Unit III: Graph Algorithms

CISC 380 Algorithms

Dr. Sarah Miracle

Graph Representation Questions

Determine the running time of each of the tasks using (1) an adjacency list and (2) an adjacency matrix.

- Is a particular edge (i, j) present in the graph?
- What is the degree of a particular vertex i?
- What is the maximum degree vertices in the graph?
- How much space does each of the two formats require?

When analyzing graph algorithms we will assume the graph has n vertices and m edges.

Binary Tree Class in Java

```
 private static class BinaryTreeNode<E> {
     private E data;
     private BinaryTreeNode<E> parent = null;
     private BinaryTreeNode<E> left = null;
     private BinaryTreeNode<E> right = null;
     private BinaryTreeNode(E dataItem){
          data = dataItem;
     }
}

public class BinaryTree<E>{
     private BinaryTreeNode<E> root = null;
     private int size;
     . . .
     }
```

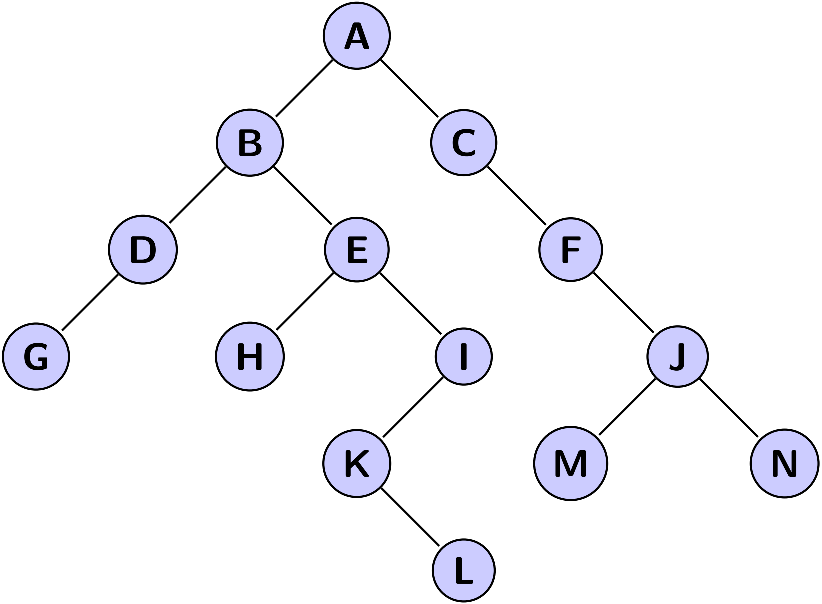
Breadth First Search in Trees

```
public class BFIterator{
   private Queue<BinaryTreeNode<E>> q
                     = new LinkedList<BinaryTreeNode<E>>();
   public BFIterator (BinaryTreeNode<E> root){
      if(root!= null){ q.offer(root);}
   }
   public boolean hasMoreElements(){
      if(!hasMoreElements()){
         throw new NoSuchElementException
                     ("tree ran out of elements");}
      BinaryTreeNode<E> node = q.remove();
      if (node.right!= null) {q.offer(node.right);}
      if (node.left != null) {q.offer(node.left);}
      return node.data;
   }
}
```

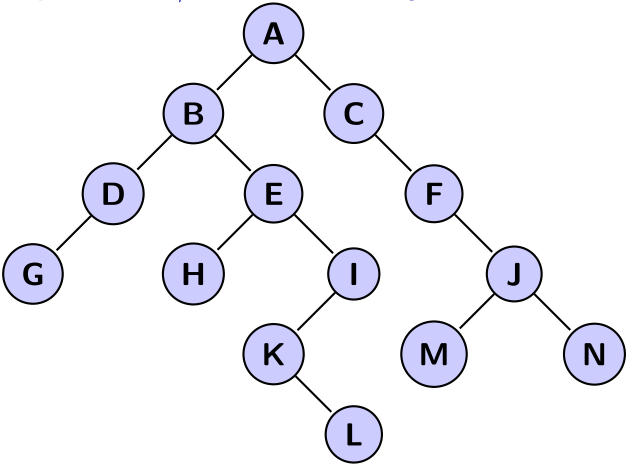
Depth First Search in Trees

```
public class DFIterator{
   private Stack<BinaryTreeNode<E>> s
                 = new Stack<BinaryTreeNode<E>>();
   public DepthFirstIterator (BinaryTreeNode<E> root){
      if(root!= null){ s.push(root);}
   }
   public boolean hasMoreElements(){
      if(!hasMoreElements()){
         throw new NoSuchElementException
                   ("tree ran out of elements");}
      BinaryTreeNode<E> node = s.pop();
      if (node.right!= null) {s.push(node.right);}
      if (node.left != null) {s.push(node.left);}
      return node.data;
   }
}
```

Example: DFS/BFS on Binary Trees



Example: DFS/BFS on Binary Trees



- Depth First (pre-order) - *ABDGEHIKLCFJMN*
- Depth First (in-order) - *GDBHEKLIACFMJN*
- Depth First (post-order) - *GDHLKIEBMNJFCA*
- Breadth First - *ABCDEFGHIJKMNL*

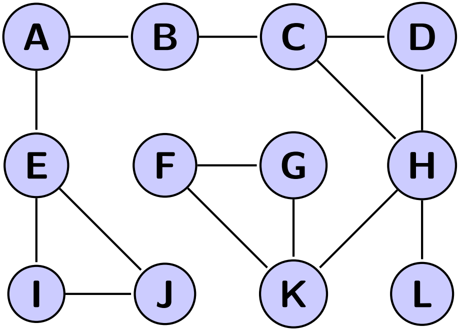
Depth First Search in Graphs

Running DFS on a graph $G = (V, E)$ starting from a vertex v

```
function DFS(G)
    for all w ∈ V do
        visited(w) = False
    Explore(v)            ▷ pick an arbitrary vertex v to start from

function Explore(w)
    visited(w) = True
    for all (w, z) ∈ E do
        if not visited(z) then
            Explore(z)
```

Example: DFS on General Graph



function DFS(G)
  for all $w \in V$ **do**
    visited(w) $=$ False
  EXPLORE(v)

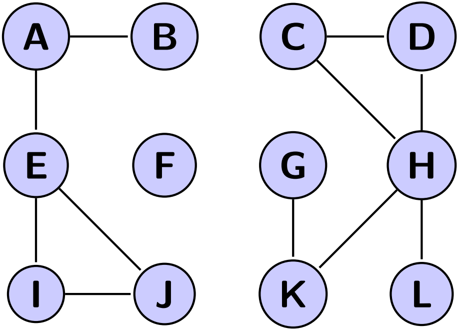
function EXPLORE(w)
  visited(w) $=$ True
  for all $(w, z) \in E$ **do**
    if not visited(z) **then**
      EXPLORE(z)

Finding Connected Components

```
function DFS(G)
    for all w ∈ V do
        visited(w) = False
    CC = 0                          ▷ CC keeps track of the component #
    for all w ∈ V do
        if not visited(w) then
            CC ++
            Explore(w)

function Explore(w)
    visited(w) = True
    ccnum(w) = CC                   ▷ Set the component #
    for all (w, z) ∈ E do
        if not visited(z) then
            Explore(z)
```

Example: Finding Connected Components



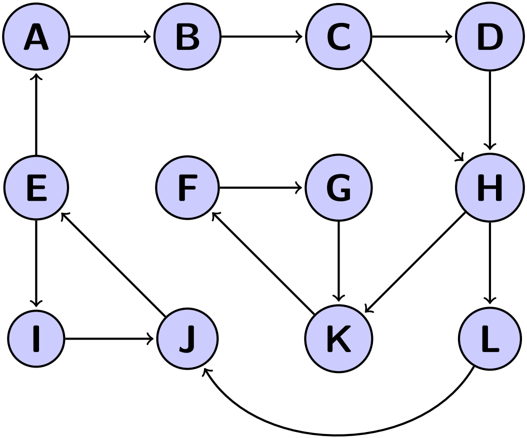

```
function DFS(G)
    for all w ∈ V do
        visited(w) = False
    CC = 0
    for all w ∈ V do
        if not visited(w) then
            CC ++
            EXPLORE(w)

function EXPLORE(w)
    visited(w) = True
    ccnum(w) = CC
    for all (w, z) ∈ E do
        if not visited(z) then
            EXPLORE(z)
```

Directed Graphs

- A directed graph (or digraph) is a set of *vertices* and a collection of *directed edges* that each connect an *ordered* pair of vertices.
- A directed path is a sequence of vertices in which there is a directed edge pointing from each vertex in the sequence to its successor in the sequence.
- A vertex w is reachable from v if there is a *directed path* from v to w.
- For directed graphs, vertices v and w are strongly connected if there is a path from v to w and a path from w to v.
- A graph is strongly connected if every pair of vertices is strongly connected.

Example: Components in Directed Graphs



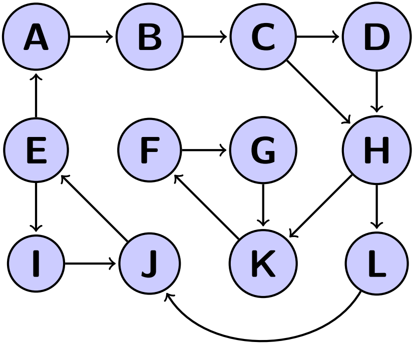
What if you remove the edge (L, J)?

Recording Pre and Post Order Numbers

```
function DFS(G)
    for all w ∈ V do
        visited(w) = False
    clock = 1
    for all w ∈ V do
        if not visited(w) then
            EXPLORE(w)
```

```
function EXPLORE(w)
    visited(w) = True
    pre(w) = clock
    clock ++
    for all (w, z) ∈ E do
        if not visited(z) then
            EXPLORE(z)
    post(w) = clock
    clock ++
```

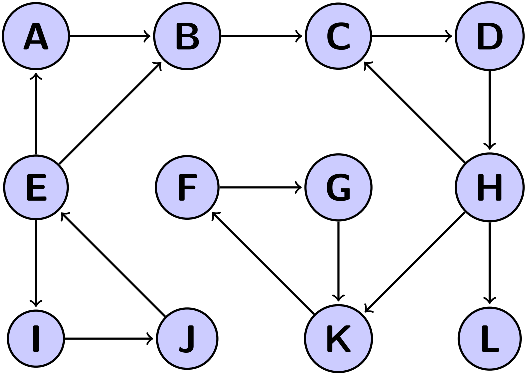
Example: Recording Pre and Post Order Numbers



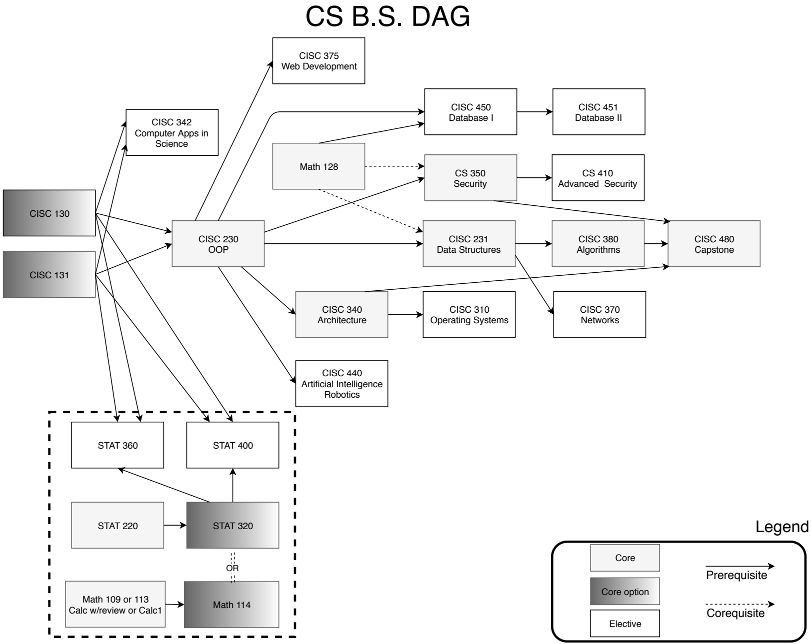
function $\mathrm{DFS}(G)$
  for all $w \in V$ **do**
    visited(w) = False
  clock = 1
  for all $w \in V$ **do**
    if not visited(w) **then**
      $\textsc{Explore}$(w)

function $\textsc{Explore}$(w)
  visited(w) = True
  pre(w) = clock, clock ++
  for all $(w, z) \in E$ **do**
    if not visited(z) **then**
      $\textsc{Explore}$(z)
  post(w) = clock, clock++

Example: Component Graph



Example Directed Acyclic Graph (DAG)



Finding Strongly Connected Components

SCC Algorithm

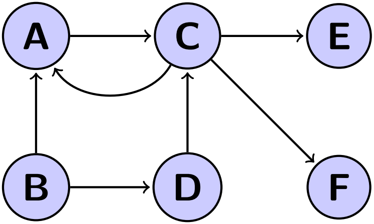
1. Construct the reverse graph G^R
2. Run DFS (and record the post-numbers) on G^R
3. Order the vertices by decreasing post # from step(2)
4. Run the Undirected CC algorithm on the directed graph G

Finding Strongly Connected Components

```
function DFS-CC(G)
    for all w ∈ V do
        visited(w) = False
    CC = 0
    for all w ∈ V (ordered by decreasing post #) do
        if not visited(w) then
            CC++
            Explore(w)

function Explore(w)
    visited(w) = True
    ccnum(w) = CC
    for all (w, z) ∈ E do
        if not visited(z) then
            Explore(z)
```

Example: Components in Directed Graphs



SCC Algorithm

1. Construct G^R
2. Run DFS on G^R
3. Order vertices by decreasing post # from step(2)
4. Run the Undirected CC algorithm on directed G

Why does the SCC algorithm work?

Lemma

If S and S' are SCCs and there is a $v \in S$, $w \in S'$ with an edge $v \to w$ then the max post $\#$ in S is greater than the max post $\#$ in S'.

Proof.

Since there is a path $S \rightsquigarrow S'$ (the edge $v \to w$) there is no $S' \rightsquigarrow S$ path (otherwise S and S' would be in the same SCC).
Let v be the first vertex in $S \cup S'$ that's visited by DFS. Then we have 2 cases:

1. If $v \in S'$, then we visit all of S' before seeing any of S, so we're done.
2. If $v \in S$, then we see all of S and S' before finishing v so v has the max post $\#$ in $S \cup S'$.

$\square$

Why does the SCC algorithm work?

Lemma

The vertex with the highest post # lies in a source SCC.

Proof.

From the previous lemma, we know if S and S' are SCCs and there is a $v \in S$, $w \in S'$ with an edge $v \to w$ then the max post # in S is greater than the max post # in S'.

Hence, we can topologically sort the SCCs by the max post # in each SCC.

So, the SCC with the max post # is a source SCC. $\square$

Single Source (s) shortest path: BFS

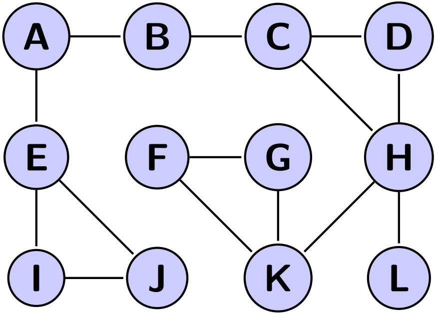
Input: directed or undirected $G = (V, E)$ in adjacency list representation and starting vertex $s \in V$.

Output: For all $w \in V$,

$$\text{dist(w)} = \text{min } \# \text{ of edges to go from } s \text{ to } w.$$

function BFS(G,s)
    for all $w \in V$ **do**
        dist(w) $= \infty$, prev(w) $=$ NULL
    dist(s) $=$ 0, Q $= \{$s$\}$ ▷ Create a queue containing s
    while Q $\neq$ 0 **do**
        w $=$ dequeue(Q)
        for all $(w, z) \in E$ **do**
            if $dist(z) = \infty$ **then**
                enqueue(Q,Z)
                dist(z) $=$ dist(w) $+$ 1
                prev(z) $=$ w

Example: BFS on General Graph



function BFS(G,s)

  for all $w \in V$ **do**

    dist(w) $= \infty$

    prev(w) $=$ NULL

  dist(s) $= 0$, Q $= \{$s$\}$

  while Q $\neq 0$ **do**

    w $=$ dequeue(Q)

    for all $(w, z) \in E$ **do**

      if $dist(z) = \infty$ **then**

        enqueue(Q,Z)

        dist(z) $=$ dist(w)$+1$

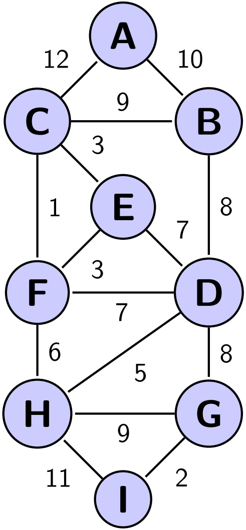
        prev(z) $=$ w

Dijkstra's Algorithm

Input: directed or undirected $G = (V, E)$ in adjacency list representation with weights $length(e)$ for each edge $e \in E$ and starting vertex $s \in V$.

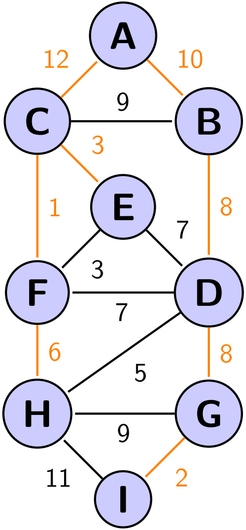
```
function DIJKSTRA(G,length)
    for all u ∈ V do
        dist(u) = ∞
        prev(u) = NULL
    dist(s) = 0                                  ▷ s is the start vertex
    Q = {}          ▷ empty priority queue using dist values as keys
    for all u ∈ V do
        INSERT(Q,u,dist(u))
    while Q ≠ {} do     ▷ repeat until the priority queue is empty
        u = DELETEMIN(Q)
        for all (u, z) ∈ E do
            if dist(z) > dist(u) + length(u,z)  then
                dist(z) = dist(u) + length(u,z)
                prev(z) = u
                DECREASEKEY(Q,z,dist(z))
```

Example: Dijkstra's Algorithm




```
function DIJKSTRA(G,l)
    for all u ∈ V do
        dist(u) = ∞
        prev(u) = NULL
    dist(s) = 0
    Q = {}                                  ▷ priority queue
    for all u ∈ V do
        INSERT(Q,u,dist(u))
    while Q ≠ {} do
        u = DELETEMIN(Q)
        for all (u, z) ∈ E do
            if dist(z) > dist(u) + l(u,z) then
                dist(z) = dist(u) + l(u,z)
                prev(z) = u
                DECREASEKEY(Q,z,dist(z))
```

Example: Dijkstra's Algorithm (Solution)



Node	Dist	Prev
A	0	null
B	10	A
C	12	A
D	18	B
E	15	C
F	13	C
G	26	D
H	19	F
I	28	G

Min-Heap Data Structure H

Contains a set of elements (in this case vertices)
Each has a key (in this case $cost(v)$)

<u>Operations:</u>

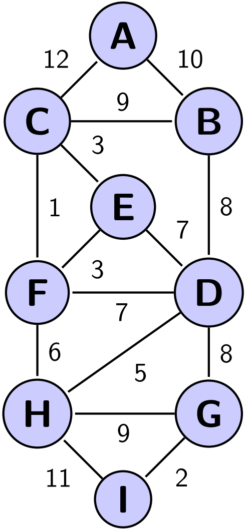
- `Insert(H,v,cost(v))`: Add element v with key $cost(v)$ to H
- `DecreaseKey(H,v,cost(v))`: For $v \in H$, decrease its key to value $cost(v)$
- `DeleteMin(H)`: Output the element in H with the smallest key and delete it from H

For heap with $\leq n$ element this takes $O(\log n)$ time per operation.

Prim's MST Algorithm

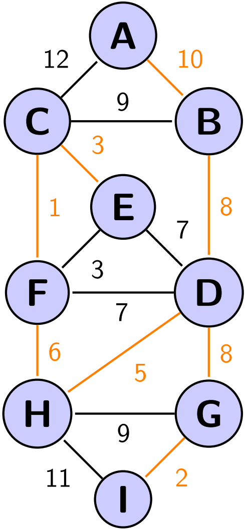
```
function PRIM(G,w)
    for all u ∈ V do
        cost(u) = ∞
        prev(u) = NULL
    cost(s) = 0                  ▷ s is an arbitrarily chosen start vertex
    Q = {}          ▷ empty priority queue using cost values as keys
    for all u ∈ V do
        INSERT(Q,u,cost(u))
    while Q ≠ {} do     ▷ repeat until the priority queue is empty
        u = DELETEMIN(Q)
        for all (u, z) ∈ E do
            if cost(z) > w(u, z) then
                cost(z) = w(u,z)
                prev(z) = u
                DECREASEKEY(Q,z,cost(z))
```

Example: Prim's MST Algorithm



function $\textsc{Prim}$(G,w)
    for all $u \in V$ **do**
        cost(u) $= \infty$
        prev(u) = NULL
    cost(s) = 0
    Q = {}   ▷ priority queue
    for all $u \in V$ **do**
        $\textsc{Insert}$(Q,u,cost(u))
    while Q $\neq$ {} **do**
        u = $\textsc{DeleteMin}$(Q)
        for all $(u, z) \in E$ **do**
            if $cost(z) > w(u, z)$ **then**
                cost(z) = w(u,z)
                prev(z) = u
                $\textsc{DecreaseKey}$(Q,z,cost(z))

Example: Prim's MST Algorithm



function PRIM(G,w)
  for all $u \in V$ **do**
    cost(u) $= \infty$
    prev(u) $=$ NULL
  cost(s) $= 0$
  Q $= \{\}$ ▷ priority queue
  for all $u \in V$ **do**
    INSERT(Q,u,cost(u))
  while Q $\neq \{\}$ **do**
    u $=$ DELETEMIN(Q)
    for all $(u, z) \in E$ **do**
      if $cost(z) > w(u, z)$ **then**
        cost(z) $=$ w(u,z)
        prev(z) $=$ u
        DECREASEKEY(Q,z,cost(z))

Cut Property

For $G = (V, E)$, consider $X \subset E$ where $X \subset T$ for a MST T.

Take any subset S of vertices where no edge of X crosses between S and $\bar{S} = V - S$.

Let e^* be the min weight edge of E between S and $\bar{S}$.

Then, $X \cup e^* \subseteq T'$ for some MST T'.

Union-Find Data Structure

-A collection of sets.
-Each set has a unique name. Operations:

- `Makeset(x)`: Create a new set containing just x
- `Find(x)`: Return the name of the set containing x
- `Union(x,y)`: Merge the sets containing x and y

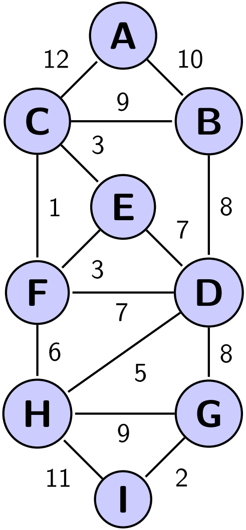
Assuming n elements, $O(1)$ time per `Makeset` and $O(\log n)$ for each `Find`,`Union`.

Kruskal's MST Algorithm

Input: undirected, connected $G = (V, E)$ with edge weights $w(e) > 0$ for all $e \in E$
Output: MST defined by $X \subset E$

function KRUSKAL(G,w)
    for all $u \in V$ **do**
        MAKESET(u)
    $X = \emptyset$
    Sort E by increasing weight
    for all $e = (y, z) \in E$ **do** ▷ By increasing weight!
        if FIND(y)$\neq$ FIND(z) **then**
            add e to X
            UNION(y,z)
    return X

Example: Kruskal's MST Algorithm



function KRUSKAL(G,w)

  for all $u \in V$ **do**

    MAKESET(u)

  $X = \emptyset$

  Sort E by increasing weight

  for all $e = (y, z) \in E$ **do** ▷ By increasing weight!

    if FIND(y)$\neq$ FIND(z) **then**

      add e to X

      UNION(y,z)

  return X

Union-Find Data Structure

function MAKESET(x)
    $\pi(x) = x$
    $rank(x) = 0$

function FIND(x)
    while $x \neq \pi(x)$ **do**
        $x = \pi(x)$

function UNION(x,y)
    $r_x = \text{FIND}(x)$
    $r_y = \text{FIND}(y)$
    if $rank(r_x) > rank(r_y)$ **then**
        $\pi(r_y) = r_x$
    if $rank(r_y) > rank(r_x)$ **then**
        $\pi(r_x) = r_y$
    if $rank(r_x) == rank(r_y)$ **then**
        $\pi(r_x) = r_y$
        $rank(r_y) + +$