DarthPrince's blog

# Codeforces Round #459 Editorial

By DarthPrince, history, 8 hours ago,

## 918A - Eleven

Calculate the first $x$ Fibonacci sequence elements, where $x$ is the greatest integer such that $f_x \le n$. Let $s$ be a string consisting of $n$ lowercase 'o' letters. Then for each $i \le x$, perform $s_{f_i}$ = 'O'. The answer is $s$.

Pseudo code:

```
s = ""
for i = 0 to n-1
    s[i] = 'o'
x = y = 1
while y <= n
    s[y-1] = 'O'
    tmp = y
    y = y + x
    x = tmp
print(s)
```

Total time complexity: $\mathcal{O}(n)$

Writer: DarthPrince

## 918B - Radio Station

Save the names and ips of the servers. Then for each command find the server in $\mathcal{O}(n)$ and print its name.

Total time complexity: $\mathcal{O}(nm)$

Writer: DarthPrince

## 917A - The Monster

First, let's denote $s[l..r]$ as the substring $s_l s_{l+1} \dots s_r$ of string $s$. Also $s.count(t)$ is the number of occurrences of $t$ in $s$.

A string consisting of parentheses and question marks is pretty if and only if:

1. $|s|$ is even.
2. $0 \le s[1..i].count('(') + s[1..i].count('?') - s[1..i].count(')')$ for each $1 \le i \le |s|$.
3. $0 \le s[i..|s|].count(')') + s[i..|s|].count('?') - s[i..|s|].count('(')$ for each $1 \le i \le |s|$.

**Proof**: If $s.count('?') = 0$ then $s$ is a correct bracket sequence. Otherwise, let $q$ be an integer between $1$ to $|s|$ such that $s_q$ = '?'.

**Lemma**: We can replace $s_q$ by either '(' or ')' such that the three conditions above remain satisfied.

Proof: We'll use proof by contradiction. If we can replace $s_q$ by either '(' or ')' such that the conditions remain satisfied, the lemma is proven. Otherwise, the conditions will be violated if we replace $s_q$ by '(' or ')'.

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 $1 \le i \le q$ 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 $q \le j \le |s|$ and $f(s[1..j]) < 2$.

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

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 $0 \le g(c), f(a)$.

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

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

Also, $0 \le g(c), f(a)$. So, $1 \le f(a) - 1 + 2 \times b.count('?') + g(c) \le 1$. 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 $0 \le f(a) + f(b)$ and $0 \le g(b) + g(c)$, thus $0 \le f(b), g(b)$. Also, $f(a) + f(b), g(b) + g(c) \le 1$, thus $0 \le f(b), g(b) \le 1$, 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 $1 \le b.count('?')$ then $g(b) \ne 0$.

Thus, we have $0 \ne 0$, 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

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 $\{ichar(c), ichar(c + 1), ..., 'z'\}$ if $ichar(i) = char('a' + i)$ ($c$ is an integer).

Denote $adj(v) = \{x; v \to x\}$ 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 $i, i + 1, ... i + k - 1$, 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 $i, i + 1, ... i + k - 1$, 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 $i, i + 1, ... i + k - 1$, 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 $0 \le K \le N - 1$ 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 $sz[0]... sz[N - 1]$. (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. $(sz[0] + sz[1] + ... + sz[N - 1])^{N-2}$. 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 $\mathcal{O}(interpolation) + \mathcal{O}(Determinant \times N)$. 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 $reverse(x) + y$. If number of occurrences of $s$ in $t$ is $f(s, t)$ then $f(s_k, reverse(x) + y) = f(t_k, x) + f(s_k, y) + A$. 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 $reverse(x)$ and some to $y$.

So, so far the 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 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 $2(|s_1| + ... + |s_n|)$ 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 $(reverese(s), |s| - k)$ 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 $N = n + q + |s_1| + |s_2| + ... + |s_n|$.

Writer: **DarthPrince**

**codeforces**, **round**, **459**, **#editorial**, **princeofpersia**

+38 DarthPrince 8 hours ago 23

## Comments (23)

Write comment?

teja349

8 hours ago, # | +18

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

→ Reply

3 hours ago, # ^ | ☆ 0

tfg

3 hours ago, # ^ | ☆ 0

Probably something like

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

teja349

82 minutes ago, # ^ | ☆ ← Rev. 2 0

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

Spoiler

→ Reply

tfg

49 minutes ago, # ^ | ☆ 0

O(n^3 * logn)? 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

43 minutes ago, # ^ | ☆ 0

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

→ Reply

notking

18 minutes ago, # ^ | ☆ 0

teja trying to be smart but fails :P

→ Reply

mouse_wireless

7 hours ago, # | ☆ ← Rev. 2 +30

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.

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

6 hours ago, # ^ | ☆ 0

108 minutes ago, # ^ | ☆ 0

loveWithLegend1

Nice One :) .

→ Reply

14 minutes ago, # ^ | ☆ 0

92anurag

Thanks for such a nice solution.

→ Reply

12 minutes ago, # ^ | ☆ 0

Seeker98

Huge thanks!!!!!

→ Reply

6 hours ago, # | ☆ +14

jtnydv25

Solution #2 for problem D is just beautiful!

→ Reply

6 hours ago, # ^ | ☆ -30

→ Reply

rahil056

The comment is hidden because of too negative feedback, click here to view it

6 hours ago, # | ☆ 0

Szymanski_w

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

→ Reply

2 hours ago, # | ☆ +2

Diegogrc

How can i solve div2 C with dp?

→ Reply

97 minutes ago, # | ☆ 0

__Darkmoon

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

80 minutes ago, # ^ | ☆ ← Rev. 2 +2

Jiburiru

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

c < int(ch(v, x) - 'a') and dp(u, x, ch(v, x)) = 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

15 minutes ago, # ^ | ☆ 0

__Darkmoon

Thanks a lot!

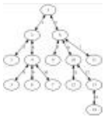
→ Reply

**rohansumant**

22 minutes ago, # | ☆ 0

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

→ Reply



**vatsalsharma376**

15 minutes ago, # ^ | ☆ ← Rev. 3 +1

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**

14 minutes ago, # ^ | ☆ 0

thanks

→ Reply

**rohansumant**

8 minutes ago, # | ☆ 0

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

→ Reply