San Skulrattanakulchai
Gustavus Adolphus College

Breadth-first Search

CLRS: Ch 22.2

Breadth-first search is another method for exploring graphs. It is good for computing distances from a specified vertex. It works on both directed and undirected graphs.
Throughout this handout $G = (V, E)$ can either be a directed or undirected graph.
Fix a vertex s. For $v \in V$, define the *distance* from s to v, written $\delta(s, v)$, to be the length of a shortest path from s to v, if one exists. (A shortest path is a path with the fewest number of edges.) For a vertex v not reachable from s, define $\delta(s, v) = \infty$.

[Example Distances Here]

Problem: Given a source vertex s in G, find, for all $v \in V$, a shortest s, v-path.
Solution: We construct inductively a shortest-path tree rooted at s. A shortest-path tree is an out-tree (out-arborescense) with the property that for any vertex v, the unique path in the tree from s to v is an s, v-shortest path in G.
Let $V_0 = \{s\}$. Let $V_1 = \{\, v \in V \setminus V_0 : sv \in E \,\}$. Let $V_2 = \{\, v \in V \setminus (V_0 \cup V_1) : u \in V_1, uv \in E \,\}$. In general, given that $V_0, \ldots, V_{i-1}$ have been defined, we define V_i to be the set of all vertices $v \in V \setminus (V_0 \cup \cdots \cup V_{i-1})$ such that there is some vertex $u \in V_{i-1}$ with $uv \in E$. We stop when $V_i = \emptyset$.
The shortest-path tree T consists of all the vertices in $\bigcup V_i$. For any vertex $v \neq s$ in V_i, we choose exactly one edge uv to belong to the tree, where $u \in V_{i-1}$ and $uv \in E$. □

[Example Shortest Path Tree Here]

Breadth-first search is an implementation of the above algorithm.

BFS idea: Repeatedly scan an edge that is incident to the least recently discovered vertex with unscanned edges.

We use an array $d[\cdot]$ of size n to keep the distances, i.e., when the algorithm finishes, $d[v] = \delta(s, v)$ for all $v \in V$. We also use an array $p[\cdot]$ to keep track of the parent pointers in the bfs tree, i.e., by the end of the algorithm $p[v]$ is the parent of v in the bfs tree, for all $v \in V \setminus s$. The tree root s is the parent of itself.

```
BFS(v) {
    for each vertex v ∈ V do
        d[v] ← ∞
    d[s] ← 0
    p[s] ← s
    Q ← {s} // let Q be a queue having vertex s as its only element
    while Q is not empty do {
        v ← dequeue(Q)
        for each edge vw do
            if d[w] = ∞ {
                d[w] ← d[v] + 1
                p[w] ← v
                enqueue(w, Q)
            }
    }
}
```

[Example Execution of BFS Here]

Notes

1. A BFS tree is a shortest path tree, but not every shortest path tree is a BFS tree.
2. BFS is a natural algorithm for solving puzzles like Sam Lloyd's 15-puzzle, River Crossing, etc.
3. For each i, let's call the vertices in V_i vertices at level i. The non-tree edges of the graph cannot skip forward any levels.