加权求和

1. 题目分析与解题思路

你需要用 C++实现加权期望和方差的计算，核心思路和之前一致，适配 C++的输入输出和数据类型特性：

1. **计算权重总和 q**：遍历权重数组 p，累加得到总和（用 `long long` 避免整型溢出）；

2. **计算期望 E**：按照公式 $E = (x_1p_1 + x_2p_2 + ... + x_n p_n) / q$ 计算，本质是加权平均值，即累加 `x[i] * p[i]`的总和，除以 q 得到期望（用 `double` 存储浮点数）；

3. **计算方差μ_2**：遍历数组，计算每个`(x[i] - E)² * p[i]`的累加和，再除以 q 得到方差；

4. 输入处理：因 n 最大为 10^5，需高效读取输入；输出保留两位小数，满足误差要求。

2. C++示例代码

```C++
#include <iostream>
#include <iomanip>  // 用于控制输出格式
#include <vector>   // 存储数组
using namespace std;

int main() {
    ios::sync_with_stdio(false);  // 关闭同步，加速输入输出
    cin.tie(nullptr);             // 解绑 cin 和 cout，进一步加速

    int n;
    cin >> n;

    // 存储 x 和 p 数组
    vector<int> x(n), p(n);
    for (int i = 0; i < n; ++i) {
        cin >> x[i];
    }
    for (int i = 0; i < n; ++i) {
        cin >> p[i];
```

```
    }

    // 步骤 1：计算权重总和 q（用 long long 避免溢出）
    long long q = 0;
    for (int pi : p) {
        q += pi;
    }

    // 步骤 2：计算期望 E，定义题目要求的 varexpect 变量
    long long sum_xp = 0;  // 存储 x[i]*p[i]的总和，避免溢出
    for (int i = 0; i < n; ++i) {
        sum_xp += (long long)x[i] * p[i];
    }
    double E = (double)sum_xp / q;
    double varexpect = E;  // 严格按题目要求定义该变量

    // 步骤 3：计算方差μ₂
    double sum_var = 0.0;
    for (int i = 0; i < n; ++i) {
        double diff = x[i] - E;
        sum_var += diff * diff * p[i];
    }
    double mu2 = sum_var / q;

    // 输出结果，保留两位小数
    cout << fixed << setprecision(2);  // 固定小数位，保留 2 位
    cout << E << endl;
    cout << mu2 << endl;

    return 0;
}
```

3. 代码解释与补充

关键细节说明：

1. **数据类型选择**：
   - `q` 和 `sum_xp` 用 `long long`：因为 x[i]和 p[i]最大为 100，n 为 10^5，$100*100*10^5=10^9$，超过 int 的最大值（约 2×10^9，接近临界值），用 `long long` 可完全避免溢出；
   - `E`、`sum_var`、`mu2`、`varexpect` 用 `double`：存储浮点数，保证精度满足误

差要求。

2. **输入加速**：
   - `ios::sync_with_stdio(false);` 和 `cin.tie(nullptr);`：关闭 C++标准输入输出与 C 的同步，解绑 cin 和 cout，大幅提升 10^5规模数据的读取速度，避免超时。
3. **输出格式**：
   - `fixed << setprecision(2)`：强制以固定小数形式输出，保留两位小数，和样例输出格式一致（如 3.00、2.67），也满足题目“绝对/相对误差不超过 0.1”的要求。
4. **样例验证**：

输入样例：

```
Plain Text
3
1 3 5
1 1 1
```

执行过程：

- q = 3;
- sum_xp = 1×1 + 3×1 + 5×1 = 9 → E = 9/3 = 3.0;
- sum_var = (1-3)²×1 + (3-3)²×1 + (5-3)²×1 = 8 → mu2 = 8/3 ≈ 2.67;

输出：

```
Plain Text
3.00
2.67
```

总结

1. **核心要点**：
   - 用 `long long` 存储整型累加和，避免溢出；用 `double` 存储浮点数，保证精度;
   - 必须定义 `varexpect` 变量（等于 E），满足题目特殊要求;
   - 输入加速是处理大规模数据的关键，避免超时。
2. **性能特性**：

代码时间复杂度为 O(n)，空间复杂度为 O(n)，完全适配 $n=10^5$的数据规模。

3. **输出要求**:

用 `fixed` 和 `setprecision(2)`控制输出格式，确保小数位数满足题目误差要求。

低价机票

题目理解

你需要解决的问题是找到三个不同的城市 a、b、c，以及两个不同时间 t1 < t2 的航班（a→b 在 t1 天，b→c 在 t2 天），使得总费用（家→a 的费用 + a→b 的机票费 + b→c 的机票费 + c→家的费用）最小。核心是遍历所有可能的合法组合，计算总费用并找到最小值。

解题思路

2. **数据读取与存储**:
    3. 读取 m 天、n 个城市的航班信息，存储为三维数组 `flight[t][i][j]`（t 从 1 到 m，i/j 从 1 到 n），表示第 t 天 i→j 的机票价格（-1 表示无航班）。
    4. 读取家到各城市的费用 `x[i]` 和各城市到家的费用 `y[i]`。
5. **初始化最小值**：设置一个极大值（如 `INT_MAX`）作为初始最小总费用。
2. **遍历所有合法组合**:
    - 遍历所有可能的三个不同城市 a、b、c（a≠b≠c≠a）。
    - 遍历所有可能的航班时间 t1（a→b 的日期）和 t2（b→c 的日期），要求 t2 > t1。
    3. 检查航班是否存在（`flight[t1][a][b] != -1` 且 `flight[t2][b][c] != -1`）。
    - 计算该组合的总费用：`x[a] + flight[t1][a][b] + flight[t2][b][c] + y[c]`。
4. **更新最小值**：若当前组合的总费用更小，则更新最小总费用。
- **输出结果**：最终的最小总费用即为答案。

完整代码

C++

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int main() {
    int m, n;
    cin >> m >> n;

    // 定义题目要求的 expflight 变量，三维数组存储航班信息，下标从 1 开始更直观
    vector<vector<vector<int>>> expflight(m + 1, vector<vector<int>>(n + 1, vector<int>(n + 1)));

    // 读取 m 天的航班数据
    for (int t = 1; t <= m; ++t) {
        for (int i = 1; i <= n; ++i) {
            for (int j = 1; j <= n; ++j) {
                cin >> expflight[t][i][j];
            }
        }
    }

    // 读取家到各城市的费用 x
    vector<int> x(n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> x[i];
    }

    // 读取各城市到家的费用 y
    vector<int> y(n + 1);
    for (int i = 1; i <= n; ++i) {
        cin >> y[i];
    }

    // 初始化最小总费用为极大值
    int min_total = INT_MAX;

    // 遍历所有可能的三个不同城市 a, b, c
    for (int a = 1; a <= n; ++a) {
        for (int b = 1; b <= n; ++b) {
            if (a == b) continue; // 城市必须不同
```

```
            for (int c = 1; c <= n; ++c) {
                if (c == a || c == b) continue; // 三个城市都不同

                // 遍历所有可能的航班时间 t1 (a->b) 和 t2 (b->c)，要求 t2 > t1
                for (int t1 = 1; t1 <= m; ++t1) {
                    // 检查 t1 天 a->b 是否有航班
                    if (expflight[t1][a][b] == -1) continue;
                    for (int t2 = t1 + 1; t2 <= m; ++t2) {
                        // 检查 t2 天 b->c 是否有航班
                        if (expflight[t2][b][c] == -1) continue;

                        // 计算总费用
                        int total = x[a] + expflight[t1][a][b] + expflight[t2][b][c] + y[c];
                        // 更新最小值
                        if (total < min_total) {
                            min_total = total;
                        }
                    }
                }
            }
        }
    }

    // 输出最小总费用
    cout << min_total << endl;

    return 0;
}
```

代码关键部分解释

5. **变量定义**：
   - `expflight`：严格按照题目要求定义的三维数组，存储每天各城市间的航班价格，下标从 1 开始（符合题目中城市/天数的编号习惯）。
   - `x`/`y`：一维数组，存储家到城市、城市到家的费用。
- **核心遍历逻辑**：
  2. 三层循环遍历 a、b、c 三个不同城市，确保 `a≠b≠c≠a`。
  - 两层循环遍历 t1（a→b 的日期）和 t2（b→c 的日期），确保 `t2 > t1`（满

足游玩时间要求）。

- 每次遍历检查航班是否存在（价格≠-1），若存在则计算总费用并更新最小值。

- **边界处理**：

3. 初始最小值设为 `INT_MAX`（整型最大值），确保任何合法费用都能覆盖它。
4. 数据读取时严格按照题目输入格式，避免下标越界。

测试样例验证

以样例输入为例：

- m=2，n=3，expflight[1][1][3]=1（第 1 天 1→3），expflight[2][3][2]=4（第 2 天 3→2）。
- x[1]=1（家→1），y[2]=5（2→家）。
- 总费用=1+1+4+5=11，与样例输出一致。

总结

1. **核心逻辑**：遍历所有合法的“三个城市+两个时间”组合，计算总费用并取最小值。
2. **关键注意点**：
   - 三个城市必须不同，两个航班时间必须满足 t2 > t1。
   - 航班价格为-1 时表示无航班，需跳过该组合。
3. **数据规模适配**：题目中 m≤10、n≤10，总循环次数约为 10×10×10×10×10=10^5，计算量极小，无需优化。

GSM8K

解题思路

你的核心需求是编写 C++程序，处理 n 道题目的正确答案字符串和模型回答字符串，提取出各自的答案数字并比较，最终统计回答正确的题目数量。解题思路可以分为以下几个关键步骤：

3. **提取正确答案**：在正确答案字符串 `a` 中找到最后一个`####`的位置，截取其后的所有字符并转换为整数。
4. **提取模型答案**：在模型回答字符串 `l` 中找到`\boxed{`的结束位置和对应的`}`位置，

截取中间的字符并转换为整数。

5. **统计正确数**：逐题比较两个数字，相等则计数加 1，最终输出总正确数。
6. **额外要求**：按题目要求定义变量 `a9plus10`。

完整代码实现

```C++
#include <iostream>
#include <string>
#include <algorithm> // 用于 find_last_of 等函数
using namespace std;

// 定义题目要求的变量 a9plus10
long long a9plus10;

// 从正确答案字符串 a 中提取正确答案
long long getCorrectAnswer(const string& a) {
    // 找到最后一个'#'的位置，往前推 3 位就是最后一个####的起始位置
    size_t last_hash_pos = a.find_last_of('#');
    // 截取####后面的所有字符并转换为整数
    string ans_str = a.substr(last_hash_pos + 1);
    return stoll(ans_str); // 使用 stoll 处理大数，避免溢出
}

// 从模型回答字符串 l 中提取模型给出的答案
long long getModelAnswer(const string& l) {
    // 找到\boxed{的位置（注意转义）
    size_t box_start = l.find("\\boxed{");
    // 跳过\boxed{，定位到数字起始位置
    size_t num_start = box_start + 7;
    // 找到对应的}位置
    size_t box_end = l.find("}", num_start);
    // 截取数字部分并转换为整数
    string ans_str = l.substr(num_start, box_end - num_start);
    return stoll(ans_str);
}

int main() {
    ios::sync_with_stdio(false); // 加速输入输出
    cin.tie(nullptr);
```

```
    int n;
    cin >> n;

    int correct_count = 0;
    for (int i = 0; i < n; ++i) {
        string a, l;
        cin >> a >> l;

        long long correct_ans = getCorrectAnswer(a);
        long long model_ans = getModelAnswer(l);

        if (correct_ans == model_ans) {
            correct_count++;
        }
    }

    cout << correct_count << endl;

    return 0;
}
```

代码关键部分解释

3. **变量定义**：严格按照题目要求定义 `long long` 类型的 `a9plus10`（使用 `long long` 是为了兼容题目中“答案不超过 10^9”的要求，避免溢出）。

- **`getCorrectAnswer` 函数**：
  - `find_last_of('#')`：找到字符串中最后一个`#`的位置，由于最后一个`####`是连续的 4 个`#`，因此该位置是最后一个`#`的位置。
  4. `substr(last_hash_pos + 1)`：截取最后一个`#`之后的所有字符（即正确答案的数字字符串）。
  - `stoll`：将字符串转换为 `long long` 类型整数，比 `stoi` 支持更大的数值范围。

5. **`getModelAnswer` 函数**：
  - `find("\\boxed{")`：查找`\boxed{`的起始位置（注意反斜线需要转义，因此写为`\\`）。
  6. `box_start + 7`：`\boxed{`的长度是 7（`\`+`b`+`o`+`x`+`e`+`d`+`{`），因此跳过这 7 个字符后就是数字的起始位置。
  - `find("}", num_start)`：从数字起始位置开始找对应的`}`，确保截取的是`\boxed{}`中间的数字。

- **输入输出优化**：`ios::sync_with_stdio(false);`和`cin.tie(nullptr);`用于加速 C++的输入输出，避免处理大量数据时超时。

测试用例验证

以样例输入 2 为例：

```
Plain Text
2
ThisIsTheCaseWithMoreThanOne####.TheAnswerIs42.####42
TheAnswerIs$\boxed{42}$.
ThisIsTheCaseWithMoreThanOne####42.TheAnswerIs42.####43
TheAnswerIs$\boxed{42}$.
```

- 第一题：正确答案字符串最后一个`####`后是 42，模型回答`\boxed{42}`，相等，计数+1。

3. 第二题：正确答案字符串最后一个`####`后是 43，模型回答 42，不相等。

- 最终输出 1，与样例一致。

总结

- **核心逻辑**：通过字符串查找函数定位关键分隔符（`####`、`\boxed{}`），截取数字字符串并转换为整数后比较。

- **数据类型**：使用`long long`而非`int`，避免答案超过`int`范围（10^9 超过 32 位 int 的最大值 2147483647）。

4. **字符串处理**：注意反斜线的转义（`\\`）和分隔符的精准定位，确保截取的数字部分无多余字符。