Solution - ZJ

括号计数 (bracket)

本意是考枚举/搜索的，直接搜索每个字符的可能性并 $O(|S|)$ 检查，时间复杂度 $O(8^{|S|})$，得分：70pts（当然，已经确定的不要再搜/枚举了，要不然你就 40pts 了）。

加一些剪枝，得分：70~90pts（90pts 是给恰好 $O(ans)$ 的，也即 $O(\text{Catalan}(\frac{|S|}{2})4^{|S|})$）。

正解是给区间 dp 的，设 $f_{l,r}$ 表示区间 $[l,r]$ 是合法的方案数，转移如下：

$$f_{l,r} = \sum_{i=l+1}^{r} \text{count}(S_l, S_i) f_{l+1,i-1} \times f_{i+1,r}$$

$$f_{i+1,i} = 1$$

其中 $\text{count}(p,q)$ 表示 p,q 有几种能够匹配的可能。

时间复杂度：$O(|S|^3)$，开 long long 即可，如果考虑高精乘法的复杂度为 $O(nm)$，那么复杂度为 $O(|S|^5)$。

不降序列 (sequence)

$l_i = r_i$ 的是送分的。

$-10 \le l_i \le r_i \le 10$ 的，可以写一个 dp：

> *设 $f_{i,j}$ 表示 $a_i = j$，以 i 为末尾的可能的最长不降连续子序列，转移可以用前缀 $\min$ 优化到 $O(nV)$，或者 $O(nV^2)$ 可能也能通过。*

考虑如何判断一个区间 $[p, q]$ 是否可行，从前往后确定 a_i，a_i 贪心选择最小的一个不小于 a_{i-1} 和 l_i 的，即 $a_i = \max(a_{i-1}, l_i)$，再 检查 $a_i \le r_i$ 即可，时间复杂度：$O(n^3)$，轻松优化到 $O(n^2)$。

然后发现这个东西可以双指针，再把 $[p, q]$ 的要求重写一下：

$$\forall p \le i \le j \le q, l_i \le r_j$$

- 考虑 p 加一：如果 $[p, q]$ 满足，那么 $[p+1, q]$ 也一定满足；
- 考虑 q 加一：判断 $r_q \ge \max_{i=p}^{q}\{l_i\}$ 即可，使用倍增/线段树询问区间 $\max$ 即可，时间复杂度：$O(n \log n)$。

但实际上可以优化到 $O(n)$，由于有单调性的限制，所以可以使用单调队列维护 $\max_{i=p}^{q}\{l_i\}$：如果 $x < y$ 且 $l_x \le l_y$，那么 l_x 就不需要记录了，维护一个单调递减的 l 序列，每次弹出最前面的直到开头的 $l \le r_q$，更新答案即可，时空复杂度：$O(n)$。

最优方案 (plan)

分析性质，发现所有边（对应到一条路径）一定互相不交（边不交），而且最后每条边一定被一条路径覆盖一次，比较抽象，有点难描述。

并且最后不可能出现 $u \to v, v \to w$ 的两条边，一定可以调整成 $u \to w$ 的一条边，显然更优。

那么一个子树，向上能够连向祖先的点，就只有恰好一个，那么就可以设计 dp，$f_{u,i}$ 表示 u 子树中，剩下 i 需要向上连，子树剩下部分的最优值，那么转移为（找到 u 的某个儿子 v 使得 i 在 v 的子树中）：

$$f_{u,i} = f_{v,i} + \sum_{w \in \mathrm{son}(u) \setminus \{v\}} \min_{j \in \mathrm{sub}(w)} \{f_{w,j} + a_u a_j\}$$
$$f_{u,u} = \sum_{w \in \mathrm{son}(u)} \min_{j \in \mathrm{sub}(w)} \{f_{w,j} + a_u a_j\}$$

$O(n^2)$ 轻松转移，进一步优化，需要首先掌握李超线段树和线段树合并。

对于一个 j，就可以看成一条直线 $a_j x + f_{w,j}$，那么每次去查询 f_w 的若干直线中，$x = a_u$ 处取值的最大值即可。

子树之间的影响只有对于所有直线整体加一个常数，然后再把 u 的所有子树的直线合并到 u 的线段树中，最后插入 u 代表的直线即可，直接使用李超线段树的合并即可。

时间复杂度：$O(n \log n)$，空间复杂度：$O(n)$，偏难的一道题，对数据结构的要求较高。

排序求值 (sort)

算法零

暴力，时间复杂度 $O(n\log^2 n)$，期望得分：10pts。

算法一

优化后的暴力，时间复杂度 $O(n)/O(n\log n)$，期望得分：20~30pts。

算法二

首先求逆排列，设 b_i 为 i 的字典序排名，那么答案就是：

$$\left(\sum_{i=1}^{n}(b_i-i)\bmod 998244353\right)\bmod (10^9+7)$$

考虑 $<10^{12}$ 的情况，我们考虑先把前 6 位搜索出来。

设 p,q 分别为 x 的前六/后六位，那么有 $b_x=b_{\overline{p000000}}+b_q$。

$$(b_i-i)\bmod 998244353=((b_{\overline{p000000}}-p\times 10^6)+(b_q-q))\bmod 998244353$$

可以发现除了 p 等于 n 前六位的情况下，其余的情况右边都可以预处理，然后二分一下那些 b_q-q 与左边加起来需要减去一个 998244353 就好了。

而 p 等于 n 前六位，q 的取值只有最多 10^6 种，直接暴力解决即可。

时间复杂度 $O(\sqrt{n}\log n)$。期望得分：70~90pts。

算法三

开始简单的推一下式子。

$$\sum(i-a_i)\bmod p=\sum(rk_i-i-p\times\lfloor\frac{rk_i-i}{p}\rfloor)=-p\times\sum\lfloor\frac{rk_i-i}{p}\rfloor,p=998244353$$

于是，只需求出 $\sum\lfloor\frac{rk_i-i}{p}\rfloor$ 即可。

由于 p 达到了 10^9 级别，而且 $rk_i-i\in[-n,n]$。

所以 $\lfloor\frac{rk_i-i}{p}\rfloor$ 在 $O(\frac{n}{p})$ 级别。

考虑枚举 $k=\lfloor\frac{rk_i-i}{p}\rfloor$，计算符合条件的 i 的个数。

对于 i 的限制即为：$k\times p\le rk_i-i<(k+1)\times p$。

直接差分掉，问题转化为求出 $rk_i-i\le lim$ 的 i 的个数。

对于位数相同的数 $i-1,i$，发现字典序大小即为数值本身的大小。

所以 $rk_{i-1}+1\le rk_i \iff rk_{i-1}-(i-1)\le rk_i-i$。

发现 rk_i-i 在相同位数的时候是递增的。

于是，我们可以先枚举 i 的位数 len。

然后二分出该位数下满足 $rk_i-i<lim$ 的最大的 i 即可。

还留下来最后一个问题，如何求出 rk_i?

计算 $str(j)<str(i)$ 的个数，可以枚举 j 的位数，直接计算个数即可。

总时间复杂度：$O(\frac{n}{p}\log^3 n)$，常数很小。期望得分：80pts。

参考实现：

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
using ll=long long;
const int N=20,p=998244353,mod=1e9+7;
char num[N];
int len;
ll n,pw[N];
int k,a[N];
ll getrk(ll x){
    k=0;
    for(ll y=x;y;y/=10)a[++k]=y%10;
    for(int i=k+1;i<=len;i++)a[i]=0;
    reverse(a+1,a+1+k);
    ll now=0,ans=0;
    for(int i=1;i<k;i++){
        now=now*10+a[i];
        ans+=now-pw[i-1]+1;
    }
    for(int i=k;i<len;i++){
        now=now*10+a[i];
        ans+=now-pw[i-1];
    }
    now=now*10+a[len];
    if(now<=n)ans+=now-pw[len-1];
    else ans+=n-pw[len-1]+1;
    return ans+1;
}
ll query(ll lim){
    ll ans=0;
    for(int i=1;i<=len;i++){
```

```
        ll l=pw[i-1]-1,r=min(pw[i],n+1),mid;
        for(;l+1<r;){
            mid=(l+r)>>1;
            if(getrk(mid)-mid<lim)l=mid;
            else r=mid;
        }
        ans+=l-pw[i-1]+1;
    }
    return ans;
}
signed main(){
    scanf("%s",num+1),len=strlen(num+1);
    for(int i=1;i<=len;i++)n=n*10+num[i]-'0';
    for(int i=pw[0]=1;i<=len;i++)pw[i]=pw[i-1]*10;
    ll las=0,ans=0;
    for(ll k=-n/p-1;k*p<=n;k++){
        ll cnt=query((k+1)*p);
        ans-=k*(cnt-las),las=cnt;
    }
    cout<<(ans%mod*p%mod+mod)%mod;
    return 0;
}
```

算法四

对于上一个做法，把二分去掉，枚举每一位的贡献，直接做除法求出分界点即可去掉一个 $\log$，时间复杂度：$O(\frac{n}{p}\log n^2)$，期望得分：100pts。

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
#ifdef DEBUG
#include"debug.h"
#else
#define debug(...) void()
#endif
#define all(x) (x).begin(),(x).end()
template<class T>
auto ary(T *a,int l,int r){
    return vector<T>{a+l,a+1+r};
}
using ll=long long;
using ull=unsigned long long;
const int N=20,p=998244353,mod=1e9+7;
char num[N];
int len;
```

```cpp
ll n,pw[N];
int k,a[N];
ll calc1(int k,ll lim){
    ll cur=lim+(pw[len]-1)/9-k,res=0;
    if(cur<=0)return -1;
    for(int i=1;i<=k;i++){
        ll w=(pw[len-i+1]-1)/9-pw[k-i];
        int p=w>0?min((cur-1)/w,9ll):9;
        res=res*10+p,cur-=w*p;
    }
    return res;
}
ll calc2(int k,ll lim){
    ll cur=lim+(pw[len]-1)/9-k-(n+1),res=0;
    if(cur<=0)return -1;
    for(int i=1;i<=k;i++){
        ll w=(pw[len-i]-1)/9-pw[k-i];
        int p=w>0?min((cur-1)/w,9ll):9;
        res=res*10+p,cur-=w*p;
    }
    return res;
}
ll query(ll lim){
    ll ans=0;
    for(int k=1;k<=len;k++){
        ll cur=max(calc1(k,lim),calc2(k,lim));
        cur=max(cur,pw[k-1]-1);
        cur=min({cur,pw[k]-1,n});
        ans+=cur-pw[k-1]+1;
    }
    return ans;
}
int main(){
    scanf("%s",num+1),len=strlen(num+1);
    for(int i=1;i<=len;i++)n=n*10+num[i]-'0';
    for(int i=pw[0]=1;i<=len;i++)pw[i]=pw[i-1]*10;
    ll las=0,ans=0;
    for(ll k=-n/p-1;k*p<=n;k++){
        ll cnt=query((k+1)*p);
        ans-=k*(cnt-las),las=cnt;
    }
    cout<<(ans%mod*p%mod+mod)%mod<<endl;
    return 0;
}
#ifdef DEBUG
#include"debug.hpp"
#endif
```