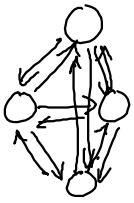
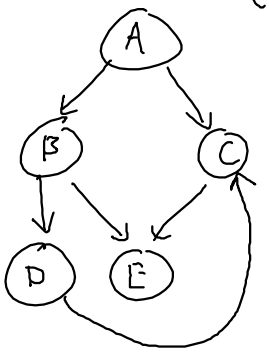
Graph algorithms



Clique

Function shortestLengthPathBFS(graph, src, dst): $O(V^2)$ $O(E)$

```
    frontier ← new Queue
    frontier.insert(src)
    previous ← new Dictionary
    previous.insert(src, src)
    While frontier is not empty:                          O(V)
        current ← frontier.remove()
        If current == dst:
            Return buildpath(previous, src, dst)
        Else:
            For each neighbor of current:                 O(V)
                If neighbor is not a key in previous:     O(E)
                    previous.insert(neighbor, current)
                    frontier.insert(neighbor)
                EndIf
            EndFor
        EndIf
    EndWhile
EndFunction
```



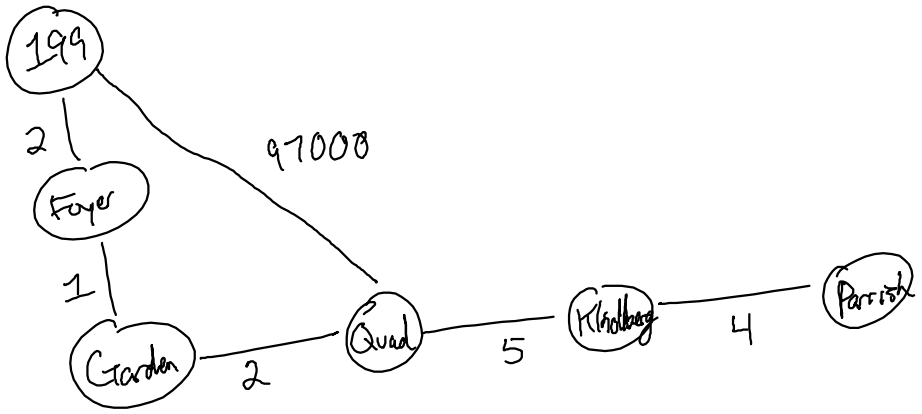
frontier : [~~A~~, ~~B~~, ~~C~~, ~~D~~, ~~E~~]

previous : A ↦ A
B ↦ A
C ↦ A
D ↦ B
E ↦ B

current : E

```
Function everyShortestLengthPathBFS( graph, src ):
    frontier ← new Queue
    frontier.insert(src)
    previous ← new Dictionary
    previous.insert (src, src)
    While frontier is not empty:
        current ← frontier.remove()
        For each neighbor of current:
            If neighbor is not a key in previous:
                previous.insert (neighbor, current)
                frontier.insert (neighbor)
            EndIf
        EndFor
    EndWhile
    Return previous
EndFunction
```

O(E)



Function sssp(g, src):

"single source shortest path" — Dijkstra's algorithm

(shortest = cheapest)

```
    frontier ← new MinHeap
    frontier.insert(0, src)
    cost ← new Dictionary
    cost.insert(src, 0)
    While frontier is not empty:
        currentCost ← frontier.peekPriority()
        current ← frontier.remove()
        For each neighbor of current:
            If neighbor is not a key in cost:
                cost.insert(neighbor, currentCost + weight)
                frontier.insert(currentCost+weight, neighbor)
            Else If cost.get(neighbor) > currentCost + weight:
                cost.update(neighbor, currentCost+weight)
                frontier.insert(currentCost+weight, neighbor)
            End If
        End For
    End While
    Return cost
End Function
```

Initialization

frontier loop

neighbor loop

condition