D. Anton and School - 2

time limit per test

2 seconds

memory limit per test

256 megabytes

input

standard input

output

standard output

As you probably know, Anton goes to school. One of the school subjects that Anton studies is Bracketology. On the Bracketology lessons students usually learn different sequences that consist of round brackets (characters "(" and ")" (without quotes)).

On the last lesson Anton learned about the regular simple bracket sequences (RSBS). A bracket sequence *s* of length *n* is an RSBS if the following conditions are met:

* It is not empty (that is *n* ≠ 0).
* The length of the sequence is even.
* First https://espresso.codeforces.com/a64c1b2df604f8cfa9acf6716a3cc1424488361e.png charactes of the sequence are equal to "(".
* Last https://espresso.codeforces.com/a64c1b2df604f8cfa9acf6716a3cc1424488361e.png charactes of the sequence are equal to ")".

For example, the sequence "((()))" is an RSBS but the sequences "((())" and "(()())" are not RSBS.

Elena Ivanovna, Anton's teacher, gave him the following task as a homework. Given a bracket sequence *s*. Find the number of its distinct subsequences such that they are RSBS. Note that a subsequence of *s* is a string that can be obtained from *s* by deleting some of its elements. Two subsequences are considered distinct if distinct sets of positions are deleted.

Because the answer can be very big and Anton's teacher doesn't like big numbers, she asks Anton to find the answer modulo 109 + 7.

Anton thought of this task for a very long time, but he still doesn't know how to solve it. Help Anton to solve this task and write a program that finds the answer for it!

**Input**

The only line of the input contains a string *s* — the bracket sequence given in Anton's homework. The string consists only of characters "(" and ")" (without quotes). It's guaranteed that the string is not empty and its length doesn't exceed 200 000.

**Output**

Output one number — the answer for the task modulo 109 + 7.

**Examples**

**input**

**Copy**

)(()()

**output**

**Copy**

6

**input**

**Copy**

()()()

**output**

**Copy**

7

**input**

**Copy**

)))

**output**

**Copy**

0

**Note**

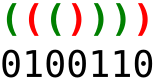
In the first sample the following subsequences are possible:

* If we delete characters at the positions 1 and 5 (numbering starts with one), we will get the subsequence "(())".
* If we delete characters at the positions 1, 2, 3 and 4, we will get the subsequence "()".
* If we delete characters at the positions 1, 2, 4 and 5, we will get the subsequence "()".
* If we delete characters at the positions 1, 2, 5 and 6, we will get the subsequence "()".
* If we delete characters at the positions 1, 3, 4 and 5, we will get the subsequence "()".
* If we delete characters at the positions 1, 3, 5 and 6, we will get the subsequence "()".

The rest of the subsequnces are not RSBS. So we got 6 distinct subsequences that are RSBS, so the answer is 6.

At first, let's simplify the problem: let our string consists of *x* + *y* characters, begins with *x* characters "(" and ends with *y* characters ")". How to find the number of RSBS in such string?

Let's prove that this number is equal to https://espresso.codeforces.com/b61b32e9e62b166b135e575311807bfd5a0d0596.png. It's easy to observe that this formula also means the number of ways to match the string with the sequence of zeros and ones of the same length, which contains exactly *x* ones. Now prove that for every such sequence of zeros and ones we can find an RSBS subsequence. How can we do it? Let's consider it on the example of the following string:



Let's include to our subsequence all the opening brackets that match zeros and all the closing brackets that match ones. In our example, we include brackets number 1, 3, 5 and 6, so we get the subsequence "(())", which is an RSBS.

Why every sequence we got in this way is an RSBS? Let the number of ones that match closing brackets is equal to *z*. So *x* - *z* ones match opening brackets (because we have *x* ones, as we remember) and, therefore, *z* zeros match opening brackets. So the number of opening brackets is equal to the number of closing brackets in our subsequence. Also opening brackets appear earlier than closing brackers. So such subsequence is always an RSBS, and the statement above is proved.

Now we must understand how to solve the entire problem. Let's iterate over an opening bracket that is the last opening bracket in our subsequence. Now observe that only opening brackets may come before this bracket, and only closing brackets may come after this bracket. The rest of the brackets will definitely not appear in the subsequence. Let's count the number of opening brackets before the iterated one, incluing the iterated one (let this number is equal to *x*), and also the number of closing brackets after the iterated one (let this number is equal to *y*). To calculate these numbers, we can precalc them for all the positions in https://espresso.codeforces.com/a9e52a3aa139b288bbd0be22431bc50717f5456f.png using prefix sums.

Now, we have reduced our problem to the already solved, because we have *x* opening brackets and then *y* closing brackets. But we also have an additional condition: we must necessarily take the last opening bracket. So the answer is equal to https://espresso.codeforces.com/d75c4d55b68b3f1e13ee5b8859ddb06471694e7a.png, not https://espresso.codeforces.com/aa956d9cb52ae2ee9a6c9955270b08a8efb5fa6f.png, because on the position with the last opening bracket we must put a zero. So we must put *x* ones on *x* + *y* - 1 positions instead of *x* + *y* positions.

Time complexity is https://espresso.codeforces.com/ce8033aa3fa1bf15b6ce927eb5feca373632b0ff.png (logarithm is to divide by modulo, that is necessary to calculate the number of combinations).

1. **#include**<bits/stdc++.h>
2. **#define** pb push\_back
3. **#define** **int** **long** **long** **int**
4. **#define** vec **vector<int>**
5. **#define** REP(i,a,b) **for**(i=a;i<b;i++)
6. **using** **namespace** std;
7. **int** mod=1e9+7;
8. **int** fact[3000001];
9. **void** factorial()
10. {
11. fact[1]=1,fact[0]=1;
12. **for**(**int** i=2;i<3000001;i++)
13. fact[i]=((i%mod)\*(fact[i-1]%mod))%mod;
14. }
15. **int** inverse(**int** n)
16. {
17. n%=mod;
18. **int** b=mod-2;
19. **int** res=1;
20. **while**(b>0)
21. {
22. **if**(b&1)
23. res=((res%mod)\*(n%mod))%mod;
24. n=((n%mod)\*(n%mod))%mod;
25. b>>=1;
26. }
27. **return** res%mod;
28. }
29. main()
30. {
31. ios\_base::sync\_with\_stdio(**false**);
32. cin.tie(NULL);
33. cout.tie(NULL);
34. factorial();
35. string s;
36. cin>>s;
37. **int** n=s.length();
38. **int** l=0,r=0;
39. **for**(**int** i=0;i<n;i++)
40. **if**(s[i]==**')'**)
41. ++r;
42. **int** ans=0;
43. **for**(**int** i=0;r;i++)
44. {
45. **if**(s[i]==**'('**)
46. {
47. **int** res=fact[l+r]%mod;
48. res=((res%mod)\*(inverse(fact[l+1])%mod))%mod;
49. res=((res%mod)\*(inverse(fact[r-1])%mod))%mod;
50. ans=((ans%mod)+(res%mod))%mod;
51. l++;
52. }
53. **else**
54. r--;
55. }
56. cout<<ans%mod<<**"\n"**;
57. }

Code 2:

1. **#include<cstdio>**
2. **#define** MN 200000
3. **#define** MOD 1000000007
4. **char** s[MN+5];
5. **int** f[MN+5],v[MN+5];
6. **inline** **int** inv(**int** x)
7. {
8. **int** r=1,y=MOD-2;
9. **for**(;y;y>>=1,x=1LL\*x\*x%MOD)**if**(y&1)r=1LL\*r\*x%MOD;
10. **return** r;
11. }
12. **int** main()
13. {
14. **int** i,l,r,ans=0;
15. scanf(**"%s"**,s);
16. **for**(i=f[0]=1;i<=MN;++i)f[i]=1LL\*f[i-1]\*i%MOD;
17. **for**(v[i=MN]=inv(f[MN]);i--;)v[i]=1LL\*v[i+1]\*(i+1)%MOD;
18. **for**(i=l=r=0;s[i];++i)**if**(s[i]==**')'**)++r;
19. **for**(i=0;r;++i)s[i]==**'('**?(ans=(ans+1LL\*f[l+r]\*v[l+1]%MOD\*v[r-1])%MOD,++l):--r;
20. printf(**"%d"**,ans);
21. }