darkshadows's blog

DP on Trees Tutorial

By **darkshadows**, history, 3 years ago,

A certain question on Quora and some junior asking about DP on Trees is what inspired this post. Its been a long time since I wrote any tutorial, so, its a welcome break from monotonicity of events.

Pre-requisites:

- Will to read this post thoroughly. :)
- Also, you should know basic dynamic programming, the optimal substructure property and memoisation.
- Trees(basic DFS, subtree definition, children etc.)

Dynamic Programming(DP) is a technique to solve problems by breaking them down into overlapping sub-problems which follow the optimal substructure. We all know of various problems using DP like subset sum, knapsack, coin change etc. We can also use DP on trees to solve some specific problems.

We define functions for nodes of the trees, which we calculate recursively based on children of a nodes. One of the states in our DP usually is a node i, denoting that we are solving for the subtree of node i.

As we do examples, things will get clear for you.

Problem 1

==========

The first problem we solve is as follows: Given a tree T of N nodes, where each node i has C_i coins attached with it. You have to choose a subset of nodes such that no two adjacent nodes(i.e. nodes connected directly by an edge) are chosen and sum of coins attached with nodes in chosen subset is maximum.

This problem is quite similar to 1-D array problem where we are given an array $A_1, A_2, ..., A_N$; we can't choose adjacent elements and we have to maximise sum of chosen elements. Remember, how we define our state as $\mathrm{dp}(i)$ denoting answer for $A_1, A_2, ..., A_i$. Now, we define our recurrence as $dp(i) = \max(A_i + \mathrm{dp}(i-2), \mathrm{dp}(i-1))$(two cases: choose A_i or not, respectively).

Now, unlike array problem where in our state we are solving for first i elements, in case of trees one of our states usually denotes which subtree we are solving for. For defining subtrees we need to root the tree first. Say, if we root the tree at node 1 and define our DP $\mathrm{dp}(V)$ as the answer for subtree of node V, then our final answer is $\mathrm{dp}(1)$.

Now, similar to array problem, we have to make a decision about including node V in our subset or not. If we include node V, we can't include any of its children(say $v_1, v_2, ..., v_n$), but we can include any grand child of V. If we don't include V, we can include any child of V.

So, we can write a recursion by defining maximum of two cases.
$\mathrm{dp}(V) = \max(\sum_{i=1}^{n} \mathrm{dp}(v_i), C_V + (\sum_{i=1}^{n} \text{ sum of dp(j) for all children j of } v_i))$.

As we see in most DP problems, multiple formulations can give us optimal answer. Here, from an implementation point of view, we can define an easier solution using DP. We define two DPs, $\mathrm{dp1}(V)$ and $\mathrm{dp2}(V)$, denoting maximum coins possible by choosing nodes from subtree of node V and if we include node V in our answer or not, respectively. Our final answer is maximum of two case i.e. $\max(\mathrm{dp1}(1) + \mathrm{dp2}(1))$.

And defining recursion is even easier in this case. $\mathrm{dp1}(V) = C_V + \sum_{i=1}^{n} \mathrm{dp2}(v_i)$(since we cannot include any of the children) and $\mathrm{dp2}(V) = \sum_{i=1}^{n} \max(\mathrm{dp1}(v_i), \mathrm{dp2}(v_i))$(since we can include children now, but we can also choose not include them in subset, hence max of both cases).

About implementation now. You must notice that answer for a node is dependent on answer of its children. We write a recursive definition of DFS, where we first call recursive function for all children and then calculate answer for current node.

```cpp
//adjacency list
//adj[i] contains all neighbors of i
vector<int> adj[N];

//functions as defined above
int dp1[N],dp2[N];

//pV is parent of node V
void dfs(int V, int pV){

    //for storing sums of dp1 and max(dp1, dp2) for all children of V
    int sum1=0, sum2=0;

    //traverse over all children
    for(auto v: adj[V]){
    if(v == pV) continue;
    dfs(v, V);
    sum1 += dp2[v];
    sum2 += max(dp1[v], dp2[v]);
    }

    dp1[V] = C[V] + sum1;
    dp2[V] = sum2;
}

int main(){
    int n;
    cin >> n;

    for(int i=1; i<n; i++){
    cin >> u >> v;
    adj[u].push_back(v);
    adj[v].push_back(u);
    }
```


Akikaze → Codeforces Round #538 (Div. 2)

kun → The Editorial of the First Codeforces Global Round

giorgigagua20006 → Itos

F-35 → disconnect specific pair of vertices in graph

harshit2202 → Another DP Problem

Swift → Becoming IM by 2020

kfqg → Problem Difficulty Algorithm Seems Broken

at1 → Beta Round #4 - Tutorial

K.Chaitanya → Need help with problem 500 E . New Year Domino

vintage_Petr_Mitrichev → Not getting problem D of yesterday's round .

Wsl_F → CF-Predictor — Know your rating changes!

Rebryk → Codeforces Round #291 (Div. 2) Editorial

Updown → Free Competitive Programming Classes

hmehta → Topcoder SRM 750 - Registration Open

atlasworld → Any good priority queue tutorial / blog.

Laggy → ICPC World Finals 2019 Team Ratings — What is your prediction?

PikMike → Educational Codeforces Round 57 Editorial

KAN → Codeforces Global Round 1

Nickolas → Announcement: Microsoft Q# Coding Contest – Winter 2019

2fast4me → Kattis problemset

newhandleeeeee → How to know minimal operations to make array same but not change its sum?

```
    dfs(1, 0);
    int ans = max(dp1[1], dp2[1]);
    cout << ans << endl;
}
```

Complexity is $O(N)$.

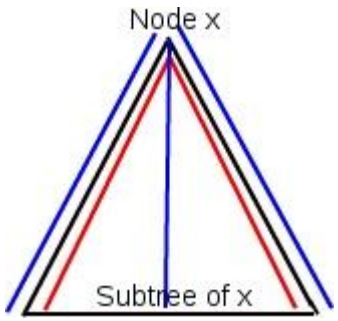
Problem 2:

==========

Given a tree T of N nodes, calculate longest path between any two nodes(also known as diameter of tree).

First, lets root tree at node 1. Now, we need to observe that there would exist a node x such that:

- Longest path starts from node x and goes into its subtree(denoted by blue lines in the image). Lets define by $f(x)$ this path length.
- Longest path starts in subtree of x, passes through x and ends in subtree of x(denoted by red line in image). Lets define by $g(x)$ this path length.



If for all nodes x, we take maximum of $f(x)$, $g(x)$, then we can get the diameter. But first, we need to see how we can calculate maximum path length in both cases.

Now, lets say a node V has n children v_1, v_2, ..., v_n. We have defined $f(i)$ as length of longest path that starts at node i and ends in subtree of i. We can recursively define $f(V)$ as $1 + \max(f(v_1), f(v_2), ..., f(v_n))$,

spookywooky → Still don't get the "You should compute P·Q−1 modulo 109+7, where Q−1 denotes the multiplicative inverse of Q modulo 109+7."

RAD → Codeforces Round #166 Tutorial

ojuz → A blog post for people who want to connect their account with oj.uz

Detailed →

because we are looking at maximum path length possible from children of V and we take the maximum one. So, optimal substructure is being followed here. Now, note that this is quite similar to DP except that now we are defining functions for nodes and defining recursion based on values of children. This is what DP on trees is.

Now, for case 2, a path can originate from subtree of node v_i, and pass through node V and end in subtree of v_j, where $i \neq j$. Since, we want this path length to be maximum, we'll choose two children v_i and v_j such that $f(v_i)$ and $f(v_j)$ are maximum. We say that

$$g(V) = 1 + \text{sum of two max elements from set} \{f(v_1), f(v_2), ..., f(v_n)\}.$$

For implementing this, we note that for calculating $f(V)$, we need f to be calculated for all children of V. So, we do a DFS and we calculate these values on the go. See this implementation for details.

If you can get the two maximum elements in $O(n)$, where n is number of children then total complexity will be $O(N)$, since we do this for all the nodes in tree.

```
//adjacency list
//adj[i] contains all neighbors of i
vector<int> adj[N];

//functions as defined above
int f[N],g[N],diameter;

//pV is parent of node V
void dfs(int V, int pV){
    //this vector will store f for all children of V
    vector<int> fValues;

    //traverse over all children
    for(auto v: adj[V]){
    if(v == pV) continue;
    dfs(v, V);
    fValues.push_back(f[v]);
    }
```

```
    //sort to get top two values
    //you can also get top two values without sorting(think about it) in O(n)
    //current complexity is n log n
    sort(fValues.begin(),fValues.end());

    //update f for current node
    f[V] = 1;
    if(not fValues.empty()) f[V] += fValues.back();

    if(fValues.size()>=2)
         g[V] = 2 + fValues.back() + fValues[fValues.size()-2];

    diameter = max(diameter, max(f[V], g[V]));
}
```

Now, we know the basics, lets move onto solving a little advanced problems.

Problem 3:

==========

Given a tree T of N nodes and an integer K, find number of different sub trees of size less than or equal to K.

First, what is a sub tree of a tree? Its a subset of nodes of original tree such that this subset is connected. Note a sub tree is different from our definition of subtree.

Always think by rooting the tree. So, say that tree is rooted at node 1. At this moment, I define $S(V)$ as the subtree rooted at node V. This subtree definition is different from the one in problem. In $S(V)$ all nodes in subtree of V are included.

Now, lets try to count total number of sub trees of a tree first. Then, we'll try to use same logic for solving original problem.
Lets define $f(V)$ as number of sub trees of $S(V)$ which include node V i.e. you choose V as root of the sub trees that we are forming. Now, in these subtrees, for each child u of node V, we have two options: whether to

include them in sub tree or not. If you are including a node u, then there are $f(u)$ ways, otherwise there is only one way(since we can't choose any nodes from $S(u)$, otherwise the subtree we are forming will get disconnected).

So, if node V has children $v_1, v_2, ..., v_n$, then we can say that $f(V) = \prod_{i=1}^{n} 1 + f(v_i)$. Now, is our solution complete? $f(1)$ counts number of sub trees of T which are rooted at 1. What about sub trees which are not rooted at 1? We need to define one more function $g(V)$ as number of subtrees of $S(V)$ which are not rooted at V. We derive a recursion for $g(V)$ as $\sum_{i=1}^{n} f(i) + g(i)$ i.e. for each child we add to $g(V)$ number of ways to choose a subtree rooted at that child or not rooted at that child.

Our final answer is $f(1) + g(1)$.

Now, onto our original problem. We are trying to count sub trees of T whose size doesn't exceed K. We need to have one more state in our DP at each node. Lets define $f(V, k)$ as number of sub trees with k nodes and V as root. Now, we can define recurrence relation for this. Let's say for node V, there are direct children nodes $v_1, v_2, ..., v_n$. Now, to form a subtree with $k + 1$ nodes rooted at V, lets say $S(v_i)$ contributes a_i nodes. Of course, k must be $\sum_{i=1}^{n} a_i)$ since we are forming a sub tree of size $k + 1$(one node is contributed by V). We should realise that $f(V, k)$ is sum of the value $\prod_{i=1}^{n} f(v_i, a_i)$ for all possible distinct sequences $a_1, a_2, ..., a_n$.

Now, to do this computation at node V, we will form one more DP denoted by $\mathrm{dp1}$. We say $\mathrm{dp1}(i, j)$ as number of ways to choose a total of j nodes from subtrees defined by $v_1, v_2, ..., v_i$. The recurrence can be defined as $\mathrm{dp1}(i, j) = \sum_{k=0}^{K} \mathrm{dp1}(i - 1, j - k) * \mathrm{f}(i, k)$, i.e. we are iterating over k assuming that subtree of v_i contributes k nodes.

So, finally $f(V, k) = \mathrm{dp1}(n, k)$.
And our final solution is sum $\sum_{i=1}^{K} f(V, i)$ for all nodes V.

So, in terms of pseudo code we write:

```
f[N][K+1]

void rec(int cur_node){

     f[cur_node][1]=1
```

```
     dp_buffer[K] = {0}
     dp_buffer[0] = 1

     for(all v such that v is children of cur_node)
      rec(v)

      dp_buffer1[K] = {0}
      for i=0 to K:
           for j=0 to K-i:
                dp_buffer1[i + j] += dp_buffer[i]*f[v][j]

      dp_buffer = dp_buffer1

     f[cur_node] = dp_buffer
}
```

Now, lets analyse complexity. At each node with n children, we are doing a computation of $n * K^2$, so total complexity is $O(N * K^2)$.

Another similar problem is : We are given a tree with N nodes and a weight assigned to each node, along with a number K. The aim is to delete enough nodes from the tree so that the tree is left with precisely K leaves. The cost of such a deletion is the sum of the weights of the nodes deleted. What is the minimum cost to reduce to tree to a tree with K leaves? Now, think about the states of our DP. Derive a recurrence. Before actually proceeding to the solution give it atleast a good thinking. Find solution here.

Problem 4:

==========

Given a tree T, where each node i has cost C_i. Steve starts at root node, and navigates to one node that he hasn't visited yet at random. Steve will stop once there are no unvisited nodes. Such a path takes total time equal to sum of costs of all nodes visited. What node should be assigned as root such that expected total time is minimised?

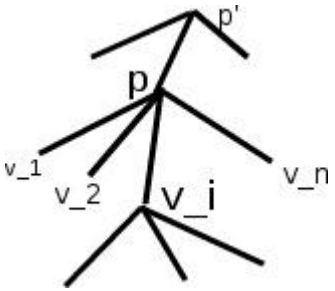
First, lets say tree is rooted at node 1, then we calculate total expected time for the tree formed. We define $f(V)$ as expected total time if we start at node V and visit in subtree of V. If V has children $v_1, v_2, ..., v_n$, we can say that $f(V) = C_V + \frac{\sum_{i=1}^{n} f(v_i)}{n}$, since with same probability we'll move down each of the children.

Now, we have to find a node v such that if we root tree at v, then $f(v)$ is minimised. Now, $f(v)$ is dependent on where we root the tree. If we do a brute force, it'll be $O(N^2)$. We need faster than this to pass.

We'll try to iterate over all nodes V and quickly calculate the value of $f(V)$ if tree is rooted at V. We need to see the contribution of $f(\text{parent}(V))$ if tree is rooted at V. We already know the contribution of children of V. So, if we define one more quantity $g(V)$ as the expected total time at node $\text{parent}(V)$, if we don't consider contribution of subtree of V.

Now, if I want to root my whole tree at V, then total expected time at this node will be $C_V + \frac{g(V) + \sum_{i=1}^{n} f(v_j)}{n+1}$. To realise this is correct, have a look at definition of $g(V)$.

Lets see how we can calculate $g(V)$. Keep referring to image below this paragraph while reading. Consider a node p which has parent p' and children $v_1, v_2, ..., v_n$. Now, lets try to find $g(v_i)$. $g(v_i)$ means root tree at node p and don't consider subtree of v_i for calculating $f(p)$. We can say that $g(v_i) = C_p + \frac{g(p) + f(v_1) + ... + f(v_{i-1}) + f(v_{i+1} + ... + f(v_n)}{n}$, since $g(p)$ gives us the expected total time at p' without considering subtree of p. We divide by n, because p will have n children i.e. $p', v_1, ..., v_{i-1}, v_{i+1}, ..., v_n$.



We can calculate both functions f and g recursively in $O(N)$.

Problem 5:

==========

Another very interesting problem goes as: Given two rooted trees T_1 and T_2, you want to make T_1 as structurally similar to T_2. For doing that you can insert leaves one by one in any of the trees. You have to tell the minimum number of insertions required to do so.

Lets say both trees are rooted at nodes 1. Now, say $T1_1$ has children $u_1, u_2, ..., u_n$ and $T2_1$ has children $v_1, v_2, ..., v_m$, then we are going to create a mapping between nodes in set u and v i.e. we are going to make subtree of some node u_i exactly same as v_j, for some i, j, by adding required nodes. If $n \neq m$, then we are going to add the whole subtree required.

Now, how do we decide which node in $T1$ is mapped to which in $T2$. Again, we use DP here. We define $\mathrm{dp}(i, j)$ as minimum additions required to make subtree of node i in $T1$ similar to subtree of node j in $T2$. We need to come up with a recurrence.

Lets say node i has children $u_1, u_2, ..., u_n$ and node j has children $v_1, v_2, ..., v_m$. Now, if we assign node u_i with node v_j, then the cost is going to be $\mathrm{dp}(u_i, v_j)$. Now, to all nodes in u, we have to assign nodes from v such that total cost is minimised. This can be solved by solving assignment problem. In assignment problem there is a cost matrix C, where $C(i, j)$ denotes cost if task i is assigned to person j. Our aim is to assign one task to one person such that total cost is minimised. This can be done in $O(N^3)$, if there are N tasks. Here in our problem $C(i, j) = \mathrm{dp}(v_i, v_j)$ and by solving this assignment problem, we can get value of $\mathrm{dp}(i, j)$.

Total complexity of this solution is $O(N^3)$, where N is maximum number of nodes in T_1 and T_2.

That's the end of it. Now time for some person advice :) The more you practice DP/DP on trees, the more comfortable you are going to be. So, get on your practice shoes and run over the obstacles! There are lot of DP on trees problem which you can try to solve and if you don't get the solution look at the tutorial/editorial, if you still don't get solution ask on various platforms.

Problems for practice:
1 2 3 4 5 6 7 (Solution for 7) 8 9 10 11 12 13

tutorial, **dynamic programming**, **trees**

+251 darkshadows 3 years ago 68

Comments (68)

Write comment?

k-maxx

3 years ago, # | ← Rev. 2 +1

thankyou **darkshadows** bro

→ Reply

Manurung

3 years ago, # | ← Rev. 2 0

What is C[V] stand for in the problem 1 Description?

→ Reply

darkshadows

3 years ago, # ^ | 0

Its the number of coins attached with node V.

→ Reply

code_fille

3 years ago, # | 0

Love u man! :D Please keep putting up more interesting tutorials on anything everything. They re amazing.

→ Reply

Swistakk

3 years ago, # | +13

Note that in problem 3., if we will iterate to size of a min(size of subtree, k), then complexity will be $O(n \cdot min(n, k^2))$, which can be faster by an order of magnitude. Why? I will leave you that as an exercise, which I highly encourage you to solve.

→ Reply

sanchit_h

3 years ago, # ^ | ☆ 0

Consider K >> N and a tree of size N such that it consists of a chain of length N/2 and N/2 nodes attached to the tail of the chain. Now if we root the tree at the head of the chain, wouldn't the actual runtime be O(N^3) because we do a total work of O(N^2) on N/2 nodes.

→ Reply

retrograd

21 month(s) ago, # ^ | ☆ ← Rev. 2 0

I've actually seen a proof somewhere that what you described is actually $O(n * min(n, k)) = O(n * k)$. It relies on the fact that you do k^2 work only on nodes that have two children of size at least k and there's just n / k such nodes and similar observations.

→ Reply

tdas

new, 3 months ago, # ^ | ☆ 0

Can someone share this proof?

→ Reply

gabrielsimoes

18 months ago, # ^ | ☆ +4

I know this is rather old, but as a reference, I'll leave the link to a problem that requires this optimization: http://codeforces.com/problemset/problem/815/C

The contest announcement comments and the editorial and its comments are a good resource to learn about it, see the proof, etc.

→ Reply

rishus23

17 months ago, # ^ | ☆ 0

Swistakk can you please explain why is it so? I have seen it in few places but couldn't understand it completely.

→ Reply

3 years ago, # | ☆ +5

Implementation of problem 2 : diameter = max(diameter, f[v] + g[v]);

karunasagark

Shouldn't this be diameter = max(diameter, max(f[V], g[V])); ?

→ Reply

darkshadows

3 years ago, # ^ | ☆ 0

Fixed that. Thanks!

→ Reply

grv95

3 years ago, # | ☆ 0

In problem1,instead of

sum1 += dp1[v];

.... dp1[V] = C[V] + sum1;

shouldn't it be sum1+=dp2[v];

because on including a vertex,all of it's children can't be included.

→ Reply

darkshadows

3 years ago, # ^ | ☆ 0

Fixed.

→ Reply

darkshadows

3 years ago, # | ☆ 0

Auto comment: topic has been updated by ***darkshadows*** *(previous revision, new revision, compare).*

→ Reply

3 years ago, # | ☆

Shouldn't "dp_buffer[i + j] += f[v][j]*f[v][i]" (in pseudocode of problem 3) be "dp_buffer[i+j]

sudharsansai

Shouldn't "dp_buffer[i + j] += f[v][i] * f[v][j]" (in pseudocode of problem 3) be "dp_buffer[i+j] +=f[cur_node][i]*f[v][j]" ?
Correct me if I am wrong ..

← Rev. 2 0

→ Reply

mskd96

3 years ago, # ^ | ☆ 0

I think it should be "dp_buffer[i+j] += dp_buffer[i]*f[v][j]". This is because, we should multiply existing number of subtrees containing i nodes with the number of subtrees containing j nodes in which v is the root.

→ Reply

sudharsansai

3 years ago, # ^ | ☆ 0

Yep..Now its fine .

→ Reply

sudharsansai

3 years ago, # ^ | ☆ 0

Oh ..One more doubt. Shouldn't dp_buffer[1] be initialised to '1' for each vertex.

→ Reply

3 years ago, # | ☆ 0

In problem 3 (or any), you have taken **node 1** as a root, but could you prove that how the solution remains valid if we take **any node** as a root ??**

yashpal1995

I got the intuition that suppose we make any other node as root, let's say **r** (instead of **1**) then the extra answer added in **r** due to the subtree containing **node 1** is already included in answer of **node 1** when we are taking **node 1** as root.

Or is it right prove that: the *answer* we need to calculate is independent of root of the tree, so it *does not* depend on the choices of *root* ..

→ Reply

3 years ago, # | ☆ 0

*Auto comment: topic has been updated by **darkshadows** (previous revision, new revision*

darkshadows

Auto comment: topic has been updated by darkshadows (previous revision, new revision, compare).

→ Reply

karunasagark

3 years ago, # | ☆ 0

Problem 4: Could somebody explain how would one go about implementing this? g and f are interdependent; g(v) depends on values from siblings and grandparent while f(v) depends on values from children.

→ Reply

hrithik96

23 months ago, # ^ | ☆ 0

Use two functions with memoization:

1) To Calculate f: Initialize f[vertex] with the value of cost[vertex], then use recursion at all it's children nodes. Then, use another function to calculate g, and call that function within this function.

2) To Calculate g: Initialize g[vertex] with cost[parent[vertex]] if it's not the root. Then recursively calculate the value of f for all the children of it's parent excluding the current vertex.

3) Call f on the root node in the main function. It will calculate all the f and g values, then calculate the total expected time for each of the nodes using a loop. This will be linear due to memoization.

This is how I implemented it, there can be tweaks to further fasten up but this is the basic way to implement it.

Make sure that the order is correct.

→ Reply

Deathstar

3 years ago, # | ☆ ← Rev. 2 0

Can someone explain how to solve Problem 11?

→ Reply

→ Reply

3 years ago, # | ☆ ← Rev. 2 0

problem 3 : someone please tell me what's wrong with my dfs function.

void dfs(int V,int pv) { f[V][1]=1; mem(dp1); dp1[0]=1;

```
for(int l=0; l<vec[V].size(); l++)
{
    int v=vec[V][l];
    if(v==pv) continue;
    dfs(v,V);
    mem(dp2);
    for(int i=0; i<=k; i++)
    {
        for(int j=0; j<=k-i; j++)
        {
            dp2[i+j]+=dp1[i]*f[v][j];
        }
    }
    for(int i=1; i<=k; i++)
    {
        dp1[i]=dp2[i];
    }
}
for(int i=1; i<=k; i++)
{
    f[V][i+1]=dp1[i];
}

}
```

backPacker

→ Reply

3 years ago, # ^ | ☆ 0

never mind. solved

→ Reply

backPacker → Reply

effutue

18 months ago, # ^ | ☆ 0

BlueGold Can you Please post what was the problem in your code? I am also stuck here.

→ Reply

backPacker

18 months ago, # ^ | ☆ 0

I think the problem was , i declared both the dp arrays globally, whereas these should be declared locally (inside the dfs function)

→ Reply

effutue

18 months ago, # ^ | ☆ 0

Thanks a lot, worked for me as well!

→ Reply

siddharth.s.atwork

17 months ago, # ^ | ☆ 0

it should be for(int i=1; i<=k; i++) dp1[i]+=dp2[i];

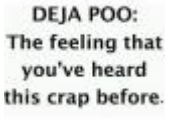
→ Reply

naruto09

3 years ago, # | ☆ 0

can anyone help me understand problem number 3..I have been trying but i dont seem to get the explanation clearly

→ Reply



SarvagyaAgarwal

3 years ago, # | ☆ 0

In problem 2 :
Instead of $g(V) = 1 + sum\ of\ two\ max\ elements\ from\ set\ \{f(v1), f(v2), ..., f(vn)\}$
shouldn't it be $g(V) = 2 + sum\ of\ two\ max\ elements\ from\ set\ \{f(v1), f(v2), ..., f(vn)\}$.

→ Reply

wineColoredDays

3 years ago, # ^ | ☆ 0

I think the first one is correct as he is counting number of verticles . See, f[V] = 1. Correct me if i'm wrong.

→ Reply

sumit.asr

3 years ago, # ^ | ☆ 0

Yes it should be g(V) = 2 + sum of two max elements from set {f(v1), f(v2), ..., f(vn)} because we need to consider length of 2 edges .

→ Reply

SProf

3 years ago, # | ☆ 0

in problem 2 why f[v]=1 when we have only 1 vertex?

→ Reply

Sullipopa

2 years ago, # ^ | ☆ -8

Yes, it's a typo.

→ Reply

JavaKurious

3 years ago, # | ☆ 0

In the explained Problem 3, are subtree and sub tree different terms ?

→ Reply

shahidul_brur

2 years ago, # | ☆ 0

In problem 1, you said, "Our final answer is maximum of two case i.e. $\max(\mathrm{dp1}(1) + \mathrm{dp2}(1))$ " Shouldn't it be max(dp1(1), dp2(1)) ?

→ Reply

2 years ago, # ^ | ☆ 0

yaa, its a typo!

ankeshgupta007

→ Reply

2 years ago, # | ☆ 0

In problem-2, won't g(v) always be greater than or equal to f(v)?

ShalinShah28

f(v) = 1 + max(f(v1),f(v2)..)

g(v) = 2 + sum of two max elements from (f(v1),f(v2)...)

Hence, g(v) >= f(v)?

→ Reply

2 years ago, # | ☆ 0

In Problem 3, you have written :

rahulkhairwar

$$dp1(i,j) = \sum_{k=0}^{K} dp1(i-1, j-k) * f(i,k)$$

But, what if the `j` value we are currently looking at is less than K?

Shouldn't the summation be $\sum_{k=0}^{j}$?

→ Reply

Sullipopa

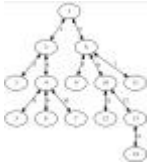
2 years ago, # ^ | ☆ -8

Well, it should be *min(j,K)*.

→ Reply

2 years ago, # | ☆ ← Rev. 2 +3

nvm



vatsal

→ Reply

Sullipopa

2 years ago, # | ☆ ← Rev. 2 -8

Shouldn't you initialize *f[v]=0*, instead of *f[v]=1*.? Since for a leaf node, the length of the path in its subtree will be 0.

Code.

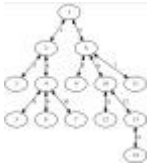
→ Reply

hulk_baba

2 years ago, # | ☆ 0

Can the problem 1 which you explained not be solved by greedy... If I take all the nodes at a level and sum alternate nodes and find maximum of both stating with zero and starting with one.. would yield me correct answer?

→ Reply



vatsal

23 months ago, # | ☆ 0

The practice problem 13 is not linked to any website.

→ Reply

AaronSwartz

21 month(s) ago, # | ☆ 0

Where can I found a problem like Problem 3?

→ Reply

PrakharJain

20 months ago, # ^ | ☆ 0

This is somewhat like this : http://codeforces.com/contest/816/problem/E I'm not completely sure though.

→ Reply

deadpool_18

18 months ago, # | ☆ 0

has anyone got any idea where were these questions taken from... ?

→ Reply

pieby2

17 months ago, # | ☆ 0

In problem 3rd, should'nt f(i,j) be written as f(i,j)+1 in the second part because there will be case when the Node i is not choosen

→ Reply

shahidul_brur

17 months ago, # | ☆ 0

Can anyone give the problem links for all five problems, which are discussed in the post?

→ Reply

shahidul_brur

17 months ago, # | ☆ 0

Link to problem 1 in discussion: https://www.e-olymp.com/en/contests/7461/problems/61451

→ Reply

nimom

16 months ago, # | ☆ 0

This tutorial is great! But Problem 3 is not clear to me. :(What do you mean by your definition of sub tree and the actual definition of sub tree?

→ Reply

siddharth.s.atwork

16 months ago, # ^ | ☆ +1

Yes it is a bit confusing. I think first of all he tried to explain how can you find the number of subtrees of a given tree. I will try to explain what I understood.

lets take a tree and make it rooted at 1 where node 2 and 3 are connected directly to node 1 and we know that a node itself a subtree. We will define a recursive function F(V) means number of subtrees rooted at V and with dp we will define dp[V]=1 as base case as we know that every node will contain at least one subtree that is itself. so in recursively while counting subtrees we have two option whether to include a node or not. Lets try to

understand this way we will make sets for node node 2 we have (null 2) null when we are

understand this way we will make sets for node node 2 we have (null,2) null which we are not choosing 2 and 2 for when we are choosing itself. similary for node three we have (null,3) that's why we used 1+f(v) in problem 3. So product of these subsets gives us (null,null),(null,3),(2,null),(2,3) where (null,null) means when we are neither choosing 2 nor 3 which gives us (1) alone as a subtree ,(null,3) means when we chose only 3 so we get (1,3) as subtree with (2,null) we got (1,2) and with (2,3) we got (1,2,3) while we already had (2) and (3) rooted at themselves so total number of subtrees are (1),(2),(3),(1,2),(1,3),(1,2,3).I hope it's true and makes sense.

→ Reply

KAISER17

14 months ago, # | ☆ 0

can you suggest any codeforces or any other online judge problems which are similar to problem 3?

→ Reply

sebach

14 months ago, # | ☆ 0

Hey, really nice post, thank you very much!

In the code for calculating the diameter, you forgot to change the code of g[V]=1 + ... as you changed in the explanation. The "2" for "1"

→ Reply

palindromeguy

7 months ago, # ^ | ☆ -8

Actually we are counting the no of edges and not the vertices. That's why the +2

→ Reply

alphaWizard

14 months ago, # | ☆ ← Rev. 2 0

[deleted]

→ Reply

11 months ago, # | ☆ 0

What does dp_buffer and dp_buffer1 represent in problem 3 ?

→ Reply

karanbatra

10 months ago, # | ☆ 0

ultra_instinct

In discussion problem 5, how does the total complexity becomes $O(N^3)$? If we consider a particular node from T_1, then matching it's children with children of all the nodes from T_2 should give $O(N^3)$. so, overall complexity should be $O(N^4)$. Am I calculating wrong somewhere?

→ Reply

8 months ago, # | ☆ 0

prathmesh_01

Can you please explain how to solve first and second pratice problem, I dont understand the editorial;(

→ Reply

7 months ago, # | ☆ 0

knakuul853

Thank you for such clear and concise tutorial

→ Reply

6 months ago, # | ☆ 0

Patti

Tanks, this blog is really really helpful orz!!!

→ Reply

6 months ago, # | ☆ 0

devarshi09

Awesome stuff!

→ Reply

5 months ago, # | ☆ 0

prashantkn94

Is there any judge where we can submit problem 4?

→ Reply

5 months ago, # | ☆ 0

Is there any code for problem 5?

Is there any code for problem 3?

→ Reply

Testduk

atlasworld

5 months ago, # | ☆ 0

for problem 1 : this can also be the solution :

```
#include<bits/stdc++.h>
using namespace std;
vector<int> v[12345];
int dp[12345];
void dfs(int curr,int parent)
{
    for(auto child:v[curr]){
        if(child==parent) continue;
        dfs(child,curr);
    }
    int take = dp[curr];
    int not_take = 0;
    for(auto child:v[curr]){
        if(child==parent) continue;
        not_take+=dp[child];
    }
    dp[curr]=max(dp[curr],max(take,not_take));
}
int main()
{
    int nodes;
    cin>>nodes;
    for(int i=1;i<nodes;++i){
        int x,y;
        cin>>x>>y;
        v[x].push_back(y);
        v[y].push_back(x);
    }
```

```
    // enter value of each node
    for(int i=1;i<=nodes;++i){
        cin>>dp[i];
    }
    dfs(1,-1);
    cout<<dp[1]<<endl;
    cout.flush();
    return 0;
}
```

→ Reply

atlasworld

4 months ago, # | ☆ 0

shouldn't g[v] = 1+.... in the code

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
vector<ll> gr[12345];
ll f[12345],g[12345];
ll diameter=-1;
void dfs(ll curr,ll par){
      vector<ll> v;
      for(auto child:gr[curr]){
            if(child == par) continue;
            dfs(child,curr);
            v.push_back(f[child]);
      }
      sort(v.begin() , v.end());
      f[curr] = 1;
      if(v.empty() == false) {
            f[curr] = 1 + v[v.size()-1];
      }
      if(v.size()>=2){
            g[curr] = 1 + v[v.size()-1] + v[v.size()-2];
```

```
            g[curr] = 1 + v[v.size()-1] + v[v.size()-2];
        }
        diameter = max(diameter , max(f[curr] , g[curr]));

    }
    int main()
    {
        ll n;
        cin>>n;
        ll t=n-1;
        while(t--){
            ll f ,t ;
            cin >> f>> t;
            gr[f].push_back(t);
            gr[t].push_back(f);
        }

        dfs(1,0);
        cout<<diameter<<endl;
        return 0;
    }
```

→ Reply

ITMO UNIVERSITY