BFS

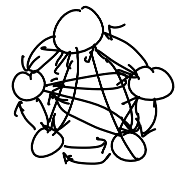
```
Func BFS(Graph g, V src, V dest):
    Queue<V> q ← new LQ
    q.enqueue(src)
    Dictionary<V, V> prev ← new HT
    prev.insert(src, src)
    While q is not empty:
        V current ← q.dequeue()
        If current == dest:
            List<V> path
            path.insertAtHead(current)
            While current ≠ src:
                current ← prev.get(current)
                path.insertAtHead(current)
            Return path
        For each neighbor of current:
            If not prev.containsKey(neighbor):
                prev.insert(neighbor, current)
                q.enqueue(neighbor)
    EndWhile
```

$O(|V|)$

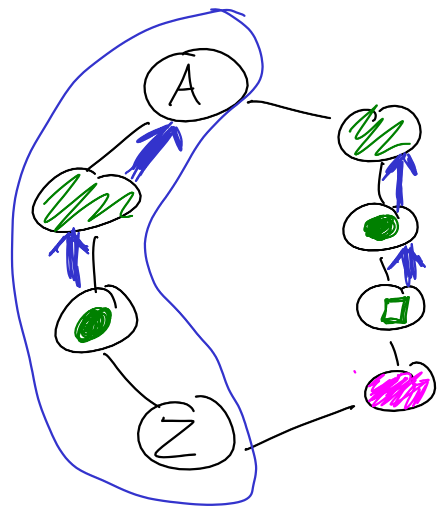
$O(|V|)$

$O(|E|)$

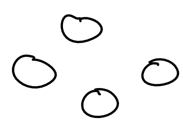
$O(|V|^2)$



Clique



BFS always gives shortest path

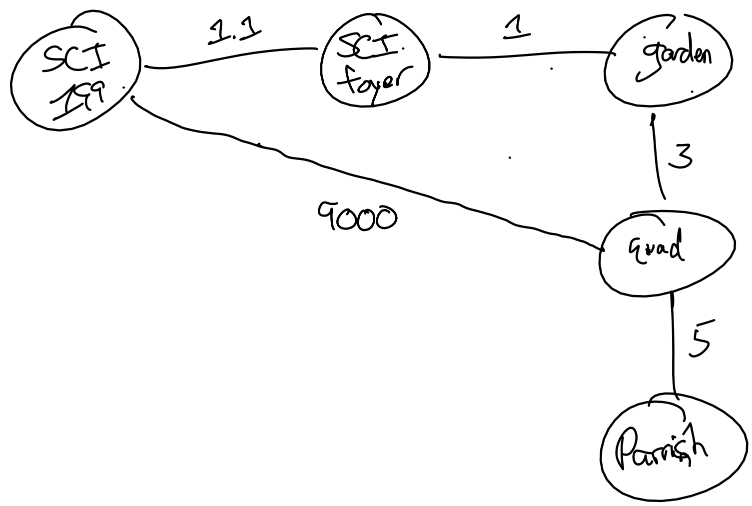


BFSAll

Return dictionary V ⟺ cost

```
Function BFSAll( Graph g,  V src)
    Queue<V> q ← new LQ
    q.enqueue(src)
    Dictionary <V, int> cost ← new HT
    cost.insert(src, 0)
    While q is not empty:
        V current ← q.dequeue()
        For each neighbor of current:
            If not cost.containsKey(neighbor):
                q.enqueue(neighbor)
                newcost ← cost.get(current) + 1
                cost.insert(neighbor, new cost)
    Return cost
```

Length of path vs. cost of path



```
Function Dijkstra's( Graph g, V src):
    PQ<int, V> pq ← new MinHeap
    pq.insert(0, src)
    Dictionary<V, int> cost ← new HT
    cost.insert(src, 0)
    While pq is not empty:
        current ← pq.remove()
        For each outgoing edge e of current:
            neighbor ← e.dest
            newCost ← cost.get(current) + e.weight
            If not cost.containsKey(neighbor):
                cost.insert(neighbor, newCost)
                pq.insert(newCost, neighbor)
            Else If cost.get(neighbor) > newCost:
                cost.update(neighbor, newCost)
                pq.insert(newCost, neighbor)
    Return cost
```

Only usable w/ non-negative weight

today

$O(|E| \log |E|)$

$O(|V|^2)$

different versions