breadth-first search

properties

like DFS, BFS is another general technique for graph traversals, where a BFS traversal of a graph G:

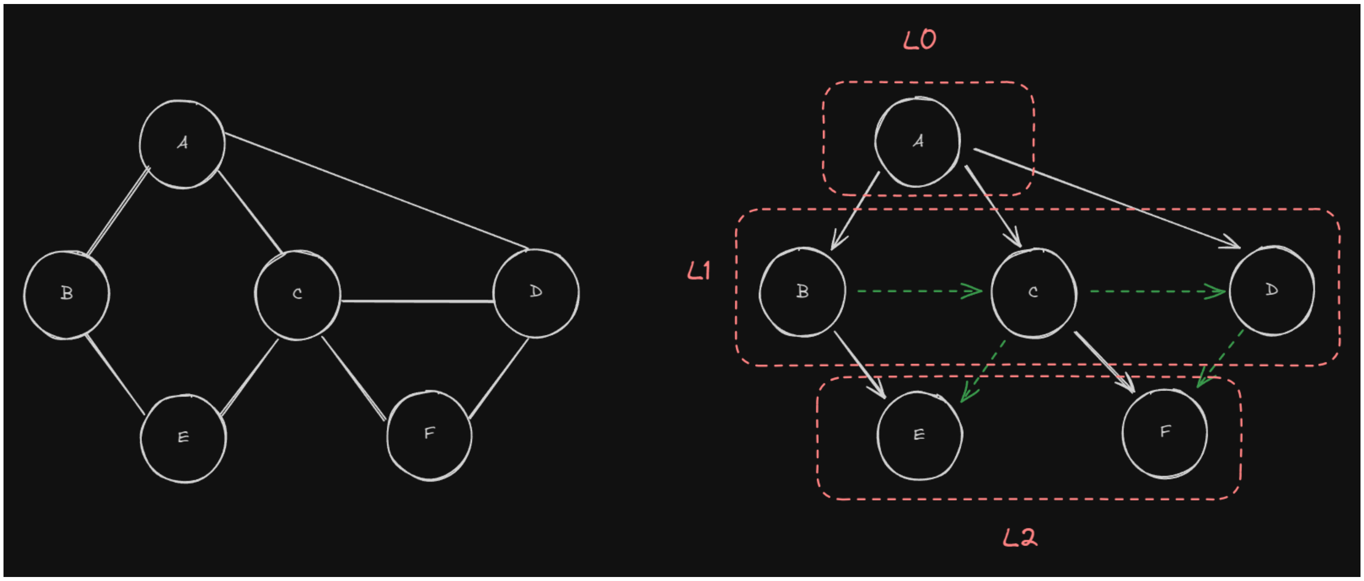
- visits all the vertices and edges of G
- determines whether G is connected or not
- computes the connected components of G
- computes a spanning forest of G

however, the applications for problem solving will be different. BFS is especially good at optimal paths, such as:

- finding a path with minimum edges between two vertices
- finding a simple cycle

runtime: $O(V + E)$

- `BFS(G, v)` will visit all vertices and edges in the connected component of `v`.
- the discovered edges by the above function call will form a spanning tree T_s of the component G_s
- for each vertex v in L_i:
  - the path of T_s from **s** to **v** has i edges
  - every path from **s** to **v** has ***at least*** i edges.



implementation

whereas DFS uses a stack, BFS makes use of a queue, and is mostly done iteratively.

```
procedure BFS(G,v) {
    Queue q;
    Set visited;
    Q.enqueue(v);
    visited.add(v);
    while ( Q is not empty) {
        w = Q.dequeue()
        for (vertex s: G.adjacentVertices(w)) {
            if (s not in visited) {
                Q.enqueue(s)
                visited.add(s)
            }
        }
    }
}
```

applications

- copying garbage collection
- shortest path (unweighted)

- Ford-Fulkerson method for computing the maximum flow in a flow network
- computing the connected components of a graph
- get spanning forest of a graph
- testing bipartiteness of a graph

analysis

each vertex is inserted once to a sequence L_i

- method `adjacentVertices()` is called once for each vetex
- BFS runs in $O(V+E)$ time, provided the graph is represented with an **adjacency list**