这道题的数据范围非常大，需要对区间内的数字和进行高效计算。暴力枚举 (L) 到 (R) 范围的每个数字显然不现实，我们需要利用**数位 DP** 来快速求解。

解题思路

1. **问题转化**

    - 区间 ([L, R]) 的数字和等于 (S(R) - S(L-1))，其中 (S(x)) 表示从 1 到 (x) 的数字和。
    - 通过数位 DP，我们可以快速计算 (S(x))。

2. **数位 DP 基本思想**

    - 设 (dp[pos][sum][tight]) 表示在当前数位为 (pos)，前面数字和为 (sum)，且是否受上界约束的情况下的合法方案。
    - (pos)：当前正在处理的数位；
    - (sum)：当前已经累加的数字和；
    - (tight)：是否受当前范围的上界限制。
    - 转移：
        - 如果 (tight) 为 1，则本位的取值范围是 ([0, \text{当前位的值}])；
        - 如果 (tight) 为 0，则本位的取值范围是 ([0, 9])。

3. **具体实现**

    - 首先定义函数 `digitSum(x)`，利用数位 DP 计算从 1 到 (x) 的数字和。
    - 对每个查询 ([L, R])，计算结果为： [\text{result} = (S(R) - S(L-1)) \mod (10^9 + 7)]

代码实现

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

const int MOD = 1e9 + 7;

int dp[20][200][2]; // dp[pos][sum][tight]

// 数位 DP，计算 1 到 num 的数字和
int digitSumDP(const string &num, int pos, int sum, bool tight) {
    if (pos == num.size()) return sum; // 数位结束，返回累加的和

    if (dp[pos][sum][tight] != -1) return dp[pos][sum][tight];

    int limit = tight ? num[pos] - '0' : 9;
    int res = 0;

    for (int digit = 0; digit <= limit; digit++) {
        res = (res + digitSumDP(num, pos + 1, sum + digit, tight && (digit ==
limit))) % MOD;
```

```cpp
    }

    return dp[pos][sum][tight] = res;
}

// 包装函数，将数字转化为字符串并计算数字和
int digitSum(long long x) {
    string num = to_string(x);
    memset(dp, -1, sizeof(dp));
    return digitSumDP(num, 0, 0, 1);
}

int main() {
    int T;
    cin >> T;

    while (T--) {
        long long L, R;
        cin >> L >> R;

        // 计算区间和
        int sumR = digitSum(R);
        int sumL = digitSum(L - 1);

        int result = (sumR - sumL + MOD) % MOD;
        cout << result << endl;
    }

    return 0;
}
```

代码说明

1. **`digitSumDP` 函数**
   - 递归计算从 1 到 `num` 的数字和；
   - 使用 `tight` 来表示当前是否受上界约束；
   - `sum` 累加当前的数字和。
2. **预处理**
   - 每次调用 `digitSum` 时，清空 DP 表 (`memset(dp, -1, sizeof(dp))`)。
3. **主逻辑**
   - 对于每组区间 ([L, R])，分别调用 `digitSum(R)` 和 `digitSum(L-1)`；
   - 通过两者差值计算最终结果。

样例测试

输入：

```
2
24 69
70 120
```

输出：

```
411
498
```

复杂度分析

1. **时间复杂度**
   - 每次调用 `digitSum(x)` 的复杂度为 (O(\text{digits}))，即 (O(\log_{10} x))；
   - 对于 (T) 个区间，总复杂度为 (O(T \cdot \log_{10} R))。
2. **空间复杂度**
   - 使用了一个 (dp) 表，其大小为 (20 \times 200 \times 2)，即 (O(8000))，常数级别。

这种方法在 (R) 达到 (10^{18}) 的范围内依然能够高效运行。