## 数据溢出

## 639. Decode Ways II

题目大义：对一串数字和'\*'组成的字符串进行到字母的译码，求译码方案数。

题目本身比较简单，一看其题目结构应该很容易知道应该是用DP来做，但是一直到下午才ac，思考和细节方面没有错误，经过大量时间纠察才发现是数据溢出了。还是基础C++没有顾全到。

前面做了一道题目，因为每次都申请了最大范围long long []数组，正确但是Memory Limit Error，所以这次动态申请，且用了int，虽然考虑到了数据可能溢出，将结果用临时long long变量保存，以为int\*int赋值为long long结果会对，其实int\*int=int，溢出不管，赋给long long转为long long，所以错误！后面明白了可能是这个错误，于是在常数后加了L，但是还是不对，因为gcc是linux编译器，sizeof(long)=4,sizeof(long long)=8，必须为LL。

经验总结：这种数据溢出的错误不太容易找出，有几次都犯了同样错误，还是不长记性！一定要铭记在心。

找错误最重要的是仔细分析、逻辑思考：

若题目有错误提示，可以估计Error case大概处于什么位置，可知通过了一些什么case，错误地方最可能在什么位置发生。

若题目没有提示，只能凭自己经验，在肯定思维是正确的话，只有可能是一些傻逼错误或者隐藏很深的错误，比如数据溢出。

// t=dp[i]\*(long long)9; //正确

t=dp[i]\*9LL; //正确

// t=dp[i]\*(long )9; //错误

// t=dp[i]\*9L; //错误

cout<<"sizeof (long ) = "<<sizeof(long )<<endl; //4

cout<<"sizeof (long long) = "<<sizeof(long long )<<endl; //8

cout<<"sizeof (0L ) = "<<sizeof(0L )<<endl; //4

cout<<"sizeof (0LL ) = "<<sizeof(0LL )<<endl; //8