Analysis of Algorithms: General Approach

CSCI 432

November ~~8~~ 12, 2023

Name: Class

Who did you work with today? Everyone!.

What algorithm are you analyzing today?

Answer

Whatever First Search (WFS)

ex: DFS (D = depth)
BFS (B = breadth)
Best FS

1 What?

When analyzing algorithms, we first ask WHAT? That is, what is the problem that we want solved? Typically, in this description, we need to understand both what are the inputs *and* what are the outputs of the algorithm. When describing WHAT, it should be independent of HOW.

So, describe the WHAT for the algorithm you wish to analyze today.

What is the input?

Answer

Given: Graph $G = (V, E)$

(sometimes) $v_0 \in V$, the start node ↘ if not given it, can choose one arbitrarily

(sometimes) query $q \in V$ ↳ is it in the graph? how do I get from v_0 to q?

What is the output?

Answer

a traversal of the graph (a way to "walk" from v_0 to every other vertex)

Use the previous two answers to concisely describe what is the problem?

Answer

graphs

2 How?

If you are presenting an algorithm, you should describe how it works in pseudocode. More importantly, explain what is going on in the pseudocode in the prose text. The prose should point to the algorithm. (Remember, in LaTex, algorithms are floating environments, so we need to be reminded to look at them.

Answer

```
WFS (v_0)
1:  X.push(v_0)
2:  while X ≠ ∅
3:      u ← X.pop
4:      if u is unmarked
5:          mark u
6:          for each edge e=(u,w) containing u
7:              X.push(w)
8:          end for
        end if
    end while
```

global data structures:
- graph $G=(V,E)$
- stack/queue to hold "processing next" list of verts

Often, it helps to walk through small examples. If you came up with the algorithm, this is a nice sanity check. If you are studying an algorithm given to you, this is helpful to better understand the algorithm.

Answer

Q1: What DS do we want to use if we want to do DFS? Q BFS? STACK (Best-First: priority queue)

Q2: What if I cared about a path from v to some $q \in V$. How do I augment this algo?

Store the "last visited" for each vertex?

Q3: example.

3 How Fast?

What is the runtime? If this is a recursive algorithm, you are expected to explicitly state the recurrence relation.

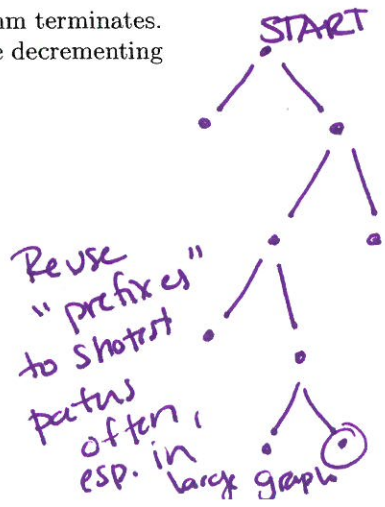
Answer

time complexity: $\Theta(V + E \cdot T)$

↑ time to push/pop

Sometimes, it is hard to pin down the runtime, yet we still want to ensure that the algorithm terminates. For this, we use decrementing functions to prove that loops/recursions terminate. What is the decrementing function for this algorithm?

Answer



Justify (=prove) why function is well-defined. That is, why is the output of the function a natural number?

Answer

Justify (=prove) why this decrements each time it reaches the top of the loop (or each time it goes into a new recursive call).

Answer

4 Why? The Loop Invariant

Analyzing the while loop

Finally, we prove why the algorithm works. Typically, this is done with a loop (or recursion) invariant. Here, we focus on setting up the loop invariant. What are the following statements (start with your best guess, then come back and revise if needed):

The loop guard G: $X \neq \emptyset$

$\neg G$: $\neg(X \neq \emptyset) \iff X = \emptyset$

The post-condition Q:
all vertices have been marked/visited

The pre-condition P: $X = \{v_0\}$ G is a graph $= (V, E)$; $v_0 \in V$

The loop invariant $L = L_i$: (a) X is a "set" of unmarked vertices in V for which there exists a marked path from $v_0 \in V$ to.

Can you use this loop invariant to prove partial correctness (Remember, there are three parts to partial correctness: Initialization, Maintenance, and End).

Initialization $(P \Rightarrow L)$
Answer
There exist a trivial path $\{v_0\}$ from v_0 to
Since $X = \{v_0\}$, we conclude $L(a)$

Maintenance $(L_i \wedge G \Rightarrow L_{i+1})$
Answer

End $(L \wedge \neg G \Rightarrow Q)$
Answer