



Let's denote $f(s)$ as $s.count('(') + s.count('?') - s.count(')')$ and $g(s)$ as $s.count('(') + s.count('?') - s.count(')')$. Please note that $f(s) = -g(s) + 2 \times s.count('?')$ and $g(s) = -f(s) + 2 \times s.count('?')$.

By assumption, if we replace s_q by '(' the conditions will be violated. By replacing s_q the second condition can't be violated, thus the third condition will be violated. So, there's an integer i such that $1 \leq i \leq q$ and $g(t[i..|t|]) < 0$ (t is s after replacing s_q by '('). Thus, $g(s[i..|s|]) < 2$. Similarly, there's an integer j such that $q \leq j \leq |s|$ and $f(s[1..j]) < 2$.

Since all three conditions are satisfied for s (by assumption), then $0 \leq g(s[i..|s|]), f(s[1..j]) \leq 1$.

Let's break s into three parts (they could be empty): $a = s[1..(i-1)]$, $b = s[i..j]$ and $c = s[(j+1)..|s|]$.

$g(s[i..|s|]) = g(b) + g(c)$ and $f(s[1..j]) = f(a) + f(b)$. Since the three conditions are satisfied for s , then $0 \leq g(c), f(a)$.

$f(a) + f(b) \leq 1$ so $f(a) - 1 \leq -f(b)$. Thus $f(a) - 1 \leq g(b) - 2 \times b.count('?')$, so $f(a) - 1 + 2 \times b.count('?') \leq g(b)$.

So $f(a) - 1 + 2 \times b.count('?') + g(c) \leq g(b) + g(c) \leq 1$. So $f(a) - 1 + 2 \times b.count('?') + g(c) \leq 1$. Since $i \leq q \leq j$, then $2 \leq 2 \times b.count('?')$.

Also, $0 \leq g(c), f(a)$. So, $1 \leq f(a) - 1 + 2 \times b.count('?') + g(c) \leq 1$. So $f(a) - 1 + 2 \times b.count('?') + g(c) = 1$. This requires that $f(a) = g(c) = 0$ and $b.count('?') = 1$.

Since $f(a)$ and $g(c)$ are even, then $|a|$ and $|c|$ are even, and since $|s|$ is even (first condition), then $|b|$ is also even (because $|s| = |a| + |b| + |c|$).

$f(a) = g(c) = 0$ and $0 \leq f(a) + f(b)$ and $0 \leq g(b) + g(c)$, thus $0 \leq f(b), g(b)$. Also, $f(a) + f(b), g(b) + g(c) \leq 1$, thus $0 \leq f(b), g(b) \leq 1$, since $|b|$ is even, $f(b)$ and $g(b)$ are also even, thus, $f(b) = g(b) = 0$. $g(b) = -f(b) + 2 \times b.count('?')$ and since $1 \leq b.count('?')$ then $g(b) \neq 0$.

Thus, we have $0 \neq 0$, which is false. So the lemma is true.

Using the lemma above, each time we can replace a question mark by parentheses and at the end we get a correct bracket sequence.

After proof: Knowing this fact, we can find all such substrings by checking the three conditions. Pseudo code:

```
f[n][n] = {}
g[n][n] = {}
for l = 1 to n
    cur = 0
    ok = true
    for r = l to n
        if s[r] == ')'
            cur = cur - 1
        else
            cur = cur + 1
        if cur < 0:
            ok = false
        f[l][r] = ok

for r = 1 to n
    cur = 0
    ok = true
    for l = r to 1
        if s[l] == '('
            cur = cur - 1
        else
            cur = cur + 1
        if cur < 0:
            ok = false
        g[l][r] = ok
```

[View all →](#)

→ Find user

 Handle:

→ Recent actions

[jinlifu1999](#) → [Codeforces Round #460 \(Div. 2\)](#)
[DarthPrince](#) → [Codeforces Round #459 Editorial](#)
[usernameon](#) → [An interesting randomised approach to a graph problem](#)
[usaxena95](#) → [SOS Dynamic Programming \[Tutorial\]](#)
[asifthegreat](#) → [Help me please :D](#)
[reality420](#) → [HackerEarth January Circuits '18](#)
[DarthPrince](#) → [Codeforces Round #459](#)
[csacademy](#) → [Round #67 \(Div. 1 + Div. 2\) with prizes](#)
[matthew99](#) → [Congrats to FizzyDavid on becoming the youngest Legendary Grandmaster ever](#)
[okaybody10](#) → [Help please](#)
[kissu_pari_na](#) → [Merging two sets in c++](#)
[Nickolas](#) → [Marathon Match 97](#)
[keerthu](#) → [Idleness limit exceeded — help](#)
[keerthu](#) → [Help!!!](#)
[Luqman](#) → [Teams going to ACM ICPC World Finals 2018](#)
[LeCalin](#) → [The MaxFlow problem](#)
[phpduke](#) → [Introduction to Competitive Machine Learning](#)
[tourist](#) → [Topcoder SRM 728](#)
[ciphreck](#) → [Trie Implementation](#)
[MikeMirzayanov](#) → [Rule about third-party code is changing](#)
[brdy](#) → [Finding bridges with an added condition in O\(V+E\)](#)
[Tobby_And_Friends](#) → [Number Theory Problem from UVA \(UVA 11774 - Doom's Day\)](#)
[Erona](#) → [Problems in PDF format](#)
[Superty](#) → [CodeCraft-18 and Codeforces Round #458 \(Div. 1 + Div. 2, combined\) Editorial](#)
[usernameon](#) → [What are some good resources on randomised algorithms for competitive programming?](#)
[Detailed →](#)



```
ans = 0
for l = 1 to n
    for r = l to n
        if f[l][r] and g[l][r] and (r-l+1) % 2 == 0
            ans = ans + 1
print(ans)
```

Total time complexity: $\mathcal{O}(n^2)$ where $n = |s|$

Writer: **DarthPrince**

917B - MADMAX

Denote $dp(v, u, c)$ as the winner of the game (the person that starts it or the other one?, a boolean, true if first person wins) if the first person's marble is initially at vertex v and the second one's initially at u and our set of letters is $\{ichar(c), ichar(c+1), \dots, 'z'\}$ if $ichar(i) = char('a' + i)$ (c is an integer).

Denote $adj(v) = \{x; v \rightarrow x\}$ and $ch(x, y)$ as the character written on edge from x to y .

Now if there's some x in $adj(v)$ such that $c < int(ch(v, x) - 'a')$ and $dp(u, x, ch(v, x)) = false$, then the first person can move his/her marble to vertex x and win the game, thus $dp(v, u, c) = true$, otherwise it's false.

Because the graph is a DAG there's no loop in this dp, thus we can use memoization. The answer for i, j is $dp(i, j, 0)$.

Total time complexity: $\mathcal{O}(|sigma| \times n \times (n + m))$

Writer: **DarthPrince**

917C - Pollywog

What would we do if n was small? Notice that at any given time if i is the position of the leftmost pollywog and j is the position of the rightmost pollywog, then $j - i < k$. Thus, at any given time there's an i such that all pollywogs are on stones $i, i+1, \dots, i+k-1$, in other words, k consecutive stones.

x pollywogs are on k consecutive stones, thus, there are $\binom{k}{x}$ different ways to sit these pollywogs on k stones, that's about 70 at most. Denote $dp[i][state]$ as the minimum amount of energy the pollywogs need to end up on stones $i, i+1, \dots, i+k-1$, and their positions are contained in $state$ (there are $\binom{k}{x}$ states in total). We assume $init$ is the initial state (pollywogs on the x first stones) and $final$ is the final state (pollywogs on the x last stones).

Thus, we could easily update dp in $\mathcal{O}(k)$ (where would the first pollywog jump?) using dynamic programming and this would work in $\mathcal{O}(n \times k \times \binom{k}{x})$ since the answer is $dp[n-k+1][final]$.

But n is large, so what we could do is using matrix multiplication (similar to matrix multiplication, but when multiplying two matrices, we use minimum instead of sum and sum instead of multiplication, that means if $C = A \times B$ then $C[i][j] = \min(A[i][k] + B[k][j])$ for all k) to update the dp, in case $q = 0$ to solve the problem in $\mathcal{O}(\binom{k}{x}^3 \times \log(n))$.

For $q > 0$, we combine the dynamic programming without matrix and with matrix. Note that the special stones only matter in updating the dp when there's a special stone among $i, i+1, \dots, i+k-1$, that means at most for $k \times q$ such i , for the rest we could use matrices for updating.

Total time complexity: $\mathcal{O}(\log(n) \binom{k}{x}^3 + qk^2 \binom{k}{x})$

Writer: **DarthPrince**

917D - Stranger Trees

Solution #1:



First, for every K such that $0 \leq K \leq N - 1$ we are going to find for every K edges in the original tree we are going to find the number of labeled trees having these K edges, then we will add them all to $res[K]$.

But Mr. Author aren't we going to count some tree that has exactly E (where $E > K$) common edges with the original tree in $res[K]$? Yes, that's true. But we only count it $\binom{E}{K}$ times! So, after computing the res array we are going to iterate from $N - 1$ to 0 assuming that the res is correct for all $J > I$ (our current iteration), and then reduce $res[J] \times \binom{J}{I}$ (the fixed res) from $res[I]$. Then we'll have the correct value for $res[I]$.

But Mr. Author, how are we going to find $res[K]$ in the first place? Let's first find out for a fixed K edges forest, in how ways we connect the remaining vertices to get a tree. Let's look at the components in the forest. Only their sizes are relevant because we can't connect anything inside them. Let the sizes be $sz[0] \dots sz[N - 1]$. (if you assume that the sizes are all 1, the number of resulting trees are N^{N-2} (Kayley's theorem)).

To solve this subproblem, let's go to another subproblem. Let's assume that for every additional edge, we know which components it is going to go connect. Then, the number of resulting trees is $\prod_{i=0}^{N-1} sz[i]^{d[i]}$ where $d[i]$ is the degree of component i (edges between this component and other components). The reason is that we have $sz[v]$ vertices inside component v to give to every edge that has one endpoint in v .

Ok going back to the parent subproblem. $d[i]$ huh? I've heard that vertex v appears in the Prufer code of a tree $d[v] - 1$ times. so we've gotta multiply the answer by $sz[v]$ every time it appears in Prufer code. It's also multiplied by $\prod_{i=0}^{N-1} sz[i]$ because we haven't multiplied it one time ($d[v] - 1$ not $d[v]$). But how to make it get multiplied by $sz[v]$ every time component v is chosen? Look at this product. $(sz[0] + sz[1] + \dots + sz[N - 1])^{N-2}$. If in the i -th parenthesis $sz[v]$ is chosen, then let the i -th place on the Prufer code of the tree connecting the components be the component v . The good thing about this product is that if component v has come in the Prufer code K times, then the multiplication of the parenthesis has $sz[v]^K$ in it. So it counts exactly what we want to count.

$(\prod_{i=0}^{N-1} sz[i]) \times (\sum_{i=0}^{N-1} sz[i])^{N-2}$ is the answer for some fixed K edges. $\sum_{i=0}^{N-1}$ corresponds to N in the original problem and $N - 2$ corresponds to $Number_of_components - 2$. so we want to count $(\prod_{i=0}^{N-1} sz[i]) \times N^{number_of_components-2}$

Okay Mr. Author so how do we count this for every K fixed edges in the original tree. Lets count

$dp[top_vertex\ v][size_of_component_containing_top_vertex\ s][the_number_of_edges_we_have_fixed\ e]$

which contains $\prod_{i=0}^{N-1} sz[i]$ of every component inside v 's subtree which doesn't include v 's component and $N^{the_number_of_components_not_including_v's}$. We can update this from v 's children. Let's add v 's children one by one to the dp by assuming that the children we didn't go over don't exist in v 's subtree.

let's go over $old_dp[v][vs][ve]$ and $dp[u][us][ue]$, we either fix the edge between u and v then it'll add $dp[u][us][ue] \times dp[v][vs][ve]$ to $next_dp[v][us + vs][ve + ue + 1]$ and otherwise it'll add $dp[u][us][ue] \times dp[v][vs][ve] \times N \times us$ to $dp[v][vs][ve + ue]$. We can also divide it by N^2 at the end with modulo inverses. We can find $res[K]$ with the sum of $dp[root][s][K] \times s \times N$. (with $s = N$ as a corner case).

The solution may look that it's N^5 because it's inside 5 fors. But it's actually N^4 if the us and vs fors go until $sz[u]$ and $sz[v]$ (actually only the subtree of v that we've iterated on). So the cost is $sz[u] \times sz[v] \times N^2$. Let's look at it like every vertex from u 's subtree is handshaking with every vertex of v 's subtree and the cost of their handshaking is N^2 . We know that two vertices handshake only once. That's why it'll be $\binom{N}{2} \times N^2$ which is of $\mathcal{O}(N^4)$.

Solution #2:

Let's define $F(X)$ as the number of spanning trees of the graph K_N plus $X - 1$ copies of T (the original tree). If we look at $F(X)$ we'll see that it is actually

$\sum_{i=0}^{N-1} X^i \times number_of_trees_with_i_common_edges_with_T$

because it has X^i ways to choose which of the X multiple edges it should choose for the common edges. So the problem is to find F 's coefficients. We can do that by polynomial



interpolation if we have N sample answers of F . Let's just get N instances of F for $X = 1$ till $X = N$. We can find that using Kirchhoff's matrix tree theorem to find the number of spanning trees of a graph. So the complexity is $\mathcal{O}(\text{interpolation}) + \mathcal{O}(\text{Determinant} \times N)$. So we have an $\mathcal{O}(N^4)$ complexity. This is how to do it in N^2 -> (I don't know it yet, I'll update it when I have it ready)

Writer: **Reyna**

917E - Upside Down

Assume t_i is reverse of s_i . Use centroid-decomposition. When solving the problem for subtree S , assume its centroid is c . For a fixed query, assume v and u are both in S and path from v to u goes through the centroid c (this happens exactly one time for each v and u). Assume x is string of path from c to v and y is string of path from c to u . We should find the number of occurrences of s_k in $\text{reverse}(x) + y$. If number of occurrences of s in t is $f(s, t)$ then $f(s_k, \text{reverse}(x) + y) = f(t_k, x) + f(s_k, y) + A$. First two variables can be calculated using aho-corasick and segment tree.

A is the number of occurrences of s_k in the path such that some part of it belongs to $\text{reverse}(x)$ and some to y .

So, so far the time complexity is $\mathcal{O}(n \log^2(n))$.

Now for counting A , first calculate the suffix tree for each string (for each s_k and t_k). A suffix tree is a trie, so let $sf(v, s)$ be the vertex we reach when we ask the string of path from c (root) to v one by one in the suffix tree of string s . We can calculate this fast for every v and s if we merge this suffix trees into one trie (we do this before we start the centroid-decomposition algorithm in per-process).

We associate a value val to each vertex of the trie, initially zero for every vertex. Now we traverse this trie like DFS. When we reach a vertex x , we iterate over all suffixes (there are $2(|s_1| + \dots + |s_n|)$ suffixes) that end in x (the suffixes that equal the string of path from root of the trie to vertex x), and for each suffix (s, k) (suffix of string s with length k), we add 1 to the val of each vertex in the subtree of the vertex where the suffix $(\text{reverse}(s), |s| - k)$ ends and we subtract this number when we're exiting vertex x (in DFS).

Now back to the centroid-decomposition, A equals val of vertex in trie where the suffix (t_k, b) when in DFS we're at vertex in trie where (s_k, a) ends where a is the size of the longest suffix of s_k that is a prefix of the string of the path from c (root) to u and similarly, b is the size of the longest suffix of t_k that is a prefix of the string of the path from c (root) to v . For achieving this goal, we can use persistent segment tree on the starting time-finishing time range of vertices in the trie (or without using persistent segment tree, we could calculate every A after the centroid-decomposition is finished, kind of offline).

Total time complexity: $\mathcal{O}(N \log^2(N))$ where $N = n + q + |s_1| + |s_2| + \dots + |s_n|$.

Writer: **DarthPrince**

codeforces, round, 459, #editorial, princeofpersia

+75



[DarthPrince](#)



18 hours ago



[29](#)



Comments (29)

[Write comment?](#)



teja349

18 hours ago, # | ☆

+28

Bonus question for Div1B: try to solve when sigma can be of size of m.

→ [Reply](#)



Kallaseldor

13 hours ago, # ^ | ☆

0

Could you please explain how to solve this (bonus) problem? I've no idea...

→ [Reply](#)

13 hours ago, # ^ | ☆
Probably something like

+5



tfg

DP[u][v] is the maximum character needed for a person to be on u (on his turn), the other on v and the first wins no matter what.

→ [Reply](#)

12 hours ago, # ^ | ☆ ← Rev. 2 ▲ +2 ▼



teja349

Yes ,you are right.I misunderstood it.
Expected Complexity: $O(n*n*n)$

Spoiler

→ [Reply](#)



tfg

11 hours ago, # ^ | ☆

▲ 0 ▼

$O(n^3 * \log n)$? The solution I described is $O(n * (n + m))$.
Here's a submission of that solution:

<http://codeforces.com/contest/917/submission/34697087>

→ [Reply](#)

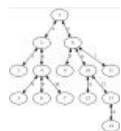


teja349

11 hours ago, # ^ | ☆ 0 ▼

Ohh I am sorry.I was thinking something different. Yes you are right!!!

→ [Reply](#)



vatsalsharma376

9 hours ago, # ^ | ☆ +5 ▼

How is this $O(n(n+m))$?

→ [Reply](#)



notking

10 hours ago, # ^ | ☆

▲ -22 ▼

teja trying to be smart but fails :P

→ [Reply](#)

17 hours ago, # | ☆

← Rev. 2 ▲ +50 ▼

I have a (maybe simpler and elegant?) solution for div1 A.

How would you solve this if no question marks were involved? Well, this is a standard problem. You iterate through the possible substrings and keep a `score` which starts from 0, which represents how many open brackets we have. Whenever you iterate over a '(', you add 1 and when you iterate over ')', you subtract one. The string is valid iff at the end the score is 0 and at no point during the iteration was the score negative.



mouse_wireless

For the problem at hand, we do the exact same thing, except we keep one more counter, say `qmarks`, which represents the number of question marks so far. Obviously, when we iterate over '?', we increment this counter. Also, if at any point of the iteration we have `qmarks > score`, then at least one of the question marks has to be an open bracket (if they were all closed brackets, we would have negative score, which is illegal). Therefore, if this situation occurs, increment `score` and decrement `qmarks`.

At the end, a substring is legal iff its length is even and `qmarks >= score` (we can use question marks to close off all remaining open brackets).

Seems simpler than the solution in the editorial (and also doesn't use additional memory, if that's relevant).

→ [Reply](#)



OptxPrime

16 hours ago, # ^ | ☆

▲ 0 ▼

simple solution, thank u so much...u saved me from official solution

→ [Reply](#)

12 hours ago, # ^ | ☆

▲ 0 ▼

Nira One \



loveWithLegend1

nice one .

→ [Reply](#)

92anurag

10 hours ago, # ^ | ☆

▲ 0 ▼

Thanks for such a nice solution.

→ [Reply](#)

Seeker98

10 hours ago, # ^ | ☆

▲ 0 ▼

Huge thanks!!!!

→ [Reply](#)

92 minutes ago, # ^ | ☆

▲ 0 ▼

I had basically the same solution but (again maybe?) just a little more natural to explain.

At each moment we will define l and r as minimum and maximum number of unclosed brackets in the sequence before given that it is a prefix of a correct bracket sequence. For every starting position initially $l = r = 0$. When we iterate over '(' or ')' we obviously increase or decrease both by one correspondingly. When we iterate over '?' we decrease l and increase r by one. If $r < 0$ at any point we stop here and go to a next starting position. l can't be less than 0, so if $l < 0$ we increase it by 2. Finally the ending position is possible if and only if $l = 0$.

[Code](#)→ [Reply](#)

Vercingetorix

63 minutes ago, # ^ | ☆

▲ 0 ▼

Could you please explain it on a test: "???" ? . Here, if I consider the whole substring, right from the 1st iteration, $qmarks > score$. so: $qmarks = 1$ $score = -1$; $qmarks = 2$; $score = -2$; $qmarks = 3$; $score = -3$; $score = -2$



Talk_less

Hence, the length is even and $qmarks \geq score$. Hence it's legal. But you can't form any valid bracket with this sequence. Am i missing something?

→ [Reply](#)

57 minutes ago, # ^ | ☆

▲ +1 ▼



mouse_wireless

When $qmarks > score$, you decrement $qmarks$ and increase $score$, not the other way around. The logic is that you turn a question mark into an open bracket, since at least one of the question marks has to be open.

→ [Reply](#)

Talk_less

new, 23 minutes ago, # ^ | ☆

▲ 0 ▼

Thanks a lot. I must read carefully!

→ [Reply](#)

jtnydv25

16 hours ago, # | ☆

▲ +17 ▼

Solution #2 for problem D is just beautiful!

→ [Reply](#)

rahil056

16 hours ago, # ^ | ☆

▲ -42 ▼

→ [Reply](#)

The comment is hidden because of too negative feedback, click [here](#) to view it

16 hours ago, # | ☆

▲ 0 ▼

Div 1 E : "First two variables can be calculated using aho-corasick and segment



Szymanski_w

Div 1 E : "First two variables can be calculated using and-or-graph and segment tree." Can someone explain how to do this? Thanks in advance.

→ [Reply](#)

Diegogrc

12 hours ago, # | ☆

▲ +3 ▼

How can i solve div2 C with dp?

→ [Reply](#)

_Darkmoon

12 hours ago, # | ☆

▲ 0 ▼

I hope someone can answer my confusion...To Div2 D,I still can't understand the rightness of the editorial.Can this ensure that both players play optimally?

→ [Reply](#)

12 hours ago, # ^ | ☆

← Rev. 2

▲ +3 ▼

if one's make "a" move can make other lose then it is optimally move right? so you have to find a path that the $ichar(i)$ is big enough that the next person cant go any further



Jiburiru

$c < \text{int}(\text{ch}(v, x) - 'a')$ and $\text{dp}(u, x, \text{ch}(v, x)) = \text{false}$ its mean the ichar you have is larger than the limit now and the ichar you have ,can make the next player fail to move,(dp is false) , thus the state now is true

→ [Reply](#)

_Darkmoon

10 hours ago, # ^ | ☆

▲ 0 ▼

Thanks a lot!

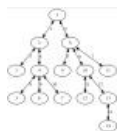
→ [Reply](#)

rohansumant

11 hours ago, # | ☆

▲ 0 ▼

Could anyone please tell me what is sigma in 917B editorial?

→ [Reply](#)

vatsalsharma376

10 hours ago, # ^ | ☆

← Rev. 3

▲ +3 ▼

Sigma is the size of the constraint on the edge. In this case,its 26. If we allowed numbers(from 0 to 9) and alphabets both in the problem, then it would have been 36.

→ [Reply](#)

rohansumant

10 hours ago, # ^ | ☆

▲ 0 ▼

thanks

→ [Reply](#)

rohansumant

10 hours ago, # | ☆

▲ 0 ▼

Could anyone confirm that [this](#) solution is actually $O(N^4)$? I have a nested loop in the recurrence function which already is $O(N^2)$.

→ [Reply](#)

4 hours ago, # | ☆

▲ +1 ▼

Can someone please explain more precise how to make the the dp with matrix multiplication on Div1C/Div2E? Especially how to construct the matrix to be multiplied and the initial step of the dp? I have tried to multiply the matrix $m[i][j]$ as cost to go from state number i to state number j , but I couldn't find the recurrence

→ [Reply](#)

Juve45



The only programming contests Web 2.0 platform
Server time: Jan/30/2018 20:49:23^{UTC+5.5} (d1).
Desktop version, switch to [mobile version](#).
[Privacy Policy](#)