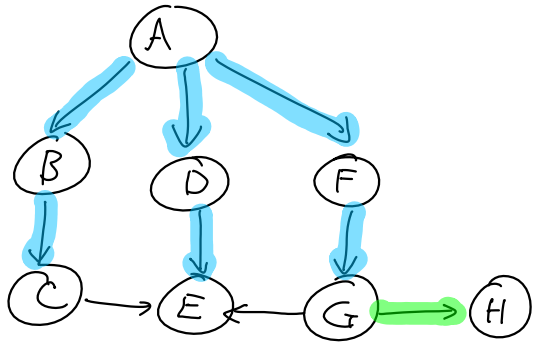
```
Function shortestPathBFS(g, src, dst):
    exploration ← new Queue
    exploration.enqueue(src)
    previous ← new Dictionary
    previous.insert(src, src)
    While exploration is not empty:
        current ← exploration.dequeue()
        If current == dst:
            path ← new List
            While current ≠ src:
                path.insertFirst(current)
                current ← previous.get(current)
            EndWhile
            path.insertFirst(src)
            Return path
        EndIf
        For neighbor In g.getNeighbors(current):
            If neighbor is not a key in previous:
                previous.insert(neighbor, current)
                exploration.enqueue(neighbor)
            EndIf
        EndFor
    EndWhile
EndFunction
```



shortestPathBFS(, A, E)

exploration = [~~A~~, ~~B~~, ~~D~~, ~~F~~, ~~C~~, ~~E~~, G

previous = { A ↦ A, F ↦ A, G ↦ F
B ↦ A, C ↦ B
D ↦ A, E ↦ D

current = A

neighbor =

path = [A, D, E]

Dijkstra

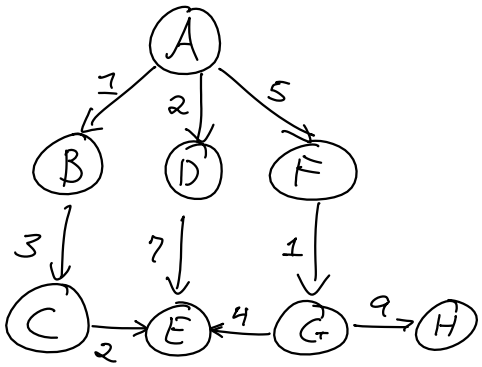
```
Function sssp(g, src):
    exploration ← new MinHeap()
    exploration.insert(0, src)
    costs ← new Dictionary()
    costs.insert(src, 0)
    While exploration is not empty:
        current ← exploration.remove()
        currentCost ← costs.get(current)
        For each edge In g.getOutgoingEdges(current)
            neighbor ← edge.destination
            newCost ← currentCost + edge.weight
            If neighbor not a key in costs:
                costs.insert(neighbor, newCost)
                exploration.insert(newCost, neighbor)
            Else If costs.get(neighbor) > newCost:
                costs.update(neighbor, newCost)
                exploration.insert(newCost, neighbor)
            EndIf
        End For
    End While
    Return costs
End Function
```

exploration

accounting

dequeue

consider each neighbor



exploration = [0 9 15 / A B D C F E G E H]

costs = { A ↦ 0, B ↦ 1, D ↦ 2, F ↦ 5, C ↦ 4, E ↦ 6, G ↦ 6, H ↦ 15 }

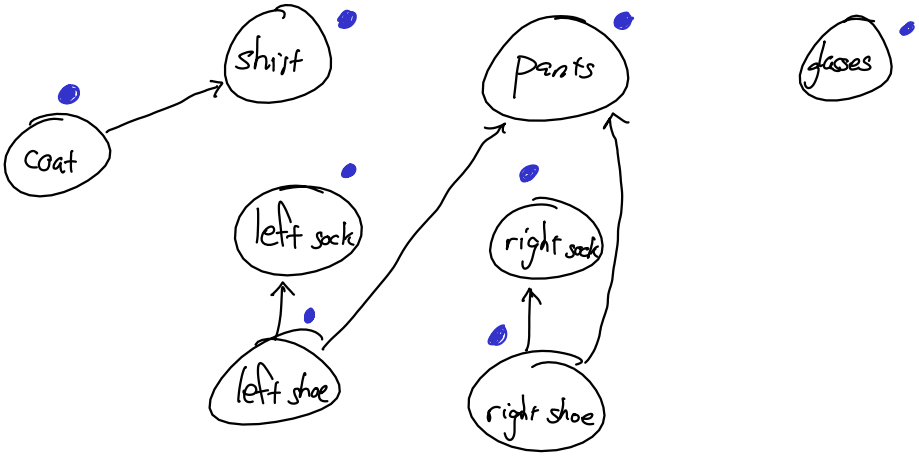
current =

edge =

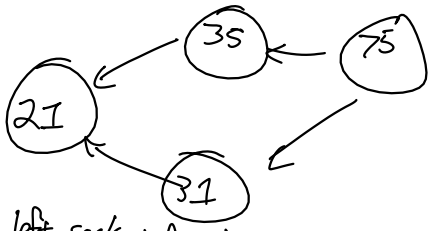
currentCost =

neighbor =

newCost =



- not visited
- in progress
- completed



right sock, pants, right shoe, shirt, coat, glasses, left sock, left shoe
shirt, pants, left sock, right sock, left shoe, right shoe; coat, glasses

Topological Sort

```
Function toposort(g):
    answer ← new List
    While any vertex is not visited
        visit(g, some unvisited vertex)
```

```
Function visit(g, vertex):
    If vertex already visited:
        Return
    If vertex in progress:
        ||
    Mark vertex in progress
    For each neighbor:
        visit(g, neighbor)
    Mark vertex complete
    add vertex To answer
```