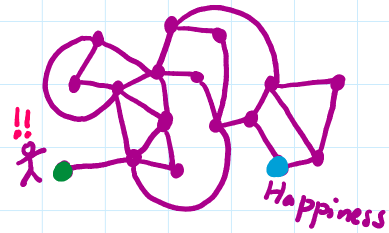
CMSC351-2023-11-03

Friday, November 3, 2023 8:32 AM

SHORTEST PATH ALGORITHM (FOR UNWEIGHTED GRAPHS)

① **Intro:** Sps we have a graph and choose two vertices s, t. How can we find the shortest path (least # edges) from s to t?



② **Idea:** We start at s. First check all vertices adj. to s.
If we find t, great. If not, check the vertices adj. to those!
If we find t, great; if not, etc.
As we go we'll:
- label the predecessor of each vertex to be the one which was one edge closer and adj to it.
- label # edges from starting vertex.

note: once we visit/tag/label a vertex, never visit it again!

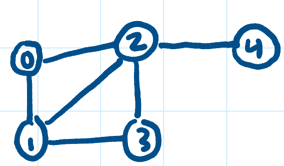
guarantee: $s, t \in$ graph
graph will be unweighted, undirected, simple, connected

③ **More Formally:** we do the following.
Initialize two lists: d = list of V distances, all $= \infty$ except $d[s]=0$
p = list of V predecessors, all = NULL
create a queue Q and push s onto it.
Then until we find t we:
$x = Q$.dequeue. if $x == t$, done
look at each y adj. to x w/ dist $= \infty$:

typically we check adj vertices in some sensible order, like increasing.

assign it a distance: $d[y] = d[x]+1$
assign it a predecessor: $p[y] = x$
add y to end of Q.

ex



Find shortest path from $s=1$ to $t=4$.

$d = [\infty, 0, \infty, \infty, \infty]$

$p = [$NULL$, N, N, N, N]$

$Q = [1]$

w/ distance of ∞

$Q = [1]$

w/ distance of ∞

then $x = Q.\text{dequeue} = 1$. Well, $1 \neq t$ so visit all adj. vertices!

We'll do 0, 2, 3. we assign $d[0] = d[2] = d[3] = d[1] + 1 = 0 + 1 = 1$

$p[0] = p[2] = p[3] = 1$

so now $d = [1, 0, 1, 1, \infty]$

$p = [1, \text{NULL}, 1, 1, \text{NULL}]$

$Q = [0, 2, 3]$

w/ dist $= \infty$

then $x = Q.\text{deque} = 0$. well $0 \neq t$ so visit all adj. vertices!

there are none! b/c it's adj to 1, 2 but $d[1], d[2] \neq \infty$!

so now d, p unchanged but $Q = [2, 3]$

w/ dist $= \infty$

then $x = Q.\text{dequeue} = 2$. well $2 \neq t$ so visit all adj. vertices

We'll do 4. We assign $d[4] = d[2] + 1 = 1 + 1 = 2$

$p[4] = 2$

so now $d = [1, 0, 1, 1, 2]$

$p = [1, \text{NULL}, 1, 1, 2]$

$Q = [3, 4]$

note: b/c we have found 4 adj to 2 and we've assigned d, p we can stop now and we're done. We don't actually need to wait to dequeue 4.

For the path, look at $P = [1, \text{NULL}, 1, 1, 2]$

we start at $t = 4$: $P[4] = 2$

$P[2] = 1$ and $P[1] = \text{NULL}$

so the path backwards is 4, 2, 1.

④ Pseudocode:

```
function shortestpath(G,s,t)
    dist = distance array of size V full of inf
    pred = predecessor array of size V full of NULL
    Q = empty queue
    dist[s] = 0
    Q.push(s)
    while Q is nonempty
        x = Q.pop
        for each infinity vertex y adjacent to x
            dist[y] = dist[x] + 1
            pred[y] = x
            if y == t
                return(pred)
            end
            Q.push(y)
        end
    end
end
```

$\Theta(V)$ each / together

$\Theta(1)$

$\Theta(V)$ times → WC

⊗

$\Theta(1)$

$\Theta(1)$

WORST-CASE:

⊗ If G is stored as an adj matrix then this loop iterates exactly V times b/c we must scan row x and check each vertex.

In which case we'd have $T(V) = \Theta(V^2)$

BETTER: store G as an adj. list.

Then we get the argument from wednesday:

- initialization: $\Theta(V)$
- while loop body w/out for loop runs $\Theta(V)$ times at $\Theta(1)$ each so $\Theta(V)$
- for loop body runs twice for each edge at $\Theta(1)$ each time, so $\Theta(2E)$

In total $\Theta(V)+\Theta(V)+\Theta(2E)$ thus

$$T(V,E) = \Theta(V+E).$$

addendum: For a simple unweighted connected graph:

- $V \leq 2E$ b/c each edge conn. 2 vert, but vert are prob. shared
- $E \leq C(V,2)$ b/c at most every pair of vertices could be conn. by an edge.

$E \leq \frac{1}{2}V(V-1)$

Thus we can also say $T(E) = O(2E+E) = O(E)$

and $T(V) = O(V + \frac{1}{2}V(V-1)) = O(V^2)$

Thus we can also say $T(E) = O(2E + E) = O(E)$

and $T(V) = O(V + \frac{1}{2}V(V-1)) = O(V^2)$