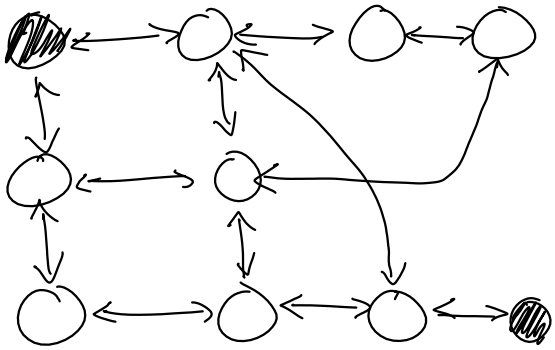
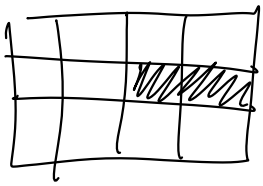
Graph Algorithms

Graph: <V, E>

Edge: <V, V, W, L>
src dst weight label

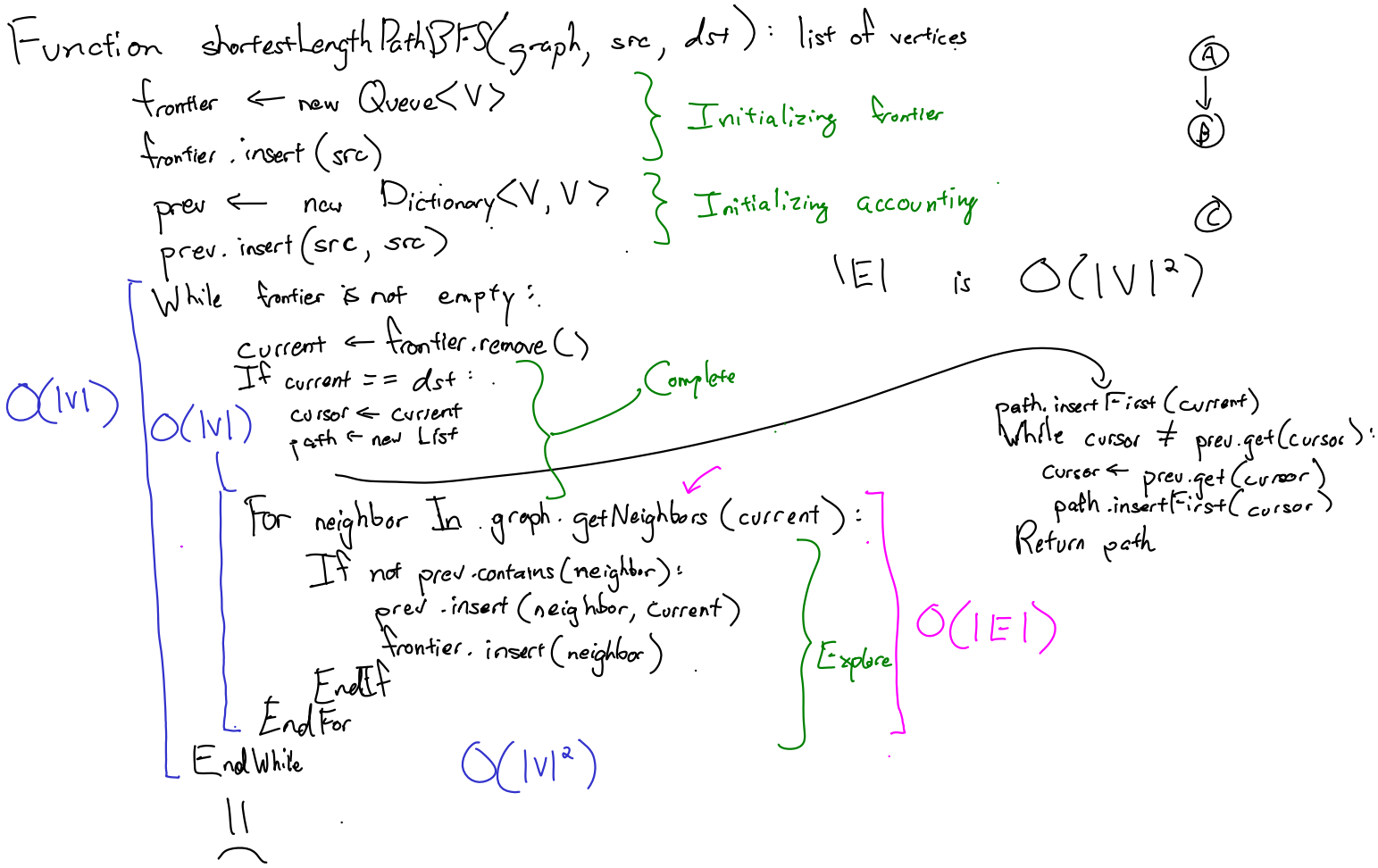




4 Graph Algorithms

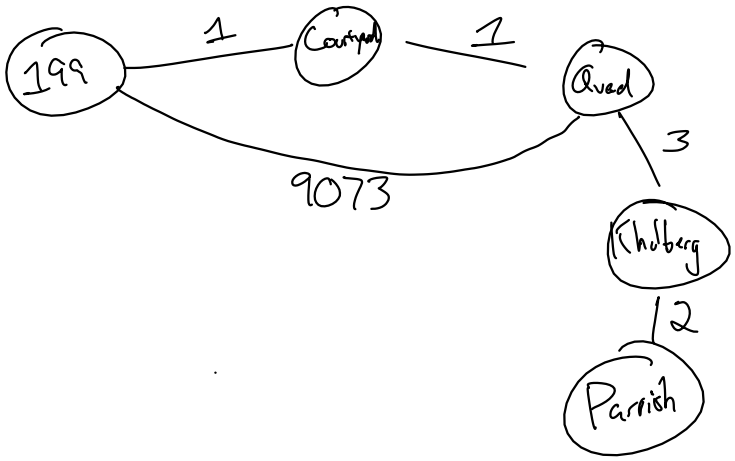
- reachable DFS (last week)
- shortestLengthPath BFS
- allLengthsPaths BFS
- singleSourceShortestPath (Dijkstra's)

For Lab 9


Function shortestLengthPathBFS(graph, src, dst): list of vertices
frontier ← new Queue<V>
frontier.insert(src)
Initializing frontier
prev ← new Dictionary<V, V>
prev.insert(src, src)
Initializing accounting
A
B
C
|E| is O(|V|²)
While frontier is not empty:
current ← frontier.remove()
If current == dst:
cursor ← current
path ← new List
Complete
O(|V|)
O(|V|)
For neighbor In graph.getNeighbors(current):
If not prev.contains(neighbor):
prev.insert(neighbor, current)
frontier.insert(neighbor)
EndIf
EndFor
Explore
O(|E|)
EndWhile
O(|V|²)
path.insertFirst(current)
While cursor ≠ prev.get(cursor):
cursor ← prev.get(cursor)
path.insertFirst(cursor)
Return path

```
Function allLengthsBFS(graph, src) : Dictionary<V, int>
    frontier ← new Queue<V>
    frontier.insert(src)
    length ← new Dictionary<V, int>          # of edges
    length.insert(src, 0)
    While frontier is not empty:
        current ← frontier.remove()
        For each neighbor of current:
            If not length.contains(neighbor):
                length.insert(neighbor, length.get(current) + 1)
                frontier.insert(neighbor)
    EndWhile
    Return length
EndFunction
```

O(|E|)




```
Function SSSP(graph, src) : Dictionary<V, int>        <- total cost
    frontier <- new PQ<int, V>          <- low priority is next
    frontier.insert(0, src)
    cost <- new Dictionary<V, int>
    cost.insert(src, 0)
    While frontier is not empty:
        current <- frontier.remove()          <- by the time a vertex is dequeued, we have found its best path
        For each edge leaving current:
            total <- cost.get(current) + edge.weight
            If (not cost.contains(neighbor)):
                cost.insert(neighbor, total)
                frontier.insert(total, neighbor)
            Else If cost.get(neighbor) > total:
                cost.update(neighbor, total)
                frontier.insert(total, neighbor)        <- update
            End If
        End For
    End While
    Return cost
End Function
```