

Pre-Announcements

Blockchain event is today 7 - 10 PM at International House at Chevron House.

- Lots of fancy people from fancy places will be there.
- Blockchain is a topical concept.
- Pizza is a topical concept.

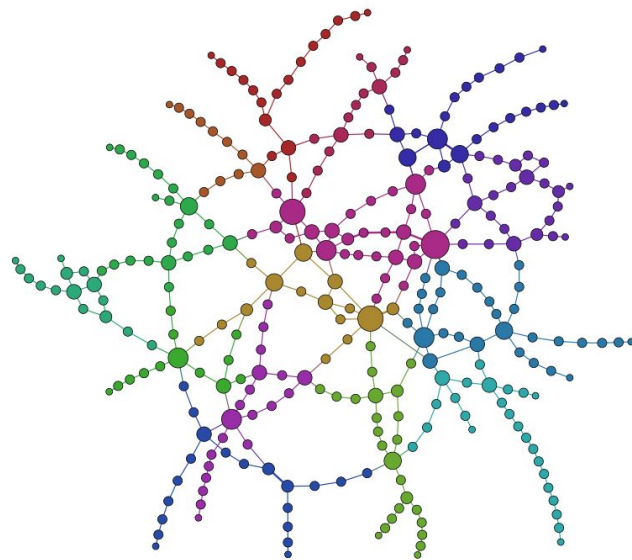
Flyers available.

Announcements

Exam solution exists, not sure why it's not posted, but will be posted soon.

- Exam was really hard, but that's how exams go.
- More later.

Examples



CS61B

Lecture 26: Graphs

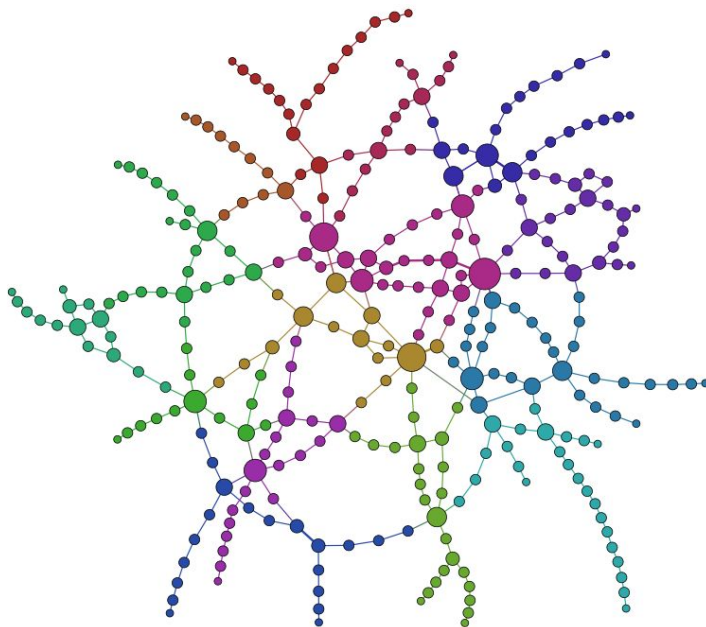
- Intro
- Graph Implementations
- Depth First Traversal

Graph

Graph: A set of nodes (a.k.a. vertices) connected pairwise by edges.

Introduction to **Network Visualization** with GEPHI – Martin Grandjean

Examples

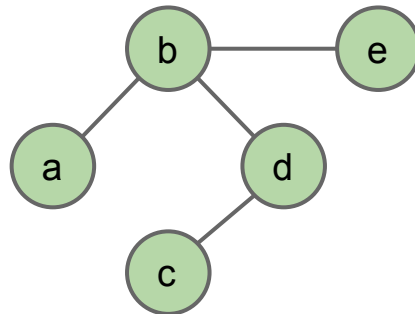
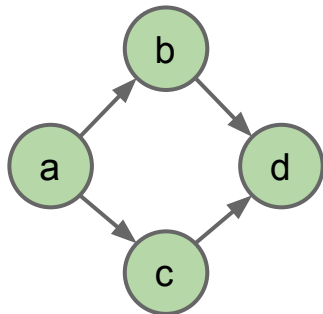


Graph Types

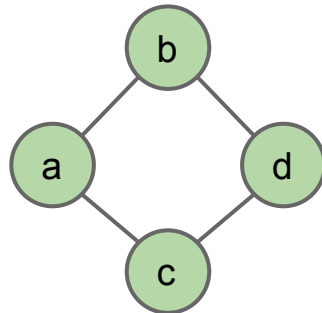
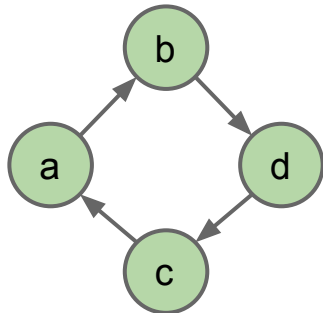
Directed

Undirected

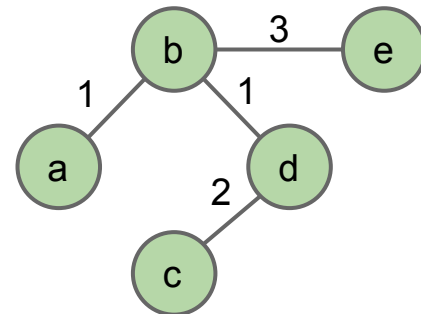
Acyclic:



Cyclic:



With Edge Labels



Graph Terminology

- Graph:
 - Set of **vertices**, a.k.a. **nodes**.
 - Set of **edges**: Pairs of vertices.
 - Vertices with an edge between are **adjacent**.
 - Optional: Vertices or edges may have **labels** (or **weights**).
- A **path** is a sequence of vertices connected by edges.
- A **cycle** is a path whose first and last vertices are the same.
 - A graph with a cycle is 'cyclic'.
- Two vertices are **connected** if there is a path between them. If all vertices are connected, we say the graph is connected.

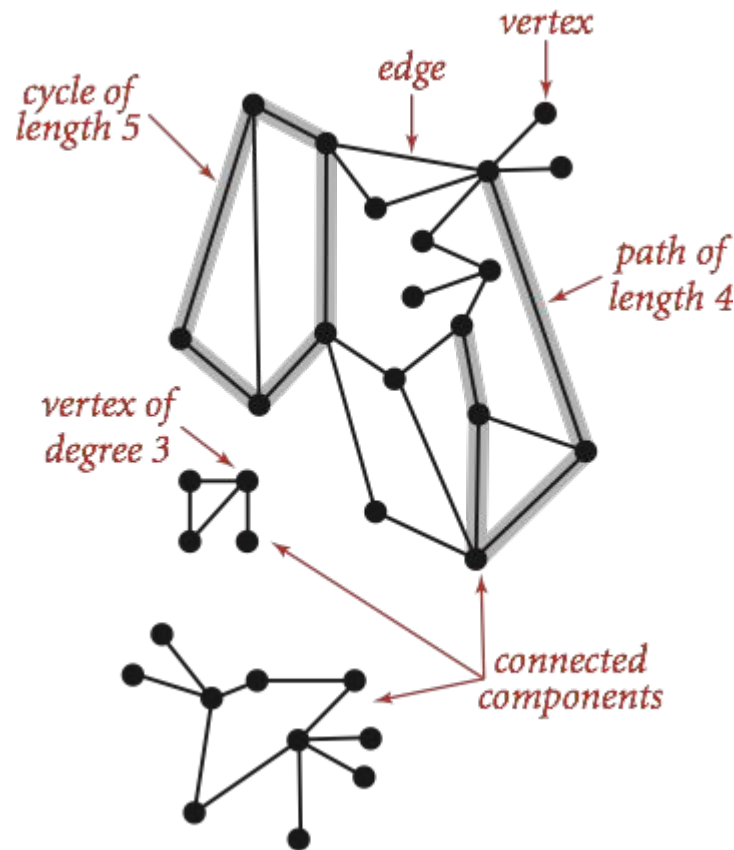


Figure from Algorithms 4th Edition

Some Graph-Processing Problems

s-t Path. Is there a path between vertices s and t ?

Shortest s-t Path. What is the shortest path between vertices s and t ?

Cycle. Does the graph contain any cycles?

Euler Tour. Is there a cycle that uses every edge exactly once?

Hamilton Tour. Is there a cycle that uses every vertex exactly once?

Connectivity. Is the graph connected, i.e. is there a path between all vertex pairs?

Biconnectivity. Is there a vertex whose removal disconnects the graph?

Planarity. Can you draw the graph on a piece of paper with no crossing edges?

Isomorphism. Are two graphs isomorphic (the same graph in disguise)?

Graph problems: Unobvious which are easy, hard, or computationally intractable.

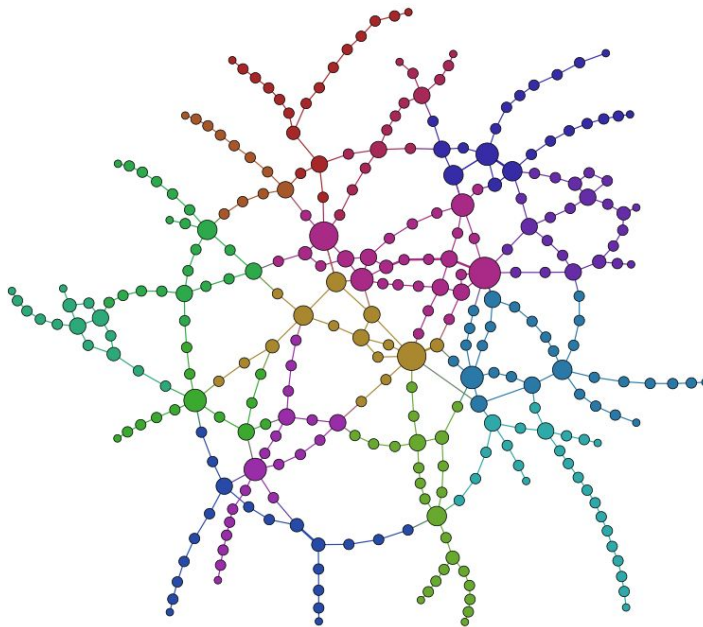
Graph Example: The Paris Metro

This subway map of Paris is:

- Undirected
- Connected
- Cyclic (not a tree!)
- Vertex-labeled

Introduction to **Network Visualization** with GEPHI – Martin Grandjean

Examples



Graph Example: BART

Is the BART graph a tree?





facebook

December 2010

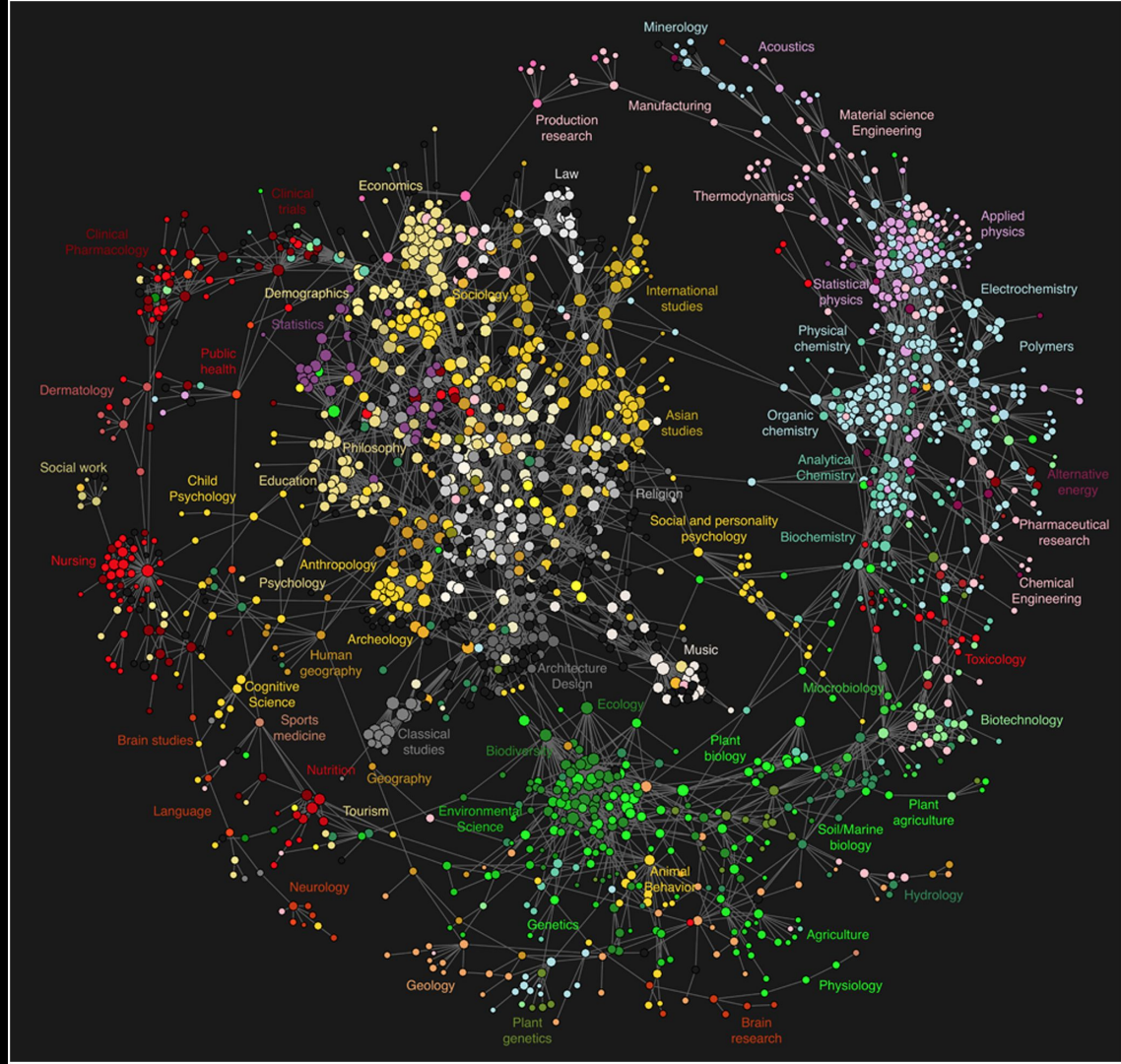
Nodes: Cities. Edge Weights: ~Number of friends between cities

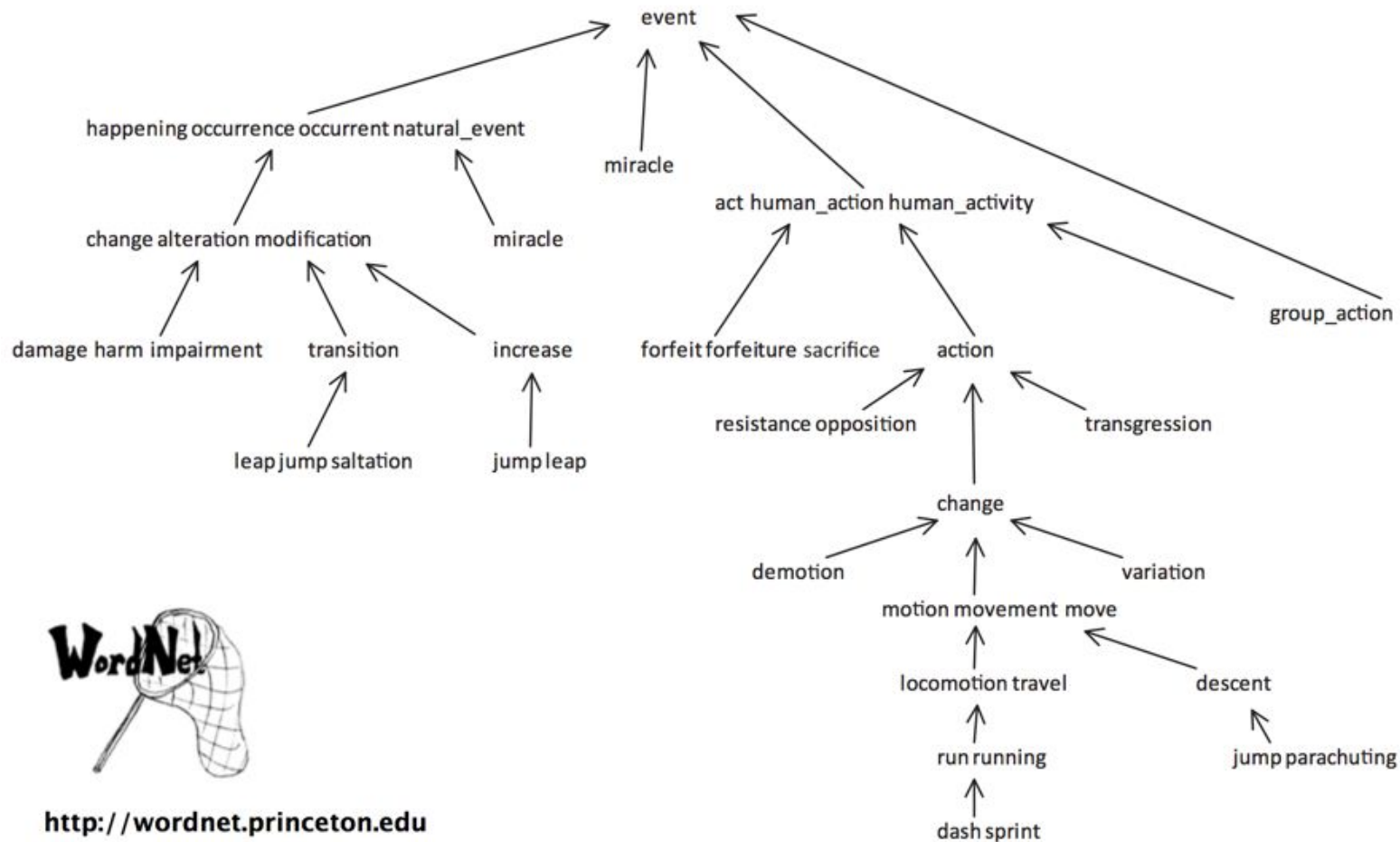
Nodes: Scientific Journals.

- Label: AAT classification (the topic that it covers)

Edges:

- Based on clickthrough data.
- Clickthrough from v to w means that someone reading an article in journal v clicked on a link to an article in journal w .
- Edge assigned from v to w if clickthrough rate from v to w is above some arbitrary threshold.



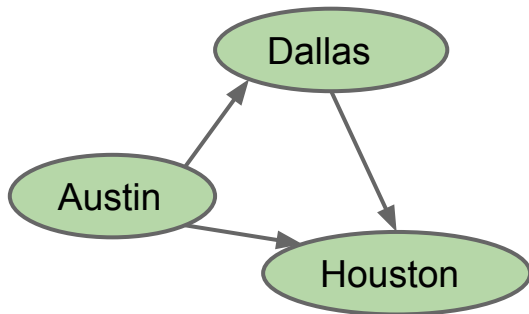


Edge captures 'is-a-type-of' relationship. Example: descent is-a-type-of movement.

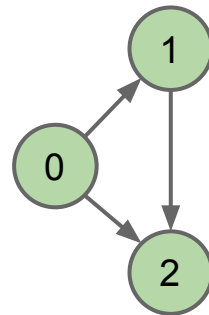
Graph Representations

Common Simplification: Integer Vertices

Common convention: Number nodes irrespective of label, and use number throughout the graph implementation. To lookup a vertex by label, use a `Map<Label, Integer>`.



Intended graph.



```
Map<String, Integer>  
Austin: 0  
Dallas: 1  
Houston: 2
```

What you get.

Graph API

Using a graph in Java:

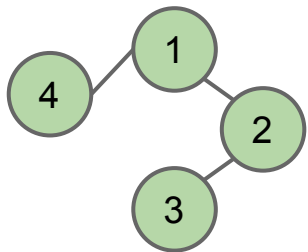
```
public class Graph {
    public Graph(int V):           Create empty graph with v vertices
    public void addEdge(int v, int w): add an edge v-w
    Iterable<Integer> adj(int v):  vertices adjacent to v
    int V():                       number of vertices
    int E():                       number of edges
    ...
}
```

Graph API

Using a graph in Java:

```
public class Graph {  
    public Graph(int V):           Create empty graph with v vertices  
    public void addEdge(int v, int w): add an edge v-w  
    Iterable<Integer> adj(int v):  vertices adjacent to v  
    int V():                       number of vertices  
    int E():                       number of edges  
    ...  
}
```

Example client:



degree(G, 2) = 2

```
/** degree of vertex v in graph G */  
public static int degree(Graph G, int v) {  
    int degree = 0;  
    for (int w : G.adj(v)) {  
        degree += 1;  
    }  
    return degree; }  

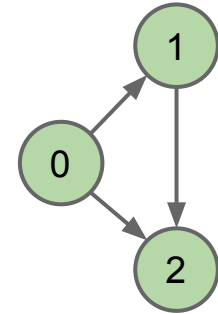
```

(degree = # edges)

Graph Representations

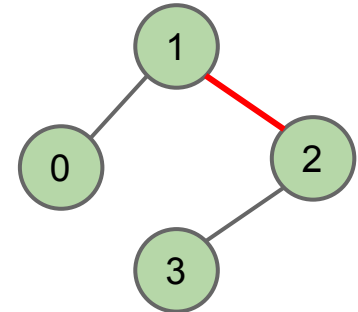
- Representation 1: Adjacency Matrix.

s \ t	0	1	2
0	0	1	1
1	0	0	1
2	0	0	0



For undirected graph:
Each edge is
represented twice in the
matrix. Simplicity at the
expense of space.

v \ w	0	1	2	3
0	0	1	0	0
1	1	0	1	0
2	0	1	0	1
3	0	0	1	0



Graph Printing Runtime: <http://yellkey.com/paper>

What is the order of growth of the running time of the following code if the graph uses an adjacency-matrix representation, where V is the number of vertices, and E is the total number of edges?

- A. $\Theta(V)$
- B. $\Theta(V + E)$
- C. $\Theta(V^2)$
- D. $\Theta(V \cdot E)$

```
for (int v = 0; v < G.V(); v++) {  
    for (int w : G.adj(v)) {  
        System.out.println(v + "-" + w);  
    }  
}
```

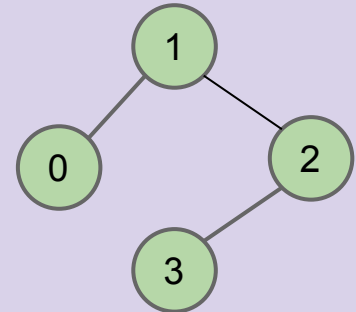
What is the runtime of the for-each?

-

How many times is the for-each run?

-

	0	1	2	3
0	0	1	0	0
1	1	0	1	0
2	0	1	0	1
3	0	0	1	0



Graph Printing Runtime: <http://yellkey.com/paper>

What is the order of growth of the running time of the following code if the graph uses an adjacency-matrix representation, where V is the number of vertices, and E is the total number of edges?

- A. $\Theta(V)$
- B. $\Theta(V + E)$
- C. $\Theta(V^2)$
- D. $\Theta(V \cdot E)$

```
for (int v = 0; v < G.V(); v++) {  
    for (int w : G.adj(v)) {  
        System.out.println(v + "-" + w);  
    }  
}
```

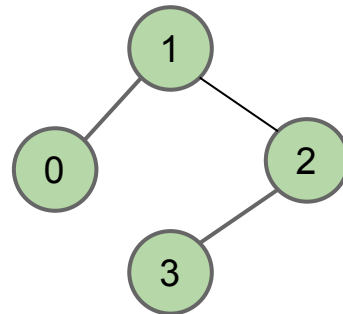
What is the runtime of the for-each?

- $\Theta(V)$.

How many times is the for-each run?

- V times.

	0	1	2	3
0	0	1	0	0
1	1	0	1	0
2	0	1	0	1
3	0	0	1	0



Graph Printing Runtime: <http://yellkey.com/paper>

What is the order of growth of the running time of the following code if the graph uses an adjacency-matrix representation, where V is the number of vertices, and E is the total number of edges?

- A. $\Theta(V)$
- B. $\Theta(V + E)$
- C. $\Theta(V^2)$
- D. $\Theta(V \cdot E)$

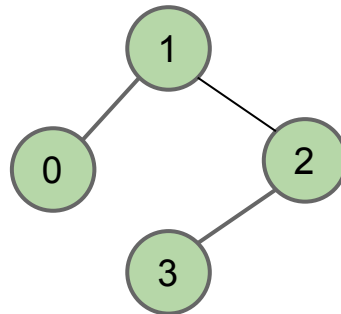
```
for (int v = 0; v < G.V(); v++) {  
    for (int w : G.adj(v)) {  
        System.out.println(v + "-" + w);  
    }  
}
```

E is the number of 1s

What does $G.\text{adj}(1)$ return?

- An iterator with $\text{next}() = 0$, then $\text{next}() = 2$.

	0	1	2	3
0	0	1	0	0
1	1	0	1	0
2	0	1	0	1
3	0	0	1	0



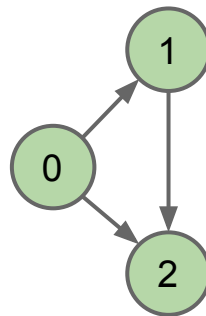
More Graph Representations

Representation 2: Edge Sets: Collection of all edges.

- Example: `HashSet<Edge>`, where each Edge is a pair of ints.

$\{(\emptyset, 1), (\emptyset, 2), (1, 2)\}$

a set

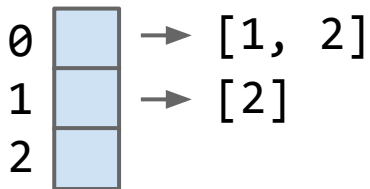


More Graph Representations

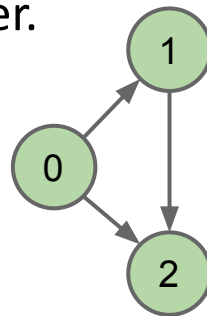
Representation 3: Adjacency lists.

most common

- Common approach: Maintain array of lists indexed by vertex number.
- Most popular approach for representing graphs.



every vertex: a list of number



Graph Printing Runtime: <http://shoutkey.com/laugh>

What is the order of growth of the running time of the following code if the graph uses an **adjacency-list** representation, where V is the number of vertices, and E is the total number of edges?

- A. $\Theta(V)$
- B. $\Theta(V + E)$
- C. $\Theta(V^2)$
- D. $\Theta(V * E)$

```
for (int v = 0; v < G.V(); v++) {  
    for (int w : G.adj(v)) {  
        System.out.println(v + "-" + w);  
    }  
}
```

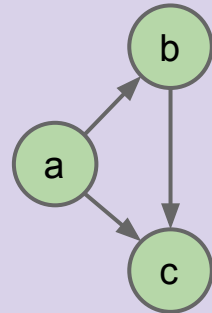
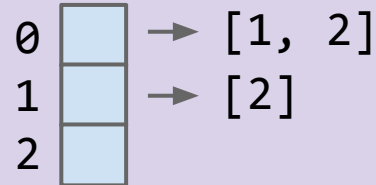
What is the runtime of the for-each?

•

between V and V^2

How many times is the for-each run?

•



Graph Printing Runtime: <http://shoutkey.com/laugh>

What is the order of growth of the running time of the following code if the graph uses an **adjacency-list** representation, where V is the number of vertices, and E is the total number of edges? **Best case: $\Theta(V)$ Worst case: $\Theta(V^2)$**

- A. $\Theta(V)$
- B. $\Theta(V + E)$**
- C. $\Theta(V^2)$
- D. $\Theta(V * E)$

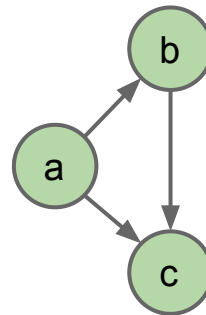
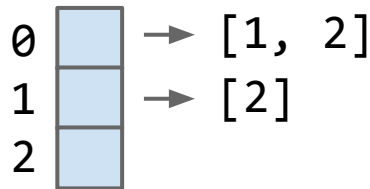
```
for (int v = 0; v < G.V(); v++) {  
    for (int w : G.adj(v)) {  
        System.out.println(v + "-" + w);  
    }  
}
```

What is the runtime of the for-each? List can be between 1 and V items.

- $\Omega(1), O(V)$.

How many times is the for-each run?

- V .



Graph Printing Runtime: <http://shoutkey.com/ready>

What is the order of growth of the running time of the following code if the graph uses an **adjacency-list** representation, where V is the number of vertices, and E is the total number of edges? **Best case: $\Theta(V)$ Worst case: $\Theta(V^2)$**

A. $\Theta(V)$

B. $\Theta(V + E)$

C. $\Theta(V^2)$

D. $\Theta(V \cdot E)$

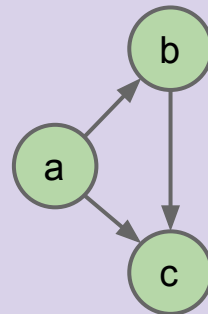
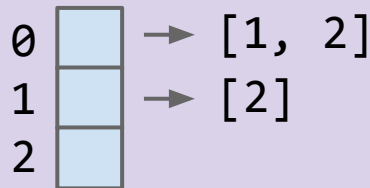
```
for (int v = 0; v < G.V(); v++) {  
    for (int w : G.adj(v)) {  
        System.out.println(v + "-" + w);  
    }  
}
```

What is the runtime of the for-each? List can be between 1 and V items.

- $\Omega(1), O(V)$.

How many times is the for-each run?

- V .



Graph Printing Runtime: <http://shoutkey.com/ready>

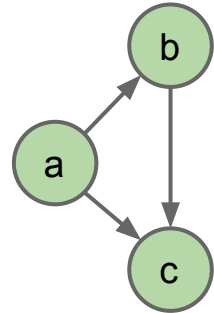
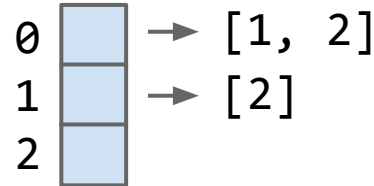
What is the order of growth of the running time of the following code if the graph uses an **adjacency-list** representation, where V is the number of vertices, and E is the total number of edges?

- A. $\Theta(V)$
- B. $\Theta(V + E)$
- C. $\Theta(V^2)$
- D. $\Theta(V \cdot E)$

```
for (int v = 0; v < G.V(); v++) {  
    for (int w : G.adj(v)) {  
        System.out.println(v + "-" + w);  
    }  
}
```

Best case: $\Theta(V)$ Worst case: $\Theta(V^2)$

All cases: $\Theta(V + E)$



Graph Printing Runtime: <http://shoutkey.com/ready>

Runtime: $\Theta(V + E)$

V is total number of vertices.

E is total number of edges in the entire graph.

```
for (int v = 0; v < G.V(); v++) {  
    for (int w : G.adj(v)) {  
        System.out.println(v + "-" + w);  
    }  
}
```

How to interpret: No matter what “shape” of increasingly complex graphs we generate, as V and E grow, the runtime will always grow exactly as $\Theta(V + E)$.

- Example shape 1: Very sparse graph where E grows very slowly, e.g. every vertex is connected to its square: 2 - 4, 3 - 9, 4 - 16, 5 - 25, etc.

- E is $\Theta(\sqrt{V})$. Runtime is $\Theta(V + \sqrt{V})$, which is just $\Theta(V)$ see how many square roots we can possibly have and trace back

- Example shape 2: Very dense graph where E grows very quickly, e.g. every vertex connected to every other.

- E is $\Theta(V^2)$. Runtime is $\Theta(V + V^2)$, which is just $\Theta(V^2)$.

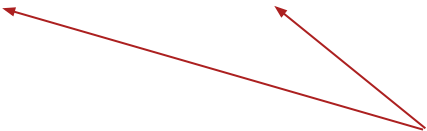
Graph Representations

Runtime of some basic operations for each representation:

idea	addEdge(s, t)	for(w : adj(v))	printgraph()	hasEdge(s, t)	space used
adjacency matrix	$\Theta(1)$	$\Theta(V)$	$\Theta(V^2)$	$\Theta(1)$	$\Theta(V^2)$
list of edges	$\Theta(1)$	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$	$\Theta(E)$
adjacency list	$\Theta(1)$	$\Theta(1)$ to $\Theta(V)$	$\Theta(V+E)$	$\Theta(\text{degree}(v))$	$\Theta(E+V)$

In practice, adjacency lists are most common.

- Many graph algorithms rely heavily on $\text{adj}(s)$.
- Most graphs are sparse (not many edges in each bucket).



Note: These operations are not part of the Graph class's API.

Bare-Bones Undirected Graph Implementation

```
public class Graph {
    private final int V;    private List<Integer>[] adj;

    public Graph(int V) {
        this.V = V;
        adj = (List<Integer>[]) new ArrayList[V];
        for (int v = 0; v < V; v++) {
            adj[v] = new ArrayList<Integer>();
        }
    }

    public void addEdge(int v, int w) {
        adj[v].add(w);    adj[w].add(v);
    }

    public Iterable<Integer> adj(int v) {
        return adj[v];
    }
}
```

Cannot create array of anything involving generics, so have to use weird cast as with project 1A.

Depth-First Traversal

Maze Traversal / s-t Path

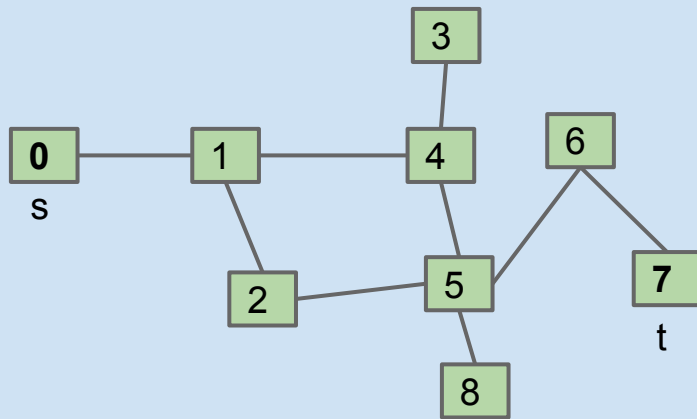
Suppose we want to know if there exists a path from vertex $s=0$ to vertex $t=7$. What is **wrong** with the following recursive algorithm for `connected(s, t)`?

- Does $s == t$? If so, return true.
- Otherwise, check all of s 's children for connectivity to t .

recursive connectivity

Example:

- `connected(0, 7)`:
 - Does $0 == 7$? No, so...
 - if (`connected(1, 7)`) return true;
 - return false;
- `connected(1, 7)`: ...

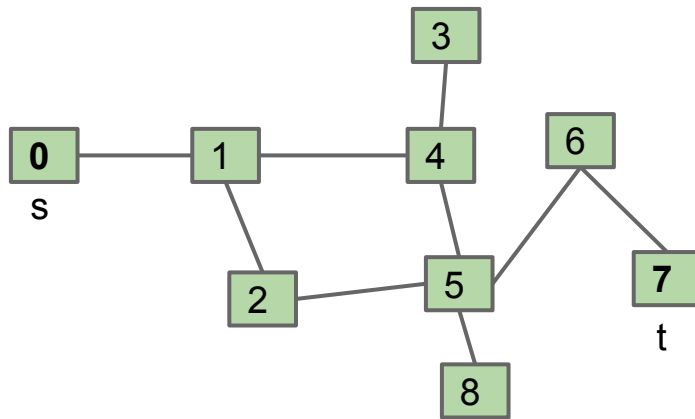


Improving Our Connectivity Algorithm

Goal: Search for a path from s to t , but visit each vertex at most once. To do this, we can mark each vertex as we search. Resulting algorithm for `connected(s , t)` is as follows:

- Mark s .
- Does $s == t$? If so, return true.
- Check all of s 's **unmarked neighbors** for connectivity to t .

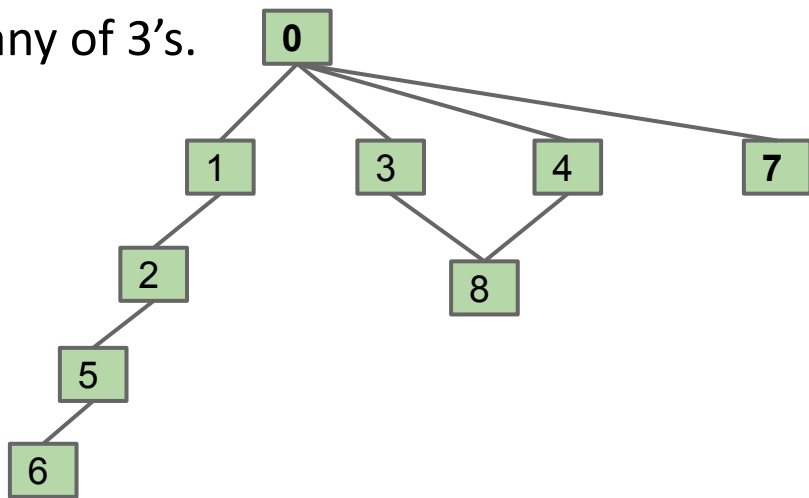
[Recursive connectivity demo.](#)

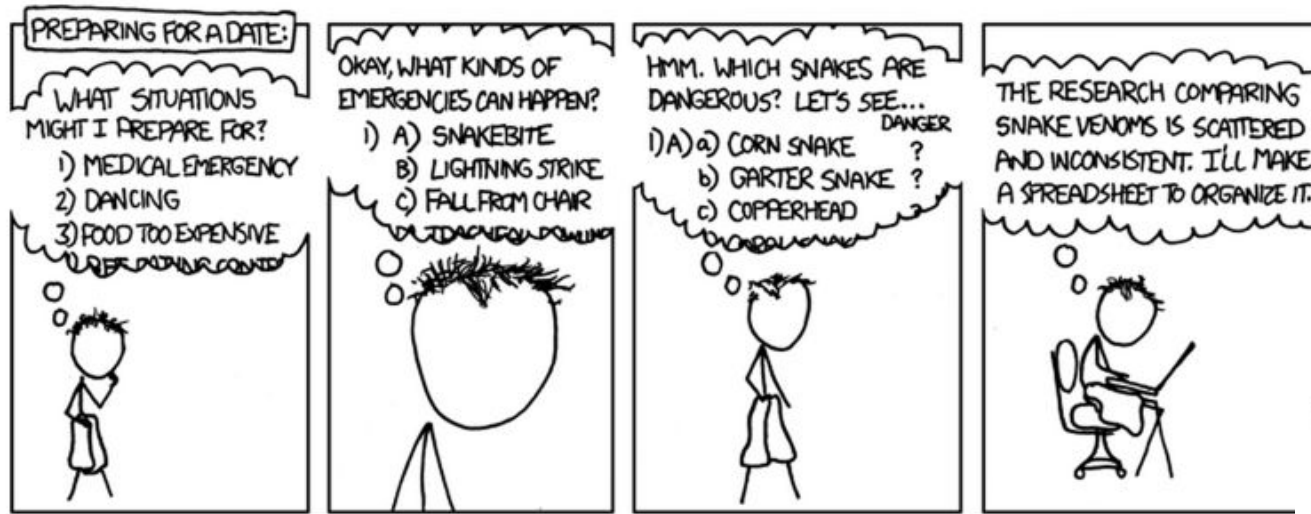


Depth First Traversal

This idea of exploring the entire subgraph for each child is known as Depth First Traversal.

- Ex. Visit all of 1's children before we visit any of 3's.





Or a more visceral example: <https://xkcd.com/761/>

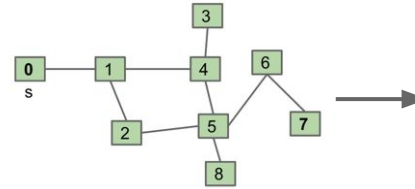


I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

Depth First Search Implementation

Common design pattern in graph algorithms: Decouple type from processing algorithm.

- Create a graph object.
- Pass the graph to a graph-processing method (or constructor) in a client class.
- Query the client class for information.



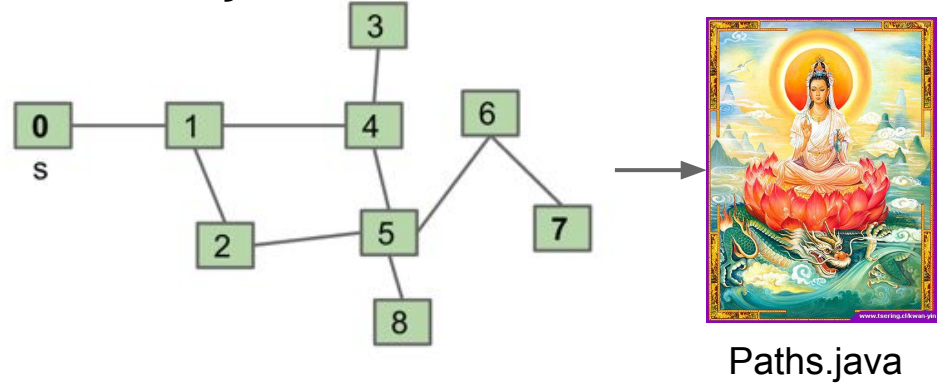
```
public class Paths {  
    public Paths(Graph G, int s):    Find all paths from G  
    boolean hasPathTo(int v):        is there a path from s to v?  
    Iterable<Integer> pathTo(int v): path from s to v (if any)  
}
```

Paths.java

Example Usage

Start by calling: `Paths P = new Paths(G, 0);`

- `P.hasPathTo(3);` //returns true
- `P.pathTo(3);` //returns `{0, 1, 4, 3}`



```
public class Paths {  
    public Paths(Graph G, int s):    Find all paths from G  
    boolean hasPathTo(int v):        is there a path from s to v?  
    Iterable<Integer> pathTo(int v): path from s to v (if any)  
}
```

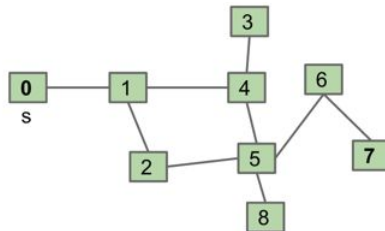
Implementing Paths With Depth First Search

To visit a vertex v :

- Mark vertex v .
- Recursively visit all unmarked vertices adjacent to v .

Data Structures:

- `boolean[] marked`
- `int[] edgeTo`
 - `edgeTo[4] = 1`, means we went from 1 to 4.



Paths.java

```
public class Paths {  
    public Paths(Graph G, int s):    Find all paths from G  
    boolean hasPathTo(int v):        is there a path from s to v?  
    Iterable<Integer> pathTo(int v): path from s to v (if any)  
}
```

DepthFirstPaths

Demo: [DepthFirstPaths](#)

DepthFirstPaths, Recursive Implementation



```
public class DepthFirstPaths {  
    private boolean[] marked;  
    private int[] edgeTo;  
    private int s;  
  
    public DepthFirstPaths(Graph G, int s) {  
        ...  
        dfs(G, s);  
    }  
  
    private void dfs(Graph G, int v) {  
        marked[v] = true;  
        for (int w : G.adj(v)) {  
            if (!marked[w]) {  
                edgeTo[w] = v;  
                dfs(G, w);  
            }  
        }  
    }  
}
```

marked[v] is true iff v connected to s
edgeTo[v] is previous vertex on path from s to v

not shown: data structure initialization
find vertices connected to s.

recursive routine does the work and stores results
in an easy to query manner!

Question: How would we write hasPathTo(v)?

DepthFirstPaths Summary

Demo: [DepthFirstPaths](#)

Properties of Depth First Search:

- Guaranteed to reach every node.
- Runs in $O(V + E)$ time.
 - Analysis next time, but basic idea is that every edge is used at most once, and total number of vertex considerations is equal to number of edges.
 - Runtime may be faster than $\Theta(V+E)$ for problems which quit early on some stopping condition (for example connectivity).