Graph: V of vertices

E of edges — source vertex
target vertex
label
weight

Implementing requires interface

Graph is ADT

Operations

insert Vertex
insert Edge
remove Edge
updateEdge
getAllVertices
getAllEdges

removeVertex
getNeighbors
getOutgoingEdges
getIncomingEdges

Algorithms

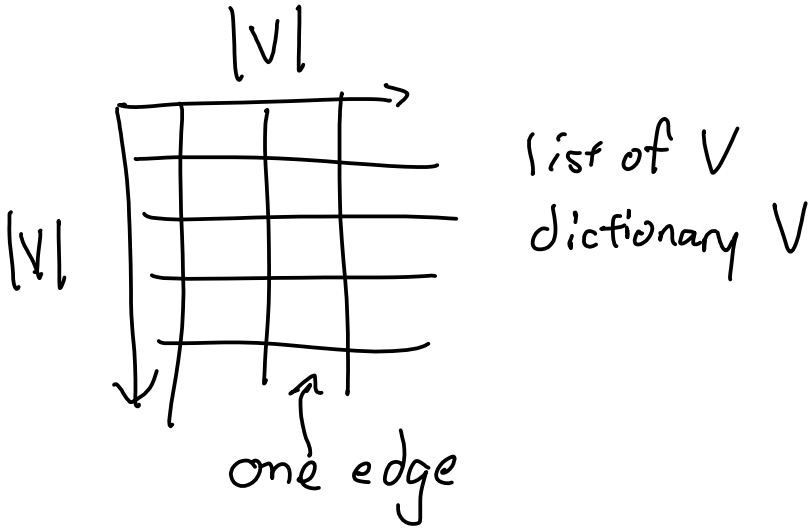
find path pt A to pt B
isConnected?

Implementations:

2 LL — vertices
edges

ListGraph

containsVertex O(|V|)



AdjacencyMatrixGraph

Dictionary as a set: V List of edges

Dictionary<V, Dictionary<V, EdgeData>>

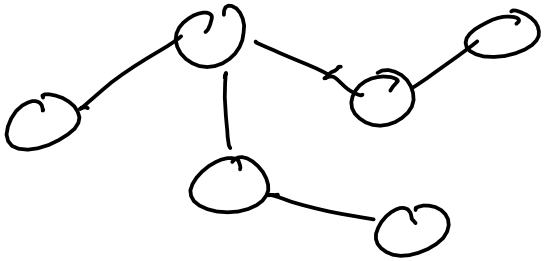
source target w/l

AdjacencyListGraph

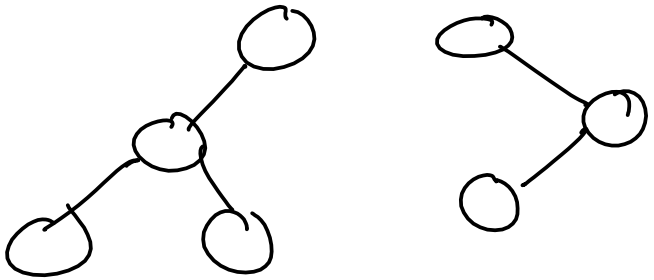
AdjacencyDictionaryGraph

Is Connected?

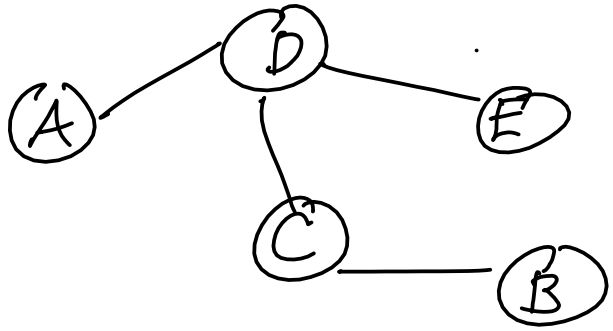
connected



disconnected



Can I get from A to B?



```
Function DFS(Graph g, V start, V end):                    return bool
    If start = end:                                       } O(1)
        Return true
    Stack<V> s ← new LS                                   } O(1)
    s.push(start)
    Dictionary<V, bool> visited ← new HT                  } O(1)
    visited.insert(start, tru
    while s is not empty:                                 } O(|V|)
        V curr ← s.pop()                                  1 pop
        if curr = end:
            Return true
        For each neighbor n of current:                   } O(|V|)
            If n not in visited:
                s.push(n)
                visited.insert(n, true)
    Return false
```

clique

O(|E|)

total # of times this loop runs

total O(|V|) pushes

O(|E|)

```
Function BFS(Graph g, V start, V end):                    returns path
                                                          list of V
    If start = end:
        Return [start]
    Queue<V> q ← new LQ
    q.enqueue(start)
    Dictionary<V, V> prev ← new HT
    prev.insert(start, start)
    while q is not empty:
        V current ← q.dequeue()
        if current = end:
            [redacted]
        for each neighbor n of current:
            if n not a key in prev:
                prev.insert(n, current)
                q.enqueue(n)
    throw exception !!

    List<V> path ← new LL
    while (current ≠ start):
        path.insertAtFront(current)
        current ← prev.get(current)
    path.insertAtFront(start)
    return path
```