Shortest Cycle in a Graph

Calvin Fung

August 7, 2025

1 Problem

There is a bi-directional graph with n vertices, where each vertex is labeled from 0 to $n-1$. The edges in the graph are represented by a given 2D integer array edges, where $edges[i] = [u_i, v_i]$ denotes an edge between vertex u_i and vertex v_i. Every vertex pair is connected by at most one edge, and no vertex has an edge to itself.

Return the length of the shortest cycle in the graph. If no cycle exists, return -1.

A cycle is a path that starts and ends at the same node, and each edge in the path is used only once.

2 Solution

We present an $O(E + V(\text{size of the largest connected component})) = O(V(V+E))$ time solution.

2.1 analysis

Consider a simple *connected* graph $G = (V, E)$, doing a breadth first search (BFS) starting at a vertex r gives us a breadth first tree (BFT) $T = (V, E_T)$ rooted at r. Notice T is a spanning tree of G. Below we define several quantities :

- Partition $E = E_T \cup E_{T'}$ where $E_{T'} = E - E_T$
- Let $d(u), u \in V$ be the shortest distance from r to u. Notice this's also the depth/level of u in T.
- Let $D(\{u, v\}), \{u, v\} \in E$ be equal to the ordered pair $(\min\{d(u), d(v)\}, \max\{d(u), d(v)\})$. Notice for any $\{u, v\} \in E$, we have $|d(u) - d(v)| \leq 1$, it follows $D(\{u, v\})$ has either the form (x, x) or $(x-1, x)$ for some $x \geq 1$.
- For any 2 ordered pair $P_1 = (x_1, y_1), P_2 = (x_2, y_2)$, we order them in lexicographical order ie. (i) If either $x_1 < x_2$ or $(x_1 = x_2 \wedge y_1 < y_2)$, we say $P_1 < P_2$ (ii) Else if either $x_1 > x_2$ or $(x_1 = x_2 \wedge y_1 > y_2)$, we say $P_1 > P_2$ (iii) The only remaining case is $x_1 = x_2, y_1 = y_2$ which we say $P_1 = P_2$.
- Let V_x be the set of vertices within distance x from r. Furthermore, let $G[V_x], T[V_x]$ be the respective subgraph in G, T.

Lemma 1. Consider an edge $\{u, v\} \in E_{T'}$ with the smallest D value in $E_{T'}$ ie. for any $\{u', v'\} \in E_{T'}$ we have $D(\{u, v\}) \leq D(\{u', v'\})$. WLOG assume $d(u) \leq d(v)$. Then $G[V_{d(v)-1}] = T[V_{d(v)-1}]$, in particular there's no cycle in $G[V_{d(v)-1}]$.

proof. Clearly $T[V_{d(v)-1}] \subseteq G[V_{d(v)-1}]$. If there's an edge $\{u', v'\}$ in $G[V_{d(u)-1}]$ but not in $T[V_{d(u)-1}]$, it must be from $E_{T'}$. Observe that since $d(u'), d(v') \leq d(v) - 1$ we have $D(\{u', v'\}) < D(\{u, v\})$ a contradiction ie. such an edge cannot exists and $G[V_{d(v)-1}] \subseteq T[V_{d(v)-1}]$. ■

Lemma 2. Consider an edge $\{u, v\} \in E_{T'}$ with the smallest D value in $E_{T'}$ ie. for any $\{u', v'\} \in E_{T'}$ we have $D(\{u, v\}) \leq D(\{u', v'\})$. Assume $d(u) = d(v) - 1$. Then there's no cycle involving r having size $< 2d(v)$.

proof. We prove by contradiction

Case 1 : There's an even cycle C of size $2s < 2d(v)$ involving r. As there's a path from r to any vertex of C of length $\leq s$, we have for all $c \in C, d(c) \leq s$. It follows C is a subgraph of $G[V_s]$ which in turn is a subgraph of $G[V_{d(v)-1}]$ (notice $s < d(v)$). However, by lemma 1 there's no cycle in $G[V_{d(v)-1}]$ thus a contradiction.

Case 2 : There's an odd cycle C of size $2s + 1 < 2d(v)$ involving r. The rest of the argument is exactly same as case 1. ■

Lemma 3. Consider an edge $\{u, v\} \in E_{T'}$ with the smallest D value in $E_{T'}$ ie. for any $\{u', v'\} \in E_{T'}$ we have $D(\{u, v\}) \leq D(\{u', v'\})$. Assume $d(u) = d(v)$. Then there's no cycle involving r having size $< 2d(v) + 1$.

proof. We prove by contradiction

Case 1 : There's an even cycle C of size $2s < 2d(v) + 1$ involving r. Notice C is a subgraph of $G[V_{d(v)}]$ because $s \leq d(v)$ and there's a path from r to any vertex of C of length $\leq s$. If C is a subgraph of $G[V_{d(v)-1}]$, then we immediately have a contradiction by lemma 1. Otherwise, there's at least a node of C in $V_{d(v)} - V_{d(v)-1}$. If there're more than 1 node of C in $V_{d(v)} - V_{d(v)-1}$, notice C will have length at least $2d(v) + 1$ (to see this imagine walking on C starting at r to the left until we hit a vertex in $V_{d(v)} - V_{d(v)-1}$, this walk will have length $\geq d(v)$. Now do the same for the right, that walk will also have length $\geq d(v)$. The two walks must be edge disjoint so there must be at least an edge joining them. The length of C is the sum of those two walks and the edges joint them) . Else there's exactly 1 node z of C in $V_{d(v)} - V_{d(v)-1}$, recall as we mentioned at the beginning, any edge incident to z in the subgraph $G[V_{d(v)}]$ must have D value either of the form $(d(z), d(z))$ or $(d(z) - 1, d(z))$, since there're 2 edges incident to z in C and C is a subgraph of $G[V_{d(v)}]$, one of the edge must be from $E_{T'}$ and must have the form $(d(z) - 1, d(z))$ as there's exactly 1 node of C in $V_{d(v)} - V_{d(v)-1}$. However, $(d(z) - 1, d(z)) = (d(v) - 1, d(v)) < (d(v), d(v)) = D(\{u, v\})$ a contradiction.

Case 2 : There's an odd cycle C of size $2s + 1 < 2d(v) + 1$ involving r. The rest of the argument is exactly same as the proof for lemma 2. ■

Now suppose a shortest cycle in G is C^* which contains r. Also suppose w.r.t the BFT rooted at r, $\{u, v\} \in E_{T'}$ is an edge with the smallest D value in $E_{T'}$ (WLOG assume $d(u) \leq d(v)$). If $d(u) = d(v) - 1$, then by lemma 2 we know $2d(v) \leq |C^*|$. Notice there're two different paths from r to v each of length $d(v)$ (consider $T \cup \{u, v\}$), those two paths must be edge disjoint otherwise there's a cycle shorter than C^*, thus those two paths form a cycle C' which contains r of length $2d(v)$, which means it must be a shortest cycle. The other case is similar, if $d(u) = d(v)$, then by lemma 3 we know $2d(v) + 1 \leq |C^*|$. Notice there're two different paths from r to v, the one going through u has length $d(v) + 1$ and the other has length $d(v)$, those two paths must be edge disjoint otherwise there's a cycle shorter than C^*, thus those two paths form a cycle C' which contains r of length $2d(v) + 1$, which means it must be a shortest cycle.

To find an edge $\{u, v\} \in E_{T'}$ with the smallest D value in $E_{T'}$, we do a BFS starting at r, let S_l be the set of edges in $E_{T'}$ encountered when exploring vertices in level l, notice for any $e \in S_l$, $D(e)$ could only be $(l - 1, l)$ or (l, l) or $(l, l + 1)$. It follows that for $e_i \in S_i, e_j \in S_j$ where $i < j$, we have $D(e_i) \leq D(e_j)$. As $\cup_l S_l = E_{T'}$, an edge with the smallest D value in $E_{T'}$ can be found in S_j where $S_j \neq \emptyset$ and $\forall i < j, S_i = \emptyset$.

As we don't actually know which vertex r^* a shortest cycle in G contains, for a vertex r where an edge with the smallest D value in $E_{T'}$ (w.r.t the BFT of r) is $\{u, v\}$ (WLOG assume $d(u) \leq d(v)$), let $f(r) = 2d(v)$ if $d(u) = d(v) - 1$ and $f(r) = 2d(v) + 1$ otherwise. As argued above we know $f(r^*)$ is the length of the shortest cycle. However, can we simply take a minimum over r ie. is $f(r^*) = \min_{r \in V} f(r)$? Yes, it's not hard to see that either $f(r)$ corresponds to a cycle involving r or corresponds to 2 different paths that are not edge disjoint starting at r and ending at the same vertex, which implies existence of a cycle of length $< f(r)$.

2.2 algorithm

We present an $O(E + V(\text{size of the largest connected component})) = O(V(V + E))$ time solution.

```
f(E,n){
        G = array of size n
        for((u,v) in E){
                G[u].push(v)
                G[v].push(u)
        }

        d = array of size n initialized to inf

        f = inf

        for(r = 0 to n-1){
                //stores all vertices in the component of r, used to reset d fast
                C = empty list
                C.push(r)

                //the smallest l such that S_l is non-empty
                l = inf

                //smallest D value for an edge in E_T' is (l,l) or (l,l+1)
                b = 0

                d[r] = 0
                Q = empty queue
                Q.push(r)
                while (!Q.empty()) {
                        u = Q.pop()

                        if(d[u] > l)    break;

                        for(v in G[u]){
                                if(d[v] == inf){
                                        d[v] = d[u] + 1
                                        Q.push(v)
                                        C.push(v)
                                }else if(d[v] <= d[u]){
                                        l = d[u]
                                        b = d[v] - d[u] //either 1 or 0
                                        //we're sure that (u,v) is the best edge
                                        if(b == 0)      break;
                                }
                        }
                }

                if(b == 0){
                        f = min(f, 2*l + 1)
                }else{
                        f = min(f, 2*(l+1))
                }

                //reset d fast
                while (!C.empty()){
                        u = C.pop()
                        d[u] = inf
                }
        }

        return f==inf ? -1 : f;
}
```