所以 $g(x)$ 的取值是 $\mathbf{O}(\frac{N}{\frac{N}{n+1}}) = \mathbf{O}(n)$ 级别的。所以，整体复杂度为 $\mathbf{O}(n+n) = \mathbf{O}(n)$。

提示 有些时候，题目给出的某些量的值会比较特殊（如本题 $ratio = \lfloor \frac{N}{n+1} \rfloor$），代表着出题人可能想要隐藏某些做法，但不得不为了让时间复杂度正确而妥协。在没有思路的时候，可以作为突破口。

考虑范围问题：假设当前左端点为 l，如何找到右端点 r，满足 $f(l) = f(l+1) = \cdots = f(r), g(l) = g(l+1) \cdots = g(r)$ 且 $f(l) \neq f(r+1)$ or $g(l) \neq g(r+1)$。我们可以对 $f(x), g(x)$ 分别考虑:

1. 对于 $f(x)$ 而言，第一个满足 $f(x) = f(l) + 1$ 的 x 值为 A_{f_l+1}。
2. 对于 $g(x)$ 而言，因为分母 $ratio$ 是固定的，所以值相同的区间长度也是固定为 $ratio$。我们不妨将 $g(x)$ 值相同的数字为一组，则可以得到 $[0, ratio-1], [ratio, 2\cdot ratio-1], \cdots, [n\cdot ratio, (n+1)\cdot ratio-1], \cdots$ 这样的分组序列，每组的 $g(x)$ 取值为 $0, 1, \cdots, n, \cdots$。可以发现，对于一个数 l，其所属的分组是 $\lfloor \frac{l}{ratio} \rfloor$，也即 $g(l)$；而下一组开始的第一个数为 $ratio\cdot(g(l)+1)$，从而可以得到右端点 $r = ratio\cdot(g(l)+1)-1$。
3. 在 $f(x), g(x)$ 计算得到的右端点中，选择较小的一个作为计算的右端点。

计算完一段后，设 $l = r+1$ 继续计算下一段，直到结束。时间复杂度 $\mathbf{O}(n)$。

3.3.3.2 C++ 实现

```
#include <algorithm>
#include <cmath>
#include <cstdio>
```

```cpp
#include <cstring>
#include <iostream>
using namespace std;
#define ll long long
const int maxn = 100010;
int n, N;
int a[maxn];
int rat, f, g;
ll ans = 0;
int main() {
    scanf("%d%d", &n, &N);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
    }
    rat = N / (n + 1); // 为了防止冲突，题目中 r 改为 rat
    int cur = 0;      // 用来计算 f(x)
    bool flag = false; // 如果需要更新 f(x) 值，则 flag = true
    for (int l = 0, r; l < N; l = r + 1) {
        flag = false;
        // 利用 f(x) 的值确定 r 的范围
        if (cur < n)
            r = a[cur + 1] - 1;
        else
            r = N - 1;
        // 判断 f(x), g(x) 谁先变化，选择较小的区间
        if ((l / rat + 1) * rat - 1 < r) {
            // 如果 g(x) 先变化，则改为选择 g(x)
            r = (l / rat + 1) * rat - 1;
        } else {
            // 如果 f(x) 先变化，则确定选择 f(x)，计算后更新 f(x)
            flag = true;
        }
        // [l, r] 区间内的值是相等的，可以求和
        ans += 1ll * (r - l + 1) * abs(l / rat - cur);
        // 更新 f(x) 的值
        if (flag)
            ++cur;
    }
    printf("%lld\n", ans);
    return 0;
}
```

3.3.4 100% 思路——以 $f(x)$ 为单位，讨论内部 $g(x)$ 求和

感谢 DoctorLazy 提供的宝贵思路，原文可以查看 第二题 100 分题解 by DoctorLazy.md 。

3.3.4.1 思路

和上文同样的思路，我们需要进行区间求和来降低复杂度。一种思路是，整体上对 $f(x)$ 进行求和，而在内部对 $g(x)$ 的情况进行分类讨论。

我们单独考虑每一个 $f(x)$ 的区间，每个区间上 $f(x)$ 的值相同。可以观察到，对于一个区间上的下标 i，可能存在 $g(i) \geq f(i)$，也可能存在 $g(i) < f(i)$。求绝对值时，前者用 $g(x) - f(x)$，后者用 $f(x) - g(x)$。

观察到，由于 $g(x)$ **单调不减**的性质，我们可以得到：对于该区间，一定存在一个下标 p，如同一个分界线，当 $i \geq p$ 时，有 $g(i) \geq f(i)$，当 $i < p$，有 $g(i) < f(i)$。这样，就把该区间分成了两个“小区间”。我们就可以用“乘法思想”来加速两个“小区间”的求解了。

更规范些，用 $contribution(i)$ 代表区间 $[A_i, A_{i+1})$ 对答案的贡献，用 $len(l,r) = r - l + 1$ 代表区间长度，用公式可以表达为：

$$contribution(i) = len(A_i, p-1) \times f(x) - \sum_{x=A[i]}^{p-1} g(x) \\ + \sum_{x=p}^{A_{i+1}-1} g(x) - len(p, A_{i+1}-1) \times f(x)$$

上式中，$f(x)$ 是一个常数，所以乘以“小区间”的长度即可；$g(x)$ 的求和，大家可以发挥数学思维：因为 $g(x)$ 其实非常规律，它的每一块是定长的，我们可以通过除法和取余来确定相同值的数量，再利用乘法思想求和，灵活实现，在 $\mathbf{O}(n)$ 时间内求出即可。p 的具体值可以通过在 $g(x)$ 中二分查找，$\mathbf{O}(\log n)$ 时间内求出，n 为区间的长度。

一个例子:

x	$\cdots$	4	5	6	7	$\cdots$
$f(x)$	$\cdots$	2	2	2	2	$\cdots$
$g(x)$	$\cdots$	1	1	2	2	$\cdots$

上面的表格截取了一个小区间，$f(x)$ 的值固定 2，$p = 6$，那么 p 的左边用 $f(x)-g(x)$，p 的右边用 $g(x)-f(x)$。

当然，有一个特殊的边界情况，那就是该区间上有可能所有的 $g(x)$ 都绝对大于或小于 $f(x)$，这时候 p 可能会在区间外。该情况大家可以对 p 设置初值，然后在写完二分后加以判断即可。

3.3.4.2 C++ 实现

```cpp
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <map>
#include <queue>
#include <vector>
using namespace std;
typedef long long LL;
typedef unsigned long long uLL;
typedef pair<int, int> pii;
```

```
const int mod = 1e9 + 7;
const int maxn = 1e5 + 5;

LL N, n;
LL arr[maxn];  // 题中 A 数组
vector<LL> f;  // 存储每个区间上f的值
vector<pii> pos; // 存储每个区间的边界，是左闭右闭
LL r, ans;     // 题中的 r，ans为计算的答案

// 下面的函数用于计算g(x)在区间上的和
// 这一步比较细，具体可以灵活实现
// 下面的思路还是比较冗杂的
LL totG(LL be, LL ed) {
    // 右边界小于左边界，返回0
    if (ed < be) {
        return 0;
    }
    // 两边界重合，返回一个g值
    if (be == ed) {
        return be / r;
    }
    // 如果两边界g值相同，返回该值乘以区间长度
    if (be / r == ed / r) {
        return (be / r) * (ed - be + 1);
    }
    // 将区间分为三部分，分别累计
    LL tot = 0;
    // 对于左边界，其值为be/r,数目为 r - be % r
    tot += (r - (be % r)) * (be / r);
    // 对于右边界，其值为ed/r，数目为 ed % r + 1
    tot += (ed % r + 1) * (ed / r);
    // 对于不在边界上的g值，我们用等差数列求和公式
    if (ed / r - be / r > 1) {
        be = be / r + 1;
        ed = ed / r - 1;
        tot += r * ((be + ed) * (ed - be + 1) / 2);
    }
    return tot;
}

void solve() {
    // 输入
    scanf("%lld%lld", &n, &N);
    r = N / (n + 1);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &arr[i]);
    }
    // 根据数组，生成f(x)的每个区间，值存入f，区间边界存入pos
    LL last = 0ll; // 记录上一个边界
```

```
    // 这里的逻辑参考第一题
    for (int i = 1; i <= n; i++) {
        if (arr[i] > arr[last]) {
            f.push_back(last);
            pos.push_back({arr[last], arr[i] - 1});
            last = i;
        }
    }
    // 单独处理下最后一个区间，即[A[n],N-1]
    f.push_back(n);
    pos.push_back({arr[last], N - 1});
    // 对于每个f区间，将g分成两个小区间
    int si = f.size();
    for (int i = 0; i < si; i++) {
        LL num = f[i];
        LL be = pos[i].first;
        LL ed = pos[i].second;
        LL length = ed - be + 1;
        // 因为be和ed在二分过程其值发生变化，所以下面再存一份
        LL bbe = be, eed = ed;
        // 下面使用二分，在g中寻找分界p
        LL pin = -1;
        while (be <= ed) {
            LL mid = (be + ed) / 2;
            LL cur = mid / r;
            if (cur >= num) {
                pin = mid;
                ed = mid - 1;
            } else {
                be = mid + 1;
            }
        }
        // 如果f的值一直大于g，p值不会被二分的过程赋值，所以还是初值
        if (pin == -1) {
            ans += num * length - totG(bbe, eed);
        } else {
            // 左边的用f-g，右边用g-f。就算g的值一直大于f，即左边的部分长度为0
            ans += num * (pin - bbe) - totG(bbe, pin - 1);
            ans += totG(pin, eed) - num * (eed - pin + 1);
        }
    }
    printf("%lld", ans);
}

int main() {
    int t;
    t = 1;
    while (t--) {
        solve();
```

```
    }
    return 0;
}
```

3.4 202112-3 序列查询新解

3.4.1 40% 数据——直接模拟

3.4.1.1 思路

这一部分数据满足 $s = -1$，即校验码为空。我们按照题目要求进行对应操作即可，大体分为以下几个步骤：

1. 得到数字序列，注意不同模式的切换以及最后的补全。
2. 将得到的数字转换为码字。
3. 根据有效数据区每行能容纳的码字数 w 及目前码字个数，在末尾补充码字。注意不要忽略长度码字。
4. 输出结果。

3.4.1.2 C++ 实现

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>
using namespace std;
const int maxn = 210;
const int mod = 929;
int w, lev;
char s[100010];
int n;
vector<int> numberList; // 数字序列
vector<int> codeWord; // 码字序列
int currentMode; // 目前编码器模式。0:大写模式 1:小写模式 2:数字模式
void checkmode(char c) {
    /*
        检查将要输出的下个字符与目前模式是否匹配，
        若不匹配，则输出对应更改模式步骤。
    */
    if (c >= '0' && c <= '9') {
        if (currentMode != 2) {
            numberList.push_back(28);
            currentMode = 2;
        }
    } else if (c >= 'a' && c <= 'z') {
        if (currentMode != 1) {
            numberList.push_back(27);
            currentMode = 1;
        }
    } else if (c >= 'A' && c <= 'Z') {
        if (currentMode == 1) {
            numberList.push_back(28);
            numberList.push_back(28);
```

```
            currentMode = 0;
        }
        if (currentMode == 2) {
            numberList.push_back(28);
            currentMode = 0;
        }
    }
}
int main() {
    scanf("%d%d", &w, &lev); // lev 表示校验级别
    scanf("%s", s);
    n = strlen(s);
    // 步骤一：得到数字序列
    currentMode = 0; // 初始为大写模式
    for (int i = 0; i < n; ++i) {
        checkmode(s[i]);
        if (s[i] >= '0' && s[i] <= '9') {
            numberList.push_back(s[i] - '0');
        } else if (s[i] >= 'a' && s[i] <= 'z') {
            numberList.push_back(s[i] - 'a');
        } else if (s[i] >= 'A' && s[i] <= 'Z') {
            numberList.push_back(s[i] - 'A');
        }
    }
    if (numberList.size() % 2)
        numberList.push_back(29);
    // 步骤二：转换为码字
    for (int i = 0; i < numberList.size(); i += 2) {
        codeWord.push_back(30 * numberList[i] + numberList[i + 1]);
    }
    // 步骤三：补充码字
    while ((1 + codeWord.size()) % w != 0) {
        codeWord.push_back(900);
    }
    // 步骤四：输出结果
    codeWord.insert(codeWord.begin(), codeWord.size() + 1);
    for (int i = 0; i < codeWord.size(); ++i) {
        printf("%d\n", codeWord[i]);
    }
    return 0;
}
```

3.4.2 100% 数据——模拟 + 多项式除法

3.4.2.1 思路

这部分数据要求我们对校验码进行处理，所以步骤变为：

1. 得到数字序列，注意不同模式的切换以及最后的补全。

2. 将得到的数字转换为码字。
3. 根据有效数据区每行能容纳的码字数 w、目前码字个数以及**校验码的位数**，在末尾补充码字。注意不要忽略长度码字。
4. 输出数据码部分结果。
5. **计算得出校验码，并输出。**

校验码的位数能比较方便得出，关键在于校验码的计算。考虑关键公式：

$$x^k d(x) \equiv q(x)g(x) - r(x)$$

其中 $d(x)$ 是 $n-1$ 次多项式（已知），$g(x)$ 是 k 次多项式（已知），未知项有 $q(x), r(x)$，其中 $r(x)$ 为所求。考虑消去 $q(x)$ 的影响：可以在两端同时对 $g(x)$ 取余，则 $q(x)g(x)$ 项会被直接消去，可以化所求式为：

$$x^k d(x) \equiv -r(x) \mod q(x)$$

所以目前问题转化为求解 $x^k d(x) \mod q(x)$。

定义 3.1 (多项式带余除法)

若 $f(x)$ 和 $g(x)$ 是两个多项式，且 $g(x)$ 不等于 0，则存在唯一的多项式 $q(x)$ 和 $r(x)$，满足：

$$f(x) = q(x)g(x) + r(x)$$

其中 $r(x)$ 的次数小于 $g(x)$ 的次数。此时 $q(x)$ 称为 $g(x)$ 除 $f(x)$ 的商式，$r(x)$ 称为余式。

♣

定义 3.2 (多项式长除法)

求解多项式带余除法的一种方法，步骤如下：

1. 把被除式、除式按某个字母作降幂排列，并把所缺的项用零补齐；
2. 用被除式的第一项除以除式第一项，得到商式的第一项；
3. 用商式的第一项去乘除式，把积写在被除式下面（同类项对齐），消去相等项，把不相等的项结合起来；
4. 把减得的差当作新的被除式，再按照上面的方法继续演算，直到余式为零或余式的次数低于除式的次数时为止。

♣

下面展示的是一个多项式长除法的例子：

$$
\begin{array}{rl}
 & 6x^5 + 286x^4 + 4111x^3 + 42431x^2 + 398624x + 3638751 \\
\hline
x^2 - 12x + 27) & 6x^7 + 214x^6 + 841x^5 + 821x^4 + 449x^3 + 900x^2 \\
 & -6x^7 + 72x^6 - 162x^5 \\
\hline
 & 286x^6 + 679x^5 + 821x^4 \\
 & -286x^6 + 3432x^5 - 7722x^4 \\
\hline
 & 4111x^5 - 6901x^4 + 449x^3 \\
 & -4111x^5 + 49332x^4 - 110997x^3 \\
\hline
 & 42431x^4 - 110548x^3 + 900x^2 \\
 & -42431x^4 + 509172x^3 - 1145637x^2 \\
\hline
 & 398624x^3 - 1144737x^2 \\
 & -398624x^3 + 4783488x^2 - 10762848x \\
\hline
 & 3638751x^2 - 10762848x \\
 & -3638751x^2 + 43665012x - 98246277 \\
\hline
 & 32902164x - 98246277
\end{array}
$$

得到求解多项式带余除法的步骤后，考虑求解 $r(x)$ 的步骤：

1. 计算 $g(x) = (x-3)(x-3^2)\cdots(x-3^k)$；
2. 计算 $x^k d(x)$；
3. 计算 $x^k d(x) \mod g(x)$，得到 $-r(x)$；
4. 对得到的每一项取反即可得到 $r(x)$。

计算 $g(x)$：考虑到每一次多项式乘以的因子都是 $(x-a)$ 的格式，所以可以把 $A(x-a)$ 的多项式相乘转化为 $xA - aA$ 的格式。xA 可以通过整体移项实现；在移项后，原本在 x^i 的系数成为 x^{i+1} 的系数，所以可以在一个数组上，从低位到高位依次计算，得到结果。

计算 $x^k d(x)$：这部分比较简单，将低 k 位的系数赋 0，再将已计算出的数据位放入对应位置即可。

计算 $x^k d(x) \mod g(x)$：利用上文提到的多项式长除法即可。本题 $g(x)$ 的最高位系数恒为 1，简化了计算。

3.4.2.2 C++ 实现

```cpp
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <vector>
using namespace std;
const int maxn = 210;
const int mod = 929;
int w, lev;
char s[100010];
int n;
vector<int> numberList;
vector<int> codeWord;
int verifyCodeLen; // 校验码长度
int currentMode;
void checkmode(char c) {
    /*
```

```
        检查将要输出的下个字符与目前模式是否匹配，
        若不匹配，则输出对应更改模式步骤。
    */
    if (c >= '0' && c <= '9') {
        if (currentMode != 2) {
            numberList.push_back(28);
            currentMode = 2;
        }
    } else if (c >= 'a' && c <= 'z') {
        if (currentMode != 1) {
            numberList.push_back(27);
            currentMode = 1;
        }
    } else if (c >= 'A' && c <= 'Z') {
        if (currentMode == 1) {
            numberList.push_back(28);
            numberList.push_back(28);
            currentMode = 0;
        }
        if (currentMode == 2) {
            numberList.push_back(28);
            currentMode = 0;
        }
    }
}
vector<int> get_gx(int k) {
    // 根据 k 计算 g(x)
    vector<int> res;
    res.push_back(mod - 3);
    res.push_back(1);
    int a0 = 3;
    for (int i = 2; i <= k; ++i) {
        a0 = (a0 * 3) % mod;
        res.insert(res.begin(), 0); // 在最低位插入 1，即整体次数 +1
        for (int j = 0; j < i; ++j) {
            res[j] = (res[j] - (a0 * res[j + 1]) % mod + mod) % mod;
        }
    }
    return res;
}

void get_verify_code() {
    // 计算校验码并输出
    vector<int> tmp;
    vector<int> g = get_gx(verifyCodeLen);
    // 初始化 x^kd(x)
    for (int i = 1; i <= verifyCodeLen; ++i) {
        tmp.push_back(0);
    }
```

```
    for (int i = codeWord.size() - 1; i >= 0; --i) {
        tmp.push_back(codeWord[i]);
    }
    // 多项式长除法计算结果
    for (int i = tmp.size() - 1; i >= verifyCodeLen; --i) {
        int val = tmp[i];
        for (int j = 0; j < g.size(); ++j) {
            tmp[i - j] =
                (tmp[i - j] - (val * g[g.size() - 1 - j]) % mod + mod) % mod;
        }
    }
    // 将 -r(x) 转化为 r(x)
    for (int i = 0; i < verifyCodeLen; ++i) {
        // 注意：不能直接 mod - tmp[i]，因为 tmp[i] 可能为 0
        tmp[i] = (mod - tmp[i]) % mod;
    }
    // 输出结果
    for (int i = verifyCodeLen - 1; i >= 0; --i) {
        printf("%d\n", tmp[i]);
    }
}
int main() {
    scanf("%d%d", &w, &lev);
    scanf("%s", s);
    n = strlen(s);
    // 步骤一：得到数字序列
    currentMode = 0;
    for (int i = 0; i < n; ++i) {
        checkmode(s[i]);
        if (s[i] >= '0' && s[i] <= '9') {
            numberList.push_back(s[i] - '0');
        } else if (s[i] >= 'a' && s[i] <= 'z') {
            numberList.push_back(s[i] - 'a');
        } else if (s[i] >= 'A' && s[i] <= 'Z') {
            numberList.push_back(s[i] - 'A');
        }
    }
    if (numberList.size() % 2)
        numberList.push_back(29);
    // 步骤二：转换为码字
    for (int i = 0; i < numberList.size(); i += 2) {
        codeWord.push_back(30 * numberList[i] + numberList[i + 1]);
    }
    if (lev == -1)
        verifyCodeLen = 0;
    else {
        verifyCodeLen = 1;
        for (int i = 0; i <= lev; ++i) {
            verifyCodeLen *= 2;
```

```
        }
    }
    // 步骤三：补充码字
    while ((1 + verifyCodeLen + codeWord.size()) % w != 0) {
        codeWord.push_back(900);
    }
    codeWord.insert(codeWord.begin(), codeWord.size() + 1);
    // 步骤四：输出数据码结果
    for (int i = 0; i < codeWord.size(); ++i) {
        printf("%d\n", codeWord[i]);
    }
    // 步骤五：如果有校验码，则计算并输出
    if (verifyCodeLen != 0) {
        get_verify_code();
    }
    return 0;
}
```

3.5 202112-4 磁盘文件操作

3.6 202112-5 极差路径