


Computer Science Stack Exchange is a question and answer site for students, researchers and practitioners of computer science. It only takes a minute to sign up.

Join this community

Anybody can ask a question

Anybody can answer

The best answers are voted up and rise to the top

COMPUTER SCIENCE

# Does reachability belong to P?

Asked 8 years ago    Active 7 years, 4 months ago    Viewed 1k times

▲

Reachability is defined as follows: a digraph  $G = (V, E)$  and two vertices  $v, w \in V$ . Is there a directed path from  $v$  to  $w$  in  $G$ ?

▼

-2

Is it possible to write a polynomial time algorithm for it?

▼

I asked this question on mathematics and got no answer by far.

★

🕒

complexity-theory

graphs

time-complexity


Share   Cite   Improve this question   Follow

edited May 28 '13 at 7:11

 **Raphael** ♦

69.7k   🏆27   🗳️160   🔄354

asked May 26 '13 at 5:57

 **Gigili**

2,113   🏆3   🗳️20   🔄29

- 3   This was fully answered by my comment at [Mathematics](#) a couple of minutes after you posted it: [math.stackexchange.com/questions/401884/...](http://math.stackexchange.com/questions/401884/...) – [András Salamon](#) May 26 '13 at 9:57
- @AndrásSalamon: Thank you for your comment over there, but I wouldn't say it's fully answered since the answers different are completely different. Also the paper you linked to wasn't really related to what I asked. – [Gigili](#) May 26 '13 at 13:25
- 3   Strange that you ask such a question after receiving 18 upvotes for this answer: [cs.stackexchange.com/a/308/6716](http://cs.stackexchange.com/a/308/6716) . OK, it is quoted, but nevertheless ... – [frafl](#) May 27 '13 at 21:53

## 3 Answers

Active

Oldest

Votes

▲

6

▼

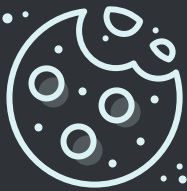
✓

🕒

Although you already know from the other answers that the question is solvable in polynomial time, I thought I would expand on the computational complexity of reachability since you used complexity terminology in your question.

Reachability (or st-connectivity) in digraphs is the prototypical [NL-complete problem](#) where *NL* stands for non-deterministic log space and we use deterministic log-space reductions (although I think it remains complete for  $NC^1$  reductions, too).

- To see why it is in *NL*, notice that you can guess a next vertex at every step and verify that it is connected to the previous vertex. A series of correct guesses exists if and only if there is a path from  $s$  to  $t$ .
- To see why is *NL*-hard, notice that the behavior of a non-deterministic Turing machine can be represented by a [configuration graph](#). The nondeterministic machine accepts only if there exists a path from the start configuration to an accept configuration, and if the machine only uses  $S(n)$  tape, then the configuration graph is of size  $O(|\Gamma|^{S(n)})$  where  $\Gamma$  is the tape alphabet. If  $S(n)$  is logarithmic, then the problem is in *NL*.



### Your privacy

By clicking "Accept all cookies", you agree Stack Exchange can store cookies on your device and disclose information in accordance with our [Cookie Policy](#).

Accept all cookies

Customize settings

So why should I care? Well, we know lots of things about *NL*; one of those is that is in  $AC^1$ . There are tighter facts that can be useful:

$AC^1 \subseteq NC^1 \subseteq P$  and  $AC^1 \subseteq PTIME$  (even on a deterministic machine you don't need that much space to solve the problem).

In circuit model, your question can be solved by a polynomial sized circuit of depth  $O(S(n))$ . This shows that the problem is parallelizable since [Nick's Class](#) captures the idea of quick solutions.

It is not harder (up to log-space reductions) than membership checking in context-free languages.

If the graph is undirected then we believe the question is significantly easier. In particular, it is in *SL* (log space) under first-order reductions (Reingold 2004; [pdf](#)).

▲ Yes, this problem is solveable in linear time,  $O(|V| + |E|)$  to be precise.

5 The two classic solutions to this are Breadth-First search and Depth-First search.

▼ The algorithms basically look like this:

⌛

```
current = v
while (current has an edge to an unmarked vertex)
  if current == w
    return true
  mark current as visited
  for each u where (v,u) is in E
    add u to the Open List
  current = a vertex from the Open List
return false
```

BFS uses a queue as the open list, adding to the back and taking from the front. DFS uses a stack. In any implementation like this, each node and each edge are visited at most once, so the algorithm runs in linear time.

▲ Yes, it is in P.

3 The natural algorithm for is very simple, so simple it doesn't really serve as a learning experience to state it here (it's readily available in almost any text or on the web).

▼ To put you on the right path:

- ⌛
1. What algorithms do you know for exploring a graph?
  2. If you were at the start vertex  $v$ , what's the obvious way of trying to find  $w$ ?



Your privacy

By clicking “Accept all cookies”, you agree Stack Exchange can store cookies on your device and disclose information in accordance with our [Cookie Policy](#).

- Accept all cookies
- Customize settings