Algorithms – CMSC-272

Breadth-First Search[1]

Instructor: László Babai

Single-source shortest path problem

The input is a rooted digraph $G = (V, E, s)$ where $s \in V$ is a special vertex called the *root* or *source* (hence the letter s). The *distance* from vertex u to vertex v, denoted $\mathrm{dist}(u, v)$, is the length of a shortest (directed) path from u to v. The distance is ∞ if v is not accessible from u. The algorithm determines, in linear time, the distance of each vertex from the root. It does so by organizing the vertices into layers by distance; the i-th layer consists of the vertices at distance i. (Layer zero is the root.) The last layer consists of the vertices at infinite distance, i. e., the vertices that are inaccessible from the root.

The algorithm scans all vertices that are accessible from the root. Vertices go through three phases: initially, each vertex is *unknown* (has not been discovered yet, status: WHITE). When a vertex v is *discovered* (first encountered by the algorithm), its status changes to GRAY. At that point the vertex enters a queue (ENQUEUE), and when it is its turn (it is DEQUEUE-d), its out-neighbors are discovered. Once all out-neighbors of v have been discovered, v is *finished* (status: BLACK). The vertices that are inaccessible from the root remain forever WHITE.

The root is discovered as part of the initialization. If vertex u (not the root) is discovered while exploring a vertex v, i. e., scanning the out-neighbors of v, then we say that v is the *parent* of u, denoted $v = p(u)$. The parent links organize the set of accessible vertices in a tree. We shall also maintain an array $d[1 \dots n]$. The intended meaning of $d[v]$ at the termination of the algorithm is $\mathrm{dist}(s, v)$; we shall have to verify, that this is indeed the case.

The digraph (V, E) is given in adjacency list representation, which is an array of linked lists. The array $V[1 \dots n]$ lists the vertices; and vertex i starts a linked list $\mathrm{Adj}[i]$ that lists all outneighbors of i in some order (the adjacency list of vertex i). The "**for** $j \in \mathrm{Adj}[i]$" statement instructs the program to scan all outneighbors of i in the order they are listed. Moving to the next member of the list takes one step.

The queue is a FIFO list (first-in, first-out) and permits three operations:
1. creation of an empty queue
2. ENQUEUE(Q, u) adds item u at the end of the queue Q
3. DEQUEUE(Q) outputs the first item in Q and removes it from Q, or reports that Q is empty.

For the analysis of the process, it is helpful to keep track of time. We start at time $t = 0$ and advance the time by 1 each time a vertex is discovered. Lines 04 and 11 of the pseudocode below manage the time variable; we can omit these two lines from the code with no effect on any of the other variables.

Here is the pseudocode for the algorithm.

[1] Last updated January 27, 2021.

BFS(V, E, s)

Initialization

```
01    for v ∈ V
02        status(v) := WHITE, d(v) := ∞, p(v) := NIL, Q := empty queue
03    end(for)
04        t := 1
05        status(s) := GRAY, d(s) := 0          (: s discovered :)
06        ENQUEUE(Q, s)
```

Main loop

```
07    while Q not empty do
08        v ← DEQUEUE(Q)        (: begin exploring v :)
09        for u ∈ Adj(v) do     (: explore edge v → u :)
10            if status(u) = WHITE then do
11                t := t + 1
12                status(u) := GRAY, p(u) := v, d(u) := d(v) + 1        (: u discovered :)
13                ENQUEUE(Q, u)
14            end(if)
15        end(for)              (: end exploring v :)
16        status(v) := BLACK    (: v finished :)
17    end(while)
18    return arrays p and d
```

Analysis: correctness

Our goal is to show that for all vertices $v \in V$, the value $d(v)$ returned by the algorithm is equal to $\text{dist}(s, v)$.

We associate three points in time with each vertex $u \in V$: the time $t_1(u)$ when u is discovered, the time $t_2(u)$ when the exploration of u begins and $t_3(u)$ when the exploration of u ends. If u is never discovered, we set $t_1(u) := t_2(u) := t_1(u) := \infty$. We can define these values by adding the following lines to the code. Line 02a should be inserted after line 02, etc.

```
02a       t1(i) := t2(i) := t1(i) := ∞
          ...
05a       t1(s) := t


08a       t2(v) := t
          ...
12a               t1(u) := t
          ...
15a       t3(v) := t
```

The addition of these lines has no effect on any of the original variables.

Exercise 1. Verify the following loop invariants.
(a) $(\forall v \in V)$(if $v \in Q$ then status$(v) =$ GRAY)
(b) $(\forall v \in V)(p(v) = \text{NIL} \iff \text{status}(v) = \text{WHITE} \iff d(v) = \infty)$
(c) $(\forall v \in V)$(if $p(v) \neq$ NIL then $d(v) = d(p(v)) + 1$).

Exercise 2. (a) Prove: $(\forall u \in V)(t_1(u) \leq t_2(u) \leq t_3(u))$.
(b) Find a rooted digraph (V, E, s) and a vertex $u \in V$ such that all vertices are accessible from s, the vertex u has at least one out-neighbor (i. e., the list Adj$[u]$ is not empty, yet in other words the out-degree $\deg^+(u) \geq 1$), and $t_2(u) = t_3(u)$ (the time variable is not advanced between the start and the end of the exploration of u). Make your digraph as small as possible (minimize the number of edges). Describe your digraph by an adjacency list. Attach a picture of the digraph. (You may draw it by hand.)
(c) Find a rooted digraph (V, E, s) and a vertex $u \in V$, $u \neq s$ such that all vertices are accessible from s, the vertex u has at least one sibling (another vertex w such that $p(u) = p(w)$) and $t_1(u) = t_2(u) < t_3(u)$. Make your digraph as small as possible. Describe your digraph by an adjacency list. Attach a picture of the digraph. (You may draw it by hand.)

Example of an adjacency list description:

1:2,4,3
2:1,4
3:
4:3,1

This means that set of vertices is $V = \{1, 2, 3, 4\}$ and for instance $\deg^+(4) = 2$ and the two out-neighbors of vertex 4 are 3 and 1.

Exercise 3. Prove: $(\forall v \in V)$(if $v \neq s$ then $t_2(p(v)) \leq t_1(v) \leq t_3(p(v))$).

Exercise 4. (a) Prove: $(\forall u, v \in V)$(if $u \neq v$ then $t_1(u) \neq t_1(v)$).
(b) Prove: if $t_1(u) < t_1(v)$ then $t_3(u) \leq t_2(v)$.
(c) Find a rooted digraph (V, E, s) and vertices $u, v \in V$ such that $u \neq v$, both u and v are accessible from s, and $t_2(u) = t_3(v)$. Make your digraph as small as possible. Describe your digraph by an adjacency list. Attach a picture of the digraph. (You may draw it by hand.)

Exercise 5. (a) Verify that $(\forall v \in V)$ the values $p(v)$ and $d(v)$ are updated at most once, namely, at the time v is discovered.
(b) Prove: $(\forall v \in V)$(if $v \neq s$ then $d(v) = d(p(v)) + 1$).

* * * * *

Below, $d(v)$ and $p(v)$ denote the final value of these variables (the value returned by the algorithm). Let V_0 denote the set of vertices accessible from the root.

Exercise 6. Prove: $v \in V_0$ if and only if $d(v) < \infty$.

Hint. For the "if" part, the assumption is $d(v) < \infty$ and the desired conclusion is $\text{dist}(s, v) < \infty$. Prove this by induction on $d(v)$.
For the "only if" part, the assumption is $\text{dist}(s, v) < \infty$ and the desired conclusion is $d(v) < \infty$. Prove this by induction on $\text{dist}(s, v)$.

Exercise 7. Let $v \in V_0$. Prove: $\text{dist}(s, v) \leq d(v)$.
Use Ex. 6 and part (b) of Ex. 5. State where you use each of these in your argument, and which direction of Ex. 6 you use.

Hint. Induction on $d(v)$.

Exercise 8. Let $u, v \in V_0$, $u, v \neq s$. Prove: if $t_1(u) \leq t_1(v)$ then
(a) $t_1(p(u)) \leq t_1(p(v))$ (b) $d(u) \leq d(v)$. Use part (b) of Ex. 4. State where you use it.

Now we are ready to prove the main result of the analysis.

Theorem 9. *For all $v \in V$ we have $d(v) = \text{dist}(s, v)$.*

Proof. If $v \notin V_0$ then $d(v) = \text{dist}(s, v) = \infty$ by Ex. 6.
Assume now $v \in V_0$. We already know $\text{dist}(s, v) \leq d(v)$ (Ex. 7). We need to show that $d(v) \leq \text{dist}(s, v)$. We prove this by induction on $\text{dist}(s, v)$.
Base case: $\text{dist}(s, v) = 0$. This means $v = s$ an therefore $d(v) = 0$ by line 05 of the algorithm.
Assume now $k := \text{dist}(s, v) \geq 1$.
Inductive Hypothesis: For all $w \in V$, if $\text{dist}(s, w) < k$ then $d(w) \leq \text{dist}(s, w)$ (and therefore $d(w) = \text{dist}(s, w)$).
Let $s = x_0 \to x_1 \to \cdots \to x_k = v$ be a shortest path from s to v. We need to show that $d(v) \leq k$.
DO: show that for every $i \leq k$, $\text{dist}(s, x_i) = i$.
So $\text{dist}(s, x_{k-1}) = k - 1$ and therefore, by the Inductive Hypothesis, $d(x_{k-1}) = k - 1$. If $x_{k-1} = p(v)$, it follows that $d(v) = k$ by Ex. 5.
Consider now the case that $x_{k-1} \neq p(v)$. This means that at time $t = t_2(x_{k-1})$, the status of v was not "WHITE," which means by that time, v had been discovered: $t_1(v) < t_2(x_{k-1})$. But $t_1(v) \geq t_2(p(v))$ (Ex. 3). So $t_2(p(v)) < t_2(x_{k-1})$.

Claim. $t_1(p(v)) < t_1(x_{k-1})$.
Indeed, by part (a) of Ex. 5 we have $t_1(p(v)) \neq t_1(x_{k-1})$. Now assume for a contradiction that $t_1(p(v)) > t_1(x_{k-1})$. By part (b) of Ex. 5 we infer $t_3(x_{k-1}) \leq t_2(p(v))$. By part (a) of Ex. 2 this implies $t_2(x_{k-1}) \leq t_2(p(v))$, contradicting the inequality $t_2(p(v)) < t_2(x_{k-1})$ established above. This proves the Claim.
Combining the Claim with part (b) of Ex. 8 we conclude that $d(p(v)) \leq d(x_{k-1}) = k-1$ and therefore, by part (b) of Ex. 5, $d(v) = d(p(v)) + 1 \leq k$. □

This completes the proof of correctness.
The next exercise makes an important structural observation.

Exercise 10. As before, let V_0 denote the set of vertices accessible from the root. Let $P = \{\{v, p(v)\} \mid v \in V_0, v \neq \text{root}\}$. Prove: the graph (V_0, P) is a tree.

Analysis: efficiency

Theorem 11. *The BFS algorithm runs in linear time.*

"Linear time" means O(length of the input). We use the "unit cost" model where our alphabet is the set $\{1, \ldots, n\}$ and copying such a number or incrementing such a number costs one unit of time. So the length of the input is $\Theta(n + m)$ (where n is the number of vertices and m is the number of directed edges). "Linear time" therefore means time $O(n + m)$ in this model.

Exercise 12 (Directed Handshake Theorem)**.** For a directed graph (V, E) with m directed edges,

$$\sum_{v \in V} \deg^+(v) = m.$$

Proof of Theorem 11. In this model, the cost of the initialization is $O(n)$. The **while** loop is executed at most once for each vertex v. (No vertex is DEQUEUE-ed more than once. Why?) For each v, the execution of this **while** loop (exploration of v) involves a constant number of steps, plus a constant number of steps for each edge (v, u). The total cost of the exploration of v is $O(1 + \deg^+(v))$, adding up to $O(m + n)$ by Ex. 12. □

* * * * *

Definition 13. A digraph is **strongly connected** if each vertex is accessible from each vertex.

Exercise 14. Given a digraph G in adjacency list representation, decide in linear time whether G is strongly connected.
Give a very simple solution based on BFS. Do not use DFS (Depth-first search).