GESP C++ 六级认证（三）

动态规划专题 - 代码实战

相关题目

- P10250 [GESP样题 六级] 下楼梯(跳过过于简单)
- P10262 [GESP样题 六级] 亲朋数
- P10108 [GESP202312 六级] 闯关游戏
- P10721 [GESP202406 六级] 计算得分
- P11246 [GESP202409 六级] 小杨和整数拆分
- P14075 [GESP202509 六级] 划分字符串
- P13015 [GESP202506 六级] 学习小组

P10262 [GESP样题 六级] 亲朋数

题意

给定一个长度为 L 的数字串 S 和一个正整数 p，求 S 的所有连续子串中，对应数值是 p 的倍数的子串个数（允许有前导零）。

数据范围：$2 \le p \le 128$，$1 \le L \le 10^6$。

分析

1. **问题转化**：需要统计所有连续子串中能被 p 整除的个数。直接枚举所有子串会达到 $O(L^2)$，对于 $L \leq 10^6$ 不可行。

2. **观察**：利用模运算的性质，对于子串 $S[i..j]$ 对应的数值，可以通过前缀和的思想计算。具体地，设 $num(i,j)$ 表示子串 $S[i..j]$ 对应的数值，则：

$$num(i,j) = \sum_{k=i}^{j} (S[k] -' 0') \times 10^{j-k}$$

如果 $num(i,j) \equiv 0 \pmod{p}$，则该子串是亲朋数。

3. **思路**：

- 定义 $dp[k]$ 表示以当前数字为结尾的子串中，模 p 余数为 k 的子串个数
- 遍历字符串，对于每个数字 d：
  - 新的余数可以通过旧余数 j 计算：$(j \times 10 + d) \bmod p$
  - 当前数字单独作为一个子串：余数为 $d \bmod p$
- 每次统计余数为 0 的子串个数

4. **正确性**：

- 对于位置 i 的数字 d，以它结尾的子串包括：
  - 长度为 1 的子串：d
  - 长度 > 1 的子串：由前 $i-1$ 个字符的某个后缀加上 d 组成
- 通过维护模 p 的余数分布，可以高效计算新增的亲朋数

时间复杂度：$O(L \times p)$，在给定数据范围内可行。

```cpp
const int N = 128;
int p, dp[N], lstdp[N];
long long ans;
string s;

int main() {
    cin >> p >> s;
    for (char ch : s) {
        int d = (ch - '0') % p; // 当前数字模p的余数
        // 保存上一轮的状态
        for (int j = 0; j < p; j++)
            lstdp[j] = dp[j], dp[j] = 0;

        // 状态转移：从上一轮的余数j转移到新的余数
        for (int j = 0; j < p; j++) {
            int new_remainder = (j * 10 + d) % p;
            dp[new_remainder] += lstdp[j];
        }
        // 当前数字单独作为一个子串
        dp[d]++;
        // 统计余数为0的子串个数
        ans += dp[0];
    }

    cout << ans << endl;
    return 0;
}
```

P10108 [GESP202312 六级] 闯关游戏

题意

游戏有 N 关，每关有 M 个通道，第 i 个通道可以前进 a_i 关。离开第 s 关时获得 b_s 分。从第 0 关开始，求通关时能获得的最大总分。

数据范围：$1 \le N \le 10^4$，$1 \le M \le 100$，$1 \le a_i \le N$，$|b_i| \le 10^5$。

分析

1. **问题转化**：这是一个动态规划问题，需要找到从第 0 关到达第 N 关或之后关卡的最大得分路径。

2. **观察**：

   - 定义状态 $f[i]$ 表示到达第 i 关时能获得的最大分数
   - 状态转移：对于每个关卡 i，考虑所有可能的通道 j，如果 $i \geq a_j$，则可以从第 $i - a_j$ 关通过通道 j 到达第 i 关
   - 转移方程：$f[i] = \max_j \{f[i - a_j] + b[i - a_j]\}$

3. **算法步骤**：
   - 初始化 $f[0]=0$，其他为负无穷（因为可能有负分关卡）
   - 对于 i 从 1 到 $N+\max(a)$：
     - 对于每个通道 j：
       - 如果 $i \geq a_j$，则更新 $f[i]=\max(f[i], f[i-a_j]+b[i-a_j])$
   - 答案取 $f[N]$ 到 $f[N+\max(a)]$ 中的最大值（因为可能超过 N 关通关）

4. **正确性证明**：
   - 最优子结构：到达第 i 关的最优解必然由到达某个 $i - a_j$ 关的最优解加上 $b[i - a_j]$ 构成
   - 无后效性：当前状态只依赖于之前的状态
   - 通过遍历所有可能的转移，确保找到全局最优解

时间复杂度：$O((N + \max(a)) \times M)$，在给定数据范围内可行。

```cpp
const int N = 20010; // 最大可能到达 N + max(a)
const int INF = 0x3f3f3f3f;
int f[N], a[110], b[10010];
int n, m, max_a;

int main() {
    // 初始化
    memset(f, -0x3f, sizeof(f));
    f[0] = 0;

    cin >> n >> m;
    // 读入通道信息
    for (int i = 0; i < m; i++)
        cin >> a[i], max_a = max(max_a, a[i]); // 记录最大步长

    // 读入关卡得分
    for (int i = 0; i < n; i++)
        cin >> b[i];
```

```
    // 动态规划
    for (int i = 1; i <= n + max_a; i++)
        for (int j = 0; j < m; j++)
            if (i - a[j] >= 0)
                // 从第 i-a[j] 关通过通道 j 到达第 i 关
                // 离开第 i-a[j] 关时获得 b[i-a[j]] 分
                f[i] = max(f[i], f[i - a[j]] + b[i - a[j]]);

    // 在通关状态中找最大值
    int ans = -INF;
    for (int i = n; i <= n + max_a; i++)
        ans = max(ans, f[i]);

    cout << ans << endl;
    return 0;
}
```

P11246 [GESP202409 六级] 小杨和整数拆分

题意

将一个正整数 n 拆分成若干个完全平方数的和，求所需完全平方数的最小数量。

数据范围：$1 \le n \le 10^5$。

分析

1. **问题转化**：这是一个典型的完全背包问题，需要找到用完全平方数组合成 n 的最小数量。
2. **观察**：
    - 完全平方数可以看作是背包问题中的"物品"
    - 目标值 n 可以看作是背包的"容量"
    - 需要找到组合成 n 所需的最少"物品"数量

3. **思路**：
   - 定义状态 $dp[i]$ 表示组成数字 i 所需的最少完全平方数个数
   - 预处理所有不超过 n 的完全平方数，存储在数组 p 中
   - 状态转移：对于每个完全平方数 j，从 j 到 n 更新：

$$dp[i] = \min(dp[i], dp[i-j] + 1)$$

   - 初始化：$dp[0] = 0$，其他为无穷大

4. **正确性**：
    - 最优子结构：组成 i 的最优解必然由组成 $i-j$ 的最优解加上 1 构成，其中 j 是完全平方数
    - 通过枚举所有可能的完全平方数，并使用完全背包的更新顺序，确保每个完全平方数可以使用多次
    - 预处理完全平方数避免了重复计算

时间复杂度：$O(n\sqrt{n})$，在 $n \leq 10^5$ 时可行。

```cpp
#define int long long
const int N = 2e5 + 10;
int dp[N], n;
vector<int> p; // 记录 n 以内所有的 完全平方数 i*i
void solve() {
    // 抵达数字 i 的完全平方数方案
    for (int i = 1; i * i <= n; i++)
        p.emplace_back(i * i);
    // 可达性
    dp[0] = 0;
    // 递推（完全背包），求最小值方案
    for (int j : p)                    // 物品
        for (int i = j; i <= n; i++) // 体积
            dp[i] = min(dp[i], dp[i - j] + 1);
}
signed main() {
    memset(dp, 0x3f, sizeof dp); // 求最小值，初始化最大
    cin >> n;
    solve();
    cout << dp[n] << endl;
    return 0;
}
```

P13015 [GESP202506 六级] 学习小组

题意

将 n 名同学划分为若干个学习小组，每个小组有 k 名同学时，其讨论积极度为 a_k。求所有划分方案中，讨论积极度之和的最大值。

数据范围：$1 \le n \le 1000$，$0 \le a_i \le 10^4$。

分析

1. **问题转化**：这是一个典型的完全背包问题，需要找到将 n 名同学划分为若干小组的最优方案，使得总积极度最大。

2. **观察**：

   - 每个小组的人数 k 可以看作是背包问题中的"物品重量"
   - 对应的积极度 a_k 可以看作是"物品价值"
   - 总人数 n 可以看作是背包的"容量"
   - 需要找到在容量为 n 的背包中能装入的最大价值

3. **动态规划思路**：
   - 定义状态 $dp[i]$ 表示将 i 名同学划分成学习小组能获得的最大积极度之和
   - 状态转移：对于每个可能的小组人数 k（从 1 到 n），更新

$$dp[i] = \max(dp[i], dp[i-k] + a_k)$$

   - 初始化：$dp[0] = 0$，其他为 0（因为积极度非负）

4. **正确性**：
   - 最优子结构：将 i 名同学划分的最优解必然由将 $i-k$ 名同学划分的最优解加上一个 k 人小组的积极度构成
   - 通过枚举所有可能的小组人数，并使用完全背包的更新顺序，确保每个小组人数可以使用多次
   - 由于积极度非负，初始化时设为 0 是安全的

时间复杂度：$O(n^2)$，在 $n \leq 1000$ 时可行。

参考代码

```cpp
const int N = 1005;
int dp[N], cost[N], n;
// dp[i] : 所有的在 n 种分类方案情况下，总人数为 i 名同学情况下的最大分值。
int main() {
    ios::sync_with_stdio(false), cin.tie(nullptr);
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> cost[i];
    dp[0] = 0;
    for (int i = 1; i <= n; i++)      // 物品，分组人数
        for (int j = i; j <= n; j++) // 体积，总人数
            dp[j] = max(dp[j], dp[j - i] + cost[i]);
    cout << dp[n] << endl;
    return 0;
}
```

P14075 [GESP202509 六级] 划分字符串

题意

给定一个长度为 n 的字符串 s，需要将其划分为若干个子串，使得每个子串中每个字母至多出现一次。划分后长度为 i 的子串价值为 a_i，求所有划分方案中子串价值之和的最大值。

数据范围：$1 \le n \le 10^5$，$1 \le a_i \le 10^9$。

分析

1. **问题转化**：这是一个动态规划问题，需要找到最优的字符串划分方式，使得每个子串中无重复字符，且总价值最大。

2. **观察**：

- 定义状态 $f[i]$ 表示考虑前 $i+1$ 个字符（下标 0 到 i）时的最大价值
- 对于每个位置 i，有两种选择：
  - 将第 i 个字符单独作为一个子串：$f[i] = f[i-1] + a_1$
  - 将第 i 个字符与前面的若干字符组成一个子串：$f[i] = \max(f[j-1] + a_{i-j+1})$，其中子串 $s[j..i]$ 无重复字符

3. **优化**：

- 由于字符集只有 26 个小写字母，从位置 i 往前枚举时，最多只需要检查 26 个字符就会遇到重复
- 使用集合记录当前子串中的字符，遇到重复立即停止枚举
- 这样内层循环的时间复杂度为 $O(26)$，整体为 $O(26n)$

4. **正确性**：

- 最优子结构：到达位置 i 的最优解必然由某个位置 j 的最优解加上子串 $s[j..i]$ 的价值构成
- 通过枚举所有可能的 j 并确保子串无重复字符，保证找到全局最优解
- 使用集合快速判断字符重复，提高算法效率

时间复杂度：$O(26n)$，在 $n \leq 10^5$ 时可行。

参考代码

```
// f[i]: 以 i 为结尾的子串最大分值。
// 转移: 选和不选，不选 f[i] = f[i-1] + cost[ 1 ]; 独立一段
// 选择与前面一段拼接为子串，其中 j 为子串开头的位置，i 末尾
// f[i] = max(f[j-1] + cost[ i-j+1 ], f[i])
// 初始: f[0] = a[1], 下标从 0 开始，长度为 1 的分值
// 答案: f[n-1]
const int N = 1e5 + 10;
int cost[N], f[N], n;
string p;
signed main() {
    cin >> n;
    cin >> p;
    for (int i = 1; i <= n; i++)
        cin >> cost[i];
    f[0] = cost[1];
    // ...
```

```
    for (int i = 1; i < n; i++) {
        f[i] = f[i - 1] + cost[1]; // 独立一段
        set<int> s;
        s.insert(p[i]);
        // 枚举子串的起点
        for (int j = i - 1; j >= 0; j--) {
            if (s.count(p[j])) // 重复字符，直接断
                break;
            // 特判子串从下标 0 的情况，也就是子串可以开头到现在位置
            f[i] = max(f[i], (j == 0 ? 0 : f[j - 1]) + cost[i - j + 1]);
            s.insert(p[j]);
        }
    }
    cout << f[n - 1] << endl;
    return 0;
}
```

P10721 [GESP202406 六级] 计算得分

题意

给定一个长度为 m 的字符串，由小写字母组成。设置一个计分序列 $A = [a_1, a_2, \ldots, a_n]$，如果字符串的一个子串由 k 个 abc 首尾相接组成（$1 \le k \le n$），则得分为 a_k。字符不能重复使用，求字符串的最大总得分。

数据范围：$1 \le n \le 20$，$1 \le m \le 10^5$，$1 \le a_i \le 1000$。

分析

1. **问题转化**：字符串中可能存在多个连续的 abc 序列段，每个段由 k 个连续的 abc 组成。对于每个段，我们可以将其划分为若干个子段（每个子段由 i 个 abc 组成，$1 \leq i \leq n$），得分分别为 a_i。目标是最大化每个段的总得分。

2. **观察**：对于连续 k 个 abc 组成的段，其最大得分可以通过动态规划（完全背包）预处理得到。定义 $dp[j]$ 表示由 j 个 abc 组成的段能获得的最大得分。状态转移方程为：

$$dp[j] = \max(dp[j], dp[j-i] + a_i) \quad \text{对于 } i = 1 \text{ 到 } n$$

这表示将 j 个 abc 划分为一个长度为 i 的子段（得分 a_i）和剩余 $j-i$ 个 abc 组成的段（得分 $dp[j-i]$）。

3. **步骤**：
    - 预处理 dp 数组：使用完全背包思想，计算每个 j 对应的最大得分。
    - 遍历字符串，统计连续 abc 序列的长度 k。每当连续序列中断时，将 $dp[k]$ 加入总得分，并重置 k。
    - 注意：字符串遍历时需检查三个字符是否为 abc，以避免重复计算。
4. **正确性证明**：预处理 dp 数组确保了对于任意 k，都能得到划分后的最大得分。字符串遍历正确识别了所有连续的 abc 序列段，且字符不重复使用。

时间复杂度：预处理 dp 为 $O(n \cdot M)$（M 为最大连续 abc 数量，取 20005），遍历字符串为 $O(m)$，总时间复杂度为 $O(n \cdot M + m)$。

参考代码

```
const int N = 25;       // 计分序列最大长度
const int M = 20005;    // 最大连续abc数量估计值
int cost[N];            // 存储计分序列a_i
int dp[M];              // dp[j]：j个连续abc的最大得分
int n, m;
string s;

int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> cost[i];
        // 完全背包预处理：对于每个长度i，更新所有j>=i的dp值
        for (int j = i; j < M; j++)
            dp[j] = max(dp[j], dp[j - i] + cost[i]);
    }

    cin >> m;
    cin >> s;
    s += "   ";  // 添加三个空格，防止越界
    int ans = 0; // 总得分
    int k = 0;   // 当前连续abc数量
    int i = 0;   // 字符串索引
```

```
    while (i < m) {
        // 检查当前位置开始的三个字符是否为"abc"
        if (s[i] == 'a' && s[i+1] == 'b' && s[i+2] == 'c') {
            k++;          // 增加连续计数
            i += 3;       // 跳过三个字符
        } else {
            ans += dp[k]; // 当前连续段结束，加入得分
            k = 0;        // 重置计数
            i++;          // 移动到下一个字符
        }
    }
    // 处理末尾可能剩余的连续段
    ans += dp[k];

    cout << ans << endl;
    return 0;
}
```

GESP C++ 六级认证（三）

动态规划专题 - 代码实战