Algorithm to find diameter of a tree using BFS/DFS. Why does it work?

This link provides an algorithm for finding the diameter of an undirected tree **using BFS/DFS**. Summarizing:

> Run BFS on any node s in the graph, remembering the node u discovered last. Run BFS from u remembering the node v discovered last. d(u,v) is the diameter of the tree.

Why does it work ?

Page 2 of this provides a reasoning, but it is confusing. I am quoting the initial portion of the proof:

> Run BFS on any node s in the graph, remembering the node u discovered last. Run BFS from u remembering the node v discovered last. d(u,v) is the diameter of the tree.
>
> Correctness: Let a and b be any two nodes such that d(a,b) is the diameter of the tree. There is a unique path from a to b. Let t be the first node on that path discovered by BFS. If the paths p_1 from s to u and p_2 from a to b do not share edges, then the path from t to u includes s. So
>
> $d(t, u) \geq d(s, u)$
>
> $d(t, u) \geq d(s, a)$
>
> ....(more inequalities follow ..)

http://i61.tinypic.com/rji9uq.png

The inequalities do not make sense to me.

`algorithms` `graphs` `search-algorithms` `trees` `graph-traversal`

edited Apr 13 '17 at 12:48
Community ♦
1

asked Mar 20 '14 at 7:09
curryage
216 2 3 8

I don't find the quote in the linked question. – Raphael ♦ Mar 20 '14 at 13:30

1 Try replacing "do not share edges" with "do not share vertices" in the solution. – Yuval Filmus Mar 20 '14 at 13:42

You are using only BFS, not DFS. – Thumbnail May 24 '17 at 11:14

5 Answers

All parts of proving the claim hinge on 2 crucial properties of trees with undirected edges:

- 1-connectedness (ie. between any 2 nodes in a tree there is exactly one path)
- any node can serve as the root of the tree.

Choose an arbitrary tree node s. Assume $u, v \in V(G)$ are nodes with $d(u, v) = diam(G)$. Assume further that the algorithm finds a node x starting at s first, some node y starting at x next. wlog $d(s, u) \geq d(s, v)$. note that $d(s, x) \geq d(s, y)$ must hold, unless the algorithm's first stage wouldn't end up at x. We will see that $d(x, y) = d(u, v)$.

The most general configuration of all nodes involved can be seen in the following pseudo-graphics (possibly $s = z_{uv}$ or $s = z_{xy}$ or both):

```
(u)                                              (x)
  \                                              /
   \                                            /
    \                                          /
     ( z_uv )---------( s )----------( z_xy )
    /                                          \
   /                                            \
  /                                              \
(v)                                              (y)
```

we know that:

1. $d(z_{uv}, y) \leq d(z_{uv}, v)$. otherwise $d(u, v) < diam(G)$ contradicting the assumption.
2. $d(z_{uv}, x) \leq d(z_{uv}, u)$. otherwise $d(u, v) < diam(G)$ contradicting the assumption.
3. $d(s, z_{xy}) + d(z_{xy}, x) \geq d(s, z_{uv}) + d(z_{uv}, u)$, otherwise stage 1 of the algorithm wouldn't have stopped at x.

4. $d(z_{xy}, y) \geq d(v, z_{uv}) + d(z_{uv}, z_{xy})$, otherwise stage 2 of the algorithm wouldn't have stopped at y.

1) and 2) imply .

$$\begin{aligned} d(u,v) &= d(z_{uv}, v) + d(z_{uv}, u) \\ &\geq d(z_{uv}, x) + d(z_{uv}, y) = d(x,y) + 2\, d(z_{uv}, z_{xy}) \\ &\geq d(x,y) \end{aligned}$$

3) and 4) imply

$$\begin{aligned} & d(z_{xy}, y) + d(s, z_{xy}) + d(z_{xy}, x) \\ &\geq d(s, z_{uv}) + d(z_{uv}, u) + d(v, z_{uv}) + d(z_{uv}, z_{xy}) \end{aligned}$$

equivalent to .

$$\begin{aligned} d(x,y) &= d(z_{xy}, y) + d(z_{xy}, x) \\ &\geq 2 * \ d(s, z_{uv}) + d(v, z_{uv}) + d(u, z_{uv}) \\ &\geq d(u,v) \end{aligned}$$

therefore $d(u,v) = d(x,y)$.

analogue proofs hold for the alternative configurations

```
                (u)                          (x)
                 \                           /
                  \                         /
                   \                       /
     ( s )---------( z_uv )----------( z_xy )
                   /                       \
                  /                         \
                 /                           \
                (v)                          (y)
```

and

```
                         (x)        (u)
                         /            \
                        /              \
                       /                \
     ( s )---------( z_xy )----------( z_uv )
                       \                /
                        \              /
                         \            /
                         (y)        (v)
```

these are all possible configurations. in particular, $x \notin path(s,u), x \notin path(s,v)$ due to the result of stage 1 of the algorithm and $y \notin path(x,u), y \notin path(x,v)$ due to stage 2.

edited Mar 20 '14 at 23:37

answered Mar 20 '14 at 23:28

collapsar
810 5 8

(1) Regarding the first graphic, shouldn't the path from s to x always contain vertices u and v in some order since they are present on tree generated by BFS? (2) Could you clarify how the inequalities are obtained ? (3) Since the BFS starting from s and that starting from x contain u,v somewhere on the path, I believe the graphic should be as shown in the link imgur.com/jQ94erY . How does the reasoning you provided apply here ? – curryage Mar 21 '14 at 5:39

@curryage note that the tree is given and not being constructed by the bfs! specific answers: ad 1) no. imagine a refinement of the tree in graphics (1) by adding arbitrarily many nodes on the edge (s, z_{xy}) and exactly 1 node on the edge (z_{xy}, x). the first stage bfs will then end at x. ad 2) which inequality/ies are unclear? we are always assuming that (u, v) be a path the length of the graph's diameter $diag(G)$. this is well defined as G is 1-connected. ad 3) no: 3.1 there is more than 1 path between any 2 nodes apart from (s, y), so the graph is not a tree. ... – collapsar Mar 21 '14 at 15:27

@curryage ... 3.2 $d(x,y) > d(u,v)$; this is impossible as $d(u,v) = diam(G)$ by assumption and a graph's diameter is the maximal minimum distance between any two nodes. in the case of a tree there is exactly 1 path between any 2 nodes, so the definition reduces to 'maximum distance between any two nodes'. – collapsar Mar 21 '14 at 15:31

The intuition behind is very easy to understand. Suppose I have to find longest path that exists between any two nodes in the given tree.

After drawing some diagrams we can observe that the longest path will always occur between two leaf nodes(nodes having only one edge linked). This can also be proved by contradiction that if longest path is between two nodes and either or both of two nodes is not a leaf node then we can extend the path to get a longer path.

So one way is to first check what nodes are leaf nodes, then start BFS from one of the leaf node to get the node farthest from it.

Instead of first finding which nodes are leaf nodes , we start BFS from a random node and then see which node is farthest from it. Let the node farthest be x. It is clear that x is a leaf node. Now if we start BFS from x and check farthest node from it, we will get our answer.

But what is the guarantee that x will be a end point of a maximum path?

Let's see by an example :-

```
      1
    / /\ \
   6 2  4 8
         \ \
          5 9
            \
             7
```

Suppose I started my BFS from 6. The node at maximum distance from 6 is node 7. Using BFS we can get this node. Now we star BFS from node 7 to get node 9 at maximum distance. Path from node 7 to node 9 is clearly the longest path.

What if BFS that started from node 6 gave 2 as the node at maximum distance. Then when we will BFS from 2 we will get 7 as node at maximum distance and longest path will be then 2->1->4->5->7 with length 4. But the actual longest path length is 5. This cannot happen because BFS from node 6 will never give node 2 as node at maximum distance.

Hope that helps.

answered May 24 '17 at 8:31
MayankPratap
49 1 2

Here's a proof that follows the MIT solution set linked in the original question more closely. For clarity, I will use the same notation they use so the comparison can be more easily made.

Suppose we have two vertices a and b such that the distance between a and b on the path $p(a, b)$ is a diameter, e.g. the distance $d(a, b)$ is maximum possible distance between any two points in the tree. Suppose we also have a node $s \neq a, b$ (if $s = a$, then it would be obvious that the scheme works, since the first BFS would get b, and the second would return to a). Suppose also that we have a node u such that $d(s, u) = \max_x d(s, x)$.

Lemma 0: Both a and b are leaf nodes.

Proof: If they weren't leaf nodes, we could increase $d(a, b)$ by extending the endpoints to leaf nodes, contradicting $d(a, b)$ being a diameter.

Lemma 1: $\max[d(s, a), d(s, b)] = d(s, u)$.

Proof: Suppose for the sake of contradiction that both $d(s, a)$ and $d(s, b)$ were strictly less than $d(s, u)$. We look at two cases:

Case 1: path $p(a, b)$ does *not* contain vertex s. In this case, $d(a, b)$ cannot be the diameter. To see why, let t be the unique vertex on $p(a, b)$ with the smallest distance to s. Then, we see that $d(a, u) = d(a, t) + d(t, s) + d(s, u) > d(a, b) = d(a, t) + d(t, b)$ since $d(s, u) > d(s, b) = d(s, t) + d(t, b) > d(t, b)$. Similarly, we would also have $d(b, u) > d(a, b)$. This contradicts $d(a, b)$ being a diameter.

Case 2: path $p(a, b)$ contains vertex s. In this case, $d(a, b)$ *again* cannot be the diameter, since for some vertex u such that $d(s, u) = \max_x d(s, x)$, both $d(a, u)$ and $d(b, u)$ would be greater than $d(a, b)$.

Lemma 1 gives the reason why we start the second Breadth-First search at the last-discovered vertex u of the first BFS. If u is the unique vertex with the greatest possible distance from s, then by Lemma 1, it *must* be one of the endpoints of some path with a distance equal to the diameter, and hence a second BFS with u as the root unambiguously finds the diameter. On the other hand, if there is at least one other vertex v such that $d(s, v) = d(s, u)$, then we know that the diameter is $d(a, b) = 2d(s, u)$, and it doesn't matter whether we start the second BFS at u or v.

edited Jan 15 at 21:33

answered Jan 15 at 5:27
xdavidliu
158 6

Awesome. Thanks for posting this answer. I am surprised that this answer didn't receive any upvotes. – Zephyr Feb 27 at 17:55

By the definition of BFS, the distance (from the starting node) of each node explored is either equal to the distance of the previous node explored or greater by 1. Thus, the last node explored by BFS will be among those farthest from the starting node.

Thus, the algorithm of using BFS twice amounts to "Pick an arbitrary node x. Find the node a farthest from x (last node found by BFS starting from x). Find the node b farthest from a (last node found by BFS starting from a).", which thus finds two nodes of maximum distance from eachother.

answered Jan 3 '17 at 16:15
Extrarius
651 3 9

1 Thanks for the answer with the intuition. However, the "thus" in the your last sentence is not obvious. Why does

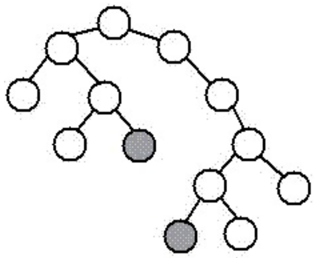
that follow? Why does the node farthest from x have to be one of the two nodes at maximum distance from each other? It seems like that needs some proof. – D.W. ♦ Jan 3 '17 at 16:41

I'm not sure how to construct such a proof. I feel like the converse is intuitively true: if two nodes are at maximum distance from each other, then, for any given node, one of the two is at the greatest possible distance from it. – Extrarius Jan 3 '17 at 20:41

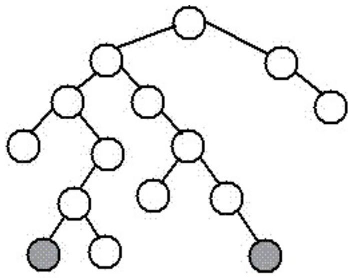
The "intuitively true" claim isn't true in general for general graphs. See the graph in cs.stackexchange.com/a/213/755, and imagine starting the BFS from node v (i.e., let $x = v$); then it will pick $a = u$ and find the node b at greatest distance from a, but that doesn't find the two nodes of maximum distance from each other. So the claimed statement, if true, must rely on some special property of trees that doesn't hold for general graphs. – D.W. ♦ Jan 3 '17 at 23:53

Yes, but this question specifies undirected trees, which is the context I'm intuiting in. Barring cycles and directed edges makes many graph problems significantly simpler to reason about. – Extrarius Jan 4 '17 at 0:41

First run a DFS from a random node then the diameter of a tree is the path between the deepest leaves of a node in its DFS subtree:



diameter, 9 nodes, through root



diameter, 9 nodes, NOT through root

edited Dec 16 '16 at 9:30
Yuval Filmus
176k 12 163 320

answered Dec 16 '16 at 8:48
seddik11
1

4 Why does this work? – Yuval Filmus Dec 16 '16 at 9:30