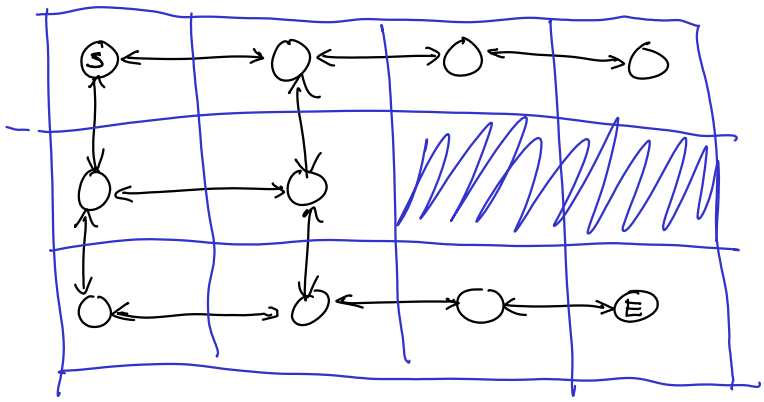
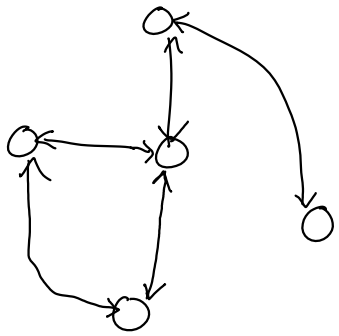
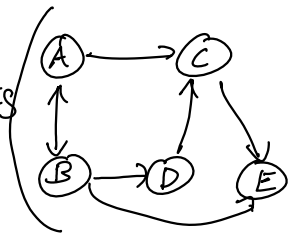




Depth-First Search

```
Function reachableDFS(graph, src, dest) : bool
    Set frontier to new Stack
    Set visited to new Set
    frontier.push(src)
    visited.add(src)
    While frontier is not empty:
        Set current to frontier.pop()
        If current == dest:
            → Return true
        EndIf
        For each neighbor of current:
            If neighbor not in visited:
                frontier.push(neighbor)
                visited.add(neighbor)
            End If
        EndFor
    EndWhile
    Return false
EndFunction
```

Set<T>
Dictionary<T, int>

reachableDFS(

, A, E)

frontier B

current E
neighbor E

visited {A, B, C, E}

Breadth-First Search

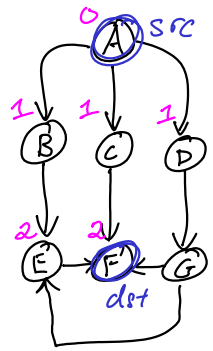
```
Function shortestLengthPath(graph, src, dst) : path        # list of vertices
    Set frontier to new Queue
    Set previous to new Dictionary from V (current) to V (prev)
    frontier.enqueue(src)
    previous.insert(src, src)
    While frontier is not empty
        Set current to frontier.dequeue()
        If current == dst:
            Reconstruct path by following previous pointers
            Return reconstructed path
        EndIf
        For each neighbor of current:
            If neighbor is not a key in previous:
                previous.insert(neighbor, current)
                frontier.enqueue(neighbor)
            EndIf
        EndFor
    EndWhile
    Return no path
EndFunction
```

Initializing search frontier and accounting structures

result

exploration

result



frontier D, E, F

previous A↦A, E↦B, B↦A, F↦C, C↦A, D↦A

current C

neighbor F

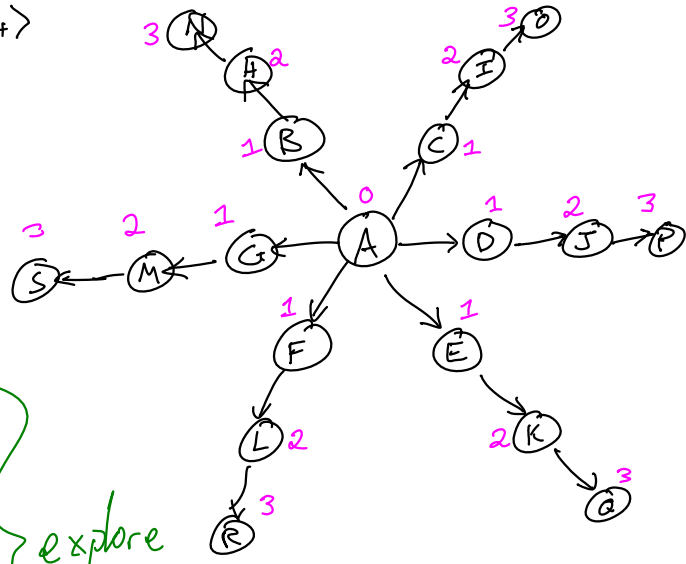
All-Lengths BFS

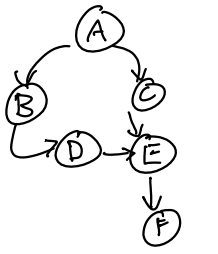
length of shortest path from src

```
Function allLengthsBFS(graph, src) : Dictionary<V, int>
    Set frontier to new Queue
    Set lengths to new Dictionary<V, int>
    frontier.enqueue(src)
    lengths.insert(src, 0)
  → While frontier is not empty:
        Set current to frontier.dequeue()
        For each neighbor of current:
            Set newLength to lengths.get(current) + 1
            If neighbor is not a key in lengths:
                frontier.enqueue(neighbor)
                lengths.insert(neighbor, newLength)
            EndIf
        EndFor
    EndWhile
    Return lengths  results
EndFunction
```

init

explore





frontier:

lengths: {A↦0, D↦2, B↦1, E↦2, C↦1, F↦3}

current: newLength:

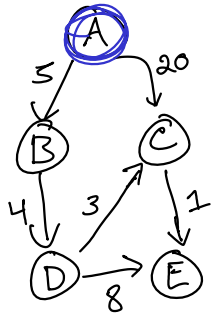
neighbor:

Dijkstra's Algorithm (SSSP)

```
Function dijkstra's allLengthsBFS(graph, src) : Dictionary<V, int>
    Set frontier to new MinPriorityQueue<int, V>
    Set lengths to new Dictionary<V, int>
    frontier.enqueue(0, src)
    lengths.insert(src, 0)
    While frontier is not empty:
        Set current to frontier.dequeue()
        For each outgoingEdge of current:
            Set neighbor to outgoingEdge.destination
            Set stepCost to outgoingEdge
            Set newLength to lengths.get(current) + stepCost
            If neighbor is not a key in lengths:
                frontier.enqueue(neighbor)
                lengths.insert(neighbor, newLength)
            Else If lengths.get(neighbor) > newLength:
                frontier.enqueue(newLength, neighbor)
                lengths.update(neighbor, newLength)
            EndIf
        EndFor
    EndWhile
    Return lengths
EndFunction
```

dequeue only these vertices w/ best cost

require all edge weights to be non-negative



frontier:

costs: A ↦ 0, B ↦ 5, C ↦ 12, D ↦ 9, E ↦ 13

current: C

outgoingEdge: C → E

neighbor: E newCost: 13

stepCost: 1